



FER, AG 2023.-2024.

Optimiranje evolucijskim računanjem

ORGANIZACIJA KOLEGIJA



Nastavnici

- Na kolegiju sudjeluju:
 - Nositelji:
 - Marko Čupić
 - Domagoj Jakobović
 - Predavanja
 - Marko Đurasević



Predavanja (1)

- Opterećenje predavanjima je $3 \times 13 = 39$ školskih sati
- Dolazak na predavanja se ne evidentira (ali svakako potiče)
- Literatura: Udžbenik
Marko Čupić. Prirodom inspirirani optimizacijski algoritmi. Metaheuristike.
Uploadan na Ferka u repozitorij.



Predavanja (2)

- Okvirni raspored tema
 1. Lokalna pretraga
 2. Numerička optimizacija
 3. Simulirano kaljenje + tabu pretraga
 4. Genetski (1)
 5. Genetski (2)
 6. Genetsko programiranje
 7. Ant algoritmi
 8. PSO algoritmi + ClonAlg

Gradivo do međuispita

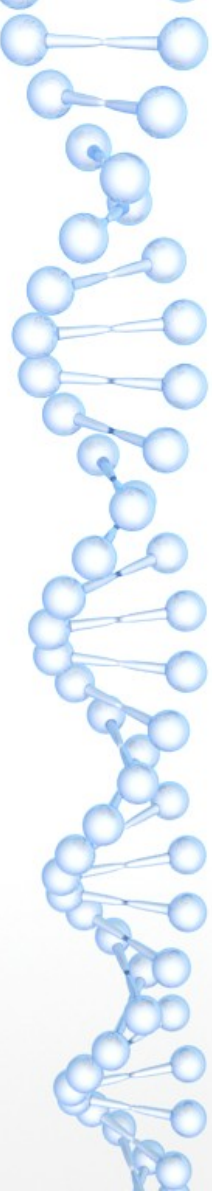
Gradivo nakon međuispita





Predavanja (3)

- Okvirni raspored tema
 - 9. Diferencijska evolucija
 - 10. Višekriterijska optimizacija
 - 11. Paralelizacija
 - 12. Usporedbe algoritama (SOOP i MOOP)
 - 13. Napredne teme iz evolucijskog računarstva



Domaće zadaće (1)

- Imamo 6 domaćih zadaća
 - Nije ih nužno predati
- Zadaće se pišu kod kuće i uploadaju na Ferka do zadanog roka
- Nema naknadnih uploada
- Domaće zadaće se prezentiraju pred nastavnikom u terminima laboratorijskih vježbi koji idu 4 puta u semestru
- Svaka zadaća nosi 5 bodova, ukupno $6 \times 5 = 30$



Domaće zadaće (2)

- Podržani programski jezici
 - Java
 - C#
 - C++
- U okviru uputa za domaće zadaće ponekad će biti dani “kosturi” rješenja kroz objektnu paradigmu, što se onda može implementirati u bilo kojem od gornjih programskih jezika



Laboratorijske vježbe

- Bit će dva tjedna prije međuispita, i dva tjedna nakon međuispita u kojima će se održati predaje domaćih zadaća (tj. Laboratorijske vježbe). Tjedni će biti definirani neknađno.
- Raspored termina unutar tjedna ćemo nanovo napraviti i objaviti kroz FERWeb (uz popratnu obavijest na FERWebu) i mikrosatnicu kroz Ferka



Domaće zadaće ↔ Laboratorijske vježbe

- Vjerojatno; moguća odstupanja – pratiti obavijesti
 - LAB1: DZ1 (teme 1-2)
 - LAB2: DZ2 (teme 3-4), DZ3 (teme 5-6)
 - LAB3: DZ4 (teme 7-9)
 - LAB4: DZ5 (teme 10-11), DZ6 (tema 12)



Kontinuirano polaganje predmeta

- Međuispit: 35 bodova
- Završni ispit: 35 bodova
- Domaće zadaće: 30 bodova
- Nema pojedinačnih pragova; vrijede samo pragovi za ocjene (dva slide-a dalje)



Polaganje predmeta na ispitnim rokovima

- Pismeni ispit: 70 bodova
- Domaće zadaće: 30 bodova (prenosimo bodove)
- Nema pojedinačnih pragova; vrijede samo pragovi za ocjene (sljedeći slide)



Bodovni pragovi za ocjene

- Dovoljan (2): 50 bodova
- Dobar (3): 63 boda
- Vrlo dobar (4): 75 bodova
- Odličan (5): 88 bodova



Infrastruktura

- Obavijesti:
<https://www.fer.unizg.hr/predmet/oer>
- Nastavni materijali, upload zadaća, bodovi:
<https://ferko.fer.hr/ferko/>
- Na kraju će se samo ukupni broj bodova pretočiti u FERWeb
- Komunikacija e-mailom:
 - sve upite slati na: oer@fer.hr

Rješavanje optimizacijskih problema algoritmima evolucijskog računanja u Javi Uvod

dr.sc. Marko Čupić

Fakultet elektrotehnike i računarstva
Sveučilište u Zagrebu
Akademska godina 2013./2014.

03. listopada 2013.

O čemu se tu radi?

Što je u imenu?

- Rješavanje:

O čemu se tu radi?

Što je u imenu?

- **Rješavanje**: suprotno od: naćuo sam nešto o tome, nemam pojma što bih s time radio

O čemu se tu radi?

Što je u imenu?

- **Rješavanje**: suprotno od: našao sam nešto o tome, nemam pojma što bih s time radio
- **Optimizacijski problema**:

O čemu se tu radi?

Što je u imenu?

- **Rješavanje**: suprotno od: naćuo sam nešto o tome, nemam pojma što bih s time radio
- **Optimizacijski problema**: puno stvari koje susrećemo u životu može se svesti na problem optimizacije – traćenje ekstrema funkcije: minimalna kazna, maksimalna dobit

O čemu se tu radi?

Što je u imenu?

- **Rješavanje**: suprotno od: naćuo sam nešto o tome, nemam pojma što bih s time radio
- **Optimizacijski problema**: puno stvari koje susrećemo u životu može se svesti na problem optimizacije – traćenje ekstrema funkcije: minimalna kazna, maksimalna dobit
- **Algoritmima evolucijskog računanja**:

O čemu se tu radi?

Što je u imenu?

- **Rješavanje**: suprotno od: naćuo sam nešto o tome, nemam pojma što bih s time radio
- **Optimizacijski problema**: puno stvari koje susrećemo u životu može se svesti na problem optimizacije – traćenje ekstrema funkcije: minimalna kazna, maksimalna dobit
- **Algoritmima evolucijskog računanja**: široka porodica algoritama, najćeće populacijskih no ne nužno, kojima upomoć može priskoćiti još širi skup algoritama

O čemu se tu radi?

Što je u imenu?

- **Rješavanje**: suprotno od: naćuo sam nešto o tome, nemam pojma što bih s time radio
- **Optimizacijski problema**: puno stvari koje susrećemo u životu može se svesti na problem optimizacije – traćenje ekstrema funkcije: minimalna kazna, maksimalna dobit
- **Algoritmima evolucijskog računanja**: široka porodica algoritama, najćeće populacijskih no ne nužno, kojima upomoć može priskoćiti još širi skup algoritama
- **U Javi**:

O čemu se tu radi?

Što je u imenu?

- **Rješavanje**: suprotno od: našao sam nešto o tome, nemam pojma što bih s time radio
- **Optimizacijski problema**: puno stvari koje susrećemo u životu može se svesti na problem optimizacije – traženje ekstrema funkcije: minimalna kazna, maksimalna dobit
- **Algoritmima evolucijskog računanja**: široka porodica algoritama, najčešće populacijskih no ne nužno, kojima upomoć može priskočiti još širi skup algoritama
- **U Javi**: programski jezik u kojem ćete raditi sve implementacije i GUI gdje bude potrebno

Organizacija

- Termini predavanja
- Domaće zadaće
- Predaja domaćih zadaća

Osooblje

- Marko Čupić – predavanja i sve grozno vezano uz vještinu
- TA
 - ... to be found yet ...

Što želimo rješavati?

Optimizacijske probleme koji su teški!

- Ako su optimizacijski problemi lagani, postoje jednostavniji algoritmi.
- Jednostavniji algoritmi su efikasniji!
- Evolucijsko računanje nije optimalno rješenje za svaki optimizacijski problem.
- Populacijski algoritmi su posebno zahtjevni – barataju s čitavim populacijama mogućih kandidata za rješenje
 - Visok računski trošak
 - Memorijski mogu biti vrlo zahtjevni

Što je optimizacijski problem?

Imamo podjele po nekoliko kriterija:

- Prema domeni nad kojom je problem definiran:
 - Kontinuirana domena (skup realnih brojeva)
 - Diskretna domena (kombinatorički problemi)

Što je optimizacijski problem?

Imamo podjele po nekoliko kriterija:

- Prema domeni nad kojom je problem definiran:
 - Kontinuirana domena (skup realnih brojeva)
 - Diskretna domena (kombinatorički problemi)
- Prema ograničenjima koja su postavljena na domenu:
 - Bez ograničenja (npr. čitav skup realnih brojeva)
 - Postoje ograničenja (primjerice, mora vrijediti $\sin(x_1) + e^{x_2 \cdot \cos x_3} \cdot x_4 > 0$)

Što je optimizacijski problem?

Imamo podjele po nekoliko kriterija:

- Prema domeni nad kojom je problem definiran:
 - Kontinuirana domena (skup realnih brojeva)
 - Diskretna domena (kombinatorički problemi)
- Prema ograničenjima koja su postavljena na domenu:
 - Bez ograničenja (npr. čitav skup realnih brojeva)
 - Postoje ograničenja (primjerice, mora vrijediti $\sin(x_1) + e^{x_2 \cdot \cos x_3} \cdot x_4 > 0$)
- Prema broju oprečnih kriterija koje pokušavamo zadovoljiti:
 - Jednokriterijska optimizacija (TSP: nađi najkraći Hamiltonov ciklus)
 - Višekriterijska optimizacija (TSP, ali dodatno minimiziraj broj preleta preko rijeka, i minimiziraj broj preleta preko planina; vjerojatno – više planina \leftrightarrow manje rijeka)

Vrste ograničenja

Razlikujemo dvije vrste ograničenja:

- Tvrda ograničenja (engl. *hard constraints*)
rješenje koje ih ne zadovoljava je neprihvatljivo; tvrda ograničenja dijele prostor rješenja u dva podprostora: prostor prihvatljivih rješenja (engl. *feasible solutions*) te u prostor neprihvatljivih rješenja (engl. *unfeasible solutions*).

Vrste ograničenja

Razlikujemo dvije vrste ograničenja:

- Tvrda ograničenja (engl. *hard constraints*)
rješenje koje ih ne zadovoljava je neprihvatljivo; tvrda ograničenja dijele prostor rješenja u dva podprostora: prostor prihvatljivih rješenja (engl. *feasible solutions*) te u prostor neprihvatljivih rješenja (engl. *unfeasible solutions*).
- Meka ograničenja (engl. *soft constraints*)
rješenje koje ih ne zadovoljava i dalje je prihvatljivo; no što su ova ograničenja manje prekršena, to je rješenje bolje; ova ograničenja govore o kvaliteti rješenja (raspored u kojem studenti imaju pauzu za ručak vs. raspored u kojem studenti nemaju pauzu za ručak vs. raspored u kojem studenti imaju puno pauza za ručak)

Jednokriterijska optimizacija

Definition (Problem jednokriterijske optimizacije.)

Općeniti problem jednokriterijske optimizacije definiran je na sljedeći način.

$$\begin{array}{ll} \text{Minimiziraj / maksimiziraj} & f(\vec{x}), \\ \text{uz zadovoljenje ograničenja} & g_j(\vec{x}) \geq 0, \quad j = 1, 2, \dots, J \\ & h_k(\vec{x}) = 0, \quad k = 1, 2, \dots, K \\ & x_i^L \leq x_i \leq x_i^U, \quad i = 1, 2, \dots, n. \end{array}$$

Rješenje \vec{x} je vektor decizijskih varijabli $\vec{x} = \{x_1, x_2, \dots, x_n\}$. Prvi i drugi skup ograničenja predstavljaju skup nejednakosti odnosno jednakosti koje moraju biti zadovoljene za sva prihvatljiva rješenja. Treći skup ograničenja definira donju i gornju granicu na vrijednosti koje decizijske varijable smiju poprimiti.

Globalni i lokalni optimumi

Definition (Globalni optimum.)

Kod jednokriterijske optimizacije, rješenje \vec{x}^* naziva se globalnim optimumom ako i samo ako rješenje \vec{x}^* pripada prostoru prihvatljivih rješenja i ako za svako drugo rješenje \vec{x} iz prostora prihvatljivih rješenja vrijedi $f(\vec{x}^*) \geq f(\vec{x})$, gdje je f funkcija dobrote rješenja.

Nije nužno jedinstven!

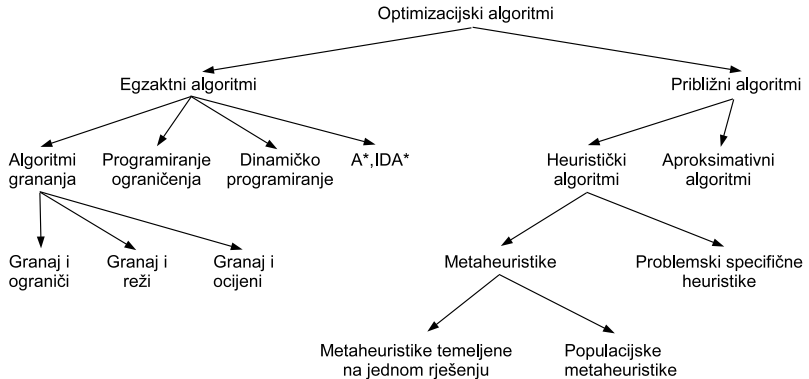
Globalni i lokalni optimumi

Definition (Lokalni optimum.)

Kod jednokriterijske optimizacije, rješenje \vec{x}^* naziva se lokalnim optimumom ako i samo ako rješenje \vec{x}^* pripada prostoru prihvatljivih rješenja i ako za svako drugo rješenje \vec{x} iz δ -okoline od \vec{x}^* , tj. uz $\delta > 0$ i $|\vec{x} - \vec{x}^*| \leq \delta$ vrijedi $f(\vec{x}^*) \geq f(\vec{x})$, gdje je f funkcija dobre rješenja.

Globalni optimum jest lokalni optimum!

Optimizacijski postupci



Slika: Podjela optimizacijskih algoritama

*ristika

Višestruke moguće interpretacije; smatrat ćemo da vrijedi sljedeće:

- Heuristika:

*ristika

Višestruke moguće interpretacije; smatrat ćemo da vrijedi sljedeće:

- **Heuristika**: jednostavna iskustvena pravila koja su specifična za pojedine probleme i pomažu u njegovom efikasnijem rješavanju; na prethodnom slideu – problemski specifične heuristike

*ristika

Višestruke moguće interpretacije; smatrat ćemo da vrijedi sljedeće:

- **Heuristika**: jednostavna iskustvena pravila koja su specifična za pojedine probleme i pomažu u njegovom efikasnijem rješavanju; na prethodnom slideu – problemski specifične heuristike
- **Metaheuristika**:

*ristika

Višestruke moguće interpretacije; smatrat ćemo da vrijedi sljedeće:

- **Heuristika**: jednostavna iskustvena pravila koja su specifična za pojedine probleme i pomažu u njegovom efikasnijem rješavanju; na prethodnom slideu – problemski specifične heuristike
- **Metaheuristika**: skup algoritamskih koncepata koji koristimo za definiranje heurističkih metoda primjenjivih na širok skup problema; heuristika opće namjene čiji je zadatak usmjeravanje problemski specifičnih heuristika prema području u prostoru rješenja u kojem se nalaze dobra rješenja

*ristika

Višestruke moguće interpretacije; smatrat ćemo da vrijedi sljedeće:

- **Heuristika**: jednostavna iskustvena pravila koja su specifična za pojedine probleme i pomažu u njegovom efikasnijem rješavanju; na prethodnom slideu – problemski specifične heuristike
- **Metaheuristika**: skup algoritamskih koncepata koji koristimo za definiranje heurističkih metoda primjenjivih na širok skup problema; heuristika opće namjene čiji je zadatak usmjeravanje problemski specifičnih heuristika prema području u prostoru rješenja u kojem se nalaze dobra rješenja
- **Hiperheuristika**:

*ristika

Višestruke moguće interpretacije; smatrat ćemo da vrijedi sljedeće:

- **Heuristika**: jednostavna iskustvena pravila koja su specifična za pojedine probleme i pomažu u njegovom efikasnijem rješavanju; na prethodnom slideu – problemski specifične heuristike
- **Metaheuristika**: skup algoritamskih koncepata koji koristimo za definiranje heurističkih metoda primjenjivih na širok skup problema; heuristika opće namjene čiji je zadatak usmjeravanje problemski specifičnih heuristika prema području u prostoru rješenja u kojem se nalaze dobra rješenja
- **Hiperheuristika**: heuristika čija je zadaća odabrati i podesiti prikladnu metaheuristiku

*ristika

Višestruke moguće interpretacije; smatrat ćemo da vrijedi sljedeće:

- **Heuristika**: jednostavna iskustvena pravila koja su specifična za pojedine probleme i pomažu u njegovom efikasnijem rješavanju; na prethodnom slideu – problemski specifične heuristike
- **Metaheuristika**: skup algoritamskih koncepata koji koristimo za definiranje heurističkih metoda primjenjivih na širok skup problema; heuristika opće namjene čiji je zadatak usmjeravanje problemski specifičnih heuristika prema području u prostoru rješenja u kojem se nalaze dobra rješenja
- **Hiperheuristika**: heuristika čija je zadaća odabrati i podesiti prikladnu metaheuristiku

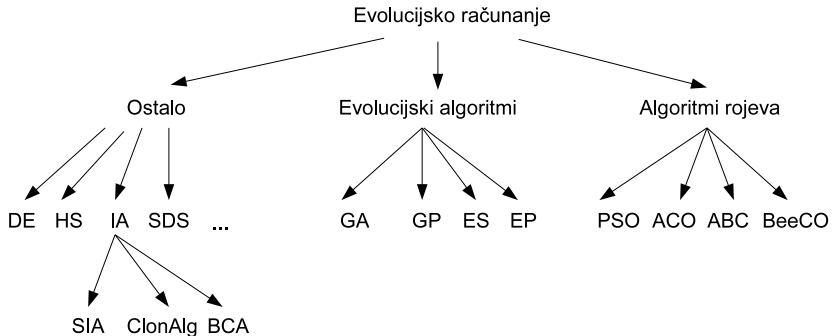
Možemo li optimirati parametre metaheuristike?

Heuristike

Heuristike dijelimo na:

- Konstrukcijske algoritme
- Algoritme lokalne pretrage (tj. algoritme iterativnog poboljšavanja)

Evolucijsko računanje: podskup metaheuristika



Slika: Podjela evolucijskog računanja na glavne grane. Uz svaku granu prikazani su i odabrani algoritmi.

Proces pretraživanja

Pretraživanje prostora kreću od jednog ili više početnih rješenja i potom temeljem postojećih rješenja (ili njihovog indirektnog utjecaja) generiraju nova rješenja.

Da bi optimizacijski proces u takvim uvjetima mogao biti uspješan, postupak pretraživanja prostora rješenja provodi se u dvije faze.

Proces pretraživanja

Pretraživanje prostora kreću od jednog ili više početnih rješenja i potom temeljem postojećih rješenja (ili njihovog indirektnog utjecaja) generiraju nova rješenja.

Da bi optimizacijski proces u takvim uvjetima mogao biti uspješan, postupak pretraživanja prostora rješenja provodi se u dvije faze.

- 1 **Gruba pretraga.** U fazi grube pretrage algoritam nasumično uzorkuje rješenja iz (nadamo se) čitavog prostora pretraživanja kako bi pronašao podprostor koji sadrži obećavajuća rješenja.

Proces pretraživanja

Pretraživanje prostora kreću od jednog ili više početnih rješenja i potom temeljem postojećih rješenja (ili njihovog indirektnog utjecaja) generiraju nova rješenja.

Da bi optimizacijski proces u takvim uvjetima mogao biti uspješan, postupak pretraživanja prostora rješenja provodi se u dvije faze.

- 1 **Gruba pretraga.** U fazi grube pretrage algoritam nasumično uzorkuje rješenja iz (nadamo se) čitavog prostora pretraživanja kako bi pronašao podprostor koji sadrži obećavajuća rješenja.
- 2 **Fina pretraga.** Nakon lociranja obećavajućeg podprostora nastupa faza fine pretrage u kojoj se pretraživanje fokusira. U ovom fazi istražuje se okolica pronađenog podprostora u potrazi za globalnim optimumom.

Proces pretraživanja

Pretraživanje prostora kreću od jednog ili više početnih rješenja i potom temeljem postojećih rješenja (ili njihovog indirektnog utjecaja) generiraju nova rješenja.

Da bi optimizacijski proces u takvim uvjetima mogao biti uspješan, postupak pretraživanja prostora rješenja provodi se u dvije faze.

