SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 142

MOGUĆNOSTI KORIŠTENJA PARALELNIH ALGORITAMA U LINEARNOM OPTIMIRANJU

Mladen Karan

Zagreb, siječanj 2011.

Sadržaj

1. Uvod

U raznim vrstama organizacija uvijek se nastoji iskoristiti dostupne resurse (osoblje, strojevi, materijali …) na što je moguće bolji način da se ostvari neki cilj. Primjeri takvih ciljeva su što veća zarada, što manje vrijeme izrade nekog proizvoda ili što kvalitetnija usluga (npr. brzina prijenosa informacija u komunkacijskoj mreži).

Ponekad je moguće modelirati proces postizanja cilja kao minimizaciju ili maksimizaciju linearne funkcije. Često uz to ide i niz uvjeta koji također mogu biti izraženi u obliku linearnih nejednakosti (npr. količina sirovine potrebna za proizvodnju određenog broja proizvoda mora biti manja od ukupne dostupne količine te sirovine). Ako se stvarna situacija može dovoljno precizno modelirati na ovakav način moguće je primijeniti neki od postupaka za linearno programiranje.

Tehnologija izrade računalnog sklopovlja se iznimno brzo razvija. Zbog toga ona se polako približava fizikalnim ograničenjima koja onemogućuju izradu bržih sklopova. Sve više pažnje posvećuje se paralelizaciji kao izvoru daljnjeg ubrzanja (npr. višejezgreni procesori). Kao posljedica toga mogućnosti paralelizacije algoritama postaju sve važnija tema.

U ovom radu proučiti ćemo neke od postupaka za linearno programiranje sa posebnim naglaskom na mješovito cjelobrojno programiranje i algoritam grananja i ograđivanja. Razmotriti ćemo i mogućnosti paralelizacije tog algoritma. Konačno opisati ćemo program razvijen kao praktični dio rada.

2. Linearno programiranje

Linearno programiranje je matematički optimizacijski problem definiran na sljedeći način:

$$MAX\ Z = \boldsymbol{c}^T \boldsymbol{x}$$

$$uz\ uvjete:$$

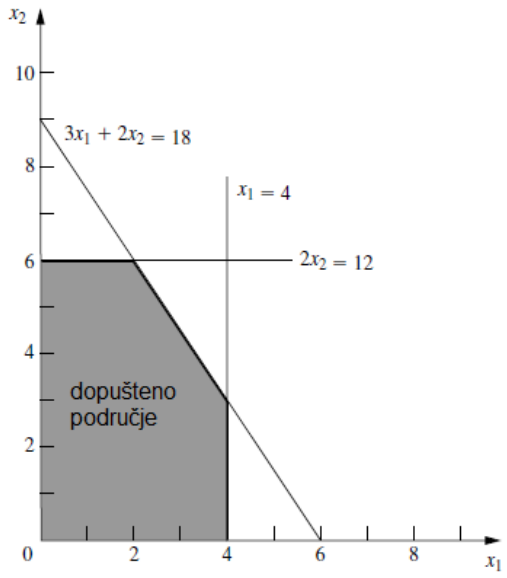
$$\boldsymbol{A} \cdot \boldsymbol{x} \leq \boldsymbol{b}$$

$$\boldsymbol{x} \geq \boldsymbol{0}$$

Ovaj oblik zove se standardni oblik linearnog programa. Svi ostali oblici (minimizacija, ograničenja tipa ≥ i =, varijable koje mogu biti negativne) mogu se svesti na ovaj standardni oblik. Varijable $(x_1, \dots, x_n)$ sadržane u vektoru **x** nazivamo strukturne varijable.

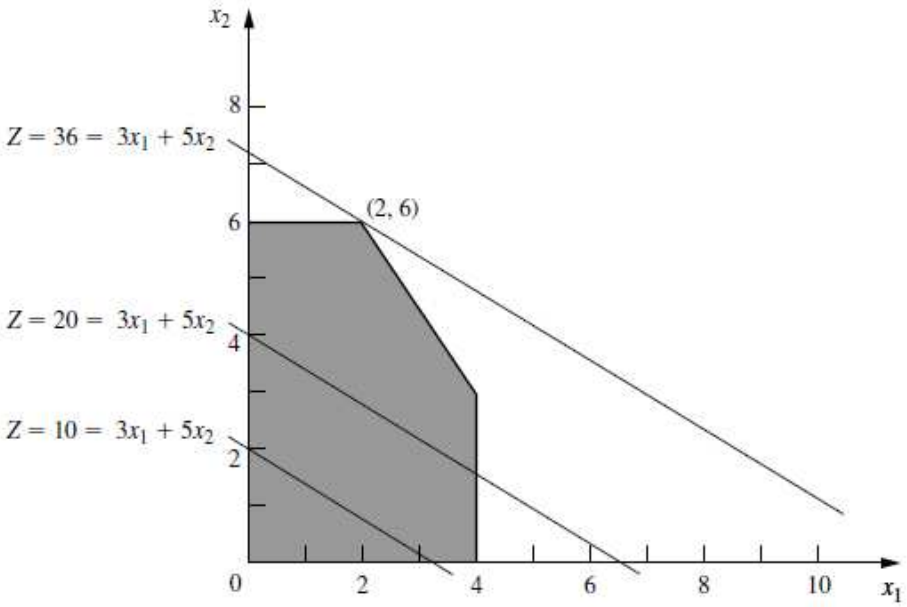
Ograničenja definiraju određeno područje unutar n-dimenzionalnog prostora u kojem se nalaze dopuštena rješenja. Primjer jednog takvog područja prikazan je na slici 2.1.

U slučaju linearnog programiranja dopušteno područje je uvijek konveksno (spojnica svaka dva moguća rješenja unutar dopuštenog područja također u potpunosti leži unutar dopuštenog područja).



Slika 2.1 primjer dopuštenog područja u 2D prostoru

U dvodimenzionalnom prostoru sve točke koje odgovaraju istoj vrijednosti funkcije cilja leže na istom pravcu. Na slici 2.2. je prikazano nekoliko takvih pravaca za vrijednosti funkcije cilja 10, 20 i 36.



Slika 2.2, primjer pravaca koji odgovaraju vrijednostima funkcije cilja 10, 20 i 36

Možemo vidjeti da se iznos funkcije cilja popravlja pomakom pravca u smjeru njezinog rasta. Optimizaciju bismo mogli provesti tako da pravac pomičemo u smjeru rasta funkcije cilja dokle god na njemu ima barem jedna točka koja je unutar dopuštenog područja. Tako bismo došli do pravca $36 = 3x_1 + 5x_2$ na slici.

Možemo primijetiti da se optimalno rješenje nalazi u jednom od vrhova konveksnog skupa mogućih rješenja. To će uvijek biti slučaj. Optimalno rješenje je uvijek jedan od vrhova dopuštenog područja ili dva vrha i cijela njihova spojnica [Kalpić, 1996].

3. Postupci za rješavanje LP problema

Ovdje su ukratko iznesene osnovne ideje metoda za rješavanje LP problema koje se koriste u programskoj realizaciji ovog rada. Mnogo opširniji opis može se naći u [Kalpić, 1996] ili [Hillier, 2001].

3.1 Simpleksna metoda

Kao veoma dobar postupak za rješavanje LP problema pokazala se simpleksna metoda. Razvijena je još 1947 godine a njen tvorac je G.B. Dantzig.

Simpleksna metoda oslanja se na činjenicu da je optimalno rješenje LP problema sigurno jedan od vrhova dopuštenog područja. Njena strategija je da iterativno posjećuje niz rješenja u vrhovima. Sljedeće susjedno rješenje koje se bira je uvijek takvo da ne kvari funkciju cilja. Postupak staje kada dođemo u rješenje u kojem više nema boljeg susjednog rješenja, takvo rješenje je optimalno.

Da bismo pokazali kako simpleksna metoda funkcionira pokazati ćemo ju na jednostavnom primjeru. Rješavati ćemo sljedeći LP problem.

$$MAX\ Z = 3x_1 + 5x_2$$

$$uz\ uvjete:$$

$$x_1 \leq 4$$

$$2x_2 \leq 12$$

$$3x_1 + 2x_2 \leq 18$$

$$x_1, x_2 \geq 0$$

Prvi korak je priprema problema za obradu simpleksnim postupkom. To ćemo učiniti tako da nejednakosti pretvorimo u jednakosti čime ćemo dobiti ekvivalentan problem:

$$MAX\ Z = 3x_1 + 5x_2$$

$$uz\ uvjete:$$

$$x_1 + x_3 = 4$$

$$2x_2 + x_4 = 12$$

$$3x_1 + 2x_2 + x_5 = 18$$

$$x_1, x_2, x_3, x_4, x_5 \geq 0$$

Kako bismo nejednakosti pretvorili u jednakosti dodali smo nove varijable x3, x4 i x5, za svako ograničenje po jednu. Te varijable nazivamo dopunskim varijablama.

Dopunske varijable možemo tumačiti kao mjeru koja nam opisuje koliko je resursa opisanog nekim ograničenjem još preostalo u nekom konkretnom rješenju. Ako je neka dopunska varijabla jednaka 0 to znači da je taj resurs iskorišten u potpunosti. Vrijednost veća od 0 znači da ukupna dozvoljena količina resursa nije potpuno iskorištena. Vrijednost manja od 0 pak znači da je iskorištena veća količina tog resursa nego što to ograničenje dozvoljava tj. da ograničenje nije zadovoljeno.

U originalnom problemu imali smo 2 varijable dok ih sada ima čak 5. Za prošireni skup varijabli $(x_1, \ldots, x_5)$ koje zadovoljavaju ovaj sustav jednadžbi (i ograničenje nenegativnosti) vrijedi da varijable x_1 i x_2 zadovoljavaju nejednadžbe ograničenja iz početnog problema. Zbog toga se rješenje početnog problema može lako očitati iz proširenog rješenja tako da se zanemare dopunske varijable.

Prošireno rješenje možemo dobiti rješavajući dobiveni sustav od 3 jednadžbe i 5 nepoznanica. U ovom primjeru sustav možemo jednoznačno riješiti tako da fiksiramo bilo koje dvije varijable.

U općenitom slučaju ćemo za problem s *n* varijabli i *m* ograničenja dobiti sustav jednadžbi s *n+m* nepoznanica i *m* jednadžbi. U simpleksnoj metodi ukupno *n* varijabli se fiksira na 0, te varijable zovemo nebazičnima. Ostale varijable su bazične.

Ovako dobiveno prošireno rješenje zove se bazično rješenje. Simpleksna metoda koristi samo ona bazična rješenja koja su moguća (sve varijable su veće od 0). Takva rješenja odgovaraju vrhovima dopuštenog područja [Hillier, 2001].

Sada kada smo pripremili problem, radi preglednosti, dobiven sustav jednadžbi zapisujemo u tablicu. Radi lakšeg zapisa definiciju funkcije Z shvatiti ćemo kao još jednu jednadžbu sa još jednom varijablom Z. Time u sustav dodajemo jednadžbu $Z - 3x_1 - 5x_2 = 0$. Konačan izgled tablice prije prve iteracije je u tablici 2.1.

Tablica 2.1. izgled simpleksne tablice prije prve iteracije

Bazična var.	**Z**	**X1**	**X2**	**X3**	**X4**	**X5**	**Desna str.**
Z	1	-3	-5	0	0	0	0
X3	0	1	0	1	0	0	4
X4	0	0	2	0	1	0	12
X5	0	3	2	0	0	1	18

U početku je pogodno za nebazične varijable (fiksirane u 0) odabrati strukturne varijable. Sada možemo izravno iz tablice očitati početno bazično rješenje (0,0,4,12,18), kao i pripadnu vrijednost funkcije cilja koja je 0.

Iteraciju algoritma izvodimo tako da prvo odaberemo ulaznu nebazičnu varijablu. U ovom primjeru odabrati ćemo x2 jer će njen porast najviše doprinijeti povećanju funkcije cilja (njen koeficijent u prvom redu je najviše negativan).

Potom odabiremo izlaznu bazičnu varijablu, to je ona varijabla koja će povećanjem ulazne nebazične varijable prva pasti u 0. Koja će to varijabla biti možemo odrediti tako da u retku svake od bazičnih varijabli izračunamo omjer koeficijenta desne strane i pozitivnog koeficijenta u stupcu ulazne nebazične varijable. Ona bazična varijabla koja ima najmanji omjer postaje izlazna bazična varijabla.

U našem primjeru omjer za varijablu x3 je 4/0, što ne moramo uzimati u obzir jer 0 nije pozitivan koeficijent. Koeficijent 0 zapravo znači da povećanjem x2 nećemo utjecati na x3. Omjer za varijablu x4 je 12/2 = 6 a za varijablu x5 18/2 = 9. Najmanji omjer je 6 i to za varijablu x4 pa nju biramo za izlaznu bazičnu varijablu.

Sada kada smo odabrali ulaznu nebazičnu i izlaznu bazičnu varijablu vršimo stožerni razvoj [Kalpić, 1996] oko koeficijenta u retku izlazne bazične i stupcu ulazne nebazične varijable. Time postižemo da u stupcu ulazne nebazične varijable svi koeficijenti postanu 0 osim onoga u retku izlazne bazične varijable koji postane 1. Ovo radimo kako bismo mogli izravno iz tablice očitati novo bazično rješenje (koje je susjedno prethodnom).

Konkretno u ovom slučaju ćemo redak izlazne bazične varijable (x4) pomnožiti sa 1/2, a zatim dodati retku funkcije cilja pomnoženog sa 5 i retku varijable x5 pomnoženog sa -2. Konačno varijabla x2 postaje bazična umjesto x4 koja je sada nebazična, iteracijom smo se pomaknuli u susjedno rješenje. Stanje tablice nakon prve iteracije je na prikazano u tablici 2.2. Novo bazično rješenje sada je (0,6,4,0,6) a vrijednost funkcije cilja 30.

Tablica 2.2. izgled simpleksne tablice nakon prve iteracije

Bazična var.	Z	X1	X2	X3	X4	X5	Desna str.
Z	1	-3	0	0	5/2	0	30
X3	0	1	0	1	0	0	4
X2	0	0	1	0	1/2	0	6
X5	0	3	0	0	-1	1	6

Identičnim postupkom možemo provesti i drugu iteraciju. Ulazna nebazična varijabla ispada x1 a izlazna bazična varijabla x5. Konačan izgled tablice nakon druge iteracije dan je u tablici 2.3.

Tablica 2.3. izgled simpleksne tablice nakon druge iteracije

Bazična var.	Z	X1	X2	X3	X4	X5	Desna str.
Z	1	0	0	0	3/2	1	36
X3	0	0	0	1	1/3	-1/3	2
X2	0	0	1	0	1/2	0	6
X1	0	1	0	0	-1/3	1/3	2

Vidimo da više nema pogodnih kandidata za ulaznu nebazičnu varijablu (koji bi povećavali funkciju cilja) jer u retku funkcije cilja nema negativnih koeficijenata. Time možemo zaključiti da smo pronašli optimalno rješenje i ono je (2,6,2,0,0) koje daje vrijednost funkcije cilja 36.

3.2 Dualna simpleksna metoda

Dualna simpleksna metoda provodi se na način identičan običnoj simpleksnoj metodi uz razliku načina odabira ulazne nebazične i izlazne bazične varijable.

Prvo se odabire izlazna bazična varijabla kao ona u čijem redku je koeficijent desne strane najviše negativan.

Potom se odabire ulazna nebazična varijabla kao ona koja ima najmanji omjer koeficijenta u funkciji cilja i negativnog koeficijenta u retku izlazne bazične varijable.

Zanimljivo je da je provođenje dualne simpleksne metode na primalnom problem zapravo ekvivalentno provođenju obične simpleks metode na dualnom problemu. Detaljan opis dualnosti može se naći u [Kalpić, 1996.]

