GEM++: a tool for solving substitution-tolerant subgraph isomorphism

Julien Lerouge[1], Pierre Le Bodic[2], Pierre Héroux[1], and Sébastien Adam[1]

[1]University of Rouen, LITIS EA 4108
BP 12 - 76801 Saint-Etienne du Rouvray, France
[2]H. Milton Stewart School of Industrial and Systems Engineering
Georgia Institute of Technology
765 Ferst Drive NW, Atlanta, GA 30332, United States
{Julien.Lerouge,Pierre.Heroux,Sebastien.Adam}@litislab.eu
lebodic@gatech.edu

Abstract. The substitution-tolerant subgraph isomorphism is a particular error-tolerant subgraph matching that allows label substitutions for both vertices and edges. Such a matching is often required in pattern recognition applications since graphs extracted from images are generally labeled with features vectors computed from raw data which are naturally subject to noise. This paper describes an extended version of a Binary Linear Program (BLP) for solving this class of graph matching problem. The paper also presents GEM++, a software framework that implements the BLP and that we have made available for the research community. GEM++ allows the processing of different sub-problems (induced isomorphism or not, directed graphs or not) with complex labelling of vertices and edges. We also present some datasets available for evaluating future contributions in this field.

Keywords: binary linear programming, subgraph isomorphism, graph matching toolkit, graph datasets

1 Introduction

Given a query and a target graph, the subgraph isomorphism problem consists in deciding whether there exists a subgraph of the target which is isomorphic to the query graph, i.e. there exists a one-to-one mapping between the vertices (respectively the edges) of the query and the vertices (resp. the edges) of a subgraph of the target. This theoretical problem has been the subject of many contributions in the literature [1, 2, 8, 11–13] since it finds applications in various fields such as biosciences, chemistry, knowledge management, social network analysis, scene analysis... In the context of structural pattern recognition for image analysis, algorithms targeting subgraph isomorphism are particularly useful since they can be used to simultaneously consider segmentation and recognition of objects of interest (the query graphs) in a whole image (the target graph).

Beyond its computational complexity (subgraph isomorphism is an NP-complete problem [7]), a main drawback of existing approaches for subgraph isomorphism

is the requirement of a strict matching between the query graph and a subgraph in the target graph. Now, in pattern recognition applications, graphs to be analyzed are usually affected by distorsions, that can result from the intrinsic variability of patterns in the image, from the digitization procedure, or from the graph construction steps. Among the possible distorsions, we are concerned in this paper with modifications of vertices/edges labels. Hence, we tackle a particular class of matching problem we call substitution-tolerant subgraph isomorphism. A subgraph isomorphism is said to be substitution-tolerant when editing operations on vertices and edges labels are allowed at a given cost. The graph topology remains unchanged.

In [9], we have considered the substitution-tolerant subgraph isomorphism as an optimization problem, modeled with a Binary Linear Program (BLP). A first formulation has been described and some tests have shown the efficiency of the approach for a real world application concerning symbol detection. This paper extends this work with three contributions. First, from a theoretical point of view, we present two extensions to the previous formulation. They concern (i) the handling of both directed or undirected graphs and (ii) the computation of induced subgraph matching. The second contribution is related to the implementation of the formulation in a framework called GEM++. The software implements the BLP with the ability to tackle many graph labelling functions (nominal, real, vectorial...) and to customize vertices and edges edit costs according to these labellings. GEM++ is available online for the research community[1]. Finally, we present some datasets dedicated to the evaluation of substitution-tolerant subgraph isomorphism approaches. These datasets are free to use for evaluating future contributions in this field.

The paper is organized as follows. Section 2 presents the extended formulation, in order to introduce the description of the software toolkit GEM++ which is given in section 3. Then, in section 4, we describe synthetic and real datasets dedicated to substitution-tolerant subgraph isomorphism. Finally, section 5 draws a conclusion of the paper and proposes future directions.

2 Proposed formulations

2.1 Definitions

Definition 1. A *graph* $\mathcal{G}$ is a couple $\mathcal{G} = (V_\mathcal{G}, E_\mathcal{G})$, where $V_\mathcal{G}$ is a set of vertices and $E_\mathcal{G} \subseteq V_\mathcal{G} \times V_\mathcal{G}$ is a set of edges. The graph $\mathcal{G}$ is said undirected if its edges have no orientation, i.e. $\forall (i,j) \in E_\mathcal{G}, (i,j) = (j,i) \in E_\mathcal{G}$. Otherwise, $\mathcal{G}$ is a directed graph.

