Exact Algorithms via Monotone Local Search

Saket Saurabh

The Institute of Mathematical Sciences, India
and University of Bergen, Norway.
(Joint work with Fedor V. Fomin, Daniel Lokshtanov and Serge Gaspers)

Hong Kong Theory Day 2017,
Hong Kong, January 6, 2017
Outline

- Introduction to Parameterized and Exact Algorithms
Outline

- Introduction to Parameterized and Exact Algorithms
- Basic Questions
Outline

- Introduction to Parameterized and Exact Algorithms
- Basic Questions
- Local Search and Monotone Local Search
Outline

- Introduction to Parameterized and Exact Algorithms
- Basic Questions
- Local Search and Monotone Local Search
- Our Algorithm
Outline

- Introduction to Parameterized and Exact Algorithms
- Basic Questions
- Local Search and Monotone Local Search
- Our Algorithm
- Proof of Correctness and Running Time Analysis
Outline

- Introduction to Parameterized and Exact Algorithms
- Basic Questions
- Local Search and Monotone Local Search
- Our Algorithm
- Proof of Correctness and Running Time Analysis
- Applications
Outline

- Introduction to Parameterized and Exact Algorithms
- Basic Questions
- Local Search and Monotone Local Search
- Our Algorithm
- Proof of Correctness and Running Time Analysis
- Applications
- Counting and Derandomization
Introduction to Parameterized and Exact Algorithms
**Vertex Cover**

**Input:** A graph $G = (V, E)$ and a positive integer $k$.

**Parameter:** $k$

**Question:** Does there exist a subset $V' \subseteq V$ of size at most $k$ such that for every edge $(u, v) \in E$ either $u \in V'$ or $v \in V'$?
Example of Vertex Cover
Example of Vertex Cover
Search for 8 sized vertex cover
Search for 8 sized vertex cover
Search for 8 sized vertex cover
Search for 8 sized vertex cover
Search for 8 sized vertex cover
Search for 8 sized vertex cover
If there are $n$ nodes and we are searching for a $8$ sized vertex cover the algorithm essentially takes:

$$\binom{n}{8}$$

(time.
If there are $n$ nodes and we are searching for a 100 sized vertex cover the algorithm essentially takes:

$$\binom{n}{100}$$

time.
If there are $n$ nodes and we are searching for a $k$ sized vertex cover then the algorithm essentially takes:

$$\binom{n}{k}$$

time steps.
Algorithm

If there are $n$ nodes and we are searching for a $k$ sized vertex cover then the algorithm essentially takes:

\[
\binom{n}{k}
\]

time steps.

If the network/graph has 100’s of nodes then even finding a 100-sized vertex cover can take hundreds’s of centuries by searching.
Algorithm

If there are $n$ nodes and we are searching for a $k$ sized vertex cover then the algorithm essentially takes:

$$\binom{n}{k}$$

time steps.

If the network/graph has 100’s of nodes then even finding a 100-sized vertex cover can take hundreds’s of centuries by searching.

Is searching really necessary?
Algorithm

If there are $n$ nodes and we are searching for a $k$ sized vertex cover then the algorithm essentially takes:

$$\binom{n}{k}$$

time steps.

If the network/graph has 100’s of nodes then even finding a 100-sized vertex cover can take hundreds’s of centuries by searching.

Is searching really necessary?

We don’t know.
Suppose we give up our hope of making a polynomial time algorithm for $\text{VERTEX COVER}$ problem and ask if we can make “some thing like poly time algorithm” but nevertheless still a good algorithm!
Suppose we give up our hope of making a polynomial time algorithm for Vertex Cover problem and ask if we can make “some thing like poly time algorithm” but nevertheless still a good algorithm!

For example an algorithm that given a graph on $10^5$ nodes can find a vertex-cover of size 17, if exists, in time $2^{17}10^5$. 
Vertex Cover Continues

- Suppose we give up our hope of making a polynomial time algorithm for VERTEX COVER problem and ask if we can make “some thing like poly time algorithm” but nevertheless still a good algorithm!

- For example an algorithm that given a graph on $10^5$ nodes can find a vertex-cover of size 17, if exists, in time $2^{17}10^5$.

- In other words an algorithm that given a graph on $n$ nodes can find a vertex-cover of size $k$, if exists, in time $2^k n$. 
Vertex Cover Continues

- Suppose we give up our hope of making a polynomial time algorithm for Vertex Cover problem and ask if we can make “some thing like poly time algorithm” but nevertheless still still a good algorithm!
- For example an algorithm that given a graph on $10^5$ nodes can find a vertex-cover of size 17, if exists, in time $2^{17}10^5$.
- In other words an algorithm that given a graph on $n$ nodes can find a vertex-cover of size $k$, if exists, in time $2^k n$.
For example an algorithm that given a graph on $10^5$ nodes can find a vertex-cover of size 17, if exists, in time $2^{17}10^5$. Let us compare these running times.
Vertex Cover Continues

\begin{equation}
\binom{n}{k} \text{ versus } 2^k n
\end{equation}

- For example an algorithm that given a graph on $10^5$ nodes can find a vertex-cover of size 17, if exists, in time $2^{17}10^5$. Let us compare these running times.
Some Comparision

<table>
<thead>
<tr>
<th></th>
<th>$n = 50$</th>
<th>$n = 100$</th>
<th>$n = 150$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k = 2$</td>
<td>625</td>
<td>2,500</td>
<td>5,625</td>
</tr>
<tr>
<td>$k = 3$</td>
<td>15,625</td>
<td>1,25,000</td>
<td>4,21,875</td>
</tr>
<tr>
<td>$k = 5$</td>
<td>30,625</td>
<td>62,50,000</td>
<td>3,16,40,625</td>
</tr>
<tr>
<td>$k = 10$</td>
<td>$10^{12}$</td>
<td>$0.8 \times 10^{14}$</td>
<td>$3.7 \times 10^{16}$</td>
</tr>
<tr>
<td>$k = 20$</td>
<td>$1.8 \times 10^{26}$</td>
<td>$0.5 \times 10^{31}$</td>
<td>$2.1 \times 10^{35}$</td>
</tr>
</tbody>
</table>

**Table:** The ratio $\frac{n^{k+1}}{2^k n}$ for various values of $n$ and $k$. 
What did we do – A message.

- We used the size of a vertex-cover to measure the running time of the algorithm!
What did we do – A message.

- We used the size of a vertex-cover to measure the running time of the algorithm!
- This is not what we do in classical complexity!
In classical complexity, a decision problem is specified by two items of information:

- The input to the problem – a number \( n \).
- The question to be answered.
In classical complexity, a decision problem is specified by two items of information:

- The input to the problem – a number $n$.
- The question to be answered.

In multivariate algorithmics, a decision problem is specified by three items of information:

- The input to the problem – a number $n$.
- The aspect of the input that constitutes the parameter – a number $k$.
- The question to be answered.
Multivariate Algorithmics and a weak magnet!

In multivariate algorithmics, a decision problem is specified by three items of information:

- The input to the problem – a number \( n \).
- The aspect of the input that constitutes the parameter – a number \( k \).
- The question to be answered.

