An integer linear program for substitution-tolerant subgraph isomorphism and its use for symbol spotting in technical drawings

Pierre Le Bodic, Pierre Héroux, Sébastien Adam, Yves Lecourtier

An Integer Linear Program for Substitution-Tolerant Subgraph Isomorphism and its Use for Symbol Spotting in Technical Drawings

Pierre Le Bodic[a], Pierre Héroux[b], Sébastien Adam[b], Yves Lecourtier[b]

[a] *Université Paris-Sud – LRI UMR 8623, 91405 Orsay cedex, France*
[b] *Université de Rouen – LITIS EA 4108, 76800 Saint-Etienne du Rouvray cedex, France*

Abstract

This paper tackles the problem of substitution-tolerant subgraph isomorphism which is a specific class of error-tolerant isomorphism. This problem aims at finding a subgraph isomorphism of a pattern graph $\mathcal{S}$ in a target graph $\mathcal{G}$. This isomorphism only considers label substitutions and forbids vertex and edge insertion in $\mathcal{G}$. This kind of subgraph isomorphism is often needed in pattern recognition problems when graphs are attributed with real values and no exact matching can be found between attributes due to noise.

Our proposal to solve the problem of substitution-tolerant subgraph isomorphism relies on its formulation in the Integer Linear Program (ILP) formalism. Using a general ILP solver, the approach is able to find, if one exists, a mapping of a pattern graph into a target graph such that the topology of the searched graph is kept and the editing operations between the labels have a minimal cost.

This technique is evaluated on both a set of synthetic graphs and a problem of symbol detection in technical drawings. In the second case, document and symbol images are represented by vector-attributed Region Adjacency Graphs built from a segmentation process. Obtained results demonstrate the relevance of considering subgraph isomorphism as an optimization process.

Keywords: Substitution-Tolerant Subgraph Isomorphism, Integer Linear Programming, Symbol Spotting

Email addresses: Pierre.Lebodic@lri.fr (Pierre Le Bodic), Pierre.Heroux@univ-rouen.fr (Pierre Héroux), Sebastien.Adam@univ-rouen.fr (Sébastien Adam), Yves.Lecourtier@univ-rouen.fr (Yves Lecourtier)

1. Introduction

Labeled graphs are data structures able to model complex entities. In a graph-based representation, vertices and their labels describe objects or entity components whereas edges represent the relationships between the components. Due to the intrinsic genericity of structural representations and thanks to the improvement of the computational power of computers, graphs are being increasingly used in a large scope of application domains such as biology, chemistry, computer vision, information retrieval or pattern recognition. As a consequence of the intensive use of graph-based representations, a growing interest is being shown to graph algorithms including graph mining [1, 2, 3], graph classification [4, 5] or clustering [6, 7, 8], graph isomorphism [9, 10, 11, 12].

In the context of pattern recognition, graph-based techniques are generally used to match a graph representing a known entity to a graph resulting from an observation. The general framework of graph matching covers different problems according to the kind of constraints which must be respected or those which can be relaxed. A particular class of matching problem is the substitution-tolerant subgraph isomorphism. A subgraph isomorphism is said to be substitution-tolerant when the mapping does not affect the topology, *i.e.* each vertex and each edge of the pattern graph has a one-to-one mapping into the target graph, but when editing operations between vertex and edge labels are allowed. A substitution-tolerant mapping is generally needed when no exact mapping between vertex and/or edge labels can be found, but when the mapping can be associated to a cost. For example, this case occurs when vertex and edge labels are numerical values (scalar or vectorial) resulting from a feature extraction step as often in pattern analysis. The cost for the mapping can then be defined as the sum of the distances between label values. A first solution to tackle such problems relies on a discretization or a classification procedure to transform the numerical values into nominal labels. The main drawback of such approaches is their sensitivity to frontier effects of the discretization or misclassification. A subsequent exact matching algorithm would then be unsuccessful. A second solution consists in using exact matching algorithms and to customize the compatibility function for pairing vertices and edges. The main drawback of such approaches is the need to define thresholds for these compatibilities. A last way consists in

using a substitution-tolerant matching procedure that overcomes this drawback by integrating the numerical values during the mapping search. In this case, the matching problem turns from a decision one to an optimization one.

In this paper, we propose a new modelling of the substitution-tolerant subgraph isomorphism as an optimization problem modeled by an integer linear program. This original approach is parameter free and solves problems that could not be solved optimally using existing algorithms. This paper is an extended and improved version of [13]. The new contributions concern an extension of the approach for the search for multiple solutions and a strategy that enables the learning of a threshold for this search. Moreover, many new experimental results are proposed on both synthetic and real datasets. They include a comparison with state of the art approaches, in order to show the strong points and the weaknesses of the proposed method.

The paper is organized as follows. Section 2 introduces substitution-tolerant subgraph isomorphism and reviews related approaches from the literature. Section 3 presents Integer Linear Programming as a way to express optimization problems and our formulation of the substitution-tolerant subgraph isomorphism. Section 4 reports many experimental results obtained by the proposed approach on both synthetic and application-dependent datasets. The application concerns the localization of symbols on technical drawings. Finally, section 5 draws a conclusion of the paper and proposes future directions.

2. Subgraph isomorphism

2.1. Definitions and notations

The approach proposed in this paper deals with directed graphs, defined as follows :

Definition 1. A directed attributed multigraph[1] $\mathcal{G}$ is a *4-tuple* $\mathcal{G} = (V, E, \mu, \xi)$ where V is the set of vertices of $\mathcal{G}$, E is a multiset of ordered pairs $e = (v_1, v_2)$ with $v_1 \in V$ and $v_2 \in V$, i.e. edges of $\mathcal{G}$. $\mu : V \rightarrow L_V$ is a function assigning a *label* to a vertex, L_V being the set of possible labels for vertices. $\xi : E \rightarrow L_E$ is a function assigning a *label* to an edge, L_E being the set of possible labels for edges.

[1]In the remaining of the paper, the term graph denotes a directed attributed multigraph.

Considering two graphs, one can be interested in finding the instances of the first one (the pattern graph) in the second one (the target graph). In graph theory, this problem is called subgraph isomorphism [2]. It relies on the definition of subgraph :

Definition 2. Given a graph $\mathcal{G} = (V, E, \mu, \xi)$, a subgraph of $\mathcal{G}$ is a graph $\mathcal{S} = (V_\mathcal{S}, E_\mathcal{S}, \mu_\mathcal{S}, \xi_\mathcal{S})$ such that $V_\mathcal{S} \subseteq V$, $E_\mathcal{S} \subseteq E$, $\forall e = (v_1, v_2) \in E_\mathcal{S}, v_1 \in V_\mathcal{S}, v_2 \in V_\mathcal{S}$ and $\mu_\mathcal{S}$ and $\xi_\mathcal{S}$ are the restrictions of μ and ξ to $V_\mathcal{S}$ and $E_\mathcal{S}$, i.e. $\mu_\mathcal{S}(v) = \mu(v)$ and $\xi_\mathcal{S}(e) = \xi(e)$

Using this definition, the subgraph isomorphism problem between a pattern graph $\mathcal{G}$ and a target graph $\mathcal{G}'$ is defined by :

Definition 3. An injective function $f : V \to V'$ is a subgraph isomorphism from a graph $\mathcal{G} = (V, E, \mu, \xi)$ to a graph $\mathcal{G}' = (V', E', \mu', \xi')$ if there exists a subgraph $\mathcal{S}$ of $\mathcal{G}'$ such that f is a graph isomorphism from $\mathcal{G}$ to $\mathcal{S}$:

- $\forall v \in V, f(v) = v' \in V', f^{-1}(v') = v$
- for all $e = (v_1, v_2) \in E$, there exists a distinct edge $e' = (f(v_1), f(v_2)) \in E'$

Note that extra edges may exist in G' between mapped vertices, *i*.e. a subgraph does not need to be induced.

In its exact formulation, the subgraph isomorphism must preserve the labelling, *i*.e. $\mu(v) = \mu'(v')$ and $\xi(e) = \xi'(e')$. In pattern recognition applications, where vertices and edges are labeled with measures which may be affected by noise, a substitution-tolerant formulation which allows differences between labels of mapped vertices and edges is mandatory. However, these differences are associated to costs and one is interested in finding the mapping corresponding to the minimal global cost, if one exists.

