

**The following problems are NP Complete:**

- SAT
- 3-SAT
- 3-COLOR
- CLIQUE
- Subset Sum
- Knapsack

# Knapsack

Given  $n$  objects:

object	$o_1$	$o_2$	$\cdots$	$o_n$	
weight	$w_1$	$w_2$	$\cdots$	$w_n$	$\geq 0$
value	$v_1$	$v_2$	$\cdots$	$v_n$	

$W =$  max weight I can carry in my knapsack.

## Optimization Problem:

choose  $S \subseteq \{1, \dots, n\}$

to maximize  $\sum_{i \in S} v_i$

such that  $\sum_{i \in S} w_i \leq W$

## Decision Problem:

Given  $\bar{w}, \bar{v}, W, V$ , can I get total value  $\geq V$  while total weight is  $\leq W$ ?

**Proposition 20.1** *Knapsack is NP-Complete.*

**Proof:** Let  $I = \langle m_1, \dots, m_n, T \rangle$  be an instance of Subset Sum.

**Problem:**  $(\exists S \subseteq \{1, \dots, n\}) (\sum_{i \in S} m_i = T)$

Let  $f(I) = \langle m_1, \dots, m_n, m_1, \dots, m_n, T, T \rangle$  be an instance of Knapsack.

**Claim:**  $I \in \text{Subset Sum} \iff f(I) \in \text{Knapsack}$

$$(\exists S \subseteq \{1, \dots, n\}) (\sum_{i \in S} m_i = T)$$

$\iff$

$$(\exists S \subseteq \{1, \dots, n\}) (\sum_{i \in S} m_i \geq T \quad \wedge \quad \sum_{i \in S} m_i \leq T) \spadesuit$$

**Fact 20.2** *Even though Knapsack is NP-Complete there is an efficient dynamic programming algorithm that can closely approximate the maximum possible  $V$ .*

**Fact:** NP-complete decision problems are all equivalent.

**Belief:** NP-complete problems require exponential time in the worst case.

**Fact:** Difficulty of NP Approximation problems varies widely.

**Definition 20.3**  $A$  is an *NP-optimization problem* iff

- For each instance  $x$ ,  $n = |x|$ ,  $F(x) \subseteq \Sigma^{p(n)}$  is the set of *feasible solutions*. We can test in  $\mathbf{P}$  whether  $s \in F(x)$ .
- Each  $s \in F(x)$  has a cost  $c(s) \in \mathbf{Z}^+$ . The cost  $c(s)$  is computable in  $F(\mathbf{P})$ .

For minimization problems,

$$\text{OPT}(x) = \min_{s \in F(x)} c(s)$$

For maximization problems,

$$\text{OPT}(x) = \max_{s \in F(x)} c(s)$$



**Definition 20.4** Let  $M$  be a polynomial-time algorithm s.t. on any instance  $x$ ,

$$M(x) \in F(x)$$

$M$  is an  $\epsilon$ -approximation algorithm iff for all  $x$ ,

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

$$0 < \epsilon < 1$$

$\epsilon = .01$  is an excellent approximation! Minimization problems: at most  $\frac{100}{99}$  times optimal. Maximization problems: at least 99 percent of optimal.

$\epsilon = \frac{1}{2}$ : Minimization problems: no more than twice optimal. Maximization problems: at least half optimal.

$\epsilon = .99$ : not a very good approximation! Minimization problems: at most 100 times more than optimal; Maximization problems: at least one percent of optimal.

