Georgia Tech

CSE 6140/ CX 4140
Computational Science and Engineering
ALGORITHMS

Coping with NP-completeness - 2

Instructor: Xiuwei Zhang

Assistant Professor

School of Computational Science and Engineering

Partially based on slides by Prof. Ümit V. Çatalyürek

Course logistics

- Test 2
  10/28 Wednesday, 9am - 11:59pm US Eastern Time
  Duration 3 hours

- Project team submission
  Due Friday 10/30 11:59pm

- Project competition (optional, for a small number of bonus points)
  Two independent algorithm tracks:
  (1) branch-and-bound
  (2) local search
  Do not have separate tracks for different programming languages

Dealing with NP-complete problems

Branch & bound	Sacrifice running time: create an algorithm with running time exponential in the input size (but which might do well on the inputs you use)
Approximation	Sacrifice quality of the solution: quickly find a solution that is *provably not very bad*
Local search	Quickly find a solution for which you cannot give any quality guarantee (but which might often be good in practice on real problem instances)
Restriction	By restricting the structure of the input (e.g., to planar graphs, 2SAT), faster algorithms are usually possible.
Randomization	Use randomness to get a faster average running time, and allow the algorithm to fail to find optimum with some small probability.
Parameterized algorithms	Sacrifice running time: allow the running time to have an exponential factor, but ensure that the exponential dependence is not on the entire input size but just on some parameter that is hopefully small

Exact Solution Strategies

- ***exhaustive search*** (*brute force*)
  - useful only for small instances

- ***backtracking***
  - Construct the alternatives component by component
  - eliminates some unnecessary cases from consideration
  - yields solutions in better time for many instances but worst case is still exponential

- ***branch-and-bound***
  - further refines the backtracking idea (eliminating unnecessary cases) for optimization problems

Branch-and-Bound

- An enhancement of backtracking
- Applicable to optimization problems (assume we are minimizing)
- Keep track of BEST solution found so far (upper bound on optimal)
- For each node (partial solution), computes a lower bound LB on the value of the objective function for all descendants of the node (extensions of the partial solution)
  - any extension of this partial solution will have quality at least LB
- Uses the bound for:
  - ruling out certain nodes as "nonpromising" to prune the tree – if a node's bound is not better than the best solution seen so far
  - guiding the search through state-space as a measure of "promise"

Branch-and-Bound algorithm

Branch-and-Bound(P) // Input: minimization problem P
01 F <- {(∅,P)} // Frontier set of configurations
02 B <- (**+∞ , (∅**,P)) // Best cost and solution
03 **while** F not empty **do**
04 *Choose* (X,Y) in F – the most ”promising” configuration
05 *Expand* (X,Y), by making a choice(es)
06 Let (X_1, Y_1), (X_2, Y_2), ..., (X_k, Y_k) be new configurations
07 **for** each new configuration (X_i,Y_i) **do**
08 ”*Check*” (X_i,Y_i)
09 **if** ”solution found” **then**
10 **if** cost(X_i) < B cost **then** // update upper bound
11 B <- (cost(X_i),(X_i,Y_i))
12 **if** not ”dead end” **then**
13 **if** *lb*(X_i) < B cost **then** //
14 F <- F U {(X_i,Y_i)} // else prune by lb
15 **return** B

Traveling Salesman Problem (TSP)

- TSP. Given a set of n cities and a pairwise distance function d(u, v), find a tour with minimum total cost.

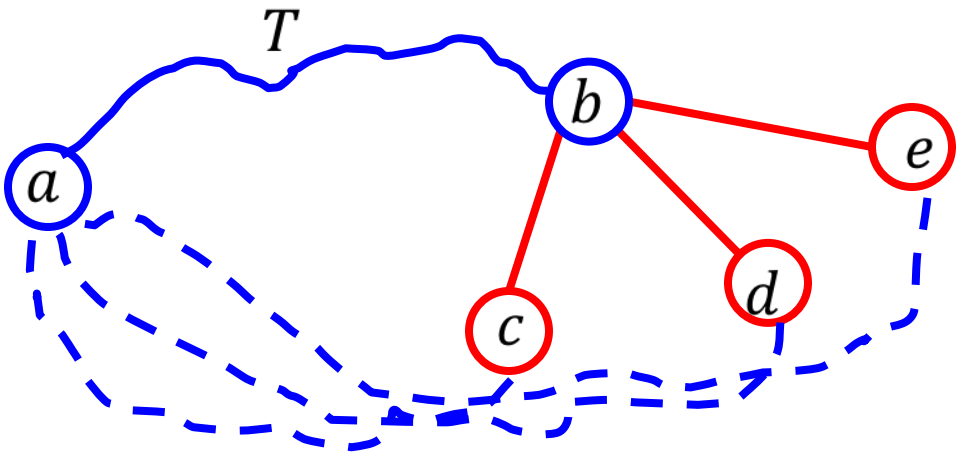
- Partial solution (a,T,b): a path from a start node a to b, going through nodes T (same as HamCycle)

