Zavod za elektroniku, mikroelektroniku, računalne i inteligentne sustave
Fakultet elektrotehnike i računarstva
Sveučilište u Zagrebu

Višekriterijska optimizacija funkcije

dr. sc. Marko Čupić

Materijali za kolegij:

Rješavanje optimizacijskih problema algoritmima evolucijskog računanja u Javi

Zagreb, prosinac, 2013.

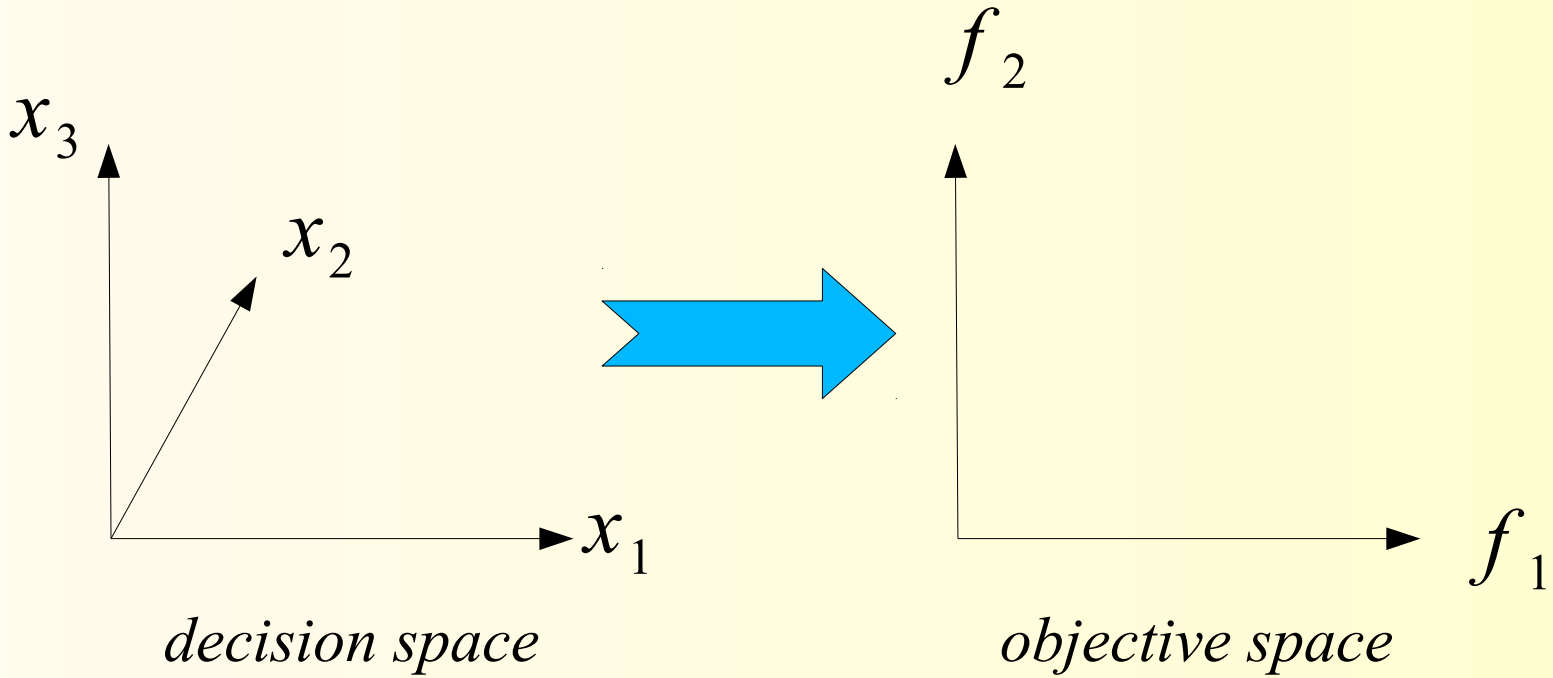
MOOP – formalni opis

$$\begin{aligned}
&\text{minimiziraj/maksimiziraj } f_m(\vec{x}), && m = 1,2,\ldots,M \\
&\text{uz ograničenja } g_j(\vec{x}) \geqslant 0, && j = 1,2,\ldots,J \\
&h_k(\vec{x}) = 0, && k = 1,2,\ldots,K \\
&\vec{x}_i^{(L)} \leqslant \vec{x}_i \leqslant \vec{x}_i^{(U)} && i = 1,2,\ldots,n
\end{aligned}$$

Feasible solution – rješenje koje zadovoljava sva ograničenja
Infeasible solution – ne zadovoljava barem jedno ograničenje

MOOP – formalni opis

- Radimo s dva prostora
  - *Decision space – prostor rješenja*
  - *Objective space – prostor ciljnih funkcija*



MOOP – formalni opis

- Rješenja su vektori u D
- Dobrota su vektori u Z
- Problem: usporedba vektora!
- Radimo *minimizaciju* prema dva kriterija i imamo rješenja dobrota:
  $f(R_1)=(17,20)$, $f(R_2)=(15,21)$ te $f(R_3)=(13,15)$
  - Je li $f(R_1)<f(R_2)$, $f(R_1)<f(R_3)$, $f(R_2)<f(R_3)$ u smislu "<" je bolje?

