Available online at www.sciencedirect.com
SCIENCE DIRECT®
Theoretical Computer Science 349 (2005) 484–490
Theoretical Computer Science
www.elsevier.com/locate/tcs

Note

An algorithm for reporting maximal *c*-cliques

F. Cazals[a,*], C. Karande[b]

[a] *INRIA Sophia-Antipolis, France*
[b] *IIT Bombay, India*

Received 26 July 2005; received in revised form 2 September 2005; accepted 2 September 2005

Communicated by G. Ausiello

Abstract

Given two graphs, a fundamental task faced by matching algorithms consists of computing either the (connected) maximal common induced subgraphs ((C)MCIS) or the (connected) maximal common edge subgraphs ((C)MCES). In particular, computing the CMCIS or CMCES reduces to reporting so-called *c-connected cliques* in product graphs, a problem for which an algorithm has been presented in I. Koch, Fundamental study: enumerating all connected maximal common subgraphs in two graphs, Theoret. Comput. Sci. 250 (1-2), (2001) 1–30. This algorithm suffers from two problems which are corrected in this note.

Keywords: Maximal c-connected cliques; Maximal-cliques; Maximal common induced/edge subgraphs; Shape matching

1. Introduction

Given two graphs, *partial combinatorial shape matching* is the problem concerned with the calculation of common patterns of the graphs. The natural way to address partial combinatorial shape matching consists of seeking the largest common induced subgraph (LCIS) or largest common edge subgraph (LCES) between the graphs, which are NP-hard problems. Alternatively, one may be interested not only in the largest common graphs, but in all maximal common graphs. Then, one seeks maximal common induced subgraphs (MCIS) or maximal common edge subgraphs (MCES) between the graphs. Notice that these are enumeration rather than optimization problems, so that an important parameter of algorithms solving these problems is their output-sensitivity [1]. One may also wish to further restrict the problems by focusing on connected MCIS or MCES, which yields the CMCIS and CMCES variants. These variants are ubiquitous in applications, among which computational structural biology [8,3,7,2].

As noticed long ago [5], reporting the MCIS can be done using the vertex product graph of the graphs, while MCES can be reported using the edge product graph [9,6,4]. Moreover, as noticed in [4], the problems of reporting the CMCIS or CMCES are similar to those of MCIS and MCES, and reduce to seeking *c-connected cliques* in the product graphs —see definitions below. An algorithm solving the problem, *C*-Clique, has been presented in [4]. Unfortunately, this algorithm is not complete, and the purpose of this note is to correct it.

* Corresponding author. Tel. : +33 4 92 38 71 88.
E-mail addresses: frederic.cazals@inria.fr (F. Cazals), ckarande@gmail.com (C. Karande).

0304-3975/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2005.09.038

2. Notations

A graph is denoted $G = (V[G], E[G])$. Given a node $u \in G$, $N[u]$ denotes the neighbors of u, i.e. $N[u] = \{v \mid (u, v) \in E[G]\}$. We assume a node is not a neighbor of itself. The edges of our graph will carry labels in the enum $\{c, d\}$. For such a graph, neighbors of a node u through a c-edge and a d-edge are, respectively, denoted $C[u]$ and $D[u]$. Notice that $N[u] = C[u] \cup D[u]$.

Given a graph, consider two disjoint subsets A and B of $V[G]$. Subset A is called *strongly connected* to B, which is denoted $A \vDash > B$, if every node of A is connected to every node of B. The notion of strongly connected disjoint subsets extends to graphs with labeled edges. However, because of the distinct labels, we have to specify what kind of connectivity is maintained between two disjoint sets. We introduce the following notations for graphs with edges in enum $\{c, d\}$:

- $A \vDash c^+d^* \Rightarrow B$: Every node of A is connected to every node of B by a c-edge or a d-edge and $\forall u \in A\ \exists v \in B$ such that (u, v) is a c-edge.
- $A \vDash c^*d^* \Rightarrow B$: Every node of A is connected to every node of B by either a c-edge or a b-edge.
- $A \vDash d^* \Rightarrow B$: Every node of A is connected to every node of B by a d-edge.

