Lower Bounds on Exponential Time Algorithms

Dieter Kratsch

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

iETA 2006
Venice - Italy
July 16, 2006

Outline

Branch & Reduce Algorithms

- Techniques to design exact exponential-time algorithms :

  Enumeration, Dynamic Programming, Branch & Reduce etc.

- 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

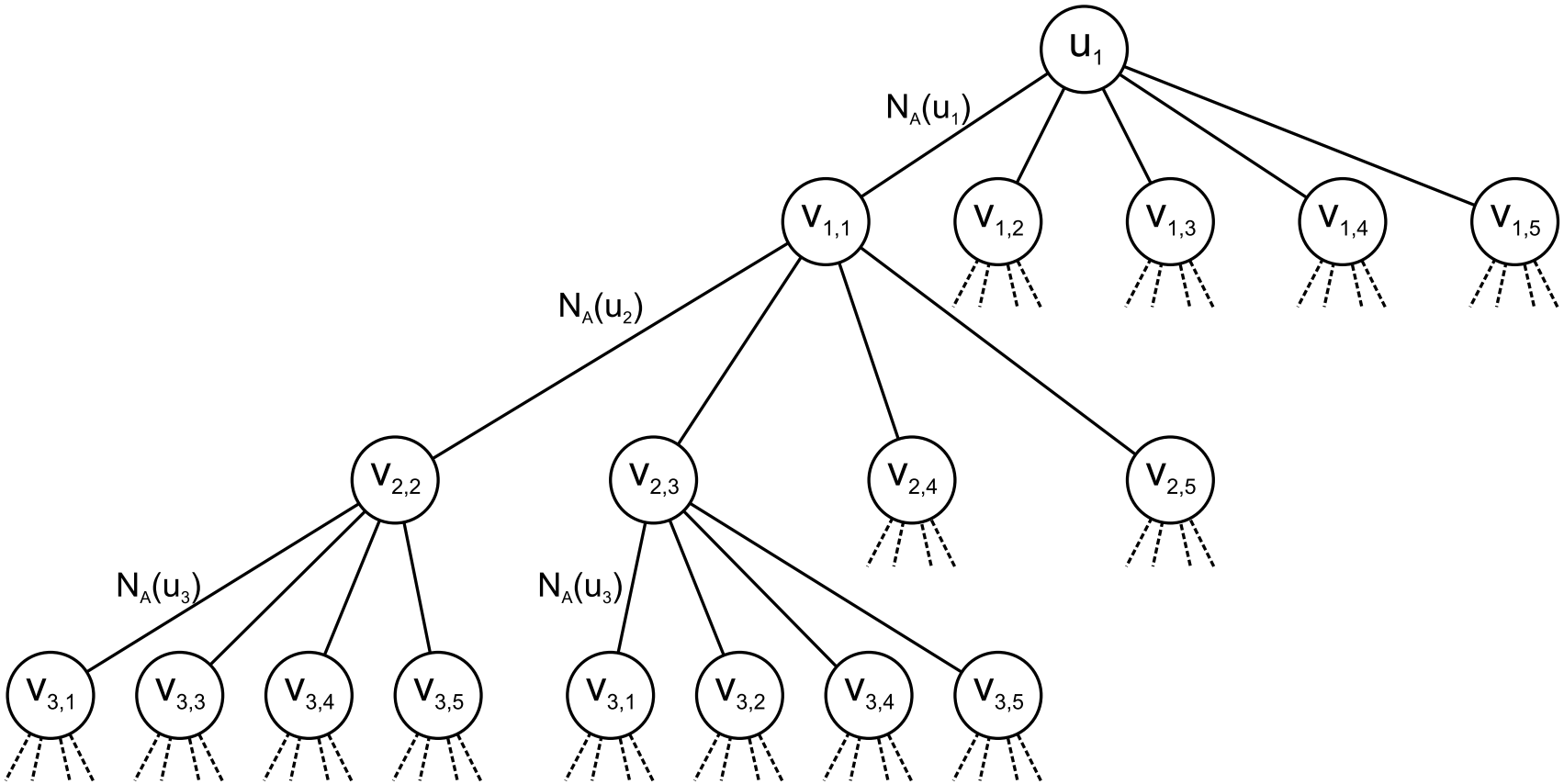
Search Trees

- 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, Fomin et al.].

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, Fomin et al.].

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, Fomin et al.].

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, Fomin et al.].

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, Fomin et al.].

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, Fomin et al.].

Lower Bounds

Lower Bounds

Lower Bounds ...

- of the computational complexity of problems,
- for solving a particular problem by any algorithm of a class of algorithms,
- of the worst case running time of a particular algorithm.

Known Results [Alekhnovich et al., Pudlak and Impaglazzio]

Exponential lower bounds for Davis-Putnam type algorithms (DPLL) on k-SAT and SAT

Lower Bounds

Lower Bounds ...

- of the computational complexity of problems,
- for solving a particular problem by any algorithm of a class of algorithms,
- of the worst case running time of a particular algorithm.

