Lyhimpien polkujen hakualgoritmit ja -järjestelmät

Rodion Efremov

Tietojenkäsittelytieteen laitos, Helsingin yliopisto

Verkot

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

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)

$Q = \langle s \rangle$
$\pi(s) =$ **nil**
while $|Q| > 0$ **do**
    $u =$ DEQUEUE(Q)
    if u **is** t **then**
        return TRACEBACK-PATH$(u, \pi,$ **nil**$)$
    for $(u, v) \in G.A$ **do**
        if v **is not yet mapped in** π **then**
            $\pi(v) = u$
            ENQUEUE(Q, v)
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.
- Aikavaatiivuus 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)

```
Q, π, d = (⟨s⟩, (s, nil), (s, 0))
Q_REV, π_REV, d_REV = (⟨t⟩, (t, nil), (t, 0))
τ, μ = (nil, ∞)
while |Q||Q_REV| > 0 do
    if τ is not nil and d(HEAD(Q)) + d_REV(HEAD(Q_REV)) ≥ μ then
        return TRACEBACK-PATH(τ, π, π_REV)
    u = DEQUEUE(Q)
    if u is mapped in π_REV and μ > d(u) + d_REV(u) then
        μ = d(u) + d_REV(u)
        τ = u
    for (u, v) ∈ G.A do
        if v is not yet mapped in π then
            π(v) = u
            ENQUEUE(Q, v)
            d(v) = d(u) + 1
    u = DEQUEUE(Q_REV)
    if u is mapped in π and μ > d(u) + d_REV(u) then
        μ = d(u) + d_REV(u)
        τ = u
    for (v, u) ∈ G.A do
        if v is not yet mapped in π_REV then
            π_REV(v) = u
            ENQUEUE(Q_REV, v)
            d_REV(v) = d_REV(u) + 1
return ⟨⟩
```

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})

$u = x$

$p = \langle\rangle$

while u **is not nil do**

  $p = \langle u \rangle \circ p$

  $u = \pi(u)$

Kaksisuuntainen haku?

if π_{REV} **is not nil then**

  $u = \pi_{REV}(x)$

  while u **is not nil do**

    $p = p \circ \langle u \rangle$

    $u = \pi_{REV}(u)$

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 edeltäjäänsä lyhimpien polkujen puussa).
- Kun solmu poistetaan avoimesta listasta, sen g-arvo on optimaali.

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

```
OPEN, CLOSED, g, π = ({s}, ∅, {(s, 0)}, {(s, nil)})
while |OPEN| > 0 do
    u = arg min_{x∈OPEN} g(x)
    if u is t then
        return TRACEBACK-PATH(u, π, nil)
    OPEN = OPEN − {u}
    CLOSED = CLOSED ∪ {u}
    for (u, x) ∈ G.A do
        if x ∈ CLOSED then
            continue
        g′ = g(u) + w(u, x)
        if x ∉ OPEN then
            OPEN = OPEN ∪ {x}
            g(x) = g′
            π(x) = u
        else if g(x) > g′ then
            g(x) = g′
            π(x) = u
return ⟨⟩
```

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: $\textsc{Update}(x, \text{CLOSED}, g', g, \mu, m)$

if $x \in \mathit{CLOSED}$ **then**
  $p = g' + g(x)$
  if $\mu > p$ **then**
    $\mu = p$
    $m = x$

Kaksisuuntainen Dijkstran algoritmi

Algoritmi 3: $\textsc{Expand}(\text{OPEN}, \text{CLOSED}, \text{CLOSED}_2, g, g_2, \pi, \mu, m, e, w)$

$u = \arg\min_{x \in \text{OPEN}} g(x)$
$\text{OPEN} = \text{OPEN} - \{u\}$
$\text{CLOSED} = \text{CLOSED} \cup \{u\}$
for $x \in e(u)$ **do**
    if $x \in \mathit{CLOSED}$ **then**
        continue
    $g' = g(u)$
    if $e(u)$ **gives child nodes of** u **then**
        "Normaali" haku.
        $g' = g' + w(u, x)$
    else
        Käännetty haku.
        $g' = g' + w(x, u)$
    if $x \notin \mathit{OPEN}$ **then**
        $\text{OPEN} = \text{OPEN} \cup \{x\}$
        $g(x) = g'$
        $\pi(x) = u$
        $\textsc{Update}(x, \text{CLOSED}_2, g', g_2, \mu, m)$
    else if $g(x) > g'$ **then**
        $g(x) = g'$

Kaksisuuntainen Dijkstran algoritmi

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

$\text{OPEN}, \text{CLOSED}, g, \pi = \{s\}, \emptyset, \{(s, 0)\}, \{(s, \textbf{nil})\}$
$\text{OPEN}_{REV}, \text{CLOSED}_{REV}, g_{REV}, \pi_{REV} = \{t\}, \emptyset, \{(t, 0)\}, \{(t, \textbf{nil})\}$
$\mu = \infty$
$m = \textbf{nil}$
while $|OPEN| \cdot |OPEN_{REV}| > 0$ **do**
  if m **is not nil then**
    $p =$ TERMINATE$(\text{OPEN}, \text{OPEN}_{REV},$
      $g, g_{REV},$
      $\pi, \pi_{REV},$
      $\mu, m)$
    if p **is not nil then**
      return p
  Triviaali kuormantasaus
  if $|OPEN| < |OPEN_{REV}|$ **then**
    EXPAND$(\text{OPEN},$
      $\text{CLOSED},$
      $\text{CLOSED}_{REV},$
      $g, g_{REV}, \pi, \mu, m, e, w)$
  else
    EXPAND$(\text{OPEN}_{REV},$
      $\text{CLOSED}_{REV},$
      $\text{CLOSED},$

Kaksisuuntainen A$*$

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

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 asymptoottisesti 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 $V_1, \ldots, V_k$ 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 $r\colon V \to \{1, 2, \ldots, k\}$.

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

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

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 $(u, v) \in T$ $i$des 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.