We call an algorithm \( A \), a \textit{fixed parameter tractable (or FPT)} algorithm, for a multivariate problem \( \Pi \), if it runs in time

\[
\textit{f}(k) \cdot n^c.
\]
Vertex Cover problem has an algorithm with running time $2^k n$, say $A$, to check whether a $k$-sized vertex cover exists in a graph on $n$ nodes.
**Vertex Cover** problem has an algorithm with running time $2^k n$, say $A$, to check whether a $k$-sized vertex cover exists in a graph on $n$ nodes. That is, Vertex Cover is FPT.
Branching Algorithm for **VERTEX COVER**

1. Try all subsets of size at most $k - \mathcal{O}(n^k m)$. 
Branching Algorithm for Vertex Cover

1. For every edge \((x, y)\) recursively check whether \(G - x\) or \(G - y\) has a vertex cover of size at most \(k - 1\).
Branching Algorithm for VERTEX COVER

1. For every edge \((x, y)\) recursively check whether \(G - x\) or \(G - y\) has a vertex cover of size at most \(k - 1\).

Basic Idea: Given any edge \((u, v)\) either \(u\) or \(v\) is in the solution.

\[
\langle \emptyset, G \rangle
\]
\[
\langle \{u\}, G - u \rangle
\]
\[
\langle \{v\}, G - v \rangle
\]
\[
\langle \{u, x\}, G' - x \rangle
\]
\[
\langle \{u, y\}, G' - y \rangle
\]

- no. of nodes \(\leq 2^k\)
- time spent at each node \(= O(m)\)
- total time taken \(= O(2^k \cdot m)\)
For decision problems with input size $n$, and a parameter $k$, (which typically is the solution size), the goal here is to design an algorithm with running time $f(k) \cdot n^{O(1)}$, where $f$ is a function of $k$ alone.

Problems that have such an algorithm are said to be fixed parameter tractable (FPT).
**Vertex Cover**

**Input:** A graph $G = (V, E)$ and a positive integer $k$.

**Parameter I:** $k$

**Parameter II:** $n = |V|$

**Question:** Does there exist a subset $V' \subseteq V$ of size at most $k$ such that for every edge $(u, v) \in E$ either $u \in V'$ or $v \in V'$?
Vertex Cover

**Vertex Cover**

**Input:** A graph $G = (V, E)$ and a positive integer $k$.

**Parameter I:** $k$

**Parameter II:** $n = |V|$

**Question:** Does there exist a subset $V' \subseteq V$ of size at most $k$ such that for every edge $(u, v) \in E$ either $u \in V'$ or $v \in V'$?

What about an algorithm for **Vertex Cover** with respect to parameter $n$?
Vertex Cover

**Vertex Cover**

**Input:** A graph $G = (V, E)$ and a positive integer $k$.

**Parameter I:** $k$

**Parameter II:** $n = |V|$

**Question:** Does there exist a subset $V' \subseteq V$ of size at most $k$ such that for every edge $(u, v) \in E$ either $u \in V'$ or $v \in V'$?

What about an algorithm for **Vertex Cover** with respect to parameter $n$?

- Try all possible subsets gives us an algorithm with running time $2^n n^{O(1)}$. 
Branching Algorithm for Vertex Cover

1. For every edge \((x, y)\) recursively check whether \(G - x\) or \(G - N[x]\) has a vertex cover of size at most \(k - |N(x)|\).
Branching Algorithm for **VERTEX COVER**

1. For every edge \((x, y)\) recursively check whether \(G - x\) or \(G - N[x]\) has a vertex cover of size at most \(k - |N(x)|\).
Branching Algorithm for VERTEX COVER

1. For every edge \((x, y)\) recursively check whether \(G - x\) or \(G - N[x]\) has a vertex cover of size at most \(k - |N(x)|\). For any edge \((u, v)\) either \(u\) is in or all its neighbors are in the solution.

\[
\begin{align*}
&\langle \emptyset, G \rangle \\
&\langle \{u\}, G - u \rangle \\
&\langle \{N(v)\}, G - N[v] \rangle
\end{align*}
\]

- \(T(n) \leq T(n - 1) + T(n - 2)\)
- no. of nodes \(\leq 1.618^n\)
- time spent at each node \(= O(m)\)
- total time taken \(= 1.618^n\)
• **Vertex Cover** admits an algorithm with running time $1.1996^n$. 
**Vertex Cover** admits an algorithm with running time $1.1996^n$.

For this talk we use $n$ to denote the size of the vertex set or the number of variables in a cnf-sat formula of an input to the SAT problem.
Vertex Cover admits an algorithm with running time $1.1996^n$.

For this talk we use $n$ to denote the size of the vertex set or the number of variables in a cnf-sat formula of an input to the SAT problem.

This parameterization will be called exact exponential time algorithms or exact algorithms.
Weighted 3-SAT

3-SAT

Input: A cnf-formula $\varphi$ with $n$ variables and $m$ clauses.
Question: Is $\varphi$ satisfiable?
Weighted 3-SAT

3-SAT

Input: A cnf-formula $\varphi$ with $n$ variables and $m$ clauses.

Question: Is $\varphi$ satisfiable?

#1’s in a satisfying assignment is called its weight.
Weighted 3-SAT

3-Sat
Input: A cnf-formula $\varphi$ with $n$ variables and $m$ clauses.
Question: Is $\varphi$ satisfiable?

#1’s in a satisfying assignment is called its weight.

Weighted 3-Sat
Input: A cnf-formula $\varphi$ with $n$ variables and $m$ clauses.
Parameter: $k$
Question: Does there exists weight $k$-satisfying assignment?
Weighted 3-SAT

3-Sat
Input: A cnf-formula $\varphi$ with $n$ variables and $m$ clauses.
Question: Is $\varphi$ satisfiable?

#1's in a satisfying assignment is called its weight.

Weighted 3-Sat
Input: A cnf-formula $\varphi$ with $n$ variables and $m$ clauses.
Parameter I: $k$
Parameter II: $n$
Question: Does there exists weight $k$-satisfying assignment?
Weighted 3-SAT

3-Sat
Input: A cnf-formula $\varphi$ with $n$ variables and $m$ clauses.
Question: Is $\varphi$ satisfiable?

#1’s in a satisfying assignment is called its weight.

Weighted 3-Sat
Input: A cnf-formula $\varphi$ with $n$ variables and $m$ clauses.
Parameter I: $k$
Parameter II: $n$
Question: Does there exists weight $k$-satisfying assignment?

3-Sat and Weighted 3-Sat are polynomially equivalent.
Weighted 3-SAT

**Input:** A cnf-formula $\varphi$ with $n$ variables and $m$ clauses.

**Parameter I:** $k$

**Parameter II:** $n$

**Question:** Does there exists weight $k$-satisfying assignment?
Weighted 3-SAT

Input: A cnf-formula $\varphi$ with $n$ variables and $m$ clauses.
Parameter I: $k$
Parameter II: $n$
Question: Does there exists weight $k$-satisfying assignment?