Known Results [Alekhnovich et al., Pudlak and Impaglazzio]

Exponential lower bounds for Davis-Putnam type algorithms (DPLL) on k-SAT and SAT

Why Study Lower Bounds ?

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

The simple Maximum Independent Set Algorithm `sis`

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.

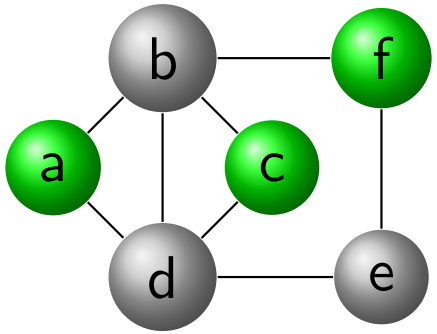
The simple Maximum Independent Set Algorithm `sis`

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.



The Algorithm `sis`

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

Analysis of Running Time (Upper Bound)

- 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` : $O(3^{n/3})$.

Tight Upper Bound !

Analysis of Running Time (Upper Bound)

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

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

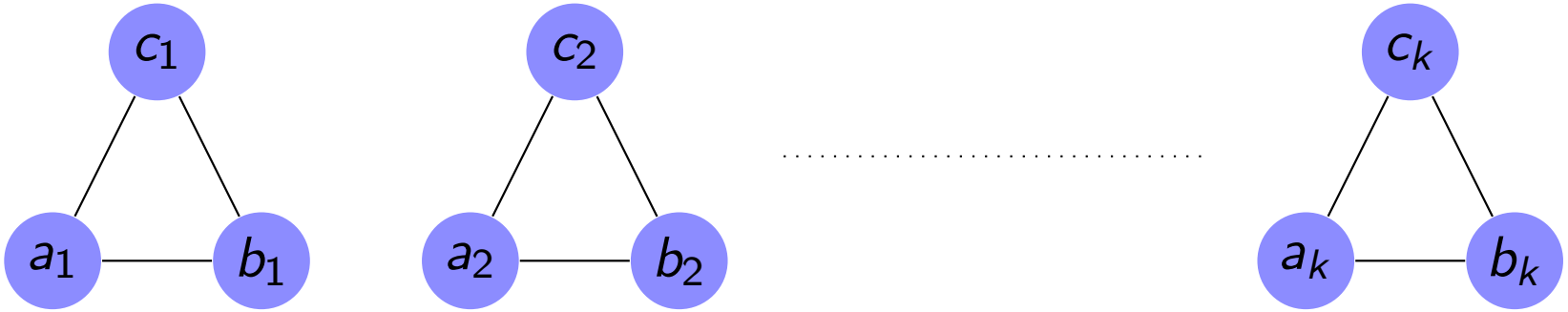
  where d is the degree of the chosen vertex v.
- Solution of recurrence : $O((d+1)^{n/(d+1)})$, being maximum for $d = 2$,
- Running time of `sis` : $O(3^{n/3})$.

Tight Upper Bound !

A Tight Lower Bound for Algorithm `sis`

Theorem

The algorithm `sis` computes a maximum independent set in time $\Omega(3^{n/3})$.



- Lower bound graph G_k : disjoint union of k triangles.
- Algorithm `sis` 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 ;

The Algorithm `tt` of Tarjan and Trojanowski

- Algorithm `tt` :
  - Branch & Reduce algorithm to compute a maximum independent set of a graph.
  - published in 1977
  - lengthy and tedious case analysis
- Classical analysis using standard measure : running time $O(2^{n/3})$.
- More precisely : authors analysis establishes $O(2^{0.3289n})$.

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)".

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 !

