A Relational View of Subgraph Isomorphism*

J. Cortadella and G. Valiente

Department of Software, Technical University of Catalonia, Barcelona, Spain

Abstract. This paper presents a novel approach to the problem of finding *all* subgraph isomorphisms of a (pattern) graph into another (target) graph. A relational formulation of the problem, combined with a representation of relations and graphs by Boolean functions, allows to handle the combinatorial explosion in the case of small pattern graphs and large target graphs by using *Binary Decision Diagrams* (BDDs), which are capable to represent large relations and graphs in small data structures. Examples are presented that show how all subgraph isomorphisms (120 303) of a small graph into a large graph can be efficiently computed and represented with a small BDD (57 980 nodes).

1 Introduction

Subgraph isomorphism is one of the most important techniques in pattern recognition. It has been studied thoroughly for both theoretical and practical interests.

The subgraph isomorphism problem can be formulated as follows: given two graphs G_1 and G_2, find out if G_2 contains a subgraph that is isomorphic to G_1, or find all such isomorphic subgraphs. The problem is known to be NP-complete [8]. The best known algorithm for subgraph isomorphism is [21], which is based on exhaustive search with backtracking and solves both formulations of the problem. As a matter of fact, most of the research on subgraph isomorphism algorithms has been based either on heuristic search techniques [1, 7, 9, 21] or on constraint satisfaction techniques [14].

In this paper, a novel approach to the problem of finding all subgraph isomorphisms is presented. A relational formulation of the problem, combined with a representation of relations and graphs by Boolean functions, allows to handle the combinatorial explosion in the case of small pattern graphs and large target graphs by using Binary Decision Diagrams (BDDs), which are capable to represent large relations and graphs in small data structures.

The rest of the paper is organized as follows. Sections 2 and 3 present background material on relations and functions and on BDDs, respectively. A bridge between graphs and BDDs is established in Section 4. Section 5 presents the relational view of the problem of finding all subgraph isomorphisms. Experimental results are summarized in Section 6. Finally, some conclusions and lines of future research are drawn in Section 7.

2 Relations and Functions

Given two sets A and B, a *binary relation* $\mathcal{R}$ between A and B is a subset of $A \times B$. Given a set of sets $A_1, \ldots, A_n$, an n-ary relation $\mathcal{R}$ over $A_1, \ldots, A_n$ is a subset of $A_1 \times \cdots \times A_n$. For a binary relation we say that $x\mathcal{R}y$ if and only if $(x, y) \in \mathcal{R}$.

A binary relation $\mathcal{R}$ can be represented by a $|A| \times |B|$ Boolean matrix $M_\mathcal{R}$ in which $M_\mathcal{R}[x, y] = x\mathcal{R}y$. A binary relation $\mathcal{R}$ between a set A and itself is a subset of A^2 and can also be represented by a directed graph $G = (V, E)$, where $V = A$ and $E = \mathcal{R}$.

A *function* f from A to B, $f : A \to B$, is a relation that associates exactly one element of B to each element of A. A function f is injective if $x \neq y$ implies $f(x) \neq f(y)$. A function f is surjective if for every $y \in B$ there exists an element $x \in A$ such that $f(x) = y$. A function is bijective if it is injective and surjective.

* Partially supported by the EC TMR Network GETGRATS (General Theory of Graph Transformation Systems) and by the ESPRIT Basic Research Working Group APPLIGRAPH (Applications of Graph Transformation) through the University of Bremen, by the Spanish DGES project PB96-0191-C02-02 and CICYT projects TIC98-0410-C02-01 and TIC98-0949-C02-01 HEMOSS, and by the Joint Commission USA-Spain for Scientifical and Technological Cooperation project 98191.

We will often talk about Boolean functions over the set $\mathbb{B} = \{0, 1\}$. An n-variable *Boolean function* is a function $f : \mathbb{B}^n \to \mathbb{B}$. Typically we will represent Boolean functions with Boolean formulae in which the operators $+$ and $\cdot$ will denote the disjunction and conjunction respectively, $\cdot$ having higher precedence than $+$. For simplicity, the operator $\cdot$ will be often omitted. For example, the 3-variable Boolean function

$$\begin{gathered} f(0,0,0) = f(0,0,1) = f(0,1,1) = f(1,1,1) = 1 \\ f(0,1,0) = f(1,0,0) = f(1,0,1) = f(1,1,0) = 0 \end{gathered}$$

can be represented by the Boolean formula

$$f(x_1, x_2, x_3) = \overline{x}_1\overline{x}_2 + x_2x_3$$

Each of the elementary product terms of an n-variable Boolean formula

$$\overline{x}_1 \ldots \overline{x}_{n-1}\overline{x}_n, \qquad \overline{x}_1 \ldots \overline{x}_{n-1}x_n, \qquad \ldots, \qquad x_1 \ldots x_{n-1}x_n$$

are called *minterms*. A Boolean formula represented as a sum of minterms is called a *minterm canonical form*. For the previous example, the minterm canonical form is

$$f(x_1, x_2, x_3) = \overline{x}_1\overline{x}_2\overline{x}_3 + \overline{x}_1\overline{x}_2x_3 + \overline{x}_1x_2x_3 + x_1x_2x_3$$

Given a set A, an *encoding function* of A is an injective function $\sigma : A \to \mathbb{B}^n$. A necessary condition for σ to be injective is that $|A| \leqslant 2^n$. Given an encoding function σ of a set A, the characteristic function of A is an n-variable function χ_A defined as follows:

$$\chi_A(x) = 1 \;\Leftrightarrow\; \exists a \in A : \sigma(a) = x$$

where x is a vector of n Boolean variables, $x = (x_1, \ldots, x_n)$.

