Usporedba rješenja višeciljne optimizacije na primjeru dimenzioniranja jednostavne brodske konstrukcije

Hrvojić, Veron

Master's thesis / Diplomski rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Mechanical Engineering and Naval Architecture / Sveučilište u Zagrebu, Fakultet strojarstva i brodogradnje**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:235:831099

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Veron Hrvojić

Zagreb, 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mentori:

Izv. prof. dr. sc. Pero Prebeg

Student:

Veron Hrvojić

Zagreb, 2022.

Izjavljujem da sam ovaj rad izradio samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

Zahvaljujem se svojem mentoru izv. prof. dr. sc. Peri Prebegu koji mi je pružio svu svoju stručnu pomoć i dragocjeno vrijeme koliko je već bilo potrebno. Bio je uvijek dostupan, pun razumijevanja i vodio me kako je najviše smisleno prema uspješnom završetku mog diplomskog studija izradom ovog rada.

Zahvaljujem se roditeljima koji su mi pružili sve u životu i kojima sve to dugujem vratiti. Svojim bakama, djedu, prijateljima koji su me dali mnoge okuse života. Zbog dobrih ljudi oko mene tu sam gdje jesam, i malo sam toga sam zaslužio. Bog me zadužio mnogim darovima koje nikako neću moći otplatiti, ali ću dati sve od sebe.

Veron Hrvojić

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za diplomske ispite studija strojarstva za smjerove:
Procesno-energetski, konstrukcijski, inženjersko modeliranje i računalne simulacije i brodostrojarski

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: **602 - 04 / 22 - 6 / 1**	
Ur.broj: **15 - 1703 - 22 -**	

DIPLOMSKI ZADATAK

Student: **Veron Hrvojić** JMBAG: 0035209245

Naslov rada na hrvatskom jeziku: **Usporedba rješenja višeciljne optimizacije na primjeru dimenzioniranja jednostavne brodske konstrukcije**

Naslov rada na engleskom jeziku: **Comparison of the multi-objective optimization results for simple ship structure design example**

Opis zadatka:

Višeciljna optimizacija omogućuje generiranje Pareto fronte koja projektantima daje jasan uvid u mogućnosti poboljšanja proizvoda prema jednom od korištenih ciljeva u odnosu na ostale ciljeve. U ranim fazama projektiranja konstrukcije vrlo često koriste se analitičke metode za proračun odziva, koje je moguće jednostavno implementirati u softver. Tako implementiran matematički model moguće je povezati s optimizacijskim algoritmima, s obzirom da su u programskim jezicima, poput Pythona, dostupne biblioteke s implementacijama različitih višeciljnih optimizacijskih algoritama. U radu je potrebno evaluirati kvalitetu Pareto fronte koju ostvaruju implementacije optimizacijskih algoritama u nekoliko odabranih biblioteka, dostupnih u programskom jeziku Python, na primjeru dimenzioniranja jednostavne brodske konstrukcije.

Zadatak obuhvaća sljedeće:

- pregled optimizacijskih biblioteka dostupnih, u programskom jeziku Python, koje omogućuju optimizaciju višeciljnih problema s ograničenjima
- pregled mjera kvalitete koje se koriste za usporedbu Pareto fronti odnosno nedominiranih rješenja generiranih različitim optimizacijskim algoritmima
- odabir optimizacijskih biblioteka i mjera kvalitete Pareto fronti koje će se koristiti za usporedbu
- povezivanje odabranih biblioteka i odabranih algoritama s postojećom implementacijom modela okvirnog rebra jednostavne brodske konstrukcije u kojem se analiza odziva provodi metodom pomaka
- definiranje višeciljnog optimizacijskog problema dimenzioniranja okvirnog rebra jednostavne brodske konstrukcije te rješavanje istog primjenom odabranih optimizacijskih algoritama
- usporedbu rješenja višeciljne optimizacije ostvarenih različitim optimizacijskim algoritmima na primjeru dimenzioniranja jednostavne brodske konstrukcije.

U radu je potrebno navesti korištenu literaturu i eventualno dobivenu pomoć.

Zadatak zadan:	Datum predaje rada:	Predviđeni datumi obrane:
5. svibnja 2022.	7. srpnja 2022.	18. – 22. srpnja 2022.

Zadatak zadao:
Izv.prof.dr.sc. Pero Prebeg

Predsjednik Povjerenstva:
Prof. dr. sc. Tanja Jurčević Lulić

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

POPIS OZNAKA

Oznaka	Jedinica	Opis
A	-	
b_f	m	širina pojasa I ili T nosača,
C		sposobnost *capability,*
d	-	razlika vektora ciljeva projekta iz dobivene Pareto fronte od najbližeg projekta iz referentne Pareto fronte ili promjer šipkastog presjeka
D		zahtjev, *demand*
d_u	m	unutarnji promjer cjevastog nosača,
d_v	m	vanjski promjer cjevastog nosača,
E	$N \cdot m^{-2}$	Youngov modul elastičnosti,
f, f_m		funkcija cilja,
g	$m \cdot s^{-2}$	gravitacijska konstanta,
g, g_j		ograničenje u obliku nejednadžbe,
GD	-	pokazatelj kvalitete generacijska udaljenost,
h	m	visina struka (rebra) I nosača,
h, h_k		ograničenje u obliku jednadžbe,
h_u	m	unutarnja visina nosača pravokutnog presjeka,
HV	-	pokazatelj (mjera) kvalitete hipervolumen
h_v	m	vanjska visina nosača pravokutnog presjeka,
h_w	m	visina rebra I ili T nosača,
IGD	-	inverzna generacijska udaljenost,
I_{ij}	m^4	moment tromosti presjeka grede između čvorova i i j
I_ε	-	mjera kvalitete epsilon pokazatelj,
J	-	broj ograničenja u obliku nejednadžbi,
K	-	broj ograničenja u obliku jednadžbi,
k_{ij}	$N \cdot m \cdot rad^{-1}$	koeficijent krutosti grede između čvorova i i j na savijanje,
M	-	broj funkcija cilja,
m	-	broj projekata referentne Pareto fronte Z
$M(x)$	N·m	funkcija raspodjele momenta,
M_{ij}	N·m	moment upetosti u čvoru i elastično upete grede između čvorova i i j,
N	-	broj projektnih varijabli,
n	-	broj rješenja dobivenog Pareto skupa,
p	$N \cdot m^{-2}$	tlak od opterećenja,
q	$N \cdot m^{-1}$	kontinuirano opterećenje,
T	m	gaz,
t	m	debljina rebra I nosača,
t_1, t_2	m	debljina pojaseva I nosača,

t_f	m	debljina pojasa I ili T nosača,
t_p	m	debljina pojasa C nosača,
t_s	m	debljina rebra C nosača,
t_w	m	debljina rebra I ili T nosača,
w	m	razmak okvira ili širina C nosača,
w_1, w_2	m	širina pojaseva I nosača,
W_{ij}	m^3	moment otpora,
w_u	m	unutarnja širina nosača pravokutnog presjeka,
w_v	m	vanjska širina nosača pravokutnog presjeka,
x		projektna varijabla ili vektor projektnih varijabli,
Z	-	skup referetnih Pareto rješenja
ρ	$kg \cdot m^3$	gustoća,
σ_{ij}	MPa	normalno naprezanje u nosaču između čvorova *i* i *j*,
ϕ_i	rad	kut zakreta čvora
ψ_i	rad	kut zakreta

SAŽETAK

U diplomskom radu provedena je usporedba kvalitete rezultata optimizacije višeciljnog problema s ograničenjima ostvarenih različitim optimizacijskim algoritmima iz dviju Python biblioteka. Usporedba je, osim na jednostavnom standardnom problemu, provedena i na primjeru okvira jednostavne brodske konstrukcije s tri palube i tri tanka ispod najdonje palube, pri čemu je za analizu odziva korištena metoda pomaka. Da bi se omogućilo formuliranje optimizacijskog problema neovisno o biblioteci koja se koristi za optimizaciju te za automatizirano provođenje optimizacija više optimizacijskih problema optimizacije i njezinu evaluaciju, nadograđena je biblioteka za usporedbu rješenja višeciljnih optimizacijskih problema *moobench*. Biblioteka moobench softver je otvorenog koda dostupan na mrežnom servisu *GitHub.* U okviru rada implementirana su sučelja prema Python bibliotekama Pymoo i jMetalPy i dodana toj biblioteci čime su optimizacijski algoritmi iz tih dvaju biblioteka postali dostupni biblioteci moobench. Isto tako, u biblioteku je implementiran i optimizacijski problem dimenzioniranja okvira jednostavne brodske konstrukcije. Usporedba kvalitete rješenja temeljena je na mjerama kvalitete Pareto fronte koje su u radu također implementirane u moobench.

Ključne riječi: *moobench*, optimizacija, usporedba, mjere kvalitete, Pareto

SUMMARY

In final thesis a comparison on multiobjective constrained optimization results was conducted with a variety of optimization algorithms from two Python libraries. A comparison is done, on a standard benchmark problem, and on model of a simple ship structure, with three decks and three tanks below the lowest deck, optimization problem. The displacement method is used to determine simple ship structure frame response, which is also implemented in Python. In order to enable optimization problem formulation independent of library used for optimization and to automate optimization procedure for multiple runs and evaluation of results, a framework for comparison of multiobjective optimization results *moobench* was used and extended. Moobench is open-source software available at GitHub. In scope of this thesis, two interfaces to Python libraries Pymoo and jMetalPy were implemented and added to moobench, so that optimization algorithms from those two libraries are now available in moobench. Also, a model of simple ship frame design optimization problem is added to moobench. Quality comparison of solutions is based on perfomance indicators for Pareto fronts that are also implemented in moobench in a scope of this thesis.

Key words:*moobench*, optimization, comparison, qualitiy indicators, performance indicators, Pareto

1. UVOD

Gotovo svaka proizvodna tvrtka danas nalazi se na surovom polju tržišnog nadmetanja. One se ekonomski i tehnički natječu kako bi pravodobno prepoznale i odgovorile na potrebe tržišta koje stalno evoluiraju, šire se i postaju zahtjevnije. Proizvodi koje tvrtke razvijaju nastoje zadovoljiti te potrebe svojom ukupnošću traženih svojstava. Proizvode je stoga potrebno neprestano usavršavati što pred projektante postavlja sve teže zahtjeve. Raste njihova složenost, a sam proces konstruiranja treba biti brz i ekonomski isplativ. Stoga, ni malo ne čudi da su se u procese razvoja i konstruiranja probili i postupci optimizacije koji su postali jednostavno dostupni današnjim inženjerima.

1.1. Optimizacija u procesu razvoja proizvoda

Razvojem osobnih računala postalo je moguće provesti i složenije optimizacijske proračune gotovo na svakom suvremenom radnom mjestu strojarskih ili drugih inženjera. Uz sklopovlje računala tj. hardver, potrebna je i odgovarajuća programska podrška koja dolazi u najrazličitim oblicima što se tiče optimizacije ili općenito računalne analize.

U procesu konstruiranja često je prikladno u ranim fazama projektiranja konstrukcije koristiti analitičke modele za različite proračune, primjerice proračun odziva konstrukcije. Oni omogućuju dobivanje smjernica, olakšavaju odabire i omogućuju dobro definiranje nekog predrješenja u fazama kad je malo toga poznato. Takve je analitičke modele moguće oblikovati u softver. Time se omogućuju sve prednosti izvršenja proračuna na računalu poput ponovljivosti, brzine, točnosti i dr., uz pretpostavku da je model točno postavljen, odabrani prikladni alati i sl. Jednom implementiran, matematički model je tada moguće povezati s raznim optimizacijskim postupcima s obzirom da su razvijene, napisane i dostupne mnoge optimizacijske biblioteke u raznim programskim jezicima koje imaju ostvarene mnoge provjerene optimizacijske algoritme koje je iznjedrila znanstvena zajednica. Odličan primjer dostupnosti je količina optimizacijskih biblioteka, modula i programskih okvira dostupnih u programskom jeziku Python.

Otuda i potreba za ovakvim radom, da se na primjeru složenijeg[1] analitičkog modela za proračun jednostavne brodske konstrukcije usporede rješenja koja daju različiti višeciljni optimizacijski algoritmi, s različitim kombinacijama nekih od kontrolnih parametara, kako bi

[1] Složenijeg u usporedbi s testni problemima često dostupnim zajedno s bibliotekama koji su često analitičke, ali zahtjevne funkcije.. Model je predstavljen u poglavlju 4.

se na temelju toga donijeli zaključci koji bi mogli biti od koristi korisnicima višeciljne optimizacije. Naime, pri prvim susretima s višeciljnom optimizacijom moguća je preplavljenost nesigurnošću koje algoritme odabrati s obzirom na problem koji se rješava, kako i koje parametre podesiti i kako procijeniti je li algoritam konvergirao.

1.2. Ukratko o optimizaciji i optimizacijskim algoritmima

Postupkom optimizacije uvijek se nastoji riješiti optimizacijski problem. Optimizacijski problem jest dakle predmet rješavanja, zadatak nad kojim se provodi postupak optimizacije. Optimizacija zahtjeva definiranje optimizacijskog problema u obliku matematičkog modela. Komponente projektnog ili optimizacijskog problema općenito su: projektne varijable, ciljevi[2] i ograničenja. Optimizacijske probleme rješavaju optimizacijske metode odnosno optimizacijski algoritmi.

U postupku optimizacije projektant može na jedan problem postaviti jedan ili više ciljeva. S obzirom na to kakve probleme rješavaju, postoji podjela optimizacijskih algoritama na jednociljne i višeciljne. Uz to, postoji podjela na determinističke, stohastičke i algoritme matematičkog programiranja.

Deterministički algoritmi sadrže potpuno određen način pretraživanja prostora. Od njih se traži da uvijek pronađu globalni optimum u konačnom vremenu. Neke skupine determinističkih algoritama su:

- Pohlepni algoritmi (*greedy*)
- Grananje i ograničavanje (*branch and bound*)
- Algoritam penjanja uzbrdo (*hill-climbing*)
- Pretraživanje u širinu i pretraživanje u dubinu nad grafom[3] (*breadth-first, depth-first*)
- Algoritmi bazirani na diferencijalnom i integralnom računu

Stohastičkim algoritmima s druge strane svojstven je određen stupanj nasumičnosti. U dijelu koraka u stohastičkim algoritmima inheretno je ugrađen i stupanj nasumičnosti. Neke skupine stohastičkih algoritama su:

- Slučajno pretraživanje (*random search*)
- Monte Carlo

[2] Ciljevi se još nazivaju "atributi s aspiracijom". Atributi jer su svojstvo sustava odnosno problema, primjerice konstrukcije. Riječ aspiracija označava težnju za poboljšanjem. Dakle, to su neki atributi sustava, problema, konstrukcije koje želimo poboljšati: smanjiti aerodinamički otpor, smanjiti masu, povećati krutost, itd.
[3] Graf je u računarstvu još jedna u nizu struktura podataka.

- Evolucijske strategije (*evolutionary*)
- Algoritmi rojeva – mrava, pčela, čestica (*swarm*)
- Tabu pretraga (*tabu search*)
- Simulirano žarenje (*simulated annealing*)

Algoritmi matematičkog programiranja koriste se načinima svojstvenim matematici za dosezanje optimuma putem operacija interpolacije, derivacija, integracija i korištenjem drugih matematičkih zakonitosti i teorema. Neki algoritmi su:

- Linearno programiranje (*Linear programming*)
- Kvadratično programiranje (*Quadratic programming*)
- Nelinearno programiranje – sekvencijalno linearno (SLP) i sekvencijalno kvadratično (SLQP) programiranje

Svaka skupina ima sebi svojstvene prednosti i nedostatke pa su stoga prikladni u različitim situacijama.

1.3. Realni problemi u optimizaciji

Realni proizvodi složeni su i svaki inženjerski odabir učinjen tijekom procesa razvoja na nekom elementu proizvoda ili sustava nekako se odražava na cijeli niz drugih elemenata i aspekata s kojima je odabrani element u interakciji. Uobičajena je karakteristika realnog proizvoda ili tehničkog sustava da ga najčešće opisuje velik broj parametara koje treba definirati tj. izabrati. Svi ti parametri mogu postati projektne varijable. Uz to, velik je broj proizvodnih ili praktičnih ograničenja koja se moraju zadovoljiti. Uspješnost konstrukcije možemo najčešće ocijeniti s barem nekoliko mjera kvalitete. Svaka od njih može postati funkcija cilja.

1.4. Organizacija rada

Nakon ovog poglavlja u kojem je dan je uvod u optimizaciju, u poglavlju 2. prikazan je opis višeciljne optimizacije, kratak uvod u mjere kvalitete i pregled četiriju mjera kvalitete koje će se koristiti za usporedbu rješenja. U poglavlju 3. slijedi pregled optimizacijskih biblioteka u Pythonu, od kojih su odabrane dvije: Pymoo i jMetalPy. One su detaljnije opisane u poglavljima 3.2. i 3.3. U poglavlju 4. slijedi detaljan opis problema okvirne konstrukcije pontona zajedno s analitičkom metodom pomaka. Poglavlje 5. opisuje biblioteku *moobench* koja je značajno proširena i nadopunjena tijekom izrade ovog rada, a koje se koristila za izvedbu optimizacije. U poglavlju 6., dan je prikaz rezultata optimizacije dobivenih s više optimizacijskih algoritama te usporedba rezultata na osnovu ostvarenih mjera kvalitete Pareto fronte. Na kraju slijedi zaključak i pregled literature.

2. Višeciljna optimizacija i mjere kvalitete Pareto fronte

Jednociljna optimizacija obično ima za zadatak minimizaciju funkcije cilja. Preoblikovanjem te funkcije cilja može se jednostavno postići maksimizacija ili postizanje određene vrijednosti realnog atributa kod predmeta optimizacije, recimo nosive konstrukcije. Ono što je ključno jest da jednodimenzionalnost u prostoru atributa jamči da će postupak optimizacije iznjedriti maksimalno jedno rješenje – ukoliko postupak završi uspješno. A stvarni proces razvoja proizvoda je takav da se za daljni razvoj nakon sinteze[4] najčešće odabire jedno rješenje.

Višeciljna optimizacija inheretno je različita. Zbog višedimenzionalnosti prostora funkcija cilja, tzv. atributnog prostora (detaljno u [2], poglavlje 3.1.2.), kao izravni rezultat optimizacije dobivaju se skupovi rješenja. Pojednostavljeno, za razliku od jednociljne optimizacije u kojoj svaka iteracija vraća jedno najbolje rješenje, u višeciljnoj optimizaciji svaka iteracija vraća skup nedomiranih rješenja[5] (vidi poglavlje 2.2.). Naime, često nije moguće jednoznačno i izravno zaključiti koje je rješenje najbolje, već su konačna rješenja kompromisna jedno u odnosu na drugo – tj. nijedno rješenje nije bolje u svim aspektima. Izdvojiti najbolje rješenje iz skupa rješenja može se jedino po projektantovim preferencijama, za što je uobičajeno potrebno iskustvo. Za dobivanje konačnog skupa rješenja koristi se vrlo važan i opći koncept nedominiranosti. Za uspoređivanje rješenja višeciljne optimizacije potrebne su određene mjere kvalitete koje će obraditi skup nedominiranih rješenja i tako ukazati na njegovu kvalitetu.

2.1. Višeciljna optimizacija

Općeniti optimizacijski problem (1) ima M ciljeva koje je potrebno minimizirati ili maksimizirati mijenjajući pritom N projektnih varijabli na raspolaganju. Pritom je potrebno zadovoljiti J ograničenja u obliku nejednakosti i K ograničenja u obliku jednakosti. Ograničenja su jednadžbe koje smanjuju stupnjeve slobode pretrage, tj. smanjuju projektni prostor, domenu projektnih varijabli uvodeći neizvediva[6] područja. Projektne varijable obično se moraju nalaziti između neke gornje i donje konačne granice što također utječe na projektni prostor. Cilj je dakle kod višeciljne optimizacije pronaći skup rješenja tj. skup vektora varijabli koje zadovoljavaju

[4] Sinteza kao dio razvojnog procesa. Proces kada se strukturiranim matematičkim i drugim metodama dolazi do rješenja koje se dalje razrađuje.
[5] Svaka iteracija zapravo vraća skup rješenja širi od samo nedominiranih rješenja. Sva rješenja koja su sortirana u fronte. Optimalna rješenja su nedominirana rješenja, koja pripadaju prvoj od tih fronti.
[6] Neizvediva područja smanjuju područja pretrage jer obično praktično rješenja koja bi u njih spadala su teško ostvariva, ekonomski neisplativa, tehnološki neizvediva, fizikalno nemoguća, itd.

sva ograničenja i imaju što bolje vrijednosti funkcije cilja. Prilikom minimizacije, bolje su niže vrijednosti funkcija cilja.

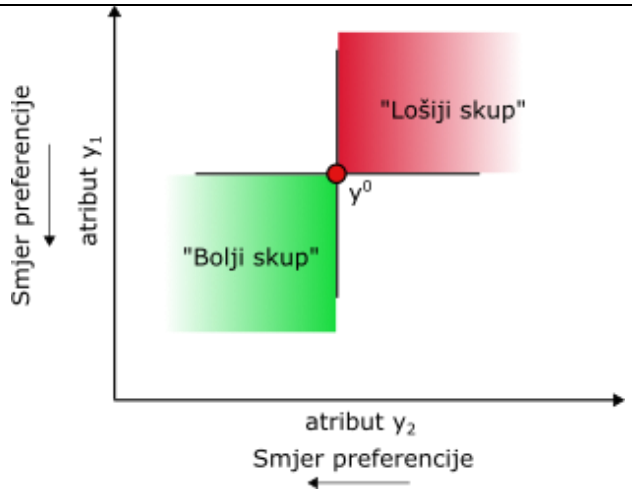
$$\begin{aligned} \min \quad & f_{\mathrm{m}}(x) && m = 1, \dots, M, \\ tako\ da\ & g_{\mathrm{j}}(x) \leq 0, && j = 1, \dots, J, \\ & h_{\mathrm{k}}(x) = 0, && k = 1, \dots, K, \\ & x_{\mathrm{i}}^{\mathrm{L}} \leq x_{\mathrm{i}} \leq x_{\mathrm{i}}^{\mathrm{U}}, && i = 1, \dots, N. \end{aligned} \tag{1}$$

2.2. Pareto fronta

Koncept nedominiranost vrlo je važan u višeciljnoj optimizaciji, jer je njime definiran skup optimalnih rješenja. Optimalni skup rješenja zove se nedominirani skup, a rješenja iz tog skupa nedominirana rješenja. Zamislimo skup izvedivih rješenja, tj. skup projekata[7] raspršen u prostoru. Zatim zamislimo neki projekt $\mathbf{y^0}$ u atributnom prostoru. Radi lakoće vizualizacije, neka postoje dva cilja (M = 2). Za oba cilja neka se vrši minimizacija tj. niža vrijednost cilja predstavlja preferirano rješenje. Preferencija projektanta ugrađena je[8] dakle u samo funkciju cilja s obzirom da algoritam ovdje pretpostavlja njenu minimizaciju. Ako je tako, tada se može definirati „bolji skup“ od $\mathbf{y^0}$ i „lošiji skup“ od $\mathbf{y^0}$ [Slika 1.]. „Boljem skupu“ pripadaju oni projekti kojima su svi atributi preferirani u odnosu na odabrani projekt $\mathbf{y^0}$. „Lošijem skupu“ pripadaju oni projekti nad kojima $\mathbf{y^0}$ ima preferirane sve atribute. Još se kaže da – $\mathbf{y^0}$ dominira nad svim projektima toga skupa. Kao što svaki projekt iz „boljeg skupa“ dominira nad projektom $\mathbf{y^0}$**.** Dakle, kada projekt dominira nad nekim drugim projektom, tada mu je barem jedan atribut preferiran u odnosu na drugi projekt, a svi ostali atributi barem jednaki ili preferirani u odnosu na drugi projekt.

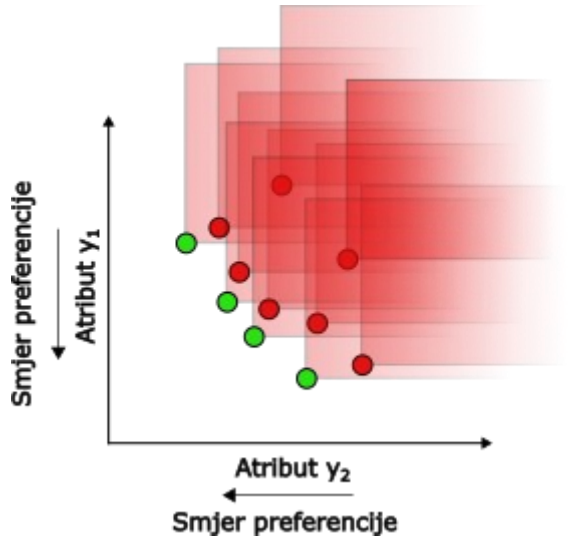
[7] Projekt je pojednostavljeno definiran kao moguć skup projektnih varijabli koje ga definiraju, s pripadajućim vrijednostima atributa.

[8] Moguće je jednostavno preoblikovati funkciju da se minimizacijom funkcije cilja maksimizira određeni atribut.



Slika 1. Definicija „boljeg“ i „lošijeg“ skupa

Konačno, skup nedominiranih projekata tj. optimalnih rješenja optimizacije jest definiran kao onaj koji nema bolji skup od sebe, tj. nijedan projekt skupa nije dominirano od strane nijednog drugog projekta [2]. Slika takvog nedominiranog skupa rješenja u atribunom prostoru naziva se i Pareto fronta jer u dvodimenzionalnim i trodimenzionalnim grafičkim prikazima taj skup prikazan u obliku točki oblikuje frontu poput granice potprostora, u dvodimenzionalnom prostoru krivulju, a u trodimenzionalnom prostoru površinu [7]. Odsada nadalje – skup nedominiranih rješenja češće će se nazivati Pareto fronta.



Slika 2. Pareto fronta

Ovdje je sve prikazano na dvodimenzionalnom prostoru atributa, no princip je sasvim opći i primjenjiv i na višedimenzionalne prostore. Tipični izgled Pareto fronte prikazan je na [Slika 2.]. Zelene točke predstavljaju projekte koji dominiraju nad crvenim i čine Pareto frontu.

2.3. Mjere kvalitete Pareto fronte

Iz Pareto fronte moguće je saznati mnogo toga, stoga postoje razni načini mjerenja njezine kvalitete. Kako Pareto fronta predstavlja konačno rješenje optimizacijskog algoritma, mjerenjem njezine kvalitete moguće je donositi zaključke o kvaliteti samog algoritma. To se opsežno koristi u razvojnim procesima novih stohastičkih i drugih višeciljnih optimizacijskih algoritama prilikom kojih se novi algoritmi tipično uspoređuju s nekoliko drugih široko priznatih suvremenih algoritama. Poželjno je da optimizacijom dobivena Pareto fronta bude [2]:

- jednoliko popunjena,
- što bliža teoretskoj Pareto fronti i
- u potpunosti razapeta između rubova atributnog prostora.

S obzirom da je jedan od poželjnih ishoda da dobivena Pareto fronta bude što bliža teoretskoj – očito ju je potrebno i poznavati. No, teoretsku frontu moguće je dobiti jedino putem kontinuiranog matematičkog modela što kod mnogo realnih problema jednostavno nije moguće. Zato se teoretska fronta aproksimira Pareto frontom dobivenom nekim priznatim optimizacijskim algoritmom, a da se pritom postupak vrši s takvim postavkama da se osigura vrlo kvalitetna Pareto fronta.

Važno je da se za mjerenje definira računalna oprema i provedeni usporedni postupci budu izvedeni u jednakim uvjetima. To je uvjet za dosljednu kvantitativnu usporedbu rezultata višeciljnih optimizacijskih algoritama [2]. U nastavku će ukratko biti predstavljene četiri mjere kvalitete implementirane u biblioteci jMetalPy, a koje su korištene za usporedbu rezultata optimizacije u ovom radu. To su: generacijska udaljenost, inverzna generacijska udaljenost, hipervolumen i epsilon indikator. Generacijska udaljenost, inverzna generacijska udaljenost i epsilon indikator su binarne, a hipervolumen unarna mjera. Unarna mjera zahtjeva da joj se proslijedi jedan skup rješenja tj. jedna aproksimacija Pareto fronte, dok je binarnim mjerama potrebna uz dobivenu i referentna Pareto fronta.

2.3.1. Generacijska udaljenost

Generacijska udaljenost (skraćeno *GD*) mjera je blizine dobivene fronte referentnoj Pareto fronti. Ako je skup rješenja koje je pronašao algoritam definiran kao $A = \{a_1, a_2, \dots, a_n\}$ i referentni skup $Z = \{z_1, z_2, \dots, z_m\}$, tada se *GD* izračunava na sljedeći način:

$$GD(A) = \frac{1}{n}\left(\sum_{i=1}^{n} d_i^p\right)^{\frac{1}{p}}, \quad \textbf{(2)}$$

gdje su:

GD(*A*) - vrijednost pokazatelja za skup A,

n - broj rješenja skupa *A*,

d_i - razlika koordinata a_i iz skupa *A* do najbližeg iz rješenja iz referente Pareto fronte,

Dio izraza u zagradi predstavlja poopćenu formulu za udaljenost. Najčešće je p=2, čime se dobiva formula za Euklidsku udaljenost. Konačna vrijednost *GD* predstavlja prosječnu udaljenost rješenja iz skupa *A* do najbližeg rješenja iz skupa *Z*.

2.3.2. Inverzna generacijska udaljenost

Inverzna generacijska udaljenost (skraćeno *IGD*) također je mjera blizine dobivene fronte referentnoj Pareto fronti. Kod ovog indikatora udaljenost se mjeri na način da se za svaku točku iz referentnog skupa *Z* (referentne Pareto fronte) traži najbliža točka iz skupa *A* (optimizacijom dobivenog) te na taj način računa prosječna udaljenost rješenja:

$$IGD(A) = \frac{1}{m}\left(\sum_{i=1}^{m} \hat{d}_i^p\right)^{\frac{1}{p}}, \quad \textbf{(3)}$$

gdje su:

IGD(*A*) - vrijednost pokazatelja za skup A,

m - broj rješenja u skupu *Z*,

$\hat{d}_i$ - razlika koordinata udaljenost rješenja z_i iz skupa *Z* do najbližeg

Dio izraza u zagradi predstavlja poopćenu formulu za udaljenost. Najčešće je p=2, čime se dobiva formula za Euklidsku udaljenost. Konačna vrijednost *IGD* predstavlja prosječnu udaljenost rješenja iz skupa *Z* do najbližeg rješenja iz skupa *A*.

2.3.3. Hipervolumen (HV)

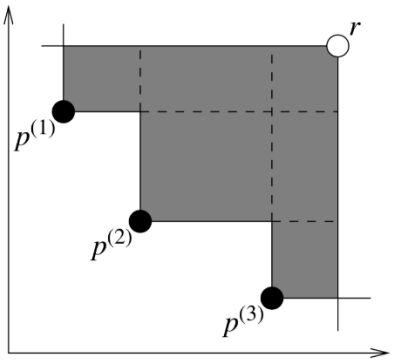
Hipervolumen je mjera koja pokazuje kombinirano konvergenciju i distribuciju u obliku jedne vrijednosti za danu frontu [5][6]. Da bi se izračunala *HV*, umjesto referentne fronte, potrebna je referetna točka, tj. referetni projekt. S obzirom da referentna fronta često nije poznata kod višeciljnih složenih realnih problema, ovaj pokazatelj je primamljiv za procjenu rješenja u

takvim slučajevima. Moguće je formirati pokazatelj i pomoću referente fronte na način da se načini razlomak [7].

Hipervolumen je jednak iznosu površine/volumena[9] nad kojim dominira pronađeni Pareto skup rješenja *p* u odnosu na referetnu točku [Slika 3.]. Pareto skup *p* na slici sastoji se od točaka $p^{(1)}$, $p^{(2)}$ i $p^{(3)}$. Referentna točka označena je kao *r*.

