Average-case Analysis for Combinatorial Problems,

with Subset Sums and Stochastic Spanning Trees

Abraham D. Flaxman

Mathematical Sciences,
Carnegie Mellon University

February 2, 2006

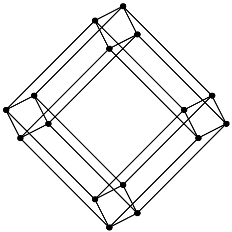
Outline

Combinatorial Problems

You've got some finite collection of objects and you'd like to find a special one.

For example

- In graphs
  - A spanning tree
  - A perfect matching
  - A Hamiltonian cycle
- In Boolean formulas
  - A satisfying assignment

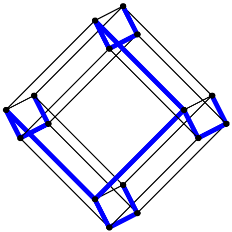


Combinatorial Problems

You've got some finite collection of objects and you'd like to find a special one.

For example

- In graphs
  - A spanning tree
  - A perfect matching
  - A Hamiltonian cycle
- In Boolean formulas
  - A satisfying assignment

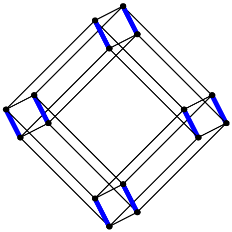


Combinatorial Problems

You've got some finite collection of objects and you'd like to find a special one.

For example

- In graphs
  - A spanning tree
  - A perfect matching
  - A Hamiltonian cycle
- In Boolean formulas
  - A satisfying assignment

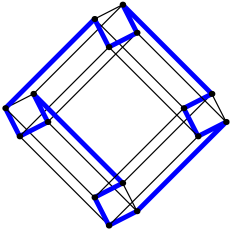


Combinatorial Problems

You've got some finite collection of objects and you'd like to find a special one.

For example

- In graphs
  - A spanning tree
  - A perfect matching
  - A Hamiltonian cycle
- In Boolean formulas
  - A satisfying assignment

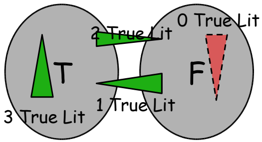


Combinatorial Problems

You've got some finite collection of objects and you'd like to find a special one.

For example

- In graphs
  - A spanning tree
  - A perfect matching
  - A Hamiltonian cycle
- In Boolean formulas
  - A satisfying assignment



Combinatorial Problems

- This is a trivial matter to a mathematician in the 1940s
- Not trivial anymore by the 1970s

Combinatorial Problems

- This is a trivial matter to a mathematician in the 1940s
- Not trivial anymore by the 1970s

Edmonds, 1963:

"For practical purposes computational details are vital. However, my purpose is only to show as attractively as I can that there is an efficient algorithm. According to the dictionary, "efficient" means "adequate in operation or performance." This is roughly the meaning I want—in the sense that it is conceivable for maximum to have no efficient algorithm. Perhaps a better word is "good." I am claiming, as a mathematical result, the existence of a *good* algorithm for finding a maximum matching in a graph.

There is an obvious finite algorithm, but that algorithm increases in difficulty exponentially with the size of the graph. It is by no means obvious whether *or not* there exists an algorithm whose difficulty increases only algebraically with the size of the graph.

Average-case Analysis of Algorithms

Problems in the real-world incorporate elements of chance, so an algorithm need not be good for all instances, as long as it is likely to work on the instances that show up.

Example: Simplex Algorithm

Linear programming asks for a vector $\mathbf{x} \in \mathbb{R}^n$ which satisfies

$$\begin{aligned} \mathbf{Ax} &\leq \mathbf{b} \\ \mathbf{x} &\geq \mathbf{0}. \end{aligned}$$

The simplex algorithm is known to take exponential time on certain inputs, but it has still been remarkably useful in practice. Could be because the computationally difficult instances are unlikely to come up.

