A New Approach and Faster Exact Methods for the Maximum Common Subgraph Problem*

W. Henry Suters[†§], Faisal N. Abu-Khzam[‡§], Yun Zhang[¶],
Christopher T. Symons[‖], Nagiza F. Samatova[‖**] and Michael A. Langston[¶**]

Abstract

The Maximum Common Subgraph (MCS) problem appears in many guises and in a wide variety of applications. The usual goal is to take as inputs two graphs, of order m and n, respectively, and find the largest induced subgraph contained in both of them. MCS is frequently solved by reduction to the problem of finding a maximum clique in the order mn association graph, which is a particular form of product graph built from the inputs. In this paper a new algorithm, termed "clique branching," is proposed that exploits a special structure inherent in the association graph. This structure contains a large number of naturally-ordered cliques that are present in the association graph's complement. A detailed analysis shows that the proposed algorithm requires $O((m+1)^n)$ time, which is a superior worst-case bound to those known for previously-analyzed algorithms in the setting of the MCS problem.

1 Introduction

A popular metric for the similarity of two graphs is the size of their Maximum Common Subgraph (MCS), which is most frequently defined as the largest graph isomorphic to some induced subgraph in each of them. Deciding MCS is $\mathcal{NP}$-complete. It has been studied in bioinformatics [21], chemistry [16, 19], pattern recognition [7, 15], and an assortment of other application areas. A vast literature exists for approximating the size of an MCS. Notably, it is $\mathcal{NP}$-hard to guarantee solutions even within $|V|^{\epsilon}$, where $\epsilon > 0$ and $|V|$ is the size of the MCS [13]. Here we focus on exact MCS algorithms. These can be roughly classified into three main categories: clique-based methods, non-clique-based backtracking strategies, and various other techniques.

Clique-based methods are the most widely used in the literature. These depend on finding a maximum clique in the association graph (see, for example, [5]). We will define this and other terms in the sequel. Let us just say for now that the association graph is a particular form of product graph built from the two original input graphs. Many clique-based algorithms employ maximal clique enumeration procedures [6, 19], and so are not particularly well suited for maximum clique finding. The general purpose maximum clique

*Research sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory, managed by UT-Battelle, LLC, for the U. S. Department of Energy under Contract DE–AC05–00OR22725, by the U.S. National Science Foundation under grant CCR–0311500, by the Office of Naval Research under grant N00014–01–1–0608, and by the U.S. Department of Energy's Genomes to Life program under the ORNL-PNNL project "Exploratory Data Intensive Computing for Complex Biological Systems."

[†]Department of Mathematics and Computer Science, Carson-Newman College, CN Box 71958, Jefferson City, TN 37760, USA

[‡]Division of Computer Science and Mathematics, Lebanese American University, Beirut, Lebanon

[§]These two authors contributed equally to this work.

[¶]Department of Computer Science, University of Tennessee, Knoxville, TN 37996–3450, USA

[‖]Computer Science and Mathematics Division, Oak Ridge National Laboratory, P.O. Box 2008, Oak Ridge, TN 37831–6367, USA

[**]**Communicating authors: samatovan@ornl.gov, langston@cs.utk.edu**

algorithm of [20] has a time complexity of $O(1.19^{mn})$, where m and n denote the respective sizes of the input graphs. An example of a non-clique-based backtracking strategy is that of [17], which was developed over two decades ago but rarely used today. In fact it is known to perform well only on graphs of small size [10]. An improved backtracking algorithm has recently been proposed with time complexity $O(m^{n+1}n)$ [14]. Other techniques include algorithms designed for special classes of inputs. One example of this is the dynamic programming approach of [4], which has been proposed for "almost trees of bounded degree." Other methods rely on a set of known graphs against which matching is to be performed [8, 18]. For more information we refer the interested reader to [9].

In this paper, we present and analyze a new algorithm for the exact MCS problem. To solve clique on the association graph, we actually exploit a special clique structure that we have identified as inherent in the complement of the association graph. There we employ an efficient tree search technique designed to branch on this structure. The resultant "clique branch" algorithm is used to solve vertex cover in the complement and hence clique in the association graph itself. It requires $O((m+1)^n)$ time, which (with inputs of course set to ensure that m is at least as large as n) is a better asymptotic worst-case bound than the $O(m^{n+1}n)$ limit of [14] and that known for other previously-analyzed algorithms in the setting of the MCS problem.

