JOURNAL OF ALGORITHMS 7, 425–440 (1986)

Algorithms for Maximum Independent Sets

J. M. ROBSON

Department of Computer Science, Australian National University, GPO Box 4, Canberra, ACT 2600, Australia

Received February 5, 1985

An algorithm is presented which finds (the size of) a maximum independent set of an n vertex graph in time $O(2^{0.276n})$ improving on a previous bound of $O(2^{n/3})$. The improvement comes principally from three sources: first, a modified recursive algorithm based on a more detailed study of the possible subgraphs around a chosen vertex; second, an improvement, not in the algorithm but in the time bound proved, by an argument about connected regular graphs; third, a time–space trade-off which can speed up recursive algorithms from a fairly wide class. © 1986 Academic Press, Inc.

1. INTRODUCTION

Finding a maximum independent set (m.i.s.) of a graph is a well-known NP-hard problem, equivalent to finding a maximum clique of the complementary graph [3]. Several algorithms [1, 2] have been published intended to give reasonable average behaviour on instances of these problems with some input distribution. One algorithm [5] is specially designed to achieve a good worst case; it runs in time $O(2^{cn})$ for $c < 1/3$ where n is the order of the graph and returns the size of a m.i.s.; it is easy to modify it if required to return one independent set of this maximum size.

This paper gives two versions of a new but similar algorithm which returns the size of a m.i.s. with the constant c reduced significantly below $\frac{1}{3}$. The first version runs in polynomial space in time $O(2^{cn})$ for $c < 0.296$. The second version, which uses exponential space, reduces the value of the constant to about 0.276. The way in which use of exponential space can speed up an exponential time recursive algorithm is simple and of wider applicability. For instance a naive algorithm to decide satisfiability of a CNF expression E of n clauses over variables $x_1, \ldots x_n$ (or to decide the QBF expression $\forall x_1 \exists x_2 \ldots x_n(E)$) would take time $O(2^n)$; this can be reduced to $O(2^{0.773n})$ by the same technique.

0196-6774/86 $3.00

Section 2 of the paper introduces the terminology, notation, and ideas which are common to the two versions of the algorithm. Sections 3 and 4 give the code and analysis of the polynomial space version of the algorithm which depends on an auxiliary function, presented separately in Section 5. Section 6 discusses the acceleration of this and other algorithms by use of exponential space and Section 7 draws some conclusions and suggests directions for further work.

2. Basic Concepts

2.1. *Notation and Terminology*

Given a graph G or (V, E), if $v \in V$, the degree of v in G will be written $d(v : G)$ or simply as $d(v)$ where no ambiguity is likely. $N(v)$ denotes the set of neighbours of V in G and $N^2(v)$ denotes the set of neighbours in G of vertices in $N(v)$ excluding v itself; $\bar{N}(v) = v + N(v)$. If $u \in V$ or $U \subset V$, we write $G - u$ or $G - U$ for the graph induced on $V - \{u\}$ or $V - U$ by G and in general we write "subgraph" where strictly we mean induced subgraph.

In the program we write $edge(v, w)$ for the predicate that $(v, w) \in E$ and we adhere to Pascal-like conventions on bracketing, using { } strictly for comments and [ ] for set constructors.

2.2. *The Basic Recursive Structure*

The algorithm is presented as a recursive function ms such that $ms(G) = |\text{m.i.s. of } G|$. Most of the ideas in ms are very simple. The most fundamental idea is that given a vertex of G (called B in the program), any maximal independent set either contains B (and therefore no neighbour of B) or does not contain B. This gives the simple recurrence

$$ms(G) = max(1 + ms(G - \bar{N}(B)), ms(G - B)). \tag{1}$$

If $d(B)$ is large (≥ 8), the subproblem $G - \bar{N}(B)$ is so much smaller than the original that the time bound of $k2^{cn}$ for G follows inductively from the same bound for $(G - B)$ and $G - \bar{N}(B)$ and the inequality $2^{-9c} + 2^{-c} < 1$. If the algorithm does not find a vertex with high degree, it must consider the neighbourhood of some vertex in more detail.

2.3. *The Auxiliary Function ms^2*

In the neighbourhood of a vertex (called A) of low degree, another idea is useful. If the function is going to consider independent sets containing A,

then it may ignore those that contain exactly one element of $N(A)$ since such a set has the same size as one containing A instead; even more so it may ignore an independent set containing no element of $\overline{N}(A)$. Thus we have another recurrence

$$ms(G) = max(1 + ms(G - \overline{N}(A)), ms^2(G - A, N(A))). \quad (2)$$

Here $ms^2(G, S)$ is the maximum size of an independent set of G containing at least two elements of S. (In fact the ms^2 we use is not exactly as defined. It sometimes "ignores" the extra information S and returns the size of some independent set of G which is at least as large as every independent set containing two or more elements of S. That change does not invalidate Eq. (2)).

If $d(A)$ is small, then $ms^2(G - A, N(A))$ may be much faster to compute than $ms(G - A)$. This suggests the algorithm's overall approach: two adjacent vertices are chosen with $d(A)$ small and $d(B)$ large; if $d(A)$ is small enough for Eq. (2) to ensure fast computation then it is used; otherwise Eq. (1) is used and in the larger subproblem $(G - B)$, the degree of A has been reduced producing an approach to a graph where (2) will be useful.

