Back to MAXSAT

- In MAXSAT, the ϕ_i's are clauses.
- Hence $p(\phi_i) \geq 1/2$, which happens when ϕ_i contains a single literal.
- And the heuristic becomes a polynomial-time ϵ-approximation algorithm with $\epsilon = 1/2$.[a]
- If the clauses have k *distinct* literals, $p(\phi_i) = 1 - 2^{-k}$.
- And the heuristic becomes a polynomial-time ϵ-approximation algorithm with $\epsilon = 2^{-k}$.
  - This is the best possible for $k \geq 3$ unless P = NP.

[a]Johnson (1974).

MAX CUT Revisited

- The NP-complete MAX CUT seeks to partition the nodes of graph $G = (V, E)$ into $(S, V - S)$ so that there are as many edges as possible between S and $V - S$ (p. 318).
- **Local search** starts from a feasible solution and performs "local" improvements until none are possible.
- Next we present a local search algorithm for MAX CUT.

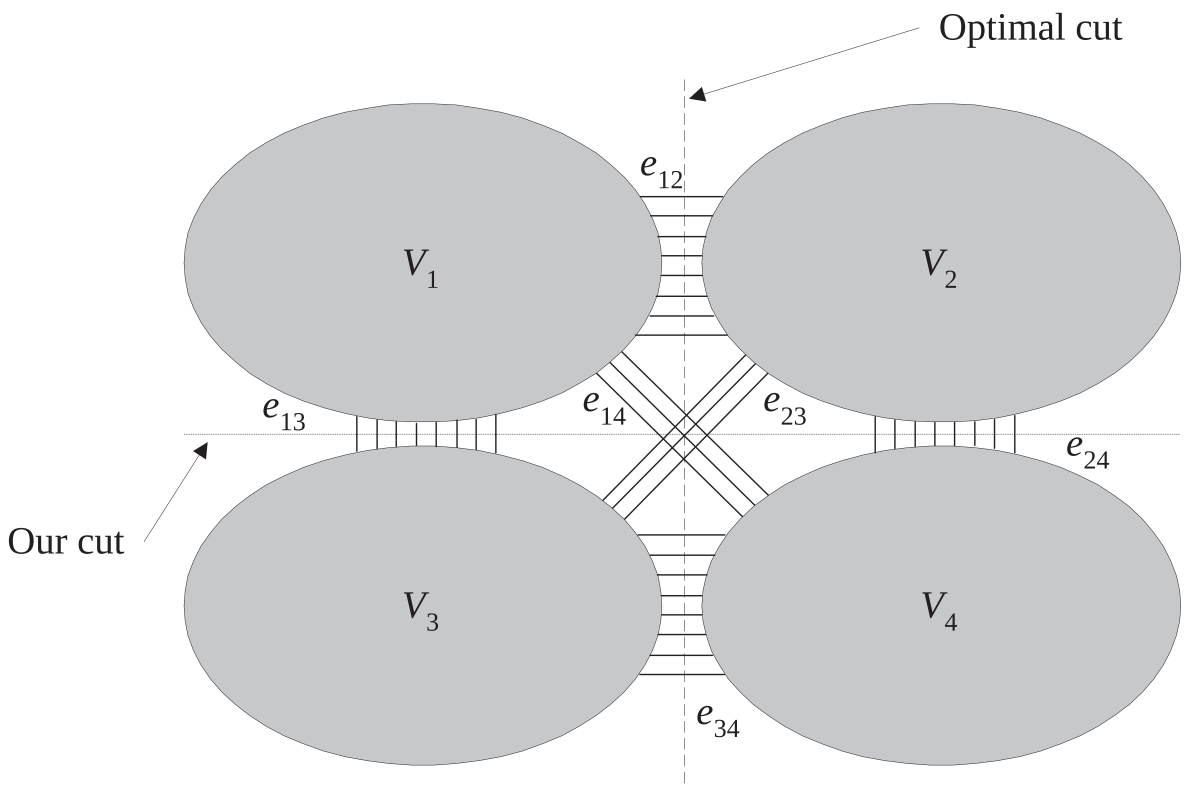
A 0.5-Approximation Algorithm for MAX CUT

$S := \emptyset$;
while $\exists v \in V$ whose switching sides results in a larger cut **do**
  Switch the side of v;
end while
return S;

- A 0.12-approximation algorithm exists.[a]
- 0.059-approximation algorithms do not exist unless NP = ZPP.

