NODE COVER

- NODE COVER seeks the smallest $C \subseteq V$ in graph $G = (V, E)$ such that for each edge in E, at least one of its endpoints is in C.
- A heuristic to obtain a good node cover is to iteratively move a node with the *highest degree* to the cover.
- This turns out to produce an approximation ratio of[a]

$$\frac{c(M(x))}{\text{OPT}(x)} = \Theta(\log n).$$

- So it is not an ϵ-approximation algorithm for any constant $\epsilon < 1$ according to Eq. (19).

---

[a]Chvátal (1979).

A 0.5-Approximation Algorithm[a]

$C := \emptyset$;
while $E \neq \emptyset$ **do**
    Delete an arbitrary edge $\{ u, v \}$ from E;
    Add u and v to C; {Add 2 nodes to C each time.}
    Delete edges incident with u or v from E;
end while
return C;

[a] Johnson (1974).

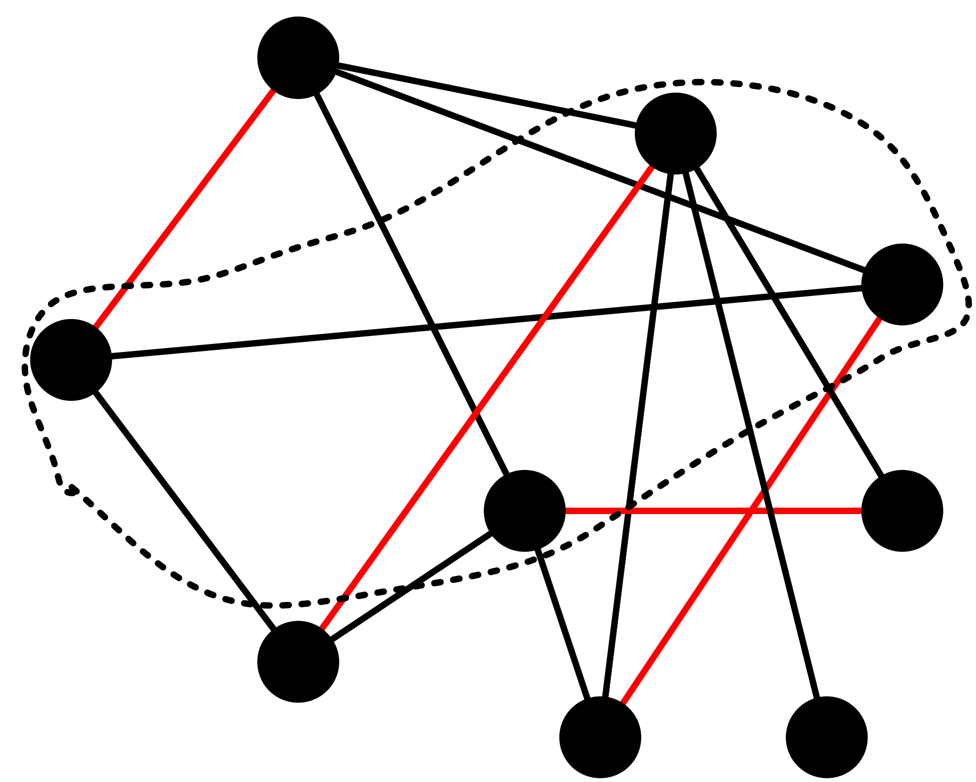
Analysis

- It is easy to see that C is a node cover.
- C contains $|C|/2$ edges.[a]
- No two edges of C share a node.[b]
- *Any* node cover must contain at least one node from each of these edges.
  - If there is an edge in C both of whose ends are outside the cover, then that cover will not be valid.

---

[a] The edges deleted in Line 3.

[b] In fact, C as a set of edges is a *maximal* matching.



