OBRAVNAVANJE OMEJITEV V VEČKRITERIJSKI OPTIMIZACIJI

EVA ERZIN

Fakulteta za matematiko in fiziko
Univerza v Ljubljani

68W25, 78M32, 78M50, 90C29, 90C59

Ta članek obravnava večkriterijske optimizacijske probleme z omejitvami. Dober pristop k reševanju le-teh so genetski algoritmi, ki so najprej opisani v splošnem, nato pa sta dva izmed njih, NSGA-II in MOEAD opisana bolj podrobno. Podan je pregled obstoječih načinov obravnavanja omejitev v večkriterijski optimizaciji, s katerimi je mogoče genetske algoritme za večkriterijsko optimizacijo prilagoditi tako, da obravnavajo tudi probleme z omejitvami. Predstavljena sta dva testna večkriterijska optimizacijska problema z omejitvami in rezultati na njima izvedenih preizkusov prej predstavljenih algoritmov ter dveh izmed načinov obravnavanja omejitev.

CONSTRAINT HANDLING IN MULTIOBJECTIVE OPTIMIZATION

This article deals with constrained multiobjective optimization problems. A good approach to solving them is using genetic algorithms that this article first describes in general terms and then explains two of them, NSGA-II and MOEAD more thoroughly. Some constraint handling methods that allow for the existing multiobjective algorithms to be adapted for constrained multiobjective optimization are presented. Finally, two multiobjective constrained test problems are presented along with the results of testing the aforementioned genetic algorithms and constraint handling techniques on them.

1. Uvod

Večkriterijska optimizacija je v najrazličnejših oblikah prisotna od nekdaj, od odločanja za najboljši avto po najugodnejši ceni do načrtovanja konzole, za katero bo porabljenega čim manj materiala in bo prenesla čim večjo obtežitev. Taki večkriterijski problemi zaradi nasprotujočih si kriterijev ponavadi nimajo ene same rešitve. Še več, le-teh je lahko veliko, za eno samo pa se lahko odločimo na podlagi osebne preference šele, ko jih poznamo čim več. Ena od strategij za spopadanje s temi problemi je uporaba genetskih algoritmov, ki posnemajo naravno selekcijo in genetiko. Ti so pridobili na popularnosti, saj v kratkem času lahko vrnejo raznoliko množico rešitev.

Medtem ko obstaja nemalo genetskih algoritmov za večkriterijsko optimizacijo, je takih, ki zmorejo upoštevati še omejitve, malo. Probleme lahko z različnimi metodami prevedemo na enokriterijske optimizacijske probleme, za katere že obstajajo algoritmi, vendar pa s tem izgubimo raznolikost optimalnih rešitev in možnost izbire na podlagi osebne preference, ki jo ponujajo genetski algoritmi za večkriterijsko optimizacijo.

V tem članku je predstavljena večkriterijska optimizacija z omejitvami. Opisani so omejeni večkriterijski problemi in njihove omejitve, nato pa predstavljeni še genetski algoritmi in različice le-teh, ki so prilagojene za večkriterijsko optimizacijo, na koncu pa je podan še pregled nekaj možnosti za obravnavo omejitev pri večkriterijski optimizaciji. Dve izmed njih sta eksperimentalno ovrednoteni z vključitvijo v algoritma NSGA-II in MOEA/D.

2. Večkriterijska optimizacija z omejitvami

Problem večkriterijske optimizacije z omejitvami je definiran kot iskanje vektorja spremenljivk $x = (x_1, x_2, \ldots, x_n)$, ki optimira kriterijsko preslikavo

$$f : P \to K \subseteq \mathbb{R}^k,$$

$$f(x) = (f_1(x), f_2(x), \ldots, f_k(x)),$$

kar pomeni, da išče minimume ali maksimume posameznih komponent f_i, ki jih imenujemo *kriterijske funkcije*, hkrati pa zadošča omejitvam

$$g_i(x) < 0\,;\;\; i = 1, \ldots, q,$$
$$h_j(x) = 0\,;\;\; j = 1, \ldots, p.$$

Množica P se imenuje *prostor spremenljivk*, množica K pa *prostor kriterijev*. Elemente P imenujemo *rešitve* ali *osebki*.

Hitro opazimo, da lahko maksimizacijo katerekoli komponente prevedemo na minimizacijo tako, da namesto maksimuma funkcije f_i iščemo minimum funkcije $-f_i$. V nadaljevanju bomo zato predpostavljali, da kriterijsko preslikavo minimiziramo po vseh komponentah.

Definicija 1. Element $x = (x_1, x_2, \ldots, x_n) \in P$ je *dopustna* rešitev optimizacijskega problema natanko tedaj, ko zadošča vsem neenakostim g_i, $i = 1, \ldots, q$ in enakostim h_j, $j = 1, \ldots, p$.

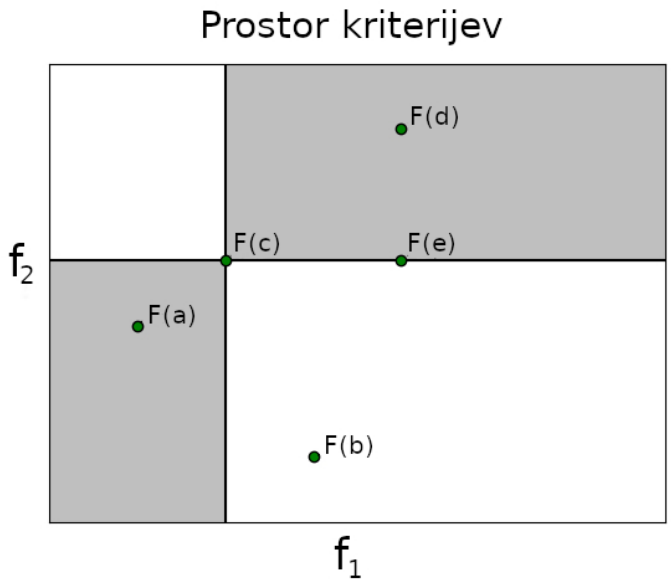
Vprašanje, ki se porodi ob definiciji omejenega večkriterijskega problema, je, kaj bi lahko definirali kot najboljšo rešitev takega problema in ali je takšna le ena. Kriterijske funkcije, ki jih minimiziramo, si med seboj pogosto nasprotujejo. To pomeni, da ob izboljševanju vrednosti ene poslabšujemo vrednost druge. Ker nekatere rešitve med seboj zato niso primerljive (ena ima morda v nekaterih kriterijskih funkcijah večje vrednosti kot druga, v drugih pa manjše), ne moremo reči, da vedno obstaja ena sama najboljša rešitev. Videli bomo, da v splošnem obstaja cela množica optimalnih rešitev, ki so med seboj neprimerljive. Da bi jo lahko dobro opisali, moramo najprej definirati relacijo dominantnosti.

Definicija 2. Naj bosta $x = (x_1, x_2, \ldots, x_n)$ in $y = (y_1, y_2, \ldots, y_n)$ rešitvi problema optimizacije s kriterijsko preslikavo $F(x) = (f_1(x), f_2(x), \ldots, f_k(x))$. Rešitev x *dominira* nad rešitvijo y (označimo z $x \preceq y$), če velja

1. $f_i(x) \leq f_i(y)$ za vse $i = 1, \ldots, k$,

2. Obstaja $i \in \{1, \ldots, k\}$, za katerega velja $f_i(x) < f_i(y)$.

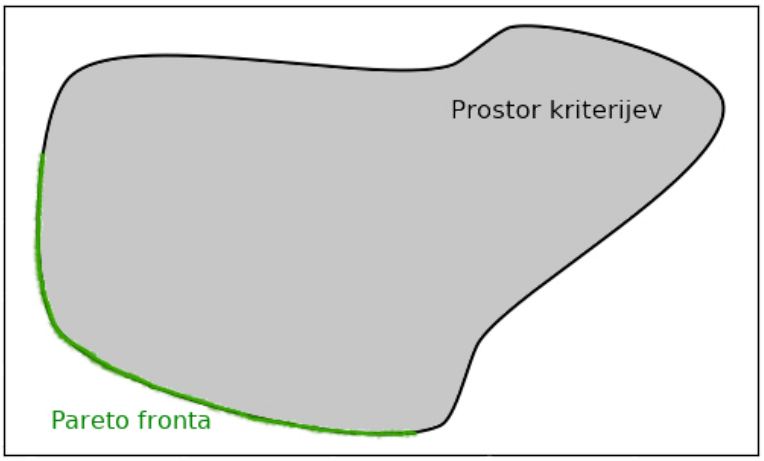
Slika 1 prikazuje dominiranost rešitev optimizacijskega problema, kjer minimiziramo dve kriterijski funkciji.



Slika 1. Prikaz dominiranosti rešitev na primeru mimimizacije kriterijev f_1 in f_2. Rešitev c dominira nad rešitvama e in d, nad njo pa dominira rešitev a. Rešitev b ni primerljiva z rešitvama a in c, saj je njena vrednost v f_2 manjša, v f_1 pa večja.

Definicija 3. *Pareto optimalna množica* je množica vseh rešitev v prostoru spremenljivk, nad katerimi ne dominira nobena druga rešitev.

Definicija 4. *Pareto optimalna fronta* (ali krajše *Pareto fronta*) je slika Pareto optimalne množice v prostoru kriterijev.



Slika 2. Primer Pareto fronte.

Na sliki 2 je prikazan primer Pareto optimalne fronte minimizacijskega optimizacijskega problema z dvema kriterijskima funkcijama. Množica dopustnih rešitev je označena s sivo barvo, Pareto optimalna fronta pa z zeleno barvo.