2.2. Existing approaches

A very complete and detailed review of the numerous graph matching techniques may be found in [14]. This paper proposes a taxonomy which distinguishes mainly exact and inexact techniques. In the following, we only examine approaches that deal with subgraph isomorphism, also called monomorphism. Exact techniques may be used to solve the substitution-tolerant subgraph isomorphism problem. For that purpose, a compatibility

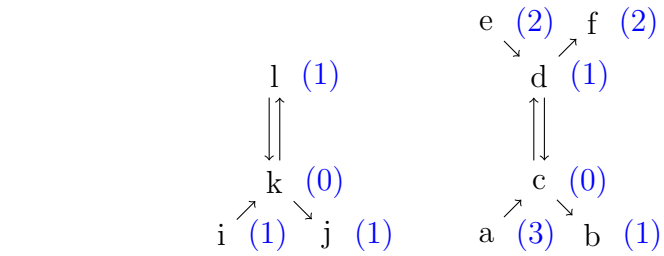
[2] This kind of mapping is also called monomorphism

function must compare labels and decide whether each vertex (resp. edge) of the pattern graph may be mapped to each vertex (resp. edge) of the target graph or not. The compatibility function is generally based on a threshold between label differences. Most of the exact techniques rely on a tree based exploration in a search space, in which each state corresponds to a partial mapping. Backtracking, forward checking and procedures maintaining arc consistency are used to reach a state where all vertices and edges of the pattern graph are mapped to the target one. The exact subgraph isomorphism problem can be modeled as a constraint satisfaction problem [15, 12]. The different techniques proposed in the literature differ from (i) the order in which partial mappings are explored (ii) the filtering caused by the constraint propagation which allows to reduce the search space. In [12], Solnon reviews and compares several approaches by examining the strength of the filtering. In this paper, the author proposes the LAD-filtering (locally all different) which is proven to be stronger than other known techniques (especially [16, 15] when examining the filtering criterion. Experimental results reported in this paper also shows that the LAD-filtering allows to resolve more instances quicker than VF/VF2.

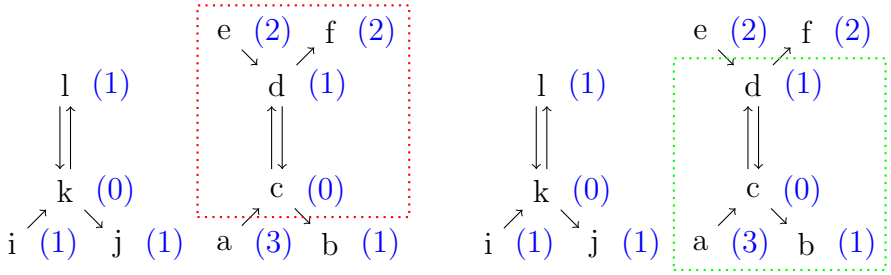
VF [17] is a solver which performs forward checking propagation of edge and difference constraints. The constraint propagation is limited to vertices that are adjacent to vertices already matched. This implies that the subgraph induced by the matched vertices is connected. VF2 [9] is an improved version of VF which reduces its spatial complexity and achieve better performance on large graphs. Experimental resultats reported in [9] illustrate that VF2 clearly outperforms Ullmann's algorithm on the subgraph isomorphism task.

The main drawback of exact techniques for pattern recognition applications is the difficulty to set the thresholds used to decide whether two vertices (resp. edges) can be matched or not, based on their attribute values (regardless structural consideration). If one is interested in a subgraph isomorphism that minimizes the sum of all label differences (or another measure), then this sum must be computed for all matching candidates. In this context, choosing a threshold is puzzling. On the one hand, setting a small threshold reduces the number of matching candidates and speeds up the search, but at the cost of possibly discarding all optimal mappings (see Fig. 1). On the other hand, setting a large threshold results in a costly search among more matching candidates.

In order to overcome this drawback, inexact optimal approaches are guided by the cost of the partial mapping and a heuristic which approx-



(a) $\mathcal{S}$ and $\mathcal{G}$



(b) Best solution for $t = 1 \rightarrow d = 4$ (c) Best solution for $t = 2 \rightarrow d = 2$

Figure 1: Example showing the difficulty of selecting the best threshold t for substitution-tolerant subgraph isomorphism. (a) Definition of $\mathcal{S}$ and $\mathcal{G}$. Two structurally isomorphic matchings exist between $\mathcal{S}$ and $\mathcal{G}$. (b) Solution found by a customized exact matching procedure if the distance threshold t between vertices is set to 1, which leads to a matching error of $d = 4$ [(l,c),(k,d),(i,e),(j,f)]. (c) Solution found by a customized exact matching procedure if t is set to 2, which leads to a matching error of $d = 2$ [(l,d),(k,c),(i,a),(j,b)].

imates the future mapping to explore the search space. The branch and bound algorithm allows to prune branches of the tree that lead to unsuccessful solution as in [18, 19], whereas the A* algorithm dynamically orders branches to explore as a priority. In this latter case, the computationnal performance highly depends on the quality of approximation of the future matching by the heuristics, but the optimal solution is always reached. One the other hand, the complexity issue is tackled by suboptimal optimization techniques. Among them, relaxation techniques like the one proposed in [20] transform the discrete optimization process in a continuous one which can be solve in polynomial time. Other optimization techniques such as those based on neural networks or genetic algorithms [10] have been proposed for graph matching problems. If the polynomial complexity of suboptimal techniques

is appealing, they do not guaranty to provide the matching with the minimal global cost.

This paper proposes an integer linear program to solve the problem of substitution-tolerant subgraph isomorphism, which is an inexact approach for monomorphism. In the literature, linear programming has been proposed to model other graph matching problems. For example, the weighted graph matching problem has been successfully tackled by a linear approach by Almohamad and Duffuaa [21]. More recently, Justice and Hero have proposed a binary linear program for the graph edit distance [22]: two input graphs are first embedded into a clique of size the sum of the graph sizes. Then, an adjacency matrix is shown to encode an isomorphism between an input graph and the clique, while a permutation matrix encodes edit operations. The mathematical formulation then consists in finding a permutation matrix which yields a minimum isomorphism cost.

In the following of this article, we propose an optimization-based approach for solving substitution-tolerant subgraph isomorphism via an Integer Linear Program.

3. Solving the Subgraph Isomorphism Problem using Integer Linear Programming

The main contribution of this paper is the modelization of the substitution-tolerant subgraph isomorphism problem as an optimization problem. The general technique used to solve this problem is Mathematical Programming (MP) which provides tools to model optimization problems as well as resolution algorithms. More precisely, we provide an Integer Linear Program (ILP), i.e. a MP with linear equations and integer variables [23, 24]. (Solving an) ILP is NP-hard, therefore there is no known polynomial-time algorithm to solve general ILP. However, non problem-specific algorithms are continuously designed, improved, and implemented by the ILP community. ILP is known to be one of the most efficient technique for numerous NP-hard optimization problems that appear for instance in production[25], logistics[26], or network design[27]. In the following, we briefly explain how to solve a problem using ILP. Then, we explain how we model the subgraph isomorphism problem. Finally, we disclose the methods used to find multiple close-to-optimal solutions, rather than a single optimal solution.

3.1. Integer Linear Programming

Solving a problem using ILP is a two-step process: it first requires a mathematical modelling of the problem as an integer linear program. The model must then be implemented using a solver. The general form of an ILP is as follows:

$$\min_x \ c^T x \tag{1a}$$

$$\text{subject to } Ax \leq b \tag{1b}$$

$$x \in C \subseteq \mathbb{Z}^n \tag{1c}$$

where $c \in \mathbb{R}^n, A \in \mathbb{R}^{n*m}, b \in \mathbb{R}^m$ are data of the problem.

A solution is a vector x of n variables. In the case of integer programming, x belongs to a discrete set $C \subseteq \mathbb{Z}^n$. C has to be defined according to the modeled problem. A is used to express linear inequality constraints (1b). Hence, a feasible solution for the problem is a vector x such that $x \in C$ and constraints (1c) are respected. The objective function $c^T x$ is a linear combination of all variables x weighted by the vector c. To find an optimal solution, the objective function (1a) is minimized over the set of feasible solutions.