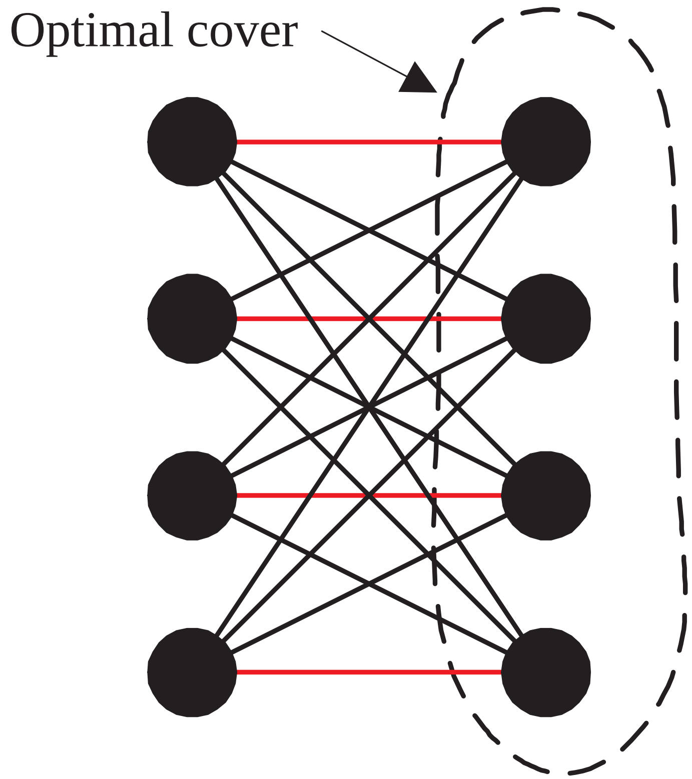
Analysis (concluded)

- This means that $\text{OPT}(G) \geq |C|/2$.
- The approximation ratio is hence

$$\frac{|C|}{\text{OPT}(G)} \leq 2.$$

- So we have a 0.5-approximation algorithm.
- And the approximation threshold is therefore ≤ 0.5.

The 0.5 Bound Is Tight for the Algorithm[a]



[a]Contributed by Mr. Jenq-Chung Li (R92922087) on December 20, 2003. Recall that König's theorem says the size of a maximum matching equals that of a minimum node cover in a bipartite graph.

Remarks

- The approximation threshold is at least[a]

$$1 - \left(10\sqrt{5} - 21\right)^{-1} \approx 0.2651.$$

- The approximation threshold is 0.5 if one assumes the unique games conjecture.[b]
- This ratio 0.5 is also the lower bound for any "greedy" algorithms.[c]

---

[a] Dinur and Safra (2002).
[b] Khot and Regev (2008).
[c] Davis and Impagliazzo (2004).

Maximum Satisfiability

- Given a set of clauses, MAXSAT seeks the truth assignment that satisfies the most.
- MAX2SAT is already NP-complete (p. 347), so MAXSAT is NP-complete.
- Consider the more general k-MAXGSAT for constant k.
  - Let $\Phi = \{\phi_1, \phi_2, \ldots, \phi_m\}$ be a set of boolean expressions in n variables.
  - Each ϕ_i is a *general* expression involving up to k variables.
  - k-MAXGSAT seeks the truth assignment that satisfies the most expressions.

A Probabilistic Interpretation of an Algorithm

- Let ϕ_i involve $k_i \leq k$ variables and be satisfied by s_i of the 2^{k_i} truth assignments.
- A random truth assignment $\in \{ 0, 1 \}^n$ satisfies ϕ_i with probability $p(\phi_i) = s_i / 2^{k_i}$.
  - $p(\phi_i)$ is easy to calculate as k is a constant.
- Hence a random truth assignment satisfies an average of

$$p(\Phi) = \sum_{i=1}^{m} p(\phi_i)$$

expressions ϕ_i.

The Search Procedure

- Clearly

$$p(\Phi) = \frac{1}{2}\,\{\,p(\Phi[\,x_1 = \texttt{true}\,]) + p(\Phi[\,x_1 = \texttt{false}\,])\,\}.$$

- Select the $t_1 \in \{\texttt{true}, \texttt{false}\}$ such that $p(\Phi[\,x_1 = t_1\,])$ is the larger one.
- Note that $p(\Phi[\,x_1 = t_1\,]) \geq p(\Phi)$.
- Repeat the procedure with expression $\Phi[\,x_1 = t_1\,]$ until all variables x_i have been given truth values t_i and all ϕ_i are either true or false.

