Shortest Paths

Jon Sneyers
jon@cs.kuleuven.be

K.U.Leuven, Belgium
This presentation is based on the lectures at the

Summer School on Shortest Paths
Between Algorithms and Optimization
July 4-8 2005
Dept. Computer Science (DIKU)
University of Copenhagen, Denmark
Overview

- Introduction
- Variants of the problem
- Scanning methods
  - Non-negative weights
  - Arbitrary weights
- Fast matrix multiplication methods
- State of the art: an overview
- Conclusion
The Shortest Path problem

- One of the most basic (and most studied) problems in algorithmic graph theory
- Sub-problem in many graph problems
- Studied for over 50 years; still very active area of research
- Topic of the 9th DIMACS implementation challenge (2005-2006)
The Shortest Path problem

- Given a (simple) graph \( G = (V, E) \), find the distance between nodes (length of a shortest path)

- notation: \( n = |V|, m = |E| \)

\[ d(a, h) = ? \]
The Shortest Path problem

- Given a (simple) graph $G = (V, E)$, find the distance between nodes (length of a shortest path)

- notation: $n = |V|$, $m = |E|$
The Shortest Path problem

- Given a (simple) graph $G = (V, E)$, find the distance between nodes (length of a shortest path)

![Graph with node labels and edge weights]

- notation: $n = |V|$, $m = |E|$

$d(a, h) = 11$
Some basic notions

- \( O(f(m,n)) \) : asymptotic upper bound
- \( \Theta(f(m,n)) \) : asymptotic tight bound
- \( \tilde{O}(f(n)) \) : ignoring polylogarithmic factors
  abbreviation for \( O(f(n)(\log n)^c) \)
- \( \alpha(m,n) \) : Tarjan’s inverse-Ackermann function (constant in practice)
- we can assume that \( n \leq m \leq n(n - 1)/2 \) (undirected)
  - sparse graph: number of edges \( m \) is \( O(n) \)
  - dense graph: number of edges \( m \) is \( \Theta(n^2) \)
  - “sufficiently sparse/dense” : e.g. algorithm A runs in time \( O(n^{2.38}) \),
    algorithm B in time \( O(mn) \). “For sufficiently dense graphs, algorithm A is
    faster.” (e.g. \( m = n\sqrt{n} \)
Variants of the SP problem

- Single source (SSSP) / All pairs (APSP) / Point-to-point (P2PSP)
- Directed / Undirected graph
- Edge weights $\ell$:
  - Real weights: addition-compare RAM model
  - Integer weights: word RAM model
  - Unweighted: 1 or $+\infty$
- Negative weights allowed?
  $\Rightarrow$ negative-weight cycles!
  $\Rightarrow$ not interesting in undirected case
- Exact / Approximate (additive or multiplicative error?)
- Static / Dynamic
More variants of the SP problem

- Restricted families of graphs
  - planar graphs
  - euclidean distances
  - acyclic graphs: scan nodes in topological order, linear time

- Parallel algorithms

- Lightest shortest path (minimal number of edges)

- Constrained shortest path: (NP-complete)
  edges have weight and resource consumption
  path is feasible if total resource consumption no more than $R$
  find shortest feasible path

- $k$ shortest (simple) paths
Speakers at the summer school

- **Andrew Goldberg** (Microsoft Research): *bucket-based methods (integer-weight SSSP), P2PSP*
- **Natashia Boland** (Melbourne) - **Irina Dumitrescu** (Montreal): *resource constrained SP*
- **Mikkel Thorup** (AT&T Labs): *non-negative integer-weight SSSP*
- **Uri Zwick** (Tel Aviv): *fast matrix multiplication methods (many variants)*
- **Camil Demetrescu** - **Giuseppe Italiano** (Rome): *dynamic SP*
Point-to-point shortest path

- worst-case complexity: as bad as SSSP
- many heuristic search approaches proposed that perform well in practice
  - e.g. bidirectional A* search using landmark heuristics
- we will focus on SSSP and APSP
Some crude bounds

