Polunhakualgoritmit ja -järjestelmät

Rodion Efremov

Kandidaatintutkielma-aine
HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Helsinki, 6. lokakuuta 2014

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

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	6. lokakuuta 2014	4

Tiivistelmä — Referat — Abstract

Tiivistelmä.

Avainsanat — Nyckelord — Keywords

a, bb, ccc

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Sisältö

1 Johdanto

Polunhaku painotetuissa tai painottamattomissa verkoissa on perustavanlaatuinen 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 suuntamattoman 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 $\gamma_k = \langle u_0, u_1, \ldots, u_k \rangle$, missä mikään solmu ei esiinny yhtä kertaa enempää, ja verkossa on kaari (u_i, u_{i+1}) jokaisella $i = 0, 1, \ldots, k - 1$. Polkuun liittyvä kustannus on sen kaarien painojen summa, ja mitä tulee itse painoihin, ne oletetaan olevan ei-negatiivisia. Ei-painotettujen verkojen 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 Djikstran 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 järjestelyn takana on se, että $A*$ "tietää" mihin suuntaan haku on suunnattava, jota pääsisi maalisolmuun, ainakin parempi kuin Dijkstran algoritmi, jonka hakuavaruus kasvaa laajenevan pallon tavoin "kaikkiin suuntiin". $A*$:n pseudokoodi on tasan sama kuin Dijkstran algoritmin 1, paitsi että rivillä 6 $g(x)$:n sijasta on $f(x)$, jolle siis $f(x) = g(x) + h(x)$.

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

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

3 Kaksisuuntainen haku

Vaikka $A*$ on tyypillisesti tehokkaampi kuin Dijkstran algoritmi, käyttämällä *kaksisuuntaista* hakua, voidaan päästää verrattavissa olevaan suorituskykyyn. 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

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

Algoritmi 2: TRACEBACK-PATH(x, π, π_{REV})

```
u = x
p = ⟨⟩
while u is not nil do
    prepend u to p
    u = π(u)
Kaksisuuntainen haku?
if π_REV is not nil then
    u = π_REV(x)
    while u is not nil do
        append u to p
        u = π_REV(u)
return p
```

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

3.2 Kaksisuuntainen $A*$

4 Prioriteettijonon valinta

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

1. INSERT(H, x, k) tallettaakseen solmun x sen prioriteetin k kera,

2. DECREASE-KEY(H, x, k) päivittääkseen solmun x talletetun prioriteetin (pienemmäksi),

3. EXTRACT-MINIMUM(H) poistaakseen pienimmän prioriteetin omaava solmu, ja

4. 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) tasoitetussa vakioajassa, ja poisto-operaatio (3) tasoitetussa 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 INSERT tai EXTRACT-MINIMUM voi käydä (eksaktissa tai tasoitetussa) vakioajassa, ja toisen on käyttävä ajassa $\Omega(\log n)$, koska muuten algoritmi 3 tällaisella keolla rikkoisi lajittelemisen informaatioteoreettisen rajan, joka on $\Omega(n\log n)$. Jos kuitenkin kaarien painot ovat

Algoritmi 3: GENERIC-HEAP-SORT(S, H)

$H = \varnothing$: tyhjennä keko.
for $i = 1$ **to** $|S|$ **do**
  INSERT($H, S[i], S[i]$) : $S[i]$ on itsensä prioriteetti.
for $i = 1$ **to** $|S|$ **do**
  $S[i]$ =EXTRACT-MINIMUM(H)

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 lyhimmät 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`.