SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br.1029

Primjena neparametarskih algoritama u optimizaciji

Marko Budiselić

Zagreb, lipanj 2015.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA DIPLOMSKI RAD PROFILA

Zagreb, 26. veljače 2015.

Predmet: **Analiza i projektiranje računalom**

DIPLOMSKI ZADATAK br. 1029

Pristupnik: **Marko Budiselić (0036455459)**
Studij: Računarstvo
Profil: Računarska znanost

Zadatak: **Primjena neparametarskih algoritama u optimizaciji**

Opis zadatka:

Opisati neparametarski pristup rješavanju optimizacijskih problema i istražiti postojeće neparametarske algoritme. Ostvariti različite inačice algoritama s obzirom na prikaz rješenja i područje primjene. Usporediti parametarske i neparametarske evolucijske optimizacijske algoritme. Eksperimentalno utvrditi učinkovitost algoritama na problemima kontinuirane i kombinatoričke optimizacije. Radu priložiti algoritme, izvorne tekstove programa i rezultate uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Zadatak uručen pristupniku: 13. ožujka 2015.
Rok za predaju rada: 30. lipnja 2015.

Mentor:

Izv. prof. dr. sc. Domagoj Jakobović

Djelovođa:

Doc. dr. sc. Tomislav Hrkać

Predsjednik odbora za diplomski rad profila:

Prof. dr. sc. Siniša Srbljić

Zahvaljujem roditeljima na neizmjernoj potpori tijekom mojeg dosadašnjeg života i mentoru prof. dr. sc. Domagoju Jakobovi\'cu na velikoj potpori tijekom studija.

Sadržaj

1. Uvod

Standardni evolucijski algortmi posjeduju određeni skup parametara. Ideja neparametarskih algoritama je ukloniti sve ili barem neke parametre kako bi se algoritam lakše prilagodio problemu i brže došao do optimalnog rješenja. Glavnina ovog rada bit će analiza algoritama koji rade nad binarnom reprezentacijom rješenja, dakle nizom bitova, no analizirat će se i permutacijski tip rješenja.

U ovom će se radu pokušati usporediti neparametarski algoritmi primarno s genetskim algoritmom. Najmoderniji neparametarski algoritam je algoritam P3 nastao na temelju algoritma LTGA. Ta će se dva algoritma detaljno analizirati. Jedna od osnovnih ideja P3 algoritma je piramidalna populacija beskonačne veličine i ta ideja će biti upotrijebljena kako bi se uklonio parametar veličine populacije.

U sklopu ovoga rada osmišljeni su algoritmi PGA i PCGA. Oni su varijante algoritama GA i CGA koje imaju piramidalnu populaciju po uzoru na algoritam P3. Za algoritam P3 implementirana je i podrška za cjelobrojni prikaz rješenja. Svi navedeni algoritmi bit će ispitani na nekoliko standardnih optimizacijskih problema kao i na stvarnim problemima poput kombinatoričke mreže [12] i Booleovih funkcija [14].

2. Algoritmi

2.1. Teorem sheme

Za prikaz nizom bitova vrijedi teorem sheme koji govori kako prosječna dobrota populacije genetskog algoritma raste eksponencijalno s brojem generacija. Teorem je postavio John Holland 1975. godine [1]. Iz tog teorema proizlazi da ima smisla koristiti genetski algoritam za rješavanje optimizacijskih problema jer jamči eksponencijalni napredak rješenja kroz generacije. Shema je niz bitova, primjerice 1*01*1. Na mjestima gdje je znak * može se postaviti vrijednost 1 ili 0 i to je varijabilni dio u shemi.

U izrazu (1) simboli su: H je shema, p je vjerojatnost narušavanja sheme, p_c je vjerojatnost križanja, p_m je vjerojatnost mutacije, l je duljina niza bitova, $m(H,t)$ je broj različitih nizova u shemi, $f(H)$ je prosječna dobrota za cijelu shemu, a_t je prosječna dobrota za generaciju t, $o(H)$ je broj varijabilnih bitova, $\delta(H)$ je udaljenost od prvog i zadnjeg bita koji su specifični.

$$\begin{aligned} E(m(H,t+1)) &\geq \frac{m(H,t)f(H)}{a_t}[1-p] \\ p &= \frac{\delta(H)}{l-1}p_c + o(H)p_m \end{aligned} \tag{1}$$

Ovo je bio kratak teorijski osvrt na činjenicu da ima smisla koristiti genetski algoritam za razne optimizacijske probleme jer postoji garancija napredovanja. Problemi koji posjeduju velik skup mogućih rješenja, gdje detaljno pretraživanje prostora rješenja nije moguće, moraju se rješavati primjenom raznih heurističkih metoda. Dobro je znati da postoji teorijski dokaz za korištenje, primjerice, genetskog algoritma. U narednom dijelu bit će opisani svi implementirani optimizacijski algoritmi, njihovi parametri i operatori.

2.2. Genetski algoritam - GA

Rad na genetskim algoritmima (u daljnem tekstu GA) započeo je John H. Holland krajem 60-ih godina u sklopu svojih istraživanja adaptivnih umjetnih sustava. Tijekom posljednja tri desetljeća genetski su se algoritmi pokazali kao moćne i općenite metode za rješavanje raznolikih problema na područjima inžinjerske prakse. Razlog tome leži u njihovoj jednostavnosti implementacije i prilagodljivosti velikom broju problema. Prema načinu rada

genetski algoritmi spadaju u metode usmjerenog slučajnog pretraživanja prostora rješenja (engl. *guided random search techniques*). Osnovna snaga tih metoda, u odnosu na razne determinističke postupke optimizacije, je mogućnost određivanja optimuma u višemodalnom prostoru. U tom slučaju deterministički algoritmi predugo traju i nemoguće je s njima naći pravo rješenje. Ali, za razliku od determinističkih metoda pomoću kojih je, ako ne traju predugo, moguće dobiti rješenje sa željenom točnošću, stohastičke metode ne garantiraju pronalaženje globalnog optimuma kao ni traženu točnost.

Genetski algoritmi imitiraju prirodnu evoluciju na način da proces koji se optimira predstavlja okolinu u kojoj žive jedinke. Svaka jedinka predstavlja jednu kombinaciju ulaznih parametara kodiranih na primjeren način. Podaci koje sadrži jedinka predstavljaju njezin genetski materijal, a vrsta genetskog materijala naziva se genotip. Jednako kao i u prirodnoj selekciji, selekcijom u genetskom algoritmu biraju se jedinke prema svom genetskom materijalu: one sa kvalitetnijim genima (tj, one koje daju bolje rezultate) imat će veću šansu za preživljavanje i dobit će priliku da prenesu svoj genetski materijal na potomstvo. Na taj način populacija u genetskom algoritmu napreduje dajući sve bolja rješenja za problem koji se optimira. Proces selekcije, reprodukcije i manipulacije genetskim materijalom se ponavlja sve dok nije zadovoljen uvjet zaustavljanja genetskog algoritma.

Genetski algoritam sastoji se od nekoliko ključnih faza. U fazi stvaranja početne populacije najčešće se generiraju nasumična rješenja iz prostora svih rješenja. Nadalje, u fazi evolucije svakom se rješenju iz populacije pridružuje realan broj koji predstavlja dobrotu te jedinke (mjera u odnosu na ostale jedinke). Budući da algoritam ima više iteracija, prije svake nove iteracije provjerava se je li zadovoljen uvjet zaustavljanja, primjerice, konstantan broj iteracija ili broj iteracija algoritma u kojem nije poboljšano najbolje rješenje ili broj evaluacija funkcije cilja. U svakoj iteraciji, operatorom selekcije, biraju se po dvije jedinke koje će biti kombinirane za dobivanje novoga rješenja; to se kombiniranje obavlja operatorom križanja. Svako novo rješenje podvrgava se operatoru mutacije uz određenu vjerojatnost. Nakon operatora mutacije, operatorom zamjene se novo generirano rješenje ubacuje u populaciju rješenja. Populacija može biti stalna (nad postojećom populacijom se odabire jedinka koja će biti zamijenjena s novom) ili generacijska (cijela se populacija mijenja novom populacijom).

Genetski algoritam ne spada u neparametarske algoritme, ali u ovom radu biti će razma-

tran kao referentna točka za usporedbu s ostalim algoritmima. Vrijednost genetskog algoritma direktno slijedi iz teorije sheme koja govori da će algoritam kroz generacije napredovati i tako doći do zadovoljavajućeg rješenja. Genetski algoritam ima tri bitna parametra: veličina populacije, faktor križanja i faktor mutacije. Kako bi genetski algoritam producirao dobre rezultate potrebno je ugoditi te parametre za dani problem. To može biti problem jer da se ugode navedeni parametri potrebno je provesti pretraživanje prostora parametara, što može biti vremenski zahtjevno. Osim složenosti ugađanja parametara, algoritam može imati negativne posljedice zbog fiksne vrijednosti nekog parametra, primjerice, zbog fiksne vrijednosti faktora križanja algoritam može prebrzo ili presporo konvergirati.

Operatori koji se koriste u genetskom algoritmu popisani su u nastavku, a pseudokod algoritma dan je u algoritmu 1. Za dani tip rješenja (genotipa), uniformnom razdiobom bira se jedan od mogućih operatora. Detaljnije o operatorima može se pronaći u knjizi: *Prirodom inspirirani optimizacijski algoritmi* [11].

Operatori križanja permutacije:

- OX
- PMX

Operatori križanja niza bitova:

- UX
- One point
- Two point

Operatori mutacije permutacije:

- Zamijene se 2 elementa N puta s vjerojatnošću mutacije. N puta se pokušavaju zamijeniti dva elementa genotipa, ali zamjena se radi samo ako je izgenerirani nasumični broj manji od faktora mutacije (N je veličina genotipa).

Operatori mutacije niza bitova:

- Invertirati bit na svakoj poziciji u genotipu s vjerojatnošću faktora mutacije.

Dakle, skup parametra genetskog algoritma ograničit će se na:

- veličina populacije

- faktor križanja
- faktor mutacije

```
1  generate initial population
2  while True do
3      put the best solution into the new population
4      while new population is not full do
5          pair = tournament(population)
6          solution = cross(pair)
7          solution = mutate(solution)
8          add solution into the new population
9      end
10     evaluate new population and store the best solution
11     population = new population
12 end
```

Algoritam 1: Genetski algoritam

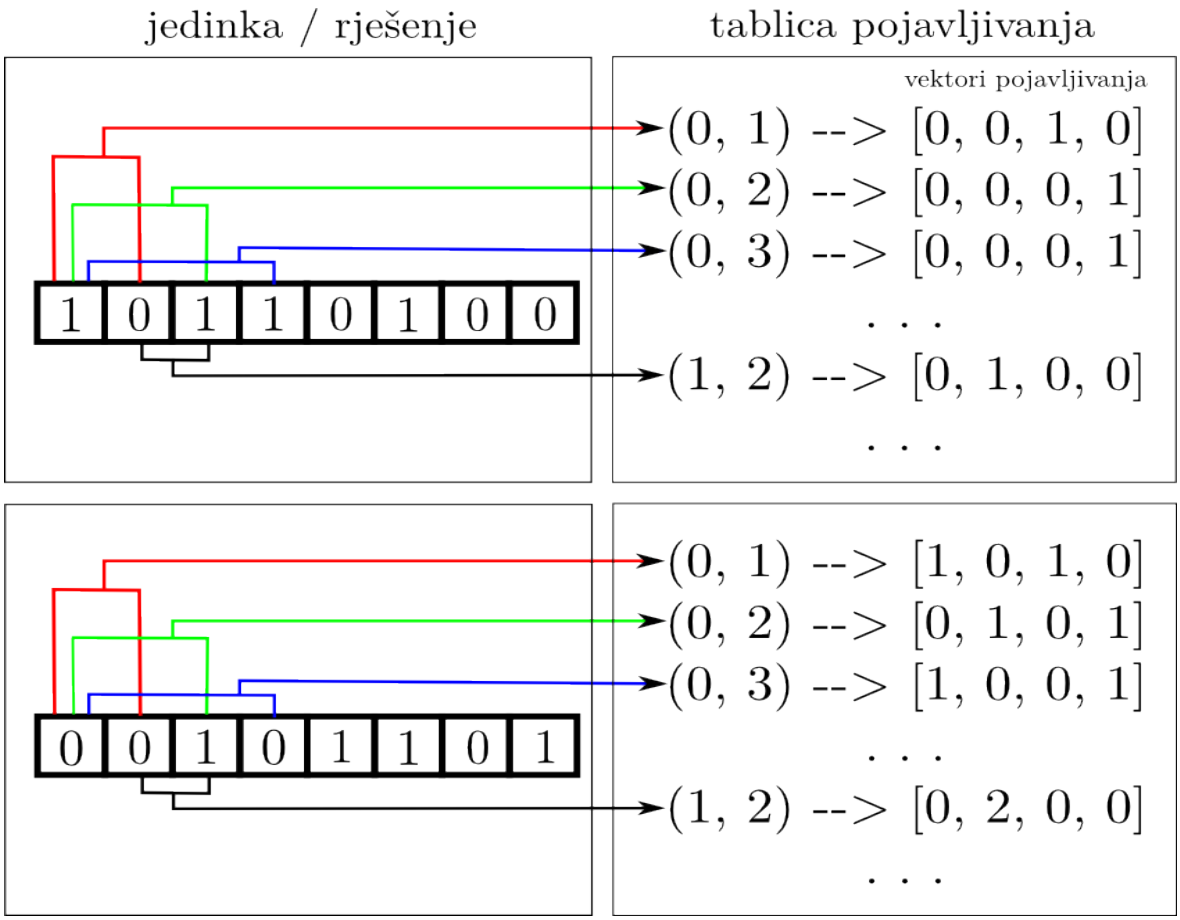
2.3. Genetski algoritam povezanog stabla - LTGA

LTGA (engl. *Linkage Tree Genetic Algorithm*) je algoritam iz porodice genetskih algoritama koji zadržava samo parametar veličine populacije. Samim time je jednostavnije prilagoditi algoritam za neki konkretan problem. U nastavku teksta će se detaljno opisati elementi algoritma.

2.3.1. Tablica pojavljivanja

Algoritam se temelji na računanju entropije pojavljivanja pojedinih bitova unutar populacije. Kako bi računanje entropije bilo moguće, potrebno je unutar populacije držati podatkovnu strukturu koja pamti koliko se puta pojedini bit, odnosno kombinacija bitova, unutar populacije pojavila. To se radi tako da svaki put kada u populaciju uđe novo rješenje, za svaki se par bitova tog rješenja ažurira njegovo pojavljivanje unutar tablice pojavljivanja. Točnije, za svaki par bitova unutar rješenja postoji vektor pojavljivanja veličine 4 (za 2 bitovne po-

zicije postoje 4 kombinacije bitova). Kada u populaciju uđe novo rješenje, za svaki se par bitova ažurira njegov vektor pojavljivanja. Primjerice, ako je unutar rješenja na poziciji 1 bit 0, a na poziciji 2 bit 1, tada će se unutar vektora pojavljivanja na poziciji $(1,2)$ povećati brojač na poziciji 1 (bitovna kombinacija 01 odgovara poziciji 1 unutar vektora pojavljivanja). Opisano je prikazano na slici 1. Složenost jednog ažuriranja tablice pojavljivanja je $O(N^2)$, gdje je N broj bitova u rješenju. Ekvivalentno bitovima, tablica pojavljivanja može pratiti i učestalost cijelih brojeva. Tada je veličina vektora pojavljivanja M^2, gdje je M broj različitih cijelih brojava unutar rješenja.



Slika 1: Izgradnja tablice pojavljivanja

2.3.2. Uzajamna informacija

Algoritam hijerarhijskog grupiranja koristi mjeru udaljenosti koja mjeri povezanost između grupa varijabli. Algoritam LTGA [3] koristi mjeru uzajamne informacije kako bi izračunao stablo povezanosti nad populacijom rješenja. Na temelju stabla povezanosti radi se grupiranje.