2 The Association Graph and its Properties

2.1 Definitions

In this section we provide some formal definitions related to the MCS problem. All graphs are assumed to be simple, finite, and undirected. Two graphs G_1 and G_2 are said to be *isomorphic* if there is a one-to-one correspondence between their vertex sets that preserves the adjacency of vertices. A graph M is a *maximum common subgraph* (henceforth MCS) of graphs G_1 and G_2 if M consists of the largest number of vertices and is isomorphic to induced subgraphs of G_1 and G_2. Such definition of the MCS is often referred to as the *maximum common induced subgraph* ($MCIS$). If M consists of the largest number of edges then the MCS is referred as the *maximum common edge subgraph* ($MCES$) [17]. Throughput this paper, the MCS graph denotes the $MCIS$ graph. Note that the MCS is neither unique nor connected.

Given two undirected graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, the *association graph* $G = (V, E)$ is an undirected graph defined on the vertex set $V = V_1 \times V_2$ with two vertices (u_1, v_1) and (u_2, v_2) being adjacent whenever:

$u_1 \neq u_2$ and $v_1 \neq v_2$, and

either $(u_1, u_2) \in E_1$ and $(v_1, v_2) \in E_2$ or $(u_1, u_2) \notin E_1$ and $(v_1, v_2) \notin E_2$.

From this point on, we denote by G_1 and G_2 the two input graphs of MCS. Moreover, we shall assume that $n = |G_1| \leq |G_2| = m$.

2.2 Preliminaries

From the definition of the association graph, we know that two vertices (u_1, v_1) and (u_2, v_2) are adjacent in G only if $u_1 \neq u_2$ and $v_1 \neq v_2$. In other words, any two vertices (u_i, v_j), (u_i, v_k) are not adjacent for $\forall u_i \in V_1$ and $v_j, v_k \in V_2$, and any two vertices (u_i, v_j), (u_k, v_j) are not adjacent for $\forall v_j \in V_2$ and $u_i, u_k \in V_1$. Therefore, we have the following theorem.

Theorem 1 *If two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are used to create an association graph then, for any $u \in V_1$ and any $v \in V_2$, each of the sets $\{u\} \times V_2$ and $V_1 \times \{v\}$ forms a clique in the complement of the association graph.*

$V_1 \times V_2$	v_1	v_2	$\cdots$	v_m
u_1	(u_1, v_1)	(u_1, v_2)	$\cdots$	(u_1, v_m)
u_2	(u_2, v_1)	(u_2, v_2)	$\cdots$	(u_2, v_m)
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
u_n	(u_n, v_1)	(u_n, v_2)	$\cdots$	(u_n, v_m)

Proof. Let $|V_1| = n$ and $|V_2| = m$. If we view $V = V_1 \times V_2$ as a table as follows. Then, because of the first two conditions of the definition of the association graph, for $\forall u_i \in V_1$, the set of vertices $(u_i, v_1), \ldots, (u_i, v_m)$ in G forms an independent set, and for $\forall v_j \in V_2$, the set of vertices $(u_1, v_j), \ldots, (u_n, v_j)$ in G also forms an independent set. Thus, in the complement graph, each row and column of the table will form a clique. ■

Let k be the size of a common subgraph of G_1 and G_2. We can restrict our search for a minimum vertex cover to covers with sizes that are at least $nm - n$ and at most $nm - k$. Because the maximum value of k is unknown upfront, it could be the size of any common subgraph that is trivially found (e.g. the graph with the minimum of the sizes of two independent sets of G_1 and G_2 or induced paths of equal size).

Theorem 2 *Let k be the size of any common subgraph of G_1 and G_2, and let G be their association graph. Then any vertex cover of the complement of G must contain at least $nm - \min\{n, m\}$ and at most $nm - k$ vertices.*

Proof. Let H_1 and H_2 be subgraphs of G_1 and G_2 that are isomorphic. If $|H_1| = |H_2| = k$, then the ordered pairs of the set $\{(u, v) : u \in H_1 \text{ and } v \in H_2\}$ form a clique of size k in G, the association graph of G_1 and G_2. So the complement of G has an independent set of size k. This proves the claim concerning the $mn - k$ upper bound.

