SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 4333

EVOLUCIJSKI ALGORITMI ZA VIŠEKRITERIJSKU I MNOGOKRITERIJSKU OPTIMIZACIJU

Leon Šamec

Zagreb, lipanj 2016.

Sadržaj

1. Uvod

U ovom Završnom radu obrađivat će se višekriterijska i mnogokriterijska optimizacija te pripadni evolucijski algoritmi. No, da bi se moglo započeti s pravom temom treba pojasniti evolucijske algoritme. Evolucijski algoritmi su algoritmi optimizacije i metaheuristike čiji se mehanizam zasniva na Darwinovoj teoriji evolucije. Glavna premisa je da matematičke funkcije (ili neki drugi formalno definirani objekti), kao i živa bića, mogu evoluirati ako ih se stavi u određeno okruženje. Okruženjem bi se smatrala populacija rješenja konkretnog problema zajedno s operatorima koji će na nju djelovati. Svojstvo evolucijskih algoritmima je da prolaze kroz iteracije, kao što u pravoj evoluciji živa bića prolaze kroz generacije. Iteracije algoritama će se povremeno zvati generacije. Ovdje se tema može suziti jer će se govoriti o podvrsti evolucijskih algoritama – genetskim algoritmima. Iako većina algoritama u nastavku u svom nazivu sadrži riječ evolucijski, oni su također i genetski. Genetskim algoritmima je svojstveno da određenoj jedinki (rješenju) pridružuje dobrotu, funkciju čiji će iznos mjeriti kvalitetu rješenja. Jedinke nakon dodjele dobrote prolaze kroz selekciju (odabir najboljih jedinki po nekoj strategiji), a na odabrane djeluju operatori rekombinacije i mutacije. Nakon većeg broja iteracija možemo očekivati da će prosječni iznos dobrote porasti tj. da ćemo dobivati sve bolja rješenja. Građa samih jedinki se može razložiti na genotip i fenotip. Genotip bi bio dio jedinke na koji će djelovati operatori križanja i mutacije, a fenotip sam iznos rješenja u nekom obliku. Bitna komponenta genetskog algoritma je i zaustavni kriterij. U ovom radu će se koristiti broj generacija, ali postoje i druge strategije. Također, neće biti razlike između genotipova i fenotipova jer će informacije o jedinki biti zapisane u obliku vektora realnih brojeva.

2. Višekriterijski problem

2.1. Definicija problema

Klasični genetski algoritam radi s jednodimenzionalnom funkcijom dobrote, dakle svakoj se jedinki pridaje skalarna veličina koja označava kvalitetu jedinke. Takav tip optimizacije naziva se jednokriterijska optimizacija. U višekriterijskoj optimizaciji funkcija dobrote je višedimenzionalna, dakle imamo više kriterija po kojima moramo optimizirati rješenje. Iznos svih komponenata dobrote dolazi od istih parametara, dakle povećnjem jedne komponente može doći, a najčešće i dolazi, do smanjenja neke druge komponente. To je jedan od glavnih problema s kojima se višekriterijska optimizacija mora nositi. Naime, taj se problem ne može riješiti, nego se treba zaobići i pronaći kompromise (engl. *trade-off)* između iznosa komponenata dobrote i zatim odabrati najpovoljniju za zadani problem. Ovime se dolazi do pitanja kako uopće definirati slučaj kad je jedna jedinka bolja od druge, budući da nemamo skalare koje jednostavno možemo staviti u relacije veće, manje i jednako. Postoje neke tehnike kod kojih se svakom kriteriju daje određena težina i onda se svi zajedno sumiraju [2]. Primijenivši takvu tehniku višekriterijski problem se pretvara u jednokriterijski jer dobivamo dobrotu u obliku skalara. Za višekriterijski problem će nam biti potrebna nova relacija definirana nad vektorima istih dimenzija, a zove se dominacija. Naime, jedan vektor dominira drugi ako ima barem jednu odgovarajuću komponentu bolju (to mogu biti ili veće ili manje vrijednosti, ovisno želimo li maksimizirati ili minimizirati po nekom kriteriju), a ostale komponente jednake [1]. Na primjer, vektor (3,5,4) dominira vektor (2,5,4), ali nije u relaciji dominacije s vektorom (2,6,4). Ta se relacija može opisati i nekim karakterističnim matematičkim svojstvima. Relacija dominacije nije refleksivna, drugim riječima, jedan određeni vektor ne dominira sam sebe. Također, ta relacija nije niti simetrična, drugim riječima, ako vektor a dominira vektor b, tada vektor b sigurno ne dominira vektor a. I na kraju, relacija dominacije je tranzitivna, drugim riječima, ako vektor a dominira vektor b i vektor b dominira vektor c, onda vektor a dominira vektor c. Uočimo da je ovdje dobrota jedinke vektor koji promatramo. Nju ćemo staviti u relaciju s dobrotama drugih jedinki. Uočimo dalje razlog zašto se

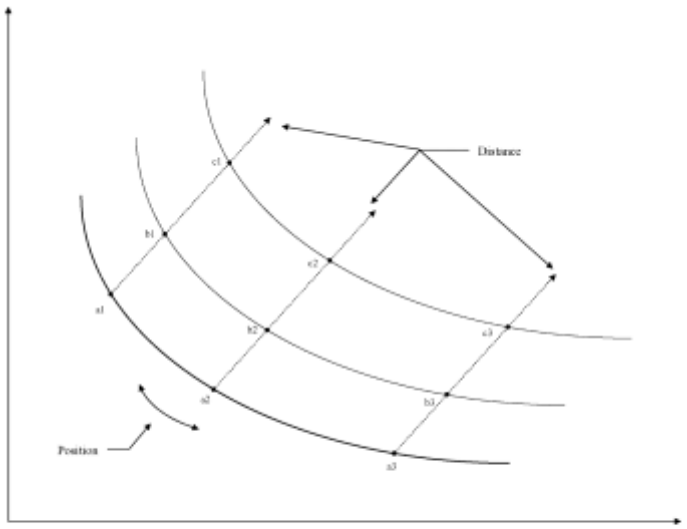
relacija zove baš dominacija. Uzmimo dva vektora kod kojih je jedan bolji od drugog po svim komponentama. Za njega bi se i bez formalne definicije dominacije moglo reći da dominira nad drugim vektorom. Nadalje, bitno je uočiti da uvijek postoje vektori istih dimenzija koji se ne mogu staviti u relaciju dominacije. To su dva vektora kod kojih jedan vektor ima neku određenu komponentu bolju od drugog vektora, a neku drugu lošiju. Relacija dominacije će nam pomoći u određivanju je li neka jedinka bolja ili lošija od neke druge, a to dosad nismo mogli odrediti. Sad kad znamo relaciju dominacije možemo definirati jedan karakteristični skup. To je skup svih vektora unutar prostora pretraživanja koji nisu dominirani niti od jednog drugog vektora unutar tog skupa (sjetimo se da vektor ne može sam sebe dominirati). Za te vektore vrijedi pravilo – ako poboljšamo jednu komponentu, a želimo da je vektor i dalje u našem željenom prostoru, onda sigurno moramo pogoršati neku drugu komponentu. Taj se skup u sklopu višekriterijske optimizacije zove Pareto optimalna fronta, a vektori u njemu se zovu vektori dobrote (engl. *objective vectors*) [1]. Skup pripadnih vektora parametara se naziva Pareto optimalni skup. Bitno je napomenuti da vektori između kojih će se određivati dominiranost neće biti svi vektori unutar prostora pretraživanja, nego samo trenutni vektori u populaciji. Svi algoritmi višekriterijske optimizacije rade tako da postupno grade Pareto optimalnu frontu tj. neku njezinu aproksimaciju. U radu će svi problemi biti sastavljeni od funkcija realnih brojeva (R -> R), a onda će i vektori parametara (domena) i vektori dobrote (kodomena) biti vektori realnih brojeva. Također ćemo se ograničiti na funkcije bez dodatnih ograničenja tj. neće postojati parametri x i y koji će osim svojih domena imati ograničenje poput x + y > 0. U nastavku će se opisati karakteristična svojstva različitih višekriterijskih problema što uključuje klasifikaciju različitih tipova parametara, tipova kriterija, odnosa vektora parametara i vektora dobrote te klasifikaciju geometrija Pareto optimalne fronte.

2.2. Tipovi parametara

Pojedini parametri se mogu klasificirati po načinu kako njihova promjena utječe na dominiranost vektora dobrote. Uzmimo dva jednaka vektora dobrote y11 i y2 koja su dobivena djelovanjem funkcije dobrote na vektore parametara x1 i x2. Ako

promjenom parametra $x1_i$ u vektoru x1 dobijemo novi vektor dobrote y12 koji dominira, je dominiran ili je jednak vektoru y2 tada se parametar $x1_i$ naziva parametar udaljenosti (engl. *distance parameter*) *(Slika 2.1).* Ako pak promjenom parametra $x1_i$ dobijemo vektor dobrote y12 koji je nije u relaciji dominacije s vektorom y2 ili mu je jednak tada se parametar $x1_i$ zove parametar pozicije (engl. *position parameter*). Parametar koji nije niti parametar pozicije niti parametar udaljenosti naziva se miješani parametar (engl. *mixed parameter*). Njegovom promjenom može se dobiti dominirani, dominirajući ili pak neusporediv vektor dobrote [1].

Parametar koji nije niti parametar pozicije niti parametar udaljenosti naziva se miješani parametar (engl. *mixed parameter*). Njegovom promjenom može se dobiti dominirani, dominirajući ili pak neusporediv vektor dobrote [1].

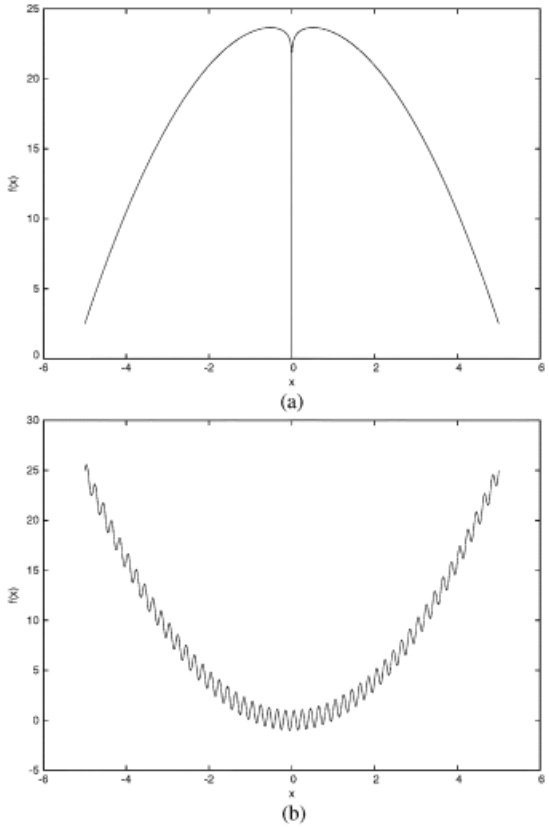


Prikaz parametara udaljenosti (engl. *distance*) i parametara pozicije (engl. *position*)

Slika 2.1. Tipovi parametara (preuzeto iz [1])

2.3. Tipovi kriterija

Kriteriji ili pojedinačne funkcije koje djeluju na vektor parametara mogu se svrstati u određene skupine s obzirom na njihova svojstva. Jedno od tih svojstva je modalnost *(Slika 2.2).* Funkcija je unimodalna ako ima samo jedan globalni optimum. Funkcija je pak multimodalna ako ima više lokalnih optimuma. Probleme s takvim funkcijama nazivamo multimodalni problemi. Oni su posebno teški za optimizacijske algoritme jer prilikom rješavanja takvih problema algoritmi mogu zapeti u lokalnim optimumima umjesto na napreduju prema globalnom optimumu. Posebni oblik multimodalne funkcije je zavaravajuća (engl. *deceptive*) funkcija. Takva funkcija ima najmanje dva optimuma, jedan pravi i jedan zavaravajući. Problem u rješavanju tih problema nastaje kad optimizacijski algoritam počne favorizirati zavaravajući optimum. Analogno, problemi sa zavaravajućim funkcijama zovu se zavaravajući problemi.



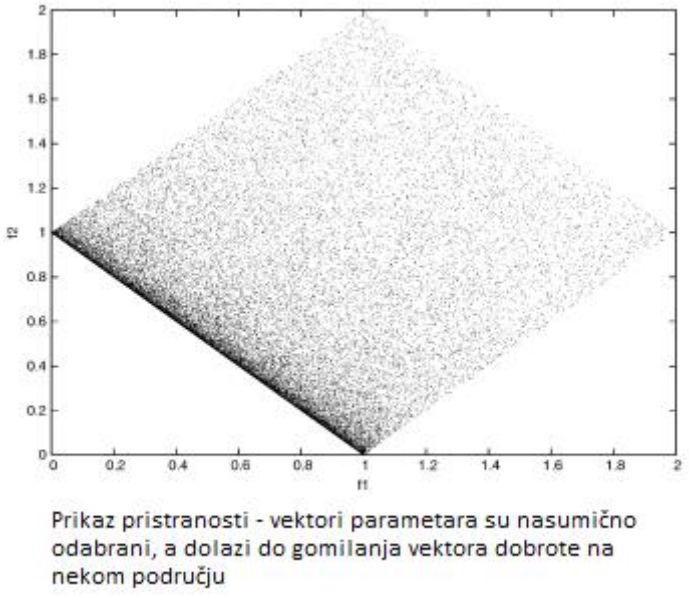
(a) zavaravajuća višemodalna funkcija
(b) višemodalna funkcija (nije zavaravajuća)

Slika 2.2. Tipovi kriterija (preuzeto iz [1])

2.4. Odnos vektora parametara i vektora dobrote

Kod optimizacijskih problema mogu se stvoriti razne podjele ako se gleda odnos vektora parametara i vektora dobrote (uočite da ovdje nije korištena riječ višekriterijski jer se iste podjele mogu primijeniti na jednokriterijske probleme). Jedna od tih je podjela na jedan-na-jedan (engl. *one-to-one*) i više-na-jedan (engl. *many-to-one*) probleme. Jedan-na-jedan problemi su oni kod kojih se svaki različiti vektor parametara preslikava u različiti vektor dobrote. Više-na-jedan problemi su pak oni kod kojih postoje dva različita vektora parametara koji se preslikavaju u isti vektor dobrote. Više-na-jedan problemi su teži za optimizacijske algoritme. Posebni oblik više-na-jedan problema je kad se u nekom od kriterija cijeli interval vektora parametara (u nekim komponentama) preslikava u isti vektor dobrote. To se onda zovu ravne regije (engl. *flat regions*). One su posebno teške za optimizacijske algoritme zbog nedostatka informacije o gradijentu, ili drugim riječima, algoritam ne može dobiti informaciju u kojem smjeru u domeni tog parametara treba ići tražiti optimum. Još je veći problem kad ravne regije pokrivaju veliki dio prostora pretraživanja. Optimumi u takvim problemima se nazivaju izolirani optimumi (engl. *isolated optima*). Druga podjela optimizacijskih problema je na to koliki se interval domene vektora parametara preslikava u koliki interval domene vektora dobrote. Ako ti intervali variraju, onda se može govoriti o problemu s pristranošću (engl. *bias*) *(Slika 2.3.)*. Ne postoji matematički definirana granica kad neki problem postaje problem s pristranošću, ali ako se nacrta graf problema može se aproksimativno odrediti. Svojstva ovisnosti parametara o kriteriju također čine jednu podjelu kriterija unutar problema. Definirajmo okruženje komponente vektora parametara x_i kao n-torku ostalih komponenata tog vektora parametara. Ako je za neku konkretnu vrijednost x_i u svim mogućim okruženjima vrijedi da je vrijednost komponente vektora dobrote k_j bolja ili jednaka u odnosu na sve ostale vrijednosti k_j sa pripadnim svim ostalim vrijednostima x_i u podudarajućim okruženjima, onda možemo reći da je x_i odvojiv (engl. *separable*) s obzirom na kriterij k_j. Ako prethodno ne vrijedi, onda je x_i neodvojiv (eng. *nonseparable*) s obzirom na k_j. Ako je svaki parametar koji utječe na kriterij k_j ujedno i odvojiv s obzirom na njega, onda je k_j odvojiv kriterij (engl. *separable objective*). Ako prethodno ne vrijedi, onda je k_j neodvojiv kriterij (engl. *nonseparable objective*). Ako su svi kriteriji u nekom problemu odvojivi, onda je to

odvojiv problem (engl. *separable problem*). Ako prethodno ne vrijedi, onda je to neodvojiv problem (engl. *nonseparable problem*) [1]. Odvojivi kriteriji se mogu optimizirati zasebno parametar po parametar. Konačna aproksimacija Pareto optimalnog skupa (zapravo aproksimacija njegovog podskupa) je onda kartezijev produkt dobivenih optimalnih skupova parametara. Nalaženje točaka Pareto optimalne fronte je znatno lakše kod odvojivih problema nego kod neodvojivih, baš zbog ovakvog posebnog pristupa [1].



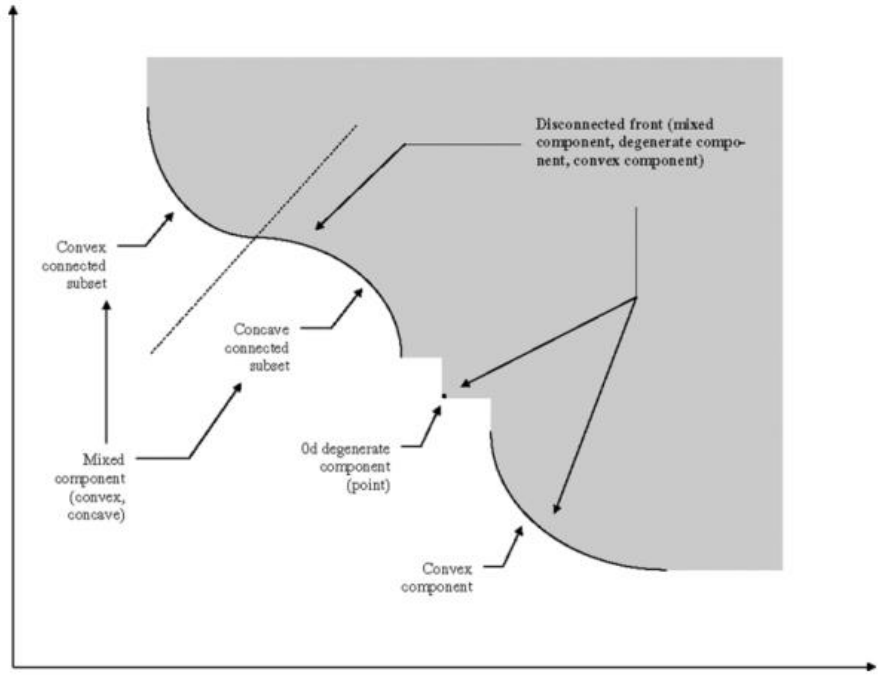
Slika 2.3. Problem s pristranošću (preuzeto iz [1])

2.5. Geometrija Pareto optimalne fronte

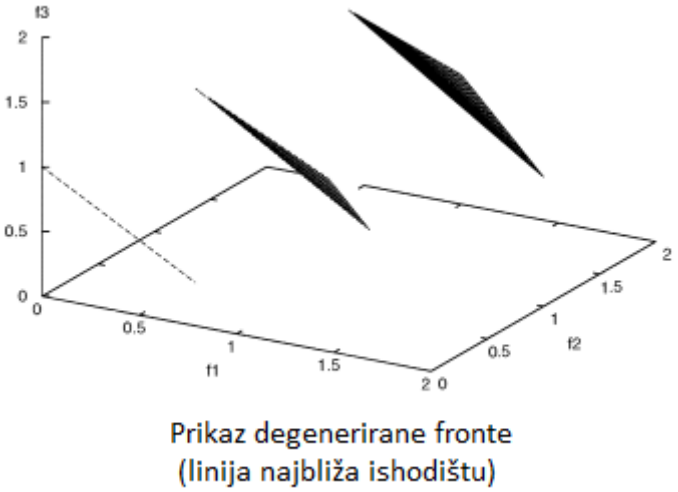
U jednokriterijskim problemima Pareto optimalna fronta je samo točka dok u višekriterijskim ona poprima različite višedimenzionalne oblike. Samo kod višekriterijskih problema s dva i tri kriterija Pareto optimalnu frontu možemo prikazati na grafu. Za probleme s više kriterija bi se mogle definirati analogne geometrije, ali one se ne bi mogle vizualizirati. Dva osnovna oblika geometrije fronte su konveksna i konkavna fronta (Slika 2.4.). Konveksna je ona fronta čija je izbočina okrenuta prema ishodištu, a kod konkavne je ona okrenuta od ishodišta. Miješana fronta (engl. *mixed front*) je ona fronta koja je djelomično konveksna i djelomično konkavna. Poseban oblik fronte je degenerirana fronta. To je fronta kod

koje je dimenzija geometrije manja od broja kriterija (npr. za problem s dva kriterija geometrija fronte je pravac). Takve fronte mogu stvarati problem u dijelovima optimizacijskih algortama koji se brinu za raspršenost točaka unutar konstruirane Pareto optimalne fronte. Fronte koje se ne mogu prikazati kao kontinuirana krivulja (tj. nisu kontinuirani skup) nazivaju se nekontinuirane fronte.

Kod konstrukcije ispitnih problema za višekriterijske algoritme teži se da ti problemi zajedno imaju što više različitih, dosad navedenih, karakteristika. Također se teži i da svaki problem zasebno ima što više tih karakteristika [1].



Slika 2.4. Geometrije fronti (preuzeto iz [1])



Slika 2.5. Degenerirana fronta (preuzeto iz [1])

2.6. Mnogokriterijski problem

Mnogokriterijski problemi se definiraju kao problemi s četiri i više kriterija. Za njih je karakteristično da se njihova Pareto optimalna fronta ne može prikazati na grafu. Kod njih se javljaju i problemi vezani uz vizualizaciju i izvođenje optimizacijskih algoritama. Veliki postotak populacije kod evolucijskog algoritama koji rješava mnogokriterijski problemom je nedominiran [3]. To stvara teškoće jer se uvelike gubi informacija o dominiranosti (informacija o lošijim genotipovima) koja može biti ključna za usmjeravanje pretraživanja. Mjera raznovrsnosti rješenja (engl. *diversity measure*) postaje vremenski skupa za izračunavanje s porastom broja kriterija, a ona je bitna za održavanje pokrivenosti Pareto optimalne fronte. Rekombinacija jedinki može biti nedjelotvorna jer raste udaljenost između jedinki u prvoj fronti (trenutnoj aproksimaciji Pareto optimalne fronte), a dvije jako udaljene jedinke mogu dati potomak koji je jako udaljen od obje jedinke čime pada djelotvornost rekombinacije. Postoje neke metode poput ograničavanja roditelja (engl. *mating restriction*) [2] koje mogu pomoći pri rješavanju tog problema. Kako raste broj kriterija, tako je i potreban veći broj jedinki za pokrivanje Pareto optimalne fronte, a to povlači i teži odabir odgovarajuće trade-off točke. Metrika za mjerenje performansi algoritama postaje vremenski skupa za izračunavanje sa porastom

broja kriterija. Na primjer, cijena izračunavanja jedne metrike raste eksponencijalno s brojem kriterija. I na kraju, vizualizacija Pareto optimalne fronte postaje jako teška, budući da se ne može prikazati na grafu. Problemi s velikim postotkom nedominiranih jedinki, skupom mjerom raznolikosti i neučinkovitom rekombinacijom mogu se ublažiti određenim promjenama u metodologiji algoritama. Veliki broj mnogokriterijskih problema u praksi sadrži degenerirane fronte, dakle problem s velikim brojem kriterija često degenerira u onaj s 2 ili 3 kriterija. Identifikacijom redundantnih kriterija i primjenom određenih postupaka početni mnogokriterijski problem se u konačnici može riješti višekriterijskim optimizacijskim algoritmom [3]. Ugradnjom postupaka poput PCA (*principal component analysis*) u neke višekriterijske optimizacijske algoritme poput NSGA-2 mnogokriterijski problemi se mogu uspješno rješavati, čak i oni do 50 kriterija. [3] Jedna posebna tehnika se ugrađuje u algoritme specijalizirane za mnogokriterijsku optimizaciju. Naime, umjesto da se koristi mjera udaljenosti između jedinki koja onda nagrađuje ili kažnjava jedinke i da se tako spontano stvori pokrivenost Pareto optimalne fronte, u algoritam se ugrađuju vanjski mehanizmi koji osiguravaju da Pareto optimalna fronta bude pokrivena. U nastavku će biti opisana dva takva. Prvi mehanizam stvara uniformno raspoređene smjerove pretraživanja i tako osigurava pokrivenost Pareto optimalne fronte. Njega koristi algoritam MOEA/D [4]. Drugi mehanizam koristi unaprijed definirane referentne linije pomoću kojih se osigurava da u populaciji uvijek ostanu jedinke vezane na pojedine linije. Ako te linije rasporedimo uniformno, a jedinku najbližu pojedinoj liniji proglasimo vezanom na nju, dobivamo pokrivenost cijele Pareto optimalne fronte. Taj mehanizam koristi algoritam NSGA-3 [3] i on će biti detaljno analiziran u nastavku. Treba napomenuti da će se u radu ponekad koristiti izraz višekriterijski problem za probleme s dva ili više kriterija (dakle i za mnogokriterijske probleme), a na što se točno misli bi trebalo biti vidljivo iz konteksta.

3. Osnovni gradivni blokovi algoritama za višekriterijsku i mnogokriterijsku optimizaciju

3.1. Nedominirano sortiranje

Nedominirano sortiranje je osnovni gradivni blok većine algoritama za višekriterijsku optimizaciju u kojemu se vidi stvarna korisnost relacije dominacije. Pomoću njega se odvajaju skupovi jedinki, nazvani fronte, između kojih se doista može reći da je jedan skup bolji od drugoga. Osnovna inačica tog gradivnog bloka (podalgoritma) je opisana u nastavku. Unutar populacije se pronađu jedinke koje nisu dominirane niti od jedne druge jedinke unutar te populacije. Njih se proglasi prvom frontom, najboljim jedinkama, unutar koje se pak ne može reći da je jedna jedinka bolja od druge. Treba primijetiti da unutar tog skupa također niti jedna jednika ne dominira neku drugu. Taj se skup zatim izbaci iz populacije. Postupak se ponavlja dok sve jedinke unutar populacije nisu smještene u neku od fronti tj. dok populacija nije prazna (naravno treba čuvati kopiju populacije za daljnje korake algoritama). Nakon eliminacije prve fronte, sljedeći nedominirani skup se proglasi drugom frontom i tako redom. Postoji poboljšana inačica algoritma koja ima manju složenost, a daje iste rezultate. Složenost osnovne verzije nedominiranog sortiranja je $O(MN^3)$, gdje je M broj kriterija, a N veličina populacije. Složenost poboljšane inačice je $O(MN^2)$. Pseudokod i programski kod poboljšane inačice je dan u nastavku [5]. *(Slika 2.5., Slika 2.6.)*

```
fast-non-dominated-sort(P)
for each p ∈ P
   S_p = ∅
   n_p = 0
   for each q ∈ P
      if (p ≺ q) then                    ako p dominira q
         S_p = S_p ∪ {q}                 dodaj q u skup rješenja koje p dominira
      else if (q ≺ p) then
         n_p = n_p + 1                   uvećaj dominacijski brojač od p
   if n_p = 0 then                       p pripada prvoj fronti
      p_rank = 1
      F_1 = F_1 ∪ {p}
i = 1                                    inicijaliziraj brojač fronti
while F_i ≠ ∅
   Q = ∅                                 koristi se za spremanje pripadnika iduće fronte
   for each p ∈ F_i
      for each q ∈ S_p
         n_q = n_q − 1
         if n_q = 0 then                 q pripada idućoj fronti
            q_rank = i + 1
            Q = Q ∪ {q}
   i = i + 1
   F_i = Q
```

Slika 3.1. Pseudokod bržeg nedominiranog sortiranja (preuzeto iz [5])

```
private void nonDominatedSortingFast() {
    List<Individual> populationCopy = new ArrayList<Individual>(populationR);
    for (int i = 0; i < (2 * populationSize); i++) {
        for (int j = 0; j < (2 * populationSize); j++) {
            if (j == i)
                continue;
            Individual unit1 = populationCopy.get(i);
            Individual unit2 = populationCopy.get(j);
            if (firstIsDominatingSecond(unit1.results, unit2.results)) {
                unit1.dominates.add(unit2);
            }
            if (firstIsDominatingSecond(unit2.results, unit1.results)) {
                unit1.ni++;
            }
        }
    }
    Integer noOfFront = 1;
    while (true) {
        List<Individual> front = new ArrayList<Individual>();
        for (Individual unit : populationCopy) {
            if (unit.ni == 0) {
                unit.rank = noOfFront;
                front.add(unit);
            }
        }
        if (front.size() > 0) {
            for (Individual unit : front) {
                for (Individual unit1 : unit.dominates) {
                    unit1.ni--;
                }
            }
            fronts.put(noOfFront, front);
            populationCopy.removeAll(front);
            noOfFront++;
        } else {
            break;
        }
    }
}
```

Slika 3.2. Java kod - Brzo nedominirano sortiranje

3.2. Selekcija pomoću niše

Uzmimo neku frontu, skup nedominiranih jedinki. Između njih zasad ne možemo reći da je neka bolja od neke druge, u smislu da neka ima veći prioritet biti selektirana u iduću generaciju. Ovdje se moramo sjetiti jednog cilja višekriterijskog optimizacijskog algoritma, a to je očuvanje raznolikosti, ili raspršenosti, tj. aproksimativno pokrivanje cijele Pareto optimalne fronte. Da bi taj cilj ostvarili, trebat ćemo nekako kažnjavati skupljanje jedinki u malom prostoru, ili nagrađivati njihovu veću udaljenost. Promatrajmo tu udaljenost, za potrebe definiranja, kao Euklidsku. Nišom bi se nazivao taj relativno mali prostor unutar kojeg se skupljaju

jedinke. Ovim mehanizmom ćemo jedinke unutar iste fronte moći sortirati po važnosti za selektiranjem u iduću generaciju. Naime, većina višekriterijskih optimizacijskih algoritama koristi neki oblik tog mehanizma [3, 4, 5, 6], a konkretni postupci će biti analizirani kasnije.

3.3. Rekombinacija i mutacija

Sadržavanje operatora rekombinacije i mutacije je razlog zašto u nastavku analizirani višekriterijski optimizacijski algoritmi pripadaju evolucijskim algoritmima (MOEA) *(Slika 3.3.)*. Ti operatori omogućuju pomicanje po prostoru pretraživanja. Rekombinacija je mehanizam spajanja genotipa dvaju jedinki koje su prvobitno prošle selekciju; dakle to je binarni operator (kako je složeniji, zvat će se i mehanizam). U prirodi bi rekombinacija odgovarala križanju živih bića. Ako na neki način definiramo udaljenost između dvije jedinke (za primjer uzmimo Euklidsku udaljenost između genotipa), tada će rekombinacijom dobivena jedinka, ili dijete, imati vrlo vjerojatno manju udaljenost od roditelja nego neka jedinka s nasumično generiranim genotipom. Naravno, postupak križanja genotipa treba biti razuman da bi se prethodno ispunilo. Postoji izvjesna vjerojatnost da će dijete biti jedinka s dobrotom u rangu roditelja, ako ne i s boljom. U sklopu višekriterijskih problema moglo bi se reći da će dijete biti nedominirano od roditelja ili da će njih dominirati. Za primjer, definirajmo neki postupak križanja za koji se može reći da je razuman. Uzmimo komponente vektora parametara oba roditelja koji su na istim mjestima. Napravimo interval od manje komponente do veće i proširimo interval za 25% u oba smjera. Uniformnom raspodjelom generirajmo točku unutar tog intervala. Kada ponovimo postupak za sve komponente vektora parametara, dobivene točke predstavljaju vektor parametara djeteta.

Drugi operator (mehanizam) za pretraživanje je mutacija. Ona u prirodi odgovara mutaciji živih bića. Mogla bi se opisati kao translacija vektora parametara za neki vektor s malim iznosom. Definira se kao unarni operator jer se pomoćni vektor generira nasumično. Naravno, konačni vektor mora ostati u prostoru pretraživanja vektora parametara. Isto vrijedi i za rekombinaciju. Mutacija postoji kako bi se u gensku zalihu (skup svih trenutnih genotipova) unijela nova informacija. Bitna je

ako se rekombinacijom ne može unijeti nova informacija u gensku zalihu, što vrijedi za neke kombinacije tipa zapisa genotipa i strategije rekombinacije (npr. binarni zapis i križanje sa jednom točkom prekida). U algoritmima analiziranim u nastavku rada mutacija se koristi kako bi se povećao unos novih informacija u gensku zalihu i tako potencijalno dobile nedominirane ili čak istovremeno dominirajuće i nedominirane jedinke.

```
private List<Double> crossover(List<Double> inputList1,
        List<Double> inputList2) {
    List<Double> crossoverInputList = new ArrayList<Double>();
    Integer size = inputList1.size();
    for (int i = 0; i < size; i++) {
        Double greater;
        Double lower;
        Double arg1 = inputList1.get(i);
        Double arg2 = inputList2.get(i);
        if (arg1 > arg2) {
            greater = arg1;
            lower = arg2;
        } else {
            greater = arg2;
            lower = arg1;
        }
        Double result = lower + rand.nextDouble() * (greater - lower);
        crossoverInputList.add(result);
    }
    return crossoverInputList;
}

private List<Double> mutate(List<Double> inputList) {
    List<Double> mutatedInputList = new ArrayList<Double>();
    for (Double input : inputList) {

        Integer mutPow = rand.nextInt(60) - 30;
        Double delta = 0.0;
        if (mutPow >= 0) {
            delta = (((double) mutPow) / 100.0) * (uBound - input);
        } else {
            delta = (((double) mutPow) / 100.0) * (input - lBound);
        }
        Double mutatedInput = input + delta;
        mutatedInputList.add(mutatedInput);
    }
    return mutatedInputList;
}
```

Slika 3.3. Java kod - Mutacija i križanje

4. Procjena performansi algoritama

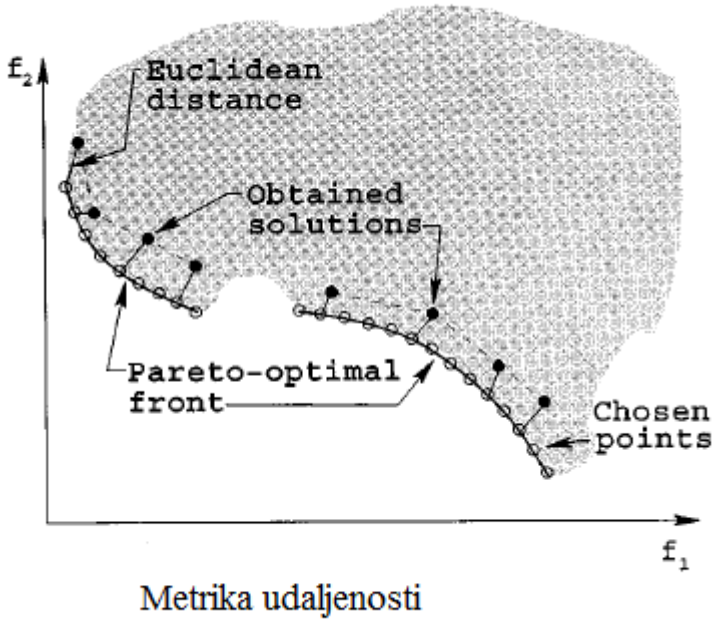
4.1. Metrika udaljenosti i metrika raznolikosti

U nastavku će se navesti i analizirati neki od postupaka procjene ostvarenja ciljeva višekriterijske optimizacije. Tri su osnovna cilja. Prvi je konvergencija prema Pareto optimalnoj fronti. Drugi je održavanje raznolikosti (raspršenja) među nedominiranim jedinkama [5]. Treći je pak što veći postotak nedominiranih jedinki u konačnoj populaciji.

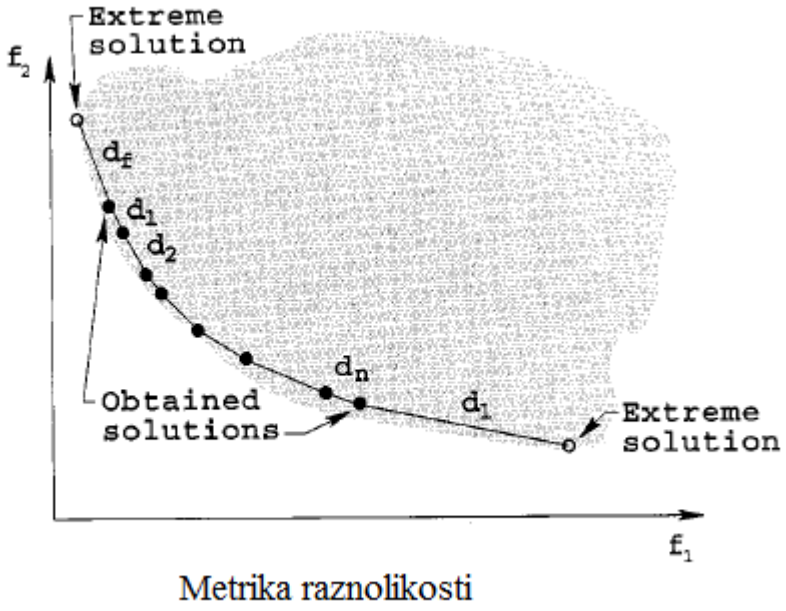
Prva metrika koja će se definirati je ona koja mjeri konvergenciju prema Pareto optimalnoj fronti. Naime, ta metrika se ne može upotrijebiti ako se ne zna stvarna Pareto optimalna fronta što znači da se ne može koristiti na proizvoljnom problemu. To neće biti preveliki problem jer će se ionako koristiti na testnim problemima kod kojih se uglavnom zna stvarna Pareto optimalna fronta. Metrika se zove metrika udaljenosti, a definirana je ovako *(Slika 4.1.).* Uzmimo određen broj točaka na stvarnoj Pareto optimalnoj fronti. Za svako dobiveno rješenje (vektor dobrote svake jedinke u konačnoj populaciji) nađimo Euklidsku udaljenost do najbliže odabrane točke na stvarnoj Pareto optimalnoj fronti. Uprosječimo sve dobivene udaljenosti i dobili smo iznos metrike za jedno pokretanje optimizacijskog algoritma. Često se kao mjera konvergencije uzima prosjek te metrike u više pokretanja. Najbolja vrijednost 0 će se poprimiti samo ako sva rješenja leže na odabranim točkama, a ne ako su samo na Pareto optimalnoj fronti. Budući da je metrika udaljenosti nenegativna i najbolja za vrijednost 0 može se zaključiti da algoritam bolje konvergira k Pareto optimalnoj fronti ako je iznos te metrike udaljenosti manji. Metrika je korištena za višekriterijske optimizacijske algoritme, ali ne i za mnogokriterijske.

Druga metrika mjeri raspršenost vektora dobrote jedinki u nedominiranom skupu (jedino je razumno uzeti prvu frontu). Nazvana je metrikom raznolikosti [5] *(Slika 4.2.).* Onako kako ćemo je definirati u nastavku, moći će se koristiti samo za problem s dva kriterija. Cijeli izračun se obavlja u prostoru vektora dobrote. Nađimo najbolju jedinku po prvom kriteriju. Nju ćemo zvati ekstremna jednika-1. Zatim, nađimo Euklidsku udaljenost od te jedinke do njoj najbliže. Nazovimo tu

novu jedinku prva jedinka, a udaljenost do nje df. Maknimo ekstremnu jedinku-1 iz nedominiranog skupa i ponavljajmo traženje, sada sa prvom jedinkom. Udaljenost od prve jedinke do njoj najbliže - druge jedinke nazovimo d1. Postupak se ponavlja sve dok se ne dođe do ekstremne jedinke-2. Ta jedinka je najbolja u drugom kriteriju. Udaljenost od zadnje jedinke do ekstremne jedinke-2 nazovimo dl. Sve ostale numerirajmo po broju jedinke od koje smo najbližu tražili. Metrika raznolikosti se računa po formuli u nastavku. Primijetimo da za računanje ove metrike ne trebamo znati stvarnu Pareto optimalnu frontu. Najbolji iznos metrike raznolikosti je 0, a i za nju vrijedi da algoritam ocjenjuje kao bolji ako je po iznosu manja. Iznos 0 se dobiva ako su rješenja u nedominiranom skupu uniformno raspoređena tj. ako su sve prethodno definirane udaljenosti jednake. Postoje neke tehnike kojima se metrika raznolikosti može proširiti na probleme s više kriterija (triangularization technique, Voronoi diagram approach) [5].



Slika 4.1. Metrika udaljenosti (preuzeto iz [5])

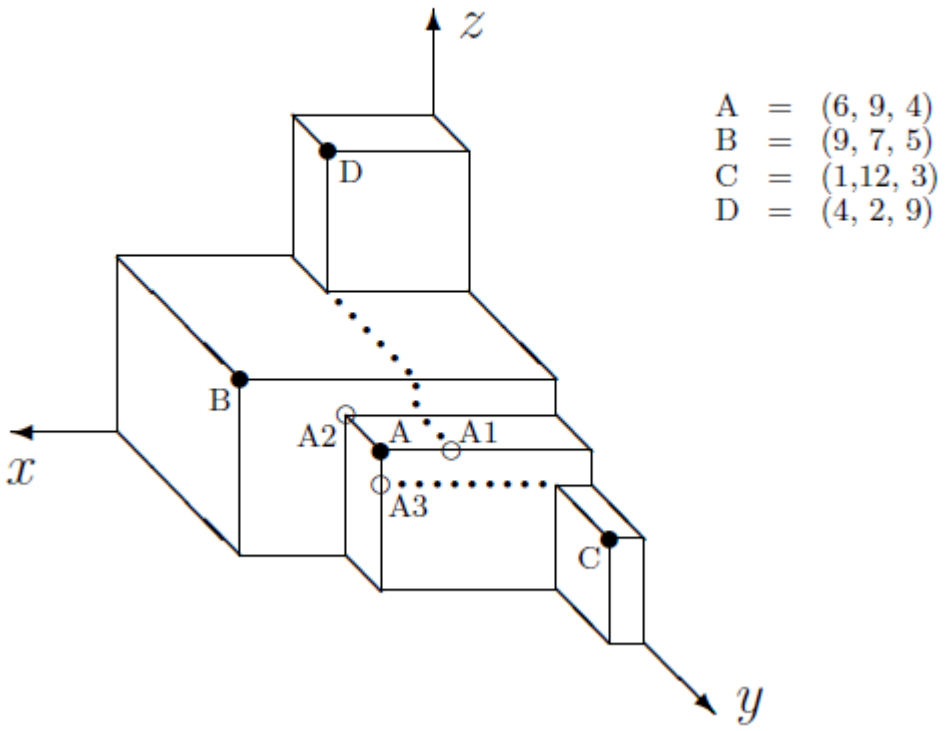


Slika 4.2. Metrika raznolikosti (preuzeto iz [5])

4.2. Hipervolumen indikator

Kako smo vidjeli u prethodnom poglavlju, navedene metrike su nastojale izmjeriti kvalitetu rješenja višekriterijskog problema po nekom svojstvu. Za razliku od njih postoje mjere, ili indikatori, koji u jednoj skalarnoj vrijednosti nastoje izreći kvalitetu rješenja po svim bitnim svojstvima, poput konvergencije, raspršenosti, relativne ukupne dobrote nedominiranih rješenja i zastupljenosti nedominiranih rješenja u populaciji. Jedan od trenutno najpopularnijih indikatora je hipervolumen (znan i kao S-metrika ili Lebesgue-ova mjera) [7]. On mjeri ukupan volumen kojeg čine sva nedominirana rješenja i neka referentna točka. Najčešće je to ishodište koordinatnog sustava. U problemu s dva kriterija svako rješenje i referentna točka bi činili pravokutnik. Iznos hipervolumena bi u tom slučaju bila površina unije svih pravokutnika (dakle ne čista suma površina pravokutnika). U problemu s tri kriterija bi se umjesto unije površina pravokutnika tražila unija volumena kvadra *(Slika 4.3.)* itd. Jedno njegovo dobro i bitno svojstvo je da je njegov iznos za prvu frontu uvijek veći od iznosa za drugu frontu, a iznos za drugu frontu je uvijek veći od iznosa za treću frontu itd. Matematički rečeno on je striktno monoton. Za razliku od metrika iz prethodnog poglavlja, hipervolumen se može koristiti za probleme s proizvoljnim

brojem kriterija (dakle i višekriterijske i mnogokriterijske probleme). Bitan problem tog indikatora je što vremenska cijena njegovog izračunavanja raste eksponencijalno s brojem kriterija. Osim kao indikator, hipervolumen se može koristiti i kao operator selekcije. Postoje više algoritama koji su u svoj process selekcije ugradili hipervolumen operator, poput SMS-EMOA i FV-MOEA. On radi tako da iz populacije izbacuje jedinke koje najmanje pridonose iznosu hipervolumena. Trenutno se proučavaju i istražuju algoritmi za njegovo izračunavanje, a neki su opisani u Bradstreet, 2011 [7]. Cilj istraživanja je što više smanjiti vremensku cijenu njegovog izračunavanja, a stoga se predlažu i neke aproksimativne tehnike [7].



Slika 4.3. Prikaz hipervolumena kod problema s 3 kriterija (preuzeto iz [7])

4.3. R2 indikator

Uz hipervolumen, drugi popularni indikator je R2 indikator [8]. Ta dva indikatora su po svrsi i po informaciji koju daju vrlo slični jer oba u svojem iznosu uključuju ocjenu bitnih svojstava rješenja višekriterijskog problema. U nastavku ćemo navesti neke razlike. R2 indikator je monoton, ali nije striktno monoton tj. njegov iznos kod prve fronte može biti jednak onom kod druge fronte, ali ne može biti manji. U tom smislu hipervolumen indikator je bolji od R2 indikatora. R2 indikator prednjači u vremenskoj cijeni potrebnoj za izračunavanje. Također je bolji u distribuciji iznosa među sličnim rješenjima. R2 indikator uniformnije raspoređuje svoj iznos od hipervolumen indikatora. Isto kao i hipervolumen koristi se kao operator selekcije. Neki algoritmi poput R2-EMOA i MOMBI ga ugrađuju u svoj process selekcije.

5. Algoritmi

5.1. NSGA

Algoritam NSGA [2] je evolucijski algoritam za višekriterijsku optimizaciju (MOEA; ne i za mnogokriterijsku) prve generacije [9]. Prvu generaciju MOEA karakterizira neelitistički pristup tj. prisutstvo vjerojatnosti (veće od 0) da će se trenutna najbolja jednika izgubiti prilikom selekcije. Do tog dolazi zbog stohastičkog svojstva njihovih selekcija [9]. Konkretno, NSGA radi sa zamjenskom (dijeljenom) dobrotom koja u sebi sadrži informaciju i o rangu rješenja (broju fronte kojemu pripada) i o njegovom okruženju (kažnjava se ako je blizu drugih rješenja). Na temelju te zamjenske dobrote obavlja se selekcija sa stohastičkim svojstvom - proporcionalna selekcija. Rang rješenja se dobiva pomoću nedominiranog sortiranja, a svojstvo okruženja jedinke pomoću identificiranja svih rješenja unutar neke udaljenosti tog rješenja. Zamjenska dobrota je skalar pa se može uspoređivati pomoću osnovnih relacija veće, manje i jednako za razliku od vektora dobrote. Njezino dodjeljivanje ide ovako. Prvo se odredi početna dobrota koja se dodijeli svim jedinkama u prvoj fronti. Zatim se definira *sigma shared* udaljenost od rješenja unutar koje će se tražiti susjedna rješenja, ali samo među onima u istoj fronti. Na temelju broja susjednih rješenja i ukupne udaljenosti od njih, svakoj se jedinki umanji početno dodijeljena dobrota. Zatim se za neki iznos umanji najmanja dobrota unutar trenutne fronte i dodjeli svim rješenjima u idućoj fronti, što je onda njihova početna dobrota. Postupak se ponavlja sve dok više nema rješenja bez dodijeljene dobrote. Proporcionalna selekcija [2] radi tako da se na temelju udjela zamjenske dobrote nekog rješenja u sumi zamjenskih dobroti svih rješenja gradi vjerojatnost izabiranja u iduću generaciju *(Slika 5.1.).* Primijetimo da s takvom selekcijom postoji vjerojatnost da izgubimo rješenje s najvećom dobrotom. Drugi problem NSGA je potreba za specificiranjem *sigma shared* udaljenosti, o kojemu znatno ovise performanse algoritma, jer se rješenja drugačije raspoređuju u prostoru kod različitih problema [5]. Naime, u radu Fonseca, 1993 [10] navodi se rješenje tog problema s automatskim dodjeljivanjem parametara *sigma shared* pomoću funkcije veličine populacije te maksimuma i

minimuma domena kriterija. I drugi parametri se mogu mijenjati, ali će algoritam imati slične performanse na većini problema uzevši neke pretpostavljene. Rekombinacija i mutacija se mogu obavljati proizvoljnim strategijama [2]. Neki drugi MOEA prve generacije su NPGA i MOGA [9]. Pseudokod je dan u nastavku, a detaljniji opis algoritma postoji u radu Čupić, 2013 [2].

