Višekriterijska optimizacija

Motivacija

- Obično je potrebno optimizirati više kriterija istodobno
- Kriteriji su najčešće konfliktni, nije ih moguće sve istodobno optimizirati
- Algoritmi koje smo do sada radili optimiziraju jedan kriterij

Višekriterijski problem

Minimiziraj/maksimiziraj

$$f_m(\vec{x}), \qquad m = 1,2,\ldots,M$$

Uz ograničenja

$$g_j(\vec{x}) \geq 0, \qquad j = 1,2,\ldots,J$$

$$h_k(\vec{x}) = 0, \qquad k = 1,2,\ldots,K$$

$$\vec{x}_i^{(L)} \leq \vec{x}_i \leq \vec{x}_i^{(L)}, \qquad i = 1,2,\ldots,n$$

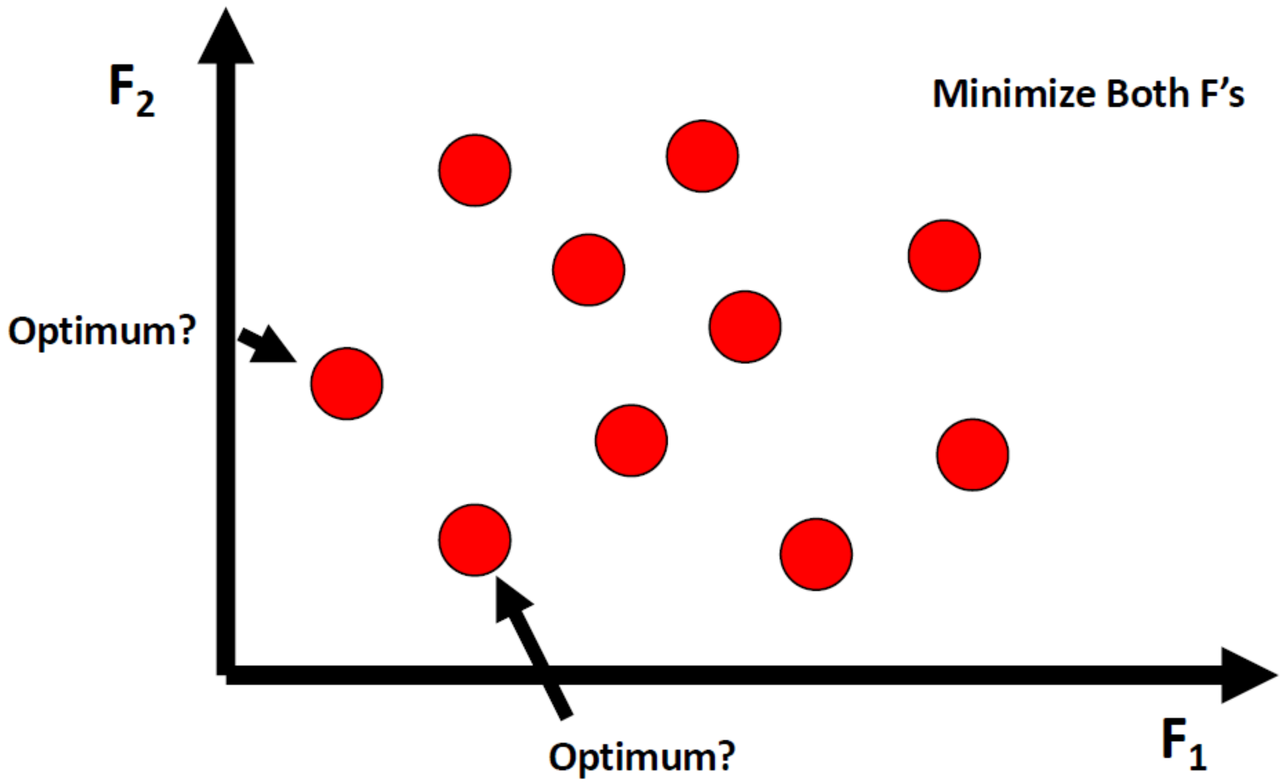
- *M* predstavlja broj kriterija koje je potrebno optimizirati

Višekriterijski problem

- *Feasible solution* – rješenje koje zadovoljava sva ograničenja
- *Unfeasible solution* – rješenje koje ne zadovoljava neko od ograničenja
- Radimo s dva prostora:
  - *Decision space* – prostor rješenja (onako koji pretražujemo)
  - *Objective space* – prostor kriterija koje optimiziramo (ocjena kvalitete rješenja)

Višekriterijski problem

- Koje rješenje je najbolje?



