Tietojenkäsittelytiede **26** Heinäkuu 2007 sivut 54–67
Toimittaja: Antti Valmari

"**P** = **NP**" -ongelma ja laskennan vaativuusteoria*

Pekka Orponen
Teknillinen korkeakoulu
Tietojenkäsittelyteorian laboratorio
pekka.orponen@tkk.fi

Johdanto: miljoonan dollarin ongelma

Amerikkalainen *Clay Mathematics Institute* -tutkimuslaitos nimesi keväällä 2000 vuosituhannen vaihtumisen kunniaksi seitsemän avointa matemaattista ongelmaa, joista kunkin ratkaisijalle on luvassa miljoonan dollarin palkinto. Yhtenä instituutin listan [6] tehtävänä on selvittää, onko "**P** = **NP**". Listan taustamateriaalista löytyvän täsmällisen määritelmän mukaan tämä lyhyt merkintä tarkoittaa kysymystä, voidaanko jokainen "epädeterministisellä Turingin koneella polynomisessa ajassa tunnistettava kieli tunnistaa myös jollakin polynomisessa ajassa toimivalla deterministisellä Turingin koneella".

Miksi näin oudon tuntuinen kysymys on nostettu uuden vuosituhannen matemaattisten ongelmien miljoonapalkintolistalle? Mitä merkitystä on "epädeterministisillä polynomisessa ajassa toimivilla Turingin koneilla" matematiikan perustutkimuksen tai käytännön tietojenkäsittelytehtävien kannalta?

1 Työläät ongelmat

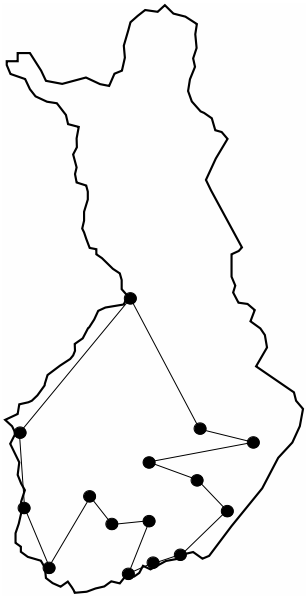
Tarkastellaan selventävänä esimerkkinä niin sanottua *kauppamatkustajan ongelmaa* (englanniksi *Traveling Salesman Problem*, TSP). Tässä ongelmassa on kuvitteelliselle kauppamatkustajalle annettu tiekartta, jolle merkittyjen n kaupungin kautta hänen tulisi löytää kokonaispituudeltaan mahdollisimman lyhyt kiertoreitti takaisin lähtökaupunkiinsa. Reitin pitää siis kulkea kunkin kartalle merkityn kaupungin kautta täsmälleen yhden kerran. Esimerkiksi kuvassa 1 on esitetty minimipituinen kauppamatkustajan reitti Suomen viidentoista suurimman kaupungin kiertämiseen. (Koko pääkaupunkiseutua on tässä pidetty yhtenä kaupunkina.) Reitin kokonaispituus maanteitä pitkin on 2205 km.[1]

Matemaattisesti ilmaistuna TSP-ongelmassa on tavoitteena löytää yleinen menetelmä tai algoritmi seuraavan laskentatehtävän ratkaisemiseen: syötteenä annetun $n \times n$ -kustannusmatriisin $D \in \mathbb{N}^{n \times n}$ ("kaupunkien etäisyystaulukon") pohjalta on määritettävä sellainen indeksien $1, \ldots, n$ permutaatio (järjestys)

*Artikkeli on alunperin kirjoitettu vuonna 2002 Tietojenkäsittelytieteen seuran toteutumatta jäänyttä 20-vuotisjuhlakirjaa varten.

[1] Kuva vastaa vuoden 2002 tilannetta. Vuoden 2006 alussa tapahtuneen kuntaliitoksen myötä Rovaniemi nousi viidentoista asukasluvultaan suurimman kaupungin joukkoon; vastaavasti Porvoo on pudonnut joukosta pois.

OHJE KIRJAPAINOLLE: B5-arkin vasen- ja yläreuna kohdistetaan A4-arkin vasempaan ja yläreunaan. Näin pitäisi marginaaliksi tulla taitteen puolella noin 33 mm ja muualla noin 22 mm.



Kuva 1: Kauppamatkustajan reitti Suomen 15 suurimmalle kaupungille (2002).

$\pi(1),\ldots,\pi(n)$, joka minimoi permutaation kokonaiskustannusta (“reitin kokonaispituutta”) kuvaavan summan

$$\sum_{i=1}^{n-1} D[\pi(i),\pi(i+1)] + D[\pi(n),\pi(1)] \; .$$

TSP-ongelma esiintyy erilaisina muunnelmina lukuisissa käytännön sovelluksissa. Edellä kuvattua konkreettista kauppamatkustajan tehtävää lähellä on esimerkiksi jakeluautojen reitin suunnittelu, mutta ongelman variaatioihin törmätään muun muassa pyrittäessä optimoimaan tietokoneen mikropiirien johdotusta, annetun työtehtävien joukon suoritusjärjestystä, geneettisten nukleotidijonojen sekvenointia ja niin edelleen [10, 15].

Millaisia ratkaisumenetelmiä tälle tärkeälle ongelmalle sitten voitaisiin löytää? Abstraktin matemaattiselta kannalta tehtävä on tavallaan triviaali, koska annetussa ongelman tapauksessa mahdollisia reittejä on vain äärellinen määrä — täsmällisemmin sanoen n kaupungin kartalla $\frac{1}{2}\cdot(n-1)! = \frac{1}{2}\cdot(n-1)\cdot(n-2)\cdot\ldots\cdot 2\cdot 1$ kappaletta. (Lähtökaupunkia seuraavan kaupungin kauppamatkustaja voi valita $n-1$ tavalla, sitä seuraavan $n-2$ tavalla ja niin edelleen. Lisäksi huomataan, että reitit, jotka kiertävät annetut kaupungit täsmälleen vastakkaisissa järjestyksissä ovat samanpituiset ja voidaan siten tämän tehtävän kannalta samaistaa.) Näin ollen voitaisiin “periaatteessa” yksinkertaisesti kokeilla järjestyksessä kaikki mahdolliset

reitit ja valita niistä lyhin. Tämä ei kuitenkaan onnistu käytännössä, sillä jo esimerkiksi 23 kaupungin kartalla mahdollisten reittien määrä on $\frac{1}{2} \cdot 22! \approx 6 \cdot 10^{20}$ kappaletta. Jos suunniteltu läpikäynti toteutettaisiin tietokoneella, joka tutkii yhden reitin millisekunnissa, kuluisi kaikkien mahdollisuuksien läpikäyntiin lähes 18 miljardia vuotta, mikä ylittää koko maailmankaikkeuden tähänastisen noin 13,7 miljardin vuoden iän!

