Teoría de algoritmos

Capítulo 1

Justo Puerto,
Dep. Estadística e I.O. Univ. Sevilla

UNIVERSIDAD
DE SEVILLA

Properties of Algorithms

A given problem can be solved by many different algorithms. Which ALGORITHMS will be useful in practice?

A working definition: (Jack Edmonds, 1962)

- Efficient: polynomial time for ALL inputs.
- Inefficient: "exponential time" for SOME inputs.

Robust definition has led to explosion of useful algorithms for wide spectrum of problems.

- Notable exception: simplex algorithm.

Our goal is to find **efficient** algorithms for some problems.
What does *"efficient algorithm"* mean?

In our framework we looks for algorithms that run **as fast as** possible

The computation model must be **computer independent** so that time is referred to elemental operations.

- Access to memory is an operation (reading, writing)
- Real arithmetic operations (sum, substraction, multiplication, division) and comparisons are elemental operations.
- Some elemental function evaluations (trigonometric, n-th radical ...) can be considered elemental operations.

We are only interested in:

- Worst case behavior (there are other ways to analyze problem complexity: average, best case ...)
- Asymptotic analysis when the data size goes to infinity: $O(|\text{size}|)$ notation.

Assume that one elemental operation takes 10^{-6} seconds.

	100	1000	10000	100000
$O(n)$	$10^{-4}s.$	$10^{-3}s.$	10^{-2} s.	10^{-1} s.
$O(n\log n)$	$2\times10^{-4}s.$	$3\times10^{-3}s.$	4×10^{-2} s.	0.5 s.
$O(n^2)$	10^{-2} s.	1 s.	1 m. 40 s.	2.77 h.
$O(n^3)$	1 s.	≈ 17 m.	11.5 d.	31,7 y.
$O(n^4)$	1m. 40 s.	11.5 d.	31.7 y.	3×10^6 y.

Exponential Growth

Exponential growth dwarfs technological change.

- Suppose each electron in the universe had power of today's supercomputers.
- And each works for the life of the universe in an effort to solve TSP problem using N! algorithm.

Some Numbers	
Quantity	**Number**
Home PC instructions / second	10^9
Supercomputer instructions / second	10^{12}
Seconds per year	10^9
Age of universe †	10^{13}
Electrons in universe †	10^{79}

† Estimated

- Will not succeed for 1,000 city TSP!

$$1000! \gg 10^{1000} \gg 10^{79} \times 10^{13} \times 10^{9} \times 10^{12}$$

Properties of Problems

Which PROBLEMS will we be able to solve in practice?

- Those with efficient algorithms.
- How can I tell if I am trying to solve such a problem?
  - Theory of NP-completeness helps.

Yes	Probably No
Shortest path	Longest path
Euler cycle	Hamiltonian cycle
Min cut	Max cut
2-SAT	3-SAT
PLANAR-2-COLOR	PLANAR-3-COLOR
PLANAR-4-COLOR	PLANAR-3-COLOR
Matching	3D-Matching
Baseball elimination	Soccer elimination
Bipartite vertex cover	Vertex cover

Unknown
Factoring
Graph isomorphism

P

Decision problem X.

- X is a (possibly infinite) set of binary strings.
- Instance: finite binary string s, of length |s|.
- Algorithm A solves X if A(s) = YES $\Leftrightarrow$ s $\in$ X.

Polynomial time.

- Algorithm A runs in polynomial-time if for every instance s, A terminates in at most p(s) "steps", where p is some polynomial.

Definition of P.

- Set of all decision problems solvable in polynomial time on a deterministic Turing machine.

Examples:

- MULTIPLE: Is the integer y a multiple of x?
- RELPRIME: Are the integers x and y relatively prime?
- PERFECT-MATCHING: Given graph G, is there a perfect matching?

Strong Church-Turing Thesis

Definition of P fundamental because of SCT.

Strong Church-Turing thesis:

- P is the set of decision problems solvable in polynomial time on REAL computers.

Evidence supporting thesis:

- True for all physical computers.
- Can create deterministic TM that efficiently simulates any real general-purpose machine (and vice versa).

Possible exception?

- Quantum computers: no conventional gates.

Efficient Certification

Certification algorithm.

- Design an algorithm that checks whether proposed solution is a YES instance.

Algorithm C is an efficient certifier for X if:

- C is a polynomial-time algorithm that takes two inputs s and t.
- There exists a polynomial p( ) so that for every string s, $s \in X \Leftrightarrow$ there exists a string t such that $|t| \leq p(|s|)$ and C(s, t) = YES.

Intuition.

- Efficient certifier views things from "managerial" viewpoint.
- It doesn't determine whether $s \in X$ on its own.
- Rather, it evaluates a proposed proof t that $s \in X$.
- Accepts if and only if given a "short" proof of this fact.