Neka je X_k diskretna varijabla sa gustoćom pojavljivanja $p(X_k)$; entropija je tada definirana kao:

$$H(X_k) = -\sum_i p_i(X_k) log(p_i(X_k)), \tag{2}$$

a uzajamna informacija I između skupa slučajnih varijabli definirana je kao:

$$I(X_1, ..., X_l) = \sum_{k=1}^{l} H(X_k) - H(X_1, ..., X_k). \tag{3}$$

Uzajamna informacija je posebno zanimljiva za uporabu u algoritmu hijerarhijskog grupiranja zbog toga što posjeduje zanimljivo svojstvo grupiranja. Svojstvo grupiranja kaže da je uzajamna informacija između tri grupe slučajnih varijabli C_1, C_2, C_3 ista kao suma uzajamne informacije između grupe C_1 i C_2 te uzajamne informacije između unije $C_1 \cup C_2$ i C_3:

$$I(C_1, C_2, C_3) = I(C_1, C_2) + I((C_1 \cup C_2), C_3). \tag{4}$$

Važno je shvatiti da je uzajamna informacija I mjera sličnosti između objekata, ali ona nije mjera udaljenosti. Algoritam hijerarhijskog grupiranja zahtijeva mjeru udaljenosti kako bi mogao izgraditi grupe varijabli. Mjera udaljenosti temeljena na uzajamnoj informaciji $d(X_1, X_2)$ je razlika između združene entropije (engl. *join entropy*) $H(X_1, X_2)$ i uzajamne informacije $I(X_1, X_2)$:

$$d(X_1, X_2) = H(X_1, X_2) - I(X_1, X_2). \tag{5}$$

U hijerarhijskom grupiranju grupiraju se grupe različite veličine, stoga je potrebno normalizirati mjeru udaljenosti $d(X_1, X_2)$. Normalizacija se radi tako da se $d(X_1, X_2)$ podijeli s ukupnom količinom informacije koja je predstavljena entropijom $H(X_1, X_2)$, stoga je normalizirana mjera udaljenosti između grupa X_1, X_2 predstavljena kao $D(X_1, X_2)$:

$$D(X_1, X_2) = 1 - \frac{I(X_1, X_2)}{H(X_1, X_2)} \tag{6}$$

$$D(X_1, X_2) = 2 - \frac{H(X_1) + H(X_2)}{H(X_1, X_2)} \tag{7}$$

$H(X_1)$ i $H(X_2)$ su entropije pojavljivanja svakog bita zasebno. Dakle, za neki se par bitova prati koliko se puta u tom paru pojavila 0 na mjestu 0 i 1 na mjestu 0, isto tako 0 na mjestu 1 i 1 na mjestu 1. $H(X_1) + H(X_2)$ se naziva neovisna entropija (engl. *separate entropy*). $H(X_1, X_2)$ je zajednička entropija (engl. *together entropy*), tj. entropija pojavljivanja svake kombinacije bitova koja prati koliko puta su se pojavile kombinacije 00, 01, 10 i 11.

2.3.3. Grupiranje

Algoritam UPGMA (engl. *Unweighted Pair Group Method with Arithmetic Mean*) koristi srednju vrijednost udaljenosti između grupa kako bi grupirao elemente. Izraz (8) opisuje udaljenost između grupa A i B:

$$d(A,B) = \frac{1}{|A| \cdot |B|} \sum_{x \in A} \sum_{y \in B} d(x,y) \tag{8}$$

Algoritam se može izvesti u složenosti $O(n^2)$ gdje je n broj negrupiranih elemenata. Pseudokod kreiranja grupa, počevši od negrupiranih elemenata, prikazan je u algoritmu 2.

```
1  unmerged ← {{0},{1},{2},...,{n − 1}}
2  useful ← unmerged
3  while |unmerged| > 1 do
4      C_i, C_j ← min_{C_i,C_j ∈ unmerged} D(C_i, C_j)
5      unmerged ← unmerged − {C_i, C_j} + {C_i ∪ C_j}
6      useful ← useful + {C_i ∪ C_j}
7      if D(C_i, C_j) == 0 then
8          useful ← useful − {C_i, C_j}
9      end
10 end
11 Order useful based on cluster size, smallest first.
12 Remove largest cluster from useful.
13 return useful
```

Algoritam 2: UPGMA kreiranje grupa

2.3.4. Baza algoritma

Glavni dio algoritma čini evolucija rješenja unutar populacije koja sadrži grupe gena s kojima se rješenje može križati. Pseudokod je prikazan u algoritmu 3. Na temelju izračunatih grupa unutar populacije vrši se križanje nasumično odabranih rješenja unutar populacije. Ako su oba djeteta bolja od roditelja, onda se djeca ubacuju u populaciju na mjesto svojih roditelja. Na kraju svake iteracije potrebno je izvršiti ponovno generiranje grupa.

```
1  while True do
2      generate population
3      clusters = linkage tree from population
4      for cluster in clusters do
5          take random two pair of solutions from population
6          swap genes from cluster
7          if if new pair is better than the old one then
8              replace the old pair with the new one
9              save best solution
10         end
11     end
12 end
```

Algoritam 3: LTGA

2.4. Neparametarski algoritam piramidalne populacije - P3

P3 (engl. *Parameter-less Population Pyramid*) [4] je neparametarski optimizacijski algoritam nastao na temelju algoritma LTGA. Prednost mu je što nema stalnu veličinu populacije, nego gradi piramidalnu populaciju neograničene veličine. Algoritmu za rješavanje nekog problema nije potrebno treniranje što je velika prednost u odnosu na ostale optimizacijske algoritme jer nije potrebno utrošiti vrijeme kako bi se podesili parametri algoritma. Kao što je već navedeno, algoritam gradi piramidalnu strukturu populacija. Dakle, postoji više populacija rješenja u algoritmu. Neko rješenje koje je optimirano pohlepnim (engl. *greedy*) algoritmom prvo se stavlja u populaciju na dnu piramide. Nakon toga se pokuša križati sa svakom populacijom rješenja unutar piramide. Ako se dobije bolje rješenje od trenutnog onda se ta jedinka stavlja u iduću populaciju unutar piramide. Kanonska implementacija algoritma radi nad nizom binarnih brojeva. U sklopu ovoga rada ta je implementacija proširena na skup N^+ brojava, kako bi se algoritam iskoristio u permutacijskim optimizacijskim problemima. Pseudokod algoritma P3 dan je u algoritmu 4. Bitni dijelovi algoritma su lokalno pretraživanje na početku (prije ubacivanja rješenja u piramidu) i križanje jedinke sa svakom razinom piramide. Svako novo generirano rješenje provjerava se sa već generiranim rješenjima i ako takvo rješenje već postoji ono se zanemaruje. Dakle, algoritam uz piramidalnu

populaciju posjeduje i skup rješenja koja su već nastala.

```
1  Create random solution
2  Apply hill climber
3  if solution ∉ hashset then
4      Add solution to P_0
5      Add solution to hashset
6  end
7  for allP_i ∈ pyramid do
8      Mix solution with P_i
9      if solution fitness has improved then
10         if solution ∉ hashset then
11             Add solution to the P_{i+1} and the hashset
12         end
13     end
14 end
```

Algoritam 4: P3

Valja još istaknuti križanje u algoritmu P3 prikazano u algoritmu 5. Za križanje se koriste grupe gena, ekvivalentno algoritmu LTGA. No samo križanje je nešto drugačije jer se pokušava napraviti križanje sa svim elementima unutar jedne populacije, elementima jedne razine piramide. Ukoliko se dobrota tako nastalog rješenja poveća onda se to uzima kao novo rješenje. U suprotnom se samo zanemaruje grupa gena koja se koristila tijekom križanja. Redoslijed jedinki za križanje je slučajan.

2.5. Kompaktni genetski algoritam - CGA

Kompaktni genetski algoritam (engl. *Compact Gentetic Algorithm*), u daljnjem tekstu CGA, je algoritam koji spada u grupu ED algoritama (engl. *Estimation of Distribution*). Implementacija algoritma je jednostavna. Pojednostavljeno, algoritam iz frekvencija pojavljivanja pojedinih gena u populaciji generira nove gene i na taj način nastaju nova rješenja. Jedini parametar algoritma je veličina populacije n, što je relativna prednost u odnosu na, primjerice, genetski algoritam koji ima veći skup parametara. Tekst programa algoritma pri-

```
1  for all C_i ∈ useful do
2      for all d ∈ shuffled(P_i) do
3          Copy d's gene values for C_i into solution
4          if solution has changed then
5              if solution's fitness decresed then
6                  Revert changes
7              end
8          end
9      end
10 end
```

Algoritam 5: P3 križanje

kazan je u programu 1. Najprije se inicijalizira vektor vjerojatnosti tako da se svaki element tog vektora postavi na vrijednost 0.5. Nakon toga se u svakoj iteraciji generiraju dva nova rješenja koja služe kako bi se podesio vektor vjerojatnosti na način da se uspoređuju geni između dva generirana rješenja. Ako se razlikuju i ako je gen u boljem rješenju 1, tada se vjerojatnost unutar vektora vjerojatnosti povećava za $1/n$; inače se smanjuje za $1/n$. Nova rješenja generiraju se u odnosu na vektor vjerojatnosti. Primjerice, ako je na poziciji dva unutar vektora vjerojatnosti broj 0.32 onda je vjerojatnost da će drugi element/gen unutar novog rješenja biti 1, upravo 0.32.

```
1  # compare: compare function        n: population size
2  # rand: random function            l: solution length
3  # iterations_no: iterations number p: probability vector
4
5  def generate(l, p, rand): # solution generator
6      return [1 if rand() < p[i] else 0 for i in range(l)]
7
8  def cga(n, l, compare, rand, iterations_no): # algorithm
9      # initialize probability vector
10     p = [0.5 for x in range(l)]
11     for iteration in range(iterations_no):
12         a, b = generate(p, l, rand), generate(p, l, rand)
13         winner, loser = compare(a, b)
14         # update probabilty vector
15         for i in range(l):
16             if winner[i] != loser[i]:
17                 if winner[i] == 1:
18                     p[i] += 1/n
```

```
19                  else:
20                      p[i] -= 1/n
21          best, = compare(winner, best)
22      return best
```

Tekst programa 1: Kompaktni genetski algoritam u programskom jeziku Python

U sklopu ovog rada konstruirana su dva algoritma po uzoru na P3, GA i CGA. Iz algoritma P3 preuzeta je ideja piramidalne populacije te su nastali piramidalni genetski algoritam i piramidalni kompaktni genetski algoritam. U nastavku će ti algoritmi biti detaljnije opisani.

2.6. Piramidalni genetski algoritam - PGA

PGA (engl. *Pyramid Gentetic Algorithm*) je varijanta genetskog algoritma koji nema veličinu populacije. Veličina populacije je teorijski beskonačna i čini piramidalnu strukturu. Inicijalno slučajno rješenje križa se s prvom razinom piramide te nakon toga mutira i ako se dobije bolje rješenje od prethodnog ono se ubacuje u iduću razinu piramide. Odabir jedinke s kojom se križa bit će k-turnirska selekcija. Pseudokod algoritma prikazan je u algoritmu 6. Inicijalno rješenje pokušava se križati sa svakom razinom unutar piramide i ako se dobije bolje rješenje onda se ono koristi u daljnjim križanjima. Svako nasumično rješenje se čuva unutar prve razine piramide, a svako se bolje rješenje čuva unutar neke više razine piramide. Piramidalnom populacijom otklonjen je samo parametar veličine populacije. Dakle parametri koje algoritam ima su: faktor mutacije i faktor križanja.

2.7. Piramidalni kompaktni genetski algoritam - PCGA

PCGA (engl. *Pyramid Compact Gentetic Algorithm*) je modifikacija algoritma CGA inspirirana algoritmom P3. Algoritam gradi piramidalnu strukturu u kojoj je svaka razina piramide jedna CGA populacija. Točnije, svaka razina piramide je jedan vektor vjerojatnosti iz kojeg se onda generiraju nova rješenja. Jedna iteracija algoritma sastoji se od toga da se inicijalno generira slučajno rješenje. To rješenje se uspoređuje s rješenjem dobivenim iz prve razine piramide. Bolje rješenje od ta dva se koristi za ažuriranje vektora vjerojatnosti prve razine i za ekvivalentan postupak u kojem se koristi naredna razina piramide.

Ovaj piramidalni algoritam je drugačiji u odnosu na ostale piramidalne algoritme utoliko što se ne pamte sva generirana rješenja. Populacija kod algoritma CGA je predstavljena

```
1  while True do
2      generate initial solution
3      for population in pyramid do
4          existing solution = tournamen(population)
5          new solution = cross(existing solution, solution)
6          mutate new solution
7          if new solution fitness >= solution fitness then
8              add new solution to next population inside pyramid
9              solution = new solution
10         end
11     end
12 end
```

Algoritam 6: PGA

vektorom vjerojatnosti. Kod PCGA algoritma postoji, u teoriji, beskonačan broj vektora vjerojatnosti. Dakle svaka razina piramide je jedan vektor vjerojatnosti. Što se ide više po piramidi to su parametri kojima se ažurira vektor vjerojatnosti manji, čime se postiže preciznije generiranje rješenja. Prva razina piramide ima parametar ažuriranja vektora vjerojatnosti $\frac{1}{4}$, a svaka sljedeća razina ima taj parametar dva puta manji.

U algoritmu 7 vidi se pseudokod algoritma PCGA. Prvo se generira slučajno rješenje, nakon toga se iz prve razine piramide također generira rješenje i s ta dva rješenja se onda ažurira vektor vjerojatnosti prve razine piramide. Još je bitno da ako je rješenje generirano iz prve razine piramide bolje od slučajno generiranog rješenja, ono se uzima kao inicijalno rješenje za drugu razinu piramide. Taj postupak se onda dalje ponavlja dok se ne dođe do vrha piramide. Nova razina piramide dodaje se samo ako smo u zadnjoj razini piramide dobili rješenje koje je bolje od prethodno generiranih rješenja. Na taj način usmjeravamo pretragu k preciznijem ažuriranju vektora vjerojatnosti.

2.8. Lokalna pretraga

Bitan segment optimizacijskih postupaka su algoritmi lokalne pretrage. U nastavku slijedi pregled algoritama lokalne pretrage koji su korišteni u "globalnim" optimizacijskim pos-

tupcima. Lokalna pretraga se u ovom slučaju koristi odmah nakon generiranja inicijalnog rješenja kako bi se poboljšalo inicijalno rješenje. Primjerice, P3 jako ovisi o ovom operatoru. Pretpostavka algoritma je da su inicijalna rješenje bolja od nasumično odbranih rješenja.

```
1  while True do
2      Generate random solution
3      for population in populations do
4          Generate candidate from population
5          winner, loser = compare(candidate, solution)
6          Update population probability vector
7          if winner.fitness > solution.fitness then
8              solution = winner
9              if top pyramid population then
10                 Add new population on top of pyramid
11             end
12         end
13     end
14 end
```

Algoritam 7: PCGA

2.8.1. Prvo poboljšanje

Algoritam prvog poboljšanja (engl. *first improvement*) se spominje u [4]. U svakoj iteraciji indeksi bitova se nasumično pomiješaju i nakon toga se svaki bit na izmiješanim indeksima mijenja sa inverznom vrijednosti. Ako dođe do poboljšanja rješenja, dozvoljava se još jedna iteracija algoritma, a ako ni za jedan indeks nema poboljšanja, algoritam se prekida. Algoritam ima zanimljivo svojstvo da za problem varljive zamke pronalazi rješenje kojem je jedan blok, ili više njih, postavljen na vrijednost koja daje najbolju dobrotu. Upravo je to svojstvo jedan od elemenata konvergencije algoritma P3 na problemu varljive zamke.

2.8.2. Binarni turnir

Ovaj algoritam se također spominje u sklopu [4]. Generiraju se dva nasumična rješenja i uzima se bolje rješenje. Ovo je dobar algoritam ukoliko se želi dobiti što veća pokrivenost prostora rješenja jer su oba generirana rješenja slučajna, a ipak postoji malo usmjeravanje prema boljim vrijednostima.

2.8.3. Blok traženje

Blok traženje je algoritam lokalne pretrage za permutacijski tip rješenja. Rješenje se dijeli u nepreklapajuće blokove i unutar tih blokova vrši se potpuno pretraživanje, a najbolji blokovi koriste se u izgradnji čitavog rješenja. Veličina bloka je parametar kojim se kontrolira prostor pretraživanja; za veličinu bloka jednaku veličini problema pretražio bi se čitav prostor rješenja.

2.8.4. Permutacijski mjehurić

Permutacijski mjehurić je algoritam lokalne pretrage za permutacijski tip rješenja. Nasumično se bira jedan element permutacije i taj se element mijenja sa svim ostalim elementima permutacije. Uzima se najbolje rješenje. Algoritam nalikuje na putovanje mjehurića: permutacijski element putuje s početka na kraj permutacije i traži najbolje mjesto.

3. Optimizacijski problemi

U ovom će poglavlju biti ukratko definirani optimizacijski problemi i postupc koji su se primijenili kako bi se problemi pripremili za primjenu u optimizacijskim algoritmima. Cilj je ispitati algoritme na raznim tipovima problema: diskretnim, kontinuiranim i realnim problemima iz inženjerske prakse.

3.1. Problem Boolean funkcija

Boolean funkcija koristi se u šifriranju tokova podataka kao jedini nelinearni element. Booleova funkcija f na $\mathbb{F}_2^n$ može se predstaviti kao tablica istinitosti u kojoj se svakom retku pridružuje vrijednost istine ili laži. Retci su poredani u leksikografskom poretku. Produkt dva vektora **a** i **b** označava se sa $\mathbf{a} \cdot \mathbf{b}$ i definiran je sa $\oplus_{i=1}^{n} a_i b_i$, gdje $\oplus$ predstavlja zbrajanje modulo 2 (XOR operaciju). Hammingova težina $HW(f)$, gdje je f Booleova funkcija, predstavlja broj jedinica u tablici istinitosti. Booleova funkcija je balansirana ako je Hammingova udaljenost dotične Booleove funkcije jednaka 2^{n-1}.

