

Approximability

Tackling Intractable Problems

- Many important problems are NP-complete or worse.
- **Heuristics** have been developed to attack them.
- They are **approximation algorithms**.
- How good are the approximations?
 - We are looking for theoretically *guaranteed* bounds, not “empirical” bounds.
- Are there NP problems that cannot be approximated well (assuming $NP \neq P$)?
- Are there NP problems that cannot be approximated at all (assuming $NP \neq P$)?

Some Definitions

- Given an **optimization problem**, each problem instance x has a set of **feasible solutions** $F(x)$.
- Each feasible solution $s \in F(x)$ has a cost $c(s) \in \mathbb{Z}^+$.
 - Here, cost refers to the quality of the feasible solution, not the time required to obtain it.
 - It is our **objective function**, e.g., total distance, satisfaction, or cut size.
- The **optimum cost** is $\text{OPT}(x) = \min_{s \in F(x)} c(s)$ for a minimization problem.
- It is $\text{OPT}(x) = \max_{s \in F(x)} c(s)$ for a maximization problem.

Approximation Algorithms

- Let algorithm M on x returns a feasible solution.
- M is an ϵ -**approximation algorithm**, where $\epsilon \geq 0$, if for all x ,

$$\frac{|c(M(x)) - \text{OPT}(x)|}{\max(\text{OPT}(x), c(M(x)))} \leq \epsilon.$$

- For a minimization problem,

$$\frac{c(M(x)) - \min_{s \in F(x)} c(s)}{c(M(x))} \leq \epsilon.$$

- For a maximization problem,

$$\frac{\max_{s \in F(x)} c(s) - c(M(x))}{\max_{s \in F(x)} c(s)} \leq \epsilon. \quad (9)$$

Lower and Upper Bounds

- For a minimization problem,

$$\min_{s \in F(x)} c(s) \leq c(M(x)) \leq \frac{\min_{s \in F(x)} c(s)}{1 - \epsilon}.$$

- So **approximation ratio** $\frac{\min_{s \in F(x)} c(s)}{c(M(x))} \geq 1 - \epsilon$.

- For a maximization problem,

$$(1 - \epsilon) \times \max_{s \in F(x)} c(s) \leq c(M(x)) \leq \max_{s \in F(x)} c(s). \quad (10)$$

- So approximation ratio $\frac{c(M(x))}{\max_{s \in F(x)} c(s)} \geq 1 - \epsilon$.

- They are alternative definitions of ϵ -approximation.

Range Bounds

- ϵ takes values between 0 and 1.
- For maximization problems, an ϵ -approximation algorithm returns solutions within $[(1 - \epsilon) \times \text{OPT}, \text{OPT}]$.
- For minimization problems, an ϵ -approximation algorithm returns solutions within $[\text{OPT}, \frac{\text{OPT}}{1 - \epsilon}]$.
- For each NP-complete optimization problem, we shall be interested in determining the *smallest* ϵ for which there is a polynomial-time ϵ -approximation algorithm.
- Sometimes ϵ has no minimum value.

Approximation Thresholds

- The **approximation threshold** is the greatest lower bound of all $\epsilon \geq 0$ such that there is a polynomial-time ϵ -approximation algorithm.
- The approximation threshold of an optimization problem can be anywhere between 0 (approximation to any desired degree) and 1 (no approximation is possible).
- If $P = NP$, then all optimization problems in NP have an approximation threshold of 0.
- So we assume $P \neq NP$ for the rest of the discussion.

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

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

- Hence the approximation ratio is $\Theta(\log^{-1} n)$.
- It is not an ϵ -approximation algorithm for any $\epsilon < 1$.

A 0.5-Approximation Algorithm^a

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

^aJohnson (1974).

Analysis

- C contains $|C|/2$ edges.
- No two edges of C share a node.^a
- *Any* node cover must contain at least one node from each of these edges.
- This means that $\text{OPT}(G) \geq |C|/2$.

