Tietojenkäsittelytiede **23** Kesäkuu 2005 sivut 42–59
Toimittaja: Antti Valmari

Visuaalinen algoritmisimulaatio ja sen sovelluksia

Ari Korhonen
Teknillinen korkeakoulu
Tietotekniikan osasto
Ari.Korhonen@hut.fi

Tiivistelmä

Ohjelmistojen havainnollistaminen on eräs ohjelmistotekniikan tutkimussuunnista. Sen sovellusalueita ovat mm. integroidut ohjelmointiympäristöt, visuaalinen testaus ja virheenjäljitys sekä opetuskäyttö.

Eräs keskeinen tutkimuksen kohde tällä hetkellä on kehittyneiden vuorovaikutustekniikoiden tuominen osaksi havainnollistamiseen liittyvää prosessia. Tällöin työkalujen ja sovellusten käyttäjä ei ole pelkästään passiivinen sivustakatsoja, vaan hänellä on aktiivinen rooli esimerkiksi visualisaation tuottamisessa. Tällaisille vuorovaikutustekniikoille on käyttöä mm. opiskelussa, jolloin oppijan tehtävänä voi olla esimerkiksi algoritmianimaation tuottaminen. Tuotetun animaation oikeellisuutta voidaan verrata vastaavaan algoritmisesti tuotettuun animaatioon ja käyttäjälle voidaan antaa tämän perusteella automaattisesti palautetta suorituksesta.

Eräs aktivoiva vuorovaikutustekniikka on algoritmisimulaatio, jossa käyttäjä manipuloi kuvaruudulla graafisia elementtejä simuloiden jonkin algoritmin toimintaa. Tässä artikkelissa raportoidaan Teknillisessä korkeakoulussa saatuja tuloksia, kun tätä tekniikkaa on sovellettu tietorakenteiden ja algoritmien opetuksessa käytetyissä TRAKLA- ja TRAKLA2-oppimisympäristöissä. Tutkimuksissa on mm. havaittu, että menetelmä sopii erityisen hyvin verkko-opiskeluun eikä oppimistuloksissa ole eroa verrattuna opiskelijoihin, jotka tekevät vastaavia tehtäviä luokkaopetuksessa.

1 Johdanto

Nykyaikaisissa käyttöliittymissä grafiikan ja animaatioiden käyttö on arkipäivää. Hyvänä esimerkkinä niiden täysimittaisesta käytöstä ovat pelit. Kyse ei ole pelkästään esteettisistä arvoista, vaan visuaalisuus tuo mukanaan lisäarvoa, jota ei perinteisillä tekstipohjaisilla käyttöliittymillä voi saavuttaa. Tässäkin pätee vanha viisaus: kuva kertoo enemmän kuin tuhat sanaa. Osa ohjelmoijista työskentelee kuitenkin edelleen ohjelmointiympäristöissä, joissa ohjelmakoodi esitetään yhdellä yksivärisellä kirjasintyypillä, sitä muokataan yhdessä ikkunassa ja ohjelman suoritusta yritetään seurata kirjoittamalla `print`-lauseita ohjelmakoodin sekaan.

Ohjelmistojen havainnollistaminen (*Software Visualization*) on ohjelmistotekniikan haara, joka tutkii erilaisten ha-

vainnollistamistekniikoiden käyttöä tietorakenteiden ja algoritmien sekä ohjelmakoodin ja sen kontrollivuon esittämiseen [25]. Tällä hetkellä tutkimus on vilkasta erityisesti visuaalisten esitystekniikoiden saralla, jossa keskeisinä keinoina ovat mm. grafiikan ja animaation käyttö. Olen kuitenkin valinnut käännökseeni sanan havainnollistaminen ("visualisointi" sijasta) painottamaan sitä tosiasiaa, että englannin kielen sana visualization ei rajaa myöskään esimerkiksi äänen hyödyntämistä pois [23]. Havainnollistaminen saa siis tässä laajemman merkityksen kuin pelkkä visualisointi. Käytän jatkossa sanaa visualisointi suppeammassa merkityksessä tarkoittamaan erityisesti kuvien käyttämistä havainnollistamiseen. Lisäksi käännöksessä ohjelmisto-sana on monikossa alleviivaamassa sitä seikkaa, että kyse ei ole pelkästään yksittäisten ohjelmien tai algoritmien tasolla tapahtuvasta tarkastelusta vaan tutkimuskohteena ovat myös laajojen ohjelmistojen havainnollistamistarpeet.

Ohjelmia ja niiden toimintaa on käytännössä pyritty havainnollistamaan yhtä kauan kuin ohjelmia on puettu tietokoneen ymmärtämään muotoon. Tästä esimerkkinä Goldsteinin ja von Neumannin vuokaavioesitykset konekielen havainnollistamiseen [2]. Kun ihminen kirjoittaa tietokoneohjelmaa, hänellä on mielessään jokin mentaalinen malli eli käsitys siitä kuinka ohjelma toimii. Ohjelmien havainnollistamisen perusideana voidaankin ajatella olevan tämän mentaalisen mallin selittäminen visuaalisin keinoin. Toisaalta tavoitteena on muodostaa visualisaation avulla mentaalinen malli visualisoinnin kohteesta. Visualisaation lähtökohtana voi olla esimerkiksi valmis ohjelma, jolloin pyrkimyksenä on havainnollistaa epäsuorasti se mentaalinen malli, joka ohjelmoijalla oli alun perin ohjelmaa kirjoittaessaan. Myös alkuperäinen ohjelmoija voi hyötyä tästä monin eri tavoin. Visualisaatio voidaan tuottaa automaattisesti lähdekoodista [9], jolloin se voi toimia osana ohjelman dokumentaatiota. Toisaalta, mikäli tuotettu visuaalinen malli ei vastaakaan ohjelmoijan omaa mentaalista mallia, on jossakin kohtaa prosessia tullut todennäköisesti virhe. Mitä aikaisemmassa vaiheessa virhe havaitaan, sitä parempi.

Menetelmät visualisaatioiden tuottamiseen voidaan jakaa karkeasti kahteen joukkoon. *Staattiset visualisaatiot*, kuten edellä mainittu vuokaavio, ovat olleet jo pitkään arkipäivää ja lähes kaikki nykyaikaiset ohjelmakoodin kirjoittamiseen tarkoitetut tekstinkäsittelyohjelmat tukevat ainakin joitakin tällaisia visualisointitekniikoita. Esimerkkeinä mainittakoon automaattinen sisentäminen; ohjelmointikielen varattujen sanojen, muuttujien, jne. näyttäminen eri väreillä; tai vaikkapa olio-ohjelmointikielellä toteutetun ohjelman luokkien välisten riippuvuuksien esittäminen verkkona.

Toisaalta, pelkkä käännösaikainen tieto ohjelmasta ei kerro kaikkea ohjelman toiminnasta, jolloin luonnollinen seuraava askel kehityksessä on tarkastella myös ohjelman suorituksen aikaista tilaa. Näin päädytään *dynaamisiin visualisaatioihin*, joiden tarve havaittiin myös hyvin varhaisessa vaiheessa [7, 8]. Niiden kehityksessä tehtiin kuitenkin todellinen harppaus vasta 1980-luvulla, jolloin graafiset päätteet alkoivat yleistyä. Syntyi uudenlainen tutkimusalue, jossa yksittäisten algoritmien toimintaa pyrittiin havainnollistamaan suorituksenaikaisesti. Kuuluisin esimerkki lienee *Sorting out Sorting* -video [1], jossa vertaillaan monipuolisesti eri järjestämismenetelmiä toisiinsa.

Toinen tapa luokitella erilaisia visualisaatioita on tarkastella sitä abstraktiotasoa, jolla ohjelman toimintaa pyritään ku-

vaamaan. Jos lähtökohtana on ohjelmakoodi ja sen (yksityiskohtainen) kuvaaminen, voidaan puhua ohjelmien visualisoinnista (*program visualization*). Toisaalta esimerkiksi edellä mainittu *Sorting out Sorting* -video ei keskity niinkään ohjelmakoodin esittämiseen vaan näyttää kuinka tietoaineisto järjestyy annetulla järjestämismenetelmällä askel askeleelta. Abstraktiotaso on korkeampi kuin ohjelmien visualisoinnissa ja tällöin puhutaankin algoritmien visualisoinnista. Video on lisäksi luonteeltaan dynaaminen visualisaatio, jolloin voidaan puhua algoritmianimaatiosta (*algorithm animation*). Edellä mainitun videon tekemiseen meni tekijöiden mukaan kolme vuotta, joten erääksi haasteeksi algoritmianimaatioiden tuottamisessa onkin noussut ajankäytön minimointi esitysmateriaalia valmistettaessa. Tänä päivänä vastaava voitaisiin tehdä esimerkiksi luentotilanteessa mielivaltaisilla syötteillä reaaliaikaisesti [6].

