Pseudo-buleanska ograničenja i ograničenja kardinalnosti i svodjenje na SAT

Marko Nikitović

Septembar 2025

Sažetak

Rad predstavlja mali prevodilac pseudo-buleanska ograničenja, odnosno ograničenja kardinalnosti u specijalnom slučaju, u CNF formu koja se dalje može proslediti SAT rešavačima. Program parsira ograničenja, svodi ih na kanonski oblik, pa u zavisnosti od tipa ograničenja primenjuje kodiranje. Za ograničenja kardinalnosti primenjena je tehnika sekvencijalnog brojanja (eng. Sequential Counter Encoding), dok za opšti slučaj koristi kodiranje pomoću mreže sabirača (eng. Adder Network Encoding).

Sadržaj

1 Uvod

Jedna od primena SAT rešavača je rešavanje problema kombinatorne pretrage, što problem zadovoljavanja pseudo-buleanskih ograničenja i ograničenja kardinalnosti jeste. Odnosno, cilj ovoga rada je prevesti proizvoljnu instancu ovih problema u problem zadovoljivosti iskaznih formula, čije dalje rešavanje prepuštamo SAT rešavačima.

Pseudo-buleanska ograničenja predstavljaju prirodnu generalizaciju klasičnih klauza u logici, jer umesto proste disjunkcije literala uključuju i koeficijente, odnosno težine, čime omogućavaju izražavanje bogatijih uslova. Poseban slučaj

ovih ograničenja su ograničenja kardinalnosti, gde se traži da najmanje određeni broj literala bude istinit. Njihovim daljim uopštavanjem dolazimo do linearnih celobrojnih ograničenja. Mnogi problemi mogu se izraziti kao optimizacija vrednosti linearne pseudo-buleanske funkcije pod linearnim pseudo-buleanskim ograničenjima.

U nastavku rada, u drugom poglavlju predstavljene su osnovne definicije i notacija. Treće poglavlje daje detaljniji opis algoritama korišćenih za kodiranje, dok četvrto poglavlje sadrži implementacione detalje programa. Konačno, u petom poglavlju dat je zaključak sa osvrtom na rezultate i mogućnosti daljeg proširenja rada.

2 Osnove

Neka je $X = \{x_1, x_2, \ldots, x_n\}$ skup atoma. Svakoj vrednosti x_i možemo dodeliti vrednost 0 (što odgovara logičkoj vrednosti netačno) odnosno 1 (što odgovara tačno). Literal je ili atom x_i ili njegova negacija $\bar{x_i}$, pri čemu važi $x_i + \bar{x_i} = 1$. Literale označavamo sa l_i.

Definicija. Pseudo-buleansko ograničenje je linearno celobrojno ograničenje nad literalima oblika:

$$\sum_{i=1}^{n} a_i \ell_i \bowtie A,$$

Gde važi:

- ℓ_i je literal
- koeficijenti $a_i \in \mathbb{Z}$,
- konstanta $A \in \mathbb{Z}$,
- $\bowtie \in \{\leq, \geq, =, <, >\}$,

Poseban slučaj psuedo-buleanskih ograničenja su **ograničenja kardinalnosti**, kod kojih su svi koeficijenti jednaki 1 ($a_i = 1$). Proizvod $a_i * l_i$ ćemo nazivati termom.

U daljem tesktu ćemo koristiti i pojmove klauza i konjuktivna normalna forma. Klauza je disjunkcija literala, tj. logički izraz u kojem su literali povezani operatorom „ili“ ($\vee$). Na primer, $(x_1 \vee \neg x_2 \vee x_3)$ je klauza.

Konjunktivna normalna forma (eng. CNF) je konjunkcija klauza, tj. formula oblika:

$$C_1 \wedge C_2 \wedge \cdots \wedge C_m,$$

gde je svaka C_i klauza. Na primer, $(x_1 \vee \neg x_2) \wedge (x_3 \vee x_4)$ je formula u CNF-u.