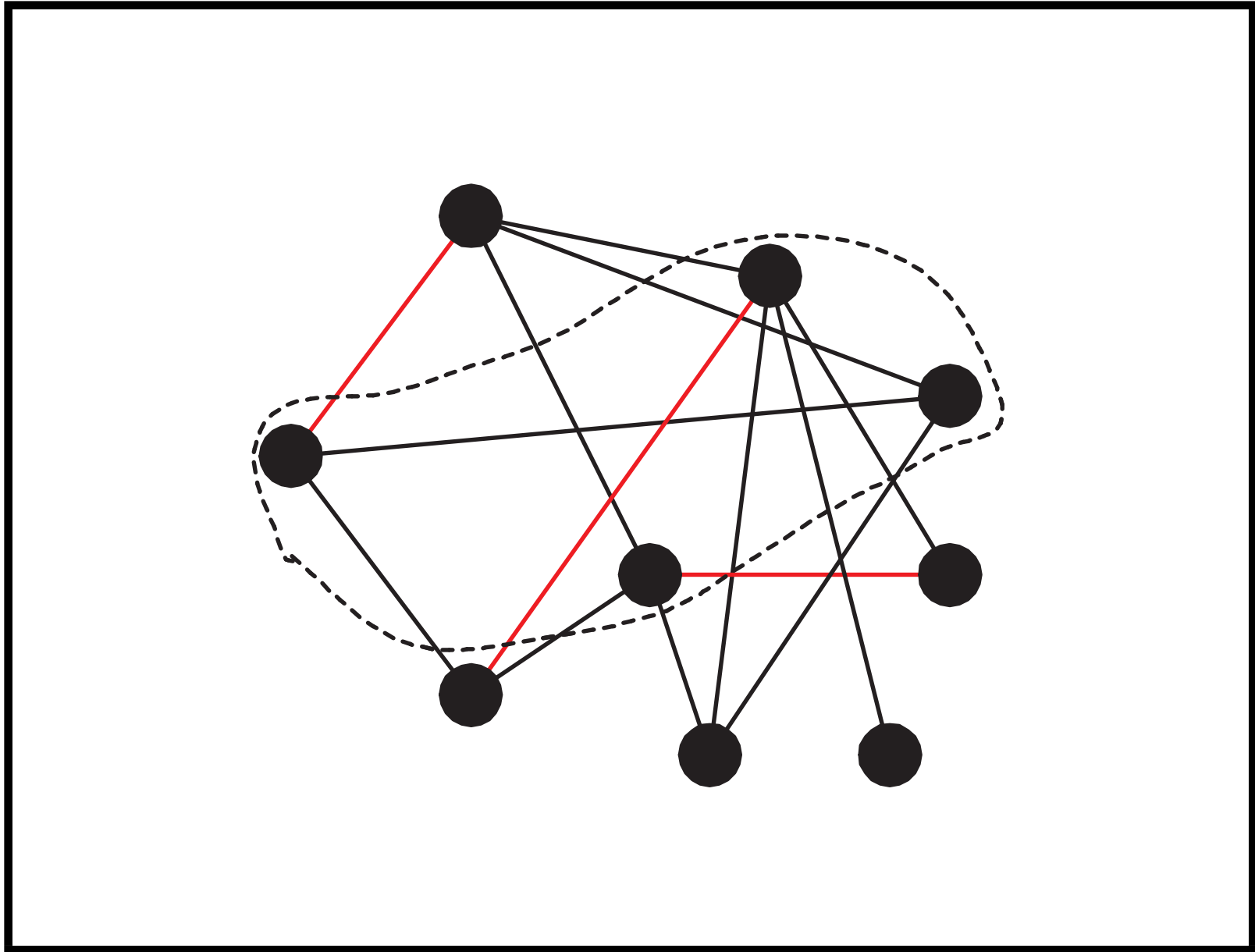
- So

$$\frac{\text{OPT}(G)}{|C|} \geq 1/2.$$

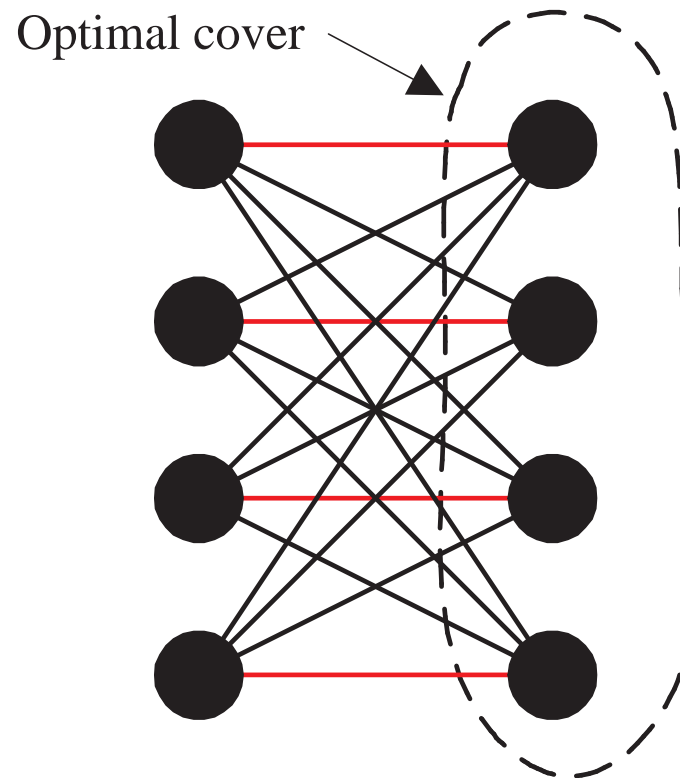
- The approximation threshold is ≤ 0.5 .^b

^aIn fact, C is a *maximal* matching.

^b0.5 is also the lower bound for any “greedy” algorithms (see Davis and Impagliazzo (2004)).



The 0.5 Bound Is Tight for the Algorithm^a



^aContributed by Mr. Jenq-Chung Li (R92922087) on December 20, 2003.

Maximum Satisfiability

- Given a set of clauses, MAXSAT seeks the truth assignment that satisfies the most.
- MAX2SAT is already NP-complete (p. 289).
- Consider the more general k -MAXGSAT for constant k .
 - Given a set of boolean expressions $\Phi = \{\phi_1, \phi_2, \dots, \phi_m\}$ in n variables.
 - Each ϕ_i is a *general* expression involving k variables.
 - k -MAXGSAT seeks the truth assignment that satisfies the most expressions.

A Probabilistic Interpretation of an Algorithm

- Each ϕ_i involves exactly k variables and is satisfied by t_i of the 2^k truth assignments.
- A random truth assignment $\in \{0, 1\}^n$ satisfies ϕ_i with probability $p(\phi_i) = t_i/2^k$.