Višekriterijski problem

- U višekriterijskim problemima općenito imamo više rješenja koja su jednako „dobra”
- Rješenja koja su dobra po jednom kriteriju mogu biti loša po nekom drugom
- Kako odrediti odnose između rješenja?

Težinska kombinacija kriterija

- Ako je zadano više kriterija xy, može se definirati jedno-kriterijska funkcija kao linearna težinska kombinacija
$$f(x) = \lambda_1 f_1(x) + \lambda_2 f_1(x) + \ \dots + \lambda_n f_n(x)$$
- $\lambda_1, \lambda_2, \dots, \lambda_n$ predstavljaju pripadajuće težine

Težinska kombinacija kriterija

- Prednosti:
  - Višekriterijski problem je preveden u jednokriterijski
  - Može se iskoristiti bilo koji algoritam jednokriterijske optimizacije

Težinska kombinacija kriterija

- Nedostaci
  - Kako odrediti težine pojedinih kriterija
    - Odnosi između kriterija nisu unaprijed poznati
    - Nekada je teško preko težina izraziti odnose između kriterija
  - Dobije se samo jedno rješenje
    - Algoritam je potrebno pokrenuti više puta (s različitim težinama) kako bi dobili širi spektar rješenja
  - Možemo lako promašiti neka rješenja

Dominacija

- Definira odnos među rješenjima
- Kažemo da rješenje x_1 (Pareto) dominira (uz traženje minumuma po svim kriterijima) rješenje x_2 ako vrijedi:
  - $f_i(x_1) \leq f_i(x_2), \forall i \in \{1,2, \ldots k\}$ i
  - $f_j(x_1) < f_j(x_2)$, za barem jedan $j \in \{1,2, \ldots k\}$

Dominacija - svojstva

- Refleksivna - NE
- Simetrična - NE
- Tranzitivna - DA

Dominacija

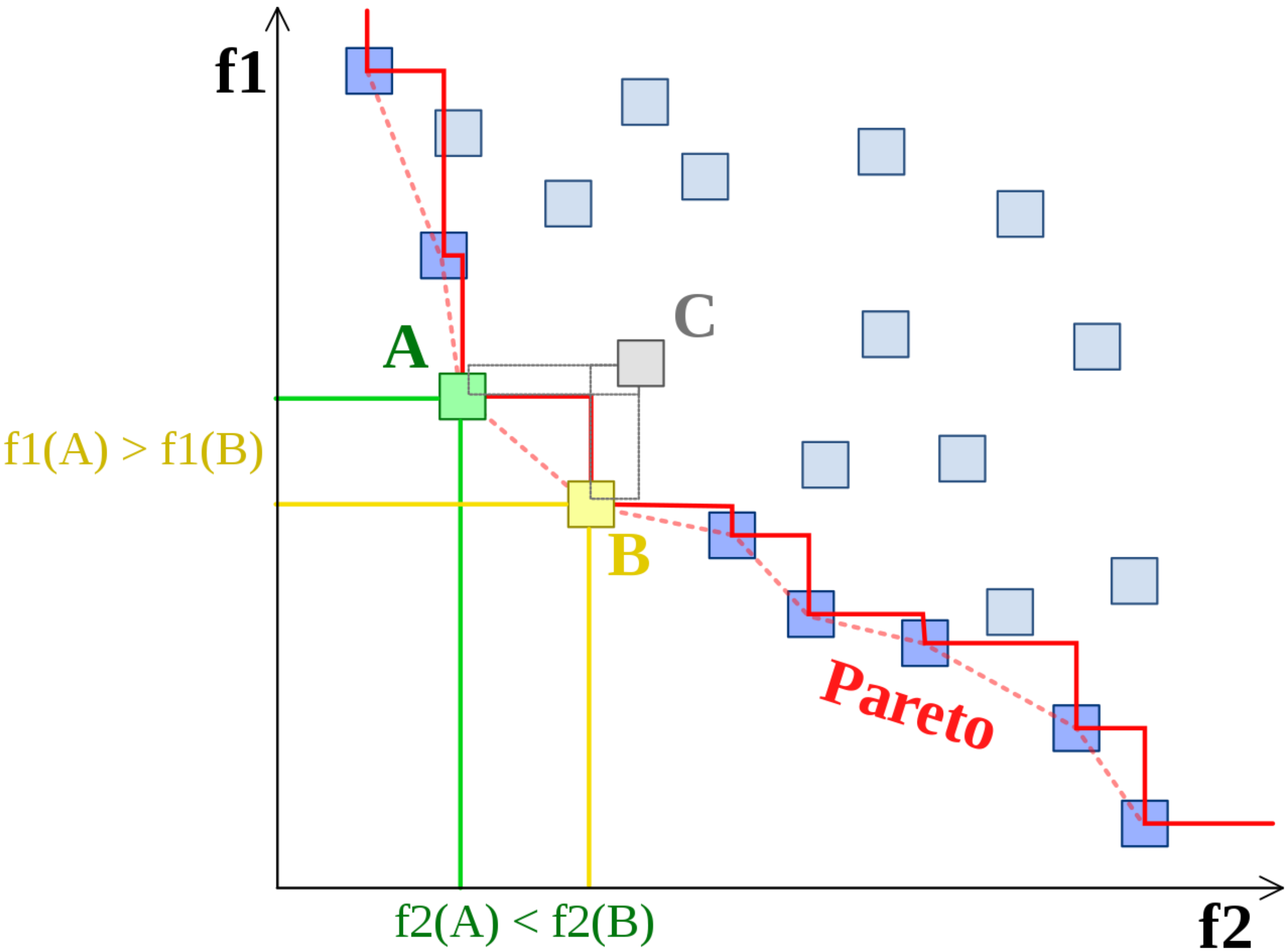
- Primjer: potrebno je odrediti dominaciju između rješenja (minimizacija po oba kriterija)