Walshova transformacija, izraz (9), predstavlja mjeru sličnosti između Booleove funkcije i linearne funkcije. Nelinearnost Booleove funkcije se pomoću Walshove trasformacije definira izrazom (10), a ograničenje nelinearnosti prikazano je u izrazu (11).

$$W_F(a) = \sum_{x \in F_2^n} (-1)^{f(x) \oplus a \cdot x} \tag{9}$$

$$N_f = 2^{n-1} - \frac{1}{2} max_{a \in F_2^n} |W_f(a)| \tag{10}$$

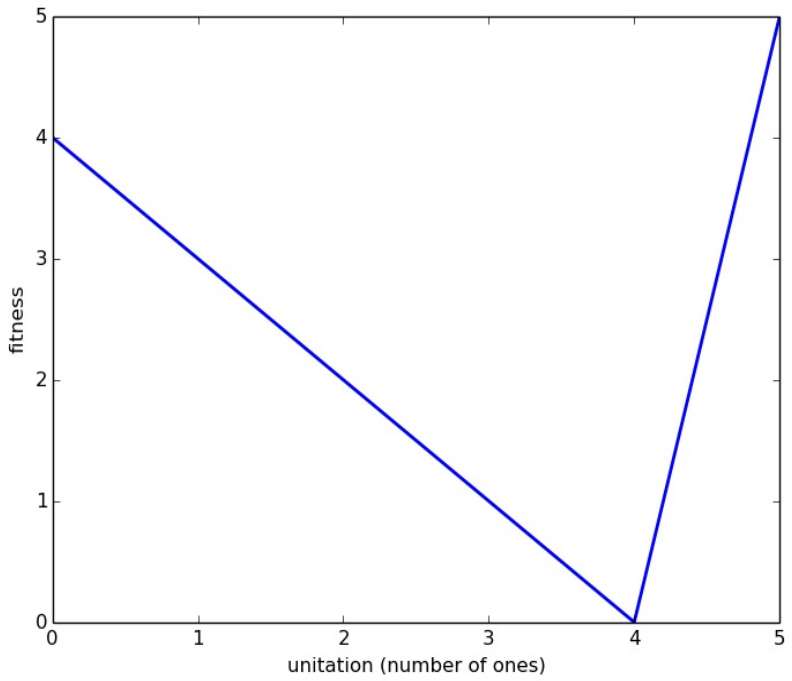
$$N_f \leq 2^{n-1} - 2^{\frac{n}{2}-1} - 2 \tag{11}$$

Prikaz rješenja za ovaj problem bit će niz bitova koji predstavlja vrijednosti unutar tablice istinitosti. Takvih vrijednosti ima 256, stoga je prostor pretraživanja 2^{256}. U sklopu ovog rada bit će korištena funkcija dobrote koja na vrijednost nelinearnosti dodaje 1, ako je Booleova funkcija balansirana, ili dodaje kaznu na nelinearnost ako je Booleova funkcija nebalansirana. Stoga je maksimalna dobrota za dani problem ograničena sa iznosom 119.

Ako se u obzir uzme samo mjera nelinearnosti, jer je to najvažniji kriterij, onda iz izraza (11) slijedi da je maksimalna nelinearnost 118. Još nije pronađena Booleanova funkcija s tim svojstvom. Zanimljivost ovog problema je što postoji mnogo rješenja koja imaju jednak iznos funkcije dobrote. Zbog toga je evolucijskom algoritmu teško izaći iz lokalnog optimuma jednom kad dođe do njega. Detaljnija analiza ovog problema može se pronaći u [13] i [14].

3.2. Varljiva zamka

U problemu varljive zamke (engl. *deceptive trap*) funkcija dobrote izgleda kao na slici 2. Dakle, dobrota ovisi o broju jedinica, no s porastom broja jedinica pada dobrota. Osim kada je broj jedinica maksimalan onda je i dobrota maksimalna. Primjerice, ako imamo pet bitova tada je dobrota za 0 bitova koji su jedan četiri i linearno pada s brojem jedinica. Osim kada je broj jedinica pet, onda je i dobrota pet. Standardni pohlepni algoritmi ovdje ne mogu pronaći rješenje jer odmah zapnu u lokalnom optimumu. Algoritmi će tražiti rješenje na problemu koji je sastavljen od više malih blokova.



Slika 2: Dobrota varljiva zamka s veličinom zamke 5 bitova

3.3. Varljiva stepenasta zamka

Varljiva stepenasta zamka (engl. *deceptive step trap*) je malo teža varijanta problema varljive zamke jer se funkcija dobrote konstruira tako da se nekoliko različitih rješenja stavi u istu vrijendnost funkcije dobrote što onda otežava pronalazak optimuma. Ravan dobrote (engl. *fitness plateus*) je područje jednakih dobrota i ono se konfigurira s parametrom *step*. U izrazu (12) prikazan je točan izračun dobrote.

$$stepTrap(t) = \left\lfloor \frac{(k - s) \quad (mod \quad s) + trap(t)}{s} \right\rfloor \tag{12}$$

3.4. Vodeće jedinice

Optimizacijski problem vodećih jedinica (engl. *leading ones*) je problem u kojemu treba maksimizirati broj bitova s početka niza bitova koji su postavljeni na vrijednost 1. Primjerice, dobrota niza bitova 11110100 je četiri jer su prva četiri bita postavljena na vrijednost 1. U sklopu algoritama dobrota će biti normalizirana na vrijednost od 0 do 1.

3.5. Maksimum jedinica

Optimizacijski problem maksimum jedinica (engl. *one max*) je problem u kojemu treba maksimizirati broj bitova postavljanih na 1 unutar nekog niza bitova. Primjerice za niz bitova 11110100 dobrota je pet, jer pet bitova ima vrijednost 1. U sklopu algoritama dobrota će biti normalizirana na vrijednost od 0 do 1.

3.6. Kapacitivni problem usmjeravanja vozila iz višebrojnih skladišta

Kapacitivni problem usmjeravanja vozila iz višebrojnih skladišta (engl. *Capacited Vehicle Routing Problem*), u daljnjem tekstu skraćeno CVRP, sastoji se od odabira skladišta i pronalaska Hamiltonovih ciklusa, tj. ruta koje minimiziraju težine u težinskom grafu, poštujući kapacitivna ograničenja ciklusa na način da su svi čvorovi posluženi. Cilj je pronaći skup skladišta koje je potrebno otvoriti i ruta koje je potrebno obići kako bi se minimizirao ukupni trošak usmjeravanja paketa do korisnika (trošak otvaranja skladišta, trošak pokretanja

vozila te trošak svih odabranih ruta).

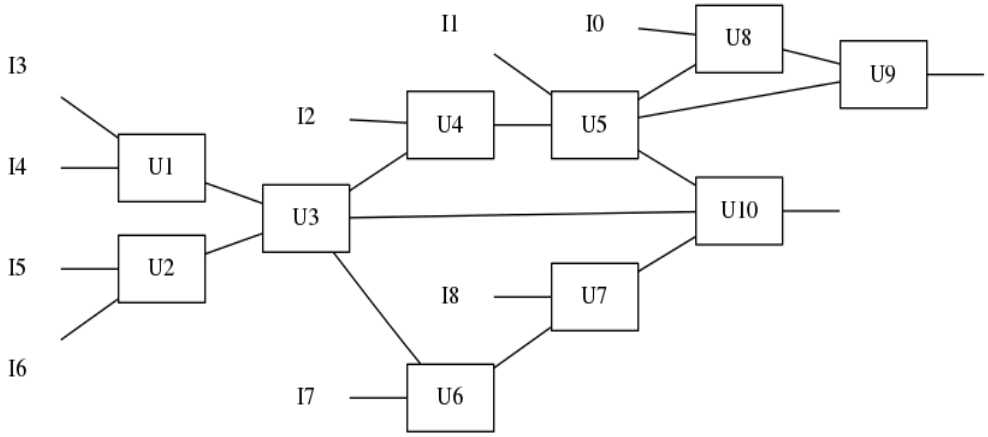
Kapacitivni problem usmjeravanja vozila iz višebrojnih skladišta zadan je kako slijedi. Zadan je usmjereni težinski graf $G = (V, A, C)$. Skup V čvorova se sastoji od podskupa I čvorova, veličine m, kao mogućih lokacija skladišta, i podskupa $J = \frac{V}{I}$ čvorova kao skupa korisnika. Podskup I čvorova predstavlja potencijalne početne i završne čvorove (skladišta). Ostali čvorovi predstavljaju korisnike koji imaju određenu potražnju veličine paketa. Težina grane $a = (i, j)$ iz skupa grana A je dana kao C_a. Za svako skladište $i \in I$ je dan kapacitet skladišta W_i i trošak otvaranja skladišta O_i. Svaki korisnik $j \in J$ potražuje d_j kapaciteta (veličine paketa). Dostupan je skup K identičnih vozila kapaciteta Q. Svako odabrano vozilo ima fiksni inicijalni trošak uporabe F i može izvesti samo jednu rutu, tj. obilazi podskup korisnika kojima isporučuje pakete te se vraća u pripadajući završni (početni) čvor i time radi Hamiltonov ciklus.

Instanca ovog problema ima 105 čvorova, 5 skladišta, 100 korisnika (u 2D kordinatnom sustavu), maksimalan kapacitet ciklusa (vozila) koji iznosi 70, te inicijalni trošak vozila koji iznosi 1000. Svako skladište sadrži podatke o poziciji skladišta u koordinatnom sustavu, kapacitet skladišta te trošak otvaranja skladišta. Svakom korisniku su pridijeljeni podaci o njegovoj poziciji u koordinatnom sustavu i traženi kapacitet robe. Zbrojem kapaciteta posjećenih korisnika u jednom ciklusu dobiva se vrijednost kapaciteta ciklusa pri čemu svaki ciklus mora imati manji ili jednak kapacitet od zadanog. Cijena puta između dva čvora računa se pomoću Euklidske udaljenosti. Opis problema preuzet je iz [17]. Vizualizacija problema vidi se na slici 32. Obojane točke predstavljaju skladišta, a crne točke predstavljaju korisnike. Iz slike je vidljivo da su korisnici relativno uniformno raspoređeni po cjelokupnom prostoru, dok skladišta nisu.

3.7. Kombinatorička mreža

Problem se sastoji u optimizaciji rasporeda bistabila unutar logičkog sklopa kako bi se dobila protočna struktura. Na slici 3 vidimo apstraktni prikaz jednog logičkog sklopa. Logički elementi, čiji naziv počinje s U, prikazani su u obliku pravokutnika, a s I su označeni ulazi u sklop. Potrebno je postaviti bistabile na veze između elemenata (na slici 3 su to linije) kako bi se dobila protočna struktura. Protočna struktura pretpostavlja da na svakom putu od ulaza do izlaza mora biti jednak broj bistabila. Primjerice, ako postoji točno jedan bistabil

na svakom putu, to znači da sklop ima dvorazinsku protočnu strukturu. Nadalje, ako postoje točno dva bistabila na svakom putu to znači da sklop ima trorazinsku protočnu strukturu.



Slika 3: Kombinatorička mreža

Ukupan broj mogućih kombinacija postavljanja bistabila je 2^n, gdje je n broj veza unutar logičkog sklopa. U realnom sklopu broj veza najmanje je reda veličine 10^2 što znači da pretraživanje grubom silom nije moguće.

3.7.1. Evaluator

Za logički sklop definirana je funkcija kazne koja se sastoji od dva dijela. Svaki element ima kašnjenje i ono utječe na ukupno kašnjenje sklopa. Kašnjenje na nekom putu zbroj je kašnjenja svih elemenata na tom putu. Ukupno kašnjenje sklopa određeno je s najvećim kašnjenjem na nekom putu. Ukoliko se na neki put postavi bistabil, onda se dobiva protočna struktura od dva dijela i kašnjenje tog puta je definirano većim kašnjenjem na nekom od ta dva dijela. Osim kašnjenja, funkcija kazne definirana je i brojem neispravnih puteva. Neispravni put je onaj put na kojem je krivi broj bistabila. Primjerice, ako se radi dvorazinska struktura tada je broj bistabila po svakom putu točno jedan. Ukoliko se broj bistabila na putu razlikuje od tog broja, u ukupnu kaznu dodaje se neki konstantni iznos kazne. U izrazu (13), C je konstanta i ona je eksperimentalno određena. Vrijednost konstante određena je na

vrijednost 1000.

$$cost = latency + C \cdot wrongPathsNumber \tag{13}$$

Evaluator je implementiran iterativno. U postupku predprocesiranja obilazak stabla izravnat je u listu kojom se kasnije iterira kako bi se izračunala ukupna kazna.

3.7.2. Algoritam ispravnih slučajnih rješenja

Budući da nasumična rješenja imaju veliku kaznu, potrebno je izgraditi algoritam koji daje dobra početna rješenja. Algoritam prilikom generiranja rješenja koristi podatkovnu strukturu koja za svaku vezu unutar sklopa zna reći koje su ostale veze (veze su dijelovi puta) pogođene. Ako se na neku vezu postavi bistabil onda to povlači da svi putevi koji imaju tu vezu imaju bistabil i na njih se više ne može staviti bistabil. U tekstu programa 2 vidi se cjelokupan proces generiranja rješenja. To rješenje je uvijek ispravno (ima točno jedan bistabil na svakom putu) i slučajno (svaki poziv metode za generiranje daje drugačije rješenje).

Na prikazani način dobiva se slučajno rješenje koje je valjano (ima točno jedan bistabil na svakom putu). Cilj generiranja valjanih početnih rješenja je da neki globalni optimizacijski algoritam [1] ima dobru početnu točku kako bi mogao u realnom vremenu unaprijediti rješenje. Složenost algoritma je $O(N^2 \cdot M)$ gdje je M broj grana, a N broj veza između elemenata.

```
1 def build_solution(self, length):
2
3   # build pipeline solution
4   solution = [0] * length
5   affected_links = set()
6
7   for branch in self.paths:
8     set_branch = set(branch)
9     diff = list(set_branch.difference(affected_links))
10
11    # diff is whare bistabil can be placed
12    if len(diff) > 0:
13      random.shuffle(diff)
14
15      for random_elem in diff:
16        set_link = self.link2link[random_elem]
```

[1] primjerice genetski algoritam ili P3 algoritam

```
17          affected_links = affected_links.union(set_link)
18          solution[random_elem] = 1
19          affected_links.union(set_link)
20          break
21
22      return solution
```

Tekst programa 2: Algoritam ispravnih slučajnih rješenja za problem kombinatoričke mreže

3.8. Kontinuirana optimizacija

Budući da algoritmi u ovom radu koriste bitovni prikaz rješenja, za evaluaciju kontinuirane funkcije korišteno je kodiranje Grayevim kodom. Kako bi se evaluacija brže računala, vrijednosti funkcije za pojedini Grayev kod spremljene su u priručnu memoriju iz koje se onda dohvaćaju prilikom evaluacije. Za potrebe ispitivanja odabrane su kontinuirane funkcije za koje je lagano izgenerirati vrijednosti koje će se pospremiti. Izabrane su funkcije kojima su varijable međusobno nezavisne (separabilne) kako ne bi došlo do kombinatorne eksplozije prilikom spremanja izračunatih vrijednosti.

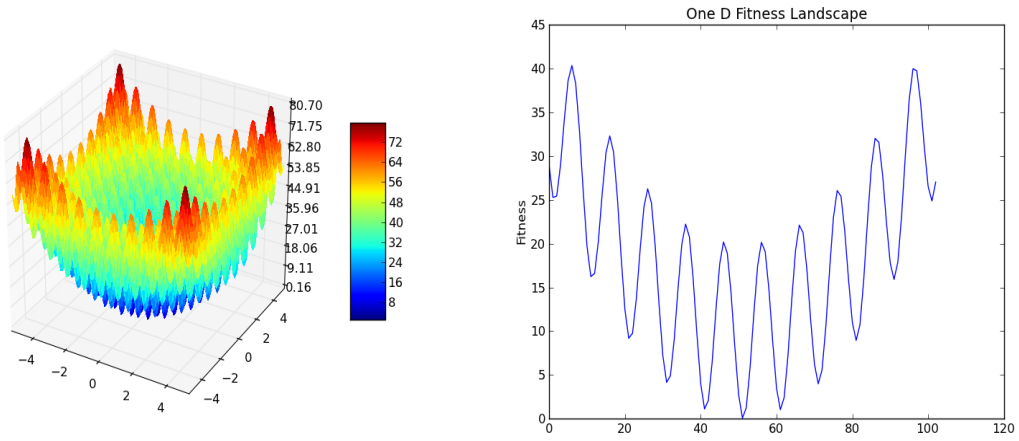
U ovom radu jedna kontinuirana varijabla kodirat će se sa 20 bitova ($\sim 10^6$ različitih vrijednosti po varijabli). Razlučivost ovisi o konkretnom problemu. Primjerice, za Rastrigin problem raspon vrijednosti jedne varijable je od -5.12 do 5.12 te je razlučivost $9.766 \cdot 10^{-6}$. Za ostale probleme vrijedi ekvivalentan račun.

Sve slike u ovom poglavlju preuzete su iz [10].