Justo Puerto,
Dep. Estadística e I.O. Univ. Sevilla

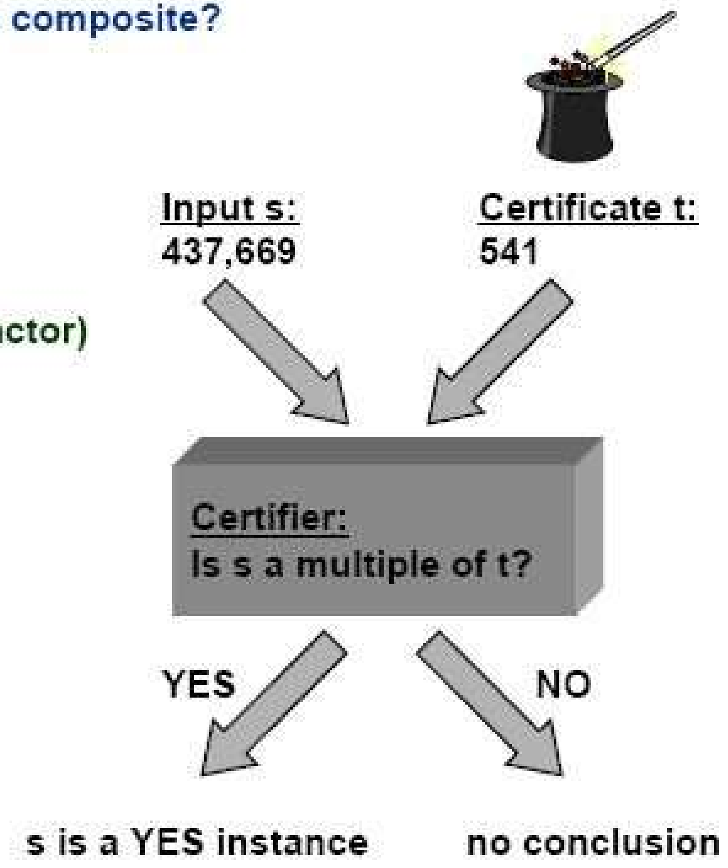
UNIVERSIDAD
DE SEVILLA

Certifiers and Certificates

COMPOSITE: Given integer s, is s composite?

Observation. s is composite ⇔ there exists an integer $1 < t < s$ such that s is a multiple of t.

- YES instance: s = 437,669.
  - certificate t = 541 or 809 (a factor)



Justo Puerto,
Dep. Estadística e I.O. Univ. Sevilla

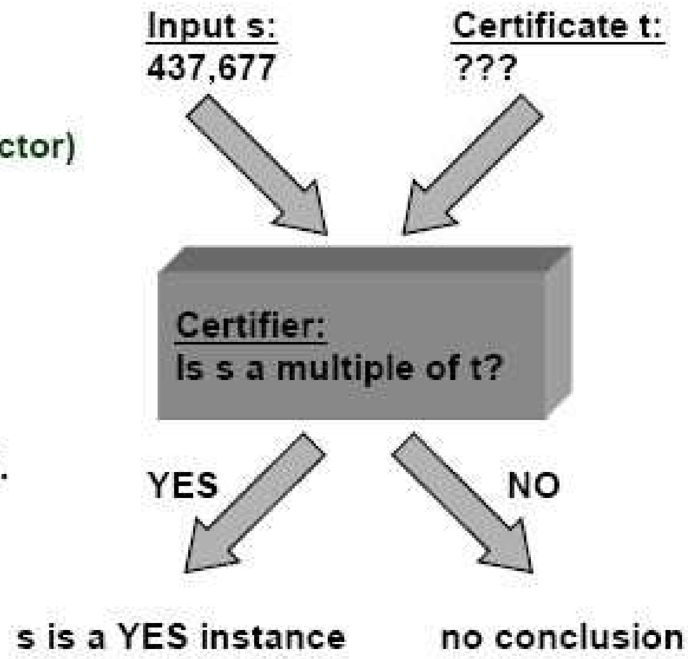
UNIVERSIDAD
DE SEVILLA

Certifiers and Certificates

COMPOSITE: Given integer s, is s composite?

Observation. s is composite ⇔ there exists an integer $1 < t < s$ such that s is a multiple of t.

- YES instance: s = 437,669.
  - certificate t = 541 or 809 (a factor)
- NO instance: s = 437,677.
  - no witness can fool verifier into saying YES
- Conclusion: COMPOSITE $\in$ NP.



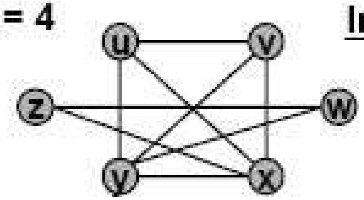
Justo Puerto,
Dep. Estadística e I.O. Univ. Sevilla

UNIVERSIDAD
DE SEVILLA

Certifiers and Certificates

CLIQUE: Given an undirected graph, is there a subset S of k nodes such that there is an arc connecting every pair of nodes in S?

k = 4



Input s:

Certificate t:
{u, v, w, x}

Certifier:
1. For all pairs of nodes v and w in t, check that (v, w) is an edge in the graph.
2. Check that number of nodes in t $\leq$ k.

YES

NO

s is a YES instance

no conclusion

CLIQUE $\in$ NP.

