Information Processing Letters 27 (1988) 119–123
North-Holland

25 March 1988

ON GENERATING ALL MAXIMAL INDEPENDENT SETS

David S. JOHNSON and Mihalis YANNAKAKIS

AT&T Bell Laboratories, Murray Hill, NJ 07974, U.S.A.

Christos H. PAPADIMITRIOU

Department of Computer Science, Stanford University, Stanford, CA 94305, U.S.A.

Communicated by A.V. Aho
Received 15 September 1986
Revised 24 August 1987

We present an algorithm that generates all maximal independent sets of a graph in lexicographic order, with only polynomial delay between the output of two successive independent sets. We also show that there is no polynomial-delay algorithm for generating all maximal independent sets in *reverse* lexicographic order, unless P = NP.

Keywords: Independent set, enumeration, NP-complete, lexicographic order, polynomial time

Generating all configurations that satisfy a given specification (e.g., all permutations of n objects that do not fix any object) is a well-studied problem in combinatorics [6]. Graph theory suggests many interesting problems of this type (see, e.g., [7]). Among them, generating all maximal independent sets of a given graph is one that has attracted considerable attention in the past [4,5,8]. (A *maximal independent set* of a graph $G=(V, E)$ is a subset $V' \subseteq V$ of the vertices such that no two vertices in V' are joined by an edge in E, and such that each vertex in $V-V'$ is joined by an edge to some vertex in V'.)

One has to be careful in defining notions of 'performance' or 'complexity' of such algorithms. In most interesting problems of this sort, the number of configurations to be generated is potentially *exponential* in the size of the input (say, a graph), and our notions of performance must take this into account. Even so, there are many different notions of what it means to solve a problem of generating configurations 'in polynomial time'. We examine several possibilities below:

(a) *Polynomial total time*. The least that we could ask is that the time required to output all configurations be bounded by a polynomial in n (the size of the input) and C (the number of configurations). Several algorithms satisfying this criterion exist for the problem at hand. The algorithm of Paull and Unger in [5], for example, runs for $O(n^2C)$ time with no output, and then generates all C maximal independent sets in rapid succession. Note that even this weakest notion of 'polynomial time' is not always possible: If the configurations to be generated have some inherent complexity (e.g., the *maximum* independent sets of a graph, or the satisfying truth assignments of a Boolean formula), then, of course, no polynomial total time algorithm exists for generating all of them unless P = NP [4].

(b) *Incremental polynomial time*. An algorithm meeting this criterion can, given an input and several configurations (say, a graph and a collection of maximal independent sets), find another configuration, or determine that none exists, in time polynomial in the combined sizes of the input and the given configurations. It is not hard to see that if such an algorithm exists, then the set of all configurations can be generated in polynomial *total* time (assuming that each configuration is of polynomial size). Thus (b) implies (a). A notion much stronger than either (a) or (b) is the following.

(c) *Polynomial delay.* An algorithm meeting this criterion generates the configurations, one after the other in some order, in such a way that the delay until the first is output, and thereafter the delay between any two consecutive configurations, is bounded by a polynomial in the input size. For maximal independent sets there is a clever algorithm doing just this, due to Tsukiyama et al. [8]. This algorithm, a variant of the one in [5], works by doing a depth-first search in a 'dynamic' binary tree with the maximal independent sets as leaves, whose forward and backward edges are constructed as we go. The tree has depth n, the number of vertices. Each vertex at level j is a maximal independent subset J of the first j vertices of the graph. As for its children, there are two cases. Let $\Gamma(v)$ denote the set of vertices that are adjacent to v. If $J \cap \Gamma(j+1) = \emptyset$ (i.e., if J does not contain any vertices adjacent to vertex $j+1$), then J's only child is $J \cup \{j+1\}$. If there are some vertices in the neighborhood $\Gamma(j+1)$ of $j+1$ in J, then J potentially has two children. The first, or leftchild, is a copy of itself, and is always present. The potential rightchild is $J' = J - \Gamma(j+1) \cup \{j+1\}$ if J' is a maximal independent subset of the first $j+1$ vertices. Note that, in this case, J' is potentially the child of several sets on the same level in the tree, namely, of any maximal independent subset of the first j vertices that contains $J' - \Gamma(j+1)$. Of all these sets, J' is the child of the lexicographically smallest. This completes the description of the tree of independent sets used in the algorithm of [8]. Notice that the functions *leftchild*, *rightchild*, and *parent*, as defined above for this tree, can all be computed in polynomial time. (The lexicographically first maximal independent subset of $\{1, 2, \ldots, j\}$ that contains $J - \Gamma(j+1)$ is easy to compute.) It follows that the tree can be traversed in a depth-first manner with polynomial delay per step of the traversal, and thus the leaves can be output with only a polynomial delay.

