Finding Maximum Common Connected Subgraphs using clique detection or Constraint Satisfaction algorithms

Philippe VISMARA[1,2] and Benoît VALERY[1]

[1] LIRMM - 161 rue Ada - 34392 Montpellier Cedex 5 - France
[2] LASB - Montpellier SupAgro - 2 place Pierre Viala - 34060 Montpellier Cedex 1
`vismara@lirmm.fr` - `valery@lirmm.fr`

Summary. This paper investigates the problem of Maximum Common Connected Subgraph (MCCS) which is not necessarily an induced subgraph. This problem has so far been neglected by the literature which is mainly devoted to the MCIS problem. Two reductions of the MCCS problem to a MCCIS problem are explored: a classic method based on linegraphs and an original approach using subdivision graphs. Then we propose a method to solve MCCS that searchs for a maximum clique in a compatibility graph. To compare with backtrack approach we explore the applicability of Constraint Satisfaction framework to the MCCS problem for both reductions.

Key words: Maximum common subgraph; linegraph; subdivision graph, compatibility graph; constraints satisfaction algorithm; clique detection

1 Introduction

A classic method for comparing two graphs is to find the largest pattern between them. Most of the time, this question is interpreted as a *maximum common induced subgraph* (MCIS) problem. Nevertheless, some slightly different problems can be relevant in many areas. For instance, finding connected subgraphs can be preferred to compare molecules in the design of organic synthesis.

In this paper we investigate the problem of *Maximum Common Connected Subgraph* (MCCS) which is not necessarily an induced subgraph. This problem has so far been neglected by the literature which is mainly devoted to the MCIS problem.

The algorithms that solve MCIS are generally classified into two main categories: backtrack algorithms and methods that find a maximum clique in a *compatibility graph*. This latter approach is one of the most popular and is generally based on variants of the Bron and Kerbrosch's algorithm [3] that finds the maximal cliques of a graph. Koch [8] has proposed an extension of the method adapted to the MCCIS problem (connected MCIS). The non-clique based backtrack approach is symbolized by McGregor's algorithm [12]. This method has several similarities with the framework of Constraint Satisfaction Problems.

Common subgraph problems for chemical structures matching are explored in [14]. The MCIS problem is NP-hard except for almost trees of bounded degree [1]. As for the MCCIS problem, it is polynomial for partial k-tree [18].

Based on Withney's theorem [17] on *linegraphs*, a reduction of the MCS problem (neither induced nor connected) to a MCIS problem is often suggested (but never detailed) in the literature. In this paper we explore this reduction for MCCS on labeled graphs. We also investigate another reduction based on the *subdivision graph* notion. Then we study how to solve this problem using a clique-based algorithm for both reductions. In section 4 we explore the applicability of constraint satisfaction algorithms to the MCCS problem. For each approach we compare the efficiency of using linegraphs or subdivision graphs to transform the problem into a MCCIS problem. Experimental results are reported in section 5.

2 Preliminaries

We consider connected graphs with labeled nodes and edges. Formally, a graph is a 4-tuple, $G = (V, E, \mu, \nu)$, where V is the set of vertices, $E \subseteq V \times V$ is the set of edges, $\mu : V \to L_V$ is a function assigning to each vertex a label from the set of labels L_V and similarly $\nu : E \to L_E$ is the edge labeling function.

For any edge $e = xy$ we define $ends(e) = \{x, y\}$.

A graph $H = (V', E', \mu', \nu')$ is a *subgraph* of G iff $V' \subseteq V$, μ' and ν' are the restrictions of μ and ν respectively and $E' \subseteq E \cap (V' \times V')$. The graph H is an *induced subgraph* of G if $E' = E \cap (V' \times V')$.

Given two graphs G_1 and G_2, a Common Connected Subgraphs (CCS) of G_1 and G_2 is a connected graph H isomorphic to both subgraphs of G_1 and G_2.

