Title:	Exact Algorithms for Maximum Independent Set
Name:	Fabrizio Grandoni
Affil./Addr.	IDSIA, University of Lugano, Switzerland
Keywords:	Maximum independent set; Exact exponential algorithms
SumOriWork:	1977; Tarjan, Trojanowski 1985; Robson 1999; Beigel 2009; Fomin, Grandoni, Kratsch

Exact Algorithms for Maximum Independent Set

Fabrizio Grandoni

IDSIA, University of Lugano, Switzerland

Years aud Authors of Summarized Original Work

1977; Tarjan, Trojanowski
1985; Robson
1999; Beigel
2009; Fomin, Grandoni, Kratsch

Keywords

Maximum independent set; Exact exponential algorithms

Problem Definition

Let $G = (V, E)$ be an n-node undirected, simple graph without loops. A set $I \subseteq V$ is called an *independent set* of G if the nodes of I are pairwise not adjacent. The *Maximum Independent Set* problem (MIS) asks to determine the maximum cardinality $\alpha(G)$ of an independent set of G. MIS is one of the best studied NP-hard problems.

We will need the following notation. The (open) *neighborhood* of a vertex v is $N(v) = \{u \in V : uv \in E\}$, and its *closed neighborhood* is $N[v] = N(v) \cup \{v\}$. The degree $deg(v)$ of v is $|N(v)|$. For $W \subseteq V$, $G[W] = (W, E \cap \binom{W}{2})$ is the *graph induced* by W. We let $G - W = G[V - W]$.

Key Results

A very simple algorithm solves MIS (exactly) in $O^*(2^n)$ time: it is sufficient to enumerate all the subsets of nodes, check in polynomial time whether each subset is an independent set or not, and return the maximum cardinality independent set. We recall

that the O^* notation suppresses polynomial factors in the input size. However, much faster (though still exponential-time) algorithms are known. In more detail, there exist algorithms that solve MIS in worst-case time $O^*(c^n)$ for some constant $c \in (1, 2)$. In this section we will illustrate some of the most relevant techniques that have been used in the design and analysis of exact MIS algorithms. Due to space constraints, our description will be slightly informal (please see the references for formal details).

Bounding the Size of the Search Tree

All the non-trivial exact MIS algorithms, starting with [7], are recursive branching algorithms. As an illustration, consider the following simple MIS algorithm `Alg1`. If the graph is empty, output $\alpha(G) = 0$ (base instance). Otherwise, choose any node v of maximum degree, and output

$$\alpha(G) = \max\{\alpha(G - \{v\}), 1 + \alpha(G - N[v])\}.$$

Intuitively, the subgraph $G - \{v\}$ corresponds to the choice of not including v in the independent set (v is *discarded*), while subgraph $G - N[v]$ to the choice of including v in the independent set (v is *selected*). Observe that, when v is selected, the neighbors of v have to be discarded. We will later refer to this branching as a *standard branching*.

The running time of the above algorithm, and of branching algorithms more in general, can be bounded as follows. The recursive calls induce a *search tree*, where the root is the input instance and the leaves are base instances (that can be solved in polynomial time). Observe that each branching step can be performed in polynomial time (excluding the time needed to solve subproblems). Furthermore, the height of the search tree is bounded by a polynomial. Therefore, the running time of the algorithm is bounded by $O^*(L(n))$, where $L(n)$ is the maximum number of leaves of any search tree that can be generated by the considered algorithm on an input instance with n nodes. Let us assume that $L(n) \leq c^n$ for some constant $c \geq 1$. When we branch at node v we generate two subproblems containing $n - 1$ and $n - |N[v]|$ nodes, respectively. Therefore c has to satisfy $c^n \geq c^{n-1} + c^{n-|N[v]|}$. Assuming pessimistically $|N[v]| = 1$, one obtains $c^n \geq 2c^{n-1}$ and therefore $c \geq 2$. We can conclude that the running time of the algorithm is $O^*(2^n)$. Though the running time of `Alg1` does not improve on exhaustive search, much faster algorithms can be obtained by branching in a more careful way and using a similar type of analysis. This will be discussed in next subsections.

