1367

A (Sub)Graph Isomorphism Algorithm for Matching Large Graphs

Luigi P. Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento

Abstract—We present an algorithm for graph isomorphism and subgraph isomorphism suited for dealing with large graphs. A first version of the algorithm has been presented in a previous paper, where we examined its performance for the isomorphism of small and medium size graphs. The algorithm is improved here to reduce its spatial complexity and to achieve a better performance on large graphs; its features are analyzed in detail with special reference to time and memory requirements. The results of a testing performed on a publicly available database of synthetically generated graphs and on graphs relative to a real application dealing with technical drawings are presented, confirming the effectiveness of the approach, especially when working with large graphs.

Index Terms—Graph-subgraph isomorphism, large graphs, attributed relational graphs.

---

1 INTRODUCTION

IN the last years, the scientific community active in the fields of pattern analysis, pattern recognition, and computer vision has considered graphs with increasing interest, and the applications employing graphs have multiplied. Graphs are commonly used for providing structural descriptions of images by decomposing them into parts and associating graph nodes and branches to components and their relationships. Handwritten characters, ideograms, and symbols in documents and 3D scenes, just to mention a few examples, have been described in this way [1], [2], [3], [4], [5].

From the point of view of pattern analysis and recognition, the most important problem of graph processing is matching graphs or subgraphs for comparing them. An extensive review of graph matching algorithms for pattern recognition has been recently made [6]. In exact graph matching, a strict correspondence between the two graphs is sought; another basic research problem, called inexact matching, concerns the extension of the matching concepts to the case in which similarity between two graphs, not their exact correspondence, is of interest [2]. The most common inexact algorithms (error-correcting algorithms) use a set of editing operations, such as node and branch insertion, deletion, or substitution, in order to find an exact matching between the two graphs [7], [8]. Algorithms evaluating the distance between graphs in order to estimate their degree of similarity have also been proposed (e.g., see [9]).

In this paper, the attention will be devoted to exact matching. Besides being part of error correcting matching procedures, exact matching may be of interest in different pattern analysis and recognition contexts, thus deserving attention by the pattern recognition community.

As it is well-known, among the different types of graph matching (monomorphism, isomorphism, graph subgraph isomorphism) subgraph isomorphism is a NP-complete problem, while it is still an open question if also graph isomorphism is a NP-complete problem. The exponential time requirement of matching algorithms has been the main impediment for applications requiring graphs of large size (hundreds or thousands of nodes).

Low complexity algorithms suited for matching large graphs have been a subject of research during the last three decades. Some of the proposed algorithms reduce the overall computational complexity of the matching process by imposing restrictions on the graphs (e.g., polynomial algorithms for trees, planar graphs, or bounded valence graphs [10]).

An alternative approach is that of using an adequate representation of the searching process and pruning unprofitable paths in the search space, without imposing any restriction on the graph structure.

A procedure that significantly reduces the size of the search space is the backtracking algorithm proposed by Ullmann [11]. This algorithm, devised for both graph isomorphism and subgraph isomorphism, is still today one of the most commonly used for exact graph matching. In [12], it is compared with other algorithms, resulting the most convenient in terms of matching time, in case of one-to-one matching. During the process, the algorithm allows the integrated comparison of semantic information. For the above reasons, in the following, we will systematically compare our results with those of the Ullmann's algorithm.

Among graph isomorphism algorithms, it is also necessary to mention the Nauty algorithm [13], which transforms the graphs to be matched to a canonical form before checking for the isomorphism. Even if it is considered one of the fastest graph isomorphism algorithms available, it has been shown that there are categories of graphs for which it employs exponential time. Furthermore, it cannot be used for solving the graph-subgraph isomorphism problem.

A rather recent method [14] attempts to reduce the overall computational cost when matching a sample graph against a large set of prototypes, resulting in a quadratic time with respect to graph size, but with an exponential memory requirement and preprocessing time. Other existing techniques, such as nondeterministic ones (e.g., [15]), are so powerful as to reduce the complexity, in most cases, from exponential to polynomial, but are not guaranteed to find an exact and optimal solution.