[a] Goemans and Williamson (1995).

Analysis



Analysis (continued)

- Partition $V = V_1 \cup V_2 \cup V_3 \cup V_4$, where
  - Our algorithm returns $(V_1 \cup V_2, V_3 \cup V_4)$.
  - The optimum cut is $(V_1 \cup V_3, V_2 \cup V_4)$.
- Let e_{ij} be the number of edges between V_i and V_j.
- For each node $v \in V_1$, its edges to $V_1 \cup V_2$ are outnumbered by those to $V_3 \cup V_4$.
  - Otherwise, v would have been moved to $V_3 \cup V_4$ to improve the cut.

Analysis (continued)

- Considering all nodes in V_1 together, we have $2e_{11} + e_{12} \leq e_{13} + e_{14}$
  - It is $2e_{11}$ is because each edge in V_1 is counted twice.
- The above inequality implies

$$e_{12} \leq e_{13} + e_{14}.$$

Analysis (concluded)

- Similarly,

$$\begin{aligned} e_{12} &\leq e_{23} + e_{24} \\ e_{34} &\leq e_{23} + e_{13} \\ e_{34} &\leq e_{14} + e_{24} \end{aligned}$$

- Add all four inequalities, divide both sides by 2, and add the inequality $e_{14} + e_{23} \leq e_{14} + e_{23} + e_{13} + e_{24}$ to obtain

$$e_{12} + e_{34} + e_{14} + e_{23} \leq 2(e_{13} + e_{14} + e_{23} + e_{24}).$$

- The above says our solution is at least half the optimum.

Approximability, Unapproximability, and Between

- KNAPSACK, NODE COVER, MAXSAT, and MAX CUT have approximation thresholds less than 1.
  - KNAPSACK has a threshold of 0 (p. 664).
  - But NODE COVER and MAXSAT have a threshold larger than 0.
- The situation is maximally pessimistic for TSP: It cannot be approximated unless P = NP (p. 662).
  - The approximation threshold of TSP is 1.
    - The threshold is 1/3 if the TSP satisfies the triangular inequality.
  - The same holds for INDEPENDENT SET.

Unapproximability of TSP[a]

Theorem 77 *The approximation threshold of* TSP *is 1 unless $P = NP$.*

- Suppose there is a polynomial-time ϵ-approximation algorithm for TSP for some $\epsilon < 1$.
- We shall construct a polynomial-time algorithm for the NP-complete HAMILTONIAN CYCLE.
- Given any graph $G = (V, E)$, construct a TSP with $|\, V \,|$ cities with distances

$$d_{ij} = \begin{cases} 1, & \text{if } \{\, i, j \,\} \in E \\ \frac{|V|}{1-\epsilon}, & \text{otherwise} \end{cases}$$

[a] Sahni and Gonzales (1976).

The Proof (concluded)

- Run the alleged approximation algorithm on this TSP.
- Suppose a tour of cost $|V|$ is returned.
  - This tour must be a Hamiltonian cycle.
- Suppose a tour with at least one edge of length $\frac{|V|}{1-\epsilon}$ is returned.
  - The total length of this tour is $> \frac{|V|}{1-\epsilon}$.
  - Because the algorithm is ϵ-approximate, the optimum is at least $1-\epsilon$ times the returned tour's length.
  - The optimum tour has a cost exceeding $|V|$.
  - Hence G has no Hamiltonian cycles.

