Paikalliseen hakuun pohjautuva deterministinen $(2-2/(k+1))^n$ k-SAT-algoritmi

Tommi Vainikainen
thv@iki.fi

26.3.2003

Tiivistelmä

Artikkelissa [1] esitetään algoritmi k-SAT-ongelman ratkaisuun. Algoritmin suoritusaikaraja on $(2-\frac{2}{k+1})^n$. Algoritmi on täydellinen, deterministinen ja perustuu paikalliseen hakuun. Nämä ominaisuudet yhdessä tekevät tästä erittäin hyvän algoritmin hajautettuun laskentaan. Algoritmin ideana on jakaa kaikki mahdolliset totuusjakelut sopivaan määrään Hamming-aliavaruuksia ja tutkia nämä aliavaruudet paikallisella haulla.

1 Johdanto

Toteutuvuustarkistimia käytetään nykyään moniin ongelmiin, joista ehkä yleisimpänä on verifiointi (esim. piirikaavion oikeellisuuden tarkistaminen). Kaikki lauselogiikan lauseet voidaan muuntaa konjuktiiviseen normaalimuotoon, joita nimitetään k-CNF-lauseiksi, jossa k suurimman disjunktion kokoon lauseessa. Siksi monet toteutuvuustarkistusalgoritmit kehitetäänkin juuri pelkästään k-CNF-lauseille ja siten puhutaan k-SAT-algoritmeista.

Ongelmana k-SAT kuuluu NP-luokkaan (kun $k \geq 3$). Monesti kuitenkin lauseet, joiden toteutuvuutta halutaan tutkia, nousevat insinöörityön ongelmista ja siten lauseissa on eräänlaista redundanssia, jonka huomioimisella voidaan nopeuttaa tarkistusta.

Näitä k-SAT-algoritmeja voidaan luokitella eri tavoin riippuen niiden käytöksestä. Osa algoritmeista (kuten tässä esitelty) perustuu paikalliseen hakuun, jolloin haku käynnistetään uudestaan useilla eri alkuarvoilla ja se tarkistaa vain osan mahdollisista totuusjakeluista. Tällaisissa menetelmissä kätevää on laskennan yksinkertainen rinnakkaistaminen käynnistämällä useampi paikallinen haku samaan aikaan tarvittaessa vaikka hajautetussa laskentaklusterissa. Globaalit menetelmät sen sijaan eivät käynnisty uudelleen vaan ne etsivät ratkaisua käyden koko hakuavaruuden läpi samalla suorituskerralla.

Kaikki algoritmit eivät käy läpi hakuavaruutta kokonaan, jolloin ei tutkittavan lauseen toteutumattomuutta (UNSAT) ei voida taata täysin. Joskus kuitenkin tällaiset algoritmit takaavat toteutumattomuuden tietyllä todennäköisyydellä, joka voidaan valita käytettävissä olevan laskenta-ajan pohjalta. Tällaiset algoritmit kuitenkin löytävät usein keskimäärin nopeammin yhden totuusjakelun, jolla lause toteutuu, ja ovat siksi tietyissä tilanteissa erittäin käteviä. Näitä kutsutaan epätäydellisiksi algoritmeiksi täydellisen vastakohtana.

Lisäksi jotkut algoritmi ovat satunnaisia, jolloin algoritmi toimii satunnaisesti joissain tilanteissa (esim. haun haarautuessa). Nykyään tunnetaan satunnaisia algoritmeja, joiden keskimääräinen suoritusaika on parempi kuin deterministisillä. Deterministisillä algoritmeilla voidaan kuitenkin vastaus saada tietyssä ajassa, kun satunnaisilla algoritmeilla on sitten niiden nopeiden tapausten vastakohtana usein erittäin hitaita tapauksia.

Tässä esiteltävä k-SAT-algoritmi on siis paikalliseen hakuun pohjautuva, deterministinen ja täydellinen. Artikkelissa [1] on kiinnitetty huomiota pelkästään eksponentin kantaluvun pienentämiseen, jossa onkin saatu erittäin hyviä tuloksia, mutta käytännön toteutukseen tarvitaan sellaisten tutkimustulosten yhdistämistä, joissa polynomisella lauseen käsittelyllä saadaan suoritusajan eksponenttia pienennettyä, jolloin käytännön tapauksista löytyvä redundanssi saadaan hyödynnettyä laskennan nopeutumisena.

2 Algoritmi

Alussa selitetään käytettävät notaatiot, sen jälkeen esitellään paikallinen haku yksinkertaisessa muodossa. Paikalliselle haulle annetaan parametri, joka kertoo miltä Hamming-etäisyydeltä annetusta totuusjakelusta tutkitaan toteutuvuus. Jotta tällä paikallisella haulla saataisiin tutkittua lauseen toteutuvuus ylipäätään millä tahansa sijoituksella, täytyy mahdollisten totuusjakeluiden avaruus jakaa aliavaruuksiin, joiden säteenä on aina tietty Hamming-etäisyys. Tällainen aliavaruuksiin jako selitetään paikallisen haun jälkeen. Lopuksi muodostetaan näistä lopullinen algoritmi ja analysoidaan sen suoritusaika. Lopuksi tutkitaan tietyn tyyppisiä lauseita ja kuinka paikallisen haun haarautumista voidaan rajoittaa valitsemalla haarautumispiste oikein.

2.1 Notaatioita

$F_{|l=1}$ tarkoittaa literaalin l asettamista todeksi eli F:stä poistetaan sellaiset klausuulit, joista löytyy l ja $\overline{l}$ poistetaan kaikista muista klausuuleista.