Given a binary relation $\mathcal{R}$ between two sets A and B and two encoding functions $\sigma_A : A \to \mathbb{B}^n$ and $\sigma_B : B \to \mathbb{B}^m$, the characteristic function of $\mathcal{R}$ is an $(n+m)$-variable Boolean function $\chi_{\mathcal{R}} : \mathbb{B}^{n+m} \to \mathbb{B}$ defined as follows:

$$\chi_{\mathcal{R}}(x, y) = 1 \;\Leftrightarrow\; \exists (a, b) \in \mathcal{R} : \sigma(a) = x \;\wedge\; \sigma(b) = y$$

where $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_m)$. Characteristic functions can be trivially extended to n-ary relations. Henceforth, and for the sake of simplicity, we will use the symbols A and $\mathcal{R}$ to represent the characteristic functions χ_A and $\chi_{\mathcal{R}}$ respectively under some implicit encoding functions.

Now, Stone's representation theorem [20] establishes the basis of the approach presented in this paper.

Theorem 1. *Every finite Boolean algebra is isomorphic to the Boolean algebra of subsets of some finite set.*

Stone's representation theorem states that reasoning in terms of concepts such as *union*, *intersection*, *empty set*, etc. in a finite set of elements, in particular in a finite set of relations, is isomorphic to performing Boolean operations $(+, \cdot)$ with Boolean functions. This means that all operations on relations can be applied directly to their characteristic functions. For example, given two binary relations $\mathcal{R}$ and $\mathcal{S}$,

$$\chi_{\mathcal{R} \cup \mathcal{S}} = \chi_{\mathcal{R}} + \chi_{\mathcal{S}}, \qquad \chi_{\mathcal{R} \cap \mathcal{S}} = \chi_{\mathcal{R}} \cdot \chi_{\mathcal{S}}$$

When implemented with BDDs, characteristic functions provide, in general, compact and efficient representations.

3 Binary Decision Diagrams

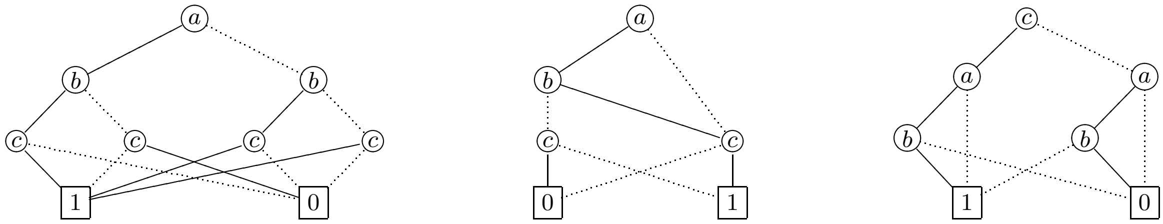
A Boolean function can be represented in many ways, such as *truth tables*, *Karnaugh maps* or *minterm canonical forms* [3]. Another form that can be much more compact is the *sum of products*, where the Boolean function is represented by means of an equation, that is,

$$f = bc + a\overline{b}\overline{c} + \overline{a}c \tag{1}$$

These techniques are inefficient for fairly large functions. However, all these forms can be canonical [3]. A form is canonical, if the representation of any function in that form is unique. Canonical forms are useful for verification techniques, because an equivalence test between functions is easily computable.

Recently, Binary Decision Diagrams (BDDs) have emerged as an efficient canonical form to manipulate large Boolean functions. The introduction of BDDs is relatively old [12], but only after the recent work of Bryant [4] they transformed into a useful tool. For a good review on BDDs we refer to [5].

We will present BDDs by means of an example. Given (1), its BDD is shown in Fig. 1(a). A BDD is a directed acyclic graph with one root and two leaf nodes (0 and 1). Each node has two outgoing arcs, labeled 1 and 0 and shown in the drawings as continuous and dotted lines, respectively, oriented from top to bottom. To evaluate f for the assignment of variables $a = 1$, $b = 0$, and $c = 1$, we only have to follow the corresponding directed arcs from the root node. The first node we encounter is labeled with variable a, whose value is 1. Given this assignment, the 1-arc must be taken. Next, a node labeled with variable b is found. Since $b = 0$, the 0-arc must be taken now. Finally, the 1-arc for variable c reaches the 0 leaf node, which is the value of f for that assignment.