Motivaationa sekä varhaisissa ohjelmistojen havainnollistamiseen pyrkivissä esityksissä että nykyaikaisissa välineissä on siis ohjelman toiminnan ymmärtäminen ja selittäminen. Toisin sanoen tavoitteena on, että ihminen kykenee seuraamaan sitä logiikkaa, jolla ohjelma suoriutuu sille asetetusta tehtävästä (tai vaihtoehtoisesti miksi se ei suoriudu). Tästä seuraa, että ohjelmien havainnollistamiseen kykeneviä työkaluja voidaan soveltaa mm. ohjelmien kehittämiseen, niiden suunnitteluun ja määrittelyyn, testaukseen, virheenjäljitykseen, opiskeluun ja opetukseen. Monet integroidut kehitysympäristöt (IDE, *Integrated Development Environment*) sisältävät jo nyt komponentteja, joilla voidaan havainnollistaa työn alla olevan ohjelmiston toimintaa. Voidaan kuitenkin todeta, että niiden osalta viimeaikaisen tutkimuksen siirtyminen käytäntöön antaa odottaa itseään.

Opetuspuolella tulokset tuntuvat siirtyvän käytäntöön nopeammin. Tähän on luonnollisena selityksenä se, että suuri osa alan tutkimusryhmistä tekee tutkimusta yliopistoissa ja harjoittaa myös opetusta, jolloin tekijän on helppo siirtää omat tutkimustuloksensa suoraan käytäntöön. Alkuperäinen motivaatio tutkia alaa onkin usein lähtöisin omakohtaisista tarpeista. Myös opetuspuolella on kuitenkin omat haasteensa mm. tulosten saamiseksi laajaan käyttöön, koska tyypillisesti välineet leviävät yliopistoista toiseen hyvin hitaasti. Toisaalta opetuksen avuksi kehitettävien järjestelmien vaatimukset voivat olla hyvin erilaisia kuin yleisesti ohjelmistotekniikan tarpeisiin suunnitelluilla järjestelmillä, vaikka perusperiaatteet olisivatkin samat. Tämä näkyy erityisesti käyttäjän ja järjestelmän välisessä vuorovaikutuksessa ja siinä palautteessa, jota käyttäjän (opettaja tai opiskelija) tulee saada järjestelmältä.

Tarkastelen seuraavassa erään opetuskäyttöön tarkoitetun järjestelmän kehitysprojektia. Lähtökohtana on ollut opiskelijan aktivoiminen opiskelutilanteessa toteuttamalla ympäristö, jossa oppija voi ratkoa tietorakenteisiin ja algoritmeihin liittyviä tehtäviä visuaalisessa ympäristössä. Oppija ei ainoastaan katsele algoritmianimaatioita, vaan hänen tehtävänään on tuottaa oikeanlainen animaatio annetun tehtävän puitteissa. Tutkimuksen tavoitteena on ollut kehittää aivan uudentyyppinen vuorovaikutustekniikka käyttäjän ja havainnollistamiseen kykenevän järjestelmän välillä. Käytämme siitä nimeä *visuaalinen algoritmisimulaatio*.

Luvussa 2 kuvataan tämä vuorovaikutustekniikka ja esitellään toteutettu sovelluskehys, joka mahdollistaa visuaalisen algoritmisimulaation. Luvussa 3 kuvataan sovelluskehyksen avulla toteutetun varsinaisen järjestelmän toimintaperiaate sekä

sen ympärille kehitetty toimintaympäristö. Järjestelmän soveltuvuutta opetuskäyttöön on tutkittu kokeellisesti ja luvussa 4 esitetään saadut keskeiset tulokset. Lopuksi esitetään johtopäätökset ja eräitä jatkotutkimusaiheita luvussa 5.

2 Visuaalinen algoritmisimulaatio

2.1 Periaate

Visuaalisessa algoritmisimulaatiossa [11] keskeisessä asemassa on vuorovaikutus käyttäjän ja ohjelman välillä. Menetelmä voidaan nähdä algoritmianimaation laajennuksena, jossa tieto kulkee myös vastakkaiseen suuntaan — eli käyttäjältä ohjelmalle — verrattuna pelkkään algoritmianimaatioon, jossa se kulkee ohjelmalta käyttäjälle. Algoritmianimaatio on keskeisessä asemassa tässä vuorovaikutuksessa, koska visuaalisen algoritmisimulaation tuloksena voidaan nähdä syntyvän juuri algoritmianimaatio.

Algoritmianimaatio on prosessi, jossa *algoritmi* muokkaa suorituksen aikaisesti joukkoa tietorakenteita ja nämä muutokset havainnollistetaan *käyttäjälle* sopivalla käsitteellisellä tasolla. Algoritmianimaatio voi sisältää myös koodianimaation, jossa algoritmin suoritus kytketään askel askeleelta tietorakenteissa tapahtuviin muutoksiin. Tähän tarkoitukseen on kehitetty suuri määrä järjestelmiä, joista mainittakoon esimerkkeinä Animal [24], JAWAA [22] ja Jeliot 3 [21].

Visuaalinen algoritmisimulaatio on prosessi, jossa *käyttäjä muokkaa* joukkoa *todellisia tietorakenteita* oletetun *algoritmin suorituksen mukaisesti* sopivan graafisen *käyttöliittymän avulla*. Tyypillisesti tämä tarkoittaa graafisten entiteettien — kuten esimerkiksi verkon solmujen, niissä olevien avainten ja solmujen välisten kaarien — operoimista vedä ja pudota -tekniikalla. Käyttäjä ei kuitenkaan muokkaa pelkästään käyttöliittymän graafisia elementtejä vaan epäsuorasti myös todellisia tietorakenteita, jolloin käyttäjän tekemä operaatio välittyy alla olevalle tietorakenteen toteutukselle, jonka jälkeen sitä vastaava visualisaatio päivittyy automaattisesti. Operaatioiden toiminnallisuutta ei ole mitenkään rajoitettu, koska alla olevaan tietorakenteeseen voidaan kohdistaa mikä tahansa sitä muokkaava algoritmi, jonka käyttäjä voi aktivoida yksinkertaisella käyttöliittymäoperaatiolla. Käyttäjä voi esimerkiksi lisätä uusia avaimia AVL-puun tyhjiin lehtisolmuihin tai aktivoida sopivilla painikkeilla AVL-puun rotaatiooperaation (palaamme tähän esimerkkiin myöhemmin kuvan 1 yhteydessä).

Kannattaa huomata, että visuaalisessa algoritmisimulaatiossa käyttäjä ei kirjoita lainkaan ohjelmakoodia. Toisaalta, mikäli simulaation seurauksena syntyisi ohjelmakoodia, voitaisiin puhua visuaalisesta ohjelmoinnista.

Algoritmisimulaation seurauksena syntyy algoritmianimaatio vastaavalla tavalla kuin jos tietorakenteita olisi muokannut jokin tietokoneohjelma. Tällöin kaikki se toiminnallisuus, joka on totuttu näkemään algoritmianimaatioiden yhteydessä, on yhtä lailla käytettävissä myös ympäristöissä, joissa keskeisenä vuorovaikutustekniikkana on visuaalinen algoritmisimulaatio. Esimerkiksi opetuskäytössä eräs keskeinen toiminnallisuus on animaation ajaminen etu- ja takaperin. Tästä on hyötyä kun halutaan tarkastella jotakin yksittäistä kohtaa (tietorakenteen peräkkäisiä tiloja) algoritmin suorituksessa. Edelliseen tilaan ja siitä takaisin on helppo siirtyä suorittamatta koko algoritmia uudelleen (kuten tyypillisesti on tapana esimerkiksi virheenjäljittimissä).