1. Check if all $\bar{0}$ is a satisfying assignment.
Weighted 3-SAT

**Weighted 3-Sat**

**Input:** A cnf-formula $\varphi$ with $n$ variables and $m$ clauses.

**Parameter I:** $k$

**Parameter II:** $n$

**Question:** Does there exist weight $k$-satisfying assignment?

1. Check if all $\bar{0}$ is a *satisfying assignment*.
2. Else, we know that there exists a *clause* $C$ that has only positive literals – positive clause.
**Weighted 3-SAT**

**Input:** A cnf-formula \( \varphi \) with \( n \) variables and \( m \) clauses.

**Parameter I:** \( k \)

**Parameter II:** \( n \)

**Question:** Does there exists weight \( k \)-satisfying assignment?

1. Check if all \( \overline{0} \) is a *satisfying assignment*.
2. Else, we know that there exists a *clause* \( C \) that has only positive literals – positive clause.
3. Branch on a positive clause \( C = (x, y, z) \).
Weighted 3-SAT

\[ \langle \tau = \overline{0}, \phi, C, k \rangle \]

\[ \langle \tau_x, \phi, C, k - 1 \rangle \]

\[ x = 1 \]

\[ \langle \tau_y, \phi, C, k - 1 \rangle \]

\[ y = 1 \]

\[ \langle \tau_z, \phi, C, k - 1 \rangle \]

\[ z = 1 \]

\[ \tau_x = \tau \text{ with } x \text{ changed to 1} \]

\[ T(k) \leq 3T(k - 1) \]

\[ 3^k n^{\mathcal{O}(1)} \]
**3-Hitting Set**

**Input:** A universe $U$ with $n$ elements and a family $F$ of sets, of size at most 3, over $U$ of size $m$.

**Parameter I:** $k$

**Parameter II:** $n$

**Question:** Does there a set $S \subseteq U$ of size at most $k$ such that for every $F \in F$, $F \cap S \neq \emptyset$?
3-Hitting Set

3-Hitting Set
Input: A universe $U$ with $n$ elements and a family $\mathcal{F}$ of sets, of size at most $3$, over $U$ of size $m$.
Parameter I: $k$
Parameter II: $n$
Question: Does there a set $S \subseteq U$ of size at most $k$ such that for every $F \in \mathcal{F}$, $F \cap S \neq \emptyset$?

- $S$ is called 3-hitting set/hitting set.
3-Hitting Set

**Input:** A universe $U$ with $n$ elements and a family $\mathcal{F}$ of sets, of size at most 3, over $U$ of size $m$.

**Parameter I:** $k$

**Parameter II:** $n$

**Question:** Does there a set $S \subseteq U$ of size at most $k$ such that for every $F \in \mathcal{F}$, $F \cap S \neq \emptyset$?

- $S$ is called 3-hitting set/hitting set.
- Start with $S = \emptyset$. 

3-Hitting Set

Input: A universe $U$ with $n$ elements and a family $\mathcal{F}$ of sets, of size at most 3, over $U$ of size $m$.

Parameter I: $k$

Parameter II: $n$

Question: Does there a set $S \subseteq U$ of size at most $k$ such that for every $F \in \mathcal{F}$, $F \cap S \neq \emptyset$?

- $S$ is called 3-hitting set/hitting set.
- Start with $S = \emptyset$.
- If $S$ is not a hitting set then Branch on a set $F = \{x, y, z\} \in \mathcal{F}$ that does not intersect $S$!
3-Hitting Set

**Input:** A universe $U$ with $n$ elements and a family $\mathcal{F}$ of sets, of size at most 3, over $U$ of size $m$.

**Parameter I:** $k$

**Parameter II:** $n$

**Question:** Does there a set $S \subseteq U$ of size at most $k$ such that for every $F \in \mathcal{F}$, $F \cap S \neq \emptyset$?

- $S$ is called 3-hitting set/hitting set.
- Start with $S = \emptyset$.
- If $S$ is not a hitting set then Branch on a set $F = \{x, y, z\} \in \mathcal{F}$ that does not intersect $S$!
- The algorithm runs in time $3^k(n + m)^O(1)$.
Books in the area
Basic Questions
The main actors of our next part will be:
The main actors of our next part will be:

- **Weighted $c$-Sat** – 3 sized clause replaced by $c$ sized clause – $c^k(n + m)^O(1)$.
The main actors of our next part will be:

- **Weighted $c$-Sat** – 3 sized clause replaced by $c$ sized clause – $c^k(n + m)^\mathcal{O}(1)$.

- **$c$-Hitting Set** – 3 sized set replaced by $c$ sized set – $c^k(n + m)^\mathcal{O}(1)$.

- Will forget *polynomial factor* from the running time of the parameterized algorithm during the talk.
Subset Optimization Problems

- An *implicit set system* as a function $\Phi$ that takes as input a string $I \in \{0, 1\}^*$ and outputs a set system $(U_I, \mathcal{F}_I)$, where $U_I$ is a universe and $\mathcal{F}_I$ is a collection of subsets of $U_I$. 
An implicit set system as a function $\Phi$ that takes as input a string $I \in \{0, 1\}^*$ and outputs a set system $(U_I, \mathcal{F}_I)$, where $U_I$ is a universe and $\mathcal{F}_I$ is a collection of subsets of $U_I$.

$U_I$ = \{subsets of $U_I$\}
Subset Optimization Problems

Φ-SUBSET

Input: An instance $I$

Question: A set $S \in \mathcal{F}_I$ if one exists.
Subset Optimization Problems

Φ-

Input: An instance $I$

Question: A set $S \in \mathcal{F}_I$ if one exists.

Input $= I$

$\mathcal{F}_I = \{\text{subsets of } U_I\}$
Examples of Subset Optimization Problems

- $\text{SAT}$ or $c$-$\text{SAT}$: $\varphi$ is a formula.
  $U_\varphi = V$ (set of variables)

$$F_\varphi = \{X \mid X \subseteq V \text{ and setting } X \text{ to } 1 \text{ and } V \setminus X \text{ to } 0 \text{ yields a satisfying assignment}\}$$
Examples of Subset Optimization Problems

- **SAT** or **c-SAT**: $\varphi$ is a formula.
  $U_{\varphi} = V$ (set of variables)
  \[
  F_{\varphi} = \{ X \mid X \subseteq V \text{ and setting } X \text{ to } 1 \text{ and } V \setminus X \text{ to } 0 \text{ yields a satisfying assignment} \}
  \]

- **c-Hitting Set**: $I = (U, F, k)$
  $U_I = U$
  \[
  F_I = \{ X \mid X \subseteq V, |X| \leq k \text{ and } X \text{ is a hitting set} \}
  \]
Examples of Subset Optimization Problems

- **Weighted $c$-Sat**: $I = (\varphi, k)$, where $\varphi$ is a formula.
  $U_I = V$ (set of variables of $\varphi$)

  $$\mathcal{F}_I = \{ X \mid X \subseteq V, |X| \leq k \text{ and setting } X \text{ to } 1 \text{ and } V\setminus X \text{ to } 0 \text{ yields a satisfying assignment} \}$$
Examples of Subset Optimization Problems

- **Weighted $c$-Sat**: $I = (\phi, k)$, where $\phi$ is a formula.
  $U_I = V$ (set of variables of $\phi$)

  $$\mathcal{F}_I = \{ X \mid X \subseteq V, |X| \leq k \text{ and setting } X \text{ to } 1 \text{ and } V\setminus X \text{ to } 0 \text{ yields a satisfying assignment} \}$$

- **Feedback Vertex Set**: $I = (G, k)$ (does $G$ have a set of size at most $k$ such that $G\setminus S$ is acyclic)
  $U_I = V(G)$

  $$\mathcal{F}_I = \{ X \mid X \subseteq V(G), |X| \leq k \text{ and } G\setminus X \text{ is acyclic} \}$$
Two Basic Questions?

Q1

FPT

EXACT ALGO

Q2

SAT

SUBSET OPTIMIZATION PROBLEMS
What is known for the first question?

**Proposition**

If a subset optimization problem is solvable in time $c^k$, then there exists an exact algorithm running in time
What is known for the first question?

**Proposition**

If a subset optimization problem is solvable in time $c^k$, then there exists an exact algorithm running in time

$$\max_{k \leq n} \min \left\{ c^k, \binom{n}{k} \right\}$$
What is known for the first question?