(a) Ordered BDD

(b) Ordered, reduced BDD

(c) Ordered BDD with a different variable ordering

Fig. 1. Examples of Binary Decision Diagrams (BDDs) for the Boolean function $f = bc + a\bar{b}\bar{c} + \bar{a}c$.

The representation of a function by means of a BDD is not unique. Figures 1(a), 1(b) and 1(c) depict different BDDs representing (1). The BDD in 1(b) can be obtained from 1(a) by successively applying reduction rules that eliminate *isomorphic subgraphs* from the representation [5]. The BDD in 1(c) has a different variable ordering.

All BDDs shown in Fig. 1 are *ordered* BDDs. In an ordered BDD, all variables appear in the same order along all paths from the root to the leaves. If a BDD is *ordered* and *reduced* (no further reductions can be applied) then we have a Reduced Ordered BDD (ROBDD). Given a total ordering of variables, an ROBDD is a canonical form [5]. Figures 1(b) and 1(c) are ROBDDs with variable ordering $a < b < c$ and $c < a < b$ respectively. The shape and size of an ROBDD depend on the ordering of its variables.

Some important properties of ROBDDs are:

- The size of the BDD can be exponential in the number of variables [13]; however, BDDs are a compact representation for many functions.
- Boolean binary operations can be calculated in polynomial time in the size of the BDDs.
- Given a Boolean function represented as a BDD, some interesting problems like satisfiability, tautology and complementation are solved in constant time.

Henceforth, we will implicitly assume that BDDs are reduced and ordered. Note that each BDD node represents at the same time a function whose root is the node itself. This property allows the implementation of BDD packages managing all BDDs using the same set of variables in only one multi-rooted graph [5].

Most Boolean operations performed with BDDs are based on the recursive paradigm supported by *Boole's expansion.* Given a function $f(x_1, \ldots, x_n)$, the functions

$$f_{x_i} = f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n) \quad \text{and} \quad f_{\overline{x}_i} = f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)$$

are called the *cofactor* of f with respect to x_i and $\overline{x}_i$ respectively. The definition of cofactor can also be extended to products of literals. If $c = x_1 c_1$, x_1 being a literal and c_1 another product of literals, then $f_c = (f_{x_1})_{c_1}$.

Theorem 2 (Boole's expansion). *If $f : \mathbb{B}^n \to \mathbb{B}$ is a Boolean function, then for any variable x_i:*

$$f(x_1, x_2, \ldots, x_n) = x_i f_{x_i} + \overline{x}_i f_{\overline{x}_i}$$

Let us see, first, how to calculate the BDD for (1) given the ordering $a < b < c$. We will use $(v,\ f_1,\ f_0)$ to denote a BDD with a root node labeled with variable v, and f_1 and f_0 as the functions after the 1-arc and 0-arc, respectively. Applying Boole's expansion theorem to expand f with variable a we have:

$$f \ = \ a\ f_a \ + \ \overline{a}\ f_{\overline{a}}$$

with $f_a = bc + \overline{b}\overline{c}$, and $f_{\overline{a}} = bc + c = c$. Expanding variable b in f_a and $f_{\overline{a}}$ yields to

$$f = a\ (b\ f_{ab} \ + \ \overline{b}\ f_{a\overline{b}}) \ + \ \overline{a}\ (b\ f_{\overline{a}b} \ + \ \overline{b}\ f_{\overline{a}\overline{b}})$$

with $f_{ab} = c$, $f_{a\overline{b}} = \overline{c}$, $f_{\overline{a}b} = c$, and $f_{\overline{a}\overline{b}} = c$. Thus the BDD for (1) is

$$f = (a, (b, c, \overline{c}), (b, c, c))$$

Note that the Boolean functions $f_{ab} = f_{\overline{a}b} = f_{\overline{a}\overline{b}} = c$ are isomorphic and must be represented with the same node if we want to preserve canonicity.

BDDs can be created by combining existing BDDs by means of Boolean operations like AND, OR, and XOR. This approach is implemented using the *if-then-else* operator (ite), defined as follows:

$$\mathrm{ite}(F, G, H) = F\,G + \overline{F}\,H$$

where F, G, H are Boolean functions represented by BDDs. The interesting property of the ITE operator is that it can directly implement all binary Boolean functions. For example:

$$\mathrm{AND}(F, G) = \mathrm{ite}(F, G, 0)\,, \qquad \mathrm{XOR}(F, G) = \mathrm{ite}(F, \overline{G}, G)$$

Let $Z = \mathrm{ite}(F, G, H)$, and let v be the *top* variable of F, G, H. Then the BDD for Z is recursively computed as follows [4]:

$$Z = (v, \mathrm{ite}(F_v, G_v, H_v), \mathrm{ite}(F_{\overline{v}}, G_{\overline{v}}, H_{\overline{v}}))$$

where the terminal cases are:

$$\mathrm{ite}(1, F, G) = \mathrm{ite}(0, G, F) = \mathrm{ite}(F, 1, 0) = \mathrm{ite}(\overline{F}, 0, 1) = \mathrm{ite}(G, F, F) = F$$

The code for the ITE algorithm is shown in Fig. 2. Note that the algorithm keeps the BDD reduced by checking if T equals E, and checking in a *unique-table* if the produced node already exists in the graph. In this way, all isomorphic subgraphs are always eliminated.