The Search Procedure (continued)

- By our hill-climbing procedure,

$$\begin{aligned} & p(\Phi) \\ \leq\ & p(\Phi[\, x_1 = t_1 \,]) \\ \leq\ & p(\Phi[\, x_1 = t_1, x_2 = t_2 \,]) \\ \leq\ & \cdots \\ \leq\ & p(\Phi[\, x_1 = t_1, x_2 = t_2, \ldots, x_n = t_n \,]). \end{aligned}$$

- So at least $p(\Phi)$ expressions are satisfied by truth assignment $(t_1, t_2, \ldots, t_n)$.

The Search Procedure (concluded)

- Note that the algorithm is *deterministic*!
- It is called **the method of conditional expectations**.[a]

[a] Erdős and Selfridge (1973); Spencer (1987).

Approximation Analysis

- The optimum is at most the number of satisfiable ϕ_i—i.e., those with $p(\phi_i) > 0$.
- Hence the ratio of algorithm's output vs. the optimum is[a]

$$\geq \frac{p(\Phi)}{\sum_{p(\phi_i)>0} 1} = \frac{\sum_i p(\phi_i)}{\sum_{p(\phi_i)>0} 1} \geq \min_{p(\phi_i)>0} p(\phi_i).$$

- So this is a polynomial-time ϵ-approximation algorithm with $\epsilon = 1 - \min_{p(\phi_i)>0} p(\phi_i)$.
- Because $p(\phi_i) \geq 2^{-k}$ for a satisfiable ϕ_i, the heuristic is a polynomial-time ϵ-approximation algorithm with $\epsilon = 1 - 2^{-k}$.

[a] Recall that $\sum_i a_i / \sum_i b_i \geq \min_i (a_i / b_i)$.

Back to MAXSAT

- In MAXSAT, the ϕ_i's are clauses (like $x \vee y \vee \neg z$).
- Hence $p(\phi_i) \geq 1/2$, which happens when ϕ_i contains a single literal.
- The heuristic becomes a polynomial-time ϵ-approximation algorithm with $\epsilon = 1/2$.[a]
- Suppose we set each boolean variable to true with probability $(\sqrt{5} - 1)/2$, the golden ratio.
- Then follow through the method of conditional expectations to derandomize it.

[a] Johnson (1974).

Back to MAXSAT (concluded)

- We will obtain a $[(3-\sqrt{5})]/2$-approximation algorithm.[a]
  - Note $[(3-\sqrt{5})]/2 \approx 0.382$.
- If the clauses have k *distinct* literals,

$$p(\phi_i) = 1 - 2^{-k}.$$

- 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] Lieberherr and Specker (1981).

MAX CUT Revisited

- 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 $V - S$.
- It is NP-complete.[a]
- **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] Recall p. 378.

