



Seminarvortrag zum Thema:  
**Verkehr und Logistik**

**Beschleunigung Kürzester-Wege-Algorithmen in der Praxis**

basierend auf der Veröffentlichung

**Speeding Up Dynamic Shortest-Path Algorithms**

von Luciana Buriol, Mauricio Resende & Mikkel Thorup



Was erwartet uns gleich:

- Einleitung
- Technik, erläutert anhand von Ramalingam & Reps
  - Datenstruktur & Parameter
  - Inkrementierung ohne rh
  - Inkrementierung mit rh
  - Dekrementierung und andere Algorithmen
- Fazit
  - Vergleich
  - Resultate



## Kurze Einleitung

- Problem ist bereits bekannt: gesucht ist der kürzeste Weg  
mathematisch berechenbar
- Erweiterung des Problems: dynamische Graphen
- bereits viele Lösungen, weiter verbesserbar



- Speeding up dynamic shortest-path algorithms von Buriol, Resende & Thorup
- Beschleunigung von Algorithmen durch “reduce heap”
- in erster Linie Reduzierung von Speicher dadurch Reduzierung der Laufzeit
  
- als Ausblick: Beschleunigung mit der Hilfe der reduce heap Technik bis zum Faktor 5



## Die reduce heap Technik - Ausgangslage

- als Ausgangslage: Ausgangsgraph und Subgraph
- “Zielgraph” beinhaltet alle kürzesten Wege zu einem Ziel
- Unterschied zwischen Graph und Baum: alternative Routen
- Präsentiert wird erst der Originalalgorithmus, dann erst die Technik



## Algorithmus nach Ramalingam & Reps - Idee

- allgemeine Idee: Algorithmus nach Dijkstra  
jedoch nach Eingrenzung der zu betrachteten Knoten
- somit “Hoffnung” auf weniger Rechenaufwand
  
- Algorithmus unterscheidet zwischen Inc und Dec



## Algorithmus nach Ramalingam & Reps - Datenstruktur

Ausgangsgraph wird dargestellt durch

- $fw$  - array mit (tail-Knoten, head-Knoten)
- $p$  - array mit Positionen für  $fw$
- $w$  - array mit Kantenlängen

Zielgraph wird dargestellt durch

- $d$  - array mit Abstand von Knoten zum Zielknoten
- $gSP$  - array mit 0/1, Kante  $\in$  Zielgraphen
- $\delta$  - array mit Anzahl ausgehender Kanten



## Algorithmus nach Ramalingam & Reps - Parameter

procedure RR+ ( $a$ ,  $w$ ,  $d$ ,  $\delta$ , gSP)

- $a$  - inkrementierte Kante
- für alle Parameter gilt: call-by-reference  
um Speicher zu sparen
  
- Aufruf des Algorithmus nur beim dynamischen Verändern  
des Ausgangsgraphen



## Algorithmus nach Ramalingam & Reps

- Überprüfen, ob Kante  $a$  Element vom Zielgraphen ist  
falls `false`, dann sind weitere Berechnungen nicht nötig
- Überprüfen, ob  $u$  weitere ausgehende Kanten hat  
falls `true`, dann sind weitere Berechnungen nicht nötig

```
1  if  $gSP_a = 0$  return;  
2   $gSP_a = 0$ ;  
3   $\delta_u = \delta_u - 1$ ;  
4  if  $\delta_u > 0$  then return;
```

- sonst...



- Überprüfen, ob Knoten vor  $\text{tail}(a) = u$  einen anderen Weg haben

```
1  Q = {u}
2  for u ∈ Q do
3    du = ∞;
4    for e = (s, u) ∈ IN(u) do
5      if gSPe = 1 then
6        gSPe = 0;
7        δs = δs - 1;
8        if δs = 0 then
9          Q = Q ∪ {s};
10       end if
11     end for
12 end for
```



- Überprüfen, ob ein Weg außerhalb von  $Q$  kürzer ist

```
1 for u ∈ Q do
2   for e = (u, v) ∈ OUT(u) do
3     if  $d_u > d_v + w_e$  then
4        $d_u = d_v + w_e$  ;
5     end for
6   if  $d_u \neq \infty$  then
7     InsertIntoHeap(H , u,  $d_u$  );
8 end for
```

```
... // dijkstra(H);
```

- Ausführung des Dijkstra bezogen auf dem heap



## Algorithmus nach Ramalingam & Reps

- Zusammenfassend ist zu sagen, dass die Idee von Ramalingam & Reps darin besteht, den Algorithmus nach Dijkstra zu beschleunigen, indem sie die Menge der zu betrachtenden Knoten einschränken.
- $O(m + n \log(n))$ , wobei  $m, n$  nur eine Teilmenge darstellen



## Algorithmus nach Ramalingam & Reps mit rh (reduce heap)

- Grundidee: die Menge  $Q$  ist weiter einschränkbar durch weitere Berechnung und Betrachtung der Inkrementierung
- Genau untersuchen, ob Kante wirklich betroffen ist



## Algorithmus nach Ramalingam & Reps mit rh - Parameter

procedure rhRR+ (a, w, d ,  $\delta$ , gSP, inc)

- Parameterliste gleich, nur erweitert durch inc
- inc - Größe der Inkrementierung



