Problem: Large Graphs

- It is expensive to find optimal paths in large graphs, using BFS or Dijkstra's algorithm (for weighted graphs)
- How can we search large graphs efficiently by using “commonsense” about which direction looks most promising?

Best-First Search

- The Manhattan distance ($\Delta x + \Delta y$) is an estimate of the distance to the goal
  - It is a search heuristic
- Best-First Search
  - Order nodes in priority to minimize estimated distance to the goal
- Compare: BFS / Dijkstra
  - Order nodes in priority to minimize distance from the start
Obstacles

- Best-FS eventually will expand vertex to get back on the right track

Non-Optimality of Best-First

Path found by Best-first

Shortest Path

Improving Best-First

- Best-first is often tremendously faster than BFS/Dijkstra, but might stop with a non-optimal solution
- How can it be modified to be (almost) as fast, but guaranteed to find optimal solutions?
  - One of the first significant algorithms developed in AI
  - Widely used in many applications

A*

- Exactly like Best-first search, but using a different criteria for the priority queue:
  - minimize (distance from start) + (estimated distance to goal)
  - priority \( f(n) = g(n) + h(n) \)
    - \( g(n) \) = true distance from start
    - \( h(n) \) = heuristic distance to goal

Optimality of A*

- Suppose the estimated distance is always less than or equal to the true distance to the goal
  - heuristic is a lower bound
- Then: when the goal is removed from the priority queue, we are guaranteed to have found a shortest path!
Application of A*: Speech Recognition

(Simplified) Problem:
- System hears a sequence of 3 words
- It is unsure about what it heard
  - For each word, it has a set of possible "guesses"
  - E.g.: Word 1 is one of { "hi", "high", "I" }
- What is the most likely sentence it heard?

Speech Recognition as Shortest Path

- Convert to a shortest-path problem:
  - Utterance is a "layered" DAG
  - Begins with a special dummy "start" node
  - Next: A layer of nodes for each word position, one node for each word choice
  - Edges between every node in layer i to every node in layer i+1
    - Cost of an edge is smaller if the pair of words frequently occur together in real speech
    - Technically: log probability of co-occurrence
- Finally: a dummy "end" node
- Find shortest path from start to end node

Summary: Graph Search

- Depth First
  - Little memory required
  - Might find non-optimal path
- Breadth First
  - Much memory required
  - Always finds optimal path
- Iterative Depth-First Search
  - Repeated depth-first searches, little memory required
- Dijkstra’s Short Path Algorithm
  - Like BFS for weighted graphs
- Best First
  - Can visit fewer nodes
  - Might find non-optimal path
- A*
  - Can visit fewer nodes than BFS or Dijkstra
  - Optimal if heuristic estimate is a lower-bound

Dynamic Programming

- Algorithmic technique that systematically records the answers to sub-problems in a table and re-uses those recorded results (rather than re-computing them).
- Simple Example: Calculating the Nth Fibonacci number.
  $\text{Fib}(N) = \text{Fib}(N-1) + \text{Fib}(N-2)$

Floyd-Warshall

- for (int $k = 1; k <= V; k++)$
- for (int $i = 1; i <= V; i++$)
  - for (int $j = 1; j <= V; j++$)
  - if ( $(M[i][k] + M[k][j]) < M[i][j]$ )
    $M[i][j] = M[i][k] + M[k][j]$

Invariant: After the kth iteration, the matrix includes the shortest paths for all pairs of vertices $(i,j)$ containing only vertices $1..k$ as intermediate vertices.
Initial state of the matrix:

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>2</td>
<td>-4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>-</td>
<td>0</td>
<td>-2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>c</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

\[ M[i][j] = \min(M[i][j], M[i][k]+ M[k][j]) \]

Floyd-Warshall - for All-pairs shortest path

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>-4</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>-</td>
<td>0</td>
<td>-2</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>c</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Network Flows

- Given a weighted, directed graph \( G=(V,E) \)
- Treat the edge weights as \textit{capacities}
- How much can we flow through the graph?

