A Direct Algorithm to Find a Largest Common Connected Induced Subgraph of Two Graphs

Bertrand Cuissart and Jean-Jacques Hébrard

Groupe de Recherche en Électronique, Informatique et Imagerie de Caen,
CNRS-UMR6072, Université de Caen, France
{cuissart, hebrard}@info.unicaen.fr

Abstract. We present a direct algorithm that computes a largest common connected induced subgraph of two given graphs. It is based on an efficient generation of the common connected induced subgraphs of the input graphs. Experimental results are provided.

1 Introduction

Graphs are widely used to represent objects in various domains such as chemical information, computer imaging etc [1–p. 527]. Many applications in these domains necessitate the use of similarity measures which are often based on the calculation of a largest common subgraph of two graphs [2, 3, 4, 5, 6] .

There are numerous definitions of a largest common subgraph, depending on the notions of subgraph and size of a graph taken into account. We are interested here in the case where the considered subgraphs are the connected induced subgraphs, and where the size of a graph is the number of its vertices. This notion has been successfully applied in chemistry [7].

Recall that given two graphs G_1, G_2 and a positive integer k, the problem of deciding whether there exists a common subgraph of G_1 and G_2 of size greater than k is NP-complete [8]. In fact, the problem remains NP-complete if we restrict the search to common connected induced subgraphs. However, in practice, it is useful to define specialized algorithms for this particular case, since they could be significantly more efficient than a general one. A first algorithm that computes a largest common connected subgraph of two graphs was presented by I. Koch in [9]. This algorithm can be straightforwardly adapted to compute a largest common connected *induced* subgraph. It reduces the problem to the search of a largest clique of a compatibility graph associated with the input graphs. In this paper, we present a direct algorithm based on a procedure that efficiently generates the common connected induced subgraphs of the two input graphs. We experimentally compare the two algorithms.

2 Preliminaries

A *graph* G is denoted by $G(V, E)$ where V is the set of its vertices and E the set of its edges. The subgraph of G *induced* by a subset of its vertices, $V' \subseteq V$,

L. Brun and M. Vento (Eds.): GbRPR 2005, LNCS 3434, pp. 162–171, 2005.
© Springer-Verlag Berlin Heidelberg 2005

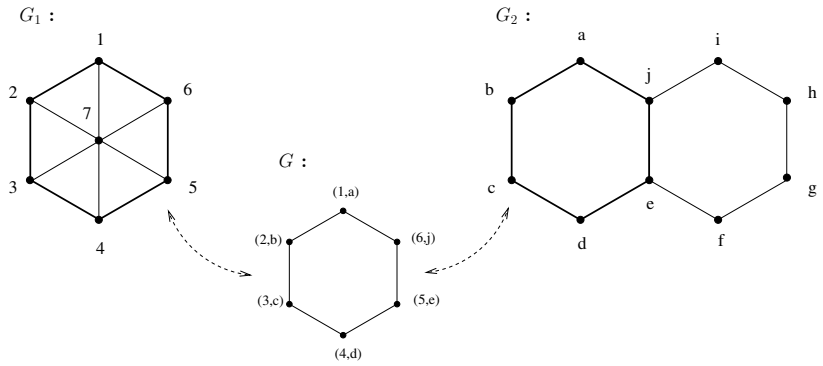

Fig. 1. A common subgraph of two graphs

is a graph consisting of V' and those edges of V with both vertices in V'. The subgraph of G induced by V' is denoted by $G[V']$. A graph G' is an induced subgraph of G if there exists $V' \subseteq V$ such that $G[V'] = G'$. *In all the paper, we say subgraph for induced subgraph.*

Two graphs $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ are *isomorphic* if there exists a bijection $f : V_1 \longrightarrow V_2$ such that for every $u_1, v_1 \in V_1$, $\{u_1, v_1\} \in E_1$ if and only if $\{f(u_1), f(v_1)\} \in E_2$; f is called an *isomorphism.* A *common subgraph of* G_1 *and* G_2 is a set of ordered pairs $\{(u_1, v_1), \ldots, (u_k, v_k)\}$ such that the function $f : \{u_1, \ldots, u_k\} \longrightarrow \{v_1, \ldots, v_k\}$ defined by $f(u_i) = v_i$ $(1 \leq i \leq k)$ is an isomorphism between $G_1[\{u_1, \ldots, u_k\}]$ and $G_2[\{v_1, \ldots, v_k\}]$.

