Exponential-Time Algorithms for NP Problems: Prospects and Limits

Andrew Drucker

IAS

Oct. 4, 2013

Basic concepts

- NP **problems**: decision problems whose "Yes" instances have short certificates

  (checkable in time polynomial in input length)

- NP-**complete problems**: "hardest" problems in this class.

- Believed not to be solvable in polynomial time. ("P $\neq$ NP")

- Exponential Time Hypothesis **(ETH)**[Impagliazzo, Paturi, Zane '97]: NP-complete problems require exponential time (roughly speaking)

Basic concepts

- NP **problems**: decision problems whose "Yes" instances have short certificates

  (checkable in time polynomial in input length)

- NP-**complete problems**: "hardest" problems in this class.

- Believed not to be solvable in polynomial time. ("$P \neq NP$")

- Exponential Time Hypothesis **(ETH)**[Impagliazzo, Paturi, Zane '97]:
  NP-complete problems require exponential time (roughly speaking)

Basic concepts

- NP **problems**: decision problems whose "Yes" instances have short certificates

  (checkable in time polynomial in input length)

- NP-**complete problems**: "hardest" problems in this class.

- Believed not to be solvable in polynomial time. ("P $\neq$ NP")

- Exponential Time Hypothesis **(ETH)**[Impagliazzo, Paturi, Zane '97] :

  NP-complete problems require exponential time (roughly speaking)

Example: Subset Sum

- **INPUT:** integers $a_1, \ldots, a_n, T$ *//each of bitlength $O(n)$*
- **DECIDE:** is there a subset $J \subseteq [n]$ such that $\sum_{j \in J} a_j = T$?

Natural certificate: the set J.

Naïve algorithm: $\sim n^2 \cdot 2^n$ steps.

Example: Subset Sum

- **INPUT:** integers $a_1, \ldots, a_n, T$ *//each of bitlength $O(n)$*
- **DECIDE:** is there a subset $J \subseteq [n]$ such that $\sum_{j \in J} a_j = T$?

Natural certificate: the set J.

Naïve algorithm: $\sim n^2 \cdot 2^n$ steps.

Example: Subset Sum

- **INPUT:** integers $a_1, \ldots, a_n, T$ *//each of bitlength* $O(n)$
- **DECIDE:** is there a subset $J \subseteq [n]$ such that $\sum_{j \in J} a_j = T$?

Natural certificate: the set J.

Naïve algorithm: $\sim n^2 \cdot 2^n$ steps.

Example: Subset Sum

- **INPUT:** integers $a_1, \ldots, a_n, T$
- **DECIDE:** is there a subset $J \subseteq [n]$ such that $\sum_{j \in J} a_j = T$?

- NP-complete, so $\mathsf{P} \neq \mathsf{NP}$ conjecture rules out a poly(n)-time algorithm for this problem...
- ETH rules out a $2^{o(n)}$-time algorithm.
- But, neither hypothesis rules out improvements on brute-force search!

Example: Subset Sum

- **INPUT:** integers $a_1, \ldots, a_n, T$
- **DECIDE:** is there a subset $J \subseteq [n]$ such that $\sum_{j \in J} a_j = T$?

- NP-complete, so $\mathrm{P} \neq \mathrm{NP}$ conjecture rules out a $\mathrm{poly}(n)$-time algorithm for this problem...
- ETH rules out a $2^{o(n)}$-time algorithm.
- But, neither hypothesis rules out improvements on brute-force search!

Example: Subset Sum

- **INPUT:** integers $a_1, \ldots, a_n, T$
- **DECIDE:** is there a subset $J \subseteq [n]$ such that $\sum_{j \in J} a_j = T$?

- NP-complete, so P $\neq$ NP conjecture rules out a poly(n)-time algorithm for this problem...
- ETH rules out a $2^{o(n)}$-time algorithm.
- But, neither hypothesis rules out improvements on brute-force search!

Example: Subset Sum

- **INPUT:** integers $a_1, \ldots, a_n, T$
- **DECIDE:** is there a subset $J \subseteq [n]$ such that $\sum_{j \in J} a_j = T$?