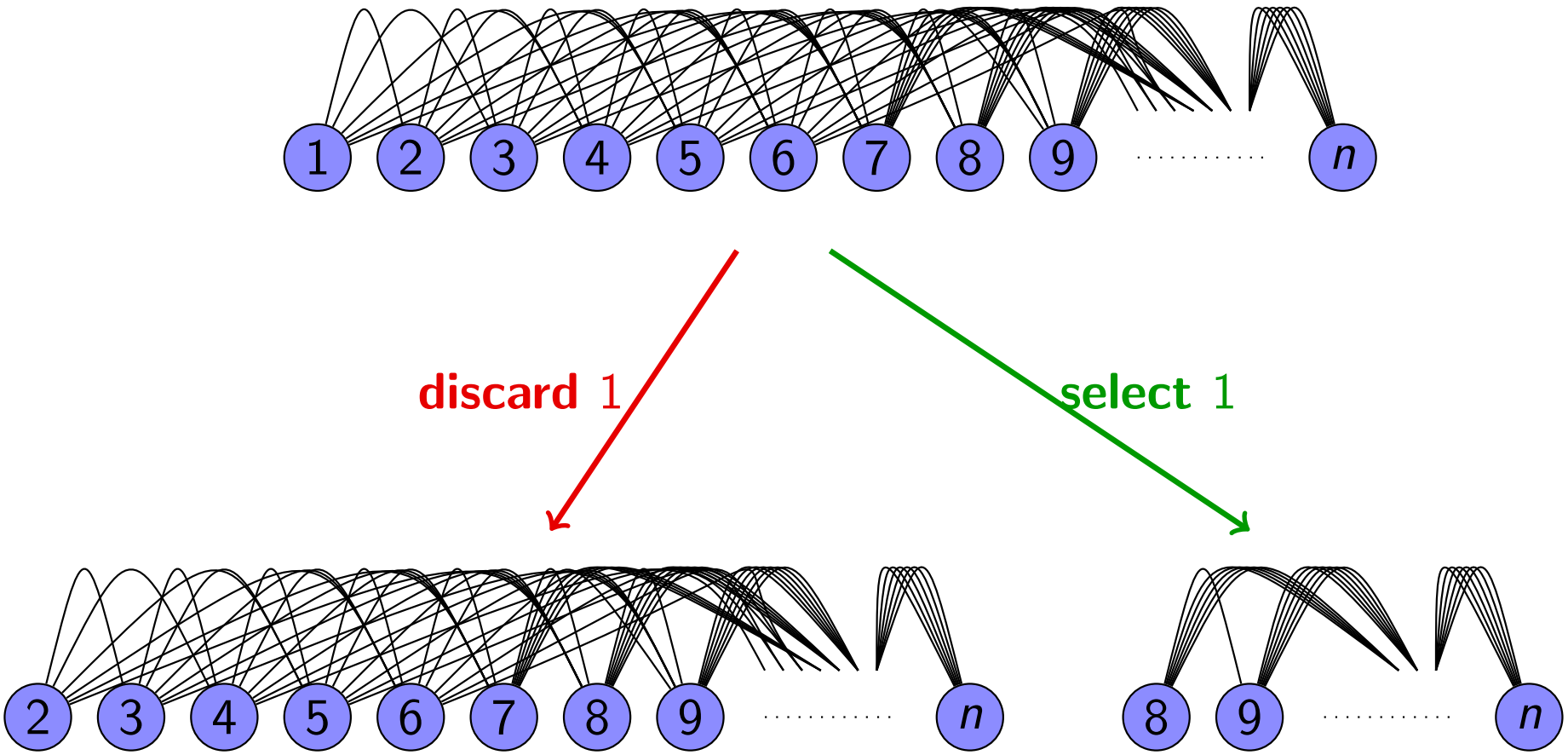
Lower Bound Graphs

- 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 $\Omega(2^{0.328173n})$.

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 $\Omega(2^{0.328173n})$.

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 $\Omega(2^{0.328173n})$.

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 $\Omega(2^{0.328173n})$.

REMINDER : Upper Bound $O(2^{0.3289n})$.

Why Not a Maximum Degree Rule ?

Minimum Degree at least 5

If the minimum degree of G is at least 5 the algorithm "chooses a vertex w of maximum degree and branches to $G - N[w]$ (select w) and $G - w$ (discard w)".

- No improvement of upper bound in classical analysis.
- Destroys our lower bound arguments.

Why Not a Maximum Degree Rule ?

Minimum Degree at least 5

If the minimum degree of G is at least 5 the algorithm "chooses a vertex w of maximum degree and branches to $G - N[w]$ (select w) and $G - w$ (discard w)".

- No improvement of upper bound in classical analysis.
- Destroys our lower bound arguments.

Why Not a Maximum Degree Rule ?

Minimum Degree at least 5

If the minimum degree of G is at least 5 the algorithm "chooses a vertex w of maximum degree and branches to $G - N[w]$ (select w) and $G - w$ (discard w)".

- No improvement of upper bound in classical analysis.
- Destroys our lower bound arguments.

Find upper and lower bounds for new algorithm !

The Independent Set Algorithm `bei` of Beigel

- Algorithm `bei` :
  - Branch & Reduce algorithm to compute a maximum independent set of a graph.
  - SODA 1999
  - uses Beigel's algorithm for sparse graphs
- claims running time $O(2^{0.303n})$.
- no analysis of upper bound available

Important Rules of `bei`

Sparse Graph Rule

If the problem instance G has at most $5n/2$ edges then algorithm `bei` uses Beigel's algorithm SparseFindMIS.

Domination Rule

If two adjacent vertices have comparable (by set inclusion) closed neighbourhoods then remove the (resp. one) vertex with largest neighbourhood.

Important Rules of `bei`

Sparse Graph Rule

If the problem instance G has at most $5n/2$ edges then algorithm `bei` uses Beigel's algorithm SparseFindMIS.

Domination Rule

If two adjacent vertices have comparable (by set inclusion) closed neighbourhoods then remove the (resp. one) vertex with largest neighbourhood.

Important Rules of `bei`

Maximum Degree Rule for Maximum Degree at least 8

If G has maximum degree at least 8 then choose a vertex w of maximum degree and branch to $G - N[w]$ (select w) and $G - w$ (discard w)".

How to construct lower bound graphs ?

Lower bound graphs of maximum degree 8 ...

... to avoid "second half of algorithm".

Make sure that reduction rules cannot be applied.

How to construct lower bound graphs ?

Lower bound graphs of maximum degree 8 ...

