

# **Polunhakualgoritmit ja -järjestelmät**

Rodion Efremov

Kandidaatintutkielma-aine  
HELSINGIN YLIOPISTO  
Tietojenkäsittelytieteen laitos

Helsinki, 11. lokakuuta 2014

Tiedekunta — Fakultet — Faculty		Laitos — Institution — Department	
Matemaattis-luonnontieteellinen		Tietojenkäsittelytieteen laitos	
Tekijä — Författare — Author			
Rodion Efremov			
Työn nimi — Arbetets titel — Title			
Polunhakualgoritmit ja -järjestelmät			
Oppiaine — Läroämne — Subject			
Tietojenkäsittelytiede			
Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages	
Kandidaatintutkielma-aine	11. lokakuuta 2014	7	
Tiivistelmä — Referat — Abstract			
Tiivistelmä.			
Avainsanat — Nyckelord — Keywords			
a, bb, ccc			
Säilytyspaikka — Förvaringsställe — Where deposited			
Muita tietoja — Övriga uppgifter — Additional information			

## Sisältö

<b>1</b>	<b>Johdanto</b>	<b>1</b>
<b>2</b>	<b>Tavallisimmat algoritmit</b>	<b>1</b>
<b>3</b>	<b>Kaksisuuntainen haku</b>	<b>3</b>
3.1	Kaksisuuntainen Dijkstran algoritmi . . . . .	3
3.2	Kaksisuuntainen $A^*$ . . . . .	6
<b>4</b>	<b>Prioriteettijonon valinta</b>	<b>6</b>
<b>5</b>	<b>Kaikkien parien lyhimmät polut</b>	<b>7</b>
	<b>Lähteet</b>	<b>7</b>

## 1 Johdanto

Polunhaku painotetuissa tai painottamattomissa verkoissa on perustavanlaatuisen ongelma, joka ei ole mielenkiintoinen vain itsessään, vaan on toisinaan tarvittava alioperaatio muissa algoritmeissa. Esimerkiksi Edmond-Karpin algoritmi käyttää leveyssuuntaisen haun ratkaistaessaan maksimivuo-ongelmaa; multiple sequence alignment -ongelmaa on ruvettu viime vuosikymmeninä ratkomaan myös heuristisin polunhakualgoritmein.

Verkoista puhuttaessa verkko  $G$  on kaksikko  $(V, A)$ , jossa  $V$  on solmujen joukko, ja  $A \subset V \times V$  on (suunnattujen) kaarien joukko. Suuntaamaton verkko  $G' = (V, E)$  voidaan aina simuloida suunnatulla verkolla  $G = (V, A)$  siten, että jokaista suuntaamatonta kaarta  $\{u, v\} \in E$  kohti laitetaan  $A$ :han kaaret  $(u, v)$  ja  $(v, u)$ . (Suunnattu verkko on suuntaamattoman yleistys.) Polunhakua varten, verkosta erotellaan kaksi solmua: lähtösolmu  $s$  ja maalisolmu  $t$ . Jatkossa,  $n = |V|$  ja  $m = |E|$ ; näin esimerkiksi leveyssuuntaisen haun aikavaativuus on  $O(n + m)$ . Polku on  $\gamma_k = \langle u_0, u_1, \dots, u_k \rangle$ , missä mikään solmu ei esiinny yhtä kertaa enempää, ja verkossa on kaari  $(u_i, u_{i+1})$  jokaisella  $i = 0, 1, \dots, k - 1$ . Polkuun liittyvä kustannus on sen kaarien painojen summa, ja mitä tulee itse painoihin, ne oletetaan olevan ei-negatiivisia. Eipainotettujen verkkojen kohdalla, jokaisen kaaren paino oletetaan olevan 1.