Example: Simplex Algorithm

Explain with average-case analysis:

Example: Simplex Algorithm

Explain with average-case analysis:

- Attempt 1: Analyze performance when each $\mathbf{A}_{ij}$ is independent, normally distributed random variable

Example: Simplex Algorithm

Explain with average-case analysis:

- Attempt 1: Analyze performance when each $\mathbf{A}_{ij}$ is independent, normally distributed random variable
- Attempt 2: Make $\mathbf{A}_{ij}$ i.i.d., distributed from *some* symmetric distribution

Example: Simplex Algorithm

Explain with average-case analysis:

- Attempt 1: Analyze performance when each $\mathbf{A}_{ij}$ is independent, normally distributed random variable
- Attempt 2: Make $\mathbf{A}_{ij}$ i.i.d., distributed from *some* symmetric distribution
- Smoothed Analysis: Start with $\mathbf{A}_{ij}$ arbitrary, and *perturb* it by adding normally distributed r.v. to each entry (prove that run-time depends on variance of r.v. σ^2)

Example: Simplex Algorithm

Explain with average-case analysis:

- Attempt 1: Analyze performance when each $\mathbf{A}_{ij}$ is independent, normally distributed random variable
- Attempt 2: Make $\mathbf{A}_{ij}$ i.i.d., distributed from *some* symmetric distribution
- Smoothed Analysis: Start with $\mathbf{A}_{ij}$ arbitrary, and *perturb* it by adding normally distributed r.v. to each entry (prove that run-time depends on variance of r.v. σ^2)

Smoothed Analysis of some connectivity problems in (Flaxman and Frieze, RANDOM-APPROX 2004)

Assumptions

- Average-case explanation of observed performance requires making assumptions about how instances are random.

Assumptions

- Question these assumptions.
- Use distributions that are more accurate assumptions.

Assumptions

- Question these assumptions.
- Use distributions that are more accurate assumptions.

Power-law Graphs
(Flaxman, Frieze, Fenner, RANDOM-APPROX 2003)
(Flaxman, Frieze, Vera, SODA 2005)

Geometric Random Graph
(Flaxman, Frieze, Upfal, J. Algorithms 2004),
(Flaxman, Frieze, Vera, STOC 2005),

Geometric Power Law Graphs
(Flaxman, Frieze, Vera, WAW 2005)

Searching for difficult distributions

- If you knew a distribution for which no good algorithms exist (and especially if this distribution gave problem instances together with a solution) then you could use it as a cryptographic primitive.
- And besides, knowing where the hard problems are is interesting in its own right, right?

Example: Planted 3-SAT

- Choose an assignment for n Boolean variables, and generate a 3-CNF formula satisfied by this assignment by including each clause consistent with the assignment independently at random.

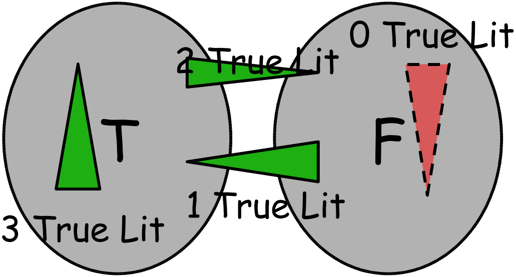
Example: Planted 3-SAT

- Choose an assignment for n Boolean variables, and generate a 3-CNF formula satisfied by this assignment by including each clause consistent with the assignment independently at random.
- Take all consistent clauses with the same probability and efficient algorithm succeeds **whp** (for dense enough instances). (Flaxman, SODA 2003)

Example: Planted 3-SAT

- Choose an assignment for n Boolean variables, and generate a 3-CNF formula satisfied by this assignment by including each clause consistent with the assignment independently at random.
- Take all consistent clauses with the same probability and efficient algorithm succeeds **whp** (for dense enough instances). (Flaxman, SODA 2003)
- But carefully adjust the probabilities so clauses with 2 true literals don't appear too frequently then no efficient algorithm is known.

End of the philosophy section



The (Modular) Subset Sum Problem

Input: Modulus $M \in \mathbb{Z}$,

Numbers $a_1, \ldots, a_n \in \{0, 1, \ldots, M-1\}$,

Target $T \in \{0, 1, \ldots, M-1\}$.

Goal: Find $S \subseteq \{1, 2, \ldots, n\}$ such that

$$\sum_{i \in S} a_i \equiv T \mod M$$

(if such a set exists.)

The (Modular) Subset Sum Problem

Subset sum is **NP**-hard.
But in **P** when $M = \text{poly}(n)$.

A natural distribution for random instances is

- Make M some appropriate function of n,
- Pick $a_1, \ldots, a_n$ independently and uniformly at random from $\{0, 1, \ldots, M-1\}$,
- Make T the sum of a random subset of the a_i's.

