Koneoppimisesta agenttijärjestelmissä: esimerkkinä EduAgents-oppimisympäristö

Petri Vuorenmaa

Tampereen yliopisto
Tietojenkäsittelyopin laitos
Pro gradu -tutkielma
Marraskuu 2001

Tampereen yliopisto
Tietojenkäsittelyopin laitos
Petri Vuorenmaa: Koneoppimisesta agenttijärjestelmissä: esimerkkinä EduAgents-oppimisympäristö
Pro gradu -tutkielma, 80 sivua, 4 liitesivua
Marraskuu 2001

EduAgents-järjestelmä on sosiaalinen oppimisympäristö, jossa varsinaisen oppilaan apuna toimii virtuaalisia kumppanioppijia, agentteja, jotka pyydettäessä antavat oman ehdotuksensa meneillään olevan harjoitustehtävän ratkaisemiseksi. Tässä tutkielmassa käsitellään näiden kumppaniagenttien opettamista tunnistamaan tehtävänratkaisuun liittyvät sallitut toimenpiteet virheellisistä toimenpiteistä. Agenttien opettamiseen on käytetty neljää erilaisia koneoppimismenetelmää: ID3:a, C4.5:ttä, geneettistä algoritmia sekä k-nearest neighbor -algoritmia.

Koska harjoitustehtävien ratkaisu koostuu useasta perättäisestä toimenpiteestä, myös toimenpiteiden yhdistäminen sopiviksi sarjoiksi on tärkeätä. Kumppaniagenteille on tästä johtuen suunniteltu erilaisia valintastrategioita, jotka ohjaavat niitä kohti tehtävän kokonaisratkaisua.

Sekä oppimisalgoritmeja että valintastrategioita on kokeellisesti vertailtu. Vertailuissa huomiota on oppimisalgoritmien kohdalla kiinnitetty niiden kykyyn erotella sallitut toimenpiteet virheellisistä, valintastrategioiden kohdalla siihen, kuinka usein niiden avulla tehtävän ratkaisu löytyy. Myös suoritusaikoja ja toimivuutta yhdessä olemassa olevan järjestelmän kanssa on tutkittu.

SISÄLLYS

1. Johdanto

Lyhyesti määriteltynä koneoppivia ovat sellaiset ohjelmat, jotka kokemustensa avulla kykenevät muuttamaan itseään niin, että niiden toiminta jollakin tavoin muuttuu paremmaksi. Koneoppimismenetelmät havaitsevat usein asioiden välisiä yhteyksiä, jotka muilta menetelmiltä jäävät huomaamatta, ne myös sopeutuvat muuttuvaan tai lisääntyvään tietoon luonnostaan. Lisäksi, vaikka monet koneoppimistekniikat on johdettu ihmisen oppimistavoista, on myös päinvastainen kehitys mahdollista: koneoppimisessa toimivat ideat saattavat auttaa ymmärtämään biologisen oppimisen tiettyjä osa-alueita (Nilsson, 1996). Koneoppimisalgoritmien yksityiskohtainen ymmärtäminen saattaa Mitchellin (1997) mukaan johtaa ihmisen oppimiskyvyn tai -kyvyttömyyden parempaan ymmärtämiseen. Samoilla linjoilla on myös Langley (1996), jonka mukaan koneoppimisen tutkimusala käsittää sekä koneellisten että inhimillisten tiedonkäsittelytapojen tutkimisen. Tässä valossa koneoppimismenetelmien liittäminen oppimisympäristöjen kumppanioppija-agentteihin on houkutteleva ajatus. Eräs oppimisympäristö, jonka agentteihin koneoppimista voi soveltaa, on EduAgents.

EduAgents-järjestelmä on Tampereen yliopistossa kehitetty sosiaalinen agenttiperustainen oppimisympäristö. Järjestelmän tarkoitus on tukea oppijan ongelmanratkaisutaitojen kehittämistä sekä erilaisten oppimistyökalujen että sosiaalisen ympäristön avulla. Sosiaalisen ympäristön muodostaa opettaja-agentti, joka antaa ja tarkistaa harjoitustehtäviä sekä kumppanioppija-agentti, jonka kanssa oppija voi tehdä monenlaista yhteistyötä tehtäviä ratkaistessaan. (Niemirepo, 1997a,b.)

EduAgents-järjestelmää käyttävä oppilas saa ratkaistavakseen tehtäviä, jotka koostuvat useammasta lopputulokseen johtavasta välivaiheesta. Annettu tehtävä on aina ratkaistava askel kerrallaan, askeleiden suoritusjärjestystä ei kuitenkaan ole määrätty, kunhan oikea lopputulos saavutetaan. Sallitut askeleet eli toimenpiteet on määritelty

oppiaineen kuvaavassa sovellusaluemoduulissa sääntömuodossa. Kumppaniagentin tehtävä on joka vaiheessa tehtävän edetessä tarjota omaa ehdotustaan perusteluineen seuraavaksi suoritettavaksi toimenpiteeksi. Tarjoamansa toimenpiteen se valitsee hakemalla ensin oppiaineen säännöstön avulla kaikki tilanteeseen sopivat ratkaisuehdotukset ja etsimällä näistä sellaisen jota on jo aiempien tehtävien yhteydessä käytetty. Jos tilanteeseen sovellettavissa olevista toimenpiteistä ainoatakaan ei ole aiemmin käytetty, kumppaniagentti valitsee jonkin näistä satunnaisesti.

Kumppaniagentti voi kuitenkin valita toimenpiteen toisinkin: kumppaniagentille voidaan etukäteen opettaa toimenpiteiden sallittua käyttöä ja se voi käyttää oppimaansa valintaa tehdessään. Opetusvaiheessa agentti muodostaa itselleen säännöstön tai muun sen kaltaisen päätöksentekoa auttavan tietorakenteen, jonka avulla se erottaa toisistaan sallitut ja virheelliset toimenpiteet. Tätä säännöstöä kumppaniagentti voi myöhemmin lisää oppiessaan täydentää. Kumppanien opettamisella saavutettavia etuja ovat muun muassa ajoaikainen nopeus ja toimenpide-ehdotuksen perustelujen tarkkuus ja yksilöllisyys.

Oleellinen osa tätä työtä on neljän kumppaniagenttien koulutukseen soveltuvan oppimisalgoritmin toteuttaminen Prolog-kielellä sekä niiden liittäminen olemassa olevaan EduAgents-järjestelmään. Kolme näistä menetelmistä – ID3, C4.5 ja k-nearest neighbor -algoritmi – on toteutettu muutamia tarkennuksia lukuun ottamatta kirjallisuudesta löytyviä kuvauksia noudattaen. Menetelmistä neljäs eli geneettinen algoritmi sitä vastoin seuraa vain hyvin yleisellä tasolla Hollandin (1986) esittämiä suuntaviivoja, eloonjäämisfunktion, risteytystavat ja mutaatioiden toteutuksen olen itse suunnitellut.

Toinen seikka, jota alkuperäisen järjestelmän kumppaniagenttien toiminnassa voi tehostaa, on se, että ne eivät ota lainkaan huomioon toimenpide-ehdotustensa hyödyllisyyttä tehtävän ratkaisemisen kannalta, toisin sanoen ne eivät suunnittele toimenpidesarjoja, jotka johtavat kohti lopullista vastausta, vaan keskittyvät pelkästään tehtävän kulloiseenkin tilaan soveltuvaan yksittäiseen toimenpiteeseen. Kuitenkin tehtävän jokaisessa vaiheessa on usein kymmeniä tilanteeseen soveltuvia toimenpiteitä ja tehtävän kokonaisratkaisu saattaa vaatia kymmenenkin toimenpiteen sarjan, eikä lopputulokseen johtavia toimenpideketjuja ole kuitenkaan kuin muutamia. Tällöin on erittäin todennäköistä, että kumppaniagentin ehdotukset, vaikka olisivatkin täysin korrekteja, eivät ollenkaan auta ratkaisemaan tehtävää kokonaisuutena. On siis tarpeellista opettaa kumppaniagentit myös suunnittelemaan sellaisia toimenpidesekvenssejä, jotka johtavat tehtävän lopulliseen ratkaisuun.

Perinteisiä koneoppimismenetelmiä ja agentteja on aiemminkin yhdistetty: ajankäyttöä suunnitteleva Calendar Apprentice (Mitchell *et al.* 1994) käyttää ID3-algoritmia hyväkseen, Pattie Maes (1994) kertoo sähköpostitoimintaa helpottavasta agentista nimeltä Maxims, joka käyttää nearest neighbor -algoritmia sekä NewT-agentista, joka oppii geneettisen algoritmin avulla suodattamaan uutisryhmien jyviä akanoista, ID3-algoritmia hyödyntää myös Knoblockin ja Ambiten (1997) esittelemä tiedonetsintäagentti. Koneoppivien agenttien yhdistämistä oppimisympäristöihin on niin ikään tutkittu: Boy (1997) kirjoittaa ACTIDOC-ympäristöstä, missä opiskellaan yhteistyössä mm. tapauskohtaista päättelyä hyödyntävien agenttien kanssa, Canut *et al*. (1999) esittelevät geneettisen algoritmin avulla oppivan Systemion-agentin, joka toimii ohjaajana erilaisilla sovellusalueilla ja Keeling (1999) pohtii yleisellä tasolla tapoja, joilla koneoppimismenetelmiä voitaisiin käyttää hyväksi sovellusaluetietämyksen muokkaamisessa ympäristön agenttien ymmärtämään muotoon.

Tutkielman seuraavassa luvussa tarkastellaan EduAgents-järjestelmää tarkemmin. Luku 3 sisältää tutkielman laajimman ja teknisimmän osuuden: siinä esitellään yksityiskohtaisesti neljä mainittua koneoppimisalgoritmia. Luvussa 4 käsitellään induktiivisessa koneoppimisessa välttämättömän esimerkkiaineiston automaattista generointia, aineistoon tallennettavaa informaatiota sekä aineiston tehokkaaseen käyttöön liittyviä kysymyksiä. Luvun lopuksi esitetään vielä oppimismenetelmien vertailun tulokset ja keskustellaan algoritmien käyttömahdollisuuksista EduAgents-järjestelmän yhteydessä. Luku 5 käsittelee toimenpidesarjojen suunnittelua. Siinä tutkitaan täydellisen etsinnän käyttämistä parhaan toimenpidesarjan löytämiseksi sekä esitellään joitakin heuristisia hakumenetelmiä. Luvussa pohditaan myös tapauskohtaisen päättelyn avulla tapahtuvaa ratkaisustrategioiden opettamista. Luvussa 6 palataan EduAgents-järjestelmään: siinä käsitellään kaiken yllä mainitun yhdistämistä olemassa olevaan kokonaisuuteen. Luvussa tarkastellaan muun muassa sitä, kuinka kumppaniagentti käyttää oppimiansa sääntöjä hyväkseen ja kuinka tehtävän ratkaisun suunnittelu käytännössä toteutetaan. Lisäksi mietitään mahdollisuuksia agenttien ajoaikaiseen jatkokoulutukseen ja käsitellään agenttien tapaa perustella tekemiään ehdotuksia.

2. EduAgents

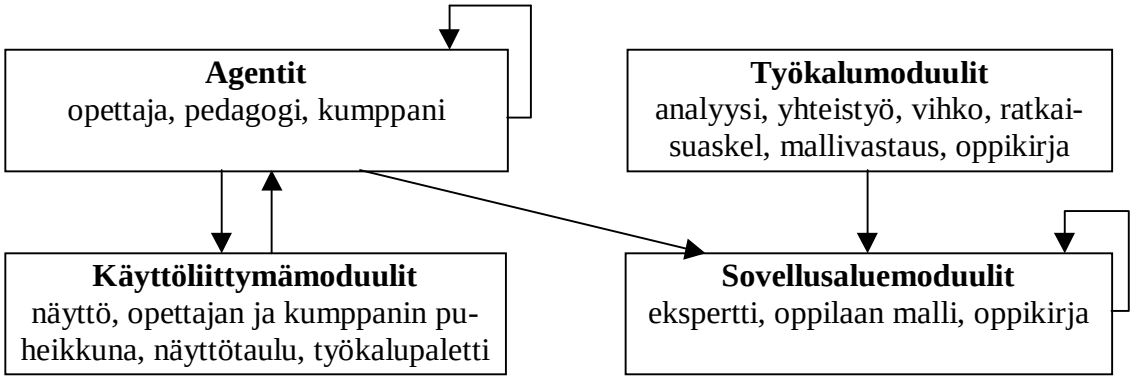
Tampereen yliopistossa käynnissä olevan EduAgents-projektin tavoitteena on yleisen, eri sovellusalueilla toimivan opetusjärjestelmän toteuttaminen. Systeemin perusidea on pedagogisen tietämyksen erottaminen opetettavan sovellusalueen tietämyksestä, joten oppiaineen vaihtaminen onnistuu sovellusaluekomponenttia vaihtamalla (Niemirepo, 1997b). Järjestelmä on agenttiperustainen sosiaalinen oppimisympäristö, joka tukee oppijan ongelmaratkaisutaitojen kehittämistä: sosiaalisen puolen muodostavat oppilaan kanssa työskentelevät opettaja- ja kumppanioppija-agentit, ongelmanratkaisutaitojen kehittämiseen on taas tarjolla useita oppimistyökaluja (Niemirepo, 1997a, b).

Aluksi (kohta 2.1.) esittelen EduAgents-järjestelmän rakenteen: agentit ja moduulit. Kohdassa 2.2. käsittelen sosiaalisten toimijoiden eli agenttien toimintatapoja tarkemmin ja kohdassa 2.3. kuvaan järjestelmän toiminnan pääpiirteissään.

2.1. Järjestelmän rakenne

EduAgents-järjestelmä koostuu moduuleista, jotka ovat agentteja, oppimistyökaluja, käyttöliittymä- tai sovellusaluemoduuleita. Jokaiselle järjestelmän moduulille on määritelty kiinteä tehtävä opetusjärjestelmässä ja tästä johtuen jokainen moduuli voidaan korvata uudella, jos uusi moduuli vain kykenee hoitamaan sille määrätyt tehtävät (Niemirepo, 1997a).

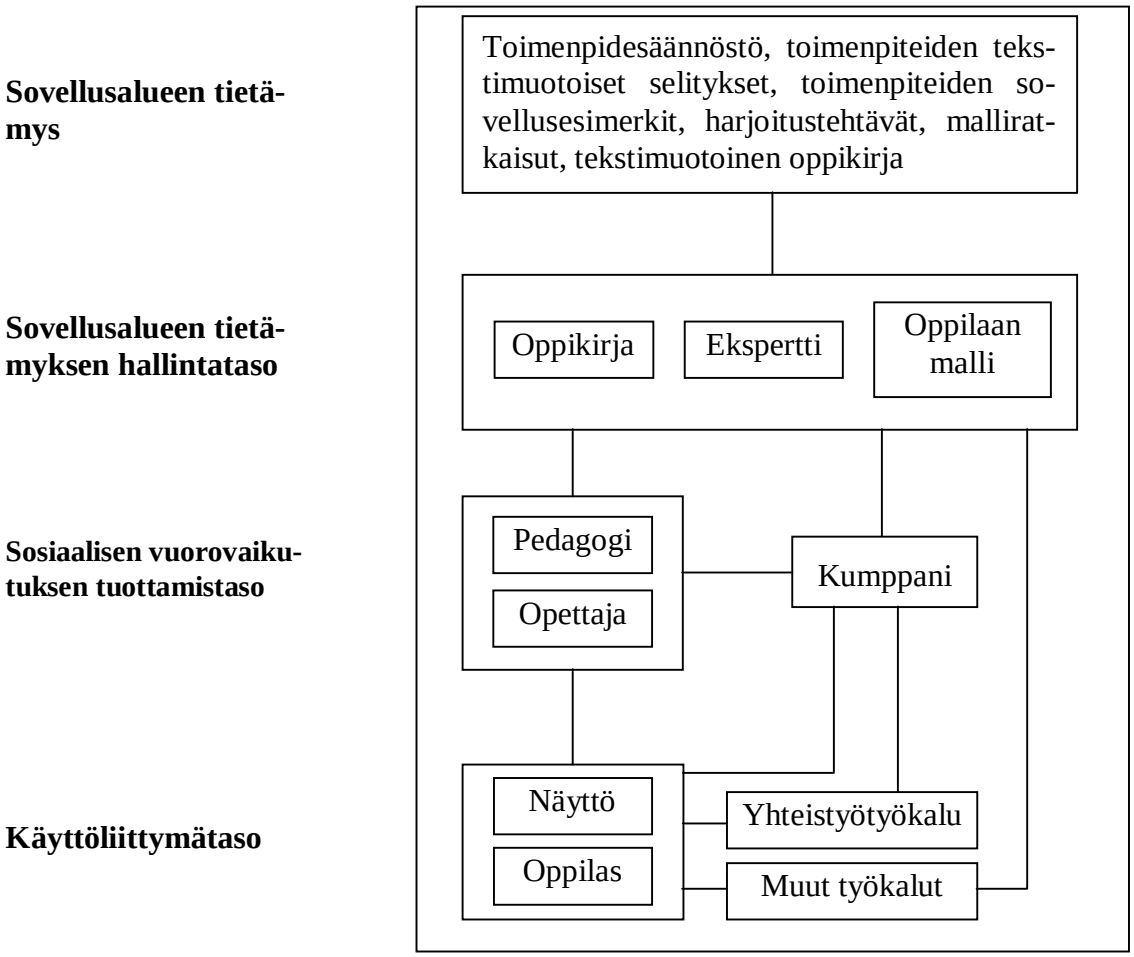
Agenttien kanssa kommunikoidaan sovituilla viesteillä ja muiden moduulien kanssa näiden tarjoamien palveluiden kautta (kuva 2.1.). Agentit käyttävät sovellusalue- ja käyttöliittymämoduulien palveluja hyväkseen. Työkalumoduulit toteuttavat itse omat



Kuva 2.1. EduAgents-järjestelmän moduulien välinen kommunikointi.

käyttöliittymänsä, joten ne käyttävät hyväkseen pelkästään sovellusaluemoduulien tarjoamaa tietämystä.

Moduulit muodostavat neljä tasoa: sovellusalueen tietämystaso, tietämyksen hallintataso, sosiaalisen vuorovaikutuksen tuottamistaso ja käyttöliittymätaso (kuva 2.2.). Siirryttäessä oppiaineesta toiseen, muutoksia tehdään pelkästään sovellusalueen tietämystasolla, muut järjestelmän osat ovat sovellusalueriippumattomia.



Kuva 2.2. EduAgents-järjestelmän arkkitehtuuri.

Sovellusalueen tietämystason pääosan muodostaa säännöstö, joka määrittelee, mitkä toimenpiteet tehtävän kussakin tilassa ovat sallittuja. Muita tälle tasolle kuuluvia asioita ovat oppilaan ratkaistavaksi laaditut harjoitustehtävät malliratkaisuineen ja sallittujen toimenpiteiden tekstimuotoiset selitykset sekä toimenpiteiden soveltamisesimerkit. Tällä hetkellä EduAgents-järjestelmään on toteutettu kaksi sovellusaluekomponenttia, toinen hallitsee ensimmäisen asteen yhtälönratkaisun ja toinen osaa lauselogiikkaa.

Sovellusalueen tietämyksen hallintatasolla toimivat oppikirja-, eksperttі- ja oppilaan malli -moduulit. Oppikirjamoduuli toimii välittäjänä sovellusaluetietämyksen ja muun järjestelmän välillä: se esimerkiksi muuntaa oppilaan antamat toimenpide-ehdotukset merkkijonomuodosta systeemin ymmärtämään muotoon, tarkistaa näiden syntaksin, antaa oppilaalle esimerkkejä toimenpiteiden käytöstä, tarjoaa oppilaan luettavaksi tekstimuotoista tietoa sovellusalueen toimenpiteistä ja tehtävänratkaisusta jne. Eksperttimoduuli tarkistaa oppilaan ja kumppaniagentin vastauksia oppikirjan ja sovellusaluetietämyksen avulla. Myös osa oppimistyökaluista käyttää ekspertin palveluita hyväkseen. Oppilaan malli -moduuli ylläpitää listaa oppilaan oikein ja väärin käyttämistä toimenpiteistä sekä oppilaan käyttäytymisestä oppimisympäristössä.

Työkalu	*Tarkoitus*
Oppikirja	näyttää määritelmät, toimenpiteet, syntaksin, esimerkit ja selitykset
Analyysi	auttaa oppilasta analysoimaan tekemiään tehtäviä
Yhteistyö	mahdollistaa yhteistyön kumppaniagentin kanssa
Ratkaisuaskel	antaa tehtävän senhetkiseen tilaan soveltuvia toimenpiteitä
Mallivastaus	antaa yhden mahdollisen ratkaisun tehtävään
Vihko	kerää automaattisesti oppilaan suoritushistoriaa

Taulukko 2.1. EduAgents-järjestelmän oppimistyökalut.

Sosiaalisen vuorovaikutuksen tasolla toimii opettaja-, pedagogi- ja kumppaniagentti. Opettaja-agentti antaa tehtäviä oppilaan ratkaistavaksi ja tarkistaa (ekspertin palveluiden avulla) tämän ratkaisuehdotuksia. Opettaja-agentin toiminta riippuu pedagogiagentista, joka toimii sen tukena. Avustavat pedagogit antavat oppilaalle enemmän palautetta ja innostavat oppilasta eri työkalujen käyttöön, toteavat pedagogit taas ilmoittavat oppilaalle ainoastaan, onko tämän vastaus oikein vai väärin. Kump-

paniagentti on oppilaan luokkatoveri, joka työskentelee opettaja-agentin johdolla saman tehtävän kimpussa kuin oppilaskin.

Käyttöliittymätasolle kuuluva näyttömoduuli välittää oppilaan toimet opettajalle. Tällaisia toimia ovat esimerkiksi pyyntö tarkistaa vastaus, pyyntö tarkistaa kumppanin vastaus tai oppimistyökalun käynnistys. Tälle tasolle on toteutettu myös valikoima erilaisia työkaluja (taulukko 2.1.), joita oppilas voi halutessaan käyttää. Työkalujen käyttöä esitellään tarkemmin kohdassa 2.3.

2.2. Järjestelmän agentit

EduAgents-järjestelmän jokaiselle neljälle opettaja-agentille on määritelty oma yksilöllinen aikuisen ihmisen ulkonäkö. Hahmoja ei ole mitenkään animoitu. Aloittaessaan ohjelman käytön oppilas valitsee yhden neljästä agentista opettajakseen ja voi halutessaan vaihtaa opettajaa myöhemmin oppitunnin kuluessa. Opettaja-agenteista kaksi on avustavia ja toiset kaksi toteavia. Molempiin ryhmiin kuuluu naispuolinen ja miespuolinen hahmo, mutta näiden ainoa ero on ulkonäkö: toiminta tuotetaan samoilla säännöillä.

Avustavat opettaja-agentit pyrkivät oppilaan tehtyä virheen määrittämään virheen laadun ekspertti- ja oppikirjamoduulien avulla ja ilmoittamaan sen oppilaalle. Toteavat opettajat kertovat vain sen, onko virhe toimenpiteessä vai vastauksessa. Avustavat opettajat ehdottavat virhetilanteissa oppilaalle työkalujen käyttöä, lisäksi ne tarjoutuvat tarkistamaan kumppaniagentin vastausehdotuksia. Toteavat opettajat taas eivät innosta oppilasta työkalujen käyttöön, eivätkä suostu tarkistamaan kumppanin ehdotuksia.

Järjestelmässä on neljä kumppaniagenttia, joista yhden oppilas valitsee kumppanikseen oppitunnin alussa. Kumppanin vaihtaminen kesken oppitunnin on mahdollista. Kumppaniagenteista kaksi on sovellusaluetaidoiltaan vahvoja, toiset kaksi heikompia, niin, että kumpaankin ryhmään kuluu yksi tyttö- ja yksi poikahahmo. Eritasoiset kumppanit eroavat sovellusaluetaitojensa lisäksi myös puhetavaltaan toisistaan. Samantasoisten agenttien ero on sitä vastoin pelkästään ulkonäössä, sillä vastaukset tuotetaan samalla ohjelmakoodilla.

Vahvat kumppanit eivät koskaan vastaa väärin, vaikka heidän vastauksensa ei välttämättä olekaan tehtävän ratkaisun kannalta paras mahdollinen. Puhetavaltaan ne ovat hieman ylimielisiä ja käyttävät käskeviä ilmauksia. Heikot kumppanit saattavat vastata välillä väärinkin ja käyttävät keskustelussa varovaisempia ilmauksia.

Kumppaniagentit ylläpitävät listaa oppilaan suorittamista toimenpiteistä. Oppitunnin alussa luetaan tiedostosta oppilaan edellisillä tunneilla ratkaisemien tehtävien vaiheet ja oppitunnin edetessä kerätään muistiin kaikki oppilaan tekemät tehtävien ratkaisuyritykset, jotka oppitunnin lopussa taas tallennetaan tiedostoon. Ratkaisuyrityksistä tallennetaan muun muassa tieto siitä, mitä toimenpidettä oppilas on käyttänyt ja onko hän käyttänyt sitä oikein vai väärin.

EduAgents-järjestelmän sovellusaluetietämykseen on sallittujen toimenpiteiden lisäksi määritelty myös muutamia *virhe-toimenpiteitä*, jotka mallintavat toimenpiteitä suoritettaessa yleisesti sattuvia virheitä. Sallittuja toimenpiteitä kutsutaan jäljempänä *ok-toimenpiteiksi*.

Kumppaniagentit eivät opi sallittujen toimenpiteiden käyttöä, vaan pyytävät kaikki tehtävän kulloiseenkin tilaan soveltuvat toimenpiteet eksperttimoduulilta. Ne vertaavat näin saamaansa mahdollisten toimenpiteiden listaa muistissaan olevaan listaan oppilaan aiemmin suorittamista toimenpiteistä, ja valitsevat ratkaisuehdotuksekseen mieluiten toimenpiteen, joka löytyy molemmilta listoilta. Vahvat ja heikot kumppanit suorittavat valinnan eri tavoilla. Seuraavissa kuvissa on esitetty nämä ehdotettavan toimenpiteen valintastrategiat.

Pyydetään kaikki tehtävään sovellettavissa olevat toimenpiteet **Tp**

Valitaan ratkaisuehdotukseksi seuraavista ensimmäinen mahdollinen

- *ok-toimenpide*, joka on listalla **Tp** ja jota oppilas on aiemmin käyttänyt oikein
- *ok-toimenpide*, joka on listalla **Tp** ja jota oppilas on aiemmin käyttänyt väärin
- mikä tahansa *ok-toimenpide*, joka on listalla **Tp**

Kuva 2.3. Vahvan kumppaniagentin tapa valita toimenpide.

Pyydetään kaikki tehtävään sovellettavissa olevat toimenpiteet **Tp**

Jos listalla **Tp** on toimenpide, jota oppilas on aiemmin käyttänyt

arvotaan käytetäänkö toimenpidettä oikein (*ok-toimenpide*) vai väärin (vastaava *virhe-toimenpide*). Mitä useammin oppilas on käyttänyt toimenpidettä oikein, sitä suuremmalla todennäköisyydellä valitaan ok-toimenpide

Jos listalla **Tp** ei ole toimenpidettä, jota oppilas on aiemmin käyttänyt

valitaan mikä tahansa *ok-* tai *virhe-toimenpide*, joka on listalla **Tp**

Kuva 2.4. Heikon kumppaniagentin tapa valita toimenpide.

Vahva kumppani valitsee aina ok-toimenpiteen, se ei siis koskaan vastaa väärin. Se valitsee mieluiten sellaisen toimenpiteen, jota oppilas on aiemmin käyttänyt. Myös heikko kumppani asettaa etusijalle toimenpiteet, joita oppilas on jo käyttänyt, mutta se voi valita arpaonnesta riippuen myös vastaavan virhe-toimenpiteen. Tilanteissa, joissa mitään oppilaalle tuttua toimenpidettä ei voi käyttää, heikko kumppani voi yhtä hyvin ehdottaa niin sallittua kuin virheellistäkin toimenpidettä.