2.1 Pristopi k reševanju večkriterijskih optimizacijskih problemov

Reševanja večkriterijskih optimizacijskih problemov se lahko lotimo na dva načina. Ločimo ju glede na to, kdaj se želimo odločiti za le eno izmed rešitev.

Če lahko kriterije že vnaprej razvrstimo po pomembnosti, se odločimo za *prednostni pristop*, kjer poskusimo s pomočjo tega, da vemo, kateri kriteriji so za nas pomembnejši, večkriterijski optimizacijski problem prevesti na enokriterijskega, ki upošteva naše želje. Tako določimo, v kateri smeri bomo iskali optimum preslikave F. To olajša računanje, vendar pa s tem morda izgubimo možnost odkrivanja več rešitev iz Pareto fronte, poleg tega pa moramo, če želimo različne rešitve, večkrat pognati algoritem. Oglejmo si primer takega pristopa – metodo uteženih vsot.

Imejmo večkriterijski optimizacijski problem, kot je opisan zgoraj, kjer iščemo rešitve, ki minimizirajo kriterijske funkcije preslikave $F = (f_1, \ldots, f_k)$. Vsaki izmed kriterijskih funkcij f_i, glede na to, koliko nam je pomembna, priredimo utež $w_i \geq 0$. Večkriterijski optimizacijski problem nato v enokriterijskega prevedemo tako, da definiramo novo kriterijsko funkcijo

$$f(x) = \sum_{i=1}^{k} w_i f_i(x)$$

in iščemo njen minimum.

Vsaka rešitev x takega problema je element Pareto optimalne fronte originalnega večkriterijskega problema. Če ne bi bila, bi morala obstajati taka druga rešitev y, da bi veljalo $f(y) \preceq f(x)$, torej bi obstajal indeks $i \in 1, 2, \ldots, k$, za katerega bi bilo res $f_i(y) < f_i(x)$, za ostale $j \in 1, 2, \ldots, i-1, i+1, \ldots, k$ pa bi veljalo $f_j(y) \leq f_j(x)$. Zato bi veljalo

$$f(y) = \sum_{i=1}^{k} f_i(y) w_i < \sum_{i=1}^{k} f_i(x) w_i = f(x),$$

kar bi pomenilo, da rešitev x ni zares minimum novega optimizacijskega problema.

Drug pristop pa je *idealni pristop*, pri katerem najprej poiščemo množico med seboj neprimerljivih nedominiranih rešitev, nato pa se na podlagi dobljenega odločimo za nam najljubšo. Prednost tega pristopa je, da si lahko najprej, že po enem zagonu algoritma, ogledamo več rešitev iz Pareto optimalne fronte. Izmed teh si lahko nato izberemo eno samo. Slabost idealnega pristopa pa je, da za iskanje več rešitev hkrati porabimo več računske moči.

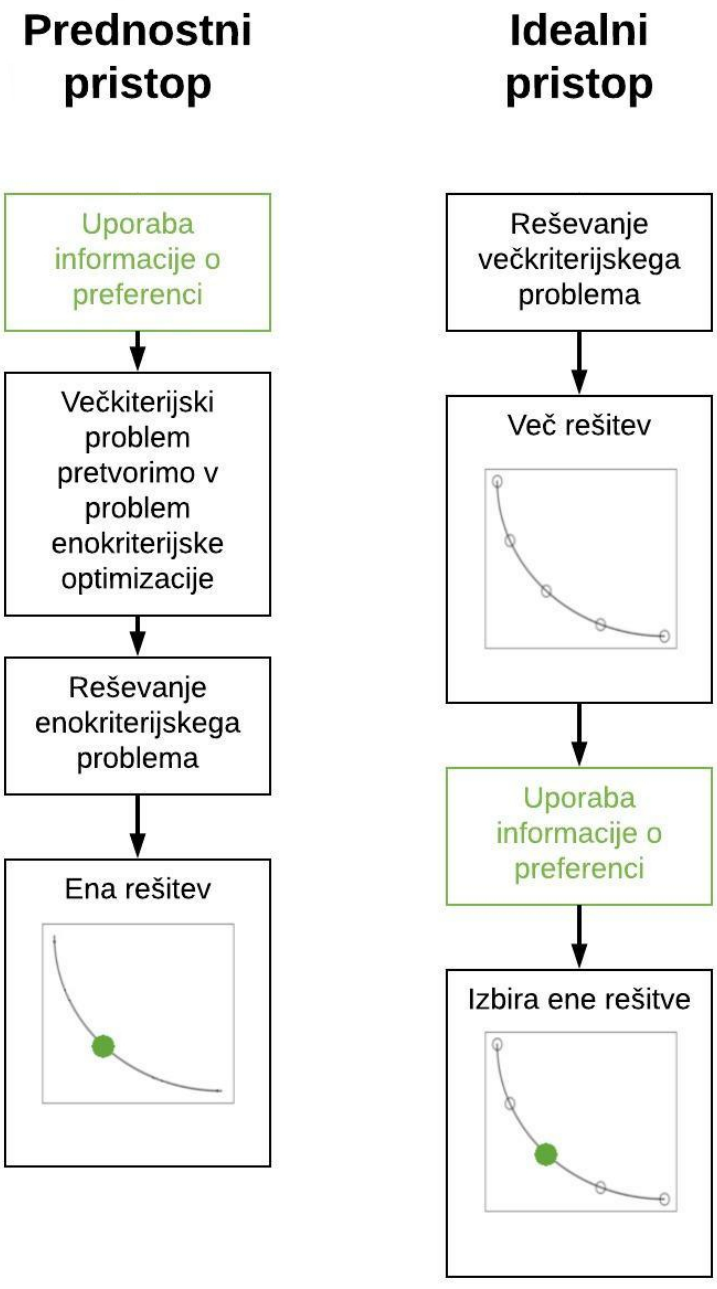
Primerjava obeh pristopov je predstavljena na sliki 3.

3. Genetski algoritmi

Pogost pristop k reševanju računsko zahtevnih optimizacijskih problemov so metahevristični algoritmi. To so nedeterministični optimizacijski algoritmi, ki sicer ne zagotavljajo, da bomo v končnem času našli globalno najboljšo rešitev, vendar pa bomo najverjetneje v zadovoljivem času našli tako, ki bo "dovolj dobra". Uporaba teh algoritmov je smiselna, ko je prostor spremenljivk prevelik, da bi pregledali celega. To lahko pomeni, da problemi ne ustrezajo zahtevam za uporabo deterministič nih algoritmov – niso linearni, zvezni itd., ali pa je na voljo le omejena računska moč.

Ena od strategij, ki jo pogosto najdemo v metahevristič nih algoritmih, je posnemanje evolucije. Poseben primer t. i. evolucijskih algoritmov, ki uporabljajo to strategijo, pa so genetski algoritmi.

V šestinšestdesetih letih prejšnjega stoletja jih je razvil profesor psihologije, računalništva in elektrotehnike z Univerze v Michiganu, John Holland. Njegov algoritem je simuliral Darwinov princip preživetja najuspešnejših in vseboval principe selekcije, križanja in mutacije. Po letih sodelovanja s študenti je leta 1975 izdal knjigo [4], v kateri je kot prvi predstavil genetske algoritme kot abstrakcijo biološke evolucije in postavil teoretične temelje za računsko evolucijo.



Slika 3. Pristopa k reševanju večkriterijskih optimizacijskih problemov.

V prihodnjih desetletjih je število raziskav na tem področju skupaj z razvojem tehnologije hitro naraščalo. Znanstveniki so odkrivali omejitve tradicionalnih optimizacijskih metod (npr. analitičnih metod, izčrpnega in naključnega preiskovanja prostora) pri reševanju kompleksnih problemov. Genetski algoritmi pa ne potrebujejo dodatnih informacij o problemu, kot so na primer odvodi, in se ne zanašajo na zveznost prostora kriterijev. Zato so se izkazali za uporabne pri iskanju rešitev optimizacijskih problemov, ki jim ostale metode niso bile kos.

Genetski algoritmi v splošnem delujejo po načelu naravne selekcije na populaciji, sestavljeni iz kandidatov za rešitve, imenovanih tudi *osebki*, ki jo razvijajo proti boljšim rešitvam. Običajen način predstavljanja osebkov je z nizi bitov, sprejemljivi pa so tudi nizi znakov, permutacije, vektorji naravnih ali realnih števil itd. Evolucijski algoritmi so iterativni. Ponavadi začnemo s populacijo naključno generiranih osebkov, ki jih nato v vsaki iteraciji med seboj križamo in jih mutiramo, nato pa iz množice starih in novonastalih osebkov izberemo novo populacijo. Posamezni iteraciji algoritma rečemo tudi *generacija*.

Algoritem 1 Splošen genetski algoritem

Vhod: Kriterijska preslikava $F = (f_1, f_2, \ldots, f_k)$ in omejitve $o = (g_1, \ldots, g_q, h_1, \ldots, h_p)$
Izhod: Končna populacija $\mathcal{P}_t$, ki aproksimira Pareto fronto danega optimizacijskega problema

function GENETSKI(F, o)
  Generiraj in ovrednoti začetno populacijo staršev $\mathcal{P}_0$.
  Nastavi $t = 0$.
  while $\neg$ ustavitveni pogoj **do**
    Z uporabo selekcije izberi nekaj osebkov iz populacije $\mathcal{P}_t$ v populacijo za reprodukcijo $\mathcal{R}_t$.
    S križanjem in mutacijo iz osebkov v $\mathcal{R}_t$ ustvari populacijo potomcev $\mathcal{Q}_{t+1}$.
    Ovrednoti osebke iz $\mathcal{Q}_{t+1}$.
    Pripravi novo prazno populacijo $\mathcal{P}_{t+1}$.
    Z uporabo selekcije izberi najboljše osebke iz $\mathcal{P}_t \cup \mathcal{Q}_{t+1}$ in z njimi napolni $\mathcal{P}_{t+1}$.
    $t = t + 1$.
  end while
  return $\mathcal{P}_t$
end function

3.1 Selekcija

Selekcija je postopek izbire osebkov iz populacije za križanje. Izbrane osebke imenujemo *starši*, saj bodo kasneje iz njih nastajali novi osebki. Ker stremimo k optimalnim populacijam, želimo za starše izbrati čim boljše osebke, vseeno pa želimo čim bolje preiskati prostor in se ne ustaviti v lokalnih minimumih, zato z majhno verjetnostjo za nadaljnje iskanje optimumov izberemo tudi slabše osebke.

