Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

Pattern Recognition Letters 24 (2003) 1067–1079

Pattern Recognition Letters

www.elsevier.com/locate/patrec

A large database of graphs and its use for benchmarking graph isomorphism algorithms

M. De Santo [a], P. Foggia [b], C. Sansone [b], M. Vento [a,*]

[a] *Dipartimento di Ing. dell'Informazione e di Ing. Elettrica, Università di Salerno, Via Ponte Don Melillo, 1 I-84084 Salerno, Italy*
[b] *Dipartimento di Informatica e Sistemistica, Università di Napoli "Federico II" Via Claudio, 21 I-80125 Napoli, Italy*

Abstract

Despite of the fact that graph-based methods are gaining more and more popularity in different scientific areas, it has to be considered that the choice of an appropriate algorithm for a given application is still the most crucial task.

The lack of a large database of graphs makes the task of comparing the performance of different graph matching algorithms difficult, and often the selection of an algorithm is made on the basis of a few experimental results available.

In this paper we present an experimental comparative evaluation of the performance of four graph matching algorithms. In order to perform this comparison, we have built and made available a large database of graphs, which is also described in detail in this article.

Keywords: Graph database; Benchmarking; Exact graph matching; Graph isomorphism

1. Introduction

Graphs are widely used in several areas for representing many different kinds of information: syntactic structures of a language, chemical compounds, geographic maps, computer networks, software systems architectures, database structures, complex objects or scenes found in static or motion pictures are just a few examples of what has been described using graphs (Rouvray and Balaban, 1979; Wong, 1992; Shearer et al., 1998; Abdulrahim and Misra, 1998).

Many of these applications require some kind of comparison between graphs, which can be formulated as a graph matching problem. In particular, in several important cases, the application entails the search for graph isomorphism or for graph–sub-graph isomorphism. In the last three decades, plenty of algorithms for this problem, using either optimal or approximate methods, have been proposed with the specific aim of reducing computational complexity of the matching process. Far from giving an exhaustive list, some widely known algorithms are described in (Hopcroft and Wong, 1974; Luks, 1982; Ullmann, 1976; Bunke and Messmer, 1995; Cordella et al., 1999).

* Corresponding author. Tel.: +39-08996-4254; fax: +39-08996-4218.
E-mail addresses: desanto@unisa.it (M. De Santo), foggiapa@unina.it (P. Foggia), carlosan@unina.it (C. Sansone), mvento@unisa.it (M. Vento).

Although the authors of each novel algorithm usually provide experimental results supporting the claim that their method can outperform the previous ones at least under some circumstances, it is almost always very difficult for a user to choose the algorithm which is best suited for dealing with the problem at hand. In fact, a report of the algorithm performance on a specific test case can often provide no useful clues about what will happen in a different domain.

Unfortunately, only a few papers face the problem of making an extensive comparison of graph matching algorithms in terms of key performance indices (memory and time requirements, maximum graph size, etc.) (Bunke and Vento, 1999). So, it seems that the habit of proposing more and more new algorithms is prevailing against the need of assessing the performance of the existing ones in an objective way. As a consequence, the users of graph-based approaches can only use qualitative criteria to select the algorithm that seems to better fit with the application constraints.

As the popularity of graphs as a representation tool grows also for complex problems requiring hundreds or even thousands of nodes, it becomes more and more important for building robust and effective systems that we are able to choose the most appropriate algorithms to be used as building blocks. Hence, some effort must be put in the task of comparing graph matching algorithms.

If we try to understand the reasons why up to now no serious attempt has been made for comparing graph matching algorithms, we can easily recognize that one of the main difficulties is the lack of standard databases of graphs specifically designed for this purpose. In other research fields (for example, OCR), the availability of large de-facto standard databases improves the verifiability and the comparability of the experimental results of each method; why this cannot be done also for graph matching?

A possible answer is that the creation of a graph database is definitely not a simple task, since several issues have to be faced. The first question is: should the graphs be collected from real-world applications or should they be generated according to some probabilistic model? The latter choice, besides being simpler to implement, permits a finer control over the features of the graphs; but then which model(s) will faithfully reflect the performance obtainable in a given real application? Further questions: how large should the graphs be? How dense? How many graphs should be comprised in the database to make it statistically representative? Should semantic attributes be attached to nodes and edges, and if so, how should those attributes be obtained?

Since these problems have no simple answer on which the majority of the research community can agree, it is unreasonable that a single comprehensive database can be built which will satisfy all the needs of everyone. Nevertheless, it is our opinion that it is important to start building and sharing graph data sets, to provide a common ground for comparing the algorithms' performance. A database which lacks some property desirable for some class of users is better than no database at all.

Even after settling the question of the database, some other issues remain open. In order to provide a fair comparison, it should be taken into account that algorithms often are optimized for different usage patterns. Inexact graph matching algorithms are more robust, but also considerably slower than exact matching algorithms. Suboptimal algorithms can be quite faster, but may fail in finding a solution even if it exists. Some kinds of algorithms can be quite slow when matching two graphs, but show a considerable speed-up when matching one graph against a large set of prototypes. Other algorithms can be impressive on small graphs, but, due to a significant memory usage, can result definitely inapplicable to larger ones. As a consequence, a comparison is meaningful only if the algorithms being compared have similar characteristics; otherwise little or no useful information can be gained.

In this paper we present an experimental comparison of four exact graph matching algorithms. In order to perform this benchmarking activity, we have built and made available to the public domain a large database of graphs.

In the rest of the paper, we will describe in more detail the composition of the database. Then we will provide an overview of graph matching algorithms, followed by some more information about the algorithms used for this comparison and the reasons behind their choice. Finally, the obtained performance of the algorithms on the database will be presented and discussed.

