
Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 11, no. 1, pp. 99–143 (2007)

Challenging Complexity of Maximum Common Subgraph Detection Algorithms: A Performance Analysis of Three Algorithms on a Wide Database of Graphs

Donatello Conte

Dipartimento di Ingegneria dell'Informazione ed Ingegneria Elettrica
Università di Salerno, via P.te Don Melillo, I-84084 Fisciano (SA), Italy
dconte@unisa.it

Pasquale Foggia

Dipartimento di Informatica e Sistemistica
Università di Napoli "Federico II", Via Claudio, 21 I-80125 Napoli (Italy)
foggiapa@unina.it

Mario Vento

Dipartimento di Ingegneria dell'Informazione ed Ingegneria Elettrica
Università di Salerno, via P.te Don Melillo, I-84084 Fisciano (SA), Italy
mvento@unisa.it

Abstract

Graphs are an extremely general and powerful data structure. In pattern recognition and computer vision, graphs are used to represent patterns to be recognized or classified. Detection of maximum common subgraph (MCS) is useful for matching, comparing and evaluate the similarity of patterns. MCS is a well known NP-complete problem for which optimal and suboptimal algorithms are known from the literature. Nevertheless, until now no effort has been done for characterizing their performance. 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, three optimal and well-known algorithms for maximum common subgraph detection are described. Moreover a large database containing various categories of pairs of graphs (e.g. random graphs, meshes, bounded valence graphs), is presented, and the performance of the three algorithms is evaluated on this database.

Article Type	Communicated by	Submitted	Revised
Regular Paper	U. Brandes	September 2005	January 2007

1 Introduction

Graphs are a powerful and versatile tool that is used in various subfields of science and engineering. There are several applications, for example, in pattern recognition [12, 13, 22, 28, 29, 35, 37], machine learning [21], computer vision [14], image and video analysis [9, 11, 24, 26, 38] and information retrieval [19], where there is the need to measure the similarity between objects. If graphs are used for the representation of structured objects, then matching and comparing objects becomes equivalent to determining the similarity between graphs ([1]).

There are several well known relations between graphs that are a suitable basis for defining graph similarity measures. Graph isomorphism is useful to find out if two graphs have identical structure [39]. More generally, subgraph isomorphism (i.e. an isomorphism between a graph and a subgraph of another graph) can be used to check if one graph is part of another [39, 18]. In two recent papers [7, 8], graph similarity measures based on *maximum common subgraph* and *minimum common supergraph* have been proposed, verifying if two graphs share a common part.

Detection of the maximum common subgraph of two given graphs is a well-known problem. In [27], an algorithm for solving this problem is described and in [15, 36] the use of this algorithm for comparing molecular structurs has been discussed. In [32] a maximum common subgraph algorithm that uses a backtrack search strategy is introduced. Other algorithms adopt a different strategy for deriving the maximum common subgraph, first obtaining the association graph of the two given graphs and then detecting its *maximum clique* [2, 5, 20, 33].

It is well known that both maximum common subgraph and maximum clique detection are NP-complete problems [25]. Therefore many approximate algorithms have been developed. A survey of such approximate algorithms, including an analysis of their complexity and potential applications is provided in [4].

Although a significant number of maximum common subgraph detection algorithms have been proposed in the literature, until now no effort has been spent for characterizing their performance: the authors of each novel algorithm usually provide experimental results supporting the claim that, under suitable hypotheses, their method can outperform the previous ones. Nevertheless it is almost always very difficult for an 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 an extensive comparison of graph matching algorithms in terms of key performance indices (memory and time requirements, maximum graph size, etc.) [10, 17]. 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 the application constraints. There is little or no information on how the behavior of these algorithms varies as the type and the size of the graphs to be matched change from an application

to another.

The first type of comparison that can be easily performed between different algorithms is a theoretical comparison. In fact it is possible to estimate the computational complexity of each algorithm in the worst case and in the best case. A common problem is that users of graph matching algorithms have to choose an algorithm, in a group of algorithms, that best fits their problem. In many cases a real problem can be suitably represented using one or more categories of graphs having known parameters (e.g. number of nodes, edge density); thus a description of an algorithm through its the behavior in the best or the worst case could be insufficient.

We can suppose, for instance, that algorithms A_1 and A_2 are available to solve a given graph matching problem, and that the algorithm A_1 is faster of the algorithm A_2 in the best case and in the worst case. Is this characterization enough to prefer the algorithm A_1 to the other? Of course it is not. The user needs more details on the behavior of the two algorithms, to choose the best one. In particular the information that the user needs is: which algorithm performs better on those graphs describing his problem? The answer of this question is not simple at all.

Firstly a more detailed theoretical analysis should be performed. Since the information concerning the complexity in the best and in the worst case is not sufficient for comparing algorithms, another parameter that can be used is the computational complexity in the *average case*. Indeed, even if the computational complexity in the worst case of the algorithm A_2 is higher than the complexity of A_1, the average computational complexity for A_2 may happen to be lower than the one of A_1.

Unfortunately the average case complexity can be analytically determined for simple algorithms, but this may prove an impossible task for several algorithms solving graph matching problems. The only possibility is to perform a wide experimental comparison of different graph matching algorithms, for measuring their performance on a large graph database containing many categories of graphs.

Moreover the comparison between different graph matching algorithms is a very important task because in general it is impossible to find the 'best' algorithm: it is just possible to find an algorithm that performs better on a restricted category of graphs, but till now no effort has been spent to establish which algorithm is more convenient on each category of graphs, probably because of the lack of standard databases of graphs specifically designed for this purpose.

In other research fields (for example, OCR), the availability of large defacto standard databases improves the verifiability and the comparability of the experimental results of each method: thus our aim is to provide a standard database of graphs also for graph matching problems.

The creation of a graph database is definitely not a simple task, since several issues have to be taken into account. The first problem is to decide whether the graphs should be collected from real-world applications or they should be synthetically generated (as in [6]), according to some probabilistic model. The latter choice, besides being simpler to implement, permits a finer control over

the features of the graphs; the models for the synthetic generation of graphs have to be derived from the analysis of graphs in real applications.

In this paper we present a synthetically generated large database containing various categories of attributed graph, i.e. randomly connected graphs, 2D, 3D and 4D regular and irregular meshes, regular and irregular bounded valence graphs; for each category we have generated pairs of graphs having a known maximum common subgraph.

Graphs used in pattern recognition applications have usually attributes on nodes and edges, due to the fact that graphs are used to represent the structural information of the patterns and nodes and edges attributes are used to store the quantitative information of the single parts of each pattern and of their interconnections. Thus, to make the proposed database more useful in the pattern recognition community, attributes on nodes and edges are introduced. The main problem is that attributes are strongly application dependent, but our aim is to realize a database that can be used to test graph matching algorithms, apart from their application domain.

MCS algorithms can be grouped in many categories: optimal graph matching algorithms are more robust, but also considerably slower than suboptimal ones. Suboptimal algorithms can be quite faster, but may fail in finding a solution even if it exists. Some algorithm 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 three optimal algorithms are described and used for the purposes of the benchmarking activity.

The first algorithm searches for the maximum common subgraph by finding all common subgraphs of the two given graphs and choosing the largest [32]; the second algorithm builds the association graph between the two given graphs and then searches for the maximum clique of the latter graph [20]. The third algorithm also searches for the maximum clique, but uses more sophisticated graph theory concepts for determining upper and lower bounds during the search process.

We have chosen the most representative algorithms between those present in scientific literature. As we show in [16] the maximum common subgraph is an exact matching problem and it is solved, mainly, by techniques based on tree search. The chosen algorithms are widely used and many other algorithms, also based on tree search, can be considered as derived from them.

The remainder of the paper is organized as follows. In Section 2, basic terminology and concepts will be introduced. Next, in Section 3 the three algorithms for maximum common subgraph detection will be described , while in Section 4 the database of graphs is presented. Experimental results are reported in Section 5. Finally, future work is discussed and some conclusions are drawn in Section 6.

2 Basic Definitions

Let L denote a finite set of labels for nodes and edges.

Definition 1 *A* graph *is a 4-tuple $g = (V, E, \alpha, \beta)$, where*

- *V is the finite set of vertices (also called nodes)*
- *$E \subseteq V \times V$ is the set of edges*
- *$\alpha : V \to L$ is a function assigning labels to the vertices*
- *$\beta : E \to L$ is a function assigning labels to the edges*

Edge (u, v) originates at node u and terminates at node v. An undirected graph is obtained as a special case if there exists an edge $(v, u) \in E$ for any edge $(u, v) \in E$ with $\beta(u, v) = \beta(v, u)$. Node and edge labels come from the same alphabet, for notational convenience.

Definition 2 *Let $g = (V, E, \alpha, \beta)$ and $g' = (V', E', \alpha', \beta')$, be graphs; g' is an* induced subgraph *of g, $g' \subseteq g$, if*

- *$V' \subseteq V$*
- *$\alpha(v) = \alpha'(v)$ for all $v \in V'$*
- *$E' = E \cap (V' \times V')$*
- *$\beta(e) = \beta'(e)$ for all $e \in E'$*

From Definition 2 it follows that, given a graph $g = (V, E, \alpha, \beta)$, any subset $V' \subseteq V$ of its vertices uniquely defines a subgraph. This subgraph is called the subgraph *induced* by V'.

A matching process between two graphs g and g' consists in the determination of a mapping M which associates nodes of the graph g with nodes of g' and vice versa. As it is well known, different constraints can be imposed to M, and consequently different mapping types can be obtained: isomorphism, subgraph isomorphism and maximum common subgraph are the most frequently used.

Definition 3 *Let g and g' be graphs. A* graph isomorphism *between g and g' is a bijective mapping $f : V \to V'$ such that*

- *$\alpha(v) = \alpha'(f(v))$ for all $v \in V$*
- *for any edge $e = (u, v) \in E$ there exists an edge $e' = (f(u), f(v)) \in E'$ such that $\beta(e) = \beta'(e')$, and for any edge $e' = (u', v')$ there exists an edge $e = (f^{-1}(u), f^{-1}(v)) \in E$ such that $\beta(e) = \beta'(e')$*

If $f : V \to V'$ is a graph isomorphism between graphs g and g', and g' is an induced subgraph of another graph g'', i.e., $g' \subseteq g''$, then f is called a *subgraph isomorphism* from g to g''.

Definition 4 *Let $g_1 = (V_1, E_1, \alpha_1, \beta_1)$ and $g_2 = (V_2, E_2, \alpha_2, \beta_2)$ be graphs. A* common subgraph *of g_1 and g_2, $cs(g_1, g_2)$, is a graph $g = (V, E, \alpha, \beta)$ such that there exist subgraph isomorphisms from g to g_1 and from g to g_2. We call g a* maximum common subgraph *of g_1 and g_2, $mcs(g_1, g_2)$, if there exists no other common subgraph of g_1 and g_2 that has more nodes than g.*

Notice that, according to Definition 4, $mcs(g_1, g_2)$, is not necessarily unique for two given graphs; usually there exist more than one maximum common subgraph for two given graphs. We will call the set of all maximum common subgraphs of a pair of graphs their *MCS set*.

Example 1 A graphical representation of two graphs, g_1 and g_2, is given in Figure 1. For those graphs, we have:

- $V_1 = 1, 2, 3$; $V_2 = 4, 5, 6$; $L = a, b, c, 1, 2$
- $E_1 = (1, 2), (2, 1), (1, 3), (3, 1), (2, 3), (3, 2)$;
  $E_2 = (4, 5), (5, 4), (4, 6), (6, 4), (5, 6), (6, 5)$
- $\alpha_1 : 1 \rightarrow a, 2 \rightarrow b, 3 \rightarrow c$
- $\alpha_2 : 4 \rightarrow a, 5 \rightarrow b, 6 \rightarrow c$
- $\beta_1 : (1, 2) \rightarrow 1, (2, 1) \rightarrow 1, (1, 3) \rightarrow 1, (3, 1) \rightarrow 1, (2, 3) \rightarrow 1, (3, 2) \rightarrow 1$
- $\beta_2 : (4, 5) \rightarrow 2, (5, 4) \rightarrow 2, (4, 6) \rightarrow 1, (6, 4) \rightarrow 1, (5, 6) \rightarrow 1, (6, 5) \rightarrow 1$

There exist two maximum common subgraphs $g_3 = (V_3, E_3, \alpha_3, \beta_3)$ and $g_4 = (V_4, E_4, \alpha_4, \beta_4)$:

- $V_3 = 7, 8$; $V_4 = 9, 10$
- $E_3 = (7, 8), (8, 7)$; $E_4 = (9, 10), (10, 9)$
- $\alpha_3 : 7 \rightarrow a, 8 \rightarrow c$
- $\alpha_2 : 9 \rightarrow a, 10 \rightarrow c$
- $\beta_3 : (7, 8) \rightarrow 1, (8, 7) \rightarrow 1$
- $\beta_4 : (9, 10) \rightarrow 1, (10, 9) \rightarrow 1$

These graphs are also shown in Figure 1.