 - $p(\phi_i)$ is easy to calculate as k is a constant.
- Hence a random truth assignment satisfies an expected number

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

of expressions ϕ_i .

The Search Procedure

- Clearly

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

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

The Search Procedure (concluded)

- By our hill-climbing procedure,

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

- So at least $p(\Phi)$ expressions are satisfied by truth assignment (t_1, t_2, \dots, t_n) .
- The algorithm is deterministic.

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

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

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

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 $P = NP$.

^aJohnson (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. 317).
- **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.

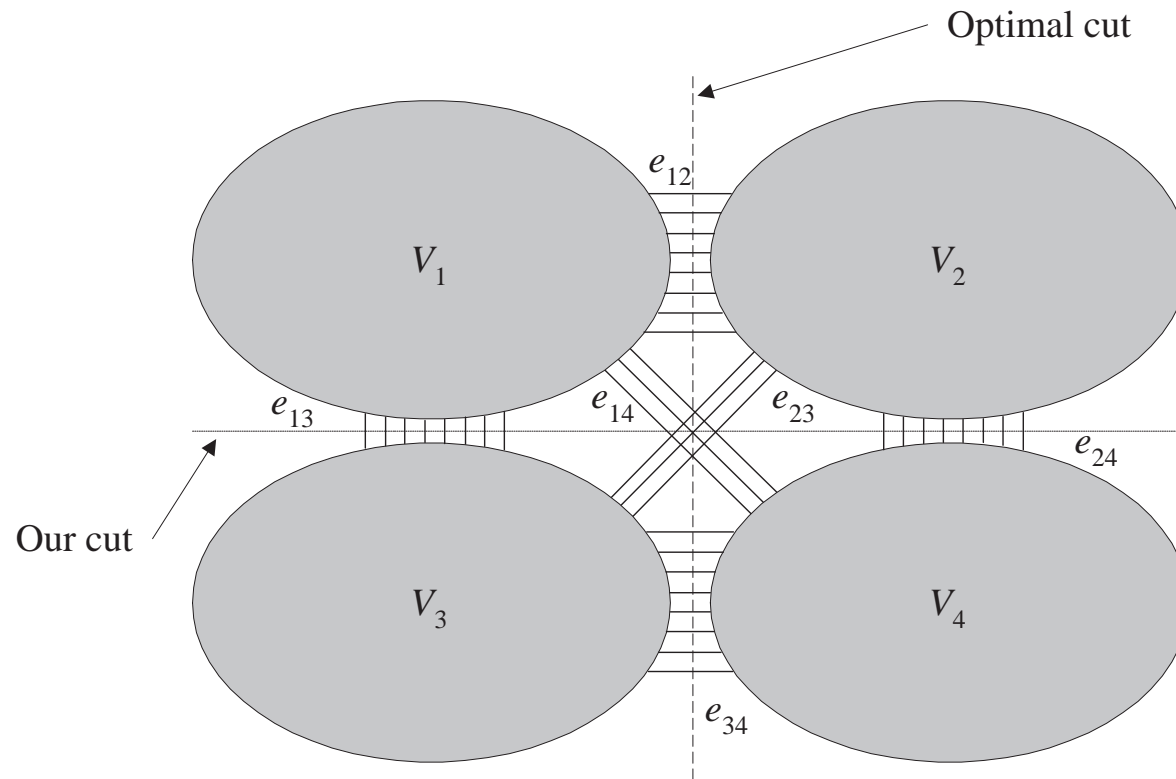
A 0.5-Approximation Algorithm for MAX CUT

