

Teknillinen korkeakoulu
Tietotekniikan osasto
Tietotekniikan koulutusohjelma

Panu Silvasti

**Tilastollisen datan kerääminen algoritmisten
harjoitustehtäväsovelmien käytöstä**

Diplomityö, joka on jätetty opinnäytteenä tarkastettavaksi diplomi-insinöörin tutkintoa varten

Työn valvoja ja ohjaaja Professori Lauri Malmi

Espoo, 11. syyskuuta, 2003

Tekijä:	Panu Silvasti	
Työn nimi:	Tilastollisen datan kerääminen algoritmisten harjoitustehtäväsovelmien käytöstä	
English title:	Collecting statistical data of the usage of algorithmic exercise applets	
Päivämäärä:	11. syyskuuta, 2003	Sivumäärä: 58
Osasto:	Tietotekniikan osasto	
Professuuri:	Tik-106 Ohjelmistojärjestelmät	
Työn valvoja:	Professori Lauri Malmi	
Työn ohjaaja:	Professori Lauri Malmi	
<p>Tässä työssä esitetään menetelmä opiskelijoiden oppimista kuvaavan tilastollisen datan keräämiseen Teknillisen korkeakoulun (TKK) Tietojenkäsittelyopin laboratoriossa kehitetty TRAKLA2-opetusohjelman käytöstä. TRAKLA2 on ohjelma tietorakenteita ja algoritmeja koskevien harjoitustehtävien tekemiseen. TRAKLA2-harjoitustehtävät ovat WWW-selaimessa toimivia Java-sovelmia.</p> <p>Datan keruumenetelmän tekninen toteutus perustuu käyttöliittymäoperaatioita vastaavien Java-olioiden luomiseen TRAKLA2-sovelmassa ja niiden välittämiseen palvelimelle RMI-protokollaa käyttäen. Palvelimella data tallennetaan helposti analysoitavaksi tekstitiedostoksi. Harjoitustehtävän ratkaisu on tietorakenteiden graafisten esitysten muokkausta opetusohjelman käyttöliittymällä. Opiskelijoiden vastaussekvenssi sisällytetään oliorakenteseen, joka tallennetaan palvelimella Javan sarjallistamismekanismia käyttäen. Työssä esitetään myös työkalut ja menetelmät sekä tekstitiedostojen että sarjallistettujen oliorakenteiden muokkaamiseen siten, että datan analyysi voidaan suorittaa yleisillä tilasto-ohjelmilla, kuten Microsoft Excelillä.</p> <p>Opetusohjelman evaluointi on tärkeä osa sen kehitysprosessia. Opetusohjelman tärkein kriteeri on, että sen käyttäminen tukee ja edistää oppimista. Tutkimuksessa esitetään suuntaaviivat menetelmälle TRAKLA2-opetusohjelman evaluointiin tilastollisella analyysillä kerätystä datasta. Opetusohjelmaa on pystyttävä arvioimaan kokonaisuutena sekä myös käytettyjen opetusteknologioiden kannalta. TRAKLA2-opetusohjelmassa käytetyt opetusteknologiat ovat algoritmien visualisointi, animointi ja simulointi sekä automaattinen palaute. Tutkimuksessa esitetään suuntaaviivat yksittäisen opetusteknologian evaluointiin kerätyn datan tilastollista analyysiä käyttäen.</p> <p>Tutkimukseen sisältyy myös analyysi opiskelijoiden toiminnasta opetusohjelmassa TKK:n kevään 2003 Tietorakenteet ja algoritmit -kurssin aikana kerätystä datasta. Analyysi osoittaa, että opiskelijoiden harjoitustehtävissä suorittamien käyttöliittymäoperaatioiden tilastollisella analyysillä voidaan tehdä päätelmiä tehtävien keskinäisestä vaikeudesta.</p>		
Avainsanat:	tietojenkäsittelyopin opetus, opetusohjelmat, evaluaatiotutkimukset	
Ei lainata ennen:	Työn sijaintipaikka:	

Author:	Panu Silvasti	
Title of thesis:	Collecting statistical data of the usage of algorithmic exercise applets	
Finnish title:	Tilastollisen datan kerääminen algoritmisten harjoitustehtäväsovelmien käytöstä	
Date:	September, 11, 2003	Pages: 58
Department:	Department of Computer Science and Engineering	
Chair:	Tik-106 Software Technology	
Supervisor:	Professor Lauri Malmi	
Instructor:	Professor Lauri Malmi	
<p>In this study a method for collecting statistical data of the usage of educational software is presented. TRAKLA2 is educational software for solving exercises of algorithms and data structures. TRAKLA2 exercises are Java applets and can be used with a browser. TRAKLA2 is developed in the Laboratory of Information Processing Science at Helsinki University of Technology (HUT).</p> <p>The technical solution for collecting data is based on creating Java objects corresponding to the user interface operations. The objects are sent to a server through Java RMI protocol. The server saves basic information of the log entries into a text file, which is easy to analyze. More complex object structures, which contain student's answer sequence in the algorithmic exercise, are saved as serialized Java objects. Tools and methods for converting the data from text files as well as from Java objects into a format readable with statistical software, such as Microsoft Excel, are presented.</p> <p>Evaluation is an important part of the development process of educational software. The goal of evaluation must be studying how the usage of educational software affects students' learning. In this study research frames for evaluation studies of TRAKLA2 software by analyzing the collected data are presented. Educational technologies used in TRAKLA2 are algorithm visualization, animation, simulation, and automatic assessment and feedback. A research frame for studying the impact of individual learning technology into learning by statistical analysis of the collected data is presented.</p> <p>The study includes an analysis of the data collected during the data structures and algorithms course at HUT in spring 2003. The study concludes that it is possible to make deductions of the difficulty of the individual exercises in the TRAKLA2 system by statistically analyzing the number of user interface operations students perform.</p>		
Keywords:	computer science education, educational software, evaluation studies	
Not borrowable till:	Library code:	

Esipuhe

Kiitän esimiestäni ja ohjaajaani professori Lauri Malmia hänen ohjauksestaan ja kannustuksestaan. Hän löysi aina aikaa keskustelulle ja neuvonpidolle sekä osoitti suurta kiinnostusta työtäni kohtaan. Kiitos myös puitteiden luomisesta kirjoitustyölle, muiden töiden raivaamisesta sivuun ja joustavuudesta aikataulujen kanssa.

Haluan myös kiittää äitiä, isää, Villeä ja Hannaa tuestanne ja ymmärtäväisyydestänne kirjoitustyötäni kohtaan, vaikka itsellenekin oli kädet täynnä työtä juuri perustetun perheyriityksen ja talon rakennuksen parissa. Äitiä kiitän lämmöstä ja huolenpidosta sekä myös kielenhuoltoon liittyvistä neuvoista ja oppaista.

Tässä työssä keskeisellä sijalla on Tietojenkäsittelyopin laboratoriossa kehitetty Matrix-algoritmisimulaatiosovelluskehys ja TRAKLA2-opetusohjelma. Kiitos Matrix-tutkimusryhmälle ja sille kokoonpanolle, jolla TRAKLA2 tehtiin: Ari Korhoselle, Ville Karavirrälle, Pekka Mårdille Jussi Nikanderille, Harri Saloselle ja Petri Tenhuselle. On ollut hienoa olla mukana tekemässä töitä teidän kanssanne!

Kiitos myös monille muille kollegoille Tietojenkäsittelyopin laboratoriossa arvokkaasta keskusteluavusta sekä Satu Virtaselle avusta \LaTeX -järjestelmän kanssa.

Espoo, 11. syyskuuta, 2003

Panu Silvasti

Sisältö

1	Johdanto	1
2	Ohjelmoinnin opetusohjelmista	4
2.1	Yleistä	4
2.2	Visualisointi ohjelmoinnin oppimisessa	5
2.2.1	Ohjelman visualisointi	6
2.2.2	Algoritmien visualisointi	6
2.2.3	Visuaalinen ohjelmointi	7
2.3	Automaattinen palaute	8
2.3.1	Palautteen välittömyys	8
2.3.2	Palautteen esitystapa	9
2.3.3	Palautteen yksityiskohtaisuus	9
2.4	Automaattisesta tarkastuksesta	10
2.4.1	Ohjelmakoodin automaattinen tarkastus	10
2.4.2	Algoritmisten harjoitustehtävien tarkastus	11
2.4.3	Tietämyskantaan perustuva tarkastus	11
2.5	Adaptiivisuus	12
2.6	Ohjelmoinnin opetusohjelmien tutkimusmenetelmistä	12
3	Tietorakenteiden ja algoritmien opetusohjelmat TKO-laboratoriossa	14
3.1	TRAKLA	14
3.2	Scheme-Robo	15

3.3	TraklaEdit ja WWW-TRAKLA	16
3.4	Matrix-algoritmisimulaatiojärjestelmä	17
3.5	TRAKLA2	20
3.6	Tutkimustuloksia	24
4	Tutkimuksen pääkysymys ja ongelmat	25
4.1	Taustaa	25
4.2	Datan keräämisen motivaatio	26
4.2.1	Algoritmien oppimisen tutkiminen	26
4.2.2	Opetusteknologian sovellusten tutkiminen	27
4.3	Analysoitavat operaatiot	28
4.4	Datan keräämisen teknisen toteutuksen vaatimukset	29
4.5	Yhteenveto	30
5	Datan keräämisen toteutus	31
5.1	Kerättävä data	31
5.2	Tietomalli	33
5.3	Järjestelmäarkkitehtuuri	34
5.4	Analyysityökalut	36
6	Tulokset ja johtopäätökset	39
6.1	Algoritmien oppimisen tutkimisen työkalu	39
6.1.1	Algoritmien vertailu	39
6.1.2	Algoritmin osien keskinäisen vaikeuden analyysi	43
6.1.3	Johtopäätökset	46
6.2	Opetusteknologian tutkimuksen työkalu	47
6.3	Teknisen toteutuksen analyysi	48
6.3.1	Tehokkuus	49
6.3.2	Muunneltavuus	51

7 Yhteenveto	53
Kirjallisuutta	54

Luku 1

Johdanto

Teknillisen korkeakoulun (TKK) Tietojenkäsittelyopin (TKO) laboratoriossa opetetaan ohjelmoinnin perusteita kaikille korkeakoulun opiskelijoille. TKK:ssa opintonsa aloittaa vuosittain noin 1500 uutta opiskelijaa, joista suurin osa sisällyttää opintoihinsa ohjelmoinnin kursseja. Suurten opiskelijamäärien vuoksi TKO-laboratoriossa on tutkittu, kehitetty ja käytetty opettajien työmäärää vähentäviä opetusohjelmia jo yli kymmenen vuotta. Tietorakenteiden ja algoritmien peruskurssilla sähköpostipohjainen täysin automaattinen TRAKLA-kotitehtäväjärjestelmä saatiin käyttöön vuonna 1991. Automaattisen arvioinnin järjestelmien kehitystyö on kulkenut TRAKLA:n viitoittamaa polkua; TRAKLA:a on kehitetty eteenpäin, automaattisen arvioinnin järjestelmiä on kehitetty myös muillekin ohjelmoinnin kursseille ja automaattisen arvioinnin lisäksi algoritmien visualisointia käyttäviä kognitiivisia työkaluja on kehitetty. Tietorakenteiden ja algoritmien sekä ohjelmoinnin perusteiden opetusohjelmiin liittyvä kehitys- ja tutkimustyö tuottanut useita lopputöitä ja julkaisuja [14, 17, 18, 19, 20, 21, 22, 23, 24, 25, 27, 28, 29, 42], ja tämä diplomityötutkimus jatkaa omalta osaltaan tätä yli vuosikymmenen kestänyttä tutkimustyötä.

Vuoden 2003 keväällä käyttöön otettu tietorakenteiden ja algoritmien TRAKLA2-kotitehtäväjärjestelmä on TRAKLA:n seuraaja. WWW-selaimella käytettävä opetusohjelma perustuu algoritmien visualisointiteknoologiaan, jossa algoritmien tietorakenteet on visualisoitu eri tavoin. Opetusohjelma antaa opiskelijalle välitöntä palautetta hänen vastauksestaan. Vuosittain tietorakenteiden ja algoritmien peruskurssille osallistuu yli 500 opiskelijaa, jotka tekevät kurssin kotitehtäviä TRAKLA2-opetusohjelmalla.

TRAKLA:n ja vastaavien innovatiivisia ideoita ja opetusteknologioita esittelevien opetusohjelmien kehitysprosessi saattaa kestää vuosia ja sisältää useita kehitys- ja evaluointivaiheita. Oleellinen osa opetusohjelman evaluointia on sen tutkiminen, miten opetusohjelman käyttö edistää opiskelijan oppimista vai edistääkö lainkaan? Tämä herättää kysymyksen, miten opetusohjelman käytön ja opiskelijoiden oppimisen välistä yhteyttä voi tutkia? Ohjelmoinnin opetusohjelmista tehdyssä tutkimuksessa on esitetty tutkimusasetelmia, joissa on käytetty kysely- ja haastattelututkimuksia, opiskelijoiden osaamisen mittaamista tentein ja kokein ennen ja jälkeen opetusoh-

jelman käyttämistä sekä opiskelijoiden osaamisen kehittymisen seuraamista opetusohjelman käytön aikana. Tässä tutkimuksessa esitetään menetelmä, jolla opiskelijoiden toiminnasta TRAKLA2-opetusohjelman käyttöliittymässä kerätään opiskelijoiden osaamista ja oppimisprosessia kuvaavaa dataa.

Opetusohjelmat ovat tällä hetkellä yksi nopeimmin kasvavista ohjelmistotekniikan alueista. Tämänkin vuoksi on erityisen tärkeää suorittaa pitkäjänteistä evaluointitutkimusta opetusohjelmien vaikutuksesta oppimiseen. Tässä työssä on pyritty kehittämään menetelmiä ja suuntaviivoja näiden evaluointitutkimusten tekemiseen tähän erityisen hyvin sopivassa ympäristössä, jossa opetusohjelmia ja niiden prototyyppinä voidaan testata hyvällä opiskelija-aineksella ja suurilla käyttäjämäärillä TKK:n Tietotekniikan osastolla.

Oppijan oppimisen tutkiminen opetusohjelmassa on tärkeää myös adaptiivisten opetusohjelmien kehitystyön kannalta. Adaptiivinen opetusohjelma mukautuu opiskelijan osaamistason ja oppimistyylin mukaiseksi. Tapa ja järjestys, jolla opetusohjelma esittää opetusohjelman eri sisällöt, kuten teoriaosat, animaatiot, simulaatiot tai harjoitustehtävät, voidaan adaptiivisessa opetusohjelmassa räätälöidä henkilökohtaisesti jokaiselle oppijalle sopiviksi. Samoin esimerkiksi harjoitustehtävien vaikeustaso tai rakenne voidaan valita oppijakohtaisesti. Adaptiivisten opetusohjelmien tutkimus on tällä hetkellä eräs opetusteknologian tutkimuksen kuuma peruna, siinä tapahtuu paljon ja myös yritysmaailma on kiinnostunut adaptiivisuudesta opetusohjelmissa.

Opetusohjelman käytöstä kerättyä datan analyysiä voidaan myös käyttää opetuksen kehittämisen. Esimerkiksi voidaan tutkia miten luentojen painotuksen muuttaminen vaikuttaa opiskelijoiden osaamiseen eri harjoitustehtävissä. Tämä edellyttää datan keräämistä useiden vuosien ajalta. Opetusohjelma ja kurssin sisältö kytkeytyvät joka tapauksessa toisiinsa hyvin tiiviisti, ja ne on pyrittävä rakentamaan toisiaan mahdollisimman hyvin tukeviksi.

Tutkielmassa luodaan ensin (luku 2) katsaus tärkeimpiin ohjelmoinnin opetusohjelmissä käytettyihin opetusteknologioihin sekä ohjelmoinnin opetusohjelmien evaluointimenetelmiin. Sovelluksia ja tutkimustuloksia opetusohjelmista ja automaattisesta arvioinnista löytyy runsaasti juuri ohjelmoinnin perusteiden opetuksesta, koska niissä on pystytty soveltamaan valmiita ohjelmien oikeellisuuden, tehokkuuden, ohjelmointityylin ja muiden ominaisuuksien analysoinnin työkaluja ja menetelmiä ohjelmointitehtävävastausten analysointiin. Opettajien kiinnostus ja taidot kehittää itse omaa työtään helpottavia työkaluja ovat myös vaikuttaneet siihen, että ohjelmoinnin opetusohjelmien kehitys- ja tutkimustyötä on tehty jo 1970-luvulta lähtien. TKK:n Tietojenkäsittelyopin laboratoriossa tehty tietorakenteiden ja algoritmien opetusohjelmien kehitys- ja tutkimustyö muodostaa tämän tutkielman perustan. Luvussa 3 luodaan katsaus tähän yli 10 vuotta kestäneeseen kehitys- ja tutkimustyöhön. Luvussa 4 määritellään tutkimusongelma edellisissä luvuissa esitettyjen opetusohjelmien, niiden tutkimusmenetelmien sekä opetuksen kehittämisen valossa. Tutkimusongelma luo vaatimukset seuraavassa luvussa (luku 5) esiteltävälle datan keruu- ja analysointimenetelmälle TRAKLA2-opetusohjelmasta. Luvussa 6 sovelletaan kehitettyä datan keruu- ja analysointimenetelmää esittämällä analyysi TKK:n kevään 2003 Tie-

torakenteet ja algoritmit -kurssin aikana opetusohjelman käytöstä kerätystä datasta. Datan keruu- ja analysointimenetelmän toimivuutta tarkastellaan datan analyysin pohjalta. Tutkitaan, voidaanko menetelmällä tehdä pätevää tilastollista analyysiä oppimisesta. Menetelmän soveltuvuutta opetusohjelmien evaluointitutkimukseen tarkastellaan esittämällä ideoita menetelmän mahdollistamista tutkimusasetelmista ja vertaamalla niitä kirjallisuudessa esitettyjen opetusohjelmien evaluointitutkimusten tutkimusasetelmiin. Menetelmän teknistä toteutusta arvioidaan sen opetusohjelman käyttöön aiheuttamien aikaviiveiden analyysillä sekä ohjelmiston muunneltavuutta analysoimalla. Luvussa 7 esitetään yhteenveto tutkimuksesta.

Luku 2

Ohjelmoinnin opetusohjelmista

Tämä tutkimus pohjautuu TRAKLA2-opetusohjelmaan (kappale 3.5), jossa sekä algoritmien visualisointi, automaattinen tarkastus että automaattinen palaute näyttelevät tärkeää osaa. Tämän vuoksi tässä luvussa tehdään kirjallisuuskatsaus näihin opetusteknologioihin, niihin liittyviin käsitteisiin sekä sovelluksiin.

Opetusohjelmien tekninen toteutus, ohjelmistoarkkitehtuuri ja käytetyt algoritmit ovat tietojenkäsittelyopin näkökulmasta mielenkiintoisia. Tämän vuoksi tässä luvussa luodaan katsaus myös automaattisen tarkastuksen menetelmiin ja algoritmeihin.

Tutkimuksessa kehitetään menetelmä opetusohjelman käytön tilastolliseen tutkimukseen, jolloin tässä kirjallisuuskatsauksessa on perusteltua selvittää alan kirjallisuudessa esitettyjä opetusohjelmien käytöstä tehdystä tilastollisen tutkimuksen menetelmiä.

2.1 Yleistä

Opetusohjelma on rajattuun aihepiiriin keskittyvä tietokoneohjelma, joka pyrkii autamaan käyttäjää asian omaksumisessa. Yleisiä opetusohjelmia ovat harjaannuttamisohjelmat eli drillit, joilla harjoitellaan mekaanisia taitoja. Harjaannuttamisohjelman aihepiiri voi olla vieraan kielen sanasto, yksinkertaiset lasku- tai ohjelmointitehtävät. Opetusohjelma voi myös olla rakennettu pelin tai simulaation muotoon. Perinteisen kynällä ja paperilla suoritettavan kuulustelun korvaava kuulusteluohjelma voi myös olla opetusohjelma. Pehdyttämisharjoitukset ovat systemaattisia johdatuksia opetettavaan asiaan ja asiasisällön lisäksi ne voivat sisältää tehtäviä. [33]

Tässä työssä käsitellään opetusohjelmia, joiden sovellusalue on ohjelmoinnin perusteiden ja siihen liittyvien käsitteiden opetus. Ohjelmoinnin perusteita opetetaan esimerkiksi yliopistojen ja korkeakoulujen ohjelmoinnin peruskursseilla, joissa käsitellään vastaavat asiat kuin ACM:n ja IEEE:n tietojenkäsittelyopin opetussuunnitelmassa [1]. Tarkastelu rajoittuu harjoitustehtäväohjelmiin, joissa opetusohjelma esittää opiskelijalle kysymyksiä tai harjoitustehtäviä ja opiskelija vastaa niihin opetus-

ohjelmaa käyttäen. Toteutukseltaan mielenkiintoisimpia ja monimutkaisimpia ovat opetusohjelmat, jotka automaattisesti analysoivat opiskelijan vastauksen ja antavat siitä palautetta opiskelijalle. Automaattista tarkastusta käyttävissä ohjelmoinnin opetusohjelmissa on usein sekä harjaannuttamisohjelmien että perehdyttämishjelmien ominaisuuksia. Simulaatiota käytetään erityisesti tietorakenteiden ja algoritmien opetusohjelmissa.

Opetusteknologia viittaa sellaisiin menetelmiin tai laitteisiin, joissa sovelletaan tietotai viestintäteknikkaa tai muita teknisiä välineitä opetukseen tai opiskeluun [33]. Esimerkkejä opetusteknologian sovelluksista voivat olla opetettavan asian esittäminen dokumenttielokuvan keinoin tai radio-ohjelmana. Voidaan ajatella, että opetukseen käytettävä tietokoneohjelma on sellaisenaan opetusteknologian sovellus. Toisaalta kuitenkin tietokoneohjelmaan sisältyy useita opetukseen ja oppimiseen liittyviä teknisiä menetelmiä ja innovaatioita, jolloin voidaan ajatella yhdessä tietokoneohjelmassa sovellettavan useita eri opetusteknologioita. Esimerkiksi erilaisia tiedon visualisointitekniikoita voidaan pitää opetusteknologisina sovelluksia, jos ne liittyvät opittavien asioiden ymmärrettävyyden lisäämiseen.

2.2 Visualisointi ohjelmoinnin oppimisessa

Näkökyvyllä on havainnollistamisen suhteen suurin ilmaisuvoima, joten kognitiiviset työkalut mallintavat käyttäjän ajattelua usein visuaalisesti. Visualisointiprosessi tähtää asian syvällisempään ymmärtämiseen; siinä käytettävissä olevaan tietoa kiitetään tai vahvistetaan. Esimerkiksi erilaisten kaavioiden piirtäminen voi havainnollistaa numeromuodossa olevaa tietoa.

Card et al [9] mukaan visualisointiprosessi edistää ajattelua kuudella eri tavalla:

1. Visualisointi antaa uusia resursseja käyttäjälle, koska työmuisti ei rajoitu aivojen sisälle laajemman kokonaisuuden ollessa näkyvillä tietokoneen näytöllä.
2. Tieto voidaan esittää tiiviissä ja mielekkäässä muodossa, jolloin tutkittava etsintäavaruus pienenee ja tarvittavan tiedon löytäminen nopeutuu.
3. Tieto hahmottuu paremmin käyttäjälle.
4. Johtopäätöksen teko kuvista on usein suoraviivaista.
5. Monen asian yhtäaikainen seuranta on helppoa visuaalisten vihjeiden perusteella.
6. Visualisointeja on helppo muokata, mikä auttaa tutkimaan esimerkiksi parametrien vaikutusta annettuun muuttujaan.

Tietokoneohjelmiin ja algoritmeihin liittyvät käsitteet ovat luonteeltaan abstrakteja, ja voidaankin olettaa erilaisten visualisointitekniikoiden auttavan näiden käsitteiden oppimisessa ja ymmärtämisessä.

Käsitteellisesti *ohjelmistojen visualisointi* (software visualization) kattaa kaiken ohjelmiston ymmärtämistä edistävän visualisoinnin. Ohjelmistoja visualisoidaan erilaisten kaavioiden, animaatioiden sekä interaktiivisten sovellusten avulla, ja jopa ohjelmakoodin sisennystä voidaan pitää visualisointina. Ohjelmistojen visualisointi on usein jaettu kahteen alakäsitteeseen: *ohjelman visualisointiin* (program visualization) ja *algoritmien visualisointiin* (algorithm visualization). [40]

2.2.1 Ohjelman visualisointi

Ohjelman visualisoinnissa visualisoidaan ohjelman koodia sekä ohjelman muokkaa-
maa dataa. Visualisointi voi olla staattista tai dynaamista. Esimerkiksi staattista
koodin visualisointia on ohjelman kutsuketjun visualisointi verkoksi ja staattista da-
tan visualisointia on linkitetyn listan visualisointi esittämällä lista-alkio laatikkona ja
osoittimet nuolina. Animaatio, jossa koodirivit väritetään ohjelman suorittaessa ky-
seisen rivin, on dynaamista koodin visualisointia, kun taas animaatio edellä kuvatun
linkitetyn listan dynaamisista muutoksista ohjelman muokatessa sitä on esimerkki
dynaamisesta datan visualisoinnista. [40]

Dynaamista ohjelman koodin ja datan visualisointia käytetään Jeliot-perheen ope-
tusohjelmissa [6]. Esimerkkiohjelmaa ajettaessa Jeliot näyttää aktiivisen koodirivin,
visualisoi ohjelman käsittelemät muuttujat sekä ohjelman tulosteen. Opiskelija voi
kontrolloida esimerkkiohjelman ajoa; liikkua ohjelmassa eteenpäin askel askeleelta,
ajaa ohjelman animaationa sekä aloittaa ohjelman suorituksen alusta.