Unaprijed zadani *tradeoff*

- Ako znamo u kojoj su nam mjeri pojedini kriteriji važni, možemo problem prevesti u klasičnu optimizaciju konstrukcijom težinske sume

$$f = \sum_{i=1}^{M} \omega_i \cdot f_i$$

- Pretpostavka MOOP jest da su svi kriteriji podjednako važni
  - Odluku će kasnije napraviti čovjek

Pojam dominacije

- Uvodi se relacija *dominacije*
- Definicija
  Rješenje $x^{(1)}$ dominira nad $x^{(2)}$ ako vrijede **oba** sljedeća uvjeta
  - Rješenje $x^{(1)}$ nije gore od rješenja $x^{(2)}$ niti u jednoj kriterijskoj funkciji
  - Rješenje $x^{(1)}$ je *strogo bolje* od rješenja $x^{(2)}$ barem u jednoj kriterijskoj funkciji
- Nije refleksivna, nije simetrična, jest tranzitivna

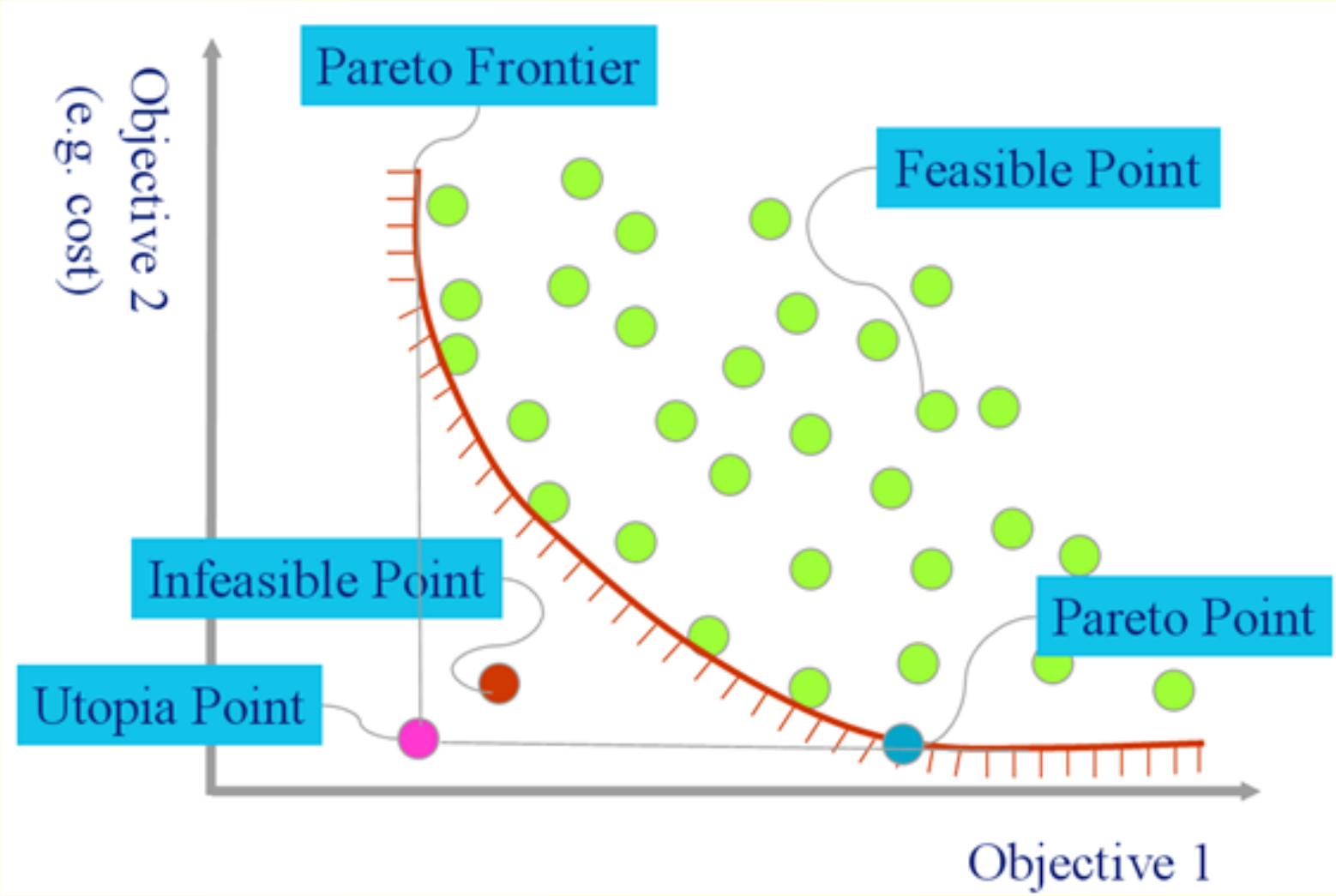
Dominacija

- Pretpostavka: minimizacija po oba kriterija; $f(R_1)=(17,20)$, $f(R_2)=(15,21)$ te $f(R_3)=(13,15)$
- R_1 *dom* R_2 ==> NE
  R_1 *dom* R_3 ==> NE
  R_2 *dom* R_1 ==> NE
  R_2 *dom* R_3 ==> NE
  R_3 *dom* R_1 ==> DA
  R_3 *dom* R_2 ==> DA

Cilj MOOP-a

- U skupu rješenja P, ***nedominirani skup*** P' čine ona rješenja iz P nad kojima niti jedno rješenje iz P ne dominira
- P' je podskup od P
- Nedominirani skup čitavog *feasible* prostora rješenja zove se ***Pareto optimalni skup***.
  (dakle, to je skup vektora iz *decision space*-a)
- Za dani Pareto optimalni skup ***Pareto fronta*** je skup pripadnih vektora iz *objective space*-a.

Primjer Pareto fronte



Nedominirano sortiranje

- MOOP algoritmi koje ćemo raditi trebaju sortirati rješenja
- Algoritmi sortiranja zahtijevaju da je definirana relacija R koja je relacija potpunog uređaja:
  - za svaka dva elementa vrijedi (x,y) je u R ili je (y,x) u R
- Relacija dominacije je relacija parcijalnog uređaja
  - postoje *neusporedivi* elementi:
    niti je (x,y) u R, niti je (y,x) u R