Network flow: definitions

- Define special \textit{source} \( s \) and \textit{sink} \( t \) vertices
- Define a \textit{flow} as a function on edges:
  - \textbf{Capacity}: \( f(v,w) \leq c(v,w) \)
  - \textbf{Conservation}: \( \sum_{u \neq s} f(u,v) = 0 \) for all \( v \) except source, sink
  - \textbf{Value of a flow}: \( |f| = \sum_{v \neq s} f(s,v) \)
  - Saturated edge: when \( f(v,w) = c(v,w) \)

Network flow: definitions

- \textbf{Capacity}: you can’t overload an edge
- \textbf{Conservation}: Flow entering any vertex must equal flow leaving that vertex
- We want to maximize the value of a flow, subject to the above constraints
Network Flows

- Given a weighted, directed graph $G=(V,E)$
- Treat the edge weights as capacities
- How much can we flow through the graph?

A Good Idea that Doesn’t Work

- Start flow at 0
- “While there’s room for more flow, push more flow across the network!”
  - While there’s some path from $s$ to $t$, none of whose edges are saturated
  - Push more flow along the path until some edge is saturated
  - Called an “augmenting path”

How do we know there’s still room?

- Construct a residual graph:
  - Same vertices
  - Edge weights are the “leftover” capacity on the edges
  - If there is a path $s \rightarrow t$ at all, then there is still room

Example (1)

Example (2)

Include the residual capacities

Example (3)

Augment along ABFD by 1 unit (which saturates BF)
Example (4)
Augment along ABEFD (which saturates BE and EF)

Now what?
• There’s more capacity in the network...
• ...but there’s no more augmenting paths

Network flow: definitions
• Define special source s and sink t vertices
• Define a flow as a function on edges:
  – Capacity: \( f(v, w) \leq c(v, w) \)
  – Skew symmetry: \( f(v, w) = -f(w, v) \)
  – Conservation: \( \sum_{v \in \text{domain}} f(u, v) = 0 \) for all \( u \) except source, sink
  – Value of a flow: \( \sum_{v} f(u, v) \)
  – Saturated edge: when \( f(v, w) = c(v, w) \)

Main idea: Ford-Fulkerson method
• Start flow at 0
• "While there’s room for more flow, push more flow across the network!"
  – While there’s some path from s to t, none of whose edges are saturated
  – Push more flow along the path until some edge is saturated
  – Called an “augmenting path”

How do we know there’s still room?
• Construct a residual graph:
  – Same vertices
  – Edge weights are the “leftover” capacity on the edges
  – Add extra edges for backwards-capacity too!
  – If there is a path s \( \rightarrow \) t at all, then there is still room
Example (5)
Add the backwards edges, to show we can "undo" some flow

Example (6)
Augment along AEBCD (which saturates AE and EB, and empties BE)

Example (7)
Final, maximum flow

How should we pick paths?
- Two very good heuristics (Edmonds-Karp):
  - Pick the largest-capacity path available
    - Otherwise, you'll just come back to it later...so may as well pick it up now
  - Pick the shortest augmenting path available
    - For a good example why...

Don’t Mess this One Up
Augment along ABCD, then ACBD, then ABCD, then ACBD…
Should just augment along ACD, and ABD, and be finished

Running time?
- Each augmenting path can’t get shorter...and it can’t always stay the same length
  - So we have at most O(E) augmenting paths to compute for each possible length, and there are only O(V) possible lengths.
  - Each path takes O(E) time to compute
- Total time = O(E^2V)
Network Flows

• What about multiple sources?

One more definition on flows

• We can talk about the flow from a set of vertices to another set, instead of just from one vertex to another:

\[ f(X,Y) = \sum_{x \in X} \sum_{y \in Y} f(x,y) \]

- Should be clear that \( f(X,X) = 0 \)
- So the only thing that counts is flow between the two sets

Minimum cuts

• If we cut \( G \) into \( (S, T) \), where \( S \) contains the source \( s \) and \( T \) contains the sink \( t \),

• Of all the cuts \( (S, T) \) we could find, what is the smallest (max) flow \( f(S, T) \) we will find?