A Maximum Common Connected Subgraphs (MCCS) is a Common Connected Subgraphs which size is maximum according to the number of edges. By analogy, we can define a Maximum Common Connected Induced Subgraph (MCCIS). Generally, MCCIS is maximum according to number of vertices. Figure 1 illustrate differences between MCS, MCCS, MCIS, and MCCIS.

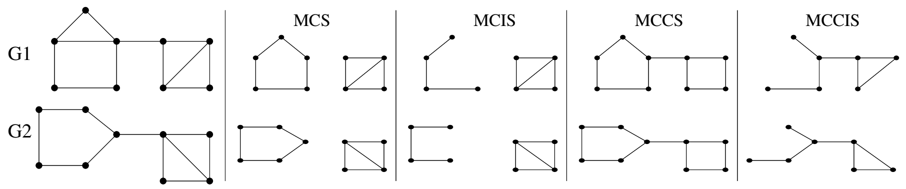

Fig. 1. Differences between MCS, MCCS, MCIS, and MCCIS

Linegraph

The linegraph $L(G)$ of a graph $G = (V, E, \mu, \nu)$ is a graph that has a vertex for each edge of G, and two vertices of L(G) are adjacent if they correspond to two edges of G with a common extremity.

According to Withney's theorem [17], two connected graphs with isomorphic linegraphs are isomorphic unless one is a triangle (K_3) and the other is a trinode

($K_{1,3}$) since both graphs have their linegraph equal to K_3. Hence, the MCCS problem between two unlabeled connected graphs can be solved as an MCCIS problem between their linegraphs. Checking for triangle / trinode exchange must be done only for solutions including less than 4 edges.

It is important to note that there is no such a direct equivalence for the MCS problem (not necessarily connected) because a MCIS between two linegraphs can have many connected components reduced to K_3. Since solutions with triangle / trinode exchanges can be larger than solutions without exchanges, the test for exchanges must be done during the search.

Now we consider the MCCS problem for labeled graphs. To insure the equivalence with MCCIS for labeled graphs, the corresponding linegraphs must be labeled on both nodes and edges (see figure 2). We define the labeling functions of the linegraph $L(G) = (E, \mathcal{E}, \mu_L, \nu_L)$ as follows: $\forall e \in E,$ where $e = xy,$ $\mu_L(e) = (\nu(e), \mu(x), \mu(y))$ and $\forall \alpha\beta \in \mathcal{E},$ $\nu_L(\alpha\beta) = \mu(ends(\alpha) \cap ends(\beta))$. Using this definition, Withney's theorem can be extended to labeled graphs. To demonstrate this result, one can adapt the proof presented in [6]. Given an isomorphism of $L(G_1)$ onto $L(G_2)$ preserving the labels, it is easy to derive an isomorphism of G_1 onto G_2 that preserves the labels.

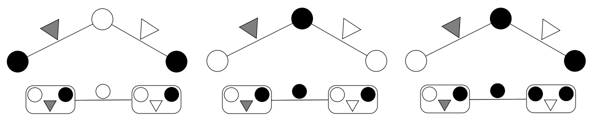

Fig. 2. Labeling linegraphs

Subdivision graph

The *subdivision graph* $S(G)$ is obtained from a graph $G = (V, E, \mu, \nu)$ by replacing each edge $e = xy$ by a new vertex e connected to both x and y.

Formally, $S(G) = (V \cup E, \mathcal{E}, \mu_s, f_0)$ where $\mathcal{E} = \{x\alpha \in V \times E \mid x \in ends(\alpha)\}$, f_0 is the zero function and $\forall x \in V, \mu_s(x) = \mu(x)$ and $\forall e \in E, \mu_s(e) = \nu(e)$.

Definition 1. *A balanced subgraph of a subdivision graph* $S(G) = (V \cup E, \mathcal{E}, \mu_s, f_0)$ *is a subgraph in which any vertex from* E *has an odd degree.*

Then, the MCCS problem between two graphs is clearly equivalent to find maximum common connected and balanced subgraphs between their subdivision graphs.