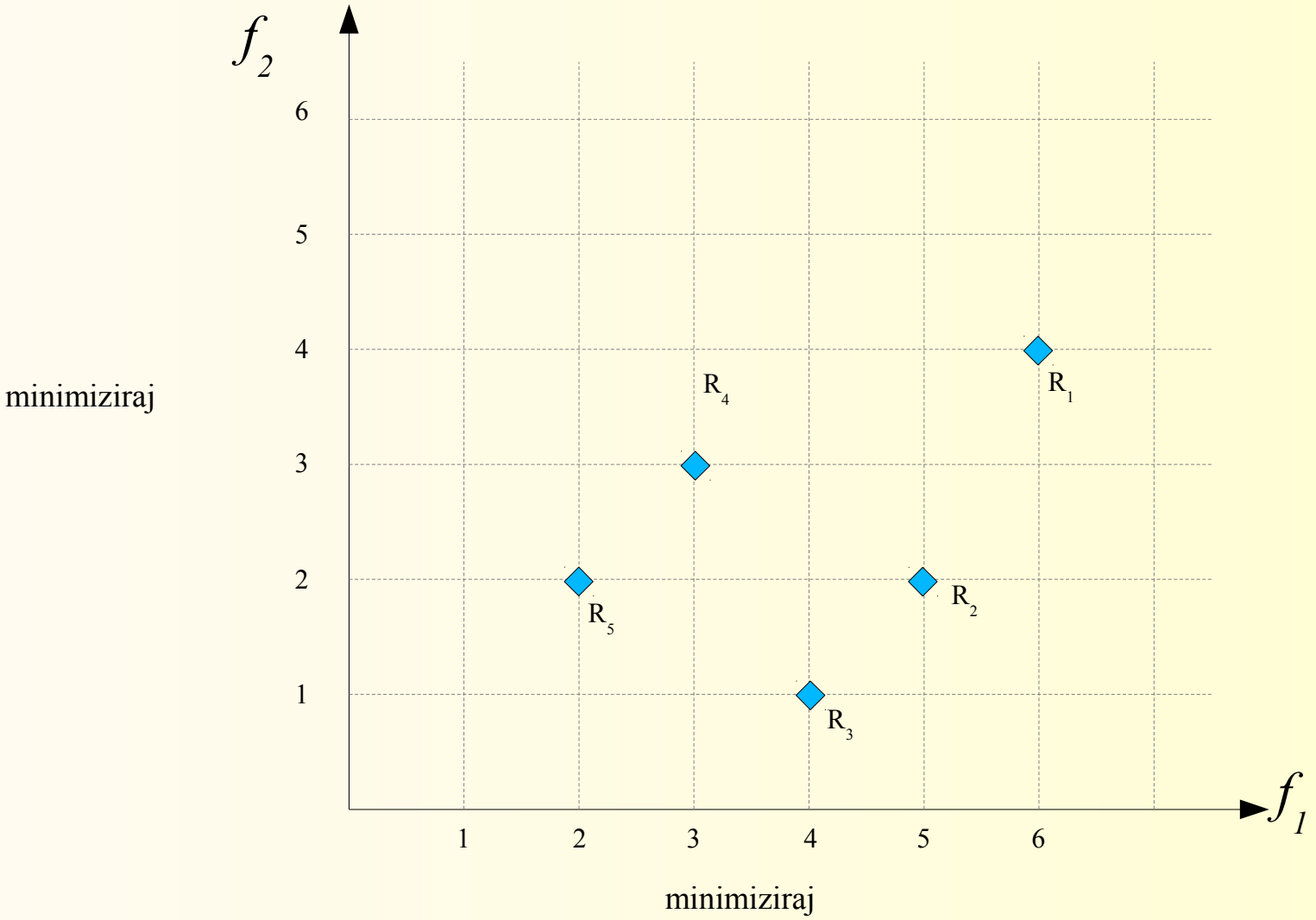
Nedominirano sortiranje

- **Problem**: ne mogu usporediti proizvoljna dva rješenja (u smislu relacije *dominacije*)
- **Rješenje problema**:
  - za svako rješenje definiramo skalarnu mjeru
  - rješenja uspoređujemo prema toj mjeri
  - mjera je *stupanj nedominiranosti*, tj. za neko konkretno rješenje, gledamo koliko ima drugih rješenja u skupu rješenja koja ga dominiraju (očito – manje je bolje)
  - ovaj stupanj ćemo još zvati *rang* rješenja, r_i

