An algorithm for graph pattern-matching

Gabriel Valiente[1,2] Conrado Martínez[2]

[1] Universität Bremen
Fachbereich Mathematik und Informatik
D-28334 Bremen, Germany
[2] Technical University of Catalonia
Department of Software
E-08034 Barcelona, Catalonia, Spain

Abstract. Graph pattern-matching is a generalization of string matching and two-dimensional pattern-matching that offers a natural framework for the study of matching problems upon multi-dimensional structures. We present in this paper an algorithm for pattern-matching on arbitrary graphs that is based on reducing the problem of finding a homomorphic image of a pattern graph in a target graph, to that of finding homomorphic images of every connected component of the pattern in the target. For every connected component, the algorithm performs a combinatorial search bounded by a pruning operator. The algorithm can be applied to directed graphs as well as to undirected graphs, and it can also be specialized to find isomorphic images only.

1 Introduction

This paper deals with graph pattern-matching, the problem of finding a homomorphic (or isomorphic) image of a given graph, called the *pattern*, in another graph, called the *target*, and it is also known as the *subgraph homomorphism* (or *subgraph isomorphism*) problem. As a generalization of string matching and two-dimensional pattern-matching, it offers a natural framework for the study of matching problems upon multi-dimensional structures.

A main drawback of graph pattern-matching, however, lies in its inherent computational complexity. The *subgraph isomorphism* problem is known to be NP-complete [6] and, as a matter of fact, a naive graph pattern-matching algorithm, which generates each possible mapping from the n nodes in the pattern to the m nodes in the target and tests whether these mappings are graph homomorphisms, requires in the worst case $O(m^n)$ tests [1, 18].

Most efforts have been then directed at finding efficient pattern-matching algorithms for restricted classes of graphs [3, 5, 7, 8, 9, 10, 12, 15]. On the other hand, effort has been also directed at finding useful heuristics for graph pattern-matching, such as prunning the search tree [16] or the heuristic incorporated in the PROGRES system [18]. The RETE string pattern-matching algorithm has also been generalized to graph grammars [1], but it still requires in the worst case $O(m^n)$ tests, although after an initial match has been found, subsequent graph rewriting followed by graph pattern-matching becomes efficient.

More recently, further efforts have been directed at reducing the pattern-matching problem for graphs to an equivalent pattern-matching problem on smaller *partial subgraphs* of both the source and the target graphs [2, 13].

An algorithm is presented in this paper that is based on decomposing the pattern-matching problem along connected components of the pattern graph, motivated by the fact that if a pattern graph has k connected components, pattern matching along connected components would require in the worst case $O(m^{n/k})$ tests, which is much better than $O(m^n)$ for $k > 1$, since then $m^{n/k} \ll m^n$ for a large n. For every connected component, the algorithm performs a combinatorial search bounded by a pruning operator.

The rest of the paper is organized as follows. Section 2 introduces the notation and terminology used throughout the paper. Section 3 discusses the problem of graph pattern-matching, and presents some theoretical results which allow to develop a decomposition of the graph pattern-matching problem along connected components. Section 4 presents the pattern-matching algorithm in detail. Experimental results are presented in Section 5. Finally, Section 6 presents the main conclusions of this paper.

2 Graphs and graphs homomorphisms

We recall in this section some basic notions, which will be used in the next section to develop an algorithm for graph pattern-matching.

Definition 1. A (directed) graph $G = (V, B)$ consists of a set V and a relation $B \subseteq V \times V$. The elements of V are called nodes, and B is called the relation associated to G. It is said that there is an arc from a node v to a node v' if $(v, v') \in B$. A graph $G = (V, B)$ is finite if the order $|V|$ of G is finite, and it is undirected if B is symmetric.

A finite graph $G = (V, B)$ —actually, the relation B associated to the graph G— can be represented by a boolean matrix A, where an element $a_{i,j} = \mathit{true}$ (one) if $(a_i, a_j) \in B$, and it is *false* (zero) otherwise. We shall usually denote the boolean matrix A and the relation B associated to a graph $G = (V, B)$ by the same name B.

Example 1. The graph $G = (V, B)$ with $V = \{a, b, c, d, e\}$ and

$$B = \{(a, b), (b, b), (c, e), (d, a), (d, b), (e, c)\}$$

can be seen as the following boolean matrix:

$$B = \begin{array}{c|ccccc} & a & b & c & d & e \\ \hline a & 0 & 1 & 0 & 0 & 0 \\ b & 0 & 1 & 0 & 0 & 0 \\ c & 0 & 0 & 0 & 0 & 1 \\ d & 1 & 1 & 0 & 0 & 0 \\ e & 0 & 0 & 1 & 0 & 0 \end{array}$$

Graph homomorphisms are structure-preserving relations over the relations associated to the graphs; cf. [14].