Definition 2. Given $\mathcal{S} = (V_\mathcal{S}, E_\mathcal{S})$ and $\mathcal{G} = (V_\mathcal{G}, E_\mathcal{G})$ two graphs verifying $|V_S| \leq |V_G|$, an injective function $f : V_\mathcal{S} \to V_\mathcal{G}$ is a *subgraph isomorphism* from a graph $\mathcal{S}$ to a graph $\mathcal{G}$ if and only if f verifies $\forall (i,j) \in V_\mathcal{S} \times V_\mathcal{S}, (i,j) \in E_\mathcal{S} \Rightarrow (f(i), f(j)) \in E_\mathcal{G}$.

[1] http://litis-ilpiso.univ-rouen.fr/

2.2 Binary linear programming

Binary linear programming, also known as 0-1 linear programming, is a restriction of linear programming where the variables are binary. Hence, the general form of a BLP is as follows:

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

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

$$x \in \{0,1\}^n \tag{1c}$$

where $c \in \mathbb{R}^n, A \in \mathbb{R}^{n \times m}$ and $b \in \mathbb{R}^m$ are data of the problem. A solution of this optimization problem is a vector x of n binary variables. A is used to express linear inequality constraints (1b). The objective function $c^T x$ is a linear combination of all variables of x weighted by the components of the vector c. The optimal solution is the one that minimizes the objective function (1a) and respects constraints (1b) and (1c). Once equations (1a) to (1c) are correctly formulated, the second step consists in implementing this model using a mathematical solver. Given an instance of the problem, the solver explores the tree of solutions with the branch-and-cut algorithm, and finds the best feasible solution, in terms of the objective function optimization.

2.3 Solving the problem with a BLP

As shown in figure 1, the formulation of the substitution-tolerant subgraph isomorphism problem as a BLP implies the definition of two sets of variables:

- 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.

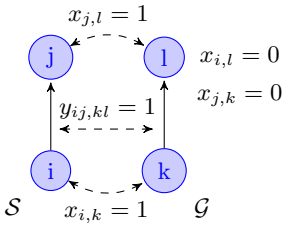

Fig. 1: An example of matching

Using such variables, the objective function, i.e. the global cost for matching $\mathcal{S}$ to a subgraph of $\mathcal{G}$, is defined as follows:

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

where $c_V : V_\mathcal{S} \times V_\mathcal{G} \to \mathbb{R}^+$ as well as $c_E : E_\mathcal{S} \times E_\mathcal{G} \to \mathbb{R}^+$ are functions which define the cost of associating vertices and edges.

The following constraints encode the substitution-tolerant subgraph isomorphism problem (see [9] for a justification of these constraints):

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

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

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

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

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

2.4 Extensions

Induced subgraph isomorphism An *induced subgraph isomorphism* is a more stringent problem, defined by :

Definition 3. Given $\mathcal{S} = (V_\mathcal{S}, E_\mathcal{S})$ and $\mathcal{G} = (V_\mathcal{G}, E_\mathcal{G})$ two graphs verifying $|V_S| \leq |V_G|$, an injective function $f : V_\mathcal{S} \to V_\mathcal{G}$ is an *induced subgraph isomorphism* from a graph $\mathcal{S}$ to a graph $\mathcal{G}$ if and only if $\forall (i,j) \in V_\mathcal{S} \times V_\mathcal{S}, (i,j) \in E_\mathcal{S} \Leftrightarrow (f(i), f(j)) \in E_\mathcal{G}$.