- NP-complete, so $\mathsf{P} \neq \mathsf{NP}$ conjecture rules out a $\mathrm{poly}(n)$-time algorithm for this problem...
- ETH rules out a $2^{o(n)}$-time algorithm.
- But, neither hypothesis rules out improvements on brute-force search!

Improved algorithm for Subset Sum

- **INPUT:** integers $a_1, \ldots, a_n, T$
- **DECIDE:** is there a subset $J \subseteq [n]$ such that $\sum_{j \in J} a_j = T$?

"Meet-in-the-middle" algorithm [Horowitz, Sahni '74] :

1. Compute $L :=$ (all possible subsums of $a_1, \ldots, a_{n/2}$);
2. Compute $R :=$ (all possible subsums of $a_{n/2+1}, \ldots, a_n$);
3. **SORT** each of L, R;
4. Check if $T \in L + R$.

Claim

Each step can be performed in $2^{n/2} \cdot \text{poly}(n)$ steps.

Improved algorithm for Subset Sum

- **INPUT:** integers $a_1, \ldots, a_n, T$
- **DECIDE:** is there a subset $J \subseteq [n]$ such that $\sum_{j \in J} a_j = T$?

"Meet-in-the-middle" algorithm [Horowitz, Sahni '74] :

1. Compute $L :=$ (all possible subsums of $a_1, \ldots, a_{n/2}$);
2. Compute $R :=$ (all possible subsums of $a_{n/2+1}, \ldots, a_n$);
3. **SORT** each of L, R;
4. Check if $T \in L + R$.

Claim

Each step can be performed in $2^{n/2} \cdot \text{poly}(n)$ steps.

Improved algorithm for Subset Sum

- **INPUT:** integers $a_1, \ldots, a_n, T$
- **DECIDE:** is there a subset $J \subseteq [n]$ such that $\sum_{j \in J} a_j = T$?

"Meet-in-the-middle" algorithm [Horowitz, Sahni '74] :

1. Compute $L :=$ (all possible subsums of $a_1, \ldots, a_{n/2}$);
2. Compute $R :=$ (all possible subsums of $a_{n/2+1}, \ldots, a_n$);
3. **SORT** each of L, R;
4. Check if $T \in L + R$.

Claim

Each step can be performed in $2^{n/2} \cdot \text{poly}(n)$ steps.

Improved algorithm for Subset Sum

"Meet-in-the-middle" algorithm [Horowitz, Sahni '74] :

4. Check if $T \in L + R$.

1	4	5	9	13

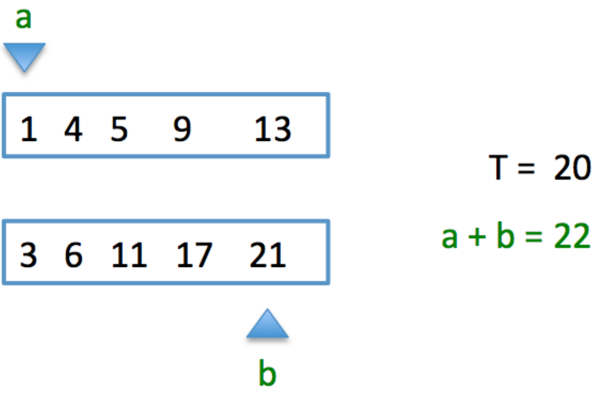
3	6	11	17	21

T = 20

Improved algorithm for Subset Sum

"Meet-in-the-middle" algorithm [Horowitz, Sahni '74] :

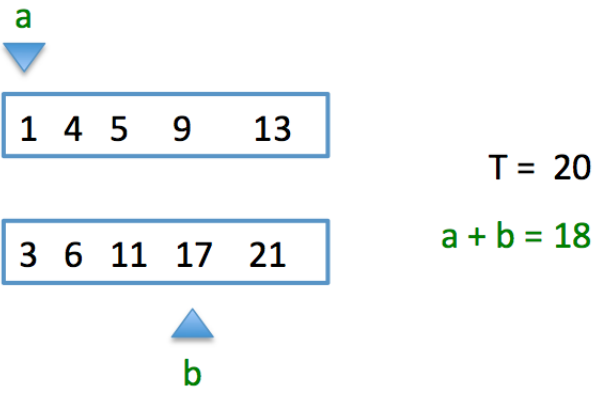
4. Check if $T \in L + R$.



