Comp. & Maths. with Appls. Vol. 9, No. 4, pp. 583–589, 1983
Printed in Great Britain.

0097-4943/83 $3.00 + .00

A NEW BACKTRACKING ALGORITHM FOR GENERATING THE FAMILY OF MAXIMAL INDEPENDENT SETS OF A GRAPH

E. LOUKAKIS
University of Thessaloniki, School of Engineering, Thessaloniki, Greece

(*Received August* 1982; *in revised form October* 1982)

Communicated by E. Y. Rodin

Abstract—The family of maximal independent sets also known as stable sets or anticliques has a variety of many interesting applications and is fundamental to a class of graph colouring algorithms. In this paper we present a new backtracking algorithm to generate the family of maximal independent sets. A comparative computational study on more than 1000 randomly generated graphs ranging from 30 to 220 vertices and 10% to 90% densities has shown that the proposed algorithm is substantially more efficient than the best algorithms developed so far.

1. INTRODUCTION

Let $G = (V, E)$ be a finite undirected simple graph where $V \neq \emptyset$ is its vertex set and $E \subseteq \{\{u, v\} : u, v \in V, u \neq v\}$ is its edge set. The complement of $G = (V, E)$ is defined as $\bar{G} = (V, \bar{E} = \{\{u, v\} \in V \times V : u \neq v, \{u, v\} \notin E\})$. The adjacency vertex sets of $u \in V$, and $S \subseteq V$ of a graph $G = (V, E)$ are defined as $\mathrm{Adj}(u) = \{v : \{u, v\} \in E\}$ and $\mathrm{Adj}(S) = \cup_{x \in S} \mathrm{Adj}(x)$ respectively.

A set $M \subseteq V$ such that $\mathrm{Adj}(M) \cap M = \emptyset$ or equivalently $\{u, v\} \notin E$, $\forall u, v \in M$ is called independent abbreviated as IS (also known as internally stable or anticlique). An IS, Q such that $\mathrm{Adj}(S) \cap S \neq \emptyset$, $\forall S \supset Q$ is called maximal and is abbreviated as MIS. A set $K \subseteq V$ such that $\{u, v\} \in E$, $\forall u, v \in K$ is called clique. A clique K such that Q is not a clique $\forall Q \supset K$, is called maximal. A set $X \subseteq V$ such that $\mathrm{Adj}(x) \cap X \neq \emptyset$, $\forall x \in V \backslash X$ is called externally stable or vertex covering. A vertex covering T such that S is not a vertex covering $\forall S \subset T$, is called minimal.

From the above definitions it easily follows that the maximal independent sets, maximal cliques, and minimal coverings of a graph $G = (V, E)$ are pairwise complementary in the following sense. A subset $M \subseteq V$ is an MIS of G iff M is a maximal clique of $\bar{G}$. Further, a subset $S \subseteq V$ is a minimal covering of G iff $V \backslash S$ is an MIS of G.

Thus, the problem of generating the family of MIS of a graph G can be reduced to that of generating the family of minimal coverings of G or the family of maximal cliques of $\bar{G}$. The algorithm of this paper to be presented below generates the family of maximal independent sets which we will denote by $\mathcal{M}[G]$. The family of independent sets will be denoted by $\mathcal{I}[G]$. The family $\mathcal{M}[G]$ has a variety of many interesting applications (see, eg. [1–5]). Further, the efficient construction of $\mathcal{M}[G]$, is fundamental to a class of algorithms for determining the chromatic number of a graph, (see, e.g. [6–8]) among others. Various algorithms for generating $\mathcal{M}[G]$ have been devised, (see, e.g. [1, 2, 9–18]). Among the proposed algorithms those in [10, 11, 13, 18] report computational experience. The results of a comparative computational study conducted in [13] has shown: (i) the algorithm of Tsukiyama *et al.* [18] called TIAS is slightly superior to that of Bron and Kerbosch [10] for graphs of low density. However, TIAS algorithm rapidly deteorates as both the size and the density of the graph increase and (ii) the algorithm of Loukakis and Tsouros [13] called LTMIS is at least three times faster than the Bron and Kerbosch (B & K) algorithm.

