# Genetski algoritam sa nedominiranim sortiranjem (NSGA-II)

Seminarski rad u okviru kursa
Računarska inteligencija
Matematički fakultet

Staša Đorđević

19. avgust 2025.

## Sadržaj

# 1 Uvod u genetske algoritme

Genetski algoritmi predstavljaju grupu optimizacionih metoda koje se zasnivaju na principima prirodne selekcije i evolucije. Inspirisani su biološkim procesima kao što su selekcija, ukrštanje (kroz reprodukciju), mutacija i nasleđivanje, koji omogućavaju preživljavanje i adaptaciju organizama u prirodi. Slično tome, osnovni koraci u implementaciji genetskih algoritama uključuju selekciju, ukrštanje i mutaciju. Ovi koraci se ponavljaju kroz više generacija kako bi se iz populacije rešenja razvila najbolja moguća rešenja za dati problem.

- **Selekcija** je proces odabira jedinki za ukrštanje na osnovu njihove prilagođenosti. Postoje dve osnovne varijante selekcije:

  1. **Turnirska selekcija** - Odabir k slučajnih jedinki iz populacija i "turnirsko takmičenje"gde pobeđuje najprilagođenija od izabranih jedinki
  2. **Ruletska selekcija** - U ovoj metodi, verovatnoća selekcije jedinke zavisi od njene uspešnosti u odnosu na ostale jedinke u populaciji. Prilagođenije jedinke imaju veću verovatnoću da budu izabrane, slično kao u ruletu.

- **Ukrštenje** omogućava kombinovanje gena odabranih jedinki, stvarajući nove potomke koji mogu naslediti najbolje karakteristike svojih "roditelja". Postoji nekoliko osnovnih varijanti ukrštanja:

  1. **Jednopoziciono ukrštanje** - Genetski materijal od roditelja se deli na osnovu jedne slučajno odabrane tačke preseka, a potomci nasleđuju deo od oba roditelja prema toj tački.
  2. **Višepoziciono ukrštanje** - Ova metoda koristi više tačaka preseka na genomima roditelja, što omogućava veću raznovrsnost u potomcima.
  3. **Uniformno ukrštanje** - U ovoj varijanti, gene sa oba roditelja se nasumično kombinuju kako bi se stvorio potomak, bez fiksnih tačaka preseka.

- **Mutacija** se koristi da bi se unela nasumična promena u genetski kod jedinke, što omogućava istraživanje novih mogućnosti i sprečava algoritam da se 'zaglavi' u lokalnim ekstemumima.

**Elitizam** je metoda koja garantuje da će najbolje jedinke iz trenutne generacije biti prenete u sledeću generaciju bez promena. Elitizam se koristi kako bi se sprečilo da se najbolja rešenja izgube tokom evolucije.

Kroz ove procese, genetski algoritmi omogućavaju efikasno istraživanje prostora rešenja i postepeno poboljšanje kvaliteta rešenja tokom vremena.

# 2 Opis algoritma NSGA-II

Algoritam NSGA (Non-dominated Sorting Genetic Algorithm) je popularan genetski algoritam zasnovan na nedominaciji za višeciljnu optimizaciju. Njegova modifikovana verzija, NSGA-II, koja rešava neke probleme zbog kojih je kritikovana osnovna verzija algoritma, često se koristi kao efikasnije rešenje u primenama višeciljne optimizacije. Višeciljna optimizacija podrazumeva istovremenu optimizaciju dva ili više međusobno suprostavljenih ciljeva. Cilj je naći skup rešenja koji je najbolji kompromis između ciljeva.

Ta rešenja formiraju tzv. **Pareto front**, u kojem nijedno rešenje nije bolje od drugog, osim ako se jedan cilj ne poboljša na račun pogoršanja drugog.

U NSGA-II algoritmu, termini *non-dominated* i *dominated* se koriste da opišu odnos između rešenja na osnovu njihovih performansi u odnosu na više ciljeva optimizacije.

Rešenje se smatra **nedominiranim** (engl. *non-dominated*) u odnosu na drugo rešenje ako nijedno od njih nije bolje u svim ciljevima. Drugim rečima, rešenje $A$ je *nedominirano* u odnosu na rešenje $B$ ako:

- $A$ nije lošije u svim ciljevima od $B$,
- i $B$ nije lošije u svim ciljevima od $A$.

Rešenje se smatra **dominiranim** (engl. *dominated*) u odnosu na drugo rešenje ako postoji rešenje koje je bolje u svim ciljevima. Drugim rečima, rešenje $A$ je *dominirano* u odnosu na rešenje $B$ ako:

- $B$ je bolje ili jednako u svim ciljevima od $A$,
- i u barem jednom cilju $B$ je bolje od $A$.

Kratak opis algoritma: Prvo se populacija inicijalizuje na standardan način, u skladu sa problemom koji rešavamo. Nakon toga, jedinke u njoj se sortiraju po frontovima prema principu nedominacije. Prvi front je potpuno nedominirani skup u trenutnoj populaciji, tj. skup svih rešenja od kojih ne postoji bolje rešenje u svim ciljevima. Drugi front sadrži jedinke koje su dominirane samo od strane jedinki iz prvog fronta, i tako dalje. Svakoj jedinki se dodeljuje rang na osnovu fronta kojem pripadaju - one iz prvog fronta dobijaju rang 1, iz drugog 2, i tako dalje. Pored ranga, svaka jedinka ima i novi parametar - *distanca gužve* (engl. *crowding distance*). To je mera koja se koristi za održavanje raznolikosti između rešenja unutar jednog pareto fronta. Predstavlja meru bliskosti jedinke njenim susedima. Veća prosečna distanca gužve rezultira boljom raznovrsnošću u populaciji. Favorizuje manje naseljene regione. Nakon sortiranja, unutar svakog fronta, računa se distanca gužve za jedinke u tom frontu. Primarni kriterijum za selekciju je rang. Ako dve jedinke imaju isti rang, bolja je ona sa većom distancom gužve. Ovaj pristup osigurava da algoritam održava i intenzifikaciju (kroz rang) i diverzifikaciju (kroz distancu gužve). Roditelji se biraju iz populacije koristeći turnirsku selekciju. Odabrana populacija generiše potomke pomoću operacija ukrštanja i mutacije, koje će biti detaljnije opisane u narednom poglavlju. Populacija, zajedno sa trenutnom populacijom i trenutnim potomcima, ponovo se sortira prema principu nedominacije, i samo se najboljih N jedinki selektuje, gde je N veličina populacije. Selekcija se zasniva na rangu i distanci gužve u poslednjem pareto frontu.