In this paper, we propose a deterministic matching method for verifying both isomorphism and subgraph isomorphism. The algorithm has general validity since no constraints are imposed on graph topology. A state space representation (SSR) of the matching process is used and a set of five feasibility rules for pruning the search tree are introduced. The adopted representation allows one to simultaneously carry out the syntactic and semantic comparison of the pairs of nodes to be matched. With respect to a preliminary version of the algorithm, described in [16] and referred to as the *VF* algorithm, the main improvement introduced is that the data structures employed during the exploration of the search space are organized in such a way to significantly reduce memory requirements. Thus, the algorithm is suitable for matching graphs with thousands of nodes and branches. An accurate testing has been performed on a publicly available database of synthetically generated graphs [17] and on attributed graphs obtained from a real application in the field of technical drawings. A comparative experimental analysis completes the performance characterization of the algorithm.

The paper is organized as follows: In Section 2, a short description of the improved graph-matching algorithm, named *VF2*, is described and the results of the theoretical analysis of its overall efficiency, in terms of computational and spatial complexity, are presented. Section 3 is devoted to algorithm testing and comparative analysis of the obtained results. Final notes and conclusions are in Section 4.

2 THE VF2 ALGORITHM

A matching process between two graphs $G_1 = (N_1, B_1)$ and $G_2 = (N_2, B_2)$ consists in the determination of a mapping M which associates nodes of G_1 with nodes of G_2 and vice versa, according to

- *L.P. Cordella, P. Foggia, and C. Sansone are with the Dipartimento di Informatica e Sistemistica, Università di Napoli "Federico II" Via Claudio 21, I-80125 Napoli, Italy. E-mail: {cordel, foggiapa, carlosan}@unina.it.*
- *M. Vento is with the Dipartimento di Ingegneria dell'Informazione e di Ingegneria Elettrica, Università di Salerno Via Ponte Don Melillo, 1 I-84084, Fisciano (SA), Italy. E-mail: mvento@unisa.it.*

```
PROCEDURE Match(s)
    INPUT:  an intermediate state s; the initial state s₀ has M(s₀)=∅
    OUTPUT: the mappings between the two graphs

    IF M(s) covers all the nodes of G₂ THEN
      OUTPUT M(s)
    ELSE
      Compute the set P(s) of the pairs candidate for inclusion in M(s)
      FOREACH p in P(s)
        IF the feasibility rules succeed for the inclusion of p in M(s) THEN
          Compute the state s´ obtained by adding p to M(s)
          CALL Match(s')
        END IF
      END FOREACH
      Restore data structures
    END IF
END PROCEDURE Match
```

Fig. 1. The VF2 matching algorithm.

some predefined constraints. Generally, the mapping M is expressed as the set of pairs (n, m) (with $n \in G_1$ and $m \in G_2$) each representing the mapping of a node n of G_1 with a node m of G_2. A mapping $M \subset N_1 \times N_2$ is said to be an isomorphism iff M is a bijective function that preserves the branch structure of the two graphs. A mapping $M \subset N_1 \times N_2$ is said to be a graph-subgraph isomorphism iff M is an isomorphism between G_2 and a subgraph of G_1.

The process of finding the mapping function can be suitably described by means of a State Space Representation (SSR) [18]. Each state s of the matching process can be associated to a partial mapping solution $M(s)$, which contains only a subset of M. $M(s)$ univocally identifies two subgraphs of G_1 and G_2, say $G_1(s)$ and $G_2(s)$, obtained by selecting from G_1 and G_2 only the nodes included in $M(s)$, and the branches connecting them. In the following, we will denote by $M_1(s)$, $M_2(s)$, $B_1(s)$, and $B_2(s)$ the sets of nodes of $G_1(s)$ and $G_2(s)$ and the corresponding branches.

According to these definitions, a transition from a generic state s to a successor s' represents the addition to the partial graphs associated to s in the SSR, of a pair (n, m) of matched nodes.

Among all the possible SSR states, only a small subset is *consistent* with the wanted morphism type, in the sense that there are no conditions that preclude the possibility of reaching a complete solution. It can be proven that the consistency condition, in case of isomorphism or graph subgraph isomorphism, is that the partial graphs $G_1(s)$ and $G_2(s)$ associated to $M(s)$ are isomorphic.

Our algorithm introduces a set of rules able to verify the consistency conditions, making possible the generation of consistent states only. Moreover, the number of states generated in the process can be further reduced by adding a set of rules (that we call *k-look-ahead rules*) for checking in advance if a consistent state s has no consistent successors after k steps.

Hereinafter, all the mentioned rules will be called *feasibility rules*. For the sake of convenience, let us introduce the so-called *feasibility function* $F(s, n, m)$, which is true if the addition to a state s of the pair (n, m) satisfies all the feasibility rules.

