A Measure & Conquer Approach for the Analysis of Exact Algorithms

Fabrizio Grandoni

Tor Vergata Rome

grandoni@disp.uniroma2.it

The Importance of Being Tight

- (Accurately) measuring the size of relevant quantities is a crucial step in science and engineering

- Computer science, and in particular algorithm design, is not an exception

- Tight measures of (worst-case) time/space complexities, approximation ratios etc. are crucial to understand how good an algorithm is, and whether there is room for improvement

The Importance of Being Tight

- Tight bounds sometimes are shown years after the design of an algorithm

- Still, for several poly-time algorithms we are able to provide tight running time bounds

EG: The worst-case running time of `MergeSort` is $\Theta(n \log n)$

- Similarly, we have tight approximation bounds for many approximation algorithms

EG: The approximation ratio of the classical primal-dual algorithm for Steiner forest is exactly 2

The Importance of Being Tight

- The overall situation for exact (exp-time) algorithms for NP-hard problems is much worse

- Typically, tight time bounds are known only for trivial or almost trivial (enumerative) algorithms

- Nonetheless, most of the research in this field was devoted to the design of better algorithms, not of better analytical tools

$\Rightarrow$ The aim of this talk is introducing a non-standard analytical tool, sometimes named *Measure & Conquer*, which leads to much tighter (though possibly non-tight) running time bounds for branch & reduce exact algorithms

Exact Algorithms

Exact Algorithms

- The aim of exact algorithms is solving NP-hard problems exactly with the smallest possible (exponential) running time
- Exact algorithms are interesting for several reasons
  - ◇ Need for exact solutions (e.g. decision problems)
  - ◇ Reducing the running time from, say, $O(2^n)$ to $O(1.41^n)$ roughly doubles the size of the instances solvable within a given (large) time bound. This can't be achieved using faster computers!!
  - ◇ Classical approaches (heuristics, approximation algorithms, parameterized algorithms...) have limits and drawbacks (no guaranty, hardness of approximation, $W[1]$-completeness...)
  - ◇ New combinatorial and algorithmic challenges

Branch & Reduce Algorithms

- The most common exact algorithms are based on the *branch & reduce* paradigm

- The idea is to apply some *reduction rules* to reduce the size of the problem, and then branch on two or more subproblems which are solved recursively according to some *branching rules*

- The analysis of such recursive algorithms is typically based on the *bounded search tree* technique: a *measure* of the size of the subproblems is defined. This measure is used to lower bound the *progress* made by the algorithm at each branching step.

- Though these algorithms are often very complicated, measures used in their analysis are usually trivial (e.g., number of nodes or edges in the graph).

Bounded Search Trees

- Let $P(n)$ be the number of base instances generated to solve a problem of size $n \geq 0$

- Suppose, as it is usual the case, that the application of reduction and branching rules takes polynomial time (in n). Assume also that the branching depth is bounded by a polynomial

- Then the running time of the algorithm is $O(P(n)n^{O(1)}) = O^*(P(n))$
  - $O^*()$ suppresses polynomial factors

- It is possible to show by induction that $P(n) \leq \lambda^n$ for a proper constant $\lambda > 1$

Bounded Search Trees

- Consider a branching/reduction rule b which generates $h(b) \geq 1$ subproblems. Let $n - \delta_j^b$ be the size of the j-th subproblem
  - ◇ It must be $\delta_j^b \geq 0$ (indeed $\delta_j^b > 0$ for $h(b) > 1$)
  - ◇ $(\delta_1^b, \ldots, \delta_{h(b)}^b)$ is the *branching vector*
- We obtain the following inequalities

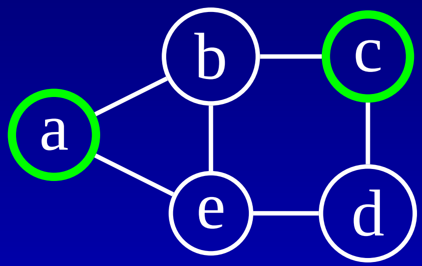
$$P(n) \leq \sum_{j=1}^{h(b)} P(n - \delta_j^b) \leq \sum_{j=1}^{h(b)} \lambda^{n-\delta_j^b} \leq \lambda^n \;\Rightarrow\; f^b(\lambda) := 1 - \sum_{j=1}^{h(b)} \lambda^{-\delta_j^b} \leq 0$$

- This gives a lower bound $\lambda \geq \lambda^b$, where λ^b is the unique positive root of $f^b(\cdot)$ (*branching factor*).
- We can conclude that $\lambda = \max_b\{\lambda^b\}$

The Independent Set Problem

Independent Set

Def: Given $G = (V, E)$, the *maximum independent set* problem (MIS) is to determine the maximum cardinality $\alpha(G)$ of a subset of pairwise non-adjacent nodes (*independent set*)



$$\alpha(G) = 2$$

Known Results

- NP-hard [Karp'72]
- Not approximable within $O(n^{1-\epsilon})$ unless $P = NP$ [Zucherman'06]
- $W[1]$-complete [Downey&Fellows'95].
- No exact $O(\lambda^{o(n)})$ algorithm unless SNP⊆SUBEXP [Impagliazzo,Paturi,Zane'01]

$\Rightarrow$ The best we can hope for is a $O(\lambda^n)$ exact algorithm for some small constant $\lambda \in (1, 2]$.

Known Results

- $O(1.261^n)$ poly-space [Tarjan&Trojanowski'77]
- $O(1.235^n)$ poly-space [Jian'86]
- $O(1.228^n)$ poly-space, $O(1.211^n)$ exp-space [Robson'86]
- better results for sparse graphs [Beigel'99, Chen,Kanj&Xia'03]

- Thanks to Measure & Conquer, a *much simpler* poly-space algorithm ($\simeq 10$ lines of pseudo-code against $\simeq 100$ lines in [Robson'86]) is shown to have time complexity $O(1.221^n)$ [Fomin, Grandoni, Kratsch'06]

$\Rightarrow$ We will consider a similar algorithm, and analyze it in a similar (but simplified) way

Reduction Rules

- Let us introduce a few standard reduction rules for MIS
  - connected components
  - domination
  - folding
  - mirroring
  - ...
- We will use only folding, but in the exercises the other rules might turn to be useful

Connected components

Lem 1: Given a graph G with connected components $G_1, \ldots, G_h$,

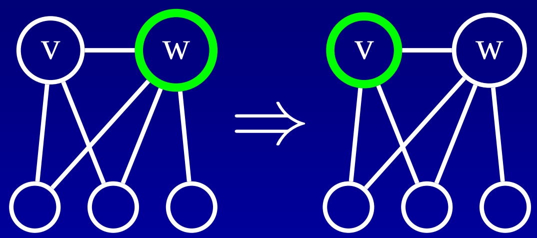
$$\alpha(G) = \sum_i \alpha(G_i)$$

Rem: One can solve the problems induced by the G_i's independently

Domination

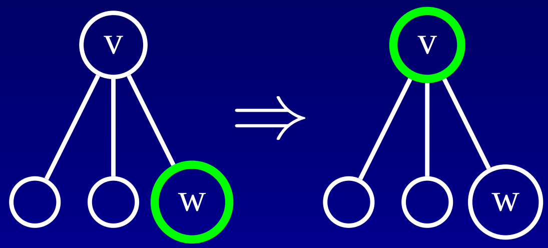
Lem 2: If there are two nodes v and w such that $N[v] \subseteq N[w]$, there is a maximum independent set which does not contain w

- $N[x] = N(x) \cup \{x\}$



Domination

Lem 3: For every node v, there is a maximum independent set which either contains v or at least two nodes in $N(v)$.

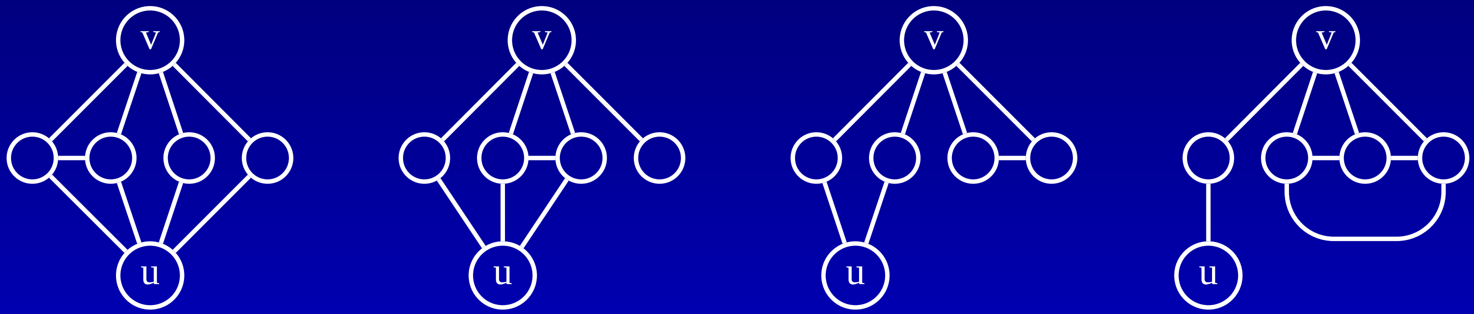


Exr 1: Prove Lemmas 1, 2, and 3

Mirroring

Def: A *mirror* of a node v is a node $u \in N^2(v)$ such that $N(v) - N(u)$ is a (possibly empty) clique

- $N^2(v)$ are the nodes at distance 2 from v
- mirrors of v are denoted by $M(v)$



Mirroring

Lem 4: For any node v,
$\alpha(G) = \max\{\alpha(G - v - M(v)), \alpha(G - N[v])\}$

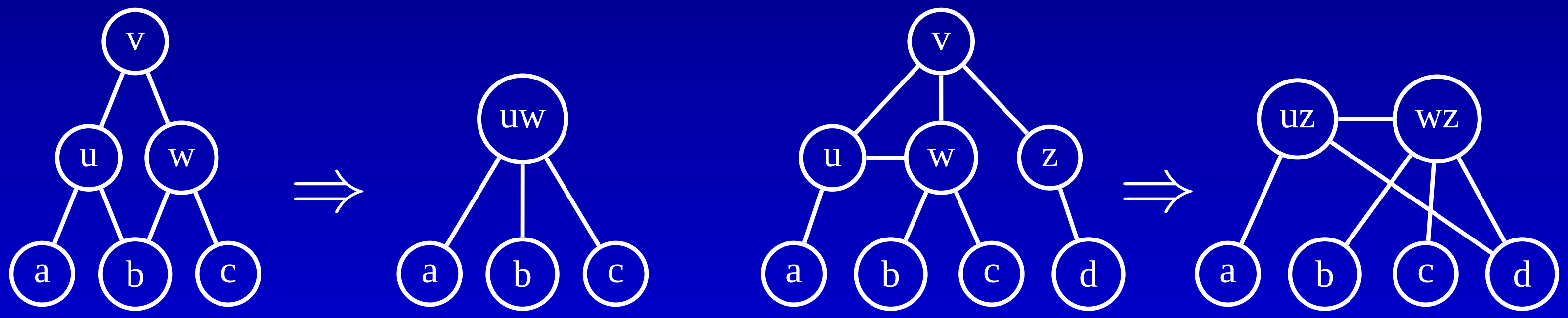
Exr: Prove Lem 4 (Hint: use Lem 3)

Folding

Def: Given a node v with no anti-triangle in $N(v)$, *folding* v means

- replacing $N[v]$ with a clique containing one node uw for each anti-edge uw of $N(v)$;
- adding edges between each uw and N(u)∪ N(w)-N[v].