2.2.2 Algoritmien visualisointi

Algoritmien visualisoinnissa (algorithm visualization) visualisoidaan ohjelmiston
korkeamman tason käsitteitä. Esimerkiksi algoritmin toimintaa voidaan havainnol-
listaa vuokaavion avulla. Rajanveto sille, milloin kyseessä onkin algoritmien visua-
lisointi ohjelman visualisoinnin sijasta, on joskus vaikea vetää. Jos visualisointi kes-
kittyy tietyn algoritmin jollain ohjelmointikielellä tehdyn toteutuksen visualisointiin,
niin kyseessä on enemmänkin ohjelman visualisointi. Jos taas visualisoinnissa keski-
tytään algoritmin toiminnan käsitteellisen toiminnan havainnollistamiseen, voidaan
visualisaatiota pitää algoritmien visualisaationa. Dynaamista algoritmivisualisaatio-
ta kutsutaan *algoritmianimaatioksi* (algorithm animation) [40].

Tämän työn taustalla oleva TRAKLA2-opetusohjelma käyttää algoritmien visua-
lisointiin Matrix-algoritmisimulaatiokirjastoa, joka pohjautuu seuraaviin algoritmia-
nimaation ja algoritmisimulaation määritelmiin. Algoritmin suorituksen edetessä al-
goritmin taustalla olevien tietorakenteiden tilat muuttuvat. Tietorakenteista voidaan
näyttää visualisaatio jokaisen tilamuutoksen jälkeen. Algoritmianimaatio on näiden
tilamuutosten sarja. Animaatiossa voidaan liikkua eteen- ja taaksepäin. Animaatios-
sa voidaan myös säätää sitä, kuinka monta algoritmin operaatiota yksi tilasiirtymä
sisältää. Käyttäjä voi myös manipuloida algoritmin tietorakenteiden visualisaatioita
graafisen käyttöliittymän operaatioilla. Esimerkiksi hän voi manipuloida tietoraken-

teita kuten tietty algoritmi niitä muuttaisi. Tietorakenteiden manipulointia graafisen käyttöliittymän operaatioilla kutsutaan *algoritmisimulaatioksi*. TRAKLA2-opetusohjelma kuvataan kappaleessa 3.5 ja Matrix-algoritmisimulaatiokirjasto kappaleessa 3.4.

PILOT [3, 8] on opetusohjelma verkkoalgoritmien opiskeluun, kuten verkon minimaalisen virittävän puun laskeminen käyttäen Kruskalin tai Primin algoritmia. PILOT esittää harjoitustehtävänannon, opiskelijan vastauksen sekä tarkistuksen tuottaman palautteen visuaalisesti. Opiskelijat ratkaisevat harjoitustehtäviä järjestelmän graafisella käyttöliittymällä. Esimerkiksi Primin algoritmia koskevassa harjoitustehtävässä on valittava hiirellä käyttöliittymään visualisoidun verkon kaaria siinä järjestyksessä, kun ne lisätään virittävään puuhun. PILOT generoi harjoitustehtäviin liittyvät verkot siten, että algoritmista harjoitustehtävästä tulee oppimisen kannalta tarkoituksenmukainen. Verkkojen ominaisuuksia, kuten kaarien suhdetta solmuihin, voi säätää. Tehtävän mallivastaus on algoritmianimaatio käsillä olevasta tehtävästä. Esimerkiksi Primin algoritmista kaaret, jotka muodostavat virittävän puun, väritetään oransseiksi, ja pienimmällä painoarvolla oleva kaari vihreäksi.

Tapa, jolla PILOT:ssa matkitaan algoritmin toimintaa olemalla vuorovaikutuksessa tietorakenteiden graafisen esityksen kanssa, muistuttaa paljon Matrix-visualisaatiokirjaston algoritmisimulaatio-operaatioita. Algoritmianimaation, algoritmisimulaation kaltaisten operaatioiden ja automaattisen tarkastuksen ja palautteen vuoksi käsittelemistäni järjestelmistä PILOT muistuttaa eniten oman työni pohjana olevaa TRAKLA2-järjestelmää.

Algoritmien visualisointi- ja animointitekniikoiden käytön yhteyttä oppimistuloksiin on tutkittu useaan otteeseen. Hundhausen et al. julkaisema tutkimus [13] kokoaa 24:n viime vuosina suoritettujen algoritmivisualisaatiotekniikoiden soveltamisesta ohjelmoinnin oppimiseen tehdyn empiirisen tutkimuksen tulokset. Tutkimuksessa todettiin algoritmivisualisaation käytön edistävän algoritmien oppimista. Tutkimuksessa havaittiin asiayhteyden, jossa algoritmivisualisaatiota käytettiin, olevan oppimistulosten kannalta oleellisen. Tehokkainta algoritmivisualisaatio oli, kun se oli yhdistetty aktiiviseen vaiheeseen oppimisprosessissa, kuten esimerkiksi harjoitustehtävän tekemiseen.

2.2.3 Visuaalinen ohjelmointi

Eräs ohjelmistojen visualisointiin läheisesti liittyvä käsite on *visuaalinen ohjelmointi* (visual programming). Käsitteiden suurin ero on siinä, että ohjelmistojen visualisoinnin tavoitteena on auttaa olemassa olevan ohjelmiston ymmärtämisessä, kun taas visuaalisen ohjelmoinnin tavoitteena on auttaa ohjelmistojen määrittelyssä. Luonnollisesti visuaalisen ohjelmoinnin järjestelmät voivat käyttää ohjelmistojen visualisoinnin tekniikoita ohjelmiston kuvaamiseen, mutta se ei kuitenkaan ole niiden pääasiallinen käyttötarkoitus [40].

Esimerkki visuaalista ohjelmointia soveltavista opetusohjelmista on Reiser et al. ke-

hittämä Lisp-opetusohjelma GIL [41]. Opetusohjelmassa opiskelija ratkaisee Lisp-ohjelmointitehtäviä ohjelmoimalla visuaalisesti; tekstuaalista ohjelmakoodia ei kirjoiteta. Lisp-ohjelma voidaan ajatella suunnattuna verkkona, jonka solmuja ovat atomiset data-alkiot ja Lisp-operaatioita kuvaavat solmut. Jokaista Lisp-operaatiota vastaavaan solmuun tulee yksi tai useampi kaari ja solmusta lähtee tasan yksi kaari. Harjoitustehtävän alussa ohjelmaikkuna sisältää harjoitustehtävän lähtöarvoja vastaavat data-alkiot ikkunan alareunassa ja ohjelman tulostusta vastaavat data-alkiot yläreunassa. Opiskelijan tehtävänä on ohjelmoida vaadittu algoritmi täydentämällä verkko sopivilla Lisp-operaatioilla siten, että ohjelma generoi tulostuksen lähtöarvoista. Esimerkiksi, jos FIRST-operaatiota vastaavaan solmuun kytketään kaari solmusta, jonka ulostulona on lista $(a\ b\ c\ d)$, niin operaatio tuottaa ulostuloonsa atomin a . Lisp-operaatiot lisätään verkkoon valitsemalla operaatiota vastaava kuvake valintapaneelistä, vetämällä se ohjelmaikkunaan ja määrittelemällä operaation sisääntulot ja ulostulo. Sisääntulot ja ulostulo määritellään yksikäsitteisesti lisäämällä verkkoon data-alkiot, antamalla niille arvot ja kytkemällä data-alkiot operaatioalkioihin kaarilla. Joka toinen verkon solmu on siis operaatio-alkio ja joka toinen data-alkio. Ohjelma on valmis, kun verkko yhdistää ohjelmaikkunan alareunan lähtöarvot yläreunan tulostukseen.

2.3 Automaattinen palaute

Palaute näyttää merkittävää osaa oppimisprosessissa. Hyvin määritelty palaute opitun asian soveltamisesta kertoo oppijalle mahdollisista väärinkäsityksistä tai puutteista tiedoissa ja ohjaa hankkimaan lisää tietoja tai harjoittelemaan opitun asian soveltamista uudelleen [33].

Automaattisesta palautteesta (automatic feedback) on kyse silloin, kun opiskelijan harjoitustehtävävastaus tarkastetaan automaattisesti ja palaute opiskelijalle annetaan automaattisesti joko opetusohjelman kautta tai sähköpostitse. Automaattisesta tarkastuksesta lisää luvussa 2.4.

2.3.1 Palautteen välittömyys

Välittömästi palautteesta on kyse silloin, kun automaattisen tarkastuksen tuottama palaute annetaan opiskelijalle välittömästi harjoitustehtävän palautuksen jälkeen tai joissain tapauksissa jo harjoitustehtävää tehtäessä.

Corbett et al. [10] ovat tutkineet palautteen välittömyyden merkitystä oppimiseen Lisp-opetusohjelmassa, jonka harjoitustehtävät olivat pieniä ohjelmointitehtäviä. Opetusohjelman toteutus mahdollisti opiskelijan ratkaisun oikeellisuuden tarkastuksen jokaisen symbolin kirjoittamisen jälkeen ja välittömän palautteen antamisen mahdollisista virheistä. Heidän tutkimustuloksensa mukaan välitön palaute oli oppimisen kannalta tehokkainta. Välitöntä palautetta saaneet opiskelijat saivat harjoitustehtävät ratkaistua nopeimmin ja heidän tenttituloksensa olivat yhtä hy-

vät kuin samaa opetusohjelmaa käyttäneiden, mutta palautetta vasta koko tehtävän tekemisen jälkeen saaneilla opiskelijoilla.

Toisaalta voidaan ajatella, että välittömän palautteen käyttäminen estää korkeamman abstraktiotason mentaalisten mallien syntymisen. Ben-Ari [5] esittää ajatuksen, että opiskelijoille tulisi nimenomaan ensin opettaa korkeamman tason mielikuvamalli tietokoneesta, koska ohjelmointia on myös mahdollista oppia yrityksen ja erehdyksen menetelmällä ja tällöin pätevä mielikuvamalli tietokoneen ja tietokoneohjelman toiminnasta saattaa muodostua vääräksi.

2.3.2 Palautteen esitystapa

Palaute voi tekstuaalista tai visuaalista. PILOT [3] antaa opiskelijalle välitöntä palautetta verkkoalgoritmiharjoitustehtävän tekemisen aikana. Jos valitaan väärä kaari, niin PILOT huomauttaa virheestä värjäämällä kaaren punaiseksi ja antaa opiskelijalle tästä myös tekstuaalista palautetta teksti-ikkunaan. Toinen esimerkki PILOT:n antamasta välittömästä visuaalisesta palautteesta on silmukoiden korostaminen. Jos opiskelijan ratkaisu luo silmukan, niin silmukan kaaret väritetään sinisiksi.

PILOT antaa myös tekstimuotoista palautetta. Kun opiskelija tekee tehtävää, niin järjestelmä tulostaa teksti-ikkunaan ilmoituksen virheestä ja virheen kuvauksen. Oteetaan esimerkiksi Primin algoritmi. Siinä yksi solmu valitaan puun juureksi ja puuta kasvatetaan valitsemalla painoltaan pienin kaari, joka johtaa puuhun kuulumattoon solmuun. Jos opiskelija valitsee väärän kaaren, PILOT voi antaa seuraavan tyyllisen viestin: "Sinun on löydettävä kaari, jolla on pienin paino".

Useimmat ohjelmakoodia tarkastavat opetusohjelmat esittävät analyysin opiskelijan vastauksesta tekstimuotoisena. Näistä enemmän luvussa 2.4.

2.3.3 Palautteen yksityiskohtaisuus

Palautteen yksityiskohtaisuus voi vaihdella. Korkean tason palautteessa voidaan vain kertoa opiskelijalle, onko hänen ratkaisunsa oikein vai väärin ja tarjota vihjeitä ratkaisun korjaamiseksi. Yksityiskohtaisemmassa palautteessa voidaan kertoa tarkasti, missä mahdollinen virhe esiintyi sekä selvittää virheen ja väärinymmärryksen laatua yksityiskohtaisesti.

Mitrovic et al [37] suorittamassa tutkimuksessa SQL-Tutor -opetusohjelmasta arvioitiin opetusohjelman antaman automaattisen palautteen yksityiskohtaisuuden yhteyttä oppimistulokseen. Opiskelijoiden käyttöön annettiin kaksi versiota opetusohjelmasta, joista ensimmäinen antoi palautteena vain vihjeitä opiskelijoiden tekemistä virheistä. Toinen versio tarjosi palautteena yksityiskohtaisen kuvauksen opiskelijan tekemästä virheestä. Tutkimuksen tuloksena selvisi, että kyseisessä opetusohjelmassa korkean tason palaute tuki oppimista parhaiten.

2.4 Automaattisesta tarkastuksesta

Automaattisen palautteen antamisen edellytyksenä on harjoitustehtävävastausten *automaattinen tarkastus* (automatic assessment). Automaattinen tarkastus tarkoittaa elektronisessa muodossa palautetun harjoitustehtävävastauksen automaattista tarkastusta ja palautteen muodostamista tarkastusprosessin tuotoksena. Tässä kappaleessa esitellään kolmea automaattisen tarkastuksen sovellusalueita ohjelmointin opetuksessa.

2.4.1 Ohjelmakoodin automaattinen tarkastus

Ensimmäisiä ohjelmakoodia tarkastavia järjestelmiä oli ASSYST [15, 16]. Ohjelma oli tarkoitettu opettajien työkaluksi C- ja Ada-kielen ohjelmointiharjoitustehtävien tarkastamisessa. ASSYST analysoi opiskelijan ohjelman oikeellisuuden, tehokkuuden, kompleksisuuden sekä ohjelmointityylin.

Ohjelman oikeellisuuden tarkastus aloitetaan tarkastamalla ohjelman kääntyminen. Tämän jälkeen ohjelmaa ajetaan testidataa vastaan, jolloin opiskelijan ohjelman tulostusta verrataan malliratkaisuohjelman tulostukseen samalla testidatalla. Ohjelman tehokkuutta arvioidaan vertailemalla opiskelijan ratkaisun käyttämää CPU-aikaa malliratkaisuohjelman käyttämään CPU-aikaan. Tämän lisäksi ASSYST arvioi ohjelman tehokkuutta staattisella ohjelman rakenteen analyysillä. ASSYST osaa arvioida ohjelman kompleksisuutta ohjelman välikoodin kontrollivuoverkkoa analysoimalla. ASSYST antaa myös tyylipisteitä, joiden kriteereinä ovat lohkojen (block) pituus koodiriveinä, sisennyksen käytön oikeellisuus, jne. Opiskelijoiden oletetaan testaavan ohjelmiaan itse. Ohjelman palautuksen mukana heidän on palautettava myös käyttämänsä testidata. ASSYST osaa arvioida opiskelijoiden testidatan kattavuuden; kuinka perusteellisesti ohjelman toimintoja testiajot ovat testanneet.

Palaute automaattisen analyysistä täydennettynä opettajan kommentilla lähetetään sähköpostitse opiskelijoille. Järjestelmän analyysin tarkkuutta pidettiin parempana kuin manuaalisessa tarkastuksessa. Etuna manuaalisesti tehtävään tarkastukseen pidettiin myös yhdenmukaisuutta arvostelussa.

Laajimmin käytössä olevia ohjelmakoodia tarkastavia järjestelmiä on Ceilidh [7]. Opetusohjelman ensimmäinen on otettu tuotantokäyttöön vuonna 1988. Nykyään Ceilidh on käytössä yli 300 oppilaitoksessa 30 eri maassa. Alunperin Ceilidh kehitettiin C-kielisten ohjelmien tarkastamiseen, mutta järjestelmää on käytetty myös C++, Pascal-, SML-, SQL-, Java- ja Prolog [30] -kielen ohjelmointiharjoituksissa. Ohjelmointiharjoitustöiden lisäksi Ceilidhiä on käytetty matemaattisella Z-notaatiolla tehtyjen ohjelmistomäärittelyjen automaattiseen tarkastamiseen [11]. Z on formaali, puhtaasti matemaattiseen notaatioon perustuva spesifikaatiokieli ohjelmistojen määrittelyyn.

Ceilidhissä tehtävien automaattiseen tarkastukseen voidaan laatia ohjelmien toimintaa testaavia dynaamisia testejä ja ohjelman rakennetta analysoivia staattisia testejä.

Ohjelman oikeellisuutta voidaan testata ajamalla ohjelmaa testidataa vastaan ja vertaamalla tulostusta malliratkaisuohjelman tulostukseen. Ceilidh mahdollistaa myös ohjelman tehokkuuden testaamisen dynaamisella analyysillä, jossa lasketaan suoritettujen koodirivien lukumäärä ja verrataan tätä malliratkaisuohjelman suoritukseen. Ceilidh mahdollistaa staattisen tyylianalyysin, jossa ohjelmointityyliä voidaan arvioida koodirivien pituuden, muuttujien nimien, funktioiden pituuden, kommenttien käytön ja sisennyksen käytön perusteella. Ohjelman kompleksisuutta arvioidaan varattujen sanojen, goto-lauseiden, silmukoiden, ehtolauseiden, operaattoreiden, ja aliohjelmakutsujen lukumääriä analysoimalla. Kriteereinä voidaan myös käyttää silmukoiden tai sulkuhierarkioiden syvyyttä. Puutteita ohjelman rakenteessa arvioidaan Unixin Lint-ohjelman avulla. Analyysissä huomataan esim. turhat muuttujan määrittelyt ja kuolleet kohdat koodissa.

TKK:n Tietojenkäsittelyopin laboratoriossa on kehitetty automaattinen kotitehtäväjärjestelmä Scheme-kielen harjoitustehtävien generointiin ja tarkastamiseen. Järjestelmästä enemmän luvussa 3.2.

2.4.2 Algoritmisten harjoitustehtävien tarkastus

Algoritmiset harjoitustehtävät voidaan rakentaa siten, että tehtävän vastaus on esitys algoritmin tietorakenteiden tiloista algoritmin suorituksen ajalta. TRAKLA-järjestelmässä (esitellään kappaleessa 3.1) tietorakenteiden tilat koodataan tekstimuotoon. Vastaus voidaan myös konstruoida manipuloimalla tietorakenteiden visuaalisia esityksiä ohjelman graafisessa käyttöliittymässä, kuten TRAKLA2-opetusohjelmassa (esitellään kappaleessa 3.5) tai PILOT-opetusohjelmassa (esitelty kappaleessa 2.2.2).

Edellä kuvatuissa opetusohjelmissa tehtävien tarkastus suoritetaan vertailemalla opiskelijan esittämiä tietorakenteiden tiloja mallivastausalgoritmin tuottamiin.

2.4.3 Tietämyskantaan perustuva tarkastus

SQL-kielen opetusohjelmassa SQL-Tutorissa [34] kohdealueen tietämys on mallinnettu *rajoiteperusteista mallinnusta* (constraint-based modeling) käyttäen. Esimerkiksi tietämyskantaan voidaan määritellä rajoite, että SELECT -lauseen FROM -lause ei saa olla tyhjä lause. Esimerkki monimutkaisemmasta rajoitteesta on vaatimus, että jos WHERE -lauseessa käytetään predikaattia BETWEEN, niin sen jälkeen täytyy olla kaksi vakiota erotettuna avainsanalla AND.

SQL-tutorin antama palaute perustuu tietoon siitä, mitä rajoitetta opiskelijan vastaus rikkoo. Yksinkertaisin palaute kertoo opiskelijalle, onko hänen vastauksensa oikein vain väärin. Jos opiskelija toistaa saman virheen useaan kertaan, niin järjestelmä antaa yksityiskohtaisempaa palautetta, jossa kerrotaan missä virhe esiintyi ja selvitetään virheen ja väärinymmärryksen laatua yksityiskohtaisemmin.

2.5 Adaptiivisuus

Adaptiiviset opetusohjelmat tai ITS-järjestelmät (intelligent tutoring systems) ovat oppimisympäristöjä, jotka mukautuvat oppilaan mukana oppimisprosessin eri vaiheissa. Aluksi järjestelmä mukautuu opiskelijan lähtötason mukaiseksi ja oppimisprosessin edistyessä järjestelmä mukautuu opiskelijan taitojen kasvaessa. ITS-Järjestelmä oppii opiskelijan toiminnasta ja muodostaa opiskelijamallin tämän perusteella. Tärkeä osa ITS-järjestelmää on siis sen havainnoiminen, miten opiskelija järjestelmässä toimii.

ELM-ART [46] on WWW-pohjainen, adaptiivinen opetusohjelma Lisp-ohjelmointikielen opiskeluun. Järjestelmä koostuu hypertekstimuodossa olevasta oppimateriaalista, yksinkertaisista monivalintaharjoitustehtävistä ja pienistä ohjelmointiharjoitustehtävistä. Järjestelmä luo opiskelijamallin opiskelijoiden harjoitustehtävävastausten perusteella.

Hypertekstin sisällysluettelossa ja tekstin joukossa olevat linkit on merkitty värikoodilla opiskelijan osaamisen perusteella. Vihreä linkki kertoo opiskelijalle, että hän hallitsee linkin osoittaman sivun esitiedot ja että on suositeltavaa vierailla sivulla. Punainen linkki kertoo, että sivun esitiedot eivät ole kunnossa. Keltainen linkki on merkki jo vierailusta sivusta. Oranssi linkki on merkinä sivusta, jolla on jo vierailtu, mutta jonka alapuolella olevilla sivuilla ei ole käyty. Opiskelija voi myös pyytää järjestelmää valitsemaan hänen tietotasoonsa parhaiten sopivan sivun painamalla Next-painiketta. Teorian ja harjoitustehtävien lisäksi opetusohjelmassa käytetään kooditason esimerkkejä, jotka valitaan kullekin opiskelijalle sopiviksi.

2.6 Ohjelmoinnin opetusohjelmien tutkimusmenetelmistä

Ohjelmoinnin opetusohjelmien käyttöä on tutkittu useasta näkökulmasta. Opetusohjelmien tutkimuksen motivaatio on yleensä aina ohjelman käyttämisen ja parantuneen oppimistuloksen välisen yhteyden löytäminen. Usein myös opiskelijoiden mielihiteille järjestelmästä annetaan paljon painoarvoa.

Opetusohjelmista julkaistuissa tutkimuspapereissa on yleensä kuvattu kyselytutkimus, jolla on selvitetty opiskelijoiden mielipiteitä ohjelmasta. Yleensä opiskelijoiden yleinen mielipide järjestelmästä on ollut myönteinen, kuten esimerkiksi Ceidhin käytöstä tehdyssä kyselytutkimuksessa [12]. Joissain tutkimuksissa kyselytutkimuksen perusteella on saatu tarkempiakin tuloksia. Esimerkiksi JDSL Visualizerin käytöstä 120 opiskelijalle tehdyssä kyselyssä [4] selvisi, että vain 4% opiskelijoista ei pitänyt järjestelmää hyvänä apuvälineenä oman ohjelmakoodin debuggaukseen.

Yleinen tapa mitata oppimistulosta on järjestää paperilla ja kynällä tehtävät tentit ennen ja jälkeen opetusohjelman käyttöä. Opetusohjelmaa käyttäneiden opiskelijoiden oppimistulosta olisi hyvä verrata sellaisten opiskelijoiden oppimistulokseen,

jotka eivät käyttäneet järjestelmää. Näin on selvietty oppimistuloksia esimerkiksi SQL-Tutor-opetusohjelmasta tehdyssä tutkimuksessa [37]. Huhtikuussa 1998, touku-
kuussa 1999 ja lokakuussa 1999 järjestetyillä tietokantakurssilla järjestelmää käytti
20, 33 ja 48 opiskelijaa. Kyselytutkimuksissa suurin osa opiskelijoista piti järjestel-
mää hyvänä ja oppimista edistävänä. Tutkimuksessa, jossa järjestelmää käyttänei-
den opiskelijoiden oppimistuloksia verrattiin samalla kurssilla opiskelleen kontrolli-
ryhmän oppimistuloksiin, saatiin tulokseksi, että SQL-Tutor:ia käyttäneiden opiske-
lijoiden arvosanat kurssilla olivat huomattavasti parempia.