Lower bounds:
- *Single-source*: looking at input takes $O(m)$
- *All pairs*: writing output takes $O(n^2)$

Upper bounds:
- *Single-source, non-neg*: naive Dijkstra takes $O(n^2)$
- *Single-source, arbitrary*: BFM takes $O(mn)$
- *All pairs*: $n$ times single-source: $O(n^3) / O(mn^2)$
Overview

- Introduction
- Variants of the problem
- Scanning methods
  - Non-negative weights
  - Arbitrary weights
- Fast matrix multiplication methods
- State of the art: an overview
- Conclusion
The scanning method

Single source shortest path from node $s$: (all pairs: $n$ times)

init: $\forall v \in V(v \neq s): d(v) = \infty, S(v) = \text{unreached}, d(s) = 0, S(s) = \text{labeled}$

main loop: while $\exists$ labeled $v$, pick one and scan it.

$\text{scan}(v):$

for each $(v, w) \in E$ do $\text{edgescan}(v, w)$;

$S(v) = \text{scanned}$

$\text{edgescan}(v, w):$

if $d(w) > d(v) + \ell(v, w)$ then

{$d(w) = d(v) + \ell(v, w);$ $S(w) = \text{labeled}$}
Space is (often) dominated by graph representation

In \( \text{scan}(v) \) we need the edges from \( v \)

Time complexity of scan depends on representation

\( n \times n \) matrix: best for dense graphs

\( \Rightarrow \) sparse graphs: bad time (getting the \( O(1) \) neighbours takes \( O(n) \) time!)

\[ \text{and bad space (storing } O(n) \text{ edges takes } O(n^2) \text{ space)} \]

adjacency list representation: best for sparse graphs

\( \Rightarrow \) dense graphs: same time but space increases by a constant factor

edge list representation: e.g. in Prolog/CHR: \texttt{edge/3}

facts/constraints (good when we have first argument indexing)

many real-life graphs (e.g. maps) are sparse

\( \Rightarrow \) usually adjacency list representation used
Dijkstra - priority queues

- Pick a labeled $v$ with minimum $d(v)$
- Works for non-negative edge weights
- Priority queue: insert, decreaseKey, extractMin
- Total complexity: $O(nI + mD + nE)$
- Naive (linear search) [Dijkstra 1959]
  \[ I = D = O(1), \quad E = O(n) \implies \text{total } O(n^2) \]
- Binary heaps: [Williams 1964]
  \[ I = D = E = O(\log n) \implies \text{total } O(m \log n) \]
- Fibonacci heaps: [Fredman-Tarjan 1987]
  \[ I = D = O(1), \quad E = O(\log n) \implies \text{total } O(m + n \log n) \]
Dijkstra - sorting bottleneck

- Dijkstra returns distances in *sorted order*!
- Sort using Dijkstra: star graph, weights are items to sort
- “Sorting bottleneck”: $O(n \log n)$ is tight bound for sort (in the addition-compare model)
- Thus $O(m + n \log n)$ optimal for (add-comp) Dijkstra-based algorithms
- Monotone heap: minimal key does not decrease (sufficient for Dijkstra’s algorithm)
- In the word RAM model, sorting integers can be done faster than $O(n \log n)$
- Linear Dijkstra $\Leftrightarrow$ linear sorting [Thorup 1996]
Bucket-based (monotone) priority queues (integer weights in \(\{0, \ldots, U\}\), can be made to work for (discretized) reals)

**Buckets** [Dial 1969] \(\Rightarrow O(m + nU)\)

**Multilevel buckets** [Denardo-Fox 1979] \(\Rightarrow O(m + n\frac{\log U}{\log \log U})\)

**MB + Fib. heap** [Ahuja-Mehlhorn-Tarjan 1990] \(\Rightarrow O(m + n\sqrt{\log U})\)

**HOT q.** [Cherkassky-Goldberg-Silverstein 1997] \(\Rightarrow O(m + n(\log U)^{1/3+\epsilon})\)

**HOT q. (improved heap)** [Raman 1997] \(\Rightarrow O(m + n(\log U \log \log U)^{1/4+\epsilon})\)

**Smart queue (MB + caliber heuristic)** [Goldberg 2001]: worst case same as MB; linear expected time (e.g. uniform weight distr.); indep. weights: linear time w.h.p.

