Independent Set Algorithms:
Improved performance and improved analysis
Preliminary Report

J.M. Robson
LaBRI
Université Bordeaux 1

1 Introduction

We are interested in algorithms with provable upper bounds for finding a maximum independent set of a graph. This is a maximum clique of the complementary graph or the complement of a minimum vertex cover of the original graph. We prefer to talk about independent sets rather than cliques, simply because in the interesting cases for our algorithms the graph is much sparser than its complement.

[2] showed an algorithm running in time $O(2^{n/3})$ where n is the size of the graph and finding the size of a maximum independent set. Given an algorithm for the size, one to find an example set of this size multiplies the time by only a polynomial in n. [1] described some minor modifications which reduced the constant c in the $O(2^{cn})$ time bound to about 0.276. In this paper we take this work a little further describing two modifications which reduce c to about 0.272.

[3] independently improved the algorithm in [2] obtaining a constant of about 0.304.

1.1 Terminology and notation

We will abbreviate "maximum independent set" by "*m.i.s.*".

If $G = (V, E)$ is a graph and v a vertex of G, we write $N(v)$ for the neighbourhood of v, namely $\{u \in V | \ (u, v) \in E\}$ and $\bar{N}(v)$ for $v + N(v)$. If $X \subset V$, we write $G - X$ for the graph induced on $V - X$. $d(v)$ is the degree of v in the graph currently under consideration.

2 Outline of the algorithm

We give here a very brief description of the algorithm in [1] which should be enough to make it possible to follow the suggested improvements.

2.1 Recursive structure

The algorithm returns the size of an m.i.s. of graph G by choosing a vertex A of G examining the structure of the graph around A and making recursive calls corresponding to a choice to include or exclude various vertices around A in the independent set. The result is obtained by maximising over an exhaustive set of apparently plausible choices of the value of the recursive call plus the number of vertices included in the independent set. The analysis inductively proves that the number of calls from a graph G is at most $k_G 2^{c|G|}$ where k_G depends on the local structure found. For many cases the choice of A is simply to minimize $d(A)$ and maximize the degree of some neighbour B of A.

2.2 Dominance

One special case occurs when a vertex B is adjacent both to A and to all neighbours of A; in this case it is clear that an independent set containing B can be modified to one of the same size containing A and so B can be deleted from the graph, that is $ms(G) = ms(G - B)$.

It will be useful to exploit this idea in the case of *subdominance*. If B is adjacent to A and all neighbours of A except one (C), then in a recursive call on the graph $G - C$, A does dominate B.

One other example of dominance turns out to be useful (at least for the analysis of the algorithm; perhaps it doesn't actually affect the performance appreciably); if a cycle $(v_0,\ v_1, \cdots,\ v_n)$ has all even numbered vertices $(v_0, v_2, \cdots)$ with degree 2, the independent set $\{v_0, v_2, \cdots\}$ dominates every other subset of the vertices in the cycle and we can simply include this set in our independent set and exclude all the odd numbered v_i.

Another case similar to dominance concerns the existence of *twins*, that is pairs of vertices which are not connected but have exactly the same set of neighbours. In this case any m.i.s. contains both the twins or neither.

2.3 Connected non-regular graphs

If a graph is not connected, we simply call the function on each of its connected components. It can be argued that connected regular graphs occur rarely enough to be irrelevant to the asymptotic analysis so we will always assume that G is not regular. In particular the choice of A described above will always be assumed to yield a neighbour B with $d(B) > d(A)$.

2.4 Special algorithms when the m.i.s. must contain a specified number of elements of a subset

We will use auxiliary functions ms^1 and ms^2 such that $ms^i(G, G')$ lies in the range between the largest m.i.s. of G containing at least i elements of G' and the unconstrained largest m.i.s. [1] used ms^1 only in the case that G' contained exactly 2 elements; we will also need to consider cases with $|G'| = 3$. ms^1 and ms^2 are defined mutually recursively with ms and bounds on their run time are proved similarly to those on that of ms and by a process of mutual induction.

3 A simple improvement

3.1 The local structure

In the case of minimum degree 4, 5 or 6, the algorithm chooses a vertex A of minimum degree i and a neighbour B of A of maximum degree $j > i$. Since A does not dominate B, there is another neighbour B' of A which is not a neighbour of B. We write j' for the degree of B'. By the choice of A and B, we have $i \leq j' \leq j$. The most interesting case is when $j = i + 1$ since this gives the worst case of the algorithm; in this case j' has only two possible values. The algorithm proceeds by two recursive calls on the graphs $G - B$ and $G - \bar{N}(B)$. In the first, we know that there is a vertex (A) of degree $i - 1$; in the second we have no special information to constrain the time.