... to avoid "second half of algorithm".

Make sure that reduction rules cannot be applied.

How to construct lower bound graphs ?

Lower bound graphs of maximum degree 8 ...

... to avoid "second half of algorithm".

Make sure that reduction rules cannot be applied.

A Polynomial Example

LB graphs ?

For all positive integers n, $G_n = (\{1, 2, \ldots, n\}, E_8)$, where

$$\{i, j\} \in E_8 \Leftrightarrow |i - j| \leq 8.$$

- Minimum degree 8
- Algorithm `bei` solves MIS on G_n in polynomial time ; mainly due to domination rule.

A Polynomial Example

LB graphs ?

For all positive integers n, $G_n = (\{1, 2, \ldots, n\}, E_8)$, where

$$\{i, j\} \in E_8 \Leftrightarrow |i - j| \leq 8.$$

- Minimum degree 8
- Algorithm `bei` solves MIS on G_n in polynomial time ; mainly due to domination rule.

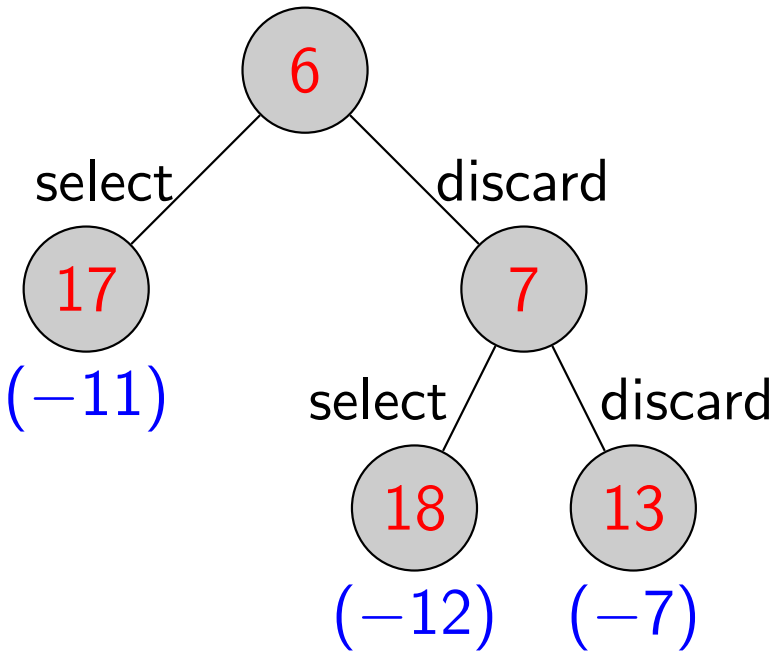
Lower Bound Graphs

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

$$\{i, j\} \in E \Leftrightarrow |i - j| \in \{2, 3, 4, 5\}.$$

- Tie break : For graphs of maximum degree 8, the algorithm chooses smallest (resp. leftmost) maximum degree vertex for branching.

A search tree



A Lower Bound

Definition

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

Recurrence

$$T(n) = T(n-7) + T(n-11) + T(n-12)$$

Lower Bound of `bei`

The running time of algorithm `bei` is $\Omega(2^{0.16297n})$.

A Lower Bound

Definition

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

Recurrence

$$T(n) = T(n-7) + T(n-11) + T(n-12)$$

Lower Bound of `bei`

The running time of algorithm `bei` is $\Omega(2^{0.16297n})$.

A Lower Bound

Definition

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

Recurrence

$$T(n) = T(n-7) + T(n-11) + T(n-12)$$

Lower Bound of `bei`

The running time of algorithm `bei` is $\Omega(2^{0.16297n})$.

A Lower Bound

Definition

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

Recurrence

$$T(n) = T(n-7) + T(n-11) + T(n-12)$$

Lower Bound of `bei`

The running time of algorithm `bei` is $\Omega(2^{0.16297n})$.

REMINDER : Upper Bound $O(2^{0.303n})$.

Better Lower Bounds

A more complicated construction seems to provide a lower bound of $\Omega(2^{0.17724n})$ for `bei`.

Find better lower bounds !

Better Lower Bounds

A more complicated construction seems to provide a lower bound of $\Omega(2^{0.17724n})$ for `bei`.

Find better lower bounds !

The Set Cover and Domination Algorithm of FGK

Definition (Dominating Set)

Let $G = (V, E)$ be a graph. A subset $D \subseteq V$ of vertices of G is a dominating set of G if every vertex of $V - D$ is adjacent to a vertex of D.

Definition (Minimum Dominating Set (MDS))

Given a graph $G = (V, E)$, compute a minimum dominating set of G.

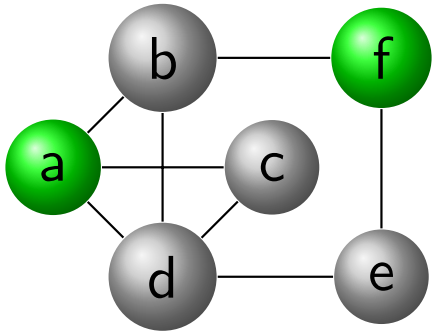
The Set Cover and Domination Algorithm of FGK

Definition (Dominating Set)

Let $G = (V, E)$ be a graph. A subset $D \subseteq V$ of vertices of G is a dominating set of G if every vertex of $V - D$ is adjacent to a vertex of D.

