Algorithmic Graph Theory: How hard is your combinatorial optimization problem?

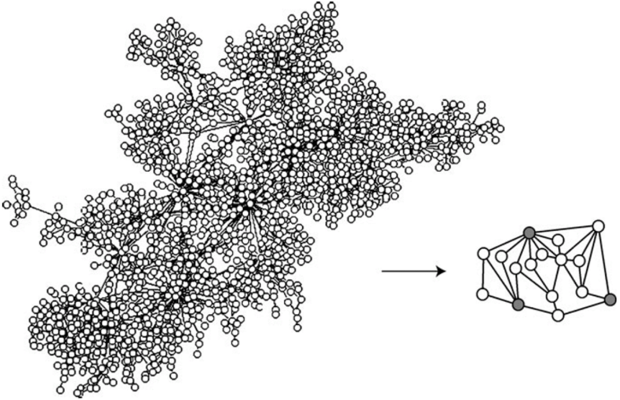
Arie M.C.A. Koster

Lecture 11

Clemson, June 14, 2017

Lehrstuhl II für Mathematik

Yesterday: Dominating Set



Power of kernelization

Algorithm mis1(G).
Input: Graph $G = (V, E)$.
Output: The maximum cardinality of an independent set of G.

if $|V| = 0$ **then**
  return 0
choose a vertex v of minimum degree in G
return $1 + \max\{\texttt{mis1}(G \setminus N[y]) : y \in N[v]\}$

Fig. 1.2 Algorithm mis1 for MAXIMUM INDEPENDENT SET

What is the running time of MIS1 ?

Algorithm mis1(G).
Input: Graph $G = (V, E)$.
Output: The maximum cardinality of an independent set of G.

if $|V| = 0$ **then**
  return 0
choose a vertex v of minimum degree in G
return $1 + \max\{\texttt{mis1}(G \setminus N[y]) : y \in N[v]\}$

Fig. 1.2 Algorithm mis1 for MAXIMUM INDEPENDENT SET

Definition

In the context of exact exponential algorithms, we write

$$f(n) = O^*(g(n))$$

if $f(n) = O(g(n)poly(n))$, where $poly(n)$ is an arbitrary polynomial.

Algorithm `mis1`**(G).**
Input: Graph $G = (V, E)$.
Output: The maximum cardinality of an independent set of G.

if $|V| = 0$ **then**
  return 0
choose a vertex v of minimum degree in G
return $1 + \max\{\mathtt{mis1}(G \setminus N[y]) : y \in N[v]\}$

Fig. 1.2 Algorithm `mis1` for MAXIMUM INDEPENDENT SET

Definition

In the context of exact exponential algorithms, we write

$$f(n) = O^*(g(n))$$

if $f(n) = O(g(n)poly(n))$, where $poly(n)$ is an arbitrary polynomial.

For n large, $(\sqrt{2})^n \cdot poly(n)$ lies between 1.4142135^n and 1.4142136^n.

Algorithm `mis1`**(G).**
Input: Graph $G = (V, E)$.
Output: The maximum cardinality of an independent set of G.

if $|V| = 0$ **then**
  return 0
choose a vertex v of minimum degree in G
return $1 + \max\{\texttt{mis1}(G \setminus N[y]) : y \in N[v]\}$

Fig. 1.2 Algorithm `mis1` for MAXIMUM INDEPENDENT SET

Definition

In the context of exact exponential algorithms, we write

$$f(n) = O^*(g(n))$$

if $f(n) = O(g(n)poly(n))$, where $poly(n)$ is an arbitrary polynomial.

For n large, $(\sqrt{2})^n \cdot poly(n)$ lies between 1.4142135^n and 1.4142136^n.
We write $O(\sqrt{2})^n \cdot poly(n)) = O^*(\sqrt{2}^n) = O(1.4143^n)$.

Algorithm `mis1`**(G).**
Input: Graph $G = (V, E)$.
Output: The maximum cardinality of an independent set of G.

if $|V| = 0$ **then**
  return 0
choose a vertex v of minimum degree in G
return $1 + \max\{\texttt{mis1}(G \setminus N[y]) : y \in N[v]\}$

Fig. 1.2 Algorithm `mis1` for Maximum Independent Set

Definition

In the context of exact exponential algorithms, we write

$$f(n) = O^*(g(n))$$

if $f(n) = O(g(n)poly(n))$, where $poly(n)$ is an arbitrary polynomial.