Totuusjakeluita käsitellään binäärilukuina. Kaikkien n-pituisten binäärilukujen joukko on *Hamming-avaruus*, jota merkitään $H_n = \{0,1\}^n$. Hamming-etäisyys kahden alkion välillä on sellaisten paikkojen määrä, joissa nämä alkiot eroavat. Alkion a ympäristössä r-säteinen *pallo* on niiden alkioiden joukko, joiden Hamming-etäisyys alkiosta a on enintään r.

Koodi, jonka pituus on n, on H_n osajoukko. Koodin C *peittosäde* r määritellään

$$r = \max_{u \in \{0,1\}^n} \min_{v \in C} d(u,v),$$

jossa $d(u,v)$ on Hamming-etäisyys u:n ja v:n välillä. Peittosäde on siis pienin sellainen säde, jolla koodin alkioiden ympäristöjen unioni peittää koko avaruuden. *Normalisoitu peittosäde* ρ on $\rho = r/n$.

Siispä r-säteinen *peittokoodi* tarkoittaa sellaista joukkoa, jossa jokainen H_n:n alkio kuuluu johonkin r-säteiseen palloon, jossa pallon keskipiste kuuluu koodiin.

2.2 Paikallinen haku

Paikallinen haku ottaa parametreiksi aloitustotuusjakelun ja tutkii sen r-säteisen ympäristön tarkastaen löytyykö sieltä sellaisia malleja, joilla annettu lause toteutuu. Mahdollisten tutkittavien vaihtoehtojen määrä voidaan siis ajatella pallon tilavuutena

$$V(n,r) = \sum_{i=0}^{r} \binom{n}{i}.$$

Jos normalisoitu peittosäde $\rho = r/n$ on välillä $0 < \rho \leq 1/2$, voidaan tilavuutta $V(n,r)$ arvioida seuraavasti [2]

$$\frac{1}{\sqrt{8n\rho(1-\rho)}} \cdot 2^{h(\rho)n} \leq V(n,r) \leq 2^{h(\rho)n}, \tag{3}$$

jossa $h(\rho) = -\rho log_2 \rho - (1-\rho) log_2 (1-\rho)$ on binäärientropiafunktio. Tässä on siis tilavuudelle sellaiset rajat, jotka eroavat ainoastaan polynomisen tekijän verran funktiosta $2^{h(\rho)n}$.

Hakualgoritmin ei tarvitse tutkia kaikkia $V(n,r)$ totuusjakelua, vaan hakupuuta voidaan karsia tutkimalla vain sellaiset uudet totuusjakelut, joissa muutetaan joku tutkittavaksi otetun totuusjakelun epätodesta klausuulista poimittu literaali.

Lemma 1 *Kun F on lause, a ei ole sen malli ja C on epätosi F:n klausuuli tällä totuusjakelulla (ja tutkitaan paikallisella haulla etäisyyttä r). Tällöin, jotta F saataisiin todeksi a:n r-ympäristössä, on oltava literaali l siten, että $F_{|l=1}$ saadaan todeksi a:n $r-1$-ympäristössä.*

Ylläoleva muunnos on sama kuin toteutuvuuden tarkistus F:lle $a_{|l=1}$:n $r-1$-ympäristössä, joka on helppo ymmärtää intuitiivisesti, koska nyt a:sta muutetaan yhtä literaalia (Hamming-etäisyys alkuperäisen a:n ja $a_{|l=1}$:n välillä on yksi, joten sädettä vähennetään yhdellä). Ainoa ero F:n ja a:n muuntamisessa on, että a:ta muuttamalla voidaan sama literaali muuttaa takaisin seuraavassa rekursion tasossa (eli informaatiota ei häviä), kun taas F:n tapauksessa löydetään jotain oikeasti hyödyllistä etsimällä uusi epätosi klausuuli ja valitsemalla uusi literaali sieltä, koska F:stä on poistettu kaikki toteutumattomaan totuusjakeluun viittaavat literaalit.

$Search(F,a,r)$
1. Jos kaikki F:n klausuulit ovat tosia a:ssa, palauta tosi.
2. Jos $r \leq 0$, palauta epätosi.
3. Jos F sisältää tyhjän klausuulin, palauta epätosi.
4. Valitse (jollain säännöllä) a:ssa epätosi klausuuli C:
   Jokaiselle literaalille $l \in C$:
      Jos $Search(F_{|l=1}, a, r-1)$ on tosi, palauta tosi.
   Muutoin palauta epätosi.

On helppo todistaa induktiolla, että tämä algoritmi löytää lauseelle mallin jos sellainen on olemassa. Itse asiassa lemma 1 on induktioaskel sellaisenaan.

Esimerkki 1 *Olkoot* $F = (A_1 \vee A_2) \wedge (\overline{A_1} \vee \overline{A_3}) \wedge (\overline{A_3})$, $a = 111$ *ja* $r = 1$. *Lasketaan siis* $Search(\{\{A_1, A_2\}, \{\overline{A_1}, \overline{A_3}\}, \{\overline{A_3}\}\}, 111, 1)$.