## Four Classes of NP Optimization Problems

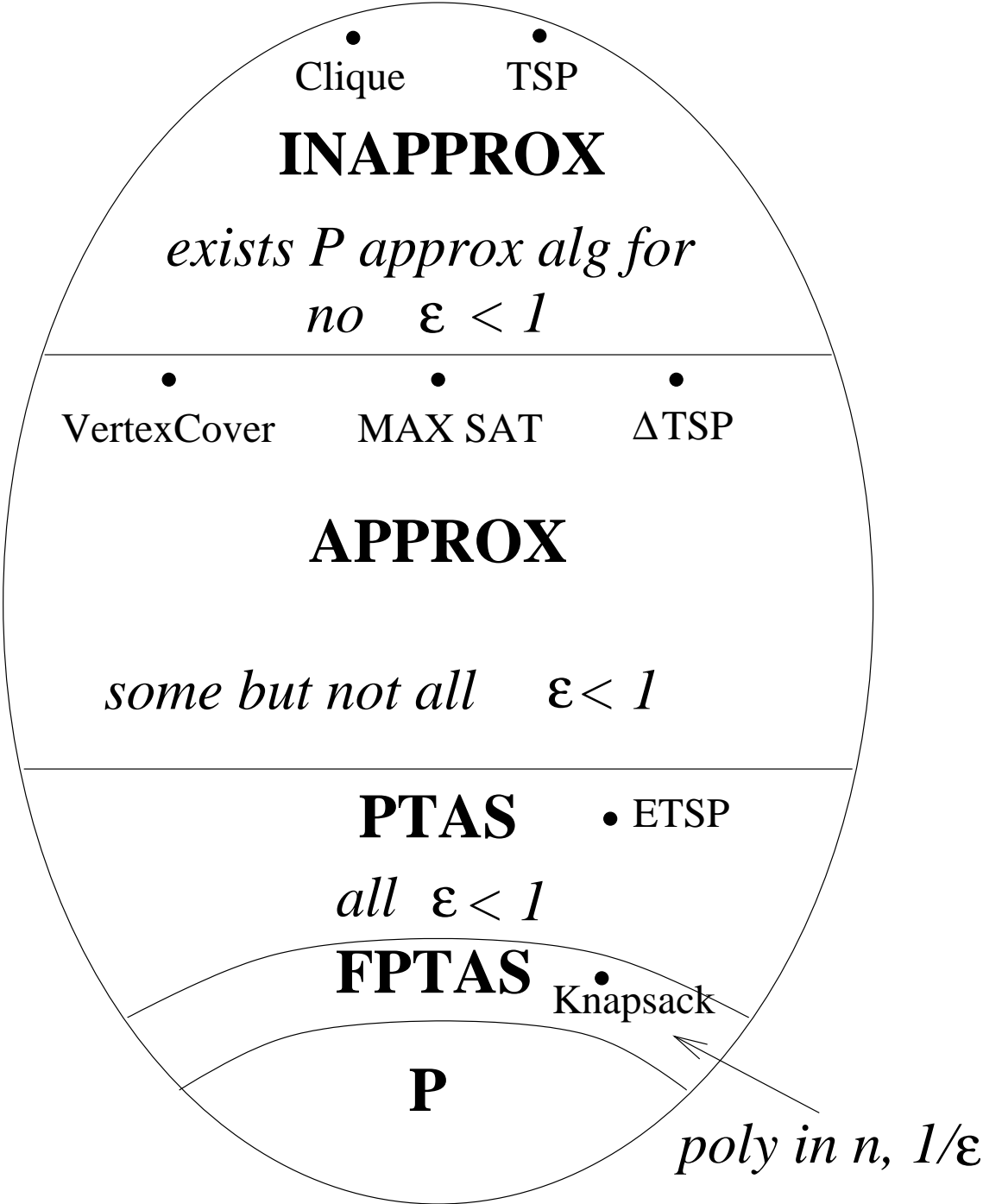
INAPPROX  $\equiv$  no PTIME  $\epsilon$ -approx alg if  $P \neq NP$

APPROX  $\equiv$  ( $\exists \epsilon_1 \epsilon_2 . 0 < \epsilon_1 < \epsilon_2 < 1$ )  
exists PTIME  $\epsilon_2$ -approx alg  
no PTIME  $\epsilon_1$ -approx alg if  $P \neq NP$

PTAS  $\equiv$  ( $\forall \epsilon > 0$ ) exists PTIME  $\epsilon$ -approx alg

FPTAS  $\equiv$  ( $\forall \epsilon > 0$ ) exists uniform  $\epsilon$ -approx alg  
running in time  $\text{poly}(n, \frac{1}{\epsilon})$

