MINIMAALISEN VIRITTÄVÄN PUUN MÄÄRÄÄMISESTÄ

Osa I

TURUN YLIOPISTO
Tietojenkäsittelyoppi
JYRKI KATAJAINEN
Lisensiaattitutkielma
Maaliskuu 1983

SARJAT:

A: Raportit
B: Käsikirjoitukset
C: Luentomonisteet
D: Käsikirjat

SERIES:

A: Reports
B: Manuscripts
C: Lecture Notes
D: Manuals

TURUN YLIOPISTO

MATEMAATTISTEN TIETEIDEN LAITOS
TIETOJENKÄSITTELYOPPI
20500 TURKU 50

UNIVERSITY OF TURKU

DEPARTMENT OF MATHEMATICAL SCIENCES
COMPUTER SCIENCE
SF-20500 TURKU 50, FINLAND

MINIMAALISEN VIRITTÄVÄN PUUN MÄÄRÄÄMISESTÄ

Osa I

TURUN YLIOPISTO
Tietojenkäsittelyoppi
JYRKI KATAJAINEN
Lisensiaattitutkielma
Maaliskuu 1983

TURUN YLIOPISTO
Matemaattisten tieteiden laitos

KATAJAINEN, JYRKI: Minimaalisen virittävän puun määräämisestä.
Lisensiaattityö, 115s.
Tietojenkäsittelyoppi
Maaliskuu 1983

Työ koostuu kahdesta osasta - johdannosta ja kuudesta erillisestä artikkelista. Johdanto on katsaus minimaalisen virittävän puun määrääviin algoritmeihin. Viime aikoina ovat tutkimuksen kohteena olleet pahimmalta ja keskimääräiseltä suoritusajaltaan nopeat algoritmit sekä likimääräiset algoritmit, todennäköisyys- ja rinnakkaisalgoritmit.

Erillisissä artikkeleissa tutustutaan tarkemmin minimaalisen virittävän puun määräämiseen moniulotteisessa avaruudessa ja keskimäärin nopeisiin minimaalisen virittävän puun määrääviin algoritmeihin. Olen tehnyt artikkelit yhteistyössä Olli Nevalaisen ja Jarmo Ernvallin kanssa. Artikkeleista kaksi on julkaistu ja kaksi julkaistaan lähitulevaisuudessa:

(1) ERNVALL J., KATAJAINEN J., NEVALAINEN O.: A minimal spanning spanning tree algorithm for a point set in Euclidean space, Rep. A28, Comput. Sci., Univ. of Turku (1980). (lyhennettynä aikakauslehdessä BIT 21,1 (1981), 46-54)
(2) KATAJAINEN J.: An implementation of an algorithm for finding minimal spanning tree in a Euclidean coordinate space, Rep. D22, Comput. Sci., Univ. of Turku (1981).
(3) KATAJAINEN J.: On the worst case of a minimal spanning tree algorithm for Euclidean space. (julkaistaan aikakauslehdessä BIT)
(4) NEVALAINEN O., KATAJAINEN J.: Experiments with a closest point algorithm in Hamming space, Angewandte Informatik 5 (1982), 277-281.
(5) KATAJAINEN J., NEVALAINEN O.: An alternative implementation of Kruskal's minimal spanning tree algorithm. (julkaistaan aikakauslehdessä Science of Computer Programming)
(6) KATAJAINEN J.: An implementation of Cheriton-Tarjan's minimal spanning tree algorithm using ordered subsets. (käsikirjoitus)

Kolmessa ensimmäisessä artikkelissa tutkitaan minimaalisen virittävän puun määräämisen kompleksisuutta euklidisessa avaruudessa. Artikkelissa (1) on teoreettisesti analysoitu erästä algoritmia ja tutkittu sen tehokkuutta kokeellisesti. Algoritmin implementoinnin yksityiskohdat esitetään artikkelissa (2). Artikkelissa (3) esitetään eräitä erikoistapauksia, joissa algoritmi käyttäytyy pahimmalla mahdollisella tavalla. Algoritmi havaittiin kokeissa nopeaksi, kun avaruuden ulotteisuus on pieni ja pisteet ovat normaalisti jakautuneita. Nopeus perustuu suurelta osin siihen, että algoritmissa usein toistuva annetun pisteen lähimmän pisteen haku suoritetaan niin kutsutun k-d puun avulla.

Edellinen algoritmi toimii myös Hamming-metriikassa. Artikkelissa (4) tutkitaan kokeellisesti, onko k-d puu paras mahdollinen lähimmän pisteen hakua tukeva hakurakenne Hamming-metriikan tapauksessa. Burkhardin ja Kellerin esittämä puurakenne, joka on suunniteltu erityisesti Hamming-metriikkaa ajatellen, osoittautuikin joskus k-d puuta paremmaksi.

Kahdessa muussa artikkelissa tutustutaan keskimääräiseltä suoritusajaltaan nopeisiin algoritmeihin. Artikkelissa (5) esitetään Kruskalin algoritmille implementointi ja osoitetaan se keskimääräisessä tapauksessa sekä teoreettisesti että kokeellisesti hyväksi. Pahimman tapauksen kompleksisuudessa on kuitenkin parantamisen varaan. Artikkelissa (6) esitetään algoritmi, jonka pahimman tapauksen kompleksisuus on Kruskalin algorimia parempi ja joka on myös keskimääräisessä tapauksessa tehokas. Käytännön kokeissa tämä algoritmi oli kuitenkin huomattavasti Kruskalin algoritmia hitaampi.

FIALLE

kirjoituspöytä
pyyhekumi
pölyinen pöytälamppu valoaan väärään suuntaan jakaen
avonainen kirja

$$P_k = \frac{\Delta W}{\Delta t}$$

kynäkuppi
yksi
kaksi
kolme
neljä
viisi
kuusi
seitsemän kynää
kaksi näistä on oikestaan tusseja
kahdeksas on pudonnut kupin viereen

istun tämän maailmani ääressä
työskentelen ja kilpailen kunnianhimoisten ajatusteni puolesta
ja sinä pöydän vastakkaisella puolella kaipaat huomiotani
silmäsi ovat vetiset
miksi en voisi hellittää hetkeksi
miksi emme voisi istua lattialle ja piirtää

perkele
nyt panen kirjan kiinni ja jätän työt huomiseen
tehdään jotain kivaa yhdessä
huomenna menemme katsomaan jääkiekkoa
pannaan vapaalle eikä ajatella

kevväällä 1982

ESIPUHE

Tämä tutkielma on syntynyt toimiessani Turun yliopiston tietojenkäsittelyopin osastolla. Työssäni tutustuin algoritmien suunnitteluun ja analysointiin. Erityisesti kohdistin mielenkiintoni minimaalisen virittävän puun määrääviin algoritmeihin.

Tutkielma koostuu kahdesta osasta - johdannosta ja kuudesta erillisestä artikkelista. Johdanto on katsaus minimaalisen virittävän puun määrääviin algoritmeihin. Viime aikoina ovat tutkimuksen kohteena olleet pahimmalta ja keskimääräiseltä suoritusajaltaan nopeat algoritmit sekä likimääräiset algoritmit, todennäköisyys- ja rinnakkaisalgoritmit. Johdannossa esitellään kaikkia tällaisia minimaalisen virittävän puun määrääviä algoritmeja.

Artikkelit olen tehnyt yhteistyössä FT Olli Nevalaisen ja FL Jarmo Ernvallin kanssa. Artikkeleista kaksi on julkaistu ja kaksi julkaistaan lähitulevaisuudessa:

1) ERNVALL J., KATAJAINEN J., NEVALAINEN O.: A minimal spanning spanning tree algorithm for a point set in Euclidean space, Rep. A28, Comput. Sci., Univ. of Turku (1980). (lyhennettynä lehdessä BIT 21,1 (1981), 46-54)
2) NEVALAINEN O., KATAJAINEN J.: Experiments with a closest point algorithm in Hamming space, Angewandte Informatik 5 (1982), 277-281.
3) KATAJAINEN J.: On the worst case of a minimal spanning tree algorithm for Euclidean space. (julkaistaan lehdessä BIT)
4) KATAJAINEN J., NEVALAINEN O.: An alternative implementation of Kruskal's minimal spanning tree algorithm. (julkaistaan lehdessä Science of Computer Programming)

Muut kaksi artikkelia ovat

5) KATAJAINEN J.: An implementation of an algorithm for finding minimal spanning tree in a Euclidean coordinate space, Rep. D22, Comput. Sci., Univ. of Turku (1981).
6) KATAJAINEN J.: An implementation of Cheriton-Tarjan's minimal spanning tree algorithm using ordered subsets. (käsikirjoitus)

Haluan kiittää kaikkia, jotka ovat myötävaikuttaneet tämän tutkielman syntyyn. Kiitokset kuuluvat erityisesti apulaisprofessori Olli Nevalaiselle, jonka oppipoikana olen saanut olla, ja lehtori Martti Penttoselle, jonka erinomainen luentosarja "Algoritmien analyysi" sai minut innostumaan algoritmien teoriasta. Työtä tehdessäni on minulle ollut suunnattomasti hyötyä siitä, että olen ollut kiinnostunut samoista asioista kuin lehtori Jarmo Ernvall. Kiitokset myös kaikille tietojenkäsittelyopin osaston henkilökunnan sekä entisille että nykyisille jäsenille, koska ymmärsitte minua ordoneuroosissani.

