SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINAR

Primjena genetskog algoritma na rješavanje problema optimalne kupovine proizvoda i usluga

Leo Osvald

Voditelj: *Doc. dr. sc. Domagoj Jakobović*

Zagreb, svibanj, 2010.

Sadržaj

1 UVOD

U svakodnevnom životu ljudi se često nalaze u situaciji gdje pametnim ponašanjem mogu uštedjeti novac ili vrijeme. Jedna takva situacija je npr. odlazak u trgovinu; kupac obično ima neke svoje kupovne želje koje želi ispuniti i on skoro uvijek nastoji proći što jeftinije. Tako primjerice, bira u koje će trgovine otići, koje proizvode će nabaviti, te u kojoj količini. Činjenica je da zbog konkurencije, svaki prodavač nekim proizvodima spušta cijene dok istovremeno, radi ravnoteže, drugim proizvodima povećava cijene. Kako bi privukao što više kupaca, često uvodi različite akcije, poput popusta na iznos veći od 1000 kn, promotivne ponude tipa "3+1 gratis", popuste na količinu, posebne veleprodajne cijene i sl.

Kupac dakle može pametnim odabirom minimizirati svoj trošak, a da pritom ispuni svoje kupovne želje.

1.1 OPIS PROBLEMA

Na nekom tržištu postoji jedan ili više prodavača odnosno pružatelja usluga (u daljnjem tekstu prodavača). Svaki prodavač ima točno određene proizvode (ili usluge) koje prodaje, pri čemu nekim proizvodima raspolaže u konačnim količinama, a nekih ima beskonačno mnogo.

Za svakog prodavača na tržištu unaprijed je poznat popis svih ponuda koje taj prodavač nudi. Popis ponuda je nepromjenjiv (invarijantan) u vremenskom periodu u kojem promatramo tržište, kao i same ponude.

Svaka ponuda ima sljedeće značajke:

- identifikator – jednoznačno određuje ponudu
- prodavač – ponuđač
- cijena ponude – netto cijena koju kupac plaća ako prihvati ovu ponudu točno jednom, bez uključenih dodatnih troškova (kao što su troškovi eventualnog transporta, narudžbe ili pak neki popusti ili drugi modifikatori)
- popis proizvoda koji su uključeni u ponudu, pri čemu se isti proizvod može pojaviti više puta
- popis ponuda koje moraju biti prihvaćene da bi ova ponuda mogla biti prihvaćena. Drugim riječima, popis preduvjeta koje kupac mora ispuniti kako bi mu bila omogućena dotična ponuda. U tom popisu, brojnost pojedinih ponuda (tj. preduvjeta) može biti veća od jedan (npr. može se tražiti da kupac mora prihvatiti neku ponudu više puta kako bi stekao pravo na ovu ponudu).

Opcionalne značajke:

- veleprodajna cijena po komadu – općenito, diskretna funkcija koja određuje netto cijenu po komadu za određeni broj prihvaćenih ponuda, pri čemu je taj broj veći od jedan. Ako je definirana veleprodajna cijena, kupac može iskoristiti eventualni popust u odnosu na maloprodajnu cijenu (ako želi).
- skup ostalih svojstava koji uglavnom služe kao opis ponude
- broj ponuda do isteka zaliha – dostupna količina

Također, svaka ponuda mora ispunjavati sljedeća dva zahtjeva:

1) Cijena ponude je nenegativni konačan broj

2) Barem jedan proizvod koji je uključen u ponudu je količinski ograničen ili je sama ponuda količinski ograničena.

Izostanak bilo kojeg od dva gornja zahtjeva može dovesti do situacije u kojem optimalno rješenje uključuje kupovinu beskonačne količine proizvoda, beskonačnu cijenu (pozitivnu ili negativnu) ili oboje.

Nadalje, postoji točno jedan kupac koji je savršeno informiran i zna sve ponude svih prodavača. Taj kupac ima popis zahtjeva na izbor proizvoda koji se moraju ispoštovati.

Ti zahtjevi definiraju ograničenja na proizvode koje kupac mora kupiti (u određenim količinama) kako bi zadovoljio zahtjev.

Postoje dvije vrste zahtjeva na proizvode:

a) Disjunktni (međusobno isključivi)

Svaki disjunktan zahtjev vrijedi isključivo na jednom proizvodu koji ispunjava taj zahtjev. Drugim riječima, jedan kupljeni proizvod može ispuniti najviše jedan zahtjev ove vrste. Npr. ako se zahtjeva da se kupi neki sok, te neki proizvod s okusom naranče, ukoliko se kupi sok od naranče time ne mogu biti ispunjena oba zahtjeva već točno jedan (da bi i drugi bio ispunjen, potrebno je kupiti još barem jedan proizvod).

b) Ne-disjunktni

Ova vrsta zahtjeva ne isključuje mogućnost da se drugi zahtjevi primjene na isti kupljeni proizvod. Npr. zahtjeva se da kupimo neki sok, te neki proizvod s okusom naranče; ako kupnjom soka od naranče ispunjena su oba uvjeta istovremeno (čak i da se zahtjev za sokom pojavljuje 100 puta, dovoljno bi bilo kupiti jedan jedini proizvod – sok od naranče).

Dakle, jedan komad nekog proizvoda može zadovoljavati najviše jedan disjunktan zahtjev i proizvoljno mnogo ne-disjunktnih zahtjeva (ili niti jedan).

Cilj kupca jest prihvatiti neke ponude tako da kupi barem sve proizvode koje diktiraju zahtjevi , na način da minimizira ukupan trošak (tj. prođe najjeftinije moguće).

Osim zahtjeva vezanih uz proizvode, kupac teoretski može imati i neke druge zahtjeve poput zahtjeva da kupuje samo od određenih prodavača, da ne pristaje na ponude veće od određenog iznosa, ili pak da ne želi ni pod koju cijenu kupiti neke proizvode i sl.. Međutim takvi zahtjevi će se ovdje zanemariti iz jednostavnog razloga: uvijek je moguće pročistiti tržište tj. promatrati novo tržište na kojem su prisutne samo kupcu zanimljive ponude, proizvodi ili pak prodavači.

1.2 DEFINICIJA

Na tržištu M postoji skup S pružatelja usluga odnosno prodavača (u daljnjem tekstu prodavača):

$$S = \{ s_0, s_1, \ldots, s_n \} \in M$$

Svaki prodavač s_i nudi skup ponuda O_i. Ponude možemo opisati skupom uređenih parova (P_j, Q_j) koji opisuju količine pojedinih proizvoda; P_j predstavlja neki proizvod, a Q_j brojnost s kojom se pojavljuje u toj ponudi (tj. količinu).

$$O_i = (P, Q)$$
$$Q : P \times Q \rightarrow \{ \mathbb{N} \cup 0 \}$$

Skup svih uređenih parova (P_j, Q_j) nazivamo *konfiguracijom ponuda*.

Konfiguracija ponuda je, dakle, multiskup ponuda koji nužno ne zadovoljava postavljena ograničenja.

U daljnjem tekstu će se radi bolje jasnoće umjesto pojma "multiskup ponuda" koristiti pojam konfiguracije rješenja, ili samo "konfiguracija".

Konfiguracija ponuda je valjana ako i samo ako se sva disjunktna ograničenja mogu bijektivno pridružiti proizvodima (u odgovarajućim količinama), i za sva nedisjunktna ograničenja postoji neki proizvod koji ga zadovoljava (koji nije u bijektivnoj korespondenciji).

Cilj promatranog problema kupovine jest pronaći valjanu konfiguraciju ponuda čija je ukupna cijena minimalna.

2 ALGORITAMSKO RJEŠAVANJE PROBLEMA OPTIMIZACIJE KUPOVINE

Mnogi algoritmi rješavanja problema optimizacije kupovine rade na način da pretražuju prostor rješenja (engl. *search space*) tj. grade multiskup[1] ponuda koje se trenutno razmatraju kao moguće rješenje. Dakle, oni će tražiti moguća rješenja na način da isprobavaju razne multiskupove ponuda (koje bi kupac kao prihvatio) s ciljem nalaženja optimalnog rješenja koje mora zadovoljavati ograničenja koja je kupac postavio. Znači, jedan od potproblema s kojim ćemo se često susretati prilikom rješavanja ovog problema kupovine jest sljedeći:

Kako u razumnom vremenu provjeriti da li neki multiskup ponuda zadovoljava ograničenja koje je kupac postavio putem njegovih zahtjeva?

