Efficient Counting of Maximal Independent Sets in Sparse Graphs

Fredrik Manne and Sadia Sharmin

Dep. of Informatics, Univ. of Bergen, Norway
{fredrikm,Sadia.Sharmin}@ii.uib.no

Abstract. There are a number of problems that require the counting or the enumeration of all occurrences of a certain structure within a given data set. We consider one such problem, namely that of counting the number of maximal independent sets (MISs) in a graph. Along with its complement problem of counting all maximal cliques, this is a well studied problem with applications in several research areas.

We present a new efficient algorithm for counting all MISs suitable for sparse graphs. Similar to previous algorithms for this problem, our algorithm is based on branching and exhaustively considering vertices to be either in or out of the current MIS. What is new is that we consider the vertices in a predefined order so that it is likely that the graph will decompose into multiple connected components. When this happens, we show that it is sufficient to solve the problem for each connected component, thus considerably speeding up the algorithm. We have performed extensive experiments comparing our algorithm with the previous best algorithms for this problem using both real world as well as synthetic input graphs. The results from this show that our algorithm outperforms the other algorithms and that it enables the solution of graphs where other approaches are clearly infeasible.

As there is a one-to-one correspondence between the MISs of a graph and the maximal cliques of its complement graph, it follows that our algorithm also solves the problem of counting the number of maximal cliques in a dense graph. To our knowledge, this is the first algorithm that can handle this problem.

1 Introduction

Enumerating all configurations that conforms with a given specification is a well studied problem in combinatorics. Graph theory deals with many interesting problem of this type. Enumerating all maximal independent sets (MISs) of a graph is one of these problems that has attracted considerable attention in the past [5,10,12,14,20]. This problem is also equivalent to enumerating all the maximal cliques of a graph as there is a one-to-one correspondence between the MISs of a graph and the maximal cliques of its complement graph. For a recent overview of applications of this problem, see [6] and the references therein.

In the classical MIS enumeration problem, the number of configurations to be generated is potentially exponential in the size of the input. Moon and Moser showed that a graph on n vertices can have at most $3^{\frac{n}{3}}$ MISs and that this bound is tight [16]. Thus for

V. Bonifaci et al. (Eds.): SEA 2013, LNCS 7933, pp. 103–114, 2013.

graphs coming close to this bound one can only expect to be able to list or enumerate all MISs of graphs of fairly limited size.

In this paper we study algorithms for *counting* the number of MISs, a problem one might expect can be solved more efficiently than the enumeration problem. However, it is known that the counting problem is ♯P-complete even when restricted to chordal graphs [17], and therefore no polynomial time algorithm exists unless P=NP [12].

The currently fastest algorithm for solving the MIS counting problem on general graphs is by Gaspers et al. who presented a branch and bound algorithm that runs in $O(1.3642^n)$ time [8]. As $3^{\frac{n}{3}} \approx 1.44^n$ this shows that it is possible to count MISs faster than by generating each one. For the case of sub-cubic graphs Junosza-Szaniawski and Tuczyński recently gave an algorithm with running time $O(1.2570^n)$ [11]. For trees, Wilf presented a simple linear time dynamic programming algorithm [21]. He also showed that the number of MISs in a tree is at most $2^{n/2-1} + 1$ and that there are graphs that meet this bound.

The counting problem can obviously be solved by enumeration, a problem which has seen a variety of approaches by a number of authors, see [5] for an overview. The standard algorithm for this problem is the Bron–Kerbosch algorithm [1] which is a recursive backtracking algorithm that searches for all maximal cliques in a given graph G, (which in the complement graph corresponds to the MISs). This algorithm was later improved by Tomita et al. [18] using a pivoting heuristic that reduces the number of recursive calls. We also note that Eppstein and Strash gave a variation of the Tomita algorithm by initially reordering the vertices using a degeneracy ordering [7], something that is advantageous for very sparse graphs.

Experimental work on these (and other) algorithms for enumerating cliques has mainly focused on sparse graphs [2,3,7,18]. This means that they are applicable on dense graphs for enumerating (or counting) MISs. Enumerating MISs on sparse graphs (or enumerating cliques in dense graphs) is a substantially harder problem as one would expect the number of MISs to decrease as the graph becomes denser. We are not aware of any experimental studies of algorithms for this problem.

Our Results: We present the first algorithm specifically suited for counting MISs in sparse graphs. The algorithm combines a branching approach with a divide and conquer strategy. This is achieved by making the branching follow vertex separators in the graph. In this way the remaining graph will become disconnected and one can solve the problem for each connected component separately. To find the separators we initially use graph partitioning software to compute a nested dissection ordering on the graph.