Opetusohjelman käytön ja oppimistulosten yhteyttä voidaan myös tutkia ope-
tusohjelman käytöstä kerättyjä lokitietoja ja tehtävävastauksia analysoimalla.
SQL-Tutor:in lokitietoja tutkimalla huomattiin, että tietyn rajoitteen hallitseminen
oli kyseisen rajoitteen harjoittelun funktio. Esimerkiksi SQL SELECT -lauseen
BETWEEN -predikaatin osaaminen oli suoraan verrannollinen siihen, kuinka paljon
kyseistä predikaattia käyttäviä harjoitustehtäviä opiskelija oli tehnyt. [37]

Toinen mielenkiintoinen tilastollinen tutkimustulos koski palautteen yksityiskohtai-
suuden merkitystä oppimistuloksiin. Järjestelmästä tehtiin kaksi versiota, joita mo-
lempia käyttivät eri opiskelijaryhmät. Ensimmäinen järjestelmä antoi rajoitteisiin pe-
rustuvaa palautetta, jossa annettiin ainoastaan tieto mahdollisesta virheestä ja sen
sijainnista. Virhe liittyy siis aina jonkin tietyn rajoitteen rikkomiseen. Toinen järjes-
telmä tarjosi täydellisempää palautetta esittäen opiskelijalle edellisen palautteen li-
säksi koko malliratkaisun. Hypoteesina oli, että rajoitteisiin perustuva palaute tukisi
oppimista paremmin. Tutkimustulokset tukivat tätä hypoteesia; ensimmäistä järjes-
telmää käyttäneet opiskelijat saivat tehtävät ratkaistua huomattavasti nopeammin
ja vähemmällä yrityskerroilla kuin toista järjestelmää käyttäneet opiskelijat. Voidaan
olettaa, että valmiin malliratkaisun esittäminen opiskelijalle ei käynnistä niin syvä-
listä ajatus- ja oppimisprosessia kuin vähemmän informaatiota tarjoavan rajoitteisiin
perustuvan palautteen esittäminen. [37]

Myös PILOT-opetusohjelman käytöstä on tehty tilastollista tutkimusta. Tutkimus-
hypoteesina oli, että PILOT:n antaman arvosanat korreloivat perinteisten lasku-
harjoitusarvosanojen kanssa. Toinen hypoteesi oli, että PILOT:ia kauemmin käyttävien
opiskelijoiden laskuharjoitusarvosana paranee PILOT:n käyttämisen ansiosta. Tut-
kimustulokset eivät vahvistaneet hypoteeseja, koska 94% opiskelijoista sai täydet
pisteet perinteisistä laskuharjoitustehtävistä ja 93% opiskelijoista sai täydet pisteet
PILOT-kotitehtävistä. Mitään yhteyttä molemmissa kotitehtävissä huonosti menes-
tyneiden opiskelijoiden välillä ei löydetty. Tilastollisesti merkittävien tulosten saa-
miseksi sekä PILOT-kotitehtävistä että perinteisistä kotitehtävistä täytyy laatia vai-
keampia. [3]

Luku 3

Tietorakenteiden ja algoritmien opetusohjelmat TKO-laboratoriossa

Tietorakenteiden ja algoritmien opetusohjelmien kehittäminen ja niihin liittyvä tutkimustyö TKK:n Tietojenkäsittelyopin laboratoriossa alkoi jo 1990-luvun alussa. Vuoden 1990 keväällä Tietorakenteet ja algoritmit -kurssin iso harjoitustyö korvattiin useilla pienillä kotitehtävillä (n. 20 kpl). Kurssille osallistui vuosittain lähes 500 opiskelijaa, joten kotitehtävävastausten tarkastaminen käsin huomattiin pian hyvin työlääksi ja kurssin resurssit loppuivat kesken. Koska oppimistulokset pienten algoritmiharjoitustehtävien käytöstä olivat hyviä, niin mallia haluttiin jatkaa ja aloitettiin kehitysprojekti automaattisen kotitehtäväjärjestelmän laatimiseksi. Oppilastyönä kehitettiin TRAKLA-opetusohjelma [14] (Tietorakenteet ja algoritmit, kotilaskujen arvostelu) tietorakenteille ja algoritmeille.

Seuraavissa kappaleissa kuvataan TRAKLA-järjestelmän ja sen seuraajien toiminnot sekä järjestelmien käytöstä saatuja kokemuksia ja tutkimustuloksia.

3.1 TRAKLA

Vuonna 1991 käyttöön otetussa TRAKLA-opetusohjelmassa [14] kotitehtävänannot lähetettiin opiskelijoille sähköpostitse ja tehtävät myös palautettiin sähköpostitse. Järjestelmä tarkastaa vastaukset ja lähettää palautteen opiskelijoille sähköpostitse.

TRAKLA:n tehtävät on suunniteltu siten, että algoritmien toiminta kuvataan tekstinä. Esimerkiksi lajittelualgoritmia koskevan tehtävän vastaus saattaa sisältää lajiteltavan merkkijonon jokaisen operaation jälkeen. Menetelmälle algoritmin toiminnan kuvaamiseen esittämällä tietorakenteiden tiloja algoritmin suorituksen aikana määriteltiin uusi käsite – *algoritmisimulaatio*.

TRAKLA:n tehtävänannot ovat yksilöllisiä. Esimerkiksi lajittelualgoritmit tehtävässä lajiteltava merkkijono on erilainen jokaiselle opiskelijalle. TRAKLA:n kotitehtävät sisältävät mm. seuraavia algoritmeja koskevia tehtäviä:

- *Perustietorakenteet*: linkitetyn listan, pinon tai jonon manipulointi tai binääripuiden läpikäynti
- *Järjestämisalgoritmit*: kuorilajittelu (bubble sort), pikalajittelu (quick sort), digitaalinen vaihtojärjestäminen (radix exchange sort), suora digitaalinen järjestäminen (straight radix sort), binäärikekojen manipulointi ja lomitusjärjestely (merge sort)
- *Hakurakenteet*: puolitus- ja interpolaatiohaku, binäärisen hakupuun, 2-3-4 -puun, digitaalisen hakupuun tai radix-hakupuun rakentaminen sekä erilaisten hajautusrakenteiden toiminta
- *Merkkijonoalgoritmit*: hahmontunnistus tilakoneilla, pakkaus- ja salausalgoritmit
- *Verkkoalgoritmit*: seuraajalistaesitys verkosta, syvyyshaku, rintamahaku, union-find -rakenteet, verkon minimaalisen virittävän hakupuun laskeminen ja topologinen lajittelu

TRAKLA toimii Unix-ympäristössä. Sähköpostit vastaanotetaan normaalisti Unixin sendmailin avulla. Postien lajitteluun ja käsittelyyn käytetään Unixin työkaluohjelmia sekä C-kielillä ohjelmoituja apuohjelmia. Järjestelmä tarjoaa kehyksen automaattisesti tarkastettavien algoritmien tehtävien tai pienten ohjelmointitehtävien laatimiseen. Tehtävän laatimisen yhteydessä määritellään, miten alkuarvot generoidaan. Automaattisen tarkastuksen toiminnot voidaan määritellä erillisenä moduulina, joka voi olla C-ohjelma, tai käytännössä millä ohjelmointikielillä tahansa ohjelmoitu testeri.

3.2 Scheme-Robo

Scheme-Robo [42] on TRAKLA:an integroitu moduuli Scheme-kielisten ohjelmointitehtävien yksilölliseen generointiin ja automaattiseen tarkastamiseen. Tehtävien lähettämiseen opiskelijoille, vastausten palauttamiseen ja automaattisen tarkastuksen palautteen lähettämiseen opiskelijoille käytetään TRAKLA:a. TRAKLA:a käytetään myös kotitehtävapisteen kirjanpitoon sekä tilastojen laskentaan.

Scheme-ohjelmista tarkastetaan niiden oikeellisuus Scheme-funktioiden paluuarvoja tutkimalla. Opiskelijoiden ohjelmia ja malliratkaisuja ajetaan testidataa vastaan, joka voi olla satunnaisesti generoitua. Ohjelmointitehtävän koodissa voidaan vaatia olevan joku tietty avainsana, tai tietyn avainsanan käyttö voidaan kieltää. Scheme-Robo osaa myös tehdä yksinkertaisia ohjelman rakenteen analyysieja, kuten vastaako ohjelman rakenne tehtävänannossa annettua ohjelman runkoa.

Harjoitustehtävien ratkaisemista yrityksen ja erehdyksen menetelmällä on haluttu vähentää rajoittamalla tehtävän ratkaisuyrityskertojen määrää 20:een sekä asettamalla 10 minuutin viive automaattisen tarkastuksen palautteen lähettämiseen sähköpostitse.

Scheme-Robo osaa etsiä plagioituja vastauksia. Ohjelmointitehtävävastaukset muutetaan abstraktin syntaksipuun muotoon, jolloin muuttujien nimet poistetaan sekä määrittelyiden ja argumenttien järjestys yhtenäistetään. Samankaltaisuutta tutkitaan vertailemalla näitä syntaksipuita toisiinsa. Koska Scheme-Robo -harjoitustehtävät ovat lyhyitä, niin parhaat tulokset samankaltaisuuden tutkimisessa on saatu vertailemalla kaikkia harjoitustehtäviä ja etsimällä opiskelijoita, joiden vastaukset ovat samankaltaisia usean harjoitustehtävän osalta.

Scheme-Robo on ollut tuotantokäytössä ohjelmoinnin peruskurssilla jo usean vuoden ajan, ja järjestelmä on toiminut moitteettomasti. Vuosittain kurssille osallistuu yli 300 opiskelijaa, ja kurssilla on yli 50 kotitehtävää. Tarkastettavia kotitehtävävastauksia on vuosittain siis tuhansia. Syksyllä 2000 ohjelmoinnin peruskurssille osallistui noin 350 opiskelijaa. Kurssilla oli 5 harjoitustehtäväkierrosta. Jokaisella kierroksella oli 12 tehtävää, joista 4 oli pakollisia. Opiskelijoiden mielipiteitä Scheme-Robosta selvitettiin palautekyselyllä. 229 vastaajasta 80% koki automaattisen palautteen hyvänä tai erinomaisena ideana. Vain 10% vastaajista ei ollut tyytyväinen Scheme-Robon tekemään arvosteluun. Scheme-Robon antaman palautteen sisältöä ja yksityiskohtaisuutta voisi kuitenkin vielä parantaa, koska vain 45% vastaajista piti Scheme-Robon virheellisistä vastauksista antamia kommentteja riittävän hyvinä tai hyvinä. [42]

Täysin automaattinen kotitehtäväjärjestelmä mahdollistaa tehtävien tekemisen mihin vuorokaudenaikaan tahansa. Vuosien 1999 ja 2000 aikana Scheme-Robon palautuksista suurin osa on tehty kuitenkin päivällä. Tehtäväpalautuksien arvosanoja tutkimalla tehtiin mielenkiintoinen havainto, että myöhään yöllä 03:00–06:00 aikaan tehdyistä palautuksista hyväksytyn pistemäärän ylittäviä oli huomattavasti vähemmän kuin muina vuorokauden aikoina. Tästä on tehtävissä johtopäätös, että opiskelijoita olisi hyvä kannustaa työskentelemään päivisin esimerkiksi välttämällä tehtävien määräaikojen asettamista aamuiksi. [28]

3.3 TraklaEdit ja WWW-TRAKLA

Vastausten koodaaminen tekstiksi oli kömpelö tapa tehdä tietorakenteita ja algoritmeja koskevia tehtäviä, koska oppikirjoissa ja kurssin luennoilla tietorakenteet esitettiin visuaalisesti. Tämä antoi sysäyksen projektille tietorakennetehtävien graafisen editorin kehittämiseen. Projektin tuotoksena syntyi TraklaEdit-ohjelma, jolla tietorakennetehtäviä pystyi ratkomaan graafisen käyttöliittymän operaatioilla. Vuonna 1993 käyttöön otettu ohjelma oli toteutettu Macintosh -ympäristöön. TraklaEdit esitti TRAKLA-tehtävien tietorakenteet visuaalisessa muodossa. Ohjelman käyttöliittymä mahdollisti tehtävien tekemisen tietorakenteiden visuaalista esitystä manipuloimalla, esim. vetämällä ja pudottamalla avaimia tietorakenteen visuaalisen esityksen solmus-

ta toiseen. Ohjelma mahdollisti operaatioiden kelaamisen eteen- ja taaksepäin Undo- ja redo-operaatioilla. TraklaEdit generoi tehtävän vastauksen TRAKLA:n tekstimuotoon ja palautti tehtävän TRAKLA:an sähköpostitse. TraklaEdit oli täysin yhteensopiva sähköpostipohjaisen TRAKLA:n kanssa. [14]

TraklaEditin huono puoli oli riippuvaisuus Macintosh-laitteistoista, joita opiskelijoiden käytössä ei juurikaan ollut. Seuraava tutkimusprojekti olikin TraklaEditin muuttaminen Java-yhteensopivaksi. TraklaEditin Java-version valmistaikin Ari Korhosen diplomityöprojektina vuonna 1997. Java-ohjelmasta tehtiin WWW-selaimessa toimiva sovelma ja se integroitiin kurssin WWW-sivujen yhteyteen. TraklaEditin Java-version ja vanhan TRAKLA:n synnyttämä kokonaisuus nimettiin WWW-TRAKLA:ksi. [17]

Kokemukset WWW-TRAKLA:n käytöstä olivat myönteisiä. Kurssin opettajien kokemus oli, että opiskelijoiden osaaminen kurssin vaativassa suunnittelutehtävässä oli huomattavasti parempi, kun suunnittelutehtävää edelsi 25 pakollista WWW-TRAKLA-kotitehtävää. Myös opiskelijoiden palaute WWW-TRAKLA:n käytöstä oli hyvin myönteistä. [19]

WWW-TRAKLA:ssa nähtiin myös kehitystarpeita. Automaattinen palaute lähetettiin edelleen sähköpostitse, eikä sen tekstimuoto ollut kovin havainnollinen. Vaikka tehtävän vastaus konstruointiin algoritmivisualisaatiota käyttäen, niin malliratkaisun esitysmuoto oli vielä tekstipohjainen. Uusien harjoitustehtävien laatimista hankaloitti niiden ohjelmoinnin työläisyys. [19]

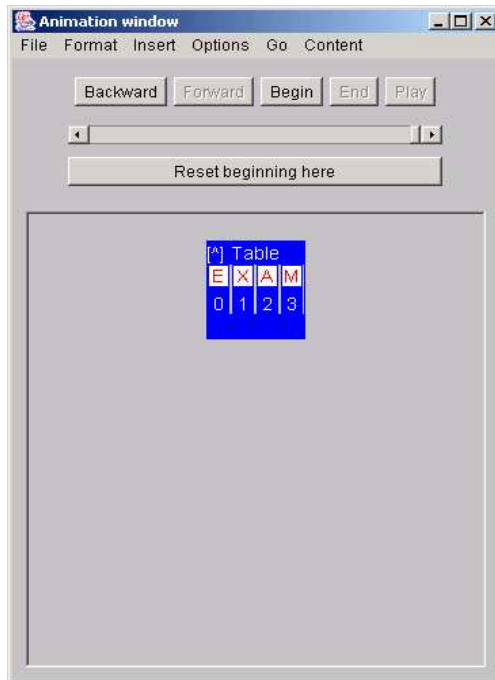
3.4 Matrix-algoritmisimulaatiojärjestelmä

TraklaEditin ja WWW-TRAKLA:n synnyttämää polkua jatkoi Ari Korhosen lisen-siaattityötutkimuksen puitteissa kehitetty Matrix-järjestelmä, joka on järjestelmä algoritmien ja tietorakenteiden visualisointiin, animointiin ja simulointiin. [18, 21]

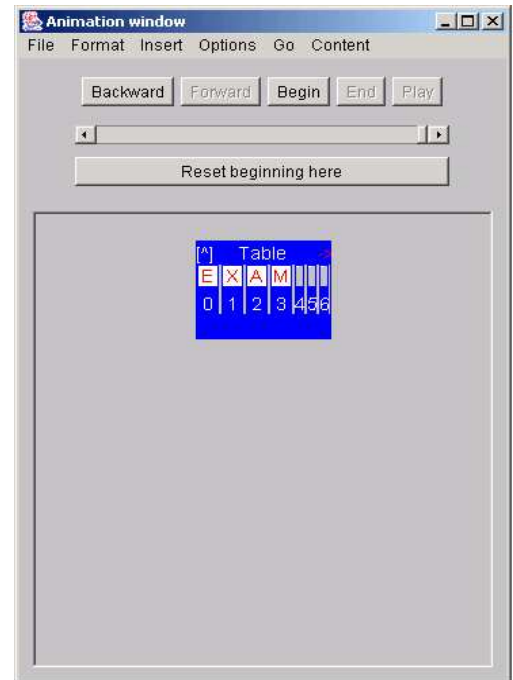
Matrix-järjestelmä perustuu ajatukseen, että algoritmien ja tietorakenteiden toimintaa havainnollistetaan visualisoimalla tietorakenteita. Algoritmin suoritus esitetään tietorakenteiden tilamuutosten kautta. Matrixin algoritmianimaatio on sarja tietorakenteiden visuaalisten esitysten muutoksia, joita voi kelata eteen- ja taaksepäin askel askeleelta. Kuvassa 3.1 on esimerkki Matrix-järjestelmän käyttöliittymästä. Käyttöliittymäikkunan yläreunassa on animaattori-paneeli, jonka toiminnoilla tietorakenteiden tilojen muutoksia voi selata. Matrixin käyttöliittymän komennoilla käyttäjä voi manipuloida tietorakenteiden visuaalisia esityksiä. Menu-komennoilla ja kontekstiriippuvaisilla veto ja pudotus (drag&drop) -operaatioilla voi muokata tietorakenteiden sisältöä ja esitystapaa.

Matrixin tukemat *perustietorakenteet*¹ ovat taulukko, linkitetty lista, binääripuu,

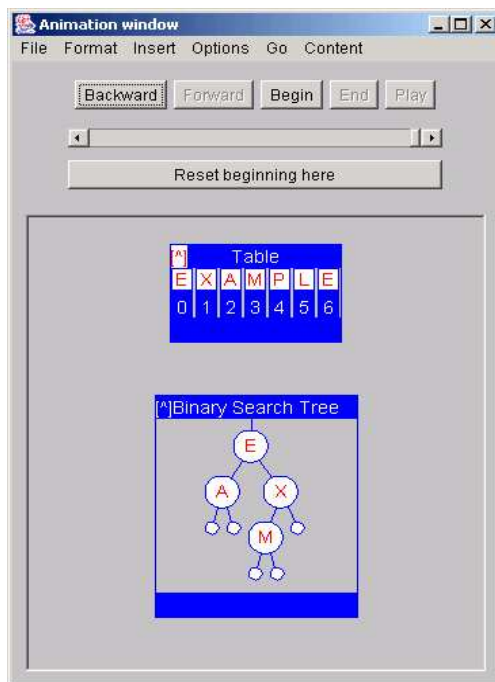
¹Matrix-järjestelmän yhteydessä on määritelty perustietorakenteen käsite (fundamental data type, FDT). Matrixin tukemat perustietorakenteet ovat taulukko, linkitetty lista, binääripuu, puu ja verkko. Kaikkien Matrixin tietorakennetoteutusten täytyy toteuttaa vähintään yksi perustietora-



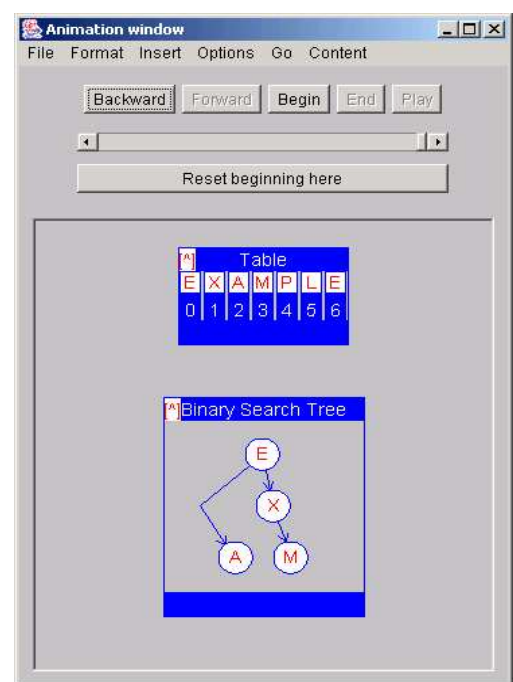
(A) 4-paikkainen taulukko, jossa on avaimet E, X, A ja M.



(B) Taulukon koko kasvatettuna 7:ään.



(C) Taulukkoon on lisätty avaimet P, L ja E ja taulukon avaimista on allaolevaan binääriseen hakupuuhun lisätty neljä ensimmäistä.



(D) Hakupuun visualisaatiota on muuttettu siten, että visualisoidaan tietorakennerakenne verkkona.

KUVA 3.1: Matrix-järjestelmän käyttöliittymä

puu ja verkko. Jokainen perustietorakenne visualisoidaan vastaavan visualisaatio-konseptin mukaan. Käytännössä tämä tarkoittaa sitä, että esimerkiksi kaikki binääripuut visualisoidaan saman konseptin mukaan. Visualisaatiokonsepti paitsi esittää alla olevan tietorakenteen, niin se myös tarjoaa käyttäjälle toimintoja, joilla rakennetta voi manipuloida tai sen esitystapaa vaihtaa. Esimerkiksi taulukon visualisaatiokonsepti tarjoaa käyttäjälle mahdollisuuden kasvattaa taulukon kokoa käyttöliittymäoperaatiolla, joka vastaa graafisten käyttöliittymien ikkunan suurennusoperaatiota; hiirellä tartutaan objektin reunaan ja vedetään se halutun kokoiseksi (Kuvat 3.1 A ja 3.1 B).

Rakenteiden visuaalisia esityksiä muokkaamalla alla oleva tietorakennekin voi muuttua. Esimerkiksi edellä kuvattu taulukon visuaalisen esityksen koon kasvattaminen kasvatti myös alla olevan taulukon kokoa muistissa. Samalla tavoin tietorakenteiden sisältöä voi muokata käyttöliittymäoperaatioilla, kuten esimerkiksi vetämällä avaimia rakenteesta toiseen. Tämä mahdollistaa algoritmisisimulaation, joka on tietorakenteiden manipulointia graafisen käyttöliittymän operaatioilla.

Kuvassa 3.1 C avain P voitaisiin lisätä binääriseen hakupuuhun kahdella eri tavalla algoritmisisimulaatiota käyttäen. Ensinnäkin voidaan käyttää hyväksi tietorakenteen olemassa olevaa insert-rutiinia. Tällöin avain pudotetaan hakupuuhun visualisaation otsikkopalkkiin (jossa lukee "Tree"). Insert-rutiini aktivoituu ja lisää avaimen oikeaan paikkaan hakupuussa. Toinen tapa lisätä avain on vetää se taulukosta puun M-solmun oikeaan alipuuhun. Jos kyse olisi binääristä hakupuuta koskevasta harjoitustehtävästä, niin jälkimmäisessä tapauksessa oppimisympäristö voisi tarkastaa operaation oikeellisuuden ja antaa opiskelijalle palautetta.

Algoritmisisimulaatiota käytetään hyväksi Matrixin ohjelmistokomponentteja käyttäen rakennetussa TRAKLA2-harjoitustehtäväohjelmistossa, jossa opiskelijoiden tehtävänä on muokata käsillä olevia tietorakenteita kuten tehtävässä opeteltava algoritmikin olisi niitä muokannut.

Tietorakenteiden ja algoritmien oppimisen kannalta Matrixissa on muitakin hyödyllisiä ominaisuuksia. Matrixin mukana tulee kokoelma erilaisia tietorakenne- ja algoritmitoteutuksia, joiden toimintaa tutkimalla opiskelija voi muodostaa käsityksensä niiden toiminnasta. Matrixin tietorakennekirjastoon kuuluu mm. erilaisia tasapainotettuja puita ja kekorakenteita. Tyypillisiä Matrixiin toteutettuja algoritmeja ovat mm. hakualgoritmit verkoissa.

Matrixia voi myös käyttää omien tietorakennetoteutuksien toiminnan tutkimiseen. Omat tietorakennetoteutukset voi ladata Matrixiin sen dynaamista luokkalataustointia käyttäen. Kun itse tehty tietorakenne toteuttaa tietyt rajapinnat, niin Matrixin konseptivisualisointi osaa visualisoida rakenteet. Omien rakenteiden animointi on myös mahdollista, mutta tällöin omat tietorakenneluokat olisi perittävä Matrixin kirjastotietorakenteista.

Kuvassa 3.1 D on esimerkki tietorakenteen esitystavan muuttamisesta. Siinä binäärikennetta vastaava rajapinta.

rinen hakupuu esitetään verkkona. Esitystavan muutos ei vaikuta muistissa olevaan binäärisen hakupuun tietorakenteeseen.