The above feasibility rules depend only on the structure of the input graphs. However, if the input graphs have node and branch attributes, they also must be taken into account. Thus, the most general form of the feasibility function is:

$$F(s, n, m) = F_{\text{syn}}(s, n, m) \wedge F_{\text{sem}}(s, n, m), \qquad (1)$$

where F_{syn} (*syntactic* feasibility) depends only on the structure of the graphs, and F_{sem} (*semantic* feasibility) depends on the attributes.

A high-level description of the algorithm we propose is outlined in Fig. 1. In the initial state s_0, the mapping function does not contain any component, i.e., $M(s_0) = \emptyset$. For each intermediate state s, the algorithm computes the set $P(s)$ (see next section for more details) of the node pairs that are candidate to be added to the current state s. For each pair p belonging to $P(s)$, the

feasibility rules are evaluated; if they succeed, i.e., $F(s, n, m)$ is true, being $p = (n, m)$, the successor state $s' = s \cup p$ is computed and the whole process recursively applies to s'. Note that the algorithm explores the search graph in the SSR according to a depth-first search strategy. Using this simple formulation, a state can be reached through different paths. In order to avoid that, during the matching process, the algorithm generates useless and already generated states, a special procedure for generating a node successor is used. An arbitrary, total order relation (denoted by $\prec$) is defined on those nodes of G_2 which belong to the set $P(s)$. Since the node insertion order in the partial solution $M(s)$ does not influence the resulting state, the algorithm ignores any pair (n_i, m_j) in $P(s)$ if this set already contains a node $m_k \prec m_j$. This simple strategy allows the algorithm to generate each state only once.

In the following section, we will examine how the set $P(s)$ is defined; then, in Section 2.2, we will address the definition of the feasibility rules.

2.1 Computation of the Candidate Pairs Set $P(s)$

The set $P(s)$ of all the possible pairs candidate to be added to the current state is obtained by considering first the sets of the nodes directly connected to $G_1(s)$ and $G_2(s)$. Let us denote with $T_1^{out}(s)$ and $T_2^{out}(s)$ the sets of nodes, not yet in the partial mapping, that are the destination of branches starting from $G_1(s)$ and $G_2(s)$, respectively; similarly, with $T_1^{in}(s)$ and $T_2^{in}(s)$, we will denote the sets of nodes, not yet in the partial mapping, that are the origin of branches ending into $G_1(s)$ and $G_2(s)$.

The set $P(s)$ will be made of all the node pairs (n, m), with n belonging to $T_1^{out}(s)$ and m to $T_2^{out}(s)$, unless one of these two sets is empty. In this case, the set $P(s)$ is likewise obtained by considering $T_1^{in}(s)$ and $T_2^{in}(s)$, respectively. In presence of not connected graphs, for some state s, all of the above sets may be empty. In this case, the set of candidate pairs making up $P(s)$ will be the set $P^d(s)$ of all the pairs of nodes not contained neither in $G_1(s)$ nor in $G_2(s)$.

2.2 Feasibility Rules

Five feasibility rules are defined: R_{pred}, R_{succ}, R_{in}, R_{out}, and R_{new}. The first two rules check the consistency of the partial solution $M(s')$ obtained by adding the considered candidate pair (n, m) to the current partial solution $M(s)$. The remaining three rules are introduced for pruning the search tree; in particular, R_{in} and R_{out} perform a 1-look-ahead in the searching process, and R_{new} a 2-look-ahead. In conclusion, the proposed feasibility function is:

$$F_{\text{syn}}(s, n, m) = R_{\text{pred}} \wedge R_{\text{succ}} \wedge R_{\text{in}} \wedge R_{\text{out}} \wedge R_{\text{new}}. \qquad (2)$$

Given a graph $G = (N, B)$ and a node $n \in N$, the sets, respectively, containing the predecessors and the successors of n will be denoted by $\text{Pred}(G, n)$ and $\text{Succ}(G, n)$. Also, in the following definitions, we will use the sets $T_1(s) = T_1^{in}(s) \cup T_1^{out}(s)$ and