3.2 Improvements depending on j'

If $j' = i$, the recursive call on $G - \bar{N}(B)$ has a vertex (B') of degree $\leq i - 1$, a fact which reduces our upper bound on its time. But if $j' > i$, the recursive call on $G - B$ has a vertex of degree j' adjacent to the vertex A of minimum degree, again reducing our bound. So we obtain a better upper bound for these cases by taking the maximum over all possible j' of the sum of the bounds on these two recursive calls.

3.3 Analysis

For all i, j such that $4 \leq i < j \leq 8$, we define a constant $k_{i,j}$ such that we can prove inductively that when we choose vertices A and B with degrees i and j, the run time is bounded by $k_{i,j} 2^{cn}$ provided it is bounded by 2^{cn} in those recursive calls with minimum degree greater than 8. We also slightly improve the analysis in some cases of the neighbourhood of a vertex of degree 2. A simple calculation now gives the optimum values of the $k_{i,j}$ for a given c. The table shows the $k_{i,j}$ found for $c = 0.293$. Since all $k_{i,j} \leq 1$, we deduce that the algorithm runs in time $O(2^{0.293n})$.

i	j	$k_{i,j}$
4	5	0.932995
4	6	0.895246
4	7	0.854043
4	8	0.817854
5	6	0.986661
5	7	0.945280
5	8	0.911504
6	7	0.999652
6	8	0.963933
7	8	0.976625

The time-space trade-off described in [1] then achieves a reduction to $O(2^{0.273n})$

4 Avoiding the worst case

4.1 The worst case

We now observe that the worst case of the algorithm (more precisely of our upper bound) comes when a vertex A of degree 6 has a neighbour B of degree 7 such that every vertex in $N(A) - N(B)$ has degree 6 and in the subsequent recursive call on $G - B$, (where A is still the vertex of minimum degree), A has no neighbour of degree > 6 nor two non-adjacent neighbours each of degree 6. We will now show that if this would occur for every possible choice of A and B, we can always improve the performance of the algorithm.

4.2 Three conditions necessary for the worst case

First we show that if the basic algorithm will produce the worst case behaviour for every possible choice of A and B, one of three patterns must occur in the induced subgraph on $N(A)$:

- a vertex v has degree ≥ 4,
- three vertices $v_1,\ v_2,\ v_3$ form a triangle,
- or the vertex B of degree 7 (in the original G) is adjacent to three vertices $v_1,\ v_2,\ v_3$ which have no edges between them and the remaining vertices $v_4,\ v_5$ are not adjacent to B but do have an edge between them. moreover, v_1 is adjacent (in G) to a vertex of degree 6 which is not v_4 or v_5.

Proof: Suppose that the worst case does occur and that neither of conditions (a) or (b) holds. For the worst case to occur in $G - B$, A must not have two neighbours of degree 6 in $G - B$ which are not adjacent. So any two neighbours of A (in G) with degree 7 must be adjacent and any pair of vertices in $N(A) - N(B)$ must be connected. Thus, if any vertex of degree 7 has three non-neighbours, there is a triangle. If there are two vertices B_1 and B_2 of degree 7, no other vertex can be adjacent to both and there can be at most two vertices not adjacent to B_i since otherwise any three would form a triangle; so we must have two vertices adjacent to B_1 but not to B_2. But then these two vertices together with B_2 form a triangle. We conclude that there is a single vertex B of degree 7. We know that B cannot have three non-neighbours and it cannot have four neighbours; the only remaining possibility is that it has three neighbours and two non-neighbours. since its non-neighbours must be connected and its neighbours cannot form a triangle with it, we conclude that the induced graph must be as stated in condition (c). Finally if v_1 is one of the neighbours of B, it has degree 6 and has a neighbour of degree 7 and so it is a candidate to be chosen as A; if that choice would also give the worst case, then v_1 similarly has two non-connected neighbours of degree 6; only one of them can be v_4 or v_5 since (v_4, v_5) is an edge and (v_1, v_4, v_5) is not a triangle.