3 Clique detection

The detection of a MCIS between two graphs (G_a and G_b) can be solved by finding maximum clique in the *compatibility graph* (G_C). A *compatibility graph* of two graphs, also called modular graph, is a graph whose node set is $V_a \times V_b$. A node (x_i, x_j) in G_C represents a mapping between the vertex x_i from G_a and the vertex x_j from G_b. An edge between two nodes in G_C represents two compatible mapping. Then a clique in G_C of size k is a compatible mapping of k vertices in G_a with k vertices of G_b.

Reducing the MCIS problem to the maximum clique has been discovered independently by numerous authors such as Levi [11]. Clique detection is a common approach to compute MCIS. I. Koch [8] proposed a method to find MCCIS involving labels on edges of the compatibility graph.

In this section we present the ways to solve MCCS using clique detection algorithms on compatibility graphs constructed from linegraphs or subdivision graphs.

Clique detection based on linegraphs

According to section 2, we can reduce MCCS to MCCIS. The compatibility graph is constructed with $L(G_1)$ and $L(G_2)$ instead of G_1 and G_2.

Given two labeled linegraphs $L(G_1) = (E_1, \mathcal{E}_1, \mu_1, \nu_1)$ and $L(G_2) = (E_2, \mathcal{E}_2, \mu_2, \nu_2)$, the compatibility graph $G_C = (V_C, E_C, f_0, \nu_C)$ is defined as :

- $V_C = \{(x_1, x_a) \in E_1 \times E_2 \mid \mu_1(x_1) = \mu_2(x_a)\}$
- $E_C = \{(x_1, x_a)(x_2, x_b) \in V_C \times V_C\}$ such that :
  - $x_1 \neq x_2$ and $x_a \neq x_b$ and
  - $(x_1x_2 \in \mathcal{E}_1$ and $x_ax_b \in \mathcal{E}_2)$ or $(x_1x_2 \notin \mathcal{E}_1$ and $x_ax_b \notin \mathcal{E}_2)$
- $\nu_C : E_C \rightarrow \{\text{strong}, \text{weak}\}$ such that :
  - $\nu_C((x_1, x_a)(x_2, x_b)) = \begin{cases} \text{strong if } (x_1x_2 \in \mathcal{E}_1 \text{ and } x_ax_b \in \mathcal{E}_2) \\ \text{weak otherwise} \end{cases}$

A *clique* is a subset of nodes such that each pair of nodes is connected by an edge. An edge e is a *strong* edge iff $\nu_C(e) =$ strong. Two nodes a and b in G_C are said *strongly* (resp. *weakly*) *connected* iff ab is a strong edge (resp. weak). A clique is a *strong clique* if it contains a covering tree that consists of strong edges. Hence, the common subgraph corresponding to a strong clique is necessarily connected.

Once the compatibility graph is constructed (in $O(|E_1| \times |E_2|)$), a clique detection algorithm is used to find maximum cliques. The maximum clique problem is a classical problem in combinatorial optimization and has been widely studied [2]. The Bron and Kerbrosch's algorithm[3] is one of the first and most popular. This algorithm computes all *maximal cliques* but is often used for finding the *maximum clique*. The benefit of the backtracking method in [3] is that it avoids generating non-maximal cliques.

This algorithm has known several modifications such as Johnston's heuristic [7]. During the backtrack search, once the current clique K has been extended with z, K must be extended without z. [7] showed that the next node y to extend K can be taken within nodes disconnected to z since any maximal clique without z must include such a node.

Koch's algorithm [8] is based on [3] and computes all maximal strong cliques. The current clique K is extended with a node z strongly connected to K. Unfortunately, Johnston's heuristic[7] cannot be applied since nodes disconnected to z aren't necessarily strongly connected to K (see fig. 3). We propose to modify the heuristic such that the next node y (strongly connected to K) is added to K if either y is disconnected to z or y is strongly connected to a node t weakly connected to K and such that t is disconnected to z. In this way, the maximum strong clique found represents a MCCIS of $L(G_1)$ and $L(G_2)$ and therefore a MCCS of G_1 and G_2.