Luonnollinen parannusidea olisi nopeuttaa tehtävän suorittamista hankkimalla tehokkaampi tietokone — vaikkapa miljoonan prosessorin rinnakkaislaitteisto, jonka kukin yksikkö tutkii yhden kauppamatkustajan reitin nanosekunnissa. Tämä laskentatehon biljoonakertaistaminen kylläkin lyhentäisi 23-kaupunkisten reittien läpikäynnin vaatiman ajan vajaaseen viikkoon, mutta ei muuten auttaisi pitkällekään, sillä tälläkään laskentateholla ei universumin ikä riittäisi kaikkien mahdollisten reittien läpikäyntiin kuin enintään 31 kaupungin kartoilla.

TSP-ongelmaan sisältyvä perusvaikeus tällaisten "raa'an voiman" ratkaisumenetelmien kannalta on vaihtoehtoisten reittien määrän *eksponentiaalinen kasvu* tai niin sanottu "kombinatorinen räjähdys", joka nopeasti syö kaiken laskentaan suoraviivaisesti sijoitetun tehonlisäyksen.

Sanotaan, että kasvava funktio $f : \mathbb{N} \to \mathbb{N}$ on *polynomisesti rajoitettu*, jos funktion $f(n)$ kasvua rajoittaa jokin argumentin n polynomi, tai yhtäpitävästi jos jollakin vakiolla c ja kaikilla n on $f(2n) \leq c \cdot f(n)$. Jos tarkasteltavan ongelmanratkaisumenetelmän aikavaativuus on polynomisesti rajoitettu, niin laskentatehon lisäämisestä on oleellista hyötyä, koska tehon c-kertaistaminen tekee mahdolliseksi käsitellä kaksi kertaa alkuperäisen kokoisia syötteitä.

Jos taas käytettävän ratkaisumenetelmän vaativuus on *eksponentiaalinen*, esimerkiksi tyyppiä c^n jollakin vakiolla $c > 1$, niin laskentatehon c-kertaistaminen vain *lisää* käsiteltävien syötteiden kokoon ykkösen. Erityisesti kauppamatkustajan ongelman tapauksessa mahdollisten reittien lukumäärä kasvaa kaupunkien määrän suhteen peräti supereksponentiaalisesti, koska kertomafunktiolle on selvästi voimassa kasvuarvio $n! \geq (n/2)^{n/2}$ ja siten millä tahansa vakiolla c on $n! \geq c^n$, kun $n \geq 2c^2$.

Laskentatehon tuloksettoman kasvattamisen sijaan parempi lähestymistapa olisi korvata eksponentiaalisen vaativa ratkaisualgoritmi paremmalla. Kauppamatkustajan ongelman osalta tällä suunnalla onkin tehty runsaasti tutkimusta, ja lukuisia heuristisia ja likimääräisiä ratkaisumenetelmiä on kehitetty: näihin kuuluvat erilaiset niin sanotut rajoiteheuristiikat, simuloitu jäähdytys, geneettiset algoritmit ja niin edelleen [1, 10]. Nämä auttavat muutamien kymmenien tai satojenkin kaupunkien tapauksiin asti — onpa näytösluontoisesti onnistuttu ratkomaan joitakin useiden tuhansienkin kaupunkien TSP-tapauksia [15] — mutta yleispätevää polynomisesti rajoitettua ratkaisumenetelmää TSP-ongelmalle ei ole vielä löydetty, ja siten jokainen tunnettu algoritmi lopulta törmää vaihtoehtoisten reittien määrän eksponentiaalisen kasvun muodostamaan läpäisemättömään esteeseen.

Koska tehokasta ratkaisumenetelmää TSP-ongelmalle ei ole löydetty, olisi tilanteen selkeyttämiseksi toivottavaa todistaa, että tällainen menetelmä on mahdoton, s.o. että mikä tahansa yleispätevä menetelmä TSP-ongelman ratkaisemiseksi vaatii kaupunkien määrän suhteen eksponentiaalisesti, tai ainakin ylipolynomisesti kasvavan suoritusajan. Merkillistä kyllä, tämänkään seikan todistaminen ei ole onnistunut 1970-luvun alkupuolelta

jatkuneesta intensiivisestä tutkimustyöstä huolimatta.

Kauppamatkustajan ongelma ei myöskään ole ainoa käytännössä merkittävä laskentatehtävä, jota tällainen epätietoisuus koskee. Itse asiassa TSP-ongelma kuuluu suureen "vaikean tuntuisten" ongelmien perheeseen, niin sanottuihin **NP**-täydellisiin ongelmiin, joilla on se hämmästyttävä ominaisuus, että ne ovat joko *kaikki* helppoja (polynomisessa ajassa ratkeavia) tai *kaikki* vaikeita, mutta ei tiedetä, kumpi vaihtoehto on tosi. Tämän tilanteen selvittäminen puoleen tai toiseen on kuuluisa "**P** = **NP**?" -ongelma.

2 Turingin koneet

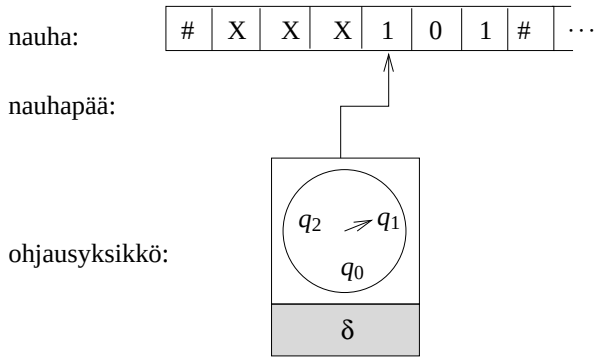
Mitä sitten ovat Clay-instituutin tehtävänmäärittelyssä mainitut "deterministiset" ja "epädeterministiset Turingin koneet" ja mitä tekemistä niillä on esimerkiksi kauppamatkustajan ongelman kanssa?

Kyse on ensinnäkin siitä, että jos halutaan todistaa täsmällisiä tuloksia jonkin laskennan mahdottomuudesta, täytyy sopia jostakin matemaattisesti eksaktista *laskennan mallista* jonka suhteen tämä mahdottomuus todistetaan. Turingin koneet ovat brittimatemaatikko Alan Turingin vuonna 1936 — siis noin kymmenen vuotta ennen ensimmäisten tietokoneiden kehittämistä! — esittelemä yksinkertainen formaali malli, jolla on edelleen tärkeä sija tietojenkäsittelyjärjestelmien teoreettisia ominaisuuksia tarkasteltaessa. Syynä mallin pysyvyyteen tietojenkäsittelymenetelmien muuten jatkuvasti muuttuessa on, että se on toisaalta riittävän pelkistetty soveltuakseen kuvaamaan päällisin puolin hyvinkin eri näköisiä tietojenkäsittelyjärjestelmiä, ja toisaalta kuitenkin riittävän voimakas kuvatakseen universaalilla ja kohtuullisen tehokkaalla tavalla kaikkea mahdollista laskentaa.