For the lower bound, we know the size of the MCS is bounded above by $\min\{n, m\}$, which implies the size of the maximum clique of the association graph G does not exceed $\min\{n, m\}$. This implies any vertex cover of the complement of G has at least $(nm - \min\{n, m\})$ vertices. ■

3 The Structural Decomposition Algorithm

In order to motivate our approach it is important to observe that, for a clique of size k, any vertex cover must have at least $k - 1$ vertices. Moreover, if any one of the vertices is excluded from the vertex cover, all of its neighbors must be included. In the complement of the association graph, each vertex (u, v) is involved in at least two cliques that only overlap at this vertex. The two cliques correspond to the row and column that intersect at (u, v) in aforementioned table.

We shall refer to these two cliques by the row-clique and the column-clique of (u, v). If (u, v) is to be excluded from the cover, then all vertices in both of these cliques must be included in the cover. Moreover, any other vertices that are adjacent to this vertex will also be included in the cover. This means the size of the problem is greatly reduced when we decide to exclude a vertex from the vertex cover.

We show that our vertex cover branching algorithm can do at least as well as any other known algorithm for MCS. For this purpose, we present vertex cover branching in a way that makes use of the presence of row-cliques and column-cliques in the complement of the association graph.

The idea is that when we attempt to find a vertex cover, we can select at most one vertex from each clique to be excluded from the cover. Thus, in a row-clique of size m, there are $m + 1$ possible choices for any vertex cover; m choices each exclude one of the vertices while the remaining choice includes all the

vertices. This forms the basis for our clique branching algorithm below.

algorithm *CliqueBranch* (G)

Input: the complement of the association graph G created from two graphs G_1 and G_2 of sizes n and m respectively
Output: a minimum vertex cover of the association graph's complement

```
begin
    MinimumCover = G
    CurrentCover= G
    NumberExcluded = 0
    ExcludeColumns = ∅
    i = 1
    Branch(i)
    output MinimumCover
end

function Branch(i)
    begin
        if i > n then
            if |CurrentCover| < |MinimumCover| then
                MinimumCover = CurrentCover
        else
            NumberExcluded = NumberExcluded + 1
            loop over all v_j ∈ V_2 where j ∉ ExcludedColumns
                if (u_i, v_j) has a neighbor (u_k, v_l) ∉ CurrentCover then
                    do nothing
                else
                    add j to ExcludedColumns
                    remove (u_i, v_j) from CurrentCover
                    Branch (i + 1)
                    remove j from ExcludedColumns
                    add (u_i, v_j) to CurrentCover
            NumberExcluded = NumberExcluded - 1
            Branch (i + 1)
    end
```

Theorem 3 *The clique branching algorithm, CliqueBranch (G), produces a minimum vertex cover of the complement of the association graph G.*

Proof. We walk through the rows of the aforementioned table, branching at each row. Since a row represents a clique, we could select either to exclude exactly one of its vertices from the vertex cover or to include all of them. We cannot, however, choose a vertex that belongs to a column from which a vertex was selected at a previous branching since each column also represents a clique. The clique branching algorithm examines all possible vertex covers that satisfy these conditions and selects the one with minimum size, thus it produces a minimum vertex cover. ■

In order to get a rough estimate of the complexity of the algorithm, consider the vertex table established in Theorem 1. Branching on the cliques represented by each row, we have two possibilities. If no vertex is excluded from the cover then we have identified m vertices as belonging to the vertex cover. If we select to exclude a particular vertex (u, v), then this vertex will have at least $m + n - 2$ neighbors that must be

included in the cover; there are $m-1$ other vertices in the same row-clique and $n-1$ other vertices in the same column-clique of (u, v).

We can branch recursively until we arrive at the final row of the matrix. Since there will be n levels of branching, each with at most $m+1$ possible paths, this produces an algorithm that is at most $O((m+1)^n)$. This bound is not tight, since once we select a vertex to exclude, its neighbors cannot be selected for a similar role in a later branching. Thus, for a later branchings, there will be fewer than $m+1$ choices.

Moreover, the number of neighbors of a vertex is more than the number of vertices in its corresponding row and column cliques. This is guaranteed by the following lemma. A formal proof of the computational complexity will be shown in Section 4.