Example 1. In Figure 1, $G = \{(1, a), (2, b), (3, c), (4, d), (5, e), (6, j)\}$ is a common subgraph of G_1 and G_2.

A graph G is *connected* if any two of its vertices are linked by a path in G. The problem *LCCIS (Largest Common Connected Induced Subgraph)* is to find a common connected induced subgraph C of two given graphs, such that the cardinality of C, $|C|$, is maximum.

Recall that a *tree* is a connected graph containing no circuit. The vertices of a tree will be called *nodes.* A *rooted tree* is a tree in which one node, the *root*, is distinguished. In a rooted tree, any node of degree one, unless it is the root, is called a *leaf.* If $\{u, v\}$ is an edge of a rooted tree such that u lies on the path from the root to v, then u is said to be the *parent* of v and v is a *child* of u. An *ancestor* of u is any node of the path from the root to u. If u is an ancestor of v, then v is a *descendant* of u, and we write $u \leq v$; if $u \neq v$, we write $u < v$. It is convenient to denote a tree T with root r by (T, r).

We present, in Section 5, an algorithm for LCCIS based on an efficient method for generating the common connected subgraphs of two graphs; this method is described in Section 4. But before generating the common connected subgraphs of two graphs, we must be able to simply generate the connected subgraphs of a graph; Section 3 is devoted to this question.

3 Connected Subgraphs Generation

We present a structure designed to generate the connected subgraphs of a graph.

Throughout this section, $G_1(V_1, E_1)$ is a fixed graph, and (T, r) is a rooted tree in which each node, except r, is labelled by an element of $\{v_+ : v \in V_1\} \cup \{v_- : v \in V_1\}$. The label of a node x, $x \neq r$, is denoted by $L(x)$. A label of the form v_+ (resp. v_-) is said to be *positive* (resp. *negative*). For each node x of T, we define $PL_+(x) = \{v \in V_1 : \exists y,\ y \neq r,\ y \leq x \text{ and } L(y) = v_+\}$; $PL_+(x)$ is the set of vertices of G_1 occurring in a positive label of an ancestor of x. Similarly, we define $PL_-(x) = \{v \in V_1 : \exists y,\ y \neq r,\ y \leq x \text{ and } L(y) = v_-\}$, and $PL(x) = PL_+(x) \cup PL_-(x)$; $PL(x)$ is the set of vertices of G_1 used for labelling the ancestors of x.

In order to generate the connected subgraphs of G_1, we build T in such a way that for each node x of T, $G_1[PL_+(x)]$ is connected. The set of vertices of G_1 that may label the children of x is denoted by $F(x)$ and defined as follows.

If $PL_+(x) \neq \emptyset$ then $F(x) = \{v \in V_1 : \exists u \in PL_+(x),\ \{v, u\} \in E_1\} \setminus PL(x)$, otherwise $F(x) = V_1 \setminus PL(x)$.

Example 2. In Figure 2, $PL_+(x) = \{a, b\}$, $PL(x) = \{a, b, c\}$, $F(x) = \{e\}$.

Remark that, by definition, $PL_+(r) = \emptyset$, $PL(r) = \emptyset$ and $F(r) = V_1$.

Definition 3. A tree of the connected subgraphs (TOCS) *of G_1 is a rooted tree (T, r) such that :*

1. *Every node of T, except r, is labelled by an element of $\{v_+ : v \in V_1\} \cup \{v_- : v \in V_1\}$.*
2. *For every node x of T, if $F(x) = \emptyset$, then x is a leaf, otherwise x has two children respectively labelled by v_+ and v_-, with $v \in F(x)$.*

See Figure 2 for an example of a TOCS.