Toinen syy Turingin kone -mallin käyttöön "**P** = **NP**" -ongelman määrittelyssä on, että niin sanottujen "epädeterministisen siirtymien" mahdollisuuden lisääminen malliin osuu täsmällisesti kuvaamaan juuri sitä suuren vaihtoehtoisten ratkaisujen etsintäavaruuden aiheuttamaa laskennallista haastetta, johon TSP-ongelman kohdalla törmättiin. Tähän asiaan palataan tarkemmin seuraavissa kappaleissa.

Turingin laskentamallin lähtökohtana on perustava idea, jonka mukaan mikä tahansa säännönmukaista laskentaa suorittava järjestelmä voidaan jakaa äärelliseen "ohjausosaan" ja mahdollisesti rajoittamattoman kokoiseen "muistiin", josta ohjausosa kuitenkin pystyy käsittelemään yhdessä laskenta-askelessa vain rajallisen alueen. (Turing pohtii artikkelissaan [16] huolellisesti myös näiden oletusten filosofista perustaa, mutta tämä tarkastelu joudutaan tässä nyt sivuuttamaan.) Turing pelkistää tämän osituksen kuvan 2 mukaiseksi mekanistiseksi malliksi: laskentakoneen ohjausosa on (nykyterminologian mukaan) *äärellinen automaatti*, ja sen muisti on erillisiin merkkipaikkoihin jaettu, rajoittamattoman pituinen *nauha*, josta ohjausautomaatti pystyy koneen *nauhapään* välityksellä käsittelemään (s.o. lukemaan ja kirjoittamaan) yhtä merkkipaikkaa kerrallaan.

Ohjausautomaatilla on äärellisen monta vaihtoehtoista sisäistä tilaa q_0, q_1, $\ldots$, q_k, q_{yes}, q_{no} sekä äärellinen säännöstö tai "ohjelma" δ, joka määrittää automaatin noudattamat säännöt siirtymiselle tilasta toiseen. Säännöstö δ koostuu joukosta viisikomponenttisia jonoja $\langle q, X, q', Y, \Delta \rangle$, missä kunkin edellä mainitun tyyppisen jonon tulkinta on: "jos ohjausautomaatti tietyllä ajanhetkellä on tilassa q ja lukee nauhapään välityksellä nauhalta merkin X, niin se siirtyy seuraavalla ajanhetkellä



Kuva 2: Turingin kone.

tilaan q', kirjoittaa nauhalle merkin X tilalle merkin Y ja siirtää nauhapäätä yhden merkkipaikan suuntaan Δ." Nauhapään siirtosuunta Δ voi olla joko L (vasemmalle), R (oikealle) tai S (paikallaan).

Koneen käsiteltäväksi tarkoitettu syötemerkkijono $w = a_1 \cdots a_n$ sijoitetaan laskennan aluksi nauhan alkuun erityisillä alku- ja loppumerkeillä "#" ympäröitynä, koneen nauhapää asetetaan osoittamaan jonon ensimmäistä merkkiä a_1 (tai jos jono w on tyhjä niin suoraan loppumerkkiä #), ja ohjausautomaatti käynnistetään *alkutilassa* q_0. Tämän jälkeen kone toimii itsenäisesti ohjausautomaatin säännöstön ohjaamana, kunnes se mahdollisesti päätyy joko *hyväksyvään lopputilaan* q_{yes} tai *hylkäävään lopputilaan* q_{no}. Kummassakin tapauksessa kone pysähtyy, ja edellisessä tapauksessa se *hyväksyy*, jälkimmäisessä tapauksessa *hylkää* sille tarjotun syötteen w. On myös mahdollista, että kone jää "ikuiseen silmukkaan", s.o. ei pysähdy annetulla syötteellä. Myös tässä tapauksessa koneen tulkitaan hylkäävän syötteen.

Esimerkiksi kuvan 3 säännöstöllä varustettu Turingin kone tutkii, sisältääkö annettu 1- ja 0-merkeistä koostuva syötejono parittoman määrän 1-merkkejä. Koneen toimintaperiaate on, että se käy syötejonon läpi merkki kerrallaan vasemmalta oikealle, ja pitää ohjausautomaattinsa sisäisessä tilassa kirjaa kulloinkin nähtyjen 1-merkkien määrän pariteetista, s.o. siitä onko niitä ollut pariton (tila q_1) vai parillinen (tila q_2) määrä. Kone vaihtaa tilaansa jokaisen 1-merkin kohdalla; sen sijaan 0-merkeillä ei ole tilaan vaikutusta. Kone hyväksyy syötteen, jos se loppumerkin # kohdalle tullessaan on tilassa q_1 ja hylkää, jos se on tilassa q_2. Alkutila q_0 vastaa toiminnaltaan tilaa q_2. Syötteen yli kulkiessaan kone myös korvaa kaikki alkuperäiset ykköset ja nollat X-merkeillä; tämä piirre on tosin mukana vain esimerkin vuoksi, syötemerkit voitaisiin yhtä hyvin jättää ennalleen.

Turingin laskentamalli on siis erittäin yksinkertainen, ainakin verrattuna nykyisten tietokoneiden ja ohjelmointikielten kompleksisuuteen. Siitä huolimatta niin sanotun *Churchin–Turingin teesin* mukaan mitä tahansa fysikaalisesti toteutettavissa olevaa laskentaa voidaan jäljitellä (deterministisellä) Turingin koneella, ja niin sanotun *vahvan Churchin–Turingin teesin* mukaan tämä jäljittely aiheuttaa

$$\begin{array}{lll}
\langle q_0, 1, q_1, X, R\rangle & \langle q_1, 1, q_2, X, R\rangle & \langle q_2, 1, q_1, X, R\rangle \\
\langle q_0, 0, q_2, X, R\rangle & \langle q_1, 0, q_1, X, R\rangle & \langle q_2, 0, q_2, X, R\rangle \\
\langle q_0, \#, q_{\mathsf{no}}, \#, S\rangle & \langle q_1, \#, q_{\mathsf{yes}}, \#, S\rangle & \langle q_2, \#, q_{\mathsf{no}}, \#, S\rangle
\end{array}$$

Kuva 3: Parittomuustestaussäännöstö.

enintään polynomisen laskennan hidastumisen. (Toisin sanoen: mihin tahansa fysikaalisesti toteutettavissa olevaan laskennan malliin M liittyy sellainen polynomi $p_M(x)$, että jos jokin ongelma voidaan n kokoisilla syötteillä ratkaista mallissa M ajassa $T_M(n)$, niin se voidaan ratkaista Turingin koneella ajassa $p_M(T_M(n))$.) Siten ei teorian kannalta ole merkitystä sillä käytetäänkö laskennan mallina Turingin koneita, PC-mikroja, supertietokoneita, tai jotain vasta tulevaisuudessa keksittäviä laskulaitteita. Nämä ovat *kaikki periaatteelliselta laskentakyvyltään yhtä vahvoja.*

Churchin–Turingin teesiä ei tietenkään voida todistaa oikeaksi, koska se koskee myös menetelmiä joita ei ole vielä edes keksitty, mutta selkeää "empiiristä" näyttöä sen puolesta saatiin jo 1930-luvulla, kun useat toisistaan riippumatta kehitetyt ja päällisin puolin hyvin eri näköiset mekaanisen laskennan mallit osoittautuivatkin lähemmin tarkastellessa keskenään ekvivalenteiksi. Myöhempi kokemus oikeista fysikaalisista tietokoneista on edelleen vahvistanut teesin asemaa: mitään Turingin koneen periaatteellisen laskentakyvyn ylittävää fysikaalista laitetta ei ole näköpiirissä.