Improved algorithm for Subset Sum

"Meet-in-the-middle" algorithm [Horowitz, Sahni '74] :

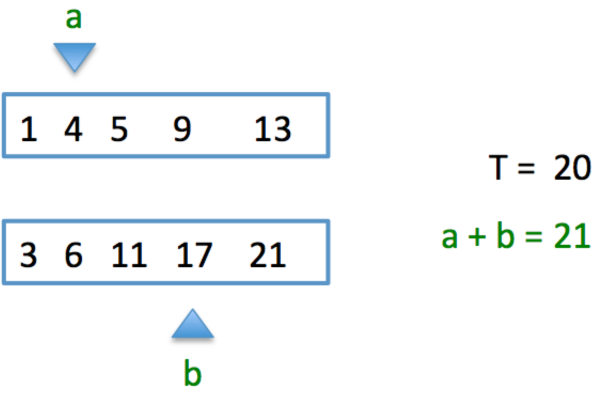
4. Check if $T \in L + R$.



Improved algorithm for Subset Sum

"Meet-in-the-middle" algorithm [Horowitz, Sahni '74] :

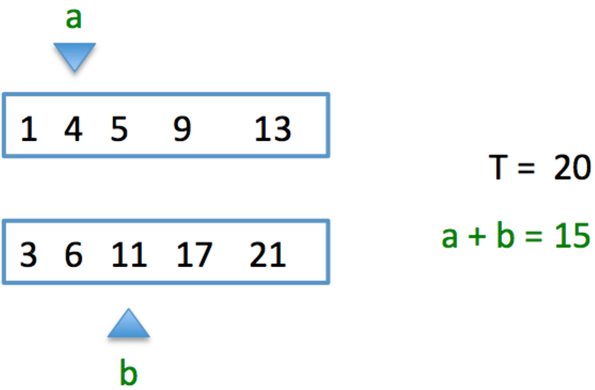
4. Check if $T \in L + R$.



Improved algorithm for Subset Sum

"Meet-in-the-middle" algorithm [Horowitz, Sahni '74] :

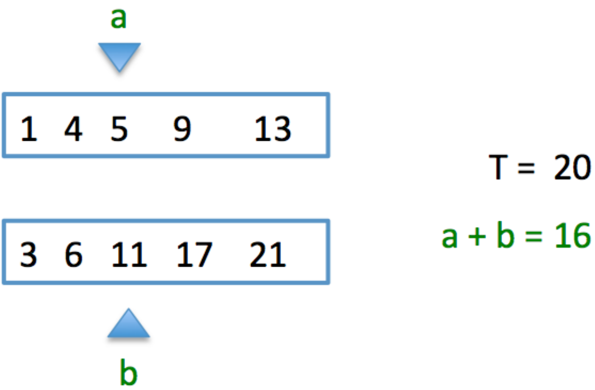
4. Check if $T \in L + R$.



Improved algorithm for Subset Sum

"Meet-in-the-middle" algorithm [Horowitz, Sahni '74] :

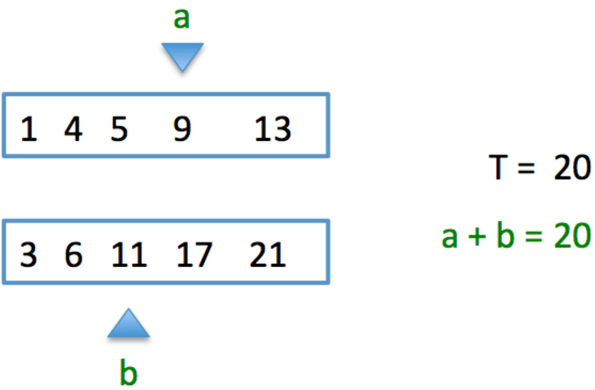
4. Check if $T \in L + R$.