Definition 2. A relation $\Phi \subseteq V \times V'$ is a homomorphism from a graph $G = (V, B)$ to a graph $G' = (V', B')$ if

$$\Phi^T \Phi \subseteq I, \quad I \subseteq \Phi\Phi^T, \quad B \subseteq \Phi B' \Phi^T;$$

and it is also written $\Phi : G \to G'$.

Example 2. The relation

$$\Phi = \{(a, x), (b, y), (c, x), (d, z), (e, z)\}$$

is a homomorphism from the graph $G = (V, B)$ with $V = \{a, b, c, d, e\}$ and $B = \{(a, b), (b, b), (c, e), (d, a), (d, b), (e, c)\}$ to the graph $G' = (V', B')$ with $V' = \{x, y, z\}$ and $B' = \{(x, y), (x, z), (y, y), (z, x), (z, y)\}$:

$$B = \begin{array}{c|ccccc} & a & b & c & d & e \\ \hline a & 0 & 1 & 0 & 0 & 0 \\ b & 0 & 1 & 0 & 0 & 0 \\ c & 0 & 0 & 0 & 0 & 1 \\ d & 1 & 1 & 0 & 0 & 0 \\ e & 0 & 0 & 1 & 0 & 0 \end{array} \qquad \Phi = \begin{array}{c|ccc} & x & y & z \\ \hline a & 1 & 0 & 0 \\ b & 0 & 1 & 0 \\ c & 1 & 0 & 0 \\ d & 0 & 0 & 1 \\ e & 0 & 0 & 1 \end{array} \qquad B' = \begin{array}{c|ccc} & x & y & z \\ \hline x & 0 & 1 & 1 \\ y & 0 & 1 & 0 \\ z & 1 & 1 & 0 \end{array}$$

Proposition 3. *Graph homomorphisms are closed under composition.*

Proof. Let $\Phi : G \to G'$ and $\Phi' : G' \to G''$ be two graph homomorphisms. Then $I \supseteq {\Phi'}^T \Phi' = {\Phi'}^T I \Phi' \supseteq {\Phi'}^T \Phi^T \Phi\Phi' = (\Phi\Phi')^T \Phi\Phi'$; $I \subseteq \Phi\Phi^T = \Phi I \Phi^T \subseteq \Phi\Phi'{\Phi'}^T\Phi^T = \Phi\Phi'(\Phi\Phi')^T$; and $B \subseteq \Phi B' \Phi^T \subseteq \Phi\Phi' B'' {\Phi'}^T \Phi^T = (\Phi\Phi') B'' (\Phi\Phi')^T$. Therefore, $\Phi\Phi' : G \to G''$ is a graph homomorphism.

Definition 4. Let $\Phi : G \to G'$ be an injective graph homomorphism. Then G is called a partial subgraph of G'. It is called a subgraph of G', denoted by $G \subseteq G'$, if $B = \Phi B' \Phi^T$. Moreover, if G is a subgraph of G' and $V \subseteq V'$, then G is also called the subgraph of G' supported on V.

Example 3. Let $G = (V, B)$ be a graph with $V = \{a, b, c, d\}$ and $B = \{(a, c), (b, a), (b, c), (c, d), (d, b)\}$. Then $G' = (\{b, c, d\}, B')$ is a partial subgraph of G, while $G'' = (\{b, c, d\}, B'')$ is the subgraph of G supported on $\{b, c, d\}$:

$$B = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & 0 & 1 & 0 \\ b & 1 & 0 & 1 & 0 \\ c & 0 & 0 & 0 & 1 \\ d & 0 & 1 & 0 & 0 \end{array} \qquad B' = \begin{array}{c|ccc} & b & c & d \\ \hline b & 0 & 1 & 0 \\ c & 0 & 0 & 0 \\ d & 1 & 0 & 0 \end{array} \qquad B'' = \begin{array}{c|ccc} & b & c & d \\ \hline b & 0 & 1 & 0 \\ c & 0 & 0 & 1 \\ d & 1 & 0 & 0 \end{array}$$

Proposition 5. *Given two graphs G and G' such that $G \subseteq G'$, every graph homomorhism $\Phi' : G' \to G''$ to a graph G'' induces a graph homomorphism $\Phi : G \to G''$.*

Proof. It follows from graph homomorphisms being closed under composition. Let $\Phi : G \to G'$ be the inclusion homomorphism of G in G'. Then $\Phi = \Phi'\Phi''$ is a graph homomorphism $\Phi : G \to G''$, by Proposition 3.