Testing each vector x for feasibility is exponential in the number of variables. Hence, the second step consists in implementing this model using a solver. Such a tool is a complex set of algorithms that interact to implicitly explore the tree of solutions with the so-called Branch and Bound method. Given an instance, the solver finds an optimal solution if one exists.

3.2. Modeling the Subgraph Isomorphism Problem

In the following, for the sake of clarity, the term graph represents a simple directed graph with loops, what means that no more than one edge exists linking a vertex to an other, but the modeling holds for multigraphs. As a consequence, an edge e originating from i and targeting to j and can be denoted ij without any ambiguity.

In order to model the problem as an ILP, we use binary variables, i.e. $C = \{0, 1\}^n$. As depicted in figure 2, two kinds of variable are defined:

- For each pair of vertices $i \in V_\mathcal{S}$ and $k \in V_\mathcal{G}$, there is a variable $x_{i,k}$, such that $x_{i,k} = 1$ if vertices i and k are matched together, 0 otherwise.

- For each pair of edges $ij \in E_{\mathcal{S}}$ and $kl \in E_{\mathcal{G}}$, there is a variable $y_{ij,kl}$, such that $y_{ij,kl} = 1$ if edges ij and kl are matched together, 0 otherwise.

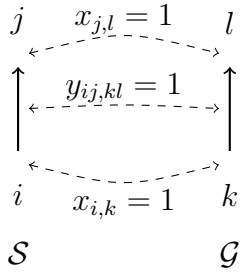


Figure 2: an example of matching. $\mathcal{S}$ and $\mathcal{G}$ both contain a single edge, respectively ij and kl. The following solution is represented on this figure : $x_{i,k} = 1$ (resp. $x_{j,l} = 1$, $y_{ij,kl} = 1$), i.e. i (resp. j, ij) is matched with k (resp. l, kl). Conversely, since i (resp. j) is not matched with l (resp. k), $x_{i,l} = 0$ (resp. $x_{j,k} = 0$).

Given $\mathcal{S} = (V_{\mathcal{S}}, E_{\mathcal{S}})$ and $\mathcal{G} = (V_{\mathcal{G}}, E_{\mathcal{G}})$, let us assume that a cost function $c_V : V_{\mathcal{S}} \times V_{\mathcal{G}} \to \mathbb{R}^+$ as well as a cost function $c_E : E_{\mathcal{S}} \times E_{\mathcal{G}} \to \mathbb{R}^+$ are known. Let us consider two vertices $i \in V_{\mathcal{S}}$ and $k \in V_{\mathcal{G}}$. Pairing i and k (i.e. $x_{i,k} = 1$) costs $c_V(i,k)$, whereas not pairing them (i.e. $x_{i,k} = 0$) costs 0. The cost can thus be written as the linear expression $c_V(i,k) * x_{i,k}$. Similarly, the cost between two edges $ij \in E_{\mathcal{S}}$ and $kl \in E_{\mathcal{G}}$ is $c_E(ij,kl) * y_{ij,kl}$. Let us now write the objective function : the global cost for matching $\mathcal{S}$ to a subgraph of $\mathcal{G}$, which is the sum of the costs for matching vertices and edges, should be minimized:

$$\min_{x,y} \left(\sum_{i \in V_{\mathcal{S}}} \sum_{k \in V_{\mathcal{G}}} c_V(i,k) * x_{i,k} + \sum_{ij \in E_{\mathcal{S}}} \sum_{kl \in E_{\mathcal{G}}} c_E(ij,kl) * y_{ij,kl} \right) \tag{2a}$$

Obviously, equation (2a) does not encode the subgraph isomorphism problem. Let us now consider the constraints of the ILP.

- Every vertex of $V_{\mathcal{S}}$ must be matched to a unique vertex of $V_{\mathcal{G}}$:

$$\sum_{k \in V_{\mathcal{G}}} x_{i,k} = 1 \quad \forall i \in V_{\mathcal{S}} \tag{2b}$$

- Every edge of $E_{\mathcal{S}}$ must be matched to a unique edge of $E_{\mathcal{G}}$:

$$\sum_{kl \in E_{\mathcal{G}}} y_{ij,kl} = 1 \quad \forall ij \in E_{\mathcal{S}} \tag{2c}$$

- Every vertex of $V_{\mathcal{G}}$ must be matched to at most a vertex of $E_{\mathcal{S}}$:

$$\sum_{i \in V_{\mathcal{S}}} x_{i,k} \leq 1 \quad \forall k \in V_{\mathcal{G}} \tag{2d}$$

- If two vertices are matched together, an edge originating the vertex of $\mathcal{S}$ must be matched with an edge originating the vertex of $\mathcal{G}$:

$$\sum_{kl \in E_{\mathcal{G}}} y_{ij,kl} = x_{i,k} \quad \forall k \in V_{\mathcal{G}}, \forall ij \in E_{\mathcal{S}} \tag{2e}$$

- If two vertices are matched together, an edge targeting the vertex of $\mathcal{S}$ must be matched with an edge targeting the vertex of $\mathcal{G}$:

$$\sum_{kl \in E_{\mathcal{G}}} y_{ij,kl} = x_{j,l} \quad \forall l \in V_{\mathcal{G}}, \forall ij \in E_{\mathcal{S}} \tag{2f}$$

Finally, let us properly write the aforementioned domain constraints of the variables:

$$x_{i,k} \in \{0, 1\} \quad \forall i \in V_{\mathcal{S}}, \forall k \in V_{\mathcal{G}} \tag{2g}$$

$$y_{ij,kl} \in \{0, 1\} \quad \forall ij \in E_{\mathcal{S}}, \forall kl \in E_{\mathcal{G}} \tag{2h}$$

Equations (2a) to (2h) form the ILP used to solve the Subgraph Isomorphism Problem. It might be the case that the problem is infeasible, if there exists no graph isomorphism between $\mathcal{S}$ and any subgraph of $\mathcal{G}$. Otherwise, a solver implementing this model will return the best solution found, i.e. the best subgraph isomorphism.

3.3. Finding Multiple Close-to-Optimal Solutions

Depending on the application context, it may be the case that the pattern graph that is searched for has many instances in the target graph. As defined in subsection 3.2, the ILP model is only capable of finding the optimal solution. There is no out-of-the-box free implementation to find multiple close-to-optimal solutions in ILP. There are, however, multiple ways to achieve

this [28]. In the context of our study, we have chosen to call iteratively the model and to discard the successive optimal solutions after each call. Such a solution is linear in the number of instances. There are multiple ways to discard an optimal solution. The general idea is that a new constraint cutting the current solution is added to the model. Hence, the current optimal solution becomes infeasible for the next run. The solver can be called again and will be able to find another optimal solution. In the sake of our study, the following constraint is added to the formulation :

$$\sum_{i \in V_{\mathcal{S}}, k \in V_{\mathcal{G}}} \left(\sum_{j \in V_{\mathcal{S}}} \bar{x}_{j,k} \right) * x_{i,k} = 0$$

Its purpose is to discard every vertex of $\mathcal{G}$ that has been used in the current optimal solution $(\bar{x}\ \bar{y})^T$. It means that for every vertex k of $\mathcal{G}$, if there exists a vertex j of $\mathcal{S}$ matched to k, then $x_{i,k}$ equals 0 for every vertex i of $\mathcal{S}$.

4. Experiments and results

The evaluation of subgraph isomorphism algorithms is a difficult task since it implies to have at disposal (i) some graph and subgraph datasets and (ii) the position of each pattern graph in the target graphs, *i.e.* the mapped vertices and edges. For graphs extracted from a real world application, this information is difficult and time consuming to be collected since the mapping has to be defined in relation to the application. As an example, for graphs representing objects in an image, defining the mapping require to consider both graphs and images. This difficulty is increased when error-tolerant isomorphism is involved. In order to overcome this difficulty, most of the papers dealing with subgraph isomorphism propose an evaluation relying on synthetic datasets, such as the well known VF dataset described in [17] or those provided on the IAPR TC-15 website[3]. However, to the best of our knowledge, all existing datasets rely on a nominal labeling and we are not aware of publicly available graph dataset with scalar or vectorial numerical attributes.