**Proposition**

If a subset optimization problem is solvable in time $c^k$, then there exists an exact algorithm running in time

$$\max_{k \leq n} \min \left\{ c^k, \binom{n}{k} \right\}$$

- For any fixed $\varepsilon > 0$, if $k \notin \left( \frac{n}{2} - \varepsilon n, \frac{n}{2} + \varepsilon n \right)$, then $\binom{n}{k} \leq (2 - \alpha)^n$ for some fixed $\alpha > 0$. 
What is known for the first question?

**Proposition**

If a subset optimization problem is solvable in time $c^k$, then there exists an exact algorithm running in time

$$\max_{k \leq n} \min \left\{ c^k, \binom{n}{k} \right\}$$

- For any fixed $\varepsilon > 0$, if $k \notin \left( \frac{n}{2} - \varepsilon n, \frac{n}{2} + \varepsilon n \right)$, then $\binom{n}{k} \leq (2 - \alpha)^n$ for some fixed $\alpha > 0$.
- So if $c \leq (4 - \varepsilon)^k$ then we get $(2 - \varepsilon')^n$ algorithm!
What is known for the first question?

**Proposition**

If a subset optimization problem is solvable in time $c^k$, then there exists an exact algorithm running in time

$$\max_{k \leq n} \min \left\{ c^k, \binom{n}{k} \right\}$$

- For any fixed $\varepsilon > 0$, if $k \notin \left( \frac{n}{2} - \varepsilon n, \frac{n}{2} + \varepsilon n \right)$, then $\binom{n}{k} \leq (2 - \alpha)^n$ for some fixed $\alpha > 0$.
- So if $c \leq (4 - \varepsilon)^k$ then we get $(2 - \varepsilon')^n$ algorithm!
- **Example:** Using the $3^k$ algorithm for Weighted 3-SAT, we get $1.95^n$ algorithm for 3-SAT!
Proposition

If a subset optimization problem is solvable in time $c^k$, $c < 4$, then there exists an exact algorithm running in time

$$\max_{k \leq n} \min \left\{ c^k, \binom{n}{k} \right\} \leq (2 - \varepsilon')^n$$
Proposition

If a subset optimization problem is solvable in time $c^k$, $c < 4$, then there exists an exact algorithm running in time

$$\max_{k \leq n} \min \left\{ c^k, \binom{n}{k} \right\} \leq (2 - \varepsilon')^n$$

A rather unsatisfactory answer!
Proposition

If a subset optimization problem is solvable in time $c^k$, $c < 4$, then there exists an exact algorithm running in time

$$\max_{k\leq n} \min \left\{c^k, \binom{n}{k} \right\} \leq (2 - \varepsilon')^n$$

A rather unsatisfactory answer!

Proposition

If a subset optimization problem is solvable in time $c^k$, for some fixed $c$, then there exists an exact algorithm running in time $(2 - f(c))^n$. 
Moving to the second question.

**SAT**
- Random Sampling and doing local search.
Moving to the second question.

**Sat**

- Random Sampling and doing local search. Sampling an assignment $\sigma$ and doing local search around it.
Moving to the second question.

**Sat**

- Random Sampling and doing local search. Sampling an assignment $\sigma$ and doing local search around it.
- Random decision and inference
Moving to the second question.

**Sat**

- Random Sampling and doing local search. Sampling an assignment $\sigma$ and doing local search around it.
- Random decision and inference

**Other Subset Optimization Problems**

- Branching!
Moving to the second question.

**SAT**
- Random Sampling and doing local search. Sampling an assignment $\sigma$ and doing local search around it.
- Random decision and inference

**Other Subset Optimization Problems**
- Branching!
  - Pick up some element $x$ from the universe cleverly and then decide whether it is inside the solution or not!
Moving to the second question.

**SAT**
- Random Sampling and doing local search. Sampling an assignment $\sigma$ and doing local search around it.
- Random decision and inference

**Other Subset Optimization Problems**
- Branching!
  - Pick up some element $x$ from the universe cleverly and then decide whether it is inside the solution or not!
- Even for **SAT** branching gave the best algorithm initially.
Moving to the second question.

**Sat**
- Random Sampling and doing local search. Sampling an assignment $\sigma$ and doing local search around it.
- Random decision and inference

**Other Subset Optimization Problems**
- Branching!
  Pick up some element $x$ from the universe *cleverly* and then decide whether it is inside the solution or not!
- Even for *Sat* branching gave the best algorithm initially.
- **GOAL:** Transfer ideas from *Sat* to other problems.
Towards Goal

Focus

Random Sampling + Local Search for SAT
Two Gems in Schöning’s algorithm for $c$-Sat: IDEA I

Theorem
If Local Search $c$-Sat can be solved in time $\alpha^\ell$ then $c$-Sat can be solved in time
Two Gems in Schöning’s algorithm for $c$-SAT: IDEA I

**Theorem**

If **Local Search** $c$-Sat can be solved in time $\alpha^\ell$ then $c$-Sat can be solved in time

$$\left(2 - \frac{2}{\alpha + 1}\right)^n$$
Two Gems in Schöning’s algorithm for $c$-SAT: IDEA I

Theorem

If Local Search $c$-Sat can be solved in time $\alpha^\ell$ then $c$-Sat can be solved in time

$$\left(2 - \frac{2}{\alpha + 1}\right)^n$$

- What is the local search?
- What does the Schöning’s algorithm do?
Local Search
Local Search \( c\)-Sat

**Local Search \( c\)-Sat (LS \( c\)-SAT)**

**Input:** A \( c\)-cnf-formula \( \varphi \), an assignment \( \sigma \) and a positive integer \( \ell \).

**Parameter:** \( \ell \)

**Question:** Does there a satisfying assignment \( \sigma' \) such that \( d(\sigma, \sigma') \leq \ell \)?
Local Search $c$-Sat

\begin{itemize}
  \item $d(\sigma, \sigma')$ counts the number of places $\sigma$ and $\sigma'$ differs.
  \item Basically given $\sigma$ (may not be satisfying) can we flip at most $\ell$ variables in $\sigma$ and obtain a satisfying assignment.
\end{itemize}

**LOCAL SEARCH $c$-SAT (LS $c$-SAT)**

**Input:** A $c$-cnf-formula $\varphi$, an assignment $\sigma$ and a positive integer $\ell$.

**Parameter:** $\ell$

**Question:** Does there a satisfying assignment $\sigma'$ such that $d(\sigma, \sigma') \leq \ell$?
Algorithm for Local Search $c$-Sat

Same as WEIGHTED $c$-SAT!
Algorithm for Local Search \( c \)-Sat

**Same as WEIGHTED \( c \)-SAT!**

**Algorithm**

Pick an unsatisfied clause \( C \), guess the variable in \( C \) that needs to be flipped and flip!
Algorithm for Local Search $c$-Sat

Same as WEIGHTED $c$-SAT!

**Algorithm**

Pick an unsatisfied clause $C$, guess the variable in $C$ that needs to be flipped and flip!

- Gives us $c^\ell$ algorithm for LS $c$-SAT.
Algorithm for Local Search \( c \)-Sat

**Same as WEIGHTED \( c \)-SAT!**

**Algorithm**

Pick an unsatisfied clause \( C \), guess the variable in \( C \) that needs to be flipped and flip!

