Maximum Satisfiability

- Given a set of clauses, MAXSAT seeks the truth assignment that satisfies the most.
- MAX2SAT is already NP-complete (p. 266).
- Consider the more general k-MAXGSAT for constant k.
  - Given a set of boolean expressions $\Phi = \{\phi_1, \phi_2, \ldots, \phi_m\}$ 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

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

- 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 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,

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

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

[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. 290).
- **Local search** starts from a feasible solution and performs "local" improvements until none are possible.

A 0.5-Approximation Algorithm for MAX CUT

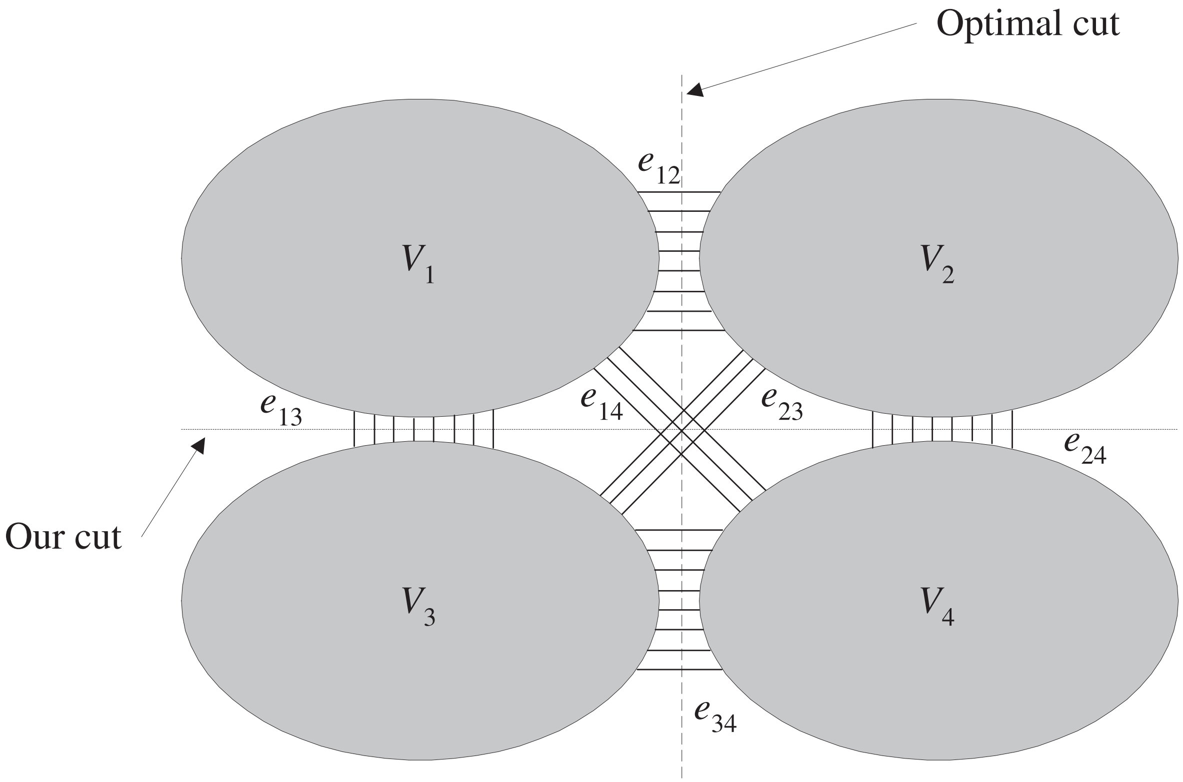
$S := \emptyset$;
while $\exists v \in V$ whose switching sides results in a larger cut **do**
    Switch the side of v;
end while
return S;

- 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 $(V_1 \cup V_2, V_3 \cup V_4)$ 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.
- Because no migration of nodes can improve the algorithm's cut, for each node in V_1, its edges to $V_1 \cup V_2$ are outnumbered by those to $V_3 \cup V_4$.
- Considering all nodes in V_1 together, we have $2e_{11} + e_{12} \leq e_{13} + e_{14}$, which implies

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

Analysis (concluded)

- Similarly,

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

- 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. 590).
  - 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. 588).
  - 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 74 *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 75 *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.
- Let
$$V = \max\{v_1, v_2, \ldots, v_n\}.$$

---

[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)

- 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

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

- 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)

- Given the instance $x = (w_1, \ldots, w_n, W, v_1, \ldots, v_n)$, we define the approximate instance
$$x' = (w_1, \ldots, w_n, W, v'_1, \ldots, v'_n),$$
where
$$v'_i = 2^b \left\lfloor \frac{v_i}{2^b} \right\rfloor.$$
- Solving x' takes time $O(n^2 V / 2^b)$.
  - The algorithm only performs subtractions on the v_i-related values.
  - So these b bits can be *removed* from the calculations.

