UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Računalništvo in matematika – 2. stopnja

Nace Sever

# ALGORITEM ZA IZRAČUN RAZDALJE DO NEDOMINIRANEGA OBMOČJA

Magistrsko delo

Mentor: prof. dr. Sergio Cabello Justo
Somentorica: doc. dr. Tea Tušar

Ljubljana, 2025

# Zahvala

Zahvaljujem se mentorju, prof. dr. Sergiu Cabellu Justu za dragocene nasvete in ure skupnega razmišljanja. Prav tako se zahvaljujem somentorici, doc. dr. Tei Tušar, da me je s tako zanimivim problemom seznanila, me tekom dela usmerjala in bila vedno na voljo. Še posebej pa se zahvaljujem svoji družini in punci Lizi za podporo, potrpežljivost in spodbudo.

# Kazalo

## Algoritem za izračun razdalje do nedominiranega območja


Povzetek

V magistrskem delu predstavimo nov algoritem ARRNO za računanje razdalje do nedominiranega območja. Razdalja med dominirano točko in nedominiranim območjem je metrika za razvrščanje dominiranih rešitev pri znanem dvokriterijskem optimizacijskem algoritmu COMO-CMA-ES. Novost algoritma ARRNO je, da omogoča izračun razdalje za tri ali več dimenzionalne točke, medtem ko obstoječi algoritem deluje le za množice dvodimenzionalnih točk.

Algoritem implementiramo v programskem jeziku Python ter eksperimentalno preverimo njegovo pravilnost, prostorsko zahtevnost in časovno zahtevnost na raznovrstnih množicah nedominiranih točk. Za množice tridimenzionalnih točk moči $n$ algoritem doseže časovno zahtevnost $O(n \log n)$, z višanjem dimenzije $D$ pa njegova časovna zahtevnost postane $O(n^{D-1})$.

Implementacija algoritma za tri in štiridimenzionalne množice je vključena tudi v odprtokodno knjižnico `moarchiving`, ki omogoča hranjenje množic nedominiranih rešitev in računanje indikatorjev v večkriterijski optimizaciji. Algoritem ARRNO tako omogoča razširitev algoritma COMO-CMA-ES na več kot dva kriterija.


## An algorithm for computing the distance to the nondominated area


Abstract

In this thesis, we present a new algorithm, ARRNO, for computing the distance to the nondominated area. The distance between a dominated point and the nondominated region is a metric used for ranking dominated solutions in the well-known bi-objective optimization algorithm COMO-CMA-ES. The novelty of ARRNO is that it enables the computation of this distance for points in three or more dimensions, whereas the existing algorithm is limited to sets of two-dimensional points.

The algorithm is implemented in Python, and its correctness and computational complexity are experimentally evaluated on various sets of nondominated points. For three-dimensional point sets of size $n$, ARRNO achieves a time complexity of $O(n \log n)$. For constant dimension $D \geq 4$, the time complexity is $O(n^{D-1})$.

An implementation of the algorithm for three and four-dimensional point sets is also included in the open-source `moarchiving` library, which provides efficient storage of nondominated solution sets and computation of indicators in multiobjective optimization. ARRNO thus enables the extension of COMO-CMA-ES to optimization problems with more than two objectives.



**Math. Subj. Class. (2020):** 68W05,68U05,90C29

**Ključne besede:** večkriterijska optimizacija, računska geometrija, dominirane točke

**Keywords:** multiobjective optimization, computational geometry, dominated points

# 1 Uvod

Optimizacija je področje matematike in računalništva, ki se ukvarja z iskanjem najboljše rešitve glede na določen kriterij. Pojavlja se na veliko področjih, na primer v inženirstvu, ekonomiji in drugih. Pri reševanju praktičnih problemov pa pogosto ne želimo optimirati le enega kriterija, ampak več, ponavadi nasprotujočih si kriterijev. Kadar, na primer, iščemo hotel ob morju, si želimo hkrati minimirati tako ceno, kot tudi oddaljenost od morja, kar pa pogosto ni moč doseči. Namesto ene optimalne rešitve tako iščemo množico rešitev, ki predstavljajo različne kompromise med kriteriji. Takšnim problemom pravimo večkriterijski optimizacijski problemi in jih rešujemo z uporabo večkriterijskih optimizacijskih algoritmov [5].

Obstaja veliko večkriterijskih optimizacijskih algoritmov, ki se razlikujejo predvsem po strategiji preiskovanja prostora rešitev in načinu primerjave med rešitvami. V večdimenzionalnem prostoru kriterijev namreč to ni enoznačno. Veliko algoritmov za rangiranje rešitev uporablja nedominirano urejanje (angl. *nondominated sorting*), ki rešitve razdeli na zaporedne fronte, glede na relacijo dominiranosti med rešitvami [7]. Pri algoritmu NSGA-II se rešitve znotraj posamezne fronte rangirajo glede na zgoščenost (angl. *crowding distance*) [7], pri algoritmu NSGA-III pa glede na oddaljenost do referenčnih vektorjev [6]. Drugače je pri algoritmu SMS-EMOA, ki za rangiranje rešitev uporablja prispevek hipervolumna (angl. *hypervolume contribution*) posamezne rešitve k vrednosti hipervolumna celotne množice rešitev [1].

Večkriterijska evolucijska strategija s prilagajanjem kovariančne matrike in selekcijo z vejico (angl. *Comma-Selection Multiobjective Covariance Matrix Adaptation Evolution Strategy* - COMO-CMA-ES) [13] je algoritem za reševanje dvokriterijskih optimizacijskih problemov, ki prostor preiskuje z algoritmom CMA-ES [9], za primerjavo med rešitvami pa uporablja izpopolnjeno različico hipervolumna [15], UHV (angl. *Uncrowded Hypervolume*). V študiji [13] COMO-CMA-ES dosega obetavne rezultate na testnih optimizacijskih problemih.

V izpopolnjeni različici hipervolumna UHV izračuna algoritem za dano rešitev razdaljo do nedominiranega območja. Avtorji COMO-CMA-ES so ta algoritem zasnovali le za dva kriterija, torej v dveh dimenzijah.

V magistrskem delu predstavimo nov algoritem za računanje razdalje do nedominiranega območja, imenujemo ga ARRNO. Deluje za poljubno dimenzijo $D \geq 3$, kar omogoča razširitev algoritma COMO-CMA-ES na poljubno število kriterijev. Nov algoritem ARRNO implementiramo in testiramo, da pokažemo njegovo pravilnost ter hitrost delovanja v različnih scenarijih in dimenzijah.

Implementacija algoritma ARRNO za tri in štiri dimenzije je že na voljo tudi kot del knjižnice `moarchiving` [11], v programskem jeziku Python. Knjižnica omogoča hranjenje arhiva nedominiranih rešitev večkriterijskih optimizacijskih problemov, hitro računanje hipervolumna ter drugih indikatorjev in pomožnih funkcij, med katerimi je tudi računanje razdalje do nedominiranega območja.

## 1.1 Pregled področja

V literaturi povezani z večkriterijsko optimizacijo je opisanih veliko sorodnih geometrijskih problemov in algoritmov za njihovo reševanje, ki se večinoma osredotočajo na problem učinkovitega računanja hipervolumna. V [10] je predstavljen algoritem deli

in vladaj za iskanje nedominiranih točk v poljubno dimenzionalnem prostoru, v [2] pa se avtorji ukvarjajo z zahtevnostjo računanja hipervolumna, ter zanjo predstavijo nekatere spodnje ter zgornje meje. Bringmann v [3] pokaže, da je računanje hipervolumna v poljubni dimenziji lažje od računanja mere unije osi vzporednih enotskih kock, s čimer poda zgornjo mejo $O(n^{\lfloor d/2 \rfloor}\text{polylog}(n))$ za računanje hipervolumna.

V [8] avtorji predstavijo hiter algoritem za računanje hipervolumna v treh in štirih dimenzijah (HV3D+, HV4D+), kjer uporabljajo kompleksne podatkovne strukture in rekurziven algoritem pometanja. Algoritem za računanje hipervolumna v treh dimenzijah implementirajo s časovno zahtevnostjo $O(n \log n)$, glede na število nedominiranih točk $n$, kar doseže teoretično spodnjo mejo iz [2]. Algoritem za računanje hipervolumna v štirih dimenzijah pa ima časovno zahtevnost $O(n^2)$. Za oba algoritma avtorji pokažejo, da sta trenutno najhitrejša izmed znanih implementacij, s testiranjem na množicah točk različnih oblik in velikosti.

Za razliko od računanja hipervolumna pa je razdalja do nedominiranega območja novejši in manj znan koncept. V dveh dimenzijah je algoritem za računanje razdalje do nedominiranega območja relativno preprost, avtorji algoritma COMO-CMA-ES ga implementirajo v [11]. Implementacije ali ideje za algoritem, ki bi deloval v več kot dveh dimenzijah, pa ne zasledimo. Nov algoritem ARRNO, ki ga vpeljemo v tem delu, se zgleduje tako po obstoječi implementaciji algoritma v dveh dimenzijah, kot tudi po algoritmih HV3D+ in HV4D+. Iz algoritma v dveh dimenzijah prevzame idejo računanja razdalje do vpetih točk, iz algoritmov HV3D+ in HV4D+ pa idejo o uporabi rekurzivnega algoritma pometanja.

## 1.2 Struktura dela

V poglavju 2 predstavimo večkriterijsko optimizacijo in definiramo osnovne pojme. V poglavju 3 najprej definiramo problem računanja razdalje med točko in nedominiranim območjem in predstavimo obstoječ algoritem za njegovo reševanje v dveh dimenzijah. Nato predstavimo nov algoritem za reševanje problema v treh dimenzijah ter ga razširimo na več dimenzij. V poglavju 4 predstavimo metodo, s katero testiramo pravilnost algoritma ter rezultate testiranja. V poglavju 5 pa najprej teoretično analiziramo časovno in prostorsko zahtevnost algoritma, nato pa časovno zahtevnost testiramo tudi na primerih različnih dimenzij in velikosti ter rezultate prikažemo v obliki grafov. V poglavju 6 rezultate povzamemo in predstavimo ideje za prihodnje delo.

# 2 Večkriterijska optimizacija

V tem poglavju definiramo splošni večkriterijski optimizacijski problem ter podamo osnovne pojme in definicije povzete po [15], ki jih potrebujemo v nadaljevanju.

Pri večkriterijski optimizaciji optimiramo funkcijo $f = (f_1, \ldots, f_D)$,

$$\begin{aligned} &f : X \to Z, \\ &f(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_D(\mathbf{x})), \end{aligned}$$

kjer so $f_i$ funkcije ene ali več spremenljivk, $X$ prostor spremenljivk in $Z \subseteq \mathbb{R}^D$ prostor kriterijev. Skozi celotno delo bomo brez škode za splošnost predpostavljali, da vse kriterije maksimiramo[1]. V okviru tega dela se bomo osredotočili na kriterijske vektorje $\mathbf{z} \in Z$, ki ustrezajo neki rešitvi $\mathbf{x}$ iz prostora spremenljivk $X$, torej velja $\mathbf{z} = f(\mathbf{x})$. Sledeče definicije bomo torej podali v prostoru kriterijev $Z$, vrednosti kriterijev $(z_1, \ldots, z_D)$ pa bomo obravnavali kot točko v $\mathbb{R}^D$.

**Definicija 2.1.** Točka $\mathbf{z} = (z_1, \ldots, z_D)$ *šibko dominira* točko $\mathbf{y} = (y_1, \ldots, y_D)$, če velja

$$\forall i \in \{1, \ldots, D\} : \quad z_i \geq y_i.$$

V tem primeru pišemo $\mathbf{z} \succeq \mathbf{y}$.

**Definicija 2.2.** Če točka $\mathbf{z} = (z_1, \ldots, z_D)$ šibko dominira točko $\mathbf{y} = (y_1, \ldots, y_D)$ in za nek $i \in \{1, \ldots, D\}$ velja

$$z_i > y_i,$$

potem rečemo, da $\mathbf{z}$ *dominira* $\mathbf{y}$ in pišemo $\mathbf{z} \succ \mathbf{y}$.

**Definicija 2.3.** Če za točki $\mathbf{z} = (z_1, \ldots, z_D)$ in $\mathbf{y} = (y_1, \ldots, y_D)$ velja

$$\forall i \in \{1, \ldots, D\} : \quad z_i > y_i,$$

potem rečemo, da $\mathbf{z}$ *strogo dominira* $\mathbf{y}$ in pišemo $\mathbf{z} \gg \mathbf{y}$.

**Definicija 2.4.** Če za točki $\mathbf{z} = (z_1, \ldots, z_D)$ in $\mathbf{y} = (y_1, \ldots, y_D)$ obstajata taka $i$ in $j$, da velja

$$z_i > y_i \wedge z_j < y_j$$

potem rečemo, da sta $\mathbf{z}$ in $\mathbf{y}$ *neprimerljivi* in pišemo $\mathbf{z} \parallel \mathbf{y}$.

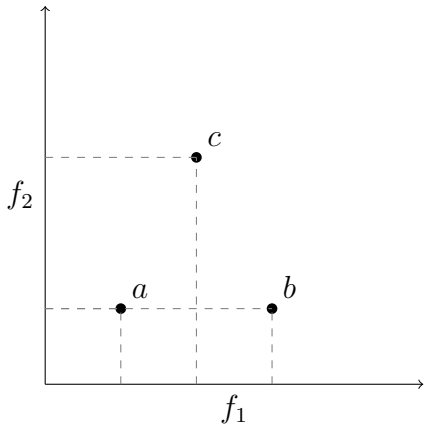
Očitno za relacijo $\parallel$ velja simetričnost, za relacije $\gg$, $\succ$ in $\succeq$ pa velja tranzitivnost ter tudi naravna ureditev

$$\mathbf{z} \gg \mathbf{y} \implies \mathbf{z} \succ \mathbf{y} \implies \mathbf{z} \succeq \mathbf{y}.$$

Za vsak par točk $\mathbf{y}$ in $\mathbf{z}$ velja natanko ena izmed relacij $=$, $\parallel$, $\succ$ ali $\prec$. Na sliki 1 so prikazane tri točke in relacije dominiranosti, ki veljajo med njimi v prostoru dveh kriterijev.

Relacijo dominiranosti lahko razširimo tudi na dve množici.

[1] Kriterij, ki bi ga sicer minimirali lahko pomnožimo z $-1$ ter tako dobimo ekvivalenten kriterij, ki ga maksimiramo.




Slika 1: Primeri relacij dominiranosti med točkami $a$, $b$ in $c$. Veljajo naslednje relacije: $c \succcurlyeq a$, $b \succ a$, $b \parallel c$, $c \parallel b$, $a \succeq a$, $b \succeq b$, $c \succeq c$.

**Definicija 2.5.** Množica $\mathcal{P}$ *dominira* množico $\mathcal{Q}$, če za vsako točko $\mathbf{q} \in \mathcal{Q}$ obstaja taka točka $\mathbf{p} \in \mathcal{P}$, da $\mathbf{p}$ dominira $\mathbf{q}$. Potem pišemo $\mathcal{P} \succ \mathcal{Q}$. V primeru, ko je $\mathcal{Q} = \{\mathbf{q}\}$, rečemo, da množica $\mathcal{P}$ dominira točko $\mathbf{q}$. Na soroden način definiramo tudi *šibko* in *strogo dominaranje* med množicama.

