The Maximum Common Subgraph Problem: Faster Solutions via Vertex Cover*

Faisal N. Abu-Khzam†
Division of Computer Science and Mathematics
Lebanese American University
Beirut, Lebanon
faisal.abukhzam@lau.edu.lb

Nagiza F. Samatova
Computer Science and Mathematics Division
Oak Ridge National Laboratory
P.O. Box 2008, Oak Ridge, TN 37831–6367, USA
samatovan@ornl.gov

Mohamad A. Rizk
Division of Computer Science and Mathematics
Lebanese American University
Beirut, Lebanon
mohammad.rizk@lau.edu.lb

Michael A. Langston
Department of Computer Science
University of Tennessee
Knoxville, TN 37996–3450, USA
langston@cs.utk.edu

Abstract

In the maximum common subgraph (MCS) problem, we are given a pair of graphs and asked to find the largest induced subgraph common to them both. With its plethora of applications, MCS is a familiar and challenging problem. Many algorithms exist that can deliver optimal MCS solutions, but whose asymptotic worst-case run times fail to do better than mere brute-force, which is exponential in the order of the smaller graph. In this paper, we present a faster solution to MCS. We transform an essential part of the search process into the task of enumerating maximal independent sets in only a part of only one of the input graphs. This is made possible by exploiting an efficient decomposition of a graph into a minimum vertex cover and the maximum independent set in its complement. The result is an algorithm whose run time is bounded by a function exponential in the order of the smaller cover rather than in the order of the smaller graph.

*This research has been supported in part by the Lebanese American University under grant URC-c2004-63, by the U.S. Department of Energy Office of Advanced Scientific Computing Research (DOE OASCR). The work of N. F. Samatova was also sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory. Oak Ridge National Laboratory is managed by UT-Battelle for the LLC U.S. D.O.E. under contract no. DE-AC05-00OR22725. The work of M. A. Langston was supported in part by the U.S. National Institutes of Health under Grant 1-R01-MH-074460-01.

†Communicating author

1 Introduction and Background

Graph H is said to be *common* to graphs G_1 and G_2 if both G_1 and G_2 contain induced subgraphs isomorphic to H. *Maximum Common Subgraph* (MCS) is usually posed as the following decision problem.

> Inputs: A pair of graphs, G_1 and G_2, and a positive integer k.
>
> Question: Do G_1 and G_2 have a common subgraph of order k or more?

MCS finds application in many domains. It has been used in bioinformatics [15], chemistry [9, 10], pattern recognition [3, 8], and a variety of other areas. Unfortunately, MCS is an exceedingly difficult problem, with several well-known $\mathcal{NP}$-complete problems reducing to it. The maximum clique problem, for example, corresponds to the special case in which $|G_1| = |G_2|$ and G_1 is complete. (Note

that maximum clique is so hard that it cannot even be approximated to within a sublinear factor unless $NP = coR$ [6].)

Graphs we consider are simple and undirected. We adopt the following terminology:

- $n = |G_1|$ and $m = |G_2|$, with $n \leq m$;
- $V(G_i)$ is the vertex set of G_i;
- For a set S of vertices of a graph, $N(S)$ denotes the set of neighbors of the elements of S that are not in S.
- $G_i[S]$ is the subgraph induced by S in G_i.

Graph theory is full of studies that focus on special types of subgraphs. A subgraph type is referred to by a *structure* in this paper. A structure we use throughout this effort is a graph's vertex cover, perhaps best known for the central role it plays in the theory of fixed-parameter tractability [1]. A vertex cover of a graph is a set of vertices whose complement induces an edgeless subgraph. Moreover, vertices that induce an edgeless subgraph form an *independent set*. Finding a vertex cover of minimum size in a graph is equivalent to finding an independent set of maximum size. Both problems are NP-hard.

In this paper, we provide a solution for the search version of MCS. Our approach targets vertex cover structures. Unless the two input graphs are pathologically dense, the smaller minimum vertex cover is apt to be much smaller than the order of the smaller input graph. Restricting the usual exhaustive search to a vertex cover of one graph, say G_1, can be more efficient, because it would be followed by a relatively easy search for an independent set in G_2 that can be matched with the complement of the vertex cover of G_1. As we shall show, this resulting problem can then be reduced to one of finding a maximum matching for which the endpoints of the matching edges must satisfy a few additional constraints. We call this problem "constrained maximum matching" (CMM for short).

