More on Graph Algorithms

Konstantin Tretyakov (kt@ut.ee)

MTAT.03.238 Advanced Algorithmics
April 13, 2011
So far

- Graph traversals
- Topological sorting
- Connected & strongly connected components
- Minimal spanning trees
- SSSP
- Transitive Closure
- APSP
- Max-Flow
- Clustering
So far

- Graph traversals (BFS, DFS, Dijkstra, Random)
- Topological sorting
- Connected & strongly connected components
- Minimal spanning trees (Prim, Kruskal)
- SSSP (Bellman-Ford, Dijkstra)
- Transitive Closure (Warshall)
- APSP (Floyd-Warshall)
- Max-Flow (Ford-Fulkerson)
- Clustering (MCL)
So far

- 1956: Graph traversals (BFS, DFS, Dijkstra, Random)
- 1962: Topological sorting
- 1972: Connected & strongly connected components
- 1957: Minimal spanning trees (Prim, Kruskal)
- 1958: SSSP (Bellman-Ford, Dijkstra)
- 1962: Transitive Closure (Warshall)
- 1959: APSP (Floyd-Warshall)
- 1962: Max-Flow (Ford-Fulkerson)
- 2000: Clustering (MCL)
Today: Some more tricks

1995  ▶ All-pairs shortest paths
2001  ▶ Approximate shortest paths
1995  ▶ Euler tour trees
All-pairs shortest paths (APSP)

- In general, $O(n^3)$
All-pairs shortest paths (APSP)

- In general, $O(n^3)$
  - E.g. Floyd-Warshall
  - Also, $n$ times Dijkstra traversal ($O(nm + n^2\log n)$)

- For particular cases, improvements are possible
All-pairs shortest paths (APSP)

- **Seidel’s algorithm**
  - Given: Unweighted, undirected graph $G$ with adjacency matrix $A$
  - 1) Computes all-pairs distances matrix
  - 2) Computes *witnesses* for all-pairs shortest paths
Seidel’s algorithm: Part 1

- Make a 2-step graph $H$: 

![Diagram of Seidel's algorithm](image-url)
Seidel’s algorithm: Part 1

- Make a 2-step graph $H$:

  $A$ $\rightarrow$ $(A \times A)$ or $A$
Seidel’s algorithm: Part 1

- Make a 2-step graph $H$:

A

(A x A) or A

Naïve approach $O(n^3)$, but it can be done in $O(n^{2.5})$
Seidel’s algorithm: Part 1

- Make a 2-step graph $H$
- Find APSP in $H$, recursively

\[
\begin{array}{cccc}
0 & 1 & 1 & 2 \\
1 & 0 & 1 & 1 \\
1 & 1 & 0 & 1 \\
2 & 1 & 1 & 0 \\
\end{array}
\]
Seidel’s algorithm: Part 1

- Make a 2-step graph $H$
- Find APSP in $H$, recursively
- Derive original distances:
  - If $\text{dist}_H(a, b) = k$ then
    - $\text{dist}_G(a, b) = 2k$ OR $\text{dist}_G(a, b) = 2k-1$
  - Depends on whether $\text{dist}_H(a, c) = k-1$
    - for $c$ – a neighbor of $b$
Seidel’s algorithm: Part 1

- Make a 2-step graph $H$
- Find APSP in $H$, recursively
- Derive original distances:
  - If $\text{dist}_H(a,b) = k$ then
    - $\text{dist}_G(a,b) = 2k$ OR
    - $\text{dist}_G(a,b) = 2k - 1$
  - Depends on whether $\text{dist}_H(a,c) = k - 1$
    - for $c$ – a neighbor of $b$
Seidel’s algorithm: Part 1

- Make a 2-step graph H
- Find APSP in H, recursively
- Derive original distances:
  - If $\text{dist}_H(a,b) = k$ then
    - $\text{dist}_G(a,b) = 2k$ \text{ OR }
    - $\text{dist}_G(a,b) = 2k - 1$
  - Depends on whether $\text{dist}_H(a,c) = k - 1$
    for c – a neighbor of b
Seidel’s algorithm: Part 1

- Make a 2-step graph \( H \)
- Find APSP in \( H \), recursively
- Derive original distances:
  - If \( \text{dist}_H(a,b) = k \) then
    \[
    \text{dist}_G(a,b) = 2k \quad \text{OR} \quad \text{dist}_G(a,b) = 2k - 1
    \]
  - Depends on whether \( \text{dist}_H(a,c) = k - 1 \)
    for \( c \) – a neighbor of \( b \)

More on Graph Algorithms
Seidel’s Algorithm: Part 1

function APD(A):
  Z = ((A*A) or A)
  Z(i,i) = 0 for i = 1..n
  if (Z == 1) for all (i!=j):
    return D = 2*Z - A
  else:
    T = APD(Z)
    D(i,j) = 2*T(i,j)-1
    if (exists neighbor k of j with T(i,k)<T(i,j))
      else 2*T(i,j)
Seidel’s Algorithm: Part 1

APD(graph with diameter < n)
  O(n^{2.5}) computations

APD(graph with diameter < n/2)
  O(n^{2.5}) computations

APD(graph with diameter < n/4)
  ...

