Coping with NP-completeness

Q. Suppose I need to solve an NP-hard problem. What should I do?

A. Sacrifice one of three desired features.
   i. Solve arbitrary instances of the problem.
   ii. Solve problem to optimality.
   iii. Solve problem in polynomial time.

Coping strategies.
   i. Design algorithms for special cases of the problem.
   ii. Design approximation algorithms or heuristics.
   iii. Design algorithms that may take exponential time.

Independent set on trees

Independent set on trees. Given a tree, find a maximum cardinality subset of nodes such that no two are adjacent.

Fact. A tree has at least one node that is a leaf (degree = 1).

Key observation. If node \( v \) is a leaf, there exists a max cardinality independent set containing \( v \).

Pf. [exchange argument]
   • Consider a max cardinality independent set \( S \).
   • If \( v \in S \), we’re done.
   • Let \( (u, v) \) be some edge.
     • if \( u \notin S \) and \( v \notin S \), then \( S \cup \{v\} \) is independent \( \Rightarrow S \) not maximum
     • if \( u \in S \) and \( v \notin S \), then \( S \cup \{v\} - \{u\} \) is independent
Independent set on trees: greedy algorithm

**Theorem.** The following greedy algorithm finds a max cardinality independent set in forests (and hence trees).

**Pf.** Correctness follows from the previous key observation. •

\[
\text{INDEPENDENT-SET-IN-A-FOREST} (F) \\
S \leftarrow \emptyset. \\
\text{WHILE} (F \text{ has at least 1 edge}) \\
\quad e \leftarrow (u, v) \text{ such that } v \text{ is a leaf.} \\
\quad S \leftarrow S \cup \{ v \}. \\
\quad F \leftarrow F - \{ u, v \}. \quad \text{delete } u \text{ and } v \text{ and all incident edges} \\
\text{RETURN } S \cup \{ \text{nodes remaining in } F \}. \\
\]

**Remark.** Can implement in \( O(n) \) time by considering nodes in postorder.

Weighted independent set on trees: dynamic programming algorithm

**Theorem.** The dynamic programming algorithm finds a max weighted independent set in a tree in \( O(n) \) time.

\[
\text{WEIGHTED-INDEPENDENT-SET-IN-A-TREE} (T) \\
\text{Root the tree } T \text{ at a node } r. \\
S \leftarrow \emptyset. \\
\text{FOREACH (node } u \text{ of } T \text{ in postorder) } \\
\quad \text{IF (} u \text{ is a leaf) } \\
\quad \quad M_{in}[u] = w_u. \\
\quad \quad M_{out}[u] = 0. \\
\quad \text{ELSE } \\
\quad \quad M_{in}[u] = w_u + \sum_{v \in \text{children}(u)} M_{out}[v]. \\
\quad \quad M_{out}[u] = \sum_{v \in \text{children}(u)} \max \{ M_{in}[v], M_{out}[v] \}. \\
\text{RETURN } \max \{ M_{in}[r], M_{out}[r] \}. \\
\]

**Dynamic programming solution.** Root tree at some node, say \( r \).

- \( OPT_{in}(u) = \max \) weight independent set of subtree rooted at \( u \), containing \( u \).
- \( OPT_{out}(u) = \max \) weight independent set of subtree rooted at \( u \), not containing \( u \).
- \( OPT = \max \{ OPT_{in}(r), OPT_{out}(r) \} \).

\[
OPT_{in}(u) = w_u + \sum_{v \in \text{children}(u)} OPT_{out}(v) \\
OPT_{out}(u) = \sum_{v \in \text{children}(u)} \max \{ OPT_{in}(v), OPT_{out}(v) \} \\
\]

NP-hard problems on trees: context

**Independent set on trees.** Tractable because we can find a node that breaks the communication among the subproblems in different subtrees.

