Graph Algorithms

Overview

• Graph: abstract data type
  – \( G = (V,E) \) where \( V \) is set of nodes, \( E \) is set of edges \( \subseteq V \times V \)
• Structural properties of graphs
  – Power-law graphs, uniform-degree graphs
• Graph representations: concrete data type
  – Compressed-row/column, coordinate, adjacency list
• Graph algorithms
  – Operator formulation: abstraction for algorithms
  – Algorithms for single-source shortest-path (SSSP) problem
• Machine learning algorithms
  – Page-rank
  – Matrix-completion for recommendation systems

Structural properties of graphs

Graph-matrix duality

• Graph \((V,E)\) as a matrix
  – Choose an ordering of vertices
  – Number them sequentially
  – Fill in \(|V| \times |V|\) matrix
    • \( A(i,j) = w \) if graph has edge from node \( i \) to node \( j \) with label \( w \)
    – Called adjacency matrix of graph
  – Edge \((u \rightarrow v)\):
    • \( v \) is out-neighbor of \( u \)
    • \( u \) is in-neighbor of \( v \)
• Observations:
  – Diagonal entries: weights on self-loops
  – Symmetric matrix \( \leftrightarrow \) undirected graph
  – Lower triangular matrix \( \leftrightarrow \) no edges from lower numbered nodes to higher numbered nodes
  – Dense matrix \( \leftrightarrow \) clique (edge between every pair of nodes)
Sparse graphs

- **Terminology:**
  - Degree of node: number of edges connected to it
  - (Average) diameter of graph: average number of hops between two nodes

- **Power-law graphs**
  - Small number of very high degree nodes (see next slide for example)
  - Low diameter
  - "Six degrees of separation" (Károlyi 1929, Milgram 1967), on Facebook, it is 4.74
  - Typical of social network graphs like the Internet graph or the Facebook graph

- **Uniform-degree graphs**
  - Nodes have roughly same degree
  - High diameter
  - Road networks, IC circuits, finite-element meshes

- **Random (Erdős-Rényi) graphs**
  - Constructed by random insertion of edges
  - Mathematically interesting but few real-life examples

Airline route map: power-law graph

Node degree distribution of power-law graphs

Road map: uniform-degree graph

Graph representations: how to store graphs in memory
Three storage formats: CSR, CSC, COO

Labels on nodes are stored in a separate vector (not shown)

Adjacency list representation

Permits you to add and remove edges from graph.
Deleting edges: often it is more efficient to just to mark an edge as deleted rather than delete it physically from the list.

Overview

- Algorithms: usually specified by pseudocode
- We take a different approach:
  - operator formulation of algorithms
- Data-centric abstraction in which data structures play central role
- Advantages of operator formulation abstraction:
  - Connections between seemingly unrelated algorithms
  - Sources of parallelism and locality become evident
  - Suggests common set of mechanisms for exploiting parallelism and locality for all algorithms

Graph algorithms

From: https://www.thecrazyprogrammer.com
**Operator formulation of algorithms**

- **Algorithm** = Operator + Schedule

- **Operator**: local view of algorithm
  - Active node/edge: place in graph where some computation is needed
  - Operator: specification of computation
  - Activity: application of operator to active node
  - Neighborhood: Set of nodes/edges read/written by activity

- **Schedule**: global view of algorithm
  - Unordered algorithms:
    - Active nodes can be processed in any order
    - All schedules produce the same answer but performance may vary
  - Ordered algorithms:
    - Problem-dependent order on active nodes

**Graph problem: SSSP**

- **Problem**: single-source shortest-path (SSSP) computation
- **Formulation**:
  - Given an undirected graph with positive weights on edges, and a node called the source
  - Compute the shortest distance from source to every other node
- **Variations**:
  - Negative edge weights but no negative weight cycles
  - All-pairs shortest paths
  - Breadth-first search: all edge weights are 1
- **Applications**:
  - GPS devices for driving directions
  - Social network analyses: centrality metrics

**SSSP Problem**

- **Many algorithms**
  - Dijkstra (1959)
  - Bellman-Ford (1957)
  - Chained relaxation (1959)
  - Delta-stepping (1998)
- **Common structure**:
  - Each node has a label if that is updated repeatedly
  - Initially 0 for source and infinity for all other nodes
  - During algorithm, shortest known distance to that node from source
  - Terminate: shortest distance from source
- **SSSP Problem**
  - All of them use the same operator
    - \( \text{new-label}(u) \)
    - \( \text{if}(d(u) > d(u) + w(u)) \)
    - \( \text{then}(d(u) = d(u) + w(u)) \)
  - Differences between algorithms: structure
Parallelization:

- **Schedule**
  - pick active node at random
  - use a work-set or a priority queue to track active nodes
- **Main inefficiency:** number of node relaxations depends on the schedule
- **Parallelization:**
  - $??$

**Chaotic relaxation (1969)**

- **Active node**
  - node whose label has been updated
  - initially, only source is active
- **Implementation**
  - can be exponential in the size of graph
- **Algorithm:**
  - $O(|E|*|V|)$
  - prefer nodes with smaller labels since they are more likely to have reached final values
- **Schedule for processing nodes**
  - initial, only source is active
  - do this for all vertices
  - when a node is relaxed, it is moved to the final set
  - nodes in $S$ are processed only once

**Dijkstra’s algorithm (1959)**

- **Active nodes**
  - node whose label has been updated
  - initially, only source is active
- **Schedule for processing nodes**
  - prefer nodes with smaller labels since they are more likely to have reached final values
  - node is relaxed just once
  - $O(|E|*|V|)$
- **Parallelization:**
  - $??$
- **Main inefficiency:**
  - as we will see later, there is little parallelism for most graphs

**Delta-stepping (1998)**

- **Controlled chaotic relaxation**
  - Exploit the fact that SSSP is robust to priority inversions
  - "soft" priorities
- **Implementation of work-set:**
  - parameter: $\Delta$
  - sequence of sets
  - nodes whose current distance is between $\Delta$ and $(n+1)\Delta$ are put in the $n^{th}$ set
  - nodes in set $n$ are completed before processing of nodes in set $(n+1)$ are started
- **Algorithm:**
  - Initialize all vertices with infinity
  - Relax in each round
  - use a worklist
  - terminate when no change

**Bellman-Ford (1957)**

- **Algorithm:**
  - Initialize all vertices with infinity
  - Relax in each round
  - use a worklist
  - terminate when no change
  - $O(|E|*|V|)$
- **Main inefficiency:**
  - as we will see later, there is little parallelism for most graphs
Summary of SSSP Algorithms

- **Chaotic relaxation**
  - unordered, data-driven algorithm
  - use sets/multisets for work-set
  - amount of work depends on schedule: can be exponential in size of graph

- **Dijkstra’s algorithm**
  - ordered, data-driven algorithm
  - use priority queue for work-set
  - \( O(|V|\log(|E|)) \): work-efficient but little parallelism

- **Delta-stepping**
  - controlled chaotic relaxation: parameter \( \Delta \)
  - \( \Delta \) permits trade-off between parallelism and work-efficiency

- **Bellman-Ford algorithm**
  - unordered, topology-driven algorithm
  - \( O(|V||E|) \) time

Machine learning

- Many machine learning algorithms are sparse graph algorithms

- Examples:
  - Page rank: used to rank webpages to answer Internet search queries
  - Recommender systems: used to make recommendations to users in Netflix, Amazon, Facebook etc.

Web search

- **When you type a set of keywords to do an Internet search, which web-pages should be returned and in what order?**

  **Basic idea:**
  - offline:
    - crawl the web and gather webpages into data center
    - build an index from keywords to webpages
  - online:
    - when user types keywords, use index to find all pages containing the keywords
  - key problem:
    - usually you end up with tens of thousands of pages
    - how do you rank these pages for the user?

Ranking pages

- **Manual ranking**
  - Yahoo did something like this initially, but this solution does not scale

- **Word counts**
  - order webpages by how many times keywords occur in webpages
  - problem: easy to mess with ranking by having lots of meaningless occurrences of keyword

- **Citations**
  - analogy with citations to articles
  - if lots of webpages point to a webpage, rank it higher
  - problem: easy to mess with ranking by creating lots of useless pages that point to your webpage

- **PageRank**
  - extension of citations idea
  - weight link from webpage A to webpage B by “importance” of A
  - if A has few links to it, its links are not very “valuable”
  - how do we make this into an algorithm?
**Web graph**

- Directed graph: nodes represent webpages, edges represent links
  - Edge from u to v represents a link in page u to page v
- Size of graph: commoncrawl.org (2012)
  - 3.5 billion nodes
  - 128 billion links
- Intuitive idea of pageRank algorithm:
  - Each node in graph has a weight (pageRank) that represents its importance
  - Assume all edges out of a node are equally important
  - Importance of edge is scaled by the pageRank of source node

**PageRank (simple version)**

Graph: $G = (V,E)$
$|V| = N$

- Iterative algorithm:
  - Compute a series $PR_0, PR_1, PR_2, \ldots$ of node labels
- Iterative formula:
  - $\forall v \in V. PR_0(v) = \frac{1}{N}$
  - $\forall v \in V. PR_{i+1}(v) = \frac{1-d}{N} + d \sum_{u \in \text{in-neighbors}(v)} \frac{PR(u)}{\text{out-degree}(u)}$