$$f(R_1) = (17,20)$$
$$f(R_2) = (15,21)$$
$$f(R_3) = (13,15)$$

Cilj višekriterijske optimizacije

- Pronaći skup rješenja nad kojim niti jedno drugo rješenje ne dominira – **Nedominirani skup**
- **Pareto optimalni skup** – skup svih nedominiranih rješenja koja zadovoljavaju sva ograničenja problema (*decision space*)
- **Pareto fronta** – skup vrijednosti rješenja iz Pareto optimalnog skupa u prostoru kriterija *(objective space)*

Pareto fronta



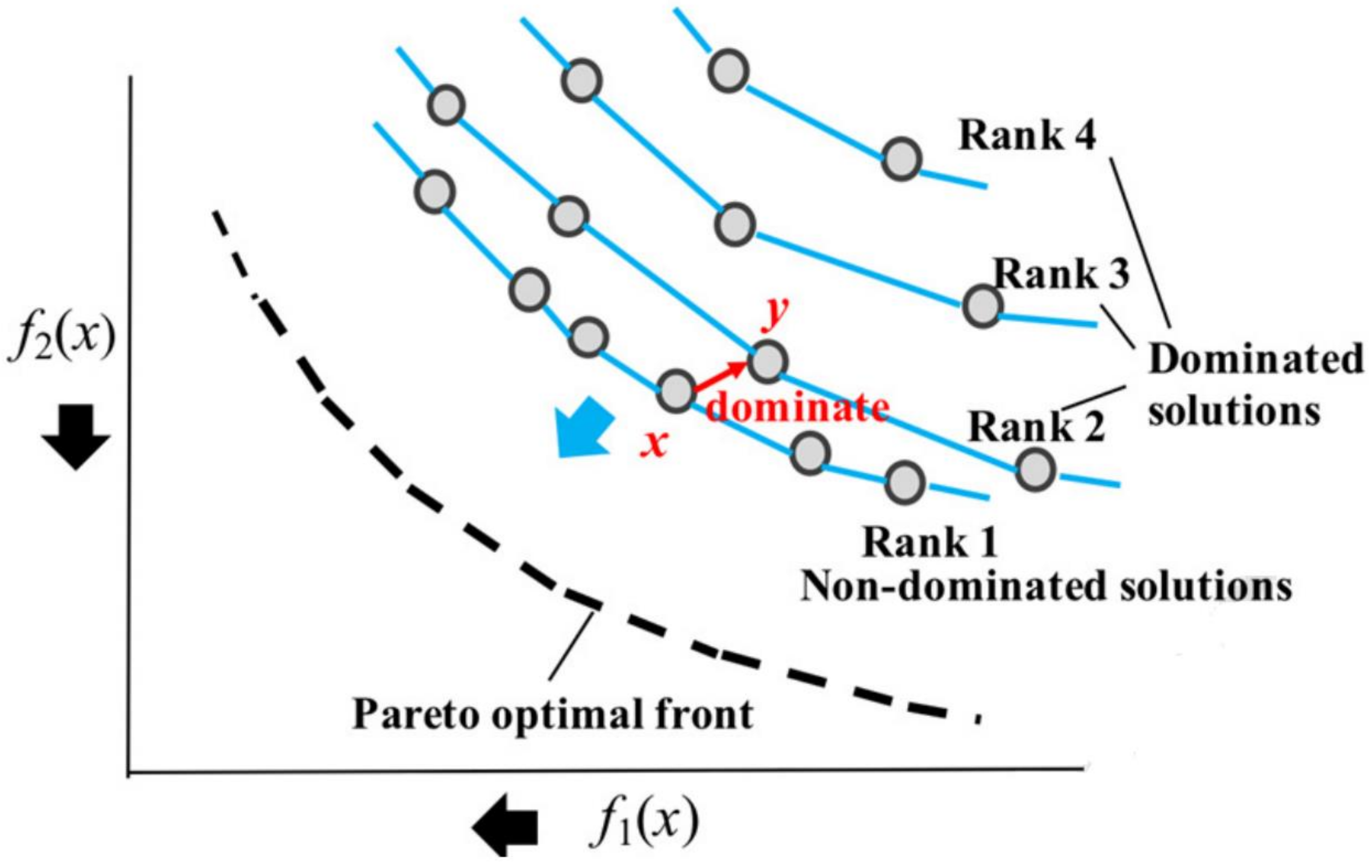
Cilj optimizacije

- Pronaći što bolju aproksimaciju prave Pareto fronte
- Na neki način moraju sortirati rješenja u fronte
- Rješenja možemo sortirati prema kriteriju dominacije

Nedominirano sortiranje

- Ideja je rješenja u populaciji sortirati u fronte.
- U prvoj fronti nalaze se rješenja u koja nisu dominirana od bilo kojeg drugog rješenja u populaciji
- U drugoj fronti nalaze se sva rješenja koja nisu dominirana od niti jednog drugog rješenja osim onih iz prve fronte
- Itd...