Metoda je pogodna za upotrebu na tablicama koje definiraju rješenje koje zadovoljava uvjet optimalnosti ali je nemoguće jer takvo je rješenje moguće za dualni problem. Takve tablice se često javljaju kao posljedica dodavanja dodatnih ograničenja u postupku grananja i ograđivanja (detaljno opisan u poglavlju 4.1.3).

Također u postupku grananja i ograđivanja često se događa da dodavanjem ograničenja u tablicu stvorimo takav skup ograničenja koji nema moguće rješenje. Ako probamo primijeniti dualnu simpleksnu metodu na takvu tablicu nemoguće rješenje prepoznajemo tako što u redu izlazne bazične varijable nećemo naći negativnih koeficijenata oko kojih bi mogli obaviti stožerni razvoj.

3.3 Složenost simpleksne metode

Gornja granica na broj koraka simpleksnog postupka je ukupan broj bazičnih rješenja. Za problem sa *n* varijabli i *m* ograničenja ukupan broj bazičnih rješenja možemo dobiti kao broj načina na koji možemo među *n*+*m* varijabli odabrati njih *n* koje ćemo fiksirati u 0. Taj broj je jednak $\binom{n+m}{n}$ što je jednako $\frac{(n+m)!}{n!m!}$. Ovo ukazuje na eksponencijalnu složenost rješavanja problema [Kalpić, 1996]. Ipak simpleksna metoda ne ispituje sva bazična rješenja već samo ona moguća. Čak i među mogućim rješenjima ona na svom putu prema optimumu ne prolazi kroz sva već samo neka od njih. U praktičnoj primjeni pokazuje se da vrijeme rješavanja problema raste sa trećom potencijom broja ograničenja [Kalpić, 1996]

3.4 Ostali postupci za linearno programiranje

Od ostalih postupaka za rješavanje LP problema važna je revidirana simpleksna metoda. Do sada prikazana tablična simpleksna metoda nije pogodna za ugradnju na računalo jer u svakoj iteraciji iznova računamo sve koeficijente. U pravilu komercijalni programi za rješavanje LP problema koriste revidiranu simpleksnu metodu. U njoj se iskorištava činjenica da su praktični problemi većih dimenzija u pravilu predstavljeni rijetko punjenom matricom [Kalpić, 1996]. Ova metoda pruža znatnu uštedu u pogledu broja računskih operacija i potrebnog memorijskog prostora, kao i bolju kontrolu nad numeričkim pogreškama.

N. Karmarkar je 1984 izumio prvi praktično uporabiv postupak za rješavanje LP problema koji se pomiče prema optimumu kroz unutrašnjost dopuštenog područja. Za ovaj postupak može se dokazati da pronalazi optimalno rješenje u polinomijalnom broju koraka [Hillier, 2001]. Postupak je pokazao obećavajuće rezultate na iznimno velikim LP problemima (više desetaka tisuća ograničenja) [Kalpić, 1996].

4. Proširenja Linearnog Programiranja

4.1 Mješovito cjelobrojno programiranje

Među mnogim primjenama linearnog programiranja postoje i takve gdje neke od strukturnih varijabli označavaju cjelobrojne veličine. Njihova vrijednost nema smisla ako nije cjelobrojna. Primjeri takvih varijabli su broj radnika koje treba zaposliti ili broj proizvoda koje treba proizvesti.

4.1.1 Definicija

U [Kalpić, 1996] navodi se sljedeća definicija standardnog mješovitog cjelobrojnog programa (*engl. Mixed Integer Program*, MIP, MILP):

$$MAX\ Z = \sum_{j=1}^{n} c_j x_j + \sum_{k=1}^{p} c_{k+n}\delta_k$$

$$uz\ uvjete:$$

$$\sum_{j=1}^{n} a_{ij}x_j + \sum_{k=1}^{p} a_{ik+n}\delta_k \ \leq\ b_i \quad , i = 1, \dots, m$$

$$x_j \geq 0, j = 1, \dots, n$$

$$\delta_k\ cjelobrojna\ , k = 1, \dots, p$$

Radi se zapravo o formulaciji običnog LP problema sa ukupno *n*+*p* varijabli i *m* ograničenja. Pri tome prvih *n* varijabli ($x_1, \dots, x_n$) smiju poprimati realne vrijednosti. Na preostalih *p* varijabli ($\delta_1, \dots, \delta_p$) dodano je još ograničenje cjelobrojnosti.

Interesantno je da je linearno programiranje zapravo specijalan slučaj mješovitog-cjelobrojnog programiranja u kojem nemamo cjelobrojnih varijabli. Često se znaju pojaviti i sljedeća dva specijalna slučaja:

1. Cjelobrojno programiranje (*engl. Integer Programming*, IP) – slučaj mješovitog cjelobrojnog programiranja u kojem nema varijabli koje smiju imati realne vrijednosti. U ovoj vrsti problema sve strukturne varijable moraju biti cjelobrojne.

2. Binarno programiranje (*engl. Binary Programming*, BIP) – slučaj mješovitog cjelobrojnog programiranja u kojem osim što nema realnih varijabli, cjelobrojne strukturne varijable smiju poprimiti isključivo vrijednosti 0 ili 1. Jako često se javlja u praksi.

U nastavku poglavlja opisuju se neki od postupaka za rješavanje mješovitih-cjelobrojnih programa.

4.1.2 Zaokruživanje

U slučaju da se radi o relativno velikim brojevima, pogreške zaokruživanja su u praksi često sumjerljive s točnosti podataka. Tada je moguće zaokruživanjem postići cjelobrojne vrijednosti varijabli. Primjerice, ako ispadne da tvornica postiže maksimalnu dobit ako proizvede točno 114.71 pari cipela ako to zaokružimo na 115 neznatno ćemo pokvariti dobit. U ovakvim slučajevima koristimo linearno programiranje sa varijablama iz realnog područja. Tako dobiven rezultat potom zaokružujemo na cjelobrojne vrijednosti. Pri tome valja biti pažljiv jer moguće je da zaokruživanjem dobijemo rješenje koje nije moguće tj. krši neko od ograničenja.

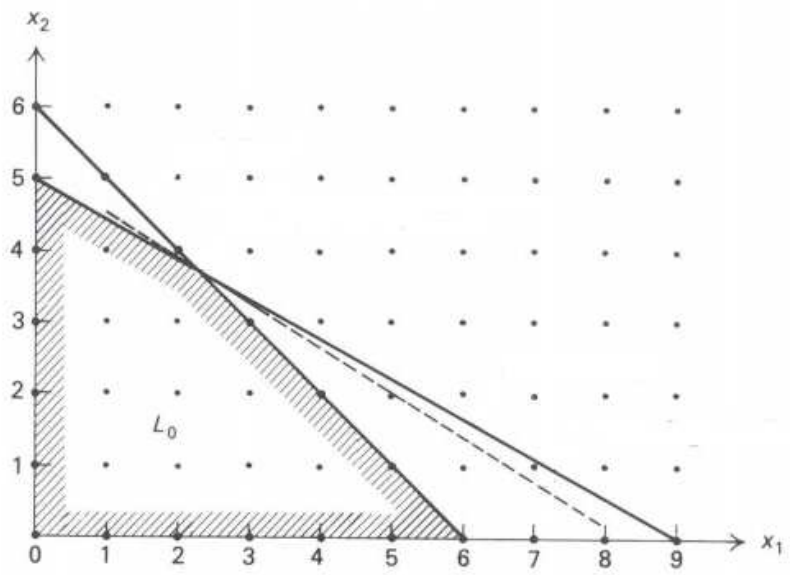
Ipak često zaokruživanje nije dovoljno da bismo dobili zadovoljavajući rezultat. Takvi slučajevi su oni u kojima bi zaokruživanje izazvalo iznimno grube pogreške. Jedan ekstreman slučaj je kada strukturne varijable zapravo predstavljaju varijable odluke 1 ili 0 – DA ili NE. Neki primjeri korištenja takvih varijabli su primjerice odluka o investiciji (gradnji tvornice/skladišta/trgovine na nekom mjestu), odluka o postavljanju određenog voda/cijevi/ceste, odluka isključivog izbora (poljoprivredna kultura koja će se zasaditi na određenoj parceli) [Kalpić, 1996]. Ovakve vrste problema često se javljaju u praksi. Na žalost poznavanje optimalnih realnih

vrijednosti varijabli odluke nije nam od koristi jer takvo rješenje nije primjenjivo. Da bismo riješili ovakve probleme potrebno je primijeniti neki od postupaka za mješovito cjelobrojno programiranje.

Vrijedi napomenuti da se može dogoditi da rješenje relaksiranog MIP problema (problema sa odbačenim ograničenjima cjelobrojnosti) slučajno ispadne cjelobrojno. Isplati se uvijek prije korištenja naprednijih metoda provjeriti je li se dogodio ovakav povoljan ishod. Također, postoje određeni problemi koji svojom specijalnom strukturom garantiraju cjelobrojno rješenje relaksiranog MIP problema. Jedan takav je problem protoka minimalne cijene (*engl. Minimum cost flow problem*). Također i neki njegovi specijalni slučajevi kao što su problem asignacije (*engl. Assignment problem*) i transportni problem (*engl. Transport problem*) [Hillier, 2001].

4.1.3 Postupak grananja i ograđivanja

U običnom linearnom programu rješenje smo tražili među bazičnim rješenjima koja su se nalazila na vrhovima konveksnog skupa mogućih rješenja. U cjelobrojnom programu rješenja se nalaze na cjelobrojnoj rešetki unutar konveksnog skupa mogućih rješenja relaksiranog problema (istovjetnog problema ali bez uvjeta cjelobrojnosti). Situacija je ilustrirana na slici.



Slika 4.1. prikaz odnosa rješenja cjelobrojnog programa i njegove relaksirane varijante

Vidi se da se u slučaju potrage za cjelobrojnim rješenjem moraju uzimati u obzir i točke na cjelobrojnoj rešetki unutar područja L_0 na slici. U općenitom slučaju točke cjelobrojne rešetke ne poklapaju se sa bazičnim rješenjima [Kalpić, 1996]. To značajno otežava rješavanje ovakve vrste problema.

Za rješavanje MIP problema koristi se varijanta algoritma grananja i ograđivanja (*engl. Branch & bound*). Ovo je samo jedna u nizu primjena ove ideje. Algoritam radi na principu podijeli i ovladaj.

Ideja je da se prvo riješi relaksirani problem i tako dobije realno rješenje. Potom se odabire neka varijabla koja ima realnu vrijednost i na njoj se obavlja grananje. Generiraju se dva nova linearna programa koji predstavljaju dva podproblema. U prvom podproblemu promatrana varijabla smije poprimiti vrijednosti manje ili jednake prvom cijelom broju koji je manji od njene realne vrijednosti. U drugom podproblemu promatrana varijabla smije poprimiti vrijednosti veće ili jednake prvom cijelom broju većem od njene realne vrijednosti. Tako dobiveni podproblemi se riješe (njihova relaksirana varijanta). Varijabla koju nastojimo učiniti cjelobrojnom će u rješenjima podproblema imati cjelobrojnu vrijednost. Razlog tome je što je upravo tada ona najbliže svojoj optimalnoj realnoj vrijednosti. Ovo nije slučaj jedino kada je rješenje podproblema nemoguće.

Na ovaj smo način iz početnog problema dobili dva podproblema. Početni problem možemo smatrati čvorom roditeljem u binarnom stablu. Njegovo dvoje djece su upravo generirani podproblemi. Čvorovi djeca tj. podproblemi mogu se ako je to potrebno (neka druga varijabla nije cjelobrojna) dalje granati (po nekoj drugoj varijabli). Tako se stablo grana dalje tj. raste.

Primjerice, recimo da varijabla x5 ima vrijednost 3.14 u relaksiranom problemu. Generirati ćemo dva čvora djecu. U čvorovima djeci imati ćemo isti problem kao i prije uz jedno dodatno ograničenje. U lijevom čvoru dodati ćemo ograničenje $x5 \leq 3$. U desnom čvoru dodati ćemo ograničenje $x5 \geq 4$. Ovim ograničenjima smo zabranili varijabli x5 da poprimi svoju optimalnu realnu vrijednost. U lijevom čvoru smo je povukli prema dolje dok smo je u desnom čvoru gurnuli prema gore.

Važno je naglasiti da su djeca generirana iz roditelja dodavanjem ograničenja na varijablu grananja. Zbog toga njihova vrijednost funkcije cilja ne može nikako biti bolja od one u čvoru roditelju. Zato vrijednost funkcije cilja u roditelju predstavlja ogradu na vrijednost funkcije cilja u svoj njegovoj djeci. U slučaju maksimizacije radi se o gornjoj ogradi dok je u slučaju minimizacije to donja ograda.

Algoritam radi tako da obrađuje čvorove dok svi nisu obrađeni. Ipak ukupna količina obrađenih čvorova dodatno se može smanjiti zaustavljanjem grananja. Grananje iz nekog čvora može se zaustaviti iz sljedećih razloga koje nazivamo testovima eliminacije čvora:

1. Rješenje se može odbaciti ograđivanjem
2. Rješenje je nemoguće
3. Rješenje je cjelobrojno

Prvi slučaj nastupa kada se na temelju ograde u nekom čvoru može zaključiti da niti jedno od njegove djece ne može imati povoljniju vrijednost funkcije cilja od trenutno najboljeg poznatog mogućeg (cjelobrojnog) rješenja. U slučaju maksimizacije grananje iz čvora možemo zaustaviti ako je gornja ograda manja od trenutno najveće poznate vrijednosti funkcije cilja. U slučaju minimizacije donja ograda mora biti veća od dosada najmanje poznate vrijednosti funkcije cilja.

Upravo ograđivanje daje najveću snagu algoritmu grananja i ograđivanja jer se njime može prekinuti grananje za koje smo sigurni da nas ne vodi do rješenja boljeg od onog koje već imamo. Ovo drastično smanjuje ukupan broj čvorova koje je potrebno obraditi.

Drugi slučaj je jednostavan, ako je rješenje u nekom čvoru nemoguće nema smisla dalje iz njega granati postupak.

Treći slučaj nastupa kada je rješenje relaksiranog problema u čvoru cjelobrojno. Time smo pronašli i cjelobrojno (dakle moguće) rješenje originalnog problema. Takvo rješenje se obično uspoređuje sa do sada najboljim viđenim cjelobrojnim rješenjem i po potrebi se osvježava vrijednost najboljeg.

U nastavku dan je pseudokod algoritma grananja i ograđivanja za MIP probleme [Hillier, 2001] (pretpostavlja se da maksimiziramo funkciju cilja):

Inicijalizacija:

Postavi da je najbolje rješenje $Z^* = -\infty$

Riješi relaksiranu verziju originalnog problema

Provedi testove eliminacije

Ako nije eliminiran smjestiti ovaj problem u skup podproblema za obradu

Dok (Skup podproblema za obradu nije prazan) {

1. *Grananje:*
   1.1 među preostalim podproblemima odaberi, prema nekom kriteriju, problem za obradu, neka se on zove P
   1.2 među svim varijablama koje bi trebale biti cjelobrojne a to nisu odaberi jednu za grananje po nekom kriteriju, neka se ona zove x_i *i njezina vrijednost u rješenju relaksiranog problema neka je* x_i*
   1.3 stvori dva nova podproblema identična problemu P uz to da prvi ima dodano ograničenje $x_i \leq \lfloor x^* \rfloor$ *a drugi dodano ograničenje* $x_i \geq \lceil x^* \rceil$
2. *Računanje ograda*
   Za svaki od stvorenih problema izračunaj njegovu gornju ogradu (vrijednost funkcije cilja u relaksiranom problemu) koristeći simpleksni ili dualni simpleksni postupak.
3. *Testovi eliminacije, za svaki od stvorenih podproblema provedi testove eliminacije dane u nastavku:*
   Test 1: ako je gornja ograda u čvoru $\leq Z^*$ *eliminiraj čvor*
   Test 2: ako je rješenje nemoguće eliminiraj čvor
   Test 3: ako je rješenje cjelobrojno
   eliminiraj čvor (ako je rješenje bolje od Z osvježi Z* i primjeni Test 1 na sve podprobleme u skupu podproblema sa sada većim Z*)*

}

*Kraj algoritma, optimalno rješenje nalazi se u Z**

Pseudokod nešto niže razine može se naći u [Kalpić, 1996]. Opis nekih kriterija za izbor varijable grananja i sljedećeg problema za obradu dan je u poglavlju 4.1.6.

