Isomorphism Testing via Polynomial-Time Graph Extensions

Daniel Cosmin Porumbel[*]

October 24, 2010

Abstract

This paper deals with algorithms for detecting graph isomorphism (GI) properties. The GI literature consists of numerous research directions, from highly theoretical studies (e.g. defining the GI complexity class) to very practical applications (pattern recognition, image processing). We first present the context of our work and provide a brief overview of various algorithms developed in such disparate contexts. Compared to well-known NP-complete problems, GI is only rarely tackled with general-purpose combinatorial optimization techniques; however, classical search algorithms are commonly applied to graph matching (GM). We show that, by specifically focusing on exploiting isomorphism properties, classical GM heuristics can become very useful for GI. We introduce a polynomial graph extension procedure that provides a graph coloring (labeling) capable of rapidly guiding a simple-but-effective heuristic toward the solution. The resulting algorithm (GI-Ext) is quite simple, very fast and practical: it solves GI within a time in the region of $O(|V|^3)$ for numerous graph classes, including difficult (dense and regular) graphs with up to 20.000 vertices and 200.000.000 edges. GI-Ext can compete with recent state-of-the-art GI algorithms based on well-established GI techniques (e.g. canonical labeling) refined over the last three decades. In addition, GI-Ext also solves certain GM problems, e.g. it detects important isomorphic structures induced in non-isomorphic graphs.

1 Introduction

Graph encodings can describe a large variety of connected objects in virtually all fields of science. The graph isomorphism (GI) problem and its generalizations are essential in large application areas directly dealing with similarity problems, e.g. pattern and image recognition [1,8,9,20,28,33,35,38,39] or chemical informatics [23,31]. Furthermore, algorithms for these problems often find unexpected applications in other more specific applications, from biometric identification or scheduling to monitoring computer networks—numerous such examples are available in the literature [8,10].

In theoretical computer science, GI is one of the few NP problems not proved to be either in P or $NP - P$ (we assume $P \neq NP$) and a lot of effort has been done to classify

[*]Univ. Lille–Nord de France, UArtois, LGI2A, Technoparc Futura 62400 Béthune, France (e-mail: daniel.porumbel@univ-artois.fr)

it. Although polynomial time algorithms are available for many graph classes [3, 17, 27], all existing algorithms are still exponential for certain well-known graphs classes—e.g. regular graph isomorphism is GI-complete [6, 40], that is, if regular graphs can be tested for isomorphism in polynomial time, then so can be any graphs. The exact GI algorithms are usually based on tree search procedures with various pruning techniques [12,13,34,36], or on comparing canonical graph forms invariant under isomorphism [2, 4, 14, 15, 22, 29].

Numerous *heuristic* GI algorithms are also available, but they have been developed rather for graph matching (GM) than for GI. Indeed, heuristic optimization algorithms are well adapted to practical matching problems that require finding a best (but not necessarily perfect) isomorphism. In this context, one often defines an *objective function of similarity*—a measure of closeness—and the best matching is searched using (meta) heuristic methods. However, these heuristics are often developed for specific applications and they are targeted to small and very small graphs (i.e. hundreds or tens of vertices, see Section 4).

In order to put our work in context, we first present a brief overview of the most related (exact or heuristic) approaches from the literature of GI and GM. Even if this literature is very vast (hundreds of algorithms are available [10]), small order graphs are commonly employed in practice (hundreds, maximum thousands of vertices). Next, we introduce GI-Ext, a combinatorial optimization algorithm that appears to identify isomorphisms very rapidly: it runs in low-order polynomial time on numerous standard graph classes. We also show certain polynomial complexity bounds for some cases. Numerical experiments are carried out for dense regular graphs with up to 20000 vertices, graphs from an existing GI-testing database, complete benchmark sets from other papers, etc.

GI-Ext has three stages: (i) polynomial *graph extension* to propagate new isomorphism constraints from the initial edge structures, (ii) linear deterministic construction of an initial potential solution, and, if necessary, (iii) local search in the proximity of this point. To our knowledge, an advantage of GI-Ext is the simplicity: no deep discrete mathematics is needed for the first two stages and the (meta-)heuristic from the last step is related to methods used in GM combinatorial optimization algorithms. We show evidence that the new approach competes well with state-of-the-art algorithms on numerous graph classes. While certain crafted graphs might be approached more effectively only using very recent canonical labeling techniques [22,26], these methods employ more complex mathematics based on refining work which began 3 decades ago [29]. Furthermore, GI-Ext is also very useful to detect large isomorphic sub-structures induced in non-isomorphic graphs—e.g. we also provide experiments on a GM problem.

The next section briefly outlines the problem notations and formulations. Section 3 reviews certain exact and inexact methods from the large GI literature, selecting the most relevant ideas in the context of our work. Next, in Section 4 we present several objective functions commonly employed by GM heuristic algorithms and we also introduce our new function. Section 5 presents the GI–Ext algorithm and the polynomial graph extension that supports its performance. The last section shows numerical experiments on several graph families, followed by conclusions.

2 Graph Isomorphism (GI) and Graph Matching (GM) Definitions

We denote the input graphs by $G(V, E)$ and $G'(V', E')$ and their adjacency matrices by M and M', respectively. The number of vertices (denoted by $|G|$ or $|V|$) is commonly referred to as the graph order. For graph isomorphism, we can consider without loss of generality that $V = V' = \{1, 2, \ldots, |V|\}$. A mapping between G and G' is represented by a bijective function on the vertex set $h : V \to V'$. We say that h^* is an isomorphism if and only if $\{i, j\} \in E \Leftrightarrow \{h^*(i), h^*(j)\} \in E'$; the *graph isomorphism (GI) problem* consists of determining whether or not such an isomorphism exists.

For an optimization algorithm, the set of all bijections h constitute the search space S. An *objective function* f (see examples in Eq. (2), Section 4) is employed to measure the quality of each mapping h. The optimization GI problem requires finding the minimum of f, e.g. a mapping h^* such that $f(h^*) = 0$ if h^* is an isomorphism. In this context, the elements of S can also be referred to as search space *potential solutions* (or configurations).

The mapping $h : V \to V'$ can be represented as a permutation of V when $V = V'$, i.e. it is encoded as an array $[h(1),\ h(2), \ldots\ h(|V|)]$ with all values pairwise different. The value at position i represents the vertex association $i \overset{h}{\to} j$ and $(i, j) \overset{h}{\to} (i', j')$ is a double (edge) association. A double association $(i, j) \overset{h}{\to} (i', j')$ is a *conflict* if h violates the isomorphism constraint on (i, j), i.e. if either (i) $\{i, j\} \in E$ and $\{i', j'\} \notin E'$, or (ii) $\{i, j\} \notin E$ and $\{i', j'\} \in E'$. Throughout the paper, we prefer to express the same conditions simply as:

$$(i, j) \overset{h}{\to} (i', j') \text{ is a } \textit{conflict} \Longleftrightarrow M_{ij} \neq M_{i'j'}$$

Finally, a mapping h can also be encoded as a *match matrix* P_h, such that:

$$P_{h_{ii'}} = [h(i) = i'], \tag{1}$$

where the square brackets operator is 1 if the predicate inside brackets is true and 0 otherwise.

Graph matching While GI is a decision problem, graph matching (GM) usually consists of finding the best matching between two graphs, i.e. the best mapping h. An optimization algorithm can solve a GM problem in the same manner as a GI problem—the only difference is that there might be no mapping h^* such that $f(h^*) = 0$ (no isomorphism). One should be aware that there are several *distinct* problems that are all referred to as GM, depending on the interpretation of "best matching". Thus, we only touch on the most GI-related research in this paper and we consider GM formulations defining an objective function f. GM problems can also be stated using algebraic analysis (using eigen decomposition, spectral graph theory), or using probabilistic methods (e.g. determining the likelihood that a data graph was obtained from a model graph through a noisy process)—a classification can be found in [28, §1.1].

GM provides a more flexible approach to practical applications in which perfect graph isomorphism is quite rare—e.g. two images of the same person are never identical, but a graph matching algorithm can find them very similar. Depending on the context, GM algorithms can also be referred to as error-correcting (or error-tolerant) graph isomorphism, inexact (or partial) isomorphism, matching with mismatches, etc. Graph matching problems often use edge-weighted, vertex-labeled or attributed relational graphs (i.e. where both the edges and the vertices posses weighted values). To deal with such graphs, labels can be encoded in the diagonal elements of M and weights on the other elements.

3 A Brief Historical Algorithmic Overview

There are numerous research threads in the literature of both graph isomorphism (GI) and graph matching (GM). While GI is more important from a theoretical viewpoint, GM algorithms are more common in relatively-young practical applications dealing with various types of similarity problems. Although there are several distinct problems referred to as GM, most of them can be considered direct generalizations of GI and they all have higher computational complexities. As such, *exact* GM algorithms are limited to very small graphs (i.e. several tens of vertices [8, 37]) and the GM literature is dominated by inexact algorithms. Most heuristic algorithms available for GI were actually developed for specific applications involving GM problems.

3.1 Exact Algorithms for Graph Isomorphism

As for the exact GI methods, the algorithm of Ulmann [36] is historically the most famous. The main idea of the approach is to recursively (try to) construct the match matrix P_{h^*} of isomorphism h^* using a backtracking algorithm; a refining procedure prunes the search tree using a look-ahead technique that anticipates isomorphism violations. This type of tree-search methods were first developed in the early 1970s [13] and, since then, numerous pruning techniques have been proposed to reduce the search space. Another example of such a backtracking strategy is the algorithm of Schmidt and Druffel [34] that reduces the search tree using an initial partition of the vertex sets. More recent algorithms are also available, for instance the VF2 algorithm is very effective and much faster than the algorithm of Ullmann [12].

