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 $m = |E|$.
- 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**.

Leveysuuntainen 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.

Algoritmi 2: $\textsc{Breadth-First-Search}(G, s, t)$

$Q = \langle s \rangle$
$\pi(s) = \textbf{nil}$
while $|Q| > 0$ **do**
    $u = \textsc{Dequeue}(Q)$
    if u **is** t **then**
        return $\textsc{Traceback-Path}(u, \pi, \textbf{nil})$
    for $(u, v) \in G.A$ **do**
        if v **is not yet mapped in** π **then**
            $\pi(v) = u$
            $\textsc{Enqueue}(Q, u)$
    return $\langle\rangle$

Voiko leveysssuuntaisen haun nopeuttaa?

Voi!

Kaksisuuntainen leveyssuuntainen haku

- Ajaa 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)

$Q, \pi, d = (\langle s \rangle, (s, \textbf{nil}), (s, 0))$

$Q_{REV}, \pi_{REV}, d_{REV} = (\langle t \rangle, (t, \textbf{nil}), (t, 0))$

$\tau, \mu = (\textbf{nil}, \infty)$

while $|Q||Q_{REV}| > 0$ **do**

  if τ **is not nil and** $d(\text{Head}(Q)) + d_{REV}(\text{Head}(Q_{REV})) \geq \mu$ **then**

    return Traceback-Path(τ, π, π_{REV})

  $u = \text{Dequeue}(Q)$

  if u **is mapped in** π_{REV} **and** $\mu > d(u) + d_{REV}(u)$ **then**

    $\mu = d(u) + d_{REV}(u)$

    $\tau = u$

  for $(u, v) \in G.A$ **do**

    if v **is not yet mapped in** π **then**

      $\pi(v) = u$

      Enqueue(Q, v)

      $d(v) = d(u) + 1$

  $u = \text{Dequeue}(Q_{REV})$

  if u **is mapped in** π **and** $\mu > d(u) + d_{REV}(u)$ **then**

    $\mu = d(u) + d_{REV}(u)$

    $\tau = u$

  for $(v, u) \in G.A$ **do**

    if v **is not yet mapped in** π_{REV} **then**

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

      Enqueue(Q_{REV}, v)

      $d_{REV}(v) = d_{REV}(u) + 1$

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: $\textsc{Traceback-Path}(x, \pi, \pi_{REV})$

$u = x$

$p = \langle\rangle$

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 käy polynomisessa ajassa.
- LIFO 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 pienempään 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. $a \circ b$

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

$\text{OPEN}, \text{CLOSED}, g, \pi = (\{s\}, \emptyset, \{(s, 0)\}, \{(s, \textbf{nil})\})$
while $|OPEN| > 0$ **do**
    $u = \arg\min_{x \in \text{OPEN}} g(x)$
    if u **is** t **then**
        return TRACEBACK-PATH$(u, \pi, \textbf{nil})$
    $\text{OPEN} = \text{OPEN} - \{u\}$
    $\text{CLOSED} = \text{CLOSED} \cup \{u\}$
    for $(u, x) \in G.A$ **do**
        if $x \in \text{CLOSED}$ **then**
            continue
        $g' = g(u) + w(u, x)$
        if $x \notin \text{OPEN}$ **then**
            $\text{OPEN} = \text{OPEN} \cup \{x\}$
            $g(x) = g'$
            $\pi(x) = u$
        else if $g(x) > g'$ **then**
            $g(x) = g'$
            $\pi(x) = u$
return $\langle\rangle$