KNAPSACK Has an Approximation Threshold of Zero[a]

Theorem 78 *For any ϵ, there is a polynomial-time ϵ-approximation algorithm for* KNAPSACK.

- We have n weights $w_1, w_2, \ldots, w_n \in \mathbb{Z}^+$, a weight limit W, and n values $v_1, v_2, \ldots, v_n \in \mathbb{Z}^+$.[b]
- We must find an $S \subseteq \{1, 2, \ldots, n\}$ such that $\sum_{i \in S} w_i \leq W$ and $\sum_{i \in S} v_i$ is the largest possible.

---

[a]Ibarra and Kim (1975).

[b]If the values are fractional, the result is slightly messier but the main conclusion remains correct. Contributed by Mr. Jr-Ben Tian (R92922045) on December 29, 2004.

The Proof (continued)

- Let
$$V = \max\{v_1, v_2, \ldots, v_n\}.$$
- Clearly, $\sum_{i \in S} v_i \leq nV$.
- Let $0 \leq i \leq n$ and $0 \leq v \leq nV$.
- $W(i, v)$ is the minimum weight attainable by selecting some of the first i items with a total value of v.
- Set $W(0, v) = \infty$ for $v \in \{ 1, 2, \ldots, nV \}$ and $W(i, 0) = 0$ for $i = 0, 1, \ldots, n$.[a]

[a]Contributed by Mr. Ren-Shuo Liu (`D98922016`) and Mr. Yen-Wei Wu (`D98922013`) on December 28, 2009.

The Proof (continued)

- Then, for $0 \leq i < n$,

$$W(i+1, v) = \min\{W(i, v), W(i, v - v_{i+1}) + w_{i+1}\}.$$

- Finally, pick the largest v such that $W(n, v) \leq W$.
- The running time is $O(n^2 V)$, not polynomial time.
- Key idea: Limit the number of precision bits.

The Proof (continued)

- Define
$$v_i' = 2^b \left\lfloor \frac{v_i}{2^b} \right\rfloor.$$
  - This is equivalent to zeroing each v_i's last b bits.
- From the original instance
$$x = (w_1, \ldots, w_n, W, v_1, \ldots, v_n),$$
define the approximate instance
$$x' = (w_1, \ldots, w_n, W, v_1', \ldots, v_n').$$

The Proof (continued)

- Solving x' takes time $O(n^2 V / 2^b)$.
  - The algorithm only performs subtractions on the v_i-related values.
  - So the b last bits can be *removed* from the calculations.
  - That is, use $v_i' = \left\lfloor \frac{v_i}{2^b} \right\rfloor$ in the calculations.
  - Then multiply the returned value by 2^b.
- The solution S' is close to the optimum solution S:

$$\sum_{i \in S'} v_i \geq \sum_{i \in S'} v_i' \geq \sum_{i \in S} v_i' \geq \sum_{i \in S} (v_i - 2^b) \geq \sum_{i \in S} v_i - n2^b.$$

The Proof (continued)

- Hence
$$\sum_{i \in S'} v_i \geq \sum_{i \in S} v_i - n2^b.$$
- Without loss of generality, assume $w_i \leq W$ for all i.
  - Otherwise, item i is redundant.
- V is a lower bound on OPT.
  - Picking an item with value V is a legitimate choice.
- The relative error from the optimum is $\leq n2^b/V$:
$$\frac{\sum_{i \in S} v_i - \sum_{i \in S'} v_i}{\sum_{i \in S} v_i} \leq \frac{\sum_{i \in S} v_i - \sum_{i \in S'} v_i}{V} \leq \frac{n2^b}{V}.$$

The Proof (concluded)

- Suppose we pick $b = \lfloor \log_2 \frac{\epsilon V}{n} \rfloor$.
- The algorithm becomes ϵ-approximate (see Eq. (10) on p. 640).
- The running time is then $O(n^2 V / 2^b) = O(n^3/\epsilon)$, a polynomial in n and $1/\epsilon$.[a]

---

[a]It hence depends on the *value* of $1/\epsilon$. Thanks to a lively class discussion on December 20, 2006. If we fix ϵ and let the problem size increase, then the complexity is cubic. Contributed by Mr. Ren-Shan Luoh (`D97922014`) on December 23, 2008.

