

Davis-Putnam-menetelmän toteuttaminen

Käsiteltäviä asioita:

1. Toteutusten keskeisiä piirteitä
2. Esimerkkitoteutuksia: ntab, relsat, satz, chaff ja minsat.
3. Vaikeiden k -SAT-ongelmien tuottaminen
4. Esimerkkitoteutusten vertailua

Perusalgoritmi

Syöte: klausuulijoukko S ja literaalijoukko M (malliehdot).

function DP(S, M): literaalijoukko tai arvo 'UNSAT';

begin

$\langle S', M' \rangle := \text{simplify}(S, M)$;

if $S' = \emptyset$ **then** return M'

else if $\perp \in S'$ **then** return 'UNSAT'

else begin

$L := \text{choose}(S', M')$;

$M'' := \text{DP}(S' \cup \{L\}, M' \cup \{L\})$;

if $M'' = \text{'UNSAT'}$ **then** return DP($S' \cup \{\bar{L}\}, M' \cup \{\bar{L}\}$)

else return M''

end

end

1. Toteutusten keskeisiä piirteitä

- Yksinkertaisuutensa vuoksi Davis-Putnam-menetelmä on toteutettavissa varsin suoraviivaisesti.
- Ylärajana hakuvaruuden koolle on 2^v (v on atomilauseiden lkm.).
- Käyttämällä syyvyshakua algoritmi voidaan toteuttaa polynomisessa tilassa klausuulijoukon kokoon nähden.
- Jokaisessa hakuvaruuden pisteessä:
 - (a) yksinkertaistetaan klausuulijoukkoa ja
 - (b) valitaan uusi atomilause haarautumista varten.
- Hakuheuristiikka vaikuttaa suorituskyykyyn olennaisesti.
- Pahimmassa tapauksessa algoritmin suoritus vaatii eksponentiaalisen ajan klausuulijoukon kokoon nähden.

- Funktio $\text{simplify}(S, M)$ pyrkii
 1. yksinkertaistamaan klausuulijoukkoa S ja
 2. täydentämään malliehtojen joukkoa M .
- Funktio $\text{choose}(S, M)$ valitsee
 1. atomilauseen A haarautumissääntöä varten ja
 2. ensimmäisenä käsiteltävän malliehdon (A tai $\neg A$).
- Suorituskyky riippuu olennaisesti heuristiikasta.
- Kaupankäynti: yksinkertaistaminen vs. haarautuminen.
 - Yksinkertaistamisella voidaan vähentää haarautumistarvetta ja siten pienentää käsiteltävää hakuvaruutta.
 - Lineaarinen yksinkertaistaminen (esim. yhden literaalin sääntö) kannattaa tehdä jokaisessa hakuvaruuden pisteessä.

Esimerkki. Olkoon $S = \{A, \neg A \vee B, \neg B \vee C, \neg C\}$.

- Kyseinen klausuulijoukko voidaan osoittaa totutumattomaksi yhden literaalin säännöllä (malliehdot: A , B ja C).

$$S \rightsquigarrow \{B, \neg B \vee C, \neg C\} \rightsquigarrow \{C, \neg C\} \rightsquigarrow \{\perp\}$$

- Toteutumattomuus voidaan osoittaa myös haarautumissäännöllä:

$\{A, \neg A \vee B, \neg B \vee C, \neg C\}$	
(A)	(¬A)
$\{B, \neg B \vee C, \neg C\}$	$\{\perp, \neg B \vee C, \neg C\}$
(B)	(¬B)
$\{C, \neg C\}$	$\{\perp, \neg C\}$
(C)	(¬C)
$\{\perp\}$	$\{\perp\}$

2. Esimerkkitoteutuksia

- ntab: J.M. Crawford ja L.D. Auton [AIJ, 1996]: *Experimental Results on the Crossover Point in Random 3SAT*
- relsat: R.J. Bayardo ja R.C. Schrag [AAAI, 1997]: *Using CSP Look-Back Techniques to Solve Real-World SAT Instances*
- satz: C.M. Li [IPL, 1999]: *A Constraint-based Approach to Narrow Search Trees for Satisfiability*
- chaff: M.M. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang ja S. Malik [DAC, 2001]: *Chaff: Engineering an Efficient SAT Solver*
- minisat: N. Eén ja N. Sörensson [SAT, 2005]: *MiniSat – A SAT Solver with Conflict-Clause Minimization* (poster)