3.8.1. Funkcija Rastrigin

Funkcija Rastrigin je multimodalna kontinuirana funkcija definirana izrazom (14). Na slici 4 vidi se vizualizacija Rastrigin funkcije.

$$\begin{gathered} f(x_1 \cdots x_n) = 10n + \sum_{i=1}^{n} (x_i^2 - 10cos(2\pi x_i)) \\ -5.12 \leq x_i \leq 5.12 \\ \text{minimum u } f(0, \cdots, 0) = 0 \end{gathered} \tag{14}$$

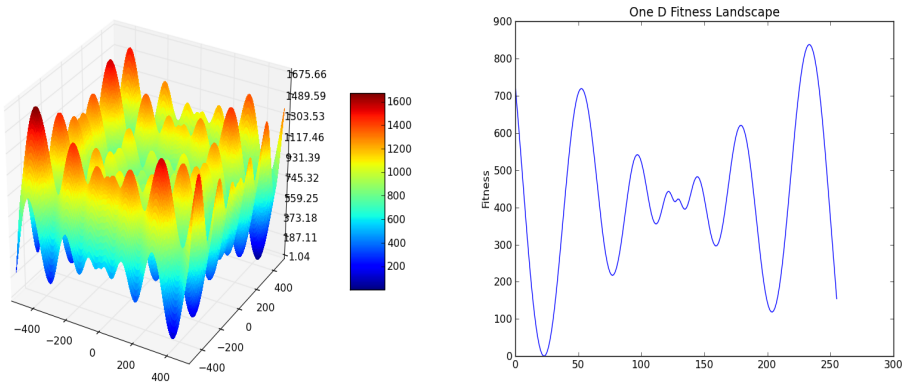


Slika 4: Prikaz Rastrigin funkcije

3.8.2. Funkcija Schwefel

Funkcija Schwefel je multimodalna i asimetrična kontinuirana funkcija definirana izrazom (15). Na slici 5 vidi se vizualizacija Schwefel funkcije.

$$
\begin{gathered}
f(x_1 \cdots x_n) = \sum_{i=1}^{n} (-x_i sin(\sqrt{|x_i|})) + \alpha \cdot n \\
\alpha = 418.982887 \\
-512 \leq x_i \leq 512 \\
\text{minimum u } f(420.968746, 420.968746, \cdots, 420.968746) = 0
\end{gathered}
\tag{15}
$$

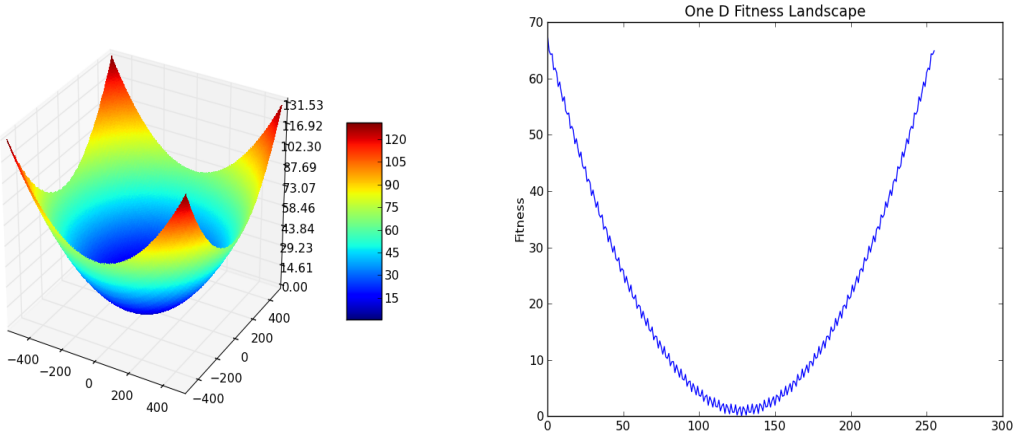


Slika 5: Prikaz Schwefel funkcije

3.8.3. Funkcija Griewank

Funkcija Griewank je multimodalna kontinuirana funkcija definirana izrazom (16). Na slici 6 vidi se vizualizacija Griewank funkcije.

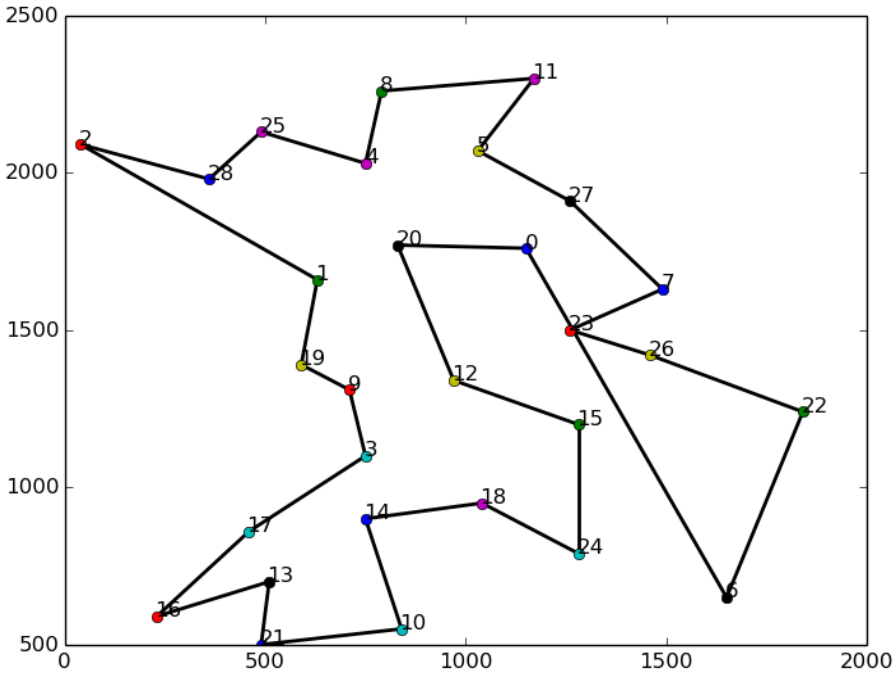
$$f(x_1 \cdots x_n) = 1 + \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} cos(\frac{x_i}{\sqrt{i}})$$
$$-512 \leq x_i \leq 512 \tag{16}$$
$$\text{minimum u} f(0, \cdots, 0) = 0$$



Slika 6: Prikaz Griewank funkcije

3.9. Problem trgovačkog putnika

U problemu trgovačkog putnika (engl. *travelling salesman problem*) potrebno je jednom obići sve gradove točno jednom uz minimalan trošak puta. Točnije, potrebno je naći minimalan Hamiltonski ciklus u neusmjerenom grafu. Složenost problema uz primjenu grube sile je $O(n!)$, a uz primjenu dinamičkog programiranja $O(2^n)$. U ovom radu algoritmi će biti ispitani na dva standardna TSP problema: Bays 29 i Oliver 30. Na slici 7 vidi se 2D prikaz Bays 29 problema i jedno predloženo rješenje.



Slika 7: Bays 29 raspored gradova

3.10. Optimizacija parametara

Budući da neki algoritmi posjeduju određen skup parametara, taj skup parametara potrebno je optimizirati kako bi algoritam bolje pretraživao prostor rješenja problema. Prednost neparametarskih algoritama je to što se postupak optimizacije parametara ne mora provoditi (odmah su spremni za primjenu na nekom konkretnom problemu).

Rešetkasto pretraživanje (engl. *grid search*) je standardni postupak za pretraživanje prostora rješenja, no karakterizira ga kombinatorna eksplozija u slučaju da imamo veliki skup parametara. U ovom se radu pretraživanje skupa parametara neće raditi rešetkastim pretraživanjem, nego pretraživanjem po svakom parametru. Primjerice, prvo se pretraže vrijednosti iz skupa nekog od parametara uz fiksirane vrijednosti ostalih parametara. Kada se odredi optimalna vrijednost za dotični parametar, njegova se vrijednost fiksira i prelazi se na ostale parametre. Za svaki skup parametara pokus se ponavlja $N = 10$ puta i uzima se medijan najboljih dobrota kao najbolja vrijednost za taj skup parametara. Izvršavanje jedne iteracije za neki skup parametara traje do maksimalnog broja evaluacija, gdje je taj broj posebno određen za svaki algoritam i problem na način da se iz grafa dobrote u ovisnosti o broju evaluacija očita broj evaluacija nakon kojeg dobrota počne stagnirati.

4. Rezultati

4.1. Optimizacija parametara

Ukoliko algoritam posjeduje neki parametar, taj parametar se birao na logaritamskoj skali, osim ako to nije bilo moguće iz vremenskih razloga (predugo trajanje podešavanja parametara). Za neku kombinaciju parametara algoritam se na danom problemu izvodio 10 puta te ja kao najbolja vrijednost izabran medijan svih dobivenih vrijednosti. Među parametrima prvo se traži optimalna veličina bloka, nakon toga veličina populacije pa faktora mutacije i naposlijetku faktora križanja.

4.1.1. GA

Parametri su traženi u skupu: veličina populacije - [10, 15, 20, 25, 35, 45, 50], faktor mutacije - [0.05, 0.10, 0.15, 0.20, 0. 25, 0.25, 0.40], faktor križanja - [0.10, 0.20, 0.30, 0.40, 0.50, 0.60, 0.70] i veličina bloka za permutacijske probleme - [2, 3, 4, 5].

problem	veličina populacije	faktor križanja	faktor mutacije	blok
Maksimum jedinica	10	0.10	0.05	-
Vodeće jedinice	10	0.70	0.05	-
Varljiva zamka	20	0.2	0.05	-
Varljiva stepenasta zamka	20	0.1	0.25	-
Rastrigin	25	0.70	0.05	-
Schwefel	20	0.40	0.05	-
Bays 29	10	0.6	0.05	5
Oliver 30	15	0.30	0.05	4

Tablica 1: Optimalni parametri za GA

4.1.2. CGA

Jedini parametar algoritma je veličina populacije. Parametar je tražen u skupu: [5, 10, 20, 40, 80, 150, 300, 600, 1200].

problem	veličina populacije
Maksimum jedinica	40
Vodeće jedinice	80
Varljiva zamka	5
Varljiva stepenasta zamka	5
Rastrigin	150
Schwefel	1200

Tablica 2: Optimalni parametri za CGA

4.1.3. LTGA

Parametar veličine populacije biran je iz skupa: [10, 20, 30, 40, 50].

problem	veličina populacije
Maksimum jedinica	50
Vodeće jedinice	30
Varljiva zamka	20
Varljiva stepenasta zamka	30
Rastrigin	40
Schwefel	40
Bays 29	10
Oliver 30	10

Tablica 3: Optimalni parametri za LTGA

4.1.4. PGA

Parametri su traženi u skupu: faktor mutacije - [0.05, 0.10, 0.15, 0.20, 0. 25, 0.25, 0.40], faktor križanja - [0.10, 0.20, 0.30, 0.40, 0.50, 0.60, 0.70].

problem	faktor križanja	faktor mutacije
Maksimum jedinica	0.20	0.05
Vodeće jedinice	0.60	0.05
Varljiva zamka	0.10	0.05
Varljiva stepenasta zamka	0.20	0.05
Rastrigin	0.50	0.05
Schwefel	0.20	0.15
Bays 29	0.20	0.15
Oliver 30	0.04	0.15

Tablica 4: Optimalni parametri za PGA

Navedeni parametri će biti korišteni u daljnjim usporedbama između algoritama.

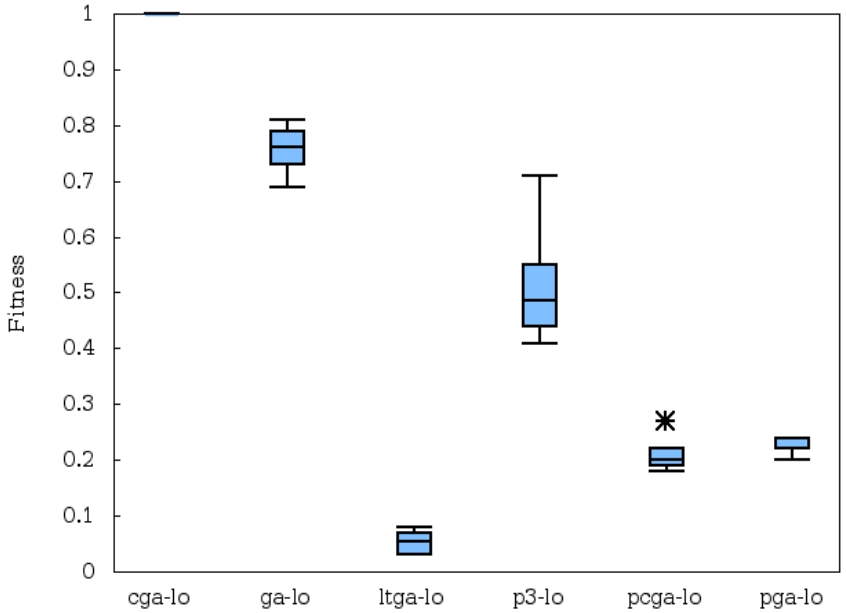
4.2. Dobrota u ovisnosti o broju evaluacija

U ovom će dijelu biti prikazani rezultati izvođenja algoritama uz ograničen broj evaluacija. Ograničenje iznosi 10^4 evaluacija, što nije mnogo. Cilj je samo da se usporede algoritmi po tom kriteriju. Razlog takvom broju evaluacija je i složenost samog pokretanja svih pokusa jer ukupno treba pokrenuti $10 \cdot 8 \cdot 6 = 480$ pokusa (10 pokretanja za svaki algoritam i problem, 8 problema, 6 algoritama). Kod permutacijskih problema za algoritam lokalne pretrage koristi se blok traženje s veličinom bloka 3. Taj izbor se eksperimentalno pokazao kao najbolji, a ovdje nije cilj pokazati ovisnost o tom parametru, nego samo o evaluacijskom ograničenju.

4.2.1. Vodeće jedinice

Instanca problema je veličine 100 bitova. Na slici 8 vidi se kako algoritam CGA daje najbolje rezultate za dani problem. CGA spada u grupu jednostavnih algoritama, čime se

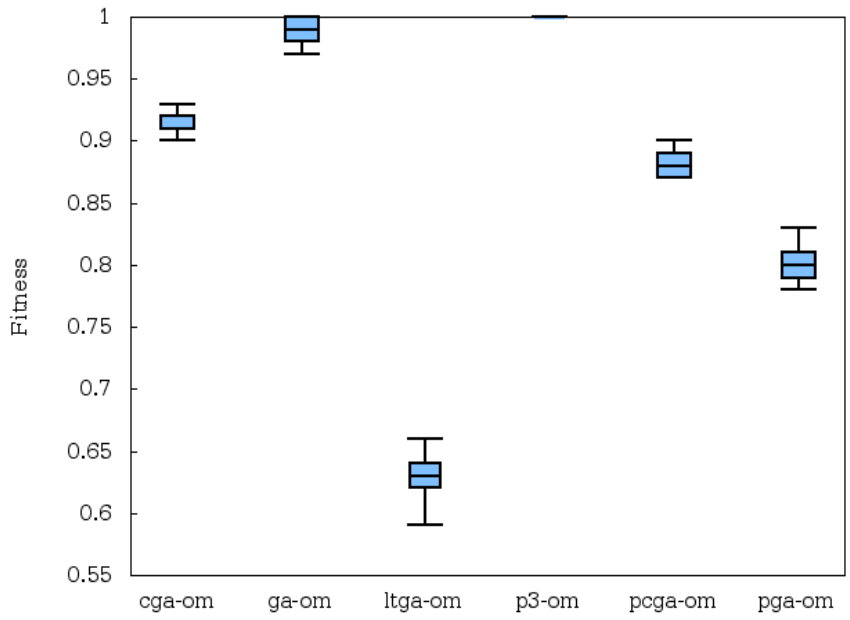
pokazuje da i takve algoritme ima smisla koristiti. Najlošiji je LTGA, a loše su i piramidalne verzije GA i CGA. Razlog tomu može biti to što su parametri za GA i CGA puno bolje pogođeni, a u piramidalnoj verziji prilagodba na problem se nije dogodila na vrijeme. Algoritmi u ovom pokusu, osim algoritma P3, nemaju algoritme lokalne pretrage. No, algoritam lokalne pretrage kod algoritma P3 nije prilagođen za ovakav tip problema. Dakle, niti jedan algoritam ne koristi neku loklanu pretragu koja bi trivijalno riješila dani problem.



Slika 8: Boxplot problema vodećih jedinica

4.2.2. Maksimum jedinica

Instanca problema je veličine 100 bitova. Na slici 9 vidi se kako je P3 najbolji za dani problem, no to proizlazi iz činjenice da P3 ima posebno dizajniran operator lokalne pretrage koja praktički odmah nalazi optimum. Vidi se također da su CGA i PCGA dosta slični po dobivenom rezultatu, dok PGA dosta zaostaje za GA. LTGA i dalje daje najlošiji rezultat, što je neočekivano jer bi za ovaj problem rezultat trebao biti sličan rezultatu algoritma P3.

