6.231 Dynamic Programming and Stochastic Control
Fall 2008

6.231 DYNAMIC PROGRAMMING

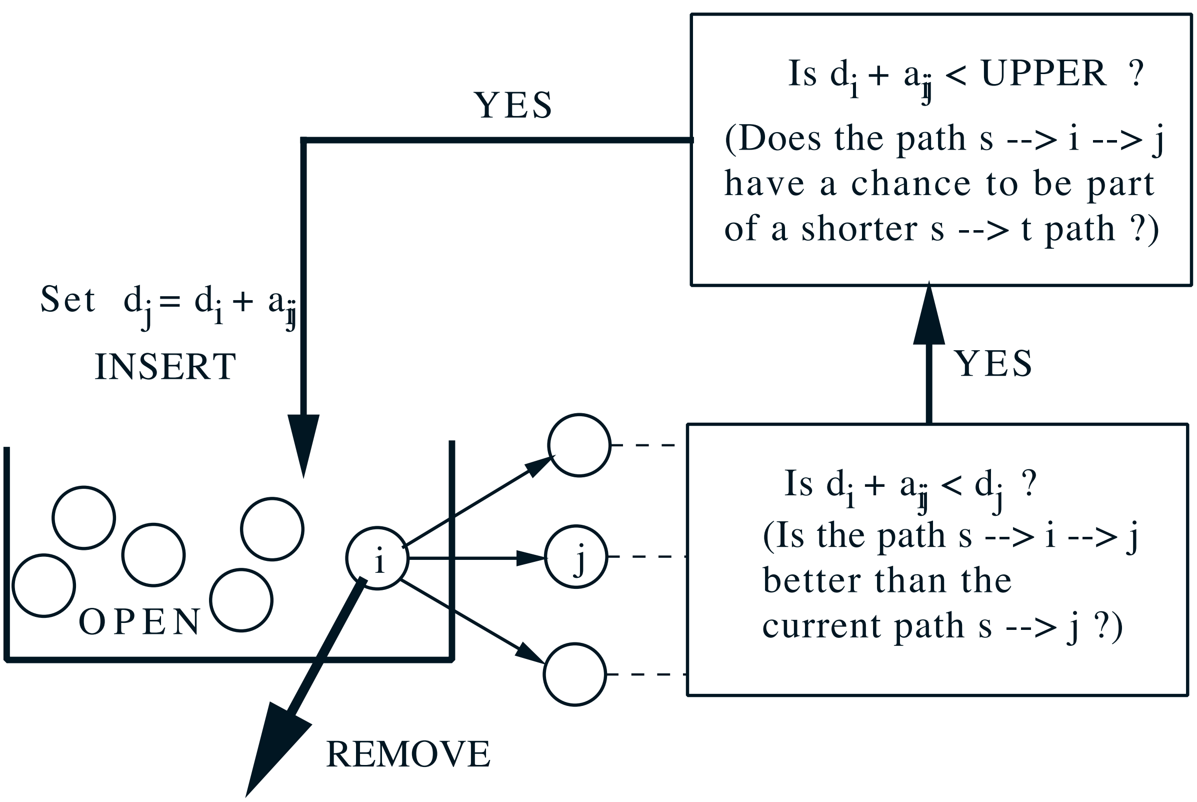
LECTURE 4

LECTURE OUTLINE

- Label correcting methods for shortest paths
- Variants of label correcting methods
- Branch-and-bound as a shortest path algorithm

LABEL CORRECTING METHODS

- Origin s, destination t, lengths a_{ij} that are ≥ 0
- d_i (label of i): Length of the shortest path found thus far (initially $d_i = \infty$ except $d_s = 0$). The label d_i is implicitly associated with an $s \to i$ path
- UPPER: Label d_t of the destination
- OPEN list: Contains "active" nodes (initially OPEN=$\{s\}$)



VALIDITY OF LABEL CORRECTING METHODS

Proposition: If there exists at least one path from the origin to the destination, the label correcting algorithm terminates with UPPER equal to the shortest distance from the origin to the destination

Proof: (1) Each time a node j enters OPEN, its label is decreased and becomes equal to the length of some path from s to j

(2) The number of possible distinct path lengths is finite, so the number of times a node can enter OPEN is finite, and the algorithm terminates