We apply the algorithm to both real world as well as synthetic sparse graphs and show that it outperforms other suggested algorithms designed for counting or enumerating MISs. Although the individual aspects of our algorithm are not new, this is, as far as we know, the first time they have been combined together to create an efficient code for counting MISs in sparse graphs.

2 Notation

We consider an undirected finite graph $G = (V, E)$ without loops, where V is the set of vertices of G and E is the set of edges. We denote the neighborhood of a vertex v in the

graph G by $N_G(v)$, that is the set of vertices u such that the edge $(u, v) \in E$. The closed neighborhood of a vertex v is denoted by $N_G[v]$, which is $N_G(v) \cup \{v\}$. The degree of a vertex $d_G(v) = |N_G(v)|$. If I is a set of vertices in G then $N_G(I) = (\cup_{v \in I} N_G(v)) \setminus I$. Also, the induced subgraph of I in G is the graph $G[I] = (I, E_I)$ where $(v, w) \in E_I$ if and only if $v, w \in I$ and $(v, w) \in E$.

A set of vertices S is an independent set if no two vertices in S are adjacent. A vertex $v \notin S$ which is adjacent to a vertex $w \in S$ is said to be *dominated* by w, or just dominated. A vertex not in S which is not dominated is undominated. A maximal independent set is an independent set that is not a subset of any other independent set.

3 Previous Algorithms for Counting and Enumerating MISs

In the following we present previously suggested algorithms for counting or enumerating all MISs of a graph G. For the enumeration problem we present algorithms that have been used in experimental studies and that are fairly straight forward to implement. We also outline the counting algorithm by Gaspers et al. Since our main interest is to count the number of MISs, we describe all algorithms as applied to this problem.

The Bron-Kerbosch algorithm in its basic form uses recursive backtracking to list all maximal cliques in a given graph [1]. In the following we present the dual of this algorithm, so that instead of cliques the algorithm counts all MISs in G.

Given three vertex sets R, P, and X, Algorithm 1: BKMIS(R, P, X) finds all MISs that include all vertices in R, any possible legal subset of the vertices from P, and none of the vertices in X. The recursion is initiated by setting both R and X to $\emptyset$ and $P = V$. Within each recursive call, the algorithm considers in turn every vertex in P for inclusion in R. Thus for each $v \in P$ the algorithm makes a recursive call in which v is moved from P to R and any neighbor of v is removed from P and X. In any subsequent call where both P and X are empty, R is counted as a MIS. This will find all maximal independent set extensions of R that contain v. When the recursive call returns, v is moved from P to X before the algorithm continues with the next vertex in P.

Intuitively, one can think of the algorithm as having already found the MISs that contain any vertex from X. Thus any set that does not dominate every vertex in X cannot be a new MIS.

Algorithm 1. BKMIS(R, P, X)

Input: Three vertex sets R, P, and X.
Output: Number of MISs containing all vertices in R, some from P and none from X.
if $P \cup X = \emptyset$ **then**
  Count R as a MIS
for each vertex $v \in P$ **do**
  BKMIS$(R \cup \{v\}, P \setminus N_G[v], X \setminus N_G(v))$
  $P \leftarrow P \setminus \{v\}$
  $X \leftarrow X \cup \{v\}$

The Bron–Kerbosch algorithm is not output-sensitive meaning that it does not run in polynomial time per generated set. The worst-case running time of the Bron–Kerbosch algorithm is $O(3^{\frac{n}{3}})$, matching the Moon and Moser bound [18].

Tomita et al. presented an improved variant of the Bron-Kerbosch algorithm by using a pivoting heuristic [18]. Here we present its dual for computing MISs. In Algorithm 1, $|P|$ recursive calls are made, one for each vertex in P. The pivoting strategy seeks to reduce this number. Consider a vertex $u \in P \cup X$. It follows that no vertex in $N_G[u]$ has been added to R so far. But for the current R to be expanded to a MIS at least one vertex of $P \cap N_G[u]$ must be included in R, otherwise R will not be maximal. Thus once the *pivot* u has been selected, it is sufficient to iterate over the vertices in $P \cap N_G[u]$ for inclusion in R. The idea in Algorithm 2: TOMITAMIS(R, P, X) is then to choose u such that this number is as small as possible. Computing both the pivot and the vertex

Algorithm 2. TOMITAMIS(R, P, X)