Unless there is a terminal case, every call to the procedure generates two other calls, so the total number of ITE calls would be exponential in the number of variables. To avoid this exponentiality, ITE uses a table of pre-computed operations (*computed table*). The *computed table* acts as a cache memory, in such a way that the most recently used results are stored in this table. The effect of this computed table is to cause ITE to be called at most once for each possible combination of the nodes in F, G, H. Therefore, the time complexity of the algorithm, under the assumptions of infinite memory and constant access time (hash) tables, is reduced to $\Theta(|F| \cdot |G| \cdot |H|)$.

An important consequence of representing all BDDs in the same graph is that checking the equivalence between two BDDs can be done in constant time (two BDDs representing the same function have the same root node). Counting the number of vertices represented by a BDD can be done in linear time in the size of the BDD.

```
ite (F,G,H) {
    if ( terminal case ) return result for terminal case;
    else if ( {F, G, H} is in computed-table )
        return pre-computed result;
    else {
        let v be the top variable of { F, G, H };
        T = ite (F_v,G_v,H_v);
        E = ite (F_v',G_v',H_v');
        if T equals E return T;
        R = find_or_add_unique_table (v,T,E);
        insert_computed_table ({ F, G, H }, R);
        return R;
}   }
```

Fig. 2. The *ITE* algorithm

4 Representation of Directed Graphs

The approach to subgraph isomorphism presented in this paper is based on unlabeled, directed, simple graphs, also called relational graphs [2, 17].

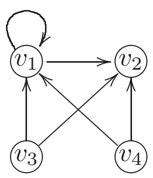
Definition 1. *A graph $G = (V, E)$ consists of a set V and a relation $E \subseteq V \times V$. The elements of V are called vertices, and E is called the (arc) relation associated to G. It is said that there is an arc from a node u to a node v if $(u, v) \in E$. A graph $G = (V, B)$ is finite if the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

We will deal only with finite graphs in the rest of the paper, without further mention. A graph $G = (V, E)$ can be represented by a Boolean adjacency matrix A, where an element $a_{i,j} = true$ (one) if $(a_i, a_j) \in E$, and it is *false* (zero) otherwise. We shall usually denote the boolean matrix A and the relation E associated to a graph $G = (V, E)$ by the same name E.

Example 1. The graph $G = (V, E)$ with $V = \{v_1, v_2, v_3, v_4\}$ and

$$E = \{(v_1, v_1), (v_1, v_2), (v_3, v_1), (v_3, v_2), (v_4, v_1), (v_4, v_2)\}$$

can be represented by the following Boolean adjacency matrix:



$$\begin{array}{c|cccc} & v_1 & v_2 & v_3 & v_4 \\ \hline v_1 & 1 & 1 & 0 & 0 \\ v_2 & 0 & 0 & 0 & 0 \\ v_3 & 1 & 1 & 0 & 0 \\ v_4 & 1 & 1 & 0 & 0 \end{array}$$

Given an encoding function σ_V for the set of vertices V of a graph $G = (V, E)$, the set of arcs E can also be represented by a characteristic function χ_{σ_V}.

Proposition 1. *The set of vertices of a graph $G = (V, E)$ with $|V| = n$ can be encoded using $\lceil \log_2 n \rceil$ variables.*

Proof. Let $k = \lceil \log_2 n \rceil$ and let $\sigma_V : V \to \mathbb{B}^k$ be a function mapping each vertex of V to a distinct k-bit Boolean vector. Then σ_V is an encoding of V, and it has $\lceil \log_2 n \rceil$ variables. □

Proposition 2. *Given an encoding $\sigma_V : V \to \mathbb{B}^k$ for the set of vertices of a graph $G = (V, E)$, where $|V| = n$ and $k = \lceil \log_2 n \rceil$, the set E of arcs of the graph can be represented by a characteristic function on $2\lceil \log_2 n \rceil$ variables.*

Proof. Let $\sigma_V : V \to \mathbb{B}^k$ be an encoding of the set of vertices V of a graph $G = (V, E)$, where $|V| = n$ and $k = \lceil \log_2 n \rceil$, and let $\boldsymbol{x}$ and $\boldsymbol{y}$ denote k-bit Boolean vectors, which will be used to represent the source and target vertices of each arc in E. Then

$$\chi_E(\boldsymbol{x}, \boldsymbol{y}) = \sum_{u \in V} \sum_{\substack{v \in V \\ uEv}} \left[\prod_{1 \leqslant i \leqslant k} x_i \overline{\oplus} \sigma_{V\,i}(u) \right] \cdot \left[\prod_{1 \leqslant i \leqslant k} y_i \overline{\oplus} \sigma_{V\,i}(v) \right]$$

represents the set of arcs E, and it has $2\lceil \log_2 n \rceil$ variables. $\square$

