Moderately Exponential Time Algorithms

Dieter Kratsch

Laboratoire d'Informatique Théorique et Appliquée
Université Paul Verlaine - Metz
57000 Metz Cedex 01
France

LaBRI, Bordeaux, France
June 4, 2009

Polynomial time solvable vs. NP-completeness

- wanted : efficient algorithm to solve problem exactly
- however for many problems :
  - no polynomial-time algorithm known
  - no proof of non existence of a polynomial time algorithm
- NP-complete problems
- P=NP ? problem

How to attack NP-hard problems

Various techniques have been developped to attack NP-hard problems :

- approximation algorithms
- heuristics
- parameterized algorithms
- randomized algorithms
- restricted inputs
- moderately exponential time algorithms

ModEx Algorithms

- one of the oldest methods
- dating back to the early sixties
- Davis, Putnam (1960) and Held, Karp (1962)
- tries to cope with NP-completeness in the strongest sense
- analysis of worst-case running time

How good can exponential algorithms be ?

For all inputs of size n, can we reach worst-case running time

- $O(n!)$
- $O(2^{n^2})$
- $O(10^n)$
- $O(2^n)$
- $O(c^n)$ for some constant $c \geq 1$ sufficiently close to 1

or even a subexponential algorithm ? ?

Why Studying Exponential-Time Algorithms?

- certain applications require exact solutions of NP-hard problems
- approximation algorithms and fixed parameter algorithms are not always satisfactory
- reduction of the base of the exponential running time, say from $O(2^n)$ to $O(1.4^n)$, increases the size of the instances solvable within a given amount of time by a constant multiplicative factor
- running a given exponential algorithm on a faster computer can enlarge the mentioned size only by an additive factor
- leads to a better understanding of NP-hard problems
- initiates interesting new combinatorial and algorithmic challenges

Some Old Algorithms

Before 1990

TSP	$O^*(2^n)$	Held, Karp (1962)
COLORING	$O(2.4422^n)$	Lawler (1976)
3-COLORING	$O(1.4422^n)$	Lawler (1976)
3-SAT	$O(1.6181^n)$	Speckenmeyer, Monien (1985)
INDEPEDENT SET	$O(1.2108^n)$	Robson (1986)

Some New Algorithms

Last Decade

BANDWIDTH	$O^*(10^n)$	Feige, Kilian (2000)
3-SAT	$O(1.4802^n)$	Dantsin et al. (2002)
MAXIMUM CUT	$O(1.7315^n)$	Williams (ICALP 2004)
TREEWIDTH	$O(1.9601^n)$	Fomin et al. (ICALP 2004)
3-COLORING	$O(1.3289^n)$	Beigel, Eppstein (2005)
DOMINATING SET	$O(1.5137^n)$	Fomin et al. (ICALP 2005)
COLORING	$O^*(2^n)$	Björklund, Husfeldt (FOCS 2006)

I. Independence and Coloring

Independent Sets

Definition (Independent Set)

Let $G = (V, E)$ be a graph. A subset $I \subseteq V$ of vertices of G is an independent set of G if no two vertices in I are adjacent.

Definition (Maximum Independent Set (MIS))

Given a graph $G = (V, E)$, compute an maximum independent set of G.

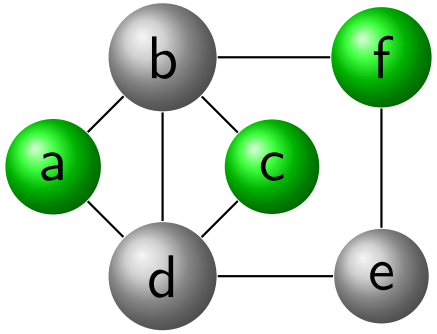
Independent Sets

Definition (Independent Set)

Let $G = (V, E)$ be a graph. A subset $I \subseteq V$ of vertices of G is an independent set of G if no two vertices in I are adjacent.

Definition (Maximum Independent Set (MIS))

Given a graph $G = (V, E)$, compute an maximum independent set of G.



Coloring

Definition (Coloring)

A function $c : V \to \{1, 2, ..., k\}$ is a proper coloring of a graph $G = (V, E)$ if $uv \in E$ implies $c(u) \neq c(v)$

Definition (Minimum Coloring)

Given a graph $G = (V, E)$, compute a coloring of G with the smallest number of colors.