2.2 Matrix

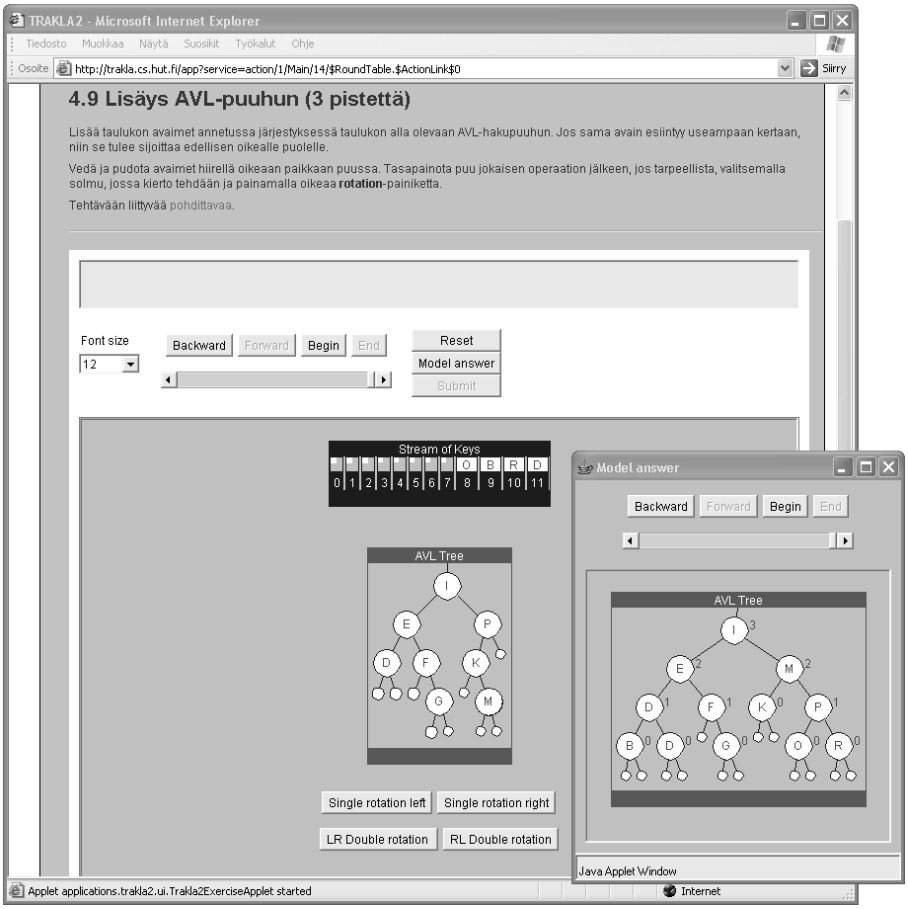
Visuaalisen algoritmisimulaation käyttömahdollisuudet ovat laajat. Tästä syystä Teknillisen korkeakoulun (TKK) Tietojenkäsittelyopin laboratoriossa lähdettiin 2000-luvun vaihteessa kehittämään eräässä projektissa sovelluskehystä — ei pelkästään yhden, vaan usean sovelluksen tarpeisiin. Ensisijainen motiivi oli kuitenkin kehittää opiskelijoille tarjottavien automaattisesti tarkastettavien kotitehtävien tarkastukseen sopiva järjestelmä Tietorakenteet ja algoritmit -kurssille. Tätä varten kehitettiin ensin Matrix-sovelluskehys, jonka pohjalta syntyi uusi TRAKLA2-järjestelmä korvaamaan viime vuosikymmenen alussa käyttöön otettu TRAKLA[1]. [18]

Matrix on Javalla toteutettu sovelluskehys, joka mahdollistaa mm. algoritmisimulaatiotehtävien teettämisen opiskelijoilla TRAKLA2-ympäristössä. Menetelmä perustuu visuaaliseen algoritmisimulaatioon, jossa oppijan tehtävänä on simuloida algoritmin toimintaa sopivalla syötteellä. Esimerkiksi tehtävänä voi olla annettujen avainten lisääminen yksi kerrallaan alun perin tyhjään AVL-puuhun kuten kuvassa 1. Tehtävä ratkaistaan lisäämällä avaimet yksi kerrallaan oikeisiin paikkoihin visualisaatiossa, joka esitetään binääripuuna. Rotaatiot voidaan tehdä jokaisen lisäyksen jälkeen joko päivittämällä solmujen välisiä kaaria (tähän on itse asiassa erillinen tehtävä, jossa harjoitellaan vain rotaatioita) tai annetulla valmiilla rutiinilla, jolloin käyttäjän tulee valita hiirellä alin epätasapainoinen solmu ja painikkeella oikea rotaatio.

Jotta jokaista tietorakennetta tai tehtävää varten ei tarvitsisi toteuttaa uutta visualisaatiota on Matrix-sovelluskehyksen perusperiaatteena tarjota sovellusohjelmille kattava joukko erilaisia visualisaatioita ja niitä vastaavat tietorakenteiden toteutukset, joista käytämme nimeä *talletusrakenteet*. Talletusrakenteita on mahdollista luoda lisää toteuttamalla uudelle tietorakenteelle haluttuja visualisaatioita vastaavat rajapinnat. Nykyinen talletusrakenteiden joukko riittää kuitenkin kattamaan lähes kaikki mahdolliset tietorakenteiden toteutukset. Kyse on lähinnä esitystavasta.

Järjestelmä tarjoaa myös mahdollisuuden räätälöidä tietorakenteiden esitystapaa paremmin tarkoitukseen sopivaksi, mutta koska rakenne piirretään ruudulle automaattisesti, niin on luonnollista, että aivan kaikkia tarpeita on mahdotonta tyydyttää. Esitysmuotojakin on toki mahdollista lisätä, mutta se on huomattavasti työläämpää kuin valmiiden komponenttien uudelleenkäyttö. Tällä hetkellä talletusrakenteina on valmiiksi toteutettu taulukko, linkitetty lista, binääripuu, yleinen puu, sekä verkko. Näiden avulla voidaan toteuttaa teoriassa mikä tahansa abstrakti tietotyyppi, koska rakenteet voivat sisältää rekursiivisesti toisia rakenteita. Esimerkiksi edellä mainittu AVL-puu voidaan toteuttaa helposti binääripuun avulla laajentamalla vastaavan talletusrakenteen toiminnallisuutta. Kuvassa 2 esitetty B-puu on puolestaan toteutettu sisäkkäisten talletusrakenteiden avulla siten, että jokainen B-puun solmu (yleinen puu) sisältää taulukon.

[1] TRAKLA on akronyymi sanoista TietoRakenteet ja Algoritmit; KotiLaskujen Arvostelu.



Kuva 1: TRAKLA2-tehtävä, jossa tehtävänantona on “Lisää taulukon avaimet annetussa järjestyksessä taulukon alla olevaan AVL-puuhun. Jos sama avain esiintyy useampaan kertaan, niin se tulee sijoittaa edellisen oikealle puolelle”. Simulaatiossa käyttäjä on lisännyt ensin kolme avainta (*E*, *I* ja *P*) alun perin tyhjään AVL-puuhun sekä tehnyt tarvittavan rotaation juuressa valitsemalla ensin solmun *E* ja sen jälkeen painikkeen `Single rotation left`. Tämän jälkeen tehtävä on jatkunut avainten *D*, *K*, *F*, *G* ja *M* lisäämisellä, jolloin on saavutettu kuvan tilanne. Rakenne tasapainotettaisiin valitsemalla solmu *P* ja painike `LR double rotation`. Etualalla on tehtävän malliratkaisun (`Model answer`) lopputila. Muita kuin näkyvissä olevia tiloja voi selata sekä omassa että malliratkaisussa rakenteiden yläpuolella olevilla `Backward`-, `Forward`-, `Begin`- ja `End`-painikkeilla, jolloin omaa ratkaisua voi verrata malliratkaisuun. Malliratkaisun katsomisen jälkeen tehtävää ei voi enää palauttaa (painike `Submit` ei ole enää aktiivinen) ennen kuin tehtävä on alustettu uusilla alkuarvoilla (`Reset`).

3 TRAKLA2

TRAKLA2 on verkkoympäristö tietorakenteisiin ja algoritmeihin liittyvien algoritmisimulaatiotehtävien jakeluun, tehtävien ratkomiseen ja palauttamiseen, tehtävien automaattiseen tarkastamiseen ja palautteen antamiseen, tehtäviin liittyvän oppimateriaalin levittämiseen, pistekirjanpitoon jne. Järjestelmä otettiin TKK:lla käyttöön vuonna 2003 ja se korvasi kokonaan vanhemman TRAKLA-järjestelmän vuonna 2004, jolloin se oli ensimmäistä kertaa käytössä myös Turun yliopistossa (TY) sekä testikäytössä Åbo Akademissa. [16]

Tehtävien tekemiseen tarkoitettu osa järjestelmää perustuu Matrix-sovelluskehykseen. Sen ympärille on rakennettu verkkoympäristö, josta tehtävät voidaan ladata Java-sovelmana (ks. kuva 1). Sovelma ottaa yhteyttä palvelimeen, jolle tehtävät myös palautetaan. Palvelin on yhteydessä verkkoympäristön perustana olevaan tietokantaan, jonne talletetaan mm. opiskelijan saamat pisteet tehtävistä. Näin opiskelija voi seurata omaa etenemistään verkkosivuston kautta.