2.4. *Regular Graphs*

The approach described above will be least effective in a regular graph (one where all vertices have the same degree), because then it is not even possible to choose A and B such that $d(B) > d(A)$. For this reason, in the algorithm of [5], regular graphs required the most intricate analysis and produced the worst behaviour. In this paper we circumvent this problem by showing that regular graphs are rare enough to be irrelevant even to the worst case behaviour of the algorithm. Since disconnected graphs are easily dealt with and a connected regular graph of degree d has no proper subgraph which is regular with degree d, any chain of recursive calls can include only one regular degree d graph. As shown in Section 4, this implies that ignoring regular graphs of degree < 8 affects only the constant hidden in the $O(2^{cn})$ notation.

2.5. *Dominance*

One last idea is occasionally useful. A vertex A is said to "dominate" B if $\overline{N}(A) \subset \overline{N}(B)$. If this is so, any independent set containing B has the same size as another containing A instead of B; this gives the recurrence

$$ms(G) = ms(G - B). \quad (3)$$

This is a very restricted form of the notion of dominance discussed in [5]. In Section 5 we will also need a slightly more general case where an independent subset A of V "dominates" another subset B in that $\bigcup_A \bar{N}(A) \subset \bigcup_B \bar{N}(B)$ and $|A| \geq |B|$; in this case by similar reasoning there is some m.i.s. which does not contain all of B.

3. The Function *ms*

This is the polynomial space version of *ms*. Since the time bound to be shown is $O(\text{polynomial}(n)2^{cn})$ for graphs of order n, we can omit details which merely affect the polynomial. Thus we give the function in a Pascal-like language with many graph operations described in a loose mixture of English, Pascal, and mathematics on the understanding that they can be coded into Pascal procedures which will run in time polynomial in the order of the graph. The syntax of Pascal has also been modified to use *return* statements as the method of returning a result from a function and to include conditional expressions.

```
function ms(G: graph): integer;
begin
if not connected (G) then
  begin
  C := smallest connected component of G;
  return ms(G − C) + (if |C| ≤ 2 then 1 else ms(C))
  end;
if |G| ≤ 1 then return |G|;
choose A, B vertices of G such that
(i) d(A) is minimal and
(ii) (A, B) is an edge of G and d(B) is maximal over all neighbours of
vertices with degree d(A);
if d(A) = 1 then return 1 + ms(G − N̄(A));
if d(A) = 2 then
  begin
  B' := N(A) − B; {the other neighbour of A}
  if edge(B, B') then return 1 + ms(G − N̄(A));
  return max(2 + ms(G − N̄(B) − N̄(B')), 1 + ms²(G − N̄(A), N²(A)))
  end;
if d(A) = 3 then return max(ms²(G − A, N(A)), 1 + ms(G − N̄(A)));
if A dominates B then return ms(G − B);
return max(ms(G − B), 1 + ms(G − N̄(B)))
end;
```

Only the case $d(A) = 2$ should need any further explanation. If (B, B') is an edge then clearly there is some m.i.s. containing A. Otherwise the function considers independent sets containing either (B and B') or (A and at least two elements of $N^2(A)$); there must be some m.i.s. of one of these forms since one containing A and at most one element of $N^2(A)$ could be modified to include B and B' instead, without decreasing its size.

4. Analysis of *ms*

4.1. *Irregular Graphs*

In this section a time bound of $O(\text{polynomial } (n)2^{cn})$ is proved for the running time of *ms* with $c < 0.296$. This bound depends for the moment on an oracle which returns the value of *ms* on regular graphs of order 4 to 7 and on assumptions about the speed of the auxiliary function ms^2. Section 4.2 shows that the dependence on the oracle can be removed with only a multiplication of the time bound by a constant. Section 5 gives the function ms^2 and justifies the assumptions used here.

The central fact to be proved is that a call of *ms* on a graph of order n produces at most $k2^{cn}$ "trivial" calls of *ms*, that is, calls which return a result with no recursion. Since a nontrivial call involves a polynomial bounded amount of work plus one or more calls on smaller graphs, this implies the stated time bound of $O(\text{polynomial } (n)2^{cn})$. k is chosen so that the result is true for all small graphs and the result is then proved by induction on n. In the interests of brevity we use a somewhat loose terminology and refer to the number of trivial calls arising from a call of *ms* or ms^2 as the "time" taken by that call.

In order to complete the inductive step of the proof, we need to strengthen the bound claimed for certain classes of graphs. To be precise we prove the following theorem.