Certifiers and Certificates

3-COLOR: Given planar map, can it be colored with 3 colors?

Input s:

Certificate t:

Certifier:
1. Check that s and t describe same map.
2. Count number of distinct colors in t.
3. Check all pairs of adjacent states.

YES → s is a YES instance

NO → no conclusion

3-COLOR $\in$ NP.

Justo Puerto,
Dep. Estadística e I.O. Univ. Sevilla

UNIVERSIDAD
DE SEVILLA

NP

Definition of NP:

- Does NOT mean "not polynomial."

Definition of NP:

- Set of all decision problems for which there exists an efficient certifier.
- Definition important because it links many fundamental problems.

Claim: P $\subseteq$ NP.

Proof: Consider problem X $\in$ P.

- Then, there exists efficient algorithm A(s) that solves X.
- Efficient certifier B(s, t): return A(s).

NP

Definition of EXP:

- Set of all decision problems solvable in exponential time on a deterministic Turing machine.

Claim: NP $\subseteq$ EXP.

Proof: Consider problem X $\in$ NP.

- Then, there exists efficient certifier C(s, t) for X.
- To solve input s, run C(s, t) on all strings t with $|t| \leq p(|s|)$.
- Return YES, if C(s, t) returns YES for any of these.

Useful alternate definition of NP:

- Set of all decision problems solvable in polynomial time on a NONDETERMINISTIC Turing machine.
- Intuition: act of searching for t is viewed as a non-deterministic search over the space of possible proofs. Nondeterministic TM can try all possible solutions in parallel.

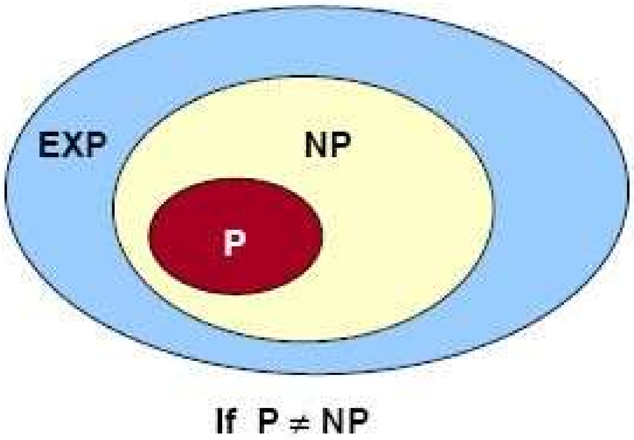
The Main Question

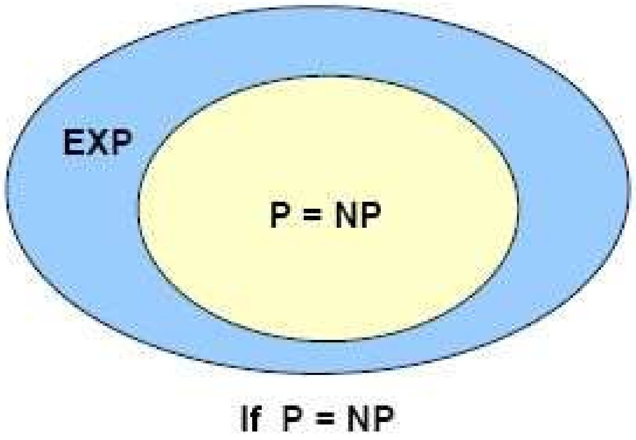
Does P = NP? (Edmonds, 1962)

- Is the original DECISION problem as easy as CERTIFICATION?
- Does nondeterminism help you solve problems faster?

Most important open problem in computer science.

- If yes, staggering practical significance.
- Clay Foundation Millennium $1 million prize.





The Main Question

Generator (P)

- Factor integer s.
- Color a map with minimum # colors.
- Design airfoil of minimum drag.
- Prove a beautiful theorem.
- Write a beautiful sonnet.
- Devise a good joke.
- Vinify fine wine.
- Orate a good lecture.
- Ace an exam.

Certifier (NP)

- Is s a multiple of t?
- Check if all adjacent regions have different colors.
- Compute drag of airfoil.
- Understand its proof.
- Appreciate it.
- Laugh at it.
- Be a wine snob.
- Know when you've heard one.
- Verify TA's solutions.

Imagine the wealth of a society that produces optimal planes, bridges, rockets, theorems, art, music, wine, jokes.

The Main Question

Does P = NP?

- Is the original DECISION problem as easy as CERTIFICATION?

If yes, then:

- Efficient algorithms for 3-COLOR, TSP, FACTOR, . . .
- Cryptography is impossible (except for one-time pads) on conventional machines.
- Modern banking system will collapse.
- Harmonial bliss.

If no, then:

- Can't hope to write efficient algorithm for TSP.
- But maybe efficient algorithm still exists for FACTOR . . . .

The Main Question

Does P = NP?

- Is the original DECISION problem as easy as CERTIFICATION?

Probably no, since:

- Thousands of researchers have spent four frustrating decades in search of polynomial algorithms for many fundamental NP problems without success.
- Consensus opinion: P $\neq$ NP.