**Linear-time on trees.** \( \text{VERTEX-COVER, DOMINATING-SET, GRAPH-ISOMORPHISM, ...} \)
**Planarity**

**Def.** A graph is **planar** if it can be embedded in the plane in such a way that no two edges cross.

---

**Applications.** VLSI circuit design, computer graphics, ...

---

**Planarity testing**

**Theorem.** [Hopcroft-Tarjan 1974] There exists an \(O(n)\) time algorithm to determine whether a graph is planar.

---

**Efficient Planarity Testing**

**JOHN HOPCROFT AND ROBERT TARJAN**

**Cornell University, Ithaca, New York**

**Abstract.** This paper describes an efficient algorithm to determine whether an arbitrary graph \(G\) can be embedded in the plane. The algorithm may be viewed as an iterative version of a method originally proposed by Auslander and Parter and currently formulated by Goldberg. The algorithm uses depth-first search and has \(O(G^2)\) time and space bounds, where \(G\) is the number of vertices in \(G\). An Atlas implementation of the algorithm successfully tested graphs with as many as 900 vertices in less than 10 seconds.

---

**Polynomial time detour**

**Graph minor theorem.** [Robertson-Seymour 1980s]

**Pf of theorem.** Tour de force.

**Corollary.** There exist an \(O(n^3)\) algorithm to determine if a graph can be embedded in the torus in such a way that no two edges cross.

---

**Mind boggling fact 1.** The proof is highly nonconstructive!

**Mind boggling fact 2.** The constant of proportionality is enormous!

---

"Unfortunately, for any instance \(G = (V, E)\) that one could fit into the known universe, one would easily prefer \(n^{10}\) to even constant time, if that constant had to be one of Robertson and Seymour’s." — David Johnson

---

**Theorem.** There exists an explicit \(O(n)\) algorithm.

**Practice.** LEDA implementation guarantees \(O(n^3)\).
Planar map 3-colorability

**Planar-Map-3-Color.** Given a planar map, can it be colored using 3 colors so that no two adjacent regions have the same color?

**Yes instance.**

**No instance.**
Planar graph and map 3-colorability reduce to one another

**Theorem.** PLANAR-3-COLOR \( \not\equiv_P \) PLANAR-MAP-3-COLOR.

**Pf.**

- Nodes correspond to regions.
- Two nodes are adjacent iff they share a nontrivial border.

---

Planar 3-colorability is NP-complete

**Lemma.** \( W \) is a planar graph such that:
- In any 3-coloring of \( W \), opposite corners have the same color.
- Any assignment of colors to the corners in which opposite corners have the same color extends to a 3-coloring of \( W \).

---

Planar 3-colorability is NP-complete

**Lemma.** \( W \) is a planar graph such that:
- In any 3-coloring of \( W \), opposite corners have the same color.
- Any assignment of colors to the corners in which opposite corners have the same color extends to a 3-coloring of \( W \).

**Pf.** The only 3-colorings (modulo permutations) of \( W \) are shown below. •
Planar 3-colorability is NP-complete

Construction. Given instance $G$ of 3-COLOR, draw $G$ in plane, letting edges cross. Form planar $G'$ by replacing each edge crossing with planar gadget $W$.

Lemma. $G$ is 3-colorable iff $G'$ is 3-colorable.
- In any 3-coloring of $W$, $a \neq a'$ and $b \neq b'$.
- If $a \neq a'$ and $b \neq b'$ then can extend to a 3-coloring of $W$.

Planar map $k$-colorability

Theorem. [Appel-Haken 1976] Every planar map is 4-colorable.
- Resolved century-old open problem.
- Used 50 days of computer time to deal with many special cases.
- First major theorem to be proved using computer.

Remarks.
- Appel-Haken yields $O(n^3)$ algorithm to 4-color of a planar map.
- Best known: $O(n^2)$ to 4-color; $O(n)$ to 5-color.
- Determining whether 3 colors suffice is NP-complete.

Planar 3-colorability is NP-complete

Construction. Given instance $G$ of 3-COLOR, draw $G$ in plane, letting edges cross. Form planar $G'$ by replacing each edge crossing with planar gadget $W$.

Lemma. $G$ is 3-colorable iff $G'$ is 3-colorable.
- In any 3-coloring of $W$, $a \neq a'$ and $b \neq b'$.
- If $a \neq a'$ and $b \neq b'$ then can extend to a 3-coloring of $W$.

Polynomial-time special cases NP-hard problems

Trees. VERTEX-COVER, INDEPENDENT-SET, DOMINATING-SET, GRAPH-ISOMORPHISM, ...

Bipartite graphs. VERTEX-COVER, 2-COLOR, ...

Chordal graphs. $K$-COLOR, CLIQUE, INDEPENDENT-SET, ...

Planar graphs. MAX-CUT, ISING, CLIQUE, GRAPH-ISOMORPHISM, 4-COLOR, ...