Proposition 6. *Given two graphs G' and G'' such that $G' \subseteq G''$, every graph homomorhism $\Phi' : G \to G'$ from a graph G induces a graph homomorphism $\Phi : G \to G''$.*

Proof. It follows from graph homomorphisms being closed under composition. Let $\Phi'' : G' \to G''$ be the inclusion homomorphism of G' in G''. Then $\Phi = \Phi'\Phi''$ is a graph homomorphism $\Phi : G \to G''$, by Proposition 3.

3 Graph pattern-matching

Pattern-matching on graphs is the problem of finding a homomorphic or isomorphic image of a given graph, called the *pattern*, in another graph, called the *target*. Since the image of the pattern is then a subgraph of the target, the problem is also known as the *subgraph isomorphism* problem, although in some application areas, for instance in graph transformation, the image of the pattern need not necessarily be isomorphic and a homomorphic image suffices.

Since subgraph isomorphism is known to be NP-complete [6], most efforts have been directed either at finding efficient pattern-matching algorithms for restricted classes of graphs, at finding useful heuristics for graph pattern-matching, or at reducing the pattern-matching problem along partial subgraphs of the pattern, as already mentioned in Section 1. A simpler reduction of the pattern-matching problem, however, can be performed when the pattern graph is not connected, as stated by the following theorem.

Theorem 7. *Pattern-matching from a non-connected pattern to a target is equivalent to pattern-matching from every connected component of the pattern to connected components of the target.*

Proof. Let $G = (V, B)$ be a non-connected pattern graph and let $G' = (V', B')$ be a target graph. It can be taken, without loss of generality,

$$B = \left(\begin{array}{c|c} A & 0 \\ \hline 0 & C \end{array}\right) \quad \text{and} \quad B' = \left(\begin{array}{c|c} X & Y \\ \hline Y^T & Z \end{array}\right).$$

Then it suffices to consider graph homomorphisms of the form

$$\Phi = \left(\begin{array}{c|c} \Phi_1 & 0 \\ \hline 0 & \Phi_4 \end{array}\right),$$

and the conditions for graph homomorphism become:

$$\left(\begin{array}{c|c} \Phi_1^T & 0 \\ \hline 0 & \Phi_4^T \end{array}\right)\left(\begin{array}{c|c} \Phi_1 & 0 \\ \hline 0 & \Phi_4 \end{array}\right) = \left(\begin{array}{c|c} \Phi_1^T\Phi_1 & 0 \\ \hline 0 & \Phi_4^T\Phi_4 \end{array}\right) \subseteq \left(\begin{array}{c|c} I & 0 \\ \hline 0 & I \end{array}\right)$$

$$\left(\begin{array}{c|c} I & 0 \\ \hline 0 & I \end{array}\right) \subseteq \left(\begin{array}{c|c} \Phi_1 & 0 \\ \hline 0 & \Phi_4 \end{array}\right)\left(\begin{array}{c|c} \Phi_1^T & 0 \\ \hline 0 & \Phi_4^T \end{array}\right) = \left(\begin{array}{c|c} \Phi_1\Phi_1^T & 0 \\ \hline 0 & \Phi_4\Phi_4^T \end{array}\right)$$

$$\left(\begin{array}{c|c} A & 0 \\ \hline 0 & C \end{array}\right) \subseteq \left(\begin{array}{c|c} \Phi_1 & 0 \\ \hline 0 & \Phi_4 \end{array}\right)\left(\begin{array}{c|c} X & Y \\ \hline Y^T & Z \end{array}\right)\left(\begin{array}{c|c} \Phi_1^T & 0 \\ \hline 0 & \Phi_4^T \end{array}\right) = \left(\begin{array}{c|c} \Phi_1 X\Phi_1^T & \Phi_1 Y\Phi_4^T \\ \hline \Phi_4 Y^T\Phi_1^T & \Phi_4 Z\Phi_4^T \end{array}\right)$$

That is, there exists a graph homomorphism $\Phi : G \to G'$ if, and only if, there exist graph homomorphisms $\Phi_1 : A \to X$ and $\Phi_4 : C \to Z$.

Since a graph homomorphism from a subgraph of the pattern to a subgraph of the target extends to the whole target, as stated by Proposition 6, we also have the following lemma.

Lemma 8. *Pattern-matching from a non-connected pattern to a target is equivalent to pattern-matching from every connected component of the pattern to the whole target.*

Proof. Let $G = (V, B)$ be a non-connected pattern graph and let $G' = (V', B')$ be a target graph. It can be taken, without loss of generality, G to be the disjoint union of two (not necessarily connected) components $G_1 = (V_1, B_1)$ and $G_2 = (V_2, B_2)$, such that

$$B = \left(\begin{array}{c|c} B_1 & 0 \\ \hline 0 & B_2 \end{array}\right)$$

It has to be shown that there is a graph homomorphism $\Phi : G \to G'$ if, and only if, there are graph homomorphisms $\Phi_1 : G_1 \to G'$ and $\Phi_2 : G_2 \to G'$.

