Fast Graph Matching for Detecting CAD Image Components

L.P. Cordella, P. Foggia, C. Sansone, M. Vento
Dipartimento di Informatica e Sistemistica – Via Claudio, 21 - I-80125 Napoli (Italy)
E-mail: {cordel,foggiapa,carlosan,vento}@unina.it – http://amalfi.dis.unina.it

Abstract

The performance of an Attributed Relational Graph (ARG) matching algorithm[1], tailored for dealing with large graphs, is evaluated in the context of a real application: the detection of component parts in CAD images of mechanical drawings. The matching problem is a graph-subgraph isomorphism and the algorithm exploits semantic information about nodes while does not require information about the topology of the graphs to be matched.

Experimental results, compared with those obtained with a different method, show the overall efficiency of the algorithm and the matching time reduction obtainable by exploiting the semantic information held by ARGs.

1. Introduction

Graphs are data structures widely used for representing information both in low-level and high-level vision tasks. We consider Attributed Relational Graphs (ARG's) since they represent a very general type of graph and structural descriptions of visual patterns can be conveniently put in form of ARG's [1]. ARG's provide both syntactic information, held by the layout of unlabeled nodes and branches respectively identifying the structural primitive components of the pattern and their relations, and semantic information, consisting of the attributes associated with nodes and branches.

One of the most relevant problems, dealing with graphs, is that of matching a sample graph against a reference graph; to this concern, a main issue consists in limiting the computational cost of the matching algorithm.

Purpose of this paper is that of experimentally evaluating the performance of an ARG matching algorithm [2] when dealing with large graphs (thousands of nodes). It is shown that a significant reduction of the matching time can be achieved by exploiting the semantic information held by the ARGs. Such results are also compared with those achieved with one of the most commonly used graph matching algorithm, which was first described by J. R. Ullmann in [3].

A theoretical analysis of the overall efficiency of our algorithm was already presented in [4], with reference to the case of graph isomorphism and without taking into account the semantics held by the ARG.

In the literature there are plenty of graph matching algorithms that lower the overall computational complexity of the matching process by taking into account some peculiar topological properties of the graphs to be matched. For instance, algorithms for special classes of graphs, like planar graphs [5] or trees [6] have been proposed. Despite their efficiency, their applicability is obviously restricted to those domains that satisfy the hypotheses on which the algorithms were based.

The choice of the Ullmann's algorithm for comparison is due to the fact that both ours and the Ullmann's algorithm do not rely on any special topological property of the graphs and allow us to find all graph or graph-subgraph isomorphisms between two given graphs.

Other methods (e.g., [7],[8]), although in some cases perform faster than the Ullmann's algorithm, do not fit well with our intended application and thus have not been considered for comparison. In [7] suitable transformations are applied to the graphs in order to find a different representation for which the matching is easier. However, it is not evident if it is possible to take advantage, in the matching process, of the semantic information provided by the ARG's. In [8] the maximum graph size that can be dealt with is limited to about twenty nodes.

In the next section we will briefly summarize our graph matching algorithm. Section 3 is devoted to the experimental analysis of the algorithm, with reference to a graph-subgraph isomorphism problem of interest in the field of technical line drawing analysis. It is shown how, using a set of feasibility rules and exploiting the semantics of the ARG, a significant reduction of the computational cost of the matching process can be achieved. The results are compared with those obtained by using the Ullmann's algorithm modified so as to exploit semantic information.

2. The matching algorithm

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

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

[1]The program code is available on the WEB at the address: http://amalfi.dis.unina.it/graph/

0-7695-0750-6/00 $10.00 © 2000 IEEE

the mapping of a node n of G_1 with a node m of G_2:

$$M = \{(n,m) \in N_1 \times N_2 \mid n \text{ is mapped onto } m\} \quad (1)$$

The State Space Representation (SSR from now on) can be effectively used to describe a graph matching process, if each state s of the SSR represents a partial mapping solution achieved during the matching process. A partial mapping solution $M(s)$ is a subset of M, i.e. contains only some components of M. $M(s)$ univocally identifies also 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 the components of $M(s)$ and the branches connecting them.

In the adopted SSR, a transition between two states corresponds to the addition of a new pair of matched nodes. In principle, the solutions to the matching problem could be obtained by computing all the possible partial solutions and selecting the ones satisfying the wanted mapping type (Brute Force approach). In order to reduce the number of paths to be explored during the search, we impose that for each state on the path from s_0 to a goal state, the corresponding partial solution verifies some consistency conditions, depending on the desired mapping type. For example, to have a subgraph isomorphism, it is necessary that the corresponding subgraphs in the partial mappings are isomorphic. If the addition of a node pair to the partial mapping produces a state that does not meet the consistency condition, further exploration of that path can be avoided, since it certainly cannot lead to a goal state.