In this paper we present a new algorithm which we call LMIS to generate the family $\mathcal{M}[G]$. Extensive computational experience has shown that LMIS is at least three times faster than LTMIS. The remaining parts of the paper are organized as follows: In Section 2 we develop the algorithm LMIS and illustrate its workings with an example. In Section 3 we present the data structure employed in the implementation of LMIS. The results of

computational experience are presented in Section 4. Finally, in Section 5 we examine the main differences between the *B & K* algorithm and LMIS.

2. DEVELOPMENT OF THE ALGORITHM

Algorithm LMIS belongs to the class of backtracking algorithms. The state of the search at any stage of computations and its associated node of the search tree is completely specified by the ordered triple $W=(W^+, W^-, \tilde{W})$ of pairwise disjoint subsets of V. Specifically, W^+ is an independent set. W^- is the vertex set the elements of which were used earlier in the search and are therefore explicitly excluded from every MIS containing W^+. Formally we have, $\forall u \in W^-$, $\exists \hat{W} \in \mathscr{T}[G]$ such that $W^- \subseteq \hat{W}$ and every MIS containing $\hat{W} \cup \{u\}$ has been generated. Alternatively, the search from the node of the search tree specified by W generates the maximal independent sets which contain every vertex of W^+ and none of W^-. Finally, $\tilde{W} = V \backslash (W^- \cup W^- \cup \mathrm{Adj}(W^-))$ is the set of candidate vertices. i.e. $W^+ \cup \{u\} \in \mathscr{T}[G]$, $\forall u \in \tilde{W}$.

Branching chooses a vertex say $u \in \tilde{W}$ and sets: $W^+ \leftarrow W^+ \cup \{u\}$ and $\tilde{W} \leftarrow \tilde{W} \backslash (\{u\} \cup (\mathrm{Adj}(u) \cap \tilde{W}))$.

Backtracking removes the most recently introduced vertex in W^-, say v, places it in W^- and revises $\tilde{W}$ by setting:

$$\tilde{W} \leftarrow \tilde{W} \cup (\mathrm{Adj}(v) \cap \tilde{W}).$$

Finally, the state of W^- and $\tilde{W}$ determines whether W^+ is or is not an MIS. Specifically we consider the following four possible cases.

(i) $W^- \not\subseteq \mathrm{Adj}(W^+)$ and $\tilde{W} = \emptyset$. By definition $W^+ \cap W^- = \emptyset$. Thus, $W^+ \cap \mathrm{Adj}(u) = \emptyset$ for some $u \in W^- \backslash \mathrm{Adj}(W^+)$ implying $W^+ \cup \{u\} \in \mathscr{T}[G]$ and $W^+ \notin \mathscr{M}[G]$.

(ii) $W^- \subseteq \mathrm{Adj}(W^+)$ and $\tilde{W} \neq \emptyset$. By definition of W^+ and $\tilde{W}$ we have $\mathrm{Adj}(\tilde{W}) \cap W^+ = \emptyset$. Thus, $\tilde{W} \neq \emptyset \Rightarrow W^+ \cup \{x\} \in \mathscr{T}[G]$ for some $x \in \tilde{W}$ implying $W^+ \notin \mathscr{M}[G]$.

(iii) $W^- \not\subseteq \mathrm{Adj}(W^+)$ and $\tilde{W} \neq \emptyset$. In this case either (i) or (ii) apply and $W^+ \notin \mathscr{M}[G]$.

(iv) $W^- \subseteq \mathrm{Adj}(W^+)$ and $\tilde{W} = \emptyset$. By definition of W^+, W^- and $\tilde{W}$ we have $V = W^+ \cup \mathrm{Adj}(W^+) \cup W^- \cup \tilde{W}$ or $V = W^+ \cup \mathrm{Adj}(W^+)$ by $W^- \subseteq \mathrm{adj}(W^+)$ and $\tilde{W} = \emptyset$. This means that for $\hat{W} = W^+ \cup K$ we have $K \subseteq \mathrm{Adj}(W^+)$ implying $\mathrm{Adj}(\hat{W}) \cap \hat{W} \neq \emptyset$, $\forall \hat{W} \supset W^+$ and therefore $W^+ \in \mathscr{M}[G]$.