4.1.4 Primjer rješavanja MIP problema

Kako bi postupak izložen u prethodnom poglavlju bio jasniji pokazati ćemo što se događa na primjeru [Hillier, 2001]. Rješavati ćemo sljedeći mješoviti cjelobrojni program:

$$MAX\ Z = 4x_1 - 2x_2 + 7x_3 - x_4$$

$$uz\ uvijete:$$

$$x_1 + 5x_3 \leq 10, \quad x_1 + x_2 - x_3 \leq 1$$

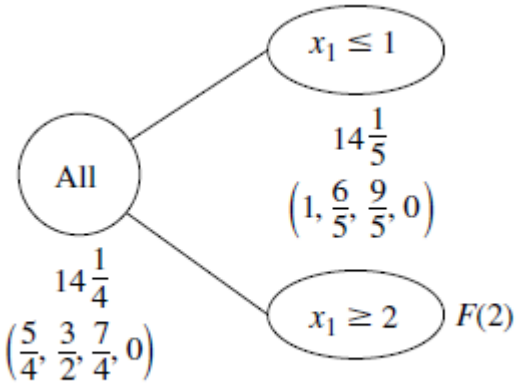
$$6x_1 - 5x_2 \leq 0, \quad 8x_4 \leq 93$$

$$-x_1 + 2x_3 - 2x_4 \leq 12$$

$$x_1, x_2, x_3, x_4 \geq 0$$

$$x_1, x_2, x_3\ cjelobrojne$$

Nakon prve iteracije postupka stanje stabla je prikazano slikom 4.2.

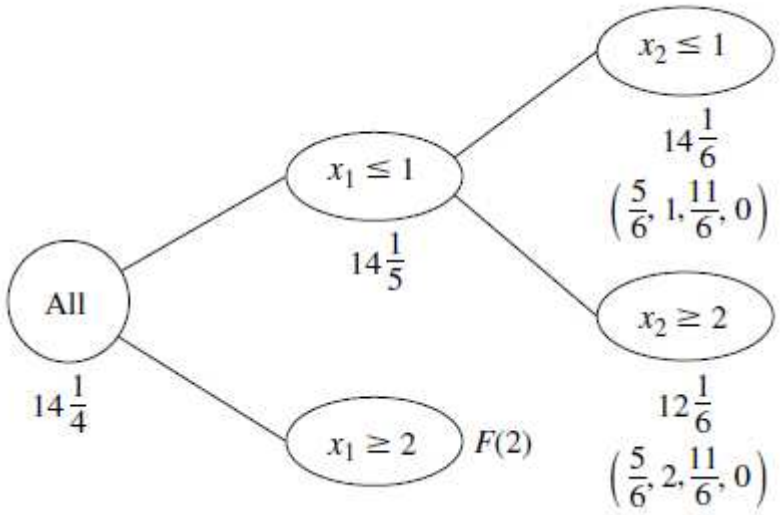


Slika 4.2. Stablo nakon prve iteracije postupka grananja i ograđivanja

Vidimo da je rješenje relaksiranog originalnog problema ($\frac{5}{2}$, $\frac{3}{2}$, $\frac{7}{4}$, 0), uz iznos funkcije cilja 14 $\frac{1}{4}$. Nakon tako obavljene inicijalizacije obavljena je prva iteracija

algoritma. Kao varijabla grananja odabrana je varijabla x1. U podproblemu sa dodanim ograničenjem $x_1 \leq 2$ dobiveno je rješenje $(1, \frac{6}{5}, \frac{9}{5}, 0)$ sa vrijednosti funkcije cilja $14\frac{1}{5}$. Drugi generirani podproblem sa dodanim ograničenjem $x_1 \geq 3$ je elimininiran testom 2 (rješenje je nemoguće).

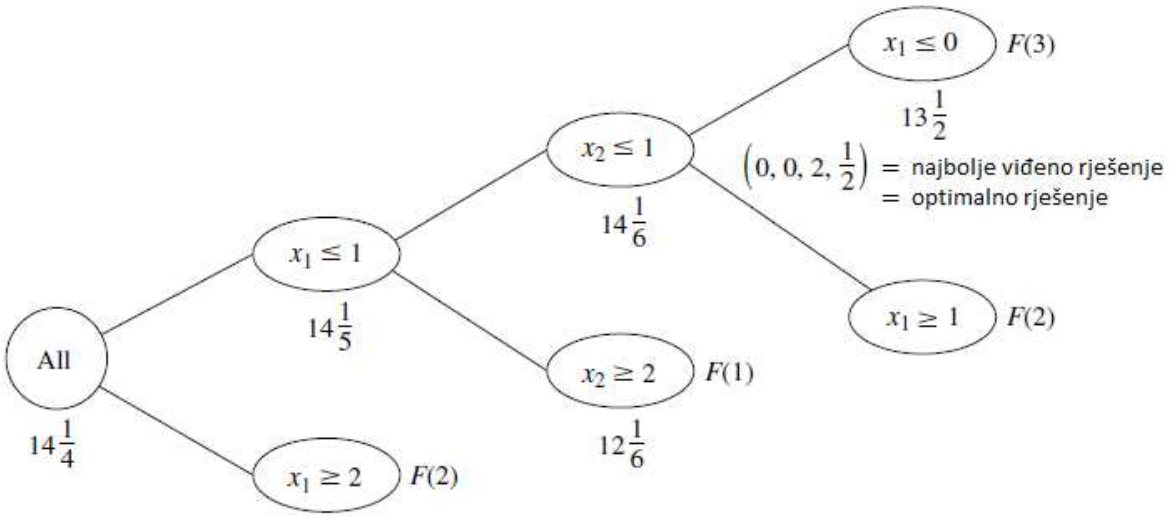
Slika 4.3. prikazuje stanje nakon druge iteracije postupka.



Slika 4.3. stablo nakon druge iteracije postupka grananja i ograđivanja

Obavljeno je grananje iz jedinog čvora koji nije elimininiran. Kao varijabla grananja odabrana je varijabla x2. U čvoru sa dodanim ograničenjem $x_2 \leq 1$ ispalo je rješenje $(\frac{5}{6}, 1, \frac{11}{6}, 0)$ uz vrijednost funkcije cilja $14\frac{1}{6}$. U čvoru sa dodanim ograničenjem $x_2 \geq 2$ ispalo je rješenje $(\frac{5}{6}, 2, \frac{11}{6}, 0)$ uz vrijednost funkcije cilja $12\frac{1}{6}$. Niti jedan od ovih čvorova nije mogao biti elimininiran testovima eliminacije.

Konačno stanje, nakon treće iteracije prikazano je na slici 4.4.



Slika 4.4. stablo nakon treće iteracije postupka grananja i ograđivanja

U trećoj iteraciji imamo dva čvora u skupu neobrađenih podproblema. Jedan od njih ima vrijednost funkcije cilja $14\frac{1}{6}$ dok drugi ima $12\frac{1}{6}$. U ovom primjeru bira se onaj koji ima povoljniju vrijednost , dakle $14\frac{1}{6}$. Kao varijabla grananja odabran je ponovno x1 (može se dogoditi da ista varijabla bude odabrana više puta).

U podproblemu sa dodanim ograničenjem $x_1 \geq 1$ rješenje je nemoguće pa je taj čvor eliminiran.

U podproblemu sa dodanim ograničenjem $x_1 \leq 1$ Izračunato rješenje je (0,0,2,$\frac{1}{2}$) sa vrijednošću funkcije cilja $13\frac{1}{2}$. Ovaj put to je i moguće rješenje (za varijablu x4 nije definirano da mora biti cjelobrojna) što ispunjava treći test eliminacije. Najbolje rješenje se osvježava. Kao što je u pseudokodu napomenuto, primjenjuje se Test 1 (ograđivanje) sa tim sada boljim rješenjem na sve čvorove koji još nisu eliminirani. Taj test eliminira jedini preostali čvor sa funkcijom cilja $12\frac{1}{6}$.

Nema više neobrađenih čvorova pa je postupak gotov. Najbolje viđeno rješenje $13\frac{1}{2}$ uz pripadne vrijednosti varijabli (0,0,2,$\frac{1}{2}$) je ujedno i optimalno rješenje.

4.1.5 Dodatne napomene

Do sada nismo puno pažnje posvetili metodama odabira varijable grananja i sljedećeg čvora za obradu. To vrlo važno i može imati iznimno velik utjecaj na ponašanje algoritma pa ćemo u ovom odjeljku iznijeti neke od mogućih strategija.

Najpoznatije strategije za izbor otvorenog čvora su:

1. Best-first , strategija kao sljedeći otvoreni čvor bira onaj sa najboljom vrijednosti funkcije cilja. Problem sa ovom strategijom je što usmjerava pretragu u širinu pa dugo ne nalazi moguće rješenje. Razlog tome je što čvorovi koji su bliže cjelobrojnosti obično imaju slabiju vrijednost funkcije cilja pa strategija zapravo preferira čvorove koji su dalje od cjelobrojnosti.

2. Depth-first, strategija kao sljedeći otvoreni čvor bira onaj koji je na najvećoj dubini. Prednost ove strategije je što mnogo brže pronalazi inicijalno moguće rješenje.

U [Kalpić, 1996] opisane su i neke druge strategije koje omogućuju relativno brz pronalazak mogućeg rješenja koje je prilično dobro za praktične svrhe. Cijena za takvo ubrzanje je da nemamo dokaza da je to stvarno optimalno rješenje. U [Talbi, 2006] spominje se i oldest-first strategija koja bira najstariji čvor u stablu.

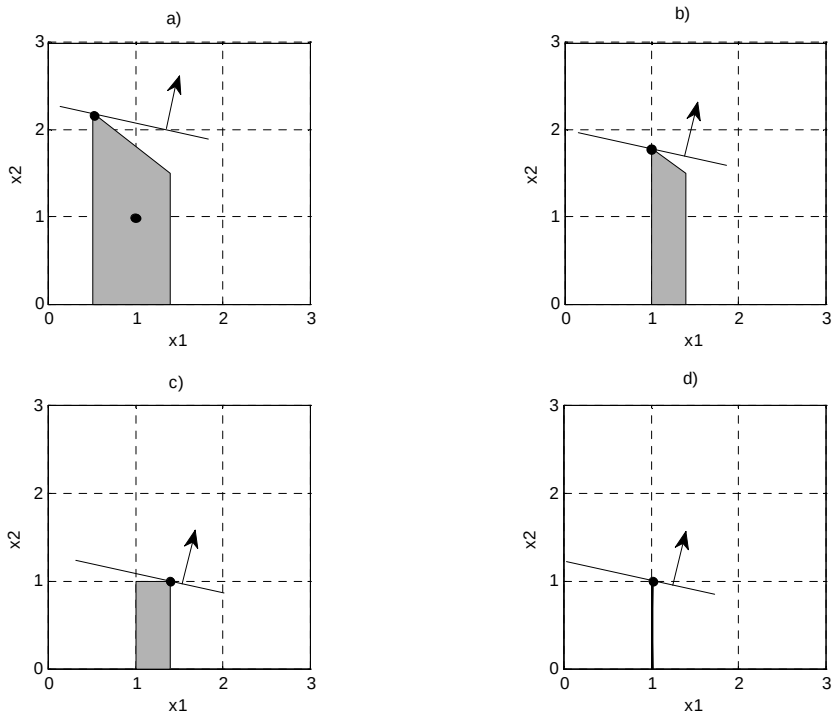
Najčešće strategije za odabir varijable po kojoj ćemo granati:

1. Odabrati onu varijablu koja ima najveći utjecaj na funkciju cilja
2. Birati varijablu koja je najdalje od cjelobrojnosti
3. Kombinacija 1. i 2.

4.1.6 Složenost

Analiza složenosti ovog algoritma je teška jer količina posla koji se obavlja jako ovisi o instanci problema koja se rješava. Ipak u slučaju binarnih programa svaka varijabla se tjeranjem na cjelobrojnost zapravo fiksira. Zbog toga je u binarnim programima gornja granica za broj čvorova je 2^N. To ukazuje na nepolinomijalnu složenost algoritma [Kalpić, 1996]. Ovo je u skladu s činjenicom da se mnogi NP-teški optimizacijski problemi mogu modelirati u obliku binarnih programa.

U slučaju općenitog cjelobrojnog programa granica od 2^N čvorova ne stoji. Razlog tome je što se može dogoditi da tjeranjem neke varijable na cjelobrojnost neka druga varijabla koja je već prije postala cjelobrojna to svojstvo izgubi. Da si to lakše predočimo slika 4.5. prikazuje primjer.



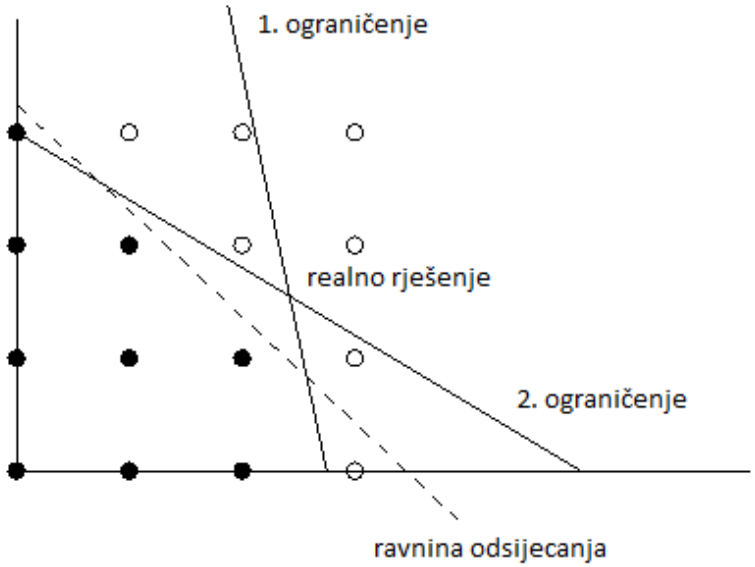
Slika 4.5. primjer slučaja gdje varijabla koja je postala cjelobrojna izgubi to svojstvo (varijabla x1)

Na slici 4.5. a) vidimo sivo označeno područje dopuštenih rješenja za problem. Strelicom je označen smjer rasta funkcije cilja a cjelobrojni optimum je točka (1,1). Rješavanjem ovog problema postupkom grananja i ograđivanja kao varijablu grananja prvo bismo mogli odabrati x1 jer je ona najdalje od cjelobrojnosti. Pri tome bismo prostor mogućih rješenja podijelili na dva dijela. Podproblem za dodanim ograničenjem $x_1 \leq 0$ nema mogućih rješenja dok je podproblem sa dodanim ograničenjem $x_1 \geq 1$ prikazan na slici 4.5. b). Sada još samo varijabla x2 nije cjelobrojna pa nju biramo za varijablu grananja. Slično kao i prije podproblem sa dodanim ograničenjem $x_2 \geq 2$ nema mogućih rješenja dok je problem sa dodanim ograničenjem $x_2 \leq 1$ prikazan na slici 4.5. c). Vidimo sada da smo tjeranjem varijable x2 na cjelobrojnost pokvarili x1 koja je već bila cjelobrojna. Konačno na slici 4.5. d) prikazana je situacija nakon što se obavi još jedno grananje (opet po varijabli x1) i postupak pronalazi optimum.

