Polunhakualgoritmit ja -järjestelmät – referaatti

Rodion "rodde" Efremov

23. syyskuuta 2014

Lyhimpien polkujen haku painotetuissa tai painottamattomissa verkoissa on perustavanlaatuinen ongelma, joka ei ole tärkeä vain itsessään, vaan sillä on sovelluksia joukossa algoritmeja, joissa lyhimmän polun haku on tarvittava alioperaatio; esim. Edmonds-Karp algoritmi hakee aina lyhimmät painottamattomat polut verkossa (engl. *residual graph*) ratkaistaakseen maksimivuo-ongelman (engl. *maximum flow problem*). Tietenkin, varsinainen sovellus on reitin optimointi erilaisten mittojen mukaan: toisinaan halutaan päästää pisteestaä A pisteeseen B mahdollisimman pienessä ajassa (henkilöauto), toisinaan mahdollisimman ekologisesti (polkupyörä), jne.

Vuonna 1959 Edsger W. Dijkstra esitti polynomisessa ajassa toimivan algoritmin [1], joka, saatuaan lähtösolmun s, laskee lyhimpien polkujen puun solmusta s kaikkiin muihin saavutettaviin solmuihin painotetuissa verkoissa. Kuten monet muut, Dijkstran algoritmi vaatii prioriteettijonorakenteen (engl. *priority queue*, *heap*), mikä on vain yksi optimointiulottuvuus. Helpoin, tehokkaaksi kutsuttu prioriteettijonorakenne on binäärikeko, joka toteuttaa kaikki sisältöä muokkaavat operaatiot ajassa $O(\log n)$, jolloin Dijkstran algoritmin aikavaativuus on $O((m+n)\log n)$. Parannus edelliseen on ilmestynyt vuonna 1987 Fibonacci-keon myötä, jonka minimin poisto käy tasoitetussa (engl. *amortized*) ajassa $O(\log n)$, ja muut sisältöä muokkaavat operaatiot tasoitetussa tai eksaktissa vakioajassa, jolloin paremmaksi Dijkstran algoritmin aikavaativuudeksi on tullut $O(m + n\log n)$.

Mitä tulee muihin nopeutustekniikoihin, Hart et al. esittivät vuonna 1968 $A*$-polunhakualgoritmin [2], joka ei lajittele solmut pelkän g-arvon mukaan (solmun u g-arvo on toistaiseksi paras etäisyys lähtösolmusta u:hun), vaan käyttää f-arvoa, jolle $f = g+h$, missä h on solmun optimistinen (eli aliarvioi-

va) etäisyys maalisolmuun. Intuitio tämän järjestelyn takana on se, että $A*$ "tietää" mihin suuntaan kannattaa lähteä päästääkseen maalisolmuun, ainakin paremmin kuin Dijkstran algoritmi, joka etenee "kaikkiin suuntiin". Toisaalta, $A*$ sai varteenotettavan kilpailijan, nimittäin *kaksisuuntainen* Dijkstran algoritmi kasvattaa kaksi lyhimpien polkujen puuta kunnes kaksi "kohtaavat keskellä". Jos abstrahoidaan prioriteettijonon operaatioiden aikavaativuudet pois, alkuperäinen Dijkstran algoritmi tekee $\sum_{i=0}^{N} d^i$ verran työtä, kun kaksisuuntainen versio tekee vain noin $2\sum_{i=0}^{\lceil N/2 \rceil} d^i$, missä d on verkon solmujen keskiarvoinen aste ja N on lyhimmän polun solmujen määrä. (Ylläoleva analyysi pätee kaksisuuntaiseen leveyssuuntaiseen hakuun sellaisenaan.)

Mikäli kysytään, mikä on tehokkain tapa laskea lyhin polku solmujen s, v välillä, niin todennäköisesti päädytään aikaan $\Theta(N)$, missä N on lyhimmän s, v-polun solmujen määrä. Tällainen tehokkuus kuitenkin vaatii ns. "edeltäjämatriisin' (engl. *predecessor matrix*), jonka voi laskea ajaamalla kaikkien parien algoritmin (engl. *all-pairs shortest paths*). Niistä kaksi tunnetuinta ja helpommin toteuttavaa ovat Floyd-Warshall- ja Johnsonin algoritmit, jotka käyvät ajassa $O(V^3)$ ja $O(V^2 \log V + VE)$ vastaavasti, jolloin on selvää, ettei niitäkään pysty ottaa käyttöön, kun verkkona on esimerkiksi kokonaisen valtion tieverkosto, jossa solmuja on reilusti yli 10000. Tähän on kuitenkin tullut parannus: Möhring et al. esittivät ajatuksen osittaa *harvan* verkon solmujoukko V osioihin $V_1, V_2, \ldots, V_k$, joille $\cup_{i=1}^{k} V_i = V$ ja $V_i \cap V_j = \varnothing$ jokaisella $i \neq j$, minkä jälkeen jokaiselle verkon kaarelle $a \in E$ liitetään bittivektori $(b_1, b_2, \ldots, b_k)$, jossa bitti b_i on päällä, jos a on lyhyellä polulla johonkin osion i solmuun [**?**]. Käytännössä k:n osion ositus toteutetaan funktiolla $r\colon V \to \{1, 2, \ldots, k\}$; lisäksi, määritellään *rajasolmun* käsite: osion i rajasolmujen joukko on $B_i = \{v \in V_i\colon \exists (u, v) \in E$ siten että $r(v) \neq r(u)\}$. Jo tässä vaiheessa on tärkeää huomata tekniikan vaativan esiprosessointia saadaakseen bittivektorit (engl. *arc-flags*) asetettua oikein. Siis olkoon $s(b)$ joukko solmuja u siten että b on rajasolmu ja jokaisella $u \in s(b)$ on voimassa $(u, b) \in E$ ja $r(u) \neq r(b)$. Nyt jokaisen rajasolmun b jokaisesta solmusta $u \in s(b)$ on laskettava "takaperin" lyhimpien polkujen puu, jonka jokaiseen kaareen a liittyvään bittivektorin $f(a)$ laitetaan $r(b)$:s bitti päälle. (Varsinainen polun haku solmuun t ko. järjestelmässä karsii kaikki kaaret a pois, joille asianmukainen bitti ei ole päällä; ainakin niin pitkään kunnes pääsee samaan osioon maalisolmun kanssa.)