◇ we use G_v to denote the graph after folding

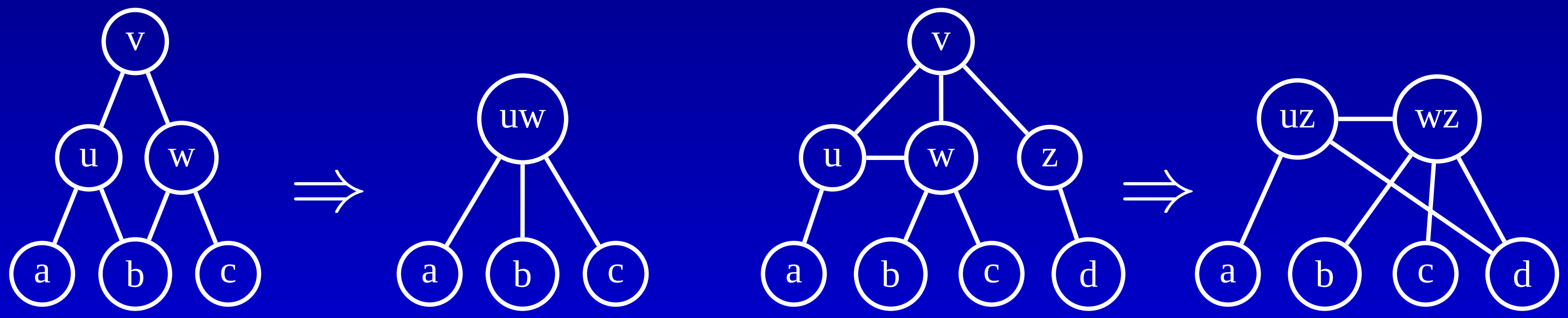


Folding

Def: Given a node v with no anti-triangle in $N(v)$, *folding* v means

- replacing $N[v]$ with a clique containing one node uw for each anti-edge uw of $N(v)$;
- adding edges between each uw and N(u)∪ N(w)-N[v].

◇ we use G_v to denote the graph after folding



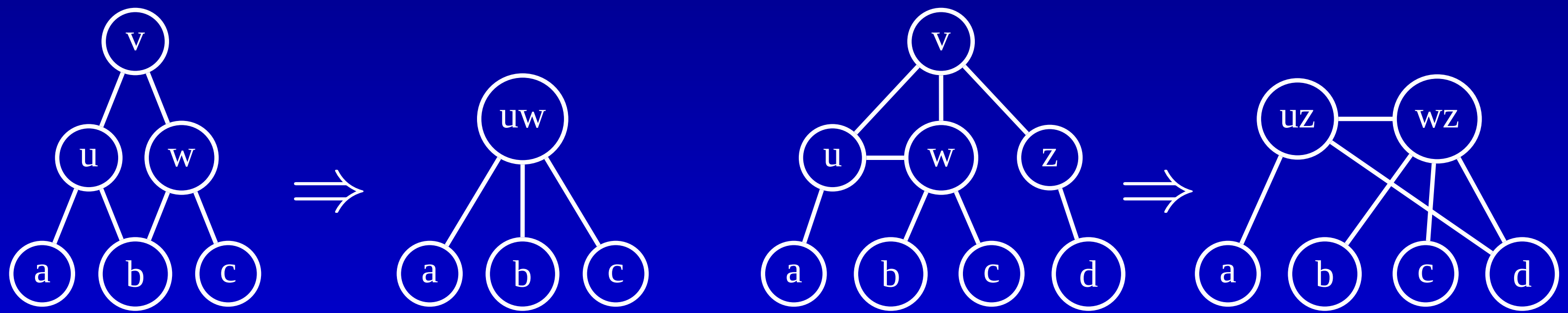
Rem 1: Folding can increase the number of nodes!

Folding

Def: Given a node v with no anti-triangle in $N(v)$, *folding* v means

- replacing $N[v]$ with a clique containing one node uw for each anti-edge uw of $N(v)$;
- adding edges between each uw and N(u)∪ N(w)-N[v].

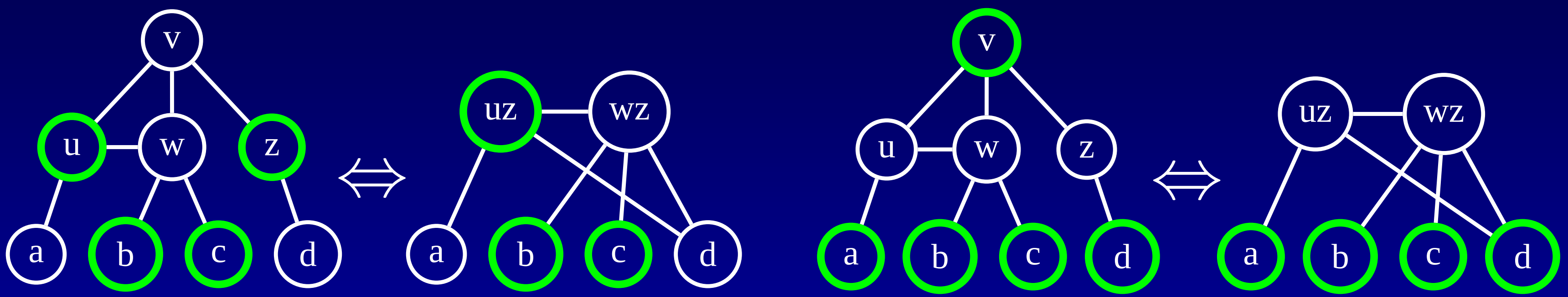
◇ we use G_v to denote the graph after folding



Rem 2: Nodes of degree ≤ 2 are always *foldable*

Folding

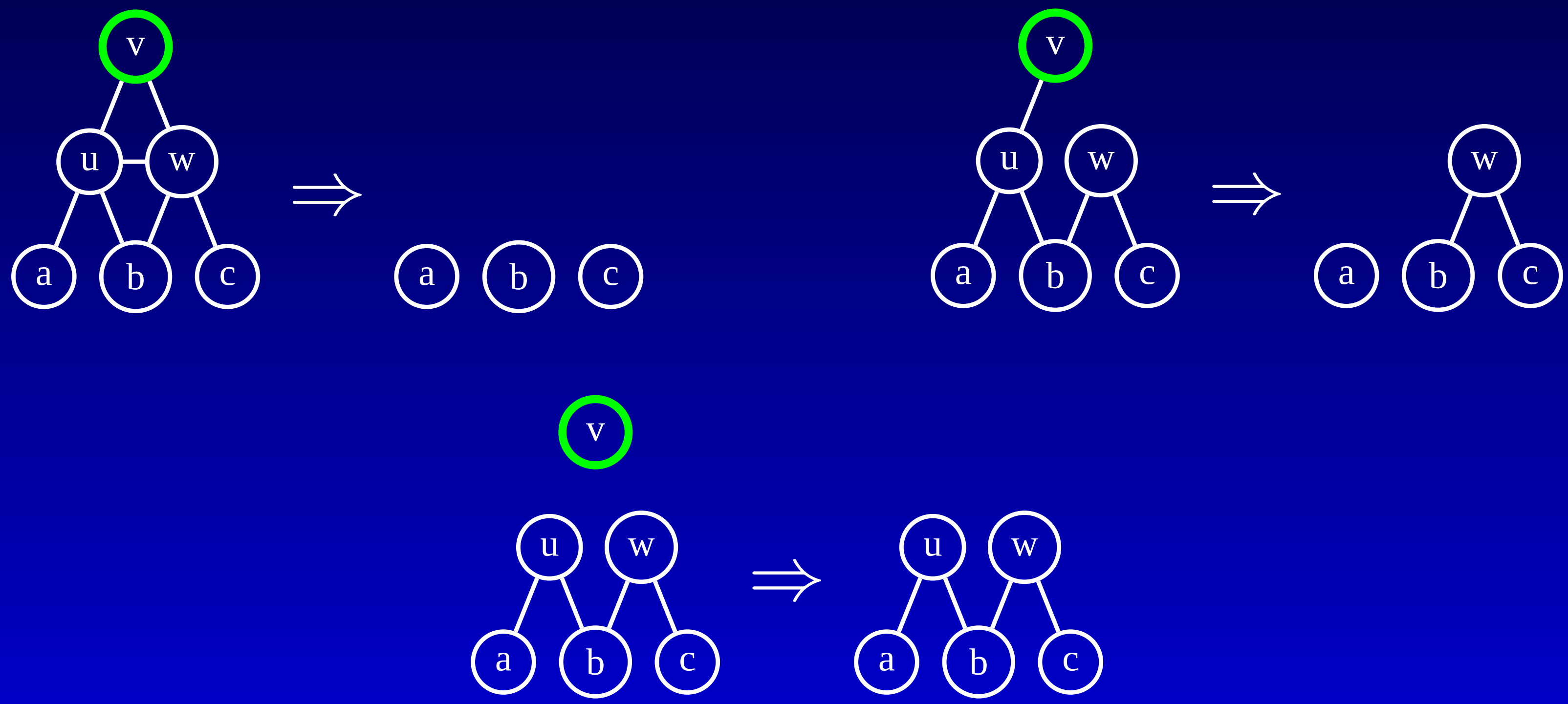
Lem 5: For a foldable node v, $\alpha(G) = 1 + \alpha(G_v)$



Exr 3: Prove Lem 5 (Hint: use Lem 3)

Folding

Rem: Lem 5 includes a few standard reductions as special cases



A Simple MIS Algorithm

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

  if $(G = \emptyset)$ **return** 0; *//Base case*

  //Folding

  Take v of minimum degree;

  if $(d(v) \leq 2)$ **return** $1 + \texttt{mis}(G_v)$;

  //"Greedy" branching

  Take v of maximum degree;

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

}

Standard Analysis of `mis`

Thr: Algorithm `mis` solves MIS in $O^*(1.33^n)$ time

Standard Analysis of `mis`

Thr: Algorithm `mis` solves MIS in $O^*(1.33^n)$ time

Prf:

- Let $P(n)$ be the number of base instances generated by the algorithm. We will show by induction that $P(n) \leq \lambda^n$ for $\lambda < 1.33$
- In the base case $P(0) = 1 \leq \lambda^0$
- When the algorithm folds a node, the number of nodes decreases by at least one

$$P(n) \leq P(n-1) \leq \lambda^{n-1} \leq \lambda^n$$

Standard Analysis of `mis`

Thr: Algorithm `mis` solves MIS in $O^*(1.33^n)$ time

Prf:

- When the algorithm branches at a node v with $d(v) \geq 4$, in one subproblem it removes 1 node (i.e. v), and in the other it removes $1 + d(v) \geq 5$ nodes (i.e. $N[v]$):

$$\begin{aligned} P(n) &\leq P(n-1) + P(n-5) \\ &\leq \lambda^{n-1} + \lambda^{n-5} \leq \lambda^n \qquad\qquad (\lambda \geq 1.32\ldots) \end{aligned}$$

Standard Analysis of `mis`

Thr: Algorithm `mis` solves MIS in $O^*(1.33^n)$ time

Prf:

- Otherwise, the algorithm branches at a node v of degree exactly 3, hence removing either 1 or 4 nodes. However, in the first case a node of degree 2 is folded afterwards, with the removal of at least 2 more nodes

$$\begin{aligned} P(n) &\leq P(n-3) + P(n-4) \\ &\leq \lambda^{n-3} + \lambda^{n-4} \leq \lambda^n \qquad (\lambda \geq 1.22\ldots) \end{aligned}$$

Standard Analysis of `mis`

Thr: Algorithm `mis` solves MIS in $O^*(1.33^n)$ time

Prf:

• Otherwise, the algorithm branches at a node v of degree exactly 3, hence removing either 1 or 4 nodes. However, in the first case a node of degree 2 is folded afterwards, with the removal of at least 2 more nodes

$$\begin{aligned} P(n) &\leq P(n-3) + P(n-4) \\ &\leq \lambda^{n-3} + \lambda^{n-4} \leq \lambda^n \qquad\qquad (\lambda \geq 1.22\ldots) \end{aligned}$$

Rem: This is the best one can get with a standard analysis

Measure & Conquer

Measure & Conquer

- The classical approach to *improve* on `mis` would be designing refined branching and reduction rules. In particular, one tries to improve on the *tight* recurrences. The analysis is then performed in a similar fashion

- In the standard analysis, n is both the measure used in the analysis and the quantity in terms of which the final time bound is expressed

- However, one is free to use any, possibly sophisticated, measure m in the analysis, provided that $m \leq f(n)$ for some known function f

- This way, one achieves a time bound of the kind $O^*(\lambda^m) = O^*(\lambda^{f(n)})$, which is in the desired form

Measure & Conquer

- The idea behind Measure & Conquer is focusing on the choice of the measure

- In fact, a more sophisticated measure may capture phenomena which standard measures are not able to exploit, and hence lead to a tighter analysis of a *given* algorithm

- We next show how to get a much better time bound for `mis` thanks to a better measure of subproblems size (without changing the algorithm!)