Theorem 1. *For $c \geq 0.296$ there exists k such that the time taken by ms on a graph G is at most $t(G, |G|)$ provided calls on regular connected graphs G of degree 4 to 7 are replaced by calls on an oracle which uses time $t(G, |G|)$, where*

$t(G, n) =$ if G has a vertex of degree 1 then $k2^{c(n-2)}$
else if G has a vertex of degree 2 then $k2^{c(n-1\frac{1}{2})}$
else if G has a vertex of degree 3 then c_3k2^{cn}
else if G is not connected then $k2^{c(n-1)}$
else if G has a vertex of degree 4 then c_4k2^{cn}

else if G has a vertex of degree 5 then $c_5 k2^{cn}$
else $k2^{cn}$
and $c_3 = 2^{-4c} + 2^{-5c} < 0.799$
$c_4 = c_3 2^{-c}/(1 - 2^{-6c}) < 0.919$
$c_5 = c_4 2^{-c}/(1 - 2^{-7c}) < 0.982$,
on the assumption that the function ms^2 *on graphs* G, S *takes time* $t^2(G, |G|, S, |S|)$
where $t^2(G, n, S, m) =$
if $m \geq 5$ then $t(G, n)$
else if $m = 4$ then $k2^{c(n-1)}$
else if $m = 3$ or $m = 2$ then
if S has a vertex of degree 0 (in G) then $k2^{c(n-2)}$
else if S has a vertex of degree 1 (in G) then $k2^{c(n-3)}$
else $k2^{c(n-4)}$
else 1.

Proof. The proof is by induction on n. k is chosen > 1 and such that the result holds for all graphs of order < 11. Next we assume the result proved for all $n' < n$ and prove it for n.

The structure of the inductive step follows that of the function. We write $t(n;$ *condition*$)$ for the maximum of $t(G, |G|)$ over graphs G such that $|G| = n$ and *condition* holds; $t(n)$ is an abbreviation for $t(n,$ *true*$)$, that is, $k2^{cn}$. Similarly we write $t^2(n, m;$ *condition*$)$ for the maximum of $t^2(G, |G|, S, |S|)$ over graphs G and S such that $|G| = n, |S| = m$ and *condition* holds; $t^2(n, m)$ is an abbreviation for $t^2(n, m;$ *true*$)$.

(i) If G is not connected and $|C| = i$,
time $\leq t(i) + t(n - i)$ which is maximised by minimising i but $i = 1$ and $i = 2$ are special cases, giving
time $\leq max(t(n - 1), t(3) + t(n - 3))$
$= t(n - 1)$.
Moreover if G has a vertex of degree 1, 2, or 3, then
either ($i >$ this degree and C has the low degree vertex)
or ($G - C$ has the low degree vertex).
In each case this establishes the stronger bound; for instance, for degree 1,
time $\leq max(t(n - 1;$ G has a vertex with degree 1),
$t(3) + t(n - 3;$ G has a vertex with degree 1),
$t(3;$ G has a vertex with degree 1$) + t(n - 3))$
$\leq max(t(n - 3), t(1) + t(n - 3))$
$< t(n - 2)$.
(ii) if $d(A) = 1$, there is one recursive call $ms(G - \bar{N}(A))$ giving
time $\leq t(n - 2)$ as required.

(iii) If $d(A) = 2$, there are two recursive calls (except in the case *edge*(B, B') which is trivial), $ms(G - \bar{N}(B) - \bar{N}(B'))$ and $ms^2(G - \bar{N}(A), N^2(A))$; let $x = |N^2(A)|$.

(iiia) If $x \leq 1$, the ms^2 call is trivial giving
time $\leq 1 + t(n-3) < t(n-2)$;

(iiib) if $x = 2$, time $\leq t(n-5) + t^2(n-3, 2)$
$\leq 2t(n-5) < k2^{c(n-1\frac{1}{2})}$;

(iiic) If $x = 3$, time $\leq t(n-6) + t^2(n-3, 3)$
$\leq t(n-6) + t(n-5) < k2^{c(n-1\frac{1}{2})}$;

(iiid) if $x = 4$, time $\leq t(n-7) + t^2(n-3, 4)$
$\leq t(n-7) + t(n-4) < k2^{c(n-1\frac{1}{2})}$;

(iiie) if $x \geq 5$, time $\leq t(n-8) + t(n-3) < k2^{c(n-1\frac{1}{2})}$ as required.

(iv) If $d(A) = 3$, there are two recursive calls $ms^2(G - A, N(A))$ and $ms(G - \bar{N}(A))$ taking time
$\leq t^2(n-1, 3;$ G has no vertex with degree $< 2) + t(n-4)$
$\leq t(n-5) + t(n-4) = c_3 k 2^{cn}$.

(v) If G is not connected but has no vertices of degree ≤ 3, $ms(C)$ is called for each C a connected component of G. Each component must have order at least 5 giving
time $\leq t(5) + t(n-5)$ with $n \geq 10$
$\leq k(3 + 2^{c(n-1)}2^{-4c})$
$\leq k(3 + 2^{c(n-1)-1})$
$< k2^{c(n-1)}$ since $2^{9c} > 6$.

(vi) Otherwise, unless A dominates B there are two recursive calls $ms(G - B)$ and $ms(G - \bar{N}(B))$.