Pseudo-Polynomial-Time Algorithms

- Consider problems with inputs that consist of a collection of integer parameters (TSP, KNAPSACK, etc.).
- An algorithm for such a problem whose running time is a polynomial of the input length and the *value* (not length) of the largest integer parameter is a **pseudo-polynomial-time algorithm**.[a]
- On p. 666, we presented a pseudo-polynomial-time algorithm for KNAPSACK that runs in time $O(n^2V)$.
- How about TSP (D), another NP-complete problem?

---

[a] Garey and Johnson (1978).

No Pseudo-Polynomial-Time Algorithms for TSP (D)

- By definition, a pseudo-polynomial-time algorithm becomes polynomial-time if each integer parameter is limited to having a *value* polynomial in the input length.
- Corollary 42 (p. 335) showed that HAMILTONIAN PATH is reducible to TSP (D) with weights 1 and 2.
- As HAMILTONIAN PATH is NP-complete, TSP (D) cannot have pseudo-polynomial-time algorithms unless P = NP.
- TSP (D) is said to be **strongly NP-hard**.
- Many weighted versions of NP-complete problems are strongly NP-hard.

Polynomial-Time Approximation Scheme

- Algorithm M is a **polynomial-time approximation scheme** (**PTAS**) for a problem if:
  - For each $\epsilon > 0$ and instance x of the problem, M runs in time polynomial (depending on ϵ) in $|\,x\,|$.
    * Think of ϵ as a constant.
  - M is an ϵ-approximation algorithm for every $\epsilon > 0$.

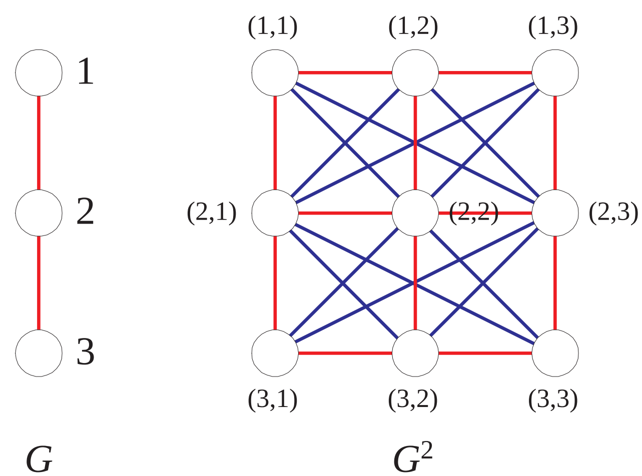
Fully Polynomial-Time Approximation Scheme

- A polynomial-time approximation scheme is **fully polynomial** (**FPTAS**) if the running time depends polynomially on $|x|$ and $1/\epsilon$.
  - Maybe the best result for a "hard" problem.
  - For instance, KNAPSACK is fully polynomial with a running time of $O(n^3/\epsilon)$ (p. 664).

Square of G

- Let $G = (V, E)$ be an undirected graph.
- G^2 has nodes $\{(v_1, v_2) : v_1, v_2 \in V\}$ and edges

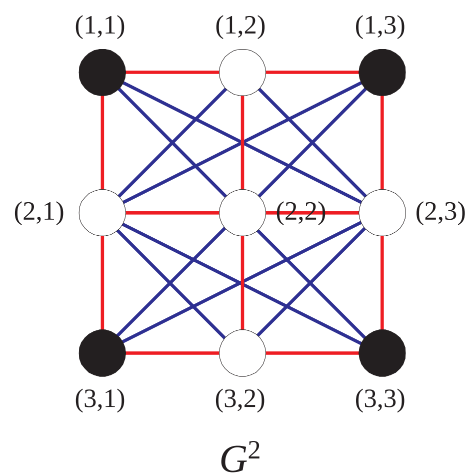
$$\{\{(u, u'), (v, v')\} : (u = v \land \{u', v'\} \in E) \lor \{u, v\} \in E\}.$$