If there exists a set B such that $B \vDash r \Rightarrow A$ according to one of the previous notations, then the largest set S such that $S \vDash r \Rightarrow A$ is given by $S = SC(A, r)$.

Finally, given a graph whose edges are labeled, we refine the notion of clique as follows:

Definition 1 (*c-clique*). A c-clique is a clique consisting of c-edges and d-edges which is c-connected —i.e. connected through c-edges.

3. The original algorithm for *c*-cliques

3.1. The algorithm

An Algorithm to report all maximal c-cliques in a vertex product or an edge product graph is given by [4]. Figs.1 and 2 present the pseudo-code of these algorithms —the later uses set-operations rather than individual operations on nodes [4]. The algorithm maintains four sets R, Q, P, X. (Notice these sets are denoted C, D, P, S in [4].) Set R is the c-clique under expansion; set P holds prospective nodes —each such node form is connected to R through c or d edges, with at least one c-edge); set Q holds nodes strongly connected to R through d-edges only; set X contains forbidden nodes, i.e. nodes already processed. The expansion of R runs through c-edges, which requires recovering from Q nodes which are c-connected to the new addition —a fishing out process. A c-clique can be reported even if $Q \neq \emptyset$ since these nodes are d-connected to the set R. The algorithm is presented on Fig. 2. Although this algorithm proceeds in the right direction, it is incomplete for two reasons.

Initialization. First, we must point out that the initialization algorithm listed in [4], Fig. 1, does not in fact call the function C-Clique(R, P, Q, X) with correct arguments. The problem lies in the fact that in its current form, nodes in T can be added to Q during initialization. Q represents the set of nodes which can at some stage be added to the growing cliques, whereas T is the set of already processed and hence forbidden nodes.

As an example, consider the graph of Fig. 3, and run the algorithm in Fig. 1 on the input product graph shown in Fig. 3, assuming the node (b, y) is processed after (a, x) and before (c, z). We can see that when inside the outer loop, $u = (b, y)$, (a, x) will be in T, and yet the call to algorithm made is C-Clique$(\{(b, y)\}, \{(c, z)\}, \{(a, x)\}, \emptyset)$. In other words, the node (a, x) appears in Q, in spite of having been processed earlier. In fact, if we follow the execution of this algorithm further, we find that eventually (a, x) enters R via P, and hence the maximal c-clique $\{(a, x), (b, y), (c, z)\}$ is reported twice. (It has already been reported once when (a, x) was processed by the initialization algorithm.)

This flaw can be rectified by a slight change in line 14 of the algorithm in Fig. 1. A check can be made, so that node v is added to Q only if it does not belong to T. For the following discussion, we assume this correction.

On the symmetric role of P and X. In spite of this correction, the algorithm C-Clique suffers from a second problem, inherently related to the asymmetric way sets P and X are handled —while the two sets maintain the same connectivity to R. According to the invariant (iii) listed on page 17 of [4], $X \vDash c^*d^* \Rightarrow R$ holds. Note again that the initialization

C-CLIQUE-INIT(G)

1: Input is the graph G. This function initializes the sets R, P, Q and X.
2: Let T be the set of nodes already been used in the initialization.
3: $T \leftarrow \emptyset$
4: **for all** $u \in V[G]$ **do**
5: $P \leftarrow \emptyset$
6: $Q \leftarrow \emptyset$
7: $X \leftarrow \emptyset$
8: **for all** $v \in N[u]$ **do**
9: **if** u and v are adjacent via a c-edge **then**
10: **if** $v \in T$ **then**
11: $X \leftarrow X \cup \{v\}$
12: **else**
13: $P \leftarrow P \cup \{v\}$
14: **else if** u and v are adjacent via a d-edge **then**
15: $Q \leftarrow Q \cup \{v\}$
16: C-Clique$(\{u\}, P, Q, X)$
17: $T \leftarrow T \cup \{u\}$

Fig. 1. Initialization algorithm for C- Clique.