Formiranje pokazatelja korištenjem Pareto fronte moguće je načiniti na sljedeći način. Ako je *HV*(*A, r*) vrijednost hipervolumena aproksimacije Pareto skupa *A* s referetnom točkom *r* i *HV*(*Z, r*)-referentne Pareto fronte, može se načiniti razlomak:

$$HR(Z,A,r) = \frac{HV(A,r)}{HV\ (Z,r)} \tag{4}$$



Slika 3. Ilustracija hipervolumena (preuzeto s [4])

Relativna vrijednost dva pokazatelja hipervolumena nad istim setovima ovisi o izboru referentne točke. Dakle, ne može se reći da je uvijek bolja ona fronta koja ima veći iznos *HV* pokazatelja što je pokazano u [7]. Kao referentnu točku za izračunavanje ovog pokazatelja kvalitete za realne probleme potrebno je izabrati smisleno, primjerice kao „najlošije“ zamislivo rješenje. No, ipak, bolji je pristup izabrati nešto lošiju točku od NADIR točke koju treba aproksimirati [7]. S obzirom da za izračunavanje ostalih pokazatelja koji se planiraju izračunati u ovom radu je potrebno poznavati referetnu frontu, tako će se na temelju te referetne fronte formirati nešto lošija točka od NADIR točke. NADIR točka predstavlja točku u atributnom prostoru koja se dobije dekompozicijom višeciljnog problema na više jednociljnih koji se zasebno optimiziraju i to maksimiziraju [7]. Predstavlja, takoreći točku suprotnu idealnom smjeru kretanja optimizacije što je kod minimizacije – pojednostavljeno rečeno – prema ishodištu koordinatnog sustava.

[9] Općenito, volumen n-dimenzionalnog prostora.

No, iako se ovisno o izboru referentne točke mogu općenito dobiti različiti relativni odnosi dva indikatora nad potpuno istim nedominiranim setovima *A* i *B*, hipervolumen ima jedno važno svojstvo. Hipervolumen je jedini unarni indikator (preslikava skup Pareto fronte u jedan realan broj) koji je striktno monoton. To znači da ako jedna aproksimacija Pareto skupa *A* striktno dominira nad drugom *B,* tada vrijedi, $HV(A, r) > HV(B, r)$. To znači da ako barem jedno rješenje iz *A* striktno dominira nad svim rješenjima iz *B*, bit će pokazatelj hipervolumena od A veći od pokazatelja od skupa *B.* Tada i promjenom referentne točke *HV* striktno dominantnog skupa će biti veći od *HV* pokazatelja drugog skupa. Referentna točka, za dvije usporedbe uvijek mora biti jednaka, tj. ne mogu se uspoređivati dvije vrijednosti s razlilčitom referentnom točkom. Na primjer *HV(A, r)* i *HV(B, r')* se ne mogu uspoređivati. Tako da unatoč spomenutoj relativnosti, ipak se može reći da veća vrijednost predstavlja bolju konvergenciju i distribuciju od niže vrijednosti.

Takvo je značenje implementirano u jMetalPy biblioteci, tj. veća vrijednost *HV* predstavlja općenito veću kvalitetu fronte [5].

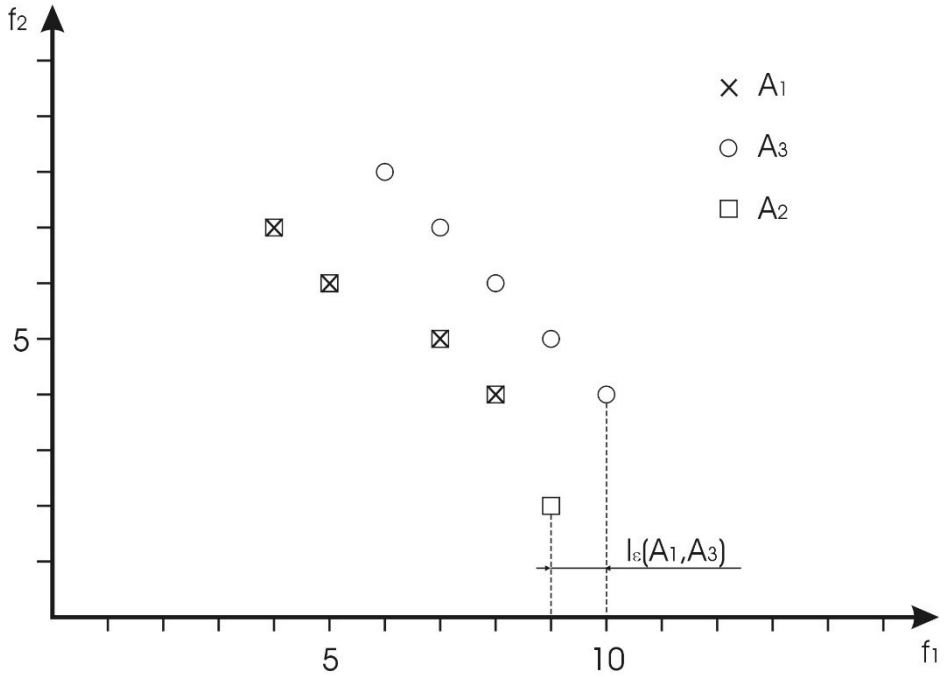
2.3.4. Epsilon pokazatelj (I_ε)

Epsilon pokazatelj još je jedna mjera za konvergenciju rješenja [5] i kvantitativno definira za koliko je jedan skup rješenja lošiji od drugog [2]. Smatra se naprednijom mjerom konvergencije od *GD* i *IGD.* U jMetalPy-u implementirana je aditivna verzija epsilon pokazatelja.

Pokazatelj I_ε uspoređuje dva skupa. Dakle, izračunati skup, moguće je uspoređivati s drugim ili s referentnom Pareto frontom. I_ε jednak je minimalnom faktoru ε koji definira da postoji barem jedno rješenje u skupu *A*, koje nije lošije za iznos faktora ε po svim ciljevima za bilo koje rješenje u skupu *B*. tj. *A* slabo dominira *B*. Da skup *A* slabo dominira nad skupom *B* znači da je svaki projekt iz *B,* tj. $x_B \in B$ slabo dominiran barem jednim projektom iz A, tj $x_A \in A$. Da projekt slabo dominira nad drugim projektom znači da nije lošiji po nijednom cilju. Tj. ako projekt x_A slabo dominira projekt x_B to to znači da slika projekta nije lošija u nijednom cilju od slike projekta. Tj, $f(x^1)$ nije lošiji od $f(x^2)$ u svim ciljevima

Iz psilon indikatora izveden je aditivni epsilon indikator, čija je implementacija dostupna u jMetalPy-u. Na primjer, na [Slika 4.] prikazano je određivanje epsilon pokazatelja. Definirana su tri skupa A_1, A_2 i A_3. Tako je $I_\varepsilon(A_2, A_1) = 1$, $I_\varepsilon(A_2, A_3) = \frac{9}{10}$. Kod jednociljnih problema, $I_\varepsilon(A,B)$ je odnos dvije vrijednosti cilja, koje predstavljaju *A* i *B*.

Detaljan i formalan opis epsilon pokazatelja, kao i definicije relacija nad projekatima i skupovima može se pronaći u [2].



Slika 4. Ilustracija epsilon indikatora [2]

3. PREGLED OPTIMIZACIJSKIH BIBLIOTEKA ZA VIŠECILJNU OPTIMIZACIJU S OGRANIČENJIMA

Korištenje Pythona u različitim područjima istraživanja općenito je u porastu zato što je to jezik visoke razine, iznimno čitljiv, a njegova sintaksa bliska je ljudskom jeziku pa ga je relativno jednostavno naučiti i razumijeti. Uz to, kompatibilan je s različitim OS platformama. Python je interpreterski jezik, u kojem je relativno lako razvijati, ali koji je sporiji pri izvršavanju u odnosu na neke prevođene jezike (*compile*) poput C ili C++ što je važno pri rješavanju složenih problema poput optimizacijskih. Python nudi bogatu paletu raznih biblioteka i programskih okvira, pa tako i optimizacijskih. Pregledan tablični prikaz dan je u [Tablica 1.] koja je preuzeta iz [3].

U poglavlju 3.1. dan je opis programskih okvira i biblioteka iz u [Tablica 1.]. Na kraju poglavlja predstavljene su odabrane biblioteke koje su opisane u narednim poglavljima 3.2. i 3.3.

Tablica 1. Programski okviri i biblioteke za višeciljnu optimizaciju

Ime	Licenca	Fokus na višeciljnu opt.	Čisti Python jezik	Vizualizacija	Odluči-vanje
jMetalPy	MIT	DA	DA	DA	NE
PyGMO	GPL-3.0	DA	NE	NE	NE
Platypus	GPL-3.0	DA	DA	NE	NE
DEAP	LGPL-3.0	NE	DA	NE	NE
Inspyred	MIT	NE	DA	NE	NE
Pymoo	Apache 2.0	DA	DA	DA	DA

3.1. Opis programskih okvira i biblioteka u Pythonu

Poznata i često korištena biblioteka jMetal razvijena u Javi dostupna je i u programskom jeziku Python u biblioteci jMetalPy. Autori su je proširivali kako bi obuhvatili širok skup funkcionalnosti, poput analize podataka i vizualizacije rješenja. Naknadno je moguće napraviti analizu mjera kvalitete izvršenog eksperimenta. Trenutna dostupna verzija je 1.5.5.

PyGMO, što je skraćenica za Paralelni globalni višeciljni optimizator (*Parallel Global Multiobjective Optimizer*), je optimizacijska biblioteka specijalno namijenjena provedbi

paralelnih izračuna. Opsežni zadaci mogu se pomoću ove biblioteke na jednostavan način distribuirati na nekoliko procesorskih jedinica kako bi se bolje iskoristili dostupni resursi i smanjilo vrijeme izračuna. Zahvaljujući korištenju tzv. općeg modela otoka, dozvoljava korištenje asinkronih i distribuiranih algoritama.

Platypus je višeciljni optimizacijski programski okvir koji nudi implementacije najnovijih priznatih algoritama. Omogućava korisniku da kreira eksperiment s različitim algoritmima i provede naknadnu analizu rezultata pomoću izračuna mjera kvalitete i vizualizacije. Vizualizacija pritom nije na poseban način implementirana u Platypus okvir, već se korisnik treba koristiti drugim bibliotekama namijenjenim crtanju grafičkih prikaza poput matplotlib. Platypus ipak sprema rezultate na način da ih je relativno jednostavno dohvatiti. Prvenstveno se koristi evolucijskim strategijama s naglaskom na višeciljne evolucijske algoritme.

DEAP (skraćenica za distribuirani evolucijski algoritmi, *Distributed Evolutinary Algorithms*) više je platforma za razvijanje evolucijskih algoritama, ponajviše za brzo prototipiranje novih algoritama i njihovo idejno testiranje. Iako nije usmjeren na višeciljnu optimizaciju, zbog modularnosti i proširljivosti mogu se razviti višeciljni algoritmi. Koristeći implementirani okvir nekih standardnih operacija koji su dijelom evolucijskih izračuna, korisnik može mnogo brže doći do ostvarenja svoje zamisli. Uz to, podržani su paralelizacija i distribuiranje zadataka, koje je moguće jednostavno ostvariti uz par linija programskog kôda.

Slično DEAP-u, Inspyred je programski okvir za kreiranje algoritama, i to prirodom inspiriranih algoritama računalne inteligencije. Zbog modularnosti okvira, moguće je implementirati višeciljne algoritme.

Pymoo biblioteka pruža najnovije priznate algoritme s fokusom na višeciljnu optimizaciju iako nudi i mnogo algoritama za jednociljnu optimizaciju. Nudi i jednociljne i višeciljne probleme s i bez ograničenja da se olakša testiranje algoritama na riješenim primjerima. Uz to, cilj ove biblioteke jest pružiti podršku kroz mnoge druge aspekte poput vizualizacije rezultata, mjera kvalitete skupa rezultata, paralelnog i distribuiranog izračunavanja. Biblioteka je napravljena da bude modularna i proširljiva.

Tu su još neki razni drugi popularni okviri u drugim programskim jezicima poput: PlatEMO (Matlab), MOEA i jMetal (Java), jMetalCpp i PaGMO (C++).

3.1.1. Optimizacijske biblioteke odabrane za daljnji rad

S obzirom na predmet rada, odabrane su dvije biblioteke: Pymoo i jMetalPy.

One nude spremne različite višeciljne algoritme i jednostavno sučelje prema optimizacijskom problemu koji je posebno modeliran. Nude vizualizacijske alate prikladne za vizualnu usporedbu rješenja višeciljne optimizacije. Tu najčešće spada crtanje grafova u prostoru ciljeva[10]. Pružaju i naknadni izračun mjera kvalitete rješenja što je važan čimbenik usporedbe rješenja.

U sljedećim potpoglavljima 3.2 i 3.3 ukratko će se predstaviti neke specifičnosti obje biblioteke. Takav opis nije dovoljan za vlastito korištenje njima, već je prosto pregledan kako bi se uputilo čitatelja na neke osnovne zahtjeve pojedinih biblioteka. Za detaljnije informacije o korištenju, najbolje je proučiti dostupnu dokumentaciju ([4], [7])

3.2. Pymoo

Pymoo biblioteka razvijena je sa svrhom da ponudi sveobuhvatni skup alata koji pruža niz mogućnosti vezanih uz jednociljnu ili višeciljnu optimizaciju. Zbog njezine modularne strukture, moguće ju je proširiti vlastitim algoritmima pritom koristeći postojeće operatore, ili postojećim algoritmima dodati vlastite prilagođene operatore. Za testiranje algoritama, biblioteka pruža skup problema s ograničenjima i bez ograničenja, za jednociljnu i višeciljnu optimizaciju sa poznatim Pareto frontama. Za opsežne zadatake nudi mogućnost paralelizacije evaluacije funkcija, nudi razne metode vizualizacije i za višedimenzionalne prostore, i nekoliko alata za višekriterijsko odlučivanje.

Prednost je i ta što je biblioteka dobro dokumentiranas mnogim isječcima programskog kôda koji pokazuju korištenje. Također, biblioteka je relativno jednostavno dostupna za instalaciju preko PyPI centralnog repozitorija putem upravitelja paketima, tzv. pip-a (*package installer for Python*).

3.2.1. Formuliranje problema

U Pymoo biblioteci optimizacijski problemi općenito su formulirani kao minimizacijski. Svaku funkciju cilja potrebno je postaviti tako da se njezinom minimizacijom postiže željeni rezultat. Primjerice, ako funkciju cilju želimo maksimizirati, možemo isti rezultat postići minimizacijom te funkcije cilja pomnožene s minus jedan.

Druga konvencija je da sva ograničenja u obliku nejednadžbi trebaju biti formulirana tako da vrijednost lijeve strane manja ili jednaka od nule. Jednako, sve nejednadžbe koje su suprotnog

[10] Prostor ciljeva razapinju osi pojedinih ciljeva. Primjerice, za optimizaciju sa dva cilja moguće je generirati jednostavni graf - apscisa predstavlja vrijednosti jednog cilja, a ordinata vrijednosti drugog cilja – za sva rješenja Pareto fronte.

znaka uspoređivanja treba pomnožiti s minus jedan. Preporuča se normalizacija svih ograničenja kako bi im se dala jednaka važnost [3]. To matematički definira problem.

Programski, optimizacijski problem definiramo kao klasu po paradigmi objektno orijentiranog programiranja. Tu klasu treba definirati korisnik u vlastitom modulu[11] definirajući tako optimizacijski problem. Korisnička klasa koja predstavlja problem treba nasljediti Pymoo klasu Problem. U konstruktoru korisničke klase potrebno je proslijediti konstruktoru roditeljske klase Problem nekoliko svojstva optimizacijskog problema u obliku programskih varijabli: broj projektnih varijabli kao integer (n_var), broj funkcija cilja kao integer (n_obj), broj ograničenja kao integer (n_constr) i gornje i donje granice projektnih varijabli u obliku NumPy polja. To se najčešće čini putem funkcije super() i prosljeđivanjem imenovanih argumenata (keyword arguments). Korisnička klasa treba prepisati jednu nužnu metodu roditeljske klase - _evaluate. Ta metoda ima za argumente dvodimenzijsko NumPy polje x s n redova i m stupaca i rječnik[12] out. Svaki red predstavlja jedno rješnje, tj. jedinku po rječniku evolucijskih strategija i algoritama. Svaki stupac predstavlja jednu projektnu varijablu – tj. gen. Jedinka dakle posjeduje sekvencu gena koja ju određuje. Metoda _evaluate provodi potrebne izračune nad modelom problema, i zapisuje vrijednosti funkcija cilja pod ključem „F“ u obliku polja, a vrijednosti ograničenja pod ključe „G“ u obliku polja (vidi poglavlje 3.2.5.).

Pymoo pruža i nekoliko predefiniranih problema koji služe za procjenu izvođenja vlastitih algoritama i operatora na testnim problema. Pomoću testnih problema moguće je otkriti za koju klasu problema je algoritam prikladan.

3.2.2. Inicijalizacija algoritma

Nakon definiranja problema, sljedeći je korak definiranje algoritma. Algoritam je uređeni postupak koji će voditi rješavanje problema tijekom cijelog izračuna. Programski, algoritam je implementiran kao objekt kojeg je potrebno instancirati za optimizaciju. Svaki algoritam zasebna je klasa sa vlastitim argumentima koje očekuje[13]. Putem tih argumenata moguće je znatno utjecati na konačni rezultat, brzinu konvergencije i sl. Te je argumente potrebno proslijediti konstruktoru algoritma prilikom njegova instanciranja.

3.2.3. Algoritmi

[11] Python datoteka može se nazivati modul i ima datotečni nastavak .py.
[12] Rječnik (*dictionary*) je python-ov tip podataka koji se sastoji od parova vrijednosti. Prvi element u paru naziva se ključ, a drugi vrijednost.
[13] Detaljna dokumentacija pruža API (*Application programming interface*) za svaki algoritam, tj. očekivano sučelje za prilagođavanje svakog algoritma putem parametara koji su predviđeni za prosljeđivanje objektu algoritma. API svakog algoritma dakle definira moguće argumente i što oni predstavljaju.

Većina algoritama u pymoo-u je inspirirana evolucijom. Takvi algoritmi koriste se operatorima koji predstavljaju događaje u procesu evolucije poput: borbu za pravo na razmnožavanje, razmjenu geneteskog materijala, nasumičnu promjenu genetskog materijala, To su radnje odabira, križanja, mutacije, preživljavanja koje imaju svoje pripadajuće operatore u sklopu genetskih algoritama. Operatori su ostvareni u zasebnim modulima, a algoritmima se prosljeđuju objekti koji utjelovljuju pojedini operator. Tako je na vrlo jednostavan način moguće prilagoditi izvođenje algoritma. Takva arhitektura nudi prednosti poput mogućnosti korištenja operatora za kreiranje vlastitih algoritama, pojednostavljenje programskog kôda, bolja čitljivost, lakše proširivanje modula vlastitim funkcionalnostima.

Operator preživljavanja obično je središnji dio samog algoritma [4], pa se algoritmi uglavnom po njemu razlikuju.

Popis algoritama sadržani u Pymoo dan je u sljedećoj tablici [Tablica 2.].

Tablica 2. Popis algoritama u Pymoo biblioteci

Jednociljni	Višeciljni
Genetic algorithm	NSGA-II
Differential evolution	R-NSGA-II
Biased Random Key Genetic Algorithm	NSGA-III
Nelder Mead	U-NSGA-III
Pattern Search	R-NSGA-III
CMAES	MOEAD
Evolutionary Strategy	AGE-MOEA
Stochastic Ranking Evolutionary Strategy	C-TAEA

3.2.4. Operatori

Popis operatora sadržanih u Pymoo dani su u sljedećoj tablici [Tablica 3.]. Pomoću njih vrše se glavne prilagodbe svakog genetskog algoritma. Detalje o svakom operatoru moguće je pronaći u dokumentaciji [4].

Tablica 3. Popis operatora Pymoo biblioteke

Uzorkovanje	Selekcija
Random	Random
Latin Hypercube Sampling	Tournament Selection
Random Permutation Sampling	
Mutacija	**Križanje**
Polynomial	Simulated Binary
Bitflip	Uniform
Inverse Mutation	Half Uniform
	Differential Evolution
	One Point
	Two Point
	K Point
	Exponential
	Order Crossover
	Edge Recombination Crossover

3.2.5. Optimizacija

Za optimizaciju potrebno je još definirati kriterij zaustavljanja (termination criterion). On se može definirati na nekoliko načina od kojih su najjednostavniji: broj generacija, broj provedbi funkcije _evaluate ili vrijeme izvršavanja. Uz te, mogući su i napredniji kriteriji, koji se temelje na praćenju nekih mjera tijekom izvođenja. U Pymoo biblioteci implementirani su dva takva kriterija: praćenje promjena tj. pomaka u prostoru projektnih varijabli i u prostoru ciljeva.

Optimizacija se poziva putem funkcije minimize kojoj se prosljeđuju svi spomenuti objekti i argumenti: problem, algoritam, kriterij zaustavljanja i drugi mogući argumenti. Funkcija minimize vraća objekt tipa *Result*. U njemu je pohranjen skup nedominiranih rješenja koje je postigao algoritam kao i druge informacije vezane uz proračun.

Pymoo se koristi NumPy bibliotekom za efikasni vektorski i matrični račun, po čemu je NumPy poznat. Vektorizacijom problema i funkcije cilja moguće je paralelizirati proces evaluacije što može ubrzati izvođenje optimizacije.

Primjer zadavanja optimizacije u Pymoo biblioteci na nekom dvociljnom (f_1 i f_2) analitičkom problemu s dva ograničenja (g_1 i g_2) i granicama [-2, 2] nad objema varijablama prikazan je u nastavku [Tablica 4.].

Tablica 4. Primjer definiranja vlastitog problema u Pymoo

```
import autograd.numpy as anp
from pymoo.model.problem import Problem

class MyProblem(Problem):
     def __init__(self):
          super().__init__(n_var = 2, n_obj = 2, n_constr = 2,
                           xl = anp.array([-2, -2]),
                           xu = anp.array([2, 2]))

     def _evaluate(self, x, out, *args, **kwargs):

          #IZRAČUN FUNKCIJA CILJA
          f1  =  x[:, 0]**2 + x[:, 1]**2
          f2 = ( x[:, 0] - 1 )**2 + x[:, 1]**2

          #IZRAČUN OGRANIČENJA
          g1 = 2 * ( x[:,  0] - 0.1 ) * ( x[:, 0] - 0.9 ) / 0.18
          g2 = - 20 * ( x[:, 0] - 0.4 ) * ( x[:, 0] - 0.6 ) / 4.8

          #SPREMANJE U DICTIONARY OUT
          out["F"] = anp.column_stack( [f1, f2] )
          out["G"] = anp.column_stack( [g1, g2] )
```

3.2.6. Mjere kvalitete

U Pymoo biblioteci dane su ove mjere kvalitete: generacijska udaljenost i inverzna generacijska udaljenost, proširena generacijska udaljenost i proširena inverzna generacijska udaljenost i hipervolumen[14]. Hipervolumen se smatra najkvalitetnijim pokazateljem od spomenutih.

Svaku mjeru kvalitete predstavlja objekt. Prilikom instanciranja objekata potrebno je poslati referentu frontu u obliku *numpy* matrice (dvodimenzionalno polje) s m redaka i n stupaca gdje je m broj rješenja, a n broj ciljeva. Hipervolumenu je potrebno umjesto fronte proslijediti referentnu točku u obliku jednodimenzionalnog *numpy* vektora. Pozivanjem metode *do()* nad tim objetktima uz argument koji sadržava dvodimenzionalno polje Pareto skupa kojeg evaluiramo, dobiva se vrijednost mjere kvalitete.

[14] Na engleskom jeziku su redom: *generational distance* (GD)*, inverted generational distance* (IGD)*, generational distance* + (GD+)*, inverted generational distance* + (IGD+)*, hypervolume*.

3.3. jMetalPy

jMetalPy okvir nastao je po uzoru na jMetal projekt iz 2006. g. u Javi koji je jedan od najpriznatijih programskih okvira za optimizaciju – s osobitim naglaskom na višeciljnu optimizaciju stohastičkim algoritmima. jMetal se kontinuirano razvijao i u 2015. g. objavljena je temeljita rekonstrukcija. Izvorni kodovi za jMetal, i njegovu Python inačicu jMetalPy, javno su dostupni na GitHub-u pod MIT licencom.

jMetalPy je razvijen u Pythonu zbog mnogih njegovih prednosti koje su već spomenute i činjenice da Python postaje sve istaknutiji jezik sa većim brojem korisnika. jMetalPy nastoji pružiti i one funkcionalnosti koje u izvornom Java projektu nisu tako kvalitetno pokrivene. Osobit je naglasak na pružanju alata za analizu rezultata algoritama, interaktivnu i istovremenu (*real-time*) vizualizaciju, metode donošenja odluke i rješavanje dinamičkih problema i problema s ograničenjima. Širok raspon interesantnih biblioteka pruža odlično okruženje za brz razvoj - biblioteke za numeričke i znanstvene izračune, analizu podataka, strojno učenje, vizualizaciju i paralelno i distribuirano rješavanje zadataka[15] – sve su one korištene u razvoju jMetalPy-a.

Autori navode kako Python predstavlja i okretnije okruženje za razvoj novih algoritama za višeciljnu optimizaciju [5]. Stoga, jMetalPy nudi i širok skup statističkih testova značajnosti i povezanih alata koji su temelj usporedbe izvođenja stohastičkih višeciljnih algoritama.

3.3.1. Arhitektura

Arhitektura jMetalPy je objektno-orijentirana pa su svi entiteti podijeljeni u određenu strukturu klasa. Takav pristup, kao što je poznato, pruža fleksibilnost i proširljivost. Temeljna funkcionalnost ostvarena putem apstraktnih klasa je da: klasa *Algorithm* rješava *Problem* korištenjem skupa *Operatora* koji vrše operacije nad skupom *Solution* objekata. Tih četiri temeljnih klasa definiranju svaki načina izračuna optimizacijskog problema u jMetalPy-u.

Dodatno je arhitektura pojedinih klasa i njezinih podklasa objašnjenja u sljedećim poglavljima, a detaljnije o jMetalPy-u je moguće pronaći u [5].

3.3.2. Algoritmi

jMetalPy nudi niz suvremenih algoritama za rješavanje višeciljnih problema poput: NSGA-II, NSGA-III, GDE3, SMPSO, OMOPSO, MOCell, IBEA, SPEA2, HypE MOEA/DDE,

[15] Scipy, NumPy, Pandas, Matplotlib, Plotly, Dask, PySpark.

MOEA/D-DRA. Algoritmi su podijeljeni u dvije skupine: većina su evolucijski dok su SMPSO i OMOPSO algoritmi rojeva. U nastavku podnaslova 3.3.2 dan je kratak opis objektno-orijentirane strukture na kojoj je zasnova jMetalPy.

Temeljna klasa od koje sve ostale klase algoritama nasljeđuje je *Algorithm*. Ona je definirana kao apstraktna klasa čija u glavna metoda *run()* ona koja utjelovlje tipično ponašanje u obliku slijeda operacija spomenutih dviju stohastičkih skupina algoritama. Uz to, klasa *Algorithm* sadrži važnu listu rješenja. Lista rješenja kod evolucijskih algoritama predstavlja populaciju, a kod druge skupine - roj.

Svi evolucijski algoritmi nasljeđuju od klase *EvolutionaryAlgorithm*. Genetski algoritmi podskupina su evolucijskih, stoga klasa *GeneticAlgorithm* nasljeđuje od klase *EvolutionaryAlgorithm*.

Klasa evolucijskog algoritma definirana je kao apstraktna klasa, kao stohastički algoritam koja u metodi *step()* primjenjuje slijed operacija selekcije, reprodukcije i zamjene [5]. Metode selekcije, reprodukcije i zamjene zapravo su definirane kao apstraktne.Klasa genetskog algoritma nasljeđuje klasu evolucijskog algoritma i podrazumijeva da se metoda reprodukcije sastoji od dva operatora: operatora križanja i mutacije.

Prilikom inicijalizacije objekta algoritma konstruktoru je potrebno proslijediti objekt problema, sve očekivane objekte operatora, kriterij završetka i neke druge argumene poput veličine populacije i broja potomaka u novoj generaciji.

Metoda *run()* zapravo pokreće optimizaciju.

3.3.3. Operatori

Klase operatora *Mutation, Crossover* i *Selection* imaju metodu *execute()*. Operatori mutacije uzimaju jedan objekt rješenja kao argument, djeluju nad njim i vraćaju izmijenjeni objekt rješenja. Operatori križanja uzimaju listu rješenja (roditelje) i proizvedu drugu listu rješenja (potomke). Operatori odabira obično uzimaju listu rješenja i vraća jedno rješenje ili podlistu te liste rješenja.

Klase operatora dani su u sljedećoj tablici [Tablica 5.]. Neki od operatora djeluju nad cjelobrojnim problemima, neki na realnim, neki na permutacijskim, a neki na binarnim. Nije svaki operator direktno primjenjiv na svaki tip problema.

Tablica 5. Operatori u jMetalPy

Selekcija	
BestSolutionSelection	RandomSolutionSelection
BinaryTournament2Selection	RankingAndCrowdingDistanceSelection
BinaryTournamentSelection	RankingAndFitnessSelection
NaryRandomSolutionSelection	RouletteWheelSelection
Mutacija	Križanje
BitFlipMutation	CXCrossover
IntegerPolynomialMutation	DifferentialEvolutionCrossover
NonUniformMutation	NullCrossover
NullMutation	PMXCrossover
PermutationSwapMutation	SBXCrossover
PolynomialMutation	SPXCrossover
ScrambleMutation	
SimpleRandomMutation	
UniformMutation	

Teško je reći nudi li Pymoo ili jMetalPy više različitih operatora s obzirom da se generalna arhitektura razlikuje, pa Pymoo ima verzije svojih operatora za cjelobrojne, binarne ili realne varijable unutar jednog operatora. .

3.3.4. Problem

jMetalPy nudi i nekoliko skupina standardnih testnih problema poput: ZDT, DTLZ, WFG i LZ09.

Problem je klasa koja kreira i evaluira rješenja. Evaluacija rješenja zbiva se u *evaluate()* metodi. Problem opisuju tri važna atributa: broj projektnih varijabli, broj ciljeva i broj ograničenja. Slično je u Pymoo. S obzirom na vrstu projektnih varijabli defiirani su cjelobrojni, realni i binarni problemi. U [Tablica 6.] dan je isječak koda koji ilustrira definiranje vlastitog problema.

Tablica 6. Prikaz definiranja vlastitog problema u jMetalPy

```
class MyProblem(FloatProblem):

    def __init__(self, n_var, n_objs, n_cons, xl, xu):

        self.number_of_variables = n_var
        self.number_of_objectives = n_objs
        self.number_of_constraints = n_cons
        self.lower_bound = list(xl)
```

```
            self.upper_bound = list(xu)

    def evaluate(self, solution):

        x = solution.variables

        f1 = 2.0 + (x[0] - 2.0) * (x[0] - 2.0) + (x[1] -
1.0) * (x[1] - 1.0)
        f2 = 9.0 * x[0] - (x[1] - 1.0) * (x[1] - 1.0)

        c1 = 1.0 - (x[0] * x[0] + x[1] * x[1]) / 225.0
        c2 = 3.0 * x[1] - x[0]) / 10.0 - 1.0

        f = [f1, f2]
        c = [c1, c2]

        solution.objectives = objectives
        solution.constraints = constraints

        return solution
```

3.3.5. Mjere kvalitete

Biblioteka jMetalPy ima širok „arsenal“ statistički testova značajnosti i povezanih alata čija je svrha usporedba izvođenja višeciljnih algoritama – najviše stohastičkih. Za usporedbe kvalitete Pareto fronte dostupni su sljedeće mjere kvalitete: generacijska udaljenost GD, inverzna generacijska udaljenost IGD, hipervolumen HV i aditivini epsilon pokazatelj I_ε. Za GD, IGD i I_ε potrebno je pripremiti referentne Pareto fronte u obliku *numpy* polja. Za HV referentnu točku je dovoljno uobličiti u običnu Python listu. Za sve pokazatelje potrebno je dobivenu Pareto frontu također proslijediti u obliku *numpy* polja.

