CS472 Module 10 Part D - Dealing with Computational Complexity

Athens State University

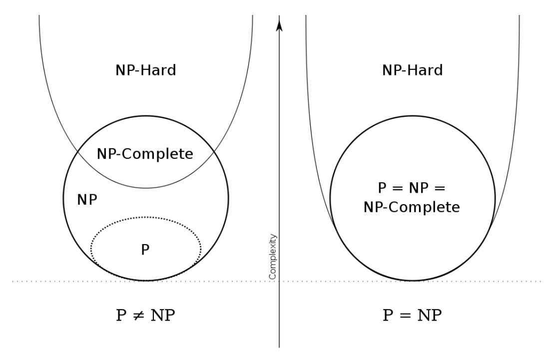
April 4, 2016

Outline

Contents

1 Algorithmic implications of the $P \neq NP$ conjecture

P vs. NP: Context



Most interesting problems are in the *NP*-hard complexity class

Remember what we said about $O(n)$?

n	lg n	n	n lg n	n^2	2^n	n!
10	0.003 us	0.01 us	0.033 us	0.1 us	1 us	3.363 ms
20	0.004 us	0.02 us	0.086 us	0.4 us	1ms	77.1y
30	0.005 us	0.03 us	0.147 us	0.9 us	1s	8.4x10^15y
40	0.005 us	0.04 us	0.213 us	1.6 us	18.3m	
50	0.006 us	0.05 us	0.282 us	2.5 us	13d	
100	0.007 us	0.10 us	0.644 us	10 us	4x10^13y	
1x10^3	0.010 us	1.00 us	9.966 us	1 ms		
1x10^4	0.013 us	10 us	130 us	100 ms		
1x10^5	0.017 us	0.10 ms	1.67 ms	10s		
1x10^6	0.020 us	1 ms	19.93 ms	16.7m		
1x10^7	0.023 us	0.01 s	0.23 s	1.16d		
1x10^8	0.027 us	0.10 s	2.66 s	115.7d		
1x10^9	0.030 us	1 s	29.9 s	31.7y		

Remember what we said about $O(n)$?

Note what the table tells us about working with $O(n)$

- All such algorithms take roughly the same time for $n = 10$
- Any algorithm that is $O(n!)$ becomes useless for $n \geq 20$
- Algorithms that are $O(2^n)$ become impractical for $n > 40$
- Algorithms that are $O(n^2)$ remain usable up to about $n = 10000$ but quickly deteriorate with larger inputs
- Linear ($O(n)$) and log-linear ($O(n * lg(n))$) algorithms remain practical for large input sizes
- Linear is best, but constant is even better.

What can we do?

- OK... but what can we do if most everything interesting is is NP?
  - Use a problem-solving strategy that can find an exact solution to the problem but will not be able to do so in polynomial time
  - Use an approximation algorithm that can find an approximate (sub-optimal) solution in polynomial time

2 Approximation Strategies

Approximation Strategies

- Apply a fast approximation algorithm to get a solution that is not necessarily optimal but hopefully close to it
- Accuracy measures
  - accuracy ratio of an approximate solution s_a
    - $r(s_a) = f(s_a)/f(s^*)$ for minimization problems
    - $r(s_a) = f(s^*)/f(s_a)$ for maximization problems

  - where $f(s_a)$ and $f(s^*)$ are values of the objective function for the approximate solution and actual optimal solution

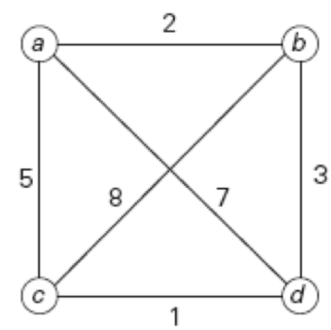
- Performance ratio of the Algorithm A
  - lowest upper bound of $r(s_A)$ on all instances

3 Approximation Algorithms for the TSP

The Traveling Salesman Problem (TSP)

- Suppose we have a weighted connected graph G that we wish traverse s.t. we visit each vertex once and only once
- This is known as a Hamiltonian circuit (or cycle) of the graph
- The Traveling Salesman Problem asks what is the minimum cost Hamiltonian circuit in the graph G

Example: The Traveling Salesman Problem



Tour	Length	
$a \to b \to c \to d \to a$	$l = 2 + 8 + 1 + 7 = 18$	
$a \to b \to d \to c \to a$	$l = 2 + 3 + 1 + 5 = 11$	optimal
$a \to c \to b \to d \to a$	$l = 5 + 8 + 3 + 7 = 23$	
$a \to c \to d \to b \to a$	$l = 5 + 1 + 3 + 2 = 11$	optimal
$a \to d \to b \to c \to a$	$l = 7 + 3 + 8 + 5 = 23$	
$a \to d \to c \to b \to a$	$l = 7 + 1 + 8 + 2 = 18$	

FIGURE 3.7 Solution to a small instance of the traveling salesman problem by exhaustive search.