Input: Three vertex sets R,P and X.
Output: Number of MISs containing all vertices in R, some vertices from P and no vertex from X.
if $P \cup X = \emptyset$ **then**
  Count R as a MIS
Choose a pivot $u \in P \cup X$ that minimizes $|P \cap N_G(u)|$
for each vertex $v \in P \cap N_G[u]$ **do**
  TOMITAMIS($R \cup \{v\}$, $P \setminus N_G[v]$, $X \setminus N_G(v)$)
  $P \leftarrow P \setminus \{v\}$
  $X \leftarrow X \cup \{v\}$

sets for the recursive calls can be done in time $O(|P|(|P| + |X|))$ within each call to the algorithm using an adjacency matrix, giving an overall running time of $O(3^{\frac{n}{3}})$. Experimental comparisons have shown that the maximal clique algorithm by Tomita et al. is faster by orders of magnitude compared to other algorithms [18]. However, both the theoretical analysis and implementation rely on the use of an adjacency matrix representation of the input graph. For this reason, the algorithm has limited applicability for large graphs, whose adjacency matrix may not fit into working memory [7].

Eppstein et al. [6] also proposed a variant of the Bron-Kerbosch algorithm. On the top level this algorithm is similar to the Bron-Kerbosch algorithm, although the vertices are processed according to a degeneracy ordering. Such an ordering can be found by repeatedly selecting and removing a minimum degree vertex. The algorithm then makes $|V|$ calls to the algorithm by Tomita et al., each time with R initially set to the next vertex in the ordering and with P and X updated accordingly. With this setup the algorithm can be implemented to list all maximal cliques of an n-vertex graph in time $O(dn3^{\frac{d}{3}})$, where a graph has degeneracy d if every subgraph has a vertex of degree at most d. In a recent study Eppstein and Strash [7] show that the algorithm is highly competitive with the algorithm by Tomita et al. This is particularly true for large sparse graphs where it in many cases outperform the algorithm by Tomita et al. by orders of magnitude.

Gaspers et al. gave a fast exponential time algorithm of complexity $O(1.3642^n)$ for counting the number of MISs in a graph [8]. This running time is lower than the Moon and Moser bound, something that is possible since the algorithm, unlike the previous mentioned ones, does not enumerate the MISs but only counts their number.

The structure of the algorithm is similar to TOMITAMIS in that a vertex $u \in P \cup X$ is selected as a pivot according to a degree based criterion before branching on the vertices in $P \cap N[u]$. But unlike the previous algorithms, it will in each call first try if any of seven reduction rules can be applied to achieve a smaller but equivalent instance. If this is possible then the instance is reduced accordingly before calling the recursive function again. We note that all rules but one, will return the value given by the following recursive call. The only exception being a rule which checks if there exist two vertices u and v such that their current neighborhoods are identical. In this case v will be removed from the graph and the value of the recursive call will be returned plus the number of MISs discovered in this call that contained u. Another difference is that the algorithm tests if there is a vertex in X having no neighbor in P indicating that the current configuration cannot be expanded to a MIS. If this is the case then the algorithm returns immediately. In the paper it is also noted that if the graph at some stage should become disconnected then the algorithm is called (recursively) for each of its connected components, and the product of the returned values then gives the number of MISs. As far as we know there has been no study of how practical the algorithm is. We refer the interested reader to [8] for the details of the algorithm.

4 A New Algorithm

In the following we present a simple recursive branching algorithm for counting the number of MISs in a graph. Our algorithm is based on locating and exploiting vertex separators of the graph, and is similar in spirit to the algorithm by Lipton and Tarjan for computing a maximum independent set in a planar graph [13].

The Lipton and Tarjan algorithm initially finds a vertex separator $S \subset V$ such that $|S| = O(\sqrt{n})$ and such that no component of $G \setminus S$ contains more than $\frac{2}{3}|V|$ vertices. This is possible since G is assumed to be planar. Then for every independent set I_S of S the algorithm recursively finds a maximum independent set for each connected component of $G \setminus (S \cup N_G(I_S))$. The solution giving the combined largest solution is then the maximum independent set of G. The running time of the algorithm is $2^{O(\sqrt{n})}$.

We modify the Lipton and Tarjan algorithm to compute the number of MISs by using ideas from BKMIS and TOMITAMIS. Note however first that it is not possible to use the algorithm of Lipton and Tarjan to count MISs. The reason for this is that if we pick a particular independent set I_S from a separator S in G and (recursively) calculate the number of MISs in each component of $G[V \setminus (S \cup N_G(I_S))]$, then it is not given that I_S together with every combination of MISs from each of the components will form a MIS in G as some combinations might leave undominated vertices in $S \setminus I_S$.