Refined Branching Rules

Several refined branching rules have been developed for MIS. Let us start with some *reduction rules*, which reduce the problem without branching (alternatively, by branching on a single subproblem). An isolated node v can be selected w.l.o.g.:

$$\alpha(G) = 1 + \alpha(G - N[v]).$$

Observe that if $N[u] \subseteq N[v]$, then node v can be discarded w.l.o.g. (*dominance*):

$$\alpha(G) = \alpha(G - \{v\}).$$

This rule implies that nodes of degree 1 can always be selected.

Suppose that we branch at a node v, and in the branch where we discard v we select exactly one of its neighbors, say w. Then by replacing w with v we obtain a solution of the same cardinality including v: this means that the branch where we

select v has to provide the optimum solution. Therefore, we can assume w.l.o.g. that the optimum solution either contains v or at least 2 of its neighbors. This idea is exploited in the *folding* operation [1], that we next illustrate only in the case of degree-2 nodes. Let $N[v] = \{w_1, w_2\}$. Remove $N[v]$. If $w_1w_2 \notin E$, create a node v' and add edges between v' and nodes in $N(w_1) \cup N(w_2) - \{v\}$. Let $G_{fold}(v)$ be the resulting graph. Then one has

$$\alpha(G) = 1 + \alpha(G_{fold}(v)).$$

Intuitively, including node v' in the optimal solution to $G_{fold}(v)$ corresponds to selecting both w_1 and w_2, while discarding v' corresponds to selecting v.

Let `Alg2` be the algorithm that exhaustively applies the mentioned reduction rules, and then performs a standard branching on a node of maximum degree. Reduction rules reduce the number of nodes at least by 1, hence we have the constraint $c^n \geq c^{n-1}$. If we branch at node v, $deg(v) \geq 3$. This gives $c^n \geq c^{n-1} + c^{n-4}$, which is satisfied by $c \geq 1.380\ldots$. Hence the running time is in $O^*(1.381^n)$.

Let us next briefly sketch some other useful ideas that lead to refined branchings. A *mirror* [3] of a node v is a node u at distance 2 from v such that $N(v) - N(u)$ induces a clique. By the above discussion, if we branch by discarding v we can assume that we select at least two neighbors of v and therefore we have also to discard the mirrors $M(v)$ of v. In other terms, we can use the refined branching

$$\alpha(G) = \max\{\alpha(G - \{v\} - M(v)), 1 + \alpha(G - N[v])\}.$$

A *satellite* [5] of a node v is a node u at distance 2 from v such that there exists a node $u' \in N(v) \cap N(u)$ that satisfies $N[u'] - N[v] = \{u\}$. Observe that, if an optimum solution discards u, then we can discard v as well by dominance since $N[u'] \subseteq N[v]$ in $G - \{u\}$. Therefore we can assume that in the branch where we select v we also select its satellites $S(v)$. In other terms,

$$\alpha(G) = \max\{\alpha(G - \{v\}), 1 + |S(v)| + \alpha(G - N[v] - \cup_{u \in S(v)} N[u])\}.$$

Another useful trick [4] is to branch on nodes that form a *small separator* (of size 1 or 2 in the graph), hence isolating two or more connected components that can be solved independently (see also [2; 5]).

Measure and Conquer

Above we always used the number n of nodes as a *measure* of the size of subproblems. As observed in [3], much tighter running time bounds can be achieved by using smarted measures. As an illustration, we will present a refined bound on the running time of `Alg2`.

Let us measure the size of subproblems with the number n_3 of nodes of degree at least 3 (*large* nodes). Observe that, when $n_3 = 0$, G is a collection of isolated nodes, paths, and cycles. Therefore, in that case `Alg2` only applies reduction rules hence solving the problem in polynomial time. In other terms, $L(n_3) = L(0) = 1$ in this case. If the algorithm applies any reduction rule, the number of large nodes cannot increase and we obtain the trivial inequality $c^{n_3} \geq c^{n_3}$. Suppose next that `Alg2` performs a standard branching at a node v. Note that at this point all nodes in the graph are large. If $deg(v) \geq 4$, then we obtain the inequality $c^{n_3} \geq c^{n_3-1} + c^{n_3-5}$ which is satisfied by $c \geq 1.324\ldots$. Otherwise ($deg(v) = 3$), observe that the neighbors of v have degree 3 in G and at most 2 in $G - \{v\}$. Therefore the number of large nodes is at most $n_3 - 4$ in both subproblems $G - \{v\}$ and $G - N[v]$. This gives the inequality $c^{n_3} \geq 2c^{n_3-4}$ which is satisfied by $c \geq 2^{1/4} < 1.1893$. We can conclude that the running time of the