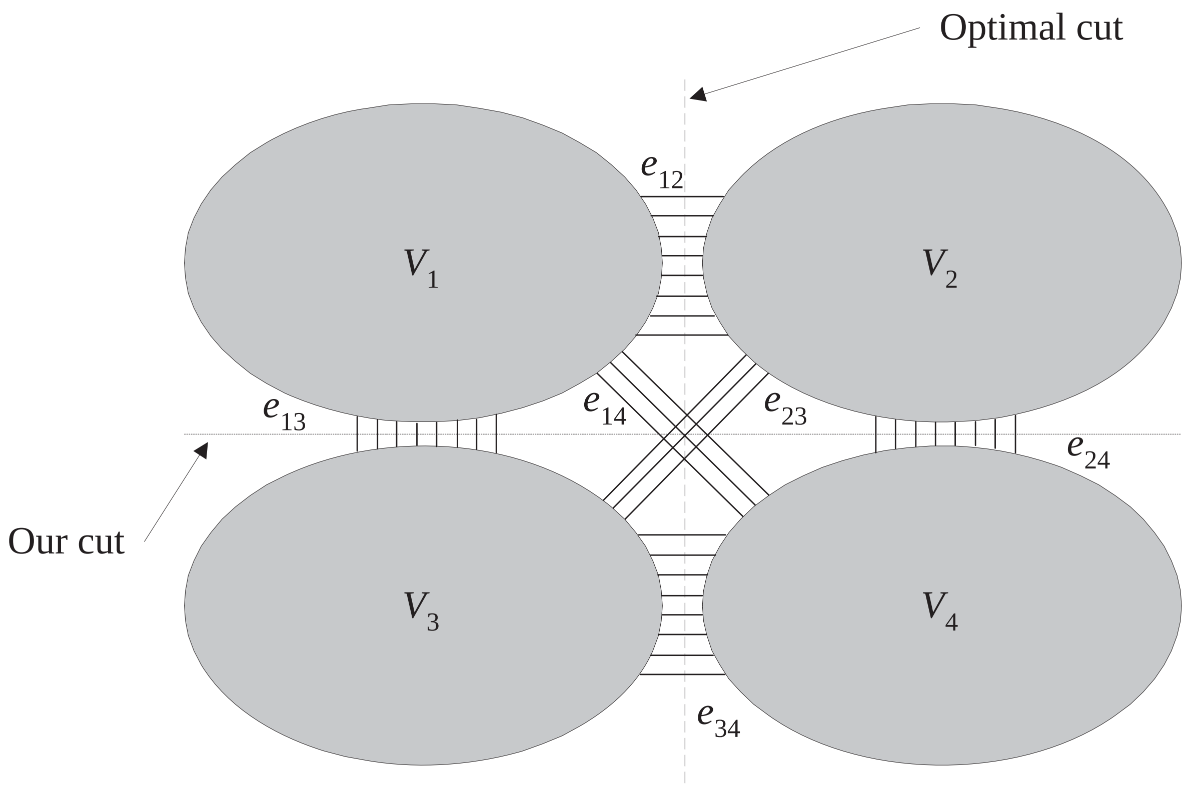
A 0.5-Approximation Algorithm for MAX CUT

$S := \emptyset$;
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.
- Our algorithm returns a cut of size

$$e_{13} + e_{14} + e_{23} + e_{24}.$$

- The optimum cut size is

$$e_{12} + e_{34} + e_{14} + e_{23}.$$

Analysis (continued)

- 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.
- Considering all nodes in V_1 together, we have

$$2e_{11} + e_{12} \leq e_{13} + e_{14}.$$

  - $2e_{11}$, 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. 745).
  - But NODE COVER (p. 725) and MAXSAT have a threshold larger than 0.
- The situation is maximally pessimistic for TSP, which cannot be approximated (p. 743).
  - The approximation threshold of TSP is 1.
    * The threshold is 1/3 if TSP satisfies the triangular inequality.
  - The same holds for INDEPENDENT SET (see the textbook).

Unapproximability of TSP[a]

Theorem 85 *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 to solve 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 that includes an 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 86 *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 $I \subseteq \{1, 2, \ldots, n\}$ such that $\sum_{i \in I} w_i \leq W$ and $\sum_{i \in I} v_i$ 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 (`B89902011`, `R93922045`) on December 29, 2004.

The Proof (continued)

- Let
$$V = \max\{v_1, v_2, \ldots, v_n\}.$$
- Clearly, $\sum_{i \in I} v_i \leq nV$.
- Let $0 \leq i \leq n$ and $0 \leq v \leq nV$.
- $W(i, v)$ is the minimum weight attainable by selecting only from the first i items and with a total value of v.
  - It is an $(n+1) \times (nV+1)$ table.