But maybe yes, since:

- No success in proving P $\neq$ NP either.

Polynomial Transformation

Problem X polynomial reduces (Cook-Turing) to problem Y ($X \leq_P Y$) if arbitrary instances of problem X can be solved using:

- Polynomial number of standard computational steps, plus
- Polynomial number of calls to oracle that solves problem Y.

Problem X polynomial transforms (Karp) to problem Y if given any input x to X, we can construct an input y such that x is a YES instance of X if and only if y is a YES instance of Y.

- We require |y| to be of size polynomial in |x|.

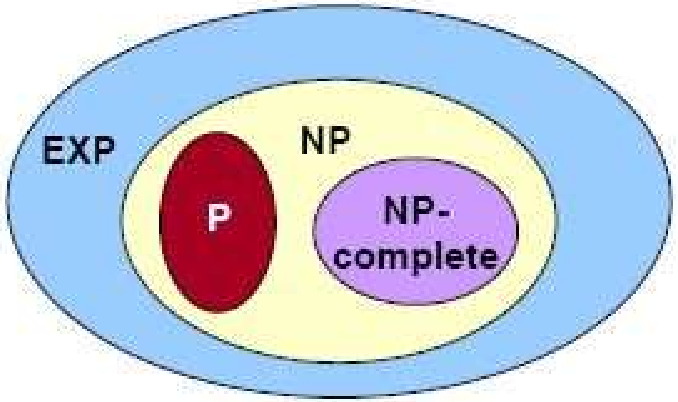
Polynomial transformation is polynomial reduction with just one call to oracle for Y, exactly at the end of the algorithm for X.

Note: all previous reductions were of this form!

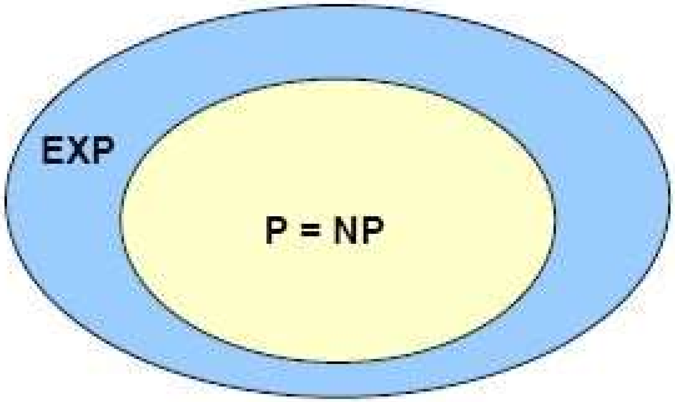
NP-Complete

Definition of NP-complete:

- A problem Y in NP with the property that for every problem X in NP, X polynomial transforms to Y.
- "Hardest computational problems" in NP.



If $P \neq NP$



If $P = NP$

NP-Complete

Definition of NP-complete:

- A problem Y in NP with the property that for every problem X in NP, X polynomial transforms to Y.

Significance.

- Efficient algorithm for any NP-complete problem $\Rightarrow$ efficient algorithm for every other problem in NP.
- Links together a huge and diverse number of fundamental problems:
  - TSP, 3-COLOR, CNF-SAT, CLIQUE, ...
- Can implement any computer program in 3-COLOR.

Notorious complexity class.

- Only exponential algorithms known for these problems.
- Called "intractable" - unlikely that they can be solved given limited computing resources.

Some NP-Complete Problems

Most natural problems in NP are either in P or NP-complete.

Six basic genres and paradigmatic examples of NP-complete problems.

- Packing problems: SET-PACKING, INDEPENDENT-SET.
- Covering problems: SET-COVER, VERTEX-COVER.
- Sequencing problems: HAMILTONIAN-CYCLE, TSP.
- Partitioning problems: 3-COLOR, CLIQUE.
- Constraint satisfaction problems: SAT, 3-SAT.
- Numerical problems: SUBSET-SUM, PARTITION, KNAPSACK.

Caveat: FACTOR not known to be NP-complete.

Prime is in P

Establishing NP-Completeness

Definition of NP-complete:

- A problem $Y \in$ NP with the property that for every problem X in NP, X polynomial transforms to Y.

Cook's theorem. CNF-SAT is NP-complete.

Recipe to establish NP-completeness of problem Y.

- Step 1. Show that $Y \in$ NP.
- Step 2. Show that CNF-SAT (or any other NP-complete problem) transforms to Y.

Example: CLIQUE is NP-complete.

✓ Step 1. CLIQUE $\in$ NP.

✓ Step 2. CNF-SAT polynomial transforms to CLIQUE.

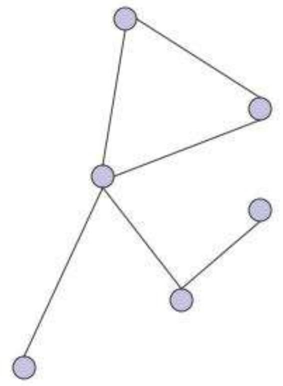
CLIQUE AND 3-SAT: 1

- Satisfiability problem (SAT):
  - Given: a formula φ with m clauses $C_1,\ldots,C_m$ over n variables.