- 1 **Gruba pretraga.** U fazi grube pretrage algoritam nasumično uzorkuje rješenja iz (nadamo se) čitavog prostora pretraživanja kako bi pronašao podprostor koji sadrži obećavajuća rješenja.
- 2 **Fina pretraga.** Nakon lociranja obećavajućeg podprostora nastupa faza fine pretrage u kojoj se pretraživanje fokusira. U ovom fazi istražuje se okolica pronađenog podprostora u potrazi za globalnim optimumom.

Lokalna pretraga, memetički algoritmi?

Proces pretraživanja

Genetski algoritam kao jedan od primjera algoritama evolucijskog računanja koristi dva operatora:

Proces pretraživanja

Genetski algoritam kao jedan od primjera algoritama evolucijskog računanja koristi dva operatora:

- 1 Operator **križanja**:

Proces pretraživanja

Genetski algoritam kao jedan od primjera algoritama evolucijskog računanja koristi dva operatora:

- 1 Operator **križanja**: primjerice, dijete je aritmetička sredina svojih roditelja

Proces pretraživanja

Genetski algoritam kao jedan od primjera algoritama evolucijskog računanja koristi dva operatora:

- 1 Operator **križanja**: primjerice, dijete je aritmetička sredina svojih roditelja; dovodi do kompresije populacije → lokalni optimum?

Proces pretraživanja

Genetski algoritam kao jedan od primjera algoritama evolucijskog računanja koristi dva operatora:

- 1 Operator **križanja**: primjerice, dijete je aritmetička sredina svojih roditelja; dovodi do kompresije populacije → lokalni optimum?
- 2 Operator **mutacije**:

Proces pretraživanja

Genetski algoritam kao jedan od primjera algoritama evolucijskog računanja koristi dva operatora:

- 1 Operator **križanja**: primjerice, dijete je aritmetička sredina svojih roditelja; dovodi do kompresije populacije → lokalni optimum?
- 2 Operator **mutacije**: uvođenje nasumične promjene u rješenje

Proces pretraživanja

Genetski algoritam kao jedan od primjera algoritama evolucijskog računanja koristi dva operatora:

- 1 Operator **križanja**: primjerice, dijete je aritmetička sredina svojih roditelja; dovodi do kompresije populacije → lokalni optimum?
- 2 Operator **mutacije**: uvođenje nasumične promjene u rješenje; dovodi do raspršenja populacije → uništava "naučeno"?

Proces pretraživanja

Genetski algoritam kao jedan od primjera algoritama evolucijskog računanja koristi dva operatora:

- 1 Operator **križanja**: primjerice, dijete je aritmetička sredina svojih roditelja; dovodi do kompresije populacije → lokalni optimum?
- 2 Operator **mutacije**: uvođenje nasumične promjene u rješenje; dovodi do raspršenja populacije → uništava "naučeno"?

Potrebno je pronaći dobar balans između ovih operatora!

Proces pretraživanja

Genetski algoritam kao jedan od primjera algoritama evolucijskog računanja koristi dva operatora:

- 1 Operator **križanja**: primjerice, dijete je aritmetička sredina svojih roditelja; dovodi do kompresije populacije → lokalni optimum?
- 2 Operator **mutacije**: uvođenje nasumične promjene u rješenje; dovodi do raspršenja populacije → uništava "naučeno"?

Potrebno je pronaći dobar balans između ovih operatora!

Većina algoritama su parametrizirani i za prilagodbu problemu koji rješavaju potrebno je pronaći povoljan skup parametara.

Najbolji optimizacijski algoritam?

Matematički je dokazano: ništa od toga...

Najbolji optimizacijski algoritam?

Matematički je dokazano: ništa od toga...

Wolpert i Macready dokazali da nema najboljeg algoritma:
no-free-lunch theorem

All algorithms that search for an extremum of a cost function perform exactly the same, according to any performance measure, when averaged over all possible cost functions. In particular, if algorithm A outperforms algorithm B on some cost functions, then loosely speaking there must exist exactly as many other functions where B outperforms A.

Najbolji optimizacijski algoritam?

Matematički je dokazano: ništa od toga...

Wolpert i Macready dokazali da nema najboljeg algoritma:
no-free-lunch theorem

All algorithms that search for an extremum of a cost function perform exactly the same, according to any performance measure, when averaged over all possible cost functions. In particular, if algorithm A outperforms algorithm B on some cost functions, then loosely speaking there must exist exactly as many other functions where B outperforms A.

To je upravo razlog za upoznavanje većeg broja optimizacijskih algoritama – treba naučiti kako radi algoritam te na koju je vrstu problema primjenjiv!

Prikaz rješenja

Ovisno o domeni nad kojom je definiran optimizacijski problem, rješenja možemo u računalu prikazivati na različite načine.

- Prikaz nizom bitova.
- Prikaz poljem ili vektorom brojeva.
- Prikaz permutacijama i matricama.
- Prikaz složenijim strukturama podataka.
- Prikaz stablima.
- Ostali prikazi.

Prikaz nizom bitova – direktno

Ponekad se kao rješenje traži lista (popis) zastavica što se trivijalno preslikava u polje bitova.

Primjerice, neka je s \mathcal{O} označen skup objekata $\{o_1, o_2, \dots, o_n\}$; zadatak je pronaći podskup skupa $o \subseteq \mathcal{O}$ nad kojim funkcija $f(o)$ poprima maksimum; funkcija $f(o)$ definirana je kao funkcija koja za svaki podskup $o \in \mathcal{O}$ vraća mjeru kvalitete tog podskupa.

Polje bitova može direktno kodirati svaki podskup; npr. rješenje 0011 bi predstavljalo podskup $\{o_3, o_4\}$.

Prikaz nizom bitova – kodiranjem

Ako su rješenja realni brojevi, može se koristiti neka vrsta kodiranja (primjerice, prirodni binarni kod, Grayev kod i slično).

- Pretpostavimo da koristimo binarni prikaz rješenja koji se sastoji od n bitova.
- Neka su legalne vrijednosti varijable x vrijednosti iz raspona $[x_{min}, x_{max}]$.
- Koristit ćemo prirodni binarni kod.

Prikaz nizom bitova – kodiranjem

Uz pretpostavke s prethodnog slidea:

- 1 Binarni niz se protumači kao cijeli broj. Označimo njegovu vrijednost s k . Ako se koriste n -bitovni binarni nizovi, tada će k poprimiti vrijednost iz skupa cijelih brojeva $\{0, 1, \dots, 2^n - 1\}$.
- 2 Vrijednost k preslika se na interval $[x_{min}, x_{max}]$. Pri tome se najčešće $k = 0$ preslika u vrijednost x_{min} , $k = 2^n - 1$ preslika u vrijednost x_{max} a ostale vrijednosti se linearno preslikaju na interval $[x_{min}, x_{max}]$. U tom slučaju se koristi izraz:

$$x = x_{min} + \frac{k}{2^n - 1} \cdot (x_{max} - x_{min}). \quad (1)$$

Preciznost pretrage? Proširivost na prikaz više varijabli?

Svi drugi prikazi

Pogledati u knjizi poglavlje 3 te načine modificiranja rješenja i kombiniranja rješenja.

- operatori modificiranja rješenja uobičajeno imaju ulogu diverzifikacije odnosno bijega iz lokalnog optimuma
- operatori kombiniranja rješenja uzimaju u obzir dva ili više postojećih rješenja te temeljem njih generiraju novo rješenje koje (nadamo se) zadržava i unaprijeđuje dobre karakteristike rješenja iz kojih je nastalo

Pogledati u knjizi diskusiju "Genotipski i fenotipski prikaz rješenja" na kraju poglavlja 3.

Klasika na djelu

Prije no što krenemo na algoritme evolucijskog računanja, pogledat ćemo nekoliko klasičnih optimizacijskih algoritama koji se često koriste kao lokalne pretrage (odnosno kao dopuna) algoritmima evolucijskog računanja.

- Iterativni algoritam pretraživanja
- Pohlepni algoritam uspona na vrh
- Višekratno pokretanje lokalne pretrage
- Iterativna lokalna pretraga
- Pohlepna randomizirana adaptivna procedura pretrage

Susjedstvo

Definition (Susjedstvo)

Neka je $x \in \mathcal{X}$ neko rješenje iz skupa mogućih rješenja \mathcal{X} .
Susjedstvo rješenja x , oznaka $\mathcal{N}(x)$, definirano je funkcijom susjedstva \mathcal{N} koja predstavlja preslikavanje $\mathcal{N} : \mathcal{X} \rightarrow 2^{\mathcal{X}}$. Ova funkcija svakom rješenju x pridružuje podskup skupa \mathcal{X} koji čini skup svih rješenja koja zovemo susjedima od rješenja x .

Kod problema optimizacije nad kontinuiranim prostorom, susjedstvo rješenja x najčešće se definira kao skup svih rješenja x' koja su od rješenja x udaljena za ne više od nekog fiksnog iznosa ϵ ; u tom slučaju susjedstvo je definirano kao:

$$\mathcal{N}(x) = \{x' : |x' - x| \leq \epsilon, \ x' \in \mathcal{X}\}.$$

Susjedstvo

Kod kombinatoričkih problema definiramo funkciju pomaka $\pi(x)$. Primjerice, ako kao prikaz rješenja koristimo prikaz nizom bitova, operator $\pi(x)$ možemo definirati kao operator koji na najviše jednom slučajno odabranom mjestu mijenja vrijednost bita. U tom slučaju, susjedstvo možemo definirati kao skup svih riječi koje je moguće dobiti uporabom operatora π nad zadanim rješenjem x .

Iterativni algoritam pretraživanja

$x(0)$... početno rješenje

$t = 0$

ponavljaj

Generiraj $N(x(t))$ tj. susjedstvo rješenja $x(t)$

$x(t+1)$ = odaberi neko rješenje iz generiranog
susjedstva $N(x(t))$

$t = t+1$

dok nije zadovoljen uvjet zaustavljanja

vрати $x(t)$

Pohlepni algoritam uspona na vrh

```
x(0) ... početno rješenje  
t = 0  
ponavljaj  
    Generiraj  $N(x(t))$  tj. susjedstvo rješenja  $x(t)$   
     $x(t+1)$  = odaberi najbolje rješenje iz generiranog  
        susjedstva  $N(x(t))$   
     $t = t+1$   
    ako je  $x(t)=x(t-1)$  prekini petlju  
dok nije zadovoljen uvjet zaustavljanja  
vrati  $x(t)$ 
```

Pohlepni algoritam uspona na vrh

U prethodnom algoritmu kompletno susjedstvo može se, ali i ne mora se generirati. Uobičajene su tri inačice.

- Generira se cjelokupno susjedstvo te se pronalazi najbolje rješenje. U slučaju da je susjedstvo veliko, ovaj pristup će biti računski izuzetno zahtjevan.
- Susjedstvo se generira rješenje po rješenje i postupak se zaustavlja čim se pronađe prvo bolje rješenje. U slučaju da je trenutno rješenje lokalni optimum, postupak se pretvara u prethodni slučaj u kojem se radi iscrpna pretraga cjelokupnog susjedstva.
- Posredstvom slučajnog mehanizma odabire se neko od rješenja iz susjedstva koja su bolja od trenutnog (dakle, nije nužno da će biti odabrano najbolje rješenje iz susjedstva).

Povećanje robusnosti algoritama

Jedan od problema koji je preostao jest zaglavljivanje u lokalnom optimumu. Strategije kojima se borimo protiv toga su:

Iterativno pokretanje algoritma s različitim početnih rješenja.

Pronađeni lokalni optimum ovisi o početnom rješenju.
Ideja: ponavljati postupak puno puta s različitim početnih rješenja.

Prihvatanje i rješenja koja nisu bolja od trenutnog. Ideja je sama po sebi jasna?

Promjena susjedstva. I opet, ideja je jasna: lokalni optimum koji je rezultat načina kako smo definirali susjedstvo može nestati u nekom drugom susjedstvu.

Višekratno pokretanje lokalne pretrage

Algoritam je poznat pod nazivom engl. *Multistart local search*.

$x'' = \text{null}$... još nema najboljeg rješenja

ponavljaj

x = generiraj slučajno početno rješenje

x' = primjeni lokalnu pretragu na rješenje x

 ako je $x'' = \text{null}$ ili ako je x' bolji od x'' tada

$x'' = x'$

 kraj

dok nije zadovoljen uvjet zaustavljanja

vraća x''

Iterativna lokalna pretraga

Poboljšanje prethodnog algoritma je algoritam poznat pod nazivom engl. *Iterated local search*.

```
x0 = generiraj slučajno početno rješenje  
    ili ga preuzmi izvana  
x = primjeni lokalnu pretragu na x0  
ponavljaj  
    x' = perturbiraj(x, povijesni podatci)  
    x'' = primjeni lokalnu pretragu na x'  
    x = prihvati(x, x'', memorija)  
dok nije zadovoljen uvjet zaustavljanja  
vрати najbolje pronađeno rješenje
```

Pohlepna randomizirana adaptivna procedura pretrage

Algoritam *pohlepna randomizirana adaptivna procedura pretrage* poznat pod nazivom GRASP, što je kratica od engl. *Greedy randomized adaptive search procedure* - GRASP.

```
x = null
ponavljaj
    x' = pohlepnim slučajnim algoritmom konstruiraj
        novo početno rješenje
    x'' = primjeni lokalnu pretragu na x'
    x = prihvati(x, x'')
dok nije zadovoljen uvjet zaustavljanja
vрати najbolje pronađeno rješenje, tj. x
```

Pohlepna randomizirana adaptivna procedura pretrage

Umjesto da se početna rješenja stvaraju sasvim slučajno, početna se rješenja konstruiraju element po element koristeći neku pohlepnu (i po mogućnosti randomiziranu) proceduru. Ovime se osigurava da rješenja od kojih se kreće nisu skroz slučajna, već u sebi sadrže dijelove rješenja koja se vjerojatno nalaze i u rješenju koje je globalni optimum, a također zbog načina na koji su konstruirana, već imaju kvalitetu koja je bitno bolja od tipičnih nasumično stvorenih rješenja.

Tipično se koriste ograničene liste kandidata, RCL, engl. *rescricted candidate list*.

Sljedeći puta...

- Numeričke optimizacije
- Popravljanje uspješnosti generiranja početnih rješenja
- ...

Što nas sve čeka?

- Numeričke optimizacije
- Algoritam simuliranog kaljenja
- Genetski algoritam
- Algoritam diferencijske evolucije
- Mravlji algoritmi
- Algoritam roja čestica
- Umjetni imunološki algoritmi
- Višekriterijska optimizacija
- Genetski algoritam za višekriterijsku optimizaciju
- Paralelizacija algoritama

Rješavanje optimizacijskih problema algoritmima evolucijskog računanja u Javi Numeričke optimizacije. Uspješnost generiranja početnih rješenja.

dr.sc. Marko Čupić

Fakultet elektrotehnike i računarstva
Sveučilište u Zagrebu
Akademska godina 2013./2014.

10. listopada 2013.

Layout

- 1 Optimizacija derivabilnih funkcija
 - Primjer funkcije jedne varijable
 - Primjer funkcije više varijabli
 - Odgovarajući pomak

- 2 Povećanje uspješnosti generiranja početnih rješenja

Prvi primjer

Pretpostavimo da je zadana funkcija $f(x) = 3x^2 - 6x - 45$ čiji tražimo minimum. To je skalarna funkcija koja je definirana nad skalarom $x \in \mathcal{R}$.

- Funkcija ima prvu derivaciju definiranu u svim točkama.

Prvi primjer

Pretpostavimo da je zadana funkcija $f(x) = 3x^2 - 6x - 45$ čiji tražimo minimum. To je skalarna funkcija koja je definirana nad skalarom $x \in \mathcal{R}$.

- Funkcija ima prvu derivaciju definiranu u svim točkama.

$$\frac{df}{dx} = 6x - 6$$

Prvi primjer

Pretpostavimo da je zadana funkcija $f(x) = 3x^2 - 6x - 45$ čiji tražimo minimum. To je skalarna funkcija koja je definirana nad skalarom $x \in \mathcal{R}$.

- Funkcija ima prvu derivaciju definiranu u svim točkama.

$$\frac{df}{dx} = 6x - 6$$

- Funkcija ima drugu derivaciju definiranu u svim točkama.

Prvi primjer

Pretpostavimo da je zadana funkcija $f(x) = 3x^2 - 6x - 45$ čiji tražimo minimum. To je skalarna funkcija koja je definirana nad skalarom $x \in \mathcal{R}$.

- Funkcija ima prvu derivaciju definiranu u svim točkama.

$$\frac{df}{dx} = 6x - 6$$

- Funkcija ima drugu derivaciju definiranu u svim točkama.

$$\frac{d^2 f}{dx^2} = 6$$

Prvi primjer

Pretpostavimo da je zadana funkcija $f(x) = 3x^2 - 6x - 45$ čiji tražimo minimum. To je skalarna funkcija koja je definirana nad skalarom $x \in \mathcal{R}$.

- Funkcija ima prvu derivaciju definiranu u svim točkama.

$$\frac{df}{dx} = 6x - 6$$

- Funkcija ima drugu derivaciju definiranu u svim točkama.

$$\frac{d^2 f}{dx^2} = 6$$

- Kako izgleda graf ove funkcije?

Prvi primjer

Pretpostavimo da je zadana funkcija $f(x) = 3x^2 - 6x - 45$ čiji tražimo minimum. To je skalarna funkcija koja je definirana nad skalarom $x \in \mathcal{R}$.

- Funkcija ima prvu derivaciju definiranu u svim točkama.

$$\frac{df}{dx} = 6x - 6$$

- Funkcija ima drugu derivaciju definiranu u svim točkama.

$$\frac{d^2 f}{dx^2} = 6$$

- Kako izgleda graf ove funkcije?
- Gdje je minimum?

Prvi primjer

Pretpostavimo da je zadana funkcija $f(x) = 3x^2 - 6x - 45$ čiji tražimo minimum. To je skalarna funkcija koja je definirana nad skalarom $x \in \mathcal{R}$.

- Funkcija ima prvu derivaciju definiranu u svim točkama.

$$\frac{df}{dx} = 6x - 6$$

- Funkcija ima drugu derivaciju definiranu u svim točkama.

$$\frac{d^2 f}{dx^2} = 6$$

- Kako izgleda graf ove funkcije?
- Gdje je minimum? Za $f' = 0$.

Prvi primjer

Pretpostavimo da je zadana funkcija $f(x) = 3x^2 - 6x - 45$ čiji tražimo minimum. To je skalarna funkcija koja je definirana nad skalarom $x \in \mathcal{R}$.

- Funkcija ima prvu derivaciju definiranu u svim točkama.

$$\frac{df}{dx} = 6x - 6$$

- Funkcija ima drugu derivaciju definiranu u svim točkama.

$$\frac{d^2 f}{dx^2} = 6$$

- Kako izgleda graf ove funkcije?
- Gdje je minimum? Za $f' = 0$. $6 \cdot x_{min} - 6 = 0 \rightarrow x_{min} = 1$

Prvi primjer

Predznak prve derivacije u nekoj točki x definira što treba napraviti toj točki da bi vrijednost funkcije porasla.

$$f' = \frac{df}{dx} = 6x - 6$$

Prvi primjer

Predznak prve derivacije u nekoj točki x definira što treba napraviti toj točki da bi vrijednost funkcije porasla.

$$f' = \frac{df}{dx} = 6x - 6$$

- Koja je vrijednost u minimumu?

Prvi primjer

Predznak prve derivacije u nekoj točki x definira što treba napraviti toj točki da bi vrijednost funkcije porasla.

$$f' = \frac{df}{dx} = 6x - 6$$

- Koja je vrijednost u minimumu?

$$\left. \frac{df}{dx} \right|_{x=1} = 6x - 6|_{x=1} = 6 - 6 = 0$$

Prvi primjer

Predznak prve derivacije u nekoj točki x definira što treba napraviti toj točki da bi vrijednost funkcije porasla.

$$f' = \frac{df}{dx} = 6x - 6$$

- Koja je vrijednost u $x > 1$

Prvi primjer

Predznak prve derivacije u nekoj točki x definira što treba napraviti toj točki da bi vrijednost funkcije porasla.

$$f' = \frac{df}{dx} = 6x - 6$$

- Koja je vrijednost u $x > 1$

$$\left. \frac{df}{dx} \right|_{x>1} = 6x - 6|_{x>1} > 0$$

Prvi primjer

Predznak prve derivacije u nekoj točki x definira što treba napraviti toj točki da bi vrijednost funkcije porasla.

$$f' = \frac{df}{dx} = 6x - 6$$

- Koja je vrijednost u $x > 1$

$$\left. \frac{df}{dx} \right|_{x>1} = 6x - 6|_{x>1} > 0$$

- Koja je vrijednost u $x < 1$

Prvi primjer

Predznak prve derivacije u nekoj točki x definira što treba napraviti toj točki da bi vrijednost funkcije porasla.

$$f' = \frac{df}{dx} = 6x - 6$$

- Koja je vrijednost u $x > 1$

$$\left. \frac{df}{dx} \right|_{x>1} = 6x - 6|_{x>1} > 0$$

- Koja je vrijednost u $x < 1$

$$\left. \frac{df}{dx} \right|_{x<1} = 6x - 6|_{x<1} < 0$$

Prvi primjer

Predznak prve derivacije u nekoj točki x definira što treba napraviti toj točki da bi vrijednost funkcije porasla.

$$f' = \frac{df}{dx} = 6x - 6$$

Prvi primjer

Predznak prve derivacije u nekoj točki x definira što treba napraviti toj točki da bi vrijednost funkcije porasla.

$$f' = \frac{df}{dx} = 6x - 6$$

- Ako je $f'(x) = 0$, vrijednost x je lokalni optimum (ili točka infleksije, ili ...).