Pseudokod NSGA:

```
inicijalizacija populacije;
dok uvjet zaustavljanja nije ispunjen{
        evaluacija jedinki;
        nedominirano sortiranje;
        dodjeljivanje dijeljene dobrote;
        selekcija;
        križanje;
        mutacija;
}
```

```
private static void relativeProportionalSelection() {
    List<Unit> survivors = new ArrayList<Unit>();
    while (survivors.size() < aliveParentsAfterSelection) {
        List<Double> fitness = new ArrayList<Double>();
        for (Unit unit : population) {
            fitness.add(unit.fitness);
        }
        Double minFitness = Collections.min(fitness);
        Double relativeFitnessSum = 0.0;
        for (Unit unit : population) {
            relativeFitnessSum += unit.fitness - minFitness;
        }
        for (Unit unit : population) {
            unit.probability = (unit.fitness - minFitness)/ relativeFitnessSum;
        }
        Double probabilitySum = 0.0;
        Double startingProbability = 0.0;
        for (Unit unit : population) {
            unit.lowerRange = startingProbability;
            unit.upperRange = startingProbability + unit.probability;
            startingProbability = unit.upperRange;
            probabilitySum += unit.probability;
        }
        Double hitNumber = rand.nextDouble() * probabilitySum;
        Unit unitForRemove = null;
        for (Unit unit : population) {
            if (unit.lowerRange <= hitNumber && unit.upperRange > hitNumber) {
                survivors.add(unit);
                unitForRemove = unit;
                break;
            }
        }
        population.remove(unitForRemove);
    }
    population.clear();
    population.addAll(survivors);
}
```

Slika 5.1. Java kod - Proporcionalna selekcija

5.2. NSGA2

Algoritam NSGA-2 [2, 4] je evolucijski algoritam za višekriterijsku optimizaciju (ne i mnogokriterijsku) druge generacije. Druga generacija MOEA je specifična po tome što sadrži elitizam tj. prilikom selekcije trenutno najbolje rješenje se ne može izgubiti. Njega NSGA-2 ostvaruje slijednom selekcijom po frontama; dakle prvo se u sljedeću generaciju prenosi prva fronta, zatim druga itd. Da bi se to omogućilo, NSGA-2 postupke rekombinacije i mutacije obavlja na početku iteracije te stvara pomoćnu populaciju dvostruko veću od početne (može se i na nekom drugom mjestu u iteraciji, ali efektivno treba biti prije selekcije). Kada neka fronta više ne može cijela stati u populaciju iduće generacije, provodi se poseban oblik selekcije

– grupirajuća turnirska selekcija (engl. *crowding tournament selection*) [2]. Ta selekcija u prioritet izabiranja uključuje položaj u prostoru u odnosu na druge jedinke tj. njihov stupanj izoliranosti (gleda se vektor dobrote). Dakle, to je selekcija pomoću niše kojom se ostvaruje raspršenost jedinki radi pokrivanja cijele Pareto optimalne fronte. Skalarna vrijednost koju dobiva svaka jedinka u toj zadnjoj uključenoj fronti naziva se grupirajuća udaljenost (engl. *crowding distance*) *(Slika 5.3.)* [2] i dodjeljuje se sljedećim pseudokodom.

Pseudokod dodjeljivanja grupirajuće udaljenosti:

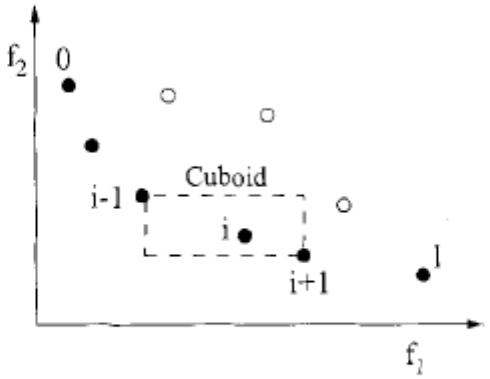
```
za svaku jedinku postaviti grupirajuću udaljenost na 0;
za svaki kriterij:
        sortirati jedinke po to vrijednosti tog kriterija;
        postaviti grupirajuću udaljenost prve i zadnje jedinke u toj sortiranoj listi na
        beskonačno;
        svim ostalim jedinkama uvećati grupirajuću udaljenost za razliku vrijednosti kriterija
        sljedeće i prethodne jedinke podijeljene s razlikom maksimalne i minimalne
        vrijednosti tog kriterija u listi jedinki;
```

Jedinke s većom grupirajućom udaljenosti imat će veći prioritet za izabiranje u iduću generaciju *(Slika 5.4.).* Sam odabir jedinki se obavlja k-turnirskom selekcijom. Postupak k-turnirske selekcije ide ovako. Iz željene populacije (ovdje zadnje uključene fronte) se nasumično odabere k jedinki između kojih se u sljedeću generaciju prenosi ona s najvećom dobrotom (ovdje s najvećom grupirajućom udaljenosti). Ta se jedinka zatim izbacuje iz trenutne populacije i postupak se ponavlja sve dok nova populacija nema zadovoljavajuć broj jedinki. Može se uočiti da je elitistička selekcija u drugoj generaciji MOEA velikim djelom deterministička za razliku od selekcije u prvoj generaciji MOEA koja je potpuno stohastička. Stohastički element elitističkoj selekciji daje k-turnirska selekcija. Detaljniji prikaz i analiza algoritma NSGA-2 se može pronaći u Deb, 2002. [5]. Pseudokod je prikazan u nastavku *(Slika 5.2.)*. Treba primijetiti da za izvođenje algoritma ne treba odrediti niti jedan poseban parametar (poput *sigma shared* u NSGA) što je veliki plus za NSGA-2. Drugim riječima, algoritam se može koristiti za rješavanje svih višekriterijskih problema bez njihovog prvobitnog pretjeranog analiziranja. Izvorna inačica NSGA-2 koristi bržu inačicu nedominiranog sortiranja. NSGA-2 je u prosjeku brži algoritam (manje je složenosti) od svog prethodnika NSGA te pokazuje bolje performanse po dosad navedenim metrikama [5].

Također, pokazuje bolje performanse i od drugih MOEA svoje (druge) generacije poput SPEA i PAES [5]. NSGA-2 je vrlo zastupljen algoritam i stoga postoje brojne modifikacije izvorne inačice.

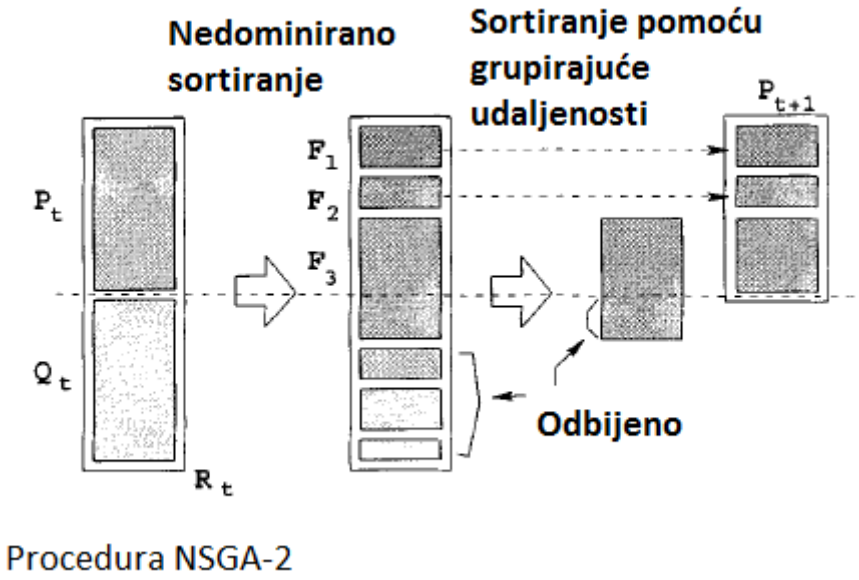
$R_t = P_t \cup Q_t$	spoji populaciju roditelja i populaciju djece
$\mathcal{F} =$ fast-non-dominated-sort(R_t)	$\mathcal{F} = (\mathcal{F}_1, \mathcal{F}_2, \ldots)$, sve nedominirane fronte R_t
$P_{t+1} = \emptyset$ and $i = 1$	
until $\lvert P_{t+1}\rvert + \lvert\mathcal{F}_i\rvert \leq N$	dok populacija roditelja nije popunjena
crowding-distance-assignment$(\mathcal{F}_i)$	izračunaj grupirajuću udaljenost u $\mathcal{F}_i$
$P_{t+1} = P_{t+1} \cup \mathcal{F}_i$	uključi i-tu nedominiranu frontu u populaciju roditelja
$i = i + 1$	provjeri uključivanje iduće fronte
Sort$(\mathcal{F}_i, \prec_n)$	sortiraj u padajućem redosljedu koristeći grupirajuću udaljenost
$P_{t+1} = P_{t+1} \cup \mathcal{F}_i[1 : (N - \lvert P_{t+1}\rvert)]$	izaberi prvih $(N - \lvert P_{t+1}\rvert)$ elemenata $\mathcal{F}_i$
$Q_{t+1} =$ make-new-pop(P_{t+1})	koristeći selekciju, križanje i mutaciju stvori novu populaciju Q_{t+1}
$t = t + 1$	povećaj brojač generacija

Slika 5.2. Pseudokod NSGA2 (preuzeto iz [5])



Računanje grupirajuće udaljenosti. Ispunjene točke predstavljaju rješenja iste fronte.

Slika 5.3. Računanje grupirajuće udaljenosti



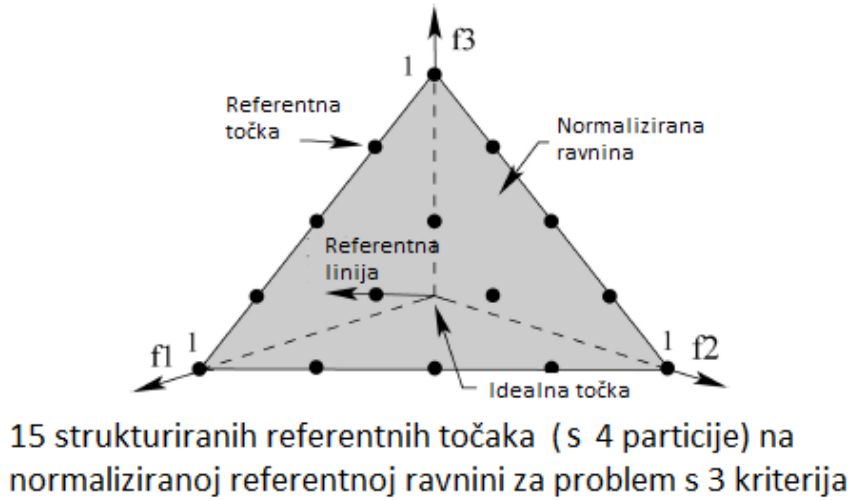
Slika 5.4. Selekcija kod NSGA2 (preuzeto iz [5])

5.3. NSGA3

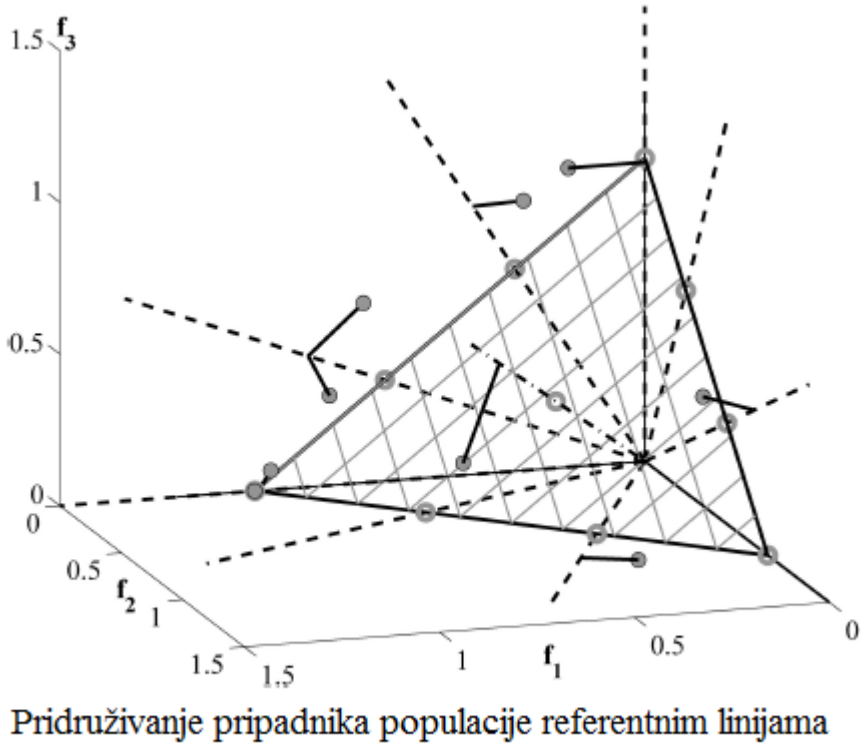
Algoritam NSGA-3 [3] je evolucijski algoritam za mnogokriterijsku optimizaciju. Rađen je po uzoru na NSGA-2, a to je i vidljivo stoga što je jedina razlika među njima način ostvarenja selekcije pomoću niše. NSGA-2 definira grupirajuću udaljenost te pušta operatoru selekcije da spontano stvori raznolikost među nedominiranim rješenjima. NSGA-3 stvara posebne referentne linije pomoću kojih koordinira selekciju. Moglo bi se reći da se NSGA-2 više oslanja na zakon velikih brojeva od NSGA-3, što zbog posebnosti mnogokriterijskih problema nije najefikasniji pristup. U nastavku će se analizirati selekcija pomoću niše koja se primjenjuje u algoritmu NSGA-3. Analizirat će se na problemu s tri kriterija jer je takav problem moguće prikazati na grafu, ali se analogno može primijeniti na problem s četiri i više kriterija. Prvo ćemo postaviti referentne linije za upravljanje selekcijom. Njih će definirati ishodište koordinatnog sustava i referentne točke koje ćemo zadati. Referentne točke se mogu zadati automatski *(Slika 5.5, Slika 5.7., Slika 5.8, Slika 5.9.)* tako da se postave određeni parametri i onda izvrši algoritam generiranja, ili ih može zadati korisnik. Automatsko zadavanje se najčešće koristi kad se želi pokriti cijela Pareto optimalna fronta, a korisničko zadavanje kad se želi fokusirati na pokrivanje samo određenog dijela Pareto optimalne fronte. U implementaciji će se koristiti automatsko zadavanje. Ono se obavlja tako da se

uniformno generiraju točke na trokutu kojeg definiraju točke na koordinatnim osima s jednom koordinatom jednakom jedan. Da bi se konačno generirale točke na tom trokutu potrebno je odrediti na koliki će se broj particija jedna stranica trokuta podijeliti. Taj broj i broj kriterija definiraju broj referentnih točaka tj. broj referentnih linija, koji se računa po formuli navedenoj u nastavku. Programski kod za određivanje referentnih točaka je dan u nastavku. Referentne linije su pravci koji prolaze kroz pojedine referentne točke i ishodište koordinatnog sustava. Sad kad imamo referentne linije može se započeti s iteracijama algoritma. Svi koraci su isti kao u NSGA-2 do trenutka kada dođe do potrebe određivanja koje će se jedinke iz zadnje uključene fronte prenijeti u iduću generaciju. Tada kreće sljedeći postupak. Vektori dobrote svih jedinki u zadnjoj uključenoj fronti se prvo normaliziraju tako da su im vrijednosti svih komponenti između 0 i 1. Zatim se za svaki vektor dobrote traži referentna linija koja mu je u prostoru najbliža i na nju se onda asocira *(Slika 5.6.).* Selekcija se provodi tako da se nasumično izabere neka referentna linija na koju je asociran barem jedan vektor dobrote tj. jedinka. Ta se jedinka zatim prenosi u novu generaciju. Ako je više jedinki asocirano na istu referentnu liniju onda se prenosi ona bliža liniji. Nakon prenošenja jedinke u novu generaciju ona se briše iz trenutne populacije. Postupak se ponavlja sve dok nije prenesen zadovoljavajuć broj jedinki, ali tako da referentna linija čija je asocirana jedinka prenesena u novu generaciju ima manji prioritet od one čija nije. Odabir druge jedinke za prenošenje neke referentne linije može se obavljati nasumično [3]. Treba istaknuti odsustvo potrebe za specificiranjem parametara algoritma NSGA-3 za rješavanje različitih problema. Naime, postoji pravilo da se broj referentnih linija izjednači (barem aproksimativno) s brojem jedinki u populaciji. Tako bi u konačnici trebali dobiti uniformno raspoređenu aproksimaciju Pareto optimalne fronte. Iz prethodnog se može uočiti da se broj particija ne treba posebno zadavati nego da je određen veličinom populacije i brojem kriterija. Usporedba algoritma NSGA-3 s različitim verzijama algoritma MOEA/D pomoću IGD mjere prikazana je u radu Deb, 2014 [3]. Rezultati pokazuju da se NSGA-3 dobro snalazi na većini problema dok različite inačice MOEA/D ne pokrivaju toliko dobro sve vrste problema. U nastavku je prikazana usporedna analiza algoritama NSGA-3 i NSGA-2 pomoću hipervolumen indikatora koja bi trebala pokazati mnogokriterijsko svojstvo NSGA-3 i višekriterijsko svojstvo NSGA-2. Usporedna analiza i optimizacijski algoritmi su

implementirani u sklopu ovog rada. Detaljniji opis algoritma se može pronaći u Deb, 2014 [3].



Slika 5.5. Prikaz automatski generiranih referentnih točaka i linija (preuzeto iz [3])



Slika 5.6. Pridruživanje jedinki referentnim linijama (preuzeto iz [3])

```
private void createReferenceSet(List<IntegerWrapper> sticks) {
    Collections.reverse(sticks);
    IntegerWrapper last = sticks.get(0);
    IntegerWrapper copy = null;
    Boolean firstInIter = true;
    Boolean outOfBounds = false;
    Boolean afterOutOfBounds = false;
    for (IntegerWrapper stick : sticks) {
        if (stick.value.equals(partitions.value + 1)) {
            outOfBounds = true;
        }
        if (firstInIter && outOfBounds)
            return;
        if (outOfBounds && !afterOutOfBounds) {
            last.value++;
            copy = last;
            stick.value = copy.value;
            afterOutOfBounds = true;
        }
        if (afterOutOfBounds) {
            stick.value = copy.value;
        }
        if (firstInIter)
            firstInIter = false;
        last = stick;
    }
    Collections.reverse(sticks);
    if (outOfBounds) {
        createReferenceSet(sticks);
        return;
    }
    List<Double> refPoint = new ArrayList<Double>();
    last = partitions;
    for (IntegerWrapper stick : sticks) {
        refPoint.add(((double) (last.value - stick.value))/ ((double) partitions.value));
        last = stick;
    }
    refPoint.add(((double) (last.value)) / ((double) partitions.value));

    refPoints.put(refPointId, refPoint);
    refPointId++;
    sticks.get(0).value++;
    createReferenceSet(sticks);
}
```

Slika 5.7. Java kod - Računanje refenentnih točaka

```
List<IntegerWrapper> sticks = new ArrayList<IntegerWrapper>();
for (int i = 0; i < dimensionOfResult - 1; i++) {
    sticks.add(new IntegerWrapper(0));
}
createReferenceSet(sticks);
```

Slika 5.8. Java kod – Pozivanje metode za računanje referentnih točaka

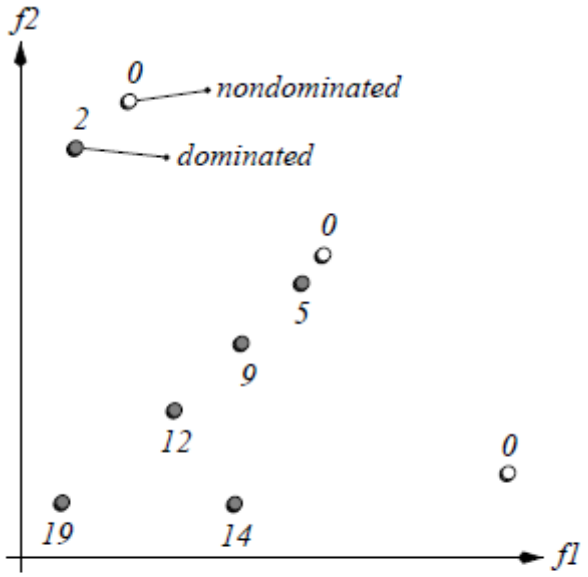
$$H = \binom{M + p - 1}{p}$$

Slika 5.9. Formula za računanje broja referentnih točaka (M – broj kriterija, p – broj particija) (preuzeto iz [3])

5.4. SPEA2

Algoritam SPEA-2 [6] je evolucijski algoritam za višekriterijsku generaciju druge generacije. To je drugi algoritam druge generacije MOEA koji ćemo analizirati. Njemu je specifično da sadrži elitističku selekciju, no on je obavlja na drugačiji način od NSGA-2. Elitistička selekcija se kod SPEA-2 obavlja pomoću vanjske arhive jedinki. Iz naziva se može uočiti da je SPEA-2 zapravo nadogradnja algoritma SPEA, no razlika ta dva ovdje neće biti analizirana, ali se može potražiti u Zitzler 2001. [6]. Selekciju pomoću niše SPEA-2 ostvaruje dodjeljivanjem posebnog oblika dobrote u koju su ukomponirane informacije o dominiranosti i o okruženju jedinke. Dobrota se dobiva zbrajanjem sirove (engl. *raw*) dobrote i gustoće jedninke. Sirova dobrota se definira kao ukupan broj jedinki (jedinke se mogu ponavljati) koje su dominirane od svake pojedine jedinke koja dominira tu jedinku čiju sirovu dobrotu tražimo *(Slika 5. 10.).* Matematički zapis prethodnog računa je priložen u nastavku. Gustoća neke jedinke se definira kao recipročna udaljenost te jedinke do svojeg k-tog susjeda tj. k-te najbliže jedinke, s malom izmjenom radi graničnih slučajeva. Broj k se određuje kao korijen zbroja veličine populacije i veličine arhive. Točna formula je dana u nastavku *(Slika 5.11.).* Skup unutar kojeg se računaju oba dijela dobrote je unija trenutne populacije i trenutne arhive. Svi izračuni se obavljaju u prostoru vektora dobrote. Može se primijetiti da je jedinka s manjim iznosom ovako definirane dobrote zapravo bolja jedinka u terminima dominiranosti i raspršenja. Postupak stvaranja arhive iduće generacije ide ovako. Prvo se u novu arhivu prenose sva nedominirana rješenja iz unije trenutne arhive i trenutne populacije. Nakon ovog dijela postupka u arhivi može biti ili previše ili premalo jedinki. Ako ih je premalo, u arhivu se dodaju dominirane jedinke, ali one s najmanjim iznosima dobrote (to su zapravo najbolje jedinke). Ako ih je previše, iz arhive se izbacuju nedominirane jedinke po sljedećem pravilu. Jedinka s najbližim prvim susjedom bi se trebala prva izbaciti. Naravno, uvijek postoje barem dvije takve jedinke te se stoga kao idući kriterij po prioritetu gleda

drugi najbliži susjed. Ako imaju jednako udaljenog i drugog susjeda onda se gleda treći itd. Ovaj postupak se također može svrstati u selekciju pomoću niše, a u algoritmu se naziva selekcija pomoću okruženja (engl. *environmental selection*) *(Slika 5.12.).* Nakon što arhiva ima zadovoljavajuć broj jedinki mehanizmom binarne turnirske selekcije stvara se bazen križanja (engl. *mating pool*). Iz bazena križanja se proizvoljnim mehanizmima rekombinacije i mutacije stvara populacija nove generacije. Treba uočiti da će u prvoj iteraciji algoritma arhiva biti prazna, ali to ne ometa izvođenje algoritma. Algoritam se, kao i svi dosad opisani MOEA-i, zaustavlja nakon željenog broja generacija, a konačno rješenje predstavljaju nedominirane jedinke unutar trenutne arhive. Na testnim problemima SPEA-2 je u rangu s NSGA-2 gledajući iznose određenih MOEA indikatora [6]. Detaljniji opis algoritma se može pronaći u Zitzler, 2001 [6].



Računanje sirove dobrote (SPEA2)

Slika 5.10. Prikaz računanja sirove dobrote (preuzeto iz [6])

$$S(i) = |\{j \mid j \in P_t + \overline{P}_t \wedge i \succ j\}|$$

$$R(i) = \sum_{j \in P_t + \overline{P}_t, j \succ i} S(j)$$

$$D(i) = \frac{1}{\sigma_i^k + 2}$$

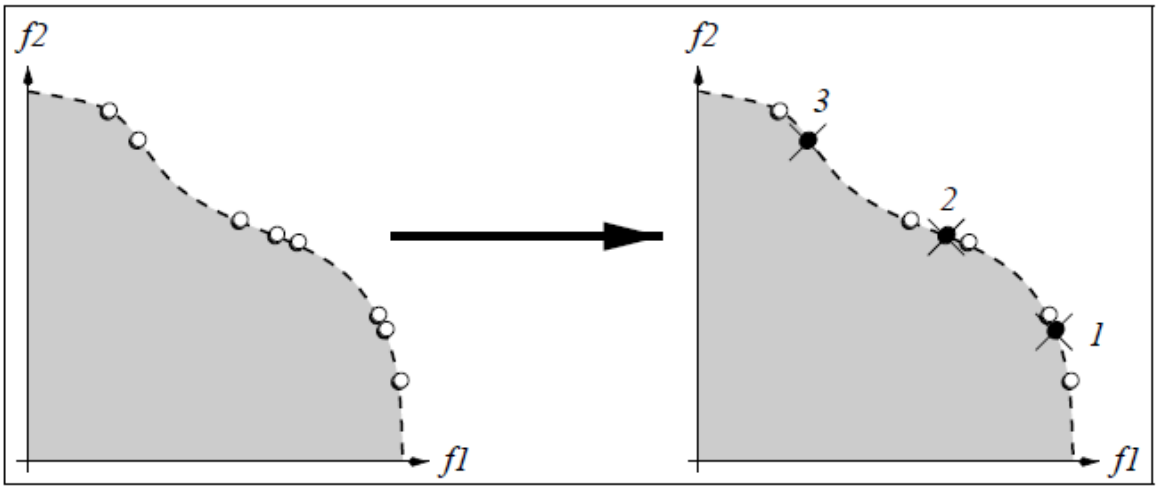
$$F(i) = R(i) + D(i)$$

$P_t + \overline{P}_t$
unija populacije i arhive

σ_i^k
udaljenost do k-tog susjeda

$\succ$
relacija dominacije

Slika 5.11. Računanje dobrote - SPEA2 (preuzeto iz [6])

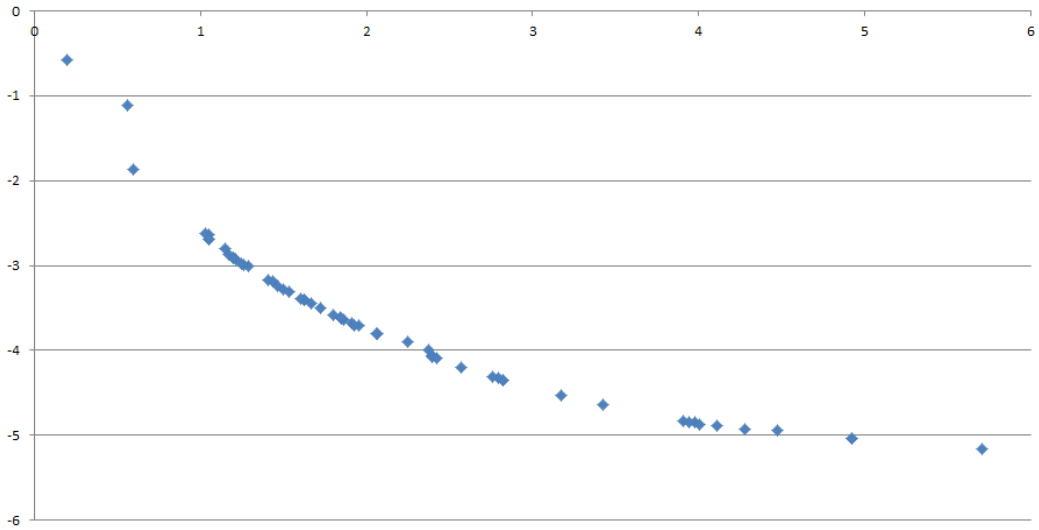


Provedba selekcije pomoću okruženja (SPEA2)

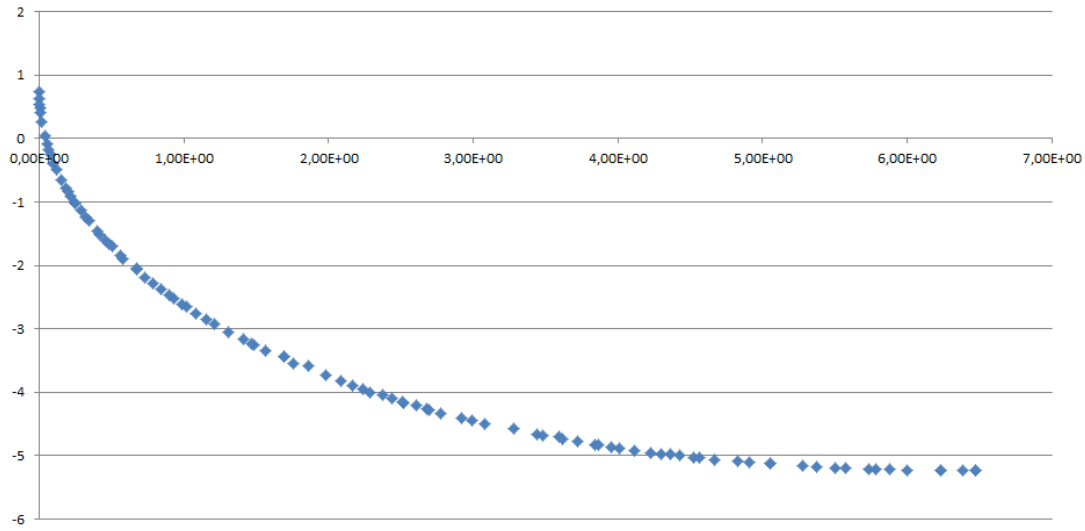
Slika 5.12. Prikaz selekcije pomoću okruženja (preuzeto iz [6])

6. Rezultati

U nastavku će biti prikazani rezultati prethodno opisanih algoritama na nekim ispitnim problemima. Ispitni problemi će varirati od onih jednostavnijih do onih složenijih. Prvo će se pokazati izgled prve fronte na jednom relativno jednostavnom problemu i usporediti za različite algoritme. Zatim će se prikazati iznos metrike hipervolumena u ovisnosti o broju generacije kako bi se uočilo postupno napredovanje algoritma tj. kvalitete rješenja. Na kraju će biti prikazane stvarne performanse algoritama na pojedinim problemima pomoću metrike hipervolumen. Za to će nam pomoći *boxplot* prikaz koji sadrži statističku analizu rezultata.



Slika 5.13. Prva fronta algoritma NSGA za JP



Slika 5.14. Prva fronta algoritma NSGA2 za JP

Iz prethodne dvije slike *(Slika 5.13., Slika 5.14.)* može se uočiti da algoritam NSGA2 puno bolje rješava problem JP *(Slika 5.15.)* s obzirom na željena svojstva rješenja višekriterijskog problema. Izračunavanjem metrike udaljenosti i metrike raznolikosti mogao bi se dobiti i brojčani rezultat. Algoritmi su pokrenuti s toliko sličnim parametrima koliko je to moguće kako bi usporedba rješenja bila relevantna. Oba algoritma su pisana u Javi.

Problem JP:

- $f_1(x_1, x_2) = (x_1 + 3)^2 + (x_2 - 1)^2$
- $f_2(x_1, x_2) = 3\sin(x_1) - \cos(x_2) + 2\sin(x_2)$

Oba kriterija se minimiziraju.

Grafovi u nastavku će prikazivati rezultate rješavanja modificiranih DTLZ problema [1] *(Slika 5.16.)* pomoću opisanih MOEA (bez SPEA2). Rezultati će biti grupirani u skupine s obzirom vrste operatora mutacije i križanja koje će algoritmi koristiti, no to se odnosi samo na algoritme implementirane u sklopu ovog rada. Na grafovima će za usporedbu biti dodana implementacija algoritma NSGA2 iz radnog okvira jMetal. Grupiranje je učinjeno kako bi rezultati bili što relevantniji. Problemi DTLZ

su modificirani kako bi metrika hipervolumena s referentnom točkom u ishodištu mogla biti relevantna za usporedbu algoritama. Izvorni DTLZ problemi su problemi minimizacije, a kod takvih problema ovaj oblik metrike gubi neka poželjna svojstva. U nastavku je objašnjeno zašto je to tako. Isprva se čini da će kod minimizacije manji iznos takvog hipervolumena značiti bolje rješenje, no povećavanje prve fronte za jednu jedinku poboljšava rješenje, a povećava iznos metrike. Također, razlike u predznaku rješenja nisu poželjne jer bi se dodavanjem novih jedinki u prvu frontu iznos metrike mogao i povećavati i smanjivati. Kako bi se zaobišla prethodna dva problema DTLZ problemi su modificirani na sljedeći način: f_{novi} (x) = - f(x) + pozitivna_konstanta, te se umjesto minimizacije provodi maksimizacija. Konkretno, kod DTLZ1 problema pozitivna konstanta je 500, a kod DTLZ2 problema ona je 10.