Tyypillinen toteutus

1. Tehdään esikäsittely, jossa yksinkertaistetaan klausuulijoukkoa ja vähennetään muuttujien määrää (polynominen aikavaativuus) esim. käyttämällä yhden literaalin, komplementti puuttuu, literaali epäonnistuu ja poista 1-literaalit sääntöä.
2. Jokaisessa hakupisteessä yksinkertaistetaan klausuulijoukkoa (yhden literaalin sääntö ja joskus komplementti puuttuu -sääntö).
3. Kussakin hakupisteessä lasketaan heuristiikka (lasketaan literaalien esiintymistiheyksiä klausuuleissa ja/tai mitä johtopäätöksiä saataisiin yhden literaalin säännöllä).
4. Hakua voidaan edelleen tehostaa
 - käyttämällä oikeasevaa palautuvaa hakua (backjumping) ja
 - lisäämällä klausuuleja (rajoitettu oppiminen).

Esimerkki: ntab

Funktion $\text{simplify}(S, M)$ toteutus:

- Yksinkertaistetaan klausuulijoukkoa S ja laajennetaan malliehtojen joukkoa M käyttämällä yhden literaalin sääntöä (unit propagation).
- Yksinkertaistaminen voidaan toteuttaa lineaarisessa ajassa (Dowling-Gallier algoritmi).
- Samalla päivitetään heuristiikan laskennassa tarvittavia laskureita.

Esimerkki. Olkoon $S = \{A, \neg A \vee \neg B, C \vee B, \neg C \vee B \vee \neg D \vee A\}$ ja $M = \emptyset$.

Yhden literaalin säännöllä saadaan malliehdot $M' = \{A, \neg B, C\}$ ja yksinkertaistettu klausuulijoukko $S' = \emptyset$.

Heuristiikka

Käytössä Freemanin [väitöskirja, s. 51] heuristinen funktio

$$H(A) = w(\neg A) * w(A) * 2^{10} + w(\neg A) + w(A) \quad (1)$$

missä $w(L)$ on literaalin L esiintymien lukumäärä *avoimissa klausuuleissa* C (joiden toteutuminen ei vielä ole taattu).

- Tavoitteena on suosia atomisia lauseita A , joille $w(A) \neq 0$, $w(\neg A) \neq 0$ ja nämä luvut ovat suunnilleen saman suuruiset.
- Kahden viimeisen summatermin tarkoituksena on antaa järjestys tapauksille, joissa $w(A) = 0$ tai $w(\neg A) = 0$.
- Eksponentti (10) on valittu sanapituuden ja painojen maksimiarvojen perusteella sopivaksi.

Huomio. Paino $w(L)$ voidaan valita eri tavoin eri toteutuksissa!

Esimerkki: relsat

- Funktio $\text{simplify}(S, M)$ toteuttaa yhden literaalin säännön.
- Valintafunktio $\text{choose}(S, M)$ eroaa hieman ntabin vastaavasta:
 - $H(A)$:n lausekkeesta (1) puuttuu kerroin $2^{10} = 1024$,
 - toiselle kierrokselle valitaan atomit A , joilla $H(A)$ poikkeaa korkeitaan 20% hyvyyslukujen maksimista,
 - näistä valitaan satunnaisesti jatkoon max. 10 atomia, ja
 - uuden hyvyysluvun $H'(A)$ laskennassa painona $w(L)$ on yhden literaalin säännöllä saatavien uusien literaalien lukumäärä olettaessa L todeksi.
- Tämän lisäksi *relsat* sisältää toteutuksen oikaisevasta palautuvasta hausta (backjumping).

Funktion $\text{choose}(S, M)$ toteutus ntabissa:

1. Lasketaan jokaiselle *avoimelle* atomilauseelle A hyvyysluku $H(A)$ yhtälöllä (1), kun painona $w(L)$ on literaalin L esiintymien lukumäärä *avoimissa* binääriklausuuleissa.
2. Valitaan k parasta atomilauseetta ($k = v - 21 * v'$, missä v' on niiden atomilauseiden lukumäärä, joilla on jo totuusarvo).
3. Lasketaan näille atomeille A uusi hyvyysluku $H'(A)$ yhtälöllä (1), kun literaalille L painona $w(L)$ on yhden literaalin säännöllä saatavien uusien (avointen) binääriklausuulien lukumäärä olettaessa L .
4. Valitaan atomilause, jolla on paras hyvyysluku $H'(A)$.

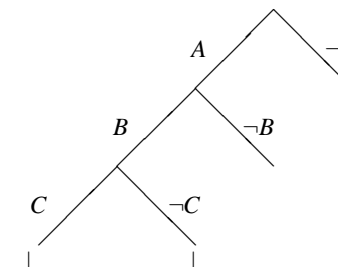
Muuta huomion arvoista:

- *Oikaiseva* palautuva haku on toteutettu versiossa *ntab_back*.

Esimerkki. Oikaiseva palautuva haku (backjumping):

Tarkastellaan klausuulijoukkoa $S =$

$$\{ \neg A \vee D \vee E, \quad \neg A \vee \neg D \vee \neg E, \quad \neg C \vee \neg D, \quad \neg C \vee \neg E, \\ C \vee D, \quad C \vee E, \quad B \vee F, \quad B \vee G, \\ B \vee H, \quad \neg B \vee F, \quad \neg B \vee G, \quad \neg B \vee H \}$$



Oletuksilla $\{A, B, C\}$ ja $\{A, B, \neg C\}$ saadaan johdetuksi tyhjä klausuuli \perp kuuden ensimmäisen klausuulin avulla. Tyhjän klausuulin \perp johdot eivät siten riipu B :n arvosta.

Vaihtamalla A :n arvo vältetään oletuksiin $\{A, \neg B\}$ liittyvä turha laskenta.

Esimerkki: satz

- Funktio $\text{simplify}(S, M)$ toteuttaa yhden literaalien säännön.
- Heuristiikan $\text{choose}(S, M)$ toteutus:
 1. Jokaiselle avoimelle atomilauseelle A lasketaan hyvyysluku $H(A)$ yhtälöllä (1), kun literaalien L painona $w(L)$ on yksinkertaistettavissa olevien klausuulien lukumäärä olettaessa L todeksi.
 2. Valitaan atomi A , jolla on paras hyvyysluku $H(A)$.

Huomio. Heuristiikka ei ota kantaa literaalien A ja $\neg A$ käsittelyjärjestykseen: literaali A oletetaan aina ensin todeksi.

Esimerkki: chaff

Funktion $\text{simplify}(S, M)$ toteutus chaffissa:

- Perustuu yhden literaalien säännön nerokkaaseen toteutukseen.
- Jokaisesta klausuulista $L_1 \vee \dots \vee L_n$ (missä $n > 1$) nimetään aluksi kaksi *vahtiliteraalia* V_1 ja V_2 ($V_1 \neq V_2$).
Jos V_1 (V_2) merkitään epätodeksi malliehdolla $\overline{V_1}$ ($\overline{V_2}$),
 1. korvataan V_1 (V_2) literaalilla $V \in \{L_1, \dots, L_n\} - \{V_1, V_2\}$, joka ei ole epätosi senhetkisten malliehtojen nojalla, **tai**
 2. mikäli tämä ei ole mahdollista, päätellään malliehto V_2 (V_1) **tai** todetaan ristiriita (jos molemmat vahtiliteraalit ovat epätodet).
- Mahdollistaa klausuulijoukon yksinkertaistamisen *linearisessa* ajassa ja tehostaa palautuvaa hakua ja tähytystä, koska malliehdon L peruuttaminen voidaan toteuttaa *vakioajassa*.