A drawback of tree search methods is that they can waste important time exploring potential solutions (mappings) equivalent up to an automorphism (the search space can have many symmetries). As such, a second approach for exact GI resides in constructing canonical graph labelings (invariant under automorphism) that can be more easily compared. In this context, the oldest and most established algorithm is *nauty* [29], in which a canonical graph is associated to any input graph—i.e. such that all isomorphic graphs have the same canonical graph. Constructing the canonical graph is based on computing the automorphisms of the input graph. The automorphism group is built with a partition refinement procedure that backtracks through a tree of colorings (or ordered partitions) that correspond to potential automorphisms. In fact, automorphisms are discovered at the leaf nodes of the tree—a leaf coloring encodes a bijective relation between $|V|$ colors and $|V|$ vertices, corresponding to a re-labeling of V. A rigorous description of this process requires complex mathematics, but, fortunately, there are high-quality descriptions available [14, 15, 22].

Canonical labeling algorithms represent one of the most effective approach for pure GI problems and important progress has been made in the last five years. Indeed, there are at least two recent tools—*bliss* [22] and *saucy* [14,15]—showing impressive results in discovering automorphisms for large and sparse graphs. Furthermore, canonical labeling techniques can be combined with three search methods as in another very recent algorithm *conauto* [26] that uses precomputed graph automorphisms to prune the search tree. Another popular strategy often employed in GI is to consider graph vertex invariants; indeed, many examples of invariants are available in the literature [18] and they are also essential for canonical labeling.

In a practical context, it is worth mentioning that the VFLib Graph Matching Library provides implementations for VF2, Ulmann and Schmidt-Druffel, and that the LEDA Library (Library of Efficient Data types and Algorithms) for VF2 and *conauto*. However, *bliss* has been also implemented in a public library and it seems very fast in finding canonical labellings for a large variety of graphs with different types of regularities. Furthermore, *saucy* was used to find the automorphism group for very sparse graphs using techniques adapted to this situation.

One should be aware there are many particular graph families that can be tested for isomorphism in polynomial time: graphs of bounded degree (valence) [27], graph of bounded genus (including planar graphs or trees) [17], or graphs with adjacency matrices with bounded eigenvalue multiplicity [3]. For the general case, the best theoretically-proved worst-case running time is $O(e^{\sqrt{|V|log(|V|)}})$ [4] but this algorithm is not typically used in practice. However, it is quite difficult to make absolute comparisons of practical algorithms because they all have worst-case exponential running times and their performance can vary from one graph class to another. External factors should also be taken into consideration, e.g. different machines, different programming languages, different implementation styles, etc.

3.2 Inexact Algorithms for Graph Isomorphism

Regarding the incomplete GI algorithms, a lot of research work is actually available in the vast graph matching (GM) literature; many GM heuristic algorithms can be directly applied to GI. We focus here on heuristic combinatorial optimization algorithms that (try to) minimize certain measures of closeness between graphs, i.e. an objective function f that reaches the minimum value $f(h^*) = 0$ only for isomorphism mappings $h^* : G \to G'$. Finding the minimum value of such function f can also be approached with other methods besides heuristics—tree-search, algebraic analysis, algorithms based on probability theory, see more classifications in [8,28,37]—but they lie outside the scope of this article.

The first closeness measures were proposed in the early 70's [25,33] and they are based on the maximum common subgraph and on generalizations of the string editing (Levenstein) distance. From a practical perspective, these measures are usually computationally very demanding (e.g. the maximum common subgraph problem is NP-complete [19]) and they are typically employed only in applications using small graphs—i.e. such NP-hard algorithms were used for molecular similarity [31], but not for applications with larger graphs (i.e. image recognition). The editing distance is also an NP-Hard problem [41], but, since many variants exist, some graph editing distance algorithms are available to more applications [7,35].

Since the introduction of these first theoretical indicators, the GM problems gradually gained in importance and various new closeness measures were proposed, especially for pattern and image recognition [1, 5, 9, 37] or chemistry informatics [23]. They usually present ad-hoc measures, using application-specific features that might be difficult to express in a unified GI context. However, most GM encodings are generalized GI encodings and the heuristics can be used in exactly the same way to solve either GM or GI.

The optimization algorithms can be broadly classified into continuous methods (i.e. using various continuous embeddings [20, 28]) or discrete heuristic algorithms (i.e. Tabu search [38], genetic algorithms [9, 35, 37], estimation of distribution algorithms [5], ant algorithms [32], etc.). More specific algorithms (i.e. using decision trees, neural networks) are also available in the literature [1, 20, 28, 38] but they are less related to our paper.

4 Objective functions

Since most GM heuristics are developed for specific applications, they are typically tested and compared with respect to the success of finding good solutions for matching tasks *inside* these applications. GM optimization algorithms often work with distinct graph representations and with different semantics—for example, the edge weights can signify geometric distances, temporal causality, concept similarity, etc.

However, a common point is the construction of an objective function representing the measure of closeness that is minimized over the space S of all bijections $h : G \to G'$. We provide examples of such functions, as used by various papers in different contexts:

$$\begin{aligned}
f_1(h) &= \frac{\alpha}{|V|}\sum_{v\in V} c_V(v,h(v)) + \frac{1-\alpha}{|E|}\sum_{e\in E} c_E(e,h(e)) && [9, \S 2.2]\\
f_2(h) &= ||M - P_h M' P_h^T|| && [37, \S 2.2]\\
f_3(h) &= -\frac{1}{2}\sum_{i\in V}\sum_{j\in V}\sum_{i'\in V}\sum_{j'\in V} \hat{P}^h_{ii'}\hat{P}^h_{jj'}C_{ii'jj'} && [20, \S 2.1] \quad (2)\\
f_4(h) &= |V| + |V'| - 2\max\{|G_c| \ : \ G_c \sqsubseteq h(G) \text{ and } G_c \sqsubseteq G'\} && [7, 25]\\
f_5(h) &= \min\{\text{Cost}(E_{\text{dt}}) \ : \ E_{\text{dt}} = \textit{edit} \text{ operations s. t. } h(G) \xrightarrow{E_{\text{dt}}} G'\} && [7, 30, 33]
\end{aligned}$$

The first function (f_1) evaluates attributed graphs using a measure of vertex similarity (C_V) and a measure of edge similarity (c_E); their global contributions are weighted using the α factor. It was utilized with three algorithms in order to carry out a comparative study of optimization methods for image recognition [9]; the order of considered graphs is in the range 147–228 vertices. Related strategies, especially Estimation Distribution Algorithms are presented in greater detail in [5].

The f_2 function uses the match matrix P_h (see Section 2, (1)) and a matrix norm $||\cdot||$. This function was tested with several algorithms and with various versions of the matrix norm, but they always use graphs with 10–20 vertices [37]. The matrix $\hat{P^h}$ for f_3 is a relaxed version of P_h, i.e. its elements are real numbers in $[0, 1]$ and, for any $i \in V$, $\sum_{i'\in V'} \hat{P}^h_{ii'} = 1$; $C_{ii'jj'}$ is a similarity measure between the vertex pairs (i, j) in $V \times V$ and (i', j') in $V \times V$. The experiments are performed on graphs from images or randomly-generated with 100 vertices at maximum [20]. The same function is used in a series of papers about molecular structures [23] with Simulated Annealing (SA) and Tabu search, but small graphs are usually used.

Function f_4 is a reformulation of a measure based on the order of the maximum common subgraph. In this equation, the $\sqsubseteq$ relation denotes subgraph isomorphism; if h^* is an isomorphism, the inclusion becomes equality (i.e. $G_c = h^*(G) = G'$) and $f(h^*) = 0$. Many other reformulations can be found in the literature [8, 25, 31] and it was shown [7] that this measure is equivalent to a graph edit distance.

Graph editing distance (function f_5) represent a large family of measures and several heuristic algorithms were developed for them [35]. Typical algorithms using such functions are often limited for graphs of small order, e.g. maximum 100 vertices [30, 35].

There exist several other application-specific functions (e.g. using angles [1] or using various specific feature-based graph descriptions [39]), but the goal of the paper is not to enter into complex formulas. However, a common point of all these objective functions is that they require algorithms with high computational costs that are usually applied on small graphs ($|V| < 1000$) as GM problems are computationally very demanding.

4.1 Proposed GI approach and objective function

The simplest formulation of our measure of closeness between graphs is:

$$f(h) = \sum_{1 \le i < j \le |V|} [M_{i,j} \neq M'_{h(i),h(j)}], \tag{3}$$

where the square brackets represent an indicator that is 0 if the predicate (inside the brackets) is false and 1 otherwise—if $M_{i,j} \neq M'_{h(i),h(j)}$, we say that the association $(i, j) \xrightarrow{h} (i', j')$ is a conflict. The objective of our proposed heurstics is to search for a matching with the lowest number of conflicts.

This function can be seen as a particular case of f_1 and f_3 if we consider a specific interpretation of the cost functions used by these functions. As such, it could be directly compared with optimization algorithms using these function; however, these algorithms were tested on smaller non-isomorphic graphs, making such a comparison very difficult—GI-Ext is focused on finding isomorphisms, or, at least, large similarities between graphs.