Network cuts

• Intuitively, a cut separates a graph into two disconnected pieces

• Formally, a cut is a pair of sets \( (S, T) \), such that

\[ V = S \cup T \]

\[ S \cap T = \{\} \]

and \( S \) and \( T \) are connected subgraphs of \( G \)

Min Cut - Example (8)

Capacity of cut = 5
Coincidence?
• NO! Max-flow always equals Min-cut
• Why?
  – If there is a cut with capacity equal to the flow, then we have a maxflow:
    • We can’t have a flow that’s bigger than the capacity cutting the graph! So any cut puts a bound on the maxflow, and if we have an equality, then we must have a maximum flow.
  – If we have a maxflow, then there are no augmenting paths left
    • Or else we could augment the flow along that path, which would yield a higher total flow.
    – If there are no augmenting paths, we have a cut of capacity equal to the maxflow
    • Pick a cut (S,T) where S contains all vertices reachable in the residual graph from s, and T is everything else. Then every edge from S to T must be saturated (or else there would be a path in the residual graph). So c(S,T) = f(S,T) = f(s,t) = |f| and we’re done.

GraphCut
http://www.cc.gatech.edu/projects/graphcuttextures/

Dictionary Coding
• Does not use statistical knowledge of data.
• Encoder: As the input is processed develop a dictionary and transmit the index of strings found in the dictionary.
• Decoder: As the code is processed reconstruct the dictionary to invert the process of encoding.
• Examples: LZW, LZ77, Sequitur,
• Applications: Unix Compress, gzip, GIF

CSE 326: Data Structures
Dictionaries for Data Compression

LZW Encoding Algorithm
Repeat
find the longest match w in the dictionary
put wa in the dictionary where a was the unmatched symbol

LZW Encoding Example (1)

<table>
<thead>
<tr>
<th>Dictionary</th>
<th>a b a b a b a</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 a</td>
<td>1 b</td>
</tr>
</tbody>
</table>
LZW Encoding Example (2)

Dictionary
0  a
1  b
2  ab

Dictionary
0  a
1  b
2  ab

LZW Encoding Example (3)

Dictionary
0  a
1  b
2  ab

Dictionary
0  a
1  b
2  ab

LZW Encoding Example (4)

Dictionary
0  a
1  b
2  ab
3  ba
4  aba

Dictionary
0  a
1  b
2  ab
3  ba
4  aba
5  abab

LZW Encoding Example (5)

Dictionary
0  a
1  b
2  ab
3  ba
4  aba
5  abab

Dictionary
0  a
1  b
2  ab
3  ba
4  aba
5  abab

LZW Encoding Example (6)

Dictionary
0  a
1  b
2  ab
3  ba
4  aba
5  abab

Dictionary
0  a
1  b
2  ab
3  ba
4  aba
5  abab

LZW Decoding Algorithm

• Emulate the encoder in building the dictionary. Decoder is slightly behind the encoder.

initialize dictionary;
decode first index to w;
put w in dictionary;
repeat
decode the first symbol s of the index;
complete the previous dictionary entry with s;
finish decoding the remainder of the index;
put w in the dictionary where w was just decoded;
LZW Decoding Example (1)

**Dictionary**

0  a
1  b
2  a?

**012436**

a

LZW Decoding Example (2a)

**Dictionary**

0  a
1  b
2  ab

**012436**

a  b

LZW Decoding Example (2b)

**Dictionary**

0  a
1  b
2  ab
3  b?

**012436**

a  b

LZW Decoding Example (3a)

**Dictionary**

0  a
1  b
2  ab
3  ba

**012436**

a  b a

LZW Decoding Example (3b)

**Dictionary**

0  a
1  b
2  ab
3  ba
4  ab?

**012436**

a  b ab

LZW Decoding Example (4a)

**Dictionary**

0  a
1  b
2  ab
3  ba
4  aba

**012436**

a  b ab a
LZW Decoding Example (4b)

Dictionary
0  a
1  b
2  ab
3  ba
4  aba
5  abab

012436
a  b  ab ab a

LZW Decoding Example (5a)

Dictionary
0  a
1  b
2  ab
3  ba
4  aba
5  abab

012436
a  b  ab ab a  b

LZW Decoding Example (5b)

Dictionary
0  a
1  b
2  ab
3  ba
4  aba
5  abab
6  bab

012436
a  b  ab ab a  b  a

LZW Decoding Example (6a)

Dictionary
0  a
1  b
2  ab
3  ba
4  aba
5  abab
6  bab
7  bab?