2. The database

In general, two approaches can be followed for generating a database; a first way is to start from graphs obtained from real data, otherwise the database can be obtained synthetically. Although the first approach allows us to obtain rather realistic graphs, it is generally more expensive as it requires the collection of real data and the selection of the set of algorithms to be used for obtaining graphs from data. In this case the graphs are dependent on both the domain under consideration and the pre-processing algorithm used, reducing significantly the generality of the database and its reusability in other contexts. On the contrary, the artificial generation of graphs is not only simpler and faster than collecting graphs from real applications, but also allows us to control the variation of several critical parameters of the underlying graph population, such as the average number of nodes, average number of edges per node, number of different labels, and so on. Starting from these considerations, we decided to generate the database synthetically, so obtaining a quite large database of graphs, that can be also easily expandable, in a relatively short time.

The choice of the kind of graphs to be included in the database started with an analysis of the graphs mainly used by members of the IAPR-TC15 community (see http://www.iapr-tc15.unisa.it); a classification of various kinds of graphs used within the pattern recognition field has been recently proposed in (Bunke and Vento, 1999). The database is structured in pairs of graphs. Two categories of pairs of graphs have been introduced. Pairs made of isomorphic graphs and pairs made of graphs in which the second graph is a sub-graph of the first one. A total of 72,800 pairs of graphs have been generated: 18,200 pairs of isomorphic graphs and 54,600 pairs for which a sub-graph isomorphism exists. Each category of pairs is made up of graphs that are different for structure and size. In particular, the following kinds of graphs have been considered:

- Randomly connected graphs;
- Regular meshes, with different dimensionality: 2D, 3D and 4D;
- Irregular meshes;
- Bounded valence graphs;
- Irregular bounded valence graphs.

Each kind of graphs has pairs of different size, ranging from few dozens to about 1000 nodes (i.e. *small* and *medium* size graphs according to the classification presented in (Bunke and Vento, 1999)). For each size and kind of graphs 100 different pairs have been generated.

A brief description of the properties of each kind of graph and of the motivation inspiring the choice of including them in the database is given in Section 2.1, together with the number of generated isomorphic pairs per kind.

In case of graph sub-graph isomorphism we have generated pairs in which the two graphs (satisfying the graph–sub-graph condition) have three different size ratios, namely 0.2, 0.4 and 0.6. So, each kind of graphs contains a number of pairs three times higher than that obtained in case of the isomorphism.

The graphs composing the whole database have been distributed on a CD during the 3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition (*GbR2001*) and are also publicly available on the web at the http://amalfi.dis.unina.it/graph.

Graphs are stored in a binary format. Each graph is contained in a file, and the files are grouped into directories according to the kind of graphs and to the value of the parameters used in the generation.

Each graph file is composed by a sequence of 16-bit words; the words are encoded in little-endian format (e.g. LSB first). The first word represents the number of nodes in the graph. Then, for each node, there is a word encoding the number of edges coming out of that node, followed by a sequence of words encoding the endpoints of those edges.

In addition to the graphs, we have also made available the sources of the programs used for their generation, which are coded in C++. For the generation of the whole database, a Unix shell script has been written. The database has been generated on a computer with a 350 MHz Pentium III processor and 128 MB of RAM. The underlying operating system was Linux 2.2.14, and the C++ compiler was egcs release 1.1.2. The total time required by the generation has been of about 7 h, and the overall size of the compressed archive is about 420 MB. The database is completed with a simplified ground truth; for each pair of graphs (both in the case of isomorphism and/or sub-graph isomorphism) the number of existing solutions is reported. This information has been obtained by using the graph matching algorithms described in Section 3.

2.1. Kind of graphs

Randomly connected graphs are graphs in which the edges connect nodes without any structural regularity. They have been introduced for simulating applications in which the entities (represented by nodes) can establish relations (represented by edges) with any other entity (not only the surrounding ones) independently of the relative positions. This hypothesis typically occurs in the middle and high processing levels of a computer vision task.

It is assumed that the probability of an edge connecting 2 nodes of the graph is independent on the nodes themselves. The same model proposed in (Ullmann, 1976) has been adopted for generating these graphs: it fixes the value η of the probability that an edge is present between two distinct nodes n and n'. The probability distribution is assumed to be uniform.

According to the meaning of η, if N is the total number of nodes of the graph, the number of its edges will be equal to $\eta N(N-1)$. However, if this number is not sufficient to obtain a connected graph, further edges are suitably added until the graph being generated becomes connected.

Three different values of the edge density η has been considered (0.01, 0.05 and 0.1) and 1000 pairs of isomorphic graphs of different size, ranging from 20 to 1000 have been generated for each value of η.

Regular meshes are introduced for simulating applications dealing with regular structures as those operating at the lower levels of a vision task. Furthermore, it is generally agreed that graphs having a regular structure represent a worst case for general graph matching algorithms (i.e. algorithms working on any kind of graphs) (Ullmann, 1976). This problem determined the born of specialized graph matching methods, with kind of algorithms able to efficiently perform the matching for given graph structures. So the database includes, as regular graphs, the mesh connected graphs (2D, 3D and 4D).

The considered 2D mesh are graphs in which each node (except those belonging to the border of the mesh) is connected with its 4 neighborhood nodes. Similarly, each node of a 3D and 4D graph has respectively connections with its 6 and 8 neighborhood nodes. In all cases the meshes are open, i.e. the border nodes are connected only with internal nodes.

In particular, 1000 pairs of isomorphic 2D meshes with size ranging from 16 to 1024 nodes have been generated, as well as 800 pairs of 3D meshes (with size ranging from 27 to 1000 nodes) and 500 pairs of 4D meshes (with size ranging from 16 to 1296 nodes).

Irregular mesh-connected graphs have been introduced for simulating the behavior of the algorithms in presence of slightly distorted meshes.

They have been obtained from regular 2D meshes by the addition of a certain number of edges. Each added edge connects nodes that have been randomly determined according to a uniform distribution. The number of added branches is ρN, where ρ is a constant greater than 0. Note that, the closer ρ to 0 is, the more symmetric the graphs are.