algorithm is in $O^*(1.325^n)$. In [3] each node is assigned a weight which is a growing function of its degree, and the measure is the sum of node weights (a similar measure is used also in [2; 5]).

In [2] it is shown how to use a fast MIS algorithm for graphs of maximum degree Δ to derive faster MIS algorithms for graphs of maximum degree $\Delta + 1$. Here the measure used in the analysis is a combination of the number of nodes and edges.

Memorization

So far we described algorithms with polynomial space complexity. *Memorization* [6] is a technique to speed up exponential-time branching algorithms at the cost of an exponential space complexity. The basic idea is to store the optimal solution to subproblems in a proper (exponential size) data structure. Each time a new subproblem is generated, one first checks (in polynomial time) whether that subproblem was already solved before. This way one avoids to solve the same subproblem several times.

In order to illustrate this technique, it is convenient to consider the variant `Alg3` of `Alg2` where we do not apply folding. This way, each subproblem corresponds to some induced subgraph $G[W]$ of the input graph. We will also use the standard measure though memorization is compatible with measure and conquer. By adapting the analysis of `Alg2`, one obtains the constraint $c^n \geq c^{n-1}+c^{n-3}$ and hence a running time in $O^*(1.466^n)$. Next consider the variant `Alg3mem` of `Alg3` where we apply memorization. Let $L_k(n)$ be the maximum number of subproblems on k nodes generated by `Alg3mem` starting from an instance with n nodes. A slightly adaptation of the standard analysis shows that $L_k(n) \leq 1.466^{n-k}$. However, since there are at most $\binom{n}{k}$ induced subgraphs on k nodes and we never solve the same subproblem twice, one also has $L_k(n) \leq \binom{n}{k}$. Using Stirling's formula, one obtains that the two upper bounds are roughly equal for $k = \alpha n$ and $\alpha = 0.107\ldots$. We can conclude that the running time of `Alg3mem` is in $O^*(\sum_{k=0}^{n} L_k(n)) = O^*(\sum_{k=0}^{n} \min\{1.466^{n-k}, \binom{n}{k}\}) = O^*(\max_{k=0}^{n} \min\{1.466^{n-k}, \binom{n}{k}\}) = O^*(1.466^{(1-0.107)n}) = O^*(1.408^n)$. The analysis can be refined [6] by bounding the number of *connected* induced subgraphs with k nodes in graphs of small maximum degree.

Recommended Reading

1. Beigel R (1999) Finding maximum independent sets in sparse and general graphs. In: ACM-SIAM Symposium on Discrete Algorithms (SODA), pp 856–857
2. Bourgeois N, Escoffier B, Paschos VT, van Rooij JMM (2012) Fast algorithms for max independent set. Algorithmica 62(1-2):382–415
3. Fomin FV, Grandoni F, Kratsch D (2009) A measure & conquer approach for the analysis of exact algorithms. Journal of the ACM 56(5)
4. Fürer M (2006) A faster algorithm for finding maximum independent sets in sparse graphs. In: Latin American Theoretical Informatics Symposium (LATIN), pp 491–501
5. Kneis J, Langer A, Rossmanith P (2009) A fine-grained analysis of a simple independent set algorithm. In: Foundations of Software Technology and Theoretical Computer Science (FSTTCS), pp 287–298
6. Robson JM (1986) Algorithms for maximum independent sets. Journal of Algorithms 7(3):425–440
7. Tarjan R, Trojanowski A (1977) Finding a maximum independent set. SIAM Journal on Computing 6(3):537–546