4. OPTIMIZACIJSKI PROBLEM - DIMENZIONIRANJE OKVIRNOG REBRA JEDNOSTAVNE BRODSKE KONSTRUKCIJE

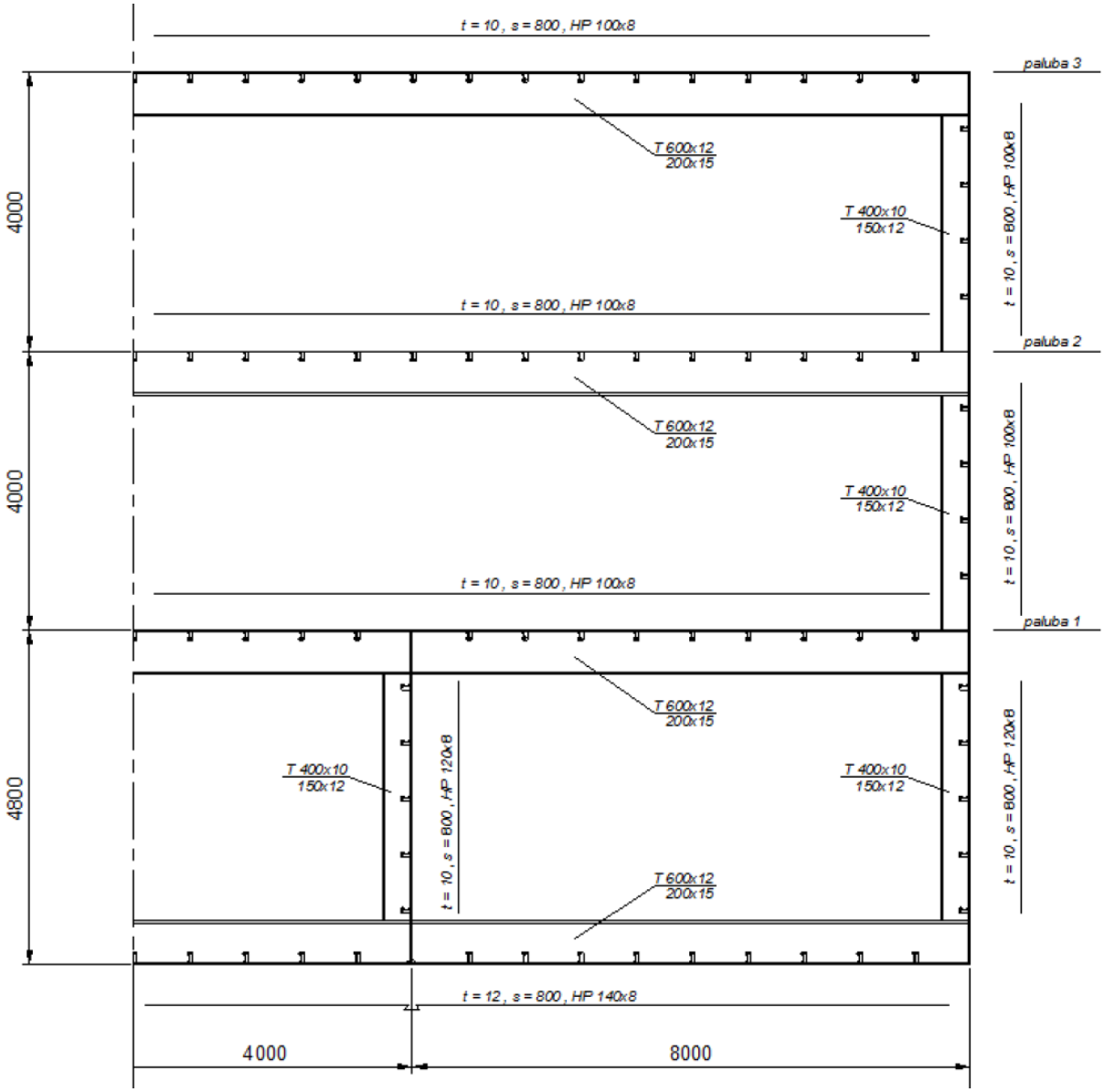
Usporedba u ovom radu vrši se na rješenjima optimizacije provedene na problemu jednostavne brodske konstrukcije poprečnog okvira koja će biti pokazana u 4.1. Metoda pomaka bit će ukratko predstavljena u poglavlju 0. U ovom poglavlju bit će objašnjenja programska struktura i glavni dijelovi tog modela.

4.1. Opis modela

Konkretni model na kojem će se vršiti optimizacija bit će jednostavna konstrukcija pontona sa tri palube i tri tanka pri dnu, točnije njegov poprečni okvir. Geometrija modela okvira koji daje poprečnu čvrstoću pontonu zajedno sa dimenzijama raspona i poprečnih presjeka nosača prikazana je na [Slika 5.]. Okvir je simetričan pa je prikazana samo polovica. Da bi kreirali analitički model ponton koji će se proračunavati metodom pomaka potrebno je numerirati čvorove i grede modela [Slika 7.].Čvorovi su točke modela okvira u kojem se sastaju dvije ili više greda i na slici su označeni brojevima, a grede ih povezuju i označene su brojevima u kružićima [Slika 7.]. Okvir pontona je dakle sastavljen od 16 greda, a one se spajaju u 12 čvorova.

Sljedeći korak je dodijeliti proračunsko opterećenje pojedinim nosačima okvira na nekakav smislen način. Usvojeno projektno opterećenje koje djeluje na okvir predstavlja kombinaciju dva slučaja opterećenja, i to takvu da se za svaku gredu odabire ono opterećenje koje izaziva veće naprezanje od ta dva slučaja. Za realnu analizu konstrukcije potrebno je nezavisno provjeriti zadovoljavanje ograničenja u svakom od slučajeva opterećenja, no korišteno pojednostavljenje je zadovoljavajuće s obzirom na namjenu modela u ovom slučaju. Na najdonjoj palubi, označenoj kao paluba 1, djeluje tlak p_{deck1} = 0,13 MPa. Paluba 1 sastavljena je od tri nosača: 4, 7 i 10. Na palubi 2 djeluje tlak u iznosu p_{deck2} = 0,02 MPa, a na palubi 3 p_{deck3} = 0,016 MPa. Paluba 2 nosač je 15, a paluba 3 je 16. S obirom da se donji dio pontona nalazi ispod površine vode, on je opterećen hidrostatskim tlakom tekućine izvana. Površina vode nalazi se točno u razini čvora 3. Bokovi pontona, nosači 3 i 12, opterećeni su linearnim raspodijeljenim tlakom koji raste od 0 do iznosa p_{bottom}. Iznos tlaka p_{bottom} je jednak hidrostatskom tlaku na dubini od 4800 mm, koliko je vanjska oplata dna ispod površine vode. Dno, kojeg čine nosači 5, 8 i 11, opterećeni su konstantnim kontinuiranim opterećenjem p_{bottom}. Tank C ispunjen je tekućinom, koja ima istu gustoću kao voda, do vrha, pa je stoga opterećenje pregrada tanka na nosačima 6 i 9 jednako kao na vanjskim bokovima, tj. nosačima 12 i 3, tim

redoslijedom. Pritom je zanemarena debljina rebrenica i dna. Gustoća vode neka je ρ = 1025,9 kg·m^{-3}. Izloženi podaci dani su pregledno u [Tablica 7.].



Slika 5.. Poprečni presjek modela okvira (2D) [1]

Sva opterećenja od tlaka potrebno je preračunati i izraziti u obliku opterećenja raspodijeljenog po duljini nosača (u jedinicama N·mm^{-1}). Pritom je važno imati na umu da je okvir koji se razmatra izdvojen je iz konstrukcije pontona koja je monotona tj. građena od ponavljajućih segmenata [Slika 6]. Tako je razmak između dva okvira jednak 2200 mm, a oplata koja ih spaja uzima se u proračun kao sunosiva širina. To znači da oplata također preuzima opterećenje, a s oplatom spojeni T nosači tvore I nosač [Slika 5.]. Opterećenje se preračunava pomoću sljedeće formule:

$$q = p \cdot w \ , \tag{5}$$

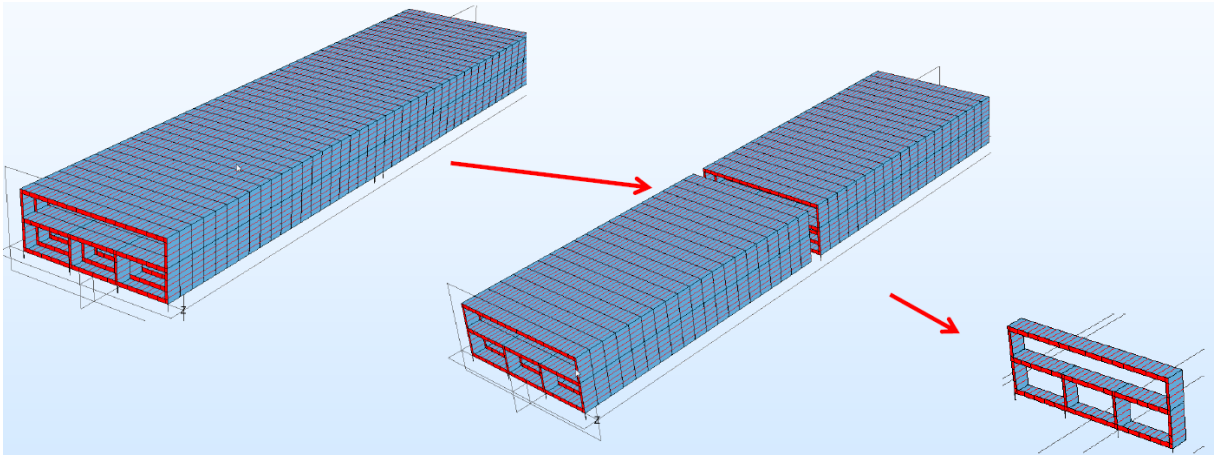
gdje je:

q - kontinurano opterećenje, $N \cdot mm^{-1}$,

p - tlak koji djeluje na oplatu, $N \cdot mm^{-2}$,

w - razmak između okvira, mm.

Za definiranje modela pontona putem tekstualne datoteke potrebno je odrediti položaj koordinatnog sustava. Za ishodište koordinatnog sustava OXY uzeta je točka na simetriji okvira pontona, na polovici nosača 8. Koordinatna os X gleda prema desno, a koordinatna os Y gleda prema gore. U odnosu na taj koordinatni sustav potrebno je definirati sve čvorove okvira. Grede se definiraju s dva čvora i dodjeljivanjem presjeka i materijala objedinjenih u tzv. svojstvo (*property*).

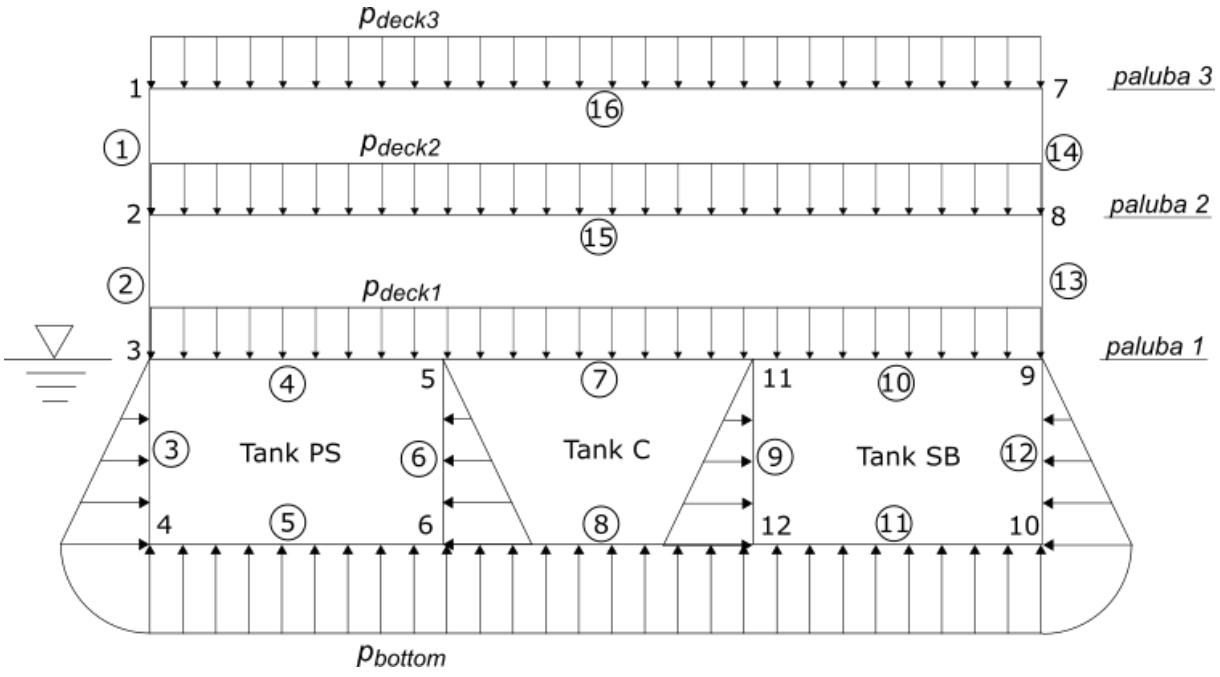


Slika 6. Monotona brodska konstrukcija i izdvajanje okvira

Tablica 7. Neke veličine modela okvira jednostavne brodske konstrukcije

Razmak između okvira, w	2200 mm
Gaz, T	4800 mm
Gustoća morske vode, ρ	$1.0259 \cdot 10^{-6}$ kg·mm^{-3}
Gravitacijska konstanta, g	9806,65 mm·s^{-2}
Tlak na dnu, p_{bottom}	0,048 MPa
Tlak na palubi 1, p_{deck1}	0,13 MPa
Tlak na palubi 2, p_{deck2}	0,02 MPa
Tlak na palubi 3, p_{deck3}	0,016 MPa

Konačno, sada je moguće iz zadanih podataka kreirati ulaznu datoteku za postavljanje modela za proračun okvira koji se koristi i u optimizaciji. U ulaznoj datoteci potrebno je specificirati položaje čvorova, materijale, presjeke s dimenzijama, svojstva, grede i opterećenja kako bi model pontona bio potpun. Kreiranje ulazne datoteke pojašnjeno je u poglavlju 4.4.



Slika 7. Prikaz numeriranja i opterećenja modela okvira jednostavne brodske konstrukcije

4.2. Metoda pomaka

Teoretska podloga analitičkog proračun ravninskih okvira je metoda pomaka koja se koristi u brodogradnji za dimenzioniranje poprečnih okvira. Ovdje će se metoda izložiti skraćeno, kako bi bio jasan postupak implementiran u program. Za detalje potrebno je proučiti dostupnu literaturu. Opis metode sastavljen je prema [9].

Spomenuta metoda može rješavati uz određene pretpostavke složene ravninske okvire grednih nosača s nekoliko zatvorenih petlji. Takvi okviri statički su neodređeni i unutarnje sile ne mogu se odrediti samo na osnovu uvjeta ravnoteže, već se moraju uzeti u obzir i uvjeti kompatibilnosti deformacija elemenata sustava.

Pretpostavke koje metoda uvodi su:

- svaki pojedinačni okvir u monotonoj brodskoj konstrukciji preuzima jednako opterećenje,
- djelovanja poprečnih sila i deformacija razmatraju se razdvojeno od uzdužnih,
- često se zanemaruju male veličine drugog reda, kao što linearni pomaci čvorova uslijed progibanja nosača ili linearni pomaci uslijed djelovanja aksijalnih sila,
- moment upetosti pretpostavlja se u obliku linearne jednadžbe (6),
- ne definira eksplicitno rubne uvjete, tj. način oslanjanja okvira na okolinu.

Svaki okvir zamišlja se kao sastavljen od više greda na dva oslonca. One predstavljaju osnovne, najjednostavnije dijelove statički neodređenih okvira. Grede u okviru sastaju se u čvorovima – u čvoru se sastaju najmanje dvije grede. Da bi se izračunalo naprezanje od savijanja pojedinih greda okvira, potrebno je naći raspodjelu unutarnjeg momenta savijanja. Ta raspodjela uvjetovana je raspodjelom opterećenja po gredi (u praksi uglavnom u obliku kontinuiranog poprečnog opterećenja) i načinom učvršćenja krajeva grede tj. rubnim uvjetima.

Problem neodređenosti upravo se manifestira u određivanju načina učvršćenja krajeva greda jer krajevi greda nisu niti idealno upeti (uklješteni) niti su zglobno oslonjeni, već su krajevi greda učvrćeni na druge nosače ili dijelove brodske konstrukcije koji imaju svoju elastičnu krutost.

Samo su posebni slučajevi kada spoj promatrane grede možemo modelirati kao uklještenje ili zglobni oslonac. To je važno pri ručnom proračunu, no ukratko treba pojasniti sučajeve idealiziranja radi shvaćanja metode. Potpuno upetim uporištem moguće je idealizirati neke spojeve kod simetrično opterećenih i geometrijski simetričnih konstrukcija i spojeve u kojima

se promatrani nosač spaja na dio konstrukcije ili nosač relativno značajno veće krutosti (detaljnije se može pronaći u [1], zajedno s primjerima).

Zglobnim uporištem moguće je smatrati ono u kojem se promatrani nosač spaja na nosač ili dio konstrukcije relativno značajno manje krutosti.

No, generalno nosači u sustavu okvira su istog reda krutosti pa su grede u uporištima elastično upete. Korištenjem metode pomaka moguće je odrediti momente upetosti svih greda u svakom čvoru čija veličina odgovara razini upetosti, a na temelju toga moguće je odrediti raspodjelu unutarnjeg momenta za svaki nosač u gredi.

Rješavanje problema započinje se tako da se za svaki nosač postave dvije jednadžbe momenta upetosti, svaka za jedan kraj nosača (6). Jednadžba se ne postavlja u posebnom slučaju - ako je čvor potpuno upet, tj. kada je pod djelovanjem opterećenja kut zakreta čvora jednak nuli.

$$M_{12} = k_{12}(2\varphi_1 + \varphi_2) + m_{12}$$
$$M_{21} = k_{21}(2\varphi_2 + \varphi_1) + m_{21} \qquad \textbf{(6)}$$

gdje su:

m_{12} - moment u točki 1 potpuno upete grede oslonjene između čvorova 1 i 2 od vanjskog opterećenja,

m_{21} - moment u točki 2 potpuno upete grede oslonjene između čvorova 1 i 2 od vanjskog opterećenja,

k_{12} - koeficijent krutosti nosača na savijanje,

$k_{12} = k_{21}$,

φ_i - kut zakreta u čvoru i,

Koeficijent krutosti k_{ij} računa se puem sljedećeg izraza:

$$k_{ij} = \frac{2 \cdot E \cdot I_{ij}}{l_{ij}} \qquad \textbf{(7)}$$

gdje su:

I_{ij} - moment tromosti presjeka nosača između čvorova „i“ i „j“ (konstantan po cijelom rasponu),

l_{ij} - duljina nosača između čvorova „i“ i „j“,

E - Youngov modul elastičnosti.

Zatim se primjenjuje statički uvjet ravnoteže na čvorove okvira.. Korištenjem jednadžbi oblika (6) slijedi se sljedeći princip statike: suma momenata koji djeluju na čvor konstrukcije mora biti jednaka nuli. Drugim riječima, suma momenata upetosti pojedinih nosača (na onom kraju koji je dio čvora) je jednaka nuli (8). Indeks i označava i-ti čvor za koji se postavlja uvjet ravnoteže, a indeks j označava druge čvorove greda koje se sastaju u i-tom čvoru.

$$\sum_{j=1}^{n} M_{ij} = 0: \quad \textbf{(8)}$$

Primjerice, ako se u čvoru 2 sastaju grede 1-2, 2-3, 2-4, tada bi postavljeni uvjet ravnoteže glasio:

$$M_{21} + M_{23} + M_{24} = 0: \quad \textbf{(9)}$$

Pritom izraz greda 1-2 označava gredu koja spaja čvor 1 i 2. Redoslijed indeksa je važan jer momenti M_{12} i M_{21} općenito nisu istog iznosa.

Tim korakom supstitucije momentata upetosti s njihovim jednadžbama jasno je da se dobije onoliko jednadbži koliko je čvorova s nepoznatim kutom zakreta. Te jednadžbe su linearne i tvore sustav. Nepoznate veličine su kutevi zakreta i otuda ime metodi pomaka – nepoznanice sustava su pomaci. Njegovim rješavanjem dobiju se iznosi kuteva zakreta svakog čvora.

Za momente upetosti u čvorovima potpuno upetih greda najjednostavnije je poslužiti se tabličnim podacima kakvi su primjerice dostupni u [8].

Iznosi momenata upetosti mogu se dobiti uvrštenjem izračunatih vrijednosti kuteva zakreta φ_i i tabličnih momenata upetosti m_{ij} potpuno upetih greda u jednadžbe (6). Kada su određeni svi momenti upetosti M_{ij}, moguće je odrediti funkcije raspodjele unutarnjih momenata savijanja.

S obzirom da su za dobar dio slučajeva opterećenja potpuno upete grede unaprijed poznate raspodjele unutanjih veličina (kao što je prikazano na () za dva slučaja opterećenja), moguće je po principu superpozicije za linearne probleme[16], poznatim raspodjelama nadodati raspodjelu od vanjskih momenata i sila.

[16] Kod linearnih problema pomaci linearno ovise o silama, i konstrukcija se nakon rasterećenja vraća u prvobitni oblik. Osnovni uvjet je da je konstrukcija izrađena od linearnog-elastičnog materijala, Osim toga, pomaci moraju biti "mali" i konstrukcija ne smije mijenjati krutost tijekom deformiranja.

Ta „vanjska“ raspodjela kreira se na sljedeći način. Zamišlja se da na pojedinu gredu u sustavu okvira djeluju momenti upetosti M_{ij} i M_{ji}. S obzirom da općenito nisu jednaki, da bi greda bila u ravnoteži, u uporištima se javljaju i poprečne reaktivne sile. Te sile izazivaju linearnu raspodjelu unutarnjeg momenta savijanja. Stoga, raspodjeli unutarnjeg momenta savijanja superponira se linearna raspodjela kojoj su krajnje vrijednosti M_{ij} i M_{ji}. Te dvije raspodjele zajedno daju raspodjelu momenta savijanja po nosaču u sustavu okvira.

Završni je korak pronaći maksimalni moment savijanja u nosaču. Za grede s konstantnim poprečnim presjekom, na tom mjestu naprezanje u nosaču je najveće pa je ono relevantno za dimenzioniranje. Ono se izračunava jednostavno kao omjer momenta savijanja i momenta otpora).

$$\sigma_{ij} = \frac{\max\left(abs\big(M(x)\big)\right)}{W_{ij}} : \qquad \textbf{(10)}$$

gdje su:

σ - najveće naprezanje u gredi “i”-“j”,

$M(x)$ - raspodjela momenta savijanja po gredi,

W_{ij} - moment otpora grede “i”-“j”.

Skraćeno, koraci provedbe proračuna su:

1. Za svaki nosač okvira postaviti jednadžbe (6)
2. Za svaki čvor postaviti jednadžbu (8)
3. Riješiti linearni sustav jednadžbi po kutevima zakreta
4. Odrediti momente upetosti i raspodjele momenata savijanja kod potpuno upetih greda (najčešće iz tabličnih rješenja)
5. Odrediti momente upetosti M_{ij} uvrštavanjem kuteva zakreta φ_{ij} i momenata m_{ij}.
6. Odrediti superponirane momente dijagrame $M(x)$.
7. Izračunati naprezanje na kritičnim mjestima nosača – pozicijama maksimalnog momenta

4.3. Implementacija modela za proračun okvira metodom pomaka

Prikazani analitički proračun metodom pomaka programski je implementiran u datoteku *Frame_analysis.py*. Ona sadržava klase koje modeliraju entitete nužne za definiranje okvira. To su:

- Klasa *Node* za čvorove,
- Klasa *Section* za poprečne presjeke s potklasama za specifične oblike,
- Klasa *Material* za materijale,
- Klasa *Property* koja okuplja materijal i presjek u svojstvo koje se dodjeljuje gredi,
- Klasa *Beam* za grede tj. pojedine nosače okvira,
- Klasa *Structure* koja predstavlja okvir.

Uz njih, tu je i klasa za definiranje ulazne datoteke. Objekti modela se prvenstveno kreiraju putem jedne ulazne tekstualne datoteke. Njezina struktura je relativno strogo definirana i pojašnjena je u poglavlju 4.4. Na [Slika 8.] je dan primjer datoteke kakva je korištena za kreiranje modela nad kojim je vršena optimizacija u ovom radu.

```
1  # OKVIR iz Metoda pomaka-programski zadatak.pdf Koordinatni sus
2  # Primjer excella radi usporedbe rezultata - number of objects:
3  12,1,16,16,16,12
4  # Node, ID, x, y
5      1, -12000.0, 12800.0
6      2, -12000.0, 8800.0
7      3, -12000.0, 4800.0
8      4, -12000.0, 0.0
9      5, -4000.0, 4800.0
10     6, -4000.0, 0.0
11     7, 12000.0, 12800.0
12     8, 12000.0, 8800.0
13     9, 12000.0, 4800.0
14     10, 12000.0, 0.0
15     11, 4000.0, 4800.0
16     12, 4000.0, 0.0
17 # Material, ID, name, E, nu, density [kgm-3], sigmadop
18     1, steel, 204000 , 0.3, 8000, 160
19 # Section, ID, type, name, param, I_Beam: h,t,w1,t1,w2,t2
20     1, I_Beam, "T1000x12 400x18", 1200, 12, 2250, 10, 600, 22
21     2, I_Beam, "T1000x12 400x18", 1200, 12, 2250, 10, 600, 22
```

Slika 8. Izgled ulazne datoteke

Glavna klasa je *Structure*. Ona sadržava liste svih tipova objekata: čvorova, presjeka, materijala, greda, svojstava. Njezine dvije glavne metode su *global_equation()* i *calculate_all()*. Proračun cijele konstrukcije odvija se na sljedeći način: metoda *calculate_all()* poziva metodu *global_equation()* koja postavlja linearni sustav jednadžbi s kutevima zakreta kao nepoznanicama, rješava ga i vraća ih kao izlaz u obliku *NumPy* vektora. Metoda

calculate_all() pomoću tih kuteva zakreta izračunava momente upetosti pomoću jednadžbe (6) za sve grede i poziva izračun naprezanja.

Pri svakoj optimizaciji okvira kreira se jedan objekt tipa *Structure* koji opstaje tijekom cijele optimizacije od početka do samog kraja. On sadržava sve podatke o trenutnom stanju modela optimizacijskog problema tj. o okviru.

4.4. Izgled ulazne datoteke

Analitički model okvira prikazan u poglavlju 4.1 zadaje se putem ulazne datoteke. Ulazna datoteka dakle služi za kreiranje objekata i modela okvira putem modula *Frame_analysis.py.* Ulazna datoteka tekstualna je datoteka (.txt) i ima jednostavnu strukturu, no s obzirom da je za tipični okvir u njoj dosta podataka, potrebno je pažljivo objasniti njezinu strukturu.

Na početku datoteke moguće je napisati koliko god redova teksta iza znakova # koji označavaju komentare. Nakon početnih komentara sljedi redak koji treba imati nužno 6 brojeva odvojenih zarezom. Ti brojevi predstavljaju količinu pojedinih objekata koji će se kreirati. Potrebno je da taj niz slijedi točan redoslijed: broj čvorova, broj materijala, broj presjeka, broj svojstava, broj greda i broj opterećenja. S obzirom da ne može postojati okvir bez ijednog čvora ili ijedne grede, ili ijednog materijala, nužno je da početni redak sadrži tih 6 brojeva kao informaciju metodi za učitavanje modela iz tekstualne datoteke.

Glavna metoda klase *Input_file* nazvana *create_model()* vodi proces čitanja datoteke i poziva cijelu skupinu drugih metoda koje vrše kreiranje objekata. Na kraju čitanja datoteke sve rezultira u jednom objektu tipa *Structure* koji predstavlja okvir, a koji sadržava liste svih ostalih kreiranih objekata.

4.4.1. Specificiranje objekata putem datoteke

Kao što je rečeno, u datoteci pri vrhu je potrebno jednim retkom sa šest brojeva definirati broj svakog tipa objekta, a onda ih navesti jedne za drugim poštivajući ovaj redoslijed: *Node, Material, Section, Property, Beam, Load*. Za kreiranje pojedinog objekta potreban je redak u kojem su zarezima odvojene navedene vrijednosti koje očekuje svaki pojedini objekt. Broj bjelina ispred i iza podataka nije važan.

Potrebno je pružiti ove podatke za kreiranje objekata tipa:

- *Node*: identifikacijski broj (ID) čvora, x i y koordinatu.
- *Material*: ID, ime, modul elastičnosti, Poissonov koeficijent, gustoću, dopušteno naprezanje

- *Section:* ID, podtip, ime, parametre presjeka (važan je redoslijed za značenje pojedinog parametra)
- *Property:* ID, ime, ID materijala i ID presjeka koje objedinjuje
- *Beam:* ID, ID čvora 1, ID čvora 2 i ID svojstva
- „*Load*" (nije klasa)*:* ID opterećenja, tip opterećenja, ID grede na koju djeluje, vrijednost

Različite skupine objekata (npr. objekte *Node* od objekata *Material*) prikladno je razdvojiti s jednom (moguće i više) linijom komentara. Te linije moraju imati znak ljestvi # na prvom mjestu. One mogu poslužiti kao vizualni prikaz redoslijeda parametara, kako bi sama datoteka bila čitljivija.

Za definiranje presjeka predviđeno je šest klasa: *I_beam*, *CircTube*, *CircBar*, *C_beam*, *Rectangle*, *TableSection*. Broj presjeka lako je proširiti dodavanjem novih klasa. Pritom je potrebno naslijediti klasu *Section* i prepisati (*override*) njezine metode *__init__ ()* i *calculate().* U metodi *calculate()* potrebno je izračunati veličine poprečnog presjeka iz liste njegovih parametara i spremiti ih u atribute. Ti atributi su:

- površina poprečnog presjeka – self.A
- moment tromosti presjeka – self.Iy
- moment otpora presjeka – self.Wy

Implementirane klase imaju ovo značenje:

- *I_beam* definira I presjek greda i zahtijeva šest parametara,
- *CircTube* definira presjek okrugle cijevi i zahtijeva dva parametra,
- *CircBar* definira presjek okrugle šipke i zahtijeva jedan parametar,
- *C_beam* definira presjek grede orijentiran kao i u obliku slova C i zahtijeva šest parametara[17],
- *Rectangle* definira presjek pravokutne cijevi i zahtijeva četiri parametra.
- *TableSection* ne definira nikakav poseban oblik presjeka, već izravno učitava veličine poput površine presjeka, momenta tromosti i momenta otpora. Kod svih ostalih presjeka, parametri su dimenzije presjeka preko kojih se izračunavaju veličine poput površine ili momenta otpora. *TableSection* presjeci ne mogu se optimirati.

[17] Kod presjeka nesimetričnih oko vertikalne osi (ukoliko se djelovanje opterećenja koje savija gredu zbiva oko te osi) dolazi do kosog savijanja. Trenutačno nije modelirano takvo ponašanje.