Definition (Minimum Dominating Set (MDS))

Given a graph $G = (V, E)$, compute a minimum dominating set of G.



Minimum Set Cover

Definition (Set Cover)

Let $\mathcal{U}$ be a universe of elements and let $\mathcal{S}$ be a collection of (non-empty) subsets of $\mathcal{U}$. A subset $\mathcal{S}' \subseteq \mathcal{S}$ is a set cover of $(\mathcal{U}, \mathcal{S})$ if every element of $\mathcal{U}$ belongs to a set of $\mathcal{S}'$, i.e. $\bigcup_{S \in \mathcal{S}'} S = U$.

Definition (Minimum Set Cover (MSC))

Given a universe $\mathcal{U}$ and a collection $\mathcal{S}$ of (non-empty) subsets of $\mathcal{U}$, compute a minimum set cover $\mathcal{S}'$ of $(\mathcal{U}, \mathcal{S})$.

Reducing Minimum Dominating Set to Minimum Set Cover

Reduction of MDS to MSC

MINIMUM DOMINATING SET for input graph $G = (V, E)$ reduces to MINIMUM SET COVER by setting $\mathcal{U} = V$ and $\mathcal{S} = \{N[v] | \ v \in V\}$.

Hence D is a minimum dominating set of $G = (V, E)$ iff $\mathcal{S}' = \{N[v] | \ v \in D\}$ is a minimum set cover of $(V, \{N[v] | \ v \in V\})$.

Reducing Minimum Dominating Set to Minimum Set Cover

Reduction of MDS to MSC

MINIMUM DOMINATING SET for input graph $G = (V, E)$ reduces to MINIMUM SET COVER by setting $\mathcal{U} = V$ and $\mathcal{S} = \{N[v] | \ v \in V\}$.

Hence D is a minimum dominating set of $G = (V, E)$ iff $\mathcal{S}' = \{N[v] | \ v \in D\}$ is a minimum set cover of $(V, \{N[v] | \ v \in V\})$.

The Algorithm msc

```
int msc(𝒮) {
    if(|𝒮| = 0) return 0;
    if(∃S, R ∈ 𝒮 : S ⊆ R) return msc(𝒮\{S});
    if(∃u ∈ 𝒰(𝒮)∃ a unique S ∈ 𝒮 : u ∈ S)
        return 1+msc(del(S, 𝒮));
    take S ∈ 𝒮 of maximum cardinality;
    if(|S| = 2) return poly-msc(𝒮)
    return min{msc(𝒮\{S}), 1+msc(del(S, 𝒮))};
}
```

Reduction and Branching Rules

Reduction Rule I

If two sets $R, S \in \mathcal{S}$ are comparable by set inclusion then remove the smaller one from $\mathcal{S}$.

Reduction Rule II

If an element u belongs to precisely one set $S \in \mathcal{S}$ then "select S". (i.e. removal of S from $\mathcal{S}$, and removal of each element of S from the universe)

Reduction and Branching Rules

Reduction Rule I

If two sets $R, S \in \mathcal{S}$ are comparable by set inclusion then remove the smaller one from $\mathcal{S}$.

Reduction Rule II

If an element u belongs to precisely one set $S \in \mathcal{S}$ then "select S". (i.e. removal of S from $\mathcal{S}$, and removal of each element of S from the universe)

Reduction and Branching Rules

Branching Rule (Maximum Size Rule)

Choose a set S of $\mathcal{S}$ of maximum size and branch on S : "select S", "discard S".

Reduction and Branching Rules

Branching Rule (Maximum Size Rule)

Choose a set S of $\mathcal{S}$ of maximum size and branch on S : "select S", "discard S".

Preferences of Rules are Crucial !

Analysis of Running Time (Upper Bound)

Measure & Conquer

Sophisticated running time analysis using non standard measure

Upper Bound for `msc`

Algorithm `msc` solves MINIMUM SET COVER in time $O(2^{0.305(|\mathcal{U}|+|\mathcal{S}|)})$.

Analysis of Running Time (Upper Bound)

Measure & Conquer

Sophisticated running time analysis using non standard measure

Upper Bound for `msc`

Algorithm `msc` solves MINIMUM SET COVER in time $O(2^{0.305(|\mathcal{U}|+|\mathcal{S}|)})$.

Analysis of Running Time (Upper Bound)

Using `msc`

Combine reduction from MINIMUM DOMINATING SET to MSC with algorithm `msc` to obtain algorithm `mds`

Upper Bound for `mds`

Algorithm `mds` solves MINIMUM DOMINATING SET in time $O(2^{0.305(2n)}) = O(2^{0.610n})$.