($\Rightarrow$) It follows from graph homomorphisms being closed under composition. Let $\Phi_1' : G_1 \to G$ be the inclusion homomorphism of G_1 in G. Then $\Phi_1 = \Phi_1'\Phi$ is a graph homomorphism $\Phi_1 : G_1 \to G'$, by Proposition 3. In the same way, let $\Phi_2' : G_2 \to G$ be the inclusion homomorphism of G_2 in G. Then $\Phi_2 = \Phi_2'\Phi$ is a graph homomorphism $\Phi_2 : G_2 \to G'$, by Proposition 3.

($\Leftarrow$) Let $\Phi_1 : G_1 \to G'$ and $\Phi_2 : G_2 \to G'$ be graph homomorphisms. It has to be show that there is a graph homomorphism $\Phi : G \to G'$. Let

$$\Phi = \left(\begin{array}{c} \Phi_1 \\ \hline \Phi_2 \end{array}\right)$$

Then the first condition for graph homomorphism is satisfied,

$$\Phi^T\Phi = \left(\begin{array}{c} \Phi_1 \\ \hline \Phi_2 \end{array}\right)^T \left(\begin{array}{c} \Phi_1 \\ \hline \Phi_2 \end{array}\right) = \left(\begin{array}{c|c} \Phi_1^T & \Phi_2^T \end{array}\right)\left(\begin{array}{c} \Phi_1 \\ \hline \Phi_2 \end{array}\right) = \Phi_1^T\Phi_1 + \Phi_2^T\Phi_2 \subseteq I$$

as well as the second condition,

$$I \subseteq \Phi_1\Phi_1^T + \Phi_2\Phi_2^T = \left(\begin{array}{c} \Phi_1 \\ \hline \Phi_2 \end{array}\right)\left(\begin{array}{c|c} \Phi_1^T & \Phi_2^T \end{array}\right) = \left(\begin{array}{c} \Phi_1 \\ \hline \Phi_2 \end{array}\right)\left(\begin{array}{c} \Phi_1 \\ \hline \Phi_2 \end{array}\right)^T = \Phi\Phi^T$$

and the third condition

$$\left(\begin{array}{c|c} B_1 & 0 \\ \hline 0 & B_2 \end{array}\right) \subseteq \left(\begin{array}{c|c} \Phi_1 B'\Phi_1^T & \Phi_1 B'\Phi_2^T \\ \hline \Phi_2 B'\Phi_1^T & \Phi_2 B'\Phi_2^T \end{array}\right) = \left(\begin{array}{c} \Phi_1 \\ \hline \Phi_2 \end{array}\right) B' \left(\begin{array}{c} \Phi_1^T \\ \hline \Phi_2^T \end{array}\right) = \Phi B'\Phi^T$$

is also satisfied. Therefore, $\Phi : G \to G'$ is a graph homomorphism.

Notice that such an equivalence does not hold anymore if the pattern is a connected graph.

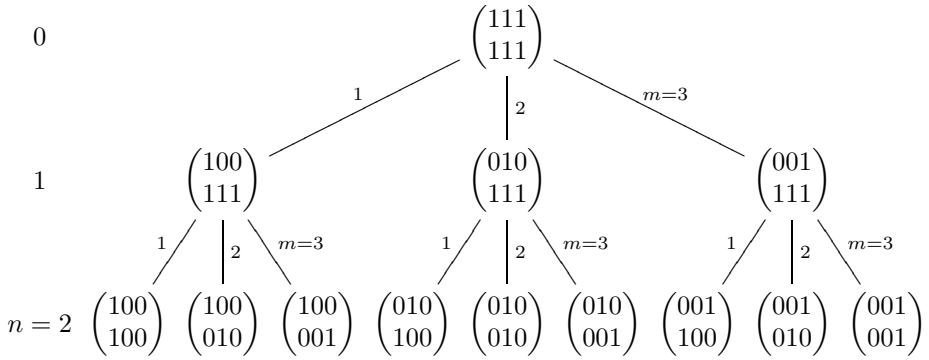
4 The graph pattern-matching algorithm

Results from the previous section allow to decompose the graph pattern-matching problem along connected components of the pattern graph.

In this section, a basic tree-search algorithm is presented which can be applied to any pattern graph. The algorithm is then improved by a pruning operator that allow a significant prunning of the search tree. The improved algorithm is finally integrated into a pattern-matching algorithm along connected components.