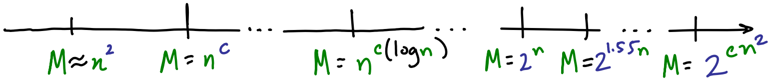
Sketch of computational difficulty as a function of M

- $M \geq 2^{n^2/2}$, a poly-time algorithm using Lovász basis reduction succeeds **whp**,
- $M \geq 2^{1.55n}$, similar algorithms seem to work,
- $M = 2^n$, seems to be “most difficult”,

Sketch of computational difficulty as a function of M

- $M \geq 2^{n^2/2}$, a poly-time algorithm using Lovász basis reduction succeeds **whp**,
- $M \geq 2^{1.55n}$, similar algorithms seem to work,
- $M = 2^n$, seems to be "most difficult",
- $M = \text{poly}(n)$, worst-case dynamic programming in $\mathcal{O}(n^2 M)$ is poly-time,
- $M \leq n^2/\log n$, alg faster than dynamic programming exists.

Sketch of computational difficulty as a function of M



- $M \geq 2^{n^2/2}$, a poly-time algorithm using Lovász basis reduction succeeds **whp**,
- $M \geq 2^{1.55n}$, similar algorithms seem to work,
- $M = 2^n$, seems to be “most difficult”,
- $M = n^{\mathcal{O}(\log n)}$, poly-time algorithm succeeds **whp**, (Flaxman, Pryzdatek, STACS 2005)
- $M = \text{poly}(n)$, worst-case dynamic programming in $\mathcal{O}(n^2 M)$ is poly-time,
- $M \leq n^2/\log n$, alg faster than dynamic programming exists.

Dense instances

- The dynamic program a 5th graders would write takes time $\mathcal{O}(n^2 M)$.

Dense instances

- The dynamic program a 5th graders would write takes time $\mathcal{O}(n^2 M)$.
- With more education, you can devise a faster algorithm. The state of the art is time $\mathcal{O}\left(\frac{n^{7/4}}{(\log n)^{3/4}}\right)$

Structure theory of set addition

Faster by considerations like

- How can all the set of sums of 2 numbers be small?

Structure theory of set addition

Faster by considerations like

- How can all the set of sums of 2 numbers be small?

Theorem

Let S be a finite subset of $\mathbb{Z}$, with $|S| = n$ and let $b \leq n$. If

$$|S + S| \leq 2k - 1 + b,$$

then S is contained in an arithmetic progression of length

$$|S| + b.$$

Aside: a puzzle

- Find $S \subseteq \mathbb{Z}^+$ with $|S| = n$ so that

$$\left|\{(s_1, s_2) : s_1, s_2 \in S \text{ and } s_1 + s_2 \text{ is prime}\}\right|$$

Aside: a puzzle

- Find $S \subseteq \mathbb{Z}^+$ with $|S| = n$ so that

$$\left|\{(s_1, s_2) : s_1, s_2 \in S \text{ and } s_1 + s_2 \text{ is prime}\}\right|$$

- Hint:
  - If s_1 and s_2 have the same parity then $s_1 + s_2$ is probably not prime.
  - So

$$\left|\{(s_1, s_2) : s_1 + s_2 \text{ is prime}\}\right| \leq \frac{n^2}{4}.$$

Aside: a puzzle

- Find $S \subseteq \mathbb{Z}^+$ with $|S| = n$ so that

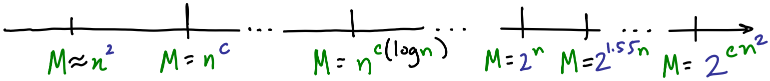
$$\left|\{(s_1, s_2) : s_1, s_2 \in S \text{ and } s_1 + s_2 \text{ is prime}\}\right|$$

- Hint:
  - If s_1 and s_2 have the same parity then $s_1 + s_2$ is probably not prime.
  - So

$$\left|\{(s_1, s_2) : s_1 + s_2 \text{ is prime}\}\right| \leq \frac{n^2}{4}.$$

  - Aim high.

Sketch of computational difficulty as a function of M



- $M \geq 2^{n^2/2}$, a poly-time algorithm using Lovász basis reduction succeeds **whp**,
- $M \geq 2^{1.55n}$, similar algorithms seem to work,
- $M = 2^n$, seems to be "most difficult",
- $M = n^{\mathcal{O}(\log n)}$, poly-time algorithm succeeds **whp**, (Flaxman, Pryzdatek, STACS 2005)
- $M = \text{poly}(n)$, worst-case dynamic programming in $\mathcal{O}(n^2 M)$ is poly-time,
- $M \leq n^2 / \log n$, alg faster than dynamic programming exists.

$M = n^{\mathcal{O}(\log n)}$ — Medium-dense instances

Input: M, $a_1, \ldots, a_n$, and T,

Goal: Find $S \subseteq \{0, 1, \ldots, n\}$ such that $\sum_{i \in S} a_i \equiv T \mod M$.