SISÄLLYSLUETTELO

Osa I: Johdanto

1. Ongelman asettelu 1
2. Sovellutuksia 2
   2.1 Läheiset graafiongelmat 2
   2.2 Verkostoihin liittyvät optimointitehtävät 2
   2.3 NP-täydellisten ongelmien approksimointi 3
   2.4 Hahmontunnistus 3
   2.5 Formatoitujen tiedostojen tiivistys 4
3. Perusalgoritmit 5
   3.1 Kruskalin algoritmi 5
   3.2 Primin algoritmi 6
   3.3 Sollinin algoritmi 6
4. Ongelma moniulotteisessa avaruudessa 8
5. Keskimääräiseltä suoritusajaltaan nopeita algoritmeja 9
6. Likimääräisistä ratkaisumenetelmistä 10
7. Todennäköisyysalgoritmeista 11
8. Rinnakkaisalgoritmeista 12
9. Yhteenveto toisen osan artikkeleista 14
Lähdeluettelo 15

Osa II: Erilliset artikkelit

(1) A minimal spanning tree algorithm for a point set in Euclidean space
1. Introduction
2. Multifragment algorithm
3. Computation time
4. Experiments
5. Conclusions
References

(2) An implementation of an algorithm for finding minimal spanning trees in a Euclidean Coordinate Space
1. Introduction
2. The data structures
2.1 k-d tree
2.2 The priority queue of the sizes of the subtrees
2.3 Structures for the trees Ti
2.4 The priority queues
3. Instructions
4. Conclusions
References
Appendix

(3) On the worst case of a minimal spanning tree algorithm for Euclidean space
1. Introduction
2. The worst case study
2.1 The upper and lower bounds for $k \geq 2$
2.2 The upper bound for k=1
3. Refinements to the algorithm
References

(4) Experiments with a closest point algorithm in Hamming space
1. Introduction
2. Burkhard-Keller search tree
3. Experiments
3.1 Burkhard-Keller tree
3.2 k-d tree
3.3 Comparison of the results
4. Concluding remarks
Appendix
References

(5) An alternative for the implementation of Kruskal's minimal spanning tree algorithm
1. Introduction
2. A modified Kruskal's algorithm
3. Analysis
4. Profiling the running time
5. Shortest route problems
6. Concluding remarks
References

(6) An implementation of Cheriton-Tarjan's minimal spanning tree algorithm using ordered subsets
1. Introduction
2. Algorithm
3. Performance
4. Experiments
5. Discussion
References
Appendix

Osa I: Johdanto

1. ONGELMAN ASETTELU

Tässä tutkielmassa tutustutaan menetelmiin, joilla voidaan ratkaista seuraava ongelma: On annettu pisteet, joista jotkut tai kaikki on yhdistetty toisiinsa viivoilla, joiden pituudet tunnetaan. Pisteet tulisi yhdistää siten, että yhdysviivojen pituuksien summa minimoituu.

Täsmällisemmin tämä ongelma voidaan määritellä graafiteorian käsitteistöä käyttäen [P57].

<u>Määritelmä</u>. Oletetaan, että $G = (P,V,d)$ on painotettu, suuntaamaton, yhtenäinen ja silmukaton graafi, missä $P = \{1,2,\ldots,N\}$ on pisteiden joukko ja V on viivojen joukko ($|V| = M$). Funktio $d: V \to \mathbb{R}+$ ilmoittaa viivan $v = (i,j)$ painon (pituuden) $d(v)$. Graafin G <u>virittävä puu</u> T on sellainen graafin G aligraafi (P,V_1,d), että $V_1 \subseteq V$, $|V_1| = N-1$ ja aina kahden eri pisteen välillä on tarkalleen yksi polku. Virittävän puun $T = (P,V_1,d)$ <u>paino</u> $d(T)$ on

$$d(T) = \sum_{v \in V_1} d(v) .$$

Virittävien puiden joukosta on etsittävä se, jonka paino on pienin - <u>minimaalinen virittävä puu</u> (MVP).

Kun tutkielmassa myöhemmin puhutaan yleisesti vain graafista sitä tarkemmin määrittelemättä, on tällöin kyse nimenomaan edellisen määritelmän mukaisesta graafista. Tähän määritelmään on syytä liittää seuraavat huomautukset.

<u>Huomautus</u> 1. Määritelmässä oletetaan, että graafi on yhdistetty. Joskus määritelmä esitetään muodossa, jossa tästä oletuksesta on luovuttu. Tällöin on määrättävä graafin minimaalinen virittävä metsä [KV72].

<u>Huomautus</u> 2. Puhuttaessa viivojen pituudesta on luonnollista, että nämä ovat positiivisia. Aivan hyvin viivan painona voitaisiin sallia myös mielivaltainen reaaliluku. Useimmat algoritmit toimivat moitteettomasti reaalilukujenkin tapauksessa.

<u>Huomautus</u> 3. Viivoihin liittyvät painot voi olla etukäteen kullekin viivalle annettu tai ne voidaan laskea viivojen päätepisteiden funktiona. Joskus pisteet on annettu metrisessä avaruudessa $\mathbb{R}_p^k$, jonka <u>metriikkana</u> on niin kutsuttu L_p-metriikka. Toisin sanoen viivan $v = (x,y)$ paino saadaan sen päätepisteiden $x = (x_1,x_2,\ldots,x_k)$ ja $y = (y_1,y_2,\ldots,y_k)$ välisenä etäisyytenä kaavasta

$$d_p(x,y) = \{ \sum_{i=1}^{k} [|x_i-y_i|]^p \}^{1/p} , \quad 1 \leq p < \infty ,$$

ja

$$d_\infty(x,y) = \max_i \{|x_i-y_i|\} .$$

Jos $p=2$, on kyseessä tuttu euklidinen etäisyys.
Kun pisteiden x ja y komponentit ovat äärellisten joukkojen alkioita, voidaan käyttää pisteiden välistä <u>Hamming-etäisyyttä</u>, joka kertoo niiden komponenttien määrän, joissa pisteet eroavat:

$$d_H(x,y) = \sum_{i=1}^{k} \delta(x_i,y_i) ,$$

missä

$$\delta(a,b) = \begin{cases} 1 , & \text{kun } a \neq b; \\ 0 , & \text{kun } a = b . \end{cases}$$

2. SOVELLUTUKSIA

Ennen tutustumista edellä määritellyn ongelman ratkaisumenetelmiin, esitetään tyypillisimmät sovellutukset, joissa MVP:ta käytetään. Sovellutukset esitellään tässä melko lyhyesti. Tarkemman kuvan halutessaan lukijan kannattaa tutkia sovellutuksen yhteydessä mainittuja lähteitä.

2.1 Läheiset graafiongelmat

MVP:den määräämisestä saattaa olla hyötyä usean samankaltaisen ongelman ratkaisussa. Maksimaalisen virittävän puun määrääminen käy usein kätevästi MVP:n ratkaisevan algoritmin avulla [P57]. Kaikkia sellaisia algoritmeja, jotka sallivat myös negatiiviset painot, voidaan käyttää sellaisenaan maksimaalisen virittävän puun määräämiseen, kunhan ennen algoritmin suorittamista kaikkien viivojen painot muutetaan vastaluvuiksi.

Virittävän puun paino lasketaan määritelmän mukaan puuhun kuuluvien viivojen painojen summana. Useimpien algoritmien toiminnan kannalta riittäisi, että puun painon määrää jokin monotonisesti kasvava symmetrinen funktio, jonka argumentteina ovat puun viivojen painot [P57]. Puun T painon voitaisiin esimerkiksi määritellä olevan

$$d(T) = \prod_{v \in T} d(v) \text{ , kun } d(v) > 0, \text{ tai}$$

$$d(T) = \{ \sum_{v \in T} [d(v)]^p \}^{1/p} \text{ , kun } d(v) > 0 \text{ ja } p > 0.$$

Koska

$$\lim_{p \to \infty} \{ \sum_{v \in T} [d(v)]^p \}^{1/p} = \max \{d(v) \mid v \in T\} \text{ , niin}$$

MVP:ssa maksimi paino minimoituu. Kalaba [K64] on käyttänyt tätä MVP ominaisuutta seuraavassa sovellutuksessa. Tarkastellaan järjestelmää, joka voi olla useassa diskreetissä tilassa. Siirryttäessä tilasta toiseen liittyy tähän riski. Tehtävänä on löytää kahden tilan välille sellainen polku, että maksimi riski tällä polulla on mahdollisimman pieni. Tämän polun määrää MVP.

MVP:n määrääminen on edullista, jos ei aivan välttämätöntä, ratkaistaessa joitakin muita vaikeampia ongelmia. Esimerkkinä mainittakoon muutama tällainen tapaus.

1) Muodostettava sellainen graafin MVP, jonka jokaisesta pisteestä i lähtee korkeintaan b_i viivaa. On siis määrättävä niin sanottu asteraajoitettu minimaalinen virittävä puu. [NH80]
2) Kauppamatkustajan reitin määrääminen on edellisen tehtävän mielenkiintoinen erikoistapaus. Kauppamatkustajan reitissä jokaisen pisteen asteeen on oltava tarkalleen kaksi. [HK70],[HK71]
3) Määrättävä K pienipainoisinta virittävää puuta. [G77],[KIM81]
4) Määrättävä sellaisen graafin, jonka jokainen viiva on väritetty joko punaiseksi tai vihreäksi, minimaalinen virittävä puu, jossa on tarkalleen q punaista viivaa. [GT79]

2.2 Verkostoihin liittyvät optimointitehtävät

Erilaisten optimaalisten verkostojen suunnittelu oli itseasiassa alkusysäys MVP määräävien algoritmien tutkimukselle. Prim [P57] puki tämän verkostoille tunnetun ongelman graafimuotoon. Tästä huolimatta nämä optimointitehtävät käsitellään nyt ainoastaan yhtenä sovellutuksena.

Seuraava tehtävä on selvästi ekvivalentti MVP:n määräämisen kanssa:
Kytke virtapiirin komponentit yhteen niin lyhyillä johdoilla kuin mahdollista [LW57].
Tällöin tavoitteena on vähentää pitkistä johdoista aiheutuvia viiveitä.

Toisaalta graafin pisteet voidaan tulkita kaupungeiksi, joiden välille tulisi vetää puhelinlinjat siten, että kaapelia kuluisi mahdollisimman vähän [P57]. Siinä tapauksessa, että puhelinkeskuksia voidaan sijoittaa vain kaupunkeihin, MVP on tällä ongelmalle optimaalinen ratkaisu. Jos kaupunkien ulkopuoliset keskukset ovat sallittuja, optimaalisen puhelinverkoston määrääminen on niin sanottu Steiner-puu ongelma. Tämä ongelma on kuitenkin osoittautunut NP-täydelliseksi[1] [GGJ76], joten käytännössä meidän on joko tyydyttävä MVP:n antamaan ratkaisuun, jonka määrääminen on paljon helpompaa kuten tulemme huomaamaan, tai käytettävä jotain likimääräistä ratkaisumenetelmää.