Tällä totuusjakelulla ($A_1 = A_2 = A_3 = \top$*) molemmat jälkimmäiset klausuulit ovat epätosia, joten käsitellään suoraan algoritmin kohtaa neljä. Nyt valitaan jompi kumpi klausuuleista, ja tällä kertaa tutkitaan (*$\{\overline{A_1}, \overline{A_3}\}$. *Jotta tämä saadaan todeksi, täytyy jompi kumpi literaaleista (*$\overline{A_1}$ *tai* $\overline{A_3}$*) muuttaa todeksi.*

Nyt algoritmi haarautuu tutkimaan $Search(\{\{\overline{A_3}\}, \{\overline{A_3}\}\}, 111, 0)$. *Myös tämä lause on annetulla totuusjakelulla epätosi ja koska* $r = 0$ *ei tämän syvemmälle haarauduta. Nyt mennään toiseen haaraan eli* $Search(\{\{A_1, A_2\}\}, 111, 0)$, *joka toteutuu ja siten palautetaan tosi.*

Tässä huomioitavaksi jää, että algoritmi ei sievennä lainkaan lauseita eli muistiin jäisi todellakin sama klausuuli kahteen kertaan toisessa haarautumisessa vaikka se tällaisella joukko-notaatiolla näyttääkin hassulta. Käytännön toteutuksissa täytyy analysoida kuinka paljon aikaa kannattaa käyttää lauseiden sieventämiseen verrattuna siitä saatavaan hyötyyn.

Tämä algoritmin aikavaatimus on $poly(n) \cdot k^r$, sillä rekursion syvyys on r ja jokaisella syvyydellä algoritmilla k haaraa, koska klausuulissa, jonka suhteen haarauduntaan on korkeintaan k literaalia. Muistetaan, että käsitellään k-CNF-lauseita.

Eksponenttiosa k^r voi olla huomattavasti pienempi kuin pallon tilavuus. Esimerkiksi $r = n/2$ ja $k = 3$, $V(n,r) \geq 2^{n-1}$ kun taas $3^r < 3^{n/2} \approx 1.733^n$. Nyt valitsemalla triviaali peittokoodi, eli $\{a_0 = 000..0, a_1 = 111..1\}$, jonka peittosäde on $r = n/2$, ja ajamalla $Search(F, a_0, n/2)$ ja $Search(F, a_1, n/2)$ saadaan yksinkertainen deterministinen k-SAT-algoritmi, jonka suoritusaikaraja on $poly(n) \cdot 1.733^n$.

2.3 Peittokoodi

Nyt täytyy muodostaa hyvä peittokoodi, niin saadaan suoritusaikarajaa alemmas. Selvästi r-säteisellä peittokoodilla $\mathcal{C}$ on raja $|\mathcal{C}| \cdot V(n,r) \geq 2^n$ (eli peittokoodin $määrä \times tilavuus$ on oltava vähintään koko avaruuden kokoinen). Käyttämällä aiemmin (3) saatua ylärajaa, saamme

$$|\mathcal{C}| \geq \frac{2^n}{V(n,r)} \geq \frac{2^n}{2^{h(\rho)n}} = 2^{(1-h(\rho))n},$$

jossa $\rho = r/n$ on normalisoitu peittosäde ja $0 < \rho < 1/2$.

Lemma 2 *Kaikille $n \geq 1$ ja r, on olemassa peittokoodi $\mathcal{C}$, jonka pituus on n, peittosäde enintään r ja koko enintään*

$$\lceil n \cdot 2^n / V(n,r) \rceil. \tag{5}$$

Todistus. Tämä osoitetaan todennäköisyysargumentilla. Valitaan H_n:stä takaisinpanolla $n \cdot 2^n / V(n,r)$ satunnaista alkiota. Nyt näytetään, että nämä alkiota muodostavat enintään r-säteisen peittokoodin nollasta poikkeavalla (itse asiassa suurella) todennäköisyydellä. Olkoot a alkio H_n:ssä. Se kuuluu satunnaisesti valitun b:n r-ympäristöön todennäköisyydellä $V(n,r)/2^n$. Todennäköisyys, että a ei kuulu mihinkään r-ympäristöön on

$$(1 - V(n,r)/2^n)^{n \cdot 2^n / V(n,r)} \leq e^{-n},$$

käyttämällä sarjakehitelmistä tuttua $1 + x \leq e^x$ kaikille x. Eli valitut alkiot muodostavat r-säteisen peittokoodin vähintään todennäköisyydellä $1 - 2^n \cdot e^{-n}$, jonka raja-arvo on 1, kun $n \to \inf$.

Seurauslause 1 *Olkoot $0 < \rho < 1/2$ ja $\beta(n) = \sqrt{n\rho(1-\rho)}$. Kaikille n on olemassa n-pituinen peittokoodi $\mathcal{C}$, jonka säde on korkeintaan ρn ja koko enintään $n\beta(n) \cdot 2^{(1-h(\rho))n}$.*

Todistus. Seuraa lemmasta 2 ja rajasta (3):

$$n \cdot 2^n / V(n,r) \leq n\beta(n) \cdot 2^{(1-h(\rho))n}.$$

Lemma 3 *Olkoot $n \geq 1$, $0 < \rho < 1/2$ ja $\beta(n) = \sqrt{n\rho(1-\rho)}$. Tällöin n-pituinen peittokoodi, jonka säde on korkeintaan ρn ja koko korkeintaan $n^2\beta(n) \cdot 2^{(1-h(\rho))n}$ voidaan muodostaa ajssa $poly(n) \cdot 2^{3n}$.*

Tällainen voidaan muodostaa yksinkertaisella ahneella algoritmilla, jossa valitaan aina sellainen alkio lisää joukkoon, jonka ympäristössä on eniten vielä peittämättömiä alkioita ja päivittämällä muille alkioille vielä peittämättömien alkioiden lukumäärää. Tällöin iteraatio kestää $poly(n) \cdot 2^{2n}$ ja iteraatioita on korkeintaan 2^n.[3]