4.1 Basic search algorithm

Graph pattern-matching can be understood as tree search; cf. [16]. Let $G = (V, B)$ be a pattern graph with $|V| = n$, and let $G' = (V', B')$ be a target graph with $|V'| = m$. A graph homomorphism $\Phi : G \to G'$ is given by a boolean $n \times m$ matrix satisfying the conditions for graph homomorphism given in Definition 2; such a matrix will have exactly one *true* (one) in each row, although it may have more than one *true* in each column. Candidates for graph homomorphism are then the leaves of the following n-depth m-ary tree (drawn for $n = 2$ and $m = 3$):



In particular, all leaves will be graph homomorphisms when the pattern graph is discrete, and no leaf will be a graph homomorphism when matching a non-discrete pattern to a discrete target.

Any traversal of such a tree could be used to enumerate all leaves and test them for homomorphism, although the problem of finding one homomorphism is best solved by performing a depth-first search.

An iterative search algorithm is sketched in Fig. 1. Starting off with an $n \times m$ matrix whose elements are all *true* (one), the algorithm returns the first leaf of the tree (in a depth-first search) that satisfies the conditions for graph

```
function homomorphism(P, T)
    return search(setup(P, T), P, T)

function setup(P, T)
    for i = 1 to n do
        for j = 1 to m do
            H[i, j] ← true
        end for
    end for
    return H

function search(H, P, T)
    if H is the one matrix then
        i ← 0                              { build first leaf }
    else
        i ← n                              { H is already a leaf; build next leaf }
    end if
    repeat
        if i < n then
            i ← i + 1
            for j = 1 to m do
                H[i, j] ← j = 1            { map node i to node 1 }
            end for
        else
            while i ≥ 1 and H[i, m] do
                for j = 1 to m do
                    H[i, j] ← true         { restore i-th row }
                end for
                i ← i − 1
            end while
            if i = 0 then
                H ← null matrix
            else
                j ← 1
                while not H[i, j] do
                    j ← j + 1
                end while
                H[i, j] ← false            { unmap node i to node j }
                H[i, j + 1] ← true         { map node i to node j + 1 }
            end if
        end if
    until i = n and test(H, P, T) or i = 0
    return H
```

Fig. 1. Basic search algorithm

homomorphism, if any, and returns the null matrix if no graph homomorphism is found.

The *test* function, sketched in Fig. 2, implements the conditions for graph homomorphism stated in Definition 2. I denotes the identity boolean matrix. This function can also be specialized to the case of subgraph isomorphism.

function test(H, P, T)
  if $H^T \cdot H$ *is included in* I **then**
    if I *is included in* $H \cdot H^T$ **then**
      return P *is included in* $H \cdot T \cdot H^T$
    end
  end
  return *false*

Fig. 2. Test for subgraph homomorphism

The search for a homomorphism can also be performed with a previous homomorphism as start matrix, returning then either the next leaf of the tree that satisfies the conditions for graph homomorphism, or the null matrix if no further graph homomorphism is found. In Fig. 3 a procedure is sketched for enumerating all homomorphisms.

procedure enumerate(P, T)
  $H \leftarrow$ setup(P, T)
  repeat
    $H \leftarrow$ search(H, P, T)
  until H *is the null matrix*

Fig. 3. Enumeration algorithm

4.2 Pruning the search tree

The basic tree-search algorithm can be improved in various ways. For instance, branches could be pruned as soon as a mapping of part of the pattern nodes to the target violates the requirements of graph homomorphisms; the start $n \times m$ matrix could be rearranged in such a way that branches could be pruned sooner (in a depth-first search); some entries in the start matrix could be set to *false* (zero) if the corresponding mapping of a pattern node to a target node would not lead to a graph homomorphism; etc.

An improved algorithm is sketched in Fig. 4 which takes into account the fact that the image of a node which is adjacent to some other node has to be adjacent to the image of the other node; cf. [16]. The *ismapable* function, sketched in Fig. 5, is called by the *mapable* function to find the next node j to which a given node i can be mapped. This function can be specialized to the case of subgraph

```
function search(H, P, T)
    if H is the one matrix then
        i ← 0                                   { build first leaf }
        j ← 0
    else
        i ← n                                   { H is already a leaf; build next leaf }
        j ← 1
        while not H[i, j] do
            j ← j + 1
        end while
    end
    repeat
        if i ≠ n then
            i ← i + 1
            j ← 1
        else
            j ← j + 1
        end if
        while not mapable(H, P, T, i, j) and i ≠ 0 do
            for k = 1 to m do
                H[i, k] ← true                  { restore i-th row }
            end for
            i ← i − 1
            if i ≠ 0 then
                j ← 1
                while not H[i, j] do
                    j ← j + 1
                end while
                j ← j + 1
            end if
        end while
        if i ≠ 0 then
            for k = 1 to m do
                H[i, k] ← j = k                 { map node i to node j }
            end for
        else
            H ← null matrix
        end if
    until i = 0 or (i = n and test(H, P, T))
    return H

function mapable(H, P, T, i, var j)
    while j ≤ m and not ismapable(H, P, T, i, j) do
        j ← j + 1
    end while
    return j ≠ m + 1
```