Lemma 1 *If the neighborhood of a vertex (u, v) is confined to its row-clique and column-clique, then either u and v are both isolated or both connected to all the vertices in each of G_1 and G_2, respectively.*

Proof. Assume the neighborhood condition as stated (and $u' \neq u$ and $v' \neq v$). Then (u, v) is connected to all vertices (u', v') in the association graph of G_1 and G_2. Assume u is neither isolated nor connected to all vertices. Then we can find u' and u" such that (u, u') is an edge of G_1 while $(u, u")$ is not an edge. If v has an edge (v, v') (or a non-edge $(v, v")$) then (u, v) is not joined to $(u", v')$ (or (u, v) is not joined to $(u', v")$). This is a contradiction. So our assumption about u (neither isolated nor connected to all vertices) is wrong. The same argument proves that if v is not isolated, then it's connected to all vertices of G_2. ■

So if such a pair of vertices (u, v) is found in $G_1 \times G_2$, they are associated together as part of any common subgraph. Moreover, we could detect their presence in the original graphs without searching the association graph. So, as a pre-processing rule applied prior to any branching step, our algorithm detects the presence of such pairs (u, v) and reduces the problem size by not including them in the vertex cover.

Figure 1 shows an example of the clique branching algorithm applied to two graphs G_1 and G_2, where $|V_1| = 4$ and $|V_2| = 3$. The complement of the association graph has 12 vertices and can be listed as a 4×3 array. Starting from the first row, there are $3 + 1 = 4$ choices (branches). Three of these select a single vertex to be excluded from the cover. The fourth branch does not exclude any vertices. For the first three branches (selecting one), the remaining graphs are much smaller than the original complement graph. This is especially true for the latter two branches. Taking the second branch as an example, excluding vertex af from the cover forces vertices bf, cf, df, in its column, ae, ag, in its row, and bg, cg, dg, in its neighborhood, into the cover. This leaves a graph containing only the three vertices be, ce, de. The next step along this branch will have only two choices: excluding or including vertex be. The fourth branch, which excludes no vertices from the first row, still reduces the graph by including all vertices in the first row in the cover, resulting in a 3×3 array remaining. The next step of this worst-case branch would still have $3 + 1 = 4$ choices.

4 Complexity Analysis

The following theorem is used to determine the complexity of the clique branching algorithm.

Theorem 4 *The complexity of performing the clique branching algorithm on the association graph constructed from two graphs of sizes n and m is*

$$\sum_{i=0}^{n} \frac{m!}{(m-n+i)!} \left(\begin{array}{c} n \\ i \end{array} \right). \tag{1}$$

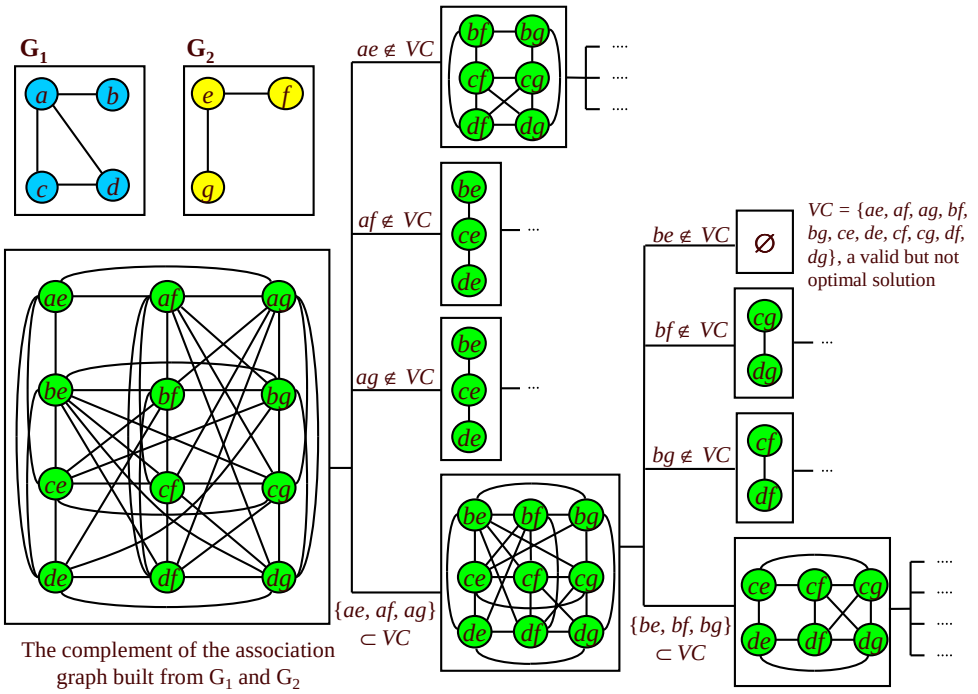

Figure 1: An example of the clique branching algorithm.