(via) If $d(A) \geq 7$, $d(B) \geq 8$ since G cannot be regular with degree 7; hence
time $\leq t(n-1) + t(n-9) \leq k2^{cn}$;

(vib) if $d(A) = 6$, $d(B) \geq 7$ and $d(A : G - B) = 5$; hence
time $\leq c_5 k 2^{c(n-1)} + k2^{c(n-8)}$
$\leq k2^{cn}$;

(vic) if $d(A) = 4$ or 5, $d(B) \geq d(A) + 1$, and if $d(B) = d(A) + 1$ then there is another vertex $B' \in N(A)$ such that $d(B') = d(A)$ or $d(A) + 1$ and (B, B') is not an edge (from the choice of A and B and the fact that A does not dominate B).

Hence either $|G - \bar{N}(B)| < n - d(A) - 2$
or $|G - \bar{N}(B)| = n - d(A) - 2$ and $G - \bar{N}(B)$ has a vertex of degree $\leq d(A : G)$;
thus $t(G, n) \leq c_{d(A)-1} k 2^{c(n-1)}$
$+ max(k2^{c(n-d(A)-3)}, c_{d(A)} k 2^{c(n-d(A)-2)})$ and the second term of the *max* is larger in each case since $c_4, c_5 > 2^{-c}$.

Thus finally $t(G, n) \leq c_{d(A)-1}k2^{c(n-1)} + k2^{c(n-d(A)-2)}c_{d(A)}$
$= c_{d(A)}k2^{cn}$ as required
by the definitions of c_4 and c_5.

4.2. *Regular Graphs*

We now justify the claim that reliance on the oracle for regular connected graphs of degree 4 to 7 only affects the constant k in the time bound. If we drop the oracle for degree 4 graphs we have an algorithm which does work in time $\leq t(n)$ for any graph which has no subgraph which is regular with degree 4; hence it works in time $\leq t(n)$ for any graph which is a proper subgraph of a connected regular degree 4 graph. Now looking at the time taken by *ms* on a regular degree 4 connected graph, we see that the two recursive calls $ms(G - B)$ and $ms(G - \overline{N}(B))$ take time $\leq t(n-1) + t(n-5) \leq k'2^{cn}$ for $k' \approx 1.17k$. Thus by the same inductive argument as before, we prove that the new algorithm, using the oracle only for degree 5 to 7, runs in time $\leq k'2^{cn}$ for all graphs.

Repeating the same argument three more times gives eventually an algorithm which uses no oracle at all and runs in time $\leq k''2^{cn}$ for $k'' \approx 1.38k$ on all graphs.

5. The Auxiliary Function ms^2

Next we present the auxiliary function $ms^2(G, S)$ and justify the assumptions about its running time used in Section 4.1. The discussion of timing is simple and will be included as comment in the program text. After a few trivial but tedious special cases have been disposed of, the logic is very straightforward: if s is an element of S, a m.i.s. must either contain s and one other element of S or contain not s but two other elements of S. Another function ms^1 is sometimes used to handle the first of these; in the second we use the same idea as in *ms* and consider only independent sets containing two or more neighbours of s.

```
function ms²(G: graph; S: vertexset): integer;
  {first comes the declaration of ms¹ which is defined similarly to ms² but
  concerns sets S of which one element is to be in the independent set}
function ms¹(G: graph; S: vertexset): integer;
  {This is only called with S a two element subset of the vertices of G. It
  returns the size of an independent set of G at least as large as the largest
  such set which contains an element of S.
  The elements of S are s₁ and s₂ with d(s₁) ≤ d(s₂).
```