3 The Selected Maximum Common Subgraph Algorithms

In this section we will provide a description of the three algorithms that will be used for our experimental comparison. These algorithms are quite similar under several respects. They all belong to the category of exact, or optimal, matching algorithms, as opposed to approximate or suboptimal ones, in the

sense that they always find the correct MCS, and not an approximate solution to the problem. Since MCS is a NP-complete problem, their worst case time complexity is exponential (more precisely, factorial) with respect to the number of nodes in the graphs. Also, as we will see in the next subsections, their structure is quite similar: they perform a depth first search, with the help of some heuristic for pruning unfruitful search paths.

The differences among the three algorithms actually lie only in the information used to represent each state of the search space (that is reflected in their different space complexity), and in the kind of heuristic adopted.

The choice of three similar algorithms has been made for the purpose of enabling a more effective interpretation of the experimental results: by reducing to a minimum the possible causes of the measured performance diversity, it will be easier to find a convincing explanation.

3.1 McGregor Algorithm

This algorithm can be suitably described through a State Space Representation [34]. Each state `s` represents a common subgraph of the two graphs under construction. This common subgraph is part of the maximum common subgraph to be eventually formed. In each state a pair of nodes not yet analyzed `(n1,n2)`, the first belonging to the first graph and the second belonging to the second graph, is selected (whenever it exists) through the function `NextPair(s,n1,n2)`. The selected pair of nodes is analyzed through the function `IsFeasiblePair(s,n1,n2)` that checks whether it is possible to extend the common subgraph represented by the current state by means of this pair,

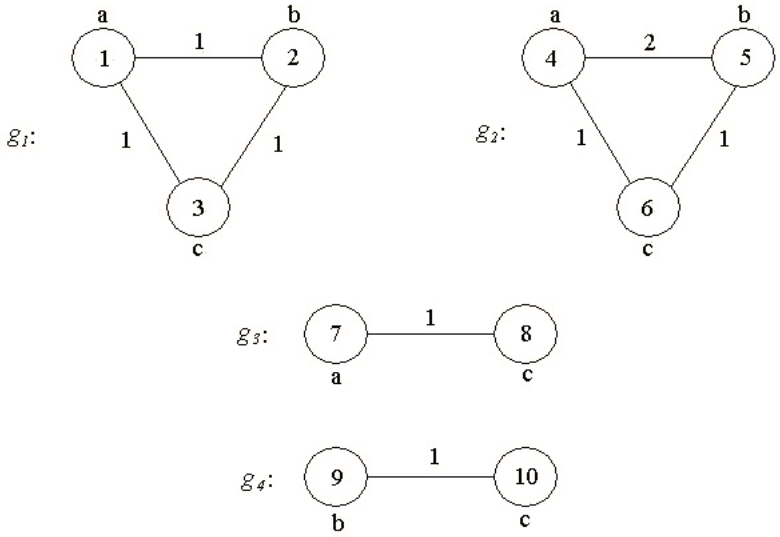


Figure 1: Two graphs and their maximum common subgraphs.

so obtaining a larger common subgraph. If the extension is possible, then the function `AddPair(n1,n2)` actually extends the current partial solution by the pair `(n1,n2)`. After that, if the current state `s` is not a leaf of the search tree, i.e. if there exists at least a node belonging to the first graph that hasn't yet been selected through the function `NextPair`, then this node is selected and the analysis of a new state is started. After the new state has been analyzed, a *backtrack* function is invoked, to restore the common subgraph of the previous state and to choose a different new state. Using this search strategy, whenever a branch of the search tree is chosen, it will be followed as deeply as possible until a leaf is reached, or until a pruning condition is verified. The algorithm stores the current level of the search tree; the value of this level is always less than or equal to the size of the smaller of the two starting graphs. The size of the maximum common subgraph is also less than or equal to the size of the smaller of the two starting graphs, thus the pruning condition checks whether the number of levels from the current one to the most distant leaf of the search tree is not enough to construct a common subgraph larger than the stored one. It is noteworthy that each branch of the search tree has to be followed, because - except for trivial examples - is not possible to foresee if a better solution exists in a branch that has not yet been explored. A special node, the *null_node*, i.e. a node that is compatible with any other node is also needed. Actually, after that a node `n1` is matched with all the nodes `n2`, it is finally matched with the node *null_node*. This process ensures the exploration of the whole search tree, avoiding that branches containing the best solution are cut before their complete exploration.

The first state is the *empty state*, in which no nodes have yet been matched. A pseudo-code description of McGregor algorithm is shown in Figure 2. An example of the McGregor algorithm application is sketched in Figure 3.

```
procedure McGregor_MCS(s)
begin
   while (NextPair(s,n1,n2))
      if (IsFeasiblePair(s,n1,n2)) then
         s' = AddPair(s,n1,n2);
         if (size(s') > CurrentMCSSize) then
            SaveCurrentMCS(s')
            CurrentMCSSize = size(s');
         end if
         if(!LeafOfSearchTree(s') and !PruningCondition(s')) then
            McGregor_MCS(s');
         end if
          BackTrack(s');
       end if
   end while
end procedure
```

Figure 2: A sketch of McGregor algorithm.

Let N_1 and N_2 be the number of nodes of the first and the second graph respectively, and let $N_1 \leq N_2$. In the worst case, i.e. when the two graphs are completely connected and the size of the alphabet of attributes is 1, the number

of states s examined by the algorithm is:

$$S = (N_2 + 1)(N_2) \cdot \ldots \cdot (N_2 - N_1 + 2) = \frac{(N_2 + 1)!}{(N_2 - N_1 + 1)!} \tag{1}$$

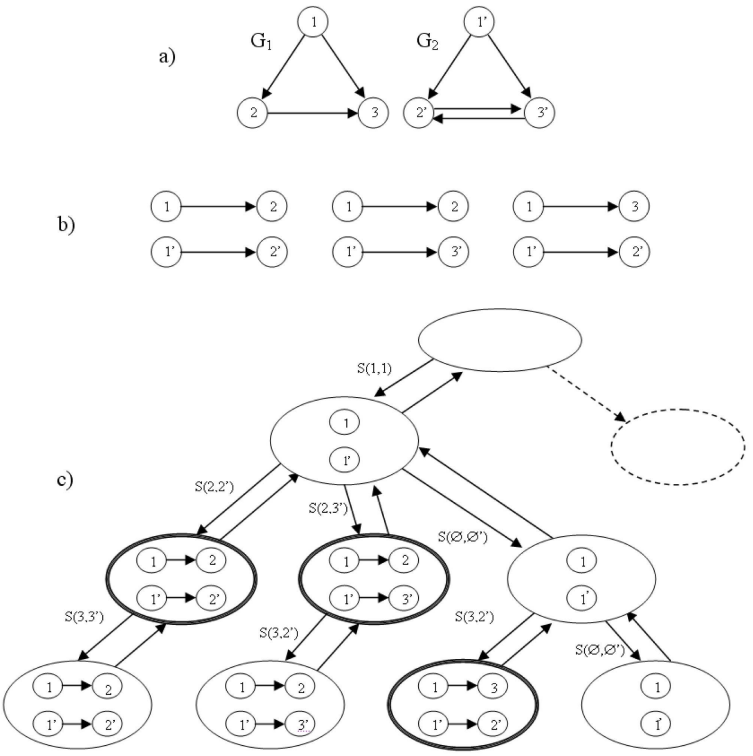


Figure 3: a) two directed graphs, G_1 and G_2; b) three maximum common subgraphs between G_1 and G_2; c) a part of the search tree explored by McGregor algorithm. In each state $S(\cdot, \cdot)$ a pair of nodes, the first belonging to the graph G_1, and the second belonging to the graph G_2 is selected and it is checked whether this pair of nodes can extend the current common subgraph. The states contained in a thick oval are those in which the current maximum common subgraph has been detected.

In this case the algorithm will explore (N_2+1) nodes at level 1, N_2 at level 2, (N_2-1) at level 3, up to (N_2-N_1+2) at level N_1. Multiplying these numbers, we obtain the number of states of the worst case.

For the case $N_1 = N_2 = N$ and $N >> 1$, eq. 1 can be approximated as follows:

$$S \cong e \cdot N \cdot N! \tag{2}$$

Notice that only $O(N_1)$ space is needed by this implementation of the algorithm, indeed only the states associated to the nodes of the branch currently in exploration need to be stored in memory.

A maximum common subgraph of two given graphs is defined in [32] as the common subgraph maximizing the number of edges; we could call it *edge induced MCS*, in contrast with the Definition 4 (*node induced MCS*), in which the maximum common subgraph maximizes the number of nodes. According to the node induced definition, a MCS graph, can be composed of smaller graphs unconnected with each other. Instead the case of a maximum common subgraph containing unconnected nodes is not considered in [32]. In fact in [32] a graph is used to represent a molecule, a node is used to represent an atom, and the graph matching algorithms serve the purpose of simulating chemical reactions. Thus, in McGregor's case, an isolated node has no meaning, because in chemical reactions it is usually impossible to create isolated atoms. The algorithm described in this section is used to find out the node induced MCS, consequently it is more general than the one introduced in [32].

3.2 Durand-Pasari Algorithm

The Durand-Pasari algorithm is based on the well known reduction of the search of the maximum common subgraph between two graphs to the problem of finding a maximal clique, i.e. the largest completely connected subgraph, in a graph [20]. The first step of the algorithm is the construction of the *association graph*, whose nodes correspond to pairs of nodes of the two starting graphs having the same attribute. The edges of the association graph (that are undirected) represent the compatibility of those pairs of nodes to be included. That is, a node corresponding to the pair `(n1,n2)` is connected to a node corresponding to `(m1,m2)` iff there is an isomorphism between the subgraph $\{n1, m1\}$ of the first graph and the subgraph $\{n2, m2\}$ of the second graph. This condition can be easily checked by looking at the edges between `n1` and `m1` and between `n2` and `m2` in the two starting graphs; node and edge attributes, if present, must also be taken into account. It can been easily demonstrated that each clique in the association graph corresponds to a common subgraph and vice versa; hence, the maximum common subgraph can be obtained by finding the maximal clique in the association graph.

The Durand-Pasari algorithm generates a list of nodes belonging to the current clique of the association graph, using a depth-first search strategy on a search tree, by systematically selecting one node at a time from successive levels of the search tree, until it is not possible to add further nodes to the list. A sketch of the algorithm is in Figure 4.

The function `NextNode(s,n)` looks for the nodes to be examined. The algorithm ends when there are no more nodes to be examined. At each level l of the tree search, the choice of the nodes in the association graph to be considered is limited to the ones which correspond to pairs `(n1,n2)` having `n1` $= l$: l is the number of the current level of the tree search and the condition `n1` $= l$ indicate that at each level we consider nodes in the associations graph correspondent to pairs that have one node of the first graph (in particular the l-th node) with all nodes in the second graph. In this way the algorithm ensures that the search space is actually a tree, i.e. it will never consider twice the same list of nodes. After considering all the nodes for level l, a special node, called the *null_node*, is added to the list. This node can be added more than once to the list. This special node is used to carry the information that no mapping is associated to a particular node of the first graph being matched.

When a node is being considered, the forward search part of the algorithm, first checks to prove whether this node is a *legal* node (with the function `IsLegalNode(s,n)`). A node is legal if it is connected to every other node already in the clique. In [20] if a node is legal the algorithm continues with the next level of the search tree. That is, the original algorithm examines any possible clique of the association graph. In our implementation if the node is legal, the algorithm checks if the size of the new clique is as large or larger than the current largest clique, in which case it is saved and, only in this case, the algorithm continues with the next level. This check is performed by `pruningCondition(s)`. With the pruning condition the algorithm examines only the promising branch. The new state is built with the addition of the new node (with the function `AddNode(s,n)`).

When all possible nodes (including the *null_node*) have been considered, the algorithm backtracks and tries to expand along a different branch of the search tree. The length of the longest list (excluding any *null_node* entries) as well as its composition is maintained. This information is updated, as needed. An

```
procedure DurandPasari_MC(s)
begin
   while (NextNode(s,n))
      if (IsLegalNode(s,n) and !PruningCondition(s)) then
         s' = AddNode(s,n);
         if (size(s')>CurrentMCSize) then
            SaveCurrentMC(s')
            CurrentMCSize = size(s');
         end if
         if (!LeafOfSearchTree(s')) then
            DurandPasari_MC(s');
         end if
         BackTrack(s');
      end if
   end while
end procedure
```

Figure 4: A sketch of the Durand Pasari algorithm for the maximum clique detection.

example of the Durand-Pasari algorithm application is sketched in the Figures 5, 6, 7, 8, 9.