Myös vahva Churchin–Turingin teesi on pitänyt hyvin pintansa aivan viime vuosiin saakka. Nyt on tosin avautunut mielenkiintoinen uusi mahdollisuus Turingin koneiden laskentatehon ylittämiseen mikrofysikaalisten järjestelmien kvanttimekaanisia ominaisuuksia hyödyntämällä [13]. Jos tällaisia aivan uudentyyppisiä *kvanttitietokoneita* joskus pystytään todella rakentamaan, joudutaan Churchin–Turingin teesin vahvaa versiota luultavasti täsmentämään. Tosin kvanttilaskennasta saatava tehonlisäys on tietyllä tapaa rajoitettua, eikä tällä hetkellä ole selvää, missä määrin kvanttilaskentaa voitaisiin periaatteessakaan soveltaa TSP-ongelman tai muiden **NP**-täydellisten ongelmien ratkaisumenetelmien tehostamiseen. Myöskään laskettavuuden periaatteellisia rajoja kvanttitietokoneet eivät horjuta, sillä myös kaikkia kvanttilaskentoja pystytään jäljittelemään tavanomaisilla Turingin koneilla — tosin nykytietämyksen mukaan eksponentiaalisen hitaasti.

Turingin kone on *epädeterministinen*, jos johonkin ohjausautomaatin tilan q ja nauhamerkin X muodostamaan pariin liittyy useita siirtymäviisikoita $\langle q, X, q', Y', \Delta'\rangle$, $\langle q, X, q'', Y'', \Delta''\rangle$, ... Epädeterministinen Turingin kone hyväksyy annetun syötejonon, jos jokin siirtymäsääntöjen sallimista vaihtoehtoisista laskennoista johtaa sen hyväksyvään lopputilaan. Tällaista konetta ei tietenkään voida sellaisenaan toteuttaa millään tavanomaisella fysikaalisella laitteella (mistä laite tietäisi, mitä useista vaihtoehtoisista siirtymistä pitää lähteä seuraamaan?), mutta kuten seuraavissa kappaleissa todetaan, epädeterminismi on hyödyllinen apukäsite laskentaongelmien tarkastelussa.

3 Päätösongelmat ja formaalit kielet

Tarkastellaan seuraavassa yleisesti Turingin koneilla (tai muilla niitä vastaavilla laskulaitteilla, esimerkiksi nykyaikaisilla tietokoneilla) ratkottavia laskentaongelmia. Perustava ongelmatyyppi ovat tällöin *päätösongelmat*, joissa tavoitteena on yksinkertaisesti ratkaista kuuluuko annettu syöte x johonkin joukkoon A. Idea on käsitteellisesti yksinkertainen, mutta itse asiassa sopivilla syötteiden ja tulosten koodauksilla voidaan lähes minkä tahansa muunkin tyyppinen ongelma muotoilla päätösongelmana.

Yleisyyttä rajoittamatta voidaan lisäksi olettaa, että annetun ongelman ratkaisevalle Turingin koneelle tarkoitetut syötteet on aina koodattu binäärijonoiksi jollakin luonnollisella tavalla — näinhän nykyaikaisissa tietokoneissakin pohjimmiltaan kaikki tieto esitetään. Siten on esimerkiksi päätösongelmassa "onko annettu luku parillinen?" koneen tehtävänä ratkaista, kuuluuko syötteenä annettu binäärijono x joukkoon

$$EVEN = \{\, x \mid x \text{ on parillisen luvun binääriesitys} \,\}.$$

Tehtävä on tässä tapauksessa helppo, koska joukolla *EVEN* on toinen luonnehdinta:

$$EVEN = \{\, x \mid \text{binäärijonon } x \text{ viimeinen bitti on 0} \,\}.$$

Joukon *EVEN* määrittämän päätösongelman ratkaisemiseksi Turingin koneen tarvitsee siis vain etsiä syötejonon x loppukohta ja tarkastaa, onko viimeisenä sijaitseva merkki 0 vai 1. Edellisessä tapauksessa kone pysähtyy hyväksyvään, jälkimmäisessä tapauksessa hylkäävään lopputilaan.

Huomattavasti mielenkiintoisempi tehtävä on ratkaista, onko syötteenä annettu luku yhdistetty luku (ei-alkuluku), s.o. kuuluuko annettu binäärijono x joukkoon

$$COMP = \{\, x \mid \text{ on olemassa luvut } y, z, \text{ joilla } y < x \text{ ja } z < x \text{ ja } x = y \cdot z \,\}.$$

(Tässä on merkintöjen yksinkertaistamiseksi samaistettu binäärijonot ja niiden esittämät luonnolliset luvut keskenään.)

Yksinkertaisesta määrittelystään huolimatta joukon *COMP* kuvaama yhdistettyjen lukujen tunnistamisongelma on sangen tärkeä. Esimerkiksi nykyisin laajasti käytössä olevan RSA-salakirjoitusjärjestelmän turvallisuus perustuu oletukseen, että yhdistettyjen lukujen tekijöiden löytämiseen ei ole olemassa mitään (tavanomaisilla tietokoneilla toimivaa) laskenta-ajaltaan polynomisesti rajoitettua menetelmää. Yksi syy aiemmin mainittua kvanttilaskentaa kohtaan tunnettuun laajaan kiinnostukseen on Peter Shorin vuonna 1994 kehittämä algoritmi [13], jolla suurtenkin lukujen tekijöinti voitaisiin kuitenkin toteuttaa tehokkaasti kvanttifysiikan merkillisiä lainalaisuuksia hyödyntäen.

Yllättäen vuonna 2002[2] Agrawal, Kayal ja Saxena osoittivat [2], että jos eksplisiittistä tekijöihinjakoa ei vaadita, niin annetun syötejonon kuuluminen joukkoon *COMP voidaan* testata syötteen pituuden suhteen polynomisesti rajoitetussa ajassa. Agrawalin et al. menetelmän avulla voidaan siis annetusta luonnollisesta luvusta x tehokkaasti päätellä, onko se alkuluku vai yhdistetty luku, mutta jälkimmäisessä tapauksessa x:n tekijöistä ei saada (juuri) mitään tietoa.

[2] Vain muutama kuukausi tämän artikkelin alkuperäisen version valmistumisen jälkeen.