U ovom radu ćemo reći da je pseudo-buleansko ograničenje **normalizovano** ukoliko je zapisano u obliku u kome je relacija isključivo $\geq$, a svi koeficijenti uz literale nenegativni brojevi. Na taj način se dobija jedinstveni kanonski oblik, pogodan za dalje kodiranje u CNF.

3 Opis metode

Da bismo na izlazu dobili problem u CNF-u, svako ograničenje prolazi kroz niz koraka. Detalji svakog koraka biće opisani u daljem tekstu.

Ulaz je tekstualni oblik pseudo-buleanskih ograničenja. Parser razdvaja levu i desnu stranu izraza, prepoznaje operatore ($\leq, \geq, >, <, =$) i formira listu termova sa pripadajućim koeficijentima.

3.1 Svodjenje na normalnu formu

Nad svakom klazom vršimo niz transformacija, pri čemu su dobijena ograničenja ekvivalentna. Ovim pristupom sve formule svodimo na istu kanonsku (normalnu) formu, koja će nam dalje olakšati prevodjenje i smanjiti broj različitih slučajeva. Koristimo sledeća pravila:

1 Zamenjujemo stroge nejednakosti:

$$\sum_{i=1}^{n} a_i \ell_i > A \implies \sum_{i=1}^{n} a_i \ell_i \geq A + 1,$$

$$\sum_{i=1}^{n} a_i \ell_i < A \implies \sum_{i=1}^{n} a_i \ell_i \leq A - 1,$$

2 Množimo sa -1 da dobijemo veće ili jednako:

$$\sum_{i=1}^{n} a_i \ell_i \leq A \implies \sum_{i=1}^{n} -a_i \ell_i \geq -A,$$

3 Sve koeficijente prebacujemo u nenegativne:

$$l_i = 1 - \bar{l_i}$$

$$-a_i * l_i = -a_i * (1 - \bar{l_i}) = -a_i + a_i * l_i$$

4 Zamenjujemo sve jednakosti:

$$\sum_{i=1}^{n} a_i \ell_i = A \implies \sum_{i=1}^{n} a_i \ell_i \geq A \quad \wedge \sum_{i=1}^{n} a_i \ell_i \leq A,$$

3.2 Prevodjenje u CNF

U zavisnosti od tipa ograničenja, koristimo različita kodiranja: ograničenja kardinalnosti prevodimo pomoću sekvencijalnih brojača, dok se opšta pseudo-buleanska ograničenja prevode pomoću mreže sabirača. Pored već pomenutih postoje i mnoge druge metode, koje proizvode različitu strukturu i broj klauza, čineći ih manje ili više efikasnim.

Jedno direktno prevodjenje $x1 + x2 + \cdots + xn \leq k$:

$$\bigwedge_{\substack{M \subseteq \{1,\ldots,n\} \\ |M| = k+1}} \bigvee_{i \in M} \neg x_i \,,$$

Ovakvo prevodjenje u najgorem slučaju proizvodi eksponencijalan broj klauza. Zbog toga se u literaturi razvijaju efikasnija kodiranja zasnovana na ideji da se izgradi hardverski sklop za brojanje i poređenje, a zatim da se takav sklop prevede u CNF. Pored samih promenljivih ograničenja x1,. . . ,xn, u takvim kodiranjima se uvode i dodatne pomoćne promenljive s1,. . . ,sm, koje omogućavaju kompaktnije izražavanje uslova. Cilj svake metode kodiranja jeste da se postigne što manji broj klauza i što efikasnija CNF reprezentacija, jer to direktno utiče na performanse SAT rešavača.

3.2.1 Prevodjenje pomoću sekvencijalnih brojača

Sekvencijalni brojač je tehnika kodiranja koja uvodi pomoćne promenljive radi praćenja parcijalnih suma, tj. koliko je literala do određenog trenutka postavljeno na 1. Svaka pomoćna promenljiva označava da među prvih i literala postoji bar j tačnih, dok posebne promenljive vi signaliziraju da je zadati prag k već premašen. Ovaj mehanizam se zatim prevodi u CNF preko sistema klauza koje povezuju ulazne promenljive sa pomoćnim, čime se obezbeđuje da parcijalne sume ispravno prate broj tačnih literala.