Bounded treewidth. 3-COLOR, HAM-CYCLE, INDEPENDENT-SET, GRAPH-ISOMORPHISM.

Small integers. KNAPSACK, PARTITION, SUBSET-SUM, ...
Approximation algorithms

\( \rho \)-approximation algorithm.
- Guaranteed to run in poly-time.
- Guaranteed to solve arbitrary instances of the problem.
- Guaranteed to find solution within ratio \( \rho \) of true optimum.

**Ex.** Given a graph \( G \), the greedy algorithms finds a VERTEX-COVER that uses \( \leq 2 \OPT(G) \) vertices in \( O(m+n) \) time.

**Challenge.** Need to prove a solution’s value is close to optimum value, without even knowing what optimum value is!

---

**Knapsack problem**

**Knapsack problem.**
- Given \( n \) objects and a knapsack.
- Item \( i \) has value \( v_i > 0 \) and weighs \( w_i > 0 \). \hspace{1cm} \( \text{we assume } w_i \leq W \text{ for each } i \)
- Knapsack has weight limit \( W \).
- Goal: fill knapsack so as to maximize total value.

**Ex:** \( \{ 3, 4 \} \) has value 40.

<table>
<thead>
<tr>
<th>item</th>
<th>value</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>18</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>22</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>28</td>
<td>7</td>
</tr>
</tbody>
</table>

original instance (\( W = 11 \))

---

**Knapsack is NP-complete**

**KNAPSACK.** Given a set \( X \), weights \( w_i \geq 0 \), values \( v_i \geq 0 \), a weight limit \( W \), and a target value \( V \), is there a subset \( S \subseteq X \) such that:

\[
\sum_{i \in S} w_i \leq W \\
\sum_{i \in S} v_i \geq V
\]

**SUBSET-SUM.** Given a set \( X \), values \( u_i \geq 0 \), and an integer \( U \), is there a subset \( S \subseteq X \) whose elements sum to exactly \( U \)?

**Theorem.** \( \text{SUBSET-SUM} \leq_P \text{KNAPSACK} \).

**Pf.** Given instance \( (u_1, \ldots, u_n, U) \) of \( \text{SUBSET-SUM} \), create \( \text{KNAPSACK} \) instance:

\[
v_i = w_i = u_i \quad \sum_{i \in S} u_i \leq U \\
V = W = U \quad \sum_{i \in S} u_i \geq U
\]
Knapsack problem: dynamic programming I

**Def.** $OPT(i, w) = \max$ value subset of items $1, \ldots, i$ with weight limit $w$.

**Case 1.** $OPT$ does not select item $i$.
- $OPT$ selects best of $1, \ldots, i-1$ using up to weight limit $w$.

**Case 2.** $OPT$ selects item $i$.
- New weight limit $= w - w_j$.
- $OPT$ selects best of $1, \ldots, i-1$ using up to weight limit $w - w_j$.

$$OPT(i, w) = \begin{cases} 0 & \text{if } i = 0 \\ OPT(i-1, w) & \text{if } w_i > w \\ \max \{ OPT(i-1, w), v_i + OPT(i-1, w - w_i) \} & \text{otherwise} \end{cases}$$

**Theorem.** Computes the optimal value in $O(nW)$ time.
- Not polynomial in input size.
- Polynomial in input size if weights are small integers.

Knapsack problem: dynamic programming II

**Def.** $OPT(i, v) = \min$ weight of a knapsack for which we can obtain a solution of value $\geq v$ using a subset of items $1, \ldots, i$.

**Note.** Optimal value is the largest value $v$ such that $OPT(i, v) \leq W$.

**Case 1.** $OPT$ does not select item $i$.
- $OPT$ selects best of $1, \ldots, i-1$ that achieves value $v$.

**Case 2.** $OPT$ selects item $i$.
- Consumes weight $w_i$ to achieve value $v - v_i$.
- $OPT$ selects best of $1, \ldots, i-1$ that achieves value $v - v_i$.

$$OPT(i, v) = \begin{cases} 0 & \text{if } v \leq 0 \\ \infty & \text{if } i = 0 \text{ and } v > 0 \\ \min \{ OPT(i-1, v), w_i + OPT(i-1, v - v_i) \} & \text{otherwise} \end{cases}$$

Knapsack problem: polynomial-time approximation scheme

**Intuition for approximation algorithm.**
- Round all values up to lie in smaller range.
- Run dynamic programming algorithm II on rounded instance.
- Return optimal items in rounded instance.