In this context, we propose in this paper a two step evaluation of the proposed algorithm. In the first step, the evaluation is led on a synthetic

[3]http://www.greyc.ensicaen.fr/iapr-tc15

graph dataset. Its aim is to validate the functioning of the ILP approach and to quantify the effectiveness of the algorithm w.r.t. the size of graphs. Both exact and inexact graph matchings are considered in this part of the experiments and a performance comparison with VF2 [9] and LAD [12] algorithms is proposed. These algorithms have been chosen because they are known to be very efficient for different graph matching problems (see 2.2 for their description). In the second step, the proposed algorithm is evaluated in a real-world context, on a document analysis problem.

4.1. Experiments on synthetic data

In this subsection, we first describe the data before presenting and discussing the obtained results.

4.1.1. Data generation

Two sets of synthetic graphs have been generated to evaluate the exact and the substitution-tolerant subgraph isomorphism. These sets have been generated considering the same procedure. Firslty, a random graph $\mathcal{S}$ is generated according the Erdös-Rényi model [29]. Then, a graph $\mathcal{G}_0$ is created as an exact copy of $\mathcal{S}$. In the case of substitution-tolerant matching, labels of $\mathcal{G}_0$ can be modified by applying a noise model. Finally, $\mathcal{G}_0$ is completed to form a graph $\mathcal{G}$ with vertex and edge insertions according the Erdös-Rényi model until a predefined size.

The informations provided during the creation and the modifications of $\mathcal{G}_0$ are recorded and considered as the ground-truth information. $\mathcal{G}_0$ is called the ground-truth solution. Note that in a noiseless model, there might be multiple optimal solutions, and in a noisy model, the ground-truth solution $\mathcal{G}_0$ may not be the optimal solution. This procedure provides a couple of graphs $(\mathcal{S}, \mathcal{G})$. The following values of the parameters have been chosen for our experiments :

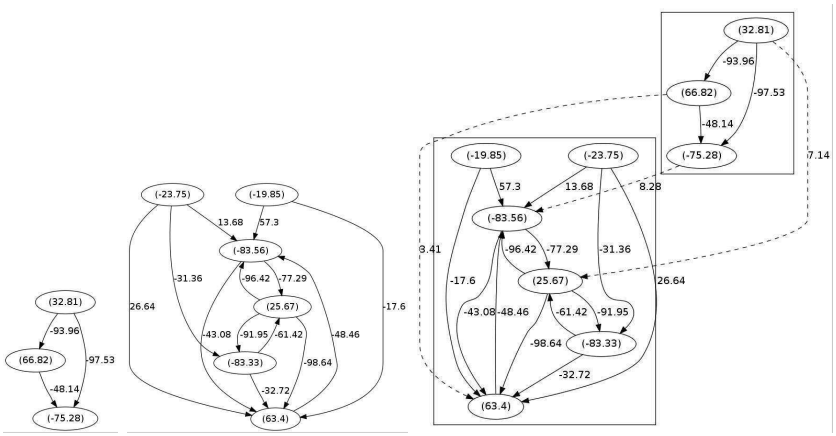
- Size of $\mathcal{G}$: $|V_G| = n_{\mathcal{G}} \in \{50, 100, 250, 500\}$
- Size of $\mathcal{S}$: $|V_S| = n_{\mathcal{S}} \in \{10, 25, 50\}$
- Probability that an edge connects two vertices : $p \in \{0.01, 0.05, 0.1\}$
- vertex and edge labels have been randomly chosen in $[-100, 100]$ with a uniform distribution.

Using this parameterization, two test sets have been generated :

- The first set, denoted as T_1, contains 5 couples $(\mathcal{S}, \mathcal{G})$ generated for each combination of $(n_\mathcal{G}, n_\mathcal{S}, p)$. In this set, no noise has been applied, so the mapping between $\mathcal{S}$ and $\mathcal{G}$ is an exact isomorphism.

- The second set, denoted as T_2 aims at evaluating the subgraph isomorphism search in presence of noise. These data have been generated using a Gaussian noise ($\sigma = 5$) applied on both vertices and edges before the graph is completed to its final size.

Figure 3 illustrates an output produced by the synthetic data generation in the presence of noise.



(a) A graph $\mathcal{S}$ (b) A graph $\mathcal{G}$ (c) Ground-truth for the mapping between $\mathcal{S}$ and $\mathcal{G}$.

Figure 3: Illustration of the generation of synthetic data for the substitution-tolerant subgraph isomorphism search with the following parameters : $n_\mathcal{S} = 3$, $n_\mathcal{G} = 6$, $p = 0.3$, labels in [-100,100] and a Gaussian noise ($\sigma = 5$). The mapping cost is 24.93

4.1.2. Obtained results

Given the two datasets T_1 and T_2 described above, two experiments have been led. They relate respectively to exact and substitution-tolerant subgraph isomorphisms.

(a) Exact subgraph isomorphism . These first experiments aim at (i) validating the ILP formulation proposed in 3.2 in the case of exact matchings and (ii) comparing the efficiency of the proposed approach w.r.t. reference approaches. In this context, the results obtained using our algorithm have been compared with those obtained with VF2 [9] and LAD [12]. For these experiments, the matching cost functions c_v and c_e have been defined as the Euclidean distance between the attribute values. For each of the couple $(\mathcal{S}, \mathcal{G})$ generated for each combination of $(n_\mathcal{G}, n_\mathcal{S}, p)$, as expected, we have observed that the isomorphisms are exactly the same for all the algorithms. Table 1 gives the time needed to solve the five subgraph isomorphism instances for each configuration.

p	$n_\mathcal{G}$	$n_\mathcal{S} = \lvert V_\mathcal{S} \rvert$								
		10			25			50		
		ILPiso	vf2	lad	ILPiso	vf2	lad	ILPiso	vf2	lad
0.01	50	0.02	0.03	0.0	0.04	0.02	0.0	0.08	0.02	0.0
	100	0.02	0.02	0.0	0.06	0.01	0.0	0.1	0.02	0.0
	250	0.028	0.02	0.02	0.1	0.02	0.02	0.28	0.02	0.02
	500	0.08	0.06	0.01	0.26	0.05	0.04	0.72	0.06	0.01
0.05	50	0.03	0.02	0.0	0.06	0.02	0.0	0.3	0.02	0.02
	100	0.06	0.01	0.02	0.21	0.02	0.01	0.76	0.02	0.02
	250	0.15	0.03	0.04	0.85	0.04	0.04	4.24	0.05	0.04
	500	0.68	0.14	0.34	2.87	0.14	0.4	14.82	0.13	0.4
0.1	50	0.05	0.02	0.01	0.22	0.02	0.01	1.28	0.02	0.02
	100	0.1	0.02	0.01	0.63	0.02	0.01	3.1	0.01	0.01
	250	0.42	0.06	0.1	3.64	0.01	0.14	15.52	0.06	0.08
	500	1.90	0.24	1.26	15.33	0.22	1.33	42.33	0.24	1.25

Table 1: Results obtained on T_1 for the proposed approach, VF2 and LAD. This table gives the sum of the processing time in seconds for the five solved instances.

As one can see in this table, VF2 and LAD are clearly more efficient than the proposed ILP approach for exact matching problem. Such a result is quite natural since approaches do not rely on the same paradigm. VF and LAD algorithms rely on a decision process, whereas the proposed approach relies on an optimization process. More precisely, exact algorithms can prune the solution tree very early, at a vertex's level, since for a vertex matching, a single label difference means every solution with this matching is infeasi-

ble. The algorithm we propose is by nature error-tolerant and accepts label differences, making the solution tree harder to prune. As a consequence, it is natural that it is outperformed by dedicated algorithms.

(b) Substitution-tolerant subgraph isomorphism. These second experiments on the T_2 dataset aim at validating the real contribution of the proposed approach, i.e. the ability of an ILP based approach to tackle problems where a substitution-tolerant isomorphism is required. First of all, one can note that if the procedure used to generate synthetic data guarantees that the mapping between $\mathcal{S}$ and $\mathcal{G}$ corresponds to an exact or substitution-tolerant subgraph isomorphism, it does however not ensure that this mapping is unique. Indeed, the completion of $\mathcal{G}$ until its final size may produce new subgraph isomorphisms, and some of them may have a lower matching cost. It is difficult to keep track of this in the case of data generation for the substitution-tolerant problem. The ground-truth information produced during the generation phase can then be only partial. However, when doing a multiple search as explained in 3.3, it is possible to check that the initial solution is in the list of returned solutions. Such tests have shown that the ground-truth solution is always in this list. Moreover, for the T_2 dataset, it is always the first one. Concerning the processing time, the results presented in table 2, compared with those of table 1 illustrate that more time is needed when an substitution-tolerant mapping is searched for. One reason is that T_1 dataset is such that there is always a feasible solution of cost 0, which is necessarily optimal, since the objective function is a sum of non-negative terms. Once this solution is identified by the Branch and Bound, the search can stop without further tree exploration. For the T_2 dataset, the optimal value is not 0 in general, thus the tree exploration may continue after finding the optimal solution, in order to certify that no better solution exists.

