Faster shortest path algorithms, part 1

Jesper Nederlof

10 October 2007

Outline

Some repetition...

$\textsc{Dijkstra}((V, E), w, s, t)$

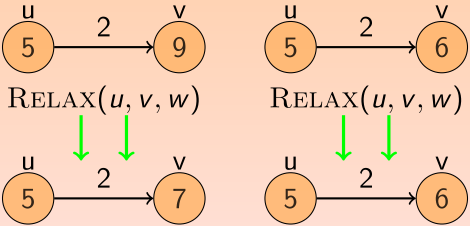
```
for each v ∈ V do d(v) ← ∞
d(s) ← 0
initialize priority queue Q with v ∈ V and priorities d(v)
while priority queue Q is not empty
    u ← remove node with smallest d(u) in Q
    if u = t return
    for each (u, v) ∈ E
        Relax(u, v, w)
```

Relaxation step

$\textsc{Relax}(u, v, w)$

```
if d(v) > d(u) + w(u, v)
then d(v) ← d(u) + w(u, v)
     p(v) ← u
```

$$\begin{aligned}&\textbf{if } d(v) > d(u) + w(u, v)\\&\textbf{then } d(v) \leftarrow d(u) + w(u, v)\\&\qquad p(v) \leftarrow u\end{aligned}$$



(a) RELAX lowers $d(v)$

(b) RELAX doesn't lower $d(v)$

Basic heuristics

- For the single pair shortest path problem, 2 techniques will be presented:
  1. Bidirectional search
  2. Goal-Directed search
- First one can be used for arbitrary graphs, second one only for plane graphs.

Bidirectional search

- After all Dijkstra iterations, for every node u not inside Q, $d(u)$ is the length of the shortest s-u-path.
- At the same time we could execute another Dijkstra on the graph with reversed arcs. Now we have the length of the shortest v-t-path for each node v not in this second priority queue too.
- When a node gets gets outside both priority queues, we know the shortest path (on white board).
- A degree of freedom in this method is the choice whether a forward or backward iteration is executed.
- Simply alternate or choose the one with lower minimum d in the queue are examples of strategies.

Goal-Directed search

- Example with the weight function of an edge its length.
- Assign to each node a potential p, in our case the euclidean distance to t.
- Use an adjusted weight function $w'(u,v) = w(u,v) - p(u) + p(t)$
- w' never gets negative due to the triangle inequality.
- Use Dijkstra with the new weights.
- Gives same shortest path.
- Should terminate faster in most cases.

A special situation

- Central server has to answer a huge number of on-line queries asking for best routes in a very large, plane network.
- Only linear storage is feasible.
- Main goal is to minimize the response time for answering the queries.
- Applications in route planning systems for private transport, public transport or web searching etc.
- Special case of single pair shortest path on a directed graph.
- If edges of negative weight exist, use the algorithm of Johnson to make them positive (this could take a while).
- So we only have positive weights.

Geometric containers: The idea

Observation

An edge that is not the first edge on a shortest path to the target can be safely ignored in any shortest path computation to this target.

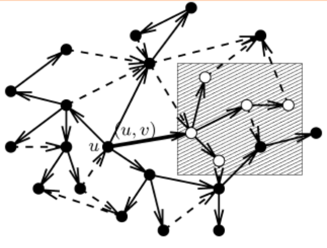
Definition

Let $S(u, v)$ be the set of nodes x for which the shortest u-x-path starts with edge (u, v).

- Now we can adjust Dijkstra's algorithm such that only edges (u, v) with $t \in S(u, v)$ are considered.
- But unfortunately, checking whether a target is in $S(u, v)$ takes $O(\log n)$ (an array based approach takes too much memory).

Containers

- A container $C(u, v)$ is some kind of geometric object which at least has to contain $S(u, v)$
- We will use containers that can be described with constant-sized information and have a constant-time point containment check.



The algorithm: Small adjustment to Dijkstra's algorithm

$\textsc{DijkstraWithPruning}((V,E),w,s,t)$

for each $v \in V$ **set** $d(u) \leftarrow \infty$
$d(s) \leftarrow 0$
initialize priority queue Q with $v \in V$ and priorities $d(v)$
while priority queue Q is not empty
    $u \leftarrow$ node with smallest $d(u)$ in Q
    if $u = t$ **return**
    for each $(u,v) \in E$
        if $t \in C(u,v)$
            $\textsc{Relax}(u,v,w)$

Computing S: Larger adjustment to Dijkstra's algorithm

A(v) stores the first edge in a shortest s-v-path

$\textsc{DijkstraForComputingS}((V,E),w)$

```
for each s ∈ V
    for each v ∈ V set d(u) ← ∞
    d(s) ← 0
    initialize priority queue Q with v ∈ V and priorities d(v)
    while priority queue Q is not empty
        u ← node with smallest d(u) in Q
        if u ≠ s enlarge S(A(v)) to contain u
        for each (u, v) ∈ E
            RELAX2(u, v, w)
```

Relaxation step

$\textsc{Relax2}(u, v, w)$

if $d(v) > d(u) + w(u, v)$
then $d(v) \leftarrow d(u) + w(u, v)$
    $p(v) \leftarrow u$
    if $u = s$
        $A(v) \leftarrow (s, v)$
    else
        $A(v) \leftarrow A(u)$

Computing C

- Because $\sum_{(u,v)\in E} |S(u,v)| \leq n$, all sets S can be stored in memory.
- Multiple kinds of shapes for the containers have been studied extensively.
- The simple bounding box seems to be one of the best.
- For each (u, v), the smallest bounding box C containing all nodes $S(u, v)$ can easily be computed.

Performance bounding box container

Performance bounding box container

- In some experiments, the discussed method seems to be 20 times faster than Dijkstra on average.
- This is based on graphs with 400 to 50000 nodes.

Combining techniques

- Most of the techniques can be combined, for example directional search and geometric containers.
- Store for each edge (u, v) all t nodes for which the shortest u-t-path starts with edge (u, v) as before.
- Also store for each edge (u, v) all s nodes for which the shortest s-v-path ends with edge (u, v).
- With this information, it is possible to update containers when an edge weight is adjusted.

Conclusions

- Several techniques can be used to speedup Dijkstra's algorithm.
- Some can be applied to arbitrary graphs, other ones only in certain scenarios.
- 'Faster shortest path algorithms, part 2' will be about another speed technique for the same situation.

References

- D. Wagner, T. Willhalm, C. Zaroliagis. Geometric containers for efficient shortest-path computation, Journal of Experimental Algorithms 10, 2005, 1.3.
- D. Wagner, Speed-Up Techniques for Shortest-Path Computations, Proceedings STACS 2007.