**new heap** (carefully balanced buffered tree) [Thorup 2004] \(\Rightarrow O(m + n \log \log U)\)
Overview

- Introduction
- Variants of the problem
- Scanning methods
  - Non-negative weights
  - Arbitrary weights
- Fast matrix multiplication methods
- State of the art: an overview
- Conclusion
Bellman-Ford-Moore algorithm

What if edge weights can also be negative? ⇒ [BFM 1958]

- Pick labeled \( v \)'s in FIFO order.
- One pass consists of processing the nodes labeled in the previous pass
- If there are no negative cycles, BFM terminates in less than \( n \) passes ⇒ total time: \( O(mn) \)
- Can abort after \( n - 1 \) passes and conclude that there is a negative cycle.
- Many heuristics have been proposed to improve practical performance, but \( O(mn) \) is still the best worst-case time bound.
Arbitrary integer weights

Again, in the word RAM model we can do better for integer weights: (weights in \(-U, \ldots, U\))

- \(O(mn^{3/4} \log U)\) [Gabow 1983]
- \(O(m \sqrt{n} \log(nU))\) [Gabow-Tarjan 1989]
- Scaling algorithm [Goldberg 1993]: \(O(m \sqrt{n} \log L)\)
  (weights in \(-L, -L + 1, \ldots\))
  \(\Rightarrow\) Depends only on lower bound!
Overview

- Introduction
- Variants of the problem
- Scanning methods
  - Non-negative weights
  - Arbitrary weights
- Fast matrix multiplication methods
- State of the art: an overview
- Conclusion
Fast matrix multiplication methods

- Algebraic matrix multiplication can be done in sub-cubic time.
- $O(n^{2.81})$ [Strassen 1969]
- ...
- $O(n^{2.38})$ [Coppersmith-Winograd 1990]
- Still major open problem: $\tilde{O}(n^2)$ possible?
- Can this be used to compute distances using repeated squaring? (cfr reachability algorithms)
  $\Rightarrow$ yes! Many algorithms are based on this! (everything with weird exponents)
Fast matrix “min-plus” product

- Problem: normal matrix multiplication is “plus-mult”: $C = AB$ is defined as $c_{ij} = \sum_k a_{ik} b_{kj}$
- We need “min-plus” product: $c_{ij} = \min_k \{a_{ik} + b_{kj}\}$
- Fast matrix multiplication can be adapted to other kinds of products, but it depends crucially on the inverse operation for “+”
- “minimum” operation has no inverse!
- Solution: use polynomials $X^{a_{ij}}$ instead, do normal matrix multiplication, result is lowest exponent
- gives a $\tilde{O}(Nn^{2.38})$ UAPSP algorithm for integer weights
Overview

- Introduction
- Variants of the problem
- Scanning methods
  - Non-negative weights
  - Arbitrary weights
- Fast matrix multiplication methods
- State of the art: an overview
- Conclusion
## Exact Static SP algorithms

<table>
<thead>
<tr>
<th></th>
<th>Single source</th>
<th>All pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>directed</td>
<td>undirected</td>
</tr>
<tr>
<td>unweighted</td>
<td>$O(m)$</td>
<td>folklore (breadth-first scan)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>weights in $\mathbb{N}$ $\ell \in [0, N]$</td>
<td>$O(m + n \log \log \min(n, N))$</td>
<td>Thorup 2004</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>weights in $\mathbb{R}^+$ $\ell \in [L, U]$</td>
<td>$O(m + n \log n)$</td>
<td>Dijkstra 1959, Fredman-Tarjan 1987</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>weights in $\mathbb{Z}$ $\ell \in [-N, N]$ $\ell \in [-L, +\infty[$</td>
<td>$O(m \sqrt{n} \log L)$</td>
<td>Goldberg 1995</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>weights in $\mathbb{R}$</td>
<td>$O(mn)$</td>
<td>Bellman-Ford-Moore 1958</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Approximate Static SP algorithms