Kasnije ćemo vidjeti da se u kontekstu genetskog algoritma radi o *valjanom* rješenju (engl. *feasible*), što u našem slučaju znači rješenje koje zadovoljava ograničenja (tj. kupčeve zahtjeve), ili pak *nevaljanom* rješenju (engl. *infeasible*).

2.1 PROBLEM PROVJERE VALJANOSTI MOGUĆEG RJEŠENJA

2.1.1 Redukcija na problem maksimalnog sparivanja

Za početak, bez smanjenja općenitosti, pretpostavimo da svaka ponuda uključuje točno jedan proizvod, te da ne postoje dvije različite ponude koje uključuju isti proizvod. (p1)

Također, pretpostavimo da je korisnik definirao N različitih zahtjeva sljedećeg oblika:

1x proizvod koji u svom opisu sadrži riječ $\boldsymbol{X_i}$ $(1 \leq i \leq N)$ (p2)

gdje je X_i proizvoljna smislena riječ koja se sastoji od brojki, slova i/ili znakova (npr. "sok", "naranča", "1", "100 g" i sl.).

Također, neka su svi ti zahtjevi međusobno disjunktni (tip *a*).

Pokušajmo odgovoriti na sljedeće pitanje: Ako je kupac prihvatio neki skup ponuda, da li taj skup udovoljava zahtjevima korisnika tj. da li on zadovoljava određena ograničenja (koja su određena kupčevim zahtjevima)?

Da bi odgovorili na ovo pitanje, pojednostavimo najprije ovaj potproblem tako da razmatramo samo proizvode: svaku ponudu zamijenimo s točno jednim[2] proizvodom. Konačno, sad imamo dva skupa:

1 Multiskup je neka vrsta skupa u kojem poredak elemenata nije bitan, ali je posebno važna brojnost svakog elemenata[1]. Drugim rječima, isti element u multiskupa se može pojaviti više puta.

2 Ovo vrijedi zbog prije spomenute pretpostavke *p1*. U općenitom slučaju, svaku ponudu bismo zamijenili s multiskupom proizvoda u kojem se svaki proizvod pojavljuje onoliko puta koliko se nalazi u ponudi.

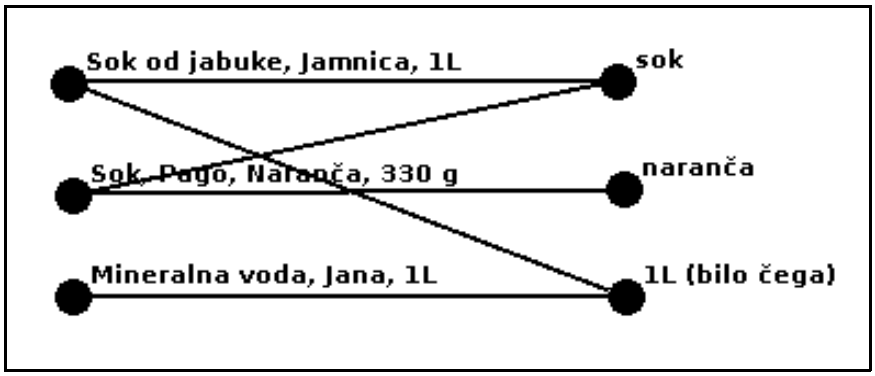
skup proizvoda, te skup ograničenja.

Primijetimo da je ovo u biti problem maksimalnog sparivanja (engl. *maximum matching problem*); svakom ograničenju moramo pridružiti točno jedan proizvod, a isti proizvod ne može biti pridružen više od jednom ograničenju (jer su ograničenja disjunktna). Cilj je spariti što je moguće više proizvoda i ograničenja, a pošto sva ograničenja moraju biti zadovoljena, potrebno je naći sparivanje koje će spariti sva ograničenja s nekim proizvodima. Budući da je broj ograničenja neka konstanta *C*, dovoljno je provjeriti da li je maksimalno sparivanje veće ili jednako tom broju, i ako je, to znači da trenutno razmatrana konfiguracija ponuda predstavlja *valjano* rješenje.

Problem sparivanja može se ilustrativno prikazati na način da se izgradi bipartitan graf u kojemu jednom skupu pripadaju ograničenja, a drugom proizvodi, gdje su svi proizvodi koji zadovoljavaju neko ograničenje povezani bridom s tim ograničenjem.

2.1.1.1 Primjer:

Ograničenja koja proizlaze iz kupčevih zahtjeva:



Slika 1: Primjer sparivanja proizvoda i ograničenja

- sok – kupac zahtjeva proizvod koji u opisu sadrži riječ "sok"
- naranča – kupac zahtjeva proizvod koji u nazivu sadrži riječ "naranča"
- 1L – kupac zahtjeva proizvod koji u nazivu sadrži riječ "1L"

Jedini način da se udovolji kupčevim zahtjevima jest da se naprave sljedeća sparivanja proizvoda (lijevo) i ograničenja (desno):

Sok od jabuke, Jamnica, 1L ↔ sok

Sok, Pago, Naranča, 330 g ↔ naranča

Mineralna voda, Jana, 1 L ↔ 1L

Dakle, zaključili smo da ako nam je poznat skup ponuda koji zadovoljava pretpostavku p1, te ako imamo zahtjeve (koji definiraju ograničenja) koji zadovoljavaju pretpostavku p2, da problem provjere da li skup prihvaćenih ponuda zadovoljava kupčeve zahtjeve možemo riješiti tako da ga svedemo na *problem sparivanja* (vidi primjer gore). Međutim, postavlja se pitanje što ako se udaljimo od gornje dvije pretpostavke?

Ispostavlja se da se i tada ovaj problem može svesti na problem sparivanja.

Sad ćemo to i pokazati:

Najprije ćemo razmotriti što se događa ako ponude koje je kupac prihvatio sadrže više istih proizvoda. U tom slučaju, bipartitni graf sadržavat će za svaki proizvod onoliko čvorova koliko se puta taj proizvod javlja u konfiguraciji ponuda. Prilikom sparivanja, uočimo da je svejedno s kojim se od identičnih proizvoda sparuje neka ponuda.

Slično, ako imamo više istih ponuda, uzet ćemo sve proizvode iz svih ponuda i tako dobiti

multiskup proizvoda, pa problem rješavamo na gore spomenut način.

Što se pak zahtjeva tj. ograničenja tiče, imamo nekoliko novina:

- može postojati više istih zahtjeva (tipa 10x "sok", 5x "naranča" i sl.)
- zahtjev ne mora biti disjunktan (tip *b*) – ovo ne stvara nikakav problem. Naime, ako zahtjev nije disjunktan (tip *b*) situacija se pojednostavljuje jer taj zahtjev možemo ispuniti tako da uzmemo bilo koji proizvod koji zadovoljava taj zahtjev (bez da pritom onemogućimo nekom drugom zahtjevu da bude ispunjen na istom proizvodu)

Valja primijetiti jedan veliki nedostatak ovog pristupa. Naime, kupčevi zahtjevi često će sadržavati više istih ograničenja (npr. kupac želi kupiti 10 kutija mlijeka), a apsurdno je očekivati da će konfiguracija ponuda uvijek sadržavati svaki proizvod u količini 1. Dakle, u klasičnom scenariju postojat će puno istih proizvoda i ograničenja, a samim time i puno čvorova koji predstavljaju iste proizvode i ograničenja u bipartitnom grafu. U najboljem slučaju, koristeći *Hopcroft-Karpov* algoritam, ovaj problem se može riješiti u vremenskoj složenosti $O(\sqrt{V}\cdot E)$ gdje je V broj čvorova u grafu, a E broj bridova. Ako pretpostavimo da je prosječan broj istih proizvoda i ograničenja (gledajući zajedno) jednak A, imamo $A\cdot V$ vrhova, što znači da možemo imati čak A^2V^2 bridova (u slučaju da svaka proizvod zadovoljava svaki zahtjev), pa time dobivamo vremensku složenost od $O(\sqrt{A\cdot V}\cdot A^2\cdot V^2)=O(A^{2.5}\cdot V^{2.5})$. Ono što je zabrinjavajuće jest član $A^{2.5}$ koji govori da vremenska složenost raste brže od kvadratno s porastom broja "duplih" proizvoda i ograničenja. Drugim riječima, korisnikovi zahtjevi poput npr. "1000 čokolada; 10000 listova papira; 50000 bilo kakvih proizvoda jeftinijih od 5 kn" mogli bi uzrokovati da se algoritam izvodi neprimjereno dugo.