Cases (i), (ii), (iii) and (iv) prove the following theorem

THEOREM 1

Let $W=(W^+, W^-, \tilde{W})$. Then $W^+ \in \mathscr{M}[G]$ if and only if $W^- \subseteq \mathrm{Adj}(W^+)$ and $\tilde{W} = \emptyset$.

A vertex $u \in W^-$ will be called *closed* or *open* according to whether we have $u \in \mathrm{Adj}(W^+)$ or $u \notin \mathrm{Adj}(W^+)$ respectively. The above theorem states that for the triple $W=(W^+, W^-, \tilde{W})$, W^+ is an MIS iff $\tilde{W} = \emptyset$ and every vertex of W^- is closed. As an immediate consequence of theorem 1 we have the following bounding test.

BT. Let $W=(W^+, W^-, \tilde{W})$, $W^- \subseteq \mathrm{Adj}(W^+)$ and $u \in \tilde{W}$. If $\mathrm{Adc}(u) \cap \tilde{W} = \emptyset$ or equivallently $\mathrm{Adj}(u) c\, \mathrm{Adj}(W^+) \cup W^-$, then repeated branching from the node of the search tree specified by $\hat{W} = (W^+, W^- \cup \{u\}, \tilde{W} \backslash \{u\})$ will lead to $\tilde{W} = \emptyset$ and leave $u \in W^-$ open. Thus, according to theorem 1 any branching from the node $\hat{W}$ cannot produce an MIS. This means that, if $\mathrm{Adj}(u) \cap \tilde{W} = \emptyset$ then any MIS which contains W^+ must contain u. Equivalently, let $W=(W^+, W^-, \tilde{W})$ and $W^- \backslash \mathrm{Adj}(W^+) = \{x\}$. If $\mathrm{Adj}(x) \cap \tilde{W} = \{y\}$ then any branching from $(W^+, W^- \cup \{y\}, \tilde{W} \backslash \{y\})$ will leave $x \in W^-$ open and therefore y is forced to be in every MIS which contains W^+.

The structure of LMIS is also based on theorem 1 and can be described as follows. Since $W^+ \cap W^- = \emptyset$ we employ a set M to accomodate W^+ and W^- with the vertices of W^+ and W^- signed positive and negative respectively. If $W^- \subseteq \mathrm{Adj}(W^+)$ then the algorithm generates an MIS by repeated branching from the elements of $\tilde{W}$ until $\tilde{W} = \emptyset$. Further, when branching on, say, $u \in \tilde{W}$ and the conditions of *BT* are satisfied u is marked. Backtracking finds the rightmost unmarked positive vertex of M, negates it and deletes every element added after it. When backtracking on say, v, vertex v and the vertices of

W^- closed by v (if any) are becoming open. The process then, closes the open vertices by branching on vertices of $\tilde{W}$ adjacent to them, and returns to repeated branching to generate a new MIS. Finally, when the first (leftmost) element of M, say, x is open and cannot be closed the family $\mathcal{M}[G]$ has been generated and the process terminates. This happens when x and every vertex of $\mathrm{Adj}(x)$ becomes negative i.e. $\{x\} \cup \mathrm{Adj}(x) \subseteq W^-$.

We next present a complete statement of LMIS which formalizes the above discussion.

Algorithm LMIS

$$t = \begin{cases} 0 & \text{if } W^- \subseteq \mathrm{Adj}(W^+) \\ 1 & \text{otherwise} \end{cases}$$

$M(j)$ = The vertex positionioned in the jth place from left to right in M.

Step 1 (Initialize). Set $M = \emptyset$, $\tilde{W} = V$, $t = 0$, choose u such that $|\mathrm{Adj}(u)| = \min\{|\mathrm{Adj}(j)|: j = 1, \ldots, |V|\}$ and go to 3.