The equivalence in definition 3 requires an additional set of constraints in the BLP :

$$\sum_{i \in V_\mathcal{S}} x_{i,k} + \sum_{j \in V_\mathcal{S}} x_{j,l} - \sum_{ij \in E_\mathcal{S}} y_{ij,kl} \leq 1 \quad \forall kl \in E_\mathcal{G} \tag{2g}$$

Undirected graphs If $\mathcal{S}$ and $\mathcal{G}$ are undirected graphs, the set of constraints (2e) and (2f) are respectively modified into (2h) and (2i) and, if necessary, the set of constraints (2g) is modified into (2j) :

$$\sum_{kl \in E_\mathcal{G}} y_{ij,kl} \leq x_{i,k} + x_{j,k} \quad \forall k \in V_\mathcal{G}, \forall ij \in E_\mathcal{S} \tag{2h}$$

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

$$\sum_{i \in V_\mathcal{S}} (x_{i,k} + x_{i,l}) + \sum_{j \in V_\mathcal{S}} (x_{j,l} + x_{j,k}) - \sum_{ij \in E_\mathcal{S}} y_{ij,kl} \leq 1 \quad \forall kl \in E_\mathcal{G} \tag{2j}$$

2.5 Search for multiple solutions

Depending on the application context, it may be the case that the query graph that is searched for has many instances in the target graph. As defined in section 2, the BLP model is only able to find the optimal solution. There are multiple ways to search for many solutions [3]. 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. We implemented three different strategies to discard an optimal solution. After each iteration, given the optimal solution $(\bar{x}\ \bar{y})^T$, our algorithm may perform one of the following cut by adding the corresponding constraint to the BLP:

- cut exactly the optimal solution:

$$\sum_{i \in V_\mathcal{S}, k \in V_\mathcal{G}} \bar{x}_{i,k} \cdot x_{i,k} \leq |V_\mathcal{S}| - 1 \tag{3a}$$

- cut any solution containing a matching between vertices (or edges) of $\mathcal{S}$ and $\mathcal{G}$ present in the optimal solution:

$$\sum_{i \in V_\mathcal{S}, k \in V_\mathcal{G}} \bar{x}_{i,k} \cdot x_{i,k} = 0 \tag{3b}$$

- cut any solution involving a vertex (or an edge) of $\mathcal{G}$ already matched to a vertex (edge) of $\mathcal{S}$ in the optimal solution:

$$\sum_{i \in V_\mathcal{S}, k \in V_\mathcal{G}} \left(\sum_{j \in V_\mathcal{S}} \bar{x}_{j,k} \right) \cdot x_{i,k} = 0 \tag{3c}$$

The new constraint, that cuts the current optimal solution, is added to the model. Hence, this solution becomes infeasible for the next run. The solver can be called again and will be able to find, if it exists, another optimal solution.

3 GEM++

GEM++ is a Graph Extraction and Matching software that we developed in order to implement the subgraph matching approach described in section 2. It is written in C++ and is multiplatform. The framework is composed of shared software libraries, that gathers the common functionalities provided by our framework (graphs and weights handling, integer programming, call to a mathematical solver), and the command-line utility `GEM++sub`, that solves our subgraph isomorphism problem.

3.1 Graphs and substitution costs

GEM++ is able to import directed or undirected, labeled or unlabeled graphs, formatted either in Graph Modeling Language (GML) or in Graph eXchange Language (GXL). Figure 2 shows two examples of directed labeled graphs. The target graph is transcripted in GXL and GML formats in listings 1.1 and 1.2.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <gxl>
3  <graph id="query" edgemode="directed">
4  <node id="a"><attr name="x"><float>0.9</float></attr></node>
5  <node id="b"><attr name="x"><float>0.3</float></attr></node>
6  <edge from="a" to="b"><attr name="y"><float>0.4</float></attr></edge>
7  <edge from="b" to="a"><attr name="y"><float>0.2</float></attr></edge>
8  </graph>
9  </gxl>
```

Listing 1.1: GXL example (query.gxl)

```
graph [
  label "target"
  directed 1
  node [ id 0 x 0.5 ]
  node [ id 1 x 0.7 ]
  node [ id 2 x 0.4 ]
  edge [ source 0 target 1 y 0.6 ]
  edge [ source 1 target 2 y 0.1 ]
  edge [ source 2 target 1 y 0.8 ]
]
```

Listing 1.2: GML example (target.gml)