    Example: $x_1 \vee x_2 \vee x_5$, $x_3 \vee \neg x_5$
  - Check if there exists TRUE/FALSE assignments to the variables that makes the formula satisfiable

CLIQUE AND 3-SAT: 2

- Clique:
  - Input: undirected graph $G=(V,E)$, K
  - Output: is there a subset C of V, $|C| \geq K$, such that every pair of vertices in C has an edge between them



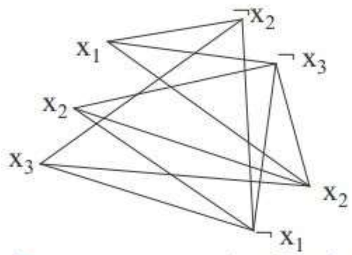
Justo Puerto,
Dep. Estadística e I.O. Univ. Sevilla

CLIQUE AND 3-SAT: 3

- Given a SAT formula $\varphi = C_1, \ldots, C_m$ over $x_1, \ldots, x_n$, we need to produce $G=(V,E)$ and K, such that φ satisfiable iff G has a clique of size $\geq K$.
- Notation: a literal is either x_i or $\neg x_i$
- For each literal t occurring in φ, create a vertex v_t
- Create an edge $v_t - v_{t'}$ iff:
  - t and t' are not in the same clause, and
  - t is not the negation of t'

CLIQUE AND 3-SAT: 4

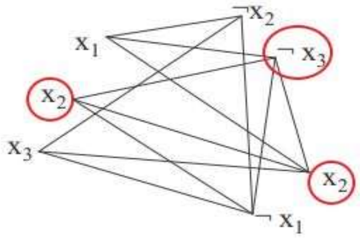
- Formula: $x_1 \vee x_2 \vee x_3\, ,\ \neg x_2 \vee \neg x_3,\ \neg x_1 \vee x_2$
- Graph:



- Claim: φ satisfiable iff G has a clique of size $\geq m$

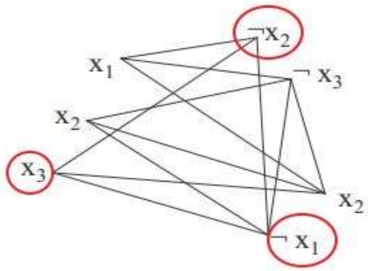
- "→" part:
  - Take any assignment that satisfies φ.

    E.g., x_1=F, x_2=T, x_3=F
  - Let the set C contain one satisfied literal per clause
  - C is a clique



- "←" part:
  - Take any clique C of size ≥ m (i.e., = m)
  - Create a set of equations that satisfies selected literals.

    E.g., x_3=T, x_2=F, x_1=F
  - The set of equations is consistent and the solution satisfies φ



Justo Puerto,
Dep. Estadística e I.O. Univ. Sevilla

UNIVERSIDAD
DE SEVILLA

CONCLUSION

- We constructed a reduction that maps:
  - YES inputs to SAT to YES inputs to Clique
  - NO inputs to SAT to NO inputs to Clique
- The reduction works in poly time
- Therefore, SAT $\leq$ Clique $\rightarrow$Clique NP-hard
- Clique is in NP $\rightarrow$ Clique is NP-complete

Algoritmos aproximados
Capítulo 2

- Approximation Algorithms for NP-Complete Problems
  - Approximation ratios
  - Polynomial-Time Approximation Schemes
  - 2-Approximation for Vertex Cover
  - 2-Approximation for TSP special case
  - Log n-Approximation for Set Cover

Approximation Ratios

- Optimization Problems
  - We have some problem instance x that has many feasible "solutions".
  - We are trying to minimize (or maximize) some cost function c(S) for a "solution" S to x. For example,
    - Finding a minimum spanning tree of a graph
    - Finding a smallest vertex cover of a graph
    - Finding a smallest traveling salesperson tour in a graph
- An approximation produces a solution T
  - T is a **k-approximation** to the optimal solution OPT if $c(T)/c(OPT) \leq k$ (assuming a min. prob.; a maximization approximation would be the reverse)

Polynomial-Time Approximation Schemes

- A problem L has a **polynomial-time approximation scheme (PTAS)** if it has a polynomial-time $(1+\varepsilon)$-approximation algorithm, for any fixed $\varepsilon > 0$ (this value can appear in the running time).
- 0/1 Knapsack has a PTAS, with a running time that is $O(n^3/\varepsilon)$.

Knapsack is NP-Hard

KNAPSACK: Given a finite set X, nonnegative weights w_i, nonnegative values v_i, a weight limit W, and a desired 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 finite set X, nonnegative values u_i, and an integer t, is there a subset $S \subseteq X$ whose elements sum to t?

Claim. SUBSET-SUM $\leq_P$ KNAPSACK.