Step 2 (Test for MIS). If $\tilde{W} = \emptyset$ record $M \setminus \{j: -j \in M\}$ as a new MIS and go to 4. Otherwise set $u \leftarrow \tilde{W}(1)$ and proceed to 3.

Step 3 (Apply BT). If $\mathrm{Adj}(u) \cap \tilde{W} = \emptyset$ mark u by setting $u \leftarrow u^*$. Proceed to 4.

Step 4 (Branch). Set $M \leftarrow M \cup \{u\}$ and $\tilde{W} \leftarrow \tilde{W} \setminus (\{u\} \cup (\mathrm{Adj}(u) \cap \tilde{W}))$. If $t = 1$ go to 7. Otherwise go to 2.

Step 5 (Backtrack, Terminate). Denote by v the rightmost unmarked element of M and define.

$R(v) = \{j: j \in M$ or $-j \in M$ and j or $-j$ is right to $v\}$.

Set $M \leftarrow M \setminus R(v)$, $v \leftarrow -v$ in M, $\tilde{W} \leftarrow \tilde{W} \cup (R(v) \cup (\mathrm{Adj}(v) \cap \tilde{W}))$ and construct the set of open vertices.

$Z = \{v\} \cup \{j: -j \in M$ and $j \in \mathrm{Adj}(v)\}$. Let $x = M(1)$.

If $\{x\} \cup \mathrm{Adj}(x) \subseteq Z$ terminate. Otherwise proceed to 6.

Step 6 (Close the vertices of Z). Set $k \leftarrow Z(1)$. If $\mathrm{Adj}(k) \cap \tilde{W} = \emptyset$ go to 5. Otherwise set $u \leftarrow a$ vertex of $\mathrm{Adj}(k) \cap \tilde{W}$, $t \leftarrow 1$ and go to 3.

Table 1.

Steps	M	$\tilde{W}$	MIS
1	$\emptyset$	V	-
3,4,2	{2}	{1,3,6,7}	-
3,4,2	{2,1}	{6}	-
3,4,2	{2,1,6*}	$\emptyset$	{2,1,6}
5	{2,-1}	{3,6,7}	-
6,3,7,2	{2,-1,3*}	{6,7}	-
3,4,2	{2,-1,3*,6}	$\emptyset$	{2,3,6}
5	{2,-1,3*,-6}	{7}	-
6,3,7,2	{2,-1,3*,-6,7*}	$\emptyset$	{2,3,7}
5	{-2}	{1,3,4,5,6,7,8}	-
6,3,7,2	{-2,4}	{1,3,5,8}	-
3,4,2	{-2,4,1}	$\emptyset$	{4,1}
5	{-2,4,-1}	{3,5,8}	-
6,3,7,2	{-2,4,-1,3}	$\emptyset$	{4,3}
5	{-2,4,-1,-3}	{5,8}	-
6,3,7,2	{-2,4,-1,-3,5*}	{8}	-
3,4,2	{-2,4,-1,-3,5*,8*}	$\emptyset$	{4,5,8}
5	{-2,-4}	{1,3,5,6,7,8}	-
6,3,7	{-2,-4,5}	{1,8}	-
6,5	{-2,-4,-5}	{1,3,6,7,8}	-
6,3,7	{-2,-4,-5,8}	{1,7}	-
6,3,7,2	{-2,-4,-5,8,7}	$\emptyset$	{8,7}
5	{-2,-4,-5,8,-7}	{1}	-
6,5	{-2,-4,-5,-8}	Terminate since {2}∪Adj(2)⊆	

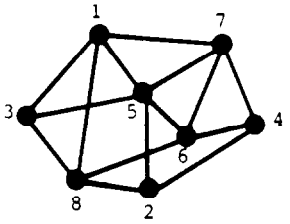

Fig. 1. Adj (1) = {3, 5, 7, 8}; Adj (2) = {4, 5, 8}; Adj (3) = {1, 5, 8}; Adj (4) = {2, 6, 7}; Adj (5) = {1, 2, 3, 6, 7}; Adj (6) = {4, 5, 7, 8}; Adj (7) = {1, 4, 5, 6}; Adj (8) = {1, 2, 3, 6}.

