Branching Algorithms

Dieter Kratsch

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

AGAPE 09, Corsica, France
May 24 - 29, 2009

I. Our First Independent Set Algorithm: `mis1`

Independent Set

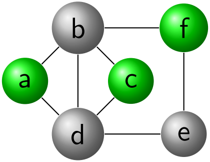
Definition (Independent Set)

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 the maximum cardinality of an independent set of G, denoted by $\alpha(G)$.

[or a maximum independent set of G]



Standard Branching Rule

- For every vertex v: "there is a maximum independent set containing v, or there is a maximum independent set not containing v"
- Branching into two smaller subproblems: "select v" and "discard v" to be solved recursively
- "discard v": remove v
- "select v": remove $N[v]$
- branching rule:

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

Algorithm `mis1`

int mis1$(G = (V, E))$;

$\{$

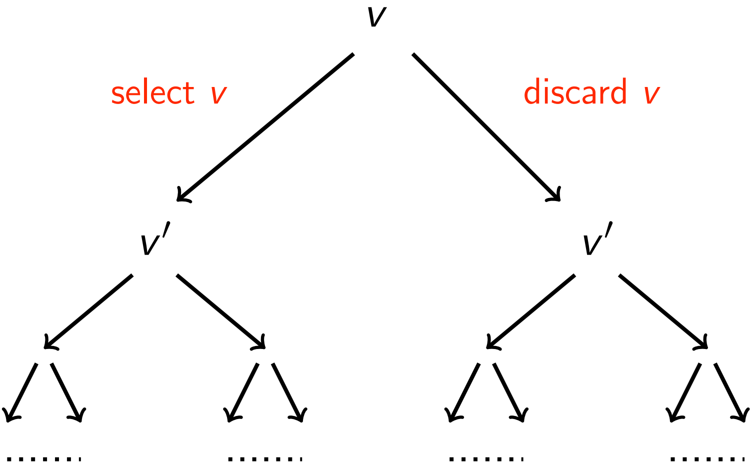
if $(\Delta(G) \geq 3)$ choose any vertex v of degree $d(v) \geq 3$

return $\max(1 + \text{mis1}(G - N[v]), \text{mis1}(G - v))$;

if $(\Delta(G) \leq 2)$ compute $\alpha(G)$ in polynomial time

and return the value;

$\}$



v
select v
discard v
v'
v'

Correctness

- standard branching rule correct; hence branching does not miss any maximum independent set
- graphs of maximum degree two are disjoint union of paths and cycles
- $\alpha(G)$ easy to compute if $\Delta(G) \leq 2$ [exercice]
- `mis1` outputs $\alpha(G)$ for input graph G
- `mis1` can be modified s.t. it outputs a maximum independent set

Time Analysis via recurrence

- Running time of `mis1` is $O^*(T(n))$, where
- $T(n)$ is largest number of base cases for any input graph G on n vertices
- Base case = graph of maximum degree two for which α is computed by a polynomial time algorithm
- branching rule implies recurrence:

$$T(n) \leq T(n-1) + T(n-d(v)-1) \leq T(n-1) + T(n-4)$$

Solving the Recurrence

- Solutions of recurrence of form c^n
- Basic solutions root of characteristic polynomial

$$x^n = x^{n-1} + x^{n-4}$$

- largest root of characteristic polynomial is its unique positive real root
- Maple, Mathematica, Matlab etc.

Running Time of `mis1`

Theorem: Algorithm `mis1` has running time $O^*(1.3803^n)$.

Question: Is this the worst-case running time of `mis1`? [Exercice]

II. Fundamental Notions and Time Analysis

Branching Algorithms

are also called

- branch & bound algorithms
- backtracking algorithms
- search tree algorithms
- branch & reduce algorithms
- splitting algorithms

The technique is also called "Pruning the search tree"

(e.g. in Woeginger's well-known survey).

Branching and Reduction Rules

Branching algorithms are recursively applied to instances of a problem using branching rules and reduction rules.

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

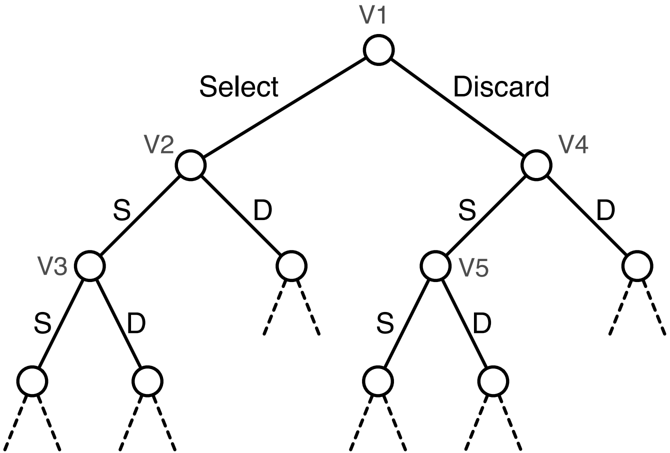
Search Trees

- Search Tree:

  used to illustrate, understand and analyse an execution of a branching algorithm

  - root: assign the input to the root
  - node: assign to each node a solved problem instance
  - child: each instance reached by a branching rule is assigned to a child of the node of the original instance of the problem

A search tree



Analysing a Branching Algorithm

- Correctness:

  Correctness of reduction and branching rules

- Running Time:

  Upper Bound the (maximum) number of leaves in any search tree of an input of size n:

  1. Define a size of 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 rules.
  4. Solve all those recurrences (to obtain a running time of the form $O^*(c_i^n)$ for each).
  5. Take the worst case over all solutions: $O^*(c^n)$ with $c = \max c_i$

Analysing a Branching Algorithm

- Correctness:

  Correctness of reduction and branching rules

- Running Time:

  Upper Bound the (maximum) number of leaves in any search tree of an input of size n:

  1. Define a size of 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 rules.
  4. Solve all those recurrences (to obtain a running time of the form $O^*(c_i^n)$ for each).
  5. Take the worst case over all solutions: $O^*(c^n)$ with $c = \max c_i$

Analysing a Branching Algorithm

- **Correctness**:

  Correctness of reduction and branching rules

- **Running Time**:

  Upper Bound the (maximum) number of leaves in any search tree of an input of size n:

  1. Define a size of 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 rules.
  4. Solve all those recurrences (to obtain a running time of the form $O^*(c_i^n)$ for each).
  5. Take the worst case over all solutions: $O^*(c^n)$ with $c = \max c_i$

Analysing a Branching Algorithm

- Correctness:

  Correctness of reduction and branching rules

- Running Time:

  Upper Bound the (maximum) number of leaves in any search tree of an input of size n:

  1. Define a size of 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 rules.
  4. Solve all those recurrences (to obtain a running time of the form $O^*(c_i^n)$ for each).
  5. Take the worst case over all solutions: $O^*(c^n)$ with $c = \max c_i$

Analysing a Branching Algorithm

- Correctness:

  Correctness of reduction and branching rules

- Running Time:

  Upper Bound the (maximum) number of leaves in any search tree of an input of size n:

  1. Define a size of 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 rules.
  4. Solve all those recurrences (to obtain a running time of the form $O^*(c_i^n)$ for each).
  5. Take the worst case over all solutions: $O^*(c^n)$ with $c = \max c_i$

Analysing a Branching Algorithm

- Correctness:

  Correctness of reduction and branching rules

- Running Time:

  Upper Bound the (maximum) number of leaves in any search tree of an input of size n:

  1. Define a size of 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 rules.
  4. Solve all those recurrences (to obtain a running time of the form $O^*(c_i^n)$ for each).
  5. Take the worst case over all solutions: $O^*(c^n)$ with $c = \max c_i$

Simple Time Analysis : Search Tree

- Assumption: for any node of search tree polynomial running time.
- Time analysis of branching algorithms means to upper bound the number of nodes of any search tree of an input of size n.
- Let $T(n)$ be (an upper bound of) the maximum number of leaves of any search tree of an input of size n.
- Running time of corresponding branching algorithm: $O^*(T(n))$
- Branching rules to be analysed separately

Simple Time Analysis : Branching Vectors

- Application of branching rule b to any instance of size n
- Problem branches into $r \geq 2$ subproblems of size at most $n - t_1$, $n - t_2$, ... , $n - t_r$ for all instances of size n
- $\vec{b} = (t_1, t_2, \ldots t_r)$ branching vector of branching rule b.