Analysis of Running Time (Upper Bound)

Using `msc`

Combine reduction from MINIMUM DOMINATING SET to MSC with algorithm `msc` to obtain algorithm `mds`

Upper Bound for `mds`

Algorithm `mds` solves MINIMUM DOMINATING SET in time $O(2^{0.305(2n)}) = O(2^{0.610n})$.

How to construct lower bound graphs ?

Make sure that reduction rules cannot be applied.

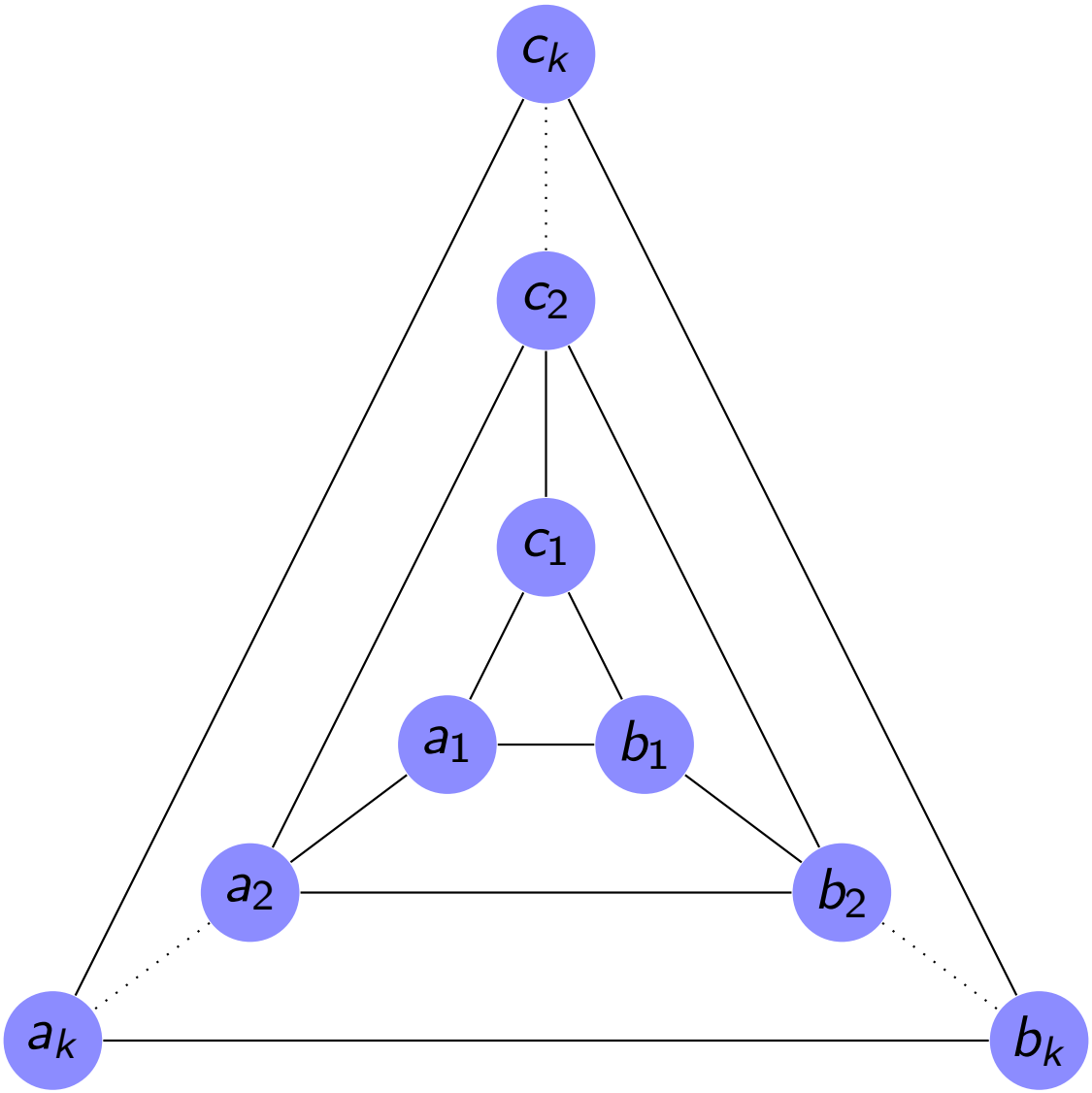
Lower bound graphs should have small maximum degree.

How to construct lower bound graphs ?

Make sure that reduction rules cannot be applied.

Lower bound graphs should have small maximum degree.

Lower Bound Graphs



Important Properties of Lower Bound Graphs

1. G_k has maximum degree 4.
2. Each set $S_v := N[v]$ of $\mathcal{S}$ has size at most 5.
3. There is a (bad) execution of `msc` (resp. `mds`) on G_k choosing the branching vertices such that no reduction rule (not even a connectedness rule) can ever be applied.

How to Choose a Selection Rule ?

- A selection rule specifies how ties will be broken in a (bad) execution of the algorithm.

Objective I

We (i.e. an adversary of the algorithm) choose a selection rule of the algorithm, specifying how to break ties, such as to maximize the running time of the algorithm.