Parametri presjeka moraju u tekstualnoj datoteci biti napisani određenim redoslijedom. Redoslijed parametra presjeka za pojedine presjeke je ovaj:

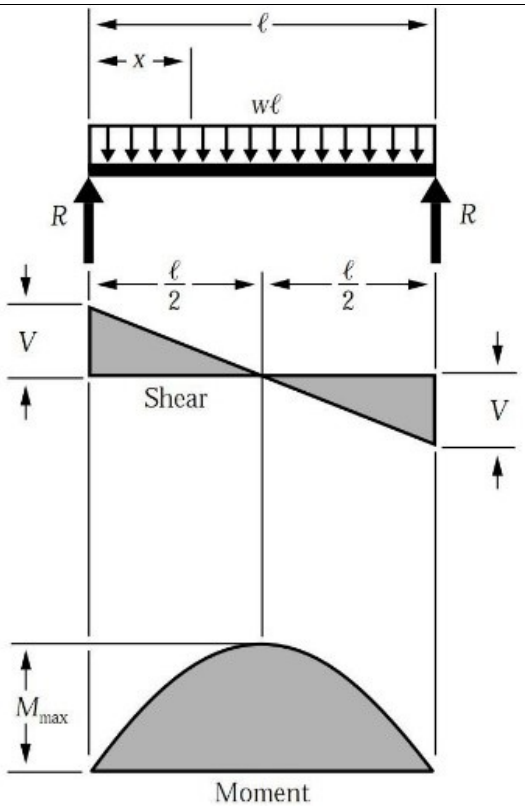
- *I_beam*: *h, t,* w_1, t_1, w_2, t_2
- *CircTube*: d_u, d_v
- *CircBar*: *d*
- *C_beam*: *h*, *w*, *ts*, *tp*
- *Rectangle*: *hv*, *wv, hu, wu*

Pritom su to kod *I_beam* visina (*h*) i debljina rebra (*t*), i širine (w_1 i w_2) i debljine (t_1 i t_2) pojaseva , kod *CircTube* unutarnji (d_u) i vanjski (d_v) promjer okrugle cijevi, kod *CircBar* promjer (*d*) okrugle šipke, kod *C_beam* visina i širina (*h*, *w*) presjeka s debljinom rebra i pojaseva (*ts*, *tp*) i kod *Rectangle* vanjska visina i širina (*hv*, *wv*) te unutarnja visina i širina (*hu, wu*).

Za definiranje opterećenja trenutno je implementirano pet tipova opterećenja:

- „F“ za koncentriranu silu na proizvoljno definiranom mjestu duž grede,
- „q“ za konstantno kontinuirano opterećenje duž cijele grede,
- „qlinl“ za linearno raspodijeljeno kontinuirano opterećenje koje opada od maksimalnog iznosa do nula u smjeru od prvog čvora do drugog,
- „qlinr“ za linearno raspodijeljeno kontinuirano opterećenje koje raste od nula do maksimalnog iznosa u smjeru od prvog čvora do drugog,
- „M“ za koncentirani moment na proizvoljno definiranom mjestu na gredi,

Znakovi navodnika označuju da opterećenja nisu uobličena u klasu. Opterećenja se umjesto toga izravno pri učitavanju pretvaraju u momente upetosti m_{ij} i raspodjelu momenta savijanja kod zglobno oslonjenog nosača. Raspodjela unutarnjeg momenta savijanja dobiva se na sljedeći način. Funkcija koja opisuje raspodjelu momenta savijanja duž zglobno oslonjenog nosača duljine *l* opterećenog kontinuiranim konstantnim opterećenje *w* je parabola 2. reda s maksimum na polovici nosača u iznosu od $M_{max} = wl^2 / 8$ [Slika 9.]. U oba oslonca iznos momenta savijanja jednak je nuli, a javljaju se jednake poprečne reakcije *R* i raspodjela unutarnje poprečne sile je linearna.



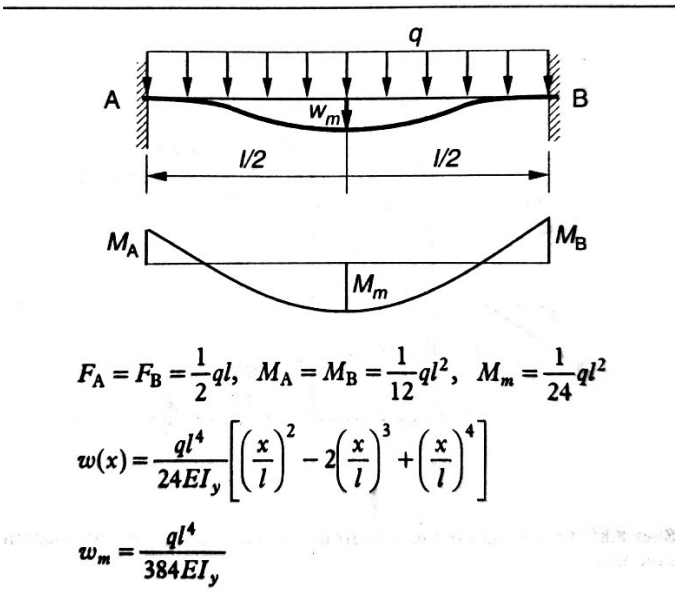
Slika 9. Raspodjela poprečnih sila i momenata savijanja u gredi na dva zglobna oslonca opterećenoj konstanim kontinuiranim opterećenjem [10]

Funkcija raspodjele momenta savijanja kod potpuno upetog nosača duljine *l* opterećenog istim konstantnim opterećenjem ovaj put označenim s *q* također je parabola 2. reda s maksimum na polovici nosača u iznosu $M_m = 1 / 24 \cdot ql^2$. No, kod potpuno upetog nosača postoje reakcije momenta u uporištima, i one iznose $M_A = M_B = 1 / 24 \cdot ql^2$. Raspodjela poprečnih sila identična je kao i kod zglobno oslonjenog nosača. Očito je da je raspodjelu potpuno upetog nosača moguće dobiti jednostavnom "translacijom" funkcije raspodjele zglobno oslonjenog nosača. Ako raspodjeli zglobno oslonjenog nosača cijelom njegovo duljinom superponiramo konstantnu raspodjelu u iznosu momenata $M_A = M_B$, dobit će raspodjela potpuno upetog nosača.

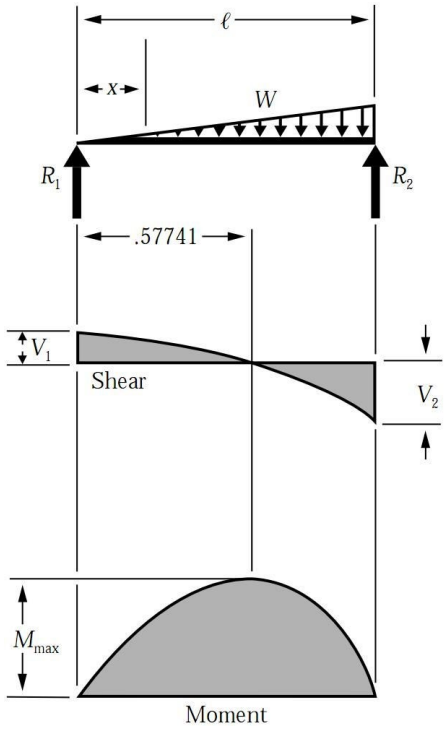
No, općenitiji slučaj predstavlja zglobno oslonjena greda s linearnom raspodjelom opterećenja. Funkcija raspodjele momenta parabola je trećeg reda, a momenti reakcije su jednaki nula. Kod potpuno upete grede, momenti reakcije M_A i M_B su različiti. Može se utvrditi da se raspodjela od upete grede može dobiti zbrajanjem one od zglobno oslonjene i linearne raspodjele u obliku trapeza koju čine momenti M_A i M_B.

Tako da je moguće dobiti momentne dijagrame potpuno upetih nosača i pri različitim momentima upetosti M_A i M_B iz dijagrama zglobno oslonjenih nosača, ako im se doda linearna raspodjela u obliku „trapeza" koju oni tvore. Prošireno, to znači da se mogu dobiti momentni

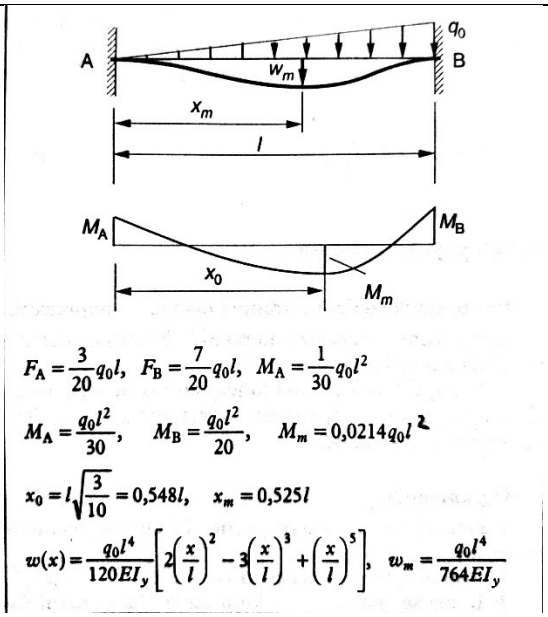
dijagrami elastično upetih nosača superponiranjem dijagrama zglobno oslonjenih nosača i spomenutog „trapeza“.



Slika 10. Potpuno upeti nosač s konstantnim kontinuiranim opterećenjem [8]



Slika 11. Raspodjela poprečnih sila i momenata savijanja u gredi na dva zglobna oslonca opterećenoj linearnim kontinuiranim opterećenjem [10]



Slika 12. Potpuno upeti nosač s linearnim kontinuiranim opterećenjem [8]

Važno je napomenuti da korisnik sam određuje jedinice u kojima unosi veličine. Jedinice je potrebno unositi konzistentno. Primjerice, ako su koordinate čvorova u milimetrima, onda i dimenzije presjeka trebaju biti u milimetrima, a modul elastičnosti u $N \cdot mm^{-2}$ tj. MPa.

4.5. Formulacija optimizacijskog problema

Ako se okvir želi optimizirati po nekom od relevantnih kriterija, tada je potrebno ovaj postupak proračuna mnogo puta ponoviti. U optimizaciji koja će biti prikazana u narednim poglavljima naprezanje u gredama koristi se za definiranje ograničenja. To je samo po sebi jasno s obzirom da je sa stanovišta projektiranja glavni cilj osigurati čvrstoću konstrukcije u eksploataciji. Taj kriterij u projektiranju postaje ograničenje optimizacijskom algoritmu.

I drugi se podaci o konstrukciji mogu koristit za kreiranje ograničenja i funkcija cilja, a cijeli se proračun spomenut u poglavlju 0 ponavlja mnogo puta kako bi se dobio uvid u stanje svih vrijednosti ograničenja i funkcija cilja što je nužno za rad algoritma.

4.5.1. Ciljevi

Općenito, na ovakvu konstrukciju okvira moguće je zadati mnogo ciljeva. Odabrana su dva:

- minimizirati masu konstrukcije,
- sniziti poziciju težišta konstrukcije.

Oba cilja realna su i poželjna. Sniženje mase konstrukcije općenito znači smanjenje proizvodnih i eksploatacijskih troškova, a sniženje pozicije težišta donosi povećanu stabilnost plovila. Ciljevi su normalizirani u odnosu na iznos mase i težišta konstrukcije na način zadane putem tekstualne datoteke. Za konstrukciju zadanu datotekom izračuna se masa w_0 i vertikalni položaj težišta VCG_0. Vrijednosti funkcija cilja $f_1(x)$ i $f_2(x)$ su putem korištenja *RatioGetCallbackConnector* (opisano kasnije u poglavlju) definirane na sljedeći način:

$$f_{i,n} = \frac{f_i(x)}{f_{i,0}} \tag{11}$$

gdje su:

$f_{i,n}$ - normalizirana vrijednost i-te funkcije cilja,

$f_i(x)$ - vrijednost i-te funkcije cilja,

$f_{i,0}$ - početna vrijednost funkcije cilja za dimenzije zadane putem tekstualne datoteke.

Iako će algoritam u početku uzimati uzorke na razne načine iz prostora, na ovaj način je ipak osigurano da su ciljevi približno jednako vrijedni, a razlike u apsolutnim vrijednostima ne utječu znatno na preferirano kretanje algoritma u jednom smjeru.

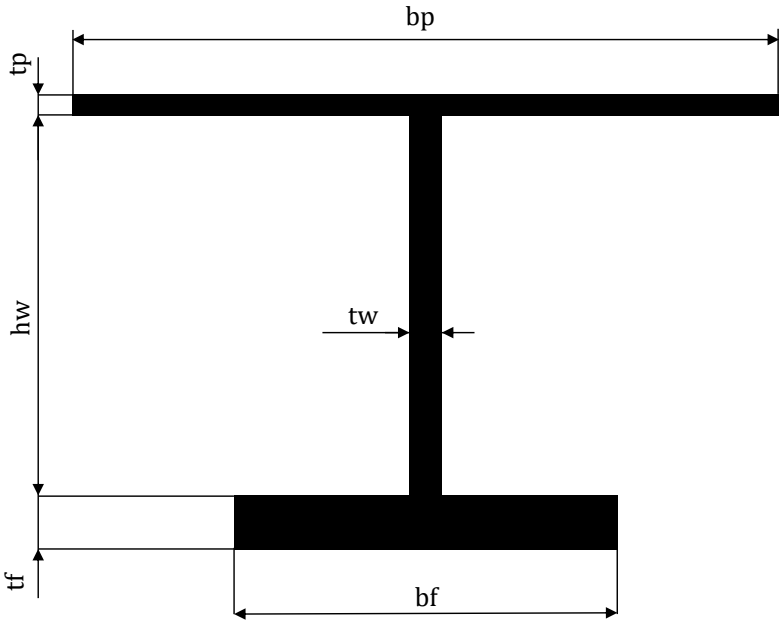
4.5.2. Projektne varijable

U poglavlju 4.1. spomenuto je da su nosači okvira sačinjeni od T profila i oplate. Oni zajedno tvore I profil. Oplati se ne može mijenjati širina jer je određena razmakom okvira – što dolazi kao predeterminirano. Debljina se također ne može mijenjati. Stoga, za projektne varijable problema uzete su četiri dimenzije T profila: *hw, tw, bf* i *tf* [Slika 13.]. Ipak, putem tekstualne datoteke okvir je sastavljen od greda I profila, ali u optimizaciju kao projektne varijable nisu uvršteni treći i četvrti parametar profila – što odgovara upravo dimenzijama oplate, tj. *bp* i *tp* na slici.

Na te projektne varijable postavljene su granice s donje i gornje strane, tj. varijabla se ne smije naći izvan zadanih granica. Zato se dimenzije poprečnog presjeka grede (T profila) na kraju uspješne optimizacije moraju kretati u sljedećim intervalima:

- $500 \leq h_w \leq 1500$,
- $5 \leq t_w \leq 25$,

- $50 \leq b_f \leq 500$,
- $5 \leq t_f \leq 30$.



Slika 13. Poprečni presjek I nosača okvira pontona

4.5.3. Ograničenja

Dodatna su postavljena neka ograničenja u obliku odnosa pojedinih dimenzija. To može predstavljati tehnološko ograničenje kod najčešće tehnologije spajanja u brodogradnji - zavarivanja.

Ograničenja dimenzija postavljena na svaki presjek na sljedeći način:

- $h_w/t_w < 90$,
- $4 < b_f / t_f < 25$,
- $0{,}2 < b_f / h_w < 0{,}5$,
- $1 < t_f / t_w < 3$.

Također, svaka greda podliježe i uvjetu čvrstoće. Maksimalno naprezanje grede treba biti manje od dopuštenog $\sigma < \sigma_{dopušteno} = 160$ MPa.

Ograničenja dimenzija su normalizirana pomoću sljedeće formule:

$$\frac{C-D}{C+D} \geq 0 \qquad \textbf{(12)}$$

gdje:

C - označava "sposobnost nošenja" tj. granicu (*capability*),

D - označava vrijednost "zahtjeva" koji ne smije prijeći granicu (*demand*).

Po konvenciji, izraz (12) se koristi za nejednakosti tipa D ≤ C, i to tako da, prema gore izvedenoj formuli, vrijednosti razlomka tj. lijeve strane nejednadžbe su uvijek u intervalu $[-1, 1]$. Dozvoljeno područje smatra se [0, 1] kada je zaista D ≤ C, a nedozvoljeno područje smatra se $[-1, 0\rangle$. Na primjer, ako je b_f / t_f =10, prema gore definiranim ograničenjima, taj bi slučaj trebao zadovoljavati. Ako se supstituira D = 10, a C = 25, tada je:

$$\frac{C-D}{C+D} = \frac{25-10}{25+10} = \frac{3}{5} \geq 0. \qquad \textbf{(13)}$$

Očito, ova je nejednadžba zadovoljena.

No, definirana je i nejednakost b_f / t_f > 4 koja ne odgovara nejednakosti tipa D < C, već D>C ako zadržimo da je omjer jednak D, a definirana granica C. Jasno je da je u nejednadžbi D>C izgubljen konvencionalni smisao jer zahtjev treba biti veći od „sposobnosti nošenja" da bi nejednadžba bila zadovoljena. Iako je taj smisao izgubljen, i dalje se formula može iskoristi da bi normalizirali nejednadžbe D>C, samo potrebno je izmijeniti nejednadžbu (12) na sljedeći način:

$$\frac{D-C}{C+D} \geq 0 \qquad \textbf{(14)}$$

Posebnost Pymoo biblioteke spomenuta u 3.2.1. jest da se ograničenja smatraju ispunjenim ako je lijeva strana nejednadžbe manja od desne. Stoga je kod njega potrebno obrnuto definirati prikazana dva tipa ograničenja.

Tako se efektivno zamijenjuje definicija podobnog i nepodobnog područja, te se za podobno područje dobiva $[-1,\ 0]$, a za nepodobno područje $[0, 1\rangle$.

5. Biblioteka za usporedbu rješenja višeciljnih optimizacijskih problema s ograničenjima – moobench

U ovom poglavlju bit će po modulima predstavljen programski okvir odnosno biblioteka *moobench* i njezina implementacija i korištenje.

Biblioteka *moobench* je softver otvorenog kôda, koji je napravljen dostupnim na *GitHub*-u koji je izveden s namjerom pojednostavljenja usporedbe rješenja jednociljnih i višeciljnih optimizacijskih problema s ograničenjima koje postižu optimizacijski algoritmi iz različitih biblioteka dostupnih u programskom jeziku Python. U biblioteci je postojala implementacija bibioteke *SciPy.Optimize*, a u okviru ovog rada implementirana su sučelja prema bibliotekama *Pymoo* i *jMetalPy*.

S obzirom da svaka biblioteka ima neke svoje posebnosti za pokretanje optimizacije određenog problema, to za korisnika predstavlja potrebu potankog poznavanja biblioteka. Uz to, postavljanje optimizacije traži pisanje mnogo redova programskog kôda što može biti zamorno i vremenski zahtjevno za imalo složeniji problem.. Naime, najzahtjevniji dio postavljanja optimizacije je ukomponirati svoj složeni problem u optimizaciju. Uz to, često je puta mnogo toga potrebno iznova ponovno napisati samo za drugu biblioteku jer su njihove implementacije gotovo uvijek različite. Biblioteka *moobench* nastoji smanjiti razlike prema korisniku u definiranju problema za različite optimizacijske biblioteke. Tako, korisnik može jednom definirati problem unutar zasebnog modula korištenjem funkcionalnosti *moobench* biblioteke, i pritom se fokusirati jednostavno na definiranje optimizacije i dodjeljivanja čisto optimizacijskih operatora, algoritama, kriterija zaustavljanja, itd.

Drugim riječima, *moobench* biblioteka ima za cilj omogućiti korisniku jednostavne naredbe u korisničkom programu neovisne ili vrlo malo ovisne o načinu implementacije modela problema ili o načinu implementacije pojedinih optimizacijskih biblioteka. Kreiranjem takvog sučelja korištenje optimizacije postaje znatno jednostavnije i brže – što znači da bi se razni korisnici u potrebi češće odlučivali na nju.

5.1. Arhitektura biblioteke

Sučelje je napisano u Pythonu i sastoji se od nekoliko modula (datoteka tipa .py). Modul *optbase.py* temeljni je i najsloženiji modul koji definira opće entitete na višoj razini koji su potrebni svakoj optimizaciji i opće ponašanje optimizacije. Modul *optlib_pymoo_proto.py* spona je između *optbase* modula i biblioteke Pymoo. Za jMetalPy biblioteku postoji modul

optlib_jmetalpy_proto.py. Kod usporedbe rješenja potrebno je provesti mnogo optimizacijskih proračuna, pa je pokretanje i paralelizacija izračuna obuhvaćena u modulu *jobbase.py*.

Korisniku preostaje prilikom postavljanja optimizacija kreirati modul u kojem je sadržan problem i kreirati modul koji će definirati parametre optimizacije kakve nude biblioteke i vršiti provođenje zadataka. Prilikom pisanja problema korisnik se koristi ujednačenim naredbama pretežito iz *optbase.py*. Pri obje radnje, korisnik ne mora učiniti niti jedan uvoz iz biblioteka Pymoo ili jMetalPy što znatno olakšava korištenje.

5.2. Modul optbase.py

Modul *optbase.py* kako je već rečeno sadrži temeljne klase koje definiraju različite elemente optimizacijskog postupka. Pojednostavljeno, ona sadrži sljedeće klase:

- *DesignVariable* koja predstavlja projektne varijable,
- *DesignConstraint* koja predstavlja ograničenja,
- *DesignObjective* koja predstavlja funkcije cilja,
- razni tipovi konektora poput: *NdArrayGetSetConnector, CallbackGetConnector, RatioDesignConnector* koji služe za povezivanje vrijednosti modela problema i elemenata optimizacijskog postupka,
- *AnalysisExecutor* koja predstavlja izvršni dio evaluacije modela,
- *OptimizationProblem* koja predstavlja cjelokupni problem,
- *OptimizationOutput* koja predstavlja izlaz
- *OptimizationSolution* koja predstavlja konačna rješenja koje daje algoritam nad optimizacijom.

Središnja klasa je *OptimizationProblem* koja sadrži sve informacije potrebne za pokretanje optimizacije. Ona predstavlja optimizacijski problem sa svim svojim obilježjima.

Primjer djelovanja konektora prikazan je na [Slika 15.] (poglavlje 5.6) uz opis u tekstu iznad.

5.3. Modul *optlib_pymoo_proto*

Unutar modula optlib_pymoo_proto sadržano je sučelje *optbase*-a prema Pymoo biblioteci. Da bi korisnik povezao *optlib_pymoo_proto* sa svojim optimizacijskim problem, treba dodati samo jednu liniju kôda: *op.opt_algorithm = PymooOptimizationAlgorithmMulti()*[18].Objektu tipa

[18] Postoji i klasa *PymooOptimizationAlgorithmSingle* za jedonciljne probleme.

OptimizationProblem nazvanom *op* potrebno je atribut u obliku klase *PymooOptimizationAlgorithmMulti*.

Pojednostavljeno, modul sadrži sljedeće klase:

- *PymooConstraint*
- *WrappedPymooProblem*
- *PymooOptimizationAlgorithm*

Klasa *PymooConstraint* nasljeđuje opću klasu *DesignConstraint* iz *optbase* modula. Ona koristi metodu klase *DesignConstraint* nazvanu value_lt_0_normalized() u kojoj se izvršava normalizacija ograničenja kako je pojašnjeno u 4.5.3.

PymooOptimizationAlgorithm klasa pokreće generiranje problema kakvog očekuje *Pymoo* biblioteka, instancira sve potrebne objekte iz rječnika *alg_ctrl* i pokreće dodijeljeni algoritam. Na kraju optimizacije poziva spremanje rješenja unutar *OptimizationProblem* objekta kojem je dodijeljena.

WrappedPymooProblem je *wrapper* klasa koja nasljeđuje od *Pymoo* klase *Problem* što je Pymoo algoritmima nužno za definiranje problema. Prepisuje dvije nužne funkcije: *__init__* i *_evaluate*.

5.4. Modul *optlib_jmetalpy_proto*

Slično kao za Pymoo, načinjeno je sučelje *optbase* modula prema jMetalPy biblioteci. Da bi povezali *optlib_jmetalpy_proto* modul sa optimizacijskim problemom, treba dodati sljedeću liniju kôda: *op.opt_algorithm = jMetalOptimizationAlgorithmMulti()*[19].

Skraćeno, modul čine sljedeće klase:

- *jmetalConstraint*
- *WrappedjmetalProblem*
- *jmetalOptimizationAlgorithm*

Iako izvedba ima svojih specifičnosti, po konceptu su ove tri klase identične odgovarajućim klasama u *optlib_pymoo_proto*.

5.5. Modul *jobbase*

Modul *jobbase* ključan je za paralelizaciju izračuna prilikom zadavanja više optimizacijskih postupaka odjednom što znatno ubrzava proračun. Uz to, poziva ispis mjera kvalitete u zasebnu .csv datoteku *quality_indicators*. Sadrži dvije klase:

[19] Postoji i klasa *jMetalOptimizationAlgorithmSingle* za rješavanje jednociljnih optimzacijskih problema

- *MultibjectiveOptimizationComparer*,
- *MultibjectiveOptimizationComparerFromWrittenResults*.

Uobičajeno se za paralelizaciju zadataka može koristiti *MultibjectiveOptimizationComparer*.

Ukoliko su prije provedeni proračuni i postoje zapisane izlazne .csv datoteke, tada se može koristiti *MultibjectiveOptimizationComparerFromWrittenResults*. Korištenjem takvog *job* objekta će se preskočiti ponovna optimizacija ako izlazna datoteka s istim imenom već postoji, samo će se izvršiti izračun mjera kvalitete i ponovno crtanje grafova.

5.6. Definiranje problema u modulu *Frame_problem*

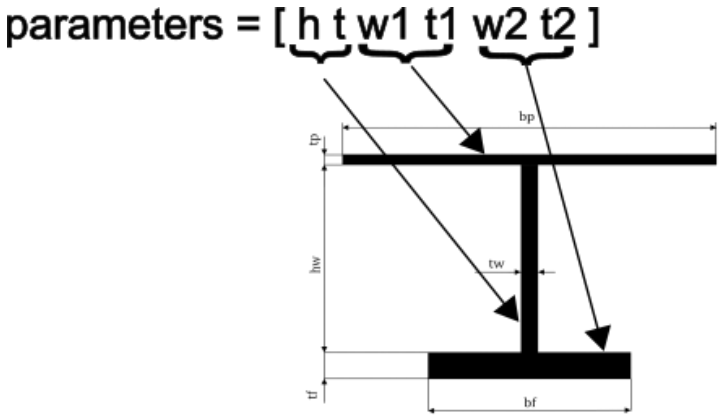
Da bi problem bio definiran nezavisno i fizički izdvojen od bilo kojeg drugog dijela kôda, odabran je takav pristup da je u novom zasebnom modulu nazvanom *Frame_problem.py* kreiran problem. Općenito je moguće i preporučeno sve probleme pri korištenju *moobench* biblioteke na isti način definirati. To znači, da je poželjno da korisnik *moobench* biblioteke neki proizvoljni problem formulira u izdvojenoj datoteci. Isječak kôda koji se koristio za definiranje problema okvira prikazan je u [Tablica 8.] pa se sljedeći opis može popratiti tamo.

Definirane su dvije klase: *Frame_analysis_model* koja nasljeđuje *AnalysisExecutor* i *Frame_analysis_OptimizationProblem* koja nasljeđuje *OptimizationProblem*. Klasa *Frame_analysis_model* predstavlja sponu modela problema i proračuna metodom pomaka sa optimizacijom. U klasi *Frame_analysis_OptimizationProblem* definiraju se projektne varijable sa granicama, ograničenja, ciljevi i sam model. Slično tome, i svaki drugi problem koji bi *moobench* korisnik htio optimizirati trebao bi sadržavati barem dvije klase koje bi nasljeđivale *OptimizationProblem* i *AnalysisExecutor* klase.

U konstruktoru *Frame_analysis_OptimizationProblem* klase prvi korak je instanciranje objekta modela putem tekstualne datoteke koja je napisana. Ta tekstualna datoteka, kao što je ranije objašnjeno, sadrži sve informacije o pontonu kakav je predstavljen u poglavlju 4.1. Tako je kreiran objekt *model* koji je zapravo tipa *Frame_analysis.Structure()* i sadrži liste svih greda, presjeka, čvorova i dr., i ima mogućnost pozvati funkcije koje proračunavaju okvir metodom pomaka. Zatim se instancira objekt *am* (skraćeno za *analysis model*) tipa *Frame_analysis_model* kojemu se prosljeđuje novokreiran objekt *model*.

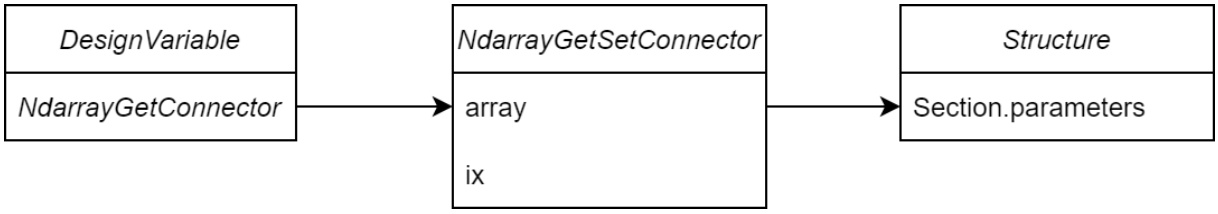
Taj objekt *am* vrlo je jednostavan. U konstruktoru sprema *model* kao vlastiti atribut. I, u metodi *analyze()* koju svaki objekt tipa *AnalysisExecutor* mora imati – poziva proračun okvira metodom pomaka putem funkcije iz *Frame_analysis.py* zvane *calculate_model()*.

U konstruktoru *Frame_analysis_OptimizationProblem* nastavlja se sa definiranjem problema. Slijedi dodijeljivanje projektnih varijabli. Putem metode *self.add_design_variable()* prolaskom petlje po svim gredama, dodaju se kao projektne varijable četiri parametra T nosača [Slika 14.], pritom izostavljajući dva parametra koja određuju oplatu (w_1, t_1).



Slika 14. Značenje pojedinih parametara u listi

Pritom je važno spomenuti da se koristi jedan tip konetora – *NdArrayGetSetConnector* koji povezuje projektnu varijablu *DesignVariable* pohranjenu u objektu tipa *OptimizationProblem* sa elementom u listi parametara *Section* objekta. Taj je odnos slikovito prikazan na [Slika 15.] pri čemu nije korištena nikakva konvencija u crtanju dijagrama te ga treba shvatiti konceptualno. Klasi *DesignVariable* dodjeljen je konektor koji sprema referencu na listu tj. *array* nekog objekta i indeks elementa u toj listi s kojim želimo povezati projektnu varijablu. Taj konektor može dohvaćati i mijenjati vrijednost tog parametra presjeka (*GetSet*) što je nužno za projektne varijable jer ih optimizacijski algoritmi nastoje mijenjati da bi dobili nova rješenja. Prilikom definiranje projektnih varijabli, prosljeđuju im se gornje i donje granice između kojih se njihove vrijednosti moraju naći.



Slika 15. Slikoviti prikaz jednog od konektora

Sljedeći je korak definirati funkcije cilja pomoću metode *self.add_objective()*. Njoj se prosljeđuje konektor tipa *RatioGetCallbackConnector()*. Ovaj konektor samo dohvaća vrijednost (*Get* za razliku od *GetSet*) jer algoritam ne može i ne treba izravno mijenjati vrijednost funkcije cilja[20]. *RatioGetCallbackConnector()* stavlja u razlomak dvije vrijednosti:

[20] Taj dio mora napraviti *AnalysisExecutor* koji vrši proračun metodom pomaka.

za brojnik dohvaća vrijednost *callback* funkcije koja mu je proslijeđena, a u nazivnik neku konstantnu vrijednost. Na taj način moguće je napraviti jednostavnu normalizaciju neke funkcije cilja u odnosu na neku konstantnu vrijednost. U slučaju pontona, definirani su ciljevi mase konstrukcije i vertikalne pozicije težišta. Konektor dohvaća *callback* metodu koja vraća trenutni iznos mase cijele konstrukcije i dijeli s početnom masom kod prve funkcije cilja, a kod druge dohvaća *callback* koji vraća vertikalnu poziciju težišta i dijeli s početnom visinom težišta. Te početne vrijednosti izračunate su za okvir kakav je definiran u ulaznoj tekstualnoj datoteci.