Proof. Let $R(m,n)$ be the computational complexity of processing an $n \times m$ array of vertices. The first branching will eliminate one row of the array, resulting in $(n-1)$ rows remaining. Also, there are $(m+1)$ possible paths at this branching. There are m paths, each selects a vertex to exclude from the vertex cover, which also eliminates a column from future consideration, resulting in $(m-1)$ remaining columns. The final path does not exclude any vertices from the cover, resulting in m remaining columns. These observations result in the following recurrence relation.

$$R(m,n) = mR(m-1,n-1) + R(m,n-1) \tag{2}$$

with initial condition $R(j,0) = 1$, for $0 \leq j \leq m$.

We next need to demonstrate that the formula

$$R(m,n) = \sum_{i=0}^{k} \frac{m!}{(m-k+i)!} \left(\begin{array}{c} k \\ i \end{array} \right) R(m-k+i, n-k), \quad \text{for } 0 \leq k \leq n \tag{3}$$

satisfies the recurrence relation.

We do inductively. First, if $k = 0$ observe that

$$\sum_{i=0}^{k} \frac{m!}{(m-k+i)!} \left(\begin{array}{c} k \\ i \end{array} \right) R(m-k+i, n-k) = R(m,n).$$

Next, we need to show that

$$R(m,n) = \sum_{i=0}^{k} \frac{m!}{(m-k+i)!} \left(\begin{array}{c} k \\ i \end{array} \right) R(m-k+i, n-k) \tag{4}$$

implies

$$R(m,n) = \sum_{i=0}^{k+1} \frac{m!}{(m-(k+1)+i)!} \left(\begin{array}{c} k+1 \\ i \end{array} \right) R(m-(k+1)+i, n-(k+1)). \tag{5}$$

Notice that the recurrence relation (Equation 2) implies that

$$R(m-k+i, n-k) = (m-k+i)R(m-k+i-1, n-k-1) + R(m-k+i, n-k-1). \tag{6}$$

Combining Equation 6 with the induction hypothesis (Equation 4), we see that

$$\begin{aligned} R(m,n) &= \sum_{i=0}^{k} \frac{m!}{(m-k+i)!} \left(\begin{array}{c} k \\ i \end{array} \right) [(m-k+i)R(m-k+i-1, n-k-1) \\ &\qquad + R(m-k+i, n-k-1)] \\ &= \sum_{i=0}^{k} \frac{m!}{(m-k+i-1)!} \left(\begin{array}{c} k \\ i \end{array} \right) R(m-k+i-1, n-k-1) \\ &\quad + \sum_{i=0}^{k} \frac{m!}{(m-k+i)!} \left(\begin{array}{c} k \\ i \end{array} \right) R(m-k+i, n-k-1). \end{aligned}$$

As long as $k+1 \leq n$ we define $l = k+1$, re-index the second sum, pull the first term from the first sum and the last term from the second sum to get

$$\begin{aligned} R(m,n) &= \frac{m!}{(m-l)!} R(m-l, n-l) \\ &\quad + \sum_{i=1}^{k-1} \frac{m!}{(m-l+i)!} \left[\left(\begin{array}{c} l-1 \\ i \end{array} \right) + \left(\begin{array}{c} l-1 \\ i-1 \end{array} \right) \right] R(m-l+i, n-l) + R(m, n-l). \end{aligned}$$

Since $\left(\begin{array}{c} l-1 \\ i \end{array} \right) + \left(\begin{array}{c} l-1 \\ i-1 \end{array} \right) = \left(\begin{array}{c} l \\ i \end{array} \right)$ we can include the first and last terms to show

$$R(m,n) = \sum_{i=0}^{l} \frac{m!}{(m-l+i)!} \left(\begin{array}{c} l \\ i \end{array} \right) R(m-l+i, n-l),$$

which is equivalent to Equation 5. This completes the inductive argument.