For n large, $(\sqrt{2})^n \cdot poly(n)$ lies between 1.4142135^n and 1.4142136^n.
We write $O(\sqrt{2})^n \cdot poly(n)) = O^*(\sqrt{2}^n) = O(1.4143^n)$.
Note: Not only running time plays a role, also the memory requirements.

A branching algorithm is a recursive procedure allowing to determine the optimal solution by dividing the problem in smaller subproblems with the benefit

- memory requirements are polynomial (or even linear)

A branching algorithm is a recursive procedure allowing to determine the optimal solution by dividing the problem in smaller subproblems with the benefit

- memory requirements are polynomial (or even linear)
- the running time can be significantly better than the "worst-case" analysis

A branching algorithm is a recursive procedure allowing to determine the optimal solution by dividing the problem in smaller subproblems with the benefit

- memory requirements are polynomial (or even linear)
- the running time can be significantly better than the “worst-case” analysis
- simple improvements can be included, improving the practical performance

A branching algorithm is a recursive procedure allowing to determine the optimal solution by dividing the problem in smaller subproblems with the benefit

- memory requirements are polynomial (or even linear)
- the running time can be significantly better than the "worst-case" analysis
- simple improvements can be included, improving the practical performance

Branching algorithms consist of

- reduction rules for preprocessing instances I or stopping the recursion

A branching algorithm is a recursive procedure allowing to determine the optimal solution by dividing the problem in smaller subproblems with the benefit

- memory requirements are polynomial (or even linear)
- the running time can be significantly better than the "worst-case" analysis
- simple improvements can be included, improving the practical performance

Branching algorithms consist of

- reduction rules for preprocessing instances I or stopping the recursion
- branching rules to divide an instance in two or more smaller instances

Branching Algorithms

A branching algorithm is a recursive procedure allowing to determine the optimal solution by dividing the problem in smaller subproblems with the benefit

- memory requirements are polynomial (or even linear)
- the running time can be significantly better than the “worst-case” analysis
- simple improvements can be included, improving the practical performance

Branching algorithms consist of

- reduction rules for preprocessing instances I or stopping the recursion
- branching rules to divide an instance in two or more smaller instances

Correctness of a branching is often easy to prove; the analysis more difficult.

If the procesisng time of a search node is polynomial, the number of nodes determines the overall running time of the algorithm.

Analysis

If the procesisng time of a search node is polynomial, the number of nodes determines the overall running time of the algorithm.

The number of nodes of the search tree is at most twice its number of leafs.

If the procesisng time of a search node is polynomial, the number of nodes determines the overall running time of the algorithm.

The number of nodes of the search tree is at most twice its number of leafs.

Let $T(n)$ be the maximum number of leafs of any search tree, given an instance of size n.

If the procesisng time of a search node is polynomial, the number of nodes determines the overall running time of the algorithm.

The number of nodes of the search tree is at most twice its number of leafs.

Let $T(n)$ be the maximum number of leafs of any search tree, given an instance of size n.

Branching vector $b = (t_1, t_2, \ldots, t_r)$: instance of size n branches into r instances with sizes at most $n - t_1, n - t_2, \ldots, n - t_r$.

If the procesisng time of a search node is polynomial, the number of nodes determines the overall running time of the algorithm.

The number of nodes of the search tree is at most twice its number of leafs.

Let $T(n)$ be the maximum number of leafs of any search tree, given an instance of size n.

Branching vector $b = (t_1, t_2, \ldots, t_r)$: instance of size n branches into r instances with sizes at most $n - t_1, n - t_2, \ldots, n - t_r$.

Lemma

Given branching vector $b = (t_1, t_2, \ldots, t_r)$, it holds

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

If the procesisng time of a search node is polynomial, the number of nodes determines the overall running time of the algorithm.

The number of nodes of the search tree is at most twice its number of leafs.

Let $T(n)$ be the maximum number of leafs of any search tree, given an instance of size n.

Branching vector $b = (t_1, t_2, \ldots, t_r)$: instance of size n branches into r instances with sizes at most $n - t_1, n - t_2, \ldots, n - t_r$.

Lemma

Given branching vector $b = (t_1, t_2, \ldots, t_r)$, it holds

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

In worst-case, equality holds, and we search c such that $T(n) = c^n$.