Prvi primjer

Predznak prve derivacije u nekoj točki x definira što treba napraviti toj točki da bi vrijednost funkcije porasla.

$$f' = \frac{df}{dx} = 6x - 6$$

- Ako je $f'(x) = 0$, vrijednost x je lokalni optimum (ili točka infleksije, ili ...).
- Ako je $f'(x) > 0$, povećanjem x -a vrijednost funkcije će se također povećati.

Prvi primjer

Predznak prve derivacije u nekoj točki x definira što treba napraviti toj točki da bi vrijednost funkcije porasla.

$$f' = \frac{df}{dx} = 6x - 6$$

- Ako je $f'(x) = 0$, vrijednost x je lokalni optimum (ili točka infleksije, ili ...).
- Ako je $f'(x) > 0$, povećanjem x -a vrijednost funkcije će se također povećati.
- Ako je $f'(x) < 0$, smanjenjem x -a vrijednost funkcije će se također povećati.

Prvi primjer

Predznak prve derivacije u nekoj točki x definira što treba napraviti toj točki da bi vrijednost funkcije porasla.

$$f' = \frac{df}{dx} = 6x - 6$$

Za traženje minimuma funkcije tada ćemo okrenuti pravilo: potrebno je kretati se u smjeru negativne derivacije!

Prvi primjer

Predznak prve derivacije u nekoj točki x definira što treba napraviti toj točki da bi vrijednost funkcije porasla.

$$f' = \frac{df}{dx} = 6x - 6$$

Za traženje minimuma funkcije tada ćemo okrenuti pravilo: potrebno je kretati se u smjeru negativne derivacije!

- Ako je $f'(x) = 0$, vrijednost x je lokalni optimum (ili točka infleksije, ili ...).

Prvi primjer

Predznak prve derivacije u nekoj točki x definira što treba napraviti toj točki da bi vrijednost funkcije porasla.

$$f' = \frac{df}{dx} = 6x - 6$$

Za traženje minimuma funkcije tada ćemo okrenuti pravilo: potrebno je kretati se u smjeru negativne derivacije!

- Ako je $f'(x) = 0$, vrijednost x je lokalni optimum (ili točka infleksije, ili ...).
- Ako je $f'(x) > 0$, x ćemo smanjiti kako bi vrijednost funkcije pala: $x \leftarrow x - \delta$, $\delta > 0$.

Prvi primjer

Predznak prve derivacije u nekoj točki x definira što treba napraviti toj točki da bi vrijednost funkcije porasla.

$$f' = \frac{df}{dx} = 6x - 6$$

Za traženje minimuma funkcije tada ćemo okrenuti pravilo: potrebno je kretati se u smjeru negativne derivacije!

- Ako je $f'(x) = 0$, vrijednost x je lokalni optimum (ili točka infleksije, ili ...).
- Ako je $f'(x) > 0$, x ćemo smanjiti kako bi vrijednost funkcije pala: $x \leftarrow x - \delta$, $\delta > 0$.
- Ako je $f'(x) < 0$, x ćemo povećati kako bi vrijednost funkcije pala: $x \leftarrow x + \delta$, $\delta > 0$.

Prvi primjer

Predznak prve derivacije u nekoj točki x definira što treba napraviti toj točki da bi vrijednost funkcije porasla.

$$f' = \frac{df}{dx} = 6x - 6$$

Za traženje minimuma funkcije tada ćemo okrenuti pravilo: potrebno je kretati se u smjeru negativne derivacije!

- Ako je $f'(x) = 0$, vrijednost x je lokalni optimum (ili točka infleksije, ili ...).
- Ako je $f'(x) > 0$, x ćemo smanjiti kako bi vrijednost funkcije pala: $x \leftarrow x - \delta$, $\delta > 0$.
- Ako je $f'(x) < 0$, x ćemo povećati kako bi vrijednost funkcije pala: $x \leftarrow x + \delta$, $\delta > 0$.
- Ako je $f'(x) \neq 0$, možemo pisati $x \leftarrow x - \lambda \cdot f'(x)$, $\lambda > 0$.

Prvi primjer

Predznak prve derivacije u nekoj točki x definira što treba napraviti toj točki da bi vrijednost funkcije porasla.

$$f' = \frac{df}{dx} = 6x - 6$$

Za traženje minimuma funkcije tada ćemo okrenuti pravilo: potrebno je kretati se u smjeru negativne derivacije!

- Ako je $f'(x) = 0$, vrijednost x je lokalni optimum (ili točka infleksije, ili ...).
- Ako je $f'(x) > 0$, x ćemo smanjiti kako bi vrijednost funkcije pala: $x \leftarrow x - \delta$, $\delta > 0$.
- Ako je $f'(x) < 0$, x ćemo povećati kako bi vrijednost funkcije pala: $x \leftarrow x + \delta$, $\delta > 0$.
- Ako je $f'(x) \neq 0$, možemo pisati $x \leftarrow x - \lambda \cdot f'(x)$, $\lambda > 0$.

Za koliki iznos δ to smijemo promijeniti x ?

Layout

1 Optimizacija derivabilnih funkcija

- Primjer funkcije jedne varijable
- Primjer funkcije više varijabli
- Odgovarajući pomak

2 Povećanje uspješnosti generiranja početnih rješenja

Drugi primjer

Pretpostavimo da je zadana funkcija

$f(\vec{x}) = (x_1 - 4)^2 + (x_2 + 8)^2 + (x_3 + 5)^2$ čiji tražimo minimum. To je skalarna funkcija koja je definirana nad n -dimenzijskim vektorom $\vec{x} \in \mathcal{R}^3$.

- Više ne možemo računati *derivaciju* jer je to funkcija od više varijabli.

Drugi primjer

Pretpostavimo da je zadana funkcija

$f(\vec{x}) = (x_1 - 4)^2 + (x_2 + 8)^2 + (x_3 + 5)^2$ čiji tražimo minimum. To je skalarna funkcija koja je definirana nad n -dimenzijskim vektorom $\vec{x} \in \mathcal{R}^3$.

- Više ne možemo računati *derivaciju* jer je to funkcija od više varijabli.
- Možemo računati **gradijent** – pisat ćemo ga kao jednostupčani vektor čiji je element u i -tom retku parcijalna derivacija zadane funkcije po i -toj varijabli odnosno i -toj komponenti vektora \vec{x} .

Drugi primjer

Pretpostavimo da je zadana funkcija

$f(\vec{x}) = (x_1 - 4)^2 + (x_2 + 8)^2 + (x_3 + 5)^2$ čiji tražimo minimum. To je skalarna funkcija koja je definirana nad n -dimenzijskim vektorom $\vec{x} \in \mathcal{R}^3$.

- Više ne možemo računati *derivaciju* jer je to funkcija od više varijabli.
- Možemo računati **gradijent** – pisat ćemo ga kao jednostupčani vektor čiji je element u i -tom retku parcijalna derivacija zadane funkcije po i -toj varijabli odnosno i -toj komponenti vektora \vec{x} .

$$\nabla f(\vec{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \frac{\partial f}{\partial x_3} \end{bmatrix} = \begin{bmatrix} 2x_1 - 8 \\ 2x_2 + 16 \\ 2x_3 + 10 \end{bmatrix}$$

Drugi primjer

$$\nabla f(\vec{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \frac{\partial f}{\partial x_3} \end{bmatrix} = \begin{bmatrix} 2x_1 - 8 \\ 2x_2 + 16 \\ 2x_3 + 10 \end{bmatrix}$$

Drugi primjer

$$\nabla f(\vec{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \frac{\partial f}{\partial x_3} \end{bmatrix} = \begin{bmatrix} 2x_1 - 8 \\ 2x_2 + 16 \\ 2x_3 + 10 \end{bmatrix}$$

- Funkcija poprima ekstremnu vrijednost (ili sedlo, ili ...) u točkama u kojima je $\nabla f(\vec{x}) = \vec{0}$.

Drugi primjer

$$\nabla f(\vec{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \frac{\partial f}{\partial x_3} \end{bmatrix} = \begin{bmatrix} 2x_1 - 8 \\ 2x_2 + 16 \\ 2x_3 + 10 \end{bmatrix}$$

- Funkcija poprima ekstremnu vrijednost (ili sedlo, ili ...) u točkama u kojima je $\nabla f(\vec{x}) = \vec{0}$.
- Rezultira sustavom jednadžbi koji nije uvijek moguće riješiti egzaktno \rightarrow vodi na iterativne postupke koji malo po malo mijenjaju vrijednost trenutnog rješenja \vec{x} .

Drugi primjer

$$\nabla f(\vec{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \frac{\partial f}{\partial x_3} \end{bmatrix} = \begin{bmatrix} 2x_1 - 8 \\ 2x_2 + 16 \\ 2x_3 + 10 \end{bmatrix}$$

- Funkcija poprima ekstremnu vrijednost (ili sedlo, ili ...) u točkama u kojima je $\nabla f(\vec{x}) = \vec{0}$.
- Rezultira sustavom jednadžbi koji nije uvijek moguće riješiti egzaktno \rightarrow vodi na iterativne postupke koji malo po malo mijenjaju vrijednost trenutnog rješenja \vec{x} .
- Interpretacija gradijenta je međutim slična kao i interpretacija derivacije: promjena vrijednosti \vec{x} za skalirani iznos $\lambda \cdot \nabla f(\vec{x})$, $\lambda > 0$, u okolini od \vec{x} vodi ka povećanju vrijednosti funkcije.

Drugi primjer

$$\nabla f(\vec{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \frac{\partial f}{\partial x_3} \end{bmatrix} = \begin{bmatrix} 2x_1 - 8 \\ 2x_2 + 16 \\ 2x_3 + 10 \end{bmatrix}$$

- Funkcija poprima ekstremnu vrijednost (ili sedlo, ili ...) u točkama u kojima je $\nabla f(\vec{x}) = \vec{0}$.
- Rezultira sustavom jednadžbi koji nije uvijek moguće riješiti egzaktno \rightarrow vodi na iterativne postupke koji malo po malo mijenjaju vrijednost trenutnog rješenja \vec{x} .
- Interpretacija gradijenta je međutim slična kao i interpretacija derivacije: promjena vrijednosti \vec{x} za skalirani iznos $\lambda \cdot \nabla f(\vec{x})$, $\lambda > 0$, u okolini od \vec{x} vodi ka povećanju vrijednosti funkcije.
- Za minimizaciju stoga biramo korekciju proporcionalnu iznosu $-\lambda \cdot \nabla f(\vec{x})$. Za koliki λ se smijemo pomaknuti?

Layout

- 1 Optimizacija derivabilnih funkcija
 - Primjer funkcije jedne varijable
 - Primjer funkcije više varijabli
 - Odgovarajući pomak

- 2 Povećanje uspješnosti generiranja početnih rješenja

Koliko je prikladno?

- U slučaju funkcije od jedne varijable došli smo do ažuriranja:

$$x \leftarrow x - \lambda \cdot f'(x)$$

Koliko je prikladno?

- U slučaju funkcije od jedne varijable došli smo do ažuriranja:

$$x \leftarrow x - \lambda \cdot f'(x)$$

- U slučaju funkcije od više varijabli došli smo do ažuriranja:

$$\vec{x} \leftarrow \vec{x} - \lambda \cdot \nabla f(\vec{x})$$

Koliko je prikladno?

- U slučaju funkcije od jedne varijable došli smo do ažuriranja:

$$x \leftarrow x - \lambda \cdot f'(x)$$

- U slučaju funkcije od više varijabli došli smo do ažuriranja:

$$\vec{x} \leftarrow \vec{x} - \lambda \cdot \nabla f(\vec{x})$$

- U oba slučaja ostalo je neodgovoreno za koliko se smijemo pomaknuti?

Koliko je prikladno?

- U slučaju funkcije od jedne varijable došli smo do ažuriranja:

$$x \leftarrow x - \lambda \cdot f'(x)$$

- U slučaju funkcije od više varijabli došli smo do ažuriranja:

$$\vec{x} \leftarrow \vec{x} - \lambda \cdot \nabla f(\vec{x})$$

- U oba slučaja ostalo je neodgovoreno za koliko se smijemo pomaknuti?
 - Ako je $\lambda > 0$ premali, da smo uzeli veći, mogli smo funkciju više minimizirati u trenutnom koraku.

Koliko je prikladno?

- U slučaju funkcije od jedne varijable došli smo do ažuriranja:

$$x \leftarrow x - \lambda \cdot f'(x)$$

- U slučaju funkcije od više varijabli došli smo do ažuriranja:

$$\vec{x} \leftarrow \vec{x} - \lambda \cdot \nabla f(\vec{x})$$

- U oba slučaja ostalo je neodgovoreno za koliko se smijemo pomaknuti?
 - Ako je $\lambda > 0$ premali, da smo uzeli veći, mogli smo funkciju više minimizirati u trenutnom koraku.
 - Ako je $\lambda > 0$ preveliki, izlazimo iz *okolice* od \vec{x} pa tako daleko funkcija se može ponašati drugačije (npr. rasti).

Algoritmi pretraživanja u zadanom smjeru?

U oba prethodna slučaja zadana je točka (x ili \vec{x}) te smjer u kojem se treba pomaknuti ($-\lambda \cdot f'(x)$ ili $-\lambda \cdot \nabla f(\vec{x})$) – potrebno je pronaći prikladnu vrijednost za λ .

Algoritmi pretraživanja u zadanom smjeru?

U oba prethodna slučaja zadana je točka (x ili \vec{x}) te smjer u kojem se treba pomaknuti ($-\lambda \cdot f'(x)$ ili $-\lambda \cdot \nabla f(\vec{x})$) – potrebno je pronaći prikladnu vrijednost za λ .

- Postoji niz algoritama koji se mogu koristiti za rješavanje ovog problema.

Algoritmi pretraživanja u zadanom smjeru?

U oba prethodna slučaja zadana je točka (x ili \vec{x}) te smjer u kojem se treba pomaknuti ($-\lambda \cdot f'(x)$ ili $-\lambda \cdot \nabla f(\vec{x})$) – potrebno je pronaći prikladnu vrijednost za λ .

- Postoji niz algoritama koji se mogu koristiti za rješavanje ovog problema.
- Rijetko kada imamo egzaktno rješenje – najčešće se radi o iterativnim postupcima.

Algoritmi pretraživanja u zadanom smjeru?

U oba prethodna slučaja zadana je točka (x ili \vec{x}) te smjer u kojem se treba pomaknuti ($-\lambda \cdot f'(x)$ ili $-\lambda \cdot \nabla f(\vec{x})$) – potrebno je pronaći prikladnu vrijednost za λ .

- Postoji niz algoritama koji se mogu koristiti za rješavanje ovog problema.
- Rijetko kada imamo egzaktno rješenje – najčešće se radi o iterativnim postupcima.
- Kako je višedimenzijски slučaj općenitiji (\vec{x}), njega ćemo razmotriti.

Algoritmi pretraživanja u zadanom smjeru?

Zadana je funkcija $f(\vec{x})$ čiji se traži minimum, pri čemu je $\vec{x} \in \mathcal{R}^n$. Neka je $\vec{x}^{(0)}$ odabrano početno rješenje. Pretpostavimo također da je $\vec{d}^{(0)} \in \mathcal{R}^n$ vektor koji pokazuje u smjeru u kojem je potrebno modificirati trenutno rješenje kako bi vrijednost funkcije pala, gledano iz trenutne točke $\vec{x}^{(0)}$.

Uvodimo parametar λ te definiramo novu funkciju:

$$\theta(\lambda) = f(\vec{x}^{(0)} + \lambda \cdot \vec{d}^{(0)}).$$

Funkcija θ ovisi samo o λ ; za nju su $\vec{x}^{(0)}$ i $\vec{d}^{(0)}$ konstante. Sada je zadatak pronaći vrijednost λ^* koja minimizira funkciju $\theta(\lambda)$ i potom kao novo rješenje uzeti vrijednost:

$$\vec{x}^{(1)} = \vec{x}^{(0)} + \lambda^* \cdot \vec{d}^{(0)}.$$

Algoritmi pretraživanja u zadanom smjeru?

Neka je $f(\vec{x}) = (x_1 - 4)^2 + (x_2 + 8)^2 + (x_3 + 5)^2$,
 $\vec{x}^{(0)} = [6, -10, -9]^T$ te $\vec{d} = [-1, 2, 1]^T$. Lagano se možemo
 uvjeriti da će za male vrijednosti λ funkcija padati te će potom u
 jednom trenutku početi rasti:

λ	$\vec{x}^{(0)} + \lambda \cdot \vec{d}^{(0)}$	$\theta(\lambda)$
0	$[6, -10, -9]^T$	24
1	$[5, -8, -8]^T$	10
2	$[4, -6, -7]^T$	8
3	$[3, -4, -6]^T$	18
4	$[2, -2, -5]^T$	40

Algoritmi pretraživanja u zadanom smjeru?

Pogledajmo

$$\frac{d\theta(\lambda)}{d\lambda} = \nabla f(\vec{x})^T \cdot \vec{d}.$$

Algoritmi pretraživanja u zadanom smjeru?

Pogledajmo

$$\frac{d\theta(\lambda)}{d\lambda} = \nabla f(\vec{x})^T \cdot \vec{d}.$$

- Vrijedi

$$\left. \frac{d\theta(\lambda)}{d\lambda} \right|_{\lambda=0} < 0.$$

Zašto to vrijedi? Definirat ćemo stoga da je $\lambda_{lower} = 0$.

Algoritmi pretraživanja u zadanom smjeru?

Pogledajmo

$$\frac{d\theta(\lambda)}{d\lambda} = \nabla f(\vec{x})^T \cdot \vec{d}.$$

- Vrijedi

$$\left. \frac{d\theta(\lambda)}{d\lambda} \right|_{\lambda=0} < 0.$$

Zašto to vrijedi? Definirat ćemo stoga da je $\lambda_{lower} = 0$.

- Trebamo gornju ogradu za λ , oznaka λ_{upper} , uz koju će funkcija f ponovno početi rasti, odnosno za koju će $\frac{d\theta(\lambda)}{d\lambda_{upper}} > 0$.

Algoritmi pretraživanja u zadanom smjeru?

Pogledajmo

$$\frac{d\theta(\lambda)}{d\lambda} = \nabla f(\vec{x})^T \cdot \vec{d}.$$

- Vrijedi

$$\left. \frac{d\theta(\lambda)}{d\lambda} \right|_{\lambda=0} < 0.$$

Zašto to vrijedi? Definirat ćemo stoga da je $\lambda_{lower} = 0$.

- Trebamo gornju ogradu za λ , oznaka λ_{upper} , uz koju će funkcija f ponovno početi rasti, odnosno za koju će $\frac{d\theta(\lambda)}{d\lambda_{upper}} > 0$.
 - Ako takvu nemamo, možemo isprobavati $\lambda_{upper} = 1$, $\lambda_{upper} = 2$, $\lambda_{upper} = 4$, $\lambda_{upper} = 8$, ... dok ne pronađemo potreban λ_{upper} .

Algoritmi pretraživanja u zadanom smjeru?

Jednom kada imamo λ_{lower} i λ_{upper} , sigurno znamo da se λ koji minimizira funkciju nalazi između te dvije ograde. Možemo koristiti, primjerice, binarno raspolavljanje.

Metoda bisekcije

Korak 0. Postavi $k = 0$. Postavi $\lambda_l = \lambda_{lower}$ te $\lambda_u = \lambda_{upper}$.

Korak k. Postavi $\lambda = \frac{\lambda_l + \lambda_u}{2}$ i izračunaj $\left. \frac{d\theta(\lambda)}{d\lambda} \right|_{\lambda}$.

Ako je $\left. \frac{d\theta(\lambda)}{d\lambda} \right|_{\lambda} > 0$, redefiniraj $\lambda_u = \lambda$, $k = k + 1$.

Ako je $\left. \frac{d\theta(\lambda)}{d\lambda} \right|_{\lambda} < 0$, redefiniraj $\lambda_l = \lambda$, $k = k + 1$.

Ako je $\left. \frac{d\theta(\lambda)}{d\lambda} \right|_{\lambda} \approx 0$, stani jer smo došli dovoljno blizu.

Postoji još niz drugih sličnih metoda.

Algoritam pretraživanja u zadanom smjeru

Jednom kada znamo kako pronaći prikladan korak, možemo definirati porodicu algoritama pretraživanja u zadanom smjeru (engl. *line-search algorithms*).

Pretraživanje u zadanom smjeru.

Ponavljaj za $k = 1, 2, \dots$

Ako je $\vec{x}^{(k)}$ optimum, prekini s izvođenjem i vrati $\vec{x}^{(k)}$.