For simplicity,

- Let M to be a power of 2, roughly $M = 2^{(\log n)^2}$,
- Let $T = 0$.

$M = n^{\mathcal{O}(\log n)}$ — Medium-dense instances

Input: M, $a_1, \dots, a_n$, and T,

Goal: Find $S \subseteq \{0, 1, \dots, n\}$ such that $\sum_{i \in S} a_i \equiv T \mod M$.

For simplicity,

- Let M to be a power of 2, roughly $M = 2^{(\log n)^2}$,
- Let $T = 0$.

My approach is to “zero out” the least significant bits, $(\log n)/2$ at a time.

Medium-dense algorithm execution, $M = 256$, $T = 0$

$a_1 = 35$
$a_2 = 29$
37
$\vdots$ 27
191
29
3
155
147
$a_{10} = 221$

Medium-dense algorithm execution, $M = 256$, $T = 0$

$a_1 = 35 = 0010\ 0011$
$a_2 = 29 = 0001\ 1101$
$37 = \dots\ 0101$
$\vdots \quad 27 = \dots\ 1011$
$191 =$
$29 = \quad \vdots$
$3 =$
$155 =$
$147 =$
$a_{10} = 221 = \dots\ 1101$

Medium-dense algorithm execution, $M = 256$, $T = 0$

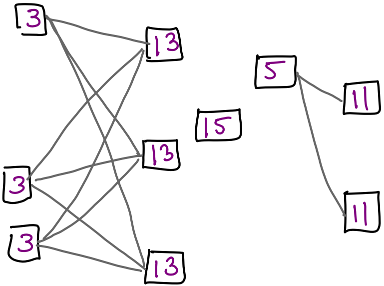
$$
\begin{aligned}
a_1 &= 35 = 0010\ 0011 \equiv 3 \pmod{16} \\
a_2 &= 29 = 0001\ 1101 \equiv 13 \\
&\quad 37 = \ldots\ 0101 \equiv 5 \\
&\quad 27 = \ldots\ 1011 \equiv 11 \\
&\quad 191 = \quad \equiv 15 \\
&\quad 29 = \quad \equiv 13 \\
&\quad 3 = \quad \equiv 3 \\
&\quad 155 = \quad \equiv 11 \\
&\quad 147 = \quad \equiv 3 \\
a_{10} &= 221 = \ldots\ 1101 \equiv 13
\end{aligned}
$$

Medium-dense algorithm execution, $M = 256$, $T = 0$

a_1	$= 35$	$= 0010\ 0011$	$\equiv$	3
a_2	$= 29$	$= 0001\ 1101$	$\equiv$	13
	37	$= \dots 0101$	$\equiv$	5
$\vdots$	27	$= \dots 1011$	$\equiv$	11
	191	$=$	$\equiv$	15
	29	$= \vdots$	$\equiv$	13
	3	$=$	$\equiv$	3
	155	$=$	$\equiv$	11
	147	$=$	$\equiv$	3
a_{10}	$= 221$	$= \dots 1101$	$\equiv$	13

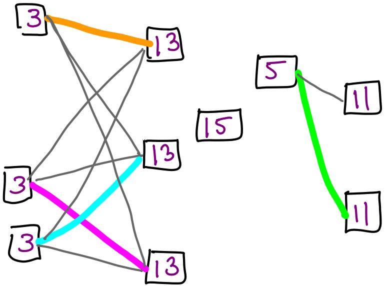
Medium-dense algorithm execution, $M = 256$, $T = 0$

$a_1 = 35 = 0010\ 0011 \equiv$
$a_2 = 29 = 0001\ 1101 \equiv$
$37 = \dots\ 0101 \equiv$
$\vdots\quad 27 = \dots\ 1011 \equiv$
$191 = \quad \vdots \quad \equiv$
$29 = \equiv$
$3 = \equiv$
$155 = \equiv$
$147 = \equiv$
$a_{10} = 221 = \dots\ 1101 \equiv$



Medium-dense algorithm execution, $M = 256$, $T = 0$

$a_1 = 35 = 0010\ 0011 \equiv$
$a_2 = 29 = 0001\ 1101 \equiv$
$37 = \ldots\ 0101 \equiv$
$\vdots\quad 27 = \ldots\ 1011 \equiv$
$191 = \quad \vdots \quad \equiv$
$29 = \equiv$
$3 = \equiv$
$155 = \equiv$
$147 = \equiv$
$a_{10} = 221 = \ldots\ 1101 \equiv$