Example: Branching vector $b = (1, 1)$:

Branching factor

Example: Branching vector $b = (1, 1)$: $T(n) = 2^n$

Example: Branching vector $b = (1, 1)$: $T(n) = 2^n$

Theorem

Let b be a branching rule with branching vector $(t_1, t_2, \ldots, t_r)$. The running time is $O^*(c)$ where c is the unique positive zero of the equation

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

Branching factor

Example: Branching vector $b = (1, 1)$: $T(n) = 2^n$

Theorem

Let b be a branching rule with branching vector $(t_1, t_2, \ldots, t_r)$. The running time is $O^*(c)$ where c is the unique positive zero of the equation

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

We denote $c =: \tau(t_1, t_2, \ldots, t_r) = \tau(b)$ the branching factor of b.

Example: Branching vector $b = (1,1)$: $T(n) = 2^n$

Theorem

Let b be a branching rule with branching vector $(t_1, t_2, \ldots, t_r)$. The running time is $O^*(c)$ where c is the unique positive zero of the equation

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

We denote $c =: \tau(t_1, t_2, \ldots, t_r) = \tau(b)$ the branching factor of b.

Lemma

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

- $\tau(t_1, t_2, \ldots, t_r) > 1$

Branching factor

Example: Branching vector $b = (1, 1)$: $T(n) = 2^n$

Theorem

Let b be a branching rule with branching vector $(t_1, t_2, \dots, t_r)$. The running time is $O^*(c)$ where c is the unique positive zero of the equation

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

We denote $c =: \tau(t_1, t_2, \dots, t_r) = \tau(b)$ the branching factor of b.

Lemma

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

- $\tau(t_1, t_2, \dots, t_r) > 1$
- $\tau(t_1, \dots, t_r) = \tau(t_{\pi(1)}, \dots, t_{\pi(r)})$ *for every permutation π*

Example: Branching vector $b = (1, 1)$: $T(n) = 2^n$

Theorem

Let b be a branching rule with branching vector $(t_1, t_2, \ldots, t_r)$. The running time is $O^*(c)$ where c is the unique positive zero of the equation

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

We denote $c =: \tau(t_1, t_2, \ldots, t_r) = \tau(b)$ the branching factor of b.

Lemma

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

- $\tau(t_1, t_2, \ldots, t_r) > 1$
- $\tau(t_1, \ldots, t_r) = \tau(t_{\pi(1)}, \ldots, t_{\pi(r)})$ *for every permutation* π
- *If* $t_1 > t_1'$, *then* $\tau(t_1, \ldots, t_r) < \tau(t_1', t_2, \ldots, t_r)$

Lemma

Let i, j, k positive real numbers.

- $\tau(k,k) \leq \tau(i,j)$ *for all* (i,j) *with* $i + j = 2k$

Lemma

Let i, j, k positive real numbers.

- $\tau(k,k) \leq \tau(i,j)$ *for all* (i,j) *with* $i+j=2k$
- $\tau(i,j) > \tau(i+\epsilon, j-\epsilon)$ *for all* $0 < i < j$ *and all* $0 < \epsilon < \frac{j-i}{2}$

Branching factor

Lemma

Let i, j, k positive real numbers.

- $\tau(k,k) \leq \tau(i,j)$ *for all* (i,j) *with* $i + j = 2k$
- $\tau(i,j) > \tau(i+\epsilon, j-\epsilon)$ *for all* $0 < i < j$ *and all* $0 < \epsilon < \frac{j-i}{2}$

$\tau(3,3) < 1.2600$, $\tau(2,4) = \tau(4,2) < 1.2712$, $\tau(5,1) = \tau(1,5) < 1.3248$

Branching factor

Lemma

Let i, j, k positive real numbers.

- $\tau(k,k) \leq \tau(i,j)$ *for all* (i,j) *with* $i+j=2k$
- $\tau(i,j) > \tau(i+\epsilon, j-\epsilon)$ *for all* $0 < i < j$ *and all* $0 < \epsilon < \frac{j-i}{2}$

A first example:

Algorithm mis1(G).
Input: Graph $G = (V,E)$.
Output: The maximum cardinality of an independent set of G.

```
if |V| = 0 then
  return 0
choose a vertex v of minimum degree in G
return 1 + max{mis1(G \ N[y]) : y ∈ N[v]}
```

Fig. 1.2 Algorithm mis1 for MAXIMUM INDEPENDENT SET

Lemma

Let i, j, k positive real numbers.