Lemma 4 *Olkoot $n \geq 1$ sekä $d \geq 2$ sen jakaja ja $0 < \rho < 1/2$. Tällöin on polynomi q_d siten, että n-pituinen peittokoodi, jonka säde on korkeintaan ρn ja koko korkeintaan $q_d(n) \cdot 2^{(1-h(\rho))n}$ voidaan muodostaa ajassa $q_d(n) \cdot (2^{3n/d} + 2^{(1-h(\rho))n})$.*

Todistus. Jaetaan H_n:n n-bittiset sanat d:n osaan, jolloin jokaisen pituus on n/d. Lemman 3 mukaan muodostetaan peittokoodi $\mathcal{C}'$ säteeltään $\rho n/d$ $H_{n/d}$:lle. Nyt $\mathcal{C}$ muodostetaan $\mathcal{C}'$:n alkioiden summina eli kaikkina mahdollisina konkatenaatioina.

$\mathcal{C}'$ muodostaminen kestää $O(q(n) \cdot 2^{3n/d})$. $\mathcal{C}$:n koko on enintään $(n^2\beta(n) \cdot 2^{(1-h(\rho))n/d})^d = n^{2d}\beta(n)^d \cdot 2^{(1-h(\rho))n}$.

On helppo havaita, että $\mathcal{C}$ todella on peittokoodi, sillä jokaiselle $a \in H_n$ jaetaan a osiin $a_1, a_2, \ldots, a_d$, jolloin on olemassa $w_i \in \mathcal{C}'$ etäisyydellä $\rho n/d$ jokaiselle a_i. Konkatenaatio $w_1 w_2 \ldots w_d \in \mathcal{C}$ on etäisyydellä ρn a:sta.

Esimerkki 2 *$n = 4$ ja $\rho = 1/2$ eli etsitään peittokoodia, jonka säde $r = \rho n = 2$. Jaetaan tämä kahtia ($d = 2$) ja etsitään $\mathcal{C}'$, joka on $\{00, 11\}$ (eli $\mathcal{C}'$ peittää koko H_2-avaruuden säteellä $r' = 1$). Nyt mikä tahansa H_4 alkio voidaan jakaa kahteen kahden bitin pituiseen osaan, joista molemmat ovat korkeintaan etäisyydellä 1 jommasta kummasta $\mathcal{C}'$ alkiosta. Nyt siis tämä H_4 alkio on korkeintaan etäisyydellä 2 jostain joukon $\mathcal{C} = \{0000, 0011, 1100, 1111\}$ alkiosta.*

Lemmassa 4 jokainen koodisana muodostuu vakiomäärästä d osia, joista jokainen on n/d pitkä. Nämä osat muodostetaan ahneella algoritmilla. Seuraavassa lemmassa koodisanat muodostuvat lineaarisesti kasvavasta määrästä n/b osia, joista jokainen on b pituinen.

Lemma 5 *Olkoot $\delta > 0$ ja $0 < \rho < 1/2$. On olemassa vakio $b = b(\delta, \rho)$ siten, että jokaiselle $n = bl$, n-pituinen peittokoodi, jonka säde on enintään ρn ja koko enintään $2^{(1-h(\rho)+\delta)n}$, voidaan muodostaa polynomisessa tilassa käyttämällä polynominen aika per koodisana.*

Tässä siis δ on mielivaltaisen pieni vakio, josta riippuu suoritusaikaan tullut lisäys $2^{\delta n}$.

Todistus. Kun b valitaan siten, että $b\beta(b) \leq 2^{\delta b}$, seurauslauseen 1 nojalla on olemassa koodi $\mathcal{C}' = \mathcal{C}'(\delta, \rho)$ pituudeltaan $b = b(\delta, \rho)$ ja säteeltään korkeintaan ρb ja jonka koko on enintään $2^{(1-h(\rho)+\delta)b}$. Nyt n/b tällaisen $\mathcal{C}'$ alkion summa on haluttu koodi.

Esimerkki 3 *Olkoot $\delta = 1/2$ ja $\rho = 1/3$. Tällöin valitsemalla $b = 3$, jolloin saadaan $b\beta(b) = b\sqrt{b\rho(1-\rho)} = 3\sqrt{\frac{2}{3}} \approx 2.449 \leq 2^{\delta b} = 2^{1.5} \approx 2.828$. Seurauslauseen 1 nojalla siis on olemassa koodi $\mathcal{C}'$, jonka pituus on $b = 3$ ja säde korkeintaan $\rho b = 1$. Tämän koodin koko on enintään $\lfloor 2^{(1-h(\rho)+\delta)b} \rfloor \approx \lfloor 3.35 \rfloor = 3$.*

Etsitään nyt peittokoodia H_{21}:lle arvolla $\rho = 1/3$. Tiedetään, että $\{000, 111\}$ on peittokoodi H_3:lle säteellä $r = 1$ eli tämä on ehdot täyttävä peittokoodi $\mathcal{C}'$.

Nyt H_{21} voidaan peittää konkatenoimalla seitsemän kappaletta tämän b-pituisen peittokoodin alkioita peräkkäin. Näin saadaan muodostettua peittokoodi $\mathcal{C} = \{000, 111\}^7$, joka peittää avaruuden H_{21} halutulla säteellä $r = \rho n = 7$.

Kun valitaan b algoritmia toteutettaessa ja esilasketaan mahdollisimman hyvä b-pituinen peittokoodi, päästään polynomiseen muistinkäyttöön ja koko avaruuden H_n peittävän koodin koodisanat voidaan laskea polynomisessa ajassa b-pituisesta peittokoodista.