Comparing these results with existing approaches such as those used for exact matching is a difficult task since they have not been designed to tackle such problems. Nevertheless, as explained in 2.2, it is possible to customize them in order to make them applicable. Hence, given two thresholds t_v and t_e (one for the vertices, and one for the edges), two vertices (resp. edges) are considered as compatible if the distance between their attribute is lower than t_v (resp. t_e). For our experiments, we have used an extended version of LAD supplied by the author of [12] to evaluate such a strategy [4]. In this

[4]We would like to thank the author for his help concerning our experiments

p	$n_{\mathcal{G}} = \lvert V_{\mathcal{G}} \rvert$	$n_{\mathcal{S}} = \lvert V_{\mathcal{S}} \rvert$		
		10	25	50
0.01	50	0.02	0.04	0.08
	100	0.04	0.05	0.12
	250	0.07	0.14	0.46
	500	0.08	0.39	1.76
0.05	50	0.02	0.10	0.52
	100	0.04	0.27	1.82
	250	0.19	1.58	12.68
	500	1.17	9.51	49.87
0.1	50	0.07	0.48	9.60
	100	0.19	1.60	26.34
	250	1.55	16.18	108.76
	500	2.50	98.75	370.8

Table 2: Results obtained on T_2 for the proposed approach. This table gives the sum of the processing time in seconds for the five solved instances.

experiment, since the label distribution is the same for vertices and edges, we used a unique threshold $t = t_v = t_e$. The obtained results are given in table 3. It provides the number of instances that are solved by LAD when submitted the same five queries than in table 2 by varying the value of t in $\{5, 10, 20\}$ *i.e* in $\{\sigma, 2\sigma, 4\sigma\}$.

As one can see in table 3, many instances (389/540) are not solved using LAD. Two reasons explain these failures: (i) For "small" values of t, few solutions are found by the system, because of the phenomenon explained in 2.2 and illustrated by figure 1. Hence, a threshold of 5 (resp. $10, 20$) only solves 9 (resp. $29, 113$) of the 180 instances. (ii) For larger values of t, the number of solved instances increase. However, the problem sometimes becomes intractable because of the important number of vertices (and edges) that can be matched. Thus, the system is flooded with feasible solutions. This result has to be correlated to the density of the graphs. Hence, when the graph density increases, more instances are solved: for $p = 0.01$ (resp. $0.05, 0.1$), 14 (resp. $65, 72$) instances are solved.

Hence, the choice of the threshold values is crucial since low values can eliminate good mappings, and since large values can lead to an intractable problem. As an illustration, figure 4 shows the number of solutions and the

p	$n_{\mathcal{G}}$	$n_{\mathcal{S}} = \lvert V_{\mathcal{S}} \rvert$								
		10			25			50		
		5	10	20	5	10	20	5	10	20
0.01	50	2/3	4/1	2/0	0/5	0/2	0/0	0/5	0/5	4/0
	100	2/2	0/1	0/0	0/5	0/1	0/0	0/5	0/4	0/0
	250	0/2	0/0	0/0	0/5	0/1	0/0	0/5	0/4	0/0
	500	0/1	0/0	0/0	0/5	0/1	0/0	0/5	0/4	0/0
0.05	50	0/5	5/0	5/0	0/5	0/5	5/0	0/5	0/5	5/0
	100	2/3	2/2	4/0	0/5	1/4	5/0	0/5	0/5	5/0
	250	1/3	1/2	2/0	0/5	1/4	4/0	0/5	0/5	5/0
	500	1/4	2/1	2/0	0/5	0/5	3/0	0/5	0/5	4/1
0.1	50	0/5	2/3	5/0	0/5	0/5	5/0	0/5	0/5	5/0
	100	0/5	3/2	5/0	0/5	0/5	5/0	0/5	0/5	5/0
	250	0/5	3/2	5/0	0/5	0/5	5/0	0/5	0/5	5/0
	500	1/4	4/1	3/0	0/5	1/4	5/0	0/5	0/5	5/0

Table 3: Results of LAD on the T_2 dataset for threshold values in $\{5, 10, 20\}$. The first number gives the number of solved instances among the 5 queries. The second number gives the number of instances for which LAD did not found any solution. The remaining instances are not solved by LAD.

processing time necessary to find all the solutions according to the threshold values (the same value is considered here for both vertices and edges but as a matter of fact, they can be distinguished). As one can see, the number of compatible graphs as well as the time needed rapidly grows when the threshold increases. One can mention that all the solutions have to be sorted after this search. Moreover, when using an exact approach, there is no absolute guaranty that the optimal solution is among the found instances (as for the example of figure 1). According to us, these results illustrate the important interest to consider subgraph isomorphism as an optimization process, which is driven by the objective function under some constraints. Hence, the proposed approach is not the most efficient for all problems, but it does not need to define a threshold, and it always give the optimal solution.

In the next section, we present the second part of our experiments, that concerns an application based on substitution-tolerant subgraph isomorphism for symbol detection.

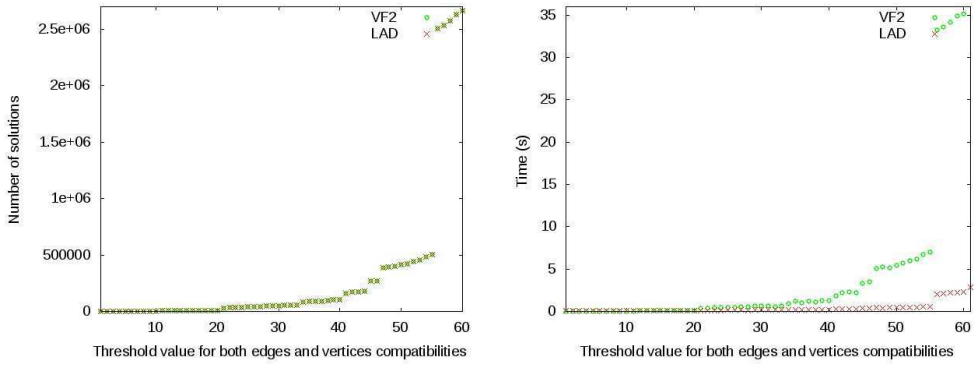


Figure 4: Evolution of the number of subgraph isomorphism solutions and the computation time found by VF2 and LAD according to a threshold value for the following parameters values : $(n_{\mathcal{G}} = 250, n_{\mathcal{S}} = 25, p = 0.1)$

4.2. Symbol detection

Symbol detection is a problem related to document analysis that aims at searching the occurrences of some reference symbols in an input document image. In the literature, this problem is tackled using either pixel based approaches [30, 31], symbol signature [32, 33] or structural approaches [34, 35, 36, 37]. In the latter case, the proposed techniques theoretically allow to overcome the segmentation/recognition paradigm, but they require a subgraph isomorphism algorithm to detect the instances of the model graph in the graph describing the whole document. The remaining of this section presents how our approach for subgraph isomorphism has been applied to solve the symbol detection problem. First, the graph database is described. Then, the experimental protocol and the results obtained for multiple experiments are presented.

4.2.1. Graph database

The database used for this experiment is made of Region Adjacency Graphs (RAG) extracted from images of technical documents. In a RAG, vertices are associated to the regions of an image and edges are created between pairs of vertices if the corresponding regions are adjacent. The algorithm used for extracting such RAG's is fully described in [13]. Using this algorithm, vertices are labelled with a feature vector corresponding to a set of Zernike Moments (ZM)[38] and edges are labelled with *relative scale* and *relative distance*. Once a document image and the symbol to be searched have

been described as RAG's, the symbol detection problem turns into a subgraph isomorphism problem. Moreover, this subgraph isomorphism search has to be substitution-tolerant. Indeed, due to noise on images, the labeling may differ between the graph which represents the symbol and its instance in the graph representing the whole document.