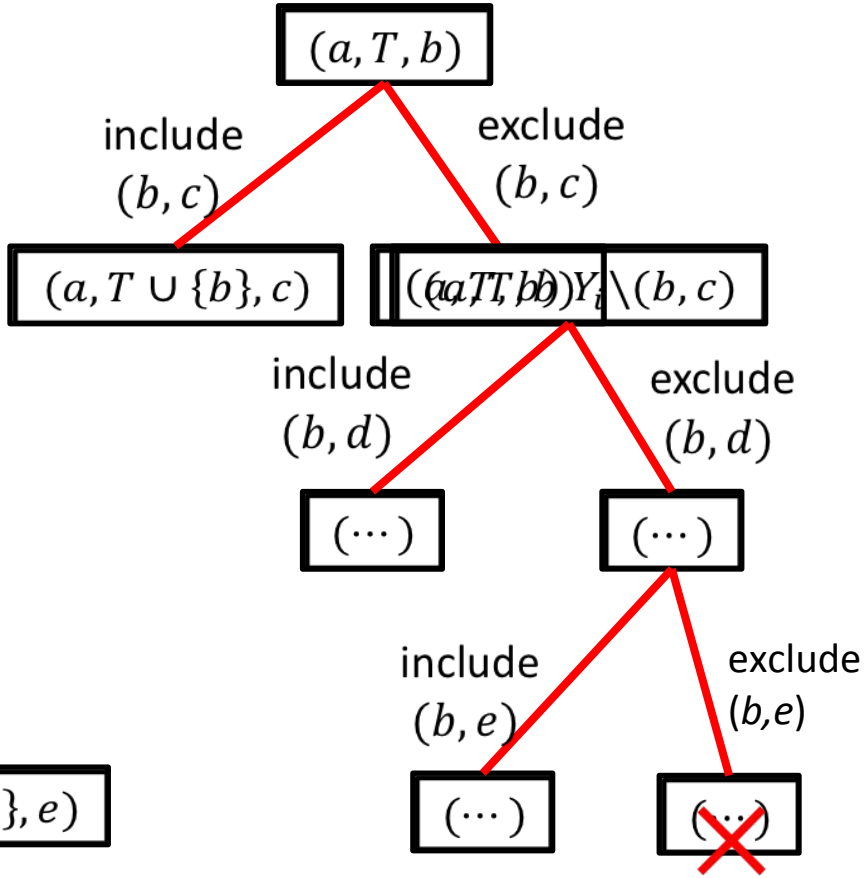
- *Expand*: choose an edge from b to V-T-{a,b} (same as HamCycle)

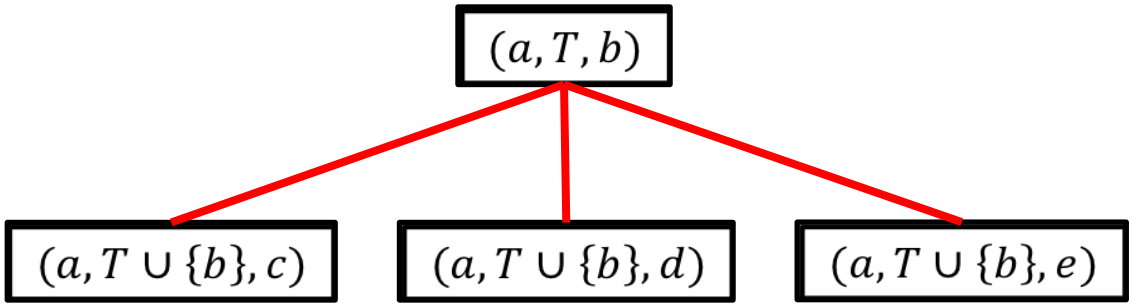
- *Choose*: what can be the best-first criteria?
  - The partial assignment with smallest lower bound (most promising of having a short TSP tour)

- *Check*:
  - Same feasibility checks as HamCycle to detect dead-end
  - How do we compute a Lower Bound given a partial solution (a,T,b)?