Sljedeći korak je definiranje ograničenja. Pozivom metode *self.add_constraint()* prvo su definirana ograničenja naprezanja. Za dohvaćanje vrijednosti koristi se konektor tipa *CallbackGetConnector*. On izvršava *callback* metodu grede koja vraća vrijednost najvećeg naprezanja. Desna strana nejednadžbe jest vrijednost dopuštenog naprezanja za materijal grede pohranjen u atributu *beam.prop.mat.sigmadop*.

Pozivom metode *self.add_constraint()* i korištenjem konektora *RatioDesignConnector* kreirana su ograničenja dimenzija za presjek kako je objašnjeno u poglavlju 4.5.3. Prolaskom po petlji, svakom konektoru u ograničenju su pomoću rječnika (*dictionary*) dodijeljene dvije projektne varijable čiji su odnosi ograničeni.

Konačno, definirano je još jedno ograničenje. Ograničena je cjelokupna masa konstrukcije prepoznajući da ipak masa predstavlja važan cilj i da mora biti nekako ograničena, jer rješenja s znatnom većom masom nego što je potrebno za čvrstoću (primjerice 30 ili 40%) neće dolaziti u obzir zbog svih loših posljedica koje povećana masa sa sobom nosi.

Tablica 8. Primjer kôda modula za definiranje problema

```
import Frame_analysis as fa
import numpy as np
from optbase import *

class Frame_analysis_model(AnalysisExecutor):

    def __init__(self,model):
        super().__init__()
        self.model = model

    def analyze(self):

        fa.calculate_problem(self.model)
        return AnalysisResultType.OK

class Frame_OptimizationProblem(OptimizationProblem):

    def __init__(self,name):

        super().__init__(name)
```

```
InputFile = fa.Input_file('pontoon_input_file.txt',40)
InputFile.load_file()
model = InputFile.create_model()
fa.calculate_problem(model)

#AnalysisExecutor
am = Frame_analysis_model (model)
self.add_analysis_executor(am)

#PROJECT VARIABLES

sections_to_opt = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]

lbs = [500, 5, None,None, 50, 5]
ubs = [1500, 25, None, None, 500, 30]

for section in model.sections:

    if section.ID in sections_to_opt:

        if type(section) == fa.I_Beam:

            names = ['hw','tw','bfa','tfa','bfb','tfb']
            index = 0

            for (parameter,lb,ub,name) in zip(
                     section.parameters, lbs, ubs, names):

                if ((index! = 2) & (index! = 3)):
                    self.add_design_variable(
                                           DesignVariable(
                                       name+str(section.ID),
                                   NdArrayGetSetConnector(
                    section.parameters,index),lb, ub))

                index+=1

#OBJECTIVES
w0 = model.get_mass()
VCG0 = model.get_vertical_CG_position()

self.add_objective(DesignObjective(
                   'mass',RatioGetCallbackConnector(
                   model.get_mass, w0)))
self.add_objective(DesignObjective(
                   'VCG',RatioGetCallbackConnector(
                   model.get_vertical_CG_position, y0)))

#CONSTRAINTS

#STRESS
i = 0
for beam in model.beams:
    self.add_constraint(DesignConstraint(
```

```
                                'stress'+str(i),
                               CallbackGetConnector(beam.get_stress),
                              beam.prop.mat.sigmadop, ConstrType.LT))
            i+=1

        #DIMENSION CONSTRAINTS

        dictionary={}   #dictionary of all desvars with their
names

        for ix in range(self.num_var):

            desvar=self.get_desvar(ix)
            name=str(desvar.name)
            dictionary[name]=desvar

        i=0
        for section in model.sections:

            if section.ID in sections_to_opt:

                    if type(section)==fa.I_Beam:

                        name1='bfb'+str(section.ID)
                        name2='tfb'+str(section.ID)
                        self.add_constraint(DesignConstraint(
                                          'g_bfb_tfb'+str(i),
                                        RatioDesignConnector(
                                           dictionary[name1],
                                         dictionary[name2]),4))
                        i+=1
                        self.add_constraint(DesignConstraint(
                                          'g_bfb_tfb'+str(i),
                                        RatioDesignConnector(
                                           dictionary[name1],
                                       dictionary[name2]),20,
                                               ConstrType.LT))
                        i+=1

                        name1='hw'+str(section.ID)
                        name2='tw'+str(section.ID)
                        self.add_constraint(DesignConstraint(
                                           'g_hw_tw'+str(i),
                                        RatioDesignConnector(
                                           dictionary[name1],
                                       dictionary[name2]),90,
                                               ConstrType.LT))
                        i+=1

                        name1='bfb'+str(section.ID)
                        name2='hw'+str(section.ID)
                        self.add_constraint(DesignConstraint(
                                          'g_bfb_hw'+str(i),
                                        RatioDesignConnector(
                                           dictionary[name1],
```

```
                                dictionary[name2]),0.2))
                        i+=1
                        self.add_constraint(DesignConstraint(
                                        'g_bfb_hw'+str(i),
                                    RatioDesignConnector(
                                       dictionary[name1],
                                    dictionary[name2]),0.5,
                                             ConstrType.LT))
                        i+=1

                        name1='tfb'+str(section.ID)
                        name2='tw'+str(section.ID)
                        self.add_constraint(DesignConstraint(
                                        'g_tfb_tw'+str(i),
                                    RatioDesignConnector(
                                       dictionary[name1],
                                    dictionary[name2]),1))
                        i+=1

self.add_constraint(DesignConstraint('g_tfb_tw'+str(i),

RatioDesignConnector(dictionary[name1],dictionary[name2]),3,

ConstrType.LT))
                        i+=1


        w_upper = 4.331567857152000 * 10 ** 13 * 1.20

        self.add_constraint(DesignConstraint(
                        'g_mass_uper'+str(i),CallbackGetConnector(
                         model.get_mass),w_upper,  ConstrType.LT))
```

5.7. Skripta *Run_task_executor*

Primjer skripte kakvu je poželjno definirati za pokretanje i provođenje analize prikazana je u [Tablica 9.]. Kao što je vidljivo, potreban broj uvoza dijelova biblioteka nije zamoran. U jednoj funkciji dovoljno je definirati optimizaciju ili pak niz optimizacija koje će se izvršiti. Ovdje je u funkciji *opttest_NSGA2* definirano provođenje jednog proračuna optimizacije Pymoo bibliotekom. U početnom dijelu funkcije instanciran je *Problem* iz prije izrađenog modula *Frame_problem* opisanog u prošlom poglavlju (5.6.). Zatim slijedi postavljanja nekih osnovnih postavki genetskog algoritma:

- veličina populacije i broj iteracija,
- operatora mutacije i križanja,
- kriterija završetka.

Operatore mutacije i križanja dodjeljuje se u obliku rječnika. Jednako tako, kriterij završetka dodjeljuje se u obliku *tuple-a*. Sve je postavke potrebno složiti u jedan rječnik *alg_ctrl* koji se prosljeđuje klasi *PymooOptimizationAlgorithmMulti*. Taj rječnik interpretira razvijeni modul *optlib_pymoo_proto* koji kreira prikladne objekte prije nego što pokrene optimizacija. Konačno, kreira se objekt *job* koji predstavlja rukovodeći objekt izvršenja optimizacije kako je pojašnjeno u poglavlju o modulu *jobbase*. Dodjeljivanje nekih ulaznih argumenata i pozivanje funkcije *opttest_NSGA2* odvija se u glavnom dijelu skripte.

Tablica 9. Primjer koda skripte

```
from Frame_problem import Frame_Analysis_OptimizationProblem
from typing import List, Dict
from jobbase import MultibjectiveOptimizationComparer
from optlib_pymoo_proto import PymooOptimizationAlgorithmMulti

def opttest_NSGA2(name,max_workers,out_folder_path):

    ops:List[OptimizationProblem] = []
    op = Frame_Analysis_OptimizationProblem ('Frame')

    pop_size = 100
    num_iter = 200
    max_evaluations = pop_size * num_iter

    mutation = {'name':'real_pm', 'eta':20, 'prob': 1 /
op.num_var}
    crossover = {'name':'real_sbx', 'eta':20, 'prob':0.8}
    termination = ('n_eval', max_evaluations)

    alg_ctrl = {'pop_size': pop_size,'n_offsprings':pop_size,
                'mutation': mutation,'crossover': crossover,
                'termination':termination}

    op.opt_algorithm = PymooOptimizationAlgorithmMulti(
                              'pymoo_nsga_ii_test_values', 'nsga2',
                               alg_ctrl = alg_ctrl)
    ops.append(op)

    #JOB
    job = MultibjectiveOptimizationComparer(name, max_workers,
                                            out_folder_path, ops)
    job.execute()

if __name__ == '__main__':

    out_folder_path = 'C:\Documents\out'
    max_number_of_workers = 1
    opttest_NSGA2('Test of values',1, out_folder_path)
```

6. Usporedba kvalitete rješenja višeciljne optimizacije ostvarenih različitim optimizacijskim algoritmima

Optimizacija je izvođena na HP Pavilion Gaming Laptop 15 sa sljedećim specifikacijama:

- CPU Intel® Core™ i7-8750H 2,20 GHz
- 8,00 GB RAM (2 jedinice SK Hynix 2667 MHz DDR4)

Korišten je Python 3.9 interpreter. Za paralelizaciju procesa korišten je *ProcessPoolExecutor* iz *concurrent.futures* u Pythonu komu je proslijeđeno *max_workers* = 5. CPU ima 6 fizičkih jezgri, pa je na taj način osigurano da ne dolazi neravnomjernog dijeljena opterećenja između jezgri što bi dovelo do toga da bi rezultati bili neusporedivi.

Općenito se može reći – svaki problem zahtjeva odabir prikladnog optimizacijskog algoritma za njegovo rješavanje. Uz odabir algoritama, obično je nužno prilagoditi operatore algoritma. S obzirom na tip problema, postoje indikacije i istraživanja koja pokazuju kakvi su operatori prikladni za njegovo rješavanje.

Za kreiranje referentne Pareto fronte korišten je NSGA-II algoritam iz Pymoo biblioteke u tzv. stacionarnoj varijanti (*steady-state variant*) [5]. Stacionarna varijanta postiže se odabirom određenih specifičnih postavki algoritma. Prvo, roditeljska populacija u odnosu na potomke mora biti znatno veća. U ovom radu je uzeta populacija od 300 jedinki, dok su operacije reprodukcije generirale u svakoj generaciji 3 nove jedinke. Za operator križanja odabran je *SBXCrossover* s vjerojatnošću *prob* = $1/n_{var}$. Za operator mutacije uzeta je polinomna mutacija (*polynomial*) sa vjerojatnošću *prob* = $1/n_{var}$ i η=20, gdje n_{var} stoji za broj projektnih varijabli. To su često viđene postavke algoritma. Postakve operatora za referentnu frontu prikazane su u [Tablica 10.]

Tablica 10. Odabrane veličine za provođenje

Veličina populacije, *pop_size*	300
Veličina potomstva, *offspring_size*	3
Operator križanja: *SBXCrossover*	prob = 0,8 η = 20
Operator mutacije: *Polynomial*	prob = $1/n_{var}$ η = 20
Kriterij završetka: *time*	2 h

6.1. Korišteni algoritmi

Za usporedbu su odabrani algoritmi prikazani u [Tablica 11.]. Tri algoritma iz biblioteke Pymoo predstavljaju srodne genetske algoritme, tj. varijaciju NSGA-II algoritma. Iz biblioteke jMetalPy za usporedbu je korišteno 6 algoritama. NSGA-II radi usporedbe implementacije s onom iz Pymoo biblioteke. GDE3 predstavlja algoritam temeljen na operatorima diferencijalne evolucije. IBEA je genetski algoritam u kojem se pretraživanje vodi vrijednošću sposobnosti (*fitness*) uz korištenje binarnog pokazatelja kvalitete. Algoritam MOCell označava grupu ćelijskih genetskih algoritama algoritama za višeciljnu optimizaciju koji kombiniraju različite strategije. Kod ćelijskih algoritama, reprodukcija se odvija samo između susjednih rješenja. Uz, to nedominirana rješenja pohranjuju se u arhivu koja se koristi za unapređenje pretraživanja prostora. Algoritmi OMOPSO i SMPSO predstavljaju algoritme rojeva (*pso – particle swarm optimization*).

Tablica 11. Uspoređeni algoritmi

Pymoo	jMetalPy
NSGA-II	NSGA-II
NSGA-III	GDE3
UNSGA-III	IBEA
	MOCell
	OMOPSO
	SMPSO

6.1.1. Korišteni operatori – njihove kombinacije

Korišteni operatori prikazani su u [Tablica 12.]. Polinomsku mutaciju i SBX križanje koriste NSGA-II, NSGA-III, U-NSGA-III, IBEA. Diferencijalno križanje koristi GDE3 algoritam. Uniformnu i ne-uniformnu mutaciju koristi algoritam OMOPSO. SMPSO algoritmu dodijeljen je samo operator polinomne mutacije.

Tablica 12. Korišteni operatori.

Mutacije	Križanja
Polinomska	SBX križanje
Uniformna	
Ne-uniformna	
Diferencijalna	

6.2. Postavke algoritama

Postavljene su i provedene 23 različite kombinacije algoritama s različitim postavkama operatora. S obzirom da je priroda algoritama stohastička, potrebno je jednim algoritmom optimizirati više puta da bi se na temelju rezultata moglo nešto zaključiti. Ovdje je svaki algoritam sa svojim postavkama izvršen 10 puta.

Više su uspoređivani različiti algoritmi i njihove implementacije nego različite kombinacije operatora jer bi broj kreiranih izlazni datoteka bio vrlo velik, a samo postavljanje usporedbe bilo bi znatno kompleksnije I, u ovom slučaju kreirano je (23 × 10) 230 datoteka. Ipak, na osmišljavanje takvog eksperimenta na temelju *moobench* biblioteke bilo bi vrijedno utrošiti vrijeme. U prvu skupinu uvršteni su oni algoritmi koji se koriste kombinacijom operatora simuliranog binarnog križanja (*SBX Crossover*) i polinomske mutacije *(Polynomial Mutation)* [Tablica 13.]. U stupcima „križanje“ i „mutacija“ dane su skraćenice postavki koje su prikazane u [Tablica 15.]. U kriterijima završetka prvi broj u umnošku označava veličinu popuacije ili roja (300), dok drugi broj označava broj iteracija.

Tablica 13. Postavke prve skupine generacijskih algoritama

Slučaj	Biblioteka	Algoritam	Križanje	Mutacija	Kriterij završetka (broj evaluacija)
1.	Pymoo	NSGA-II	Csbx1	Mp1	300 × 500 = 150 000
2.	Pymoo	NSGA-II	Csbx1	Mp2	300 × 500 = 150 000
3.	Pymoo	NSGA-II	Csbx2	Mp1	300 × 500 = 150 000
4	Pymoo	NSGA-III	Csbx1	Mp1	300 × 500 = 150 000
5.	Pymoo	NSGA-III	Csbx1	Mp2	300 × 500 = 150 000
6.	Pymoo	NSGA-III	Csbx2	Mp1	300 × 500 = 150 000

7.	Pymoo	U-NSGA-III	Csbx1	Mp1	300 × 500 = 150 000
8.	Pymoo	U-NSGA-III	Csbx1	Mp1	300 × 500 = 150 000
9	Pymoo	U-NSGA-III	Csbx1	Mp1	300 × 700 = 210 000
10.	jMetalPy	NSGA-II	Csbx1	Mp1	300 × 500 = 150 000
11.	jMetalPy	NSGA-II	Csbx2	Mp2	300 × 500 = 150 000
12.	jMetalPy	IBEA	Csbx1	Mp1	300 × 500 = 150 000
13.	jMetalPy	IBEA	Csbx1	Mp2	300 × 500 = 150 000
14.	jMetalPy	MOCell	Csbx1	Mp1	300 × 500 = 150 000
15.	jMetalPy	MOCell	Csbx1	Mp1	300 × 700 = 210 000

Ono što se pokazalo kao jedan od važnih parametara kod algoritama NSGA-III i U-NSGA-III jest broj referetnih smjerova [Tablica 16.]. Ono što je moguće dodatno istražiti i kvalitetnije usporediti nego što je učinjeno u ovom je radu je kako broj referentnih smjerova utječe na broj rješenja i kvalitetu Pareto fronte i koliko je važan. Zasigurno, to je jedan parametar koji je uveden kako bi se poboljšala distribucija rješenja kod NSGA-2 algoritma i predstavlja na neki način izvravniju kontrolu korisnika na distribuciju rješenja duž referente fronte. No, ovdje je na neki način pokazano kako je to jedan osjetljiv parametar, za čije pridruživanje za nepoznate probleme očito treba slijediti određene upute. Ako ne, tada će automatski mehanizam povećanja distribucije rješenja na Pareto fronti zvan *Crowding Distance Sorting* primijenjen u NSGA-II algoritmu dati bolja rješenja.

Tablica 14. Postavke druge skupine algoritama

Slučaj	Biblioteka	Algoritam	Kombinacije kontrolnih parametara	Kriterij završetka
16.	jMetalPy	GDE3	cr = 0,2, f = 0,5	1h
17.	jMetalPy	GDE3	cr = 0,5,f = 0,8	1h
18.	jMetalPy	OMOPSO	„prob“ = 1/num_var	300 × 500 = 150 000
19.	jMetalPy	OMOPSO	„prob“ == 10 · 1/num_var	300 × 500 = 150 000
20.	jMetalPy	OMOPSO	„prob“ == 1/num_var	300 × 500 = 150 000
21.	jMetalPy	OMOPSO	„prob“ == 10 · 1/num_var	300 × 500 = 150 000
22.	jMetalPy	SMPSO	„prob“ == 1/num_var	300 × 500 = 150 000

23.	jMetalPy	SMPSO	„prob“ == 1/num_var	300 × 700 = 210 000

Tablica 15. Postavke operatora SBX križanja i polinomske mutacijske

Csbx1	η = 20, „prob“ = 0,8	Csbx2	η = 10, „prob“ = 0,25
Mp1	η = 20, „prob“ = 1/num_var	Mp2	η = 10, „prob“ = 0,25-

Kriterij završetka kod algoritma GDE3 postavljen je na 1h (za jedno izvršenje od 10 koje se provode za jednu postavku algoritma) jer algoritam nije davao rješenja u razumnom vremenu – tj. njegovo izvršenje trajalo je vrlo dugo što nije bilo praktično za čekanje.

Algoritmi kojima je dodijeljen povećan broj makismalnih evaluacija su (slučajevi 9., 15. i 21.) su oni kod kojih se željelo uočiti hoće li doći do znatnog povećanja kvalitete rješenja s povećanjem broja evaluacija, s obzirom da su pri brojci 150 000 uglavnom podbacili sa zadanim postavkama.

Tablica 16. Postavke referentnih smjerova

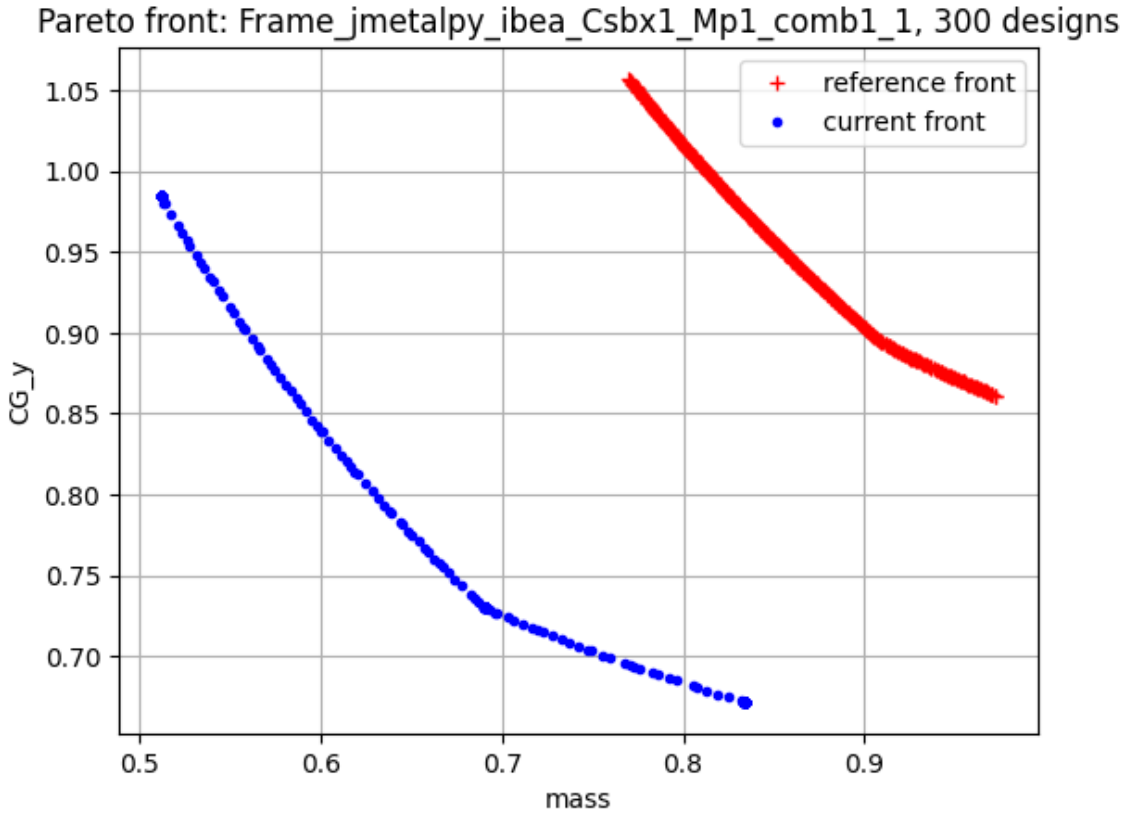
Slučaj	Algoritam	Broj referentnih smjerova
4	NSGA-III	120
5	NSGA-III	120
6	NSGA-III	120
7	U-NSGA-III	84
8	U-NSGA-III	240
9	U-NSGA-III	240

6.3. Usporedba kvalitete rješenja pri dimenzioniranju okvirnog rebra jednostavne brodske konstrukcije

U ovom poglavlju prikazani su rezultati za okvir pontona, problem koji je pojašnjen u poglavlju 4. zajedno s pojašnjenjem mjera kvalitete i dobivenih rezultata

Za određivanje referente fronte je korišten NSGA-II algoritam u stacionarnoj varijanti za koji je pokazano da daje dobro distribuirana rješenja [5]. Pymoo implementacija je otprilike 4 puta brža impementacija nego ona iz jMetalPy-a, pa je korištena Pymoo verzija algoritma. Kao kriterij završetka stavljeno je vrijeme od 2 sata u nadi da će proizvedena fronta biti dobra aproksimacija nepoznate teoretske Pareto fronte problema. Proizvedena fronta dominirala je

veliku većinu fronti ovdje proizvedenih algoritama, ali je algoritam IBEA iz jMetalPy-a naizgled tu frontu znatno pomakao u samo 150 000 evaluacija [Slika 16.]. No, takav je ishod bio sumnjiv, pa je u modul *optbase* dodan vlastiti kriterij kvalitete – broj prekršenih ograničenja u skupu rješenja koje je vratio algoritam optimizacijske biblioteke Utvrđeno je da je algoritam IBEA prekršio znatno više ograničenja negoli ikoji drugi algoritam, i to prosječno preko 4000 po fronti (tu je uzeta u obzir određena tolerancija prekoračenja u odnosu na normalizirano ograničenje). To ispada preko 13 ograničenja po projektu za 300 rješenja u fronti. Stoga se grafovi IBEA algoritma neće pokazivati, a općenito njegove mjere kvalitete jednostavno zanemariti jer nije ispunio svoju zadaću sa zadanim postavkama, a to je da pronađe optimalna rješenja i pritom zadovolji ograničenja.

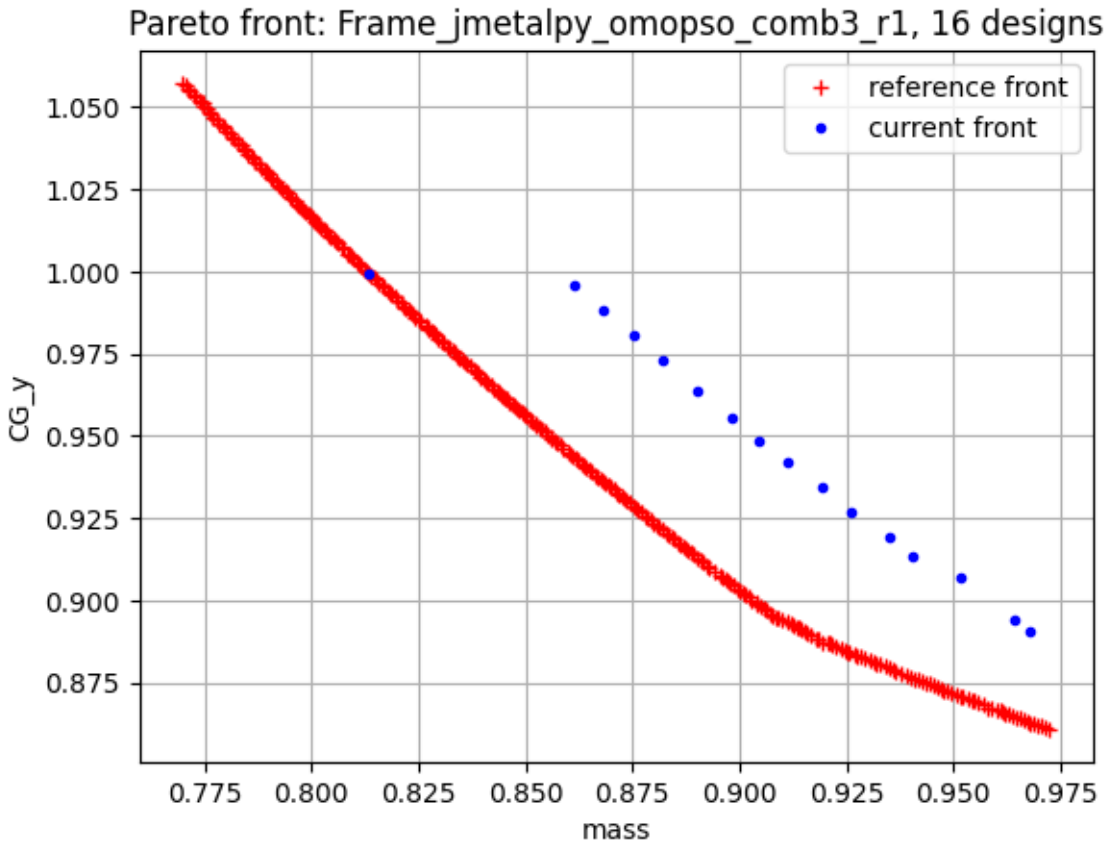


Slika 16. IBEA fronta u odnosu na referentnu frontu
NSGA-II algoritma – nije važeća

Za očekivati je da algoritam optimizacijske biblioteke vraća kao rezultat samo nedomirani skup rješenja. To su jedino optimalna rješenja, jedina rješenja koja imaju smisla biti rezultatom jedne optimizacije. No, s obzirom na nepotpunu dokumentaciju biblioteke jMetalPy nije točno utvrđeno zašto svi algoritmi ne vraćaju samo Pareto frontu, a ne cijelu populaciju ili neki drugi skup. To se jasno vidi na grafovima koji su prikazani u poglavlju 6.3.1.S obzirom da je svaki algoritam sa svakom postavkom proveden 10 puta, na temelju 10 rješenja kreirani su statistički

podaci – medijan i interkvartilni raspon. Na temelju njih kreirani su box-and-whiskers grafovi koji prikazuje obrađene mjere kvalitete fronti.

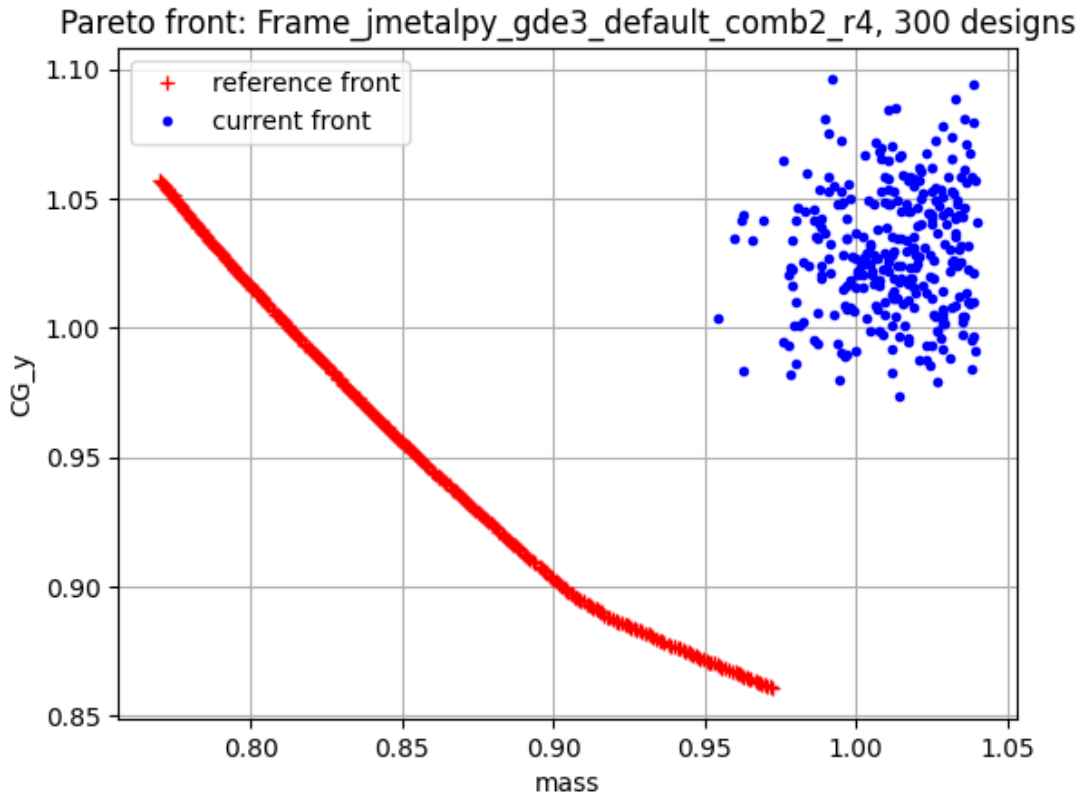
S obzirom da je algoritam OMOPSO dao loša rješenja promatrajući frontu i broj pronađenih rješenja, on je izostavljen iz razmatranja [Slika 17.].