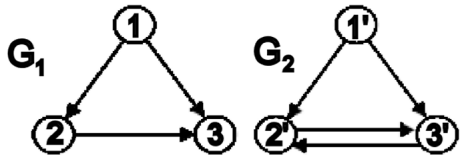


Figure 5: Two directed graphs, G_1 and G_2.

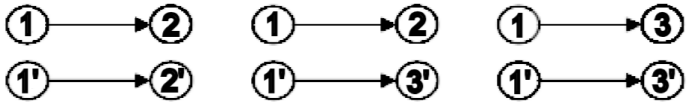


Figure 6: A subset of the maximum common subgraph set between G_1 and G_2 of Figure 5.

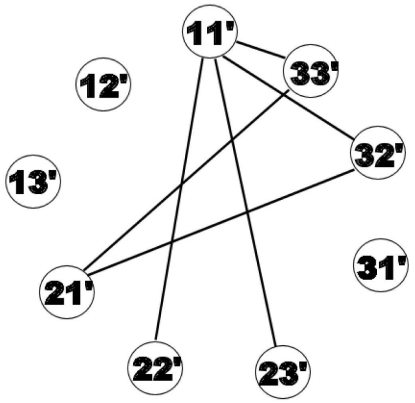


Figure 7: The association graph of the two graphs in Figure 5.

If N_1 and N_2 are the sizes of the starting graphs, with $N_1 \leq N_2$, it can be demonstrated that the algorithm execution will require a maximum of N_1 levels. Since at each level the space requirement is constant (the node list can be shared across levels, since it is accessed in a stack-like fashion), the total space requirement of the algorithm is $O(N_1)$. To this, however, the space needed to represent the association graph must be added. In the worst case the association graph can be a complete graph of $N_1 \cdot N_2$ nodes. In the worst case the algorithm will have to explore $(N_2 + 1)$ nodes at level 1, N_2 at level 2, $(N_2 - 1)$ at level 3, up to $(N_2 - N_1 + 2)$ at level N_1. Multiplying these numbers we obtain a worst case number of states

$$S = (N_2 + 1)(N_2) \cdot \ldots \cdot (N_2 - N_1 + 2) = \frac{(N_2 + 1)!}{(N_2 - N_1 + 1)!} \tag{3}$$

which, for $N_1 = N_2 = N$ reduces to $O(N \cdot N!)$.

3.3 The Balas Yu Algorithm

In order to find a maximum common subgraph between two attributes graphs, in the first step the association graph of the two starting graphs is determined. It has been already observed that the research of a maximum clique of the association graph is equivalent to the research of a maximum common subgraph between the two starting graphs. Balas and Yu proposed in [2] an algorithm to find a maximum clique in a connected graph. The problem is that association graph can also be unconnected, thus for finding out a maximum clique

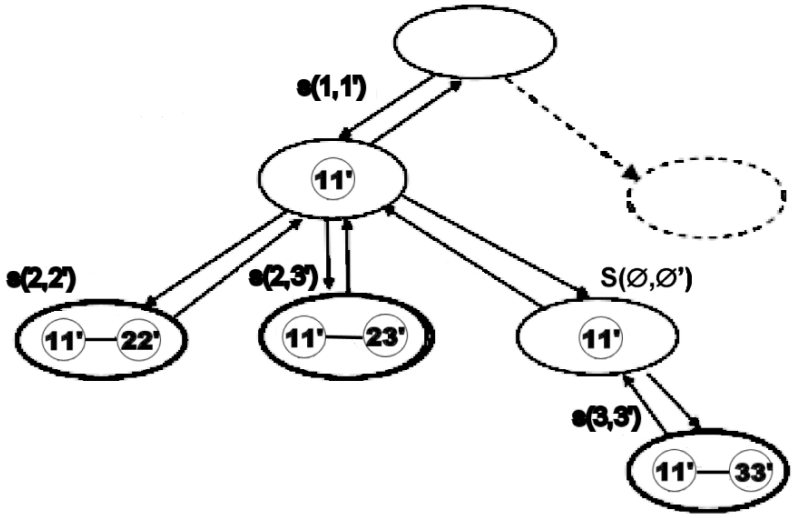


Figure 8: A part of the search tree developed by Durand-Pasari algorithm for graph in Figure 7. In each state $S(\cdot, \cdot)$ a node of the association graph is selected and it is checked whether this node can extend the current clique; the states contained in a thick oval are those in which a maximum common subgraph has been detected.

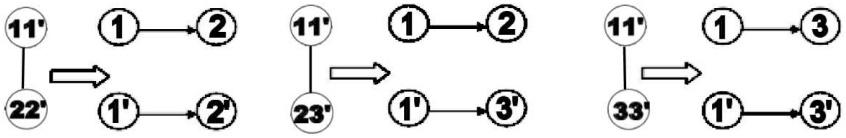


Figure 9: The correspondence between each found maximum clique and the related maximum common subgraph.

of an association graph, this algorithm has been generalized. Consequently the algorithm proposed in this section is more general than the one introduced in [2].

Some basic definitions are needed to describe how this algorithm works.

Let G an undirected graph, let V and E respectively the number of nodes and edges of G, and let $\omega(G)$ the size of a maximum clique.

A *node coloring* of G assigns colors to the nodes of G in such a way that no two adjacent nodes get the same color. The cardinality of a minimum nodes coloring is called the *chromatic number* $\chi(G)$ of G. It is worth noting that $\chi(G)$ is an upper bound for $\omega(G)$ and that the coloring problem has a $O(V + E)$ complexity.

A graph G_T is *triangulated* (or *chordal*) if every cycle of G_T, whose length is at least 4, has a chord. Let the graph $MTS(G_T)$ be a largest triangulated subgraph of G. It can be shown that finding out the graph $MTS(G_T)$, has a $O(V+E)$ complexity and a maximum clique K_T of $MTS(G_T)$ can be found as a byproduct during the search of $MTS(G_T)$. In Figure 10 an undirected graph G and its maximum triangulated subgraph $MTS(G_T)$ are represented.

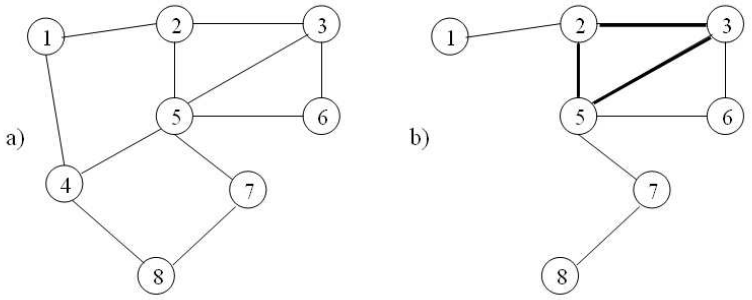


Figure 10: a) an undirected graph; b) a maximum triangulated subgraph of G, $MTS(G_T)$; the edges of a maximum clique of $MTS(G_T)$ are represented with thick lines. The computational complexity to find out a $MTS(G_T)$ is $O(V+E)$ and a maximum clique is obtained as a byproduct.

The algorithm proposed by Balas and Yu can be suitably described through a State Space Representation [34]. Each state `s` is associated to the subproblem of finding a maximum clique in a subgraph of the starting graph. Each subproblem is characterized through the size k of the current maximum clique and a partition of the nodes of the starting graph into three sets: included nodes I (i.e. those nodes that are forcibly included into the subproblem), excluded nodes Ex (i.e. those nodes that are forcibly excluded from the subproblem), unclassified nodes S (i.e. all the other nodes), and a node n chosen into the S set. The solution of the subproblem consists of finding how many nodes can be colored in a graph G_T, whose nodes are the nodes of the set S and whose edges are the edges of the starting graph connecting the nodes of S, fixing a priori the number

$|K_T|$ of colors. The uncolored nodes are stored in a set W. As a byproduct of this coloring procedure a clique, whose size is, in the best case, the sum of $|I|$ and $|K_T|$, is calculated. If this clique is larger than the current one, then it is stored as the current maximum clique. If the set W is empty, then the node n is excluded, and a new subproblem is defined on the modified E and S sets; otherwise, for each node v_i in the set W, a new subproblem, in which v_i is inserted in I, is defined. When it is not possible to choose a further node n, an empty state is obtained and the exploration of the current branch is terminated.

In the first state, the sets I and Ex are empty, all the nodes V are included in S and the size k of the current maximum clique is 0. A node belonging to the unclassified set S is selected through the function `SelectSubProblem(n)`. The function `SolveSubProblem(n)` checks whether it is possible to exclude the node n; in this case no other subproblem descending from the current one will be solved and a new subproblem, chosen in a further branch of the search tree is then analyzed. If the exclusion is not possible, then the subproblem is feasible and $|W|$ new subproblems, descending from the current one, will be defined and solved. If, during the solution of any subproblem, a clique whose size is larger than k is found, then it is stored and it becomes the current maximum clique. After the subproblem has been determined, if S is not empty, the sets I, Ex and S are updated through the function `Update(s)` and the first descending subproblem is immediately solved. After that, a `BackTrack(s)` function is invoked, to restore the previous state and the previous sets, in order to choose a different node n from the set S to built a different descending subproblem. Using this search strategy, whenever a branch is chosen, it will be followed as deeply as possible in the search tree until a leaf is reached. It is noteworthy that every branch of the search tree not excluded by the pruning rules has to be followed, because - except for trivial examples - is not possible to foresee if a better solution exists in a branch that has not yet been explored. A pseudo-code description of Balas Yu algorithm is shown in Figure 11.

The main characteristic of Balas Yu algorithm is that the feasibility function can cut the a large number of branches in the search tree in a polynomial time. For the sake of the clarity, further definitions are needed.

Let the graph S be a graph whose nodes are all the nodes of the set S and whose edges are the edges of the starting graph G, connecting the nodes of the set S; let $MTS(S)$ be a maximum triangulated subgraph of the graph S, and let the graph K_T a maximum clique of $MTS(S)$. Finally, we say that the graph $C_\lambda(G)$ is a λ-chromatic induced subgraph of G if $C_\lambda(G)$ is the largest subgraph of G colored using just λ colors.

These properties are true for every graph G:

- α the size of the maximum clique is smaller than the chromatic number $\chi(G)$ and $\chi(G)$ can be found in a time $O(V + E)$;
- β the size of the maximum clique is larger than the size of the clique K_T, and K_T can be found in a time $O(V + E)$.

A flow diagram of the `SolveSubProblem(n)` function is shown in Figure 12.

Firstly, it is checked whether $|V - Ex| < k$, in this case there are too few nodes to find out a clique larger than the current one; thus n can be inserted in Ex and the function terminates. Otherwise if $|I| = k$ then $MTS(S)$ and its maximum clique K_T are evaluated. The clique, whose nodes are the nodes of I and the nodes of K_T, is the maximum current clique. If its size is larger than k, than it can be stored as the maximum current clique. If $MTS(S) = S$ (i.e. if S is a triangulated graph) then n can be inserted in Ex and the function terminates. Otherwise if $W_{|KT|}(S) = S$ (i.e. if S can be colored using $|K_T|$ colors) then n can be inserted in Ex and the function terminates. If $|I| \neq k$ then if $W_{|k-I|}(S) = S$ (i.e. if S can be colored using $|k - I|$ colors) then n can be inserted in Ex and the function terminates. In all those cases in which it is not possible to color the whole graph S, a set of uncolored nodes $U = \{v_1, \ldots, v_m\}$ is obtained and m new subproblems are generated. Each new subproblem is characterized as follows: $I_{ti} = I \cup v_i$, all other nodes of W and all the neighbors of v_i are inserted in Ex.

It is noteworthy that in all those cases in which it is possible to color the whole graph in polynomial time (i.e. the considered graph is chordal), a branch of the search tree is cut using the property α. An example of the Balas-Yu algorithm application is sketched in Figure 13.

Let N_1 and N_2 be the number of nodes of the first and the second graph respectively. Since at each level of the search tree only one subproblem is solved a time, and only the solving subproblem need of memory resources, the total space requirement of the algorithm is $O(max(N_1, N_2))$. To this, however, the space needed to represent the association graph must be added. In the worst case the association graph can be a complete graph of $N_1 \cdot N_2$ nodes.

```
procedure BalasYu_Mc(s)
begin
   while (SelectSubProblem(s,n))
      begin
         SolveSubProblem(s,n);
         if (size(s)>CurrentMCSize) then
            SaveCurrentMC(s);
            CurrentMCSize = size(s);
         end if
         while(ExistNewSubproblem(s))
            begin
               s'= update(s);
               BalasYu_Mc(s');
            end
         BackTrack(s');
      end
end procedure
```

Figure 11: A sketch of the Balas Yu algorithm for the maximum clique detection.

Table 1: Running times and Space complexities for the seclected algorithms on two graphs G_1 and G_2 with dimension, respectively, of N_1 and N_2.