The time to compute $ms^1(G, S)$
$\leq t(n-1)$ if (s_1, s_2) is an edge of G or $d(s_1) \leq 1$
$\leq t(n-2)$ otherwise.
t^1 is defined analogously to t^2}
begin
if $d(s_1) \leq 1$ **then return** $ms(G)$; {time $\leq t(n-1)$}
if $edge(s_1, s_2)$ **then**
  if $d(s_1) \leq 3$ **then return** $ms(G)$ {time $\leq t(n-1)$}
    else return $max(ms(G - \bar{N}(s_1)), ms(G - \bar{N}(s_2))) + 1$;
    {time $\leq 2t(n-5) < t(n-1)$}
if $N(s_1) \cap N(s_2)\langle\rangle\phi$ **then return** $ms^1(G - N(s_1) \cap N(s_2), S)$;
    {time $\leq t^1(n-1) \leq t(n-2)$}
if $d(s_2) = 2$ **then**
  begin
  $E, F :=$ the elements of $N(s_1)$;
  {independent sets to be considered contain s_1 or (s_2, E and F)}
  if $edge(E, F)$ **then return** $1 + ms(G - \bar{N}(s_1))$;
    {time $\leq t(n-3)$}
  if $N(E) + N(F) - s_1 \subset N(s_2)$ **then return** $3 + ms(G - \bar{N}(s_1) - \bar{N}(s_2))$
    {$\bar{N}(s_1) + \bar{N}(s_2)$ has no 4 element independent set containing s_1 or s_2
    and $[E, F, s_2]$ dominates every other 3 element independent set
    time $\leq t(n-6)$}
    return $max(1 + ms(G - \bar{N}(s_1)), 3 + ms(G - \bar{N}(E) - \bar{N}(F) - \bar{N}(s_2)))$
  {time $\leq t(n-3$; G has a vertex of degree 2) $+ t(n-7)$
      $\leq t(n - 4\frac{1}{2}) + t(n-7)$
      $< t(n-2)$}
  end;
return $max(ms(G - \bar{N}(s_2)), ms^2(G - \bar{N}(s_1) - s_2, N(s_2))) + 1$
  {independent set contains s_2 or (s_1 and two elements of $N(s_2)$)}
  if $d(s_1; G) = 2$ **then** also $d(s_1; G - \bar{N}(s_2)) = 2$, giving
    if $d(s_2) = 3$ time $\leq t(n - 5\frac{1}{2}) + t^2(n-4, 3) < t(n-2)$
    if $d(s_2) = 4$ time $\leq t(n - 6\frac{1}{2}) + t^2(n-4, 4) < t(n-2)$
    if $d(s_2) \geq 5$ time $\leq t(n - 7\frac{1}{2}) + t(n-4) < t(n-2)$
  if $d(s_1; G) = 3$ **then** also $d(s_1; G - \bar{N}(s_2)) = 3$, giving
    if $d(s_2) = 3$ time $\leq t(n-5) + t^2(n-5, 3) < t(n-2)$
    if $d(s_2) > 3$ time $\leq t(n-6) + t(n-5) < t(n-2)$
  if $d(s_1) \geq 4$ **then** time $\leq t(n-5) + t(n-6) < t(n-2)$}
end {of ms^1};
begin {ms^2. The elements of S are $s_1, s_2, \ldots$ with $d(s_i) \leq d(s_{i+1})$}
if $|S| \leq 1$ **then return** 0;
if $|S| = 2$ **then if** $edge(s_1, s_2)$ **then return** 0
    else return $2 + ms(G - \bar{N}(s_1) - \bar{N}(s_2))$;

{time $\leq t(n-2-d(s_1))$}
if $|S| = 3$ **then**
{This is the only complicated case and is the crucial one arising from ms with $d(A) = 3$.}
begin
if $d(s_1) = 0$ **then return** $1 + ms^1(G - s_1, S - s_1)$;
{time $\leq t^1(n-1) \leq t(n-2)$}
if $edge(s_1, s_2)$ and $edge(s_2, s_3)$ and $edge(s_3, s_1)$ **then return** 0;
if $edge(s_i, s_j)$ and $edge(s_i, s_k)(j \langle\rangle k)$ **then**
return $2 + ms(G - \bar{N}(s_j) - \bar{N}(s_k))$;
{time $\leq t(n-2-d(s_j))$}
if $edge(s_i, s_j)$ **then return** $1 + ms^1(G - \bar{N}(s_k), [s_i, s_j])(i \langle\rangle k \langle\rangle j)$:
{independent set cannot contain s_i and s_j and so contains one of them and s_k.
time $\leq t^1(n-1-d(s_k)) \leq t(n-2-d(s_k))$}
if vertex $v \in N(s_i) \cap N(s_j)(i \langle\rangle j)$ **then return** $ms^2(G - v, S)$;
{independent set contains s_i or s_j and so not v.
time $\leq t^2(n-1, |S|$; degrees reduced by at most 1)}
if $d(s_1) = 1$ **then return** $1 + ms^1(G - \bar{N}(s_1), S - s_1)$;
time $\leq t^1(n-2) \leq t(n-3)$}
return $max(1 + ms^1(G - \bar{N}(s_1), S - s_1), ms^2(G - \bar{N}(s_2) - \bar{N}(s_3) - s_1, N(s_1)))$
{**if** $d(s_1) = 2$ time $\leq t(n-5) + t^2(n-7, 2) < t(n-5) + t(n-9) < t(n-3)$
if $d(s_1) = 3$ time $\leq t(n-6) + t^2(n-9, 3) \leq t(n-6) + t(n-11) < t(n-4)$
if $d(s_1) \geq 4$ time $\leq t(n-7) + t(n-11) < t(n-4)$
since, in all three cases, the call of ms^1 has its second parameter $S - s_1$
a set of two vertices of degree ≥ 2 with no edge between them}
end $\{|S| = 3\}$;
if $|S| = 4$ **then**
if G has a vertex of degree ≤ 3 **then return** $ms(G)$ {time $\leq t(n-1)$}
else return $max(1 + ms(G - \bar{N}(s_1)), ms^2(G - s_1, S - s_1))$;
{time $\leq t(n-5) + t^2(n-1, 3$; members of S have degree $\geq 3)$
$\leq 2t(n-5)$
$< t(n-1)$}
return $ms(G)\{|S| \geq 5$: time $\leq t(n)\}$
end;

{In each case the time has been shown to be bounded as assumed in Theorem 1. Strictly the proof of Theorem 1 and the bounds on ms^1 and ms^2 are a triple simultaneous induction.}

6. A Time–Space Trade-off

6.1. *Accelerating Exponential Recursive Algorithms*

It is a commonplace observation that functions on integer arguments may be most simply expressed recursively but be grossly inefficient in their recursive form because of a tendency to repeat some subcomputations very many times. This tendency can be removed, either by a dynamic programming approach or by what we call the "memory" method, that is, by retaining the recursive structure while storing all values of the function which have already been computed and never reevaluating such a stored value. The same technique can be used with functions on combinatorial arguments such as graphs and enables us to reduce the constant c in our time bound for *ms* to about 0.276.