Step 7 (Revise Z). Set $Z \leftarrow Z \setminus (\{k\} \cup (\text{Adj}(u) \cap Z))$. If $Z \neq \emptyset$ go to 6. Otherwise set $t \leftarrow 0$ and go to 2.

Example. Table 1 above sumarises the workings of LMIS when applied to the graph $G = (V, E)$ of Fig. 1 above with its adjacency lists.

3. DATA STRUCTURE

Besides the set M the implementation of LMIS employs the following: The set B which has the same cardinality as M and $B(|M|)$ is the position of the rightmost unmarked element of M i.e. $v = M(B(|M|))$ is the vertex on which backtracking takes places. B is revised by setting:

$$B \leftarrow \begin{cases} B \cup \{B(|B| - 1)\} & \text{When backtracking or when branching and the conditions of BT are satisfied} \\ B \cup \{|M|\} & \text{When branching and conditions of BT are not satisfied.} \end{cases}$$

The state of $\tilde{W}$ is specified by a discrete function $F: V \to \{0, 1\}$ such that

$$F(u) = \begin{cases} 1 & \text{if } u \in W^+ \cup W^- \cup \text{Adj}(W^+) \\ 0 & \text{if } u \in \tilde{W}. \end{cases}$$

When branching on vertex $u \in \tilde{W}$; F is revised by setting

$$F(x) \leftarrow 1, \forall x \in (\text{Adj}(u) \cap \tilde{W}) \cup \{u\}.$$

When backtracking on vertex v, F is revised by setting.

$$F(x) \leftarrow 0, \forall x \in (\text{Adj}(v) \cap \tilde{W}) \cup R(v)$$

$R(v)$ as defined in the previous section.

With each $u \in W^+$ we associate the set $\text{AFW}(u) = \text{Adj}(u) \cap \tilde{W}$. Further since for $u \in W^+$, $\tilde{W}$ is revised by setting $\tilde{W} \leftarrow \tilde{W} \setminus (\{u\} \cup \text{AFW}(u))$ we have $\text{AFW}(x) \cap \text{AFW}(y) = \emptyset$, $\forall x, y \in W^+$ and $V \supseteq \cup_{u \in W^+} \text{AFW}(u)$. Thus, the sets $\text{AFW}(x)$, $x \in W^+$ are packed in one dimensional array of length $|V|$ and are separated by an array of pointers. Further, for $u \in W^+$ the construction of $\text{AFW}(u)$ which requires $|\text{Adj}(u)|$ logical comparisons and the updating of F which sets $F(x) \leftarrow 1$, $\forall x \in \text{AFW}(u) \cup \{u\}$, are carried out within the same loop with computational effort of $0(|\text{Adj}(u)|)$ time. A further advantage of this data structure is that the application of BT which tests whether or not $\text{AFW}(x) = \emptyset$ for $x \in W^+$ requires one comparison only. Finally, if $W^- \subseteq \text{Adj}(W^+)$, the time complexity to generate an MIS, say S, is

$$\sum_{x \in S} 0(|\text{Adj}(x)|) = 0(|E|).$$

We also employ a predecessor function $P: V \to \{0, 1, 2, \ldots, |M|\}$ such that

$$P(u) = \begin{cases} k \text{ if } M(k) = -v \text{ and } v \text{ is closed by } u \\ t \text{ if } M(t) = u \text{ and } u \text{ does not close any } v \in W^- \\ 0 \text{ if } u \in W^+. \end{cases}$$

The function P is used to detect the vertices which become open when the process backtracks. Specifically backtracking on vertex x we distinguish the following two cases.

(1) $P(x) = |M|$. Then vertex x becomes open only and is closed by the first vertex of AFW(x) which augments M.

