

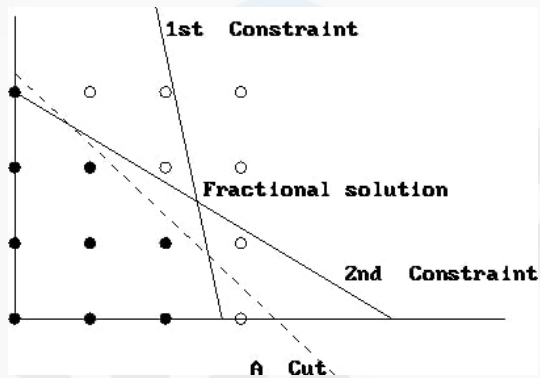
Metoda sjekućih ravni. Gomorijev rez.

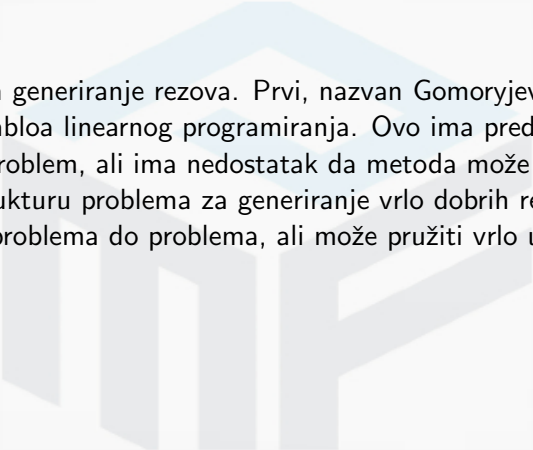
Ema Djedović

Odsjek za matematičke i kompjuterske nauke
Prirodno-matematički fakultet
Univerzitet u Sarajevu

06/2024

Algoritam sjekućih ravni rješava cijelobrojne programe modificirajući rješenja linearnih programa dok se ne dobije cjelobrojno rješenje. Ne dijeli dopušteno područje na podpodručja, kao u pristupima grananja i ograničavanja, već radi s jednim linearnim programom koji se rafinira dodavanjem novih ograničenja. Nova ograničenja sukcesivno smanjuju dopušteno područje dok se ne pronađe optimalno cijelobrojno rješenje. U praksi, postupci grananja i ograničavanja gotovo uvijek nadmašuju algoritam reznih ravnina. Ipak, algoritam je bio važan za evoluciju cijelobrojnog programiranja. Povijesno gledano, to je bio prvi algoritam razvijen za cijelobrojno programiranje za koji se moglo dokazati da konvergira u konačnom broju koraka. Osim toga, iako se općenito smatra vrlo neučinkovitim, algoritam je dao uvid u cijelobrojno programiranje koji je doveo do drugih, učinkovitijih algoritama.





Postoje dva načina za generiranje rezova. Prvi, nazvan Gomoryjevi rezovi, generira rezove iz bilo kojeg tabloa linearnog programiranja. Ovo ima prednost da može “rješavati” bilo koji problem, ali ima nedostatak da metoda može biti vrlo spora. Drugi pristup je koristiti strukturu problema za generiranje vrlo dobrih rezova. Ovaj pristup zahtijeva analizu od problema do problema, ali može pružiti vrlo učinkovite tehnike rješenja.

Osmotrimo sljedeći cjelobrojni program:

$$\text{Maksimizirati } Z = 7x_1 + 9x_2$$

$$\text{Uz ograničenja: } -x_1 + 3x_2 \leq 6$$

$$7x_1 + x_2 \leq 35$$

$$x_1, x_2 \geq 0 \text{ (cjelobrojni)}$$

$$\text{Maksimizirati } Z = 7x_1 + 9x_2$$

$$\text{Uz ograničenja: } -x_1 + 3x_2 + s_1 = 6$$

$$7x_1 + x_2 + s_2 = 35$$

$$x_1, x_2, s_1, s_2 \geq 0 \text{ (cjelobrojni)}$$

Prije nego riješimo pridruženu linearnu relaksaciju, bit će nam od koristi da izrazimo s_1 i s_2 preko varijabli x_1 i x_2 :

$$s_1 = 6 + x_1 - 3x_2$$

$$s_2 = 35 - 7x_1 - x_2$$

Ako zanemarimo cjelobrojnost, dobit ćemo sljedeću optimalnu tablicu (s ažuriranim stupcima i smanjenim troškovima prikazanim za nebazične varijable):

Variable	x_1	x_2	s_1	s_2	$-z$	RHS
x_2	0	1	7/22	1/22	0	7/2
x_1	1	0	-1/22	3/22	0	9/2
$-z$	0	0	28/11	15/11	1	63