The new algorithm, Algorithm 3: CCMIS, is recursive and uses two vertex sets P and X to count the number of MISs in $G[P \cup X]$ containing any combination of vertices from P while using none of the vertices in X. Thus if $P \cup X = \emptyset$ this will be counted as one MIS. Also, similar to the algorithm by Gaspers et al. if there exist a vertex in X that is not adjacent to any vertex in P then the algorithm will return 0, as this indicates that the current solution cannot be expanded into a complete MIS. The algorithm also tests at each level of recursion if $G[P \cup X]$ is connected. If this is not the case then the recursive procedure will be called once for each connected component and the product

of the number of MISs in each component will be returned. Checking for connectedness and listing the components is done using a linear depth first search through $G[P \cup X]$.

In the case that none of the mentioned conditions apply, the algorithm picks one remaining vertex v from P and then performs two recursive calls, first to compute the number of MISs containing v and then to compute the number of MISs excluding v. Finally, the sum of these two numbers is returned. When counting the number of MISs containing v, any vertex in $N[v]$ is first removed from P and X as these will be dominated by v. Similarly, when counting the number of MISs not containing v, the vertex v is moved from P to X as it must then be dominated by some other vertex in P in a MIS. Note that it is only following a recursive call where v is set to be in the current MIS that the structure of $G[P \cup X]$ will change so that there is any possibility of getting a disconnected graph. The recursion is initiated by setting $X = \emptyset$ and $P = V$.

Algorithm 3. CCMIS(P, X)

Input: Two vertex sets P and X.
Output: Number of MISs in $G[P \cup X]$ containing only vertices from P.
if $P \cup X = \emptyset$ **then**
  return 1
if $\exists w \in X$ with no neighbor in P **then**
  return 0
if $G[P \cup X]$ is not connected **then**
  $count \leftarrow 1$
  for each connected component $CC(V_{CC}, E_{CC})$ of $G[P \cup X]$ **do**
    $count \leftarrow count *$ CCMIS$(V_{cc} \cap P, V_{cc} \cap X)$
  return $count$
Select a vertex $v \in P$ to branch on
$count \leftarrow$ CCMIS$(P \setminus N_G[v], X \setminus N_G(v))$
$count \leftarrow count +$ CCMIS$(P \setminus \{v\}, X \cup \{v\})$
return $count$

As we explain in the following CCMIS differs substantially from the previous algorithms in which order the vertices are selected from P to branch on. It is clear from the description of CCMIS that one can select any vertex $v \in P$ to branch on. Thus one could similar to the previous algorithms use degree based information when selecting the branching vertex v. Picking a maximum degree vertex could be advantageous for the the first recursive call as it would give a maximum reduction in the size of $G[P \cup X]$, thus making it more likely that the remaining graph is disconnected. Picking a minimum degree vertex could be advantageous for the second recursive call as there would be fewer remaining vertices in P that could dominate v. However, as our main interest is in computing the number of MISs for sparse graphs we use a different selection criterion that exploits this. Algorithm 3: CCMIS has a considerable advantage over the Bron-Kerbosh type enumeration algorithms whenever the remaining graph becomes disconnected. This follows since the CCMIS algorithm does not have to generate every MIS but only needs to find the number of MISs in each connected component and then to multiply these numbers together. Although the algorithm by Gaspers et al. also exploit connected components in this way, their algorithm is bound to using a degree

based criterion when selecting a pivot. Thus this might limit how often the remaining graph becomes disconnected. Since we have no restrictions in CCMIS when selecting the branching vertex $v \in P$ we do so with the sole objective that the remaining graph should become disconnected.

Prior to running the algorithm we compute a *nested dissection* ordering $\alpha = \{v_1, v_2, \ldots, v_{|V|}\}$ on the vertices of G [9]. Such an ordering strives to number vertices that make up a (preferably small) separator S of G first, with the added constraint that the remaining components of $G \setminus S$ should be of roughly equal size. This is then repeated recursively for each connected component. One can also view a nested dissection ordering as an *elimination tree* [4]. This tree displays the separators in α, with vertices in a separator S making up a path hanging of the preceding separator S' on the component containing S. Within each separator, a vertex $v_j \in S$ will be a child of the highest numbered vertex $v_k \in S$ where $k < j$. If v_j, $j \neq 1$, is the first ordered vertex in S then v_j will be a child of the last ordered vertex of S', where S' is as defined above. It follows that a low elimination tree height is an indication that it was possible to (recursively) partition the graph using small separators.