2.4 Pääalgoritmi

Algoritmi ottaa syötteenä k-CNF-muotoisen lauseen F, jossa on n muuttujaa. Aluksi peitetään H_n palloilla, joiden säde on ρn, käyttämällä lemmaa 4. Peittämiseen tarvitaan $B(n,\rho,d) = q_d(n) \cdot 2^{(1-h(\rho))n}$ palloa ja peitteen muodostaminen kestää $T_1(n,\rho,d) = q_d(n) \cdot (2^{3n/d} + 2^{(1-h(\rho))n})$, jossa q_d on polynomi. Jokaista palloa kohti suoritetaan paikallinen haku, joka kestää $T_2(n,\rho) = p(n) \cdot k^{\rho n}$ per pallo, jossa p on polynomi. Kokonaissuoritusaika on siis

$$T_1(n,\rho,d) + B(n,\rho,d) \cdot T_2(n,\rho). \tag{8}$$

Eksponenttiosuus voidaan minimoida valitsemalla $\rho = 1/(k+1)$. (Voidaan havaita esim. laskemalla derivaatan nollakohta Mathematicalla.) Lisäksi valitsemalla d riittävän suureksi, saadaan kaavan (8) ensimmäinen termi pienemmäksi kuin toinen. On helppo havaita, että esim. $d = 6$ tulee ensimmäiseksi termiksi $2^{3n/d} = (\sqrt{2})^n \approx 1{,}414^n < (2 - \frac{2}{k+1})^n$ kun $k \geq 3$.

Yksinkertaisuuden vuoksi voidaan olettaa n olevan jaollinen d:llä. Lisäämällä vakiomäärä uusia muuttujia kasvattaa ajoaikaa korkeintaan vakiotekijällä.

Toisaalta jos d on suuri, kasvaa q_d ja pyöristyksessä lisätyistä muuttujista aiheutuva tekijä. Kuitenkin tekijä q_d (polynomi) on häviävän pieni verrattuna eksponentiaalisesti kasvavaan tekijään, joten d valinnalla ei ole merkitystä kun n on suuri kunhan vain termi $2^{3n/d}$ ei pääse dominoimaan suoritusaikaa.

Pääalgoritmi.

1. Aseta $\rho = 1/(k+1)$.
2. Käytä lemmaa 4 arvolla $d = 6$ muodostamaan n-pituinen peittokoodi $\mathcal{C}$ säteeltään korkeintaan ρn.
3. Jokaiselle koodisanalla $a \in \mathcal{C}$ suorita $Search(F,a,\rho n)$. Palauta *tosi* jos jokin suorituksista palauttaa *tosi*, muutoin *epätosi*.

Lasketaan vielä tämän ajoaika. Aluksi heitetään pois pienemmän eksponenttitermit, jolloin saadaan jonkin $poly(n)$ ja dominoivan eksponenttitermin tulona suoritusaika. Sen jälkeen pienellä logaritmipyörittelyllä saadaan haluttu lopputulos:

$$\begin{aligned}
&T_1(n,\rho,d) + B(n,\rho,d) \cdot T_2(n,\rho) \\
&= q_d(n) \cdot (2^{3n/d} + 2^{(1-h(\rho))n}) + q_d(n) \cdot 2^{(1-h(\rho))n} \cdot p(n) \cdot k^{\rho n} \\
&= poly(n) \cdot 2^{n(1+\frac{1}{k+1}\log_2\frac{1}{k+1}+\frac{k}{k+1}\log_2\frac{k}{k+1}+\frac{1}{k+1}\log_2 k)} \\
&= poly(n) \cdot 2^{n(1-\frac{1}{k+1}\log_2(k+1)+\frac{k}{k+1}\log_2 k-\frac{k}{k+1}\log_2(k+1)+\frac{1}{k+1}\log_2 k)} \\
&= poly(n) \cdot 2^{n(1+\log_2\frac{k}{k+1})} \\
&= poly(n) \cdot \left(\frac{2k}{k+1}\right)^n \\
&= poly(n) \cdot \left(2 - \frac{2}{k+1}\right)^n.
\end{aligned} \tag{9}$$

Teoreema 1 *Jokaiselle $\varepsilon > 0$ ja kokonaisluvulle k pääalgoritmia voidaan muuttaa siten, se toimii ajassa $poly(n) \cdot (2 - 2/(k+1) + \varepsilon)^n$ polynomisin tilavaatimuksin.*

Ideana tässä on käyttää lemman 4 sijaan lemmaa 5 arvolla $\delta = \log_2(\varepsilon(k+1)/(2k)+1)$. Näin suoritusajaksi tulee tuo jo mainittu. Valitsemalla δ näin ε:n funktiona, voidaan ε valita tekemällä kompromissi suoritusajan ja muistinkulutuksen suhteen.

2.5 Paikallisen haun optimointi

Aiemmin esitellyssä paikallisessa haussa valittiin *jokin* epätosista klausuuleista haarautumista varten välittämättä tarkemmin klausuulin valinnasta. Tällöin kompleksisuus oli $poly(n) \cdot k^r$. Valitsemalla haarautumisklausuuli paremmin, voidaan tätä rajaa parantaa ja siten myös pääalgoritmin kokonaissuoritusaikaa. Tässä kappaleessa esitellään, kuinka kompleksisuus saadaan rajattua $poly(n) \cdot 2,848^r$ aiemman $poly(n) \cdot 3^r$ sijaan. Tällöin pääalgoritmille tulee 3-SAT-ajaksi $poly(n) \cdot 1,481^n$.