Nearest Neighbor: the greedy approach to the TSP

- Pick a starting vertex v

- Each vertex keeps some state of whether or not that vertex has been visited in the tour
- Look at the neighbors of v and select the neighbor with the lowest cost that has not been visited

Nearest Neighbor: Just how bad is the greedy approach

- Nearest-neigbor tours may depend on the starting city
- Note that $r_A = \infty$, i.e. unbounded above
  - Make the largest weight arbitrarily large

Multifragment-Heuristic - Shortest edge: another greedy approach to the TSP

- Inspired by Kruskal's algorithm
- Sort the edges in increasing order of weight
- Start with the least cost edge, look at edges 1-by-1 and select an edge only if the edge, together with already selected edges,
  - does not cause a vertex have degree of three or more (# edges)
  - does not form a cycle, unless the number of selected edges equals the number of vertices in the graph

Other approaches: Twice around the tree

- Algorithm
  - Construct a MST of the graph
  - Start at any vertex, create a path that goes around twice around the tree and returns to the same vertex
  - Create a tour from the circuit by making shortcuts to avoid visiting intermediate vertices more than once
- Still has $r_A = \infty$ for general instances, but get better tours than with nearest-neighbor

Other appraoches: Twice around the tree

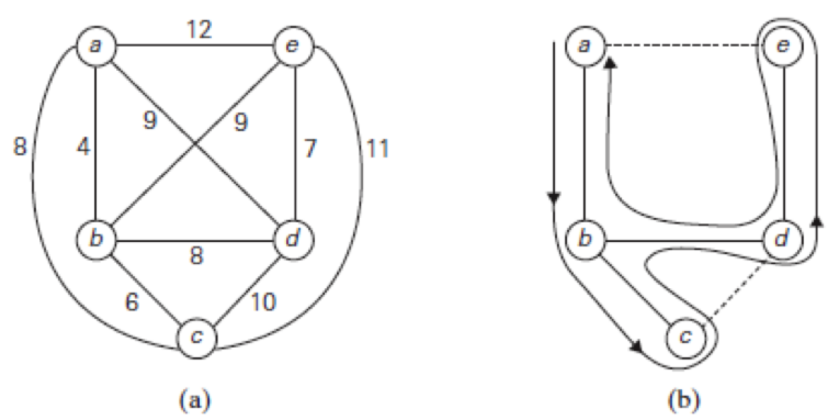


FIGURE 12.11 Illustration of the twice-around-the-tree algorithm. (a) Graph. (b) Walk around the minimum spanning tree with the shortcuts.

Other approaches: Chistofiedes Algorithm

- Algorithm
  - Construct MST of the graph
  - Add edges of a min-weight matching of all the odd vertices in the MST
  - Find an Eulerian circuit of the multigraph from step 2
  - Create a tour from the path from the previous state by making shortcuts to avoid visiting nodes more than once
- Still ha $r_A = \infty$ but gets better results than twice-around-the-tree

