Algorithms and Data Structures 2
CS 1501

Fall 2022

Sherif Khattab

ksm73@pitt.edu

(Slides are adapted from Dr. Ramirez's and Dr. Farnan's CS1501 slides.)

Announcements

- Upcoming Deadlines
  - Homework 11: Friday 12/9 @ 11:59 pm
  - Lab 12: Monday 12/12
  - Homework 12: Monday 12/19
  - Assignment 3: Friday 12/16 @ 11:59 pm
  - Assignment 4: Friday 12/16 @ 11:59 pm

Bonus Opportunities

- Bonus Lab due on 12/19
- Bonus Homework due on 12/19
- Assignment 5 is bonus and is due on 12/19
- 1 bonus point for entire class when OMETs response rate >= 80%
  - Currently at ~46% for both sections
  - Deadline is Sunday 12/11

Final Exam

- Same format as midterm
- Non-cumulative
- Date, time and location on PeopleSoft
  - MW Section: Monday 12/12 8-9:50 am (coffee served)
  - TuTh Section: Thursday 12/15 12:00-1:50 pm
- Same classroom as lectures
- Study guide and practice tests have been posted

Previous Lecture

- Two specific ways of finding augmenting paths in Ford-Fulkerson MaxFlow
  - BFS (Edmonds-Karp)
  - Priority First Search (PFS)
- The minimum-cut problem

This Lecture

- Push-Relabel Algorithm for Max Flow
- Using Dynamic Programming for solving Markov Decision Processes
- Lloyd’s local search algorithm
- Optimization to Dijkstra’s Shortest Paths algorithm in Google Maps

Push-Relabel Algorithm for Max Flow

- More efficient than Edmonds-Karp BFS implementation of Ford-Fulkerson
  - Running time: Theta(V^3) instead of Theta(E^2V)
- Local per vertex operations instead of global graph-wide updates

Push-Relabel Algorithm for Max Flow

- Each vertex has a height and excess flow value
- Flow can be **pushed** from a higher vertex to a lower neighbor over an edge with available residual capacity
- If a vertex has an excess flow and has no lower neighbors, **relabel** the vertex's height to be 1+min height of all neighbors
- Repeat relabel and flow push operations until all vertices except source and sink have 0 excess flow

Markov Decision Process

- A set of states
- A set of agent actions
- Probability of moving from each state to each other start by taking each action
- Reward function depends on state and action
- Expected value of a state =
  - Sum over all possible actions
    - probability of taking the action (depends on agent policy) * expected reward from the action
    - Expected reward from an action = immediate reward (from reward function) + Sum over all state transitions of probability of transition * value of next state

Using Dynamic Programming to Solve a MDP

- We are solving for an optimal policy
  - the probabilities of taking each action at each state
- Step 0: Start with an initial policy
  - e.g., all actions equally likely
- Step 1: Fill up the expected state values using the equations on the previous slide
  - optional: iterate until values converge
- Step 2: Modify policy to take the best action with probability 1 (given the current state values)
- Repeat Step 1 and 2 until policy converges

Clustering Problem

- Input: a set of n data points and a target number of clusters K
- Output: K clusters
  - K cluster centroids (central points)
  - A label from the set {1, .., K} for each of the n data points
  - such that:
  - Sum of squared distances from each point to its cluster's centroid is **minimum**
  - Sum_{all clusters}
    - Sum_{all points in cluster}
      - square of distance between data point and cluster centroid
    - for(int i=0; i<n; i++)
      - distance += (data[i] – centroid[cluster[i]])*(data[i] – centroid[cluster[i]])

Useful but hard problem!

- Used in
  - unsupervised machine learning algorithms
  - dimensionality reduction problems
- NP-hard!

Lloyd's Local Search Algorithm

- Start with an initial cluster assignment
- Assign each data point to the closest cluster centroid
- Recompute cluster centroids
- Repeat previous steps until no change in centroids and cluster assignments

Limitations of Lloyd's Algorithm

- Sensitive to initial clustering
- One of way of attempting to fix that is to select the initial centroids as far from each other as possible

K-Means ++

- Select the first one with uniform probability over all data points
- Repeat for each of the remaining K-1 initial centroids
  - For each data point
    - Compute its distance to the nearest previously selected centroid
  - Select the next centroid from the data points following a probability distribution that assigns ***higher probability to data points with larger distances***

Dijkstra’s Optimization

- Bidirectional search
  - start Dijkstra’s both from source **and**
  - from destination on reverse graph
  - alternate between the two runs or run them in parallel
  - Once a common vertex is reached by both runs, we update the shortest known distance
  - Stop both runs when the top of both heaps give a distance larger then the shortest known