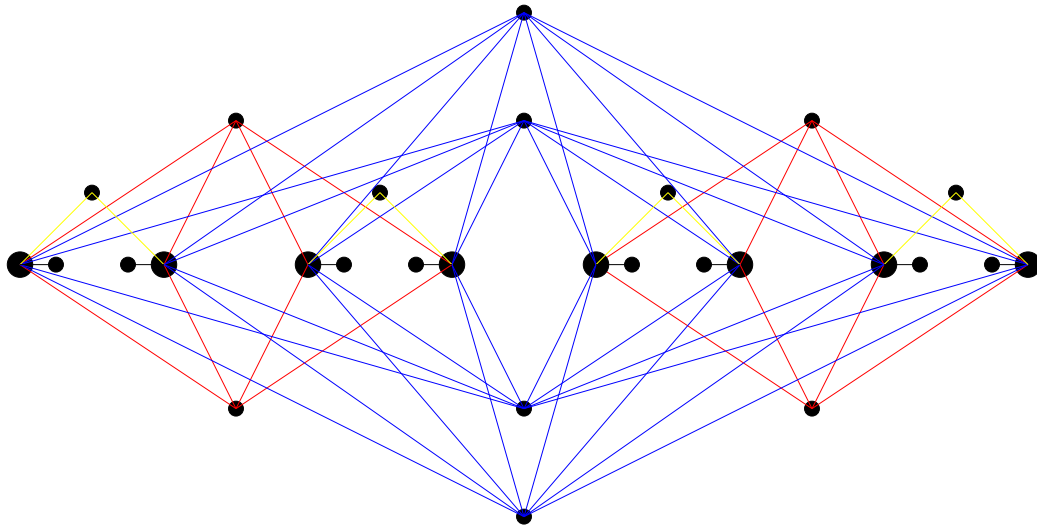
(F)PTAS stands for *(Fully) Polynomial-Time Approximation Scheme*.





Input: an undirected graph  $G = (V, E)$ .

Output: min size  $C \subset V$  s.t.  $C$  touches every edge.



**Greedy:** (nodes of high degree) About  $\log n$  times optimal, in the example above. (There are about  $n \log n$  total nodes, the  $n$  fat ones are a vertex cover, but the greedy algorithm takes most of the others first.)

## Better: Find a Maximal Matching

1.  $C := \emptyset$
2. **while** ( $E \neq \emptyset$ ) **do** {
3.     pick  $(u, v) \in E$
4.      $C := C \cup \{u, v\}$
5.     delete  $u, v$  from  $G$  }

The edges picked are a *maximal matching*, a disjoint set of edges to which we can't add another disjoint edge. If there are  $m$  edges in this matching, we've used  $2m$  nodes in  $C$  but any algorithm would have to use at least  $m$ .

$$|C| \leq 2\text{opt}(G) \quad \epsilon = \frac{1}{2} \quad \textbf{Best known approx ratio}$$

A *Hamilton circuit* for an undirected graph  $G$  is a cycle that starts and ends at some vertex  $v$  and visits every other vertex exactly once.

$$\text{HC} = \{G \mid G \text{ has a Hamilton Circuit}\}$$

**Fact 20.5** *HC is NP-Complete. (Nicest proof is in Sipser.)*

$$\text{TSP} = \{G = (V, E, w), L \mid G \text{ has a HC of weight } \leq L\}$$

$$G = (V, E), n = |V|, \text{ let } h(G) = (V, \binom{V}{2}, w_h), L_h = n,$$

$$w_h(u, v) = \begin{cases} 1 & \text{if } (u, v) \in E \\ n \cdot 10^6 & \text{otherwise} \end{cases}$$

**Observation 20.6** *For any undirected graph  $G$ ,*

$$G \in \text{HC} \iff h(G) \in \text{TSP}$$

**Corollary 20.7** *If TSP has a polynomial-time  $\epsilon$ -approximation algorithm for any  $\epsilon < 1$ , then  $P = NP$ . Thus,  $\text{TSP} \in \text{INAPPROX}$ .*

$G = (V, E)$ ,  $n = |V|$ , let  $e(G) = (V, \binom{V}{2}, w_e)$ ,  $L_e = n \cdot 10^6$ ,

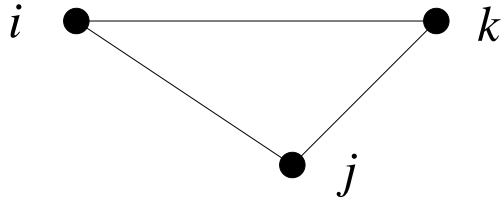
$$w_e(u, v) = \begin{cases} 1,000,000 & \text{if } (u, v) \in E \\ 1,000,001 & \text{otherwise} \end{cases}$$

Let Fair-TSP be the subset of TSP s.t. no edge weight is more than 0.0001 percent more than any other edge weight.