# 3 Opis mog rešenja

## 3.1 Grupisanje u pareto frontove - sortiranje

Ovaj algoritam koristi metod nedominiranog sortiranja da bi organizovao populaciju u Pareto frontove. U prvom koraku se inicijalizuje broj dominacija i lista dominiranih jedinki za svaku jedinku. Zatim, kroz dvostruki for petlju, algoritam poredi svaku jedinku sa svim ostalim u populaciji i ažurira broj dominacija i listu dominiranih jedinki. Nakon

što su svi odnosi dominacije utvrđeni, jedinke se grupišu u Pareto frontove prema njihovoj dominaciji. Nakon što je jedinka pridružena određenom Pareto frontu, dodeljuje joj se rang na osnovu njega, koji će nam kasnije koristiti u selekciji.

```
1: Input: populacija
2: Output: pareto frontovi
3: pareto frontovi ← prazna lista
4: broj dominacija ← prazna mapa
5: dominirane jedinke ← prazna mapa
6: n ← dužina populacije
7: for svaku jedinku u populaciji do
8:    broj dominacija[jedinka] ← 0
9:    dominirane jedinke[jedinka] ← prazna lista
10: end for
11: for i = 0 to n-1 do
12:    for j = 0 to n-1 do
13:       if i ≠ j then
14:          if dominira(populacija[i], populacija[j]) then
15:             dodaj populacija[j] u dominirane jedinke[populacija[i]]
16:          else if dominira(populacija[j], populacija[i]) then
17:             broj dominacija[populacija[i]] ← broj dominacija[populacija[i]] + 1
18:          end if
19:       end if
20:    end for
21: end for
22: trenutni front ← sve jedinke sa brojem dominacija 0
23: trenutni indeks ← 1
24: while trenutni front nije prazan do
25:    dodaj trenutni front u pareto frontovi
26:    sledeći front ← prazna lista
27:    for svaku jedinku u trenutnom frontu do
28:       rang jedinke ← trenutni indeks
29:       for svaku dominiranu jedinku do
30:          broj dominacija[dominirana] ← broj dominacija[dominirana] - 1
31:          if broj dominacija[dominirana] == 0 then
32:             dodaj dominiranu u sledeći front
33:          end if
34:       end for
35:    end for
36:    trenutni front ← sledeći front
37:    trenutni indeks ← trenutni indeks + 1
38: end while
39: return  pareto frontovi
```

## 3.2 Određivanje distance gužve

Ova funkcija izračunava distancu gužve za svaku jedinku u Pareto frontu, koristeći sve ciljeve optimizacije. Za svaku jedinku u Pareto frontu, vrednost distance gužve se inicijalizuje na 0. Zatim se za svaki cilj, Pareto front sortira prema vrednostima cilja, a

prvoj i poslednjoj jedinki dodeljuje se beskonačna vrednost distance gužve. Distanca za svaku jedinku i svaki cilj se izračunava kao odnos razlike između fitnesa susednih jedinki i razlike između minimalne i maksimalne vrednosti fitnesa (unutar trenutnog Pareto fronta) za dati cilj. Izračunata distanca se dodaje na ukupnu distancu gužve. Ovaj proces se ponavlja za svaki cilj.

**Input:** pareto front, broj ciljeva
**Output:** ažurirani pareto front sa izračunatim distancama gužve
n ← dužina pareto fronta
**for** svaka jedinka u pareto frontu **do**
    jedinka.distanca gužve ← 0
**end for**
**for** i = 0 **to** broj ciljeva - 1 **do**
    sortiraj pareto front prema fitnesu[i]
    pareto front[first].distanca gužve ← ∞
    pareto front[last].distanca gužve ← ∞
    f_min ← pareto front[first].fitness[i]
    f_max ← pareto front[last].fitness[i]
    **for** k = 1 **to** n-2 **do**
        distanca ← (pareto front[k+1].fitness[i] - pareto front[k-1].fitness[i]) / (f_max - f_min)
        pareto front[k].distanca gužve ← pareto front[k].distanca gužve + distanca
    **end for**
**end for**
**return** pareto front

## 3.3 Selekcija

U implementaciji je korišćena turnirska selekcija. Bira se k nasumičnih jedinki iz populacije, i vraća se najbolja od njih. Kriterijum za određivanje najbolje jedinke je na osnovu ranga - što manji rang - to je bolja jedinka. Ako dve jedinke imaju isti rang, bolja je ona koja ima veću distancu gužve.

## 3.4 Ukrštanje

Zbog testiranja algoritma nad funkcijama koje koriste neprekidne vrednosti, koristimo poseban algoritam ukrštanja: **SBX** (engl. *Simulated Binary Crossover*). Cilj SBX-a je da simulira ponašanje jednodelnog binarnog ukrštanja, ali u prostoru realnih brojeva. Njegova glavna prednost je kontrola stepena intenzifikacije i diverzifikacije kroz **distributivni indeks ukrštanja** ($\eta_c$).

SBX generiše dve nove jedinke (potomke) na osnovu dva roditelja, gde vrednosti gena potomaka leže između ili blizu vrednosti gena roditelja. Veća vrednost $\eta_c$ dovodi do stvaranja potomaka bližih roditeljima (podstiče intenzifikaciju), dok manja vrednost omogućava šire istraživanje prostora rešenja (podstiče diverzifikaciju). Tipične vrednosti za $\eta_c$ su u opsegu od 5 do 20, ali konkretan izbor zavisi od prirode problema i ciljeva optimizacije.

U implementaciji koristimo uniformno ukrštanje - gde se za svaki gen jedinke sa verovatnoćom 0.5 računa nova vrednost gena za potomke koristeći SBX formulu.

**Koraci algoritma za SBX ukrštanje**

1. **Inicijalizacija i iteracija po genima:**
   - Veličina hromozoma ($n$) određuje se kao dužina atributa `code` roditelja.
   - Algoritam iterira kroz svaki gen (poziciju) hromozoma.
2. **Verovatnoća ukrštanja:**
   - Sa verovatnoćom 50% ($random.random() \leq 0.5$), algoritam računa nove vrednosti gena za potomke koristeći SBX formulu.
   - Ako ukrštanje ne treba da se desi, potomci direktno nasleđuju gene roditelja.
3. **Računanje parametra $\beta_k$:**
   - $\beta_k$ određuje proporciju ukrštanja
   - Generiše se uniformni slučajan broj $u \in [0, 1]$.
   - Ako je $u \leq 0.5$, računa se:
   $$\beta_k = (2u)^{\frac{1}{\eta_c+1}}$$
   - Inače, računa se:
   $$\beta_k = \left(\frac{1}{2(1-u)}\right)^{\frac{1}{\eta_c+1}}$$
4. **Generisanje gena za potomke:**
   - Koristeći $\beta_k$, potomci nasleđuju kombinacije roditeljskih gena prema sledećim formulama:
   $$child1[i] = 0.5 \times ((1+\beta_k) \times parent1[i] + (1-\beta_k) \times parent2[i])$$
   $$child2[i] = 0.5 \times ((1-\beta_k) \times parent1[i] + (1+\beta_k) \times parent2[i])$$
5. **Ograničenje vrednosti gena:**
   - Osigurava se da svaki gen potomaka ostane u opsegu $[0, 1]$ pomoću:
   $$child.code[i] = \min(\max(child.code[i], 0), 1)$$

## 3.5 Mutacija

U ovom radu koristimo **polinomijalnu mutaciju** (engl. *Polynomial Mutation*), koja je popularna tehnika u evolutivnim algoritmima za probleme sa realnim vrednostima. Njen cilj je da unese dovoljno varijacije u populaciju kako bi se omogućilo efikasnije pretraživanje prostora rešenja i izbegavanje lokalnih ekstremuma.

