IN7K3: Flujo en Redes

2: Shortest Path Problem

Lecturer: Fernando Ordóñez

1 Shortest Path Problem

Single/Multiple source(s) to single/multiple destination(s).
Non-negative cost, no negative cost cycles, negative cost cycles.

1.1 Assumptions

- Integral, non-negative data
- There is a directed path from the source node s to all other nodes.
- Applications
  - Vehicle Routing
  - Communication Systems

1.2 Examples

Approximating a piece-wise linear function.
INPUT: n points $(x_1, y_1), \ldots, (x_n, y_n)$, such that $x_1 \leq x_2 \leq \ldots \leq x_n$. c^* is the cost per point included. c_{ij} is the cost of approximating the line on points $i, i+1, \ldots, j-1$ by a straight line from i to j.

OUTPUT: The cheapest piece-wise linear approximation.

Example Knapsack Problem.

$$\begin{array}{ll} \max & \sum_{i=1}^{n} b_i x_i \\ \text{s.t.} & \sum_{i=1}^{n} w_i x_i \leq W \\ & x_i \in \{0, 1\} \end{array}$$

2 Dijksta

2.1 Central Idea in Shortest Path Algorithms

- Let $d(i)$ denote the temporary shortest distance label from the origin, node 1, to i.
- There is some path from 1 to i of distance $d(i)$.
- Procedure **Update**(i):
  For each $j \in A(i)$ do
  if $d(j) > d(i) + c_{ij}$ then $d(j) = d(i) + c_{ij}$ and $pred(j) = i$.

- Let $d^*(i)$ denote the shortest distance from 1 to i. If Update(i) is run when $d(i) = d^*(i)$ then we do not need to run Update(i) again.

2.2 Dijkstra's Algorithm

S: The set of permanently labeled nodes. This means $d(j) = d^*(j)$ for $j \in S$.

T: The set of temporarily labeled nodes. This means $d(j) \geq d^*(j)$ for $j \in T$.

Set $S = \{1\}$, $T = N \setminus \{1\}$
Set $d(1) = 0$ and pred$(1) = 0$
Set $d(j) = \infty$ for $j \in T$
Update(1)
while $S \neq N$ do
  select $i \in T$ such that $d(i) = \min\{d(j) \mid j \in T\}$
  $S = S \cup \{i\}$, and $T = T \setminus \{i\}$
  Update(i)
end

3 Example

3.1 Discussion and Complexity

What things are true at each iteration? (Algorithm invariance)

What things change at each iteration?

To prove that algorithm invariants remain true: Use induction

Step 1 Check that invariants are true after initialization

Step 2 Use induction. Assume that they are true at some step, prove that they remain true at the next iteration

(**iteration:** find min node in T followed by Update.)

Complexity of Update(i):

Complexity of "find min node in T":

Total Complexity: Can we improve this? When G is not dense!

4 Dijkstra: A Bucket Scheme

Let $C = 1 + \max\{c_{ij} \mid (ij) \in A\}$. Then what is an upperbound on the shortest path from 1 to any node i?

Problem: Find a quick way to select the min distance node in T.

Solution: Create buckets from 0 to nC, such that $\text{BUCKET}(k) = \{j \in T \mid d(j) = k\}$.

Does this help? **Example:**

Complexity of Update(i):

Complexity of "find min node in T":

Total Complexity:

Bottleneck operation:

5 Radix Heaps

Consider variable and adjustable width buckets.

Begin defining
BUCKET(0) $= \{j \in T \mid d(j) = 0\}$ and
for $k = 1, \ldots, K$: BUCKET(k) $= \{j \in T \mid 2^{k-1} \leq d(j) \leq 2^k - 1\}$.

We modify the "find min in T" step by incorporating the following:

Let B be the minimum non-empty bucket
If B is either bucket 1 or 2
    Update(i)
    For each j that $d(j)$ changes, Reinsert(j)
else
    Redistribute the range of B on the first $B - 1$ buckets
    For each j in B, Reinsert(j).

Example

5.1 Complexity of Radix Heaps

What steps to count?

- "scan left" operations. The operations to move a node to its new bucket.
- arc scan operations. When updating node i.

Why can we avoid counting:

- modify $d(j)$?
- scan current bucket to see if node j should stay in bucket?
- delete j from current bucket?
- insert j into its correct bucket?

Let K = number of buckets.

- $O(m)$ scanning arcs over all algorithm
- $O(K)$ for finding minimum non-empty bucket for each find min. $O(nK)$ over all.
- $O(nK)$ for redistributing the ranges
- $O(nK)$ for reinserting nodes into buckets.

- $O(m + nK)$ in total
- We can use $K = \lceil \log nC \rceil$.

6 Label Correcting Algorithms

The label-setting algorithms presented so far for the shortest path problem assume that there are no negative cost cycles. We now remove this assumption and develop algorithms that will either detect the existence of a negative cost cycle, or if no such cycle exists solve the shortest path problem.

Example Run Dijkstra's algorithm on the following problem.

6.1 Optimality Conditions

If we are given a set of node labels that satisfy:

$$d(1) = 0$$
$$d(j) \leq d(i) + c_{ij} \text{ for all } (i,j) \in A$$
$$d(i) \text{ is the length of a path from 1 to } i$$

Then these node labels guarantee:

7 Generic Label-Correcting Algorithm

Label Correcting
Set $d(1) = 0$, pred$(1) = 0$
Set $d(j) = \infty$ for each $j \in N \setminus \{1\}$
while exists $(i, j) \in A$ such that $d(j) > d(i) + c_{ij}$ do
    set $d(j) = d(i) + c_{ij}$, pred$(j) = i$
endwhile

7.1 Example

7.2 Complexity

Theorem 1. *If all data is integral, and there are no negative cost cycles in the network, then the label correcting algorithm ends after a finite number of iterations with the optimal solution.*

Why is the algorithm finite?

Why does the algorithm return the optimal solution?

What if data is rational or there are negative cost cycles?

Worst case per iterations?

Bound on number of iterations?

The following simple implementation, which organizes the search for violating arcs by keeping track of nodes on a LIST (similar to Dijkstra), takes $O(mn)$ time to implement the label correcting algorithm.

Modified Label Correcting

Set $d(1) = 0$, pred$(1) = 0$
Set $d(j) = \infty$ for each $j \in N \setminus \{1\}$
Set LIST $= \{1\}$
while LIST $\neq \emptyset$ do
    delete i from LIST
    Update(i)
    for each j such that $d(j)$ decreases, add to LIST
endwhile

7.3 Example

8 Detecting Negative Cost Cycles

When should we stop the label correcting algorithm?

- Stop if $d(j)$ is sufficiently small for some j, e.g.

- Stop if FIFO Modified Label Correcting (add at the end of LIST, delete from the beginning) is creating a path from 1 to i with more than $n - 1$ arcs.
- Stop if the intree formed by the predecessor subgraph has a cycle. If this tree has a cycle, then it is a negative cost cycle. (most efficient)