**Definicija 2.6.** *Hiperkvader* $H$, s stranicami vzporednimi s koordinatnimi osmi, ki je razpet med točkama $\mathbf{r} = (r_1, \ldots, r_D)$ in $\mathbf{p} = (p_1, \ldots, p_D)$, definiramo kot

$$H(\mathbf{r}, \mathbf{p}) = [r_1, p_1] \times \cdots \times [r_D, p_D].$$

**Definicija 2.7.** *Hipervolumen* množice točk $\mathcal{P}$ glede na *referenčno točko* $\mathbf{r}$, ki ga označimo s $HV(\mathcal{P}, \mathbf{r})$ je definiran kot Lebesguova mera $\lambda(\cdot)$ unije hiperkvadrov $H(\mathbf{r}, \mathbf{p})$, ki jih določajo točke iz $\mathbf{p} \in \mathcal{P}$ in referenčna točka $\mathbf{r} = (r_1, \ldots, r_D)$. Torej

$$HV(\mathcal{P}, \mathbf{r}) = \lambda \left( \bigcup_{\mathbf{p} \in \mathcal{P}} H(\mathbf{r}, \mathbf{p}) \right).$$

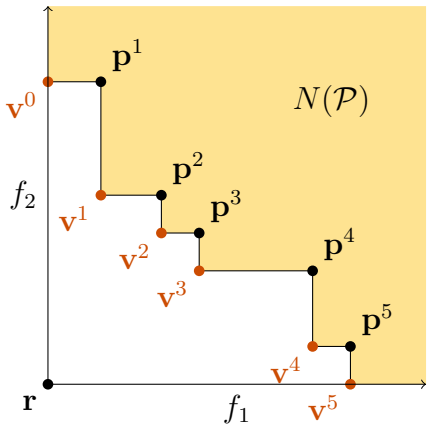
Hipervolumna kasneje ne bomo uporabili, vseeno pa smo ga definirali, ker smo ga omenili v uvodu in je relevantni koncept v povezanih delih. Brez škode za splošnost bo tekom celotnega dela referenčna točka $\mathbf{r} = (0, \ldots, 0)$.

**Definicija 2.8.** *Dominirano območje* množice točk $\mathcal{P}$ je množica točk $\mathbf{z} \in \mathbb{R}^D$, ki dominirajo referenčno točko $\mathbf{r} = \mathbf{0}$ in jih šibko dominira vsaj ena izmed točk v $\mathcal{P}$.

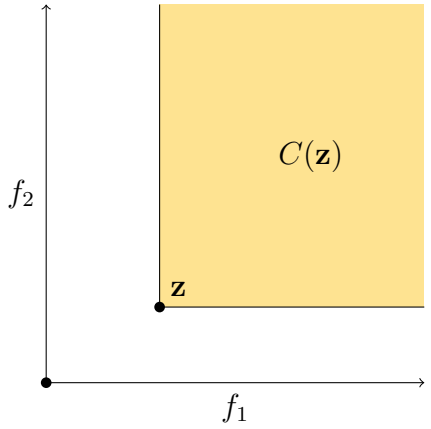
**Definicija 2.9.** *Nedominirano območje* $N(\mathcal{P})$ množice točk $\mathcal{P}$ je množica točk $\mathbf{z} \in \mathbb{R}^D$, ki dominirajo referenčno točko $\mathbf{r} = \mathbf{0}$ in jih ne strogo dominira nobena izmed točk v $\mathcal{P}$. Matematično to zapišemo kot

$$N(\mathcal{P}) = \{\mathbf{z} \in \mathbb{R}^D \mid \mathbf{z} \succeq \mathbf{0} \wedge \neg(\mathcal{P} \succcurlyeq \mathbf{z})\}.$$

Primer nedominiranega območja $N(\mathcal{P})$ za neko množico točk $\mathcal{P}$ v dvokriterijskem prostoru je prikazan na sliki 2. Presek dominiranega in nedominiranega območja za poljubno množico $\mathcal{P}$, ki vsebuje vsaj eno točko iz $\mathbb{R}^D_{\geq 0}$, je neprazen, saj obe množici vsebujeta rob.




Slika 2: Slika prikazuje množico točk $\mathcal{P} = \{\mathbf{p}^1, \ldots, \mathbf{p}^5\}$, nedominirano območje $N(\mathcal{P})$ ter vpete točke $\mathcal{V}(\mathcal{P}) = \{\mathbf{v}^0, \ldots, \mathbf{v}^5\}$. Del množice $N(\mathcal{P})$ je tudi rob, označen s črno črto.




Slika 3: Na sliki sta prikazana točka $\mathbf{z}$ in pripadajoči stožec $C(\mathbf{z})$, označen z oranžnim območjem. Del stožca $C(z)$ je tudi rob, označen s črno črto.

**Definicija 2.10.** Množici točk, ki šibko dominira neko točko $\mathbf{z} \in \mathbb{R}^D$ rečemo *stožec* in jo označimo s $C(\mathbf{z})$. Primer dvodimenzionalnega stožca vidimo na sliki 3.

**Definicija 2.11.** Točki $\mathbf{v} \in N(\mathcal{P})$ za katero velja

$$\nexists \mathbf{z} \in N(\mathcal{P}) : \quad \mathbf{v} \succ \mathbf{z}$$

rečemo *vpeta točka*. Množico vseh vpetih točk $\mathbf{v}$ za neko množico paroma nedominiranih točk $\mathcal{P}$ označimo z $\mathcal{V}(\mathcal{P})$. Primer množice vpetih točk za množico dvodimenzionalnih točk $\mathcal{P}$ vidimo na sliki 2.

Tekom dela računamo vpete točke za množice različnih dimenzij. Za to uporabimo različne algoritme, ki jih v nadaljevanju predstavimo. Verjamemo, da množice izračunanih vpetih točk natanko ustrezajo definiciji vpetih točk, vendar tega ne pokažemo.

# 3 Računanje razdalje med točko in nedominiranim območjem

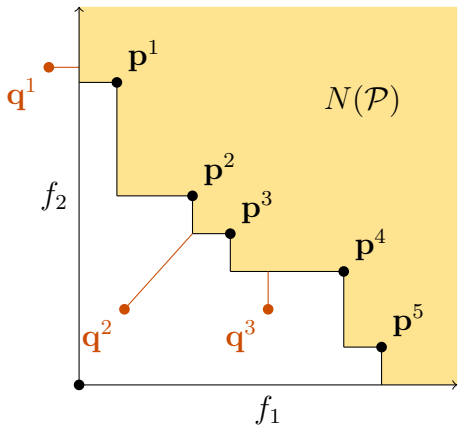
V tem poglavju najprej definiramo problem računanja razdalje med točko in nedominiranim območjem in predstavimo delovanje algoritma zanj v dveh dimenzijah. Na kratko predstavimo algoritme pometanja, na katerih temelji nov algoritem. Nato predstavimo nov algoritem ARRNO za računanje razdalje do nedominiranega območja, ki rešuje problem v treh dimenzijah. Na koncu algoritem posplošimo za reševanje problemov v poljubni višji dimenziji.

## 3.1 Definicija problema in osnovne lastnosti

Dana je množica $\mathcal{P}$, ki vsebuje $n$ paroma nedominiranih $D$-dimenzionalnih točk. Vse točke iz $\mathcal{P}$ dominirajo referenčno točko $\mathbf{r} = \mathbf{0}$ in so urejene padajoče po zadnji koordinati. Dana je tudi točka poizvedbe $\mathbf{q} \notin N(\mathcal{P})$. Izračunati želimo razdaljo med točko $\mathbf{q}$ in nedominiranim območjem $N(\mathcal{P})$ množice $\mathcal{P}$, torej

$$d(\mathbf{q}, N(\mathcal{P})) = \min_{\mathbf{z} \in N(\mathcal{P})} d(\mathbf{q}, \mathbf{z}).$$

Primer različnih poizvedbenih točk in njihovih razdalj do nedominiranega območja v dvodimenzionalnem prostoru vidimo na sliki 4.




Slika 4: Tri točke poizvedbe $\mathbf{q}^1$, $\mathbf{q}^2$, $\mathbf{q}^3$ in njihove razdalje do nedominiranega območja $N(\mathcal{P})$ za $\mathcal{P} = \{\mathbf{p}^1, \ldots, \mathbf{p}^5\}$.

**Trditev 3.1.** *Množico $N(\mathcal{P})$ lahko zapišemo kot unijo stožcev, ki jih razpenja množica vpetih točk $\mathcal{V}(\mathcal{P})$, torej*

$$N(\mathcal{P}) = \bigcup_{\boldsymbol{v} \in \mathcal{V}(\mathcal{P})} C(\boldsymbol{v}).$$

*Dokaz.* Najprej pokažemo, da velja

$$\forall \mathbf{v} \in \mathcal{V}(\mathcal{P}): \quad C(\mathbf{v}) \subseteq N(\mathcal{P}).$$

Predpostavimo, da obstaja $\mathbf{x} \in C(\mathbf{v})$, tako da $\mathbf{x} \notin N(\mathcal{P})$. Torej obstaja $\mathbf{p} \in \mathcal{P}$ da velja $\mathbf{p} \gg \mathbf{x} \succeq \mathbf{v}$. Ker pa je $\mathbf{v}$ po definiciji iz $N(\mathcal{P})$, neenakost $\mathbf{p} \gg \mathbf{v}$ ne more veljati. S tem smo dobili protislovje.

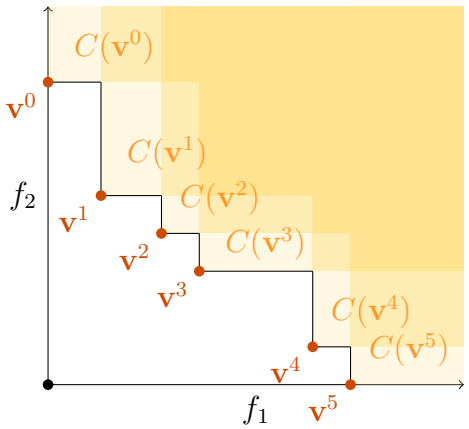
Pokažimo še neenakost v drugo smer

$$N(\mathcal{P}) \subseteq \bigcup_{\mathbf{v} \in \mathcal{V}(\mathcal{P})} C(\mathbf{v}).$$

Naj bo $\mathbf{x} = (x_1, x_2, \ldots, x_D) \in N(\mathcal{P})$. Torej za vsak $i \in \{1, \ldots, D\}$ velja $x_i \geq 0$. Nadalje naj bo $v_1 = \min\{v \mid (v, x_2, \ldots, x_D) \in N(\mathcal{P})\}$ in $\mathbf{x}^1 = (v_1, x_2, \ldots, x_D)$. Tak $v_1$ gotovo obstaja, saj velja $v_1 \in \{0\} \cup \{p_1 \mid p \in \mathcal{P}\}$. Podobno naj bo $v_2 = \min\{v \mid (v_1, v, x_3, \ldots, x_D) \in N(\mathcal{P})\}$ in $\mathbf{x}^2 = (v_1, v_2, x_3, \ldots, x_D)$. Tako lahko definiramo vse $v_i$ in $\mathbf{x}^i$ za $i \in \{1, \ldots, D\}$, torej $v_i = \min\{v \mid (v_1, \ldots, v_{i-1}, v, x_{i+1}, \ldots, x_D) \in N(\mathcal{P})\}$ ter $\mathbf{x}^i = (v_1, \ldots, v_i, x_{i+1}, \ldots, x_D)$.

Naj bo $\mathbf{v} = \mathbf{x}^D = (v_1, v_2, \ldots, v_D)$. Očitno je $\mathbf{x} \succeq \mathbf{v}$, saj za vsak $i$ velja $x_i \geq v_i$. Torej velja $\mathbf{x} \in C(\mathbf{v})$. Pokažimo, da velja tudi $\mathbf{v} \in \mathcal{V}(\mathcal{P})$. Predpostavimo, da to ni res in da obstaja nek $\mathbf{z} \in N(\mathcal{P})$, tako da velja $\mathbf{v} \succ \mathbf{z}$. Torej za nek $i$ velja $v_i > z_i$. Definirajmo točko $\mathbf{y} = (v_1, \ldots, v_{i-1}, z_i, x_{i+1}, \ldots, x_D)$. Velja $\mathbf{y} \in N(\mathcal{P})$, saj je $\mathbf{y} \succeq \mathbf{z}$. Tu smo prišli do protislovja, saj smo pokazali da je $z_i < v_i$, ki je po definiciji najmanjša taka vrednost $v$ da je $(v_1, \ldots, v_{i-1}, v, x_{i+1}, \ldots, x_D) \in N(\mathcal{P})$. □

Ilustracija stožcev $C(\mathbf{v}), \mathbf{v} \in \mathcal{V}(\mathcal{P})$, za množico dvodimenzionalnih točk $\mathcal{P}$ je vidna na sliki 5.




Slika 5: Primer stožcev $C(\mathbf{v}^0), \ldots, C(\mathbf{v}^5)$ s svetlo rumeno barvo, katerih unija je enaka območju $N(\mathcal{P})$.

## 3.2 Problem v dveh dimenzijah

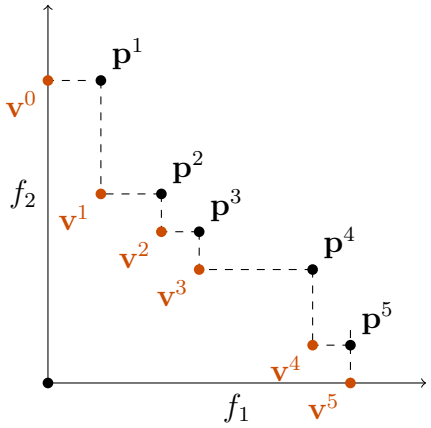
Za razumevanje intuicije algoritma v treh in več dimenzijah, si je najprej vredno ogledati kako lahko problem rešujemo v dveh dimenzijah [11].

Dana je množica $\mathcal{P}$, ki vsebuje $n$ paroma nedominiranih dvodimenzionalnih točk

$$\mathcal{P} = \{\mathbf{p}^1, \ldots, \mathbf{p}^n\},$$

kjer je $\mathbf{p}^i = (p_x^i, p_y^i) \gg \mathbf{0}$. Točke so urejene po zadnji, torej $y$ koordinati. Dana je tudi poizvedbena točka $\mathbf{q} = (q_x, q_y) \notin N(\mathcal{P})$.

Najprej izračunamo množico vpetih točk $\mathcal{V}(\mathcal{P}) = \{\mathbf{v}^0, \ldots, \mathbf{v}^n\}$ za množico točk $\mathcal{P}$. Za dve zaporedni nedominirani točki $(p_x^i, p_y^i)$ in $(p_x^{i+1}, p_y^{i+1})$, kjer je $p_x^i < p_x^{i+1}$ in $p_y^i > p_y^{i+1}$, izračunamo njuno vpeto točko kot $\mathbf{v}^i = (p_x^i, p_y^{i+1})$. Za prvo točko $(p_x^1, p_y^1)$ izračunamo vpeto točko kot $\mathbf{v}^0 = (0, p_y^1)$, za zadnjo točko pa kot $\mathbf{v}^n = (p_x^n, 0)$. Primer točk iz $\mathcal{P}$ in njim pripadajočih vpetih točk vidimo na sliki 6.

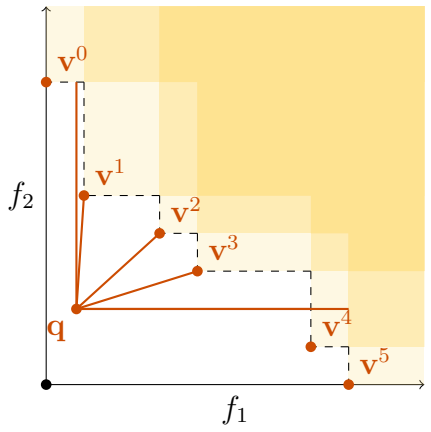



Slika 6: Primer množice petih paroma nedominiranih točk $\mathbf{p}^1, \ldots, \mathbf{p}^5$ ter njim pripadajočih vpetih točk $\mathbf{v}^0, \ldots, \mathbf{v}^5$.