Fig. 4. Search algorithm improved by pruning the search tree

```
function ismapable(H, P, T, i, j)
    found ← false
    x ← 1
    while x ≤ i and not found do
        y ← image(H, x)
        if P[i, x] and not T[j, y] or P[x, i] and not T[y, j] then
            found ← true            { i cannot be mapped to j }
        end if                      { because x is not adjacent to y }
        x ← x + 1
    end while
    return not found

function image(H, x)
    y ← 1
    while not H[x, y] do
        y ← y + 1
    end while
    return y
```

Fig. 5. Pruning operator for subgraph homomorphism

```
function ismapable(H, P, T, i, j)
    found ← false
    x ← 1
    while x ≤ i and not found do
        if H[x, j] then
            found ← true            { i cannot be mapped to j }
        else                        { because x has already been mapped to j }
            y ← image(H, x)
            if P[i, x] and not T[j, y] or P[x, i] and not T[y, j] then
                found ← true        { i cannot be mapped to j }
            end if                  { because x is not adjacent to y }
        end if
        x ← x + 1
    end while
    return not found
```

Fig. 6. Pruning operator for subgraph isomorphism

isomorphism, by also testing that no other node x has already been mapped to node j, as sketched in Fig. 6.

Further considerations could also be taken into account in the pattern-matching algorithm, some of which belong to specific application domains.

1. In the case of subgraph isomorphism, the net degree of the image of a node cannot be less than the net degree of the node; cf. [16].

2. The distance between the images of two nodes cannot be greater than the distance between the nodes. In particular, in the case of subgraph isomorphism, these distances have to coincide.

3. In the case of labeled graphs, not only the structure of the pattern but also the labeling of nodes and arcs have to match the target. Then the image of a labeled node or arc has to carry the same label as the node or arc.

4. In the case of algebraic graph transformation, pattern nodes and arcs can be partitioned into the set candidates to be removed and the set of candidates to be preserved during the transformation of the target graph. Then the image of a node or arc which is a candidate to be removed cannot be also the image of a node or arc which is a candidate to be preserved, as stated by the *identification condition* in double-pushout graph transformation; cf. [4].

5. Also in algebraic graph transformation, consider nodes and arcs which are candidates to be removed first. Since a node to be removed can only be adjacent to a node of any kind through an arc to be removed, while a node to be preserved can be adjacent to a node to be preserved through an arc of any kind, or to a node to be removed through an arc to be removed, it may improve the effect of other heuristics. This heuristic follows from the *dangling condition* in double-pushout graph transformation; cf. [4].

4.3 Improved search algorithm

The algorithm for decomposing the pattern-matching problem along connected components of the pattern graph is sketched in Fig. 7. The *decompose* function returns the first graph homomorphism found from the pattern graph to the target graph, if any, and returns the null matrix if no graph homomorphism is found.

The *partition* function, sketched in Fig. 8, computes the transitive closure of the pattern graph, following [17], in order to find all connected components, and returns a vector assigning no each node the number of the connected component to which it belongs. Connected components are extracted out of the pattern graph by the *submatrix* function, sketched in Fig. 9.

Composition of the images of the connected components of the pattern in the target is done in an incremental fashion, where variable *hash* is used to compute the position of a connected component node in the whole pattern graph.

```
function decompose(H, P, T)
    R ← partition(P)
    conn ← 0
    for i = 1 to n do
        if R[i] > conn then
            conn ← R[i]                    { number of connected components }
        end if
    end for
    failed ← false
    i ← 1
    while i ≤ conn and not failed do       { i-th connected component }
        N ← submatrix(P, R, i)
        S ← search(one matrix, N, T)
        if S is the null matrix then
            failed ← true                  { subsearch has failed }
        else                               { the whole search fails }
            hash ← 0
            for j = 1 to number of rows in S do
                repeat
                    hash ← hash + 1
                until R[hash] = i
                for k = 1 to n do
                    H[hash, k] ← S[j, k]   { assign hash-th row in S }
                end for                    { to j-th row in H }
            end for
        end if
        i ← i + 1
    end while
    if failed then
        return null matrix
    else
        return H
    end if
```