$\tilde{N}_1(s) = N_1 - M_1(s) - T_1(s)$. Similar expressions hold for T_2 and $\tilde{N}_2$. Here, are the formal definitions of the five rules for the subgraph isomorphism case:

$$R_{\text{pred}}(s, n, m) \Longleftarrow\Longrightarrow$$
$$(\forall n' \in M_1(s) \cap \text{Pred}(G_1, n) \exists m' \in \text{Pred}(G_2, m) \mid (n', m') \in M(s)) \wedge$$
$$(\forall m' \in M_2(s) \cap \text{Pred}(G_2, m) \exists n' \in \text{Pred}(G_1, n) \mid (n', m') \in M(s)),$$
$$(3)$$

$$R_{\text{succ}}(s, n, m) \Longleftarrow\Longrightarrow$$
$$(\forall n' \in M_1(s) \cap \text{Succ}(G_1, n) \exists m' \in \text{Succ}(G_2, m) \mid (n', m') \in M(s)) \wedge$$
$$(\forall m' \in M_2(s) \cap \text{Succ}(G_2, m) \exists n' \in \text{Succ}(G_1, n) \mid (n', m') \in M(s)),$$
$$(4)$$

$$R_{\text{in}}(s, n, m) \Longleftarrow\Longrightarrow$$
$$(\text{Card}(\text{Succ}(G_1, n) \cap T_1^{\text{in}}(s)) \geq \text{Card}(\text{Succ}(G_2, m) \cap T_2^{\text{in}}(s))) \wedge \quad (5)$$
$$(\text{Card}(\text{Pred}(G_1, n) \cap T_1^{\text{in}}(s)) \geq \text{Card}(\text{Pred}(G_2, m) \cap T_2^{\text{in}}(s))),$$

$$R_{\text{out}}(s, n, m) \Longleftarrow\Longrightarrow$$
$$(\text{Card}(\text{Succ}(G_1, n) \cap T_1^{\text{out}}(s)) \geq \text{Card}(\text{Succ}(G_2, m) \cap T_2^{\text{out}}(s))) \wedge \quad (6)$$
$$(\text{Card}(\text{Pred}(G_1, n) \cap T_1^{\text{out}}(s)) \geq \text{Card}(\text{Pred}(G_2, m) \cap T_2^{\text{out}}(s))),$$

$$R_{\text{new}}(s, n, m) \Longleftarrow\Longrightarrow$$
$$\text{Card}(\tilde{N}_1(s) \cap \text{Pred}(G_1, n)) \geq \text{Card}(\tilde{N}_2(s) \cap \text{Pred}(G_2, n)) \wedge \quad (7)$$
$$\text{Card}(\tilde{N}_1(s) \cap \text{Succ}(G_1, n)) \geq \text{Card}(\tilde{N}_2(s) \cap \text{Succ}(G_2, n)).$$

Considering the graph isomorphism instead of the subgraph isomorphism, the rules R_{succ} and R_{pred} maintain the same form, while in rules R_{in}, R_{out}, and R_{new}, the $\geq$ operator must be substituted by $=$.

2.3 Semantic Feasibility

When we turn our attention to attributed graphs, the inclusion of node/branch attributes in the matching algorithm can be performed in two ways, depending on whether we have symbolic or (real-valued) numeric attributes. For symbolic attributes that, in some cases, are derived from numeric attributes through a quantization process, we suppose that a compatibility relation $\approx$ is defined between two node/branch attributes. For some applications, $\approx$ may coincide with the equality relation, while in other cases a more "tolerant" definition may be necessary. Each time we check the feasibility of a new pair, the attributes of the nodes and branches being added are tested for semantic compatibility. Formally, we can define:

$$F_{sem}(s, n, m) \Longleftarrow\Longrightarrow n \approx m$$
$$\wedge \forall (n', m') \in M(s), (n, n') \in B_1 \Rightarrow (n, n') \approx (m, m') \quad (8)$$
$$\wedge \forall (n', m') \in M(s), (n', n) \in B_1 \Rightarrow (n', n) \approx (m', m).$$

For numeric attributes, we exploit this information in two ways. First, a compatibility relation is defined on the basis of a thresholding on the absolute difference of the attributes being matched, leading to a semantic feasibility function analogous to (8). Furthermore, a cost function is introduced to give a quantitative evaluation of the dissimilarity between two nodes or branches. The algorithm, in its exploration of the search space, saves only the matching that obtains the minimum total cost. The cost is actually computed for each state s, as the sum of the cost of its parent state and of the costs due to the newly added nodes and branches. Since these latter are assumed to be not negative, the total cost of a state will be greater than or equal to the costs of all its ancestors. We can use this information to prune all the states whose cost is greater than the cost of the best goal state reached so far, further reducing the search space.

2.4 Data Structures and Implementation Issues

In order to make the algorithm run with an acceptable time and space complexity also on large graphs, it is important to employ well-devised data structures for performing the computation of $P(s)$ and of $F(s, n, m)$. In the actual implementation, the following data structures are used:

- Two vectors, core_1 and core_2, whose dimensions correspond to the number of nodes in G_1 and G_2, respectively, containing the current mapping; in particular, core_1[n] contains the index of the node paired with n, if n is in $M_1(s)$, and the distinguished value NULL_NODE otherwise. The same encoding is used for core_2.
- Four vectors, in_1, out_1, in_2, out_2, whose dimensions are equal to the number of nodes in the corresponding graphs, describing the membership of the terminal sets. In particular, in_1[n] is nonzero if n is either in $M_1(s)$ or in $T_1^{in}(s)$; similar definitions hold for the other three vectors. The actual value stored in the vectors is the depth in the SSR tree of the state in which the node entered the corresponding set.

Using the vectors described above, the tests for the membership of the various sets require a constant time. It follows that the computation of $P(s)$ can be done in a time in the worst case proportional to $|N_1| + |N_2|$, while the computation of $F(s, n, m)$ can be performed in a time proportional to the number of the branches involving n and m.

It is important to note that all the vectors have the following property: If an element is nonnull in a state s, it will remain nonnull in all the states descending from s. This property, together with the depth-first strategy of the search, is used to avoid the need to store a different copy of the vectors for each state: When the algorithm backtracks, it restores the previous value of the vectors. The memory requirement, with respect to the number of nodes N, is quite lower than in other similar algorithms. In fact, except for the six vectors, that are shared among the states, each state needs a constant (and small) amount of memory, and the depth-first search strategy ensures that there can be at most N states in memory at a time. It follows that the memory required is $\Theta(N)$, with a small constant factor. Table 1 summarizes the time and spatial complexity of our algorithm compared with that of Ullmann's Algorithm as can be deduced from [11] and [12], in the best and worst case. The analytical estimation of the computational complexity in the average case is not a simple task unless some very restrictive assumptions are made. For this reason, we have performed a set of tests aimed at evaluating the average time required by the matching in case of both isomorphism and graph-subgraph isomorphism, as reported in the following section.

Time complexity has been obtained considering that in the best case our algorithm visits N states, while in the worst case $N!$ states need to be explored.

3 EXPERIMENTAL RESULTS

We have systematically compared the results of the VF2 algorithm with those obtained on the same data by Ullmann's Algorithm, for the reasons mentioned in Section 1. Although an implementation of the latter algorithm was already available on the Web (ftp://ftp.iam.unibe.ch/pub/Tools/GUB_toolkit.tar.Z), for our testing we have developed a more effective code and made it also available on the Web (http://amalfi.dis.unina.it/graph/). Moreover, as regards the isomorphism case, we also compared the results obtained by Ullmann's Algorithm and by ours with those obtained by the Nauty Algorithm. In particular, in our tests, we used the version 2.0b9 of the Nauty Algorithm made available by B.D. McKay at the URL: http://cs.anu.edu.au/~bdm/nauty.

For the isomorphism case, the database used for testing algorithms' performance was made of 10,000 couples of isomorphic graphs: This is part of a wider database of synthetically

TABLE 1
Spatial and Time Complexity of VF2 and of Ullmann's Algorithm in the Best and Worst Case

	VF2		Ullmann's algorithm	
Complexity	Best Case	Worst Case	Best Case	Worst Case
Time	$\Theta(N^2)$	$\Theta(N!\,N)$	$\Theta(N^3)$	$\Theta(N!\,N^2)$
Spatial	$\Theta(N)$	$\Theta(N)$	$\Theta(N^3)$	$\Theta(N^3)$

TABLE 2
A Comparison between VF2 and Nauty as a Function of the Graph Size and of the Kind of the Graphs