Kolmantena esimerkkinä tarkastellaan kauppamatkustajan ongelman yksinkertaistettua päätösongelmaversiota, niin sanottua Hamiltonin kehä -ongelmaa, jossa tehtävänä on ratkaista, sisältääkö annettu verkko ylipäätään yhtään sallittua kauppamatkustajan reittiä. (Oletuksena on tässä tapauksessa tietenkin, että kaikki verkon solmut tai "kaupungit" eivät ole suoraan yhteydessä toisiinsa. Muutoinhan solmut voitaisiin triviaalisti kiertää missä järjestyksessä tahansa.) Verkkoteoreettisen terminologian mukaan verkon kierrosta, joka kulkee kaikkien sen solmujen kautta täsmälleen yhden kerran, sanotaan verkon *Hamiltonin kehäksi*. Tarkasteltavaa ongelmaa vastaava joukko on siis

$$HAM = \{\, G \mid \text{binäärijonon } G \text{ esittämä verkko sisältää ainakin yhden Hamiltonin kehän} \,\}.$$

Samoin kuin täysimuotoinen TSP-ongelma, myös Hamiltonin kehä -ongelma on **NP**-täydellinen, eikä sille tunneta syötteen pituuden suhteen polynomisessa ajassa toimivaa ratkaisumenetelmää. Tässä tapauksessa myöskään kvanttilaskennan ei tiedetä auttavan ongelman ratkaisussa.

Tietojenkäsittelyteoriassa vakiintuneen terminologian mukaan mitä tahansa merkkijonoista koostuvaa joukkoa A sanotaan *(formaaliksi) kieleksi*. Turingin koneen sanotaan *tunnistavan* kielen A, jos se hyväksyy täsmälleen kieleen A kuuluvat syötteet. Annetun kielen tunnistaminen on ilmeisestikin yhtäpitävää kielen kuvaaman päätösongelman ratkaisemisen kanssa. Siten voidaan esimerkiksi puhua vaihdellen kielen *HAM* tunnistamisesta ja Hamiltonin kehä -ongelman ratkaisemisesta Turingin koneella. Seuraavassa merkitään lyhyesti annetun Turingin koneen M tunnistamaa kieltä $L(M)$:llä.

4 Laskennan aikavaativuus; vaativuusluokka P

Edellisten tarkastelujen mukaisesti olisi tärkeätä pystyä erottamaan toisistaan päätösongelmat, joilla on syötteen pituuden suhteen polynomisessa ajassa toimiva ratkaisumenetelmä ja sellaiset, joiden ratkaiseminen vaatii ylipolynomisen määrän laskenta-askelia. Vahvan Churchin–Turingin teesin mukaan voidaan yleisyyttä rajoittamatta ratkaisualgoritmien kuvausformalismina käyttää (deterministisiä) Turingin koneita. (Sivuutetaan siis toistaiseksi kvanttifysiikan mahdollisesti tulevaisuudessa tarjoamat uudet toteutustekniikat.) Tällöin voidaan myös laskennan alkeisoperaationa yksiselitteisesti pitää yhtä Turingin koneen tilasiirtymää ja mitata laskenta-aikaa tarkasteltavan Turingin koneen M *aikavaativuusfunktiolla:*

$$\text{time}_M(x) = \text{Turingin koneen } M \text{ syötteellä } x \text{ suorittamien tilasiirtymien määrä.}$$

Koska yleensä ollaan kiinnostuneita ennen muuta aikavaativuusfunktion kasvunopeudesta syötteen x pituuden $|x|$ suhteen, määritellään edelleen kaikilla luonnollisilla luvuilla n:

$$\text{time}_M(n) = \max\{\, \text{time}_M(x) : |x| = n \,\}.$$

Koneen M laskenta-aika on *polynomisesti rajoitettu*, jos jollakin polynomilla p ja kaikilla luonnollisilla luvuilla n on voimassa $\text{time}_M(n) \leq p(n)$.

Ryhmitellään sitten yhteen *vaativuusluokkaan* kaikki sellaiset formaalit kielet (päätösongelmat), joilla on jokin polynomisessa ajassa toimiva tunnistusmenetelmä:

$$\mathbf{P} = \{\, A \mid A = L(M) \text{ jollakin polynomisesti rajoitetulla Turingin koneella } M \,\}.$$

Edellä tarkastelluista päätösongelmia kuvaavista formaaleista kielistä kieli *EVEN* selvästi kuuluu luokkaan **P**, koska se voidaan tunnistaa Turingin koneella, joka tarvitsee minkä tahansa n merkin mittaisen syötteen käsittelyyn vain n tilasiirtymää. Agrawalin et al. yllättävän tuloksen mukaan myös kieli *COMP* kuuluu tähän luokkaan, joskin sen tapauksessa tunnistusalgoritmin vaativuuspolynomin asteluku on korkeampi[3] ja menetelmä perustuu edistyneisiin lukuteoreettisiin tuloksiin. Sen sijaan kielen *HAM* ei tiedetä eikä uskota sijoittuvan luokkaan **P**.

Formaalin kielen A kuuluminen luokkaan **P** voidaan periaatteessa aina todentaa esittämällä A:lle polynomisessa ajassa toimiva tunnistusalgoritmi, mutta mitään yleisesti käyttökelpoista menetelmää ei ole löydetty tällaisen algoritmin *puuttumisen* todistamiseen. Lähimmäksi tulee seuraavassa esitettävä **NP**-täydellisyyden teoria, joka on erittäin käyttökelpoinen työkalu osoittamaan, että annettu kieli on *melko varmasti* luokan **P** ulkopuolella — mutta lopullinen vahvistus jää tässäkin tapauksessa riippumaan "**P** = **NP**" -ongelman ratkaisusta.

5 Arvaus–tarkastus -ongelmat; vaativuusluokka NP

NP-täydellisyysteorian perustana on se merkittävä havainto, että monet luonnollisia päätösongelmia kuvaavat formaalit kielet A, joiden ei tiedetä kuuluvan luokkaan **P**, ovat seuraavaa "arvaus–tarkastus" -muotoa: syötteen x kuuluminen kieleen A voitaisiin tarkastaa helposti, jos tunnettaisiin (tai osattaisiin "arvata") tietty enintään polynomisen pituinen "varmistejono" w. Tämänmuotoisen ongelman vaikeus kiteytyy siihen, että mahdollisia varmistejonoja on liian paljon suoraviivaisesti järjestyksessä kokeiltaviksi: jos binäärisen varmistejonon pituutta rajoittaa polynomi p, niin n-merkkisellä syötteellä x tutkittavia vaihtoehtoisia jonoja tulee $2^{p(n)}$ kappaletta.