- Gives us \( c^\ell \) algorithm for LS \( c \)-SAT.
- Gives us

\[
\left( 2 - \frac{2}{3+1} \right)^n = 1.5^n
\]

algorithm for 3-SAT
Two Gems in Schöning’s algorithm for \(c\text{-SAT: IDEA II}\)

**Theorem**

If Permissive Local Search \(c\text{-Sat}\) can be solved in time \(\alpha^\ell\) then \(c\text{-Sat}\) can be solved in time

\[
\left(2 - \frac{2}{\alpha + 1}\right)^n
\]
Two Gems in Schöning’s algorithm for $c$-Sat: Idea II

**Theorem**

If Permissive Local Search $c$-Sat can be solved in time $\alpha^\ell$ then $c$-Sat can be solved in time

$$\left(2 - \frac{2}{\alpha + 1}\right)^n$$

- What is permissive local search?
Permissive Local Search $c$-Sat

**Permissive Local Search $c$-Sat (PLS $c$-Sat)**

**Input:** A $c$-cnf-formula $\varphi$, an assignment $\sigma$ and a positive integer $\ell$.

**Parameter:** $\ell$

**Question:** If there exists a satisfying assignment $\sigma'$ such that $d(\sigma, \sigma') \leq \ell$ then output some satisfying assignment $\rho$. 

Differences between LS $c$-Sat and PLS $c$-Sat

Assume there exists a satisfying assignment $\sigma_1$ such that $d_p(\sigma, \sigma_1) \leq \ell$, then

In LS $c$-Sat we always need to output $\sigma_1$ such that $d(\sigma, \sigma_1) \leq \ell$. In PLS $c$-Sat output any satisfying assignment – even outside the ball.

If no such assignment, then LS $c$-Sat always outputs NO, but PLS $c$-Sat could output a satisfying assignment.
Permissive Local Search $c$-Sat

**Permissive Local Search $c$-Sat (PLS $c$-Sat)**

**Input:** A $c$-cnf-formula $\varphi$, an assignment $\sigma$ and a positive integer $\ell$.

**Parameter:** $\ell$

**Question:** If there exists a satisfying assignment $\sigma'$ such that $d(\sigma, \sigma') \leq \ell$ then output some satisfying assignment $\rho$.

Differences between **LS $c$-Sat** and **PLS $c$-Sat**

- Assume there exists a satisfying assignment $\sigma'$ such that $d(\sigma, \sigma') \leq \ell$, then
  - In **LS $c$-Sat** we always need to output $\sigma'$ such that $d(\sigma, \sigma') \leq \ell$. In **PLS $c$-Sat** output any satisfying assignment – even outside the ball.
Permissive Local Search $c$-Sat

**Permissive Local Search $c$-Sat (PLS $c$-Sat)**

**Input:** A $c$-cnf-formula $\varphi$, an assignment $\sigma$ and a positive integer $\ell$.  
**Parameter:** $\ell$ 
**Question:** If there exists a satisfying assignment $\sigma'$ such that $d(\sigma, \sigma') \leq \ell$ then output some satisfying assignment $\rho$.

Differences between **LS $c$-Sat** and **PLS $c$-Sat**

- Assume there exists a satisfying assignment $\sigma'$ such that $d(\sigma, \sigma') \leq \ell$, then
  - In **LS $c$-Sat** we always need to output $\sigma'$ such that $d(\sigma, \sigma') \leq \ell$. In **PLS $c$-Sat** output any satisfying assignment – even outside the ball.
- If no such assignment, then **LS $c$-Sat** always outputs NO, but **PLS $c$-Sat** could output a satisfying assignment.
Algorithm for Permissive Local Search

c-Sat

- Schöning’s gave $(c - 1)^\ell$ algorithm for PLS $c$-SAT.
Schöning’s gave \((c - 1)^\ell\) algorithm for PLS \(c\text{-SAT}\).

Gives us

\[
\left(2 - \frac{2}{2 + 1}\right)^n = \left(\frac{4}{3}\right)^n = 1.333..^n
\]

algorithm for \(3\text{-SAT}\)
Schöning Algorithm

**Theorem**

If (Permissive) Local Search $c$-Sat can be solved in time $\alpha^\ell$ then $c$-Sat can be solved in time

$$\left(2 - \frac{2}{\alpha + 1}\right)^n$$

**Question:** How does this reduction from local search to solving works?
Schöning Algorithm

**Theorem**

If (Permissive) Local Search $c$-Sat can be solved in time $\alpha^\ell$ then $c$-Sat can be solved in time

$$
\left(2 - \frac{2}{\alpha + 1}\right)^n
$$

**Question:** How does this reduction from local search to solving works?

- Based on the universe size $n$, choose an integer $\ell$ (solely based on $n$).
Schöning Algorithm

**Theorem**

If (Permissive) Local Search $c$-Sat can be solved in time $\alpha^{\ell}$ then $c$-Sat can be solved in time

$$\left(2 - \frac{2}{\alpha + 1}\right)^n$$

**Question:** How does this reduction from local search to solving works?

- Based on the universe size $n$, choose an integer $\ell$ (solely based on $n$).
- Pick a random assignment $\sigma$ and do permissive local search with the parameter $\ell$. 

Schöning Algorithm

Theorem

If (Permissive) Local Search $c$-Sat can be solved in time $\alpha^\ell$ then $c$-Sat can be solved in time

$$\left(2 - \frac{2}{\alpha + 1}\right)^n$$

Question: How does this reduction from local search to solving works?

- Based on the universe size $n$, choose an integer $\ell$ (solely based on $n$).
- Pick a random assignment $\sigma$ and do permissive local search with the parameter $\ell$.
- Bigger the value of $\ell$ –
Schöning Algorithm

**Theorem**

If (Permissive) Local Search $c$-Sat can be solved in time $\alpha^\ell$ then $c$-Sat can be solved in time

$$\left(2 - \frac{2}{\alpha + 1}\right)^n$$

**Question:** How does this reduction from local search to solving works?

- Based on the universe size $n$, choose an integer $\ell$ (solely based on $n$).
- Pick a random assignment $\sigma$ and do permissive local search with the parameter $\ell$.
- Bigger the value of $\ell$ – better success probability of ending at closer to a solution –
Schöning Algorithm

**Theorem**

If (Permissive) Local Search $c$-Sat can be solved in time $\alpha^\ell$ then $c$-Sat can be solved in time

$$\left(2 - \frac{2}{\alpha + 1}\right)^n$$

**Question:** How does this reduction from local search to solving works?

- Based on the universe size $n$, choose an integer $\ell$ (solely based on $n$).
- Pick a random assignment $\sigma$ and do permissive local search with the parameter $\ell$.
- Bigger the value of $\ell$ – *better success probability of ending at closer to a solution* – more time you take to find the solution.
What is nice about the reduction?

Theorem

If Local Search $c$-Sat can be solved in time $\alpha^\ell$ then $c$-Sat can be solved in time $\left(2 - \frac{2}{\alpha+1}\right)^n$.

The reduction has nothing to do with $c$-SAT problem!
What is nice about the reduction?

**Theorem**

If Local Search $c$-Sat can be solved in time $\alpha^\ell$ then $c$-Sat can be solved in time $\left(2 - \frac{2}{\alpha+1}\right)^n$.

The reduction has nothing to do with $c$-SAT problem!

Let $\Pi$ be a subset optimization problem.