Recall that our algorithm encodes the function h as a permutation of V (see Section 2). The neighborhood function (used by local search to move from one mapping to another) consists of swapping two elements (i, j) of this permutation, i.e. $h(i)$ takes the value of $h(j)$ and $h(j)$ takes the value of $h(i)$. We write $h := h+ < i, j >$ to denote that a move in the neighborhood is performed. This interchange neighborhood has the advantage that it does not require re-computing the whole sum of the objective function after each move, it can only compute the variation induced in the sum by the elements i and j.

Using this standard configuration encoding and neighborhood, we carried out certain preliminary tests with several (meta-) heuristics algorithms. Simulated annealing [21, 24] seemed to be the most promising: with a good choice of the annealing schedule, it can slowly but surely converge from a random bijection to the isomorphism. However, for large graphs with many regularities, the algorithm could easily get blocked into local optima; to avoid this, we use Iterated Local Search principles so as "iterate" the cooling schedule (see also Section 5.2.3). Furthermore, in order to deal with larger graphs, we propose to reinforce the algorithm by propagating new isomorphism constraints from adjacency matrix M, as described in the next section.

5 A GI algorithm based on graph extensions

The first part of this section (§5.1) only presents the graph extension procedure. Then, the second part (§5.2) provides the complete description of GI-Ext, discussing the importance of applying the graph extension. The motivation for the extended graphs arise from the following situation: if there is no edge between vertex i and j in G (i.e. $M_{i,j} = 0$) and no edge between $h(i)$ and $h(j)$ in G', then the association $(i,j) \xrightarrow{h} (h(i), h(j))$ is not seen as a conflict—there is no mechanism to directly detect whether (i,j) and $(h(i), h(j))$ are indeed compatible or not. But, by exploiting the structure of the graph, one can find many conditions in which (i,j) and $(h(i), h(j))$ are incompatible even if they are both disconnected (e.g. by checking the shortest path between them).

5.1 Polynomial graph extension

5.1.1 Extension construction

We define the $|V| \times |V|$ matrix M^K, in which the element $M^K_{i,j}$ represents the number of paths of length K (i.e. with K edges) from i to j. Obviously $M^1 = M$, and we now show that M^{K+1} can be computed in polynomial time from M and M^K using the following algorithm based on dynamic programming.

Algorithm 1 Polynomial time graph extension step

```
Input: $M^K$
Result: $M^{K+1}$
 1. Set all elements of $M^{K+1}$ to 0
 2. For each $\{i,j\} \in E$ with $i < j$
      For $k = 1$ to $|V|$
         • $M^{K+1}[i,k] = M^{K+1}[i,k] + M^K[j,k]$
         • $M^{K+1}[j,k] = M^{K+1}[j,k] + M^K[i,k]$
         • $M^{K+1}[k,j] = M^{K+1}[j,k]$ and $M^{K+1}[k,i] = M^{K+1}[i,k]$
```

The idea behind this algorithm is to notice that all the paths of length $K+1$ between two vertices i and k are composed of an edge $\{i,j\}$ and a path of length K from j to k. Since the number of paths of length K from j to k is known, we only need to sum up the number of paths of length K from all neighbors of i to k. To achieve this, the algorithm lists all edges $\{i,j\} \in E$ and adds all contributions of $\{i,j\}$ to paths $i \to j \overset{K \text{ steps}}{- \cdots \to} k$ in the constructed M^{K+1} matrix.

The extended graph is straightforwardly defined as the weighted graph with vertex set V and edge set E^K defined as follows: if $M^K_{ij} \neq 0$, then $\{i,j\}$ is an edge of weight M^K_{ij} in E^K. If two graphs G and G' are isomorphic, then their extended graphs are isomorphic as well—because the same extension operations are applied in the same manner for any two isomorphic vertices i and $h^*(i)$.

5.1.2 Complete extended graphs

Once the matrices $M^1, M^2, \dots M^K$ are computed, one can construct a *complete extended matrix* $\overline{M}$ such that

$$\overline{M} = a^1 M^1 + a^2 M^2 + \cdots + a^K M^K, \tag{4}$$

where a is a positive number. If $a > \max M^K$, then a ensures in (4) that two different series of M_i's can not result in identical complete extended matrices.

The complete extended graph of order K is the weighted graph $\overline{G}$ with vertex set V and with edges $\{i, j, \overline{M}_{ij}\}$. Two graphs are isomorphic if and only their complete extended graphs are isomorphic. Indeed, if the initial graphs are isomorphic, then so are the extended graphs, and so are the complete extended graphs—any isomorphism relation is conserved by the sum (4). In the other direction, if the complete extended graphs are isomorphic, then $\overline{M} = \overline{M'}_{h^*(i),h^*(j)}$ implies that $M^1_{i,j} = M'^1_{h^*(i),h^*(j)}$—in (4), one can see a as the base of a numeral system. The sum (4) can be computed in polynomial time for any $a \in O(|V|)$ with the appropriate data structure. In practice, we execute our optimization algorithm on complete extended graphs with $a = 1$; eventually, any resulting extended graph isomorphism is tested for consistency on the initial graphs.

To search the isomorphism of complete extended graphs, we generalize our objective function (see Equation 3) for the complete extended matrix $\overline{M}$:

$$\bar{f}(h) = \sum_{1 \le i < j \le |V|} [\overline{M}_{i,j} \ne \overline{M'}_{h(i),h(j)}] \tag{5}$$

If there exists an isomorphism $h^* : G \to G'$, then $\overline{M}_{i,j} = \overline{M'}_{h^*(i),h^*(j)}, \forall i, j \in V$ and $\bar{f}(h^*)$ is 0; the isomorphism h^* is the same for both the initial graphs and the extended graphs. An optimization algorithm based on the classical objective function (see Equation 3) can be used in the same way to minimize the new objective function.

5.1.3 Incompatibility detection using extended edges

If only exact isomorphism is targeted, the matrices $M^1, M^2, \dots M^K$ can also be used to discover incompatible (forbidden) associations: vertices $(i, i') \in V \times V'$ that can never be mapped by a graph isomorphism function. These forbidden associations might recall of the way other algorithms use ordered partitions (colors) for discovering automorphisms; however, we do not only partition the vertex set, but these forbidden associations can discover impossible isomorphic mappings between vertices of different graphs.

Definition 1 *(Compatible and forbidden associations) Vertices $(i, i') \in V \times V'$ are* compatible *if and only if: (i) $\overline{M}_{i,i} = \overline{M'}_{i',i'}$ and (ii) line i of $\overline{M}$ is equal to a permuted line i' of $\overline{M'}$. An association $i \to i'$ that is not compatible, is also called* forbidden.

Indeed, if h^* is an isomorphism, then all associations $(i \xrightarrow{h^*} i')$ are compatible because each element M^K_{ij} of line i of M^K, can also be found in M^K at line i' and column $j' = h^*(j)$. Therefore, any GI algorithm presented later in Section 5.2 avoids associating two incompatible (*forbidden*) vertices. We introduce a matrix F encoding these forbidden associations, such that if $F_{i,i'} = 1$, the association $i \to i'$ if forbidden (i.e. the local search can never assign i' to i). This matrix of forbidden associations is empty at start (all elements are 0), and the extension routine gradually fills its elements while constructing the matrices $M^1, M^2, \ldots M^K$.

The matrices M^1, M^2, M^3, ... are very rich in information that is implicitly checked via the matrix F. Each edge value from the extended graph is in fact a a hash function of some larger structures around it in the initial graph. Indeed, the fact that an association $i \to i'$ is not forbidden (i.e. $F_{ii'} = 0$) implies numerous hidden conditions: i and i' need to have the same degree (otherwise $M^2_{i,i} \neq M'^2_{i',i'}$), they need to be part in the same number of triangles (otherwise, $M^3_{i,i} \neq M'^3_{i',i'}$), they need to have the same number of 2-step neighbors, etc. Many other such theoretical conditions can be derived and proved, but the goal of this specific paper is only to present a very practical, high-speed algorithm; many more such conditions can be found by a theoretical study of greater detail.

5.2 GI-Ext—Algorithm Applying the Extension Procedure

The GI-Ext algorithm (see Algorithm 2) consists of three stages: (1) graph extension, (2) construction of an initial configuration (initial mapping), and (3) a heuristic search for finding the optimal solution in the proximity of this initial configuration. The first stage builds the information-rich extended adjacency matrix $\overline{M}$ and it also provides a matrix F of forbidden vertex associations (used only for pure isomorphism). The second stage tries to use this information for a linear construction of an initial good mapping. The last stage is a heuristic that searches in the proximity of the initial mapping so as to minimize $\bar{f}$ (Equation 5). This heuristic easily reduces the number of conflicts because it implicitly exploits a large amount of structural information hashed in the matrices M^1, M^2, M^3, ..., M^K, and implicitly in $\overline{M}$. Recall that a double association $(i,j) \xrightarrow{h} (i',j')$ is non conflicting for $\bar{f}$ if and only if the number of paths of length 1, 2, ..., K between i and j (in G) are the same as for i' and j' (in G')—which is equivalent to important similarities on the structures around the edges $\{i,j\}$ and $\{i',j'\}$

Algorithm 2 GI-EXT Algorithm

```
Input:  Graphs $G$ and $G'$
Result:  True, False, or Undetermined (probably not isomorphic)
  1. Repeat
     – Graph extension step                              /*see Alg. 1*/
     – $K := K + 1$                                  /*initially, $K = 1$*/
     Until a stopping condition is reached        /*see Sec. 5.2.1*/
   - Construct matrix $F$ of forbidden associations /*see Sec. 5.1.3*/
   - If incompatibility found Then Return False
  2. Construct Initial Potential Solution         /*see Sec. 5.2.2*/
   - If isomorphism found Then Return True
  3. Run Heuristic Search with stopping conditions:   /*Sec. 5.2.3*/
      If isomorphism found Then Return True
      If time elapsed Then Return Undetermined
```

We provide all details of the above-presented steps in the next three sections (Section 5.2.1, 5.2.2 and 5.2.3). Notice that the most important and GI-related steps are Step 1 and Step 2; the heuristic search could be replaced by other method, or even by an integer programming approach.