TRAKLA2-ympäristössä opiskelija kirjautuu järjestelmään opiskelijanumerolla ja itse valitsemallaan salasanalla. Pääsivulla kerrotaan millä kurssilla opiskelija on ja mitä tehtäväkierroksia kurssiin kuuluu. Tyypillisesti tehtävät on jaettu kierroksiin aihepiireittäin. Jokaisella tehtäväkierroksella on määräaika, johon mennessä tehtävät tulee palauttaa. Tällä hetkellä tehtäviä on mm. seuraaviin aihepiireihin: perusalgoritmit, järjestämisalgoritmit, prioriteettijonot, hakupuut, hajautus ja verkot. Liitteessä A on tarkempi luettelo tehtävistä selityksineen. Kunkin kierroksen tehtävät voi tehdä haluamassaan järjestyksessä.

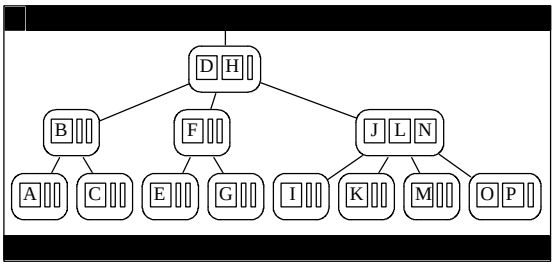
Valitsemalla tehtävän opiskelija siirtyy Java-sovelmaan, jossa tehtävä alustetaan henkilökohtaisilla alkuarvoilla. Kuvassa 1 näitä alkuarvoja vastaavat taulukossa olevat avaimet, jotka on valittu siten, että tehtävässä esiintyy aina vähintään yksi yksinkertainen rotaatio ja yksi kaksinkertainen rotaatio. Tehtävän ratkaistuaan opiskelija palauttaa sen palvelimelle. Samalla tehtävä arvioidaan automaattisesti vertaamalla opiskelijan ratkaisua malliratkaisuun, joka on tuotettu algoritmisesti. Palautteena opiskelija saa oikeiden simulaatioaskelien lukumäärän sekä ratkaisun oikeellisuuden perusteella annetun pistemäärän.

Järjestelmien (sekä TRAKLA että TRAKLA2) ympärillä on toteutettu useita erilaisia koeasetelmia ja tutkimuksia. Käyttökokemuksia ja saavutettuja pisteitä on seurattu lähes vuosittain. Sen lisäksi on järjestetty laajempia kyselyjä erityisesti vuosina, jolloin järjestelmää on voimakkaasti kehitetty. Seuraavassa on lueteltu tärkeimmät tutkimukset:

1991 Käyttäjäkysely ja tenttiarvosanojen vertailututkimus edelliseen vuoteen, jolloin vastaavaa järjestelmää ei vielä ollut [3].

1997 Käyttäjätyytyväisyyskysely, jolloin järjestelmään tuotiin ensi kertaa verkkopohjainen käyttöliittymä [10, 12].

2001 Interventiotutkimus, jossa seurattiin opiskelijoiden oppimistuloksia kolmessa satunnaisesti valitussa ryhmässä. Yksi ryhmä teki tehtäviä verkossa ja oppimistuloksia verrattiin toiseen vastaavia tehtäviä luokkaopetuksessa tekevään ryhmään. Lisäksi kolmas ryhmä teki vaativampia tehtäviä (joita ei voida tarkastaa automaattisesti) luokkaopetuksessa [14].



Kuva 2: B-puu (2-3-4-puu), jonka esitysmuoto koostuu yleisestä puusta, jonka solmuihin on upotettu 3-paikkaisia taulukoita. Taulukoiden alkioina on avaimia. Kuva on tuotettu MatrixPro-työkalulla [5] (joka myös perustuu Matrix-sovelluskehykseen) tallentamalla näkymä texDraw-formaatissa [4]. Lopputulos voitiin liittää tähän artikkeliin sellaisenaan ilman mitään jatkokäsittelyvaiheita.

2003 Käyttäjätyytyväisyys- ja vertailukysely (TRAKLA2:n koekäyttö TRAKLA:n rinnalla) [15].

2004 Käyttäjätyytyväisyyskysely sekä verkkopohjaiseen opiskeluun liittyvä asennekysely TY:ssä ja käytettävyystutkimus ÅA:ssa [16].

2005 Meneillään on vuoden 2001 koeasetelman kaltainen tutkimus TRAKLA2:lla (tosin vain 2 ryhmällä).

Lisäksi järjestelmää seurataan jatkuvasti pitkittäistutkimuksen avulla [20], jossa opiskelijoiden saavuttamia suhteellisia pistemääriä tilastoidaan vuosittain. Ennen vuotta 2003 laaditut tilastot liittyvät vanhaan TRAKLA-järjestelmään. Vuoden 2003 jälkeen tehdyt tilastot liittyvät puolestaan uuteen TRAKLA2-järjestelmään. Vuonna 2003 käytössä olivat molemmat järjestelmät.

Alusta asti on ollut selvää, että opettajan näkökulmasta järjestelmä tuo sellaista lisäarvoa opetukseen, jota perinteisillä menetelmillä ei voida saavuttaa: opiskelijoilla voidaan teettää suuri määrä pakollisia harjoitustehtäviä, vaikka kurssin opiskelijamäärä olisi erittäin suuri. Kyse ei ole pelkästään rahasta vaan myös muista resursseista. Kontaktiopetus vaatii asiantuntevaa työvoimaa ja salitilaa, jota ei välttämättä ole tarjolla vaikka rahaa lähiopetuksen toteuttamiseen olisikin. Vaikka automaattisesti tarkastettavan tehtävän toteuttamiseen meneekin paljon enemmän aikaa kuin perinteisiä laskuharjoituksia suunniteltaessa[2], maksaa vaiva itsensä helposti takaisin, koska järjestelmä tarkastaa vastaukset automaattisesti. Lisäksi tehtävät ovat käytettävissä vuodesta toiseen, koska jokaiselle opiskelijalle voidaan räätälöidä yksilöllinen tehtävä. Näin esimerkiksi perinteisiin harjoituksiin liittyviä plagiointiongelmia ei ole.

Viime vuosina järjestelmän tuomaa lisäarvoa on yhä enemmän lähdetty arvioimaan myös oppijan kannalta. Järjestelmästä kerätään palautetta systemaattisesti, jolloin mahdolliset epäkohdat on voitu korjata jo kurssien aikana. Mm. tästä syystä järjestelmästä saatava palaute käyttäjätyytyväisyyskyselyissä on ollut erittäin kiittävää.

[2]TRAKLA2-järjestelmässä tämä aika vaihtelee tehtävästä riippuen muutamasta päivästä (tyypillinen tilanne) muutamaan viikkoon (täysin uudentyyppisen tehtävän toteuttaminen).

Taulukko 1: Ryhmien perustiedot. Opiskelijoiden lukumäärä ryhmässä on merkitty sarakkeeseen N. Jokainen ryhmä jaettiin lisäksi pienryhmiin, joiden lukumäärä on ilmoitettu sarakkeessa s ja joiden keskimääräinen koko vastaavasti sarakkeessa k. Tutkimuksessa tutkittavana muuttujana oli opetusmenetelmä.

Ryhmä	N	s	k	Opetusmenetelmä
A	372			algoritmisimulaatiotehtävät verkossa
B	77	3	26	algoritmisimulaatiotehtävät luokassa
C	102	2	51	analyyttiset laskuharjoitukset luokassa
$\sum$	550			

Toisaalta uuden järjestelmän myötä myös opiskelijalle tehtävästä annettavan palautteen laatuun on alettu kiinnittää huomiota. Esimerkiksi tehtävien mallivastaukset voidaan esittää askel askeleelta algoritmianimaatioina (ks. ikkuna etualalla kuvassa 1), kun aikaisemmin ne olivat tekstimuodossa ja sisälsivät vain lopputilan. Oppijan kannalta mahdollisuus saada palaute välittömästi on myös eräs etu, jota perinteinen luokkaopetus ei voi ainakaan tässä laajuudessa tarjota. Tehtävän ei myöskään tarvitse päättyä palautteen saamiseen, vaan opiskelija voi halutessaan (esimerkiksi mikäli vastaus oli virheellinen tai kertauksena myöhemmin) ratkaista tehtävän uudelleen uusilla lähtöarvoilla, jolloin mallivastauksen näkeminenkään ei pilaa oppimiskokemusta.