Parannuksia satzin perusversioon:

- Tuotetaan resoluutiolla klausuulijoukkoon korkeintaan kolmen literaalien mittaisia klausuuleja (lisärajoitteita).
Esimerkki. Klausuuleista $A \vee B \vee C$ ja $\neg A \vee B \vee \neg D$ saadaan resoluutiolla $B \vee C \vee \neg D$.
- Literaalien L paino $w(L)$ on lausekkeiden $f(\overline{L_1}) + f(\overline{L_2})$ summa yli binääriklausuulien $L_1 \vee L_2$, jotka saadaan olettamalla L .
Yllä $f(L)$ on literaalien L binääriklausuuliesiintymien lukumäärä (mikäli näitä on) tai summa painoista $5^{-(r-2)}$ jokaiselle L :n esiintymälle r :n mittaisessa klausuulissa.
- Lisänä 2-tasoinen *tähytys* (look-ahead): jos $\text{simplify}(S, M \cup \{L\})$ yksinkertaistaa riittävän montaa (parametri) klausuulia sekä $\text{simplify}(S, M \cup \{L, L_1\})$ ja $\text{simplify}(S, M \cup \{L, \overline{L_1}\})$ tuottavat ristiriidan jollekin L_1 , päätellään \overline{L} .

Funktion $\text{choose}(S, M)$ toteutus chaffissa:

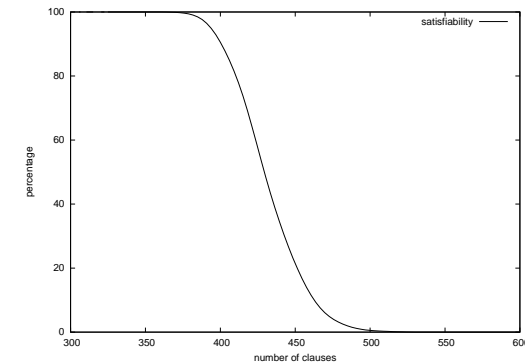
1. Jokaisella literaalilla L annetaan oma laskuri $c(L)$ (alussa 0).
 2. Kun käsitellään klausuulia $L_1 \vee \dots \vee L_n$ kasvatetaan laskurien $c(L_1), \dots, c(L_n)$ arvoja yhdellä.
 3. Tietyin aikavälein laskurien arvot jaetaan jollain vakiolla.
 4. Funktio $\text{choose}(S, M)$ palauttaa satunnaisesti jonkin literaaleista L , joilla laskurien $c(L)$ arvo on suurin.
- **Eritysispiire I:** Havaittuja konflikteja kirjataan ylös klausuuleina.
 - Muistinkäyttöä hallitaan poistamalla näin opittuja klausuuleja myöhemmin laskennan aikana.
 - **Eritysispiire II:** Laskenta saatetaan keskeyttää ja aloittaa alusta.
 - Mahdolliset opitut klausuulit kuitenkin säilytetään.
 - Menettelyn käyttö on rajoitettu täydellisyysden takaamiseksi.

Esimerkki: minisat

- Toteutus perustuu kahden vahtiliteraalin käyttöön (kuten chaff).
- **Erityispiirteitä:**
 1. Käytössä VSDIS (Variable State Independent Decaying Sum) muuttujajärjestys, jossa arvoja pienennetään 5% jokaisen konfliktin jälkeen.
 2. Binääriklausuulien erityistoteutus.
 3. Opittuja klausuuleja poistetaan aktiivisesti.
 4. Konflikteista opittuja klausuuleja pyritään minimoimaan.
- Hyvän suorituskyvyn taustalla esikäsittely SatELiteGTI:
N. Eén, A. Biere [SAT, 2005]: *Effective Preprocessing in SAT through Variable and Clause Elimination*

Vaikkeimmat ongelmat faasitransitioalueella

- Esimerkiksi 3-SAT-ongelman vaikeimmat instanssit saadaan tuottamalla klausuulijoukkoja, joilla suhde $\frac{c}{v}$ on noin 4,3.



- Faasitransitiopisteessä ongelmien instanssi muuttuvat äkillisesti toteutuvista toteutumattomiksi.