- We will start by introducing an alternative, simple, measure. This measure does not immediately give a better time bound, but it will be a good starting point to define a really better measure

An Alternative Measure

- Nodes of degree ≤ 2 can be removed without branching
- Hence they do not really contribute to the *size* of the problem
- For example, if the maximum degree is 2, then `mis` solves the problem in polynomial time!

Idea: define the size of the problem to be the number of nodes of degree at least 3

An Alternative Measure

Def: Let n_i be the number of nodes of degree i, and $n_{\geq i} = \sum_{j \geq i} n_j$

• We define the size of the problem to be $m = n_{\geq 3}$ (rather than $m = n$)

Rem: $m = n_{\geq 3} \leq n$. Hence, if we prove a running time bound in $O^*(\lambda^m)$, we immediately get a $O^*(\lambda^n)$ time bound

An Alternative Analysis

Thr: Algorithm `mis` solves MIS in $O^*(1.33^n)$ time

An Alternative Analysis

Thr: Algorithm `mis` solves MIS in $O^*(1.33^n)$ time

(Alternative) Prf:

- Let us define G a base instance if the maximum degree in G is 2 (which implies $m = n_{\geq 3} = 0$)

- Let moreover $P(m)$ be the number of base instances generated by the algorithm to solve an instance of size m

- By the usual argument the running time is $O^*(P(m))$. We prove by induction that $P(m) \leq \lambda^m$ for $\lambda < 1.33$

An Alternative Analysis

Thr: Algorithm `mis` solves MIS in $O^*(1.33^n)$ time

(Alternative) Prf:

- In the base case $m = 0$. Thus

$$P(0) = 1 \leq \lambda^0$$

- Let m' be the size of the problem after folding a node v. It is sufficient to show that $m' \leq m$, from which

$$P(m) \leq P(m') \leq \lambda^{m'} \leq \lambda^m$$

- This condition trivially holds when folding only removes nodes

An Alternative Analysis

Thr: Algorithm `mis` solves MIS in $O^*(1.33^n)$ time

(Alternative) Prf:

- In the remaining case $N(v) = \{u, w\}$ with $uw \notin E$. In this case we remove $\{v, u, w\}$ and add a node uw with $d(uw) \leq d(u) + d(w) - 2$. By case analysis $m' \leq m$ also in this case

$d(u)$	$d(w)$	$d(uw)$	m'
2	2	2	m
2	≥ 3	≥ 3	$m - 1 + 1$
≥ 3	≥ 3	≥ 4	$m - 2 + 1$

An Alternative Analysis

Thr: Algorithm `mis` solves MIS in $O^*(1.33^n)$ time

(Alternative) Prf:

- Suppose now that we branch at a node v with $d(v) \geq 4$. Note that all the nodes of the graph have degree ≥ 3 (since we do not fold). For $t_3 = |\{u \in N(v) : d(u) = 3\}|$,

$$\begin{aligned} P(m) &\leq P(m-1-t_3) + P(m-1-d(v)) \\ &\leq P(m-1) + P(m-5) \leq \lambda^{m-1} + \lambda^{m-5} \leq \lambda^m \quad (\lambda \geq 1.32\ldots) \end{aligned}$$

- Eventually, consider branching at v, $d(v) = 3$. In this case we remove either 1 or 4 nodes of degree 3. However, in the first case the degree of the 3 neighbors of v drops from 3 to 2, with a consequent further reduction of the size by 3

$$P(m) \leq P(m-4) + P(m-4) \leq \lambda^{m-4} + \lambda^{m-4} \leq \lambda^m \quad (\lambda \geq 1.18\ldots)$$

A Better Measure

- When we branch at a node of large degree, we decrement by 1 the degree of many other nodes
- This is beneficial on long term, since we can remove nodes of degree ≤ 2 without branching
- We are not exploiting this fact in the current analysis

Idea: assign a larger *weight* $\omega_i \leq 1$ to nodes of larger degree i, and let the size of the problem be the sum of node weights. This way, when the degree of a node decreases, the size of the problem decreases as well

A Better Measure

Def:

- for a constant $\omega \in (0, 1]$ to be fixed later,

$$\omega_i = \begin{cases} 0 & \text{if } i \le 2; \\ \omega & \text{if } i = 3; \\ 1 & \text{otherwise.} \end{cases}$$

- Let $\omega(v) = \omega_{d(v)}$
- the size $m = m(G)$ of G is

$$m = \sum_{v \in V(G)} \omega(v) = \omega \cdot n_3 + n_{\ge 4}$$

A Better Analysis

Thr: Algorithm `mis` solves MIS in $O^*(1.29^n)$ time

Prf:

- With the usual notation, let us show that $P(m) \leq \lambda^m$ for $\lambda < 1.29$

- In the base case $m = 0$, $P(0) = 1 \leq \lambda^0$

- In case of folding, let m' be the size of the subproblem. it is sufficient to show that $m' \leq m$. Then

$$P(m) \leq P(m') \leq \lambda^{m'} \leq \lambda^m$$

- This condition is satisfied when nodes are only removed (being the weight increasing with the degree)

A Better Analysis

Thr: Algorithm `mis` solves MIS in $O^*(1.29^n)$ time

Prf:

- The unique remaining case is that $N(v) = \{u, w\}$, with u and w not adjacent. In this case we remove $\{v, u, w\}$, and add a node uw with $d(uw) \leq d(u) + d(w) - 2$. Hence it is sufficient to show that

$$\omega(v) + \omega(u) + \omega(w) - \omega(sw) = \omega(u) + \omega(w) - \omega(uw) \geq 0$$

A Better Analysis

Thr: Algorithm `mis` solves MIS in $O^*(1.29^n)$ time

Prf:

- By a simple case analysis

$d(u)$	$d(w)$	$d(uw)$	$\omega(u) + \omega(w) - \omega(uw) \geq 0$
2	2	2	$0 + 0 - 0 \geq 0$
2	3	3	$0 + \omega - \omega \geq 0$
2	≥ 4	≥ 4	$0 + 1 - 1 \geq 0$
3	3	4	$\omega + \omega - 1 \geq 0$
3	≥ 4	≥ 4	$\omega + 1 - 1 \geq 0$
≥ 4	≥ 4	≥ 4	$1 + 1 - 1 \geq 0$

- We can conclude that $\omega \geq \frac{1}{2}$ (new constraint on the weights!)

A Better Analysis

Thr: Algorithm `mis` solves MIS in $O^*(1.29^n)$ time

Prf:

- Consider now branching at a node v, $d(v) \geq 5$. Let d_i be the degree of the ith neighbor of v (which thus has weight ω_{d_i}). Then

$$P(m) \leq P(m - \omega_{d(v)} - \sum_i (\omega_{d_i} - \omega_{d_i - 1})) + P(m - \omega_{d(v)} - \sum_i \omega_{d_i})$$

$$\leq P(m - 1 - \sum_{i=1}^{5} (\omega_{d_i} - \omega_{d_i - 1})) + P(s - 1 - \sum_{i=1}^{5} \omega_{d_i})$$

- Observe that we can replace $d_i \geq 6$ with $d_i = 5$. In fact in both cases $\omega_{d_i} = 1$ and $\omega_{d_i} - \omega_{d_i - 1} = 0$. Hence we can assume $d_i \in \{3, 4, 5\}$ (finite number of recurrences!!!)

A Better Analysis

Thr: Algorithm `mis` solves MIS in $O^*(1.29^n)$ time

Prf:

- By case enumeration

$$P(m) \leq \begin{cases} P(m - 1 - 5\cdot\omega - 0\cdot(1\text{-}\omega) - 0\cdot 0) + P(m - 1 - 5\cdot\omega - 0\cdot 1 - 0\cdot 1) \\ P(m - 1 - 4\cdot\omega - 1\cdot(1\text{-}\omega) - 0\cdot 0) + P(m - 1 - 4\cdot\omega - 1\cdot 1 - 0\cdot 1) \\ P(m - 1 - 4\cdot\omega - 0\cdot(1\text{-}\omega) - 1\cdot 0) + P(m - 1 - 4\cdot\omega - 0\cdot 1 - 1\cdot 1) \\ P(m - 1 - 3\cdot\omega - 2\cdot(1\text{-}\omega) - 0\cdot 0) + P(m - 1 - 3\cdot\omega - 2\cdot 1 - 0\cdot 1) \\ \quad\ldots \\ P(m - 1 - 0\cdot\omega - 0\cdot(1\text{-}\omega) - 5\cdot 0) + P(m - 1 - 0\cdot\omega - 0\cdot 1 - 5\cdot 1) \end{cases}$$

A Better Analysis

Thr: Algorithm `mis` solves MIS in $O^*(1.29^n)$ time

Prf:

- Consider now branching at a node v, $d(v) = 4$. By a similar argument (but with $d_i \in \{3, 4\}$)

$$P(m) \leq \begin{cases} P(m - 1 - 4\cdot\omega - 0\cdot(1-\omega)) + P(m - 1 - 4\cdot\omega - 0\cdot 1) \\ P(m - 1 - 3\cdot\omega - 1\cdot(1-\omega)) + P(m - 1 - 3\cdot\omega - 1\cdot 1) \\ P(m - 1 - 2\cdot\omega - 2\cdot(1-\omega)) + P(m - 1 - 2\cdot\omega - 2\cdot 1) \\ P(m - 1 - 1\cdot\omega - 3\cdot(1-\omega)) + P(m - 1 - 1\cdot\omega - 3\cdot 1) \\ P(m - 1 - 0\cdot\omega - 4\cdot(1-\omega)) + P(m - 1 - 0\cdot\omega - 4\cdot 1) \end{cases}$$

A Better Analysis

Thr: Algorithm `mis` solves MIS in $O^*(1.29^n)$ time

Prf:

• Consider eventually branching at a node v, $d(v) = 3$. By an analogous argument (but with $\omega(v) = \omega_3 = \omega$ and $d_i = 3$)

$$P(m) \leq P(m - \omega - 3\,\omega) + P(m - \omega - 3\,\omega)$$

• For every $\omega \in [0.5, 1]$, the set of recurrences above provides an upper bound $\lambda(\omega)$ on λ. Our goal is minimizing $\lambda(\omega)$ (hence getting a better time bound)

• Via exhaustive (grid) enumeration, we obtained $\omega = 0.7$ which gives $\lambda(\omega) < 1.29$

An Even Better Measure

- We can extend the previous approach to larger degrees

$$\omega_i = \begin{cases} 0 & \text{if } i \leq 2; \\ \omega & \text{if } i = 3; \\ \omega' & \text{if } i = 4; \\ 1 & \text{otherwise.} \end{cases}$$

where $0 < \omega \leq \omega' \leq 1$

Thr 3: Algorithm `mis` solves MIS in $O^*(1.26^n)$ time

Exercises

Exr 5: Prove Thr 3 (Hint: $\omega = 0.750$, $\omega' = 0.951$)

Exr 6: What do you expect that would happen if we added one extra weight $\omega_5 = \omega''$? Can you guess any pattern?

Exr 7*: Design a better algorithm for MIS, using possibly the other mentioned reduction rules. Analyze your algorithm in the standard way and via Measure & Conquer

Exr 8*: Can you imagine an alternative, promising measure for MIS?