The Proof (continued)

- Set $W(0, v) = \infty$ for $v \in \{ 1, 2, \ldots, nV \}$ and $W(i, 0) = 0$ for $i = 0, 1, \ldots, n$.[a]
- Then, for $0 \leq i < n$,

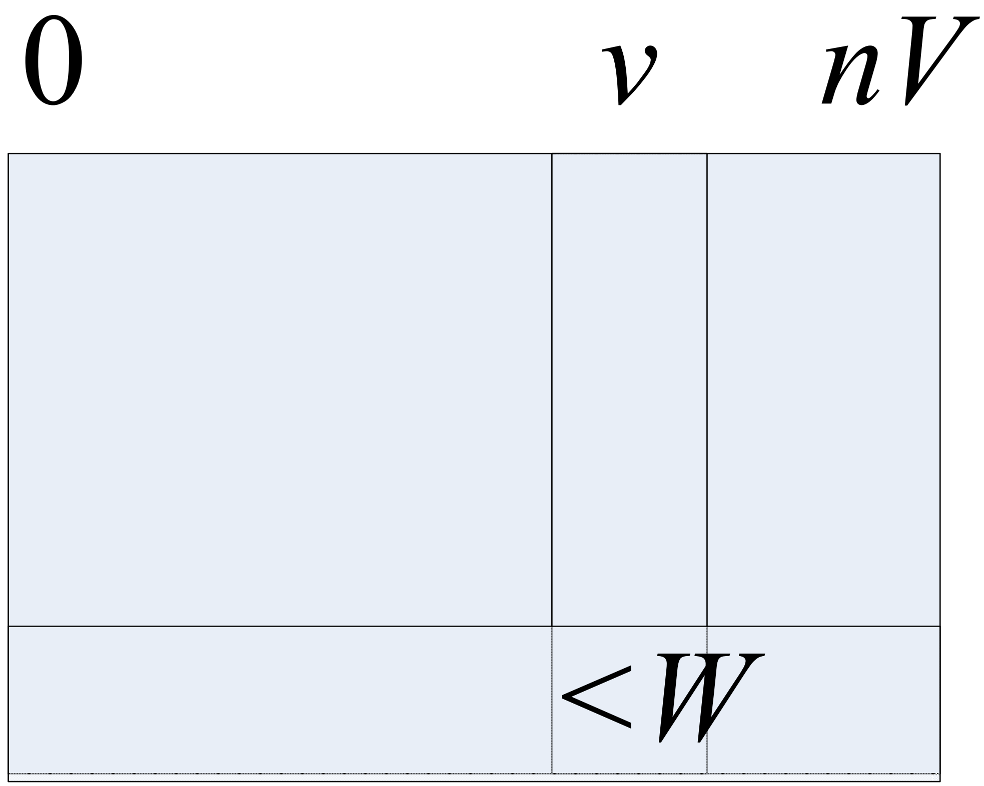
$$W(i + 1, v) = \min\{W(i, v), W(i, v - v_{i+1}) + w_{i+1}\}.$$

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

---

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

[b]Lawler (1979).


0
v
nV
<W

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.
- Call the original instance
$$x = (w_1, \ldots, w_n, W, v_1, \ldots, v_n).$$
- Call 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 $v_i'' = \left\lfloor \frac{v_i}{2^b} \right\rfloor$ and $V'' = \max(v_1'', v_2'', \ldots, v_n'')$ in dynamic programming.
  - It is now an $(n+1) \times (nV+1)/2^b$ table.
  - Then multiply the returned value by 2^b.
- The selection I' is optimal for x'.

The Proof (continued)

- The selection I' is close to the optimal selection I, for x:

$$\sum_{i \in I'} v_i \geq \sum_{i \in I'} v'_i \geq \sum_{i \in I} v'_i \geq \sum_{i \in I} (v_i - 2^b) \geq \left(\sum_{i \in I} v_i \right) - n2^b.$$

- Hence

$$\sum_{i \in I'} v_i \geq \left(\sum_{i \in I} v_i \right) - n2^b.$$

- 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 Proof (concluded)

- The relative error from the optimum is:

$$\frac{\sum_{i \in I} v_i - \sum_{i \in I'} v_i}{\sum_{i \in I} v_i} \leq \frac{\sum_{i \in I} v_i - \sum_{i \in I'} v_i}{V} \leq \frac{n 2^b}{V}.$$

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

---

[a]See Eq. (17) on p. 715.