Proof: Given instance (X, t) of SUBSET-SUM, create KNAPSACK instance:

- $v_i = w_i = u_i$
- $V = W = t$

$$\sum_{i \in S} u_i \leq t$$
$$\sum_{i \in S} u_i \geq t$$

Justo Puerto,
Dep. Estadística e I.O. Univ. Sevilla

UNIVERSIDAD
DE SEVILLA

Knapsack: Dynamic Programming Solution 2

OPT(n, v) = min knapsack weight that yields value exactly v using subset of items {1, . . . , n}.

- Case 1: OPT selects item n.
  - new value needed = $v - v_n$
  - OPT selects best of {1, 2, . . . , n – 1} using new value
- Case 2: OPT does not select item n.
  - OPT selects best of {1, 2, . . . , n – 1} that achieves value v

$$OPT(n,v) = \begin{cases} 0 & \text{if } n = 0 \\ OPT(n-1, v) & \text{if } v_n > v \\ \min\{OPT(n-1, v),\ \ w_n + OPT(n-1, v - v_n)\} & \text{otherwise} \end{cases}$$

Directly leads to O(N V *) time algorithm.

- V* = optimal value.
- Not polynomial in input size!

Knapsack: FPTAS

Intuition for approximation algorithm.

- Round all values down to lie in smaller range.
- Run O(N V*) dynamic programming algorithm on rounded instance.
- Return optimal items in rounded instance.

Item	Value	Weight
1	134,221	1
2	656,342	2
3	1,810,013	5
4	22,217,800	6
5	28,343,199	7

W = 11

Original Instance

Item	Value	Weight
1	1	1
2	6	2
3	18	5
4	222	6
5	283	7

W = 11

Rounded Instance

Knapsack: FPTAS

Knapsack FPTAS.

- Round all values: $\bar{v}_n = \left\lfloor \frac{v_n}{\theta} \right\rfloor$
  - V = largest value in original instance
  - ε = precision parameter
  - θ = scaling factor = ε V / N

- Bound on optimal value V *:

$$V \le V^* \le N V$$

assume $w_n \le W$ for all n

Running Time

$$\begin{aligned} O(N\,\bar{V}^*) &\in O(N\,(N\,\bar{V})) \\ &\in O(N^2\,(V/\theta)) \\ &\in O(N^3\,\tfrac{1}{\varepsilon}) \end{aligned}$$

$\bar{V}$ = largest value in rounded instance

$\bar{V}^*$ = optimal value in rounded instance

Knapsack: FPTAS

Knapsack FPTAS.

- Round all values: $\overline{v_n} = \left\lfloor \frac{v_n}{\theta} \right\rfloor$
  - V = largest value in original instance
  - ε = precision parameter
  - θ = scaling factor = ε V / N

- Bound on optimal value V *:

$$V \leq V^* \leq N V$$

$\boldsymbol{S}^*$ = opt set of items in original instance

$\overline{\boldsymbol{S}^*}$ = opt set of items in rounded instance

Proof of Correctness

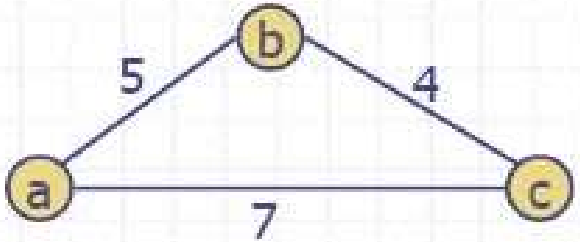
$$\begin{aligned}
\sum_{n \in \overline{S^*}} v_n &\geq \sum_{n \in \overline{S^*}} \theta \overline{v_n} \\
&\geq \sum_{n \in S^*} \theta \overline{v_n} \\
&\geq \sum_{n \in S^*} (v_n - \theta) \\
&\geq \sum_{n \in S^*} v_n - \theta N \\
&= V^* - (\varepsilon V / N) N \\
&\geq (1 - \varepsilon) V^*
\end{aligned}$$

Justo Puerto,
Dep. Estadística e I.O. Univ. Sevilla

UNIVERSIDAD
DE SEVILLA

Special Case of the Traveling Salesperson Problem

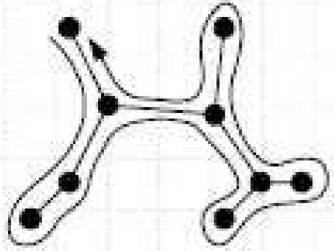
- **OPT-TSP**: Given a complete, weighted graph, find a cycle of minimum cost that visits each vertex.
  - OPT-TSP is NP-hard
  - Special case: edge weights satisfy the triangle inequality (which is common in many applications):
    - $w(a,b) + w(b,c) \geq w(a,c)$



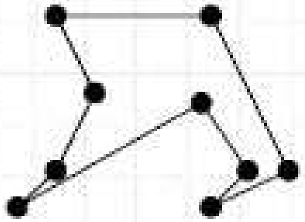
Justo Puerto,
Dep. Estadística e I.O. Univ. Sevilla

UNIVERSIDAD
DE SEVILLA

A 2-Approximation for TSP Special Case



Euler tour P of MST M



Output tour T

Algorithm *TSPApprox*(G)