Example 2. Consider the graph of Example 1, where the encoding function $\sigma_V : V \to \mathbb{B}^2$ defined as

$$\sigma(v_1) = (0, 0), \qquad \sigma(v_2) = (0, 1), \qquad \sigma(v_3) = (1, 0), \qquad \sigma(v_4) = (1, 1)$$

has been chosen. The set of arcs can be represented by the characteristic function

$$\chi_E(\boldsymbol{x}, \boldsymbol{y}) = \overline{y}_1(x_1 + \overline{x}_2)\,,$$

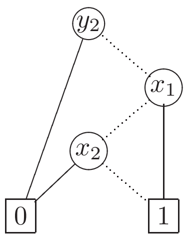
where the vectors $\boldsymbol{x} = (x_1, x_2)$ and $\boldsymbol{y} = (y_1, y_2)$ are used to represent the source and target vertices of each arc. Note that the minterm canonical form of the function $\chi_E(\boldsymbol{x}, \boldsymbol{y})$ has six minterms,

$$\overline{x}_1\overline{x}_2\overline{y}_1\overline{y}_2 + \overline{x}_1\overline{x}_2\overline{y}_1 y_2 + x_1\overline{x}_2\overline{y}_1\overline{y}_2 + x_1\overline{x}_2\overline{y}_1 y_2 + x_1 x_2\overline{y}_1\overline{y}_2 + x_1 x_2\overline{y}_1 y_2\,,$$

corresponding to the six arcs of the graph.

A graph can also be represented by a BDD.

Example 3. Given the variable ordering $y_2 < x_1 < x_2$, the characteristic function $\chi_E(\boldsymbol{x}, \boldsymbol{y}) = \overline{y}_2(x_1 + \overline{x}_2)$ for the graph of Examples 1 and 2 is represented by the following BDD:



5 Subgraph Isomorphism

Subgraph isomorphism is the problem of finding an isomorphic image of a given graph, called the *pattern*, in another graph, called the *target*. The name of this problem comes from the image of the pattern being then a subgraph of the target.

Graph homomorphisms are structure-preserving relations over the relations associated to the graphs [17, 22, 23]. In particular, a structure-preserving relation between the sets of vertices of two graphs is a subgraph isomorphism if it is injective. Let I denote the identity relation.

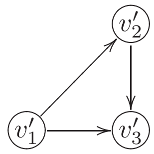
Definition 2. *A relation $\phi \subseteq V' \times V$ is an isomorphism of a graph $G' = (V', E')$ into a graph $G = (V, E)$ if*

$$\phi^T\,\phi \subseteq I, \quad \phi\,\phi^T = I, \quad E \subseteq \phi\,E'\,\phi^T\,;$$

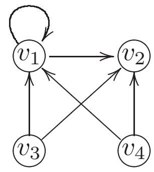
and it is also written $\phi : G' \to G$.

While the first two conditions assure that ϕ is an injective function from V' to V, the third condition guarantees that ϕ preserves the structure of G', that is, that it maps all arcs of G' to arcs of G. In other words, an injective function $\phi : V' \to V$ is a subgraph isomorphism of $G' = (V', E')$ into $G = (V, E)$ if it satisfies the following condition: $(u', v') \in E'$ implies $(\phi(u'), \phi(v')) \in E$ for all $u', v' \in V'$.

Example 4. Given the graphs



$G' = (V', E')$



$G = (V, E)$

there are two subgraph isomorphisms of G' into G:

$$\phi_1 = \{(v'_1, v_3), (v'_2, v_1), (v'_3, v_2)\}$$
$$\phi_2 = \{(v'_1, v_4), (v'_2, v_1), (v'_3, v_2)\}$$

The set of all subgraph isomorphisms of a graph $G' = (V', E')$ into a graph $G = (V, E)$ can be represented by an n-ary relation $\mathbf{I} \subseteq V^n$, where $|V'| = n$. The subgraph isomorphism relation $\mathbf{I}$ is defined as follows:

$$(v_1, \ldots, v_m) \in \mathbf{I} \;\Leftrightarrow\; \exists \text{ an isomorphism } \phi : G' \to G \text{ such that } \phi(v'_1) = v_1, \ldots, \phi(v'_n) = v_n$$

Given two graphs $G' = (V', E')$ and $G = (V, E)$ with $|V'| = n$, let $E_{i,j} \subseteq V^n$ denote the n-ary relation on V containing exactly those (pairwise disjoint) vertices of V which are joined by some arc in G, that is,

$$E_{i,j} = \{(x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n) \in V^n \mid (x_i, x_j) \in E, x_r \neq x_s \text{ for all } r \neq s\}$$

The subgraph isomorphism relation $\mathbf{I}$ can be computed as follows.

Theorem 3. $\mathbf{I} = \cap_{(v'_i, v'_j) \in E'} E_{i,j}$.