Tietorakenteita visualisoivia järjestelmiä on esitetty alan kirjallisuudessa. Eräs tietorakenteita visualisoiva ja niiden muutoksista animaation rakentava järjestelmä on JDSL Visualizer [4]. Opiskelija voi visualisoida järjestelmällä omia tietorakennetöteutuksiaan, jotka toteuttavat vaadittavat kirjastorajapinnat. Tietorakenteen visualisaation kanssa voi myös olla vuorovaikutuksessa; järjestelmä sallii tietorakennetöteutuksen metodien kutsumisen käyttäjän määrittelemillä parametreilla. Matrixin algoritmisimulaation kaltainen tietorakenteiden manipulointi ilman tiedon syöttöä näppäimistöltä ei kuitenkaan ole mahdollista.

3.5 TRAKLA2

TRAKLA:n, WWW-TRAKLA:n ja Matrixin ajatuksia sekä ohjelmistokomponentteja hyväksi käyttäen on kehitetty taas uusi versio tietorakenteiden ja algoritmien kotehtäväjärjestelmästä, TRAKLA2 [26]. Järjestelmä otettiin käyttöön keväällä 2003 siten, että osa WWW-TRAKLAN kotehtävistä korvattiin TRAKLA2-tehtävillä. TRAKLA2-harjoitustehtävät ovat myös Java-sovelmia, joten ne oli mahdollista integroida WWW-TRAKLA:n yhteyteen.

TRAKLA2-harjoitustehtävä (Kuva 3.2) koostuu harjoitustehtävänannosta sekä interaktiivisesta harjoitustehtäväsovelmasta. Harjoitustehtäväsovelmien graafisella käyttöliittymällä opiskelija voi simuloida algoritmin toimintaa erilaisia visuaalisia symboleja manipuloimalla. Käyttöliittymässä näkyy algoritmin tietorakenteiden visuaalinen esitys, jota opiskelija voi muuttaa käyttöliittymäoperaatioilla. Harjoitustehtävän ideana on muuttaa tietorakenteita kuten harjoiteltava algoritmikin niitä muuttaisi. Opiskelija voi pyytää sovelmalta arvostelua ratkaisustaan, jolloin sovelma tarkastaa opiskelijan vastauksen ja antaa tästä välitöntä palautetta opiskelijalle. Opiskelija voi myös katsoa tehtävän malliratkaisun, joka on algoritmianimaatio algoritmin suorituksesta harjoitustehtävän lähtöarvoilla.

Algoritmiharjoitustehtävän lähtöarvot generoidaan satunnaisesti harjoitustehtävän alustuksen yhteydessä. Esimerkiksi kuvan 3.2 kekotehtävän lähtöarvoina on 15 aakkosta, joissa ei ole duplikaatteja. Tehtävän voi tällöin alustaa $26 \cdot 25 \cdot \dots \cdot 13 \cdot 12 = 1,0 \cdot 10^{19}$ tavalla. Tehtävän voi alustaa uudestaan milloin tahansa Reset-painikkeesta.

Tehtäviä ratkotaan manipuloimalla tietorakenteiden visuaalisia esityksiä veto ja pudotus -operaatioilla. Joissain tehtävissä saattaa tämän lisäksi olla erikoistoimintoja, joilla on omat toimintopainikkeensa. Esimerkiksi kuvan 3.2 kekotehtävässä alkion poistamiselle on Delete-painike. Muita toimintopainikkeilla suoritettavia toimintoja ovat esimerkiksi tasapainotettujen puiden rotaatio-operaatiot. Opiskelija voi tutkia vastaustaan askel askeleelta Back- ja Forward-painikkeilla.

Tehtävän malliratkaisu avautuu omaan ikkunaansa Model answer -painikkeesta. Malliratkaisu on algoritmianimaatio algoritmin suorituksesta tehtävän lähtöarvoilla.

TAULUKKO 3.1: TRAKLA2-kotitehtäväjärjestelmän algoritmiset harjoitustehtävät TKK:n kevään 2003 kursseilla.

Nimi	Kuvaus	Kurssi (T tai Y)
Lisäys binääriseen hakupuuhun	Tehtävässä tyhjään binääriseen hakupuuhun lisätään 17 satunnaista avainta vetämällä ja pudottamalla avaimet oikeisiin kohtiin binääristä hakupuuta.	Y
Poisto binäärisestä hakupuusta	Tehtävässä binäärisestä hakupuusta poistetaan 4 avainta manipuloimalla puuta algoritmisimulaatio-operaatioilla. Puussa on alunperin 15–20 avainta.	Y
Lisäys AVL-puuhun	Tehtävässä tyhjään AVL-puuhun lisätään 13 satunnaista avainta. Tarvittaessa puu tasapainotetaan rotaatio-operaatioilla.	Y
Lisäys punamustaan puuhun	Tehtävässä tyhjään punamustaan puuhun lisätään 10 satunnaista avainta. Tarvittaessa solmujen väriä vaihdellaan punaisen ja mustan välillä ja puu tasapainotetaan rotaatio-operaatioilla.	Y
Lisäys digitaaliseen hakupuuhun	Tehtävässä lisätään avaimia digitaaliseen hakupuuhun.	Y
Radix search trie insertion	Tehtävässä lisätään avaimia radix-hakupuuhun.	Y
Keon muodostuminen	Tehtävässä lisätään avaimia binäärikekkoon. Tarvittaessa kekoehto on palautettava voimaan manipuloimalla keon binääripuuesitystä algoritmisimulaatio-operaatioilla.	T
Keon operaatiot	Tehtävässä lisätään avaimia binäärikekkoon kuten edellisessä tehtävässä. Lopuksi poistetaan kolme pienintä avainta delete min -operaatioilla.	T
Puolitushaku	Tehtävässä simuloidaan hakualgoritmin toimintaa listaamalla 30-paikkaisen taulukon avaimet, joissa algoritmi vierailee.	T
Interpolaatiohaku	Vrt. puolitushakutehtävä.	T
Puun läpikäyntialgoritmit (esi-, sisä-, jälki- ja tasojärjestys)	Tehtävissä listataan annetun binääripuun avaimet siinä järjestyksessä, kun ko. algoritmi niissä vierailee.	T

Malliratkaisun askelia voi kelata eteen- ja taaksepäin Back- ja Forward -painikkeilla. Malliratkaisun avaamisen jälkeen tehtävää voi edelleen tehdä, mutta sitä ei voi arvostella eikä palauttaa ennen kuin tehtävän on alustanut uudestaan.

Grade-painikkeesta opiskelijan vastaus tarkastetaan malliratkaisualgoritmeilla ja arvostellaan välittömästi. Arvostelussa annettava palaute kertoo kuinka monta askelta opiskelija on saanut oikein algoritmin kaikista askeleista (Kuva 3.3). Arvostelun jälkeen on mahdollista palauttaa vastaus kurssin tietokantaan tai yrittää ratkaista tehtävä uudestaan. Tehtävän ratkaisuyrityksien määrää ei ole rajoitettu, mutta uusintayrityksillä tehtävä on alustettava uudestaan ja ratkaistava alusta asti. Arvostelun voi suorittaa vaikka vastausta ei olisikaan vielä tehty valmiiksi.

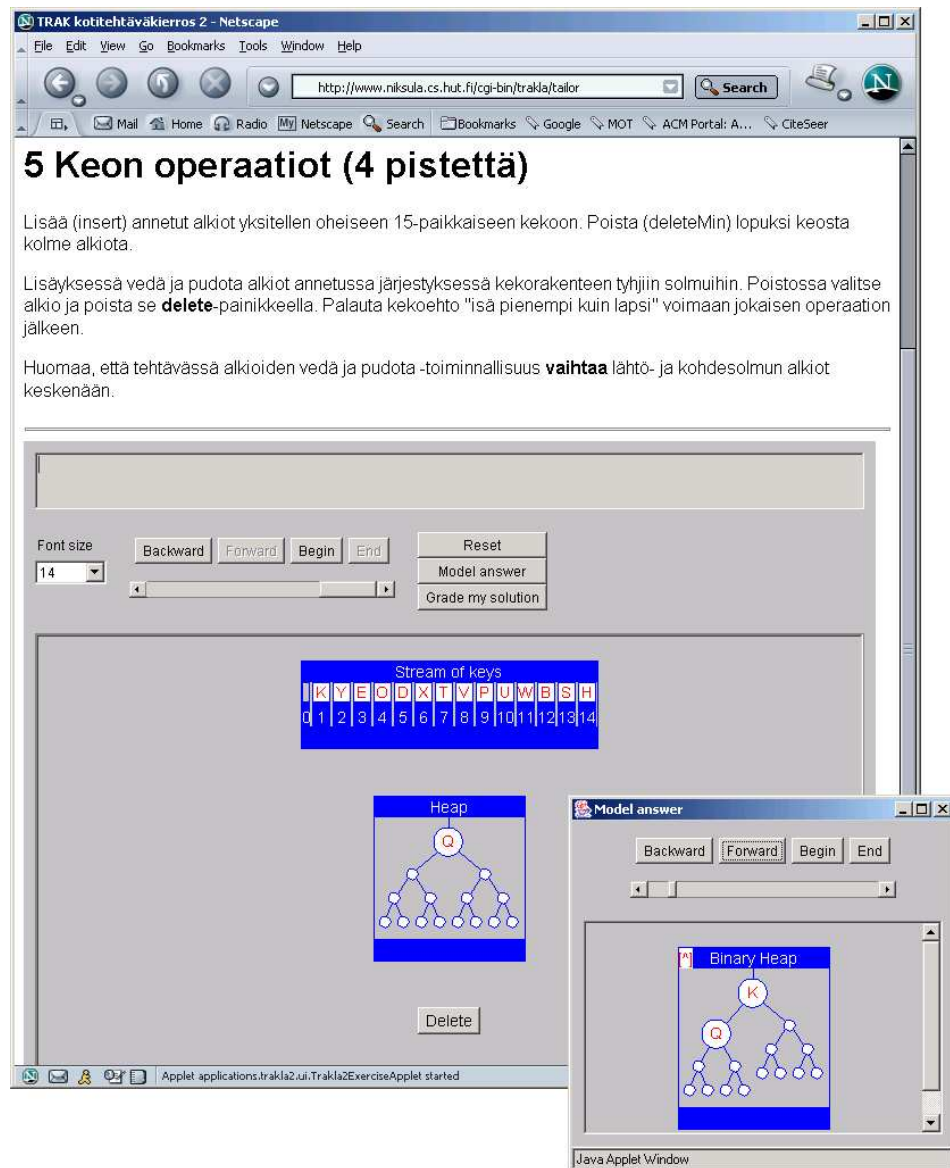
TRAKLA2-harjoitustehtävät olivat ensimmäistä kertaa käytössä keväällä 2003 järjestetyllä Tietorakenteet ja algoritmit -kursseilla. TRAKLA2-harjoitustehtävät olivat osa kurssien arvosteltavia osasuorituksia. Tietotekniikan koulutusohjelman opiskelijoille suunnattua T-kurssia suoritti n. 230 opiskelijaa ja muiden koulutusohjelmien opiskelijoille suunnattua Y-kurssia n. 390 opiskelijaa. T-kurssilla oli käytössä 8 TRAKLA2-harjoitustehtävää, Y-kurssilla 6. Keväällä 2003 käyttöön otetut TRAKLA2-harjoitustehtävät on kuvattu taulukossa 3.1.

Järjestelmä toimi muuten hyvin, mutta kaksi sen tekniseen toteutukseen liittyvää ominaisuutta aiheutti hieman ongelmia opiskelijoiden keskuudessa.

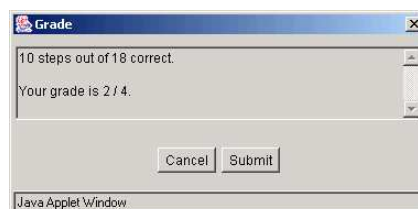
Ensinnäkin TRAKLA2-sovelmat vaativat Java 2 -yhteensopivan selaimen toimiakseen. Kävi kuitenkin ilmi, että useitten Java 2 -yhteensopiviksi väitettyjen selainten Java-toteutus oli puutteellinen eivätkä sovelmat toimineet niissä. Java-toteutus oli puutteellinen mm. Microsoft Internet Explorer -selaimessa sekä Netscape Navigator 7.x -selaimen ja joidenkin Linux-/Unix -käyttöjärjestelmän versiossa. Hyvin toimiviksi selain ja käyttöjärjestelmäyhdistelmiksi koettiin mm. Netscape Navigator 7.x selain Windows 2000/XP -järjestelmissä. Vaikka kaikki halukkaat eivät saaneetkaan TRAKLA2-järjestelmää toimimaan kotikoneissaan, niin TKK:n mikroloukissa oli yli 50 työasemaa, joissa TRAKLA2:n oli testattu toimivan moitteettomasti, joten selainyhteensopimattomuudet eivät estäneet kenenkään kotitehtävien tekemistä.

Toiseksi sovelmien ja palvelimen välinen tietoliikenne ei toiminut, jos asiakkaan päässä oli tiettyjen tietoliikenneporttien liikenteen tukkiva palomuri. Käytännössä tästä ei ollut haittaa kuin muutamalle työpaikallaan tehtäviä tehneelle opiskelijalle, koska yritysten tietoturvarajoitukset ovat yleensä tiukemmat kuin oppilaitosten tai opiskelijoiden kotitietokoneiden.

Opiskelijoiden suhtautuminen järjestelmään oli hyvin myönteistä. Kurssin lopuksi järjestettyyn palautekyselyyn vastasi 364 opiskelijaa, joista 94% piti järjestelmää erittäin tai kohtuullisen helppokäyttöisenä. 80% opiskelijoista antoi järjestelmälle arvosanan 4 tai 5 asteikolla 0–5.



KUVA 3.2: TRAKLA2 -harjoitustehtäväsovelma; malliratkaisu on avattu omaan ikkunaansa.



KUVA 3.3: TRAKLA2 -palauteikkuna; arvostelun jälkeen on mahdollista palauttaa vastaus tai yrittää ratkaista tehtävä uudestaan.

3.6 Tutkimustuloksia

Automaattisen kotitehtäväjärjestelmän palaute harvoin on yhtä perinpohjaista kuin opettajan. Toisaalta kuitenkin automaattisen järjestelmän käyttämisessä on hyötyjä, joita henkilökohtaisessa ohjauksessa ei saavuteta. Palautteen välittömyys on etu perinteiseen opetukseen verrattuna. Perinteisessä luokkahuoneopetuksessa opettaja harvoin pystyy antamaan jokaiselle opiskelijalle kovin välitöntä palautetta heidän edistymisestään tehtävissä, kun taas TRAKLA antaa välittömästi palautetta jokaisesta tehtäväpalautuksesta, ja mahdollistaa myös tehtävän tekemisen uudelleen palautteesta oppia ottaen. Koska tehtävänannot ovat yksilöllisiä ja plagiointi lähes mahdotonta, niin TRAKLA:a käyttäviä opiskelijoita voidaan myös kannustaa yhteistyöhön algoritmien toiminnan opiskelussa.

Malmi et al. [22] suorittamassa evaluaatiotutkimuksessa TRAKLA:n käytöstä opetuksessa vertailtiin TRAKLA:a käyttäneiden opiskelijoiden ja kurssin perinteistä luokkahuoneopetusta saaneiden opiskelijoiden oppimistuloksia. Tutkimustulokset osoittivat, että Tietorakenteet ja algoritmit -kursilla, jonka harjoitukset olivat useita kymmeniä pieniä harjoitustehtäviä, ryhmien välillä ei havaittu eroa oppimistuloksissa. Kurssin keskeyttämisprosentit olivat myös samat ryhmien kesken, tosin TRAKLA:n kautta ohjausta ja palautetta saaneet opiskelijat keskeyttivät kurssin aikaisemmassa vaiheessa. Koska kyseessä on useiden satojen opiskelijoiden massakurssi, jolla riittävän luokkahuoneopetuksen ja henkilökohtaisen ohjauksen järjestäminen vaatisi valtavasti resursseja, niin tulos puhuu vahvasti automaattisen kotitehtäväjärjestelmän käyttämisen puolesta.

Luku 4

Tutkimuksen pääkysymys ja ongelmat

4.1 Taustaa

TKK:n Tietojenkäsittelyopin laboratorion tutkimushankkeena tehty TRAKLA2-kotitehtäväjärjestelmän ensimmäinen versio saatiin viimeisteltyä tuotantokäyttöön soveltuvaksi vuodenvaihteessa 2003. Neljäätoista TRAKLA2-kotitehtävää oli tarkoitus kokeilla TKK:n kevään 2003 Tietorakenteet ja algoritmit -kursseilla pakollisena osasuorituksena. Läpäistäkseen kurssin kaikkien opiskelijoiden olisi siis suoritettava hyväksytysti ainakin muutama TRAKLA2-kotitehtävä. Hallinnollisista syistä Tietorakenteet ja algoritmit -kurssi on jaettu kahteen osaan, joiden sisältö on kuitenkin sama. Kurseille oli yhteensä odotettavissa noin 500 opiskelijaa.

Idea tähän työhön syntyi huomiosta, että TRAKLA2-kotitehtävien Java-sovelmiin perustuva arkkitehtuuri mahdollistaisi lokitietojen tallentamisen siitä, miten opiskelijat sovelmaa käyttävät. Sovelmien on joka tapauksessa oltava yhteydessä palvelimeen, joka määrittelee tehtävien satunnaiset alkuarvot ja tallentaa pistetiedot. Järjestelmän asiakas-palvelin -arkkitehtuuria olisi teknisesti mahdollista käyttää muunkinlaisen tiedon, kuin pistetietojen, tallentamiseen palvelimelle.

Graafisena käyttöliittymänä TRAKLA2-harjoitustehtävät ovat yksinkertaisia ja yhdenmukaisia. Perusoperaatiot, joita ovat tehtävän alustus, mallivastauksen avaaminen, tehtävän arvostelu ja palautus, ovat kaikissa tehtävissä samat. Tehtävän ratkaisu määritellään kaikissa tehtävissä Matrix-sovelluskehityksen algoritmisimulaatio-operaatioilla. Tämä herättää ajatuksen, että tiedon tallentaminen käyttöliittymäoperaatioista voisi olla yksinkertaista, samoin kuin tämän tiedon analysointi.

Tehdäänpä kerätyn tiedon perusteella sitten kvantitatiivista tai kvalitatiivista tutkimusta, on suuresta tutkimusaineistosta hyötyä. TRAKLA2 on ihanteellinen työkalu opetusohjelman käytön tilastolliseen tutkimukseen, koska vuosittain opetusohjelmaa käyttää noin 500 opiskelijaa. Hyvällä datan keräämis- ja analysointimenetelmällä on

mahdollista tutkia riittävän suuria otoksia tilastollisesti merkittävien tulosten saamiseksi.

Tutkimusongelman määrittely lähti siis huomiosta, että teknisesti on mahdollista kerätä tilastollista dataa harjoitustehtäväsovelman käytöstä. Perusteen datan keräämiselle on löydyttävä ongelmasta tai ongelmista, joihin dataa analysoimalla voidaan etsiä vastausta. Datan keräämisen motivaatio vaikuttaa luonnollisesti myös kerättävän datan tietomalliin. Tutkimusongelman määrittelyyn voidaan ajatella jakaantuvan niiden tutkimusongelmien määrittelyyn, joihin datan keruulla voidaan etsiä vastausta sekä datan keruumenetelmän teknisen toteutuksen määrittelyyn. Datan keruun teknisen toteutuksen täytyy tehdä edellä mainittujen ongelmien tutkimisen kerätystä aineistosta mahdolliseksi.

4.2 Datan keräämisen motivaatio

Seuraavissa kappaleissa kuvataan niitä tutkimusaloja ja tutkimusongelmia, joihin kehitettävällä datan keruumenetelmällä voidaan kerätä aineistoa.

4.2.1 Algoritmien oppimisen tutkiminen

Opiskelijoiden oppimis- ja ongelmanratkaisuprosessin on vaikutettava siihen, miten he harjoitustehtäväsovelmia käyttävät. Opiskelija hahmottaa opittavan asian opetusohjelman eri toimintoja käyttämällä sekä luonnollisesti etsimällä tietoa oppikirjoista, www-lähteistä ja muusta oppimateriaalista. On myös mahdollista, että opiskelija käyttää kynää ja paperia tai muita opetusohjelmia, kuin TRAKLA2:sta harjoitustehtävään liittyvän asian opiskeluun. Viime kädessä itse tehtävän vastaus kuitenkin konstruoidaan TRAKLA2-järjestelmän käyttöliittymällä. Voidaan myös olettaa, että useat opiskelijat käyttävät TRAKLA2-järjestelmän palautetoimintoa sekä mallivastauksia asian opiskeluun.

Tästä herää kysymys, että voiko opiskelijoiden käyttöliittymässä suorittamien operaatioiden perusteella päätellä jotain opittavan asian luonteesta, vaikeudesta ja ymmärrettävyydestä? Kertooko opiskelijoiden toiminta harjoitustehtävissä jotain siitä, miten tietorakenteet ja algoritmit opitaan ja ymmärretään? Voiko tämän osoittaa tilastollisin menetelmin? Koska tämä työ on pääasiassa tekninen raportti datan keräysmenetelmän toteutuksesta eikä raportin teoriaosassa esitetä oppimiseen liittyviä teorioita, niin kehitetyn menetelmän soveltuvuutta algoritmien ja tietorakenteiden oppimisen tutkimiseen selvitetään tarkastelemalla kerättyä dataa muutamasta oppimisen kannalta itsestään selvistä näkökulmista. Ensinnäkin oletetaan, että harjoitustehtävien vaikeusaste vaihtelee ja yritetään selvittää tehtävien keskinäinen vaikeus niiden käytöstä kerätyn datan perusteella. Toiseksi, tutkitaan opiskelijoiden toimintaa tietyssä harjoitustehtävässä ja yritetään selvittää tehtävässä käsiteltävän algoritmin osien keskinäinen vaikeus.

Edellä kuvattu tarkastelu pyritään suorittamaan sillä tavoin, että tarkasteltavien tehtävien ja algoritmien vaikeuden erot ovat itsestään selviä tietojenkäsittelyopin perusteisiin perehtyneelle. Analyysillä pyritään vain osoittamaan, että kehitetyn menetelmän avulla saatu tieto vastaa itsestään selväksi tiedettyä.

Kirjallisuudessa on myös esimerkkejä siitä, että opetusohjelmaa on käytetty oppimisprosessin tutkimiseen. On tutkittu esimerkiksi opiskelijoiden itsearviointitaitoja [35]. Tämän kaltaisessa tutkimuksessa teknologia ei näyttele pääosaa, vaan tarjoaa vain menetelmän opiskelijoiden oppimis- ja ongelmanratkaisuprosessin tutkimiseen. Tutkimusaineisto kerätään opiskelijoiden toiminnasta opetusohjelmassa.

4.2.2 Opetusteknologian sovellusten tutkiminen

Opetusteknologian tutkimuksessa kehitetään ja kokeillaan erilaisia teknologioita. Opetusteknologian elinkaari alkaa yleensä teoreettisesta tarkastelusta. Teorian perusteella voidaan kehittää ohjelmistoprototyyppejä, joita tutkimalla saadaan lisää tietoa tutkittavasta teknologiasta. Hyväksi todettu ohjelmistoprototyyppi voidaan tuotteistaa opetusohjelmaksi, antaa se opiskelijoiden käytettäväksi jollain kurssilla ja suorittaa evaluaatiotutkimuksia siitä, miten opetusohjelmassa käytetyt opetusteknologiat vaikuttavat opiskelijoiden oppimiseen.

Tutkittavia opetusteknologian sovelluksia voivat esimerkiksi olla informaation visualisointi jossain muodossaan, automaattinen tarkastus ja palaute sekä adaptiivinen tutorointi tai muiden älykkäiden ominaisuuksien soveltaminen. Yleensä tutkimuksissa opiskelijat jaetaan ryhmiin, joiden käyttämistä opetusohjelman versiosta tutkittava toiminto on toteutettu hieman eri lailla. Tutkittavan toiminnon vaikutusta oppimistulokseen voidaan mitata esi- ja jälkitentteillä, mutta myös sillä, miten opiskelijat järjestelmää käyttävät – esimerkiksi vaikuttaako käytetty kognitiivinen työkalu tai menetelmä opetusohjelman harjoitustehtävissä tehtyjen virheiden määrään tai laatuun. Tämän kaltaisessa tutkimuksessa on oleellista kerätä dataa opiskelijan toiminnasta opetusohjelmassa.

Hyvä esimerkki useita vuosia kestäneestä opetusohjelman kehitysprosessista, jossa ohjelmaa arvioitiin useiden evaluaatiotutkimusten kautta, on SQL-Tutor -ohjelman kehitys. Vuosien 1999–2000 aikana ohjelman käytöstä tehtiin kuusi evaluaatiotutkimusta, joissa tutkittiin käytettyjen opetusteknologioiden vaikutusta opiskelijoiden oppimistuloksiin [31, 32, 35, 36, 37, 38, 39]. SQL-Tutorista tehtyjä evaluaatiotutkimuksia on kuvattu tarkemmin luvussa 2.