Esimerkiksi Hamiltonin kehä -ongelma (kieli *HAM*) on arvaus–tarkastus -tyyppinen. Jos syötteenä annetun verkon G solmuille osataan arvata oikea järjestys w, niin on helppo tarkastaa, että ne tässä järjestyksessä määrittelevät Hamiltonin kehän. Jos verkossa G on n solmua, niin minkä tahansa järjestyksen esittämiseen tarvitaan vain noin $n \cdot \log_2 n$ bittiä, mutta vaihtoehtoisia järjestyksiä on kaikkiaan $\frac{1}{2} \cdot (n-1)!$ kappaletta, kuten TSP-ongelman yhteydessä aiemmin todettiin.

Samoin yhdistettyjen lukujen tunnistamisongelma (kieli *COMP*) on tällaista tyyppiä, sillä vaikka annetun luvun x mahdollisten tekijöiden löytäminen voi olla hankalaa, niin jos tekijät y ja z on valmiiksi annettu, niiden oikeellisuus voidaan selvittää yksinkertaisella kertolaskulla.[4]

Voidaan osoittaa, että kielen A arvaus–tarkastus -ominaisuus on yhtäpitävä sen kanssa, että A voidaan tunnistaa jollakin polynomisessa ajassa toimivalla *epädeterministisellä* Turingin koneella. Kuten muistetaan, epädeterministisellä koneella voi olla annetussa laskennan tilanteessa useita mahdollisia jatkovaihtoehtoja: tällaisen koneen aikavaativuus syötteellä x

[3] Agrawalin et al. alkuperäisen algoritmin aikavaativuus oli luokkaa n^{12}, mutta sittemmin menetelmästä on kehitetty jo luokkaa n^6 oleva versio, ja vaativuuspolynomin asteluku tullee pienenemään tästä edelleen.

[4] Korostettakoon vielä kerran, että vaikka *COMP-ongelma* määrittelynsä mukaan onkin arvaus-tarkastus -tyyppinen, niin Agrawalin et al. ratkaisu*algoritmi* ei perustu syöteluvun x tekijöiden etsimiseen.

määritellään *lyhimmän* hyväksyvään lopputilaan johtavan laskennan pituuden mukaan, tai jos tällaista ei ole niin pisimmän hylkäävän laskennan pituuden mukaan. Mallissa ei siten laskuteta lainkaan rinnakkaisten laskentapolkujen tuomasta lisäpotentiaalista. Tämä on tietenkin sinänsä täysin epärealistista, mutta luonnehtii täsmälleen sen eron, mikä on tavanomaisen deterministisen laskennan ja edellä mainitun tyyppisen arvaus–tarkastus -laskennan välillä. Vaihtoehtoisten arvausjonojen muodostaminen vastaa epädeterministisen koneen epädeterministisiä siirtymiä, joista otetaan laskennan vaativuutta määritettäessä huomioon vain yksi "paras" vaihtoehto.

Tämän vastaavuuden takia arvaus–tarkastus -tyyppisiä ongelmia sanotaan myös **NP**-tyyppisiksi (englanniksi "nondeterministic polynomial time"). Näitä luonnehtiva vaativuusluokka voidaan siis määritellä:

$$\mathbf{NP} = \{\, A \mid A = L(N) \text{ jollakin polyn. rajoitetulla epädeterministisellä Turingin koneella } N \,\}.$$

Vaihtoehtoinen luonnehdinta on todeta formaalin kielen A kuuluvan luokkaan **NP**, jos ja vain jos on olemassa sellaiset "varmistejonon" w pituutta rajoittava polynomi p ja polynomisessa ajassa toimiva deterministinen Turingin "tarkastuskone" M, joilla on voimassa kaikilla syötteillä x:

$$x \in \mathrm{A} \;\Leftrightarrow\; (\exists w : |w| \leq p(|x|) \text{ ja } \langle x, w\rangle \in L(M)).$$

Huomataan, että triviaalisti on $\mathbf{P} \subseteq \mathbf{NP}$, s.o. jokainen polynomisessa ajassa ratkeava päätösongelma kuuluu myös luokkaan **NP**, koska deterministiset Turingin koneet ovat epädeterminististen koneiden erikoistapaus.

NP-tyyppisiä ongelmia esiintyy lukuisissa käytännön sovelluksissa. Seuraavassa on lueteltu muutamia tunnettuja esimerkkejä tällaisista ongelmista, joiden ei tiedetä (eikä uskota) kuuluvan luokkaan **P**. Gareyn ja Johnsonin [8] nyt jo klassinen teos listaa kolmisensataa muutakin samansukuista ongelmaa, ja tuon teoksen ilmestymisen jälkeen on kirjallisuudessa esiintynyt vielä toistatuhatta vastaavaa ongelmaa lisää.

1. *Lausekalkyylin toteutuvuusongelma (SAT):* Annettu binääriarvoisista "propositiomuuttujista" $x_1, \ldots, x_n$, vakioista 1 ("tosi") ja 0 ("epätosi") sekä konnektiiveista $\wedge$ ("ja"), $\vee$ ("tai") ja $\neg$ ("ei") koostuva lausekalkyylin kaava $F = F(x_1, \ldots, x_n)$. On ratkaistava, onko F *toteutuva*, s.o. onko olemassa sellaista muuttujien $x_1, \ldots, x_n$ totuusarvojakelua $t : \{x_1, \ldots, x_n\} \to \{0,1\}$, joka tekee F:stä toden: $F(t(x_1), \ldots, t(x_n)) = 1$.

   Esimerkiksi kaavan $F = (x_1 \wedge (\neg x_1 \vee \neg x_2))$ toteuttaa totuusarvojakelu $t(x_1) = 1$, $t(x_2) = 0$. Sen sijaan kaava $G = (x_1 \wedge \neg x_1)$ ei toteudu millään totuusarvojakelulla.

2. *Kauppamatkustajan päätösongelma (TSP):* Annettu täydellinen painotettu verkko $\langle G, d\rangle$ ja kokonaisluku k, siis "kartta, etäisyystaulukko ja reitin maksimipituus". Ratkaistava, sisältääkö G sellaisen Hamiltonin kehän, jonka pituus on enintään k.

3. *Solmupeiteongelma (VC):* Annettu verkko G ja kokonaisluku k. Ratkaistava, sisältääkö G enintään k solmun muodostamaa *solmupeitettä*, s.o. solmujoukkoa, joka peittää jokaisesta verkon kaaresta ainakin toisen pään (katso kuva 4).

4. *Riippumaton joukko -ongelma (IS):* Annettu verkko G ja kokonaisluku k. Ratkaistava, sisältääkö G vähintään k *riippumatonta solmua*, s.o. solmua, joiden välillä ei ole yhtään verkon kaarta (katso kuva 4).

5. *Klikkiongelma (CLIQUE):* Annettu verkko G ja kokonaisluku k. Ratkaistava, sisältääkö G vähintään k solmun muodostamaa *klikkiä*, s.o. solmujoukkoa, jonka kaikkien solmuparien välillä on kaari (katso kuva 4).