2.1.2 Redukcija na problem najvećeg protoka (*Maximum flow problem*)

Gornje opisanu neefikasnost ipak je moguće izbjeći; modificirat ćemo algoritam tako da on ne ovisi o količini istih proizvoda ili ograničenja. To ćemo učiniti tako što ćemo problem reducirati na problem maksimalnog protoka u grafu (engl. *Maximum flow problem*). To ćemo napraviti u nekoliko koraka:

1) Za svaki različiti proizvod p_i ($1 \le i \le P$) stvaramo po jedan čvor s kapacitetom koji je jednak ukupnom broju pojavljivanja tog proizvoda u našim ponudama, $Q(p_i)$. Također, za svako različito ograničenje c_i ($1 \le i \le C$) stvaramo po jedan čvor s kapacitetom koji je jednak ukupnom broju pojavljivanja tog ograničenja, $Q(c_i)$.
2) Za svaki par proizvoda (pi, cj) provjeravamo da li proizvod p_i zadovoljava ograničenje c_i; ako da, napravimo usmjereni brid ($p_i \rightarrow c_j$) s kapacitetom $Q(c_j)$, gdje je $Q(c_j)$ broj pojavljivanja ograničenja p_j.
3) U graf G' dodamo dva umjetna čvora, *izvor* i *ponor*. Za svaki čvor c_i koji predstavlja proizvod dodat ćemo usmjereni brid (*izvor* $\rightarrow c_i$) koji iz izvora vodi u taj čvor i čiji je kapacitet $Q(c_i)$, a za svaki čvor p_i koji predstavlja ograničenje dodat ćemo usmjereni brid ($p_i \rightarrow$ *ponor*) koji iz tog čvora vodi u ponor i čiji je kapacitet $Q(p_i)$. Pri tome $Q(c_i)$ i $Q(p_i)$ odgovaraju ukupnom broju pojavljivanja proizvoda c_i, odnosno ograničenja p_i.

2.1.2.1 Primjer:

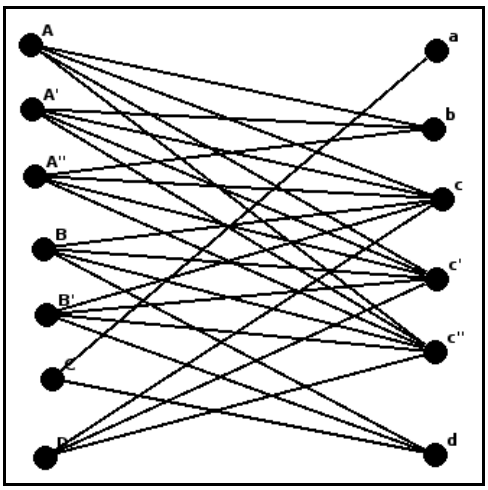
Razmatramo skup proizvoda {A, A', A'', B, B', C, D} i skup ograničenja {a, b, c', c', c'', d}.

Proizvodi A, A, A' i A'' su identični, isto kao i B i B', kao i ograničenja c, c' i c'', pa bismo tako prethodne skupove mogli zamijeniti multiskupovima {A, A, A, B, B, C, D} i {a, b, c, c, c, d}.

Ustanovili smo da su skupovi proizvoda koji redom zadovoljavaju ograničenja sljedeći:

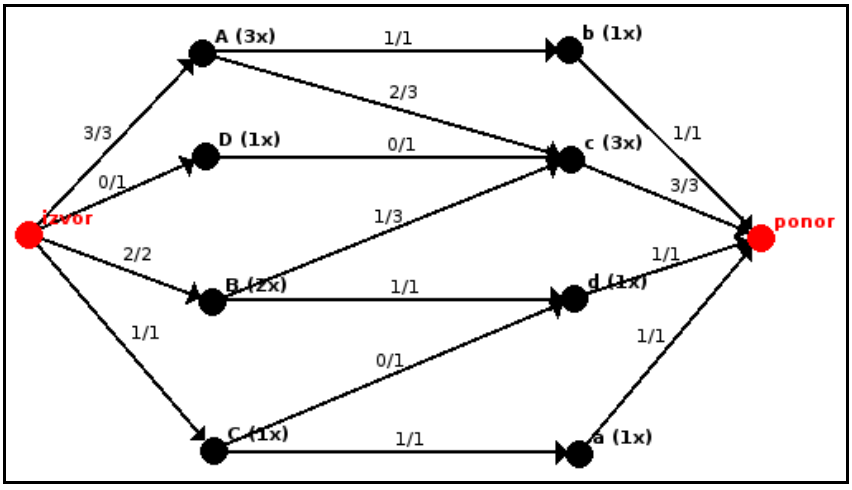
S(a) = {C}, S(b) = {A}, S(c) = {A, B, D}, S(d) = {A} (proizvodi A', A'' i B', te ponude c' i c'' izostavljene su radi jasnoće).

Najprije izgradimo bipartitan graf na način koji je objašnjen u prethodnom odjeljku:



Slika 2: Bipartitan raf iz primjera

Slika 2. jasno upućuje na nedostatak efikasnosti prijašnjeg pristupa čak i u ovom malenom primjeru od svega nekoliko proizvoda. Možemo primijetiti da postoji 13 čvorova i 40 bridova.



Slika 3: Graf protoka dobiven iz prošlog grafa

Pogledajmo sada kako izgleda naš novi pristup:

Na slici 3. vidimo graf koji je dobiven primjenom gore opisanog postupka.

Brojevi u zagradama iza oznaka čvorova označavaju količinu proizvoda predstavljenog pojedinim čvorom. Bridovi su predstavljeni strelicama koje su usmjerene kao i brid, a na njima se nalaze dva cijela broja (*c*/*f*) odvojena znakom '/'. Broj *c* označava kapacitet pojedinog brida, a broj broj *f* predstavlja protok koji je ostvaren u slučaju maksimalnog protoka mreže.

Kako bismo lakše shvatili o čemu se radi, ovaj graf možemo promatrati kao cjevovodnu mrežu; bridovi predstavljaju jednosmjerne cijevi s ventilom koji propušta vodu samo u smjeru strelice, a čvorovi su bezdimenzionalna spojišta (s beskonačnim kapacitetom). Kroz cijevi pritom smije teći samo jedinična količina vode (uključujući i nulu). Problem je sljedeći: ako *izvor* pumpa beskonačnu količinu vode, postavlja se pitanje koliko će vode stići u *ponor*.

Ovaj problem može se riješiti primjenom nekoliko algoritama, a mi ćemo pretpostaviti da rješavamo jednim od najefikasnijih, a to je tzv. *Push-relabel maximum flow algorithm* implementiran s *FIFO* strukturom. Vremenska složenost tog algoritma jest $O(V^3)$, gdje je V broj čvorova u grafu[2].

U našem slučaju, $V = P+C+2$ jer imamo P proizvoda i C ograničenja, te dva dodatna čvora. Ako usporedimo ovu vremensku složenost ovog algoritma s prethodnim, u kojem smo tražili maksimalno sparivanje, primjećujemo da je ovaj algoritam bolji čim vrijedi $A^{2.5} > \sqrt{V}$ tj. $A^5 > V$ (prisjetimo se, A je prosječan broj istih proizvoda odnosno ograničenja). Budući da za

veliki V problem najvećeg protoka ionako nije moguće dovoljno brzo riješiti , već i za jako mali A ovaj algoritam ima prednost u odnosu na prošli.

Što ako znamo maksimalni protok, kakve to ima veze s početnim pitanjem?

Primijetimo da su ta dva problema usko povezana, da bi sva ograničenja bila zadovoljena, ukupan protok mora biti jednak ukupnom broju svih ograničenja. U gornjem primjeru vidimo da je maksimalni protok jednak 6, a pošto ograničenja ima ukupno 1+1+3+1 = 6, zaključujemo da svaka konfiguracija ponuda koja daje ovaj multiskup proizvoda zadovoljava dana ograničenja, i time predstavlja valjano rješenje.

U svim daljnjim razmatranjima često ćemo se osvrtati na ovaj algoritam koji vrlo efikasno (u vremenskoj složenosti $O(V^3)$) provjerava da li neka konfiguracija ponuda zadovoljava ograničenja koje je kupac postavio, i od sada pa nadalje nazivat ćemo ga "algoritam protoka".

2.2 POTRAGA ZA DETERMINISTIČKIM ALGORITMOM

2.2.1 Pohlepna strategija

Jedan od najjednostavnijih načina rješavanja problema optimalne kupovine jest pohlepan pristup*.

