Laskentaongelmien yksittäistapausten vaativuudesta

Pekka Orponen
Helsingin yliopisto, tietojenkäsittelytieteen laitos
PL 26, 00014 Helsingin Yliopisto
orponen@cs.helsinki.fi

1. Taustaa

Laskentaongelmalla tarkoitetaan mitä tahansa sellaista tietokoneella ratkaistavaksi tarkoitettua tehtävää, jossa annettuun *syötteeseen* on liitettävä sitä tietyn säännön mukaan vastaava *tulos*. Syötteet ja tulokset ajatellaan koodatuiksi jonkin aakkoston, tavallisimmin binääriaakkoston $\{0,1\}$ merkkijonoiksi.

Yksinkertainen esimerkki laskentaongelmasta on vaikkapa kokonaislukujen yhteenlasku: syötteinä annetaan kaksi binäärijonoina esitettyä kokonaislukua, ja tuloksena tulee olla näiden lukujen binäärimuodossa esitetty summa. Mielenkiintoisempi, ja huomattavasti vaikeampi laskentaongelma on ns. *lausekalkyylin toteutuvuusongelma*: syötteenä annetaan jokin sopivalla tavalla binäärijonoksi koodattu lausekalkyylin kaava φ, ja tehtävänä on ratkaista voidaanko kaavassa esiintyville vapaille muuttujille asettaa totuusarvot niin, että koko kaavasta tulee tosi[1]. Esimerkiksi kaavoista $x \wedge y$, $(x \vee \neg y) \wedge (\neg x \vee y)$ ja $x \wedge \neg x$ ensimmäinen ja toinen ovat toteutuvia (asetuksella $x = y = T$), kolmas ei.

Yleispätevää, mekaanisesti toteutettavissa olevaa annetun laskentaongelman ratkaisumenetelmää sanotaan sen *ratkaisualgoritmiksi*. Tietyllä laskentaongelmalla voi olla useita, hyvinkin erilaisia ratkaisualgoritmeja: esimerkiksi muistinumeroa voidaan yhteenlaskussa kuljettaa eri tavoin[2]. Myös toteutuvuusongelman ratkaisua voidaan lähestyä monin eri tavoin, joita ei kuitenkaan tässä ole mahdollista esitellä laajemmin

[1] Toteutuvuusongelma on esimerkki *päätösongelmasta*, jossa tehtävänä on ratkaista, onko syötteenä annetulla binäärijonolla jokin ominaisuus Π; kuhunkin syötteeseen liittyvä tulos on tällöin yksinkertaisesti joko "kyllä" tai "ei".

[2] Yhteenlaskun toteutustavalla on merkitystä esimerkiksi nopeita aritmetiikkapiirejä suunniteltaessa: tavanomaista "peruskoulumenetelmää" käyttäen kahden n-bittisen luvun yhteenlasku vaatii noin n aikayksikköä, mutta järjestelemällä alkeisoperaatioita sopivasti rinnakkain suoritettaviksi voidaan koko laskutoimitus saada valmiiksi noin $\log_2 n$ aikayksikössä [22, ss. 39–50].

On tärkeää huomata, että laskentaongelmaan kuuluu tyypillisesti useita, yleensä ääretön määrä *tapauksia*, so. mahdollisia syötteitä, ja ongelman ratkaisualgoritmi on menetelmä, joka kussakin ongelman tapauksessa tuottaa sitä vastaavan oikean tuloksen. Esimerkiksi toteutuvuusongelman tapauksia ovat kaikki (binäärijonoiksi koodatut) lausekalkyylin kaavat φ, ja ongelman ratkaisualgoritmi on menetelmä M, joka jokaisella kaavalla φ tuottaa tuloksen 1 tai 0, siten että $M(\varphi) = 1$ jos ja vain jos kaava φ on toteutuva.

Tietojenkäsittelyteoriassa ei yleensä tarkastella ongelmien yksittäistapauksia, so. yksittäisiä syöte–tulospareja, koska näistä tuntuu olevan vaikea sanoa mitään mielenkiintoista: jos algoritmin M halutaan tietyllä syötteellä x_0 tuottavan tuloksen y_0, niin melkein millä tahansa ohjelmointikielellä toteutettaessa algoritmiin voidaan ohjelmoida *taulukkohaku*, joka tutkii onko käsillä oleva syöte juuri x_0, ja jos on, niin tuottaa tuloksen y_0. Siten näyttäisi, että mihin tahansa yksittäiseen syötteeseen (tai yleisemmin äärelliseen joukkoon syötteitä) voidaan triviaalisti liittää mikä tahansa tulos (mitkä tahansa tulokset), ja kysymyksenasettelu muuttuu kiinnostavaksi vasta, kun algoritmin täytyy toimia oikein *äärettömällä joukolla* syötteitä.

Erityisesti laskentaongelman vaikeutta t. *laskennallista vaativuutta* tarkastellaan tavallisesti ongelman "asymptoottisena" ominaisuutena: tutkitaan, miten monta alkeisoperaatiota ongelman parhaankin ratkaisualgoritmin vähintään täytyy suorittaa n merkin mittaisilla syötemerkkijonoilla, kun n kasvaa rajatta. Esimerkiksi yhteenlaskuongelma on vaativuudeltaan vain lineaarinen syötteen pituuden suhteen, sillä peruskoulualgoritmi selviää kahden n-bittisen syötteen yhteenlaskusta noin n:llä alkeisoperaatiolla. (Tarkemmin sanoen tarvittavien alkeisoperaatioiden määrä on enintään cn, missä kerroin c on jokin n:stä riippumaton vakio.) Toisaalta lausekalkyylin toteutuvuusongelmalle ei tunneta mitään sellaista ratkaisumenetelmää, jonka ajantarvetta rajoittaisi jokin syötekaavan pituuden polynomi, ja yleisesti arvellaan ettei tällaista polynomista algoritmia tälle ongelmalle ole edes mahdollista löytää. Jos tämä arvelu (joka on yksi muoto kuuluisasta $P \neq NP$ -hypoteesista [7]) pitää paikkansa, voidaan siis sanoa, että toteutuvuusongelma on (asymptoottiselta) vaativuudeltaan ylipolynominen.

Selvästikään tämänkaltainen asymptoottinen teoria ei pysty sanomaan mitään ongelmien yksittäistapausten vaikeudesta: esimerkiksi minkä tahansa yksittäisen kaavan φ_0 toteutuvuus voidaan ratkaista sen pituuden suhteen lineaarisessa ajassa erikoistapausalgoritmilla, joka taulukoi tiedon juuri *tämän* kaavan toteutuvuudesta. Vaikeaksi toteutuvuusongelman tekee se, että täydellisen ratkaisualgoritmin täytyy selvitä *kaikista mahdollisista* syötekaavoista.

Kaikesta huolimatta monissa käytännön tilanteissa oltaisiin silti kiinnostuneita arvioimaan laskentaongelmien vaikeutta juuri tietyissä yksittäistapauksissa: tekoälysovelluksissa haluttaisiin tietää, kuinka vaikeaa jonkin *tietyn* lausekalkyylin kaavan toteutuvuuden testaaminen on, ei niinkään miten vaikeaa toteutuvuuden testaaminen on yleensä; shakkipelissä haluttaisiin tie-

tää, miten vaikea on löytää valkoisen voittostrategia (jos sellainen on) 8×8-laudalla, ei yleisellä $n \times n$ -laudalla; ja salakirjoitussovelluksissa haluttaisiin tietää, miten turvallinen (so. vaikeasti murrettavissa) tietty käsillä oleva salakirjoitusavain on, ei miten turvallisia avaimet ovat asymptoottisesti. Jos käytettävä ratkaisualgoritmi M on kiinnitetty, voidaan toki tarkastella yksittäisten tapausten vaikeutta suhteessa algoritmiin M; mutta pulmallista on, miten määritellä ratkaisualgoritmista *riippumaton* yksittäisen tapauksen vaikeuden käsite?

Yhden lähestymistavan tähän määrittelyongelmaan tarjoaa Chaitinin [4, 5], Kolmogorovin [13] ja Solomonoffin [21] esittämä *merkkijonokompleksisuuden* idea (ks. myös [3, 15]). Chaitinin ym. ajatuksena on mitata yksittäisen merkkijonon monimutkaisuutta pienimmän sen tuottamiseen kykenevän ohjelman koolla. Osoittautuu, että laskentaongelman yksittäistapauksen vaikeutta voidaan mitata samaan tapaan *pienimmän sellaisen ohjelman koolla, joka ratkaisee tämän ongelman tapauksen oikein, eikä tee virheitä muissakaan tapauksissa* (mutta saattaa joskus jättää tuloksen avoimeksi). Tämän idean pohjalle rakentuva yksittäistapausvaativuuden teoria, jota seuraavassa esitellään lähemmin, on teoreettisesti melko tyydyttävä: määritelmä on luonteva ja saavutetut tulokset ovat hyvin sopusoinnussa intuitiivisten ennakkoodotusten kanssa, olematta kuitenkaan triviaaleja (ks. [12, 18, 2, 6, 11, 14]).

Käytännön sovellusten kannalta teorialla tosin on sama puute kuin merkkijonokompleksisuudellakin: konkreettisten syötteiden yksittäistapausvaativuuden numeerinen määrittäminen voi olla hyvin vaikeaa tai peräti mahdotonta. Toisaalta merkkijonokompleksisuuden tarkasteleminen rajoitetuissa malliperheissä on johtanut käytännössäkin hyvin merkittävään ns. Rissasen MDL-periaatteeseen [19], ja jokin samaan tapaan rajoitettu versio yksittäistapausvaativuudestakin voisi olla myös käytännössä hyödyllinen.

1. Peruskäsitteitä

Rajoitutaan yksinkertaisuuden vuoksi tarkastelemaan päätösongelmia, ja oletetaan että kaikki käsiteltävät tietoalkiot on koodattu binääriaakkoston $\{0, 1\}$ merkkijonoiksi. Merkkijonon x pituutta merkitään $|x|$:llä. Merkkijonojoukkoja sanotaan myös (*formaaleiksi*) *kieliksi*. Ominaisuuteen Π liittyvä päätösongelma voidaan samaistaa sen "kyllä"-tapausten muodostamaan kieleen

$$A_\Pi = \{x \mid x\text{:llä on ominaisuus } \Pi\};$$

esimerkiksi lausekalkyylin toteutuvuusongelmaa (engl. "satisfiability") vastaa kieli

$$\mathrm{SAT} = \{\varphi \mid \varphi \text{ on toteutuva lausekalkyylin kaava}\}.$$

Seuraavassa käytetään usein lyhyttä ilmaisua "päätösongelma A_Π," kun tarkoitetaan ominaisuuteen Π liittyvää päätösongelmaa, jonka esitys formaalina kielenä on A_Π.

Jotta laskentaongelmien algoritmista ratkaisemista voitaisiin käsitellä täsmällisesti, täytyy käytetty ohjelmointikieli kiinnittää. Seuraavassa oletetaan tietojenkäsittelyteoriassa vakiintuneen tavan mukaan algoritmit esitetyiksi ns. *Turingin koneina* [9], mutta periaatteessa esitysformalismiksi voitaisiin valita mikä tahansa yleiskäyttöinen ohjelmointikieli (LISP, Pascal, C tms.) Turingin koneen (so. ohjelman) M *tunnistama kieli* $L(M)$ määritellään niiden syötteiden kokoelmana, joilla kone tuottaa tuloksen 1 ("kyllä"):

$$L(M) = \{x \mid M(x) = 1\}.$$

Kone M siis ratkaisee ominaisuuteen Π liittyvän päätösongelman, jos ja vain jos $L(M) = A_\Pi$.

Koneen M *ajantarve syötteellä* x, merk. $\mathrm{time}_M(x)$, määritellään sen tällä syötteellä suorittamien alkeisoperaatioiden määränä. Päätösongelma A on *polynomisessa ajassa ratkeava*, jos jollakin Turingin koneella M ja vakiolla $c > 0$ on $L(M) = A$ ja $\mathrm{time}_M(x) \leq |x|^c + c$ kaikilla syötteillä x. Kaikkien polynomisessa ajassa ratkeavien päätösongelmien luokkaa merkitään P:llä. Yleisemmin voidaan mille tahansa aikarajafunktiolle $t : \mathcal{N} \to \mathcal{N}$ määritellä ajassa t ratkeavien ongelmien muodostama *vaativuusluokka*

$$\mathrm{DTIME}(t) = \{A \mid A = L(M) \text{ jollakin } M \text{ s.e. } \mathrm{time}_M(x) \leq t(|x|) \text{ kaikilla } x\};$$

tällöin siis

$$P = \bigcup_{c>0} \mathrm{DTIME}(n^c + c).$$

Luokan P ohella toinen tärkeä päätösongelmien vaativuusluokka on NP (engl. "nondeterministic polynomial time"). Tämän ongelmaluokan täsmälliseen määrittelyyn ei tässä ole mahdollisuutta (ks. [7, 9]), mutta todetaan sen keskeiset ominaisuudet:

(i) $P \subseteq NP$, ja luultavasti myös $P \neq NP$, joskin tarkkaan ottaen jälkimmäisen väitteen todistus on huomattava avoin kysymys;

(ii) Luokkaan NP kuuluu suuri joukko käytännössä merkittäviä, mutta laskennallisesti vaikeita ns. *NP-täydellisiä ongelmia*, joilla on se ominaisuus että jos $P \neq NP$ (kuten yleisesti uskotaan), niin mikään niistä ei ole luokassa P, so. polynomisessa ajassa ratkeava. Esimerkiksi lausekalkyylin toteutuvuusongelma on NP-täydellinen.

Tarkastellaan sitten Turingin koneita M, jotka kullakin syötteellä x tuottavat tulokseksi joko $M(x) = 1$ ("kyllä"), $M(x) = 0$ ("ei") tai $M(x) = ?$ ("en tiedä"). Tällainen kone M on *yhteensopiva* kielen A kanssa, jos kaikilla x joilla $M(x) \neq ?$, on $M(x) = 1$ jos ja vain jos $x \in A$. (Intuitiivisesti kone M kuvaa ongelman A jonkin *erikoistapausalgoritmin*, joka toimii aina oikein,

mutta ei kata kaikkia ongelman tapauksia.) Sanotaan, että kone M *ratkaisee* syötteen x, jos $M(x) \neq ?$. Koneen M kokoa ("ohjelmarivien määrää") merkitään $|M|$:llä [3].

Olkoon $t : \mathcal{N} \to \mathcal{N}$ jokin aikarajafunktio. Ongelman A tapauksen x *t-rajoitettu tapausvaativuus* (engl. "instance complexity") määritellään tällöin:

$$\mathrm{ic}^t(x : A) = \min\{|M| \mid M \text{ on } A\text{:n kanssa yhteensopiva kone, jonka ajantarve on enintään } t \text{ ja joka ratkaisee } x\text{:n}\}.$$

Toisin sanoen: syötteen x tapausvaativuus suhteessa ongelmaan A on pienimmän sellaisen ajassa t toimivan A:n erikoistapausalgoritmin koko, joka ratkaisee x:n. Huomataan, että tämä määritelmä välttää johdantoluvussa kuvatun taulukkohakuongelman, koska yksittäisten syötteiden taulukointi tosin nopeuttaa algoritmia näillä syötteillä, mutta samalla väistämättä kasvattaa algoritmin kokoa.

Määritelmä on ilmeistä sukua Chaitinin, Kolmogorovin ym. [4, 13, 21, 8, 10] tarkastelemalle merkkijonokompleksisuuden käsitteelle, jonka mukaan merkkijonon x *t-rajoitettu kompleksisuus* määritellään

$$C^t(x) = \{|M| \mid M(\lambda) = x, \mathrm{time}_M(\lambda) \leq t(|x|)\},$$

missä λ merkitsee tyhjää syötemerkkijonoa. Erona on, että kun merkkijonokompleksisuus mittaa merkkijonon x rakenteellista monimutkaisuutta sinänsä, niin tapausvaativuus mittaa sitä, miten vaikeata on jonkin tietyn *ominaisuuden* testaaminen merkkijonosta x. Esimerkiksi jos x on jonkin kokonaisluvun binääriesitys, voidaan esitetyn luvun parillisuus tai parittomuus selvittää tutkimalla ainoastaan x:n viimeistä bittiä, aivan siitä riippumatta miten kompleksinen merkkijono x muuten on. Toisaalta voidaan osoittaa (ks. Lause 1), että merkkijonon x kompleksisuus antaa aina ylärajan x:n tapausvaativuudelle suhteessa mihin tahansa ongelmaan.

Tapausvaativuudesta ja merkkijonokompleksisuudesta voidaan määritellä myös aikarajattomat versiot, joissa vaaditaan vain, että tarkasteltavat laskennat lopulta pysähtyvät (so. että tarkasteltavat koneet M ovat *totaalisia*). Näitä käsitteitä merkitään yksinkertaisemmin $\mathrm{ic}(x : A)$ ja $C(x)$.

2. Tapausvaativuuden perusominaisuuksia

Tapausvaativuusmitalla on joukko miellyttäviä formaaleja ominaisuuksia, jotka osoittavat että asetettu määritelmä on intuitiivisesti oikeansuuntainen. Esiteltyjen peruskäsitteiden vähäisyyden takia voidaan tässä näistä ominaisuuksista mainita vain muutama.

[3] Tämä merkintä on tarkkaan ottaen hieman epätäsmällinen, koska koneen koko riippuu käytetystä koodauksesta; täsmälliset määritelmät löytyvät esim. lähteistä [15, 18].

Lause 1 ([18]) *Jokaisella aikarajafunktiolla*[4] $t : \mathcal{N} \to \mathcal{N}$ *on vakio* $c > 0$, *siten että kaikilla* A *ja* x *on voimassa*

$$\mathrm{ic}^{t'}(x : A) \leq C^t(x) + c,$$

missä $t'(n) = ct(n) \log t(n) + c$. *Aikarajattomassa tapauksessa voidaan määrittää vakio* $c > 0$, *siten että kaikilla* A *ja* x *on voimassa*

$$\mathrm{ic}(x : A) \leq C(x) + c.$$

Lauseen mukaan siis merkkijonon x tapausvaativuus on aina enintään vakiota vaille x:n kompleksisuuden suuruinen, jos tarkasteltavien koneiden ajantarpeessa sallitaan pieni kasvu. Lauseen todistus perustuu siihen yksinkertaiseen havaintoon, että merkkijono x voidaan aina ratkaista ajassa t "taulukkohakualgoritmilla," jonka koko on suunnilleen $C^t(x)$: syötteellä y algoritmi purkaa sisäänrakennetusta $C^t(x)$ bitin "taulukostaan" merkkijonon x ajassa $t(|x|)$, tutkii onko $y = x$, ja jos on, antaa ennalta koodatun vastauksen; muilla syötteillä algoritmi vastaa "en tiedä." Todistuksen yksityiskohdat sivuutetaan.

Toinen yksinkertainen lause tarjoaa vaativuusluokkien karakterisoinnin tapausvaativuuden avulla:

Lause 2 ([18]) *Päätösongelma* A *kuuluu luokkaan* $\mathrm{DTIME}(t)$, *jos ja vain jos on olemassa vakio* $c > 0$, *siten että kaikilla* x *on* $\mathrm{ic}^t(x : A) \leq c$.

Seuraus 3 *Päätösongelma* A *kuuluu luokkaan* P, *jos ja vain jos on olemassa polynomi* t *ja vakio* $c > 0$, *siten että kaikilla* x *on* $\mathrm{ic}^t(x : A) \leq c$.

Myös Lauseen 2 (helppo, mutta ei aivan triviaali) todistus sivuutetaan.

Melko mielenkiintoinen on seuraava tulos, joka osoittaa että jos $P \neq NP$, niin NP-täydellisiä ongelmia ei voida ratkaista polynomisessa ajassa toimivilla "pienillä" erikoistapausalgoritmeilla:

Lause 4 ([18]) *Oletetaan, että* $P \neq NP$, *ja olkoon* A *jokin* NP*-täydellinen ongelma. Tällöin on jokaisella polynomilla* t *ja vakiolla* $c > 0$ *äärettömän monta syötettä* x, *joilla* $\mathrm{ic}^t(x : A) > c \log_2 |x|$.

3. Vaikeat yksittäistapaukset

Olkoon A jokin päätösongelma, jota ei voida ratkaista asymptoottisesti ajassa t, so. jolla $A \notin \mathrm{DTIME}(t)$. Lauseen 2 mukaan A:n tapausten t-rajoitettu tapausvaativuus $\mathrm{ic}^t(x : A)$ kasvaa rajatta, kuitenkin niin että arvo $C^t(x)$ on likimääräinen yläraja minkä tahansa tapauksen x vaativuudelle (Lause 1).

[4] Tarkkaan ottaen funktiolta t joudutaan tässä vaatimaan tietty säännöllisyysehto, ns. "aikakonstruoituvuus" [1, s. 42], mutta tämä ei oleellisesti rajoita tarkasteltavien funktioiden luokkaa.

On luonnollista kysyä, onko A:lla myös tapauksia joilla tämä yläraja saavutetaan. Kysymys on mielenkiintoinen, koska intuitiivisesti tapaukset x, joilla $\mathrm{ic}^t(x : A) \approx C^t(x)$, ovat sellaisia, että niille ei ole oleellisesti parempaa ajassa t toimivaa ratkaisumenetelmää kuin triviaali taulukkohaku. Artikkelissa [18] väite, että kaikilla asymptoottisesti t-vaikeilla ongelmilla välttämättä on myös tällaisia maksimaalisen vaikeita yksittäistapauksia, muotoiltiin seuraavana "tapausvaativuskonjektuurina":

Konjektuuri 5 ([18]) *Olkoon A päätösongelma, jolla $A \notin \mathrm{DTIME}(t)$. Tällöin on olemassa vakio $c > 0$ siten, että äärettömän monella x on*

$$\mathrm{ic}^t(x : A) \geq C^{t'}(x) - c,$$

missä $t'(n) = ct(n) \log t(n) + c$.

Tämän konjektuurin selvittely on johtanut joihinkin sangen mielenkiintoisiin ja yllättäviinkin tuloksiin. Määritellään ensinnäkin luokan P "eksponentiaalinen" vastine, syötteen pituuden suhteen eksponentiaalisessa ajassa ratkeavien ongelmien luokka E:

$$E = \bigcup_{c>0} \mathrm{DTIME}(c^n).$$

Myös luokalla NP on eksponentiaalinen vastineensa, luokka NE (ks. tarkemmin teos [1]). Luokista E ja NE tiedetään, että $P \subsetneq E$ ja $NP \subsetneq NE$, ja jos $E \neq NE$, niin myös $P \neq NP$.

Sanotaan, että ongelmalla A on *p-vaikeita tapauksia*, jos jokaista polynomia t kohden voidaan löytää polynomi $t' \geq t$ ja vakio c siten, että äärettömän monella x on

$$\mathrm{ic}^t(x : A) \geq C^{t'}(x) - c.$$

Artikkelissa [18] todistetaan seuraavat tulokset:

Lause 6 ([18]) *Jos $E \neq NE$, niin jokaisella "luonnollisella"* [5] *NP-täydellisellä ongelmalla on p-vaikeita tapauksia.*

Lause 7 ([18]) *Kaikilla E- ja NE-täydellisillä ongelmilla on p-vaikeita tapauksia.*

Artikkelissa [6] on näiden tulosten todistuksia yksinkertaistettu, ja Lause 6 vahvistettu muotoon:

Lause 8 ([6]) *Jos* P $\neq$ NP*, niin jokaisella "luonnollisella"* NP*-täydellisellä ongelmallla on p-vaikeita tapauksia.*

[5] Tähänkin ilmaisuun sisältyy tietty säännöllisyysehto, ns. "reilujen palautusten" [1, s. 61] käyttö, jota ei voida tässä tarkemmin eritellä. Kaikki yleisesti tunnetut *NP*-täydelliset ongelmat, mukaanlukien lausekalkyylin toteutuvuusongelma SAT, täyttävät tämän ehdon.

Lisäksi artikkelissa todistetaan seuraava yleinen tulos:

Lause 9 ([6]) *Olkoon $t(n) \geq n$ jokin aikakonstruoituva funktio ja $A \notin \mathrm{DTIME}(t)$. Tällöin on olemassa vakio $c > 0$ siten, että äärettömän monella x on*

$$\mathrm{ic}^t(x : A) \geq C^{t'}(x) - c,$$

missä $t'(n) = c2^{2n}t(n)(n + \log t(n))$.

(Myös artikkeliin [18] sisältyy vastaava, mutta huomattavasti heikompi tulos.)

Artikkelissa [2] Lause 7 vahvistetaan ideaaliseen muotoonsa:

Lause 10 ([2]) *Jokaisella E- ja NE-täydellisellä ongelmalla A on olemassa vakio $c > 0$ siten, että äärettömän monella x on*

$$\mathrm{ic}^t(x : A) \geq C^t(x) - c.$$

Itse asiassa Lauseen 10 todistus osoittaa, että tällaisia maksimaalisen vaikeita tapauksia on jokaisella E-täydellisellä ongelmalla *paljon*: enintään n-merkkisten syötteiden joukosta aina vähintään 2^{n^ϵ} kappaletta toteuttaa Lauseen 10 ehdon, jollakin n:stä riippumattomalla vakiolla $\epsilon > 0$.

Artikkeleissa [2, 14] osoitetaan myös, että aikarajattomassa muodossaan tapausvaativuuskonjektuuri on tosi kaikille *r.e.-täydellisille* ongelmille (tämän ja muiden seuraavassa käytettävien rekursioteoreettisten käsitteiden määrittelystä ks. [16, 20]):

Lause 11 ([2, 14]) *Jokaisella r.e.-täydellisellä ongelmalla A on olemassa vakio $c > 0$ siten, että äärettömän monella $x \in A$ ja äärettömän monella $x \notin A$ on*

$$\mathrm{ic}(x : A) \geq C(x) - c.$$

Sovellettuna esimerkiksi ensimmäisen kertaluvun predikaattikalkyylin kaavojen validisuusongelmaan, joka on r.e.-täydellinen [16, s. 165], Lause 11 sanoo, että on olemassa äärettömän monta — sekä validia että ei-validia — predikaattikalkyylin kaavaa, joiden validisuutta ei voida testata millään oleellisesti paremmalla efektiivisellä menetelmällä kuin koodaamalla kaavat itse testiohjelmaan "taulukkoon".

Sangen yllättävää on, että aikarajaton tapausvaativuuskonjektuuri *ei* ole tosi kaikille r.e.-epätäydellisille ongelmille:

Lause 12 ([14]) *On olemassa luokan r.e. ei-rekursiivisia ongelmia A, joilla on jollakin vakiolla $c > 0$ ja kaikilla x voimassa:*

$$\mathrm{ic}(x : A) \leq \log_2 C(x) + c.$$

Paradoksaalisesti siis tällaista ongelmaa A ei voida ratkaista millään vakiokokoisella rekursiivisella algoritmilla, mutta sen kukin yksittäistapaus x voidaan ratkaista A:n kanssa yhteensopivalla erikoisalgoritmilla, joka ei kuitenkaan sisällä juuri mitään informaatiota x:stä.