(d) *Specified order.* A more complex case is the one in which we wish the configurations output in some specified order, such as lexicographic. Obviously, this matters only in the case of polynomial delay; if we are interested only in polynomial total time, then we can generate all configurations, sort them, and then output them in the desired order. An interesting example of generating configurations in a specified order with polynomial delay is given in [2], which describes such an algorithm for generating all tuples in the Cartesian product of k finite, weighted sets in order of increasing total weight of the k-tuples. Until now, no polynomial-delay algorithm had been known for generating all maximal independent sets of a graph in lexicographic (or any other natural) order.

(e) *Polynomial space.* Some of the algorithms in the above categories, like the algorithm of [5] and that of [2] build exponentially large data structures. Ideally, one would like to avoid this. For example, the algorithm in [8] needs only linear space, since it generates the tree as it goes, and need save only the current node of the tree.

In this paper we examine whether it is possible to generate all maximal independent sets of a given graph in lexicographic order and polynomial delay. (We say a subset S of an ordered set comes lexicographically before T if the first element at which they disagree is in S.) It is not at all obvious that such an algorithm exists, especially in the light of the following result.

1. Theorem. *Given a graph G and a maximal independent set S, it is coNP-complete to tell whether S is the lexicographically **last** maximal independent set of G.*

Proof. The problem is in coNP since if S is not the lexicographically last, a short proof of this fact can be obtained by exhibiting a maximal independent set that follows S. To show completeness, we sketch a polynomial transformation from SATISFIABILITY. Given an instance of satisfiability with variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$, we shall construct a graph G (with ordered vertices) and a maximal independent set S such that there is a maximal independent set T lexicographically after S if and only if there is a truth assignment that satisfies all the clauses. Our graph has a vertex for each clause, a vertex for each literal x_i or $\neg x_i$, and a special vertex a. The order of the vertices is as follows: First the clauses, then a, then the literals (in any order). Vertex a is adjacent to all other vertices in G, and there is an edge connecting any two contradicting literals x_i and $\neg x_i$. Finally, we make each clause adjacent to all literals it contains. These are all edges of G. Let us now define $S = \{a\}$ (clearly a maximal independent set).

Is there a maximal independent set T lexicographically after S? If there is, it must be a subset of the literals, and, since contradicting literals are adjacent, a truth assignment. But is it maximal? For such an independent set to be maximal, it must be the case that, for each clause, there is at least one literal in T which is adjacent to it. In other words, T is a satisfying truth assignment. Therefore, S is the lexicographically last maximal independent set of G if and only if the given Boolean formula is unsatisfiable. □

Notice that it is easy to generate the lexicographically *first* maximal independent subset of a graph: Scan the vertices in the specified order, and never leave out a vertex unless it is adjacent to a vertex already in the set. This can be done in $O(n+m)$ time, where n and m are the numbers of vertices and edges, respectively, in G. Similarly, we can find the lexicographically first independent set that is maximal and contains a given independent set of vertices. (Recall that this fact was used in the proof sketched above that the algorithm of [8] runs with polynomial delay.)

Theorem 1 has interesting consequences, as far as generating algorithms are concerned.