Three values of ρ has been considered (0.2, 0.4 and 0.6) and 1000 pairs of 2D meshes of different size have been generated for each value of ρ, as well as 800 pairs of 3D meshes and 500 pairs of 4D meshes, giving rise to a total of 6900 pairs of isomorphic graphs.

Bounded valence graphs model those applications in which each object (i.e. a node) establish a fixed number of relations (edges) with other objects, not necessarily with those belonging to its neighborhood. They are graphs in which every node has a number of edges (among ongoing and outgoing) lower than a given threshold, called *valence*. A particular case occurs when the number of edges is equal for all the nodes; in this case the graph is commonly called fixed valence graph.

The database includes graphs with a fixed valence, that have been generated by inserting random edges (using an uniform distribution) with the constraint that the valence of a node cannot exceed a selected value; edge insertion continues until all the nodes reach the desired valence. It is worth noting that it is impossible to have fixed valence graphs with an odd number of nodes and an odd valence, but in our database we have only considered graphs with an even number of nodes.

Three different values of the valence v (3, 6 and 9) have been considered and 1000 pairs of graphs of different size have been generated for each value of v. Again, sizes ranging from 20 to 1000 nodes have been considered.

In order to introduce some irregularities in the bounded valence graphs, we have considered also the *Irregular bounded valence graphs*. For such graphs, the average valence of the nodes (that is, the ratio between the number of edges and the number of nodes) is still bounded, but the single nodes may have a valence which is quite different from the average, and which is not bounded by a constant value. To this aim, first a fixed valence graph is generated. Then, a certain number of edges are moved from the nodes they are attached to, to other nodes. The number of movements is equal to $M = 0.1NV$, where V is the valence. This is equivalent to say that 10% of all the edges are moved.

The edges to be moved are chosen according to a random distribution with uniform probability. However, the new endpoints to which these edges are connected are not chosen uniformly, since this choice would affect only very slightly the overall variance of the valence of the nodes. Instead, after a random permutation of the nodes, the moved edges are distributed among the nodes using a probability distribution in which the node whose index is i has a probability of receiving an edge evaluated as $\alpha\exp(-\beta i)$ where α and β depend on the number N of nodes, and satisfy the following constraints: (i) the sum of the probabilities of the nodes of the graph is equal to 1 and (ii) the probability of the node i multiplied by the number of edges to be moved is equal to $0.5\sqrt{N}$. Using this distribution the maximum valence of the resulting graph will not be independent of N, and so special-purpose algorithms for bounded valence graphs cannot be employed, even though the graph is isomorphic for 90% of its edges to a fixed valence graph.

As in case of bounded valence graphs, three values of v $(3, 6$ and $9)$ have been considered and 1000 pairs of graphs of different size (from 20 to 1000 nodes) have been generated for each value of v.

3. A brief review of algorithms for graph isomorphism

During the last decades significant research efforts have been devoted to improve performance of the graph matching algorithms, in terms of both computational time and memory requirements.

Some algorithms reduce the computational complexity of the matching process by imposing topological restrictions on the graphs (Hopcroft and Wong, 1974; Luks, 1982; Jiang and Bunke, 1996). An alternative approach is that of using an adequate representation of the searching process and pruning unprofitable paths in the search space. In this way, no restrictions must be imposed on the structure of the input graphs and the obtained algorithms can be applied in more general cases.

A procedure that significantly reduces the size of the search space is the backtracking algorithm proposed by Ullmann (1976). This algorithm is devised for both graph isomorphism and subgraph isomorphism and is still today one of the most commonly used for exact graph matching. This is confirmed by the fact that in (Messmer, 1996) it is compared with other algorithms and it results the more convenient in terms of matching time in case of one-to-one matching.

Another backtracking algorithm is the one presented by Schmidt and Druffel (1976). It uses the information contained in the distance matrix representation of a graph to reduce the search tree of possible mappings.

A more recent algorithm, known as VF, is based on a depth-first search strategy, with a set of rules to efficiently prune the search tree. Such rules in case of isomorphism are shown in (Cordella et al., 1998). An improved version, referred to as VF2, is based on the same rationale, but stores the information of the state space search in more effective data structures, so as to significantly reduce the matching time and the memory requirements. Details on these aspects are given in (Cordella et al., 2001).

As regards the graph isomorphism problem, it is also necessary to mention the McKay's Nauty algorithm (McKay, 1981). It is based on a set of transformations that reduce the graphs to be matched to a canonical form on which the testing of the isomorphism is significantly faster. Even if Nauty is considered one of the fastest graph isomorphism algorithms today available, it has been shown that there are categories of graphs for which it requires exponential time (Miyazaki, 1997).

Another possible approach to the isomorphism problem is the one presented in (Bunke and Messmer, 1995); instead of reducing the complexity of matching two graphs, the authors attempt to reduce the overall computational cost when matching a sample graph against a large set of prototypes. The method performs the matching in quadratic time with the size of the input graph and independently on the number of prototypes. It is obviously convenient in applications requiring the matching of a graph against a database, but the memory required to store the pre-processed database grows exponentially with the size of the graphs, making the method suitable only for small graphs. So one of the authors concludes in (Messmer, 1996) that in case of one-to-one matching other algorithms (in particular, in (Messmer, 1996) the Ullmann's one is cited) are more suitable.

All the above-cited algorithms are exact ones; i.e. they find the exact solution to the matching problem, if any. Besides them, other techniques (as those based on non-deterministic paradigms: Christmas et al., 1995; De Jong and Spears, 1989), able to find approximate solutions to the matching problem have been proposed, especially in the recent past. We do not explicitly consider them in this paper, since they are really so powerful to reduce the complexity (in most cases) from exponential to polynomial, but, as said, they do not guarantee finding an exact and optimal solution.

4. Experimental results

The algorithms we have chosen for our experimentation are the Ullmann's algorithm, the algorithm by Schmidt and Druffel (in the following referred to as SD), the VF2 algorithm and Nauty.