Obstaja mnogo različnih načinov selekcije, do razlik pride že pri algoritmih za enokriterijsko in večkriterijsko optimizacijo. Pri enokriterijski optimizaciji kriterijska funkcija podaja popolno urejenost osebkov v populaciji, zato za nadaljnje preiskovanje brez težav izberemo najboljše. Oglejmo si dva pogosto uporabljena načina selekcije pri enokriterijski optimizaciji.

Turnirska selekcija je preprost način izbiranja boljših osebkov iz populacije s sledečim postopkom:

1. Naključno izberemo dva osebka x_i in x_j iz populacije.

2. Za križanje izberemo osebek z manjšo vrednostjo kriterijske funkcije.

Ruletna selekcija pa poteka na sledeči način:

1. Osebke razvrstimo padajoče glede na vrednosti kriterijske funkcije tako, da je prvi osebek najslabši glede na kriterijsko funkcijo, zadnji pa najboljši. Tako ima x_1 največjo vrednost, x_m pa najmanjšo, kjer je m število osebkov v populaciji.

2. Izračunamo vsoto vrednosti kriterijskih funkcij za celotno populacijo:

$$T = \sum_{i=1}^{m} f(x_i).$$

3. Za vsak osebek izračunamo verjetnost izbire

$$p_i = \frac{f(x_i)}{T}.$$

4. Za vsak osebek izračunamo njegovo kumulativno verjetnost

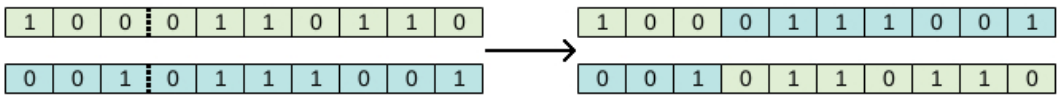
$$q_i = \sum_{j=1}^{i} p_i.$$

5. Sledi "vrtenje rulete". To storimo tolikokrat, kot je velika populacija. Izberemo naključno realno število s z intervala $[0, 1]$ in za križanje izberemo i-ti osebek, če je $q_{i-1} < s \leq q_i$.

Pri večkriterijski optimizaciji so algoritmi za selekcijo večinoma enaki kot tisti za enokriterijsko optimizacijo. Razlikujejo se le v tem, da osebkov ne moremo neposredno primerjati med seboj, zato jih primerjamo glede na relacijo dominiranosti. Če en osebek dominira nad drugim, tega proglasimo za boljšega, sicer naključno izberemo enega.

3.2 Križanje

Po izbiri staršev sledi naslednji korak, križanje, s katerim ustvarjamo nove osebke, ki od vsakega od staršev podedujejo nekaj lastnosti. Ta proces je v veliki meri odvisen od tega, kako so osebki v populacij predstavljeni. Najlažje si ga predstavljamo na primeru križanja osebkov predstavljenih z nizom ničel in enic oziroma bitov dolžine n, kot je prikazano na sliki 4. Najpreprostejši način križanja je eno-točkovno križanje, kjer naključno izberemo naravno število m med 0 in $n-1$, ki bo predstavljalo mesto križanja. Nato tvorimo dva potomca, prvi bo imel prvih m bitov enakih kot prvi izmed staršev in ostalih $n-m$ enakih kot drugi, drugi potomec pa ravno obratno.



Slika 4. Križanje osebkov predstavljenih z binarnimi nizi.

Oglejmo si še primer *aritmetičnega križanja* križanja osebkov, predstavljenih z vektorjem realnih števil, ki poteka po sledečem postopku:.

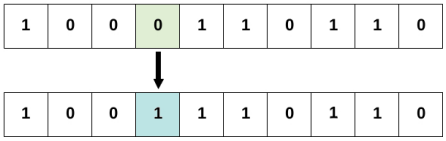
1. Izberemo naključno realno število q med 0 in 1.

2. Potomca p_1 in p_2 nato iz staršev s_1 in s_2 dobimo s sledečim postopkom:

$$p_1 = qs_1 + (1-q)s_2,$$
$$p_2 = (1-q)s_1 + qs_2.$$

3.3 Mutacija

Mutacija je naslednji izmed genetskih operatorjev, ki omogoča boljše preiskovanje prostora. Nekaj novonastalih osebkov je naključno izbranih in nato nekoliko spremenjenih.



Slika 5. Primer mutacije na enem bitu.

Oglejmo si, kako mutacija najpogosteje poteka za osebke predstavljene z binarnimi nizi.
Za vsak osebek izvedemo sledeči postopek:

1. Generiramo naključno število q med 0 in 1. Če velja $q > P_{mutacija}$, kjer je $P_{mutacija}$ vnaprej izbrano število, ki podaja verjetnost, da bo osebek izbran za mutacijo, ta osebek izberemo, sicer pa ne.

2. Za vsak bit izbranega osebka generiramo še eno naključno število in če je večje od vnaprej določenega, potem ta bit spremenimo iz 0 v 1 oz. obratno.

Primer mutacije na enem bitu je prikazan na sliki 5.
Pri številskih vektorjih sta primera mutacije prištevanje majhnih števil in množenje s faktorjem blizu 1.

3.4 Genetski algoritmi za večkriterijsko optimizacijo

Največja razlika pri prehodu iz enokriterijske v večkriterijsko optimizacijo je sortiranje osebkov, saj teh zaradi delne urejenosti relacije dominiranosti ne moremo vedno primerjati med sabo.

3.4.1 NSGA-II

Algoritem *NSGA-II* (Elitist Non-Dominated Sorting Genetic Algorithm) so leta 2000 razvili Deb in sodelavci in zanj že predlagali način obravnavanja omejitev [3].

Problem sortiranja osebkov so Deb in sodelavci rešili s t. i. nedominiranim sortiranjem, katerega grafično prestavitev si lahko ogledamo na sliki 6. Pri tem osebke razvrstimo v fronte tako, da najprej v populaciji poiščemo vse nedominirane osebke in jim priredimo prvo fronto. Nato med ostalimi osebki spet poiščemo vse nedominirane in jim priredimo drugo fronto. Ta postopek ponavljamo, dokler niso vsi osebki razvrščeni v fronte.

Naiven algoritem bi za iskanje prve nedominirane fronte vsak osebek primerjal z vsemi ostalimi, da bi ugotovil ali kateri izmed njih dominira nad njim in porabil $\mathcal{O}(m^2k)$ časa, kjer je m število osebkov v populaciji, k pa število kriterijskih funkcij. V najslabšem primeru (ko je število osebkov, ki ne pripadajo prvi fronti velikostnega reda $\mathcal{O}(m)$) bi algoritem tudi za iskanje druge fronte porabil $\mathcal{O}(m^2k)$ časa. Enako velja za vse nadaljne fronte, kar pripelje do končne najslabše časovne zahtevnosti nedominiranega sortiranja $\mathcal{O}(m^3k)$.

Algoritem 2 NSGA-II

Vhod: Kriterijska preslikava $F = (f_1, f_2, \ldots, f_k)$, omejitve $o = (g_1, \ldots, g_q, h_1, \ldots, h_p)$ in število generacij $n \in \mathbb{N}$

Izhod: Končna populacija $\mathcal{Q}_n$, ki aproksimira Pareto fronto danega optimizacijskega problema

function NSGA2(f, o, n)
  Generiraj in ovrednoti začetno populacijo staršev $\mathcal{P}_0$.
  Pripravi prazno začetno generacijo potomcev $\mathcal{Q}_0$.
  $t = 0$.
  while $t \leq n$ **do**
    $t = t + 1$.
    Združi populaciji staršev in potomcev $\mathcal{R}_{t-1} = \mathcal{P}_{t-1} \cup \mathcal{Q}_{t-1}$.
    Nedominirano sortiraj osebke v $\mathcal{R}_{t-1}$ v fronte $\mathcal{F}_i, i = 1, 2, \ldots$
    Pripravi novo populacijo $\mathcal{P}_t$ in jo napolni s prvimi i frontami, ki gredo cele vanjo.
    Fronto $i + 1$ sortiraj z uporabo metrike nakopičenosti.
    $\mathcal{P}_t$ dopolni z najmanj nakopičenimi osebki iz $\mathcal{F}_{i+1}$.
    Uporabi turnirsko selekcijo, križanje in mutacijo za generiranje nove populacije potomcev $\mathcal{Q}_t$.
    Ovrednoti osebke iz generacije $\mathcal{Q}_t$.
  end while
  return $\mathcal{Q}_n$
end function

Ker je najslabša možna časovna zahtevnost algoritma določena z zahtevnostjo nedominiranega sortiranja (natančna razlaga je dostopna v članku [3]), so Deb in sodelavci razvili algoritem za *hitro nedominirano sortiranje*, ki deluje v $\mathcal{O}(m^2k)$ časa.