Fig. 7. Search algorithm along connected components of the pattern graph

```
function partition(P)
    for i = 1 to n do
        for j = 1 to n do
            if P[j, i] then
                for k = 1 to n do
                    P[j, k] ← P[j, k] or P[i, k]        { transitive closure of P }
                end for
            end if
        end for
    end for
    for i = 1 to n do
        R[i] ← 0
    end for
    k ← 0                                               { number of next connected }
    for i = 1 to n do                                   { component to be assigned }
        found ← false
        j ← 1
        while not found and j ≤ n do
            if P[i, j] then
                found ← true                            { there is some node j }
            end if                                      { adjacent to node i }
            j ← j + 1
        end while
        if found then                                   { i-th node is not isolated }
            found ← false
            j ← 1
            while not found and j ≤ n do
                if P[i, j] and R[j] = 0 then
                    found ← true                        { there is some node j }
                end if                                  { adjacent to node i }
                j ← j + 1                               { that has to be assigned to }
            end while                                   { a new connected component }
            if found then
                k ← k + 1
            end if
            for j = 1 to n do
                if P[i, j] and R[j] = 0 then
                    R[j] ← k                            { assign j-th node }
                end if                                  { to k-th connected component }
            end for
        else                                            { i-th node is isolated }
            k ← k + 1
            R[i] ← k                                    { assign i-th node }
        end if                                          { to k-th connected component }
    end for
    return R
```

Fig. 8. Partition of a graph into connected components

```
function submatrix(P, R, i)
    for j = 1 to n do
        G[j] ← R[j] = i
    end for
    N ← null matrix
    for i = 1 to n do
        if G[i] then
            for j = 1 to n do
                if P[i, j] then
                    N[subindex(G, i), subindex(G, j)] ← true
                end if
            end for
        end if
    end for
    return N

function subindex(G, i)
    j ← 0
    for k = 1 to i do
        if G[k] then
            j ← j + 1
        end if
    end for
    return j
```

Fig. 9. Extraction of a connected component

5 Experimental results

The algorithm has been implemented in Oberon-2, and a series of experiments has been carried out on random pattern and target graphs.

Random graphs have been generated using the multiplicative linear congruential random number generator algorithm [11]. Table 1 shows the number of arcs, the number of connected components and the average connected component order versus arc probability for random directed graphs with 100 nodes. For each variable the mean and the standard deviation, which have been estimated over a sample of 50 observations, are given.

Table 1. Number of arcs, number of connected components and average connected component order versus arc probability for random directed graphs with 100 nodes

arc	no. arcs		no. components		avg. comp. order	
probability	mean	std. dev.	mean	std. dev.	mean	std. dev.
0.05	255	16.20	28.50	3.66	2.94	0.43
0.10	502	20.20	14.60	3.15	6.58	1.88
0.15	757	24.70	9.92	2.42	10.10	2.55
0.20	1010	28.40	7.14	1.95	14.70	4.84
0.25	1260	31.20	5.56	1.70	19.70	8.35
0.30	1510	31.70	4.82	1.42	22.50	8.09
0.35	1770	32.40	4.08	1.26	28.10	16.20
0.40	2020	33.70	3.42	1.21	34.80	19.50
0.45	2270	37.60	2.90	1.16	41.50	21.90
0.50	2530	41.80	2.62	1.12	47.60	26.10

Matching experiments have been carried out on random pattern graphs of order n and random target graphs of order m, for selected values m and n such that $n \geq m$ and for a fixed arc probability value, same for pattern and target. For each generated pattern and target random graphs, the boolean matrix of the pattern has been added to the boolean matrix of the target, a procedure already followed by [16], and null pattern matrices have been discarded, since otherwise all leaves of the search tree would be graph homomorphisms and all the algorithms would find the same leaf first, namely the first leaf in a depth-first search.

Tables 2 and 3 show statistics for the number of leaves visited and the time (in seconds) taken to find one homomorphism from a random pattern graph to a random target graph, both with edge probability 0.25, using the basic search algorithm (*search* column) and using the algorithm improved by pruning the search tree (*prune* column). Table 3 also shows time statistics when decomposing the problem along connected components of the pattern graph and pruning the search tree for each connected component (*decompose* column). For each variable the mean and the standard deviation, which have been estimated over a sample of 50 observations, are given. Time statistics correspond to a Sun SPARCstation 5 running SPARC-Oberon (TM) V4 Release 2.9.1 for Solaris 2.

Table 2. Number of leaves visited while searching for a homomorphism

order of P	order of T	search		prune	
		mean	s. d.	mean	s. d.
6	6	397	618	67	281
6	8	1 400	1 893	57	195
6	10	3 713	4 756	184	895
6	12	7 196	10 042	99	462

Table 3. Time in seconds taken to find one homomorphism

order of P	order of T	search		prune		decompose	
		mean	s. d.	mean	s. d.	mean	s. d.
6	6	295	459	51	213	1	1
6	8	1 516	2 052	62	211	2	5
6	10	5 527	7 100	278	1 326	4	7
6	12	14 036	19 647	204	957	12	28

6 Conclusion

Graph pattern-matching, also known as the *subgraph homomorphism* problem, is dealt with in this paper. An algorithm is presented that is based on reducing the problem of finding a homomorphic image of a pattern graph in a target graph, to that of finding homomorphic images of every connected component of the pattern in the target. For each connected component, the algorithm performs a combinatorial search bounded by a pruning operator.