<table>
<thead>
<tr>
<th>item</th>
<th>value</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>934221</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>5956342</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>17810013</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>21217800</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>27343199</td>
<td>7</td>
</tr>
</tbody>
</table>

original instance ($W = 11$)

<table>
<thead>
<tr>
<th>item</th>
<th>value</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>18</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>22</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>28</td>
<td>7</td>
</tr>
</tbody>
</table>

rounded instance ($W = 11$)
Knapsack problem: polynomial-time approximation scheme

Round up all values:
- $v_{\text{max}}$ = largest value in original instance.
- $\varepsilon$ = precision parameter.
- $\theta$ = scaling factor = $v_{\text{max}} / n$.

Observation. Optimal solutions to problem with $v$ are equivalent to optimal solutions to problem with $\hat{v}$.

Intuition. $v$ close to $v$ so optimal solution using $v$ is nearly optimal; $\hat{v}$ small and integral so dynamic programming algorithm II is fast.

Knapsack problem: polynomial-time approximation scheme

Theorem. For any $\varepsilon > 0$, the rounding algorithm computes a feasible solution whose value is within a $(1 + \varepsilon)$ factor of the optimum in $O(n^2 / \varepsilon)$ time.

Pf.
- We have already proved the accuracy bound.
- Dynamic program II running time is $O(n^2 \hat{v}_{\text{max}})$, where

$$\hat{v}_{\text{max}} = \left\lceil \frac{v_{\text{max}}}{\theta} \right\rceil = \left\lceil \frac{n}{\varepsilon} \right\rceil$$

PTAS. $(1 + \varepsilon)$-approximation algorithm for any constant $\varepsilon > 0$.
- Produces arbitrarily high quality solution.
- Trades off accuracy for time.
- But such algorithms are unlikely to exist for certain problems...

Knapsack problem: polynomial-time approximation scheme

Round up all values: $v_i = \left\lceil \frac{v_i}{\theta} \right\rceil \theta$, $\hat{v}_i = \left\lceil \frac{v_i}{\theta} \right\rceil$

Theorem. If $S$ is solution found by rounding algorithm and $S^*$ is any other feasible solution, then

$$(1 + \varepsilon) \sum_{i \in S^*} v_i \geq \sum_{i \in S} v_i$$

Pf. Let $S^*$ be any feasible solution satisfying weight constraint.

$$\sum_{i \in S^*} v_i \leq \sum_{i \in S} v_i$$
$$\leq \sum_{i \in S} (v_i + \theta)$$
$$\leq \sum_{i \in S} v_i + n\theta$$
$$\leq (1 + \varepsilon) \sum_{i \in S} v_i$$

Inapproximability

MAX-3-SAT. Given a 3-SAT instance $\Phi$, find an assignment that satisfies the maximum number of clauses.

Theorem. [Karloff-Zwick 1997] There exists a $\frac{7}{8}$-approximation algorithm.

Theorem. [Håstad 2001] Unless $P = NP$, there does not exist a $p$-approximation for any $p > \frac{7}{8}$.

A $7/8$-Approximation Algorithm for MAX 3SAT?

Howard Karloff*  
Uri Zwick  

We describe a randomized approximation algorithm which takes as input a MAX 3SAT instance. For any instance—a collection of clauses each of length at most three—is satisfiable, then the expected weight of the assignment found is at least $\frac{7}{8}$ of optimal. We provide strong evidence that no other algorithm performs equally well on arbitrary MAX 3SAT instances.

Some Optimal Inapproximability Results

JOHAN HÅSTAD
Royal Institute of Technology, Stockholm, Sweden

Abstract. We prove optimal, up to an arbitrary $\varepsilon > 0$, inapproximability results for Max-3-SAT for $\varepsilon > 0$, increasing the number of satisfiable linear equations in an over-determined system of linear equations modulo a prime $p$ and for Max-CUT. As a consequence of these results we get improved lower bounds for the efficient approximability of many optimization problems studied previously. In particular, for Max-E2-Cut, Max-Cut, Max-Dicut, and vertex cover.