Za vsak osebek p poiščemo množico osebkov S_p nad katerimi dominira in število osebkov, ki dominirajo nad njim - *dominacijsko število* n_p. Prvo nedominirano fronto natanko določajo osebki, za katere velja $n_p = 0$. Drugo poiščemo tako, da za vsak osebek p iz prve nedominirane fronte vsem osebkom iz S_p za ena zmanjšamo dominacijsko število. Drugi fronti pripadajo osebki, katerih dominacijsko število se je v tem postopku zmanjšalo na nič. S ponavljanjem postopka smo našli vse fronte. Za določanje n_p in S_p, moramo vsak osebek primerjati z vsemi ostalimi, kar prinese časovno zahtevnost $\mathcal{O}(m^2k)$, pri samem razvrščanju v fronte pa opazimo, da ima vsak osebek dominacijsko število enako kvečjemu $m - 1$, torej mu le-to lahko zmanjšamo največ $(m - 1)$-krat, preden mu določimo fronto. Skupna časovna zahtevnost razvrščanja v fronte je tako $\mathcal{O}(m^2)$, kar pomeni skupno časovno zahtevnost hitrega nedominiranega sortiranja $\mathcal{O}(m^2k)$.

Ko so fronte določene, populacijo potomcev izberemo tako, da izberemo vse osebke iz prvih m front, ki gredo cele v populacijo, osebke iz fronte $m+1$ pa sortiramo glede na metriko nakopičenosti, ki jo bomo opisali v nadaljevanju. Ta zagotavlja, da imajo osebki, ki jih izberemo iz fronte, ki ne gre cela v populacijo, čim večji razpon. Najbolje oceni skrajne osebke, ostalim pa priredi vrednosti glede na oddaljenost od najbližjih sosedov. Za vsak kriterij $j = 1, \ldots, k$ osebke najprej naraščajoče uredi, nato pa za vsak osebek i izračuna razdaljo z upoštevanjem njemu (glede na to kriterijsko funkcijo) najbližjih osebkov a in b (tako, da je $f_j(a) \leq f_j(i) \leq f_j(b)$):

$$d_j(i) = \frac{f_j(b) - f_j(a)}{f_j^{max} - f_j^{min}},$$

kjer sta

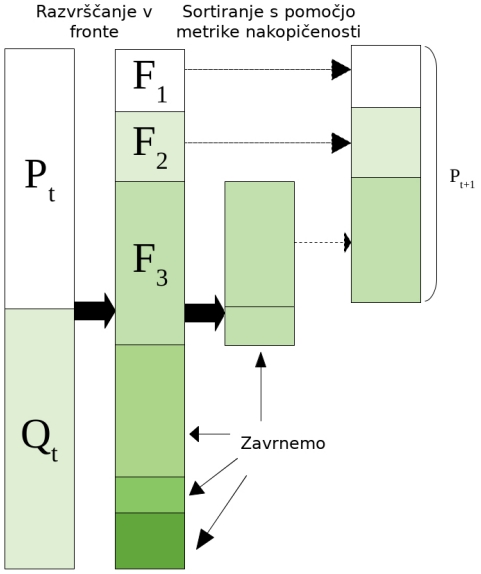
$$f_j^{max} = \max_{x \in \mathcal{F}_{m+1}} f_j(x)$$

in

$$f_j^{min} = \min_{x \in \mathcal{F}_{m+1}} f_j(x).$$

Metrika nakopičenosti za osebek i je vsota teh razdalj za vse kriterije

$$c(i) = \sum_{j=1}^{k} d_j(i).$$



Slika 6. Grafični prikaz nedominiranega sortiranja. Oznake so enake kot v psevdokodi algoritma 2.

Algoritem 3 Računanje metrike nakopičenosti

Vhod: Množica osebkov I
Izhod: Slovar razdalj osebkov R

function DOLOCI_NAKOPICENOST(I)
  $l = |I|$
  pripravi tabelo $R = [0$ **for** $i \in \{2, 3, \ldots l-1\}]$.
  for all kriterij m **do**
    $I = \text{sort}(I, m)$ ▷ Uredi I glede na kriterijsko funkcijo m
    $R[1] = R[l] = \infty$
    for all $i \in \{2, 3, \ldots l-1\}$ **do**
      $R[i] = R[i] + (I[i+1]_m - I[i-1]_m)/(f_m^{max} - f_m^{min})$ ▷ $I[i]_m$ označuje vrednost m-te kriterijske funkcije osebka i.
    end for
  end for
  return R
end function

Algoritem 4 Hitro nedominirano sortiranje

Vhod: Populacija P
Izhod: Fronte osebkov $F_1, F_2, \ldots$

```
1: function SORT(P)
2:     for all p ∈ P do
3:         S_p = ∅
4:         n_p = 0
5:         for all q ∈ P do
6:             if (p ≺ q) then
7:                 S_p = S_p ∪ {q}
8:             else if (q ≺ p) then
9:                 n_p = n_p + 1
10:            end if
11:        end for
12:        if n_p = 0 then
13:            p_rank = 1
14:            F_1 = F_1 ∪ {p}
15:        end if
16:    end for
17:    i = 1
18:    while F_i ≠ ∅ do
19:        Q = ∅
20:        for all p ∈ F_i do
21:            for all q ∈ S_p do
22:                n_q = n_q − 1
23:                if n_q = 0 then
24:                    q_rank = i + 1
25:                    Q = Q ∪ {q}
26:                end if
27:            end for
28:        end for
29:        i = i + 1
30:        F_i = Q
31:    end while
32:    return F_1, F_2, ..., F_i
33: end function
```

3.4.2 MOEA/D

Algoritem MOEA/D (Multiobjective Evolutionary Algorithm based on Decomposition) je dekompozicijski, kar pomeni, da večkriterijski optimizacijski algoritem prevede na več enokriterijskih optimizacijskih problemov. To je mogoče storiti na več načinov, opisanih v članku [10], v tem delu pa si bomo ogledali enega izmed njih, pristop uteženih vsot.

Naj bo $\lambda = (\lambda_1, \ldots, \lambda_k)$ tak vektor uteži, da za vsak $i = 1, \ldots, k$, velja $\lambda_i \geq 0$. Rešitev enokriterijskega optimizacijskega problema, kjer iščemo optimum funkcije

$$f_\lambda(x) = \sum_{i=1}^{k} \lambda_i f_i(x)$$

je Pareto optimalna rešitev originalnega večkriterijskega problema.

Da dobimo več rešitev iz Pareto optimalne fronte, poiščemo rešitve zgoraj opisanih enokriterijskih problemov za več različnih vektorjev uteži λ. Ta pristop dobro deluje le za optimizacijske probleme, katerih Pareto optimalna fronta je konveksna množica, sicer pa vseh rešitev iz le-te na ta način ne moremo poiskati.

Oglejmo si kako algoritem MOEA/D deluje. Naj bodo $\lambda^1, \ldots, \lambda^N$ enakomerno porazdeljeni vektorji uteži v prostoru $\mathbb{R}^k$. Na opisani način lahko večkriterijski optimizacijski problem prevedemo na N enokriterijskih podproblemov iskanje minimumov funkcij $f_{\lambda^i}(x)$ za $i = 1, \ldots, N$. MOEA/D vse te funkcije minimizira istočasno.

V kontekstu optimizacije s tem algoritmom je *okolica vektorja uteži* λ^i množica nekaj najbližjih vektorjev vektorja λ^i v $\lambda^1, \ldots, \lambda^N$. *Okolica i-tega podproblema* so podproblemi z vektorji uteži iz okolice λ^i. Populacija algoritma MOEA/D je v vsaki generaciji sestavljena iz dotlej najdene najboljše rešitve za vsakega izmed N podproblemov, ki jo iščemo le s pomočjo trenutnih rešitev njegove okolice.

Algoritmu podamo optimizacijski problem, ustavitveni pogoj, število podproblemov oziroma osebkov v populaciji N, enakomerno porazdeljene vektorje uteži $\lambda^1, \ldots, \lambda^N$ in velikost okolic T.

Algoritem 5 MOEA/D

Vhod: Kriterijska preslikava $F = (f_1, f_2, \ldots, f_k)$ in omejitve $o = (g_1, \ldots, g_q, h_1, \ldots, h_p)$
Izhod: Približek Pareto optimalne fronte ZP

function MOEAD(F, o)
  Pripravi zunanjo populacijo $ZP = \emptyset$ za shranjevanje nedominiranih rešitev.
  Generiraj N enakomerno porazdeljenih vektorjev uteži $\lambda^1, \lambda^2, \ldots, \lambda^N$.
  Za vsak vektor uteži λ^i poišči množico T njemu najbližjih vektorjev uteži $\lambda^{i_1}, \ldots, \lambda^{i_T}$ in nastavi $B(i) = \{i_1, \ldots, i_T\}$.
  Pripravi začetno populacijo $x_1, \ldots, x_N$.
  Pripravi vektor do sedaj najboljših rešitev posameznih kriterijskih funkcij $z = (z_1, \ldots, z_k)$.
  while $\neg$ ustavitveni pogoj **do**
    for $i = 1, 2, \cdots, N$ **do**
      Naključno izberi indeksa k, l iz $B(i)$ in iz x_k in x_l generiraj nov osebek y.
      Mutiraj y.
      Če za $j = 1, \ldots, k$ velja $z_j > f_j(y)$, nastavi $z_j = f_j(y)$.
      Če za $j \in B(i)$ velja $f_{\lambda^j}(y) < f_{\lambda^j}(x_j)$, nastavi $x_j = y$.
      Posodobi ZP.
    end for
  end while
  return ZP
end function

4. Pregled možnosti za prilagoditev algoritmov za večkriterijsko optimizacijo z omejitvami

Težava pri obravnavanju optimizacijskih problemov z omejitvami se ponovno pojavi pri selekciji. Ne vemo, kako naj obravnavamo osebke, ki kršijo katero izmed omejitev, oziroma natančneje, ne vemo, kako naj osebke primerjamo med sabo in razvrstimo, da za naslednjo generacijo izberemo najboljše.