Polinomijalna mutacija funkcioniše tako što modifikuje gene jedinke u skladu sa slučajnim brojem $r \in (0, 1)$ i distribucionim indeksom mutacije $\eta_m$. Veće vrednosti $\eta_m$ dovode do manjih promena u vrednostima gena, dok manje vrednosti omogućavaju veće promene, čime se podstiče istraživanje šireg prostora rešenja.

**Koraci algoritma za polinomijalnu mutaciju**

1. **Iteracija kroz gene jedinke:**
   - Algoritam prolazi kroz svaki gen $i$ hromozoma jedinke.
   - Verovatnoća mutacije za svaki gen je definisana parametrom $p$.
2. **Generisanje slučajnog broja $r$:**
   - Ako $random.random() < p$, gen se menja.
   - Generiše se uniformno slučajan broj $r \in (0, 1)$.
3. **Računanje promene $\Delta_q$:**
   - Ako je $r < 0.5$, računa se:
   $$\Delta_q = (2r)^{\frac{1}{\eta_m+1}} - 1$$
   - Ako je $r \geq 0.5$, računa se:
   $$\Delta_q = 1 - (2(1-r))^{\frac{1}{\eta_m+1}}$$
4. **Ažuriranje gena:**
   - Gen se ažurira dodavanjem $\Delta_q$:
   $$child.code[i] \mathrel{+}= \Delta_q$$
   - Osigurava se da gen ostane u opsegu $[0, 1]$:
   $$child.code[i] = \min(\max(child.code[i], 0), 1)$$

Efikasnost polinomijalne mutacije zavisi od pravilnog izbora parametara $p$ i $\eta_m$. U ovom radu koristimo $\eta_m = 20$, što omogućava umeren stepen promene vrednosti gena, dok verovatnoća mutacije $p$ zavisi od specifičnog problema i ciljeva optimizacije.

## 3.6 Implementacija algoritma NSGA-II

Implementacija sledi osnovne korake algoritma opisane u prethodnom delu, uz dodatne detalje vezane za parametre i funkcionalnost.

**Parametri funkcije `nsga2`**

- `population_size`: Broj jedinki u populaciji.
- `num_variables`: Broj promenljivih koje definišu svaku jedinku.
- `num_generations`: Broj generacija (iteracija algoritma).
- `tournament_size`: Broj jedinki uključenih u turnirsku selekciju.
- `mutation_prob`: Verovatnoća mutacije za svaki gen jedinke.
- `elitism_size`: Broj najboljih jedinki koje se direktno prenose u narednu generaciju.
- `objective_function`: Ciljna funkcija koja se koristi za evaluaciju svake jedinke.

**Opis implementacije**

Algoritam se sastoji od sledećih koraka:

1. **Inicijalizacija populacije:** Početna populacija se generiše nasumično. Svaka jedinka se inicijalizuje sa brojem promenljivih `num_variables`, a njena vrednost ciljne funkcije se računa pomoću funkcije `objective_function`.

2. **Sortiranje prema dominaciji:** Na početku svake iteracije populacija se sortira u Pareto frontove koristeći funkciju `non_dominated_sorting`. Svakom frontu se dodeljuje rang, a jedinkama unutar frontova računa se distanca gužve (*crowding distance*) radi održavanja raznovrsnosti rešenja.

3. **Elitizam:** Najboljih `elitism_size` jedinki prenosi se direktno u sledeću generaciju. Ovaj pristup osigurava očuvanje najboljih rešenja tokom evolucije.

4. **Selekcija roditelja:** Roditelji se biraju korišćenjem turnirske selekcije, gde su jedinke sa manjim rangom i većom distancom gužve preferirane. Ovo osigurava da se favorizuju kvalitetna i raznovrsna rešenja.

5. **Generisanje potomaka:** Nad izabranim roditeljima se primenjuje ukrštanje i polinomijalna mutacija kako bi se generisale nove jedinke. Mutacija se primenjuje sa verovatnoćom `mutation_prob`.

6. **Ažuriranje populacije:** Kombinovanjem trenutne populacije i potomaka kreira se nova populacija veličine `population_size`. Selekcija novih jedinki se vrši na osnovu ranga i distance gužve.

7. **Vizualizacija rezultata:** Nakon završetka iteracija, krajnji Pareto front se prikazuje grafički kako bi se vizualizovala raspodela rešenja.

# 4 Eksperimentalni rezultati

U ovoj sekciji su predstavljeni rezultati NSGA-II algoritma za ciljne funkcije $ZDT1$, $ZDT2$, i $ZDT3$. Ove funkcije su standardni benchmark testovi u višeciljnoj optimizaciji.

## 4.1 Opis ciljnih funkcija

### ZDT1

Pareto front ove funkcije je konveksan, što je čini jednostavnom za optimizaciju. Glavni cilj je testiranje sposobnosti algoritma da pronađe ravnomerno raspodeljena rešenja na frontu.

$$f_1(x) = x_1,$$
$$f_2(x) = g(x) \cdot \left(1 - \sqrt{\frac{f_1(x)}{g(x)}}\right),$$

gde je:

$$g(x) = 1 + \frac{9}{n-1} \sum_{i=2}^{n} x_i,$$

i $x_1 \in [0,1]$, $x_i \in [0,1]$ za $i = 2, 3, \ldots, n$.

**ZDT2**

Ova funkcija ima konkavan Pareto front, što je izazovnije u poređenju sa ZDT1.

$$f_1(x) = x_1,$$
$$f_2(x) = g(x) \cdot \left(1 - \left(\frac{f_1(x)}{g(x)}\right)^2\right),$$

gde je:

$$g(x) = 1 + \frac{9}{n-1}\sum_{i=2}^{n} x_i,$$

i $x_1 \in [0,1]$, $x_i \in [0,1]$ za $i = 2, 3, \ldots, n$.

**ZDT3**

Pareto front je isprekidan i sastoji se od više nepovezanih delova, što predstavlja dodatni izazov za algoritam u održavanju raznovrsnosti.

$$f_1(x) = x_1,$$
$$f_2(x) = g(x) \cdot \left(1 - \sqrt{\frac{f_1(x)}{g(x)}} - \frac{f_1(x)}{g(x)} \cdot \sin(10\pi f_1(x))\right),$$

gde je:

$$g(x) = 1 + \frac{9}{n-1}\sum_{i=2}^{n} x_i,$$

i $x_1 \in [0,1]$, $x_i \in [0,1]$ za $i = 2, 3, \ldots, n$.