We will now show that the leaves of a TOCS of G_1 and the connected subgraphs of G_1 are in a bijective correspondence. The following proposition is an immediate consequence of Definition 3.

Proposition 4. *Let f_1 and f_2 be two leaves of a TOCS. If $f_1 \neq f_2$ then $PL_+(f_1) \neq PL_+(f_2)$.*

Let $V' \subseteq V_1$ and (T, r) be a TOCS of G_1. We associate with V' the path of T, $(x_0, \ldots, x_k)$, defined as follows: $x_0 = r$, x_k is a leaf of T, and for all i $(0 \leq i < k)$, x_{i+1} is the child of x_i with a positive label v_+ if $v \in V'$, otherwise x_{i+1} is the child of x_i with a negative label. The leaf x_k is denoted by $l(V')$.

Example 5. In Figure 2, $l(\{a, b, e\}) = l(\{a, b, e, d\}) = f$.

Remark 6. Let $V' \subseteq V_1$. We have $PL_+(l(V')) \subseteq V'$, $PL_-(l(V')) \cap V' = \emptyset$ and $PL_+(l(V')) = \emptyset$ if and only if $V' = \emptyset$. Indeed, if $PL_+(l(V')) = \emptyset$, then $F(l(V')) = V_1 \setminus PL(l(V'))$, moreover, $F(l(V')) = \emptyset$ since $l(V')$ is a leaf, thus $PL(l(V')) = V_1$, $PL_-(l(V')) = V_1$, $V_1 \cap V' = \emptyset$ and consequently $V' = \emptyset$.

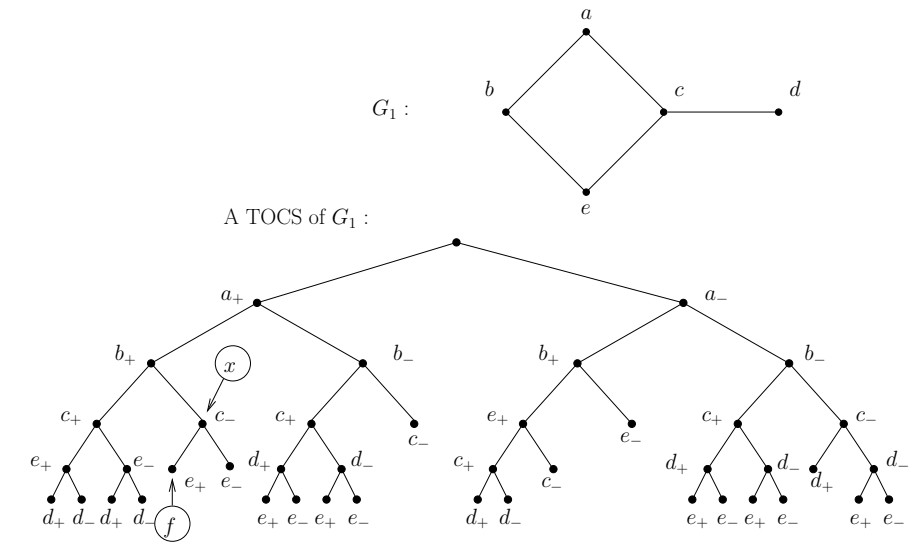

Fig. 2. A tree of the connected subgraphs of a graph

Proposition 7. *Let T be a TOCS of G_1.*

1. *(soundness) For every leaf f of T, $G_1[PL_+(f)]$ is connected.*
2. *(completeness) Let $V' \subseteq V_1$. If $G_1[V']$ is connected, then $V' = PL_+(l(V'))$.*

Proof. 1. Straightforward consequence of Definition 3.
2. 2. If $V' = \emptyset$ then $PL_+(l(V')) = \emptyset$ (Remark 6) and $V' = PL_+(l(V'))$.