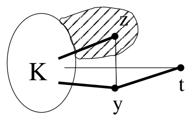

Fig. 3. Extending the current strong clique

Clique detection based on subdivision graphs

The compatibility graph can be created upon $S(G_1)$ and $S(G_2)$. As far as we know, subdivision graphs have not been used to reduce the MC(C)S problems to the MC(C)IS problems. Since a subdivision graph has two different kinds of nodes, the construction of the compatibility graph is more tricky.

The construction of the compatibility graph must ensure that a maximal clique corresponds to a balanced subgraph in $S(G_1)$ and $S(G_2)$.

Given two subdivision graphs $S(G_1) = (V_1 \cup E_1, \mathcal{E}_1, \mu_1, f_0)$ and $S(G_2) = (V_2 \cup E_2, \mathcal{E}_2, \mu_2, f_0)$, the compatibility graph $G_C = (V_C \cup E_C, \mathcal{E}_C, f_0, \nu_C)$ is defined as follows:

- $V_C = \{(x_1, x_a) \in V_1 \times V_2 \mid \mu_1(x_1) = \mu_2(x_a)\}$
  $E_C = \{(e_1, e_a) \in E_1 \times E_2 \mid \nu_1(e_1) = \nu_2(e_a) \text{ and } \mu_1(ends(e_1)) = \mu_2(ends(e_2))\}$
- $\mathcal{E}_C =$
  1. $\{(x_1, x_a)(e_1, e_a) \in V_C \times E_C \mid (x_1 \in ends(e_1) \text{ and } x_a \in ends(e_a)) \text{ or } (x_1 \notin ends(e_1) \text{ and } x_a \notin ends(e_a))\} \cup$
  2. $\{(x_1, x_a)(x_2, x_b) \in V_C \times V_C \mid x_1 \neq x_2 \text{ and } x_a \neq x_b\} \cup$
  3. $\{(e_1, e_a)(e_2, e_b) \in E_C \times E_C\}$ such that :
     - $e_1 \neq e_2$ and $e_a \neq e_b$ and
     - $|ends(e_1) \cap ends(e_2)| = |ends(e_b) \cap ends(e_b)|$
- $\nu_C : \mathcal{E_C} \rightarrow \{\text{strong}, \text{weak}\}$ such that :
  - $\nu_C((x_1, x_a)(e_1, e_a)) = \begin{cases} \text{strong if } x_1 \in ends(e_1) \text{ and } x_a \in ends(e_a) \\ \text{weak otherwise} \end{cases}$
  - $\nu_C((x_1, x_a)(x_2, x_b)) = \nu_C((e_1, e_a)(e_2, e_b)) = \text{weak}$

The time complexity of the construction is $O((|V_1| + |E_1|) \times (|V_2| + |E_2|))$.

Theorem 1. *A maximal strong clique K of the compatibility graph G_C defines a balanced connected subgraph in the subdivision graphs $S(G_1)$ and $S(G_2)$.*

Proof. The main result is to prove that for any $(a, b) \in E_C$, if $(a, b) \in K$ then K must include a couple of nodes in V_C that maps the ends of a and b. Let $(a, b) \in E_C \cap K$ such that $ends(a) = \{x, y\}$ and $ends(b) = \{i, j\}$. By construction (1) of $\mathcal{E}_C$, (a, b) is strongly connected to (x, i), (x, j), (y, j) and (y, i). By (2), (x, i) is adjacent to (y, j) and (x, j) is adjacent to (y, i). For any node $(z, k) \in V_C \cap K$ where $z \notin ends(a)$ or $k \notin ends(b)$, since (z, k) is adjacent to (a, b) we have by (1) (z, k) is adjacent to (x, i), (x, j), (y, j) and (y, i). Now let's assume that K includes another node $(c, d) \in E_C$. By (3), let $\alpha = |ends(a) \cap ends(c)| = |ends(b) \cap ends(d)|$. If $\alpha = 0$, then (c, d) is necessarily adjacent to (x, i), (x, j), (y, j) and (y, i). If $\alpha = 1$, without loss of generality, suppose that $ends(a) \cap ends(c) = \{x\}$ and $ends(b) \cap ends(d) = \{i\}$. Thus (c, d) is strongly connected to (x, i). Since $y \notin ends(c)$ and $j \notin ends(d)$ the nodes (y, j) and (c, d) must be connected. Hence, K must include (x, i) and (y, j) to be maximal.