Bazni slučaj ($j > 1$)**:**	**Induktivni korak** ($i \geq 2$, $j \geq 1$)**:**
$x_1 \Leftrightarrow s_{1,1}$	$x_i \Rightarrow s_{i,1}$
$\overline{s}_{1,j}$	$s_{i-1,j} \Rightarrow s_{i,j}$
	$(x_i \wedge s_{i-1,j}) \Rightarrow s_{i,j+1}$
	$s_{i,j+1} \Rightarrow x_i \vee s_{i-1,j+1}$
	$s_{i,j+1} \Rightarrow s_{i-1,j} \vee s_{i-1,j+1}$

Na kraju je potrebno dodati još jednu jediničnu klauzu $s_{n,k}$, čime se eksplicitno zahteva da ograničenje bude zadovoljeno. Takođe, primetimo da je u slučaju kada je $k \leq 0$ ograničenje trivijalno zadovoljeno, dok je u slučaju $k > n$ trivijalno nezadovoljeno. Složenost metode je $O(n * k)$ klauza uz $(n-1) * k$ pomoćnih promenljivih, a glavna prednost je što je kodiranje jednostavno i efikasno u praksi, jer omogućava snažnu propagaciju čim se literala postavi na tačno, sve ostale promenljive moraju biti netačne.

3.2.2 Prevodjenje preko mreže sabirača

Najpre ćemo prikazati kako se ograničenja kardinalnosti mogu prevesti u CNF koristeći mreže sabirača. Kasnije ćemo isti pristup proširiti i na opšta pseudo-buleanska ograničenja, koja se prevode upravo redukcijom na njih.

Ideja je da se leva strana ograničenja (npr. $y_1 + y_2 + \cdots + y_n$) predstavi binarno, pomoću pomoćnih promenljivih $z_0, z_1, \ldots, z_k$, tako da važi:

$$y_1 + y_2 + \cdots + y_n = z_0 + 2z_1 + \cdots + 2^k z_k.$$

Da bi se ovo ostvarilo, koristi se kombinacija polu-sabirača (eng. half-adder, HA) i punih sabirača (eng. full-adder, FA), pri čemu se oni spajaju u mrežu koja računa zbir. Za n=2 koristi se HA, za n=3 FA, a za veći broj promenljivih koristi se rekurzivna konstrukcija gde se problem deli na dve polovine i rezultati kombinuju kroz sabirače.

Neka je f kolo koje računa zbir, odnosno $(z_0, \ldots, z_k) = f(y_1, \ldots, y_n)$. Tada je za $n > 3$:

$$\begin{aligned}
(w_1, w_2, \ldots, w_k) &= f(y_1, y_2, \ldots, y_{n/2}) \\
(w_{k+1}, w_{k+2}, \ldots, w_{2k}) &= f(y_{n/2+1}, y_{n/2+2}, \ldots, y_n) \\
(z_0, c_2) &= \mathrm{HA}(w_1, w_{k+1}) \\
(z_1, c_3) &= \mathrm{FA}(w_2, w_{k+2}, c_2) \\
(z_2, c_4) &= \mathrm{FA}(w_3, w_{k+3}, c_3) \\
&\vdots \quad\quad \vdots \\
(z_{k-2}, c_k) &= \mathrm{FA}(w_{k-1}, w_{2k-1}, c_{k-1}) \\
(z_{k-1}, z_k) &= \mathrm{FA}(w_k, w_{2k}, c_k)
\end{aligned}$$

U ovom radu koristili smo sekvencijalno sabiranje umesto podele na dva dela, radi jednostavnosti implementacije. Broj sabiranja u ovom pristupu isti je kao i u balansiranoj *adder* mreži, ali razlika se ogleda u dubini konstrukcije: dok je kod sekvencijalnog sabiranja dubina $O(n)$, kod balansirane mreže ona je $O(\log n)$. Ovo ne menja ukupan broj klauza, ali može uticati na efikasnost propagacije u SAT rešavaču. Složenost ovakvog kodiranja je O(n) klauza i promenljivih.