If $V' \neq \emptyset$, then $PL_+(l(V')) \neq \emptyset$ (Remark 6). By definition of $l(V')$, we have $PL_+(l(V')) \subseteq V'$. Let $C = V' \setminus PL_+(l(V'))$. Suppose $C \neq \emptyset$. The sets C and $PL_+(l(V'))$ form a partition of V'. By hypothesis, $G[V']$ is connected, thus there exists an edge $\{u, v\} \in E_1$, with $u \in C$ and $v \in PL_+(l(V'))$. By definition of $l(V')$, $C \cap PL(l(V')) = \emptyset$, therefore $u \in F(l(V'))$. Now $l(V')$ is a leaf of T, thus $F(l(V')) = \emptyset$. A contradiction. Finally, $C = \emptyset$ and $V' = PL_+(l(V'))$. □

Example 8. In Figure 2, $G_1[\{a, b, e\}]$ is connected, $G_1[\{a, b, e, d\}]$ is not connected and $PL_+(f) = \{a, b, e\}$.

It follows from Prop. 4 and Prop. 7 that there exists a bijection from the leaves of a TOCS of G_1 to the connected subgraphs of G_1. As a consequence, the connected subgraphs of G_1 can be generated by simply performing a depth first traversal of a TOCS of G_1 [10].

4 Generation of the Common Connected Subgraphs of Two Graphs

We present an algorithm for generating the common connected subgraphs of two graphs.

Throughout this section, $G_1(V_1, E_1)$ and $G_2(v_2, E_2)$ are fixed graphs and T is a TOCS of G_1. For every leaf f of T, our algorithm build simultaneously all the subgraphs of G_2 isomorphic to $G_1[PL_+(f)]$.

We first define a rooted tree whose leaves are in a bijective correspondence with the common connected subgraphs of G_1 and G_2.

Definition 9. *Let (T, r) be a TOCS of G_1. The* tree of the common connected subgraphs of G_1 and G_2 (TOCCS) built from T *is a rooted tree (T', r') in which each node,except r', is labelled by an element of $V_1 \times V_2 \cup V_1 \times \{$"Excluded"$\}$. The label of a node x', $x' \neq r'$, is denoted by $L'(x')$. Each node x' of T' is associated with a node of T, called the* origin *of x' and denoted by $Or(x')$. (T', r') and the function Or are defined by induction as follows:*

1. *$Or(r') = r$.*
2. *Let x' be a node of T'.*
   (a) *If $Or(x')$ is a leaf of T, then x' is a leaf of T'.*
   (b) *Otherwise, $Or(x')$ has two children y and z respectively labelled by v_{1+} and v_{1-}, where $v_1 \in V_1$. Let $H(x') = \{v_2 \in V_2 :$ for any ancestor of x' labelled by (u_1, u_2) with $u_2 \in V_2, v_2 \neq u_2$ and $(\{v_1, u_1\} \in E_1 \iff \{v_2, u_2\} \in E_2)\}$, $k = |H(x')|$ and $w_1, \ldots, w_k$ denote the elements of $H(x')$. Then x' has $k+1$ children, $y'_1, \ldots, y'_{k+1}$, with $L'(y'_i) = (v_1, w_i)$ and $Or(y'_i) = y$ $(1 \leq i \leq k)$, $L'(y'_{k+1}) = (v_1,$ "Excluded") and $Or(y'_{k+1}) = z$.*

Example 10. Figure 3 displays a path from the root r' to a node x' in a TOCCS of the graphs G_1 and G_2. $Or(x') = x$, $F(x) = \{e\}$, x has two children y and z with $L(y) = e_+$ and $L(z) = e_-$. $H(x') = \{5\}$, x' has two children y' and z' with $L'(y') = (e, 5)$ and $L'(z') = (e,$"Excluded"$)$.

In a TOCCS, the labels of the form (v_1, v_2), with $v_2 \in V_2$, are called *positive* and the labels of the form $(v_1,$"Excluded"$)$ are called *negative*. For every node x' of a TOCCS, we define $PL'_+(x') = \{(v_1, v_2) \in V_1 \times V_2 : \exists y',\ y' \neq r',\ y' \leq x'$ and $L'(y') = (v_1, v_2)\}$.