Name	Problem			Parameter Domains
DTLZ1	f_1	$=$	$(1+g)0.5\prod_{i=1}^{M-1} y_i$	$[0,1]$
	$f_{m=2:M-1}$	$=$	$(1+g)0.5\left(\prod_{i=1}^{M-m} y_i\right)(1-y_{M-m+1})$	
	f_M	$=$	$(1+g)0.5(1-y_1)$	
	g	$=$	$100\left[k+\sum_{i=1}^{k}\left((z_i-0.5)^2-\cos(20\pi(z_i-0.5))\right)\right]$	
DTLZ2	f_1	$=$	$(1+g)\prod_{i=1}^{M-1}\cos(y_i\pi/2)$	$[0,1]$
	$f_{m=2:M-1}$	$=$	$(1+g)\left(\prod_{i=1}^{M-m}\cos(y_i\pi/2)\right)\sin(y_{M-m+1}\pi/2)$	
	f_M	$=$	$(1+g)\sin(y_1\pi/2)$	
	g	$=$	$\sum_{i=1}^{k}(z_i-0.5)^2$	

Slika 5.16. Izvorni DTLZ problemi na kojima će algoritmi biti ispitivani (M – broj kriterija, y –parametri pozicije, z – parametri udaljenosti, k – broj parametara udaljenosti, f_i - kriterij) (preuzeto iz [1])

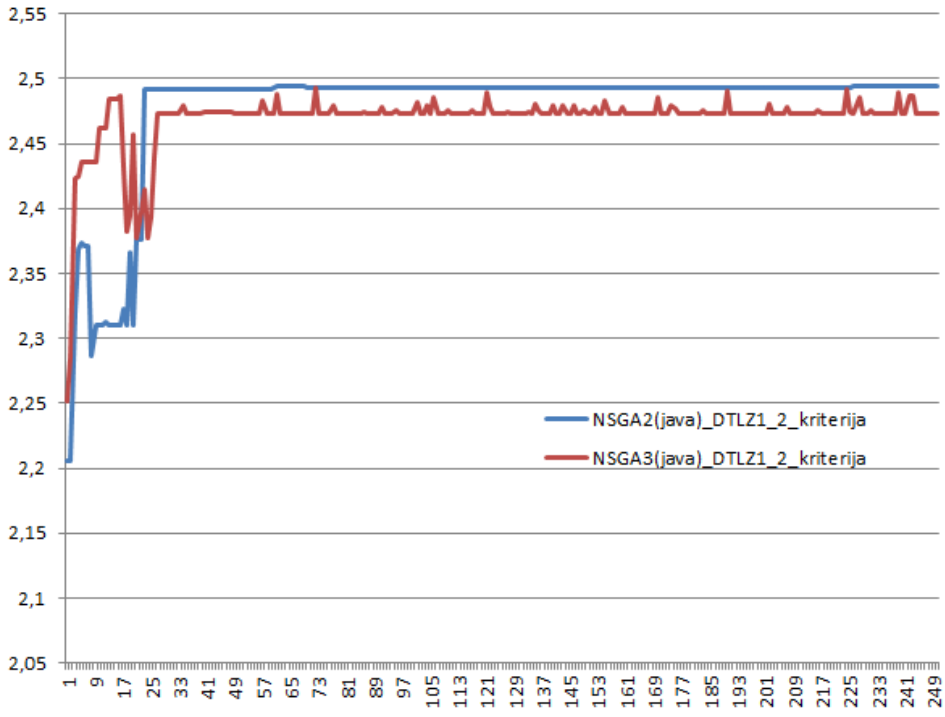
Svojstvo DTLZ problema je da imaju varijabilan broj parametara udaljenosti, a fiksan broj parametara pozicije (broj_kriterija - 1). Dodatna svojstva DTLZ1 problema su: multimodalnost, linearan oblik fronte i preslikavanje više-na-jedan. Dodatna svojstva DTLZ2 problema su: multimodalnost, konkavan oblik fronte i preslikavanje više-na-jedan. Odsustvo nekog svojstva nije navedeno. U nastavku će biti prikazan graf ovisnosti iznosa metrike hipervolumen i broja generacije kod nekih algoritama. Svi problemi će imati 3 parametra udaljenosti, a broj kriterija će varirati. Iznos hipervolumena je skaliran na interval [0,10]. Takvom modifikacijom nisu izgubljene bitne informacije o razlici iznosa između različitih algoritama i

oscilaciji iznosa. Svi problemi su rješavani s jednakom veličinom populacije i brojem generacija.

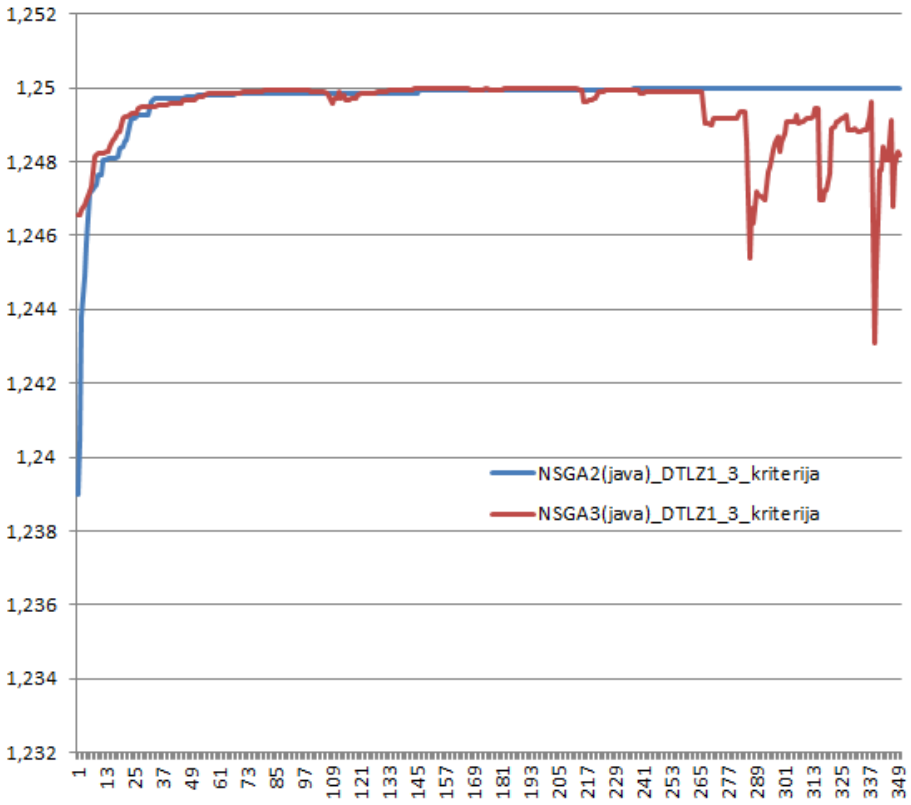
Konkretno, postavke algoritama su sljedeće:

- 2 kriterija – 100 jedinki / 4 jedinke, 250 generacija
- 3 kriterija – 200 jedinki / 68 jedinki, 350 generacija
- 5 kriterija – 332 jedinke, 220 generacija
- 7 kriterija – 212 jedinke, 150 generacija

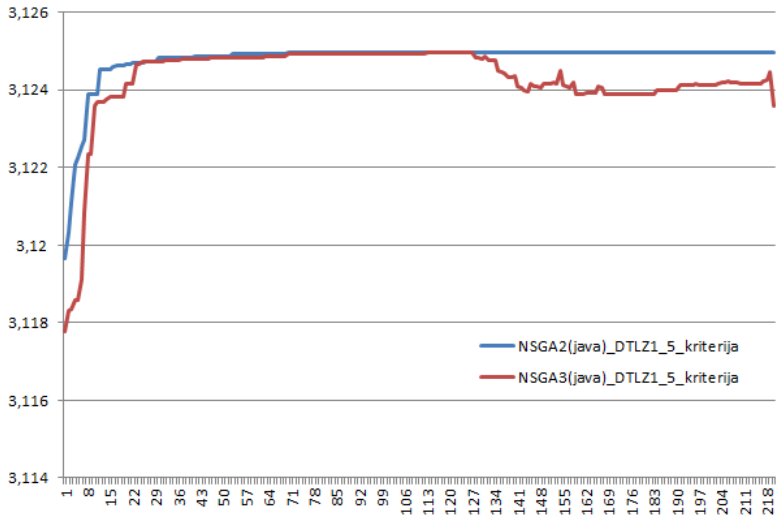
U problemima s 2 i 3 kriterija manji broj jedinki se koristi kada se uspoređuje algoritam NSGA3 zbroj automatskog generiranja referentnih točaka i predložene ovisnosti broja referentnih točaka i veličine populacije [5]. Grafovi u nastavku će prikazivati ovisnost iznosa metrike hipervolume (modificirane za potrebe prikaza) o trenutnom broju generacije. Načinjeni su od jednog pokretanja algoritama kako bi se stekao dojam o njihovom ponašanju na pojedinom problemu. Algoritmi čije će se kretanje hipervolumena mjeriti su NSGA2 i NSGA3 implementirani u sklopu ovog rada u Javi.



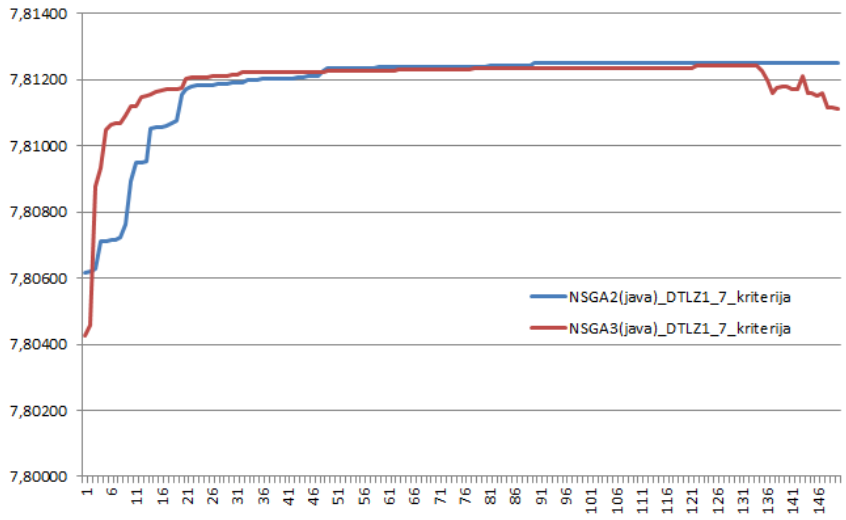
Slika 5.17. Konvergencija algoritama - DTLZ1, 2 kriterija (NSGA2(java), NSGA3(java))



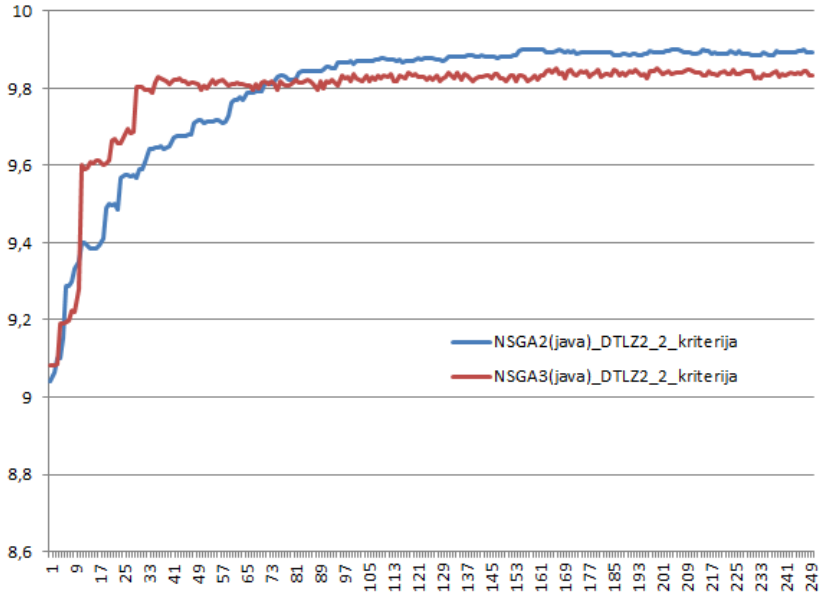
Slika 5.18. Konvergencija algoritama - DTLZ1, 3 kriterija (NSGA2(java), NSGA3(java))



Slika 5.19. Konvergencija algoritama - DTLZ1, 5 kriterija (NSGA2(java), NSGA3(java))

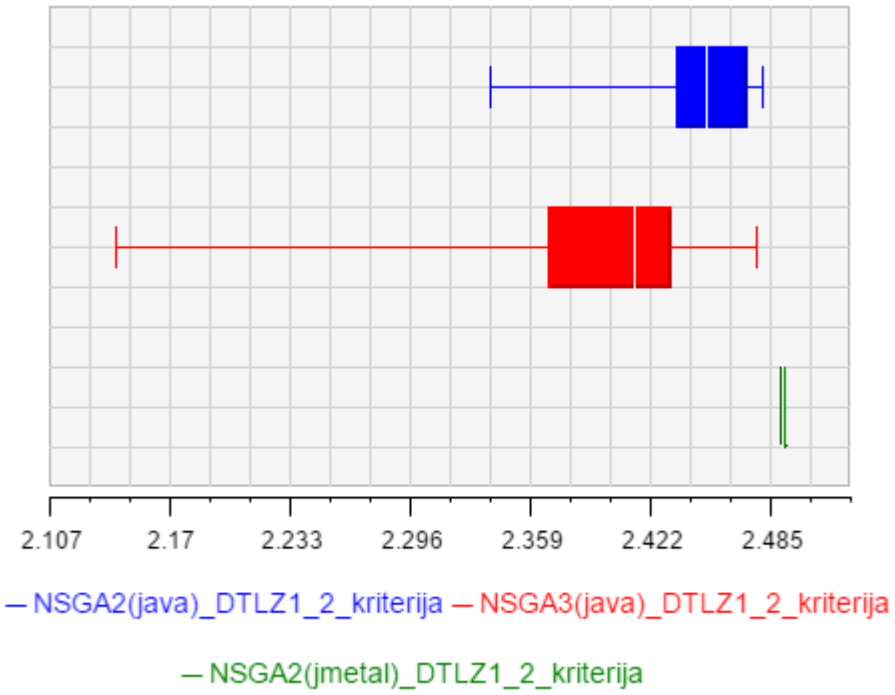


Slika 5.20. Konvergencija algoritama - DTLZ1, 7 kriterija (NSGA2(java), NSGA3(java))

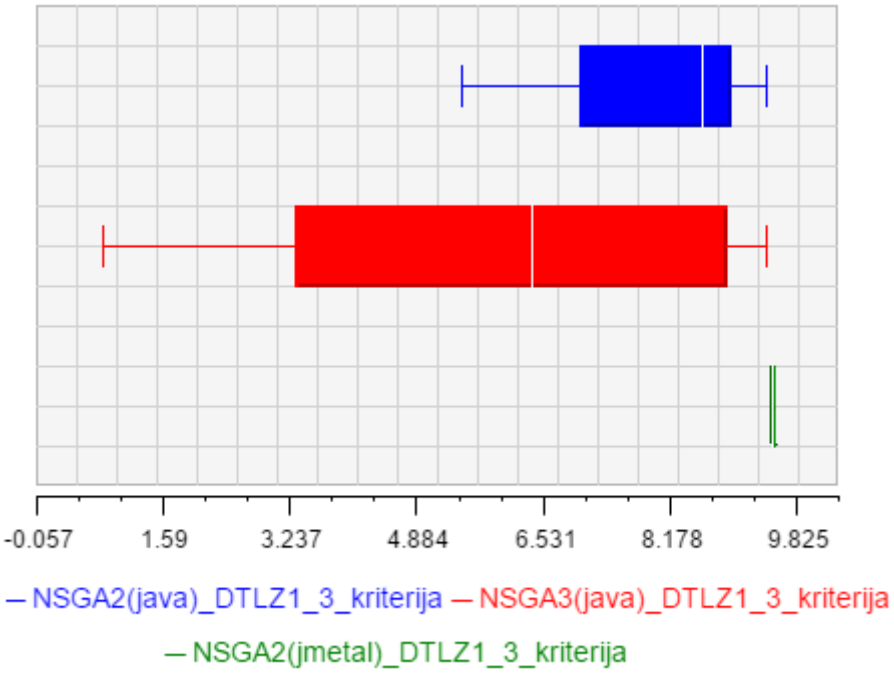


Slika 5.21. Konvergencija algoritama – DTLZ2, 2 kriterija (NSGA2(java), NSGA3(java))

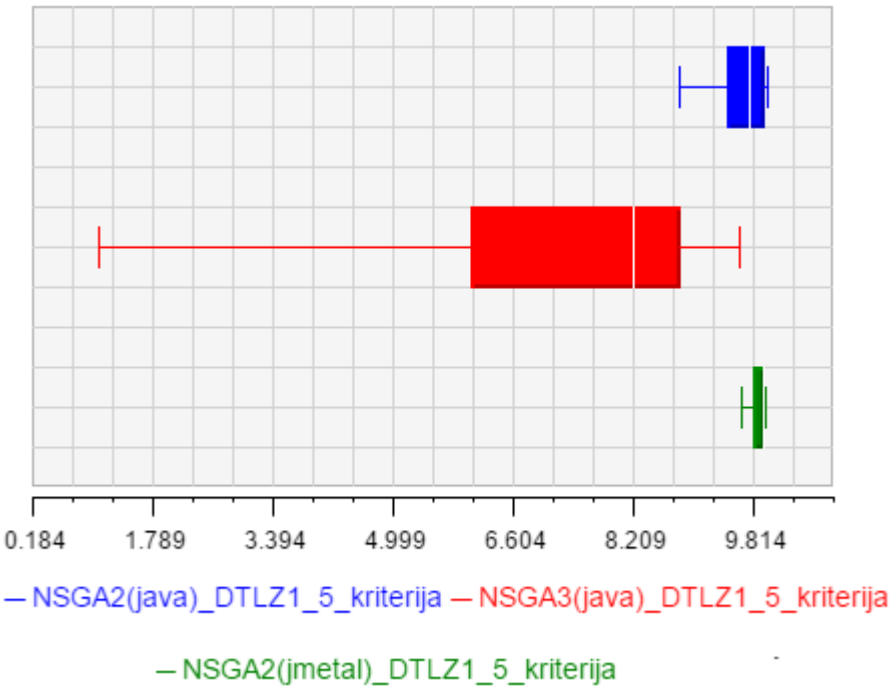
Može se uočiti da NSGA2 prednjači ispred NSGA3 u rješavanju prethodnih problema (Slika 5.17. – 5.21.) iako je dio problema mnogokriterijski, no takvi su rezultati karakteristični za problem DTLZ1 [5]. Također se može uočiti da NSGA2 uvijek konvergira. U nastavku su prikazani boxplot grafovi. Za svaki graf pojedini algoritam je pokrenut 15 puta i uzeta je konačna vrijednost metrike hipervolumen uz određene modifikacije kako bi usporedba bila lakša. Prvu grupu algoritama koji se zajedno tako ispituju čine NSGA2(Java), NSGA3(Java) i NSGA2(jMetal).