Objective II

The number of leaves in the search tree obtained when `mds` on G_k uses the specified selection rule should be as large as possible.

How to Choose a Selection Rule ?

- A selection rule specifies how ties will be broken in a (bad) execution of the algorithm.

Objective I

We (i.e. an adversary of the algorithm) choose a selection rule of the algorithm, specifying how to break ties, such as to maximize the running time of the algorithm.

Objective II

The number of leaves in the search tree obtained when `mds` on G_k uses the specified selection rule should be as large as possible.

How to Choose a Selection Rule ?

- A selection rule specifies how ties will be broken in a (bad) execution of the algorithm.

Objective I

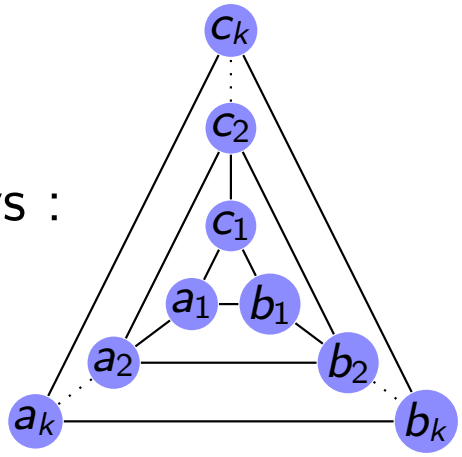
We (i.e. an adversary of the algorithm) choose a selection rule of the algorithm, specifying how to break ties, such as to maximize the running time of the algorithm.

Objective II

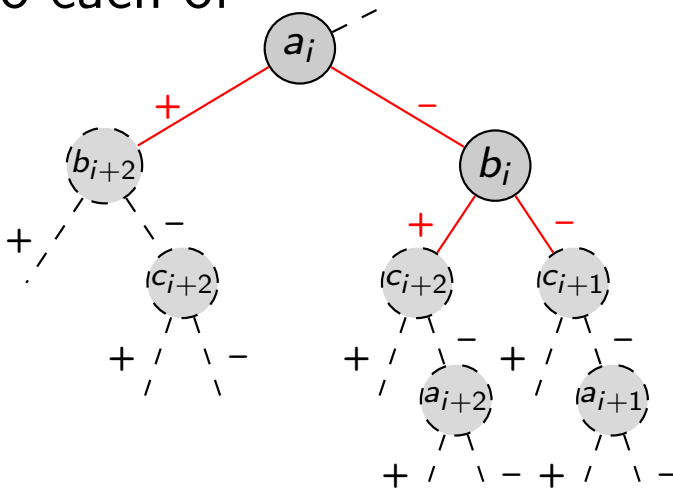
The number of leaves in the search tree obtained when `mds` on G_k uses the specified selection rule should be as large as possible.

Our Selection Rule for mds

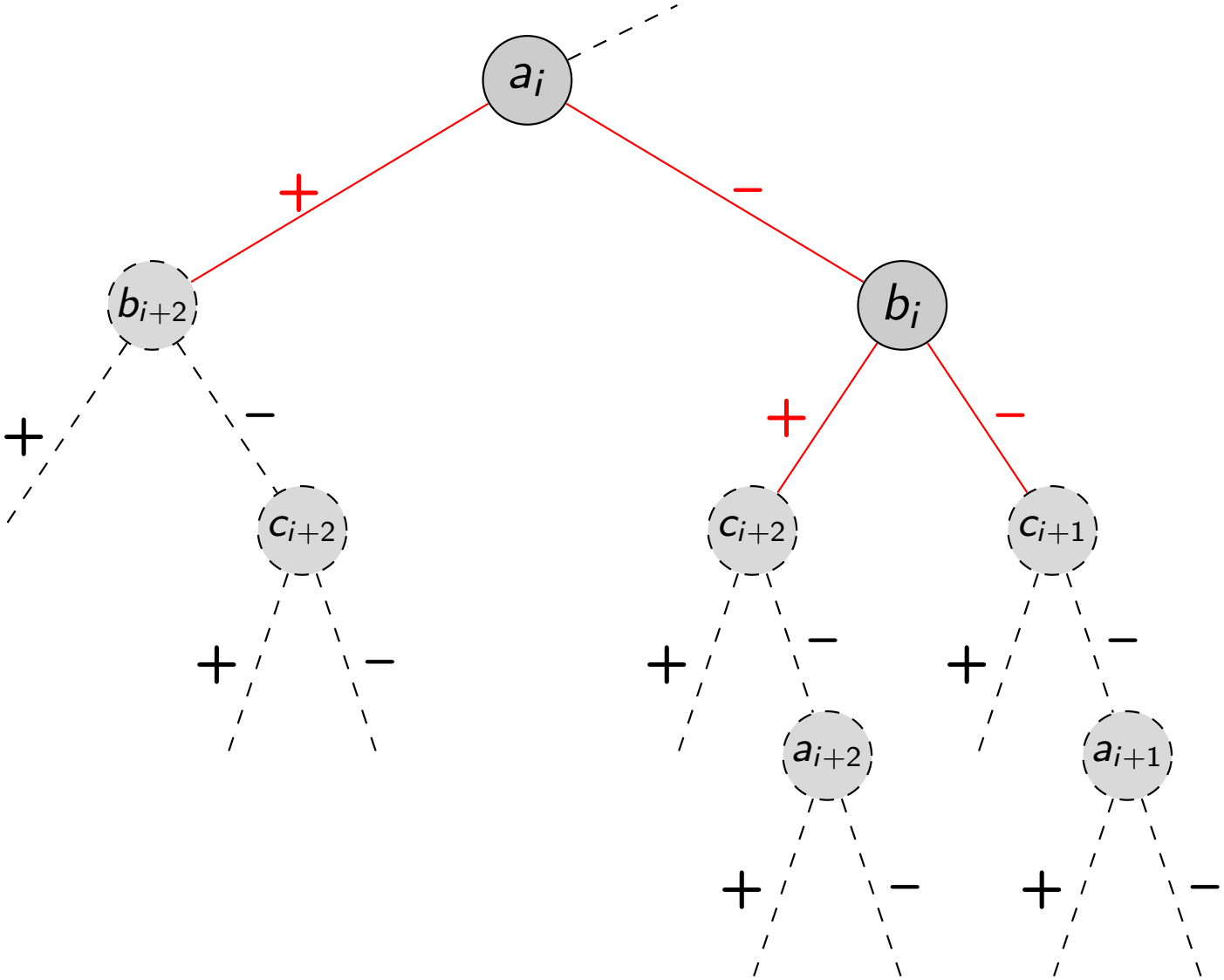
- Round i, $i \in \{2, 3, \ldots, n-1\}$: treat a pair $P = \{x_i, y_i\}$ of vertices belonging to triangle $T_i = \{a_i, b_i, c_i\}$. Initially $P = \{a_2, b_2\}$.
- For each pair $P = \{x_i, y_i\}$ branch in the following 3 ways :
  - 1) select S_{x_i}
  - 2) discard S_{x_i}, and then select S_{y_i}
  - 3) discard S_{x_i}, and then discard S_{y_i}