Olkoot F lause, C sen klausuuli ja a totuusjakelu. C luokitellaan riippuen siitä, kuinka moni sen literaaleista on tosia a:ssa. C:tä kutsutaan

$$(\underbrace{1,\ldots,1}_{p},\underbrace{0,\ldots,0}_{m}) - klausuuli$$

jos C sisältää täsmälleen p literaalia, jotka ovat tosia a:ssa, ja täsmälleen m literaalia, jotka ovat epätosia a:ssa. Esimerkiksi $(1,0,0)$-klausuuli sisältää kaksi tosi- ja yhden epätosi-literaalin.

Nyt F:ää kutsutaan *d-epätodeksi* a:ssa ($d \geq 0$) joss F:llä ei ole mallia Hamming-etäisyydellä d totuusjakelusta a. Kun d on vakio, voidaan polynomisessa ajassa tarkistaa, onko F d-epätosi a:ssa.

Seuraava lemma seuraa siitä, että F:n (1)-klausuulista $\overline{l_i}$ tulee tyhjä klausuuli $F_{|l_i=1}$:ssä.

Lemma 6 *Olkoot F lause, a ei sen malli. Jos F sisältää $(0,0,0)$-klausuulin $l_1 \vee l_2 \vee l_3$ ja (1)-klausuulin $\overline{l_i}$, jossa $i \in \{1,2,3\}$, tällöin saamme yhtäläisyyden: F toteutuu Hamming-etäisyydellä $\leq r$ a:sta joss on olemassa $j \in \{1,2,3\}$ siten, että $j \neq i$ ja $F_{|l_j=1}$ toteutuu Hamming-etäisyydella $\leq r-1$ a:sta.*

Seuraavat määritelmät kuvaavat lauseiden tyyppejä, joita proseduurin *Search* uusi versio käyttää.

1. Olkoot F 0-epätosi a:ssa. F on *I-rajoitettu* joss jompikumpi seuraavista pitää paikkansa:

   - F:ssä on (0)- tai $(0,0)$-klausuuli a:ssa.
   - F:ssä on $(0,0,0)$-klausuuli, jossa on literaali l siten, että $\overline{l}$ on F:n (1)-klausuuli

   Ideana tällaisissa rajoittuneissa muodoissa on, että tarvitsee haarautua korkeintaan kahteen eri haaraan, ei kolmeen.

2. Olkoot F 1-epätosi a:ssa. F on *II-rajoitettu* joss jompikumpi seuraavista pitää paikkansa:

   - F on I-rajoitettu a:ssa.
   - F:ssä on $(0,0,0)$-klausuuli, jossa on literaali l siten, että lause $F_{|l=1}$ on I-rajoitettu.

   Tällaisissa rajoittuneissa muodoissa, joissa jälkimmäinen ehdoista pätee, tarvitsee haarautua yhtä alemmalla tasolla kerran vähemmän, eli tuossa haarassa, joka on I-rajoitettu haaraudutaan kerran vähemmän. Jos ensimmäinen ehdoista pätee, tarvitsee, samoin kuin I-rajoitetuissa, haarautua vain kahteen jo tässä vaiheessa.

3. Olkoot F 2-epätosi a:ssa. F on *III-rajoitettu* joss jompikumpi seuraavista pitää paikkansa:

   - F on I- tai II-rajoitettu a:ssa.
   - F:ssä on $(0,0,0)$-klausuuli siten, että klausuulin *jokaisen* literaalin l suhteen lause $F_{|l=1}$ on II-rajoitettu.

   Näidenkin lapsien haarautumista hakupuussa rajoittaa tieto, että kaikki kolme ovat II-rajoitettuja eli niillä on vähemmän kuin 3^r haarautumista.

On huomattava, että lauseen rajoittuneisuus tässä mielessä riippuu valitusta totuusjakelusta eli käytännössä siitä, missä päin hakuavaruutta tämä paikallinen haku etsii. Esimerkiksi $\{\{A_1,\overline{A_2}\},\{A_1,A_2,A_3\}\}$ totuusjakelulla 000 on 0-epätosi eikä ole I-rajoitettu. Sen sijaan sijoituksella 010 se on edelleen 0-epätosi mutta tässä I-rajoitettu johtuen ensimmäisestä eli $(0,0)$-klausuulista.

Seuraava lemma ja sen seurauslause osoittavat, että rekursiivisia kutsuja ei tarvitse tehdä muille kuin III-rajoitetuille lauseille, koska muut ovat toteutuvia.

Määritellään, että lause F on triviaali joss F sisältää tyhjän klausuulin (välttämättä epätosi) tai F on tyhjä (eli tosi). Lause F on epätriviaali täsmälleen silloin kun se ei ole triviaali.

Lemma 7 *Olkoot F epätriviaali ja 3-epätosi a:ssa. Olkoot l epätosi literaali a:ssa. Olkoot $F_{|l=1}$ edelleen epätriviaali (oletuksen mukaan se on 2-epätosi). Tällöin jos $F_{|l=1}$ on III-rajoitettu a:n suhteen, silloin $F_{|l=1}$ on II-rajoitettu tai F itse on III-rajoitettu.*

Todistus. Ideana on saattaa $F_{|l=1,...}$ sellaiseen muotoon, että se havaitaan I- tai II-rajoitetuksi. Kun oletetaan, että tutkittava lause ei ole I- eikä II-rajoitettu, lähdetään liikkeelle lauseesta löytyvästä $(0,0,0)$-klausuulista. Lausehan nimittäin on epätosi annetulla totuusjakelulla ja jollei se ole I- tai II-rajoitettu, siinä ei ole (0)- tai $(0,0)$-klausuuleita eli siinä on oltava tuollaisia $(0,0,0)$-klausuuleita vähintään yksi.