[b]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.

Comments

- INDEPENDENT SET and NODE COVER are reducible to each other (Corollary 45, p. 371).
- NODE COVER has an approximation threshold at most 0.5 (p. 727).
- 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

If 50 million people believe a foolish thing,
it's still a foolish thing.
— George Bernard Shaw (1856–1950)

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 $\phi[\, t \,]$;
else
    return $\phi[\, t0 \,] \vee \phi[\, t1 \,]$;
end if

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

---

[a]The same idea was used in the proof of Proposition 79 on p. 614.

NP-Completeness and Density[a]

Theorem 87 *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. 759.
- The trick is to keep the already discovered results $\phi[t]$ in a table H.

[a]Berman (1978).

```
if |t| = n then
    return φ[t];
else
    if (R(φ[t]), v) is in table H then
        return v;
    else
        if φ[t0] = "satisfiable" or φ[t1] = "satisfiable" then
            Insert (R(φ[t]), "satisfiable") into H;
            return "satisfiable";
        else
            Insert (R(φ[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?

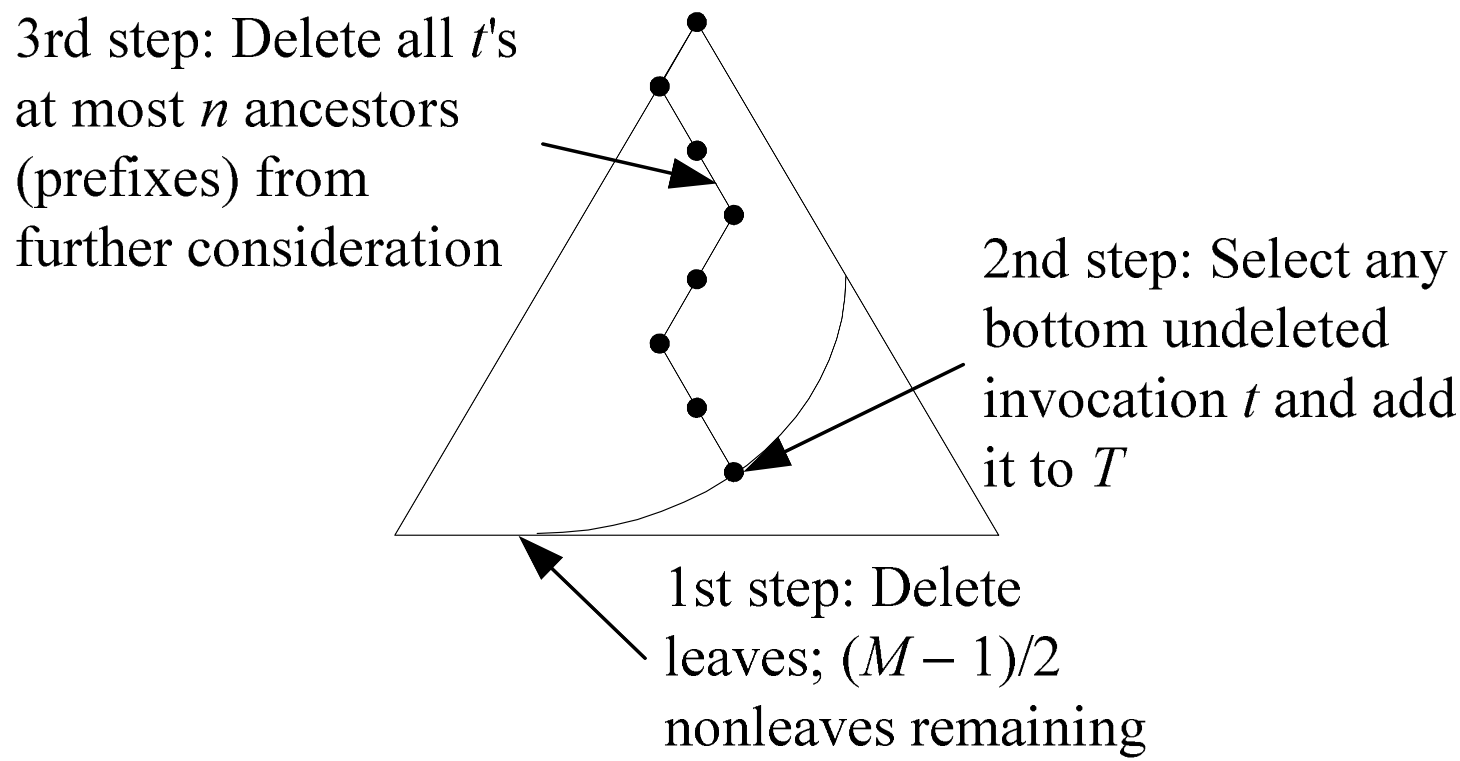