Opetusteknologian tutkimuksen kannalta TRAKLA2 voidaan ajatella tutkimusalueena, jolla voidaan suorittaa empiiristä tutkimusta eri opetusteknologioiden soveltamisesta tietorakenteiden ja algoritmien opetukseen. Sovellettavia opetusteknologioita voivat esimerkiksi olla ohjelmistojen visualisointi eri muodoissaan, visuaalinen ohjelmointi tai automaattinen ja välitön palaute. Uuden tiedon löytäminen edellä mainittujen teknologioiden käyttämisestä vaatii huolellista koeasetelmien määrittelyä ja datan keräämistä satojen opiskelijoiden toimista useiden vuosien ajan. Tämän tutki-

muksen puitteissa ei ole vielä mahdollista esitellä uutta tietoa opetusteknologioiden soveltamisesta, mutta toimivan datan keruu- ja analysointimenetelmän kehittämistä voidaan myös ajatella tieteellisenä kontribuutiona.

Alan kirjallisuudessa ei ole esitetty yhtä laajassa tuotantokäytössä olevia vastaavia algoritmivisualisaatiota, algoritmien visuaalisten esityksen graafista manipulointia ja automaattista palautetta hyödyntäviä tietorakenteiden ja algoritmien opetusohjelmia. Tämä osaltaan myös tekee TRAKLA2-ohjelmasta mielenkiintoisen alustan empiirisen tutkimuksen tekemiseen edellä mainituista opetusteknologioista.

Esimerkiksi mallivastausalgoritmianimaatio on eräs opetusteknologinen sovellus TRAKLA2-opetusohjelmassa. Mallivastausalgoritmianimaation merkitystä oppimistulokseen voitaisiin tutkia etsimällä yhteyksiä mallivastauksen katsomisen ja harjoitustehtävän tekoajan tai virheiden määrän ja laadun kanssa.

Toinen TRAKLA2-opetusohjelmassa käytetty opetusteknologinen sovellus on automaattinen palaute. Palautetoiminnon merkitystä oppimiseen voitaisiin tutkia etsimällä yhteyttä toiminnon käyttämisen ja harjoitustehtävän tekoajan, virheiden määrän ja laadun tai mallivastauksen katsomisten välillä. Mielenkiintoisia tutkimusasetelmiä saisi myös palautteen välittömyyden tai yksityiskohtaisuuden ja edellä mainittujen tekijöiden yhteyksiä etsimällä.

Tämän tutkimuksen kannalta on oleellista pystyä osoittamaan, että kehitetty menetelmä voi soveltua edellä mainittujen tutkimusasetelmien tekemiseen.

4.3 Analysoitavat operaatiot

Mitä TRAKLA2-opetusohjelman käyttöliittymäoperaatioita voidaan haluta analysoida edellisessä kappaleessa selitettyjen tutkimusmotivaatioiden valossa?

Opetusohjelmassa, jossa tehtävän ratkaisua voi yrittää useampaan kertaan, ratkaisuyritysten lukumäärä voi kertoa jotain opiskelijan osaamisesta tehtävässä. Ratkaisuyritysten lukumäärää on käytetty osaamista kuvaavana muuttujana esimerkiksi SQL-Tutor -ohjelmasta tehdyssä evaluaatiotutkimuksessa [38].

Opiskelijoiden virheet ja väärinkäsitykset tehtävissä voivat myös kertoa jotain algoritmien oppimisesta tai opiskelijan osaamisesta. Tehtävässä tehtyjen virheiden lukumäärän perusteella voi olla mahdollista päätellä jotain opiskelijan osaamistasosta tehtävän algoritmissa tai siinä algoritmin osassa, jossa virheet ovat esiintyneet. Esimerkiksi SQL-Tutor -ohjelman käytön analyysissä opiskelijoiden tekemien virheiden lukumäärää ja laatua on käytetty opiskelijan osaamisen luokitteluun [38].

Sen analysointi, kuinka paljon aikaa opiskelijat käyttävät tehtävän tekemiseen ja miten ajan käyttö jakautuu, voi myös olla hedelmällistä ohjelman käytön analyysin kannalta. SQL-Tutorin evaluointitutkimuksessa tehtävän tekemisaikaa käytettiin mittaamaan osaamista tehtävässä [38].

Voidaan myös olettaa, että tehtävää tehdessä käytetään aikaa myös muuhun oppimateriaaliin tutustumiseen ja miettimiseen. Tämän oletuksen valossa voidaan esittää kysymys, että voiko tehtävää tehdessä pidettyjen taukojen määrä ja pituus kertoa tehtävän käsittelemän asian opittavuudesta? Kehitettävän tiedonkeruumenetelmällä olisi pystyttävä selvittämään kuinka pitkä aika tehtävää on tehty aktiivisesti ja kuinka pitkän ajan TRAKLA2-opetusohjelma on ollut käynnissä muuhun materiaaliin tutustumisen ajan.

Malliratkaisun katsomisen on kerrottava jotain opiskelijan mielenkiinnosta opittavaan asiaan. Voisiko TRAKLA2-opetusohjelman malliratkaisutoiminnon käyttämisestä, kuinka usein malliratkaisua on katsottu ja miten sitä on katsottu, kerätä merkityksellistä tietoa?

4.4 Datan keräämisen teknisen toteutuksen vaatimukset

TRAKLA2-järjestelmän arkkitehtuuri ja sovelluksen käytettävyydelle asetetut vaatimukset asettavat myös omat vaatimuksensa datan keruumenetelmän tekniselle toteutukselle.

Teknisesti kysymys on ohjelmistotekniikan ongelmasta: onko ohjelmistoteknisesti mahdollista ohjelmoida TRAKLA2-sovelmiin datan keräystoiminto, joka tallentaa verkon yli dataa tietokantapalvelimelle opiskelijoiden toimista käyttöliittymässä? Datan keräystoiminnon voidaan ajatella koostuvan kerättävän datan tietomallista, tietoliikennetarkaisusta sekä tietokantaratkaisusta. Minkälainen tietomalli parhaiten kuvaa niitä käyttöliittymäoperaatioita, joista halutaan kerätä dataa? Sopivien analyysityökalujen kehittäminen on myös oma ohjelmistotekninen ongelmansa. Miten kerätty data muutetaan sellaiseen muotoon, että sitä voidaan analysoida tavanomaisilla tilastollisen analyysin ohjelmistoilla, kuten esimerkiksi Excel tai SPSS?

Menetelmän teknisen toteutuksen on oltava riittävän tehokas, jotta datan kerääminen ei häiritse kotitehtävän tekemistä. Kotitehtäväsovelma toimii opiskelijan selaimessa ja lähettää dataa tietoliikenneyhteyden yli palvelimelle. Kotitehtävien on oltava käytettävissä opiskelijoiden kotoa ISDN- tai laajakaistayhteyksien kautta, mikä asettaa datan keräämismenetelmälle teknisiä rajoituksia: tietoliikenteen sovelman ja palvelimen välillä on toimittava niin jouhevasti, että tehtävän käyttöliittymäoperaatioiden välillä ei ole häiritsevän pitkiä odotusaikoja.

TRAKLA2-sovelmat on tehty Javan versiolla 1.2 ja Matrix-sovelluskehityksen komponentteja käyttäen, mikä asettaa omat rajoituksensa sille, miten datan keräys sovelmissa ohjelmoidaan. Tietokantaratkaisun ja datan analysoinnin suhteen tekniselle toteutukselle ei ole vastaavia rajoitteita.

Kaikkia datan keruumenetelmän sovellusalueita on mahdotonta vielä tietää, ja järjestelmän toiminnalliset vaatimukset tulevat muuttumaan tutkimuksen edetessä. TRAKLA2-opetusohjelmisto on myös jatkuvan kehitystyön kohteena: siihen lisätään uusia toimintoja ja vanhoja toimintoja muutetaan. Tämän vuoksi datan keruume-

netelmän muunneltavuus on eräs tarkastelukriteeri, kun arvioidaan menetelmän teknistä toteutusta. Keskeiset vaatimukset muunneltavuudelle ovat seuraavat: uusien datan keruutoimintojen ohjelmoinnin on oltava mahdollisimman yksinkertaista ja datan täytyy olla sellaisessa muodossa, että useiden vuosien ajalta kerätty data on vertailukelpoista.

4.5 Yhteenveto

Tämän tutkimuksen tavoitteena on kehittää menetelmä TRAKLA2-opetusohjelman käytön tutkimiseen. Menetelmällä kerätyllä datalla on voitava tehdä päätelmiä opiskelijoiden oppimisesta tilastollisia menetelmiä käyttäen. Menetelmän avulla on voitava evaluoida opetusohjelman ja sen eri toimintojen käyttämisen merkitystä oppimisen kannalta. Datan keruun lisäksi on esitettävä, miten kerätty data muutetaan muotoon, josta tilastollista analyysiä voidaan tehdä yleisimpiä tilastollisen analyysin ohjelmia käyttäen.

Menetelmän teknisen toteutuksen täytyy toimia niin hyvin, että datan keruu ei häiritse opiskelijan toimintaa harjoitustehtäväsovelmassa. Ohjelmistoteknisestä näkökulmasta ajatellen, toteutetun menetelmän täytyy olla muunneltavissa tutkittavan TRAKLA2-opetusohjelman mahdollisten muutosten myötä.

Luku 5

Datan keräämisen toteutus

Tässä luvussa esitetään tutkimuksessa kehitetty datan keruumenetelmä sekä datan tilastollisen analyysin ohjelmistotyökalut ja menetelmät.

5.1 Kerättävä data

Mitä sellaisia toimintoja TRAKLA2-harjoitustehtävien käyttöliittymässä voi tehdä, joista voi kerätä dataa ja minkälaisen datan keruun käyttöliittymän ohjelmistokomponentit mahdollistavat. Mitä vaatimuksia luvussa 4 määritellyt tutkimusongelmat asettavat kerättävälle datalle?

Perusta datan keräämiselle on yksinkertainen tapahtumaloki, johon tallennetaan tiedot harjoitustehtävän tekemisen aloittamisesta ja päättämisestä. Myös mallivastauksen katsomisesta ja palautetoiminnon käyttämisestä tallennetaan merkintä. Jokaisen lokimerkinnän yhteyteen tallennetaan operaation nimi, aikaleima sekä opiskelijan ja tehtävän yksilöivät tiedot. Vastaavaa yksinkertaista operaatiologia kerätään useissa muissa kirjallisuudessa esitetyissä harjoitustehtäväohjelmistoissa [3, 10, 34, 46].

TRAKLA2-harjoitustehtävien tehtävänannot alustetaan satunnaisilla alkuarvoilla. Jotta harjoitustehtävän tekemistä voitaisiin analysoida myöhemmin, on tehtävänanto voitava generoida uudestaan. Tämän vuoksi on joko alkuarvot tai satunnaisluku-generaattorin siemenluku tallennettava. Siemenluku on kokonaisluku, joten sen tallentaminen on yksinkertaisempaa kuin alkuarvojen, jotka voivat koostua esimerkiksi 20 paikkaisesta taulukosta. Harjoitustehtävän alkuarvojen siemenluku tallennetaan harjoitustehtävän käynnistymisen ja tehtävän alustuksen (Reset) yhteydessä.

Palautetta tehtävän ratkaisusta saa Grade-toiminnolla. Tällöin opiskelijan vastaus tarkistetaan tehtävänannon tarkastusalgoritmilla ja palautteena opiskelija saa tiedon, kuinka monta askelta hänen tehtävässään on oikein. Esimerkiksi jos opiskelijan vastauksessa on 8 askelta 10:sta oikein, tällöin hän on suorittanut algoritmin 8 ensimmäistä askelta oikein. Palautteen voi pyytää, vaikka ei olisikaan tehnyt tehtävää loppuun asti. Palautetoiminnon käytön tutkimisen kannalta olisi mielenkiintoista

tietää, kuinka palautetoimintoa on käytetty eri tehtävissä. Onko tehtävä tehty ensin loppuun asti ja tämän jälkeen pyydetty palautetta? Vai onko tehtävää tehty kokeilumenetelmällä siten, että palautetta on pyydetty aina muutaman operaation jälkeen, jolloin tehtävä on täytynyt alustaa uudestaan palautteen pyytämisen jälkeen. Onko tehtäväkohtaisia eroja sille kuinka palautetoimintoa on käytetty?

Opiskelijoiden virheet ja väärinkäsitykset tehtävissä voivat myös kertoa jotain algoritmien oppimisesta. Löytyykö algoritmiharjoitustehtävien vastauksista sellaisia virheitä, jotka ovat yhteisiä kaikille opiskelijoille tai tietyille opiskelijaryhmille? Voisiko virheiden lukumäärän tai tyyppin ja mallivastauksen katsomisten tai muiden operaatioiden välillä löytää yhteyksiä?

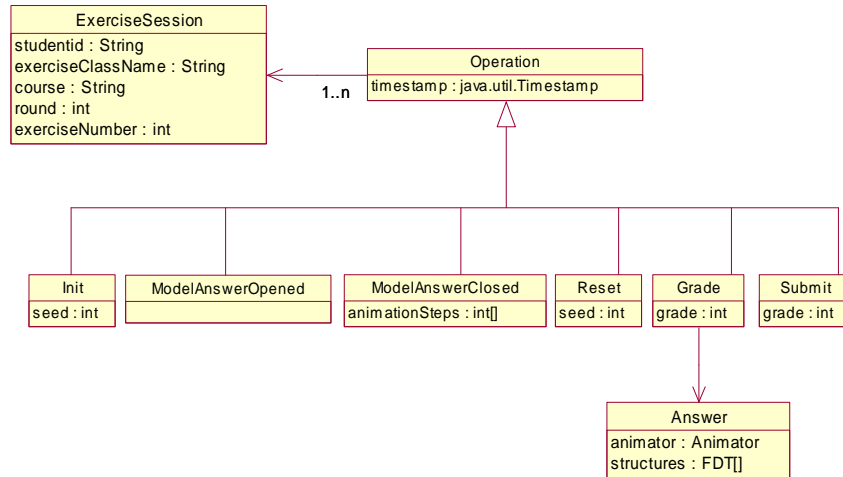
Palautetoiminnon käyttämistä ja opiskelijoiden tekemien virheiden määrää ja laatua voidaan tutkia, jos palautetoiminnon käytön yhteydessä tallennetaan harjoitustehtävävastaus, joka on opiskelijan määrittelemä algoritmisimulaatio algoritmin toiminnasta. Vastaus on sekvenssi tietorakenteiden manipulointioperaatioita, jotka opiskelija on tehtävässä suorittanut.

Mitä tietoa mallivastauksen katsomisesta voi kerätä? Mallivastaus on diskreetti animaatio algoritmin askeleista, miten algoritmin tietorakenteet ovat algoritmin operaatioiden vaikutuksesta muuttuneet. Animaatiota voi kelata eteen- ja taaksepäin ja mitä tahansa animaation askelta voi katsoa useaan kertaan. Tämä herättää kysymyksen, että voiko mallivastauksen algoritmianimaation katsominen kertoa jotain opiskelijan mielenkiinnosta näitä askeleita kohtaan ja mahdollisesti myös vastaavia algoritmin osia kohtaan? Voisiko algoritmianimaation katsomisesta päätellä jotain algoritmin oppimisesta ja kenties siitä, mitkä algoritmin askeleet koetaan vaikeimmiksi? Tätä tutkiaksemme mallivastauksen algoritmianimaation katsomisesta täytyy tallentaa tietoja tietokantaan.

Oletetaan, että tehtävää tehdessä käytetään aikaa myös muuhun oppimateriaaliin tutustumiseen ja miettimiseen. Tämän oletuksen valossa voidaan esittää kysymys, että voiko tehtävää tehdessä pidettyjen taukojen määrä ja pituus kertoa tehtävän käsittelemän asian opittavuudesta? Tämän tutkimiseksi tallennetaan tietokantaan tiedot jokaisesta yli minuutin pituisesta tauosta. Tietokantaan tallennetaan merkintä aina, kun harjoitustehtävän käyttöliittymässä ei ole suoritettu operaatioita minuuttiin, ja merkintä aina, kun tauko päätetään tekemällä jokin käyttöliittymäoperaatio. Tässä yhteydessä käyttöliittymäoperaatioksi lasketaan minkä tahansa painonapin painallus harjoitustehtävän käyttöliittymässä tai drag&drop -algoritmisimulaatiooperaatio. Pelkkää hiiren osoittimen liikuttamista ei lasketa käyttöliittymäoperaatioksi.

Operaatiolokiin tallennettujen aikaleimojen ja taukojen alkamis- sekä päättymisaikojen perustella voidaan päätellä myös tehtävän tekemiseen käytetty aika.

Toteutetussa tiedonkeruumenetelmässä sovelmat tallentavat tietoja Linux-palvelimella olevaan tietokantaan. Tietokantaan tallennetaan aikaleima, harjoitustehtävän tunniste sekä opiskelijan tunniste seuraavissa operaatioissa:



KUVA 5.1: Tietomallin UML-luokkakaavio

- harjoitustehtäväsovelman käynnistyminen (kun avataan sovelman sisältävä www-sivu)
- tehtävän alustus uusilla alkuarvoilla (Reset)
- malliratkaisun avaaminen (malliratkaisu avautuu omaan ikkunaansa Model answer -painikkeesta)
- malliratkaisun algoritmianimaation katsominen. Malliratkaisun algoritmianimaatiosta tallennetaan tieto siitä, kuinka monta kertaa opiskelija on kussakin algoritmin tilassa vierailut.
- malliratkaisun sulkeminen.
- tehtävän arvostelu. Tehtävän arvostelun yhteydessä tallennetaan tietokantaan myös opiskelijan vastaus.
- tauot tehtävän tekemisessä. Jos opiskelijan toiminnassa sovelmassa on yli minuutin kestäviä taukoja, niin näistä tallennetaan tauon alkamis- ja loppumisaika.

5.2 Tietomalli

Tässä kappaleessa esitetään, kuinka kerättävä data esitetään Java-ohjelmointikielen olioina.

Kuvassa 5.1 on UML-luokkakaavio kerättävästä tiedosta. Luokkien ominaisuudet on merkitty Java-tietotyypeillä. Sovelman käynnistyessä palvelin luo ExerciseSession-ilmentymän, jolle sovelma välittää seuraavat perustiedot: opiskelijan tunniste, harjoitustehtävän nimi, kurssin koodi, harjoitustehtäväkierroksen numero sekä kyseisen tehtävän numero kierroksella.

Operation-luokka on käyttöliittymäoperaatioiden yläluokka, jonka ainoana ominaisuutena on operaation aikaleima (timestamp), joka sisältää päiväyksen ja kellonajan millisekunnin tarkkuudella. Init-luokka pitää sisällään sovelman käynnistykseen yhteydessä tallennettavan tiedon. Vastaavasti Reset-luokka pitää sisällään tehtävän alustuksen yhteydessä tallennettavan tiedon. Nykyisessä tietomallissa sovelman käynnistykseen ja alustuksen yhteydessä palvelimelle lähetetään vain aikaleima, koska opiskelijan ja tehtävän perustiedot välitetään jo ExerciseSession-ilmentymän luonnin yhteydessä. Tehtävän siemenluku luodaan palvelimen päässä, joten sitäkään ei tarvitse lähettää uudelleen palvelimelle. Jatkossa datan keräystä voi kuitenkin laajentaa siten, että käynnistykseen ja alustuksen yhteydessä välitettäisiin joitain opiskelijan toimintaa kuvaavia tietoja.

Mallivastauksen sulkemisen tiedot tallennetaan ModelAnswerClosed-luokan ilmentymään. Kenttään animationSteps tallennetaan tiedot siitä, kuinka monta kertaa missäkin mallivastauksen tilassa on vierailtu. Esimerkiksi jos mallivastauksen algoritmianimaatioissa on kolme askelta, niin tällöin siinä on neljä tilaa ja animationSteps on tauluko, jonka koko on neljä. Taulukon indekseissä on niitä vastaavien tilojen vierailukumäärät.

Arvostelun yhteydessä luodaan Grade-luokan ilmentymä, johon tallennetaan opiskelijan vastaus. Vastaus sisältää ne tietorakenteet, joita tehtävässä on algoritmisimulaatio-operaatioilla muokattu sekä rakenteen, johon on tallennettu tietorakenteiden tilamuutokset. Matrix-sovelluskehikössä (luku 3.4) kaikki tietorakenteet ovat FDT-ilmentymiä¹. Tietorakenteiden tilamuutokset tallennetaan Animator-luokan² ilmentymään. FDT-tyyppinen taulukko sisältää harjoitustehtävän tietorakenteet, jotka voivat olla esimerkiksi taulukoita, puita tai verkkoja. Animator-ilmentymään on tallennettu tiedot FDT-tietorakenteiden alla oleviin muistirakenteisiin tehdyistä muutoksista, joita voivat olla esimerkiksi muutokset binäärisen hakupuun solmujen avaimissa tai osoittimissa. Yhdessä FDT-tilaukko ja animaattori sisältävät koko vastaussekvenssin.

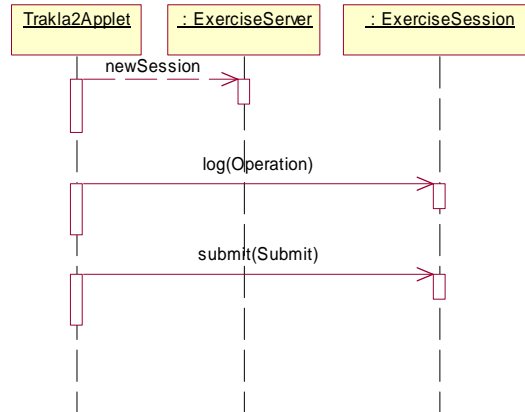
5.3 Järjestelmäarkkitehtuuri

Tässä kappaleessa esitetään sovelmien ja palvelimen välinen tietoliikenne sekä tietokantaratkaisu.

Sovelmien ja palvelimen välinen tietoliikenne on tehty käyttäen Java RMI (remote method invocation) hajautetun ohjelmoinnin protokollaa [44] käyttäen. Oheisessa kuvassa (Kuva 5.2) on UML-viestiyhteykskaavio sovelman ja palvelimen välisistä viesteistä. ExerciseServer- ja ExerciseSession-luokat ovat palvelimella olevia RMI-

¹FDT (fundamental data type) on Matrix-sovelluskehiksen rajapinta, jonka kaikki tietorakenteet toteuttavat.

²Matrixissa FDT-rakenteissa käytetään muuttujina Matrixin omia primitiivimuistirakenteita. FDT-rakenteiden tilamuutoksista voidaan tällöin pitää lukua tallentamalla vain muutokset primitiivimuistirakenteissa. Primitiivimuistirakenteiden tilamuutokset tallennetaan Animator-luokan ilmentymään.



KUVA 5.2: UML-viestiyhteyskaavio sovelman ja palvelimen välisistä viesteistä. ExerciseSession- ja ExerciseServer-luokat ovat RMI-palvelimen etäolioita, ja niiden metodien kutsuminen käynnistää tiedonsiirron RMI-protokollalla.

etäolioita, joiden etäkutsuilla sovelma lähettää lokitustiedot palvelimelle. Palvelimen newSession-metodikutsulla sovelma luo uuden istunto-olion, jonka kautta harjoitus-työistunnon yhteydenpito hoidetaan. Istunto luodaan aina, kun sovelma alustetaan avaamalla sovelman sisältävä www-sivu. Palvelimen log-metodikutsulla lähetetään Operation-luokan ilmentymä palvelimelle. Operation-luokan ilmentymä voi olla mikä tahansa sen aliluokan ilmentymä. Tehtävän palautus tehdään submit-metodikutsulla.

Tiedon tallennuksessa ei käytetä relaatiotietokantaa, vaan RMI-palvelinsovellus tallentaa sovelmilta tulevan lokitiedon suoraan Linux-palvelimen tiedostojärjestelmään lokitiedostoon sekä jäljempänä kuvattavaan hakemistorakenteeseen. Relatiotietokantatuotteen käyttämiselle ei nähty tarvetta, koska tiedon tallentaminen lokitiedostoon ja tiedostojärjestelmän on hyvin nopeaa eikä tallennetusta tiedosta tarvitse tehdä hakuja. Jos tiedon analysointivaiheessa halutaan käyttää relaatiotietokannan kyselyominaisuuksia, niin lokitiedostoon ja tiedostojärjestelmään tallennetun tiedon voi tallentaa eräajona myös tietokantaan. Relatiotietokannan käyttäminen olisi myös todennäköisesti lisännyt palvelinsovelluksen ohjelmakoodin määrää ja kompleksisuutta, koska olisi pitänyt käyttää Javan JDBC-tietokantakirjastoa (Java Database Connectivity).