Algorithm call: From a wrapper algorithm which performs the proper initializations of P and Q. See [Koc01].

C-CLIQUE(R, P, Q, X)

1: {returns all R_{max} which are maximal c-cliques such that $R \subset R_{max}$, $R_{max} \cap X = \emptyset$ and $R_{max} \subseteq R \cup P$.}
2: **if** $P = \emptyset$ and $X = \emptyset$ **then**
3: Report R as a maximal c-clique
4: **else**
5: Assume $P = \{u_1, u_2, ..., u_k\}$
6: **for** $i \leftarrow 1$ to k **do**
7: //add new node!
8: $R_{new} = R \cup \{u_i\}$
9: $P = P - \{u_i\}$
10: $P_{new} = P \cap N[u_i] \cup (C[u_i] \cap Q)$
11: $Q_{new} = Q \cap D[u_i]$
12: $X_{new} = X \cap N[u_i]$
13: C-Clique$(R_{new}, P_{new}, Q_{new}, X_{new})$
14: $X = X \cup \{u_i\}$

Fig. 2. Algorithm C- Clique to report c-cliques.

algorithm in [4] contradicts with this, since it adds to X only nodes c-connected to u: see line 11 on Fig. 1. In any case, we can show that regardless of whether X maintains c^*d^* or c^+d^* connectivity to R, the algorithm C-Clique fails.

- Assume $X \models c^*d^* \Rightarrow R$ holds: In this case, consider the situation that P is empty but X is not empty and all the elements in X are d-connected to all members of R. Clearly, we cannot extend R with any node in X because c-connectivity is not present. Hence, the current c-clique R is maximal. But we can see that because X is not empty, it will not be reported.

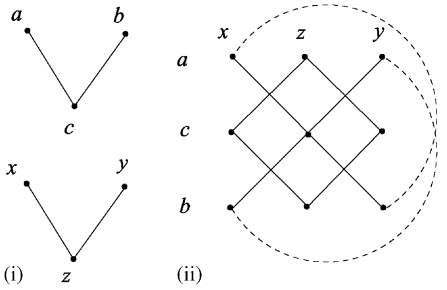

Fig. 3. (i) Example input graphs and (ii) the resultant product graph. Dashed edges in the product graph are d-edges while all others are c-edges. The label of a node in the product graph can be constructed from row and column labels.

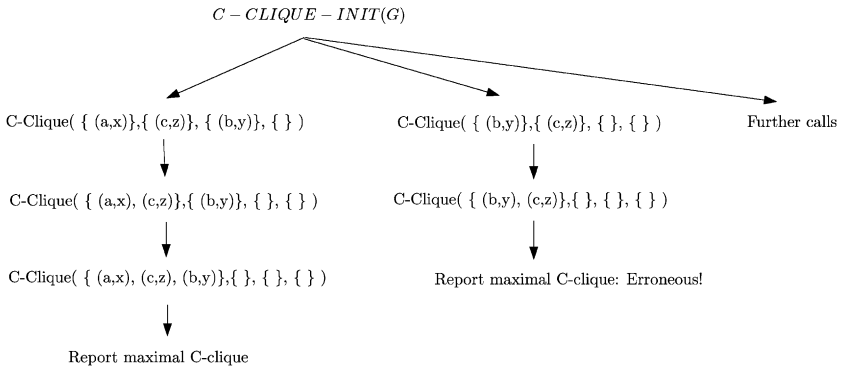

Fig. 4. Part of the recursion tree formed by executing C-Clique algorithm in Fig. 2 on product graph in Fig. 3.