The asymptotically best current algorithms for MCS run in $O^*((m+1)^n)$ time [12]. Our main result is to show that, using our approach, MCS can be solved in $O^*(3^{m/3}(m+1)^c)$ time, where c denotes the size of the smaller minimum vertex cover between the two inputs.

This paper is structured as follows. In the next section, we review the previously-best methods for solving MCS and discuss a simple generic backtracking method. In Section 3, we present our MCS algorithm and derive bounds on its performance. We define the CMM problem in Section 4 and determine its complexity. In a final section, we describe extensions to our technique, and present a few concluding remarks.

2 Exact Solutions for the MCS Problem

In the process of searching for a common subgraph, we try to find a one-to-one function, f, from $V(G_1)$ to $V(G_2)$, such that a subset of the domain and its image in the co-domain are isomorphic induced subgraphs of G_1 and G_2, respectively. In this context, if $v = f(u)$, then we say u and v are matched or v is matched with u.

Let $M = S_1 \times S_2 \subset V(G_1) \times V(G_2)$ be such that S_1 and S_2 induce isomorphic subgraphs. We refer to M as a set of compatible pairs. If a pair (u, v) of $(V(G_1) - S_1) \times (V(G_2) - S_2)$ such that $G_1[S_1 \cup \{u\}]$ is isomorphic to $G_2[S_2 \cup \{v\}]$, then we say the pair (u, v) is compatible with M.

2.1 Brute-Force and Backtracking Algorithms

To find a maximum common subgraph, brute-force techniques by definition must explore the space of all possible maximal common subgraphs. One way of doing so is to enumerate all functions from $V(G_1)$ to $V(G_2) \cup \{NONE\}$, where NONE is a dummy node that serves as the image of unmatched vertices of G_1. The number of all such functions is $(m+1)^n$, but not all of them are isomorphisms between subgraphs of G_1 and G_2. In fact, unless we are dealing with some very special (and rather trivial) instances, the majority of such functions are not interesting to us (in the sense that they are not valid subgraph isomorphisms). Hence $(m+1)^n$ is a very loose upper bound. To explicate further, note that among all functions from $V(G_1)$ to $V(G_2)$, we want a function f that satisfies the following:

1. The restriction of f to $D_f = V(G_1) - \{v \in V(G_1) : f(v) = NONE\}$ is a one-to-one function.
2. The subgraphs induced by D_f and $f(D_f)$ are isomorphic.

We want an enumerator that guarantees the above conditions. And we are not interested here in getting a list of all maximal common subgraphs. For the sake of completeness, we describe our simplified version of the generic backtracking method. This presentation makes the backtracking technique easy to understand and implement.

During the search for a maximum common subgraph, we keep track of the following items:

> The largest set, M, of compatible pairs found so far.
>
> The cardinality of M, denoted by $maxsize$.
>
> For each $i \in V(G_1)$: the set $M_1[i]$ of all vertices of G_2 that could be matched with i.
>
> For each $i \in V(G_2)$: the set $M_2[i]$ of all vertices of G_1 that could be matched with i.

For each $i \in V(G_j)$: the cardinality of $M_j[i]$, denoted by $matchnum_j[i]$ ($j \in 1, 2$).

We proceed in a branch-and-search manner, as if we are traversing a search tree T. Each node of T has a current set of compatible pairs, $currentM$, consisting of all pairs that have been matched, thus found compatible so far (along the path from root to current node). The size of $currentM$ is dubbed $currentsize$.

Note that T is a virtual tree. It is traversed (via depth-first), but not constructed explicitly. Each node of T could be visualized as being labeled with a vertex of G_1 that is selected according to a heuristic criteria (such as minimum value in $matchnum_1$). Edges of T are labeled with vertices of G_2 or $NONE$. Note that two or more nodes of T could have the same label, unless they belong to the same path from the root to a leaf (of T). If the current node is linked to its parent u via edge x, then $u \in G_1$ has been matched with $x \in G_2$ in the previous step. Also, in this case, (u, x) is an element of the $currentM$ set found along the path from the root to the child of node u that is connected to u via edge x. If no matching occurred at the previous step (or level) then the label of the edge will be $NONE$ (instead of x). Let v be the current node in the search process, then v ($\in V(G_1)$) could be associated with a vertex, say y, from $M_1[v] \cup NONE$. For each association of v to a vertex from $M_1[v]$, we perform the following steps:

1. Delete y from $M_1[i], \forall i \in V(G_1)$;
2. Delete v from $M_2[j], \forall j \in V(G_2)$;
3. Add (v, y) to $currentM$ and increment $currentsize$;
4. If $currentsize > maxsize$, then $M \leftarrow currentM$;
5. Proceed by selecting another vertex from G_1.