Quasiconvex Analysis of Backtracking Algorithms

Optimal Weights Computation

- When the number of distinct weights grows, an exhaustive exploration might be too slow
- We next describe a general tool to perform this computation in an (exponentially) faster way

Multivariate Recurrences

- Consider a collection of integral *measures* $m_1, \ldots, m_d$, describing different aspects of the size of the problem considered

EG: In the analysis of `mis` we used $m_1 = n_3$ and $m_2 = n_{\geq 4}$

- These measure naturally induce a set of multivariate recurrence of the following kind for each branching b

$$P(m_1, \ldots, m_d) \leq P(m_1 - \delta^b_{1,1}, \ldots, m_d - \delta^b_{d,1}) + \ldots$$
$$+ P(m_1 - \delta^b_{1,h(b)}, \ldots, m_d - \delta^b_{d,h(b)})$$

Rem: some of the $\delta^b_{i,j}$ might be negative. For example, deleting one edge incident to a node of degree 4, we decrease $n_{\geq 4}$ but increase n_3

Multivariate Recurrences

- Solving multivariate recurrences is typically rather complicated

- A common alternative is turning them into univariate recurrences by considering a linear combination of the measures (*aggregated measure*)

$$m(\alpha) = \alpha_1\, m_1 + \ldots + \alpha_d\, m_d$$

- The *weights* α_i must satisfy the condition $\delta_j^b := \sum_i \alpha_i\, \delta_{i,j}^b > 0$, i.e. $m(\alpha)$ decreases in each subproblem (we allow ≥ 0 for $h = 1$)

EG: In the analysis of `mis` we used $\alpha_1 = \omega$ and $\alpha_2 = 1$. The condition is satisfied for every $\omega \in [0.5, 1]$

Multivariate Recurrences

- The resulting set of univariate recurrences can be solved in the standard way (for fixed weights)

- In particular, for each branching b we compute the (unique) positive root $\lambda^b(\alpha)$ of

$$f^b(\lambda, \alpha) := 1 - \sum_j \lambda^{-\sum_i \alpha_i \delta^b_{i,j}}$$

- This gives a running time bound of the kind $O^*(\lambda(\alpha)^{\sum_i \alpha_i m_i})$ where

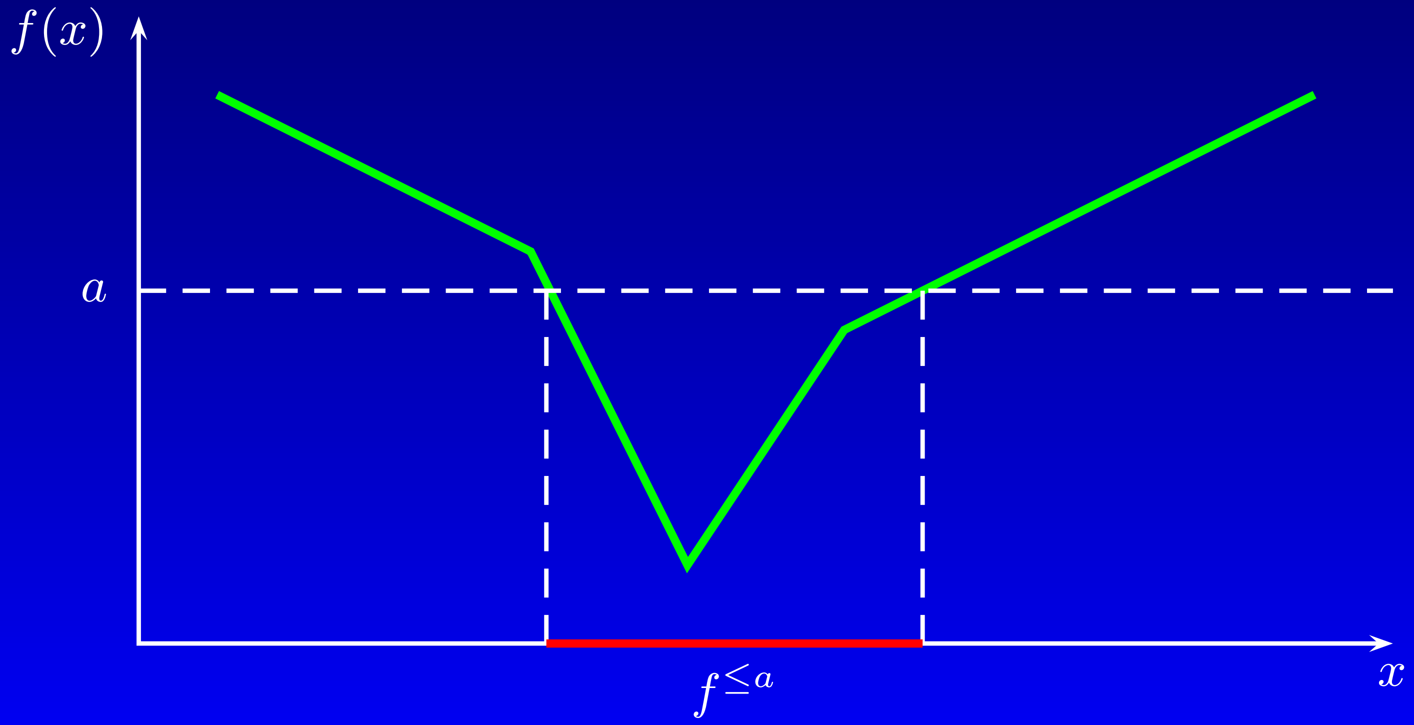
$$\lambda(\alpha) := \max_b \lambda^b(\alpha)$$

Quasiconvex Functions

Def: A function $f : D \to \mathbb{R}$, with $D \subseteq \mathbb{R}^d$ convex, is quasiconvex if the set

$$f^{\leq a} := \{x \in D : f(x) \leq a\}$$

is convex for any $a \in \mathbb{R}$



Quasiconvex Functions

Thr [Eppstein'01]: Function $\lambda(\alpha)$, $\alpha \in \mathbb{R}^d$, is quasiconvex

Prf:

- Since the max of a finite number of quasiconvex functions is quasiconvex, it is sufficient to show that each $\lambda^b(\alpha)$ is quasiconvex

- $\lambda^b(\alpha)$ is the positive root of $f^b(\lambda, \alpha) = 1 - \sum_j \lambda^{-\sum_i \alpha_i \delta^b_{i,j}}$

- Hence
$\lambda^{b,\leq a} = \{\alpha \in \mathbb{R}^d : \lambda^b(\alpha) \leq a\} = \{\alpha \in \mathbb{R}^d : \sum_j a^{-\sum_i \alpha_i \delta^b_{i,j}} \leq 1\}$

- $g^b(\alpha) := \sum_j a^{-\sum_i \alpha_i \delta^b_{i,j}}$ is convex as sum of convex functions, and trivially its level sets are convex, including $g^{b,\leq 1}$

Cor: Function $\lambda(\alpha)$ is quasiconvex over any convex $D \subseteq \mathbb{R}^d$

Applications to M&C

- We can use these facts to optimize the weights much faster in the Measure & Conquer framework

- Suppose we define a set of linear constraints on the weights such that

(a) the size of each subproblem does not increase

(b) the initial measure $m = m(\alpha)$ is upper bounded by n, where n is a *standard* measure for the problem

- This gives a convex domain of weights α. On that domain we can compute the minimum value $\lambda(\tilde{\alpha})$ of the quasiconvex function $\lambda(\alpha)$

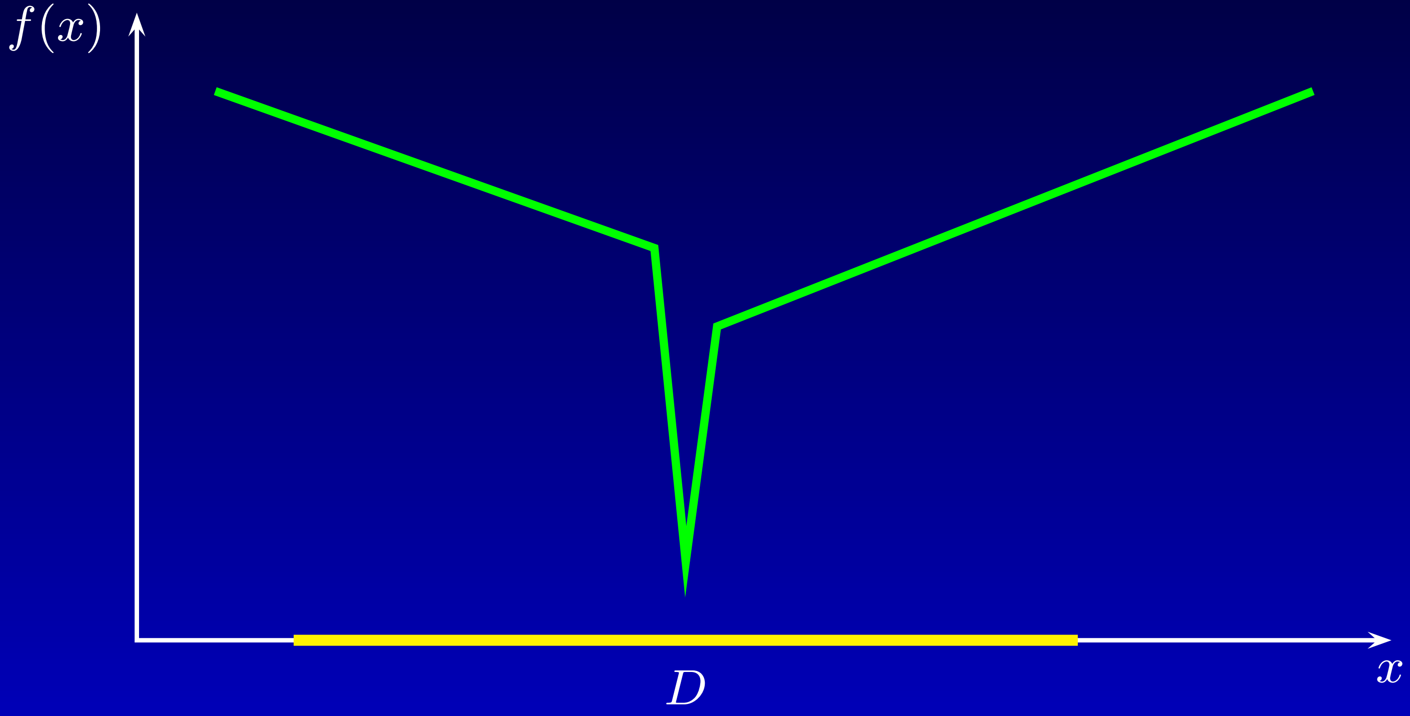
- The resulting running time is $O^*(\lambda(\tilde{\alpha})^{m(\tilde{\alpha})}) = O^*(\lambda(\tilde{\alpha})^n)$

Randomized Local Search

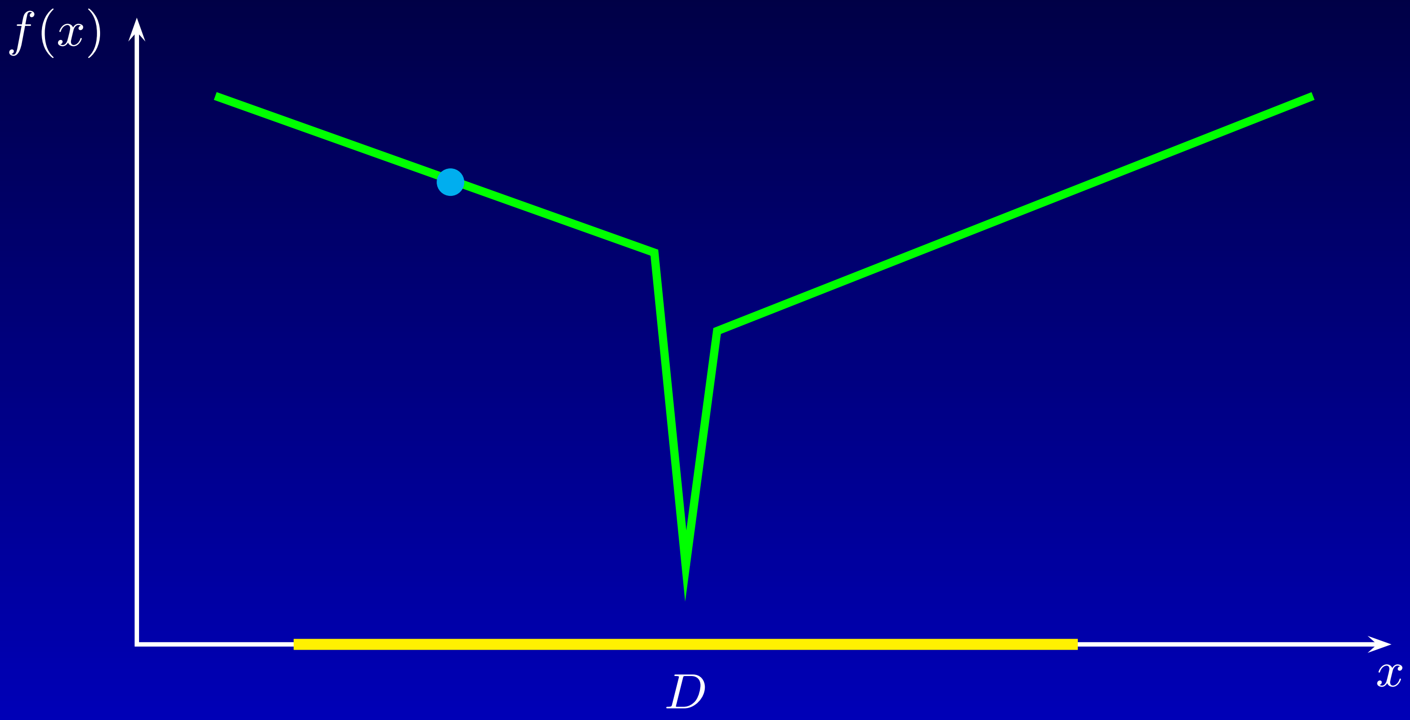
- There are known techniques to find efficiently the minimum of a quasi-convex functions (see e.g. [Eppstein'01,Gaspers])