Općenito pokazuje se da složenost rješavanja nekog IP problema ne ovisi toliko o broju ograničenja kao što je to bilo kod linearnog programiranja. Najveću ulogu igra specijalna struktura pojedinog problema [Hillier, 2001].

4.1.7 Ostali algoritmi za rješavanje MIP problema

Od ostalih algoritama za rješavanja MIP problema vrijedi spomenuti algoritam grananja i rezanja (*engl. Branch and Cut)*. Algoritam koristi ravnine odsijecanja. Ravnina odsijecanja je takvo ograničenje koje je zadovoljeno od strane svih cjelobrojnih točaka u dopuštenom području ali nije zadovoljeno od strane trenutnog realnog rješenja. Jedna ravnina odsijecanja prikazana je na slici 4.6.



Slika 4.6. primjer ravnine odsijecanja

Algoritam grananja i rezanja radi tako da u svakom čvoru stabla čije rješenje nije cjelobrojno generira ravnine odsijecanja kao dodatna ograničenja i na taj način pokušava natjerati rješenje relaksiranog podproblema da postane cjelobrojno. Grananje se obavlja tek ako postupak iscrpi sve ravnine odsijecanja a rješenje i dalje nije cjelobrojno.

Postupak dodavanja ravnina odsijecanja možemo zamisliti kao pokušaj što većeg sužavanja konveksnog skupa mogućih rješenja tako da on obuhvaća samo sva cjelobrojna rješenja. Time se znatno smanjuje i broj grananja potreban da bismo došli do mogućeg rješenja.

Ova metoda pokazala se kao veliko poboljšanje kada se koristi u kombinaciji sa postupkom grananja i ograđivanja.

5. Paralelizacija postupka grananja i ograđivanja

Postupak grananja i ograđivanja može se paralelizirati na nižoj ili višoj razini. Paralelizacija na nižoj razini podrazumijeva da krenemo od slijednog algoritma i paraleliziramo samo jedan njegov dio. To se obavlja na takav način da se interakcija paraleliziranog dijela sa ostalim dijelovima algoritma ne mijenja. U slučaju postupka grananja i ograđivanja može se primjerice paralelizirati rješavanje relaksiranog podproblema (paralelnim postupcima sa matricama) ili izbor sljedećeg podproblema za grananje. Time smo paralelizirali obradu jednog čvora.

Paralelizacija na nižoj razini ne utječe na ponašanje algoritma kao cjeline. On i dalje donosi iste odluke tj. posjećuje iste čvorove istim redoslijedom kao i slijedni algoritam. Zbog toga moguće je dobro predvidjeti ponašanje ovako paraleliziranog algoritma.

Paralelizacija na višoj razini ne utječe na pojedini dio već na algoritam kao cjelinu. Zbog toga ona značajno utječe na ponašanje algoritma. Količina posla koju obavlja takav paralelni algoritam može se razlikovati od količine posla koju bi obavio slijedni algoritam. Redoslijed kojim se posao obavlja također ne mora biti isti. Može se čak dogoditi i da neki dijelovi posla koje bi slijedni algoritam obavio paralelni ne radi, a moguće je i obrnuto. Viša razina paralelizacije u postupku grananja i ograđivanja ostvaruje se tako da dva ili više radnika istovremeno istražuju različite dijelove stabla. Budući da su obrade čvorova koji još nisu eliminirani prilično nezavisne operacije ovo ne utječe na ispravnost postupka.

U nastavku opisani su neki od važnih aspekata paralelnog ostvarenja algoritma grananja i ograđivanja navedeni u [Trienekens, 1989] i [Talbi, 2006]

5.1 Podjela posla

Kako bi se posao mogao podijeliti radnicima potrebno je definirati neki mehanizam kojim će se početni problem podijeliti na više manjih zadataka. Zadatak možemo definirati po volji. Ipak treba obratiti pažnju na činjenicu da

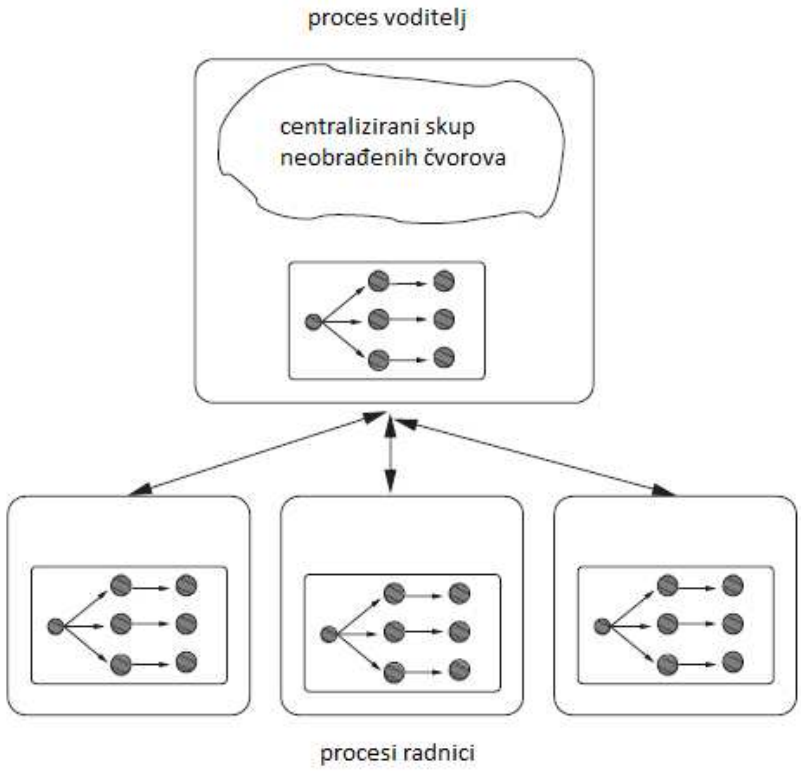
radnik nakon što obavi zadatak mora komunicirati da bi dobavio novi. U slučaju jako malih zadataka količina te komunikacije će biti iznimno velika što može zagušiti komunikacijsku mrežu. Primjeri zadataka koji se navode u [Trienekens, 1989] su grananje iz čvora, optimalno rješavanje podproblema ili izračun ograde u problemima nastalima grananjem.

Do sada neobrađeni zadaci drže se u skupu neobrađenih zadataka. Svaki novi zadatak generiran u postupku rješavanja dodaje se u skup neobrađenih zadataka. Kada neki radnik ostane bez posla dobavlja novi zadatak također iz tog skupa. Slično kao što slijednom algoritmu treba neki kriterij za odabir sljedećeg čvora za obradu i radniku u paralelnom algoritmu potreban je neki kriterij po kojem odabire sljedeći zadatak koji preuzima. Ti kriteriji mogu biti slični onima opisanima u poglavlju 4.1.5.

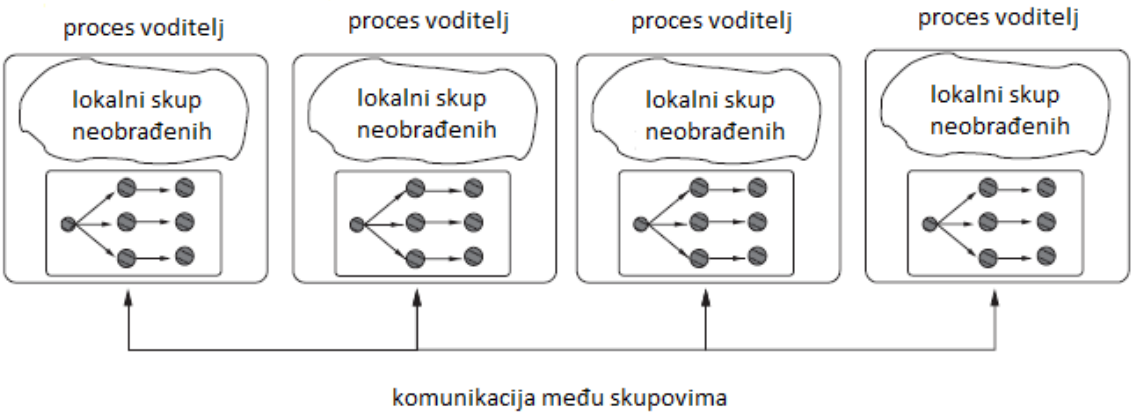
Prednost jednog centraliziranog skupa zadataka je što na jednom mjestu imamo dobar pregled preostalog posla. Zbog toga je lako svakom radniku dodijeliti dobar zadatak. Ipak budući da radnici mogu skupu zadataka pristupati samo pojedinačno uz veći broj radnika skup zadataka može lako postati usko grlo.

U slučaju da koristimo više skupova zadataka izbjegavamo problem uskog grla. Cijena koju za to plaćamo jest da će neki procesi raditi na lošijim zadacima samo zato jer u trenutku kad im je zatrebao zadatak u njihovom skupu nije bilo boljeg zadatka. Pri tome se bolji zadatak možda nalazio u nekom o drugih skupova.

U implementaciji skup zadataka najčešće se realizira pomoću procesa voditelja koji upravlja radnicima i dodjeljuje im zadatke na njihov zahtjev. U slučaju centraliziranog skupa zadataka imamo jednog voditelja koji poslužuje sve radnike (Slika 5.1.). U slučaju tzv. kolegijalne organizacije imamo više procesa voditelja od kojih svaki poslužuje svoj skup radnika (Slika 5.2.). Pri tome voditelji međusobno razmjenjuju informacije važne za algoritam kao što su najbolje rješenje, ograde i ujednačavanje opterećenja.



Slika 5.1. ilustracija centralizirane organizacije skupa neobrađenih zadataka



Slika 5.2. ilustracija kolegijalne organizacije skupa neobrađenih zadataka

5.2 Komunikacija

Komuniciranje među procesima jako je bitno jer na taj način distribuiramo ukupno znanje o problemu svim procesima. Bolje znanje o problemu pomaže postupku da donosi bolje odluke pa tako i smanji ukupnu količinu posla. Cijena koju za to plaćamo su povećani troškovi komunikacije.

Također, potrebno je definirati ponašanje procesa nakon dolaska novog znanja. Jedna mogućnost je da se ono ne uzima u obzir prije nego obrada trenutnog zadatka bude gotova. Alternativno mogli bismo prekinuti obradu trenutnog zadatka i odmah uzeti u obzir novo znanje. Takav pristup mogao bi uzrokovati da proces privremeno odgodi obradu zadatka na kojem radi jer se pojavio neki zadatak većeg prioriteta. Obrada se može čak i potpuno prekinuti jer je stigla nova ograda koja eliminira taj zadatak. Takvim pristupom dobiti ćemo kvalitetniji algoritam ali uz cijenu složenije implementacije i dodatnog utroška vremena na obradu pristiglog znanja.

5.3 Sinkronost

Paralelni algoritam grananja i ograđivanja je sinkron ako svi radnici prije nego što krenu sa obradom sljedećeg zadatka čekaju da svi ostali zadaci u njihovom skupu završe obradu. Budući da vrijeme rješavanja nije isto za sve zadatke, neki radnici morati će trošiti vrijeme na čekanje ostalih da završe obradu. To je glavna mana ovog pristupa.

U asinkronom pristupu radnici čim završe obradu dobavljaju novi zadatak pa nema nepotrebnog čekanja. Za razliku od sinkronog slučaja tu se može pojaviti nedeterminizam. On se javlja zato što se mogu pojaviti varijacije u redoslijedu dodavanja i uzimanja zadataka iz skupa zadataka kao i trenutku u kojem će neki proces biti obaviješten o npr. ogradi. Primjerice ako se zamjeni redoslijed operacija dodavanja zadatka u skup zadataka od strane jednog procesa i uzimanja zadatka iz istog skupa od strane nekog drugog procesa konačni tijek postupka može se promijeniti.

5.4 Anomalije

U slučaju paralelnog algoritma grananja i ograđivanja koji se izvodi na p procesora očekivali bismo ubrzanje od p puta. U skladu s time ako je vrijeme izvođenja slijednog algoritma jednako T očekivali bismo da vrijeme izvođenja

paralelnog algoritma bude $T_p = \frac{T}{p}$. Ipak ponekad ono može značajno odstupati od ove vrijednosti . U praksi mogu se pojaviti sljedeći slučajevi :

1. $\frac{T}{p} \leq T_p \leq T$ (anomalija kvarenja, *engl. detrimental anomaly*)
2. $T_p \leq \frac{T}{p}$ (anomalija prevelikog ubrzanja, *engl. acceleration anomaly*)
3. $T_p \geq T$ (anomalija usporenja, *engl. deceleration anomaly*)

Spomenute anomalije javljaju se zbog činjenice što količina posla koju obavlja paralelni algoritam nije jednaka količini posla koju obavlja slijedni algoritam. Primjeri ovih anomalija uz detaljnije objašnjenje nalaze se u poglavlju 7.

U [Talbi, 2006] navodi se učinkovitost pojedinih pravila za izbor otvorenog čvora s obzirom na mogućnost pojave anomalija. Pokazuje se da depth-first strategija ima slabu učinkovitost. Best-first strategija postiže dosta dobre rezultate i na p procesora daje ubrzanje jako blizu p. Zanimljiv podatak je također da se kao najotpornija na anomalije pokazuje oldest-first strategija koja bira najstariji čvor u stablu.

6. Implementacija

U okviru rada napravljen je program MIPS (kratica za MIP solver) ,koji predstavlja paralelnu implementaciju algoritma grananja i ograđivanja. Program se može koristiti kao alat za rješavanje problema koji se mogu formulirati u obliku MIP problema. Radi jednostavnosti pretpostavlja se da je problem u standardnom obliku.

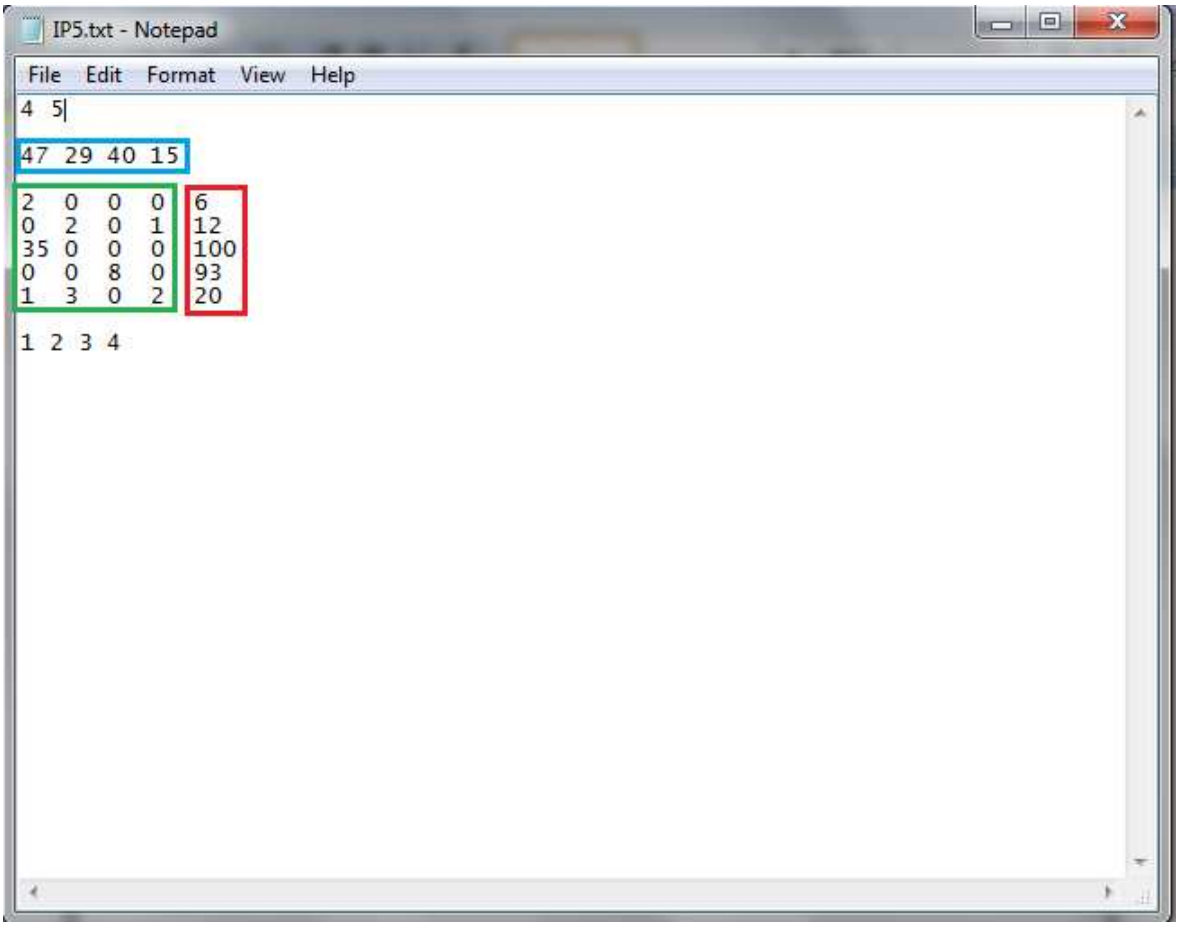
Program kao ulaz prima datoteku koja sadrži opis mješovitog cjelobrojnog programa koji je potrebno riješiti a kao izlaz generira datoteku sa rezultatom u kojoj je zapisano optimalno rješenje – iznosi svih varijabli u modelu i iznos funkcije cilja.