Nodes	Randomly Connected			2D (regular and irregular) Mesh				Bounded valence		
	η=0.01	η=0.05	η=0.1	regular	ρ=0.2	ρ=0.4	ρ=0.6	v=3	v=6	v=9
20	VF2	VF2	Nauty	VF2	Nauty	Nauty	Nauty	Nauty	Nauty	Nauty
40	VF2	Nauty	Nauty	VF2	VF2	Nauty	Nauty	Nauty	Nauty	Nauty
60	VF2	Nauty	Nauty	VF2	VF2	VF2	Nauty	VF2	Nauty	Nauty
80	VF2	Nauty	Nauty	VF2	VF2	VF2	VF2	VF2	Nauty	Nauty
100	VF2	Nauty	Nauty	VF2	VF2	VF2	VF2	VF2	Nauty	Nauty
200	VF2	Nauty	Nauty	VF2	VF2	VF2	VF2	VF2	VF2	Nauty
400	Nauty	Nauty	Nauty	VF2	VF2	VF2	VF2	VF2	VF2	Nauty
600	Nauty	Nauty	Nauty	VF2	VF2	VF2	VF2	VF2	VF2	Nauty
800	Nauty	Nauty	Nauty	VF2	VF2	VF2	VF2	VF2	VF2	VF2
1000	Nauty	Nauty	Nauty	VF2	VF2	VF2	VF2	VF2	VF2	VF2

generated graphs, especially developed for benchmarking purposes [17] and available on the Web (http://www.iapr.org/benchmarks.html). On the other hand, the performance of the graph-subgraph matching algorithm has been evaluated in the context of a real problem: the detection of component parts in large images. Namely, mechanical line drawings and topographic maps have been considered.

3.1 Graph Isomorphism

The performance of the three algorithms has been evaluated on the following kinds of graphs: *randomly connected graphs* (3,000 couples), *regular and irregular 2D meshes* (4,000 couples), and *bounded valence graphs* (3,000 couples). Each category contains couples of graphs of different size, ranging from a few dozens to about 1,000 nodes. For each size and kind of graph, 100 different couples have been considered. In the following, a brief description of each category is given.

Randomly connected graphs are graphs where edges connect nodes without any structural regularity. To generate these graphs, we have adopted the same model proposed in [11]: It fixes the value η of the probability that an edge is present between two distinct nodes. The probability distribution is assumed to be uniform, and the edges are independent.

2D mesh graphs are considered for simulating applications dealing with regular structures as those operating at the lower levels of a vision task. The considered meshes are 4-connected (i.e., each node is connected only with the nodes at north, south, east, and west) by directed edges.

Irregular 2D meshes have been introduced for simulating the behavior of the algorithms in presence of slightly distorted meshes. These have been obtained from regular 2D meshes by the addition of ρN edges (where ρ is a positive constant), each connecting nodes that have been randomly determined according to a uniform distribution.

Bounded valence graphs model those applications in which each object (i.e., a node) establishes a fixed number of relations (edges)

with other objects. Three different values of the valence v (3, 6, and 9) have been considered.

Table 2 summarizes the obtained results,[1] showing the algorithm that achieves the best performance for each combination of graph size and type. From the table, it results that VF2 performs better on 56 of the 100 considered combinations, while on the remaining 44 cases the best algorithm is Nauty. Moreover, albeit it is not evident from the table, in the cases in which Nauty obtains the best performance, VF2 is always the second best; on the other hand, there are six cases in which both VF2 and Ullmann's algorithms outperform Nauty.

From the analysis of the table, it appears that Nauty is more convenient on randomly connected graphs that exhibit no regular structure, especially when the edge density becomes high. This kind of graph, anyway, does not adequately represent the graph structures found in many applications, where the graphs often show some form of regularity. On the other hand, for graphs with a more regular structure, VF2 is more efficient, especially for large graph sizes.

3.2 Graph-Subgraph Isomorphism of Attributed Graphs

In order to test our algorithm in the context of a graph-subgraph isomorphism application, we have employed a set of attributed graphs derived from large line drawings according to a method described in [20]. In particular, we have used two publicly available images,[2] respectively, representing a mechanical drawings (ENGINE-2) and a cadastral map (MAP-1). From each image, a set of subimages corresponding to the image connected components was extracted and represented as graphs. Features of the obtained graphs and subgraphs are shown in Table 3. Fig. 2 shows the MAP-1 image with some of the component parts

---

1. Further experimental results can be found at the URL: http://amalfi.dis.unina.it/graph.
2. A CD-ROM with the images was made available by M. Burge and W. Kropatsch during the SSPR workshop held in Sidney in August 1998.