4 Constraint satisfaction problems

In [12], McGregor presents one of the rare algorithms specially intended for MCS problems. Even so, this method has often been used to solve MCIS problems. The method is based on a backtrack algorithm but the way the method is implemented has some analogy with the general framework of Constraint Satisfaction Problems (CSP). Constraint satisfaction algorithms have been applied to several problems in Graph Theory [10, 5] but the MCCS problem has not yet been formulated as a constraint satisfaction problem. Since CSP research finding can benefit such an approach, we chose to study the applicability of backtrack methods to MCCS in the scope of CSP.

From induced subgraph problem to MCCIS

A constraint satisfaction problem (CSP) is described by a constraint network defined as a triple whose elements are a set of variables $X = \{x_1, x_2, ..., x_k\}$, a set of values for each variable, and a set of constraints among variables to specify which tuples of values can be assigned to tuples of variables. A solution of the CSP is an instantiation $\mathcal{I}$ of the variables that satisfies all the constraints.

For instance, a CSP for checking whether a graph G_1 is an induced subgraph of a graph G_2 could be defined as follows: (i) a variable X_i is defined for each vertex i of G_1; (ii) a variable X_i can be assigned to any vertex of G_2 whose label is the same as that of i; the set of values that X_i can take is called the domain of X_i and denoted by $D(X_i)$; (iii) a binary constraint $C(X_i, X_j)$ is defined between each pair of variables X_i, X_j to insure that the connectivity and the labeling are preserved by the mapping. A pair of values $(y_i, y_j) \in D(X_i) \times D(X_j)$ is allowed by the constraint if $ij \in E_1 \Leftrightarrow y_i y_j \in E_2$ and when $ij \in E_1, \nu_1(ij) = \nu_2(y_i y_j)$; (iv) a constraint of difference[15] is defined on variables to ensure that they all take different values. Any solution for this constraint network is a matching of G_1 to G_2.

Although the standard approach to solve a CSP is based on backtracking, the reader interested in algorithms to solve CSP should refer to the vast literature on this domain [16]. In this paper we focus on the classical constraint network framework which is oriented towards the satisfaction of all constraints. This framework is widely used and has been implemented in many constraint programming toolkits as JChoco [9] a Java library for constraint programming. In the last decade, several extensions of the classical CSP framework have been proposed. Some of them, like *soft constraints*, could be interesting to solve the MCCS problem. But most of the CSP solvers do not implement these extensions. Hence, they have not been studied yet in the context of the present work.

In the previous example we have defined a constraint network to solve the induced subgraph isomorphism problem. Representing an MCIS problem should differ in the way that some vertices of graph G_1 are not mapped to any vertex of G_2. Hence, the corresponding variables of the CSP cannot be assigned to values in X_2. A usual solution in such a case is to add an extra value (we denote $\star$) to the domain of the variables. Note that the constraint of difference must be weakened since many variables can be assigned to the $\star$ value. Then, for any solution of the CSP, the common induced subgraph will correspond to the variables assigned to values in X_2 only. The size of the common induced subgraph is the number of variables whose value differs from $\star$. Solving the MCIS problem is then equivalent to find a solution of the constraint network that minimize the number of $\star$ values.

To solve the MCCIS problem we add a new global constraint to the previous CSP. This *connectivity constraint* checks the connectivity of all the vertices whose corresponding variable is not assigned to $\star$.

In the following sections we detail the constraint networks to solve the MCCS problem using linegraph or subdivision graph respectively.