Input weighted complete graph G, satisfying the triangle inequality

Output a TSP tour T for G

$M \leftarrow$ a minimum spanning tree for G

$P \leftarrow$ an Euler tour traversal of M, starting at some vertex s

$T \leftarrow$ empty list

for each vertex v in P (in traversal order)

  if this is v's first appearance in P **then**

    T.*insertLast*(v)

T.*insertLast*(s)

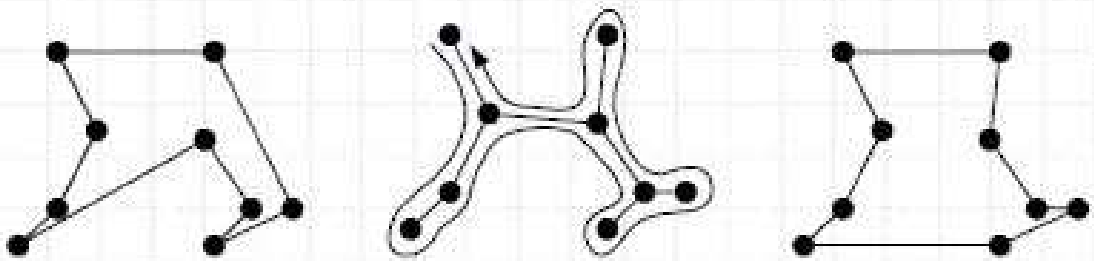
return T

Justo Puerto,
Dep. Estadística e I.O. Univ. Sevilla

UNIVERSIDAD
DE SEVILLA

A 2-Approximation for TSP Special Case - Proof

- The optimal tour is a spanning tour; hence $|M| \le |OPT|$.
- The Euler tour P visits each edge of M twice; hence $|P|=2|M|$
- Each time we shortcut a vertex in the Euler Tour we will not increase the total length, by the triangle inequality ($w(a,b) + w(b,c) \ge w(a,c)$); hence, $|T| \le |P|$.
- Therefore, $|T| \le |P| = 2|M| \le 2|OPT|$



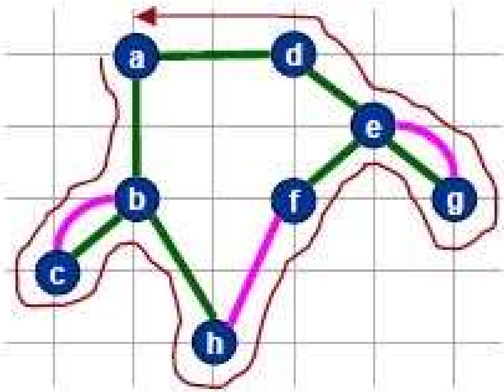
Output tour T (at most the cost of P) | Euler tour P of MST M (twice the cost of M) | Optimal tour OPT (at least the cost of MST M)

TSP: Christofides Algorithm

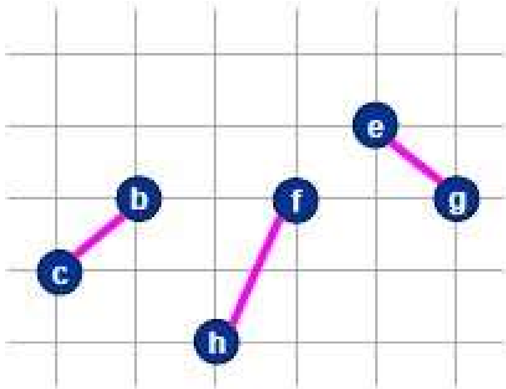
Theorem. There exists a 1.5-approximation algorithm for Δ-TSP.

CHRISTOFIDES(G, c)

- Find a minimum spanning tree T for (G, c).
- M $\leftarrow$ min cost perfect matching of odd degree nodes in T.
- G' $\leftarrow$ union of spanning tree and matching edges.
- E $\leftarrow$ Eulerian tour in G'.



E = Eulerian tour in G'



Matching M

Justo Puerto,
Dep. Estadística e I.O. Univ. Sevilla

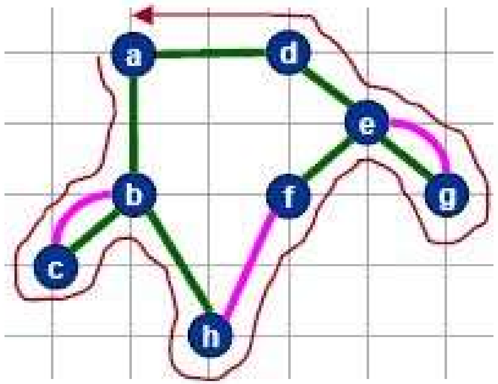
UNIVERSIDAD
DE SEVILLA

TSP: Christofides Algorithm

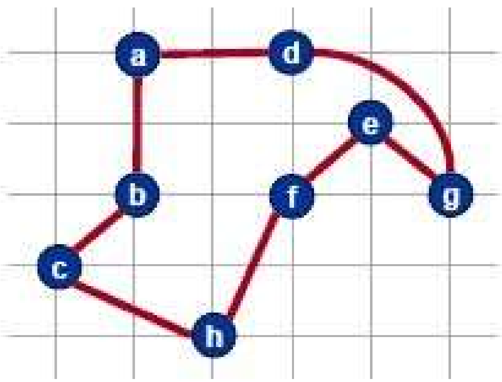
Theorem. There exists a 1.5-approximation algorithm for Δ-TSP.