Najpreprostejša in najpogostejša metoda obravnavanja omejitev so *kazenske preslikave*. Te kriterijskim preslikavam osebkov, ki kršijo katero izmed omejitev, prištejejo ali odštejejo določene

kazenske vrednosti. Preprost primer le-teh je t. i. *death penalty* ali *smrtna kazen*. Osebkov, ki kršijo katerokoli izmed omejitev, sploh ne upoštevamo. Ta metoda je preprosta za implementacijo, a lahko doseže slabe rezultate, saj zavračanje nedopustnih osebkov v bližini dopustnih osebkov omeji algoritmovo sposobnost učinkovitega preiskovanja prostora in s tem njegovo zmožnost iskanja dopustnih rešitev. Ta pojav je mogoče opaziti pri problemih, katerih majhen delež dopustnih rešitev je obkrožen z velikim deležem nedopustnih.

V splošnem kazenske preslikave delimo na dva razreda, *statične*, katerih vrednost kazenske preslikave je neodvisna od generacije, in *adaptivne*, pri katerih so informacije pridobljene med preiskovanjem uporabljene za prilagajanje velikosti kazni, ki jo pripišemo nedopustnim osebkom. Pogosto uporabljen primer statične kazenske preslikave je preslikava, ki kriterijski preslikavi osebka, ki krši omejitve, prišteje vektor uteženih vsot kršitev omejitev.

Oglejmo si še en primer kazenske preslikave, ki sta jo v [6] opisala Kuri in Quezada. Kazen sta definirala kot

$$P(x) = \begin{cases} (K - s\frac{K}{p+q}) - F(x), & s \neq p+q \\ 0, & \text{sicer} \end{cases}$$

kjer je K velika konstanta reda 10^9, p in qštevilo omejitev z neenakostjo in število omejitev z enakostjo in s število omejitev, ki jih osebek izpolnjuje. Kazen $P(x)$ prištejemo kriterijski preslikavi. Če je osebek dopusten, dobimo kar kriterijsko preslikavo samo, sicer pa veliko vrednost neodvisno od nje.

Poleg kazenskih preslikav obstajajo še mnogi drugi načini obravnave omejitev. Algoritem imenovan MOBES (Multi Objective Evolution Strategy) [1] obravnava tako vrednosti kriterijske preslikave kot tudi kršitve omejitev in osebke razvrsti v razrede glede na to, kako blizu so dopustnim rešitvam. Nato jih uredi glede na razrede.

Zanimiv način za rangiranje osebkov je tudi pojem dominiranosti z omejitvami, ki so ga v svojem algoritmu uporabili Deb in sodelavci [3].

Definicija 5. Rešitev i *dominira z omejitvami* nad rešitvijo j, če velja katerikoli izmed sledečih pogojev:

1. rešitev i je dopustna, rešitev j pa ne,
2. obe rešitvi sta nedopustni in i manj krši omejitve kot j,
3. obe rešitvi sta dopustni in i dominira nad j.

V algoritmu ENORA (Evolutionary Algorithm of Nondominated Sorting with Radial Slots) [7] je uporabljena min-max strategija za obravnavanje omejitev, kar pomeni, da največjo možno škodo (v našem primeru koliko osebki kršijo omejitve). Dopustne rešitve se razvijajo proti Pareto optimalnim, nedopustne pa proti dopustnim.

V članku [9] je predlagan način za obravnavanje omejitev, ki uporablja modificirane vrednosti kriterijskih funkcij za preverjanje dominiranosti v populaciji. Modifikacija je sestavljena iz dveh delov: mere za razdaljo in adaptivne kazenske funkcije. Prvo izmed teh dveh količin dobimo tako, da najprej izračunamo minimume in maksimume vsake izmed kriterijskih funkcij, ki so doseženi na populaciji

$$f^i_{\min} = \min_{x \in P} f_i(x),$$
$$f^i_{\max} = \max_{x \in P} f_i(x),$$

in jih uporabimo za normalizacijo vrednosti kriterijskih funkcij za vsakega izmed osebkov

$$\bar{f}_i(x) = \frac{f_i(x) - f^i_{\min}}{f^i_{\max} - f^i_{\min}},$$

kjer je $\bar{f}_i(x)$ normalizirana vrednost i-te kriterijske funkcije osebka x za $i = 1, \ldots, k$. Kršitev omejitev osebka $v(x)$ nato izračunamo kot vsoto normaliziranih kršitev vsake izmed omejitev deljeno s številom vseh omejitev

$$v(x) = \frac{1}{m} \sum_{j=1}^{p+q} \frac{c_j(x)}{c^j_{\max}},$$

kjer je

$$c_j(x) = \begin{cases} \max(0, g_j(x)), & j = 1, \ldots, p, \\ \max(0, |h_{j-p}(x)| - \delta), & j = p+1, \ldots, p+q, \end{cases}$$
$$c^j_{\max} = \max_{x \in P} c_j(x)$$

in δ toleranca za odstopanje pri omejitvah z enakostjo. Vrednost razdalje je za vsak osebek iz populacije nato izračunana kot

$$d_i(x) = \begin{cases} v(x), & \text{če je } r_f = 0, \\ \sqrt{\bar{f}_i(x)^2 + v(x)^2}, & \text{sicer}, \end{cases}$$

za

$$r_f = \frac{\text{število dopustnih osebkov v populaciji}}{\text{velikost populacije}}.$$

Za adaptivno kazensko funkcijo moramo izračunati dve kazenski funkciji, ki kaznujeta osebke glede na vrednosti kriterijskih funkcij in kršitve omejitev. Ti dve kazenski funkciji še poslabšata vrednost nedopustnih osebkov in identificirata najboljše nedopustne osebke tako, da vsakemu nedopustnemu osebku prištejeta različne kazni. Ti dve kazni sta za osebek x v i-ti dimenziji prostora kriterijev definirani kot

$$p_i(x) = (1 - r_f) X_i(x) + r_f Y_i(x)$$

za

$$X_i(x) = \begin{cases} 0, & \text{če je } r_f = 0, \\ v(x), & \text{sicer} \end{cases}$$

in

$$Y_i(x) = \begin{cases} 0, & \text{če je } x \text{ dopusten osebek}, \\ \bar{f}_i(x), & \text{če je } x \text{ nedopusten osebek}. \end{cases}$$

Če je v populaciji malo nedopustnih osebkov, prevlada prva kazen X_i, ki ima visoke vrednosti za osebke, ki veliko kršijo omejitve, sicer pa prevlada druga kazen Y_i, ki ima višje vrednosti za osebke z višjimi vrednostmi kriterijskih funkcij. Končna vrednost modificirane kriterijske funkcije osebka x, ki jo bomo uporabili za nedominirano sortiranje osebkov, je vsota mere za razdaljo in adaptivne kazenske funkcije v i-ti dimenziji prostora kriterijev

$$F_i(x) = d_i(x) + p_i(x).$$

Pri tem postopku je zanimivo, da je v primeru, ko v trenutni populaciji ni dopustnih osebkov, vsaka vrednost $d_i(x)$ enaka kršitvi omejitev $v(x)$, vsaka vrednost $p_i(x)$ pa enaka nič. Tako algoritem povsem zanemari vrednosti kriterijskih funkcij in osebke primerja glede na to, koliko kršijo omejitve, kar pomaga pri iskanju dopustnih rešitev. Če v populaciji obstajajo dopustni osebki, so najbolje ocenjeni tisti z majhnimi vrednostmi kriterijskih funkcij in kršitvami omejitev. Če pa v trenutni populaciji ni nedopustnih osebkov, so osebki primerjani le glede na vrednosti kriterijskih funkcij.

5. Eksperimentalno ovrednotenje načinov obravnavanja omejitev

5.1 Testna problema

Algoritma in načina obravnavanja omejitev bomo primerjali na dveh realnih inženirskih problemih z lastnostmi opisanimi v tabeli 1. Inženirska problema sta bila izbrana namesto "umetno ustvarjenih", saj so slednji pogosto ustvarjeni z namenom, da izpostavijo prednosti ali slabosti določenih algoritmov ter manj kompleksni, da je lažje analitično poiskati njihove rešitve. Problemi, za katere se genetski algoritmi zares uporabljajo pa so bolj podobni izbranima.

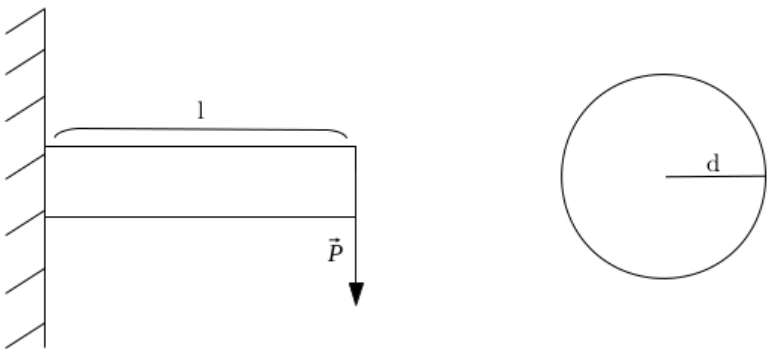
Delež dopustnih rešitev je bil določen z metodo tipa Monte Carlo tako, da smo naključno generirali veliko število osebkov in izračunali delež dopustnih.

Tabela 1. Lastnosti problemov

problem	# spremenljivk	# kriterijev	# omejitev	delež dopustnih rešitev
konzola	2	2	2	0,25724
vibrirajoča ploščad	5	2	5	0,0000031

5.1.1 Konzola

Prvi izbran problem je opisan v knjigi [2]. Oblikujemo konzolo valjaste oblike z osnovno ploskvijo s polmerom d in višine l, kot je prikazano na sliki 7.