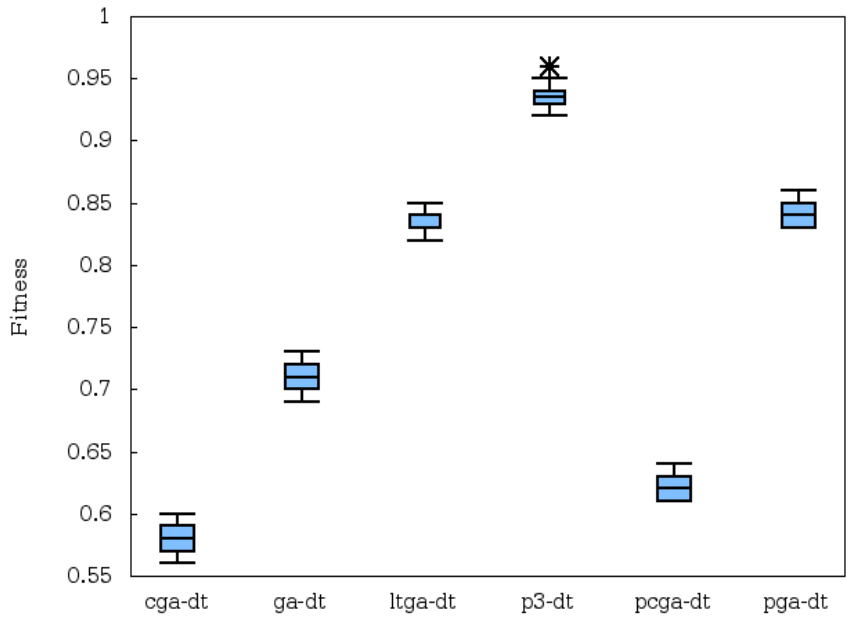


Slika 9: Boxplot problema maksimuma jedinica

Problemi kao što su vodeće jedinice i maksimum jedinica su dobri iz razloga što su jednostavni i na njima se može vidjeti kako će algoritmi raditi. Međutim, problem je što se zbog jednostavnosti lako može dogoditi da neki algoritam trivijalno riješi problem, a tada to nije dobra mjera. Konkretno ovdje je to slučaj s algoritmom P3.

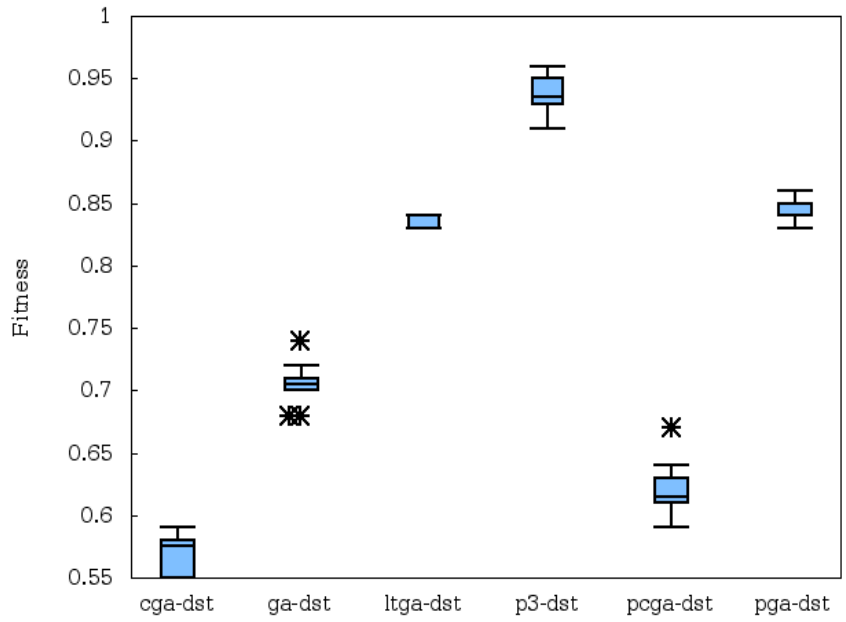
4.2.3. Varljiva zamka i varljiva stepenasta zamka

Instanca problema varljive zamke veličine je 100 bitova uz veličinu zamke od 5 bitova. Ovaj problem teži je od maksimuma jedinica i vodećih jedinica jer operatori lokalne pretrage ne mogu trivijalno riješiti problem. Algoritam P3 se opet pokazao kao najbolji jer ima dobro dizajniran operator lokalne pretrage. Taj algoritam koristi algoritam prvog poboljšanja koji nalazi optimume po dijelovima. To znači da u nizu od, primjerice, 100 bitova koji ima 20 blokova po 5 bitova, nađe optimum u nekoliko nasumično odabranih blokova. Kasnije se ti blokovi združuju u piramidalnoj evoluciji. Najlošiji je algoritam CGA. Pozitivno je što PCGA daje bolja rješenja nego CGA. Isto tako, to vrijedi i za odnos GA i PGA.



Slika 10: Boxplot varljive zamke

Instanca problema varljive stepenaste zamke veličine je 100 bitova uz veličinu zamke 5 i veličinu koraka 2. P3 opet daje najbolji rezultat, a CGA najlošiji. Iz grafa 11 se ovaj put lijepo vidi da je ovaj problem za nijansu teži od varljive zamke jer ima više vrijednosti koje odstupaju (na grafu je vrijednost koja odstupa označena sa znakom *).



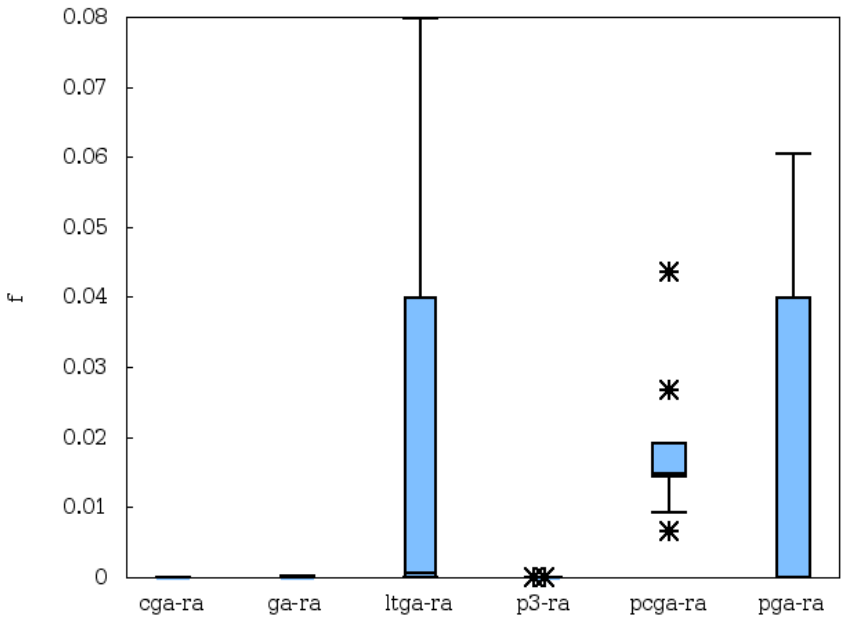
Slika 11: Boxplot varljive stepenaste zamke

Piramidalna populacija poboljšava karakteristike algoritama i to kod sva tri analizirana

algoritma. Razlog tomu bi mogao biti što algoritmi sa piramidalnom populacijom posjeduju veće znanje o prethodnim rješenjima. U ovom problemu to dolazi do izražaja zato što se rješenje mora izgraditi po dijelovima, a ako se pamti više rješenja veća je vjerojatnost križanja kojim će se dobiti bolje rješenje.

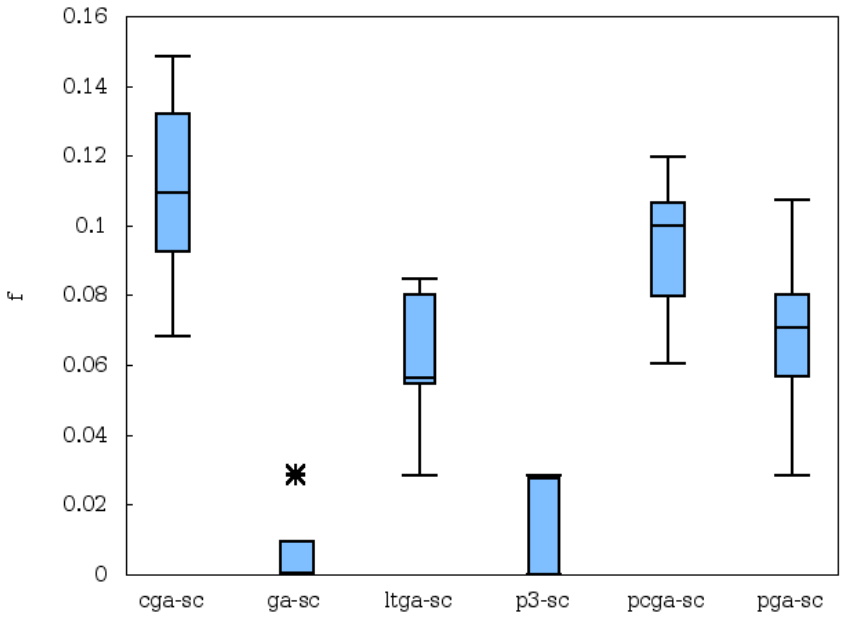
4.2.4. Rastrigin i Schwefel

Instanca problema Rastrigin i Schwefel veličine su 100 bitova uz 20 bitova po varijabli, dakle, ukupno ima 5 varijabli. Na grafovima vezanima uz kontinuirane funkcije vrijednost je na y osi normalizirana na interval od 0 do 1. Valja obratiti pozornost da se kod kontinuiranih funkcija u ovom radu govori o minimizaciji vrijednosti funkcije. Opet vrijedi istaknuti algoritam CGA koji daje najbolji rezultat iako je daleko najjednostavniji. Na slici 12 vidimo kako je LTGA opet najlošiji. U problemu Rastrigin za razliku od problema varljive zamke, piramidalna populacija ne pridonosi dobivanju boljih rješenja. Razlog tomu bi mogao biti što Rastrigin nema toliko izraženu udaljenost između lokalnih optimuma. Postoji mnogo lokalnih optimuma, ali oni su relativno blizu (lakše se kretati po lokalnim optimumima), stoga križanje ne pridonosi rješavanju problema koliko mutacija.



Slika 12: Rastrigin boxplot

Za problem Shwefel na slici 13 GA daje najbolje rješenja, blizu mu je P3, no ovaj put LTGA nije najlošiji. Najlošiji je algoritam CGA.

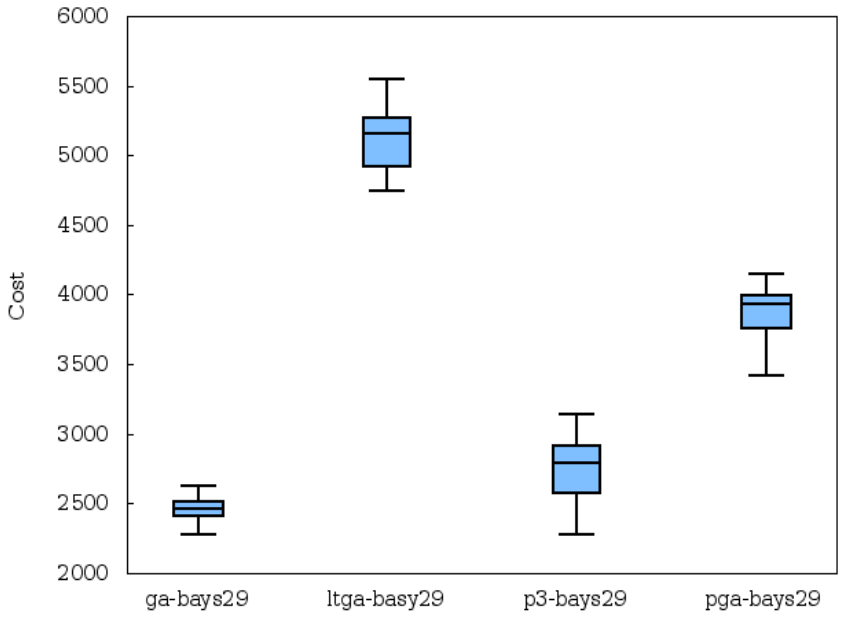


Slika 13: Schwefel boxplot

4.2.5. Bays 29 i Oliver 30

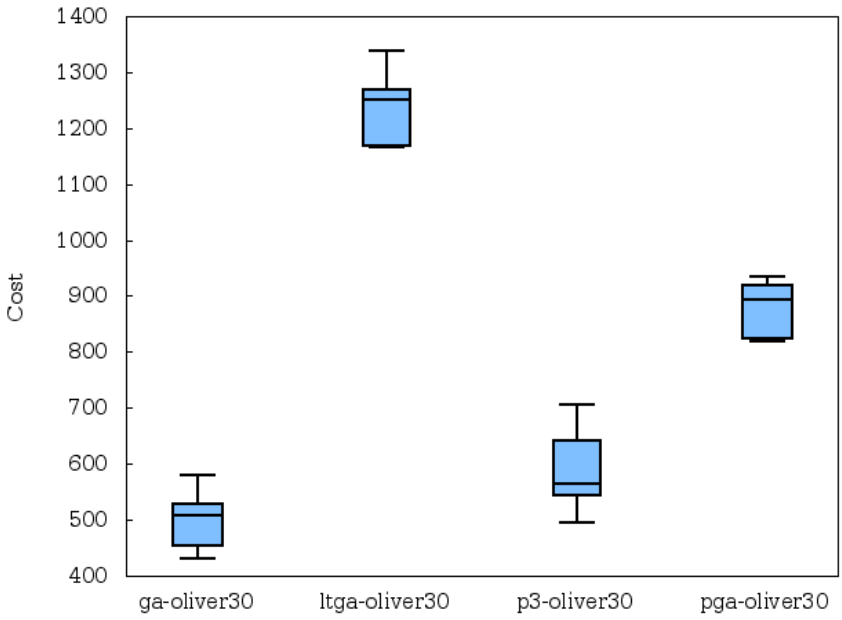
Na slici 14 vidi se boxplot prikaz iznosa troška. Za svaki algoritam pokus je pokretan 10 puta s ograničenjem broja evaluacija na 10^6. Bays 29 ukupno ima 29! mogućih puteva. Iz grafa se vidi kako GA i P3 daleko nadmašuju LTGA i PGA. GA je najbolji jer se on već dugo koristi kao alat za rješavanje permutacijskih problema. P3 je relativno nov algoritam i tek ga je potrebno testirati za permutacijske probleme i otkriti nove operatore za efikasno rješavanje problema te vrste, no definitivno ima smisla koristiti taj pristup za rješavanje permutacijskih problema.

Problem Oliver 30 ima ukupno 30! mogućih rješenja, što znači da je za nijansu teži od problema Bays 29. No iz slike 15 vide se jako slični rezultati kao za problem Bays 29 što je očekivano. GA opet daje najbolja rješenja, LTGA radi loše, P3 je blizu GA, a PGA dosta zaostaje za GA i P3, ali je bolji od LTGA.



Slika 14: Boxplot za permutacijski problem Bays 29

Iz danih pokusa s ograničenim brojem evaluacija, može se zaključiti da je upitno koliko se isplati koristiti algoritam LTGA. P3 je jednostavno puno bolji algoritam, a zadržava sva svojstva koja ima LTGA. Važno je i da ima smisla koristiti piramidalne populacije jer kod nekih problema značajno pridonose dobivanju boljih rješenja.



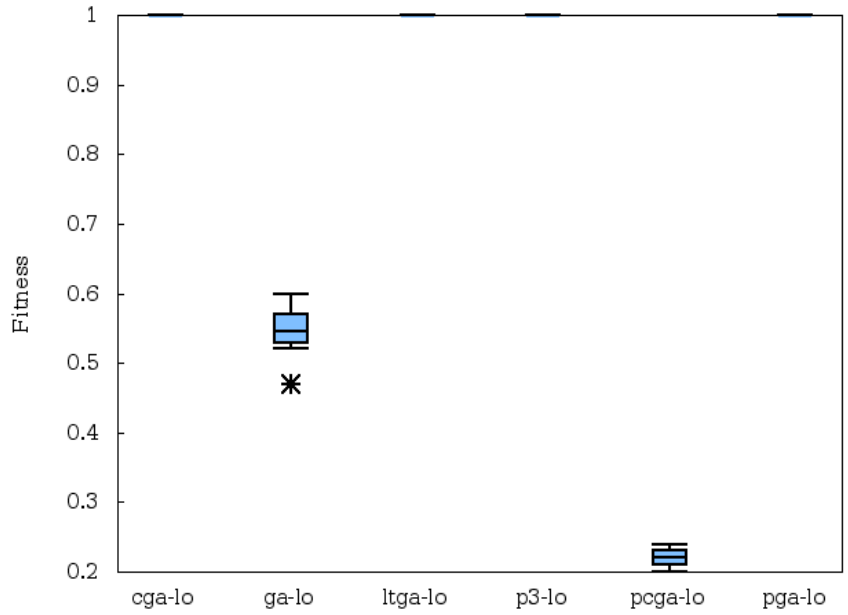
Slika 15: Boxplot za Oliver 30 permutacijski problem

4.3. Dobrota u ovisnosti o vremenskom ograničenju

U ovom dijelu cilj je pokazati kako rade algoritmi uzevši u obzir vremensko ograničenje. Ograničenje je postavljeno na 10 s, što je relativno kratko vrijeme za probleme koji se pokušavaju optimirati.