- We successfully applied the following, very fast and easy to implement, approach based on *randomized local search* (in simulated annealing style)
  - ◇ We start from any feasible initial value α
  - ◇ We add to it a random vector in a given range $[-\Delta, \Delta]^d$
  - ◇ If the resulting α' is feasible and gives $\lambda(\alpha') \leq \lambda(\alpha)$, we set $\alpha = \alpha'$
  - ◇ We iterate the process, reducing the value of Δ if no improvement is achieved for a large number of steps
  - ◇ The process halts when Δ drops below a given value Δ'

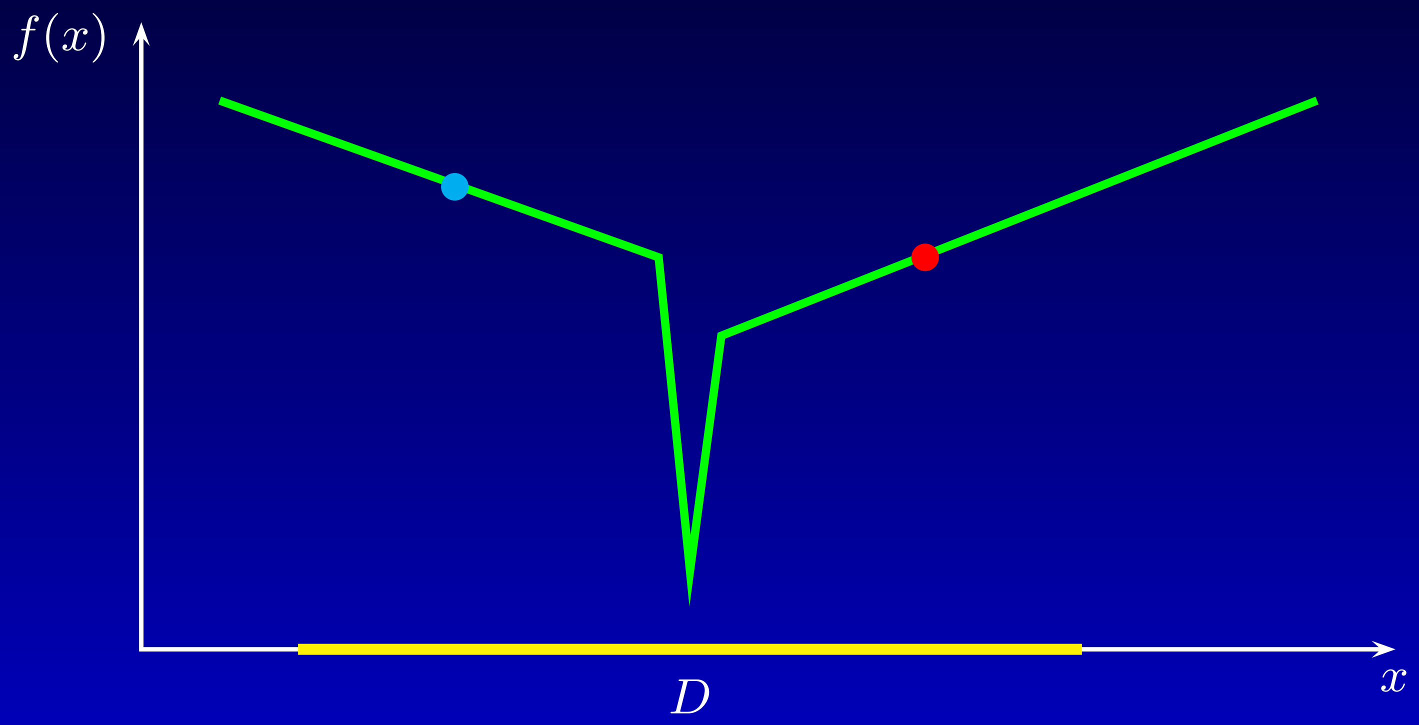
Randomized Local Search



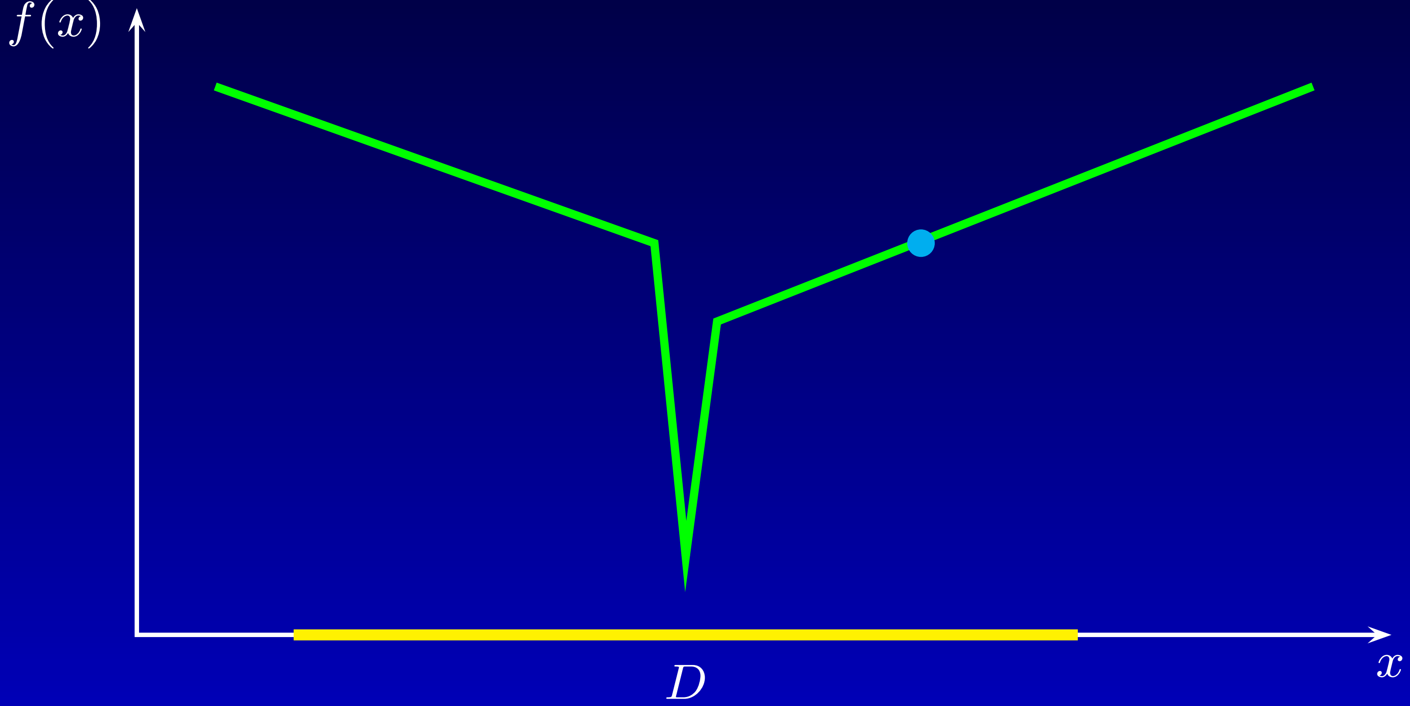
Randomized Local Search



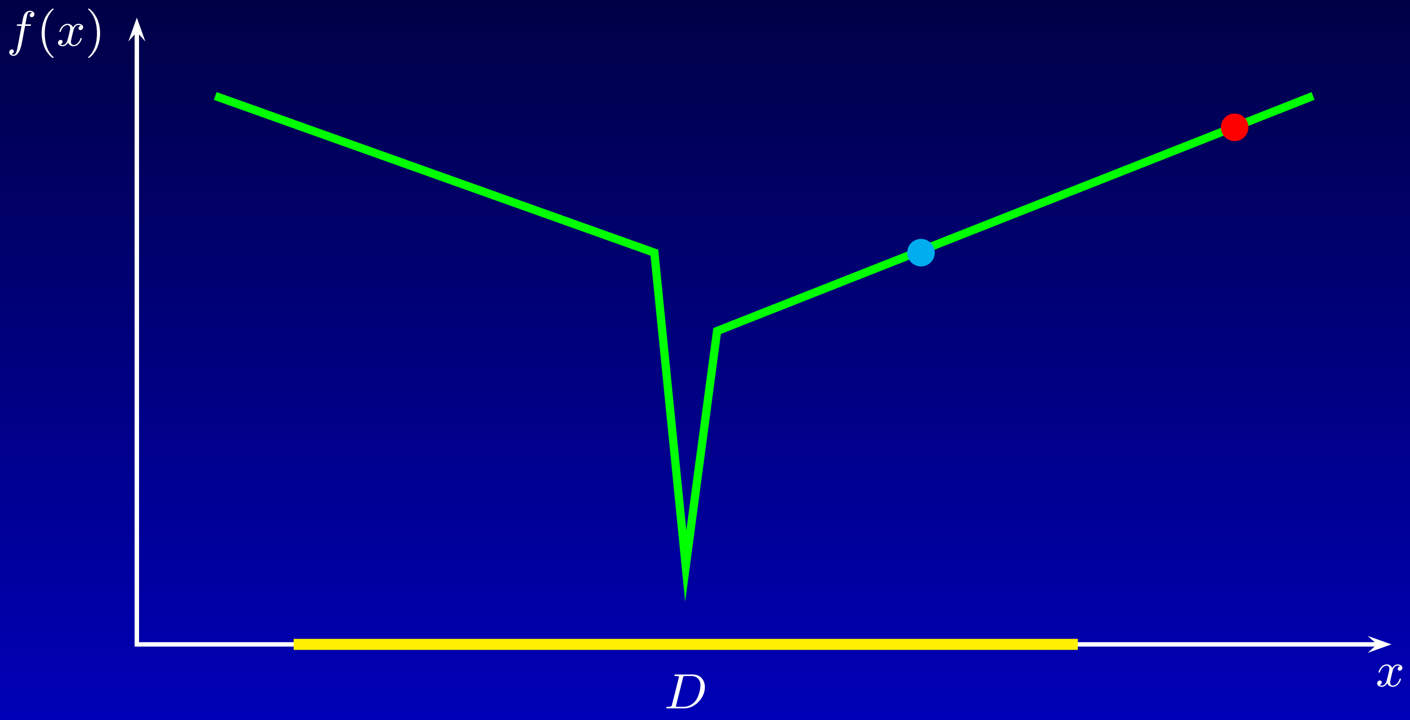
Randomized Local Search



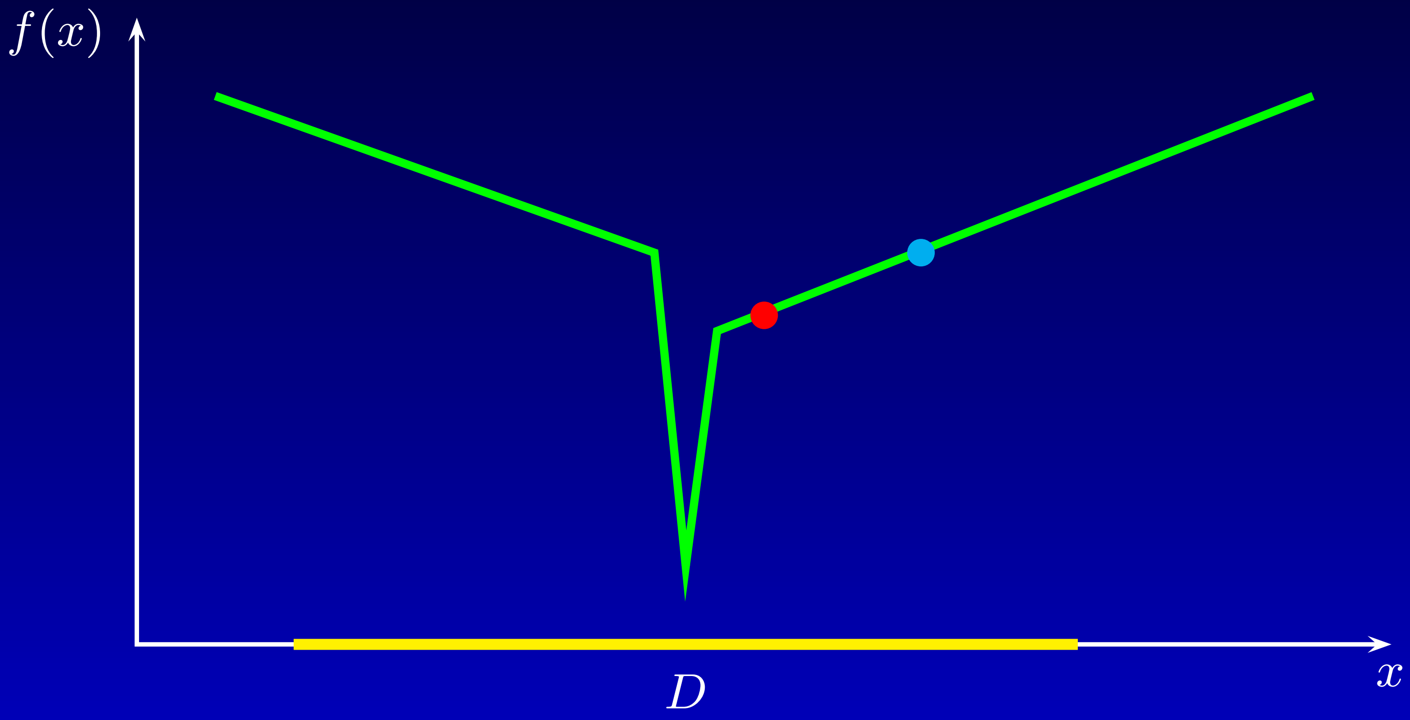
Randomized Local Search



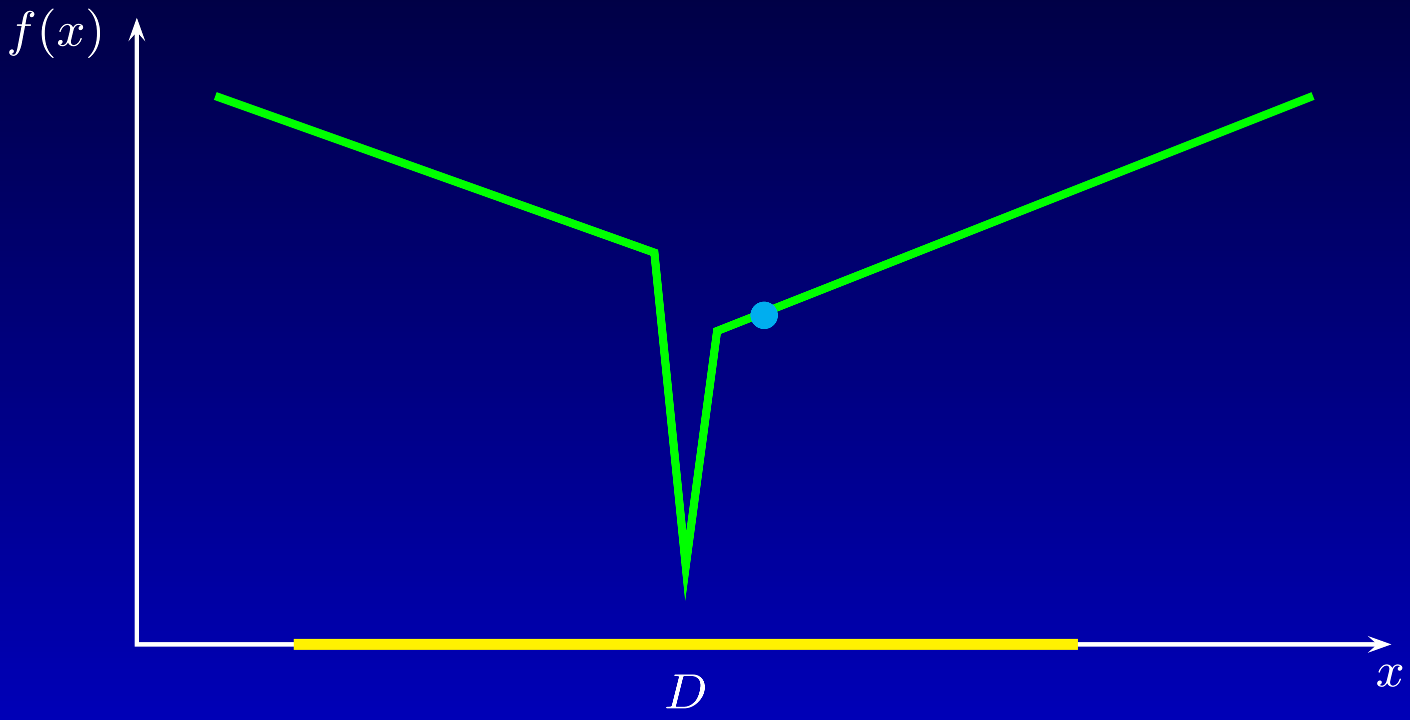
Randomized Local Search