Postoji beskonačno mnogo pohlepnih algoritama, ali mi ćemo razmotriti sljedeći:

1) Kreni od praznog multiskupa ponuda
2) U svakom koraku probaj uzeti najjeftiniju ponudu koja zadovoljava neko ograničenje. Ako je to ograničenje disjunktno (tip *a*), zapamti da je ispunjeno i više ga ne razmatraj u sljedećim koracima. Ponudu uzmi u onoj količini u najvećoj količini koja nije višak
3) Provjeri da li su zadovoljena sva ograničenja, ako jesu prijeđi na korak 4), a ako nisu, vrati se na korak 2).
4) Izračunaj ukupnu cijenu svih proizvoda, uključujući sve modifikatore (popusti i slično), te tu cijenu proglasi trenutno najmanjom mogućom.

Na prvi pogled vidimo da ova pohlepna strategija zakazuje. Očiti protuprimjeri su sljedeći:

- tip #1: Postoje 2 disjunktna ograničenja koje je potrebno ispuniti, a na tržištu postoje točno dva proizvoda, A i B, koji nisu besplatni. Proizvod A zadovoljava oba ograničenja, a proizvod B samo jedno. Optimalno je iskoristiti ono ograničenje koje zadovoljava samo jedan proizvod prvo na način da se kupi taj proizvod, a zatim će, kad kupimo drugi proizvod, drugo ograničenje biti automatski ispunjeno. Ako je proizvod koji zadovoljava oba ograničenja jeftiniji, pohlepna strategija će naći ne-optimalno rješenje: kazat će da se trebaju kupiti 3 proizvoda, a to je sigurno skuplje.
- tip #2: Postoje 2 proizvoda koja se moraju kupiti, A i B, a za svaki od njih postoji ponuda u kojoj se taj proizvod prodaje po cijeni od 5 kn, ali također svaki od njih dobiva se besplatno ako se kupi proizvod C koji košta 6 kn. Optimalno je kupiti proizvod C jer onda za 6 kn imamo i proizvode A i B, čime smo ispunili zahtjev, a pohlepna strategija bi prvo uzela proizvode A i B.
- tip #3: Treba kupiti 5 komada proizvoda A. Cijena po komadu za taj proizvod kreće se ovako: 1-5 komada – 10 kn/kom; 6+ kom – 8 kn/kom. Pohlepni algoritam našao bi optimalnu cijenu od 50 kn (5 kupljenih proizvoda), a optimalno je kupiti 6 proizvoda koji se mogu dobiti za 48 kn.

Iako je u određenim uvjetima moguće riješiti neke slučajeve poput trećeg (primjerice ako je

funkcija cijene rastuća) uz pomoć sofisticiranijih pohlepnih tehnika, kombinacijom više "vrsta" protuprimjera uvijek možemo naći neki novi za koje će pohlepno rješenje dati krivi rezultat. Štoviše, u kasnijem poglavlju [3]pokazat ćemo da čak i ako uspijemo riješiti slučajeve protuprimjera tipa #2 i #3, protuprimjer tipa #1 nije moguće zaobići pohlepnim algoritmom.

S obzirom da u praksi skoro nikada pohlepan algoritam ne radi, koliko god on sofisticiran bio, mi ovdje više nećemo razmatrati ostale varijante pohlepnog algoritma. Zato ćemo radije razmotriti način rješavanja kojim ćemo sigurno pronaći najbolje rješenje koje zadovoljava ograničenja.

2.2.2 Iscrpno pretraživanje

Gotovo svaki problem, pa tako i ovaj, može se na ovaj ili onaj način riješiti uporabom grube sile: algoritmom koji na jednostavan način razmatra "sva moguća rješenja". Takav algoritam će zasigurno naći optimalno rješenje koje mi upravo tražimo, i to u konačnom vremenu. Postavlja se pitanje u čemu je problem? Odgovor je vrlo jednostavan – često takvi algoritmi imaju eksponencijalnu ili veću vremensku složenost. To u praksi znači da će se već za jako maleno ograničenje na ulazne parametre (tj. domenu problema) algoritam izvoditi vrlo sporo, a čim malo povećamo ulazne parametre, trajanje izvođenja će drastično skočiti.

Kako bi se uvjerili u njegovu neprikladnost, predložit ćemo jedno takvo rješenje u koje možemo biti sigurno da će naći optimalan rezultat.

Ideja je sljedeća: Probamo uzeti svaku količinu svake ponude, te za svaku takvu konfiguraciju ponuda provjerimo da li sadržani proizvodi zadovoljavaju kupčeve zahtjeve. Pošto je zaliha proizvoda ograničena, postoji konačno mnogo proizvoda. Stoga, je moguće isprobati sve konfiguracije ponuda u konačnom vremenu, što ćemo upravo i napraviti. Kako bi provjerili da li neka konfiguracija zadovoljava ograničenja poslužit ćemo se algoritmom koji smo opisali u prethodnom poglavlju. Slijedi analizira vremenske složenosti ovakvog načina rješavanja.

Pod pretpostavkom da su sve zalihe proizvoda ograničene i da svakog od ukupno P proizvoda ima u količini Q, te da postoji N ponuda koje uključuju mali broj proizvoda, to znači da je broj načina prihvaćanja neke ponude proporcionalan s Q , pa postoji N^Q konfiguracija ponuda. Budući da je za svaku konfiguraciju potrebno provjeri da li ona predstavlja valjano rješenje, što možemo obaviti u vremenskoj složenosti $O(P^3)$[4], ukupna vremenska složenost ovog pohlepnog algoritma jest $O(N^Q P^3)$. Vidljivo je da vrijeme izvođenja raste eksponencijalno s porastom zaliha proizvoda. Stoga, zaključujemo da je zbog svoje vremenske složenosti ovakvo rješenje neprikladno osim za u slučaju vrlo malog broja ponuda i oskudnih zaliha svih proizvoda.

2.3 RJEŠAVANJE PROBLEMA OPTIMIZACIJE KUPOVINE GENETSKIM ALGORITMOM

2.3.1 Genetski algoritam

Genetski algoritam je vrsta heurističkog algoritma koji se koristi za rješavanje problema optimizacije i pretraživanja kako bi se našlo optimalno ili približno optimalno rješenje.

Genetski algoritam se u pravilu koristi za rješavanje problema za koje ne postoji algoritam prihvatljive složenosti koji će naći točno rješenje, a *iscrpno pretraživanje* (engl. *exhaustive search*) ima preveliku vremensku složenost. To su najčešće problemi koji pripadaju klasi *NP* problema

3 Pohlepan algoritam je vrsta metaheurističkog algoritma koji uvijek odabire trenutno najbolji izbor (tj. lokalno najbolji) u nadi da će na kraju dobiti globalno optimalno rješenje.

4 Ovo vrijedi pod pretpostavkom da je broj ograničenja (C) zanemariv u odnosu na broj različitih proizvoda (P). Bez zanemarivanja broja ograničenja, složenost je $O((P+C)^3)$.

(engl. *non-deterministic polynomial time*), za koje ne postoji rješenje polinomijalne vremenske složenosti, pa je stoga jedini izbor iscrpno pretraživanje svih rješenja na ovaj ili onaj način. Budući da nikako ne stignemo iscrpiti sve mogućnosti, tu možemo upotrijebiti genetski algoritam. Genetski algoritmi općenito pokazuju dobre rezultate na kombinatoričkim problemima.

Dakle, genetski algoritam je svojstven po tome što na sustavan način pretražuje prostor rješenja (engl. *search space*) u potrazi za točnim rješenjem, ali pritom ne garantira da će naći točno rješenje. No praksa je pokazala da za većinu teških problema, uz dobre postavke parametara genetskog algoritma, postoji dobra vjerojatnost da će on pronaći neko rješenje koje je blizu točnog.

Stoga se genetski algoritmi koriste u optimizacijskim problemima u kojima nije od presudne važnosti da je rješenje najbolje moguće (u smislu točnosti), nego je sasvim prikladno svako rješenje koje je dovoljno dobro.

2.3.1.1 Princip rada

Princip rada genetskog algoritma sličan je Darwinovoj teoriji o evoluciji živih organizama u prirodi. Ta evolucijska teorija temelji se na pet pretpostavki[3]:

1) Vrste su plodne – broj potomaka je uvijek veći od trenutnog broja jedinki
2) Veličina populacije (tj. broj jedinki) je približno stalna (blagi rast se zanemaruje)
3) Količina hrane (odnosno resursa) jest ograničena
4) Kod vrsta koja se razmnožavaju spolno, svi potomci su različiti (zbog toga što se svojstva potomaka nasljeđuju od roditelja uz moguće varijacije)
5) Varijacije se često nasljeđuju

Iz gornjih pet pretpostavki slijedi da se svaka jedinka mora boriti za preživljavanje. Samo jedinke s dobrim svojstvima će opstati, a ostale će umrijeti. Pošto se svojstva nasljeđuju, s vremenom, dominirat će jedinke s dobrim svojstvima, ali zbog varijacija će uvijek postojati i određeni broj jedinki s manje dobrim svojstvima. Zbog dinamičnosti okoline, moguće je da iznenada jedinke s lošijim svojstvima postanu otpornije i bolje prilagođene okolini i tako postaju nove jedinke s dobrim svojstvima.