Tutkimuksilla on osoitettu, että itseopiskeluna verkossa suoritettavat tehtävät eivät johda ainakaan huonompiin oppimistuloksiin kuin mitä perinteinen opetus mahdollistaisi [14, 16]. Kun tähän lisätään vielä aika- ja paikkariippumattomuus voidaan todeta, että verkossa tapahtuva opiskelu algoritmisimulaatiotehtävien avulla tuottaa huomattavaa lisäarvoa myös oppijan näkökulmasta.

4 Tulokset

Tarkastelen seuraavassa erityisesti edellä mainittuja vuoden 2001 interventiotutkimusta, pitkittäistutkimusta sekä vuoden 2004 käytettävyystutkimusta. Kaikissa tutkimuksissa tulokset ovat olleet hyvin samansuuntaisia.

4.1 Interventiotutkimus

Vuonna 2001 kurssilla järjestettiin koeasetelma, jossa seurattiin opiskelijoiden oppimistuloksia kolmessa satunnaisesti valitussa ryhmässä A, B ja C. Ensimmäinen ryhmä ($|A| = 372$ opiskelijaa) teki tehtäviä verkossa ja oppimistuloksia verrattiin toiseen vastaavia tehtäviä luokkaopetuksessa tekevään ryhmään ($|B| = 77$). Lisäksi kolmas ryhmä ($|C| = 102$) teki vaativampia tehtäviä (analyyttisiä laskuharjoitustehtäviä, joita ei voida tarkastaa automaattisesti) luokkaopetuksessa. Ryhmät valittiin satunnaisesti opiskelijanumeron viimeisen merkin (tarkastusmerkki) mukaan. Luokkaopetukseen osallistuvat ryhmät jaettiin vielä pienempiin ryhmiin luokkatilan koon mukaan. Jokaista pienryhmää ohjasi kerralla kaksi tuntiassistenttia. Taulukkoon 1 on koottu ryhmien perustiedot.

Taulukko 2: Interventiotutkimuksen tulokset. Sarakkeissa K_1 ja K_2 ovat prosenttiosuudet opiskelijoista, jotka keskeyttivät kurssin heti alussa palauttamatta yhtään tehtävää sekä opiskelijoista, jotka keskeyttivät kurssin aikana, vastaavasti. K_3 on prosenttiosuus opiskelijoista, jotka läpäisivät tehtävät, mutta joko eivät osallistuneet tenttiin tai eivät läpäisseet sitä. P_{tentti} on prosenttiosuus opiskelijoista, jotka läpäisivät sekä tehtävät että tentin. TP on kunkin ryhmän keskimääräiset tenttipisteet (max = 70).

Ryhmä	K_1	K_2	K_3	P_{tentti}	TP
A	4,6%	8,9%	4,8%	81,7%	33,04
B	0,0%	11,7%	5,2%	83,1%	32,74
C	8,9%	25,7%	2,0%	63,4%	39,56

Taulukossa 2 on esitetty interventiotutkimuksen tulokset, jossa tarkasteltiin kahta muuttujaa ryhmien *A*, *B* ja *C* osalta. Ne olivat keskeyttäneiden opiskelijoiden määrä ja eri ryhmiin osallistuneiden opiskelijoiden tenttimenestys kurssin lopuksi. Ryhmien *A* ja *B* välillä ei ole merkitseviä eroja keskeyttäneiden kokonaismäärässä eikä oppimistuloksissa (t-testi, $p = 0,871$; p ilmoittaa sen todennäköisyyden, että jos ryhmien välillä ei ole eroa, niin mittauksiin tulee silti sattumien vuoksi ainakin testissä havaitun suuruinen ero). Sen sijaan jos oppimistuloksia vertaa ryhmään *C*, havaitaan että ryhmien välillä on merkitsevät erot ($p_{AC} = 0,000$ ja $p_{BC} = 0,005$). Näiden ryhmien vertailu käy kuitenkin mahdottomaksi valikoivan kadon[3] myötä, joka näkyy ryhmän *C* suuressa keskeyttäneiden määrässä ja vastaavasti paremmissa arvosanoissa.

4.2 Pitkittäistutkimus

Järjestelmää ja kurssia on kehitetty voimakkaasti vuosien saatossa ja siihen on kohdistunut lukuisia muutoksia, jolloin on luonnollista, että näiden muutosten seurauksia on tarkasteltu myös tilastollisesti. Seuraavaan listaan on kerätty tärkeimmät muutokset kurssilla vuosina 1993–2004.

1993 TRAKLA-järjestelmän tekstipohjainen (sähköpostiin perustuva) käyttöliittymä oli vakiintunut kurssin laskuharjoitustehtävien käyttöön. Kunkin tehtäväkierroksen tehtävät tuli palauttaa määräaikaan mennessä, jonka jälkeen niistä sai automaattisesti palautetta pisteinä. Saavuttamalla 90% tehtävien maksimipisteistä saattoi korottaa kurssin kokonaisarvosanaa yhdellä numerolla (arviointi 0–5). Arvosanoja 0 tai 5 ei kuitenkaan voinut korottaa.

1997 Luennoija vaihtui. Tehtävien uudelleenpalautus tuli mahdolliseksi, jolloin yksittäisen tehtävän saattoi palauttaa 3–5 kertaa siten, että välissä järjestelmä antoi palautetta ratkaisun oikeellisuudesta pisteinä. Järjestelmän www-selaimella toimiva graafinen käyttöliittymä otettiin koekäyttöön (37,5% opiskelijoista käytti uutta käyttöliittymää) [10].

1998 Kurssin arviointiperusteet muuttuivat siten, että tehtävistä saatu ar-

[3] Ryhmässä *C* tenttimään tuli suhteellisesti muita ryhmiä enemmän aiemmissa opinnoissaan menestyneitä opiskelijoita (taustamuuttuja, jonka suhteen satunnaisesti valituissa ryhmissä ei ollut aluksi eroja).

vosana vaikutti 30% kurssin kokonaisarvosanaan. Kaikista kurssin osasuorituksista piti kuitenkin saada hyväksytty arvosana. Graafinen käyttöliittymä tuli ensisijaiseksi käyttöliittymäksi (n. 70% opiskelijoista käytti sitä). Tehtävävalikoimaa laajennettiin ja vaikeutettiin (mm. tasapainotettuja hakupuita käsittelevät tehtävät tulivat uusina mukaan ja joitain helppoja tehtäviä jätettiin pois).

1999 Kurssin työmäärää lisättiin ottamalla ohjelmaan laskuharjoitustehtävien lisäksi ryhmätyönä tehtävä suunnittelutehtävä, josta sai palautetta ja jonka kurssin muut opiskelijat arvioivat vertaisarviointina.

2001 Interventiotutkimus. Ainoa vuosi, jolloin kaikki opiskelijat eivät tehneet samoja tehtäviä vaan mm. ryhmä *C* teki haastavampia analyyttisiä laskuharjoituksia luokassa.

2002 Luennoija vaihtui ja erilliset kurssit pääaineopiskelijoille (T-kurssi) ja muille (Y-kurssi) eriytettiin toisistaan sisällön osalta. T-kurssilla suunnittelutehtävästä luovuttiin ja se korvattiin perinteisillä luokassa pidettävillä analyyttisillä laskuharjoituksilla, joissa jokainen opiskelija teki tehtävät itsenäisesti.

2003 TRAKLA2 oli testikäytössä kahdella tehtäväkierroksella.

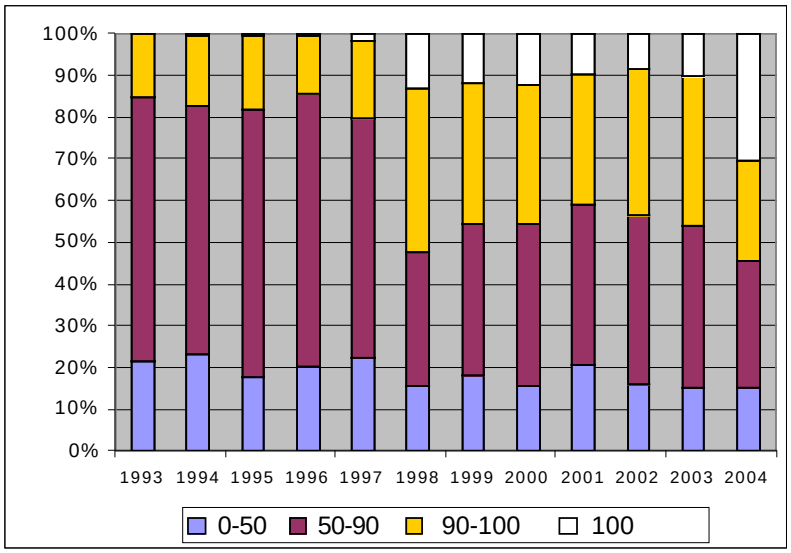
2004 TRAKLA2 korvasi kokonaan vanhan TRAKLA-järjestelmän. Tehtävälle sallittujen palautusten määrän yläraja poistettiin.