2.3 NP-täydellisten ongelmien approksimointi

Koska kauppamatkustajan reitin määrääminen on NP-vaikeata [GGJ76], on pyritty kehittämään polynomiaalisessa ajassa toimivia algoritmeja, jotka antavat hyvän kauppamatkustajan reitin approksimaation. Christofides [S78, s. 152-155] esitti algoritmin, jolla pystytään määräämään sellainen likimääräinen kauppamatkustajan reitti, jonka pituus on korkeintaan 1,5 kertainen optimaalisen reitin pituuteen verrattuna. Algoritmissa käytetään MVP:ta ja sen suoritusaika on $O(n^3)$ [2].

MVP on käyttökelpoinen apuväline määrättäessä approksimaatio Steiner-puulle [KMB81]. MVP:ta on käytetty myös muiden NP-täydellisten ongelmien approksimoinnissa [FJ81].

2.4 Hahmontunnistus

Usein törmätään tehtävään, jossa olisi kuvattava moniulotteinen tietojoukko havainnollisesti. Hahmontunnistus tarjoaa ainakin kaksi menetelmää tämän tehtävän ratkaisemiseksi - ryväsanalyysin ja erilaiset kuvaukset. Hierarkkinen ryväsanalyysi on sovellutus, jossa MVP:ta on käytetty runsaasti [GR69],[Z71]. Tässä menetelmässä moniulotteisen avaruuden pisteet ryhmitellään erillisiin joukkoihin siinä toivossa, että ne pisteet, joilla tietyt tutkitut ominaisuudet ovat samanlaisia, tulevat samaan joukkoon. Nämä pistejoukot ryhmitellään edelleen suuremmiksi joukoiksi, kunnes kaikki pisteet tulevat hierarkkisesti luokiteltua. Kun tämä ryhmittely halutaan tehdä ainoastaan yhden viivan (kahden joukon välisen lyhyimmän viivan) perusteella, on MVP käyttökelpoinen, koska näin hierarkkisesti edetessä joukkojen välinen lyhyin viiva kuuluu aina MVP:hun.

Toisaalta moniulotteisen tiedon visualisointiin sopii epälineaariset kuvaukset, jotka perustuvat MVP:hin. Esimerkiksi voidaan kuvata alkuperäinen moniulotteinen pistejoukko tasolle ja pyrkiä täten selvittämään joukon rakennetta [LSB77]. Kuvauksia voidaan käyttää myös apuna ryväsanalyysissä. Koska moniulotteisten pisteiden samankaltaisuus on usein vaikeasti selvitettävissä, rypäiden havaitseminen saattaa olla helpompaa, jos pistejoukko kuvataan joksikin toiseksi pistejoukoksi [SV75].

---

1)
NP-täydellisyyden käsite on esitelty muun muassa oppikirjassa [HS78].

2)
Symbolit O ja Ω tarkoittavat funktioille f ja g seuraavaa:
(i) $f(x) = O(g(x))$ jos ja vain jos on olemassa sellainen positiivinen vakio C, että kaikilla luvuilla x $f(x) \leq Cg(x)$;
(ii) $f(x) = \Omega(g(x))$ jos ja vain jos on olemassa sellainen positiivinen vakio D, että kaikilla luvuilla x $f(x) \geq Dg(x)$.

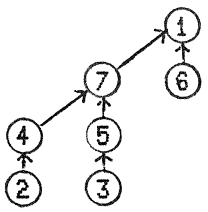
Tiedon ei tarvitse olla moniulotteista ja siitä huolimatta ihmisen on vaikea hahmottaa sitä. Osteen ja Lin käyttivät MVP:ta kuvan ääriviivoja määrättäessä [OL74]. Tässä sovellutuksessa kuva on matriisi, jonka jokaisella pisteellä on ei-negatiivinen intensiteetti. Tästä matriisista saadaan graafi vetämällä viiva kahden pisteen välille, jos pisteet ovat naapureita ja kummankin pisteen intensiteetti on nollasta eroava. Graafin viivojen painot lasketaan grey-etäisyytenä. Pisteiden välinen etäisyys saadaan jakamalla normaali euklidinen etäisyys pisteiden intensiteettien keskiarvolla. Nyt voidaan määrätä tämän graafin MVP. Tätä puuta sopivasti karsimalla saadaan kuvan ääriviivat. Menetelmää voidaan käyttää esimerkiksi hahmoteltaessa mikroskoopilla saadusta kuvasta kromosomin ääriviivat.

2.5 Formatoitujen tiedostojen tiivistys

MVP:ta voidaan käyttää myös formatoitujen tiedostojen tiivistämiseen [KLC77],[EN79]. Formatoidussa tiedostossa kaikki tietueet ja kentät ovat vakiopituisia, joten kukin kenttä voi saada vain tietyn kiinteän määrän eri arvoja. Jos tiedostossa on paljon lähes samankaltaisia tietueita, tiedosto tiivistyy, kun ensin ketjutetaan tietueet yhteen ja tämän jälkeen kunkin tietueen kohdalla kerrotaan vain, miten se eroaa edellisestä. Tiedosto voidaan ketjuttaa myös MVP:ksi Kuvan 1 tapaan [E80].

tietue	kentät
(1)	B H F
(2)	C H G
(3)	A Y G
(4)	C Y F
(5)	A Y F
(6)	B Y E
(7)	A H E

(a)



tietue	osoitin	bittilista	kentät
(1)	nil	111	B H F
(2)	↑4	011	H G
(3)	↑5	001	G
(4)	↑7	100	C
(5)	↑7	011	Y.F
(6)	↑1	011	Y E
(7)	↑1	101	A E

(b)

Kuva 1. (a) alkuperäinen tiedosto; (b) ketjutettu tiivistys MVP:n avulla.

Nyt esimerkiksi tietueen (3) purku suoritetaan seuraavasti:
-bittilistan avulla nähdään, että tietueen 3. kenttä on G;
-isäketjua pitkin kulkemalla selviää muiden kenttien arvot. Tietueen (5) ja tietueen (3) 2. kenttä on sama eli Y. Tietueen (3) 1. kentän määrää tietue (7) eli se on A.

Tiivistettyjen tiedostojen yhteydessä ongelmana on usein tiedoston päivitys. Näin tässäkin sillä, jos halutaan pitää muistitila koko ajan minimaalisena, joudutaan usein suorittamaan hyvin työläs MVP:n päivitys [CH78]. Täten päivitysten tekemiseksi on keksittävä muita ratkaisuja.

3. PERUSALGORITMIT

Graafin MVP:n määrääminen on yksi niistä harvoista graafiongelmista, jonka katsotaan täysin ratkenneen. Tällä tarkoitetaan sitä, että on löydetty useita polynomiaalisessa ajassa toimivia algoritmeja. Viime aikoina tutkimukset ovatkin keskittyneet siihen, miten tehokkaasti eri algoritmit voidaan implementoida tietokoneissa.

Tässä kappaleessa esitellään kolme algoritmia: Kruskalin, Primin ja Sollinin algoritmit. Näiden kaikkien algoritmien toiminta perustuu seuraavaan lemmaan.

<u>Lemma</u> 3.1 Olkoon T_k ja T_n kaksi mielivaltaista erillistä alipuuta, jotka on saatu MVP muodostamisen aikana yksitellen viivoja lisäämällä. Olkoon d_{kn} pienin sellaisen viivan paino, jonka toinen päätepiste on alipuussa T_k ja toinen alipuussa T_n toisin sanoen

$$d_{kn} = \min_{i \in T_k} \{ \min_{j \in T_n} [d((i,j))] \}, \; k \neq n .$$

Kun aluksi tulkitaan kaikki pisteet erillisiksi alipuiksi, niin toistamalla seuraavaa operaatiota saadaan lopulta graafin MVP:
Määrää jollekin alipuulle T_β sellainen alipuu $T_{n'}$, jolle

$$d_{\beta n'} = \min_{T_n} \{d_{\beta n}\}$$

ja olkoon v se viiva, jonka paino on $d_{\beta n'}$. Yhdistetään alipuut T_β ja T_n viivalla v.

Lemman todistus löytyy esimerkiksi Christofidesin oppikirjasta [C75, s. 135-137].

3.1 <u>Kruskalin algoritmi</u>

<u>Kruskalin algoritmissa</u> [K57] toistetaan seuraavaa askelta niin kauan kuin mahdollista:
Etsi vielä toistaiseksi valitsematta jääneiden viivojen joukosta pienipainoisin viiva, joka ei muodosta sykliä aikaisemmin valittujen viivojen kanssa.
Tämä algoritmi voidaan esittää myös seuraavassa muodossa [HS78, s. 179-183]

```
PROCEDURE Kruskal;
// Tämä ohjelma muodostaa N-pisteisen graafin G = (P,V,d) MVP:n_T //
BEGIN
   T:=∅;
   WHILE |T| < N-1 DO BEGIN
      Valitse V:n pienipainoisin viiva v;
      Poista v joukosta V;
      IF T U {v} ei sisällä sykliä THEN T:=T U {v};
   END;
END.
```

Viivojen lajittelu painon mukaan voidaan suorittaa joko <u>kekolajittelulla</u> [AHU74, s. 87-92] tai <u>osittaisella pikalajittelulla</u> [B82]. Näissä kummassakin lajittelu etenee sitä mukaa kuin viivoja tarvitaan. Kysymys, muodostaako viivakandidaatti syklin T:hen, voidaan ratkaista esimerkiksi joukkojen käsittelyyn suunniteltuja tehokkaita tietorakenteita käyttäen [HS78, s. 70-79]. Tällöin pidetään jokaisen puun pisteet yhdessä joukossa. Aluksi jokainen piste on omassa joukossaan. Kun kaksi puuta yhdistetään suoritetaan joukkojen UNIONI. Tutkittaessa, muodostaako annettu viiva syklin, suoritetaan kaksi ETSI operaatiota. Jos viivan päätepisteet kuuluvat samaan joukkoon, muodostuu sykli. Näin syklin testaus voidaan suorittaa hyvin nopeasti ja algoritmin pahimman tapauksen kompleksisuutta määrättäessä määrääväksi tekijäksi tulee painojen lajitteluun kuluva aika, joka kekolajittelulle on $O(M \log M)$ ja pikalajittelulle $O(M^2)$.

Kruskal esitti algoritmilleen seuraavan duaalin menetelmän. Toistamalla seuraavaa askelta niin kauan kuin mahdollista saadaan graafin MVP myös määrättyä:

Valitse niiden viivojen joukosta, joita ei ole vielä valittu, suuripainoisin viiva, joka ei tee graafia epäyhtenäiseksi.

