

LYHIMPIEN POLKUJEN HAKUALGORITMIT JA -JÄRJESTELMÄT

Rodion Efremov

Tietojenkäsittelytieteen laitos, Helsingin yliopisto

- Suunnattu verkko on $G = (V, A)$, missä V on solmujen joukko ja $A \subset V \times V$ on suunnattujen kaarien joukko.
- Suuntaamaton verkko $G = (V, E)$ voidaan aina simuloida suunnatulla verkolla (V, A) laittamalla A :han kaaret (u, v) ja (v, u) jokaisella suuntaamattomalla kaarella $\{u, v\} \in E$.
- Jatkossa merkitsemme $n = |V|$ ja $m = |E|$.

Verkot

- k :n kaaren polku on $\gamma = \langle u_0, u_1, \dots, u_k \rangle$, missä kukin solmu esiintyy vain kerran ja jokaisella $i \in \{0, 1, \dots, k-1\}$ $(u_i, u_{i+1}) \in A$.
- Painotettujen verkkojen kohdalla otaksumme kunkin kaaren (u, v) painon $w(u, v)$ olevan **ei-negatiivinen**.
- Polun γ paino on $\sum_{i=0}^{k-1} w(u_i, u_{i+1})$.
- Suuntamattomassa verkossa solmun u "aste" on $d(u) = |\{\{u, v\} : \{u, v\} \in E\}|$.
- Suunnatussa verkossa solmun u "sisäänaste" (engl. *indegree*) on $\deg^-(u) = |\{(v, u) : (v, u) \in A\}|$ ja "ulosaste" (engl. *outdegree*) on $\deg^+(u) = |\{(u, v) : (u, v) \in A\}|$.

Leveyssuuntainen haku

- Löytää lyhimmän polun (yhden monesta mahdollisesta) painottamattomassa verkossa.
- Toteutus vaatii vain jonon ja hajautustaulun.
- Toimii ajassa $\mathcal{O}(n + m) \approx \sum_{i=0}^N d^i$, missä N on lyhimmän polun solmujen määrä ja d keskiarvoinen solmun aste tai solmun ulosaste verkon tyypistä riippuen.

Algoritmi 2: BREADTH-FIRST-SEARCH(G, s, t)

```
1  $Q = \langle s \rangle$ 
2  $\pi(s) = \text{nil}$ 
3 while  $|Q| > 0$  do
4    $u = \text{DEQUEUE}(Q)$ 
5   if  $u$  is  $t$  then
6     return  $\text{TRACEBACK-PATH}(u, \pi, \text{nil})$ 
7   for  $(u, v) \in G.A$  do
8     if  $v$  is not yet mapped in  $\pi$  then
9        $\pi(v) = u$ 
10      ENQUEUE( $Q, v$ )
11 return  $\langle \rangle$ 
```

Voiko leveyssuuntaisen haun nopeuttaa?

Voi!

Kaksisuuntainen leveyssuuntainen haku

- Aja kaksi hakuavaruutta: yksi normaalin tapaan lähtösolmusta, ja toinen ”takaperin” maalisolmusta.
- Kun kaksi yllä mainittua hakuavaruutta kohtavat jossain ”keskellä”, rakennetaan lyhin polku.
- Aikavaativuus on $2 \sum_{i=0}^{\lceil N/2 \rceil} d^i$.
- Verkosta riippuen voi olla jopa ~ 1000 kertaa nopeampi kuin yksisuuntainen BFS.

Algoritmi 3: BIDIRECTIONAL-BREADTH-FIRST-SEARCH(G, s, t)

```
1  $Q, \pi, d = (\langle s \rangle, (s, \text{nil}), (s, 0))$ 
2  $Q_{REV}, \pi_{REV}, d_{REV} = (\langle t \rangle, (t, \text{nil}), (t, 0))$ 
3  $\tau, \mu = (\text{nil}, \infty)$ 
4 while  $|Q||Q_{REV}| > 0$  do
5   if  $\tau$  is not nil and  $d(\text{HEAD}(Q)) + d_{REV}(\text{HEAD}(Q_{REV})) \geq \mu$  then
6     return TRACEBACK-PATH( $\tau, \pi, \pi_{REV}$ )
7    $u = \text{DEQUEUE}(Q)$ 
8   if  $u$  is mapped in  $\pi_{REV}$  and  $\mu > d(u) + d_{REV}(u)$  then
9      $\mu = d(u) + d_{REV}(u)$ 
10     $\tau = u$ 
11   for  $(u, v) \in G.A$  do
12     if  $v$  is not yet mapped in  $\pi$  then
13        $\pi(v) = u$ 
14       ENQUEUE( $Q, v$ )
15        $d(v) = d(u) + 1$ 
16    $u = \text{DEQUEUE}(Q_{REV})$ 
17   if  $u$  is mapped in  $\pi$  and  $\mu > d(u) + d_{REV}(u)$  then
18      $\mu = d(u) + d_{REV}(u)$ 
19      $\tau = u$ 
20   for  $(v, u) \in G.A$  do
21     if  $v$  is not yet mapped in  $\pi_{REV}$  then
22        $\pi_{REV}(v) = u$ 
23       ENQUEUE( $Q_{REV}, v$ )
24        $d_{REV}(v) = d_{REV}(u) + 1$ 
25 return  $\langle \rangle$ 
```