- $\tau(k,k) \leq \tau(i,j)$ *for all* (i,j) *with* $i + j = 2k$
- $\tau(i,j) > \tau(i+\epsilon, j-\epsilon)$ *for all* $0 < i < j$ *and all* $0 < \epsilon < \frac{j-i}{2}$

A first example:

Algorithm mis1(G).
Input: Graph $G = (V, E)$.
Output: The maximum cardinality of an independent set of G.

```
if |V| = 0 then
  return 0
choose a vertex v of minimum degree in G
return 1 + max{mis1(G \ N[y]) : y ∈ N[v]}
```

Fig. 1.2 Algorithm mis1 for MAXIMUM INDEPENDENT SET

Lemma

If vertices of degree 0 and 1 are preprocessed, then MIS1 *has running time* $O^*(\sqrt[3]{3}) = O(1.4423^n)$.

Multiple Branching Rules

Lemma

If an algorithm uses different branching rules in different situations, the running time is determined by the branching rule with highest branching factor.

Lemma

If an algorithm uses different branching rules in different situations, the running time is determined by the branching rule with highest branching factor.

Lemma

Let (i, j) *be a branching vector and* (k, l) *the branching vector for the subproblem of siz* $n - i$. *Then,* $(i + k, i + l, j)$ *is a branching vector for the combined branching.*

Lemma (Dominance rule)

Let $G = (V, E)$ be a graph and v, w adjacent vertices with $N[v] \subseteq N[w]$. Then, $\alpha(G) = \alpha(G - w)$.

Lemma (Dominance rule)

Let $G = (V, E)$ *be a graph and* v, w *adjacent vertices with* $N[v] \subseteq N[w]$*. Then,* $\alpha(G) = \alpha(G - w)$*.*

Lemma (Simplicial rule)

Let $G = (V, E)$ *be a graph and* v *a simplicial vertex. Then,* $\alpha(G) = 1 + \alpha(G - N[v])$*.*

Lemma (Dominance rule)

Let $G = (V, E)$ *be a graph and* v, w *adjacent vertices with* $N[v] \subseteq N[w]$*. Then,* $\alpha(G) = \alpha(G - w)$*.*

Lemma (Simplicial rule)

Let $G = (V, E)$ *be a graph and* v *a simplicial vertex. Then,* $\alpha(G) = 1 + \alpha(G - N[v])$*.*

Lemma

Let G *be a non-connected graph with* $C \subset V$ *defining a connected component of* G*. Then,* $\alpha(G) = \alpha(G - C) + \alpha(G[C])$*.*

Lemma (Standard Branching)

Let $G = (V, E)$ *and* $v \in V$. *Then,*
$\alpha(G) = \max\{1 + \alpha(G - N[v]), \alpha(G - v)\}$.

Lemma (Standard Branching)

Let $G = (V, E)$ *and* $v \in V$. *Then,*
$\alpha(G) = \max\{1 + \alpha(G - N[v]), \alpha(G - v)\}$.

The standard branching vector is $(deg(v) + 1, 1)$.

More Branching Rules

Lemma (Standard Branching)

Let $G = (V, E)$ and $v \in V$. Then,
$\alpha(G) = \max\{1 + \alpha(G - N[v]), \alpha(G - v)\}$.

The standard branching vector is $(deg(v) + 1, 1)$.

Lemma

Let $G = (V, E)$ and $v \in V$. If v is not contained in any maximum independent set, then every maximum independent set contains at least two vertices from $N(v)$.

More Branching Rules

Lemma (Standard Branching)

Let $G = (V, E)$ and $v \in V$. Then,
$\alpha(G) = \max\{1 + \alpha(G - N[v]), \alpha(G - v)\}$.

The standard branching vector is $(deg(v) + 1, 1)$.

Lemma

Let $G = (V, E)$ and $v \in V$. If v is not contained in any maximum independent set, then every maximum independent set contains at least two vertices from $N(v)$.

More Branching Rules

Let $N^2(v) := \{w \in V \setminus N[v] : \exists x \in N(v) \text{ with } w \in N(x)\}$.

Let $N^2(v) := \{w \in V \setminus N[v] : \exists x \in N(v) \text{ with } w \in N(x)\}$.

Definition

A vertex $w \in N^2(v)$ is called mirror of v, if $N(v) \setminus N(w)$ defines a clique. Let $M(v)$ be the set of mirrors of v in G.

More Branching Rules

Let $N^2(v) := \{w \in V \setminus N[v] : \exists x \in N(v) \text{ with } w \in N(x)\}$.

Definition

A vertex $w \in N^2(v)$ is called mirror of v, if $N(v) \setminus N(w)$ defines a clique. Let $M(v)$ be the set of mirrors of v in G.