In the case of integer arguments an array is probably suitable for storing the already computed values. For combinatorial arguments, if the algorithm is to run on a random access machine, it will generally be effective to store (argument, value) pairs in a balanced tree structure using the argument as key; this depends on having an easily computable ordering of the argument type. In the case of graphs, it is probable that no easily computable ordering exists, so we regard the arguments as subsets of the vertices V of the original graph and order them by the natural ordering of 2^V.

If the algorithm is required to run on a Turing Machine, a slightly more complex approach achieves the same result. This will be discussed in Section 6.4.

First we use a simple argument to show that *ms* modified in this way runs in time $O(2^{cn})$ for $c \approx 0.282$. Section 6.2 uses a rather more complex argument on a slightly different *ms* to reduce the bound still further.

The time taken by the modified *ms* on a graph of order n is divided into two parts, that on small graphs (graphs of order $\leq \alpha n$ where α is a constant to be decided later) and that on large graphs (we have now dropped the definition of time as the number of trivial calls arising).

The time taken on small graphs is bounded by the observation that the number of such graphs is $\leq \sum_{i=0}^{\alpha n} \binom{n}{i}$ giving a time bound of polynomial $(n)\binom{n}{\alpha n}$.

The extra time taken on any large graph (order $n' > \alpha n$) is seen to be bounded by $k2^{c(n'-\alpha n)}$ by the same inductive argument used in Section 4.

Hence

$$\text{total time} \leq \text{polynomial}(n)\left(2^{c(1-\alpha)n} + \frac{1}{\left(\alpha^{\alpha}(1-\alpha)^{(1-\alpha)}\right)^{n}}\right)$$

and choosing $\alpha \approx 0.048$ to balance the two terms gives the bound of $O(2^{0.282n})$.

6.2. *Connected Subgraphs*

The argument of Section 6.1 can be slightly strengthened by the observation that the time spent on small graphs is determined essentially by the number of *connected* small graphs. Any small graph simply causes at most $\alpha n/3$ evaluations of *ms* on connected small graphs. For the moment we limit the discussion to graphs of degree ≤ 8 and show, for small α, an upper bound on the number of connected subgraphs which is much less than $\binom{n}{\alpha n}$.

LEMMA. *If $G = (V, E)$ is a graph of order n and degree ≤ 8, the number of connected induced subgraphs of G order $n' = O(\text{polynomial}(n)(7^7 6^{-6})^{n'})$.*

Proof. We show a simple 1-many mapping from connected induced subgraphs of order n' (other than connected components of G) to triples (v, e, t) where $v \in V, e \in E$ and t is a 7-ary tree of order n'. The conclusion follows by the enumeration of the 7-ary trees of order n' [4] and Stirling's approximation.

Choose an arbitrary ordering of the edges E and let C be a connected induced subgraph of G (not a component). Let v be a vertex of C such that $d(v : C) \leq 7$; if $d(v : G) = 8$, choose e an edge in E which is incident on v and is not an edge of C; otherwise choose e arbitrarily in E. Choose T a spanning tree of C with root v.

T has out degree ≤ 7 at each vertex and, given the ordering of E, gives the 7-ary tree t in an obvious way; the seven subtrees at a node are ordered according to the ordering of the edges and an edge which is not in T gives an empty subtree in t.

It is clear that, given v, e, and t, it is possible to reconstruct T and thereby C, establishing that the mapping is indeed 1-many and completing the proof of the lemma.

THEOREM 2. *If ms is modified in the following two ways its running time is $O(2^{0.276n})$:*

(i) *use of memory to avoid repeated computation on the same subgraph;*

(ii) *whenever G has a vertex V of degree > 8, returning $\max(ms(G - V), 1 + ms(G - \bar{N}(V))$.*

Proof. We prove the result first for graphs of degree ≤ 8. By the lemma the time spent on graphs of order $\leq \alpha n$ is polynomial$(n)(7^7 6^{-6})^{\alpha n}$. By the usual inductive argument, the extra time spent on large graphs resulting from a graph of order n' is $O(2^{c(n' - \alpha n)})$. Choosing $\alpha = 0.0667$ gives a total time of $O(2^{0.276n})$.

The general result is proved inductively where the induction is based on the degree ≤ 8 case. G either has degree ≤ 8 or is dealt with by modification (ii) in time $\leq 2^{0.276(n-1)} + 2^{0.276(n-10)} < 2^{0.276n}$ since $2^{-0.276} + 2^{-2.76} < 1$.