The following proposition is an immediate consequence of Definition 9.

Proposition 11. *Let T' be a TOCCS of G_1 and G_2, and f'_1, f'_2 two leaves of T'. If $f'_1 \neq f'_2$ then $PL'_+(f'_1) \neq PL'_+(f'_2)$.*

We now establish the correspondence between the leaves of a TOCCS and the common connected subgraphs of G_1 and G_2.

Proposition 12. *Let T' be a TOCCS of G_1 and G_2.*

1. *(soundness) For every leaf f' of T', $PL'_+(f')$ is a common connected subgraph of G_1 and G_2.*
2. *(completeness) Let G be a common connected subgraph of G_1 and G_2. There exists a leaf f' of T', such that $PL'_+(f') = G$.*

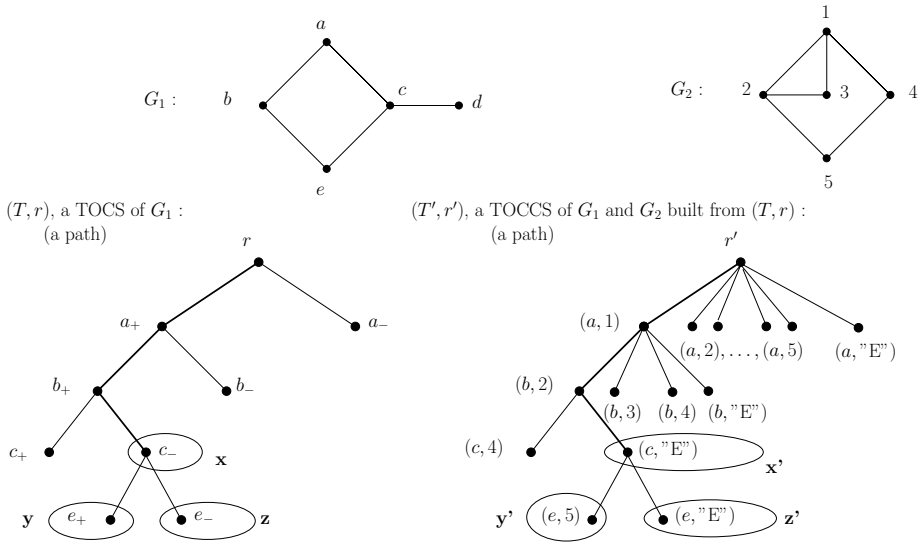

Fig. 3. A tree of the connected subgraphs common to two graphs

Proof (Sketch). 1. Straightforward consequence of Definition 9.
2. Let $G = \{(u_1, v_1), \ldots, (u_k, v_k)\}$ be a common connected subgraph of G_1 and G_2, $u_i \in V_1$ and $v_i \in V_2$ $(1 \leq i \leq k)$. Let (T, r) be the TOCS of G_1 from which (T', r') is built. By hypothesis, $G_1[\{u_1, \ldots, u_k\}]$ is connected. According to Prop. 7, we have $PL_+(l(\{u_1, \ldots, u_k\})) = \{u_1, \ldots, u_k\}$. Let $(x_0, \ldots, x_p)$ be the path from r to $l(\{u_1, \ldots, u_k\})$ in T.

For all i, $1 \leq i \leq p$, if $L(x_i)$ is a negative label, we write $\sigma(i) =$ "Excluded", otherwise, there exists a unique vertex $v \in V_2$ such that $(L(x_i), v) \in G$ and we write $\sigma(i) = v$.

Since G is a common connected subgraph of G_1 and G_2, there exists a path in T' $(x'_0, \ldots, x'_p)$ from r' $(r' = x'_0$) to a leaf f' $(f' = x'_p)$ such that $L'(x'_i) = (L(x_i), \sigma(i))$ $(1 \leq i \leq p)$. We have $PL'_+(f') = G$. □

It follows from Prop. 11 and Prop. 12 that there exists a bijection from the the leaves of a TOCCS of two graphs to their common connected subgraphs. Consequently, we can generate the common connected subgraphs of G_1 and G_2 by simply performing a depth first traversal of a TOCCS of G_1 and G_2. Notice that it is not necessary to compute and store the entire TOCCS. A depth first traversal only requires to store the path from the root to the node currently visited.