5.2.1 Extension application and complexity: GI-Ext Step 1

Even if the graph extension procedure is polynomial, it represents by far the largest computational cost. Recall that a step of this procedure (i.e. if we apply it once, so as to construct M^{K+1} from M^K) requires $O(|E| \cdot |V|)$ time. If we perform K extension steps, the total complexity is $O(K|E| \cdot |V|)$. Thus, it is important to obtain a trade-off between the number of steps (i.e. K) and the number of new derived constraints via the extended matrix. Each higher value of K adds more constraints in the matrix F (making the search space smaller) and it also adds new extended edges (the objective function becomes more informative). In the beginning, the process starts with $|E^1| = |E|$, but the number of extended edges in E^K increases with each new higher value of K. Following empirical evidences, we chose to stop this extension procedure as soon as $|E^K| \geq 66\%|V|^2$ or if the resulting E^K values become heterogenous enough,e.g. if all non-zero values of M^K are pairwise distinct.

This strategy for setting K assures that, in practice, the final complexity of the whole extension operation is in the region of $O(|V|^3)$. However, the complexity of the procedure (i.e. $O(K|E| \cdot |V|)$) is *theoretically limited by* $O(|E| \cdot |V|^3)$ because K can never reach $|V|^2$. If each extension step discovers at least one new extended edge, after at most $K = 66\%|V|^2$ steps the procedure stops (because E^K has surely at least 66% elements). If there is extension step such that $|E^K| = |E^{K+1}|$, this means that all $i, j \in V$ that can be connected through $K + 1$ edges (of E), can also be connected using a smaller number of edges. In this case, there already exists an extended edge (in E^K) between any two connected vertices of V (given any two vertices, they are either disconnected or can be connected through K edges or less). This implies that either $|E^K| = |V|^2$, or $|E^K| = |E^{K+1}| < |V|^2$ and the graph is disconnected (in this special case, the problem can be split).

5.2.2 Linear construction of initial start configuration: GI-Ext Step 2

This step incrementally constructs an initial mapping h using a linear procedure that tries to associate all $i \in V$ only to compatible values $h(i) = i' \in V'$ (such that $F_{ii'} = 0$). All vertices $i \in V$ are first sorted according to their number of compatible vertex associations (i, i')—i.e. the number of zeros in row i of F. The complete mapping is constructed by assigning all $i \in V$ in this order,i.e. starting with those with the lowest number of compatible associations (leaving less $i' \in V$ choices available). For each vertex $i \in V$, we associate a vertex i' from V', such that the new association $i \xrightarrow{h} i$ is as consistent as possible with the previously associated vertices (i.e. introduces the minimal number of conflicts). The result of this process is an initial configuration possibly with inconsistent (conflicting) edge associations that can be minimized (reduced) using local search.

5.2.3 Local search: GI-Ext Step 3

The solution encoding and the neighborhood are defined in Section 4.1 and we consider the *extended* objective function $\bar{f}$ (see Equation (5)). The search algorithm starts from configuration h constructed in Step 2.

Repairing Descent Procedure Let us first describe a simple Descent procedure. At each iteration, one searches for a conflict (i,j) of h (i.e. a pair (i,j) such that $\overline{M}_{i,j} \neq \overline{M'}_{h(i),h(j)}$) that needs to be repaired. To effectively "repair" a conflicting $(i,j) \xrightarrow{h} (h(i),h(j))$, the descent procedure searches for a vertex $k \in V$ such that $\overline{M}_{i,j} = \overline{M'}_{h(i),h(k)}$ and $\overline{M}_{i,k} = \overline{M'}_{h(i),h(j)}$. The (i,j) conflict can be solved by swapping the values of $h(j)$ and $h(k)$—we say that a move $< j,k >$ is carried out and the mapping h becomes $h+ < j,k >$. However, such a move can theoretically generate other conflicts in the graph and the procedure also checks that the Δ value of the move—i.e. $\Delta(j,k) = \bar{f}(h) - \bar{f}(h+ < j,k >)$; the move is performed only if the Δ value is positive.

In case there is no conflict that can be repaired by such a move, the descent procedure tries all possible moves (i.e. swaps (i,j)). If it finds any positive $\Delta(i,j)$, the move $h = h+ < i,j >$ is executed and the process goes on to the next iteration. In this manner, the value of function f continually decreases until either $f(h) = 0$ (i.e. the problem is solved) or until no move can improve the objective function (local optima).

Iterated Simulated Annealing In order to avoid local optima, we introduce a Simulated Annealing (SA) algorithm that works as follows. At each iteration, a random move (i,j) is picked up and this leads to one of the following situations: (i) if $\Delta(i,j) \geq 0$, perform the move, or (ii), if $\Delta(i,j) < 0$, accept the move with probability $e^{\frac{\Delta(i,j)}{T}}$, where T is the temperature parameter [24]. The cooling schedule changes the temperature by a cooling factor $\alpha < 1$ such that T becomes αT after a number of iterations R(relaxation time). By starting with a very large initial $T = T_{\text{start}}$, numerous "up" moves are accepted and the search process is allowed to leave the initial local optima. As the temperature decreases, fewer and fewer "up" moves are accepted and the process is led toward a new region and toward a new local optimum.

When the temperature becomes low enough and no new moves are accepted, the descent procedure is executed again so to perform intensification in the proximity of the potential solution provided by SA. After reaching another local optimum via descent, SA is applied again. In this context, one can also consider SA as a perturbation operator of an iterated descent search. This might recall Iterated Local Search principles, and, indeed, the general search process consists of a loop of SA and descent. The *general search process* is the following:

```
   RepairingDescent()
   $T_{\text{start}}$ = maxStartTempSA
1. SA($T_{\text{start}}$,$\alpha$, $R$)
2. RepairingDescent()
3. $T_{\text{start}} = T_{\text{start}} \cdot \gamma$
4. **if** $T_{\text{start}} \approx 0$ **then** $T_{\text{start}}$ = maxStartTempSA
5. **go to 1** while a *stopping condition* is not met.
```

By performing numerous rounds of Simulated Annealing (Step 1) and Repairing Descent (Step 2), many regions and local optima are visited. The SA start temperature decreases (Step 3) along the general search process so as to steadily decrease perturbations strength after each descent. This proved very useful in practice: the general search process starts out by performing strong perturbations (i.e. SA takes the search to regions quite far from the current local optimum), but, after a while, the general search process is gradually stabilized, i.e. perturbations become weaker and weaker. If the general search process gets into a local minimum that it can not leave (start SA temperature almost zero resulting in very weak perturbations), the whole strategy is restarted by resetting the SA start temperature (T_{start} is reset in Step 4).

The stopping condition is either to find the solution or to reach a limit of time; we used both predefined limits (e.g. 1000 seconds in Figure 4), but we prefer a time limit defined as the equal of the time spent by the extension procedure (e.g. in Section 6.5), so as to ensure a theoretical polynomial time for GI-Ext. The other parameters were calibrated with minimal tunning effort, resulting in $T_{\text{start}} = 100000$, $\alpha = 0.9$, $\gamma = 0.1$ and $R = 100000$. By searching a perfect optimal value for each parameter, one could skew the results slightly more in GI-Ext's favor, but not enough to upset our main conclusions.

5.2.4 Final Remarks

To summarize, GI-Ext is a Las Vegas algorithm, returning one of the following: (i) the graphs are surely isomorphic (if the linear construction or the heuristic finds a solution), (ii) the graphs are surely not isomorphic (if there are vertices with all possible associations forbidden by the matrix F) and (iii) the isomorphism status can not be determined in the given time but the graphs are probably not isomorphic. The third case occurs only very rarely, on crafted instances with many regularities.

Note that GI-Ext can also be used to discover isomorphic substructures induced in non-isomorphic graphs. In this case, it only returns a suitable matching and a value of the objective function f on the original graphs (Section 6.5); the information related to forbidden associations (matrix F) is not used so as to report the graphs as not isomorphic.

6 Numerical results, discussions and conclusion

For graph isomorphism, there is no common standardized benchmark on which all algorithms are tested, and, each of the previous papers introduced certain particular graph classes. As such, we present numerical experiments using both previous graph classes, as well as new instances that we consider relevant in our context. More exactly, we test GI-Ext on the following test instances:

1. the graph classes from the database for benchmarking graph isomorphism algorithms [16]: bounded valence graphs, mashes, randomly-generated graphs—this paper already showed the performance of other four algorithms on this test data set;

2. the graph classes on which the *nauty* algorithm was tested [29];

3. regular graphs—a difficult graph class that can also arise naturally in practice:

   (a) the *sparse* random regular considered in the *bliss* paper [22];

(b) dense regular graphs with up to $|V| = 20000$ vertices and $\frac{|V|(|V|-1)}{2} \approx 200.000.000$ edges;

4. strongly regular graphs—this class captures much of the difficulties of the GI problem,i.e. it represents crafted specially-constructed instances;

5. distorted isomorphic pairs of random graphs for a graph matching problem.

The running time is routinely expressed as a function of $|V|$. A classical method to obtain information whether this function is polynomial or exponential consists of plotting the log-log graph of the running time. This representation has the advantage that a polynomial function of the form x^β becomes a straight line of slope β. Furthermore, the slope of the running time function is machine independent, so we can *easily* compare algorithms developed on different machines in different contexts. Note that the results on higher order graphs are more reliable because they can better express the exponent of the leading term of a polynomial function of $|V|$.