4.3.1. Vodeće jedinice

Instanca problema veličine je 100 bitova. U ovom pokusu čak četiri od šest algoritama su u vremenskom ograničenju došli do najboljeg rješenja i to za svako pokretanje. GA je ovaj put podbacio kao i PCGA (slično kao i u pokusu sa fiksnim brojem evaluacija). GA bi barem trebao biti blizu ostalih algoritama, ali očito nije toliko dobar za ovaj problem. To bi se trivijalno riješilo s nekim specifičnim operatorom lokalne pretrage.

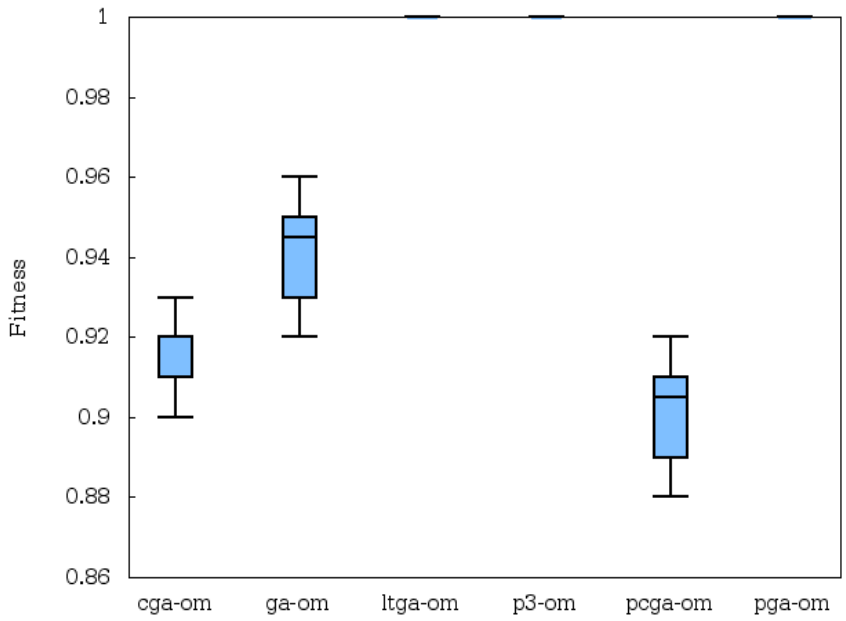


Slika 16: Boxplot vodećih jedinica s ograničenjem 10s

4.3.2. Maksimum jedinica

Instanca problema veličine je 100 bitova. Za razliku od slike 16, na slici 17 jedino CGA odstupa rezultatom. Iz toga se da zaključiti da CGA može biti izrazito dobar ili izrazito loš, a to je upravo zbog njegove jednostavnosti. Karakteristika CGA je što nema puno prostora za ugradnju nekakvih operatora lokalne pretrage. Naizgled se to čini kao nedostatak, ali

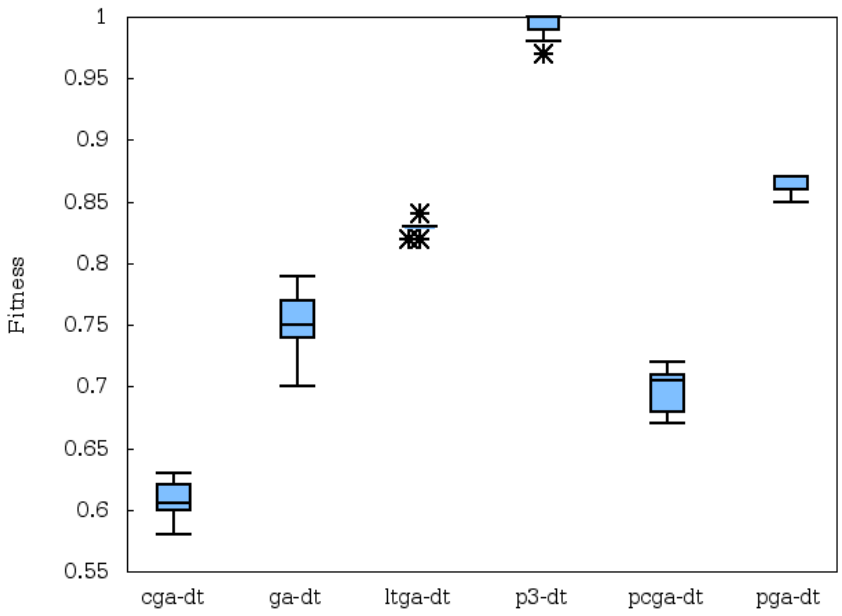
ako se bolje razmisli, takvo svojstvo može biti prednost jer se lako može isprobati radi li taj algoritam za dani problem. Ako ne radi, nema ga smisla koristiti, ali ako radi može ga se iskoristiti da brzo pretražuje prostor pretraživanja. Jedini parametar algoritma CGA je veličina populacije i njega se prema svemu prikazanom može optimizirati za dani problem ili iskoristiti kao parametar kojim se kontrolira brzina konvergencije. Kod GA takvih parametra ima više i njih je onda teže ugoditi.



Slika 17: Boxplot maksimuma jedinica s vremenskim ograničenjem 10s

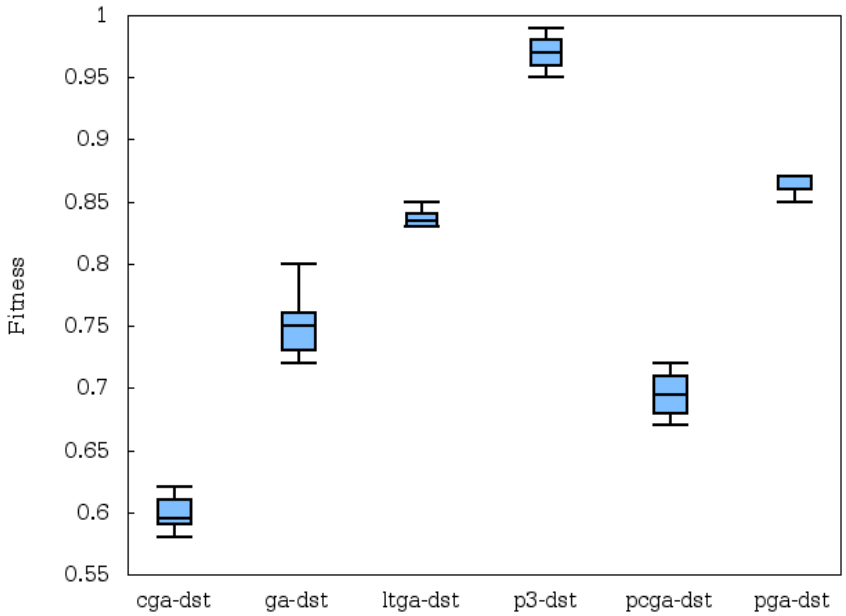
4.3.3. Varljiva zamka i varljiva stepenasta zamka

Instanca problema varljive zamke veličine je 100 bitova uz veličinu zamke od 5 bitova. P3 daje najbolje rezultate kao i za varljivu zamku sa ograničenim brojem evaluacija. Opet se vidi iz slike 18 da algoritmi sa piramidalnim populacijama imaju prednost na ovom problemu od onih koji to nemaju.



Slika 18: Boxplot varljive zamke s vremenskim ograničenjem 10s

Instanca problema varljive stepenste zamke veličine je 100 bitova uz veličinu zamke 5 i veličinu koraka 2. Rezultati su ekvivalentni prethodnom pokusu, no nema vrijednosti koje odstupaju što može biti posljedica vremenskog ograničenja koje ne dozvoljava da se dogodi odstupanje. Opet vrijedi istaknuti da piramidalna populacija poboljšava rezultat u ovakvom tipu problema.

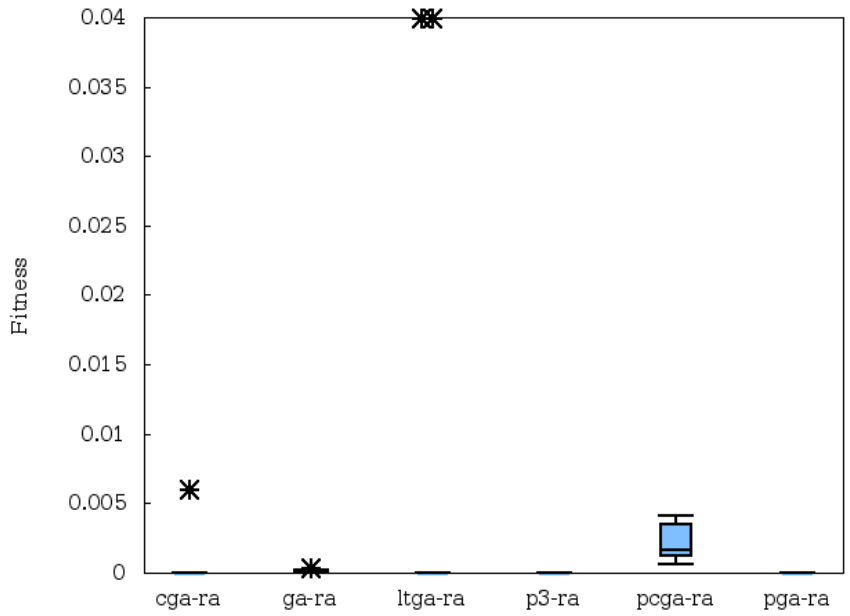


Slika 19: Boxplot varljive stepenaste zamke s vremenskim ograničenjem 10s

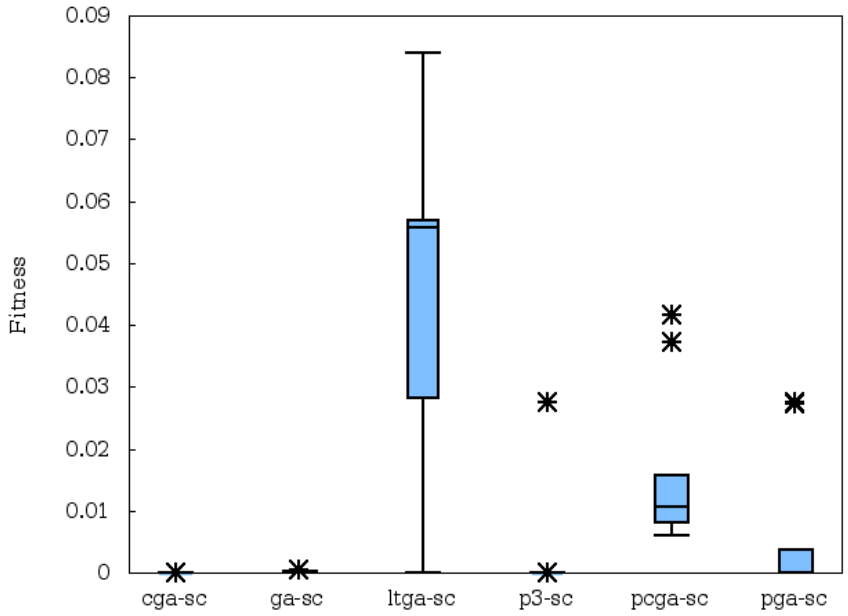
4.3.4. Rastrigin i Schwefel

Instance problema Rastrigin i Schwefel veličine su 100 bitova uz 20 bitova po varijabli, dakle, ukupno ima 5 varijabli. Kod problema Rastrigin rezultat je očekivan, jedino LTGA nekad daje rješenje koje odstupa više od ostalih. PCGA u globalu daje nešto lošija rješenja. PGA se pokazao dosta dobar za ovaj tip problema. CGA i GA izvršavanja su završila s vrijednostima koje odstupaju, što je očekivano jer nemaju nikakav operator lokalne pretrage koji bi više kontrolirao dobivena rješenja.

Na slici 21 LTGA opet radi daleko najlošije, no graf je prepun vrijednosti koje odstupaju što još jednom potvrđuje da je problem Schwefel teži od problema Rastrigin. Među najboljim algoritmima je opet CGA. Piramidalne verzije u ovom slučaju zaostaju za nepiramidalnim verzijama algoritama, osim u slučaju algoritma P3, ali isto vrijedi za većinu izvedenih pokusa.



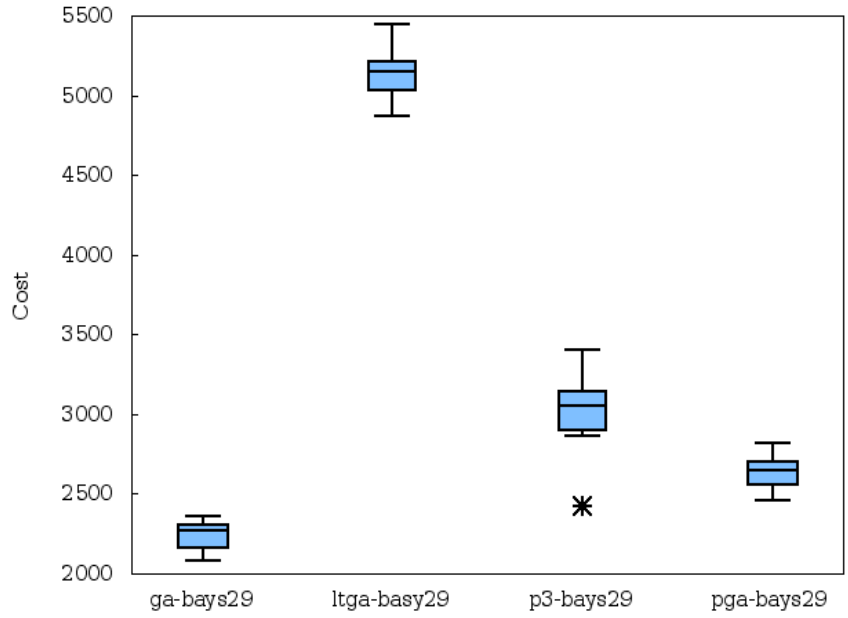
Slika 20: Rastrigin boxplot s vremenskim ograničenjem 10s



Slika 21: Schwefel boxplot s vremenskim ograničenjem 10s

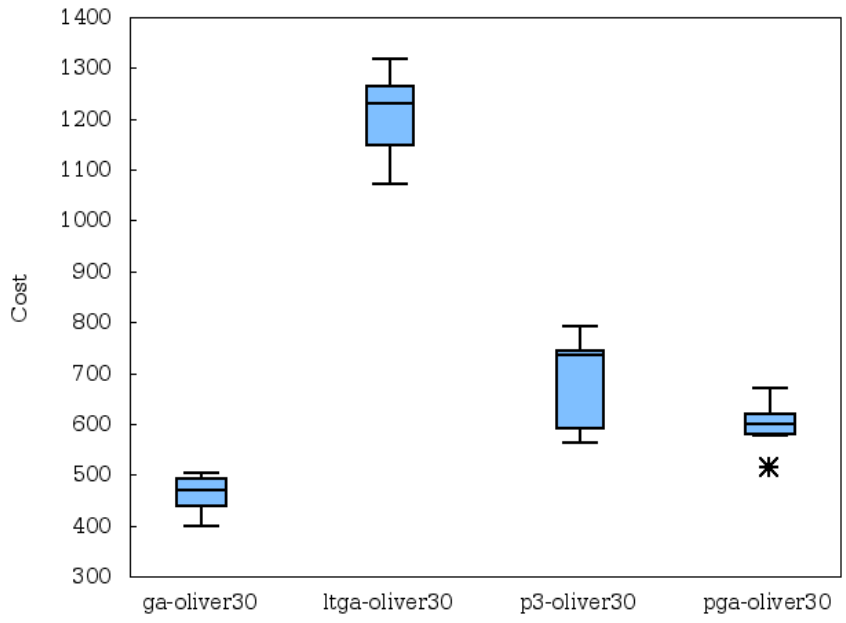
4.3.5. Bays 29 i Oliver 30

Rezultati pokusa s vremenskim ograničenjem za permutacijske probleme ekvivalentni su pokusima s evaluacijskim ograničenjem. Najbolji algoritam je GA, no P3 i PGA ne zaostaju puno, najlošiji je LTGA.



Slika 22: Bays 29 boxplot s vremenskim ograničenjem 10s

Rezultati za permutacijski problem Oliver 30 uz vremensko ograničenje prikazani su na slici 23. Opet se potvrđuje zaključak da algoritam P3 može dati dobra rješenja za permutacijske probleme.

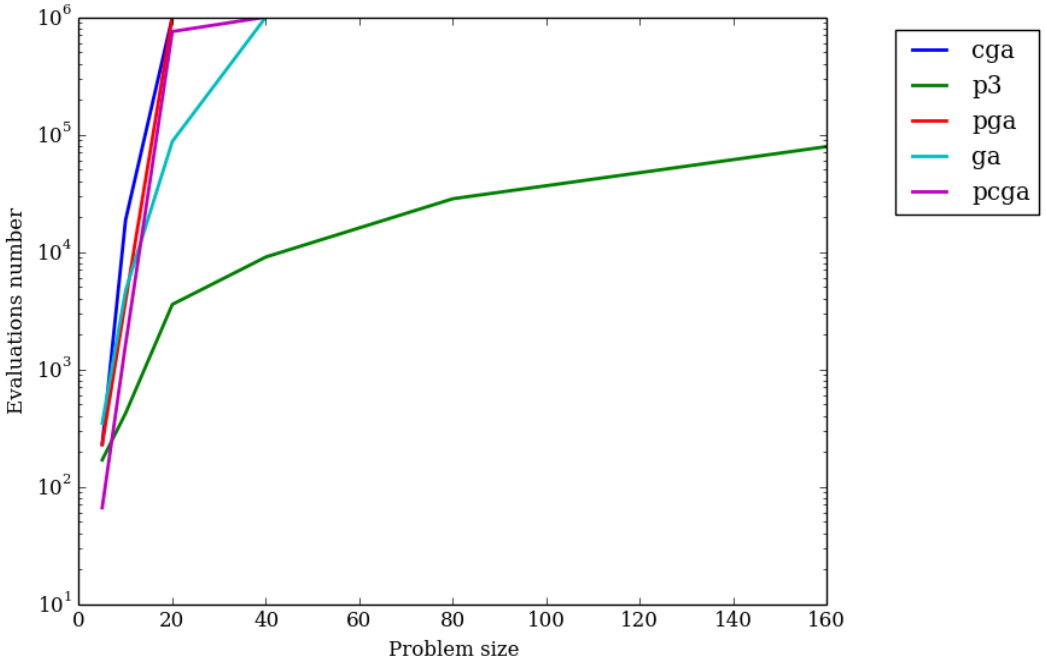


Slika 23: Oliver 30 boxplot s vremenskim ogrničenjem 10s

4.4. Broj evaluacija do optimalnog rješenja

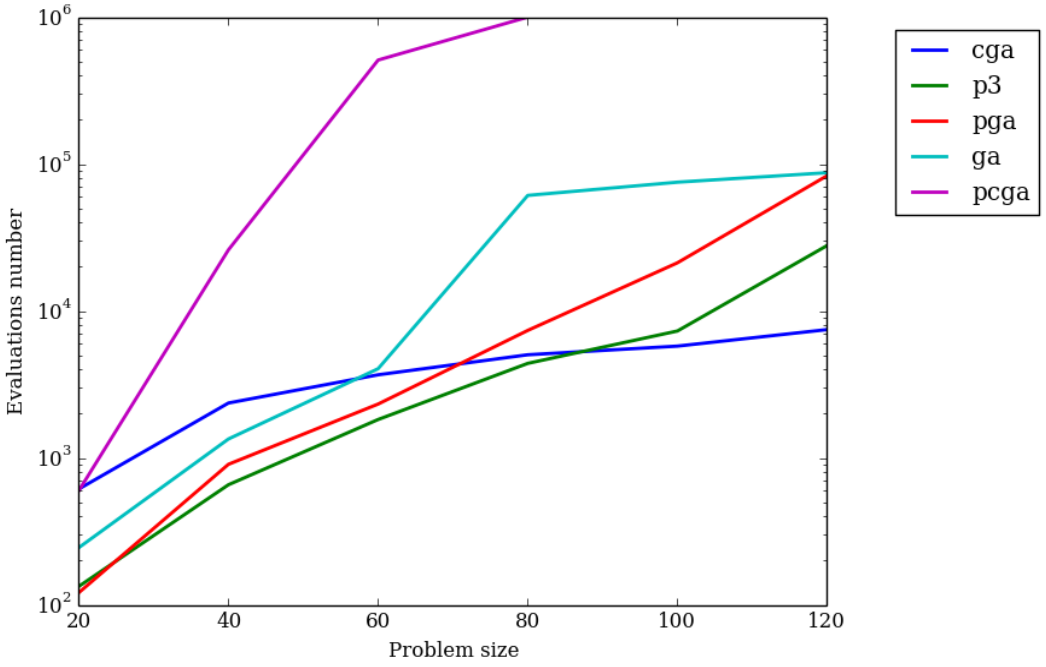
Grafovi u ovom poglavlju pokazuju rezultate izvođenja algoritama uz maksimalan broj evaluacija funkcije cilja od 10^6. Na apscisi je prikazana veličina problema, a na ordinati je broj evaluacija u logaritamskoj skali. Za svaku veličinu problema pokus se pokretao 10 puta i kao valjana vrijednost uzet je medijan broja evaluacija do optimalnog rješenja.

Na problemu varljive zamke najbolji se pokazao algoritam P3, jedino taj algoritam uspijeva pronaći minimum prije ograničenja za sve veličine problema. Svi ostali analizirani algoritmi su dosta slični po rezultatu. GA je nešto bolji, ali neznatno.



Slika 24: Broj evaluacija do optimalnog rješenja za problem varljive zamke

Na Rastrigin problemu kao najbolji algoritam pokazao se CGA jer se iz grafa 25 vidi kako broj evaluacija do optimalnog rješenja samo blago raste i za najveću instancu problema, CGA najbrže daje rješenje. P3 je vrlo blizu CGA algoritmu. Zanimljivo je da PGA bolje radi na ovom problemu od GA. PCGA je dosta lošiji od svih ostalih.

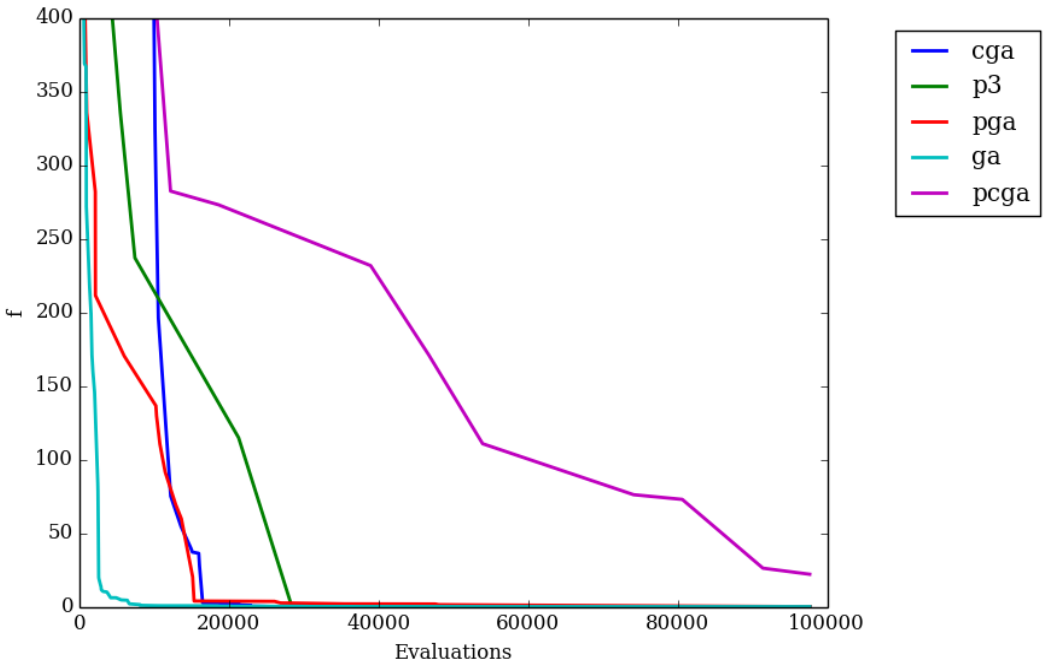


Slika 25: Broj evaluacija do optimalnog rješenja za problem Rastrigin

4.5. Konvergencija kontinuirane optimizacije

U ovom poglavlju prikazana je konvergencija kontinuiranih funkcija (Schwefel i Greiwank). Parametri koji se koriste kod algoritma GA su: veličina populacije 25, faktor križanja 0.55, faktor mutacije 0.05. Parametri korišteni kod algoritma PGA su: faktor križanja 0.40 i faktor mutacije 0.10. Veličina populacije algoritma CGA je 680. Spomenuti parametri dobiveni su kao prosjek vrijednosti parametara dobivenih u poglavlju Optimizacija parametara za kontinuirane optimizacijske probleme. Svaki algoritam izvodi se sa ograničenjem na broj evaluacija od 10^5. Veličina problema je 100 bitova uz 20 bitova po varijabli, dakle ukupno 5 varijabli.

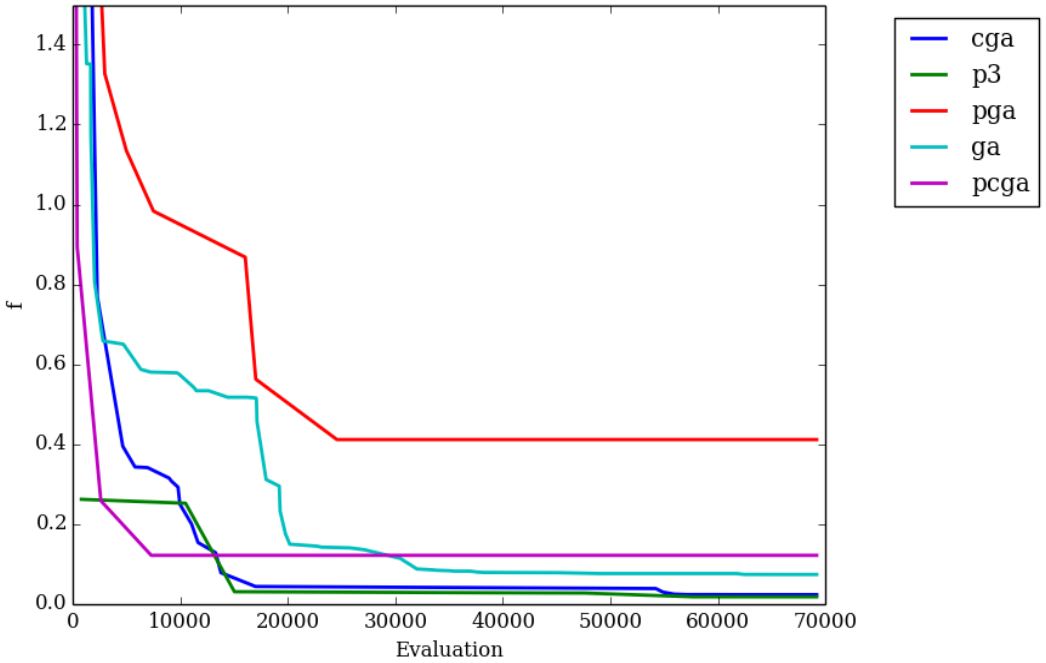
Na grafu 26 prikazana je konvergencija algoritama na funkciji Schwefel. Najsporije konvergira algoritam PCGA i daje rješenje koje nije blizu optimuma. Najbolju konvergenciju pokazuje algoritam GA, a za PGA vrijedi da na ovom problemu konvergira lošije od GA. Za CGA i algoritam P3 se na grafu vide nagla napredovanja. Kod CGA to je posljedica raspršenoga pretraživanja prostora rješenja, dok je kod P3 to posljedica inicijalnog operatora lokalne pretrage.



Slika 26: Konvergencija algoritama na funkciji Schwefel

Graf 27 prikazuje konvergenciju algoritama na funkciji Greiwank. Najbolju konvergen-

ciju daju CGA i algoritam P3, iako je P3 je za nijansu bolji. Na ovom problemu najlošiji je PGA. Kod algoritma P3 se opet vidi učinak operatora lokalne pretrage jer je prvo dobiveno rješenje znatno bolje od inicijalnih rješenja ostalih algoritama.



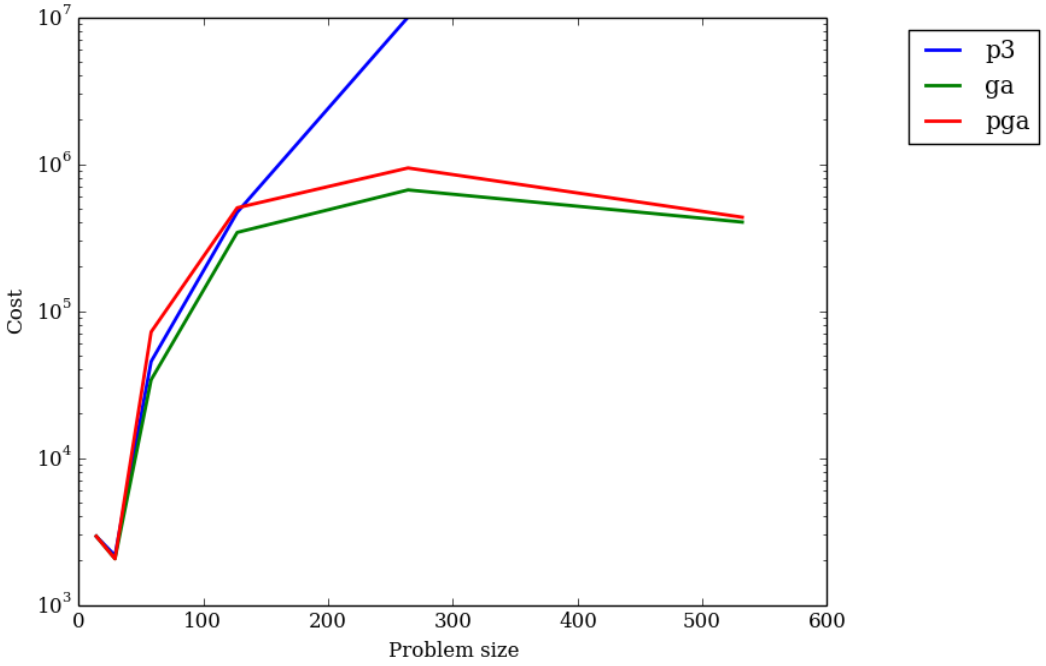
Slika 27: Konvergencija algoritama na funkciji Greiwank

4.6. Problemi trgovačkog putnika

U ovom dijelu ispitani su algoritmi na problemima trgovačkog putnika. Problemi su preuzeti iz TSPLib [18] skupa primjera (burma14, bays29, brazil58, bier127, pr264, att532). Algoritmi koji su ispitani su GA, PGA i P3. Cilj je usporediti različite algoritme i prikazati na koji način dobiveno rješenje ovisi o veličini problema. Kod GA su korišteni parametri: veličina populacije 12, faktor križanja 0.45 i faktor mutacije 0.05. Kod PGA su korišteni parametri: faktor križanja 0.15 i faktor mutacije 0.12. Za svaki algoritam i instancu problema izvršena su 3 pokretanja te je uzet medijan dobivenih vrijednosti kao referentna vrijednost. Uvjet zaustavljanja bilo je vremensko ograničenje od 400s.

Na grafu 28 se vidi da GA daje najbolje rezultate za sve instance problema. PGA daje lošija rješenja, no usporediv je s GA. Algoritam P3 za manje instance problema (burma14, bays29, brazil58, bier127) daje usporedive rezultate u odnosu na GA, no na većim instancama problema (pr264, att532), zbog svoje vremenske i prostorne složenosti, daje znatno

lošija rješenja u odnosu na GA.



Slika 28: Medijan troška za probleme trgovačkog putnika

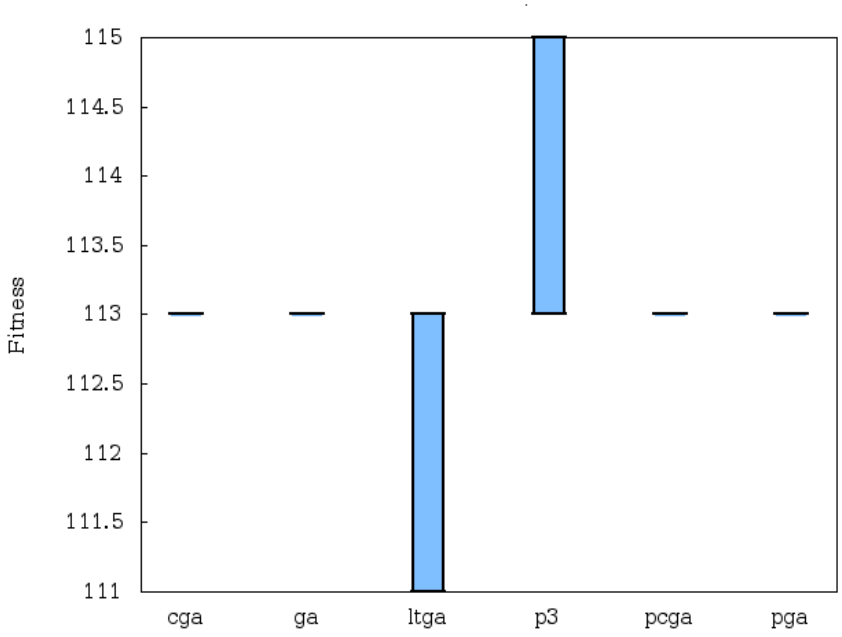
4.7. Kombinatorička mreža

Algoritam ispravnog slučajnog rješenja radi jako dobro, čak toliko dobro da nema potrebe za evolucijskim algoritmima, tj. ako evolucijski algoritmi kreću s rješenjem koje daje taj algoritam, ne mogu unaprijediti to rješenje. Konkretno, algoritam 2 se primijenio na dva problema: *sbox real* i *kccak real*. Kod *sbox real* problema prostor pretraživanja je veličine 2^{432}, a kod *kccak real* 2^{22543}. Za oba problema vrijedi da je broj nevaljanih rješenja znatno veći u usporedbi s brojem valjanih rješenja. Pretraživanje takvog prostora rješenja standardnim evolucijskim algoritmima nema smisla bez generiranih kvalitetnih rješenja koja zadovoljavaju sva ograničenja. Najbolje dobiveno rješenje, prema funkciji troška definiranoj u [12], za *sbox real* ima trošak 2051.52, a za *kccak real* 1476.55.