Valitsematta jääneet viivat muodostavat MVP:n.

3.2 Primin algoritmi

Primin algoritmissa [P57] sovelletaan lemmaa 3.1 eri tavalla kuin Kruskalin algoritmissa. Tässä kasvatetaan yhtä ja samaa alipuuta koko ajan. Tähän alipuuhun liitetään aina pienipainoisin viiva, jonka toinen päätepiste ei kuulu alipuuhun. Primin algoritmi on seuraava

```
PROCEDURE Prim;
// Määrätään N-pisteisen graafin G = (P,V,d) MVP:n T //
BEGIN
   Olkoon p mielivaltainen joukon P piste;
   A:={p}; // Joukko A sisältää MVP:hun kuuluvat pisteet //
   T:=0;
   WHILE |T| < N-1 DO BEGIN
      Etsi pienipainoisin sellainen viiva v = (p,q), että p∈A ja q∉A;
      T:=T U {v}; A:=A U {q};
   END;
END.
```

Primin algoritmi on nähtävissä erikoisena prioriteettijono-ongelmana [J75]. Jokaiseen joukoon A kuulumattomaan pisteeseen liitetään prioriteetti, joka sillä hetkellä kertoo pienimmän painon joukon A pisteeseen. Algoritmin aikana on pystyttävä löytämään piste, jolla on pienin prioriteetti (MINIMI). Koska joukkoon A lisätään koko ajan pisteitä, niin joidenkin pisteiden prioriteetti saattaa muuttua, jolloin on suoritettava prioriteetin PÄIVITYS. MINIMI joudutaan hakemaan kaikkiaan N-1 kertaa ja PÄIVITYS on tarpeen korkeintaan M kertaa.

Primin algoritmin implementoinneissa tämä prioriteettijono-ongelma on ratkaistu hyvin eri tavalla. Yksinkertaisinta on toteuttaa prioriteettijono vektorina, jossa on positio jokaista pistettä kohti. Tällöin PÄIVITYS voidaan suorittaa vakioajassa, mutta MINIMIn määrääminen edellyttää koko vektorin läpikäyntiä eli se vie ajan $O(N)$ [D59],[W72]. Algoritmin suoritusaika on täten $O(N^2)$.

Prioriteettijono voidaan toteuttaa myös kekona [KV72], jolloin PÄIVITYS vaatii $O(\log N)$ ajan ja MINIMI vaatii $O(1)$ ajan. Algoritmin suoritusaika on tällöin $O(M \log N)$. Jonhson [J75] osoitti, että β-kekoja käyttäen PÄIVITYS voidaan suorittaa ajassa $O(\beta \log_\beta N)$ ja MINIMI ajassa $O(\log_\beta N)$. Näin tiheille graafeille voisin sanoen graafeille, joilla $M = \Omega(N^{1+\epsilon})$, missä ϵ on kiinteä positiivinen vakio, saadaan algoritmi, joka toimii ajassa $O(M)$, kun β kiinnitetään sopivasti ($\beta = \lceil N^{\epsilon} \rceil$).

Lähteissä [KV72],[HJS80] on esitetty muita tapoja implementoida Primin ja Kruskalin algoritmit. Kyseessä olevissa lähteissä vertailtiin myös algoritmien tehokkuutta keskenään. Kokeet vahvistivat teorian antamia tuloksia; Kruskalin algoritmi on Primin algoritmia tehokkaampi ainoastaan erittäin harvoilla graafeilla.

3.3 Sollinin algoritmi

Lemma 3.1 on voimassa myös siinä tapauksessa, että alipuu koostuu vain yhdestä pisteestä. Näin ollen MVP sisältää kustakin pisteestä lähtevän pienipainoisimman viivan. Sollinin algoritmissa [BG85, s. 179-180] käytetään tätä MVP:n ominaisuutta. Ensimmäisessä vaiheessa valitaan jokaisesta pisteestä lähtevä pienipainoisin viiva. Jos näin valitut viivat muodostavat puun tämä on MVP. Tavallisesti viivat muodostavat kuitenkin toisistaan erillisten alipuiden metsän. Tämän jälkeen nämä alipuut kutistetaan yhdeksi pisteeksi poistamalla viivat, joiden päätepisteet ovat samassa alipuussa, ja kahden alipuun väliset viivat pienintä lukuunottamatta. Sitten samaa algoritmia sovelletaan näin saatuun uuteen graafiin. Näin jatketaan kunnes alkuperäinen graafi on kutistunut yhdeksi pisteeksi.

Tavallisesti kutistamista ei suoriteta vaan alipuut nähdään sellaisenaan. Aluksi kaikki pisteet nähdään omina alipuinaan. Tämän jälkeen tutkitaan, mistä alipuun pisteestä lähtee pienipainoisin viiva, jonka toinen päätepiste on jossain toisessa alipuussa. Sitten nämä kaksi puuta yhdistetään. Muodollisesti Sollinin algoritmi voidaan esittää seuraavasti:

```
PROCEDURE Sollin;
// Määrätään yhtenäisen N-pisteisen graafin G = (P,V,d) MVP T //
BEGIN
   Alustetaan lista L singleton joukoilla {1},{2},...,{N};
   T:=0;
   WHILE |L| > 1 DO BEGIN
      Ota listan ensimmäinen joukko S;
      Etsi pienipainoisin viiva v = (i,j), jolla i∈S ja j∈S1≠S;
      T:=T U {v};
      Yhdistä joukot S ja S1;
      Poista joukot S ja S1 listasta L;
      Liitä näin saatu uusi joukko listan L loppuun;
   END;
END.
```

Pahimmassa tapauksessa puiden määrä ainostaan puolittuu joka iteraatiokerralla, joten algoritmin kompleksisuus on O(M logN). Jos alipuiden kutistaminen suoritetaan, algoritmin kompleksisuus on O(min {M logN, N^2}) [CT76].

Etsittäessä pienipainoisinta viivaa v voidaan joutua käymään läpi kaikki joukosta S lähtevät viivat. Yao [Y75] esitti, että on edullisempaa ryhmitellä ensin kustakin pisteestä lähtevät viivat painon mukaan k:hon yhtäsuureen ryhmään siten, että ryhmässä i viivojen painot ovat pienempiä kuin ryhmässä i+1. Ryhmän sisällä viivojen järjestys on mielivaltainen. Nyt etsittäessä yhdestä pisteestä lähtevää pienipainoisinta viivaa riittää, kun käydään läpi yksi ryhmä eikä kaikkia kyseessä olevasta pisteestä lähteviä viivoja. Kun k valitaan sopivasti, niin tämä ryhmittely on kaikista työläin operaatio Sollinin algoritmissa. Jos kuitenkin käytetään lineaarisessa ajassa toimivaa <u>mediaaninhakuohjelmaa</u> (ks. esimerkiksi [AHU74, s. 97-99]) voidaan Sollinin algoritmi implementoida ajassa O(M loglogN) toimivaksi.

Yao toteaa, että edellisessä ordolausekkeessa johtavan termin M loglogN vakiokerroin on 6, kun M ≥ N logN ja 8, kun M < N logN, jos käytetään ajassa 3N toimivaa mediaaninhakuohjelmaa [SPP76]. Tätä vakiota voidaan pienentää korvaamalla ryhmittely, k:n kokoisilla <u>järjestetyillä</u> <u>osajoukoilla</u> [CT76]. Parannus johtuu siitä, että pienten joukkojen lajittelu käy nopeammin kuin ryhmittely lineaarisella mediaaninhakuohjelmalla.

Cheriton ja Tarjan [CT76] esittivät myös oman Sollinin algoritmin muunnelman. He ottivat edellisestä poiketen listasta L aina mahtavuudeltaan pienimmän joukon. Lisäksi he käyttivät pistejoukkojen käsittelyssä jo edellä mainittua tehokasta UNIONI-ETSI algoritmia sekä viivajoukkojen käsittelyssä he käyttivät <u>vasemmanpuoleisia</u> <u>puita</u>, joiden yhdistäminen suoritetaan <u>viivyttäen</u>. Näitä puita käytetään prioriteettijonoina, jolloin alipuusta lähtevän pienipainoisimman viivan haku sujuu nopeasti. He osoittivat, että algoritmin suoritusaika on Yaon algoritmin tapaan pahimmassa tapauksessa O(M loglogN). Heidän algoritmissaan vasemmanpuoleisten puiden tilalla voitaisiin käyttää myös muita rakenteita. Vuillemin onkin todennut, että hänen tutkimansa <u>binomijonot</u> ovat tässä yhteydessä Cheritonin ja Tarjanin esittämiä vaihtoehtoja parempia [V78]. Vielä on avoin kysymys, millä tietorakenteella Cheriton ja Tarjanin algoritmissa ordolausekkeen johtavan termin vakiokerroin minimoituu.

Cheriton ja Tarjan toteuttivat alipuiden kutistuksen, tai siivouksen kuten he sitä kutsuivat, myös omassa algoritmissaan. Siivouksesta on hyötyä silloin kun kahden yhdistettävän puun välillä on paljon yhdistämisviivoja. Näin esimerkiksi tiheille graafeille heidän algoritminsa toimii ajassa O(M).

Nämä teoreettisesti tehokkaat Sollinin algoritmin muunnelmat vaativat paljon muistia. Lisäksi termin M loglogN vakio on melko suuri. Vielä käytännössä perusalgoritmit alkuperäisessä muodossaan ovat kilpailukykyisiä näiden muunnelmien kanssa [K83].

4. ONGELMA MONIULOTTEISESSA AVARUUDESSA

Kappaleessa 1 tuli esille, että viivojen painojen ei tarvitse olla ennalta määrättyjä vaan ne voidaan laskea päätepisteiden funktiona algoritmin suorituksen aikana. Tavallisesti pisteet nähdään k-ulotteisen avaruuden pisteinä, jolloin pisteiden välinen etäisyys vastaa viivojen painoja. Kahden pisteen välisen etäisyyden oletetaan olevan laskettavissa vakioajassa. Tässä muodossa ongelman syöttö koostuu ainoastaan kN luvusta. Täten on luonnollista, että yleisesti graafeille suunniteltujen algoritmien antamat kompleksisuusrajat pyritään murtamaan.