The document images used for this evaluation are extracted from the `floorplan` dataset [39][5]. This dataset is made of synthetic images representing several symbol arrangements on 10 empty architectural plan templates. Our experiments have been performed using 200 images corresponding to the 20 first images for each template. Figure 5 shows two examples of plans and the corresponding RAG's. Figure 6 represent the 16 symbol models. The whole plan dataset contains 5609 symbol instances, with an average of 28 instances per document image. The graphs corresponding to symbol instances contain 4 vertices and 7 edges on average. As a comparison, the structural representations of the plans contain 121 vertices and 525 edges on average.

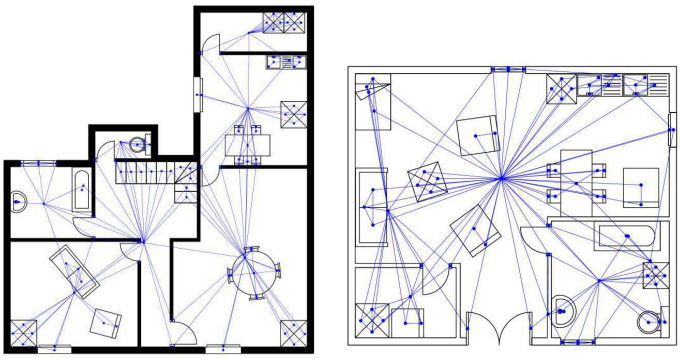


Figure 5: Examples of plans from the `floorplan` dataset with the corresponding RAG's

4.2.2. Performance criteria

The search of a substitution-tolerant subgraph isomorphism results in a one-to-one mapping between every vertices and edges from the graph representing the symbol and some vertices and edges from the graph representing the whole document. In order to be assessed, the proposed mapping has to

[5] `http://mathieu.delalandre.free.fr/projects/sesyd/`

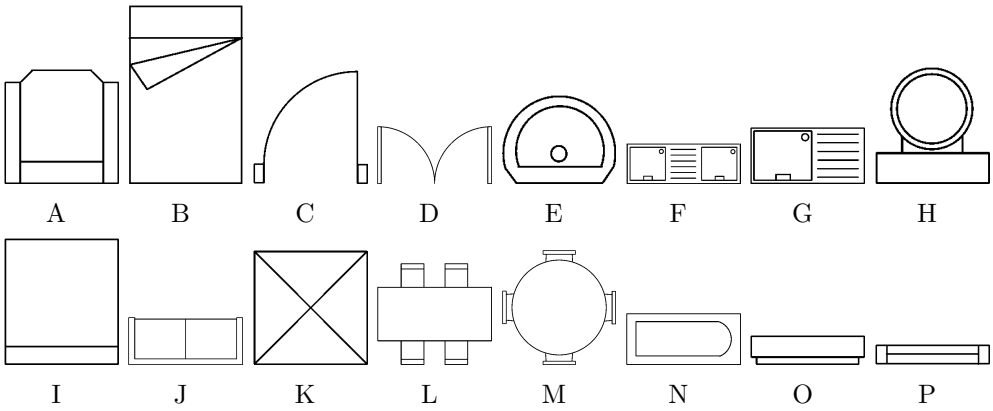


Figure 6: Symbol models

be compared with the ground-truth information. This comparison can lead to the following cases:

- the mapping exactly corresponds to the groundtruth (denoted as $=$);
- a part of the mapping corresponds to the groundtruth (denoted as $\approx$);
- the mapping does not correspond to the groundtruth (denoted as $\neq$);
- no mapping is proposed. This case occurs when no solution fulfills the constraints of the ILP (denoted as $\emptyset$).

When an erroneous mapping is proposed, two cases can be distinguished:

- the symbol appears on the document image (denoted as $\neq |+$);
- the symbol has no instance on the document image. (denoted as $\neq |-$).

4.2.3. Search of a single occurrence

The task which has been evaluated in our first experiments was to locate the most probable instance of a given symbol in a plan, *i.e.* the minimum cost subgraph isomorphism. This experiment has been performed on 16 symbols and 200 plans, leading to 3200 subgraph isomorphism "queries". For all these queries, the system has always proposed a feasible solution. The results are reported on table 4. They show that, among the 3200 found instances, 1719 exactly correspond to a symbol and 337 partly correspond to an actual mapping. 755 searches returned in an erroneous result which

is explained by the fact that the symbol was not on the document. Finally, only 389 confusions were returned among the 2445 cases where the searched symbol should have been found. Table 4 also reveals that the performance are not equally distributed among the symbol classes. For example, symbols from class E, M or N were quite well located, whereas searches of instances of C, D or G result in numerous confusions. Two explanations can be given for these errors. On one hand, it may be noted that the region adjacency graph representation is not well suited to describe symbols C or D. Indeed, these symbols only contain two non adjacent regions. Thus, the corresponding graphs are composed of two vertices without edge. As a consequence, no topological constraint is used by the subgraph isomorphism, which leads to erroneous mappings. On the other hand, many confusions occur between F and G because the graph for symbol G is a subgraph of the graph for symbol F.

Symbol	#	$=$	$\approx$	$\neq \mid +$	$\neq \mid -$	$\emptyset$
A	106	106	0	0	94	0
B	160	46	76	38	40	0
C	200	56	72	72	0	0
D	60	0	11	49	140	0
E	112	112	0	0	88	0
F	130	129	1	0	70	0
G	151	51	21	79	49	0
H	188	40	128	20	12	0
I	176	114	0	62	24	0
J	192	191	1	0	8	0
K	200	158	25	17	0	0
L	123	123	0	0	77	0
M	91	89	2	0	109	0
N	180	180	0	0	20	0
O	183	131	0	52	17	0
P	193	193	0	0	7	0
Total		1719	337	389	755	0

Table 4: Results for the search of the most probable instance of each symbol on the 200 plans of the `floorplan` dataset. # denotes here the number of plans which contain the symbol

4.2.4. Search of multiple occurrences

Considering that a symbol can have several instances on the same document image, a second experiment has been dedicated to evaluate the performance of a multiple search. The algorithm was configured so that any vertex mapping in a previous solution is excluded from the search space because a region can be part of at most one symbol. For a search of 50 instances of all symbol models in the 200 plans, a perfect system should find 5609 symbol instances. Among them, the proposed approach returns exactly 3911 mappings (69.7%) and partially 1636 mappings (29.2%). The 62 symbol instances which are not found correspond to cases where two symbols from the same class, made of identical regions (e.g. symbol K composed of four triangles) are closed in the plan so that the mapping is done with regions of both occurrences. In such a case, the second occurrence cannot be found. Considering that a partial matching is sufficient to consider the symbol as detected, the overall recall is thus 99% and the overall recall is 10%. Table 5 details the results per class when 50 occurrences are considered. As one can see in this table, a significant number of false detections are made. This can be explained by the fact that the search of 50 instances has led the system to propose solutions whereas all the actual symbols were already found.

These results highlight a need for some improvements. Indeed, if the recall rate may be considered as satisfactory, the number of false detections has to be drastically reduced. This goal may be achieved by implementing a rejection strategy which is able to filter the search results according to a threshold used to stop the search. Figure 7 illustrates the results that such a strategy can produce on precision-recall curves, when varying the value of the threshold.

4.2.5. Learning of distance threshold

The results presented above show that it is necessary to stop the search using a threshold on the matching cost when considering multiple occurrences, in order to improve the precision of the system. As shown by figure 8, which depicts the ranges of matching costs obtained for each class of symbols for the 3200 queries, the threshold has to be tuned according to the symbol class which is considered. Four our experiments, this threshold has been learned on a subset of 20 floorplans by optimizing the F1-measure which is a tradeoff between recall and precision (see eq 3). The obtained values are highlighted as blue strokes on figure 8.