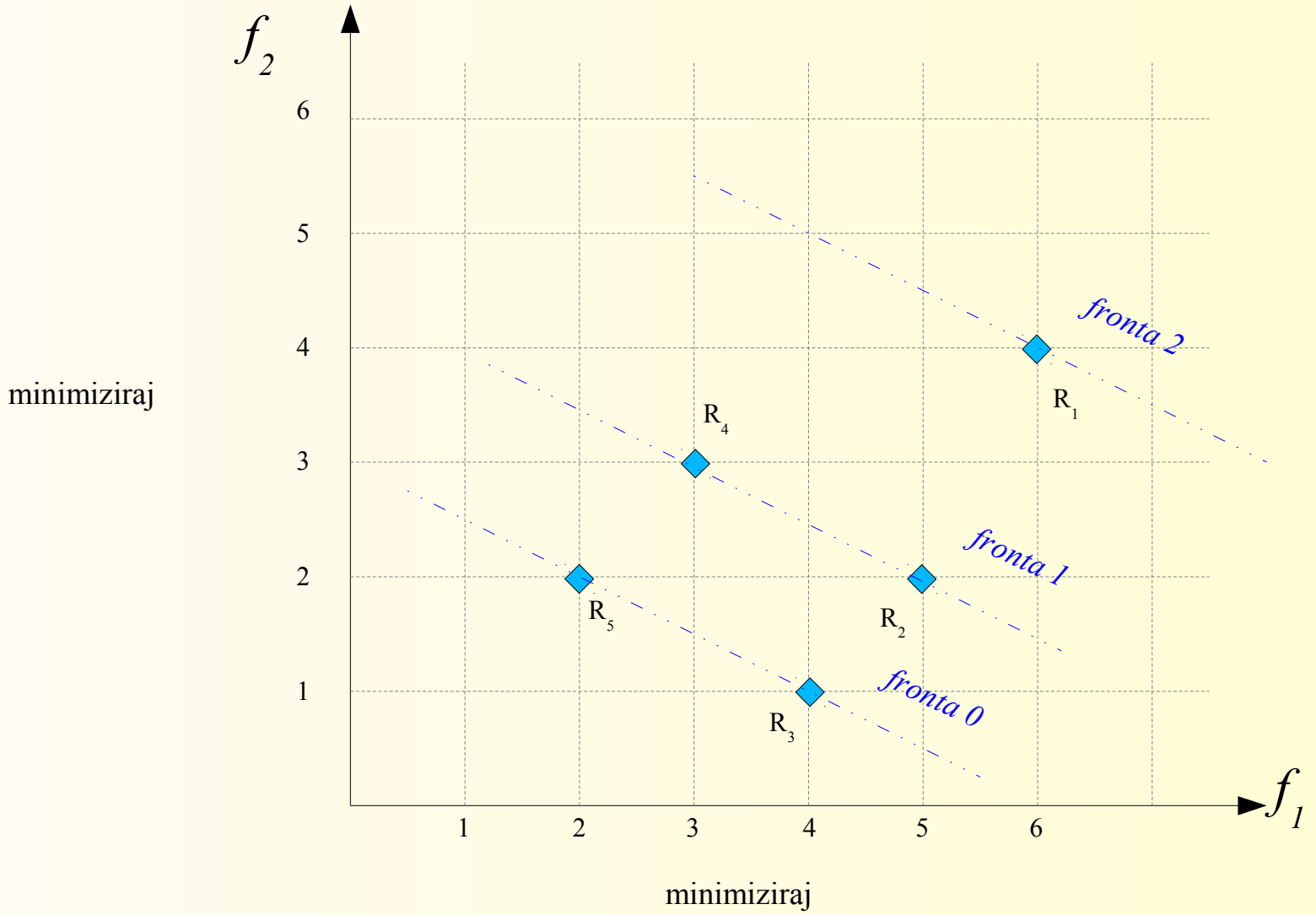
Nedominirano sortiranje

- Prema stupnju nedominiranosti rješenja ćemo podijeliti u ***fronte***
- Fronta *k* "konceptualno" sadrži svako rješenje *R* za koje u skupu koji sortiramo postoji točno *k* drugih rješenja koja ga dominiraju
  - Preciznije, izbacimo li iz skupa sva rješenja fronti 0 do *k*-1, rješenja fronte *k* sadrže nedominirana rješenja
- Fronta 0 tada predstavlja *aproksimaciju* Pareto optimalnog skupa (odnosno u prostoru ciljnih funkcija aproksimaciju Pareto fronte)