We now let $k = n$ to show that

$$R(m,n) = \sum_{i=0}^{n} \frac{m!}{(m-n+i)!} \left(\begin{array}{c} n \\ i \end{array} \right) R(m-n+i, 0).$$

Using the initial condition, this shows that

$$R(m,n) = \sum_{i=0}^{n} \frac{m!}{(m-n+i)!} \left(\begin{array}{c} n \\ i \end{array} \right). \blacksquare$$

To help interpret this result, notice that

$$(m+1)^n = \sum_{i=0}^{n} m^{n-i} \left(\begin{array}{c} n \\ i \end{array} \right).$$

Also notice that in general $\frac{m!}{(m-n+i)!}$ is much smaller than m^{n-i}. Finally notice that this is a worst case calculation. In the more general case, there will be fewer than $m+1$ branches at many levels since some potential branches would result in more neighbors being excluded from the vertex cover. This is guaranteed by Lemma 1. Thus, in all cases, the performance will be much better than $(m+1)^n$.

5 Remarks

Association graphs are used to reduce MCS to the maximum clique problem. The main contribution of this paper has been a careful analysis of a clique-branching algorithm designed to exploit the structure implicit in the complement of the association graph. It is not clear, currently, whether direct maximum clique algorithms that operate on the association graph itself can do significantly better. We do know, however, that any such algorithm should not do worse than the time bound we derive in Equation 1, at least as long as it performs a standard form of vertex branching. To illustrate, consider recursive backtracking algorithms such as that of [20]. The following branching method is employed: at each node of the search tree, a highest-degree vertex v is selected and two possible choices are explored: either v is in the clique or it is not. When v is added to the clique, all vertices not adjacent to it are eliminated. Thus, the problem size may be reduced considerably in this case. When v is not added to the clique, v alone can of course be deleted.

In the association graph, a vertex $(u, v) \in V$ where $u \in V_1, v \in V_2$ is a member of an independent set containing all vertices whose first component is u. It is also a member of another independent set consisting of all vertices whose second component is v. Therefore, when using a maximum-clique algorithm, if (u, v) is added to a (potential maximum) clique, then all such vertices are deleted. So the following recursive equation holds for the run time $T(nm)$:

$$T(nm) = T((n-1)(m-1)) + T(nm-1).$$

And by Lemma 1, we know the equation can be made better since there must be other neighbors of the vertex (u, v). The second term of the above equation can be expended as follows:

$$T(nm-1) \leq T((m-1)(n-1)) + T(nm-2).$$

Combining the above two equations, we get

$$T(nm) \leq 2T((n-1)(m-1)) + T(nm-2),$$

which leads to

$$T(nm) \leq mT((n-1)(m-1)) + T(nm-m).$$

This proves that maximum clique algorithms could achieve performance similar to that of our clique branch method. If we were to analyze (blindly) the run time of maximum-clique methods, without accounting for the number of vertices that are eliminated at each branching step, the best known algorithm would be assumed to take a running time of $O(2^{\frac{nm}{4}})$ [20].

We often solve maximum clique by reducing it to the minimum vertex cover problem, simply because a clique in a graph is the complement of a vertex cover in graph's complement. Recent efficient vertex cover algorithms based on the theory of fixed-parameter tractability [11] have proved very useful, especially when

the size of the clique is large [2]. Such algorithms target the parameterized version of a problem. A natural parameter that we could associate with the input of MCS is the size of the common subgraph. In other words, the parameterized MCS problem can be posed as follows:

Given: A pair of graphs, G_1 and G_2, and a positive integer k.

Question: Do G_1 and G_2 have a common (induced) subgraph whose order is at least k?

Let G be the association graph of G_1 and G_2. The search for a common subgraph of size k (or more) is equivalent to the search for a k-clique in G. Thus, as noted earlier, we look for a vertex cover of size $nm - k$ in the complement of G. Since k is not larger than n, the size of the sought cover is bounded below by $nm - \sqrt{nm}$, which is huge when compared to nm. So the use of fixed-parameter vertex cover algorithms may seem not feasible for k-MCS. This is also supported by the fact that MCS is W[1]-hard [11]. Nevertheless, our algorithm is a straightforward branching approach that achieves the best current running time for MCS. The advantage of using parameterized vertex cover algorithms (rather than direct clique algorithms) is mainly due to two observations. First, vertex cover branching uses the same universal strategy: if a vertex is not in the cover, then all its neighbors must be in the cover. So the two algorithms explore essentially the same search space. Second, when the degree of each vertex in the complement graph drops below a certain constant c (due to the continual removal of vertices during branching), the resulting graph must have a vertex cover whose size is smaller than $\frac{c-1}{c}$ of the resulting graph size. (This being true since a graph of maximum degree c have an independent set of size at least $1/c$ of the graph size). Thus, parameterized vertex cover techniques, such as preprocessing and kernelization [1, 3], may be applied together with branching to reduce the size of the search space and produce better run times. Finally, for completeness it probably worth pointing out that, if one is dealing with labeled graphs, then MCS is potentially an easier problem. Simpler maximum clique algorithms are often used on labeled and other restricted types of association graphs [7, 12] to achieve better performance.