Simple Time Analysis: Recurrences

- Linear recurrence for the maximum number of leaves of a search tree corresponding to $\vec{b} = (t_1, t_2, \ldots t_r)$:

$$T(n) \leq T(n - t_1) + T(n - t_2) + \cdots + T(n - t_r).$$

- Largest solution of any such linear recurrence (obtained by a branching vector) is of form c^n where c is the unique positive real root of the characteristic polynomial:

$$x^n - x^{n-t_1} - x^{n-t_2} - \cdots - x^{n-t_r} = 0.$$

- This root $c > 1$ is called branching factor of $\vec{b}$:

$$\tau(t_1, t_2, \ldots, t_r) = c$$

Properties of Branching Vectors [Kullmann]

Let $r \geq 2$. Let $t_i > 0$ for all $i \in \{1, 2, \ldots r\}$.

1. $\tau(t_1, t_2, \ldots, t_r) \in (1, \infty)$.
2. $\tau(t_1, t_2, \ldots, t_r) = \tau(t_{\pi(1)}, t_{\pi(2)}, \ldots, t_{\pi(r)})$
   for any permutation π.
3. $\tau(t_1, t_2, \ldots, t_r) < \tau(t'_1, t_2, \ldots, t_r)$
   if $t_1 > t'_1$.

Balancing Branching Vectors

Let i, j, k be positive reals.

1. $\tau(k,k) \leq \tau(i,j)$ for all branching vectors (i,j) satisfying $i + j = 2k$.
2. $\tau(i,j) > \tau(i+\epsilon, j-\epsilon)$ for all $0 < i < j$ and $\epsilon \in (0, \frac{j-i}{2})$.

Example :

- $\tau(3,3) = \sqrt[3]{2} = 1.2600$
- $\tau(2,4) = \tau(4,2) = 1.2721$
- $\tau(1,5) = \tau(5,1) = 1.3248$

Some Factors of Branching Vectors

Compute a table with $\tau(i, j)$ for all $i, j \in \{1, 2, 3, 4, 5, 6\}$:

$$T(n) \leq T(n-i) + T(n-j) \quad \Rightarrow \quad x^n = x^{n-i} + x^{n-j}$$

$$x^j - x^{j-i} - 1 = 0$$

	1	2	3	4	5	6
1	2.0000	1.6181	1.4656	1.3803	1.3248	1.2852
2	1.6181	1.4143	1.3248	1.2721	1.2366	1.2107
3	1.4656	1.3248	1.2560	1.2208	1.1939	1.1740
4	1.3803	1.2721	1.2208	1.1893	1.1674	1.1510
5	1.3248	1.2366	1.1939	1.1674	1.1487	1.1348
6	1.2852	1.2107	1.1740	1.1510	1.1348	1.1225

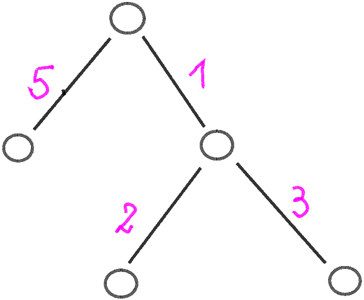
Addition of Branching Vectors

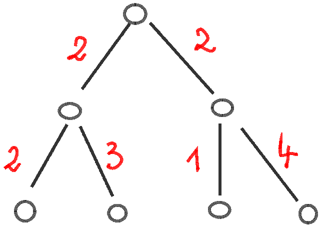
- "Sum up" consecutive branchings
- "sum" (overall branching vector) easy to find via search tree
- useful technique to deal with tight branching vector (i, j)

Example

- whenever algorithm (i, j)-branches it immediately (k, l)-branches on first subproblem
- overall branching vector $(i + k, i + l, j)$

Addition of Branching Vectors: Example





(5, 3, 4) (4, 5, 3, 6)

III. Preface

Branching algorithms

- one of the major techniques to construct FPT and ModEx Algorithms
- need only polynomial space
- major progress due to new methods of running time analysis
- many best known ModEx algorithms are branching algorithms

Challenging Open Problem

How to determine worst case running time of branching algorithms?

History: Before the year 2000

- Davis, Putnam (1960): SAT
- Davis, Logemann, Loveland (1962): SAT
- Tarjan, Trojanowski (1977): Independent Set
- Robson (1986): Independent Set
- Monien, Speckenmeyer (1985): 3-SAT

History: After the Year 2000

- Beigel, Eppstein (2005): 3-Coloring
- Fomin, Grandoni, Kratsch (2005): Dominating Set
- Fomin, Grandoni, Kratsch (2006): Independent Set
- Razgon; Fomin, Gaspers, Pyatkin (2006): FVS

IV. Our Second Independent Set Algorithm: `mis2`

Branching Rule

- For every vertex v:
  - "either there is a maximum independent set containing v,
  - or there is a maximum independent set containing a neighbour of v".
- Branching into $d(v) + 1$ smaller subproblems: "select v" and "select y" for every $y \in N(v)$
- Branching rule:

$$\alpha(G) = \max\{1 + \alpha(G - N[u]) : u \in N[v]\}$$

Algorithm `mis2`

int mis2$(G = (V, E))$;

$\{$

if $(|V| = 0)$ **return** 0;

choose a vertex v of minimum degree in G

return $1 + \max\{$ mis2$(G - N[y]) : y \in N[v]\}$;

$\}$

Analysis of the Running Time

- Input Size number n of vertices of 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:

$$O^*((d+1)^{n/(d+1)})$$

(maximum $d = 2$)
- Running time of `mis2`: $O^*(3^{n/3})$.

Enumerating all maximal independent sets I

Theorem :
Algorithm `mis2` enumerates all maximal independent sets of the input graph G in time $O^*(3^{n/3})$.

- to any leaf of the search tree a maximal independent set of G is assigned
- each maximal independent set corresponds to a leaf of the search tree

Corollary :
A graph on n vertices has $O^*(3^{n/3})$ maximal independent sets.

Enumerating all maximal independent sets II

Moon Moser 1962

The largest number of maximal independent sets in a graph on n vertices is $3^{n/3}$.

Papadimitriou Yannakakis 1984

There is a listing algorithm for the maximal independent sets of a graph having polynomial delay.

V. Our Third Independent Set Algorithm:

`mis3`

Contents

- History of branching algorithms to compute a maximum independent set
- Branching and reduction rules for Independent Set algorithms
- Algorithm `mis3`
- Running time analysis of algorithm `mis3`

History

Branching Algorithms for Maximum Independent Set

- $O(1.2600^n)$ Tarjan, Trojanowski (1977)
- $O(1.2346^n)$ Jian (1986)
- $O(1.2278^n)$ Robson (1986)
- $O(1.2202^n)$ Fomin, Grandoni, Kratsch (2006)

Domination Rule

Reduction rule: "If $N[v] \subseteq N[w]$ then remove w."

If v and w are adjacent vertices of a graph $G = (V, E)$ such that $N[v] \subseteq N[w]$, then

$$\alpha(G) = \alpha(G - w).$$

Proof by exchange:

If I is a maximum independent set of G such that $w \in I$ then $I - w + v$ is a maximum independent set of G.

Standard branching: "select v" and "discard v"

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

To be refined soon.

"Discard v" implies "Select two neighbours of v"

Lemma:

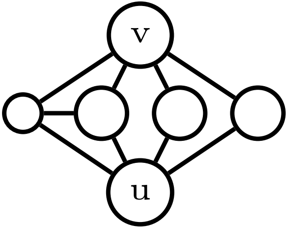
Let v be a vertex of the graph $G = (V, E)$. If no maximum independent set of G contains v then every maximum independent set of G contains at least two vertices of $N(v)$.