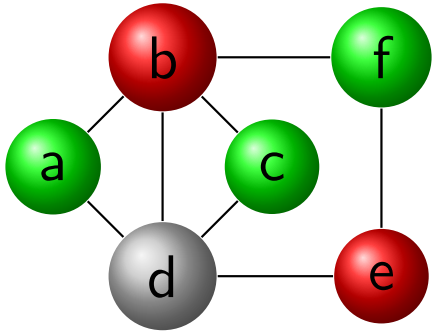
Coloring

Definition (Coloring)

A function $c : V \to \{1, 2, ..., k\}$ is a proper coloring of a graph $G = (V, E)$ if $uv \in E$ implies $c(u) \neq c(v)$

Definition (Minimum Coloring)

Given a graph $G = (V, E)$, compute a coloring of G with the smallest number of colors.



II. Dynamic Programming

Dynamic Programming

- Classical algorithm design technique
- $O(n^2\, 2^n)$ time algorithm for TSP [Held,Karp ; 1962]
- $O(2.4422^n)$ time algorithm for Coloring [Lawler ; 1967]

Lawler's Coloring Algorithm

Let $G = (V, E)$. For every $X \subseteq V$, compute $col(G, X) := \chi(G[X])$ by increasing cardinality of X.

$col(G, \emptyset) = 0$

$col(G, X) = 1 + \min\{col(G, X - S) :$
S maximal independent set of $G[X]\}$

Thus $\chi(G) = col(G, V)$.

Analysis of Running Time

A graph on i vertices has at most $3^{i/3}$ maximal independent sets and they can be listed in time $O^*(3^{i/3})$.

Thus up to a polynomial factor the running time is

$$\sum_{i=0}^{n} \binom{n}{i} 3^{i/3} = (1 + \sqrt[3]{3})^n$$

The running time of Lawler's coloring algorithm is $O(2.4422^n)$.

III. Iterative Compression

Iterative Compression

- New algorithm design technique
- incremental algorithm
- Introduced for parameterized algorithms
- Reed, Smith, Vetta [2004] : Odd cycle transversal

Notation

- input graph $G = (V, E)$
- $v_1, v_2, \ldots, v_n$ sequence of the vertices of G
- for all $i = 1, 2, \ldots, n$, $V_i = \{v_1, v_2, \ldots, v_i\}$ and $G_i = G[V_i]$
- B_i maximum independent set of G_i, i.e. $|B_i| = \alpha(G_i)$
- $A_i = V_i \setminus B_i$

Basic Approach

For $i = 1, 2, \ldots, n$, compute $\alpha(G_i)$ and a maximum independent set B_i of G_i. Thus B_n is a maximum independent set of $G_n = G$.

Compression (Expansion) : Given a maximum independent set B_i of G_i, compute a maximum independent set of $G_{i+1} = G_i + v_{i+1}$.

How are we making use of the maximum independent set B_i ?

Basic Approach

For $i = 1, 2, \ldots, n$, compute $\alpha(G_i)$ and a maximum independent set B_i of G_i. Thus B_n is a maximum independent set of $G_n = G$.

Compression (Expansion) : Given a maximum independent set B_i of G_i, compute a maximum independent set of $G_{i+1} = G_i + v_{i+1}$.

How are we making use of the maximum independent set B_i ?

Compression (Expansion)

Two possible cases

1. $\alpha(G_{i+1}) = \alpha(G_i)$ and $B_{i+1} = B_i$.

2. $\alpha(G_{i+1}) = \alpha(G_i) + 1$, $v_{i+1} \in B_{i+1}$. Thus $B_{i+1} = \{v_{i+1}\} \cup A' \cup B'$, where $A' \subseteq A_i$, $B' \subseteq B_i$ and A' independent set.

To find out whether $\alpha(G_{i+1}) = \alpha(G_i) + 1$, compute for every maximal independent set A' of $G[A_i - N[v_{i+1}]]$ in time $O(n^{2.5})$ a maximum independent set of the bipartite graph $G[(A' \cup B_i) - N[v_{i+1}]]$.

Speed up the Compression

Let $|A_i| = an$ and $|B_i| \leq (1-a)n$. Depending on the value of a :

1. Either check all at most $3^{an/3}$ maximal independent sets A' of $G[A_i]$
2. Or check all independent sets $A' \subseteq A_i - N[v_{i+1}]$ with $|A'| \leq |B_i|$

Running times (up to polynomial factor) :

1. $(3^{a/3})^n$
2. $\binom{an}{(1-a)n}$

Balancing with $a \approx 0.8$: MAXIMUM INDEPENDENT SET can be solved without Branching in time $O(1.3196^n)$.