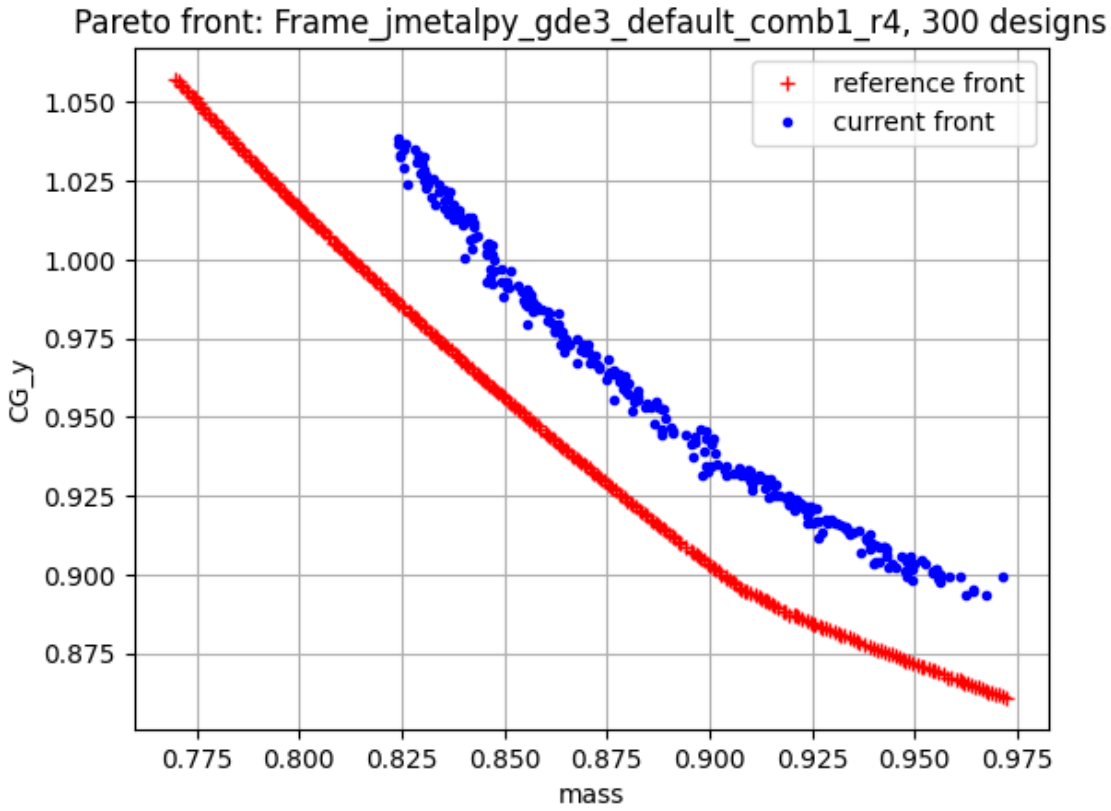
Slika 17. Rezultati OMOPSO algoritma

Očito je da GDE3 u slučaju 17. nije proizveo Pareto frontu te je izostavljen iz razmatranja [Slika 18.]. Sličan oblik rezultata postigao je i slučaj 11. korištenjem NSGA-II algoritma [Slika 20.]. Ako se pažljivije promotri, niti GDE3 u slučaju 16. nije proizveo Pareto fronte jer zasigurno nisu sva rješenja u toj fronti nedominirana, stoga je izostavljen i taj slučaj [Slika 19.]. Također, iz rezultata su izostavljene fronte IBEA algoritma jer bi iznosi pokazatelja dali sasvim krivu predodžbu o kvaliteti rješenja, a pri kojem su prekršena mnoga ograničenja. To ostavlja 15 slučajeva koji su uspoređeni putem rezultata mjera kvalitete. U [Tablica 17.] prikazani su prosječno vrijeme izvršavanja i broj prekršenih ograničenja svakog ostavljenog slučaja.

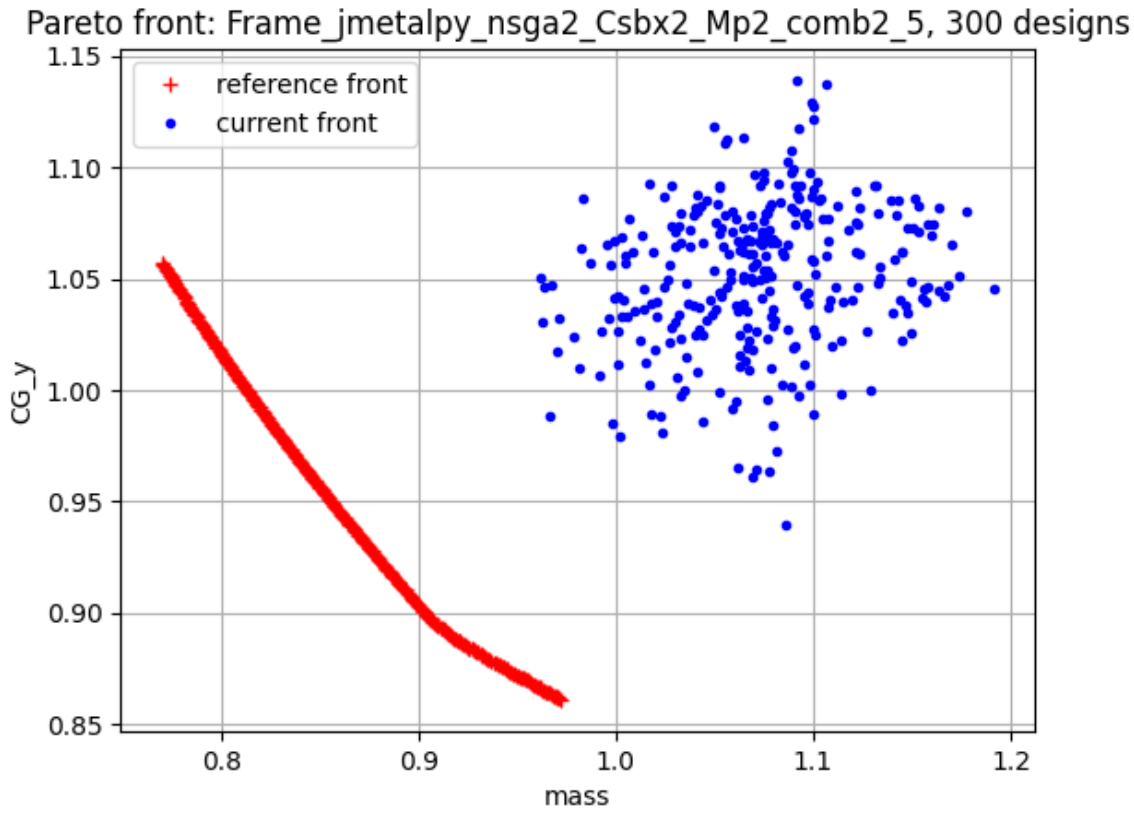
Na kraju je dan vizualni prikaz podataka skupa 10 ponovljenih rješenjavanja u obliku *box-and-whiskers* grafova. Na grafu prikaza broja prekršenih rješenja namjerno su prikazani svi algoritmi da bi se vidjelo pri kojim algoritmima je došlo do značajnog kršenja broja ograničenja.



Slika 18. Rezultati slučaja 17. s GDE3 algoritmom



Slika 19. Rezultati slučaja 16. s GDE3 algoritmom



Slika 20. Rezultati slučaja 10. s NSGA-II algoritmom

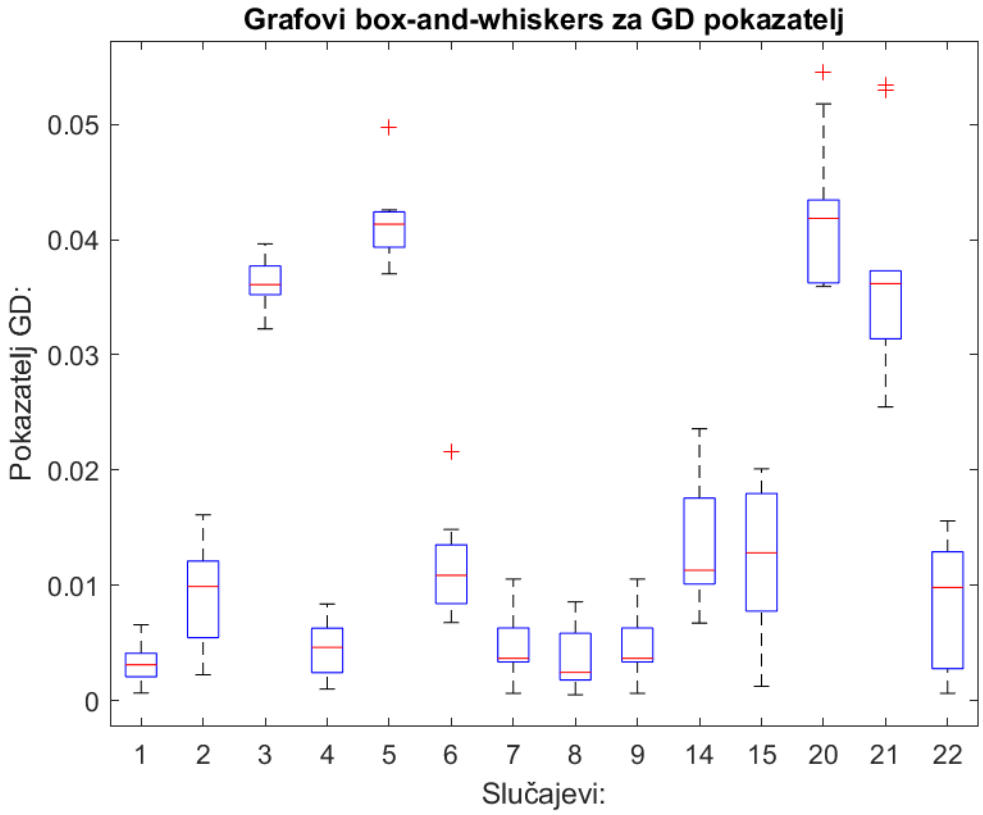
Tablica 17. Prikaz broja prekršenih rješenja i vremena izvršavanja

Slučaj	Algoritam	Broj prekršenih ograničenja	Vrijeme izvršavanja. s
1.	NSGA-II	0	332,45
2.	NSGA-II	0	338,77
3.	NSGA-II	0	327,90
4.	NSGA-III	0	337,06
5.	NSGA-III	0	328,42
6.	NSGA-III	0	342,01
7.	U-NSGA-III	0	370,04
8.	U-NSGA-III	0	355,117
9.	U-NSGA-III	0	370,39
10.	NSGA-II	0	3604,24

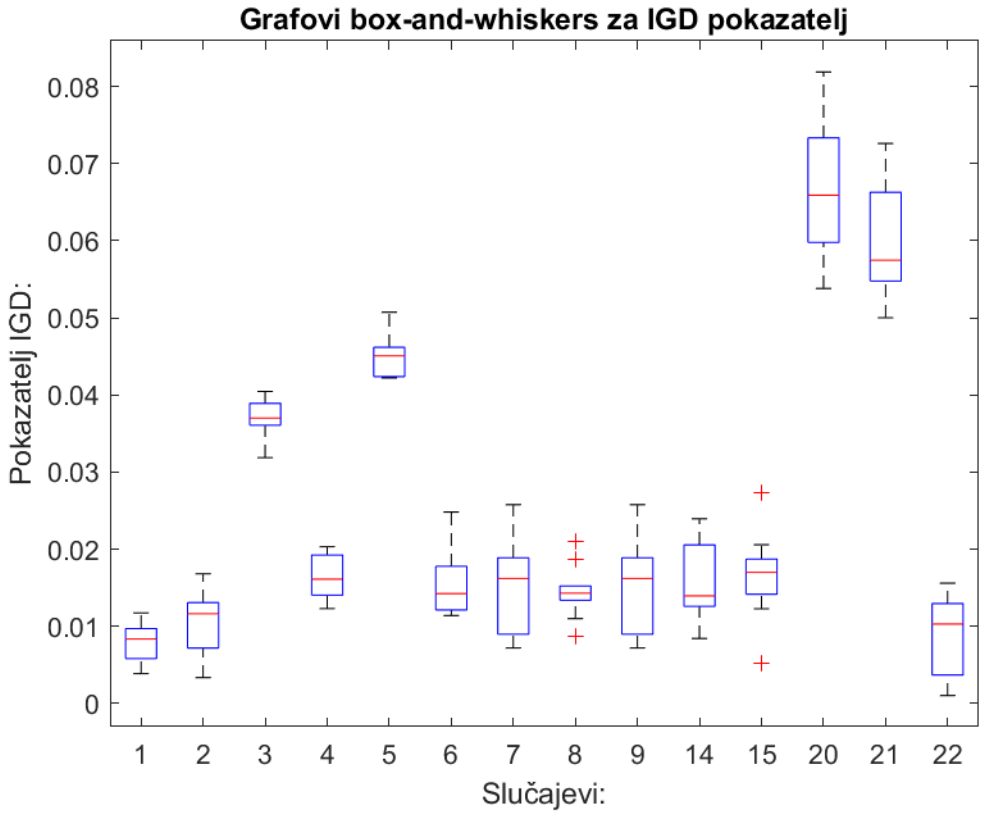
14 .	MOCell	0	1074,18
15.	MOCell	0	1636,91
22.	SMPSO	0	6217,83
23.	SMPSO	300	7236,81

Na temelju [Tablica 17.] vidljivo je da svi ostavljeni slučajevi, osim 23., nemaju prekršenih ograničenja. Broj od 300 prekršenih rješenja je vrlo vjerojatno jednak jednom projektu koji prelazi sve granice. Vrlo važno je primjetiti da se vrijeme izvršavanja razlikuje dramatično. Prva tri slučajeva traju prosječno tri puta kraće od slučaja 10 koji implementira isti algoritam u jMetalPy biblioteci. Očito, Pymoo implementacija je mnogo kvalitetnija i brža. SMPSO algoritam uzima gotovo 20 puta više vremena nego slučajevi 1. do 9.

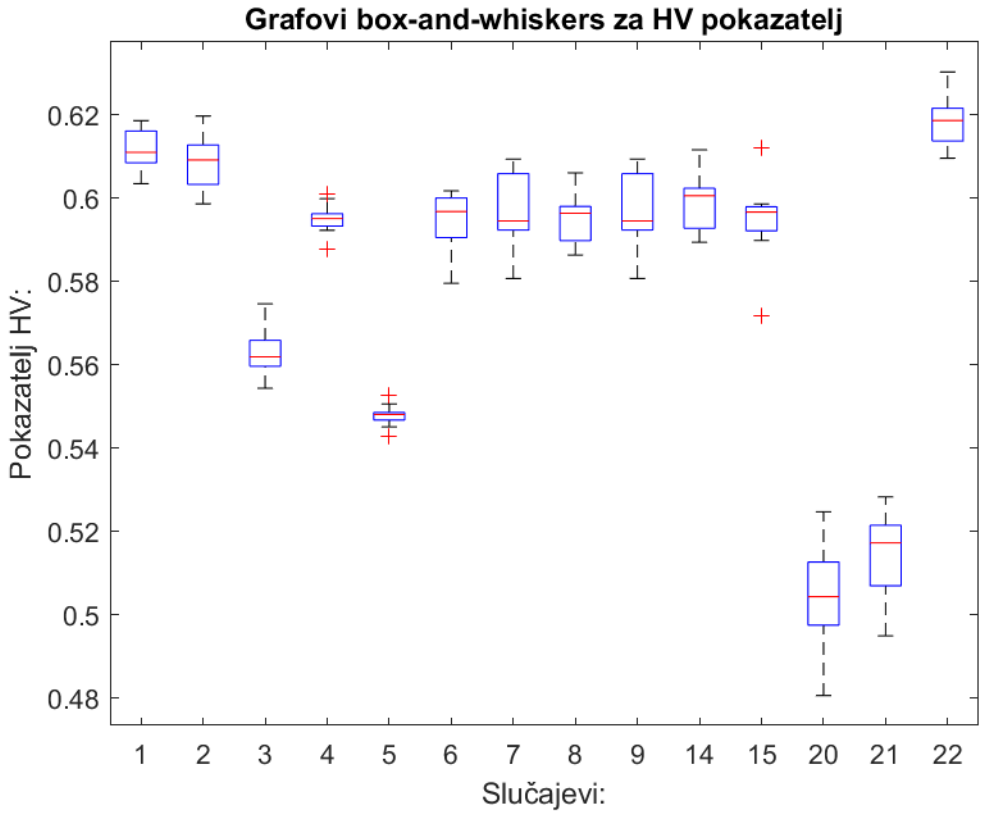
Usporedni prikaz grafovima *box-and-whiskers* prikazan je na sljedećim slikama. Na [Slika 21.] prikazan *box-and-whiskers* graf za GD pokazatelj. Slučaj 1. ističe se kao najbolji zajedno sa slučajevima 7., 8. i 9. Na [Slika 22.] ističu se slučajevi: 1., 2. i 22. Hipervolumen je najbolji kod slučajeva: 1., 2. i 22. [Slika 23.]. Epsilon pokazatelj najbolji je kod slučaja 22. [Slika 24.]. Vlastit ugrađeni pokazatelj za broj prekršenih rješenja prikazan je u sa svim slučajevima [Slika 25.]. On u potpunosti opravdava izbacivanje slučajeva 12. i 13. s algoritmom IBEA, ali donekle opravdava i slučajeve 11., 16., 17. i 19. S obzirom na strogost zadovoljenja ograničenja kod drugih algoritama, pitanje je je li slučaj 23. koji je po mnogim pokazateljima dobar, usporediv s onima koja su zadovoljiva. Zasigurno neka njegova rješenja ipak mogu ući u praktično razmatranje.



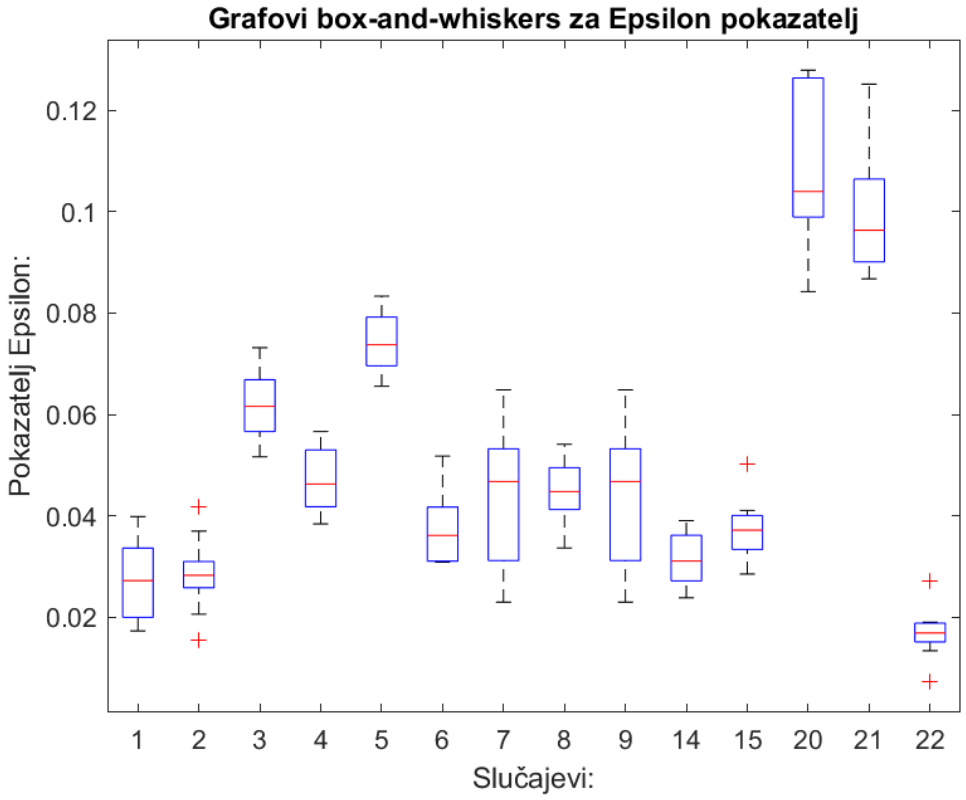
Slika 21. *Box-and-whiskers* graf za GD pokazatelj



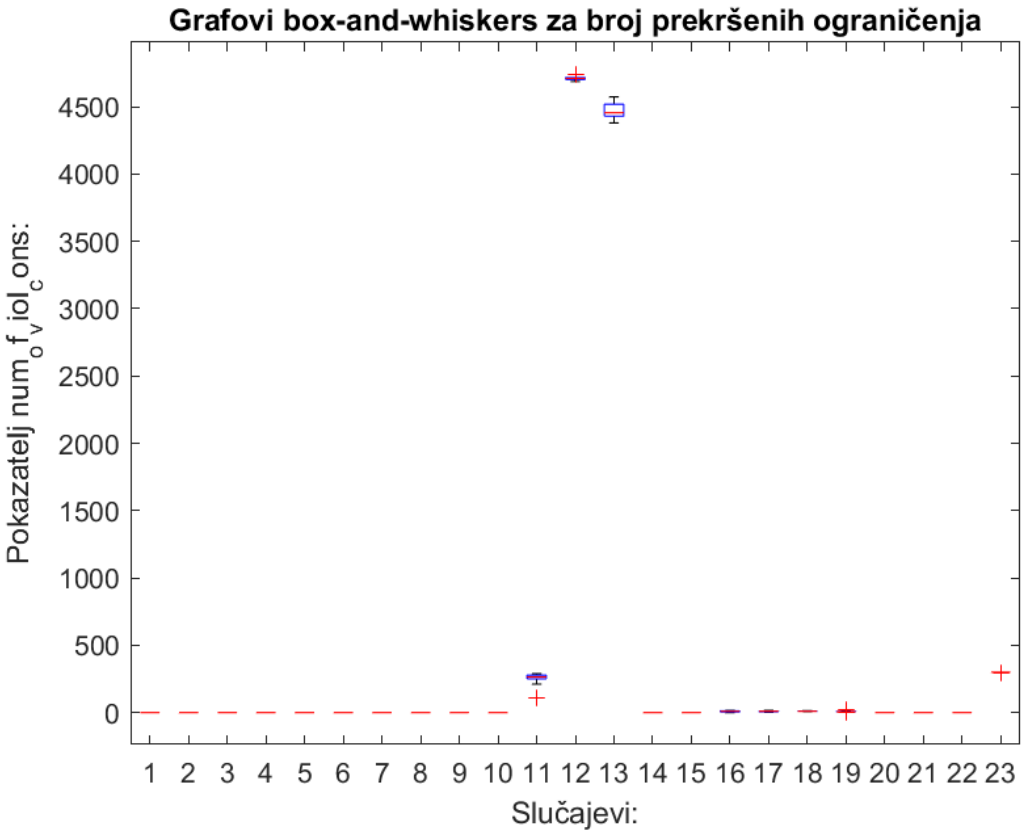
Slika 22. *Box-and-whiskers* graf za IGD pokazatelj



Slika 23. *Box-and-whiskers* graf za HV pokazatelj



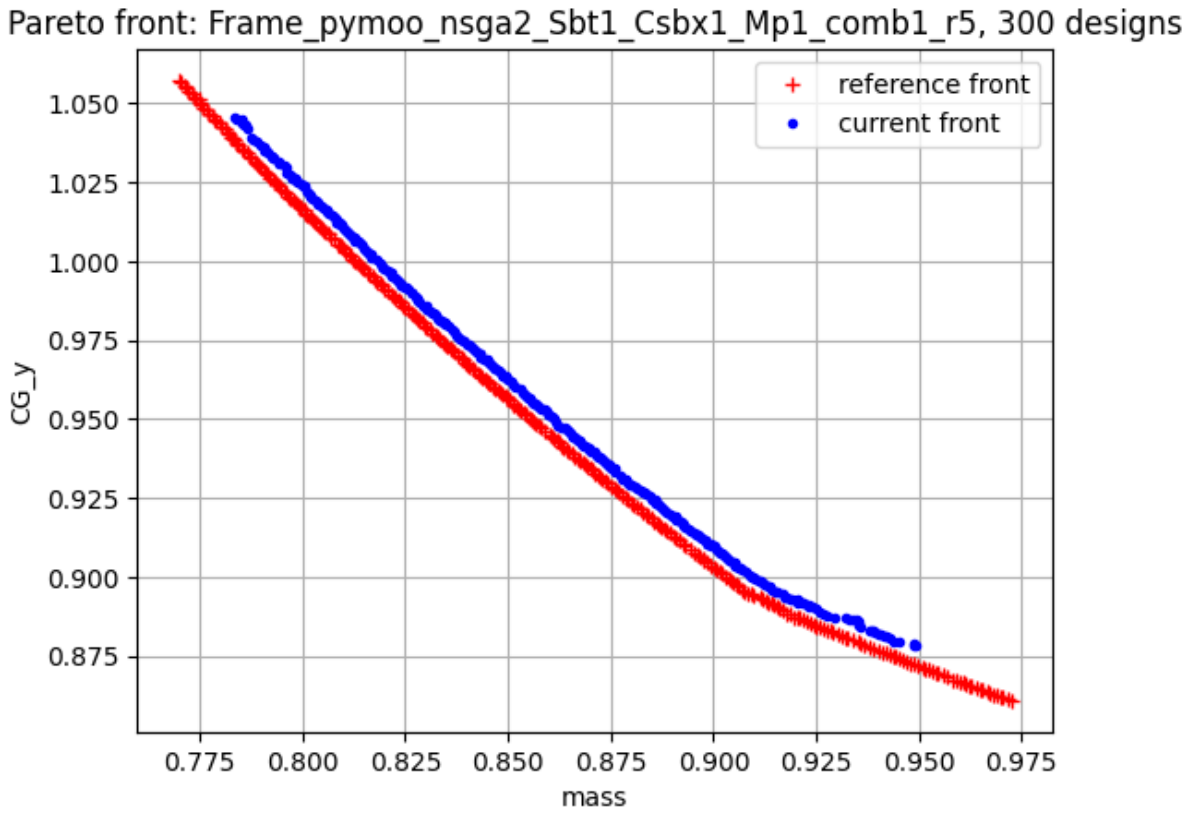
Slika 24. *Box-and-whiskers* graf za Epsilon pokazatelj



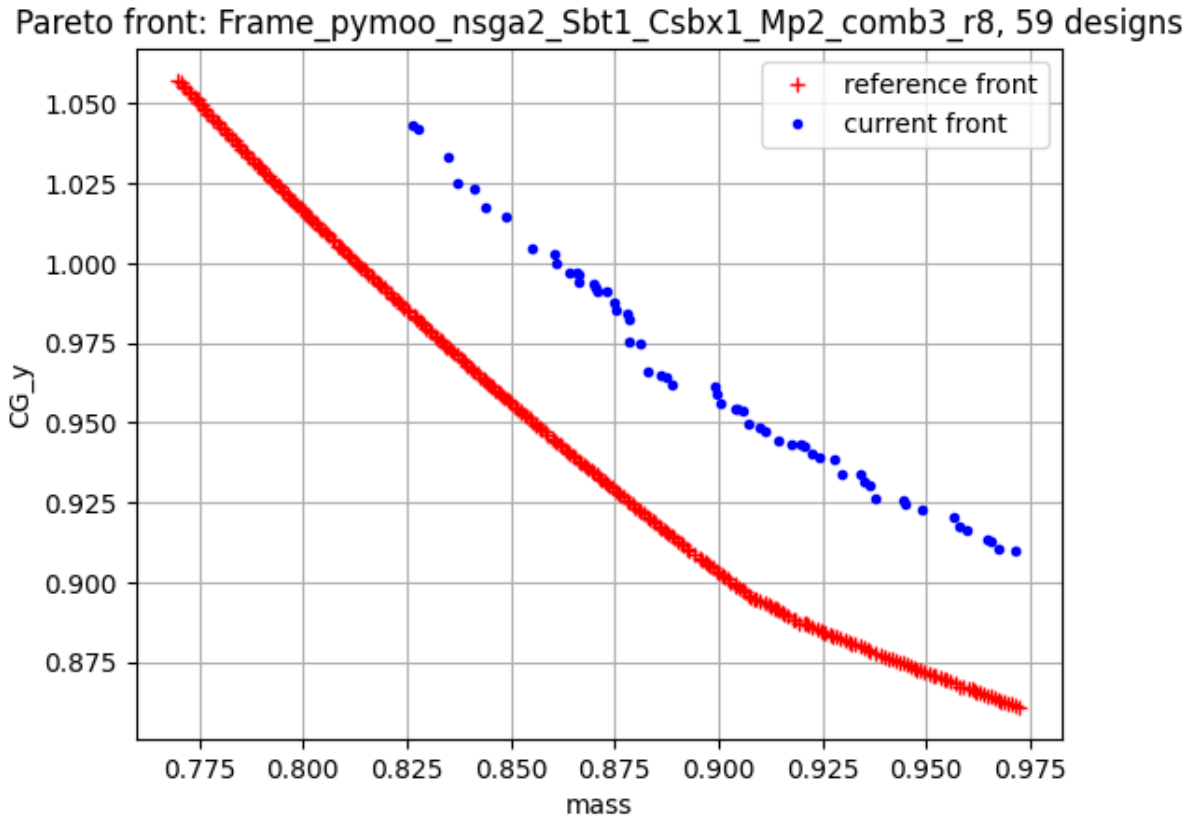
Slika 25. *Box-and-whiskers* graf za broj prekršenih rješenja

6.3.1. Grafički prikazi dobivenih Pareto fronti u odnosu na referentnu

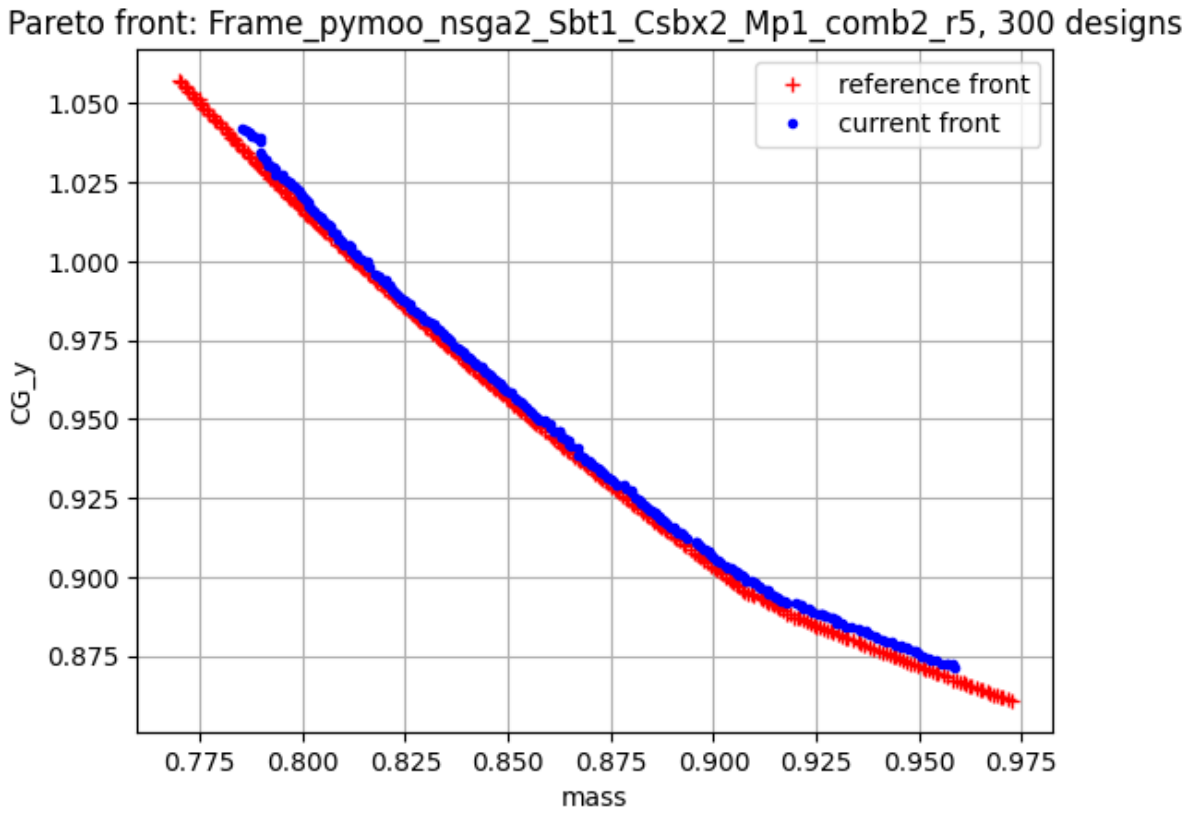
Na sljedećih nekoliko stranica dani su grafički prikazi rezultirajućih Pareto fronti u odnosu na referentu frontu s obzirom na dvije funkcije cilja. Na apscisi prikazan je cilj mase, a na ordinati prikazan je cilj vertikalnog težišta. Vrijednost 1 na osima predstavlja vrijednost koju ima inicijalni okvir zadan putem tekstualne datoteke.