Kuten yllä kävi ilmi, *Dijkstra with arc-flags* -järjestelmä uhraa aikaa esiprosessointiin nopeuttaakseen hakuoperaatiota. Tekniikka voidaan nähdä

tasapainoilevan tavallisen Dijkstran algoritmin ($V_1 = V$) ja kaikki-parit –algoritmin (jokainen ositus sisältää vain yhden solmun) välillä. Osittaessa verkkoja Möhring et al. kokeilivat neljä tapaa. Niistä helpointen toteuttava on jakaa koko verkko w sarakkeeseen ja h riviin, jolloin kaikki osiot ovat geometrisesti yhtä suuria. Tämä ei kuitenkin ole vailla ongelmia; esim. 50 neliökilometrin suorakulmio pääkaupungin kohdalla sisältäisi oleellisesti enemmän solmuja ja kaareja kuin jokin alue Kainuun maakunnassa. Hieman parempi tapa osioida verkko on käyttää "nelipuu" (engl. *quadtree*), joka jokaisella tasollaan jakaa kunkin suorakulmion neljään yhtä isoon pienempään suorakulmioon seuraavalle tasolle pysäyttäen jaon niiden suorakulmioiden kohdalla jossa on enintään n solmua (n on rakenteelle annettava parametri). Toisin kuin ensimmäinen tapa, nelipuut alkaavat ottaa solmujen jakauman huomioon, vaikkakin ei niin monipuolisesti kuin *kd*-puu (engl. *kd-tree*), joka lajittele kaikki solmut esimerkiksi x-koordinaattien perusteella, valitsee listasta mediaanin, jonka x-koordinaatti on x_{mid}, ja jakaa solmut joukkoihin $V_? = \{x\colon x\,?\,x_{mid}\}$, missä $? \in \{\leq, >\}$, minkä jälkeen jaetaan molemmat samalla tavalla, mutta y-koordinaattien perusteella ja niin vaihtelevasti edelleen, kunnes jokaisessa osiossa on enintään n solmua (n annetaan parametrina). Neljäs ja viimeinen tarkasteltu ositustapa perustuu Karypis ja Kumarin tulokseen [3]: METIS, joka toimii siten, että se yhdistää paremmin yhdistetyt solmuryppäät yhdeksi solmuksi, ja jatkaen näin tuottaa toista pienemmät verkot $G_0 = G, G_1, \ldots, G_m$ kunnes G_m on niin pieni, että sille voi laskea tehokkaasti ositus, joka minimoi niiden kaaren määrä, jotka yhdistävät kaksi erilaista osiota. Kun G_m on valmis, edetään päinvastaiseen suuntaan ja laajennetaan väliverkot $G_{m-1}, \ldots, G_1$ saadakseen alkuperäisen verkon $G = G_0$. Siis käyttämällä METIS-osituksen saadaan vähiten rajasolmuja suhteessa muihin ositustekniikoihin, jolloin säästetään aikaa esiprosessoinnissa, jota voi myös suorittaa rinnakkain, sillä yhden osion esiprosessointi on riippumaton muista.

Vielä yksi nopeutustekniikka on tietenkin rinnakkaistaminen, ja Δ-stepping on juuri sitä optimointiulottuvuutta silmälläpitävä tekniikka [**?**]. Epämuodollisesti, Δ-stepping algoritmi jakaa solmut koreihin (engl. *bucket*), joista kukin sisältää ne solmut u, joiden toistaiseksi olevat g-arvot ovat välillä $[i\Delta, (i{+}1)\Delta)$, jolloin siis eri säikkeet käsittelevät eri koreissa olevia solmuja. Meyer et al. raportoivat peräkkäisen Δ-stepping algoritmin olevan noin 3.1 kertaa nopeampi kuin "optimoitu" alkuperäinen Dijkstran algoritmi, ja 16 suorittimen koneella paralleeli Δ-stepping oli noin 9.2 kertaa nopeampi verrattuna peräkkäiseen versioonsa.

Viitteet

[1] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[2] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. 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] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, December 1998.