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

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

^aGoemans and Williamson (1995).

Analysis



Analysis (continued)

- Partition $V = V_1 \cup V_2 \cup V_3 \cup V_4$, where our algorithm returns $(V_1 \cup V_2, V_3 \cup V_4)$ and 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
$$2e_{11} + e_{12} \leq e_{13} + e_{14}$$
 - The reason 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,

$$e_{12} \leq e_{23} + e_{24}$$

$$e_{34} \leq e_{23} + e_{13}$$

$$e_{34} \leq e_{14} + e_{24}$$

- Adding all four inequalities, dividing both sides by 2, and adding the inequality

$e_{14} + e_{23} \leq e_{14} + e_{23} + e_{13} + e_{24}$, we 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 (see p. 636).
 - But NODE COVER and MAXSAT have a threshold larger than 0.
- The situation is maximally pessimistic for TSP: It cannot be approximated unless $P = NP$ (see p. 634).
 - 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

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

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

^aIbarra and Kim (1975).

^bIf 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

$$V = \max\{v_1, v_2, \dots, v_n\}.$$

- For $0 \leq i \leq n$ and $0 \leq v \leq nV$, define $W(i, v)$ to be the minimum weight attainable by selecting some among the i first items, so that their value is exactly v .
- Start with $W(0, v) = \infty$ for all v .
- 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 $O(n^2V)$, not polynomial time.

The Proof (continued)

- Key idea: Limit the number of precision bits.
- 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.
- Given the instance $x = (w_1, \dots, w_n, W, v_1, \dots, v_n)$, we define the approximate instance

$$x' = (w_1, \dots, w_n, W, v'_1, \dots, v'_n).$$

The Proof (continued)

- Solving x' takes time $O(n^2V/2^b)$.
 - The algorithm only performs subtractions on the v_i -related values.
 - So these b bits can be *removed* from the calculations.
- The solution S' is close to the optimum solution S :

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

- Hence

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

The Proof (continued)

- Without loss of generality, $w_i \leq W$ (otherwise item i is redundant).
- V is a lower bound on OPT.
 - Picking only the item with value V is a legitimate choice.
- The relative error from the optimum is $\leq n2^b/V$ as

$$\frac{\sum_{i \in S} v_i - \sum_{i \in S'} v_i}{\sum_{i \in S} v_i} \leq \frac{\sum_{i \in S} v_i - \sum_{i \in S'} v_i}{V} \leq \frac{n2^b}{V}.$$

The Proof (concluded)

- Truncate the last $b = \lfloor \log_2 \frac{\epsilon V}{n} \rfloor$ bits of the values.
- The algorithm becomes ϵ -approximate (see Eq. (9) on p. 612).
- The running time is then $O(n^2 V / 2^b) = O(n^3 / \epsilon)$, a polynomial in n and $1/\epsilon$.^a

^aIt hence depends on the *value* of $1/\epsilon$. Thanks to a lively class discussion on December 20, 2006.

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. 637, we presented a pseudo-polynomial-time algorithm for KNAPSACK that runs in time $O(n^2V)$.
- How about TSP (D), another NP-complete problem?