Opiskelijoiden suoriutumista tehtävistä voidaan tarkastella karkeasti jakamalla opiskelijat neljään joukkoon: opiskelijat, jotka 1) eivät saavuta tehtävistä vaadittavia minimipisteitä (50% maksimipisteistä), 2) saavuttavat minimipisteet, mutta eivät parasta arvosanaa (vähintään 50%, mutta alle 90% pisteistä), 3) saavuttavat parhaaseen arvosanaan vaadittavat pisteet, mutta eivät maksimipisteitä (vähintään 90%, mutta alle 100% pisteistä) ja 4) saavat täydet pisteet (100% pisteistä). Tätä aineistoa on kerätty systemaattisesti aina vuodesta 1993 lähtien ja se on esitetty kuvassa 3.

Pitkittäistutkimuksessa normaali vuosittainen vaihtelu ja siitä poikkeavat muutokset neljässä joukossa erottuvat selkeästi. Vuosien 1993–1996 jakaumissa ei ole tilastollisesti merkitseviä eroja (χ^2-testi, $p = 0,50$). Ensimmäiset selkeät erot ajoittuvat vuosien 1996–1998 väliin.

Vuonna 1998 ensi kertaa merkittävä määrä opiskelijoista kuului joukkoon 4. Lisäksi joukon 2 koko pieneni ja joukon 3 koko suureni. Näihin muutoksiin vaikuttivat seuraavat kolme eri seikaa yhdessä: i) vuonna 1997 järjestelmän graafinen käyttöliittymä otettiin ensi kertaa testikäyttöön ja samalla ii) mahdollistettiin tehtävien uudelleenpalauttaminen kesken kierroksen, saadun palautteen (pistemäärä) perusteella ja iii) vuonna 1998 kurssin arviointiperusteita muutettiin siten, että tehtävien tekeminen vaikutti huomattavasti enemmän (30%) kokonaisarvosanaan aikaisempaan verrattuna (jolloin arvosanaan lisättiin 1, jos ylsi joukkoihin 3 tai 4) .

Vuosien 1997 ja 1998 välistä eroa (χ^2-testi, $p < 0,001$) ei voi selittää tehtävien helpottumisella — itse asiassa ne vaikeutuivat, koska osa tehtävistä muuttui graafisen käyttöliittymän ansiosta lähes triviaaleiksi, jolloin ne poistettiin ja tilalle otettiin vaativampia tehtäviä (mm. tasapainotettuja hakupuita käsittelevät tehtävät).



Kuva 3: TRAKLA- ja TRAKLA2-tehtävistä saatujen arvosanojen jakauma. Opiskelijat on jaettu neljään joukkoon sen mukaan mihin heidän pistemääränsä yltävät.

Vuosien 1998–2003 jakaumissa ei ole tilastollisesti merkitseviä eroja (χ^2-testi, $p = 0,32$). Kuitenkin vuonna 2001 näyttäisi olevan poikkeuksellisen paljon keskeyttäneitä, mikä voidaan selittää koeasetelmalla. Saadun palautteenkin perusteella interventiotutkimuksen ryhmässä *C* olleet opiskelijat kokivat tehtävät vaikeammiksi verrattuna muiden ryhmien tehtäviin. Koeasetelma siis vaikutti saatuihin tuloksiin. Kato ryhmissä *A* ja *B* on täysin normaali verrattuna aikaisempiin vuosiin. Joukkoon 1 on vuoden 1997 jälkeen kuulunut reilut 15% opiskelijoista. Tämä on hyvin linjassa sen kanssa, että ryhmiin *A* ja *B* osallistuneista vain noin 18% ei läpäissyt kurssia (tenttiä tai TRAKLA-tehtäviä)[4].

Vuosien 2003 ja 2004 välillä on tilastollisesti merkitsevä ero pistemäärien jakaumassa (χ^2-testi, $p < 0,001$). Kyseessä on siirtyminen kokonaan TRAKLA2-järjestelmään. Samalla uudelleenpalautuspolitiikkaa muutettiin siten, että kun aikaisemmin opiskelija sai palauttaa tehtävään vastauksen vain rajoitetun määrän kertoja, vuonna 2004 siirryttiin käytäntöön, jossa mitään ylärajaa ei ollut. Sen sijaan tehtävän alkuarvot muuttuivat joka yrityksellä — opiskelija siis saattoi katsoa myös tehtävän mallivastauksen kesken kierroksen, mutta joutui sen jälkeen ratkaisemaan tehtävän kokonaan uusilla lähtöarvoilla. Nähty mallivastaus ei siis enää kelvannut uuden tehtävän ratkaisuksi. Aikaisemmin mallivastauksen sai vasta kierroksen päätyttyä. Seurauksena uudelleenpalautuspolitiikan muutoksesta kuvassa 3 nähdään muutos erityisesti maksimipisteet saaneiden osuudessa: lähes kolmannes opiskeli-

[4] Aineistossa on huomioitu vain kurssien ensimmäiset tentit, joten todellinen kurssin läpipääsemättömien joukko on ilmoitettua pienempi ja todelliset erot vielä pienempiä.

joista sai maksimipisteet vaikka siihen ei ollut mitään ulkoista motivaatiota (esimerkiksi suoraa vaikutusta arvosanaan).

Uudelleenpalautuspolitiikan muuttaminen ei ollut kuitenkaan täysin ongelmatonta. Suuri tai rajoittamaton määrä palautuskertoja aiheuttaa lieveilmiön, jossa eräät opiskelijat palauttavat tehtävän kerta toisensa jälkeen lyhyen ajan sisällä saamatta sitä silti oikein. TRAKLA2:n osalta hypoteesi oli ja on edelleen, että kyseistä lieveilmiötä ei esiintyisi, koska tehtävän alkuarvot vaihtuvat joka kerta. Lisäksi opiskelija saa tehtävästä palautetta ja ohjausta pisteiden sekä mallivastauksen muodossa. Näiden seikkojen ansiosta ilmiötä esiintyi vain vähän, joskin ongelma on edelleen olemassa [19]. Tämä ja edellisessä kappaleessa mainittu ilmiö (maksimipisteiden tavoittelu) ovatkin yhdessä mielenkiintoinen tulevaisuuden tutkimuskohde.

4.3 Asenne- ja käytettävyystutkimus

Vuoden 2004 keväällä TRAKLA2-järjestelmä otettiin käyttöön myös TY:ssä. Käyttöönottoa seurattiin mm. asennekyselyllä, jossa opiskelijoiden suhtautumista verkko-opetukseen kysyttiin ennen kurssia ja sen jälkeen. Kurssi oli toteutettu korvaamalla osa perinteisistä laskuharjoitustilaisuuksista TRAKLA2-kierroksilla verkossa. Lisäksi myöhemmin syksyllä ÅA:ssa järjestettiin käytettävyystutkimus, jossa tehtävien ratkaisemiseen tarkoitettua sovelmaa tutkittiin laboratorio-olosuhteissa havainnoimalla sen käyttöä tilanteessa, jossa oppija ratkaisi jälkijärjestystehtävää [16]. Viikko ennen käytettävyystestiä järjestettiin jälkijärjestyksiä koskeva esitentti ja viikko testin jälkeen jälkitentti, joiden tuloksia verrattiin toisiinsa.

TY:ssä tehtävistä saatujen pisteiden jakauma on vuoden 2004 osalta hyvin samankaltainen kuin TKK:lla. Itse asiassa siellä vielä suurempi osuus opiskelijoista kuului joukkoon 4, joskin kokonaisarvosana määräytyi hiukan eri perustein kuin TKK:lla. Vastaavia järjestelmiä ei ole TY:ssä aikaisemmin ollut käytössä, joten mielenkiinto kohdistui opiskelijoiden asenteisiin ympäristössä, jossa verkko-opetukseen liittyvät perinteet eivät vaikuttaneet mielipiteenmuodostukseen.