Proof by exchange: Assume no maximum independent set containing v.

- If I is a mis containing no vertex of $N[v]$ then $I + v$ is a mis, contraction.
- If I is a mis such that $v \notin I$ and $I \cap N(v) = \{w\}$, then $I - w + v$ is a mis of G, contradiction.

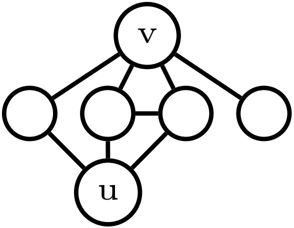
Mirrors

Let $N^2(v)$ be the set of vertices in distance 2 to v in G. A vertex $u \in N^2(v)$ is a mirror of v if $N(v) \setminus N(u)$ is a clique.



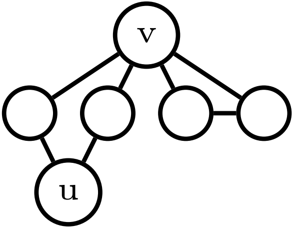
Mirrors

Let $N^2(v)$ be the set of vertices in distance 2 to v in G. A vertex $u \in N^2(v)$ is a mirror of v if $N(v) \setminus N(u)$ is a clique.



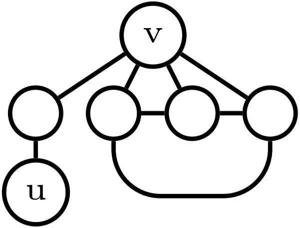
Mirrors

Let $N^2(v)$ be the set of vertices in distance 2 to v in G. A vertex $u \in N^2(v)$ is a mirror of v if $N(v) \setminus N(u)$ is a clique.



Mirrors

Let $N^2(v)$ be the set of vertices in distance 2 to v in G. A vertex $u \in N^2(v)$ is a mirror of v if $N(v) \setminus N(u)$ is a clique.



Mirror Branching

Mirror Branching: Refined Standard Branching

If v is a vertex of the graph $G = (V, E)$ and $M(v)$ the set of mirrors of v then

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

Proof by exchange: Assume no mis of G contains v

- By the lemma, every mis of G contains two vertices of $N(v)$.
- If u is a mirror then $N(v) \setminus N(u)$ is a clique; thus at least one vertex of every mis belongs to $N(u)$.
- Consequently, no mis contains u.

Simplicial Rule

Reduction Rule: Simplicial Rule

Let $G = (V, E)$ be a graph and v be a vertex of G such that $N[v]$ is a clique. Then

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

Proof:

Every mis contains v by the Lemma.

Branching on Components

Component Branching

Let $G = (V, E)$ be a disconnected graph and let C be a component of G. Then

$$\alpha(G) = \alpha(G - C) + \alpha(C).$$

Well-known property of the independence number $\alpha(G)$.

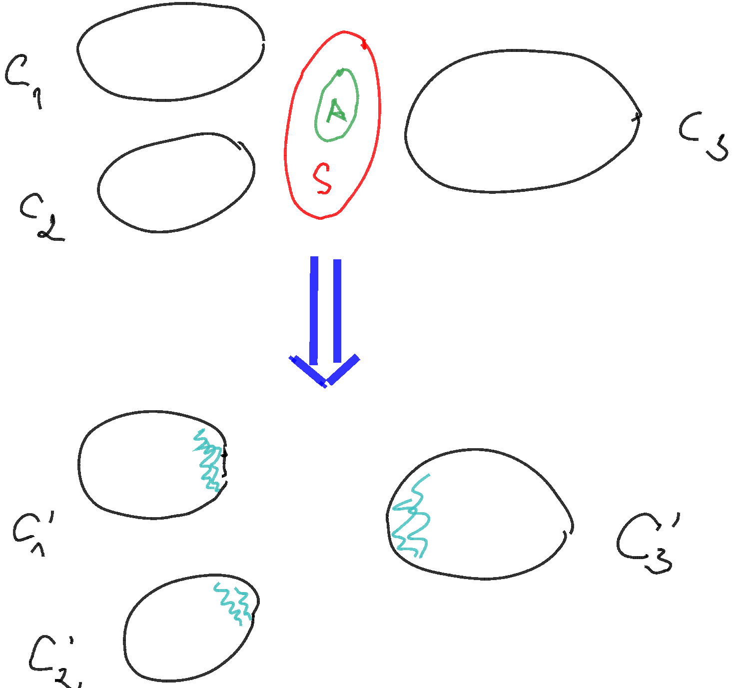
Separator branching

$S \subseteq V$ is a separator of $G = (V, E)$ if $G - S$ is disconnected.

Separator Branching: "Branch on all independent sets of separator S".

If S is a separator of the graph $G = (V, E)$ and $\mathcal{I}(S)$ the set of all independent subsets $I \subseteq S$ of G, then

$$\alpha(G) = \max_{A \in \mathcal{I}(S)} |A| \; + \; \alpha(G - (S \cup N[A])).$$



C1
C2
A
S
C3
C1'
C2'
C3'

Using Separator Branching

- separator S small, and
- easy to find.

`mis3` uses "separator branching on S" only if

- $S \subseteq N^2(v)$, and
- $|S| \leq 2$

Algorithm `mis3`: Small Degree Vertices

- minimum degree of instance graph G at most 3
- v vertex of minimum degree
- if $d(v)$ is equal to 0 or 1 then apply simplicial rule

(i) $\mathbf{d(v) = 0}$: "select v"; recursively call `mis3`$(G - v)$

(ii) $\mathbf{d(v) = 1}$: "select v"; recursively call `mis3`$(G - N[v])$

Algorithm `mis3`: Degree Two Vertices

- $\mathbf{d(v) = 2}$: u_1 and u_2 neighbors of v

  (i) $u_1 u_2 \in E$: $N[v]$ clique; simplicial rule: select v.
  call `mis3`$(G - N[v])$

  (ii) $u_1 u_2 \notin E$.

  $|\mathbf{N^2(v)}| = \mathbf{1}$: separator branching on
  $S = N^2(v) = \{w\}$
  branching vector $(|N^2[v] \cup N[w]|, |N^2[v]|)$, at least $(5, 4)$.

  $|\mathbf{N^2(v)}| \geq \mathbf{2}$: mirror branching on v
  branching vector $(N^2[v], N[v])$, at least $(5, 3)$.

Worst case for $d(v) = 2$:

$$\tau(5, 3) = 1.1939$$

Algorithm `mis3`: Degree Two Vertices

- $\mathbf{d(v) = 2}$: u_1 and u_2 neighbors of v

  (i) $u_1 u_2 \in E$: $N[v]$ clique; simplicial rule: select v.
  call $\texttt{mis3}(G - N[v])$

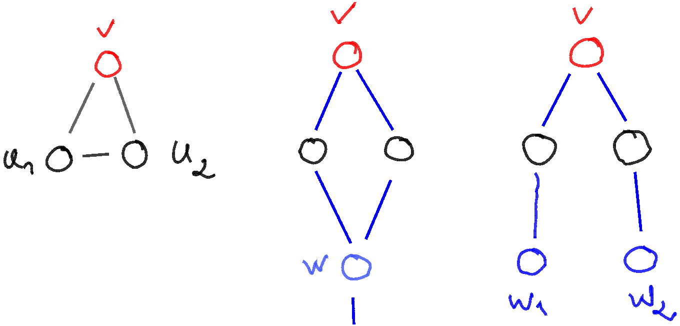
  (ii) $u_1 u_2 \notin E$.

  $|\mathbf{N^2(v)}| = \mathbf{1}$: separator branching on
  $S = N^2(v) = \{w\}$
  branching vector $(|N^2[v] \cup N[w]|, |N^2[v]|)$, at least $(5, 4)$.

  $|\mathbf{N^2(v)}| \geq \mathbf{2}$: mirror branching on v
  branching vector $(N^2[v], N[v])$, at least $(5, 3)$.