Posmatrajmo prvo ograničenje:

$$x_2 = \frac{7}{22}s_1 + \frac{1}{22}s_2 = \frac{7}{2}$$

Možemo to preurediti tako da sve cijele dijelove stavimo na lijevu stranu, a sve razlomke na desnu:

$$x_2 - 3 = \frac{1}{2} - \frac{7}{22}s_1 - \frac{1}{22}s_2$$

Sada, budući da lijeva strana sadrži samo cijele brojeve, desna strana mora zbrajati do cijelog broja. Koji cijeli broj može biti? Pa, ona se sastoji od nekog pozitivnog razlomka umanjenog za niz pozitivnih vrijednosti. Stoga desna strana može biti samo 0, -1, -2,...; ne može biti pozitivna vrijednost. Dakle, dobili smo sljedeće ograničenje:

$$\frac{1}{2} - \frac{7}{22}s_1 - \frac{1}{22}s_2 \leq 0$$

Uvrštavajući vrijednosti $s_1 = 6 + x_1 - 3x_2$ i $s_2 = 35 - 7x_1 - x_2$ dobivamo:

$$\frac{1}{2} - \frac{7}{22}(6 + x_1 - 3x_2) - \frac{1}{22}(35 - 7x_1 - x_2) \leq 0$$

čijim se sređivanjem dobije

$$-3 + x_2 \leq 0 \quad \text{odnosno} \quad x_2 \leq 3$$

Ograničenje koje smo dobili nazivamo *rez*, dodajemo ga u naš linearni program i isti ponovo rješavamo. Tako ponavljamo sve dok optimalne vrijednosti za varijable odluke ne postanu cijeli brojevi - tada algoritam staje.

Također možemo generirati rez iz drugog ograničenja.

Variable	x_1	x_2	s_1	s_2	$-z$	RHS
x_2	0	1	$7/22$	$1/22$	0	$7/2$
x_1	1	0	$-1/22$	$3/22$	0	$9/2$
$-z$	0	0	$28/11$	$15/11$	1	63

Ovdje moramo biti oprezni da dobijemo pravilne znakove.

$$x_1 - 1/22s_1 + 3/22s_2 = 9/2$$

$$x_1 + (-1 + 21/22)s_1 + 3/22s_2 = 4 + 1/2$$

$$x_1 - s_1 - 4 = 1/2 - 21/22s_1 - 3/22s_2$$

daje ograničenje:

$$\frac{1}{2} - \frac{21}{22}s_1 - \frac{3}{22}s_2 \leq 0$$

Ponovo izrazimo s_1 i s_2 preko varijabli odluke:

$$\frac{1}{2} - \frac{21}{22}(6 + x_1 - 3x_2) - \frac{3}{22}(35 - 7x_1 - x_2) \leq 0$$

čijim se sređivanjem dobije

$$x_2 \leq \frac{10}{3} \quad \text{odnosno} \quad x_2 \leq 3 \quad (\text{radi cjelobrojnosti})$$

Općenito, neka je $\lfloor a \rfloor$ najveći cijeli broj manji ili jednak a . Na primjer, $\lfloor 3.9 \rfloor = 3$, $\lfloor 5 \rfloor = 5$ i $\lfloor -1.3 \rfloor = -2$.

Ako imamo ograničenje $x_k + \sum a_i x_i = b$ gdje b nije cijeli broj, možemo pisati $a_i = \lfloor a_i \rfloor + a'_i$, za neko $0 \leq a'_i < 1$, i $b = \lfloor b \rfloor + b'$ za neko $0 < b' < 1$. Koristeći iste korake dobivamo:

$$x_k + \sum \lfloor a_i \rfloor x_i - \lfloor b \rfloor = b' - \sum a'_i x_i$$

da bismo tako dobili rez:

$$b' - \sum a'_i x_i \leq 0$$

Ovaj rez možemo dodati u linearni program i ponovo riješiti problem. Problem je zajamčeno da neće dobiti isto rješenje.

Ova metoda može garantirati pronalaženje optimalnog cjelobrojnog rješenja. Međutim, postoje neki nedostaci:

- ▶ Greška zaokruživanja može uzrokovati velike poteškoće: Je li to 3.000000001 stvarno 3, ili trebam generirati rez? Ako donesem pogrešnu odluku, mogao bih ili odbaciti izvedivo rješenje (ako je stvarno 3, ali generiram rez) ili bih mogao završiti s neizvedivim rješenjem (ako nije 3, ali ga tretiram kao takvog).
- ▶ Broj generiranih ograničenja može biti ogroman. Kao što grana i ograničenje može generirati veliki broj podproblema, ova tehnika može generirati veliki broj ograničenja.

Primjer uz Python biblioteku lippy

```
import lippy as lp
# lippy je Python biblioteka za rjesavanje problema
# linearnog programiranja

c_vektor = [3, 3, 7]
a_matrica =
[[1, 1, 1],
 [1, 4, 0],
 [0, 0.5, 3]]
b_vektor = [3, 5, 7]

gomory = lp.CuttingPlaneMethod(c_vektor, a_matrica, b_vektor)
print(" Rjesenje: -", gomory.solve())
```

- ▶ <https://pypi.org/project/lippy>
- ▶ <https://mat.tepper.cmu.edu>
- ▶ Marija Ivanović (2009) *Vežbe iz Operacionih istraživanja* Univerzitet u Beogradu, Matematički fakultet
- ▶ Bradley, S.P., Hax, A.C. and Magnanti, T.L. (1977) *Applied mathematical programming* Reading, Mass: Addison-Wesley.