IV. Inclusion Exclusion

Inclusion Exclusion Method

- Old method to design exponential-time algorithms
- Introduced by Karp 1982 (or earlier)
- Based on inclusion-exclusion principle of counting
- Solving optimization or decision problems by counting

Björklund-Husfeldt-Koivisto Algorithm

- U set of size n
- $\mathcal{S}$ family of subsets of U, enumerable in time $O^*(2^n)$
- $\mathcal{S}[X] = \{S \in \mathcal{S} : S \cap X = \emptyset\}$
- $s[X] = |\mathcal{S}[X]|$
- $\mathcal{S}^{(i)} = \{S \in \mathcal{S} : |S| = i\}$ subfamily of i-sets
- $s^{(i)}[X] = |\mathcal{S}^{(i)}[X]|$ number of i-sets avoiding X

Inclusion Exclusion Formula

For a positive integer k, let $c_k = c_k(\mathcal{S})$ denote the number of (possibly overlapping) k-covers, that is the number of ways to choose $S_1, S_2, \ldots, S_k \in \mathcal{S}$ with replacement such that

$$S_1 \cup S_2 \cup \cdots \cup S_k = U.$$

The number of k-covers c_k can be computed as follows

Inclusion Exclusion Formula

$$c_k = \sum_{X \subseteq U} (-1)^{|X|} s[X]^k$$

Computing all $s[X]$'s in time and space $O^*(2^n)$

Build a table with 2^n entries containing $s[X]$ for all $X \subseteq U$. Then evaluate the inclusion-exclusion formula in time $O^*(2^n)$.

- For instance, for every subset $W \subseteq U$ disjoint from X, let $s_W[X]$ denote the number of sets $S \in \mathcal{S}$ such that $W \subseteq S$ and $S \cap X = \emptyset$.
- $s_W[X] = s_W[X \cup \{v\}] + s_{W \cup \{v\}}[X]$ holds for all mutually disjoint X, W, and v
- compute $s[X] = s_\emptyset[X]$ recursively in time $O^*(2^n)$ and space $O^*(2^n)$

V. Branching Algorithms

Branching Algorithms

- one of the major techniques to design exponential-time algorithms
- typically polynomial space algorithm
- Davis, Putnam (1960); Tarjan, Trojanowski (1977)
- difficult to analyse the worst case running time

Branch & Reduce Algorithms

- Branch & Reduce algorithms
  (also called backtracking or search tree algorithms) :

  recursively applied to problem instances using Branching rules and Reduction rules.

  - Branching rules : solving the problem by recursively solving smaller instances
  - Reduction rules :
    - simplify the instance
    - (typically) reduce the size of the instance