# 5 Poređenje mojih rezultata i onih iz literature

## 5.1 Pymoo

Pymoo je optimizaciona biblioteka razvijena u Pythonu, namenjena rešavanju višekriterijumskih optimizacionih problema. Njena implementacija NSGA-II algoritma je dobro optimizovana i široko korišćena u istraživačkim radovima i industrijskim primenama. Za poređenje mog rešenja sa rešenjem iz biblioteke pymoo, koristiću nekoliko metrika koje omogućavaju objektivnu procenu kvaliteta rešenja:

1. Hipervolumen (HV)
2. Maksimalni raspon (MS)
3. Uniformnost distribucije rešenja

U literaturi [5] algoritam se testirao sa sledećim parametrima:

- veličina populacije = 100
- broj generacija = 200 za HV i raspodelu, 300 za MS
- ukrštanje - dvopoziciono
- verovatnoća ukrštanja = 0.9
- mutacija - polinomijalna
- verovatnoća mutacije = 0.033
- referentna tačka = (1.2, 1.5)

U našem rešenju, koristićemo iste parametre tamo gde mogu da budu isti.
Nakon isprobavanja, došla sam do zaključka da je bolje povećati broj generacija. Primetila sam mnogo veća odstupanja kad sam inicijalno stavila da je broj generacija 200. Kada sam promenila da broj generacija bude 500, dobila sam značajno bolje rezultate. Time zaključujem da pymoo algoritam brže konvergira od moje implementacije algoritma.

### 5.1.1 Hipervolumen (HV)

Hipervolumen meri oblast pokrivenu rešenjima na Pareto frontu u višedimenzionalnom prostoru. Za dva cilja, to je površina između tačaka na Pareto frontu i referentne tačke, koja se nalazi izvan dometa svih rešenja, često sa većim vrednostima nego što je moguće dostići.

Formule za izračunavanje hipervolumena su sledeće:
- Površina između dve uzastopne tačke na Pareto frontu:

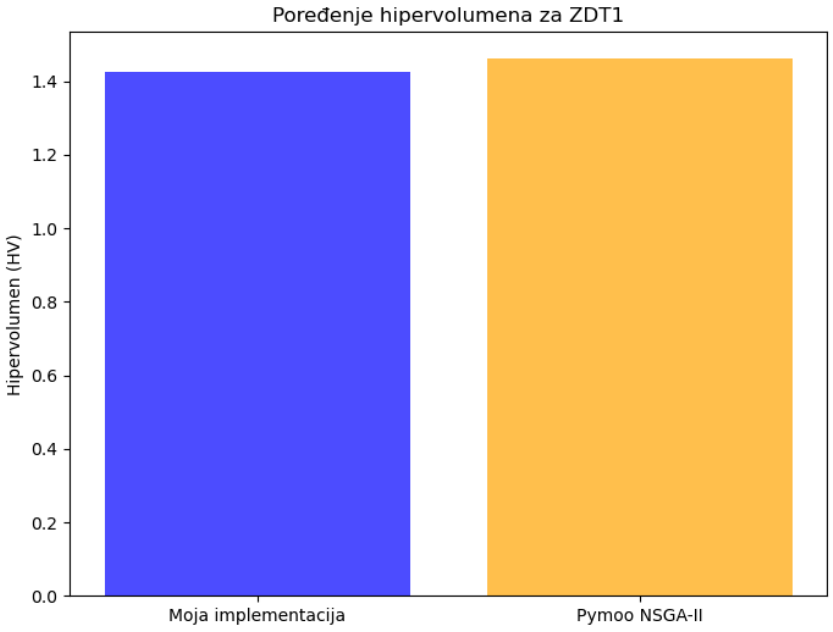
$$\text{Površina} = \text{Širina} \times \text{Visina}$$

gde je širina razlika između vrednosti prvog cilja ($f_1$) dve tačke, a visina je prosečna visina između vrednosti drugog cilja ($f_2$).
- Površina između poslednje tačke i referentne tačke:

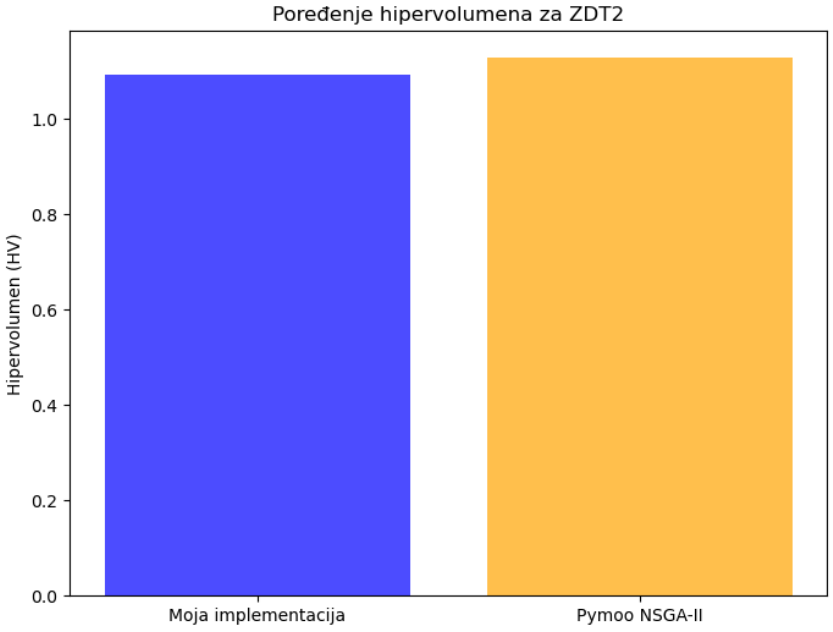
$$\text{Površina poslednje} = (r_1 - f_1) \times (r_2 - f_2)$$

gde su $f_1$ i $f_2$ koordinate poslednje tačke na Pareto frontu, a $r_1$ i $r_2$ su vrednosti referentne tačke.

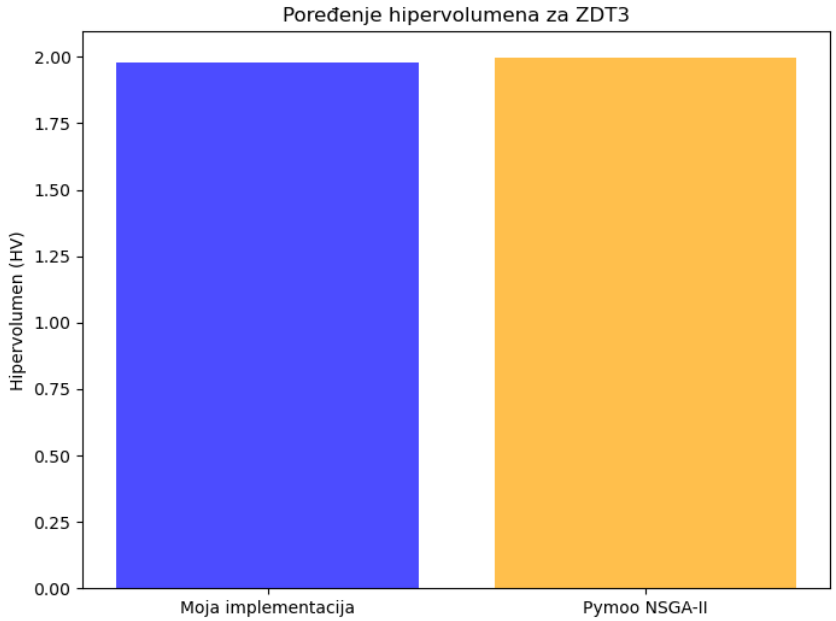
Rezultati hipervolumena nam omogućavaju da procenimo efikasnost i uniformnost distribucije rešenja na Pareto frontu. Veći hipervolumen ukazuje na bolji raspored rešenja i bolju pokrivenost prostora ciljeva.