Kun yksinkertaiset tapaukset on saatu tutkittua, jää jäljelle tapaus, jossa F sisältää kaksi eri $(0,0,0)$-klausuulia ja voidaan jättää supistus $F_{|l=1}$ välistä ja tutkia muotoa $F_{|\dots}$, jolloin F osoittautuu III-rajoitetuksi.

(Seurauslause 2.) Olkoot F epätriviaali ja 2-epätosi a:ssa. Jos F ei ole III-rajoitettu a:ssa, niin F on toteutuva.

Todistus. Jos F ei ole 3-epätosi, niin F on toteutuva. Koska F ei ole III-rajoitettu, se ei myöskään ole I-rajoitettu a:ssa. Tämän vuoksi tarkastellaan $(0,0,0)$-klausuulia $l_1 \vee l_2 \vee l_3$. Jos kaikilla $i \in \{1,2,3\}$ $F_{|l_i=1}$ olisi III-rajoitettu, niin edellisen lemman mukaan jokainen $F_{|l_i=1}$ on II-rajoitettu tai F on III-rajoitettu. Ensimmäisestä vaihtoehdosta kuitenkin seuraa, että F on III-rajoitettu.

Nyt siis on jokin $i \in \{1,2,3\}$ siten, että $F_{|l_i=1}$ on epätriviaali, 2-epätosi ja ei ole III-rajoitettu. Lisäksi $F_{|l_i=1}$:ssä on aidosti vähemmän klausuuleita, joten induktiolla lopulta tullaan sellaiseen lauseeseen, joka ei enää ole 3-epätosi ja siten toteutuva.

Nyt saadaan siis uudeksi paikalliseksi hauksi seuraava algoritmi.

$Search(F,a,r)$

1. Jos F ei ole 2-epätosi a:ssa, palauta *tosi*.
2. Jos $r \leq 0$, palauta *epätosi*.
3. Jos F sisältää tyhjiä klausuuleita, palauta *epätosi*.
4. Jos F ei ole III-rajoitettu a:n suhteen, palauta *tosi*.
5. Jos F on I-rajoitettu a:n suhteen, haaraudu sen epätoden klausuulin suhteen, joka osoittaa F:n I-rajoitetuksi.
6. Jos F on II-rajoitettu a:n suhteen, haaraudu sen epätoden klausuulin suhteen, joka osoittaa F:n II-rajoitetuksi.
7. Haaraudu sen klausuulin suhteen, joka osoittaa F:n III-rajoitetuksi.

Nyt tässä jätetään siis I-rajoitetuista tapauksista suorittamatta $Search(F_{|l_i=1,a,r-1}$ sille l_i, jolle F:stä löytyy (1)-klausuuli $\overline{l_i}$, kun verrataan aikaisempaan *Search*-algoritmiin.

Tällaisen haun haarautumisista muodostetaan hakupuu ottamalla solmuiksi funktion kutsut, lehdiksi sellaiset kutsut, jotka eivät enää haaraudu tutkimaan lisää joko havaitessaan lauseen todeksi tai säteen r ollessa nolla ja kaariksi rekursiot.

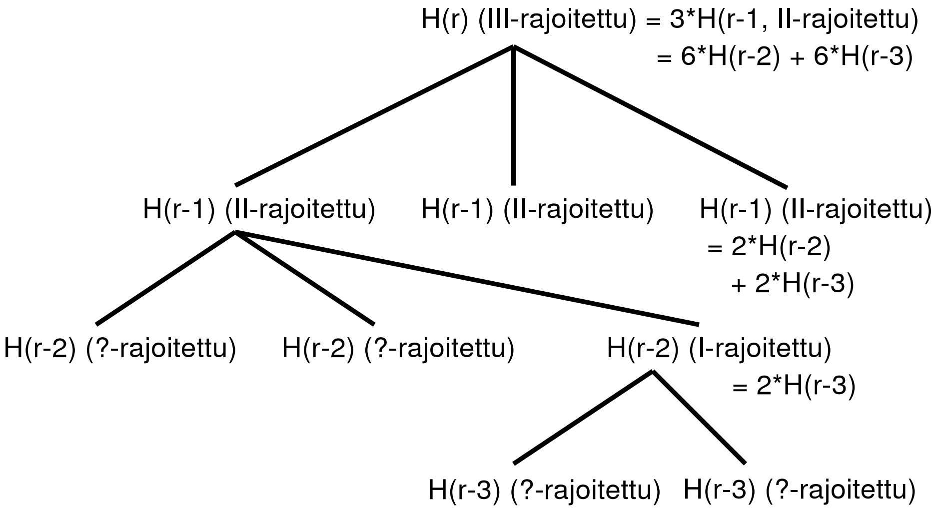
Hakupuun lehtien (Kuva 1) estimointiin käytetään funktiota $H(r)$, joka määritellään seuraavasti rekursiivisesti: $H(0) = 1$, $H(1) = 3$, $H(2) = 9$ ja $r \geq 3$

$$H(r) = 6 \cdot (H(r-2) + H(r-3)).$$

Hakupuun lehtien lukumäärä $L(F,a,r) \leq H(r)$ kaikille r.

Funktiolla H on rajat $2 \cdot H(r-1) \leq H(r)$ $(r \geq 1)$ ja $2 \cdot (H(r-1) + H(r-2)) \leq H(r)$ $(r \geq 2)$ induktiolla.