Najprije razjasnimo terminologiju[5] vezanu uz genetski algoritam:

Jedinka – jedno rješenje. U ovom kontekstu ju još nazivamo i genotipom jer jedino što nas zanima jest njezin genetski materijal odnosno svojstva koje posjeduje.

Populacija – skup jedinki tj. rješenja

Genotip – apstraktna reprezentacija jedinke. Nekad ga još nazivamo i kromosomom.

Križanje (engl. *crossover*) – izmjena genetskog materijala koja se događa prilikom spolnog razmnožavanja, pri čemu nastaje nova jedinka. U kontekstu genetskog algoritma, križanje nazivamo i operatorom križanja.

Selekcija – postupak u kojem se odlučuje koje će jedinke preživjeti, a koje neće.

Dobrota – mjera do koje neka jedinka zadovoljava kriterija tj. ograničenja. Engleski izraz za dobrotu je *fitness*.

5 Nazivi i terminologija koja se koristi u kontekstu genetskog algoritma nije u potpunosti u skladu s pojmovima iz biologije i genetike. Npr. genotip i genom nisu sinonimi, kao što niti, očigledno, kromosom nije sinonim za jedinku.

Algoritamski gledano, evoluciju možemo opisati sljedećim pseudokodom:

```
P ← inicijalizacija()  // stvori početnu populaciju
ponavljaj  {
  // provjeri je li pronađeno bolje rješenje od trenutno najboljeg
  D ← P  // odredi neki podskup rješenja za koje ćemo evaluirati dobrotu (najčešće D ≡ P)
  ∀ i ∈ D  // za svako rješenje iz tog skupa, pogledaj njegovu dobrotu
    rješenje ← min(rješenje, dobrota(D_i))

  // provjeri da li je uvjet prekida ispunjen i ako jest vrati trenutno najbolje rješenje
  ako ( prekid() ) onda
    vrati rješenje

  // stvori novu populaciju
  S ← selekcija(P) // odaberi rješenja koja ćeš križati
  R ← Ø // sačuvamo mjesto za novu populaciju
  ponavljaj dok (ne stvorimo dovoljan broj novih jedinki)  {
    // stvori novog potomka križanjem dvaju roditelja
    N ← križanje(P_i, P_j),  ∃i ∈ P ∧ ∃ j ∈ P
    R ← R ∪ mutiraj(N)  // mutiraj ga i dodaj u skup novih potomaka
  }

  // zamjena – stvori iduću generaciju jednake veličine
  P_sljed ← zamjena(P, R)  // od pripadnika stare i nove populacije
  P ← P_sljed / stara generacija izumire, ostaje nova
}
```

Slijedi pojašnjenje gore korištenih procedura koji označavaju ključne dijelove genetskog algoritma.

2.3.1.1.1 Inicijalizacija

Ovo je početna faza kojom započinje svaki genetski algoritam. Ovdje je potrebno stvoriti početnu populaciju koja će pokrenuti evoluciju. Na temelju te početne populacije križanjem će se stvoriti druga generacija, na temelju druge treća i tako dalje sve dok se ne ispuni uvjet prekida tj. genetski algoritam završi.

Pripadnike početne populacije treba pažljivo odabrati; preporučljivo je stoga odabrati dovoljno veliki uzorak raznovrsnih jedinki, po mogućnosti što različitijih, kako bi se što bolje pokrio prostor rješenja koji se pretražuje. Veličina početne populacije obično varira između par desetaka do nekoliko tisuća jedinki, no ovisi o problemi i računalnoj moći sustava na kojem se vrti algoritam (npr. kod paralelnih genetskih algoritama ovaj broj zna biti puno veći).

2.3.1.1.2 Selekcija

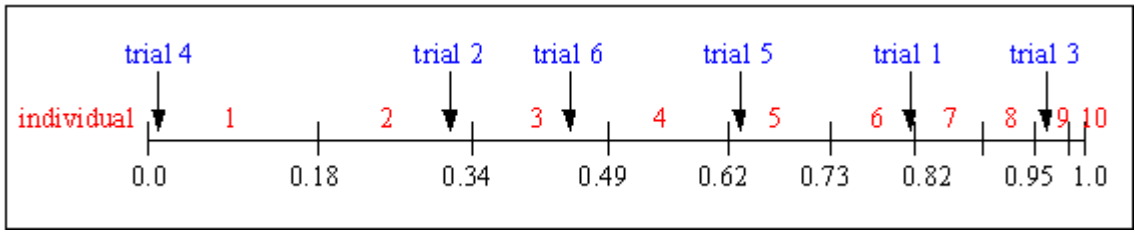
U svakoj generaciji, potrebno je odrediti koje će se jedinke križati kako bi se dobilo potomstvo za iduću generaciju.

Postoje različiti načini selekcije, od kojih ćemo razmotriti sljedeće:

- odsječena selekcija (engl. *truncation selection*) – Ovo je vjerojatno jedna od najgorih vrsta selekcije i u praksi se gotovo nikada ne koristi. Radi se o biranju određenog postotka jedinki s najvećom dobrotom.

Implementacijski detalj: Ovu vrstu selekcije možemo efikasno obaviti primjerice korištenjem gomile (engl. *heap*) u vremenskoj složenosti O(N log N), gdje je N ukupan broj jedinki.

– proporcionalna – ova selekcija, poznata još i pod nazivom *Roulette-wheel selection*, osigurava da svaka jedinka ima vjerojatnost odabira koja je proporcionalna njenoj dobroti. Prethodni naziv je ušao u širu uporabu zbog toga što se zaista može uzeti jedno kolo te obojiti isječke kruga u različite boje tako da svaki isječak ima površinu koja je proporcionalna dobroti odgovarajuće jedinke, zatim zavrtjeti to kolo i kad se ono zaustavi pogledati boju isječka na koju kazaljka pokazuje te se na taj način izabire odgovarajuća jedinka.[4]

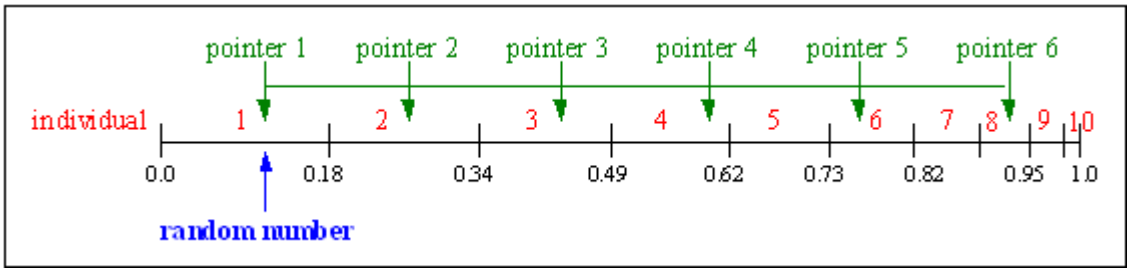


Slika 4: Prikaz propercionalne selekcije [5]

Implementacijski detalj: Ovu selekciju možemo efikasno implementirati pomoću binarnog pretraživanja. Na početku napravimo polje kumulativnih suma, i onda prilikom svake selekcije pogledamo u kojem se "pretincu" nalazi nasumično odabran broj. Složenost selekcije S jedinki ovakvom metodom jednak je $O(N + S \log N)$, gdje je N broj ukupan jedinki.

– turnirska selekcija (engl. *tournament selection*) – jednostavna, popularna i relativno dobra metoda selekcije. Ideja K-turnirske selekcije jest sljedeća: u svakom koraku K nasumično odabranih jedinki "odigraju turnir", te pobjednik turnira ostaje (tj. odabire se). Za $K = 1$ niti ne treba evaluirati dobrotu jedinke (rješenja) pošto se zapravo radi o običnoj nasumičnoj selekciji.

– stohastičko univerzalno uzorkovanje – ideja je vrlo slična kao *Roulette-wheel selekcije*; napravimo N pretinaca čija je veličina proporcionalna dobroti pripadajuće jedinke, zatim sortiramo jedinke po dobroti, te po interval kojeg čine svi pretinci zajedno ravnopravno postavimo S selekcijskih kazaljki. Izabiremo one jedinke na čije pripadajuće pretince pokazuju kazaljke, onoliko puta koliko kazaljka pokazuje na taj pretinac. N i S označavaju redom ukupan broj jedinki, te broj jedinki koje izabiremo.