As an example of a nested dissection ordering, consider the graph in Figure 1a. Then a possible α could be $\{d, c, a, b, f, e, g, h\}$. The d vertex is the first separator and c and f the two remaining ones. Note that the relative ordering between the vertices of the two components of $G \setminus \{d\}$ can be changed as long as c is ordered before a and b, and f is ordered before e, g, and h. Also, when a separator consists of multiple vertices their relative order is not necessarily important.

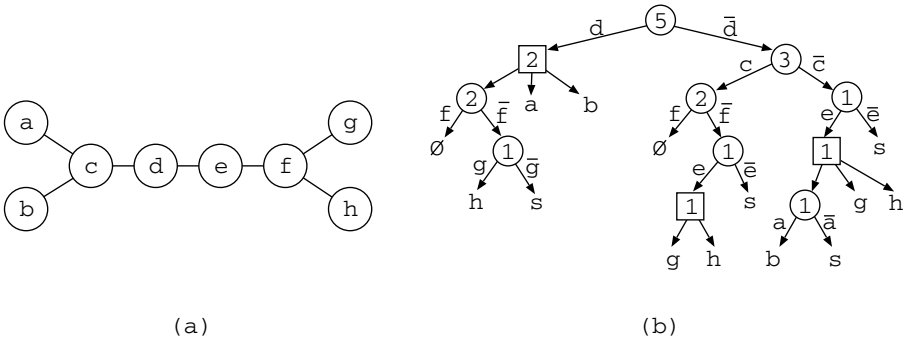

Fig. 1. A possible execution of Algorithm 3

The strategy we employ is now to choose the first vertex $w \in \alpha$ that is also in $P \cup X$. We have two cases for selecting the vertex v to branch on. If $w \in P$ then we set $v = w$ and if $w \in X$ then we select v to be a vertex in $P \cap N_G(w)$. Such a vertex must exist since the algorithm would already have returned if $w \in X$ had no neighbor in P. The effect of following α in this way is that we will only expand solutions where each vertex in S has either been included in the current MIS or is being dominated. Thus we are ensured that the remaining graph will be disconnected. Note that the strategy of picking a vertex in $N_G(w)$ to branch on whenever $w \in X$ is similar to the pivoting strategy in

TOMITAMIS. Comparing with the algorithm by Lipton and Tarjan the difference is that even though we follow the separator structure, for a particular separator S we allow for vertices in $N_G(S)$ to be assigned values before deciding exactly which vertices from S should be in the MIS.

The tree in Figure 1b shows the recursion tree of the algorithm when applied to the graph in Figure 1a. Each time the algorithm branches on a particular vertex is denoted by a round node, where the left branch denotes that the branching vertex is in the current MIS and the right branch that it is not. Whenever the remaining graph consist of just one vertex in P we only show the name of the vertex as it must be in any MIS. When the remaining graph is empty we write $\emptyset$, and if a particular branch cannot be extended to a MIS we write s. We use a square node to indicate when the graph has become disconnected and then draw one branch for each connected component. The number inside each node is the number of MISs returned by a particular branch.

With the current description of the algorithm there is still some freedom as to the order in which the branching vertices are selected. As already pointed out, we can reorder the vertices within a separator in α. Also, once a vertex $v \in S$ has been chosen to branch on then in the configuration where v is considered to be out of the current MIS, we are free to decide the order in which we pick vertices from $N_G(v) \cap P$ to dominate v. We will expand further on these issues in Section 5.

5 Experiments

In the following we describe experiments performed to evaluate the presented algorithms. All implementations have been performed on a Linux workstation running 64-bit Fedora 14, with Intel Core 2 Duo E6500 processors, and with 2GB of main memory. The programs are written in C (compiled with `gcc` (version 4.5.1) with the `-O3` flag) and Java (compiled with `javac` version 1.6.0_30). Each reported running time is the average of five runs.

We use graphs from TreewidthLIB [19]. This is a collection of approximately 700 graphs, among which we have chosen a set of 22 graphs drawn from areas such as computational biology, frequency assignment, register allocation problem, evaluation of probabilistic inference systems. The graphs were chosen so that in most cases our implementation of TOMITAMIS would terminate within 24 hours. This limited the maximum size to about 200 vertices. Moreover we also avoided most graphs having fewer than 10^6 MISs as all algorithms would spend less than a second on these. Table 1 gives the statistics for the chosen 22 graphs. Here p gives the edge density, eth gives the elimination tree height, while $MISs$ gives the number of maximal independent sets. In addition to these graphs we have performed experiments using rectangular grids.