**Theorem**

If Local Search $\Pi$ can be solved in time $\alpha^\ell$ then $\Pi$ can be solved in time

$$\left(2 - \frac{2}{\alpha+1}\right)^n$$
## Local Search for $\Pi$

<table>
<thead>
<tr>
<th>Local Search $\Pi$ (LS $\Pi$)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> An instance $I$, a set $X \subseteq U_I$, and an integer $\ell$.</td>
</tr>
<tr>
<td><strong>Parameter:</strong> $\ell$</td>
</tr>
<tr>
<td><strong>Question:</strong> Does there exists a subset $S \subseteq U_I$ such that $S \Delta X$ is a solution?</td>
</tr>
</tbody>
</table>
Local Search for $\Pi$

**Local Search $\Pi$ (LS $\Pi$)**

**Input:** An instance $I$, a set $X \subseteq U_I$, and an integer $\ell$.

**Parameter:** $\ell$

**Question:** Does there exists a subset $S \subseteq U_I$ such that $S \Delta X$ is a solution?

- Basically remove some element and add some element and get a solution.

Let us try this for $c$-Hitting Set
3-Hitting Set

**Input:** A universe $U$ with $n$ elements and a family $\mathcal{F}$ of sets, of size at most 3, over $U$ of size $m$.

**Parameter:** $k$

**Question:** Does there a set $S \subseteq U$ of size at most $k$ such that for every $F \in \mathcal{F}$, $F \cap S \neq \emptyset$?
### 3-Hitting Set

**Input:** A universe $U$ with $n$ elements and a family $\mathcal{F}$ of sets, of size at most 3, over $U$ of size $m$.  
**Parameter:** $k$  
**Question:** Does there exist a set $S \subseteq U$ of size at most $k$ such that for every $F \in \mathcal{F}$, $F \cap S \neq \emptyset$?

### Local Search 3-Hitting Set

**Input:** An instance $(U, \mathcal{F}, k)$, a set $X \subseteq U$, and an integer $\ell$.  
**Parameter:** $\ell$  
**Question:** Does there exist a subset $S \subseteq U$ such that $S \Delta X$ is a solution of size at most $k$?
**3-Hitting Set**

**Local Search 3-Hitting Set**

**Input:** An instance \((U, \mathcal{F}, k)\), a set \(X \subseteq U\), and an integer \(\ell\).

**Parameter:** \(\ell\)

**Question:** Does there exists a subset \(S \subseteq U\) such that \(S \Delta X\) is a solution of size at most \(k\)?

- If there is a set \(F\) that is *not hit* by \(X\), find it and branch as before.
3-Hitting Set

**Local Search 3-Hitting Set**

**Input:** An instance $(U, \mathcal{F}, k)$, a set $X \subseteq U$, and an integer $\ell$.

**Parameter:** $\ell$

**Question:** Does there exists a subset $S \subseteq U$ such that $S \triangle X$ is a solution of size at most $k$?

- If there is a set $F$ that is *not hit* by $X$, find it and branch as before.

- If $X$ is hitting set and $|X| > k$ (for example $X$ could be $U$) – what should we do?
Local Search 3-Hitting Set

**Input:** An instance \((U, \mathcal{F}, k)\), a set \(X \subseteq U\), and an integer \(\ell\).

**Parameter:** \(\ell\)

**Question:** Does there exist a subset \(S \subseteq U\) such that \(S \triangle X\) is a solution of size at most \(k\)?

- If there is a set \(F\) that is not hit by \(X\), find it and branch as before.

- If \(X\) is hitting set and \(|X| > k\) (for example \(X\) could be \(U\)) – what should we do?
  
  Nothing!

Local Search 2-Hitting Set is W[1]-hard.
But adding elements work!
But adding elements work!
So let us just do that.
Monotone Local Search
New Local Search

- Pick a random solution $S$.
- Now do a local search around $S$ that only adds elements – that is only goes upwards – \textit{a monotone local search}. 
New Local Search

- Pick a random solution $S$.
- Now do a local search around $S$ that only adds elements – that is only goes upwards – a monotone local search.
- Probability of success might not be as good as Schöning but local search algorithm of this nature could be efficient.
Extension Problem

**Φ-Extension**

**Input:** An instance $I$, a set $X \subseteq U_I$, and an integer $k$.

**Parameter:** $k$

**Question:** Does there exist a subset $S \subseteq (U_I \setminus X)$ such that $S \cup X \in \mathcal{F}_I$ and $|S| \leq k$?

If $\Phi$ is c-Hitting Set then it has extension algorithm with running time $c_k$. Find an a set that is not hit by $X$ and branch in $c$ ways!
**Extension Problem**

**Φ-Extension**

**Input:** An instance $I$, a set $X \subseteq U_I$, and an integer $k$.

**Parameter:** $k$

**Question:** Does there exists a subset $S \subseteq (U_I \setminus X)$ such that $S \cup X \in \mathcal{F}_I$ and $|S| \leq k$?

- If $Φ$ is $c$-Hitting Set then it has extension algorithm with running time $c^k$. 
Extension Problem

Φ-Extension

**Input:** An instance $I$, a set $X \subseteq U_I$, and an integer $k$.

**Parameter:** $k$

**Question:** Does there exist a subset $S \subseteq (U_I \setminus X)$ such that $S \cup X \in \mathcal{F}_I$ and $|S| \leq k$?

- If $Φ$ is $c$-Hitting Set then it has extension algorithm with running time $c^k$.
- Find an a set that is not hit by $X$ and branch in $c$ ways!
Our Algorithm
Algorithm

1. Choose an integer $t \leq k$ depending on $c$, $n$ and $k$. 
Choose an integer $t \leq k$ depending on $c$, $n$ and $k$.
Select a random subset $X$ of $U_I$ of size $t$. 
Choose an integer $t \leq k$ depending on $c$, $n$ and $k$.
Select a random subset $X$ of $U_I$ of size $t$.
Do extension from $X$ with $\ell = k - t$. 
Running Time one Round

1. Choose an integer $t \leq k$ depending on $c$, $n$ and $k$.
2. Select a random subset $X$ of $U_I$ of size $t$.
3. Do extension from $X$ with $\ell = k - t$.

The running time of the algorithm for one round is:

$$c^{k-t} = c^\ell$$
Running Time one Round

1. Choose an integer $t \leq k$ depending on $c$, $n$ and $k$.
2. Select a random subset $X$ of $U_I$ of size $t$.
3. Do extension from $X$ with $\ell = k - t$.

When does this algorithm succeed:

$$|S| \leq k$$
Running Time one Round

1. Choose an integer $t \leq k$ depending on $c$, $n$ and $k$.
2. Select a random subset $X$ of $U_I$ of size $t$.
3. Do extension from $X$ with $\ell = k - t$.

When does this algorithm succeed:

$$|S| \leq k$$

$$\Pr[\text{Success} = (X \subseteq S')] = \frac{k}{n} \binom{k}{t} \binom{n}{t}$$
For constant probability of success ...

\[
\frac{\binom{n}{t}}{\binom{k}{t}} \cdot c^{k-t}
\]
For constant probability of success ... 

\[
\min_{t \leq k} \left\{ \binom{n}{t} \frac{(n)}{(t)} \right\} \cdot c^{k-t}
\]
For constant probability of success ...

\[
\max_{k \leq n} \min_{t \leq k} \left\{ \binom{n}{t} \binom{k}{t} \right\} \cdot c^{k-t}
\]
For constant probability of success ...

$$\max_{k \leq n} \min_{t \leq k} \left\{ \frac{\binom{n}{t}}{\binom{k}{t}} \right\} \cdot c^{k-t}$$

High school calculus shows this to be:

$$\left(2 - \frac{1}{c}\right)^n$$
An algorithm for 3-HITTING SET with running time:

\[
\left(2 - \frac{1}{3}\right)^n = 1.6666...^n
\]
Looking again at Extension Problem

Φ-EXTENSION

Input: An instance $I$, a set $X \subseteq U_I$, and an integer $k$.
Parameter: $k$
Question: Does there exists a subset $S \subseteq (U_I \setminus X)$ such that $S \cup X \in \mathcal{F}_I$ and $|S| \leq k$?
Looking again at Extension Problem

**Φ-EXTENSION**

**Input:** An instance $I$, a set $X \subseteq U_I$, and an integer $k$.

**Parameter:** $k$

**Question:** Does there exists a subset $S \subseteq (U_I \setminus X)$ such that $S \cup X \in F_I$ and $|S| \leq k$?

- Let $Φ$ is $c$-Hitting Set then it has extension algorithm with running time $c^k$. 
Looking again at Extension Problem

Φ-Extension

**Input:** An instance $I$, a set $X \subseteq U_I$, and an integer $k$.

**Parameter:** $k$

**Question:** Does there exists a subset $S \subseteq (U_I \setminus X)$ such that $S \cup X \in \mathcal{F}_I$ and $|S| \leq k$?