Improved algorithm for Subset Sum

"Meet-in-the-middle" algorithm [Horowitz, Sahni '74] :

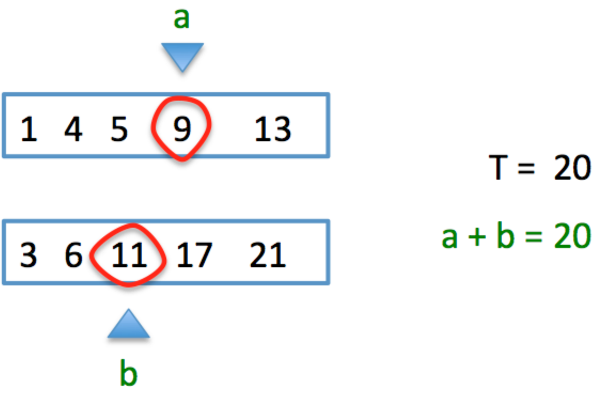
4. Check if $T \in L + R$.



Improved algorithm for Subset Sum

"Meet-in-the-middle" algorithm [Horowitz, Sahni '74] :

4. Check if $T \in L + R$.



Improved algorithm for Subset Sum

- $2^{n/2}$ runtime: quite an gain over 2^n...
- **BUT**, since 1974, no further speedups for this problem!

    (except for special cases)

- No evidence $2^{n/2}$ is optimal!
- One route to progress: the "k-SUM" problem...

Improved algorithm for Subset Sum

- $2^{n/2}$ runtime: quite an gain over 2^n...
- **BUT**, since 1974, no further speedups for this problem!

  (except for special cases)
- No evidence $2^{n/2}$ is optimal!
- One route to progress: the "k-SUM" problem...

Improved algorithm for Subset Sum

- $2^{n/2}$ runtime: quite an gain over 2^n...
- **BUT**, since 1974, no further speedups for this problem!

  (except for special cases)
- No evidence $2^{n/2}$ is optimal!
- One route to progress: the "k-SUM" problem...

Improved algorithm for Subset Sum

- $2^{n/2}$ runtime: quite an gain over 2^n...
- **BUT**, since 1974, no further speedups for this problem!

  (except for special cases)
- No evidence $2^{n/2}$ is optimal!
- One route to progress: the "k-SUM" problem...

k-Sum

$k \geq 3$ a fixed integer.

- **INPUT:** sets of integers $A_1, \ldots, A_k$ each of size n; a target value T.
- **DECIDE:** Is $T \in A_1 + \ldots + A_k$?
- Best known algorithm: $\sim n^{\lceil k/2 \rceil}$ steps.
  Significant improvements would also improve the best algorithms for SUBSET-SUM and other NP-complete problems.
  OTOH, the ETH implies that k-SUM requires time $n^{\Omega(k)}$.
  [Woeginger '04], [Patrascu, Williams '10]
- Many such connections were found between the complexity of polynomial-time solvable problems (like k-SUM) and NP-complete problems (like SUBSET-SUM).

  Deeper connections may exist.

k-Sum

$k \geq 3$ a fixed integer.

- **INPUT:** sets of integers $A_1, \ldots, A_k$ each of size n; a target value T.
- **DECIDE:** Is $T \in A_1 + \ldots + A_k$?
- Best known algorithm: $\sim n^{\lceil k/2 \rceil}$ steps.
  Significant improvements would also improve the best algorithms for SUBSET-SUM and other NP-complete problems.
  OTOH, the ETH implies that k-SUM requires time $n^{\Omega(k)}$.
  [Woeginger '04], [Patrascu, Williams '10]
- Many such connections were found between the complexity of polynomial-time solvable problems (like k-SUM) and NP-complete problems (like SUBSET-SUM).

  Deeper connections may exist.

k-Sum

$k \geq 3$ a fixed integer.