Independent Sets of G and G^2

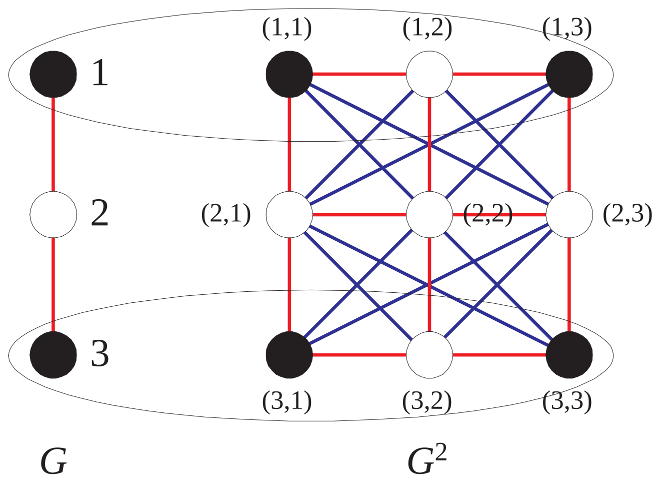
Lemma 79 *$G(V, E)$ has an independent set of size k if and only if G^2 has an independent set of size k^2.*

- Suppose G has an independent set $I \subseteq V$ of size k.
- $\{(u, v) : u, v \in I\}$ is an independent set of size k^2 of G^2.



The Proof (continued)

- Suppose G^2 has an independent set I^2 of size k^2.
- $U \equiv \{u : \exists v \in V\ (u, v) \in I^2\}$ is an independent set of G.



- $|\,U\,|$ is the number of "rows" that the nodes in I^2 occupy.

The Proof (concluded)[a]

- If $|U| \geq k$, then we are done.
- Now assume $|U| < k$.
- As the k^2 nodes in I^2 cover fewer than k "rows," there must be a "row" in possession of $> k$ nodes of I^2.
- Those $> k$ nodes will be independent in G as each "row" is a copy of G.

[a] Thanks to a lively class discussion on December 29, 2004.

Approximability of INDEPENDENT SET

- The approximation threshold of the maximum independent set is either zero or one (it is one!).

Theorem 80 *If there is a polynomial-time ϵ-approximation algorithm for* INDEPENDENT SET *for* any $0 < \epsilon < 1$, *then there is a polynomial-time approximation scheme.*

- Let G be a graph with a maximum independent set of size k.
- Suppose there is an $O(n^i)$-time ϵ-approximation algorithm for INDEPENDENT SET.
- We seek a polynomial-time ϵ'-approximation algorithm with $\epsilon' < \epsilon$.

The Proof (continued)

- By Lemma 79 (p. 676), the maximum independent set of G^2 has size k^2.
- Apply the algorithm to G^2.
- The running time is $O(n^{2i})$.
- The resulting independent set has size $\geq (1-\epsilon)\, k^2$.
- By the construction in Lemma 79 (p. 676), we can obtain an independent set of size $\geq \sqrt{(1-\epsilon)\, k^2}$ for G.
- Hence there is a $(1-\sqrt{1-\epsilon})$-approximation algorithm for INDEPENDENT SET by Eq. (11) on p. 641.

The Proof (concluded)

- In general, we can apply the algorithm to G^{2^ℓ} to obtain an $(1-(1-\epsilon)^{2^{-\ell}})$-approximation algorithm for INDEPENDENT SET.
- The running time is $n^{2^\ell i}$.[a]
- Now pick $\ell = \lceil \log \frac{\log(1-\epsilon)}{\log(1-\epsilon')} \rceil$.
- The running time becomes $n^{i \frac{\log(1-\epsilon)}{\log(1-\epsilon')}}$.
- It is an ϵ'-approximation algorithm for INDEPENDENT SET.