Slika 7. Skica konzole.

Minimiziramo njeno težo

$$f_1(d,l) = \rho\frac{\pi d^2}{4}l$$

in njen odklon δ, ko na njen konec delujemo s silo P

$$f_2(d,l) = \delta = \frac{64Pl^3}{3E\pi d^4},$$

pri danih omejitvah na dolžino in polmer ter pogojih, da sta maksimalna obremenitev konzole σ_{max}

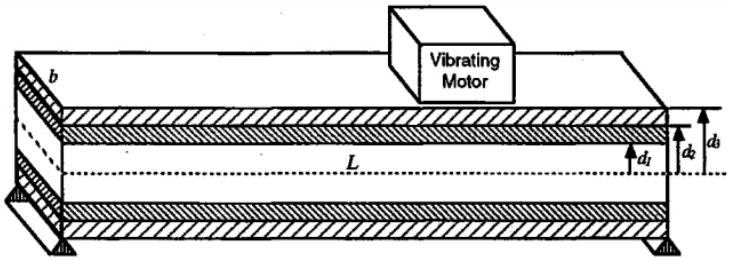
in končni odklon δ manjša od največjih dovoljenih vrednosti S_y in δ_{max}:

$$\begin{aligned} 0,01 \text{ m} \leq d \leq 0,05 \text{ m}, \\ 0,2 \text{ m} \leq l \leq 1,0 \text{ m}, \\ \sigma_{max} = \frac{32Pl}{\pi d^3} \leq S_y, \\ \delta \leq \delta_{max}, \end{aligned}$$

kjer so fizikalne konstante, ki določajo zgornje vrednosti $\rho = 7800$ kg/m^3, $P = 1$ kN, $E = 207$ GPa, $S_y = 300$ MPa in $\delta_{max} = 5$ mm.

5.1.2 Vibrirajoča plošča

Drugi problem je opisan v članku [5].



Slika 8. Skica vibrirajoče plošče, katere lastnosti optimiziramo. Slika iz [5].

Imamo ploščo dolžine L in širine b, izdelano iz sendviča treh materialov kot na sliki 8. Števila d_1, d_2 in d_3 označujejo dimenzije teh materialov, tako da je $2d_1$ debelina prvega, $d_2 - d_1$ debelina drugega in $d_3 - d_2$ debelina tretjega materiala. Na ploščo bomo pritrdili motor, ki vibrira s frekvenco 10 Hz. Da bi preprečili interferenco in zagotovili zadostno dušenje vibracij, želimo čim večjo lastno frekvenco plošče

$$f_1(d_1, d_2, d_3, b, L) = \frac{\pi}{2L^2} \left(\frac{EI(d_1, d_2, d_3, b, L)}{\mu(d_1, d_2, d_3, b, L)} \right)^{1/2}$$

in čim manjšo ceno izdelave

$$f_2(d_1, d_2, d_3, b) = 2b \left[c_1 d_1 + c_2(d_2 - d_1) + c_3(d_3 - d_2) \right],$$

pri omejitvah

$$\begin{aligned} 0{,}01 \leq d_1 \leq 0{,}6, \\ 0{,}01 \leq d_2 \leq 0{,}6, \\ 0{,}01 \leq d_3 \leq 0{,}6, \\ 0 \leq d_2 - d_1 \leq 0{,}01, \\ 0 \leq d_3 - d_2 \leq 0{,}01, \end{aligned}$$

$$\begin{aligned} 3 \leq L \leq 6, \\ 0{,}35 \leq b \leq 0{,}5, \\ \mu L \leq 2800, \end{aligned}$$

kjer sta

$$EI(d_1, d_2, d_3, b, L) = \frac{2b}{3}\left[E_1 d_1^3 + E_2(d_2^3 - d_1^3) + E_3(d_3^3 - d_2^3)\right],$$
$$\mu(d_1, d_2, d_3, b, L) = 2b\left[\rho_1 d_1 + \rho_2(d_2 - d_1) + \rho_3(d_3 - d_2)\right]$$

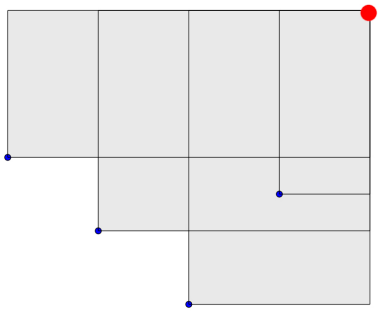
in trojice (ρ_1, E_1, c_1), (ρ_2, E_2, c_2) in (ρ_3, E_3, c_3) označujejo gostoto, prožnostni modul in ceno posameznih materialov.

5.2 Analiza in rezultati

Analizo smo izvedli v programskem jeziku Python z uporabo knjižnice PyGMO [11], ki omogoča reševanje optimizacijskih problemov z različnimi algoritmi. Algoritma NSGA-II in MOEA/D sta v knjižnici že implementirana, njune parametre lahko po želji prilagajamo, poleg tega pa omogoča tudi obravnavanje omejitev v večkriterijskih optimizacijskih problemih s smrtno kaznijo in Kurijevo metodo opisano v poglavju 4.

Vsakega izmed testnih problemov smo reševali z algoritmoma NSGA-II in MOEA/D, ter pri vsakem preizkusili tako smrtno kazen, kot tudi Kurijevo metodo. Pri tem smo uporabljali populacijo stotih osebkov v 250 generacijah.

Za primerjavo uspešnosti algoritmov smo uporabili ploščino dobljene končne populacije glede na vnaprej določeno referenčno točko (določili smo jo eksperimentalno, tako da je vsaka njena koordinata gotovo večja od največjih koordinat osebkov), kot je to prikazano na sliki 9.



Slika 9. Ploščina množice (siva površina) označene z modrimi pikami izračunana glede na referenčno točko, označeno z rdečo.

Celotna Pareto optimalna fronta skupaj z referenčno točko opisuje največjo možno ploščino. Ploščina končne populacije se torej veča, ko se njeni osebki bližajo Pareto optimalni fronti, ali pa se veča število osebkov v prvi nedominirani fronti. Genetski algoritmi uporabljajo naključnost, zato lahko dobimo ob različnih zagonih različne rešitve. Ker pa so lahko včasih optimizacijski problemi računsko zelo zahtevni, si več zagonov algoritma ne moremo vedno privoščiti, zato želimo, da bi si bili rezultati čim bolj podobni ali celo enaki. Tudi za indikator tega lahko uporabimo ploščino. Torej ne želimo le, da bi ob posameznih zagonih algoritma dobili čim večjo ploščino končne populacije, temveč želimo tudi, da bi čim manj odstopala od povprečja več zagonov.

Za vsak algoritem smo izvedli 30 zagonov za vsakega izmed izbranih načinov obravnavanja omejitev in narisali skupne grafe ploščin glede na generacijo za 10 zagonov in grafe končnih približkov Pareto optimalne fronte.

Kot lahko opazimo na sliki 10 in v tabeli 2, se na problemu konzole, neodvisno od izbire metode obravnavanja omejitev, bolje obnese algoritem NSGA-II, saj je standardni odklon ploščine za kar tri velikostne razrede manjši od standardnega odklona pri algoritmu MOEA/D. To lahko pojasnimo, če si ogledamo tabelo 1, in opazimo, da je delež dopustnih rešitev v prostoru kriterijev za ta problem velik in način obravnavanja omejitev zaradi tega ni tako pomemben.

Testni problem	Algoritem	Način obravnavanja omejitev	Povprečna ploščina	Standardni odklon ploščine
Konzola	NSGA-II	Smrtna kazen	0,657	$3{,}03 \times 10^{-7}$
		Kuri	0.657	$4{,}07 \times 10^{-7}$
	MOEA/D	Smrtna kazen	0,656	$6{,}52 \times 10^{-4}$
		Kuri	0,656	$8{,}19 \times 10^{-4}$
Vibrirajoča plošča	NSGA-II	Smrtna kazen	63814,7	134648,7
		Kuri	351248,1	24368,9
	MOEA/D	Smrtna kazen	20526,3	64909,9
		Kuri	464129,2	20481,4

Tabela 2. Povprečne vrednosti in standardni odkloni ploščin za 250 generacij.