Worst case for $d(v) = 2$:

$$\tau(5, 3) = 1.1939$$


v
u_1
u_2
v
w
v
w_1
w_2

Analysis for $d(v) = 2$

$|\mathbf{N^2(v)}| = \mathbf{1}$: separator branching on $S = N^2(v) = \{w\}$

Subproblem 1: "select v and w" call $\texttt{mis3}(G - (N[v] \cup N[w]))$
Subproblem 2: "select u_1 and u_2"; call $\texttt{mis3}(G - N^2[v])$

Branching vector $(|N[v] \cup N[w]|, |N^2[v]|) \geq (5, 4)$.

$|\mathbf{N^2(v)}| \geq \mathbf{2}$: mirror branching on v

"discard v": select both neighbors of v, u_1 and u_2
"select" v": call $\texttt{mis3}(G - N[v])$

Branching vector $(|N^2[v]|, |N[v]|) \geq (5, 3)$

Algorithm `mis3`: Degree Three Vertices

$\mathbf{d(v) = 3}$: u_1, u_2 and u_3 neighbors of v in G.

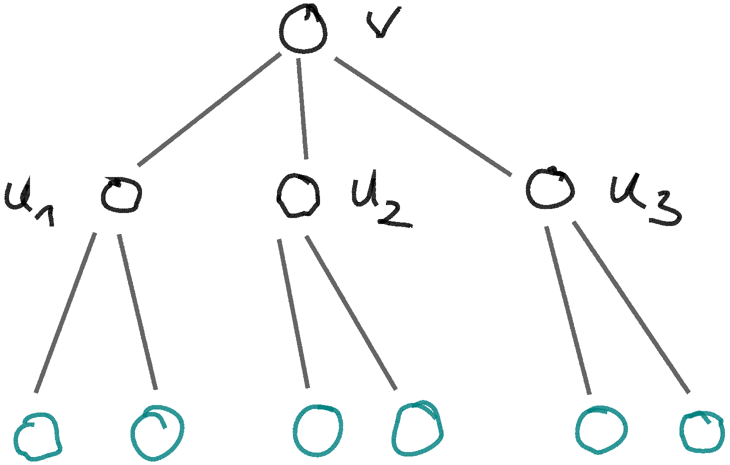
Four cases: $|E(N(v))| = 0, 1, 2, 3$

Case (i): $|E(N(v))| = 0$, i.e. $N(v)$ independent set.

every u_i has a neighbor in $N^2(v)$; else domination rule applies

Subcase (a): number of mirrors 0 [other subcases: 1 or 2]

- each vertex of $N^2(v)$ has precisely one neighbor in $N(v)$
- minimum degree of G at least 3, hence every u_i has at least two neighbors in $N^2(v)$


v
u1
u2
u3

$\mathbf{d(v) = 3}$, $N(v)$ independent set, v has no mirror

Algorithm branches into four subproblems:

- select v
- discard v, select u_1, select u_2
- discard v, select u_1, discard u_2, select u_3
- discard v, discard u_1, select u_2, select u_3

Branching vector $(4, 7, 8, 8)$ and $\tau(4, 7, 8, 8) = 1.2406$.

More subcases. More Cases. ...

Exercice:
Analyse the Subcases (b) and (c) of Case (i), and Case (ii).

Algorithm `mis3`: Degree Three Vertices

Case (iii): $|E(N(x))| = 2$.
$u_1 u_2$ and $u_2 u_3$ edges of $N(v)$.

Mirror branching on v:

"select v": call $\texttt{mis3}(G - N[v])$
"discard v": discard v, select u_1 and u_3

Branching factor $(4, 5)$ and $\tau(4, 5) = 1.1674$

Case (iv): $|E(N(x))| = 3$.
simplicial rule: "select v"

Worst case for $d(v) = 3$:

$$\tau(4, 7, 8, 8) = 1.2406$$

Algorithm `mis3`: Degree Three Vertices

Case (iii): $|E(N(x))| = 2$.
$u_1 u_2$ and $u_2 u_3$ edges of $N(v)$.

Mirror branching on v:

"select v": call $\texttt{mis3}(G - N[v])$
"discard v": discard v, select u_1 and u_3

Branching factor $(4, 5)$ and $\tau(4, 5) = 1.1674$

Case (iv): $|E(N(x))| = 3$.
simplicial rule: "select v"

Worst case for $d(v) = 3$:

$$\tau(4, 7, 8, 8) = 1.2406$$

Algorithm `mis3`: Large Degree Vertices

Maximum Degree Rule [$\delta(G) \geq 4$]

"Mirror Branching on a maximum degree vertex"

$\mathbf{d(v) \geq 6}$:

mirror branching on v

Branching vector $(d(v) + 1, 1) \geq (7, 1)$

Worst case for $d(v) \geq 6$:

$$\tau(7, 1) = 1.2554$$

Algorithm `mis3`: Large Degree Vertices

Maximum Degree Rule $[\delta(G) \geq 4]$

"Mirror Branching on a maximum degree vertex"

$\mathbf{d(v) \geq 6}$:

mirror branching on v

Branching vector $(d(v)+1, 1) \geq (7, 1)$

Worst case for $d(v) \geq 6$:

$$\tau(7, 1) = 1.2554$$

Algorithm `mis3`: Regular Graphs

Mirror branching on r-regular graph instances:

NOT TAKEN INTO ACCOUNT !

For every r, on any path of the search tree from the root to a leaf there is only one r-regular graph.

Algorithm `mis3`: Degree Five Vertices $\Delta = 5$ and $\delta = 4$

Mirror branching on a vertex v with a neighbor w s.t. $d(v) = 5$ and $d(w) = 4$

Case (i): v has a mirror:

Branching vector $(2, 6)$, $\tau(2, 6) = 1.2107$.

Case (ii): v has no mirror:

immediately mirror branching on w in $G - v$

$d(w) = 3$ in $G - v$: Worst case branching factor for degree three: $(4, 7, 8, 8)$ Adding branching vector to $(6, 1)$ sums up to $(5, 6, 8, 9, 9)$

Worst case for $d(v) = 5$:

$$\tau(5, 6, 8, 9, 9) = 1.2547$$

Algorithm `mis3`: Degree Five Vertices $\Delta = 5$ and $\delta = 4$

Mirror branching on a vertex v with a neighbor w s.t. $d(v) = 5$ and $d(w) = 4$

Case (i): v has a mirror:

Branching vector $(2, 6)$, $\tau(2, 6) = 1.2107$.

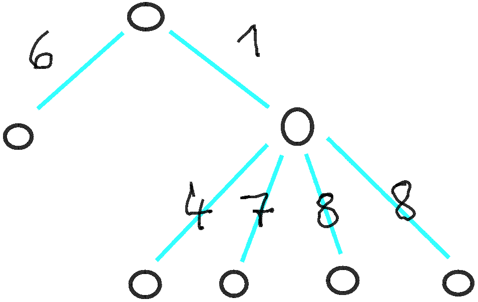
Case (ii): v has no mirror:

immediately mirror branching on w in $G - v$

$d(w) = 3$ in $G - v$: Worst case branching factor for degree three: $(4, 7, 8, 8)$ Adding branching vector to $(6, 1)$ sums up to $(5, 6, 8, 9, 9)$

Worst case for $d(v) = 5$:

$$\tau(5, 6, 8, 9, 9) = 1.2547$$



(6, 5, 8, 9, 9)

Running time of Algorithm `mis 3`

Theorem :
Algorithm `mis3` runs in time $O^*(1.2554^n)$.

Theorem :
The algorithm of Tarjan and Trojanowski has running time $O^*(2^{n/3}) = O^*(1.2600^n)$. [$O^*(1.2561^n)$]

VI. A DPLL Algorithm

The Satisfiability Problem of Propositional Logic

Boolean variables, literals, clauses, CNF-formulas

- A CNF-formula, i.e. a boolean formula in conjunctive normal form is a conjunction of clauses

$$F = (c_1 \wedge c_2 \wedge \cdots \wedge c_r).$$

- A clause

$$c = (\ell_1 \vee \ell_2 \vee \cdots \vee \ell_t)$$

  is a disjunction of literals.

- A k-CNF formula is a CNF-formula in which each clause consists of at most k literals.