Symbol	Recall (%)	Precision (%)
A	100	9
B	99	10
C	100	9
D	97	1
E	100	2
F	100	14
G	100	7
H	100	5
I	100	5
J	100	7
K	88	32
L	100	12
M	100	8
N	100	2
O	100	6
P	100	28
overall	99	10

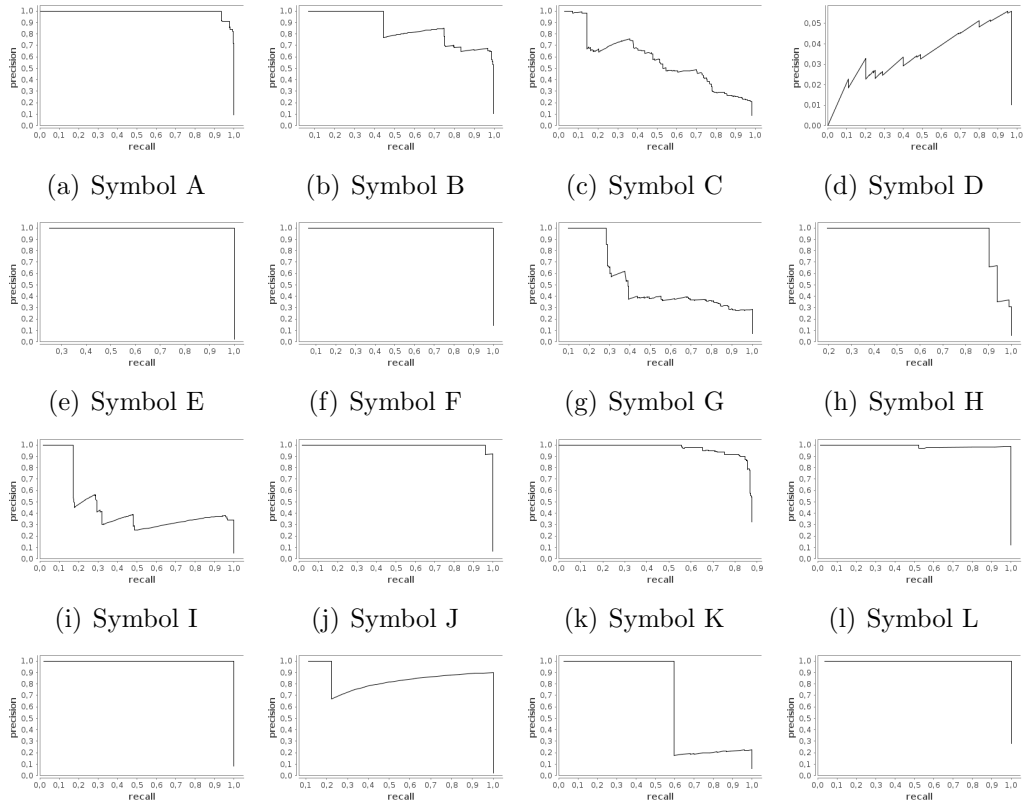
Table 5: Recall and precision for the search of 50 instances

$$\text{F1-mesure} = 2\frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3}$$

Table 6 presents the rejection threshold for each class and the new values for recall and precision, obtained on 180 floorplans. The diversity of the threshold values justifies the choice to set a class dependent threshold. It might be noted that these values are somehow correlated with the size of the model graph. However the main observation is the performance evolution achieved by the use of this rejection strategy. Indeed, for each symbol class, a significant improvement in precision was obtained at the cost of a slight decrease of the recall. The overall performance of the system reaches a precision of 81% while keeping a high value for the recall (90%).

5. Conclusion

In this paper, an integer linear formulation has been proposed to solve the problem of substitution-tolerant subgraph isomorphism. This optimization



(a) Symbol A (b) Symbol B (c) Symbol C (d) Symbol D

(e) Symbol E (f) Symbol F (g) Symbol G (h) Symbol H

(i) Symbol I (j) Symbol J (k) Symbol K (l) Symbol L

(m) Symbol M (n) Symbol N (o) Symbol O (p) Symbol P

Figure 7: Precision-recall curves per class for the search of 50 instances

approach enables us to find instances of a query graph into a larger one the attribute of which may differ. Results have been presented for experiments performed on synthetic graph datasets. We also report results obtained on graph-based representations extracted from architectural plans where a subgraph isomorphism corresponds to a symbol instance. For these latter, we have proposed a rejection strategy based on the learning of matching cost thresholds. This strategy improves the precision by greatly reducing false detections while keeping the good recall performance achieved by a multiple search.

The main characteristics of the proposed approach are the following. First, it is very general in the sense that it can be applied to the most general class of graphs. Indeed, it can process directed multigraphs (several edges

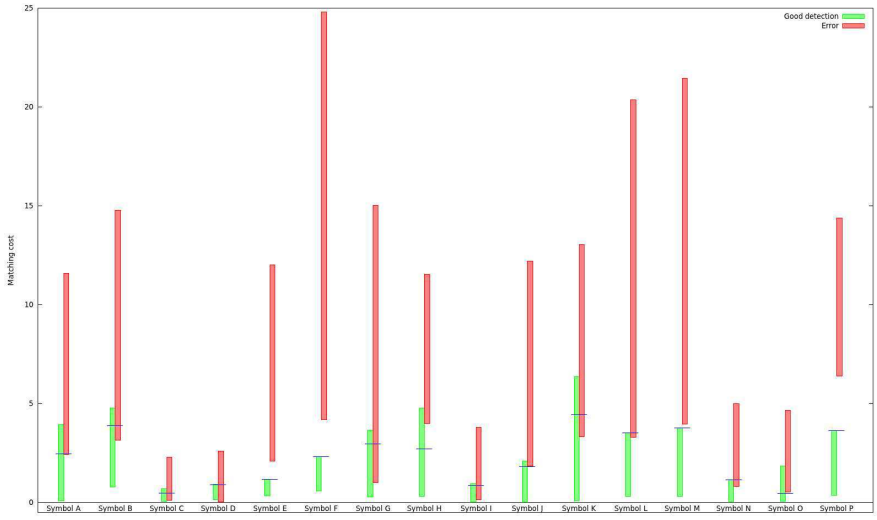


Figure 8: Ranges of mapping costs for good detection and errors according to the class of symbol. The stroke line represents the threshold that has been learned.

may exist between edges, even loops). The graph can be labeled on both vertices and edges and there is no restriction considering the type of label, only a distance or a cost function to compare labels of two different vertices or edges are needed. Neither the pattern nor the target graph need to be connected. Second, in contrast to the approach by Messmer and Bunke [40], the graph database does not need to be preprocessed. The subgraph isomorphism search can be directly applied to two newly presented graphs. Finally, the approach presented in this paper separates the formulation of problem and the method used to solve it. The error-tolerant subgraph isomorphism is modeled as an optimization problem under constraints. The responsibility to solve this problem can be delegated to any solver able to process an ILP formulation. Such solvers are continuously developed by the combinatorial optimization community.

The work described in this paper can be pursued following several leads. First, concerning the symbol detection problem, the subgraph isomorphism approach could be applied to vector or contour-based structural descriptions of symbols and documents as the ones proposed by [37] or [41], in order to overcome some of the limits of region-based descriptions. At the same time, in order to manage possible segmentation errors which may result in merged or split vertices, operators establishing distances for one-to-many or

Symbol	Matching cost threshold	Recall (%)	Precision (%)
A	2.451	93	100
B	3.875	75	85
C	0.489	70	49
D	0.889	95	5
E	1.165	100	100
F	2.315	100	100
G	2.969	76	38
H	2.710	90	100
I	0.862	95	38
J	1.831	96	100
K	4.435	84	90
L	3.513	100	98
M	3.767	100	100
N	1.144	100	90
O	0.447	60	100
P	3.625	100	100
overall		90	81

Table 6: Precision and recall of the symbol spotting system implementing the rejection strategy based on the maximization of the F-measure

many-to-one mappings [42] should be considered. This remark could even be integrated at the isomorphism level. Indeed, the integer linear programming formulation proposed for substitution-tolerant subgraph isomorphism could be adapted to other mapping problems such as substitution-tolerant induced subgraph isomorphism or isomorphisms allowing topological transformations. Also, the overall running time of the algorithm can be improved by further customizing the ILP solver heuristics.

Finally, considering the lack of public ground-truthed dataset to evaluate subgraph isomorphism in presence of graph labeled with numerical values, the ones used in this paper are available on demand and we intend to make them publicly accessible for download.

6. References

[1] M. Kuramochi, G. Karypis, Finding frequent patterns in a large sparse graph, Data Mining and Knowledge Discovery 11 (3) (2005) 243–271.