Slika 5: Prikaz stohastičkog univerzalnog uzorkovanja [6]

Implementacijski detalj: Nakon što smo efikasno sortirali jedinke po dobroti, moramo paziti da pretinac na koji pokazuje selekcijskih kazaljki ne tražimo svaki puta iznova slijeva na desno. Zato ćemo razmatrati kazaljke selekcije slijeva na desno i pamtiti indeks najdesnijeg

pretinca koji počinje prije kazaljke, što znači da selekciju na sortiranom nizu možemo izvesti linearno. Uzevši u obzir i sortiranje, imamo vremensku složenost $O(S + N \log N)$.

2.3.1.1.3 Križanje

Prilikom stvaranja nove jedinke, kombinira se genetski materijal oba roditelja pri čemu nastaje novi genotip koji ima sličnosti s oba ishodišna ali je, u cijelosti gledano, različit. Zato se križanje još naziva i rekombinacijom.

U općem slučaju, genotip može biti bilo koja konačna struktura koja sadrži informaciju, i tada se križanje obavlja tako da se uzmu dijelovi genetskog materijala iz oba roditelja te se ti dijelovi "spoje" kako bi se dobio novi genetski materijal koji je u određenoj mjeri sličan roditeljskim. Pritom je važno da se novi genetski materijal može pravilno interpretirati.

Ipak, zbog analogije s kromosomom u prirodi, te jednostavnosti, najčešće se kao genotip koristi neka linearna struktura fiksne duljine kao što je primjerice vektor. Česte metode križanja nad takvom linearnom strukturom su sljedeći (napomenimo unaprijed da operacija križanja nije destruktivna za vektore roditelje, iako ćemo koristiti riječ "zamjena"):

- križanje s jednom točkom prekida – ovo je najjednostavnija vrsta križanja, koja je za jednostavne probleme sasvim zadovoljavajuća. Ako pozicije vektora označimo indeksima od 1 do L, gdje je L duljina vektora, odaberemo nasumičan indeks i u intervalu [1, L-1], te na mjestu između indeksa i i (i+1) vektore roditelje "prerežemo" i potom zamijenimo prerezane dijelove koji predstavljaju iste indekse (npr. ako imamo vektor ABC i DEF, a indeks je 2, novonastali vektori su ABF i DEC).
- križanje s više točaka prekida – ako se radi o križanju s N točaka prekida, odabere se N različitih cjelobrojnih indeksa u intervalu [1, L-1], gdje je L duljina vektora, potom se vektori oba roditelja izrežu na tim mjestima, i konačno se neki nasumično odabrani dijelovi vektora jednog roditelja zamijene odgovarajućim dijelovima drugog vektora). Jedan od tako dobivena 2 vektora proglasimo novom jedinkom (ili oba).
- uniformno križanje – svaki član vektora nezavisno se bira u odnosu na susjedne članova tj. nije povezan. Drugim riječima, za prvi član novog vektora uzmemo nasumično odabran prvi član nekog od roditelja vektora, zatim za drugi član opet uzmemo neki od dva druga člana roditeljskih vektora itd.

Kod implementacije genetskog algoritma važno je da vjerojatnost križanja bude puno veća od vjerojatnosti mutacije.

2.3.1.1.4 Mutacija

Mutacijom se osigurava da se svojstva jedinke tj. njezin genetski materijal izmjeni. Primjerice, ako je genotip jedinke predstavljen nizom bitova, mogli bismo nasumično odabrati neke bitove i promijeniti im vrijednosti. Kod složenih genotipova mutacija je nešto kompliciranija.

Osim što povećava različitosti između jedinki, primarna svrha mutacija jest omogućiti da neka generacija jedinki povrati dobar genetski materijal (tj. svojstva) koja su se izgubila.

Zbog toga je kod same implementacije bitno osigurati da se primjenom uzastopnih mutacija može teoretski dobiti bilo koja jedinka koja je neko rješenje tj. da se pokrije cijeli prostor rješenja (ili širi). Drugim riječima treba se omogućiti da svaka jedinka može biti stvorena primjenom jedne ili više uzastopnih mutacija. Ipak, ne treba pretjerati s učestalošću mutacija da ne bi došlo do prevelikog raspršenja; jer zbog prečestog mijenjanja (mutacije) može doći do odbacivanja dobrog genetskog materijala (a time i jedinki koje ga nose), pa se narušava kvaliteta genetskog algoritma. Iz tog se

razloga za vjerojatnost mutacije jedinke uglavnom uzima broj između 0.01 i 0.001.

2.3.1.1.5 Zamjena

Nakon što se križanjem i mutiranjem stvori nova populacija, potrebno je odrediti koje će novostvorene jedinke zamijeniti jedinke iz stare populacije.

Postoje različite metode zamjene, poput:

- stvori jednak broj pripadnika nove populacije koliko je velika stara, te zamijeni cijelu staru populaciju novom (čista zamjena) – ovo je najjednostavnije i najlošija metoda, pa ju stoga nije preporučljivo koristiti osim za vrlo jednostavne probleme
- elitistička strategija – određen broj najgorih jedinka iz stare generacije se zamjenjuje novima tj. jedinke s najboljim svojstvima dugo žive
- jedinke koje su preživjele određeni broj generacija (tj. živjele su dovoljno dugo) bivaju zamijenjene novonastalim jedinkama. Ovime se svakoj jedinki pruži prilika da prenosi svoja dobra svojstva nekoliko generacija.[7]
- stvori se dovoljno velika nova generacija te se iz nje uzmu samo jedinke s najvećom dobrotom koje će zamijeniti neke jedinke iz stare populacije (npr. one s najmanjom dobrotom)

U praksi se često koriste kombinacije ovih metoda, npr. elitistička u kojoj se uzme nekoliko najboljih jedinki iz nove generacije pa se zamijeni isti broj najgorih jedinki iz stare populacije. Time se dobivaju metode koje imaju dobra svojstva ostalih drugih metoda s kojima se kombiniraju.

2.3.1.1.6 Evaluacija dobrote

Kako bi genetski algoritam uopće radio, moramo mu odrediti kako da vrednuje (evaluira) rješenja koja pronalazi. Upravo tome služi *funkcija dobrote.* Funkcija dobrote rangira rješenja prema optimalnosti.

Funkcija evaluacije dobrote je usko povezana s ciljem problema. Na primjer, ako nam je cilj naći najmanji X koji je veći od Y, logično bi bilo da rješenja koja su bliža Y (a veća od njega) imaju veću dobrotu.

Zbog same prirode genetskog algoritma, ta funkcija će često bivati pozvana, pa je od izuzetne važnosti da ona bude učinkovito implementirana[8]. Zbog toga se u praksi nekad koriste aproksimacijske metode koje mogu brzo odrediti približnu optimalnost rješenja uz određeni rizik da previde neka rješenja koja su mogla biti još optimalnija. Ipak takva rješenja svojom brzinom kompenziraju taj nedostatak s obzirom na to da će algoritam moći evaluirati više rješenja u istom vremenu (budući da je vrijeme trajanja evaluacije kraće).

2.3.1.1.7 Uvjet prekida

Kada se zadovolji uvjet prekida, algoritam prestaje i više se ne stvaraju nove generacije.

Neki često korišteni uvjeti prekida (i njihove kombinacije) su sljedeći[9]:

- pronađeno je dovoljno dobro rješenje ili rješenje koje zadovoljava minimalne kriterije
- ručna intervencija, npr. čovjek vidi da je algoritam skrenuo u pogrešnom smjeru i odluči prekinuti izvođenje algoritma kako bi ga ponovno mogao pokrenuti kad bolje podesi parametre genetskog algoritma

- došlo je do stagnacije u pronalaženju boljih rješenja – u nekoliko uzastopnih generacija nije stvorena jedinka koja bi bila bolja od najbolje dosadašnje
- dostignuta je određena generacija (npr. stota, tisućita itd.)
- prekoračeno je dopušteno vremensko ograničenje (npr. algoritam se automatski zaustavlja nakon 4 sata rada)

2.3.2 PRIMJENA GENETSKOG ALGORITMA NA PROBLEM KUPOVINE

Da bi uopće problem kupovine mogli riješiti primjenom genetskog algoritma, potrebno je razmotriti kako ćemo prikazati podatke specifične za naš problem; što će nam biti jedinke, kakav genotip ćemo koristiti, kako ćemo izvesti evaluaciju dobrote i sl.