- Prva fronta predstavlja aproksimaciju prave Pareto fronte

Pareto fronte



Nedominirano sortiranje

- Kako raspodijeliti rješenja po frontama?
- Potrebno prvo odrediti rješenja u prvoj fronti, pa na temelju njih rješenja u idućoj, itd.
- Cijelu proceduru možemo napraviti u 2 prolaza

Nedominirano sortiranje

- Svakom rješenju u populaciji dodijeli se *rang*
- Rang određuje koliko rješenja dominira nad trenutnim rješenjem
- Možemo izračunati usporedbom (dominacije) svakog rješenja sa svim ostalima
- Za svako rješenje se također zapisuje skup koji govori nad kojim rješenjima ono dominira
- Složenost $O(MN^2)$ usporedbi

Nedominirano sortiranje

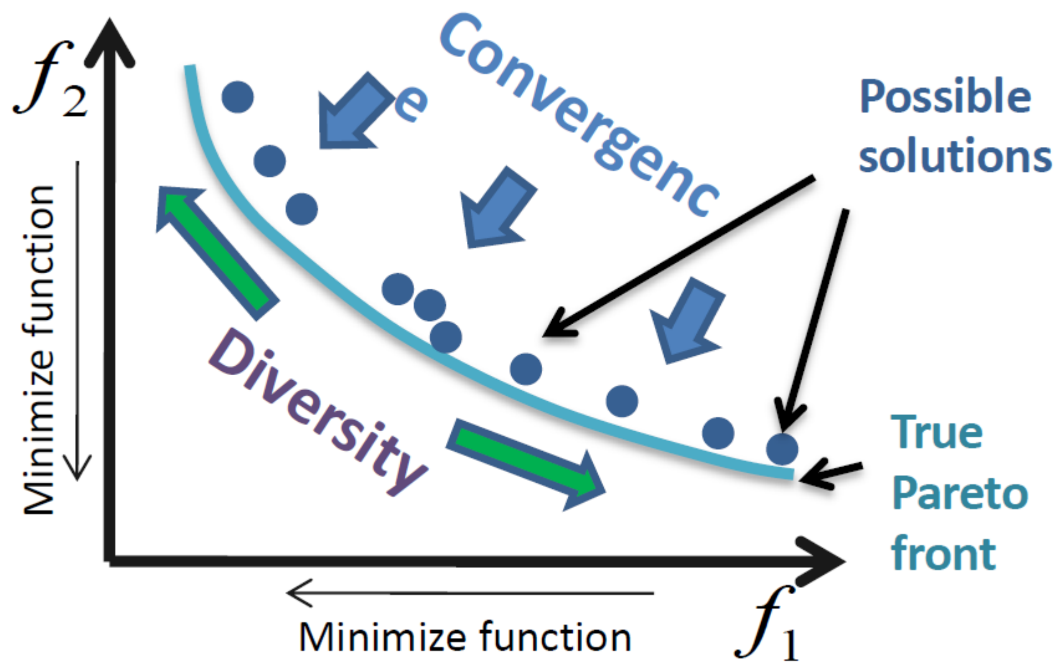
- Sva rješenja koja imaju rang 0 su nedominirana
- Ona pripadaju u 1. Pareto frontu
- Za svako rješenje u toj fronti pogledamo njegov skup rješenja nad kojima on dominira i smanjimo im rang za 1
- Sada sva rješenja koja imaju rang 0 pridaju idućoj fronti i postupak se ponavlja dok se sva rješenja ne dodijele nekoj fronti