Algorithm	Space Complexity (worst case)	Time Complexity (worst case)
McGregor [32]	$O(N_1)$	$\frac{(N_2+1)!}{(N_2-N_1+1)!}$
Durand-Pasari [20]	$O(N_1 \cdot N_2)$	$\frac{(N_2+1)!}{(N_2-N_1+1)!}$
Balas-Yu [3]	$O(N_1 \cdot N_2)$	$\frac{(N_2+1)!}{(N_2-N_1+1)!}$

3.4 Summary

The running time and space complexity of the selected algorithms for two graphs G_1 and G_2 with dimension, respectively, of N_1 and N_2 is summarized in Table 1.

4 The Database

In general, two approaches can be followed for generating a graph database; a first way is to collect graphs obtained by processing real data [31], the second possibility is to generate graphs synthetically.

The first approach will ensure that the graphs are realistic, i.e. they are not

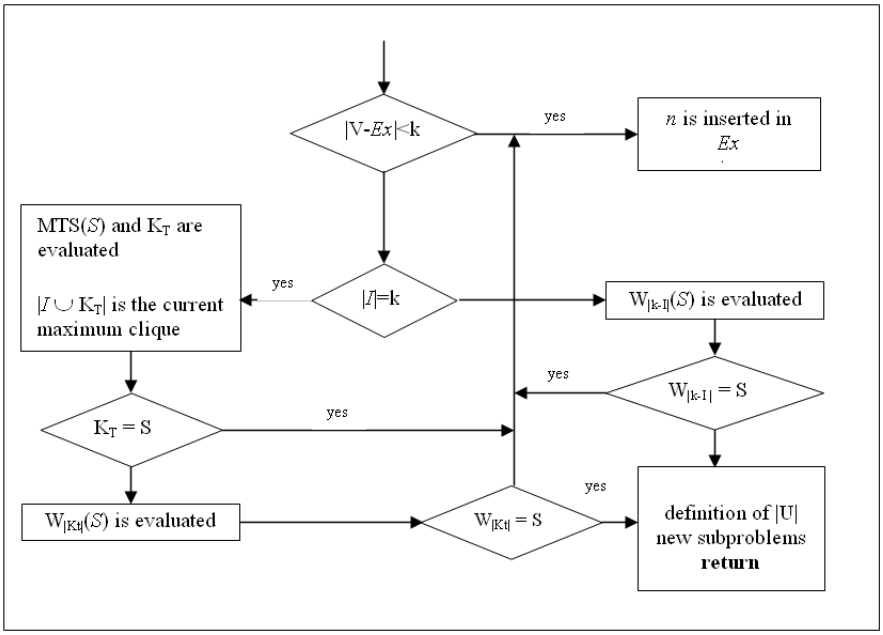


Figure 12: A sketch of the function `SolveSubProblem(n)`.

toy graphs with different properties than the ones encountered in real applications. However in most cases this approach is very expensive, because it may require a huge collection of real data in order to obtain a set of graphs that is representative also of less frequent situations. Moreover the achieved graphs are dependent both on the considered application and on the pre-processing algorithm used, remarkably reducing the general purpose of the database and its usefulness in different contexts. On the other side, the artificial generation of graphs is not only simpler and faster than collecting graphs from real applications, but it allows to control several critical parameters of the underlying graph population, such as the average number of nodes, the average number of edges per node, the number of different attributes, and so on. Starting from these considerations, a quite large database of graphs has been generated synthetically. This database is also easily expandable, in a relatively short time.

The choice of the graph categories to be included in the database, has been

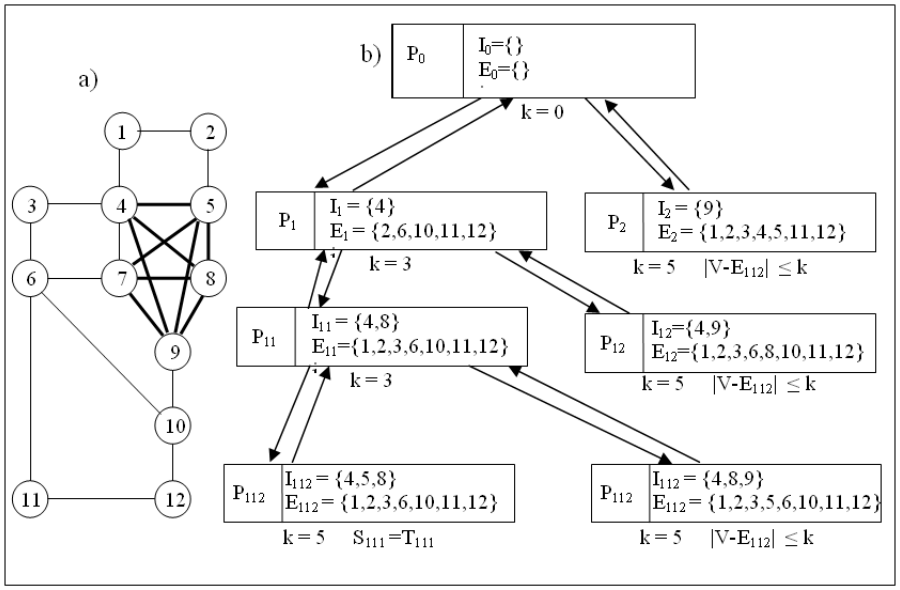


Figure 13: a) an association graph. The stressed edges make evident a maximum clique; b) the search tree constructed by Balas-Yu algorithm. In each state a node is included in the set I and, as a consequence, one or more nodes included in the set E, i.e. the set of those nodes that cannot be included in a clique containing the nodes of the set I. In each state the clique of size k is evaluated. If the difference between the number of nodes and E is smaller than k, a lager maximum common subgraph cannot be found. Also if the set T, i.e. the nodes of maximum triangulated subgraph and the set S, i.e. the set of those nodes not included in both I and E have the same dimension, no larger common subgraph can be found.

realized considering the kinds of graphs that are more frequently used by members of the IAPR-TC15 community (see http://www.iapr-tc15.unisa.it/); a classification of various categories of graphs used within the Pattern Recognition field has been recently proposed in [10]. The proposed database is structured into pairs of graphs. Each pair is characterized through a maximum common subgraph constructed with a specified structure and size. A total of 81,400 pairs of graphs have been generated. In particular, the following categories of graphs have been considered:

- Randomly Connected Graphs;
- Regular Meshes, with differents dimensionalities: 2D, 3D and 4D;
- Irregular Meshes;
- Bounded Valence Graphs;
- Irregular Bounded Valence Graphs;

This kinds of graph have been introduced in [23] for the isomorphism and subgraph isomorphism algorithms evaluation.

Labeled graphs whose size is from 10 to 100 nodes are included in each category. For each size and category of graphs, 500 different pairs have been generated considering five different size of the maximum common subgraph that the pair holds between them (i.e. for each size of the maximum common subgraph have been generated 100 different pairs).

As regards the labeling, random values for the attributes have been generated, since any other choice would imply assumptions about an application dependent model of the represented graphs.

Choosing a uniform distribution of the values, it is possible to assume, without any loss of generality, that attributes are represented by integer numbers. In fact, in most real cases attributes can be represented using a fine alphabet after a quantization stage. Also, the use of floating point values can be somewhat more problematic, because:

- usually it makes little sense to compare floating point numbers for strict equality, and,
- there is no universally useable binary format for storing binary numbers;

These disadvantages are not repaid for by significant advantages, since also integer attributes can be used to perform arithmetic tasks, e.g. distance evaluation.

One of the most important parameter characterizing the difficulty of the matching problem is the number A of different attributes values (i.e. the size of the alphabet): obviously the higher this number, the easier is the matching problem.

It should be important to have in the database different values of A: in such case it is possible to decide either to measure the matching time keeping A constant and varying the size of the graphs, or to increase A as the size of

the graphs increases; both choices make sense for estimating the performance of different types of applications.

In order to avoid the need to have several copies of the database with different values of A, in this database, each attribute is generated as a 16-bit value, using a random number generation algorithm ensuring that each bit is sufficiently random. Then, it is possible to choose any value of the form 2^k, with k not greater than 15, just by using, in the attributes comparison function, only the first k bits of the attribute. Furthermore, for values of A that are not powers of 2, attribute value modulo A can be used, if A is sufficiently smaller than 2^{16}, without introducing any undesired bias in the distribution of the values. Using this technique an attributed graph database has beet built. The database is enough general for experimenting with many different attribute cardinalities, avoiding the explosion of the size required to store the database.

A brief description of the properties of each category of graphs and of the motivation inspiring the choice of including them in the database is given later in this section, together with the number of generated pairs of graphs per category.

Each category of graphs included into the database is characterized by a size from 10 to 100 nodes. In particular thirteen values of size have been considered (i.e., 10, 15, 20, 25, 30, 35, 40, 50, 60, 70, 80, 90 and 100). The considered graphs are simple, i.e. there are neither self loops nor multiple edges connecting the same two nodes. Graphs have been generated in pairs, and for each pair the characteristic parameters of the maximum common subgraph are fixed, as detailed in the following. Furthermore, for each selected category of graphs, five different size of maximum common graph (i.e. 10%, 30%, 50%, 70% and 90% of the size of the starting pair of graphs) have been taken into account. Finally, for each value of the generation parameters (i.e. graph size, MCS size and the parameters specific to each category) 100 pairs of graphs are included into the database.

The organization of the entire database is shown in Table 2.

4.1 Randomly Connected Graphs

In graphs belonging to this category, edges connect nodes without any structural regularity. This category of graphs has been introduced for modeling applications in which each entity, represented by a node, can establish relations, represented by edges, with any other entity, independently of the relative positions. This hypothesis typically occurs in the middle and high processing levels of a computer vision task [3].

In randomly connected graphs, it is assumed that the probability of an edge connecting two nodes of the graph is independent of the nodes themselves. The same model proposed in [39] has been adopted for generating these graphs: let n_i and n_j be two distinct nodes of the graph; the probability η that an edge is connecting n_i and n_j is fixed and assumed to be uniform.

According to the meaning of η, if N is the total number of nodes of the graph, the expected number of its edges is $\eta \cdot N \cdot (N-1)$. However, if this number is

Table 2: Graph Database organization: for each kind of graph, for each size of the maximum common subgraph and for each value of the characteristic parameters, the number of pairs that have been generated, is shown.

DATABASE ORGANIZATION (81,400 pairs of graphs)	Graph Categories	MCS dimension					
			10%	**30%**	**50%**	**70%**	**90%**
	Random connected graphs *18600 pairs*	$\eta = 0.01$	100	100	100	100	100
		$\eta = 0.05$	1100	1100	1100	1100	1100
		$\eta = 0.1$	1100	1300	1300	1300	1300
		$\eta = 0.2$	1100	1300	1300	1300	1300
	Regular Meshes *8100 pairs*	2D	700	700	1000	1000	1000
		3D	500	700	400	400	400
		4D	100	500	300	100	300
	Irregular Meshes *23900 pairs*	2D $\rho = 0.2$	400	700	900	1000	1000
		2D $\rho = 0.4$	700	700	1000	1000	1000
		2D $\rho = 0.6$	700	700	1000	1000	1000
		3D $\rho = 0.2$	500	700	400	400	400
		3D $\rho = 0.4$	500	700	400	400	400
		3D $\rho = 0.6$	500	700	400	400	400
		4D $\rho = 0.2$	100	500	300	100	300
		4D $\rho = 0.4$	100	500	300	100	300
		4D $\rho = 0.6$	100	500	300	100	300
	Bounded Valence Graphs *15400 pairs*	$V = 3$	700	1200	1300	1300	1300
		$V = 6$	400	1000	1200	1300	1300
		$V = 9$	100	800	1100	1200	1200
	Irregular bounded Valence Graphs *15400 pairs*	$V = 3$	700	1200	1300	1300	1300
		$V = 6$	400	1000	1200	1300	1300
		$V = 9$	100	800	1100	1200	1200

not enough large to obtain a connected graph, further edges are suitably added until the generated graph becomes connected.

Three different values of the edge density η have been considered (0.05, 0.1 and 0.2). In the database we have not considered all the sizes of the graphs because, for some values of the size, the resulting graphs were not meaningful (e.g. for the graphs with 10 nodes, the maximum common subgraph with 10% of the size of the graphs is a 1 node graph, and it was not considered).

4.2 Regular Meshes

These graphs are introduced for modeling applications characterized through regular structures (e.g. lower levels of a vision task). Furthermore, it is generally agreed that regular structured graphs often represent a worst case for general graph matching algorithms (i.e. algorithms working on any type of graphs) [39]. To solve this problem, specialized graph matching methods have been developed to efficient perform the matching for given graph structures. Thus the database includes, as regular graphs, the mesh connected graphs (2D, 3D and 4D).