Po trditvi 3.1 velja, da je razdalja poljubne točke poizvedbe $\mathbf{q}$ do nedominiranega območja $N(\mathcal{P})$ enaka minimalni razdalji točke $\mathbf{q}$ do vseh stožcev

$$d(\mathbf{q}, N(\mathcal{P})) = \min_{\mathbf{v} \in \mathcal{V}(\mathcal{P})} d(\mathbf{q}, C(\mathbf{v})),$$

kar vidimo tudi na sliki 7.




Slika 7: Primer točke $\mathbf{q}$ z označenimi razdaljami do vsakega izmed šestih stožcev $C(\mathbf{v}^0), \ldots, C(\mathbf{v}^5)$. Črti, ki označujeta razdaljo do stožcev $C(\mathbf{v}^4)$ in $C(\mathbf{v}^5)$, se delno prekrivata.

Razdaljo $d(\mathbf{q}, C(\mathbf{v}))$ pa izračunamo kot

$$d(\mathbf{q}, C(\mathbf{v})) = \sqrt{\max\{0, v_x - q_x\}^2 + \max\{0, v_y - q_y\}^2}.$$

Algoritem 1 prikazuje psevdokodo algoritma za računanje razdalje med točko in nedominiranim območjem v dveh dimenzijah. Kot smo pojasnili, najprej izračunamo vpete točke $\mathcal{V}(\mathcal{P})$, nato pa poiščemo minimalno razdaljo med točko $\mathbf{q}$ in stožci $C(\mathbf{v})$, ki jih definirajo vpete točke $\mathbf{v} \in \mathcal{V}(\mathcal{P})$.

**Algoritem 1** Algoritem ARRNO za računanje razdalje med točko in nedominiranim območjem.

**function** ARRNO($\mathcal{P}, \mathbf{q}$)
  $\mathcal{V}(\mathcal{P}) \leftarrow$ VPETE TOČKE($\mathcal{P}$)
  $d_{\min} \leftarrow \infty$
  **for all** $\mathbf{v} \in \mathcal{V}(\mathcal{P})$ **do**
    $d \leftarrow$ RAZDALJA DO STOŽCA($\mathbf{q}, \mathbf{v}$)
    **if** $d < d_{\min}$ **then**
      $d_{\min} \leftarrow d$
  **return** $d_{\min}$

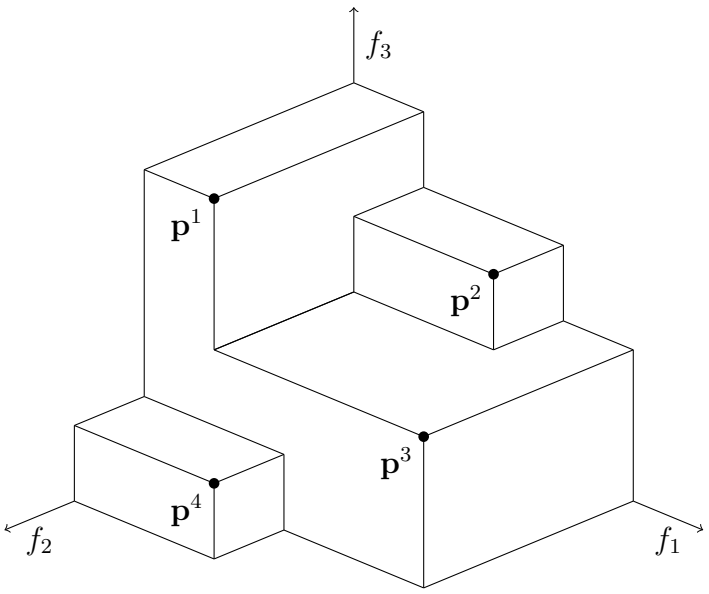
## 3.3 Problem v treh dimenzijah

Podobno kot pri problemu v dveh dimenzijah, je pri problemu v treh dimenzijah dana množica paroma nedominiranih točk

$$\mathcal{P} = \{\mathbf{p}^1, \ldots, \mathbf{p}^n\},$$

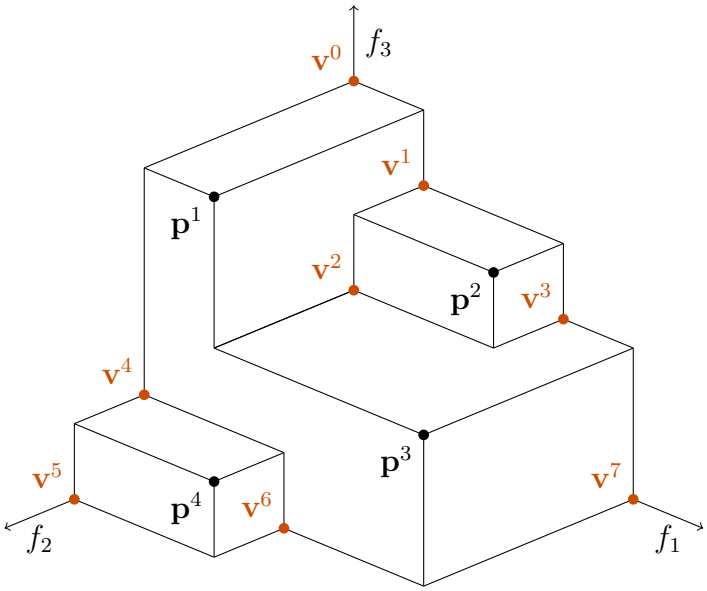
kjer je $\mathbf{p}^i = (p^i_x, p^i_y, p^i_z) \gg \mathbf{0}$. Točke so urejene padajoče po tretji koordinati. Prav tako je dana točka $\mathbf{q} = (q_x, q_y, q_z) \notin N(\mathcal{P})$.

Na sliki 8 vidimo primer množice točk $\mathcal{P}$ in območja, ki ga točke dominirajo v treh dimenzijah. Zanima nas torej razdalja med točko $\mathbf{q}$, ki se nahaja znotraj tega telesa, in površjem telesa.




Slika 8: Točke $\mathbf{p}^1$, $\mathbf{p}^2$, $\mathbf{p}^3$ in $\mathbf{p}^4$ definirajo dominirano območje v treh dimenzijah.

Po trditvi 3.1 velja, da se množica nedominiranih točk zapiše kot unija tridimenzionalnih stožcev, ki izhajajo iz točk $\mathcal{V}(\mathcal{P})$, kot vidimo na sliki 9.




Slika 9: Množica vpetih točk $\mathcal{V}(\mathcal{P}) = \{\mathbf{v}^0, \ldots, \mathbf{v}^7\}$ za dano množico štirih nedominiranih točk $\mathcal{P}$.

Razdaljo med točko in tridimenzionalnim stožcem enostavno razširimo na tri dimenzije

$$d(\mathbf{q}, C(\mathbf{v})) = \sqrt{\max\{0, v_x - q_x\}^2 + \max\{0, v_y - q_y\}^2 + \max\{0, v_z - q_z\}^2}.$$

Torej lahko uporabimo Algoritem 1 in se iskanje razdalje točke do nedominiranega prostora prevede na problem iskanja množice vpetih točk $\mathcal{V}(\mathcal{P})$. Vendar pa je iskanje množice $\mathcal{V}(\mathcal{P})$ v treh dimenzijah precej bolj zapleteno kot v dveh dimenzijah, kjer lahko za vsak par sosednjih točk enostavno izračunamo njuno vpeto točko. Vidimo sicer nekatere podrobnosti, točka $\mathbf{v}^2 = (p_x^1, p_y^2, p_z^3)$ iz slike 9 dobi po eno koordinato od vsake izmed njej "sosednjih" točk iz $\mathcal{P}$, vendar pojem sosednosti v treh dimenzijah ni jasno definiran. Prav tako bi se radi izognili kubični časovni zahtevnosti preverjanja vsake trojice točk $(\mathbf{p}^i, \mathbf{p}^j, \mathbf{p}^k)$ in računanja njihove vpete točke, če ta sploh obstaja. Zato za iskanje vpetih točk uporabimo algoritem pometanja.

### 3.3.1 Algoritmi pometanja

Algoritmi pometanja so pomemben pristop v računski geometriji, ki omogoča sistematično reševanje različnih geometrijskih problemov, kot so učinkovito iskanje presečišč daljic, določanje konveksne ovojnice točk, Delaunayeva triangulacija in številni drugi [4]. Osnovna ideja, skupna vsem algoritmom, je sistematična obdelava objektov v neki smeri premikanja oziroma pometanja ter sprotno posodabljanje stanja. Ko premica, ravnina (ali kak drug objekt s katerim pometamo) doseže novo točko, se stanje ustrezno posodobi. Ko prostor pometamo s hiperravnino, pri tem na vsakem koraku fiksiramo koordinato pometanja. S tem problem razdelimo na več podproblemov manjše dimenzije, kar olajša njegovo reševanje. Takemu pristopu pravimo pometanje dimenzije (angl. dimension sweep).

Pri analizi časovne zahtevnosti algoritmov pometanja je ključno določiti največje število posodobitev stanja ter časovno zahtevnost posamezne posodobitve. Pohitritve navadno dosežemo s podatkovnimi strukturami za stanje, ki podpirajo posodobitve v čim hitrejšem času, največkrat konstantnem oziroma logaritemskem, glede na velikost stanja.

### 3.3.2 Iskanje vpetih točk v treh dimenzijah

Vpete točke iščemo z algoritmom pometanja. Pometamo z navidezno ravnino vzporedno z ravnino $xy$ od $z = \infty$ proti $z = 0$ ter obravnavamo dogodke, ko se ravnina dotakne kakšne izmed točk $\mathbf{p}^i$. Stanje predstavimo z dvema množicama $S_\mathrm{p}, S_\mathrm{v} \subset \mathbb{R}^2$. Množica $S_\mathrm{p}$ vsebuje množico paroma nedominiranih projekcij že obiskanih točk na ravnino $xy$, množica $S_\mathrm{v}$ pa vsebuje vpete točke te množice, torej $S_\mathrm{v} = \mathcal{V}(S_\mathrm{p})$. Projekcijo točke $\mathbf{p} = (p_x, p_y, p_z)$ oziroma $\mathbf{v} = (v_x, v_y, v_z)$ na ravnino $xy$ označimo s $\overline{\mathbf{p}} = (p_x, p_y)$ oziroma $\overline{\mathbf{v}} = (v_x, v_y)$. Za pravilno obravnavo primera, ko ima več točk iz $\mathcal{P}$ enako $z$ koordinato, definiramo še zgoščevalno funkcijo $h$. Pri dodajanju nove točke $\overline{\mathbf{v}}$ v stanje $S_\mathrm{v}$ kot ključ v zgoščevalno funkcijo $h$ dodamo točko $\overline{\mathbf{v}}$ kot vrednost pa tisto vrednost $z$, pri kateri je bila točka $\overline{\mathbf{v}}$ dodana v stanje $S_\mathrm{v}$. Takšna struktura za vsako točko $\overline{\mathbf{v}}$ omogoča tudi možnost hitrega preverjanja ali je točka že bila dodana v $S_\mathrm{v}$.

Stanje $S_\mathrm{p}$ na začetku inicializiramo prazno, stanje $S_\mathrm{v}$ pa vsebuje točko $\overline{\mathbf{v}}^0 = (0, 0)$. Prav tako inicializirimo prazno množico $\mathcal{V}(\mathcal{P})$, kamor bomo tekom algoritma dodajali že izračunane vpete točke. Nato začnemo obravnavo točk. Na $i$-tem koraku, torej ko obravnavamo točko $\mathbf{p}^i$, naredimo sledeče:

1. Iz stanja $S_\mathrm{v}$ odstranimo množico dvodimenzionalnih točk $\overline{\mathcal{V}}_i$, ki jih točka $\overline{\mathbf{p}}^i$ strogo dominira.

2. V izhodno množico $\mathcal{V}(\mathcal{P})$ dodamo množico točk $\{(\overline{v}_x, \overline{v}_y, p_z^i) \mid (\overline{v}_x, \overline{v}_y) \in \overline{\mathcal{V}}_i \wedge h((\overline{v}_x, \overline{v}_y)) \neq p_z^i\}$. Točk za katere velja $h((\overline{v}_x, \overline{v}_y)) = p_z^i$ v $\mathcal{V}(\mathcal{P})$ ne dodamo, saj ne ustrezajo definiciji vpetih točk.

3. Točko $\overline{\mathbf{p}}^i$ dodamo v stanje $S_\mathrm{p}$, iz katerega odstranimo vse točke, ki jih dominira.

4. Stanju $S_\mathrm{v}$ dodamo vpeti točki, ki sta nastali ob dodajanju $\overline{\mathbf{p}}^i$ v stanje $S_\mathrm{p}$. Za vsako izmed točk, nastavimo vrednost funkcije $h$ na $p_z^i$.

Te korake ponovimo za vsako točko $\mathbf{p}^i \in \mathcal{P}$, po padajočem vrstnem redu zadnje koordinate. Po zadnjem koraku dodamo vsem točkam $\overline{\mathbf{v}}$, ki so še vedno v stanju $S_\mathrm{v}$ tretjo koordinato 0 ter jih dodamo v izhodno množico vpetih točk $\mathcal{V}(\mathcal{P})$. Podrobno psevdokodo predstavimo v algoritmu 2. Funkcija ODSTRANI DOMINIRANE$(S_\mathrm{v}, \overline{\mathbf{p}})$ odstrani in vrne točke iz $S_\mathrm{v}$, ki jih točka $\overline{\mathbf{p}}$ dominira, funkcija DODAJ$(S_\mathrm{p}, \overline{\mathbf{p}})$ v stanje $S_\mathrm{p}$ doda točko $\overline{\mathbf{p}}$ ter pri tem odstrani dominirane točke, funkcija NOVI VPETI TOČKI$(S_\mathrm{p}, \overline{\mathbf{p}})$ pa vrne vpeti točki, ki nastaneta z dodajanjem $\overline{\mathbf{p}}$ v $S_\mathrm{p}$.

Intuitivno si lahko delovanje algoritma predstavljamo kot zaporedno rezanje telesa iz slike 9 na različnih višinah $z$, kjer stanji $S_\mathrm{p}$ in $S_\mathrm{v}$ hranita obliko trenutnega prereza.

**Algoritem 2** Računanje vpetih točk v treh dimenzijah

```
1  function VPETE TOČKE 3D(P)
2      V(P) ← {}
3      S_p ← {}
4      S_v ← {(0,0)}
5      h((0,0)) ← ∞
6      for p = p^1, ..., p^n do                ▷ Točke so urejene po z koordinati
7          S_v, V̄ ← ODSTRANI DOMINIRANE(S_v, p̄)
8          for all v̄ ∈ V̄ do
9              if p̄ ≫ v̄ ∧ p_z < h(v̄) then
10                 V(P) ← V(P) ∪ {(v̄_x, v̄_y, p_z)}
11         S_p ← DODAJ(S_p, p̄)
12         {ṽ_1, ṽ_2} ← NOVI VPETI TOČKI(S_p, p̄)
13         for all v ∈ {ṽ_1, ṽ_2} do
14             if v ∉ h then
15                 h(v) ← p_z
16                 S_v ← S_v ∪ {v}
17     for all v ∈ S_v do
18         V(P) ← (v_x, v_y, 0)
19     return V(P)
```

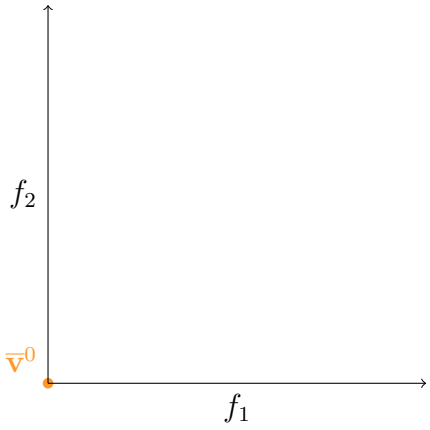
**Primer delovanja** Oglejmo si delovanje algoritma na primeru točk, ki smo jih videli že na sliki 8, torej $\mathbf{p}^1 = (1, 3, 4)$, $\mathbf{p}^2 = (3, 1, 3)$, $\mathbf{p}^3 = (4, 3, 2)$ in $\mathbf{p}^4 = (2, 4, 1)$. Na sliki 10 poleg vsakega koraka dodamo vizualizacijo, ki prikazuje stanji $S_\mathrm{p}$ in $S_\mathrm{v}$ po obravnavi dogodka.

a) Stanje $S_\mathrm{p}$ inicializiramo prazno, stanje $S_\mathrm{v}$ pa na začetku vsebuje točko $\overline{\mathbf{v}}^0 = (0, 0)$, kjer je $h(\overline{\mathbf{v}}^0) = \infty$.