Nedominirano sortiranje

- Primjer
  - R_1: $f_{R_1}(6,4)$
  - R_2: $f_{R_2}(5,2)$
  - R_3: $f_{R_3}(4,1)$
  - R_4: $f_{R_4}(3,3)$
  - R_5: $f_{R_5}(2,2)$

Ciljevi optimizacijskih algoritama

- Želimo 2 stvari:
  - Dobru raznolikost
  - Dobru konvergencije



NSGA

- Radi slično kao obični genetski algoritam
- U svakoj iteraciji podijeli populaciju u fronte
  - Prvoj fronti dodijeli vrijednost dobrote $f = N$ (N je broj jedinki)
  - Napravi korekciju dobrote u fronti
  - Najmanja vrijednost dobrote pomnožena nekim brojem blizu 1 koristi se za iduću frontu
- Radi se proporcionalna selekcija na temelju dodijeljenih dobrota

NSGA

```
Inicijaliziraj populaciju P
dok(kriterij zaustavljanja nije zadovoljen)
      T=P
      dok(|T|>0)
            B = nedominirana rješenja iz T
            postavi dobrotu jedinki u B
            odstrani sve jedinke u B iz T
      D – sastoji od N djece dobivene križanjem jedinki iz P
      P=D
```

NSGA

- Kako određujemo dobrotu
  - Jedinke u prvoj fronti imaju najveću dobrotu (npr. jednaku N)
  - Svaka iduća fronta ima manju dobrotu
  - Ideja je da sve jedinke unutar fronte nemaju apsolutno istu dobrotu
  - Naime, ako imamo puno rješenja u bliskom prostoru, ne želimo da sve imaju istu dobrotu -> želimo korigirati dobrotu takvih jedinki
    - Koristimo *fitness sharing*

NSGA

- Računamo normaliziranu udaljenost između rješenja *i, j*

$$d_{ij} = \sqrt{\sum_{k=1}^{M} \left(\frac{x_k^i - x_k^j}{x_k^{max} - x_k^{min}} \right)^2}$$

- Računa se udaljenost u prostoru rješenja (ne u prostoru funkcije cilja)

NSGA

- Na temelju izračunate težine računamo distancu za svako rješenje

$$Sh(d) = \begin{cases} 1 - \left(\frac{d}{\sigma_{share}}\right)^{\alpha} & ako\ d \leq \sigma_{share} \\ 0 & inače \end{cases}$$

- σ_{share} predstavlja parametar koji određuje koliko bliska rješenja se razmatraju

NSGA

- Konačno, za svako rješenje sumiramo sve vrijednosti
- Uvijek je >=1 (jer se rješenje uspoređuje samo sa sobom)

$$nc_i = \sum_{j=1}^{N} Sh(d_{ij})$$

$$f_i' = \frac{f_i}{nc_i}$$

- Funkcija cilja se onda prilagodi za sva rješenja

NSGA

- Puno nedostataka:
  - Nema elitizma!
  - Dijeljenje dobrote radi se u *decision space*-u (bliska rješenja možda imaju drugačije funkcije cilja)
  - Dodatan parametar koji je potrebno postaviti

NSGA-II

- Jedan od najpoznatijih i najčešće korištenih algoritama za višekriterijsku optimizaciju
- Rješava mnoge probleme NSGA algoritma
  - Uvodi elitizam
  - Koristi nedominirano sortiranje
  - Koristi posebnu mjeru za održavanje raznovrsnosti *crowding distance*

NSGA-II

- U svakoj iteraciji se od populacije P genetskim operatorima stvori nova populacija Q iste veličine
- Obje populacije se spoje u populaciju R
- Rješenja iz R se nedominirano sortiraju u fronte
- Imamo 2N rješenja, kako odabrati ona za iduću generaciju?

NSGA-II

- Uzimamo prvu frontu
- Ako sva rješenja iz nje stanu u iduću generaciju, sve ih kopiramo
- Isto radimo za sve fronte dok ne dođemo do one fronte kada se više cijela fronta ne može ubaciti u iduću generaciju
- Potrebno odrediti koja rješenja ćemo prebaciti

NSGA-II

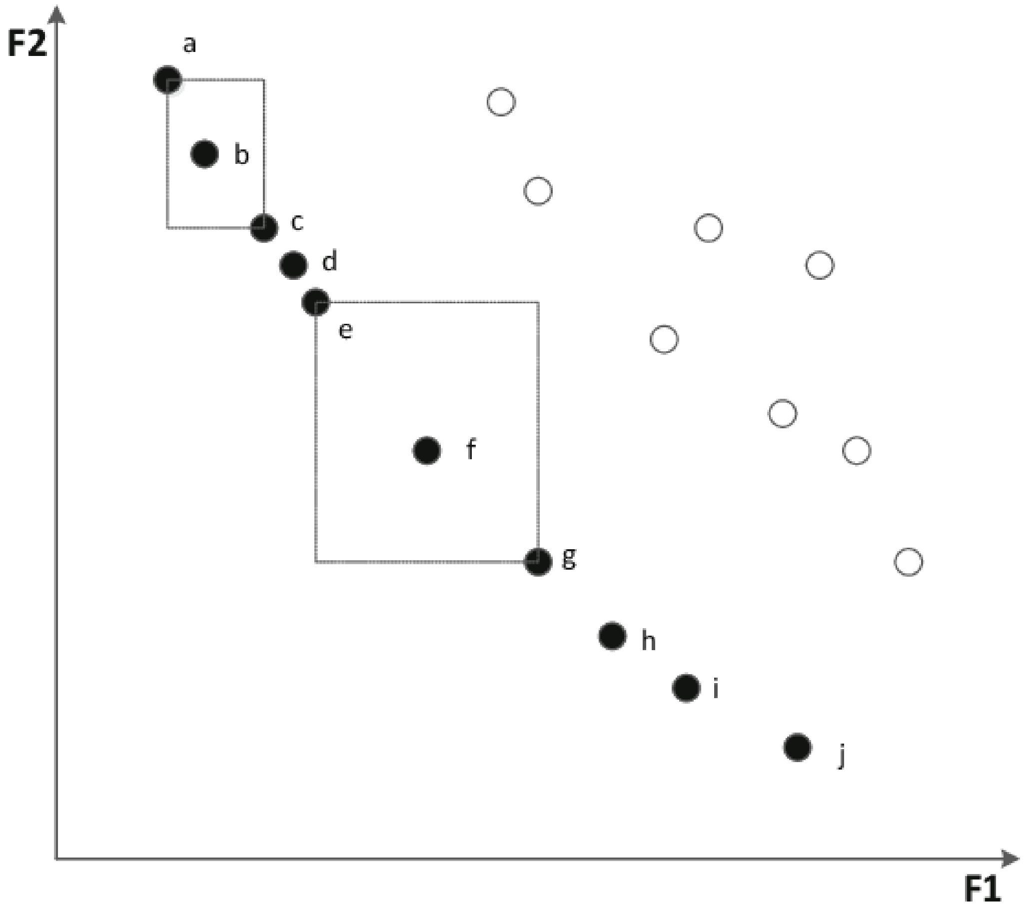
- Za rješenja u toj fronti računa se *crowding distance* tj. koliko su blizu susjedna rješenja iz te fronte
- Za svaki kriterij rješenja se sortiraju
- Za svako rješenje u fronti:
  - Ako su rubna rješenja postavi im vrijednost na ∞
  - Za sva ostala rješenja izračunaj udaljenost do susjeda

$$d_j = d_j + \frac{f_m^{(j+1)} - f_m^{(j-1)}}{f_m^{max} - f_m^{min}}$$

NSGA-II

- Prenesi ona rješenja koja imaju najveće udaljenosti
- To znači da se u blizini tih rješenja ne nalaze druga rješenja
- Bolje je uzeti njih jer na taj način možemo bolje očuvati raznolikost