Lemma (Mirror Branching)

Let $G = (V, E)$ and $v \in V$. Then

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

Let $N^2(v) := \{w \in V \setminus N[v] : \exists x \in N(v) \text{ with } w \in N(x)\}$.

Definition

A vertex $w \in N^2(v)$ is called mirror of v, if $N(v) \setminus N(w)$ defines a clique. Let $M(v)$ be the set of mirrors of v in G.

Lemma (Mirror Branching)

Let $G = (V, E)$ and $v \in V$. Then

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

Lemma (Separator Branching)

Let $G = (V, E)$ and S separator of G. Let $\mathcal{I}(S)$ be the collection of all independent sets of S. Then,

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

Lemma (Separator Branching)

Let $G = (V, E)$ and S separator of G. Let $\mathcal{I}(S)$ be the collection of all independent sets of S. Then,

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

We apply separator branching only in two cases: $S = N^2(v)$ and $|S| \leq 2$.

Lemma (Separator Branching)

Let $G = (V, E)$ and S separator of G. Let $\mathcal{I}(S)$ be the collection of all independent sets of S. Then,

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

We apply separator branching only in two cases: $S = N^2(v)$ and $|S| \leq 2$.
See page 27, Fomin & Kratsch, 2010.

Theorem

MIS2 *solves the max. independent set problem in* $O(1.2786^n)$.

Refining the Analysis

Idea: use a different measure to bound the running time.

Refining the Analysis

Idea: use a different measure to bound the running time.

Here, the measure has to be satisfy following conditions:

1. the measure has to be smaller for smaller instances
2. the measure should be nonnegative
3. the measure should be bounded from above by a function of the "natural" parameter of the input

Refining the Analysis

Idea: use a different measure to bound the running time.

Here, the measure has to be satisfy following conditions:

1. the measure has to be smaller for smaller instances
2. the measure should be nonnegative
3. the measure should be bounded from above by a function of the "natural" parameter of the input

Algorithm mis3**(G).**
Input: A graph $G = (V, E)$.
Output: The maximum cardinality of an independent set of G.

1 **if** $\exists v \in V$ with $d(v) = 0$ **then**
  return $1 + \texttt{mis3}(G \setminus v)$

2 **if** $\exists v \in V$ with $d(v) = 1$ **then**
  return $1 + \texttt{mis3}(G \setminus N[v])$

3 **if** $\Delta(G) \geq 3$ **then**
  choose a vertex v of maximum degree in G
  return $\max(1 + \texttt{mis3}(G \setminus N[v]), \texttt{mis3}(G \setminus v))$

4 **if** $\Delta(G) \leq 2$ **then**
  compute $\alpha(G)$ using a polynomial time algorithm
  return $\alpha(G)$

Fig. 6.1 Algorithm mis3 for MIS

First Analysis: $O(1.3803^n)$

First Analysis: $O(1.3803^n)$

New measure: $K_1(G') := |\{v \in V(G') : deg_{G'}(v) \geq 3\}| =: n_{\geq 3}$

First Analysis: $O(1.3803^n)$
New measure: $K_1(G') := |\{v \in V(G') : deg_{G'}(v) \geq 3\}| =: n_{\geq 3}$
Branching factor: $\tau(1,1) = 2$.

First Analysis: $O(1.3803^n)$
New measure: $K_1(G') := |\{v \in V(G') : deg_{G'}(v) \geq 3\}| =: n_{\geq 3}$
Branching factor: $\tau(1,1) = 2$.

- Vertices of degree ≤ 1 have weight 0
- Vertices of degree 2 have weight $w_2 \in [0,1]$
- Vertices of degree ≥ 3 have weight 3

Theorem

With measure $K_2(G') := w_2 n_2 + n_{\geq 3}$, MIS3 has running time $O(1.3248^n)$.

First Analysis: $O(1.3803^n)$
New measure: $K_1(G') := |\{v \in V(G') : deg_{G'}(v) \geq 3\}| =: n_{\geq 3}$
Branching factor: $\tau(1,1) = 2$.

- Vertices of degree ≤ 1 have weight 0
- Vertices of degree 2 have weight $w_2 \in [0,1]$
- Vertices of degree ≥ 3 have weight 3

Theorem

With measure $K_2(G') := w_2 n_2 + n_{\geq 3}$, MIS3 has running time $O(1.3248^n)$.