Undoing the above steps is one of the bottlenecks of the implementation. In particular, the code has to account for the temporary deletion of vertices from M_1 and M_2. We save the reader from the implementation details as they are not unique. It should be clear, by now, that a recursive backtracking algorithm for MCS takes a very long time on graphs of small sizes.

Our presentation (above) of the backtracking algorithm is similar to the one proposed in [7], which offers the current best enhancement of the old backtracking algorithm proposed by Ullman in his work on subgraph isomorphism [14]. As we said earlier, such backtracking algorithms suffer from their worst-case behavior, which seems inevitable. The worst-case run-time of the above algorithm is $O((m+1)^n)$.

2.2 The Use of Compatibility Graphs

Another common approach to the MCS problem is to look for the largest set of compatible pairs in a compatibility graph called the association graph.

The association graph of G_1 and G_2, henceforth $A(G_1, G_2)$, is a graph whose vertices are the elements of $V(G_1) \times V(G_2)$. An edge joins (u, u') and (v, v') if the pairs (u, v) and (u', v') exhibit the same relationship (both adjacent or both non-adjacent) in G_1 and G_2, respectively. In other words, (u, u') and (v, v') are adjacent if they are locally compatible ($\{u, v\}$ and $\{u', v'\}$ induce isomorphic subgraphs of size two).

To get the largest common subgraph, we need the largest set of pairwise compatible pairs. This is a maximum clique in $A(G_1, G_2)$. So we can rely on optimization algorithms for maximum clique to solve MCS. However, the best known general purpose maximum clique algorithm has a worst-case run time in $O(2^{|A(G_1,G_2)|/4})$, which is $O(2^{mn/4})$ [11]. This looks far worse than the brute-force method described previously. Nevertheless, recent analysis has shown that clique-based methods can take advantage of structure, are relatively straightforward to implement, and have exactly the same asymptotic worst-case behavior as brute-force backtracking [12]. These results rely on exploiting the fact that pairs with a common component cannot be part of the desired clique, a property that is automatically assumed in backtracking methods because vertices that are matched are subsequently deleted or ignored.

3 Exploiting Structure via Vertex Covers

We describe our main approach and discuss the details of our algorithm, which is dubbed MCS_VC in the sequel.

Let C be a vertex cover of size k in G_1. We try to match the vertices of C with vertices of G_2 by exhaustive search, as in the brute-force backtracking algorithm. This leads to a number of candidate subgraphs of $G_1[C]$ that can be extended to a common subgraph of G_1 and G_2. Each such candidate subgraph, say S, is a common subgraph between G_1 and G_2, but it could be extended further by adding vertices from the complement of C.

3.1 Maximal Independent Set Enumeration

Let us assume that a common subgraph S containing exactly i vertices of C has been produced, in such a way that no other vertex of C is to be considered ($0 \leq i = |S| \leq |C|$). Then we extend S by adding vertices of the independent set $I = G_1 \backslash C$. Elements of C that are not in S are deleted from M_1 and M_2 (not considered in subsequent steps of the matching process), together with elements of I

that cannot be matched with any vertex of G_2. Similarly, if a vertex of G_2 cannot be matched with any vertex of I, then it is deleted.

We observe that vertices of G_2 that can be added to the target MCS must form an independent set, otherwise the resulting MCS would not be induced in G_2. After deleting all unmatchable vertices, we construct a graph H as follows:

- Vertices of H are partitioned into two sets H_1 and H_2 such that:
  - H_1 contains the elements of I
  - H_2 contains the vertices of G_2 that are not part of S and are yet to be matched.
- Edges of H are colored with either red or green such that:
  - A green edge connects $u \in H_1$ to $u' \in H_2$ iff (u, u') forms a compatible pair. In other words, any matching that consists of green edges of H indicates a possible extension of S.
  - A red edge joins vertices of H_2 only (note that H_1 is an independent set). The endpoints of a red edge are adjacent vertices of G_2. Hence, a red edge (u, v) of H is an indication that at most one of u and v can be added to the common subgraph that is isomorphic to S.