**Observation 20.8** *For any undirected graph  $G$ ,*

$$G \in HC \Leftrightarrow e(G) \in \text{TSP} \Leftrightarrow e(g) \in \text{Fair-TSP}$$

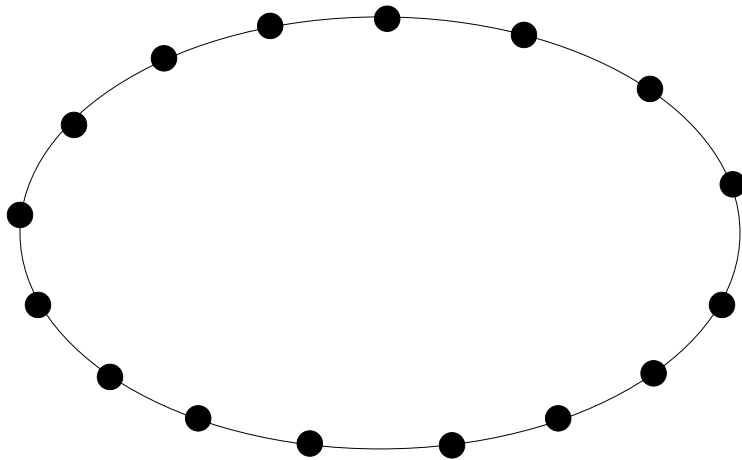
**Observation 20.9** *Fair-TSP is NP-complete as a decision problem and as an optimization problem it has a polynomial-time  $10^{-6}$ -approximation algorithm.*



$$\Delta\text{TSP: } d(i, k) \leq d(i, j) + d(j, k)$$

**Claim 20.10** *Minimum Spanning Tree is a lower bound for  $\Delta\text{TSP}$  :  $c(\text{MST}) \leq c(\Delta\text{TSP})$ .*

**Proof:** Visualize optimal tour:

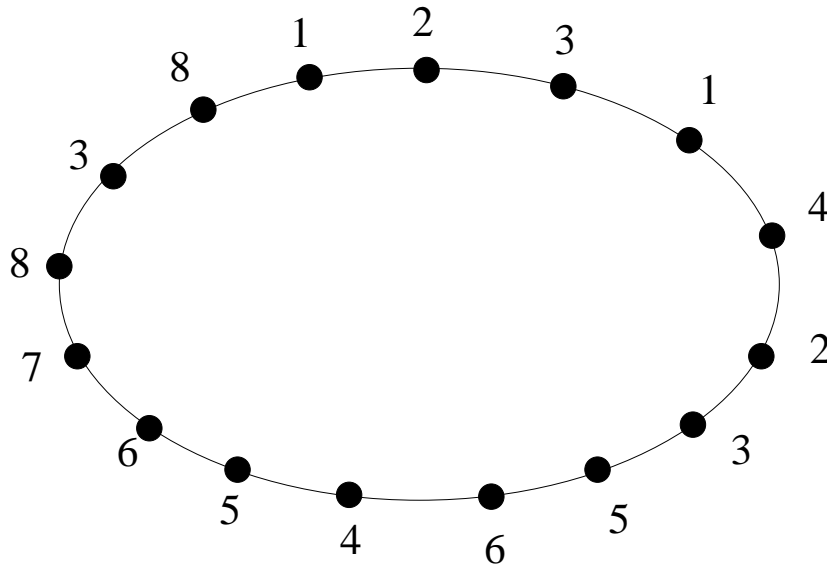


Delete one edge and we have a spanning tree.



**Theorem 20.11**  $c(\text{MST}) \leq c(\Delta\text{TSP}) \leq 2 \cdot c(\text{MST})$

**Proof:** The multigraph  $2 \cdot \text{MST}$ , made by taking two copies of each edge in the tree, is connected and all its nodes have even degree.



Thus it has an Euler's tour, providing an  $\epsilon = \frac{1}{2}$  approximation algorithm. ♠

**Aside:** A **multigraph**  $G = (V, E)$  is a graph except that  $E$  may be a multiset, i.e., there can be more than one edge between a certain pair of vertices. An **Euler's Tour** of an undirected multigraph  $G$  is a tour that starts and ends at the same vertex and traverses **each edge exactly once**.