4.3 Improvements for these conditions

Subdominance If v has degree 4 in $N(A)$, there is only one vertex, say u in $N(A) - v$. In the graph $G - u$, A dominates v; so we have $ms(G) = max(1 + ms(G - \bar{N}(u)), ms(G - u - v))$. The two recursive calls are on graphs of sizes $\leq n - 6$ and $n - 2$ giving significantly improved performance.

Triangles If v_1, v_2 and v_3 form a triangle and the remaining vertices of $N(A)$ are v_4, v_5 and v_6, knowing that there is a m.i.s. containing A or one containing two elements of $N(A)$ gives time bound of $1 + max(ms(G - \bar{N}(A)), ms(G - \bar{N}(v_4)), ms(G - \bar{N}(v_5) - v_4), ms(G - \bar{N}(v_6) - v_4 - v_5))$ The recursive calls after the first are on graphs of size at most $n - 7$ which already reduces the upper bound on the time (and a little more detailed analysis would certainly reduce it further).

The special case The argument is similar to the case of a triangle except that in the cases where B or (v_3 and not v_1 or v_2) are put into the independent set the second element of $N(A)$ to be in the set must be one of v_4 and v_5. The time bound is $1 + max(ms(G - \bar{N}(A)), ms^1(G - \bar{N}(B), \{v_4, v_5\}), ms(G - \bar{N}(v_1)), ms(G - \bar{N}(v_2) - v_1), ms^1(G - \bar{N}(v_3) - v_1 - v_2), \{v_4, v_5\})$. Moreover in the three calls of ms (as opposed to ms^1), there is always a vertex of degree 5 or less, namely v_4 or v_5 or the anonymous vertex adjacent to v_1 and having degree 6 in G. Hence these three calls have their time bounds reduced.

4.4 Extent of the improvement

Although the improvements described in the previous section are all fairly effective (around 4-5% at least), the improvement in the overall performance of the algorithm is slight because only a minor improvement is obtained in the cases where our conditions do not hold and the recursive call on $G - B$ is not worst case because A has two non connected neighbours of degree 6. Nevertheless the constant c is reduced, to 0.292 or to 0.272 with the time-space trade-off.

5 Small subgraphs

A useful gain in time is obtained at the cost of using exponential space by remembering the result found for any induced subgraph already analysed and avoiding doing the same calculation several times. This gain is evaluated by observing that the number of calls of the algorithm on graphs of size greater than m is $O(2^{c(n-m)})$ so that if m is chosen so that the number of induced subgraphs of size $\leq m$ is also $O(2^{c(n-m)})$ then so is the overall time.

5.1 Connected induced subgraphs

In fact it suffices to consider connected induced subgraphs and to consider graphs with degree bounded by 8 since the performance bounds we will obtain are easily proved recursively for all graphs if they hold for degree ≤ 8 graphs. A bound on the number of small connected induced subgraphs is then obtained by considering the number of 7-ary trees of size m (where a 7-ary tree is either empty or has a root node and 7 subtrees; note that the order of the subtrees is important). Relative to a given root node and an ordering of the edges at each vertex, the tree is interpreted as a spanning tree of the subgraph.

This bound overestimates the number of subgraphs for three reasons:

- Where the tree indicates that an edge is to be followed, the graph may have no edge; this is not significant since we want an upper bound on the number of connected subgraphs and adding more edges can only increase the number.
- An edge may lead to a vertex which has already been visited.
- A connected subgraph may have many spanning trees.

For the second of these three phenomena we have only an estimate of its effect on the performance of the algorithm. For the third we can show a definite improvement. We rely on the fact that the time taken by the algorithm is bounded by a polynomial times the number of graphs considered where two or more recursive calls are made. Thus, we need only count subgraphs with minimum degree at least 2 and no occurrences of dominance of the types mentioned in Section 2.2. In the following section we will assume that subgraphs are of this type.

5.2 Weighted Spanning Trees

Suppose a spanning tree of a subgraph has l leaves. Since each of these vertices has degree at least 2 *qua* subgraph vertex, we can obtain other spanning trees (also with at most l leaves) by attaching any leaf v to another neighbour and breaking the resulting cycle just below the first ancestor of v (in the original spanning tree) which is either of degree greater than 2 in the spanning tree or is its root. If we start with a spanning tree which minimises the number of leaves, no two leaves can be adjacent in the subgraph and this process of obtaining new spanning trees can be done without interference for any subset of the leaves and results in a different spanning tree in each case.