<table>
<thead>
<tr>
<th>directed, weights in {0, 1, \ldots, M}</th>
<th>undirected, unweighted</th>
<th>undirected, weighted</th>
</tr>
</thead>
<tbody>
<tr>
<td>stretch</td>
<td>surplus</td>
<td>stretch</td>
</tr>
</tbody>
</table>
| 1 | $\tilde{O}(\sqrt{Mn^{2.80}})$  
Takaoka 1998 | $\tilde{O}(n^{2.38})$  
Seidel 1995  
$O(mn)$  
folklore | $1$  
Dijkstra 1959,  
Fredman-Tarjan 1987 |
| $1 + \epsilon$ | $\tilde{O}(n^{2.38} \log M / \epsilon)$  
Zwick 1998 | $2$  
$\tilde{O}(n \sqrt{mn})$  
$\tilde{O}(n^2 \sqrt[3]{n})$  
Dor-Halperin-Zwick 1996 | $2$  
$\tilde{O}(n \sqrt{mn})$  
Cohen-Zwick 1997 |
| $2(k - 1)$ | $\tilde{O}(n^2 \sqrt[m]{m/n})$  
$\tilde{O}(n^{2-1/(3k-4)})$  
Dor-Halperin-Zwick 1996 | $7/3$  
Cohen-Zwick 1997 | $3$  
$\tilde{O}(n^2)$  
Cohen-Zwick 1997 |

non-negative weights, all pairs shortest path  
surplus: $\text{dist} \leq \text{estimate} \leq \text{dist} + \text{surplus}$  
stretch: $\text{dist} \leq \text{estimate} \leq \text{dist} \times \text{stretch}$
Exact Dynamic SP algorithms

(directed graphs, constant query time)

- Amortized bounds, Single source:
  - Non-negative weights: no better than rebuilding from scratch
    ⇒ [Ramalingam-Reps 1996] efficient in practice
  - Arbitrary weights: update in $O(m + n \log n)$ [Demetrescu 2001]

- Amortized bounds, All pairs:
  - Non-negative weights: $\tilde{O}(n^2)$ update [Demetrescu-Italiano 2003]
  - Arbitrary weights: $\tilde{O}(n^2)$ update [Thorup 2003]

- Worst-case bounds, All pairs:
  - Non-negative weights: $\tilde{O}(n^{2.75})$ update [Thorup 2005]
For example [Roditty-Zwick 2004]:
all pairs, undirected graph, integer weights in \{0, 1, \ldots, M\}:
for every \( \epsilon, \delta > 0 \), and \( t \leq m^{1/2-\delta} \), we have an algorithm with:

- expected amortized update time \( \tilde{O}(Mmn/t) \)
- worst-case query time \( O(Mt) \)
- distance estimates of stretch \( 1 + \epsilon \) (w.h.p.)

(for unweighted graphs, drop the \( M \))
Approximate Distance Oracles

- Storing all $n^2$ distances can be too costly
  $\Rightarrow$ e.g. road map of USA is a sparse graph with about 30 million nodes, storing the graph: a few gigabytes; storing all distances: a few petabytes.

- Solution: *approximate distance oracles*

- Algorithm of [Thorup-Zwick 2001] takes $O(mn^{1/k})$ time to produce a compact data structure ($O(n^{1+1/k})$ space) which can be used to get stretch $2k - 1$ distance estimates in constant time (for weighted undirected graphs)

- Essentially optimal stretch-space tradeoff.
Conclusion

- Non-negative single-source: almost solved in theory and practice
- Big open problem: linear algorithm for directed non-negative SSSP?
- Other variants: many open problems remaining, e.g.
  - $O(n^{2.38})$ algorithm for direct unweighted APSP?
  - $O(n^{3-\epsilon})$ algorithm for APSP with weights in $\{1, \ldots, n\}$?
  - $O(n^{2.5-\epsilon})$ algorithm for SSSP with weights in $\{-n, \ldots, n\}$?
- Theoretical work on P2PSP lags behind.
Questions?

Further reading: