Mitä tietojenkäsittelyteoriaan kuuluu?

Pekka Orponen
Tietojenkäsittelyteorian laboratorio
Teknillinen korkeakoulu
orponen@tcs.hut.fi

1 Teoriainsinööri?

Kun Teknillisen korkeakoulun viestintäyksikkö tiedusteli minulta otsikkoa sille tietojenkäsittelyteorian professuuria varten pitämälleni virkaanastujaisesitelmälle,[1] johon tämä kirjoitus pohjautuu, ajattelin ensin valita provokatiivisen aiheen "Mihin tietojenkäsittelyteoriaa tarvitaan?" Onhan nimittäin "teoriainsinööri" jo käsitteenä jonkinlainen oksymoroni, tai kuten englanniksi sanotaan, "contradiction in terms."

Kun sitten kuitenkin totesin, että pari viikkoa ennen esitelmää aloittamalleni *Tietojenkäsittelyteorian perusteet* -kurssille oli ilmoittautunut kaikkiaan 752 opinhaluista tieto- ja sähköteekkaria, päättelin että ilmeisesti teorian tarpeellisuus myös insinööriopinnoissa oli tullut jo riittävässä määrin muutenkin osoitettua — jos ei muille, niin ainakin korkeakoulun tutkintovaatimuksia suunnitteleville kollegoilleni.

Niinpä otsikoin esitelmäni uudelleen ja päätin käyttää tarjoutuneen tilaisuuden hyväkseni esitelläkseni yleisöksi ajatelluille "valistuneille kadunmiehille," millaisia kysymyksiä tietotekniikan perustavaan taustateoriaan kuuluu: mitä teoria on saavuttanut ja mitkä ovat sen nykyisiä kehityslinjoja.

2 Teoria A ja teoria B

Tietojenkäsittelyteoria voidaan metodisesti jakaa pääpiirteissään kahteen laajaan alueeseen siten kuin Jan van Leeuwenin toimittama mainio kaksiosainen *Handbook of Theoretical Computer Science* [30] tekee: laskentaalgoritmien ja niiden kompleksisuuden tutkimukseen (tyyppi A) sekä tietojenkäsittelyjärjestelmien formaalien kuvausmenetelmien tutkimukseen (tyyppi B). Jako ei tietenkään ole tarkka: monessa tyypin B tutkimusongelmassa nousee esiin tyypin A kysymyksiä ja

[1] 5.2.2002.

kääntäen; lisäksi on monia suppeampia osa-aloja vaikea sijoittaa tämän luokituksen kummallekaan puolen. Jonkinlaisen hahmon saamiseksi tämä jako kuitenkin on käyttökelpoinen, ja itse asiassa useimmat tietojenkäsittelyteoreetikot näyttävät identifioituvan jompaan kumpaan tyyppiin.

Van Leeuwenin teos on vuodelta 1990 ja alkaa siten olla jo hieman ikääntynyt, mutta enimmäkseen sen sisällysluettelossa (kuva 1) luetellut alat ovat edelleen ajankohtaisia. Tämän päivän näkökulmasta kirjasta lähinnä puuttuu joitakin kymmenen viime vuoden aikana näkyvälle sijalle nousseita aloja, esimerkiksi bioinformatiikan menetelmät [9, 16, 19] ja ns. koneoppimisteoria [22, 28], ja joidenkin siinä käsiteltyjen alojen painotukset ovat muuttuneet. Esimerkiksi kirjan algoritmiosuudessa käsiteltäisiin tänään varmasti huomattavasti vuotta 1990 laajemmin kombinatorisia satunnais- ja likimääräisalgoritmeja [8, 34, 49].[2]

3 Puuhastelusta insinöörityöksi

Kerätäkseni esimerkkiaineistoa siitä, millaisia vaikutuksia tietojenkäsittelyteorian tutkimuksella on viimeisimpien vuosikymmenten aikana ollut, kävin TKK:n kirjakellarissa läpi muutaman 1960-luvun alun vuosikerran nykyisinkin tärkeää *Communications of the ACM* -ajankohtaislehteä. Noista neljänkymmenen vuoden takaisista lehdistä käy selvästi ilmi, että tuolloin tietotekniikan alan kuumin tutkimus- ja kehitystyön aihe olivat ohjelmointikielten kääntäjät. Ala oli tuolloin ja on edelleen erittäin tärkeä, koska korkean tason ohjelmointikielet ja niiden tehokkaat kääntäjät ovat teollisen mittakaavan ohjelmistotuotannon perusedellytys.

Kuusikymmenluvun alussa oli *CACM*-lehden jokaisessa numerossa useita tätä aihepiiriä käsitteleviä kirjoituksia, joista enin osa näyttää nykypäivän näkökulmasta huvittavan naiiveilta. Nykylukijan huomiota kiinnittää näissä vanhoissa kääntäjätekniikan artikkeleissa erityisesti se, että jokaisella kääntäjähankkeella näyttää olevan oma omaperäinen lähestymistapansa projektiin. Tilanne saattaa toki nykyisin olla meidän huomaamattamme sama joillakin muilla ohjelmistotuotannon aloilla. Esimerkiksi esitelmäni aikaan tuoreimmassa *CACM*:n numerossa 45:2 (helmikuu 2002) käsiteltiin ohjelmistotyön ontologioita (*Special Issue on Ontology Applications and Design*); ehkä 2040-luvulla samaan tapaan hymähdellään näiden ajatusten epäselvyydelle ja keskentekoisuudelle.

Syy siihen, että 1960-luvun alun kääntäjätekniikka-artikkelit nyt tuntu-

[2] Tyypin A teorian osalta on Van Leeuwenin teoksen rinnalle nyttemmin tarjolla myös ajantasaisempi Mikhail Atallahin toimittama *Algorithms and Theory of Computation Handbook* [7].

Vol. A: Algorithms and Complexity

1. Machine Models and Simulation
2. A Catalog of Complexity Classes
3. Machine-Independent Complexity Theory
4. Kolmogorov Complexity and its Applications
5. Algorithms for Finding Patterns in Strings
6. Data Structures
7. Computational Geometry
8. Algorithmic Motion Planning in Robotics
9. Average-Case Analysis of Algorithms and Data Structures
10. Graph Algorithms
11. Algebraic Complexity Theory
12. Algorithms in Number Theory
13. Cryptography
14. The Complexity of Finite Functions
15. Communication Networks
16. VLSI Theory
17. Parallel Algorithms for Shared-Memory Machines
18. General Purpose Parallel Architectures

Vol. B: Formal Models and Semantics

1. Finite Automata
2. Context-Free Languages
3. Formal Languages and Power Series
4. Automata on Infinite Objects
5. Graph Rewriting: An Algebraic and Logic Approach
6. Rewrite Systems
7. Functional Programming and Lambda Calculus
8. Type Systems for Programming Languages
9. Recursive Applicative Program Schemes
10. Logic Programming
11. Denotational Semantics
12. Semantic Domains
13. Algebraic Specification
14. Logics of Programs
15. Methods and Logics for Proving Programs
16. Temporal and Modal Logic
17. Elements of Relational Database Theory
18. Distributed Computing: Models and Methods
19. Operational and Algebraic Semantics of Concurrent Processes

Kuva 1: Teoksen *Handbook of Theoretical Computer Science* [30] sisältö.

vat niin vanhahtavilta ja harrastelijamaisilta, on että ohjelmointikielten kuvausformalismien tutkimus eteni 1960-luvulla pitkin harppauksin, pääosin kääntäjätekniikkaa kohtaan tunnetun kiinnostuksen takia, mutta osin myös omista sisäisistä syistään. 1970-luvun puoliväliin mennessä ohjelmointikielten kääntäjien laatiminen oli muuttunut hahmottomasta *ad hoc*-temppujen kokoelmasta järjestelmälliseksi, tiettyjä teoreettisesti perusteltuja menetelmiä noudattavaksi insinöörityöksi [4, 5, 45].

Tehokkaiden kääntäjien laatiminen nykyisille, aiempia paljon rikkaammille ohjelmointikielille on edelleen suuritöinen ja haastava tehtävä, mutta siinä noudatettava lähestymistapa on nykyisin täysin selvä ja työssä tarvittavia käsitteellisiä perustyökaluja opetetaan mm. aiemmin mainitsemallani *Tietojenkäsittelyteorian perusteet* -kurssilla ja sitä seuraavilla varsinaisilla kääntäjätekniikkakursseilla. Kuusikymmenluvun alun haastavat kääntäjäprojektit kelpaisivat nykypäivän opiskelijoille hädin tuskin harjoitustöiksi.

4 Pysähtymisongelmasta e-kauppaan

Ohjelmointikielten teorian menestystarina edustaa van Leeuwenin luokituksessa lähinnä B-tyypin formaalien kuvausmenetelmien tutkimusta, joskin siinä on ollut merkittävä panos myös A-tyypin algoritmikehitystyöllä.

Otan toisen, selkeämmin A-tyyppisen esimerkin tietojenkäsittelyteorian vaikutuslinjoista lähempää omaa tutkimusalaani.

Eräs yleinen harhaluulo, joka vallitsee jopa sellaisten tietotekniikan opiskelijoiden keskuudessa jotka eivät vielä ole suorittaneet *Tietojenkäsittelyteorian perusteet*-kurssia, on että mikä tahansa täsmällisesti muotoiltu ongelma voidaan ratkaista tietokoneella. Näinhän ei suinkaan ole.

Tarkastellaan esimerkkinä vaikkapa kaikille Windows-käyttöjärjestelmän käyttäjille tuttua *Blue Screen of Death (BSD)* -ongelmaa. Varsinkin vanhemmissa käyttöjärjestelmän versioissa tunnetusti käy usein niin, että kun käyttäjä antaa jollekin järjestelmän ohjelmalle epäsuotuisan syötteen, järjestelmä niin sanotusti "tilttaa," näkyviin ilmestyy kryptisiä virheilmoitustekstejä sisältävä sininen ruutu, eikä konetta pysty enää käyttämään käynnistämättä sitä uudelleen.

Tämän kiusallisen ilmiön välttämiseksi olisi kovin kätevää, jos Windows-järjestelmään voitaisiin asentaa apuohjelma, nimeltään sanokaamme `safe`, joka ennen käyttäjän antaman syötteen `x` suorittamista järjestelmäohjelmalla `p` testaisi, onko vaaraa järjestelmän tilttaamisesta tämän suorituksen seurauksena. Jos virhetoiminto uhkaa, ohjelmaa `p` ei suoritettaisikaan, vaan käyttäjää varoitettaisiin vaarasta ja pyydettäisiin antamaan jokin muu syö-

te. Voisipa järjestelmä tässä tapauksessa vaikka lähettää automaattisesti generoidun vikaraportin käyttöjärjestelmän valmistajalle.

Tällainen BSD-testausohjelma ei kuitenkaan ole edes periaatteessa mahdollinen, mikä voidaan todeta seuraavasti. Oletetaan, tavoitellun väitteen vastaisesti, että jollakin ohjelmointikielellä voitaisiin kirjoittaa kuvan 2(a) mukainen ohjelma `safe`. Muokataan tästä kuvan 2(b) mukainen ohjelma `paradox`.

Merkitään edellä kuvattua ohjelman `paradox` ohjelmatekstiä $\tilde{p}$:llä ja tarkastellaan ohjelman `paradox` toimintaa tällä *omalla ohjelmatekstillään*, so. laskentaa `paradox(`$\tilde{p}$`)`. Pysähtyykö tämä hallitusti vai tilttaako kone? Päätellään:

> `paradox(`$\tilde{p}$`)` pysähtyy hallitusti,
>
> jos ja vain jos
>
> `safe(`$\tilde{p}$`,`$\tilde{p}$`) = 0`,
>
> jos ja vain jos
>
> suoritus `paradox(`$\tilde{p}$`)` tilttaa koneen.

Saadusta ristiriidasta joudutaan päättelemään, ettei alunperin toivottua BSD-testausohjelmaa `safe` voi olla olemassa.

Oleellisesti tämän todistuksen tiettyjen ohjelmointiongelmien ratkeamattomuudelle esitti brittiläinen matemaatikko Alan Turing vuonna 1936 [47] — siis jo ennen Windows-järjestelmän kehittämistä, ja itse asiassa jo kymmenkunta vuotta ennen ensimmäisten tietokoneiden rakentamista. Alkuperäisessä muodossaan ongelma tunnetaan *Turingin pysähtymisongelman* nimellä.[3]

Turing oli monin tavoin mielenkiintoinen ja kompleksinen persoona, jonka elämäkerran [25] pohjalta on kirjoitettu jopa suhteellisen hyvin menestynyt näytelmä. Hän siis pohti jo 1930-luvulla nykyisen tietojenkäsittelyteorian peruskysymyksiä ja mm. määritteli teorian käsitteistöön edelleen keskeisesti kuuluvan yleispätevän laskennan mallin, ns. *Turingin koneen*. Vuodet 1936–38 Turing vietti Princetonin *Institute for Advanced Study*:ssa, jossa hän mm. suunnitteli matemaattisten ideoidensa konkreettista toteuttamista laskulaitteina.

Princetonissa Turing tutustui myös nykyisten tietokoneiden arkkitehtuurin pääsuunnittelijana pidettyyn matemaatikko John von Neumanniin, joka tarjosi Turingille paikkaa assistenttinaan IAS:ssä. Turing kuitenkin palasi Britanniaan vuonna 1938 ja toimi siellä sodan aikana keskeisissä teh-

[3] Tarkkaan ottaen Turingin artikkelin [47] ko. osuus tarkastelee päättymättöminä desimaalijonoina esitettyjen reaalilukujen mekaanista tuottamista, jolloin myönteinen tapaus onkin koneen *pysähtymättömyys*. Siten olisi historiallisesti täsmällisempää puhua "Turingin pysähtymättömyysongelmasta," mutta käytetty termi on vakiintunut ja kuvaava, ja lisäksi todistustekniikka on molemmissa tapauksissa jokseenkin sama.

(a)
$$\texttt{safe(p,x)} = \begin{cases} 1, & \text{jos suoritus } \texttt{p(x)} \text{ onnistuu OK;} \\ 0, & \text{jos suoritus } \texttt{p(x)} \text{ tilttaa koneen.} \end{cases}$$

(b)
```
safe(p,x):
    ...   {Ohjelman safe oletettu teksti.}

paradox(p):
    suorita ohjelma safe syötteillä (p,p);
    jos safe(p,p) = 0, niin pysähdy hallitusti;
    jos safe(p,p) = 1, niin tilttaa kone.
```

Kuva 2: (a) Hypoteettinen BSD-testausohjelma. (b) Ristiriitainen ohjelma.

tävissä Bletchley Parkin tiedustelu- ja salakirjoitusyksikössä, sekä sodan aikana ja sen jälkeen useissa brittiläisissä tietokoneiden rakennushankkeissa. Jälkeen päin tarkastellen vaikuttaa ilmeiseltä, että Turingin teoreettisilla ideoilla oli merkittävä vaikutus myös John von Neumannin tietojenkäsittelyä koskevaan ajatteluun ja sitä kautta kaikkien nykyisten tietokoneiden rakenteeseen [6, ss. 177–181][14].

Tietyllä tapaa Turingin ratkeamattomia ongelmia koskevan työn jatkona voidaan pitää sitä vaikeasti ratkaistavien t. *laskennallisesti vaativien ongelmien* teorian systematisointia, joka sai alkunsa Juris Hartmanisin ja Richard Stearnsin 1960-luvun töistä [20, 21] ja johti 1970-luvulla Stephen Cookin [13], Leonid Levinin [31] ja Richard Karpin [27] kehittämään *NP-täydellisyyden* teoriaan. Menemättä teorian yksityiskohtiin tilannetta voidaan kuvata seuraavasti.

Hartmanisin ja Stearnsin tulosten perusteella tiedetään, että on olemassa tietojenkäsittelyongelmia, jotka kyllä periaatteessa ovat Turingin mielessä ohjelmallisesti ratkaistavissa, mutta joiden mikä tahansa ratkaisumenetelmä vaatii jo kohtuullisen pienillä syötteillä enemmän kuin maailmankaikkeuden iän verran laskenta-aikaa. Tämän ns. eksponentiaalisen vaativien ongelmien luokan laajuutta ei kuitenkaan täysin tunneta; yleisesti ottaen on erittäin vaikeaa todistaa *mitään* konkreettista annettua ongelmaa kaikkien mahdollisten ratkaisumenetelmien suhteen vaativaksi.

Toisaalta on olemassa suuri joukko käytännön ohjelmistotyössä ja sovelletun matematiikan alalla vastaan tulevia kombinatorisia ongelmia, joil-

le ei tunneta yleispäteviä, tehokkaita ratkaisumenetelmiä ja joiden vahvasti epäillään olevan edellisessä mielessä eksponentiaalisen vaativia; varmaa todistusta tälle arvelulle ei ole kuitenkaan onnistuttu löytämään. Näitä ongelmia, joista tunnettuja esimerkkejä ovat mm. ns. kauppamatkustajan ongelma, verkonväritysongelma ja kokonaislukuohjelmointi, oli ainakin 1950-luvulta lähtien tutkittu erillisinä tutkimushaasteina, kunnes Cook ja Karp 1970-luvun alussa osoittivat, että ne ovat kaikki tietyssä mielessä ekvivalentteja, saman abstrakin "NP-täydellisen" ongelman eri ilmentymiä [17]. Edelleen on kuitenkin avoinna kysymys, ovatko nämä nyt keskenään ekvivalenteiksi tiedetyt ongelmat todella eksponentiaalisen vaativia, kuten epäillään, vai piileskeleekö jossakin niille kaikille yhteinen tehokas ratkaisumenetelmä.

Tämän kiehtovan "P vs. NP"-kysymyksen teoreettista ja käytännöllistä merkitystä kuvaa se, että kun amerikkalainen *Clay Mathematics Institute* keväällä 2000 uuden vuosituhannen kunniaksi nimesi seitsemän matemaattista tutkimushaastetta, joista kunkin selvittämisestä on luvassa miljoonan dollarin palkkio, "P vs. NP"-ongelma nostettiin listalle mukaan, sellaisten klassisen matematiikan suurten kysymysten kuin Riemannin hypoteesin ja Poincarén konjektuurin rinnalle [12].

Vaikka perustava "P vs. NP"-kysymys onkin vielä avoin, ja todella eksponentiaalisen vaikeiksi todistettuja ongelmia tunnetaan perin vähän, on niiden *mahdollisuus* kuitenkin inspiroinut merkittävää työtä esimerkiksi salakirjoitusteoriassa eli kryptografiassa.

1970-luvulle asti ajateltiin, että salakirjoitettujen viestien vaihtaminen kahden osapuolen kesken on mahdollista vain, jos heidän välillään on etukäteen jotain luotettavaa kanavaa käyttäen sovittu yhteisesti käytettävästä salausavaimesta. Uusien laskennan vaativuuteen liittyvien ideoiden innoittamina Whitfield Diffie ja Martin Hellman keksivät kuitenkin 1970-luvun alussa *epäsymmetrisen* salauksen idean, missä viestien salaaminen on laskennallisesti helppoa, mutta niiden avaaminen vaikeaa, ellei avaajan hallussa ole tiettyä lisäinformaatiota [15]. Tämä idea johtaa ns. *julkisen avaimen* salausprotokolliin, missä jokainen menetelmän käyttäjä voi toisista riippumatta julkistaa viestien salaamiseen käytettävän *salausavaimen* ja säilyttää vain omassa tiedossaan niiden avaamiseen tarvittavan *lukuavaimen* [42, 43].

Julkisen avaimen menetelmät poistavat aiempien salakirjoitusprotokollien kahdenvälisyyden muodostaman rajoituksen, ja ovat perustana lukemattomille nykyisin laajassa käytössä oleville suojatun tietoliikenteen, sähköisen kaupan ja rahaliikenteen sovelluksille. Tunnetuin julkisen avaimen salakirjoitusmenetelmä on tietojenkäsittelyteoreetikoiden Ronald Rivestin, Adi Shamirin ja Leonard Adlemanin 1970-luvun lopulla kehittämä RSA-

algoritmi [41].

Kaikkien julkisen avaimen menetelmien turvallisuus perustuu niiden pohjana olevien laskentaongelmien eksponentiaalisen vaativuuden oletukseen. Monet näistä oletuksista, esimerkiksi RSA-algoritmin taustalla oleva, eivät ole edes niin vahvasti tuettuja kuin "P $\neq$ NP"-hypoteesi, mutta melko hyvin nämä menetelmät ovat silti kestäneet käytössä.

Yhteenvetona voidaan siis johtaa kuvan 3 mukainen pitkä kaari Alan Turingin 1930-luvun laskettavuusteoreettisista töistä nykyisiin tietokoneisiin ja sähköisen kaupan järjestelmien perustana oleviin salausmenetelmiin saakka.

5 Turingin perintö

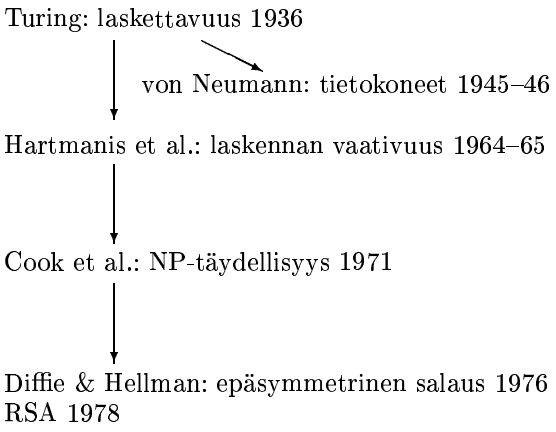
Mitä tietojenkäsittelyteoriaan sitten nykyisin kuuluu? Miten tietojenkäsittelyteoreetikot vaalivat Alan Turingin perintöä, ja mitkä ovat alan nykyisiä tutkimushaasteita?

Ala on sen verran laaja, että mitään yleispätevää kuvausta teorian nykytilasta on kohtuutonta edes yrittää. Kaikki van Leeuwenin kirjassa esitellyt tutkimusalueet ovat edelleen aktiivisia ja niiden rinnalle on noussut joukko uusia, jotka ehdottomasti vaatisivat käsittelyn kirjan uudessa laitoksessa. Salausalgoritmien käytännön merkityksen takia niiden taustateorioita tutkitaan intensiivisesti [18, 32], ja perinteisten yhdessä koneessa suoritettavien algoritmien rinnalle on noussut hajautetussa tietoverkossa tapahtuvan laskennan tutkimus [33, 46].

Kolme nähtävissä olevaa laajempaa trendiä ehkä ansaitsee tulla mainituksi yksittäisten osa-alojen kehitysraporttien sijaan.

Ensimmäinen on *tilastollisten* ja *stokastisten* menetelmien merkityksen voimakas kasvu. Viimeistään 1970-luvulla myös tietojenkäsittelyteoriassa herättiin huomaamaan, että monet ongelmat joiden ratkaiseminen deterministisillä menetelmillä on vaikeaa, ovat satunnaismenetelmillä hyvin hallittavissa. Tällä suunnalla on sittemmin sekä kehitetty erinomaisesti toimivia käytännön menetelmiä [1, 2, 34] että saavutettu vaikuttavia teoreettisia läpimurtoja [11, 34]. Toisaalta myös kiinnostus *luonnollisen*, kohinaisen signaalidatan käsittelyyn on noussut ihmiskäyttäjien tuottamien strukturoitujen aineistojen käsittelyn rinnalle, ja tämä luonnollisesti merkitsee tilastotieteen painoarvon lisääntymistä [22].

Toinen yleinen trendi on tietojenkäsittelyteorian hyödyntämän *matemaattisen perustan* jatkuva laajeneminen. Lienee tuskin enää yhtään matematiikan alaa, algebrallisesta geometriasta harmoniseen analyysiin, jolla ei olisi tietojenkäsittelyteoreettisia sovelluksia. Tämä merkitsee tiettyä haastetta alan tutkijakoulutukselle, mutta toisaalta kollegamme fyysikot ovat olleet vastaavassa tilanteessa 1700-luvulta lähtien ja periaatteella "kukin tekee mitä osaa" hekin näyttävät siinä kukoistaneen.



Kuva 3: Laskettavuusteoriasta kryptografiaan.

Kolmas trendi on lisääntyvä *vuorovaikutus* muiden tieteen ja tekniikan alojen kanssa. Tietojenkäsittelyteoria on tässä suhteessa saanut paljon vaikutteita varsinkin fysiikasta, esimerkiksi stokastisten optimointimenetelmien ja kompleksisten systeemien analyysitekniikoiden alalla. Toisaalta viime vuosina fyysikkopiireissä suurta kiinnostusta herättäneen *kvanttilaskennan* tutkimuksen pontimena oli tietojenkäsittelyteoreetikko Peter Shorin vuonna 1994 esittämä menetelmä [44] suurten kokonaislukujen nopeaan tekijöihinjakoon fysikaalisten systeemien kvanttitason epädeterministisyyttä hyväksikäyttäen. Shorin tulos oli mullistava, koska mm. RSA-salausalgoritmin turvallisuus perustuu oletukseen, että suurten kokonaislukujen tekijöinti on laskennallisesti vaativa ongelma. Nähtäväksi jää, onko todella käytännöllisen mittakaavan kvanttitietokoneiden rakentaminen mahdollista, mutta joka tapauksessa idean tutkimus on jo tähän mennessä tuottanut mielenkiintoista tietojenkäsittelyteoriaa ja ehkä myös mielenkiintoista fysiikkaa [24, 37].

Fysiikan lisäksi tietojenkäsittelyteorian vuorovaikutus on voimistumassa myös biologian suuntaan monin tavoin.

Yksi yhteistyösuunta on molekyylitasoisen ns. *DNA-laskennan* tutkimus, joka sai alkunsa RSA-algoritmistakin tutun lukuteoreetikko Leonard Adlemanin vuonna 1994 suorittamasta laboratoriokokeesta [3]. Adleman onnistui huolellisesti sommiteltuja biomolekyyleja käyttäen ratkaisemaan yksinkertaisen seitsemän kaupungin kauppamatkustajan ongelman "spontaanisti," kemiallisten vuorovaikutusten tuloksena koeputkessa. Toiveena on sittemmin ollut biosysteemien sisältä-

män massiivisen rinnakkaisuuden valjastaminen teholaskennan palvelukseen. Kuten kvanttitietokoneidenkin tapauksessa käytännön lopputulos on vielä epäselvä, mutta matkan varrella opittaneen yhtä ja toista sekä tietojenkäsittelyteoriasta että molekyylibiologiasta [38].

Toinen yhteistyösuunta ovat geneettiset ja evolutiiviset mallit, joiden alalla tietojenkäsittelyteorialla ja biologialla on mitä yllättävimpiä yhteyksiä. Tunnettu kosketuskohta on biologisten mallien soveltaminen optimointiongelmien ratkaisuun *geneettisten*, tai yleisemmin *evolutiivisten algoritmien* muodossa [10, 39]. Tämä kontakti on edelleen avannut erittäin mielenkiintoisia näkymiä biologisten ja kombinatoristen optimointiongelmien ns. *kelpoisuusmaastojen* rinnasteiseen tutkimukseen [26, 40].

Kolmas ja käytännössä varmasti tärkein tietojenkäsittelyteorian ja biologian yhteistyösuunta ovat *bioinformatiikan* ongelmat [9, 16, 19]. Biologista genomi- ja proteomidataa kertyy uusien, automatisoitujen laboratoriotekniikoiden ansiosta tietokantoihin nykyisin valtavaa vauhtia, ja tämän datan tehokas hyödyntäminen on erittäin haastava tietojenkäsittelyongelma. Monet tietojenkäsittelyteoreetikot, esimerkiksi aiemmin NP-täydellisyyden teorian kehittäjänä mainitsemani Richard Karp, ovatkin viime vuosina siirtyneet tutkimaan bioinformatiikan kysymyksiä.

Mielenkiintoista on huomata, että monipuolinen Alan Turing tunnetaan biologien keskuudessa ennen muuta ansioistaan morfogeneesin ns. *Turingin mallin* kehittäjänä; kaikki biologit tuskin edes tietävät hänen laskennallisista harrastuksistaan. Esimerkiksi J. D. Murrayn matemaattisen biologian oppikirjan mukaan [35, s. 238] Turingin morfogeneesiartikkeli vuodelta 1952 [48] on "one of the most important papers in theoretical biology in [the 20th] century." Ympyrä siis sulkeutuu sikäli että biologia on myös tietojenkäsittelyteoriassa palaamassa sille kuuluvaan rooliin tutkimuksen innoittajana.[4]

Turingin perintö on siten hyvissä käsissä: tietojenkäsittelyteorian tutkimus etenee edelleen yhtä uteliaana ja monessa suunnassa uutta luovana kuin Turinginkin töissä. Alalla on myös Suomessa ansiokkaat perinteet,[5] ja kunhan TKK:n *Tietojenkäsittelyteorian perusteet* -kurssilla ja muiden yliopistojen vastaavilla opintojaksoilla

[4]Sivuutan tässä esityksessäni ajan- ja tilanpuutteen vuoksi kokonaan tietotekniikan toisen pioneerin John von Neumannin tutkimukset itseään uusintavista soluautomaateista [51] ja pohdinnat aivojen ja tietokoneen vastaavuuksista [50], samoin kuin muutenkin neurotieteen ja tietojenkäsittelyn 1940-luvulta asti jatkuneen tiiviin vuorovaikutuksen (esim. [23, 29, 36]).

[5]Suomi on esimerkiksi pitkään ollut ykkössijalla laskettaessa tietojenkäsittelyteorian eurooppalaisen kattojärjestön *European Association for Theoretical Computer Science*'n (`http://www.eatcs.org`) jäsenmäärää suhteessa väkilukuun; tosin absoluuttiselta jäsenmäärältään Suomi oli vuoden 2002 tilastossa vasta kymmenes.

lukuisia hyödyllisiä käytännön taitoja oppineista opiskelijoista se kyvykkäin yksi prosentti kiinnostuu myös alan jatko-opinnoista, voimme mekin hyödyntää meille kuuluvan perintöosan tästä aartesta.

Viitteet

[1] Aarts, E. and Korst, J. *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing.* J. Wiley & Sons, Chichester 1989.

[2] Aarts, E. and Lenstra, J. K. (Eds.), *Local Search in Combinatorial Optimization.* J. Wiley & Sons, New York NY, 1998.

[3] Adleman, L. Molecular computation of solutions to combinatorial problems. *Science 266* (11 Nov. 1994), 1021–1024.

[4] Aho, A. V., Sethi, R. and Ullman, J. D. *Compilers: Principles, Techniques, and Tools.* Addison-Wesley, Reading, MA, 1986.

[5] Aho, A. V. and Ullman, J. D. *The Theory of Parsing, Translation, and Compiling I–II.* Prentice-Hall, Englewood Cliffs, NJ, 1972.

[6] Aspray, W. *John von Neumann and the Origins of Modern Computing.* The MIT Press, Cambridge MA, 1990.

[7] Atallah, M. J. (Ed.) *Algorithms and Theory of Computation Handbook.* CRC Press, Boca Raton, FL, 1999.

[8] Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A. and Protasi, M. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties.* Springer-Verlag, Berlin Heidelberg, 1999.

[9] Baldi, P. and Brunak, S. *Bioinformatics: The Machine Learning Approach, 2nd Ed.* The MIT Press, Cambridge MA, 2001.

[10] Bäck, T. *Evolutionary Algorithms in Theory and Practice.* Oxford University Press, 1996.

[11] Chung, F. R. K. *Spectral Graph Theory.* American Mathematical Society, Providence RI, 1997.

[12] Clay Mathematics Institute. *Millennium Prize Problems,* May 24, 2000. Online: `http://www.claymath.org/Millennium_Prize_Problems/`

[13] Cook, S. A. The complexity of theorem proving procedures. *Proc. 3rd Annual ACM Symp. on the Theory of Computing,* 151–158. ACM, New York NY, 1971.

[14] Davis, M. *The Universal Computer: The Road from Leibniz to Turing.* W. W. Norton, New York NY, 2000. (Suom. *Tietokonen esihistoria.* ArtHouse, Helsinki, 2003.)

[15] Diffie, W. and Hellman, M. E. New directions in cryptography. *IEEE Trans. on Information Theory 22* (1976), 644–654.

[16] Durbin, R., Eddy, S., Krogh, A. and Michison, G. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids.* Cambridge University Press, 1998.

[17] Garey, M. R. and Johnson, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., San Francisco CA, 1979.

[18] Goldreich, O. *Foundations of Cryptography: Basic Tools.* Cambridge University Press, 2001.

[19] Gusfield, D. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology.* Cambridge University Press, 1997.

[20] Hartmanis, J. and Stearns, R. Computational complexity of recursive sequences. *Proc. 5th Annual Symp. on Switching Circuit Theory and Logical Design,* 82–90. IEEE, New York NY, 1964.

[21] Hartmanis, J. and Stearns, R. On the computational complexity of algorithms. *Trans. Amer. Math. Soc. 117* (1965), 285–306.

[22] Hastie, T., Tibshirani, R. and Friedman, J. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer-Verlag, New York NY, 2001.

[23] Haykin, S. *Neural Networks: A Comprehensive Foundation, 2nd Ed.* Prentice-Hall, Upper Saddle River NJ, 1999.

[24] Hirvensalo, M. *Quantum Computing.* Springer-Verlag, Berlin Heidelberg, 2001.

[25] Hodges, A. *Alan Turing: the Enigma.* Burnett Books & Hutchinson Publishing, London, 1983. (Suom. *Alan Turing, arvoitus.* Terra cognita, Helsinki, 2000.)

[26] Kallel, L., Naudts, B. and Rogers, A. (Eds.) *Theoretical Aspects of Evolutionary Computing.* Springer-Verlag, Berlin Heidelberg, 2001.

[27] Karp, R. M. Reducibility among combinatorial problems. In J. Traub (Ed.), *Complexity of Computer Computations,* pp. 85–104. Plenum Press, New York NY, 1972.

[28] Kearns, M. J. and Vazirani, U. V. *An Introduction to Computational Learning Theory.* The MIT Press, Cambridge MA, 1994.

[29] Kohonen, T. *Self-Organizing Maps, 3rd Ed.* Springer-Verlag, Berlin Heidelberg, 2000.

[30] Leeuwen, J. van, *Handbook of Theoretical Computer Science. Vol. A: Algorithms and Complexity. Vol. B: Formal Models and Semantics.* Elsevier, Amsterdam, 1990.

[31] Levin, L. A. Universal sorting problems. *Problems of Information Transmission 9* (1973), 265–266. (Transl. from Russian *Problemy Peredaci Informacii.*)

[32] Luby, M. *Pseudorandomness and Cryptographic Applications.* Princeton University Press, Princeton NJ, 1996.

[33] Lynch, N. *Distributed Algorithms.* Morgan Kaufmann Publishers, San Francisco CA, 1996.

[34] Motwani, R. and Raghavan, P. *Randomized Algorithms.* Cambridge University Press, 1995.

[35] Murray, J. D. *Mathematical Biology, 2nd Ed.* Springer-Verlag, Berlin Heidelberg, 1993.

[36] *Neural Information Processing Systems, 1987–.* Online: `http://nips.djvuzone.org/`.

[37] Nielsen, M. A. and Chuang, I. L. *Quantum Computation and Quantum Information.* Cambridge University Press, 2000.

[38] Păun, G., Rozenberg, G. and Salomaa, A. *DNA Computing: New Computing Paradigms.* Springer-Verlag, Berlin 1998.

[39] Reeves, C. R. and Rowe, J. E. *Genetic Algorithms: Principles and Perspectives. A Guide to GA Theory.* Kluwer Academic, Dordrecht, 2003.

[40] Reidys, C. M. and Stadler, P. F. Combinatorial landscapes. *SIAM Review 44* (2002), 3–54.

[41] Rivest, R., Shamir, A. and Adleman, L. A method for obtaining digital signatures and public-key cryptosystems. *Comm. of the ACM 21* (1978), 120–126.

[42] Salomaa, A. *Public-Key Cryptography.* Springer-Verlag, Berlin 1990.

[43] Schneier, B. *Applied Cryptography, 2nd Edition: Protocols, Algorithms, and Source Code in C.* J. Wiley & Sons, New York, NY, 1996.

[44] Shor, P. Algorithms for quantum computation: discrete logarithms and factoring. *Proc. 35th Ann. IEEE Symp. on Foundations of*

Computer Science, 124–134. IEEE Computer Society Press, Los Alamitos CA, 1994.

[45] Sippu, S. and Soisalon-Soininen, E. *Parsing Theory I–II.* Springer-Verlag, Berlin Heidelberg, 1988/1990.

[46] Tel, G. *Introduction to Distributed Algorithms, 2nd Ed.* Cambridge University Press, 2000.

[47] Turing, A. M. On computable numbers, with an application to the Entscheidungsproblem. *Proc. London Math. Soc. (2) 42* (1936–37), 230–265; correction ibid. 43, 544–546 (1937). Reprinted: M. Davis (Ed.), *The Undecidable,* pp. 115–154. Raven Press, New York NY, 1965. Online: `http://www.abelard.org/turpap2/turpap2.htm`

[48] Turing, A. M. The chemical basis of morphogenesis. *Phil. Trans. R. Soc. London B 237*, 37–72 (1952).

[49] Vazirani, V. V. *Approximation Algorithms.* Springer-Verlag, Berlin Heidelberg, 2001.

[50] von Neumann, J. *The Computer and the Brain.* Yale University Press, New Haven CT, 1958.

[51] von Neumann, J. *Theory of Self-Reproducing Automata* (ed. A. W. Burks). University of Illinois Press, Urbana IL, 1966.