Proof. Let $(x_1 \ldots, x_n) \in \cap_{(v'_i, v'_j) \in E'} E_{i,j}$. and let $\phi : V' \to V$ be the function given by $\phi(v'_1) = x_1, \ldots, \phi(v'_n) = x_n$. Since every relation $E_{i,j}$ consists of pairwise disjoint vertices, ϕ is injective. Furthermore, since $(\phi(v'_1), \ldots, \phi(v'_i), \ldots, \phi(v'_j), \ldots, \phi(v'_n)) \in \cap_{(v'_i, v'_j) \in E'} E_{i,j}$, for every arc $(v'_i, v'_j) \in E'$ there is some $E_{i,j}$ such that $(x_1 \ldots, x_n) \in E_{i,j}$. Then $(x_i, x_j) \in E$.

Conversely, let the function $\phi : V' \to V$ given by $\phi(v'_1) = x_1, \ldots, \phi(v'_n) = x_n$ be a subgraph isomorphism of $G' = (V', E')$ into $G = (V, E)$. For every arc $(v'_i, v'_j) \in E'$ it must be $(\phi(v'_i), \phi(v'_j)) \in E$, and then $(\phi(v'_1), \ldots, \phi(v'_i), \ldots, \phi(v'_j), \ldots, \phi(v'_n)) \in E_{i,j}$. Therefore, $(x_1 \ldots, x_n) \in \cap_{(v'_i, v'_j) \in E'} E_{i,j}$. □

Table 1. Number of solutions, BDD size in number of nodes, and time in seconds taken to find all subgraph isomorphisms of a random graph with m vertices into a random graph with n vertices, both with 25% connectivity.

m	n	CPU time	isomorphisms	BDD size
3	25	0.01	575	253
4	25	0.08	234	779
5	25	0.06	672	1 045
6	25	0.63	6 334	10 216
7	25	4.22	4 102	15 013
8	25	48.21	125	1 015
9	25	137.30	520	5 744
3	50	0.01	6 816	1 173
4	50	0.92	4 544	7 860
5	50	0.62	52 210	27 048
6	50	35.42	552 076	400 954
3	100	0.03	40 964	3 119
4	100	18.88	103 416	105 743
5	100	7.11	1 539 840	278 399
3	150	0.07	193 584	7 576
4	150	99.71	512 375	440 979
5	150	50.74	17 319 542	1 560 970
3	200	0.11	480 744	12 427
4	200	57.32	1 611 593	1 229 120
5	200	131.57	74 097 604	4 445 444

Example 5. Given the graphs of Example 4, the subgraph isomorphism relation

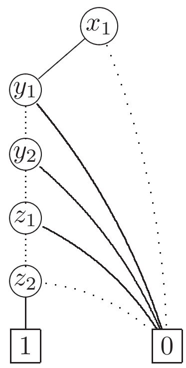
$$\mathbf{I} = \{(v_3, v_1, v_2), (v_4, v_1, v_2)\}$$

can be computed as

$$\begin{aligned} E_{1,2} \cap E_{1,3} \cap E_{2,3} =& \{(v_1, v_2, *), (v_3, v_1, *), (v_3, v_2, *), (v_4, v_1, *), (v_4, v_2, *)\} \cap \\ &\cap \{(v_1, *, v_2), (v_3, *, v_1), (v_3, *, v_2), (v_4, *, v_1), (v_4, *, v_2)\} \cap \\ &\cap \{(*, v_1, v_2), (*, v_3, v_1), (*, v_3, v_2), (*, v_4, v_1), (*, v_4, v_2)\} = \\ =& \{(v_3, v_1, v_2), (v_3, v_2, v_1), (v_4, v_1, v_2), (v_4, v_2, v_1)\} \cap \\ &\cap \{(v_1, *, v_2), (v_3, *, v_1), (v_3, *, v_2), (v_4, *, v_1), (v_4, *, v_2)\} = \\ =& \{(v_3, v_1, v_2), (v_4, v_1, v_2)\}\,, \end{aligned}$$

where an asterisc replaces all remaining variables. For instance, $(v_1, v_2, *)$ abbreviates both (v_1, v_2, v_3) and (v_1, v_2, v_4) together.

Under the encoding $\sigma_V : V \to \mathbb{B}^2$ of Example 2, and using 2-bit vectors $\boldsymbol{x} = x_1x_2$, $\boldsymbol{y} = y_1y_2$, $\boldsymbol{z} = z_1z_2$ to encode vertices v'_1, v'_2, v'_3, respectively, this corresponds to computing the BDD that represents $\chi_E(\boldsymbol{x}, \boldsymbol{y}) \cdot \chi_E(\boldsymbol{y}, \boldsymbol{z}) \cdot \chi_E(\boldsymbol{x}, \boldsymbol{z}) \cdot D(\boldsymbol{x}, \boldsymbol{y}) \cdot D(\boldsymbol{y}, \boldsymbol{z}) \cdot D(\boldsymbol{x}, \boldsymbol{z}) = \overline{y}_1(x_1 + \overline{x}_2)\overline{z}_1(y_1 + \overline{y}_2)\overline{z}_1(x_1 + \overline{x}_2) \cdot D(\boldsymbol{x}, \boldsymbol{y}) \cdot D(\boldsymbol{y}, \boldsymbol{z}) \cdot D(\boldsymbol{x}, \boldsymbol{z})$,