Kao programski jezik implementacije odabran je C. Razlog tome jest najviše autorovo veće iskustvo u korištenju MPICH2 biblioteke sa baš ovim programskim jezikom. Također kao jezik nešto niže razine C nam može ponuditi nešto veću brzinu u odnosu na npr. C# ili Javu.

Komunikacija među paralelnim procesima ostvarena je pomoću MPICH2 biblioteke. Radi se o slobodno dostupnoj biblioteci koja predstavlja implementaciju MPI (*engl. Message passing interface*) standarda za komunikaciju procesa. Ova biblioteka oslobađa programera brige o tome gdje se nalaze procesi koji moraju komunicirati (na dvije jezgre istog procesora, dva računala u mreži …). Funkcijama iz ove biblioteke komunikacija među svim procesima za potrebe paralelizacije može se ostvariti na jednostavan način. Inačica korištena u ovom radu je 1.0.8p1.

6.1 Formati ulaznih i izlaznih datoteka

Ulazne datoteke za program su u formatu prikazanom na slici:



Slika 6.1. primjer ulazne datoteke za program

Prvo se navode 2 broja – broj varijabli i broj ograničenja. Nakon toga slijedi vektor **c** (označeno plavo na slici). Zatim idu matrica **A** i vektor **b** (zeleno i crveno na slici). Zadnje dolazi popis varijabli koje moraju biti cjelobrojne. U skladu s tim, problem prikazan datotekom na slici je sljedeći:

$$MAX\ Z = 47x_1 + 29x_2 + 40x_3 + 15x_4$$

$$uz\ uvjete:$$

$$2x_1 \leq 6, \quad 2x_2 + x_4 \leq 12$$

$$35x_3 \leq 100, \quad 8x_4 \leq 93$$

$$x_1 + 3x_2 + 2x_4 \leq 20$$

$$x_1, x_2, x_3, x_4 \geq 0$$

$$x_1, x_2, x_3, x_4 \ \ cjelobrojne$$

U izlaznoj datoteci zapisane su vrijednosti svih varijabli kao i konačna pronađena optimalna vrijednost funkcije cilja. Primjer izgleda izlazne datoteke može se pronaći u odjeljku 6.7.

6.2 Osnovna implementacija

Rješavanje zadanog MIP problema odvija se tako da prvo pomoću ulazne datoteke generira početna simpleks tablica. Ta se tablica potom optimizira simpleks postupkom čime se dobiva optimalno rješenje koje najčešće krši neka ograničenja cjelobrojnosti.

Tako optimiziranu tablicu predajemo rekurzivnoj funkciji koja provodi postupak grananja i ograđivanja kakav je opisan u poglavlju X po sljedećem malo drugačijem pseudokodu.

```
Branch&Bound ( tablica )

  v = varijabla najdalje od cjelobrojnosti

  novaTablicaManje = DodajOgranicenje(tablica, v, manje)

  novaTablicaVece = DodajOgranicenje (tablica,v,vece)

  riješi LP relaksaciju za obje tablice (optimiziraj ih)

  provjeri uvjete zaustavljanja za obje tablice

  ako je potrebno pozovi Branch&Bound (novaTablicaManje)

  ako je potrebno pozovi Branch&Bound(novaTablicaVece)
```

Iz pseudokoda vidi se da se koristi depth-first strategija obilaska stabla. Ona je odabrana zato što je štedljiva u smislu memorijskog prostora što je važno ako pamtimo cijelu simpleks tablicu u svakom čvoru.

Nova ograničenja mogla bi se dodati u početnu simpleks tablicu no rješavanje LP relaksacija takvih tablica (svaki put od početka) trajalo bi jako dugo. Kako bi se to izbjeglo ograničenja se dodaju tako da se iz postojeće optimalne

simpleks tablice generira nova tablica sa dodanim ograničenjem. Takva nova tablica se može mnogo efikasnije optimizirati koristeći dualnu simpleks metodu. Način dodavanja ilustriramo sljedećim primjerom.

Pretpostavimo da u jednom trenutku postupka grananja i ograđivanja imamo sljedeću (optimalnu) simpleks tablicu (varijable x3 i x4 su dopunske).

Tablica 6.1. početna simpleksna tablica na kojoj ćemo pokazivati dodavanje ograničenja

Bazična var.	**Z**	**X1**	**X2**	**X3**	**X4**	**Desna str.**
Z	1	2	0	3	0	35
X2	0	1	1	1	0	4.2
X4	0	2	0	0	1	3

Bazično rješenje koje se može očitati jest (0, 4.2, 0, 3) sa vrijednosti funkcije cilja 35. Ipak ovo optimalno rješenje nam nije prihvatljivo zbog toga što varijabla x2 ima vrijednost koja nije cjelobrojna. Moramo se dakle dalje granati u dva podproblema sa ograničenjima $x2 \leq 4$ i $x2 \geq 5$. Ograničenje $x2 \leq 4$ dodati ćemo tako da dodamo jednadžbu $x2 + x5 = 4$ u tablicu, x5 jest nova dopunska varijabla koju ćemo proglasiti i bazičnom (odgovara joj zadnji red tablice). Varijablu x5 možemo interpretirati kao udaljenost varijable x2 od dodanog ograničenja $x2 \leq 4$.

Tablica 6.2. početna simpleksna tablica uz dodanu jednadžbu x2+x5 = 4

Bazična var.	**Z**	**X1**	**X2**	**X3**	**X4**	**X5**	**Desna str.**
Z	1	2	0	3	0	**0**	35
X2	0	1	1	1	0	**0**	4.2
X4	0	2	0	0	1	**0**	3
X5	**0**	**0**	**1**	**0**	**0**	**1**	**4**

Problem s ovom tablicom je što u retku bazične varijable x5 varijabla x2 koja je također bazična ima koeficijent različit od 0. Zbog toga se rješenje ne može izravno očitati iz tablice. To možemo jednostavno riješiti tako da od retka varijable x5 oduzmemo redak varijable x2 čime će problematični koeficijent postati 0. Dobiti ćemo sljedeću tablicu:

Tablica 6.3. tablica nakon završenog postupka dodavanja ograničenja x2≤4

Bazična var.	Z	X1	X2	X3	X4	X5	Desna str.
Z	1	2	0	3	0	**0**	35
X2	0	1	1	1	0	**0**	4.2
X4	0	2	0	0	1	**0**	3
X5	**0**	**-1**	**0**	**-1**	**0**	**1**	**-0.2**

Varijabla x5 ima vrijednost -0.2 što je u skladu sa njenom interpretacijom kao udaljenosti varijable x2 od ograničenja x2≤4, ograničenje je prekoračeno za 0.2. Kako redak funkcije cilja nismo dirali uvjet optimalnosti je i dalje zadovoljen.

U slučaju ograničenja x2≥5 postupak je veoma sličan no dodali bismo u tablicu jednadžbu x2 – x5 = 5, u obliku –x2+x5 = -5 kako slijedi:

Tablica 6.4. početna simpleksna tablica uz dodanu jednadžbu -x2+x5 = -5

Bazična var.	Z	X1	X2	X3	X4	X5	Desna str.
Z	1	2	0	3	0	**0**	35
X2	0	1	1	1	0	**0**	4.2
X4	0	2	0	0	1	**0**	3
X5	**0**	**0**	**-1**	**0**	**0**	**1**	**-5**

Dopunsku varijablu x5 sada možemo interpretirati kao udaljenost varijable x2 od ograničenja x2≥5.

Sada se slično kao i prije javlja problem. U retku bazične varijable x5 varijabla x2 koja je također bazična ima koeficijent različit od 0. Ovaj put ćemo to riješiti tako što ćemo retku varijable x5 dodati redak varijable x2 čime ćemo pokratiti problematični koeficijent. Rezultat je sljedeća tablica:

Tablica 6.5. tablica nakon završenog postupka dodavanja ograničenja x2≥5

Bazična var.	**Z**	**X1**	**X2**	**X3**	**X4**	**X5**	**Desna str.**
Z	1	2	0	3	0	**0**	35
X2	0	1	1	1	0	**0**	4.2
X4	0	2	0	0	1	**0**	3
X5	**0**	**1**	**0**	**1**	**0**	**1**	**-0.8**

Iznos x5 ispao je -0.8 što je opet u skladu sa interpretacijom x5 jer je ograničenje x2≥5 prekoračeno za 0.8.

U oba slučaja postupkom za dodavanje ograničenja smo dobili tablicu koja definira rješenje koje je optimalno ali nemoguće. Takvo rješenje je moguće za dualni problem pa možemo upotrijebiti dualnu simpleksnu metodu za učinkovitu reoptimizaciju ove tablice (najčešće u jako malo iteracija). Time ćemo dobiti opet optimalnu simpleks tablicu pomoću koje možemo po potrebi dalje granati. Postupak staje kada obradimo sve čvorove B&B stabla.

6.3 Paralelna implementacija

Procesi koji paralelno izvode program su podijeljeni na voditelja i radnike, voditelj čeka zahtjeve radnika i raspoređuje im poslove koje oni obavljaju i šalju mu rješenja.

Zadaci se generiraju tako da se pokrene standardni Branch & Bound postupak koji se grana do neke zadane maksimalne dubine, kada postupak stigne

do te dubine sprema stanje (čvor stabla tj. tablicu) u listu zadataka za daljnju obradu. Nakon što su zadaci generirani voditelj ih počinje raspoređivati radnicima.

Svaki zadatak sadrži sljedeće podatke:

1. Broj varijabli
2. Broj ograničenja
3. Id zadatka (koristi se pri zapisivanju u dnevnik)
4. Simpleks tablicu
5. Popis varijabli koje su bazične
6. Popis varijabli koje moraju biti cjelobrojne

Zadaci se obavljaju asinkrono što znači da kada radnik završi sa obavljanjem zadatka ne čeka da ostali radnici završe sa svojim zadacima već odmah šalje rezultate voditelju uz zahtjev za novim zadatkom.

Poruka o spremnosti koju radnici šalju voditelju sadrži:

1. Optimalne vrijednosti varijabli u riješenom zadatku
2. Optimalnu vrijednost funkcije cilja u riješenom zadatku

Po primitku ovakve poruke voditelj će, ako je to potrebno, osvježiti svoje najbolje rješenje. Također voditelj će radniku poslati novi zadatak ili, ako nema više zadataka, poruku da može završiti s radom.

Postupak završava kada su obrađeni svi generirani zadaci i poslane poruke o završetku svim radnicima.

Budući da se zadaci generiraju kada postupak stigne do neke fiksne dubine koja je predodređena parametrima programa njihov broj je statički. To bi moglo dovesti do problema sa neujednačenim opterećenjem ako je ta dubina premala. Tijekom testiranja pokazalo se da uz dubinu od oko 8 (čak i dosta manje) nije bilo neujednačenosti u opterećenju. Ipak za jako velike probleme za kakve ovaj program nije predviđen bilo bi bolje implementirati neku vrstu dinamičke raspodjele posla.

6.4 Pseudokod

Pseudokod voditelja:

```
generiraj zadatke
while(brPoslanoGotovo < brRadnika)
 primi poruku o spremnosti (od nekog radnika i)
   ako je uz poruku prilozeno rj.
      osvjeziti optimum po potrebi
   ako ima jos nedodjeljenih zad
      posalji sljedeci zadatak sa popisa radniku i
      makni taj zadatak sa popisa zadataka
   inace
      posalji radniku i poruku da je gotovo
      brPoslanoGotovo ++
```

Pseudokod radnika:

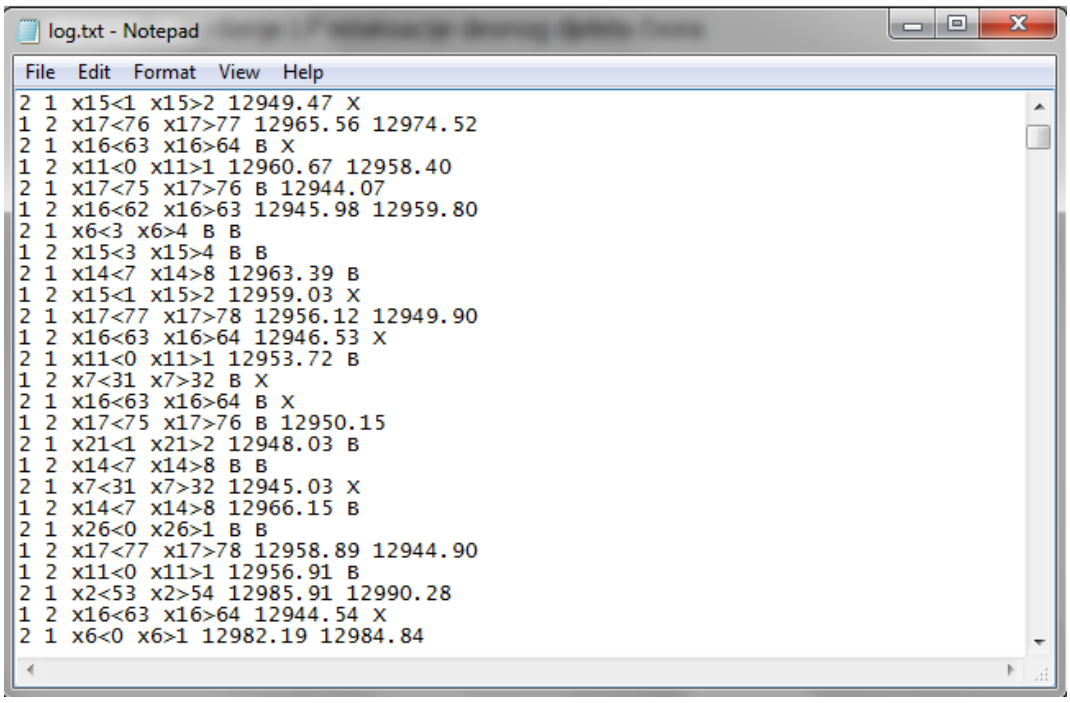
```
posalji voditelju poruku o spremnosti (bez prilozenog rj)
while(1)
  primi odgovor od voditelja
  ako je u odgovoru zadatak
    rijesi zadatak
    posalji voditelju poruku o spremnosti (i prilozeno rj)
  inace (u odgovoru je poruka da je kraj)
    break
```

U pseudokodovima nije posebno naglašeno no iznimno je važno da svi procesi uvijek imaju najsvježiju gornju granicu. To se osigurava tako da svaki radnik kada pronađe granicu bolju od trenutno najbolje svim ostalim radnicima šalje poruku sa iznosom nove granice. Prisutnost poruke o novoj granici svaki radnik provjerava pozivom neblokirajuće funkcije za primanje pri početku razmatranja svakog čvora B&B stabla.