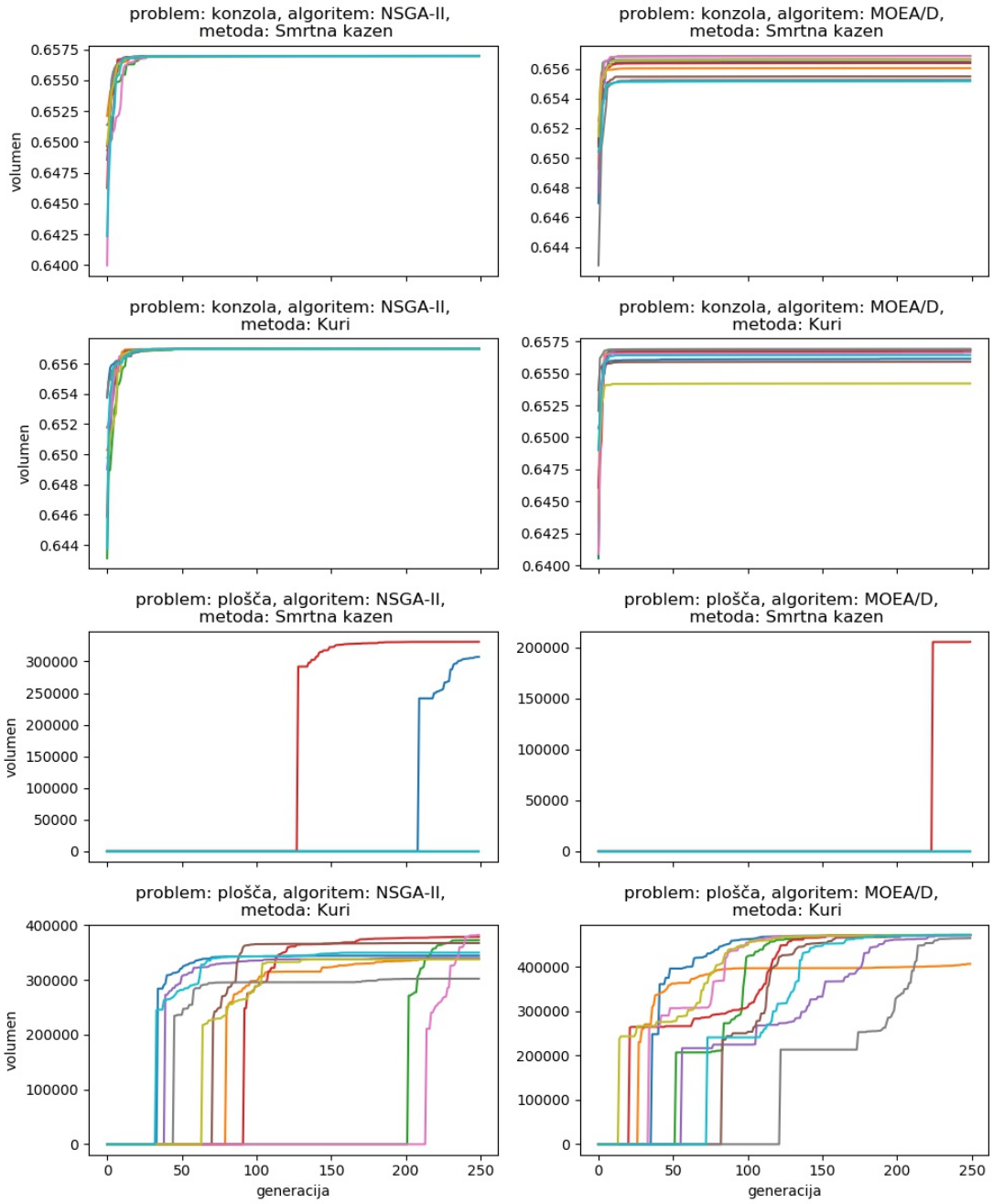
Na sliki 11 si oglejmo še nekaj grafov končnih aproksimacij Pareto fronte, ki jih dobimo za ta problem. Vidimo, da je razpon rešitev pri vseh konfiguracijah podoben, zato so podobne tudi dobljene ploščine. Glavna razlika med algoritmoma je v številu najdenih nedominiranih rešitev – MOEA/D jih očitno najde manj. To lahko pojasnimo z dejstvom, da je vsaka rešitev večkriterijskega problema, ki ga z metodo uteženih vsot prevedemo na enokriterijskega, tudi rešitev originalnega problema, obratno pa ne velja nujno (dokazano v [8]).

Na večje težave naletimo pri problemu vibrirajoče plošče. Tu je delež dopustnih rešitev v prostoru kriterijev veliko manjši, zato algoritmi težje najdejo te rešitve, še težje pa se približajo Pareto optimalni fronti, saj je zelo verjetno okoli vsake dopustne rešitve veliko nedopustnih, zato je preiskovanje prostora oteženo.

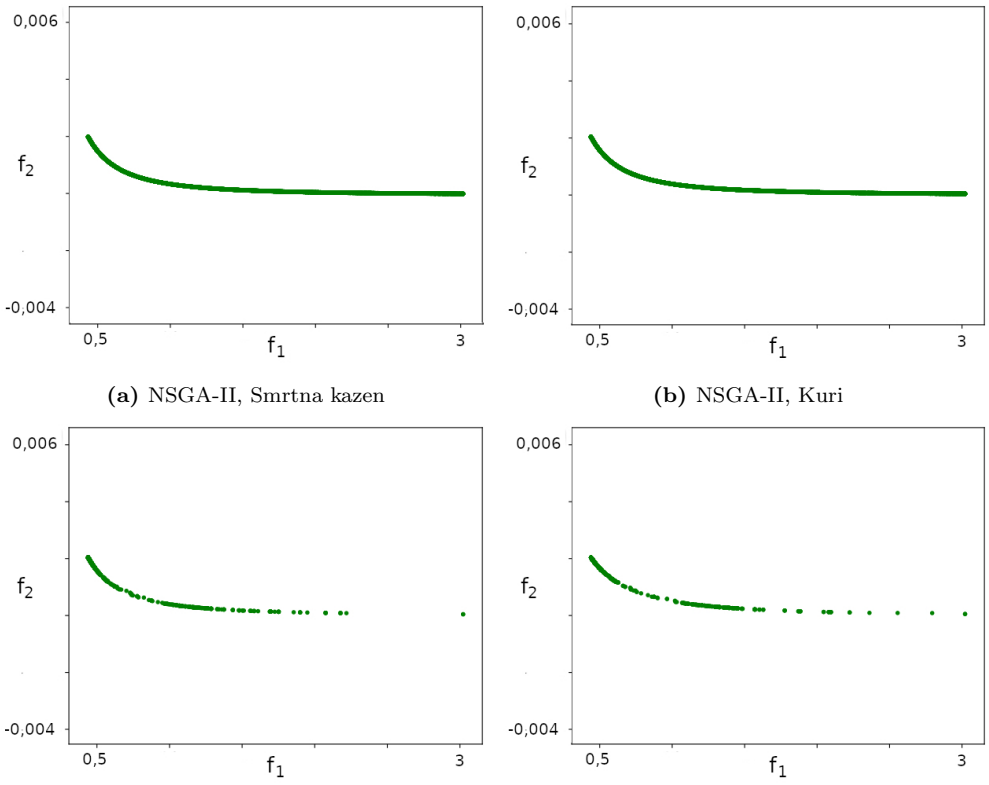
Opazimo lahko, da je obravnavanje omejitev s smrtno kaznijo popolnoma neuspešno. V kombinaciji z algoritmom NSGA-II najde dopustne rešitve v le dveh, v kombinaciji z MOEA/D pa le v enem od desetih zagonov algoritma in še tu, kot je razvidno iz slike 12, najde le tri dopustne rešitve. Bolje se obnese Kurijeva metoda. Ta v vseh zagonih algoritma najde dopustne rešitve, vendar pa se te od zagona do zagona precej razlikujejo, kot si lahko ogledamo na sliki 13. V primeru 13a se je algoritem odrezal dobro, v primerih 13b, 13c in 13d pa je našel le del množice iz primera 13a in še nekaj rešitev, ki v resnici niso dobra aproksimacija za rešitve iz Pareto fronte. To je tudi razlog, zakaj na grafih testnega problema vibrirajoče plošče na sliki 10 prihaja do takih odstopanj v ploščinah.

Opazimo pa, da na grafu te metode na sliki 10 najbolj odstopa ploščina zagona predstavljenega z oranžno barvo, vendar pa izgleda, kot da še narašča in se zato porodi ideja, da bi povečali število generacij in morda opazili izboljšanje.

Oglejmo si še nekaj rezultatov, ki smo jih dobili, ko smo število generacij povečali na 500. Vidimo, da so grafi ploščin na sliki 14 za algoritem MOEA/D in Kurijevo metodo obravnavanja omejitev na koncu res bolj enotni. Tudi iz tabele 3 lahko razberemo, da je standardni odklon ploščin končnih aproksimacij Pareto front kar za dva velikostna razreda manjši, kot standardni odklon, ko smo uporabljali le 250 generacij, torej smo rezultate res izboljšali. Zanimivo pa je,



Slika 10. Grafi ploščin za različne konfiguracije algoritmov in testnih problemov za 250 generacij.