A constraint network based on linegraphs

Given two linegraphs $L(G_1) = (E_1, \mathcal{E}_1, \mu_1, \nu_1)$ and $L(G_2) = (E_2, \mathcal{E}_2, \mu_2, \nu_2)$, we propose to define a network constraint as follows:

- a set of variables $X = \{X_i \mid i \in E_1\}$
- a domain for each variable: $\forall i \in E_1,\ D(X_i) = \{y \in E_2 \mid \mu_2(y) = \mu_1(i)\} \cup \{\star\}$
- a binary constraint $C(X_i, X_j)$ between each pair of variables that allows the set of couples: $\{(k,l) \mid k,l \in E_2$ and $k \neq l$ and $(k,l) \in \mathcal{E}_2 \Leftrightarrow (i,j) \in \mathcal{E}_1\}$
  $\cup \{(t,\star),(\star,t) \mid t \in E_2\} \cup \{(\star,\star)\}$
- a global *constraint of connectivity* on X to insure that the subgraph induced by $\{i \in E_1 \mid \mathcal{I}(X_i) \neq \star\}$ is connected, where $\mathcal{I}$ is an instantiation of the variables.

The main difficulty lies in the implementation of the *constraint of connectivity*. We maintain two sets during the search. *CComp* is the set of variables already instancied to a non-$\star$ value and such that the subgraph $\{i \in E_1 \mid X_i \in CComp\}$ is connected. The set *Candidates* includes uninstanciated variables connected to at least one variable in *CComp*. Only variables in *Candidates* can be assigned to non-$\star$ values. The sets are updated after each assignment.

Finally, to minimize the number of $\star$ values a new variable $X_{\#\star}$ is usually added to the constraint network that counts the number of variables assigned to this value. Hence, $D(X_{\#\star}) = \{0 \ldots |E_1|\}$ and a global constraint is defined on $\{X_i\}_{i\in E_1} \cup \{X_{\#\star}\}$ to ensure that $\mathcal{I}(X_{\#\star}) = |\{i \in E_1 \mid \mathcal{I}(X_i) = \star\}|$

A constraint network based on subdivision graphs

The constraint network based on subdivision graphs is quite similar to that defined in the previous section. Given two graphs $G_1 = (V_1, E_1, \mu_1, \nu_1)$ and $G_2 = (V_2, E_2, \mu_2, \nu_2)$ and their subdivision graphs $S(G_1) = (V_1 \cup E_1, \mathcal{E}_1, \mu_{s1}, f_0)$ and $S(G_2) = (V_2 \cup E_2, \mathcal{E}_2, \mu_{s2}, f_0)$, we propose to define the network constraint as follows:

1. a set of variables $X = \{X_i \mid i \in V_1 \cup E_1\}$
2. $\forall i \in V_1,\ D(X_i) = \{y \in V_2 \mid \mu_2(y) = \mu_1(i)\} \cup \{\star\}$ and
   $\forall j \in E_1,\ D(X_j) = \{y \in E_2 \mid \nu_2(y) = \nu_1(j)\} \cup \{\star\}$
3. a binary constraint $C(X_i, X_j)$ for each couple $i, j \in V_1 \times E_1$ that allows the set $\{(k,l) \in V_2 \times E_2 \mid (k,l) \in \mathcal{E}_2 \Leftrightarrow (i,j) \in \mathcal{E}_1\} \cup \{(t,\star), \mid t \in V_2\} \cup \{(\star,\star)\}$
4. a binary constraint between each pair of variables in $\{X_i\}_{i\in V_1}$ (resp. $\{X_j\}_{j\in E_1}$) that forbids equals values except of $\star$.
5. a global constraint of connectivity on X to insure that the subgraph induced by $\{i \in E_1 \mid \mathcal{I}(X_i) \neq \star\}$ is connected, where $\mathcal{I}$ is an instanciation of the variables.