6.1 The graph database

The database due to De Santo et al. [16] provides numerous instances of several widespread graph classes. These instance were used to compare four GI algorithms (*Ulmann*, *SD*, *Nauty* and *VF2*). The first class is represented by a family of random graphs with densities between 0.01 and 0.1. In fact, random graphs have been tested by most papers for indicative purposes, but they are not particularly challenging (it is known since 1980 that a straightforward $O(|V|^2)$ algorithm can be applied to almost all such graphs [2]).

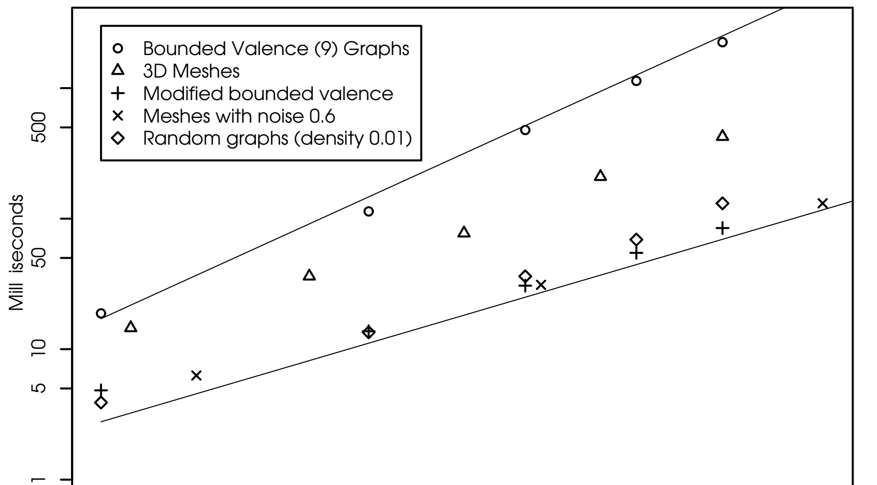

Figure 1: Log-log graph of the running time of GI-Ext on instances provided by the graph database [16]. Numerous other experiments not detailed in the paper confirm that any two graphs from the database can be processed within a time bounded by the two plotted slopes (of $\beta = 3.1$ and $\beta = 2$, respectively).

The mesh-connected graphs (2D, 3D, and 4D) represent a more regular and more difficult class. They are generated from a mesh—i.e. a regular grid of connected nodes (graph vertices). Each vertex is connected with a fixed number of mesh neighbors (i.e. 4, 6 or 8 for 2D, 3D, and respectively 4D meshes), except those at the border that have fewer neighbors. Irregular meshes are obtained by adding certain edges (according to a noise level between 0.2 and 0.6), and in some sense, regular meshes are more difficult. We provide running times only for 3D regular meshes and 4D irregular meshes with 0.6 noise, but very similar plots were obtained for other graphs from this class.

The most difficult class from the database (for GI-Ext) is represented by the *bounded valence graphs*, i.e. graphs that have the degree (the valence) of each vertex limited by a certain value (3, 6 or 9). The database includes graphs with a fixed valence (degree 3, 6 or 9). An easier class has been obtained by moving some edges, resulting in irregular bounded valence graphs. We report results on the most difficult class (9-bounded valence graphs), both the standard and the modified version.

Figure 1 shows the results on all the above mentioned graph classes. The instances are available on line (`amalfi.dis.unina.it/graph/`) and they have a maximum order of 1296; each class contains graphs with different intermediate values of $|V|$. All graphs were solved in polynomial time by GI-Ext. The two lines in Figure 1 have slope 2 and 3.1 showing that the running time of GI-Ext is always between $O(|V|^2)$ and $O(|V|)^{3.1}$. The lower bound is due to the fact that we do not use sparse matrices and the minimum complexity is always $O(|V|^2)$, as GI-Ext works with many $|V| \times |V|$ matrices. The upper bound is mostly due to the graph extension procedure that performs many steps for the bounded valence graphs. While GI-Ext is polynomial on all classes, *nauty* and *ulmann* are exponential in some cases (i.e. regular meshes, see [16, Fig 2]). On the other hand, VF2 is polynomial, too, but it is difficult to compare its slope with GI-Ext. We do not have this information for VF2, but, in absolute times, VF2 seems to be faster on certain instances and slower on others. We only mention that the tests we performed confirm this, but we would prefer not to provide detailed figures on behalf of other algorithms (their running speed can vary greatly from one machine to another).

Finally, we also tested GI-Ext on non-isomorphic pairs of graphs from the database; in this case, the decision of non-isomorphism was reached more rapidly, without needing to run the heuristic—because the extension procedure finds vertices $i \in V$ with no compatible association $i \to i' \in V'$ (via the matrix F). However, the total speed-up is limited because the running time is still a polynomial of the same order of magnitude—the number K of extension steps is the same for both isomorphic and non-isomorphic graphs. Generally speaking, non-isomorphism conflicts could be determined earlier if one checks the number of forbidden associations of matrix F earlier (e.g. after each extension step), but we prefer to focus here on a unique algorithm version.

6.2 Graph classes from the *nauty* paper

The *nauty* program was tested on several graph classes [29, pp. 79–82] and its running time was plotted on log-log graphs (Figure 3.1, p. 80), in the same manner as we do. Recall that a polynomial function of the form x^β becomes a linear function of slope β on the log-log graph; for each considered class, we also provide the β slope reported in the *nauty* paper:

1. random graphs of density 0.5 ($\beta = 2.0$);
2. m-dimensional cube ($\beta = 2.3$);
3. random circular graphs ($\beta = 2.2$)—a circular graph with vertices $\{1, 2, \ldots, |V|\}$ is built by connecting all i and j such that $|i - j| \in W$, where W is a set of cardinal 5 in $\{1, 2, \ldots, \lfloor \frac{|V|-1}{2} \rfloor\}$;
4. regular graphs with degree 6 and 20 (built essentially by superimposing 2-regular graphs with no common edge);
5. strongly regular graphs with up to 60 vertices.

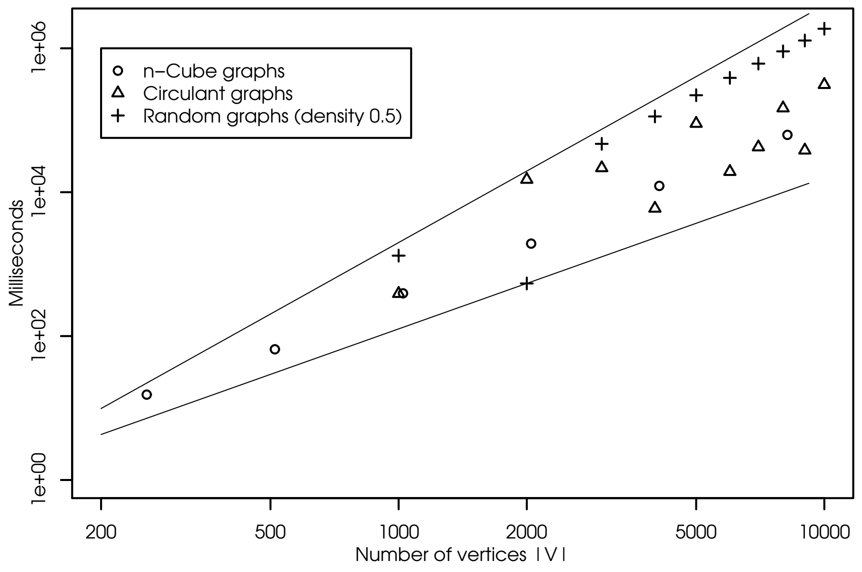

Figure 2: Log-log graph of the running time of GI-Ext on the graph classes considered in the *nauty* paper. We also show two lines of slope $\beta = 3.3$ and $\beta = 2.1$.

Figure 2 plots the results of GI-Ext on the first three graph classes above—all graphs were re-generated at orders between 100 and 1000. The case for the rest of the graph classes (i.e. regular and strongly regular graphs) is taken into consideration in greater detail in Section 6.3 and 6.4. However, this figure shows that GI-Ext has a polynomial run time on the first three classes; the exponent (i.e. slope on the log-log graph) is between 2.1 and 3.3. It seems that *nauty* is faster (β is between 2.0 and 2.3), but GI-Ext is however polynomial. The complexity of GI-Ext is in the region of $O(|V|^3)$ and it is due to the extension step. As such, it never reaches $O(|V|^4)$—in fact, the complexity of the extension operation is *theoretically* bounded (see Section 5.2.1).

6.3 Regular graphs

We consider regular graphs essential, because they are at the same time difficult and quite general. Unlike more complex crafted graph classes, regular graphs can often arise naturally in applications, as for example: graphs from images considering each pixel as a vertex with 8 (or 4) neighbors, graphs of a circuit design in which all components have a fixed number of logical inputs and outputs (e.g. binary logic gates), graphs associated with k-satisfiability problems, etc. Furthermore, regular graph isomorphism is GI-complete [6, 18, 40], i.e. if regular graphs can be solved in polynomial time, so can be any graph class.

Sparse regular graphs have been considered several times in the literature (e.g. a class of 6-regular and 20-regular graphs since the *nauty* paper), but we consider it might be more important to test dense regular graphs. Note that the graphs of bounded degree $d << |V|$ are known to be solvable in polynomial time [27] since the early 1980s. Furthermore, we provide results on regular graphs with hundreds of millions of edges (our largest dense regular graph has $\frac{20000*19999}{2}$ edges). To the best of our knowledge, it is for the first time a graph isomorphism algorithm has been tested on such graph sizes.