Opiskelijoilta kysyttiin ennen kurssia ($N = 96$) ja kurssin jälkeen ($N = 81$) i) mitä opiskelutapaa he suosivat henkilökohtaisesti ja ii) millainen opiskelutapa on heidän mielestään oppimisen kannalta tehokkain. Vaihtoehtoina olivat A) luokkaopetus, B) verkko-opetus (TRAKLA2) ja C) jokin näiden sekamuoto. Ennen kurssia mielipiteet jakautuivat lähes tasan kaikkien vaihtoehtojen kesken, joskin verkko-opetus sai vain 15% kannatuksen oppimistehon kannalta katsottuna. Kurssin jälkeen muutos oli selkeä: vahtoehto C voitti sekä ensimmäisessä kysymyksessä (50%) että toisessa kysymyksessä (73%). Verkko-opiskelu koetaan siis opiskelijoiden keskuudessa erittäin hyödylliseksi, mutta sen rajoitteet ymmärretään myös hyvin. Sillä ei ainakaan vielä voida korvata kokonaan perinteisiä laskuharjoituksia eli tehtäviä, joita ei voida automaattisesti tarkastaa.

Kaiken edellä mainitun lähtökohtana lienee kuitenkin järjestelmän käytettävyys. Tehdyssä käytettävyystutkimuksessa ei havaittu merkittäviä puutteita. Suurin osa ajasta (80%) käytettiin varsinaisen tehtävän tekemiseen, joten järjestelmän käyttö on helppoa ja helposti opittavissa. Esi- ja jälkitenttien perusteella järjestelmä myös soveltui asian opiskeluun. Esitentissä vain yksi koehenkilö viidestä osasi ratkaista tehtävän täysin oikein (ja

yksi osittain) ja jälkitentissä 4/5 (ja sama yksi osittain).

5 Johtopäätökset ja tulevaisuus

Ohjelmistojen havainnollistaminen on vilkas ohjelmistotekniikan tutkimusalue, jonka sovellukset ovat moninaiset. Tässä kirjoituksessa on esitelty visualisointitekniikoiden soveltamista opetusvälineiden kehittämiseen. Erityisesti on esitelty Matrix-sovelluskehys, jonka lähtökohtana on ollut aivan uudentyyppisen vuorovaikutuksen mahdollistaminen käyttäjän ja visualisointijärjestelmän välillä. Tätä visuaaliseksi algoritmisimulaatioksi kutsuttua menetelmää on sovellettu TRAKLA2-järjestelmässä, jonka perusidea kiteytyy algoritmisimulaatiotehtävissä. Järjestelmää ei ole pelkästään toteutettu, vaan sen soveltuvuutta tarkoitukseensa on tutkittu useiden koeasetelmien avulla.

Suurilla satunnaisesti valituilla opiskelijaryhmillä tehty interventiotutkimus on osoittanut, että oppimistuloksissa ei ole eroja ryhmien välillä, kun samoja tehtäviä teetetään joko verkossa itseopiskeluna tai luokkaopetuksessa tuntiassistenttien ohjauksessa. Myöskään kurssin keskeyttäneiden määrissä ei ole eroa. Sen sijaan teetettävillä tehtävillä on merkitystä.

Tämä johtopäätös on tehtävissä myös asennekyselystä, jossa opiskelijoiden mielipidettä kysyttiin suhteessa verkko-opiskeluun ja perinteiseen kontaktiopetukseen. Kontaktiopetuksesta ei oltu valmiita luopumaan ainakaan vielä, koska kaikenlaisia tehtäviä ei voida arvioida automaattisesti, joskin osa tehtävistä on selkeästi luonteeltaan sellaisia, että ne halutaan tehdä juuri verkossa eikä lähiopetuksessa.

Tuloksia voidaan yleistää pohtimalla sitä kuinka hyvin tehtävät kattavat laajuudeltaan ja syvyydeltään annetun kurssin. Mikäli tehtävillä voidaan kattaa koko kurssi, oppimisen kannalta ei ole merkitystä sillä tehdäänkö tehtävät verkossa vai lähiopetuksessa. Verkko-opiskelu tuo kuitenkin mm. aika- ja paikkariippumattomuuden ja palautteen nopeuden myötä sellaista lisäarvoa, että sitä suositaan tapauksissa, joissa sen järjestäminen on mahdollista. Parhaaksi vaihtoehdoksi koettiin järjestely, jossa molempien menetelmien hyvät puolet on yhdistetty.

Järjestelmän kehittymistä on seurattu myös pitkittäistutkimuksen avulla. Kurssin keskeyttäneiden osuus on muuttunut eri vuosina vähiten riippumatta järjestelmään tai kurssiin tehdyistä muutoksista. Ilmeisesti joka vuosi kurssille ilmoittautuu suhteellisesti sama määrä opiskelijoita, joilla ei todellisuudessa ole aikaa tai motivaatiota kurssille, joten järjestelmä ei helpota heidän tilannettaan. Tehtävät vaativat aikaa riippumatta siitä tehdäänkö niitä verkossa vai lähiopetuksessa.

Toisaalta automaattinen palaute, mahdollisuus palauttaa tehtäviä useaan kertaan ja kannustava arviointipolitiikka yhdessä ovat synnyttäneet mielenkiintoisella tavalla motivoituneen opiskelijaryhmän: tavoitteena on saada maksimipisteet kaikista tehtävistä, vaikka sillä ei olisikaan suoraa vaikutusta kurssin arvosanaan. Ulkoinen motivaatio saattaa vaikuttaa kuitenkin tähänkin, koska kurssin perinteisessä tentissä on tyypillisesti ollut myös ainakin yksi simulaatiotehtävä.

Monet algoritmien visualisointiin kehitetyt järjestelmät ovat jääneet prototyypeiksi, joiden käyttö on rajoittunut siihen (tyypillisesti akateemiseen) ympäristöön, jossa väline on alun perin kehitetty. Eräs keskeinen haaste onkin tuotteistaa järjestelmiä niin pitkälle, että niitä voi-

taisiin helposti ottaa käyttöön myös muualla. Opetusvälineistön osalta tähän kysymykseen nivoutuu myös kysymys oppimateriaalien vaihdosta. Eräs TRAKLA2-ympäristön kohdalla saatu tulos on se, että sitä on käytetty ja evaluoitu myös muissa yliopistoissa kuin TKK:lla, jossa se on alun perin kehitetty. Järjestelmä otettiin käyttöön Turun yliopistossa vuonna 2004 ja Åbo Akademissa sitä evaluoitiin käytettävyystestissä samana vuonna. Palaute on ollut erittäin myönteistä ja yhteistyö jatkuu.

Seuraavassa on lueteltu joitain keskeisiä tutkimussuuntauksia, joita parhaillaan tutkitaan tai on tarkoitus tutkia tulevaisuudessa:

1. Parhaillaan on meneillään vuoden 2001 koeasetelman kaltainen tutkimus TRAKLA2:lla kahdella eri opiskelijaryhmällä.

2. TRAKLA2-järjestelmän tehtävistä suurin osa on luonteeltaan sellaisia, että oppija simuloi annettua algoritmia annetulla syötteellä. Sovelluskehys mahdollistaa myös muunlaisten tehtävien toteuttamisen, jonka taustalle on kehitetty erityinen simulaatiotehtävien luokittelu [13]. Tämän luokittelun mukaisten tehtävien toteuttaminen järjestelmään on alkanut. Esimerkkinä mainittakoon tehtävä, jossa tulee värittää annettu (riittävän tasapainoinen) binäärinen hakupuu punamustaksi puuksi. Vastaukseksi kelpaa mikä tahansa mahdollinen väritys eikä ratkaisun tuottamiseksi tarvitse simuloida jotakin tiettyä algoritmia.

3. Tehtävistä saatava palaute perustuu oikeiden simulaatioaskeleiden määrään alusta laskettuna. Palautteen edelleenkehittämiseksi on perustettu tutkimushanke “Automatic Assessment and Feedback on Algorithm Simulation”, jossa tutkitaan mahdollisuuksia antaa yksityiskohtaisempaa palautetta ja mm. jatkaa ratkaisun arviointia virheellisen tilan jälkeen. Tämä mahdollistaisi humaanimman palautteen, jossa esimerkiksi yksittäinen huolimattomuusvirhe ratkaisun alussa ei veisi tehtävää kokonaan nollille. Toisaalta on kuitenkin pidettävä huoli, että virheen vuoksi triviaaliksi muuttuvasta tehtävästä ei anneta pisteitä (esimerkiksi lisättäessä alkioita AVL-puuhun täytyy silti tapahtua sekä yksinkertainen että kaksinkertainen rotaatio, jotta siitä voidaan antaa täydet pisteet). Tämä ja edellinen kohta edellyttävät uudenlaista algoritmiikkaa, joilla tehtäviä voidaan tarkastaa.