NSGA-II



NSGA-II

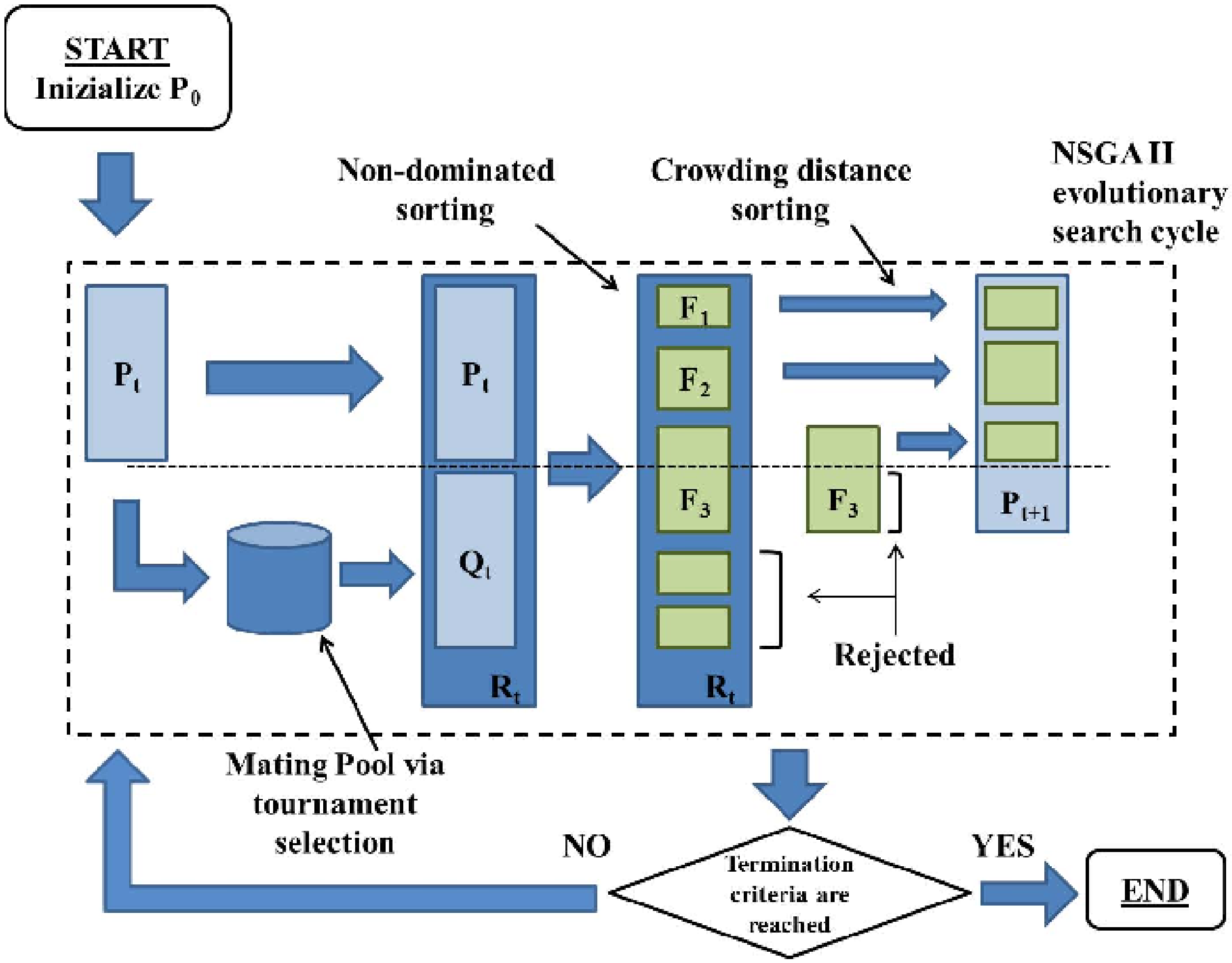
- Kako se stvara nova populacija?
- Turnirskom selekcijom odabiru se roditelji
- Pobjeđuje ona jedinka koja
  - Ima najmanji rang ako je jedina takva
  - Ako ima više jedinki s istim rangom, onda se odabire ona koja ima najveći crowding distance

Crowding distance

- Primjer

Parent population, P_t					Offspring population, Q_t				
Solution	x_1	x_2	f_1	f_2	Solution	x_1	x_2	f_1	f_2
1	0.31	0.89	0.31	6.10	a	0.21	0.24	0.21	5.90
2	0.43	1.92	0.43	6.79	b	0.79	2.14	0.79	3.97
3	0.22	0.56	0.22	7.09	c	0.51	2.32	0.51	6.51
4	0.59	3.63	0.59	7.85	d	0.27	0.87	0.27	6.93
5	0.66	1.41	0.66	3.65	e	0.58	1.62	0.58	4.52
6	0.83	2.51	0.83	4.23	f	0.24	1.05	0.24	8.54

NSGA-II


START
Inizialize P_0
Non-dominated sorting
Crowding distance sorting
NSGA II evolutionary search cycle
P_t
P_t
Q_t
R_t
F_1
F_2
F_3
F_3
R_t
P_{t+1}
Rejected
Mating Pool via tournament selection
NO
Termination criteria are reached
YES
END