- The following new pairs of vertices correspond to each of the three branches :
  - 1) $P_1 = \{a_{i+2}, b_{i+2}, c_{i+2}\} \setminus x_{i+2}$
  - 2) $P_2 = \{a_{i+2}, b_{i+2}, c_{i+2}\} \setminus y_{i+2}$
  - 3) $P_3 = \{x_{i+1}, y_{i+1}\}$
- On each pair P_j recursively repeat the process.



A search tree



A Lower Bound

Definition

Let $T(k)$ be the number of leaves in the search tree obtained when algorithm `mds` on input graph G_k uses the specified selection rule.

Of the three branches of T_i two are proceeded on T_{i+2} and one is proceeded on T_{i+1}.

Recurrence

$$T(k) = 2 \cdot T(k-2) + T(k-1)$$

A Lower Bound

Definition

Let $T(k)$ be the number of leaves in the search tree obtained when algorithm `mds` on input graph G_k uses the specified selection rule.

Of the three branches of T_i two are proceeded on T_{i+2} and one is proceeded on T_{i+1}.

Recurrence

$$T(k) = 2 \cdot T(k-2) + T(k-1)$$

A Lower Bound

Lower Bound of `mds`

The running time of algorithm `mds` is $\Omega(2^k) = \Omega(2^{n/3})$.

A Lower Bound

Lower Bound of `mds`

The running time of algorithm `mds` is $\Omega(2^k) = \Omega(2^{n/3})$.

REMINDER : Upper Bound $O(2^{0.610n})$.

Conclusions

Hopefully ...

the insights obtained when studying lower bounds for a particular Branch & Reduce algorithm might be of help for choosing sophisticated measures to improve the upper bound of the (worst case) running time.

Conclusions

Study and Improve Lower Bounds !

Provide lower bounds with exponential-time algorithms !

Conclusions

Study and Improve Lower Bounds !

Provide lower bounds with exponential-time algorithms !

Conclusions

What we really need ?

We do need better methods to analyse the running time of Branch & Reduce algorithms.

Thank you for your attention !

For Further Reading I

Alekhnovich, E.H.M., E.A. Hirsch, and D. Itsykon,
Exponential lower bounds for the running time of DPLL algorithms on satisfiable formulas,
Proc. of ICALP 2004, *LNCS* **3142**, (2004), pp. 84–96.

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.

Gaspers, S., and M. Liedloff,
A branch-and-reduce algorithm for finding a minimum independent dominating set in graphs,
Proceedings of WG 2006), *LNCS*, Springer, Berlin, 2006, to appear.

For Further Reading II

Kratsch, D., and M. Liedloff,
An exact algorithm for the minimum dominating clique problem,
Proceedings of IWPEC 2006, *LNCS*, Springer, Berlin, 2006, to appear.

Pudlak, P., and R. Impaglazzio,
A lower bound for dll algorithms for k-sat, *Proceedings of SODA'2000*, pp. 128–136.

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 III

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