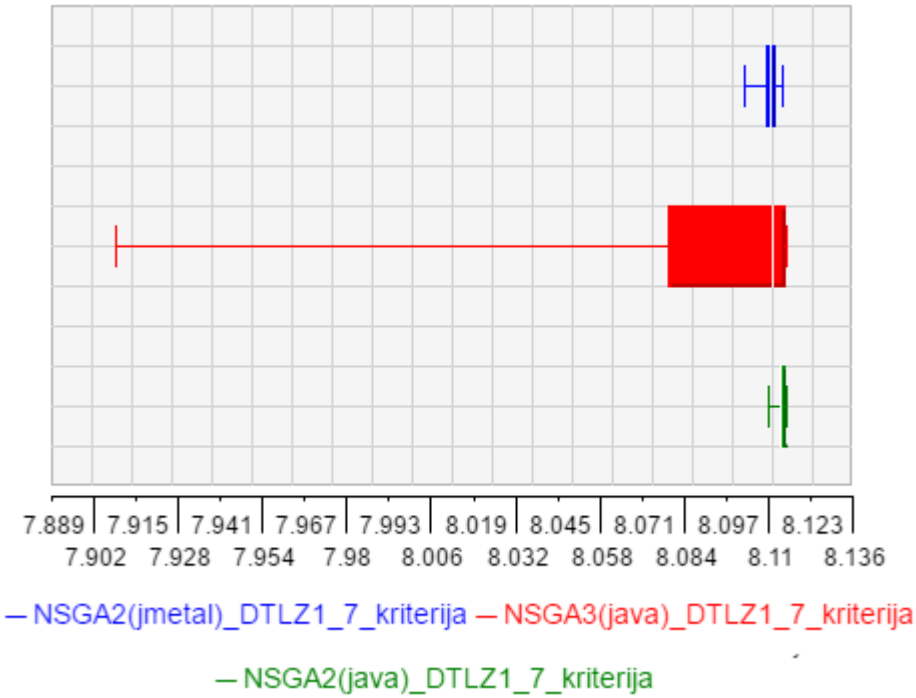
Slika 5.22. Performanse algoritama – DTLZ1, 2 kriterija (NSGA2(java), NSGA3(java), NSGA2(jMetal))



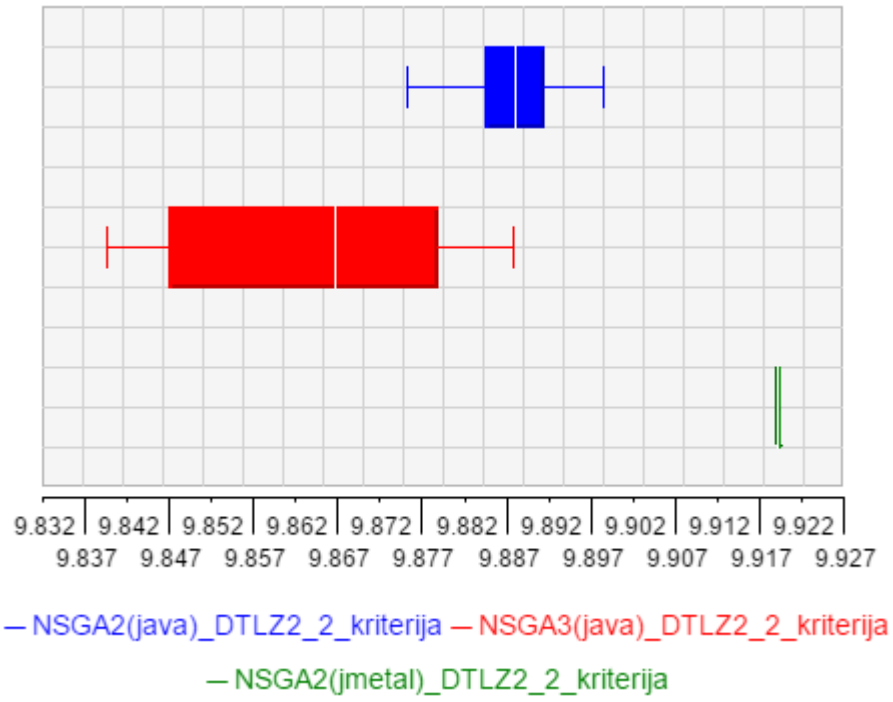
Slika 5.23. Performanse algoritama – DTLZ1, 3 kriterija (NSGA2(java), NSGA3(java), NSGA2(jMetal)



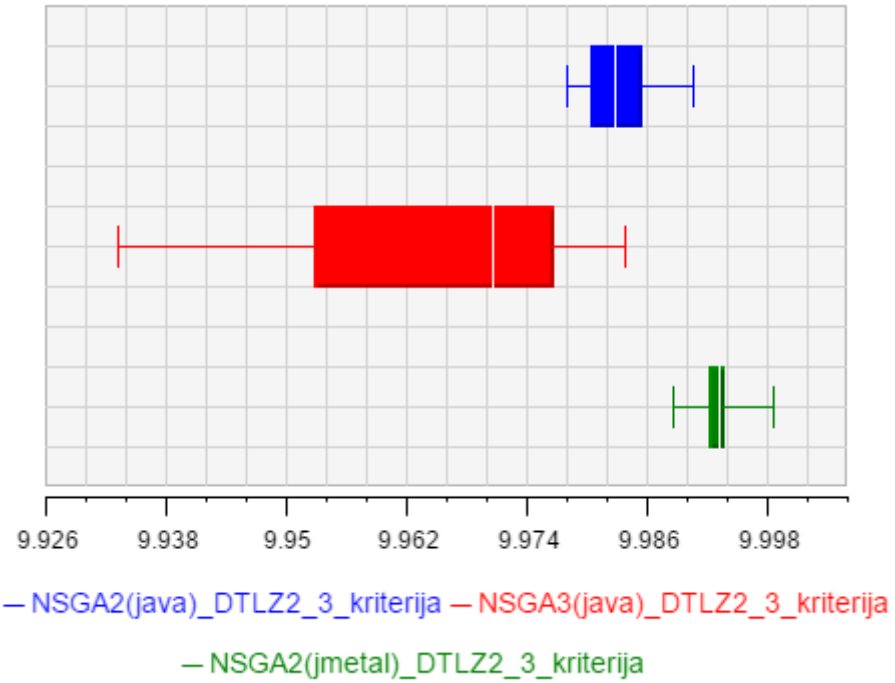
Slika 5.24. Performanse algoritama – DTLZ1, 5 kriterija (NSGA2(java), NSGA3(java), NSGA2(jMetal))



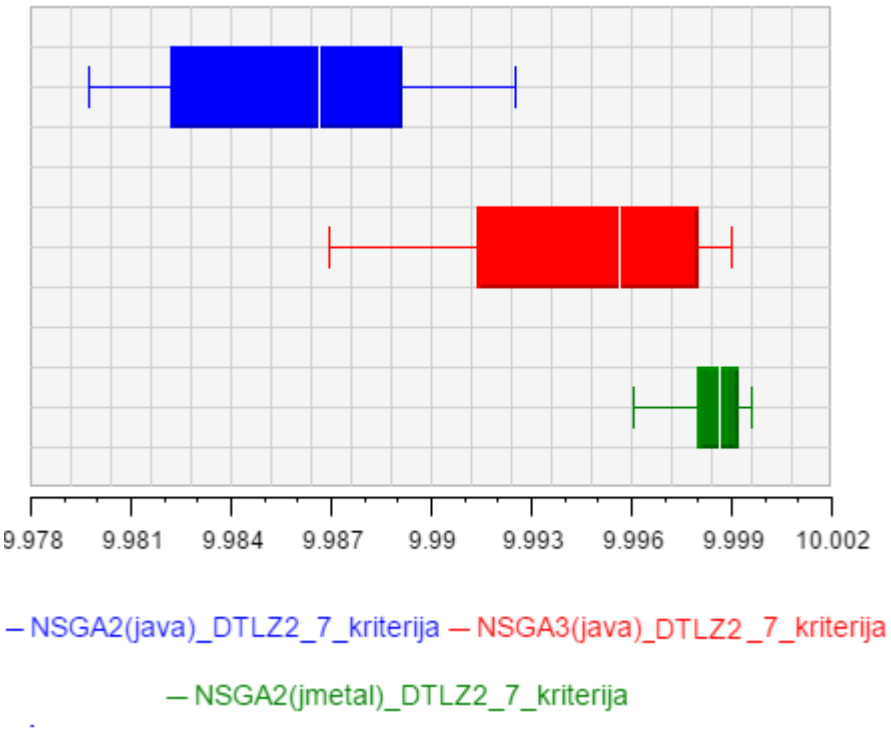
Slika 5.25. Performanse algoritama– DTLZ1, 7 kriterija (NSGA2(java), NSGA3(java), NSGA2(jMetal))



Slika 5.26. Performanse algoritama – DTLZ2, 2 kriterija (NSGA2(java), NSGA3(java), NSGA2(jMetal))

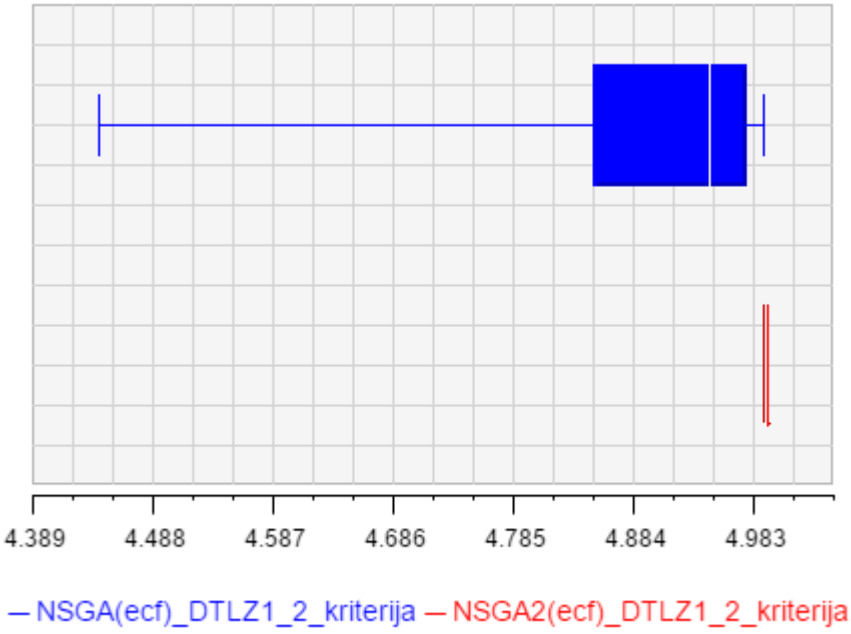


Slika 5.27. Performanse algoritama – DTLZ2, 3 kriterija (NSGA2(java), NSGA3(java), NSGA2(jMetal))

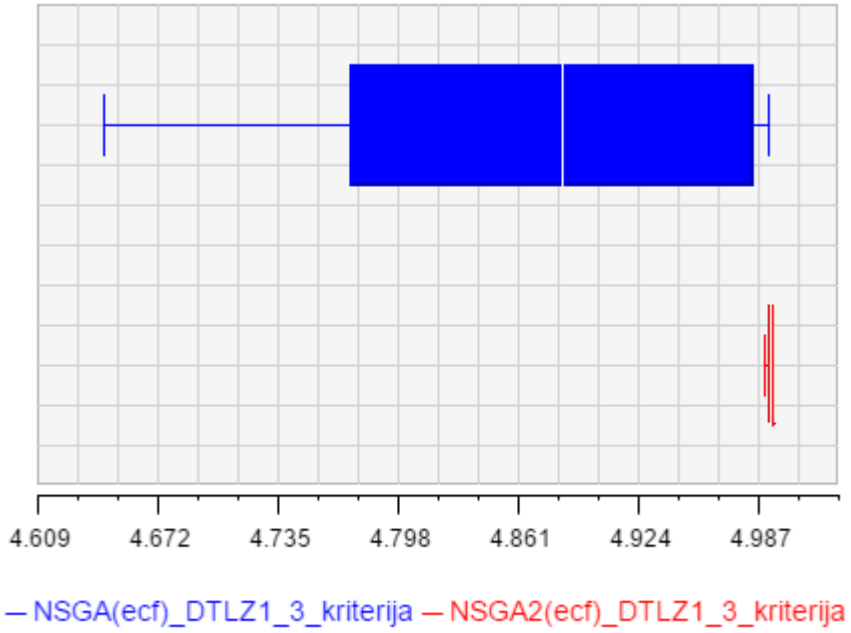


Slika 5.28. Performanse algoritama – DTLZ2, 7 kriterija (NSGA2(java), NSGA3(java), NSGA2(jMetal))

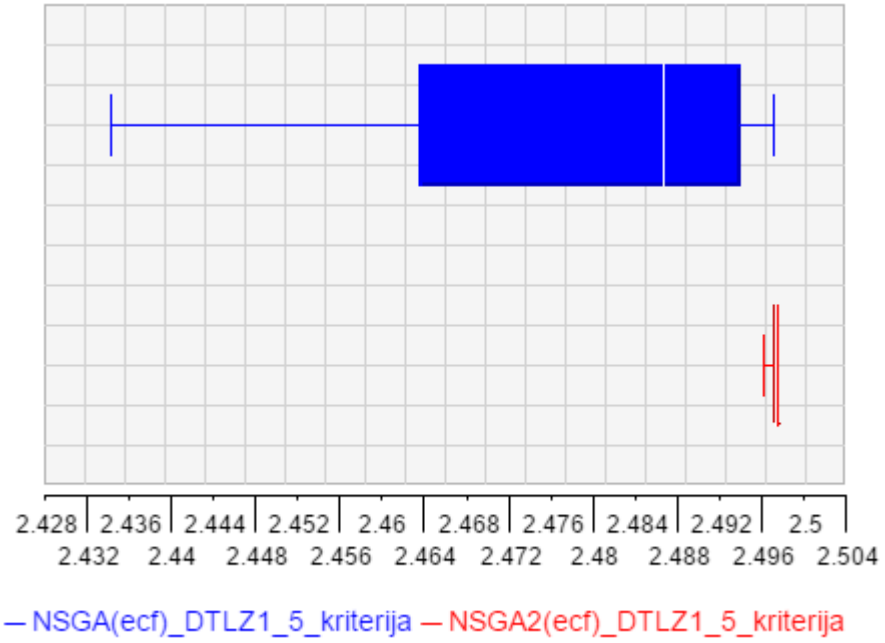
Može se uočiti da implementacija NSGA2 iz radnog okvira jMetal na svakom problemu pokazuje najbolje rezultate *(Slika 5.22.-5.28.).* No, bitno je uočiti da na DTLZ2 problemu sa 7 kriterija implementacija iz ovog rada NSGA3 postiže bolje rezultate od NSGA2 što ide u korist mnogokriterijskoj strategiji korištenoj u NSGA3. NSGA3 ima najveću disperziju među algoritmima što nije dobro svojstvo. U nastavku je prikazana usporedba algoritama NSGA i NSGA2 implementiranih u radnom okviru ECF u programskom jeziku C++. Od ta dva algoritma NSGA2 je implementiran u sklopu ovog rada, no kako su oba u radnom okviru ECF, koriste iste operatore mutacije i križanja pa se mogu svrstati u istu grupu za ispitivanje.