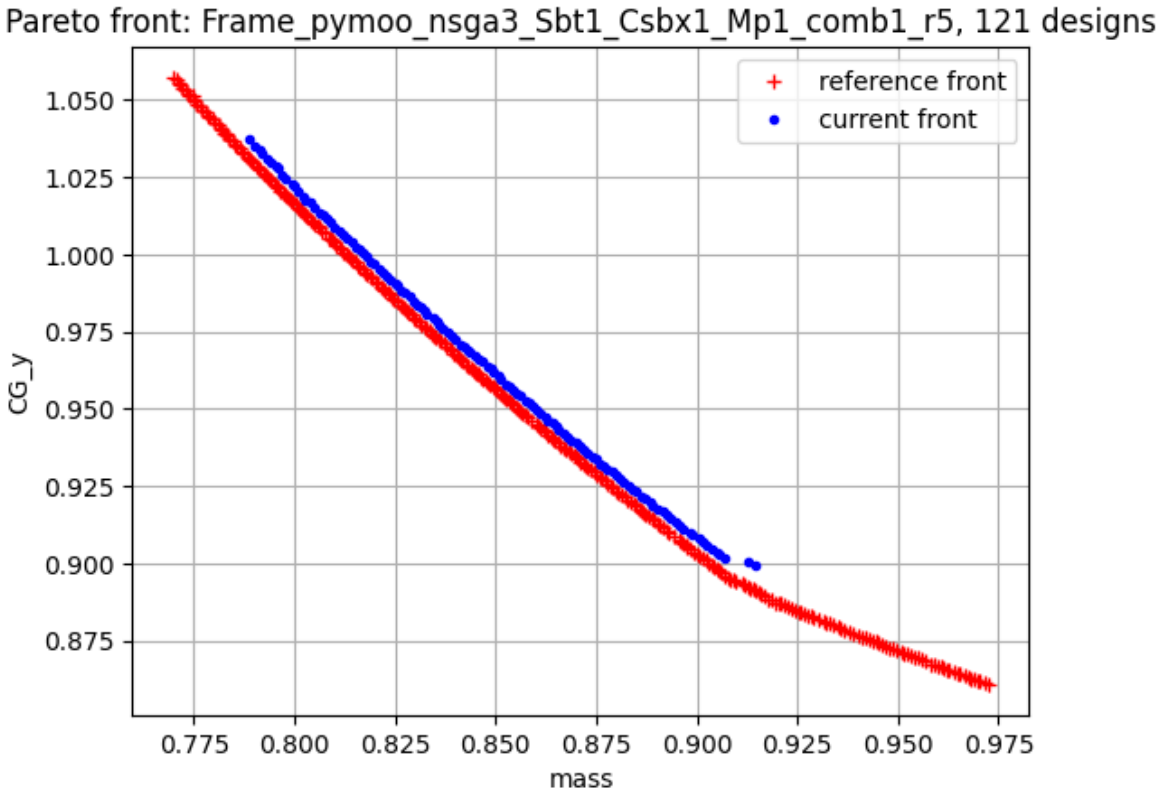
Slika 26. Slučaj 1.



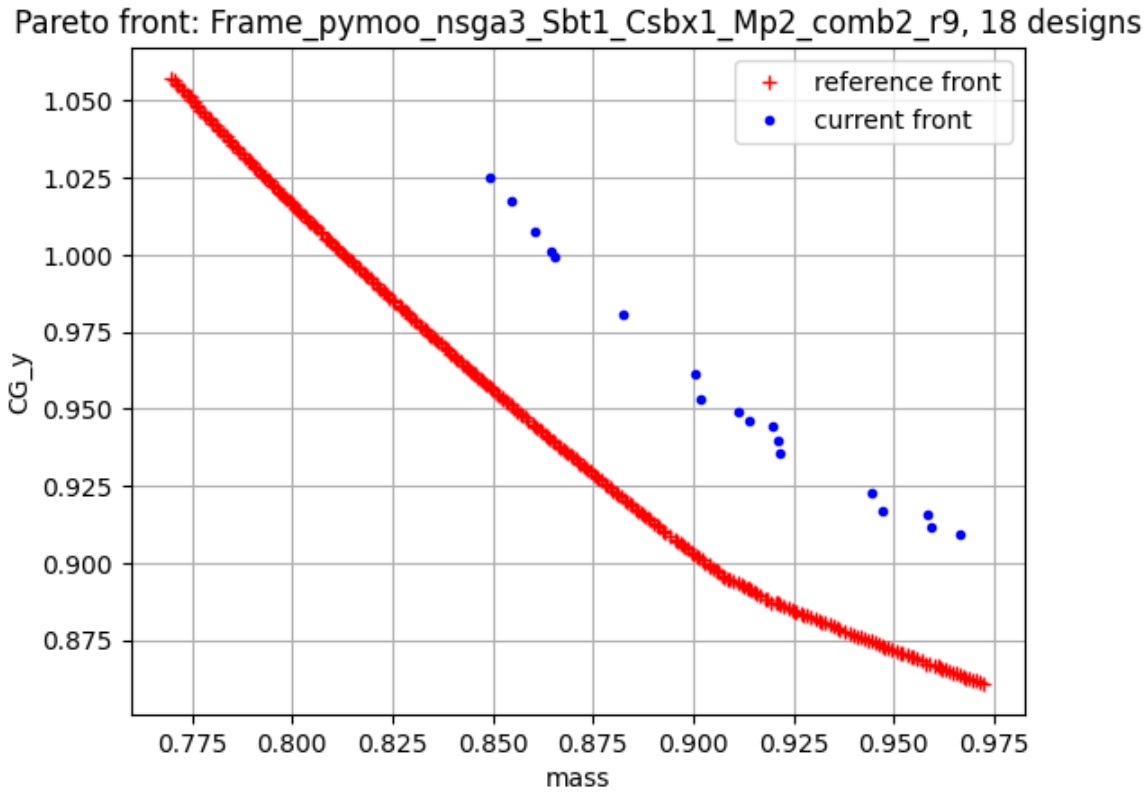
Slika 27. Slučaj 2.



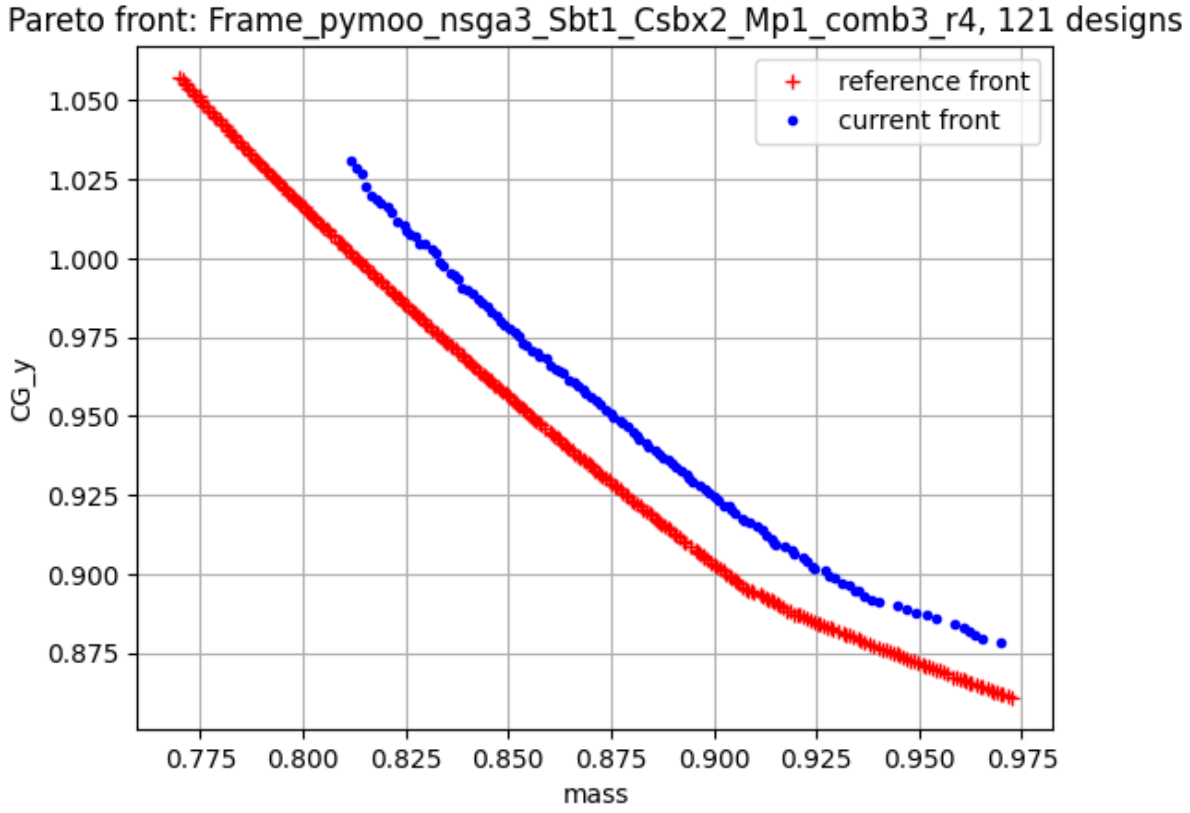
Slika 28. Slučaj 3.



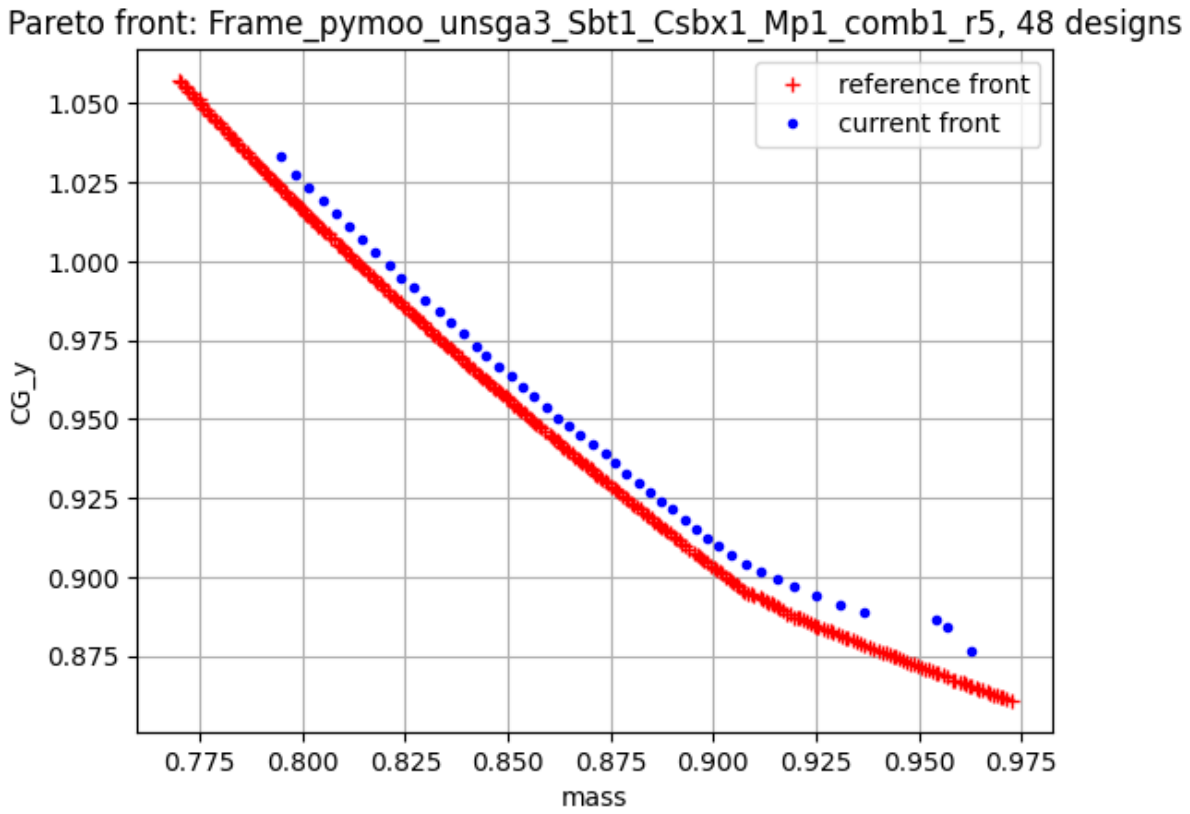
Slika 29. Slučaj 4.



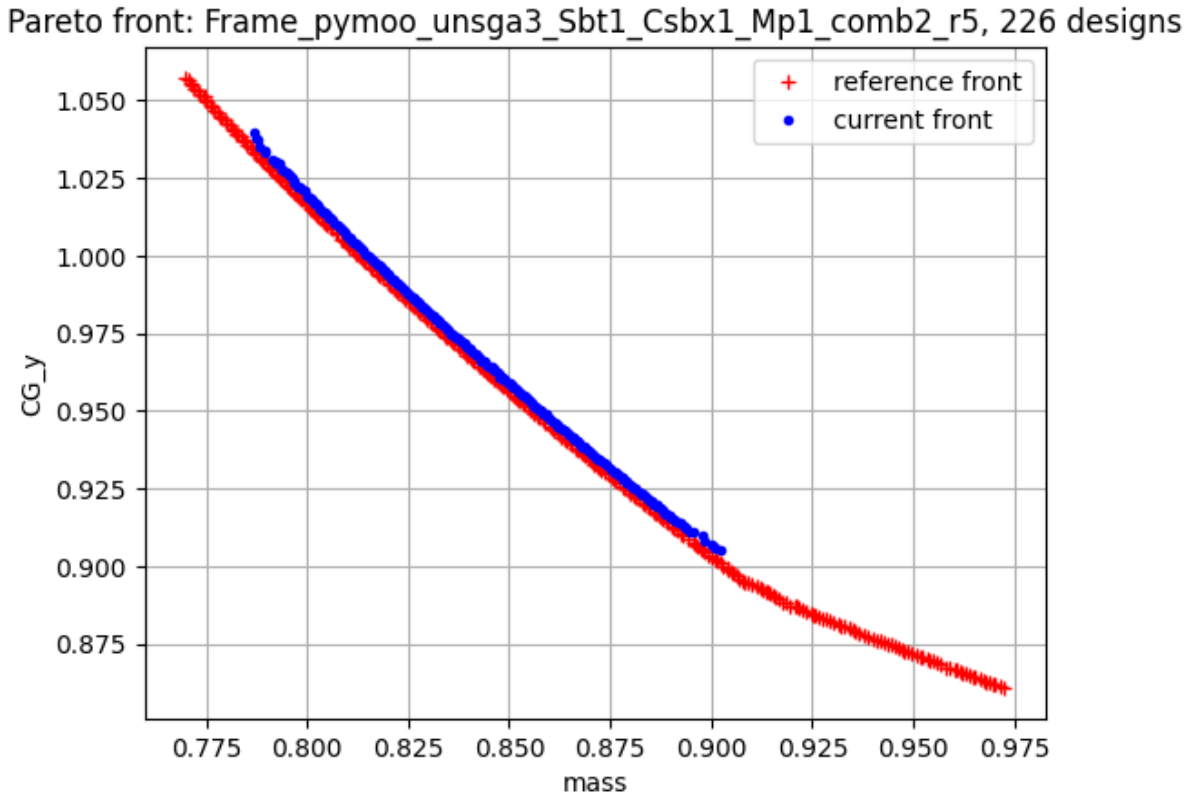
Slika 30. Slučaj 5.



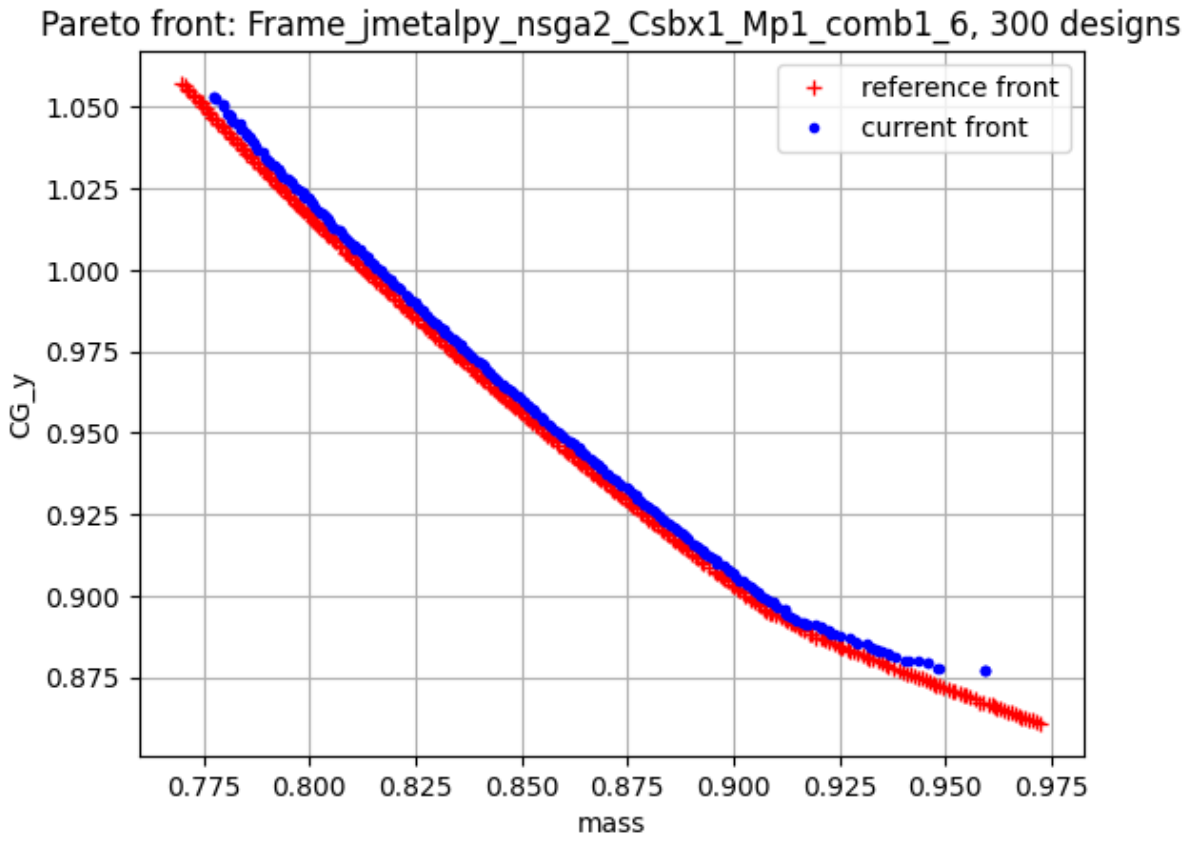
Slika 31. Slučaj 6.



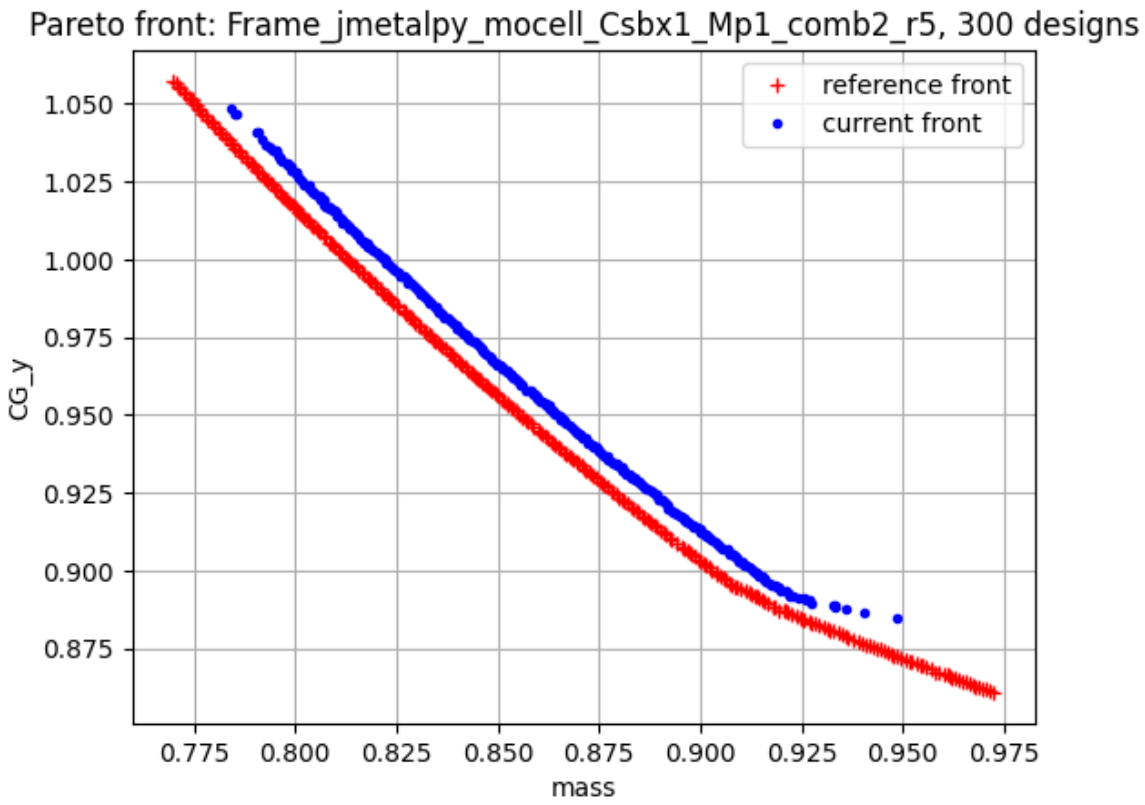
Slika 32. Slučaj 7.



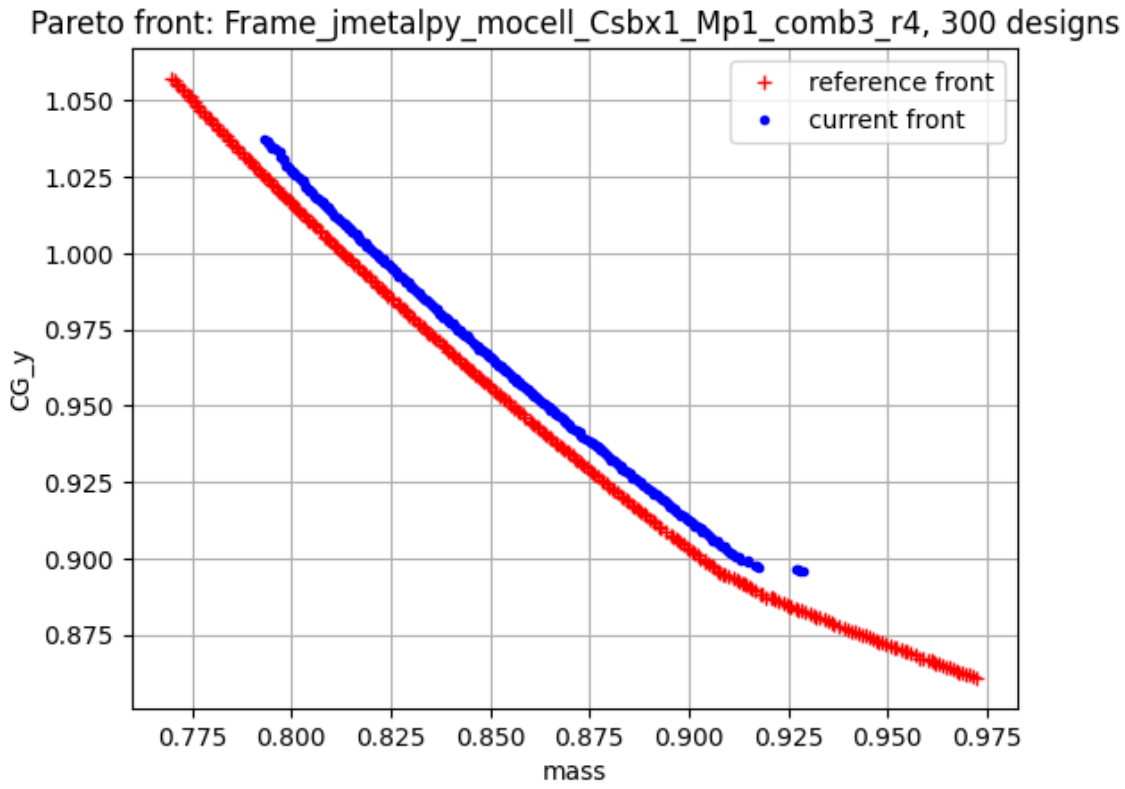
Slika 33. Slučaj 8.



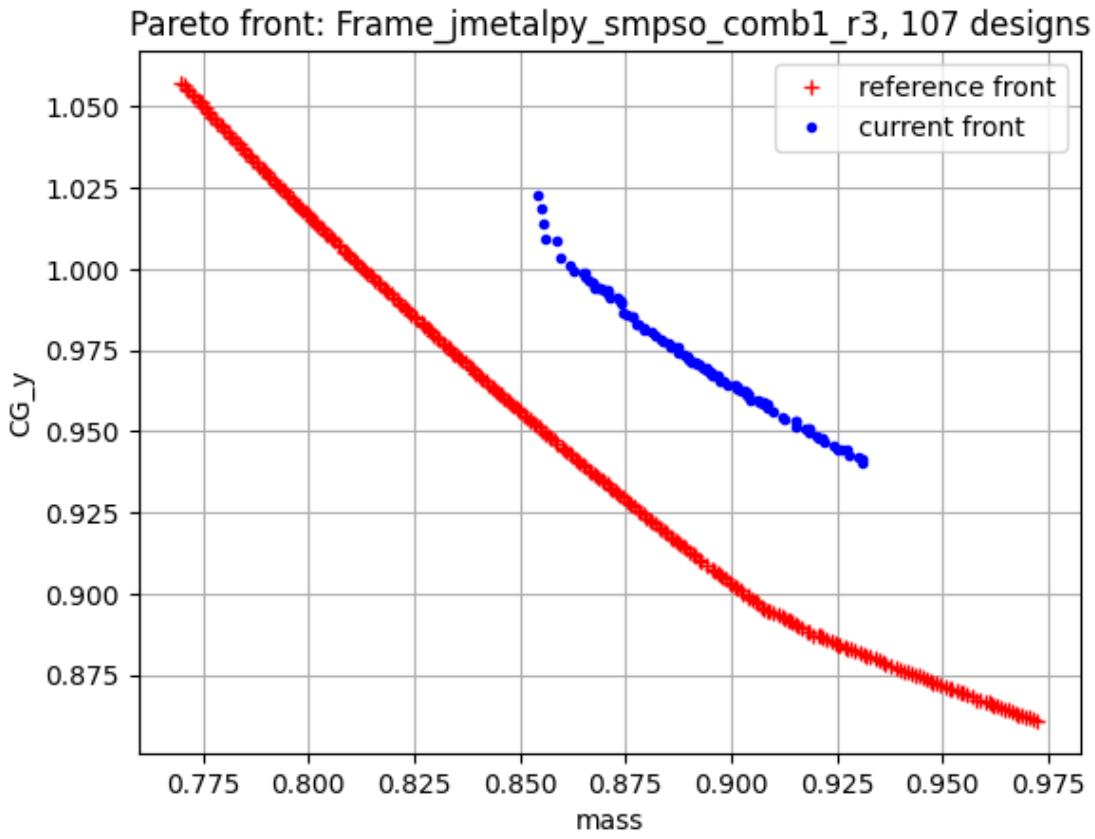
Slika 34. Slučaj 10.



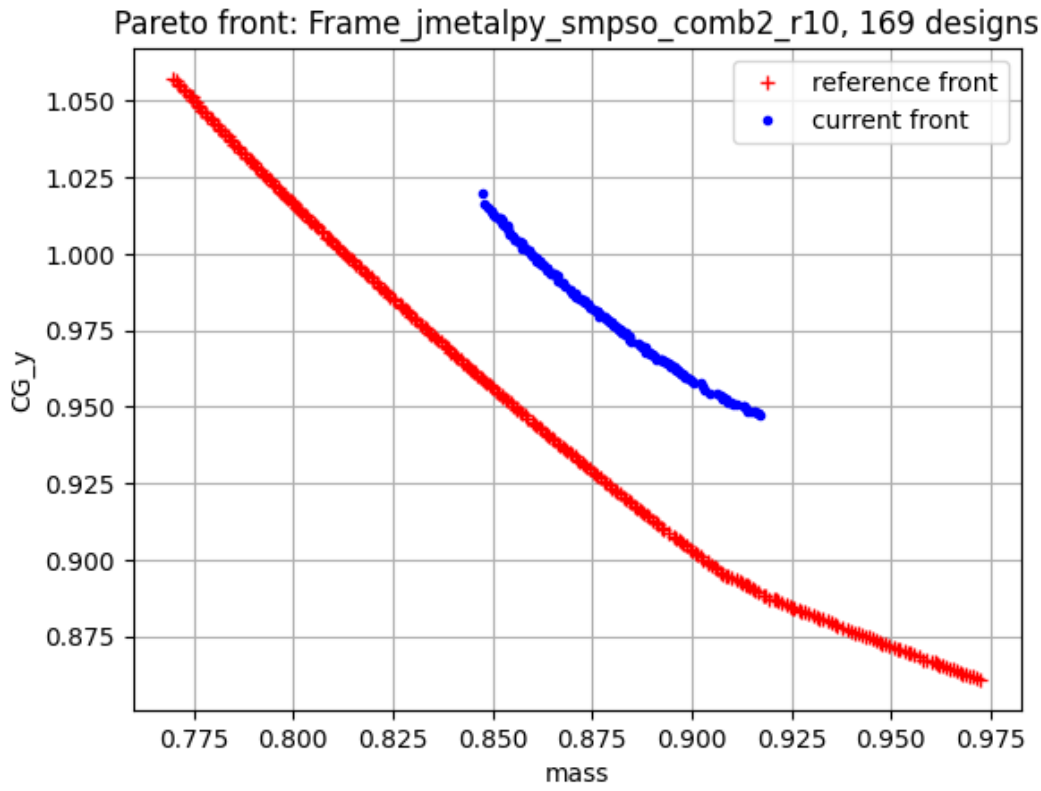
Slika 35. Slučaj 14.



Slika 36. Slučaj 15.



Slika 37. Slučaj 22.



Slika 38. Slučaj 23.

6.4. OSY

S istim postavkama kao za Okvir, proveden je i proračun na OSY problemu. Osyczka2 standardni je testni (*benchmark*) problem sa ograničenjima i dva cilja. Često je ugrađen u optimizacijske biblioteke, a za potrebe rada je rekonstruiran pomoću *moobench* biblioteke.

Ima šest projektnih varijabli definiranih kao: x_1, x_2, x_3, x_4, x_5 i x_6. Nad njima su definirana sljedeće granice:

- $0 \leq x_1 \leq 10$,
- $0 \leq x_2 \leq 10$,
- $1 \leq x_3 \leq 5$,
- $0 \leq x_4 \leq 6$,
- $1 \leq x_5 \leq 5$ i
- $0 \leq x_6 \leq 10$.