- Assume $X \vDash c^+d^* \Rightarrow R$ holds: We show below that the same input graph in Fig. 3 creates a failure for this algorithm. But before that, let us intuitively examine where the flaw lies. The sets P and Q together contain all the prospective nodes. But the algorithm assumes that X alone contains all the relevant forbidden nodes. This asymmetry is counterintuitive, since P and X are supposed to maintain the same connectivity to R, and Q plays an integral role in maintaining P's connectivity. In case of X, we have no counterpart of the 'fishing out' operation which brings nodes from Q into P. This will result in some nodes missing in X, leading to reporting of non-maximal c-cliques.

Part of the recursion tree of execution of C-Clique algorithm on input graphs in Fig. 3 is given in Fig. 4. Note that we follow the execution only till two branches because at the end of the second branch, a *non-maximal* clique is reported. Note that during the initialization of the second branch, (a, x), which although is a d-neighbor of (b, y) is not added to Q since we have assumed the corrected initialization algorithm.

4. Corrected C-Clique algorithm

Having discussed the two troubles of algorithm C-Clique, let us examine the corresponding fixes. First, the correct initialization is presented on Fig. 5. Second, we handle the fishing out process of forbidden nodes through a new set Y, which is the counterpart of Q for X. See Fig. 6 for connectivities that are preconditions for the parameters.

One can check by direct inspection from the way the new sets $P_{\text{new}}, Q_{\text{new}}, X_{\text{new}}, Y_{\text{new}}$ are formed, that these preconditions are maintained, and they are enforced by the initialization algorithm to begin with. The two completeness invariants may not be obvious, hence we shall prove them :

Lemma 1. *Algorithm C-Clique maintains the following connectivities*: $SC(R, d^*) = Q \cup Y$ *and* $SC(R, c^+d^*) = P \cup X$.

```
C-CLIQUE-INIT(G)
 1: T ← φ
 2: Let V[G] = {u_1, u_2, ..., u_k}
 3: for i = 1 to k do
 4:     R ← {u_i}
 5:     Q ← (V[G] − T) ∩ D[u_i]
 6:     P ← (V[G] − T) ∩ C[u_i]
 7:     Y ← D[u_i] ∩ T
 8:     X ← C[u_i] ∩ T
 9:     C-Clique(R, P, Q, X, Y)
10:     T ← T ∪ {u_i}
```

Fig. 5. Corrected initialization algorithm for C-Clique.

Algorithm call : Will be made from C-Clique-Init

Pre-conditions:

$Q \models d^* \Rightarrow R$ $\quad SC(R, d^*) = Q \cup Y$
$P \models c^+ d^* \Rightarrow R$ $\quad SC(R, c^+ d^*) = P \cup X$
$X \models c^+ d^* \Rightarrow R$ $\quad (P \cup Q) \cap (X \cup Y) = \emptyset$
$Y \models d^* \Rightarrow R$

```
C-CLIQUE(R, P, Q, X, Y)
 1: if P = φ and X = φ then
 2:     ReportClique(R)
 3: else
 4:     Let P = {u_1, u_2, ..., u_k}
 5:     for i = 1 to k do
 6:         P ← P − u_i
 7:         R_new ← R ∪ {u_i}
 8:         Q_new ← Q ∩ D[u_i]
 9:         P_new ← (P ∩ N[u_i]) ∪ (Q ∩ C[u_i])
10:         Y_new ← Y ∩ D[u_i]
11:         X_new ← (X ∩ N[u_i]) ∪ (Y ∩ C[u_i])
12:         C-Clique(R_new, P_new, Q_new, X_new, Y_new)
13:         X ← X ∪ {u_i}
```

Fig. 6. Corrected algorithm C-Clique.