Viitteet

[1] Balcázar, J. L., Díaz, J., Gabarró, J. *Structural Complexity I.* Springer-Verlag, Berlin Heidelberg, 1988.

[2] Buhrman, H., Orponen, P. Random strings make hard instances. *Proc. of the 9th Ann. Conf. on Structure in Complexity Theory (Amsterdam, June 1994).* IEEE Press, New York, NY, 1994. Ss. 217–222. (Täydellisempi versio ilmestyy lehdessä *J. Comput. System Sciences.*)

[3] Calude, C. *Information and Randomness: An Algorithmic Perspective.* Springer–Verlag, Berlin, 1994.

[4] Chaitin, G. J. On the length of programs for computing binary sequences. *J. Assoc. Comput. Mach. 13* (1966), 547–569.

[5] Chaitin, G. J. A theory of program size formally identical to information theory. *J. Assoc. Comput. Mach. 22* (1975), 329–340.

[6] Fortnow, L., Kummer, M. Resource-bounded instance complexity. *Proc. of the 12th Symp. on Theoretical Aspects of Computer Science (München, March 1995).* Lecture Notes in Computer Science 900. Springer-Verlag, Berlin, 1995. Ss. 597–608.

[7] Garey, M. R., Johnson, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, 1979.

[8] Hartmanis, J. Generalized Kolmogorov complexity and the structure of feasible computations. *Proc. of the 24th Ann. Symp. on the Foundations of Computer Science (Tucson, Az., Nov. 1983).* IEEE Press, New York, NY, 1983. Ss. 439–445.

[9] Hopcroft, J. E., Ullman, J. D. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, MA, 1979.

[10] Ko, K. On the notion of infinite pseudorandom sequences. *Theoret. Comput. Sci. 48* (1986), 9–33.

[11] Ko, K. A note on the instance complexity of pseudorandom sets. *Proc. of the 7th Ann. Conf. on Structure in Complexity Theory (Boston, MA, June 1992).* IEEE Press, New York, NY, 1992. Ss. 327–337.

[12] Ko, K., Orponen, P., Schöning, U., Watanabe, O. What is a hard instance of a computational problem? *Proc. of the Conf. on Structure in Complexity Theory (Berkeley, CA, June 1986).* Lecture Notes in Computer Science 223. Springer-Verlag, Berlin, 1986. Ss. 197–217.

[13] Kolmogorov, A. N. Three approaches to the quantitative definition of information. *Prob. Info. Transmission 1* (1965), 1–7.

[14] Kummer, M. The instance complexity conjecture. *Proc. of the 10th Ann. Conf. on Structure in Complexity Theory (Minneapolis, MN, June 1995).* IEEE Press, New York, NY, 1995. Ss. 111–124.

[15] Li, M., Vitányi, P. M. B. *An Introduction to Kolmogorov Complexity and Its Applications.* Springer-Verlag, New York, NY, 1993.

[16] Odifreddi, P. *Classical Recursion Theory.* Elsevier, Amsterdam, 1989.

[17] Orponen, P. On the instance complexity of NP-hard problems. *Proc. of the 5th Ann. Conf. on Structure in Complexity Theory (Barcelona, June 1990).* IEEE Press, New York, NY, 1990. Ss. 20–27.

[18] Orponen, P., Ko, K., Schöning, U., Watanabe, O. Instance complexity. *J. Assoc. Comput. Mach. 41* (1994), 96–121.

[19] Rissanen, J. *Stochastic Complexity in Statistical Inquiry.* World Scientific, Singapore, 1989.

[20] Rogers, H., Jr. *Theory of Recursive Functions and Effective Computability.* McGraw–Hill, New York, N.Y., 1967.

[21] Solomonoff, R. J. A formal theory of inductive inference. *Information and Control 7* (1964), 1–22.

[22] Wegener, I. The Complexity of Boolean Functions. Wiley–Teubner, Stuttgart, 1987.