3. Vaikeiden k -SAT-ongelmien tuottaminen

Yksi mahdollisuus on tuottaa satunnaisia klausuulijoukkoja:

- Parametrit: atomisten lauseiden määrä v ,
literaalien määrä k jokaisessa klausuulissa
klausuulien määrä c .
- Proseduuri: toistetaan c kertaa seuraava:
 1. Valitaan satunnaisesti k kpl atomisista lauseista A_1, \dots, A_v .
 2. Muodostetaan näistä k :n literaalien klausuuli siten, että jokainen valituista muuttuja A_i kirjataan
 - positiiviseksi literaaliksi A_i todennäköisyydellä 0.5 ja
 - muutoin negatiiviseksi literaaliksi $\neg A_i$.

4. Esimerkkitoteutusten vertailua

- relsat on nopeampi kuin posit [Freeman] ja ntab.
 - Logistiikkaan ja suunnitteluun liittyviä esimerkkejä: relsat (4s – 813s) vs. ntab_back (17s – yli 12h).
 - Joillain esimerkeillä jopa parempi kuin paikalliset hakumenetelmät, muilla taas selvästi huonompi.
- satz on joskus nopeampi kuin ntab ja relsat.
 - 400 muuttujan satunnaiset 3SAT-ongelmat faasitransitio-alueelta: satz (1825s) vs. posit (4872s) vs. ntab (7362s).
- chaff on vielä kertaluokkaa (tai kahta) nopeampi ja kilpailukykyinen jopa paikallisten hakumenetelmien kanssa.

Keskitetty vertailu (L. Simon, 2003)

- Käytössä 1303 testiongelmää, joilla testattu 23:a toteutusta.

Toteutus	Ajankäyttö	Suht. "hitaus"	Onnist. testit	Testien lkm.
zchaff	2d 22h 52m 29s	1.00	1280	1303
reلسat-200	5d 19h 20m 50s	1.97	1260	1303
reلسat	7d 09h 33m 20s	2.51	1243	1303
sato	8d 09h 35m 12s	2.84	1241	1303
satz-215	8d 22h 48m 56s	3.03	1237	1303
eqsatz	9d 00h 06m 08s	3.05	1237	1303
satz-213	9d 15h 18m 16s	3.26	1232	1303
sato-3.2.1	10d 09h 52m 52s	3.53	1221	1303
satz	10d 12h 59m 23s	3.57	1211	1303

- Lähde: <http://www.lri.fr/~simon/satex/satex.php3>

Linkkejä

Edellä käsiteltyjen toteutusten lähdekoodit ovat saatavilla verkossa:

- ntab: <http://www.cirl.uoregon.edu/crawford/>
- reلسat: <http://www.almaden.ibm.com/cs/people/bayardo/>
- satz: <http://www.laria.u-picardie.fr/~cli/>
- chaff: <http://www.ee.princeton.edu/~chaff/>
- minisat: <http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/>

Muuta mielenkiintoista:

- Kattava kokoelma linkkejä: <http://www.satlive.org/>
- Kilpailu: <http://www.satcompetition.org/>

SAT 2005 Competition

- Tarkoituksena on löytää haastavia ongelmia toteutusten vertailemiseksi ja kannustaa uusien toteutusten kehitystyötä tarjoamalla mahdollisuus kattavaan vertailuun.
- Esimerkkinä kilpailun industrial-sarjan tulokset:

	Gold	Silver	Bronze
SAT UNSAT	SatELiteGTI	MiniSAT 1.13	Zchaff_rand and HaifaSAT
SAT	SatELiteGTI	MiniSAT 1.13	Jerusat 1.31 and HaifaSAT
UNSAT	SatELiteGTI	Zchaff_rand	HaifaSAT

- Seuraava kilpailu järjestetään vuonna 2007!

Oppimistavoitteet

- Ymmärrät DP-menetelmän algoritmisen esitystavan
- Tunnet heuristiikan merkityksen hakualgoritmin tehokkuudelle
- Tiedät mistä tehokkaimmat toteutukset ovat saatavissa
- Osaat ratkoa loogisia tehtäviä näiden työkalujen avulla
- Kokeilet itsenäisesti muutaman työkalun käyttöä