Our algorithm introduces a set of rules able to verify the consistency conditions, making so possible the generation of only consistent states. Although the computational complexity of the matching process is consequently reduced, our aim is that of further reducing the number of states generated in the matching process. So, in addition to the consistency rules, a set of further rules is introduced in order to foresee if a state s has no consistent successors after a certain number of steps. Hereinafter, they will be called *feasibility rules*.

For the sake of convenience, let us introduce a boolean function, $F(s,n,m)$, called *feasibility function*, which is true if, given a state s and a pair (n,m) candidate to be added to s, all the feasibility rules are satisfied.

The most general form of the function $F(s,n,m)$ is:

$$F(s,n,m) = F_{syn}(s,n,m) \wedge F_{sem}(s,n,m) \quad (2)$$

F_{syn} is a boolean function, called *syntactic* feasibility function, which depends only on the structure of the graphs. The expression of the function $F_{syn}(s,n,m)$ has been given in [2] where five syntactic feasibility rules have been proposed. These rules allow the algorithm to prune the SSR search graph solely on the basis of the structure of the two graphs being compared.

F_{sem}, called semantic feasibility function, is a boolean function which is satisfied if suitable compatibility conditions defined for the set of attributes hold. Such conditions are strongly dependent not only on the type of the attributes, but also on the application requirements. For some applications they can impose strict equality between attributes, while in other cases a similarity relation must be accepted for coping with the variability of the attribute values, e.g. due to noise. In the following the compatibility relation between two node or branch attributes will be denoted by $\approx$.

Exploiting the semantic information associated with ARG's nodes and branches allows a further reduction of the search cost of a matching. In fact, in this way it is possible to exclude those subgraph mappings implying a correspondence between nodes or branches that, although syntactically equivalent, are not compatible from a semantic point of view. Semantic compatibility can be easily taken into account in the matching process, once some semantic feasibility rules have been defined. Each time a node of the graph G_1 is compared with a node of G_2, to determine if a new pair can be added to the current partial solution, the attributes of the two nodes and of the branches linking them to the nodes already in $M(s)$ are tested for semantic compatibility. More synthetically:

$$F_{sem}(s,n,m) \Leftrightarrow n \approx m$$
$$\wedge \forall (n',m') \in M(s), (n,n') \in B_1 \Rightarrow (n,n') \approx (m,m') \quad (3)$$
$$\wedge \forall (n',m') \in M(s), (n',n) \in B_1 \Rightarrow (n',n) \approx (m',m)$$

In the following Section specific semantic feasibility rules will be defined, with reference to an application of the matching algorithm to CAD drawings.

In Fig. 1 our matching algorithm is outlined. It starts from an initial state s_0 in which no component of the mapping function have been determined, i.e. $M(s_0)=\varnothing$.

For each intermediate state s, the algorithm computes the set $P(s)$ of the node pairs that are candidate to be added to s. For each pair p belonging to $P(s)$ the feasibility of its insertion is checked: if the check succeeds (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 generates the SSR according to a depth-first strategy.

```
PROCEDURE Match(s)
  INPUT: a state s; for the initial state s0, it is M(s0)=∅
  OUTPUT: the mappings between the two graphs

  IF M(s) covers all the nodes THEN
    OUTPUT M(s)
  ELSE
    Compute the set P(s) of the pairs candidate for inclusion in M(s)
    FOREACH p ∈ P(s)
      IF the feasibility check for the inclusion of p in M(s) succeeds
        THEN
          Compute the state s' obtained by adding p to M(s)
          CALL Match(s')
      END IF
    END FOREACH
  END IF
END PROCEDURE
```

Fig. 1. The matching algorithm.

Let us denote by $T^{out}_1(s)$ ($T^{in}_1(s)$) and $T^{out}_2(s)$ ($T^{in}_2(s)$) the sets of nodes outgoing from (entering) the two subgraphs $G_1(s)$ and $G_2(s)$, and let us define $T_1(s) = T^{out}_1(s) \cup T^{in}_1(s)$ and $T_2(s) = T^{out}_2(s) \cup T^{in}_2(s)$. $P(s)$ can be now defined as follows: if $T^{out}_1(s)$ and $T^{out}_2(s)$ are not empty, $P(s)$ is made of all the pairs (n,m) where n is the node with the smallest label in $T^{out}_1(s)$ and m is a node belonging to $T^{out}_2(s)$. If $T^{out}_1(s)$ and $T^{out}_2(s)$ are empty, $T^{in}_1(s)$ and $T^{in}_2(s)$ are considered.