Satisfiability

truth assignment, satisfiable CNF-formulas

- A truth assignment assigns boolean values (false, true) to the variables, and thus to the literals, of a formula.
- A CNF-formula F is satisfiable if there is a truth assignment such that F evaluates to true.
- A CNF-formula is satisfiable if each clause contains at least one true literal.

The Problems SAT and k-SAT

Definition (Satisfiability (SAT))

Given a CNF-formula F, decide whether F is satisfiable.

Definition (k-Satisfiability (k-SAT))

Given a k-CNF F, decide whether F is satisfiable.

$$F = (x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee x_3 \vee \neg x_4) \wedge (\neg x_2 \vee \neg x_3 \vee x_4)$$

Reduction and Branching Rules of a Classical DPLL algorithm

- Davis, Putnam 1960
- Davis, Logemann, Loveland (1962)

Reduction and Branching Rules

- [UnitPropagate] If all literals of a clause c except literal ℓ are false (under some partial assignment), then ℓ must be set to *true*.
- [PureLiteral] If a literal ℓ occurs pure in F, i.e. ℓ occurs in F but its negation does not occur, then ℓ must be set to *true*.
- [Branching] For any variable x_i, branch into "x_i *true*" and "x_i *false*".

VII. The algorithm of Monien and Speckenmeyer

Assigning Truth Values via Branching

- Recursively compute partial assignment(s) of given k-CNF formula F
- Given a partial truth assignment of F the corresponding k-CNF formula F' is obtained by removing all clauses containing a true literal, and by removing all false literals.
- Subproblem generated by the branching algorithm is a k-CNF formula
- Size of a k-CNF formula is its number of variables

The Branching Rule

Branching on a clause

- Branching on clause $c = (\ell_1 \vee \ell_2 \vee \cdots \vee \ell_t)$ of k-CNF formula F

- into t subproblems by fixing some truth values:

  - $F_1 : \quad \ell_1 = \text{true}$
  - $F_2 : \quad \ell_1 = \text{false}, \ \ell_2 = \text{true}$
  - $F_3 : \quad \ell_1 = \text{false}, \ \ell_2 = \text{false}, \ \ell_3 = \text{true}$

  - $F_t : \quad \ell_1 = \text{false}, \ \ell_2 = \text{false}, \ \cdots \ , \ell_{t-1} = \text{false}, \ \ell_t = \text{true}$

F is satisfiable iff at least one F_i, $i = 1, 2, \ldots, t$ is satisfiable.

Time Analysis I

- Assuming F consists of n variables then F_i, $i = 1, 2, \ldots, t$, consists of $n - i$ (non fixed) variables.

- Branching vector is $(1, 2, \ldots, t)$, where $t = |c|$.

- Solve linear recurrence
  $T(n) \leq T(n-1) + T(n-2) + \cdots + T(n-t)$.

- Compute the unique positive real root of

$$x^t = x^{t-1} + x^{t-2} + x^{t-3} + \cdots + 1 = 0$$

  which is equivalent to

$$x^{t+1} - 2x^t + 1 = 0.$$

Time Analysis II

For a clause of size t, let β_t be the branching factor.

Branching Factors: $\beta_2 = 1.6181$, $\beta_3 = 1.8393$, $\beta_4 = 1.9276$, $\beta_5 = 1.9660$, etc.

There is a branching algorithm solving 3-SAT in time $O^*(1.8393^n)$.

Speeding Up the Branching Algorithm

Observation: "The smaller the clause the better the branching factor."

Key Idea: Branch on a clause c of minimum size. Make sure that $|c| \leq k - 1$.

Halting and Reduction Rules:

- If $|c| = 0$ return "unsatisfiable".
- If $|c| = 1$ reduce by setting the unique literal *true*.
- If F is empty then return "satisfiable".

Monien Speckenmeyer 1985

For any $k \geq 3$, there is an $O^*({\beta_{k-1}}^n)$ algorithm to solve k-SAT.

3-SAT can be solved by an $O^*(1.6181^n)$ time branching algorithm.

Autarky: Key Properties

Definition

A partial truth assignment t of a CNF formula F is called autark if for every clause c of F for which the value of at least one literal is set by t, there is a literal ℓ_i of c such that $t(\ell_i) = \mathrm{true}$.

Let t be a partial assignment of F.

- t **autark:** Any clause c for which a literal is set by t is *true*. Thus F is satisfiable iff F' is satisfiable, where F' is obtained by removing all clauses c set *true* by t.

  $\Rightarrow$ reduction rule

- t **not autark:** There is a clause c for which a literal is set by t but c is not true under t. Thus in the CNF-formula corresponding to t clause c has at most $k-1$ literals.

  $\Rightarrow$ branch always on a clause of at most $k-1$ literals

VIII. Lower Bounds

Time Analysis of Branching Algorithms

Available Methods

- simple (or classical) time analysis
- Measure & Conquer, quasiconvex analysis, etc.
- based on recurrences

What can be achieved?

- establish upper bounds on the (worst-case) running time
- new methods achieve improved bounds for same algorithm
- no proof for tightness of bounds

Limits of Current Time Analysis

We cannot determine the worst-case running time of branching algorithms !

Consequences

- stated upper bounds of algorithms may (significantly) overestimate running times
- How to compare branching algorithms if their worst-case running time is unknown?

Limits of Current Time Analysis

We cannot determine the worst-case running time of branching algorithms !

Consequences

- stated upper bounds of algorithms may (significantly) overestimate running times
- How to compare branching algorithms if their worst-case running time is unknown?

We strongly need better methods for Time Analysis !

Better Methods of Analysis lead to Better Algorithms

Why study Lower Bounds of Worst-Case Running Time?

- Upper bounds on worst case running time of a Branching algorithms seem to overestimate the running time.
- Lower bounds on worst case running time of a particular branching algorithm can give an idea how far current analysis of this algorithm is from being tight.
- Large gaps between lower and upper bounds for some important branching algorithms.
- Study of lower bounds leads to new insights on particular branching algorithm.

Algorithm `mis1` Revisited

int $\text{mis1}(G = (V, E))$;

$\{$

if $(\Delta(G) \geq 3)$ choose any vertex v of degree $d(v) \geq 3$

  return $\max(1 + \text{mis1}(G - N[v]), \text{mis1}(G - v))$;

if $(\Delta(G) \leq 2)$ compute $\alpha(G)$ in polynomial time

  and return the value;

$\}$

Algorithm `mis1a`

int $\text{mis1}(G = (V, E))$;

$\{$

if $(\Delta(G) \geq 3)$ choose a vertex v of maximum degree

return $\max(1 + \text{mis1}(G - N[v]), \text{mis1}(G - v))$;

if $(\Delta(G) \leq 2)$ compute $\alpha(G)$ in polynomial time

and return the value;

$\}$

Algorithm `mis1b`

int mis1$(G = (V, E))$;

{

if there is a vertex v with $d(v) = 0$ **return** $1 + \text{mis1}(G - v)$;

if there is a vertex v with $d(v) = 1$ **return** $1 + \text{mis1}(G - N[v])$;

if $(\Delta(G) \geq 3)$ choose a vertex v of maximum degree

  return $\max(1 + \text{mis1}(G - N[v]), \text{mis1}(G - v))$;

if $(\Delta(G) \leq 2)$ compute $\alpha(G)$ in polynomial time

  and return the value;

}

Upper Bounds of Running time

Simple Running Time Analysis

- Branching vectors of standard branching: $(1, d(v) + 1)$
- Running time of algorithm `mis1`: $O^*(1.3803^n)$
- Running time of modifications `mis1a` and `mis1b`: $O^*(1.3803^n)$

Upper Bounds of Running time

Simple Running Time Analysis

- Branching vectors of standard branching: $(1, d(v) + 1)$
- Running time of algorithm `mis1`: $O^*(1.3803^n)$
- Running time of modifications `mis1a` and `mis1b`: $O^*(1.3803^n)$

Does all three algorithms have same worst-case running time ?