TABLE 3
Some Features of Graphs and Subgraphs Obtained from the Test Images

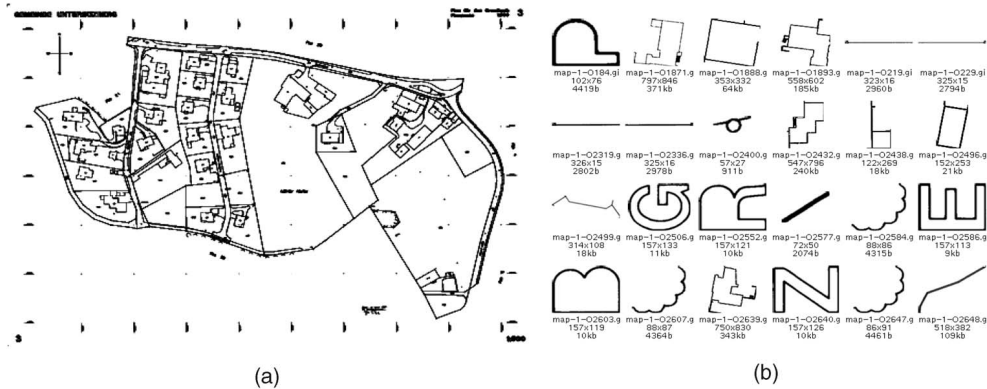| Image	Type	No. of Graph Nodes	No. of Graph Branches	No. of Subgraphs	No. of Subgraph Nodes
MAP-1	cadastral map	4484	4968	163	2-233
ENGINE-2	mech. drawing	1953	2169	120	2-542

Fig. 2. (a) The MAP-1 image and (b) some of the parts used as subgraphs.
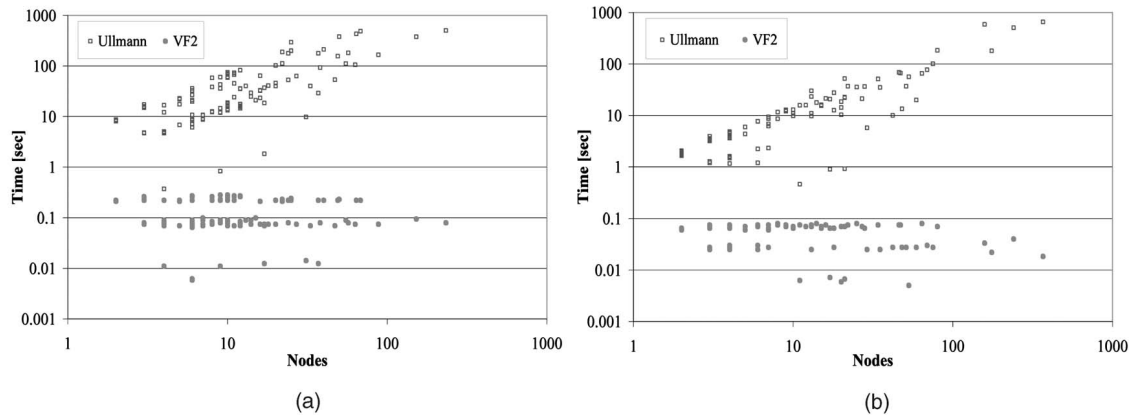

Fig. 3. Matching times for the subgraphs of the MAP-1 image (a) and of the ENGINE-2 image (b).

considered. In the adopted encoding of the images, graph branches represent strokes and graph nodes represent stroke junctions or end points. Nodes and branches are labeled with numeric attributes characterizing absolute position of the points and shape and orientation of the strokes connecting them. For our test, we have neglected node attributes and used stroke length and orientation as branch attributes.

A simple semantic feasibility rule has been defined, allowing two branches to match if they are roughly similar. A semantic feasibility rule requiring the equality of the corresponding attributes would have been of course more effective in reducing the matching effort, but, as already mentioned, an inexact rule provides a more realistic estimate of the algorithm behavior in real applications. The feasibility rule assumes that two branches are similar if their lengths differ by less than 30 percent and their orientations differ by less than 30 degrees.

The matching times of the VF2 algorithm have been compared with those obtained with Ullmann's Algorithm, modified so as to take into account the same semantic feasibility rule. Fig. 3 reports the performance of the two algorithms on the two considered images. It can be seen that our algorithm performs significantly better, especially when the size of the subgraphs is over about 20

nodes. In fact, while the matching time for Ullmann's Algorithm rises rapidly with the number of subgraph nodes, the time needed by our algorithm is almost independent of the number of nodes. The time ratio reaches four orders of magnitude for subgraphs of more than 100 nodes.