The considered 2D meshes are graphs in which each node (excluding those nodes belonging to the border of the mesh) is connected with its 4 neighbor nodes. Similarly, each node of a 3D and 4D graph has respectively connections with its 6 and 8 neighbor nodes.

Since not every number of nodes can be used for generating a mesh, the percentage of nodes composing the maximum common subgraph are not exactly the ones reported before. For instance for the pairs with graph size of 50, the percentage of 10% for the maximum common subgraph has not been considered for 3D meshes, because a graph of five nodes cannot be a 3D mesh; instead we have used a 3D mesh with 8 nodes leading to a maximum common subgraph that is the 16% of the size of the starting graph.

4.3 Irregular Mesh-Connected

Graphs introduced for the simulation of the behavior of the algorithms in presence of slightly distorted meshes. They have been obtained from regular 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 edges is ρN, where ρ is a constant greater than 0. Note that, the closer ρ to 0 is, the more symmetric graphs are.

For each category of irregular meshes (2D, 3D e 4D), three values of ρ have been considered (0.2, 0.4 and 0.6) and graphs whose size is from 10 up to 100 nodes have been generated; for each pair of graphs five different sizes of maximum common graph (10%, 30%, 50%, 70% and 90% of the size of the starting pair of graphs) are taken into account and, for each size, 100 pairs of graphs are included into the database. Some values of size are not considered for the same reason described for regular meshes. Furthermore, for irregular meshes other pairs are not considered: these pairs are that which extra edges are zero considering the size of maximum common subgraph (e.g. the pairs which graph

size is 15 with a maximum common subgraph of 4 nodes (30%) have not extra edges for ρ=0.2 (extra-edges=$\rho N = 0.8$) and are not considered).

4.4 Bounded Valence Graphs

These graphs can model applications in which each object (i.e. a node) establish a fixed number of relations (through edges) with other objects, not necessarily with those belonging to its neighborhood [30]. More in detail, 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, 9) have been generated and for each value of v. Also for the bounded valence graphs are not considered all the values of the size: for some percentage the size of the maximum common subgraph is not enough for building a graph with the fixed valence (e.g. for the graphs with 50 nodes, the maximum common graph with 5 nodes (10% of the size of the starting graph) cannot be a fixed valence graph with $v = 9$, so these pairs of graphs are not considered).

In order to introduce some irregularities in the bounded valence graphs, also *Irregular Bounded Valence Graphs* are considered.

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 node 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 $M = 0.1 \cdot N \cdot V$, 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 $\alpha e^{-\beta i}$ 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.

The number and type of irregular bounded graphs included into the database is the same as the bounded valence graphs.

5 Experimental Results

In order to perform the benchmarking activity, we implemented a version of each of the three selected algorithms in C++. Our versions find the maximum common subgraph of directed, labeled and unconnected graphs.

The value of the attributes on the graphs of the database is depending on the value of A (i.e. the number of different attributes). Thus, for each different value of A, a graph with different attributes is selected. Three values of A have been used, namely 33%, 50% and 75% of the size of the graphs. Results are clustered in 63 different groups, and each of them is detailed in a different graphic. In the current section we only summarize our experimental results; more details are provided in the electronic appendix.

For this aim the execution of each algorithm is stopped when the first maximum common subgraph is determined and however a time-out of 30 minutes for each matching problem is provided. The benchmarking has been performed on an Intel Celeron 766 Mhz PC, equipped with 128 MB of RAM.

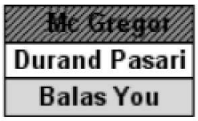
For each category of graphs (i.e. meshes, random graphs,...) a table that we call the *winner map*, is reported (see Figures 14, 15, 16, 17). Each winner map has the size of the graph on the columns and the parameters characterizing the shape of the graph on the rows (for instance, in case of random graphs, the selected parameters are the density and the number of attributes). In a winner map, each cell reports the fastest algorithm for an assigned graph matching problem. Moreover, the magnitude degree of the speed of the fastest algorithm on the second one is reported on each cell.

A different shade is associated to each algorithm, thus just observing the shade of the cells it is immediately understandable which algorithm solved the problem using the smallest time.

5.1 Randomly Connected Graphs[1]

Figure 14 shows the behavior of the selected algorithms with reference to the randomly connected graphs. McGregor always performs better on sparse graphs ($\eta = 0.05$). The main reason is that McGregor solves the problem without using the association graph. When the graph density is low, the association graph is

[1] for more details see Appendix A.1



Random Graphs

η	A \ N	10	15	20	25	30	35	40
0.05	33%	n.a.		2	2	2	2	1
	50%			2	2	3	3	3
	75%			1	2	2	3	3
0.1	33%	0	1	0	0	0	0	1
	50%	0	0	0	0	0	0	0
	75%	0	0	0	1	0	0	0
0.2	33%	1	1	1	1	1	1	1
	50%	1	1	0	0	0	0	0
	75%	0	0	1	1	0	0	0

Figure 14: The winning table for random connected graphs. In the table the percentages are the size of the alphabet of attributes respect the number of nodes. Furthermore η represents the graph density.

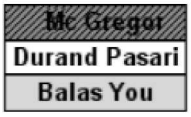
large and strongly connected, thus the transformation can result not convenient. For an increasing density ($\eta = 0.1$) of the graphs, McGregor algorithm is still winning when the graphs are small or the alphabet of attributes is large. In the other cases the problem is solved more efficiently using the Durand-Pasari algorithm. In case of large graphs with high densities, Balas-Yu is the winner. The reason is that the heuristic of the algorithm is more sophisticated and expensive to compute. So, for small graphs, the time saved using the pruning rules deriving by the heuristic is not counterbalanced by the time used to compute the heuristic itself. On the contrary, the use of this refined heuristic can give the best performance on larger graphs.

5.2 Meshes[2]

In Figure 15, Figure 16(a) and Figure 16(b) the performance of the algorithms on regular and irregular meshes are shown. The behavior of the algorithms is quite similar for the benchmark on meshes 2D, 3D and 4D.

For each type of meshes, McGregor algorithm performs better in most cases. The main reason is that for the meshes the number of edges is linear with the number of nodes, thus meshes are not very dense graphs. Then, in most cases, the association graph is large and dense and thus it is not convenient

[2]for more details see Appendices A.2, A.3, and A.4



Mesh 2D

ρ	A \ N	10	15	20	25	30	35	40
0	33%	0	1	0	1	1	0	2
	50%	0	0	1	2	3	2	3
	75%	0	1	1	2	2	2	3
0.2	33%	0	1	0	0	0	0	0
	50%	0	0	1	1	2	1	1
	75%	0	0	1	1	2	2	2
0.4	33%	0	1	1	0	0	1	0
	50%	0	0	0	0	1	0	2
	75%	0	0	1	1	2	2	2
0.6	33%	0	1	1	0	1	1	1
	50%	0	0	0	0	1	0	2
	75%	0	0	0	1	2	1	2

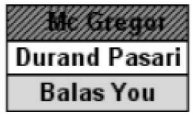
Figure 15: The winning table for meshes 2D. In the table the percentages are the size of the alphabet of attributes respect the number of nodes. Furthermore ρ represents the mesh irregularity.

to solve the problem of finding a maximum clique. When the alphabet size is more restricted, and so the association graph is more dense, for smaller graphs Durand-Pasari performs better. Moreover Durand-Pasari algorithm performs better on small graphs, when the irregularity of the meshes (the parameter ρ) increases. Finally, for larger graphs, with an higher degree of irregularity and a more restricted alphabet size, Balas Yu is the fastest algorithm. In those cases the search tree is very dense and its exploration is very time consuming. Thus a good heuristic, cutting a considerable number of branches gives a solid speed up to the algorithm.

5.3 Regular Bounded Valence Graphs[3]

In Figure 17(a) the performance of the selected algorithms on *regular bounded valence graphs* is shown. When the connection degree v is 3 the fastest algorithm is McGregor. The reason is that when the connection degree is small, the number of edges is small also, similarly to the case of meshes, thus the graphs are not very dense. Then, in most of the cases, the association graph is large and dense and it is not convenient to solve the problem of finding the maximum clique. If the connection degree is 6 or 9, and if the alphabet size is more restricted, the association graph becomes less dense, thus it becomes convenient

[3]for more details see Appendix A.5



Mesh 3D

ρ	A \ N	10	15	20	25	30	35	40
0	33%	0	1	0	0	0	0	2
	50%	0	0	0	1	1	0	2
	75%	0	0	1	2	2	1	2
0.2	33%	0	1	1	0	0	2	0
	50%	0	0	0	0	0	0	1
	75%	0	0	1	1	1	1	2
0.4	33%	0	1	1	1	1	2	1
	50%	0	0	0	0	0	0	1
	75%	0	0	1	1	1	0	2
0.6	33%	1	1	1	1	1	2	1
	50%	0	0	0	0	0	1	1
	75%	0	0	1	1	1	1	2

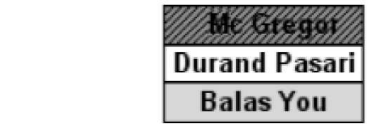
a)

Mesh 4D

ρ	A \ N	10	15	20	25	30	35	40
0	33%	n.a.		0	0	1	2	2
	50%			0	1	2	3	4
	75%			1	2	2	2	3
0.2	33%	n.a.		0	0	0	1	1
	50%			0	0	1	2	2
	75%			1	1	1	1	2
0.4	33%	n.a.		1	1	1	2	1
	50%			0	0	0	0	1
	75%			0	1	1	2	3
0.6	33%	n.a.		0	0	0	0	1
	50%			0	0	0	1	2
	75%			0	0	1	1	2

b)

Figure 16: The winning tables for meshes 3D (a) and meshes 4D (b). In the tables the percentages are the size of the alphabet of attributes respect the number of nodes. Furthermore ρ represents the mesh irregularity.



Regular Bounded Graphs

v	A \ N	10	15	20	25	30	35	40
3	33%	0	1	1	1	2	3	4
	50%	0	0	1	2	3	3	4
	75%	0	0	1	2	2	3	3
6	33%	1	1	1	1	1	0	0
	50%	0	0	0	0	0	0	0
	75%	0	0	1	1	1	1	2
9	33%	0	1	1	1	1	2	1
	50%	0	0	0	0	0	0	1
	75%	0	0	1	1	1	0	2

a)

Irregular Bounded Graphs

v	A \ N	10	15	20	25	30	35	40
3	33%	0	1	0	0	1	0	0
	50%	0	1	1	0	0	3	0
	75%	0	0	1	2	2	3	3
6	33%	1	1	1	1	1	0	0
	50%	0	1	1	0	0	0	0
	75%	0	0	0	0	0	0	0
9	33%	n. a.	0	0	0	0	0	0
	50%		0	0	0	0	0	0
	75%		0	0	0	0	0	0

b)

Figure 17: The winning tables for bounded valence graphs (a) and irregular bounded graphs (b). In the tables the percentages are the size of the alphabet of attributes respect the number of nodes. Furthermore v represents the maximum connection degree between two nodes.

to use algorithms for finding the maximum clique. In these cases Durand-Pasari performs better on smaller graphs and Balas Yu performs better on larger graphs due the more refined and complex heuristic. Finally, when the alphabet size becomes wider (namely $A = 75\%$), McGregor is always the fastest algorithm.

5.4 Irregular Bounded Valence Graphs[4]

In Figure 17(b) the performance of the algorithms on irregular bounded valence graphs is shown. In most cases, for small graphs, Durand-Pasari performs better. The only cases in which this algorithm is not the fastest is when the average connection degree is 3 and the number of attributes is medium or large. The reason is that for these cases the number of edges is not large, thus the graphs are not very dense. Then, the association graph is large and sparse and it is not convenient to solve the problem of finding a maximum clique. When the connection degree is 6 or 9, the association graph becomes less dense, thus it becomes always convenient to solve the matching problem using the maximum clique. In these cases Durand-Pasari performs better on smaller graphs and Balas Yu performs better on larger graphs due the more refined heuristic.

6 Discussion and Conclusions

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

The usefulness of the proposed benchmark lies in the choice of the algorithms and in the built database. We have chosen the most representative algorithms between those present in scientific literature. Furthermore the database covers almost the totality of graph structures used in Pattern Recognition field. Afterwards the benchmarking activities we can conclude that the first algorithm (and all algorithms that are derived from it) is more suitable than the other ones for the applications that use regular graphs (meshes, bounded valence graphs, etc.) to represent data. In fact in these cases the effort for building the association graph is not counterbalanced by a faster processing. In the other cases (when graphs have not a regular structure) the very efficient response time of the second algorithm repays the time spent to construct the association graph. For largest graphs the third algorithm can be used efficiently because of its smarter, albeit more complex, heuristic.

As it could be expected, experimental results show that no algorithm performs definitively better than the others but, depending on the structure of the graphs, each algorithm can be considerably faster than the others on a restricted set of problems.