```
nodes_features_weights
x 1.0
edges_features_weights
y 2.0
```

Listing 1.3: Feature weights (weights.fw)

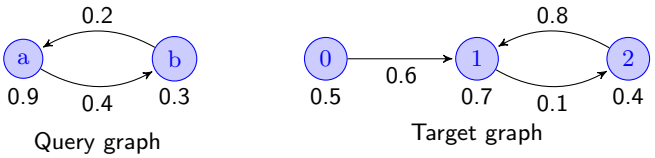

Fig. 2: An instance of the substitution-tolerant subgraph isomorphism problem

In order to tune the substitution costs c_V and c_E introduced in equation (2a), we defined a weighted euclidean distance on vertices and edges labels. Let us define $\mu : V_{\mathcal{S}} \cup V_{\mathcal{G}} \to L_V$ and $\xi : E_{\mathcal{S}} \cup E_{\mathcal{G}} \to L_E$ the functions that assign labels to each vertex and each edge of $\mathcal{S}$ and $\mathcal{G}$. The vertices and edges label spaces L_V and L_E may be composed of any combination of numeric (real) and symbolic attributes. As symbolic attributes are taken from discrete sets, for the sake of simplicity, we transform them into strictly positive integers. Therefore, we have for the vertices $L_V = \mathbb{R}^m \times (\mathbb{N}^*)^n$, with $m \geq 0$ and $n \geq 0$. Let us define $\mu_{\mathbb{R}} : V_{\mathcal{S}} \cup V_{\mathcal{G}} \to \mathbb{R}^m$ and $\mu_{\mathbb{N}^*} : V_{\mathcal{S}} \cup V_{\mathcal{G}} \to (\mathbb{N}^*)^n$, such that $\forall v \in V_{\mathcal{S}} \cup V_{\mathcal{G}}, \mu(v) = (\mu_{\mathbb{R}}(v), \mu_{\mathbb{N}^*}(v))$. Let $\mathbf{w} = (w_1, \ldots, w_m) \in \mathbb{R}^m$ and $\mathbf{z} = (z_1, \ldots, z_n) \in \{0; 1\}^n$ be two vectors of weights, the vertex substitution cost c_V is defined as follows:

$$c_V(i,j) = \begin{cases} \infty \text{ if } \exists k \in [\![1; n]\!], z_k = 1 \text{ and } \mu_{\mathbb{N}^*}(i) \neq \mu_{\mathbb{N}^*}(j), \\ \sqrt{\sum_{k=1}^{m} {w_k}^2 (\mu_{\mathbb{R}}(i)_k - \mu_{\mathbb{R}}(j)_k)^2} \text{ otherwise.} \end{cases}$$

The binary $\mathbf{z}$ weights are used to ignore some symbolic attributes. The same system is applied to the edge substitution costs. The listing 1.3 gives an example of a feature weights configuration file for vertices and edges.

3.2 Installation, solvers and usage

For Ubuntu or Debian-based Linux distributions, we provide pre-compiled packages of GEM++ in .deb format for x86 and x64 architectures. We also provide a Windows installer for 64 bits systems.

In order to solve the formulation, GEM++ relies on mathematical programming solvers. GEM++ is compatible with Gurobi[2] and IBM CPLEX[3], two widespread and performant commercial solvers. Gurobi and IBM provide free academic licenses for their solvers. GEM++ is also compatible with the GNU Linear Programming Toolkit[4] (GLPK), a free-software part of the GNU Project, available in the package repositories of Debian and Ubuntu.

The GEM++ package provides the `GEM++sub` command-line utility. Provided a couple of graphs and label substitution weights, `GEM++sub` is able to solve both substitution-tolerant and exact subgraph isomorphism problems. The available options for the command are summed up in the table 1.

Complete name	Short	Argument	Effect
`--weights`	`-w`	*file*.fw	Configures the weights for substitution costs
`--solution`	`-o`	*file*.sol	Outputs the matching solution(s)
`--induced`	`-i`		Performs induced subgraph isomorphism
`--number`	`-n`	*integer*	Searches up to n solutions
`--cut`	`-c`	*cut_strategy*	Switch to the given cut strategy
`--solver`	`-s`	*solver_name*	Switch to the given solver

Table 1: Command-line options offered by GEM++

3.3 Example

Let us run a solving example of the substitution-tolerant subgraph isomorphism problem, taking the query and target graphs showed in figure 1 as input of the problem. As the topology must be preserved while matching the query graph to a subgraph of the target graph, the only two feasible solutions are $s_1 = \left\{ \begin{array}{l|l} a \to 1 & ab \to 12 \\ b \to 2 & ba \to 21 \end{array} \right\}$ and $s_2 = \left\{ \begin{array}{l|l} a \to 2 & ab \to 21 \\ b \to 1 & ba \to 12 \end{array} \right\}$.

Let us use the weights defined in the listing 1.3, i.e. $w_x = 1$ and $w_y = 2$. Then, the overall costs of the solutions s_1 and s_2 are 2.1 and 1.9 respectively, and s_2 is the optimal solution. This solution can be discarded using strategy (3a) or (3b), in either case the algorithm will find s_1 as optimal solution at the second iteration. However, the use of strategy (3c) would forbid any new solution, since vertices 1 and 2 were already matched in s_2.

[2] http://www.gurobi.com
[3] http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/
[4] http://www.gnu.org/software/glpk

Here is the command that must be run on a terminal to retrieve the two solutions in a file named `output.sol` (see listing 1.4), using the default (3a) cut strategy :