Slika 1: Poređenje hipervolumena za $ZDT1$ funkciju.




Slika 2: Poređenje hipervolumena za $ZDT2$ funkciju.

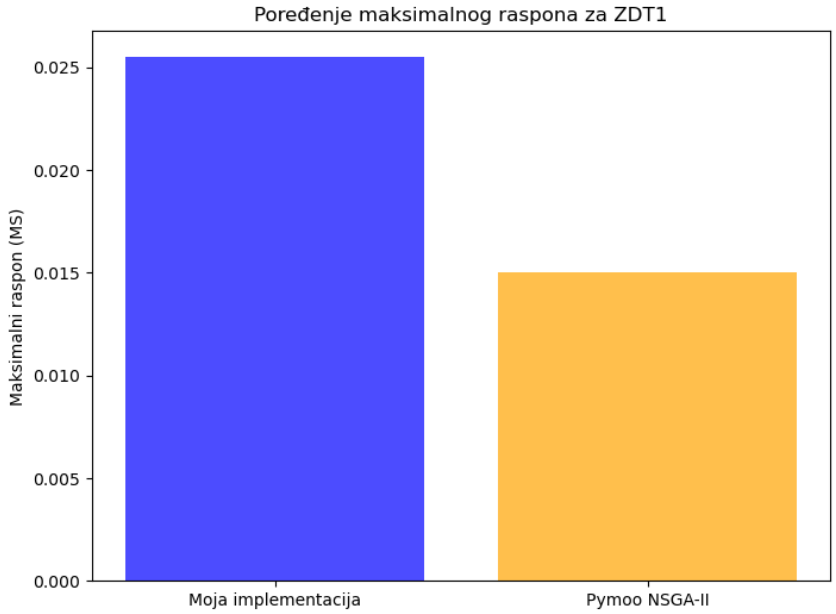



Slika 3: Poređenje hipervolumena za $ZDT3$ funkciju.

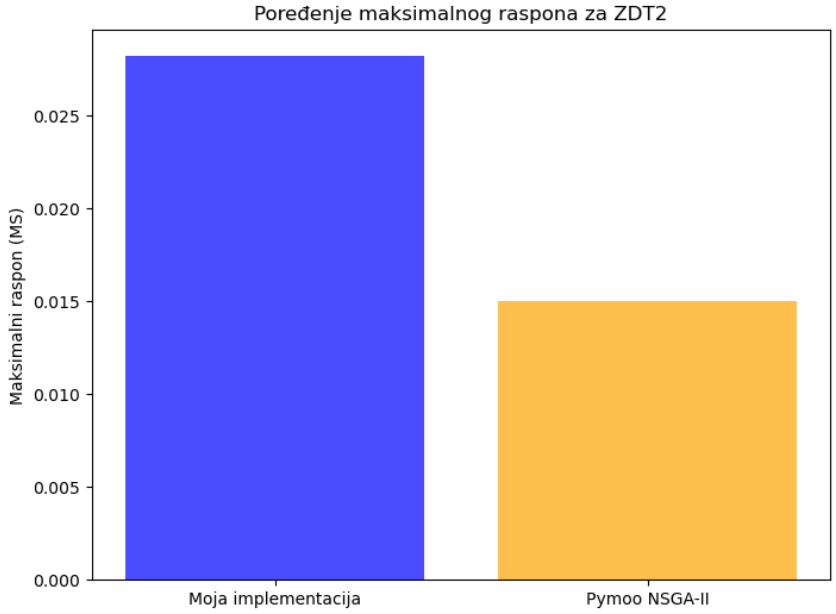
U poređenju hipervolumena između moje implementacije NSGA-II i pymoo implementacije, rezultati pokazuju gotovo identične vrednosti, s tim da je moj hipervolumen za veoma malu vrednost manji u svim testiranim funkcijama, što je i očekivano. To ukazuje na to da je moj algoritam uspešno postigao visok nivo pokrivanja prostora ciljeva, odnosno da je Pareto front ravnomerno raspoređen i da su rešenja dobar odraz kvaliteta.

### 5.1.2 Maksimalni raspon (MS)

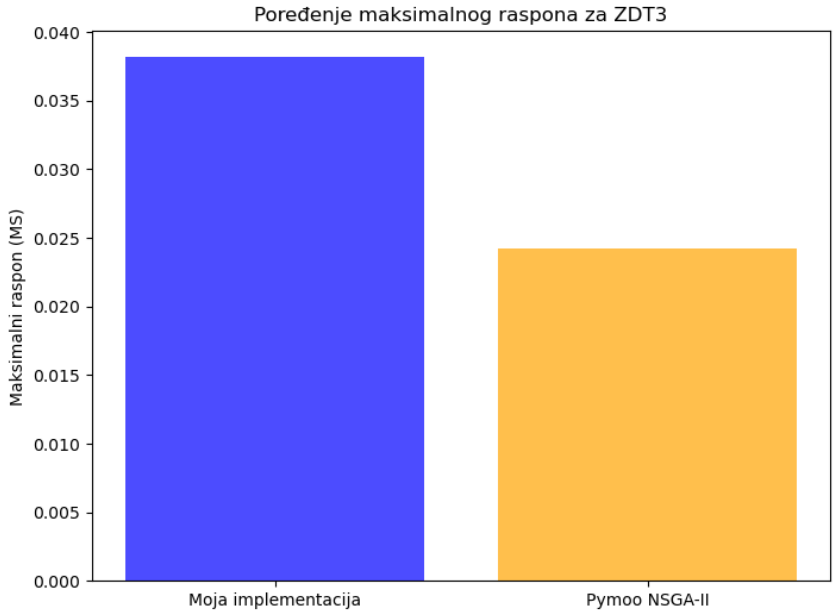
Maksimalni raspon (engl. *maximal spread*) meri udaljenost između najudaljenijih tačaka na Pareto frontu po svakom cilju. Ovo nam daje uvid u to kako su rešenja raspoređena duž fronta, tj. koliko je front širok u svakom cilju. Veća vrednost MS ukazuje na bolju pokrivenost ciljnog prostora, dok manja vrednost može značiti da su rešenja koncentrisana u užem delu fronta.