The Proof (continued)

- 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.
- If that number is a polynomial, the overall algorithm runs in polynomial time and we are done.
- The invocations of the algorithm form a binary tree of depth at most n.

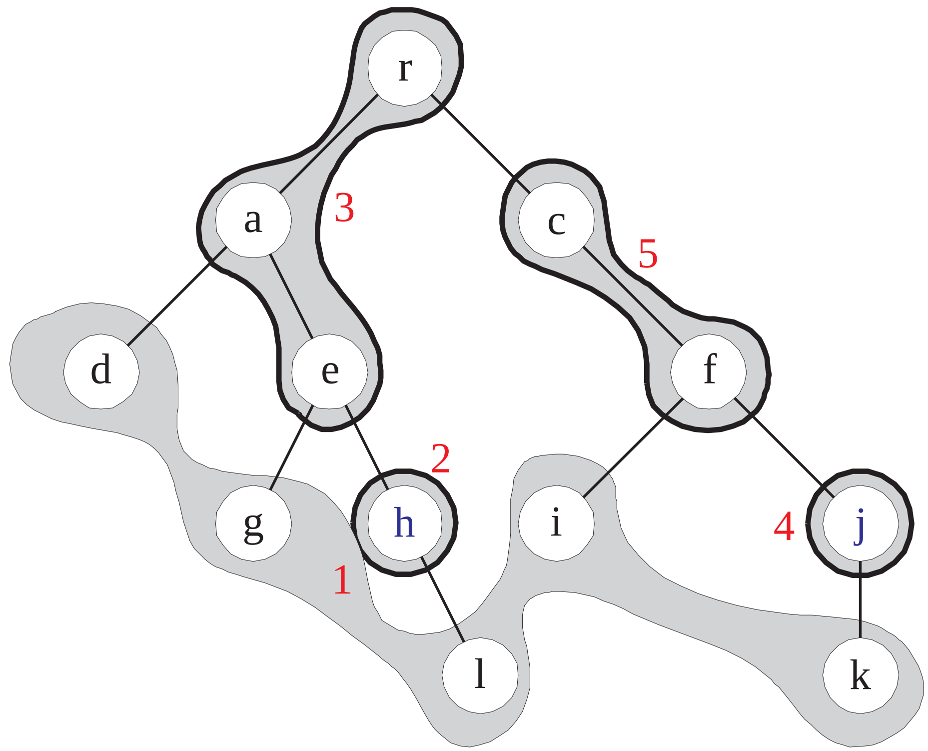
The Proof (continued)

- There is a set $T = \{ t_1, t_2, \ldots \}$ of invocations[a] 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.
- To build one such T, carry out the 1st step and then loop over the 2nd and 3rd steps on the next page.

[a]Partial truth assignments, i.e.


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



$T = \{\,\mathrm{h}, \mathrm{j}\,\}$; none of h and j is a prefix of the other.

The Proof (continued)

- All invocations $t \in T$ have different $R(\phi[\,t\,])$ values.
  - The invocation of one started after the invocation of the other had terminated.
  - If they had the same value, the one that was invoked later 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 $O(Mp(n)) = O(np^2(n))$.

Other Results for Sparse Languages

Theorem 88 (Mahaney (1980)) *If a sparse language is NP-complete, then P = NP.*

Theorem 89 (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.