Our first set of experiments concerns a comparison between the algorithm by Gaspers et al. and TOMITAMIS. In addition to the regular algorithm by Gaspers et al. we also implemented variants of it where we only apply the reduction rules at regular intervals, the most extreme case being when the reduction rules are not used at all. Since the algorithm by Gaspers et al. is by far the most complex of the considered algorithms, we have performed these comparisons using Java as this offers better support for more complex data structures such as sets. The results of the comparisons on nine representative graphs can be seen in the left plot of Figure 2. Here the first seven graphs are the

ones marked with a * in Table 1, while the 8th graph is a path on 40 vertices, and the 9th and 10th graphs are grids of size 7×7 and 8×8, respectively. The numbers are reported relative to the performance of the regular algorithm by Gaspers et al. (G100). G50 denotes the algorithm where the reduction rules are only applied in 50% of the recursive calls and G0 where they are not used at all.

Table 1. Description for benchmark real world graphs from TreewidthLIB [19]

Graph No.	Graph name	V	E	p	eth	MISs
1*	risk	42	83	0.01	13	66498
2*	pigs-pp	48	137	0.12	17	131402
3*	1sem	57	570	0.35	41	12405
4*	BN_100	58	273	0.16	31	134201
5*	1r69	63	692	0.35	46	22993
6*	1ail	69	631	0.26	44	160312
7	macaque71	71	444	0.18	30	182044
8	jean	80	508	0.16	22	1251960
9*	1aba	85	886	0.25	54	1067404
10	david	87	406	0.11	22	4.41×10^7
11	celar02	100	311	0.06	29	2.87×10^{10}
12	celar06	100	350	0.07	22	2.72×10^{10}
13	1lkk	103	1162	0.22	62	1.44×10^7
14	1fs1	114	1351	0.21	73	5.10×10^7
15	1a62-pp	120	1507	0.21	73	7.56×10^7
16	miles250	128	387	0.05	36	1.75×10^{13}
17	anna	138	493	0.05	23	2.75×10^{10}
18	mulsol1.i.5	186	3973	0.23	47	3.33×10^9
19	celar05	200	681	0.03	36	7.86×10^{20}
20	zeroin.i.3	206	3540	0.17	43	1.29×10^7
21	zeroin.i.2	211	3541	0.16	43	1.81×10^7
22	BN_93	422	1705	0.02	38	4.55×10^{11}

As can be observed there is no advantage in using the reduction rules, and when they are not used at all the performance is very similar to that of TomitaMis. Based on these results we did not pursue the algorithm by Gaspers et al. any further. For the remaining experiments all algorithms have been implemented in C as this gave considerable faster code compared to using Java.

We then compared BKMis, TomitaMis, and the algorithm by Eppstein et al. These experiments showed that, as expected, TomitaMis outperformed BKMis, while there was little difference between TomitaMis and the algorithm by Eppstein et al. We note that this last observation does not contradict the results in [7] as these were concerned with enumerating cliques in sparse graphs which is equivalent to enumerating MISs in dense graphs, while we are enumerating MISs in sparse graphs. Due to space constraints we omit these results.

Our next set of experiments concerns different variants of CCMis where we use Metis [15] to precompute a nested dissection ordering. The time spent on this was insignificant compared to the algorithm itself and is not included in the timings.

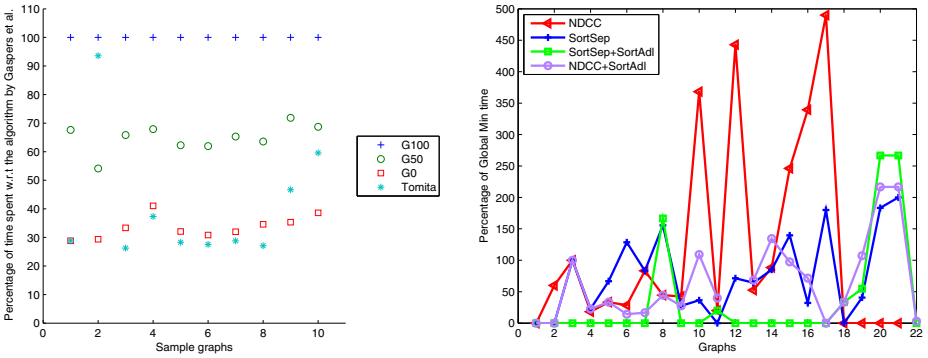

Fig. 2. Relative performance of TOMITAMIS compared to the algorithm by Gaspers et al. (left), and relative performance of different CCMIS algorithms (right).