6.3. *An Application to Some Logical Problems*

The same approach of using memory to avoid recomputation can also achieve a substantial speedup in solving QBF (Quantified Boolean Formula) problems over a fairly wide set of expressions including as a small subset formulae in CNF with a number of clauses equal to the number of variables. The special case where all the quantifiers are existential gives the same result for SAT over the same set of expressions. The expressions in question are most easily characterised in terms of boolean circuits rather than syntactically. They can be computed from $2n$ inputs $a_1, \ldots, a_n$, $\neg a_1, \ldots, \neg a_n$ by circuits of a number (linear in n and less than about $1.29n$) of **and** and **or** gates of unbounded fan-in and fan-out subject to the restriction, for each i independently, that *either* no input a_i or $\neg a_i$ goes directly into any **and** gate *or* no input a_i or $\neg a_i$ goes directly into any **or** gate. For simplicity we consider only the case where the number of gates is the same as n the number of variables.

A simple approach to solving such a problem would recursively solve the two subproblems obtained by setting a_{outer} (the variable of the outermost quantification) to true and to false and combine the two results appropriately. A slightly more sophisticated algorithm would first simplify the circuit after each assignment by removing gates whose output was "obvious" (a gate's output is "obvious" if it is directly implied by those inputs which are either the obvious outputs of other gates or already assigned literals) and second notice cases where the function computed by the current circuit was obviously monotonic in a_{outer} (because either a_{outer} or $\neg a_{\text{outer}}$ was not connected to any gate) and solve only one subproblem in these cases; for instance if the circuit is monotonic increasing in a_{outer}, this variable will be set to true if existentially quantified or to false if universally quantified. This algorithm will still take time $O(2^n)$ but its "memorising" version will run in time $O(2^{0.773n})$. The reason is simply that whenever two assignments to a variable are possible, each of them reduces the number of gates by at least one and that a nontrivial subproblem can be specified by the number of assignments made and which gates remain.

Now after n' variables have been assigned a value which was not dictated by the "monotonic" rule, the number of possible subproblems is bounded in two ways; first it is at most $2^{n'}$ since n' choices have been made; second it is at most $(n - n' + 1)\sum_{i=0}^{n-n'} \binom{n}{i}$ since the number of variables assigned to is in the range $[n, n']$ and at most $(n - n')$ gates remain.

Thus choosing $n' \approx 0.773n$ so that $2^{n'} \approx \binom{n}{n'}$ ensures that the time spent on problems with at most n' such assignments and the time spent on those with more than n' are both $O(\text{polynomial}(n)2^{0.773n})$.

If we remove the restriction on the gates into which inputs may go, we still obtain a slight improvement over time $O(2^n)$ by a slightly more complex argument. It can be shown that the time in this case is

$$O\left(\left(\begin{matrix} n \\ \frac{n}{3} \end{matrix}\right)\right) \approx 2^{0.912n}.$$

6.4. *Achieving the Time–Space Trade-off with a Turing Machine*

Since the "memorising" algorithms have space complexity almost as great as their time complexity, simulating them by a Turing Machine might almost square their time bound producing an algorithm much slower than the polynomial space version. This can be avoided by a more careful Turing Machine version of the algorithm which can still obtain exactly the same reduction in the exponent of the time bound.

The significant fact is that the later part of the sequence of recursive calls produced from a given call does not depend on the results of earlier calls. This means that, instead of making the recursive calls in sequence, we could make them in parallel or interleave them. Now a large number of references to the memory can be batched together and such a batch of references can, by sorting them, be made much more efficiently than if they had to be done sequentially.

In order to make it clear that the method being described applies to both the *ms* function and to QBF, we give some fairly general conditions for it to be applicable instead of describing it in terms of a particular function.

THEOREM 3. *If f is a function with domain D and range R such that*

(i) $\exists$ *functions* f_1, f_2, f_3 *such that*
  (ia) $f_1(d) = (d_1, d_2, \ldots, d_{f_2(d)})$ (*an element of* D^*)
  (ib) $f(d) = f_3(d, f(d_1), \ldots, f(d_{f_2(d)}))$ (*the recursive definition of* f)
  (ic) f_1, f_2 *and* f_3 *are in Ptime,*
(ii) *if* $T(d)$ *is the computation tree formed by adding to* d *instances of the trees* $T(d_i)$ *of the elements of* $f_1(d)$ *then*
  (iia) *the depth of* $T(d)$ *is bounded by a polynomial in* $|d|$
  (iib) *any element* t *of* $T(d)$ *has* $|t|$ *bounded by a polynomial in* $|d|$,
(iii) $\exists$ *a total ordering* $\leq$ *on* D *computable in Ptime,*
then $f(d)$ *is computable by a multitape Turing Machine in time bounded by polynomial*$(|d|) \times$ (# *distinct elements of* D *in* $T(d)$).

Proof. Before proceeding to describe the Turing Machine computation, we note that conditions (ic) and (iib) ensure that the degree of elements of $T(d)$ is bounded by polynomial($|d|$) and so (iia) ensures that log(# elements in $T(d)$) = O(polynomial($|d|$)) so that elements of $T(d)$ can be sorted in O(polynomial($|d|$)) passes.