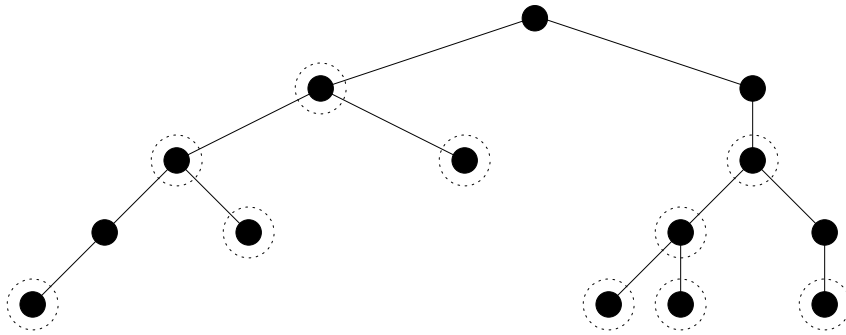
**Fact:**  $G$  has an Euler's tour iff  $G$  is connected and each vertex of  $G$  has even degree. If  $G$  has an Euler's tour, then such a tour can be computed in linear time.

## Christofides Algorithm (1976)

In the MST, only worry about the odd degree nodes.

There are an even number of vertices of odd degree.

In polynomial time we can find a minimum weight perfect matching,  $M$ , on the odd-degree nodes.



$\text{MST} \cup M$  is an Eulerian graph.

$\text{MST} \leq \text{TSP}; \quad M \leq \frac{1}{2}\text{TSP}.$

Thus, we get a tour at most 1.5 times optimal.  $\epsilon = \frac{1}{3}$

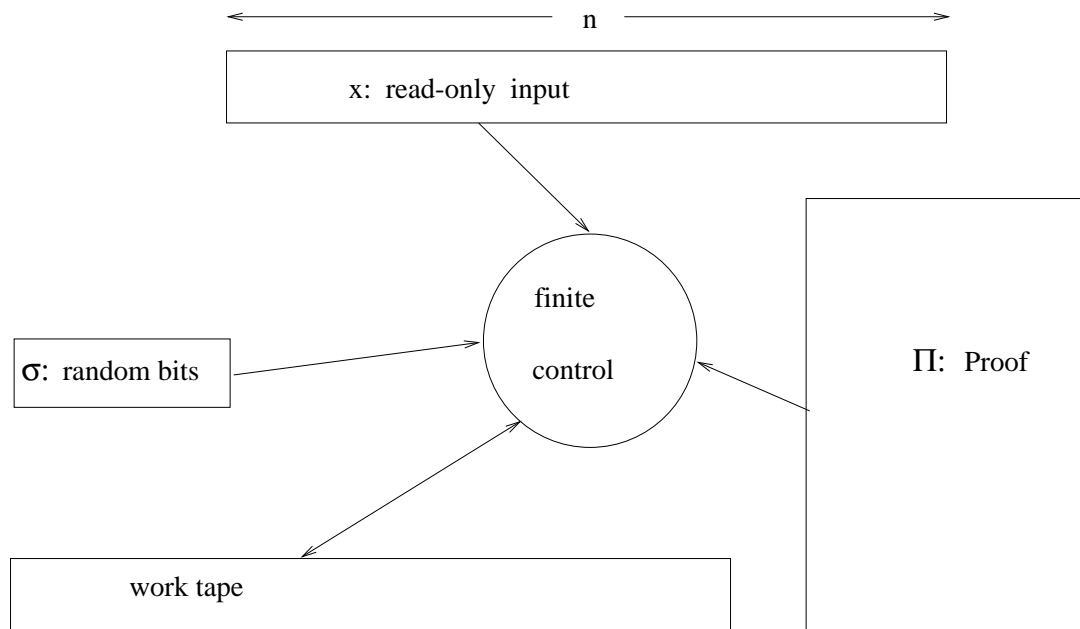


## Euclidean TSP

ETSP: Euclidean distance in plane:

$$d((x, y), (x', y')) = \sqrt{(x - x')^2 + (y - y')^2}$$

ETSP has a Polynomial-Time Approximation Scheme (PTAS)  
[Arora 1997].