An extension of S to a maximal common subgraph is a *matching* that consists of green edges only (from H) such that the vertices of H_2 that are matched form an independent set (of H_2, thus an independent set of G_2 as well). This is what we called earlier a CMM (*constrained maximum matching*). We shall prove in the next section that CMM is NP-complete. So far, we chose to solve it by enumerating all maximal independent sets of H_2. For each such independent set, say I', we find a maximum matching of green edges in the bipartite subgraph of H induced by $H_1 \cup I'$. The whole process takes $O(3^{m/3}m^{2.5})$ and is repeated for each candidate $S \subset C$. This worst-case run time is due to the following:

- Enumerating all maximal independent sets of H_2 takes $O(3^{m/3})$ time [2, 13].
- The maximum matching problem is solvable in $O(m^{2.5})$ time.

The enumeration process is repeated for each candidate $S \subset C$. There are $(m+1)^k$ such candidates. Hence, the run time of this algorithm takes

$$O^*(3^{m/3}(m+1)^k) \tag{1}$$

According to our previous discussion, the worst-case run time of any of the known MCS algorithms is not better than $O((m+1)^n)$. The algorithm just described has a better worst-case behavior, as long as k is small enough. Also, the expression $3^{m/3}$ should be small (asymptotically) when compared to $(m+1)^{n-k}$.

Note that the roles of G_1 and G_2 can be swapped in equation 1, when k becomes the size of a vertex cover of G_2. Therefore we have the following steps:

1. Compute minimum vertex covers C_1 and C_2 of G_1 and G_2, respectively. Let $k_1 = |C_1|$ and $k_2 = |C_2|$. This pre-processing takes $O^*(1.274^{k_1} + 1.274^{k_2})$ [4].
2. If $3^{m/3}(m+1)^{k_1} < 3^{n/3}(n+1)^{k_2}$, then apply the above algorithm (with $C = C_1$), otherwise C_2 is used and the above algorithm is applied with the roles of G_1 and G_2 interchanged.

The analysis conducted so far leads to the following.

Theorem 1 *Let n, m and k be as described above, then MCS is solvable in time $O(1.274^k + 3^{m/3}(m+1)^k)$, where k is the minimum vertex cover size of the input graph whose size is n.*

Proof. The proof follows from the preceding analysis. The extra 1.274^k term is due to the use of the asymptotically best current vertex cover algorithm [4]. ■

3.2 Experimental Analysis

We have implemented the above algorithm and compared it to the CSI code written by the authors of [7]. The two codes have been tested on numerous randomly generated pairs of graphs in such a way that one graph of each pair has a relatively small vertex cover. We have witnessed speedups that range from 2 to more than 1000. These were obtained on graphs whose minimum vertex cover size is smaller than $\frac{n}{2}$ (between $\frac{n}{4}$ and $\frac{n}{3}$). As the vertex cover size increases, our enhanced version slows down until it reaches cases where its behavior becomes somewhat sporadic. We have not been able to quantify (experimentally) a maximum value for k/n above which the speedup drops below one. This is not a surprise because the order of selecting vertices from G_1 (to match them) could affect the run time both in positive or negative ways. So restricting this selection to vertices of C may lead to unpredictable behaviors in cases where C has a large size.

The following table shows sample run times obtained by running the CSI code and our MCS_VC code on randomly generated pairs of graphs. The graph G_1 is first generated so that it has a small vertex cover. The same G_1 is then used in three different pairs, each with a different value of

Graph 1			Graph 2		MCS	CSI	MCS_VC
n	e	VC size	m	e	**size**	**time**	**time**
20	35	5	30	82	18	1437	2
20	35	5	40	135	18	3048	3
20	35	5	50	293	19	741	2
20	36	7	30	82	16	126	4
20	36	7	40	135	18	7	2
20	36	7	50	293	19	41	2
30	96	8	40	135	25	$>$ 4 hours	36
30	96	8	50	293	26	$>$ 4 hours	164
30	96	8	60	355	26	$>$ 4 hours	470
30	82	10	40	135	24	$>$ 4 hours	48
30	82	10	50	293	25	$>$ 4 hours	167
30	82	10	60	355	26	$>$ 4 hours	119
40	175	13	50	235	31	$>$ 2 days	20240
40	135	14	50	284	30	$>$ 2 days	18293

Table 1. Sample experimental results

$m = |G_2|$. Note that e denotes the number of edges of the corresponding graph (columns 2 and 5 of Table 1).