Funkcija cilja dane su sljedećim jednadbžama koje se minimiziraju:

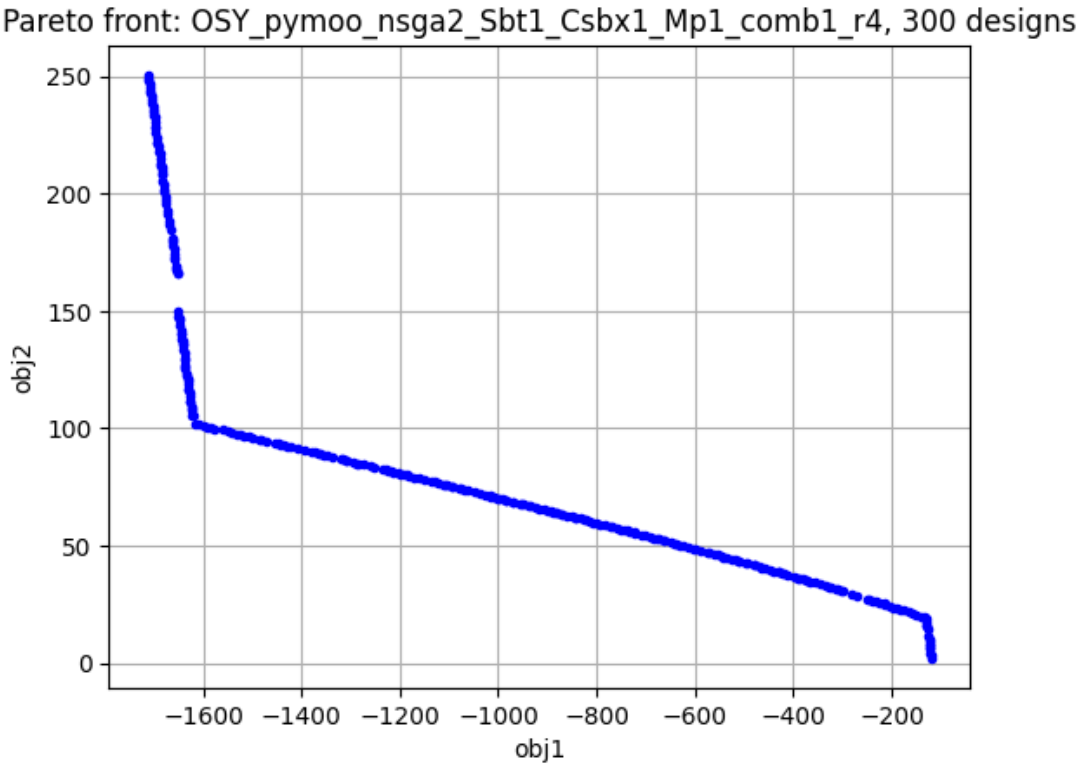
$$\begin{aligned} f_1(x) &= -[25(x_1-2)^2 + (x_2-2)^2 + (x_3-2)^2 + (x_4-2)^2 + (x_5-2)^2] \\ f_2(x) &= {x_1}^2 + {x_2}^2 + {x_3}^2 + {x_4}^2 + {x_5}^2 + {x_6}^2 \end{aligned} \quad \textbf{(15)}$$

Šest ograničenja u obliku nejednadžbi zadana su na sljedeći način:

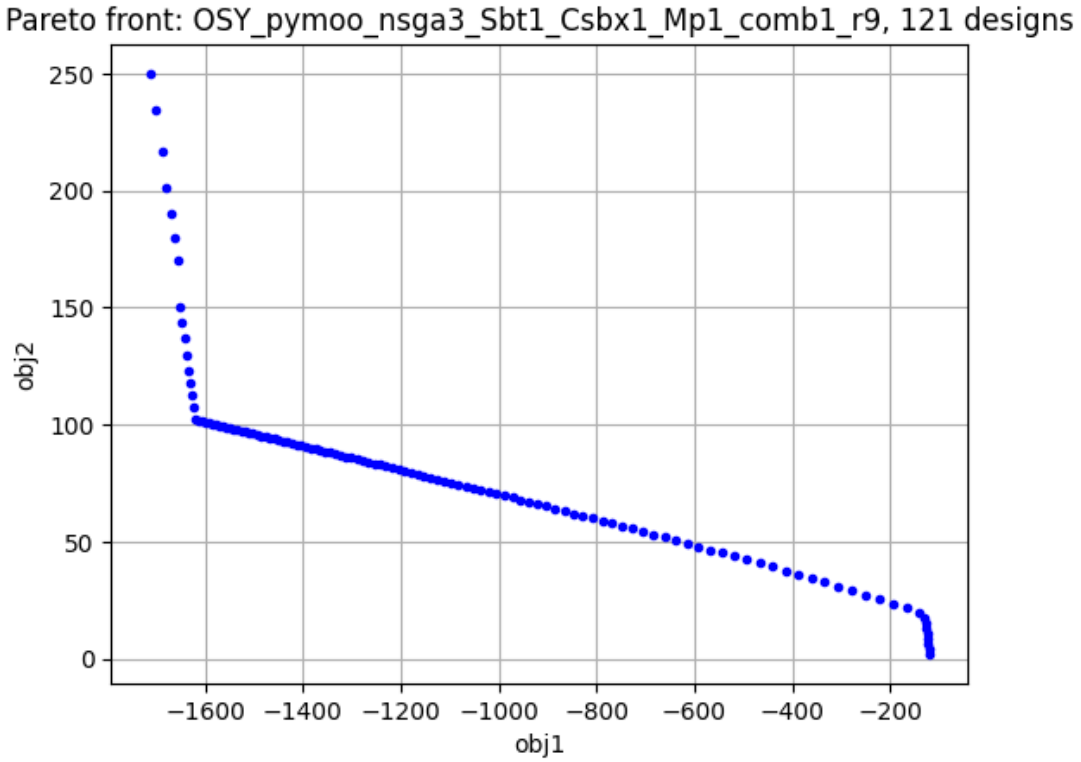
$$\begin{aligned} g_1(x) &= 2 - x_1 - x_2 \leq 0, \\ g_2(x) &= x_1 + x_2 - 6 \leq 0 \\ g_3(x) &= x_2 - x_1 - 2 \leq 0 \\ g_4(x) &= x_1 - 3x_2 - 2 \leq 0 \\ g_5(x) &= (x_3 - 3)^2 + x_4 - 4 \\ g_6(x) &= 4 - (x_5 - 3)^2 - x_6 \leq 0 \end{aligned} \quad \textbf{(16)}$$

6.4.1. Grafovi Pareto fronte

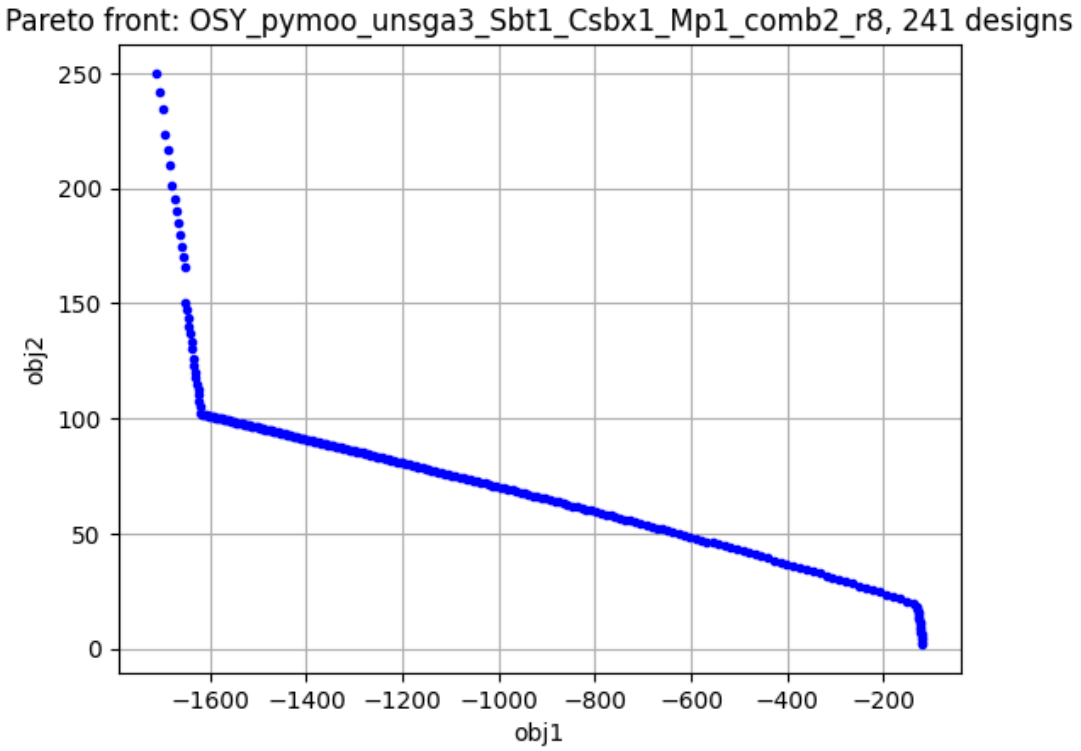
Neki odabrani grafovi prikazani su niže. Algoritam OMOPSO ponaša se neočekivano i daje preko 8000 projekata. Ostali algoritmi svi daju 300 rješenja osim Pymoo algoritama s referentnim pravcima: NSGA-III i U-NSGA-III. Naime, oni daju broj rješenja okviran broju referentnih pravaca.



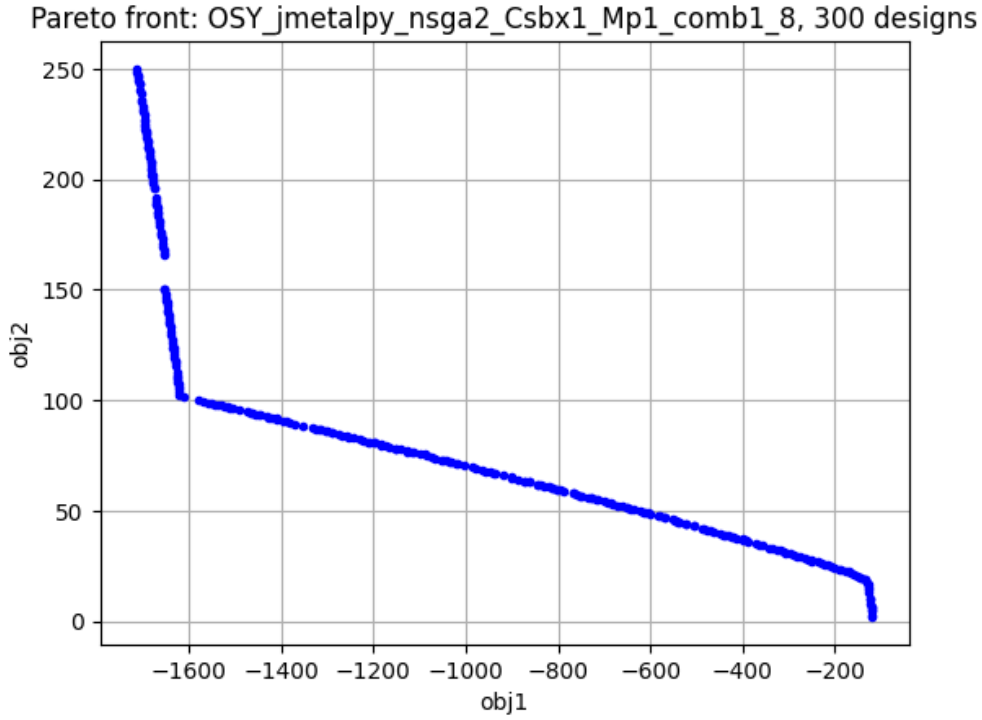
Slika 39. Slučaj 1.



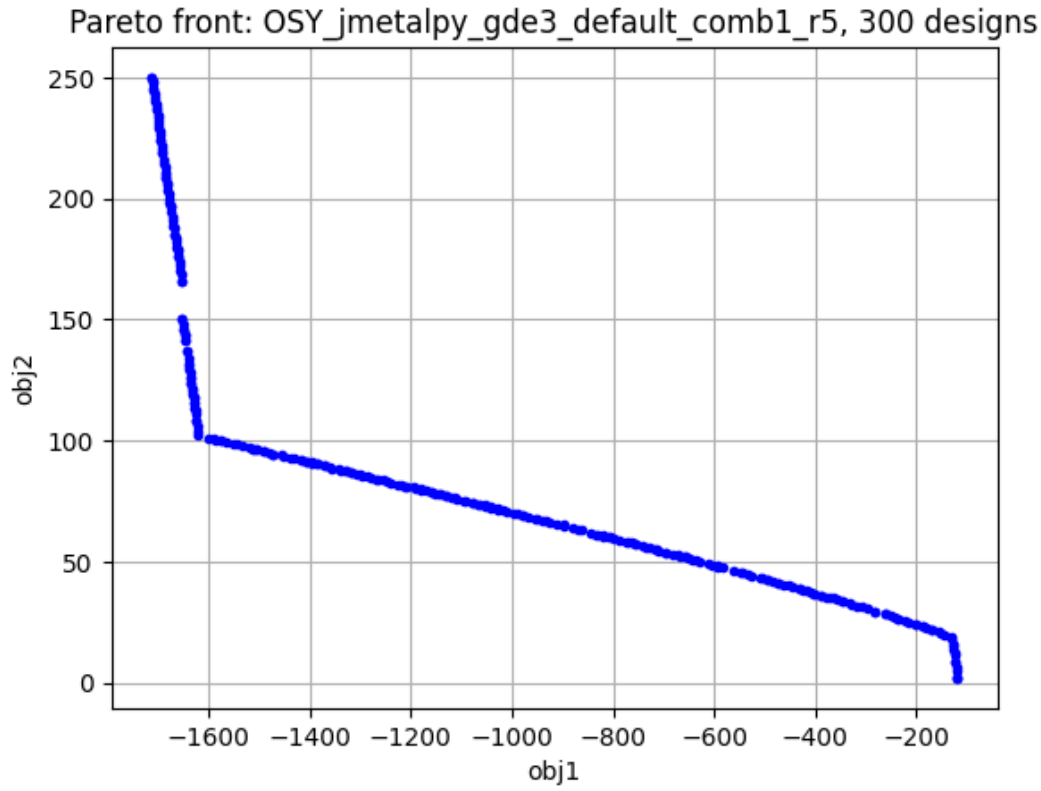
Slika 40. Slučaj 4.



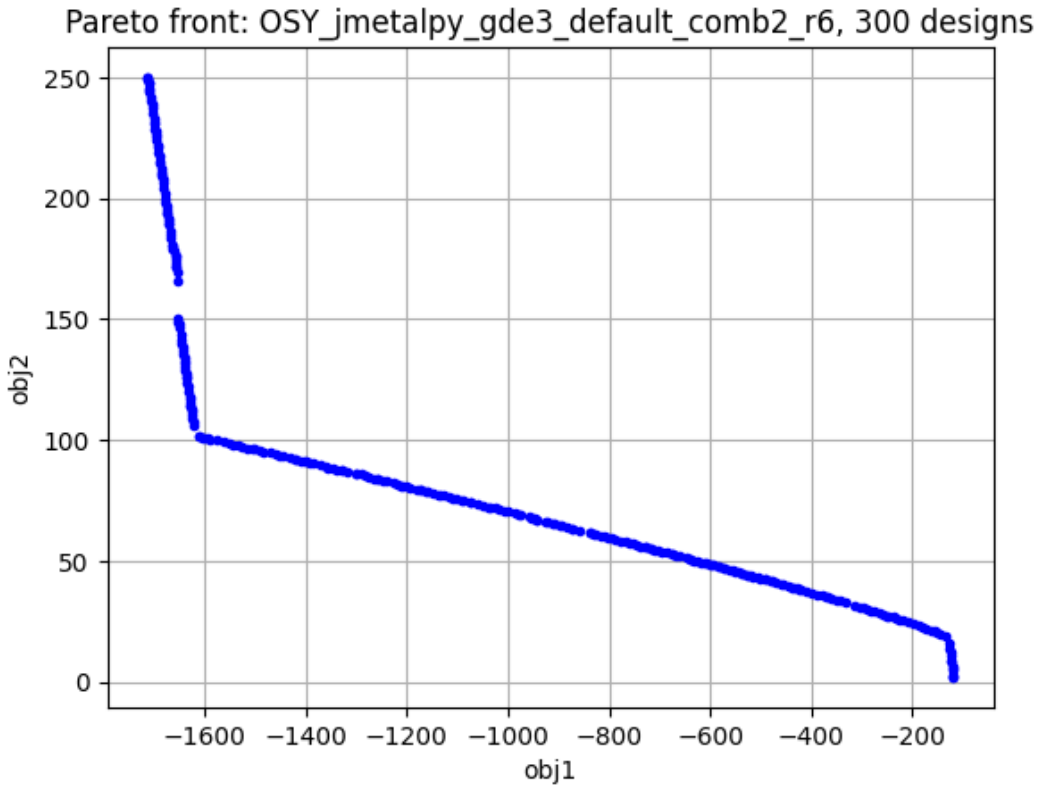
Slika 41. Slučaj 9.



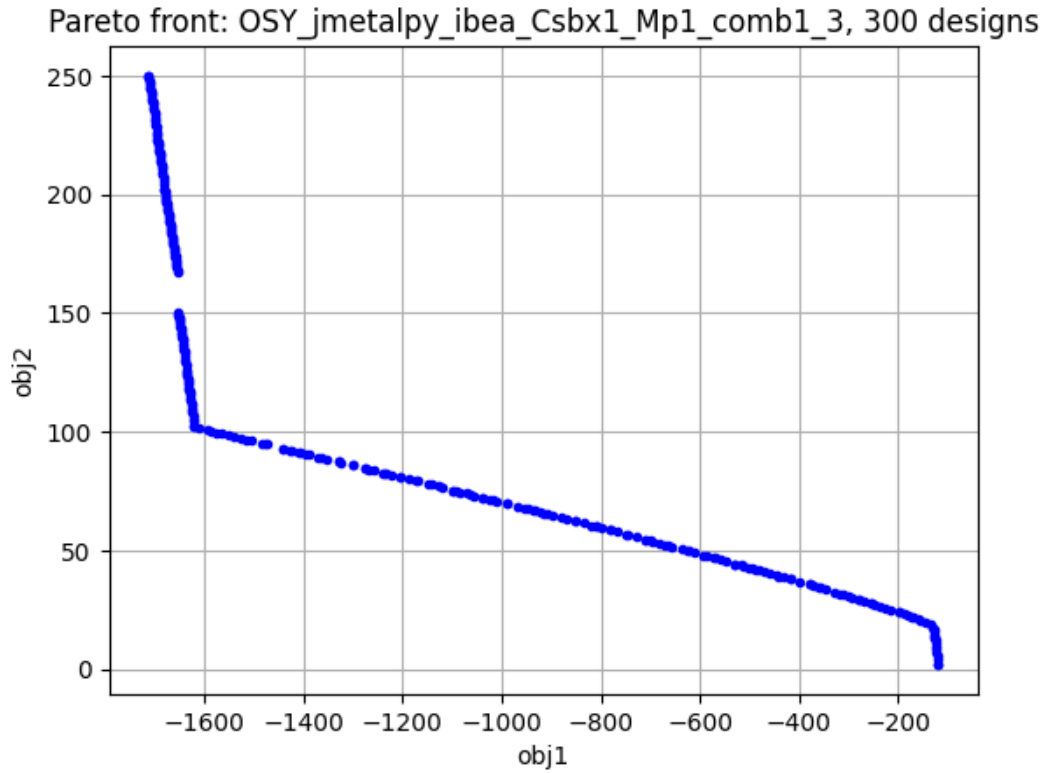
Slika 42. Slučaj 10.



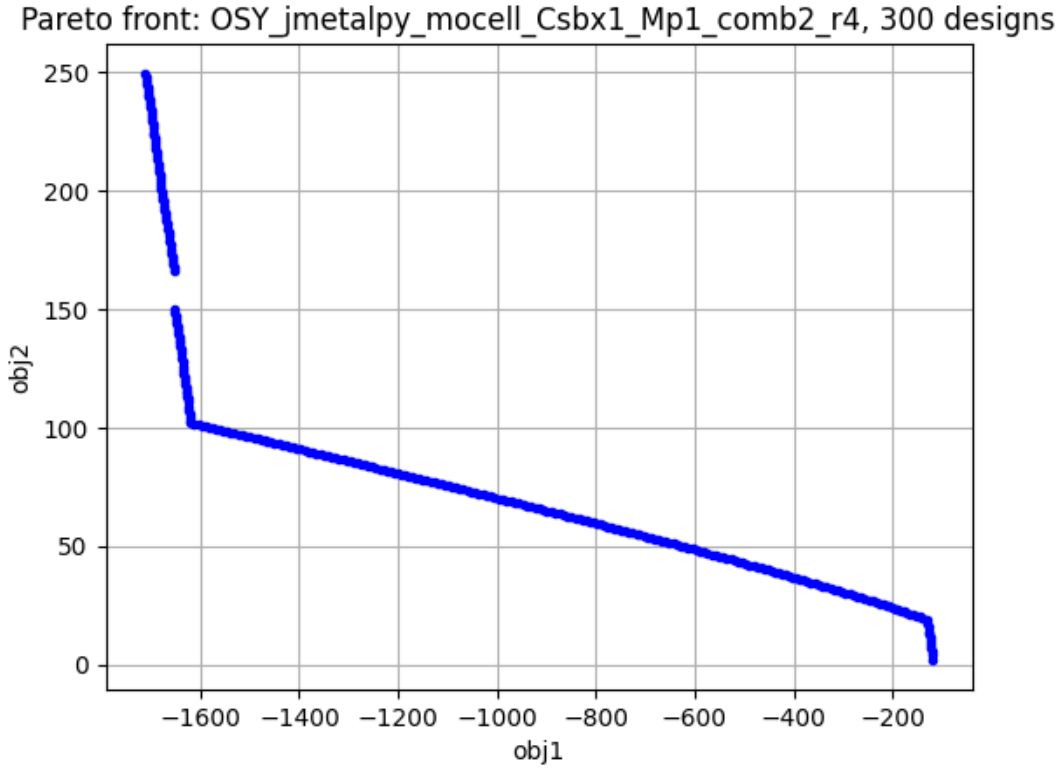
Slika 43. Slučaj 16.



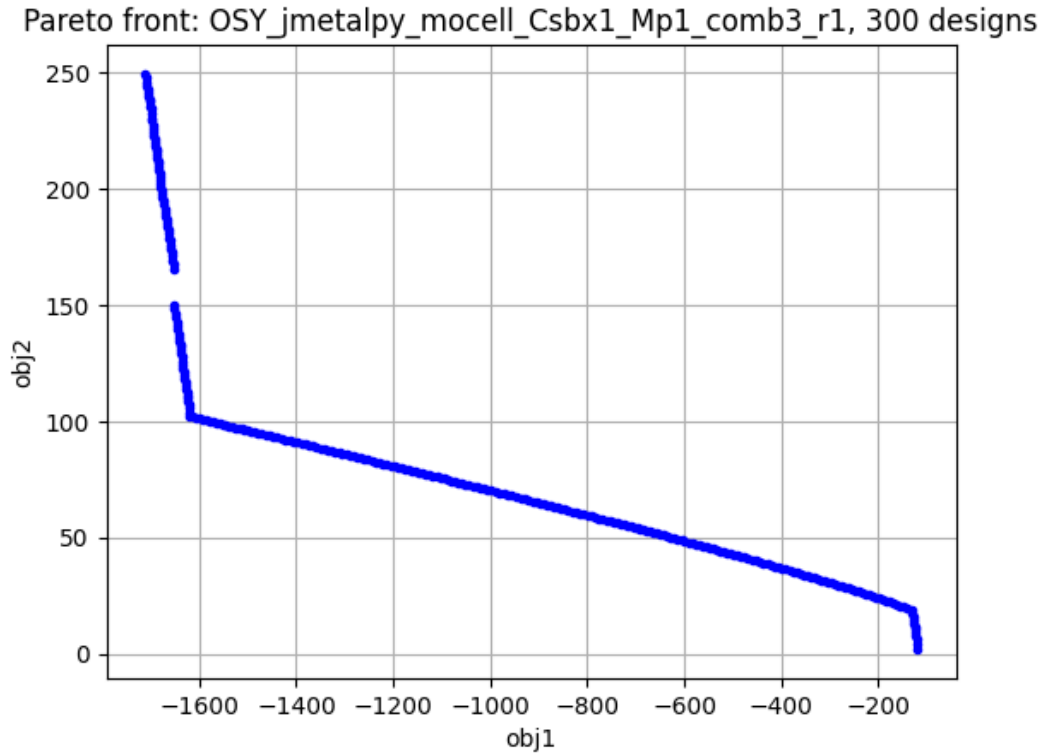
Slika 44. Slučaj 17.



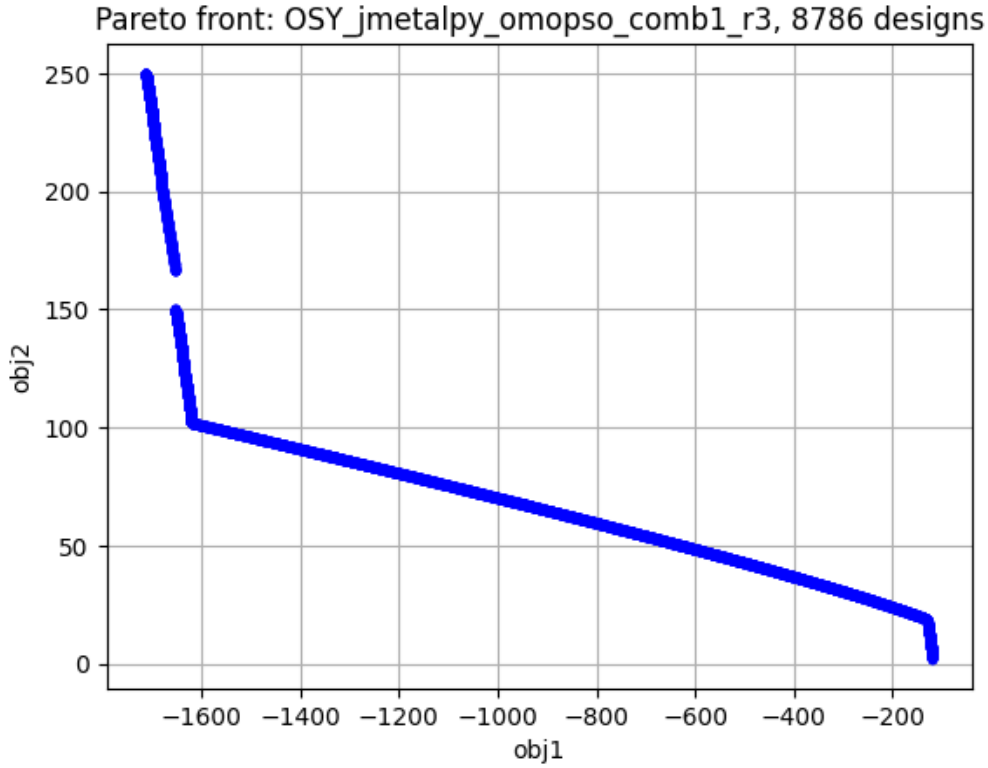
Slika 45. Slučaj 12.



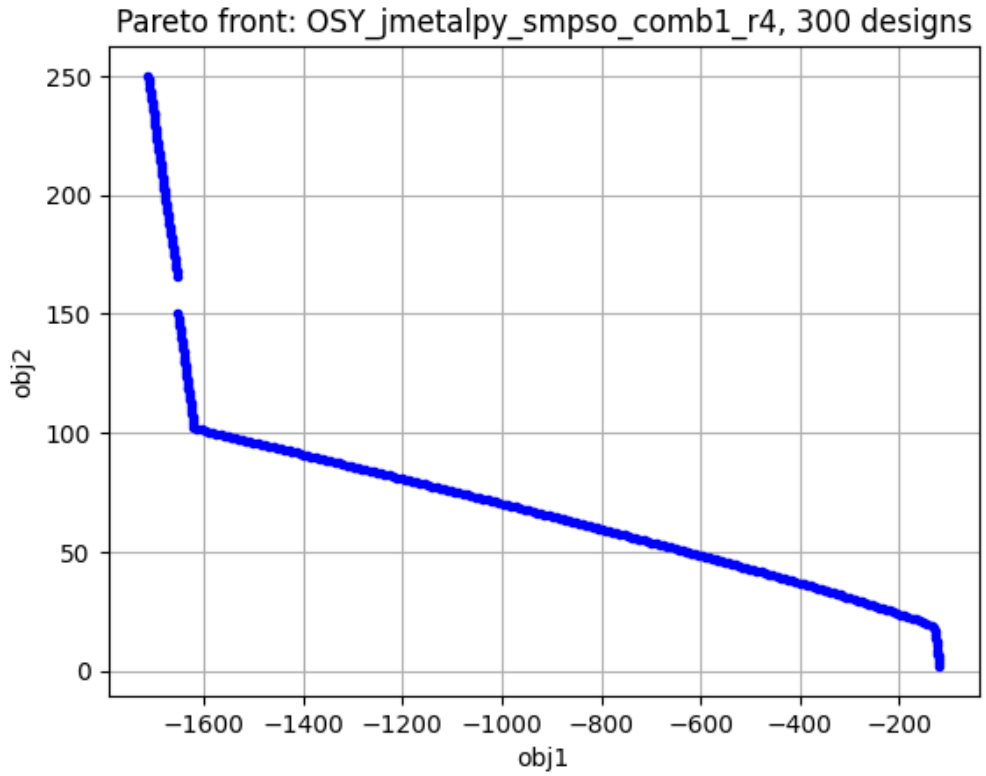
Slika 46. Slučaj 14.



Slika 47. Slučaj 15.



Slika 48. Slučaj 18.



Slika 49. Slučaj 22.

7. ZAKLJUČAK

U diplomskom radu provedena je usporedba kvalitete rezultata optimizacije višeciljnog problema s ograničenjima ostvarenih različitim optimizacijskim algoritmima iz dviju Python biblioteka. Usporedba je provedena i na primjeru okvira jednostavne brodske konstrukciju s tri palube i tri tanka ispod najdonje palube, pri čemu je za analizu odziva korištena metoda pomaka.

Rezultat ovog rada je prvenstveno unaprijeđena biblioteka ***moobench*** kojom se može služiti za postavljanje optimizacijskog problema i koja je dostupna na GitHub-u. Pomoću nje moguće je paralelizirati izvođenje više optimizacija odjednom, jednostavnije i brže definirati problem i postaviti optimizaciju, i provesti automatizirano velik broj optimizacija kreirajući pritom grafičke prikaze dobivenih Pareto fronta u odnosu na referetne i izračunavati mjere kvalitete. Rezultati analize bilježe se u obliku izlaznih datoteka što omogućuje naknadni pregled i obradu rezultata, pa i nekim drugim alatima. U okviru ovog rada su implementirana su sučelja za Python biblioteke Pymoo i jMetalPy.

Drugi rezultat ovog rada je sama usporedba rezultata i algoritama biblitoteka Pymoo i jMetalPy na složenijem primjeru optimizacijskog problema okvira jednostavne brodske konstrukciju. Provedena je i optimizacija istim algoritmima s istim postavkama na standardnom primjeru OSY zadanom preko moobench biblioteke i dani su prikazi Pareto fronti. Za primjer okvira dan je pregled mjera kvalitete u obliku *box-and-whiskers* grafova za jednostavnu brodsku konstrukciju okvira, kao i usporedna tablica s vremenima izvršavanja i brojem prekršenih ograničenja.

U okviru diplomskog rada utvrđeni su i neki mogući daljnji koraci na razvoju biblioteke moobench kao i primjeni iste za usporedbu rezultata različitih optimizacijskih metoda:

- proširenje moobench biblioteke sučeljima prema drugim optimizacijskim bibliotekama,
- uspostavljanje usporedbe utjecaja referentnih točaka na primjeru brodske konstrukcije,
- uspostava i provjera drugih složenijih analitičkih modela poput brodskog okvira,
- ostvarivanje novih funkcionalnosti moobench biblioteke poput automatske analize dobivenih mjera kvalitete Pareto fronte.

LITERATURA

[1] Podloge za nastavu iz kolegija „Višekriterijsko projektiranje i optimizacija“, izborni kolegij diplomskog studij strojarstva, Fakultet strojarstva i brodogradnje, Zagreb (dostupno na e-kolegiju Višekriterijsko projektiranje i optimizacija://e-ucenje.fsb.hr/)

[2] Prebeg P., Višekriterijsko projektiranje složenih tankostijenih konstrukcija, doktorska disertacija, Zagreb, 2011.

[3] Blank J., Deb K., Pymoo: Multi-objective Optimization in Python, IEEE Access, 2020., str. 89497. -89509., https://ieeexplore.ieee.org/document/9078759

[4] Blank J. i Deb K., Službena stranica programskog okvira Pymoo: https://pymoo.org/

[5] Hidalgo A.B., Nebro A. J. i dr., „jMetalPy: A Python framework for multi-objectie optimization with metaheuristics“, Swarm and evolutionary computation, ISSN: 2210-6502,2019. g.

[6] Audet C., Bigeon J., i dr.: Performance indicators in multiobjective optimization, European journal of operational research, 2021. g., str. 397. - 422.

[7] Hidalgo A.B., Nebro A.J., Službena stranica dokumentacije programskog okvira jMetalPy autora, i drugih: https://jmetal.github.io/jMetalPy/

[8] Skupina autora, Inženjerski priručnik IP1, Školska knjiga, Zagreb, 1996.

[9] Podloge za nastavu iz kolegija Čvrstoća broda, Preddiplomski studij brodogradnje, Fakultet strojarstva i brodogradnje, Zagreb (dostupno na e-kolegiju Čvrstoća https://e-ucenje.fsb.hr/)

[10] Stranica s rješenjima za razne sustave: https://www.linsgroup.com/MECHANICAL_DESIGN/Beam

PRILOZI

I. CD-R disk