Slika 4: Poređenje maksimalnog raspona za $ZDT1$ funkciju.




Slika 5: Poređenje maksimalnog raspona za $ZDT2$ funkciju.

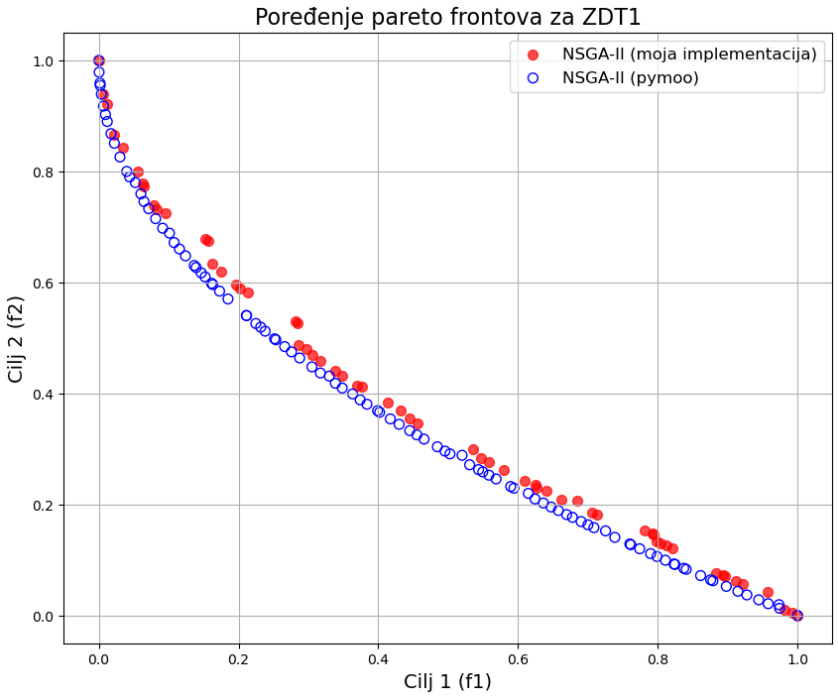



Slika 6: Poređenje maksimalnog raspona za $ZDT3$ funkciju.

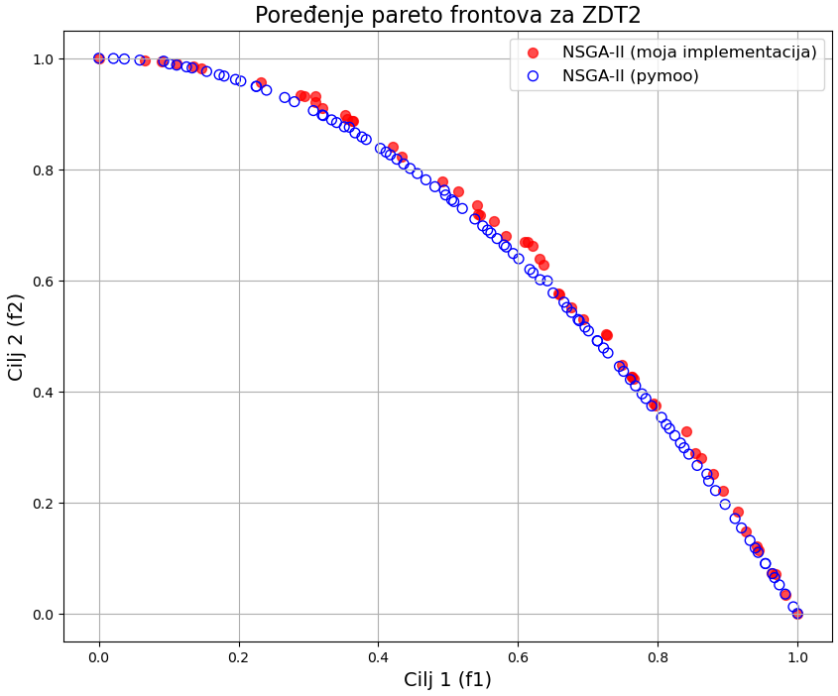
Rezultati maksimalnog raspona pokazuju da su vrednosti mog NSGA-II algoritma značajno veće u poređenju sa rezultatima pymoo implementacije. Ovo sugeriše da su rešenja generisana mojim algoritmom koncentrisanija u širem delu Pareto fronta, što znači da postiže bolju pokrivenost u širini za razliku od pymoo implementacije.

### 5.1.3 Uniformnost distribucije rešenja

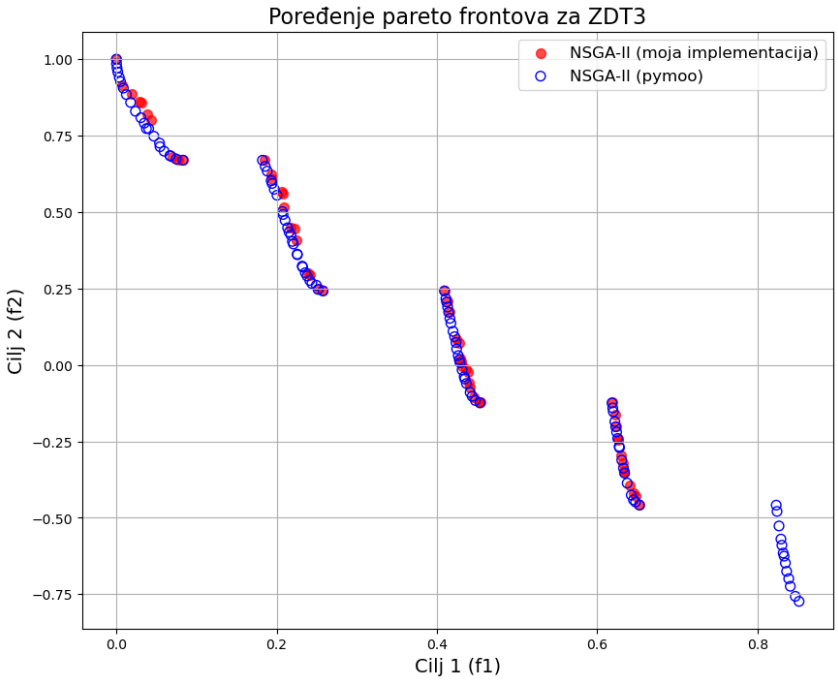
Uniformnost distribucije rešenja na Pareto frontu ukazuje na to koliko su tačke ravnomerno raspoređene. Dobar algoritam treba da postigne visok nivo uniformnosti kako bi osigurao kvalitetan i ravnomerno raspoređen Pareto front.




Slika 7: Poređenje distribucije rešenja za $ZDT1$ funkciju.




Slika 8: Poređenje distribucije rešenja za $ZDT2$ funkciju.