5 The Algorithm

A first algorithm for solving LCCIS consists in generating the common connected subgraphs of the two input graphs and returning a largest one. This algorithm

performs a complete traversal of a TOCCS of the input graphs. We will see that, in fact, parts of the TOCCS may be ignored.

Throughout this section, $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ are fixed graphs, (T, r) is a TOCS of G_1, and (T', r') is a TOCCS of G_1 and G_2 built from (T, r). Recall that for every node x of T, $PL_-(x) = \{v \in V_1 : \exists y,\ y \neq r,\ y \leq x \text{ and } L(y) = v_-\}$. For each node x' of T', we define $PL'_-(x') = \{(v_1, \text{"Excluded"}) : \exists y',\ y' \neq r', y' \leq x' \text{ and } L'(y') = (v_1, \text{"Excluded"})\}$. The following propositions allow us not to entirely examine the TOCCS explored by the algorithm.

Proposition 13. *Let $m \geq 0$ be an integer and x' a node of T'. If $|V_1| - |PL'_-(x')| \leq m$, then $|PL'_+(y')| \leq m$, for every descendant y' of x'.*

Remark that $|V_1| - |PL'_-(x')|$ is the number of vertices of V_1 not yet excluded, when x' is visited. Suppose the search has already discovered a common subgraph of size m. If $|V_1| - |PL'_-(x')| \leq m$, then it is useless to explore the principal subtree of T' at x', since, according to Prop. 13, we would not find any common subgraph of size greater than m.

Proposition 14. *Let x' be a node of T' such that $PL'_+(x') \neq \emptyset$, and f'_1 a leaf of T' descending from x' ($x' \leq f'_1$). If for every y' such that $x' < y' \leq f'_1$, $L'(y')$ is positive, then, for every leaf f'_2 descending from x', we have $|PL'_+(f'_2)| \leq |PL'_+(f_1)|$.*

Proof (Sketch). Let $x = Or(x')$, $f_1 = Or(f'_1)$ and $f_2 = Or(f'_2)$ (Definition 9). $G_1[PL_+(f_1)]$ is the largest connected subgraph of G_1 that contains the vertices of $PL_+(x)$ and does not contain the vertices of $PL_-(x)$. By hypothesis, $PL'_+(x') \neq \emptyset$, thus $PL_+(x) \neq \emptyset$. Consequently, $G_1[PL_+(f_2)]$ contains the vertices of $PL_+(x)$ and does not contain the vertices of $PL_-(x)$. Hence, $PL_+(f_2) \subseteq PL_+(f_1)$ and $|PL'_+(f'_2)| \leq |PL'_+(f'_1)|$.□

Let x' be a node of T' such that $PL'_+(x') \neq \emptyset$. After having explored a path p from x' to a leaf f'_1, such that p only contains nodes positively labelled (except x' possibly), it is useless to explore the other paths beginning at x'. Indeed we are sure, from Prop. 14, that these paths do not lead to any common subgraph larger than the one given by $PL'_+(f'_1)$.

Finally, our algorithm partially explores a TOCCS in a depth first manner, ignoring parts of it that cannot provide a common subgraph larger than the current largest one already found.

6 Experimental Results

It is essential to characterize graph matching algorithms by their performances [11, 12]. We experimentally compared our algorithm (T-TOCCS) with the algorithm based on the method proposed by I. Koch (C-CLIQUE) [9].