Merlin-Arthur games (MA) [Babai]

Decision problem:  $D$ ; input string:  $x$

**Merlin** — Prover — chooses the polynomial-length string  $\Pi$  that **Maximizes** the chances of convincing Arthur that  $x$  is an element of  $D$ .

**Arthur** — Verifier — “computes” the **Average** value of his possible computations on  $\Pi, x$ . Arthur is a polynomial-time, probabilistic Turing machine.

**Definition 20.12** We say that  $\mathbf{A}$  *accepts*  $D$  iff the following conditions hold:

1. If  $x \in D$ , there exists a proof  $\Pi_x$ , such that  $\mathbf{A}$  accepts for every random string  $\sigma$ ,

$$Pr_{\sigma} [\mathbf{A}^{\Pi_x}(x, \sigma) = \textit{Accept}] = 1$$

2. If  $x \notin D$ , for every proof  $\Pi$ ,  $\mathbf{A}$  rejects for most of the random strings  $\sigma$ ,

$$Pr_{\sigma} [\mathbf{A}^{\Pi}(x, \sigma) = \textit{Accept}] < \frac{1}{2}$$



Any decision problem  $D \in \mathbf{NP}$  has a deterministic, polynomial-time verifier satisfying Definition 20.12.

By adding randomness to the verifier, we can greatly restrict its computational power and the number of bits of  $\Pi$  that it needs to look at, while still enabling it to accept all of  $\mathbf{NP}$ .

We say that a verifier  $\mathbf{A}$  is  $(r(n), q(n))$ -restricted iff for all inputs of size  $n$ , and all proofs  $\Pi$ ,  $\mathbf{A}$  uses at most  $O(r(n))$  random bits and examines at most  $O(q(n))$  bits of its proof,  $\Pi$ .

Let  $\text{PCP}(r(n), q(n))$  be the set of boolean queries that are accepted by  $(r(n), q(n))$ -restricted verifiers.

**Fact 20.13 (PCP Theorem)**     $\mathbf{NP} = \text{PCP}[\log n, 1]$

The proof of this theorem is pretty messy, certainly more than we can deal with here. But we can look at the applications of the PCP Theorem to approximation problems.

**MAX-3-SAT:** given a 3CNF formula, find a truth assignment that maximizes the number of true clauses.

$$(x_1 \vee x_2 \vee \overline{x_3}) \wedge (x_1 \vee x_4 \vee \overline{x_5}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_4}) \wedge (x_2 \vee \overline{x_3} \vee \overline{x_4}) \\ \wedge (\overline{x_2} \vee x_3 \vee x_5) \wedge (\overline{x_3} \vee \overline{x_4} \vee \overline{x_5}) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3) \wedge (\overline{x_2} \vee \overline{x_4} \vee x_5)$$

**Proposition 20.14** *MAX-3-SAT has a polynomial-time  $\epsilon = \frac{1}{2}$  approximation algorithm.*

**Proof:** Be greedy, set each variable in turn to the better value. ♠

You can do better – a random assignment gets 7/8 of the clauses.

**Open for Years:** Assuming  $\mathbf{NP} \neq \mathbf{P}$  is there some  $\epsilon$ ,  $0 < \epsilon < 1$  s.t. MAX-3-SAT has no PTIME  $\epsilon$ -approximation algorithm?

**Theorem 20.15** *The PCP theorem ( $\mathbf{NP} = \text{PCP}[\log n, 1]$ ) is equivalent to the fact that*

*If  $\mathbf{P} \neq \mathbf{NP}$ , then*

*For some  $\epsilon$ ,  $1 > \epsilon > 0$ ,*

*MAX-3-SAT has no polynomial-time,  $\epsilon$ -approximation algorithm.*

**Fact 20.16** *MAX-3-SAT has a PTIME approximation algorithm with  $\epsilon = \frac{1}{8}$  and no better ratio can be achieved unless  $\mathbf{P} = \mathbf{NP}$ .*

## **References:**

- *Approximation Algorithms for NP Hard Problems*, Dorit Hochbaum, ed., PWS, 1997.
- Juraj Hromkovic, *Algorithmics for Hard Problems*, Springer, 2001.
- Sanjeev Arora, “The Approximability of NP-hard Problems”, STOC 98, [www.cs.princeton.edu/~arora](http://www.cs.princeton.edu/~arora).