The main difference with the CSP based on linegraphs lies in the partition of the set of variables. By the binary constraint (3) between a "vertex variable" and an "edge variable" we can easily prove that $\forall j \in E_1, \mathcal{I}(X_j) \neq \star \Rightarrow \forall i \in ends(j), \mathcal{I}(X_i) \neq \star$

Hence any solution of the CSP corresponds to a balanced subgraph of the subdivision graphs.

5 Experimental results

We have implemented our constraint networks with the Java constraint programming library JChoco [9]. The set *CComp* and *Candidates* for the connectivity constraint are handled with backtrackable structures provided by JChoco.

The Bron and Kerbrosch's algorithm [3] for clique detection has been implemented in Java. We modified the program with Koch's work [8] to find only connected solutions. Then we adapted Johnston's heuristic [7] for improving performance.

Our database consists of three sets of graphs. The first one consists of 30 undirected connected graphs without label and randomly generated. Each has at least 10 nodes and at most 20 nodes. The second set of graphs contains 30 molecules with a size between 6 and 62 atoms. The last set of graphs are molecules taken from 5487 chemical reactions. In this set, we only compare the reactant graphs with the product graphs without labels in the same reaction. A timeout was set to 10 minutes for each test.

	# of couples of graphs	average $(m_1 * m_2)$	average $(n_1 \times n_2) + (m_1 + \times m_2)$
molecules	465	475	1019
random graphs	465	835	1816
chemical reactions	8437	169	331

Table 1. Description of the database. The third column (resp. fourth) represents the average size of the compatibility graph on linegraphs (resp. subdivisions graphs)

	CSP		Clique detection	
	Linegraph	Subdivision	Linegraph	Subdivision
molecules without labels	73,99%	69,04%	65,6%	55,4%
molecules	98.15%	94.45%	99.2%	97.09%
random graphs	73.6%	71.7%	65.5%	48.5%
chemical reactions	99.68%	99,58%	99,79%	98,62%

Table 2. Percent of solved problems on the different graphs sets within the time limit.

The first statement can be done about the comparison between subdivision and the linegraph method in either CSP or clique-based approach. The subdivision method are almost always slower than the linegraph method. Subdivision graphs increase the number of nodes of a graph. Since the complexity of MCCS depends of the size of the data, the size of subdivision graphs probably slows the procedure.

For small graphs or labbeled graphs, both CSP and clique approachs solve the same number of problems within the time limit. The difference is more important for larger graphs.

One explanation could be that the library used to implement the CSP algorithms is quite complex and not very efficient for small problems. Conversely, the clique detection algorithms are easier to implement but they do not benefit of the CSP heuristics for large problems. As long as the compatibility graph has a reasonable size (see column 3 and 4 from Table 1), the maximum clique can be found within the time limit. When the size of the compatibilty graph arises, finding the maximum clique is harder and the algorithms timeout.

Even with a preliminary benchmark and a different problem, we have a similar conclusion than [4] that deals with MCIS on directed graphs: we cannot point clearly a faster method in general. Meanwhile, its seems that the size of the compatibility graph could be a threshold where the clique detection algorithms become less effective than constraints satisfaction algorithms.

6 Conclusion

We have presented two methods to reduce the MCCS problem to an MCCIS problem. The first one is an adaptation of the reduction based on linegraphs for the "induced" versions of the problems. The second method is a new approach that involves subdivision graphs. As far as we know, this reduction have not been yet applied even for the MCIS problem.

These methods have been formalized in the general scope of labeled graphs.

We have adapted Koch's algorithm [8] that computes an MCCIS by searching a clique in a labeled compatibility graph. We proposed an heuristic for the choice of the next vertex to add to the strong clique. We have extended the model of compatibility graph in order to solve the MCCS problem for both linegraph and subdivision graph reductions.

To investigate backtrack algorithms as McGregor's algorithm[12] we have chosen the general framework of Constraint Satisfaction Problems. We have studied the applicability of constraint satisfaction techniques for linegraphs and subdivision graphs reductions. The constraints we propose are quite simple and may be improved. We have implemented both constraint networks using the JChoco[9], a open source constraint programming toolkit, using the Java programming language.