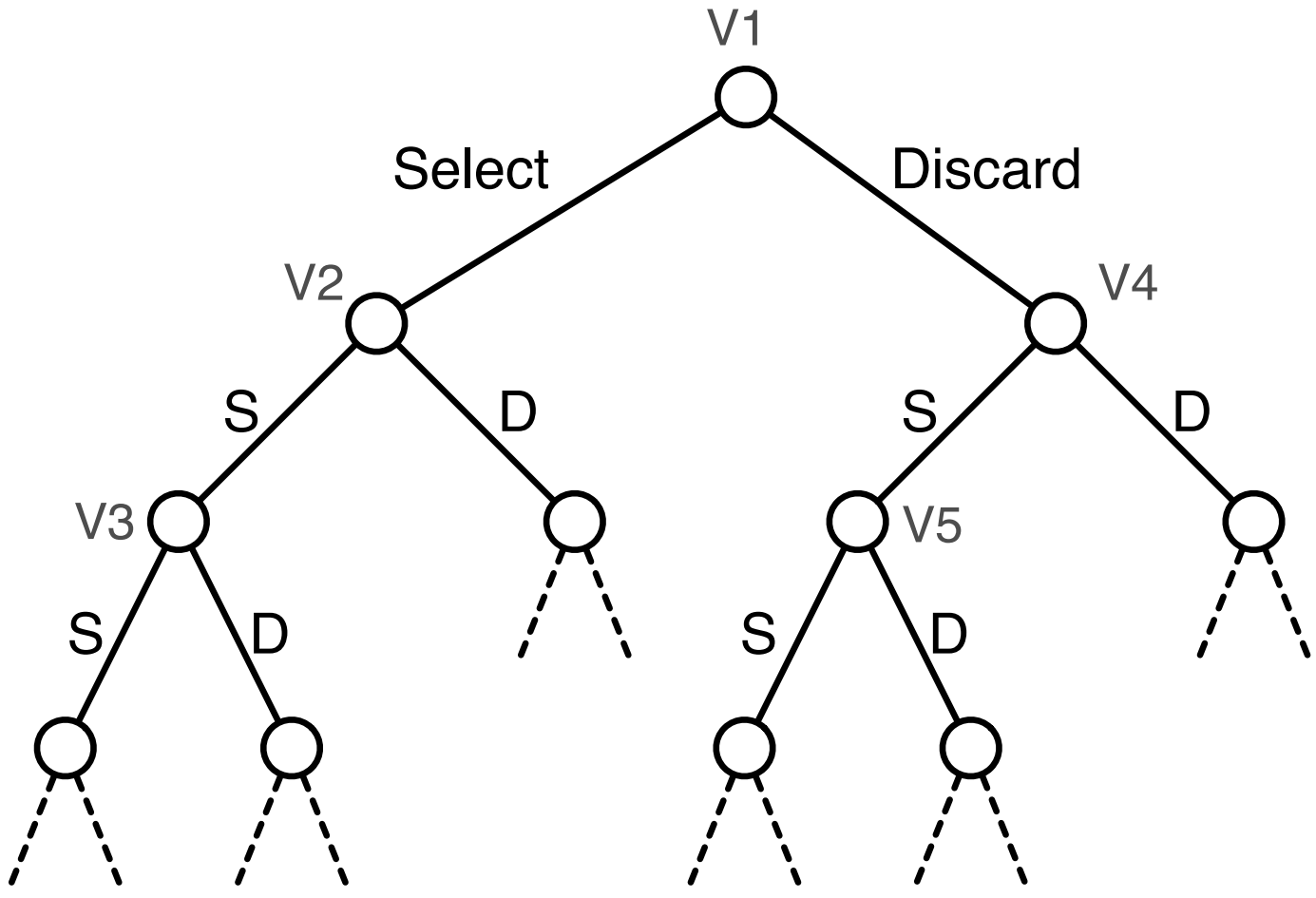
Search Trees

- Search Tree :

  used to illustrate and analyse an execution of a Branch & Reduce algorithm

  - nodes : assigns to each node a solved problem instance
  - root : assigns the input to the root
  - child : each instance (subproblem) reached by a branching rule is assigned to a child (of the node of the original problem)

A search tree



Analysis

- Analysing Branch & Reduce algorithms :

  Correctness and (Worst Case) Running Time

- Analysis of the Running Time :

  To obtain an Upper Bound on the maximum number of nodes of the search tree (for an input of size n) :

  1. Define a Measure for a problem instance.
  2. Lower bound the progress made by the algorithm at each branching step.
  3. Compute the collection of recurrences for all branching and reduction rules.
  4. Solve all those recurrences (to obtain a running time of the form $O(\alpha_i^n)$ for each).
  5. Take the worst case over all solutions.

  See the surveys [Woeginger (2003), Fomin et al. (2005)].

Analysis

- Analysing Branch & Reduce algorithms :

  Correctness and (Worst Case) Running Time

- Analysis of the Running Time :

  To obtain an Upper Bound on the maximum number of nodes of the search tree (for an input of size n) :

  1. Define a Measure for a problem instance.
  2. Lower bound the progress made by the algorithm at each branching step.
  3. Compute the collection of recurrences for all branching and reduction rules.
  4. Solve all those recurrences (to obtain a running time of the form $O(\alpha_i^n)$ for each).
  5. Take the worst case over all solutions.

  See the surveys [Woeginger (2003), Fomin et al. (2005)].

Analysis

- Analysing Branch & Reduce algorithms :

  Correctness and (Worst Case) Running Time

- Analysis of the Running Time :

  To obtain an Upper Bound on the maximum number of nodes of the search tree (for an input of size n) :

  1. Define a Measure for a problem instance.
  2. Lower bound the progress made by the algorithm at each branching step.
  3. Compute the collection of recurrences for all branching and reduction rules.
  4. Solve all those recurrences (to obtain a running time of the form $O(\alpha_i^n)$ for each).
  5. Take the worst case over all solutions.

  See the surveys [Woeginger (2003), Fomin et al. (2005)].

Analysis

- Analysing Branch & Reduce algorithms :

  Correctness and (Worst Case) Running Time

- Analysis of the Running Time :

  To obtain an Upper Bound on the maximum number of nodes of the search tree (for an input of size n) :

  1. Define a Measure for a problem instance.
  2. Lower bound the progress made by the algorithm at each branching step.
  3. Compute the collection of recurrences for all branching and reduction rules.
  4. Solve all those recurrences (to obtain a running time of the form $O(\alpha_i^n)$ for each).
  5. Take the worst case over all solutions.

  See the surveys [Woeginger (2003), Fomin et al. (2005)].

Analysis

- Analysing Branch & Reduce algorithms :

  Correctness and (Worst Case) Running Time

- Analysis of the Running Time :

  To obtain an Upper Bound on the maximum number of nodes of the search tree (for an input of size n) :

  1. Define a Measure for a problem instance.
  2. Lower bound the progress made by the algorithm at each branching step.
  3. Compute the collection of recurrences for all branching and reduction rules.
  4. Solve all those recurrences (to obtain a running time of the form $O(\alpha_i^n)$ for each).
  5. Take the worst case over all solutions.