6.3.1 Sparse regular graphs

First, we test the 3-regular graphs introduced in [22] and report results in Table 1. We compare the running time of GI-Ext with the running times of *saucy*, *nauty*, and *bliss*—the figures for these three algorithms are taken from the *bliss* paper [22]. Following the format from this paper, for each graph order, we report the minimum and maximum solving time required by GI-Ext. In the *bliss* paper, each algorithm was allowed a maximum time of 600 seconds and we are convinced our machine would report similar time values for *saucy* and *nauty* (Columns 3 and 4), i.e. the same numbers multiplied by a constant. Notice that the values for the canonical labeling algorithms represent only the time of processing *one* graph—finding an isomorphism would also require labeling the second graph. To test GI-Ext, we first construct a randomly-permuted copy of the input graph, and only then we apply the search on both graphs.

$\|V\|$	GI-Ext	*nauty*	*saucy*	*bliss*
	$T_{\min}[s]$–$T_{\max}[s]$	$T_{\min}[s]$–$T_{\max}[s]$	$T_{\min}[s]$–$T_{\max}[s]$	$T_{\min}[s]$–$T_{\max}[s]$
1000	0.36 – 0.42	9.00–9.08	1.11–1.18	0.06–0.07
2000	1.52 – 1.71	117.13–120.74	8.57–8.63	0.27–0.3
3000	3.47 – 3.94	time out	28.34–28.44	0.64–0.71
4000	8.47 – 10.03	time out	62.22–64.50	1.09–1.20
5000	18.27 – 21.39	time out	122.23–129.40	1.70–1.93
6000	17.48 – 20.35	time out	208.65–215.22	2.65–2.98
7000	29.88 – 35.72	time out	335.10–365.94	3.61–4.04
8000	53.12 – 57.56	time out	499.67–508.16	4.87–5.49
9000	79.15 – 93.71	time out	time out	7.43–8.28
10000	123.99 – 150.09	time out	time out	7.90–8.74

Table 1: Running time comparison for GI-Ext and canonical labeling algorithms on 3-regular graphs. Other algorithms not using canonical labeling techniques seem slower than *nauty*.

Interesting conclusions can be drawn from Table 1. Only *bliss* and GI-Ext can solve the largest graphs in a short time and they are both polynomial. It seems that other approaches (without canonical labeling techniques) can not compete in polynomial time—at least not well-known algorithms provided by the VFLib Graph Matching Library (`amalfi.dis.unina.it/graph/db/vflib-2.0/doc/vflib.html`). By installing this well-coded library (provided by the graph database paper [16]), since the input 3-regular graphs are also published on-line (via the *bliss* paper, `www.tcs.hut.fi/Software/benchmarks/ALENEX-2007/`), one can easily check the timing of other (non canonical labeling) algorithms, but we do not provide details here.

However, *bliss* obtains running times nearly inferior to $O(|V|^2)$ as it takes profit from refined sparse data structures achieved from an engineering effort. Notice that our GI-Ext implementation does not use sparse data structures, and so, it can never reach a complexity below $O(|V|^2)$. GI-Ext executes K extension steps of total complexity $O(K \cdot |E||V|) = O(K \cdot 3|V|^2)$ and, since K is in $O(|V|)$ for these sparse 3-regular graphs, the complexity of GI-Ext is in the region of $O(|V|^3)$. While *saucy* also seems to run in $O(|V|^3)$, GI-Ext is better in terms of absolute times—and one should be aware that the *saucy* column reports just the time of generating the automorphism group (not the final labeling).

6.3.2 Dense regular graphs

It is important to also test *dense* regular graphs because GI-Ext is not (yet) optimized for sparse structures. We tried several values of the degree d (at least $\frac{1}{10}|V|$) and the hardest graphs were those with $d = \frac{1}{2}|V|$ (for larger densities, one can complement the graph). Therefore, we generated pairs of random regular graphs of degree $\frac{1}{2}|V|$ with $|V|$ ranging from 1000 to 20000. We do not claim that these regular graphs are generated *uniformly* at random, but the theory of generating such structures is a difficult issue (see Appendix 1 presenting the generation procedure).

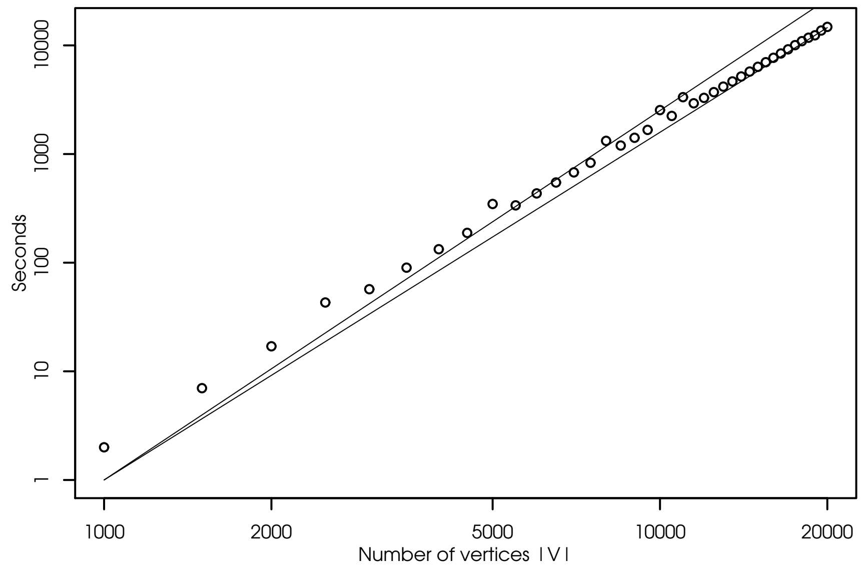

Figure 3: Log-log graph of the running time of GI-Ext on *dense* random regular graphs with $|V|$ ranging from 1000 to 20000. The functions $\left(\frac{|V|}{1000}\right)^{3.2}$ and $\left(\frac{|V|}{1000}\right)^{3.4}$ are also plotted.

Figure 3 plots the running time of finding the isomorphism between two copies of a random regular graph (i.e. for each graph, we obtain a copy by randomly permuting its vertices—as for all tested classes). This figure directly confirms that the running time of the GI-Ext is polynomial. Note that GI-Ext is available on-line and it can be very easily tested using several graph input formats.[1]

6.4 Strongly regular graphs

The strongly-regular graphs represent the hardest instances, more difficult than simply regular graphs. To the best of our knowledge, only two papers [22, 26] provide results on such graphs with $|V| > 100$, and, furthermore they do not claim to be polynomial. However, efficient algorithms for strongly graphs seem to require canonical labeling techniques (i.e. based on computing automorphism groups). In this section, we provide however results on Paley *dense* strongly-regular graphs; these graphs can be generated for any prime $|V|$ of the form $4k+1$, and so, we can easily investigate the run time as a function of $|V|$. Many other classes of strongly-regular graphs are presented in the *bliss* paper, but we do not claim there are many algorithms without canonical labeling techniques that compete well with *bliss* on this data set.

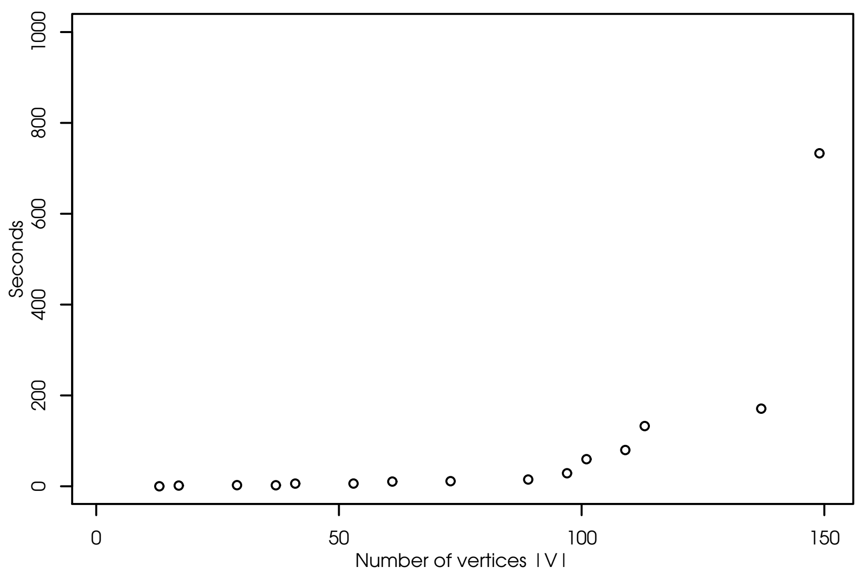

Figure 4: The running time of GI-Ext on strongly-regular Paley graphs.

Figure 4 shows the running time function for strongly-regular Paley graphs, giving evidence that GI-Ext is able to solve small strongly-regular graphs (tens or hundreds of vertices). While GI-Ext is not yet perfected to deal with this class, one might notice that *bliss* or *saucy* are based on canonical labeling work which began 30 years ago with *nauty*. We share the view of the *saucy* paper that the description of this theory has to "delve into the depths of discrete mathematics"; GI-Ext is much simpler, and so, it could be more easily (re-)implemented without requiring understanding complex theories. In this sense,

[1]`www.lgi2a.univ-artois.fr/~porumbel/giext/`; there is also a permanent mirror (an exact copy) at `sites.google.com/site/danielporumbel/giext/`. All provided results are obtained on a 2.79GHz computer, by compiling the programs using *gcc* with the *-O2* optimization flag.

we consider the simplicity rather as an advantage than as a drawback. Furthermore, the next section shows a different advantage over canonical labeling: GI-Ext can also be employed for certain graph matching problems arising naturally in practice. Indeed, since it does not depend on global structures like the automorphism group, GI-Ext can also construct only a *partial isomorphism* between two graphs,i.e. to (try to) find the largest set of vertex associations respecting the isomorphism constraints.