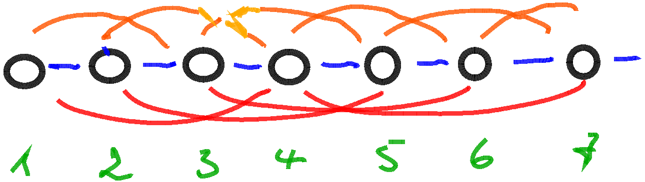
Related Questions

- What is the worst-case running time of these three algorithms on graphs of maximum degree three?
- How much can we improve the upper bounds of the running times of those three algorithms by Measure & Conquer?
- (Again) what is the worst-case running time of algorithm `mis1`?

A lower bound for `mis1`

Lower bound graph

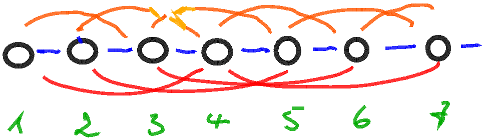
- Consider the graphs $G_n = (V_n, E_3)$
- Vertex set: $\{1, 2, \dots, n\}$
- Edge set: $\{i, j\} \in E_3 \Leftrightarrow |i - j| \leq 3$



Execution of `mis1` on the graph G_n

Tie breaks!

- Branch on smallest vertex of instance
- Always a vertex of degree three
- Every instance of form $G_n[\{i, i+1, \ldots, n\}]$
- Branching on instance $G_n[\{i, i+1, \ldots, n\}]$ calls `mis1` on $G_n[\{i+1, i+2, \ldots, n\}]$ and $G_n[\{i+4, i+5, \ldots, n\}]$



Recurrence for lower bound of worst-case running time:

$$T(n) = T(n-1) + T(n-4)$$

Theorem:

The worst-case running time of algorithm `mis1` is $\Theta^*(c^n)$, where $c = 1.3802...$ is the unique positive root of $x^4 - x^3 - 1$.

Exercice:

Determine lower bounds for the worst-case running time of `mis1a` and `mis1b`.

Algorithm `mis2` Revisited

int $\text{mis2}(G = (V, E))$;

$\{$

if $(|V| = 0)$ **return** 0;

choose a vertex v of minimum degree in G

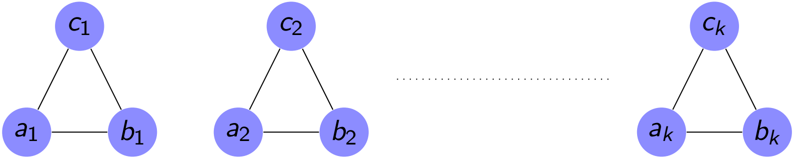
  return $1 + \max\{\ \text{mis2}(G - N[y]) : y \in N[v]\}$;

$\}$

Theorem:

The running time of algorithm `mis2` is $O^*(3^{n/3})$. Algorithm `mis2` enumerates all maximal independent sets of the input graph.

A lower bound for `mis2`



- Lower bound graph G_k: disjoint union of k triangles.
- Algorithm `mis2` applied to G_k: chooses a vertex of any triangle, branches into three subproblems G_{k-1}; (by removing a triangle from G_k)
- Search tree has $3^k = 3^{n/3}$ leaves;

Theorem:
The worst-case running time of algorithm `mis2` is $\Theta^*(3^{n/3})$.

The Algorithm tt of Tarjan and Trojanowski

- Algorithm `tt`:
  - Branching algorithm to compute a maximum independent set of a graph
  - published in 1977
  - lengthy and tedious case analysis
  - size of instance: number of vertices

- "Simple running time analysis": $O^*(2^{n/3}) = O^*(1.2600^n)$

- More precisely, author's analysis establishes $O^*(1.2561^n)$.

Important Properties of `tt`

Minimum Degree at most 4

If the minimum degree of the problem instance G is at most 4 then algorithm `tt` runs through plenty of cases.

Minimum Degree at least 5

Either G is 5-regular or algorithm `tt`
"chooses ANY vertex w of degree at least 6 and branches to
$G - N[w]$ (select w) and $G - w$ (discard w)".

Important Properties of `tt`

Minimum Degree at most 4

If the minimum degree of the problem instance G is at most 4 then algorithm `tt` runs through plenty of cases.

Minimum Degree at least 5

Either G is 5-regular or algorithm `tt`
"chooses ANY vertex w of degree at least 6 and branches to
$G - N[w]$ (select w) and $G - w$ (discard w)".

Lower bound graphs of minimum degree 6

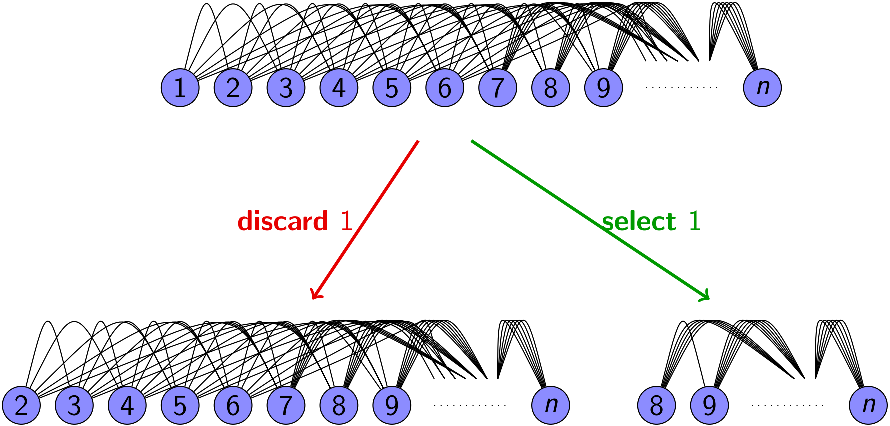
Lower Bound Graphs

- LB graphs: For all positive integers n, $G_n = (\{1, 2, \ldots, n\}, E_6)$, where

$$\{i, j\} \in E_6 \Leftrightarrow |i - j| \leq 6.$$

- Tie break: For graphs of minimum degree 6, the algorithm chooses smallest (resp. leftmost) vertex for branching.
- Branching "select[i]" removes $i, i+1, \ldots i+6$;
  "discard[i]" removes i;
  thus `tt` on G_n branches to G_{n-7} and G_{n-1}.

Branching



An Almost Tight Lower Bound

Definition

Let $T(n)$ be the number of leaves in the search tree obtained when executing algorithm `tt` on input graph G_n using the specified tie break rules.

Recurrence

$$T(n) = T(n-7) + T(n-1)$$

Lower Bound of `tt`

The running time of algorithm `tt` is $\Omega^*(1.2555^n)$.

An Almost Tight Lower Bound

Definition

Let $T(n)$ be the number of leaves in the search tree obtained when executing algorithm `tt` on input graph G_n using the specified tie break rules.

Recurrence

$$T(n) = T(n-7) + T(n-1)$$

Lower Bound of `tt`

The running time of algorithm `tt` is $\Omega^*(1.2555^n)$.

REMINDER: Upper Bound $O^*(1.2561^n)$.

Do we need lower bounds for other ModEx algorithms?

- Dynamic Programming
- Inclusion-Exclusion
- Treewidth Based
- Subset Convolution

Often claimed: "Our algorithm is faster on practical instances than its (worst case running) time we claim."

For branching algorithms the situation seems to be even better:

- faster than claimed running time on all instances
- hard to construct instances that even need a close running time
- "much better on many instances"?

IX. Memorization

Memorization: To be Used on Branching Algorithms

- GOAL: Reduction of running time of branching algorithms
- Use of exponential space instead of polynomial space
- Introduced by Robson (1986): Memorization for a MIS algorithm
- Theoretical Interest: allows to obtain branching algorithm of best running time for various well-studied NP-hard problems
- Practical Importance doubtful: high memory requirements

How does it work?

Basic Ideas

- Pruning the search tree: solve less subproblems
- Solutions of subproblems already solved to be stored in exponential-size database
- Solve subproblem once; when to be solved again, look up the solution in database
- query time in database logarithmic in number of stored solutions
- cost of each look up is polynomial.