(2) $P(x) < |M|$. Then at least x and $M(P(x))$ become open. Specifically the negative vertices $y \notin \text{Adj}(W^+)$ of M located between its $P(x)$ and $|M|$ position. Formally the set of open vertices Z is defined as

$$Z = \{y : M(j) = -y, P(x) \le j \le |M| \quad \text{and} \quad y \notin \text{Adj}(W^+)\}.$$

To store the graphs the adjacency lists were packed into a linear array called ADJ of $2|E|$ length, and were separated by a list of pointers called LINK of $|V| + 1$ length. The elements of LINK satisfy the following recursion.

$$\text{LINK}(j) = \begin{cases} 0 \quad \text{for } j = 1 \\ \\ \text{LINK}(j-1) + |\text{Adj}(j-1)| = \sum_{i=1}^{j-1} |\text{Adj}(i)|, \quad \text{for } j > 1. \end{cases}$$

with this data structure the adjacent vertices of, say, vertex u are located between the cells ADJ(P1) and ADJ(P2) where $\text{P1} = \text{LINK}(u) + 1$ and $\text{P2} = \text{LINK}(u + 1)$. Thus, the memory requirements to store the graph $G = (V, E)$ were $2|E| + |V| + 1$ computer words.

4. COMPUTATIONAL EXPERIENCE

This section presents the results of a comparative computational study in which LMIS was tested against LTMIS and TIAS. The computational study is based on over 1000 randomly generated graphs ranging from 30 to 220 vertices and from 10% to 90% densities, and the results are sumarised in Tables 2 and 3. We employed the test problems generator

Table 2.

D	|V|	MIS	LMIS	LTMIS	TIAS	D	|V|	MIS	LMIS	LTMIS	TIAS
.10	30	526	.44	1.38	1.70	.50	60	1946	1.78	9.64	171.18
.10	40	1280	1.27	4.65	5.12	.50	70	3515	5.27	22.13	*
.10	50	23654	28.16	89.68	122.64	.50	80	6127	11.53	44.58	*
.20	30	365	.26	.82	1.93	.50	90	10229	30.18	117.42	*
.20	40	2762	2.14	6.48	17.96	.50	100	16204	76.42	231.29	*
.20	50	15627	14.61	48.22	170.38	.50	110	25130	115.27	368.48	*
.20	60	62464	72.86	249.64	*	.60	30	105	.04	.23	4.82
.30	30	298	.18	.66	3.55	.60	40	250	.13	.65	20.17
.30	40	1354	1.30	4.18	25.34	.60	50	514	.56	1.82	67.81
.30	50	5167	4.22	18.11	126.18	.60	60	870	.89	3.86	190.42
.30	60	15102	15.37	69.74	452.84	.60	70	1381	2.18	7.12	460.88
.30	70	36878	82.13	271.36	*	.60	80	2360	4.53	18.85	*
.40	30	259	.22	.65	3.18	.60	90	3604	5.87	39.26	*
.40	40	815	.77	2.68	22.34	.60	100	5368	11.32	50.74	*
.40	50	2142	2.34	8.31	75.42	.60	110	7572	19.38	84.17	*
.40	60	4815	5.19	24.79	225.87	.60	120	10398	86.45	272.58	*
.40	70	10125	16.45	53.25	*	.70	30	82	.04	.17	2.83
.40	80	20874	42.56	120.64	*	.70	40	158	.11	.51	9.15
.40	90	37272	98.47	318.36	*	.70	50	267	.28	1.22	24.36
.50	30	170	.08	.45	4.25	.70	60	440	.51	1.93	71.48
.50	40	430	.46	1.53	21.34	.70	70	680	.96	3.84	153.27
.50	50	938	.94	3.71	81.62	.70	80	1029	1.48	6.72	291.45

D = density of the graph = 2|E|/|V|(|V|-1)

Table 3.