6.5 Graph matching example—partial graph isomorphism

Recall that the problem addressed by GI-Ext can actually be expressed as follows: *find the mapping* $h : V \to V'$ *such that* $f(h)$ *is minimum*—see also (3), Section 4.1. If the input graphs are isomorphic, then the problem is equivalent to finding isomorphism h^* such that $f(h^*) = 0$. But in the general case, minimizing f is equivalent with minimizing the number of edges (in any of the two graphs) not associated to edges in the other graph.

A mapping $h : V \to V'$ that satisfy $M_{ij} = M_{h(i),h(j)} \forall i, j \in \widetilde{V}$ is a $\widetilde{V}$*-partial isomorphism.* We say that the largest set $\widetilde{V}$ verifying this property indicates the *partial isomorphism order*: $\frac{|\widetilde{V}|}{|V|}$. In this sense, an isomorphism is a partial isomorphism of order 100%. Discovering partial isomorphisms could be very useful in certain applications. For example, consider taking satellite images of an area at different moments of time: if a new construction arises between the two moments, the associated graphs are essentially identical with the only difference the one of the graphs contain some additional vertices (corresponding to the new construction). Establishing a partial isomorphism could be very useful to detect unchanged parts in the two satellite images.

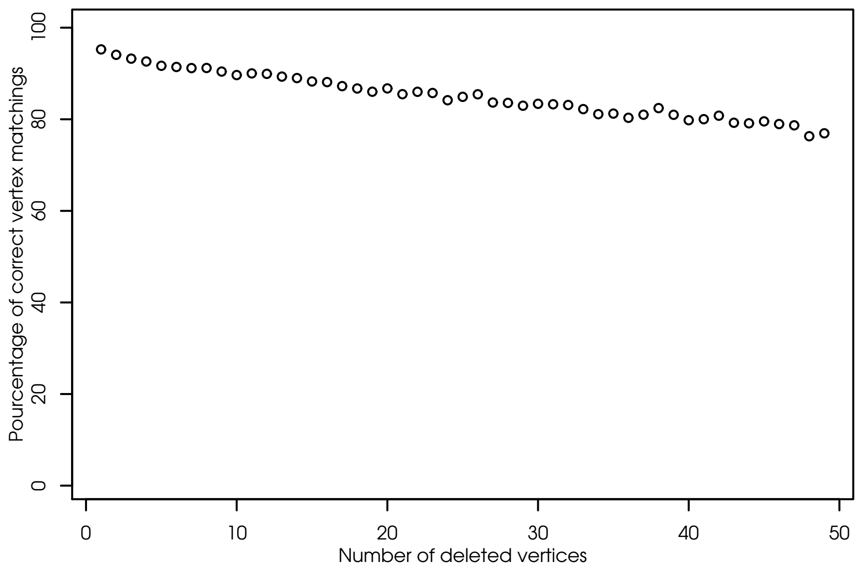

Figure 5: Order of the partial isomorphism discovered by GI-Ext in two distorted initially-isomorphic graphs—e.g. in the second graph, a number of vertices is erased (on x axis). GI-Ext identifies important isomorphic substructures (induced graphs), accounting for 80%$|V|$ even after erasing 50 vertices (out of 1000).

To test GI-Ext in finding partial isomorphisms, we consider the following problem.

Assume G and G' are isomorphic via $h^* : V \to V'$ and let $G(U)$ be the subgraph induced by a subset U. In practice, $G(U)$ is obtained by erasing vertices $V - U$ from V (along with their edges). In our tests, we will erase a number of vertices from 1 to $5\%|V|$ and we (try to) construct a partial isomorphism on the remaining vertices U. The operations needed to run exactly the same GI-Ext program on input graphs $G(U)$ and G' are the following: (i) add $|V| - |U|$ dummy singleton vertices to U because the GI-Ext specification is written for graphs of equal order; (ii) disable the matrix F of forbidden associations because we do not target exact isomorphism any more; and (iii) run GI-Ext with a polynomial time stopping condition (allowing the same amount of time as used by the extension procedure) and let h_U the best solution ever reached. We evaluate the successfulness of GI-Ext according to the order of the partial isomorphism h_U—it is essentially the pourcentage of vertices i such that $h_U(i) = h^*(i)$.

Figure 5 plots the order of the partial isomorphism returned by GI-Ext, as a function of the number of erased vertices $|V - U|$; the initial input graphs had $|V| = 1000$ and density 50%. A conclusion of this figure is that, if one erases a vertex in one graph (i.e. $|V - U| = 1$), GI-Ext can still identify almost 100% of the initial isomorphism. This is not possible with canonical labeling algorithms that depend on computing the automorphism groups of the whole graphs.

Notice that associations $i \to h^*(i)$ are not regarded as "correct" when i is a neighbor of a vertex in $V - U$. Indeed, such associations would not be isomorphic for f because the edge structures around them are not the same ($i \in V$ and $h^*(i) \in V'$ have different degrees in $G(U)$ and G'). Essentially, this is the reason GI-Ext can *not* reconstruct the isomorphism on all non-erased vertices—the isomorphism can be reconstructed only on *the vertices not connected to the erased ones.*

Most of the typical graph isomorphism algorithms work only on the whole graphs and can not be adapted to partial isomorphism. The graph extension procedure constructs vertex extended labels *not only* to partition the vertices of an individual graph; they also indicate pairs of vertices $i \in V$ and $i' \in V'$ that can be put isomorphically into correspondence. An association $i \xrightarrow{h} i'$ might satisfy the isomorphism constraints only if i and i are compatible (see Definition 1). As such, the extension procedure has a far more general applicability than pure graph isomorphism.

Gi-Ext is more adapted to partial isomorphism than to subgraph isomorphism problems because partial isomorphism focuses only on *induced* subgraphs—i.e. not general subgraphs, the whole structure on $\widetilde{V}$ is preserved via a partial isomorphism h (GI-Ext is useful on identifying isomorphisms between this type of structures). The problem from this section is more related to the maximum common subgraph problem because it is reasonably to consider that the best partial isomorphism shares many vertex associations with the solution of the maximum common subgraph problem. Recall that, in this final experiment, we allow for the heuristic part the same time as for the first polynomial steps. This results in a total polynomial time theoretically bounded by $O(|E| \cdot |V|^3)$—but it is $O(|V|^2)$ in practice. However, GI-Ext can deal with graphs with thousands of edges, while smaller graph sizes are commonly considered in the literature of the maximum common subgraph problem [11].

6.6 Final remarks on speed and memory requirements

In terms of memory costs, the spatial complexity of GI-Ext is $O(|V|^2)$ as we use a bounded number of matrix structures of size $|V| \times |V|$ (we record only the last matrix of the series $M^1, M^2, \ldots, M^K$). This is easily confirmed by experimentation: GI-Ext requires about $30 \times |V|^2$ bytes of free RAM. It is only the memory restriction of our machine (16 GB) that prevented us from testing dense graphs with more than 20.000 vertices. For certain matrices we used 8-bit integers (i.e. for F), but we used unsigned long 64-bit integers for the elements of $\overline{M}$. However, for large values of K, there might exist elements exceeding $2^{64} - 1$; therefore, we consider all addition operations in the Arithmetic Modulo 2^{64} (in our C++ implementation, the variables are encoded so that $2^{64} - 1 + 1 = 0$). This observation does not change the fact that $\bar{f}(h^*) = 0$ if h^* is an isomorphism, because if $\overline{M}_{i,j} = \overline{M'}_{h(i),h(j)}$, then $\overline{M}_{i,j} = \overline{M'}_{h(i),h(j)}$ $(Modulo\ 2^{64})$. However, it is still theoretically possible to have the Modulo equality without the non-Modulo equality; this never happens in practice but we can easily verify any reported $\bar{f}$ solution with the objective function f (Equation 3).

Finally, we note that the graph extension procedure can be easily parallelized. That would skew the results more in GI-Ext's favor, but not enough to change our main conclusions. However, the goal of the paper is not to enter into such technical speed-up discussions and we only mention that we roughly reduced by half the running time by using 4 processors; all graphs from the library [16] could be solved in less than one second. The openMp library provided easy-to-use directives for the parallelization of the `For` loops of Algorithm 1—technical details about the parallel program are also available on-line (see page 19, Footnote 1).

7 Conclusion

To conclude, GI-Ext is a flexible algorithm, very useful for both pure isomorphism and for certain more general matching problems. Experimental tests show that it is effective in finding isomorphisms for several input graph classes, including complete benchmarks from other papers. While very recent GI canonical labeling algorithms are based on work which began 30 years ago, we propose a viable new alternative that competes well with these state-of-the-art methods. Although GI-Ext has the inconvenience of not being exact, it has the advantage of being quite simple and it can be implemented without needing deep mathematics. It can deal with difficult (dense and regular graphs) with hundreds of millions, even billions of edges (more than many previous algorithms).