## 2 Tavallisimmat algoritmit

Edsger W. Dijkstra esitti vuonna 1959 kuuluisan polunhakualgoritminsa, joka käy polynomisessa ajassa [1]. Algoritmi voidaan pitää yhdistävän ”ahneuden” (engl. *greedy algorithm*), dynaamisen ohjelmoinnin ja inkrementaalisen lähestymistavan. Saatuaan lähtösolmun  $s$ , algoritmi laskee lyhimpien polkujen puun lähtien solmusta  $s$  kunnes  $t$  joutuu *avoimeen listaan* (engl. *open list; search frontier*), ja sitä kautta *suljettuun listaan* (engl. *closed list; settled node list*), jolloin lyhin  $s, t$ -polku on löytynyt. Hart et al. esittivät vuonna 1968 kuuluisan  $A^*$ -algoritminsa, joka – samoin kuten Dijkstran algoritmi – ylläpitää mm. kunkin saavutetun solmun  $u$   $g$ -arvon  $g(u)$ , joka on toistaiseksi pienin kustannus lähtösolmusta  $s$  solmuun  $u$ , ja joka on taattu olemaan pienin mahdollinen heti kun  $u$  poistuu avoimesta listasta [2]. Erona on kuitenkin se, että  $A^*$  käyttää kunkin solmun  $u$  prioriteettinä sen  $f$ -arvo, joka on siis  $f(u) = g(u) + h(u)$ , jossa  $h(u)$  on solmun  $u$  optimistinen (eli aliarvioitu) etäisyys maalisolmuun. Intuitio tämän järjestyksen takana on se, että  $A^*$  ”tietää” mihin suuntaan haku on suunnattava, jota pääsisi maalisolmuun, ainakin paremmin kuin Dijkstran algoritmi, jonka hakuavaruus kasvaa laajenevan pallon tavoin ”kaikkiin suuntiin”.

---

**Algoritmi 1:** DIJKSTRA-SHORTEST-PATH( $G, s, t, w$ )

---

Monikkosijoitus

```
1 OPEN, CLOSED,  $g, \pi = (\{s\}, \emptyset, \{(s, 0)\}, \{(s, \mathbf{nil})\})$ 
2 while  $|OPEN| > 0$  do
3    $u = \arg \min_{x \in OPEN} g(x)$ 
4   OPEN = OPEN -  $\{u\}$ 
5   if  $x$  is  $t$  then
6     return TRACEBACK-PATH( $t, \pi, \mathbf{nil}$ )
7   CLOSED = CLOSED  $\cup \{x\}$ 
   Jokaisella solmun  $x$  lapsisolmulla  $u$ , tee...
8   for  $(x, u) \in G.A$  do
9     if  $u \in CLOSED$  then
10      continue
11      $g' = g(x) + w(x, u)$ 
12     if  $u \notin OPEN$  then
13       OPEN = OPEN  $\cup \{u\}$ 
14        $g(u) = g'$ 
15        $\pi(u) = x$ 
16     else if  $g(u) > g'$  then
17        $g(u) = g'$ 
18        $\pi(u) = x$ 

   Ei  $s, t$  -polkua verkossa  $G$ .
19 return  $\langle \rangle$ 
```

---

$A^*$ :n pseudokoodi on tasan sama kuin Dijkstran algoritmin, paitsi että rivillä 3  $g(x)$ :n sijasta on  $f(x)$ , jolle siis  $f(x) = g(x) + h(x)$ . Molemmat kutsuvat TRACEBACK-PATH-rutiinia, joka siis muodostaa lyhimmän polun “edeltäjäpuusta” (engl. *predecessor tree*) ajassa  $\Theta(N)$ , missä  $N$  on lyhimmän polun sol-

---

**Algoritmi 2:** TRACEBACK-PATH( $x, \pi, \pi_{REV}$ )

---

```
1  $u = x$ 
2  $p = \langle \rangle$ 
3 while  $u$  is not nil do
4   lisää  $u$   $p$ :n alkuun
5    $u = \pi(u)$ 
6   Kaksisuuntainen haku?
7   if  $\pi_{REV}$  is not nil then
8     while  $u$  is not nil do
9       lisää  $u$   $p$ :n loppuun
10       $u = \pi_{REV}(u)$ 
11 return  $p$ 
```

---

mujen määrä.