These algorithms possess several common characteristics. They are exact isomorphism algorithms (although two of them can also be used, with small modifications, for graph–sub-graph isomorphism). They are always able to find the optimal solution, if it exists. They are general, in the sense that they do not rely on any special property of the graphs. They are optimized for one-to-one matching, as opposed to one-to-many matching (as is the case for the algorithms by Messmer and Bunke: Messmer, 1996; Messmer and Bunke, 1999). Three of them also show some similarity at the implementation level, being based on a depth-first search. While the fourth algorithm (Nauty) has a quite different implementation structure, it has been included in the experimentation because it is widely considered one of the fastest isomorphism algorithms available today.

We used, when available, the original implementations of the algorithms and run them on an Intel Celeron 500 MHz PC, equipped with 128 MB of RAM. So, as regards the Nauty algorithm, we used the version 2.0b9 made available by B.D. McKay at the http://cs.anu.edu.au/~bdm/nauty. A publicly available implementation of the Ullman's algorithm (Ullmann, 1976) can be found at the URL http://ftp.iam.unibe.ch/pub/Tools/GUB_toolkit.tar.Z; anyway, we have rewritten it in C++ with the aim of improving its efficiency. Thus, in this paper we refer to our implementation of the Ullman's algorithm, available at site http://amalfi.dis.unina.it/graph together with the VF2 algorithm and an implementation of the SD algorithm (Schmidt and Druffel, 1976).

Since not all the selected algorithms can solve the graph sub-graph isomorphism problem, in all tests only isomorphic pairs have been used. Moreover, two different evaluation criteria have been chosen. The first one measures the time needed by each algorithm for finding a solution to the isomorphism problem, while the second one considers pairs of non-isomorphic graphs and takes into account how long each algorithm needs for finding out that the two input graphs do not match.

In the following of the section plots giving the times as a function of the input graphs size are shown. Results on all the kinds of graphs present in the database are reported, but the ones on the 3D and 4D meshes, since in these cases the behavior of the algorithms is very similar to that exhibited on the 2D meshes. Note that for each kind of graphs both matching times and the times needed for finding out that the input graphs do not match are shown.

Times are always reported in seconds in a logarithmic scale. It is worth noting that some curves do not report the times obtained in correspondence with a given size. It happens when the algorithm is not able to give us a result (i.e. to find a solution to the isomorphism problem or to recognize that the input graphs are not isomorphic) in less than half an hour.

4.1. Randomly connected graphs

Fig. 1 shows the behavior of the four selected algorithms with reference to the *randomly connected graphs*. In particular, Fig. 1a–c respectively refer to values of η equal to 0.01, 0.5 and 0.1 for pairs of isomorphic graphs, while Fig. 1d–f refer to the same values of η for pairs of non-isomorphic graphs. As regards the matching times in case of isomorphic pairs, VF2 and Nauty perform always better with respect to SD and Ullmann. Moreover, Ullmann is faster than SD if the size of the graphs is smaller than 700. After this size, in fact, the Ullmann's algorithm is not able to find any isomorphism. As regards the comparison between VF2 and Nauty, it can be noted that VF2 obtains better performance for graphs of small size and for quite sparse graphs, while for dense graphs the Nauty algorithm exhibits better results. On the contrary, in case of non-isomorphic graphs the Ullmann's algorithm is always the best one. Moreover, VF2 always performs better than Nauty. Finally, it is worth noting that the performance of the SD algorithm only depends on the size of the input graphs.

4.2. 2D meshes

In Fig. 2 the performance of the algorithms on *2D regular meshes* are shown. The behavior of all algorithms is practically the same either for isomorphic or for non-isomorphic pairs. In both cases, in fact, as the size of the graphs grows up to 100 nodes, i.e. for graphs of medium size, neither Nauty nor Ullmann's are able to find solutions. For any input graph size, the VF2 algorithm is the best one and its performance are significantly better in case of non-isomorphic pairs.

Fig. 3 reports the performance on irregular *2D meshes*. In particular, in Fig. 3a–c pairs of isomorphic 2D meshes with values of ρ equal respectively to 0.2, 0.4 and 0.6 are considered, while Fig. 3d–f refer to pairs of non-isomorphic 2D meshes with the same values of ρ.

For pairs of isomorphic graphs, the main difference with respect to the case of regular meshes is that the Nauty algorithm is now always able to find a solution, but it still performs worse than VF2. However its behavior is better than those of SD and Ullmann are. Moreover, the maximum graph size for which the Ullmann's algorithm is still able to find a solution grows proportionally to the irregularity of the graphs. For a value of ρ equal to 0.6, Ullmann can solve the isomorphism problem for graphs whose size is up to 500 nodes. If it finds a solution the matching time is always better than the one obtained by SD.

Even for non-isomorphic graphs the Nauty algorithm is always able to give a result within the half an hour timeout, but in this case the difference with respect to the time needed by VF2 is higher. Also the Ullmann's algorithm always finds out the