Medium-dense algorithm execution, $M = 256$, $T = 0$

$a_1 = 35 = 0010\ 0011 \equiv$

$a_2 = 29 = 0001\ 1101 \equiv$

$37 = \ldots\ 0101 \equiv$

$\vdots \quad 27 = \ldots\ 1011 \equiv$

$191 = \quad \equiv$

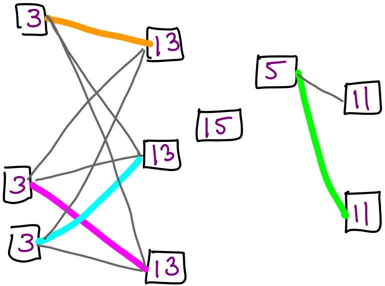
$29 = \quad \vdots \quad \equiv$

$3 = \quad \equiv$

$155 = \quad \equiv$

$147 = \quad \equiv$

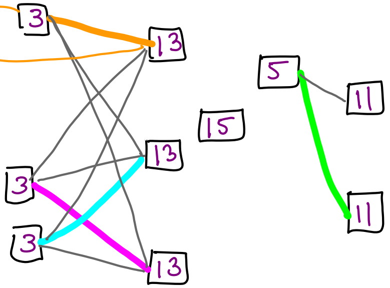
$a_{10} = 221 = \ldots\ 1101 \equiv$



Recurse:

Medium-dense algorithm execution, $M = 256$, $T = 0$

$a_1 = 35 = 0010\ 0011 \equiv$
$a_2 = 29 = 0001\ 1101 \equiv$
$37 = \ldots\ 0101 \equiv$
$\vdots\quad 27 = \ldots\ 1011 \equiv$
$191 = \quad \vdots \quad \equiv$
$29 = \quad \equiv$
$3 = \quad \equiv$
$155 = \quad \equiv$
$147 = \quad \equiv$
$a_{10} = 221 = \ldots\ 1101 \equiv$



Recurse:
$a_1 = 35 + 29 = 0100\ 0000$

Medium-dense algorithm execution, $M = 256$, $T = 0$

$a_1 = 35 = 0010\ 0011 \equiv$

$a_2 = 29 = 0001\ 1101 \equiv$

$37 = \ldots 0101 \equiv$

$\vdots$

$27 = \ldots 1011 \equiv$

$191 = \vdots \equiv$

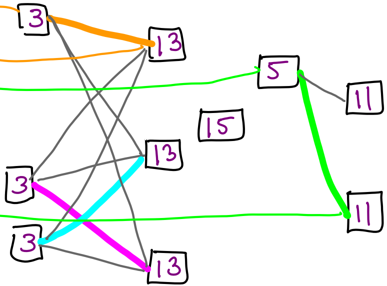
$29 = \equiv$

$3 = \equiv$

$155 = \equiv$

$147 = \equiv$

$a_{10} = 221 = \ldots 1101 \equiv$

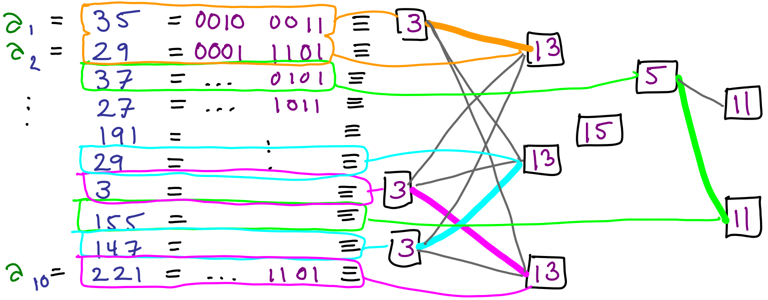


Recurse:

$a_1' = 35 + 29 = 0100\ 0000$

$a_2' = 37 + 155 = 1100\ 0000$

Medium-dense algorithm execution, $M = 256$, $T = 0$



Recurse:

$a'_1 = 35 + 29 = 0100\ 0000$

$a'_2 = 37 + 155 = 1100\ 0000$

$a'_3 = 29 + 147 = 1011\ 0000$

$a'_4 = 3 + 221 = 1110\ 0000$

Medium-dense algorithm execution, $M = 256$, $T = 0$

$a_1 = 35 = 0010\ 0011 \equiv$

$a_2 = 29 = 0001\ 1101 \equiv$

$37 = \ldots\ 0101 \equiv$

$\vdots$ $27 = \ldots\ 1011 \equiv$

$191 = \quad \vdots \quad \equiv$

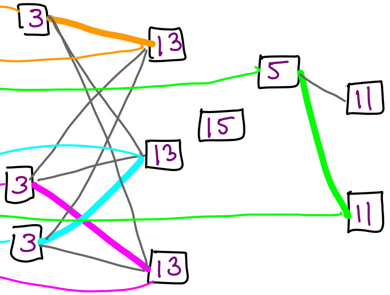
$29 = \equiv$

$3 = \equiv$

$155 = \equiv$

$147 = \equiv$

$a_{10} = 221 = \ldots\ 1101 \equiv$



Recurse:

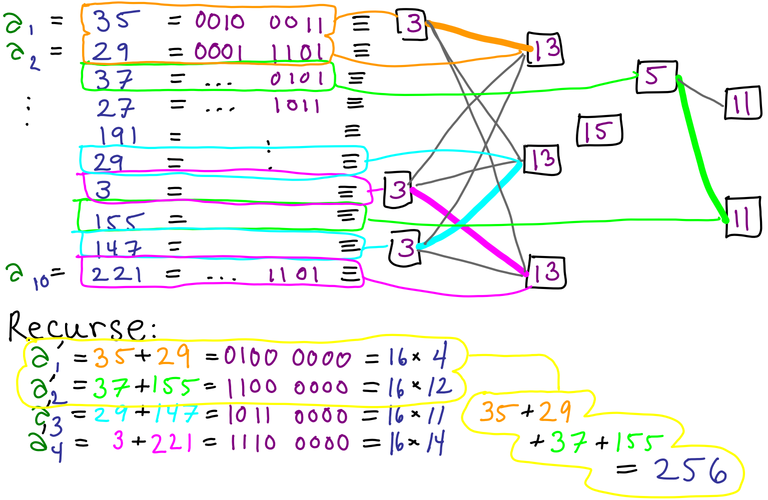
$a'_1 = 35 + 29 = 0100\ 0000 = 16 \times 4$

$a'_2 = 37 + 155 = 1100\ 0000 = 16 \times 12$

$a'_3 = 29 + 147 = 1011\ 0000 = 16 \times 11$

$a'_4 = 3 + 221 = 1110\ 0000 = 16 \times 14$

Medium-dense algorithm execution, $M = 256$, $T = 0$



“Proof” algorithm succeeds **whp**

- Let N_k denote the number of numbers at step k of the recursion.

"Proof" algorithm succeeds **whp**

- Let N_k denote the number of numbers at step k of the recursion.
- Since we pair up numbers and use each only once, $N_{k+1} \leq N_k/2$. If $N_{k+1} \geq N_k/4$ for each k, then we can recurse $(\log n)/2$ times before we run out of numbers.

"Proof" algorithm succeeds **whp**

- Let N_k denote the number of numbers at step k of the recursion.
- Since we pair up numbers and use each only once, $N_{k+1} \leq N_k/2$. If $N_{k+1} \geq N_k/4$ for each k, then we can recurse $(\log n)/2$ times before we run out of numbers.
- To see that it is unlikely that $N_{k+1} \leq N_k/4$,

"Proof" algorithm succeeds **whp**

- Let N_k denote the number of numbers at step k of the recursion.
- Since we pair up numbers and use each only once, $N_{k+1} \leq N_k/2$. If $N_{k+1} \geq N_k/4$ for each k, then we can recurse $(\log n)/2$ times before we run out of numbers.
- To see that it is unlikely that $N_{k+1} \leq N_k/4$,
  - Recursion yields numbers which are uniformly distributed,

“Proof” algorithm succeeds **whp**

- Let N_k denote the number of numbers at step k of the recursion.
- Since we pair up numbers and use each only once, $N_{k+1} \leq N_k/2$. If $N_{k+1} \geq N_k/4$ for each k, then we can recurse $(\log n)/2$ times before we run out of numbers.
- To see that it is unlikely that $N_{k+1} \leq N_k/4$,
  - Recursion yields numbers which are uniformly distributed,
  - $\mathbb{E}\left[N_{k+1} \mid N_k\right] = \frac{N_k}{2} - \mathcal{O}\left({N_k}^{1/2} n^{1/4}\right)$.