6 Polynomiset ongelmamuunnokset ("palautukset")

Tärkeä teema laskennan vaativuusteoriassa on ongelmien vaikeusvertailu. Yksi tapa todeta, että ongelma A on "helpompi" kuin ongelma B on osoittaa, että A on B:n "erikoistapaus". Tällöinhän jos ongelma B osattaisiin ratkaista, niin B:n ratkaisua muuntamalla saataisiin ratkaisu myös A:lle. A:n toteaminen B:n erikoistapaukseksi voi tietenkin yleisessä tapauksessa vaatia syötteiden muokkaamista jollakin lailla. Täsmällinen määritelmä on seuraava: sanotaan, että formaali kieli A voidaan *palauttaa (polynomisessa ajassa)* formaaliin kieleen B ja merkitään $A \leq_m^p B$, jos on olemassa sellainen syötteen pituuden suhteen polynomisessa ajassa laskettava funktio f, että kaikilla syötteillä x on voimassa:

$$x \in A \quad \Leftrightarrow \quad f(x) \in B.$$

Polynominen palautusfunktio f siis muuntaa päätösongelman A syötteet x päätösongelman B "vastaaviksi" s.o. vastaukseltaan samoiksi, syötteiksi $f(x)$. (Tarkkaan ottaen aiemmin ei ole määritelty, miten funktioiden laskeminen toteutetaan Turingin koneilla, mutta tämä laajennus edellyttää vain jotain käytäntöä siitä, miten funktion arvo $f(x)$ esitetään koneen nauhalla.)

Aiemman tarkastelun pohjalta on helppo todistaa seuraava palautusmuunnosten perusominaisuus:

Lemma 1 *Jos $A \leq_m^p B$ ja $B \in \mathbf{P}$, niin $A \in \mathbf{P}$.* □

Konkreettisena esimerkkinä palautusmuunnoksista voidaan melko helposti todeta, että edellä kuvattujen solmupeite-, riippumaton joukko- ja klikkiongelmien välillä vallitsee suhde

$$VC \leq_m^p IS \leq_m^p CLIQUE.$$

Tämän väitteen todistus perustuu seuraavaan yksinkertaiseen aputulokseen (vertaa kuva 4):

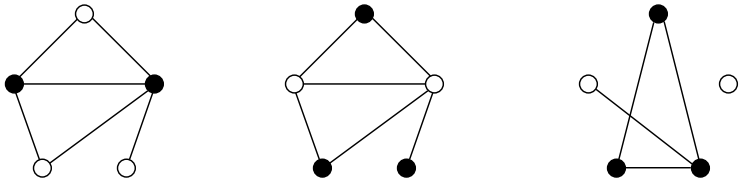
Lemma 2 *Olkoon $G = (V,E)$ suuntaamaton verkko ja $V' \subseteq V$ kokoelma sen solmuja. Tällöin seuraavat ehdot ovat keskenään yhtäpitäviä:*

(i) *V' on G:n solmupeite;*

(ii) *$V - V'$ on G:n riippumaton solmujoukko;*

(iii) *$V - V'$ on klikki G:n* komplementtiverkossa *G', jossa on samat solmut kuin G:ssä, mutta kaaret juuri niiden solmujen välillä, joiden välillä verkossa G ei ole kaarta.* □

Todistetaan tämän lemman avulla esimerkiksi väite $VC \leq_m^p IS$. Olkoon annettuna VC-ongelman syötetapaus $\langle G,k \rangle$, s.o. tehtävänä on ratkaista, sisältääkö verkko G enintään k solmun solmupeitettä. Merkitään verkon G solmujen määrää $|G|$:llä. Lemman 2 mukaan G:ssä on enintään k



Kuva 4: Solmupeite-, riippumaton joukko- ja klikkiongelman tapaukset.

solmun solmupeite, jos ja vain jos siinä on vähintään $|G| - k$ solmun riippumaton joukko. Palautusfunktioksi voidaan näin ollen valita (G:stä ja k:sta riippumatta):

$$f(\langle G,k\rangle) \;=\; \langle G, |G| - k\rangle.$$

Tämä voidaan helposti laskea polynomisessa ajassa ja täyttää kaikilla G ja k palautusehdon:

$$\langle G,k\rangle \in VC \;\;\Leftrightarrow\;\; f(\langle G,k\rangle) = \langle G, |G| - k\rangle \in IS. \qquad \square$$

7 NP-täydelliset ongelmat

Palautusmuunnosten määrittelemää ongelmien vaikeusjärjestystä hyväksi käyttäen voidaan määritellä ongelmaluokassa **NP** "maksimaalisen vaikean" päätösongelman käsite. Täsmällisesti sanoen määritellään, että päätösongelma (formaali kieli) B on **NP**-*täydellinen*, jos:

(i) $B \in \mathbf{NP}$ ja

(ii) $A \leq_m^p B$ kaikilla $A \in \mathbf{NP}$.

Ongelman B **NP**-täydellisyys on vahva argumentti sen vaikeuden puolesta, sillä tällöin B sisältää erikoistapauksinaan kaikki muut **NP**-tyyppiset ongelmat, ja siten on "ainakin niin vaikea kuin mikä tahansa muu **NP**-ongelma". Edellisen lemman 1 ja **NP**-täydellisyyden määritelmän pohjalta on helppo todistaa seuraava aputulos:

Lemma 3 *Olkoon B jokin* **NP**-*täydellinen kieli. Jos $B \in \mathbf{P}$, niin $\mathbf{P} = \mathbf{NP}$.* $\square$

Toisin sanoen: jos jollekin **NP**-täydelliselle ongelmalle onnistuttaisiin löytämään polynominen ratkaisumenetelmä, niin sen johdannaisina saataisiin polynomiset ratkaisumenetelmät *kaikille* **NP**-tyyppisille ongelmille. Tai kääntäen: jos luokassa **NP** ylipäätään on ongelmia, jotka eivät ole polynomisessa ajassa ratkeavia, niin **NP**-täydelliset ongelmat ovat tällaisia. Näin vahvan ominaisuuden voisi arvella olevan harvinainen, mutta asia onkin aivan päinvastoin: lähes kaikki luonnolliset **NP**-tyyppiset ongelmat, joille ei ole löydetty polynomista ratkaisumenetelmää, ovat osoittautuneet **NP**-täydellisiksi.