Ellei käytettävää etäisyysfunktiota tunnettaisi, olisi Primin algoritmi optimaalinen, koska kaikki pisteiden väliset etäisyydet ovat olemassa (vrt. täydellinen graafi). Kuitenkin tavallisesti tiedämme etukäteen millaisen funktion avulla pisteiden välinen etäisyys lasketaan. Tätä tietoa hyväksi käyttäen voimme tässä erikoistapauksessa luoda Primin algoritmia parempia algoritmeja.

Erityisesti pyritään kehittämään algoritmeja, jotka ovat Dijkstran [D59] antamaa Primin algoritmin implementointia tehokkaampia. Tässä algoritmissa jokaisen pisteparin välinen etäisyys joudutaan laskemaan ainoastaan kerran ja riittää, kun samanaikaisesti muistissa on talletettuna N etäisyyttä. Esimerkiksi Kruskalin algoritmissa kaikki etäisyydet on talletettava samanaikaisesti, jotta niiden lajittelu olisi mahdollista.

Kun pisteiden välinen etäisyys lasketaan euklidisena etäisyytenä, niin tasolla on pystytty löytämään ajassa O(N logN) toimiva algoritmi [S78, s. 171-194]. Tämä tulos on myöhemmin yleistetty kaikille L_p metriikoille [H79],[L80]. Algoritmissa muodostetaan ensin pistejoukolle ajassa O(N logN) <u>Voronoi-diagrammi</u>, jonka avulla suurin osa pisteiden välisistä etäisyyksistä saadaan suljettua tarkastelun ulkopuolelle. Itseasiassa MVP:hun kuuluvien viivakandidaattien määrä saadaan supistettua määrään 3N-6. Kaiken lisäksi viivakandidaatit muodostavat tasograafin, jonka MVP saadaan ajassa O(N) [CT76].

Myös tapauksessa $k>2$ on esitetty Primin algoritmin kanssa kilpailukykyisiä algoritmeja [BF78]. Bentleyn ja Friedmanin algoritmit toimivat kaikilla kappaleessa 1 esitetyillä etäisyysfunktioilla. Näissä algoritmeissa MVP:n alipuita kasvatetaan määräämällä <u>k-d puun</u> [B75] avulla alipuuta lähinnä oleva piste. Toisessa heidän algoritmissaan kasvatetaan yhtä alipuuta kerrallaan Primin algoritmin tapaisesti ja toisessa kasvatetaan useampaa alipuuta samanaikaisesti. Navalainen, Ennvall ja Katajainen [NEK81] ovat esittäneet tämän jälkimmäisen algoritmin muunnelman, joka on alkuperäistä huomattavasti yksinkertaisempi mutta silti melko tehokas. Kaikki nämä edellä mainitut algoritmit ovat osoittautuneet kokemusperäisesti hyviksi. Heikkoutena on mainittava se, että algoritmien suoritusaika on pahimmassa tapauksessa $O(N^2 \log N)$, kun $k \geq 2$ [K82].

Yao on pyrkinyt tehokkaaseen pahimmassakin tapauksessa nopeaan algoritmiin [Y82]. Hänen algoritminsa toimii L_1, L_2 ja L_∞ metriikoilla. Tehokkaiksi osoittautuneita moniulotteisien hakuongelmien ratkaisuissa käytettyjen tietorakenteiden avulla Yao osoitti, että on olemassa MVP:n määräävä algoritmi, joka toimii ajassa

$$O(N^{2-a(k)} (\log N)^{1-a(k)}), \text{ missä } a(k) = 2^{-(k+1)},$$

kaikille kiinteille kokonaisluvuille $k \geq 3$. Hän pystyi vielä parantamaan tätä tulosta 3-ulotteisessa euklidisessa avaruudessa aikaan $O((N \log N)^{1,8})$. Yaon algoritmeilla ei liene kuitenkaan käytännön merkitystä, koska niiden implementointi on vaikeaa.

Shamos [S78] osoitti, miten jo 1-ulotteisessa avaruudessa MVP:n määrääminen on yhtä vaikeata kuin pistejoukon lajittelu. Täten moniulotteisessa avaruudessa annetun pistejoukon MVP:n määrääminen vaatii vielä paljon tutkimusta; joko Shamoksen antamaa kompleksisuuden alarajaa $\Omega(N \log N)$ tai Yaon ylärajaa on pystyttävä parantamaan.

5. KESKIMÄÄRÄISELTÄ SUORITUSAJALTAAN NOPEITA ALGORITMEJA

Edellä keskityttiin pelkästään algoritmien pahimman tapauksen suoritusajan tarkasteluun. Toisin sanoen näiden algoritmien suoritusaika ei nouse koskaan tietyn ennalta lasketun rajan yläpuolelle, saatiinpa minkälainen syöte tahansa. Käytännössä saattavat olla edullisia myös sellaiset algoritmit, joiden suoritusajan odotusarvo on pieni. Tässä kappaleessa tutustutaan nimenomaan tällaisiin MVP määräävin algoritmeihin ja niissä esiintuleviin ongelmiin.

Määrättäessä tietyn MVP:n määräävän algoritmin keskimääräistä suoritusaikaa on ensimmäiseksi kiinnitettävä niiden graafien joukko, joiden MVP:ta ollaan ratkaisemassa. Yksinkertainen ja melko luonnollinen valinta on M viivaa ($N-1 \leq M \leq N(N-1)/2$) sisältävien yhtenäisten graafien joukko. Graafin viivojen painot voivat olla jonkin jakautuman F mukaisia. Olisi siis pystyttävä määräämään, miten kauan tietyn algoritmin suoritus kestää satunnaiselle tämän joukon graafille. Graafijoukko voidaan kiinnittää myös monella muulla tavalla (ks. esimerkiksi [WS01]).

Kruskalin algoritmin suoritusaikaa voidaan huomattavasti pienentää ainakin edellä mainituilla erikoisilla graafeilla [KN82]. Tämä tehdään viivojen painojen lajittelua nopeuttamalla; lajittelu kun oli Kruskalin algoritmissa aikaavievintä. Tavallisesti Kruskalin algoritmissa kekolajittelua käyttäen otetaan yksitellen pienipainoisin viiva käsittelyyn kunnes saadaan MVP luotua. Nyt ei tehdäkään näin vaan ensin ryhmitellään viivat $\lfloor M/c \rfloor$ (c on vakio) eri lokeroon siten, että lokerossa i viivojen painot ovat pienempiä kuin lokerossa i+1. Jos oletetaan, että <u>lattiafunktion</u> arvo voidaan laskea vakioajassa, niin jokaiselle viivalle oikea lokero löytyy vakioajassa, toisin sanoen viivojen siirto lokeroihin käy ajassa O(M).

Tämän jälkeen voidaan kekolajittelua käyttää erikseen kunkin lokeron viivojen lajitteluun. Jos viivojen painot noudattavat likimäärin tasaista jakautumaa, kuhunkin lokeroon osuu melkein sama määrä viivoja. Näin yhden lokeron lajitteluun voidaan katsoa kuluvan vakioaika. Kaiken kaikkiaan viivojen lajitteluun kuluu keskimäärin O(M) aika. Aina ei tarvitse edes kaikkia lokeroita lajitella, kun MVP on jo löytynyt.

Tämän parannuksen jälkeen Kruskalin algoritmissa onkin enää selvitettävä, miten paljon kuluu aikaa UNIONI-ETSI operaatioihin, joita käytetään pidettäessä kirjaa eri MVP:n komponenteista. UNIONI-ETSI algoritmien keskimääräisen suoritusajan matemaattinen analyysi on osoittautunut hankalaksi. Yao [Y76] sekä Knuth ja Schönhage [KS78] ovat onnistuneet tässä eräissä todennäköisyysmalleissa. Tästä huolimatta on vielä avoin kysymys, saadaanko näin implementoidulla Kruskalin algoritmilla satunnaisen graafin MVP O(M) ajassa.

Jos edellisessä algoritmissa melkein kaikki graafin viivat osuvat samaan lokeroon, algoritmi toimii hitaasti ja tässä pahimmassa tapauksessa suoritusaika on O(M logM). Täten on mielenkiintoista tietää, löytyykö keskimääräiseltä suoritusajalta tehokasta algoritmia, jonka pahimman tapauksen kompleksisuus olisi parempi kuin Kruskalin algoritmilla. Katajainen [K83] esittikin erään tällaisen algoritmin, mutta se osoittautui pienille graafeille huomattavasti Kruskalin algoritmia hitaammaksi.

Bentley, Weide ja Yao [BWY80] osoittivat, miten lokerointia voidaan käyttää tehokkaasti ratkaistaessa pistejoukon MVP moniulotteisessa avaruudessa. He esittivät algoritmin, jonka suoritusaika on euklidisen tason pistejoukolle keskimäärin O(N) ja algoritmin, jonka suoritusaika on k-ulotteisen (k>2) avaruuden pistejoukolle keskimäärin O(N loglogN). He todistivat, että nämä tulokset ovat voimassa kunhan pisteiden jakautuma avaruuteen ei poikkea kovin paljon tasaisesta jakautumasta (ks. tarkemmin [BWY80]).

Viivojen painojen jakautumasta tai pisteiden jakautumasta avaruuteen joudutaan tekemään melkoisia oletuksia matemaattisen analyysin mahdollistamiseksi. Näistä rajoituksista huolimatta juuri lajittelualgoritmien yhteydessä on viimeaikoina tullut selvästi esille, että lokeroinnilla on myös käytännön merkitystä. Tässä esitettyjen algoritmien käyttökelpoisuutta puoltaa keskimääräisen suoritusajan pienuuden lisäksi implementoinnin helppous.

6. LIKIMÄÄRÄISISTÄ RATKAISUMENETELMISTÄ

Jos N on suuri, niin neliöllinen algoritmi saattaa olla yhtä käyttökelvoton kuin eksponenttiaalinen algoritmi. Tämä pätee myös Primin algoritmiin, koska se tulee hyvin hitaaksi N:n kasvaessa muutamaan tuhanteen. Joidenkin sovellutusten kannalta ei ole kuitenkaan aivan välttämätöntä, että MVP määrätään täsmällisesti. Esimerkiksi määrättäessä pistejoukon rypäät saattaa riittää, että saadaan suurin piirtein selville, mitkä pisteet kuuluvat samaan rypääseen. Juuri tällaisissa tapauksissa voi olla perusteltua käyttää jotain likimääräistä algoritmia, jos tiedetään saavutettavan huomattavia säästöjä ajoajassa. Likimääräisten algoritmien käyttö tulee kysymykseen lähinnä silloin, kun on määrättävä moniulotteisen avaruuden pisteille MVP. Tällöin voidaan käyttää heuristisesti hyväksi avaruuden ominaisuuksia.