Other approaches: Chistofiedes Algorithm

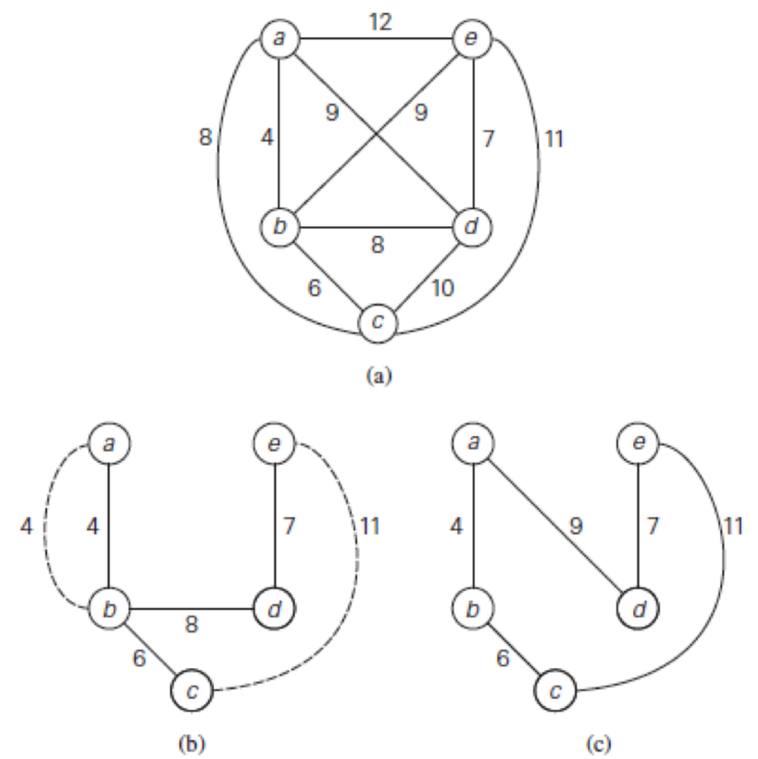


FIGURE 12.12 Application of the Christofides algorithm. (a) Graph. (b) Minimum spanning tree with added edges (in dash) of a minimum-weight matching of all odd-degree vertices. (c) Hamiltonian circuit obtained.

Other approaches: Euclidean instances

- It has been proven that if $P \neq NP$, then there exists no approximation algorithm for the TSP with a finite performance ratio.
- *Euclidean circuit*: an instance of the TSP such that the distances (weights) on the graph satisfy the conditions:
  - *symmetry*: $d[i,j] = d[j,i]$ for any pair of nodes i and j,
  - *triangle inequality*: $d[i,j] \leq d[i,k] + d[k,j]$ for any nodes i, j, and k.

Other approaches: Euclidean instances

- In different cases, the approx. tour length / optimal tour length is less than:
  - Euclidean instances: $0.5 * (\lceil (log_2 n) \rceil + 1)$
  - Nearest neighbor and shortest edge: 2
  - Twice-around-the-tree: 1.5

Other approaches: Local search Heuristics

- Start with some initial tour (e.g., nearest neighbor)
- On each iteration, explore the current tour's neighborhood by exchanging a few edges in the tour
- If a new tour is shorter, make it the current tour; otherwise, try exchanging other edges
- If no change yields a shorter tour, current tour is returned as the output

Other approaches: Local Search Heuristics

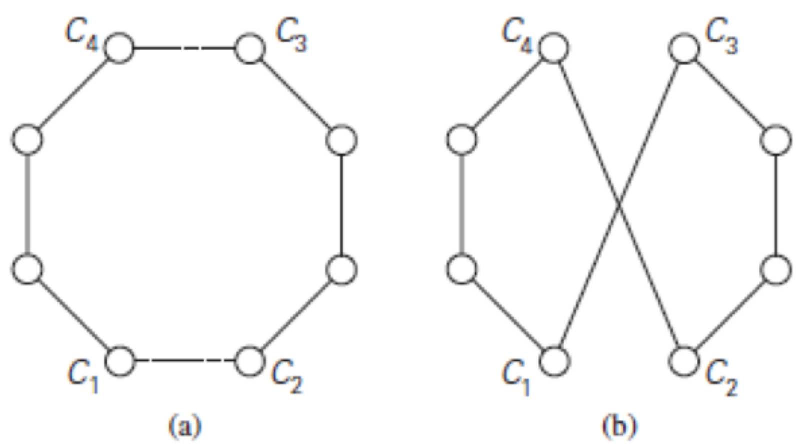


FIGURE 12.13 2-change: (a) Original tour. (b) New tour.