Slika 9: Poređenje distribucije rešenja za $ZDT3$ funkciju.

Poređenje uniformnosti distribucije rešenja na Pareto frontu između moje implementacije NSGA-II i pymoo implementacije pokazuje vrlo slične rezultate, sa minimalnim razlikama u distribuciji rešenja. Grafički prikazi pokazuju da se tačke na Pareto frontu vrlo dobro preklapaju, što ukazuje na to da su rešenja ravnomerno raspoređena. Ipak, može se primetiti da su rešenja u mojoj implementaciji nešto manje ravnomerno raspoređena u odnosu na pymoo, ali razlike nisu velike.

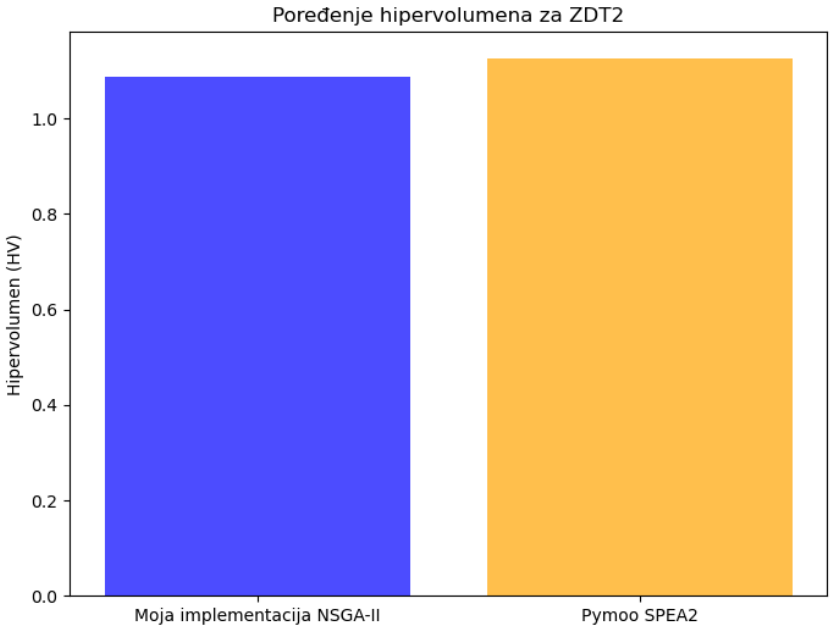
## 5.2 SPEA2

Moju implementaciju NSGA2 algoritma uporediću sa još jednim algoritmom koji je namenjen za rešavanje istih problema - u pitanju je **Evolucioni algoritam sa Pareto snagom 2** (engl. Strenght Pareto Evolutionary Algorithm - SPEA2). SPEA2 je evolucioni algoritam za višekriterijumske optimizacione probleme, poboljšana verzija SPEA algoritma. Kao i NSGA-II, koristi Pareto dominaciju za selekciju rešenja, ali uvodi dodatne mehanizme za očuvanje raznovrsnosti populacije. Ključne karakteristike uključuju unapređenu procenu gustine rešenja, eksteran arhiv za skladištenje nedominiranih rešenja i efikasnu strategiju skraćivanja arhiva. Ove osobine omogućavaju SPEA2 algoritmu da bolje aproksimira Pareto front i poboljša stabilnost optimizacije.

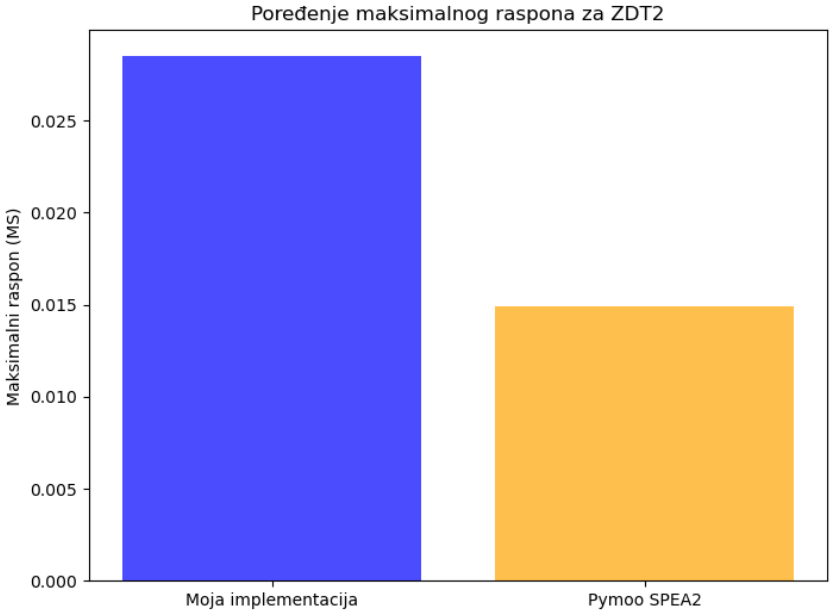
Rezultati su veoma slični kao kod poređenja moje implementacije sa bibliotečkom implementacijom NSGA-II algoritma. Na isti način je vršeno poređenje, jer u pymoo biblioteci postoji i SPEA2 algoritam.

Detaljnu implementaciju algortima SPEA2 neću ovde komentarisati, cilj ovog poređenja je samo da se uporedi moja implementacija sa još nekim algoritmom za višekriterijumske optimizacione probleme.

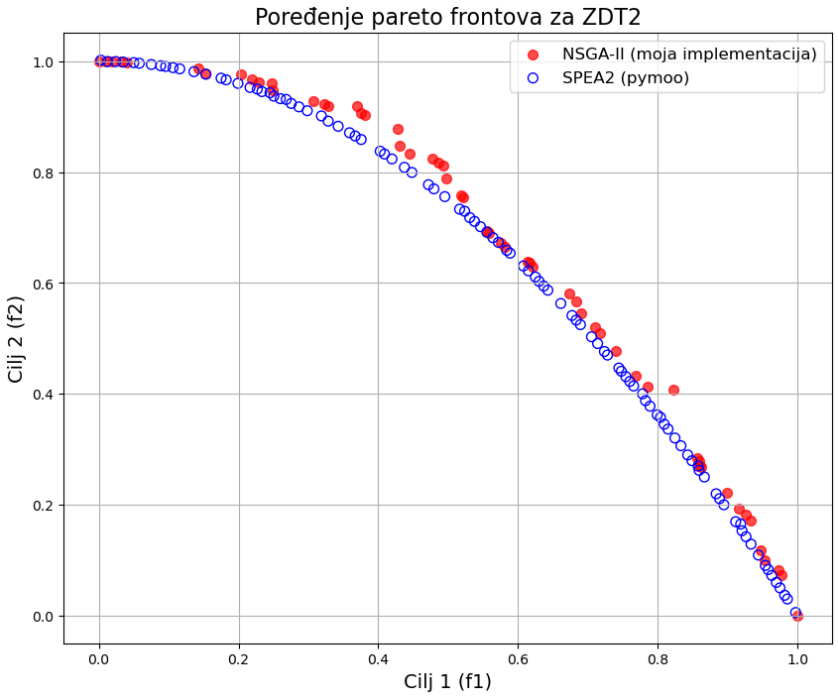
Prikazani su rezultati poređenja za funkciju ZDT2.