Lause 4 *Seuraavat päätösongelmat (oikeammin vastaavat formaalit kielet) ovat* **NP**-*täydellisiä:*

(1) lausekalkyylin toteutuvuusongelma (SAT),

(2) Hamiltonin kehä -ongelma (HAM),

(3) kauppamatkustajan päätösongelma (TSP),

(4) riippumaton joukko -ongelma (IS),

(5) klikkiongelma (CLIQUE),

(6) solmupeiteongelma (VC),

(7) – (1506) noin 1500 muuta tunnettua ongelmaa. □

Kaikki nämä ongelmat ovat siis sopivilla muunnoksilla tulkittavissa toistensa erikoistapauksiksi, tai saman "ideaalisen" **NP**-täydellisen ongelman eri ilmenemismuodoiksi. Yksi harvoista **NP**-tyyppisistä ongelmista, jonka ei toistaiseksi tiedetä kuuluvan luokkaan **P** eikä myöskään tiedetä olevan **NP**-täydellinen, on niin sanottu *verkkoisomorfiaongelma*, jossa tehtävänä on ratkaista ovatko kaksi annettua verkkoa solmujen nimentää vaille samanlaiset. Tämän sinänsä tärkeän ongelman ja sen muunnelmien lisäksi muita luonnollisia ehdokkaita tällaisiksi "**NP**-epätäydellisiksi" ongelmiksi ei juuri ole, vaikka teoreettisesti voidaankin osoittaa [9], että jos $\mathbf{P} \neq \mathbf{NP}$, niin ongelmaluokalla $\mathbf{NP} - \mathbf{P}$ on palautusjärjestyksen $\leq_m^p$ suhteen hyvin rikas sisäinen rakenne.[5]

Lauseen 4 todistus pohjautuu suurimmaksi osaksi seuraavaan helppoon aputulokseen:

Lemma 5 *Olkoon A jokin* **NP***-täydellinen kieli, $B \in \mathbf{NP}$ ja $A \leq_m^p B$. Tällöin myös kieli B on* **NP***-täydellinen.* □

Päättelyketjun perustaksi tarvitaan kuitenkin yksi ensimmäinen ("geneerinen") **NP**-täydellinen ongelma. Sellaisen löysi Stephen Cook vuonna 1971 ja riippumattomasti Leonid Levin 1973:

Lause 6 *Lausekalkyylin toteutuvuusongelmaa kuvaava kieli SAT on* **NP***-täydellinen.* □

Tämän **NP**-täydellisyysteorian peruslauseen todistus perustuu sangen mielenkiintoiseen yleiseen koodaustekniikkaan, jolla annetusta epädeterministisen, polynomisesti rajoitetun Turingin koneen N syötteestä x muodostetaan lausekalkyylin kaava $F = F_x^{(N)}$, jonka toteuttavat totuusarvojakelut vastaavat täsmälleen koneen N hyväksyviä laskentoja syötteellä x. Erityisesti siis kaava F on toteutuva ($F \in SAT$), jos ja vain jos *jokin* N:n laskenta hyväksyy x:n ($x \in L(N)$). Todistuksen yksityiskohtaiseen läpikäyntiin ei tässä esityksessä ole mahdollisuuksia, mutta kiinnostunut lukija löytää sen mistä tahansa oheisessa kirjallisuusluettelossa mainitusta vaativuusteorian oppikirjasta.

Kirjallisuus

Tällä hetkellä tarjolla olevasta kirjallisuudesta helpoimmin lähestyttävä johdatus vaativuusteorian käsitteistöön lienee Bovetin ja Crescenzin kirja [5]. Myös Sipserin [14] yleisluontoisemmassa tietojenkäsittelyteorian perusoppikirjassa on hyvä johdatus aiheeseen. Edistyneempiä oppikirjoja ovat muun muassa Balcázar et al. [4], Du ja Ko [7], Papadimitriou [12] sekä Wegener [17]. Myös Atallahin [3] ja van Leeuwenin [11] laaja-alaiset käsikirjat sisältävät paljon vaativuusteorian aineistoa.

Kiitokset

Kiitän Tietojenkäsittelytiede-lehden toimituskuntaa, erityisesti prof. Antti Valmaria, tämän kirjoituksen kannustavasta, huolellisesta ja vaivaa säästämättömästä toimitustyöstä sekä useista tekstin selkeyttä ja ajantasaisuutta koskevista parannusehdotuksista.

Viitteet

[1] Emile H. L. Aarts, Jan Karel Lenstra

[5] Vuoteen 2002 asti myös yhdistettyjen lukujen tunnistamisongelmaa pidettiin hyvänä **NP**-epätäydellisyysehdokkaana. Nyt siis kuitenkin tiedetään, että se kuuluukin luokkaan **P**.

(toim.), *Local Search in Combinatorial Optimization.* John Wiley & Sons, Chichester, 1997.

[2] Manindra Agrawal, Neeraj Kayal, Nitin Saxena, "Primes is in P". *Annals of Mathematics 160* (2004), 781–793. Esipainos: http://www.cse.iitk.ac.in/primality.pdf, 6.8.2002.

[3] Mikhail J. Atallah (toim.), *Algorithms and Theory of Computation Handbook.* CRC Press, Boca Raton, FL 1999.

[4] José Luis Balcázar, Josep Díaz, Joaquim Gabarró, *Structural Complexity I–II.* Springer-Verlag, Berlin Heidelberg 1988–1990. (2nd Ed. of Vol. I 1995).

[5] Daniel Pierre Bovet, Pierluigi Crescenzi, *Introduction to the Theory of Complexity.* Prentice-Hall, Hemel Hempstead 1994. Verkkoversio: http://piluc.dsi.unifi.it/tau/uploads/Publications/itc.pdf.

[6] Clay Mathematics Institute, "Millennium Problems". http://www.claymath.org/millennium/, 24.5.2000.

[7] Ding-Zhu Du, Ker-I Ko, *Theory of Computational Complexity.* John Wiley & Sons, New York, NY 2000.

[8] Michael R. Garey, David S. Johnson, *Computers and Intractability: A Guide to the Theory of* **NP***-Completeness.* W. H. Freeman, San Francisco, CA 1979.

[9] Richard E. Ladner, "On the structure of polynomial-time reducibility". *J. Assoc. Comput. Mach. 22* (1975), 155–171.

[10] Eugene L. Lawler, Jan Karel Lenstra, A. H. G. Rinnooy Kan, David B. Shmoys (toim.), *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization.* John Wiley & Sons, Chichester 1985.

[11] Jan van Leeuwen (toim.), *Handbook of Theoretical Computer Science, Vol. A: Algorithms and Complexity.* Elsevier, Amsterdam 1990.

[12] Christos Papadimitriou, *Computational Complexity.* Addison-Wesley, Reading, MA 1994.

[13] Peter W. Shor, "Polynomial time algorithms for prime factorization and discrete logarithms on a quantum computer". *SIAM J. Computing 26* (1997), 1484–1509. Konferenssiversio: Peter W. Shor, "Algorithms for quantum computation: discrete log and factoring". *Proceedings, 35th Annual IEEE Symposium on the Foundations of Computer Science,* 124–134. IEEE Press, New York, NY 1994.

[14] Michael Sipser, *Introduction to the Theory of Computation, 2nd Ed.* Thomson Course Technology, Boston, MA 2005.

[15] "Traveling Salesman Problem". http://www.tsp.gatech.edu/.

[16] Alan Turing, "On computable numbers, with an application to the Entscheidungsproblem". *Proc. London Math. Soc., Ser. 2, 42* (1936–37), 230–265; correction *ibid. 43* (1937), 544–546.

[17] Ingo Wegener, *Complexity Theory: Exploring the Limits of Efficient Algorithms.* Springer-Verlag, Berlin Heidelberg 2005.