[a] It is not fully polynomial.

Comments

- INDEPENDENT SET and NODE COVER are reducible to each other (Corollary 39, p. 312).
- NODE COVER has an approximation threshold at most 0.5 (p. 646).
- But INDEPENDENT SET is unapproximable (see the textbook).
- INDEPENDENT SET limited to graphs with degree $\leq k$ is called k-DEGREE INDEPENDENT SET.
- k-DEGREE INDEPENDENT SET is approximable (see the textbook).

On P vs. NP

Density[a]

The **density** of language $L \subseteq \Sigma^*$ is defined as

$$\text{dens}_L(n) = |\{x \in L : |\,x\,| \leq n\}|.$$

- If $L = \{0,1\}^*$, then $\text{dens}_L(n) = 2^{n+1} - 1$.
- So the density function grows at most exponentially.
- For a unary language $L \subseteq \{0\}^*$,

$$\text{dens}_L(n) \leq n + 1.$$

  – Because $L \subseteq \{\epsilon, 0, 00, \ldots, \overbrace{00 \cdots 0}^{n}, \ldots\}$.

[a] Berman and Hartmanis (1977).

Sparsity

- **Sparse languages** are languages with polynomially bounded density functions.
- **Dense languages** are languages with superpolynomial density functions.

Self-Reducibility for SAT

- An algorithm exhibits **self-reducibility** if it finds a certificate by exploiting algorithms for the *decision* version of the same problem.
- Let ϕ be a boolean expression in n variables $x_1, x_2, \ldots, x_n$.
- $t \in \{0, 1\}^j$ is a **partial** truth assignment for $x_1, x_2, \ldots, x_j$.
- $\phi[\,t\,]$ denotes the expression after substituting the truth values of t for $x_1, x_2, \ldots, x_{|\,t\,|}$ in ϕ.

An Algorithm for SAT with Self-Reduction

We call the algorithm below with empty t.

```
if |t| = n then
    return φ[t];
else
    return φ[t0] ∨ φ[t1];
end if
```

The above algorithm runs in exponential time, by visiting all the partial assignments (or nodes on a depth-n binary tree).

NP-Completeness and Density[a]

Theorem 81 *If a unary language $U \subseteq \{0\}^*$ is NP-complete, then $P = NP$.*

- Suppose there is a reduction R from SAT to U.
- We use R to find a truth assignment that satisfies boolean expression ϕ with n variables if it is satisfiable.
- Specifically, we use R to prune the exponential-time exhaustive search on p. 687.
- The trick is to keep the already discovered results $\phi[\,t\,]$ in a table H.

[a]Berman (1978).

if $|\,t\,| = n$ **then**
    return $\phi[\,t\,]$;
else
    if $(R(\phi[\,t\,]), v)$ is in table H **then**
        return v;
    else
        if $\phi[\,t0\,] =$ "satisfiable" or $\phi[\,t1\,] =$ "satisfiable" **then**
            Insert $(R(\phi[\,t\,])$, "satisfiable") into H;
            return "satisfiable";
        else
            Insert $(R(\phi[\,t\,])$, "unsatisfiable") into H;
            return "unsatisfiable";
        end if
    end if
end if

The Proof (continued)