Nyt yksinkertaisissa (F triviaali tai toteutuva) tapauksissa (algoritmin alkupuoli) ollaan hakupuussa lehtisolmussa, joten ehto pätee. Muutoin F on joko I-, II- tai III-rajoitettu. I-rajoitetussa käytetään ensimmäistä epäyhtälöistä, II-rajoitetussa jälkimmäistä epäyhtälöä ja jos F on III-rajoitettu, niin lehtien määrä on suoraan H:n määritelmästä. Näin induktiolla todistuu tämänkin lemma.



Kuva 1: Eri tavoin rajoitetut lauseet ja *Search*-funktion haarautuminen niissä.

Eksplisiittiset rajat saadaan akateemisella arvauksella $H(r) = 9 \cdot \alpha^r$, jossa α ratkeaa yhtälöstä $\alpha^3 = 6 \cdot \alpha + 6$. Laskemalla saadaan $\alpha = \sqrt[3]{4} + \sqrt[3]{2}$, joka on $2,847$ ja $2,848$ välillä. Arvaus voidaan todistaa oikeaksi induktiolla.

Nyt lasketaan yhtälölle (8) uusi $\rho = 0,26$, joka minimoi eksponenttiosan tulosta. Näin saadaan 3-SAT-algoritmi, jonka suoritusaika on

$$poly(n) \cdot (2,848^{0,26} \cdot 2^{1-h(0,26)})^n \leq poly(n) \cdot 1,481^n$$

Tämä 3-CNF-optimointi voidaan soveltaa myös k-CNF-algoritmiksi. Tällöin ideana on I-rajoitteeseen muuttaa ehtoja vastaamaan k:ta ja aina saadaan haarautumisia yksi vähemmän kuin aluksi esitellyssä paikallisessa haussa. Lemma 7 todistus oli jo 3-CNF-tapauksessa laajoja yksityiskohtia sisältävä ja monia erikoistapauksia huomioonottava, joten kasvattamalla k:ta, tulee näitä käsiteltäviä vaihtoehtoisia tapauksia paljon. Tällä optimoinnilla saadaan aina k^r laskettua k_+^r ajassa, jossa $k-1 < k_+ < k$ paikallisen haun osalta.

3 Yhteenveto ja pohdinnat

Tämän algoritmin laskennallisesti vaativin osa onkin kaikki mahdolliset totuusjakelut peittävän koodin generointi ja siihen menee suhteessa eniten aikaa. Tämä vie myös *eksponentiaalisesti kasvavan määrän muistia*. Toisaalta muistetaan esitetty variaatio, jossa siis oli ε lisänä, vie vähemmän muistia ja soveltuu siksi käytännössä toteutettavaksi.

Toteutuvuustarkistimen käyttötarkoitusta ajatellen pitäisi toteutuvan lauseen tapauksessa varmasti palauttaa se (vasta)esimerkki, jolla lause osoitetaan toteutuvaksi. Paperissa esitetyissä algoritmeissa tämä on (yksinkertaisuuden vuoksi) jätetty pois, mutta paikallisen haun algoritmia hiukan muokkaamalla saataisiin tuo joskus varmasti tarpeellinen totuusjakelu palautettua helposti sen löydyttyä.

Paikalliseen hakuun perustuvia algoritmeja kehitetään, koska tällöin laskenta saadaan helposti hajautettua. Globaaleilla algoritmeilla hajauttaminen ei ole kovin helppoa, jos ylipäätään mahdollista. Toisaalta tässä esitetty algoritmi on samalla myös täydellinen ja deterministinen, joten ratkaisu ongelmaan saadaan varmasti tietyssä ajassa, joka on laskentaa suunniteltaessa hyödyllinen ominaisuus. Paikalliseen hakuun perustuvilla satunnaisilla algoritmi voidaan toki saada millä tahansa halutulla varmuudella (< 100 %), mutta tällöin yleensä suoritusaika alkaa paisumaan, koska hakua täytyy uudelleenkäynnistää monen monta kertaa. Deterministisellä algoritmilla tätä ongelmaa ei ole. Koska esitelty algoritmi on lisäksi täydellinen, soveltuu se myös verifiointiin, joissa täytyy saada varma vastaus.

Paperissa esitetyn analyysin konkreettista käyttöä ajatellen heikentää se, että ainoastaan huonoimman tapauksen suoritusaikaa on analysoitu. Käytännössä algoritmeilla on keskimääräinen suoritusaika jossain määrin pienempi kuin huonoin tapaus. Siksi tämän algoritmin käyttöönotto ei ole kovin mielenkiintoista ennen tarkempaa analyysiä todellisesta käyttäytymisestä. Lisäksi tähän esitettyyn algoritmiin voidaan yhdistää monia käytännössä hyödyllisiä sievennyskeinoja lauseille, joilla saadaan lauseen rakenteisuutta käyttämällä suoritusaikaa alaspäin. Tällaisenaan algoritmi ei analysoi lainkaan lauseiden rakennetta, vaan toimii yksinkertaisen suoraviivaisesti.

Viitteet

[1] Evgeny Dantsin, Andreas Goerdt, Edward A. Hirsch, Ravi Kannan, Jon Kleinberg, Christos Papadimitriou, Prabhakar Raghavan, Uwe Schöning, A deterministic $(2 - 2/(k+1))^n$ algorithm for k-SAT based on local search, Theoretical Computer Science 289/1 (2002) 69-83, http://logic.pdmi.ras.ru/%7ehirsch/papers.html.

[2] R.B. Ash, Information Theory, Dove, New York, 1965.

[3] D.S. Hochbaum (Ed.), Approximation Algorithms for NP-hard Problems, PWS Publishing Company, MA, 1997.