Za ulazno ograničenje:

$$a_1 y_1 + a_2 y_2 + \cdots + a_n y_n = z_0 + 2z_1 + 2^2 z_2 + \cdots + 2^m z_m,$$

gde je

$$m = \lceil \log(a_1 + a_2 + \cdots + a_n) \rceil,$$

svaki koeficijent a_i se najpre zapisuje u binarnom obliku:

$$a_i = A_i^0 + 2A_i^1 + 4A_i^2 + \cdots + 2^m A_i^m, \quad A_i^j \in \{0, 1\}.$$

Tada se zbir može prepisati kao:

$$\begin{aligned}
a_1 y_1 + a_2 y_2 + \cdots + a_n y_n = {} & (A_1^0 y_1 + A_2^0 y_2 + \cdots + A_n^0 y_n) \\
& + 2(A_1^1 y_1 + A_2^1 y_2 + \cdots + A_n^1 y_n) \\
& + \cdots + 2^m (A_1^m y_1 + A_2^m y_2 + \cdots + A_n^m y_n).
\end{aligned}$$

Najpre prvu grupu radimo kao problem ograničenja:

$$z_0^0 + 2z_0^1 + \cdots + 2^{m_0} z_0^{m_0} = A_1^0 y_1 + A_2^0 y_2 + \cdots + A_n^0 y_n.$$

Zatim kada vratimo u gornju sumu dobijamo:

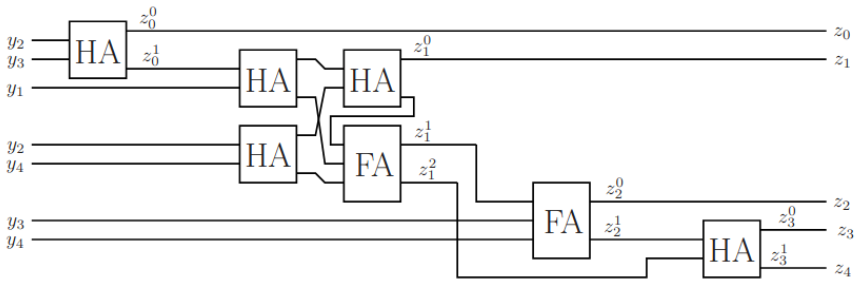
$$= z_0 + 2\big(A_1^1 y_1 + \cdots + A_n^1 y_n + z_0^1\big) + 2^2\big(A_1^2 y_1 + \cdots + A_n^2 y_n + z_0^2\big) + \cdots$$

Slično dalje i za ostale grupu, na kraju ove procedure dobijamo:

$$(A_1^0 y_1 + A_2^0 y_2 + \cdots + A_n^0 y_n) + \cdots + 2^m (A_1^m y_1 + \cdots + A_n^m y_n) = z_0 + 2z_1 + \cdots + 2^m z_m.$$

Kodiranje ovim pristupom daje $O(log(a_0) * n)$ klauza i promenljivih.

U našoj implementaciji, baziranoj na ovom pristupu, redukcija se izvodi tako što se svaka kolona obrađuje zasebno kao ograničenje kardinalnosti: kada kolona sadrži tri ili više literala, koristi se potpuni sabirač koji daje zbirni bit u istoj koloni i prenos u sledećoj; kada ostanu tačno dva literala, primenjuje se polusabirač sa istim principom. Na taj način svaka kolona se svodi na najviše jedan



Slika 1: Sabiračko kolo za ograničenje: $2 * y_1 + 3 * y_2 + 5 * y_3 + 6 * y_4 \leq 9$

bit, dok se višak prenosi u naredne kolone, čime se postupno gradi binarna reprezentacija ukupnog zbira.