APD(graph with diameter 1)

------------------------------------

Total O(n^{2.5} \log(n))
Once we have the distance matrix, how can we extract shortest paths?
Seidel’s Algorithm: Part 2

Once we have the distance matrix, how can we extract shortest paths?

Compute witnesses:
for each pair \((i, j)\) find vertex \(k\) that lies on the shortest path \((i \ldots j)\)
Seidel’s Algorithm: Part 2

Once we have the distance matrix, how can we extract shortest paths?

Compute *witnesses*:
for each pair \((i, j)\) find vertex \(k\) that lies on the shortest path \((i \ldots j)\)
Seidel’s Algorithm

- Vertex $k$ is a witness for $(i,j)$ if:
  - $A(i,k) = 1$ and $D(k,j) = D(i,j) - 1$

- Naïve computation: $O(n^3)$
- Can be done using a probabilistic algorithm in $O(n^2 \log(n))$
Seidel’s algorithm

- Part 1: $O(n^{2.5} \log(n))$
  Requires efficient matrix multiplication

- Part 2: $O(n^2 \log(n))$
  Uses a probabilistic algorithm

Total: $O(n^{2.5} \log(n))$
Seidel’s algorithm: Conclusion

- Matrix multiplication for graph problems
- Recursive reduction to a problem half the size
- Notion of witnesses for shortest paths.
Approximate shortest paths

- Main idea: Landmarks

\[ 3 \leq d(A, B) \leq 7 \]
# Landmark-based Embedding

<table>
<thead>
<tr>
<th></th>
<th>L1</th>
<th>L2</th>
<th>L3</th>
<th>L4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>
Landmark-based Embedding

A → L1: 5  L2: 3  L3: 1  L4: 7

C → L1: 3  L2: 3  L3: 2  L4: 5

...
Thorup’s Algorithm

- Idea: for each vertex $v$ maintain a (large) set of landmarks with $m$ designated ones
Thorup’s Algorithm

- Compute distance $d(u,v)$ by finding a designated landmark of one vertex, which matches some landmark of the other vertex.
Thorup’s Algorithm

- Compute landmarks as follows:
  - Make a sequence of $K+1$ nested sets:
    - $V = A_0$
    - $A_1$
    - $A_2$
    - ...
    - $A_k = \{ \}$

  (Each next set – a random sample from the previous of size $n^{-\frac{1}{k}}$)
Thorup’s Algorithm
Thorup’s Algorithm
Thorup's Algorithm

Approximation $\leq (2k-1) \text{ True distance}$
Thorup’s Algorithm

- For a given set of landmarks $A_k$, how do you compute distances from each vertex to the closest landmark in $A_k$?
Thorup’s Algorithm

- Add a new source vertex and do a BFS from it.
Thorup’s Algorithm: Conclusion

- Nontrivial landmark-based algorithm
- You can prove bounds on performance
- Introduce virtual vertices to solve problems
Euler Tour Trees

Task: Maintain a forest under operations:

- Link(v, w)
- Cut(v, w)
- Find-subtree(v)
(p o) (o n) (n n) (n o) (o o) (o p) (p p)
(a a) (a b) (b b) (b c) (c d) (d d) (d e) (e e) (e f) (f f) (f g) (g g) (g h) (h h) (h i) (i i) (i k) (k k) (k i) (i l) (l l) (l i) (i j) (j j) (j i) (i h) (h g) (g f) (f e) (e d) (d c) (c c) (c b) (b m) (m m) (m b) (b a)
(p o) (o n) (n n) (n o) (o o) (o p) (p p)
(a a) (a b) (b b) (b c) (c d) (d d) (d e) (e e) (e f) (f f) (f g) (g g) (g h) (h h) (h i) (i i) (i k) (k k) (k i) (l l) (l i) (i j) (j j) (j i) (i h) (h g) (g f) (f e) (e d) (d c) (c c) (c b) (b m) (m m) (m b) (b a)
(p o) (o n) (n n) (n o) (o o) (o p) (p p) 

(a a) (a b) (b b) (b c) (c d) (d d) (d e) (e e) (e f) (f f) (f g) (g g) (g h) (h h) (h i) (i i) (i k) (k k) (k i) (i l) (l l) (l i) (i j) (j j) (j i) (i h) (h g) (g f) (f e) (e d) (d c) (c c) (c b) (b m) (m m) (m b) (b a)
More on Graph Algorithms

(p o) (o n) (n o) (n n) (o o) (o p) (p p)

(a a) (a b) (b b) (b c) (c d) (d d) (d e) (e e) (e f) (f f) (f g) (g g) (g h) (h h) (h i) (i i) (i k) (k k) (k i) (i l) (l l) (l i) (i j) (j j) (j i) (i h) (h g) (g f) (f e) (e d) (d c) (c c) (c b) (b m) (m m) (m b) (b a)
(a a) (a b) (b b) (b c) (c d) (d d) (d e) (e e) (e f) (f f) (f g) (g g) (g h) (h h) (h i) (i i) (i k) (k k) (k i) (i l) (l l) (l i) (i j) (j j) (j i) (i h) (h g) (g f) (f e) (e d) (d c) (c c) (c b) (b m) (m m) (m b) (b a)
Summary

- All-pairs shortest paths
- Approximate shortest paths
- Euler tour trees