  See the surveys [Woeginger (2003), Fomin et al. (2005)].

Analysis

- Analysing Branch & Reduce algorithms :

  Correctness and (Worst Case) Running Time

- Analysis of the Running Time :

  To obtain an Upper Bound on the maximum number of nodes of the search tree (for an input of size n) :

  1. Define a Measure for a problem instance.
  2. Lower bound the progress made by the algorithm at each branching step.
  3. Compute the collection of recurrences for all branching and reduction rules.
  4. Solve all those recurrences (to obtain a running time of the form $O(\alpha_i^n)$ for each).
  5. Take the worst case over all solutions.

  See the surveys [Woeginger (2003), Fomin et al. (2005)].

The Algorithm `sis`

```
1   int sis(G = (V, E)) {
2       if(|V| = 0) return 0;
3       choose a vertex v of minimum degree in G
4           return 1 + max{sis(G − N[y]) : y ∈ N[v]};
5   }
```

Analysis of the worst case running time

- Classical Analysis and Standard Measure
  (i.e. the measure of the input size is the number of vertices of the input graph).
- Recurrence :

$$T(n) \leq (d+1) \cdot T(n-d-1),$$

  where d is the degree of the chosen vertex v.
- Solution of recurrence : $O((d+1)^{n/(d+1)})$, being maximum for $d = 2$,
- Running time of `sis` : $O(3^{n/3})$.

Branching Algorithms for Maximum Independent Set

Polynomial Space Algorithms

$O(1.2600^n)$	Tarjan, Trojanowski (1977)	
$O(1.2346^n)$	Jian (1986)	
$O(1.2278^n)$	Robson (1986)	

Exponential Space Algorithms

$O(1.2109^n)$	Robson (1986)	
$O(1.1893^n)$	Robson (2001)	unpublished

The Maximum Independent Set Algorithm of Fomin et al.

Maximum Independent Set

- simple Branch & Reduce algorithm
- running time $O(1.2202^n)$
- polynomial space
- time analysis based on Measure and Conquer
- lower bound of $\Omega(1.1224^n)$ for the (unknown) worst case running time of the algorithm

the (known) running time of such Branch & Reduce algorithm might be "far" from the (unknown) worst case running time of the algorithm

The Maximum Independent Set Algorithm of Fomin et al.

Maximum Independent Set

- simple Branch & Reduce algorithm
- running time $O(1.2202^n)$
- polynomial space
- time analysis based on Measure and Conquer
- lower bound of $\Omega(1.1224^n)$ for the (unknown) worst case running time of the algorithm

the (known) running time of such Branch & Reduce algorithm might be "far" from the (unknown) worst case running time of the algorithm

The simple Branch & Reduce algorithm of Fomin et al.

int $\texttt{mis}(G)$ {

(0) **if** $(|V(G)| \leq 1)$ **return** $|V(G)|$;

(1) **if** $(\exists$ component $C \subset G)$

return $\texttt{mis}(C)+\texttt{mis}(G - C)$;

(2) **if** $(\exists$ nodes v and $w : N[w] \subseteq N[v])$

return $\texttt{mis}(G - \{v\})$;

(3) **if** $(\exists$ a node v with $d(v) = 2)$

return $1+\texttt{mis}(\widetilde{G}(v))$;

(4) select a node v of maximum degree, which minimizes $|E(N(v))|$;

return $\max\{\texttt{mis}(G - \{v\} - M(v)), 1+\texttt{mis}(G - N[v])\}$;

}

Folding a vertex of degree two

Suppose v has degree two and its neighbors u_1 and u_2 are non-adjacent. Folding the node v of G is the process of transforming G into a new graph $\widetilde{G} = \widetilde{G}(v)$ by :

(1) adding a new node u_{12} ;

(2) adding edges between u_{12} and the nodes in $N(u_1) \cup N(u_2)$;

(3) removing $N[v]$.