Miten rakentaa polut lyhimpien polkujen puusta?

Tarvitaan vain kuvaus π (ja myös π_{REV} mikäli polku oli haettu kaksisuuntaisella haulla).

Algoritmi 1: TRACEBACK-PATH(x, π, π_{REV})

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

Dijkstran algoritmi

- Vuonna 1959 Edsger W. Dijkstra esitti kuuluisan polunhakualgoritminsa, joka toimii polynomisessa ajassa.
- FIFO jonon sijasta prioriteettijono; kutsutaan usein ”avoimeksi” listaksi (engl. *open set*).
- Hajautustauluun perustuva joukkorakenne; kutsutaan usein ”suljetuksi” listaksi (engl. *closed set*).
- g -kuvaus, joka kuvaa kunkin saavutetun solmun toistaiseksi pienimpään etäisyyteen lähtösolmusta laskettuna.
- π -kuvaus, aivan kuten BFS:ssä (kuvaa solmun edeltäjänsä lyhimpien polkujen puussa).
- Kun solmu poistetaan avoimesta listasta, sen g -arvo on optimaali.

Algoritmi 4: DIJKSTRA-SHORTEST-PATH(G, s, t, w)

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

Dijkstran algoritmi

- Dijkstran algoritmi voidaan mieltää BFS:n yleistykseksi painotetuissa verkoissa: samoin kuten BFS, Dijkstran algoritmi kasvattaa hakuavaruutensa "kaikkiin suuntiin" laajenevan pallon tavoin.

A* - haku

- Pseudokoodi tasan sama kuten Dijkstran algoritmilla, paitsi että rivillä 3 $g(x)$ on korvattava $f(x)$:llä, jolle siis $f(x) = g(x) + h(x)$, missä $h(x)$ on optimistinen (eli aliarvioitu) kustannus solmusta x maalisolmuun.
- Intuitio järjestelyn takana on se, että A* "tietää" mihin suuntaan kannattaa lähteä kasvattamaan hakuavaruuden päästääkseen maalisolmuun nopeammin.
- Määrittämällä $h(u) = 0$ kaikilla $u \in V$, A* palautuu Dijkstran algoritmiksi.

Kaksisuuntainen painotettu haku

- Myös A^* ja Dijkstran algoritmit voidaan kaksisuuntaista.
- Jos heuristiikkafunktio ei voida määritellä, kaksisuuntainen Dijkstran algoritmi on melkein aina parempit vaihtoehto suhteessa yksisuuntaiseen versioonsa.
- Mitä tulee A^* :n kaksisuuntaistamiseen, algoritmi ei nopeudu erityisen paljon, sillä jo ei niin hyvä heuristiikkafunktio karsii hakuavaruuden melko hyvin.