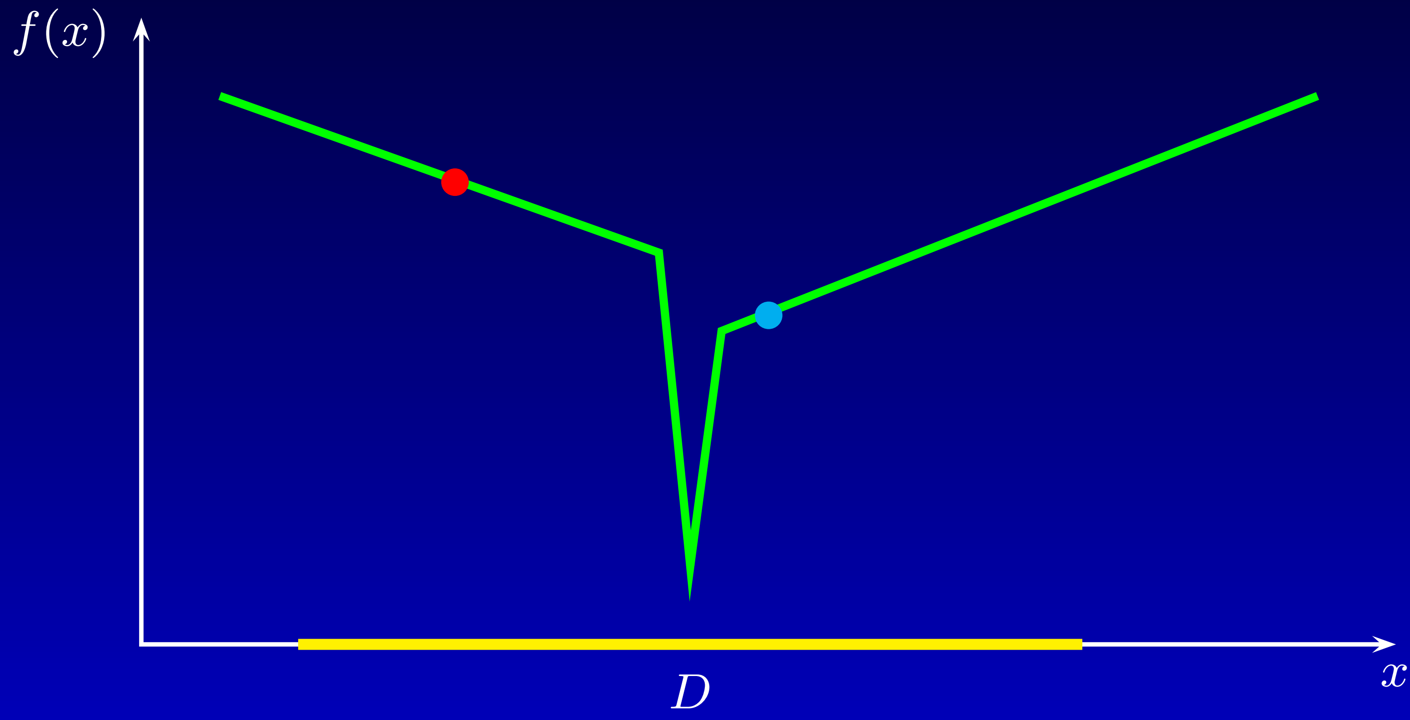
Randomized Local Search



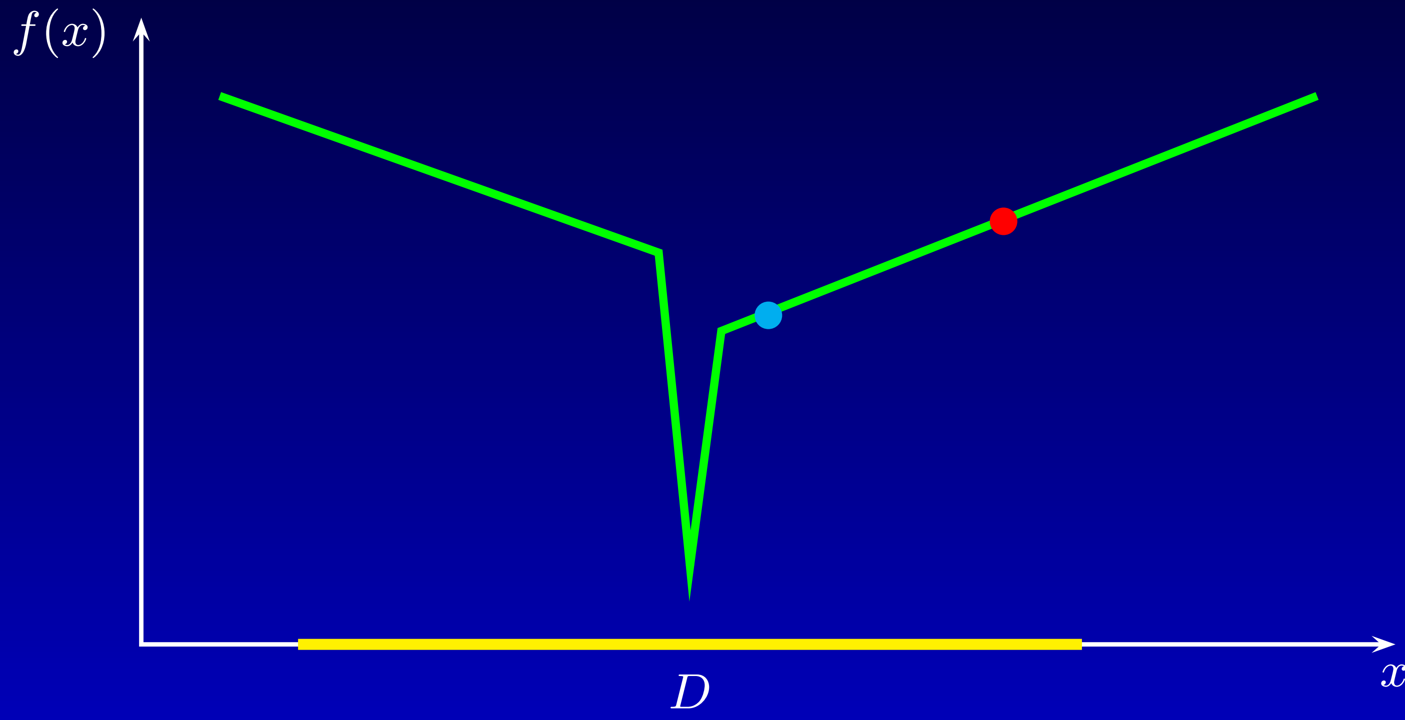
Randomized Local Search



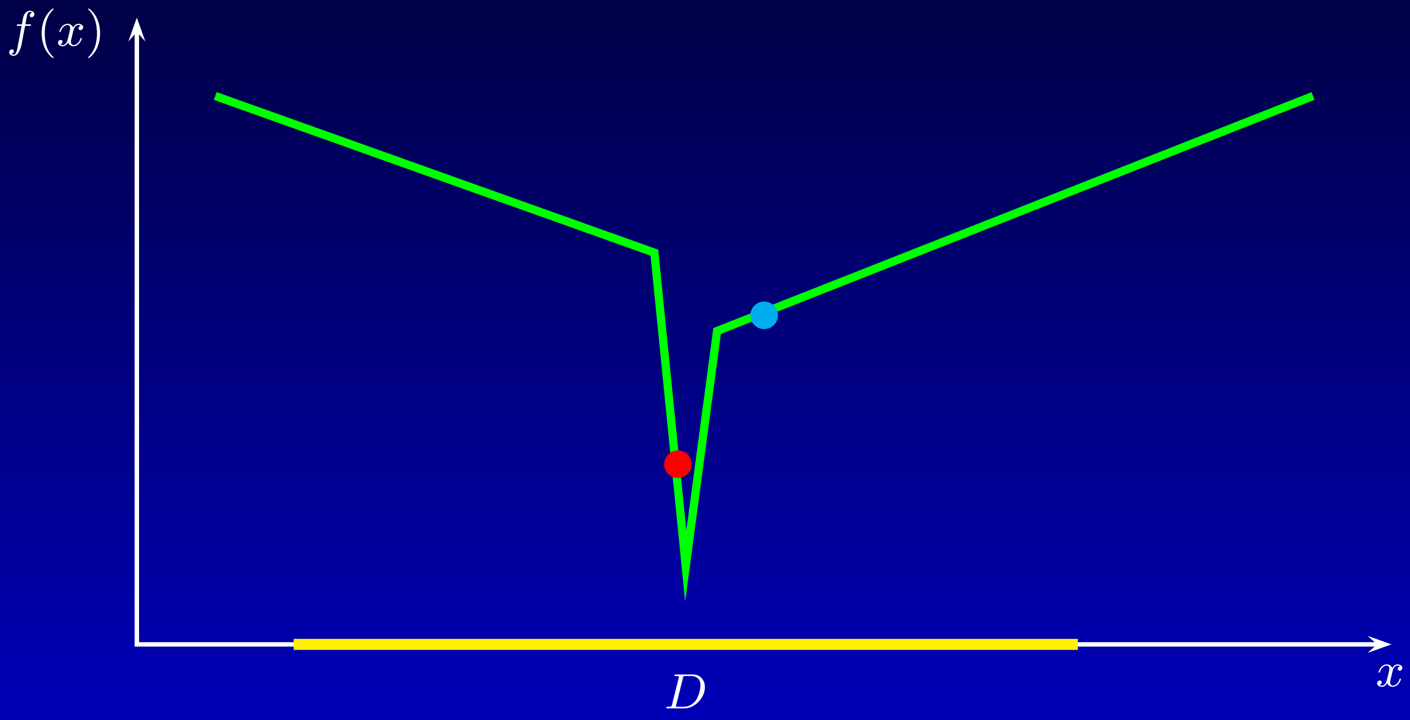
Randomized Local Search



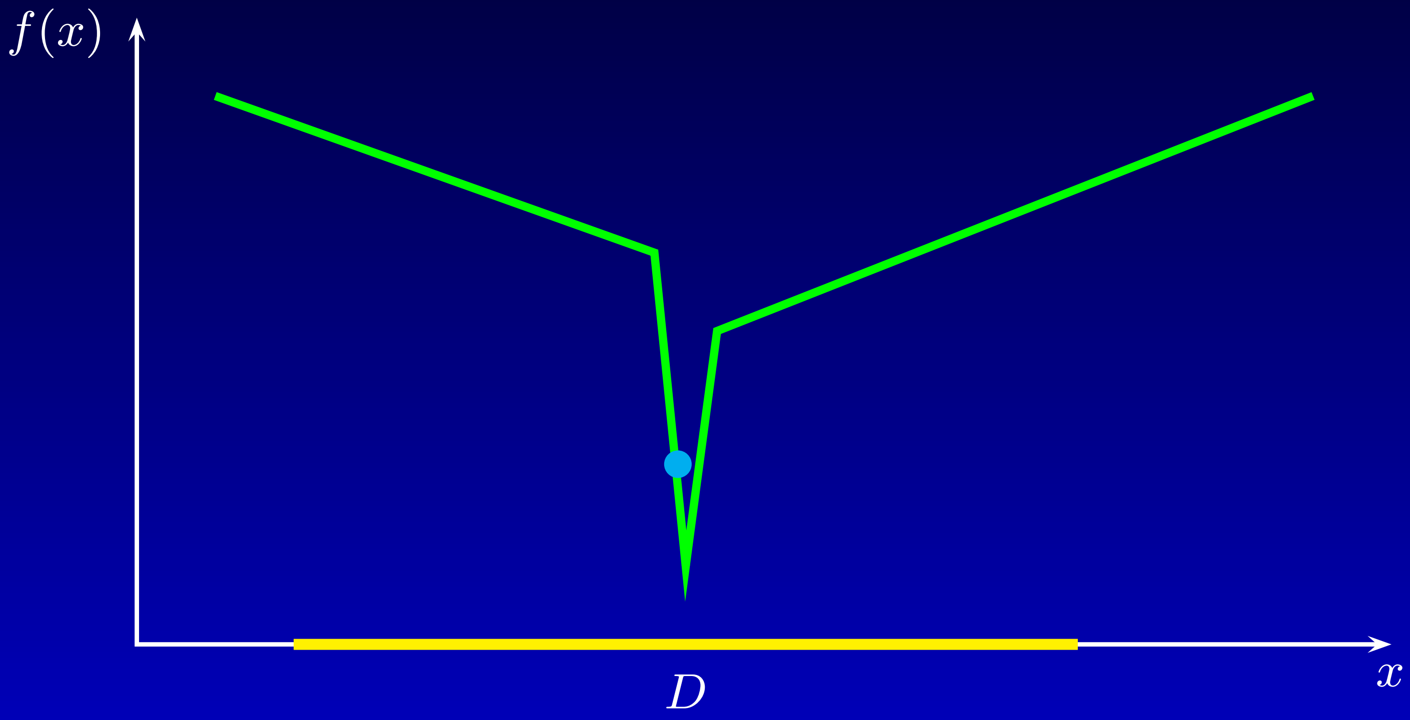
Randomized Local Search



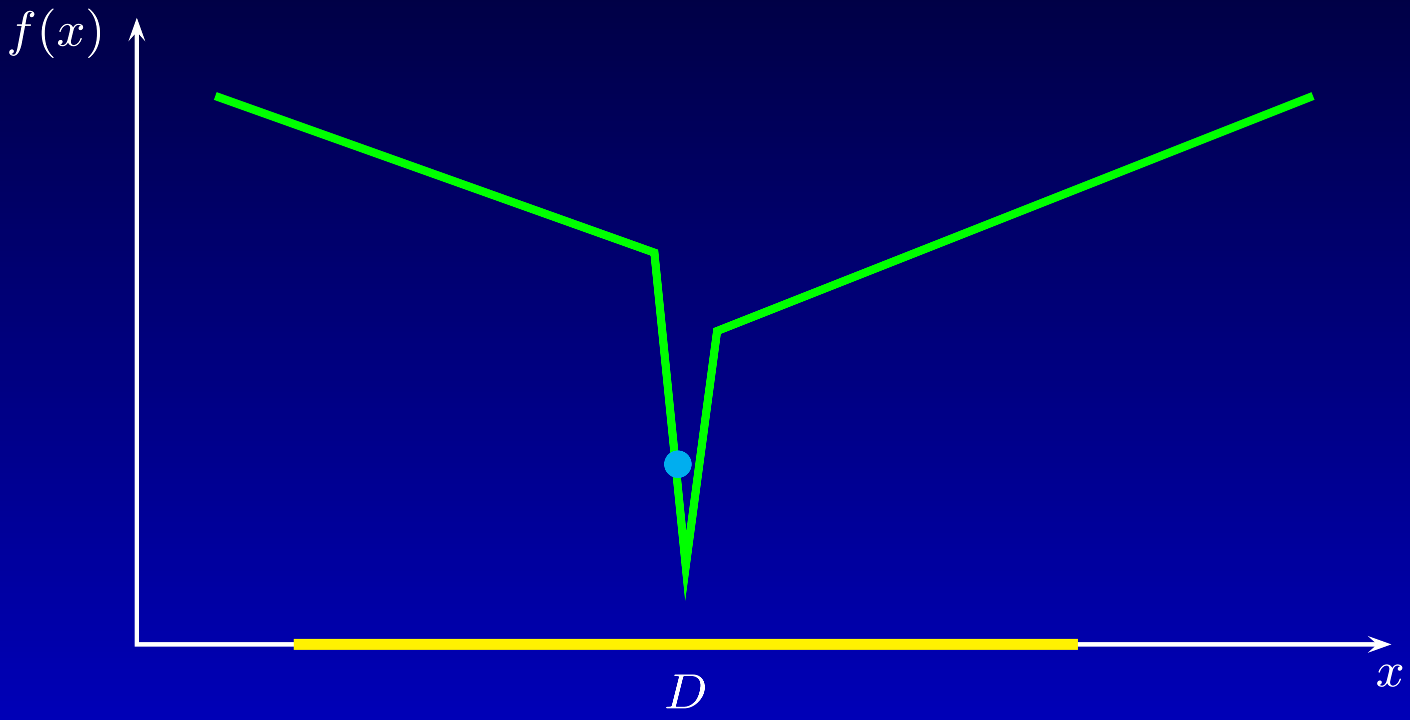
Randomized Local Search



Randomized Local Search



Randomized Local Search



Rem: This algorithm does not guarantee closeness to the optimal $\lambda(\tilde{\alpha})$. However it is accurate in practice. More important, it provides *feasible* upper bounds

Lower Bounds

Lower Bounds

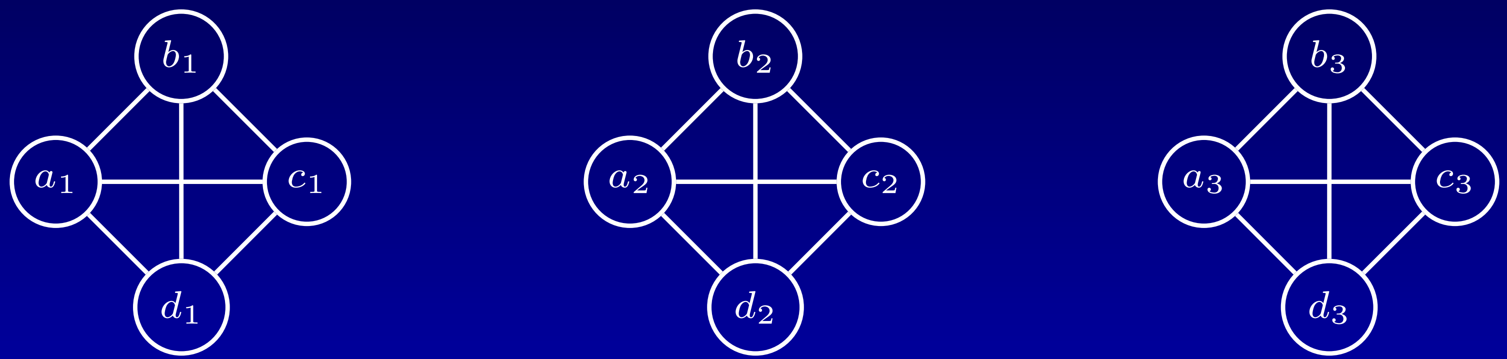
- Measure & Conquer sometimes leads to much better running time bounds
- Still, these bounds might not be tight
- Hence, it makes sense to search for (exponential) lower bounds on the running time of the algorithm considered (not of the problem!)
- A lower bound may give an idea of how far the analysis is from being tight