NSGA-II

```
Inicijaliziraj populaciju P
dok(kriterij zaustavljanja nije zadovoljen)
      Q = stvori novu populaciju provođenjem operatora nad P
      R= Q ∪ P
      F = Nedominirano sortiraj R
      C – nova populacija
      i = 0
      dok (|C|+|Fi|<|P|)
             izračunaj crowding distance za Fi
             C = C ∪ Fi
             i++
      Izračunaj crowding distance za Fi
      Sortiraj Fi po crowded distanceu
      C = C ∪ Fi[1:N-|C|]
      P = C
```

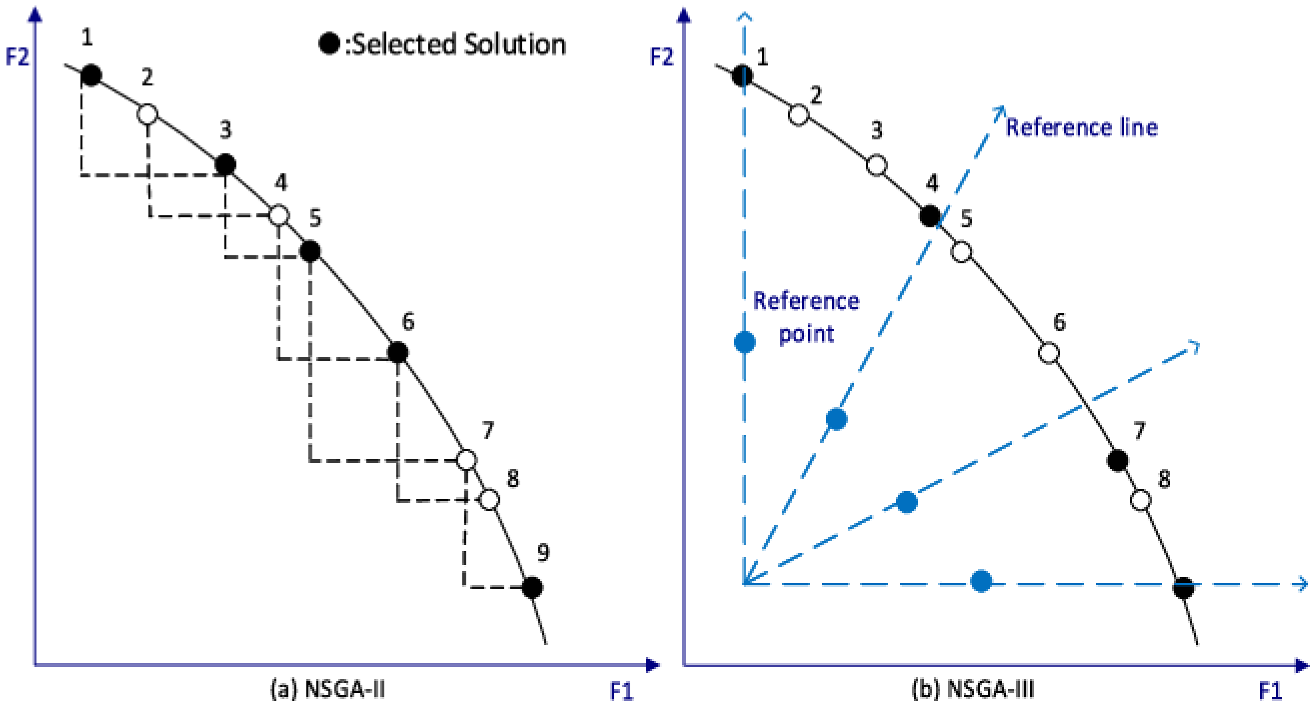
NSGA-II

- Ima inherentno ugrađen elitizam
  - Najbolja rješenja se uvijek prenose u iduću generaciju
- Nema dodatnih parametara
- Mjera *crowding distance* potiče raznolikost u zadnjoj fronti koja se prenosi u iduću generaciju

NSGA-III

- Unaprjeđenje NSGA-II algoritma
- Prikladniji za slučaj kada se optimizira više kriterija istodobno
- Jedina razlika je u načinu na koji se računa crowding distance

NSGA-III


F2
1
2
3
4
5
6
7
8
9
:Selected Solution
(a) NSGA-II
F1
F2
1
2
3
4
5
6
7
8
Reference line
Reference
point
(b) NSGA-III
F1

Ostali algoritmi

- Mnogo drugih algoritama (svakim danom sve više):
  - SPEA2
  - MOEA/D
  - ...

Zaključak

- Rijetko se koristi težinska linearna kombinacija
- Algoritmi najčešće rade na principu nedominiranog sortiranja
- Sve popularnije područje istraživanja