Na kraju dobijeni zbir pomoću komparatora poredimo sa konstantom koja se nalazi sa desne strane. Uvodimo klauzu koja zahteva da je poslednja pozajmica jednaka 0, time zahtevajući da naš zbir zadovoljava ograničenje.

3.3 Generisanje DIMACS formata

Kada završimo prevodjenje svih ograničenja u CNF formu, dobijene klauze zapisujemo u izlazni fajl u DIMACS formatu. Dalje, izlaz se može proslediti nekom SAT rešavaču (npr. MiniSat) i dobiti za koje vrednosti atoma je formula zadovoljiva, ukoliko ona to jeste.

4 Implementacija

Sa standardnog ulaza program liniju po liniju parsira pseudo-buleanska ograničenja (svaka linija predstavlja jedno ograničenje), normalizuje ih i prevodi u CNF formu primenom sekvencijalnog brojača ili mreže sabirača, u zavisnosti od tipa ograničenja. Dobijene klauze se zatim ispisuju u fajl *output.cnf* u DIMACS formatu, spremne za SAT rešavač.

Program je implementiran u C++-u i organizovan kroz nekoliko modula koji jasno razdvajaju logičke celine: parsiranje i normalizaciju ograničenja, bazne strukture za rad sa CNF formulama, implementacije različitih tehnika kodiranja, i ulazno/izlaznu kontrolu. Ovakva podela doprinosi čitljivosti i lakšem održavanju koda, kao i mogućnosti da se lako proširuje novim metodama kodiranja. Takodje, celokupan kod se nalazi u prostoru imena ***satenc***.

- `core.hpp / core.cpp`

  Nalaze se osnovne klase koje omogućavaju rad sa ograničenjima i CNF formulama. `Term` predstavlja literal sa imenom promenljive, oznakom negacije i težinom, dok `Constraint` opisuje celo pseudo-buleansko ograničenje

kao skup termina, relaciju i desnu stranu. `CNF` upravlja konstrukcijom klauza: dodaje nove promenljive, skladišti klauze i omogućava izvoz u DIMACS formatu, a `SymTab` obezbeđuje mapiranje između simboličkih imena promenljivih i internih identifikatora. Ove klase čine temelj programa jer povezuju ulazna ograničenja sa njihovim konačnim CNF zapisom.

- `normalizer.hpp / normalizer.cpp`

  Sadrži klasu Normalizer koja se bavi parsiranjem i normalizacijom pseudo-buleanskih ograničenja:

  `parse_line` - iz tekstualnog reda izvlači koeficijente, promenljive i komparator (<=, =, >=) i vraća strukturu Constraint.

  `normalize_inplace` – negativne koeficijente prebacuje u pozitivne tako što invertuje literal i uvećava desnu stranu ograničenja.

  `to_geq` – prevodi svako ograničenje u oblik sa relacijom $\geq$. Ako je bilo $\leq$, zamenjuje literale negacijama i koriguje konstantu; ako je $=$, razlaže ga na dva $\geq$ ograničenja.

- `adder_network.cpp`

  Implementira klasu `AdderNetwork` koja kodira opšta pseudo-buleanska ograničenja preko binarnog sabiranja i poredjenja: `build_sum_bits` razlaže težine u bit-kolone i kolonu-po-kolonu ih redukuje pomoću sabirača (tri ulaza $\rightarrow$ `full_adder`, dva ulaza $\rightarrow$ `half_adder`) dok svaka kolona ne svede na najviše jedan bit, čime dobija vektor suma S (binarna reprezentacija zbira).

  Funkcija `full_adder` dodaje CNF klauze za $s = x \oplus y \oplus z$ i $c = \text{maj}(x, y, z)$, a `half_adder` za $s = x \oplus y$ i $c = x \wedge y$.

  Zatim `ge_constant(S, k, cnf)` uvodi „borrow/pozajmica" signale b_j i CNF klauzama implementira komparator $S \geq k$, vraćajući literalu `ge` ekvivalentnu toj relaciji, koju glavna funkcija zatim vezuje jediničnom klauzom.