```
$ GEM++sub -o output.sol -w weights.fw -n 2 query.gxl target.gml
```

```
Solution {
    objective value : 1.9
    nodes substitution {
        a 2 0.5
        b 1 0.4
    }
    edges substitution {
        a->b 2->1 0.8
        b->a 1->2 0.2
    }
}
```

```
Solution {
    objective value : 2.1
    nodes substitution {
        a 1 0.2
        b 2 0.1
    }
    edges substitution {
        a->b 1->2 0.6
        b->a 2->1 1.2
    }
}
```

Listing 1.4: Output example (output.sol)

If we had used different feature weights, e.g. $w_x = 1$ and $w_y = 1$, the overall costs of the solutions s_1 and s_2 would have been 1.2 and 1.4 respectively, and thus the first optimal solution would have been s_1 instead of s_2. Therefore, the weighting system allows the user to tune the substitution costs in order to match the application needs.

4 Datasets

Although some datasets have been proposed to evaluate exact subgraph isomorphism [4, 6], to the best of our knowledge there are no datasets designed to benchmark the search of substitution-tolerant subgraph isomorphism. In this section, we present two datasets designed for that purpose and made available for the community.

4.1 Synthetic datasets

The synthetic dataset is composed of four subparts. Each of these datasets contains pairs of graphs. One member of the pair is the query graph and the other is the target graph. For each graph pair, a groundtruth information gives the mapping between each vertex in the query graph to the corresponding vertex in the target graph.

`ILPIso_exact_synth` and `ILPIso_noisy_synth` datasets have been generated using the Erdös-Rényi model [5] which generates random graphs given n, the number of vertices, and p, the probability that two vertices u and v are linked with an edge (u, v). `ILPIso_exact_connected_synth` and `ILPIso_noisy_connected_synth` datasets have been generated with a model, also parameterized with n and p, with the same meaning, but the model used warrants that generated graphs are connected. The procedure used to generate graph pairs was the following:

1. The query graph in generated with a specific graph model generator.
2. A copy of the query is created.
3. In the noisy versions, vertex/edges labels are added a Gaussian noise.
4. The copy is completed to the desired size of the target graph.

Labels for vertices and edges are scalar values randomly drawn in $[-100, 100]$ according to a uniform distribution. For the noisy versions, labels are added a random number drawn from a Gaussian distribution ($m = 0, \sigma^2 = 5$).

Finally, each graph pair is parameterized by p, the probability of existence of an edge, n_p, n_t, which respectively denotes the number of vertices in the query graph and in the target graph. For `ILPIso_exact_synth` and `ILPIso_noisy_synth` datasets, parameters are such that $p \in \{0.01, 0.05, 0.1\}$, $n_p \in \{10, 25, 50\}$ and $n_t \in \{50, 100, 250, 500\}$. For `ILPIso_exact_connected_synth` and `ILPIso_noisy_connected_synth` datasets, the same values are used for n_p and n_t. p is chosen in $\{0.02, 0.05, 0.1\}$ but according to the value of n_p, the connectivity constraint prevents to select too low values of p.

4.2 Real datasets

This `ILPISO_real` dataset is composed of 16 query graphs and 200 target graphs. These graphs respectively describe structural representations of architectural symbols and architectural flooplans from the SESYD dataset[5] in which these symbols appear in their context. The extraction of this graph representation is detailed in [10]. Basically, a vertex is associated to each white connected component of the image of the plan or symbol, and an edge is created between vertices representing adjacent white regions. The features that are used to label vertices are mainly shape descriptors (24 first Zernike moments), whereas edges are labeled with feature vectors that characterize the relationship between adjacent regions (relative scale and distance).