CHRISTOFIDES(G, c)

- Find a minimum spanning tree T for (G, c).
- M $\leftarrow$ min cost perfect matching of odd degree nodes in T.
- G' $\leftarrow$ union of spanning tree and matching edges.
- E $\leftarrow$ Eulerian tour in G'.
- H $\leftarrow$ short-cut version of Eulerian tour in E.



E = Eulerian tour in G'



Hamiltonian Cycle H

Justo Puerto,
Dep. Estadística e I.O. Univ. Sevilla

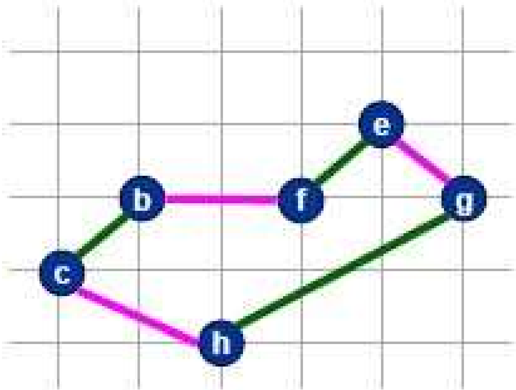
UNIVERSIDAD
DE SEVILLA

TSP: Christofides Algorithm

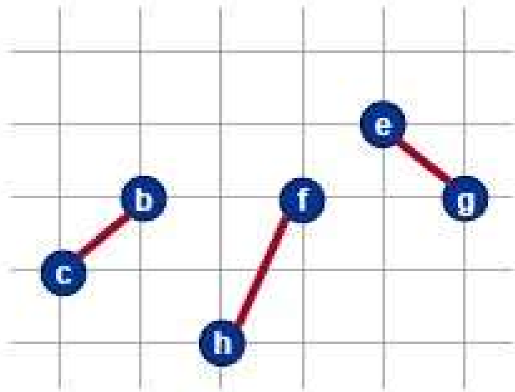
Theorem. There exists a 1.5-approximation algorithm for Δ-TSP.

Proof. Let H^* denote an optimal tour. Need to show $c(H) \leq 1.5\, c(H^*)$.

- $c(T) \leq c(H^*)$ as before.
- $c(M) \leq \frac{1}{2} c(\Gamma^*) \leq \frac{1}{2} c(H^*)$.
  - second inequality follows from Δ-inequality
  - even number of odd degree nodes
  - Hamiltonian cycle on even # nodes comprised of two matchings



Optimal Tour Γ^* on Odd Nodes



Matching M

Justo Puerto,
Dep. Estadística e I.O. Univ. Sevilla

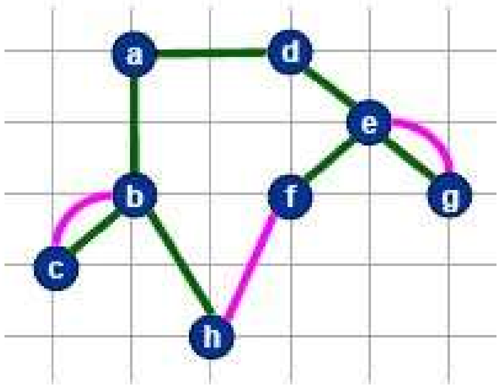
UNIVERSIDAD
DE SEVILLA

TSP: Christofides Algorithm

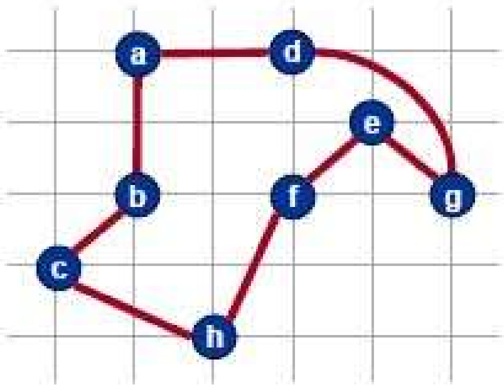
Theorem. There exists a 1.5-approximation algorithm for Δ-TSP.

Proof. Let H^* denote an optimal tour. Need to show $c(H) \leq 1.5\ c(H^*)$.

- $c(T) \leq c(H^*)$ as before.
- $c(M) \leq \frac{1}{2} c(\Gamma^*) \leq \frac{1}{2} c(H^*)$.
- Union of MST and and matching edges is Eulerian.
  - every node has even degree
- Can shortcut to produce H and $c(H) \leq c(M) + c(T)$.



MST + Matching



Hamiltonian Cycle H

Justo Puerto,
Dep. Estadística e I.O. Univ. Sevilla

UNIVERSIDAD
DE SEVILLA