### 3 Kaksisuuntainen haku

Vaikka  $A^*$  on tyypillisesti tehokkaampi kuin Dijkstran algoritmi, käyttämällä *kaksisuuntaista* hakua, voidaan päästää verrattavissa olevaan suorituskyykyyn. Ajatus kaksisuuntaisuuden takana on se, että algoritmi kasvattaa kaksi hakupuuta, yksi normaaliin tapaan ja toinen maalisolmusta ihan kuin kaaret olisivat “käännetty” päinvastaiseen suuntaan, kunnes kaksi hakuavaruutta “kohtaavat” keskellä. Nyt jos lyhin polku koostuu  $N$  kaaresta, ja verkon solmujen keskiarvoinen aste on  $d$ , tavallinen, eli yksisuuntainen haku tekee työn

$$\sum_{i=0}^N d^i,$$

kun kaksisuuntainen olisi tehnyt vain

$$2 \sum_{i=0}^{\lceil N/2 \rceil} d^i.$$

Ylläoleva pätee leveyssuuntaiseen hakuun sellaisenaan, ja painotetun haun kohdalla voidaan saada yläraja kertomalla kunkin summan termin tekijällä  $O(\log n)$ .

#### 3.1 Kaksisuuntainen Dijkstran algoritmi

Ylläolevan analyysin nojalla, on selvä, että Dijkstran algoritmi hyötyy kaksisuuntaisuudesta, eikä edellytä minkäänlaista verkon esiprosessointia. Lisäksi, algoritmin vahvuutena suhteessa  $A^*$ :iin ei ole pelkästään verrattavissa oleva tehokkuus, vaan myös heuristiikkafunktion tarpeettomuus. Alla  $\mu$  on toistaiseksi lyhimmän polun kustannus, joka suorituksen alussa on  $\infty$ . Kun algoritmi

löytää toistaiseksi lyhimmän polun hakuavaruuksien kohdatessa, “välisolmu”  $m$  ja sen implikoiva kustannus  $\mu$  päivitetään. Haku jatkuu siihen asti, kunnes molempien avointen listojen minimialkioiden kustannusten summa on vähin-

---

**Algoritmi 3:** EXPAND( $OPEN, CLOSED, CLOSED_2, g, g_2, \pi, \mu, m, e, w$ )

---

```

1  $u = \arg \min_{x \in OPEN} g(x)$ 
2  $OPEN = OPEN - \{u\}$ 
3  $CLOSED = CLOSED \cup \{u\}$ 
4 for  $x \in e(u)$  do
5   if  $x \in CLOSED$  then
6     continue
7    $g' = g(u)$ 
8   if  $e(u)$  gives child nodes of  $u$  then
9     “Normaali” haku.
9      $g' = g' + w(u, x)$ 
tään  $\mu$ . 10 else
11   Käännetty haku.
11    $g' = g' + w(x, u)$ 
12 if  $x \in OPEN$  then
13    $OPEN = OPEN \cup \{x\}$ 
14    $g(x) = g'$ 
15    $\pi = x$ 
16   UPDATE( $x, CLOSED_2, g', g_2, \mu, m$ )
17 else if  $g(x) > g'$  then
18    $g(x) = g'$ 
19    $\pi = x$ 
20   UPDATE( $x, CLOSED_2, g', g_2, \mu, m$ )

```

---



---

**Algoritmi 4:** UPDATE( $x, CLOSED, g', g, \mu, m$ )

---

```

1 if  $x \in CLOSED$  then
2    $p = g' + g(x)$ 
3   if  $\mu > p$  then
4      $\mu = p$ 
5      $m = x$ 

```

---

Rutiini UPDATE tarkistaa, että yhden hakusuunnan solmu on toisen sulje-

tussa listassa, ja jos asia on niin, yrittää päivittää välisolmun.