One of the main ideas is related to the $O(|E| \cdot |V|^2)$ extension operation that is used to construct complete extended graphs in polynomially bounded time—i.e. $O(|E| \cdot |V|^3)$. The complete extended structure is actually a hash code of the initial graph; each extended vertex along with its extended edges encode a wealth of information of the initial graph structure on a large proximity around this vertex. While classical graph matching heuristic algorithms only (try to) reduce violated constraints involving neighboring vertices, the extended graph enables GI-Ext to detect and prevent a larger class of isomorphism violations, between apparently compatible vertices—so as to seriously constrict the search space.

Last but not least, notice that, while we used the extension operations in conjunction with a heuristic, the interest in complete extended graphs is not limited to a heuristic context. Our techniques could be quite easily used to construct an exact procedure, e.g. by replacing the local search with a branch and bound algorithm or with an integer programming solver. In practical terms, the implementation would pose however certain technical issues —e.g. classical integer data structures would not be long enough to record elements of matrix $\overline{M}$. Finally, notice that the matrix F of incompatible associations can be inserted in other existing methods (including canonical labeling algorithms) in order to enhance their performance.

References

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

[2] L. Babai, P. Erdos, and M. Selkow. Random graph isomorphism. *SIAM journal of computing*, 9(3):628–635, 1980.

[3] L. Babai, D.Y. Grigoryev, and D. M. Mount. Isomorphism of graphs with bounded eigenvalue multiplicity. In *Fourteenth Annual ACM Symposium on Theory of Computing*, pages 310–7324, 1982.

[4] L. Babai and E.M. Luks. Canonical labeling of graphs. In *Proceedings of the fifteenth annual ACM symposium on theory of computing*, pages 171–183. ACM New York, NY, USA, 1983.

[5] E. Bengoetxea, P. Larrańaga, I. Bloch, A. Perchant, and C. Boeres. Inexact graph matching by means of estimation of distribution algorithms. *Pattern Recognition*, 35(12):2867–2880, 2002.

[6] K.S. Booth. Isomorphism testing for graphs, semigroups, and finite automata are polynomially equivalent problems. *SIAM Journal of Computing*, 7(3):273–279, 1978.

[7] H. Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern Recognition Letters*, 18(8):689–694, 1997.

[8] H. Bunke. Recent developments in graph matching. In *Proc. 15th International Conference on Pattern Recognition*, volume 2, pages 117–124, 2000.

[9] R.M. Cesar, E. Bengoetxea, I. Bloch, and P. Larrañaga. Inexact graph matching for model-based recognition: Evaluation and comparison of optimization algorithms. *Pattern Recognition*, 38(11):2099–2113, 2005.

[10] D. Conte, P. Foggia, C. Sansone, and M. Vento. Thirty Years of Graph Matching in Pattern Recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(3):265–298, 2004.

[11] D. Conte, P. Foggia, and M. Vento. Challenging complexity of maximum common subgraph detection algorithms: A performance analysis of three algorithms on a wide database of graphs. *Journal of Graph Algorithms and Applications*, 11(1):99–143, 2007.

[12] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 26(10):1367–1372, 2004.

[13] D.G. Corneil and C.C. Gotlieb. An Efficient Algorithm for Graph Isomorphism. *Journal of the Association for Computing Machinery*, 17(1):51–64, 1970.

[14] P.T. Darga, M.H. Liffiton, K.A. Sakallah, and I.L. Markov. Exploiting structure in symmetry detection for CNF. In *Proceedings of the 41st annual conference on Design automation*, pages 530–534. ACM New York, NY, USA, 2004.

[15] P.T. Darga, K.A. Sakallah, and I.L. Markov. Faster symmetry discovery using sparsity of symmetries. In *Proceedings of the 45th annual conference on Design automation*, pages 149–154. ACM New York, NY, USA, 2008.

[16] M. De Santo, P. Foggia, C. Sansone, and M. Vento. A large database of graphs and its use for benchmarking graph isomorphism algorithms. *Pattern Recognition Letters*, 24(8):1067–1079, 2003.

[17] I. S. Filotti and J. N. Mayer. A polynomial-time algorithm for determining the isomorphism of graphs of fixed genus. In *STOC '80: Proceedings of the twelfth annual ACM symposium on Theory of computing*, pages 236–243. ACM, 1980.

[18] S. Fortin. The graph isomorphism problem. Technical Report TR96–20, University of Alberta, Edmonton, Canada, 1996.

[19] M.R. Garey and D.S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*, pages 154–161,202. W. H. Freeman & Co., New York, NY, USA, 1979.

[20] S. Gold and A. Rangarajan. A graduated assignment algorithm for graph matching. *IEEE Transactions on pattern analysis and machine intelligence*, 18(4), 1998.

[21] D.S. Johnson, C.R. Aragon, L.A. McGeoch, and C. Schevon. Optimization by Simulated Annealing: An Experimental Evaluation. *Operations Research*, 39(3):378–406, 1991.

[22] T. Junttila and P. Kaski. Engineering an efficient canonical labeling tool for large and sparse graphs. In D Applegate et al., editors, *Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments and the Fourth Workshop on Analytic Algorithms and Combinatorics*, pages 135–149. SIAM, 2007.

[23] R.K. Kincaid. A molecular structure matching problem. *Computers and Operations Research*, 24(1):25–35, 1997.

[24] S. Kirkpatric, CD Gelatt, and MP Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[25] G. Levi. A note on the derivation of maximal common subgraphs of two directed or undirected graphs. *Calcolo*, 9(4):341–352, 1973.

[26] José Luis López-Presa and Antonio Fernández Anta. Fast algorithm for graph isomorphism testing. In *SEA*, volume 5526 of *LNCS*, pages 221–232, 2009.

[27] E. M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of computer and system sciences*, 25(1), 1982.

[28] B. Luo and E.R. Hancock. Structural graph matching using the EM algorithm and singular value decomposition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1120–1136, 2001.

[29] B.D. McKay. Practical graph isomorphism. *Congressus Numerantium*, 30:45–87, 1981.

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

[31] J.W. Raymond and P. Willett. Maximum common subgraph isomorphism algorithms for the matching of chemical structures. *Journal of Computer-Aided Molecular Design*, 16(7):521–533, 2002.

[32] O. Sammoud, C. Solnon, and K. Ghédira. Ant algorithm for the graph matching problem. In G.R. Raidl et al., editors, *Evocop*, volume 3448 of *LNCS*, pages 213–223. Springer, 2005.

[33] A. Sanfeliu and K.S. Fu. Distance measure between attributed relational graphs for pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, 13(3):353–362, 1983.

[34] D.C. Schmidt and L.E. Druffel. A Fast Backtracking Algorithm to Test Directed Graphs for Isomorphism Using Distance Matrices. *Journal of the Association for Computing Machinery*, 23(3):433–445, 1976.

[35] M. Singh, A. Chatterjee, and S. Chaudhury. Matching structural shape descriptions using genetic algorithms. *Pattern Recognition*, 30(9):1451–1462, 1997.

[36] J. R. Ullmann. An algorithm for subgraph isomorphism. *Journal of the Association for Computing Machinery*, 23:31–42, 1976.

[37] Y.K. Wang, K.C. Fan, and J.T. Horng. Genetic-based search for error-correcting graph isomorphism. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 27(4):588–597, 1997.

[38] M.L. Williams, R.C. Wilson, and E.R. Hancock. Deterministic search for relational graph matching. *Pattern Recognition*, 32(7):1255–1271, 1999.

[39] L. Wiskott, J.M. Fellous, N. Krüger, and C. von der Malsburg. Face Recognition by Elastic Bunch Graph Matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 775–779, 1997.

[40] V. N. Zemlyachenko, N. M. Korneenko, and R. I. Tyshkevich. Graph isomorphism problem. *Journal of Mathematical Sciences*, 29(4):1426–1481, 1985.

[41] K. Zhang, R. Statman, and D. Shasha. On the editing distance between unordered labeled trees. *Information Processing Letters*, 42(3):133–139, 1992.

Appendix: Simple Generation of Regular Graphs

To generate a pair of random regular graphs G and G', we first construct a random bijection $\sigma : V \to V'$ and all operations that we apply to vertex $i \in V$, will also be applied to $\sigma(i) \in V'$. The construction of G is incremental, i.e. a $(d+1)$-regular graph is constructed on top of a d-regular graph. Please note that if one needs a regular graph of degree $d > \frac{|V|}{2}$, one can generate it using the complementary of a $(|V| - d)$-regular graph; as such, it is enough to consider the case $d \leq \frac{|V|}{2}$.

To construct a $(d+1)$-regular graph from a d-regular graph, let i and j be any two (random) vertices of degree d. If $\{i, j\} \notin E$, insert an edge between i and j and their degree becomes $d + 1$. If $\{i, j\} \in E$, the algorithm searches an edge $\{i', j'\}$ such that: (i) $\{i, i'\} \notin E$ and (ii) $\{j, j'\} \notin E$ (such an edge was always found, there are more than $(|V| - d)^2 > (\frac{|V|}{2})^2$ pairs (i', j') satisfying (i)–(ii) and they are not all unlinked). The last operation consists of incrementing the degree of i and j by linking i to i', j to j' and by disconnecting i' and j'.

In this manner, we were able to increment the degree of any two pairs of vertices i and j of degree d. We repeat this operation $\frac{|V|}{2}$ times until all vertices increment their degree ($|V|$ needs to be even), and, thus, the d-regular graph is transformed into a $(d+1)$-regular graph. Examples of generated graphs are publicly available on the website with the source code (see Footnote 1, page 19), both in an adjacency matrix format and in a list-of-edges format. Although difficult to theoretically prove, one can see these graphs have and important random component and no specific particularities.