Lemma (Folding)

For any graph G and for any node v of G with degree two and two non-adjacent neighbors :

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

Folding a vertex of degree two

Suppose v has degree two and its neighbors u_1 and u_2 are non-adjacent. Folding the node v of G is the process of transforming G into a new graph $\widetilde{G} = \widetilde{G}(v)$ by :

(1) adding a new node u_{12} ;

(2) adding edges between u_{12} and the nodes in $N(u_1) \cup N(u_2)$;

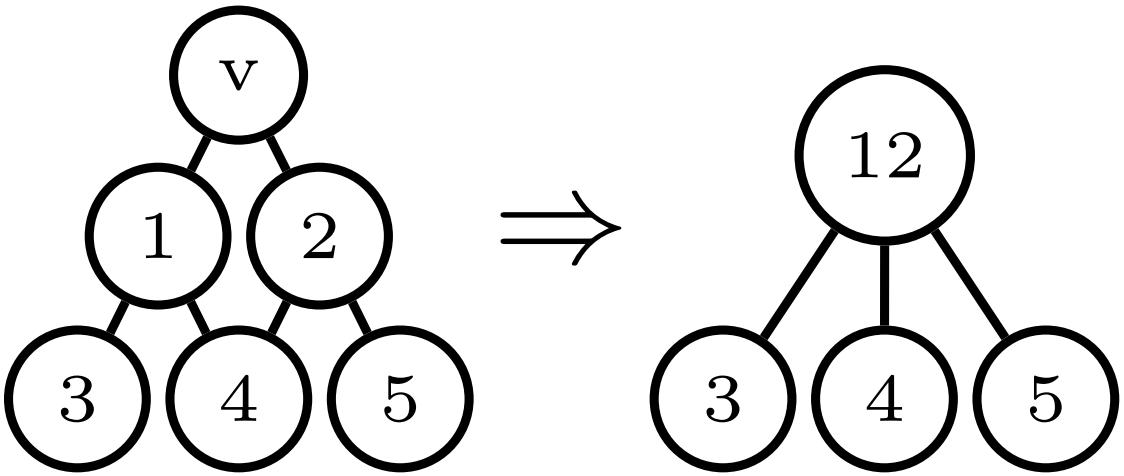
(3) removing $N[v]$.

Lemma (Folding)

For any graph G and for any node v of G with degree two and two non-adjacent neighbors :

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

An Example



Advantages and Disadvantages of Folding

Why using folding ?

The domination rule (2) and Folding (3) guarantee that the current graph has minimum degree 3 whenever the only branching rule (4) is applied.

Disadvantages of folding

Folding may increase the degree of the graph, and thus cannot be used for algorithms on graphs of bounded degree.
Folding makes it impossible to use Memorization.

Mirroring

Definition

Given a node v, a *mirror* of v is a node $u \in N^2(v)$ such that $N(v) \setminus N(u)$ is a (possibly empty) clique. We denote by $M(v)$ the set of mirrors of v.

When discarding a node v, one can also discard its mirrors as well without modifying the maximum independent set size.

Lemma (Mirroring)

For any graph G and for any node v of G :

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

Mirroring

Definition

Given a node v, a *mirror* of v is a node $u \in N^2(v)$ such that $N(v) \setminus N(u)$ is a (possibly empty) clique. We denote by $M(v)$ the set of mirrors of v.

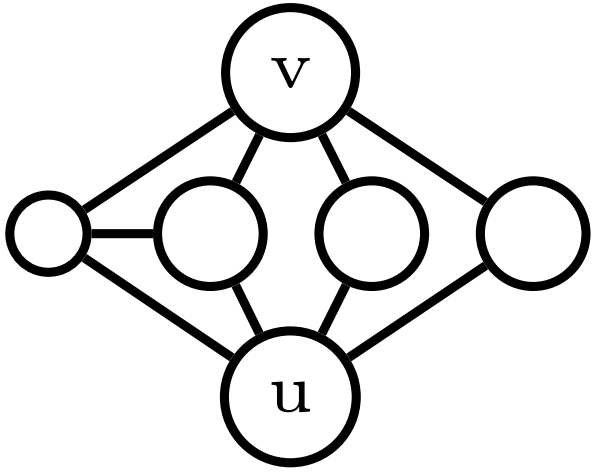
When discarding a node v, one can also discard its mirrors as well without modifying the maximum independent set size.

Lemma (Mirroring)

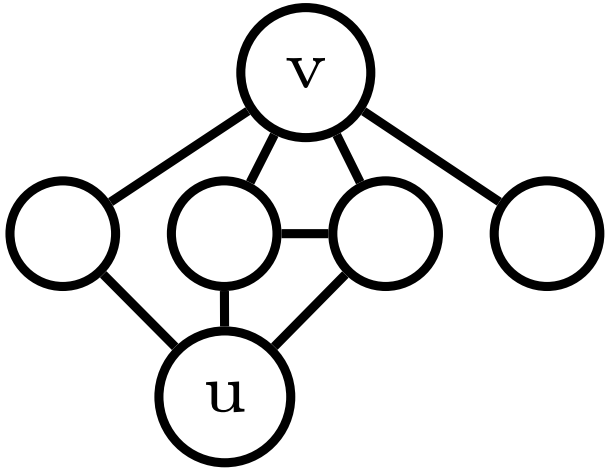
For any graph G and for any node v of G :