A Lower Bound for `mis`

Thr 4: The running time of `mis` is $\Omega(2^{n/4})$

Prf:

- Consider the graph G_k consisting of $k = n/4$ copies of a K_4



- The algorithm might branch at a_1. In both subproblems $\{a_1, b_1, c_1, d_1\}$ is removed, either immediately or later on by folding. This leaves a G_{k-1}

- We obtain a recurrence of the type $P(n) \geq 2P(n-4)$ for the number of subproblems, which gives $P(n) \geq 2^{n/4}$

A Lower Bound for `mis`

Thr 4: The running time of `mis` is $\Omega(2^{n/4})$

Exr 8: Find a larger lower bound on the running time of `mis` (Hint: $\Omega(3^{n/6}) = \Omega(1.20^n)$, maybe better)

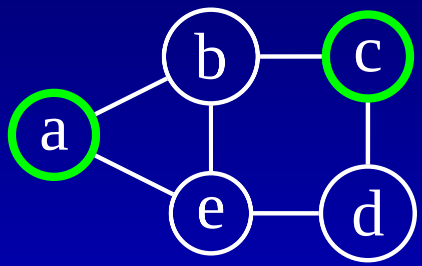
Exr 9*: Consider the variant of `mis` where the algorithm, after the base case, branches on connected components when possible. Can you find a good lower bound on the running time of this modified algorithm?