Slika 10: Poređenje hipervolumena za $ZDT2$ funkciju.




Slika 11: Poređenje maksimalnog raspona za $ZDT2$ funkciju.




Slika 12: Poređenje distribucije rešenja za $ZDT2$ funkciju.

Primećuju se iste razlike kao kod prethodnog poređenja. SPEA2 u biblioteci pymoo je implementiran tako da daje iste rezultate kao i NSGA-II iz iste biblioteke.

# 6 Diskusija o razlikama između implementacija

Iz prethodne analize, jasno je da postoji određena razlika između moje implementacije i pymoo implementacije NSGA-II algoritma.

Prva značajna razlika odnosi se na proces selekcije i ukrštanja. Pymoo koristi detaljno podešen operator selekcije roditelja koji omogućava efikasniju genetsku raznovrsnost populacije, dok moja implementacija primenjuje standardnu turnirsku selekciju sa određenom veličinom turnira. Ova razlika može uticati na način na koji se rešenja raspoređuju duž Pareto fronta, pri čemu pymoo obezbeđuje širu pokrivenost ciljnog prostora, dok moja implementacija može težiti ka centralnim delovima fronta.

Drugi faktor je način na koji se mutacija i ukrštanje sprovode. Obe implementacije koriste polinomijalnu mutaciju, samo što pymoo ima prilagođene distribucione parametre koji omogućavaju bolje balansiranje intenzifikacije i diverzifikacije rešenja, dok moja implementacija koristi fiksne parametre. Kao rezultat, pymoo može generisati rešenja koja su ravnomernije raspoređena po frontu, dok moja verzija pokazuje značajno veći maksimalni raspon (MS), što znači da obuhvata širu oblast ciljnog prostora. Veći MS u mojoj implementaciji ukazuje da algoritam uspeva da pronađe ekstremne tačke Pareto fronta, ali može dovesti do manje ravnomerne raspodele rešenja, pri čemu neka područja fronta mogu ostati slabije pokrivena.

Treći faktor koji doprinosi razlikama je algoritam sortiranja Pareto fronta. Pymoo implementacija koristi optimizovane procedure za brzo nedominirano sortiranje, koje mogu preciznije obrađivati dominaciju između rešenja i time omogućiti bolju raspodelu tačaka. U mojoj implementaciji, iako se koristi sličan algoritam, moguće su manje razlike u načinu

rangiranja i dodeljivanja distance gužve (*crowding distance*), što može dovesti do drugačijeg rasporeda rešenja.

Dodatno, implementacije se razlikuju u brzini konvergencije, tj. efikasnosti. Pymoo implementacija brže postiže stabilne rezultate zbog optimizovanog upravljanja populacijom i mehanizama selekcije, što omogućava efikasnije širenje fronta. U mom slučaju, pokazalo se da povećanje broja generacija značajno poboljšava kvalitet rešenja, što sugeriše da je mojoj implementaciji potrebno više iteracija kako bi dostigla iste nivoe pokrivenosti kao pymoo.

# 7 Zaključak

U ovom radu implementiran je NSGA-II algoritam i upoređen sa postojećom pymoo implementacijom kroz metrike hipervolumena, maksimalnog raspona i uniformnosti distribucije rešenja na Pareto frontu. Rezultati analize pokazuju da moja implementacija NSGA-II postiže gotovo identične vrednosti hipervolumena kao pymoo implementacija. Ovo potvrđuje da algoritam uspešno generiše kvalitetna rešenja i pokriva relevantan deo prostora ciljeva.

Međutim, maksimalni raspon rešenja u mojoj implementaciji pokazuje značajno više vrednosti u poređenju sa pymoo, što sugeriše da se rešenja nalaze u širem delu Pareto fronta. Veći MS sugeriše da algoritam uspešno pronalazi ekstremne tačke fronta, ali može dovesti do manje ravnomerne raspodele rešenja. Ova razlika može biti posledica parametara selekcije i mutacije.

Jedno od ključnih zapažanja tokom eksperimentisanja jeste da povećanje broja generacija značajno utiče na performanse mog algoritma. Inicijalno sam koristila broj generacija 200, kako je navedeno u literaturi, ali su rezultati pokazali veća odstupanja u odnosu na pymoo implementaciju. Kada sam povećala broj generacija na 500, dobila sam značajno bolje rezultate, što sugeriše da pymoo algoritam brže konvergira u odnosu na moju implementaciju.

Mogući pravci daljeg unapređenja uključuju:

- Podešavanje hiperparametara algoritma kako bi se poboljšala širina pokrivenosti Pareto fronta.
- Optimizaciju selekcije i mutacije za ravnomerniju distribuciju rešenja.
- Istraživanje tehnika ubrzanja konvergencije kako bi algoritam dostizao kvalitetna rešenja u manjem broju generacija.
- Proširenje analize na još neke test funkcije i stvarne optimizacione probleme radi bolje procene performansi algoritma.
- Prilagođavanje algoritma za rešavanje diskretnih optimizacionih problema.

Ova analiza pokazuje da je implementirani NSGA-II algoritam konkurentan u odnosu na pymoo rešenje, ali da postoji prostor za poboljšanja, posebno u pogledu brzine konvergencije i širine pokrivenosti Pareto fronta.

# Literatura


[1] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, *A fast and elitist multiobjective genetic algorithm: NSGA-II*, https://ieeexplore.ieee.org/abstract/document/996017

[2] A. Seshadri, *A fast elitist multiobjective genetic algorithm*, https://www.academia.edu/download/53297141/NSGA_II.pdf

[3] Materijali sa kursa *Računarska inteligencija*, Matematički fakultet, Univerzitet u Beogradu

[4] M. M. N. Rafstedt, *Analyzing the Simulated Binary Crossover Operator in Multi-Objective Evolutionary Algorithms*, https://www.duo.uio.no/handle/10852/111478

[5] Y. Sato and M. Sato, *Using Dominated Solutions at Edges to the Diversity and the Uniformity of Non-dominated Solution Distributions in NSGA-II*, https://www.researchgate.net/publication/362563469_Using_Dominated_Solutions_at_Edges_to_the_Diversity_and_the_Uniformity_of_Non-dominated_Solution_Distributions_in_NSGA-II#fullTextFileContent

[6] pymoo: Multi-objective optimization in Python, https://github.com/anyoptimization/pymoo

[7] https://pymoo.org/algorithms/moo/nsga2.html

[8] E. Zitzler, M. Laumanns and L. Thiele, *SPEA2: Improving the Strength Pareto Evolutionary Algorithm*, https://www.research-collection.ethz.ch/handle/20.500.11850/145755