The above experiments were conducted on a 3.4 GHz Pentium 4 machine with 1 GB RAM. Unless otherwise stated, the run times shown in the table are in seconds. Entries that start with the sign '$>$' ($>$ 4 hours and $>$ 2 days) correspond to experiments that were terminated manually before reaching solutions.

3.3 The Use of Maximum Independent Sets

There are cases where $3^{m/3}$ is a high price to pay, while maybe direct backtracking could lead to faster convergence, due to pruning strategies. This occurs when the number of green edges of H is relatively small, indicating that backtracking would lead to further restrictions and fewer branches at the majority of search-tree nodes. We offer an alternative approach that adds gadgets to the graph H and allows us to find optimal extensions of S using a maximum independent set solution.

Again, recall that the main problem, after constructing H, is that of finding a maximum matching of green edges such that matched vertices of G_2 (i.e., elements of H_2) form an independent set. If we start by computing a maximum independent set, J, of H_2, then we could run into the following problem:

Two or more vertices of J match with only one vertex of H_1, while we may have another non-maximum independent set whose vertices match with a larger number of elements of I.

What if we insist on finding a maximum independent set of H_2 but we prohibit the inclusion of more than one neighbor of any element of H_1? Then we have another problem:

A neighbor of $u \in H_1$ could be a neighbor of another vertex $v \in H_1$ (thus we loose matching it with v).

The above discussion motivates the following construction:

Step 1 If $x \in H_2$ has neighbors $\{u_1, u_2, ., u_t\} \subset H_1$, then replace x by the clique $\{x_1, x_2, ., x_t\}$ and add edges (x_i, u_i) to replace edges (x, u_i). Moreover, each x_i inherits all the red edges of x: if (x, y) is a red edge, then we add edges (x_i, y) for each i. This means that picking x_i in the independent set is equivalent to matching x with u_i. This guarantees that every "new" vertex of H_2 has a unique neighbor in H_1.

Step 2 If $u \in H_1$ has more than one neighbor in the new set H_2, then add red edges between any pair of neighbors of u. This guarantees that only one neighbor of u can be matched (after the construction in step 1, we cannot match a neighbor of u with another vertex).

Let H' be the graph constructed from H using the above two steps. And let H'_2 be the set resulting from the extension of H_2. Then we have the following.

Claim 2 *A maximum independent set of H'_2 yields an optimal extension of the common subgraph S.*

Proof. Let J be a maximum independent set of H_2'. By step 1, every element of H_2 appears once in J and has a unique neighbor in H_1. By step 2, no two elements of J have a common neighbor in H_1. Therefore, elements of J are matched with their neighbors in H_1. This produces an extension of S. On the other hand, let J' be an extension of S in G_2. Then J' is an independent set of H_2 whose elements form a perfect matching with $N(J')$ in H_1. It is not hard to verify that the construction in steps 1 and 2 leads to an independent set of H' whose size is bounded below by $|J'|$. ■

We apply the second approach when the number of green edges is bounded above by cm where $1.189^c < 3^{1/3}$. The reason stems obviously from the following fact: the current fastest maximum independent set algorithm runs in time $O(1.189^{cm})$ on the resulting G_2 part of H' [11]. Moreover, the second approach has the advantage of avoiding the computation of a maximum matching for each of the $O(3^{m/3})$ cases.

Finally, our methods work well when any of the two graphs has a small (relatively) vertex cover. What if none of them enjoys this property? If we insist on using the vertex cover approach, then we can check if any of the two complements of G_1 and G_2 has a small vertex cover. This relies on the following observation.