Inače

Utvrdi $\vec{d}^{(k)}$ – smjer pretrage

Pronađi $\lambda^* > 0$ – korak

Definiraj $\vec{x}^{(k+1)} = \vec{x}^{(k)} + \lambda^* \cdot \vec{d}^{(k)}$ – novo rješenje

Algoritam gradijentnog spusta

Metoda gradi linearni model funkcije oko trenutnog rješenja. Kao vektor $\vec{d}^{(k)}$ bira upravo minus gradijent funkcije koju je potrebno minimizirati.

$$\vec{d}^{(k)} = -\nabla f(\vec{x})$$

Algoritam gradijentnog spusta

Metoda gradi linearni model funkcije oko trenutnog rješenja. Kao vektor $\vec{d}^{(k)}$ bira upravo minus gradijent funkcije koju je potrebno minimizirati.

$$\vec{d}^{(k)} = -\nabla f(\vec{x})$$

- U ne baš rijetkim slučajevima, izračun gradijenta je računski skupa operacija!

Algoritam gradijentnog spusta

Metoda gradi linearni model funkcije oko trenutnog rješenja. Kao vektor $\vec{d}^{(k)}$ bira upravo minus gradijent funkcije koju je potrebno minimizirati.

$$\vec{d}^{(k)} = -\nabla f(\vec{x})$$

- U ne baš rijetkim slučajevima, izračun gradijenta je računski skupa operacija!
- Procjena optimalne vrijednosti λ^* može tražiti puno izračuna gradijenta funkcije; stoga se ponekad pomak radi i uz manje "dobru" vrijednost za λ .

Algoritam gradijentnog spusta

Metoda gradi linearni model funkcije oko trenutnog rješenja. Kao vektor $\vec{d}^{(k)}$ bira upravo minus gradijent funkcije koju je potrebno minimizirati.

$$\vec{d}^{(k)} = -\nabla f(\vec{x})$$

- U ne baš rijetkim slučajevima, izračun gradijenta je računski skupa operacija!
- Procjena optimalne vrijednosti λ^* može tražiti puno izračuna gradijenta funkcije; stoga se ponekad pomak radi i uz manje "dobru" vrijednost za λ .
- U najjednostavnijim varijantama za vrijednost λ se koristi mala pozitivna konstanta (npr. $\lambda = 0.1$ ili čak $\lambda = 0.01$) i ne provodi se pretraga za λ^* .

Algoritam gradijentnog spusta

Metoda gradi linearni model funkcije oko trenutnog rješenja. Kao vektor $\vec{d}^{(k)}$ bira upravo minus gradijent funkcije koju je potrebno minimizirati.

$$\vec{d}^{(k)} = -\nabla f(\vec{x})$$

- U ne baš rijetkim slučajevima, izračun gradijenta je računski skupa operacija!
- Procjena optimalne vrijednosti λ^* može tražiti puno izračuna gradijenta funkcije; stoga se ponekad pomak radi i uz manje "dobru" vrijednost za λ .
- U najjednostavnijim varijantama za vrijednost λ se koristi mala pozitivna konstanta (npr. $\lambda = 0.1$ ili čak $\lambda = 0.01$) i ne provodi se pretraga za λ^* .
- Primjer je vrlo poznati algoritam za učenje neuronskih mreža: *Backpropagation*.

Newtonova metoda

Metoda gradi kvadratni model $g(\vec{x})$ funkcije $f(\vec{x})$ oko trenutnog rješenja.

$$f(\vec{x} + \vec{\tau}) = f(\vec{x}) + \nabla f(\vec{x})^T \cdot \vec{\tau} + \frac{1}{2!} \vec{\tau}^T \nabla^2 f(\vec{x}) \vec{\tau} + \zeta \quad (1)$$

pri čemu su s ζ označene pogreške višeg reda. Za potrebe izgradnje Newtonovog algoritma u okolini točke \vec{x} funkcija $f(\vec{x})$ modelira se kvadratnom funkcijom $g(\vec{x})$ koja je definirana na sljedeći način:

$$g(\vec{x} + \vec{\tau}) = f(\vec{x}) + \nabla f(\vec{x})^T \cdot \vec{\tau} + \frac{1}{2!} \vec{\tau}^T \nabla^2 f(\vec{x}) \vec{\tau}. \quad (2)$$

$g(\vec{x} + \vec{\tau})$ pri tome promatramo kao funkciju od $\vec{\tau}$ i pitamo se koji $\vec{\tau}$ treba odabrati da bismo dobili minimum?

Newtonova metoda

$H(f) = \nabla^2 f(\vec{x})$ je Hesseova matrica odnosno matrica drugih parcijalnih derivacija. Ova matrica računa se na sljedeći način:

$$\begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix} \quad (3)$$

Newtonova metoda

Derivacijom

$$g(\vec{x} + \vec{\tau}) = f(\vec{x}) + \nabla f(\vec{x})^T \cdot \vec{\tau} + \frac{1}{2!} \vec{\tau}^T \nabla^2 f(\vec{x}) \vec{\tau}$$

po $\vec{\tau}$ slijedi:

$$\nabla g = \nabla f(\vec{x}) + H(\vec{x}) \cdot \vec{\tau}. \quad (4)$$

Uvjet za optimum je da je gradijent jednak 0:

$$\nabla f(\vec{x}) + H(\vec{x}) \cdot \vec{\tau} = \vec{0} \quad (5)$$

iz čega slijedi:

$$H(\vec{x}) \cdot \vec{\tau} = -\nabla f(\vec{x}) \quad (6)$$

odnosno

$$\vec{\tau} = -H(\vec{x})^{-1} \cdot \nabla f(\vec{x}) \quad (7)$$

Newtonova metoda

- Izraz $H(\vec{x})^{-1}$ predstavlja matrični inverz Hesseove matrice i upravo potreba za računom inverza čini ovaj postupak računski vrlo zahtjevnim.
- Dobiveni vektor $\vec{\tau}$ dalje se koristi kao vektor u čijem smjeru se radi pretraga prethodno opisanim algoritmom.

$$\vec{d}^{(k)} = \vec{\tau} = -H(\vec{x})^{-1} \cdot \nabla f(\vec{x})$$

- Treba odmah uočiti: ako je funkcija f kvadratna funkcija, tada joj njezin model g savršeno odgovara i postupak minimizacije će završiti u jednom koraku.

Razlog za neuspjeh?

Postupak stvaranja početnog rješenja može ne-uspjeti; najčešći razlog je kršenje tvrdih ograničenja. Jedan od prokušanih načina koji može pomoći:

Savjeti

- Pratite neuspjehe.
- Temeljem povijesti neuspjeha procijenjujte težinu (problematičnost; zahtjevnost) sastavnih dijelova rješenja.
- Dajte zahtjevnijim dijelovima rješenja priliku da prvi budu razriješeni.
- Izbor možete učiniti determinističkim ili vjerojatnosnim (teža komponenta, veća šansa da se ranije razriješi).

Pogledajte primjer i diskusiju u knjizi u poglavlju 5.3.

Rješavanje optimizacijskih problema algoritmima evolucijskog računanja u Javi Algoritam simuliranog kaljenja.

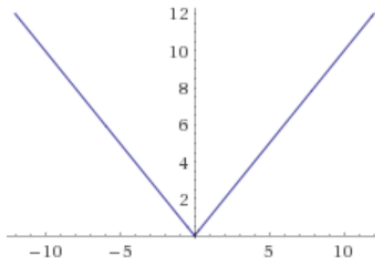
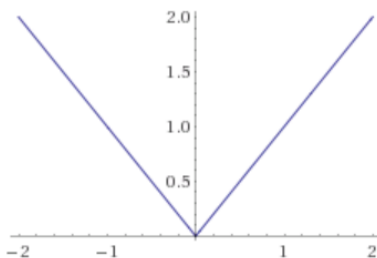
dr.sc. Marko Čupić

Fakultet elektrotehnike i računarstva
Sveučilište u Zagrebu
Akademska godina 2013./2014.

17. listopada 2013.

Primjer

Tražimo minimum funkcije: $f(x) = |x|$ u intervalu $[-10, 10]$.
Funkcija je prikazana na slici.



Minimum se postiže u $x^* = 0$ i iznosi $f(x^*) = 0$.

Pohlepni algoritam

Pretpostavimo da radimo sa sljedećim optimizacijskim algoritmom.

- Generiraj početno rješenje $\omega \in \Omega$

- Izračunaj vrijednost funkcije $f(\omega)$ te dobrotu rješenja $fit(\omega)$

- Ponavljaj dok nije zadovoljen uvjet zaustavljanja

 - Generiraj susjedno rješenje $\omega' \in N(\omega)$

 - Izračunaj vrijednost funkcije $f(\omega')$ te dobrotu rješenja $fit(\omega')$

 - Ako je $fit(\omega') > fit(\omega)$ prihvati ω' , tj. postavi $\omega \leftarrow \omega'$

- Kraj ponavljanja

- Vrati ω kao rješenje

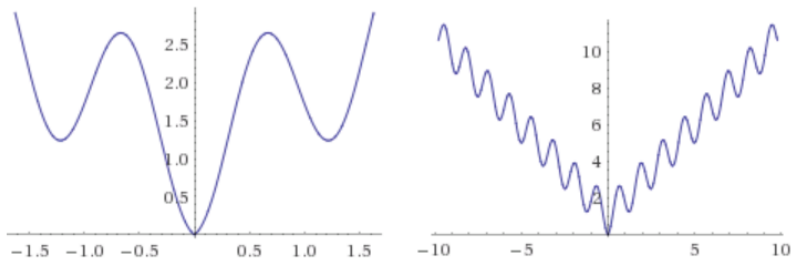
Pohlepni algoritam

Pretpostavimo sada da je u našem primjeru funkcija koja izvlači jedno slučajno rješenje iz susjedstva rješenja x definirana kao $x' = x + \text{unifRand}(-0.1, 0.1)$.

- Kakvo ponašanje možemo očekivati od pohlepnog algoritma uz takvo susjedstvo? Zašto? (isprobati)
- Što bi se promijenilo u ponašanju ako bismo susjedstvo definirali kao $x' = x + \text{unifRand}(-10, 10)$? Poopćite zaključak na funkcije velikog broja varijabli.

Primjer 2

Tražimo minimum funkcije: $f(x) = |x| + 1 - \cos(5 \cdot x)$ u intervalu $[-10, 10]$. Funkcija je prikazana na slici.



Minimum se postiže u $x^* = 0$ i iznosi $f(x^*) = 0$.

Primjer 2: pitanja

Pretpostavimo sada da je u našem primjeru funkcija koja izvlači jedno slučajno rješenje iz susjedstva rješenja x definirana kao $x' = x + \text{unifRand}(-0.1, 0.1)$.

- Kakvo ponašanje možemo očekivati od pohlepnog algoritma uz takvo susjedstvo? Zašto? (isprobati)
- Što bi se promijenilo u ponašanju ako bismo susjedstvo definirali kao $x' = x + \text{unifRand}(-10, 10)$? Poopćite zaključak na funkcije velikog broja varijabli.

Zaključak

Pohlepni algoritam već i na jednostavnim problemima može zapeti u lokalnom optimumu.

- Determinističko prihvatanje isključivo boljih rješenja nije dobra strategija!
- Treba osigurati mogućnost izlaska iz lokalnih optimuma.

Analogija iz stvarnog života

Kako bi se dobili metali s povoljnim karakteristikama, u metalurgiji se koristi proces *kaljenja* metala.

- Metal se zagrijava do vrlo visokih temperatura.
- Potom se postupno hladi.
- Posljedica:
 - konfiguracija atoma metala postupno se preslaguje u strukturu iz koje malo po malo nestaju sve nepravilnosti;
 - čitav sustav nizom takvih promjenom prelazi iz stanja visoke energije u stanje s niske energije
- Ako se proces hlađenja napravi prebrzo, neće biti dovoljno vremena da se rekonfiguriraju sve nepravilnosti \Rightarrow sustav će ostati u stanju nešto više energije.

Fizikalni opis postupka

Svaka rekonfiguracija položaja atoma sustav prevodi iz stanja energije E_1 u stanje energije E_2 . Time svakom rekonfiguracijom dolazi do promjene energije $\Delta E = E_2 - E_1$.

- Rekonfiguracije uz koje je $\Delta E < 0$ su uvijek i fizikalno moguće.
- Međutim, unatoč procesu hlađenja koji iz sustava uklanja energiju, moguće su i rekonfiguracije uz koje je $\Delta E > 0$, tj. uz koje dolazi do prelaska u više energetska stanje. Takve se rekonfiguracije događaju uz vjerojatnost određenu izrazom:

$$P(\Delta E) = \exp\left(\frac{-\Delta E}{k \cdot T}\right) \quad (1)$$

gdje je k Boltzmannova konstanta ($1.3806503 \cdot 10^{-23} \frac{\text{m}^2 \text{kg}}{\text{s}^2 \text{K}}$).
Što je ΔE veći, takva promjena je manje vjerojatna.

Rubno ponašanje

Što se događa s vjerojatnostima pri rubnim temperaturama? Za ograničeni iznos $\Delta E > 0$ te kada $t \rightarrow \infty$ vrijedi:

$$\begin{aligned}\lim_{t \rightarrow \infty} \frac{1}{\exp(\frac{\Delta E}{k \cdot t})} &= \frac{1}{\exp(\frac{\Delta E}{\infty})} \\ &= \frac{1}{\exp(0)} \\ &= \frac{1}{1} \\ &= 1\end{aligned}$$

U tom slučaju sve su promjene moguće, neovisno o tome koliko se time pravilnost konfiguracije atoma narušava (i energija raste).

Rubno ponašanje

Što se događa s vjerojatnostima pri rubnim temperaturama? Za ograničeni iznos $\Delta E > 0$ te kada $t \rightarrow 0$ vrijedi:

$$\begin{aligned}\lim_{t \rightarrow 0} \frac{1}{\exp(\frac{\Delta E}{k \cdot t})} &= \frac{1}{\exp(\frac{\Delta E}{0})} \\ &= \frac{1}{\exp(\infty)} \\ &= \frac{1}{\infty} \\ &= 0.\end{aligned}$$

U tom slučaju bilo koja promjena koja bi narušila pravilnost konfiguracije atoma i dovela do porasta energije je nemoguća.

Kaljenje je optimizacijski proces

Postupak kaljenja direktna je fizikalna realizacija optimizacijskog postupka kojim se sustav (metal) pokušava prevesti u stanje minimalne energije rekonfiguriranjem položaja atoma.

Kaljenje metala	Kombinatorička optimizacija
Moguća stanja sustava	Prihvatljiva rješenja
Energija sustava	Funkcija kazne
Promjena stanja sustava	Prelazak u susjedno rješenje
Temperatura	Parametar koji simulira temperaturu
Zamrznuto stanje	Optimum (lokalni ili globalni)

Minimizacija vs maksimizacija

Algoritmi direktno temeljeni na postupku kaljenja oponašaju proces smanjivanja energije sustava → rade minimizaciju.

Pri tome pozitivni ΔE označava **pogoršanje** rješenja i definirana je vjerojatnost prihvatanja takve promjene.

Negativni ΔE označava **poboljšanje** rješenja i takva se promjena prihvaća.

Stoga možemo postupati na sljedeći način:

- Ako radimo minimizaciju, E poistovjećujemo s iznosom funkcije čiji tražimo minimum, računamo $\Delta E = E_2 - E_1 \approx f_2 - f_1$.
- Ako radimo maksimizaciju, E poistovjećujemo s iznosom lošće rješenja (npr. minus iznos funkcije, računamo $\Delta E = E_2 - E_1 \approx (-f_2) - (-f_1) = f_1 - f_2$ odnosno ako poistovjetimo $E_i \approx f_i$ računamo $\Delta E = E_1 - E_2$. Dalje sve ostaje isto.

Algoritam simuliranog kaljenja

Generiraj početno rješenje $\omega \in \Omega$

Postavi brojač za promjenu temperature na $k = 0$

Odaberi plan hlađenja t_k i početnu temperaturu $t_0 \geq 0$

Odredi plan za M_k – broj ponavljanja petlje pri temperaturi t_k

Ponavljaj dok nije zadovoljen uvjet zaustavljanja

Ponavljaj za m je 1 do M_k

Generiraj susjedno rješenje $\omega' \in N(\omega)$

Izračunaj $\Delta_{\omega, \omega'} = f(\omega') - f(\omega)$

Ako je $\Delta_{\omega, \omega'} \leq 0$, prihvati ω' , tj. postavi $\omega \leftarrow \omega'$

Inače ako je $\Delta_{\omega, \omega'} > 0$,

postavi $\omega \leftarrow \omega'$ s vjerojatnošću $\exp(\frac{-\Delta_{\omega, \omega'}}{t_k})$

Kraj ponavljanja

Kraj ponavljanja

Vrati ω kao rješenje

Vjerojatnost prihvatanja lošijeg rješenja: vjerojatnost je skupa!

Klasično se koristi:

$$P(\Delta_{\omega, \omega'}) = \exp\left(\frac{-\Delta_{\omega, \omega'}}{t_k}\right)$$

Izračun eksponencijalne funkcije računski je vrlo skup. Alternativa:

$$P(\Delta_{\omega, \omega'}) = \begin{cases} a_1 x^{k-1} & \text{ako je } \Delta_{\omega, \omega'} > 0, \\ 1 & \text{inače.} \end{cases}$$

Uočiti: vjerojatnost ne ovisi o iznosu pogoršanja – računski efikasnije.

Neki od planova hlađenja

Plan hlađenja definira način na koji se u sustavu mijenja temperatura. U praksi istraživani čitavi niz izvedbi.

- Linearni

$$T_k = T_0 - k \cdot \beta$$

- Geometrijski

$$T_k = \alpha^k \cdot T_0$$

- vidi knjigu za druge...

Algoritam ograničenog demona

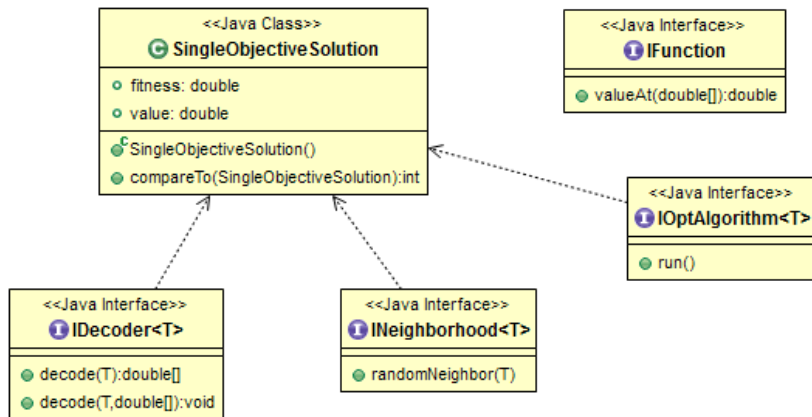
U sustavu postoji demon koji raspolaže određenom količinom energije. Promjena sustava u više energetske stanje moguća je samo ako demon može isporučiti potrebnu energiju (bez koje time ostaje). Prelaskom u niže energetske stanje energija se vraća demonu. Međutim, demon ne može imati više od unaprijed zadanog maksimuma.

- ❶ Odaberi početno rješenje ω .
- ❷ Odaberi početnu energiju demona $D = D_0 > 0$.
- ❸ Ponavljaj dok nije zadovoljen uvjet zaustavljanja:
 - ❶ Generiraj susjedno rješenje $\omega' \in N(\omega)$.
 - ❷ Izračunaj promjenu energije $\Delta E = f(\omega') - f(\omega)$.
 - ❸ Ako je $\Delta E \leq D$ prihvati novo rješenje: $\omega \leftarrow \omega'$ i korigiraj energiju demona $D \leftarrow D - \Delta E$.
 - ❹ U suprotnom odbaci rješenje ω' .
 - ❺ Ako je $D > D_0$, korigiraj $D \leftarrow D_0$.

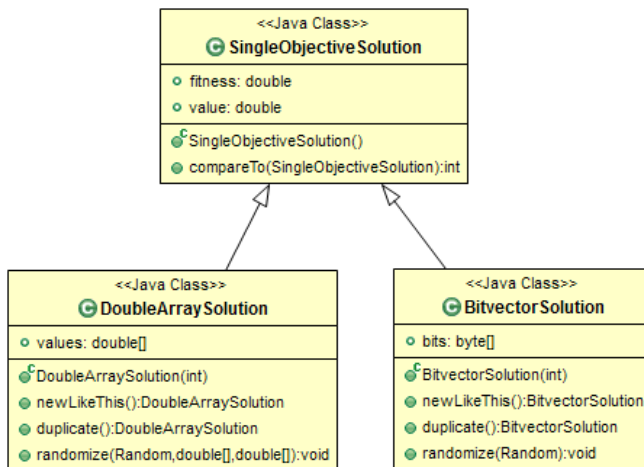
Algoritam kaljenog demona

- ❶ Odaberi početno rješenje ω .
- ❷ Odaberi početnu energiju demona $D = D_0 > 0$.
- ❸ Ponavljaj dok nije zadovoljen uvjet zaustavljanja:
 - ❶ Generiraj susjedno rješenje $\omega' \in N(\omega)$.
 - ❷ Izračunaj promjenu energije $\Delta E = f(\omega') - f(\omega)$.
 - ❸ Ako je $\Delta E \leq D$ prihvati novo rješenje: $\omega \leftarrow \omega'$ i korigiraj energiju demona $D \leftarrow D - \Delta E$.
 - ❹ U suprotnom odbaci rješenje ω' .
 - ❺ Ako je postignut ekvilibrij, umani energiju demona prema planu; npr. $D \leftarrow \alpha \cdot D$.