Proof. We start with $SC(R, d^*) = Q \cup Y$. The fact that the initialization algorithm enforces this invariant to begin with is trivial. All the prospective nodes d-connected to u_i are added to Q and all the forbidden nodes with same connectivity are added to Y. To show how the invariant is maintained during algorithm C-Clique, we observe the following property: $SC(A \cup \{v\}, d^*) = SC(A, d^*) \cap D[v]$. Hence we have

$$\begin{aligned} SC(R_{\text{new}}, d^*) &= SC(R \cup \{u_i\}, d^*) \\ &= SC(R, d^*) \cap D[u_i] \\ &= (Q \cup Y) \cap D[u_i] \\ &= (Q \cap D[u_i]) \cup (Y \cap D[u_i]) \\ &= Q_{\text{new}} \cup Y_{\text{new}}. \end{aligned}$$

Consider next $SC(R, c^+d^*) = P \cup X$. Again the initial enforcement of the invariant is easy to verify. To prove the maintenance, we observe the following property : $SC(A \cup \{v\}, c^+d^*) = (SC(A, c^+d^*) \cap N[u_i]) \cup (SC(A, d^*) \cap C[u_i])$. Again solving and substituting like above, one arrives at $SC(R_{\text{new}}, c^+d^*) = P_{\text{new}} \cup X_{\text{new}}$. □

Notice in above-mentioned preconditions and in the algorithm in Fig. 6 the symmetry between P and X, as well as between Q and Y, in term of their connectivities and the way they change during the course of the algorithm. Finally, we prove the correctness of the algorithm, that is

Theorem 1. *Algorithm C-Clique is such that*:

1. *Only maximal c-cliques as defined in Section* 2 *are reported.*
2. *A maximal c-clique is reported at most once.*
3. *All c-cliques are reported.*

Proof. 1. The stated preconditions, i.e. connectivities of various sets with R automatically imply that R, at each stage is a c-clique. Note that R is extended only by nodes in P which are c^+d^* connected to R. To prove that only maximal c-cliques are reported, observe that we report a c-clique only when $P = \emptyset$ and $X = \emptyset$. Since $SC(R, c^+d^*) = P \cup X = \emptyset$ no larger subgraph satisfying required properties can be formed by adding a node to R. Hence R must be maximal, and is reported.

2. We prove this by induction on the recursive calls.

Hypothesis : Consider a call C_1 to the function C-Clique. Let $C_{11}, C_{12}, ..., C_{1k}$ be the recursive calls made from C_1. If none of the C_{1i}'s report a maximal c-clique more than once in their recursive tree, no subgraph is reported more than once in the whole recursion tree of C_1 either.

Base case : A recursive call which is at the leaf of the recursion tree of the whole algorithm obviously does not report any c-clique more than once. A leaf call occurs when P is empty, and in such a case, it is evident from the algorithm that at most one c-clique can be reported.

Induction : Assume condition for the hypothesis is true. In this case, in order to be reported more than once, a c-clique has to be reported once by at least two calls out of $C_{11}, C_{12}, ..., C_{1k}$. Let $u_1, u_2, ..., u_k$ be the nodes from P added to R to make calls $C_{11}, C_{12}, ..., C_{1k}$, respectively. Consider any two calls C_{1i} and C_{1j} with $i < j$. Since the call C_{1i} is made with $R_{\text{new}} = R \cup u_i$, it is clear that all the c-cliques reported in the recursion subtree of C_{1i} will contain u_i. However, u_i will be in X when call C_{1j} is made, implying u_i is a forbidden node for C_{1j} and will never be added to R in its recursion subtree. All c-cliques reported in subtree of C_{1j} then will *not* contain u_i. Thus we have proved that no two C_{1i} and C_{1j} can report the same c-clique.

3. First, observe each recursive call made during the execution of the algorithm can be identified by the value of set R passed to it as parameter, i.e. each call has a unique value of R. Henceforth in this proof, by S-call, we shall mean the recursive call to which value of set R passed as parameter is $R = S$.