The Turing Machine computation proceeds in two phases. The first phase constructs top-down the representation of the directed acyclic graph $G(d)$ formed from $T(d)$ by identifying nodes with the same element of D and directing edges upwards; duplicated representations of nodes of G are avoided by regular sorting which brings duplicates together so that they can be dealt with. The second phase evaluates f at leaves and then propagates results back up through the graph evaluating f at each node when the results of its recursive calls are available; again sorting is used, this time to move results from where they are computed to where they are needed.

The representation of the graph $G(d)$ is as follows: if $(A \rightarrow B)$ is an edge of the graph, there are two records of it, $(A, ' \rightarrow ', B)$ and $(B, ' \leftarrow ', A)$; the whole graph is represented by the records of its edges sorted lexicographically (so that for each vertex A there is a contiguous sequence of records corresponding to all the edges into and out of A).

This representation of $G(d)$ is easily computed in an order corresponding to a breadth first scan of the tree. Initially the records $(d_i, ' \rightarrow ', d)$ are placed on tape 1 (for d_i the elements of $f_1(d)$) and are "unmarked"; then in successive stages tape 1 is scanned for all unmarked records of the form $(X, ' \rightarrow ', Y)$, these records are "marked" and records $(X_i, ' \rightarrow ', X)$ and $(X, ' \leftarrow ', X_i)$ are added to tape 2 for X_i the elements of $f_1(X)$ with the $' \rightarrow '$ records unmarked; then the records on tape 2 are sorted and merged into tape 1; finally, if tape 1 contains repeated sequences of records $(V, ' \rightarrow ', W)$ for the same V, then if one is already marked the rest are now marked and if all are unmarked all but one are now marked. The number of these stages required is bounded by the depth of $T(d)$ so that the total time for this first phase is bounded by polynomial($|d|$) $\times$ $|G(d)|$ as required.

The second phase now gradually overwrites all the records of the form $(A, ' \leftarrow ', A_i)$ meaning that $f(A)$ depends on $f(A_i)$ with records $(A, ' \leftarrow ', A_i, f(A_i))$; when this has happened for all A_i, $f(A)$ can be computed and passed up to all those B such that a $(A, ' \rightarrow ', B)$ record exists. Eventually we are left only with the records $(d, ' \leftarrow ', d_i, f(d_i))$ needed to compute the value $f(d)$.

In more detail, this phase also consists of a number of stages. At each stage, for every vertex A such that each $(A, ' \leftarrow ', A_i)$ record has been replaced by $(A, ' \leftarrow ', A_i, f(A_i))$, $f(A)$ is computed and then for every $(A, ' \rightarrow ', B)$ record, a new record $(B, ' \leftarrow ', A, f(A))$ is written on tape 2 (and all records of the form $(A, \ldots)$ are cleared from tape 1); then tape 2 is sorted and finally merged into tape 1 with records of the form $(X, ' \leftarrow ',$

$Y, f(Y))$ with $X \neq d$ replacing the original record $(X, \text{`} \leftarrow \text{'}, Y)$. Again this process terminates after a number of stages bounded by the depth of $T(d)$ and the whole computation is now completed in a total time of polynomial$(|d|) \times |G(d)|$.

7. Conclusions

The bound of $O(2^{n/3})$ in [5] on the time to find a maximum independent set has been reduced to $O(2^{0.276n})$ by three main methods. First, a deeper study of the neighbourhood of a chosen vertex in the graph has been made but without a more complex case structure in the program; this was made possible by the auxiliary functions ms^2 and ms^1 which by being mutually recursive hide some indefinitely complex case analysis. Second, the argument that low degree regular graphs can be discounted removed a lot of awkward cases. Third, if one is willing to use exponential space, an essentially trivial way of utilising the store achieves a nontrivial reduction in the exponent. Although the reduction of about 0.05 in the exponent by these three methods may sound modest, it looks better if thought of as a reduction of 99.9% in the time to compute ms for a 200 vertex graph.

Quite apart from the large probability that another algorithm can improve on this bound, it is quite possible that the algorithm given here can be proved to obey a lower bound. First, a more careful analysis of the possible graph structure around the edge (A, B) may show more about the possible sequences of recursive calls, thereby giving a better bound for the polynomial space version, and second, a more accurate enumeration of the small subgraphs which can arise may reveal more about the effectiveness of the "memory" modification.

Acknowledgments

Much of the work reported here was done while I was at the University of Warwick. I am grateful to Professor Mike Paterson for his many helpful comments and to the SERC for funding my stay at Warwick.

References

1. C. Bron and J. Kerbosch, Algorithm 457: Finding all cliques of an undirected graph, *Comm. ACM* **16** (1973), 575–577.
2. J. C. Johnston, Cliques of a graph: Variations on the Bron–Kerbosch algorithm, *Internat. J. Comput. Inform. Sci.*, **5** (1976), 209–238.
3. R. Karp, Reducibility among combinatorial problems, *in* "Complexity of Computer computations" (R. E. Miller and J. W. Thatcher, Eds.), Plenum, New York, 1972.
4. D. E. Knuth, "Fundamental Algorithms," p. 584, Addison–Wesley, Reading, Mass., 1968.
5. R. E. Tarjan and A. E. Trojanowski, Finding a maximum independent set, *SIAM J. Comput.* **6** (1977), 537–546.