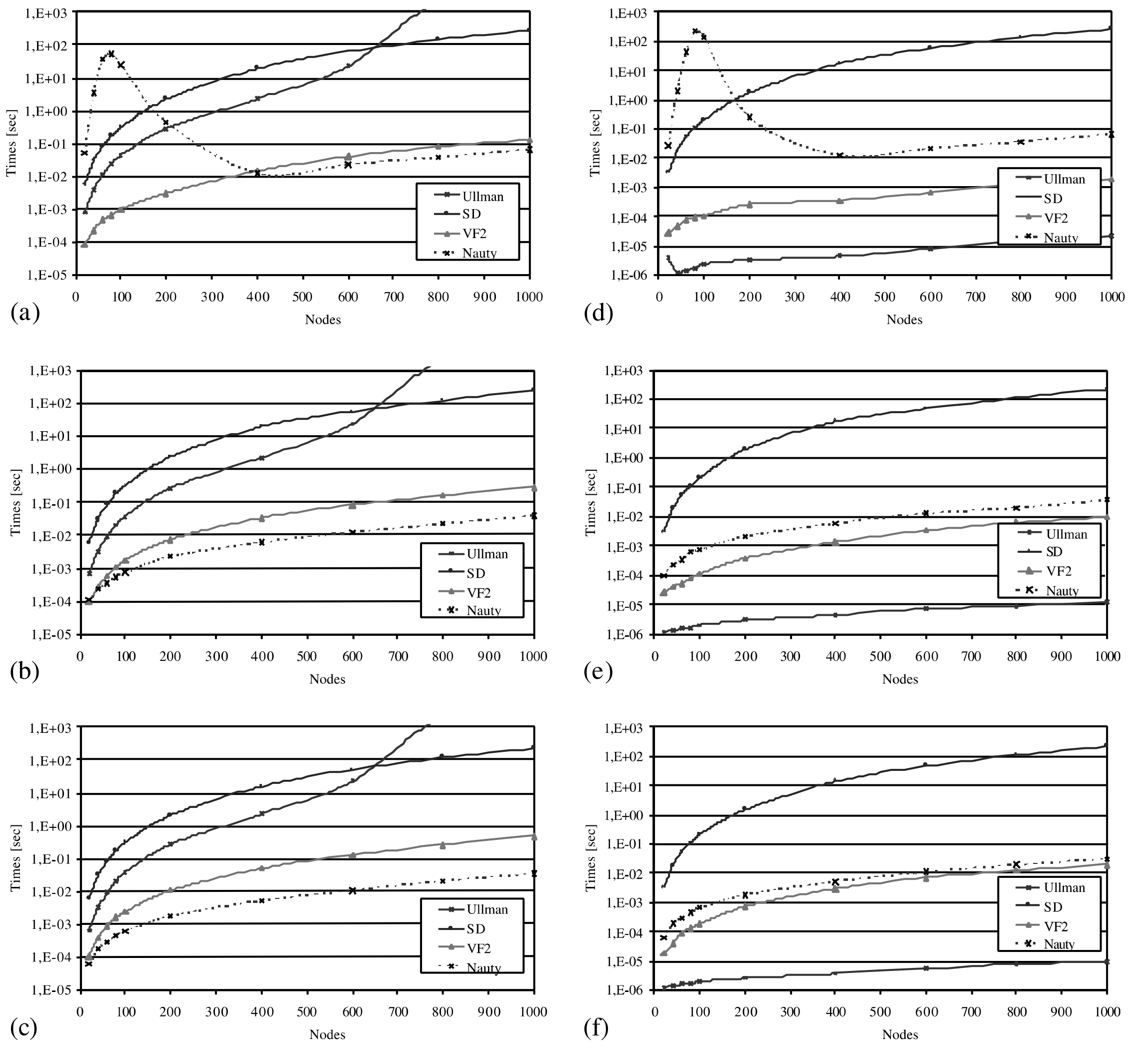

Fig. 1. The performance of the algorithms on *randomly connected graphs*, as a function of the graph size and for different values of η: (a,d) $\eta = 0.01$, (b,e) $\eta = 0.05$, (c,f) $\eta = 0.1$. Left column plots refer to pairs of isomorphic graphs, while right column plots to pairs of non-isomorphic graphs.

correct result within the timeout, but it is characterized by an oscillatory behavior. This is due to the fact that there are very few pairs of graphs for which the time needed to give a result is some orders of magnitude higher than the time needed by all the others pairs. So, it is very difficult to predict the behavior of such an algorithm as a function of the number of nodes and/or the values of ρ. This is particularly true for small values of ρ.

4.3. Bounded valence graphs

In Fig. 4 the performance of the algorithms on *bounded valence graphs* are shown. In this case the considered values of the valence v are 3, 6 and 9, as respectively reported in Fig. 4a–c for pairs of isomorphic graphs and Fig. 4d–f for pairs of non-isomorphic graphs.

In case of isomorphic pairs, the Ullmann's algorithm is not always able to find a solution; if it

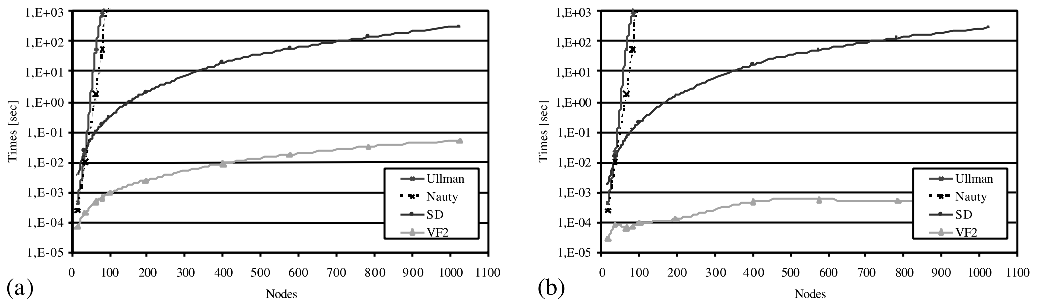

Fig. 2. The performance of the algorithms on *regular* 2*D meshes* as a function of the graph size in case of pairs of isomorphic graphs (a) and of non-isomorphic graphs (b).