Material. The experiments were realized on a computer with 1 GB of memory and the AMD Athlon processor (2,2 GHz). We implemented both algorithms

Table 1. Experimental results

Randomly Connected Graphs, fixed density.									
S	M_C	E_C	M_T	E_T	S	M_C	E_C	M_T	E_T
	Density =0.1					Density =0.2			
21	5.235	3.37	0.462	0.345	16	1.259	0.133	0.172	0.0336
22	6.366	2.007	0.469	0.22	17	3.049	0.369	0.412	0.08
23	19.604	5.023	1.4	0.602	18	7.949	0.995	1.289	0.318
24	56.197	27.122	4.405	2.342	19	18.567	1.953	3.64	0.584
25	119	70.976	8.628	6.663	20	45.966	4.805	9.375	1.765
26	286.748	105.66	21.162	9.521	21	115.653	11.851	22.631	6.003
	Density =0.5					Density =0.85			
14	1.091	0.0332	0.272	0.048	10	0.7	0.07	0.017	0.0169
15	2.656	0.086	0.601	0.127	11	4.61	0.982	0.101	0.0712
16	6.049	0.228	1.397	0.052	12	14.658	1.974	0.17	0.16
17	13.821	0.451	3.848	0.394	13	62.386	14.122	0.913	0.826
18	31.078	0.95	9.718	1.6	14	242.79	36.954	3.858	2.966

Randomly Connected Graphs, fixed size.									
D	M_C	E_C	M_T	E_T	D	M_C	E_C	M_T	E_T
	Size = 18					Size = 22			
0.2	5.927	0.739	0.763	0.141	0.1	6.366	2.007	0.469	0.220
0.3	16.68	1.323	4.29	0.346	0.15	75.005	14.696	8.365	3.599
0.4	19.82	1.745	5.92	0.476	0.2	264.17	22.162	30.29	7.871
0.5	31.078	0.950	9.718	1.600					

Irregular 2d Meshes.									
D	M_C	E_C	M_T	E_T	D	M_C	E_C	M_T	E_T
Size = 9					Size = 16				
0.333	0	0	0	0	0.208	1.394	0.092	0.008	0.0064
0.417	0.009	0.0018	0	0	0.3	3.119	0.229	0.649	0.081
0.528	0.01	0	0	0	0.4	4.418	0.261	1.337	0.113
0.611	0.01	0	0	0	0.5	6.048	0.248	1.427	0.13
0.722	0.03	0	0	0	0.6	12.55	0.898	1.702	0.267
0.833	0.125	0.019	0	0	0.7	52.403	6.697	2.842	1.298
0.917	0.716	0.066	0.004	0.0064	0.75	127.571	12.325	4.995	1.836
1	13.173	0.0042	0	0					

S : Size of the input graphs, D : Density of the input graphs,

C-CLIQUE :
- M_C : Mean duration of the calculus (in seconds).
- E_C : Standard deviation.

T-TOCCS :
- M_T : Mean duration of the calculus (in seconds).
- E_T : Standard deviation.

in C, and compiled the code with the GNU gcc 3.3 compiler with appropriate optimizations.

The algorithms were tested on two kinds of graphs : *randomly connected graphs* and *irregular 2D meshes*. These graphs are described by M. De Santo et al. in [13, 14]. Notice that they are always connected. In a *randomly connected graph*, the probability of an edge connecting two vertices is independant on the vertices themselves. We adopted the model proposed in [15] to generate our instances. Since it is generally agreed that irregular 2D meshes represent a worst case for general graph matching algorithms [15], we also considered these graphs.

Results. We have compared the performances of the two algorithms on the same instances. The results are displayed on three tables (see Table 1). The first two tables concern the results obtained with randomly connected graphs, the third table is dedicated to irregular 2D meshes. A cell of these tables corresponds to a measure on a sample of ten instances. For example, the cell at the top left corner of the first table indicates that C-CLIQUE has taken on average 5.235 seconds to find a LCCIS between two randomly connected graphs of size 21 and of density 0.1. For each series, the measures were stopped as soon as the time required to solve one instance exceeded 500 seconds.

Discussion. The examination of the tables shows that T-TOCCS operates faster than C-CLIQUE on these instances.