Categories and Subject Descriptors: F.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems
Exact algorithms for 3-satisfiability

**Brute force.** Given a 3-SAT instance with \( n \) variables and \( m \) clauses, the brute-force algorithm takes \( O((m + n) 2^n) \) time.

**Pf.**
- There are \( 2^n \) possible truth assignments to the \( n \) variables.
- We can evaluate a truth assignment in \( O(m + n) \) time.

**Exact algorithms for 3-satisfiability**

**A recursive framework.** A 3-SAT formula \( \Phi \) is either empty or the disjunction of a clause \((\ell_1 \lor \ell_2 \lor \ell_3)\) and a 3-SAT formula \( \Phi' \) with one fewer clause.

\[
\Phi = (\ell_1 \lor \ell_2 \lor \ell_3) \land \Phi' = (\ell_1 \land \Phi') \lor (\ell_2 \land \Phi') \lor (\ell_3 \land \Phi')
\]

\[
(\Phi' \mid \ell_1 = true) \lor (\Phi' \mid \ell_2 = true) \lor (\Phi' \mid \ell_3 = true)
\]

**Notation.** \( \Phi \mid x = true \) is the simplification of \( \Phi \) by setting \( x \) to \( true \).

**Ex.**
- \( \Phi = (x \lor y \lor \neg z) \land (x \lor \neg y \lor z) \land (w \lor y \lor \neg z) \land (\neg x \lor y \lor z) \).
- \( \Phi' = (x \lor \neg y \lor z) \land (w \lor y \lor \neg z) \land (\neg x \lor y \lor z) \).
- \( (\Phi' \mid x = true) = (w \lor y \lor \neg z) \land (y \lor z) \).

Each clause has \( \leq 3 \) literals.
Exact algorithms for 3-satisfiability

A recursive framework. A 3-SAT formula $\Phi$ is either empty or the disjunction of a clause $(i \lor j \lor k)$ and a 3-SAT formula $\Phi'$ with one fewer clause.

3-SAT ($\Phi$)

- If $\Phi$ is empty \textbf{RETURN} $true$.
- $(i \lor j \lor k) \land \Phi' \leftarrow \Phi$.
- If 3-SAT($\Phi'$ | $i = true$) \textbf{RETURN} $true$.
- If 3-SAT($\Phi'$ | $i = true$) \textbf{RETURN} $true$.
- If 3-SAT($\Phi'$ | $i = true$) \textbf{RETURN} $true$.
- \textbf{RETURN} $false$.

Theorem. The brute-force 3-SAT algorithm takes $O(poly(n) 3^n)$ time.

Pf. $T(n) \leq 3T(n - 1) + poly(n)$.

Exact algorithms for 3-satisfiability

Theorem. The brute-force algorithm takes $O(1.84^n)$ time.

Pf. $T(n) \leq T(n - 1) + T(n - 2) + T(n - 3) + O(m + n)$.

Key observation. The cases are not mutually exclusive. Every satisfiable assignment containing clause $(i \lor j \lor k)$ must fall into one of 3 classes:

- $i$ is $true$.
- $i$ is $false$; $j$ is $true$.
- $i$ is $false$; $j$ is $false$; $k$ is $true$.

3-SAT ($\Phi$)

- If $\Phi$ is empty \textbf{RETURN} $true$.
- $(i \lor j \lor k) \land \Phi' \leftarrow \Phi$.
- If 3-SAT($\Phi'$ | $i = true$) \textbf{RETURN} $true$.
- If 3-SAT($\Phi'$ | $i = true$) \textbf{RETURN} $true$.
- If 3-SAT($\Phi'$ | $i = true$) \textbf{RETURN} $true$.
- \textbf{RETURN} $false$.

A Full Derandomization of Schöning’s $k$-SAT Algorithm

Robin A. Moser and Dominik Scheder
Institute for Theoretical Computer Science
Department of Computer Science
ETH Zürich, 8092 Zürich, Switzerland
{robin.moser, dominik.scheder}@inf.ethz.ch
August 25, 2010

Abstract

Schöning [7] presents a simple randomized algorithm for $k$-SAT with running time $O(n 20^k poly(n))$ for $c_k = 20^k - 1/2$. We give a deterministic version of this algorithm running in time $O(n^2 r^4 poly(n))$, where $r > 0$ can be made arbitrarily small.
Exact algorithms for satisfiability

**Chaff.** State-of-the-art SAT solver.

- Solves real-world SAT instances with ~ 10K variable.
- Developed at Princeton by undergrads.