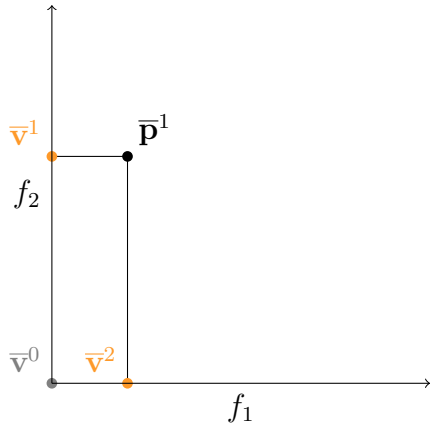
b) Na prvem koraku obravnavamo točko $\mathbf{p}^1 = (1, 3, 4)$. Točko $\overline{\mathbf{p}}^1$ dodamo v stanje $S_\mathrm{p}$ ter iz stanja $S_\mathrm{v}$ odstranimo vpeto točko $\overline{\mathbf{v}}^0$, ki jo točka $\overline{\mathbf{p}}^1$ strogo dominira. Odstranjeni točki $\overline{\mathbf{v}}^0$ dodamo tretjo koordinato točke $\mathbf{p}^1$, da dobimo točko $\mathbf{v}^0 = (\overline{v}^0_x, \overline{v}^0_y, p^1_z) = (0, 0, 4)$, ki jo dodamo v množico vpetih točk $\mathcal{V}(\mathcal{P})$. Po dodajanju točke $\overline{\mathbf{p}}^1$ v $S_\mathrm{p}$, dobimo dve novi vpeti točki $\overline{\mathbf{v}}^1 = (0, 3)$ in $\overline{\mathbf{v}}^2 = (1, 0)$, ki ju dodamo v $S_\mathrm{v}$. Za točki $\overline{\mathbf{v}}^1$ in $\overline{\mathbf{v}}^2$ nastavimo vrednost funkcije $h$ na $p_z = 4$.

c) Na drugem koraku obravnavamo točko $\mathbf{p}^2 = (3, 1, 3)$. V stanje $S_\mathrm{p}$ dodamo njeno projekcijo $\overline{\mathbf{p}}^2$, ki dominira točko $\overline{\mathbf{v}}^2$ iz $S_\mathrm{v}$. Torej v množico vpetih točk dodamo točko $\mathbf{v}^2 = (\overline{v}^2_x, \overline{v}^2_y, p^2_z) = (1, 0, 3)$. Tudi tokrat dobimo dve novi vpeti točki $\overline{\mathbf{v}}^3 = (1, 1)$ in $\overline{\mathbf{v}}^4 = (3, 0)$, ki ju dodamo v stanje $S_\mathrm{v}$, ter ustrezno posodobimo $h$.

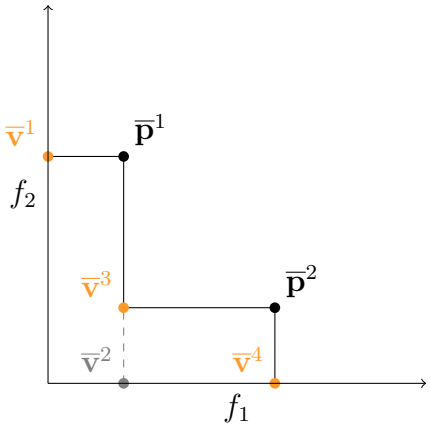
d) Naslednjo obravnavamo točko $\mathbf{p}^3 = (4, 3, 2)$. Projekcija $\overline{\mathbf{p}}^3$ strogo dominira točki $\overline{\mathbf{v}}^3$ in $\overline{\mathbf{v}}^4$, ne pa tudi točke $\overline{\mathbf{v}}^1$, saj imata enako $y$ koordinato. Točki $(\overline{v}^3_x, \overline{v}^3_y, p^3_z) = (1, 1, 2)$ in $(\overline{v}^4_x, \overline{v}^4_y, p^3_z) = (3, 0, 2)$ torej dodamo v $\mathcal{V}(\mathcal{P})$. V $S_\mathrm{p}$ dodamo $\overline{\mathbf{p}}^3$, v $S_\mathrm{v}$ pa novonastalo vpeto točko $\overline{\mathbf{v}}^5 = (4, 0)$, za katero velja $h(\overline{\mathbf{v}}^5) = 2$.



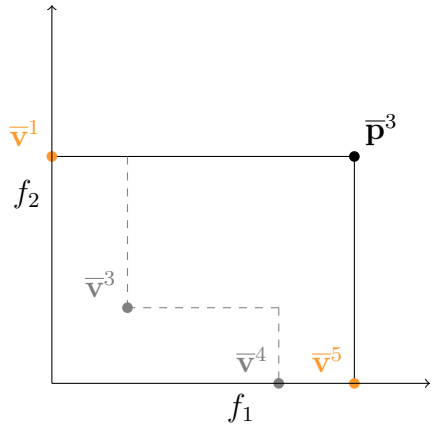

(a) Stanji pred obravnavo točke $\mathbf{p}^1$: $S_\mathrm{p} = \{\}$, $S_\mathrm{v} = \{\overline{\mathbf{v}}^0\}$.



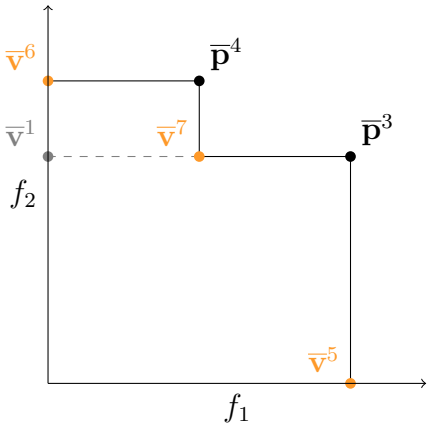

(b) Stanji po obravnavi točke $\mathbf{p}^1$: $S_\mathrm{p} = \{\overline{\mathbf{p}}^1\}$, $S_\mathrm{v} = \{\overline{\mathbf{v}}^1, \overline{\mathbf{v}}^2\}$.



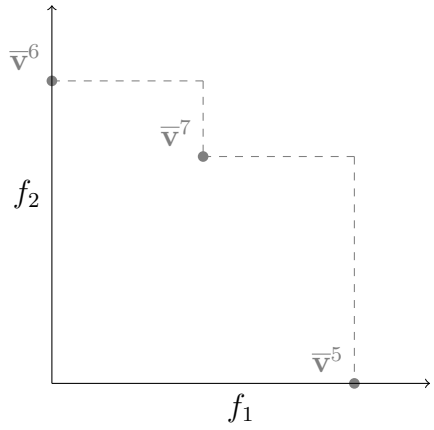

(c) Stanji po obravnavi točke $\mathbf{p}^2$: $S_\mathrm{p} = \{\overline{\mathbf{p}}^1, \overline{\mathbf{p}}^2\}$, $S_\mathrm{v} = \{\overline{\mathbf{v}}^1, \overline{\mathbf{v}}^3, \overline{\mathbf{v}}^4\}$.




(d) Stanji po obravnavi točke $\mathbf{p}^3$: $S_\mathrm{p} = \{\overline{\mathbf{p}}^3\}$, $S_\mathrm{v} = \{\overline{\mathbf{v}}^1, \overline{\mathbf{v}}^5\}$.




(e) Stanji po obravnavi točke $\mathbf{p}^4$: $S_\mathrm{p} = \{\overline{\mathbf{p}}^3, \overline{\mathbf{p}}^4\}$, $S_\mathrm{v} = \{\overline{\mathbf{v}}^5, \overline{\mathbf{v}}^6, \overline{\mathbf{v}}^7\}$.




(f) Iz stanja $S_\mathrm{v}$ odstranimo preostale točke $\overline{\mathbf{v}}^5$, $\overline{\mathbf{v}}^6$ in $\overline{\mathbf{v}}^7$.

Slika 10: Slike prikazujejo stanje $S_\mathrm{p}$ (črna barva), točke v stanju $S_\mathrm{v}$ (rumena barva) ter odstranjene točke iz $S_\mathrm{v}$ v trenutnem koraku (siva barva) na vsakem koraku algoritma.

e) Zadnjo obravnavamo točko $\mathbf{p}^4 = (2, 4, 1)$. Točka $\overline{\mathbf{p}}^4$ strogo dominira $\overline{\mathbf{v}}^1$, ki jo odstranimo iz $S_\mathrm{v}$. V $\mathcal{V}(\mathcal{P})$ torej dodamo $(v_x^1, v_y^1, p_z^4) = (0, 3, 1)$, v $S_\mathrm{p}$ pa $\overline{\mathbf{p}}^4$. Na koncu v $S_\mathrm{v}$ dodamo še novi dve vpeti točki $\overline{\mathbf{v}}^6 = (0, 4)$ ter $\overline{\mathbf{v}}^7 = (2, 3)$, za kateri je vrednost funkcije $h$ enaka 1.

f) Po obravnavi vseh točk iz $\mathcal{P}$, odstranimo preostale točke $\overline{\mathbf{v}}^5$, $\overline{\mathbf{v}}^6$ in $\overline{\mathbf{v}}^7$ iz $S_\mathrm{v}$. Vsaki izmed njih dodamo tretjo koordinato $z = 0$, da dobimo točke $\mathbf{v}^5 = (4, 0, 0)$, $\mathbf{v}^6 = (0, 4, 0)$ in $\mathbf{v}^7 = (2, 3, 0)$, ki jih dodamo v $\mathcal{V}(\mathcal{P})$.

Tako dobimo množico vpetih točk $\mathcal{V}(\mathcal{P})$, ki je, do vrstnega reda natančno, enaka množici vpetih točk iz slike 9.

## 3.4 Problem v več dimenzijah

Problem v $D$ dimenzijah, $D > 3$, definiramo na enak način kot v dveh in treh dimenzijah. Dana je torej množica paroma nedominiranih točk

$$\mathcal{P} = \{\mathbf{p}^1, \ldots, \mathbf{p}^n\},$$

kjer je $\mathbf{p}^i = (p_1^i, \ldots, p_D^i) \gg \mathbf{0}$. Točke so urejene padajoče po zadnji koordinati. Prav tako je dana poizvedbena točka $\mathbf{q} = (q_1, \ldots, q_D) \notin N(\mathcal{P})$.

Enako kot pri problemu v dveh in treh dimenzijah, tudi problem v višjih dimenzijah rešujemo z algoritmom 1. Funkcijo za računanje razdalje med točko $\mathbf{q}$ in stožcem $C(\mathbf{v})$ enostavno posplošimo za poljubno dimenzijo $D$.

$$d(\mathbf{q}, C(\mathbf{v})) = \sqrt{\sum_{i=1}^{D} \max\{0, v_i - q_i\}^2}.$$

Tudi tokrat nam torej preostane le še pravilno posplošiti funkcijo, ki za množico $\mathcal{P}$ izračuna vpete točke $\mathcal{V}(\mathcal{P})$ za poljubno višjo dimenzijo.

### 3.4.1 Iskanje vpetih točk v višjih dimenzijah

Za iskanje vpetih točk v višji dimenziji $D$ uporabimo algoritem, podoben algoritmu za iskanje vpetih točk v treh dimenzijah. Prve tri korake obravnave točke $\mathbf{p} \in \mathcal{P}$ iz razdelka 3.3.2 enostavno posplošimo v poljubno višjo dimenzijo. Pri četrtem koraku, kjer iščemo novonastale vpete točke ob dodajanju $\overline{\mathbf{p}}$ v $S_\mathrm{p}$, pa pristopimo malo drugače. Namesto da izračunamo le novonastale vpete točke ob dodajanju $\overline{\mathbf{p}}$ v $S_\mathrm{p}$, izračunamo kar vse vpete točke za množico $S_\mathrm{p}$ z rekurzivnim reševanjem problema v dimenziji $D - 1$. Računanje le novo nastalih vpetih točk, bi bilo skoraj gotovo hitrejše, vendar takega algoritma ne znamo zasnovati.

Analogno kot pri algoritmu v treh dimenzijah, tudi zdaj uvedemo označbo $\overline{\mathbf{p}} = (p_1, \ldots, p_{D-1})$, ki predstavlja projekcijo točke $\mathbf{p} = (p_1, \ldots, p_D)$ na hiperprostor prvih $D - 1$ dimenzij. Prav tako definiramo zgoščevalno funkcijo $h(\overline{\mathbf{v}})$, ki vsaki točki $\overline{\mathbf{v}}$ iz stanja $S_\mathrm{v}$ priredi tisto vrednost zadnje koordinate, pri kateri je bila točka dodana v stanje $S_\mathrm{v}$.

Algoritem 3 prikazuje psevdokodo za računanje vpetih točk v poljubni dimenziji $D \geq 3$. Podobno kot v treh dimenzijah funkcija ODSTRANI DOMINIRANE($S_\mathrm{v}, \overline{\mathbf{p}}$)

**Algoritem 3** Računanje vpetih točk v $D$ dimenzijah

```
1  function VPETE TOČKE(P, D)
2      if D = 3 then
3          return VPETE TOČKE 3D(P)
4      V(P) ← {}
5      S_p ← {}
6      S_v ← {(0, ..., 0)}
7      h((0, ..., 0)) ← ∞
8      for p = p^1, ..., p^n do                ▷ Točke so urejene po zadnji koordinati
9          S_v, V̄ ← ODSTRANI DOMINIRANE(S_v, p̄)
10         for all v̄ ∈ V̄ do
11             if p̄ ≫ v̄ ∧ p_D < h(v̄) then
12                 V(P) ← V(P) ∪ {(v̄_1, ..., v̄_{d-1}, p_D)}
13         S_p ← DODAJ(S_p, p̄)
14         S_v ← VPETE TOČKE(S_p, D − 1)
15         for all v̄ ∈ S_v do
16             if v̄ ∉ h then
17                 h(v̄) ← p_D
18     for all v̄ ∈ S_v do
19         V(P) ← (v_1, ..., v_{d-1}, 0)
20     return V(P)
```

odstrani in vrne točke iz $S_\mathrm{v}$, ki jih točka $\overline{\mathbf{p}}$ dominira, funkcija $\textsc{Dodaj}(S_\mathrm{p}, \overline{\mathbf{p}})$ v stanje $S_\mathrm{p}$ doda točko $\overline{\mathbf{p}}$ ter pri tem iz stanja odstrani točke, ki jih $\overline{\mathbf{p}}$ dominira.

Verjamemo, da algoritem $\textsc{Vpete točke}(\mathcal{P}, D)$ pravilno izračuna vpete točke za vsako dimenzijo $D$ in poljubno množico točk $\mathcal{P}$, vendar tega ne znamo dokazati. Zaradi tega algoritem testiramo eksperimentalno, kar opišemo v naslednjem poglavju.

# 4 Testiranje implementacije

V tem poglavju predstavimo postopek, s katerim testiramo pravilnost implementacije algoritma ARRNO. Kljub izpeljavi algoritma v prejšnjem poglavju, je testiranje vseeno pomembno. S tem pokažemo, da smo algoritem pravilno generalizirali v višje dimenzije, da deluje za posebne primere točk (na primer, kadar imajo točke iz $\mathcal{P}$ enako eno ali več koordinat), ter da se v implementacijo ni prikradla kakšna napaka.