Bentley ja Friedman [BF78] esittivät yhden tällaisen likimääräisen algoritmin, joka määrää annetuille pisteille <u>melkein</u> minimaalisen virittävän puun. Heidän heuristiikkansa on mielenkiintoinen.
Kappaleessa 4 mainittua tarkkaa MVP:n määräävää algoritmia suoritetaan siihen asti kunnes sen jatkaminen alkaa olla liian työlästä. Toisin sanoen siihen asti kunnes liian monelle alipuun pisteelle joudutaan määräämään lähin alipuun ulkopuolinen piste ennen kuin voidaan varmasti sanoa, mikä piste on alipuuta lähinnä. Tämän jälkeen otetaan alipuusta se piste, jolla etäisyys alipuun ulkopuolelle on tähän mennessä pienin. Tälle pisteelle p määrätään sitten lähin piste kyseessä olevan alipuun ulkopuolelta. Määrätään samalla tavalla näin löydetyn pisteen q lähin piste sen alipuun ulkopuolelta, johon piste kuuluu. Jos pisteen q lähin ulkopuolinen piste on p, niin viiva (p,q) liitetään MVP:n. Muuten asetetaan q pisteeksi p ja jatketaan kunnes edellä mainittu ehto on voimassa.

Tällä algoritmilla on avaruuteen normaalisti jakautuneen pistejoukon MVP pystytty määräämään alle prosentin epätarkkuudella tarkkaan algoritmiin nähden yli puolta nopeammin [BF78]. Suoritusajan on käytännössä havaittu olevan O(N logN). Valitettavasti, kuten Yao toteaa [Y82], ei ole pystytty osoittamaan, millä tarkkuudella tämä algoritmi antaa MVP:n pituuden. Täten seuraavaksi käsiteltävien todennäköisyysalgoritmien käyttö lieneekin järkevämpää, koska näille tavallisesti tiedetään, millä todennäköisyydellä algoritmi antaa ratkaisun ja millä todennäköisyydellä tämä ratkaisu on tarkka.

7. TODENNÄKÖISYYSALGORITMEISTA

Todennäköisyysalgoritmit eroavat tavallisista deterministisistä algoritmeista siten, että ohjelman sisällä määrätään satunnaisesti miten ongelma ratkaistaan. Tavallisesti oletetaan, että kaikki ratkaisupolut ovat yhtä todennäköisiä. Tällä satunnaistamisprosessilla saadaan ohjelman suoritusajan odotusarvo usein pieneksi. Todennäköisyysalgoritmeilla voidaan sallia likimääräisten algoritmien tapaan se, ettei niiden tarvitse aina tuottaa täsmälleen oikeata vastausta. Kuitenkin siinä tapauksessa, että ohjelman antama vastaus on väärä, virhe ei saa olla jotain ennalta kiinnitettyä pientä vakiota suurempi. Näin ollen todennäköisyysalgoritmi tuottaa melkein aina oikean vastauksen.

Toisaalta todennäköisyysalgoritmin ei tarvitse aina päättyä. Yleensä riittää, että ohjelma valitsee epäsuotuisan pitkän ratkaisupolun todennäköisyydellä, joka lähenee nopeasti kohti nollaa polun pituuden kasvaessa. Tällöin ohjelman suoritus loppuu todennäköisyydellä yksi ja ohjelman suoritusajan odotusarvo on edelleen lyhyt. Siis todennäköisyysalgoritmi päättyy melkein aina. Todennäköisyysalgoritmien tarkempi määrittely löytyy esimerkiksi Rabinin artikkelista [R76].

On ilahduttavaa, jos vaikeille ongelmille, erityisesti NP-vaikeille ongelmille, voidaan keksiä nopeita todennäköisyysalgoritmeja. Näin on mahdollista saada suurella todennäköisyydellä ainakin melkein oikea ratkaisu, joka muuten jäisi ehkä saamatta. Siitä huolimatta, että MVP:n määräämistä on pidetty helppona tehtävänä, on myös tälle ongelmalle kehitetty todennäköisyysalgoritmeja.

Rohlfin todennäköisyysalgoritmi [R78], joka on Sollinin algoritmin muunnelma, määrää k-ulotteisessa avaruudessa esitetyn pistejoukon MVP:n. Algoritmi etenee vaiheittain siten, että perättäisesti valitaan aina suuremmat kynnysetäisyydet δ. Kussakin vaiheessa MVP:hun lisätään uusia viivoja niin paljon kuin mahdollista. Kuitenkin vain ne viivat huomioidaan, joiden pituus on pienempi kuin δ. Aina tiettyä kynnysetäisyyttä δ kohti avaruus jaetaan hyperneliöihin, joiden sivun pituus on δ. Näitä neliöitä käyttäen muodostetaan 2^k erilaista 2δ sivuista ristikkoa [R76]. Nyt voidaan osoittaa, että kaikki sellaiset pisteparit, joiden etäisyys on pienempi tai yhtäsuuri kuin δ, osuvat jossakin ristikossa samaan ruutuun. Kynnysetäisyys δ määrätään etsimällä ensin N:n satunnaisen eri alipuihin kuuluvien pisteiden välisistä etäisyyksistä pienin. Näin yhden vaiheen aikana saadaan ainakin yhtä alipuuta kasvatettua.

Kun ristikoiden ruutujen säde saadaan pieneksi, joudutaan suorittamaan hyvin vähän lähimmän pisteen hakuja. Toisaalta taas, jos ruutujen säde tulee liian pieneksi algoritmista tulee tehoton, koska MVP kasvaa hyvin hitaasti ja joudutaan määräämään yhä uudelleen uusi kynnysetäisyys δ ja sitä vastaavat ristikot. Rohlf pyrki heuristisesti estämään, ettei kynnysetäisyys δ pääse milloinkaan liian pieneksi.

Rohlf on kokeellisesti havainnut tämän algoritmin toimivan ajassa $O(N \log\log N)$, kun pisteet ovat jakautuneet avaruuteen moniulotteisen normaali- tai tasaisen jakautuman mukaisesti. Tätä hän ei pystynyt todentamaan analyyttisesti. Tässä esitetty Rabinin ristikkorakenne [R76] ei ole ainut mahdollisuus. Yuval on esittänyt toisen ratkaisun [Yu76]. Yuvalin ratkaisu on osoittautunut kuitenkin kokeissa hieman Rabinin ratkaisua heikommaksi.

Tämän algoritmin teho ei perustu yksin todennäköisyysominaisuuteen. Fortune ja Hopcroft [FH79] osoittivat, miten oleellista tämän tyyppisissä algoritmeissa on hajautuksen käyttö. Algoritmissa joudutaan useasti määräämään mihin ristikon ruutuun kukin piste kuuluu. Hajautusta käyttäen tämä voidaan tehdä keskimäärin vakioajassa ja näin algoritmista tulee tehokas.

8. RINNAKKAISALGORITMEISTA

Monikone on tietokone, joka koostuu monesta prosessorista. Nämä prosessorit pystyvät suorittamaan useita toisistaan riippumattomia tehtäviä samanaikaisesti. Viime aikoina on monikoneissa toimivia rinnakkaisalgoritmeja tutkittu hyvin paljon, koska VLSI-teknologia on tehnyt mahdolliseksi monikoneiden valmistuksen edullisesti. Tässä kappaleessa esittellään lyhyesti minkälaisia monikoneissa toimivia MVP:n määrääviä algoritmeja on keksitty. Rinnakkaisalgoritmeista on laajemmin kerrottu esimerkiksi Kungin artikkelissa [K80].

Tähän päivään mennessä on ehditty esittää lukemattomia rinnakkaisen laskemisen malleja. Seuraavassa esimerkissä käytetään mallia, jossa 1) prosessoreilla on yhteinen muisti; 2) jokaisella prosessorilla on oma aritmeettis-looginen yksikkö, jossa kaikki aritmeettiset, Boolean ja loogiset operaatiot voidaan suorittaa yhdessä aikayksikössä, ja jossa kaikki rinnakkain suoritettavat operaatiot kestävät yhtä kauan; 3) yhden muistipaikan samanaikainen lukeminen on sallittua mutta samanaikainen kirjoittaminen ei; 4) muistioperaatioihin ja tiedon siirtoon kuluvaa aikaa ei huomioida [CLC82]. On huomattava, että tämä teoreettinen malli on ehkä hieman epärealistinen, koska käskyjen synkronointi tai muistipaikan samanaikainen lukeminen voi olla teknisesti vaikeasti toteutettavissa.

Useimmat rinnakkaisalgoritmit perustuvat Sollinin algoritmiin. Tällöin on ensimmäiseksi ratkaistava, miten määrätään samanaikaisesti alipuiden T_i ja T_k lähin alipuu. Nimittäin näiden puiden välillä voi olla kaksi samanpainoista viivaa, jolloin MVP:hun syntyy sykli, jos puu T_i yhdistetään puuhun T_k toisella ja puu T_k puuhun T_i toisella näistä viivoista. Siksi kahden puun välinen pienipainoisin viiva on määrättävä yksikäsitteisesti. Tässä toimitaan seuraavasti:
Määrättäessä puun T_i lähintä puuta valitaan se viivoista (u_1,u_2), jolle $u_1 \in T_i$ ja $u_2 \notin T_i$, joka on painoltaan minimaalinen ja joka on niin sanotun minimax-järjestyksen mukaan pienin. Viivojen välinen minimax-järjestys "<" määritellään seuraavasti:

$(u_1,u_2) < (v_1,v_2)$, jos ja vain jos

$\min\{u_1,u_2\} < \min\{v_1,v_2\}$ tai
$\min\{u_1,u_2\} = \min\{v_1,v_2\}$ ja $\max\{u_1,u_2\} < \max\{v_1,v_2\}$.