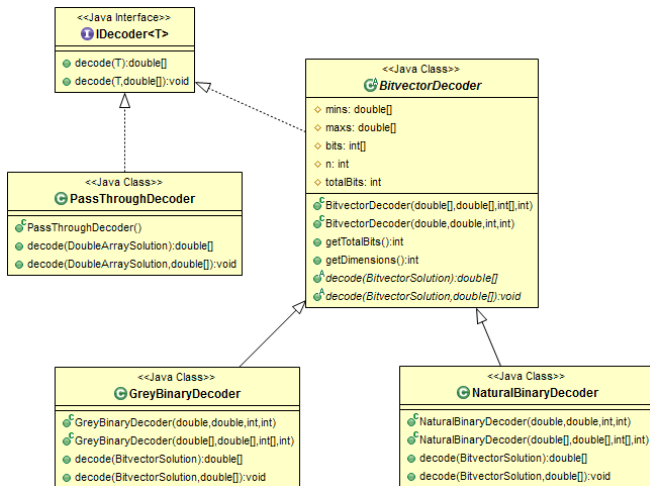
Primjer organizacije koda (1)



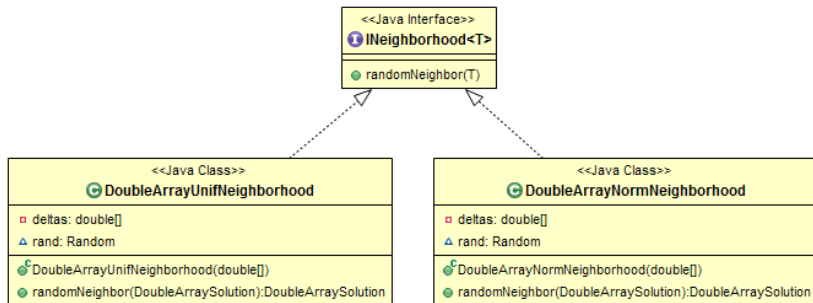
Primjer organizacije koda (2)



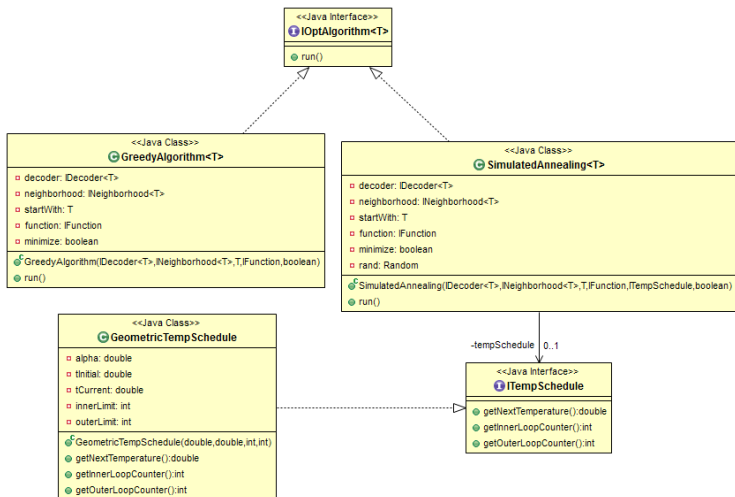
Primjer organizacije koda (3)



Primjer organizacije koda (4)



Primjer organizacije koda (5)



Genetski algoritam

Pregled

- Uvod (motivacija, koraci)
- Vrste genetskih algoritama
- Prikazi rješenja
- Dobrota
- Selekcija
- Križanje
- Mutacija
- Parametri
- Ostalo

Motivacija

- Prirodna evolucija
- Bolje jedinke preživljavaju i prenose svoje gene na potomke
- Slučajne promjene uzrokuju potencijalna poboljšanja u jedinkama
- Nakon puno generacija, dobivamo jedinke koje su dobro prilagođene svojoj okolini

Genetski algoritam

- Baziran na ideji prirodne evolucije
- Populacijski algoritam -> radi na skupu od više rješenja
- Kombinirati dobra rješenja radi dobivanja još boljih rješenja
- Uvoditi slučajne promjene u nadi da izbjegnemo lokalne optimume
- Ponavljaj određeni broj puta

Koraci GA

- Stvoriti početna rješenja (**jedinke**)
- Na neki način odabrati 2 rješenja (**selekcija**) i pomoću njih stvoriti dijete (**križanje**)
- Provesti nasumične promjene nad djetetom (**mutacija**)
- Evaluirati dijete (**funkcija dobrote**)
- Ubaciti dijete u populaciju umjesto neke druge jedinke
- Ponavljati do kriterija zaustavljanja

Primjer

- imamo n poslova koje moramo rasporediti na m dostupnih strojeva
- Svaki posao i ima ova svojstva:
 - Trajanje izvođenja na stroju j p_{ij}
 - Vrijeme željenog završetka d_i
 - Težina (važnost) posla w_i
- Kriterij, minimizirati težinsko kašnjenje svih poslova

$$Twt = \sum_i^n w_i * \max(C_i - d_i, 0)$$

- C_i predstavlja vrijeme završetka posla i

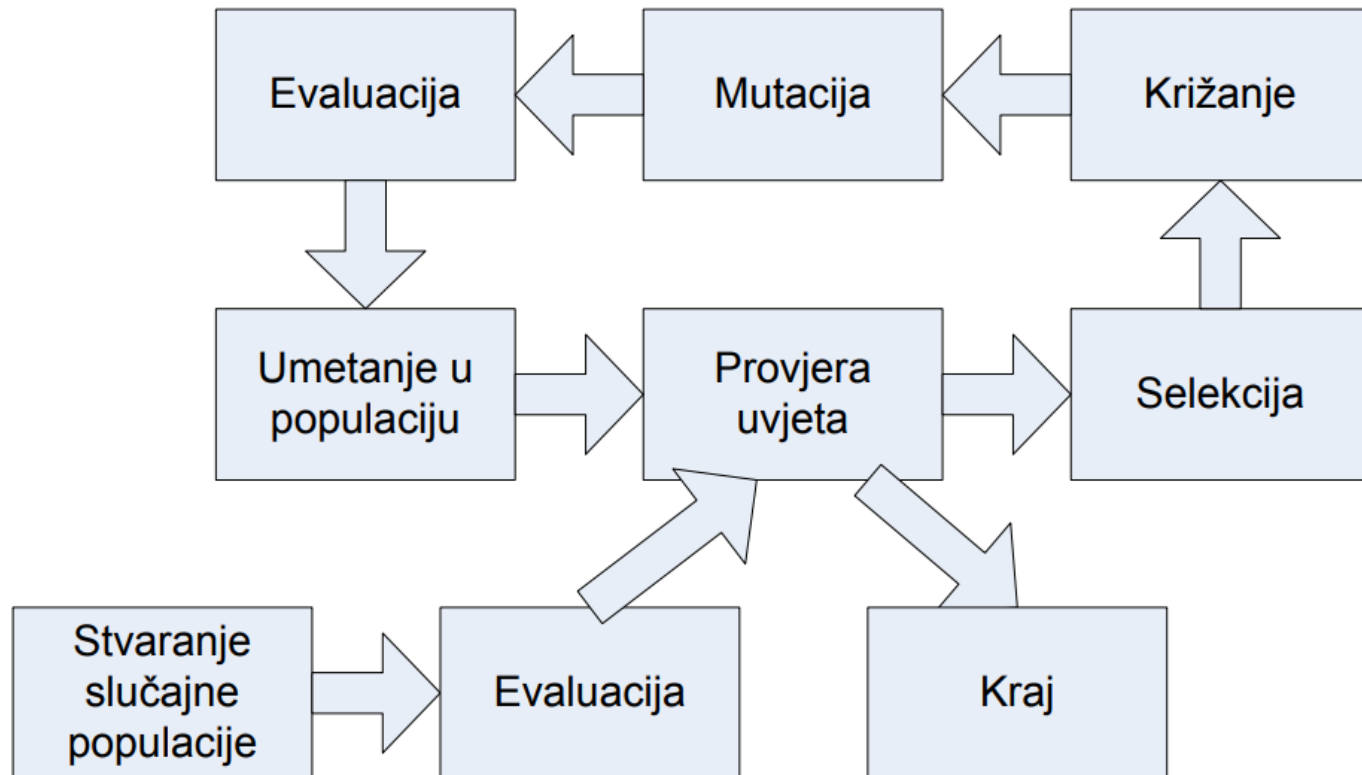
Vrste genetskog algoritma

- Eliminacijski
- Generacijski
- Druge varijante

Eliminacijski GA

- Stvara se samo jedna nova jedinka
- Na neki način odaberi dva roditelja (selekcija)
- Križaj oba roditelja i stvori novu jedinku *dijete*
- Mutiraj dijete s određenom vjerojatnošću
- Ubaci dijete u populaciju (izbacivanjem neke druge jedinke)
- Ponavljaj određeni broj iteracija

Eliminacijski GA



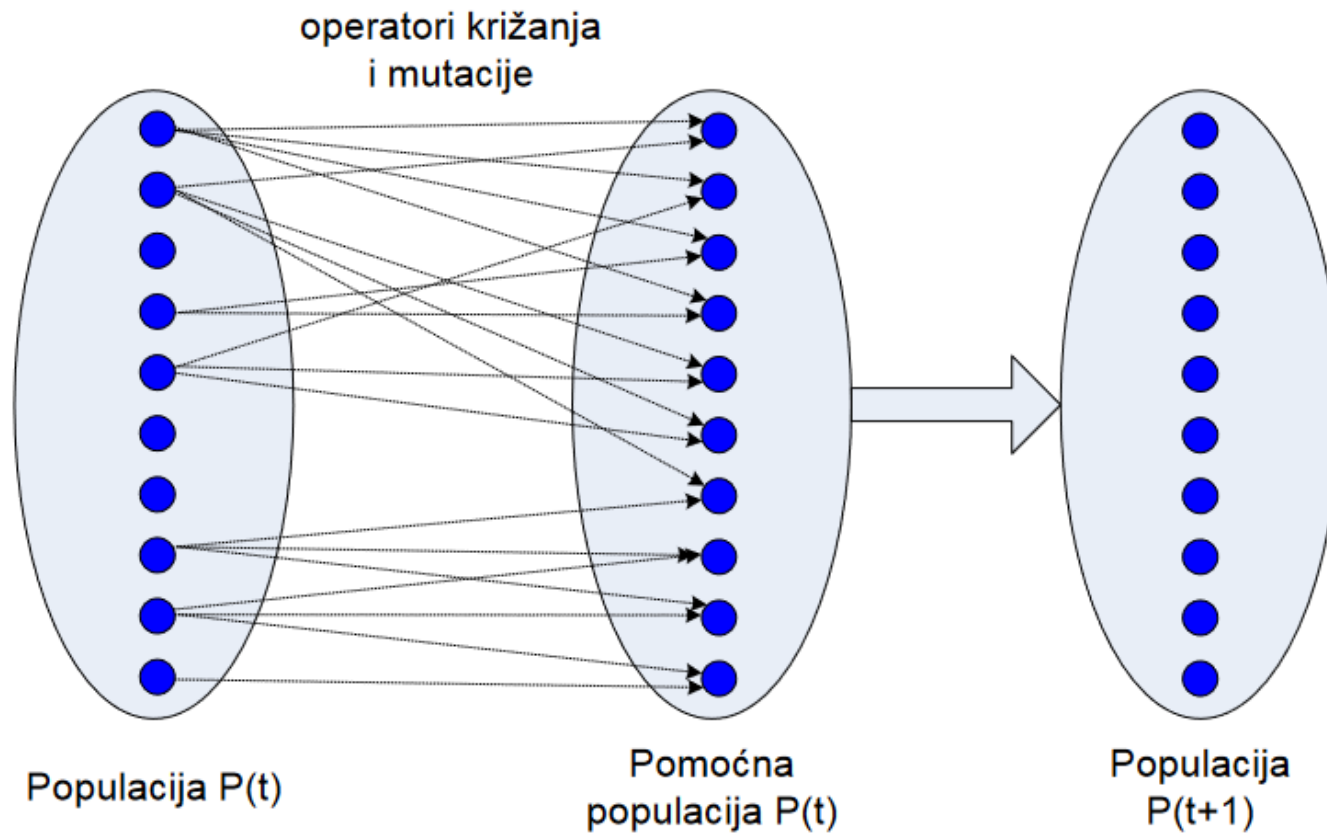
Eliminacijski GA

- Možemo definirati postotak eliminacije koji određuje koliko ćemo jedinki eliminirati iz populacije
- Odabir jedinki koje eliminiramo radimo s nekom od selekcija
- Nad preostalim jedinkama provodimo križanje i mutaciju i djecu umećemo u populaciju

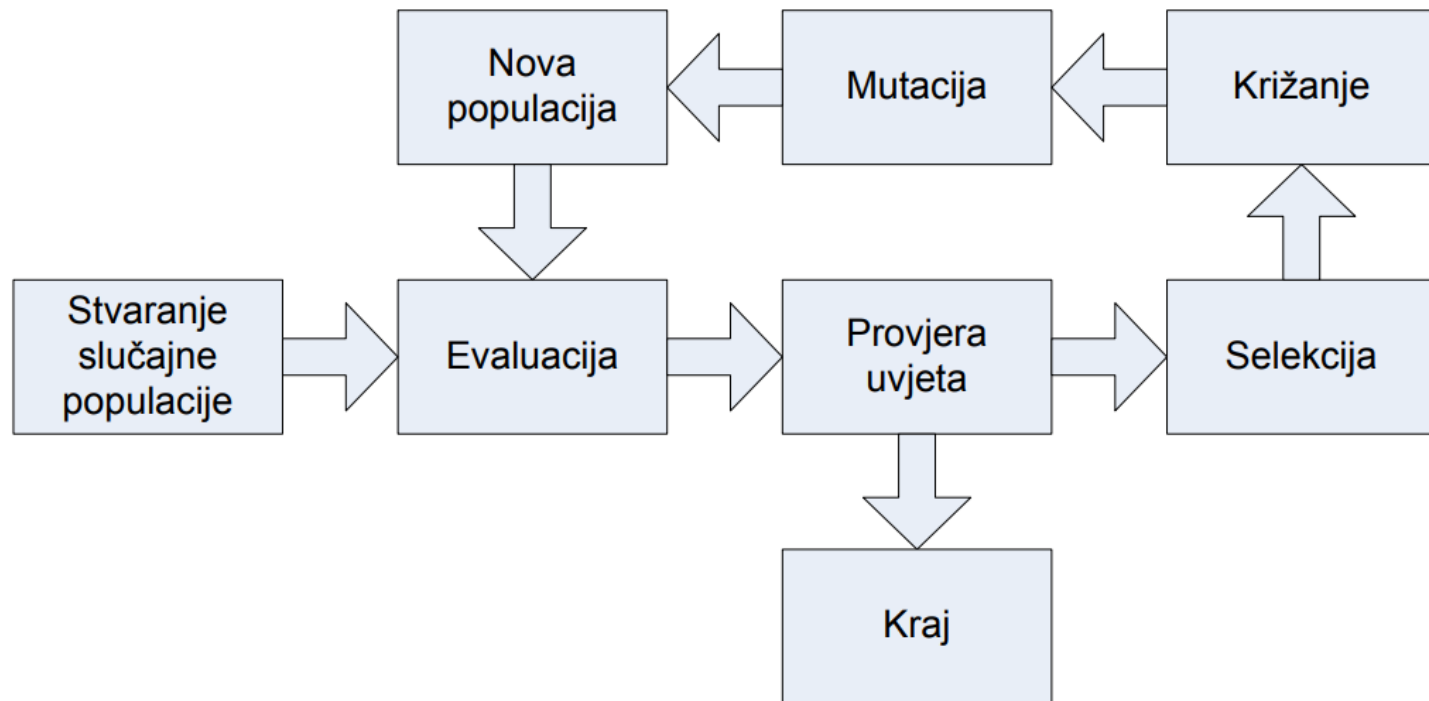
Generacijski GA

- Stvori novu populaciju jedinki (novu generaciju)
- Gradimo novu populaciju preko selekcije, križanja i mutacije
- Novu populaciju možemo graditi na razne načine:
 - Križaj dvije odabrane jedinke, mutiraj dijete i ubaci u populaciju
 - ubaci kopiju jedinke u novu generaciju
 - Mutiraj jedinku i ubaci ju u novu populaciju
- Kada izgradimo novu populaciju, brišemo staru i na njeno mjesto stavljamo novu

Generacijski GA



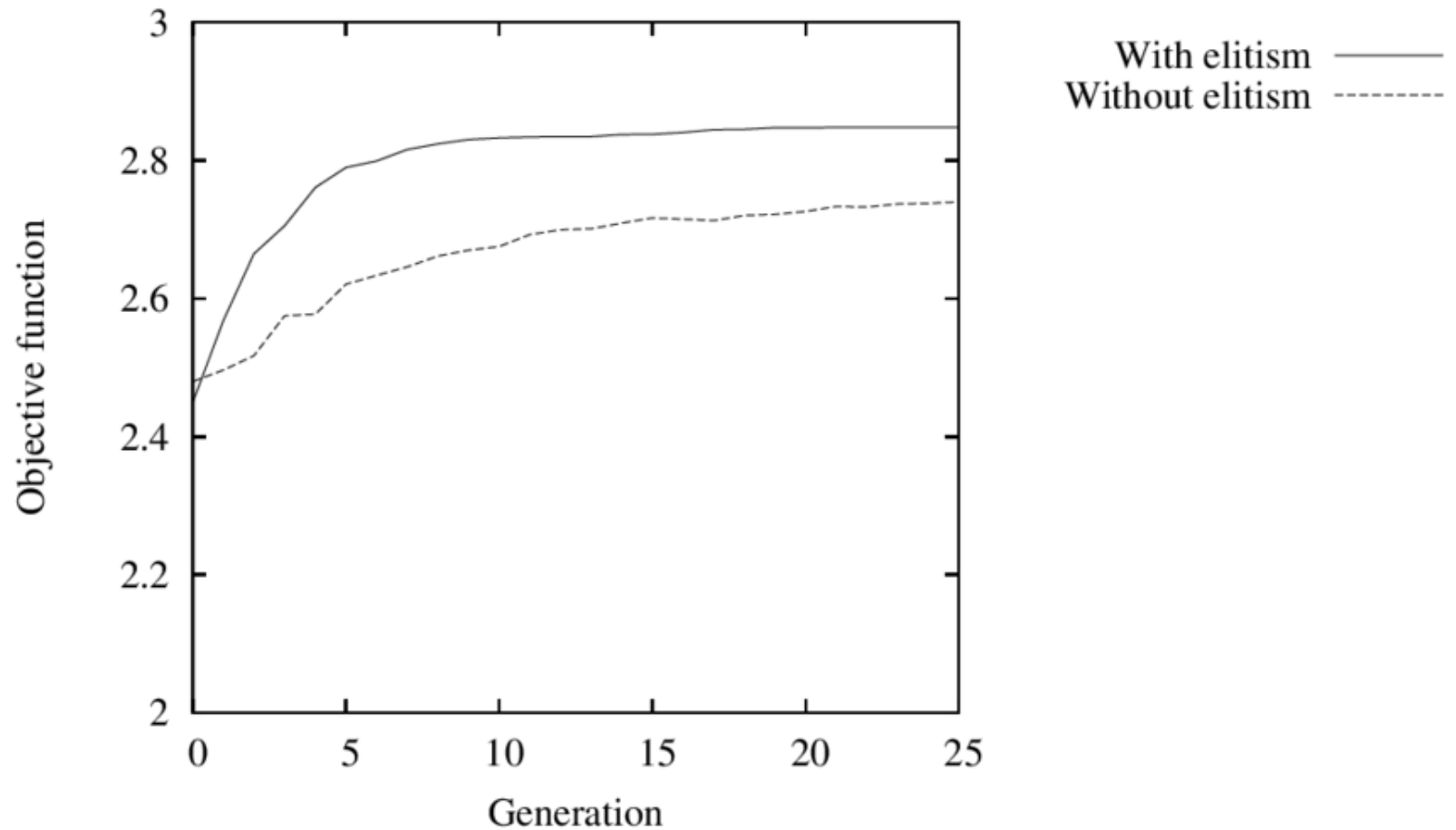
Generacijski GA



Elitizam

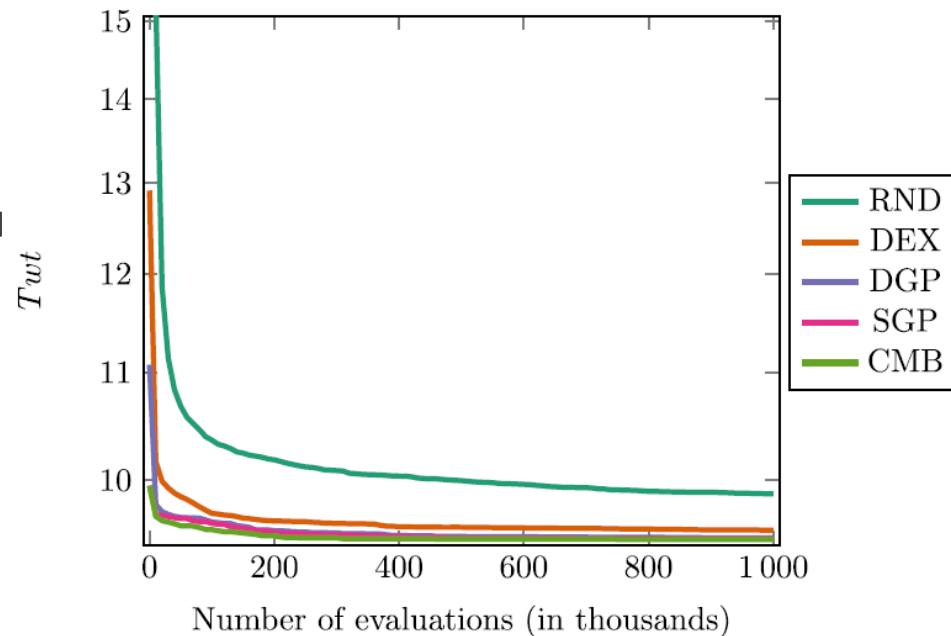
- Svojstvo da najbolja jedinka nikad ne bude eliminirane iz populacije
- Poželjno svojstvo
- Ako nema elitizma dobrota tijekom vremena oscilira
- U eliminacijskim varijantama inherentno uključen
- U generacijskim ga najčešće moramo sami ugraditi
- Može se definirati koliko najboljih jedinki želi sačuvati u populaciji

Elitizam



Stvaranje početne populacije

- Nasumično
 - Jednostavno, ali sporija konvergencija
- Heuristički
 - Koristimo neke heuristike za stvaranje rješenja
 - Moramo paziti da ne zapnemo u lokalnom optimumu
 - Kako osigurati raznolikost?