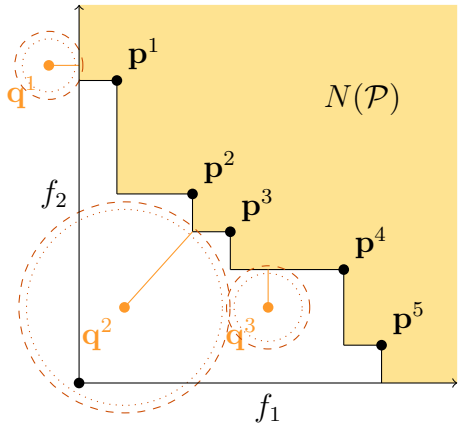
## 4.1 Ideja testiranja

Naj bo $\mathcal{P}$ množica paroma nedominiranih točk in $\mathbf{q}$ točka poizvedbe. Za vsak $r \geq 0$ velja

$$S(\mathbf{q}, r) \cap N(\mathcal{P}) \neq \varnothing \iff d(\mathbf{q}, N(\mathcal{P})) \leq r,$$

kjer je $S(\mathbf{q}, r)$ sfera s središčem v $\mathbf{q}$ in radijem $r$.

Osnovna ideja testiranja algoritma je torej, da za naključno izbrano kombinacijo množice $\mathcal{P}$ in poizvedbene točke $\mathbf{q}$ z algoritmom ARRNO izračunamo razdaljo $r_{\mathrm{A}}$. Nato pa za nek majhen $\delta > 0$ preverimo, ali obstajata preseka $S(\mathbf{q}, r_{\mathrm{A}} - \delta) \cap N(\mathcal{P})$ in $S(\mathbf{q}, r_{\mathrm{A}} + \delta) \cap N(\mathcal{P})$. Primer treh točk poizvedbe in ustrezajočih sfer v dveh dimenzijah vidimo na sliki 11. Če velja da je presek $S(\mathbf{q}, r_{\mathrm{A}} - \delta) \cap N(\mathcal{P})$ prazen in $S(\mathbf{q}, r_{\mathrm{A}} + \delta) \cap N(\mathcal{P})$ neprazen, potem vemo, da je napaka algoritma največ $\delta$. Če to velja za majhen $\delta$ pri veliko naključnih množicah $\mathcal{P}$ in poizvedbenih točkah $\mathbf{q}$, smo lahko o pravilnosti algoritma precej prepričani.




Slika 11: Tri točke poizvedbe $\mathbf{q}^1$, $\mathbf{q}^2$, $\mathbf{q}^3$, skupaj z njihovimi razdaljami do nedominiranega območja ter ustrezajočimi sferami $S(\mathbf{q}, r_{\mathrm{A}} + \delta)$ s črtkano črto in $S(\mathbf{q}, r_{\mathrm{A}} - \delta)$ s točkasto črto.

## 4.2 Diskretizacija sfere

Da lahko računamo presek sfere z $N(\mathcal{P})$, sfero diskretiziramo. Diskretizacija sfere $S(\mathbf{q}, r)$ je množica točk, označimo jo z $\Sigma(\mathbf{q}, r, \varepsilon)$, ki ležijo na $S(\mathbf{q}, r)$ in za katere velja

$$\forall \mathbf{x} \in S(\mathbf{q}, r): \quad \min_{\mathbf{s} \in \Sigma(\mathbf{q}, r, \varepsilon)} ||\mathbf{x} - \mathbf{s}|| \leq \varepsilon.$$

Presek med diskretno sfero $\Sigma(\mathbf{q}, r, \varepsilon)$ in $N(\mathcal{P})$ lahko nato izračunamo enostavno

$$\Sigma(\mathbf{q}, r, \varepsilon) \cap N(\mathcal{P}) = \{\mathbf{s} \in \Sigma(\mathbf{q}, r, \varepsilon) \mid \mathbf{s} \succeq 0 \wedge \neg(\mathcal{P} \gg \mathbf{s})\}.$$

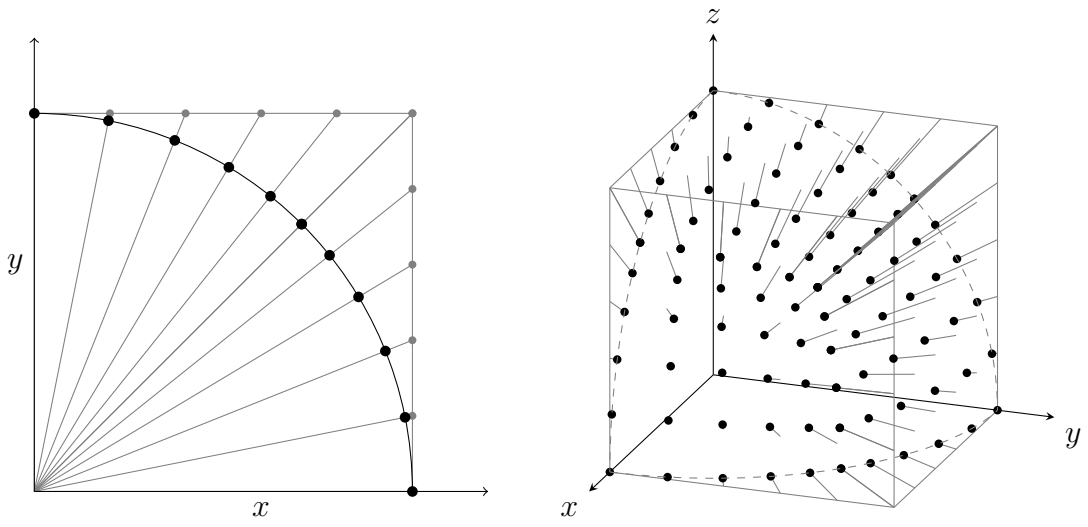
Za lažjo notacijo v tem razdelku brez škode za splošnost predpostavljamo, da je sfera centrirana v koordinatnem izhodišču. Prav tako diskretiziramo le pozitivni del sfere, torej točke ki imajo vse koordinate nenegativne, kar označimo s $\Sigma^+(\mathbf{0}, r, \varepsilon)$. V trditvi 4.1 namreč pokažemo, da to zadošča za preverjanje preseka z nedominiranim območjem.

**Trditev 4.1.** *Vse komponente vektorja od poljubne točke $\boldsymbol{q} \notin N(\mathcal{P})$ do njej najbližje točke $\boldsymbol{z} \in N(\mathcal{P})$ so nenegativne.*

*Dokaz.* Predpostavimo, da za neko množico $\mathcal{P}$ in točko $\mathbf{q} = (q_1, \ldots, q_D) \notin N(\mathcal{P})$ obstaja točka $\mathbf{z} = (z_1, \ldots, z_D) \in N(\mathcal{P})$, ki je najbližja točki $\mathbf{q}$ izmed točk v $N(\mathcal{P})$ ter da je $i$-ta komponenta vektorja $\mathbf{w} = (z_1 - q_1, \ldots, z_D - q_D)$ od $\mathbf{q}$ do $\mathbf{z}$ negativna.

Potem definirajmo vektor $\mathbf{w'} = (z_1 - q_1, \ldots 0, \ldots, z_D - q_D)$, ki je enak vektorju $\mathbf{w}$, le da ima $i$-to komponento enako 0, ter novo točko $\mathbf{z'} = \mathbf{q} + \mathbf{w'}$. Za točko $\mathbf{z'} = (z_1, \ldots, z_{i-1}, q_i, z_{i+1}, \ldots, z_D)$ potem velja $\mathbf{z'} \succ \mathbf{z}$, saj je po predpostavki $q_i > z_i$, vse ostale komponente pa so enake. Ker je $\mathbf{z} \in N(\mathcal{P})$ in $\mathbf{z'} \succ \mathbf{z}$, je torej tudi $\mathbf{z'} \in N(\mathcal{P})$. Prav tako je očitno, da je $d(\mathbf{q}, \mathbf{z'}) = ||\mathbf{w'}|| < ||\mathbf{w}|| = d(\mathbf{q}, \mathbf{z})$, torej $\mathbf{z}$ ni najbližja točka $\mathbf{q}$ izmed točk v $N(\mathcal{P})$, kar nas privede v protislovje. □

Preprost način za konstrukcijo diskretne sfere $\Sigma^+(\mathbf{0}, r, \varepsilon)$ je, da diskretiziramo površino sferi očrtane kocke $\Gamma^+(\mathbf{0}, r, \varepsilon)$, nato pa točke projeciramo na sfero, kot je vidno na sliki 12.




(a) Diskretizacija roba kocke in projekcija na sfero v dveh dimenzijah.

(b) Diskretizacija roba kocke in projekcija na sfero v treh dimenzijah.

Slika 12: Prikaz delovanja algoritma za diskretizacijo sfere v dveh in treh dimenzijah.

Pozitivni del hiperkocke v $D$ dimenzijah diskretiziramo tako, da združimo diskretizacijo vsake izmed $D$ pozitivnih stranic hiperkocke. Diskretizacija $i$-te stranice

hiperkocke s stranico dolžine $r$ na $k+1$ točk je množica točk

$$\left\{0, \frac{1}{k}r, \frac{2}{k}r, \ldots, \frac{k-1}{k}r, r\right\}^{i-1} \times \{r\} \times \left\{0, \frac{1}{k}r, \frac{2}{k}r, \ldots, \frac{k-1}{k}r, r\right\}^{D-i}.$$

Torej je

$$\Gamma^+(\mathbf{0}, r, \varepsilon) = \bigcup_{i=1}^{D} \left\{0, \frac{1}{k}r, \frac{2}{k}r, \ldots, \frac{k-1}{k}r, r\right\}^{i-1} \times \{r\} \times \left\{0, \frac{1}{k}r, \frac{2}{k}r, \ldots, \frac{k-1}{k}r, r\right\}^{D-i},$$

iz česar pa enostavno dobimo diskretizacijo sfere $\Sigma^+(\mathbf{q}, r, \varepsilon)$, tako da za vsako točko $\gamma \in \Gamma^+(\mathbf{0}, r, \varepsilon)$ izračunamo njeno projekcijo $\sigma$ na sfero

$$\sigma = \frac{\gamma}{||\gamma||}r.$$

Potrebno je le še določiti gostoto mreže, torej dolžino delitvenega intervala $\frac{r}{k}$. V dimenziji $D$ najbolj oddaljena točka $\mathbf{x}$ od diskretne mreže leži ravno na sredini med točkami mreže, kot vidimo na sliki 13, z maksimalno razdaljo enako

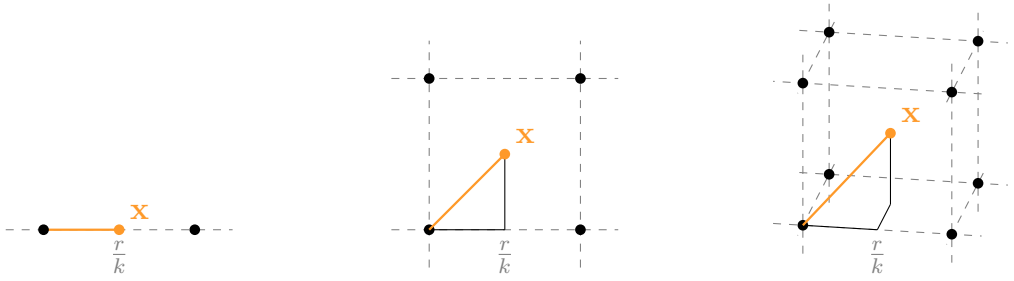
$$d_{\max} = \sqrt{(D-1)\frac{r^2}{(2k)^2}} = \frac{r}{2k}\sqrt{D-1}.$$

Ker želimo da je $d_{\max} \leq \varepsilon$ mora veljati

$$\frac{r}{k} \leq \frac{2\varepsilon}{\sqrt{D-1}},$$

kar velja za

$$k = \left\lceil \frac{r\sqrt{D-1}}{2\varepsilon} \right\rceil.$$




Slika 13: Primeri diskretizacij dela stranice dvodimenzionalne kocke na levi, tridimenzionalne kocke na sredini in štiridimenzionalne kocke na desni, skupaj s točko $\mathbf{x}$, ki je od diskretizacije najbolj oddaljena.

Za dobro izbiro $k$ torej velja, da je vsaka točka $\mathbf{x}$ iz površine hiperkocke, največ za $\varepsilon$ oddaljena od diskretizacije.

**Trditev 4.2.** *Za poljubni $D$-dimenzionalni točki $A$ in $B$ iz zunanjosti sfere in njuni projekciji na sfero $A'$ in $B'$ velja*

$$||A' - B'|| \leq ||A - B||.$$

*Dokaz.* Osredotočimo se lahko le na ravnino, ki gre skozi izhodišče sfere in točki $A$ in $B$, ki je prikazana na sliki 14. Naj bo $\theta$ kot med točkama $A$ in $B$ v tej ravnini. Po kosinusnem izreku je potem

$$||A' - B'||^2 = 2r^2 - 2r^2 \cos\theta,$$

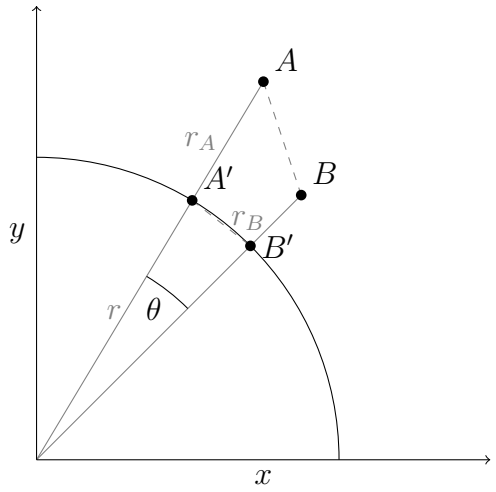
kjer je $r$ radij sfere in

$$||A - B||^2 = (r + r_A)^2 + (r + r_B)^2 - 2(r + r_A)(r + r_B)\cos\theta,$$

kjer sta $r + r_A$ in $r + r_B$ razdalji med izhodiščem sfere in točko $A$ oziroma $B$. Potem je

$$||A - B||^2 - ||A' - B'||^2 = rr_A(2 - 2\cos\theta) + rr_B(2 - 2\cos\theta) + r_A^2 + r_B^2 - 2r_A r_B \cos\theta$$

Ker sta $r_A, r_B \geq 0$ očitno velja $rr_A(2 - 2\cos\theta) \geq 0$ in $rr_B(2 - 2\cos\theta) \geq 0$. Po kosinusnem izreku enako velja za $r_A^2 + r_B^2 - 2r_A r_B \cos\theta$, saj je to ravno kvadrat dolžine nasprotne stranice trikotnika s stranicama $r_A$ in $r_B$ ter kotom $\theta$ med njima. Ker je torej $||A - B||^2 \geq ||A' - B'||^2$ je tudi $||A - B|| \geq ||A' - B'||$. □




Slika 14: Slika prikazuje točki $A$ in $B$, njuni projekciji na sfero $A'$ in $B'$ ter razdalje $r$, $r_A$ in $r_B$.

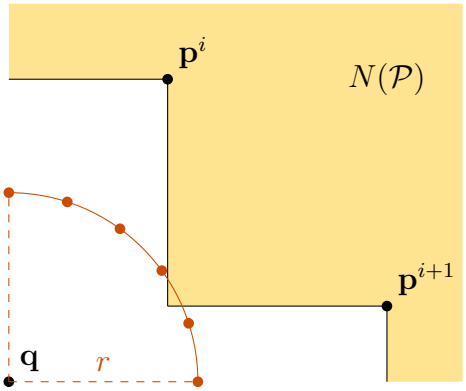
Po trditvi 4.2 torej tudi za diskretizacijo sfere velja, da je vsaka točka $\mathbf{x}$ iz površine sfere največ za $\varepsilon$ oddaljena od njene diskretizacije.