Nedominirano sortiranje - primjer



Nedominirano sortiranje - primjer



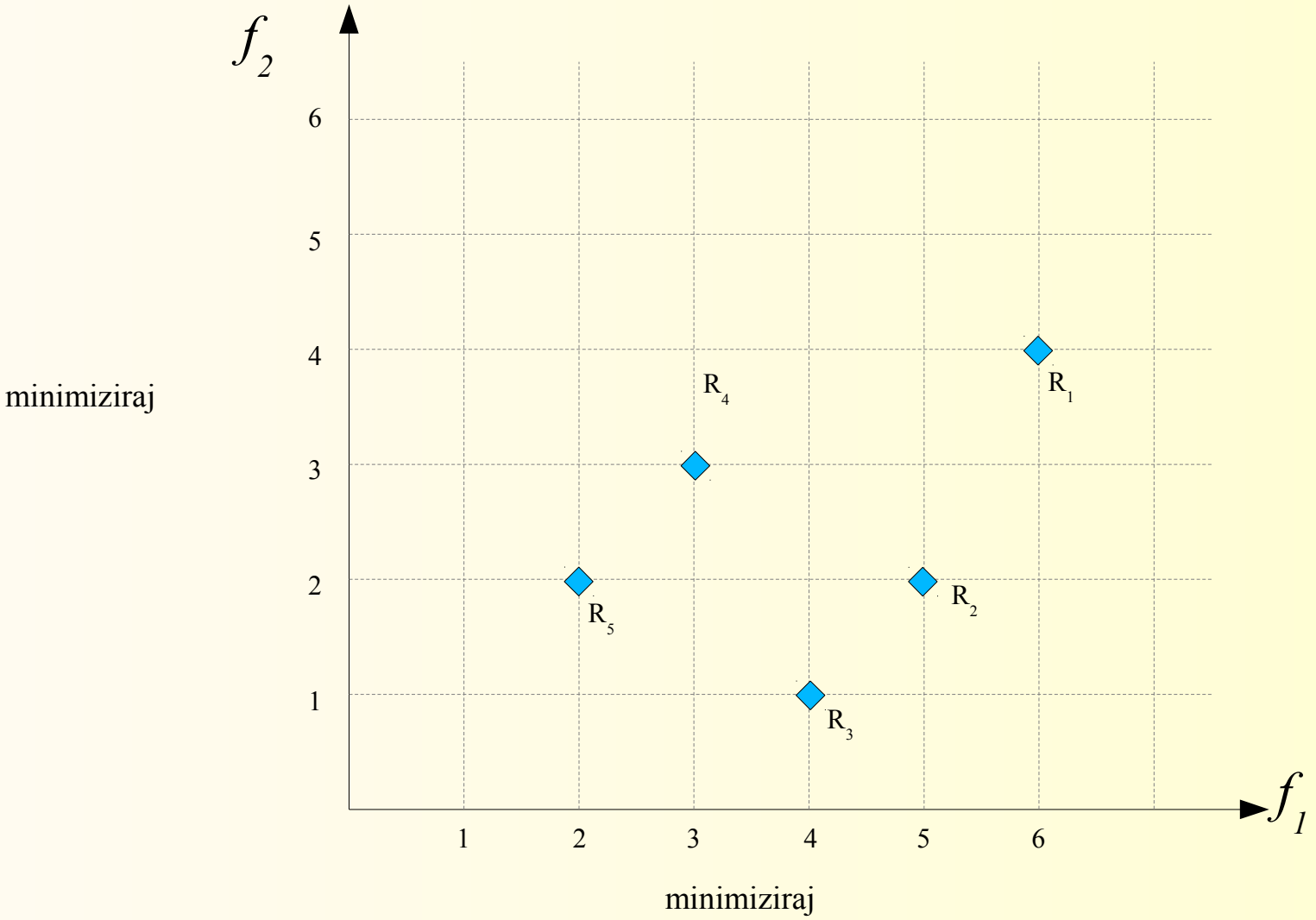
Nedominirano sortiranje - algoritam

- Za svako rješenje x_i iz predanog skupa P pronaći skup rješenja S_i nad kojima to rješenje dominira, te utvrditi stupanj nedominacije od x_i (oznaka n_i)
- Složenost ovog koraka je kvadratna s brojem rješenja (no množi se i s M) – $O(MN^2)$

Nedominirano sortiranje - algoritam

- Broj fronte $i = 0$; i-ta fronta f_i je početno prazna
- Iterativno ponavljaj dok ima rješenja
  - Pronađi sva rješenja r_k koja imaju n_i=0; makni ih i ubaci u f_i. Za svako to rješenje pogledaj njegov skup S_k i svakom rješenju iz S_k umanji stupanj nedominacije za 1.
- Vrati pronađeni slijed fronti.
- Primjer...

Nedominirano sortiranje - primjer



Nedominirano sortiranje - primjer

Stanje nakon provedenog koraka utvrđivanja S_i i n_i:

Rješenje	Dominira nad (S_i)	Je dominiran od (n_i)
1 … (6,4)		~~0~~ ~~1~~ ~~2~~ ~~3~~ 4
2 … (5,2)	1	~~0~~ ~~1~~ 2
3 … (4,1)	1, 2	0
4 … (3,3)	1	~~0~~ 1
5 … (2,2)	1, 2, 4	0

Nedominirano sortiranje - primjer

Stanje nakon utvrđivanja fronte 0:

Rješenje	Dominira nad (S_i)	Je dominiran od (n_i)
1 … (6,4)		~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~3~~ 2
2 … (5,2)	1	~~0~~ ~~1~~ ~~2~~ ~~1~~ 0
~~3 … (4,1)~~	~~1, 2~~	~~0~~
4 … (3,3)	1	~~0~~ ~~1~~ 0
~~5 … (2,2)~~	~~1, 2, 4~~	~~0~~

$f_0 = \{3, 5\}$

Nedominirano sortiranje - primjer

Stanje nakon utvrđivanja fronte 1:

Rješenje	Dominira nad (s_i)	Je dominiran od (n_i)
1 … (6,4)		~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~3~~ ~~2~~ ~~1~~ 0
~~2 … (5,2)~~	~~1~~	~~0~~ ~~1~~ ~~2~~ ~~1~~ 0
~~3 … (4,1)~~	~~1, 2~~	~~0~~
~~4 … (3,3)~~	~~1~~	~~0~~ ~~1~~ 0
~~5 … (2,2)~~	~~1, 2, 4~~	~~0~~

$f_0 = \{3, 5\}, \quad f_1 = \{2, 4\}$

Nedominirano sortiranje - primjer

Stanje nakon utvrđivanja fronte 2:

Rješenje	Dominira nad (s_i)	Je dominiran od (n_i)
1 … (6,4)		0
2 … (5,2)	1	0
3 … (4,1)	1, 2	0
4 … (3,3)	1	0
5 … (2,2)	1, 2, 4	0