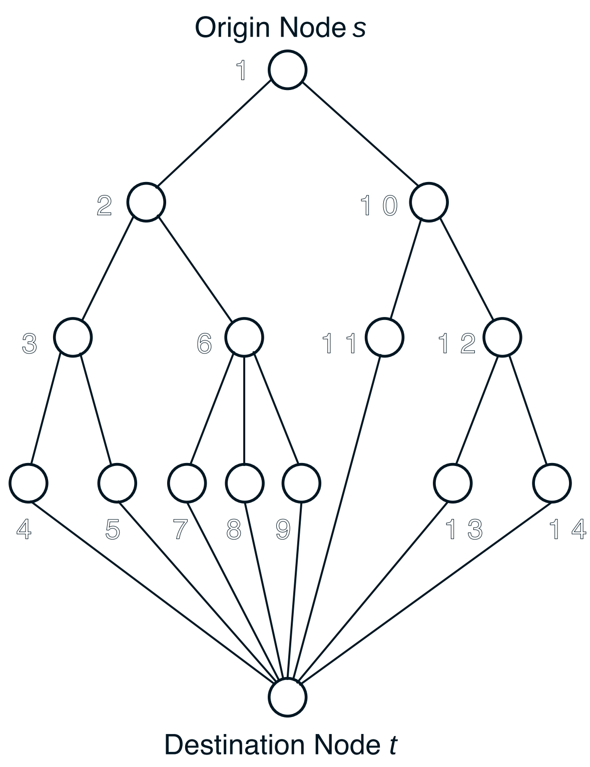
(3) Let $(s, j_1, j_2, \ldots, j_k, t)$ be a shortest path and let d^* be the shortest distance. If UPPER $> d^*$ at termination, UPPER will also be larger than the length of all the paths $(s, j_1, \ldots, j_m)$, $m = 1, \ldots, k$, throughout the algorithm. Hence, node j_k will never enter the OPEN list with d_{j_k} equal to the shortest distance from s to j_k. Similarly node j_{k-1} will never enter the OPEN list with $d_{j_{k-1}}$ equal to the shortest distance from s to j_{k-1}. Continue to j_1 to get a contradiction

MAKING THE METHOD EFFICIENT

- Reduce the value of UPPER as quickly as possible
  - Try to discover "good" $s \to t$ paths early in the course of the algorithm
- Keep the number of reentries into OPEN low
  - Try to remove from OPEN nodes with small label first.
  - Heuristic rationale: if d_i is small, then d_j when set to $d_i + a_{ij}$ will be accordingly small, so reentrance of j in the OPEN list is less likely
- Reduce the overhead for selecting the node to be removed from OPEN
- These objectives are often in conflict. They give rise to a large variety of distinct implementations
- Good practical strategies try to strike a compromise between low overhead and small label node selection

NODE SELECTION METHODS

- **Depth-first search:** Remove from the top of OPEN and insert at the top of OPEN.
  - Has low memory storage properties (OPEN is not too long). Reduces UPPER quickly.



- **Best-first search (Djikstra):** Remove from OPEN a node with minimum value of label.
  - Interesting property: Each node will be inserted in OPEN at most once.
  - Nodes enter OPEN at minimum distance
  - Many implementations/approximations

ADVANCED INITIALIZATION

- Instead of starting from $d_i = \infty$ for all $i \neq s$, start with

$$d_i = \text{length of some path from } s \text{ to } i \quad (\text{or } d_i = \infty)$$

$$\text{OPEN} = \{i \neq t \mid d_i < \infty\}$$

- Motivation: Get a small starting value of UPPER.
- No node with shortest distance $\geq$ initial value of UPPER will enter OPEN
- Good practical idea:
  - Run a heuristic (or use common sense) to get a "good" starting path P from s to t
  - Use as UPPER the length of P, and as d_i the path distances of all nodes i along P
- Very useful also in reoptimization, where we solve the same problem with slightly different data

VARIANTS OF LABEL CORRECTING METHODS

- If a **lower bound** h_j of the true shortest distance from j to t is known, use the test

$$d_i + a_{ij} + h_j < \text{UPPER}$$

for entry into OPEN, instead of

$$d_i + a_{ij} < \text{UPPER}$$

The label correcting method with lower bounds as above is often referred to as the **A^* method**.