Other approaches: Local Search Heuristics

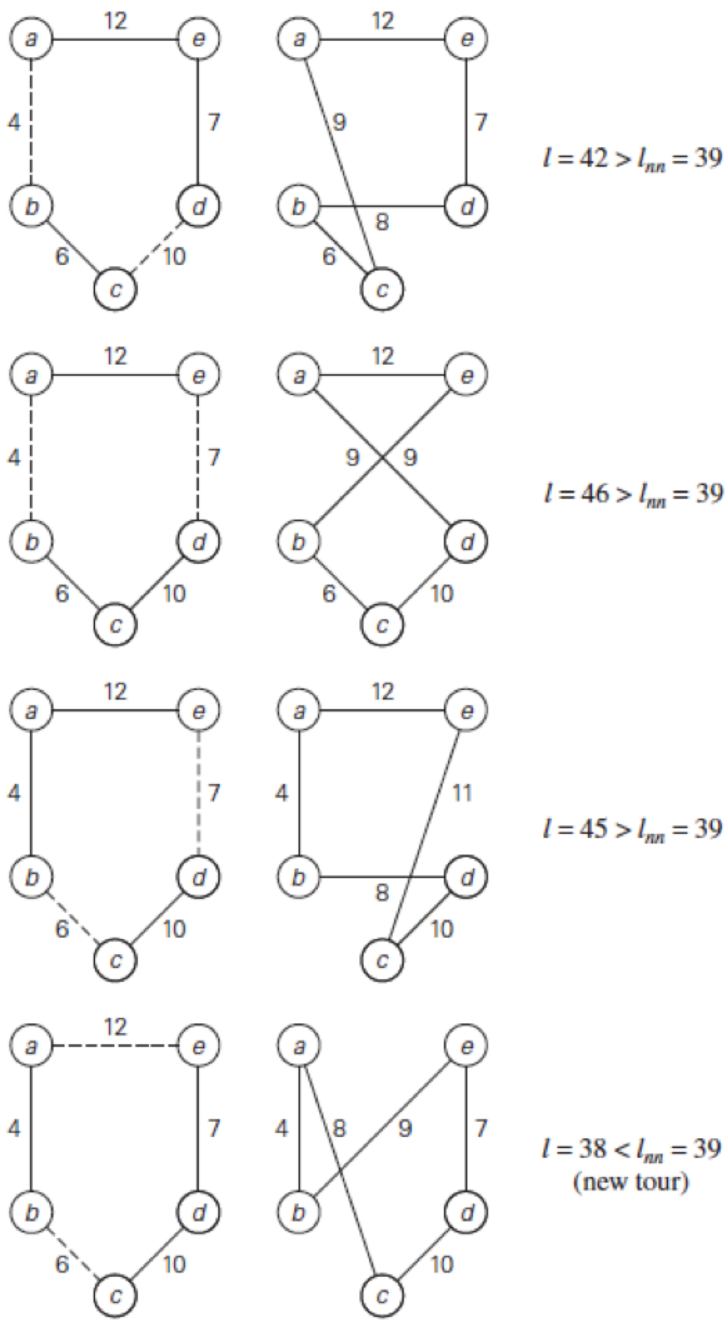


FIGURE 12.14 2-changes from the nearest-neighbor tour of the graph in Figure 12.11.

Other approaches: Local Search Heuristics

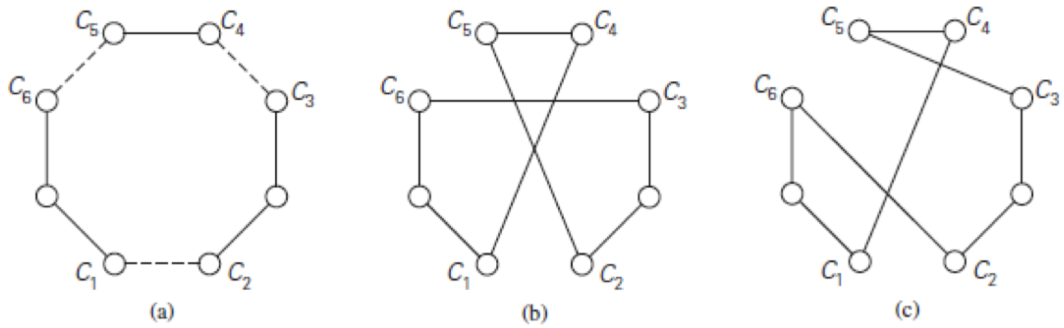


FIGURE 12.15 3-change: (a) Original tour. (b), (c) New tours.

4 Approximation Algorithms for the Knapsack Problem

Recall the greedy algorithm for the knapsack problem

- Problem
  - We have values with weights, want best value for a container with a specific capacity
- Algorithm
  - Order the items in decreasing order of relative values $v_1/w_1 \geq \ldots \geq v_n/w_n$
  - Select items in this order, skipping those that don't fit into the knapsack
- Approximation performance ration r_A is unbounded but algorithm yields exact solutions for the continuous version of the problem

Approximation Algorithm

- Algorithm
  - Order items in decreasing order of relative values
  - From some integer parameter $0 \leq k \leq n$, generate all subsets of k items or less and for each of those that fit in the container, add the remaining items in decreasing order of their value to weight ratios
  - Find the most valuable subset among those generated and output this as our result
- Accuracy: $f(s^*)/f(s_a) \leq 1 + 1/k$ for an instance of size n
- Time efficiency: $O(kn^{k+1})$
  - There are *fully polynomial schemes*: algorithms with polynomial running time as functions of both n and k

5 Key Points

Key Points

- Many *NP*-hard problems can be dealt with using approximation algorithms
- Need to have some means to measure the quality of the approximation
- Examples
  - Approximation algorithms for the TSP
  - Approximation algorithms for the Knapsack problem