$f_0=\{3, 5\}$, $f_1=\{2, 4\}$, $f_2=\{1\}$

NSGA

- Srinivas, Deb (1994): ***Non-Dominated Sorting Genetic Algorithm*** (stranice 209 do 218)
- Generacijski algoritam; ideja:
  - Napravi podjelu populacije u fronte
  - Prvoj fronti postavi dobrotu $f=N$ (broj jedinki) i to korigiraj uporabom algoritma raspodjele funkcije cilja; zabilježi minimalni iznos i nešto umanjena vrijednost postaje početna dobrota za sljedeću frontu
  - Radi proporcionalnu selekciju

NSGA

- Srinivas, Deb (1994): ***Non-Dominated Sorting Genetic Algorithm*** (stranice 209 do 218)
- Nije elitistički!
- Udaljenost računa u *decision space*-u:

$$d = \sqrt{\sum_{K=1}^{n} \left(\frac{x_k^{(i)} - x_k^{(j)}}{x_k^{max} - x_k^{max}} \right)^2}$$

- Za sh(.) uzima α=2

NSGA-II

- Deb et al. (2000): ***Elitist Non-Dominated Sorting Genetic Algorithm*** (stranice 245 do 253)
- Zbog elitizma treba mehanizam očuvanja genetske raznolikosti

NSGA-II

- Algoritam
  - Iz populacije roditelja P_t stvori populaciju djece Q_t
  - Napravi uniju $R_t = P_t \cup Q_t$
  - Provedi nedominirano sortiranje nad R_t
  - Kreni frontu po frontu i tako dugo dok čitava fronta stane, dodaj je u novu populaciju
  - Fronta na kojoj ovo pukne – posebna obrada!

NSGA-II

- Fronta na kojoj je dodavanje stalo
  - Ne želimo proizvoljno odbaciti rješenja (moguć gubitak raznolikosti)
  - Umjesto toga, nad tom frontom provedi ***crowding-sort*** pa odaberi podskup rješenja koja su na najvećim udaljenostima
- Algoritam stvara djecu uporabom operatora ***crowded tournament selection***

NSGA-II, *crowded tournament sel.*

- Operator pretpostavlja da je za svakog roditelja izračunat stupanj nedominacije (*non-domination rank* r_i) i *crowding-distance* d_i
- d_i govori koliko oko rješenja i ima praznog prostora
- Operator *CTS* tada je definiran na sljedeći način ...