Rem: Typically finding lower bounds on connected graphs is much more complicated

Applications of Measure & Conquer

Independent Set

Def: Given $G = (V, E)$, the *independent set* problem (MIS) is to determine the maximum cardinality $\alpha(G)$ of a subset of pairwise non-adjacent nodes (*independent set*)



$$\alpha(G) = 2$$

Independent Set

Thr [Fomin,Grandoni,Kratsch'06-'09]: MIS can be solved in $O^*(1.221^n)$ time and polynomial space

Prf:

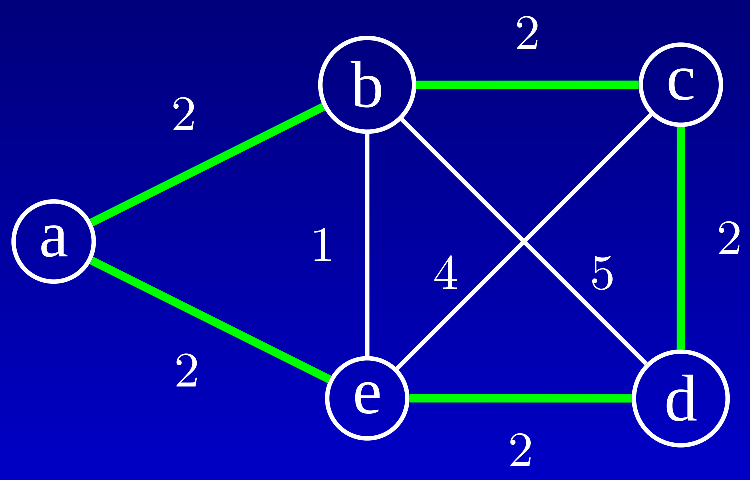
- Simple branching algorithm

int `mis`(G) {
  if($|V(G)| \leq 1$) **return** $|V(G)|$;
  if($\exists$ component $C \subset G$) **return** `mis`(C)+`mis`($G - C$);
  if($\exists$ vertices v and w: $N[w] \subseteq N[v]$) **return** `mis`($G - \{v\}$);
  if($\exists$ a vertex v, with $d(v) = 2$) **return** 1+`mis`(G_v);
  select a vertex v of maximum degree, which minimizes $|E(N(v))|$;
  return $\max\{$`mis`$(G - \{v\} - M(v))$, 1+`mis`$(G - N[v])\}$;
}

- Analysis similar to the one outlined before

Traveling Salesman Problem

Def: Given a weighted $G = (V, E)$, the *traveling salesman problem* problem (TSP) is to compute a minimum weight cycle spanning V (*TSP tour*)



Traveling Salesman Problem

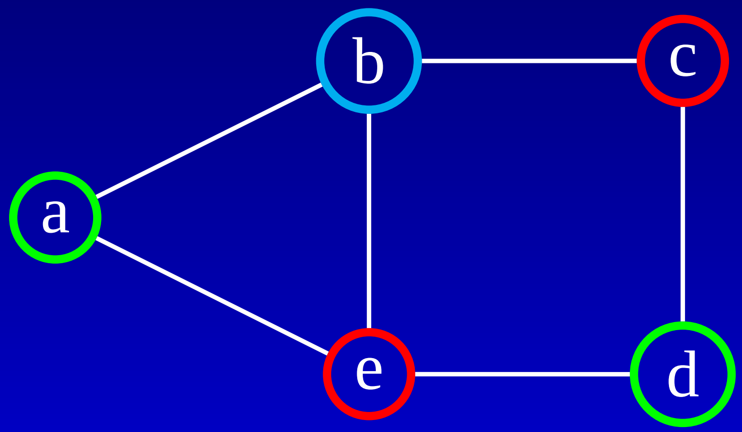
Thr [Eppstein'03-'07]: TSP can be solved in $O^*(1.260^n)$ time in cubic graphs

Prf:

- Design a non-trivial branching algorithm
- Analyze it using, as measure, $|V| - |F| - |C| \leq |V|$
- Here F is a set of *forced* edges and C the set of 4-cycles of G which induce connected components in $G - F$

3-Coloring

Def: Given $G = (V, E)$ and a set of 3 colors, the 3-*coloring* problem (3-COL) is to find an assignment of colors to nodes such that adjacent nodes are colored differently



3-Coloring

Def: Given a set of variables on domains of size $\leq a$ and a set of constraints each one involving at most b variables, the (a, b)-*constraint satisfaction* problem (CSP) is to find an assignment of the variables satisfying all the constraints

Rem: 3-COL is a special case of $(3, 2)$-CSP

3-Coloring

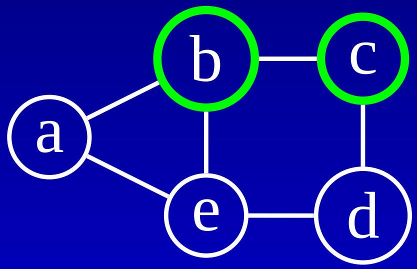
Thr [Beigel,Eppstein'00-'05]: 3-COL can be solved in $O^*(1.329^n)$ time

Prf:

- Non-trivial reduction to $(3, 2)$-CSP
- Non-trivial branching algorithm solving $(4, 2)$-CSP in $O^*(1.365^n)$ time
- In the analysis the measure is a linear combination $n_3 + \alpha\, n_4$ of the number of variables with domain of size 3 and 4 (variables with smaller domain can be filtered out)

Dominating Set

Def: Given $G = (V, E)$, the *dominating set* problem (MDS) is to determine the minimum cardinality $\delta(G)$ of a subset of nodes D such that any node in $V - D$ is adjacent to some node in D (*dominating set*)

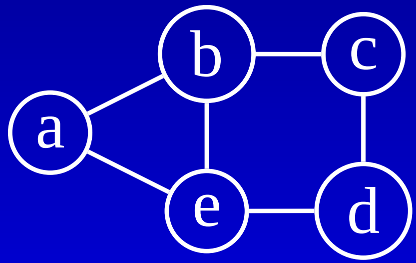


$$\delta(G) = 2$$

Dominating Set

Def: Given a universe $\mathcal{U}$ and a collection of subsets $\mathcal{S} \subseteq 2^{\mathcal{U}}$, the *set cover* problem (MSC) is to determine a minimum cardinality subcollection $\mathcal{C} \subseteq \mathcal{S}$ such that $\cup_{S \in \mathcal{C}} S = \mathcal{U}$ (*set cover*)

Rem: MDS can reduced to MSC by letting $\mathcal{U} = V$ and $\mathcal{S} = \{N[v] : v \in V\}$. This instance has n subsets and n elements



$$\mathcal{U} = \{a, b, c, d, e\}$$

$$\begin{aligned} S_a &= \{a, b, e\} \\ S_b &= \{a, b, c, e\} \\ S_c &= \{b, c, d\} \\ S_d &= \{c, d, e\} \\ S_e &= \{a, b, d, e\} \end{aligned}$$

Dominating Set

Thr [Grandoni'04-'06]: MDS can be solved in $O^*(1.803^n)$ time

Proof: Design a simple algorithm solving MSC in $O^*(1.381^{|\mathcal{U}|+|\mathcal{S}|})$ time $\Rightarrow O^*(1.381^{2n})$ time algo for MDS

int msc($\mathcal{S}$) {
  if($|\mathcal{S}| = 0$) **return** 0;
  if($\exists S, R \in \mathcal{S} : S \subseteq R$) **return** msc($\mathcal{S}\backslash\{S\}$);
  if($\exists u \in \mathcal{U}(\mathcal{S})\exists$ a unique $S \in \mathcal{S} : u \in S$) **return** $1+$msc(del(S, $\mathcal{S}$));
  take $S \in \mathcal{S}$ of maximum cardinality;
  if($|S| = 2$) **return** poly-msc($\mathcal{S}$)
  return $\min\{$msc($\mathcal{S}\backslash\{S\}$), $1+$msc(del(S, $\mathcal{S}$))$\}$;
}

Exr 10: Prove the theorem above

Dominating Set

Thr [Fomin,Grandoni,Kratsch'05-'09]: MDS can be solved in $O^*(1.527^n)$ time

Proof:

- Consider the same reduction to MSC and the same algorithm as before

- Give a different weight to sets of different cardinality and to elements of different frequency

Exr 11*: Prove the theorem above

Thr [van Rooij,Bodlaender'08]: MDS can be solved in $O^*(1.507^n)$ time

Variants of Dominating Set

Def: Given $G = (V, E)$, the *minimum independent dominating set* problem (MIDS) is to determine the minimum cardinality of a dominating set of G which is also an independent set

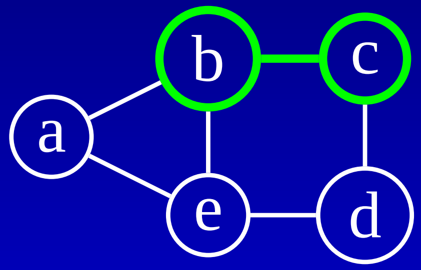
Thr [Gasper,Liedloff'06]: MIDS can be solved in $O^*(1.358^n)$ time

Def: Given $G = (V, E)$, the *minimum dominating clique* problem (MDC) is to determine the minimum cardinality of a dominating set of G which is also a clique

Thr [Kratsch,Liedloff'07]: MDC can be solved in $O^*(1.324^n)$ time

Connected Dominating Set

Def: Given $G = (V, E)$, the *connected dominating set* problem (ConDomS) is to determine the minimum cardinality $\delta'(G)$ of a dominating set of G which induces a connected graph (*connected dominating set*)



$$\delta'(G) = 2$$

Connected Dominating Set

Thr [Fomin,Grandoni,Kratsch'06-'08]: Connected dominating set can be solved in $O^*(1.941^n)$ time

Proof:

- Design an algorithm which gradually expands a connected graph, until it becomes dominating

- Assign a different weight to nodes dominating a different number of nodes not yet dominated

- Assign an extra weight to nodes which are still not selected nor discarded, giving a smaller extra weight to nodes whose removal makes the problem infeasible

Rem: without the refined measure one does not improve on trivial 2^n!

Combinatorial Bounds via M&C

- M&C can be used to derive better combinatorial bounds

Thr [Fomin,Grandoni,Pyatkin,Stepanov'05-'08]: An n-node graph has $O^*(1.716^n)$ minimal dominating sets

Prf: Design a listing algorithm and analyze it via M&C

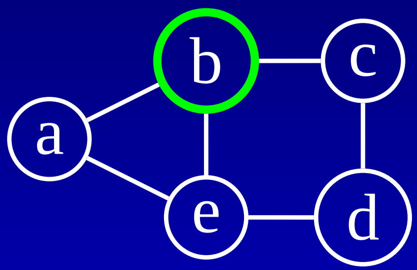
- Listing algorithms can often be used to solve weighted problems, where reduction rules are harder to get

Thr [Fomin,Grandoni,Pyatkin,Stepanov'05]: The weighted minimum dominating set problem can be solved in $O^*(1.578^n)$ time

Prf: Use a variant of the listing algorithm above, implementing a trivial weighted set cover reduction rule

Feedback Vertex Set

Def: Given $G = (V, E)$, the *feedback vertex set* problem (FVS) is to determine the minimum cardinality $\phi(G)$ of a subset of nodes whose removal makes G acyclic (*feedback vertex set*)



$$\phi(G) = 1$$

Feedback Vertex Set

Thr [Razgon+Fomin,Gaspers,Pyatkin'06-'08]: FVS can be solved in $O^*(1.755^n)$ time

Prf:

- Design an algorithm based on branching rules and maximum independent sets computation to solve the equivalent maximum induced forest problem
- Analyze it using, as measure,

$$0 \cdot |F| + 1 \cdot |N(t)| + (1+\alpha)|V - F - N(t)|$$

- Here F is a set of forced nodes and t is an *active* node

Apologies

I apologize for related and improved results that I forgot to mention

THANKS!!!