## 4.3 Gostota diskretizacije

Za dani $\delta$ želimo izbrati tak $\varepsilon$, da je gostota diskretizacije dovolj velika in velja

$$d(\mathbf{q}, N(\mathcal{P})) \leq r \implies \Sigma(\mathbf{q}, r + \delta, \varepsilon) \cap N(\mathcal{P}) \neq \varnothing.$$

Sicer bi se nam lahko zgodilo, da zaradi nesrečne izbire diskretizacije v preseku ne bi bilo nobene točke, kot je vidno na sliki 15. Očitno je, da bomo za čim manjši $\delta$ potrebovali tem gostejšo diskretizacijo sfere.

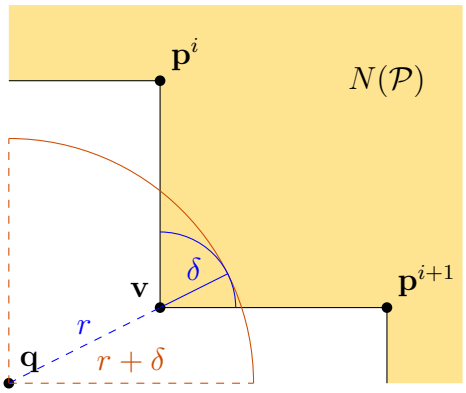



Slika 15: Gostota diskretizacije sfere $S(\mathbf{q}, r)$ je premajhna, da bi kakšna izmed točk diskretizacije $\Sigma(\mathbf{q}, r, \varepsilon)$ padla v nedominirano območje $N(\mathcal{P})$, kljub temu da je $d(\mathbf{q}, N(\mathcal{P})) < r$.

**Trditev 4.3.** *Za izbiro* $\varepsilon \leq \frac{\delta}{\sqrt{D}}$ *velja*

$$d(\boldsymbol{q}, N(\mathcal{P})) \leq r \implies \Sigma(\boldsymbol{q}, r + \delta, \varepsilon) \cap N(\mathcal{P}) \neq \varnothing.$$

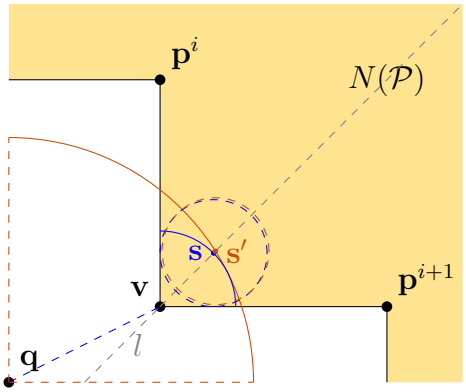
*Dokaz.* Oglejmo si sliko 16, kjer vidimo primer točke $\mathbf{q}$, ki je do najbližje točke $\mathbf{v} \in N(\mathcal{P})$ oddaljena za $r$. Prikazan je tudi pozitivni del sfere $S(\mathbf{q}, r+\delta)$ s središčem v $\mathbf{q}$ in radijem $r + \delta$ ter pozitivni del sfere $S(\mathbf{v}, \delta)$ s središčem v $\mathbf{v}$ in radijem $\delta$. Po trikotniški neenakosti sledi, da so vse točke na $S(\mathbf{q}, r + \delta)$ od $\mathbf{v}$ oddaljene več ali enako $\delta$.




Slika 16: Slika prikazuje primer točke $\mathbf{q}$ ter njej najbližjo točko iz $N(P)$, točko $\mathbf{v}$. Z oranžno barvo je prikazana testna sfera $S(\mathbf{q}, r + \delta)$, z modro barvo pa $S(\mathbf{v}, \delta)$.

Skozi $\mathbf{v}$ nato potegnemo premico $l$ pod kotom 45° na vse koordinatne osi, ki seka $S(\mathbf{v}, \delta)$ v točki $\mathbf{s}$, kot vidimo na sliki 17. Po podobnem razmisleku, kot na sliki 13 je jasno, da je vektor od $\mathbf{v}$ do $\mathbf{s}$ enak $(\frac{\delta}{\sqrt{D}}, \ldots, \frac{\delta}{\sqrt{D}})$. Torej krogla $B(\mathbf{s}, \frac{\delta}{\sqrt{D}})$ v celoti leži znotraj $N(\mathcal{P})$. Naj bo $I = S(\mathbf{v}, \delta) \cap B(\mathbf{s}, \frac{\delta}{\sqrt{D}})$ del sfere $S(\mathbf{v}, \delta)$, ki leži znotraj krogle $B(\mathbf{s}, \frac{\delta}{\sqrt{D}})$. Množica $I \subset N(\mathcal{P})$ vsebuje vse točke iz $S(\mathbf{v}, \delta)$, ki so od $\mathbf{s}$ oddaljene manj kot $\frac{\delta}{\sqrt{D}}$. Torej pri vsaki diskretizaciji $S(\mathbf{v}, \delta)$ z $\varepsilon \leq \frac{\delta}{\sqrt{D}}$ obstaja točka diskretizacije, ki leži v $I$ torej leži tudi v $N(\mathcal{P})$.

Podobno lahko definiramo točko $\mathbf{s}'$, ki je na presečišču premice $l$ in sfere $S(\mathbf{q}, r + \delta)$. Ker je del sfere $S(\mathbf{q}, r + \delta)$, ki je znotraj $N(\mathcal{P})$, kvečjemu dlje od $\mathbf{v}$ kot $S(\mathbf{v}, \delta)$,




Slika 17: S sivo črto je prikazana premica $l$, ki gre skozi točko $\mathbf{v}$, ter seka modro oziroma oranžno sfero v točkah $\mathbf{s}$ ter $\mathbf{s}'$. Okrog točk $\mathbf{s}$ in $\mathbf{s}'$ sta prikazani sferi z radijem $\frac{\delta}{\sqrt{D}}$.

bo po enakem razmisleku veljalo, da je diskretizacija z $\varepsilon \leq \frac{\delta}{\sqrt{D}}$ dovolj natančna tudi za $S(\mathbf{q}, r + \delta)$.

□

## 4.4 Testiranje

Naj bo $r_{\mathrm{A}}$ z algoritmom izračunana razdalja med nedominiranim območjem dane množice $\mathcal{P}$ ter poizvedbene točke $\mathbf{q}$. Za nek majhen $\delta$ bi radi pokazali da velja $d(\mathbf{q}, N(\mathcal{P})) - \delta \leq r_{\mathrm{A}} \leq d(\mathbf{q}, N(\mathcal{P})) + \delta$.

### 4.4.1 Spodnja meja

Izberemo $\varepsilon = \frac{\delta}{\sqrt{D}}$ in preverimo presek $\Sigma(\mathbf{q}, r_{\mathrm{A}} + \delta, \varepsilon) \cap N(\mathcal{P})$. Če presek vsebuje kako točko, vemo da velja $d(\mathbf{q}, N(\mathcal{P})) \leq r_{\mathrm{A}} + \delta$ oziroma $d(\mathbf{q}, N(\mathcal{P})) - \delta \leq r_{\mathrm{A}}$. Sicer pa po trditvi 4.3 vemo, da je $d(\mathbf{q}, N(\mathcal{P})) > r_{\mathrm{A}}$ torej algoritem ne deluje pravilno.

### 4.4.2 Zgornja meja

Iz trditve 4.3 po kontrapoziciji velja

$$\Sigma(\mathbf{q}, r, \varepsilon) \cap N(\mathcal{P}) = \varnothing \implies d(\mathbf{q}, N(\mathcal{P})) > r - \delta,$$

pri izbiri $\varepsilon = \frac{\delta}{\sqrt{D}}$. Presek $\Sigma(\mathbf{q}, r_{\mathrm{A}}, \varepsilon) \cap N(\mathcal{P})$ ob pravilnem delovanju algoritma ni nujno prazen, zato preverimo presek $\Sigma(\mathbf{q}, r_{\mathrm{A}} - \mu, \varepsilon') \cap N(\mathcal{P})$, kjer je $\mu = \frac{\delta}{10^6}$, $\varepsilon' = \frac{\delta'}{\sqrt{D}}$ in $\delta' = \delta - \mu$. Če presek ni prazen, vemo da algoritem deluje narobe, sicer pa napako omejimo na

$$r_{\mathrm{A}} - \mu - (\delta - \mu) < d(\mathbf{q}, N(\mathcal{P})),$$

iz česar sledi

$$r_{\mathrm{A}} \leq d(\mathbf{q}, N(\mathcal{P})) + \delta.$$

### 4.4.3 Rezultati testiranja

Algoritem testiramo za dimenzije 3, 4, 5 in 6. Pri vsaki dimenziji ponovimo 100 poskusov z različnimi množicami $\mathcal{P}$ in točkami $\mathbf{q}$. Da dobimo množico $\mathcal{P}$, naključno izberemo 100 točk iz $[0,1]^D$ in odstranimo dominirane. Da dobimo točko $\mathbf{q}$, prav tako naključno izbiramo točke iz $[0,1]^D$, dokler $\mathcal{P}$ točke ne dominira. Nato z algoritmom ARRNO izračunamo razdaljo $r_{\mathrm{A}}$ med točko $\mathbf{q}$ ter nedominiranim območjem množice $\mathcal{P}$. Rezultate testiramo za čim manjše vrednosti $\delta$, kolikor nam omogočajo zmogljivosti strojne opreme. Ker za preverjanje preseka $\Sigma(\mathbf{q}, r, \varepsilon) \cap N(\mathcal{P})$ preverimo vsako kombinacijo točk iz $\Sigma(\mathbf{q}, r, \varepsilon)$ in iz $\mathcal{P}$, za to potrebujemo $O(|\Sigma(\mathbf{q}, r, \varepsilon)| \cdot |\mathcal{P}|)$ primerjav. Vrednost $\delta$ izberemo v odvisnosti od $r_{\mathrm{A}}$, da je število točk diskretizacije vedno enako. Prav tako za višje dimenzije izberemo večji $\delta$, saj se število točk diskretizacije z dimenzijo veča. Tabela 1 prikazuje vrednosti $\delta$ in velikost diskretizacije sfere za različne dimenzije.

| dimenzija | vrednost $\delta$ | št. točk diskretizacije |
|---|---|---|
| 3 | $0.001 \cdot r_{\mathrm{A}}$ | $\sim 4.5 \cdot 10^6$ |
| 4 | $0.01 \cdot r_{\mathrm{A}}$ | $\sim 21.6 \cdot 10^6$ |
| 5 | $0.05 \cdot r_{\mathrm{A}}$ | $\sim 25.5 \cdot 10^6$ |
| 6 | $0.2 \cdot r_{\mathrm{A}}$ | $\sim 9.9 \cdot 10^6$ |

Tabela 1: Tabela prikazuje izbrane vrednosti za $\delta$ pri različnih dimenzijah, povprečno število točk diskretizacije sfere z izbranim $\delta$ ter povprečno število vpetih točk.

Da testiramo tudi primer, ko imata dve ali več točk kakšno izmed koordinat enako, vse poskuse ponovimo, le da tokrat točke za množico $\mathcal{P}$ izbiramo naključno iz $\{0.1, 0.2, \ldots, 0.9, 1\}^D$. Tako je verjetnost, da imata dve točki neko koordinato enako precej večja. Poskuse prav tako ponovimo še za primer, ko točka $\mathbf{q} \notin [0,1]^D$. Tokrat jo naključno izberemo iz množice $[-1,1]^D \setminus [0,1]^D$.

Skupaj torej naključno ustvarimo 800 testnih problemov, na katerih preverimo delovanje. Testiranje na namiznem računalniku s 16 GB pomnilnika in frekvenco procesorja 3,60 GHz traja nekaj dni. V vseh primerih algoritem ARRNO deluje pravilno, kar kaže na to, da je zasnova in implementacija pravilna.

# 5 Računska zahtevnost

V tem poglavju podrobno analiziramo računsko zahtevnost algoritma v odvisnosti od velikosti množice nedominiranih točk $n = |\mathcal{P}|$ ter dimenzije problema $D$. Za lažjo analizo predpostavljamo, da je dimenzija $D$ konstanta, torej da lahko namesto $O(D)$ pišemo $O(1)$[2]. Predpostavka je smiselna, saj je računanje enostavnejših indikatorjev pri visokih dimenzijah zelo zamudno, tako da se v praksi le redko srečamo z več kot nekaj kriteriji.

Najprej teoretično analiziramo časovno ter prostorsko zahtevnost, nato pa hitrost algoritma tudi testiramo za probleme dimenzij od 3 do 6, na množicah točk različnih velikost in oblik. Na koncu analiziramo še scenarij, kjer je algoritmu za množico paroma nedominiranih točk $\mathcal{P}$ danih več točk poizvedbe $\mathcal{Q} = \{\mathbf{q}^1, \ldots, \mathbf{q}^m\}$ naenkrat.

## 5.1 Teoretična analiza časovne zahtevnosti

**Izrek 5.1.** *Algoritem ARRNO za računanje razdalje med nedominiranim območjem $N(\mathcal{P})$ in točko poizvedbe $\boldsymbol{q}$, ima časovno zahtevnost $O(n \log n)$ za $D = 3$ in $O(n^{D-1})$ za $D \geq 4$, kjer je $n = |\mathcal{P}|$ in $D$ dimenzija prostora.*

Časovna zahtevnost algoritma ARRNO je sestavljena iz časovne zahtevnosti računanja množice vpetih točk $\mathcal{V}$ ter časovne zahtevnosti iskanja razdalje do najbližjega stožca, ki ga razpenjajo vpete točke. Naj bo število vpetih točk enako $v$. Potem očitno poiščemo najbližji stožec v $O(v)$. Ker pa smo pri konstrukciji vpetih točk gotovo konstruirali vsako izmed njih, je časovna zahtevnost iskanja vpetih točk vsaj $O(v)$. Torej zadošča izračunati časovno zahtevnost algoritma Vpete točke($\mathcal{P}$). Vseeno pa najprej pokažimo, kolikšno je maksimalno število vpetih točk $\mathcal{V}$, glede na dimenzijo problema.

**Trditev 5.2.** *Za dimenzijo $D = 2, 3$ je število vpetih točk $O(n)$, kjer je $n = |\mathcal{P}|$. Za $D = 4$ je število vpetih točk $O(n^2)$ in obstajajo take množice točk $\mathcal{P}$, za katere je število vpetih točk $\Omega(n^2)$. Za $D > 4$ je število vpetih točk $O(n^{D-2})$.*

*Dokaz.* V algoritmu Vpete točke 3D($\mathcal{P}$), za računanje vpetih točk v treh dimenzijah, na vsakem koraku zanke (algoritem 2, vrstice 6–16) v stanje $S_\text{v}$ dodamo največ dve točki. Skupaj s točko $(0, 0)$, ki je v množici $S_\text{v}$ od začetka, torej v $S_\text{v}$ dodamo največ $2n + 1$ točk. Ker vpete točke računamo tako, da jih postopoma odstranjujemo iz $S_\text{v}$, bo tudi vpetih točk največ $2n + 1$.

Kadar rešujemo problem v štirih dimenzijah (algoritem 3, vrstica 14), $n$-krat dodamo vpete točke tridimenzionalnega podproblema. Torej je vpetih točk v štirih dimenzijah največ

$$1 + \sum_{i=1}^{n} 2i + 1 = (n + 1) + 2\sum_{i=1}^{n} i = (n + 1) + n(n + 1) = (n + 1)^2 = O(n^2).$$

Da je kvadratično število vpetih točk v štirih dimenzijah res tesna meja, pokažemo na primeru. Sestavimo tak primer, kjer pri dodajanju prve polovice točk vsakič

[2]Seveda pa dimenzije $D$ ne zanemarimo, ko ta nastopa v eksponentu.