Kaksisuuntainen Dijkstran algoritmi

Algoritmi 4: $\text{UPDATE}(x, \text{CLOSED}, g', g, \mu, m)$

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

Kaksisuuntainen Dijkstran algoritmi

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.
10     $g' = g' + w(u, x)$ 
11  else
12    Käännetty haku.
13     $g' = g' + w(x, u)$ 
14  if  $x \notin OPEN$  then
15     $OPEN = OPEN \cup \{x\}$ 
16     $g(x) = g'$ 
17     $\pi(x) = u$ 
18    UPDATE( $x, CLOSED_2, g', g_2, \mu, m$ )
19  else if  $g(x) > g'$  then
20     $g(x) = g'$ 
21     $\pi(x) = u$ 
22    UPDATE( $x, CLOSED_2, g', g_2, \mu, m$ )
```

Kaksisuuntainen Dijkstran algoritmi

Algoritmi 5: BIDIRECTIONAL-DIJKSTRA-SHORTEST-PATH(G, s, t, w)

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

Kaksisuuntainen Dijkstran algoritmi

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

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

Kaksisuuntainen A*

- Tasan sama kuin kaksisuuntainen Dijkstra, paitsi että operaation Expand rivillä 1 oleva $g(x)$ korvattava lausekkeella $f(x)$, jolle siis $f(x) = g(x) + h(x)$.
- Operaatiossa Terminate testin ehto on

$$\max\left(\min_{x \in \text{OPEN}} f(x), \min_{x \in \text{OPEN}_{REV}} f_{REV}(x)\right) \geq \mu,$$

Mikä mahtaa olla tehokkain tapaa hakea polut?

Kaikkien parien lyhimmät polut

(1) Aja kaikkien-parit algoritmin, joka palauttaa nk. "edeltäjämatriisin".

Kaikkien parien lyhimmät polut

(2) Mikä tahansa N :n solmun polku voidaan rakentaa edellä mainitusta matriisista ajassa $\mathcal{O}(N)$!

Kaikkien parien lyhimmät polut

- Floyd-Warshall toimii ajassa $\Theta(n^3)$.
- Jos $m = o(n^2)$, Johnsonin algoritmi Fibonacci-keolla on asympotoottisesti parempi: $\mathcal{O}(n^2 \log n + nm)$.
- Kumpikaan ei siis tarpeeksi tehokas prosessoimaan kokonaisen valtion tieverkkoa, sillä pelkästään solmuja on helposti yli 100000.

Dijkstran algoritmi kaarivivuilla

Jaa kaikki solmut V osituksiin V_1, \dots, V_k siten, että

$$\bigcup_{i=1}^k V_i = V,$$

ja $V_i \cap V_j = \emptyset$ kaikilla $i \neq j$.

Dijkstran algoritmi kaarivivuilla

k osion ositus voidaan merkitä funktiolla $r: V \rightarrow \{1, 2, \dots, k\}$.

Dijkstran algoritmi kaarivivuilla

Jokaiselle kaarelle asetetaan k :n bitin bittivektorin (engl. *arc-flag vector*, kaarivipuvektori); jos i des bitti on päällä, kaari on lyhimmällä polulla johonkin osion V_i solmuun.

Dijkstran algoritmi kaarivivuilla

Osion V_i "rajasolmut" ovat

$$B_i = \{v \in V_i : \exists(u, v) \in A \text{ siten että } r(v) \neq r(u)\}.$$

Dijkstran algoritmi kaarivivuilla

Jokaisen osion V_i jokaisen rajasolmun $u \in B_i$ lähtien ajetaan "takaperin" tavallinen Dijkstra ja tuloksena syntyvässä lyhimpien polkujen puussa T asetetaan jokaisen kaaren $(u, v) \in T$ ides vipu päälle.

Dijkstran algoritmi kaarivivuilla

Nyt tuloksena syntyvässä algoritmissa haettaessa polkua solmuun $t \in V$, voidaan karsia kaikki kaaret, joiden $r(t)$:s bitti ei ole päällä.

Dijkstran algoritmi kaarivivuilla

Voidaan myös kaksisuuntaista: jokaisella kaarella kaksi kaarivipuvektoria: yksi normaalia hakusuuntaa varten ja toinen käännettyä hakua varten.

Dijkstran algoritmi kaarivivuilla

Tekniikka voidaan nähdä tasapainoilevan tavallisen Dijkstran algoritmin ($k = 1, V_1 = V$) ja kaikkien parien algoritmin ($k = n$, jokainen solmu on osio) välillä.

Dijkstran algoritmi kaarivivuilla

Tutkijat raportoivat kaksisuuntaisen kaarivipu-Dijkstran olevan keskimäärin yli 500 kertaa nopeampi kuin tavallinen yksisuuntainen Dijkstra.

Dynaamiset verkot

- Aiemmin tarkastellut verkot ovat **staattisia**.
- Dynaamisessa verkossa lyhin polku ei riipu vain lähtö- ja maalisolmuista, vaan myös siitä ajasta, jona matka aloitetaan.
- Lisäksi, kaarten painot riippuvat siitä, millä ajoneuvoilla ollaan liikkeellä (bussi, raitiovaunu, metro, kävely).
- Myös solmuihin liittyy painoja ("Pekka saappui pysäkillä, jossa hän odottaa bussin neljä minuuttia.")
- Reittiopas toimii dynaamisilla verkoilla.

Dynaamiset verkot

- Jos käyttäjä syöttää lähtöajan, haku on "normaalia" lähtösolmusta maalisolmuun.
- Jos käyttäjä syöttää saapumisajan, haku on "käännetty takaperin" ja etenee "ajassa taaksepäin" kunnes kohtaa lähtösolmun.
- Kaksisuuntainen haku ei ole dynaamisilla verkoilla mahdollista, sillä se vaatii maalisolmun saapumisajankohdan, jonka voi laskea vain ajamalla yksisuuntaisen haun lähtösolmusta.

Kiitos!

Kysymyksiä?