**Chaff: Engineering an Efficient SAT Solver**

Matthew W. Moskewicz
Department of EECS
UC Berkeley
moskewicz@alum.mit.edu

Conor F. Madigan
Department of EECS
MIT
omatgin@mit.edu

Ying Zhao,Lintao Zhang, Sharad Malik
Department of Electrical Engineering
Princeton University
(yingzhao, lintao, sharad)@ee.princeton.edu

**ABSTRACT**

Boolean Satisfiability is probably the most studied of combinatorial optimization/search problems. Significant effort has been devoted to trying to provide practical solutions to this problem for problem instances encountered in a range of applications in Electronic Design Automation (EDA), as well as in Artificial Intelligence (AI). This study has culminated in the

**Exact algorithms for satisfiability**

**DPPL algorithm.** Highly-effective backtracking procedure.

- Splitting rule: assign truth value to literal; solve both possibilities.
- Unit propagation: clause contains only a single unassigned literal.
- Pure literal elimination: if literal appears only negated or unnegated.

**Exact algorithms for TSP and Hamilton cycle**

**Theorem.** The brute-force algorithm for TSP (or HAM-CYCLE) takes \(O(n!)\) time.

**Pf.**

- There are \(\frac{1}{2} (n-1)!\) tours.
- Computing the length of a tour takes \(O(n)\) time.

**Note.** The function \(n!\) grows exponentially faster than \(2^n\).

\[2^{40} = 109951162776 \approx 10^{12},\]

\[40! = 81591283247897734346112695611589427200000000 \approx 10^{48}.\]
Euclidean traveling salesperson problem

Theorem. [Bellman 1962, Held-Karp 1962] There exists a $O(n^2 2^n)$ time algorithm for TSP (and HAMILTON-CYCLE).

Pf. [dynamic programming]

- Define $c(s, v, X) =$ cost of cheapest path between $s$ and $v$ that visits every node in $X$ exactly once (and uses only nodes in $X$).
- Observe $OPT = \min_{v \in X} c(s, v, V) + c(v, s)$.
- There are $n 2^n$ subproblems and they satisfy the recurrence:

$$c(s, v, X) = \begin{cases} c(s, v) & \text{if } |X| = 2, \\ \min_{u \in X \setminus \{s, v\}} c(s, u, X \setminus \{v\}) + c(u, v) & \text{if } |X| > 2. \end{cases}$$

- The values $c(s, v, X)$ can be computed increasing order of the cardinality of $X$.

Euclidean traveling salesperson problem

Euclidean TSP. Given $n$ points in the plane and a real number $L$, is there a tour that visits every city exactly once that has distance $\leq L$?

Proposition. EUCLIDEAN-TSP is NP-hard.

Remark. Not known to be in NP.

\[ \sqrt{3} + \sqrt{4} + \sqrt{8} < \sqrt{4} + \sqrt{4} + \sqrt{8} \]

8.92819407 < 8.928203230

Euclidean traveling salesperson problem

Theorem. [Arora 1998, Mitchell 1999] Given $n$ points in the plane, for any constant $\varepsilon > 0$, there exists a poly-time algorithm to find a tour whose length is at most $(1 + \varepsilon)$ times that of the optimal tour.

Pf idea. Structure theorem + dynamic programming.
Concorde TSP solver. [Applegate-Bixby-Chvátal-Cook]

- Linear programming + branch-and-bound + polyhedral combinatorics.
- Greedy heuristics, including Lin-Kernighan.
- MST, Delaunay triangulations, fractional b-matchings, ...

Remarkable fact. Concorde has solved all 110 TSPLIB instances.

That’s all, folks: keep searching!

That's all, folks: keep searching!

I have been hard working for so long.
I swear it’s right, and he marks it wrong.
Some how I’ll feel sorry when it’s done: GPA 2.1
Is more than I hope for.

Garey, Johnson, Karp and other men (and women)
Tried to make it order N log N.
Am I a mad fool
If I spend my life in grad school,
Forever following the longest path?

Woh-oh-oh-oh, find the longest path!
Woh-oh-oh-oh, find the longest path!
Woh-oh-oh-oh, find the longest path!

Written by Dan Barrett in 1988 while a student
at Johns Hopkins during a difficult algorithms take-home final