"Proof" algorithm succeeds **whp**

- Let N_k denote the number of numbers at step k of the recursion.
- Since we pair up numbers and use each only once, $N_{k+1} \leq N_k/2$. If $N_{k+1} \geq N_k/4$ for each k, then we can recurse $(\log n)/2$ times before we run out of numbers.
- To see that it is unlikely that $N_{k+1} \leq N_k/4$,
  - Recursion yields numbers which are uniformly distributed,
  - $\mathbb{E}\left[N_{k+1} \mid N_k\right] = \frac{N_k}{2} - \mathcal{O}\left({N_k}^{1/2} n^{1/4}\right)$.
  - So, concentration inequalities for martingales show

$$\mathbb{P}\left[N_{k+1} \leq N_k/4\right] \leq \exp\left\{-\frac{n^{3/4}}{32}\right\}.$$

Generalizations

Generalizations

- Target value that is not $T = 0$,
  - Include a special $a_{n+1} = -T$ and make sure that it appears in the solution.

Generalizations

- Target value that is not $T = 0$,
  - Include a special $a_{n+1} = -T$ and make sure that it appears in the solution.
- Modulus M that is odd,

Generalizations

- Target value that is not $T = 0$,
  - Include a special $a_{n+1} = -T$ and make sure that it appears in the solution.
- Modulus M that is odd,
  - Zero out *most* significant bits first

Generalizations

- Target value that is not $T = 0$,
  - Include a special $a_{n+1} = -T$ and make sure that it appears in the solution.
- Modulus M that is odd,
  - Zero out *most* significant bits first
  - Now the numbers in the subinstance are *not* uniformly random

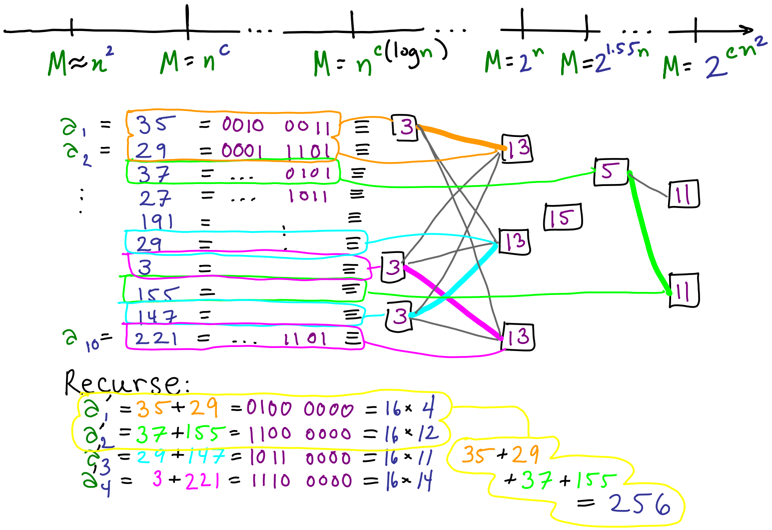
Generalizations

- Target value that is not $T = 0$,
  - Include a special $a_{n+1} = -T$ and make sure that it appears in the solution.
- Modulus M that is odd,
  - Zero out *most* significant bits first
  - Now the numbers in the subinstance are *not* uniformly random
  - But they are distributed symmetrically, which is enough

Generalizations

- Target value that is not $T = 0$,
  - Include a special $a_{n+1} = -T$ and make sure that it appears in the solution.
- Modulus M that is odd,
  - Zero out *most* significant bits first
  - Now the numbers in the subinstance are *not* uniformly random
  - But they are distributed symmetrically, which is enough
- General modulus $M = 2^k \cdot \text{odd}$,
  - First work mod 2^k, then work mod odd.

End of the subset sum section



Minimum Cost Spanning Tree

Input: Graph $G = (V, E)$,
Cost vector $\mathbf{c} \in \mathbb{R}^E$.

Goal: Find spanning tree $T \subseteq E$ such that
$Z = \sum_{e \in T} \mathbf{c}_e$ is minimized.

Random Minimum Cost Spanning Tree

If each $\mathbf{c}_e$ is an independent random variable drawn uniformly from $[0, 1]$, then as $n \to \infty$,

$$\mathbb{E}[Z] \to$$

Random Minimum Cost Spanning Tree

If each $\mathbf{c}_e$ is an independent random variable drawn uniformly from $[0, 1]$, then as $n \to \infty$,

$$\mathbb{E}[Z] \to \zeta(3) = \frac{1}{1^3} + \frac{1}{2^3} + \frac{1}{3^3} + \ldots \approx 1.2025\ldots$$