Hence if each 7-ary tree is given a weight equal to 2^{-l} where l is its number of leaves, the sum of the weights for all spanning trees of a subgraph with minimum number m of leaves over all its spanning trees, is at least $2^m 2^{-m}$ i.e. at least 1.

In fact we can justify the use of even smaller weights. Any leaf with more than two neighbours in the subgraph multiplies the number of spanning trees by at least 3 instead of by 2. A leaf whose father has degree 2 allows also the possibility of breaking the cycle in another way producing another spanning tree but with one more leaf. Finally, where a leaf v has two neighbours in the subgraph each of them having a son different from v, our method for forming a new tree is simply to join v to its "other" neighbour and detach it from its father; we can also form a new spanning tree by joining v to both its neighbours and breaking the resulting cycle beyond its father. We can do this for any subset of such leaves since the set of all new and old (leaf,father) edges cannot contain a cycle (by the comment on dominance in cycles in Section 2.2) and we can show that it can be done in such a way that the number of leaves is not increased. All this shows that we can use a weight of γ^{-l} where $\gamma = (1+\sqrt{2})$ instead of 2^{-l}. (Each leaf multiplies the sum of weights of the spanning trees obtained by at least $2 + 1/\gamma \ = \ \gamma$).

The number of 7-ary trees with size m and m_i nodes of outdegree i ($0 \leq i \leq 7$) is $\binom{m}{m_0, m_1, m_2, m_3, m_4, m_5, m_6, m_7} \prod_{i=0}^{7} \binom{7}{i}^{m_i} / m$ provided the m_i satisfy $\sum_{i=0}^{7} m_i \ = \ m$ and $\sum_{i=0}^{7} i m_i \ = \ m-1$. If we fix the proportion of tree nodes with out-degree i to $x_i + O(1/m)$ and let m tend to infinity, the sum of the weights of trees of size m is to within a polynomial factor $(\gamma^{-x_0} \prod_{i=0}^{7} (\binom{7}{i} / x_i)^{x_i})^m$. A simple calculation shows that subject to the constraints $\sum_{i=0}^{7} x_i \ = \ 1$ and $\sum_{i=0}^{7} i x_i \ = \ 1$ the constant is maximum at about 13.536 when the x_i are about
(0.26339, 0.51713, 0.18024, 0.03490, 0.00405, 0.00028, 0.00001, 0.00000). Hence summed over all possible m_i the sum is bounded by $13.536^m \times poly(m)$. This can be compared with the unweighted sum (the number of trees) which is about 17.65138^m.

This raises the level at which the memorisation becomes effective (number of subgraphs possible is less than the number of recursive calls generated) to about $0.072n$ thereby reducing the constant c to just under 0.271.

6 Further work

6.1 A catalogue of neighbourhoods

In the same way that we used in the neighbourhood of a vertex of degree 6, we intend to examine all possible induced subgraphs in the neighbourhood of a vertex A of degree 5 and 4. The number of induced graphs needing to be considered is not as large as it might appear since many graphs can be treated similarly (for instance all graphs containing a triangle, all graphs where A subdominates another vertex). Giving a series of recursive calls tailored to each such graph rather than relying on the step by step analysis currently used seems capable of improving the treatment of every possible neighbourhood. It is likely that minor improvements of this type can reduce the value of c obtainable by the basic algorithm at least as far as $c \approx 0.288$ where the inequality $2^{-3c} + 2^{-4c} < 1$ ceases to hold and all the analysis of the simple cases needs to be reexamined. With the memorisation improvement, this will give an overall performance of $O(2^{0.2675n})$.

6.2 Spanning Trees

It should still be possible to obtain a better upper bound on the probability that a randomly chosen 7-ary tree really does span a subgraph and that this subgraph contains no (polynomial recognisable) pattern allowing simplification without recursive branching. A probabilistic argument suggests that a random tree of the size being considered is very likely to contain many vertices more than once. Moreover the leaf nodes of the tree are quite likely to give rise to degree 1 vertices in the subgraph, a case which we have seen we can ignore. Analysis of these effects for the worst case original graph is not obvious.

References

[1] Robson, J. M., Algorithms for maximum independent sets, *J. Algorithms* **7** (1986), 425-440.

[2] Tarjan, R. E. and Trojanowski, A. E., Finding a maximum independent set, *SIAM J. Comput.* **6** (1977), 537-546.

[3] Jian, T., An $O(2^{0.304n})$ algorithm for solving maximum independent set problem, *IEEE Trans. Comput.* **C-35** (1986), 847-851.