Theorem

Let $k(G') = \sum_{i=0}^{n} w_i n_i$ with $w_i \in [0,1]$ and $n_i = |\{v \in V : deg(v) = i\}|$. If $w_0 = w_1 = 0$, $w_2 = 0.596601$, $w_3 = 0.928643$ and $w_t = 1$ for $t \geq 4$, then MIS3 has running time $O(1.2905^n)$.

First Analysis: $O(1.3803^n)$
New measure: $K_1(G') := |\{v \in V(G') : deg_{G'}(v) \geq 3\}| =: n_{\geq 3}$
Branching factor: $\tau(1,1) = 2$.

- Vertices of degree ≤ 1 have weight 0
- Vertices of degree 2 have weight $w_2 \in [0,1]$
- Vertices of degree ≥ 3 have weight 3

Theorem

With measure $K_2(G') := w_2 n_2 + n_{\geq 3}$, MIS3 has running time $O(1.3248^n)$.

Theorem

Let $k(G') = \sum_{i=0}^{n} w_i n_i$ with $w_i \in [0,1]$ and $n_i = |\{v \in V : deg(v) = i\}|$. If $w_0 = w_1 = 0$, $w_2 = 0.596601$, $w_3 = 0.928643$ and $w_t = 1$ for $t \geq 4$, then MIS3 has running time $O(1.2905^n)$.

Note: There exists an algorithm with running time $O(1.2209^n)$.

Example (MIN DOMINATING SET - MDS)

Given: Graph $G = (V, E)$

Find: Dominating Set $D \subseteq V$ of minimum cardinality, i.e., $\forall v \in V$: $N[v] \cap D \neq \emptyset$.

Example (MIN DOMINATING SET - MDS)

Given: Graph $G = (V, E)$

Find: Dominating Set $D \subseteq V$ of minimum cardinality, i.e., $\forall v \in V$: $N[v] \cap D \neq \emptyset$.

Example (MIN SET COVER - MSC)

Given: Ground set U, collection $\mathcal{S}$ of nonempty subsets of U.

Find: Set Cover of $(U, \mathcal{S})$, i.e., a subset $\mathcal{S}' \subseteq \mathcal{S}$ such that $\cup_{s \in \mathcal{S}'} s = U$. Minimize $|\mathcal{S}'|$.

Example (MIN DOMINATING SET - MDS)

Given: Graph $G = (V, E)$
Find: Dominating Set $D \subseteq V$ of minimum cardinality, i.e., $\forall v \in V$: $N[v] \cap D \neq \emptyset$.

Example (MIN SET COVER - MSC)

Given: Ground set U, collection $\mathcal{S}$ of nonempty subsets of U.
Find: Set Cover of $(U, \mathcal{S})$, i.e., a subset $\mathcal{S}' \subseteq \mathcal{S}$ such that $\cup_{s \in \mathcal{S}'} s = U$.
Minimize $|\mathcal{S}'|$.

MDS as MSC

Define $U = V$, $\mathcal{S} = \{N[v] = S_v : v \in V\}$.
MDS can be solved as MSC.

Example (MIN DOMINATING SET - MDS)

Given: Graph $G = (V, E)$
Find: Dominating Set $D \subseteq V$ of minimum cardinality, i.e., $\forall v \in V$: $N[v] \cap D \neq \emptyset$.

Example (MIN SET COVER - MSC)

Given: Ground set U, collection $\mathcal{S}$ of nonempty subsets of U.
Find: Set Cover of $(U, \mathcal{S})$, i.e., a subset $\mathcal{S}' \subseteq \mathcal{S}$ such that $\cup_{s \in \mathcal{S}'} s = U$.
Minimize $|\mathcal{S}'|$.

MDS as MSC

Define $U = V$, $\mathcal{S} = \{N[v] = S_v : v \in V\}$.
MDS can be solved as MSC.

Note: $|U| = |\mathcal{S}| = |V| = n$

For $u \in U$, let the frequency $f_u := |\{S \in \mathcal{S} : u \in S\}|$.

For $u \in U$, let the frequency $f_u := |\{S \in \mathcal{S} : u \in S\}|$.
W.l.o.g. $f_u \geq 1$, otherwise MSC is infeasible (i.e., $U \neq \cup_{S \in \mathcal{S}} S$).

For $u \in U$, let the frequency $f_u := |\{S \in \mathcal{S} : u \in S\}|$.
W.l.o.g. $f_u \geq 1$, otherwise MSC is infeasible (i.e., $U \neq \cup_{S \in \mathcal{S}} S$).

