Kombinatorisen huutokaupan optimaalisen voittajan määrittäminen

Viljo Petäjä

4. huhtikuuta 2002

Tiivistelmä

Kombinatorisissa huutokaupoissa kaupattavien esineiden allokointi ostajille on taloudellisesti tehokkaampaa kuin perinteisissä monen esineen huutokaupoissa. Parhaan tuoton antavan tarjousjoukon löytäminen ei kuitenkaan onnistu polynomisessa ajassa, vastaava päätösongelma on NP-täydellinen. Tässä tutkielmassa esitellään pääpiirteittäin kombinatorisen huutokaupan voittajien löytämiseksi kehitetyn hakualgoritmin tärkeimmät toimintaperiaatteet.

1 Johdanto

Perättäisessä huutokaupassa (*sequential auction*) esineet huutokaupataan yksi kerrallaan. Voittajan löytäminen on helppoa. Jos ostaja haluaa saada haltuunsa tiettyjä kombinaatioita kaupattavista esineistä, tilanne hankaloituu. Määrittääkseen arvon kullekin esineelle ostajan täytyy osata arvioida, mitä esineitä hän tulee jatkossa saamaan. Jos myytävien esineiden lukumäärä kasvaa kovin isoksi, tarjouksia ei voida enää esittää rationaalisesti.

Rinnakkaisessa huutokaupassa (*parallel action*) kaikki esineet ovat tarjolla yhtä aikaa. Nyt ostaja saa tarjouksien tekemisen tueksi tietoa muiden ostajien tekemistä tarjouksista. Tämä toisaalta johtaa tilanteeseen, jossa jokaisen ostajan kannattaa huutaa omat tarjouksensa mahdollisimman myöhään.

Kombinatorisessa huutokaupassa ostaja voi asettaa tarjouksia myytävien esineiden kombinaatioille. Näin ostaja voi suoraan ilmaista myyjälle, millaisia kombinaatioita hän arvostaa. Enää ostajilla ei ole tarvetta spekulointiin. Myyjän kannalta tilanne mutkistuu, koska huutokaupan voittajien määrittäminen on laskennallisesti vaikea ongelma. Tässä tutkielmassa tarkastellaan kyseisen ongelman ratkaisemista [1]:ssä esitetyn pohjalta.

2 Voittajien määrittäminen kombinataorisessa huutokaupassa

Olkoon M joukko huutokaupattavia esineitä. Jokainen agentti (ostaja) i voi tehdä tarjouksen $b_i(S)$ mille tahansa kombinaatiolle $S \subseteq M$. Jos agentti i ei ole tehnyt tarjousta kombinaatiosta S, voidaan sanoa, että $b_i(S) = 0$. Riittää, jos huomioidaan vain korkein kullekin kombinaatiolle tehty tarjous. Formaalisti

$$\bar{b}(S) = \max_i b_i(S). \tag{1}$$

Jokaisen kombinaation kohdalla muut tarjoukset voidaan poistaa, samoin kuin tarjoukset, joilla on ei-positiivinen hinta. Olkoon n jäljellä jäävien tarjousten lukumäärä.

Voittajamääritys kombinatorisessa huutokaupassa voidaan muotoilla seuraavanlaisena ongelmana, jossa on tarkoitus maksimoida myyjän tuotto:

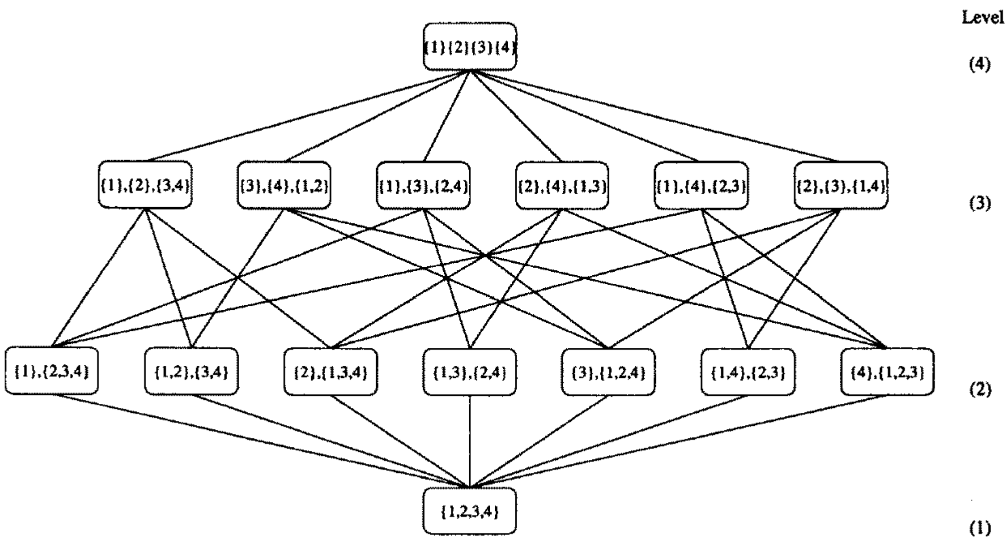
$$\max_{W \in A} \sum_{S \in W} \bar{b}(S), \tag{2}$$

jossa W on käypä lopputulos, eli sellainen tilanne, jossa jokainen huutokaupattava esine allokoidaan enintään yhdelle ostajalle. Olkoon $\mathbf{S} = \{S \subseteq M\}$, silloin käypien lopputulosten joukko on

$$A = \{W \subseteq \mathbf{S} | S, S' \in W \Rightarrow S \cup S' = \emptyset\}. \tag{3}$$

3 Täydellinen luetteloiminen

Jos jokaiselle esineiden kombinaatiolle on esitetty ainakin yksi tarjous, jolla on positiivinen hinta, allokaatioiden hakuavaruus näyttää kuvassa 1 esitetyltä graafilta.



Kuva 1: Esimerkki neljän esineen allokaatioavaruudesta. Graafin jokainen solmu vastaa yhtä mahdollista esineiden allokaatiota W.

Allokaatioiden lukumäärä kasvaa nopeasti sitä mukaa, kun agenttien määrää lisätään. Allokaatioden täsmällinen lukumäärä on

$$\sum_{q=1}^{m} Z(m,q), \tag{4}$$

missä $Z(m,q)$ on allokaatioden lukumäärä ehdolla, että on hyväksytty q tarjousta. Toisin sanoen $Z(m,q)$ on allokaatioden lukumäärä kuvan 1 kaltaisessa graafissa q:lla tasolla. Suure $Z(m,q)$ tunnetaan myös nimellä toisen lajin Stirlingin luku, ja sille pätee:

$$Z(m,q) = qZ(m-1,q) + Z(m-1,q-1), \tag{5}$$

missä $Z(m,m) = Z(m,1) = 1$. Tämä voidaan tulkita yhden uuden esineen lisäämisenä tilaan, jossa on jo $m-1$ esinettä. Termi $qZ(m-1,q)$ laskee uuden esineen lisäämisestä jo olemassa oleviin allokaatioihin syntyvät allokaatiot. Nyt on q eri vaihtoehtoa, koska olemassa olevissa allokaatioissa on q hyväksyttyä tarjousta. $Z(m-1,q-1)$ vastaa uuden esineen vaikutusta omana tarjouksena, joten nyt lasketaan allokaatiot, joissa on vain $m-1$ aikaisemmin hyväksyttyä tarjousta. Allokaatioiden lukumäärän asymptoottiselle käytökselle voidaan esittää seuraavat rajat:

$$O(m^m) \quad \text{ja} \quad \omega(m^{m/2}) \tag{6}$$

Kaikkien mahdollisten allokaatioden lukumäärä on siis hyvin suuri, eikä näin ollen niiden täydellinen luetteloiminen ole toimiva ratkaisu kombinatorisen huutokapan voittajamäärittelylle.

4 Ongelman yleisiä ominaisuuksia

Kaikille kombinaatioille ei välttämättä ole tehty tarjousta lainkaan, joten kaikkia mahdollisia allokaatioita ei tarvitse huomioda. Nyt herää kysymys, onko olemassa algoritmi algoritmi, joka löytää optimiratkaisun kobinatoriselle huutokaupalle polynomisessa ajassa annettujen tarjousten suhteen. Kombinatorisen huutokaupan voittajamäärittelyä vastaava päätösongelma on NP-täydellinen [1]. Polynomisessa ajassa toimivia approksimaatio-algoritmeja (virheellä on ongelman koosta riippumaton yläraja) löytyy vain asettamalla joitain rajoituksia tarjouksille (esim. asetetaan tarjouksen koolle maksimiarvo). Jos asetetaan tarpeeksi voimakkaita rajoituksi, optimiratkaisun etsimiselle voidaan esittää polynomisessa ajassa toimivia eksakteja algoritmeja. Rajoitusten keinotekoinen asettaminen saattaa kuitenkin huomattavasti pienentää myyjän tuottoa.

5 Optimaalinen hakualgoritmi

Tässä kappaleessa esitettävä hakualgoritmin kehittämistä motivoivat seuraavat seikat.

- Jotta huutokauppa olisi taloudellisesti tehokas, on sallittava tarjousten esittäminen kaikille kombinaatioille.
- Optimiratkaisun hakeminen approksimaation sijasta kasvattaa myyjän tuottoa. Toisaalta jos optimin hakeminen näyttää vievän liikaa aikaa, voidaan keskeyttää haku, ja saada käypä ratkaisu, joka lähestyy monotonisesti optimia laskenta-ajan kasvaessa.
- Vältetään turhien solmujen ja silmukoiden muodostamista, joita tulee haettaessa ratkaisua graafista (kuva 1).

[1] Tämä perustuu reduktiolle WSP:stä (*weighted set packing*)[2].

- Käytetään hyväksi sitä, että todellisia tarjouksia on huomattavasti vähemmän kuin kaikkia mahdollisia esinekombinaatioita. Käytännössä hakuavaruus on siis harva.

Yllä esitetyt ominaisuudet huomioidaan, jos ratkaisun etsimiseen käytetään seuraavassa kappaleessa kuvattua hakupuuta. Ennen varsinaista hakua pyritään esikäsittely-algoritmeilla pienentämään hakuavaruutta.

5.1 Hakupuu

Optimiratkaisun haku suoritetaan puussa, jossa polut (juuresta solmuihin) koostuvat sarjasta tarjouksia, ja jokainen polku loppuu, kun kaikki huutokaupattavat esineet sisältyvät jo polulle. Päättyessään polku vastaa käypää allokaatiota, ja siitä tulevaa tuottoa voidaan verrata tähän mennessä parhaaseen löytyneeseen. Määritelmiä:
U – polulla jo käytettyjen esineiden joukko
F – polulla vielä käyttämättömien esineiden joukko
g – annetussa polun pisteessä (solmussa) polulle jo kuuluvista tarjouksista kertynyt tuotto
Todettakoon, että huutokaupasta saatavaa tuottoa voidaan parantaa, jos myyjän suodaan mahdollisuus olla myymättä kaikkia esineitä. Tämä otetaan huomioon asettamalla jokaiselle esineelle ns. tyhjä tarjous, jos kyseisestä esineestä ei ole erillistä tarjousta.

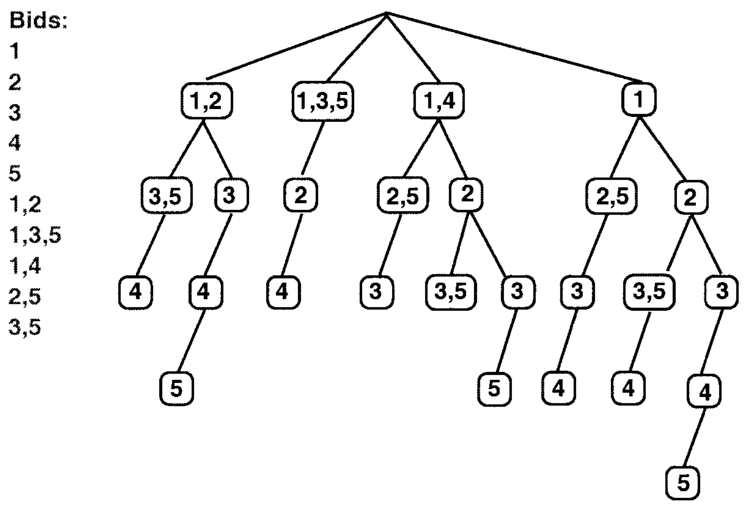
Jokaista relevanttia partitiota edustaa täsmälleen yksi hakupuun polku (juuresta solmuun), jos solmujen lapsille pätee:

- ei sisällä esineitä, jotka jo sisältyvät polulla oleviin solmuihin
- sisältää esineen, jolla varaamattomista esineistä pienin indeksi, merkitään jatkossa i^*:llä

Tällä tavalla saadaan kuvassa 2 esitetyn kaltainen hakupuu. Näin saatu haku generoi jokaisen käyvän tarjousjoukon, josta saadaan positiivista tuottoa täsmälleen kerran. Turhaa hakua ei näin ollen tapahdu. Se, että jokainen positiivisen tuoton antava käypä tarjousjoukko tarkistetaan, varmistaa optimiratkaisun löytymisen. Näin saatu hakupuu muodostaa hakuavaruuden

SEARCH1 haulle, joka voidaan toteuttaa joko syvyys-hakuna tai soveltamalla IDA* [3] hakua.

Syvyys-haulla käypä ratkaisy löytyy nopeasti, ja ratkaisu paranee monotonisesti, sillä algoritmi pitää kirjaa parhaasta löytyneestä ratkaisusta. Tämä mahdollistaa haun keskeyttämisen, jos käyttäjän mielestä siihen on kulunut tarpeettoman kauan aikaa. Algoritmilla on siis niin sanottu milloin-tahansa ominaisuus (*anytime feature*).



Kuva 2: Esimerkki hakupuusta, joka on muodostettu vasemmalla näkyvistä tarjouksista.

5.2 Hakupuun solmujen lasten generointi

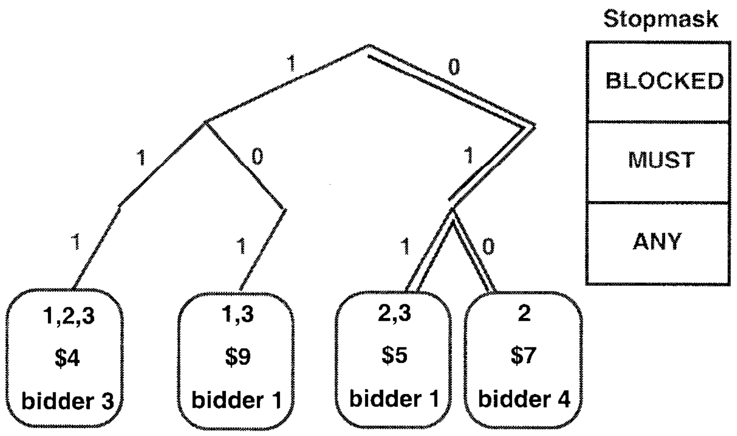
Yksinkertaisimmillaan kunkin solmun lapset löytyvät käymällä läpi kaikki tarjoukset ja tarkistamalla niiden soveltuvuus lapsiksi. Lapset voidaan löytää tehokkaammin käyttämällä SEARCH2 hakua.

SEARCH2 toteutetaan syvyys-hakuna tarjouspuuksi (*bidtree*) kutsutussa binääripuussa, jossa kutakin tarjousta vastaa yksi polku. Jokaista tarjouspuun polkua voidaan ajatella binäärivektorina, jossa 1 vastaa vasemmalle ja 0 oikealle haarautumista. Esimerkiksi tarjousta, joka koostuu esineistä 1 ja 3, vastaa vektori $(1, 0, 1)$, olettaen, että kaupattavana on vain 3 esinettä.

Hakua tarjouspuussa rajataan *stopmask* vektorin avulla. Tässä m-pituisessa vektorissa jokaista esinettä kohti on yksi hakua ohjaava muuttuuja, joka voi saada minkä tahansa alla esitetystä kolmesta arvosta.

$$\begin{aligned} stopmask[i] &= BLOCKED, \text{jos}\, i \in U \\ stopmask[i*] &= MUST \\ stopmask[i] &= ANY, \text{muutoin} \end{aligned}$$

Jos $stopmask[i] = BLOCKED$, tasolla i SEARCH2 ei voi edeta vasemmalle. $stopmask[i] = MUST$ estää haun etenemisen oikealle, ja arvo ANY ei aseta mitään rajoituksia. Kuvassa 3 on havainnollistettu SEARCH2:n hakupuuta.



Kuva 3: Esimerkki SEARCH2 hakupuusta.

5.3 Esikäsittely-algoritmit

PRE1. Kutakin kombinaatiota kohti pidetään vain paras tarjous.

PRE2. Karsitaan todistettavasti kilpailukyvyttömät tarjoukset. Tarjous karsitaan, jos löytyy kyseisen tarjouksen osittava joukko tarjouksia, joista tulee suurempi tai yhtäsuuri tuotto. Pahimmassa tapauksessa (tarjous sisältää kaikki esineet) osituksen etsiminen vastaa SEARCH1 hakua. Tämä voidaan estää asettamalla yläraja Γ osittavien tarjousten lukumäärälle tai yläraja Φ

tutkittavaan tarjoukseen sisältyvien esineiden lukumäärälle. Yleensä ylärajan Φ käyttö on tehokkaampaa.

PRE3. Jaetaan alkuperäinen tarjousjoukko kytkettyihin komponentteihin (eri komponenteilla ei ole yhteisiä esineitä), jolloin SEARCH1 ja PRE4 suoritetaan kullekin komponentille erikseen.

PRE4. Merkitään kilpailukyvyttömät tarjousparit [2]. Esimerkiksi tarjousparista $\bar{b}.\{1,3\} = \$5$, $\bar{b}.\{2,5\} = \$4$ saadaan huonompi tuotto kuin parista $\bar{b}.\{1,2\} = \$3$, $\bar{b}.\{3,5\} = \$7$. Tarjouksia, jotka muodostavat SEARCH1 hakupuuhun lisätyn tarjouksen kanssa kilpailukyvyttömän parin, ei enää käytetä lisätyn tarjouksen alapuolella hakupolun muodostamiseen. Laskentanopeuden optimoimiseksi otetaan käyttöön ylärajat Γ' ja Φ', jotka vastaavat PRE2:n ylärajoja Γ ja Φ.

5.4 IDA* ja heuristiikat

Yllä esitetty hakupuu voidaan tutkia tehokkaammin menettämättä optimiominaisuutta käyttämällä syvyys-haun sijasta IDA* (*iterative deepening* A* [3]) hakustrategiaa, johon on lisätty kombinatorista huutokauppaa varten kehitetyllä heuristiikalla [1]. Keskeisessä osassa on funktio $h(F)$, joka antaa ylärajan sille, miten paljon voittoa annetulle hakupolulle kuuluvat vielä allokoimattomat tarjoukset F voivat tuottaa annetun solmun kohdalla. Nyt voidaan laskea jokaiselle hakupuun solmulle kokonaistuoton yläraja $f = g + h(F)$, jos kyseinen solmu sisältyy ratkaisuun. IDA* -haun toimintaperiaate: Jokaisella iteraatiokierroksella suoritetaan syvyys-haku, jonka aikana jätetään tutkimatta haarat, joiden kokonaiskustannusten yläraja f alittaa annetun raja-arvon. Seuraavalla iteraatiokierroksella raja-arvona käytetään edellisen kierroksen suurinta ylärajaa, joka jäi raja-arvon alapuolelle. Perus IDA* -haun iterointia käytetään, kunnes se löytää ensimmäisen käyvän ratkaisun. Tämän jälkeen raja-arvo asetetaan aina yhtä suureksi kuin parhaan löydetyn ratkaisun tuotto.

Seuraavalla sivulla on esitettu hakualgoritmin toimintaa kuvaava pseudokoodi. Algoritmin 1 IDA* iteraatiosilmukan kohdassa 3 oleva kerroin 0.95 on todettu kokeellisesti parhaaksi valinnaksi [1]. Algoritmin 2 kohdassa 5 käydään

[2]Yleisemmin voidaan tutkia tarjousmonikkoja, laskennallisista syistä rajoitutaan vain parien merkitsemiseen.

läpi kaikki annetulle solmulle generoidut lapset. $bid.S$ on kunkin lapsen (tarjouksen) sisältämien esineiden joukko, bid kuvaa itse tarjousta ja $bid.\bar{b}$ sen hintaa.

globaali muuttuja: f_{raja}

Algortmi 1. IDA* kombinatoriselle huutokaupalle

1. $f_{raja} := \infty$
2. **Silmukka:**
   (a) $voittajat$, $f_{uusi} :=$ Syvyys-haku$(M, \emptyset, g)$
   (b) **jos** $voittajat \neq null$ **palauta** $voittajat$
   (c) $f_{raja} := \min(f_{uusi}, 0.95 \cdot f_{raja})$

Algoritmi 2. Syvyys-haku(F,$voittajat$,g**)**

1. Laske $h(F)$
2. **jos** $g + h(F) < f_{raja}$ **palauta** $null, g + h(F)$
3. **jos** SEARCH2 ei löydä solmun lapsia
   (a) $f_{raja} := g$
   (b) **palauta** $voittajat$, g
4. $Tuotto := 0$, $Voittajat := null$, $f_{seuraava} := 0$
5. **jokaiselle** tarjoukselle bid, joka on solmun lapsi
   (a) $ratkaisu$, $f_{uusi} :=$ Syvyys-haku$(F - bid.S$,$voittajat \cup \{bid\}$,$g + bid.\bar{b})$
   (b) **jos** $ratkaisu \neq null$ ja $f_{uusi} > Tuotto$
      i. $Tuotto := f_{uusi}$
      ii. $Voittajat := ratkaisu$
   (c) $f_{seuraava} := \max(f_{seuraava}, f_{uusi})$
6. **jos** $Voittajat \neq null$ **palauta** $Voittajat$, $Tuotto$
   muutoin palauta $null$, $f_{seuraava}$

Funktion $h(F)$ laskemiseksi on kehitetty kaksi alla esitettyä heuristiikkaa.

H1.

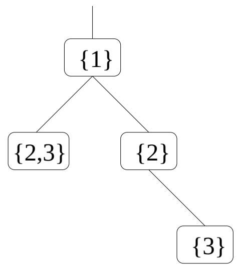
$$h(F) = \sum_{i \in F} c(i) \quad \text{jossa} \quad c(i) = \max_{S|i \in S} \frac{\bar{b}(S)}{|S|} \quad , \tag{7}$$

missä $\bar{b}(S)$ on esineen i sisältävän tarjouksen hinta ja $|S|$ siihen tarjoukseen kuuluvien esineiden lukumäärä.

H2.
Formaalisti sama kuin **H1.** Nyt osa $c(i)$:stä lasketaan uudelleen, kun hakupolkuun lisätään uusi tarjous, koska uuden tarjouksen sisällyttäminen polulle voi poissulkea joidenkin muiden tarjousten käytön.

Ylärajan $h(F)$ laskemiseksi käytetään heuristiikkaa **H2.**

5.5 Esimerkki haun toiminnasta



Kuva 4: Yksinkertainen hakupuu.

Valaistaan yllä kuvatun hakualgoritmin toimintaa yksinkertaisella kuvassa 4 esitetyllä esimerkillä, jossa $\{1\} = \$1, \{2\} = \$1, \{2\} = \$1$ ja $\{2,3\} = \$3$. Haun alkaessa $F = M$, jolloin $h(M) = c(1) + c(2) + c(3) = \max(1/1) + \max(1/1, 3/2) + \max(1/1, 3/2) = 4$. Ensimmäisellä kerralla Syvyys-haku algoritmissa kohdan 2 ehto ($4 < \infty$) täyttyy, jolloin palataan IDA*-silmukkaan, ja asetetaan $f_{raja} = 4$. Nyt Syvyys-haku algoritmissa päästään kohtaan 5 asti. Hakupuun juuren lapseksi tulee tarjous $\{1\}$, nyt $F = \{\{2\}, \{3\}, \{2,3\}\}$ ja $h(F) = 3, g_{\{1\}} = 1$. Seuraavaksi haku menee solmuun $\{2,3\}$ (SEARCH2 antaa tämän ennen tarjousta $\{2\}$). Syvyys-haku algoritmin kohdan 2 ehto ei täyty. Kohdassa 3 todetaan että lapsia ei löytynyt. Haku lopetetaan, koska päästiin hakupuun lehteen [3], voittajaehdokkaiksi asetetaan tarjoukset $\{1\}$ ja $\{2,3\}$. Jatkossa käytettävä f_{raja} on suoraan parhaasta löytyneestä ratkaisusta saatava tuotto, nyt $f_{raja} = 4$. Kun hakua jatketaan solmuun $\{2\}$, huomataan, että $g_{\{2\}} + h(\{\{3\}\}) < f_{raja}$, ja haku lopetetaan.

6 Numeeriset kokeet

Huonoimmassa tapauksessa SEARCH1:n kayttämä aika kasvaa polynomisesti tarjousten ja eksponentiaalisesti esineiden lukumäärän myötä. Jotta saataisiin jonkinlainen kuva algoritmin tehokkuudesta käytännössä, sen toimivuutta testattiin käyttäen neljää eri tarjousjakaumaa:

1. Satunnaisjaukuma: Valitaan kullekin tarjoukselle satunnainen kokonaisluku j väliltä $[1, m]$. Sisällytetään tarjoukseen satunnaisesti j esinettä ja hinnaksi määrätään satunnaisluku väliltä $[0, 1]$.
2. Painotettu satunnaisjakauma: Kuten 1., mutta tarjouksen hinta arvotaan väliltä $[0, j]$.
3. Tasajakauma: Jokaiselle tarjoukselle arvotaan yhtä monta esinettä. Hinnaksi tulee satunnaisluku väliltä $[0, 1]$.
4. Vaimeneva jakauma: Arvotaan tarjoukseen yksi esine. Lisätään uusi esine todennäköisyydellä α. Toistetaan tätä, kunnes uutta esinettä ei lisätä tai kaikki esineet kuuluvat jo tarjoukseen. Hinnaksi valitaan luku nollan ja tarjoukseen kuuluvien esineiden lukumäärän väliltä.

Koska IDA* oli kaksi kertaluokkaa yksinkertaista syvyys-hakua nopeampi, keskitytään sen käytöstä saatuihin tuloksiin. Parhaan kompromissin esikäsittelyyn ja varsinaiseen hakuun kuluvan ajan käyttöön päästiin asettamalla $\Phi = 8$ ja $\Phi' = 5$. PRE3 säästää huomattavasti aikaa jakaumien 3 (tasajakauma) ja 4 (vaimeneva jakauma) tapauksessa. Toisaalta jakaumien 1 (satunnainen) ja 2 (painotettu satunnainen) tapauksessa tarjoukset muodostivat

[3] Jos tarjousten hinnat olisi valittu eri tavalla haun lopetuksen syynä solmuun $\{2,3\}$ olisi voinut myös olla Syvyys-haku algoritmin kohdan 2 ehdon täyttyminen. Tässä tapauksessa ratkaisu olisi $\{\{1\},\{2\},\{3\}\}$.

vain yhden kytketyn komponentin, joten PRE3 oli hyödytön. Yleisesti ottaen satunnainen ja painotettu satunnainen jakauma olivat hakualgoritmille helpompia verrattuna vaimenevaan jakaumaan tai tasajakaumaan.

Testaamalla syvyys-haun milloin-tahansa -ominaisuutta kävi ilmi, että käytännössä suurin osa tuotosta saatiin jo haun varhaisessa vaiheessa. Hakuaikaa lisäämällä saatu tuoton kasvu oli varsin pientä.

7 XOR-tarjoukset

Tähän asti on oletettu, että tarjouksille pätee: $\bar{b}(S \cap S') \geq \bar{b}(S) + \bar{b}(S')$, eli tarjoukset ovat yli-additiivisia. Esimerkiksi jos agentti 1 on tehnyt seuraavanlaiset tarjoukset $b_1(\{1\}) = 5$, $b_1(\{2\}) = 4$ ja $b_1(\{1,2\}) = 7$, tarjoukset eivät enää ole super-additiivisia. Tällaisessa tapauksessa tarjoukset voidaan muotoilla XOR-kytkettyinä tarjouksina. Nyt muodostettaessa hakupuuta täytyy ottaa huomioon, että XOR-kytketyt tarjoukset ovat toisensa poissulkevia, joten ne eivät voi olla samalla polulla. Lisäksi lieventämällä preprosessoinnissa tapahtuvaa tarjousten karsintaa edellisessä kappaleessa kuvattu hakualgoritmi on helposti muokattavissa käsittelemään myös XOR-tarjouksia.

Viitteet

[1] T. Sandholm, Algorithm for Optimal Winner Determination in Combinatorial Auctions

[2] R.M. Karp, Reducibility among combinatorial problems, in: R.E. Miller, J.W. Thatcher (edit.), Complexity of Computer Computations, Plenum, New York, 1972, ss. 85-103

[3] R.E. Korf, Depth-first iterative-deepening: an optimal admissible tree search, Artificial Intelligence 27 (1)(1985) 159-175