2. Corollary. *It is NP-hard, given a graph G and a maximal independent set S, to generate the lexicographically next maximal independent set.*

3. Corollary. *Unless* P = NP, *there is no algorithm that generates the maximal independent sets of a graph in inverse lexicographic order with polynomial delay.*

Perhaps surprisingly, we now present an algorithm which generates all maximal independent subsets of a graph in lexicographic order, with polynomial delay. Intuitively, our algorithm gets around the negative implications of Corollary 2 by 'investing' work for future outputs while working on the current one. It does use potentially exponential space, in the form of a priority queue Q, which stores a potentially exponential number of maximal independent sets. These sets are inserted into Q by the algorithm at a cost of $O(n \log C)$ per insertion, where n is the number of vertices and C is the total number of maximal independent sets. (If the item to be inserted is already present in the queue, the queue is not altered, although the same time penalty is incurred.) The other operation we can perform on the queue is to find and delete the lexicographically first set it contains; this can be accomplished within the same $O(n \log C)$ time bound. Such a priority queue can be implemented using any of the standard balanced tree schemes (see, e.g., [1]). Note that the comparisons to determine lexicographic priority may take time proportional to n; this is why the time is not simply the depth of the tree, i.e., $O(\log C)$. The algorithm is the following:

```
begin
let $S^*$ be the first maximal independent set of
  $G$;
insert $S^*$ to $Q$;
while $Q$ not empty do
  begin
  $S :=$ min of $Q$;
  output $S$;
  for each vertex $j$ of $G$ adjacent to a vertex
    $i < j$ of $S$ do
   begin
     let $S_j = S \cap \{1, \dots, j\}$;
     if $S_j - \Gamma(j) \cup \{j\}$ is a maximal inde-
        pendent set of the first $j$ vertices then
        begin
        let $T$ be the lexicographically first ma-
           ximal independent set of $G$
           which contains $S_j - \Gamma(j) \cup \{j\}$;
        insert $T$ into $Q$
        end
     end
   end
end
```

4. Theorem. *The algorithm above outputs all maximal independent sets of a graph with n vertices and m edges in lexicographic order, and with* $O(n(m + n\log C)) = O(n^3)$ *delay.*

Proof. Notice that the set T inserted in Q at the time S is output is lexicographically after S. Thus, the queue always gets sets that are lexicographically after the one being output, and therefore the sequence output is indeed lexicographically increasing. We shall show by induction that, if S is the lexicographically first maximal independent set not yet output, then it is already in the queue (and thus it will indeed be output next). This certainly holds when $S = S^*$.

In the generic case, let j be the largest number such that S_j is *not* a maximal independent set of the graph restricted to the first j vertices. (If no such j exists, then it is easy to see that $S = S^*$.) Note that we must have $j < n$ since S is a maximal independent set for the whole graph; note also that, by the maximality of j, we must have $j + 1 \in S$. Enlarge S_j to a maximal independent set $S_j \cup K$ of the first j vertices; since S_j is not maximal, K is nonempty. Moreover, $j + 1$ must be adjacent to all vertices in set K, again because of the maximality of j.

We conclude that there is a maximal independent set S' which contains $S_j \cup K$, but not $j + 1$. However, S' comes lexicographically before S (since it first disagrees with S on the vertices of K). By induction, S' has already been output by the algorithm. When it was output, vertex $j + 1$ was found to be adjacent to a vertex $i < j + 1$ in S'. Since $S'_{j+1} - \Gamma(j + 1) \cup \{j + 1\} = S_{j+1}$ is a maximal independent subset of the first $j + 1$ vertices, the first maximal independent set T of G that contains S_{j+1} was inserted into the queue. T agrees with S on the first $j + 1$ vertices, because S_{j+1} is maximal on that set of vertices. Suppose $S \neq T$, and let k be the first vertex in which they differ; $k > j + 1$. We must have $k \in T$ and $k \notin S$ since T was the lexicographically first maximal independent set containing S_{j+1}. But this would imply that S_k is not maximal, contradicting the maximality of j. It follows that $S = T$, and therefore S is indeed in Q, as claimed. This concludes the proof of correctness.