References

[1] F. N. Abu-Khzam, R. L. Collins, M. R. Fellows, M. A. Langston, W. H. Suters, and C. T. Symons. Kernelization algorithms for the vertex cover problem: Theory and experiments. In *Proceedings, Workshop on Algorithm Engineering and Experiments (ALENEX)*, 2004.

[2] F. N. Abu-Khzam, M. A. Langston, and P. Shanbhag. Scalable parallel algorithms for difficult combinatorial problems: A case study in optimization. In *Proceedings, International Conference on Parallel and Distributed Computing and Systems (PDCS)*, pages 563–568, 2003.

[3] F. N. Abu-Khzam and M. A. Langston W. H. Suters. Effective vertex cover kernelization: A tale of two algorithms. In *Proceedings, ACS/IEEE International Conference on Computer Systems and Applications (AICCSA)*, 2005.

[4] T. Akutsu. A polynomial time algorithm for finding a largest common subgraph of almost trees of bounded degree. *IEICE Trans. Fundamentals*, E76-A:1488–1493, 1993.

[5] I. Bomze, M. Budinich, P. Pardalos, and M. Pelillo. The maximum clique problem. In D. Z. Du and P. M. Pardalos, editors, *Handbook of Combinatorial Optimization*, volume 4. Boston MA: Kluwer Academic Publishers, 1999.

[6] C. Bron and J. Kerbosch. Finding all cliques of an undirected graph. *Communications of the ACM*, 16:575–577, 1973.

[7] H. Bunke, P. Foggia, C. Guidobaldi, C. Sansone, and M. Vento. A comparison of algorithms for maximum common subgraph on randomly connected graphs. In *Proc. IAPR Workshop on Structural and Syntactic Pattern Recognition*, 2002.

[8] K. Shearer H. Bunke and S. Venkatesh. Video indexing and similarity retrieval by largest common subgraph detection using decision trees. *Pattern Recognition*, 34:1075–1091, 2001.

[9] D. Conte, P. Foggia, C. Sansone, and M. Vento. Thirty years of graph matching in pattern recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(3):265–298, 2004.

[10] D. Conte, C. Guidobaldi, and C. Sansone. A comparison of three maximum common subgraph algorithms on a large database of labeled graphs. In E. Hancock and M. Vento, editors, *IAPR Workshop GbRPR 2003, LNCS 2726*, pages 130–141, 2003.

[11] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.

[12] P.J. Durand, R. Pasari, J.W. Baker, and Chun che Tsai. An efficient algorithm for similarity analysis of molecules. *Internet Journal of Chemistry*, 2, 1999.

[13] V. Kann. On the approximability of the maximum common subgraph problem. In *STACS '92: Proceedings of the 9th Annual Symposium on Theoretical Aspects of Computer Science*, pages 377–388. Springer-Verlag, 1992.

[14] E. B. Krissinel and K. Henrick. Common subgraph isomorphism detection by backtracking search. *Software Practice and Experience*, 34:591–607, 2004.

[15] A. Massaro and M. Pelillo. Matching graphs by pivoting. *Pattern Recognition Letters*, 24(8):1099–1106, 2003.

[16] J. McGregor and P. Willett. Use of a maximal common subgraph algorithm in the automatic identification of the ostensible bond changes occurring in chemical reactions. *Journal of Chemical Information and Computer Science*, 21:137–140, 1981.

[17] J. J. McGregor. Backtrack search algorithms and the maximal common subgraph problem. *Software Practice and Experience*, 12:23–34, 1982.

[18] B. T. Messmer. *Efficient Graph Matching Algorithms for Preprocessed Model Graphs*. Phd, University of Bern, 1995.

[19] J. W. Raymond and P. Willett. Maximum common subgraph isomorphism algorithms for the matching of chemical structures. *Journal of Computer-Aided Molecular Design*, 16:521–533, 2002.

[20] J. M. Robson. Finding a maximum independent set in time $O(2n/4)$. Technical Report 1251-01, Universite Bordeaux I, LaBRI, 2001.

[21] A. Yamaguchi, K. F. Aoki, and H. Mamitsuka. 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.