where $D(\boldsymbol{x}, \boldsymbol{y}) = (x_1 \oplus y_1) + (x_2 \oplus y_2) = \overline{x}_1 y_1 + x_1 \overline{y}_1 + \overline{x}_2 y_2 + x_2 \overline{y}_2$ indicates that $\boldsymbol{x} \neq \boldsymbol{y}$, that is, $x_1 \neq y_1$ or $x_2 \neq y_2$, and traversing all paths (only one, in this example) from the root node to the 1 leaf node, $x_1\,\overline{y_1}\,\overline{y_2}\,\overline{z_1}\,z_2$, which encode the desired solutions:

$$\begin{aligned} x_1\,x_2\,\overline{y_1}\,\overline{y_2}\,\overline{z_1}\,z_2 &= [(1,1)\,(0,0)\,(0,1)] = (v_4, v_1, v_2) \\ x_1\,\overline{x_2}\,\overline{y_1}\,\overline{y_2}\,\overline{z_1}\,z_2 &= [(1,0)\,(0,0)\,(0,1)] = (v_3, v_1, v_2) \end{aligned}$$

6 Experimental Results

The algorithm has been implemented in C using a public-domain BDD package [19]. A series of experiments has been carried out on random pattern and target graphs, on a Sun Ultra E250 Enterprise running Solaris 7.0 at 400 MHz and with 256 MB of main memory.

Random graphs have been generated using the multiplicative linear congruential random number generator algorithm [16] for different values of arc probability: 10%, 15%, 20%, and 25%. Matching experiments have been carried out on random pattern graphs of order m and random target graphs of order n, for selected values m and n such that $m \geqslant n$ and for a fixed arc probability value.

A total of 384 experiments have been carried out, for different values of pattern arc probability (10%, 15%, 20%, and 25%), target arc probability (10%, 15%, 20%, and 25%), pattern size (3, 4, 5, and 6 vertices), and target size (10, 25, 50, 100, 150, and 200 vertices). Table 1 shows statistics for the number of solutions, the size of the BDD containing them, and the time (in seconds) taken to find all subgraph isomorphisms of a random pattern graph into a random target graph, both with arc probability 25%.

In order to compare the BDD approach to the results reported in [21], where non-directed graphs are dealt with and both random pattern and target graphs have edge probability 25%, additional experiments

Table 2. Number of solutions and time in seconds (Ullmann approach) and number of solutions, BDD size in number of nodes, and time in seconds (BDD approach) taken to find all subgraph isomorphisms of a random graph with m vertices into a random graph with n vertices, both with 25% connectivity.

		Ullmann				BDD approach					
		CPU time		isomorphisms		CPU time		isomorphisms		BDD size	
m	n	mean	std. dev.	mean	std. dev.	mean	std. dev.	mean	std. dev.	mean	std. dev.
6	12	14.5	13.11	960.8	140.4	0.17	0.08	1 097.16	1 138.35	2 227.96	1 570.29
8	12	44.5	55.40	1 223.0	142.8	2.70	2.53	2 674.18	5 393.08	6 604.84	8 230.56
10	12	124.0	90.40	949.1	121.2	40.19	51.42	2 601.78	12 153.53	10 233.32	27 336.64
7	14	97.6	118.70	4 769.9	88.9	1.32	1.14	5 266.56	7 163.19	9 353.70	9 707.76
3	32					0.02	0.01	1 794.96	429.65	1 469.28	227.22
3	64					0.10	0.02	14 899.12	3 754.98	9 036.98	1 476.83
3	128					0.73	0.14	120 302.84	26 395.13	57 980.12	8 640.77
3	256					7.60	1.48	960 418.00	232 712.75	373 710.02	60 903.30
4	32					0.21	0.04	12 527.96	5 743.39	9 595.38	3 064.30
4	64					3.99	0.90	180 141.20	95 310.55	99 254.34	37 755.45
4	128					65.26	15.33	2 984 971.80	1 535 435.34	1 237 726.68	461 343.37
5	32					2.43	0.87	71 608.80	49 433.82	52 159.20	25 246.12
5	64					73.02	21.72	2 452 451.40	1 521 057.92	1 130 156.86	540 273.34

have been carried out representing every edge by a pairs of counter-parallel arcs. For each generated pattern and target random graphs, the Boolean matrix of the pattern has been added to the Boolean matrix of the target and non-connected graphs have been discarded, as was already done in [21]. Table 2 shows statistics for the approach by Ullmann (number of solutions and time in seconds) and the BDD approach (number of solutions, size of the BDD containing them, and time in seconds) for finding all subgraph isomorphisms of a random pattern graph into a random target graph, both with arc probability 25%. For each variable the mean and the standard deviation, which have been estimated over a sample of 50 observations, are given.

7 Conclusion

The subgraph isomorphism problem is dealt with in this paper. A novel approach is presented to the problem of finding *all* subgraph isomorphisms of a (pattern) graph into another (target) graph.

A relational formulation of the problem, combined with a representation of relations and graphs by Boolean functions, allows to handle the combinatorial explosion in the case of small pattern graphs and large target graphs by using *Binary Decision Diagrams* (BDDs), which are capable to represent large relations and graphs in small data structures.