Memorization can be applied to many branching algorithms

Once again Algorithm `mis1`

int mis1$(G = (V, E))$;

{

if $(\Delta(G) \geq 3)$ choose any vertex v of degree $d(v) \geq 3$

return $\max(1 + \text{mis1}(G - N[v]), \text{mis1}(G - v))$;

if $(\Delta(G) \leq 2)$ compute $\alpha(G)$ in polynomial time

and return the value;

}

Theorem:
Algorithm `mis1` has running time $O^*(1.3803^n)$ and uses only polynomial space.

Reduction of the Running Time of `mis1`

The algorithm

- Having solved an instance G', an induced subgraph of input graph G, store $(G', \alpha(G'))$ in a database.
- Before solving any instance, check whether its solution is already available in database.

- Input graph G has at most 2^n induced subgraphs.
- Database can be implemented such that each query takes time logarithmic in its size, thus polynomial in n.

Analysis of the Exponential Space algorithm

Upper bound of the running time of original polynomial space branching algorithm is needed to analyse the exponential space algorithm.

- Search tree of `mis1`(G) on any graph of n vertices has $T(n)$ leaves: $T(n) \leq 1.3803^n$.
- Let $T_h(n)$, $0 \leq h \leq n$, be the maximum number of subproblems of size h solved when calling `mis1`(G) for any graph of n.
- $T_h(n)$ maximum number of nodes of the subtree corresponding to an instance of h vertices.
- Similar to analysis of $T(n)$, one obtains:

$$T_h(n) \leq 1.3803^{n-h}$$

Balance to analyse I

To analyse the running time a balancing argument depending on the value of h is used.

How many instances of size h are solved?

- $T_h(n) \leq \binom{n}{h}$ since G has at most $\binom{n}{h}$ induced subgraphs on h vertices.
- $T_h(n) \leq 1.3803^{n-h}$

$$T_h(n) \leq \min\left(\binom{n}{h}, 1.3803^{n-h}\right)$$

Balance to analyse II

Balance both terms using Stirling's approximation:

For each h, $T_h(n) \leq 1.3803^{(1-\alpha)n} \leq 1.3424^n$ where $\alpha \geq 0.0865$ satisfies

$$1.3803^{1-\alpha} = \frac{1}{\alpha^\alpha (1-\alpha)^{1-\alpha}}$$

Theorem:

Memorization of algorithm `mis1` establishes an algorithm running in time $O^*(1.3424^n)$ needing exponential space.

X. Branch & Recharge

Another Way to Design and Analyse Branching Algorithms

- Using weights within the algorithm; not "only" as a tool in analysis
- GOAL: Easy time analysis
- In the best case: a few simple recurrences to solve
- Sophisticated correctness proof
- Time analysis (still) "recurrence based"

Framework: Initialisation

Initialisation

- First assign a weight of one to each vertex: $w(v) = 1$
- weight (resp. size) of input graph

$$w(G) = \sum_{v \in V} w(v) = |V| = n$$

Framework: Branching

Branching: just one rule

- Fix one branching rule: "select v" and "discard v"
- Fix a branching vector $(1, 1+\epsilon)$, $\epsilon > 0$
- Make sure that for each branching
  - "discard v": gain at least 1
  - "select v": gain at least $1+\epsilon$
- running time of algorithm: $O^*({c_\epsilon}^n)$
- c_ϵ unique positive real root of

$$x^{1+\epsilon} - x^{\epsilon} - 1 = 0$$

Framework: Recharging

Recharging

- When branching on a vertex v with $w(v) = 1$,
  set $w(v) = 0$ in both subproblems
- "select v": Borrow a weight of ϵ from a neighbour of v
- When branching on a vertex v with $w(v) < 1$
  Recharge the weight of v to 1, before branching on v

Generalized Domination problem

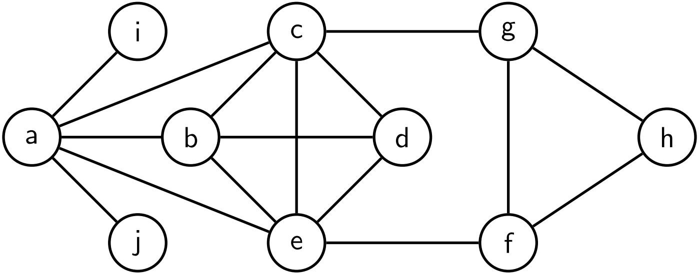
- also called (σ, ϱ)-Domination, where $\sigma, \varrho \subseteq \mathbb{N}$
- generalizes many domination-type problems

(σ, ϱ)-DOMINATING SET

Given a graph $G = (V, E)$, $S \subseteq V$ is a (σ, ϱ)-dominating set iff

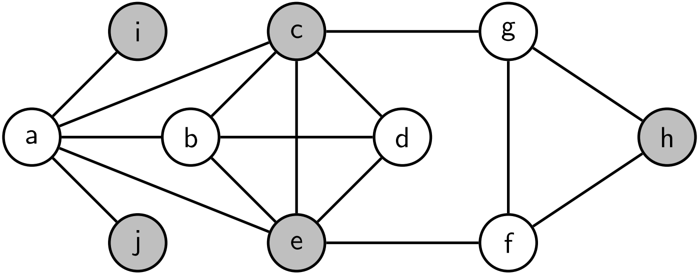
- for all $v \in S$, $|N(v) \cap S| \in \sigma$;
- for all $v \notin S$, $|N(v) \cap S| \in \varrho$.

An Example



Let $\sigma = \{0, 1\}$ and $\varrho = \{2, 4, 8\}$.

An Example



Let $\sigma = \{0, 1\}$ and $\varrho = \{2, 4, 8\}$.

Gray vertices form a (σ, ϱ)-Dominating Set.

σ and ϱ finite

Choice of ϵ

$$\epsilon_{p,q} = \frac{1}{\max(p, q)},$$

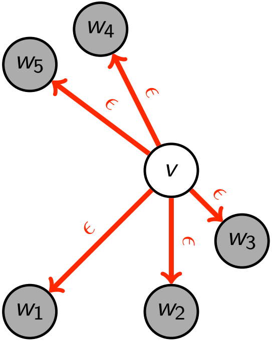
where $p = \max \sigma$ and $q = \max \rho$.

Example: Perfect Code

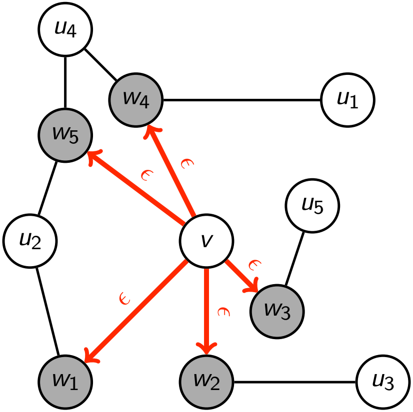
$\sigma = \{0\}$, and $\rho = \{1\}$.

$\epsilon = 1$.