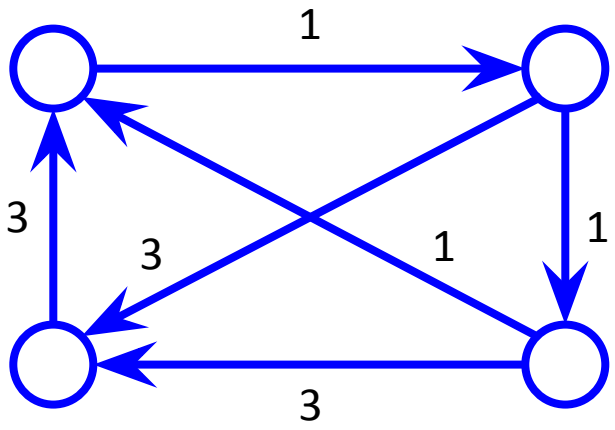
Search Tree


T
a
b
c
d
e


(a, T, b)
include (b, c)
exclude (b, c)
$(a, T \cup \{b\}, c)$
include (b, d)
exclude (b, d)
$(\cdots)$
$(\cdots)$
include (b, e)
exclude (b, e)
$(\cdots)$
$(\cdots)$


(a, T, b)
$(a, T \cup \{b\}, c)$
$(a, T \cup \{b\}, d)$
$(a, T \cup \{b\}, e)$

TSP — Bounding Function 1



- Because a tour must leave every vertex exactly once, a lower bound on the length of a tour is **the sum of the minimum cost of leaving every vertex.**
  - 1+1+1+3 = 6 ?

- Note: This is not to say that there is a tour with this length. Rather, it says that there can be no shorter tour.

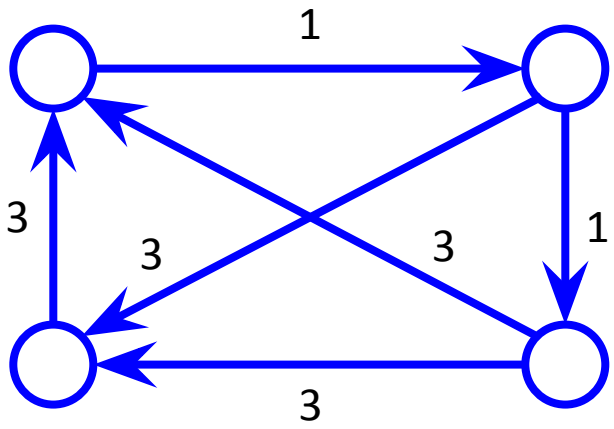
TSP — Bounding Function 1

- Because a tour must leave every vertex exactly once, a lower bound on the length of a tour is **the sum of the minimum cost of leaving every vertex.**

- Note: This is not to say that there is a tour with this length. Rather, it says that there can be no shorter tour.

- Given a partial solution (a,T,b)
  - The lower bound = (length of path from a to b) + (the sum of min cost of leaving each vertex in V-T-a)
  - We start with partial solution (v_1, $\varnothing$, v_1)

TSP — Bounding Function 2



- Every vertex must be entered and exited exactly once
- For a given edge (u, v), think of half of its weight as the exiting cost of u, and half of its weight as the entering cost of v
  - 2+1+2+3 = 8

TSP — Bounding Function 2

- Every vertex must be entered and exited exactly once
- For a given edge (u, v), think of half of its weight as the exiting cost of u, and half of its weight as the entering cost of v
- The total length of a tour = the sum of costs of visiting (entering and exiting) every vertex exactly once.
- a lower bound on the length of a tour is the sum of the lower bound on the cost of **entering and leaving** every vertex**.**
  - Simple: for each vertex, lower bound is the sum of the two shortest adjacent edges divided by 2 (incoming and outgoing)

TSP bound: reduced cost matrix

Step 1 to reduce: Search each row for the smallest value

- The Cost Matrix for a Traveling Salesperson Problem.

from i \ to j	1	2	3	4	5	6	7
1	∞	3	93	13	33	9	57
2	4	∞	77	42	21	16	34
3	45	17	∞	36	16	28	25
4	39	90	80	∞	56	7	91
5	28	46	88	33	∞	25	57
6	3	88	18	46	92	∞	7
7	44	26	33	27	84	39	∞

TSP bound: reduced cost matrix

Step 2 to reduce: Search each column for the smallest value

- Reduced cost matrix:

i \ j	1	2	3	4	5	6	7	
1	∞	0	90	10	30	6	54	(-3)
2	0	∞	73	38	17	12	30	(-4)
3	29	1	∞	20	0	12	9	(-16)
4	32	83	73	∞	49	0	84	(-7)
5	3	21	63	8	∞	0	32	(-25)
6	0	85	15	43	89	∞	4	(-3)
7	18	0	7	1	58	13	∞	(-26)
								reduced:84

TSP bound: reduced cost matrix

i \ j	1	2	3	4	5	6	7
1	∞	0	83	9	30	6	50
2	0	∞	66	37	17	12	26
3	29	1	∞	19	0	12	5
4	32	83	66	∞	49	0	80
5	3	21	56	7	∞	0	28
6	0	85	8	42	89	∞	0
7	18	0	0	0	58	13	∞
			(-7)	(-1)			(-4)

The total cost of 84+12=96 is subtracted. Thus, we know the lower bound of feasible solutions to this TSP problem is 96.