Considering their structural representations, the problem of locating occurrences of symbols on a floorplan turns into the search of subgraphs of the target graphs that are isomorphic with query graphs representing symbols. Each symbol may occur once or several times on a floorplan, or may not occur at all. For each target graph representing a flooplan, the groundtruth information provides the identifiers of vertices that are involved in the symbol occurrence. The whole `ILPISO_real` dataset contains 5609 symbol occurrences with an average of 28 occurrences per target graph. The graphs corresponding to symbol instances contain 4 vertices and 7 edges on average, whereas the structural representations of the plans contain 121 vertices and 525 edges on average.

5 Conclusion

In this paper, we have presented GEM++, a software which implements a BLP for the search for substitution-tolerant subgraph isomorphism. This work is an

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

extension of [9], which now allows to handle undirected graphs, numeric and symbolic attributes, feature weighting, and induced sugraph isomorphism. The tool is available online, has been designed to be easily installed on several operating systems and may be customized at several levels. In particular, it can be used in conjuction with several mathematical solvers. We also provide synthetic and real graph datasets specifically designed for the problem of substitution-tolerant subgraph isomorphism. Our future works concern the tuning of the weights $\mathbf{w}$ and $\mathbf{z}$, the vectors which define the relative weights between attributes in the substitution cost. They could be determined by a learning phase aiming at optimizing the performance on a validation set. Moreover, this work could also be continued by integrating tolerance to topological modifications.

References

1. Cordella, L.P., Foggia, P., Sansone, C., Vento, M.: Performance evaluation of the VF graph matching algorithm. In: Proc. of the Int'l Conf. on Image Analys. and Proc. pp. 1172–1177 (1999)
2. Cordella, L.P., Foggia, P., Sansone, C., Vento, M.: A (sub)graph isomorphism algorithm for matching large graphs. IEEE Trans. on PAMI 26(10), 1367–1372 (2004)
3. Danna, E., Fenelon, M., Gu, Z., Wunderling, R.: Generating multiple solutions for mixed integer programming problems. In: Proc. of the 12th Int'l Conf. on Integ. Prog. and Combinat. Optim. pp. 280–294 (2007)
4. De Santo, M., Foggia, P., Sansone, C., Vento, M.: A large database of graphs and its use for benchmarking graph isomorphism algorithms. Pattern Recogn. Lett. 24(8), 1067–1079 (2003)
5. Erdös, P., Rényi, A.: On random graphs. Public. Mathemat. 6, 290–297 (1959)
6. Foggia, P., Sansone, C., Vento, M.: A database of graphs for isomorphism and subgraph isomorphism benchmarking. In: Proc. Third IAPR TC-15 Int'l Workshop Graph Based Representations. pp. 176–187 (2001)
7. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman & co. (1979)
8. Ghahraman, D.E., Wong, A.K.C., Au, T.: Graph optimal monomorphism algorithms. IEEE Transactions on System, Man and Cybernetics 10, 181–188 (1980)
9. Le Bodic, P., Héroux, P., Adam, S., Lecourtier, Y.: An integer linear program for substitution-tolerant subgraph isomorphism and its use for symbol spotting in technical drawings. Pattern Recognition 45(12), 4214–4224 (2012)
10. Le Bodic, P., Locteau, H., Adam, S., Héroux, P., Lecourtier, Y., Knippel, A.: Symbol detection using region adjacency graphs and integer linear programming. In: Proc. of the Int'l Conf. on Doc. Analys. and Recog. pp. 1320–1324 (2009)
11. Solnon, C.: Alldifferent-based filtering for subgraph isomorphism. Artificial Intelligence 174(12-13), 850 – 864 (2010)
12. Ullmann, J.R.: An algorithm for subgraph isomorphism. J. ACM 23(1), 31–42 (1976)
13. Wong, A.K.C., You, M., Chan, S.C.: An algorithm for graph optimal monomorphism. IEEE Transactions on System, Man and Cybernetics 20(3), 628–638 (1990)