^aGarey 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 40 (p. 333) 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

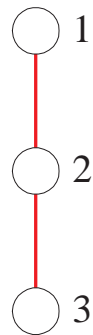
- Algorithm M is a **polynomial-time approximation scheme (PTAS)** for a problem if:
 - For each $\epsilon > 0$ and instance x of the problem, M runs in time polynomial (depending on ϵ) in $|x|$.
 - * Think of ϵ as a constant.
 - M is an ϵ -approximation algorithm for every $\epsilon > 0$.

Fully Polynomial-Time Approximation Scheme

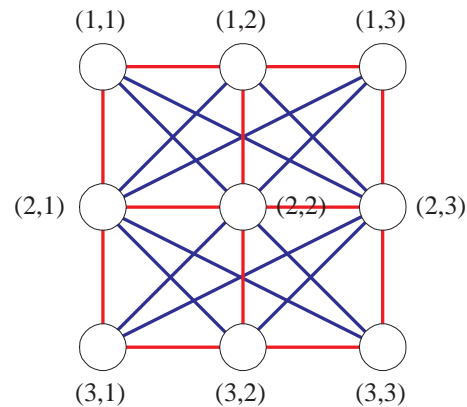
- A polynomial-time approximation scheme is **fully polynomial (FPTAS)** if the running time depends polynomially on $|x|$ and $1/\epsilon$.
 - Maybe the best result for a “hard” problem.
 - For instance, KNAPSACK is fully polynomial with a running time of $O(n^3/\epsilon)$ (p. 636).

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 $\{\{ (u, u'), (v, v') \} : (u = v \wedge \{u', v'\} \in E) \vee \{u, v\} \in E\}$.



G

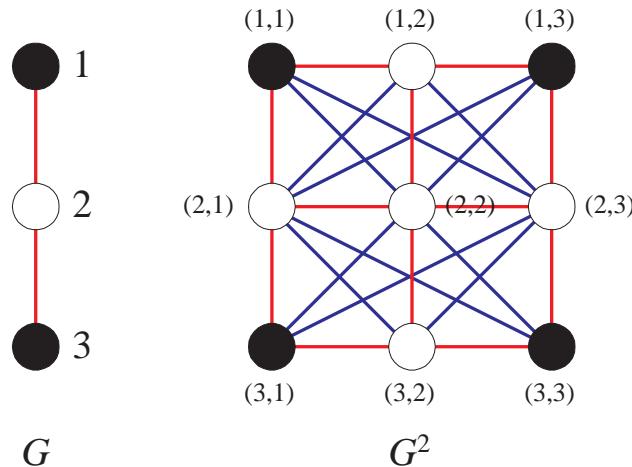


G^2

Independent Sets of G and G^2

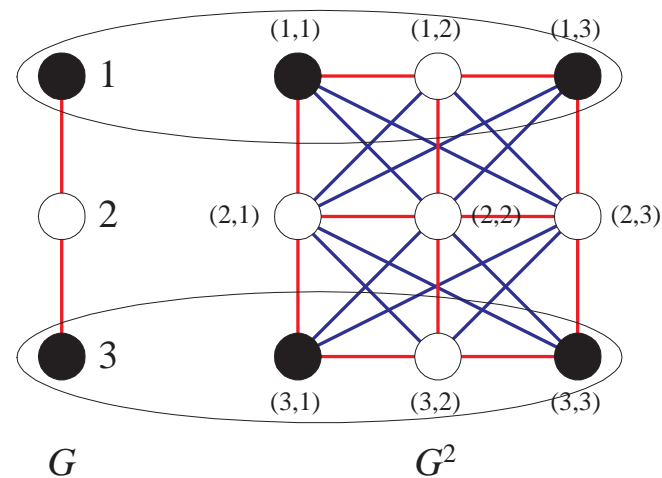
Lemma 79 $G(V, E)$ has an independent set of size k if and only if G^2 has an independent set of size k^2 .

- 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 .

^aThanks 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.

The Proof (continued)

- By Lemma 79 (p. 647), 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. 647), 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. (10) on p. 613.

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.

^aIt is not fully polynomial.

Comments

- INDEPENDENT SET and NODE COVER are reducible to each other (Corollary 37, p. 311).
- NODE COVER has an approximation threshold at most 0.5 (p. 618).
- 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).