- Implement with two fields $PR_{\text{current}}$ and $PR_{\text{next}}$ in each node

**Page Rank (contd.)**

- Small twist needed to handle nodes with no outgoing edges
- Damping factor: d
  - Small constant: 0.85
  - Assume each node may also contribute its pageRank to a randomly selected node with probability (1-d)
- Iterative formula:
  - $\forall v \in V. PR_0(v) = \frac{1}{N}$
  - $\forall v \in V. PR_{i+1}(v) = \frac{1-d}{N} + d \sum_{u \in \text{in-neighbors}(v)} \frac{PR(u)}{\text{out-degree}(u)}$

**PageRank example**

- Nice example from Wikipedia
- Note
  - B and E have many in-edges but pageRank of B is much greater
  - C has only one in-edge but high pageRank because its in-edge is very valuable
- Caveat:
  - Search engines use many criteria in addition to pageRank to rank webpages
Matrix-vector multiplication

- Matrix computation: $y = Ax$
- Graph interpretation:
  - Each node $i$ has two values (labels) $x(i)$ and $y(i)$
  - Each node $i$ updates its label $y$ using the $x$ value from each out-neighbor $j$ scaled by the label on edge $(i,j)$
  - Topology-driven, unordered algorithm
- Observation:
  - Graph perspective shows dense MVM is special case of sparse MVM
  - What is the interpretation of $y = ATx$?
- Page-rank can be expressed as generalized MVM
  - Reinterpret $+$ and $*$ operations

PageRank discussion

- Vertex program (Pregel):
  - value at node is updated using values at immediate neighbors
  - very limited notion of neighborhood but adequate for pageRank and some ML algorithms
- CombBlas: combinatorial BLAS
  - generalized sparse MVM: $+* \text{ in MVM are generalized to other operations like } \lor \text{ and } \land$
  - adequate for pageRank
- Interesting application of TAO
  - standard pageRank is topology-driven
  - can you think of a data-driven version of pageRank?

Recommender system

- Problem
  - given a database of users, items, and ratings given by each user to some of the items
  - predict ratings that user might give to items he has not rated yet (usually, we are interested only in the top few items in this set)
- Netflix challenge
  - in 2006, Netflix released a subset of their database and offered $1$ million prize to anyone who improved their algorithm by $10$
  - triggered a lot of interest in recommender systems
  - prize finally given to BellKor’s Pragmatic Chaos team in 2009

Data structure for database

- Sparse matrix view:
  - rows are users
  - columns are movies
  - $A(u,m) = v$ is user $u$ has given rating $v$ to movie $m$
- Graph view:
  - bipartite graph
  - two sets of nodes, one for users, one for movies
  - edge $(u,m)$ with label $v$
- Recommendation problem:
  - predict missing entries in sparse matrix
  - predict labels of missing edges in bipartite graph
One approach: matrix completion

- Optimization problem
  - Find $m \times k$ matrix $W$ and $k \times n$ matrix $H$ ($k << \min(m,n)$) such that $A \approx WH$
  - Low-rank approximation
  - $H$ and $W$ are dense so all missing values are predicted
- Graph view
  - Label of user nodes $i$ is vector corresponding to row $W_i$
  - Label of movie node $j$ is vector corresponding to column $H_j$
  - If graph has edge $(u,m)$, inner product of labels on $u$ and $m$ must be approximately equal to label on edge

One algorithm: SGD

- Stochastic gradient descent (SGD)
- Iterative algorithm:
  - Initialize all node labels to some arbitrary values
  - Iterate until convergence
    - Visit all edges $(u,m)$ in some order and update node labels at $u$ and $m$ based on the residual
- TAO analysis:
  - Active edges: topology-driven, unordered
  - What algorithm does this remind you of?
    - Bellman-Ford

What we have learned

- Operator formulation:
  - Data-centric view of algorithms
- TAO classification
- Location of active nodes
  - Topology-driven algorithms
  - Data-driven algorithms
  - Data-driven algorithm may be more work-efficient than topology-driven one
- Ordering of active nodes
  - Unordered algorithms
  - Ordered algorithms
- Some problems
  - Have both ordered and unordered algorithms (e.g. SSSP)
  - Have both topology-driven and data-driven algorithms (e.g. SSSP, pagerank)
Questions

• What are the sources of parallelism and locality in algorithms?
• Can the operator formulation help us in answering this question?
• How do we exploit parallelism and locality efficiently?