Slika 5.29. Performanse algoritama – DTLZ1, 2 kriterija (NSGA(ecf), NSGA2(ecf))

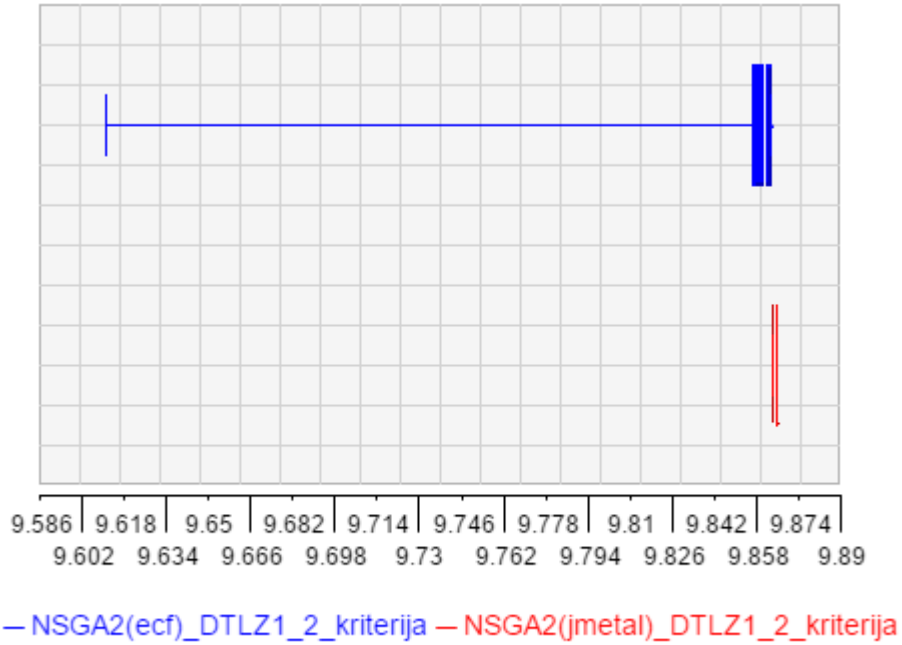


Slika 5.30. Performanse algoritama – DTLZ1, 3 kriterija (NSGA(ecf), NSGA2(ecf))

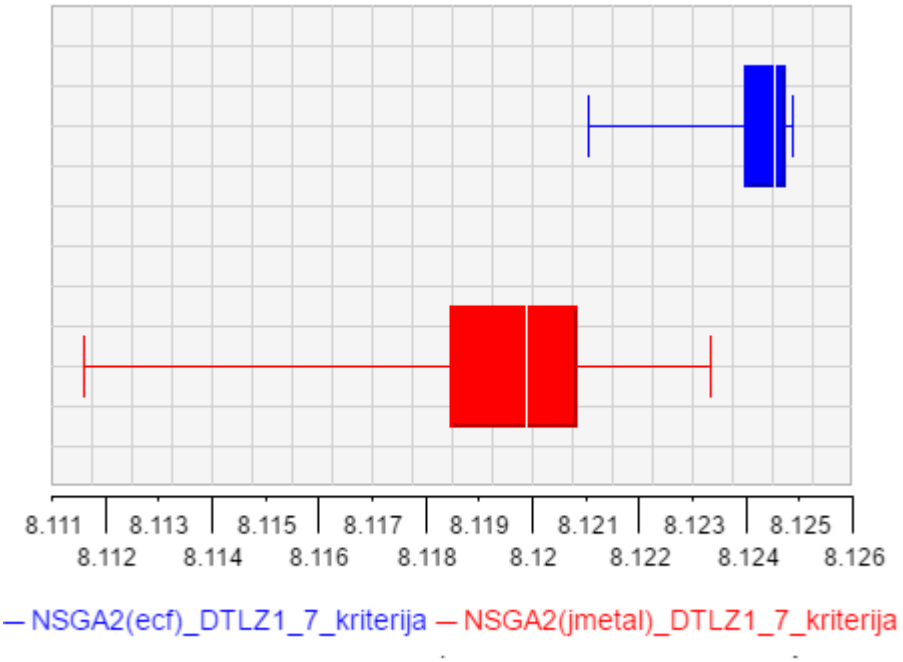


Slika 5.31. Performanse algoritama – DTLZ1, 5 kriterija (NSGA(ecf), NSGA2(ecf))

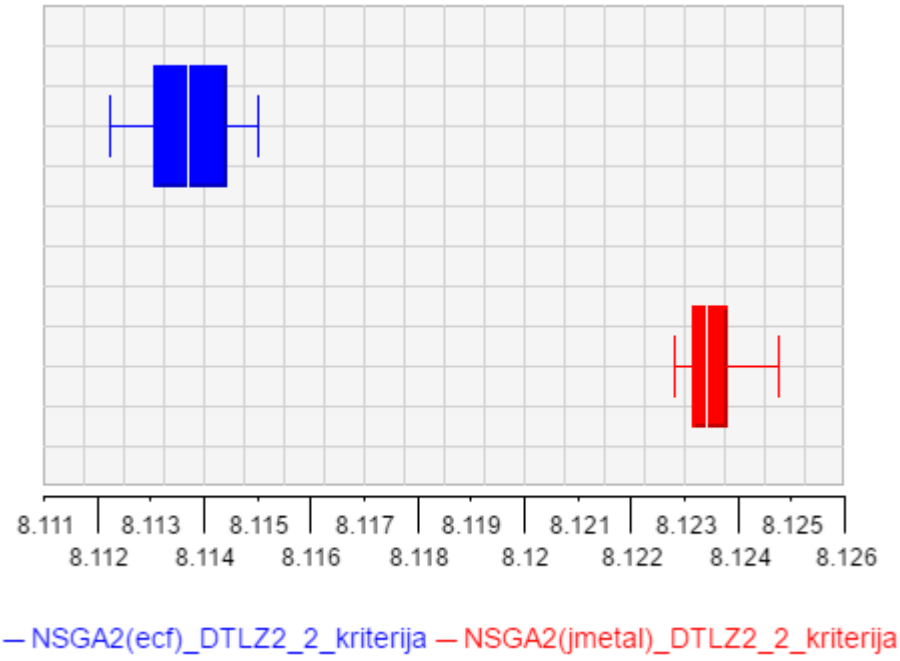
Može se uočiti da je algoritam NSGA2 znatno bolje rješava DTLZ1 probleme od NSGA *(Slika 5.29.-5.31.).* Iako postižu vrlo slične maksimume, disperzija NSGA je znatno veća što NSGA2 čini pouzdanijim. Zadnju grupu algoritama koji se zajedno ispituju čine NSGA2(ecf) i NSGA2(jMetal).



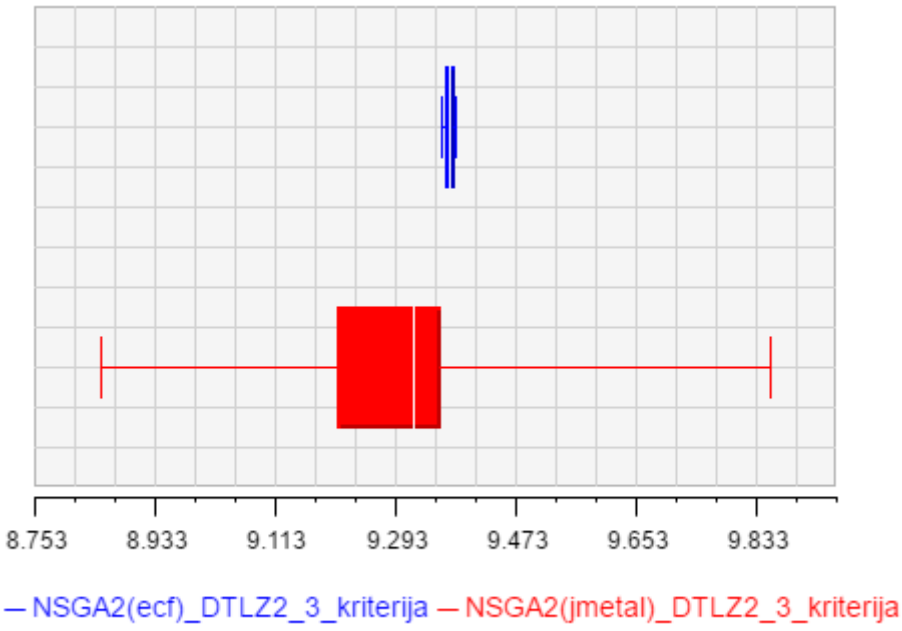
Slika 5.32. Performanse algoritama – DTLZ1, 2 kriterija (NSGA2(ecf), NSGA2(jMetal))



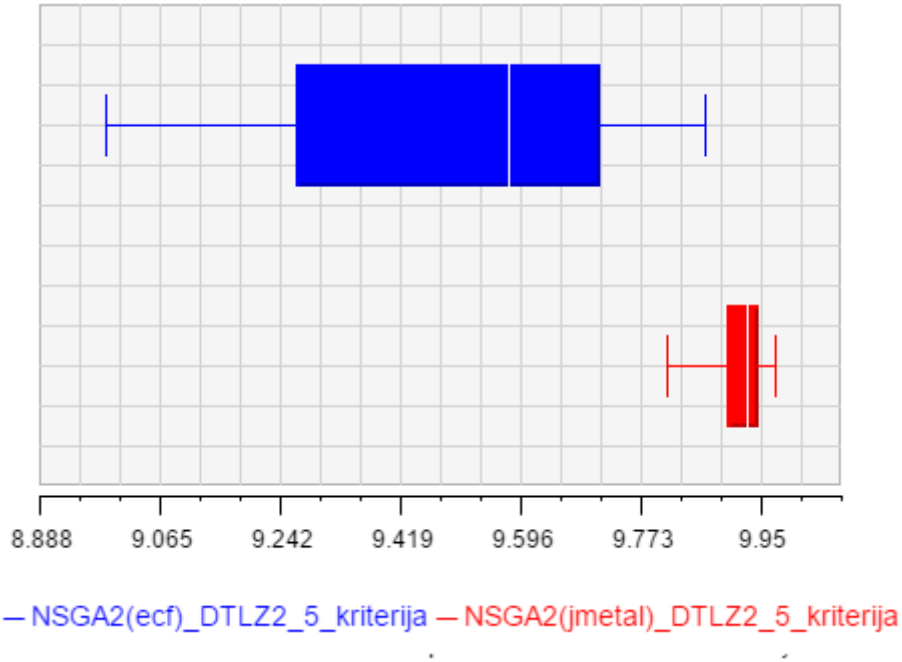
Slika 5.33. Performanse algoritama – DTLZ1, 7 kriterija (NSGA2(ecf), NSGA2(jMetal))



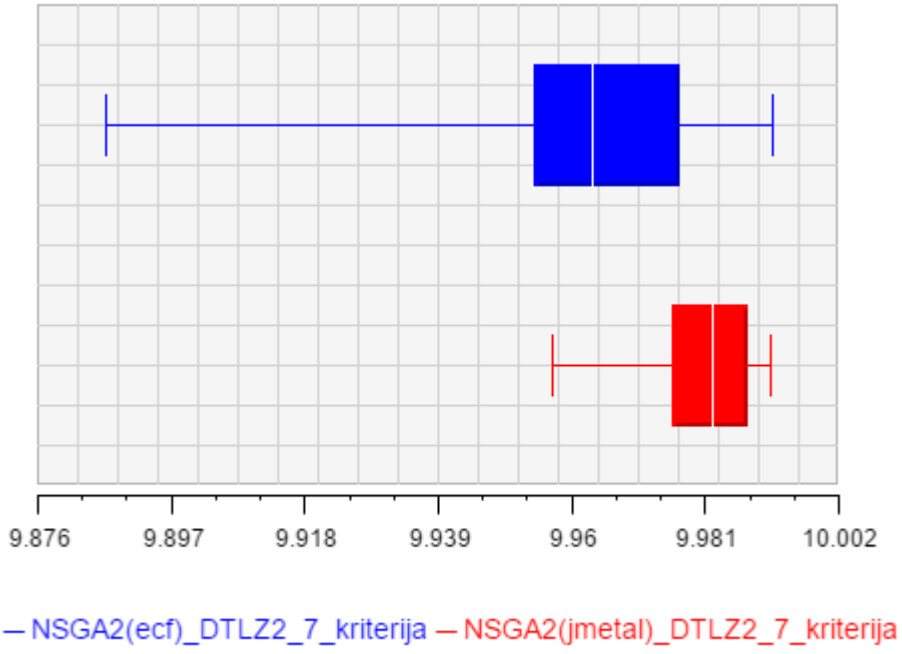
Slika 5.34. Performanse algoritama – DTLZ2, 2 kriterija (NSGA2(ecf), NSGA2(jMetal))



Slika 5.35. Performanse algoritama – DTLZ2, 3 kriterija (NSGA2(ecf), NSGA2(jMetal))



Slika 5.36. Performanse algoritama – DTLZ2, 5 kriterija (NSGA2(ecf), NSGA2(jMetal))



Slika 5.37. Performanse algoritama – DTLZ2, 7 kriterija (NSGA2(ecf), NSGA2(jMetal))

Načelno, implementacija NSGA2 iz radnog okvira jMetal postiže bolje rezultate od implementacije NSGA2 iz radnog okvira ECF. No, u jednom od šest problema NSGA2 iz ECF-a pokazuje bolje rezultate od NSGA2 iz jMetal-a. Često je kod

NSGA2 iz ECF-a problem u disperziji. Treba uočiti da je povećano mjerilo tj. da je iznos hipervolumena dodatno skaliran radi boljeg pregleda. Iako je lošiji, NSGA2 iz radnog okvira ECF postiže bolje rezultate od NSGA2 implementiranog u Javi ako se gleda NSGA2 iz jMetal-a kao referentni algoritam.

7. Zaključak

Nakon analize evolucijskih algoritama za višekriterijsku i mnogokriterijsku optimizaciju (MOEA) uočeno je da niti jedan algoritam implementiran u sklopu ovog rada nije uspio pokazati ukupno bolje rezultate od algoritma NSGA2 implementiranog u radnom okviru jMetal. No, algoritam NSGA2 implementiran u radnom okviru ECF uspio je pokazati donekle slične rezultate, pa čak i bolje na jednom problemu. Također, uspješno je prikazano ukupno prednjačenje algoritma NSGA2 naprema algoritmu NSGA te bolje performanse algoritma NSGA3 od algoritma NSGA2 na jednom mnogokriterijskom problemu. Upotreba metrike hipervolumena pokazala se adekvatnom za uspoređivanje prethodno navedenih MOEA. Algoritmi su uspješno razloženi na gradivne blokove koje čine nedominirano sortiranje, selekcija pomoću niše te rekombinacija i mutacija.

Literatura

[1] Huband, S., Hingston, P. F., Barone, L., While, L. „A Review of Multi-objective Test Problems and a Scalable Test Problem Toolkit“ IEEE Transactions on Evolutionary Computation, 10(5), 2006., 477-506

[2] Čupić, M. „Prirodom inspirirani optimizacijski algoritmi. Metaheuristike.“, FER, 2013.

[3] Deb, K., Jain, H., „An evolutionary many-objective optimization algorithm using reference-point based non-dominated sorting approach, Part I: Solving problems with box constraints,” IEEE Transactions on Evolutionary Computation, vol. 18, no. 4, 2014., pp. 577–601

[4] Q. Zhang and H. Li, “MOEA/D: A multiobjective evolutionary algorithm based on decomposition,” *IEEE Trans. Evol. Comput.*, vol. 11, no. 6, 2007., pp. 712–731

[5] Deb, K., Agrawal, S., Pratap, A., Meyarivan, T. „A fast elitist nondominated sorting genetic algorithm for multi-objective optimization: NSGA-II“ In M. S. et al. (Ed.), *Parallel Problem Solving from Nature – PPSN VI*, Berlin, 2000., pp. 849–858

[6] Zitzler, E., Laumanns, M., Thiele, L. „SPEA2: Improving the Strength Pareto Evolutionary Algorithm“ Technical Report 103, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich, 2001.

[7] Bradstreet, L. „The hypervolume indicator for multi-objective optimisation: calculation and use“, Doktorski rad, University of Western Australia, 2011.

[8] Brockhoff, D., Trautmann, H., Wagner, T. „On the Properties of the R2 Indicator“ In: Genetic and Evolutionary Computation Conference (GECCO 2012)., 2012., pp. 465–472

[9] Abraham, A., Jain, L., Goldberg, R., „Evolutionary multiobjective optimization: theoretical advances and applications“, ISBN 1852337877 , United States of America, Springer-Verlag London Limited, 2005.

[10] Fonseca, C. M., Fleming P. J. „Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization,” in *Proceedings of the Fifth International Conference on Genetic Algorithms*, S. Forrest, Ed. San Mateo, CA: Morgan Kauffman, 1993., pp. 416–423

Sažetak

Naslov: Evolucijski algoritmi za višekriterijsku i mnogokriterijsku optimizaciju

U sklopu ovog rada opisan je i analiziran višekriterijski problem sa svojom nadogradnjom – mnogokriterijskim problemom. Uz to su predstavljeni i opisani evolucijski algoritmi za njihovo rješavanje, poželjna svojstva tih algoritama i načini procjene njihovih performansi. Algoritmi su i uspješno razloženi na gradivne blokove koji se pojavljuju u gotovo svim evolucijskim algoritmima te namjene (MOEA). Metrike za procjenu performansi MOEA su analizirane i opisane te kasnije korištene u analizi rezultata. Na kraju su uspoređene implementacije tih algoritama napravljene u sklopu ovog rada s postojećim implementacijama na nekim poznatim problemima. Dobiveni rezultati su grafički prikazani i zatim analizirani.

Ključne riječi: višekriterijski problem**,** višekriterijska optimizacija, mnogokriterijski problem, mnogokriterijska optimizacija, evolucijski algoritmi, genetski algoritmi, metrike, hipervolumen

Summary

Title: Evolutionary algorithms for multi and many-objective optimization

In this work multi-objective problem was described and analysed along with its upgrade – many-objective problem. Also, evolutionary algorithms for solving that kind of problems were introduced and described along with their desired characteristics and ways of estimating their performance. Algorithms were successfully decomposed onto building blocks that appear in almost every evolutionary algorithm for that purpose (MOEA). Metrics for estimating MOEA's performance were described and analysed and were later used in result analysis. Finally, implementations of algorithms made within this work were compared with already existing ones on some known problems. Results were showed graphicly and then analysed.

Key words: multi-objective problem, multi-objective optimization, many-objective problem, many-objective optimization, evolutionary algorithms, genetic algorithms, metrics, hypervolume