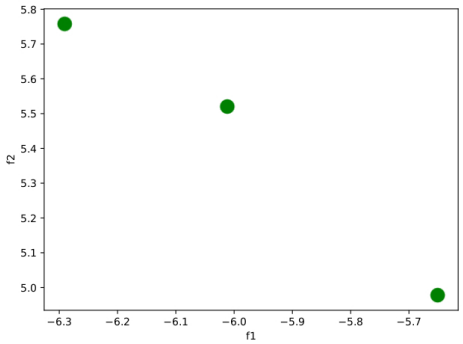
(a) NSGA-II, Smrtna kazen

(b) NSGA-II, Kuri

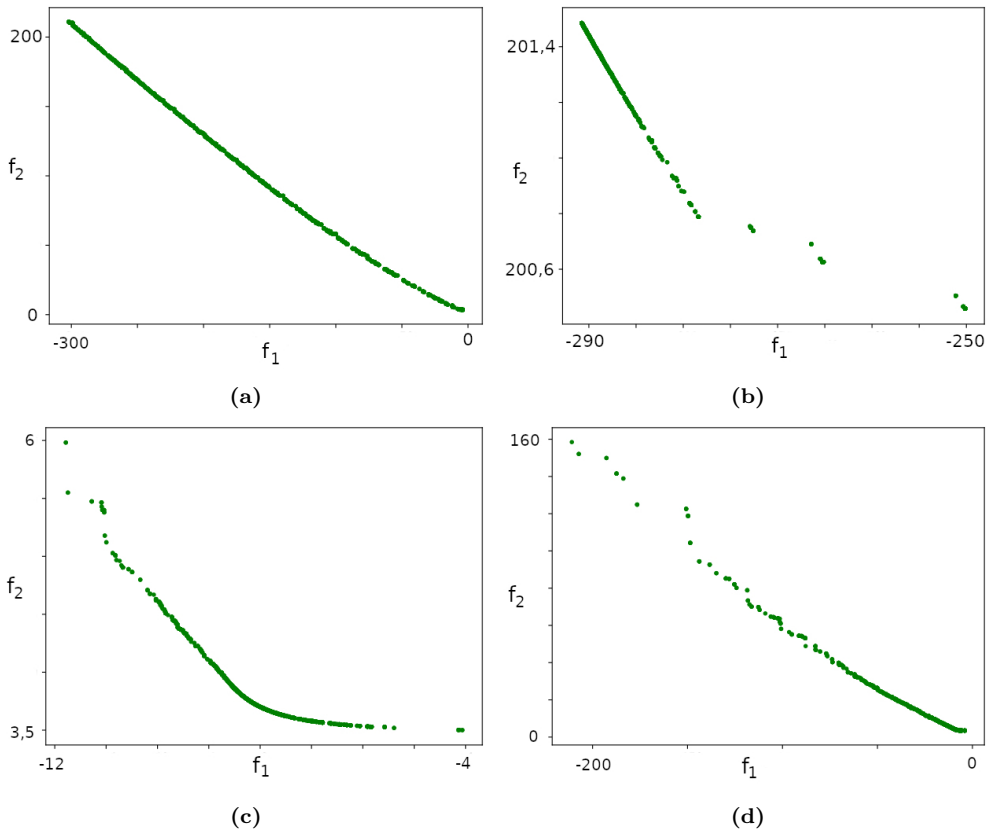
(c) MOEA/D, Smrtna kazen

(d) MOEA/D, Kuri

Slika 11. Grafi končnih aproksimacij Pareto fronte za testni problem oblikovanja konzole.



Slika 12. Algoritem MOEA/D v kombinaciji s smrtno kaznijo v enem zagonu najde le tri dopustne rešitve problema vibrirajoče plošče.



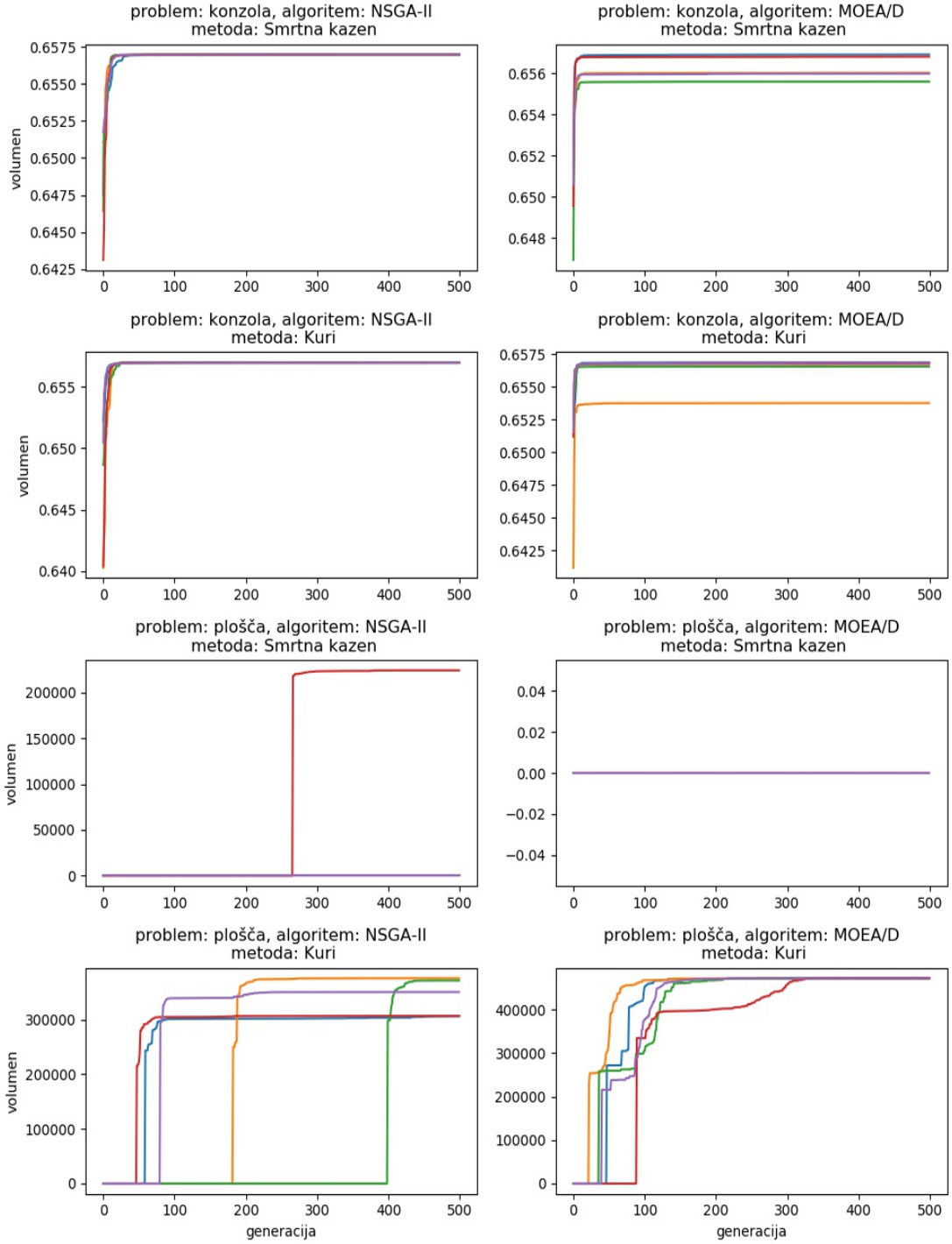
Slika 13. Grafi končnih aproksimacij Pareto fronte za testni problem oblikovanja vibrirajoče plošče z algoritmom MOEA/D in metodo obravnavanja omejitev, ki jo je predlagal Kuri, za štiri različne zagone algoritma.

da se rezultati problema načrtovanja konzole z uporabo algoritma MOEA/D ob povečanju števila generacij ne izboljšajo občutno. To lahko pomeni, da algoritem v tem primeru pride do lokalnih minimumov in tam obstane.

Testni problem	Algoritem	Način obravnavanja omejitev	Povprečna ploščina	Standardni odklon ploščine
Konzola	NSGA-II	Smrtna kazen	0,657	$2{,}17 \times 10^{-7}$
		Kuri	0.657	$1{,}39 \times 10^{-7}$
	MOEA/D	Smrtna kazen	0,656	$5{,}70 \times 10^{-4}$
		Kuri	0,656	$13{,}45 \times 10^{-4}$
Vibrirajoča plošča	NSGA-II	Smrtna kazen	44836,5	100257,6
		Kuri	342264,9	33991,3
	MOEA/D	Smrtna kazen	0	0
		Kuri	471968,9	170,8

Tabela 3. Povprečne vrednosti in standardni odkloni ploščin za 500 generacij.

Glede na vrednosti standardnega odklona ploščine pri reševanju problema načrtovanja vibrirajoče plošče opazimo, da se, za razliko od optimizacije problema konzole, najbolje obnese algoritem MOEA/D.



Slika 14. Grafi ploščin za vse konfiguracije in 500 generacij.

6. Zaključek

Kot smo ugotovili v prejšnjih razdelkih, so genetski algoritmi uporabno orodje za reševanje težkih inženirskih optimizacijskih problemov, ki jih analitično ne moremo rešiti v doglednem času. To je še posebej pomembno pri reševanju večkriterijskih optimizacijskih problemov, ki se izkažejo za dosti težje od enokriterijskih. Ogledali smo si in primerjali dva izmed vodilnih genetskih algoritmov in dva načina obravnavanja omejitev, izkazalo pa se je, da v splošnem najboljšega med njimi ni. Izbiro algoritma ter načina obravnavanja omejitev je potrebno zato prilagoditi problemu, ki ga rešujemo, to pa zahteva dobro poznavanje problema ter nekaj znanja o njegovi rešitvi, kar rabo teh algoritmov nekoliko oteži.

LITERATURA

[1] T. T. Binh in U. Korn, *MOBES: A Multiobjective Evolution Strategy for Constrained Optimization Problems*, v: Proceedings of the Third International Conference on Genetic Algorithms, MENDEL '97, 1997, str. 176–182.

[2] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*, Wiley, 2001.

[3] K. Deb in dr., *A fast and elitist multiobjective genetic algorithm: NSGA-II*, IEEE Transactions on Evolutionary Computation **6**(2) (2002) 182–197.

[4] J. H. Holland, Ann Arbor, MI.

[5] A. Messac, *Physical programming - Effective optimization for computational design*, AIAA Journal **34** (1996) 149–158.

[6] A. K. Morales in C. V. Quezada, *A universal eclectic genetic algorithm for constrained optimization*, v: Proceedings 6th European Congress on Intelligent Techniques and Soft Computing, EUFIT '98, Verlag, Aachen, 1998, str. 518–522.

[7] T. Ray in K. S. Won, *An Evolutionary Algorithm for Constrained Bi-objective Optimization Using Radial Slots*, v: Knowledge-Based Intelligent Information and Engineering Systems, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, str. 49–56.

[8] T. Turk, *Vekriterijska optimizacija z evolucijskimi algoritmi : delo diplomskega seminarja*, doktorska disertacija, Univerza v Ljubljani, Fakulteta za matematiko in fiziko, 2012.

[9] Y. G. Woldesenbet, G. G. Yen in B. G. Tessema, *Constraint Handling in Multiobjective Evolutionary Optimization*, IEEE Transactions on Evolutionary Computation **13**(3) (2009) 514–525.

[10] Q. Zhang in H. Li, *MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition*, IEEE Transactions on Evolutionary Computation **11**(6) (2007) 712–731.

[11] *PyGMO*, [ogled 27. 8. 2018], dostopno na `http://esa.github.io/pygmo/`.