6.5 Dnevnik optimizacije

Prilikom optimizacije svi radnici i voditelj zapisuju podatke o čvorovima koje obrađuju u dnevnik optimizacije. On služi tome da bi se nakon optimizacije moglo vidjeti što je algoritam radio i koje je čvorove posjećivao. Dnevnik optimizacije je zapravo preorder zapis stabla pretrage. Primjer takvog dnevnika prikazan je na slici. U svakom retku dnevnika zapisani su sljedeći podaci:

1. Id procesora koji je obavio obradu čvora
2. Id zadatka kojem čvor pripada
3. Natpis na lijevoj grani iz čvora (u obliku varijabla >/< vrijednost)
4. Natpis na desnoj grani iz čvora (u obliku varijabla >/< vrijednost)
5. Rješenje LP relaksacije lijevog djeteta čvora
6. Rješenje LP relaksacije desnog djeteta čvora

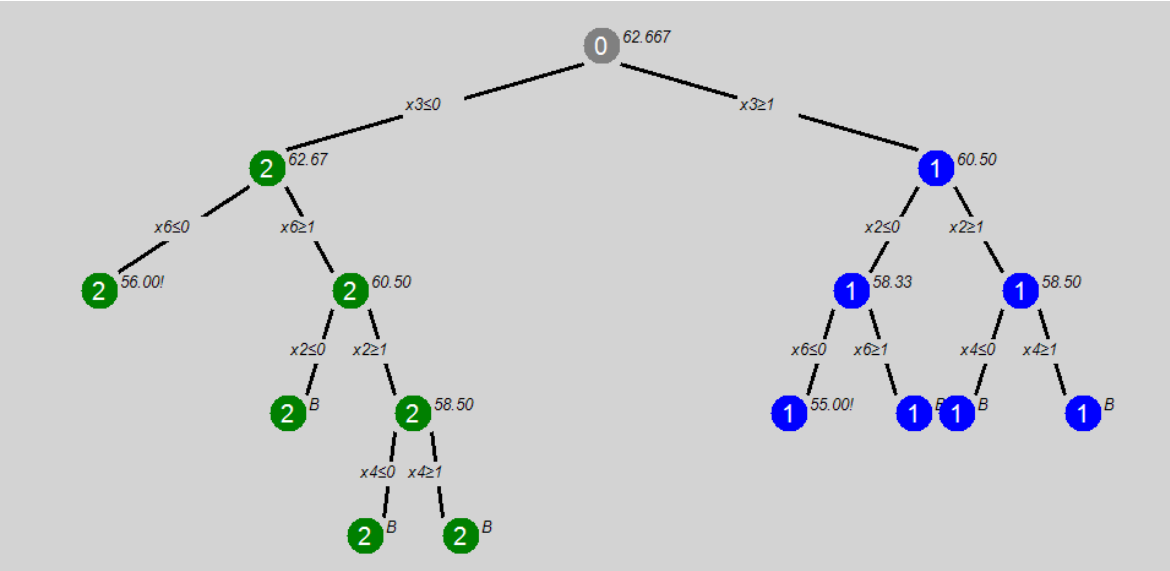


Slika 6.2. primjer izgleda dnevnika optimizacije

6.6 Vizualizacija tijeka algoritma

Kako je za čovjeka jako teško čitati dnevnik optimizacije i iz njega izvoditi neke zaključke o tijeku algoritma napravljen je pomoćni program Log viewer. Program je napravljen u jeziku C# i služi za otvaranje i vizualizaciju datoteka dnevnika optimizacije.

Program iscrtava stablo pretrage. Na slici je primjer takvog stabla:



Slika 6.3. primjer stabla koje iscrtava program Log viewer

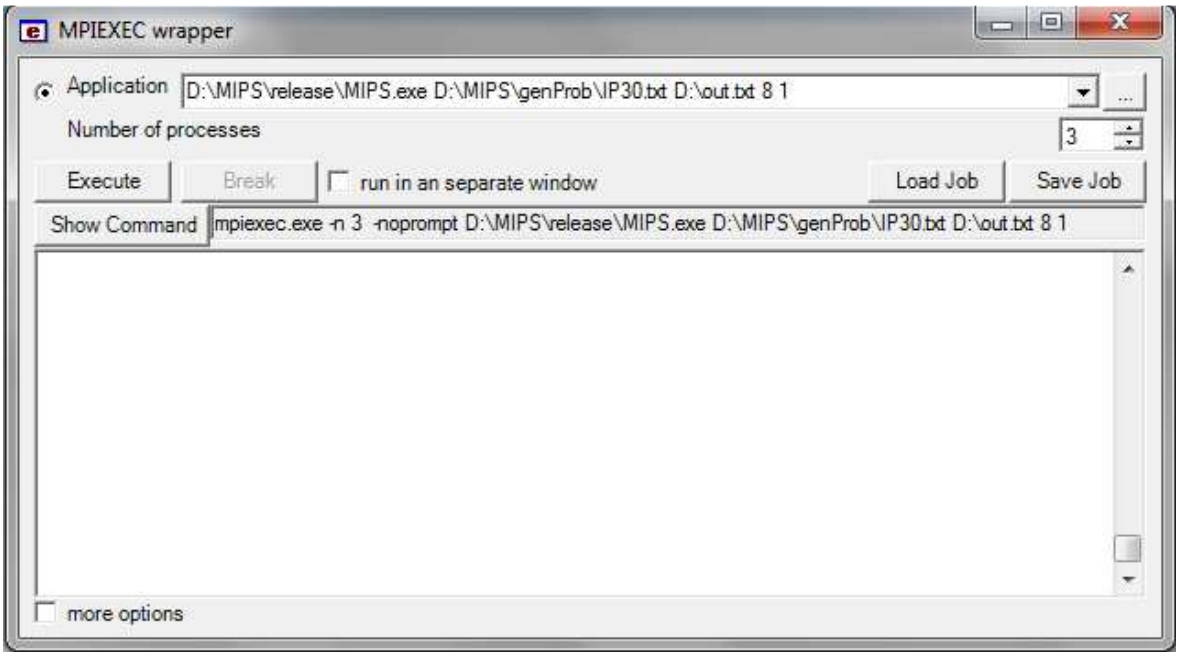
Različiti zadaci obojani su različitim bojama (na slici zeleno i plavo). Čvorovi koje obrađuje voditelj obojani su sivo. Brojevi u čvorovima označavaju koji radnik je obradio pojedini čvor. Na slici, plavi zadatak obradio je radnik 1 dok je zeleni zadatak obradio radnik 2. Na granama iz pojedinog čvora prema njegovoj djeci označena su ograničenja koja se tim prijelazom dodaju. Iznad svakog čvora piše vrijednost funkcije cilja u njegovoj LP relaksaciji. Ako nakon vrijednosti funkcije cilja stoji znak ' ! ' to znači da je to moguće rješenje. Umjesto te vrijednosti može stajati i ' X ' što znači da je rješenje nemoguće ili ' B ' što znači da je čvor eliminiran pomoću gornje ograde.

6.7 Pokretanje programa

Program ima četiri argumenta.

1. Ime ulazne datoteke
2. Ime izlazne datoteke
3. Dubinu na kojoj se generiraju zadaci
4. 0 ili 1 – kontrolira hoće li si radnici međusobno slati najbolju granicu (koristi se za pokazivanje nekih svojstava algoritma)

Program se pokreće pomoću programa wmpiexec. To je sučelje MPICH2 biblioteke za pokretanje paralelnih programa. Sučelje omogućuje definiranje broja procesa (radnika) u paralelnom algoritmu. Za naprednije opcije poput određivanja na kojem računalu u mreži će biti koji radnik potrebno je to navesti u datoteci. Detaljne upute mogu se pronaći u [MPICH2, 2009].

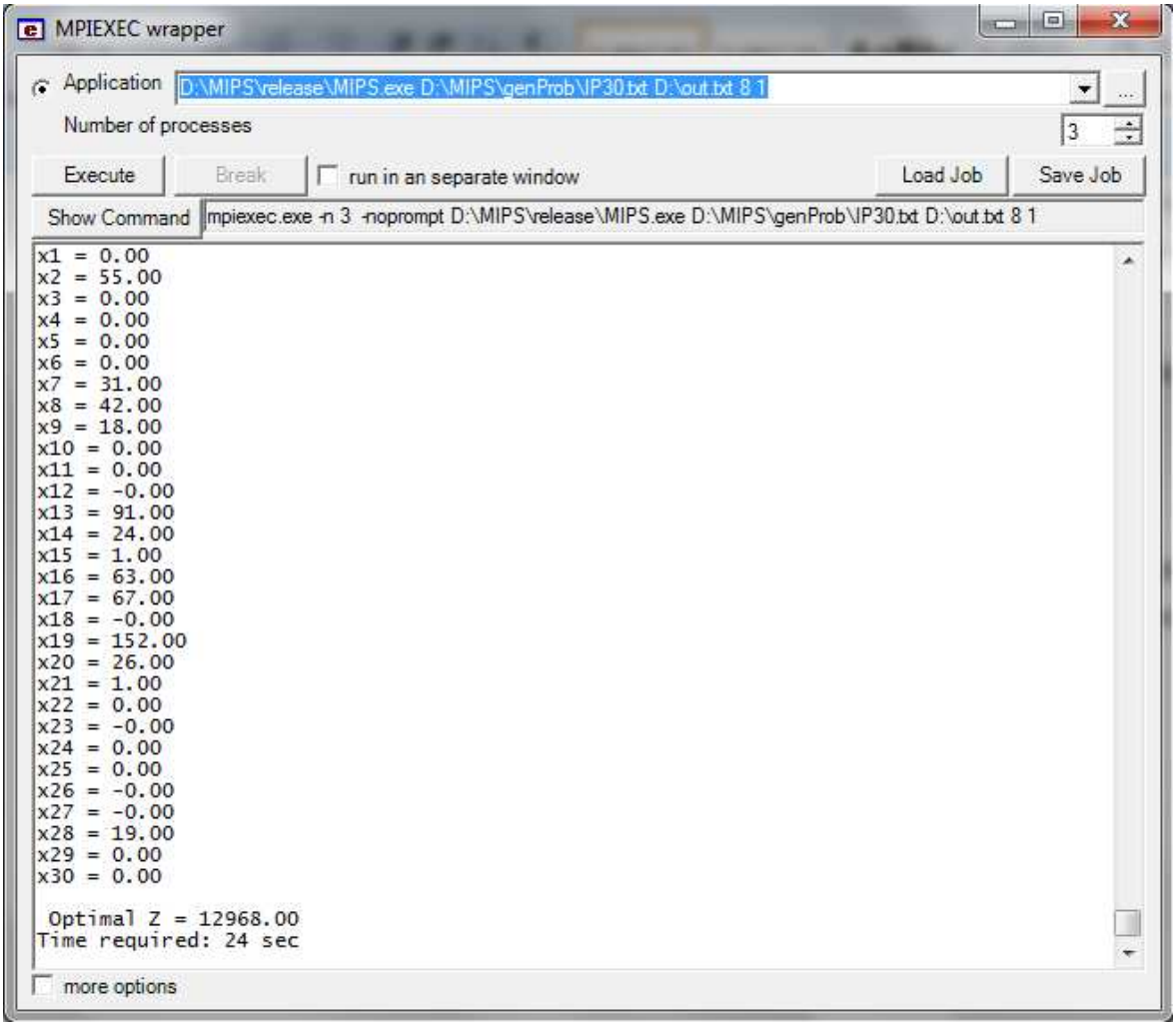


Slika 6.4. osnovno grafičko sučelje programa wmpiexec

U ispitivanju programa procesi su pokretani isključivo na jednom računalu tj. na jezgrama višejezgrenog procesora.

Također vrijedi napomenuti da proces voditelj nije imao svoj vlastiti procesor (jezgru) već je resurse dijelio sa nekim od radnika. Budući da obrada zahtjeva za poslom koju voditelj obavlja traje vrlo kratko a i zahtjevi ne dolaze prečesto smatrano je da je dodatno opterećenje koje generira voditelj zanemarivo. Ipak u slučaju jako velikog broja radnika bilo bi bolje voditelja rasporediti na poseban procesor.

Konačno, nakon što pokrenemo program i pričekamo neko vrijeme dobiti ćemo rezultat kao što je prikazano na slici.



Slika 6.5. primjer izlaza programa

Navedene su vrijednosti svih varijabli kao i optimalna vrijednost funkcije cilja. To je ujedno zapisano i u izlaznu datoteku. Također u direktoriju programa pojavila se datoteka log.txt u kojoj je zapisan dnevnik optimizacije.

6.8 Primjer koda

Na sljedeće dvije stranice dano je nekoliko primjera koda programa.

Prvo funkcija koja ostvaruje simpleks postupak:

```
int simplex(float *tab, int brVar, int brOgr, int *baza){
    int sirina = brOgr+brVar+1;
    int ulBaz, izlBaz,ret = SIM_OK;
    float minKoef, minOmjer,omjer,koef,stozer;
    char gotovo = 0;

      while(1){
        //trazenje ulazne bazicne varijable
        minKoef = 0; //trazimo najnegativniji koeficijent
        char nasaoKoef = 0;
        for(int stu = 0; stu<(sirina-1);stu++){
      if(tab[0*sirina + stu]<minKoef){
              ulBaz = stu;
            minKoef = tab[0*sirina + stu];
              nasaoKoef = 1;
            }
        }
        if(!nasaoKoef){ //  nema negativnih -> optimum
              ret = SIM_OK;
              break;
        }
        //trazenje izlazne bazicne varijable -> minimum ratio test
        minOmjer = 0;
        char nasaoOmjer = 0;
        for(int red=1;red<=brOgr;red++){
              koef = tab[red*sirina + ulBaz];
              //ako je unutar EPSILON od 0 smatra se da je 0
              if(koef>EPSILON){
                 //  desna strana / koef.
                 omjer = tab[red*sirina + sirina-1]/koef;
                 //ako je ovo manji omjer ili prvi koji je pronadjen
                  if(omjer<minOmjer || !nasaoOmjer){
                    izlBaz = red;
                    minOmjer = omjer;
                    nasaoOmjer = 1;
                  }
              }
        }
       // ako nema omjera -> neogranicena Z
        if(!nasaoOmjer){
          ret = SIM_UNB;
            break;
        }

     //transformacije da dobijemo zeljeni oblik tablice za iducu
     iteraciju...
        stozer = tab[izlBaz*sirina + ulBaz];
       //dijelimo stozerni red sa stozerom
        for(int i = 0; i<sirina;i++){
              tab[izlBaz*sirina + i]=tab[izlBaz*sirina + i] /stozer;
        }

        for(int red=0;red<=brOgr;red++){  //u svim redovima
```

```
            if(red != izlBaz){ // osim stozernog
          //koeficijent u stupcu ulazne baz. var. i redku koji poništavamo
                koef = tab[red*sirina+ulBaz];
                for(int stu = 0; stu<sirina;stu++){ //dodati stozerni red
           pomnozen s  koeficijentom * -1
                  tab[red*sirina+stu] =
                  tab[red*sirina+stu]+tab[izlBaz*sirina + stu]*koef*(-1);

                }
              }
         }

       baza[izlBaz]=ulBaz;
       }
       return ret;
}
```

Drugo, funkcija koja u procesu voditelju implementira primitak poruke o spremnosti od procesa radnika:

```
int primiSpreman(int brVar){
   char *rBuff = (char *) malloc((brVar+1)*sizeof(float)); //brVar
floatova u optRj i 1 float u optZ
   MPI_Status status;

MPI_Recv(rBuff,(brVar+1)*sizeof(float),MPI_CHAR,MPI_ANY_SOURCE,MPI_ANY_TA
G,MPI_COMM_WORLD,&status);
   if(status.MPI_TAG == RJESENJE){ //ako je rj onda mozda treba zapamtiti
       float oZ = *((float *)rBuff);
       if(oZ>optZ){ //ako je dobiveni opt bolji od trenutno najboljeg opt
       optZ = oZ;
         memcpy(optRj,rBuff+sizeof(float),brVar*sizeof(float));
//kopiramo to novo bolje rj preko staroga
       }
   }
   printf("Primio spremnost od %d tag
%d\n",status.MPI_SOURCE,status.MPI_TAG); fflush(stdout);
   free(rBuff);
   return status.MPI_SOURCE;
}
```

7. Ispitivanje

U ovom poglavlju ispituju se svojstva ovog programa i ilustriraju neke anomalije koje se mogu dogoditi. Sva ispitivanja osim zadnjeg obavljana su na intel i3 – M330, 2.13GHz procesoru sa dvije jezgre.