Palvelin tallentaa tekstimuotoiseen lokitiedostoon rivin jokaista lokitettavaa operaatiota kohden. Lokitiedoston rivi koostuu puolipisteillä erotetuista nimi-arvo -pareista. Lokitiedostoon tallennettavat attribuutit on kuvattu taulukossa 5.1. Esimerkki lokitiedoston rivistä:

```

timeStamp=Fri Mar 21 18:47:37 EET 2003;operationName=grade;
courseCode=y2003;round=3;exerciseNumber=6;studentid=12345h;
exerciseClassName=content.exercises.RST_Insert;seed=1048265079785;
grade=1;fileName=/home/trakla2/public_html/data/submitted-files/
y2003/3/6/12345h/applications.trakla2.datalogging.GradeData/
1048265257794
  
```

Harjoitustehtävävastaus on Answer-luokan ilmentymä (Kuva 5.1) ja se tallennetaan palvelimen tiedostojärjestelmään Javan sarjallistamismekanismia [43] käyttäen. Hakemiston ja tiedoston nimi, johon sarjallistettu olio on tallennettu, koostuu hakemistorakenteesta, joka on määritelty kyseisen kurssin, tehtäväkierroksen, tehtävän numeron, opiskelijanumeron, operaation nimen ja lokimerkinnän aikaleimaa kuvaavan kokonaisluvun perusteella. Tiedostonnimi on kuvattu fileName-attribuutissa. Aikaleimaa kuvaava kokonaisluku on millisekunteina erotus nykyhetken ja 1.1.1970 klo 00:00 koordinoitua maailmanaikaa (UTC, coordinated universal time) [45] välillä.

Sarjallistettujen olioiden lukeminen levytä takaisin muistiin ei onnistu, jos lukuhetkellä käytössä on eri versio olion määrittelevästä luokasta kuin olioita tallennettaessa. Sarjallistamismekanismi vertaa lukuhetkellä ladatun luokan tavukoodissa määriteltyä 64-bittistä tarkistussummaa levyllä tallennettujen olioiden yhteyteen tallennettuun tarkistussummaan. Jos tarkistussummat eroavat, lukeminen ei onnistu. Tämä on ongelmallista, koska Javassa luokan tarkistussumma muuttuu kääntämisen yhteydessä, jos luokkaan on esimerkiksi lisätty kenttä. Ohjelmistojen kehittymisen myötä kenttien lisääminen on tyypillistä ja se ei saisi tehdä luokkien uusista versiosta yhteen sopimattomia vanhoilla versiolla tallennetun datan kanssa. Luokan tarkistussumma on lisätty tavukoodiin luokan staattiseksi vakioksi, jolloin edellä kuvatun ongelman voi kiertää määrittelemällä vakion itse luokan lähdekoodiin. Vakio määritellään luokkamuuttujalla *static final long serialVersionUID*. Vakion määrittelyn jälkeen versioyhteensopivuuden rikkoo vain muutokset kenttien nimissä tai tyypeissä, kenttien poistaminen tai muutokset perintähierarkiassa [43].

Palvelimen RMI-mekanismi on monisäikeinen [44] ja useita log-metodikutsuja saattaa olla käynnissä useissa rinnakkaisissa säikeissä. Hakemistorakenteen luominen tallennettaville tiedostoille edellyttää nykyisen hakemistorakenteen tutkimista ja tarvittaessa uusien hakemistojen luomista. Tämä on määriteltävä atomiseksi operaatioksi, koska muuten rinnakkain tiedostoja tallentavat säikeet aiheuttavat ongelmia. Levyllä kirjoittamisen salliminen vain yhdelle säikeelle kerrallaan voi aiheuttaa tehokkuuspullonkaulan suurta suorituskykyä vaativissa palvelinsovelluksissa. TRAKLA2-palvelinkoneen tapauksessa vaikutus palvelinsovelluksen tehokkuuteen on kuitenkin vähäinen, koska koneessa on vain yksi kiintolevy, jolloin levyllä voi joka tapauksessa kirjoittaa vain yksi prosessi kerrallaan.

5.4 Analyysityökalut

Dataa kerätään tekstimuotoiseen lokitiedostoon sekä sarjallistettuina Java-olioina palvelinkoneen tiedostojärjestelmään. Tässä työssä datan analyysi on rajoitettu vain lokitiedoston analysointiin, mutta suuntaviivat sarjallistettuina olioina tallennettujen harjoitustehtävävastausten analysointimenetelmiksi esitetään myös.

Lokitiedostoa voi analysoida mitä erilaisimmilla työkaluilla. Tässä työssä lokitiedostoa on analysoitu Microsoft Excel -taulukkolaskentaohjelmalla, jossa on myös toimintoja tilastollisen analyysin tekemiseen. Lokitiedoston muuttaminen Excel-

muotoiseksi voidaan tehdä Unix-komennolla:

```
cat trakla2.log | awk -F";" '{print $1,""$3,""$5,""$7}' > trakla2.csv
```

Yhteenvetojen tekemiseen soveltuu hyvin Excelin Pivot-taulukointi -toiminto. Keskilukuja voi laskea ja niiden tilastollisen merkittävyyttä testata luotettavuusvälien analyysillä, johon soveltuu CONFIDENCE-funktio. Korrelaatioiden analyysi ja niiden merkittävyyden testaus t-luvun laskemisen avulla onnistuu Excelin Data analysis -työkalujen toiminnoilla. Excelillä voi piirtää numeerisia arvoja havainnollistavia kaavioita.

Sarjallistettuina oliona tallennettujen harjoitustehtävävastausten analyysistä voidaan haluta kaivaa esille seuraavia tietoja:

1. Kuinka monta askelta harjoitustehtävästä on tehty ennen tarkastusta ja palautteen pyytämistä?
2. Kuinka montaa askelta opiskelijan vastauksessa on oikein (opiskelijan pisteet)?
3. Kuinka monta askelta kyseisen harjoitustehtävän mallivastauksessa on (tehtävän maksimipisteet)?
4. Mitä algoritmin operaatiota vastaavassa askeleessa virhe on sattunut? Esimerkiksi onko virhe sattunut yksinkertaisessa vai kaksinkertaisessa rotaatioissa?

TAULUKKO 5.1: Operaatiolokitiedoston attribuutit.

Attribuutti	Kuvaus
timeStamp	operaation aikaleima java.util.Date -luokan tallentamassa muodossa
operationName	operaation nimi, joka voi olla init, reset, model_answer_opened, model_answer_closed, grade, submit, idle_start tai idle_end
studentid	opiskelijan tunniste
courseCode	kurssin koodi
round	tehtäväkierroksen numero
exerciseNumber	tehtävän numero tehtäväkierroksella
exerciseClassName	Java-luokan nimi, jossa harjoitustehtävä on määritelty
seed	satunnaislukugeneraattorin siemenluku
grade	arvosana (vain grade- ja submit-operaatioissa)
fileName	tiedostonnimi, johon harjoitustehtävävastaus on tallennettu (vain grade-operaatioissa)
animationStepIndices	mallivastausanimaation tilojen vierailulukumäärät (vain model_answer_closed -operaatioissa)

Kolmen ensimmäisen kysymyksen osalta tiedot saa pääteltyä Java-ohjelmalla, joka lukee sarjallistetun vastauksen (Answer-ilmentymä) ja tarkastaa sen harjoitustehtävän määrittelyluokassa (SimulationExercise) määritellyllä mallivastausalgoritmilla. Ohjelma tulostaa opiskelijan suorittamien askeleiden lukumäärän, oikeiden askeleiden lukumäärän (opiskelijan pisteet) sekä kaikkien askeleiden lukumäärän (maksimipisteet).

Neljännän kysymyksen osalta tieto virheen tyyppistä on vaikeammin pääteltävissä eikä virheiden luokittelua tässä työssä ole toteutettu. Virheiden luokittelu edellyttäisi algoritmien askelten luokittelua, minkä voisi tehdä harjoitustehtävän määrittelyn yhteydessä ohjelmoitujen mallivastausalgoritmien ohjelmakoodissa. Esimerkiksi lisäyksessä AVL-puuhun algoritmin askeleet voisi luokitella lisäykseen, yksinkertaiseen rotaatioon sekä kaksinkertaiseen rotaatioon.

Luku 6

Tulokset ja johtopäätökset

Tässä luvussa esitetään analyysi tutkimuksessa kehitetyn datan keruu- ja analysointimenetelmän soveltuvuudesta algoritmien oppimisen ja opetusteknologian tutkimisen työkaluksi. Datan keruumenetelmän teknistä toteutusta arvioidaan sen tehokkuuden ja muunneltavuuden näkökulmasta.

6.1 Algoritmien oppimisen tutkimisen työkalu

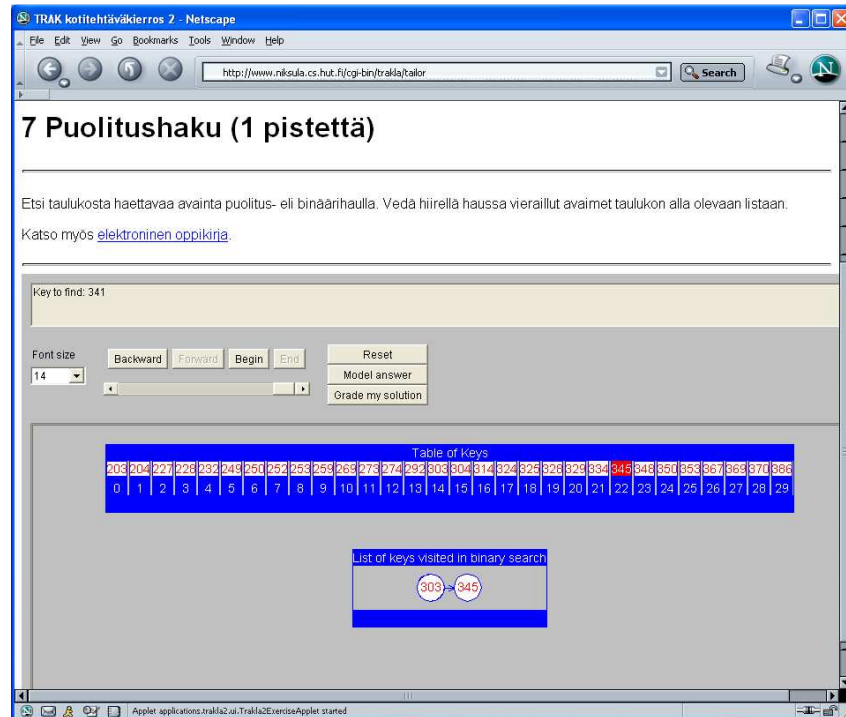
Voiko opiskelijoiden TRAKLA2-opetusohjelman käyttöliittymässä suorittamista operaatioista päätellä jotain opittavan asian luonteesta? Soveltuuko tässä työssä kehitetty datan keruumenetelmä oppimisen tutkimiseen?

Tätä pyritään selvittämään analysoimalla TKK:n kevään 2003 kahden Tietorakenteet ja algoritmit (TRAK) -kurssin aikana kerättyä dataa. TRAKLA2-harjoitustehtävät olivat osa kurssien arvosteltavia osasuorituksia. Tietotekniikan koulutusohjelman opiskelijoille suunnattua T-kurssia suoritti n. 230 opiskelijaa ja muiden koulutusohjelmien opiskelijoille suunnattua Y-kurssia n. 390 opiskelijaa. T-kurssilla oli käytössä 8 TRAKLA2-harjoitustehtävää, Y-kurssilla 6. Kurseilla ei ollut käytössä samoja tehtäviä, joten kaikkiaan keväällä 2003 käytössä oli 14 TRAKLA2-harjoitustehtävää.

Analyysi tehdään kahdesta näkökulmasta: ensinnäkin oletetaan, että harjoitustehtävien vaikeusaste vaihtelee ja yritetään selvittää tehtävien keskinäinen vaikeus niiden käytöstä kerätyn datan perusteella. Toiseksi, tutkitaan opiskelijoiden toimintaa tiettyssä harjoitustehtävässä ja yritetään selvittää tehtävässä käsiteltävän algoritmin osien keskinäinen vaikeus.

6.1.1 Algoritmien vertailu

Eroja eri algoritmien keskinäisessä vaikeudessa voidaan tutkia vertailemalla mitä operaatioita ja kuinka usein opiskelijat ovat harjoitustehtävissä tehneet. Esimerkiksi

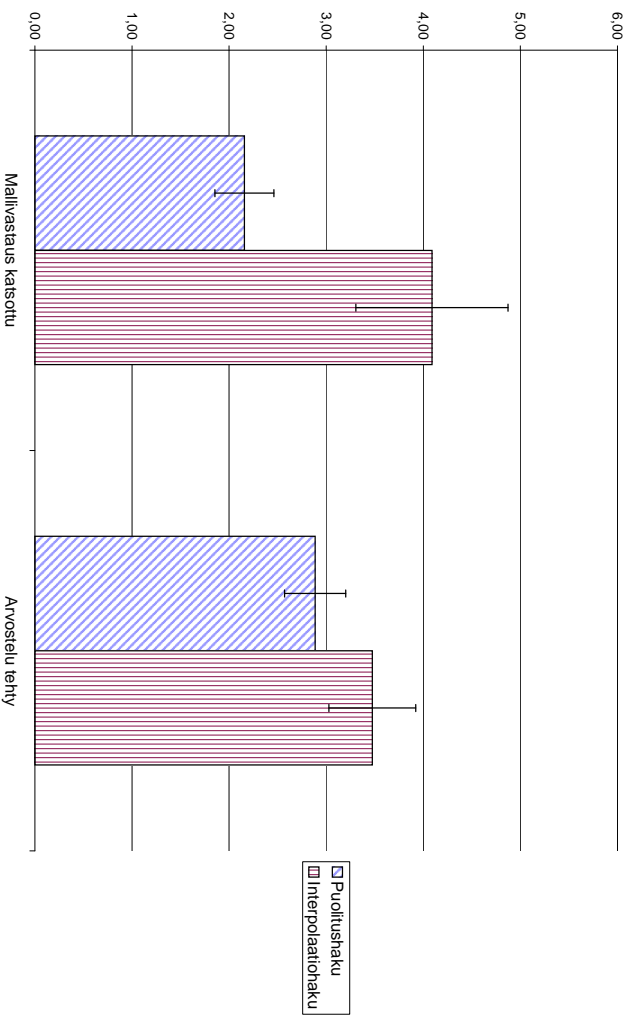


KUVA 6.1: Puolitushakutehtävässä vedetään taulukon avaimia alla olevaan listaan siinä järjestyksessä, kun puolitushakualgoritmi vertailee alkioita.

voidaan olettaa, että interpolaatiohaku on puolitushakua vaikeammin ymmärrettävä algoritmi. Tälle oletukselle voidaan hakea vahvistusta tutkimalla opiskelijoiden tekemiä operaatioita vastaavissa harjoitustehtävissä.

Harjoitustehtävät olivat TRAKLA2-oppimisympäristössä samalla kotitehtäväkierrokselle peräkkäiset tehtävät. Puolitushakutehtävä (kuva 6.1) oli esitetty ennen interpolaatiohakutehtävää, mutta järjestelmä ei asettanut rajoituksia tehtävien ratkaisujärjestykselle. Molemmat harjoitustehtävät aloitti TRAK-T -kurssilla 211 opiskelijaa. Valitaan otokseksi tehtävät hyväksyneesti palauttaneet opiskelijat. Puolitushakutehtävän palautti onnistuneesti 203 opiskelijaa ja interpolaatiohakutehtävän 179 opiskelijaa. Oheisessa kuvassa (Kuva 6.2) on esitetty kuinka usein tehtävissä on keskimäärin katsottu mallivastausta tai pyydetty arvostelua. Arvostelun jälkeen opiskelijalla on mahdollisuus tehdä tehtävä uudestaan eri alkuarvoilla. Nähdään, että interpolaatiohakutehtävässä mallivastausta on katsottu sekä tehtävän arvostelua kokeiltu useammin kuin puolitushakutehtävässä. Koska tehtävien käyttöliittymä on identtinen, niin hyvin todennäköinen syy eroille on tehtävien vaikeusaste. Suurien 95%:n luotettavuusvälien takia tämä tutkimustulos ei kuitenkaan ole tilastollisesti merkittävä.

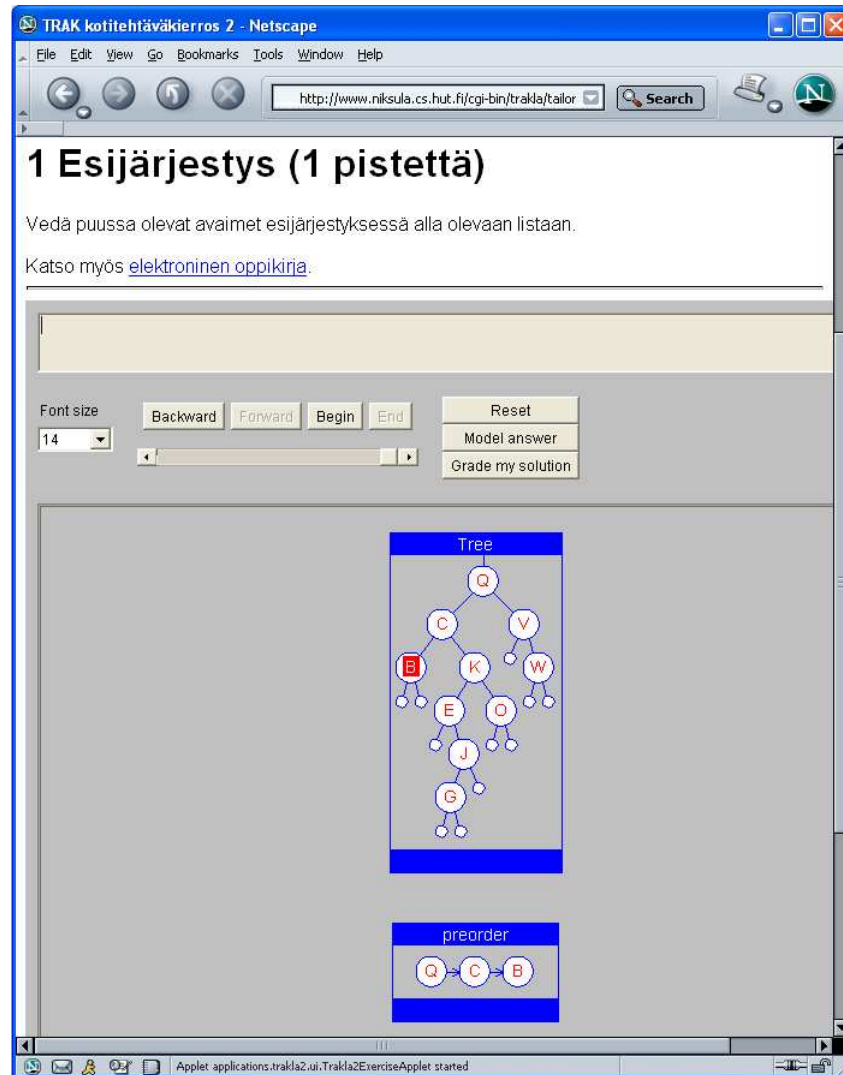
Tehtävien vaikeutta voidaan tutkia myös siten, että oletetaan vaikeampien harjoitustehtävien tekemisessä olevan enemmän taukoja kuin helppojen tehtävien. Jos harjoitustehtävän käyttöliittymässä ei tehdä mitään minuuttiin, niin tästä tallennetaan aikaleima tietokantaan. Käyttöliittymäoperaatioita ovat painonappien painaminen, algoritmianimaation kelaaminen ja tietorakenteille tehtävät drag&drop -operaatiot.



KUVA 6.2: Puolitusshaku- ja interpolatiohakutehtävien operaatiot keskimäärin opiskelijaa kohden sekä sekä 95%:n luotettavuusväliit.

Hiiren liikuttaminen ei tässä yhteydessä ole käytöllitittymäoperaatio.

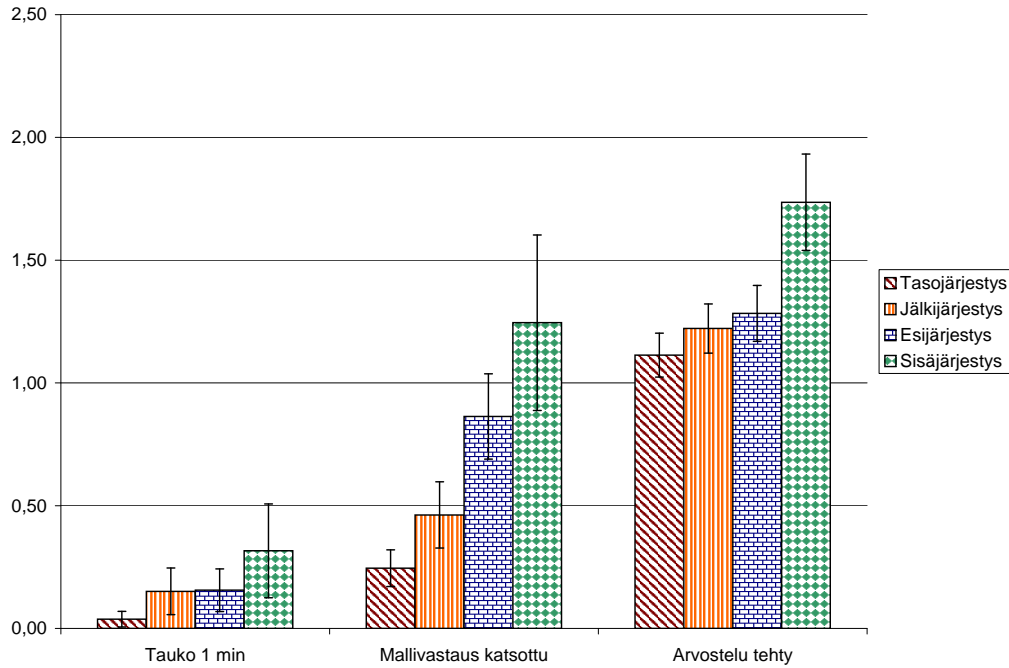
TRAK-T -kurssilla oli koitettavia binääripuun läpikäyntialgoritmien harjoittelun. TRAKLA2-oppimisympäristössä tehtävät olivat seuraavassa järjestyksessä: esijärjestys, sisäjäjestys, jälkijärjestys ja tasojärjestys. Tehtävien käyttöliittymä oli samanlainen; tehtävänä käydä binääripuun alkiot läpi oikeassa järjestyksessä. Esijärjestystehtävä on kuvassa 6.3. Järjestelmä ei asettanut rajoituksia tehtävien tekojärjestykselle, vaikkakin on luontevinta tehdä tehtävät siinä järjestyksessä, kun ne oppimisympäristössä on esitetyt. Tehtävän esijärjestysläpikäyntialgoritmista aloitti 216 opiskelijaa. Sisäjäjestystä, jälkijärjestystä ja tasojärjestystä koskevat tehtävät aloitti 213 opiskelijaa. Hyväksytyksi kaikki tehtävät sai palautettua 212 opiskelijaa. Kuvassa 6.4 on puun läpikäyntitehtävien operaatioiden määrä opiskelijaa kohden. Nähdään, että tehtävät voidaan laittaa samaan järjestykseen yli minuutin pituisten taukojen, mallivastauksen katsomisten tai tehtävän arvostelupyyntöjen mukaan. Läpikäyntitehtävien käyttöliittymä on sama, joten tässäkin erot operaatioiden määrässä voivat selittyä eroilla tehtävien sisällöissä eli opittavien algoritmien ymmärrettävyydellä. Voidaan kuitenkin olettaa, että läpikäyntialgoritmitehtävyyppin ensimmäistä tehtävää, joka tässä on ollut esijärjestystehtävä, on tehtävätyyppin totuttautumisen vuoksi harjoiteltu enemmän kuin muita tehtäviä, jolloin esijärjestystehtävän operaatioiden lukumäärät eivät olisi vertailukelpoisia muiden tehtävien kanssa. Tästä huolimatta tilastosta on selvästi nähtävissä johtopäätös, että sisäjäjestys on vaikeampi vaikeimmalta algoritmilta koitettävässä suoritetujen operaatioiden lukumäärän perusteella. Kun huomioi 95%:n luotettavuusväliit, niin tulos on tilastollisesti merkittävä verrattaessa sisäjäjestystehtävää tasojärjestystehtävään taukojen, mallivastauksen katsomiskertojen sekä arvostelukertojen perusteella. Tilastollisesti merkittävä on myös johtopäätös, että sisäjäjestystehtävä on vaikeampi kuin jälki-



KUVA 6.3: Esijärjestystehtävässä vedetään annetun binääripuun avaimet alla olevaan listaan esijärjestyksessä.

järjestystehtävä mallivastausten katsomiskertojen ja arvostelukertojen lukumäärän perusteella.