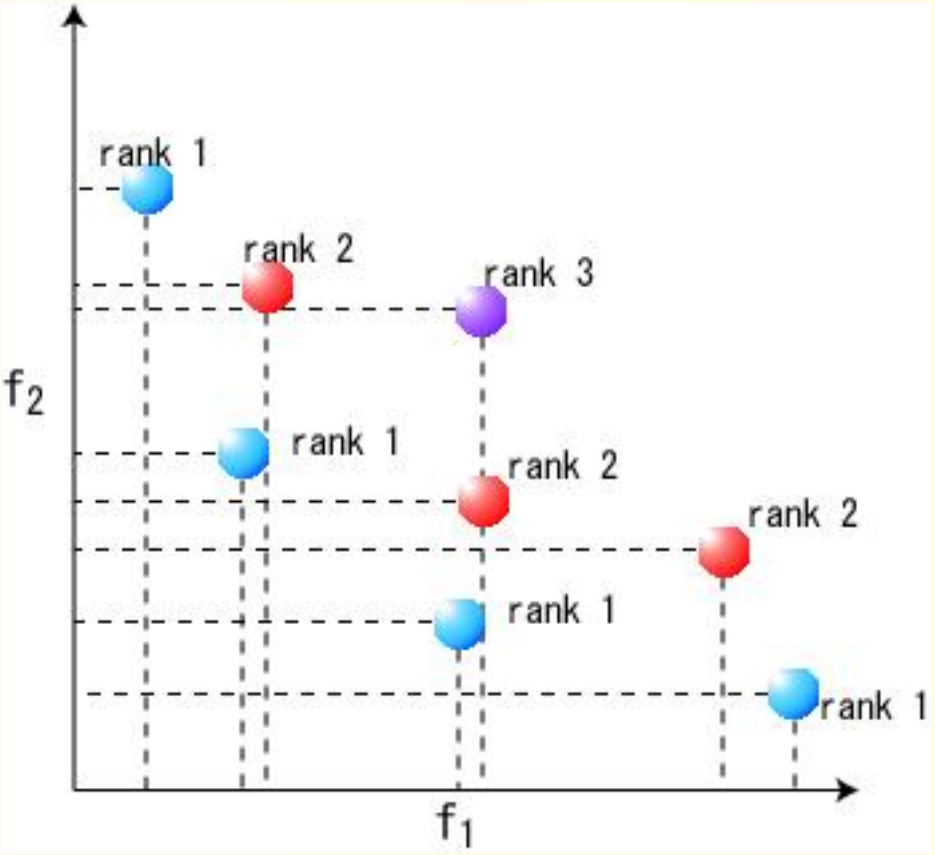
NSGA-II, *crowded tournament sel.*

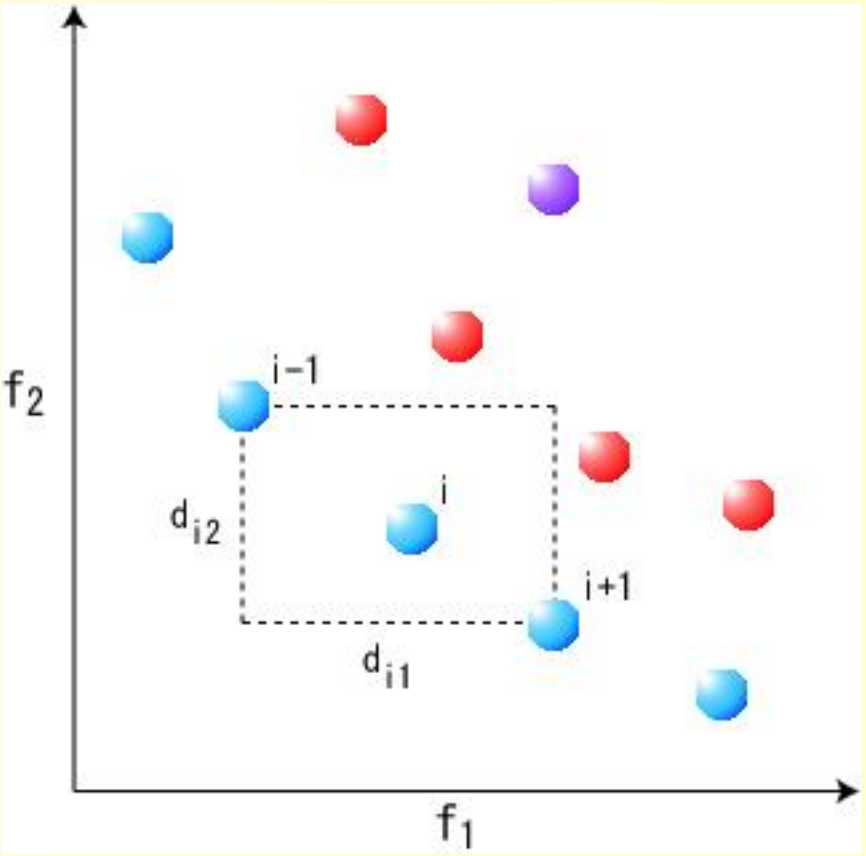
- Rješenje *i* pobjeđuje rješenje *j* ako je bilo koje od sljedećega ispunjeno
  - Stupanj nedominacije od *i* je manji od stupnja nedominacije od *j*, odnosno $r_i < r_j$
  - Ako imaju isti stupanj nedominacije ali rješenje *i* ima veći crowding-distance od *j*, tj: $r_i = r_j$ i $d_i > d_j$

NSGA-II, *crowding-sort*

- Cilj je utvrditi *crowding-distance*; mjeru koja govori o gustoći rješenja u okolici promatranog rješenja (veći *crowding-distance*, manja gustoća)
- Ideja: za svaki kriterij (ima ih M) i za svako rješenje pogledati po tom kriteriju koliko ima do najbližeg manjeg susjeda i najbližeg većeg susjeda

NSGA-II, *crowding-sort*





$$d_i = d_i + \frac{f_m^{i+1} - f_m^{i-1}}{f_m^{max} - f_m^{min}}$$

NSGA-II, *crowding-sort*

- Koraci:
  - Radimo sa skupom od F rješenja; $l=|F|$, $d_i=0$
  - Za svaki kriterij $m=1,2,...,M$ sortiraj skup rješenja od od najgoreg prema najbolje po iznosu m-te kriterijske funkcije (to mogu, to je broj)
  - Rubnim rješenjima (najmanjem i najvećem) dodaj neki veliki iznos (npr. ∞); za sva ostala rješenja (koja tada imaju lijevog i desnog susjeda) udaljenost rješenja povećaj za normiranu razliku fitnesa njegovih susjeda (slide ispred!)

NSGA-II, *crowding-sort*

- Složenost algoritma:
  - Trebam raditi onoliko sortova koliko imam kriterijskih funkcija ==> M puta
  - Za neko pametno sortiranje ==> $N \log(N)$
  - Ukupno: $M\,N \log(N)$
- Podsjetnik: kasnije još frontu na kojoj smo stali treba sortirati po tim udaljenostima