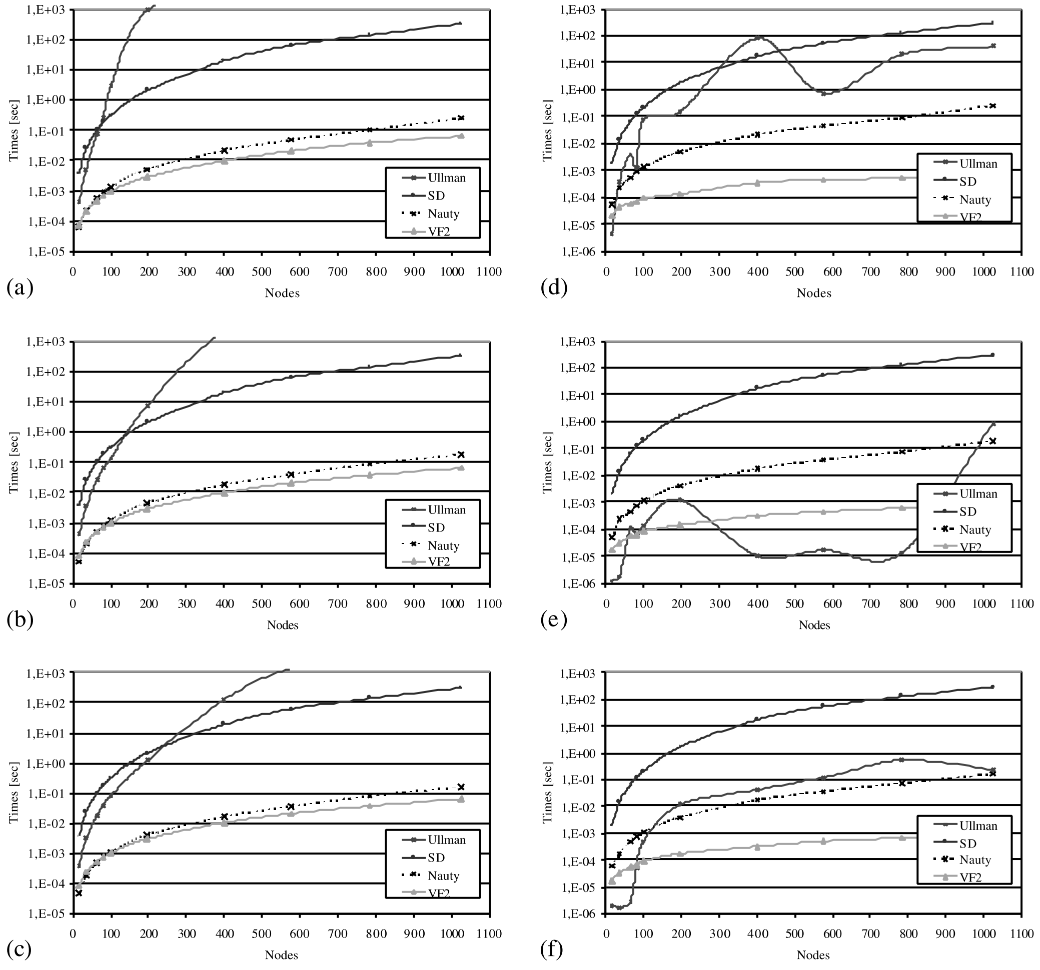

Fig. 3. The performance of the algorithms *irregular* 2*D meshes* as a function of the graph size and for different values of ρ: (a,d) $\rho = 0.2$, (b,e) $\rho = 0.4$, (c,f) $\rho = 0.6$. Left column plots refer to pairs of isomorphic graphs, while right column plots to pairs of non-isomorphic graphs.

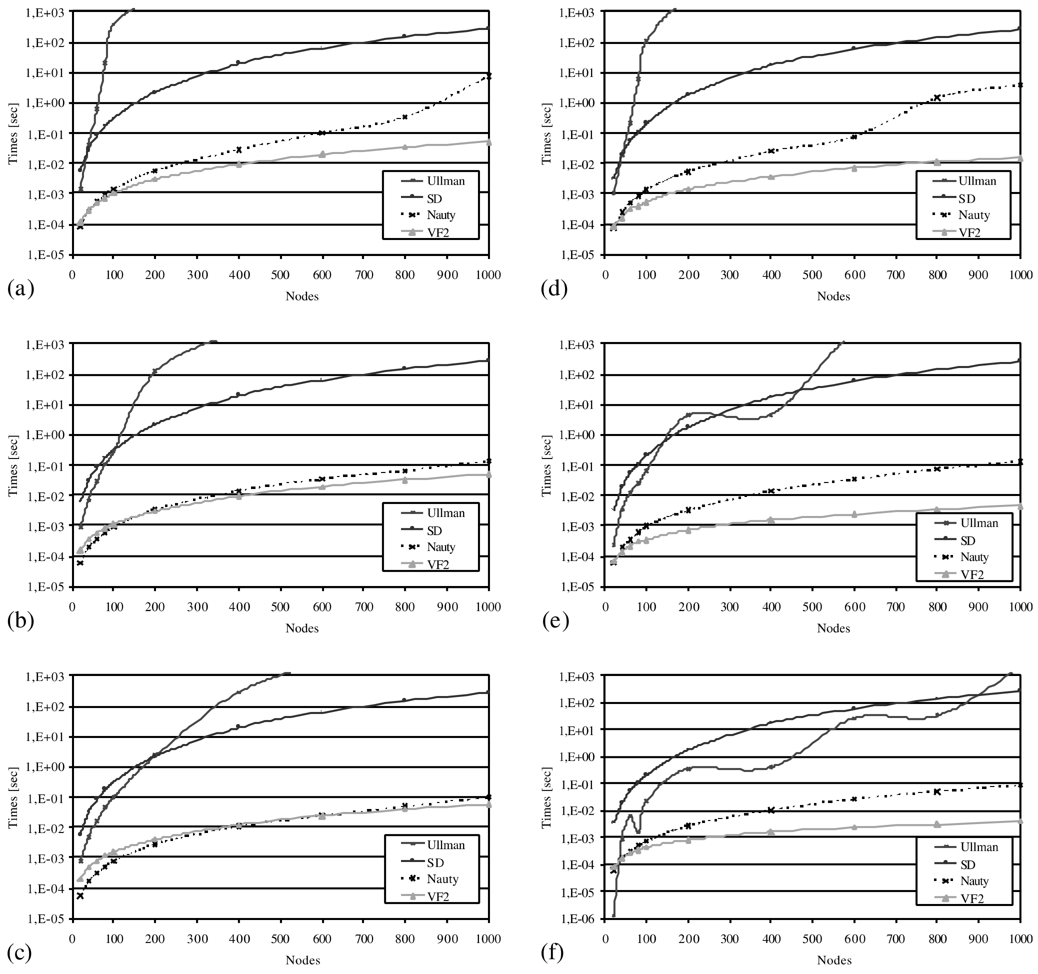

Fig. 4. The performance of the algorithms on *bounded valence graphs* as a function of the graph size and for different values of the valence v: (a,d) $v = 3$, (b,e) $v = 6$, (c,f) $v = 9$. Left column plots refer to pairs of isomorphic graphs, while right column plots to pairs of non-isomorphic graphs.

happens, however, its matching time is typically smaller than the one of SD and always higher than those obtained by the other two algorithms. Moreover, SD performs always worse than VF2 and Nauty.