Samansuuntaisia tuloksia saadaan myös, kun verrataan binääristä hakupuuta ja punamustaa hakupuuta käsitteleviä tehtäviä. Kurssilla TRAK-Y tehtävän, jossa lisätään alkioita binääriseen hakupuuhun (kuva 6.5), aloitti 318 opiskelijaa ja sai hyväksytysti palautettua 314 opiskelijaa. Samalla kurssilla tehtävän, jossa lisätään sama lukumäärä avaimia punamustaan puuhun, aloitti yhteensä 315 opiskelijaa. Tehtävän sai hyväksytysti palautettua 293 opiskelijaa. Kuvassa 6.6 on tehtävien operaatioiden keskimääräiset lukumäärät opiskelijaa kohden. Nähdään, että tehtävät voidaan laittaa samaan järjestykseen yli minuutin pituisten taukojen, mallivastauksen katsomisten tai tehtävän arvostelupyynnöiden mukaan. Tämä tulos vahvistaa sitä intuitiivista näkemystä, että punamustaan puuhun liittyvät algoritmit ovat monimutkaisempia ja vaikeammin ymmärrettäviä kuin binääriseen hakupuuhun liittyvät algoritmit. Luultavasti punamustan puun tehtävässä opiskelijat käyttivät enemmän aikaa operaati-



KUVA 6.4: Puun läpikäyntitehtävien operaatiot keskimäärin opiskelijaa kohden sekä 95%:n luotettavuusvälit.

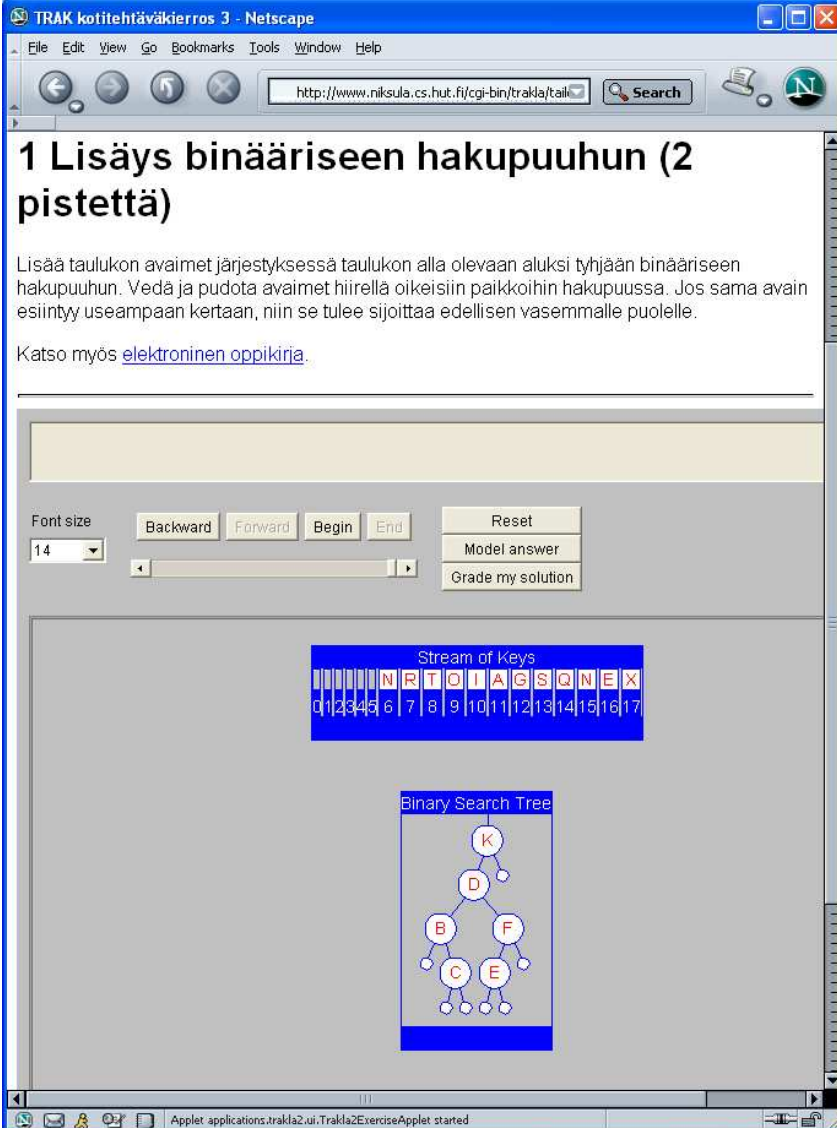
tioiden miettimiseen tai oheismateriaaliin tutustumiseen kuin binäärisen hakupuun tehtävässä. Osittain erot voivat myös selittyä tehtävien käyttöliittymän eroavaisuuksilla; vaikka binääripuu visualisoidaan molemmissa tehtävissä samalla tavalla, niin punamustan puun tehtävässä on painonapit rotaatioiden suorittamista sekä solmun värin muuttamista varten.

6.1.2 Algoritmin osien keskinäisen vaikeuden analyysi

Opettajalle voi olla hyödyllistä tietää, mitkä asiat tietyissä algoritmeissa ovat vaikeita ymmärtää. Tätä voidaan tutkia esimerkiksi siten, että katsotaan kuinka monta kertaa opiskelijat ovat suorittaneet algoritmin askeleita vastaavat tilasiirtymät mallivastauksen algoritmianimaatiossa. Voidaan olettaa, että hankaliksi koettujen askeleiden vastaavia tilasiirtymiä on tutkittu huomattavasti enemmän kuin helpoiksi koettujen.

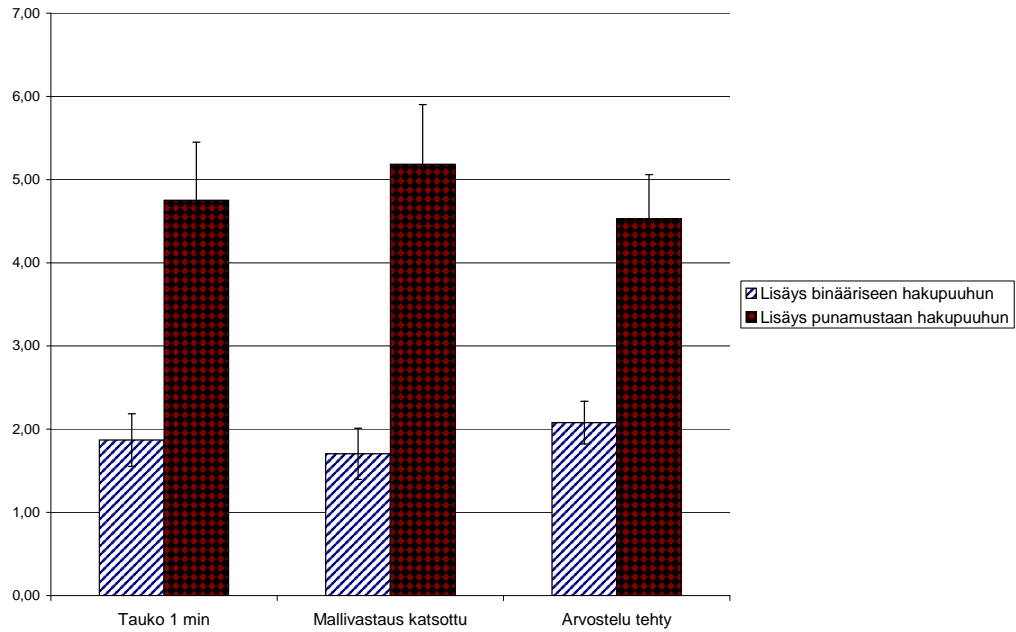
Esimerkkitapauksessa tutkitaan tehtävää, jossa lisätään annetut alkio yksitellen 15-paikkaiseen kekkoon ja lopuksi poistetaan keosta kolme alkioita. Kekoehto "isä pienempi kuin lapsi" palautetaan voimaan jokaisen operaation jälkeen. Tehtävä on esitetty kuvassa 3.2. Oletetaan, että algoritmin askel, jossa tehdään alkion poistaminen, on hankalammin ymmärrettävä kuin algoritmin askel, jossa tehdään annetun alkion lisääminen kekkoon.

Kyseinen tehtävä oli TRAK-T -kurssin kotitehtävänä. Tehtävän tekemisen oli aloittanut 213 opiskelijaa, joista 203 sai sen hyväksytysti suoritettua. 151 opiskelijaa katsoi mallivastausta keskimäärin 4,16 kertaa. Yhteensä mallivastaus avattiin 629 kertaa.



The screenshot shows a Netscape browser window with the title "TRAK kotitehtäväkierrös 3 - Netscape". The address bar contains the URL "http://www.niksula.cs.hut.fi/cgi-bin/trakla/taik". The main content area displays the heading "1 Lisäys binääriseen hakupuuhun (2 pistettä)". Below the heading is a paragraph of text: "Lisää taulukon avaimet järjestyksessä taulukon alla olevaan aluksi tyhjään binääriseen hakupuuhun. Veda ja pudota avaimet hiirellä oikeisiin paikkoihin hakupuussa. Jos sama avain esiintyy useampaan kertaan, niin se tulee sijoittaa edellisen vasemmalle puolelle." Below this text is a link: "Katso myös [elektroninen oppikirja](#)." The interface includes a font size dropdown set to 14, and buttons for "Backward", "Forward", "Begin", "End", "Reset", "Model answer", and "Grade my solution". A "Stream of Keys" section shows a sequence of keys: N, R, T, O, I, A, G, S, Q, N, E, X, with indices 0 through 17 below them. A "Binary Search Tree" diagram shows a root node 'K' with a left child 'D' and a right child 'F'. Node 'D' has a left child 'B' and a right child 'F'. Node 'B' has a left child 'C' and a right child 'E'. Node 'F' has a left child 'E' and a right child 'X'. The tree is partially filled with nodes, and the diagram is highlighted with a blue border.

KUVA 6.5: Lisäys binääriseen hakupuuhun: tehtävässä tyhjään binääriseen hakupuuhun lisätään 17 satunnaista avainta vetämällä ja pudottamalla avaimet oikeisiin kohtiin binääristä hakupuuta.



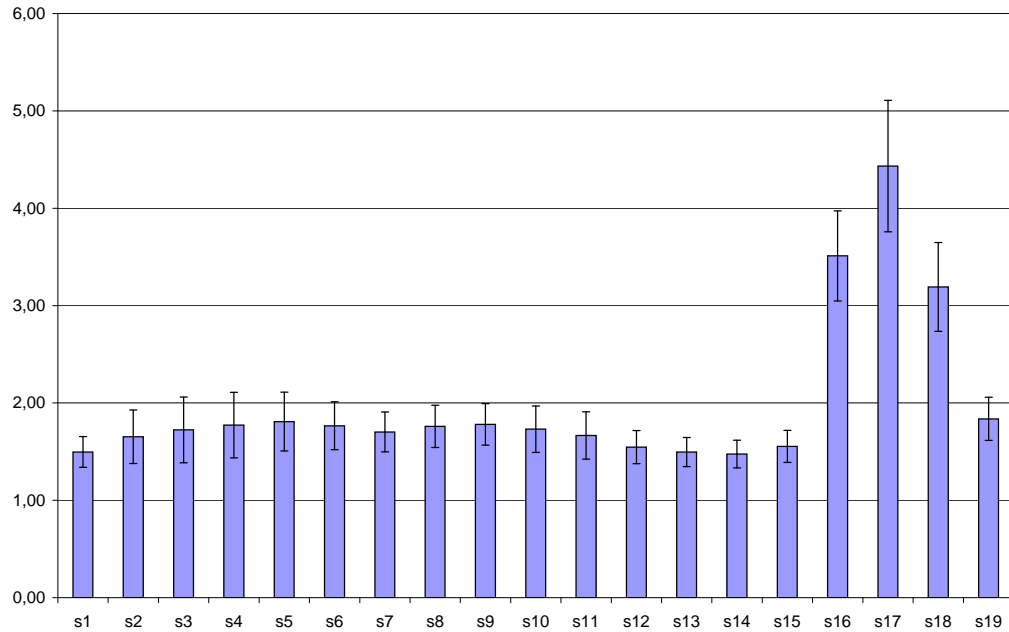
KUVA 6.6: Lisäys binääriseen hakupuuhun ja punamustaan hakupuuhun: operaatiot keskimäärin opiskelijaa kohden sekä 95%:n luotettavuusvälit.

Näistä kerroista algoritmianimaatiossa liikuttiin 376 kerralla. Animaatiossa liikkuminen voidaan aloittaa lopusta valitsemalla Backward, mikä saattaa lisätä viimeisten tilojen vierailulukumääriä. Tämän vuoksi päätettiin tutkia vain niitä mallivastauksen katsomiskertoja, joissa animaatio on käyty alusta loppuun (141 kpl).

Mallivastauksen algoritmianimaatiossa oli alkuarvoista riippumatta aina 18 askelta, jolloin näitä vastaavia tiloja on 19. Alkioiden poistoja vastaavat algoritmin askeleet 16, 17 ja 18. Näitä vastaavat tilat ovat siis 16, 17, 18 ja 19, missä 19 on lopputila. Tietokantaan on tallennettu tiedot siitä, kuinka monta kertaa missäkin tilassa on vierailtu. Tilasiirtymistä tai niiden järjestyksestä ei ole tallennettu tietoja.

Oheisessa kaaviossa (Kuva 6.7) näkyy algoritmianimaation tiloja vastaavat vierailulukumäärät, kuinka monta kertaa missäkin tilassa on keskimäärin käyty, sekä 95%:n luotettavuusvälit. Kaaviosta nähdään, että tiloissa 16, 17 ja 18 on käyty huomattavasti useammin kuin muissa tiloissa. Tästä voidaan päätellä, että opiskelijat ovat olleet kiinnostuneempia niitä tiloja vastaavista tilasiirtymistä, joissa on tehty keon alkion poistamiset. Koska tallennettuna on vain vierailujen lukumäärä joka tilassa, niin kaikkia mahdollisia algoritmianimaation polkuja ei saada selville.

Vertailun vuoksi tutkittiin toista algoritmiharjoitustehtävää, jonka askeleet voidaan olettaa keskenään yhtä vaikeiksi. Esimerkki tämän kaltaisesta harjoitustehtävästä on avainten lisäys binääriseen hakupuuhun. Vastaava analyysi osoitti, että mallivastauksen algoritmianimaation katselu oli painottunut tasaisesti kaikkien tilojen ympärille. Vertailu osoittautui oleelliseksi, koska löydettiin mielenkiintoinen ero animaation katsomisessa.



KUVA 6.7: Keskimääräinen vierailujen lukumäärä algoritmianimaation tiloissa sekä 95%:n luotettavuusvälit.

6.1.3 Johtopäätökset

Edellisissä kappaleissa kuvatut tulokset puolitushaun ja interpolaatiohaun keskinäisen vaikeuden tutkimisesta, binäärisen hakupuun ja puna-mustan hakupuun lisäysalgoritmin sekä binääripuun läpikäyntialgoritmien keskinäisen vaikeuden tutkimisesta operaatiolokin tilastollisella analyysillä antavat olettaa, että kehitetty datan keruu- ja analysointimenetelmällä voidaan tuottaa perusteltua analyysiä tutkittavien algoritmien keskinäisestä vaikeudesta.

Tässä työssä kehitetty menetelmä tukee sitä käsitystä opiskelijoiden oppimisesta, joka opettajille syntyy luentojen ja laskuharjoitusten pitämisen yhteydessä. Tutkimusmateriaalia tilastolliseen tutkimukseen asiasta kerääntyy myös tentti- ja laskuharjoitusvastauksien yhteydessä.

Kehitetty analysointimenetelmä voi soveltua opettajien apuvälineeksi kurssin edistymisen seurannassa. Menetelmä soveltuu opettajien apuvälineeksi, jolla he voivat seurata kurssin painotusten tai opetusmenetelmien muutosten vaikutusta opiskelijoiden osaamiseen harjoitustehtävissä. Tämä edellyttää kuitenkin kurssin pysymistä muuttumattomana usean vuoden ajan, jotta saadaan kerättyä vertailukelpoista dataa eri vuosilta.

Ovatko algoritmien oppimisesta yleistettävissä muuallekin kuin oppimisen tutkimiseen TRAKLA2-harjoitustehtävissä? Voidaan ajatella, että ainakin analyysin binääripuun läpikäyntialgoritmeja koskevien harjoitustehtävien keskinäisestä vaikeudesta on oltava yleistettävissä läpikäyntialgoritmien osaamiseen kurssin opiskelijoiden keskuudessa, koska kyseiset harjoitustehtäväsovelmat ovat käyttöliittymältään hyvin yksinkertaisia. Toisaalta löydettyt osaamisen erot tehtävissä voivat selittyä sillä,

miten algoritmit on kyseisen kurssin luennoilla selitetty, jolloin ne eivät ole yleis-tettävissä kyseisten algoritmien oppimiseen yleisellä tasolla. Sen selittäminen, miten oppijan kognitiiviset prosessit kutakin algoritmia opittaessa toimivat, saattaa olla tässä työssä kehitetyn tutkimusmenetelmän ulottumattomissa.

6.2 Opetusteknologian tutkimuksen työkalu

Kun tarkastellaan datan keruumenetelmän soveltuvuutta TRAKLA2-opetusohjelmassa käytettyjen opetusteknologioiden evaluointiin, keskeinen kysymys oli, että voiko kerätyn datan pohjalta tehdä tutkimusasetelmia opetusteknologioiden tutkimiseen. Kun tutkitaan opetusteknologiaa, niin tavoitteena on osoittaa tietyn teknologian vaikutus oppimiseen – edistääkö teknologian käyttäminen oppimista vai ei, ja millä tavalla se edistää oppimista.

Tutkimuksessa kehitetyn datan keruumenetelmän soveltuvuutta opetusteknologian sovellusten analysointityökaluksi voidaan arvioida tutkimalla kerättävää dataa opetusteknologian tutkimukseen soveltuvien tutkimusasetelmien valossa. Yleensä näissä tutkimuksissa opiskelijat jaetaan ryhmiin, joiden käyttämistä opetusohjelman versiosta tutkittava toiminto on toteutettu hieman eri lailla. Tutkittavan toiminnon vaikutusta oppimistulokseen voidaan tutkia sillä, miten opiskelijat järjestelmää käyttävät – esimerkiksi vaikuttaako käytetty kognitiivinen työkalu tai menetelmä opetusohjelman harjoitustehtävissä tehtyjen virheiden määrään tai laatuun.

Oletetaan, että tutkitaan esimerkiksi erilaisten algoritmivisualisaatiotekniikoiden vaikutusta oppimistulokseen TRAKLA2-opetusohjelmassa. Tällöin voitaisiin tutkia, miten eri algoritmivisualisaatiotekniikat vaikuttavat seuraaviin oppimistulosta mahdollisesti kuvaaviin muuttujiin:

- mallivastauksen katsomisaika
- mallivastauksen katsomislukumäärät; kuinka monta kertaa mallivastaus avataan.
- tehtävän arvostelujen lukumäärä; kuinka monta kertaa tehtävä arvostellaan ennen kuin se saadaan palautettua hyväksytysti.
- virheiden määrä ja laatu arvostelun yhteydessä; kuinka paljon tietynlaisia virheitä opiskelijat tekevät.
- tehtävän tekoaika
- tehtävää tehdessä pidettyjen miettimistaukojen lukumäärä

Edellä kuvattu tutkimusasetelma edellyttää useiden versioiden ohjelmoimista opetusohjelmasta; versioiden joissa tutkittava toiminto on toteutettu hieman eri lailla. Koeasetelma voidaan järjestää siten, että kurssilla kukin opetusohjelman versio annetaan satunnaisesti muodostetun opiskelijaryhmän käyttöön. Tiedot opiskelijoiden

toiminnasta opetusohjelmassa tallennetaan ja muuttujien yhteyksiä tutkitaan tilastollisella analyysillä.

Vastaavia tutkimusasetelmia, joissa oppimistuloksia kuvaavat muuttujat mitataan suoraan opetusohjelman käytöstä kerätystä datasta, löytyy myös kirjallisuudesta. Corbett et al [10] suorittamassa tutkimuksesta vertailtiin Lisp-ohjelmoinnin opetusohjelman neljää eri versiota, jotka erosivat toisistaan ohjelman antaman palautteen yksityiskohtaisuuden ja välittömyyden perusteella. Siinä oppimistuloksia mitattiin paitsi kynällä ja paperilla tehtävien testien avulla, niin myös opetusohjelman käyttötavan mukaan. Opetusohjelmasta kerätystä datasta oppimistulosta kuvaavia muuttujia olivat tehtävien tekoaika sekä virheiden todennäköisyys. Tehtävien tekoajalla tarkoitettiin sitä aikaa, jonka tehtävän ratkaiseminen hyväksytysti kesti.

Toinen esimerkki tutkimusasetelmasta, jossa oppimistuloksia mitattiin suoraan opetusohjelman käytöstä kerätyllä datalla, on Mitrovic et al [37] suorittama tutkimus SQL-Tutor -opetusohjelmasta. Tutkimuksessa arvioitiin opetusohjelman antaman automaattisen palautteen yksityiskohtaisuuden yhteyttä oppimistulokseen. Opiskelijoiden käyttöön annettiin kaksi versiota opetusohjelmasta, joista ensimmäinen antoi palautteena vain vihjeitä opiskelijoiden tekemistä virheistä. Toinen versio tarjosi palautteena täydellisen kuvauksen opiskelijan tekemästä virheestä. Oppimistuloksia kuvaavat muuttujat olivat tehtävien hyväksytyyn suoritukseen vaadittujen yritysten lukumäärä sekä tehtävien tekemiseen käytetty aika. Tutkimusasetelma ja tutkimuksen tärkeimmät tulokset on kuvattu tarkemmin kappaleessa 2.6.

Yhteenvedona voidaan todeta, että kehitetyn datan keruu- ja analysointimenetelmä mahdollistaa sellaisten muuttujien analysoinnin, joiden perusteella voidaan analysoida opetusohjelman tietyn toiminnon vaikutuksia opiskelijoiden oppimiseen. Myös luvussa 6.1.1 esitetyt analyysit antavat olettaa, että esimerkiksi käyttöliittymäoperaatioiden lukumäärällä on yhteyttä opiskelijan osaamiseen tehtävässä. Tämän kaltaiset tutkimusasetelmat ovat ensiarvoisen tärkeitä TRAKLA2-opetusohjelman kehitysprosessin kannalta. Toisaalta voidaan ajatella, että TRAKLA2-opetusohjelman käyttöä tutkimalla voidaan saada uutta tietoa, joka voidaan soveltaa myös muiden opetusohjelmien kehitystyössä tai opetusteknologian teoreettisessa tutkimuksessa. Esimerkiksi juuri edellä kuvatut evaluaatiotutkimukset automaattisen palautteen välittömyyden tai yksityiskohtaisuuden vaikutuksesta oppimistuloksiin voidaan ajatella pätevän opetusohjelmissa yleensä, ei vain niissä opetusohjelmissa, joiden käyttöä tutkimalla tulokset saatiin.

6.3 Teknisen toteutuksen analyysi

Menetelmän teknisen toteutuksen täytyy toimia niin hyvin, että datan keruu ei häiritse opiskelijan toimintaa harjoitustehtäväsovelmassa. Ohjelmistoteknisestä näkökulmasta ajatellen, toteutetun menetelmän täytyy olla muunneltavissa tutkittavan TRAKLA2-opetusohjelman mahdollisten muutosten myötä. Tässä kappaleessa kehitetyn datan keruumenetelmän teknistä toteutusta arvioidaan menetelmän tehok-

kuuden ja muunneltavuuden kannalta.

6.3.1 Tehokkuus

Toteutetun datan keruumenetelmän tehokkuuden arvioinnissa kiinnitettiin huomiota siihen, kuinka paljon datan keruu aiheutti viiveitä harjoitustehtäväsovelman käyttämisessä. Datan lähettäminen palvelimelle tapahtuu samassa säikeessä, jossa sovelma suorittaa käyttöliittymäoperaatiot ja esimerkiksi ruudun päivittämisen. Tyypillinen sekvenssi käyttöliittymäoperaation, kuten esimerkiksi arvostelun suorittamisen grade-painiketta painamalla, on seuraava:

1. Käyttäjä painaa sovelman Grade-painiketta.
2. Sovelma tarkastaa vastauksen ja generoi palautteen.
3. Sovelma lähettää lokitietona vastauksen palvelimelle.
4. Sovelma luo palauteikkunan ja näyttää sen käyttäjälle(kuva 3.3).

Datan keruun aiheuttama viive on 3. askeleen aiheuttama aika. Tämän aikaviivettä suhteessa muiden operaatioiden aikaan mitattiin JProfiler-ohjelmalla [2], joka on ohjelma Java-ohjelmien tehokkuusanalyysiin. JProfilerilla voi mitata kuinka paljon aikaa Java-ohjelma käyttää metodikutsuissa.