7.1 Način ispitivanja

Ispravnost i ponašanje programa ispitivani su na slučajno generiranim problemima. Prva vrsta su bili IP problemi u standardnom obliku:

$$MAX\ Z = \boldsymbol{c}^T x$$

$$uz\ uvijete:$$

$$\boldsymbol{A} \cdot \boldsymbol{x} \leq \boldsymbol{b}$$

$$\boldsymbol{x} \geq \boldsymbol{0}$$

$$sve\ x_i\ \ cjelobrojne$$

Koeficijenti u vektoru ***c*** odabrani su kao slučajni brojevi između 1 i 50. Raspon slučajnih koeficijenata u vektoru ***b*** jest 1 do 5000. U realnim problemima uobičajeno je velik broj koeficijenata u matrici **A** jednak 0. Zbog toga se ona generira tako da 20% koeficijenata ima slučajnu vrijednost od 1 do 50 dok su ostali 0.

Druga vrsta su bili problemi ruksaka:

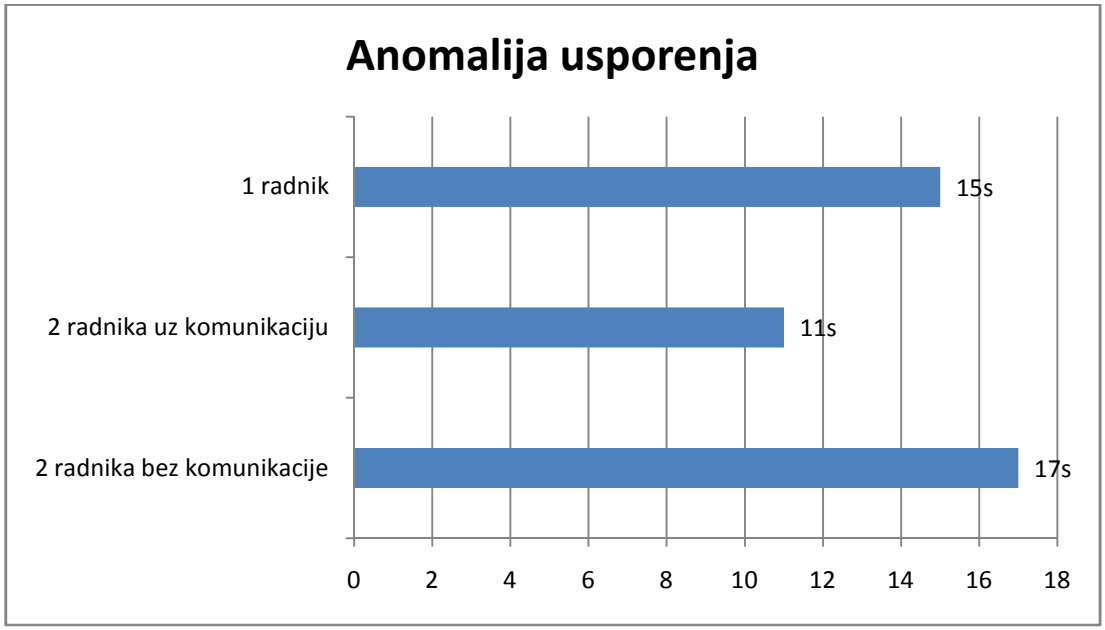
Imamo N stvari i ruksak kapaciteta W. Svaka stvar ima svoju težinu w_i i cijenu c_i potrebno je pronaći podskup stvari koji stane u ruksak a pritom ima najveću moguću ukupnu cijenu.

Ovaj problem je pogodan jer se jednostavno može se prikazati u obliku binarnog programa. Težine stvari biraju se na slučajan način između 1 i 200. Ukupna težina ruksaka odabrana je na slučajan način između 1/3 i 2/3 ukupne težine svih stvari.

7.2 Anomalija usporenja

U ovom pokusu cilj je ispitati može li se i kako u ovom programu dogoditi anomalija usporenja. Za demonstraciju koristiti ćemo problem u datoteci test250.txt u kojoj se nalazi problem ruksaka sa 250 stvari formuliran u obliku binarnog programa.

Napomenimo da je glavni razlog anomalija usporenja u ovom programu obrada velikog broja čvorova koje slijedni algoritam uopće ne obrađuje. To se donekle sprječava raspodjelom bolje granice svim radnicima čim ju jedan od njih nađe. Da bismo pokazali važnost ove komunikacije među radnicima probati ćemo je ugasiti i vidjeti kako to utječe na brzinu. Rezultati mjerenja vremena izvođenja nalaze se na slici:



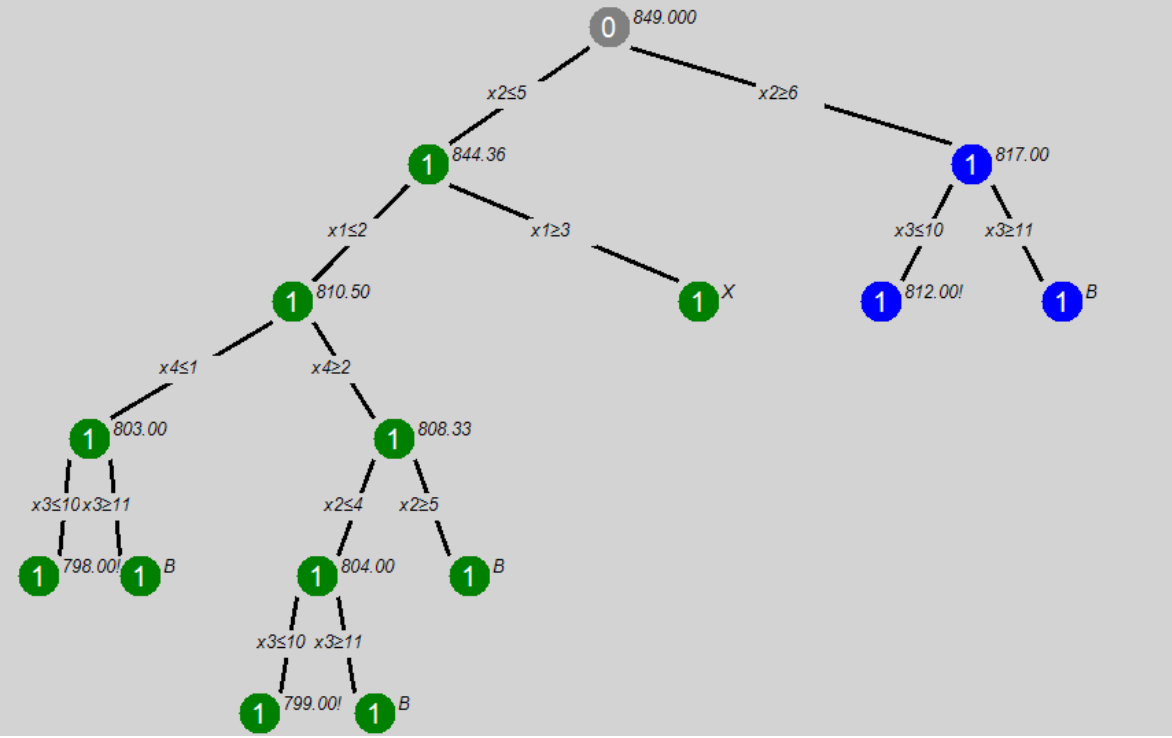
Slika 7.1., Ilustracija anomalije usporenja

Vidimo da je algoritam koji koristi dva radnika nešto brži (11s) od onog koji koristi samo jednog (15s). Ubrzanje je manje od očekivanog pa tu možemo vidjeti i anomaliju kvarenja. U slučaju da algoritam koristi dva radnika ali bez komunikacije vrijeme izvođenja je jako loše (17s). To je sporije čak od slučaja s jednim radnikom pa je nastupila anomalija usporenja.

7.3 Anomalija prevelikog ubrzanja

U ovom pokusu cilj je demonstrirati kako se može dogoditi anomalija prevelikog ubrzanja. Koristiti ćemo problem iz datoteke AnomAccel_IP5.txt. U njoj se nalazi IP problem sa 5 varijabli i 6 ograničenja.

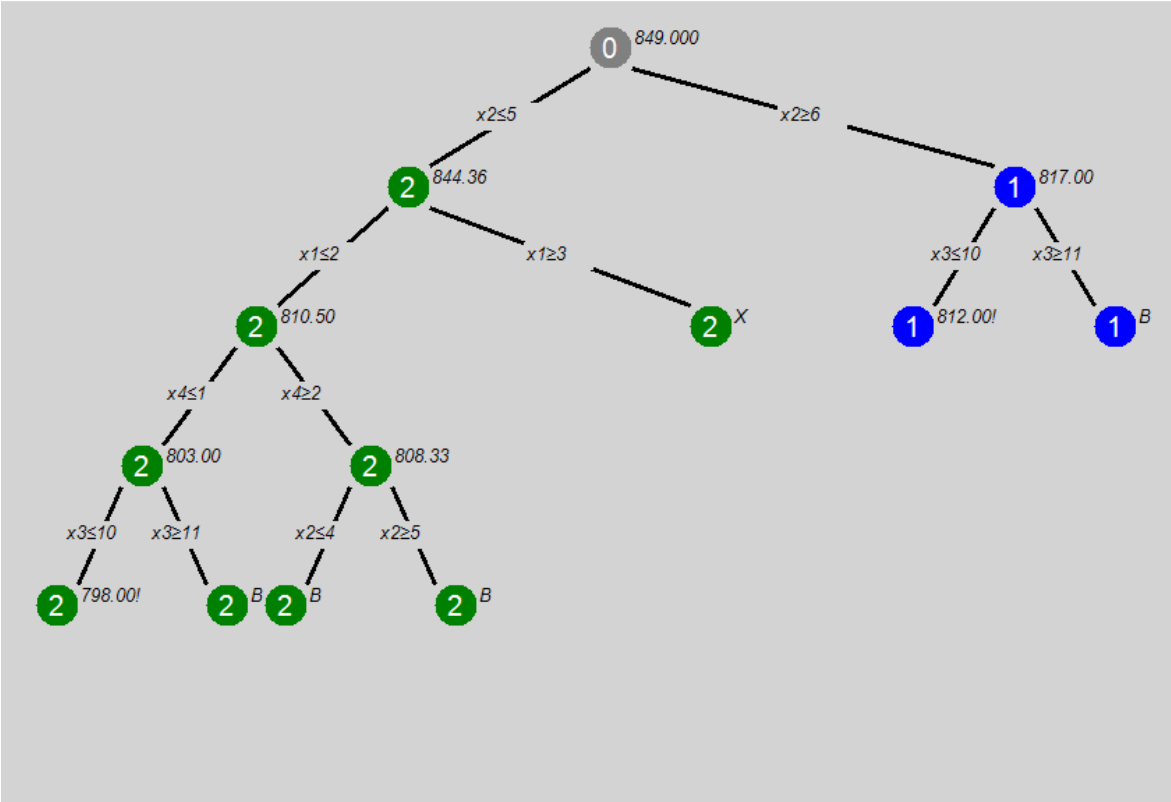
Prvo ćemo pokrenuti algoritam sa samo jednim radnikom. Time ćemo dobiti tijek jednak slijednom algoritmu, što je prikazano na slici.



Slika 7.2. Tijek optimizacije za problem AnomAccel_IP7.txt sa 1 radnikom (tijek slijednog algoritma)

Vidimo da je i zeleni i plavi zadatak obavljao radnik 1. Možemo zaključiti da je algoritam istražio cijeli zeleni zadatak u kojem je našao rješenje s iznosom funkcije cilja od 799. Nakon toga potraga u plavom zadatku je bila veoma kratka i našla je još bolje rješenje koje daje vrijednost funkcije cilja 812.

Bez promjene ostalih parametara pokrenimo sada program koristeći dva radnika. Tijek izvođenja koji tako nastaje prikazan je na slici.



Slika 7.3. Tijek optimizacije za problem AnomAccel_IP7.txt sa 1 radnikom (tijek paralelnog algoritma)

Tijek algoritma je jako sličan slijednom slučaju. Ovaj put radnik 1 obrađuje plavi zadatak dok radnik 2 istovremeno obrađuje zeleni. Ipak može se primijetiti jedna razlika. Određeni čvorovi u zelenom zadatku koji u slijednom slučaju nisu bili podrezani sada to jesu. Radi se o čvoru koji ima vrijednost funkcije cilja 804 na slici y.

Razlog zašto slijedni algoritam nije podrezao taj čvor je to što u trenutku njegove obrade još nije znao da postoji granica 812. U paralelnom slučaju radnik 1 je pronašao tu granicu i uspio ju dojaviti radniku 2 na vrijeme. Zbog toga, podrezivanje se moglo obaviti.

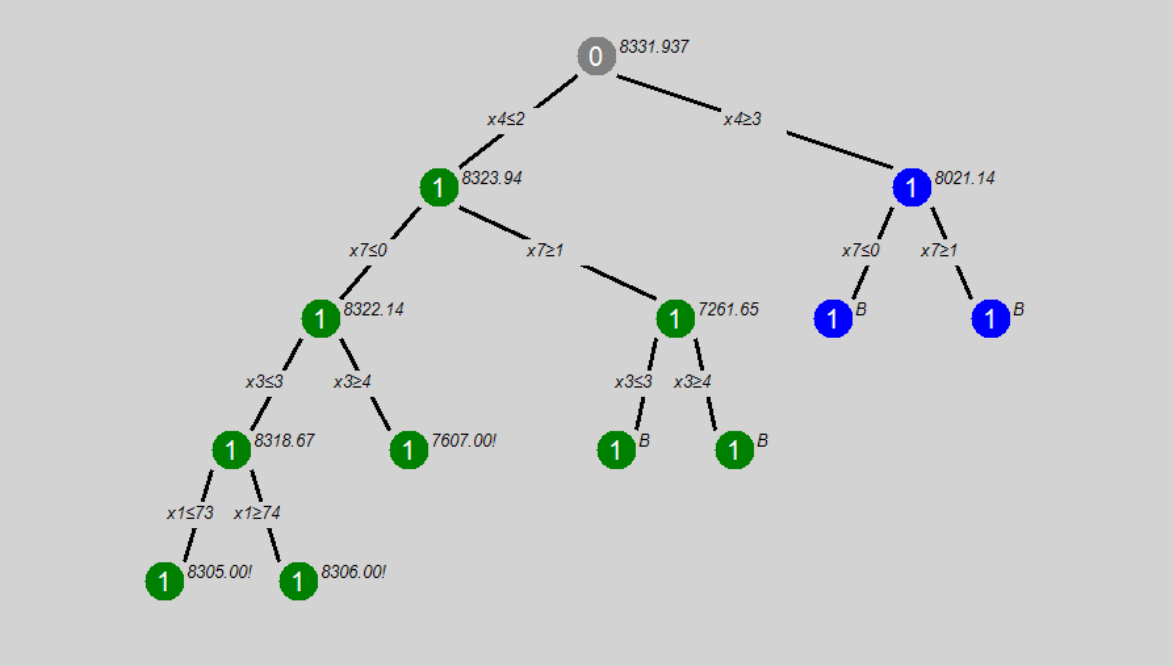
U ovom slučaju odrezan je samo jedan čvor no u praksi moglo je to biti i veliko podstablo sa puno razina čije ranije podrezivanje bi značajno ubrzalo postupak.