Observation 3 *If S is a maximum common subgraph of G_1 and G_2, then the complement of S is a maximum common subgraph of the complements of G_1 and G_2.*

Proof. This follows immediately from the fact that two graphs are isomorphic if and only if their complements are isomorphic. ■

4 Constrained Maximum Matching

A special matching problem appeared in the previous section, when we tried to match[1] an independent subset of $V(G_1)$ with a subgraph of G_2. The construction of the graph H transformed this task to the problem of finding a maximum matching with an additional constraint. We called this problem "Constrained Maximum Matching" or CMM, for short. We define a general decision version of CMM formally as follows:

> Given: a graph H, a pair of subgraphs H_1 and H_2 of H, and a positive integer k.
>
> Question: Does H have a matching M of size at least k such that every edge of M joins a vertex of H_1 to a vertex of H_2, and no two adjacent elements of H_i are matched ($i \in \{1, 2\}$).

[1]The word "match" is used here as we indicated in section 2, in the context of identifying isomorphic subgraphs.

It is sometimes convenient to refer to the particular partition of H into H_1 and H_2 when dealing with specific instances of the problem. We shall use the expression $(H_1, H_2)-CMM$ for such instances. The version of CMM that we have encountered in the previous section consists of instances where H_1 is an independent set. If H_2 is an independent set, then we are dealing with maximum matching for bipartite graphs (which is solvable in poly-time). The CMM problem is not as easy as the well-known maximum matching problem.

Theorem 4 *CMM is NP-complete.*

Proof. The proof is by reduction from the Maximum Independent Set problem (MIS). Let $(G = (V, E), k)$ be an instance of MIS. Then we construct another graph $G' = (V \cup V', E \cup E')$, with $V' = |E'| = |V|$ as follows: For each vertex $u \in V$, we add vertex u' to V' and edge $\{u, u'\}$ to E'. We claim that G has an independent set of size k if and only if G' has a (V', V)-CMM of size k. This can be seen by observing that any matched subset of V must form an independent set, and any independent set of V satisfies the constraint of CMM. ■

Note that the reduction used in our proof shows also that it is NP-hard to get any reasonable approximate solution, knowing that MIS is hard to approximate to within $n^{1-\delta}$ [5]. This justifies our application of exact exponential-time methods in the previous section.

5 Concluding Remarks

In this paper we have shown how to use vertex covers, along with their complementary independent sets, to produce algorithms for the MCS problem that are faster than previously-known methods. We observe that vertex cover structures can be used to enhance further our backtracking algorithms. For example, one possible enhancement is to use two vertex covers, one for G_1 and another for G_2. The matching process begins by enumerating all feasible matchings between the two covers, then between the cover of G_i and the independent set of G_{3-i}, and at last matching vertices of the two independent sets (an instance of the maximum matching problem). Finally, we note that our vertex cover approach may be applicable to problems other than MCS. We believe this is a possibility warranting continued study.

Acknowledgments

We are indebted to the authors of [7] for making their code available to us.

References

[1] F. N. Abu-Khzam, M. A. Langston, P. Shanbhag, and C. T. Symons. Scalable parallel algorithms for FPT problems. *Algorithmica*, 45:269–284, 2006.

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

[3] 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.

[4] J. Chen, I. Kanj, and G. Xia. Improved parameterized upper bounds for vertex cover. In *31st International Symposium on Mathematical Foundations of Computer Science (MFCS 2006), Lecture Notes in Computer, volume 4162*, pages 238–249, 2006.

[5] L. Engebretsen and J. Holmerin. Clique is hard to approximate within $n^{1-o(1)}$. In *27th International Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science, volume 1853*, pages 2–12. Springer-Verlag, Geneva, 2000.

[6] J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. In *Proc. 37th Ann. Symp. Found. Comput. Sci.*, pages 627–636, 1996.

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

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

[9] 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.

[10] 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.

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

[12] W. Henry Suters, F. N. Abu-Khzam, Y. Zhang, C. T. Symons, N. F. Samatova, and M. A. Langston. A new approach and faster exact methods for the maximum common subgraph problem. In *11th International Computing and Combinatorics Conference, Kunming, China, Lecture Notes in Computer Science, volume 3595*, pages 717–727. Springer, August 2005.

[13] E. Tomita, A. Tanaka, and H. Takahashi. The worst-case time complexity for generating all maximal cliques. In *Computing and Combinatorics Conference (COCOON)*, August 2004.

[14] J. R. Ullman. An algorithm for subgraph isomorphism. *Journal of the ACM*, 23(1):31–42, 1976.

[15] 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.