4   CONCLUSIONS

We have presented and evaluated, both analytically and experimentally, a graph matching algorithm, whose computational complexity is reduced, due to the use of a set of feasibility rules during the matching process. The algorithm is tailored for dealing with large graphs without making particular assumptions on the nature of the graphs to be matched and can be used for both isomorphism and graph-subgraph isomorphism. Another distinctive feature of the algorithm is its ability to deal also with Attributed Relational Graphs, profitably exploiting the information held by the semantic part of the ARG in order to further reducing the matching time. The achievement seems of particular interest since almost all of the algorithms presented in the literature till now do not satisfy all the above requirements together.

REFERENCES

[1] I. Rocha and T. Pavlidis, "A Shape Analysis Model with Application to a Character Recognition System," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 16, pp. 393-404, 1994.

[2] L.G. Shapiro and R.M. Haralick, "Structural Description and Inexact Matching," *IEEE Trans. Pattern Analysis and Machine Intelligence,* no. 3, pp. 505-519, 1981.

[3] L.P. Cordella and M. Vento, "Symbol Recognition in Documents: A Collection of Techniques?" *Int'l J. Document Analysis and Recognition,* vol. 3, pp. 73-88, 2000.

[4] L. Jianzhuang and L.Y. Tsui, "Graph-Based Method for Face Identification from a Single 2D Line Drawing," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 23, no. 10, pp. 1106-1119, 2001.

[5] J. Llados, E. Marti, and J.J. Villanueva, "Symbol Recognition by Error-Tolerant Subgraph Matching between Region Adjacency Graphs," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 23, no. 10, pp. 1137-1143, 2001.

[6] D. Conte, P. Foggia, C. Sansone, and M. Vento, "Thirty Years of Graph Matching in Pattern Recognition," *Int'l J. Pattern Recognition and Artificial Intelligence,* vol. 18, no. 3, pp. 265-298, 2004.

[7] L.P. Cordella, P. Foggia, C. Sansone, and M. Vento, "Subgraph Transformations for the Inexact Matching of Attributed Relational Graphs," *Computing,* vol. 12, pp. 43-52, 1998.

[8] W.H. Tsai and K.S. Fu, "Subgraph Error-Correcting Isomorphisms for Syntactic Pattern Recognition," *IEEE Trans. Systems, Man, and Cybernetics,* vol. 13, pp. 48-62, 1983.

[9] L.G. Shapiro and R.M. Haralick, "A Metric for Comparing Relational Descriptions," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 7, pp. 90-94, 1985.

[10] E.M. Luks, "Isomorphism of Graphs of Bounded Valence can be Tested in Polynomial Time," *J. Computer System Science,* pp. 42-65, 1982.

[11] J.R. Ullmann, "An Algorithm for Subgraph Isomorphism," *J. Assoc. for Computing Machinery,* vol. 23, pp. 31-42, 1976.

[12] B.T. Messmer, "Efficient Graph Matching Algorithms for Preprocessed Model Graphs," PhD Thesis, Inst. of Computer Science and Applied Mathematics, Univ. of Bern, 1996.

[13] B.D. McKay, "Practical Graph Isomorphism," *Congressus Numerantium,* vol. 30, pp. 45-87, 1981.

[14] H. Bunke and B.T. Messmer, "Efficient Attributed Graph Matching and Its Application to Image Analysis," *Proc. Image Analysis and Processing,* pp. 45-55, 1995.

[15] W.J. Christmas, J. Kittler, and M. Petrou, "Structural Matching in Computer Vision Using Probabilistic Relaxation," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 17, no. 8, pp. 749-764, 1995.

[16] L.P. Cordella, P. Foggia, C. Sansone, and M. Vento, "Evaluating Performance of the VF Graph Matching Algorithm," *Proc. 10th Int'l Conf. Image Analysis and Processing,* pp. 1172-1177, Sept. 1999.

[17] P. Foggia, C. Sansone, and M. Vento, "A Database of Graphs for Isomorphism and Sub Graph Isomorphism Benchmarking," *Proc. Third IAPR TC-15 Int'l Workshop Graph Based Representations,* pp. 176-188, 2001.

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

[19] B.T. Messmer and H. Bunke, "A Decision Tree Approach to Graph and Subgraph Isomorphism Detection," *Pattern Recognition,* vol. 32, pp. 1979-1998, 1999.

[20] M. Burge and W.G. Kropatsch, "A Minimal Line Property Preserving Representation for Line Images," *Computing,* vol. 62, no. 4, pp. 355-368, 1999.

▷ **For more information on this or any computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.