Experimental results have shown that search for a homomorphism behaves much better after decomposition along connected components for reasonably dense pattern graphs, for instance for random directed pattern graphs with arc probability lower than 0.4. For arc probability 0.25, which corresponds to the subgraph isomorphisms experiments reported in [16], searching for a homomorphism from a pattern graph of order 100 is reduced to about five searches for a homomorphism from a subgraph of the pattern of order about 20.

The algorithm has been also specialized to find isomorphic images only. In this case, however, the pattern-matching problem cannot be decomposed along connected components of the pattern graph, although it can be decomposed along connected components of the target graph; cf. [16].

Acknowledgement

We thank F. Drewes and H.-J. Kreowski for detailed comments on preliminary versions of this article and for substantial discussions about the subject. We also acknowledge with thanks the anonymous referees, whose suggestions have lead to a substantial improvement of the paper.

References

1. H. Bunke, T. Glauser, and T.-H. Tran. An efficient implementation of graph grammars based on the RETE matching algorithm. In *Proc. 4th Int. Workshop on Graph-Grammars and their Application to Computer Science and Biology*, volume 532 of *Lecture Notes in Computer Science*, pages 174–189. Springer-Verlag, 1991.

2. P. Burmeister, F. Rosselló, and G. Valiente. Double-pushout hypergraph rewriting through free completions. Technical Report LSI-96-56-R, Technical University of Catalonia, 1996.

3. H. Dörr. *Efficient graph rewriting and its implementation*, volume 922 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995.

4. H. Ehrig. Introduction to the algebraic theory of graph grammars. In *Proc. 1st Int. Workshop on Graph-Grammars and their Application to Computer Science and Biology*, volume 73 of *Lecture Notes in Computer Science*, pages 1–69. Springer-Verlag, 1979.

5. J. Fu. Pattern matching in directed graphs. In *Proc. 6th Annual Symposium on Combinatorial Pattern Matching*, volume 937 of *Lecture Notes in Computer Science*, pages 64–77. Springer-Verlag, 1995.

6. M. R. Garey and D. S. Johnson. *Computers and intractability: A guide to NP-completeness*. Freeman, 1979.

7. A. Gupta and N. Nishimura. Characterizing the complexity of subgraph isomorphism for graphs of bounded path-width. In *Proc. 15th Annual Symposium on Theoretical Aspects of Computer Science*, volume 1046 of *Lecture Notes in Computer Science*, pages 453–464. Springer-Verlag, 1997.

8. A. Lingas. Subgraph isomorphism for biconnected outerplanar graphs in cubic time. *Theoretical Computer Science*, 63:295–302, 1989.

9. A. Lingas and M. M. Syslo. A polynomial-time algorithm for subgraph isomorphism of two-connected series-parallel graphs. In *Proc. 15th Int. Colloquium on Automata, Languages, and Programming*, Lecture Notes in Computer Science, pages 394–409. Springer-Verlag, 1988.

10. D. Matula. Subtree isomorphism in $\mathcal{O}(n^{5/2})$. *Annals of Discrete Mathematics*, 2:91–106, 1978.

11. S. K. Park and K. W. Miller. Random number generations, good ones are hard to find. *Commun. ACM*, 31:1192–1201, 1988.

12. S. W. Reyner. An analysis of a good algorithm for the subtree problem. *SIAM Journal of Computing*, 6:730–732, 1977.

13. F. Rosselló and G. Valiente. Single-pushout hypergraph rewriting through free completions. Technical Report LSI-97-12-R, Technical University of Catalonia, 1997.

14. G. Schmidt and T. Ströhlein. *Relationen und Graphen*. Springer-Verlag, 1989.

15. M. M. Syslo. The subgraph isomorphism problem for outerplanar graphs. *Theoretical Computer Science*, 17:91–97, 1982.

16. J. R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, 23(1):31–42, 1976.

17. S. Warshall. A theorem on boolean matrices. *J. ACM*, 9:11–12, 1962.

18. A. Zündorf. Graph pattern-matching in PROGRES. In *Proc. 5th Int. Workshop on Graph Grammars and their Application to Computer Science*, volume 1073 of *Lecture Notes in Computer Science*, pages 454–468. Springer-Verlag, 1996.

This article was processed using the LaTeX 2_ε macro package with CUP_CS class