[4] for more details see Appendix A.6

As a future work, we are 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 that will have the need to perform an experimental comparison with these and possibly others algorithms.

References

[1] M. Abdulrahim and M. Misra. A graph isomorphism algorithm for object recognition. *Pattern Analysis and Applications*, 1(3):189–201, 1998.

[2] E. Balas and C. S. Yu. Finding a maximum clique in an arbitrary graph. *SIAM J. Computing*, 15(4), 1986.

[3] D. Ballard and C. M. Brown. *Computer Vision*. Prentice Hall, NJ, 1982.

[4] I. M. Bomze, M. Budinich, P. M. Pardalos, and M. Pelillo. *The Maximum Clique Problem*, volume 4 of *Handbook of Combinatorial Optimization*. Kluver Academic Publisher, Boston Pattern MA, 1999.

[5] C. Bron and J. Kerbosch. Finding all the cliques in an undirected graph. *Communication of the ACM*, 16:189–201, 1973.

[6] H. Bunke, M. Gori, M. Hagenbuchner, C. Irniger, and A. Tsoi. Generation of images databases using attributed plex grammars. In *Proc. of the 3rd IAPR TC-15 International Workshop on Graph-based Representations*, pages 200–209, 2001.

[7] H. Bunke, X. Jiang, and A. Kandel. On the minimum common supergraph of two graphs. *Computing*, 65(1):13–25, 2000.

[8] H. Bunke and K.Sharer. A graph distance metric based on the maximal common subgraph. *Pattern Recognition Letters*, 19(3):255–259, 1998.

[9] H. Bunke and B. Messmer. *Efficient Attributed Graph Matching and its Application to Image Analysis*, volume 974 of *Lecture Notes In Computer Science*, pages 45–55. Springer-Verlag, proceedings of the 8th international conference on image analysis and processing edition, 1995.

[10] H. Bunke and M.Vento. Benchmarking of graph matching algorithms. In *Proc. of the 2nd Workshop on Graph-based Representations*, pages 109–114, 1999.

[11] M. Burge and W. G. Kropatsch. A minimal line property preserving representation for line images. *Computing*, 62(4):355–368, 1999.

[12] A. K. Chhabra. Graphic symbol recognition: an overview. In *Second IAPR Workshop on Graphics Recognition (GREC97)*, pages 244–252, 1997.

[13] A. Chianese, L. Cordella, M. D. Santo, and M. Vento. *Classifying Character Shapes*, pages 155–164. Visual Form: Analysis and Recognition. Plenum Press, New York, proceedings of the 8th international conference on image analysis and processing edition, 1992.

[14] W. Christmas, J. Kittler, and M. Petrou. Structural matching in computer vision using probabilistic relaxation. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 17(8):749–764, 1995.

[15] M. M. Cone, R. Venkataraghven, and F. W. McLafferty. Molecular structure comparison program for the identification of maximal common substructures. *Journal of the American Chemistry Socitey*, 99(23):7668–7671, 1977.

[16] 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.

[17] L. Cordella, P. Foggia, C. Sansone, and M. Vento. Performance evaluation of the VF graph matching algorithm. In *Proc. of the 10th International Conference on Image Analysis and Processing*, pages 1172–1177, 1999.

[18] L. Cordella, P. Foggia, C. Sansone, and M. Vento. An improved algorithm for matching large graphs. In *Proc. of the 3rd IAPR-TC-15 International Workshop on Graph-based Representations*, pages 149–159, 2001.

[19] L. Cordella and M. Vento. Symbol and shape recognition. In *Third IAPR Workshop on Graphics Recognition (GREC99)*, pages 179–186, 1999.

[20] P. J. Durand, R. Pasari, J. W. Baker, and C. Tsai. An efficient algorithm for similarity analysis of molecules. *Internet Journal of Chemistry*, 2, 1999.

[21] H. Ehrig. Introduction to graph grammars with applications to semantic networks. *Computers & Mathematics with Applications*, 23:557–572, 1992.

[22] B. Falkenhainer, K. Forbus, and D. Gentner. The structure-mapping engine: algorithms and examples. *Artificial Intelligence*, 41:1–63, 1989.

[23] P. Foggia, C. Sansone, and M.Vento. A database of graphs for isomorphism and sub-graph isomorphism benchmarking. In *Proc. of the 3rd IAPR TC-15 International Workshop on Graph-based Representations*, pages 176–187, 2001.

[24] G. Ford and J. Zhang. A structural graph matching approach to image understanding. In *Proc. SPIE Intelligent Robots Computer Vision X: Algorithms Techniques*, volume 1607, pages 559–569, 1992.

[25] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman & Co., New York, 1979.

[26] L. Jianzhuang and L. Y. Tsui. Graph-based method for face identification from a single 2D line drawing. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 23(10):1106–1119, 2001.

[27] G. Levi. A note on the derivation of maximal common subgraphs of two directed or undirected graphs. *Calcolo*, 9:341–354, 1972.

[28] J. Llados, E. Marti, and J. Villanueva. Symbol recognition by error-tolerant subgraph matching between region adjacency graphs. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 23(10):1137–1143, 2001.

[29] S. Lu, Y. Ren, and C. Suen. Hierarchical attributed graph representation and recognition of handwritten chinese characters. *Pattern Recognition*, 24:617–632, 1991.

[30] E. M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of Computer System Science*, pages 42–65, 1982.

[31] R. Mathon. Sample graphs for isomorphism testing. *Congressus Numerantium*, 21:499–517, 1978.

[32] J. McGregor. Backtrack search algorithms and the maximal common subgraph problem. *Software Practice and Experience*, 12:23–34, 1982.

[33] B. T. Messmer. *Efficient Graph Matching Algorithms for Preprocessed Model Graphs*. PhD thesis, Institute of Computer Science and Applied Mathematics, University of Bern, 1996.

[34] N. J. Nilsson. *Principles of Artificial Intelligence*. Springer-Verlag, 1982.

[35] I. Rocha and T. Pavlidis. A shape analysis model with application to a character recognition system. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 16:393–404, 1994.

[36] D. Rouvray and A. Balaban. Chemical applications of graph theory. *Applications of Graph Theory*, pages 177–221, 1979.

[37] A. Sanfeliu and K. Fu. A distance measure between attributed relational graphs for pattern recognition. *IEEE Transaction on Systems, Man and Cybernetics*, 13:353–362, 1983.

[38] K. Shearer, H. Bunke, S. Venkatesh, and D. Kieronska. Graph matching for video indexing. *Computing*, 12:53–62, 1998.

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

7 Appendix

A.1 Experimental results for random graphs

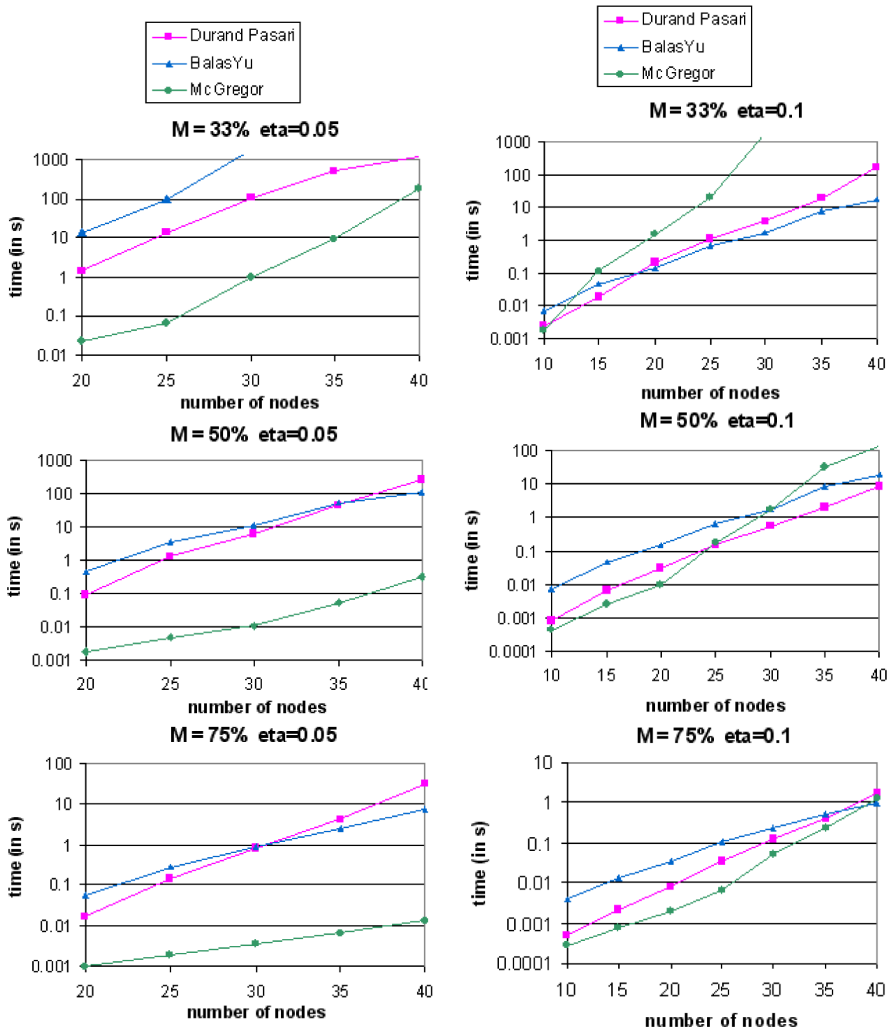


Figure 18: The size of the attribute alphabet is $M = 33\%$, $M = 50\%$ and $M = 75\%$ of the number of nodes, and the density is $\eta = 0.05$ and $\eta = 0.1$.

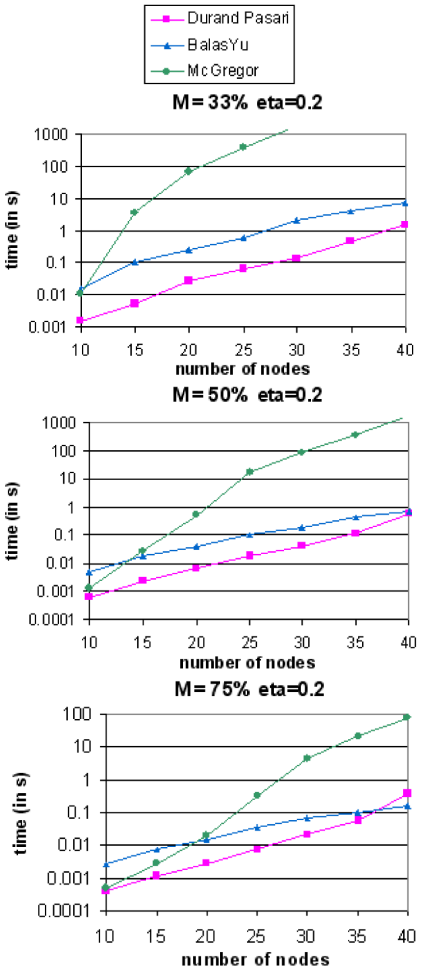


Figure 19: The size of the attribute alphabet is $M = 33\%$, $M = 50\%$ and $M = 75\%$ of the number of nodes, and the density is $\eta = 0.2$.

A.2 Experimental results for 2D meshes

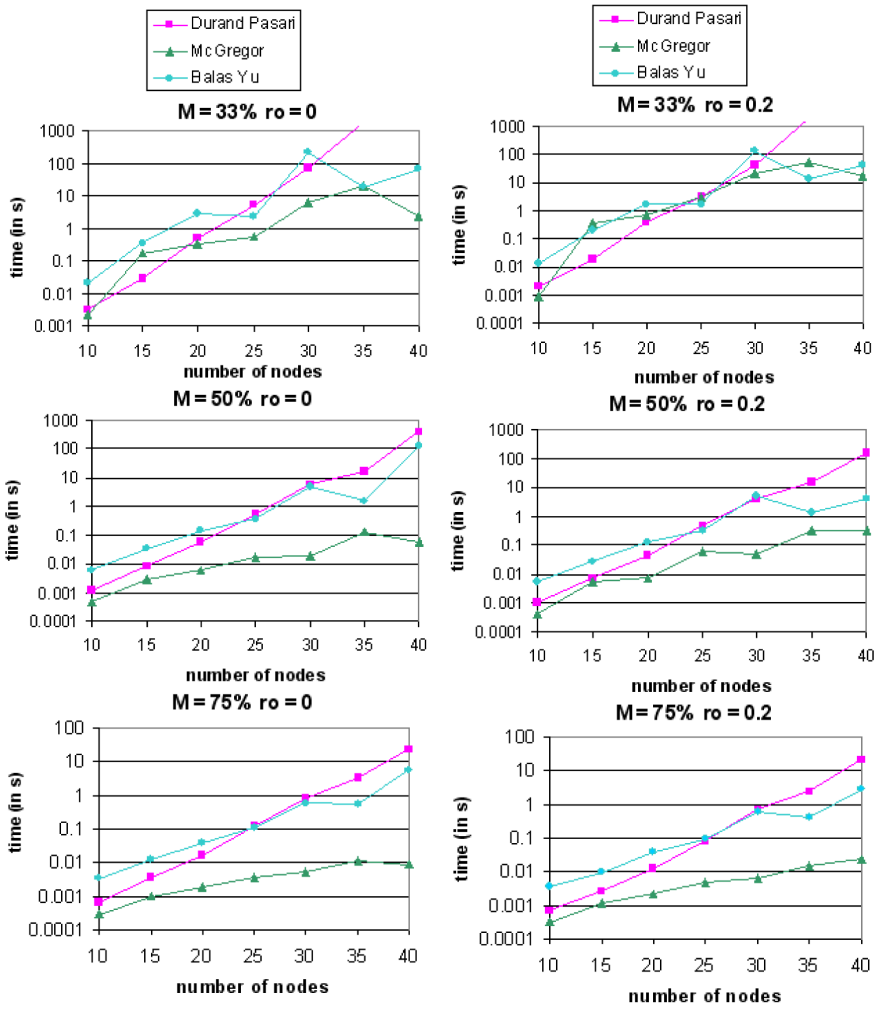


Figure 20: The irregularity parameter of the mesh is $\rho = 0$ and $\rho = 0.2$; the size of the attribute alphabet is $M = 33\%$, $M = 50\%$ and $M = 75\%$ of the number of nodes.