Tutkitaan kahta TRAKLA2-sovelman operaatiota, joiden aikana sovelma lähettää lokitietoja palvelimelle: tehtävän arvostelua, joka aloitetaan Grade-painikkeesta sekä tehtävän alustusta Reset-painikkeesta. Operaatioiden palvelimelle lähettämät lokitiedot on kuvattu tarkemmin kappaleessa 5.2. Arvostelun yhteydessä palvelimelle lähetetään opiskelijan vastaussekvenssi, joka on monimutkainen oliorakenne. Alustuksen yhteydessä palvelimelle lähetetään vain aikaleima. Arvostelu-operaatiosta tallennettava oliorakenteen koko ja monimutkaisuus vaihtelee tehtäväkohtaisesti. Lähetettävän tiedon koko vaikuttaa siihen, kuinka nopeasti tietoliikenneyhteys välittää tiedon sovelmasta palvelimelle. Oliorakenteen monimutkaisuus vaikuttaa RMI-mekanismiin tapaan käsitellä rakennetta asiakkaan ja palvelimen päässä ja siten tiedonsiirron tehokkuuteen.

Mittauksissa tutkittiin, kuinka kauan operaatio kokonaisuudessaan kestää ja mikä on datan keruun osuus operaation kokonaiskestosta. Tehtäväkohtaisia eroja pyrittiin ottamaan huomioon tutkimalla kahta eri tehtävää: esijärjestys- ja poisto binäärisestä hakupuusta. Mittauksissa TRAKLA2-sovelmaa ajettiin PC-tietokoneella, jossa oli 1400 Mhz Intel Pentium -suoritin, 768 Mt muistia ja Windows XP -käyttöjärjestelmä. Sovelmaa ajettiin Netscape- ja Internet Explorer -selaimilla, jotka käyttivät Java-sovelmien suorittamiseen Sun Java(TM) Plug-in 1.4.1_02 virtuaalikonetta. Mittaus suoritettiin kotiympäristössä, jossa tietoliikenneyhteytenä oli ADSL-yhteys, jonka nopeus oli 1 Mt sisään- ja 512 kbit/s ulospäin.

Käytetty testauskoonpano vastaa melko hyvin TKK:n opiskelijoiden käyttämiä työasemia. Keväällä 2003 järjestetyn kyselytutkimuksen mukaan TKK:n kevään 2003

Tietorakenteet ja algoritmit kurseilla Windows 2000 / XP oli käytetyin käyttöjärjestelmä (53 % vastaajista käytti ko. käyttöjärjestelmää). Windows-ympäristössä selaimena voi käyttää joko Microsoft Internet Explorer- tai Netscape 7.x -selainta. Monet käyttivät TRAKLA2-sovelmia myös Linux / Unix -ympäristöissä.

Suurin osa opiskelijoista käytti TRAKLA2-opetusohjelmaa kotona (69 %). Opiskelijoiden kotona olevista tietoliikenneyhteyksistä ei ole tietoa, mutta voidaan olettaa että ADSL- tai kaapelimodeemiteknoologiaan perustuvat laajakaistayhteydet ovat hyvin yleisiä opiskelijoiden keskuudessa. Myös monissa TKK:n opiskelijoiden opiskelija-asuntoloissa on laajakaistayhteydet.

TAULUKKO 6.1: Keskiarvot ja luottamusvälit arvostelu- (grade) ja alustus (reset) -operaatioiden aikaviiveistä TRAKLA2-sovelmassa; mittauksissa tutkittiin kuinka kauan operaatio kokonaisuudessaan kestää ja mikä on datan keruun osuus operaation kokonaiskestosta. Operaatiot suoritettiin 10 kertaa ja keskiarvot sekä luottamusvälit laskettiin. Mittaukset suoritettiin PC-tietokoneella, jossa oli 1400 Mhz Intel Pentium -suoritin, 768 Mt muistia ja Windows XP -käyttöjärjestelmä. Sovelmaa ajettiin Netscape (NS)- ja Internet Explorer (IE) -selaimilla, jotka käyttivät Java-sovelmien suorittamiseen Sun Java(TM) Plug-in 1.4.1_02 virtuaalikonetta. Mittaus suoritettiin kotiympäristössä, jossa tietoliikenneyhteytenä oli ADSL-yhteys, jonka nopeus oli 1 Mt sisään- ja 512 kbit/s ulospäin.

Tehtävä	Operaatio	Selain	Kokonaiskesto (ms)	Lokituksen osuus (ms)	%
esijärjestys	arvostelu	NS	487 ($\pm 2,06$)	315 ($\pm 1,86$)	65 %
esijärjestys	arvostelu	IE	310 ($\pm 1,35$)	212 ($\pm 1,00$)	68 %
poisto binäärisestä hakupuusta	arvostelu	IE	239 ($\pm 1,24$)	141 ($\pm 0,36$)	59 %
esijärjestys	alustus	IE	453 ($\pm 0,50$)	36,8 ($\pm 0,17$)	8 %

Mittaustulokset on esitetty taulukossa 6.1. Mittaustuloksista käy ilmi, että lokitus-tiedon lähettäminen palvelimelle vie suurimman osan arvostelu-operaation ajasta. Vastauksen tarkastaminen, ruudun piirtäminen ja muut ohjelman toiminnot vievät vähemmän aikaa. Vasteajat ovat kuitenkin niin pieniä, että lokituksen poistamisella säästetty 100–300 millisekuntia ei välttämättä paranna sovelman käytettävyyttä. On lähes yhdentekevää kestääkö palauteikkunan avautuminen 200 vai 500 millisekuntia, koska ikkunoiden avautumista on yleensä totuttu odottamaan muutama sekunti joka tapauksessa.

Operaatiot, joissa palvelimelle lokitetaan vain aikaleima, eivät juurikaan lisää käyttöliittymän viiveitä. Tästä on osoituksena alustus-operaation lokituksen vaatima alle 40 millisekunnin aika.

Suoritetuilla mittauksilla saatiin selville, kuinka paljon aikaa sovelma käyttää lokitietojen lähettämiseen palvelimelle, kun palvelin ei ole useiden yhtäaikaisten käyttäjien kuormittama. Mittausjärjestely ei ota huomioon sitä, miten palvelin käyttäytyy suuren kuorman alla, kun sadat opiskelijat tekevät tehtäviä samanaikaisesti. Aika-

viiveiden tutkiminen kuormitustilanteessa vaatii monimutkaisemman koejärjestelyn ja on tutkimisen arvoinen näkökulma datan keruumenetelmän toteutuksessa.

Nykyinen datan keruun toteutus asettaa sovelman tapahtumakäsittelijäsäikeen (event handler thread) odottamaan lokitustiedon lähettämisen ajaksi. Tämä keskeyttää ruudun piirtämisen ja käyttöliittymäoperaatioiden käsittelyn sovelmassa. Hienos-tuneempi tapa ohjelmoida datan keruu olisi antaa lokitietojen lähettäminen erillisen säikeen tehtäväksi, jolloin tapahtumankäsittelijäsäie vapautuisi piirtämään ruutua ja käsittelemään käyttäjän operaatioita samanaikaisesti, kun lokitussäie lähettäisi lokitietoja palvelimelle. Tämä rinnakkain toimiviin säikeisiin perustuva ratkaisu vähentäisi datan lokituksen aiheuttamat viiveet murto-osaan nykyisestä.

6.3.2 Muunneltavuus

Kaikkien lokitettavien käyttöliittymäoperaatioiden yläluokka on Operation (kuva 5.1). Mikäli johonkin lokitettavaan operaatioon halutaan lisätä uusi attribuutti, voi attribuutin lisätä Java-kenttänä kyseiseen Operation-luokan aliluokkaan. Luokkaan täytyy myös määritellä, miten sen attribuutit muutetaan tekstiksi. Tekstimuutos määritellään toString-metodissa.

Palvelin tallentaa Operation-ilmentymät myös sarjallistettuina Java-olioina. Jos lokituksen yhteydessä haluaa tallentaa oliorakenteita, niin Operation-ilmentymään on lisättävä viittaus oliorakenteeseen ja varmistuttava, että tallennettava oliorakenne on sarjallistuvassa muodossa. Java-luokka on sarjallistuva, kun se täyttää *java.io.Serializable* -rajapinnan [44].

TRAKLA2-ohjelmiston kehitystyön myötä käyttöliittymään saattaa tulla uusia toimintoja. Mikäli jonkin uuden toiminnon käyttämisestä halutaan lokittaa tietoa, niin on TRAKLA2-sovelluskehikseen on lisättävä toimintoa vastaava Operation-luokan aliluokka ja käytettävä kyseisen luokan ilmentymiä lokitustiedon välittämiseen sovelmasta palvelimelle.

Miten kerätyn datan vertailukelpoisuus säilytetään eri vuosien ja TRAKLA2-versioiden välillä? Kaikista lokitettavista operaatioista tallennetaan merkintä tekstitiedostoon. Lokitiedoston rivi koostuu puolipisteillä erotetuista nimi-arvo -pareista, kuten on kuvattu kappaleessa 5.3. Analyysiä tehtäessä tarvittavat attribuutit voi suodattaa tästä tekstitiedostosta attribuuttien nimien perusteella tavanomaisia Unix-työkaluja, kuten grep:iä ja awk:ia käyttäen.

Sarjallistettujen Operation-luokan ilmentymien analyysissä yhteensopivuusongelmia on ratkaistu kahdesta näkökulmasta. Ensinnäkin, jos sarjallistettujen oliorakenteiden luokkien nimet ja perintähierarakiat eivät ole muuttuneet eikä luokista ole poistettu kenttiä tai niiden nimiä tai tyyppejä ole muutettu, niin ilmentymien lukeminen levyiltä pitäisi onnistua viimeisimmällä TRAKLA2- ja Matrix-sovelluskehiksen versiolla. Sarjallistettujen oliorakenteiden levyiltä lukemisen yhteensopivuustarkastelua on tehty kappaleessa 5.3.

Toiseksi, jos Matrix- tai TRAKLA2-sovelluskehysten luokat ovat muuttuneet siten, että luokan vanhalla versiolla tallennetun ilmentymän lukeminen luokan uuden version ilmentymäksi ei onnistu, niin ilmentymät voidaan aina lukea sillä versiolla Matrix- ja TRAKLA2-sovelluskehysistä, jolla ne on kirjoitettu levyille. TRAKLA2-opetusohjelman versiopäivityksen yhteydessä vanhat versiot luokkakirjastoista tallennetaan automaattisesti erilliseen hakemistoon.

Luku 7

Yhteenveto

Tässä tutkielmassa luotiin katsaus ohjelmoinnin opetusohjelmissä käytettyihin opetusteknologioihin sekä ohjelmoinnin opetusohjelmista tehtyyn evaluaatiotutkimukseen ja sen menetelmiin. Tutkielmassa selvitettiin TKK:n Tietojenkäsittelyopin laboratoriossa yli kymmenen vuotta kestänyttä kehitys- ja tutkimustyötä tietorakenteiden ja algoritmien opetusohjelmista sekä tutkimustuloksia niiden käytöstä opetuksen apuvälineenä.

Tutkimuksessa paneuduttiin tarkemmin tietorakenteita ja algoritmeja käsittelevään TRAKLA-opetusohjelmaan ja esitettiin menetelmä, jolla opetusohjelman käytöstä voidaan kerätä opiskelijoiden oppimista opetusohjelmassa kuvaavaa dataa. Menetelmä perustui käyttöliittymäoperaatioita vastaavien Java-olioiden luomiseen TRAKLA2-sovelmassa ja niiden välittämiseen palvelimelle RMI-protokollaa käyttäen. Palvelimella data tallennetaan helposti analysoitavaksi tekstitiedostoksi. Harjoitustehtävän ratkaisu on tietorakenteiden graafisten esitysten muokkausta opetusohjelman käyttöliittymällä. Opiskelijoiden vastaussekvenssi sisällytetään oliorakenteeseen, joka tallennetaan palvelimella Javan sarjallistamismekanismia käyttäen. Työssä esitettiin myös työkalut ja menetelmät sekä tekstitiedostojen että sarjallistettujen oliorakenteiden muokkaamiseen siten, että datan analyysi voidaan suorittaa yleisillä tilasto-ohjelmilla, kuten Microsoft Excelillä.

Opetusohjelman evaluoinnin tärkein kriteeri on opetusohjelman oppimista edistävä vaikutus. Vain oppimista edistävä tai tukeva opetusohjelma on hyödyllinen. Tutkimuksessa esitettiin suuntaviivat menetelmälle TRAKLA2-opetusohjelman evaluointiin kerätyn datan tilastollista analyysiä käyttäen. Menetelmää arvioitiin muista opetusohjelmista tehtyjen evaluaatiotutkimusten valossa ja todettiin, että datan keruu mahdollistaa pätevä tilastollisen analyysin oppimista kuvaavista muuttujista, joita voivat olla harjoitustehtävän tekemiseen käytetty aika, harjoitustehtävässä suoritettujen käyttöliittymäoperaatioiden lukumäärät, suoritusjärjestys ja suoritus-aika sekä harjoitustehtävässä tehtyjen virheiden määrä ja laatu. Opetusohjelmaa voidaan evaluoida näiden muuttujien ja opetusohjelman ominaisuuksien välisiä yhteyksiä tutkimalla.

Tutkimukseen sisältyi myös analyysi opiskelijoiden toiminnasta opetusohjelmassa TKK:n kevään 2003 Tietorakenteet ja algoritmit -kurssin aikana kerätystä datasta. Analyysi osoitti, että opiskelijoiden harjoitustehtävissä suorittamien käyttöliittymäoperaatioiden tilastollisella analyysillä voidaan tehdä päätelmiä tehtävien keskinäisestä vaikeudesta.

TRAKLA2-opetusohjelman käytettävyysskriteerit asettavat omat vaatimuksensa datan keruumenetelmälle. Datan keruu ei saa häiritä opiskelijoiden toimintaa käyttöliittymässä. Datan keruun käyttöliittymäoperaatioille aiheuttamia viiveitä tutkimalle todettiin, että datan keruu ei merkittävästi häiritse harjoitustehtävän tekemistä. Analyysin lisäksi esitettiin ohjelmistotekninen ratkaisu datan keruun tehostamiseksi entisestään ja sen aiheuttamien viiveiden vähentämiseen.

TRAKLA2-opetusohjelma on jatkuvan kehitystyön alla. Tällöin myös datan keruumenetelmän on oltava muunneltavissa: eri operaatioista kerättävän datan tietomallia on pystyttävä muuttamaan ja uusia lokitettavia operaatioita on pystyttävä määrittelemään. Tämän vuoksi datan keruumenetelmän ohjelmistoteknistä toteutusta arvioitiin sen muunneltavuuden kannalta ja todettiin, että kehitettyä menetelmää on mahdollista muuttaa joustavasti opetusohjelman muutosten myötä.

Opetusohjelman evaluointitutkimuksen kannalta on ensiarvoisen tärkeää tallentaa dataa useiden vuosien ajalta. Eri vuosina tallennetun datan on oltava vertailukelpoista. Datan keruumenetelmää arvioitiin myös datan vertailukelpoisuuden säilyttämisen kannalta. Todettiin, että eri vuosina kerätyn, hyvinkin monimutkaisia tietorakenteita sisältävän datan vertailukelpoisuuden säilyttäminen on mahdollista.

Tutkimuksessa kehitetty datan keruu- ja analysointimenetelmä TRAKLA2-opetusohjelman käytöstä voi soveltua tilastollisen evaluaatiotutkimuksen tekemiseen, jolla opetusohjelman käytön yhteyttä oppimiseen voidaan tutkia. Menetelmällä kerättyä dataa voidaan käyttää myös opetuksen kehittämisen työvälineenä, kun vertaillaan opiskelijoiden osaamista eri harjoitustehtävissä useiden vuosien ajalta. Opiskelijoiden oppimisen analyysi opetusohjelmassa on tärkeää myös adaptiivisten toimintojen kehittämisen kannalta.

Kirjallisuutta

- [1] Computing curricula 2001. <http://www.computer.org/education/cc2001/>.
- [2] JProfiler. <http://www.ej-technologies.com/products/jprofiler/overview.html>.
- [3] R. S. Baker. PILOT: an interactive tool for learning and grading. Master's thesis, Brown University, June 2000.
- [4] R. S. Baker, M. Boilen, M. T. Goodrich, R. Tamassia, and B. A. Stibel. Testers and visualizers for teaching data structures. In *Proceedings of the 30th SIGCSE Technical Symposium on Computer Science Education*, pages 261–265, New Orleans, LA, USA, 1999. ACM.
- [5] M. Ben-Ari. Constructivism in computer science education. In S. Diehl, editor, *Software Visualization: International Seminar*, pages 31–45, Dagstuhl, Germany, 2001. Springer.
- [6] M. Ben-Ari, N. Myller, E. Sutinen, and J. Tarhio. Perspectives on program animation with jeliot. In S. Diehl, editor, *Software Visualization: International Seminar*, pages 31–45, Dagstuhl, Germany, 2001. Springer.
- [7] S. Benford, E. Burke, E. Foxley, N. Gutteridge, and A. M. Zin. Ceilidh: A course administration and marking system. In *Proceedings of the International Conference of Computer Based Learning*, Vienna, Austria, 1993.
- [8] S. Bridgeman, M. T. Goodrich, S. G. Kobourov, and R. Tamassia. PILOT: an interactive tool for learning and grading. In *Proceedings of the thirty-first SIGCSE technical symposium on Computer science education*, pages 139–143. ACM Press, 2000.
- [9] S. Card, J. MacKinlay, and B. Shneiderman, editors. *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann Publishers, 1998.
- [10] A. T. Corbett and J. R. Anderson. Locus of feedback control in computer-based tutoring: impact on learning rate, achievement and attitudes. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 245–252. ACM Press, 2001.

-
- [11] E. Foxley, O. Salman, and Z. Shukur. The automatic assessment of z specifications. In *The supplemental proceedings of the conference on Integrating technology into computer science education: working group reports and supplemental proceedings*, pages 129–131. ACM Press, 1997.
- [12] E. F. Foxley, C. Higgins, E. Burke, and C. Gibbon. The ceilidh system: An overview and some experiences of use. In *Proceedings of ATCM97*, 1997.
- [13] C. D. Hundhausen, S. A. Douglas, and J. T. Stasko. A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages & Computing*, 13(3):259–290, June 2002.
- [14] J. Hyvönen and L. Malmi. TRAKLA – a system for teaching algorithms using email and a graphical editor. In *Proceedings of HYPERMEDIA in Vaasa*, pages 141–147, 1993.
- [15] D. Jackson. A semi-automated approach to online assessment. In *Proceedings of The 5th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education, ITiCSE'00*, pages 164–167, Helsinki, Finland, 2000. ACM.
- [16] D. Jackson and M. Usher. Grading student programs using ASSYST. In *Proceedings of 28th ACM SIGCSE Tech. Symposium on Computer Science Education*, pages 335–339, San Jose, California, USA, 1997. ACM.
- [17] A. Korhonen. World Wide Web (WWW) tietorakenteiden ja algoritmien tietokoneavusteisessa opetuksessa. Master's thesis, Helsinki University of Technology, 1997.
- [18] A. Korhonen. *Algorithm Animation and Simulation*. Licentiate's thesis, Helsinki University of Technology, 2000.
- [19] A. Korhonen and L. Malmi. Algorithm simulation with automatic assessment. In *Proceedings of The 5th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education*, pages 160–163, Helsinki, Finland, 2000. ACM.
- [20] A. Korhonen and L. Malmi. Proposed design pattern for object structure visualization. In *The proceedings of the First Program Visualization Workshop – PVW 2000*, pages 89–100, Porvoo, Finland, 2001. University of Joensuu.
- [21] A. Korhonen and L. Malmi. Matrix – concept animation and algorithm simulation system. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, pages 109–114, Trento, Italy, May 2002. ACM.
- [22] A. Korhonen, L. Malmi, P. Myllyselkä, and P. Scheinin. Does it make a difference if students exercise on the web or in the classroom? In *Proceedings of The 7th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education, ITiCSE'02*, Aarhus, Denmark, 2002. ACM.

- [23] A. Korhonen, L. Malmi, P. Mård, H. Salonen, and P. Silvasti. Electronic course material on data structures and algorithms. In *Proceedings of the Second Annual Finnish / Baltic Sea Conference on Computer Science Education*, pages 16–20, October 2002.
- [24] A. Korhonen, L. Malmi, J. Nikander, and P. Silvasti. Algorithm simulation – a novel way to specify algorithm animations. In M. Ben-Ari, editor, *Proceedings of the Second Program Visualization Workshop*, pages 28–36, HorstrupCentret, Denmark, June 2002.
- [25] A. Korhonen, L. Malmi, and R. Saikkonen. Matrix – concept animation and algorithm simulation system. In *Proceedings of The 6th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education*, page 180, Canterbury, United Kingdom, 2001. ACM.
- [26] A. Korhonen, L. Malmi, and P. Silvasti. TRAKLA2: a framework for automatically assessed visual algorithm simulation exercises. To appear in: *Proceedings of Kolin Kolistelut - First Annual Baltic Conference on Computer Science Education*, Joensuu, Finland, 2003.
- [27] A. Korhonen, E. Sutinen, and J. Tarhio. Understanding algorithms by means of visualized path testing. In S. Diehl, editor, *Software Visualization: International Seminar*, pages 256–268, Dagstuhl, Germany, 2002. Springer.
- [28] L. Malmi, A. Korhonen, and R. Saikkonen. Experiences in automatic assessment on mass courses and issues for designing virtual courses. In *Proceedings of the 7th annual conference on Innovation and technology in computer science education*, pages 55–59. ACM Press, 2002.
- [29] L. Malmi, A. Korhonen, and R. Saikkonen. Experiences in automatic assessment on mass courses and issues for designing virtual courses. In *Proceedings of The 7th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education, ITiCSE'02*, pages 55–59, Aarhus, Denmark, 2002. ACM.
- [30] F. Z. Mansouri, C. A. Gibbon, and C. A. Higgins. Pram: prolog automatic marker. In *Proceedings of the 6th annual conference on the teaching of computing and the 3rd annual conference on Integrating technology into computer science education*, pages 166–170. ACM Press, 1998.
- [31] B. Martin and A. Mitrovic. Domain modeling: Art or science? In *Proceedings of the 11th Int. Conference on Artificial Intelligence in Education AIED 2003*, pages 183–190. IOS Press, 2003.
- [32] M. Mayo and A. Mitrovic. Using a probabilistic student model to control problem difficulty. *Lecture Notes in Computer Science*, 1839:524–533, 2000.
- [33] V. Meisalo, E. Sutinen, and J. Tarhio. *Modernit oppimisympäristöt: Tietotekniikan käyttö opetuksen ja oppimisen tukena. (Modern Learning Environments: Using Computers to Support Teaching and Learning.)*. Tietosanoma, 2000.

- [34] A. Mitrovic. Learning sql with a computerized tutor. In *Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education*, pages 307–311. ACM Press, 1998.
- [35] A. Mitrovic. Investigating students' self-assessment skills. *Lecture Notes in Computer Science*, 2109:247–250, 2001.
- [36] A. Mitrovic and B. Martin. Evaluating the effects of open student models on learning. *Lecture Notes in Computer Science*, 2347:296–305, 2002.
- [37] A. Mitrovic, B. Martin, and M. Mayo. Using evaluation to shape its design: Results and experiences with sql-tutor. *User Modeling and User-Adapted Interaction*, 12(2-3):243–279, 2002.
- [38] A. Mitrovic and S. Ohlsson. Evaluation of a constraint-based tutor for a database language. *Artificial Intelligence in Education*, 10(3-4):238–256, 1999.
- [39] A. Mitrovic and P. Suraweera. Evaluating an animated pedagogical agent. *Lecture Notes in Computer Science*, 1839:73–82, 2000.
- [40] B. Price, R. Baecker, and I. Small. A principled taxonomy of software visualization. *Journal of Visual Languages and Computing*, 4(3):211–266, 1993.
- [41] B. J. Reiser, P. Friedmann, J. Gevins, D. Y. Kimberg, and M. Ranney. A graphical programming language interface for an intelligent LISP tutor. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 39–44. ACM Press, 1988.
- [42] R. Saikkonen, L. Malmi, and A. Korhonen. Fully automatic assessment of programming exercises. In *Proceedings of The 6th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education, ITiCSE'01*, pages 133–136, Canterbury, United Kingdom, 2001. ACM.
- [43] Sun Microsystems Inc, Mountain View, CA. *Java Object Serialization Specification*, 1998. Available at: <http://java.sun.com/products/jdk/1.2/docs/guide/serialization/spec/seri%a1TOC.doc.html>.
- [44] Sun Microsystems Inc, Mountain View, CA. *Java Remote Method Invocation Specification*, 2001. Available at: <http://java.sun.com/j2se/1.4/docs/guide/rmi/spec/rmiTOC.html>.
- [45] U.S. Naval Observatory. *What is universal time?*, 2002. Available at: <http://aa.usno.navy.mil/faq/docs/UT.htm>.
- [46] G. Weber and M. Specht. User modeling and adaptive navigation support in WWW-based tutoring systems. In A. Jameson, C. Paris, and C. Tasso, editors, *User Modeling: Proceedings of the Sixth International Conference, UM97*, pages 289–300. Springer Wien New York, Vienna, New York, 1997.