Prikazi rješenja

- Ovisan o konkretnom problemu:
 - Binarni prikaz
 - Realni
 - Permutacijski
 - Cjelobrojni
 - Miješani

Binarni prikaz

- Jedan od prvih prikaza
- Geni su 0 ili 1
- Jednostavan prikaz
- Danas se rijetko koristi
- Problem decepcije

1	1	0	1	1	1	0	0	1
---	---	---	---	---	---	---	---	---

Binarni prikaz

- Primjer deceptije
- $n=4$
- Minimum je u $x=8$
- Šta ako je većina rješenja >8 ?
- Algoritam teži prema 7 (binarno 0111)
- Najbolje rješenje je 8 (binarno 1000)
- Potrebno je promijeniti sva 4 bita kako bi došli do boljeg rješenja
- Rješenje: Greyev kod

Preslikavanje u bin. prikaz

- Bilo koji realni broj možemo prebaciti u binarni prikaz s određenom preciznošću
- Ako imamo raspon $[x_{min}, x_{max}]$ i željenu preciznost \tilde{x} , broj bitova za prikaz tih brojeva potrebno je $n \geq \frac{\log(\frac{x_{max}-x_{min}}{\tilde{x}}+1)}{\log(2)}$

- Realni broj možemo onda prebaciti u binarni prikaz kao

$$b = \frac{x - x_{min}}{x_{max} - x_{min}} * (2^n - 1)$$

- Binarni broj se pretvara u realni preko

$$x = x_{min} + \frac{b}{2^n} * (x_{max} - x_{min})$$

Binarni prikaz

- Problem diskretizacije prostora -> optimum možda ni ne možemo predstaviti u zadanom prikazu
- S brojem varijabli i većom preciznošću povećava se broj bitova
- Već za jednostavne problem algoritam može zapeti u suboptimalnim rješenjima
- Dobra stvar -> rješenja su uvijek u zadanim intervalima

Realni prikaz

- Najčešće korišten za kontinuirane probleme
- Općenito potrebno definirati gornju i donju granicu za varijable x_{min} i x_{max}
- Nakon križanja i mutacije potrebno je paziti da rješenje nije izašlo iz dozvoljenog intervala

2.14	-15.2	17.3	3.33	-27.6	17.5	19.2	32.2	-34.7
------	-------	------	------	-------	------	------	------	-------

Permutacijski prikaz

- Niz jedinstvenih brojeva od 1 do n
- Koristan za probleme u kojima se mora odrediti niz određenih stvari
- Npr. niz izvršavanja poslova ili obilazak čvorova u grafu

3	2	5	1	8	9	7	6	6
---	---	---	---	---	---	---	---	---

Cjelobrojni prikaz

- Rjeđe korišten
- Potrebno definirati gornju i donju granicu za varijable x_{min} i x_{max}
- Za razliku od permutacijskog vrijednosti se smiju ponavljati i ne moraju sve cjelobrojne vrijednosti biti prisutne u rješenju

2	5	7	3	7	7	9	3	4
---	---	---	---	---	---	---	---	---

Ostali prikazi

- Matrični
- Stablo (Genetsko programiranje)
- Permutacijski s delimiterima

Miješani prikaz

- Za složene probleme jedan prikaz možda nije dovoljan za predstaviti čitavo rješenje
- Npr. za predstavljanje rješenja potrebna je kombinacija permutacijskog i cjelobrojnog prikaza
- Može se koristiti i prikaz koji ne kodira cjelovito rješenje
- Primjerice, permutacijom se određuje samo redoslijed elemenata, a njihova dodjela u neke pretince se određuje u evaluaciji rješenja nekom heuristikom

Prikazi rješenja

- Fenotip – rješenje nekog problema
- Genotip – kako je rješenje prikazano u jedinci
- Jedan fenotip može se prikazati različitim genotipovima
- Za neke probleme fenotip i genotip mogu biti identični
- Često je teško razlučiti što je fenotip a što genotip

Funkcija dobrote

- Numerička vrijednost koja predstavlja kvalitetu rješenja
- Maksimizira se
- Funkcija kazne – analogno dobroti, ali ona se minimizira
- Kao funkciju dobrote/kazne često možemo koristiti vrijednost kriterija kojeg optimiramo
- Prilagodbe:
 - Relativna funkcija dobrote: $g_i = f_i - f_{min}$
 - *Windowing*: $F_i = a + (b - a) * \frac{f_i - f_w}{f_b - f_w}$, dobrotu preslikavamo u interval $[a, b]$

Selekcija

- Odabir boljih jedinki za reprodukciju i lošijih za eliminaciju
- Zašto ne bi uvijek križali najbolje i odbacivali najgore?
- Seleksijski pritisak
 - Može se definirati na razne načine
 - Općenito, omjer vjerojatnosti preživljavanja boljih i lošijih jedinki
 - Veliki seleksijski pritisak -> prebrza konvergencija i zapinjanje u lokalnim optimumima
 - Niski seleksijski pritisak -> spora konvergencija, pretraga je više nasumična

Proporcionalna selekcija

- *Roulette wheel selection*
- Vjerojatnost odabira jedinke proporcionalna je s njenom dobrotom
- Vjerojatnost odabira jedinke i $p_i = \frac{f_i}{\sum_{j=1}^n f_j}$
- Prednost: vjerojatnost odabira ovisi o dobroti jedinke
- Relativna proporcionalna selekcija – skaliramo sve dobrote

$$p_i = \frac{f_i - f_{\min}}{\sum_{j=1}^n (f_j - f_{\min})}$$

- Uvođenje selekcijskog pritiska:

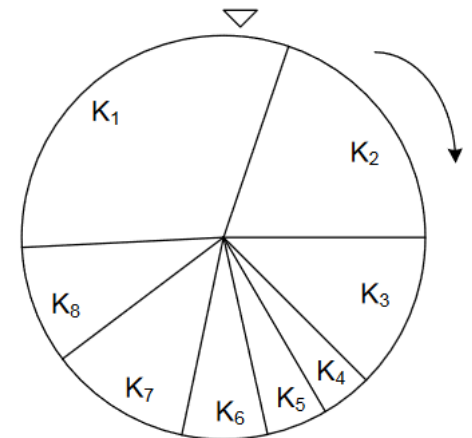
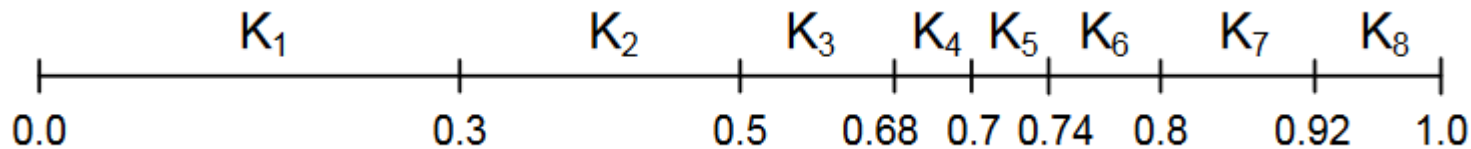
$$F_i = 1 + (SP - 1) * \frac{f_i - f_{\min}}{f_{\max} - f_{\min}}$$

Proporcionalna selekcija

- Nedostaci:
 - Problem skale – vjerojatnosti ovise o veličinama dobrote
 - Prevladavanje najbolje jedinke
 - Dobrota mora biti pozitivna
 - Velike populacije uzrokuju često slične vjerojatnosti
 - Kod promjena u populaciji moramo ponovo računati iznose vjerojatnosti

Proporcionalna selekcija

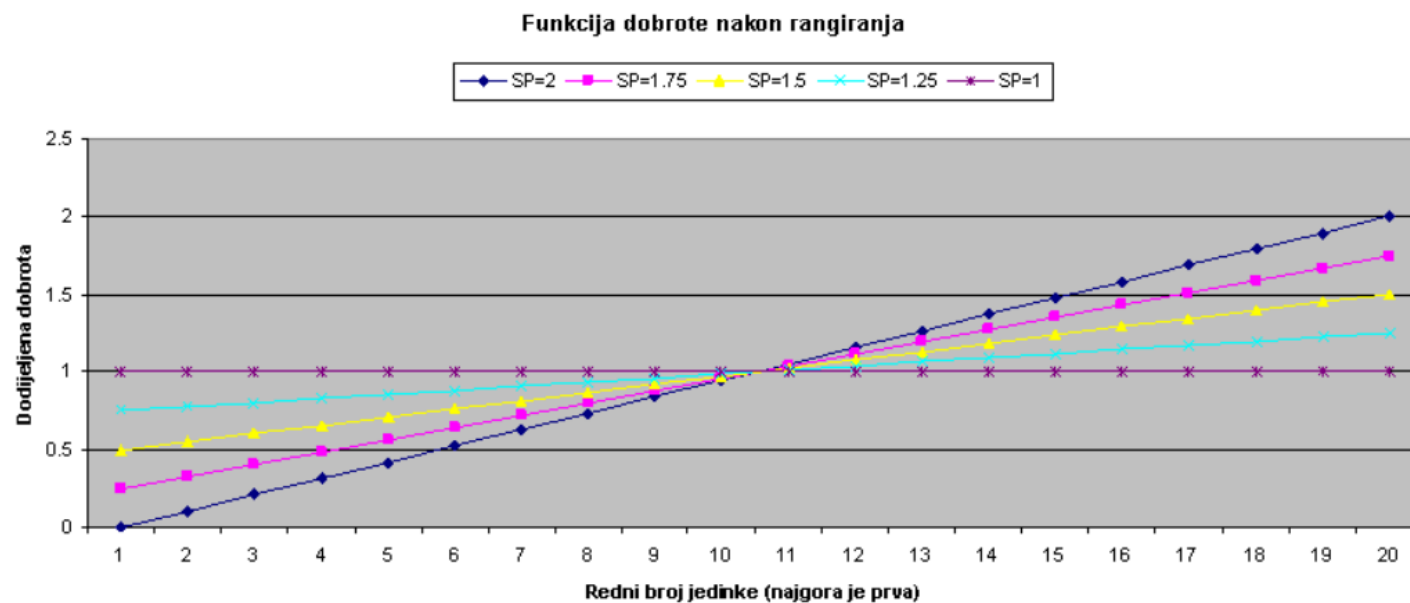
Jedinka	1	2	3	4	5	6	7	8
Dobrota jedinke	6	4	3.6	0.4	0.8	1.2	2.4	1.6
Vjerojatnost odabira	0.3	0.2	0.18	0.02	0.04	0.06	0.12	0.08



Selekcija linearnim rangiranjem

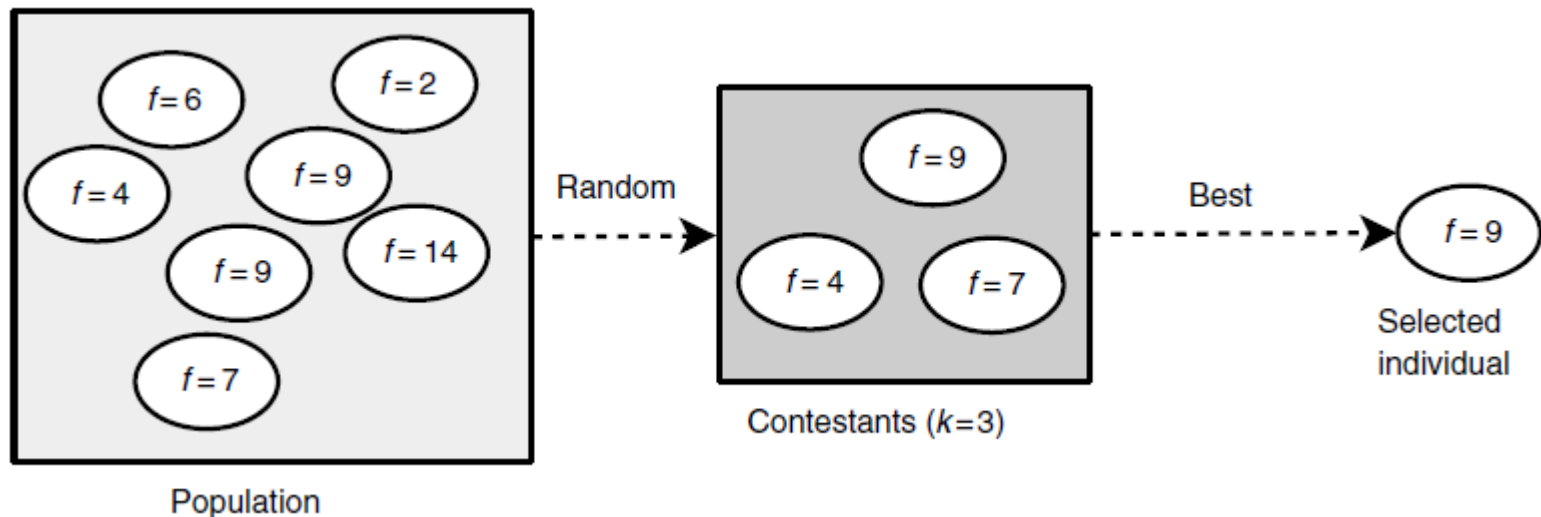
- Sortiramo sve jedinke po njihovoj dobroti
- Vjerojatnost je proporcionalna rangu
 - Rang 1 najlošije rješenje
 - Rang n najbolje rješenje
- $p_i = \frac{2-SP+2(SP-1)^{\frac{i-1}{n-1}}}{n}, SP \in [1,2]$
- SP definira selekcijski pritisak

Selekcija linearnim rangiranjem



Turnirska selekcija

- *k-tournament selection*
- Odabiremo nasumično k jedinki
- Od odabranih k jedinki odabiremo najbolju (ili najgoru)



Turnirska selekcija

- Parametar k određuje selekcijski pritisak u turnirskoj selekciji
- Mali parametar k -> SP je nizak, veća vjerojatnost odabira lošijih jedinki za preživaljanje
- Veliki parametar k -> SP je visok, manja vjerojatnost da lošije jedinke budu odabrane
- Šta bi se dogodilo kada bi parametar k bio jednak veličini populacije?

Jednostavna 3-turnirska selekcija

- Kod turnirske selekcije moramo više puta provoditi selekciju za odabir jedinki
- Ideja: provesti jednu selekciju za odabir roditelja i djeteta
- Odabrati 3 jedinke nasumično
- Dvije najbolje odabrati za roditelja
- Najlošija jedinka se eliminira i zamijeni novonastalom jedinkom

Križanje

- Kombiniranjem svojstva dviju ili više jedinki dobiti potencijalno bolje jedinke
- Svojstva od roditelja bi se trebala dobro preslikati u dijete
- Eksploatacija
- Ovisi o prikazu

Križanje – binarni prikaz

- Križanje jednom ili više točki prekida
- Odabiremo nasumično jednu poziciju u jedinkama



Uniformno križanje

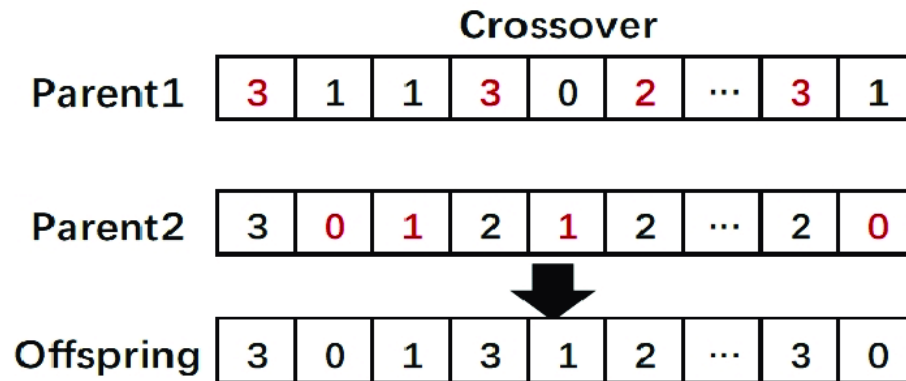
- Generiramo nasumični vektor R
- Ako je bit u oba roditelja jednak, prepisuje se u dijete, ako je različit na temelju R se odlučuje koja vrijednost se uzima.

Roditelj 1	1	1	0	1	1	1	0	0	1
Roditelj 2	0	0	1	1	0	1	1	0	0
R	1	0	0	0	0	1	1	0	0
Dijete	1	0	0	1	0	1	1	0	0



Križanje – realni prikaz

- Diskretno križanje – svaki gen nasumično odabiremo između oba roditelja



- Jednostavno križanje – isto kao i križanje s jednom točkom kod binarnog prikaza

Križanje – realni prikaz

- Aritmetičko križanje

$$h_i^1 = \lambda c_i^1 + (1 - \lambda) c_i^2$$

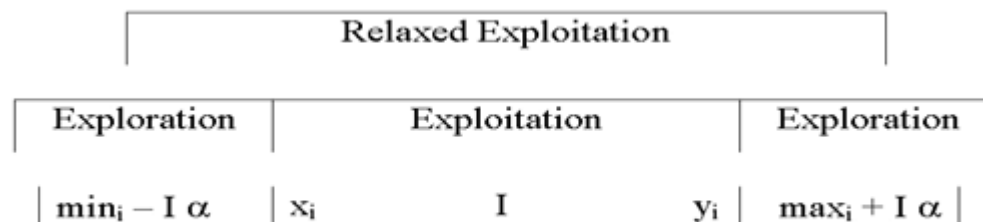
$$h_i^2 = \lambda c_i^2 + (1 - \lambda) c_i^1$$