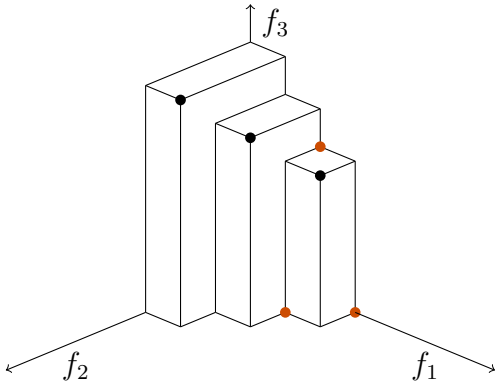
dobimo konstantno število vpetih točk, pri dodajanju druge polovice točk, pa vsakič linearno število novih vpetih točk. Tak primer je na primer množica točk

$$\left\{(i+1, \frac{n}{2} - i, n-i, n-i) \mid i = 0, 1, \ldots \frac{n}{2} - 1\right\},$$

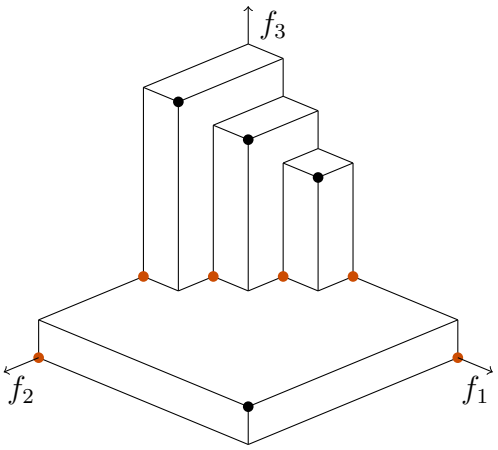
za prvo polovico točk, ter

$$\left\{(n-i, n-i, i+1, \frac{n}{2} - i) \mid i = 0, 1, \ldots \frac{n}{2} - 1\right\},$$

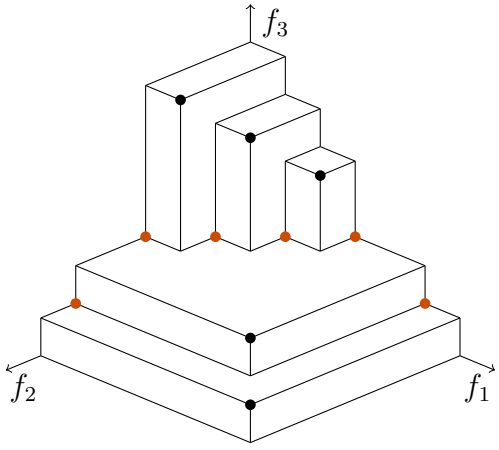
za drugo polovico točk. Na sliki 18 vidimo vizualizacijo tridimenzionalnega stanja $S_\mathrm{p}$ za primer $n = 6$ po dodajanju prvih treh, štirih, petih in šestih točk.



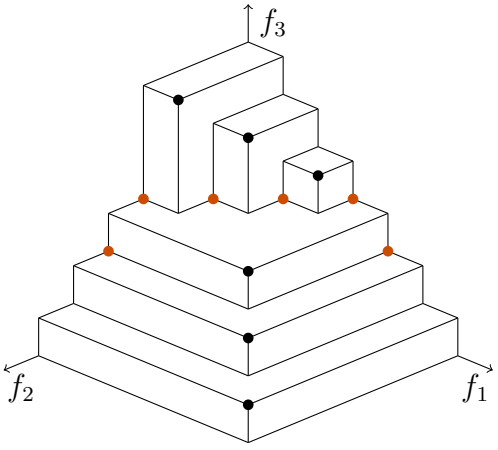

(a) Stanje po dodajanju tretje točke.




(b) Stanje po dodajanju četrte točke.




(c) Stanje po dodajanju pete točke.




(d) Stanje po dodajanju šeste točke.

Slika 18: Vizualizacija stanja $S_\mathrm{p}$ po dodajanju točk iz $\mathcal{P}$. S črno barvo so označene točke v $S_\mathrm{p}$, z oranžno barvo pa novonastale vpete točke pri dodajanju zadnje točke v $S_\mathrm{p}$. Pri dodajanju prvih $\frac{n}{2}$ točk vsakič nastanejo tri nove vpete točke, kot vidimo na sliki 18a. Pri dodajanju druge polovice točk, pa vsakič nastane $(\frac{n}{2} + 1) + 2$ novih vpetih točk, kot vidimo na slikah 18b, 18c in 18d.

Podobno kot za število vpetih točk v štirih dimenzijah, lahko tudi za dimenzije $D > 4$ pokažemo, da je vpetih točk največ $O(n^{D-2})$. Ne vemo pa, ali je ta zgornja

meja tesna. Glede na to, da je število vpetih točk v dveh in treh dimenzijah linearno, bi bilo prav tako mogoče, da je vpetih točk $O(n^{\lfloor \frac{D}{2} \rfloor})$. Taka meja se pojavi pri problemu Kleejeve mere [3], torej računanja volumna unije kvadrov vzporednih z koordinatnimi osmi. □

### 5.1.1 Časovna zahtevnost v treh dimenzijah

Najprej si oglejmo algoritem VPETE TOČKE 3D($\mathcal{P}$) (algoritem 2). Za inicializacijo struktur $\mathcal{V}$, $S_\mathrm{p}$, $S_\mathrm{v}$ in $h$ porabimo konstanten čas. Množico $\mathcal{V}$ implementiramo kot seznam, stanji $S_\mathrm{p}$ in $S_\mathrm{v}$ kot urejen seznam, ki podpira dodajanje in iskanje v logaritemskem času [12], $h$ pa kot zgoščevalno funkcijo.

Nato v zanki (vrstice 6–16) $n$-krat izvedemo več ukazov. Funkcija ODSTRANI DOMINIRANE($S_\mathrm{v}, \overline{\mathbf{p}}$) z bisekcijo poišče prvi ter zadnji dominiran element, nato pa ju odstrani in vrne, skupaj z vsemi elementi seznama vmes. Torej je zahtevnost funkcije enaka $O(\log |S_\mathrm{v}| + k)$, kjer je $k$ število dominiranih točk v $S_\mathrm{v}$. Ker je vsaka točka iz množice $S_\mathrm{v}$ odstranjena le enkrat in bo vseh vpetih točk največ $O(n)$, bo po $n$ iteracijah skupna časovna zahtevnost funkcije enaka $O(n \log n)$. Podobno se tudi notranjost naslednje zanke (vrstice 9–10), skozi vse iteracije zunanje zanke, izvede le $O(n)$-krat. Znotraj zanke so vse operacije konstantne, torej je skupna časovna zahtevnost vrstic 8–10, skozi celotno delovanje algoritma, enaka $O(n)$. V vrstici 11 stanju $S_\mathrm{p}$ dodamo novo točko, kar naredimo v $O(\log |S_\mathrm{p}|) = O(\log n)$, saj je stanje $S_\mathrm{p}$ vedno manjše od števila točk $n$. Prav tako v logaritemskem času izračunamo tudi novi vpeti točki, za izračun potrebujemo le indeks elementa $\overline{\mathbf{p}}$ v množici $S_\mathrm{p}$, nato pa točki izračunamo s pomočjo sosednjih elementov v seznamu. Ker je iskanje in nastavljanje elementa v zgoščevalni funkciji v povprečju konstantno, se tudi vrstice 13–15 izvedejo v konstantnem času, dodajanje v stanje $S_\mathrm{v}$ pa zopet porabi $O(\log n)$. Torej za celotno zanko for (vrstice 6–16) algoritem porabi $O(n \log n)$ časa. Na koncu le še poskrbimo za preostale točke v $S_\mathrm{v}$, za katere pa smo že pokazali, da jih je največ $O(n)$.

Torej je skupna časovna zahtevnost algoritma za iskanje vpetih točk v treh dimenzijah (ter potem tudi algoritma ARRNO v treh dimenzijah) $O(n \log n)$.

### 5.1.2 Časovna zahtevnost v višjih dimenzijah

Naj bo $T(n, D)$ čas, ki ga porabi algoritem za vhodno množico $|\mathcal{P}| = n$ in dimenzijo $D$. Pokazali smo že, da je $T(n, 3) = O(n \log n)$, zdaj pa bi radi izračunali $T(n, D)$ še za poljubno dimenzijo $D > 3$.

Tudi v višjih dimenzijah za inicializacijo stanj potrebujemo konstanten čas, časovno najbolj zahtevne operacije, pa se zgodijo znotraj glavne zanke algoritma 3. Funkcijo ODSTRANI DOMINIRANE($S_\mathrm{v}, \overline{\mathbf{p}}$), kličemo $n$ krat in potrebuje linearno časovno zahtevnost v odvisnosti od števila točk v $S_\mathrm{v}$. Ker je število točk v $S_\mathrm{v}$ omejeno z $O(n^{D-2})$, je časovna zahtevnost funkcije v najslabšem primeru $O(n^{D-1})$. Po enakem razmisleku kot v algoritmu treh dimenzij, algoritem za vrstice 10–12 porabi $O(n^{D-2})$ časa. Za dodajanje točke $\overline{\mathbf{p}}$ v stanje $S_\mathrm{p}$, skupaj z odstranjevanjem dominiranih točk pa je na vsakem koraku potrebno $O(n)$ časa, torej skupaj $O(n^2)$. Računanje novih vpetih točk in dodajanje točk v zgoščevalno funkcijo $h$ v vrsticah 14–17, pa ima znotraj vsake ponovitve časovno zahtevnost $T(n, D-1)$. Za dodajanje preostalih

točk iz $S_\text{v}$ v $\mathcal{V}(\mathcal{P})$ pa je potrebno največ $O(n^{D-2})$ časa, kolikor je možnih točk v $S_\text{v}$. Torej velja

$$T(n, D) = O(n^{D-1}) + nT(n, D-1).$$

Za $D = 4$ torej velja

$$T(n, 4) = O(n^3) + O(n^2 \log n) = O(n^3),$$

za splošen $D \geq 4$ pa

$$T(n, D) = O(n^{D-1}).$$

## 5.2 Analiza prostorske zahtevnosti

Za analizo prostorske zahtevnosti algoritma, spremljamo strukture $\mathcal{P}$ $\mathcal{V}$, $\overline{\mathcal{V}}$, $S_\text{p}$, $S_\text{v}$ in $h$ tekom algoritma. Vse ostale spremenljivke zasedajo konstantno veliko prostora. Vhodna množica $\mathcal{P}$ je velikosti $O(n)$, kakor tudi stanje $S_\text{p}$, ki se inicializira prazno, na koncu pa ima največ $O(n)$ točk. Za izhodno množico vpetih točk smo v trditvi 5.2 pokazali da vsebuje največ $O(n^{D-2})$ točk, množica $\overline{\mathcal{V}}$ pa $O(n^{D-3})$ točk, saj predstavlja vpete točke problema v nižji dimenziji. Enako kot $\mathcal{V}$ potrebujeta tudi stanje $S_\text{v}$ in zgoščevalna funkcija $h$ največ $O(n^{D-2})$ prostora.

Upoštevati je potrebno tudi prostor, ki ga zasede tudi klicanje rekurzivne funkcije. Znotraj funkcije sicer $n$ krat rekurzivno kličemo funkcijo z nižjo dimenzijo, vendar se hkrati izvaja vedno največ en klic funkcije za vsako dimenzijo $D \geq 3$. Torej je skupna prostorska zahtevnost enaka

$$\sum_{i=3}^{D} O(n^{i-2}) = O(n^{D-2}).$$

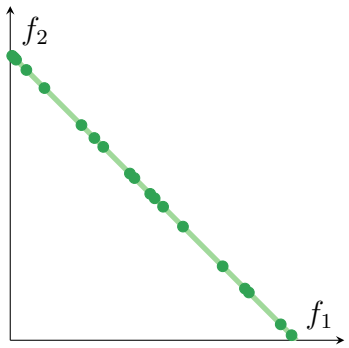
## 5.3 Eksperimentalna analiza časovne zahtevnosti

Za ovrednotenje hitrosti algoritma ARRNO izvedemo serijo poskusov pri dimenzijah problema 3, 4, 5 in 6. Testiranje algoritma pri višjih dimenzijah problema je zaradi eksponentne časovne zahtevnosti dolgotrajno, prav tako pa za to tudi ni prave motivacije, saj se z večkriterijsko optimizacijo s tolikšnim številom kriterijev v praksi srečamo redko.

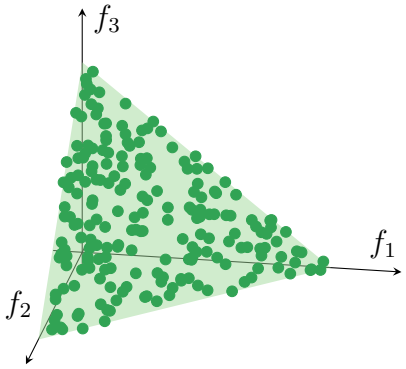
Testne instance ustvarimo tako, da čim bolj posnemajo scenarij večkriterijske optimizacije, kjer skozi proces optimizacije dobivamo množico paroma nedominiranih točk, ki ji rečemo fronta. Dva primera takih front, ki se pogosto pojavita v raziskavah in se enostavno generalizirata na poljubno število dimenzij sta sferična ter linearna fronta [14]. Prav tako pa ustvarimo tudi poseben primer fronte, za katero pričakujemo, da je za algoritem najbolj časovno zahtevna.

### 5.3.1 Linearna fronta

Linearna fronta je sestavljena iz točk, ki ležijo na hiperravnini v pozitivnem delu koordinatnega sistema in predstavlja primer večkriterijskega problema z linearnim kompromisom med kriteriji. Primera front v dveh in treh dimenzijah vidimo na sliki 19.




(a) Primer linearne fronte z 20 točkami v dveh dimenzijah.

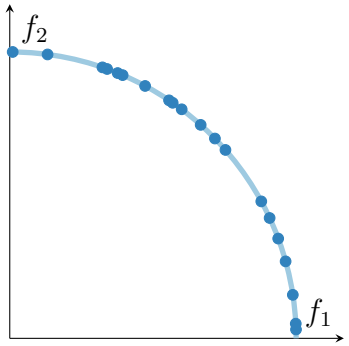



(b) Primer linearne fronte z 200 točkami v treh dimenzijah.

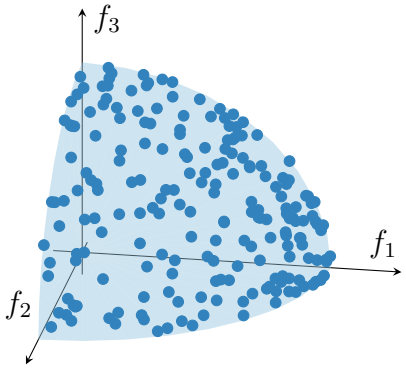
Slika 19: Primera linearne fronte v dveh dimenzijah na levi in treh dimenzijah na desni.

### 5.3.2 Sferična fronta

Sferična fronta je sestavljena iz točk, ki ležijo na sferi s središčem v izhodišču. Primera front v dveh in treh dimenzijah vidimo na sliki 20.




(a) Primer sferične fronte z 20 točkami v dveh dimenzijah.

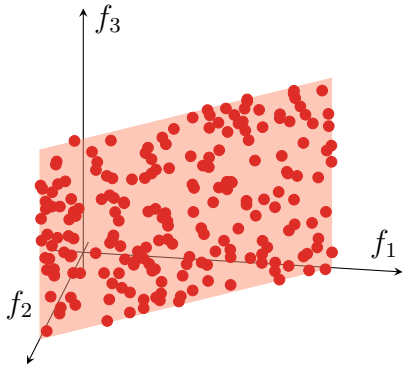



(b) Primer sferične fronte z 200 točkami v treh dimenzijah.

Slika 20: Primera sferične fronte v dveh dimenzijah na levi in treh dimenzijah na desni.