U ovakvim slučajevima paralelni algoritam ne samo da obavlja posao brže zbog više radnika nego i obavlja ukupno manje posla. Kao posljedica toga može se dogoditi da ubrzanje bude veće od očekivanog.

7.4 Anomalija kvarenja

U ovom pokusu cilj je demonstrirati kako se može dogoditi anomalija kvarenja. Ulazna datoteka koju ćemo koristiti je AnomDetr_IP7.txt. U njoj se nalazi IP problem sa 7 varijabli i 7 ograničenja.

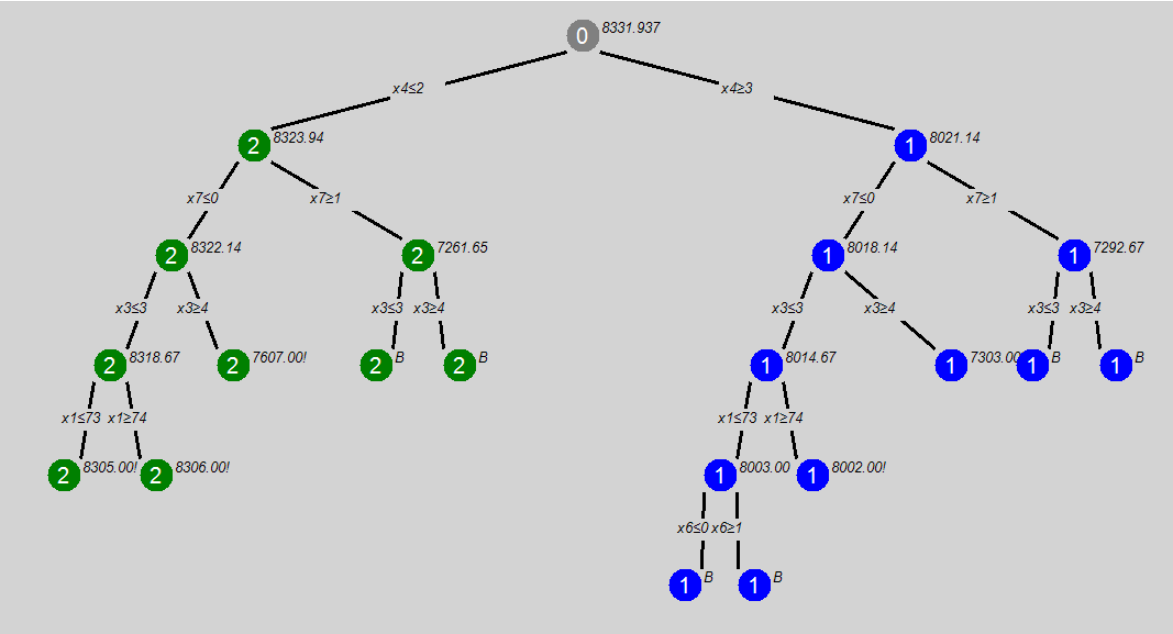
Prvo ćemo pokrenuti program sa jednim radnikom što će imati isti tijek izvođenja kao i slijedni algoritam. Tijek algoritma prikazan je na slici:



Slika 7.4. Tijek optimizacije za problem AnomDetr_IP7.txt sa 1 radnikom (tijek slijednog algoritma)

Možemo vidjeti da je sve zadatke obradio radnik 1. Prvo je obradio zeleni zadatak u kojem je našao rješenje 8306. To rješenje je i globalni optimum no da bismo to dokazali potrebno je obraditi i plavi zadatak. U obradi plavog zadatka podrezivanjem je ustanovljeno da tamo nećemo naći bolje rješenje i time postupak završava.

Sada kada znamo kako se algoritam ponaša sa jednim radnikom pokrenuti ćemo ga u istim uvjetima ali koristeći dva radnika. Dobiven tijek programa prikazan je na slici:



Slika 7.5. Tijek optimizacije za problem AnomDetr_IP7.txt sa 2 radnika

U ovom slučaju radnik 2 je obrađivao zeleni zadatak dok je radnik 1 paralelno obrađivao plavi zadatak. Rezultati su na prvi pogled zbunjujući, plavi zadatak se sada sastoji od značajno više čvorova nego prije. Da bi objasnili ovakvo čudno ponašanje pogledajmo čvorove u plavom zadatku sa vrijednostima funkcije cilja 8018.14 i 7292.67. Oni su u slijednom algoritmu bili podrezani, no u paralelnom nisu. Uzrok toga jest činjenica da slijedni algoritam u trenutku ispitivanja tih čvorova već ima gornju granicu 8306. U paralelnoj obradi to nije slučaj. Naime, u trenutku kada radnik 1 obrađuje spomenute čvorove u plavom zadatku on ne zna za tu granicu jer ju radnik 2 u zelenom zadatku još nije pronašao.

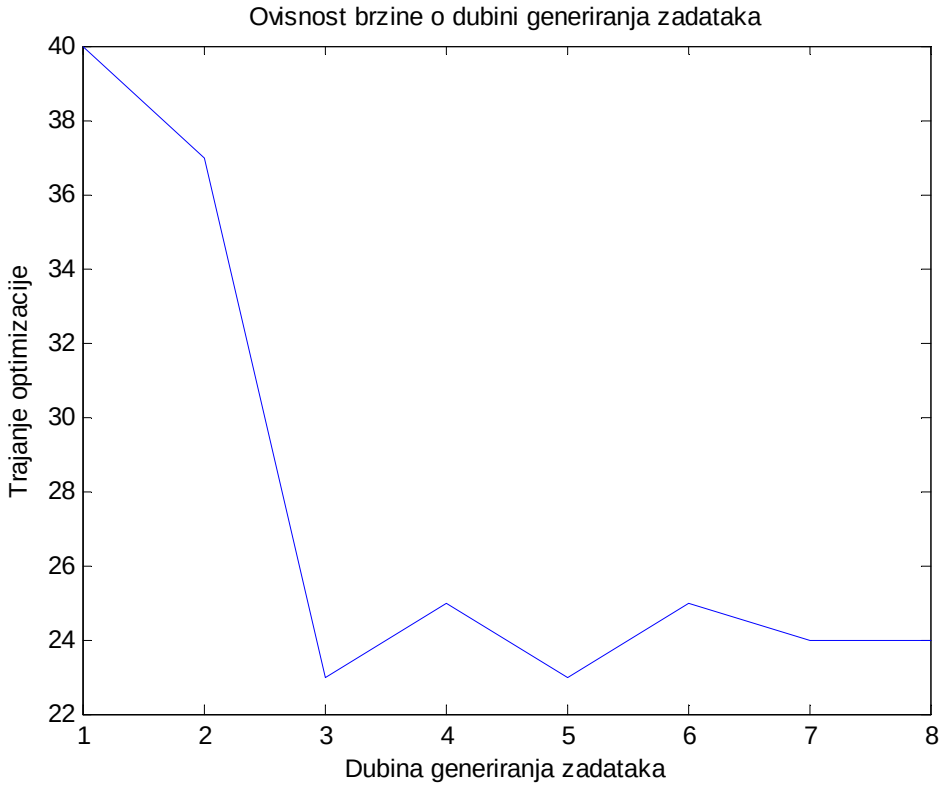
Naravno radnik 2 će podijeliti granicu sa radnikom 1 čim ju nađe, no u međuvremenu radnik 1 će trošiti vrijeme uzalud obrađujući čvorove koje bi slijedni algoritam zanemario podrezivanjem. Zbog toga, postignuto ubrzanje biti će manje od očekivanog.

7.5 Ujednačavanje opterećenja

U ovom pokusu cilj je proučiti na koji način dubina na kojoj se generiraju zadaci utječe na brzinu izvođenja programa. Da bismo to postigli program je pokrenut više puta sa različitim parametrom dubine generiranja zadataka i mjereno je trajanje izvođenja. Ispitivanje je obavljeno na problemu IP30.txt uz korištenje 2 radnika. Tako dobiveni podaci prikazani su tablicom i grafom:

Tablica 7.1. Vremena izvođenja programa dobivena za različite parametre dubine generiranja zadataka

Dubina	1	2	3	4	5	6	7	8
Trajanje	40	37	23	25	23	25	24	24



Slika 1.6. Prikaz ovisnosti trajanja optimizacije o dubini generiranja zadataka

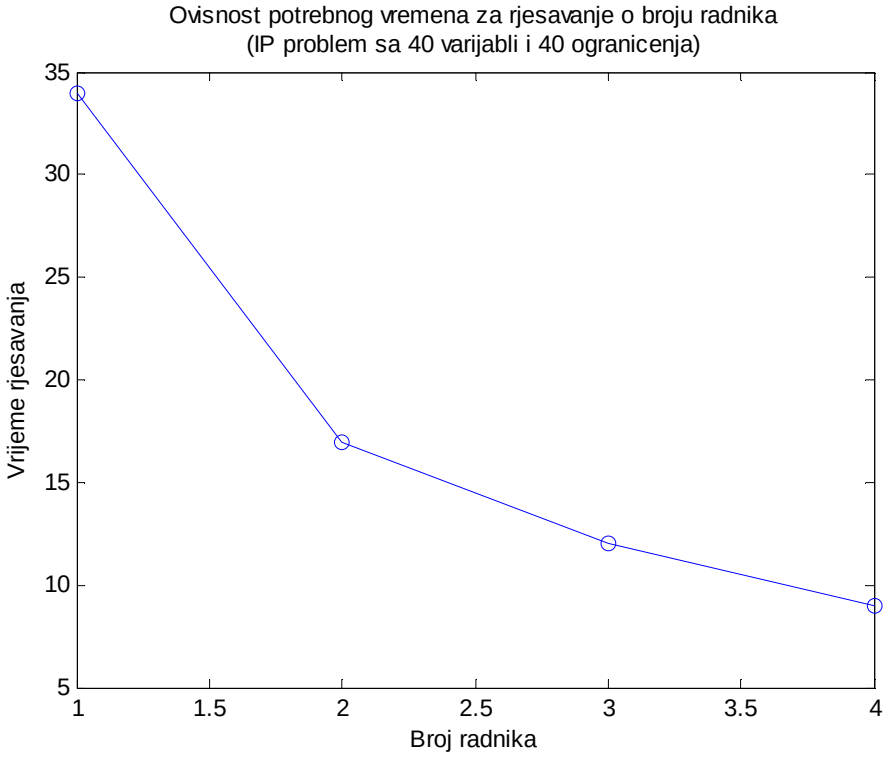
Prisjetimo se da povećanjem dubine na kojoj se generiraju zadaci povećavamo ukupan broj zadataka. Veći broj zadataka unosi ujednačeniju raspodjelu posla ali također i povećava komunikacijske troškove. Iz grafa vidimo da je povećanje dubine dovelo do veće brzine samo do dubine 3. Od te dubine na

dalje povećanje dubine nije imalo znatnog utjecaja na brzinu. Možemo zaključiti da je na dubinama ispod 3 opterećenje neravnomjerno raspoređeno. Na većim dubinama taj problem nestaje.

Također očekivali bismo da ćemo znatnim povećanjem broja zadataka imati lagano smanjenu brzinu zbog većih komunikacijskih troškova. To se nije dogodilo. Razlog tome je što se mnogo više vremena troši na obradu zadataka nego na komunikaciju. Zbog toga je utjecaj dodatne komunikacije jako mali.

7.6 Ubrzanje s obzirom na broj radnika.

Ubrzanje programa dosta varira i ovisi o instanci problema. Na slici 2. je primjer vremena optimizacije dobivenih za jedan od ispitnih problema (za koji ispadaju dosta dobri rezultati) na procesoru Intel i7-920 sa 4 jezgre.



Slika 7.7. Prikaz ovisnosti trajanja optimizacije o broju radnika

7. Zaključak

Iznijeli smo osnovne ideje algoritama za linearno i mješovito cjelobrojno programiranje. Možemo zaključiti da je mješovito cjelobrojno programiranje dosta složeniji postupak od običnog linearnog programiranja pa je za njegovu uspješnu primjenu potrebno više znanja i pažnje.

Za paralelizaciju algoritma grananja i ograđivanja možemo reći da je složenija nego što se na prvi pogled čini. Razlog tome je što količina posla koju obavlja paralelni algoritam nije jednaka količini posla koju obavlja slijedni algoritam. To uzrokuje anomalije na koje treba paziti pri oblikovanju paralelnog algoritma.

Za razvijeni program možemo zaključiti da je dovoljan za pokazati neka od svojstava paralelnog algoritma grananja i ograđivanja. Algoritam funkcionira ispravno no anomalije se mogu pojaviti. Kao jedno moguće poboljšanje možemo navesti uporabu naprednijih algoritama (revidirana simpleksna metoda ili algoritam grananja i rezanja). Još jedno moguće poboljšanje je napredniji način raspodjele posla radnicima koji bi osigurao da u svakom trenutku svaki radnik radi na nekom obećavajućem podproblemu.

8. Literatura

Kalpić D., Mornar V., Operacijska istraživanja, Zagreb 1996.

Hillier F.S. , Lieberman G.J. , Introduction to Operations Research, Seventh Edition, New York, McGraw-Hill, 2001

El-Ghazali Talbi, Parallel Combinatorial Optimization , John Wiley & Sons, Inc., Hobokoken, New Jersey, 2006

Harry W.J.M. Trienekens , *Parallel Branch and Bound and Anomalies (1989)* http://publishing.eur.nl/ir/repub/asset/1509/eur-few-cs-89-01.pdf, 24.11.2010.

Mathematics and Computer Science Division Argonne National Laboratory, MPICH2 User's Guide, http://www.mcs.anl.gov/research/projects/mpich2/documentation/files/mpich2-1.2.1-userguide.pdf , 13.1.2011

9. Sažetak

Mogućnosti korištenja paralelnih algoritama u linearnom optimiranju.

U radu su opisani neki od postupaka za linearno programiranje i mješovito-cjelobrojno programiranje. Posebna pažnja posvećena je algoritmu grananja i ograđivanja za rješavanje MIP problema. Razmatrani su različiti pristupi paralelizaciji tog algoritma. Istraženi su i problemi koji se mogu pojaviti pri paralelizaciji u obliku anomalija. Na kraju opisana je implementacija paralelnog programa za rješavanje MIP problema.

Ključne riječi: Paralelni algoritmi, Optimizacija, Cjelobrojno programiranje

Possibilities of using parallel algorithms in linear optimization.

In this work some of the methods for linear programming and mixed-integer programming are described. The branch and bound algorithm for solving MIP problems is given special focus. Different approaches to parallelization of this algorithm are studied. Problems which can arise in this parallel algorithm in the form of anomalies are also explored. Finally a description of an implemented parallel branch & bound algorithm is given.

Keywords: Parallel algorithms, Optimization, Integer programming