Lemma

For a MSC instance it holds:

- *If for two distinct sets S and R, $S \subseteq R$, then there exists a MSC not containing S.*

For $u \in U$, let the frequency $f_u := |\{S \in \mathcal{S} : u \in S\}|$.
W.l.o.g. $f_u \geq 1$, otherwise MSC is infeasible (i.e., $U \neq \cup_{S \in \mathcal{S}} S$).

Lemma

For a MSC instance it holds:

- *If for two distinct sets S and R, $S \subseteq R$, then there exists a MSC not containing S.*
- *If $f_u = 1$ for some $u \in U$, the set $S \ni u$ is contained in every set cover.*

For $u \in U$, let the frequency $f_u := |\{S \in \mathcal{S} : u \in S\}|$.
W.l.o.g. $f_u \geq 1$, otherwise MSC is infeasible (i.e., $U \neq \cup_{S \in \mathcal{S}} S$).

Lemma

For a MSC instance it holds:

- *If for two distinct sets S and R, $S \subseteq R$, then there exists a MSC not containing S.*
- *If $f_u = 1$ for some $u \in U$, the set $S \ni u$ is contained in every set cover.*

Note: If $|S| = 1$ for some $S \in \mathcal{S}$, then either there is a R with $S \subseteq R$ or $u \in S$ is covered solely by S. Hence, each one-element S can be preprocessed.

For $u \in U$, let the frequency $f_u := |\{S \in \mathcal{S} : u \in S\}|$.
W.l.o.g. $f_u \geq 1$, otherwise MSC is infeasible (i.e., $U \neq \cup_{S \in \mathcal{S}} S$).

Lemma

For a MSC instance it holds:

- *If for two distinct sets S and R, $S \subseteq R$, then there exists a MSC not containing S.*
- *If $f_u = 1$ for some $u \in U$, the set $S \ni u$ is contained in every set cover.*

Note: If $|S| = 1$ for some $S \in \mathcal{S}$, then either there is a R with $S \subseteq R$ or $u \in S$ is covered solely by S. Hence, each one-element S can be preprocessed.

Lemma

A MSC instance with $|S| = 2$ for all $S \in \mathcal{S}$ can be solved in polynomial time.

Solving MSC

Define $del(S, \mathcal{S}) = \{T : T = R \setminus S \neq \emptyset, R \in \mathcal{S}\}$

Algorithm msc($\mathcal{S}$).
Input: A collection $\mathcal{S}$ of subsets of a universe $\mathcal{U}$.
Output: The minimum cardinality of a set cover of $\mathcal{S}$.

```
1 if |𝒮| = 0 then
    return 0
2 if ∃S,R ∈ 𝒮 with S ⊆ R then
    return msc(𝒮\{S})
3 if ∃u ∈ 𝒰(𝒮) such that there is a unique S ∈ 𝒮 with u ∈ S then
    return 1 + msc(del(S,𝒮))
4 choose a set S ∈ 𝒮 of maximum cardinality
5 if |S| = 2 then
    return poly-msc(𝒮)
6 if |S| ≥ 3 then
    return min(msc(𝒮\{S}), 1 + msc(del(S,𝒮)))
```

Fig. 6.4 Algorithm msc for MSC

Solving MSC

Define $del(S, \mathcal{S}) = \{T : T = R \setminus S \neq \emptyset, R \in \mathcal{S}\}$

Algorithm msc($\mathcal{S}$).
Input: A collection $\mathcal{S}$ of subsets of a universe $\mathcal{U}$.
Output: The minimum cardinality of a set cover of $\mathcal{S}$.

1 **if** $|\mathcal{S}| = 0$ **then**
  return 0
2 **if** $\exists S, R \in \mathcal{S}$ with $S \subseteq R$ **then**
  return msc($\mathcal{S} \setminus \{S\}$)
3 **if** $\exists u \in \mathcal{U}(\mathcal{S})$ such that there is a unique $S \in \mathcal{S}$ with $u \in S$ **then**
  return $1 +$ `msc`(del($S, \mathcal{S}$))
4 choose a set $S \in \mathcal{S}$ of maximum cardinality
5 **if** $|S| = 2$ **then**
  return `poly-msc`($\mathcal{S}$)
6 **if** $|S| \geq 3$ **then**
  return min(`msc`($\mathcal{S} \setminus \{S\}$), $1 +$ `msc`(del($S, \mathcal{S}$)))

Fig. 6.4 Algorithm `msc` for `MSC`

Theorem

Algorithm MSC *solves the Minimum Set Cover problem in* $O(1.2353^{|\mathcal{S}|+|U|})$.