D	|V|	MIS	LMIS	LTMIS	TIAS	D	|V|	MIS	LMIS	LTMIS	TIAS
.70	90	1512	2.82	15.26	*	.80	200	4812	24.48	96.42	*
.70	100	1984	4.17	19.48	*	.80	210	5258	38.52	120.84	*
.70	110	2736	5.48	28.15	*	.80	220	5845	43.56	150.20	*
.70	120	3618	8.04	35.08	*	.90	40	60	.03	.16	6.38
.70	130	4656	10.38	52.67	*	.90	50	106	.08	.31	14.18
.80	30	62	.03	.14	2.43	.90	60	130	.13	.42	31.45
.80	40	89	.07	.28	8.25	.90	70	182	.24	.87	62.10
.80	50	182	.16	.69	26.38	.90	80	211	.47	1.10	85.47
.80	60	250	.41	1.18	49.64	.90	90	288	.81	1.76	163.25
.80	70	342	.72	1.60	91.25	.90	100	352	1.20	3.88	*
.80	80	438	.98	2.56	180.30	.90	110	413	1.48	4.76	*
.80	90	661	1.14	4.84	322.59	.90	120	500	1.86	6.31	*
.80	100	845	2.08	7.93	*	.90	130	568	2.18	7.88	*
.80	110	1017	2.76	9.28	*	.90	140	620	2.70	9.14	*
.80	120	1326	3.95	15.10	*	.90	150	722	3.25	11.37	*
.80	130	1741	5.12	21.42	*	.90	160	867	4.38	14.02	*
.80	140	2004	6.25	23.50	*	.90	170	961	5.04	15.13	*
.80	150	2392	8.06	29.36	*	.90	180	1120	6.19	19.26	*
.80	160	2741	9.28	36.71	*	.90	190	1270	6.93	21.50	*
.80	170	3268	11.70	58.25	*	.90	200	1458	7.27	24.08	*
.80	180	3690	12.49	64.77	*	.90	210	1642	8.06	28.45	*
.80	190	4186	18.83	81.25	*	.90	220	2405	10.32	43.30	*

D = density of the graph = 2|E|/|V|(|V|-1)

of [13] and the random numbers were drawn from a normal distribution. The test problems were run on a UNIVAC 1106 computer of the Aristotelean University of Thessaloniki and were compiled by a FORTRAN V compiler. As performance indicator we have chosen CPU time. The reported CPU times are in seconds and were obtained by calling a real time clock routine available in UNIVAC software, upon starting to solve the problem and again upon termination. Each entry is the average CPU time of a set of 10 problems. The entries marked by asterisks correspond to graphs which consume excessive time and due to computer budget limitations we have not been able to solve them.

Before starting LTMIS and LMIS the vertices of the graphs were ordered as follows:

$$i < j \text{ whenever } |\mathrm{Adj}(i)| \geqslant |\mathrm{Adj}(j)| \text{ for LTMIS}$$

$$i < j \text{ whenever } |\mathrm{Adj}(i)| \leqslant |\mathrm{Adj}(j) \text{ for LMIS.}$$

To relabel the vertices of the graphs we developed the subroutine GISOM which constructs a graph $G' = (V', E')$ isomorphic to $G = (V, E)$. The time taken by GISOM is included in the reported times. LMIS has also been tested on Moon–Moser[19] graphs and it was found to be twice a fast as LTMIS and the per MIS time was constant.

Listings of the computer programs of LMIS, LTMIS, TIAS and GISOM can be obtained from the author on request.

5. BRON AND KERBOSCH AND LMIS ALGORITHMS COMPARED

The B & K algorithm and LMIS belong to the class of branch and bound algorithms. Thus, the differences between the two algorithms need be found in the way in which the operations of branching, backtracking and bounding comprising any branch and bound algorithm are conducted. A further factor the significant importance of which is widely recognized is the computer implementation of the algorithms having as a basic ingredient the data structure employed.

B & K algorithm chooses a vertex $u \in \tilde{W}$ to branch on by determining x^* such that

$$|\mathrm{Adj}(x^*) \cap \tilde{W}| = \min \{|\mathrm{Adj}(x) \cap \tilde{W}| : x \in \tilde{W} \cup W^-\} \quad (*)$$

and setting:

$$u \leftarrow \begin{cases} x^* \text{ if } x^* \in \tilde{W} \\ y \in \text{Adj}(x^*) \cap \tilde{W} \text{ if } x^* \in W^-. \end{cases}$$