The versions we tried include the basic algorithm (NDCC) where the vertices are processed for branching according to the ordering given by Metis and versions where we reorder the vertices within each separator and also the relative order of the neighbor lists. Similar in spirit with TOMITAMIS we tried a version where one branches on a vertex v in the current separator such that $|P \cap N_G[v]|$ is minimized. This slowed down the algorithm compared to NDCC and we therefore switched to presorting each separator based on their degree in G. We label this algorithm SortSep. Next we considered the order in which the neighbor lists are ordered. This is of importance when trying to dominate a vertex v currently in X. Consider a vertex w with several undominated neighbors in the current separator S. In the configuration where w is in the current MIS all neighbors of w will be dominated, thus reducing the number of undominated vertices in S. In the configuration where w is in X each undominated neighbor of w will have one vertex less that must be tried to dominate it. Based on these observations we implemented a version (SortAdl) where the adjacency list of every vertex v was presorted according to the number of neighbors each vertex has in the same separator as v belonged to. We also tried to compute this ordering on the fly using the number of remaining undominated vertices in the current separator but this only increased the running time. In the right plot of Figure 2 we display the relative running time for all four combinations of these approaches. For each graph we report the relative performance compared to the best algorithm for that graph. In all of these implementations we only check if the graph is disconnected if the previous call to CCMIS moved a vertex into the current MIS.

The average distances from the best algorithm was for SortSep + SortAdl 36%, for NDCC 185%, for SortSep 172%,for NDCC + SortAdl 167%. Thus it is clear that sorting both the the separators and the neighbor lists is crucial for performance.

Finally we tried two versions of CCMIS where the selection criterion for which vertex to branch on was strictly based on the degree of the remaining vertices, one where we always selected the vertex of minimum degree and one where we selected the vertex of maximum degree (MaxDegCC). Both of these were considerably slower than any of the other CCMIS variations. The absolute running times for MaxDegCC,

Table 2. CPU time(sec) for benchmark real world graphs from TreewidthLIB [19]

Graph	1	2	3	4	5	6	7	8	9	10	11
TomitaMIS	0.07	0.22	0.02	0.25	0.03	0.14	0.31	0.69	1.09	29.05	20095.5
NDCC	0.01	0.08	0.02	0.26	0.04	0.09	0.11	0.13	1.04	1.03	0.06
MaxDegCC	0.02	0.23	0.02	0.46	0.05	0.14	0.15	0.09	1.59	0.31	4.56
SortSep+SortAdl	0.01	0.05	0.01	0.22	0.03	0.07	0.06	0.24	0.73	0.22	0.06
Graph	12	13	14	15	16	17	18	19	20	21	22
TomitaMIS	10648.1	16.24	61.5	87.85	-	32716.1	2722.0	-	23.81	32.66	135407.1
NDCC	0.38	8.7	16.4	84.38	3.56	1.18	0.03	187.63	0.06	0.06	1303.0
MaxDegCC	8.67	15.3	40.9	82.06	7.17	0.69	0.18	-	0.84	0.86	1658.85
SortSep+SortAdl	0.07	5.7	8.7	24.37	0.81	0.2	0.04	290.57	0.22	0.22	76.12

TOMITAMIS, NDCC, and SortSEp+SortAdl are given in Table 2. We note that the average distance from the best algorithm for each graph was for MaxDegCC 1371% and for TOMITAMIS 1.6×10^6%.

As can be seen the running time of TOMITAMIS is by far the highest, for some graphs the algorithm did not finish. Also, following a nested dissection ordering is advantageous in most cases, and as already noted presorting the separators and neighbor lists further emphasizes this effect.

We have also experimented with how often one should check if the graph is disconnected in CCMIS. We tried version where we only checked for a certain percentage of the calls, where we only checked once a separator had been dominated, and checking when the remaining graph is at least of some predefined size. From these tests we conclude that when the remaining graph has at least 10 vertices, then checking every time after some vertex has been added be in the current MIS was the best option.

6 Conclusion

We have shown the first practical algorithm for counting MISs in moderately sized sparse graphs. Comparisons with other algorithms showed that our algorithm is highly competitive for this problem. One can get an indication of how good the algorithm is likely to be by looking at the height of the elimination tree. These results also extend to counting cliques in dense graphs. We note that searching for a (small) separator in a graph is equivalent to searching for a (large) complete r-partite graph, $r \geq 2$, in its complement graph. For $r = 2$ this is equivalent to searching for a (not necessarily induced) bi-clique.