Tämän havainnon jälkeen Sollinin algoritmin implementointi N-monikoneessa, siis monikoneessa, jossa on N prosessoria, ajassa O(N logN) on melko suoraviivaista [GH77, s. 265-269]. Jokaiselle pisteelle pidetään yhteisessä muistissa tietoa siitä mihin puuhun se kuuluu. Ensin määrätään kullekin pisteelle lähin painoltaan minimaalinen, minimax-järjestyksen mukaan pienin viiva, jonka toinen päätepiste kuuluu eri alipuuhun kuin kyseessä oleva piste. Tämä tapahtuu yksinkertaisesti kulkemalla läpi yksi etäisyysmatriisin rivi. Jokainen prosessori käy läpi samanaikaisesti yhden rivin, joten tähän kuluu O(N) aika. Tämän jälkeen valitaan jokaiselle puulle lähin painoltaan minimaalinen, minimax-järjestyksen mukaan pienin viiva. Lopuksi päivitetään yhteisessä muistissa olevaa tietoa, minin puuhun kukin piste kuuluu, ja yhdistetään alipuut. Näitä toimenpiteitä joudutaan toistamaan korkeintaan logN kertaa, joten algoritmi toimii ajassa O(N logN).

Algoritmissa jokainen prosessori joutuu suorittamaan pahimmassa tapauksessa O(logN) kertaa O(N) operaatiota. Kaiken kaikkiaan voidaan joutua suorittamaan $O(N^2 \log N)$ operaatiota. Bentley [B79] onkin parantanut tätä tulosta esittämällä Primin algoritmista version, joka määrää MVP:n puurakenteisessa N/logN-monikoneessa ajassa O(N logN). Täten operaatioiden kokonaismäärä on $O(N^2)$.

Savage ja JaJa' [SJ81] esittivät, miten Sollinin algoritmi toimii N^2-monikoneessa ajassa $O(\log^2 N)$. Tässä kuten edellä tehtävän koko määrää, kuinka monta prosessoria tarvitaan. Luonnollisempi lähestymistapa olisi kuitenkin niin sanottu rajoitettu rinnakkaisuus, jolloin monikoneen prosessorien määrä kiinnitetään etukäteen. Chin, Lam ja Chen [CLC82] osoittivatkin, että on olemassa K-monikoneessa ajassa $O(N^2/K + \log^2 N)$ toimiva algoritmi. Erityisesti, jos sallitaan rajoittamaton rinnakkaisuus, algoritmin suoritusaika on $O(\log^2 N)$, kun $K = N\lceil N/\log^2 N\rceil$.

Myös muille monikone malleille on esitetty MVP:n määräämiä algoritmeja. Eräissä malleissa on luovuttu siitä oletuksesta, että prosessoreilla olisi käytössä yhteinen muisti. Tällöin jokaisella prosessorilla on oma muisti ja prosessorien välinen kommunikointi tapahtuu tietolinjoja pitkin. Tällaisissa malleissa on tavoitteena minimoida tietolinjojen määrä ja saada prosessoriverkostosta säännöllinen. Yksinkertaisin kommunikointigeometria on niin sanottu yksiulotteinen prosessorivektori. Tässä prosessorit on kytketty yhteen siten, että prosessori (reunimmaisia lukuunottamatta) pystyy siirtämään tietoa kummallakin puolella olevaan naapuriprosessoriin. Bentley ja Ottmann [BO80] osoittivat, että jo tällä yksinkertaisella N-monikoneella pystytään Sollinin algoritmi implementoimaan ajassa O(N logN) toimivaksi. Kuitenkin operaatioiden kokonaismäärä on $O(N^2 \log N)$. Toisessa ääritapauksessa kaikki prosessorit on kytketty toisiinsa. Muun muassa tällaisessa mallissa MVP:n määräämistä on tutkinut Nath ja Maheshwari [NM82].

Rinnakkaisalgoritmien alueelta löytyy vielä paljon avoimia kysymyksiä. Näistä ehkä mielenkiintoisin on se, miten pitkälle eri algoritmeissa samanaikaisuus yleensä voidaan viedä. Otetaan esimerkiksi lajittelu. Voidaanko lajittelu suorittaa N-monikoneessa ajassa O(logN)? Tällä hetkellä paras algoritmi, jolla päästään tähän aikaan käyttää N logN prosessoria [GL82, s. 54-60]. Akl [A82] esitti tähän samaan kysymykseen liittyen mielenkiintoisan tuloksen. Hän osoitti, että tietyssä mallissa tasolla olevan pistejoukon konveksi peite voidaan määrätä vakioajassa, kun käytössä on polynomiaalinen määrä prosessoreja.

9. YHTEENVETO TOISEN OSAN ARTIKKELEISTA

Tutkielman toisessa osassa tutustutaan tarkemmin kahteen kysymykseen - MVP:n määräämiseen moniulotteisessa avaruudessa ja keskimäärin nopeisiin MVP:n määräävii n algoritmeihin.

Artikkelissa (1) esitetään Bentleyn ja Friedmanin algoritmin muunnelma ja tutkitaan sen tehokkuutta kokeellisesti. Algoritmin implementoinnin yksityiskohdat esitetään artikkelissa (2). Algoritmi havaittiin keskimääräisiltä ominaisuuksiltaan hyväksi, mutta löytyy pistejoukkoja, joilla algoritmi käyttäytyy huonosti (3).

Edellinen algoritmi toimii myös Hamming-metriikassa. Täten algoritmia voidaan käyttää esimerkiksi formatoitujen tiedostojen tiivistämisessä. Artikkelissa (4) tutkitaan, onko k-d puu paras mahdollinen lähimmän pisteen hakua tukeva hakurakenne Hamming-metriikan tapauksessa. Burkhardin ja Kellerin [BK73] esittämä puurakenne osoittautuikin joskus k-d puuta paremmaksi.

Kahdessa muussa artikkelissa tutustutaan keskimääräiseltä suoritusajaltaan nopeisiin algoritmeihin. Artikkelissa (5) esitetään Kruskalin algoritmille implementointi ja osoitetaan se keskimääräisessä tapauksessa sekä teoreettisesti että kokeellisesti hyväksi. Pahimman tapauksen kompleksisuudessa on kuitenkin parantamisen varaan. Artikkelissa (6) esitetään algoritmi, jonka pahimman tapauksen kompleksisuus on Kruskalin algorimia parempi ja joka on myös keskimääräisessä tapauksessa tehokas. Käytännön kokeissa tämä algoritmi oli kuitenkin huomattavasti Kruskalin algoritmia hitaampi.

LÄHDELUETTELO[3)]

[AHU74] AHO A.V., HOPCROFT J.E., ULLMAN J.D.: The Design and Analysis of Computer Algorithms, Addison-Wesley 1974.

[A82] AKL S.G.: A constant-time parallel algorithm for computing convex hulls, BIT 22 (1982), 130-134.

AKL S.G., MEIJER H.: On the average-case complexity of "bucketing" algorithms, Journal of Algorithms 3,1 (1982), 9-13.

[B75] BENTLEY J.L.: Multidimensional binary search trees used for associative searching, Comm. ACM 18,9 (1975), 509-517.

[B79] BENTLEY J.L.: A parallel algorithm for constructing minimal spanning trees, Proc. 17th Annual Allerton Conf. on Comm., Control and Comput., 11-20, 1979.

BENTLEY J.L.: Multidimensional binary search trees in database applications, IEEE Trans. Software Engineering 5,4 (1979), 333-340.

[BF78] BENTLEY J.L., FRIEDMAN J.H.: Fast algorithms for constructing minimal spanning trees in coordinate spaces, IEEE Trans. Comput. C-27,2 (1978), 97-105.

[BO80] BENTLEY J.L., OTTMANN T.: The power of a one-dimensional vector of processors, Lect. Notes in Comput. Sci. 100: Graphtheoretic Consepts in Comput. Sci., 80-89, Springer 1980.

BENTLEY J.L., SHAMOS M.I.: Divide-and-conquer in multidimensional space, Proc. 8th Annual Symp. on Theory of Computing, 220-230, ACM 1976.

[BWY80] BENTLEY J.L., WEIDE B.W., YAO A.C.: Optimal expected-time algorithms for closest point problems, ACM Trans. Math. Software 6,4 (1980), 563-580.

[BG65] BERGE C., GHOUILA-HOURI A.: Programming, Games and Transportation Networks, Methuen 1965.

[B82] BRENNAN J.J.: Minimal spanning trees and partial sorting, Ors Res. Lett. 1,3 (1982), 113-116.

[BK73] BURKHARD W.A., KELLER R.M.: Some approaches to best-match file searching, Comm. ACM 16,4 (1973), 230-236.

[CT76] CHERITON D., TARJAN R.E.: Finding minimum spanning trees, SIAM J. Comput. 5,4 (1976), 724-742.

[CH78] CHIN F.Y., HOUCK D.: Algorithms for updating minimal spanning trees, J. Comput. System Sci. 16,3 (1978), 333-344.

[CLC82] CHIN F.Y., LAM J., CHEN I.-N.: Efficient parallel algorithms for some graph problems, Comm. ACM 25,9 (1982), 659-665.

[C75] CHRISTOFIDES N.: Graph Theory: An Algorithmic Approach, Academic Press 1975.

DENARDO E.V., FOX B.L.: Shortest-route methods: 1. reaching, pruning and buckets, Opers. Res. 27,1 (1979), 161-186.

---

3) Ilman tunnusta oleviin lähteisiin on viitattu ainoastaan toisen osan artikkeleissa.

DEVROYE L., KLINCSEK T.: Average time behavior of distributive sorting algorithms, Computing 26,1 (1981), 1-7.

DEWDNEY A.K.: Complexity of nearest neighbour searching in three and higher dimensions, Rep. #28, Dept. of Comput. Sci., Univ. of Western Ontario, London, Ontario, 1977.

DIAL R., GLOVER F., KARNEY D., KLINGMAN D.: A computational analysis of alternative algorithms and labeling techniques for finding shortest path trees, Networks 9 (1979), 215-248.

[D59] DIJKSTRA E.W.: A note on two problems in connexion with graphs, Numer. Math. 1,5 (1959), 269-271.

[E80] ERNVALL J.: On compact representation of data, Licentiate Thesis, Univ. of Turku, Finland, 1980.