4.8. Boolean funkcija

Funkcija evaluacije Booleanove funkcije za balansiranost dodaje na dobrotu maksimalno 1, a za nelinearnost maksimalno 118. Dakle, maksimalna dobrota je 119. Na slici 29 je prikazan boxplot 10 izvršavanja za svaki algoritam uz ograničenje od maksimalno 10^7 iteracija.

Najbolje preformanse ima P3, a najlošije algoritam LTGA. No, graf je zanimljiv iz više razloga. Većina algoritama, za dovoljno velik broj evaluacija, a 10^7 je očito dovoljno veliki broj, završi s najboljim rješenjem od 113. To je zato što takvih rješenja ima mnogo u prostoru pretraživanja, a algoritmi nisu dovoljno dobri da se izvuku iz tih lokalnih optimuma. Jedino algoritam P3 odskače od ostalih algoritama u skupu. Isto tako, valja naglasiti da niti jedan od analiziranih algoritama nije izrazito dobar za ovaj problem jer niti jedan nije došao do rješenja od 117 koje je pronađeno u [14].



Slika 29: Boxplot evaluacije Booleanovih funkcija

4.9. Kapacitivni problem usmjeravanja vozila iz višebrojnih skladišta

Prvo je bilo potrebno naći optimalne parametre za algoritme s kojima će se rješavati dani problem (GA, PGA, P3). Parametri su traženi u skupu: veličina populacije - [5, 10, 20, 30, 40, 60, 80, 120], faktor mutacije - [0.0001, 0.0002, 0.0004, 0.0008, 0.0016, 0.0030, 0.0060, 0.0120, 0.0250, 0.0500, 0.1000, 0.2000, 0.4000, 0.8000], faktor križanja - [0.001, 0.002, 0.004, 0.008, 0.016, 0.030, 0.060, 0.120, 0.250, 0.500] istim redoslijedom.

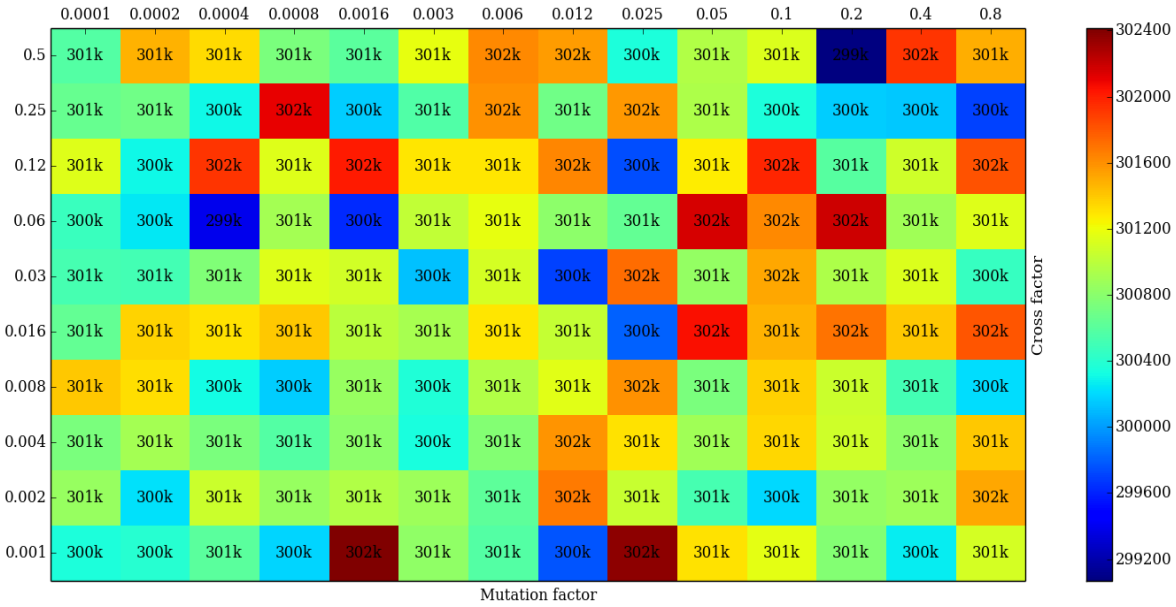
Iz tablice 5 je vidljivo da genetskom algoritmu za rješavanje ovog problema odgovara mala veličina populacije, relativno mali faktor križanja i veći faktor mutacije. Što se tiče PGA, stvar nije slična. PGA daje bolje rezultate sa puno većim faktorom križanja nego GA. Razlog tomu može biti činjenica da PGA nije ograničen populacijom i zapinjanjem u nekom

lokalnom optimumu populacije. Prema ovom pokusu dalo bi se naslutiti da su za PGA bolje veće vrijednosti faktora križanja.

algoritam	veličina populacije	faktor križanja	faktor mutacije
GA	5	0.003	0.500
PGA	-	0.500	0.2000

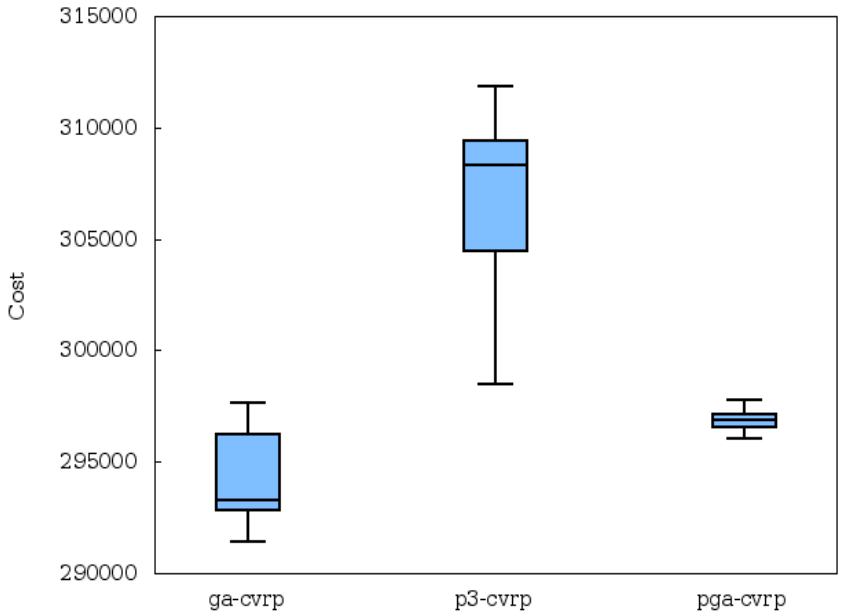
Tablica 5: Optimalni parametri za kapacitivni problem usmjeravanja vozila

Na slici 30 prikazana je temperaturna mapa (engl. *heatmap*) treniranja algoritma PGA na problemu CVRP. Iz slike je vidljivo da median cijene više oscilira što se više ide prema višim vrijednostima parametara, ali i postiže bolje vrijednosti (žarko crvena boja je loša vrijednost, a tamno plava dobra vrijednost).

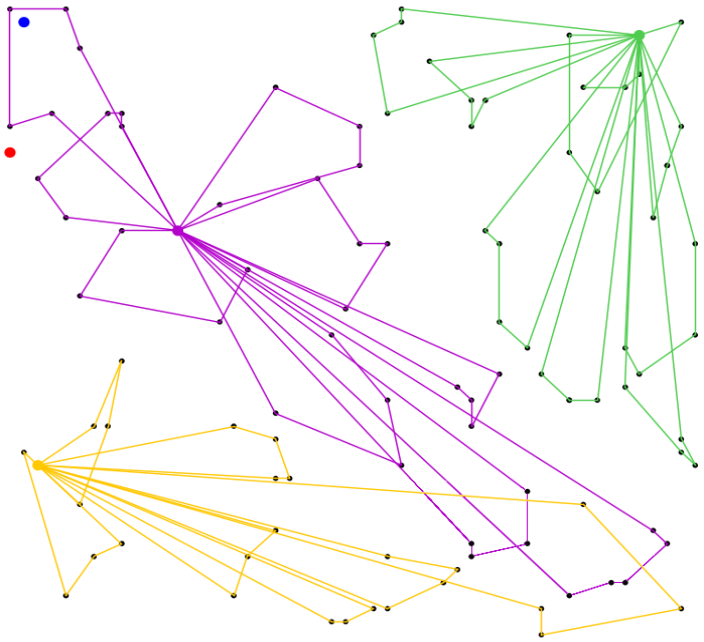


Slika 30: Podešavanje parametara kod PGA algoritma za problem CVRP

Na slici 31 vidi se usporedba tri promatrana algoritma na problemu CVRP. Vidi se da je GA najbolji, PGA je nešto lošiji od GA, a P3 je najlošiji. Na ovom problemu dolazi do izražaja složenost algoritma P3 i činjenica da je većina operacija kod algoritma P3 kvadratno ovisna o veličini problema. Konkretno, prikaz rješenja ovog problema ima veličinu $2 \cdot 10^2$. Za takve i veće redove veličina P3 postaje spor u odnosu na GA, odnosno brzina GA dolazi više do izražaja. Na slici 32 vidi se prikaz jednog rješenja problema CVRP. Konkretno, rješenje na slici ima trošak 288853 i do njega je došao GA opisan u ovom radu.



Slika 31: Usporedba GA, PGA i P3 na problemu kapacitivnog usmjeravanja vozila



Slika 32: Primjer rješenja problema kapacitivnog usmjeravanja vozila

5. Zaključak

Za početak treba ustvrditi da nema algoritma koji bi bio bolji od svih ostalih algoritama (engl. *No free lunch theorem*). Svaki algoritam radi dobro u nekom kontekstu. Potrebno je dobro pripremiti taj kontekst kako bi dao dobre rezultate. Za neparametarske algoritme vrijedi ista stvar. Na nekim problemima rade dobro, a na nekima lošije.

Algoritam CGA, iako vrlo jednostavan, pokazao se kao dobar izbor za nekolicinu problema, primjerice, za kontinuirane probleme ili za problem vodećih jedinica. U svakom slučaju, valja s njim probati optimizirati problem, ako problem može imati niz bitova kao reprezentaciju rješenja, jer CGA radi samo s nizom bitova.

P3 se pokazao dobar za prikaz rješenja nizom bitova ili za permutacijski prikaz rješenja. Na svim testnim problemima P3 je dao dobre rezultate i svakako ga vrijedi koristiti prilikom optimizacije. Općenito, piramidalna struktura populacije ima smisla jer pamti sva generirana rješenja i time otvara mogućnost da svako generirano rješenje uđe u postupak križanja s novim generiranim rješenjima. Važno je napomenuti da su dobri rezultati dobiveni strategijom u kojoj se novo generirano rješenje pokušava križati sa svakom razinom u piramidi. Na taj način pospješuje se konvergencija jer postoji šansa da naizgled loše rješenje, u križanju s nekim rješenjem koje je u višem sloju piramide, postane dobro rješenje. Osim toga, algoritmi s piramidalnom populacijom nemaju parametar veličine populacije. Samim time ne treba ugađati taj parametar što znači da je algoritam brže spreman za neki konkretan problem. Primjeri takvih algoritama su P3 i PGA.

Na problemu Boolean funkcija, od analiziranih algoritama najbolje radi algoritam P3, dok su ostali su za nijansu lošiji. No, niti P3 na tom problemu nije najsuvremeniji (engl. *State of the art*). Za problem kombinatoričkih mreža, konstruirani algoritam pronalaska ispravnog rješenja pokazao se jako dobrim i nijedan evolucijski algoritam nije uspio doći do kvalitetnijeg rješenja u usporedbi s najboljim rješenjem algoritma ispravnog slučajnog rješenja kombinatoričke mreže.

Literatura

[1] J. Holland, Adaptation in Natural and Artificial Systems, 1975.

[2] G. Harik; F. Lobo. *A parameter-less genetic algorithm*. Technical Report IlliGAL 99009, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL, 1999.

[3] Dirk Thierens. *Linkage Tree Genetic Algorithm: first results*. GECCO'10, July 7-11, 2010, Portland, Oregon, USA, 2010.

[4] Brian W. Goldman; William F. Punch. *Parameter-less Population Pyramid*. GECCO '14, ISBN: 978-1-4503-2662-9, pages 785-792, ACM New York, NY, USA 2014.

[5] Harik, G.R.; Lobo, F.G.; Goldberg, D.E. *The compact genetic algorithm*. Evolutionary Computation, IEEE Transactions on, vol. 3, no. 4, pp.287, 297, 1999.

[6] B. W. Goldman; D. R. Tauritz. *Linkage tree genetic algorithms: variants and analysis*. GECCO '12, pages 625–632, ACM Philadelphia, Pennsylvania, USA, 7-11 July 2012.

[7] J. Grefenstette. *Optimization of control parameters for genetic algorithms*. IEEE Trans. on Systems, Man, and Cybernetics, SMC-16(1):122–128, 1986.

[8] I. Gronau and S. Moran. *Optimal implementations of UPGMA and other common clustering algorithms*. Information Processing Letters, 104(6): 205–210, 2007.

[9] Andreu Sancho. *Deceptive trap*. `http://andreusancho.blogspot.com/2013_12_01_archive.html`, 2013.

[10] Holtschulte, N.; Moses, M. *Should Every Man be an Island*. `http://www.cs.unm.edu/~neal.holts/dga/index.html`, 2013.

[11] Čupić Marko. *Prirodom inspirirani optimizacijki algoritmi. Metaheuristike.*, FER, Zagreb, 2013.

[12] Šišejković Dominik. *Optimizacija povećavanja propusnosti kombinacijskih mreža evolucijskim algoritmima*, FER, Zagreb, 2014.

[13] Picek, Stjepan; Marchiori, Elena; Batina, Lejla; Jakobović, Domagoj. *Combining Evolutionary Computation and Algebraic Constructions to Find Cryptography-Relevant Boolean Functions.* Lecture Notes in Computer Science. 8672 (2014); 822-831.

[14] Stjepan Picek; Domagoj Jakobovic; Julian F. Miller; Elena Marchiori; Lejla Batina. *Evolutionary Methods for the Construction of Cryptographic Boolean Functions*. Springer International Publishing Switzerland, P. Machado et al. (Eds.): EuroGP 2015, LNCS 9025, pp. 192-204, 2015.

[15] Farhad Nadi; Ahamad Tajudin Khader. *A parameter-less genetic algorithm with customized crossover and mutation operators*. GECCO '11, pages 901-908, ISBN: 978-1-4503-0557-0, ACM New York, NY, USA 2011.

[16] Srichol Phiromlap; Sunisa Rimcharoen. *A Frequency-Based Updating Strategy in Compact Genetic Algorithm*. IEEE, ICSEC 4-6 Sept. 2013, pages 207-211, ISBN: 978-1-4673-5322-9, 2013.

[17] Ivan Slivar. *Kapacitativni problem usmjeravanja vozila iz višebrojnih skladišta*. FER, Zagreb, 2015.

[18] Universität Heidelberg. *TSPLIB*, `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95`, Germany.

Primjena neparametarskih algoritama u optimizaciji

Sažetak

Neparametarskim algoritmima postiže se brža prilagodba algoritma konkretnom problemu, što je svakako prednost u odnosu na algoritme koji posjeduju neki skup parametara. S druge strane, neparametarski algoritmi obično su ograničeni na neki mali skup prikaza rješenja koji oni mogu izgenerirati. Ako se problem može jednostavno prikazati onim prikazom s kojim radi neki neparametarski algoritam onda svakao treba probati riješiti problem koristeći neparametarske algoritme. Ovim radom također se pokazuje da ne postoji algoritam koji bi idealno radio sa svim optimizacijskim problemima (engl. *No free lunch theorem*).

Ključne riječi: algoritam, neparametarski, optimizacija

Application of nonparametric optimization algorithms

Abstract

Nonparametric optimization algorithms could be faster adopted to the specific optimization problem then other optimization algorithms with some set of parameters. On the other hand, nonparametric optimization algorithms are limited on small set of genotypes. If a problem solution could be represented by a genotype supported by nonparametric optimization algorithm then that algorithm should be tested on that problem. Through this paper the no free lunch theorem is also confirmed because none of the researched algorithms produce the best results on all test problems.

Keywords: algorithm, nonparametric, optimization