As regards the comparison between VF2 and Nauty, the first algorithm performs always better for a value of v equal to 3, while for higher values of v the matching times obtained by Nauty for quite small graph are smaller than those obtained by VF2. Anyway, independently of the considered value of v, as the size of the input graphs is bigger than 600, VF2 is the best algorithm.

On the contrary, in case of non-isomorphic graphs VF2 always performs better than Nauty. For high values of v Ullmann again exhibits an oscillatory behavior, but its performance is practically always worse than that of VF2. Even in this case the performance of the SD algorithm only depends on the size of the input graphs.

4.4. Irregular bounded valence graphs

Finally, in Fig. 5 the performance of the algorithms on *irregular bounded valence graphs* is shown. Also in this case the considered values of the valence v are 3, 6 and 9, as respectively reported in Fig. 5a–c for pairs of isomorphic graphs and Fig. 5d–f for pairs of non-isomorphic graphs.

In case of isomorphic graphs, VF2 is the best algorithm for low values of v, while its performance is practically the same of that exhibited by Nauty for higher values of v. However, independently on the values of v, VF2 has the best performance as the size of the input graphs grows. On the contrary, the Ullmann's algorithm is never able to find a solution if the size of the input graphs becomes higher than 700.

As regards non-isomorphic graphs, both SD and Nauty exhibit the same behavior of the isomorphic graph case, while VF2 gives rise to better results, performing always significantly better than Nauty. For all values of v, however, the best algorithm is Ullmann, which is able to find out the correct result in a time that is about two orders of magnitude lower than the one needed by VF2.

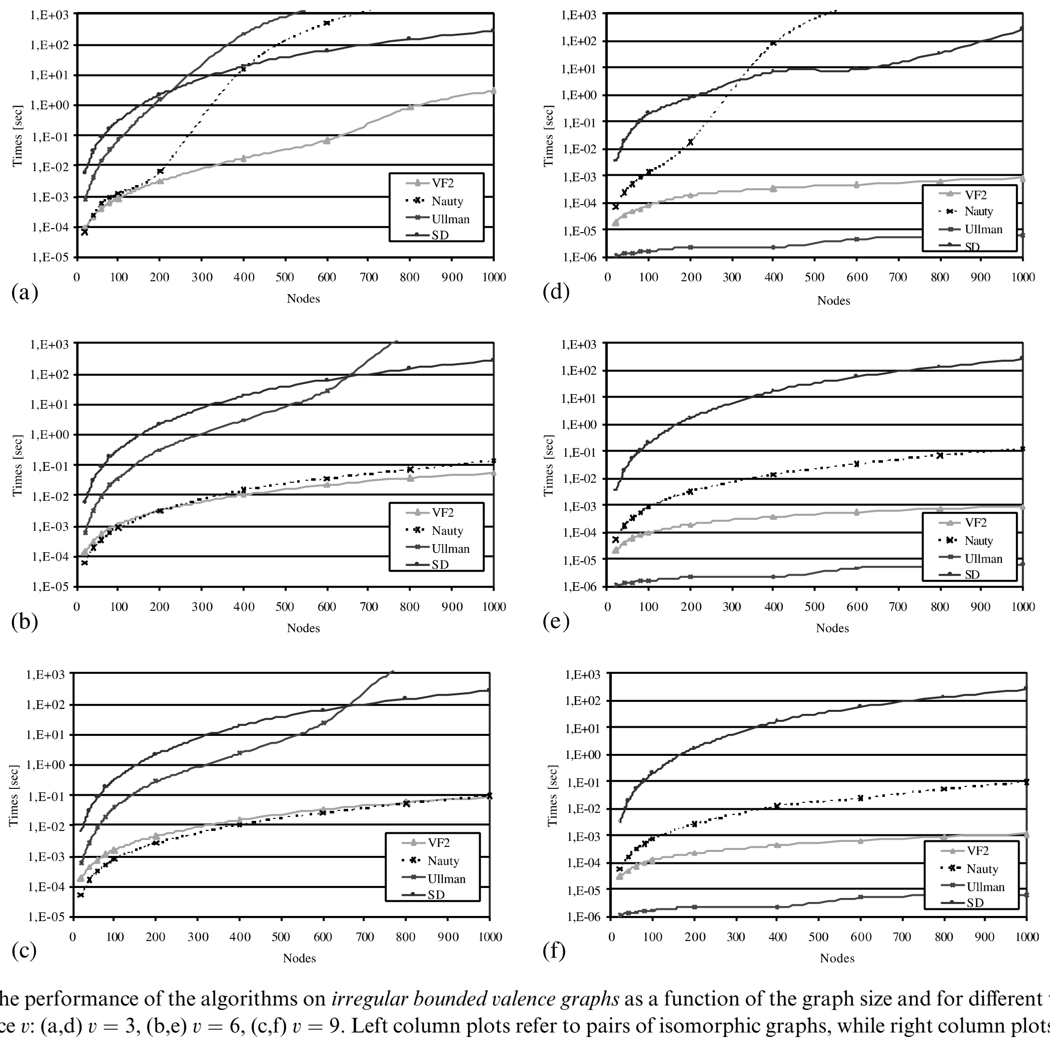

Fig. 5. The performance of the algorithms on *irregular bounded valence graphs* as a function of the graph size and for different values of the valence v: (a,d) $v = 3$, (b,e) $v = 6$, (c,f) $v = 9$. Left column plots refer to pairs of isomorphic graphs, while right column plots to pairs of non-isomorphic graphs.