3. Experimental results and discussion

In order to test our algorithm in the context of a real application and to evaluate the advantage of using the semantics of the ARGs, we have employed a set of graphs derived from a large line drawing according to a method described in [9]. Namely, we have used the 'Engine-2' image[2] (see fig. 4a), showing the technical drawing of an engine. The image was represented using a Mixed Run-Graph encoding, giving place to a graph with 1953 nodes and 2169 branches. From the image, a set of 119 sub-images (see fig. 4b), corresponding to the image connected components, was extracted. The size of the corresponding subgraphs ranges from 2 to 542 nodes (see Table 1). We have used our graph-subgraph isomorphism algorithm to match each subgraph against the whole image graph. In fact, although the problem could appear as a one-to-many graph isomorphism, for the sake of generality, we treated it as a graph-subgraph isomorphism since, in the general case, an image component of interest could be made of more than one connected component or be part of a connected component.

In the adopted encoding of the image, graph branches represent strokes and graph nodes represent stroke junction or end points. Nodes and branches are labeled with 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 said, 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% and their orientations differ by less than 30°.

The matching times have been compared with those obtained with the Ullman's algorithm modified so as to take into account the same semantic feasibility rules. Figure 2 and 3 report the performance of the two algorithms on the selected image respectively with and without the use of semantic information. It can be seen that, in the former case, a drastic reduction of the matching time is achieved, as summarized in Table 1.

From the above figure it can be noted that our algorithm performs significantly better than Ullman's, especially when the size of the subgraphs is over 20-30 nodes. In fact, while the matching time for the Ullmann's algorithm rises rapidly with the number of nodes, the time needed by our algorithm remains much more stable. The time ratio reaches three orders of magnitude for subgraphs of about 100 nodes. It is worth noting that the matching time obtained with the Ullman's algorithm for the 542 nodes subgraph is not reported in figures 2 and 3. This is due to the fact that, with the used system configuration, such algorithm revealed not able to cope with graphs of this size because of its spatial complexity.

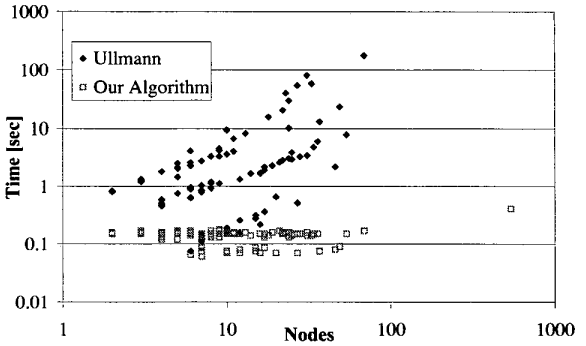

Fig. 2. Matching times for the Engine-2 image using semantic compatibility.

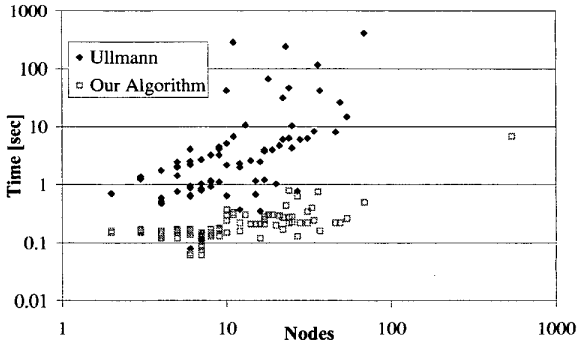

Fig. 3. Matching times for the Engine-2 image without using semantic compatibility.

	N< 10	N >=10 and < 100	N >= 100
# of subgraphs	74	44	1
Relative reduction	0.5%	50.9%	93.9%

Table 1. Relative reduction of the matching times using the semantic compatibility as a function of N

[2]A CD-ROM with the image was made available by M. Burge and W Kropatsch during the SSPR workshop held in Sidney in August 1998.