Instead LMIS branches first on vertices of $\tilde{W}$ which are adjacent to open vertices (if any) of W^- to close them, and then on the first vertex of $\tilde{W}$. Obviously the computational effort to choose the vertex x^* according to (*) is substantial and increases as both the density and size of the graph increase. This is not the case with LMIS where the computational effort to select the vertex to branch on is negligible and independent of the density and size of the graph. Further, within the branching operation of LMIS the bounding test BT is applied with computational effort of one comparison only, while this test is not employed by the B & K algorithm. In the B & K algorithm we may reach a state where $\tilde{W} = \emptyset$ and $W^- \not\subseteq \text{Adj}(W^+)$ implying $W^+ \notin \mathcal{M}[G]$. However, this cannot happen with LMIS. This is due to the differences in branching and bounding schemes employed by the two algorithms. Thus, LMIS does less fruitless search than the B & K algorithm.

Finally, LMIS dominates the B & K algorithm in terms of computer working memory requirements. Specifically, B & K algorithm requires $0(|V|^2)$ computer working memory while LMIS requires $0(|V|)$ only. This is due to the differences in the search scheme and data structure employed by the two algorithms.

Acknowledgements—The constructive comments of the referee have greatly improved this paper and are greatfully acknowledged.

REFERENCES

1. J. G. Auguston and J. Minker, An analysis of some graph theoretical cluster techniques. *JACM* **17**, 571–588 (1970).
2. C. Berge, *The Theory of Graphs and its Applications*. Wiley, New York (1962).
3. M. A. Breuer, *Design Automation of Digital Systems, Vol. 1, Theory and Techniques*. Prentice-Hall, Englewood Cliffs, New Jersey (1973).
4. N. Deo, *Graph Theory with Applications to Engineering and Computer Science*. Prentice-Hall, Englewood Cliffs, New Jersey (1974).
5. N. Jardine and R. Sibson, *Mathematical Taxonomy*. Wiley, London (1971).
6. N. Cristofides, An algorithm for the chromatic number of a graph. *Computer. J.* **14**, 38–39 (1971).
7. E. L. Lawler, A note on the complexity of the chromatic number problem. *Inform. Proc. Lett.* **5**, 66–67 (1975).
8. C. C. Wang, An algorithm for the chromatic number of a graph. *JACM* **21**, 385–391 (1974).
9. E. A. Akkoyunlu, The enumeration of maximal cliques of large graphs. *SIAM J. Comput.* **2**, 1–6 (1973).
10. C. Bron and J. Kerbosch, Finding all cliques of an undirected graph: Algorithm 457. *Comm. ACM.* **16**, 575–577 (1973).
11. L. Gerhards and W. Lindenberg, Clique detection for nondirected graphs: Two new algorithms. *Computing* **21**, 295–322 (1979).
12. E.L. Lawler, J.K. Lenstra and H.G. Rinnoy Kan, Generating all maximal independent sets: NP-Hardness and polynomial-time algorithms. *SIAM J. Comput.* **9**, 558–565 (1980).
13. E. Loukakis and C. Tsouros, A depth first search algorithm to generate the family of maximal independent sets of a graph lexicographically. *Computing* **27**, 349–366 (1981).
14. P. M. Marcus, Derivation of maximal compatibles using Boolean algebra, *IBM J. Res. Dev.* **8**, 537–538 (1964).
15. G. D. Mulligan and D. G. Corneil, Corrections to Biertone's algorithm for generating cliques. *JACM* **19**, 244–247 (1972).
16. R. E. Osteen, Clique detection algorithms based on line addition and line removal. *SIAM J. Appl. Math.* **26**, 126–135 (1974).
17. M. C. Paul and S. H. Unger, Minimizing the number of states in incompletely specified sequential switching functions. *IRE Trans. Electronic Comput.* **EC-8**, 356–367 (1959).
18. S. Tsukiyama, M. Ide, M. Ariyoshi and I. Shirakawa, A new algorithm for generating all maximal independent sets. *SIAM J. Comput.* **6**, 505–517 (1977).
19. J. W. Moon and L. Moser, On cliques in graphs. *Israel J. Math.* **3**, 23–28 (1965).