ERNVALL J., KATAJAINEN J., NEVALAINEN O.: A minimal spanning tree algorithm for a point set in Euclidean space, Rep. A28, Comput. Sci., Univ. of Turku, Finland, 1980.

[EN79] ERNVALL J., NEVALAINEN O.: Compact storage schemes for formatted files by spanning trees, BIT 19,4 (1979), 463-475.

FEURZEIG W.: Algorithm 23: Math sort, Collected Algorithms from ACM.

[FH79] FORTUNE S., HOPCROFT J.E.: A note on Rabin's nearest-neighbor algorithm, Inf. Process. Lett. 8,1 (1979), 20-23.

[FJ81] FREDERICKSON G.N., JA'JA' J.: Approximation algorithms for several graph augmentation problems, SIAM J. Comput. 10,2 (1981), 270-283.

FRIEDMAN J.H., BENTLEY J.L., FINKEL R.A.: An algorithm for finding best matches in logarithmic expected time, ACM Trans. Math. Software 3,3 (1977), 209-226.

[G77] GABOW H.N.: Two algorithms for generating weighted spanning trees in order, SIAM J. Comput. 6,1 (1977), 139-150.

[GT79] GABOW H.N., TARJAN R.E.: Efficient algorithms for simple matroid intersection problems, Proc. 20th Annual Symp. on Foundations of Comput. Sci., 196-204, IEEE 1979.

[GGJ76] GAREY M.R., GRAHAM R.L., JOHNSON D.S.: Some NP-complete geometric problems, Proc. 8th Annual Symp. on Theory of Computing, 10-22, ACM 1976.

[GL82] GOLDSCHLAGER L., LISTER A.: Computer Science: A Modern Introduction, Prentice-Hall 1982.

[GH77] GOODMAN S.E., HEDETNIEMI S.T.: Introduction to the Design and Analysis of Algorithms, McGraw-Hill 1977.

[GR69] GOWER J.C., ROSS G.J.S.: Minimum spanning trees and single linkage cluster analysis, Appl. Statistics 18,1 (1969), 54-64.

[HJS80] HAYMOND R.E., JARVIS J.P., SHIER D.R.: Computational methods for minimum spanning tree problems, Techn. Rep. #354, Dept. of Math. Sci., Clemson University, 1980.

[HK70] HELD M., KARP R.M.: The travelling-saleman problem and minimum spanning trees, Ops Res. 18 (1970), 1138-1162.

[HK71] HELD M., KARP R.M.: The travelling-saleman problem and minimum spanning trees: Part II, Mathematical Programming 1,1 (1971), 6-25.

[HS78] HOROWITZ E., SAHNI S.: Fundamentals of Computer Algorithms, Pitman 1978.

[H79] HWANG F.K.: An O(n log n) algorithm for rectilinear minimal spanning trees, J. Assoc. Comput. Mach. 26,2 (1979), 177-182.

[J75] JOHNSON D.B.: Priority queues with update and finding minimum spanning trees, Infor. Process. Lett. 4,3 (1975), 53-57.

[K64] KALABA R.: Graph theory and automatic control, Applied Combinatorial Mathematics, 237-252, Wiley 1964.

[KLC77] KANG A.N.C., LEE R.C.T., CHANG C.-L., CHANG S.-K.: Storage reduction through minimal spanning trees and spanning forests, IEEE Trans. Comput. C-26,5 (1977), 425-434.

KATAJAINEN J.: An implementation of an algorithm for finding minimal spanning trees in a Euclidean coordinate space, Rep. D22, Comput. Sci., Univ. of Turku, Finland, 1981.

[K82] KATAJAINEN J.: On the worst case of a minimal spanning tree algorithm for Euclidean space, Rep. A34, Comput. Sci., Univ. of Turku, Finland, 1982.

[K83] KATAJAINEN J.: An implementation of Cheriton-Tarjan's minimal spanning tree algorithm using ordered subsets, Comput. Sci., Univ. of Turku, Finland, 1983. (käsikirjoitus)

[KN82] KATAJAINEN J., NEVALAINEN O.: An alternative for the implementation of Kruskal's minimal spanning tree algorithm. (julkaistaan lehdessä Science of Computer Programming)

[KIM81] KATOH N., IBARAKI T., MINE H.: An algorithm for finding K minimum spanning trees, SIAM J. Comput. 10,2 (1981), 247-255.

[KV72] KERSHENBAUM A., VAN SLYKE R.: Computing minimum spanning trees efficiently, Proc. ACM 25th Annual Conf., 518-527, ACM 1972.

KNUTH D.E.: The Art of Computer Programming: Vol. 1/Fundamental Algorithms, Addison-Wesley 1973.

KNUTH D.E.: The Art of Computer Programming: Vol. 3/Sorting and Searching (2nd printing), Addison-Wesley 1975.

[KS78] KNUTH D.E., SCHONHAGE A.: The expected linearity of a simple equivalence algorithm, Theoretical Comput. Sci. 6,3 (1978), 281-315.

[KMB81] KOU L., MARKOWSKY G., BERMAN L.: A fast algorithm for Steiner trees, Acta Informatica 15 (1981), 141-145.

[K56] KRUSKAL J.B.: On the shortest spanning subtree of a graph and the traveling salesman problem, Proc. Amer. Math. Soc. 7 (1956), 48-50.

[K80] KUNG H.T.: The structure of parallel algorithms, Advances in Computers, 65-112, Academic Press 1980.

[L80] LEE D.T.: Two-dimensional Voronoi diargams in the Lp-metric, J. Assoc. Comput. Mach. 27,4 (1980), 604-618.

[LSB77] LEE R.C.T., SLAGLE J.R., BLUM H.: A triangulation method for the sequential mapping of points from N-space to two-space, IEEE Trans. Comput. C-26,3 (1977), 288-292.

[LW57] LOBERMAN H., WEINBERGER A.: Formal procedures for connecting terminals with a minimum total wire length, J. Assoc. Comput. Mach. 4,4 (1957), 428-437.

MAURER H.A., OTTMANN T.: Manipulating sets of points - a survey, Graphs, Data Structures, Algorithms, 10-29, Carl Hanser 1979.

MEIJER H., AKL S.G.: The design and analysis of a new hybrid sorting algorithm, Inf. Process. Lett. 10,4 (1980), 213-218.

[NH80] NARULA S.C., HO C.A.: Degree-constrained minimum spanning tree, Comput. & Ops Res. 7,4 (1980), 239-249.

[NM82] NATH D., MAHESHWARI S.N.: Parallel algorithms for the connected components and minimal spanning tree problems, Inf. Process. Lett. 14,1 (1982), 7-11.

NEVALAINEN O., ERNVALL J.: Implementation of a distributive sorting algorithm, Comput. Sci., Univ. of Turku, Finland, 1981. (käsikirjoitus)

[NEK81] NEVALAINEN O., ERNVALL J., KATAJAINEN J.: Finding minimal spanning trees in a Euclidean coordinate space, BIT 21,1 (1981), 46-54.

NEVALAINEN O., KATAJAINEN J.: Experiments with a closest point algorithm in Hamming space, Angewandte Informatik 5 (1982), 277-281.

[OL74] OSTEEN R.E., LIN P.P.: Picture skeletons based on eccentricities of points of minimum spanning trees, SIAM J. Comput. 3,1 (1974), 23-40.

[P57] PRIM R.C.: Shortest connection networks and some generalizations, Bell System Tech. J. 36,6 (1957), 1389-1401.

[R76] RABIN M.O.: Probabilistic algorithms, Algorithms and Complexity: New Directions and Recent Results, 21-39, Academic Press 1976.

[R78] ROHLF F.J.: A probabilistic minimum spanning tree algorithm, Inf. Process. Lett. 7,1 (1978), 44-48.

[SJ81] SAVAGE J., JA'JA' J.: Fast, efficient parallel algorithms for some graph problems, SIAM J. Comput. 10,4 (1981), 682-691.

[SV75] SCHWARTZMANN D.H., VIDAL J.J.: An algorithm for determining the topological dimensionality of point clusters, IEEE Trans. Comput. C-24,12 (1975), 1175-1182.

[SPP76] SCHONHAGE A., PATERSON M., PIPPENGER N.: Finding the median, J. Comput. System Sci. 13 (1976), 184-199.

SHAMOS M.I.: Geometric complexity, Proc. 7th Annual Symp. on Theory of Computing, 224-233, ACM 1975.

[S78] SHAMOS M.I.: Computational geometry, Ph.D. Thesis, Yale Univ., New Haven, Conn., 1978.

SHAMOS M.I., HOEY D.: Closest-point problems, Proc. 16th Annual Symp. on Foundations of Comput. Sci., 151-162, IEEE 1975.

TARJAN R.E.: Efficiency of a good but not linear set union algorithm, J. Assoc. Comput. Mach. 22,2 (1975), 215-225.

[V78] VUILLEMIN J.: A data structure for manipulating priority queues, Comm. ACM 21,4 (1978), 309-315.

WEIDE B.W.: Statistical methods in algorithm design and analysis, Ph.D. Thesis, Carnegie-Mellon Univ., Pittsburgh, PA, 1978.

[W80] WEIDE B.W.: Random graphs and graph optimization problems, SIAM J. Comput. 9,3 (1980), 552-557.

[W72] WHITNEY V.K.M.: Algorithm 422: Minimal spanning tree[H], Comm. ACM 15,4 (1972), 273-274.

WIEDERHOLD G.: Database Design, McGraw-Hill 1977.

[Y75] YAO A.C.: An O(|E|loglog|V|) algorithm for finding minimum spanning trees, Inf. Process. Lett. 4,1 (1975), 21-23.

[Y76] YAO A.C.: On the average behavior of set merging algorithms, Proc. 8th Annual Symp. on Theory of Computing, 192-195, ACM 1976.

[Y82] YAO A.C.: On constructing minimum spanning trees in k-dimensional space and related problems, SIAM J. Comput. 11,4 (1982), 721-736.

[Yu76] YUVAL G.: Finding nearest neighbours, Inf. Process. Lett. 5,3 (1976), 63-65.

[Z71] ZAHN C.T.: Graph-theoretical methods for detecting and describing gestalt clusters, IEEE Trans. Comput. C-20,1 (1971), 68-86.

ZOLNOWSKY J.E.: Topics in computational geometry, Ph.D. Thesis, Stanford Univ., Stanford, Calif., 1978.