### 5.3.3 Algoritmično zahtevna fronta

Za testiranje ustvarimo tudi fronto, za katero velja, da je projekcija točk na prvih $D-1$ koordinat paroma nedominiranih. Intuicija, zakaj naj bi bila taka fronta najtežja za algoritem, je, da bosta imeli stanji $S_\mathrm{p}$ in $S_\mathrm{v}$ tekom delovanja algoritma vedno več točk. Sestavimo jo tako, da točkam iz linearne fronte z eno manjšo dimenzijo dodamo naključno zadnjo koordinato. Taka definicija je smiselna le za $D \geq 3$, saj množica enodimenzionalnih točk ne more biti paroma nedominiranih. Primer fronte v treh dimenzijah vidimo na sliki 21.




Slika 21: Primer algoritmično zahtevne fronte z 200 točkami v treh dimenzijah.

### 5.3.4 Okolje poskusov

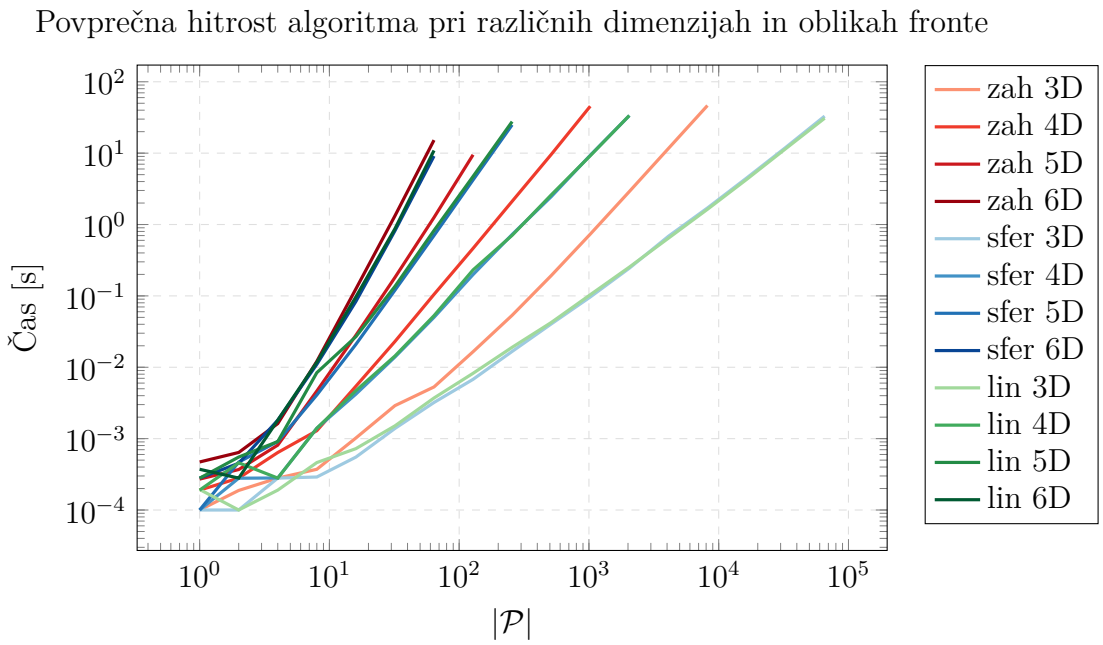
Algoritem implementiramo v programskem jeziku Python 3.10. Poskuse izvedemo na računalniku z operacijskim sistemom Windows z 2,40 GHz procesorjem Intel Core i7-12800H s 14 jedri in 32 GB pomnilnika RAM.

### 5.3.5 Rezultati poskusov

Algoritem testiramo na linearni in sferični fronti v dimenzijah 3, 4, 5 in 6. Za naključno ustvarjeno fronto $\mathcal{P}$ in naključno izbrano točko $\mathbf{q} \succeq 0$, $\mathbf{q} \notin N(\mathcal{P})$, merimo čas, ki ga algoritem porabi za izračun razdalje do nedominiranega območja. Začnemo z množico $\mathcal{P}$, ki vsebuje le eno točko, nato pa velikost množice podvajamo, dokler algoritem za računanje ne porabi več kot 60 sekund. Pri vsaki velikosti množice $\mathcal{P}$ eksperiment ponovimo desetkrat, vsakič z novo fronto in novo točko, da dobimo čim bolj robustno oceno časovne zahtevnosti.

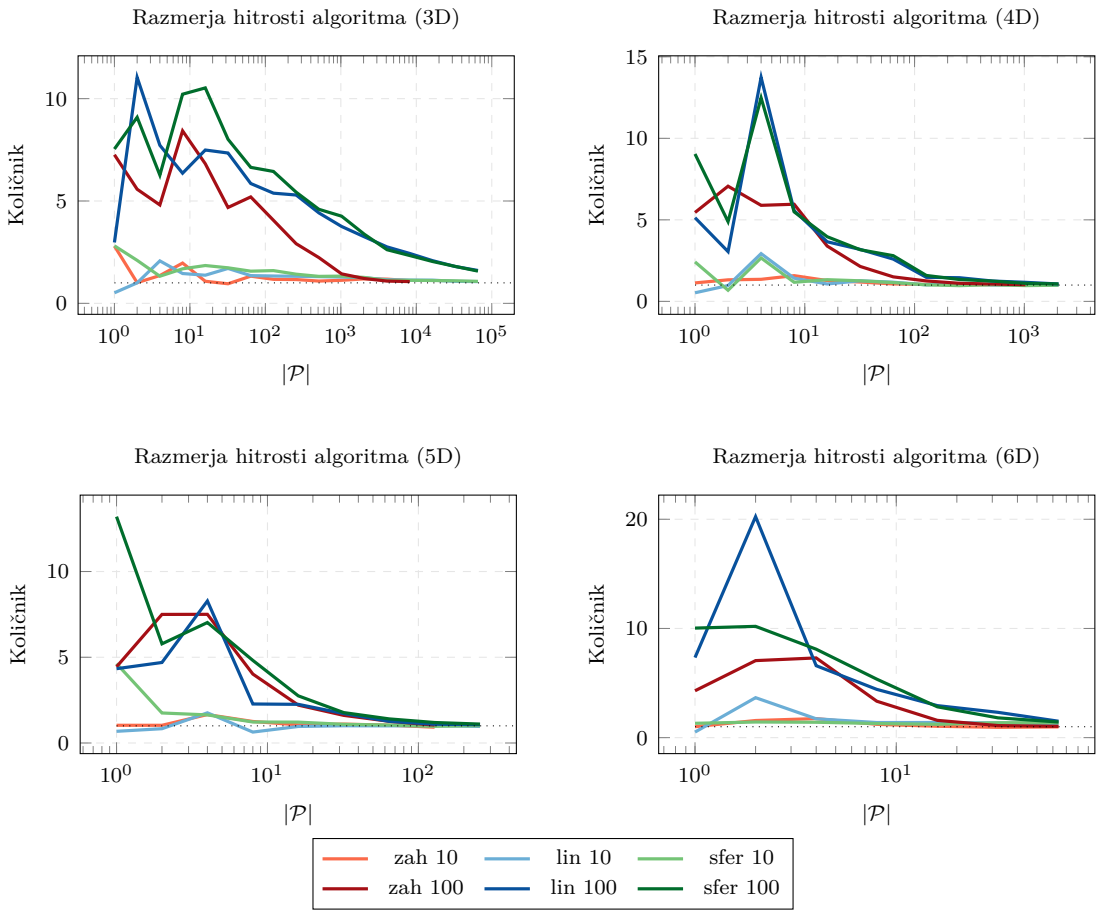
Rezultate prikažemo na sliki 22. Opazimo, da algoritem, posebno na problemih v treh in štirih dimenzijah, deluje počasneje za zahtevno fronto kot za linearno ter sferično fronto. Prav tako je očitno, da se z višjo dimenzijo zahtevnost problema veča. Vidimo, da algoritem na problemu v treh dimenzijah deluje zelo hitro, predvsem za linearno in sferično fronto. Tudi algoritem za reševanje problema v štirih dimenzijah deluje razmeroma hitro in je v praksi uporaben za množice do okrog 1000 točk. Za računanje razdalje v petih ali šestih dimenzijah pa algoritem že pri množici $\mathcal{P}$ s sto točkami potrebuje nekaj sekund. Za množice z več točkami ali višje dimenzije pa bi algoritem verjetno potreboval že več minut časa.

Do zdaj smo obravnavali primer računanja razdalje ene točke $\mathbf{q}$ do nedominiranega območja $N(\mathcal{P})$. Vendar pa je motivacija za algoritem ARRNO v uporabi za urejanje dominiranih rešitev pri algoritmu COMO-CMA-ES, kjer računamo razdaljo za več dominiranih točk $\mathbf{q}_1, \ldots, \mathbf{q}_m$ do nedominiranega območja $N(\mathcal{P})$ naenkrat. V tem primeru lahko vpete točke izračunamo le enkrat, za vsako točko $\mathbf{q}$ pa izračunamo le razdaljo do vseh stožcev definiranih z vpetimi točkami. Zato poskuse ponovimo še enkrat, le da tokrat računamo razdaljo za množice poizvedbenih točk velikosti 10 in 100. Porabljen čas algoritma primerjamo s časom porabljenim za eno točko in rezultate prikažemo na sliki 23.




Slika 22: Graf prikazuje hitrost algoritma v odvisnosti od velikosti množice $\mathcal{P}$. Z odtenki rdeče barve so prikazani povprečni časi testiranja na zahtevni fronti, z odtenki modre povprečni časi na sferični fronti, z odtenki zelene pa časi na linearni fronti. Rezultati so prikazani za fronte od dimenzije 3 z najsvetlejšim odtenkom, do fronte dimenzije 6 z najtemnejšim odtenkom.

Na grafih vidimo, da računanje razdalje do desetih oziroma stotih točk, ni desetkrat oziroma stokrat počasnejše kot računanje razdalje do ene točke. Opazimo tudi, da z naraščajočim številom točk v množici $\mathcal{P}$, količnik pada proti ena, torej vedno večji delež zahtevnosti algoritma postaja iskanje vpetih točk in ne računanje razdalje do njih.




Slika 23: Grafi prikazujejo kolikokrat počasnejši je čas algoritma, kadar računamo razdaljo za deset poizvedbenih točk (s svetlejšo barvo) oziroma sto poizvedbenih točk (s temnejšo barvo) v primerjavi s časom algoritma za eno dano poizvedbeno točko. Z odtenki rdeče barve so prikazani rezultati na zahtevni fronti, z modrimi odtenki rezultati na linearni fronti, z zelenimi pa na sferični fronti.

# 6 Zaključek

V magistrskem delu smo obravnavali problem računanja razdalje do nedominiranega območja, ki je pomemben del algoritma COMO-CMA-ES za reševanje večkriterijskih optimizacijskih problemov. Do sedaj je obstajal zgolj algoritem za računanje razdalje do nedominiranega območja v dveh dimenzijah, v tem delu pa predstavimo nov algoritem, poimenovan ARRNO, ki deluje za poljubno višjo dimenzijo. S tem smo odprli pot za nadaljnji razvoj algoritmov, kot je COMO-CMA-ES, v višjih dimenzijah.

Algoritem ARRNO temelji na obstoječem dvodimenzionalnem algoritmu, kjer avtorji ugotovijo, da je ključno iskanje vpetih točk. Prav tako se poslužuje tehnike rekurzivnega pometanja, kot jih uporabljata algoritma HV3D+ in HV4D+. Algoritem smo implementirali in vključili v odprtokodno knjižnico `moarchiving` [11].

Pravilnost algoritma smo potrdili s obsežnim testiranjem, ki smo ga podrobno izpeljali. Poleg tega smo teoretično analizirali prostorsko in časovno zahtevnost algoritma, ki znaša $O(n \log n)$ za probleme v treh dimenzijah in $O(n^{D-1})$ za probleme v več dimenzijah, kjer je $n$ velikost vhodne množice $D$ pa dimenzija problema. Teoretično analizo časovne zahtevnosti smo dopolnili tudi s serijo eksperimentov za različne dimenzije, velikosti in oblike množic točk. Rezultati kažejo, da je algoritem ARRNO za tri in štiridimenzionalne probleme učinkovit in primeren za praktično uporabo, sploh za manjše množice točk.

V prihodnje bi bilo smiselno raziskati možnosti nadaljnje optimizacije algoritma ARRNO, predvsem njegove časovne zahtevnosti za probleme v višjih dimenzijah. Morda obstaja hitrejši algoritem za iskanje vpetih točk ali pa način za omejitev števila vpetih točk, ki jih je potrebno izračunati. Prav tako, bi bilo zanimivo raziskati pristop, ki ne bi temeljil na računanju vpetih točk, ampak bi razdaljo poiskali na kak drug način.

# Literatura


[1] N. Beume, B. Naujoks in M. Emmerich, *SMS-EMOA: Multiobjective selection based on dominated hypervolume*, European Journal of Operational Research **181**(3) (2007) 1653–1669, DOI: 10.1016/j.ejor.2006.08.008.

[2] N. Beume in dr. *On the complexity of computing the hypervolume indicator*, IEEE Transactions on Evolutionary Computation **13**(5) (2009) 1075–1082, DOI: 10.1109/TEVC.2009.2015575.

[3] K. Bringmann, *Bringing order to special cases of Klee's measure problem*, v: Mathematical Foundations of Computer Science 2013 (ur. K. Chatterjee in J. Sgall), Springer, 2013, str. 207–218, DOI: 10.1007/978-3-642-40313-2_20.

[4] M. de Berg in dr. *Computational geometry: Algorithms and applications*, 3rd ed., 2008, DOI: 10.1007/978-3-540-77974-2.

[5] K. Deb, *Multi-objective evolutionary algorithms*, v: Springer Handbook of Computational Intelligence (ur. J. Kacprzyk in W. Pedrycz), Springer, 2015, 995–1015, DOI: 10.1007/978-3-662-43505-2_49.

[6] K. Deb in H. Jain, *An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, Part I: Solving problems with box constraints*, IEEE Transactions on Evolutionary Computation **18**(4) (2014) 577–601, DOI: 10.1109/TEVC.2013.2281535.

[7] K. Deb in dr. *A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II*, v: Parallel Problem Solving from Nature — PPSN VI, Springer, 2000, str. 849–858, DOI: 10.1007/3-540-45356-3_83.

[8] A. P. Guerreiro in C. M. Fonseca, *Computing and updating hypervolume contributions in up to four dimensions*, IEEE Transactions on Evolutionary Computation **22**(3) (2018) 449–463, DOI: 10.1109/TEVC.2017.2729550.

[9] N. Hansen in A. Ostermeier, *Completely derandomized self-adaptation in evolution strategies*, Evolutionary Computation **9**(2) (2001) 159–195, DOI: 10.1162/106365601750190398.

[10] H. T. Kung, F. Luccio in F. P. Preparata, *On finding the maxima of a set of vectors*, J. ACM **22**(4) (1975) 469–476, DOI: 10.1145/321906.321910.

[11] *Moarchiving*, [ogled 23. 1. 2025], dostopno na https://github.com/CMA-ES/moarchiving.

[12] *Sorted Containers*, [ogled 23. 5. 2025], dostopno na https://grantjenks.com/docs/sortedcontainers/.

[13] C. Touré in dr. *Uncrowded hypervolume improvement: COMO-CMA-ES and the sofomore framework*, v: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '19, Association for Computing Machinery, New York, NY, USA, 2019, str. 638–646, DOI: 10.1145/3321707.3321852.

[14] T. Tušar in B. Filipič, *Visualization of Pareto front approximations in evolutionary multiobjective optimization: A critical review and the prosection method*, IEEE Transactions on Evolutionary Computation **19**(2) (2015) 225–245, DOI: 10.1109/TEVC.2014.2313407.

[15] E. Zitzler in dr. *Performance assessment of multiobjective optimizers: an analysis and review*, IEEE Transactions on Evolutionary Computation **7**(2) (2003) 117–132, DOI: 10.1109/TEVC.2003.810758.