012436
a  b  ab ab a  b  a  b

LZW Decoding Example (6b)

Dictionary
0  a
1  b
2  ab
3  ba
4  aba
5  abab
6  bab
7  bab?

012436
a  b  ab ab a  b  a  b  a  b

Decoding Exercise

Base Dictionary
0  a
1  b
2  c
3  d
4  r

014020357
Bounded Size Dictionary

- Bounded Size Dictionary
  - \( n \) bits of index allows a dictionary of size \( 2^n \)
  - Doubtful that long entries in the dictionary will be useful.
- Strategies when the dictionary reaches its limit.
  1. Don’t add more, just use what is there.
  2. Throw it away and start a new dictionary.
  3. Double the dictionary, adding one more bit to indices.
  4. Throw out the least recently visited entry to make room for the new entry.

Notes on LZW

- Extremely effective when there are repeated patterns in the data that are widely spread.
- Negative: Creates entries in the dictionary that may never be used.
- Applications:
  - Unix compress, GIF, V.42 bis modem standard

LZ77

- Ziv and Lempel, 1977
- Dictionary is implicit
- Use the string coded so far as a dictionary.
- Given that \( x_1 x_2 \ldots x_i \) has been coded we want to code \( x_{n+1} x_{n+2} \ldots x_{n+k} \) for the largest \( k \) possible.

Solution A

- If \( x_{n+1} x_{n+2} \ldots x_{n+k} \) is a substring of \( x_1 x_2 \ldots x_n \)
  then \( x_{n+1} x_{n+2} \ldots x_{n+k} \) can be coded by \( <j,k> \)
  where \( j \) is the beginning of the match.
- Example
  \[
  \text{ababababa cababababababababababababababab....}
  \]
  coded
  \[
  \text{ababababa babababa babababa babababa babababa babababa babababa babababa babababa babababa babababa babababa babababa babababa bababababababababababababababab....}
  \]
  \(<2,8>\)

Solution A Problem

- What if there is no match at all in the dictionary?
  \[
  \text{ababababa cababababababababababababababab....}
  \]
  coded
- Solution B. Send tuples \( <j,k,x> \) where
  - If \( k = 0 \) then \( x \) is the unmatched symbol
  - If \( k > 0 \) then the match starts at \( j \) and is \( k \) long and the unmatched symbol is \( x \).

Solution B

- If \( x_{n+1} x_{n+2} \ldots x_{n+k} \) is a substring of \( x_1 x_2 \ldots x_n \)
  and \( x_{n+1} x_{n+2} \ldots x_{n+k} x_{n+k+1} \) is not then
  \( x_{n+1} x_{n+2} \ldots x_{n+k} x_{n+k+1} \) can be coded by
  \( <j,k,x_{n+k+1}> \)
  where \( j \) is the beginning of the match.
- Examples
  \[
  \text{ababababa cababababababababababababababab....}
  \]
  \[
  \text{ababababa c ababababababababababababababab....}
  \]
  \(<0,0,c><1,9,b>\)
Solution B Example

- The matching string can include part of itself!
- If $x_{n+1}x_{n+2}...x_{nk}$ is a substring of $x_1x_2...x_n x_{n+1}x_{n+2}...x_{nk}$ that begins at $j \leq n$ and $x_{n+1}x_{n+2}...x_{nk}x_{n+k+1}$ is not then $x_{n+1}x_{n+2}...x_{nk}x_{n+k+1}$ can be coded by $<j,k,x>$

Surprise Code!

- We want the triples $<j,k,x>$ to be of bounded size. To achieve this we use bounded buffers.
  - Search buffer of size $s$ is the symbols $x_{n-s+1}...x_n$.
  - Look-ahead buffer of size $t$ is the symbols $x_{n+1}...x_{n+t}$.
  - Match pointer can start in search buffer and go into the look-ahead buffer but no farther.

Solution C

- The matching string can include part of itself!
- If $x_{n+1}x_{n+2}...x_{nk}$ is a substring of $x_1x_2...x_n x_{n+1}x_{n+2}...x_{nk}$ that begins at $j \leq n$ and $x_{n+1}x_{n+2}...x_{nk}x_{n+k+1}$ is not then $x_{n+1}x_{n+2}...x_{nk}x_{n+k+1}$ can be coded by $<j,k,x>$