- **INPUT:** sets of integers $A_1, \ldots, A_k$ each of size n; a target value T.
- **DECIDE:** Is $T \in A_1 + \ldots + A_k$?
- Best known algorithm: $\sim n^{\lceil k/2 \rceil}$ steps.
  Significant improvements would also improve the best algorithms for SUBSET-SUM and other NP-complete problems.
  OTOH, the ETH implies that k-SUM requires time $n^{\Omega(k)}$.
  [Woeginger '04], [Patrascu, Williams '10]
- Many such connections were found between the complexity of polynomial-time solvable problems (like k-SUM) and NP-complete problems (like SUBSET-SUM).

  Deeper connections may exist.

k-Sum

$k \geq 3$ a fixed integer.

- **INPUT:** sets of integers $A_1, \ldots, A_k$ each of size n; a target value T.
- **DECIDE:** Is $T \in A_1 + \ldots + A_k$?
- Best known algorithm: $\sim n^{\lceil k/2 \rceil}$ steps.
  Significant improvements would also improve the best algorithms for SUBSET-SUM and other NP-complete problems.
  OTOH, the ETH implies that k-SUM requires time $n^{\Omega(k)}$.
  [Woeginger '04], [Patrascu, Williams '10]
- Many such connections were found between the complexity of polynomial-time solvable problems (like k-SUM) and NP-complete problems (like SUBSET-SUM).

  Deeper connections may exist.

k-Sum

$k \geq 3$ a fixed integer.

- **INPUT:** sets of integers $A_1, \ldots, A_k$ each of size n; a target value T.
- **DECIDE:** Is $T \in A_1 + \ldots + A_k$?
- Best known algorithm: $\sim n^{\lceil k/2 \rceil}$ steps.
  Significant improvements would also improve the best algorithms for SUBSET-SUM and other NP-complete problems.
  OTOH, the ETH implies that k-SUM requires time $n^{\Omega(k)}$.
  [Woeginger '04], [Patrascu, Williams '10]
- Many such connections were found between the complexity of polynomial-time solvable problems (like k-SUM) and NP-complete problems (like SUBSET-SUM).

  Deeper connections may exist.

Example: k-SAT

Example: k-SAT

$k \geq 3$ a fixed integer.

- **INPUT:** a CNF formula $\phi(x_1, \ldots, x_n)$, each clause of length $\leq k$.
- **DECIDE:** is there a variable assignment $\overline{x}$ such that $\phi(\overline{x}) = \text{TRUE}$?

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

- Exponential Time Hypothesis **(ETH)**:

For suff. small $\delta > 0$, 3-SAT can't be solved in time $2^{\delta n} \cdot \text{poly}(|\phi|)$.

Example: k-SAT

$k \geq 3$ a fixed integer.

- **INPUT:** a CNF formula $\phi(x_1, \ldots, x_n)$,
  each clause of length $\leq k$.

- **DECIDE:** is there a variable assignment $\overline{x}$ such that $\phi(\overline{x}) = \text{TRUE}$?

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

- Exponential Time Hypothesis **(ETH)**:

For suff. small $\delta > 0$, 3-SAT can't be solved in time $2^{\delta n} \cdot \text{poly}(|\phi|)$.

Improved algorithm for k-SAT

- Will see a method to solve k-SAT by **intelligent random guessing**.

Theorem [Paturi, Pudlák, Zane '97]

$\exists$ a poly(n)-time randomized algorithm A:
for any satisfiable ϕ, A finds a satisfying assignment with probability $\geq \frac{1}{n} \cdot 2^{-n+n/k}$.

$\implies$ can run A for $\sim n \cdot 2^{n-n/k}$ trials to obtain a solution w.h.p.

- Many of the known improved algs for NP-complete problems have this form! (or, can be re-expressed in this form) [Paturi, Pudlák '10]

- A rich, natural paradigm for algorithm design.

Improved algorithm for k-SAT

- Will see a method to solve k-SAT by **intelligent random guessing**.