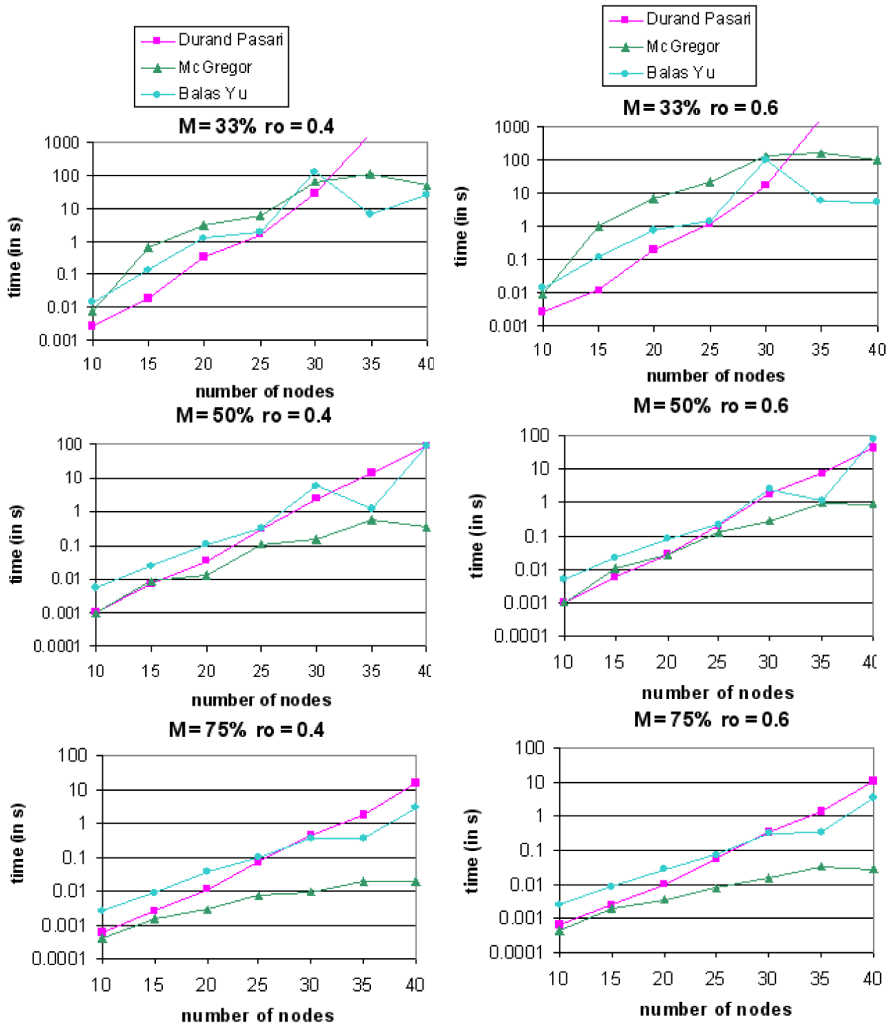


Figure 21: The irregularity parameter of the mesh is $\rho = 0.4$ and $\rho = 0.6$; the size of the attribute alphabet is $M = 33\%$, $M = 50\%$ and $M = 75\%$ of the number of nodes.

A.3 Experimental results for 3D meshes

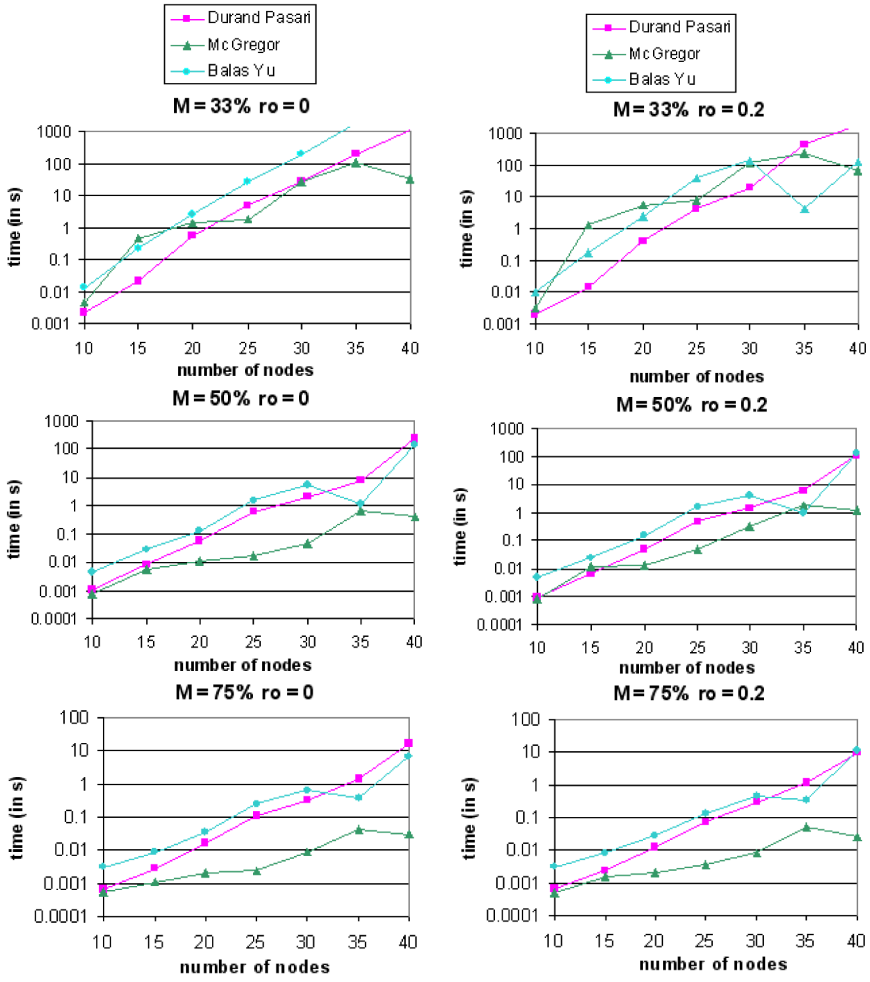


Figure 22: The irregularity parameter of the mesh is $\rho = 0$ and $\rho = 0.2$; the size of the attribute alphabet is $M = 33\%$, $M = 50\%$ and $M = 75\%$ of the number of nodes.

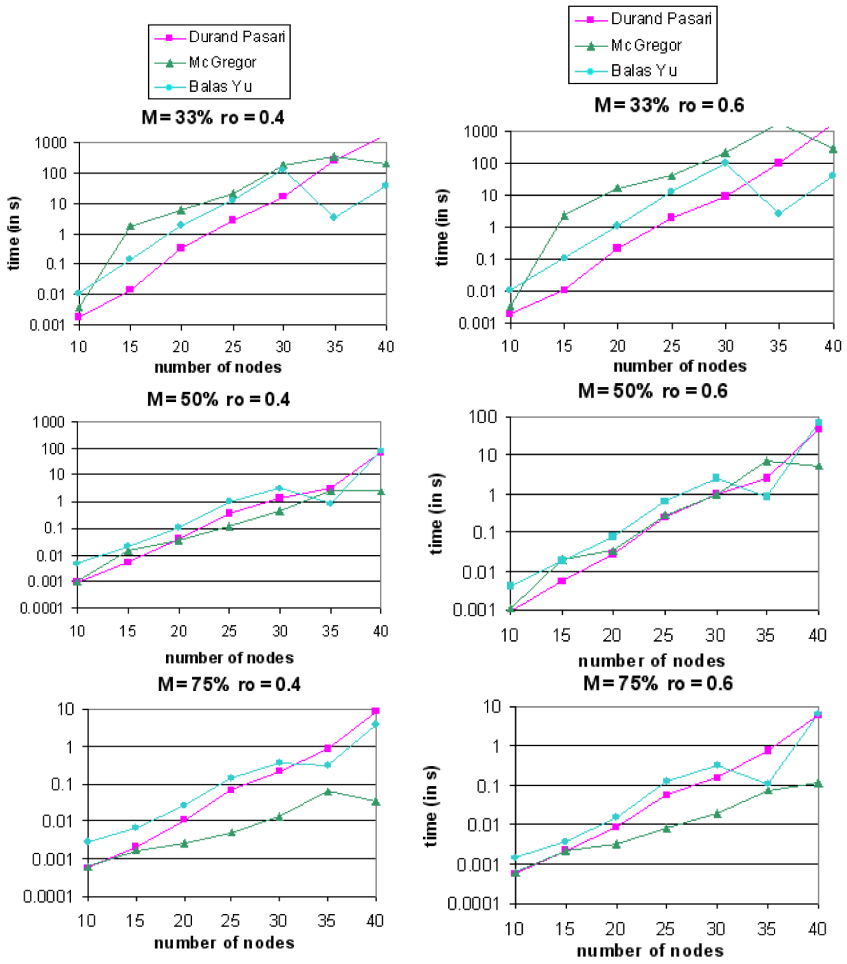


Figure 23: The irregularity parameter of the mesh is $\rho = 0.4$ and $\rho = 0.6$; the size of the attribute alphabet is $M = 33\%$, $M = 50\%$ and $M = 75\%$ of the number of nodes.

A.4 Experimental results for 4D meshes

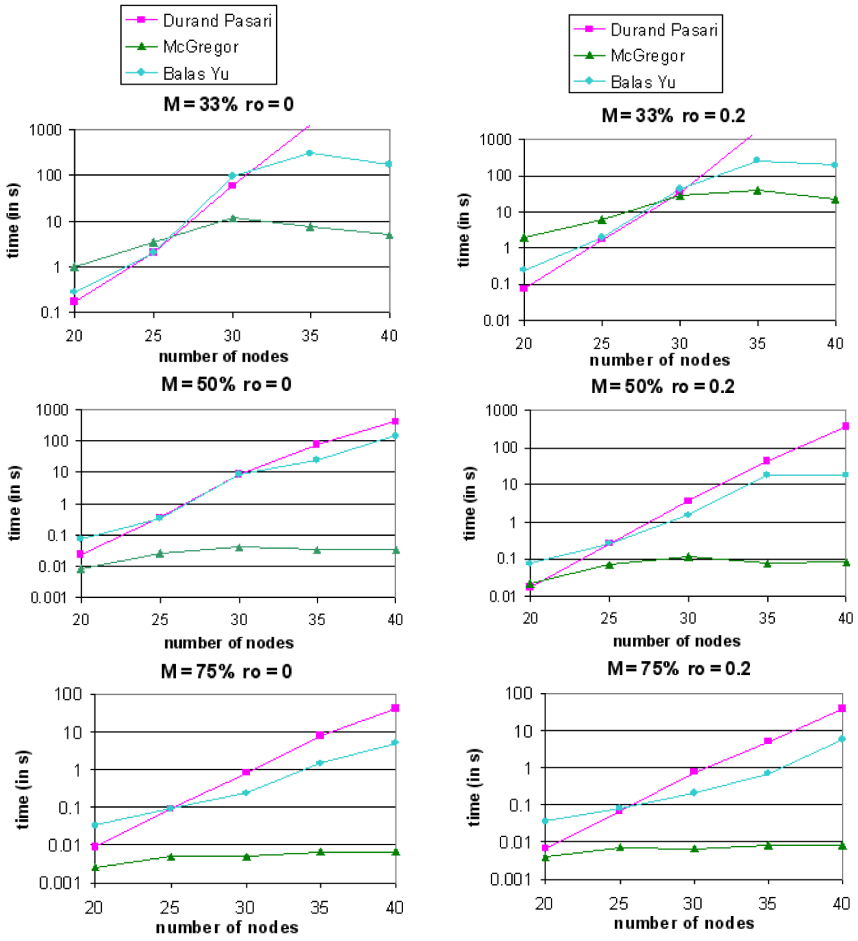


Figure 24: The irregularity parameter of the mesh is $\rho = 0$ and $\rho = 0.2$; the size of the attribute alphabet is $M = 33\%$, $M = 50\%$ and $M = 75\%$ of the number of nodes.