- Let $\Phi$ is **c-Hitting Set** then it has extension algorithm with running time $c^k$.
- **New Algorithm:** Delete all the sets from $\mathcal{F}$ that is hit from $X$ and obtain an instance $(U, \mathcal{F}', k - t)$ of c-Hitting Set!
Looking again at Extension Problem

$\Phi$-EXTENSION

**Input:** An instance $I$, a set $X \subseteq U_I$, and an integer $k$.

**Parameter:** $k$

**Question:** Does there exist a subset $S \subseteq (U_I \setminus X)$ such that $S \cup X \in \mathcal{F}_I$ and $|S| \leq k$?

- Let $\Phi$ is $c$-HITTING SET then it has extension algorithm with running time $c^k$.
- **New Algorithm:** Delete all the sets from $\mathcal{F}$ that is hit from $X$ and obtain an instance $(U, \mathcal{F}', k - t)$ of $c$-HITTING SET!
- Extension problem is basically the parameterized problem!
Extension to Solving

- Extension problem *reduces to solving* the parameterized problem.
Extension to Solving

- Extension problem *reduces to solving* the parameterized problem.
- **3-Hitting Set** has an algorithm with running time $(2.0755)^k$.
Extension to Solving

- Extension problem *reduces to solving* the parameterized problem.

- **3-Hitting Set** has an algorithm with running time \((2.0755)^k\).

\[
\left(2 - \frac{1}{2.0755}\right)^n = 1.5182^n
\]
Extension to Solving

- Extension problem *reduces to solving* the parameterized problem.

- **3-Hitting Set** has an algorithm with running time $(2.0755)^k$.

\[
\left(2 - \frac{1}{2.0755}\right)^n = 1.5182^n
\]

- This holds for many parameterized problems and if they have $c^k$ algorithm we immediately get $(2 - \frac{1}{c})^n$ exact algorithm.
Convincing you that ...

\[
\max_{k \leq n} \min_{t \leq k} \left\{ \frac{\binom{n}{t}}{\binom{k}{t}} \right\} \cdot c^{k-t}
\]

\[
\leq \left( 2 - \frac{1}{c} \right)^n
\]
An Algorithm for Subset Optimization Problem

Question is essentially:

Input: $n$, $k$

Question: Find a solution of size at most $k$ if exists?
An Algorithm for Subset Optimization Problem

Question is essentially:

Input: $n, k$

Question: Find a solution of size at most $k$ if exists?

A deterministic algorithm – try all subsets of size at most $k$:

\[
\binom{n}{k}
\]
An Algorithm for Subset Optimization Problem

**Question is essentially:**

**Input:** \( n, k \)

**Question:** Find a solution of size at most \( k \) if exists?

A randomized algorithm – pick a random subset of size \( k \), the success probability is:

\[
\frac{1}{\binom{n}{k}}
\]
Analysis

A randomized algorithm that picks a random subset of size $k$ by picking one vertex at a time and inserting it into the solution.

\[
\frac{k}{n} \cdot \frac{k-1}{n-1} \cdot \frac{k-2}{n-2} \cdot \ldots \cdot \frac{2}{n-(k-2)} \cdot \frac{1}{n-(k-1)} = \binom{n}{k}
\]
A randomized algorithm that picks a random subset of size $k$ by picking one vertex at a time and inserting it into the solution.

$$\frac{k}{n} \cdot \frac{k-1}{n-1} \cdot \frac{k-2}{n-2} \cdots \frac{2}{n-(k-2)} \cdot \frac{1}{n-(k-1)} = \binom{n}{k}$$

- In the beginning of the random process the success probability of each step is high.
A randomized algorithm that picks a random subset of size $k$ by picking one vertex at a time and inserting it into the solution.

\[
\frac{k}{n} \cdot \frac{k - 1}{n - 1} \cdot \frac{k - 2}{n - 2} \cdots \frac{2}{n - (k - 2)} \cdot \frac{1}{n - (k - 1)} = \frac{1}{\binom{n}{k}}
\]

- In the beginning of the random process the success probability of each step is high.
- It gets progressively worse, and in the very end it is close to $1/n$. 
A randomized algorithm that picks a random subset of size $k$ by picking one vertex at a time and inserting it into the solution.

$$\frac{k}{n} \cdot \frac{k-1}{n-1} \cdots \frac{k-t}{n-t} \cdots \frac{1}{n-(k-1)} = \frac{1}{\binom{n}{k}}$$

- In the beginning of the random process the success probability of each step is high.
- It gets progressively worse, and in the very end it is close to $1/n$.
- At some point we have picked $t$ vertices and $(k-t)/(n-t)$ drops below $1/c$. 
Analysis

A randomized algorithm that picks a random subset of size $k$ by picking one vertex at a time and inserting it into the solution.

\[
\frac{k}{n} \frac{k-1}{n-1} \cdots \frac{k-t}{n-t} \cdots \frac{1}{n-(k-1)} = \frac{1}{{n \choose k}}
\]

- In the beginning of the random process the success probability of each step is high.
- It gets progressively worse, and in the very end it is close to $1/n$.
- At some point we have picked $t$ vertices and $(k-t)/(n-t)$ drops below $1/c$.
- This is the time we run the extension algorithm, spending time $c^{k-t}$. 
If we continued brute force we will get..

\[
\binom{n-t}{k-t} = \frac{n-t}{k-t} \cdot \frac{n-t-1}{k-t-1} \cdots \frac{n-k+2}{2} \cdot \frac{n-k+1}{1}
\]

- Product of \(k-t\) larger and larger terms, with even the first and smallest term being greater than \(c\).
If we continued brute force we will get..

\[
\binom{n-t}{k-t} = \frac{n-t \cdot n-t-1 \cdot \ldots \cdot n-k+2 \cdot n-k+1}{k-t \cdot k-t-1 \cdot \ldots \cdot 2 \cdot 1}
\]

- Product of \(k-t\) larger and larger terms, with even the first and smallest term being greater than \(c\).
- Thus, any \(c^k\) algorithm will give some improvement over \(2^n\).
Applications