Proof in one slide

length of tree T

$$\ell(T) = \sum_{e \in T} X_e$$

$$= \sum_{e \in T} \int_0^1 \mathbb{1}_{\{X_e \geq p\}} \, dp$$

$$= \int_0^1 \sum_{e \in T} \mathbb{1}_{\{X_e \geq p\}} \, dp$$

for min. sp. tree

$$= \int_0^1 (K(G_p) - 1) \, dp$$

\# of components

$$E[\ell(T)] = \int_{p=0}^1 E[K(G_p)] \, dp - 1$$

only trees contribute and giants

$$\approx \int_{p=0}^1 \sum \binom{n}{k} k^{k-2} p^{k-1} (1-p)^{kn} \, dp$$

$$\approx \int_{p=0}^1 \sum \frac{n^k}{k!} k^{k-2} p^{k-1} (1-p)^{kn} \, dp$$

$$\approx \sum \frac{n^k}{k!} k^{k-2} \int_0^1 p^{k-1} (1-p)^{kn} \, dp$$

$$\approx \sum \frac{n^k}{k!} k^{k-2} \frac{(k-1)!\,(kn)!}{(k(n+1))!}$$

Beta Integral

$$\approx \sum k^{-3}$$

2-stage Stochastic Minimum Cost Spanning Tree

Input: Cost vector $\mathbf{c}_M \in \mathbb{R}^E$

A distribution over cost vectors $\mathbf{c}_T \in \mathbb{R}^E$

Goal: Find forest $F \subseteq E$ to buy on Monday such that when F is augmented on Tuesday by $F' \subseteq E$ to form a spanning tree,

$$Z = \sum_{e \in F} \mathbf{c}_M(e) + \mathbb{E}\left[\min_{F'} \left\{ \sum_{e \in F'} \mathbf{c}_T(e) : F \cup F' \text{ sp tree} \right\}\right]$$

is minimized.

Random 2-stage Sto. Min. Cost Sp. Tree

So what happens if $\mathbf{c}_M(e)$ and $\mathbf{c}_T(e)$ are independent uniformly random in $[0, 1]$?

(Flaxman, Frieze, Krivelevich, SODA 2005)

Some observations:

- Buying a spanning tree entirely on Monday has cost $\zeta(3)$.
- If you knew the Tuesday costs on Monday, could get away with cost $\zeta(3)/2$.

Random 2-stage Sto. Min. Cost Sp. Tree

The threshold heuristic:

- Pick some threshold value α.
- On Monday, only buy edges with cost less than α.
- On Tuesday, finish the tree.

Best value is $\alpha = \frac{1}{n}$, which yields solution with expected cost

$$E[Z] \to \zeta(3) - \frac{1}{2}.$$

Random 2-stage Sto. Min. Cost Sp. Tree

- Threshold heuristic is not optimal: by looking at the structure of the edges instead of only the cost, you can improve the objective value a little; **whp**

$$Z^\star \leq \zeta(3) - \frac{1}{2} - 10^{-256}.$$

- There is no way to attain $\zeta(3)/2$, because you must make some mistakes on Monday; **whp**

$$Z^\star \geq \zeta(3)/2 + 10^{-5}.$$

End of the Spanning Tree section

length of tree T

$$\ell(T) = \sum_{e \in T} X_e$$

$$= \sum_{e \in T} \int_0^1 \mathbb{1}_{\{X_e \geq p\}} \, dp$$

$$= \int_0^1 \sum_{e \in T} \mathbb{1}_{\{X_e \geq p\}} \, dp$$

for min. sp. tree

$$= \int_0^1 (K(G_p) - 1) \, dp$$

# of components

$$E[\ell(T)] = \int_{p=0}^1 E[K(G_p)] \, dp - 1$$

only trees contribute and giants

$$\approx \int_{p=0}^1 \sum \binom{n}{k} k^{k-2} p^{k-1} (1-p)^{kn} \, dp$$

$$\approx \int_{p=0}^1 \sum \frac{n^k}{k!} k^{k-2} p^{k-1} (1-p)^{kn} \, dp$$

$$\approx \sum \frac{n^k}{k!} k^{k-2} \int_0^1 p^{k-1} (1-p)^{kn} \, dp$$

$$\approx \sum \frac{n^k}{k!} k^{k-2} \frac{(k-1)!\,(kn)!}{(k(n+1))!}$$

Beta Integral

$$\approx \sum k^{-3}$$

Conclusion

- Average-case analysis provides a detailed picture of computational difficulty,
- Can help in the search for the hardest easy problems and the easiest hard problems,
- Even for “easy” problems the average-case has some surprises.