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

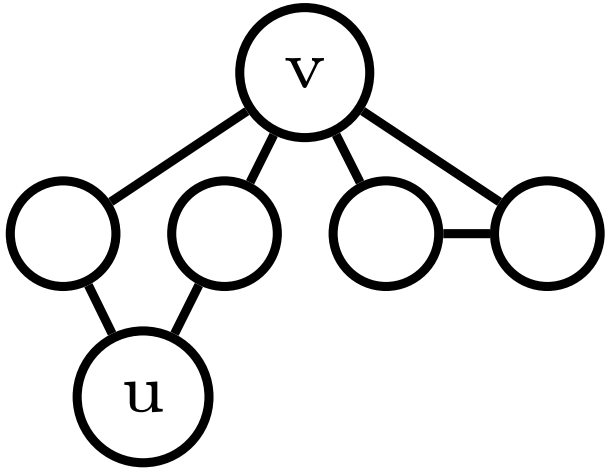
Some Examples



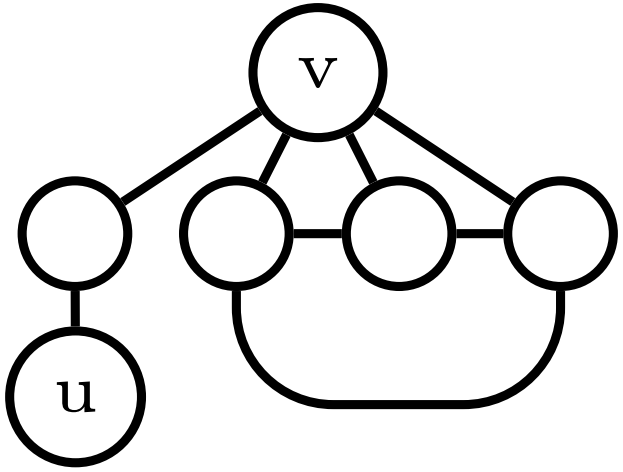
Some Examples



Some Examples



Some Examples



Measure & Conquer

A standard analysis of the worst case running time would establish running time $O(1.3251^n)$ for the algorithm. [Tarjan, Trojanowski $O(1.2600^n)$! !]

A better choice of the measure of a problem instance may allow to obtain a better upper bound on the worst case running time of the same algorithm.
Thus a nonstandard measure is a tool to improve the analysis of the worst case running time of Branch & Reduce algorithms.

Measure & Conquer

A standard analysis of the worst case running time would establish running time $O(1.3251^n)$ for the algorithm. [Tarjan, Trojanowski $O(1.2600^n)$! !]

A better choice of the measure of a problem instance may allow to obtain a better upper bound on the worst case running time of the same algorithm.
Thus a nonstandard measure is a tool to improve the analysis of the worst case running time of Branch & Reduce algorithms.

Choice of the Measure

$n_i :=$ number of nodes of degree i in G
$n_{\geq i} :=$ number of nodes of degree at least i in G

To analyze the maximum number of nodes of the search tree generated for an instance $G = (V, E)$, the following **measure** $k = k(G)$ is used :

Measure

$$k(G) = \sum_{i \geq 0} w_i \, n_i \leq n$$

The weights $w_i \in (0, 1]$ will be fixed such that the base of the exponential upper bound of the running time is minimized.
[A computer is needed to do this.]

Note that $k = k(G) \leq |V|$, hence $\alpha^k \leq \alpha^n$.

Constraints and Linear Recurrences

Analyzing the reduction rules (2) and (3)

leads to constraints for the choice of the w_i's

Due to the domination rule (2) and the folding rule (3), when we branch, we can assume that, for every node w :
(i) $deg(w) \geq 3$;
(ii) w is not dominated and does not dominate any other node.

Analyzing the main branching rule (4)

establishes a system of linear recurrences for any fixed choice of the w_i's

Analyzing the Main Branching Rule I

Suppose we branch at a given node v, with $N(v) = \{u_1, u_2, \ldots, u_d\}$ and $d_i = d(u_i)$.
We denote by $m_i = m_i(v)$ the number of nodes of degree i in $N(v)$:

$$m_i = |\{u \in N(v) \,:\, d(u) = i\}|.$$

Let $p_h = p_h(v)$ be the number of nodes in $N^2(v)$ which have exactly h neighbors in $N(v)$:

$$p_h = |\{w \in N^2(v) \,:\, |N(w) \cap N(v)| = h\}|.$$

Note that (at least) the nodes corresponding to p_{d-1} and p_d are mirrors of v. Additionally, the number of edges between $N(v)$ and $N^2(v)$ is $p = p(v) = \sum_{h=1}^{d} h\, p_h$.
We also define $m_{\geq i} = \sum_{j \geq i} m_j$ and $p_{\geq h} = \sum_{j \geq h} p_j$.