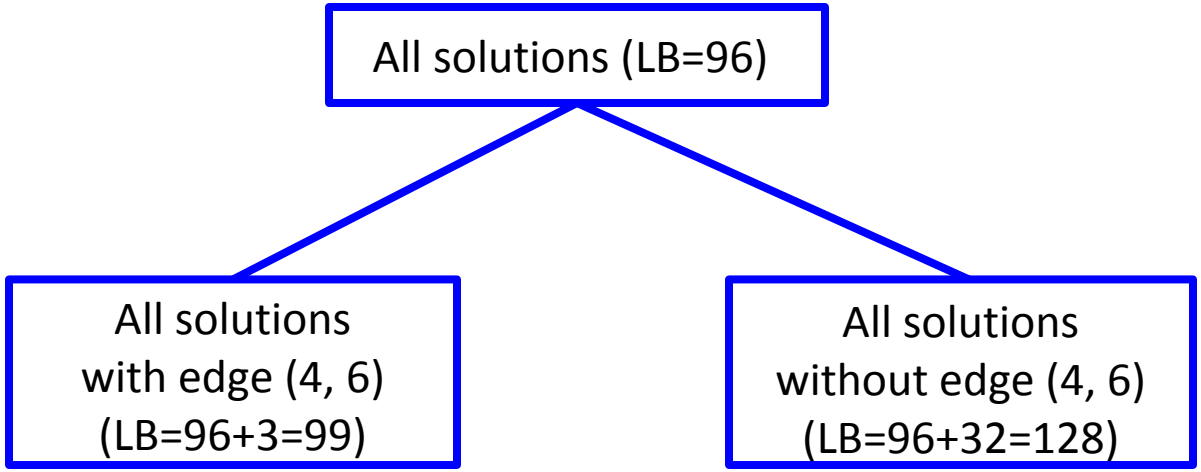
TSP

- How do we branch, i.e., expand the partial solution?

- Any next vertex not chosen yet (multi-way)
- Any next vertex with an edge from the last vertex (multi-way)
- Any next edge connected to last vertex (binary)

TSP bound: reduced cost matrix

- Total cost reduced: 84+7+1+4 = 96 (LB) decision tree:



The Highest Level of a Decision Tree.

- If we use edge (3, 5) to split, the difference on the lower bounds is 17+1 = 18.

For the left subtree (Edge (4, 6) is included)

i \ j	1	2	3	4	5	7
1	∞	0	83	9	30	50
2	0	∞	66	37	17	26
3	29	1	∞	19	0	5
5	3	21	56	7	∞	28
6	0	85	8	∞	89	0
7	18	0	0	0	58	∞

A Reduced Cost Matrix if edge (4, 6) is included.

1. 4^{th} row is deleted.
2. 6^{th} column is deleted.
3. We must set cost(6,4) to be ∞.

For the left subtree (Edge (4, 6) is included)

- The cost matrix for all solution with edge (4,6):

i \ j	1	2	3	4	5	7	
1	∞	0	83	9	30	50	
2	0	∞	66	37	17	26	
3	29	1	∞	19	0	5	
5	0	18	53	4	∞	25	(-3)
6	0	85	8	∞	89	0	
7	18	0	0	0	58	∞	

- Total cost reduced: 96+3 = 99 (new LB)

For the right subtree (Edge (4,6) is excluded)

We only have to set cost(4,6) to be ∞.

i \ j	1	2	3	4	5	6	7
1	∞	0	83	9	30	6	50
2	0	∞	66	37	17	12	26
3	29	1	∞	19	0	12	5
4	32	83	66	∞	49	∞	80
5	3	21	56	7	∞	0	28
6	0	85	8	42	89	∞	0
7	18	0	0	0	58	13	∞

Total cost reduced: 96+32 = 128 (new LB)

TSP bounds

- Smarter ideas?

- What if we had a symmetric TSP (the cost of an edge is the same in both directions, e.g., Euclidean distance, then we can treat the graph as undirected)?

- TSP variants:
- Symmetric: distance from u to v = distance from v to u
- Metric: dist(u,v) + dist(v,w) >= dist(u,w)
- Euclidean (cities are represented as (x,y) coordinates and distances are Euclidean in the plane)

TSP (symmetric)—Bounding Function 3 **Dynamic**

- Given a partial solution (a,T,b)
- We have a path from a to b using vertices $T \subseteq V - \{a, b\}$

- A lower bound is the sum of:
  - The partial path we have
  - A lower bound on exiting a and b (their shortest edge **to a vertex in V-T-{a,b}**)
  - A lower bound on visiting the remaining nodes (The cost of the Minimum Spanning Tree for the subgraph over nodes in **V-T-{a,b}**)