Experimental results have shown that finding all subgraph isomorphisms of a small pattern graph into a large target graph becomes feasible for reasonably dense pattern graphs, for instance for random graphs with arc probability between 10 % and 25%. Moreover, for small pattern and target graphs and for arc probability of 25%, which corresponds to the subgraph isomorphism experiments reported in [21], the BDD approach outperforms all the results therein.

For very large target graphs, the approach is limited in the size of the pattern graph because BDDs representing all subgraph isomorphisms become too large. An open problem is therefore to find an optimal encoding of the target graph and optimal variable orderings in order to obtain smaller BDDs. Notice that these encoding problems can be solved off-line.

The approach can be extended in several directions. One the one hand, it can be extended in a straightforward way for finding all *full* subgraph isomorphisms, that is, subgraph isomorphisms that not only preserve but also reflect the structure of the pattern in the target. Notice that subgraphs and full subgraphs are called partial subgraphs and subgraphs, respectively, in [17].

On the other hand, a problem of increasing interest in the area of pattern recognition is that of approximate matching [10, 15]. Many applications in the area of pattern recognition are based on attributed graphs [6, 11, 18, 23], whereas vertex and arc attributes can be represented by MTBDDs (multi-terminal BDDs, a straightforward extension of BDDs and their algorithms). The techniques used in this paper can also be extended to MTBDDs in order to find *all* approximate subgraph isomorphisms, at varying edit distances.

Last, but not least, the relational formulation of the problem of finding all subgraph isomorphisms can also be used for finding *one* subgraph isomorphism, for instance by using satisfiability techniques. This is of interest to application areas such as image understanding, there is usually a small number of subgraph isomorphisms.

References

1. F. A. Akinniyi, A. K. C. Wong, and D. A. Stacey. A new algorithm for graph monomorphism based on the projections of the product graph. *IEEE Trans. on Systems, Man, and Cybernetics*, 16(5):740–751, 1986.
2. C. Brink, W. Kahl, and G. Schmidt, editors. *Relational Methods in Computer Science*. Springer-Verlag, 1997.
3. F. M. Brown. *Boolean Reasoning: The Logic of Boolean Equations*. Kluwer Academic Publishers, 1990.
4. R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. on Computers*, 35:677–691, 1986.
5. R. E. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.
6. H. Bunke and B. T. Messmer. Recent advances in graph matching. *Int. J. Pattern Recognition and Artificial Intelligence*, 11(1):169–203, 1997.
7. J. K. Cheng and T. S. Huang. A subgraph isomorphism algorithm using resolution. *Pattern Recognition*, 13:371–379, 1981.
8. M. R. Garey and D. S. Johnson. *Computers and intractability: A guide to NP-completeness*. Freeman, 1979.
9. D. E. Ghahraman, A. K. C. Wong, and T. Au. Graph monomorphism algorithms. *IEEE Trans. on Systems, Man, and Cybernetics*, 10:189–197, 1980.
10. S. Gold and A. Rangarajan. A graduated assignment algorithm for graph matching. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 18(4):377–388, 1996.
11. R. M. Haralick and J. S. Kartus. Arrangements, homomorphisms, and discrete relaxation. *IEEE Trans. on Systems, Man, and Cybernetics*, 8(8):600–612, 1978.
12. C. Y. Lee. Representation of switching circuits by binary-decision programs. *Bell System Technical Journal*, 38(4):985–999, 1959.
13. H.-T. Liaw and C.-S. Lin. On the OBDD representation of generalized Boolean functions. *IEEE Trans. on Computers*, 41(6):661–664, 1992.
14. J. J. McGregor. Relational consistency algorithms and their application in finding subgraph and graph isomorphisms. *Information Sciences*, 19:229–250, 1979.
15. B. T. Messmer and H. Bunke. A new algorithm for error-tolerant subgraph isomorphism detection. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 20(5):493–504, 1998.
16. S. K. Park and K. W. Miller. Random number generators, good ones are hard to find. *cacm*, 31:1192–1201, 1988.
17. G. Schmidt and T. Ströhlein. *Relations and Graphs: Discrete Mathematics for Computer Scientists*. Springer-Verlag, 1993.
18. L. G. Shapiro and R. M. Haralick. Structural descriptions and inexact matching. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 3(5):504–519, 1981.
19. F. Somenzi. CUDD: Colorado University Decision diagram package. `ftp://vlsi.colorado.edu/pub/`, 1996.
20. M. H. Stone. The theory of representations for Boolean algebras. *Trans. AMS*, 40:37–111, 1936.
21. J. R. Ullmann. An algorithm for subgraph isomorphism. *jacm*, 23(1):31–42, 1976.
22. G. Valiente and C. Martínez. An algorithm for graph pattern-matching. In *Proc. 4th South American Workshop on String Processing*, volume 8 of *Int. Informatics Series*, pages 180–197. Carleton University Press, 1997.
23. G. Vosselman. *Relational Matching*, volume 628 of *Lecture Notes in Computer Science*. Springer-Verlag, 1992.