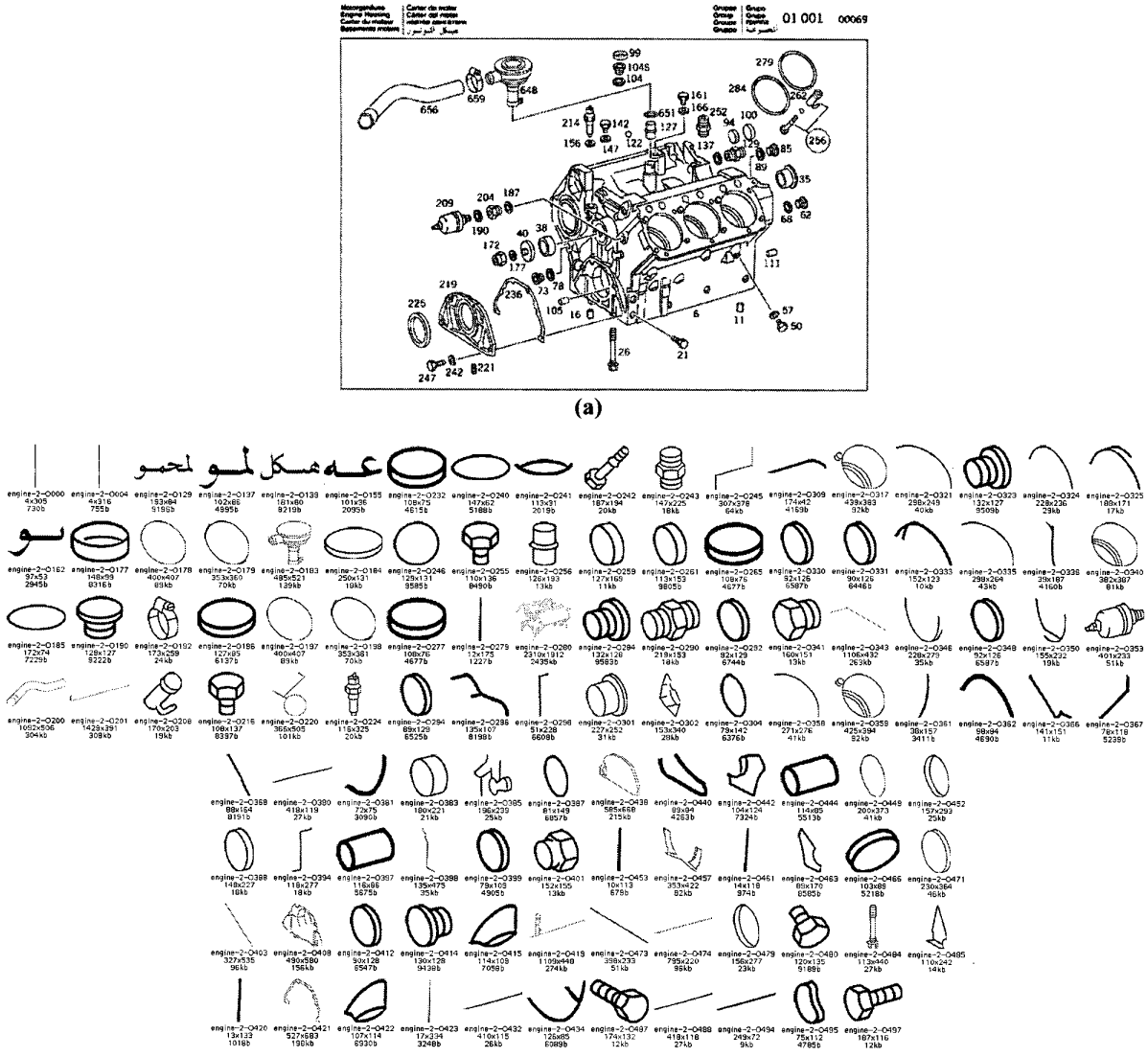

Figure 4 : (a) The ENGINE-2 image and (b) the 119 parts used as subgraphs.

References

[1] L.G. Shapiro, R.M. Haralick, "Structural Description and Inexact Matching", *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 3, pp. 505-519, 1981.

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

[3] J.R. Ullmann, "An Algorithm for Subgraph Isomorphism", *Journal of the ACM*, vol. 23, pp. 31-42, 1976.

[4] L.P. Cordella, P. Foggia, C. Sansone, M. Vento, "Performance evaluation of the VF Graph Matching Algoritmh", *Proc of the 10th ICIAP*, IEEE Computer Society Press, pp. 1172-1177, 1999.

[5] J. Hopcroft, J. Wong, "Linear time algorithm for isomorphism of planar graphs", in *Proc. 6th Annual ACM Symp. Theory of Computing*, pp. 172-184, 1974.

[6] A.V. Aho, J.E. Hopcroft, J.D. Ullman, *The design and analysis of computer algorithms*, Addison Wesley, 1974.

[7] D.G. Corneil, C.C. Gotlieb, "An efficient algorithm for graph isomorphism", *Journal of the ACM*, vol. 17, pp. 51-64, 1970.

[8] B.T. Messmer, H. Bunke, "A decision tree Approach to graph and subgraph isomorphism detection", *Pattern Recognition*, vol. 32, pp. 1979-1988, 1999.

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