Now let M be any maximal c-clique. We define u_1 to be a node such that $\{u_1\}$-call is the first recursive call made by the initialization algorithm, with $u_1 \in M$. Trivially, such a u_1 and $S_1 = \{u_1\}$ must exist, since *all* nodes in the given graph are processed by the intialization algorithm. With this base case, we inductively define u_n and S_n as follows :

- u_n is the node such that $(S_{n-1} \cup u_n)$-call is the first recursive call made by S_{n-1}-call with $u_n \in M$.
- $S_n = S_{n-1} \cup u_n$.

Note that above definitions hold for $n \leq |\ M\ |$ only. To ensure that M is indeed reported by the algorithm, all we need to prove is that $u_{|M|}$ and $S_{|M|}$ can be found in the recursion tree formed by the execution of the algorithm. We prove this by induction.

Hypothesis : If u_{n-1} and S_{n-1} exist in the recursion tree, then u_n and S_n exist as well, for $n \leq |\ M\ |$.

Base case : u_1 and S_1 definitely exist, since all nodes are processed by the initializer algorithm.

Induction : Assume we have found u_{n-1}. Since $S_{n-1} \subsetneq M$ is a c-clique, there must be at least one node in M which is c^+d^*-connected to S_{n-1}. (If not, M is not a c-clique) According to the invariants listed in Fig. 6, all node must be in $P \cup X$ during S_{n-1}-call. We claim that no such node can be in X. As a proof by contradiction, assume that a node v which is c^+d^*-connected to S_{n-1} is in X. Note that v cannot have been added to X by the initialization algorithm, because if it was, then $v \in M$ was processed earlier than $u_1 \in M$ by the initialization algorithm, and this violates the

definition of u_1. Thus, let S_k-call be the parent recursive call in which v was added to X. Since v was in X when u_k was added to R, v must have been processed before u_k. This violates the definition of u_k.

To conclude, no node v which is c^+d^*-connected to S_{n-1} can be in X. Since at least one such v must exist, it must be in P. And hence at least one recursive call with such a node will be made, and by definition u_n will be the first such node, with $S_n = S_{n-1} \cup u_n$.

We have proved that for any M, $u_{|M|}$ and $S_{|M|}$ can be found in the recursion tree of the algorithm. During $S_{|M|}$-call, P and X shall be empty (otherwise M cannot be maximal). Hence $M = R = S_n$ shall be reported. □

Acknowledgment

C. Karande acknowledges the support of the INRIA internship program.

References

[1] F.Cazals, C.Karande, Reporting maximal cliques: new insights into an old problem, Tech. Report 5615, INRIA, 2005
[2] E.J. Gardiner, P. Willett, P.J. Artymiuk, Graph-theoretic techniques for macromolecular docking, J. Chem. Inf. Comput. Sci. 40 (2000).
[3] H.M. Grindley, P.J. Artymiuk, D.W. Rice, P. Willett, Identification of tertiary structure resemblance in proteins using a maximal common subgraph isomorphism algorithm, J. Mol. Biol. 229 (1993).
[4] I. Koch, Fundamental study: enumerating all connected maximal common subgraphs in two graphs, Theoret. Comput. Sci. 250 (1–2) (2001) 1–30.
[5] G. Levi, A note on the derivation of maximal common subgraphs of two directed or undirected graphs, Calcolo 9 (1972) 341–352.
[6] V. Nicholson, C.C. Tsai, M. Johnson, M. Naim., A subgraph isomorphism theorem for molecular graphs, in: Internet. Conf. on Graph Theory and Topology in Chemistry, 1987, pp. 226–230
[7] R. Samudrala, J. Moult, A graph-theoretic algorithm for comparative modelling of protein structure, J. Mol. Bio. 279 (1998).
[8] B.K. Stoichet, D.L. Bodian, I.D. Kuntz, Molecular docking using shape descriptors, J. Comput. Chem. 13 (3) (1992).
[9] H. Whitney, Congruent graphs and the connectivity of graphs, Amer. J. Math. 54 (1932) 150–168.