5. Discussion and conclusions

In this paper we have presented a benchmarking activity for assessing the performance of some widely used exact graph matching algorithms. The comparison has been carried out on a large database of artificially generated graphs, which has been built and made publicly available to provide a common reference data set for further benchmarking activities.

As it could be expected, it does not exist an algorithm that is definitively better than all the others. In particular, for randomly connected graphs, the Nauty algorithm is the best one if the graphs are quite dense and/or of quite large size. For smaller and quite sparse graphs, on the contrary, VF2 performs better.

On more regular graphs, i.e. on 2D meshes, VF2 is definitely the best algorithm: in this case the Nauty algorithm is often not even able to find a solution for graphs bigger than few dozens of nodes. In case of bounded valence graph, if the valence is small, VF2 is always the best algorithm, while for bigger values of the valence the Nauty algorithm is more convenient if the size of the graphs is small.

If the two graphs being compared are not isomorphic, Ullmann's algorithm shows a quite singular behavior: for most of the cases, it is able to stop the search very early, requiring a time that is considerably lower than that of the other algorithms. On the other hand, in the remaining few cases, the time needed to stop the search is extremely longer (often by several orders of magnitude), making the algorithm unpractical. We have not yet been able to characterize the graphs which trigger such a behavior, nor to provide a reasonable estimate of their occurrence probability. It can be noted that for these cases, the algorithm showing the best performance is definitely VF2. Finally, it is also worth noting that SD and VF2 are the only algorithms that have always been able to find a solution to the isomorphism problem in our tests, independently of the size and the kind of the graphs to be matched.

Future steps of this benchmarking activity will involve the comparison of other algorithms, and the extension to other problems. We are also planning to extend the database with other graph categories and to add an indexing facility (based on several graph parameters), for making its use more easy and convenient to other researchers.

References

Abdulrahim, M., Misra, M., 1998. A graph isomorphism algorithm for object recognition. Pattern Anal. Appl. 1 (3), 189–201.

Bunke, H., Messmer, B.T., 1995. Efficient attributed graph matching and its application to image analysis. In: Braccini, C., De Floriani, L., Vernazza, G. (Eds.), Image Analysis and Processing. Springer, Berlin, pp. 45–55.

Bunke, H., Vento, M. 1999. Benchmarking of graph matching algorithms. In: Proceedings of the 2nd Workshop on Graph-based Representations. Haindorf, pp. 109–114.

Christmas, W.J., Kittler, J., Petrou, M., 1995. Structural matching in computer vision using probabilistic relaxation. IEEE Trans. Pattern Anal. Mach. Intell. 17 (8), 749–764.

Cordella, L.P., Foggia, P., Sansone, C., Vento, M., 1998. Subgraph Transformations for the inexact matching of attributed relational graphs. Comput. Suppl. 12, 43–52.

Cordella, L.P., Foggia, P., Sansone, C., Vento, M., 1999. Performance evaluation of the VF graph matching algorithm. In: Proceedings of the 10th ICIAP, IEEE Computer Society Press, pp. 1172–1177.

Cordella, L.P., Foggia, P., Sansone, C., Vento, M., 2001. An improved algorithm for matching large graphs. In: Proceedings of the 3rd IAPR-TC-15 International Workshop on Graph-based Representations. Italy, pp. 149–159.

De Jong, K.A., Spears, W.M., 1989. Using genetic algorithms to solve NP-complete problems. In: Schaffer, J.D. (Ed.), Genetic Algorithms. Morgan Kaufmann, Los Altos, CA, pp. 124–132.

Hopcroft, J., Wong, J., 1974. Linear time algorithm for isomorphism of planar graphs. In: Proceedings of the 6th Annual ACM Symposium on Theory of Computing, pp. 172–184.

Jiang, X., Bunke, H., 1996. Including geometry in graph representations: a quadratic time isomorphism algorithm and its applications. In: Perner, P., Wang, P., Rosenfeld, A. (Eds.), Lecture Notes in Computer Science, 1121, pp. 110–119.

Luks, E.M., 1982. Isomorphism of graphs of bounded valence can be tested in polynomial time. J. Comput. Syst. Sci., 42–65.

McKay, B.D., 1981. Practical graph isomorphism. Congressus Numerantium 30, 45–87.

Messmer, B.T., 1996. Efficient Graph Matching Algorithms for Preprocessed Model Graphs, Ph.D. Thesis, Institute of Computer Science and Applied Mathematics, University of Bern.

Messmer, B.T., Bunke, H., 1999. A decision tree approach to graph and subgraph isomorphism detection. Pattern Recogn. 32, 1979–1998.

Miyazaki, T., 1997. The complexity of McKay's canonical labeling algorithm. In: Finkelstein, L., Kantor, W.M. (Eds.), Groups and Computation, II. American Mathematical Society, Providence, RI, pp. 239–256.

Rouvray, D.H., Balaban, A.T., 1979. Chemical applications of graph theory. In: Applications of Graph Theory. Academic Press, New York, pp. 177–221.

Schmidt, D.C., Druffel, L.E., 1976. A fast backtracking algorithm to test directed graphs for isomorphism using distance matrices. J. ACM 23, 433–445.

Shearer, K., Bunke, H., Venkatesh, S., Kieronska, D., 1998. Graph matching for video indexing. Comput. Suppl. 12, 53–62.

Ullmann, J.R., 1976. An algorithm for subgraph isomorphism. J. Assoc. Comput. Mach. 23, 31–42.

Wong, E.K., 1992. Model matching in robot vision by subgraph isomorphism. Pattern Recogn. 25 (3), 287–304.