Back to MDS

Corollary

Minimum Dominating Set can be solved in $O(1.2353^{2n}) = O(1.5259^n)$.

Theorem

The worst-case running time of MSC for Min. Dominating Set is $\Omega(2^{\frac{n}{3}}) = \Omega(1.2599^n)$.

Dynamic Programming for TSP

Directed Feedback Arc Set

Example

Directed Feedback Arc Set Let $G = (V, A)$ be a directed graph. A feedback arc set is a subset of the arcs $F \subseteq A$ such that $(V, A \setminus F)$ is acyclic, i.e., every directed cycle in G contains at least one arc from F.

Directed Feedback Arc Set

Example

Directed Feedback Arc Set Let $G = (V, A)$ be a directed graph. A feedback arc set is a subset of the arcs $F \subseteq A$ such that $(V, A \setminus F)$ is acyclic, i.e., every directed cycle in G contains at least one arc from F.

Definition

A topological ordering of a directed graph $G = (V, A)$ is an ordering $\pi : V \to \{1, \ldots, n\}$ (with $n = |V|$) such that $\pi(u) < \pi(v)$, $\forall (u, v) \in A$.

All arcs are directed from left to right.

Directed Feedback Arc Set

Example

Directed Feedback Arc Set Let $G = (V, A)$ be a directed graph. A feedback arc set is a subset of the arcs $F \subseteq A$ such that $(V, A \setminus F)$ is acyclic, i.e., every directed cycle in G contains at least one arc from F.

Definition

A topological ordering of a directed graph $G = (V, A)$ is an ordering $\pi : V \to \{1, \ldots, n\}$ (with $n = |V|$) such that $\pi(u) < \pi(v)$, $\forall (u, v) \in A$.

All arcs are directed from left to right.

Lemma

Let $G = (V, A)$ be a directed graph and $w : A \to \mathbb{Z}_+$. Let $k \geq 0$, integer. There exists a feedback arc set F with weight $\sum_{a \in F} w_a \leq k$ if and only if there exists a linear ordering π of V s.t. $\sum_{(x,y) \in A: \pi(x) > \pi(y)} w(x, y) \leq k$.

π is a topological ordering of $(V, A \setminus F)$.

Theorem

The DIRECTED FEEDBACK ARC SET problem can be solved in $O(nm2^n) = O^*(2^n)$, *where* $n = |V|$ *and* $m = |A|$.

DFAS & Treewidth

Theorem

The DIRECTED FEEDBACK ARC SET problem can be solved in $O(nm2^n) = O^*(2^n)$*, where* $n = |V|$ *and* $m = |A|$.

Theorem

The treewidth of a graph with n *vertices can be determined in* $O^*(2^n)$ *time and* $O^*(2^n)$ *memory.*

Theorem

The treewidth of a graph with n *vertices can be determined in* $O^*(2.9512^n)$ *time and polynomial memory.*

Example (k-COLORABILITY)

Let $G = (V, E)$ be a graph and k integer. A k-coloring of G is an assignment $c : V \rightarrow \{1, \ldots, k\}$ such that $c(v) \neq c(w)$ for all $vw \in E$. The chromatic number $\chi(G)$ is the minimum k for which a k-coloring exists.

COLORABILITY

Example (k-COLORABILITY)

Let $G = (V, E)$ be a graph and k integer. A k-coloring of G is an assignment $c : V \to \{1, \ldots, k\}$ such that $c(v) \neq c(w)$ for all $vw \in E$. The chromatic number $\chi(G)$ is the minimum k for which a k-coloring exists.

k-COLORABILITY can be solved in $O^*(n^n) = O^*(2^{n \log n})$.

Example (k-COLORABILITY)

Let $G = (V, E)$ be a graph and k integer. A k-coloring of G is an assignment $c : V \to \{1, \ldots, k\}$ such that $c(v) \neq c(w)$ for all $vw \in E$. The chromatic number $\chi(G)$ is the minimum k for which a k-coloring exists.

k-COLORABILITY can be solved in $O^*(n^n) = O^*(2^{n \log n})$.

Theorem

$\chi(G)$ can be computed in $O^*((1 + \sqrt[3]{3})^n) = O(2.4423^n)$ with dynamic programming.

Algorithmic Graph Theory: How hard is your combinatorial optimization problem?

Arie M.C.A. Koster

Lecture 11

Clemson, June 14, 2017

CLEMSON UNIVERSITY
Lehrstuhl II für Mathematik
RWTH AACHEN UNIVERSITY