2.3.2.1 Izbor genotipa

U prethodnim poglavljima, već smo se uvjerili da sve što je dovoljno kako bi mogli izračunati ukupnu cijenu koju kupac plaća u slučaju da prihvati neki multiskup ponuda jesu identifikatori svake od prihvaćenih ponuda, te podatak o tome koliko smo puta pojedinu ponudu prihvatili. Na temelju identifikatora ponude onda možemo dohvatiti sve ostale podatke vezane za dotičnu ponudu.

Za početak, označimo sa L ukupan broj ponuda na tržištu, te identifikatore ponuda sa O_i ($1 \leq i \leq L$). Pošto su identifikatori teoretski bilo kakvi brojevi koji su međusobno različiti, bilo bi zgodno preslikati ih na interval $[1, L]$. Definirajmo stoga sljedeću bijektivnu funkciju: $H(x): O \rightarrow [1, L]$.

Dakle, konfiguracija ponuda može se prikazati skupom parova $(H(O_i), T_i)$, gdje je O_i identifikator i-te ponude, a T_i broj puta koliko smo i-tu ponudu prihvatili. To znači da bi za genotip teoretski mogli uzeti vektor parova varijabilne duljine. Osim što bi takav vektor učinkovito iskorištavao memoriju, vrijeme izvođenja operacija križanja i mutacije bile bi proporcionalno broju prihvaćenih ponuda.

Uočimo dvije nelagodne okolnosti vezane za gornji način kodiranja konfiguracije ponuda. Prvo, prilikom mutacije potrebno je paziti na to da se indeksi ponuda kreću od 1 do L, a količine pak od 0 do neke gornje granice M (određene zalihama prodavača), što stvara dodatne komplikacije. Drugo, istu konfiguraciju ponuda možemo predstaviti na $L!$ načina, pa svojstvo relativne pozicije ponuda u vektoru gubi smisao, što zapravo znači da od križanja s jednom ili više točaka prekida nemamo pretjerane koristi. Čak i u slučaju da vektor cijelo vrijeme držimo sortiranim, opet je smisao relativne pozicije izgubljen, a bilo bi i upitno kako bi ostvarili ravnopravno (engl. *unbiased*) križanje dvaju vektora različite duljine). U tom slučaju možda bi najbolje bilo posegnuti za uniformnim križanjem.

Zato ćemo predložiti i jedno jednostavnije kodiranje: fiksan vektor duljine L, čije će vrijednosti biti nenegativni brojevi manji od neke konstante M. Očigledno, sada nam broj na i-tom mjestu u vektoru govori koliko ima prihvaćenih ponuda s indeksom i (tj. ponuda za koje funkcija H vraća broj i).

2.3.2.2 Križanje

Ako za genotip uzmemo vektor fiksne duljine, križanje možemo ostvariti na način identičan onom koji je opisan u poglavlju o genetskom algoritmu. Zbog boljih rezultata, koristimo križanje s dvije točke prekida.

Dakle, novonastali vektor će imati jedan interval koji će sadržavati pripadajuće elemente jednog roditelja, a ostali elementi bit će preuzeti od drugog vektora roditelja. Gledajući sa strane konfiguracije ponuda, to znači da će u novonastaloj konfiguraciji biti izmijenjena brojnost samo onih ponuda koje zahvaća taj interval, dok će ostatak biti preuzet od jednog roditelja.

2.3.2.3 Mutacija

Za genotip predstavljen vektorom fiksne duljine, mutacija se vrši na način da se promijeni neki element vektora (ili više njih). To će za posljedicu imati da će broj jedne ili više ponuda u konfiguraciji biti izmijenjen. Nove vrijednosti su, pritom, isključivo nenegativni cijeli brojevi iz intervala zatvorenog intervala [0, Q_{max}], gdje je Q_{max} najveća količina u kojoj se neka ponuda može prihvatiti (bilo zbog zaliha proizvoda ili zbog zaliha same ponude).

2.3.2.4 Evaluacija dobrote rješenja

Postoje dvije vrste rješenja, ona koja zadovoljavaju ograničenja, te ona koja ne zadovoljavaju ograničenja. Dakle, prva stvar koju ćemo napraviti jest provjeriti da li rješenje uopće zadovoljava ograničenja tj. da li je ono *valjano* (engl. *feasible*).

Primjenjujući već spomenuti "algoritam protoka", ovu provjeru možemo obaviti u vremenskoj složenosti $O((P+C)^3)$, gdje je P ukupan broj različitih proizvoda sadržan u ponudama, a C ukupan broj različitih ograničenja. Naravno, da bi saznali ukupne proizvoda uključeno u ponude trebali bi proći kroz sve ponude što u slučaju da nam je genotip vektor fiksne veličine ima složenost $O(L + P)$ koju valja pridodati gore navedenoj.

Dobrota se potom za rješenja koja zadovoljavaju uvjete definira kao:

$-ukupna\_cijena$, jer dobrota linearno pada s porastom cijene; što je cijena manja, dobrota će biti bliže nuli, pa će samim time rješenje imati veću nagradu.

Rješenja koja ne zadovoljavaju ograničenja ćemo također vrednovati s ciljem da ona koja skoro zadovoljavaju ograničenja ipak više nagradimo u odnosu na ona daleko od toga. Međutim, svako takvo rješenje koje ne zadovoljava ograničenja imat će puno manju dobrotu od najgoreg rješenja koje zadovoljava ograničenja. Stoga bi za nevaljana rješenja mogli reći da je dobrota jednaka:

$-maks\_cijena + maks\_protok$, gdje je *maks_cijena* neka jako velika cijena koja nikad neće biti postignuta niti za jedno rješenje koje zadovoljava ograničenja.

2.4 PROGRAMSKA IMPLEMENTACIJA

Kako bi se istražilo kako se ponaša genetski algoritam za rješavanje problema kupovine (u daljnjem tekstu problema), napravljena je programska implementacija.

Aplikacija je izrađena u programskom jeziku C++, a korišteni su neke vanjske biblioteke kao što je Boost – generički skup biblioteka za C++, te ECF (*Evolutionary Computation Framework*). Što se tiče portabilnosti, radi se o višeplatformskoj aplikaciji; podržani se operativni sustavi *UNIX* familije (uključujući i *Linux*), te inačice *Windowsa*.

U programu su implementirana tri algoritma: iscrpno pretraživanje (*brute force*), pohlepan algoritam te genetski algoritam. Također, postavke (parametri) genetskog algoritma mogu se mijenjati putem konfiguracijske datoteke. Na taj način mogu se postaviti parametri koji utječu na optimalnost i brzinu genetskog algoritma.

2.4.1 BUDUĆI PLANOVI

Mogućnost nadogradnje trenutne aplikacije su velike. Prilikom same izrade obraćala se pozornost na to da implementacija bude dovoljno apstraktna kako bi se olakšala buduća nadogradnja, te da kod bude portabilan.

U budućnosti, planira se izrada grafičkog sučelje koristeći *GTK+* framework (preko sučelja *GTKmm* za jezik *C++*), koji je također *cross-platform*, kako bi se olakšalo i ubrzalo upravljanje i

korištenje aplikacije (trenutno sve ide preko parametara komandne linije). Na taj način bi se aplikacija odmah učinila primjerenijom za osobnu uporabu određenoj skupini korisnika.

U vidu još veće pristupačnosti prosječnom korisniku, koji bi mogao imati osobne koristi od ovakve aplikacije, veći dio aplikacije mogao bi se prebaciti na jedan poslužitelj, te izraditi web-sučelje preko kojeg bi prosječan korisnik vrlo lagano mogao raditi "upite" sa svojim zahtjevima za kupovinu.

2.5 USPOREDBA GENETSKOG ALGORITMA S POSTOJEĆIM POSTUPCIMA

U sljedećoj su tablici prikazani rezultati pokretanja svakog od tri implementirana algoritma (iscrpno pretraživanje – engl. *brute force*, pohlepni algoritam i genetski algoritam), na različitim primjerima. Genetski algoritam je pokrenut 5 puta, a uzet je najbolji rezultat. Izvođenja programa nisu bila striktno vremenski ograničena, ali algoritam iscrpnog pretraživanja (*brute force*) je na primjerima s većim ulaznim podacima ručno prekinut nakon nekoliko minuta izvođenja, jer je bilo očito da neće će potpuno izvođenje trajati neprimjereno dugo. U tom slučaju uzet je najbolje dotad nađeno rješenje (označeno zvjezdicom).