Analyzing the Main Branching Rule II

Let $P[k]$ be the maximum number of subproblems generated for an instance of measure k.

Reduction of the measure of the problem when discarding a vertex v :

$$\Delta_{OUT} = \Delta_{OUT}(v) \geq$$

$$w_d + p_d\, w_d + p_{d-1}\, w_{\max\{3,\, d-1\}} + \sum_{i=3}^{d} m_i\, \Delta\, w_i.$$

Reduction of the measure of a problem when selecting a vertex v :

$$\Delta_{IN} = \Delta_{IN}(v) \geq$$

$$w_d + \sum_{i=3}^{d} m_i w_i + p_1\, \Delta\, w_d + \sum_{h=2}^{d} p_h\, w_{\max\{3,h\}}.$$

System of Recurrences for the Main Branching Rule

System of Recurrences

For all m_i and p_h such that $\sum_{i=3}^{d} m_i = d$
and $\sum_{h=1}^{d} h\, p_h = d + m_3$:

$$P[k] \leq P[k - \Delta_{OUT}] + P[k - \Delta_{IN}].$$

Fortunately this can be restricted to a finite number of recurrences by arguing that we may assume $d \leq 7$.

Optimizing the Weights

Consider an assignment of the weights w_3, w_4, w_5, and w_6 which satisfies the initial assumptions and the constraints.
It turns out that the number of subproblems that need to be solved for a problem of measure k is $P[k] \leq \alpha^k$, where $\alpha > 1$ is the largest real root of the set of equations

$$\alpha^k = \alpha^{k-\Delta_{OUT}} + \alpha^{k-\Delta_{IN}}.$$

Thus the estimation of the running time reduces to choosing the weights w minimizing $\alpha = \alpha(w)$ (we refer to Eppstein's work on quasi-convex programming for a general treatment of such a problem).

Analysis of the Running Time

Tools

- Measure & Conquer based analysis of the running time
- program based on randomized local search to compute optimal values of the w_i's
- sophisticated analysis

Running Time

The algorithm of Fomin et al. has running time $O(1.2202^n)$. This is the best known running time of a polynomial space algorithm to compute a maximum independent set.

VI. Other Methods

Sort and Search

- Old method to design exponential-time algorithms
- Introduced by Horowitz, Sahni 1974
- Typical running time $O(2^{n/2}) = O(1.4142^n)$

Split and List

- New method to design exponential-time algorithms
- Introduced by Williams 2004
- Typical running time $O(2^{2.376\,n/3}) = O(1.7315^n)$
- Resembles Sort and Search

Treewidth Based Algorithms

- Technique for designing exponential-time algorithms
- mainly for special graph classes
- planar graphs, graphs of maximum degree 3

Example : $2^{n/6}$ time algorithm to compute a maximum independent set in graphs of degree at most 3

Open Questions

Fundamental open questions :

- polynomial vs. exponential space
- faster deterministic algorithms for 3-SAT
- better tools to analyze the running time of branching algorithms
- applications in enumerative combinatorics

Merci !

For Further Reading I


Fomin, F.V., F. Grandoni, and D. Kratsch,
Measure and conquer : Domination - A case study,
Proc. of ICALP 2005, *LNCS* **3380**, (2005), pp. 192–203.


Fomin, F.V., F. Grandoni, and D. Kratsch,
Some New Techniques in Design and Analysis of Exact (Exponential) Algorithms,
Bulletin of the EATCS, **87**, (2005), pp. 47–77.


Moon, J. W., and L. Moser,
On cliques in graphs,
Israel J. Math., **3**, (1965), pp. 23–28.


Robson, J. M.,
Finding a maximum independent set in time $O(2^{n/4})$,
Tech. Rep. 1251-01, LaBRI, Université Bordeaux I, 2001.


Tarjan, R.E., and A.E. Trojanowski,
Finding a maximum independent set,
SIAM Journal on Computing **6** (1977), pp. 537–546.

For Further Reading II

Williams, R.,
A new algorithm for optimal constraint 2-satisfaction and its implications,
Theoretical Computer Science **348** (2005), pp. 357-365.

Woeginger, G. J.,
Exact algorithms for NP-hard problems : A survey,
Combinatorial Optimization - Eureka, You Shrink !, *LNCS* **2570**, (2003), pp. 185–207.