When the input graphs are randomly connected graphs of density $\eta = 0.5$, T-TOCCS solves the problem 3 times faster than C-CLIQUE. This difference increases as the density of the input graphs moves away from 0.5. T-TOCCS operates on average 10 times faster than C-CLIQUE when the density is equal to 0.1. It operates 40 times faster when the input graphs have high density ($D = 0.85$).

The observations are similar for the 2D meshes. When the density of the meshes is close to 0.5, T-TOCCS operates about 3 times faster than C-CLIQUE. This ratio rapidly increases to reach 170 when the input graphs are the regular 2D meshes with 16 vertices. On the other side, this ratio reaches 25 when the input graphs have a density of 0.75.

References

1. Rosen, K., ed.: 8 : Graph theory, 9 : Trees. In: Handbook of discrete and combinatorial mathematics. CRC Press (2000) 495–628
2. Koch, I., Lengauer, T., Wanke, E.: An algorithm for finding common subtopologies in a set of protein structures. J. Comput. Biol. **3** (1996) 289–306
3. Koch, I., Lengauer, T.: Detection of distant structural similarities in a set of proteins using a fast graph based method. In Gaasterland, T., Karp, P., Karplus, K., Ouzounis, C., Sander, C., Valencia, A., eds.: Proc. 5th Int. Conf. on Intelligent systems for molecular biology, Menlo Park, AAAI Press (1997) 167–178
4. Raymond, J., Gardiner, E., Willett, P.: Heuristics for similarity searching of chemical graphs using a maximum common edge subgraph algorithm. J. Chem. Inf. Comput. Sci. **42** (2002) 305–316

5. Raymond, J., Gardiner, E., Willett, P.: RASCAL : calculation of graph similarity using maximum common edge subgraphs. Comp. J. **45** (2002) 632–643
6. Garciá, G., Ruiz, I., Gómez-Nieto, M.: Step-by-step calculation of all maximum common substructures through a constraint satisfaction based algorithm. J. Chem. Inf. Comput. Sci. **44** (2004) 30–41
7. Cuissart, B., Touffet, F., Crémilleux, B., Bureau, R., Rault, S.: The maximum common substructure as a molecular depiction in a supervised classification context: experiments in quantitative structure/biodegradability relationships. J. Chem. Inf. Comput. Sci. **42** (2002) 1043–1052
8. Garey, M., Johnson, D.: Computers and Intractability: a guide to the theory of NP-completeness. Freeman (1979)
9. Koch, I.: Enumerating all connected maximal common subgraphs in two graphs. Theor. Comput. Sci. **250** (2001) 1–30
10. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: 22. Elementary graph algorithms. In: Introduction to algorithms. McGraw-Hill (2001)
11. Bunke, H., Foggia, P., Guidobaldi, C., Sansone, C., Vento, M.: A comparison of algorithms for maximum common subgraph on randomly connected graphs. In Caelli, T., Amin, A., Duin, R.P.W., Kamel, M.S., de Ridder, D., eds.: Joint IAPR International Workshops SSPR 2002 and SPR 2002, Proceedings. Volume 2396 of Lecture Notes in Computer Science., Springer-Verlag (2002) 123–132
12. Conte, D., Guidobaldi, C., Sansone, C.: A comparison of three maximum common subgraph algorithms on a large database of labeled graphs. In Hancock, E., Vento, M., eds.: IAPR Workshop GbRPR. Volume 2726 of Lecture Notes in Computer Science., Springer-Verlag (2003) 130–141
13. Foggia, P., Sansone, C., Vento, M.: A database of graphs for isomorphism and subgraph isomorphism benchmarking. In: Proc. 3rd IAPR TC-15 GbR Worshop. (2001) 176–187
14. DeSanto, M., Foggia, P., Sansone, C., Vento, M.: A large database of graphs and its use for benchmarking graph isomorphism algorithms. Pattern Recogn. Lett. **24** (2003) 1067–1079
15. Ullman, J.: An algorithm for subgraph isomorphism. Journal of the Association for Computing Machinery **23** (1976) 31–42