We are currently working on implementing the algorithm by Gaspers et al. in C to be able to perform a more complete comparison of it with the other algorithm, although we do not expect that this will change any of our conclusions. We would also like to experiment further with what impact the partitioning strategy has on the running time.

Finally, we note that the presented ideas could be used to compute a maximum independent set in a graph in a similar fashion as the algorithm by Lipton and Tarjan. We are not aware of any practical studies of how to solve this problem on sparse graphs, although the complement problem of finding the maximum size clique has been studied extensively on sparse graphs.

References

1. Bron, C., Kerbosch, J.: Finding all cliques of an undirected graph. Com. ACM 16, 575–577 (1973)
2. Cheng, J., Ke, Y., Fuu, W., Xu Yu, J.: Finding maximal cliques in massive networks. ACM Trans. Database Syst. 36(4), 21:1 – 21:34 (2011)
3. Cheng, J., Zhu, L., Ke, Y., Chu, S.: Fast algorithms for maximal clique enumeration with limited memory. In: Proc. of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2012, pp. 1240–1248. ACM, New York (2012)
4. Eisenstat, S.C., Liu, J.W.H.: The theory of elimination trees for sparse unsymmetric matrices. SIAM J. Mat. Anal. App. 26(3), 686–705 (2005)
5. Eppstein, D.: All maximal independent sets and dynamic dominance for sparse graphs. ACM Trans. on Alg. 5 (2009)
6. Eppstein, D., Löffler, M., Strash, D.: Listing all maximal cliques in sparse graphs in near-optimal time. In: Cheong, O., Chwa, K.-Y., Park, K. (eds.) ISAAC 2010, Part I. LNCS, vol. 6506, pp. 403–414. Springer, Heidelberg (2010)
7. Eppstein, D., Strash, D.: Listing all maximal cliques in large sparse real-world graphs. In: Pardalos, P.M., Rebennack, S. (eds.) SEA 2011. LNCS, vol. 6630, pp. 364–375. Springer, Heidelberg (2011)
8. Gaspers, S., Kratsch, D., Liedloff, M.: On independent sets and bicliques in graphs. Journal of Graph Theoritic Concepts in Computer Science, 171–182 (2008)
9. George, A.: Nested dissection of a regular finite element mesh. SIAM J. on Num. Anal. 10(2), 345–363 (1973)
10. Johnson, D.S., Yannakakis, M., Papadimitriou, C.H.: On generating all maximal independent sets. Information Processing Letters 27, 119–123 (1988)
11. Junosza-Szaniawski, K., Tuczyński, M.: Counting maximal independent sets in subcubic graphs. In: Bieliková, M., Friedrich, G., Gottlob, G., Katzenbeisser, S., Turán, G. (eds.) SOFSEM 2012. LNCS, vol. 7147, pp. 325–336. Springer, Heidelberg (2012)
12. Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G.: Generating all maximal independent sets: NP-hardness and polynomial time algorithms. SIAM J. Comp. 9, 558–565 (1980)
13. Lipton, R.J., Tarjan, R.E.: Applications of a planar separator theorem. SIAM J. Comp. 9(3), 615–627 (1980)
14. Loukakis, E., Tsouros, C.: A depth first search algorithm to generate the family of maximal independent sets of a graph lexicographically. Computing 4, 349–366 (1981)
15. Metis - serial graph partitioning and fill-reducing matrix ordering, `http://glaros.dtc.umn.edu/gkhome/views/metis/`
16. Moon, J.W., Moser, L.: On cliques in graphs. Israel J. of Math., 23–28 (1965)
17. Okamoto, Y., Uno, T., Uehara, R.: Linear-time counting algorithms for independent sets in chordal graphs. In: Kratsch, D. (ed.) WG 2005. LNCS, vol. 3787, pp. 433–444. Springer, Heidelberg (2005)
18. Tomita, E., Tanaka, A., Takahashi, H.: The worst-case time complexity for generating all maximal cliques and computational experiments. Theor. Comput. Sci. 363, 28–42 (2006)
19. Treewidthlib (2004-.), `http://www.cs.uu.nl/people/hansb/treewidthlib`
20. Tsukiyama, S., Ide, M., Ariyoshi, H., Shirakawa, I.: A new algorithm for generating all maximal independent sets. SIAM J. Comp. 6, 505–517 (1977)
21. Wilf, H.S.: The number of maximal independent sets in a tree. SIAM J. Alg. Disc. Meth. 7(1), 125–130 (1986)