---

**Algoritmi 5: BIDIRECTIONAL-DIJKS**

---

```

1 OPEN, CLOSED,  $g, \pi = \{s\}, \emptyset, \{(s, C$ 
2 OPENREV, CLOSEDREV,  $g_{REV}, \pi_{REV}$ 
3  $\mu = \infty$ 
4  $m = \text{nil}$ 
5 while  $|OPEN| \cdot |OPEN_{REV}| > 0$  do
6   if  $m$  is not nil then
7      $p = \text{TERMINATE}(\text{OPEN}, \text{OPEN}_{REV},$ 
8        $g, g_{REV}, \pi, \pi_{REV}, \mu, m)$ 
9     if  $p$  is not nil then
10      return  $p$ 
11   Triviaali kuormantaus
12   if  $|OPEN| < |OPEN_{REV}|$  then
13     EXPAND( $\text{OPEN},$ 
14        $\text{CLOSED},$ 
15        $\text{CLOSED}_{REV},$ 
16        $g, g_{REV}, \pi, \mu, m, e_f,$ 
17     )
18   else
19     EXPAND( $\text{OPEN}_{REV},$ 
20        $\text{CLOSED}_{REV},$ 
21        $\text{CLOSED},$ 
22        $g_{REV}, g, \pi_{REV}, \mu, m,$ 
23   )
24 return  $\langle \rangle$ 

```

---

Yllä,  $e_f$  on kuvaus, jolle  $e_f(u) = \{v \in G.V : (u, v) \in G.A\}$  jokaisella  $u \in G.V$ , ja  $e_b(u) = \{v \in G.V : (v, u) \in G.A\}$ . Molemmat siis määrittelevät “laajentumisoperaattorit” (engl. *expansion operator*):  $e_f$  normaalissa haussa, ja  $e_b$

käännettyssä haussa.

---

**Algoritmi 6: TERMINATE( $\text{OPEN}, \text{OPEN}_{REV}, g, g_{REV}, \pi, \pi_{REV}, \mu, m$ )**

---

```

1  $x_f = \arg \min_{x \in \text{OPEN}} g(x)$ 
2  $x_b = \arg \min_{x \in \text{OPEN}_{REV}} g_{REV}(x)$ 
3 if  $g(x_f) + g_{REV}(x_b) \geq \mu$  then
4   return  $\text{TRACEBACK-PATH}(m, \pi, \pi_{REV})$ 
5 return  $\text{nil}$ 

```

---



### 3.2 Kaksisuuntainen $A^*$

Kaksisuuntaisen  $A^*$ :n saa aikaan muuttamalla algoritmin 3 rivi 1 seuraavanlaiseksi:

$$u = \arg \min_{x \in \text{OPEN}} f(x),$$

ja korvaamalla rivin 7 kutsu kutsulla  $\text{TERMINATE} * (\dots)$ , jonka määritelmä on alla.

---

**Algoritmi 7:**  $\text{TERMINATE}^*(\text{OPEN}, \text{OPEN}_{REV}, f, f_{REV}, \pi, \pi_{REV}, \mu, m)$

---

```

1  $x_f = \arg \min_{x \in \text{OPEN}} f(x)$ 
2  $x_b = \arg \min_{x \in \text{OPEN}_{REV}} f_{REV}(x)$ 
3 if  $\min(f(x_f), f_{REV}(x_b)) \leq \mu$  then
4   return  $\text{TRACEBACK-PATH}(m, \pi, \pi_{REV})$ 
5 return nil
```

---

## 4 Prioriteettijonon valinta

Polkua hakiessa painotetuissa verkoissa joudutaan käyttämään prioriteettijonoja, jotka ovat tarpeellisia pitääkseen haut optimaaleina, ja joiden oletetaan tarjoavan ainakin neljä operaatiota:

1.  $\text{INSERT}(H, x, k)$  tallettaakseen solmun  $x$  sen prioriteetin  $k$  kera,
2.  $\text{DECREASE-KEY}(H, x, k)$  päivittääkseen solmun  $x$  talletetun prioriteetin (pienemmäksi),
3.  $\text{EXTRACT-MINIMUM}(H)$  poistaakseen pienimmän prioriteetin omaava solmu, ja
4.  $\text{IS-EMPTY}(H)$  varmistaakseen, että jonossa on vielä alkioita.