For the time bound, notice that the generic step of the algorithm consist of an extraction of the lexicographically first maximal independent set from Q (taking $O(n\log C)$ time), followed by at most n calculations of a maximal independent set containing a given one (taking $O(n + m)$ time per set), and for each of these an attempt to insert the set found into the queue (taking time $O(n\log C)$ per set). The total delay is thus $O(n(\log C + n + m + n\log C)) = O(n^3)$. □

Notice that our algorithm uses potentially exponential space (although at most $O(nC)$). If we are willing to tolerate potentially exponential delay, we can generate all maximal independent sets of a graph in lexicographic order using only polynomial space: simply generate all subsets in lexicographic order, outputting only those that are independent and maximal. A remaining open question is whether there is a way to avoid this apparent tradeoff.

Another important open question is whether there is an algorithm for generating in polynomial *total* time the maximal independent sets of a *hypergraph*, not necessarily in lexicographic order. A hypergraph is a set of vertices together with a set of subsets of the vertices, called *hyperedges*. A set of vertices of a hypergraph is *independent* if it does not contain a hyperedge. (Note that this reduces to our standard notion of an independent set in a graph if we specialize to the case when all hyperedges contain precisely two elements.)

This problem has an unexpected application in databases. Consider a set of n distributed sites. A *voting pattern* in a distributed database concurrency control scheme is a hypergraph with these n cites as vertices, in which the following two properties hold: (a) all pairs of hyperedges intersect, and (b) no vertex can be deleted from any hyperedge without violating (a). (This is a restricted version of the 'coterie' of [3].) Intuitively, a voting pattern is a collection of minimal sets of sites, such that the sites in each of these sets can safely agree on a decision (say, an update), without the risk that another set has reached a contradictory decision. The most usual voting pattern contains all sets of $\lceil\frac{1}{2}(n + 1)\rceil$ sites. We say a voting pattern is *maximal* if it cannot be extended to a voting pattern with one more hyperedge. (The maximal voting patterns coincide with the 'nondominated coteries' of [3].) Testing whether a voting pattern is maximal can be shown to be equivalent to deciding whether a proposed list of maximal independent sets of a given hypergraph is complete.

Unfortunately, no algorithm for generating all maximal independent sets of a hypergraph in polynomial total time is known. In particular, the approach of [5,8] will not work. That approach constructs all maximal independent sets of the subgraph induced by $\{1, 2, \ldots, j\}$, and then derives from them the maximal independent sets of the subgraph induced by $\{1, 2, \ldots, j+1\}$. With both graphs and hypergraphs, we know that every maximal independent set I of vertices $\{1, \ldots, j+1\}$ is contained in a set $J \cup \{j+1\}$ for some maximal independent set J of $\{1, 2, \ldots, j\}$. The question of determining which such I are contained in $J \cup \{j+1\}$ for a given J is easy for graphs; the only candidates are J and $J - \Gamma(j+1) \cup \{j+1\}$. For hypergraphs, however, we are not so fortunate. The set of candidates is much larger, and, indeed, the question of whether any such I exists is NP-complete (an exercise we leave to the reader).

References

[1] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms* (Addison-Wesley, Reading, MA, 1974).

[2] A.V. Aho, T. Szymanski and M. Yannakakis, Sorting the Cartesian product, *Proc. Conf. on Information Sciences and Systems* (Princeton University, Princeton, NJ, 1980) 557–558.

[3] H. Garcia-Molina and D. Barbara, How to assign votes in a distributed system, *J. ACM* **32** (1985) 841–860.

[4] E.L. Lawler, J.K. Lenstra and A.H.G. Rinnooy Kan, Generating all maximal independent sets: NP-hardness and polynomial-time algorithms, *SIAM J. Comput.* **9** (1980) 558–565.

[5] M.C. Paull and S.H. Unger, Minimizing the number of states in incompletely specified sequential switching functions, *IRE Trans. Electr. Comput.* **EC-8** (1959) 356–367.

[6] R.C. Read, Every one a winner, or how to avoid isomorphism when cataloguing combinatorial configurations, *Ann. Discrete Math.* **2** (1978) 107–120.

[7] R.C. Read and R.E. Tarjan, Bounds on backtrack algorithms for listing cycles, paths, and spanning trees, *Networks* **5** (1975) 237–252.

[8] S. Tsukiyama, M. Ide, H. Ariyoshi and I. Shirakawa, A new algorithm for generating all maximal independent sets, *SIAM J. Comput.* **6** (1977) 505–517.