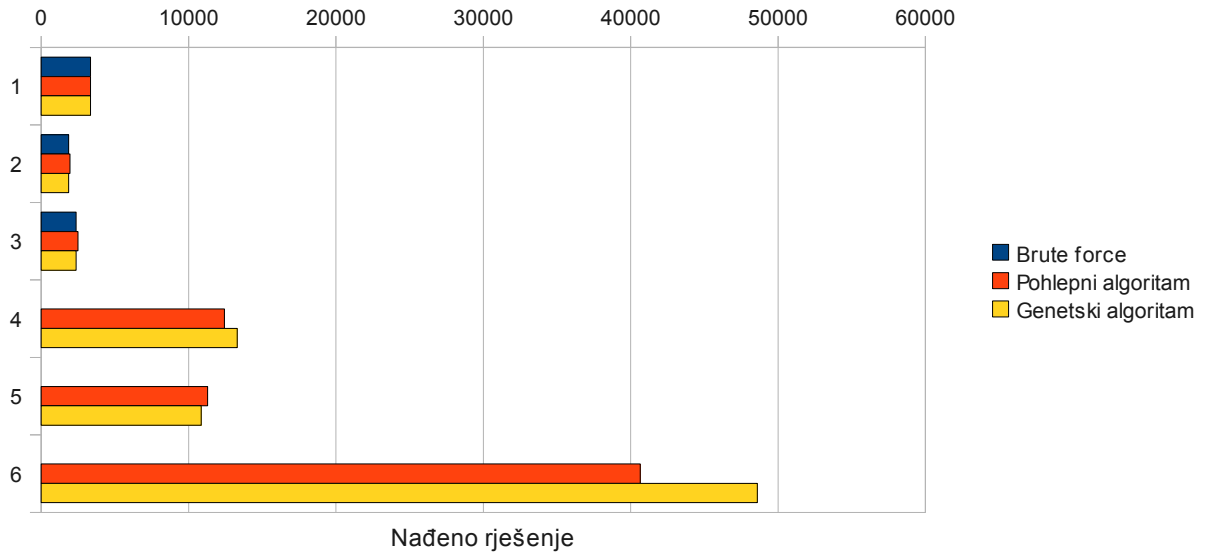
Napomena: brojevi u tablici označavaju cijenu u lipama (manje je bolje)

#	***Brute force***	**Pohlepni algoritam**	**Genetski algoritam**
1	3340	3350	3340
2	1857	1950	1857
3	2370	2487	2370
4	≤10705*	12425	13300
5	≤10408*	11290	10860
6	≤40500*	40650	48600
7	-	300557	126069
8	-	6086725	1976383
9	-	5522821	1857375
10	-	4737946	2973558

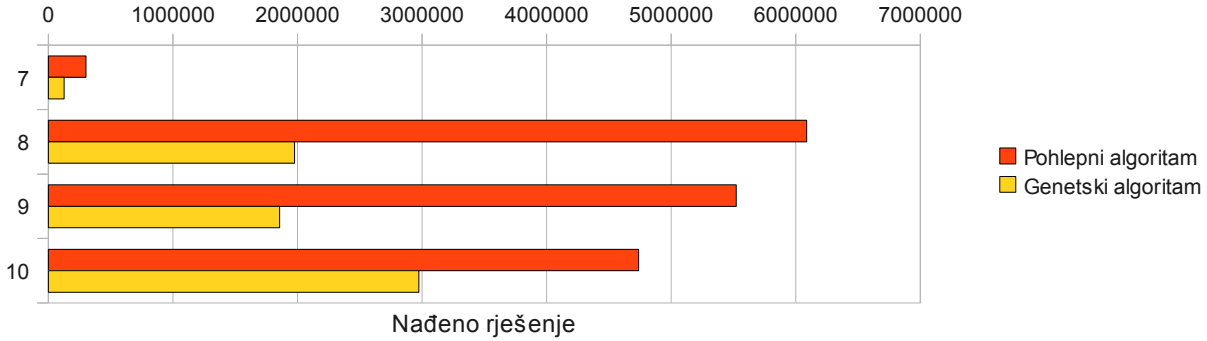
Tablica 1. Usporedni rezultati različitih algoritama

Prva polovica primjera izrađena je ručno zadavanjem najviše 5 ponuda i ograničenjem količine svakog proizvoda na najviše 10, dok je druga polovica primjera generirana programski i sadrži primjere s većim ograničenjima na spomenute veličine.

Na slikama 6. i 7. (ispod) grafički su prikazani rezultati iz Tablice 1. Rezultati su odvojeni u dva grafa zbog razlike u par redova veličine, kako bi se lakše uočili omjeri.



Slika 6: Grafički prikaz rezultata izvođenja



Slika 7: Grafički prikaz rezultata izvođenja

Iz rezultata je vidljivo da genetski algoritam skoro uvijek nalazi rješenje blizu optimalnog, a glavna je prednost manja složenost od iscrpnog pretraživanja.

Iako se iz rezultata čini da pohlepno rješenje pronalazi vrlo dobra rješenja, zbog samog načina na koji je ono napravljeno (kao što je ranije već objašnjeno), ovakva rješenja su vrlo nepouzdana. Samim time, ove rezultate treba promatrati s tom napomenom u vidu.

Također, važno je napomenuti da rješavanje ovog problema iscrpnim pretraživanjem nije efikasno, budući da se količina proizvoda mora ograničiti, kao što je to bilo učinjeno u prva tri primjera. Konkretno, najveća količina proizvoda u tim primjerima bila je oko 10, a u kasnijim u pitanju su veća ograničenja, pa taj algoritam nije završio s izvođenjem u razumnom vremenu.

3 ZAKLJUČAK

Genetski algoritam pokazao se efikasnim u rješavanju ovog problema. Iz razultata su vidljive dvije stvari. Prvo, na test primjerima s malim ulaznim ograničenjima (prva polovica), pokazalo se da genetski algoritam zaista pronalazi dovoljno dobro rješenje, čak u par primjera i optimalno (dobiveno iscrpnom pretragom). Drugo, na test primjerima s većim ulaznim ograničenjima (druga polovica), gdje optimalno rješenje nije bilo moguće naći iscrpnom pretragom, genetski algoritam je u svim pogledima nadmašio pohlepno, što se posebno vidljivo na 7. i 8. test primjeru, gdje je omjer pronađenih rješenja redom približan 4 i 3 (u korist genetskog algoritma).

Najveća prednost genetskog algoritma je upravo njegov princip rada. Taj princip omogućava da rezultati budu bliski optimalnom rješenju, a vremenski je puno efikasniji, što je pokazano pomoću simulacijskog programa na test primjerima.

Za razliku od pohlepnog algoritma, genetski algoritam nudi fleksibilnost u smislu da male modifikacije početnog problema ne zahtjevaju veće preinake u samom algoritmu; sve što je potrebno je promijeniti način evaluacije dobrote, dok kod pohlepnog algoritma i uvođenje najsitnije novine može dovesti do drastične promjene strategije.

Još jedna prednost genetskog algoritma je ta da možemo odrediti njegove parametre, pa ih sukladno prethodno dobivenim rezultatima promijeniti kako bismo eventualno poboljšali rezultate.

4 LITERATURA

Literatura

[1] Eric W. Weisstein, "Multiset" - Wolfram Mathworld, 2010., *MathWorld--A Wolfram Web Resource*, http://mathworld.wolfram.com/Multiset.html
[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, *Introduction to Algorithms*: Graph algorithms, MIT Press, 2001.
[3] Marko Čupić, *Prirodom inspirirani optimizacijski algoritmi*: Genetski algoritam, 2009.
[4] Marko Čupić, *Prirodom inspirirani optimizacijski algoritmi*: Genetski algoritam, 2009.
[5] GEATbx - *The Genetic and Evolutionary Algorithm Toolbox for Matlab*, 2010, http://www.geatbx.com/docu/algindex-12.gif
[6] GEATbx - *The Genetic and Evolutionary Algorithm Toolbox for Matlab*, 2010, http://www.geatbx.com/docu/algindex-13.gif
[7] Ali M. S. Zalzala, Peter J. Fleming, *Genetic algorithms in engineering systems*: Reinsertion, The Institution of Electrical Engineers, London, 1997.
[8] Wikipedija - suradnici, Fitness function, 2010., *Wikipedia – The Free Encyclopedia*, http://en.wikipedia.org/wiki/Fitness_function
[9] Wikipedija - suradnici, Genetic algorithm, 2010., *Wikipedia – The Free Encyclopedia*, http://en.wikipedia.org/wiki/Genetic_algorithm