Theorem [Paturi, Pudlák, Zane '97]

$\exists$ a poly(n)-time randomized algorithm A:
for any satisfiable ϕ, A finds a satisfying assignment with probability $\geq \frac{1}{n} \cdot 2^{-n+n/k}$.

$\Longrightarrow$ can run A for $\sim n \cdot 2^{n-n/k}$ trials to obtain a solution w.h.p.

- Many of the known improved algs for NP-complete problems have this form! (or, can be re-expressed in this form) [Paturi, Pudlák '10]

- A rich, natural paradigm for algorithm design.

Improved algorithm for k-SAT

- Will see a method to solve k-SAT by **intelligent random guessing**.

Theorem [Paturi, Pudlák, Zane '97]

$\exists$ a poly(n)-time randomized algorithm A:
for any satisfiable ϕ, A finds a satisfying assignment with probability $\geq \frac{1}{n} \cdot 2^{-n+n/k}$.

$\Longrightarrow$ can run A for $\sim n \cdot 2^{n-n/k}$ trials to obtain a solution w.h.p.

- Many of the known improved algs for NP-complete problems have this form! (or, can be re-expressed in this form) [Paturi, Pudlák '10]
- A rich, natural paradigm for algorithm design.

Improved algorithm for k-SAT

- Will see a method to solve k-SAT by **intelligent random guessing**.

Theorem [Paturi, Pudlák, Zane '97]

$\exists$ a poly(n)-time randomized algorithm A:
for any satisfiable ϕ, A finds a satisfying assignment with probability $\geq \frac{1}{n} \cdot 2^{-n+n/k}$.

$\implies$ can run A for $\sim n \cdot 2^{n-n/k}$ trials to obtain a solution w.h.p.

- Many of the known improved algs for NP-complete problems have this form! (or, can be re-expressed in this form) [Paturi, Pudlák '10]

- A rich, natural paradigm for algorithm design.

Improved algorithm for k-SAT

To attempt to produce a satisfying assignment to k-CNF formula $\phi(x_1, \ldots, x_n)$:

Algorithm A [PPZ]:

1. Pick a random permutation $\sigma \in S_n$ ("reordering" of $x_1, \ldots, x_n$);
2. **For** $i = 1, 2, \ldots, n$:
   - **If** $x_{\sigma(i)}$ is "critical" for ϕ under current assignment, then set accordingly;
   - **Else** set $x_{\sigma(i)}$ randomly;

Improved algorithm for k-SAT

To attempt to produce a satisfying assignment to k-CNF formula $\phi(x_1, \dots, x_n)$:

Algorithm A [PPZ]:

1. Pick a random permutation $\sigma \in S_n$ ("reordering" of $x_1, \dots, x_n$);
2. **For** $i = 1, 2, \dots, n$:
   - **If** $x_{\sigma(i)}$ is "critical" for ϕ under current assignment, then set accordingly;
   - **Else** set $x_{\sigma(i)}$ randomly;

Improved algorithm for k-SAT

To attempt to produce a satisfying assignment to k-CNF formula $\phi(x_1, \ldots, x_n)$:

Algorithm A [PPZ]:

1. Pick a random permutation $\sigma \in S_n$ ("reordering" of $x_1, \ldots, x_n$);
2. **For** $i = 1, 2, \ldots, n$:
   - **If** $x_{\sigma(i)}$ is "critical" for ϕ under current assignment, then set accordingly;
   - **Else** set $x_{\sigma(i)}$ randomly;

Improved algorithm for k-SAT

To attempt to produce a satisfying assignment to k-CNF formula $\phi(x_1, \ldots, x_n)$:

Algorithm A [PPZ] :

1. Pick a random permutation $\sigma \in S_n$ ("reordering" of $x_1, \ldots, x_n$);
2. **For** $i = 1, 2, \ldots, n$:
   - **If** $x_{\sigma(i)}$ is "critical" for ϕ under current assignment, then set accordingly;
   - **Else** set $x_{\sigma(i)}$ randomly;

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

Improved algorithm for k-SAT

To attempt to produce a satisfying assignment to k-CNF formula $\phi(x_1, \ldots, x_n)$:

Algorithm A [PPZ] :

1. Pick a random permutation $\sigma \in S_n$ ("reordering" of $x_1, \ldots, x_n$);
2. **For** $i = 1, 2, \ldots, n$:
   - **If** $x_{\sigma(i)}$ is "critical" for ϕ under current assignment, then set accordingly;
   - **Else** set $x_{\sigma(i)}$ randomly;

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

Improved algorithm for k-SAT

To attempt to produce a satisfying assignment to k-CNF formula $\phi(x_1, \ldots, x_n)$:

Algorithm A [PPZ]:

1. Pick a random permutation $\sigma \in S_n$ ("reordering" of $x_1, \ldots, x_n$);
2. **For** $i = 1, 2, \ldots, n$:
   - **If** $x_{\sigma(i)}$ is "critical" for ϕ under current assignment, then set accordingly;
   - **Else** set $x_{\sigma(i)}$ randomly;

$$(x_1 \lor 1 \lor x_4) \land (\neg 1 \lor x_3) \land (x_3 \lor \neg x_4)$$

Improved algorithm for k-SAT

To attempt to produce a satisfying assignment to k-CNF formula $\phi(x_1, \ldots, x_n)$:

Algorithm A [PPZ] :

1. Pick a random permutation $\sigma \in S_n$ ("reordering" of $x_1, \ldots, x_n$);
2. **For** $i = 1, 2, \ldots, n$:
   - **If** $x_{\sigma(i)}$ is "critical" for ϕ under current assignment, then set accordingly;
   - **Else** set $x_{\sigma(i)}$ randomly;

$$(\text{TRUE}) \wedge (\neg 1 \vee x_3) \wedge (x_3 \vee \neg x_4)$$

Improved algorithm for k-SAT

To attempt to produce a satisfying assignment to k-CNF formula $\phi(x_1, \ldots, x_n)$:

Algorithm A [PPZ]:

1. Pick a random permutation $\sigma \in S_n$ ("reordering" of $x_1, \ldots, x_n$);
2. **For** $i = 1, 2, \ldots, n$:
   - **If** $x_{\sigma(i)}$ is "critical" for ϕ under current assignment, then set accordingly;
   - **Else** set $x_{\sigma(i)}$ randomly;

$$(\neg 1 \vee x_3) \wedge (x_3 \vee \neg x_4)$$

Improved algorithm for k-SAT

To attempt to produce a satisfying assignment to k-CNF formula $\phi(x_1, \ldots, x_n)$:

Algorithm A [PPZ]:

1. Pick a random permutation $\sigma \in S_n$ ("reordering" of $x_1, \ldots, x_n$);
2. **For** $i = 1, 2, \ldots, n$:
   - **If** $x_{\sigma(i)}$ is "critical" for ϕ under current assignment, then set accordingly;
   - **Else** set $x_{\sigma(i)}$ randomly;

$$(x_3) \wedge (x_3 \vee \neg x_4)$$

Improved algorithm for k-SAT

To attempt to produce a satisfying assignment to k-CNF formula $\phi(x_1, \ldots, x_n)$:

Algorithm A [PPZ]:

1. Pick a random permutation $\sigma \in S_n$ ("reordering" of $x_1, \ldots, x_n$);
2. **For** $i = 1, 2, \ldots, n$:
   - **If** $x_{\sigma(i)}$ is "critical" for ϕ under current assignment, then set accordingly;
   - **Else** set $x_{\sigma(i)}$ randomly;

$$(x_3) \wedge (x_3 \vee \neg x_4)$$

Improved algorithm for k-SAT

To attempt to produce a satisfying assignment to k-CNF formula $\phi(x_1, \ldots, x_n)$:

Algorithm A [PPZ]:

1. Pick a random permutation $\sigma \in S_n$ ("reordering" of $x_1, \ldots, x_n$);
2. **For** $i = 1, 2, \ldots, n$:
   - **If** $x_{\sigma(i)}$ is "critical" for ϕ under current assignment, then set accordingly;
   - **Else** set $x_{\sigma(i)}$ randomly;

$$(x_3) \wedge (x_3 \vee \neg x_4)$$

Improved algorithm for k-SAT

To attempt to produce a satisfying assignment to k-CNF formula $\phi(x_1, \ldots, x_n)$:

Algorithm A [PPZ]:

1. Pick a random permutation $\sigma \in S_n$ ("reordering" of $x_1, \ldots, x_n$);
2. **For** $i = 1, 2, \ldots, n$:
   - **If** $x_{\sigma(i)}$ is "critical" for ϕ under current assignment, then set accordingly;
   - **Else** set $x_{\sigma(i)}$ randomly;

$$(1) \wedge (1 \vee \neg x_4)$$

Improved algorithm for k-SAT

To attempt to produce a satisfying assignment to k-CNF formula $\phi(x_1, \ldots, x_n)$:

Algorithm A [PPZ]:

1. Pick a random permutation $\sigma \in S_n$ ("reordering" of $x_1, \ldots, x_n$);
2. **For** $i = 1, 2, \ldots, n$:
   - **If** $x_{\sigma(i)}$ is "critical" for ϕ under current assignment, then set accordingly;
   - **Else** set $x_{\sigma(i)}$ randomly;

TRUE

Intelligent guessing procedures — Limits

- ETH $\Longrightarrow$ no poly-time procedure can achieve success probability $\geq 2^{-o(n)}$ for solving satisfiable 3-SAT *instances*.

  (But, ETH is a very strong assumption...)

- Can prove limits of poly-time random guessing under weaker hypotheses.

Intelligent guessing procedures — Limits

- ETH $\Longrightarrow$ no poly-time procedure can achieve success probability $\geq 2^{-o(n)}$ for solving satisfiable 3-SAT *instances*.

  (But, ETH is a very strong assumption...)

- Can prove limits of poly-time random guessing under weaker hypotheses.

Intelligent guessing procedures — Limits

- ETH $\implies$ no poly-time procedure can achieve success probability $\geq 2^{-o(n)}$ for solving satisfiable 3-SAT *instances*.

  (But, ETH is a very strong assumption...)

- Can prove limits of poly-time random guessing under weaker hypotheses.

Intelligent guessing procedures — Limits

Theorem [Paturi, Pudlák '10]

If some poly-time random guessing procedure can achieve success probability $\geq 2^{-.9n}$ for solving satisfiable Circuit-SAT instances,

then, Circuit-SAT has (non-uniform) algorithms of runtime $2^{n^{.99}}$.

Theorem [D '13]

If some poly-time random guessing procedure can achieve success probability $\geq 2^{-n^{.9}}$ for solving satisfiable 3-SAT instances,

then $NP \subseteq coNP/poly$.

Intelligent guessing procedures — Limits

Theorem [Paturi, Pudlák '10]

If some poly-time random guessing procedure can achieve success probability $\geq 2^{-.9n}$ for solving satisfiable Circuit-SAT instances,

then, Circuit-SAT has (non-uniform) algorithms of runtime $2^{n^{.99}}$.

Theorem [D '13]

If some poly-time random guessing procedure can achieve success probability $\geq 2^{-n^{.9}}$ for solving satisfiable 3-SAT instances,

then $\mathsf{NP} \subseteq \mathsf{coNP}/\mathsf{poly}$.

What's next?

- May be possible to prove strong limits on other restricted algorithms for solving NP-complete problems.

  (under reasonable hardness assumptions)

- Candidate: Algorithms with superpolynomial time budget, but polynomially-bounded space budget.

- E.g., unknown whether we can solve SUBSET-SUM in time $(1.99)^n$ using space $\text{poly}(n)$...

What's next?

- May be possible to prove strong limits on other restricted algorithms for solving NP-complete problems.

  (under reasonable hardness assumptions)

- Candidate: Algorithms with superpolynomial time budget, but polynomially-bounded space budget.

- E.g., unknown whether we can solve SUBSET-SUM in time $(1.99)^n$ using space $\text{poly}(n)$...