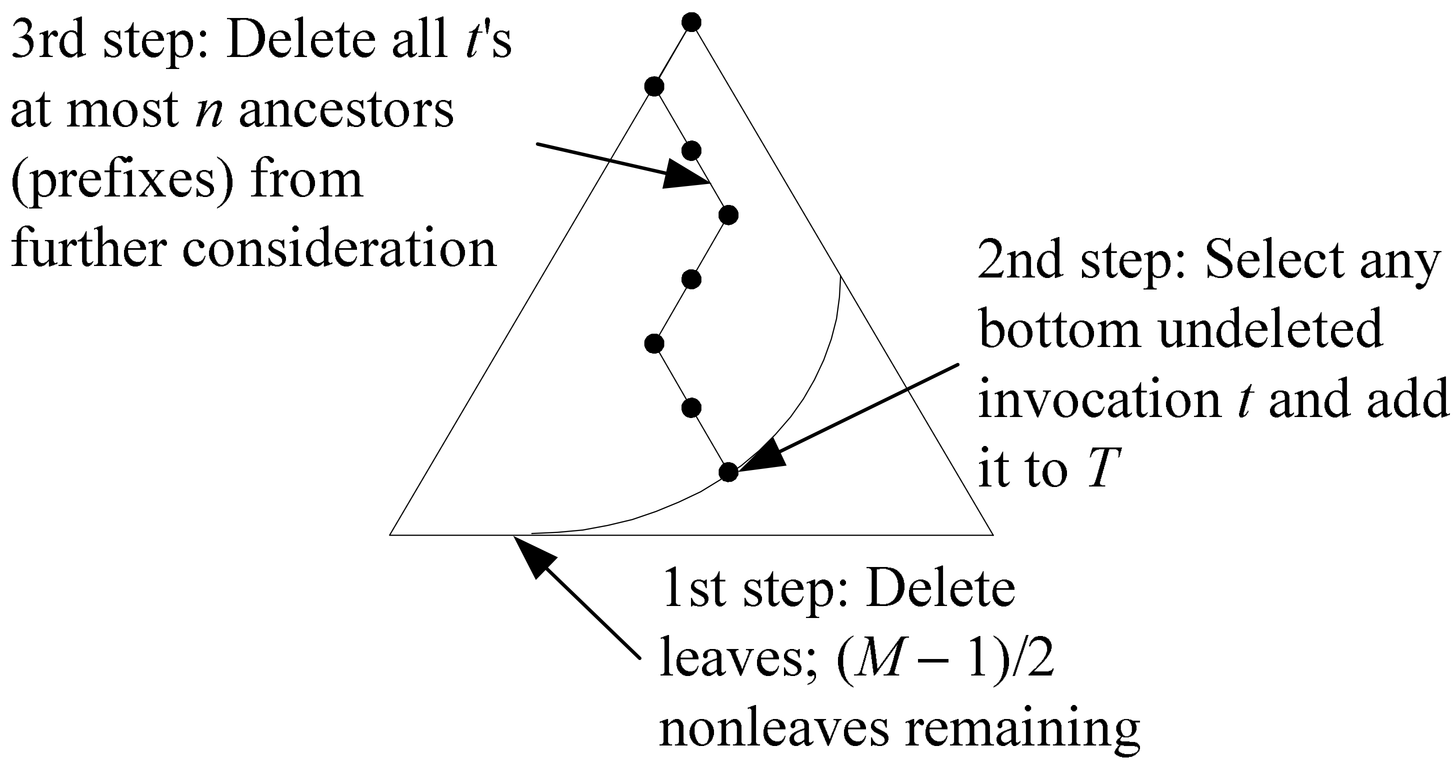
- Since R is a reduction, $R(\phi[\,t\,]) = R(\phi[\,t'\,])$ implies that $\phi[\,t\,]$ and $\phi[\,t'\,]$ must be both satisfiable or unsatisfiable.
- $R(\phi[\,t\,])$ has polynomial length $\leq p(n)$ because R runs in log space.
- As R maps to unary numbers, there are only polynomially many $p(n)$ values of $R(\phi[\,t\,])$.
- How many nodes of the complete binary tree (of invocations/truth assignments) need to be visited?
- If that number is a polynomial, the overall algorithm runs in polynomial time and we are done.