Helpoin tehokkaaksi kutsuttu prioriteettijonorakenne (jatkossa vain “keko”) on binäärikeko, jonka operaatiot (1) - (3) käyvät ajassa  $O(\log n)$ , jolloin tällaisella keolla Dijkstran ja  $A^*$ -algoritmit käyvät kumpikin ajassa  $O((m + n) \log n)$ . Teoriassa edelläoleva ylläraja voidaan parantaa käyttämällä Fibonacci-kekoa, jonka lisäysoperaatio (1) käy eksaktissa vakioajassa, päivitysoperaatio (2) tasoitettussa vakioajassa, ja poisto-operaatio (3) tasoitettussa ajassa  $O(\log n)$ , jolloin haut voidaan suorittaa ajassa  $O(m + n \log n)$ . Huomaa, että kaikki tähän asti mainitut keot perustuvat vertailuihin, ja teoriassa enintään yksi operaatiosta  $\text{INSERT}$  tai  $\text{EXTRACT-MINIMUM}$  voi käydä (eksaktissa tai tasoitettussa) vakioajassa, ja toisen on käytävä ajassa  $\Omega(\log n)$ ,

---

**Algoritmi 8:** GENERIC-HEAP-SORT( $S, H$ )

---

```
1  $H = \emptyset$  : tyhjennä keko.  
2 for  $i = 1$  to  $|S|$  do  
     $S[i]$  on itsensä prioriteetti.  
3    $\text{INSERT}(H, S[i], S[i])$   
4 for  $i = 1$  to  $|S|$  do  
5    $S[i] = \text{EXTRACT-MINIMUM}(H)$ 
```

---

koska muuten algoritmi 8 tällaisella keolla rikkoisi lajittelemisen informaatio-teoreettisen rajan, joka on  $\Omega(n \log n)$ . Jos kuitenkin kaarien painot ovat kokonaislukuja,  $O(m + n \log n)$ -rajaa voidaan parantaa: Mikkel Thorup esitti vuonna 2003 keon, jonka poisto-operaatio käy ajassa  $O(\log \log \min n)$  ja muut operaatiot vakioajassa [3]. Jos kuitenkin kokonaislukupainot ovat väliltä  $[0, N]$ , poisto-operaatio voidaan suorittaa ajassa  $O(\log \log \min\{n, N\})$ . Nyt selvästi haun aikavaativuus tällaisella keolla on  $O(m + n \log \log \min\{n, N\})$ .

## 5 Kaikkien parien lyhimät polut

### Lähteet

- [1] Dijkstra, Edsger W.: *A note on two problems in connexion with graphs*. Numerische Mathematik, 1:269–271, 1959.
- [2] Hart, Peter E., Nilsson, Nils J. ja Raphael, Bertram: *A formal basis for the heuristic determination of minimum cost paths*. IEEE Transactions on Systems, Science, and Cybernetics, SSC-4(2):100–107, 1968.
- [3] Thorup, Mikkel: *Integer Priority Queues with Decrease Key in Constant Time and the Single Source Shortest Paths Problem*. Teoksessa *Proceedings of the Thirty-fifth Annual ACM Symposium on Theory of Computing*, STOC '03, sivut 149–158, New York, NY, USA, 2003. ACM, ISBN 1-58113-674-9. <http://doi.acm.org/10.1145/780542.780566>.