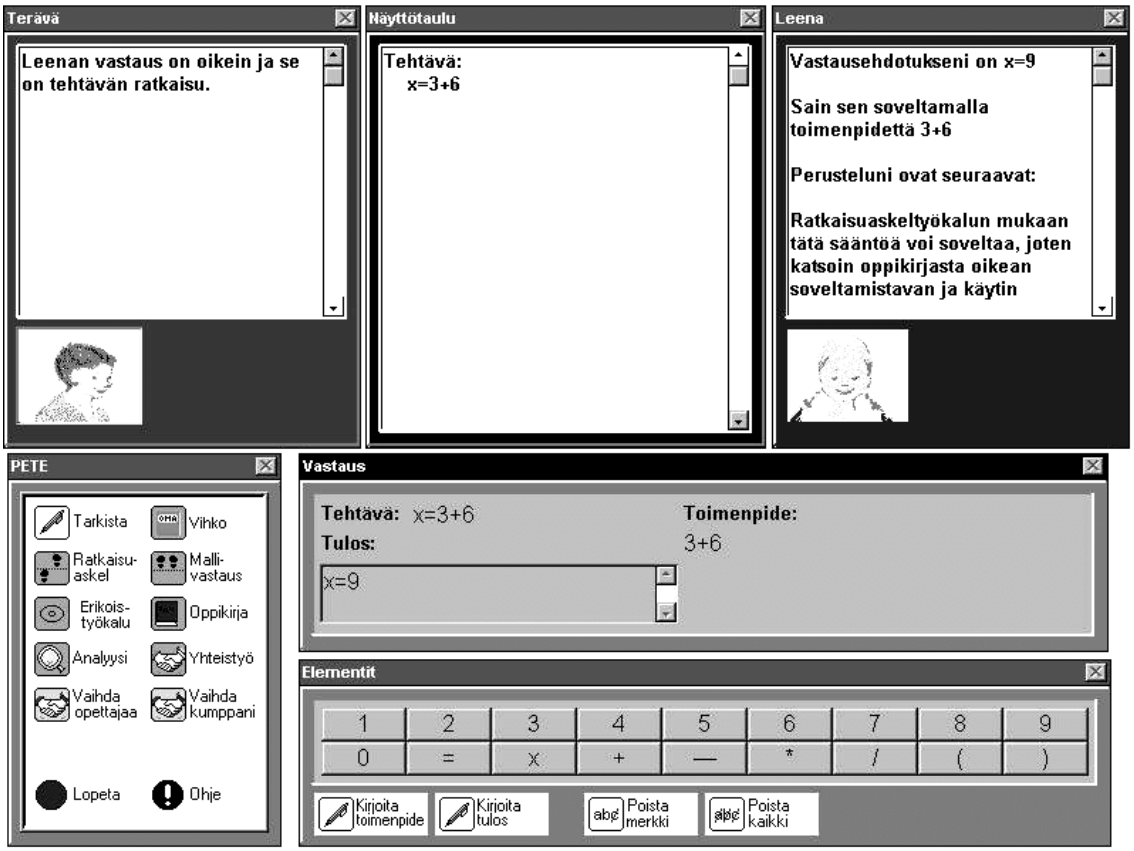
Yhtälönratkaisun sovellusalueella tehdyssä käyttötutkimuksessa (Hietala and Niemirepo, 1997) ilmeni, että koehenkilöinä toimivat peruskoulun seitsemäsluokkalaiset kokivat kumppaniagentit tärkeiksi. Yhteistyötä kumppanien kanssa arvostettiin lähinnä agenttien persoonallisten, ihmismäisten ominaisuuksien vuoksi. Koehenkilöiden mielestä yhteistyötyökalu olikin työkaluista hyödyllisin.

2.3. Järjestelmän toiminta

Oppituntia aloitettaessa alustetaan oppilaan malli –moduuli sekä kumppani- ja opettaja-agentit. Oppilaan mallin alustuksessa ladataan tiedostosta tiedot siitä, miten oppilas on aiemmilla oppitunneilla toiminut. Tiedosto sisältää muun muassa tiedot oppilaan ratkaisemista harjoitustehtävistä ja toimenpiteistä, joita hän on käyttänyt tehtäviä ratkaistessaan. Kumppaniagentit tallentavat aina oppitunnin aikana tekemänsä toimenpide-ehdotukset tiedostoon ja uuden oppitunnin alussa ne ladataan taas agentin muistiin eli kumppanit muistavat aiemmilla oppitunneilla opitut asiat. Opettaja-agenttia alustettaessa asetetaan valittu pedagogiagentti aktiiviseksi. Tämän jälkeen opettaja avaa käyttöliittymän keskeiset ikkunat (kuva 2.5.). Opettajan ja kumppanin ikkunoihin liitetään myös valittujen agenttien kuvat ja nimet.

Aloituksessa pedagogiagentti tarkistaa oppilaan mallista, jatketaanko kesken jääneen tehtävän ratkaisemista vai aloitetaanko uusi tehtävä. Pedagogi tallentaa tehtävän tilan oppilaan malliin, kertoo kumppaniagentille, mikä on aloitustehtävä ja tehtävän vaihe ja päivittää näyttötaululle tehtävän tilan.

Oppilaan pitää aina ratkaista harjoitustehtävä loppuun asti, ennen kuin hän saa ratkaistavakseen uuden tehtävän. Uuden tehtävän valitsee pedagogiagentti. Avustavat pedagogit etenevät nopeasti antamalla uusia piirteitä sisältäviä tehtäviä, jos oppilas ei tee virheitä. Toteavat pedagogit taas luottavat kertauksen voimaan: ne antavat aina neljä samankaltaista tehtävää huolimatta oppilaan edellisistä suorituksista.



Kuva 2.5. EduAgents-järjestelmän käyttöliittymä.

Oppilaan antaman vastauksen tarkistus aloitetaan oppilaan painaessa työkalupaletin Tarkista-painiketta. Näyttömoduuli lukee vastausikkunasta oppilaan ehdottaman toimenpiteen ja vastauksen, tallentaa ne oppilaan malliin ja lähettää ne edelleen opettaja-agentille. Opettaja varmistaa aluksi eksperttimoduulin avulla, että oppilaan antama toimenpide on olemassa. Jos toimenpide on olemassa eli sen kuvaus löytyy oppikirjasta, pyytää opettaja eksperttimoduulia tarkistamaan toimenpiteen ja vastauksen syntaksin ja oikeellisuuden. Tarkistuksen tulos voi olla syntaksivirhe, väärä vastaus, tuntematon toimenpide, soveltumaton toimenpide, oikea vastaus tai tehtävän ratkaisu. Myös tarkistuksen tulos tallennetaan oppilaan malliin ja tiedot sekä oppilaan vastauksesta että sen oikeellisuudesta lähetetään kumppaniagentille.

Oppilas voi tehdä yhteistyötä kumppaniagentin kanssa kahdella tavalla. Tehtäväntekotilassa oppilas voi pyytää kumppania antamaan ja perustelemaan vastausehdotuksensa. Kumppaniagentti perustelee, jos mahdollista, ehdotuksensa aiempien tehtävien avulla. Jos perustelu ei onnistu nojautumalla jo ratkaistuihin tehtäviin, käyttää kumppani perustelunaan eri työkalujen käyttöä. Yhteistyötyökalun avulla oppilas voi tehdä laajempaakin yhteistyötä kumppaninsa kanssa, esimerkiksi pyytää kumppanilta useam-

pia ratkaisuehdotuksia, pyytää tätä soveltamaan oppilaan ehdottamia toimenpiteitä ja kysyä agentin mielipiteitä näistä ehdotuksista.

Oppilaalla on lisäksi käytössään joukko työkaluja, jotka käynnistyvät työkalupaletin painikkeista (kuva 2.5.). Vihkotyökalu kerää automaattisesti talteen oppilaan tekemät vastausehdotukset ja niiden tarkistuksen tulokset. Oppilas voi vihkoa selaamalla palauttaa mieleensä, kuinka hän on jotakin toimenpidettä aikaisemmin soveltanut.

Analyysityökalun avulla oppilas voi tarkastella ratkaisemiaan tehtäviä askeleittain. Jokaisen ratkaisuaskeleen kohdalla oppilas voi pyytää selityksiä ja esimerkkejä käyttämänsä toimenpiteen käytöstä tai hän voi pyytää vaihtoehtoja käytetylle toimenpiteelle. Lisäksi oppilas voi kysyä, missä yhteydessä hän aiemmin on käyttänyt kyseistä toimenpidettä tai harjoitella erillisten pienten harjoitusten avulla valitun toimenpiteen käyttöä.

Ratkaisuaskeltyökalu näyttää oppilaalle uusia mahdollisuuksia edetä tehtävänratkaisussa. Se antaa kaikki oppikirjan määrittämät, tehtävän senhetkiseen tilaan soveltuvat toimenpiteet yleisessä muodossaan. Oppilaan tehtäväksi jää toimenpiteen parametrien alustaminen siten, että se soveltuu käytettäväksi ratkaistavana olevan tehtävän yhteydessä.

Mallivastaustyökalulla oppilas voi tutkia yhtä mahdollista tehtävänratkaisua. Ratkaisu näytetään askeleittain ja jokaisen askeleen kohdalla sovelletun toimenpiteen käyttötarkoitus selitetään.

Oppikirjatyökalu tarjoaa oppilaalle käyttöliittymän sovellusalueen tietämykseen. Käyttöliittymän painikkeista oppilas voi valita tarkasteltavakseen eri sisältökohtia kuten sovellusalueen esittely, käsitteiden määritelmät, tehtävän yleiset ratkaisuohjeet, toimenpiteet ja esimerkit sekä toimenpiteiden harjoittelu. Kolme ensiksi mainittua kohtaa sisältävät tekstimuodossa kuvauksen sovellusalueesta ja sen käsitteistä. Toimenpiteet ja esimerkit –kohdassa voi lukea ohjeita halutun toimenpiteen käytöstä ja tutkia esimerkkitapauksia, joissa sitä on sovellettu. Harjoittelu-kohdassa oppilas voi valita tietyn toimenpiteen ja harjoitella sen käyttöä sopivien tehtävien avulla.

2.4. Yhteenveto

EduAgents-järjestelmä on agenttiperustainen sosiaalinen oppimisympäristö, joka tukee käyttäjän ongelmaratkaisutaitojen kehittämistä. Sosiaalisen puolen muodostavat oppi-

laan kanssa työskentelevät opettaja- ja kumppanioppija-agentit, ongelmanratkaisutaitojen kehittämiseen on taas tarjolla useita oppimistyökaluja. Järjestelmä koostuu moduuleista, joten uuden oppiaineen lisääminen on helppoa.

EduAgents-järjestelmää käyttävä oppilas saa ratkaistavakseen opettaja-agentin antamia harjoitustehtäviä, jotka koostuvat useammasta lopputulokseen johtavasta välivaiheesta. Annettu tehtävä on aina ratkaistava askel kerrallaan, askeleiden suoritusjärjestystä ei kuitenkaan ole määrätty. Oppilaan valitseman kumppaniagentin tehtävä on tehtävän edetessä tarjota omaa ehdotustaan seuraavaksi suoritettavaksi toimenpiteeksi. Tarjoamansa toimenpiteen se valitsee hakemalla ensin oppiaineen sovellusaluetietämyksen avulla kaikki tilanteeseen sopivat ratkaisuehdotukset ja etsimällä näistä sellaisen jota on jo aiempien tehtävien yhteydessä käytetty. Jos tilanteeseen sovellettavissa olevista toimenpiteistä ainoatakaan ei ole aiemmin käytetty, kumppaniagentti valitsee jonkin näistä satunnaisesti. On myös mahdollista että kumppaniagentti vastaa väärin.

3. Käytetyt koneoppimismenetelmät

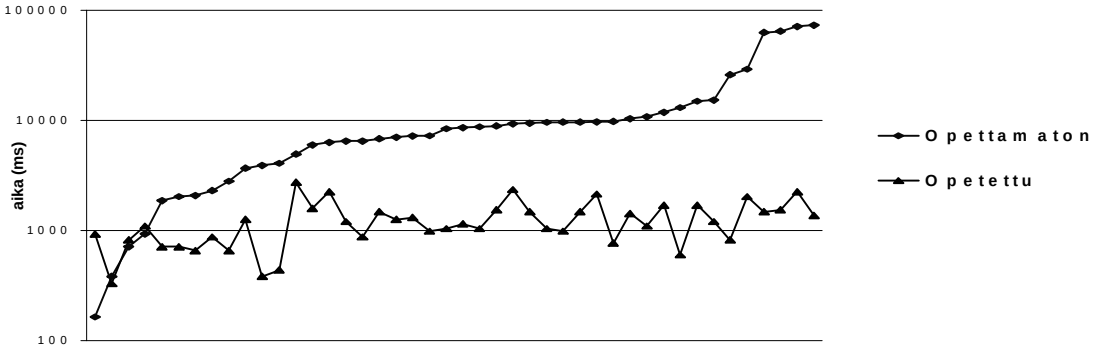
Edellisessä luvussa esitellyn EduAgents-järjestelmän kumppaniagentit tarjoavat aina tehtävän edistyessä tai vaihtuessa omaa toimenpide-ehdotustaan. Jotta ehdotuksista olisi varsinaiselle oppijalle hyötyä, niiden pitää olla järkeviä. On myös hyvä, jos ehdotukset eivät liian usein sisällä virheitä, jotta oppijan luottamus kumppaniin säilyy. Eräs tapa saavuttaa nämä tavoitteet on opettaa kumppaniagenteille etukäteen sovellusalueen toimenpiteiden sallittua käyttöä.

Induktiiviset koneoppimismenetelmät jaetaan usein (esim. Langley, 1996) karkeasti ottaen neljään ryhmään: esimerkkitapausten analysointiin perustuvat menetelmät, hermoverkot, geneettiset algoritmit ja esimerkkitapausten vertailuun (*instance-based*) perustuvat menetelmät. Tämän työn yhteydessä kumppaniagenttien opetukseen on käytetty neljää eri oppimismenetelmää: ID3:a, C4.5:ttä, k-nearest neighbor -menetelmää ja geneettistä algoritmia. Menetelmät on valittu osittain edellä mainittua jakoa silmällä pitäen: ID3 ja C4.5 edustavat esimerkkitapausten analysointiin perustuvaa suuntausta, k-nearest neighbor -menetelmä perustuu esimerkkitapausten vertailuun ja geneettinen algoritmi edustaa evoluutiota mallintavaa ryhmää. Hermoverkkotyyppistä oppimisalgoritmia ei ole valittu mukaan, koska hermoverkon opettaminen perustuu työlääseen numeeriseen laskentaan ja koska EduAgents-järjestelmän toteutuskieli Prolog soveltuu tällaiseen laskentaan huonosti. Toinen syy on se, että hermoverkkojen oppimistuloksena ei synny minkäänlaisia sääntöjenkaltaisia tietorakenteita, joten myöskään minkäänlaisia ymmärrettäviä perusteluita hermoverkon avulla tehdyille ratkaisuille ei ole saatavissa. Ensimmäiseen ryhmään kuuluvia algoritmeja on ryhmän laajuuden vuoksi valittu kaksi kappaletta. ID3 on ansainnut paikkansa, koska se on alansa klassikko, vanha ja tunnettu, oppimiskyvyiltään kuitenkin kilpailukykyinen minkä tahansa uudemman algoritmin kanssa. C4.5 taas edustaa ID3:n aloittaman kehityksen nykypäivää.

EduAgents-järjestelmän sovellusaluemoduulit sisältävät sovellusalueen täydellisen tietämyksen (järjestelmän tarpeita ajatellen) sääntöjen muodossa. Säännöt määräävät sallitut toimenpiteet ja toimenpiteiden soveltamisen tulokset kussakin tehtävän vaiheessa. Järjestelmän kumppaniagentit käyttävät tätä säännöstöä muodostaessaan ratkaisuehdotuksiaan. Kun agenteilla on siis käytössään sovellusalueen täydellinen tietämys, miksi toimenpiteiden käyttöä pitäisi niille erikseen opettaa? Tähän on kolme syytä:

- Opetettu kumppani antaa ajoaikana ratkaisuehdotuksensa nopeammin.
- Epätäydellisestä oppimisesta saattaa seurata opettavaisia virheitä. Kumppani saattaa ehdottaa toimenpidettä, joka on sinänsä järkevä, mutta jota ei saa ko. tilanteessa soveltaa.
- Opetettu kumppani osaa perustella ehdotuksensa ts. selittää miksi toimenpidettä saa tai ei saa kyseisessä tilanteessa käyttää.

Alla olevassa kuvassa on esitetty mahdollisten toimenpiteiden määrittämiseen kuluva aika käyttäen kahta eri menetelmää: alkuperäisen EduAgents-järjestelmän tapaa, jossa kumppaniagentti käyttää sovellusalueen säännöstöä ja kumppaniagenttia, joka käyttää oppimaansa päätöspuuta. Vertailussa on etsitty 44 loogiseen lauseeseen (logiikan sovellusalueen harjoitustehtävät) sovellettavissa olevat toimenpiteet. Kuvasta nähdään, että opetettu kumppani selviytyy tehtävästään noin kymmenen kertaa opettamatonta kumppania nopeammin (kuvan aika-asteikko on logaritminen). Kumppaniagenttien opettaminen tapahtuu etukäteen, ei ajoaikana. Opettamiseen kuluvalla ajalla ei siis ole ohjelman suorituksen kannalta merkitystä.



Kuva 3.1. Opettamattoman ja opetetun kumppanin toimenpiteiden etsimiseen käyttämä aika.

Alkuperäisessä järjestelmässä sovellusalueiden säännöstöihin on lisätty ns. virhesääntöjä, jotka mallintavat joitakin toimenpiteitä sovellettaessa tapahtuvia virheitä. Virhe-säännöt eivät tietenkään kata kaikkia mahdollisia (eivätkä ehkä edes yleisiä) virheitä, joita oppilas voi tehdä. Kumppani tarjoaa ajoittain näihin virhe-sääntöihin perustuvia toimenpiteitä ratkaisuehdotukseksi. Kumppaniagenttien opettamiseen perustuvassa järjestelmässä taas kumppanin tekemät virheet ovat satunnaisempia, koska ne johtuvat epätäydellisestä oppimisesta ja voivat siten olla opettavaisempia.

Oppiessaan sovellusalueen tietämystä kumppaniagentti muodostaa itselleen jossakin muodossa olevan säännöstön, jonka avulla se päättää, mitkä toimenpiteet ovat sallittuja ja mitkä eivät. Tämän säännöstön avulla on helppo muodostaa perustelu ehdotetulle toimenpiteelle. Perustelu on yksinkertaisesti selkokielelle kirjoitettuna ne säännöt, joiden avulla ehdotettu toimenpide todettiin sallituksi.

Kumppaniagenttien opettamiseen käytetyt neljä tunnettua induktiivista koneoppimisalgoritmia esitellään kohdissa 3.2. - 3.5. Kohdassa 3.1. käydään läpi kaikille algoritmeille yhteiset oppimisen perusteet.

3.1. Induktiivinen koneoppiminen

Induktiivisessa koneoppimisessa oppimisalgoritmille annetaan joukko esimerkkitapauksia, jotka kuvaavat opetettavaa aihetta tai käsitettä. Tämän aineiston perusteella algoritmi muodostaa aiheesta yleisen määritelmän tai säännöstön, joka hyväksyy kaikki positiiviset esimerkit ja sulkee pois kaikki negatiiviset esimerkit. Yleensä oppimisen tavoite on muodostaa määritelmä, joka ilmaisee kuuluuko jokin yksittäistapaus tiettyyn luokkaan vai ei. Tällöin esimerkkitapaukset kuvataan attribuuttijoukolla ja kerrotaan mihin luokkaan kukin tapaus kuuluu eli muodossa ([$Attr_1$,$Attr_2$,...,$Attr_n$],Luokka). Luokan määritelmä muodostetaan etsimällä attribuuteille sellaiset arvot, että määritelmä hyväksyy kaikki luokkaan kuuluvat esimerkit ja hylkää kaikki muihin luokkiin kuuluvat tapaukset. Kun kaikille aihepiirin luokille on muodostettu määritelmät, voidaan uuden tapauksen luokka määrittää annetun attribuuttijoukon avulla.

Kumppaniagenttien pitäisi oppia aihe ’onko toimenpide sallittu’. Tässä tapauksessa luokkia on siis kaksi: on ja ei. Seuraavaksi esitellään muutamia algoritmeja, joilla toimenpiteiden käyttöä kumppaneille opetetaan.

3.2. ID3

Quinlanin (1983, 1986) kehittämä ID3 on yksi tunnetuimmista koneoppimismenetelmistä. Se pohjautuu Earl Huntin jo vuonna 1963 esittelemään CLS (Concept Learning System) algoritmiin. CLS on TDIDT (top-down induction of decision trees) ohjelmaperheen kantaisä. TDIDT-systeemissä muodostetaan päätöspuu, jonka solmut ovat attribuutteja, oksat attribuuttien arvoja ja lehdet luokkia. Etenemällä puun juuresta lehtisolmuun saadaan määritelmä tai sääntö lehteä vastaavalle luokalle.

3.2.1. Päätöspuun muodostaminen

Algoritmi toimii rekursiivisesti muodostamalla ensin puun juuren ja sen jälkeen alipuut yksitellen (*top-down*). Algoritmi saa syötteenä joukon esimerkkitapauksia (harjoitusaineisto), jotka esitetään muodossa (Luokka, [$Attr_1$, $Attr_2$,..., $Attr_n$]). Päätöspuun T muodostaminen esimerkkijoukosta S tapahtuu seuraavasti:

- Jos kaikki esimerkit joukossa S kuuluvat samaan luokkaan C, tuloksena on puu, jossa on yksi solmu, jonka nimi on C.
- muussa tapauksessa
    1. valitaan informatiivisin attribuutti A, joka voi saada arvot ovat v_1, ... ,v_n
    2. muodostetaan solmu, jonka nimi on A
    3. jaetaan S osajoukkoihin S_1, ... ,S_n A:n arvojen mukaan
    4. muodostetaan joukkoja S_i vastaavat alipuut, jotka yhdistetään solmuun A oksilla v_i

Perusalgoritmi vaatii seuraavia tarkennuksia:

- Jos joukko S on tyhjä, tuloksena on solmu, joka nimetään sen luokan mukaan, jonka esiintymiä on eniten puun ko. haaran edellisessä solmussa.
- Kohdassa 1. attribuuttia valittaessa otetaan huomioon vain ne attribuutit, joita ei ole käytetty ylempänä puussa
- Jos joukon S esimerkit eivät kuulu samaan luokkaan, eikä valittavana ole enää attribuutteja, tuloksena on puu, jossa on yksi solmu, joka nimetään sen luokan mukaan, jonka esiintymiä joukossa S on eniten.

- Kohdassa 1. pitää olla jokin kriteeri, jonka mukaan päätetään, mikä attribuutti on paras.

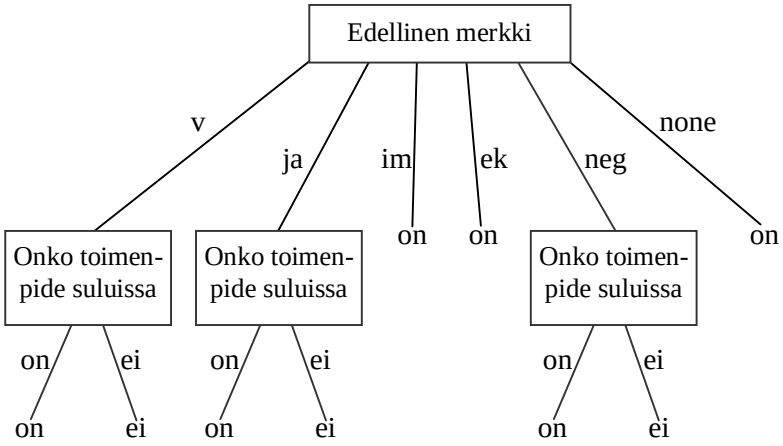
Alla oleva taulukko 3.1. sisältää pienen harjoitusaineiston, josta muodostettu päätöspuu luokittelee loogiselle lauseelle suoritetut toimenpiteet kahteen luokkaan: on (toimenpiteen käyttäminen on sallittu) ja ei (toimenpidettä ei saa käyttää). Esimerkkitapaukset on kuvattu neljällä attribuutilla. Toimenpideryhmän numero kertoo, minkä tyyppisestä toimenpiteestä on kysymys. Tämä aineisto sisältää vain kahden ryhmän toimenpiteitä: 1 (idempotenssilait) ja 2 (sulkujen poisto). Muut attribuutit kertovat onko suoritettava toimenpide suluissa ja mikä on sitä edeltävä ja seuraava merkki sievennettävässä lauseessa .

Luokka	Tp ryhmän numero	Onko tp suluissa	Edellinen merkki	Seuraava merkki
ei	1	ei	neg	none
ei	1	ei	v	none
on	1	ei	none	none
on	1	on	ja	none
on	1	on	neg	none
on	1	on	v	none
ei	2	ei	ja	none
ei	2	ei	neg	none
ei	2	ei	v	none
on	2	ei	ek	none
on	2	ei	im	none
on	2	ei	none	ek
on	2	ei	none	im
on	2	ei	none	ja
on	2	ei	none	none
on	2	ei	none	v

Taulukko 3.1. Pieni harjoitusaineisto logiikan sovellusalueelta.

Antamalla TDIDT-algoritmille syötteenä taulukossa 3.1. kuvattu esimerkkiaineisto saadaan aikaan alla olevassa kuvassa 3.2. esitetty päätöspuu, joka luokittelee täydellisesti aineiston esimerkit.

Uusi tapaus, jonka luokkaa ei tiedetä, luokitellaan päätöspuun avulla etenemällä juurisolmusta johonkin lehtisolmuista. Jokaisessa puun solmussa suoritetaan solmun nimeämää attribuuttia koskeva testi ja valitaan se eteenpäin vievä haara, jonka arvo vastaa luokiteltavan tapauksen ko. attribuutin arvoa. Lehtisolmu, johon päädytään kertoo tapauksen luokan.



Kuva 3.2. Päätöspuu, joka luokittelee lauselogiikan toimenpiteet kahteen luokkaan on (sallittu) ja ei (ole sallittu).

3.2.2. Valintakriteeri gain

Ratkaisevin ja työläin vaihe päätöspuita muodostettaessa on attribuuttien valintajärjestyksestä päättäminen, päätös siitä, mikä attribuutti kulloinkin on paras. Helpoin tapa tehdä tämä päätös on valita ensimmäinen attribuutti ensimmäiseksi, toinen seuraavaksi ja niin edelleen. Tämän 'kriteerin' mukaan muodostetut puut ovat tyypillisesti isoja ja monimutkaisia ja luokittelevat uusia tapauksia yleensä huonosti. ID3 käyttää informaatioteoriaan perustuvaa *gain-kriteeriä* valitessaan parasta attribuuttia (Quinlan, 1986). Se yrittää minimoida luokittelussa tarvittavien testien lukumäärän, toisin sanoen ID3 pyrkii muodostamaan mahdollisimman matalia päätöspuita. Jos esimerkkiaineisto S sisältää y kappaletta luokkaan Y ja n kappaletta luokkaan N kuuluvaa tapausta niin uuden tapauksen luokitteluun tarvittava informaatio on

$$I(S) = -\frac{y}{y+n}\log_2\left(\frac{y}{y+n}\right) - \frac{n}{y+n}\log_2\left(\frac{n}{y+n}\right)$$

Valitaan nyt attribuutti A, joka voi saada arvot v_1, ... ,v_k päätöspuun juureksi ja jaetaan S vastaavasti osajoukoiksi S_1, ... ,S_k . Lasketaan jokaiselle osajoukolle $I(S_i)$. Näiden pai-

notettu summa on päätöspuun tarvitsema informaatio sen jälkeen kun attribuuttia A on käytetty testinä

$$E(A) = \sum p(v_i)I(S_i),$$

missä painotuksena käytetty $p(v_i)$ on todennäköisyys sille, että A saa arvon v_i. Valitsemalla A saavutettava informaatio on erotus:

$$gain(A) = I(S) - E(A).$$

Tai useamman luokan tapauksessa:

$$gain(A) = -\sum_c p(c)\log_2 p(c) - \sum_v -p(v)\sum_c p(c,v)\log_2 p(c,v),$$

missä p(c,v) tarkoittaa todennäköisyyttä sille, että tapaus, jonka attribuutin A arvo on v kuuluu luokkaan c. ID3 käy rekursion jokaisessa vaiheessa läpi kaikki jäljellä olevat attribuutit ja valitsee aina sen attribuutin A, jolla gain(A) on suurin.

Taulukon 3.1. esimerkkiaineisto sisältää 5 luokkaan ei ja 11 luokkaan on kuuluvaa tapausta. Tästä aineistosta laskettu informaatio on siis

$$I = -\frac{5}{16}\log_2(\frac{5}{16}) - \frac{11}{16}\log_2(\frac{11}{16}) = 0.896$$

Aineiston esimerkkitapauksista 13:ssa attribuutin *onko tp suluissa* arvo on *ei*, näistä 8 kuuluu luokkaan *on* ja 5 luokkaan *ei*. Kolmen muun esimerkkitapauksen vastaavan attribuutin arvo on *on*, ja nämä kaikki kuuluvat luokkaan *on*. Joten

$$E(onko\ tp\ suluissa) = \frac{13}{16}(-\frac{5}{13}\log_2\frac{5}{13} - \frac{8}{13}\log_2\frac{8}{13}) + \frac{3}{16}(-0-0) = 0.781$$

ja

$$gain(onko\ tp\ suluissa) = 0.896 - 0.781 = 0.115.$$

Samalla tavalla laskien saadaan gain-kriteerin arvot muille attribuuteille

gain(tp ryhmän numero) = 0.0009

gain(edellinen merkki) = 0.427

gain(seuraava merkki) = 0.161

ID3 valitsee päätöspuun juureksi siis attribuutin *edellinen merkki,* jonka gain-kriteerin arvo on suurin.

Quinlan (1986, 1988) on itse myöhemmin parannellut gain-kriteeriä. Lopez de Mantaras (1989) on niin ikään jatkanut gain-kriteerin kehitystä. Muuhun kuin informaatioteoriaan perustuvia valintakriteerejä ovat esitelleet esimerkiksi Breiman *et al*. (1984), Elomaa and Ukkonen (1994) ja Kononenko (1994). Empiirisiä vertailuja valintakriteerien tehokkuudesta ovat tehneet mm. White *et al*. (1994) ja Murthy *et al.* (1994).

3.2.3. Iterointi

ID3 toimii iteratiivisesti muodostamalla sarjan päätöspuita, joiden esimerkkitapausten luokittelukyky kierros kierrokselta kasvaa, kunnes päädytään päätöspuuhun, joka luokittelee koko harjoitusaineiston oikein. Menetelmä toimii seuraavasti:

1. valitaan satunnaisesti harjoitusaineistosta joukko (Window) esimerkkitapauksia.
2. muodostetaan joukon Window avulla päätöspuu käyttämällä yllä kuvattua TDIDT -algoritmia ja gain-valintakriteeriä.
3. luokitellaan koko harjoitusaineisto muodostetun päätöspuun avulla
4. lisätään joukkoon Window ne harjoitusaineiston tapaukset, jotka päätöspuu luokittelee väärin

Toistetaan kohtia 2 – 4 kunnes päätöspuu luokittelee koko harjoitusaineiston oikein. Quinlanin (1983) mukaan tyypillisessä tapauksessa tarvitaan vain neljä toistoa oikean päätöspuun löytämiseen.

Todellisissa tapauksissa esimerkkiaineiston tiedot ovat usein epätäydellisiä. Joidenkin tapausten attribuuttien tai luokkien arvoissa voi olla virheitä, osa arvoista saattaa puuttua kokonaan. On myös tilanteita, joissa oppimisalgoritmi ei pysty täydellisesti erottelemaan luokkia toisistaan, esimerkiksi tilanne, jossa harjoitusaineisto sisältää kaksi tapausta, jotka ovat identtisiä attribuuteiltaan, mutta edustavat eri luokkaa. Yllä kuvatun silmukan lopetusehtoa pitää tästä syystä lieventää. On sallittava että, päätöspuu luokittelee joitakin tapauksia väärin.

Kirjallisuudesta ei löydy mainintoja siitä, kuinka käsitellään edellä mainitun kaltaisia tilanteita, joissa algoritmi ei pysähdy. EduAgents-järjestelmän kumppaniagentteja opetettaessa on mahdollista, varsinkin lauselogiikan sovellusalueella, päätyä päättymättömään toistoon, joten lisäsin ID3-menetelmään kaksi uutta lopetusehtoa. Toinen ehdoista on se, että iterointi toistetaan korkeintaan kymmenen kertaa ja valitaan muodostetuista kymmenestä päätöspuusta se, joka parhaiten luokittelee harjoitusaineiston. Toinen ehto keskeyttää iteroinnin, jos joukkoon Window ei tule uusia esimerkkitapauksia, eikä muodostettava päätöspuu siis enää muutu. Myös tässä tapauksessa palautetaan paras siihen mennessä aikaansaaduista päätöspuista.

3.2.4. Päätöspuun karsinta

ID3 kasvattaa päätöspuuta kunnes se luokittelee harjoitusaineiston täydellisesti, tai kunnes kaikkia aineiston attribuutteja on käytetty puun haarassa, jossa täydellinen luokittelu ei onnistu. Tämä ei ole aina toivottavaa. Jos harjoitusaineisto sisältää virheellistä tietoa päätöspuun jotkin haarat muotoutuvat näiden virheiden mukaan. Samoin, jos harjoitusaineisto on pieni, eikä kuvaa riittävän hyvin oppimisen kohteena olevaa aihepiiriä, päätöspuu saattaa erikoistua luokittelemaan vain joitakin aihepiirin erikoistapauksia. Molemmissa tapauksissa sanotaan, että päätöspuu ylisovittaa (*overfit*) harjoitusaineiston. Mitchell (1997) määrittelee ylisovittamisen seuraavasti:

> Päätöspuu T ylisovittaa harjoitusaineiston, jos on olemassa päätöspuu T', jonka tarkkuus harjoitusaineistossa on pienempi kuin puun T, mutta suurempi kaikkien aihepiiriin kuuluvien tapausten yhteydessä.

Ylisovittaminen siis heikentää päätöspuun luokittelutarkkuutta tuntemattomien tapausten yhteydessä. Ylisovittaminen pyritään välttämään päätöspuun karsinnalla. Ensiksi päätöspuu kasvatetaan täyteen mittaansa ja sen jälkeen siitä poistetaan epäluotettaviksi arvioidut alipuut. Alipuiden luotettavuuden arviointiin on kehitetty useita menetelmiä, tässä esitellään *reduced-error pruning* -menetelmä (Quinlan 1987).

Reduced-error pruning -menetelmä käyttää erillistä karsinta-aineistoa, joka on etukäteen erotettava harjoitusaineistosta. Karsinta-aineiston esimerkkitapaukset luokitellaan päätöspuun avulla, ja lasketaan jokaisessa puun sisäsolmussa virheiden lukumää-

rä jos alipuu säilytetään ja virheiden lukumäärä jos se muutetaan alipuuhun tulevien tapausten enemmistöluokan nimeämäksi lehtisolmuksi. Jos alipuun tuottamien virheiden lukumäärä suurempi se muutetaan lehtisolmuksi eli karsitaan.

Muita tunnettuja karsintamenetelmiä ovat Quinlanin (1987) julkaisema luokittelutarkkuuteen pessimistisesti suhtautuva *pessimistic error pruning* -menetelmä ja Cestnikin ja Bratkon (1991) esittelemä todennäköisyyslaskentaan perustuva *m-probability pruning* -menetelmä. Karsinta-algoritmien kokeellisia vertailuja ovat tehneet esimerkiksi Mingers (1989) ja Esposito *et al.* (1993).

3.3. C4.5

C4.5 (Quinlan 1993) on enemmänkin ohjelmistokokonaisuus kuin yksittäinen ohjelma. Se on ID3:n laajennettu versio ja monissa tapauksissa on mahdotonta sanoa, mikä ominaisuus kuuluu ID3:een ja mikä C4.5:een. Tässä on esitelty C4.5:n tapa muodostaa päätöspuu, muuntaa päätöspuu joukoksi sääntöjä, karsia säännöt ja valita säännöistä parhaat lopulliseen säännöstöön.

3.3.1. Säännöstön muodostamisen päävaiheet

Kuten sanottu, C4.5 on ID3:n laajennettu versio. Siinä missä ID3:n toiminta päättyy päätöspuun muodostamiseen ja mahdollisesti sen karsintaan, C4.5 vielä muokkaa lopputulosta monin tavoin. C4.5:n toiminta koostuu seuraavista päävaiheista.

- Muodostetaan päätöspuu gain ratio -kriteeriä käyttäen
- Muutetaan puu joukoksi sääntöjä
- Karsitaan säännöt
- Ryhmitellään säännöt sen mukaan minkä luokan ne määrittelevät
- Päätetään, mitä osajoukkoa kunkin ryhmän säännöistä käytetään
- Järjestetään sääntöryhmät
- Muodostetaan default-sääntö
- Karsitaan säännöstö

Seuraavissa alakohdissa 3.3.2. - 3.3.9. jokaista yllä mainittua vaihetta käsitellään erikseen. Koska C4.5 muodostaa päätöspuun samoin kuin ID3, sitä ei enää uudelleen esitetä.

3.3.2. Valintakriteeri gain ratio

ID3-ohjelman yhteydessä kuvatun gain-kriteerin eräs heikkous on se, että se suosii moniarvoisia attribuutteja. Mitä enemmän arvoja attribuutti voi saada, sitä suuremmat mahdollisuudet sillä on tulla valituksi. Jos esimerkkiaineiston tapauksiin lisätään attribuutiksi tapauksen järjestysnumero, gain-kriteeri valitsee puun ensimmäiseksi ja ainoaksi päätössolmuksi juuri tämän järjestysnumeron vertaamisen. Näin muodostettu päätöspuu kyllä luokittelee harjoitusaineiston täysin oikein, mutta on kuitenkin täysin käyttökelvoton tuntemattomien tapausten luokittelussa. Quinlan (1988) esittää korjauksen tämän vinoutuman poistamiseksi. Määritellään suure, joka mittaa tuotetun informaation määrää jaettaessa esimerkkiaineisto attribuutin A mukaan

$$split\ info(A) = -\sum_{v} p(v) \log_2 p(v),$$

missä p(v) on todennäköisyys, että attribuutin A arvo v. Joten

$$gain\ ratio(A) = gain(A) / split\ info(A)$$

ilmaisee luokittelussa hyödyllisen informaation osuuden koko tuotetusta informaatiosta. Käytetään jälleen esimerkkinä edellisen kohdan taulukossa 3.1 esitettyä harjoitusaineistoa. Attribuutin *onko tp suluissa* arvo on kolmessa tapauksessa *on* ja 13 tapauksessa *ei*. Joten

$$split\ info(onko\ tp\ suluissa) = -\frac{3}{16}\log_2\frac{3}{16} - \frac{13}{16}\log_2\frac{13}{16} = 0.696\ .$$

ja edelleen

$$gain\ ratio(onko\ tp\ suluissa) = \frac{0.115}{0.696} = 0.165\ .$$

Alla oleva taulukko sisältää valintakriteerien gain ja gain ratio arvot kaikille harjoitusaineiston attribuuteille siinä vaiheessa, missä valitaan päätöspuulle juurisolmua, eli vaiheessa, jossa kaikki aineiston esimerkkitapaukset ovat laskennassa mukana.

attribuutti	*gain*	*split info*	*gain ratio*
Tp ryhmän numero	0.0009	0.954	0.0009
Onko tp suluissa	0.115	0.696	0.165
Edellinen merkki	0.427	2.311	0.185
Seuraava merkki	0.161	1.310	0.123

Taulukko 3.2. Valintakriteerien arvoja päätöspuun juurta valittaessa.

Taulukosta näkee, että myös gain ratio -kriteerin perusteella ensimmäiseksi solmuksi päätöspuuhun valitaan attribuutti *edellinen merkki*, mutta ero attribuuttiin *onko tp suluissa* on melko pieni. Gain-kriterin perusteella arvostellen *onko tp suluissa* on kuitenkin vasta kolmannella sijalla. Tämä on ymmärrettävää, koska attribuutilla *onko tp suluissa* on vain kaksi mahdollista arvoa, kun taas attribuutilla *edellinen merkki* on kuusi ja attribuutilla *seuraava merkki* viisi mahdollista arvoa ja gain-kriteeri tunnetusti suosii moniarvoisia attribuutteja. Päätöspuu, joka harjoitusaineistosta saadaan käyttämällä gain ratio -kriteeriä on kuitenkin identtinen gain-kriteeriä käyttämällä muodostetun puun kanssa (ks. kuva 3.2).

3.3.3. Päätöspuun muuntaminen sääntöjoukoksi

Päätöspuun muuntaminen joukoksi sääntöjä on yksinkertainen ja suoraviivainen tehtävä. Kuljetaan vain puun juuresta kaikkiin sen lehtisolmuihin vuorollaan. Jokainen tällainen reitti muodostaa yhden säännön, jonka ehto-osana on reitin varrella olevien solmujen nimeämiä attribuutteja koskevat testit ja päätösosana lehtisolmun nimeämä luokka. Kuvan 3.2. esittämästä päätöspuusta muodostuu seuraava sääntöjoukko.

```
JOS   edellinen merkki = v
      onko tp suluissa = on
NIIN  luokka on  on

JOS   edellinen merkki = v
      onko tp suluissa = ei
NIIN  luokka on  ei

JOS   edellinen merkki = ja
      onko tp suluissa = on
NIIN  luokka on  on

JOS   edellinen merkki = ja
      onko tp suluissa = ei
NIIN  luokka on  ei

JOS   edellinen merkki = im
NIIN  luokka on  on

JOS   edellinen merkki = ek
NIIN  luokka on  on

JOS   edellinen merkki = neg
      onko tp suluissa = on
NIIN  luokka on  on

JOS   edellinen merkki = neg
      onko tp suluissa = ei
NIIN  luokka on  ei

JOS   edellinen merkki = none
NIIN  luokka on  on
```

Kuva 3.3. Päätöspuu muunnettuna sääntöjoukoksi.

Sääntöjen lukumäärä on sama kuin päätöspuun lehtisolmujen lukumäärä, tässä tapauksessa yhdeksän kappaletta. Tuntemattomien tapausten luokittelutarkkuus säännöstöllä on tarkalleen sama kuin päätöspuulla, josta se on muodostettu, tietämys esitetään vain ihmisen kannalta helpommin ymmärrettävässä muodossa. Toinen etu, joka säännöstöksi muuntamisella saavutetaan on se, että säännöstön karsinta pystytään suorittamaan hienovireisemmin kuin päätöspuun karsinta. Jos päätöspuuta karsittaessa alipuu muutetaan lehtisolmuksi, se vaikuttaa jokaiseen alipuun haaraan (yhdistää ne). Sääntöjä karsittaessa taas päätös ehtojen poistamisesta tehdään jokaisen säännön kohdalla erikseen, joten joistakin samasta alipuusta muodostetuista säännöistä voidaan tietty ehto karsia ja joistakin säännöistä jättää sama ehto karsimatta.

3.3.4. Sääntöjen karsinta

Säännön karsiminen tarkoittaa säännön yhden tai useamman ehdon poistamista. Sääntö R voidaan esittää muodossa

jos A niin luokka on C

ja tätä yleisempi sääntö R’ muodossa

jos A’ niin luokka on C,

missä A’ on saatu poistamalla ehto X ehtojoukosta A. Jokainen harjoitusaineiston esimerkkitapaus, joka täyttää ehdot A’ joko kuuluu tai ei kuulu määritettävään luokkaan C, ja joko täyttää ehdon X tai ei täytä ehtoa X. Näin saadaan neljä ryhmää, joihin kuuluvien esimerkkitapausten lukumäärät voidaan ilmaista seuraavasti.

	Luokka C	*Muut luokat*
Täyttää ehdon X	Y_1	E_1
Ei täytä ehtoa X	Y_2	E_2

Alkuperäinen sääntö R hyväksyy $Y_1 + E_1$ tapausta, joista E_1 luokitellaan virheellisesti ja karsittu sääntö R’ hyväksyy kaikki neljän ryhmän tapaukset, joista luokitellaan virheellisesti $E_1 + E_2$ kappaletta. Tästä saadaan säännön R luokitteluvirheelle arvo $E_1/(Y_1 + E_1)$ ja vastaavasti säännön R’ luokitteluvirheelle arvo $(E_1 + E_2) / (Y_1 + Y_2 + E_1 + E_2)$. On kuitenkin epätodennäköistä, että luokitteluvirheet eivät kasva, kun sääntöjä käytetään tuntemattomien tapausten luokitteluun. C4.5 käyttääkin huomattavasti pessimistisempää arviota luokitteluvirheelle.

Kun sääntö luokittelee N tapausta, joista E kappaletta väärin, voidaan tämä ilmaista tilastotieteen termein: E tapahtumaa N:llä yrityksellä. Jos harjoitusaineistoa ajatellaan otoksena, voidaan miettiä, mikä on tapahtuman (virheen) todennäköisyys koko populaatiossa tapauksia, jotka sääntö luokittelee. Virheen todennäköisyyttä ei voida laskea suoraan, mutta se noudattaa binomijakaumaa, josta voidaan annetulla luottamustasolla CF määrittää sille yläraja. Tälle ylärajalle käytetään merkintää $U_{CF}(E,N)$. C4.5:n oletusarvoisesti käyttämä luottamustaso on 25%, joten yläraja virheen todennäköisyydelle saadaan ratkaisemalla p yhtälöstä

$$\sum_{i=0}^{E} \binom{N}{i} p^i (1-p)^{N-i} = 0.25,$$

ja tällöin $U_{25\%}(E,N) = p$.

Sääntöä karsittaessa verrataan näitä pessimistisiä yläraja-arviota. Virhearvio säännölle R on $U_{25\%}(E_1, Y_1 + E_1)$ ja virhearvio säännölle R’ on $U_{25\%}(E_1 + E_2, Y_1 + Y_2 + E_1 + E_2)$. Jos säännön R’ virhearvio ei ole suurempi kuin säännön R, ehto X karsitaan.

Ehdoista voidaan poistaa useampia kuin yksi. C4.5 toteuttaa tämän ahneella (*greedy*) haulla, eli se ei käy läpi kaikkia mahdollisia ehtojoukon osajoukkoja, vaan poistaa aina ehdoista sen, jonka poistaminen eniten pienentää säännön virhearviota.

Näin jatketaan kunnes yhdenkään ehdon poistaminen ei paranna säännön luokittelutarkkuutta (tai säännöstä on kaikki ehdot poistettu). Esimerkiksi taulukon 3.1. harjoitusaineistosta muodostetusta päätöspuusta saatiin sääntö

JOS edellinen merkki = v
onko tp suluissa = on
NIIN luokka on on

joka kattaa yhden aineiston tapauksista ja luokittelee sen oikein. Pessimistinen virhearvio tälle säännölle on $U_{25\%}(0,1) = 75\%$. Alla olevassa taulukossa on esitetty virhearviot säännölle, kun ehdoista molemmat vuorollaan poistetaan.

Poistettu ehto	Y1 + Y2	E1 + E2	Pessimistinen virhearvio
edellinen merkki = v	3	0	37%
onko tp suluissa = on	3	2	93%

Taulukko 3.3. Ehdot, jotka säännöstä voidaan karsia.

Kuten taulukosta nähdään pessimistinen arvio säännön luokitteluvirheelle on pienin ilman ehtoa *edellinen merkki = v*, joten ko. ehto poistetaan. Tämän jälkeen pitää vielä laskea virhearvio säännölle, josta on poistettu viimeinenkin ehto eli *onko tp suluissa = on*. Tässä tapauksessa sääntö siis kattaa koko harjoitusaineiston ja luokittelee väärin viisi tapausta, joten virhearvio on $U_{25\%}(5,16) = 43\%$. Koska ehdon poistaminen kasvattaa luokitteluvirhettä, sitä ei poisteta. Karsinnan tuloksena on siis sääntö

JOS onko tp suluissa = on
NIIN luokka on on

Esimerkin kaikki säännöt karsinnan jälkeen löytyvät kuvasta 3.4. Nähdään, että alkuperäisistä yhdeksästä säännöstä on jäänyt jäljelle seitsemän. Kahdesta säännöstä on karsiutunut ehto-osa kokonaisuudessaan, joten säännöt on voitu poistaa. Jäljellä olevista säännöistä kolme on identtisiä keskenään, joten niistäkin voidaan kaksi poistaa luokittelutarkkuuden kärsimättä.

```
JOS  onko tp suluissa = on
NIIN  luokka on  on

JOS  edellinen merkki = v
       onko tp suluissa = ei
NIIN  luokka on  ei

JOS  onko tp suluissa = on
NIIN  luokka on  on

JOS  onko tp suluissa = ei
NIIN  luokka on  ei
```

```
JOS  onko tp suluissa = on
NIIN  luokka on  on

JOS  edellinen merkki = neg
       onko tp suluissa = ei
NIIN  luokka on  ei

JOS  edellinen merkki = none
NIIN  luokka on  on
```

Kuva 3.4. Säännöstö karsinnan jälkeen.

3.3.5. Sääntöjen ryhmittely

Karsitut säännöt eivät ole välttämättä toisiaan poissulkevia kuten karsimattoman säännöstön säännöt ovat. Useampaa kuin yhtä sääntöä voi käyttää joidenkin tuntemattomien tapausten luokitteluun. Koska luokittelu ei siis ole yksiselitteistä on tehtävä päätös, mitä tilanteeseen sopivista säännöistä kulloinkin käytetään.

C4.5 järjestää säännöt ryhmiin sen mukaan, mitä luokkaa ne määrittävät. Ryhmittelyn ansiosta säännöstö on ihmisnäkökulmasta järkevämpi ja helpompi ymmärtää. Toinen etu on se, että samaa luokkaa kuvaavien sääntöjen keskinäisellä järjestyksellä ei ole merkitystä. Sääntöryhmien järjestys on kuitenkin ratkaistava, samoin kuin se, mitkä säänöistä kuhunkin ryhmään otetaan mukaan.

3.3.6. Ryhmän sääntöjen valinta

Jotakin luokkaa C kuvaavan sääntöjoukon tehokkuus voidaan ilmaista sen avulla, kuinka monta harjoitusaineiston tapauksista, jotka eivät kuulu luokkaan C se hyväksyy (väärät positiiviset), ja kuinka monta tapauksista, joita se ei hyväksy todellisuudessa kuuluu luokkaan C (väärät negatiiviset). C4.5 päättää sääntöjoukon arvon käyttämällä Minimum Description Length (MDL) -periaatetta. MDL-periaatteen käyttöä päätöspuiden ja sääntöjoukkojen yhteydessä on tarkemmin käsitelty Quinlanin ja Rivestin (1989) artikkelissa.

MDL-periaatteen käyttö tässä yhteydessä on seuraava: Lähettäjällä ja vastaanottajalla on molemmilla tiedot harjoitusaineiston esimerkkitapauksista, mutta vastaanottajan tiedoista puuttuu tapausten luokitus. Lähettäjän pitää välittää tämä tieto lähettämällä teoria, jonka mukaan tapaukset luokitellaan sekä tiedot poikkeustapauksista (väärät positiiviset ja väärät negatiiviset). Lähettäjä voi lähettää yksinkertaisen teorian (pienen sääntöjoukon) joka sisältää huomattavan määrän poikkeustapauksia tai monimutkaisemman teorian joka sisältää vähemmän poikkeustapauksia. MDL-periaatteen mukaan paras teoria on se, joka (yhdessä poikkeustapausten kanssa) voidaan koodata mahdollisimman lyhyesti. Lyhyesti esitettynä koodipituudet saadaan seuraavasti:

- Säännön koodipituus on sen ehtojen koodipituuksien summa. Järjestyksellä, missä ehdot lähetetään ei ole merkitystä. Jos ehtoja on *x* kappaletta ne voidaan järjestää *x!* eri tavalla. Summasta pitää vähentää tällöin $log_2(x!)$ bittiä.
- Sääntöjoukon koodipituus on sen sisältämien sääntöjen koodipituuksien summa Myöskään sääntöjen järjestyksellä ei ole merkitystä, joten summasta vähennetään samoin laskettu osuus kuin edellisessä kohdassa.
- Jos sääntöjoukko kattaa *r* tapausta sääntöjoukosta, jonka koko on *n* tapausta ja väärien positiivisten määrä on *fp* ja väärien negatiivisten määrä *fn* niin poikkeusten koodaamiseen tarvittavien bittien määrä on

$$\log_2\binom{r}{fp} + \log_2\binom{n-r}{fn}.$$

Käytännössä koodaus aliarvioi poikkeusten koodaukseen tarvittavien bittien määrää. C4.5 ottaa tämän huomioon laskemalla tarvittavien bittien määrän painotettuna summana

$$poikkeuset + W \times teoria,$$

missä keroin W on pienempi kuin yksi. Oletusarvo kertoimelle on 0.5.

Tarkoitus on siis löytää luokkaa C kuvaavasta sääntöjoukosta osajoukko S, joka voidaan koodata mahdollisimman lyhyesti. Hill-climbing -tyyppinen ahne haku, jossa sääntöjä poistetaan joukosta lopullisesti yksi kerallaan ei toimi tässä yhteydessä. Sen sijaan käsitellään kaikki mahdolliset sääntöjoukon osajoukot, jos niitä ei ole liikaa, tai käytetään simuloituun jäähdytykseen (*simulated annealing*) perustuvaa menetelmää, jos mahdollisia osajoukkoja on paljon. Mahdollisten osajoukkojen lukumäärä kasvaa nope-

asti kun sääntöjoukon koko kasvaa. Jos sääntöjoukossa on n sääntöä, mahdollisia osajoukkoja on tällöin 2^n kappaletta.

Simuloitua jäähdytystä käytettäessä valitaan toistuvasti sääntöjoukosta satunnainen sääntö ja päätetään lisätäänkö se osajoukkoon S (jos se ei jo kuulu joukkoon) tai poistetaanko se osajoukosta S (jos se kuuluu joukkoon). Tämä säännön lisäys tai poisto aiheuttaa muutoksen ΔB säännöstön ja poikkeusten koodaamiseen tarvittavien bittien määrässä. Jos muutos ΔB on negatiivinen eli koodi lyhenee, säännön lisäys tai poisto hyväksytään automaattisesti. Jos taas ΔB on positiivinen eli koodipituus kasvaa, osajoukon S muutos hyväksytään todennäköisyydellä $e^{-\Delta B / K}$, missä K on kerroin, jota asteittain pienennetään. Menetelmällä löydetään osajoukko S, joka MDL-periaatteen mukaan ajatellen on lähellä parasta mahdollista osajoukkoa.

Esimerkkinä voidaan tarkastella kuvassa 3.4. esitettyä karsittua säännöstöä. Säännöstöstä löytyy kaksi erilaista sääntöä luokalle *on*:

Sääntö 1	Sääntö 2
JOS edellinen merkki = none	JOS onko tp suluissa = on
NIIN luokka on on	NIIN luokka on on

Säännön 1 koodaukseen tarvitaan 4.164 bittiä ja säännön 2 koodaukseen 2.549 bittiä. Koska säännöt voi järjestää 2! eri tavalla teorian koodaus vaatii 4.164 + 2.549 – $\log_2(2)$ bittiä. Koska säännöt kattavat yhdeksän tapausta esimerkkiaineiston 16:sta tapauksesta ja luokittelevat kaikki yhdeksän tapausta oikein (väärät positiiviset = 0) ja hylkäävät kaksi tapausta, jotka kuuluvat luokkaan *on* (väärät negatiiviset = 2), on poikkeustapausten koodaukseen tarvittavien bittien määrä

$$\log_2\binom{9}{0} + \log_2\binom{7}{2} = 4.392\,.$$

Koska sääntöjä on kaksi, niistä voi muodostaa neljä erilaista joukkoa. Taulukossa 3.4. on esitetty koodipituudet näille neljälle mahdolliselle sääntöjoukolle. Vaadittavan koodin pituus on pienin kun molemmat säännöt ovat mukana säännöstössä, joten kumpaakaan sääntöä ei poisteta.

Sääntöjoukko	Teoria	Väärät positiiviset	Väärät negatiiviset	Poikkeukset	Yhteensä
-	0	5	0	12.093	12.093
{1}	4.164	0	5	7.977	10.059
{2}	2.549	0	8	10.330	11.604
{1,2}	5.713	0	2	4.392	7.249

Taulukko 3.4. Koodaukseen tarvittavien bittien määrä eri sääntöjoukoille.

Vastaavasti menetellään luokan *ei* määrittävien sääntöjen kanssa. Sääntöjä on alkujaan kolme, joista yksi MDL-tarkastelun perusteella poistetaan. Kuva 3.5. sisältää jäljelle jääneen säännöstön.

```
JOS   edellinen merkki  = none
NIIN  luokka on  on

JOS   onko tp suluissa = on
NIIN  luokka on  on

JOS   edellinen merkki = neg
      onko tp suluissa = ei
NIIN  luokka on  ei

JOS   edellinen merkki  = v
      onko tp suluissa = ei
NIIN  luokka on  ei
```

Kuva 3.5. MDL-periaatteen mukaan valitut säännöt.

3.3.7. Sääntöryhmien järjestys

Jokaista luokkaa kuvaavat sääntöryhmät hyväksyvät usein tapauksia, jotka todellisuudessa kuuluvat johonkin toiseen luokkaan (väärät positiiviset). Sääntöryhmien keskinäisestä järjestyksestä päätettäessä ratkaiseva tekijä on juuri ryhmien väärin hyväksymien tapausten määrä. Sääntöryhmät, jotka virheellisesti hyväksyvät monia tapauksia on järkevää sijoittaa säännöstön loppuun; säännöstön aiemmat sääntöryhmät saattavat tällöin luokitella oikein ainakin osan näistä tapauksista ennen kuin ne tulevat väärinluokitelluiksi.

C4.5 järjestää sääntöryhmät siten, että se ryhmä, jonka väärien positiivisten luokitusten määrä harjoitusaineiston tapauksista on pienin sijoitetaan ensimmäiseksi, toiseksi pienin toiseksi ja niin edelleen.

Esimerkkisäännöstön molempia luokkia kuvaavat sääntöryhmät ovat tässä suhteessa samanarvoisia: kumpikaan ei virheellisesti hyväksy yhtäkään harjoitusaineiston tapausta. Ryhmien järjestyksellä ei siis tässä tapauksessa ole merkitystä.

3.3.8. Default-sääntö

Koska päätöspuusta muodostettua sääntöjoukkoa on aiemmissa vaiheissa yksinkertaistettu ja karsittu, se ei välttämättä kata kaikkia mahdollisia tapauksia, jotka sen pitäisi luokitella. Joukkoon pitää lisätä default-sääntö, jota sovelletaan tapauksissa, joissa mitään muuta sääntöä ei voi soveltaa. Koska default-sääntöä käytetään vain muiden sääntöjen hylkäämissä tapauksissa, sääntö on järkevää valita luokittelemaan tapaukset siihen luokkaan, joka on enemmistönä näissä hylätyissä tapauksissa.

Kuvan 3.5. säännöt jättävät luokittelematta kolme taulukon 3.1. harjoitusaineiston esimerkeistä. Näistä kaksi kuuluu luokkaan *on* ja yksi luokkaan *ei*. Default-säännöksi valitaan

JOS ei mikään aiemmista
NIIN luokka on on

3.3.9. Koko säännöstön karsinta

Kun sääntöryhmien järjestys on päätetty ja säännöstöön lisätty default-sääntö, koko säännöstö testataan vielä kerran. Säännöstöstä poistetaan jokainen sääntö vuorollaan, luokitellaan harjoitusaineiston tapaukset ja lasketaan luokitteluvirheiden määrä. Jos jonkin säännön poistaminen ei kasvata luokitteluvirhettä, se poistetaan säännöstöstä. Näin jatketaan kunnes luokitteluvirhe kasvaa minkä tahansa säännön poistamisen seurauksena.

Esimerkkisäännöstön tapauksessa molemmat luokan *on* määrittävistä säänöistä poistetaan, koska default-sääntö joka tapauksessa luokittelee samaan luokkaan kaikki tapaukset, jotka luokan *ei* määrittävät säännöt hylkäävät. Alla oleva kuva sisältää esimerkkinä käytetyn säännöstön lopullisessa muodossaan.

JOS edellinen merkki = neg
onko tp suluissa = ei
NIIN luokka on ei

JOS edellinen merkki = v
onko tp suluissa = ei
NIIN luokka on ei

JOS ei mikään aiemmista
NIIN luokka on on

Kuva 3.6. Lopullinen säännöstö.

3.4. Geneettinen algoritmi

Geneettiset algoritmit perustuvat simuloituun evoluutioon. Yleensä populaatio koostuu hypoteeseista, jotka määrittelevät opittavan aihealueen (DeJong *et al.* 1993). Hypoteesit kamppailevat keskenään elintilasta, risteytyvät ja lisääntyvät, huonot hypoteesit kuolevat ja vahvat jäävät henkiin. Lopulta valitaan hypoteeseista paras. Toinen tapa (Holland, 1986), mitä myös tässä työssä on käytetty, on valita populaatioksi joukko yksittäisiä sääntöjä, jotka yhdessä muodostavat opittavan aihealueen määritelmän. Tällöin säännöt kilpailevat keskenään, risteytyvät ja lisääntyvät tai kuolevat. Lopulta kaikki eloonjääneet säännöt muodostavat aihealueen määritelmän.

Algoritmin päävaiheet ovat seuraavat:

- Muodostetaan alkuperäinen populaatio
- Muodostetaan risteyttämällä joukko uusia sääntöjä
- Lisätään mutaatiot
- Testataan säännöt ja valitaan niistä parhaat, loput kuolevat

Kolmea viimeistä vaihetta toistetaan kunnes säännöstö saavuttaa halutun tarkkuuden eli kunnes säännöt luokittelevat riittävän määrän harjoitusaineiston tapauksista oikein. Lopetusehtona voidaan käyttää myös muodostettavien uusien sukupolvien lukumäärää.

Aina uuden sukupolven vakiintumisen jälkeen säännöstö järjestetään lajeittain parhaaseen mahdolliseen järjestykseen. Lisäksi säännöstöön lisätään default-sääntö, jota käytetään siinä tapauksessa, että mikään säännöstön sääntö ei kata luokiteltavaa tapausta. Lopuksi säännöstö vielä karsitaan siten, että siitä poistetaan säännöt, joiden poistaminen ei kasvata luokitteluvirhettä harjoitusaineiston tapauksia luokiteltaessa. Nämä kolme viimeistä vaihetta toteutetaan samojen periaatteiden mukaan kuin edellisessä kohdassa kuvatun C4.5:n yhteydessä on tehty.

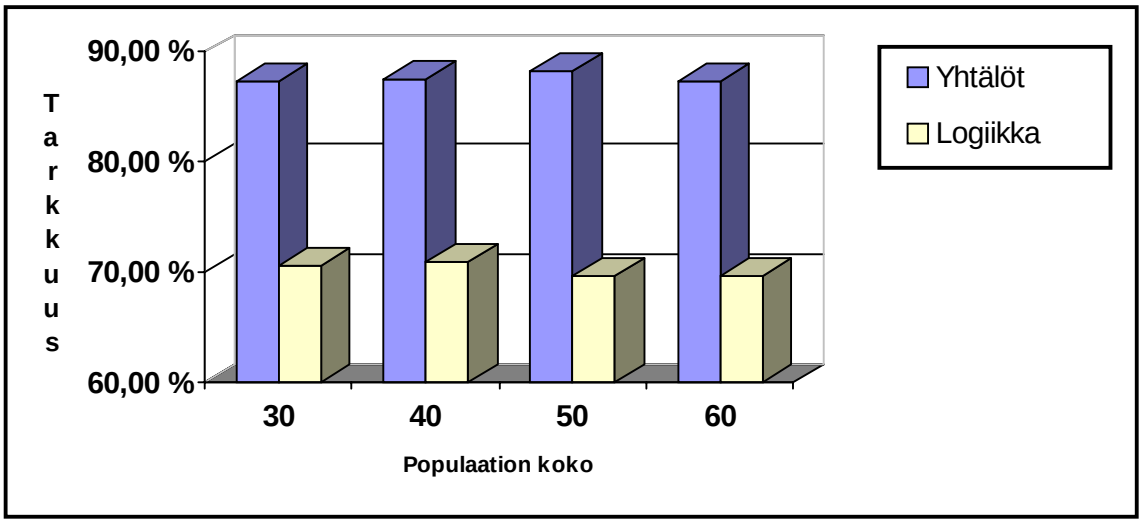
3.4.1. Alkuperäinen populaatio

Eräs mahdollisuus muodostaa alkuperäinen sääntö-populaatio on käyttää aiemmin kuvatun ID3-ohjelman oppimaa päätöspuuta, joka muutetaan säännöstöksi. Kokeilut kuitenkin osoittavat, että tämä ei ole kovinkaan hyvä lähtökohta evoluutiolle. ID3:n tuottama säännöstö on jo alunperin tiiviissä muodossa ja evoluutio karsii sitä lisää, joten monet hyvät säännöt kuolevat sukupuuttoon ja usein henkiin jää vain yhden säännön

variaatioita. (Tilanne lienee samantapainen kuin biologisessa evoluutiossa suljetuissa yhteisöissä tapahtuva geenien väheneminen). Näin käy varsinkin käytettäessä arvontaa yksilöiden eloonjäämistä ratkaistaessa (ks. kohta 3.4.4.). Myös kiinteän eloonjäämisrajan käyttäminen johtaa usein heikkoon lopputulokseen. Paras tulos ID3:n tuottamasta säännöstöstä liikkeelle lähdettäessä saavutetaan käyttämällä kiinteän kokoista populaatiota ja suurehkoa mutaatiotiheyttä.

Toinen tapa on tuottaa alkuperäinen populaatio satunnaisesti. Muodostetaan haluttu määrä sääntöjä arpomalla jokaisen säännön ehto-osaan 1 – N ehtoa (N on esimerkkiaineiston attribuuttien lukumäärä) ja joko harjoitusaineiston avulla päättelemällä tai satunnaisesti valitsemalla määrätään luokka jota kukin sääntö edustaa. Säännöt jaetaan edustamansa luokan mukaan lajeiksi. Eri lajien yksilöt eivät voi risteytyä keskenään. Tämän aloitustavan ongelma on liiallisesta satunnaisuudesta johtuva epävarmuus: on epätodennäköistä, että samaan sääntöön tulee arvottua enemmän kuin yksi hyvä ominaisuus. Tämän seurauksena monet hyvät ehdot katoavat ennen kuin ehtivät yleistyä ja hyvän lopputuloksen saavuttaminen on vähintäänkin epävarmaa.

Tämän työn yhteydessä on käytetty menetelmää, jonka avulla mahdollisimman suuri geeniaines saadaan evoluution käyttöön. Aluksi muodostetaan kaikki yhden ehdon säännöt, ja valitaan näiden luokka ehdon toteuttavien esimerkkitapausten enemmistöluokan mukaan. Alussa populaation kokoa ei siis rajoiteta, vasta ensimmäisen risteytymisvaiheen jälkeen populaatiota karsitaan valitun kriteerin mukaan. Tämä menetelmä johtaa kahta edellä mainittua menetelmää varmemmin ja nopeammin hyvään lopputulokseen.



Kuva 3.7. Alkuperäisen populaation koon vaikutus lopulliseen ennustustarkkuuteen.

Arvottujen sääntöjen lukumäärä vaikuttaa suuresti lopputulokseen, mitä enemmän evoluutiolla on materiaalia käytettävissään, sitä varmemmin se muovaa siitä elinkelpoisia yksilöitä (hyviä sääntöjä). Alkuperäisen populaation liian suuresta koosta on myös haittoja. Monet heikohkot yksilöt, joilla on kuitenkin myös hyviä ominaisuuksia (ehtoja) jäävät helposti suurten massojen jalkoihin ja nämä ominaisuudet kuolevat sääntöjen mukana ennen kuin ehtivät yleistyä. Toisaalta suurten populaatioiden evoluution simuloiminen on hidasta ja vaatii laitteistolta melkoisia resursseja.

Kuva 3.7. esittää alkuperäisten sääntöjen lukumäärän vaikutusta siihen, kuinka hyvin lopullinen, evoluution läpikäynyt säännöstö kykenee luokittelemaan tuntemattomia tapauksia. Vertailussa on käytetty samoja esimerkkiaineistoja kuin lopullisissa vertailuissa ja lopulta kumppaniagenttien opettamisessa. Aineistot on satunnaisesti puolitettu opetus- ja testausjoukkoihin. Opetusjoukkoa on käytetty evoluution aikana yksilöiden eloonjäämiskyvyn mittaamiseen ja testausjoukon avulla on mitattu lopullisen säännöstön luokittelutarkkuus. Kuvasta nähdään, että alkuperäisen populaation koon kasvattaminen tiettyyn rajaan saakka parantaa lopullisen säännöstön kykyä luokitella tuntemattomia tapauksia.

3.4.2. Risteytys

Uuden sukupolven sääntö saadaan aikaan risteyttämällä kaksi jo olemassa olevaa samaa luokkaa edustavaa sääntöä. Risteytyksessä sääntöjen ehto-osat yhdistetään jonkin periaatteen mukaan. Tässä risteytys suoritetaan tavoilla, jotka löyhästi seuraavat Gregor Mendelin (1866) perinnöllisyysteorioita.

Aluksi harjoitusaineiston esimerkkitapauksista lasketaan luokittain eri attribuuttien arvojen esiintymislukumäärät. Kun käytetään taulukossa 3.1 esitettyä esimerkkiaineistoa saadaan taulukossa 3.5 kuvattu jakauma eri attribuuttien arvoille. Alkuperäistä populaatiota muodostettaessa lisätään jokaiseen ehtoon painoarvot testattavan attribuutin arvoille laskettujen esiintymislukumäärien avulla. Risteytettäessä kaksi sääntöä, jotka sisältävät samaan attribuuttiin liittyvän ehdon, lasketaan ehtoon liittyvät painoarvot alkioittain yhteen ja attribuutin arvoksi valitaan se, jonka painoarvo on suurin.

Luokka	Tp ryhmän numero		Onko tp suluissa		Edellinen merkki						Seuraava merkki				
	1	2	on	ei	v	ja	im	ek	neg	none	v	ja	im	ek	none
on	4	7	3	8	1	1	1	1	1	6	1	1	1	1	7
ei	2	3	0	5	2	1	0	0	2	0	0	0	0	0	5

Taulukko 3.5. Harjoitusaineiston jakautuminen eri attribuuttien arvoille.

Ajatellaan, että alkuperäistä populaatiota muodostettaessa on arvottu säännöt:

JOS edellinen merkki = v
onko tp suluissa = on
NIIN luokka on on

JOS edellinen merkki = ja
onko tp suluissa = ei
NIIN luokka on on

Painoarvot ensimmäisessä säännössä attribuutille *edellinen merkki* ovat (1,0,0,0,0,0) ja attribuutille *onko tp suluissa* (3,0). Vastaavat arvot toiselle säännölle ovat (0,1,0,0,0,0) ja (0,8). Kun säännöt risteytetään attribuuttien painoarvot lasketaan alkioittain yhteen. Risteytetyn säännön painoarvot ovat siis attribuutille *edellinen merkki* (1,1,0,0,0,0) ja attribuutille *onko tp suluissa* (3,8). Ensimmäisen ehdon kohdalla kaksi ensimmäistä arvoa ovat painoarvoltaan yhtä suuria, joten valinta suoritetaan arpomalla. Toisen ehdon kohdalla valitaan arvo *ei*. Saadaan siis sääntö:

JOS edellinen merkki = v (1,1,0,0,0,0)
onko tp suluissa = ei (3,8)
NIIN luokka on on

Jos tämä sääntö edelleen risteytetään kahden alkuperäisten säännön kanssa saadaan säännöt:

JOS edellinen merkki = v (2,1,0,0,0,0)
onko tp suluissa = ei (6,8)
NIIN luokka on on

JOS edellinen merkki = ja (1,2,0,0,0,0)
onko tp suluissa = ei (3,16)
NIIN luokka on on

Jos risteytettävistä säännöistä vain toinen sisältää tiettyä attribuuttia koskevan ehdon, päätetään tasapuolisen arvonnan avulla otetaanko se mukaan uuden sukupolven sääntöön.

Toinen tapa suorittaa risteytys mallintaa suvullista lisääntymistä. Jokainen säännön ehto sisältää kaksi mahdollista arvoa (*alleelia*), joista risteytyksen yhteydessä toinen valitaan arpomalla. Seuraamalla edelleen perinnöllisyysteoriaa valitaan kullekin

attribuutille dominoiva arvo. Luonnollisin tapa on valita attribuutin dominoivaksi arvoksi se, joka on kyseessä olevan luokan (lajin) tapauksessa yleisin. Risteytyksen yhteydessä uuden säännön jokainen ehto perii molemmilta vanhemmiltaan jommankumman näiden vastaavien ehtojen alleeleista. Jos näistä perityistä alleeleista toinen on dominoiva, sen edustama arvo valitaan. Jos kumpikaan perityistä arvoista ei ole dominoiva, ratkaisu näiden välillä tehdään arpomalla.

Ajatellaan esimerkiksi, että risteytettävänä on alla olevat säännöt. Jokaisen ehdon jälkeen on suluissa ilmoitettu attribuutin 'piilevät' arvot.

```
JOS   edellinen merkki = v  (v,neg)
      onko tp suluissa = ei  (on,ei)
NIIN  luokka on  on
```

```
JOS   edellinen merkki = ja  (ja,im)
      onko tp suluissa = ei  (ei,ei)
NIIN  luokka on  on
```

Oletetaan, että arpa suosii aina ensimmäistä alleelia. Tällöin saadaan risteytetyksi seuraava sääntö

```
JOS   edellinen merkki = v  (v,ja)
      onko tp suluissa = ei  (on,ei)
NIIN  luokka on  on
```

Ehdon *onko tp suluissa* arvoksi tulee varmasti *ei,* koska *ei* on attribuutin dominoiva arvo.

Jos risteytettävistä säännöistä vain toinen sisältää tiettyä attribuuttia koskevan ehdon, päätetään tasapuolisen arvonnan avulla otetaanko se mukaan uuden sukupolven sääntöön vai ei. Mukaan otettaessa se perii molemmat alleelit vanhemmaltaan, jolta se perii ehdonkin.

Yhtälö- ja logiikka-aineistoilla suoritettujen vertailujen perusteella yllä mainitut risteytystavat toimivat jokseenkin yhtä hyvin. Yhtälöiden yhteydessä alleelimenetelmä tuottaa hieman paremman tuloksen kun taas logiikan yhteydessä painoarvomenetelmä on hieman parempi.

3.4.3. Lisääntyjät

Populaation kehitykseen vaikuttaa myös se, millä osalla populaatiota on mahdollisuus tuottaa jälkeläisiä. Yksinkertaisin tapa on antaa jokaiselle populaation yksilölle yhtä

suuri lisääntymismahdollisuus. Vanhasta sukupolvesta arvotaan kaksi yksilöä, jotka risteyttämällä tuotetaan yksi jälkeläinen. Arvontaa ja risteystä toistetaan kunnes jälkeläisten lukumäärä on yhtä suuri kuin edellisen sukupolven yksilöiden lukumäärä. Jotkin yksilöt voivat arpaonnesta riippuen tuottaa useampiakin kuin yhden jälkeläisen, huonoonnisemmat eivät välttämättä ainoatakaan. Sukupolvet yhdistetään ja yksilöiden elinkelpoisuus ratkaistaan jollakin seuraavassa kappaleessa kuvatulla tavalla.

Toinen tapa on antaa lisääntymismahdollisuus vain lajien parhaille yksilöille. Olemassa olevan sukupolven yksilöistä valitaan elinkelpoisimmat ja näitä satunnaisesti risteyttämällä tuotetaan uuden sukupolven edustajat. Tämä tapa monissa tilanteissa johtaa melko nopeaan kehitykseen, mutta se saattaa johtaa myös joidenkin hyvien ominaisuuksien (ehtojen) täydelliseen katoamiseen.

3.4.4. Eloonjääminen

Populaation yksilöiden (sääntöjen) arvo mitataan sen mukaan, kuinka hyvin ne osaavat luokitella harjoitusaineiston esimerkkitapauksia. Koska säännöt kattavat eri määrän harjoitusaineiston tapauksista, yksinkertainen oikeinluokiteltujen tapausten prosenttiosuus kaikista luokitelluista tapauksista ei ole kovinkaan hyvä mitta sääntöjen elinkelpoisuudelle. Jokin sääntö saattaa kattaa vain yhden tapauksen ja luokitella sen oikein, kun taas toinen voi kattaa kymmenen tapausta ja luokitella niistä yhdeksän oikein. Edellä mainitun laskutavan perusteella säännöistä ensimmäinen siis saisi paremman arvosanan, vaikka on luultavaa, että jälkimmäinen sääntö toimii tuntemattomien tapausten luokittelussa huomattavasti paremmin.

Arvio säännön eloonjäämiskyvylle lasketaan edellisessä kohdassa esitellyn C4.5:n käyttämän sääntöjen virhearvion avulla. Jos sääntö luokittelee N tapausta, joista E kappaletta väärin, saadaan sen eloonjäämiskyvylle arvo

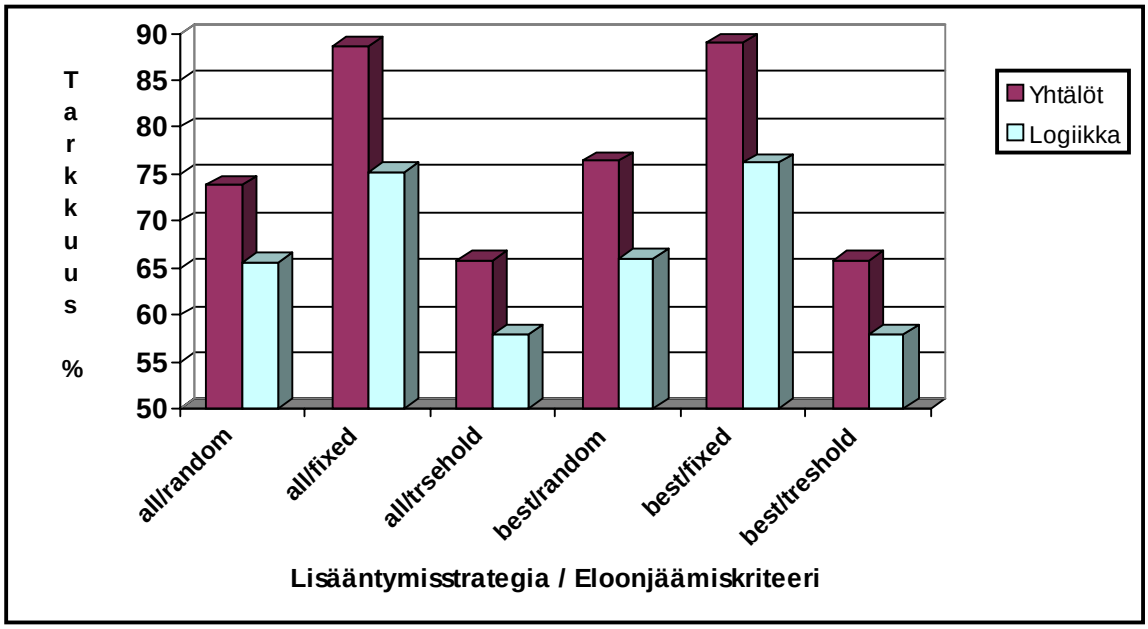
$$Fitness = 1 - U_{25\%}(E, N),$$

missä $U_{25\%}(E,N)$ on siis yläraja virheen todennäköisyydelle 25 % luottamustasolla tuntemattomia tapauksia luokiteltaessa (ks. kohta 3.3.4). Fitness-arvot ovat siis aina välillä [0,1].

Populaation yksilöiden eloonjääminen ratkaistaan tämän Fitness-arvon perusteella jollakin seuraavista tavoista:

- Kiinteän kokoinen populaatio. Yksilöt järjestetään Fitness-arvonsa mukaiseen järjestykseen, joista X kpl parasta jätetään eloon
- Satunnainen eloonjääminen: Eloonjääminen ratkaistaan arpomalla siten, että todennäköisyys eloonjäämiselle on 0.5 + Fitness/2.
- Kiinteä eloonjäämisraja: Kaikki yksilöt, joiden Fitness-arvo ylittää määrätyn rajan jäävät henkiin. Raja kasvaa evoluution edetessä sukupolvi sukupolvelta, eli yksilöiden oletetaan kehittyvän.

Kuvassa 3.8. on esitetty eri eloonjäämistapoja ja lisääntymisstrategioita käyttämällä saavutetut luokittelutarkkuudet yhtälö- ja logiikka-aineistojen esimerkkitapauksille. Kiinteän rajan (*treshold*) käyttäminen eloonjäämiskriteerinä on usein ongelmallista: pienestä raja-arvosta seuraa ylikansoitus, hieman suuremmasta joukkotuho. Evoluution alussa säännöt eivät yleensä saavuta kovinkaan korkeita fitness-arvoja, joten raja-arvon pitäisi olla pieni, loppuvaiheessa taas pieni arvo ei karsi populaatiota riittävästi. Rajaa asteittain kasvatettaessa pitäisi taas tietää missä evoluution vaiheessa säännöt ovat riittävästi kehittyneet. Arvonta (*random*) eloonjäämisestä päätettäessä toimii kohtalaisesti, joskin melko usein sattuu populaation joukkotuhoja. Pitämällä populaatio kiinteän kokoisena (*fixed*) saavutetaan parhaat tulokset. Tämä eloonjäämiskriteeri toimii hyvin annettaessa kaikkien (*all*) yksilöiden lisääntyä sekä jätettäessä lisääntyminen vain parhaiden (*best*) yksilöiden tehtäväksi.



Kuva 3.8. Eloonjäämiskriteerin ja lisääntymisstrategian vaikutus lopulliseen luokittelutarkkuuteen.

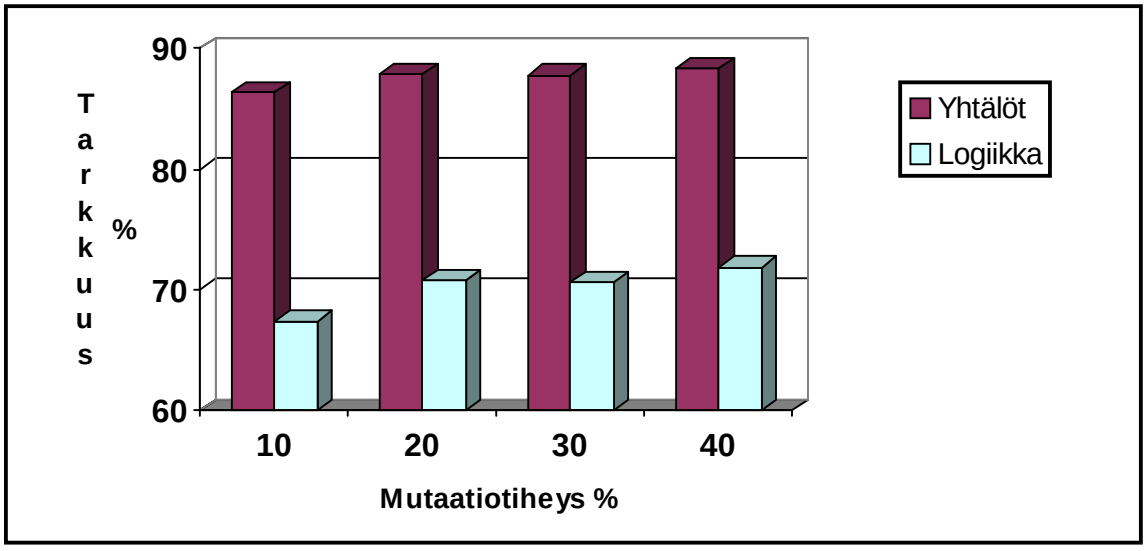
3.4.5. Mutaatiot

Myös perimässä tapahtuvat satunnaiset virheet eli mutaatiot saattavat olla hyödyllisiä lajin kehitykselle. Kun lajien yksilöt ovat tiettyä luokkaa kuvaavia sääntöjä mutaatiot voidaan toteuttaa seuraavilla tavoilla:

- Vaihdetaan jonkin ehdon attribuutin arvo. (Myös arvojen painot tai alleelit muuttuvat).
- Poistetaan kokonainen ehto.
- Lisätään kokonainen ehto.

Mutaatiot toteutetaan siten luonnosta poikkeavalla tavalla, että myös perimältään virheetön yksilö säilytetään, toisin sanoen risteytyksessä sattuvan mutaation tuloksena on kaksi uutta yksilöä: virheetön ja mutaation muuntama. Tällä varmistetaan se, että mutaatiot eivät aiheuta hyvien ominaisuuksien katoamista. Mutaatiot ovat siis säännöstön kehityksen kannalta joko hyödyllisiä tai yhdentekeviä, eivät missään tapauksessa haitallisia.

Tuotettavien mutaatioiden tapahtumatiheys voidaan valita (0 – 100 %). Jokaista mutaatiota toteutettaessa suoritetaan arvonta yllä mainittujen tapojen välillä siten, että attribuutin arvon vaihtumisen todennäköisyys on 50 %, ehdon poistamisen todennäköisyys 25 % ja ehdon lisäämisen todennäköisyys 25 %.



Kuva 3.9. Mutaatioiden esiintymistiheyden vaikutus lopullisen säännöstön luokittelutarkkuuteen.

Kuten kuvasta 3.9. on nähtävissä sääntöjen risteytyksessä aiheutetut mutaatiot parantavat säännöstöä. Yhtälöiden yhteydessä tulokset lievästi paranevat mutaatiotiheyden kasvaessa. Logiikan yhteydessä siirryttäessä kymmenestä kahteenkymmeneen prosenttiin tulos paranee selvästi. Kummallakin aineistolla paras tulos saavutetaan 40% mutaatiotiheydellä.

3.5. Esimerkkitapausten vertailuun perustuva oppiminen

Vastakohtana oppimismenetelmille, jotka muodostavat tarkan kuvauksen oppimastaan aiheesta esimerkiksi päätöspuun tai säännöstön muodossa ovat esimerkkitapausten vertailuun (*instance-based*) perustuvat oppimismenetelmät, jotka etukäteen pelkästään tallettavat esimerkkitapaukset. Näitä menetelmiä kutsutaan usein laiskoiksi oppimismenetelmiksi (*lazy learning*), koska varsinainen prosessointi tapahtuu vasta viime hetkellä uutta tapausta luokiteltaessa. Vanhin ja tunnetuin tähän ryhmään kuuluva oppimisalgoritmi on nearest neighbor -algoritmi (Mitchell, 1997; Langley, 1996; Wettschereck, 1994), jota on käytetty ja tutkittu 50-luvun alkupuolelta lähtien. Algoritmi tallettaa kaikki harjoitusaineiston esimerkkitapaukset muistiin. Uuden tuntemattoman tapauksen luokittelu tapahtuu, algoritmin nimen mukaisesti, etsimällä muistissa olevista tapauksista luokiteltavaa tapausta eniten muistuttava (*nearest neighbor*) esimerkkitapaus. Uuden tapauksen luokaksi valitaan luokka, jota tämä lähin naapuri edustaa. k-nearest neighbor -algoritmin (kNN) tapauksessa lähimpiä naapureita etsitään k kappaletta, ja uuden tapauksen luokaksi valitaan se, minkä edustajia on naapuristossa eniten.

3.5.1. kNN-algoritmi

Esimerkkitapaus x esitetään muodossa $(c,[a_1(x),a_2(x),\ldots,a_n(x)])$, missä c on tapauksen x luokka ja $a_r(x)$ tarkoittaa tapauksen x attribuutin a_r arvoa. Etäisyys kahden tapauksen x_i ja x_j välillä määritellään seuraavasti

$$d(x_i, x_j) = \sqrt{\sum_{r=1}^{n} (a_r(x_i) - a_r(x_j))^2} \ ,$$

kun attribuutit ovat jatkuva-arvoisia, jolloin arvojen erotus on mielekäs. Lisäksi eri attribuuttien arvoalueiden tulisi olla yhtä laajoja (tai ne pitäisi skaalata sellaisiksi), jotta kaikki attribuutit vaikuttaisivat yhteenlaskettuun etäisyyteen samanarvoisesti. Kun attribuutit ovat arvoiltaan diskreettejä, etäisyysfunktio voidaan määritellä seuraavasti

$$d(x_i, x_j) = \sum_{r=1}^{n} \mathrm{E}(a_r(x_i), a_r(x_j)), \quad \text{missä} \quad \mathrm{E}(x, y) = \begin{cases} 0, & x = y \\ 1, & x \neq y \end{cases}.$$

Taulukon 3.1. esimerkkiaineiston kaikki attribuutit ovat diskreettejä. Aineiston kahden ensimmäisen tapauksen attribuutit ovat `[1,ei,neg,none]` ja `[1,ei,v,none]`. Koska tapaukset eroavat vain yhden attribuutin osalta toisistaan, niiden välinen etäisyys d = 1. Vastaavasti aineiston viimeisen tapauksen kuvaus on `[2,ei,none,v]` ja sen etäisyys molemmista kahdesta ensimmäisestä tapauksesta on kolme.

Ennestään tuntematonta tapausta luokiteltaessa etsitään k kappaletta sitä edellä esitetyn etäisyysmitan mukaan lähimpänä olevaa, ennestään tunnettu tapausta ja luokitellaan tapaus näiden lähimpien naapurien keskuudessa eniten edustettuna olevan luokan mukaan. Täsmällisemmin ilmaistuna: Olkoon tapaukset x_1 … x_k luokiteltavan tapauksen X k lähintä naapuria ja aihealueen luokkien joukko C, tällöin tapauksen X luokka

$$cl(X) = \underset{c \in C}{argmax} \sum_{i=1}^{k} \delta(c, cl(x_i)), \quad \text{missä} \quad \delta(x, y) = \begin{cases} 1, & x = y \\ 0, & muulloin \end{cases}.$$

Luokitellaan taulukon 3.1. esimerkkiaineiston avulla tapaus, joka on kuvattu attribuuttijoukolla `[1,ei,ja,none]`. Kun k = 5, tapauksen lähimmät naapurit ovat:

```
(ei,[1,ei,neg,none])
(ei,[1,ei,v,none])
(ei,[2,ei,ja,none])
(on,[1,ei,none,none])
(on,[1,on,ja,none])
```

Jokaisen viiden naapurin etäisyys luokiteltavasta tapauksesta on yksi. Koska näistä kolme edustaa luokkaa *ei* ja kaksi luokkaa *on*, tapauksen `[1,ei,ja,none]` luokaksi valitaan *ei*.

3.5.2. Painotus naapurien etäisyyden mukaan

Uutta tapausta luokiteltaessa voidaan ottaa huomioon se, kuinka lähellä tai kaukana sen lähimmät naapurit sitä ovat. Mitä lähempänä naapuri on, sitä enemmän sen sana ratkaisee tapausta luokiteltaessa. Yksinkertaisin tapa toteuttaa tämä on painottaa jokaisen naapurin ääntä sen ja luokiteltavan tapauksen etäisyyden käänteisluvulla. Muodollisemmin sanottuna, lisätään edellä olevaan tapauksen X luokan määrittävän lausekkeen summaosaan kerroin

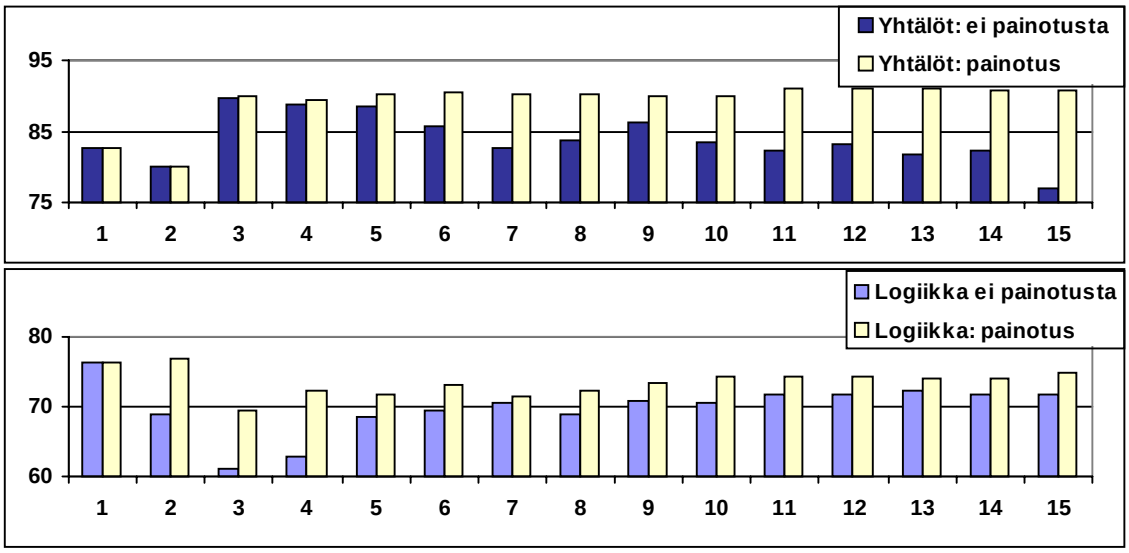
$$w_i = \frac{1}{d(X, x_i)}.$$

Luokitellaan edelleen samaa tapausta `[1,ei,ja,none]`, mutta valitaan k:n arvoksi 8. Viiden edellä mainitun naapurin lisäksi löytyy kolme seuraavaksi lähintä naapuria:

```
(ei,[2,ei,neg,none])
(on,[1,on,neg,none])
(on,[1,on,v,none])
```

Näiden kolmen tapauksen etäisyys luokiteltavasta tapauksesta on kaksi, joten etäisyyden mukaan painotettua laskentatapaa käyttäen saadaan luokan *ei* edustukselle arvo 3 × 1 + ½ = 3 ½ ja luokan *on* edustukselle vastaavasti 2 × 1 + 2 × ½ = 3. Edelleen luokaksi valitaan siis *ei*. Ilman etäisyyden huomioon ottamista molempien luokkien edustus olisi neljä, ja luokitus valittaisiin arpomalla.

Kuvassa 3.10. on verrattu kahden data-aineiston avulla lähimpien naapurien etäisyyden huomioimisen vaikutuksia tuntemattomien tapausten luokittelutarkkuuteen. Yhtälöiden tapauksessa luokittelutarkkuus selvästi parantuu etäisyyteen perustuvan painotuksen ansiosta, varsinkin kun lähimpien naapurien lukumäärä kasvaa yli viiteen. Logiikka-aineiston kohdalla painotuksesta saatava hyöty näyttäisi olevan pienempi, mutta kuitenkin selvästi havaittava.



Kuva 3.10. Etäisyyden huomioimisen vaikutus luokittelutarkkuuteen, kun lähimpien naapurien lukumäärä on 1 – 15.

3.5.3. Attribuuttien painotus

Lähimpiä naapureita etsittäessä, voidaan eri attribuuttien vaikutusta tapausten väliseen etäisyyteen painottaa siten, että jotkut attribuutit vaikuttavat siihen enemmän kuin toiset. Tällöin ajatellaan joidenkin attribuuttien tarjoavan enemmän luokitukseen liittyvää informaatiota kuin toisten. On myös mahdollista että, jotkin attribuutit sisältävät huomattavasti virheellistä tietoa, joka sopivalla painotuksella voidaan mitätöidä.

Sopivat painoarvot voidaan laskea harjoitusaineiston avulla, määrittämällä jokaisen attribuutin ja luokan välinen keskinäinen informaatio (*mutual information*). Kun aihealueen luokkien joukko on C ja attribuutin A arvojoukko V, voidaan attribuutin A painoarvo laskea kaavasta

$$w(A) = \sum_{v \in V} \sum_{c \in C} p(c,v) \log_2 \left(\frac{p(c,v)}{p(c)\,p(v)} \right),$$

missä p(c,v) tarkoittaa todennäköisyyttä (suhteellista osuutta) sille, että tapaus kuuluu luokkaan c ja että, sen attribuutin A arvo on v, p(c) sille, että tapaus kuuluu luokkaan c ja p(v) vastaavasti todennäköisyyttä sille, että tapauksen attribuutin A arvo on v.

Etäisyysfunktioon lisätään attribuuttien painotus, diskreeteillä attribuuteilla kuvattujen tapausten x_i ja x_j välinen etäisyys on nyt

$$d(x_i, x_j) = \sum_{r=1}^{n} w(a_r)\mathrm{E}(a_r(x_i), a_r(x_j)), \quad \text{missä} \quad \mathrm{E}(x, y) = \begin{cases} 0, & x = y \\ 1, & x \neq y \end{cases}$$

Vastaavasti muokataan jatkuva-arvoisille attribuuteille sopivaa etäisyysfunktiota.

Taulukon 3.1. esimerkkiaineistoa käyttämällä saadaan attribuuteille seuraavat painoarvot:

w(tp ryhmän numero) = 0.001

w(onko tp suluissa) = 0.115

w(edellinen merkki) = 0.427

w(seuraava merkki) = 0.161

Luokitellaan tapaus `[1,ei,ja,none]`. Kun k = 5, tapauksen lähimmät naapurit ja näiden etäisyydet painotettujen attribuuttien avulla laskettuna ovat:

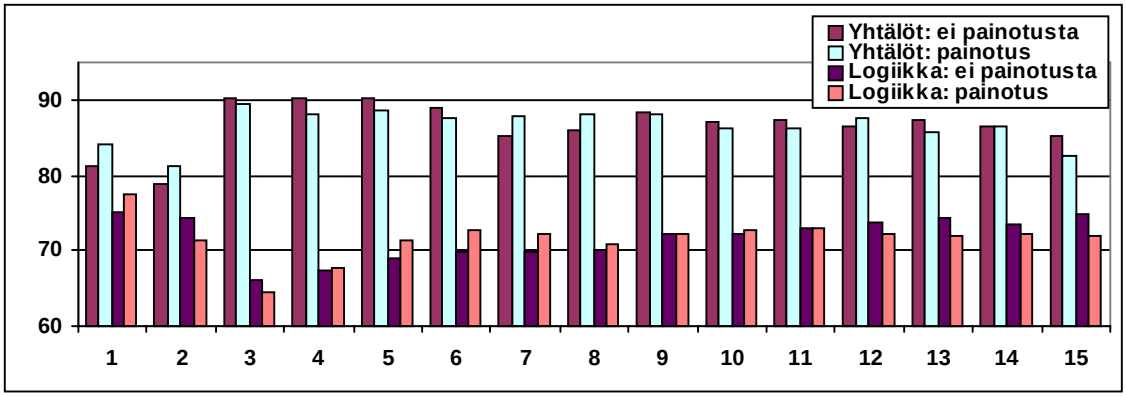
```
(ei,[2,ei,ja,none])          0.001
(on,[1,on,ja,none])          0.115
(on,[1,ei,none,none])        0.427
(ei,[1,ei,v,none])           0.427
(ei,[1,ei,neg,none])         0.427
```

Myös tässä tapauksessa tapauksen luokaksi valitaan *ei*. Tarkastellaan tilannetta kun k = 7, kaksi seuraavaksi lähintä naapuria ovat:

```
(on,[2,ei,none,none])        0.428
(on,[2,ei,im,none])          0.428
```

Nyt tapaus `[1,ei,ja,none]` luokitellaan luokkaan *on*, jos naapurien etäisyyksiä ei oteta huomioon.

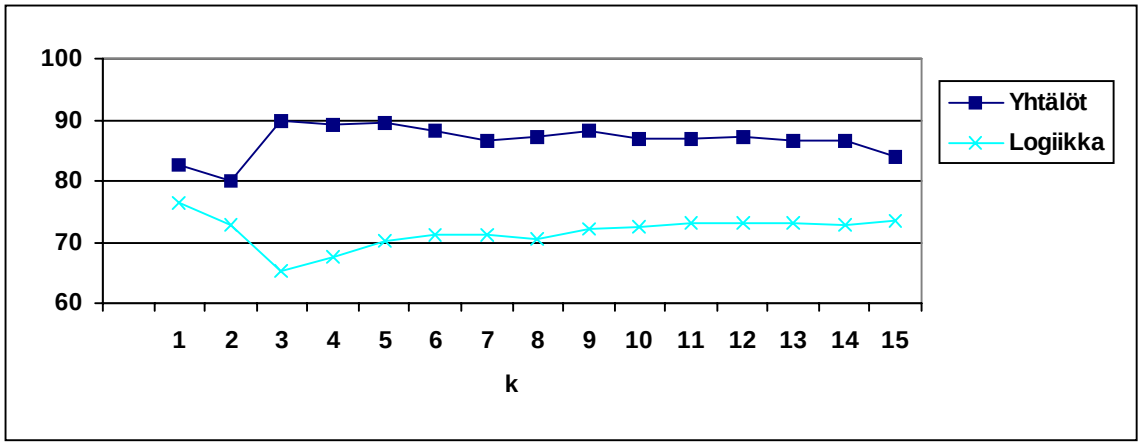
Alla olevassa kuvassa on esitetty attribuuttien painotuksen vaikutusta luokittelutarkkuuteen yhtälönratkaisun ja logiikan data-aineistojen yhteydessä. Attribuuttien painotus ei mitenkään selvästi paranna luokittelutarkkuutta kummankaan aineiston kohdalla. Näyttäisi siltä, että naapurien lukumäärän kasvaessa painotuksesta on pikemminkin haittaa kuin hyötyä.



Kuva 3.11. Attribuuttien painotuksen vaikutus luokittelutarkkuuteen.

3.5.4. Naapurien lukumäärän valinta

Lopputuloksen kannalta ratkaiseva tekijä on itse k:n arvo k-nearest neighbor -algoritmia käytettäessä. Alla oleva kuva esittää k-nearest neighbor -algoritmin luokittelutarkkuuden k:n arvoilla 1 – 15 kahden data-aineiston yhteydessä. Yhtälöiden kohdalla luokittelutarkkuus lähtee hiljalleen laskemaan, kun tarkasteltavien naapurien lukumäärä kasvaa viidestä ylöspäin. Logiikka-aineiston kohdalla taas luokittelutarkkuus on huonoimmillaan k:n arvolla kolme, mutta kasvaa hiljalleen, kun naapurien lukumäärää kasvatetaan.



Kuva 3.12. Lähimpien naapurien lukumäärän vaikutus luokittelutarkkuuteen.

Olemassa ei ole mitään helppoa ja suoraviivaista tapaa valita sopivaa k:n arvoa. Ainoa mahdollisuus on aina uuden data-aineiston kanssa aluksi kokeilla luokittelua eri k:n arvoilla ja valita sitten todelliseen käyttöön se arvo, joka testeissä tuottaa parhaat

tulokset. Testaukseen voi käyttää esimerkiksi kohdassa 4.2.2. esiteltyä *leave-one-out cross validation* -menetelmää (Moore and Lee, 1994).

3.6. Yhteenveto

Tässä luvussa on käsitelty toimenpiteiden sallitun käytön opettamista EduAgents-järjestelmän kumppaniagenteille. Opettamisella saavutettavia etuja ovat muun muassa ajoaikainen nopeus, ehdotusten yksilöllisyys ja kyky perustella miksi annettu ehdotus on sallittu.

Tarkasteltavana on ollut neljä induktiivista koneoppimisalgoritmia: esimerkkitapausten analysointia hyväkseen käyttävät ID3 ja C4.5, esimerkkitapausten vertailuun perustuva kNN-algoritmi (*k-nearest neighbor*) sekä evoluutiota simuloiva geneettinen algoritmi. Jokainen algoritmeista saa syötteenä joukon esimerkkitapauksia, jotka kuvaavat opetettavaa aihetta eli tässä tapauksessa aihetta 'onko toimenpide sallittu'. Algoritmit tuottavat esimerkkiaineiston perusteella tietorakenteen, jonka avulla ne tarkastelevat uusia tapauksia; toisin sanoen ne oppivat erottelemaan eri tilanteissa käytettävät sallitut toimenpiteet laittomista. ID3:n oppimistulos on päätöspuu, jonka haaroja seuraamalla uuden tapauksen luokitus määritetään, C4.5 ja geneettinen algoritmi taas muodostavat joukon sääntöjä vastaavaan tarkoitukseen. kNN-algoritmi toimii toisella tavalla: se luokittelee uuden tuntemattoman tapauksen etsimällä eniten sitä muistuttavia tapauksia esimerkkitapausten joukosta ja määräämällä sen luokan näiden tapausten enemmistön mukaan.

4. Menetelmien vertailu

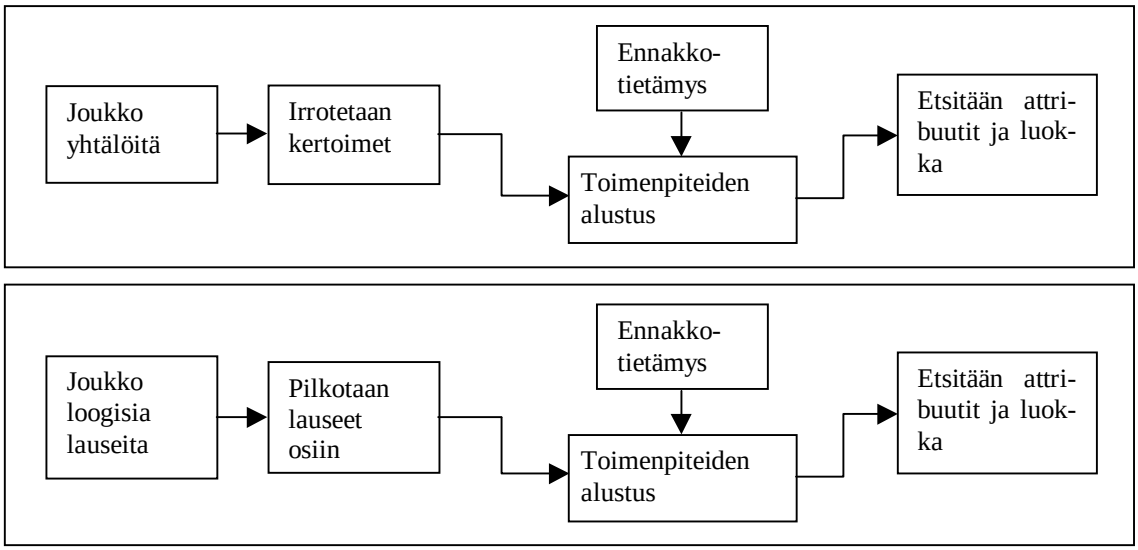
Tämän luvun aiheena on edellisessä luvussa esiteltyjen koneoppimisalgoritmien testaaminen käytännössä. Koska EduAgents-järjestelmä kykenee tarkistamaan tuntemiinsa sovellusalueisiin kuuluvia tehtäviä, voidaan oppimisalgoritmien vaatimat esimerkkitapaukset generoida automaattisesti. Kohta 4.1. käsittelee esimerkkitapausten generointia, kohdassa 4.2. tarkastellaan esimerkkiaineiston jakamista sopiviin harjoitus- ja testiaineistoihin, kohdassa 4.3. tutkitaan esimerkkitapausten sisältämän informaation määrää ja käytettävien attribuuttien valintaa. Kohta 4.4. sisältää tulokset algoritmien vertailusta ja lopuksi kohdassa 4.5. pohditaan eri algoritmien soveltumista EduAgents-järjestelmässä käytettäviksi.

4.1. Esimerkkiaineiston generointi

EduAgents-järjestelmällä on käytössään säännöstö, jossa on kuvattu sovellusalueen tietämys. Järjestelmä pystyy tämän säännöstön avulla tarkistamaan sovellusalueen tehtäviä ja siis vastaamaan kysymykseen, onko jokin toimenpide sallittu. Kumppaniagenttien koulutukseen tarvittavan esimerkkiaineiston generointi voidaan tämän ansiosta suorittaa automaattisesti. Toinen tapa muodostaa harjoitusaineisto on ohjelman ajon aikana kerätä todellisia esimerkkitapauksia: oppilaan toimenpide-ehdotus ja opettajan tarkistuksen tulos. Jälkimmäistä tekniikkaa käytetään agenttien jatkokoulutuksessa, josta enemmän kohdissa 4.5. ja 6.2.

Tällä hetkellä EduAgentsiin toteutettuja sovellusaluemoduuleita on kaksi: toinen sisältää ensimmäisen asteen yhtälönratkaisun tietämyksen ja toinen lauselogiikan tietämyksen. Esimerkkitapausten tuottaminen kummallakin sovellusalueella tapahtuu hyvin

samankaltaisesti. Logiikan alueella toimenpiteitä voi yleensä soveltaa yhtä hyvin koko lauseeseen kuin lauseen osiin, joten lauseet pilkotaan ensin osalauseiksi ja sovitetaan toimenpiteet kaikkiin näihin osiin. Yhtälönratkaisussa taas toimenpiteet tarkoittavat joko muuttujatermien kertoimien tai vakiotermien käsittelyä. Yhtälöistä irrotetaan kertoimet ja alustetaan ennakkotietämyksestä löytyvät toimenpiteet näillä numeroarvoilla. Kuvassa 4.1. on esitetty esimerkkitapausten muodostamisen päävaiheet.



Kuva 4.1. Esimerkkiaineiston generointi yhtälönratkaisun ja lauselogiikan alueilta.

Käytetään esimerkiksi yhtälöä $3x - 4 + 5x = 6$ lähtökohtana. Siitä irrotetaan kertoimet 3 ja 5. Ennakkotietämyksestä löytyy mm. toimenpide $ax + bx$, ja sovittamalla löydetyt kertoimet tähän toimenpiteen yleiseen muotoon saadaan alustettu toimenpide $3x + 5x$. Myös toimenpide $–a$ alustettuna arvolla –4 eli toimenpide $+4$ saadaan muodostettua yhtälöstä. Logiikan puolella voidaan tarkastella lausetta $p \,\&\, r \vee q$. Siitä voidaan irrottaa kolme osalausetta: $p \,\&\, r$, $r \vee q$ ja $p \,\&\, r \vee q$. Käyttämällä ennakkotietämyksestä löytyvää toimenpidettä $a \vee b \Leftrightarrow b \vee a$ saadaan alustettu toimenpide $r \vee q \Leftrightarrow q \vee r$. Lauseesta saadaan muodostettua muitakin toimenpiteitä, esimerkiksi ennakkotietämyksen toimenpiteet $a \,\&\, b \Leftrightarrow b \,\&\, a$, $a \vee b \Leftrightarrow \neg a \Rightarrow b$ ja $a \,\&\, b \Leftrightarrow \neg(\neg\, a \vee \neg b)$ voidaan alustaa ko. lauseen avulla.

Lopputuloksena saatavan aineiston laatuun vaikuttaa se, millainen lausejoukko tai yhtälöryhmä generointialgoritmille annetaan syötteenä. Kolmella eri tavalla muodostettuja lähtöjoukkoja on kokeiltu: satunnaisista lauseista tai yhtälöistä muodostettuja, EduAgentsin harjoitustehtävistä koottuja sekä päättelemällä ja kokeilemalla saavutettuja

joukkoja. Satunnaisia lähtöjoukkoja käyttämällä muodostetut aineistot sisältävät liian paljon negatiivisia esimerkkejä eli tapauksia, joiden luokka on *ei*. Toisaalta esimerkkitapauksia pitää tuottaa melkoinen määrä, jotta saataisiin aikaan myös harvinaisia (monimutkaisia) toimenpiteitä sisältäviä esimerkkejä. Pahimmassa tapauksessa tämä johtaa siihen, että oppimisalgoritmi muodostaa aineiston perusteella vain yhden ehdottoman säännön: *ei*. Toisin sanoen kumppaniagentti oppii ettei mikään toimenpide ole sallittu. EduAgentsin harjoitustehtävien avulla muodostetut aineistot taas tuntuvat suosivan yleisimpiä toimenpiteitä. Varsinkin yhtälönratkaisun yhteydessä näistä aineistoista opitut säännöt ovat usein muotoa *jos toimenpiteen nro niin luokka*, eli kumppani oppii esimerkiksi sen, että vakioiden yhteenlasku on aina sallittua ja toisaalta kertolasku aina kiellettyä.

Paras tapa on muodostaa esimerkkiaineisto lähtemällä liikkeelle päättelemällä ja kokeilemalla aikaansaadusta joukosta. Lauseita ja yhtälöitä koetetaan valita lähtöjoukkoihin niin, että aikaansaatu aineisto sisältäisi molempia luokkia, *on* ja *ei*, edustavia tapauksia sunnilleen yhtä paljon. Lisäksi pyritään aikaansaamaan aineisto, jossa kaikki toimenpideryhmät ovat suunnilleen yhtä vahvasti edustettuina. Toivottavaa on myös se, että jokaisen ryhmän negatiivisten ja positiivisten esimerkkien lukumäärät ovat samaa suuruusluokkaa. Kohdassa 4.4. kuvatussa algoritmien vertailussa käytetyt esimerkkiaineistot on muodostettu valitsemalla edellä kuvatulla tavalla lähtöjoukot generointialgoritmille. Yhtälönratkaisun alueella lähtöjoukkona on käytetty 18 yhtälöä, joista on generoitu 176 esimerkkitapausta. Vastaavasti logiikan alueella kymmenestä lauseesta on tuotettu 121 esimerkkitapausta. Lähtöjoukot on esitetty liitteessä 2.

Myös esimerkkitapauksiin talletettava informaatio eli käytettävät attribuutit vaikuttavat oppimistulokseen. Esimerkkitapauksia generoitaessa informaatiota talletetaan paljon: yhtälönratkaisun yhteydessä käytetään kahtatoista attribuuttia ja logiikan yhteydessä on käytössä 29 attribuuttia. Molemmat attribuuttiryhmät on esitetty liitteessä 1. Myöhemmässä vaiheessa kumppaniagentteja opetettaessa osa attribuuteista saatetaan jättää huomiotta (ks. kohta 4.3.).

4.2. Esimerkkiaineiston jakaminen osiin

Oppimisalgoritmin tarkkuutta arvioitaessa mitataan sitä, kuinka hyvin se pystyy luokittelemaan ennestään tuntemattomia esimerkkitapauksia. Tuntemattomilla tapauksilla

tarkoitetaan esimerkkitapauksia, jotka eivät ole olleet mukana algoritmia opetettaessa. Esimerkkiaineisto pitää jollakin tavoin siis jakaa kahteen erilliseen osaan: harjoitusaineistoon ja testiaineistoon. Alla on esitelty kaksi yleistä tapaa suorittaa aineiston jakaminen.

4.2.1. kFCV-menetelmä

kFCV-menetelmä (*k-fold cross validation*) toimii siten, että esimerkkiaineistosta muodostetaan *k* kappaletta erillistä yhtä suurta osajoukkoa. Yleensä *k* on tapana valita siten, että kunkin osajoukon koko on vähintään 30 esimerkkitapausta. Tällöin voidaan perustellusti käyttää joitakin tilastotieteen teoreemoja, esimerkiksi keskeistä raja-arvolausetta (Mitchell, 1997). Oppimisalgoritmi toistetaan tämän jälkeen *k* kertaa valitsemalla vuorotellen jokainen osajoukoista testiaineistoksi ja muut *k-1* osajoukkoa harjoitusaineistoksi. Algoritmi tulee siis testattua *k*:lla itsenäisellä testiaineistolla, ja tulosten keskiarvo antaa hyvän arvion algoritmin luokittelutarkkuudesta.

4.2.2. LOOCV-menetelmä

LOOCV-menetelmää (*leave-one-out cross validation*) käytettäessä esimerkkiaineistosta valitaan yksi tapaus vuorollaan. Loppuosaa aineistosta käytettään oppimisalgoritmin harjoitusaineistona ja valittu tapaus luokitellaan tämän jälkeen algoritmin avulla. Koko esimerkkiaineisto käydään läpi niin, että jokainen aineiston tapaus tulee luokiteltua kaikkien muiden tapausten avulla. Oikein luokiteltujen tapausten osuus koko esimerkkiaineistosta kertoo algoritmin luokittelutarkkuuden. LOOCV-menetelmää käytetään yleensä laiskojen oppimisalgoritmien, kuten k-nearest neighbor -algoritmin yhteydessä (Moore and Lee, 1994). LOOCV-menetelmä ei ole juurikaan hitaampi kuin kFCV-menetelmä laiskojen algoritmien yhteydessä, koska nämä suorittavat varsinaisen prosessoinnin vasta tapauksia luokitellessaan. Säännöstöjä muodostavien algoritmien (esim. ID3 ja C4.5) kanssa tilanne on toinen; oppimisalgoritmi suoritetaan jokaiselle aineiston tapaukselle erikseen ja tämä vie mahdollisesti paljonkin aikaa.

4.3. Käytettävien attribuuttien valinta

Hyödyllisten attribuuttien valitseminen ja epäoleellisten hylkääminen on eräs induktiivisen koneoppimisen keskeisimmistä ongelmista. Vaikka useimmat oppimisalgoritmit itse karsivat attribuutteja tai asettavat niitä tärkeysjärjestykseen, heikentää hyödyttömien attribuuttien mukanaolo oppimistulosta. Langleyn (1994) mukaan sekä teoreettiset analyysit että empiiriset tutkimukset osoittavat, että monien algoritmien luokittelutarkkuus pienenee, kun harjoitusaineisto sisältää suuren määrän luokittelun kannalta epäoleellista informaatiota.

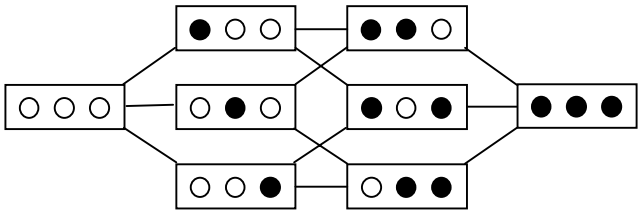
Liu ja Setiono (1996) määrittelevät attribuuttien valinnan (*feature selection*) seuraavasti: etsitään alkuperäisten N attribuutin joukosta ne M attribuuttia, joita käyttämällä minimoidaan luokitteluvirheiden todennäköisyys. Valintastrategiat voidaan jakaa kahteen ryhmään: suodatusmenetelmät ja kuorimenetelmät (*wrapper*). Suodatusmenetelmät tarkastelevat attribuutteja esimerkiksi tilastotieteen keinoin, ottamatta varsinaista oppimisalgoritmia lainkaan huomioon. Kuorimenetelmät taas käyttävät oppimisalgoritmia erilaisten attribuuttijoukkojen kanssa ja valitsevat sen joukon, jonka yhteydessä algoritmi on tuottanut parhaan tuloksen. Yleensä kuorimenetelmät tuottavat suodatusmenetelmiä parempia tuloksia, mutta ovat laskennallisesti huomattavasti raskaampia.

Kaikkien mahdollisten attribuuttijoukkojen tarkasteleminen on yleensä mahdotonta, koska N attribuutin joukosta voi muodostaa 2^N erilaista osajoukkoa. Tästä syystä on kehitetty erilaisia heuristisia menetelmiä, joiden avulla parasta attribuuttijoukkoa etsitään. Seuraavaksi esitellään kaksi kuorimenetelmää, joista ensimmäinen käyttää hill-climbing -tyyppistä hakutekniikkaa ja jälkimmäinen perustuu satunnaisuuteen ja todennäköisyyteen.

4.3.1. BSE-menetelmä

BSE-menetelmä (*Backward Stepwise Elimination*) on hakumenetelmä, joka kaikkien mahdollisten attribuuttijoukkojen avaruudesta etsii hill-climbing -tyyppisesti parasta osajoukkoa (Caruana and Freitag, 1994; Langley, 1994). Haun aloituskohta on joukko, jossa kaikki attribuutit ovat mukana (kuvassa 4.2. oikeassa reunassa). Tästä joukosta poistetaan vuorotellen jokainen attribuuteista ja eniten oppimisalgoritmin luokittelutarkkuutta parantanut poisto jätetään voimaan. Seuraavilla kierroksilla tarkastellaan

myös mahdollisuutta lisätä jokin aiemmin poistetuista attribuuteista. Kuva 4.2. esittää kolmen attribuutin osajoukkojen avaruutta. Avaruuden jokaisesta pisteestä on mahdollisuus siirtyä kolmeen muuhun pisteeseen; siirtyminen vasemmalle merkitsee attribuutin poistoa, siirtyminen oikealla attribuutin lisäämistä. Algoritmi pysähtyy, kun jokainen mahdollinen muutos huonontaa oppimisalgoritmin luokittelutarkkuutta (tai luokittelutarkkuus ei kasva, mutta attribuuttijoukko kasvaa).



Kuva 4.2. Mahdolliset tilat kolmen attribuutin osajoukkojen avaruudessa. Siirtymä tilasta toiseen tarkoittaa joko attribuutin (mustat ympyrät) lisäämistä osajoukkoon tai sen poistamista osajoukosta.

4.3.2. LVW-menetelmä

LVW-menetelmä (*Las Vegas Wrapper*) on todennäköisyyteen perustuva menetelmä, jonka Liu ja Setiono (1996) ovat kehittäneet Las Vegas -algoritmin pohjalta. Algoritmi generoi *N* attribuutin joukosta satunnaisia osajoukkoja. Oppimisalgoritmia toistetaan käyttäen vuorollaan kutakin osajoukkoja *S1* ja jokaisella toistolla lasketaan luokittelu-

```
err = 1; k = 0; C = N;
repeat
      S1 = randomSet();
      C1 = numOfFeatures(S1);
      err1 = LearnAlgo(S1,D);
      if (err1 < err OR (err1 = err AND C1 < C)) {
            k = 0; err = err1;
            C = C1; S = S1; }
      k = k +1;
until  k  > K
return S;
```

Kuva 4.3. LVW algoritmi.

virheelle arvio *err1*. Säilytetään aina pienin attribuuttijoukko, joka tuottaa pienimmän luokitteluvirheen. Kun toistoja suoritetaan riittävästi algoritmi löytää optimaalisen osajoukon. Algoritmi on esitetty pseudokoodina kuvassa 4.3. *S* on kullakin hetkellä paras attribuuttijoukko, *C* on tämän joukon koko, *C1* tarkoittaa joukon *S1* kokoa ja *k* on tehtyjen toistojen lukumäärä. *K* on tehtävien toistojen maksimimäärä. Liu ja Setiono (1996) suosittelevat sen arvoksi suuruusluokkaa 60 x N olevaa lukua.

4.4. Menetelmien vertailu

Vertailussa on yhtälönratkaisun alueella käytetty esimerkkiaineistoa, jossa on 176 esimerkkitapausta. Jokainen tapaus sisältää 12 attribuuttia ja tapauksen luokituksen. Logiikan alueella käytetty esimerkkiaineisto sisältää 121 esimerkkitapausta, joista kukin on kuvattu 29 attribuutilla ja tiedolla tapauksen luokasta. kFCV-menetelmän yhteydessä yhtälönratkaisun alueella on esimerkkiaineisto jaettu viiteen osaan, eli on käytetty k:n arvona lukua viisi, logiikan alueella vastaavasti k:n arvoksi on valittu luku neljä.

4.4.1. Algoritmien optiot

ID3:a ei kokeilemalla optimoitu, vaan alkuperäisyyden vuoksi vertailussa käytettiin valintakriteeriä gain, eikä tuloksena saatuja päätöspuita karsittu. C4.5:n yhteydessä käytettiin parametrejä, jotka ohjelman alkuperäisessä versiossa ovat oletusarvoina (Quinlan, 1993). Geneettisen algoritmin ja k-nearest neighbor -algoritmin kanssa optioiden valinta suoritettiin työläämmällä tavalla eli kokeilemalla.

Geneettinen algoritmi sisältää monia optioita ja säädettäviä lukuarvoja, joten sen optimoiminen suoritettiin kahdessa osassa. Ensimmäisessä osassa valittiin *risteytys- eloonjäänti- ja lisääntymistavat*. Valinta suoritettiin kokeilemalla kaikkia edellä mainittujen tapojen eri yhdistelmiä (12 kpl), kun mutaatioprosentti ja populaation koko pidettiin kiinteänä. Kokeilut suoritettiin käyttämällä kFCV-menetelmää ja ajot toistettiin kolme kertaa. Molemmilta aihealueilta kolme parasta yhdistelmää kelpuutettiin jatkovertailuun, jossa etsittiin sopivaa *mutaatioprosenttia* ja optimaalista *populaation kokoa*. Mutaatioprosentille kokeiltiin arvoja 10, 20, 30 ja 40, populaation koolle annettiin vuorollaan arvot 30, 40, 50 ja 60. Kaikki arvojen yhdistelmät (16 kpl) testattiin edelliseltä

kierrokselta jatkoon selvinneiden optioiden kanssa käyttäen uudelleen kFCV-menetelmää. Ajot suoritettiin kahdesti ja valittiin näiden perusteella alla olevat optiot varsinaiseen algoritmien vertailuun.

	risteytys	eloonjääntі	lisääntyjät	mutaatio %	populaation koko
Yhtälönratkaisu	alleelit	fixed	best	40	50
Logiikka	summa	fixed	all	10	40

Geneettinen algoritmi on luonteeltaan epädeterministinen: alkuperäinen populaatio arvotaan, sekä risteytys että mutaatiot toteutetaan arvonnan avulla ja myös lisääntyjien valinnasta ja joillakin optioilla vielä yksilöiden eloonjäännistäkin päättää arpa. Algoritmin optimoimiseksi suoritettujen vertailuajojen lukumäärä ei siis ollenkaan riitä takaamaan sitä, että todella parhaat arvot eri muuttujille tuli valittua. Lisäksi harhaa voi aiheuttaa optimoinnin suorittaminen kahdessa osassa, saattaahan esimerkiksi populaation koko vaikuttaa risteytystapaan tai mutaatioprosentti eloonjääntiin. Joka tapauksessa jo kuvatulla tavallakin tehdyssä optimoinnissa oppimisalgoritmi suoritettiin yli 1400 kertaa.

k-nearest neighbor -algoritmin yhteydessä optimoitavia arvoja oli vähemmän: valittavana oli se, käytetäänkö attribuuttien painotusta, huomioidaanko naapurien etäisyys ja varsinkin se, mikä on luokittelussa käytettävien naapurien lukumäärä. Valinta suoritettiin kokeilemalla kahden ensimmäisen ominaisuuden kaikkia neljää eri yhdistelmää ja antamalla naapurien lukumäärälle vuorollaan kaikki arvot yhdestä kolmeenkymmeneen. Kokeilussa käytettiin LOOCV-menetelmää ja seuraavat yhdistelmät valittiin.

	etäisyyden huomioiminen	attribuutirn painotus	naapurien lkm
Yhtälönratkaisu	kyllä	ei	6
Logiikka	kyllä	ei	2

Ajojen toistaminen tässä tapauksessa olisi turhaa, sillä nearest neighbor -algoritmi on deterministinen ja vertailussa käytetty LOOCV-menetelmä luokittelee esimerkkiaineiston jokaisen tapauksen tasan kerran.

4.4.2. Attribuutit

Kaikkien neljän oppimisalgoritmin attribuuttien valintaan käytettiin BSE-hakumenetelmää (*Backward Stepwise Elimination*), koska se on huomattavasti LVW-menetelmää (*Las Vegas Wrapper*) nopeampi. ID3:n, C4.5:n ja geneettisen algoritmin käyttämiä attribuutteja valittaessa BSE-menetelmän yhteydessä käytettiin kFCV-menetelmää (*k-fold cross validation*) kun taas k-nearest neighbor -algoritmin käyttämä attribuuttijoukko valittiin LOOCV-menetelmän (*leave-one-out cross validation*) avulla. Attribuuttien valinnat tehtiin yhden ajon perusteella, mikä on tietenkin selvä puute jokaisen algoritmin kohdalla. Geneettisen algoritmin lisäksi myös ID3 ja C4.5 sisältävät monia epädeterministisiä piirteitä ja itse BSE-hakumenetelmä turvautuu tasatilanteissa arvontaan. Ajoja toistamalla varmuus siitä, että on valittu paras mahdollinen attribuuttijoukko, kasvaisi huomattavasti. Liite 3 sisältää valitut attribuuttijoukot.

4.4.3. Vertailun tulokset

Oppimisalgoritmien testauksessa on käytetty kFCV-menetelmää siten, että jokainen algoritmi on vuorollaan käyttänyt samoja esimerkkiaineiston satunnaisia osajoukkoja. Ajot on sekä yhtälönratkaisun että logiikan aihepiirissä toistettu kymmenen kertaa ja tuloksista on laskettu keskiarvot kunkin algoritmin luokittelutarkkuudelle. Luokittelutarkkuudet on esitetty seuraavassa taulukossa.

	ID3	C4.5	GA	kNN
Yhtälöt	83.53	83.52	88.64	90.85
Logiikka	80.70	77.60	72.50	75.87

Taulukko 4.1. Algoritmien luokittelutarkkuudet prosentteina yhtälönratkaisun ja logiikan alueilla.

Oppimisalgoritmien vertailu on tehty parittaisena t-testinä (*paired t-test*), joka on nykyisin käytetyin testi oppimismenetelmiä vertailtaessa. Dietterich (1998) tosin on sitä mieltä, että t-testin käytöstä aiheutuu usein 1. lajin virhettä, eli oppimisalgoritmien luokittelutarkkuudesta löytyy eroja silloinkin, kun niitä ei todellisuudessa ole. Taulukoissa 4.2. ja 4.3. on esitetty parittaisten 2-suuntaisten t-testien tulokset: ensimmäinen luku on

t-arvo ja suluissa oleva luku on todennäköisyys sille, että algoritmien luokittelutarkkuus on sama.

	C4.5	GA	kNN
ID3	0,014 (0.989)	-5.691 (0.0)	-8.064 (0.0)
C4.5		-6.012 (0.0)	-10.285 (0.0)
GA			-3.222 (0.002)

Taulukko 4.2. Algoritmien välisten parittaisten t-testien tulokset (yhtälönratkaisu).

	C4.5	GA	kNN
ID3	2.976 (0.005)	4.787 (0.0)	3.856 (0.0)
C4.5		2.981 (0.005)	1.510 (0,139)
GA			-1.822 (0.076)

Taulukko 4.3. Algoritmien välisten parittaisten t-testien tulokset (logiikka).

Tuloksista nähdään, että yhtälönratkaisun yhteydessä k-nearest neighbor -algoritmi on ylivoimainen kaikkiin muihin algoritmeihin verrattuna. Logiikan alueella tulos on tasaväkisempi, tosin geneettinen algoritmi on selvästi muita algoritmeja huonompi. ID3 voidaan hyväksyä algoritmeista parhaaksi merkitsevyystasolla 0.05.

4.5. Algoritmien käyttömahdollisuuksista EduAgents-järjestelmässä

Tässä kohdassa tarkastellaan aiemmin kuvattujen oppimisalgoritmien käyttöä EduAgents-järjestelmässä. Huomiota kiinnitetään ensinnäkin algoritmien luokittelu- ja oppimisnopeuteen sekä siihen, pystyvätkö algoritmit perustelemaan vastauksensa. Algoritmit jakaantuvat näiden ominaisuuksiensa puolesta kahteen ryhmään: toisen muodostavat ID3, C4.5 ja geneettinen algoritmi ja toisen k-nearest neighbor -algoritmi yksinään.

Ensiksi mainitun ryhmän algoritmit luokittelevat uusia tapauksia muodostamansa säännöstön tai päätöspuun ansiosta nopeasti. k-nearest neighbor -algoritmi sen sijaan etsii uutta tapausta luokitellessaan joukon tapauksen lähimpiä naapureita ja näiden avulla määrittelee sen luokan. Prosessiin kuluu jonkin verran aikaa, joten k-nearest neighbor -algoritmi on ajoaikana muita algoritmeja hitaampi. Luokittelunopeus on

kriittinen ominaisuus silloin, kun kumppaniagentti on muodostamassa ehdotustaan ja käy läpi useita eri toimenpidevaihtoehtoja.

Sääntöjä muodostavat algoritmit (C4.5 ja geneettinen algoritmi) perustelevat tavallaan automaattisesti miksi ne luokittelevat jonkin tapauksen tiettyyn luokkaan: perusteluna kumppaniagentti voi käyttää sääntöä, jonka mukaan luokitus tapahtuu. ID3:n kohdalla tilanne on kutakuinkin yhtä hyvä, sillä päätöspuun polun muuntaminen säännöksi on triviaali toimenpide. k-nearest neighbor -algoritmia käyttävä kumppaniagentti joutuu sitä vastoin myös tässä tehtävässä vaikeuksiin: lähimpiä naapureita ei oikein hyvin voi käyttää perusteluna. Voisi tietysti ajatella, että perustelu olisi viittaus samantapaiseen tilanteeseen ja sen yhteydessä tehtyyn hyväksyttyyn ratkaisuun, mutta suurin osa ohjelman muistissa olevista esimerkkitapauksista on annettu algoritmille harjoitusvaiheessa, eivätkä ne siis ole ohjelmaa käyttävälle ihmisoppijalle tuttuja. Tällaiset perustelut saattaisivat enemmänkin sekoittaa kuin selventää opittavaa asiaa. Toinen ongelma on se, ettei ohjelman muistissa ole varsinaisia tehtäviä (yhtälöitä tai loogisia lauseita), vaan näitä kuvaavia attribuuttijoukkoja.

Kumppaniagenttien jatkokoulutuksessa käytetään todellisista tilanteista kerättyä dataa. Ohjelman suorituksen aikana sekä ihmiskäyttäjän että kumppaniagentin tekemät toimenpide-ehdotukset ja vastaavat opettaja-agentin tarkistuksen tulokset on mahdollista tallentaa muistiin. k-nearest neighbor -algoritmia käyttävällä kumppaniagentilla tämä uusi materiaali on välittömästi käytössään, koska se ei tarvitse minkäänlaista etukäteisprosessointia. Kolmea muuta algoritmia käyttävillä agenteilla ei sitä vastoin ole mahdollisuutta hyödyntää uutta tietoa heti, koska niiden käyttämien oppimisalgoritmien suorittamiseen kuluu yleensä sen verran aikaa että, niitä ei tästä syystä ole mahdollista käyttää ohjelman suorituksen aikana. Näiden kumppanien jatko-opetus hoidetaan oppituntien välillä käyttämällä kasvanutta esimerkkiaineistoa.

	ID3	**C4.5**	**GA**	**kNN**
luokittelu	nopea	nopea	nopea	hidas
perustelut	kyllä	kyllä	kyllä	ei
jatko-opetus	jälkeenpäin	jälkeenpäin	jälkeenpäin	vauhdissa

Taulukko 4.4. Algoritmien ominaisuuksia.

Taulukko 4.4. sisältää yhteenvedon EduAgents-järjestelmän kannalta tärkeistä oppimisalgoritmeihin liittyvistä ominaisuuksista. k-nearest neighbor -algoritmin vahvuus on kyky käyttää uutta tietoa heti. Lisäksi edellisen kohdan vertailun mukaan tämä

algoritmi oppi yhtälönratkaisun huomattavasti muita algoritmeja paremmin. Se on kuitenkin hidas ajoaikaisessa luokittelussa, eikä osaa perustella tekemiään luokituksia. Etukäteen oppivat algoritmit taas ovat ajoaikaisessa luokittelussa nopeita ja kykenevät perustelemaan tekemänsä ratkaisut, mutta eivät pysty hyödyntämään uutta tietoa välittömästi.

4.6. Yhteenveto

EduAgents-järjestelmään on toteutettu kaksi sovellusaluemoduulia: toinen käsittelee ensimmäisen asteen yhtälönratkaisua ja toinen lauselogiikkaa. Toimenpiteiden sallitun käytön opettamiseksi kumppaniagenteille on kummankin oppiaineen alueelta generoitu esimerkkiaineistot. Yhtälönratkaisun alueella aineisto sisältää 176 esimerkkitapausta, jotka on kuvattu kahdellatoista attribuutilla. Vastaavasti logiikan alueen aineistoon kuuluu 29 attribuutilla kuvattuja esimerkkitapauksia 121 kappaletta.

Oppimisalgoritmin tarkkuutta arvioitaessa mitataan sitä, kuinka hyvin se pystyy luokittelemaan ennestään tuntemattomia esimerkkitapauksia. Esimerkkiaineisto pitää jollakin tavoin siis jakaa kahteen osaan: harjoitusaineistoon ja testiaineistoon. Tämä voidaan tehdä esimerkiksi kFCV-menetelmällä (*k-fold cross validation*) tai LOOCV-menetelmällä (*leave-one-out cross validation*).

Hyödyllisten attribuuttien valitseminen ja epäoleellisten hylkääminen on eräs induktiivisen koneoppimisen keskeisimmistä ongelmista. Esitellyistä kahdesta attribuuttien valinnan tekevästä menetelmästä, BSE-menetelmä (*Backward Stepwise Elimination*) valittiin käytettäväksi oppimisalgoritmien vertailussa, koska se on suoritusajaltaan huomattavasti LVW-menetelmää (*Las Vegas Wrapper*) nopeampi.

Luokittelutarkkuutta vertailtaessa havaittiin, että yhtälönratkaisun yhteydessä k-nearest neighbor -algoritmi on selvästi paras, geneettinen algoritmi yhtä selvästi toiseksi paras menetelmistä. Logiikan alueella tulos on tasaväkisempi, tosin geneettinen algoritmi on muita algoritmeja huonompi, ID3 taas algoritmeista paras.

5. Käytettävän toimenpiteen valinta

Edellisissä luvuissa käsiteltiin laillisten toimenpiteiden oppimista. Toimenpiteiden sääntöjenmukainen käyttö ei kuitenkaan pelkästään riitä tehtävän ratkaisuun, vaan toimenpiteistä on pystyttävä muodostamaan sopiva sarja, joka johtaa haluttuun lopputulokseen. Tässä luvussa pohditaan erilaisia tapoja, joilla kumppaniagentti voi suunnitella toimenpideketjuja, jotka johtavat kohti lopullista tehtävän ratkaisua. Ensiksi tarkastellaan raa'an voiman hakumenetelmiä, sen jälkeen lisätään hakuihin mukaan heuristiikkaa ja lopuksi on vuorossa tapauskohtaiseen päättelyyn perustuva menetelmä. Kohta 5.3 sisältää vertailun eri valintastrategioiden käytöstä. Kaikki tässä luvussa mainittavat testiajot ja aikavertailut on suoritettu 166 MHz:n pentium-prosessorilla varustetulla PC-tietokoneella.

5.1. Täydellinen etsintä

Yksinkertaisin tapa muodostaa ratkaisun tuottava toimenpideketju on käyttää joko syvyyssuuntaista (*depth-first*) tai leveyssuuntaista (*breadth-first*) etsintää. Valitettavasti tässä yhteydessä nämä tavanomaiset etsintäalgoritmit toimivat huonosti. Syvyyssuuntainen haku juuttuu helposti silmukkaan: yhtälönratkaisun sovellusalueella esimerkiksi käyttämällä vuorotellen toimenpiteitä luvun lisäys yhtälön molemmille puolille ja poisto molemmilta puolilta, tai soveltamalla jatkuvasti vaihdantalakia logiikan lauseisiin. Myös leveyssuuntaisessa etsinnässä suuri osa hakupoluista johtaa takaisin lähtötilaan. Tilannetta voi hieman parantaa käyttämällä satunnaisuutta. Kun jokaisella askeleella arvalla valitaan soveltuvista toimenpiteistä se, jota käytetään, on mahdollisuudet lopputuloksen löytymiseen hieman paremmat, kuin valittaessa aina ensimmäinen soveltuva

toimenpide. Edellä mainitut umpikujatilanteet voi tietenkin estää myös pitämällä kirjaa tehtävän ratkaisuvaiheista ja estää palaaminen takaisin tilaan, joka on jo kertaalleen käsitelty.

Suurin ongelma on kuitenkin ohjaamattoman etsinnän hitaus. Esimerkiksi logiikan sovellusalueen ensimmäiseen harjoitustehtävään *(p v q) & (p v q)* voi suoraan soveltaa kuuttatoista eri toimenpidettä ja nämä tuottavat edelleen noin 350 ratkaisuvaihtoehtoa siirryttäessä etsinnässä askel eteenpäin. Näiden läpikäyminen vie aikaa noin 20 sekuntia, k-nearest neighbor -algoritmia käyttäen noin 20 kertaa kauemmin. Kolmannella tasolla leveyssuuntaisessa etsinnässä on jo tutkittavana yli 5000 mahdollista toimenpidettä, joiden tarkistamiseen aikaa kuluu lähestulkoon viisi minuuttia. Monimutkaisimmat logiikan harjoitustehtävät ratkeavat vasta yhdeksän toimenpiteen soveltamisen jälkeen. Vaikka yhtälönratkaisun yhteydessä vaadittavat toimenpideketjut ovat lyhyempiä ja kulloiseenkin tilaan soveltuvien toimenpiteiden määrä pienempi, on ratkaisun etsiminen edellä mainituilla raa'an voiman menetelmillä liian hidasta.

Lisäksi kokonaisen lopputulokseen johtavan toimenpideketjun etsintä jokaisen ratkaisuaskeleen yhteydessä on jokseenkin epäkiitollinen tehtävä, koska varsinainen oppilas kuitenkin päättää, mitä toimenpidettä käytetään. Usein tehtävänratkaisu siis kulkee aivan toista reittiä kuin sitä, minkä kumppaniagentti on etsinyt.

5.2. Heuristisia menetelmiä

Jotta kumppaniagentti onnistuisi järjellisessä ajassa valitsemaan toimenpiteen, joka optimaalisesti johtaa kohti tehtävänratkaisua, pitää löytää yllä mainittuja hakumenetelmiä kehittyneempiä tapoja kokonaisratkaisun etsintään. Tällaisia sovellusalueen tietämystä hyväkseen käyttäviä menetelmiä varten pitää ensiksi määritellä mitta, jolla etäisyyttä tavoiteltavasta lopputuloksesta voidaan arvioida.

Lauselogiikan yhteydessä tämä mitta on helppo määritellä. Koska tällä sovellusalueella tehtävänä on loogisten lauseiden sieventäminen mahdollisimman yksinkertaiseen muotoon, voi edistymisen mittana käyttää yksinkertaisesti lauseen pituutta. Mitä lyhyempi lause, sitä lähempänä ratkaisua ollaan. Mitä enemmän lause lyhenee, sitä enemmän toimenpiteen soveltaminen edistää tehtävänratkaisua.

Yhtälönratkaisussa taas pyritään muotoa $x = N$ olevaan yhtälöön. Tässä tapauksessa pelkästään yhtälön pituus ei ratkaise sitä, kuinka lähellä lopullinen tavoite on.

Esimerkiksi yhtälöt $x = 32$ ja $3x = 6$ ovat yhtä pitkiä, mutta toinen vaatii vielä hieman matemaattisia ponnisteluita ratketakseen. Yhtälöiden etäisyyttä ratkaisusta kasvattavat mm. yhtälön oikealla puolella olevat x-termit ja useampien kuin yhden x-termin esiintyminen yhtälön vasemmalla puolella. Lisäksi tietysti itse termien lukumäärä vaikuttaa ratkaisevasti etäisyysmitan arvoon.

5.2.1. Paras ensin -etsintä

Nimensä mukaisesti paras ensin -etsintä *(best-first)* ahneesti valitsee toimenpiteistä parhaan ensiksi. Paremmuus päätetään äsken määritellyn heuristiikan avulla. Yksinkertaisimmillaan tutkitaan vain tehtävän kulloiseenkin tilaan suoraan soveltuvat toimenpiteet ja valitaan näistä se, joka johtaa eniten kohti lopullista ratkaisua (tai vähiten siitä poispäin). Menetelmä on siis hill-climbing -tyyppinen; myös sen suurin ongelma on hill-climbing -tyyppinen: löydetty huippu ei välttämättä olekaan kukkulan korkein kohta.

Tätä hakumenetelmää käyttäen kumppaniagentit, jotka käyttävät ID3:a, C4.5:ttä tai geneettistä algoritmia löytävät toimenpiteen yleensä alle viidessä sekunnissa, k-nearest neighbor -menetelmää käyttävä kumppani yhtälöitä ratkaistessaan 1 - 8 sekunnissa ja logiikan lauseita sieventäessään 15 - 60 sekunnissa. Yhtälöitä ratkaistessa jo näinkin yksinkertainen heuristiikka riittää useimmissa tapauksessa ohjaamaan etsintää kohti ratkaisua. Logiikan sovellusalueella sitä vastoin näin suoraviivainen heuristiikka ei toimi yhtä hyvin, vaan monissa tapauksissa ratkaisua etsivä kumppaniagentti ajautuu umpikujaan tai juuttuu ikuiseen silmukkaan. Esimerkiksi lausetta $\neg(p \,\&\, q) \,\&\, (\neg p \vee \neg q)$ sievennettäessä hyödyllisin vaihtoehto olisi de Morganin säännön soveltaminen $\neg(p \,\&\, q) \Leftrightarrow (\neg p \vee \neg q)$, vaikka loogisen lauseen pituus näin kasvaakin yhdellä merkillä. Tätä ratkaisua paras ensin -ideologiaa noudattavat kumppaniagentit eivät koskaan käytä, vaan soveltavat esimerkiksi loputtomasti vaihdantalakia jommankumman sulkulausekkeen sisällä.

Etsintää voi laajentaa siten, että ei tyydytä tarkastelemaan pelkästään ensimmäistä askelta, vaan käydään läpi kaikki kahden peräkkäisen toimenpiteen sarjat ja valitaan näistä paras. Tällöin vältetään monet edellä kuvatun kaltaiset umpikujat, ei kuitenkaan niitä kaikkia. Jos edellinen esimerkki muutetaan muotoon $\neg(p \,\&\, q) \,\&\, (\neg q \vee \neg p)$, pitäisi de Morganin säännön soveltamisen jälkeen käyttää vielä vaihdantalakia jommankumman sulkulausekkeen sisällä ennen kuin haluttua idempotenssilakia päästäisiin soveltа-

maan. Toisin sanoen kahden seuraavan askeleen tarkastelu ei riittäisi oikean ratkaisun löytämiseen. Toki kumppaniagentti voi käydä läpi kuinka monen toimenpiteen sarjoja tahansa, mutta ongelmaksi muodostuu jälleen hitaus, kuten täydellisen etsinnän yhteydessä. Itse asiassa kyseessä on täydellinen etsintä, jos tarkasteltavien askelten määrää riittävästi kasvatetaan. Käytännössä jo kaksi läpikäytävää askelta (runsas 20 sekuntia) on ehdoton maksimi, k-nearest neighbor -algoritmin yhteydessä jo toinenkin askel kuluttaa liiaksi aikaa.

5.2.2. Tapauskohtainen päättely

Tapauskohtainen päättely (*case-based reasoning*) on hyvin läheistä sukua kohdassa 3.5 esitellylle k-nearest neighbor -algoritmille: itse päättely tapahtuu samalla algoritmilla kuin luokittelukin, yleensä vain esimerkkitapausten kuvauksessa on eroa.

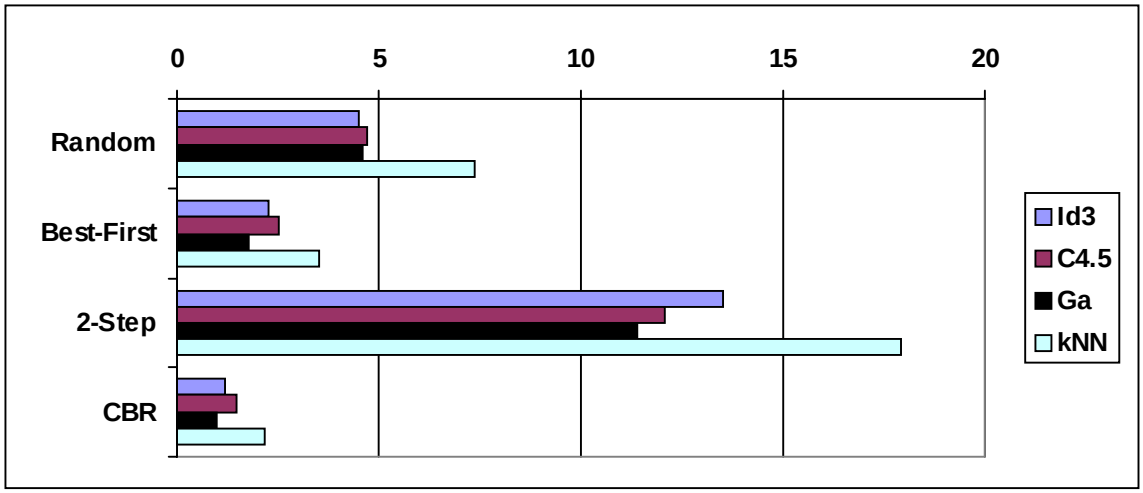
Tässä yhteydessä esimerkkitapaukset kuvaavat jotakin tehtävän tilaa ja sisältävät lisäksi tiedon siitä, kuinka monen toimenpiteen etäisyydellä tehtävän kokonaisratkaisu on. Kumppaniagentti vertaa löytämiään sallittuja toimenpiteitä tällaisiin mallitapauksiin ja valitsee toimenpiteen, jonka suorittamisen jälkeen tehtävä on tilassa, joka riittävästi muistuttaa sellaista mallitapausta, joka on lähellä ratkaisua. Valinta suoritetaan yhden esimerkkitapauksen perusteella, eikä etsitä useampia lähimpiä naapureita, koska tässä tilanteessa ei toimiteta luokittelua enemmistön äänen perusteella vaan pikemminkin luotetaan yhden ainoan esimerkin voimaan.

Tapauskohtainen päättely soveltuu toimenpiteiden valintaan hyvin nopeutensa puolesta. Lisäksi mallitapausvalikoimaa voi kasvattaa ohjelman suorituksen yhteydessä eli kumppaniagentille voi opettaa myös toimenpidesekvenssien muodostamista. Jokaisesta ratkaistusta tehtävästä voi muodostaa esimerkkitapauksia yhtä monta, kuin sen ratkaisuun on vaadittu toimenpiteitä.

5.3. Vertailua

Strategioiden vertailussa keskitytään pääasiassa kahteen seikkaan: toimenpiteen valintaan kuluneeseen aikaan sekä siihen tuottaako strategia toimenpideketjun, joka johtaa tehtävän ratkaisuun. Vertailu on suoritettu siten, että kumppaniagenttien ratkaistavaksi

on vuorollaan annettu kaikki EduAgentsin sovellusaluemoduulien sisältämät harjoitustehtävät. Eri oppimisalgoritmien tuottamia säännöstöjä ja eri valintastrategioita hyväksikäyttäen agentit ovat pyrkineet löytämään tehtäville ratkaisut. Jos agentti ei vielä kymmenennellä peräkkäisellä toimenpiteen soveltamisella ole päätynyt ratkaisuun, suoritus on keskeytetty ja katsottu yritys epäonnistuneeksi. Myös niissä tilanteissa, joissa aikaa ratkaisun etsimiseen on kulunut yli kymmenen minuuttia, suoritus on epäonnistuneena keskeytetty.

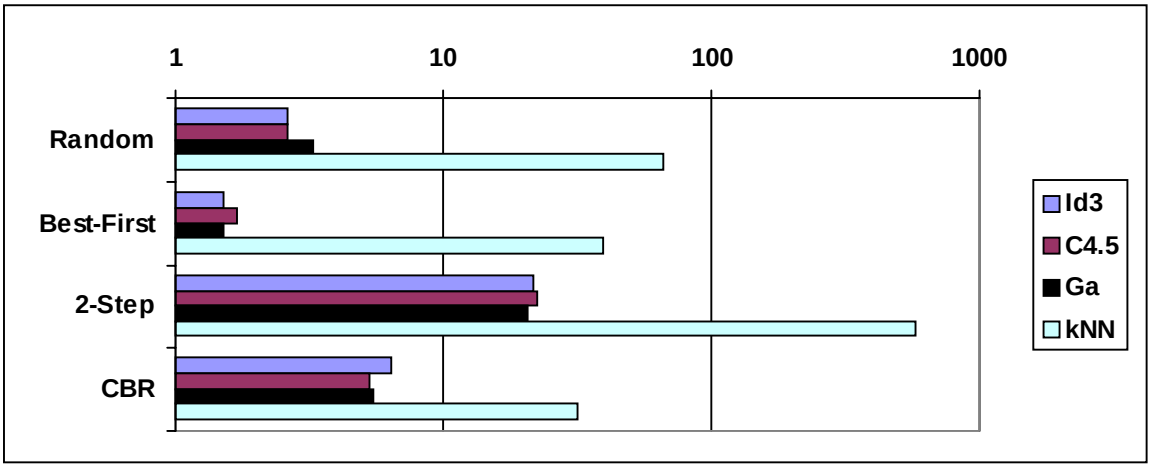


Kuva 5.1. Käytettävän toimenpiteen valintaan keskimäärin kulunut aika sekunteina yhtälönratkaisun sovellusalueella.

Kuva 5.1. sisältää tulokset aikavertailusta yhtälönratkaisun yhteydessä. Sovellusalueen kaikkien 140 tehtävän ratkaisuissa vaadittujen toimenpiteiden soveltamiseen käytetyistä ajoista on laskettu keskiarvo. Kuvassa Random tarkoittaa täydellisen etsinnän muotoa, jossa jokaisessa tehtävän vaiheessa soveltuvista toimenpiteistä on satunnaisesti valittu, mitä niistä käytetään. Best-First ja 2-Step ovat heuristisia hakuja, joista ensin mainitussa tarkastelun kohteena ovat tehtävään suoraan sovellettavissa olevat toimenpiteet ja jälkimmäisessä kahden peräkkäisen toimenpiteen sarjat. CBR on lyhenne sanoista Case-Based Reasoning ja tarkoittaa edellisessä kohdassa kuvattua etsintätapaa. Ga:lla viitataan geneettiseen algoritmiin ja kNN on lyhenne k-nearest neighbor -algoritmista.

Selvästi on nähtävissä kaksi seikkaa: ensinnäkin k-nearest neighbor -algoritmi on hitaampi kuin muut algoritmit ja toisaalta kahden toimenpiteen sarjojen etsiminen on hidasta. Tällä sovellusalueella kuitenkin kaikki oppimisalgoritmit ja toimenpiteen valintastrategiat toimivat käytäntöä ajatellen riittävän nopeasti, ehkä yhdistelmää kNN - 2-

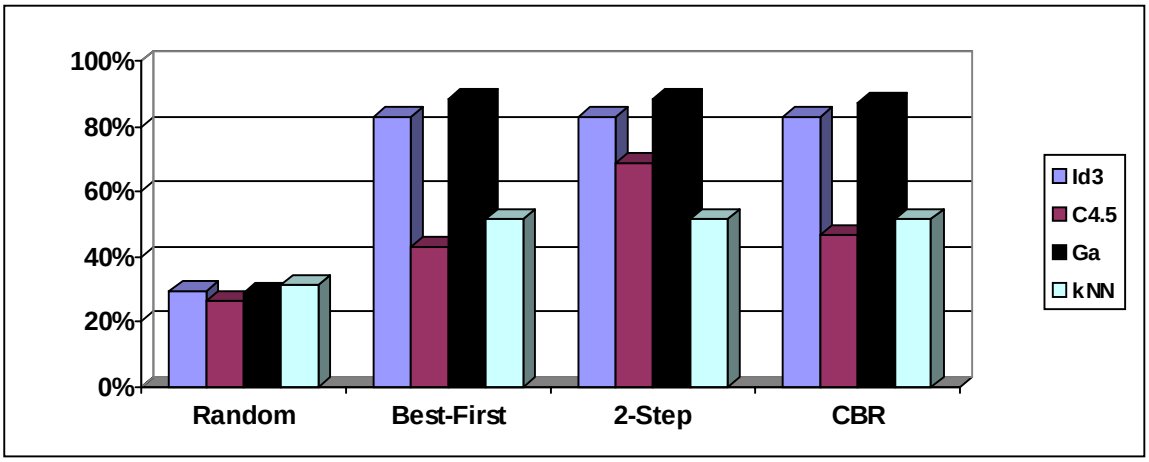
step lukuun ottamatta. Yllättävältä saattaa vaikuttaa satunnaisen valinnan hitaus paras ensin -hakuun verrattuna. Selitys tähän on se, että satunnaisesta toimenpiteen valinnasta johtuen tehtävä usein monimutkaistuu toimenpiteitä sovellettaessa ja seuraavat askeleet ovat tästä syystä monimutkaisempia ja hitaampia käsitellä.



Kuva 5.2. Käytettävän toimenpiteen valintaan keskimäärin kulunut aika sekunteina logiikan sovellusalueella.

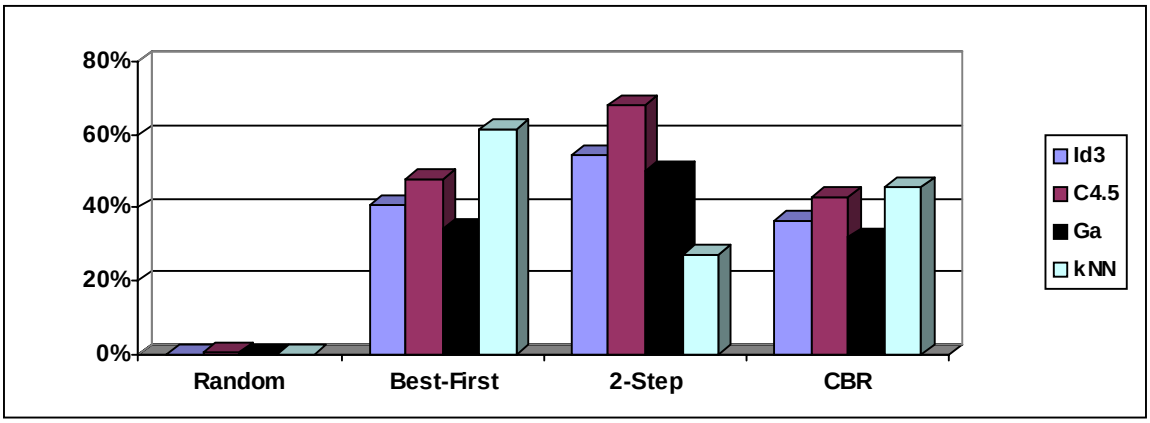
Logiikan sovellusalueella ratkaistavia tehtäviä oli 44 kappaletta. Aikavertailun tuloksiin (kuva 5.2.) pätee sama, mitä sanottiin yhtälönratkaisun yhteydessä, erot vain ovat huomattavasti suurempia (kuvan aika-asteikko on logaritminen). Varsinkin k-nearest neighbor -algoritmin ajoaikaisesta luokittelusta johtuva hitaus on tällä alueella tosiasia. On kyseenalaista, voiko tätä algoritmia lainkaan käyttää todellisissa tilanteissa, ei ainakaan yhdessä kahden askeleen sarjoja tarkastelevan valintastrategian kanssa. Kahta peräkkäistä toimenpidettä tutkivan valintastrategian käyttö yhdessä muiden oppimisalgoritmien kanssa logiikan sovellusalueella on mahdollisuuksien rajoissa: toimenpiteen valitseminen kestää keskimäärin 20 sekuntia. Uudemmilla, nopeammilla tietokoneilla tosin suoritusajat pysyvät selkeästi käyttökelpoisuusrajojen sisäpuolella.

Toinen kiinnostuksen kohde valintastrategioiden vertailussa on niiden tuottamat tulokset, eli kuinka useissa tapauksissa strategian käyttö johtaa tehtävän ratkeamiseen. Kuvassa 5.3. on esitetty yhtälöiden ratkeavuus eri strategioita käyttäen ja kuvassa 5.4. onnistumisprosentit logiikan lauseiden sievennyksessä. Nähdään, että satunnainenkin toimenpiteen valinta on tuottanut ratkaisun noin 30%:ssa tapauksista yhtälönratkaisussa, kun taas logiikan alueella vain yhdessä ainoassa tapauksessa 176:sta on löytynyt ratkaisu. Kolmen muun strategian tuottamat tulokset ovat yhtälönratkaisussa yllättävän sa-



Kuva 5.3. Kymmenellä yrityksellä ratkenneiden yhtälöiden prosentuaalinen osuus kaikista yhtälöistä eri valintastrategioita ja oppimisalgoritmeja käyttäen.

manlaisia, vain C4.5-oppimisalgoritmin tuottamaa säännöstöä käytettäessä tuloksissa on eroja. Myös C4.5 ja k-nearest neighbor -algoritmien kanssa saavutetut huonohkot tulokset tällä sovellusalueella kiinnittävät huomiota. Ilmeisesti nämä algoritmit kategorisesti tuomitsevat jonkin ratkaisuille välttämätön toimenpiteen laittomaksi edellä mainituin tuloksin. Geneettisen algoritmin tuottama säännöstö näyttää parhaiten toimivan tehtävän kokonaisratkaisun löytymisen hyväksi, niukasti paremmin kuin ID3:n muodostama päätöspuu.



Kuva 5.4. Kymmenellä yrityksellä ratkenneiden logiikan tehtävien prosentuaalinen osuus kaikista tehtävistä eri valintastrategioita ja oppimisalgoritmeja käyttäen.

Logiikan sovellusalueella tuloksissa on vaihtelua runsaammin kuin yhtälöiden yhteydessä. Kuvasta 5.4. löytyy kuitenkin selvä kaava: oppimisalgoritmeista parhaiten

menestyy k-nearest neighbor -algoritmi, sitten C4.5, ID3 ja viimeiseksi jää geneettinen algoritmi. Poikkeus tästä kaavasta on k-nearest neighbor -algoritmi yhdessä kahta toimenpidettä tarkastelevan valintastrategian kanssa. Tämän yhdistelmän huono tulos vertailussa johtunee suurelta osin siitä, että suoritus liikaa aikaa vievänä on keskeytetty ja hylätty monen tehtävän kohdalla. Valintastrategioista parhaiten toimii kahden askeleen sarjoja käsittelevä valinta, mutta kuten edellä mainittiin se on myös huomattavan hidas. Nopeammin ja silti kohtuullisin tuloksin toimivat paras ensin -etsintä ja tapauskohtainen päättely.

5.4. Yhteenveto

Se, että kumppaniagentti osaa käyttää toimenpiteitä sääntöjenmukaisesti ei pelkästään riitä tehtävän ratkaisun löytämiseen, vaan toimenpiteistä on pystyttävä lisäksi muodostamaan lopputulokseen johtava sarja. Täydellisen etsinnän käyttö tällaisen sarjan määrittämiseksi on etsintäavaruuden laajuuden takia käytännössä mahdotonta.

Heuristista hakua silmälläpitäen on määritelty mitta, jonka avulla etäisyyttä tehtävän lopullisesta ratkaisusta voi arvioida: lauselogiikan sovellusalueella mittana toimii yksinkertaisesti lauseen pituus, yhtälönratkaisussa taas pyritään muotoa x = N olevaan yhtälöön. Tätä mittaa hyväksi käyttäen kumppaniagentti voi valita laillisista toimenpiteistä sen, jonka soveltaminen johtaa eniten kohti kokonaisratkaisua. Tämä paras ensin -etsintä voi kuitenkin juuttua lokaaliin minimiin. Tilannetta voi parantaa tarkastelemalla yhden toimenpiteen sijasta kahden peräkkäisen toimenpiteen sarjoja, mutta tämä kasvattaa suoritusaikoja eksponentiaalisesti.

Myös tapauskohtaista päättelyä (*case-based reasoning*) voi soveltaa käytettävän toimenpiteen valintaan. Kumppaniagentti vertaa löytämiään sallittuja toimenpiteitä muistissaan oleviin mallitapauksiin ja valitsee toimenpiteen, jonka suorittamisen jälkeen tehtävä on tilassa, joka riittävästi muistuttaa sellaista mallitapausta, joka on lähellä ratkaisua. Menetelmän etuna on nopeuden lisäksi se, että esimerkkiaineistoa on mahdollista ohjelman suorituksen yhteydessä kasvattaa eli kumppaniagentti kykenee oppimaan myös toimenpiteiden järkevää yhdistämistä.

6. Yhdistäminen EduAgents-järjestelmään

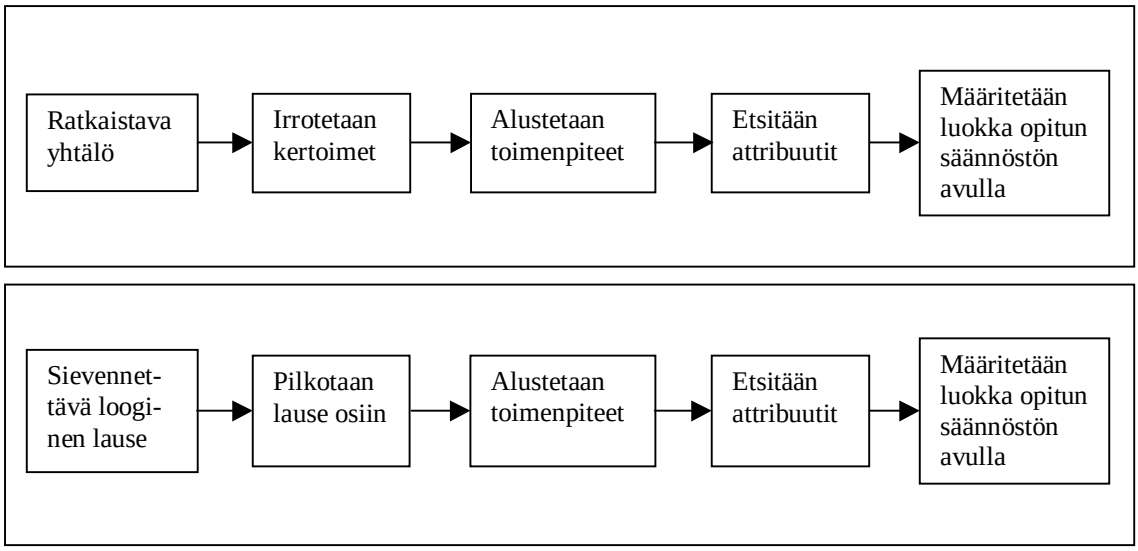
Nyt kun kumppaniagenteille on opetettu ensinnäkin toimenpiteiden laillista käyttöä ja toisaalta näiden toimenpiteiden yhdistämistä ketjuiksi siten, että ne tuottavat haluttuja tuloksia, pitää nämä koulutetut agentit vielä saada toimimaan olemassa olevassa järjestelmässä. Ensiksi, kohdassa 6.1. tarkastellaan sitä, miten oppinut agentti ratkaisee oppimaansa säännöstöä tai päätöspuuta hyväksikäyttäen toimenpiteen oikeellisuuden. Kohdassa 6.2. pohditaan mahdollisia tapoja kumppaniagentin jatkokoulutukseen, kohdassa 6.3. sitä, miten agentti perustelee vastausehdotuksensa sekä millaisia nämä perustelut ovat. Lopuksi, kohdassa 6.4. kiinnostuksen kohteena on erilaisten valintastrategioiden yhdistäminen järjestelmään.

6.1. Toimenpiteiden oikeellisuus

Kumppaniagentti muodostaa oman vastausehdotuksensa aina varsinaisen oppijan sovellettua jotakin toimenpidettä ja siten muutettua tehtävän tilaa. Alkuperäisessä järjestelmässä tehtävään tilaan sovellettavissa olevat toimenpiteet saadaan järjestelmän eksperttimoduulilta, joka taas käyttää hyväkseen sääntömuotoista sovellusaluetietämystä. Etukäteen opetettu kumppani ei tarvitse eksperttimoduulin enempää kuin sovellusalueen säännöstönkään apua sallittuja toimenpiteitä etsiessään. Ainoastaan oppimisalgoritmin tuottama päätöspuu (ID3), säännöstö (C4.5 ja geneettinen algoritmi) tai käytettävät esimerkkitapaukset (k-nearest neighbor -algoritmi) on ladattava ohjelman muistiin.

Kuvassa 6.1. on esitetty proseduuri, jolla tehtävään sovellettavissa olevat toimenpiteet etsitään etukäteisoppimista hyödyntäen. Proseduuri toimii osittain samalla tavoin kuin proseduuri, jota käytetään esimerkkitapauksia generoitaessa (kohta 4.1.). Ensiksi

looginen lause pilkotaan osalauseiksi, sen jälkeen sovitetaan tunnetut toimenpiteet näihin osalauseisiin. Yhtälöstä taas irrotetaan ensiksi kertoimet ja alustetaan toimenpiteet näillä numeroarvoilla. Sovellusaluekohtaista ennakkotietämystä siis tarvitaan myös opetettujen kumppanien yhteydessä, mutta ei enempää kuin lista kaikista käytettävissä olevista toimenpiteistä yleisessä muodossa. Näin saaduista toimenpiteistä muodostetaan esimerkkitapausten kaltaisia esityksiä muuttamalla ne attribuuttilistan muotoon. Nyt ollaan valmiita päättämään onko toimenpiteen käyttö tässä tilanteessa sallittua vai ei. Attribuuttilista annetaan käytössä olevan oppimisalgoritmille, joka säännöstönsä avulla luokittelee sen joko kategoriaan 'sallittu' tai 'ei sallittu'.



Kuva 6.1. Sovellettavien toimenpiteiden etsiminen opitun säännöstön avulla.

Esimerkiksi yhtälönratkaisun sovellusalueen ensimmäiseen harjoitustehtävään *x = 3 + 6* voidaan soveltaa toimenpiteitä *3 + 6, 6 + 3 ja -x*. Jos oppimisalgoritmi on hyvin koulutettu, pitäisi sen luokitella kaikki nämä kolme toimenpidettä sallituiksi. Logiikan sovellusalueen ensimmäinen tehtävä *(p v q) & (p v q)* tuottaa kolme erilaista osalausetta: *p v q, (p v q) ja (p v q) & (p v q)*. Näistä ensimmäinen tuottaa edelleen neljä alustettua toimenpidettä: p v q ⇔ q v p, p v q ⇔ ¬(¬p & ¬q), p v q ⇔ ¬p ⇒ q ja p v q ⇔ (¬p ⇒ q). Myös tässä tapauksessa kumppaniagentti toivottavasti hyväksyy kaikki neljä toimenpidekandidaattia. Koska attribuutteja on yhtälönratkaisussa käytössä kaksitoista ja logiikan alueella peräti kaksikymmentäyhdeksän kappaletta, muodostettuja attribuuttilistoja ei tässä esitetä.

Koska alkuperäisessä EduAgents-järjestelmässä on valittavana neljä kumppanioppijaa, ja toteutettuja oppimisalgoritmeja on myös neljä, oppimisalgoritmit on yhdistetty näihin kumppanihahmoihin. Alkuperäisen järjestelmän tarkoituksellinen jako heikkoihin ja vahvoihin kumppaneihin (ks. kohta 2.2.) on unohdettu, kumppanin antamien ehdotusten laillisuuden ratkaisee pelkästään se, kuinka hyvin käytetty oppimisalgoritmi on sääntönsä muodostanut. Agenttihahmojen luonteeseen tämä ei vaikuta muuten kuin vastausten perustelujen osalta, esimerkiksi yhteistyötyökalua käytettäessä agentit edelleen vastailevat luonteensa mukaisesti.

6.2. Jatkokoulutus

Kumppaniagenttien tietämystä toimenpiteiden oikeellisuudesta on mahdollista laajentaa aina kun EduAgents-ohjelmaa käytetään. Suoraviivainen tapa toteuttaa kumppanin jatko-opinnot on, aina tämän antaessa toimenpide-ehdotuksensa, tarkistuttaa se eksperttimoduulilla ja tallentaa toimenpide-ehdotus attribuuttilistan muodossa yhdessä ekspertiltä saadun laillisuustiedon kanssa muistiin. Tämä ei, ainakaan häiritsevästi, hidasta ohjelman suoritusta: ekspertti-moduuli tarkistaa yhden toimenpiteen aina reilusti alle sekunnissa. Näin kerättävä lisäinformaatio yhdistetään esimerkkitapauksiin, joiden avulla kumppanit on alun perin koulutettu ja suoritetaan oppimisalgoritmi uudelleen.

C4.5:n ja geneettisen algoritmin uudelleen opettaminen ei voi tapahtua ajoaikana, koska algoritmien suoritukset kestävät useita minuutteja. k-nearest neighbor -algoritmi sitä vastoin on laiska oppimismenetelmä, joten uusi esimerkkitapaus on sen käytettävissä heti kun se on tallennettu muistiin. ID3 on rajatapaus: oppimisalgoritmi kyllä pitää suorittaa etukäteen, mutta siihen ei kulu kovinkaan paljoa aikaa. Käytettäessä yhtälönratkaisun sovellusalueella datana alkuperäistä 176 esimerkkitapausta sisältävää aineistoa ID3 muodostaa päätöspuun noin viidessä sekunnissa. Logiikan 121 esimerkkitapausta käytettäessä aikaa kuluu kuitenkin jo runsaat puoli minuuttia. Yhtälönratkaisun yhteydessä ajoaikainen opettaminen on siis mahdollista, logiikan alueella taas ei, paitsi käytettäessä nopeampaa tietokonetta.

Miettimisen arvoinen on myös se seikka, kannattaako jokainen käsitelty toimenpide ottaa mukaan esimerkkiaineistoon. Oppimisalgoritmit saavuttavat keskimäärin 80 prosentin luokittelutarkkuuden alkuperäisen datan avulla, joten ne osaavat käyttää suurinta osaa läpikäytävistä toimenpiteistä oikein jo etukäteen. Tällaisten toimenpiteiden

tallentaminen esimerkkiaineistoon ei varsinaisesti lisää informaatiota, vaan pelkästään hidastaa oppimisalgoritmien suoritusta. Parempi ratkaisu lienee se, että tallennetaan ainoastaan ne tapaukset, jotka kumppani luokittelee väärin.

Eräs ongelma on se, että kaikki toimenpiteet, joita kumppaniagentti ehdottaa ovat tietysti sen mielestä sallittuja, joten käytettäessä ainoastaan näitä datana jatko-opinnoissa kumppani ei koskaan opi uusia toimenpiteitä. Se oppii ainoastaan sen, että jotkut toimenpiteet, joita se on pitänyt laillisina, eivät sitä olekaan. Ratkaisuna tähän voisi olla datan kerääminen myös varsinaisen oppijan tekemistä yrityksistä. Myös se, että käytetään eri menetelmällä koulutetun agentin tuottamaa dataa jatkokoulutuksessa, saattaa auttaa; se, minkä C4.5:n säännöt kieltävät, saattaa olla ID3:lle sallittua.

C4.5 ja geneettinen algoritmi ovat siinä mielessä samankaltaisia, että molempien oppimistulos on säännöstö. Lisäksi molemmilla tämän oppimistuloksen aikaansaaminen vie melko paljon aikaa, joten ajoaikana opettaminen on mahdotonta. On kuitenkin mahdollista jonkin verran parantaa niiden luokittelutarkkuutta ohjelman suorituksen aikanakin. Yksittäisiä esimerkkitapauksia voidaan käyttää sääntöinä sellaisenaan. C4.5:llä voi esimerkiksi olla sääntö, joka yhtälönratkaisussa yksinkertaisesti kieltää kaikki ryhmän nro 3 toimenpiteet eli sulkujen poiston. Tällöin positiivinen esimerkkitapaus, joka liittyy sulkujen poistoon jossakin tietyssä tilanteessa, säännöksi muutettuna parantaa algoritmin luokittelutarkkuutta tulevissa tapauksissa. Yleensä yksittäisen tapauksen käyttämistä suoraan sääntönä ei voi pitää oppimisena, koska tällöin ei yleistetä mitään. Tässä tapauksessa kuitenkin itse esimerkkitapausten muodostaminen, attribuuttien irrottaminen suoritettavasta toimenpiteestä, on melko yleistävää. Joten esimerkiksi sääntö, joka sallii sulkujen poistamisen tilanteessa (3 + 6), sallii sen myös tilanteessa (4x + 2x).

6.3. Perustelut

Oleellinen osa kumppaniagentin ratkaisuehdotusta on perustelu sille, miksi se otaksuu, että sen tarjoama toimenpide on kyseisessä tilanteessa luvallista suorittaa. Opetettu kumppani voi muodostaa perustelun yksinkertaisesti kirjoittamalla auki sen säännön, jonka perusteella se ehdottamansa toimenpiteen on hyväksynyt. Tarkemmin sanoen näin on tilanne C4.5:n ja geneettisen algoritmin kohdalla, ID3:n tapauksessa pitää algoritmin muodostamasta päätöspuusta toimenpiteen hyväksynyt haara ensin muuttaa säännöksi (ks. kohta 3.3.3.).

Ongelmallinen tilanne on vain k-nearest neighbor -algoritmin kohdalla, koska sen päätökset eivät perustu säännönkaltaisiin rakenteisiin, vaan esimerkkienemmistön voimaan. Tietysti perusteluna voi myös käyttää ratkaisuun vaikuttanutta lähintä naapuria tyyliin: "Koska tehtävässä x tämä toimenpide on sallittu, niin se on sallittu tässäkin tehtävässä". Asia ei kuitenkaan ole näin yksinkertainen, koska kyseinen tehtävä x on muistissa pelkästään attribuuttilistana, eikä selväkielisessä muodossa. Attribuuttilistasta taas ei ole apua (asiaan perehtymättömälle) ihmisoppijalle ja toisaalta sen palauttaminen selväkieliseksi on melko hankalaa. Lisäksi on mahdollista, että tämänkaltaisesta perustelusta on enemmän haittaa kuin hyötyä: tehtävä, jota käytetään esimerkkinä ei välttämättä ole varsinaiselle oppijalle tuttu ja se saattaa myös olla monimutkaisempi kuin ratkaistavana oleva tehtävä. Onkin parempi käyttää k-nearest neighbor -algoritmin yhteydessä vakioperusteluja alkuperäisen järjestelmän tapaan: "Ratkaisuaskeltyökalun ja oppikirjan mukaan voimme ehkä käyttää tätä…" tai "Valitsin ratkaisuaskeltyökalun antamista toimenpiteistä yhden…".

Muiden algoritmien tapauksessa perustelut eivät yleensä tuota ongelmia. Esimerkiksi sääntö `([nro = 4,jkv = ei],on)` muuntuu vaivattomasti perusteluksi "Lukujen yhteen- ja vähennyslasku on sallittu, jos yhtälön vasemmalla puolella ei ole kerto- tai jakomerkkejä". Tosin tapauksissa, joissa C4.5:n tai geneettinen algoritmi käyttää default-sääntöään toimenpide-ehdotusta hyväksyessään on myös turvauduttava vakioperusteluun.

6.4. Toimenpiteiden valinta

Alkuperäisessä järjestelmässä kumppaniagentti valitsi tarjoamansa toimenpide-ehdotuksen sen perusteella, mitä toimenpiteitä oppilaan kanssa oli aiemmin käyty läpi. Tämä proseduuri on tarkemmin selitetty kohdassa 2.2. Mikään ei tietenkään estä käyttämästä tätä vanhaa strategiaa myös opetettujen kumppanien yhteydessä; on kuitenkin houkuttelevaa koettaa opettaa agenteille toimenpiteiden hyväksyttävän käytön lisäksi myös tehtävän kokonaisratkaisun etsimistä. Ratkaisun löytämiseen tähtääviä strategioita on esitelty tutkielman luvussa 5.

Tapauskohtaisen päättelyn avulla toimenpiteitä valitsevalle agentille voi ohjelman suorituksen aikana kerätä lisää dataa eli sitä voi jatkokouluttaa. Tämä tapahtuu jokaisen harjoitustehtävän ratkeamisen jälkeen palaamalla ratkaisuaskel kerallaan kohti tehtävän

alkua ja kirjaamalla ylös jokaisessa vaiheessa tehtävän tila ja etäisyys eli askelten määrä lopulliseen ratkaisuun. Koska tapauskohtainen päättely kuuluu laiskojen oppimismenetelmien ryhmään on uusi informaatio välittömästi käytettävissä eikä erillistä opettamisvaihetta tarvita.

Agentille, joka käyttää kahden askeleen heuristista hakua (ja myös kaikille muille agenteille), voi asettaa aikarajan, jonka puitteissa sen on annettava ehdotuksensa. Sen on siis mahdollisesti keskeytettävä etsintänsä ja ehdotettava parasta aikarajaan mennessä löytynyttä toimenpidevaihtoehtoa.

Toimenpiteen valintastrategiat on yhdistetty opettajahahmoihin: valitessaan opettajan oppilas valitsee tietämättään myös tavan, jolla kumppaniagentti tehtäviä ratkaisee. Tämä ei vaikuta millään tavoin itse opettaja-agenttien toimintaan, avustavat opettajat ovat edelleen avustavia ja toteavat toteavia.

6.5. Yhteenveto

Tässä luvussa on käsitelty opetettujen kumppaniagenttien liittämistä olemassa olevaan EduAgents-järjestelmään. Etukäteen opetettu kumppani ei tarvitse sovellusalueen säännöstön apua sallittuja toimenpiteitä etsiessään. Ainoastaan oppimisalgoritmin tuottama päätöspuu (ID3), säännöstö (C4.5 ja geneettinen algoritmi) tai käytettävät esimerkkitapaukset (k-nearest neighbor -algoritmi) on ladattava ohjelman muistiin.

Ajoaikana on mahdollista parantaa agenttien oppimaa kykyä luokitella toimenpiteitä sallituiksi tai laittomiksi. Jokainen kumppanin tekemä ehdotus voidaan lisätä koulutuksen pohjana olevaan esimerkkiaineistoon ja suorittaa oppimisalgoritmi tällä laajemmalla aineistolla uudelleen.

Oleellinen osa kumppaniagentin ratkaisuehdotusta on perustelu sille, miksi sen tarjoama toimenpide on kyseisessä tilanteessa luvallista suorittaa. Opetettu kumppani voi muodostaa perustelun yksinkertaisesti aukikirjoittamalla sen säännön, jonka perusteella se ehdottamansa toimenpiteen on hyväksynyt. Poikkeustapaus on k-nearest neighbor -algoritmia käyttävä kumppani, koska kyseisen algoritmin päätökset eivät perustu säännönkaltaisiin rakenteisiin.

7. Lopuksi

EduAgents-järjestelmän sovellusalueiden tietämyksen opettaminen kumppaniagentille koneoppimismenetelmien avulla on käyttökelpoinen ratkaisu. Agentti oppii yhtälönratkaisun kaltaisilla yksinkertaisilla sovellusalueilla käyttämään toimenpiteitä jopa 90 prosenttisesti oikein ja lauselogiikan kaltaisilla monimutkaisemmillakin alueilla 80 prosenttisesti. Lisäksi jatkokoulutuksen avulla on kumppaniagenttien osaamista mahdollista parantaa.

Varsinkin logiikan sovellusalueen vaikeimpien harjoitustehtävien ratkaiseminen vie alkuperäisen EduAgents-järjestelmän tavalla toteutetulta kumppaniagentilta melkoisesti aikaa. Etukäteen oppivien algoritmien (ID3, C4.5 ja geneettinen algoritmi) avulla opetettuja kumppaniagentteja käyttämällä saavutetaan selvästi suurempi suoritusnopeus. Ajoaikana varsinaisen prosessoinnin suorittava k-nearest neighbor -algoritmikin suoriutuu yhtälönratkaisusta hyvin, mutta on lauselogiikan yhteydessä aivan liian hidas. Nopeuteen vaikuttaa tietysti myös valintastrategia, jota käytetään valittaessa soveltuvista toimenpiteistä se, mitä käytetään. Käytössä kuitenkin on useita strategioita, jotka suorittavat toimenpiteen valinnan nopeasti; tällaisia ovat esimerkiksi heuristinen paras ensin -etsintä (*best-first*) ja tapauskohtaiseen päättelyyn (*case-based reasoning*) perustuva valintamenetelmä.

Sääntöjä muodostavat algoritmit (C4.5 ja geneettinen algoritmi) oikeastaan jo luokitellessaan toimenpiteen lailliseksi tai laittomaksi perustelevat ratkaisunsa: perusteluna toimii sääntö, jonka mukaan ne tapausta luokitellessaan toimivat. ID3:n kohdalla tilanne on jokseenkin yhtä hyvä, sillä luokittelussa käytetyn päätöspuun haaran muuttaminen säännöksi ja sitä kautta perusteluksi on triviaali toimenpide. k-nearest neighbor -algoritmi sitä vastoin on tässä tehtävässä vaikeuksissa: lähimpiä naapureita kuvaavia attribuuttilistoja kun ei voi käyttää perusteluna.

Agenttien jatkokouluttamiseksi ohjelman suorituksen aikana voi sekä ihmiskäyttäjän että kumppaniagentin tekemät toimenpide-ehdotukset ja niitä vastaavat opettaja-agentin tarkistuksen tulokset tallentaa muistiin. k-nearest neighbor -algoritmilla uusi materiaali on välittömästi käytössään, C4.5 sekä geneettinen algoritmi on uudelleenopetettava kasvaneella aineistolla jälkeenpäin, ID3 on algoritmin nopeuden ansiosta rajatapaus: esimerkkiaineiston monimutkaisuudesta (attribuuttien lukumäärä) ja koosta riippuen uudelleenopetus saattaa olla mahdollista suorittaa ajoaikana.

Kaiken kaikkiaan parhaiten EduAgentsin yhteydessä käytettäväksi koneoppimisalgoritmiksi soveltuu ID3: se on nopea uudelleenkoulutuksessa, se tarjoaa agentille mahdollisuuden perustella ehdotuksensa, sen luokittelutarkkuus logiikan sovellusalueella on muita esiteltyjä algoritmeja parempi, eikä huono myöskään yhtälönratkaisun yhteydessä. Huolimatta suurimmasta luokittelutarkkuudestaan yhtälönratkaisussa ja automaattisesta jatko-oppimisestaan, k-nearest neighbor -algoritmi toimii heikoimmin yhdessä olemassa olevan järjestelmän kanssa. Tämä tuomio johtuu etupäässä algoritmin hitaudesta, jos ratkaistavana on hiemankin monimutkaisempi tehtävä. C4.5 ja geneettinen algoritmi toimivat järjestelmässä hyvin; niiden ainoa ongelma on itse oppimisalgoritmien hitaus ja sitä kautta jatkokoulutuksen ajoitus.

Toimenpiteiden valinnan yhteydessä eli etsittäessä tehtävän ratkaisuun johtavia toimenpidesekvenssejä, täydellinen etsintä on mahdotonta suorittaa järjellisessä ajassa. Tästä syystä etsintää pitää nopeuttaa käyttämällä sovellusalueen tietämystä hyväksi. Paras ensin -etsinnässä käytetään aina soveltuvista toimenpiteistä sitä, joka eniten johtaa kohti ratkaisua. Etsintää voi laajentaa siten, että tarkastellaan useampien toimenpiteiden sarjoja kerralla. Käytännössä kuitenkin jo kahden askeleen sarjojen läpikäynti on usein liian hidas tehtävä ajoaikana suoritettavaksi. Toimenpidettä valittaessa voi käyttää avuksi myös tapauskohtaista päättelyä. Tällöin käytössä pitää olla joukko esimerkkitapauksia, jotka kuvaavat tehtävien eri tiloja ja sisältävät lisäksi tiedon siitä, kuinka monen toimenpiteen etäisyydellä tehtävän kokonaisratkaisusta kussakin tapauksessa ollaan. Kumppaniagentti vertaa löytämiään sallittuja toimenpiteitä tällaisiin mallitapauksiin ja valitsee toimenpiteen, jonka suorittamisen jälkeen tehtävä on tilassa, joka riittävästi muistuttaa sellaista mallitapausta, joka on lähellä ratkaisua.

Sekä tapauskohtainen päättely että pelkästään yhtä askelta tarkasteleva paras ensin -etsintä soveltuvat toimenpiteiden valintaan hyvin nopeutensa puolesta. Paras ensin -etsinnän suurin ongelma on hill-climbing -tyyppisille algoritmeille tavanomainen välihuipuille juuttuminen. Tapauskohtaista päättelyä käytettäessä tätä ongelmaa ei ole ja

lisäksi sen käyttöön houkutteleva seikka on se, että mallitapausvalikoimaa voi kasvattaa ohjelman suorituksen yhteydessä eli kumppaniagenttien toimenpiteiden valintataitoja voi myös kehittää. Jokaisesta ratkaistusta tehtävästä voi muodostaa mallitapauksia yhtä monta, kuin sen ratkaisuun on käytetty toimenpiteitä. Huono puoli tapauskohtaisessa päättelyssä on sen vaatima aineisto, jota aluksi ei tietenkään ole, vaan se on erikseen generoitava.

Koneoppimismenetelmien liittäminen EduAgents-järjestelmän kumppaniagenttien toimintaan onnistuu jokaiselta osa-alueeltaan: agentit oppivat eri sovellusalueilla erottamaan hyväksytyt toimenpiteet virheellisistä, ne kykenevät kartuttamaan osaamistaan jälkeenpäin, ne vieläpä oppivat suunnittelemaan tehtävänratkaisua kokonaisuutena. Ja kaikki tämä saadaan tapahtumaan ohjelmaa käyttävän varsinaisen ihmisoppijan kannalta riittävän nopeasti.

Lähdeluettelo

Breiman, L., Friedman, J. H., Olhsen, R. A. and Stone, P. J. (1984) *Classification and regression trees*. Belmont, CA: Wadsworth International Group.

Boy, G. (1997) Software Agents for Cooperative Learning. In J. Bradshaw (Ed.) *Software Agents*. (pp. 223-245) Menlo Park, CA: AAAI Press.

Canut, M., Gouarderes, G. and Sanchis, E. (1999). The Systemion: A New Agent Model to Design Intelligent Tutoring System. In S. P. Lajoie and M. Vivet (Eds.) *Artificial Intelligence in Education: Open Learning Environments: New Computational Technologies to Support Learning, Exploration and Collaboration.* (pp. 54-63) Amsterdam: IOS Press.

Caruana, R. and Freitag, D. (1994). Greedy attribute selection. *Proceedings of the 11th International Conference on Machine Learning* (pp. 28-36). San Francisco: Morgan Kaufmann.

Cestnik, B. and Bratko, I. (1991). On estimating probabilities in tree pruning. *Proceedings of European Working Session on Learning 1991*, *Lecture Notes in Artificial Intelligence* 482 (pp. 138-150). Berlin: Springer-Verlag.

DeJong, K. A., Spears, W. M. and Gordon, D. F. (1993). Using genetic algorithms for concept learning. *Machine Learning*, 13, 161-188.

Dietterich, T. (1998). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7).

Elomaa, T. and Ukkonen, E. (1994), A geometric approach to feature selection. *Proceedings of European Conference on Machine Learning 1994, Lecture Notes in Artificial Intelligence* 784 (pp. 349-354). Berlin: Springer-Verlag.

Esposito, F., Malerba, D. and Semeraro, G. (1993). Decision tree pruning as a search in the state space. *Proceedings of European Conference on Machine Learning 1993, Lecture Notes in Artificial Intelligence* 667 (pp. 165-184). Springer-Verlag.

Hietala, P. and Niemirepo, T. (1997). Collaboration with software agents: what if the learning companion agent makes errors? *Proceedings of AI-ED 97 World Conference on Artificial Intelligence in Education* (pp. 159-166). IOS Press.

Holland, J. (1986). Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In R. Michalski, J. Carbonell and T. Mitchell (Eds.), *Machine learning: An artificial intelligence approach,* Volume 2 (pp. 593 -623). Los Altos, CA: Morgan Kaufmann.

Keeling, H. (1999) A Methodology for Building Intelligent Educational Agents. In S. P. Lajoie and M. Vivet (Eds.) *Artificial Intelligence in Education: Open Learning Environments: New Computational Technologies to Support Learning, Exploration and Collaboration.* (pp. 46-53) Amsterdam: IOS Press.

Knoblock, C. A. and Ambite, J. L. (1997) Agents for Information Gathering. In J. Bradshaw (Ed.) *Software Agents.* (pp. 347-373) Menlo Park, CA: AAAI Press.

Kononenko, I. (1994). Estimating attributes: Analysis and extensions of RELIEF. *Proceedings of European Conference on Machine Learning 1994, Lecture Notes in Artificial Intelligence* 784 (pp. 171-182). Berlin: Springer-Verlag.

Langley, P. (1994). Selection of relevant features in machine learning. *Proceedings of the AAAI Fall Symposium on Relevance* (pp. 140-144). New Orleans, LA: AAAI Press.

Langley, P. (1996). *Elements of Machine Learning*. San Francisco, CA: Morgan Kaufmann.

Liu, H. and Setiono, R. (1996). Feature selection and classification – a probabilistic wrapper approach. *Proceedings of the 9th International Conference on Industrial and Engineering Applications of AI and ES.* Fukuoka, Japan.

Lopez de Mantaras, R. (1989). ID3 Revisited: A distance-based criterion for attribute selection. In W. Ras (Ed.) *Methodologies for intelligent systems 4* (pp. 342-350). Elsevier Science Publishing.

Maes, P. (1994). Agents that reduce work and information overload. *Communications of the ACM*, 37(7), 31-46.

Mendel, G. (1866). Versuche über Pflanzen-Hybriden. *Verhandlungen des naturforschenden Vereines, Abhandlungen, Brünn 4* (pp. 3-47).

Mingers, J. (1989). An empirical comparison of pruning methods for decision-tree induction. *Machine Learning*, 4(2), 227-243.

Mitchell, T. (1997). *Machine Learning*. McGraw-Hill.

Mitchell, T., Caruana, R., Freitag, D., McDermott, J. Zabowski, D. (1994) Experience with a learning personal assistant. *Communications of the ACM 37(7),* 81-91.

Moore, A. W. and Lee, M. S. (1994). Efficient algorithms for minimizing cross validation error. *Proceedings of the 11th International Conference on Machine Learning* (pp. 190-198). San Francisco: Morgan Kaufmann.

Murthy, S. K., Kasif, S. and Salzberg, S. (1994). A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research,* 2, 1-32.

Niemirepo, T. (1997a). *Agenttiperustaisen sosiaalisen oppimisympäristön suunnittelu ja toteutus*, Pro gradu –tutkielma. Tietojenkäsittelyopin laitos, Tampereen yliopisto, Huhtikuu, 1997.

Niemirepo, T. (1997b). Designing agent-based learning environments. In Väliharju, T. (Ed.), *Digital Media as a Learning Environment* (pp. 51-66). Computer Centre / Hypermedia laboratory, University of Tampere, Tampere, 1997.

Nilsson, N. (1996) *Introduction to Machine Learning* (An early draft of a proposed textbook). Available as http://robotics.stanford.edu/people/nilsson/mlbook.html.

Quinlan, J. R. (1983). Learning efficient classification procedures and their application to chess endgames. In R. Michalski, J. Carbonell and T. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (pp. 463-482). Palo Alto, CA: Tioga Publishing Company.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1(1), 81-106.

Quinlan, J. R. (1987). Simplifying decision trees. *International Journal of Man-Machine Studies*, 27, 221-234.

Quinlan, J. R. (1988). Decision trees and multi-valued attributes. In Hayes, Michie and Richards (Eds.), *Machine intelligence 11* (pp. 305-318). Oxford, England: Oxford University Press.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann.

Quinlan, J. R. and Rivest, R. L. (1989). Inferring decision trees using the minimum description length principle. *Information and Computation* 80(3), 227-248.

Wettschereck, D. (1994). *A Study of Distance-Based Machine Learning Algorithms*, Ph.D. dissertation. Department of Computer Science, Oregon State University, Corvallis, OR.

White, A., Zhong, L. and Quinlan J. R. (1994). Bias in information-based measures in decision tree induction. *Machine Learning,* 15(3), 321-329.

Esimerkkitapausten attribuutit

Lauselogiikka

nimi	*arvojoukko*	*kuvaus*
luokka	[on,ei]	onko toimenpide sallittu
nro	[1 .. 10] (diskreetti)	toimenpideryhmän nro
tps	[on,ei]	onko toimenpide suluissa
tped	[v,ja,im,ek,neg,none]	toimenpidettä edeltävä merkki lauseessa
tpseur	[v,ja,im,ek,none]	toimenpidettä seuraava merkki lauseessa
taied	[on,ei]	onko toimenpiteen edellä v merkkejä
jaed	[on,ei]	onko toimenpiteen edellä & merkkejä
imed	[on,ei]	onko toimenpiteen edellä => merkkejä
eked	[on,ei]	onko toimenpiteen edellä <=> merkkejä
taiseur	[on,ei]	onko toimenpiteen jälkeen v merkkejä
jaseur	[on,ei]	onko toimenpiteen jälkeen & merkkejä
imseur	[on,ei]	onko toimenpiteen jälkeen => merkkejä
ekseur	[on,ei]	onko toimenpiteen jälkeen <=> merkkejä
as	[on,ei]	onko alustus a suluissa
bs	[on,ei]	onko alustus b suluissa
cs	[on,ei]	onko alustus c suluissa
bty	[on,ei]	onko alustuksen osa b tyhjä
cty	[on,ei]	onko alustuksen osa c tyhjä
aja	[on,ei]	onko alustuksessa a mukana & merkkiä
atai	[on,ei]	onko alustuksessa a mukana v merkkiä
aim	[on,ei]	onko alustuksessa a mukana => merkkiä
aek	[on,ei]	onko alustuksessa a mukana <=> merkkiä
bja	[on,ei]	onko alustuksessa b mukana & merkkiä
btai	[on,ei]	onko alustuksessa b mukana v merkkiä
bim	[on,ei]	onko alustuksessa b mukana => merkkiä
bek	[on,ei]	onko alustuksessa b mukana <=> merkkiä
cja	[on,ei]	onko alustuksessa c mukana & merkkiä
ctai	[on,ei]	onko alustuksessa c mukana v merkkiä
cim	[on,ei]	onko alustuksessa c mukana => merkkiä
cek	[on,ei]	onko alustuksessa c mukana <=> merkkiä

Yhtälönratkaisu

nimi	*arvojoukko*	*kuvaus*
luokka	[on,ei]	onko toimenpide sallittu
nro	[1 .. 10] (diskreetti)	toimenpideryhmän nro
samapuoli	[on,ei]	ovatko alustukset yhtälön samalla puolella
apuoli	[vasen,oikea,ei,both]	kummalla puolella yhtälöä alustus a on
bpuoli	[vasen,oikea,ei,both]	kummalla puolella yhtälöä alustus b on
jkv	[on,ei]	sisältääkö yhtälön vasen puoli / tai * merkin
jko	[on,ei]	sisältääkö yhtälön oikea puoli / tai * merkin
xoikea	[on,ei]	onko muuttujatermiä x yhtälön oikealla puolella
vlkm	[1,2,monta]	yhtälön vasemman puolen termien lkm
olkm	[1,2,monta]	yht. oikean puolen termien lkm
ax	[on,ei]	onko alustus a x:n kerroin
bx	[on,ei]	onko alustus b x:n kerroin
btyhja	[on,ei]	onko b alustamatta

Yhtälöt ja loogiset lauseet, joita on käytetty esimerkkiaineistojen muodostamisen lähtöjoukkoina

Loogiset lauseet

r v p v (q)
¬¬¬r
p & r
(p v q) & ¬p
q => ¬r
q = q & r <=> q
¬r => q
¬¬(q & ¬r)
¬(r => q) & p
¬(p v q) v (¬p & r)

Yhtälöt

-4x = 5
12x = -55
2x = 18
x - 3*5 = -12
5*6 + 3 + x = 8
x - 6 = 7*4 - 2
-3x + 8x = 4/2
x + 6/3 + 2x = 18
x + 5 - 4 = 2x
6 – x - 3 = 3x - 7
x = 3 - 12
x + 6 = -5 - 11
4x - 2x = 4
4x + 9x = 20 - 7
x - 0 = 6
x = 6 - 0*7
x = 12 + 21
x = 11*6

Valitut attribuutit

Yhtälönratkaisu	ID3	C4.5	GA	kNN
nro	x	x	x	x
samapuoli	x	x	x	
apuoli	x	x	x	x
bpuoli		x	x	x
jkv		x	x	
jko	x		x	x
xoikea		x		x
vlkm			x	x
olkm		x	x	x
ax	x	x	x	x
bx	x	x	x	
btyhja		x	x	

Logiikka	ID3	C4.5	GA	kNN
nro		x	x	x
tps	x	x	x	x
tped	x	x	x	x
tpseur		x	x	x
taied	x	x		x
jaed	x	x	x	
imed	x	x	x	x
eked	x	x	x	
taiseur	x	x	x	x
jaseur	x	x	x	x
imseur		x	x	
ekseur	x	x	x	x
bty	x	x	x	x
cty	x	x	x	
asu	x	x	x	
bsu		x	x	x
csu	x	x	x	
aja	x	x	x	x
atai	x		x	
aim	x	x	x	x
aek	x	x	x	
bja			x	x
btai	x	x	x	x
bim	x	x		
bek	x	x	x	x
cja	x	x	x	
ctai			x	
cim		x	x	
cek	x	x	x	