- Što se dobije za $\lambda = 0.5$?
- BLX- α

$$c_{i,min} = \min(c_i^1, c_i^2), c_{i,max} = \max(c_i^1, c_i^2)$$

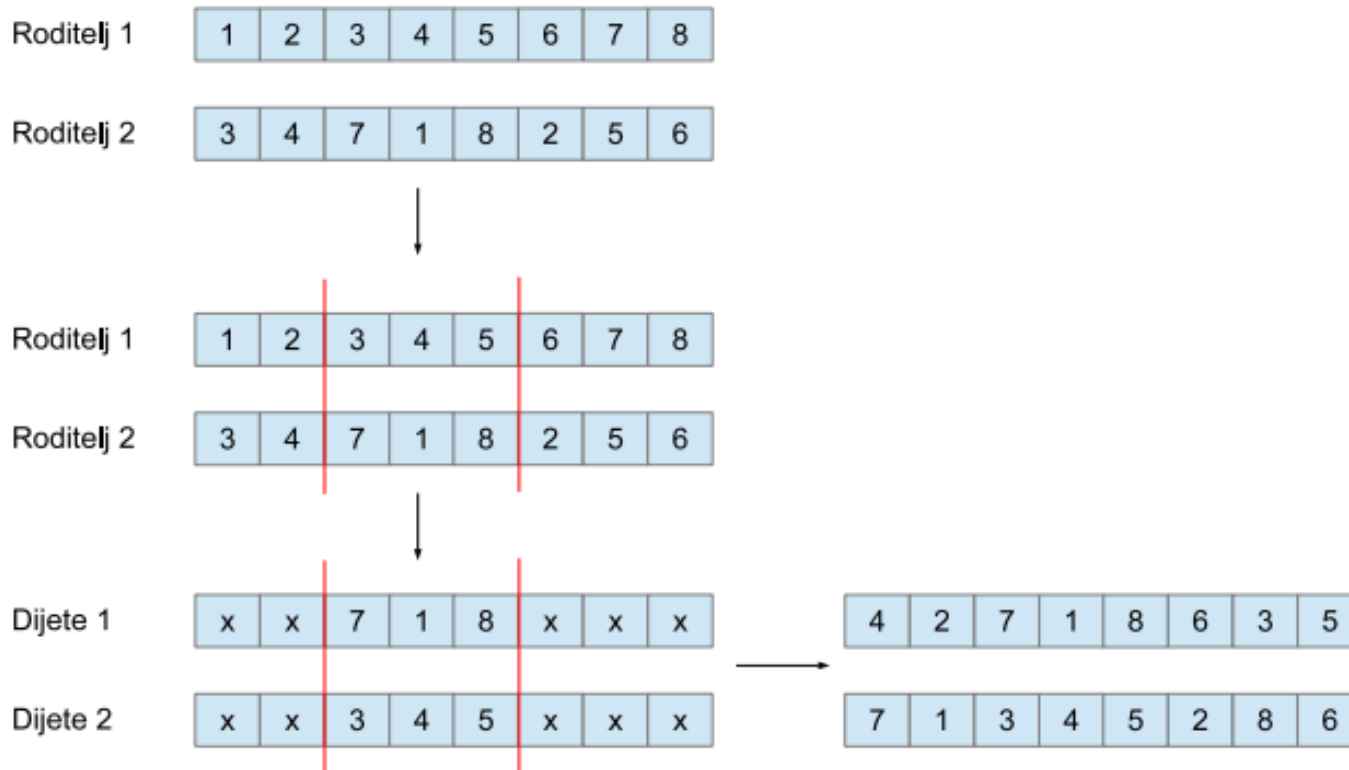
$$I_i = c_{i,max} - c_{i,min}$$

$$h_i \in [c_{i,min} - I_i * \alpha, c_{i,max} + I_i * \alpha]$$



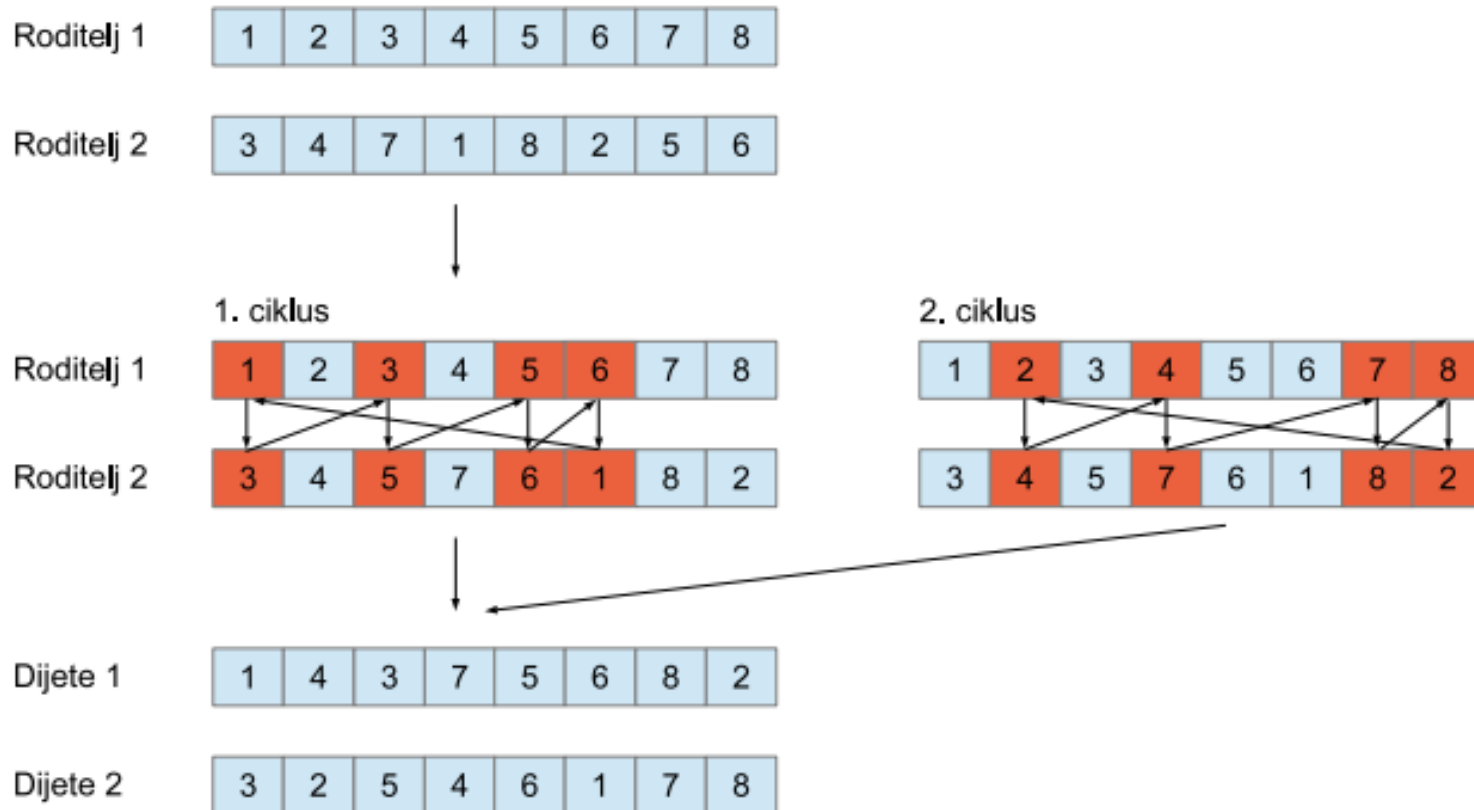
Križanje – permutacijski prikaz

- Djelomično preslikano križanje (PMX)



Križanje – permutacijski prikaz

- Križanje ciklusa



Križanje – permutacijski prikaz

- Križanje poretka

Roditelj 1

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

Roditelj 2

3	4	7	1	8	2	5	6
---	---	---	---	---	---	---	---



Roditelj 1

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

Roditelj 2

3	4	7	1	8	2	5	6
---	---	---	---	---	---	---	---



Dijete 1

x	x	3	4	5	x	x	x
---	---	---	---	---	---	---	---

Dijete 2

x	x	7	1	8	x	x	x
---	---	---	---	---	---	---	---



1	8	3	4	5	2	6	7
---	---	---	---	---	---	---	---

4	5	7	1	8	6	2	3
---	---	---	---	---	---	---	---

Mutacija

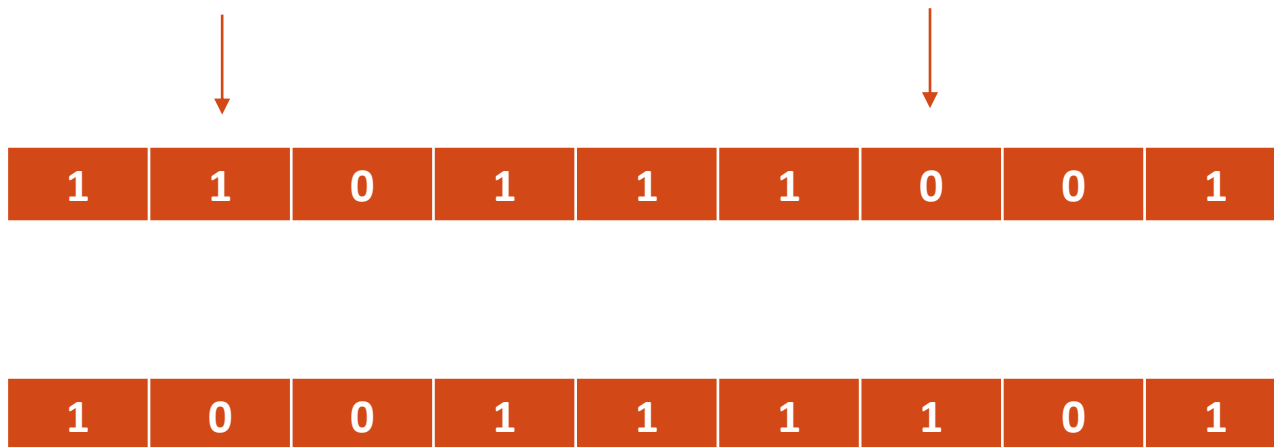
- Uvesti nasumične promjene u rješenja
- Ciljevi:
 - Izbjeći lokalne optimume
 - Vratiti izgubljene gene u populaciji
 - Dobiti možda bolja rješenja
- Ne provodimo ju uvijek, već s određenom vjerojatnošću mutacije
 - Mala vjerojatnost mutacije -> lako zapnemo u lokalnim optimumima
 - Velika vjerojatnost mutacije -> nasumična pretraga

Željena svojstva mutacije

- Korištenjem operatora mutacije trebali bi moći dobiti bilo koje rješenje
- Rješenja bi trebala biti valjana (nije uvijek moguće za probleme s ograničenjima)
- Mutacijom bi trebali dobiti rješenja koja su blizu trenutnog rješenja

Mutacija – binarni prikaz

- Zamjena bitova
- Svaki bit mutiramo s određenom vjerojatnošću
- Čemu su jednake vjerojatnosti?
 - Svaki bit ima istu vjerojatnost
 - Bitovi imaju različite vjerojatnosti

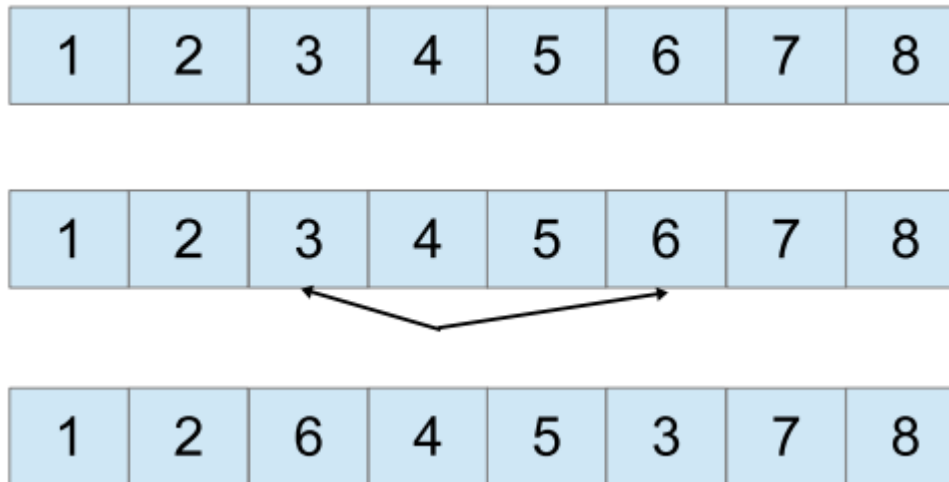


Mutacija – realni prikaz

- Uniformna mutacija – odabiremo vrijednost uniformno iz zadanog intervala
 - Zamjena cijele vrijednosti: $x = r, r \in [x_{min}, x_{max}]$
 - Dodavanje neke uniformne vrijednosti: $x = x * (b - a) + a$
generiramo rješenje iz intervala $[a, b]$
- Gaussolika mutacija – dodajemo neki Gaussov šum na jednu ili više varijabli
 - $x = x + r, r \in \mathcal{N}(0, s)$, s predstavlja standardnu devijaciju
 - Koliki je raspon rješenja?
- Kako odrediti željene parametre?

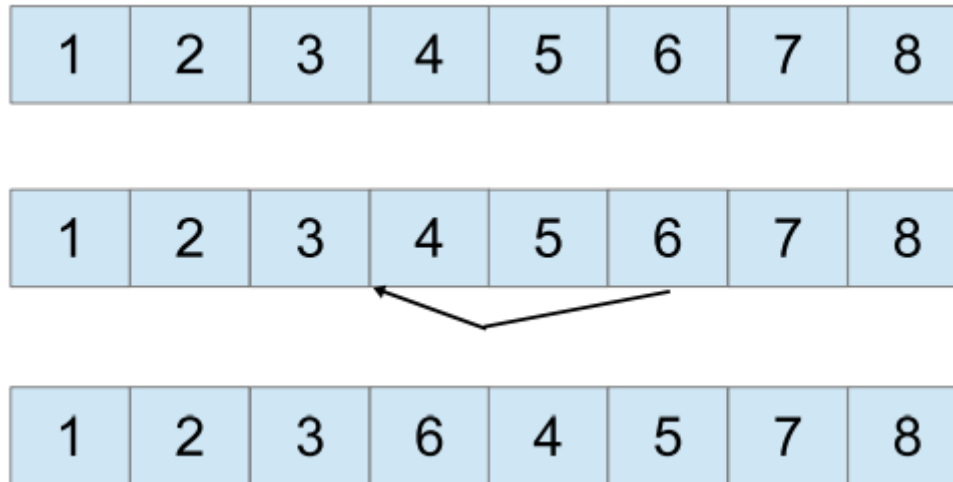
Mutacija – permutacijski prikaz

- Mutacija zamjenom



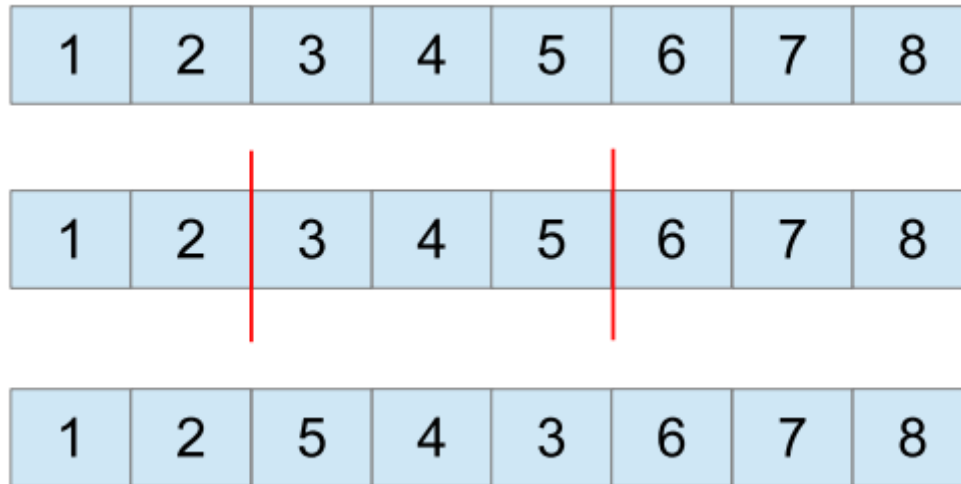
Mutacija – permutacijski prikaz

- Mutacija umetanjem



Mutacija – permutacijski prikaz

- Mutacija inverzijom



Genetski operatori

- Koje genetske operatore odabrati?
- Možemo koristiti jedan ili više operatora
- Ako koristimo više operatora:
 - Svi mogu imati jednaku vjerojatnost primjene
 - Vrijednost se može adaptivno mijenjati ovisno o uspješnosti pojedinih operatora

Kriteriji zaustavljanja

- Broj iteracija/generacija
- Broj evaluacija funkcije cilja
- Stagnacija
- Vremensko ograničenje
- Postignuto dovoljno dobro rješenje

Parametri GA

- Veličina populacije
- Vjerojatnost mutacije
- Kriterij zaustavljanja
- Parametri specifični za prikaze/križanja/mutacije i slično
- Koje vrijednosti odabrati?
- Parametri uvelike ovise jedni o drugima:
 - Vjerojatnost mutacije i veličina populacije
 - Veličina populacije i broj iteracija/evaluacija

GA i ograničenja

- Ne postoji jednostavno rješenje
- Ograničiti GA da radi samo s ispravnim rješenjima
 - Potrebno prilagoditi sve dijelove GA
 - Ne garantira dobar uspjeh, prostor može biti dosta razlomljen pa ga nije moguće raditi dobro pretražiti
- Uvesti kaznu u funkciju dobrote koja usmjerava pretragu u bolja područja
 - $F = f(x) - \lambda P(x)$
 - Kakav mora biti odnos između dodane kazne i funkcije dobrote
 - Iznos bi trebao biti adaptivan
 - Nemamo garanciju da ćemo dobiti ispravno rješenje

Memetički algoritam

- Cilj: poboljšati efikasnost genetskih algoritama
- Ugraditi operatore lokalne pretrage u GA
- U određenim fazama genetskog algoritma primijeniti postupke lokalne pretrage na neka rješenja u populaciji
- Lokalnu pretragu možemo upotrijebiti nakon
 - Mutacije nekog rješenja
 - Određenog broja iteracija nad nasumičnim/najboljim rješenjima u populaciji

Memetički algoritam

Inicijaliziraj populaciju P

Evaluiraj populaciju

Ponavljaj

- Odaberi roditelje i križaj ih

- Mutiraj dijete

- Evaluiraj dijete

- Poboljšaj dijete primjenom lokalne pretrage**

- Ubaci dijete u populaciju

Dok kriterij zaustavljanja nije zadovoljen

Memetički algoritam

Inicijaliziraj populaciju P

Evaluiraj populaciju

Ponavljaj

- Odaberi roditelje i križaj ih

- Mutiraj dijete

- Evaluiraj dijete

- Ubaci dijete u populaciju

- Ako zadovoljen kriterij primjene lokalne pretrage**

 - Provedi lokalnu pretragu nad odabranim jedinkama**

Dok kriterij zaustavljanja nije zadovoljen

Zaključak

- Jedna od najstarijih i najpopularnijih metaheuristika
- Osnovna varijanta radi dobro za veliki broj problema
- Za bolje rezultate potrebno je dobro prilagoditi algoritam problemu
- Često se hibridizira s drugim metodama: simulirano kaljenje, lokalne pretrage i slično
- Primjenjiv na širok spektar problema

Genetsko programiranje

Marko Đurasević

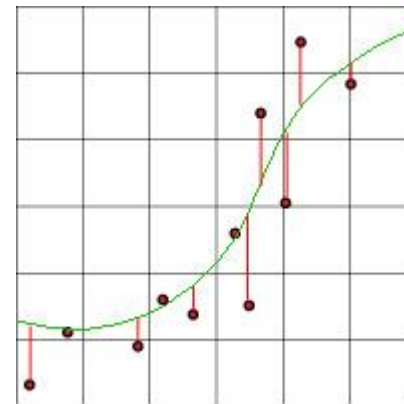
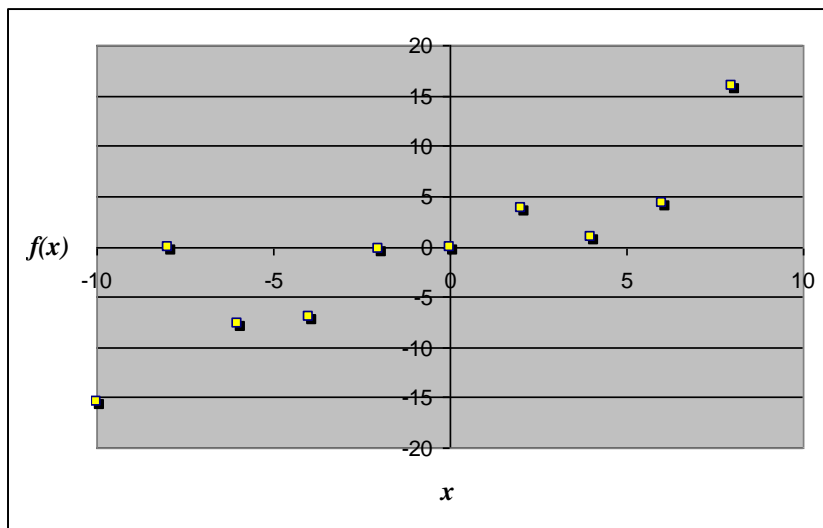
marko.durasevic@fer.hr

Motivacija

- Problem pronalaženja odgovarajućih koeficijenata (regresija):
 - $a * x_1^2 * \sin(b * x_2) + \ln(c * x_1 + d * x_2)$
- Što ako oblik funkcije nije poznat?
 - Simbolička regresija – tražimo oblik funkcije

Motivacija - simbolička regresija

- zadatak: otkriti *simbolički* oblik modela
 - nemamo pretpostavki (predznanja) o nepoznatoj funkciji



Motivacija

- Problemi klasifikacije – izrada klasifikatora
- IZRADA UPRAVLJAČA (programa)
- Pravila za trgovinu dionicama (kriptovalutama)

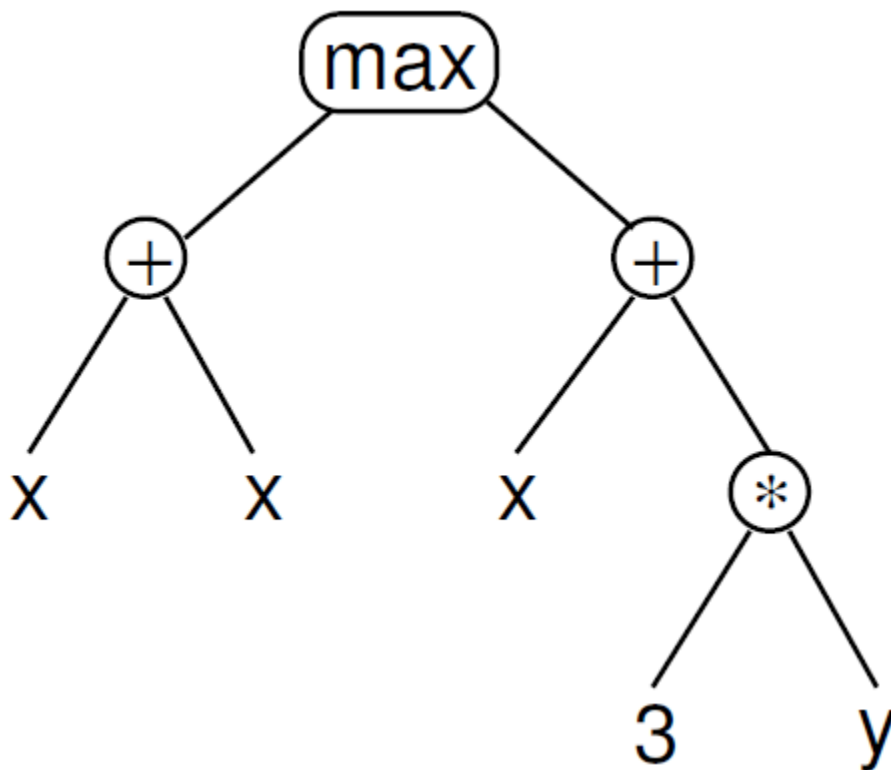
Osnove genetskog programiranja

Genetsko programiranje

- Genetski algoritam s drugačijim prikazom rješenja
- Jedinke predstavljaju matematičke izraze ili programe
 - Najčešće zapisane u obliku stabla
- Prilagođeni operatori mutacije i križanja
- Parametri: veličina populacije, vjerojatnost operatora, kriterij zaustavljanja...