The four methods we have investigated to solve the MCCS have been implemented in Java. We have experimented these algorithms on molecular labeled graphs and unlabeled random graphs. The first results show that a linegraph approach is generally faster than methods using subdivision graphs. One explanation could be that subdivision graphs include more nodes than the corresponding linegraphs. This drawback could be reduced with heuristics that exploit the specific structure of subdivision graphs. For small or very labelled graphs there is no significant difference between CSP and clique approachs. For more complex graphs, it seems that the clique detection algorithms become less effective than constraint algorithms.

Since the constraint networks we have proposed are based on quite simple constraints, it would be interesting to optimize them. Another solution would be to use extensions of the classical constraint network framework as *soft constraints*[13].

Nevertheless, the MCCS problem itself has many variants for real word problems. For instance we can use different criteria to calculate the size of a common subgraph (number of nodes and edges, ...). It should be interesting to compare the different methods according to their adaptability to these variations.

References

1. T. Akutsu. A polynomial time algorithm for finding a largest common subgraph of almost trees of bounded degree. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E76-A(9), 1993.
2. I. M. Bomze, M. Budinich, P. M. Pardalos, and M. Pelillo. The maximum clique problem. In D.-Z. Du and P. M. Pardalos, editors, *Handbook of Combinatorial Optimization (Supplement Volume A)*, pages 1–74. Kluwer Academic, 1999.
3. C. Bron and J. Kerbosch. Finding all cliques of an undirected graph. *Communication of the ACM*, 16(9):575–579, 1973.
4. D. Conte, P. Foggia, C. Sansone, and M. Vento. Thirty years of graph matching in pattern recognition. *IJPRAI*, 18(3):265–298, 2004.
5. G. Dooms, Y. Deville, and P. Dupont. Cp(graph): Introducing a graph computation domain in constraint programming. In *Proc CP 2005*. Springer Verlag, 2005.
6. F. Harary. *Graph Theory*. Addison-Wesley, 1969.
7. H.C. Johnston. Cliques of a graph-variations on the bron-kerbosch algorithm. *International Journal of Computer and Information Sciences*, 5(3):209–238, 1976.
8. I. Koch. Enumerating all connected maximal common subgraphs in two graphs. *Theoretical Computer Science*, 250:1–30, 2001.
9. F. Laburthe and N. Jussien. Jchoco: A java library for constraint satisfaction problems. http://choco.sourceforge.net.
10. J. Larossa and G. Valiente. Constraint satisfaction algorithms for graph pattern matching. *Math. Struct. Comput. Sci.*, 12(4):403–422, 2002.
11. G. Levi. A note on the derivation of maximal common subgraphs of two directed or undirected graphs. *Calcolo*, 9(4):341–352, 1972.
12. J. J. McGregor. Backtrack search algorithms and the maximal common subgraph problem. *Software Practice and Experience*, 12:23–34, 1982.
13. P. Meseguer, F. Rossi, and T. Schiex. Soft constraints. In Rossi et al. [16], pages 281–328.
14. J. W. Raymond and P. Willett. Maximum common subgraph isomorphism algorithms for the matching of chemical structures. *Journal of Computer-Aided Molecular Design*, 16(7):521–533, 2002.
15. J.-C. Régin. a filtering algorithm for constraints of difference in CSPs. In *AAAI-94, Proceedings of the National Conference on Artificial Intelligence*, pages 362–367, Seattle, Washington, 1994.
16. F. Rossi, P. van Beek, and T. Walsh, editors. *Handbook of Constraint Programming*. Elsevier, 2006.
17. H. Whitney. Congruent graphs and the connectivity of graphs. *Am. J. Math.*, 54:150–168, 1932.
18. A. Yamaguchi and H. Mamitsuka K. F. Aoki. Finding the maximum common subgraph of a partial k-tree and a graph with a polynomially bounded number of spanning trees. *Information Processing Letters*, 92(2):57–63, 2004.