## • Unterschied: Berücksichtigung von inc

```
1  if gSPa = 0 return;
2  gSPa = 0;
3   $\delta u = \delta u - 1$ ;
4  if  $\delta u > 0$  then return;
5  Q = {u}
6  for u  $\in$  Q do
7    du =  $\infty$ ;
8    for e = (s, u)  $\in$  IN(u) do
9      if gSPe = 1 then
10         gSPe = 0;
11          $\delta s = \delta s - 1$ ;
12         if  $\delta s = 0$  then
13           Q = Q  $\cup$  {s};
14         end if
15       end for
16     end for
```

```
1  if gSPa = 0 return;
2  gSPa = 0;
3   $\delta u = \delta u - 1$ ;
4  if  $\delta u > 0$  then return;
5  Q = {u}
6  for u  $\in$  Q do
7    du = du + inc;
8    for e = (s, u)  $\in$  IN(u) do
9      if gSPe = 1 then
10         gSPe = 0;
11          $\delta s = \delta s - 1$ ;
12         if  $\delta s = 0$  then
13           Q = Q  $\cup$  {s};
14         end if
15       end for
16     end for
```



- Falls  $\text{inc} = I$  ist, so ist der Algorithmus am Ende  
sonst...
- Prozedur für  $\text{tail}(a) = k$  die selbe, Rest wird gesondert betrachtet
  - 1  $\text{dist} = d_k$  ;
  - 2 for  $e = (k, v) \in \text{OUT}(k)$  do
  - 3 if  $d_k > d_v + w_e$  then
  - 4  $d_k = d_v + w_e$  ;
  - 5 end for



- Berechnen der relevante Kanten und hinzufügen im heap

```
1  diff = dist - dk ;
2  for u ∈ Q \ k do
3    du = du - diff;
4    flag = 0;
5    for e = (u, v) ∈ OUT(u) do
6      if du > dv + we then
7        du = dv + we ;
8        flag = 1;
9      end if
10   end for
11   if flag = 1 then InsertIntoHeap(H , u, du);
12 end for

... .. // dijkstra(H);
```



## Weitere Algorithmen

- reduce heap Technik auch auf Dekrementierung übertragbar
- auf weitere Algorithmen übertragbar, in der Veröffentlichung:

Ramalingam & Reps - Graph

Ramalingam & Reps - Tree

King & Thorup - Tree

Demetrescu - Tree (nur inc)

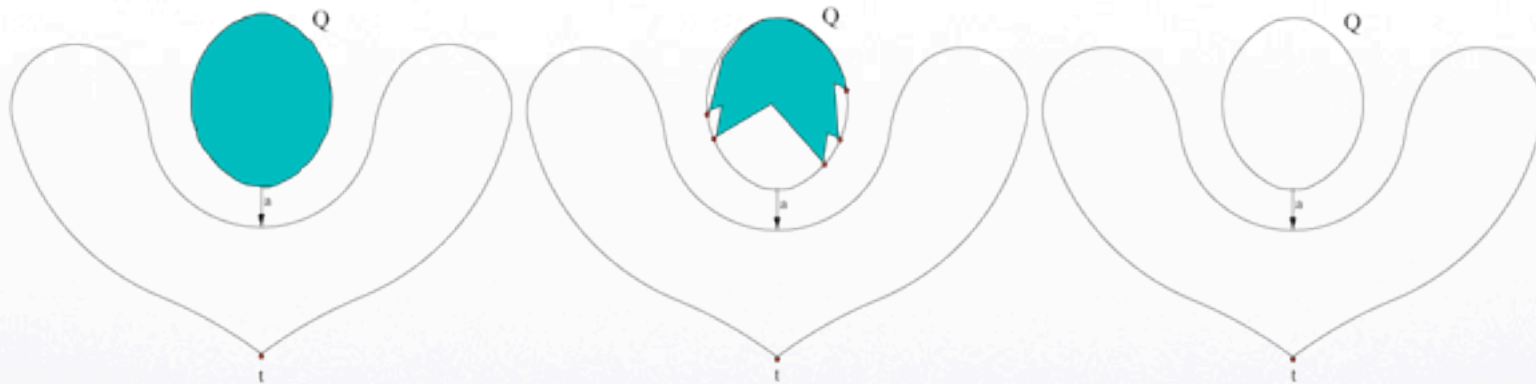


## Fazit - Vergleich

- Idee war ja durch Vorberechnung die Menge  $Q$  einzuschränken
- Technik erlaubt eine noch weitere Eingrenzung dieser Menge durch genauere Voruntersuchung



## Fazit - graphischer Vergleich





## Fazit - Resultate

- Tests ergeben:  $O(m + n \log(n))$ , mit reduziertem  $n$
- Komplexitätsklasse bleibt, weil Algorithmus nach Dijkstra weiter verwendet wird
- allerdings: durch reduce heap Technik, ergibt sich:  
 $O(k [m + n \log(n)])$ , bei  $0 < k < 1$
- bei allen untersuchten Algorithmen:  
Beschleunigung bis zum 5-fachem



## Fazit

- In der Regel egal, welchen Algorithmus man nimmt  
jeder Algorithmus hat Vor- und Nachteile
- Einsatzgebiet entscheidend: z.B.  
Ist der Graph dynamisch?  
Handelt es sich nur um Inkrementierungen?  
In welcher Größenordnung liegen  $m$  und  $n$ ?  
...
- Man sollte aber nicht den Algorithmus nach Dijkstra wählen, denn  
Verbesserungen bis zu 149.371,17-fach\* sind möglich



Platz für Fragen

...?!

Wird die Technik irgendwo eingesetzt?



Seminarvortrag zum Thema:  
**Verkehr und Logistik**

**Beschleunigung Kürzester-Wege-Algorithmen in der Praxis**

basierend auf der Veröffentlichung

**Speeding Up Dynamic Shortest-Path Algorithms**

von Luciana Buriol, Mauricio Resende & Mikkel Thorup