- `seq_counter.cpp`

  Implementira kodiranje kardinalnog ograničenja $\sum X_i \geq k$ pomoću sekvencijalnog brojača. Funkcija `geq` uvodi pomoćne promenljive $s_{i,j}$ koje označavaju da među prvih i literala ima bar j tačnih, dodaje klauze za bazu i induktivni korak, a na kraju jediničnu klauzu $s_{n,k}$ kojom se zahteva da je prag k dostignut.

- `main.cpp`

  Glavni program koristi `Normalizer` da parsira i normalizuje ulazna ograničenja, potom bira način kodiranja: kardinalna se prevode sekvencijalnim brojačem, a opšta mrežom sabirača. Rezultat se zapisuje u *DIMACS* formatu u fajl output.cnf, uz ispis mapiranja promenljivih i normalizovanih ograničenja.

4.1 Prevodjenje i pokretanje programa

Program je implementiran pomoću standarda *C++17* i koristi *CMake* kao sistem za izgradnju. Minimalna verzija *CMake*-a potrebna za prevođenje je 3.10, što je naglašeno u datoteci *CMakeLists.txt*.

Uobičajena praksa sa *CMake*-om je da se prevođenje obavlja u posebnom direktorijumu, i celokupno prevodjenje se može izvesti pomoću narednih komandi nakon pozicioniranja u koreni direktorijum projetka:

mkdir build
cd build
cmake ..

Program se pokreće iz komandne linije. On sa standardnog ulaza čita pseudo-buleanska ograničenja, svaka linija predstavlja jedno ograničenje. Primer pokretanja programa:

./sat_encoder
$x_1 + x_2 + x_3 >= 3$
$\neg x_1 - x_2 > 1$

Dobijene klauze se nalaze u *output.cnf* fajlu. Test primeri dati su u okviru direktorijuma *tests/* .

5 Zaključak

U ovom radu prikazali smo postupak prevođenja pseudo-buleanskih ograničenja u konjunktivnu normalnu formu (CNF) koja je pogodna za SAT rešavače. Implementacija je obuhvatila dva pristupa: kodiranje kardinalnih ograničenja pomoću sekvencijalnog brojača i kodiranje opštih pseudo-buleanskih ograničenja pomoću mreža sabirača. Rezultati pokazuju da ovakav prevodilac omogućava automatsko i ispravno transformisanje široke klase ograničenja u standardizovan format, čime se obezbeđuje njihova dalja upotreba u okviru postojećih SAT tehnologija. Iako implementacija nije optimizovana za performanse, već pre svega za jasnoću i razumevanje principa, ona može poslužiti kao osnova za buduća unapređenja koja bi uključivala efikasnija kodiranja i bolju podršku za propagaciju u rešavačima.

Literatura

[1] Eén, Niklas, and Niklas Sörensson. *Translating Pseudo-Boolean Constraints into SAT*. MiniSat+ documentation, 2006. Available at: `http://minisat.se/downloads/MiniSat+.pdf`.

[2] Sinz, Carsten. *Towards an Optimal CNF Encoding of Boolean Cardinality Constraints*. In: Principles and Practice of Constraint Programming (CP 2005). Springer, 2005. Available at: `https://www.carstensinz.de/papers/CP-2005.pdf`.

[3] Abío, Ignasi, Valentin Mayer-Eichberger, and Peter Stuckey. *Encoding Linear Constraints into SAT*. arXiv preprint arXiv:2005.02073, 2020. Available at: `https://arxiv.org/pdf/2005.02073`.

[4] Nordström, Jakob. *Pseudo-Boolean Solving and Optimization*. Tutorial presented at "Satisfiability: Theory, Practice, and Beyond" Boot Camp, Simons Institute for the Theory of Computing, February 4, 2021. Available at: `https://simons.berkeley.edu/sites/default/files/docs/16969/pseudobooleansolvingtutorial.pdf`