- If an **upper bound** m_j of the true shortest distance from j to t is known, then if $d_j + m_j <$ UPPER, reduce UPPER to $d_j + m_j$.

- **Important use:** Branch-and-bound algorithm for discrete optimization can be viewed as an implementation of this last variant.

BRANCH-AND-BOUND METHOD

- **Problem:** Minimize $f(x)$ over a *finite* set of feasible solutions X.

- Idea of branch-and-bound: Partition the feasible set into smaller subsets, and then calculate certain bounds on the attainable cost within some of the subsets to eliminate from further consideration other subsets.

Bounding Principle

Given two subsets $Y_1 \subset X$ and $Y_2 \subset X$, suppose that we have bounds

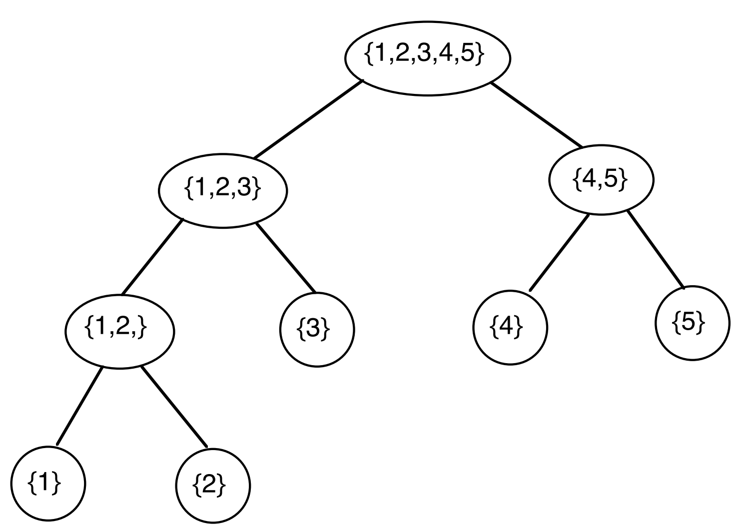
$$\underline{f}_1 \leq \min_{x \in Y_1} f(x), \qquad \overline{f}_2 \geq \min_{x \in Y_2} f(x).$$

Then, if $\overline{f}_2 \leq \underline{f}_1$, the solutions in Y_1 may be disregarded since their cost cannot be smaller than the cost of the best solution in Y_2.

- The B+B algorithm can be viewed as a label correcting algorithm, where lower bounds define the arc costs, and upper bounds are used to strengthen the test for admission to OPEN.

SHORTEST PATH IMPLEMENTATION

- Acyclic graph/partition of X into subsets (typically a tree). The leafs consist of single solutions.
- Upper/Lower bounds $\underline{f}_Y$ and $\overline{f}_Y$ for the minimum cost over each subset Y can be calculated.
- The lower bound of a leaf $\{x\}$ is $f(x)$
- Each arc (Y, Z) has length $\underline{f}_Z - \underline{f}_Y$
- Shortest distance from X to Y = $\underline{f}_Y - \underline{f}_X$
- Distance from origin X to a leaf $\{x\}$ is $f(x) - \underline{f}_X$
- Shortest path from X to the set of leafs gives the optimal cost and optimal solution
- UPPER is the smallest $f(x) - \underline{f}_X$ out of leaf nodes $\{x\}$ examined so far



BRANCH-AND-BOUND ALGORITHM

Step 1: Remove a node Y from OPEN. For each child Y_j of Y, do the following:

- Entry Test: If $\underline{f}_{Y_j} <$ UPPER, place Y_j in OPEN.
- Update UPPER: If $\overline{f}_{Y_j} <$ UPPER, set UPPER $= \overline{f}_{Y_j}$, and if Y_j consists of a single solution, mark that as being the best solution found so far

Step 2: (Termination Test) If OPEN: empty, terminate; the best solution found so far is optimal. Else go to Step 1

- It is neither practical nor necessary to generate a priori the acyclic graph (generate it as you go)
- Keys to branch-and-bound:
  - Generate as sharp as possible upper and lower bounds at each node
  - Have a good partitioning and node selection strategy
- Method involves a lot of art, may be prohibitively time-consuming ... but guaranteed to find an optimal solution