The Proof (continued)

- The solution S' is close to the optimum solution S:

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

- Hence

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

- Without loss of generality, $w_i \leq W$ (otherwise item i is redundant).
- V is a lower bound on OPT.
  - Picking the item with value V alone is a legitimate choice.

The Proof (concluded)

- The relative error from the optimum is $\leq n2^b/V$ as

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

- Truncate the last $b = \lfloor \log_2 \frac{\epsilon V}{n} \rfloor$ bits of the values.
- The algorithm becomes ϵ-approximate (see Eq. (8) on p. 567).
- 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.

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. 591, 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 38 (p. 306) 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 $\epsilon > 0$.

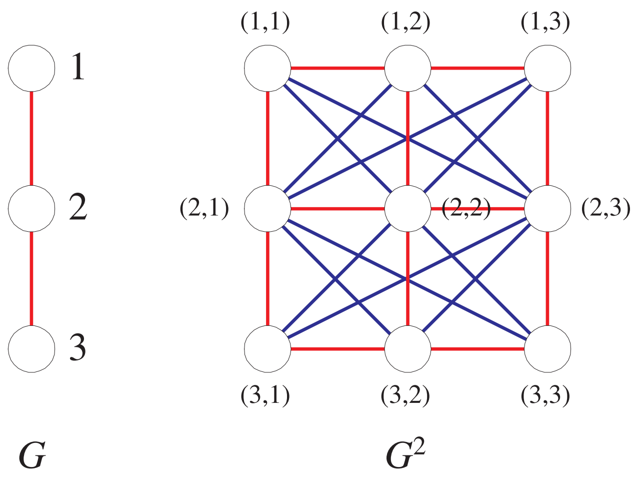
Fully Polynomial-Time Approximation Scheme

- 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. 590).

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

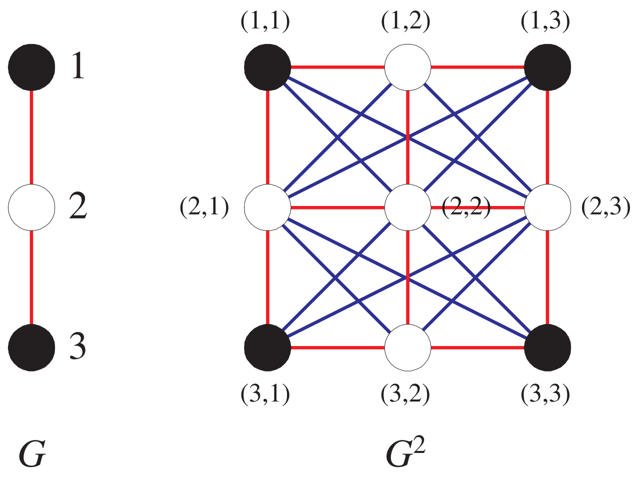
  $\{\{\,(u, u'), (v, v')\,\} : (u = v \wedge \{\,u', v'\,\} \in E) \vee \{\,u, v\,\} \in E\}$.



Independent Sets of G and G^2

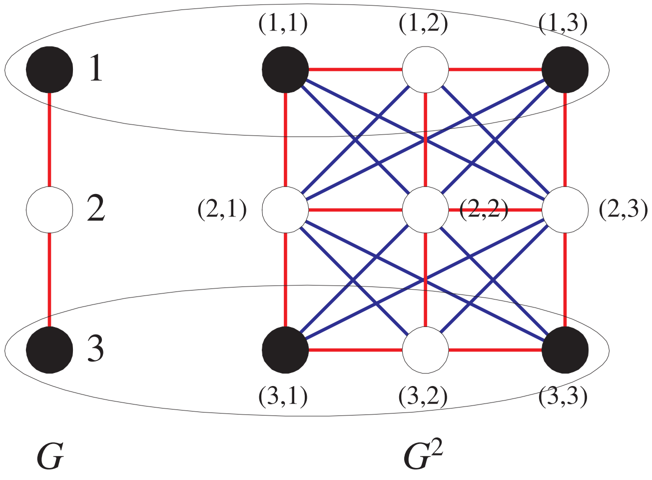
Lemma 76 *$G(V, E)$ has an independent set of size k if and only if G^2 has an independent set of size k^2.*

- 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 77 *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 76 (p. 600), 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 76 (p. 600), 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. (9) on p. 568.

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 36, p. 286).
- NODE COVER has an approximation threshold at most 0.5 (p. 573).
- 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).