The Proof (continued)

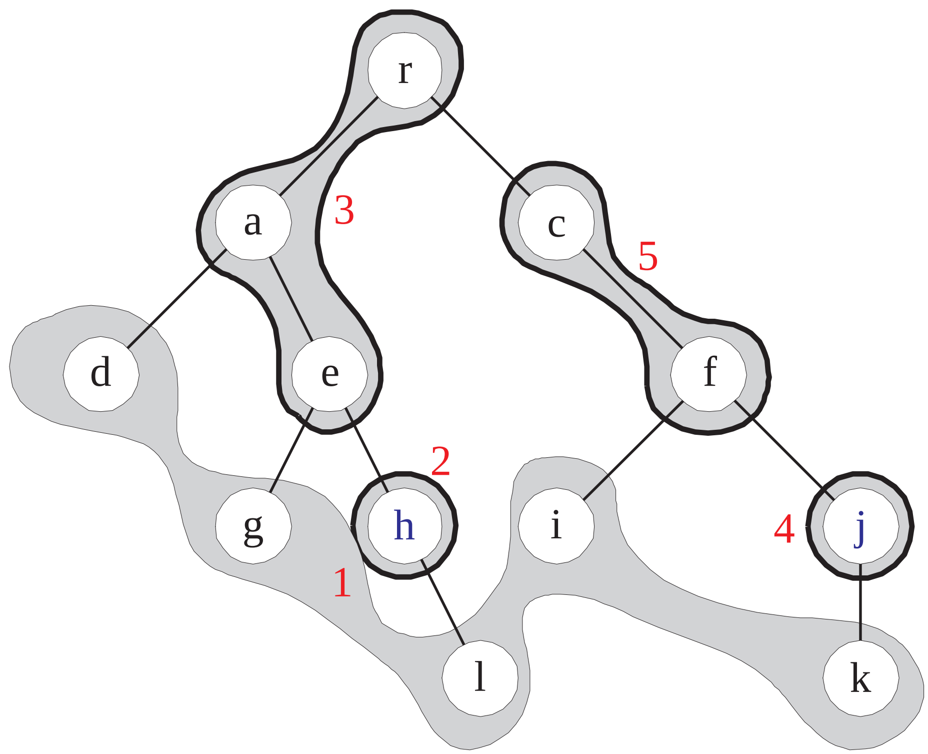
- A search of the table takes time $O(p(n))$ in the random access memory model.
- The running time is $O(Mp(n))$, where M is the total number of invocations of the algorithm.
- The invocations of the algorithm form a binary tree of depth at most n.

The Proof (continued)

- There is a set $T = \{t_1, t_2, \ldots\}$ of invocations (partial truth assignments, i.e.) such that:
  1. $|T| \geq (M-1)/(2n)$.
  2. All invocations in T are recursive (nonleaves).
  3. None of the elements of T is a prefix of another.


3rd step: Delete all t's at most n ancestors (prefixes) from further consideration
2nd step: Select any bottom undeleted invocation t and add it to T
1st step: Delete leaves; (M – 1)/2 nonleaves remaining

An Example



$T = \{\, \mathrm{h}, \mathrm{j} \,\}$.

The Proof (continued)

- All invocations $t \in T$ have different $R(\phi[\,t\,])$ values.
  - None of $h, j \in T$ is a prefix of the other.
  - The invocation of one started after the invocation of the other had terminated.
  - If they had the same value, the one that was invoked second would have looked it up, and therefore would not be recursive, a contradiction.
- The existence of T implies that there are at least $(M-1)/(2n)$ different $R(\phi[\,t\,])$ values in the table.

The Proof (concluded)

- We already know that there are at most $p(n)$ such values.
- Hence $(M-1)/(2n) \leq p(n)$.
- Thus $M \leq 2np(n) + 1$.
- The running time is therefore $O(Mp(n)) = O(np^2(n))$.
- We comment that this theorem holds for any sparse language, not just unary ones.[a]

[a] Mahaney (1980).

coNP-Completeness and Density

Theorem 82 (Fortung (1979)) *If a unary language $U \subseteq \{0\}^*$ is coNP-complete, then P = NP.*

- Suppose there is a reduction R from SAT COMPLEMENT to U.
- The rest of the proof is basically identical except that, now, we want to make sure a formula is unsatisfiable.

Finis