<table>
<thead>
<tr>
<th>Problem Name</th>
<th>Parameterized</th>
<th>New bound</th>
<th>Previous Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Feedback Vertex Set</strong></td>
<td>3(^k) (r)</td>
<td>1.6667(^n) (r)</td>
<td></td>
</tr>
<tr>
<td><strong>Feedback Vertex Set</strong></td>
<td>3.592(^k)</td>
<td>1.7217(^n)</td>
<td>1.7347(^n)</td>
</tr>
<tr>
<td><strong>Subset Feedback Vertex Set</strong></td>
<td>4(^k)</td>
<td>1.7500(^n)</td>
<td>1.8638(^n)</td>
</tr>
<tr>
<td><strong>Feedback Vertex Set in Tournaments</strong></td>
<td>1.6181(^k)</td>
<td>1.3820(^n)</td>
<td>1.4656(^n)</td>
</tr>
<tr>
<td>Group Feedback Vertex Set</td>
<td>4(^k)</td>
<td>1.7500(^n)</td>
<td>NPR</td>
</tr>
<tr>
<td>Node Unique Label Cover</td>
<td></td>
<td>(2 - (\frac{1}{\left</td>
<td>\Sigma\right</td>
</tr>
<tr>
<td>Vertex ((r, \ell))-Partization ((r, \ell \leq 2))</td>
<td>3.3146(^k)</td>
<td>1.6984(^n)</td>
<td>NPR</td>
</tr>
<tr>
<td>Interval Vertex Deletion</td>
<td>8(^k)</td>
<td>1.8750(^n)</td>
<td>(2 - (\varepsilon))(^n) for (\varepsilon &lt; 10^{-20}) [4]</td>
</tr>
<tr>
<td>Proper Interval Vertex Deletion</td>
<td>6(^k)</td>
<td>1.8334(^n)</td>
<td>(2 - (\varepsilon))(^n) for (\varepsilon &lt; 10^{-20}) [4]</td>
</tr>
<tr>
<td>Block Graph Vertex Deletion</td>
<td>4(^k)</td>
<td>1.7500(^n)</td>
<td>(2 - (\varepsilon))(^n) for (\varepsilon &lt; 10^{-20}) [4]</td>
</tr>
<tr>
<td>Cluster Vertex Deletion</td>
<td>1.9102(^k)</td>
<td>1.4765(^n)</td>
<td>1.6181(^n)</td>
</tr>
<tr>
<td>Thread Graph Vertex Deletion</td>
<td>8(^k)</td>
<td>1.8750(^n)</td>
<td>NPR</td>
</tr>
<tr>
<td>Multicut on Trees</td>
<td>1.5538(^k)</td>
<td>1.3565(^n)</td>
<td>NPR</td>
</tr>
<tr>
<td>3-Hitting Set</td>
<td>2.0755(^k)</td>
<td>1.5182(^n)</td>
<td>1.6278(^n)</td>
</tr>
<tr>
<td>4-Hitting Set</td>
<td>3.0755(^k)</td>
<td>1.6750(^n)</td>
<td>1.8704(^n)</td>
</tr>
<tr>
<td>(d)-Hitting Set ((d \geq 3))</td>
<td>((d - 0.9245)(^k)</td>
<td>(2 - (\frac{1}{(d-0.9245)}))(^n)</td>
<td>[11, 17]</td>
</tr>
<tr>
<td>Min-Ones 3-SAT</td>
<td>2.562(^k)</td>
<td>1.6097(^n)</td>
<td>NPR</td>
</tr>
<tr>
<td>Min-Ones (d)-SAT ((d \geq 4))</td>
<td>(d(^k)</td>
<td>(2 - (\frac{1}{d}))(^n)</td>
<td>NPR</td>
</tr>
<tr>
<td>Weighted (d)-SAT ((d \geq 3))</td>
<td>(d(^k)</td>
<td>(2 - (\frac{1}{d}))(^n)</td>
<td>NPR</td>
</tr>
<tr>
<td>Weighted Feedback Vertex Set</td>
<td>3.6181(^k)</td>
<td>1.7237(^n)</td>
<td>1.8638(^n)</td>
</tr>
<tr>
<td>Weighted 3-Hitting Set</td>
<td>2.168(^k)</td>
<td>1.5388(^n)</td>
<td>1.6755(^n)</td>
</tr>
<tr>
<td>Weighted (d)-Hitting Set ((d \geq 4))</td>
<td>((d - 0.832)(^k)</td>
<td>(2 - (\frac{1}{(d-0.832)}))(^n)</td>
<td>[11]</td>
</tr>
</tbody>
</table>

Table 1: Summary of known and new results for different optimization problems. NPR means that we are not aware of any previous algorithms faster than brute-force. All bounds suppress factors polynomial in the input size \(N\). The algorithms in the first row are randomized (r).
Theorem

Let $c > 1$ and $\Phi$ be an implicit set system. If $\Phi$ is $c$-uniform, then $|\mathcal{F}_I| \leq (2 - \frac{1}{c})^n n^{O(1)}$ for every instance $I$. 

Extension to Enumeration and Combinatorial upper bounds
## Applications

<table>
<thead>
<tr>
<th>Problem Name</th>
<th>c-uniform</th>
<th>New bound</th>
<th>Previous Bound</th>
<th>Ref</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimal FVSs in Tournaments</td>
<td>3</td>
<td>$1.6667^n$</td>
<td>$1.6740^n$</td>
<td>[23]</td>
</tr>
<tr>
<td>Minimal 3-Hitting Sets</td>
<td>3</td>
<td>$1.6667^n$</td>
<td>$1.6755^n$</td>
<td>[11]</td>
</tr>
<tr>
<td>Minimal 4-Hitting Sets</td>
<td>4</td>
<td>$1.7500^n$</td>
<td>$1.8863^n$</td>
<td>[11]</td>
</tr>
<tr>
<td>Minimal 5-Hitting Sets</td>
<td>5</td>
<td>$1.8000^n$</td>
<td>$1.9538^n$</td>
<td>[11]</td>
</tr>
<tr>
<td>Minimal $d$-Hitting Sets</td>
<td>$d$</td>
<td>$(2 - \frac{1}{d})^n$</td>
<td>$(2 - \epsilon_d)^n$ with $\epsilon_d &lt; 1/d$</td>
<td>[11]</td>
</tr>
<tr>
<td>Minimal $d$-SAT ($d \geq 2$)</td>
<td>$d$</td>
<td>$(2 - \frac{1}{d})^n$</td>
<td>NPR</td>
<td></td>
</tr>
<tr>
<td>Minimal FVSs in chordal graphs</td>
<td>3</td>
<td>$1.6667^n$</td>
<td>$1.6708^n$</td>
<td>[24]</td>
</tr>
<tr>
<td>Minimal Subset FVSs in chordal graphs</td>
<td>3</td>
<td>$1.6667^n$</td>
<td>NPR</td>
<td></td>
</tr>
<tr>
<td>Maximal $r$-colorable induced subgraphs of perfect graphs</td>
<td>$r+1$</td>
<td>$(2 - \frac{1}{r+1})^n$</td>
<td>NPR</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Summary of known and new results for different combinatorial bounds. NPR means that we are not aware of any previous results better than $2^n$. All bounds suppress factors polynomial in the universe size $n$. 
Conclusion and Derandomization

**Theorem**

If there exists an algorithm for $\Phi$-EXTENSION with running time $c^k n^{O(1)}$ then there exists a randomized algorithm for $\Phi$-SUBSET with running time $(2 - \frac{1}{c}) n^{O(1)}$. 
Conclusion and Derandomization

**Theorem**

If there exists an algorithm for $\Phi$-EXTENSION with running time $c^k n^{O(1)}$ then there exists a randomized algorithm for $\Phi$-SUBSET with running time $(2 - \frac{1}{c}) n^{O(1)}$.

**Theorem**

If there exists an algorithm for $\Phi$-EXTENSION with running time $c^k n^{O(1)}$ then there exists an algorithm for $\Phi$-SUBSET with running time $(2 - \frac{1}{c})^{n+o(n)} n^{O(1)}$. 
Thank You!
Any Questions?