4. ÅA:n käytettävyystutkimuksessa käytettiin vain yhtä tehtävää. Jatkossa käytettävyyttä on tarkoitus tarkastella koko järjestelmässä ja kehittää saadun palautteen perusteella sitä eteenpäin. Tähän liittyen on esitetty aloitteita myös Joensuun yliopiston taholta.

5. Matrix-sovelluskehys mahdollistaa muitakin sovelluksia kuin TRAKLA2-tyyppisen oppimisympäristön. Sen pohjalta on jo kehitetty MatrixPro-ohjelmisto [5], jolla opettaja voi havainnollistaa tietorakenteita ja algoritmeja luentotilanteessa sekä MVT (Matrix Visual Tester) [17] visuaaliseen testaukseen ja virheenjäljitykseen. Näiden järjestelmien ja ideoiden jatkokehitys on työn alla.

Kiitokset

Kiitokset Kimmo Stålnackelle ja Jan Lönnbergille hyödyllisistä kommenteista tämän kirjoituksen aikaisempaan versioon. Kiitos myös Antti Valmarille monista selventävistä parannusehdotuksista lopulliseen versioon.

Viitteet

[1] R. M. Baecker. Sorting out sorting. Elokuva, 30 minuuttia, 1981.

[2] H. H. Goldstein & J. von Neumann. Planning and coding problems of an electronic computing instrument. *Collected Works*, ss. 80–115, 1947.

[3] J. Hyvönen & L. Malmi. TRAKLA — A system for teaching algorithms using email and a graphical editor. *Proceedings of HYPERMEDIA in Vaasa*, ss. 141–147, 1993.

[4] P. Kabal. TEXdraw — PostScript drawings from TEX. Verkkosivu, (12.4.2005). `http://www.tau.ac.il/cc/sivut/docs/tex-3.1415/texdraw_toc.html`.

[5] V. Karavirta, A. Korhonen, L. Malmi & K. Stålnacke. MatrixPro — A tool for on-the-fly demonstration of data structures and algorithms. *Proceedings of the Third Program Visualization Workshop*, ss. 26–33, The University of Warwick, UK, 2004.

[6] V. Karavirta, A. Korhonen, J. Nikander & P. Tenhunen. Effortless creation of algorithm visualization. *Proceedings of the Second Annual Finnish / Baltic Sea Conference on Computer Science Education*, ss. 52–56, 2002.

[7] K. C. Knowlton. *L^6: Bell telephone laboratories low-level linked list language.* 16 mm mustavalkoinen äänielokuva, 16 minuuttia, 1966.

[8] K. C. Knowlton. *L^6: Part II. an example of L^6 programming.* 16 mm mustavalkoinen äänielokuva, 30 minuuttia, 1966.

[9] D. E. Knuth. Computer-drawn flowcharts. *Communications of the ACM*, 6:555–563, 1963.

[10] A. Korhonen. *World wide web (www) tietorakenteiden ja algoritmien tietokoneavusteisessa opetuksessa.* Diplomityö, Teknillinen korkeakoulu, 1997.

[11] A. Korhonen. *Visual Algorithm Simulation*. Väitöskirja, Teknillinen korkeakoulu, 2003.

[12] A. Korhonen & L. Malmi. Algorithm simulation with automatic assessment. *Proceedings of The 5th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education*, ss. 160–163, Helsinki, Suomi, 2000. ACM Press, New York.

[13] A. Korhonen & L. Malmi. Taxonomy of visual algorithm simulation exercises. *Proceedings of the Third Program Visualization Workshop*, ss. 118–125, The University of Warwick, UK, 2004.

[14] A. Korhonen, L. Malmi, P. Myllyselkä & P. Scheinin. Does it make a difference if students exercise on the web or in the classroom? *Proceedings of The 7th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education, ITiCSE'02*, ss. 121–124, Århus, Tanska, 2002. ACM Press, New York.

[15] A. Korhonen, L. Malmi & P. Silvasti. TRAKLA2: a framework for automatically assessed visual algorithm simulation exercises. *Proceedings of Kolin Kolistelut / Koli Calling — Third Annual Baltic Conference on Computer Science Education*, ss. 48–56, Joensuu, Suomi, 2003.

[16] M.-J. Laakso, T. Salakoski, L. Grandell, X. Qiu, A. Korhonen & L. Malmi. Multi-perspective study of novice learners adopting the visual algorithm simulation exercise system TRAKLA2. *Informatics in Education*, 4(1):49–68, 2005.

[17] J. Lönnberg, A. Korhonen & L. Malmi. MVT — a system for visual testing of software. *Proceedings of the Working Conference on Advanced Visual Interfaces*, ss. 385–388, Gallipoli, Italia, 2004. ACM Press, New York.

[18] L. Malmi. Automaattinen tarkastaminen opetuksen apuvälineenä. *Tietojenkäsittelytiede*, 17:24–35, 2002. Tietojenkäsittelytieteen Seuran julkaisu.

[19] L. Malmi & A. Korhonen. Automatic Feedback and Resubmissions as Learning Aid. *Proceedings of 4th IEEE International Conference on Advanced Learning Technologies*, ss. 186–190, Joensuu, Suomi, 2004.

[20] L. Malmi, A. Korhonen & R. Saikkonen. Experiences in automatic assessment on mass courses and issues for designing virtual courses. *Proceedings of The 7th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education, ITiCSE'02*, ss. 55–59, Århus, Tanska, 2002. ACM Press, New York.

[21] A. Moreno, N. Myller, E. Sutinen & M. Ben-Ari. Visualizing programs with Jeliot 3. *Proceedings of the International Working Conference on Advanced Visual Interfaces*, ss. 373–376, Gallipoli, Italia, 2004. ACM Press, New York.

[22] W. Pierson & S. Rodger. Web-based animation of data structures using JAWAA. *Proceedings of the 29th SIGCSE Technical Symposium on Computer Science Education*, ss. 267–271, Atlanta, GA, USA, 1998. ACM Press, New York.

[23] B. A. Price, R. M. Baecker & I. S. Small. A principled taxonomy of software visualization. *Journal of Visual Languages and Computing*, 4(3):211–266, 1993. Elsevier.

[24] G. Rößling, M. Schüler & B. Freisleben. The ANIMAL algorithm animation tool. *Proceedings of the 5th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education, ITiCSE'00*, ss. 37–40, Helsinki, Suomi, 2000. ACM Press, New York.

[25] J. T. Stasko, J. B. Domingue, M. H. Brown & B. A. Price. *Software Visualization: Programming as a Multimedia Experience*. MIT Press, Cambridge, MA, 1998.

LIITE A: Tehtävät

TRAKLA2-järjestelmän kehittämiseen tähtäävän tutkimussivuston kautta löytyy ajantasaisin luettelo mahdollisista tehtävistä, katso `http://www.cs.hut.fi/Research/TRAKLA2/exercises/index.shtml`. Seuraavassa on luettelo tämän hetken tehtävävalikoimasta.

- Perusalgoritmit
  - 1 Puolitushaku
  - 2 Interpolaatiohaku
  - 3 Esijärjestys
  - 4 Esijärjestys pinon avulla
  - 5 Sisäjärjestys
  - 6 Jälkijärjestys
  - 7 Tasojärjestys
- Järjestämisalgoritmit
  - 8 Rekursiivinen pikajärjestäminen
  - 9 Rekursiivinen Radix Exchange Sort
- Prioriteettijonot
  - 10 Keon rakentaminen: lineaarisessa ajassa toimiva algoritmi
  - 11 Binäärikeko: lisäys ja pienimmän alkion poisto
- Hakupuut
  - 12–14 Binäärinen hakupuu: haku, lisäys ja poisto
  - 15 Virheellisesti toimiva binäärinen hakupuu
  - 16 Digitaalinen hakupuu: lisäys
  - 17 Radix search trie: lisäys
  - 18 Yksinkertainen rotaatio
  - 19 Kaksoisrotaatio
  - 20 AVL-puu: lisäys
  - 21 Punamusta puu: lisäys
  - 22 Punamustan puun värittäminen
  - 23 B-puu: lisäys
- Hajautus
  - 24 Lineaarinen kokeilu
  - 25 Neliöllinen kokeilu
  - 26 Kaksoishajautus
- Verkkoalgoritmit
  - 27 Leveyssuuntainen haku
  - 28 Syvyyssuuntainen haku
  - 29 Primin algoritmi
  - 30 Dijkstran algoritmi