Recharging

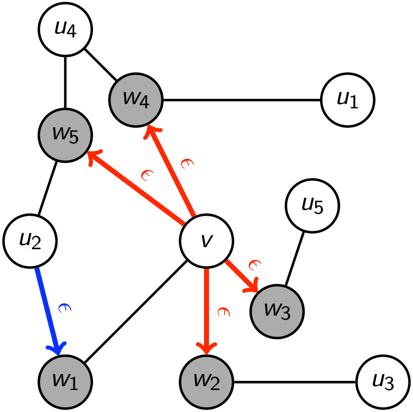


Recharging



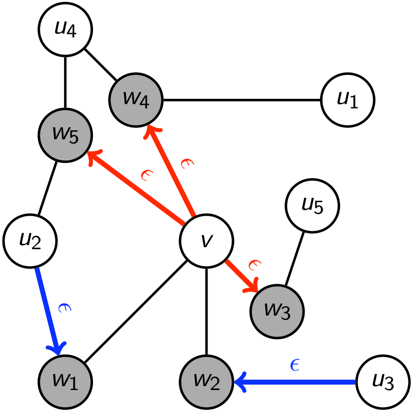
u_4
w_4
u_1
w_5
ϵ
ϵ
u_5
u_2
v
ϵ
ϵ
ϵ
w_3
w_1
w_2
u_3

Recharging



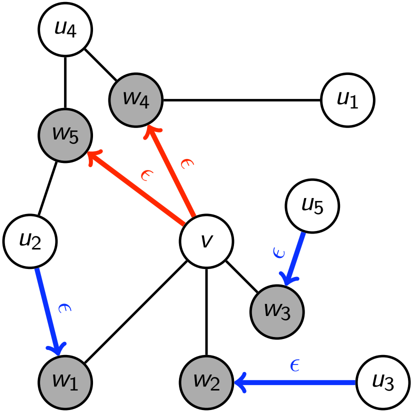
u_4
w_4
u_1
w_5
ϵ
ϵ
u_5
u_2
v
ϵ
ϵ
w_3
ϵ
w_1
w_2
u_3

Recharging



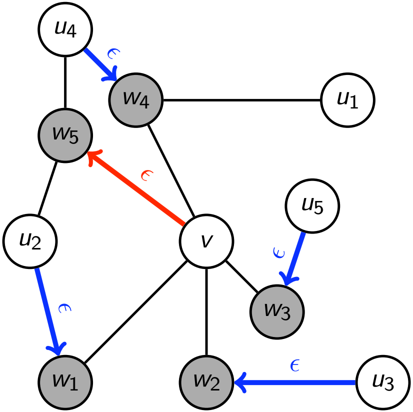
u_4
w_4
u_1
w_5
ϵ
ϵ
u_5
u_2
v
ϵ
w_3
ϵ
w_1
w_2
ϵ
u_3

Recharging



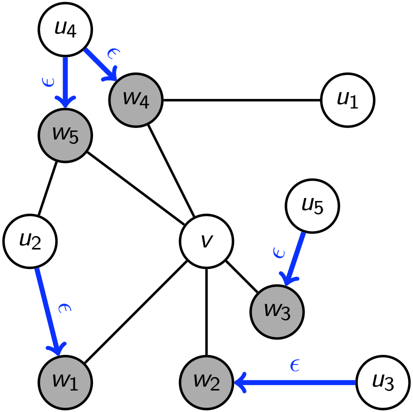
u_4
w_4
u_1
w_5
ϵ
ϵ
u_5
u_2
v
ϵ
ϵ
w_3
ϵ
w_1
w_2
u_3

Recharging



u_4
ϵ
w_4
u_1
w_5
ϵ
u_5
u_2
v
ϵ
ϵ
w_3
ϵ
w_1
w_2
u_3

Recharging



u_4
ϵ
ϵ
w_4
u_1
w_5
u_5
u_2
v
ϵ
ϵ
w_3
ϵ
w_1
w_2
u_3

We hope that Branch & Recharge will prove its potential as another method to design and analyse branching algorithms.

XI. Exercices

Exercices I

1. The HAMILTONIAN CIRCUIT problem can be solved in time $O^*(2^n)$ via dynamic programming or inclusion-exclusion. Construct a $O^*(3^{m/3})$ branching algorithm deciding whether a graph has a hamiltonian circuit, where m is the number of edges.

2. Let $G = (V, E)$ be a bicolored graph, i.e. its vertices are either red or blue. Construct and analyze branching algorithms that for input G, k_1, k_2 decide whether the bicolored graph G has an independent set I with k_1 red and k_2 blue vertices. What is the best running time you can establish?

3. Construct a branching algorithm for the 3-COLORING problem, i.e. for given graph G it decides whether G is 3-colorable. The running time should be $O^*(3^{n/3})$ or even $O^*(c^n)$ for some $c < 1.4$.

Exercices II

4. Construct a branching algorithm for the DOMINATING SET problem on graphs of maximum degree 3.

5. Is the following statement true for all graphs G: "If w is a mirror of v and there is a maximum independent set of G not containing v, then there is a maximum independent set containing neither v nor w."

6. Modify the first IS algorithm such that it always branches on a maximum degree vertex. Provide a lower bound (for `mis1a`). What is the worst-case running time of this algorithm?

Exercices III

7. Modify the first IS algorithm such that it uses a reduction rules on vertex of minimum degree, if it is 0 or 1, and if no such vertex exists it branches on a maximum degree vertex (of degree greater than three). Provide a lower bound (for `mis1b`). What is the worst-case running time of this algorithm?

8. Construct a $O^*(1.49^n)$ branching algorithm to solve 3-SAT.

More Exercices

Construct and analyse branching algorithms for the following problems:

- **Perfect Cover:** Given a graph, decide whether it has a vertex set I such that every vertex v of G belongs to precisely one neighbourhood set $N[u]$ for any $u \in I$.
- **Max 2-SAT:** Given a 2-CNF formula F, compute a truth assignment of its variables which maximizes the number of true clauses of F.
- **Weighted Independent Set:** Given a graph $G = (V, E)$ with positive integral vertex weights, compute a maximum weight independent set of G.

XII. For Further Reading

- T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein,
  Introduction to Algorithms,
  MIT Press, 2001.
- J. Kleinberg, E. Tardos,
  Algorithm Design,
  Addison-Wesley, 2005.
- R.L. Graham, D.E. Knuth, O. Patashnik
  Concrete Mathematics,
  Addison-Wesley, 1989.
- R. Beigel and D. Eppstein.
  3-coloring in time $O(1.3289^n)$.
  Journal of Algorithms 54:168–204, 2005.
- M. Davis, G. Logemann, and D.W. Loveland,
  A machine program for theorem-proving.
  Communications of the ACM 5:394–397, 1962.

M. Davis, H. Putnam.
A computing procedure for quantification theory.
Journal of the ACM 7:201–215, 1960.

D. Eppstein.
The traveling salesman problem for cubic graphs.
Proceedings of WADS 2003, pp. 307–318.

F. V. Fomin, S. Gaspers, and A. V. Pyatkin.
Finding a minimum feedback vertex set in time $O(1.7548^n)$,
Proceedings of IWPEC 2006, Springer, 2006, LNCS 4169, pp. 184–191.

F. V. Fomin, P. Golovach, D. Kratsch, J. Kratochvil and M. Liedloff.
Branch and Recharge: Exact algorithms for generalized domination.
Proceedings of WADS 2007, Springer, 2007, LNCS 4619, pp. 508–519.

F. V. Fomin, F. Grandoni, D. Kratsch.
Some new techniques in design and analysis of exact (exponential) algorithms.
Bulletin of the EATCS 87:47–77, 2005.

F. V. Fomin, F. Grandoni, D. Kratsch.
Measure and Conquer: A Simple $O(2^{0.288\,n})$ Independent Set Algorithm.
Proceedings of SODA 2006, ACM and SIAM, 2006, pp. 508–519.

K. Iwama.
Worst-case upper bounds for k-SAT.
Bulletin of the EATCS 82:61–71, 2004.

K. Iwama and S. Tamaki.
Improved upper bounds for 3-SAT.
Proceedings of SODA 2004, ACM and SIAM, 2004, p. 328.

O. Kullmann.
New methods for 3-SAT decision and worst case analysis.
Theoretical Computer Science 223: 1–72, 1999.

B. Monien, E. Speckenmeyer.
Solving satisfiability in less than 2^n steps.
Discrete Appl. Math. 10: 287–295, 1985.

I. Razgon.
Exact computation of maximum induced forest.
Proceedings of SWAT 2006, Springer, 2006, LNCS 4059, pp. 160–171.

J. M. Robson.
Algorithms for maximum independent sets.
Journal of Algorithms 7(3):425–440, 1986.

R. Tarjan and A. Trojanowski.
Finding a maximum independent set.
SIAM Journal on Computing, 6(3):537–546, 1977.

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

G. J. Woeginger.
Space and time complexity of exact algorithms: Some open problems.
Proceedings of IWPEC 2004, Springer, 2004, LNCS 3162, pp. 281–290.

G. J. Woeginger.
Open problems around exact algorithms.
Discrete Applied Mathematics, 156:397–405, 2008.