[2] A. Inokuchi, T. Washio, H. Motoda, Complete mining of frequent patterns from graphs: Mining graph data, Machine Learning 50 (3).

[3] E. Barbu, P. Héroux, S. Adam, E. Trupin, Frequent graph discovery: Application to line drawing document images, Electronic Letters on Computer Vision and Image Analysis (ELCVIA) 5 (2) (2005) 47–57.

[4] M. Neuhaus, H. Bunke, Edit distance-based kernel functions for structural pattern classification, Pattern Recognition 39 (10) (2006) 1852–1863.

[5] R. Raveaux, S. Adam, P. Héroux, E. Trupin, Learning graph prototypes for shape recognition, Computer Vision and Image Understanding 115 (7) (2011) 905 – 918. doi:10.1016/j.cviu.2010.12.015.

[6] H. Zanghi, C. Ambroise, V. Miele, Fast online graph clustering via Erdos Renyi mixture, Pattern Recognition 41 (12) (2008) 3592–3599.

[7] H. Qiu, E. Hancock, Graph matching and clustering using spectral partitions, Pattern Recognition 39 (1) (2006) 22–34.

[8] F. E. M.A. Lozano, Protein classification by matching and clustering surface graphs, Pattern Recognition 39 (4) (2006) 539–551.

[9] L. P. Cordella, P. Foggia, C. Sansone, M. Vento, A (sub)graph isomorphism algorithm for matching large graphs, IEEE Trans. Pattern Anal. Mach. Intell. 26 (10) (2004) 1367–1372.

[10] S. Auwatanamongkol, Inexact graph matching using a genetic algorithm for image recognition, Pattern Recognition Letters 28 (12) (2007) 1428–1437.

[11] S. Zampelli, Y. Deville, C. Solnon, Solving subgraph isomorphism problems with constraint programming, Constraints 15 (3) (2010) 327–353.

[12] C. Solnon, Alldifferent-based filtering for subgraph isomorphism, Artificial Intelligence 174 (12-13) (2010) 850 – 864.

[13] P. Le Bodic, H. Locteau, S. Adam, P. Héroux, Y. Lecourtier, A. Knippel, Symbol detection using region adjacency graphs and integer linear programming, in: Proceedings of the International Conference on Document Analysis and Recognition (ICDAR'09), 2009, pp. 1320–1324.

[14] D. Conte, P. Foggia, C. Sansone, M. Vento, Thirty years of graph matching in pattern recognition, International Journal of Pattern Recognition and Artificial Intelligence 18 (3) (2004) 265–298.

[15] J. Larrosa, G. Valiente, Constraint satisfaction algorithms for graph pattern matching, Mathematical Structures in Computer Science 12 (4) (2002) 403–422.

[16] J. R. Ullmann, An algorithm for subgraph isomorphism, J. ACM 23 (1) (1976) 31–42.

[17] L. P. Cordella, P. Foggia, C. Sansone, M. Vento, Performance evaluation of the VF graph matching algorithm, in: Proccedings of the International Conference on Image Analysis and Processing, 1999, pp. 1172–1177.

[18] D. E. Ghahraman, A. K. C. Wong, T. Au, Graph optimal monomorphism algorithms, IEEE Transactions on System, Man and Cybernetics 10 (1980) 181–188.

[19] A. K. C. Wong, M. You, S. C. Chan, An algorithm for graph optimal monomorphism, IEEE Transactions on System, Man and Cybernetics 20 (3) (1990) 628–638.

[20] R. C. Wilson, E. R. Hancock, Structural matching by discrete relaxation, IEEE Transactions on Pattern Analysis and Machine Intelligence 19 (1997) 634–648.

[21] H. A. Almohamad, S. O. Duffuaa, A linear programming approach for the weighted graph matching problem, IEEE Transaction on Pattern Analysis and Machine Intelligence 15 (5) (1993) 522–525.

[22] D. Justice, A. Hero, A binary linear programming formulation of the graph edit distance, IEEE Transactions on Pattern Analysis and Machine Intelligence 28 (8) (2006) 1200–1214.

[23] G. L. Nemhauser, L. A. Wolsey, Integer and combinatorial optimization, Wiley-Interscience, New York, NY, USA, 1988.

[24] A. Schrijver, Theory of Linear and Integer Programming, John Wiley & Sons, New York, NY, USA, 1998.

[25] H. Kellerer, U. Pferschy, D. Pisinger, Knapsack Problems, Springer, Berlin, Germany, 2004.

[26] D. L. Applegate, R. E. Bixby, V. Chvatal, W. J. Cook, The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics), Princeton University Press, 2007.

[27] F. K. Hwang, D. S. Richards, P. Winter, The Steiner Tree Problem, Vol. 53 of Annals of Discrete Mathematics, North-Holland, Amsterdam, Netherlands, 1992.

[28] E. Danna, M. Fenelon, Z. Gu, R. Wunderling, Generating multiple solutions for mixed integer programming problems, in: IPCO '07: Proceedings of the 12th international conference on Integer Programming and Combinatorial Optimization, Springer-Verlag, Berlin, Heidelberg, 2007, pp. 280–294.

[29] P. Erdös, A. Rényi, On random graphs, Publicationes Mathematicae 6 (1959) 290–297.

[30] S. Tabbone, L. Wendling, K. Tombre, Matching of graphical symbols in line-drawing images using angular signature information, International Journal on Document Analysis and Recognition 6 (2) (2003) 115–125.

[31] S. Adam, J. Ogier, C. Cariou, R. Mullot, J. Labiche, J. Gardes, Symbol and character recognition: application to engineering drawings, International Journal of Document Analysis and Recognition (IJDAR) 3 (2) (2000) 89–101.

[32] P. Dosch, J. Lladós, Vectorial signatures for symbol discrimination, in: Graphics Recognition: Recent Advances and Perspectives, Vol. 3088 of Lecture Notes in Computer Science, 2004, pp. 154–165.

[33] W. Zhang, L. Wenyin, A new vectorial signature for quick symbol indexing, filtering and recognition, in: Proceedings of the Ninth International Conference on Document Analysis and Recognition, 2007, pp. 536–540.

[34] J. Lladós, E. Martí, J. J. Villanueva, Symbol recognition by error-tolerant subgraph matching between region adjacency graphs, IEEE Transactions on Pattern Analysis and Machine Intelligence 23 (10) (2001) 1137–1143.

[35] E. Barbu, P. Héroux, S. Adam, E. Trupin, Using bags of symbols for automatic indexing of graphical document image databases, in: Graphics Recognition. Ten Years Review and Future Perspectives, Lecture Notes in Computer Science, 2005, pp. 195–205.

[36] H. Locteau, S. Adam, E. Trupin, J. Labiche, P. Héroux, Symbol spotting using full visibility graph representation, in: Proceedings of the seventh International Workshop on graphics Recognition, 2007, pp. 49–50.

[37] R. L. Qureshi, J.-Y. Ramel, D. Barret, H. Cardot, Spotting symbols in line drawing images using graph representations, in: Graphics Recognition. Recent Advances and New Opportunities, Lecture Notes in Computer Science, 2008, pp. 91–103.

[38] M. Teague, Image analysis via the general theory of moments, Journal of the Optical Society of America 70 (8) (1980) 920–930.

[39] M. Delalandre, E. Valveny, T. Pridmore, D. Karatzas, Generation of synthetic documents for performance evaluation of symbol recognition; spotting systems, International Journal on Document Analysis and Recognition 13 (2010) 187–207.

[40] B. T. Messmer, H. Bunke, A new algorithm for error-tolerant subgraph isomorphism detection, IEEE Transactions on Pattern Analysis and Machine Inteligence 20 (5) (1998) 493–504.

[41] S. Tabbone, L. Wendling, D. Zuwala, A hybrid approach to detect graphical symbols in documents, in: Document Analysis System VI, Vol. 3163 of Lecture Notes in Computer Science, 2004, pp. 342–353.

[42] S. Sorlin, C. Solnon, J.-M. Jolion, Applied Graph Theory in Computer Vision and Pattern Recognition, Vol. 52 of Studies in Computational Intelligence, Springer, 2007, Ch. A Generic Graph Distance Measure Based on Multivalent Matchings, pp. 151–182.