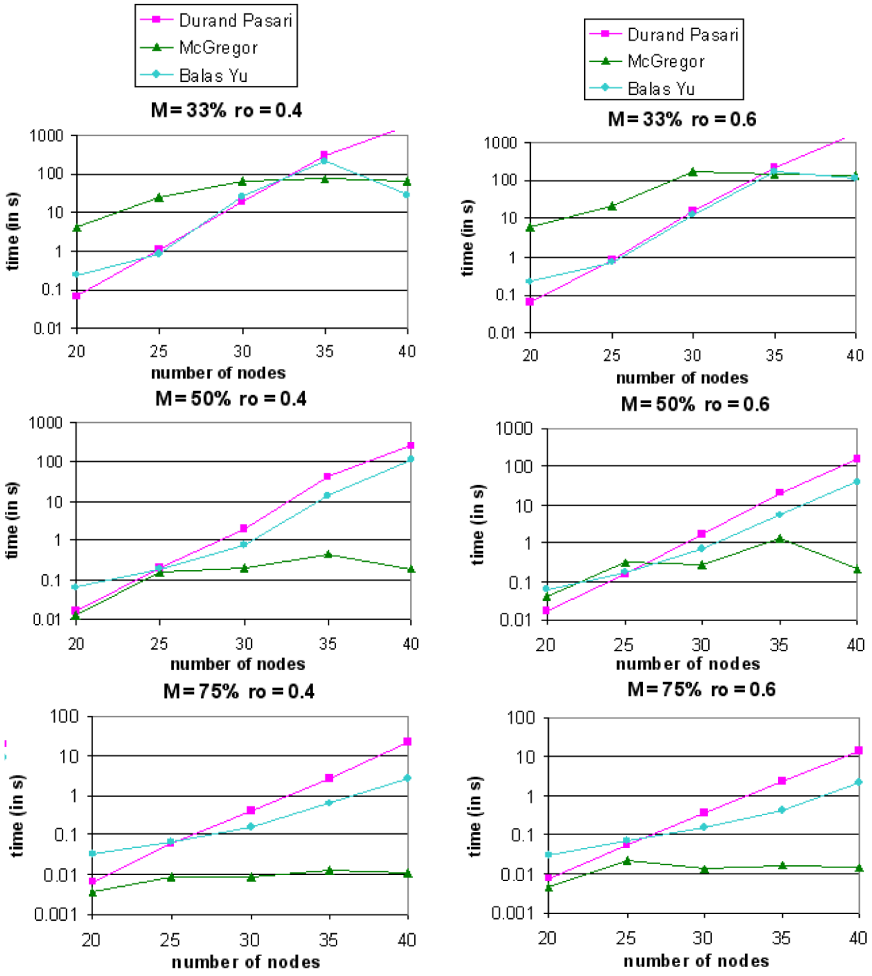


Figure 25: The irregularity parameter of the mesh is $\rho = 0.4$ and $\rho = 0.6$; the size of the attribute alphabet is $M = 33\%$, $M = 50\%$ and $M = 75\%$ of the number of nodes.

A.5 Experimental results for regular bounded graphs

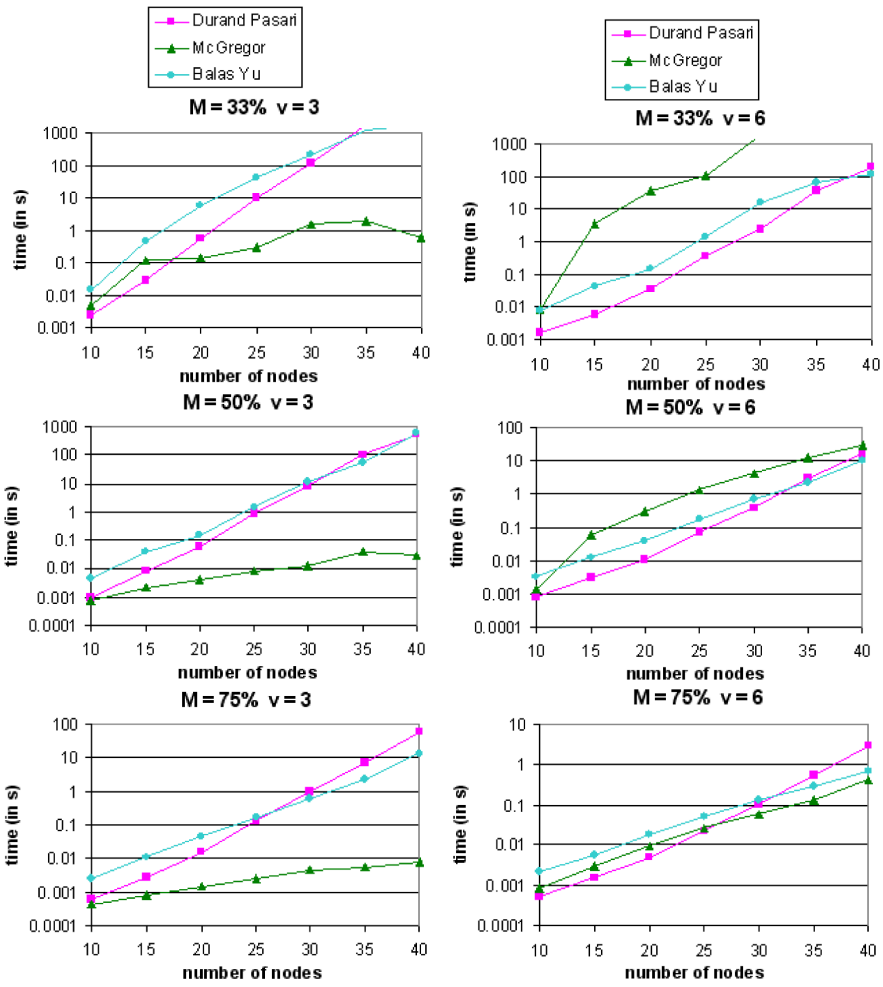


Figure 26: The size of the attribute alphabet is $M = 33\%$, $M = 50\%$ and $M = 75\%$ of the number of nodes, and the valence v is 3 and 6.

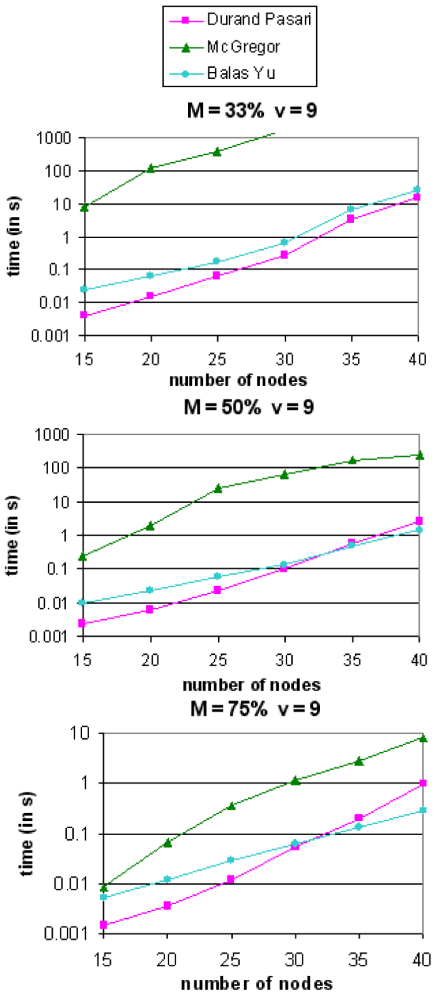


Figure 27: The size of the attribute alphabet is $M = 33\%$, $M = 50\%$ and $M = 75\%$ of the number of nodes, and the valence v is 9.

A.6 Experimental results for irregular bounded graphs

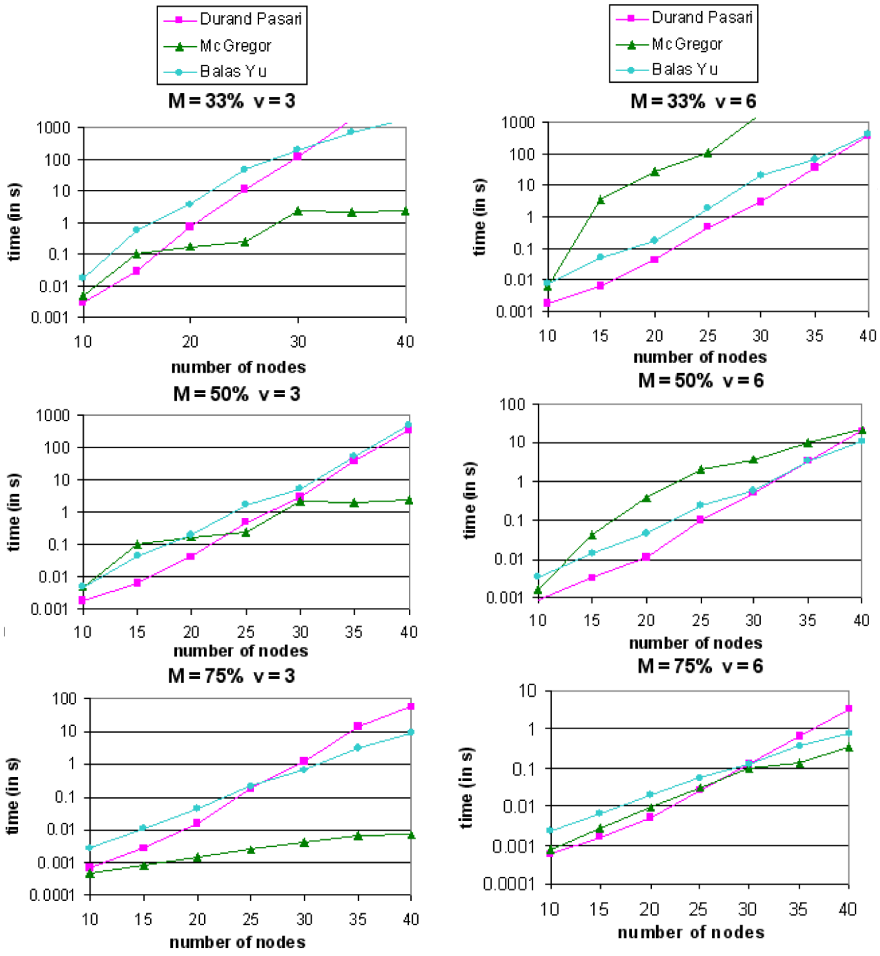


Figure 28: The size of the attribute alphabet is $M = 33\%$, $M = 50\%$ and $M = 75\%$ of the number of nodes, and the valence v is 3 and 6.

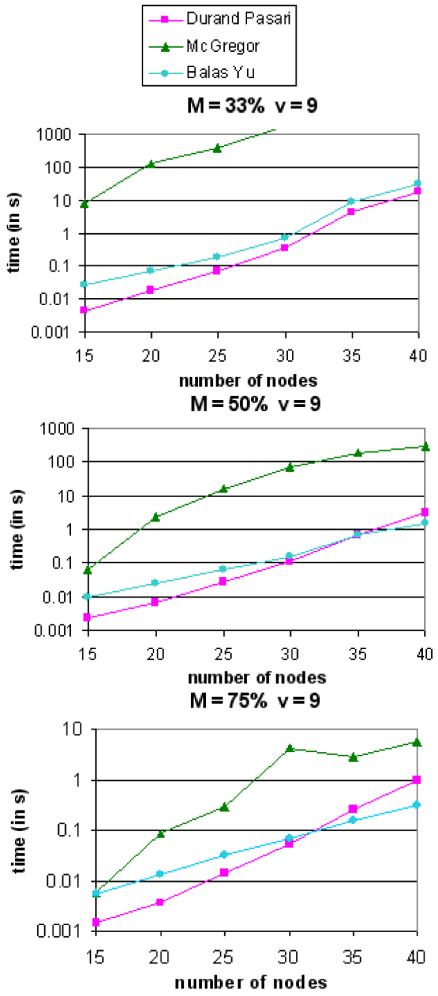


Figure 29: The size of the attribute alphabet is $M = 33\%$, $M = 50\%$ and $M = 75\%$ of the number of nodes, and the valence v is 9.