Prikaz rješenja

$$\max(x + x, x + 3 * y)$$



Prikaz rješenja

- Odabrati skup primitiva (operatora i operandi)
- Operatori (unutarnji čvorovi)
 - Aritmetički (+, -, *, /, sin, cos, log, exp, pow, sqrt), logički (AND, OR, NOT), uvjetni (IF, IFGTE), petlje
- Operandi (terminali ili vanjski čvorovi)
 - Ulazne varijable (x, y), konstante (0, 1, 3.14), funkcije bez argumenata (rand)

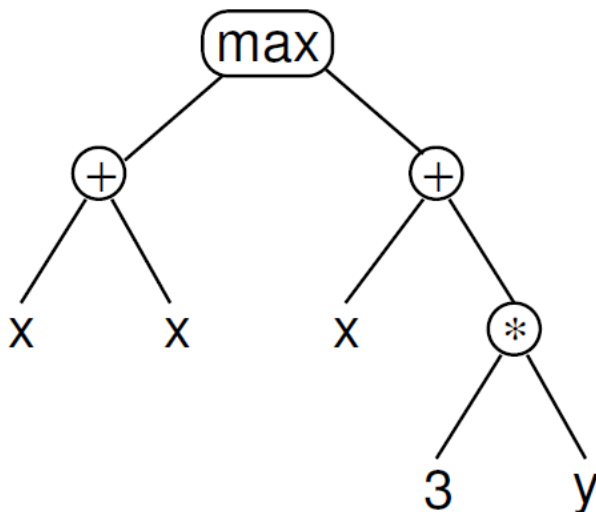
Prikaz rješenja

- Potrebna svojstva skupa primitiva:
 - Potpunost (*sufficiency*) – skupom moguće riješiti problem
 - Zatvorenost (*closure*) – definirane sve kombinacije operacija-operand
- Ograničiti veličinu rješenja:
 - Maksimalna dubina stabla
 - Maksimalni broj čvorova u stablu

Interpretacija rješenja

- Ovisno o vrsti problema, stablo se može interpretirati na različite načine

Simbolička regresija - računanje izlaza za pojedine vrijednosti ulaznih varijabli



Upravljanje agentom - prolazak kroz stablo i izvođenje čvorova redom

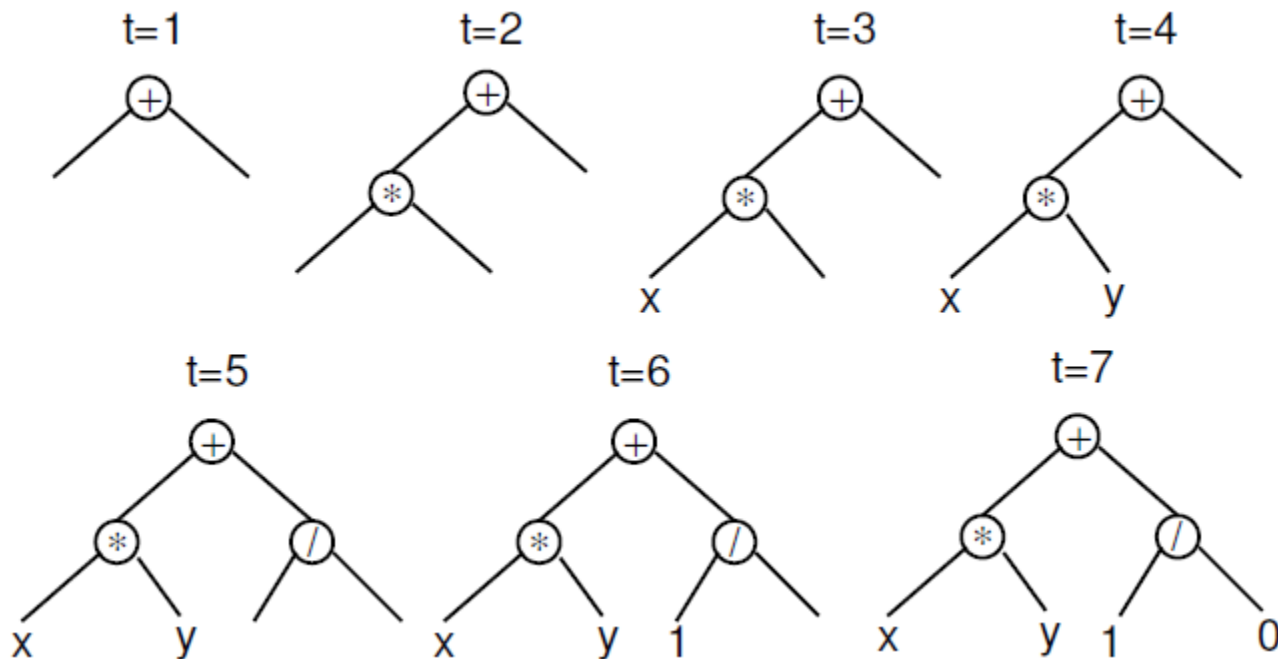


Inicijalizacija početne populacije

- Obično se zada maksimalna početna dubina
- Metode izgradnje jedinki:
 - *Full* – sva stabla su potpuna s maksimalnom dubinom
 - *Grow* – ne moraju svi terminali biti na maksimalnoj dubini
 - *Ramped half-and-half* – najčešće korišten
 - 50% full, 50% grow uz različite maksimalne dubine

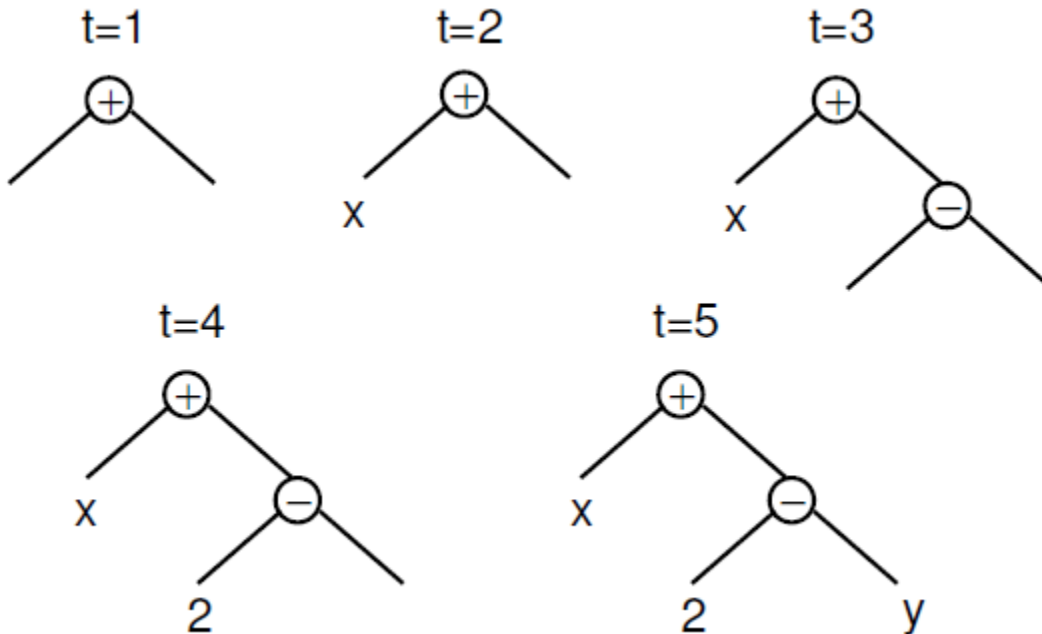
Inicijalizacija početne populacije

- Full metoda – stablo maksimalne dubine 2



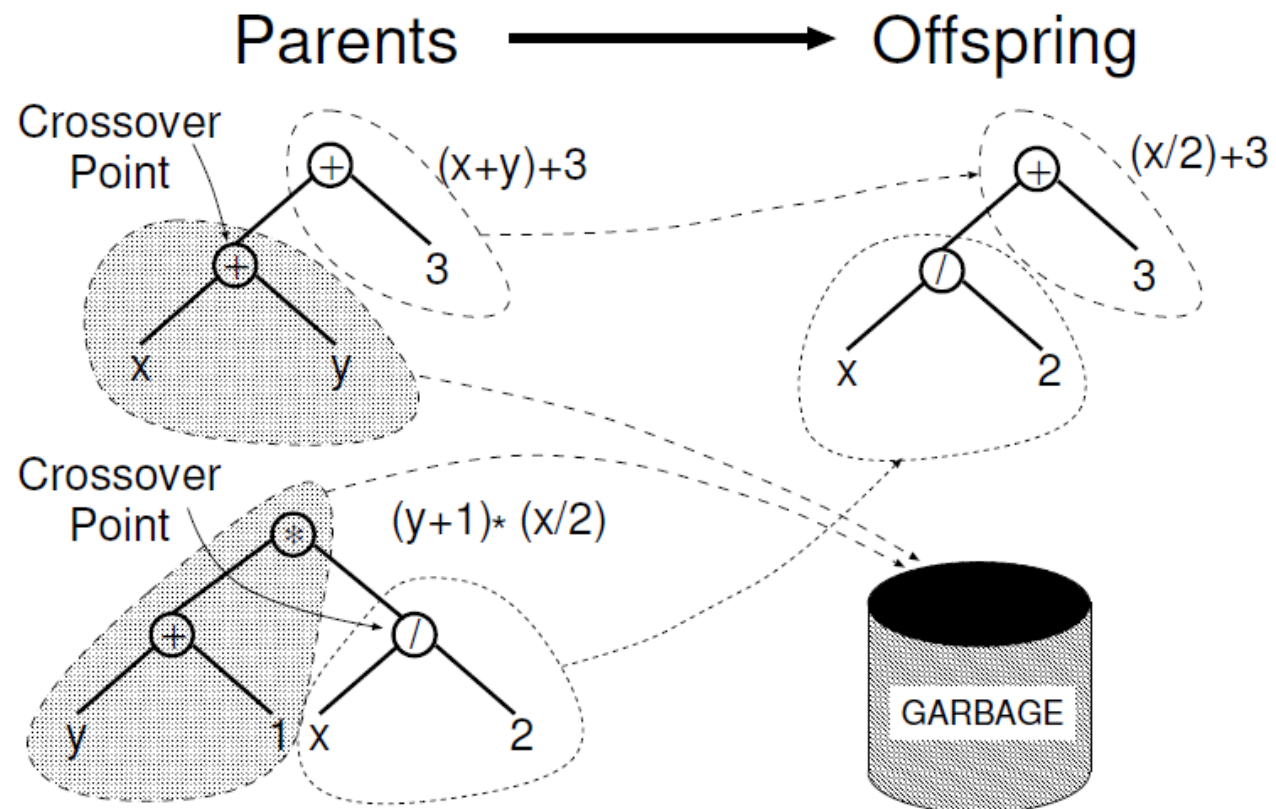
Inicijalizacija početne populacije

- Grow metoda – stablo maksimalne dubine 2



Križanje

- Subtree crossover

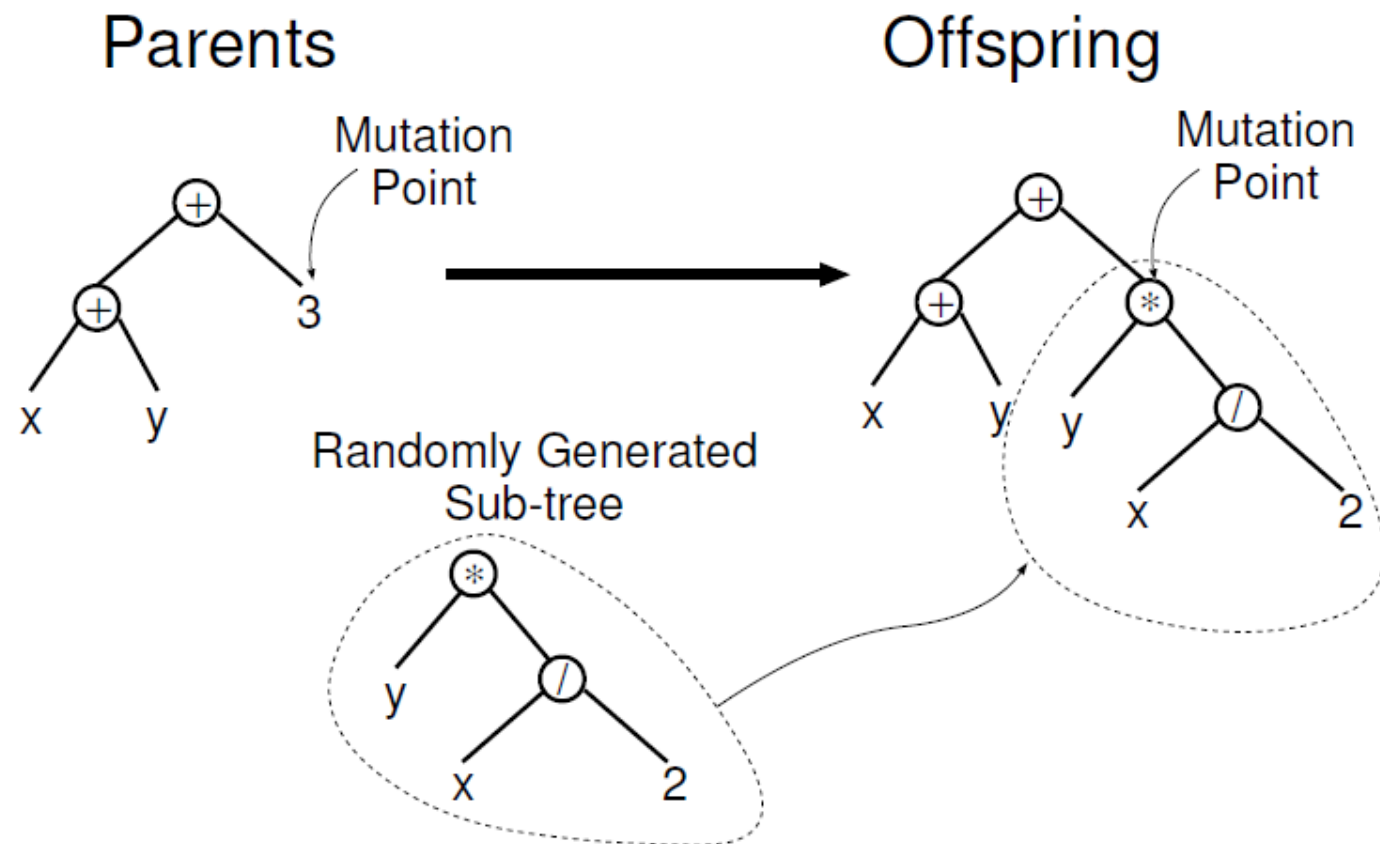


Križanje

- One-point crossover
- Uniform crossover
- Context-preserving crossover
- Size-fair crossover

Mutacija

- Subtree mutation



Mutacija

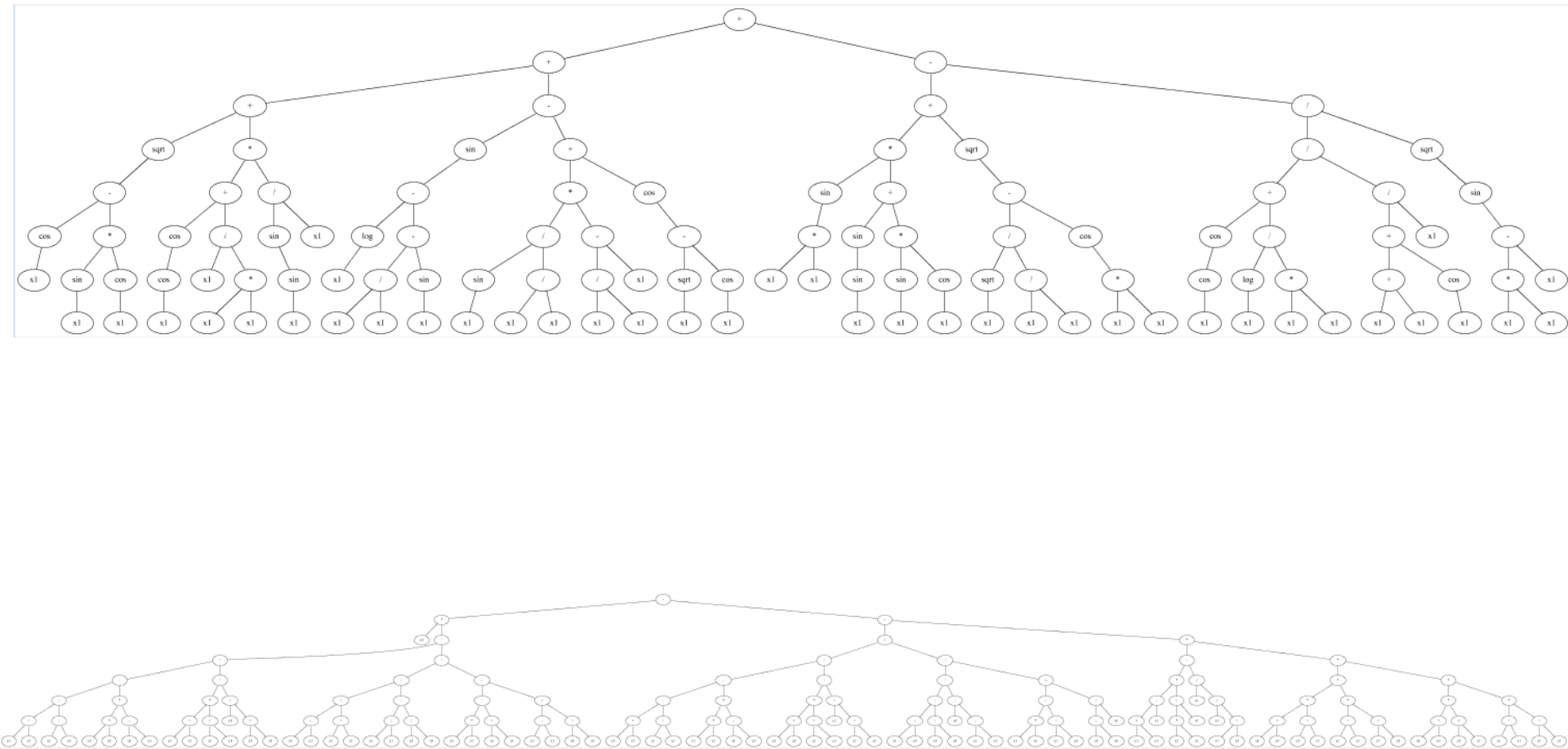
- Subtree mutation
- Size-fair subtree mutation
- Node replacement mutation
- Hoist mutation
- Shrink mutation
- Permutation mutation
- Mutating constants at random
- Mutating constants systematically

Napredni koncepti

Bloat

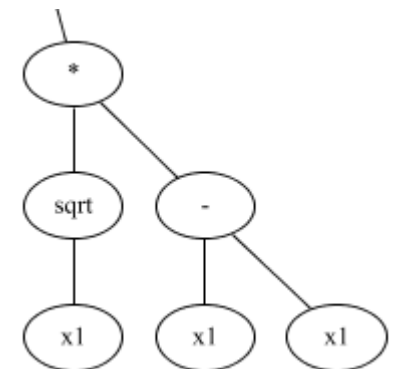
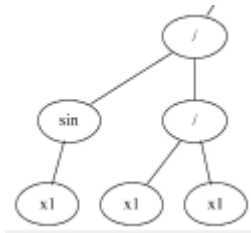
- povećanje stabala tijekom evolucije
- ponašanje za gotovo svaki problem
- puno filozofije o
 - uzrocima (fitness, introni, broj ispitnih primjera...)
 - mjerama (parsimony pressure – više cijenimo manje jedinice)
 - izbjegavanju (posebni operatori, prilagođena selekcija, višekriterijska optimizacija)
- pitanje: koja je veza veličine stabla i sposobnosti generalizacije? (*Occam's razor*)

Bloat



Bloat

- Metode sprječavanja ili popravljjanja:
 - *Parsimony pressure*
 - Mnogokriterijska optimizacija – jedan kriterij veličina stabla
 - Posebni operatori križanja i mutacije
 - Editing – uklanjanje nebitnih dijelova u stablu



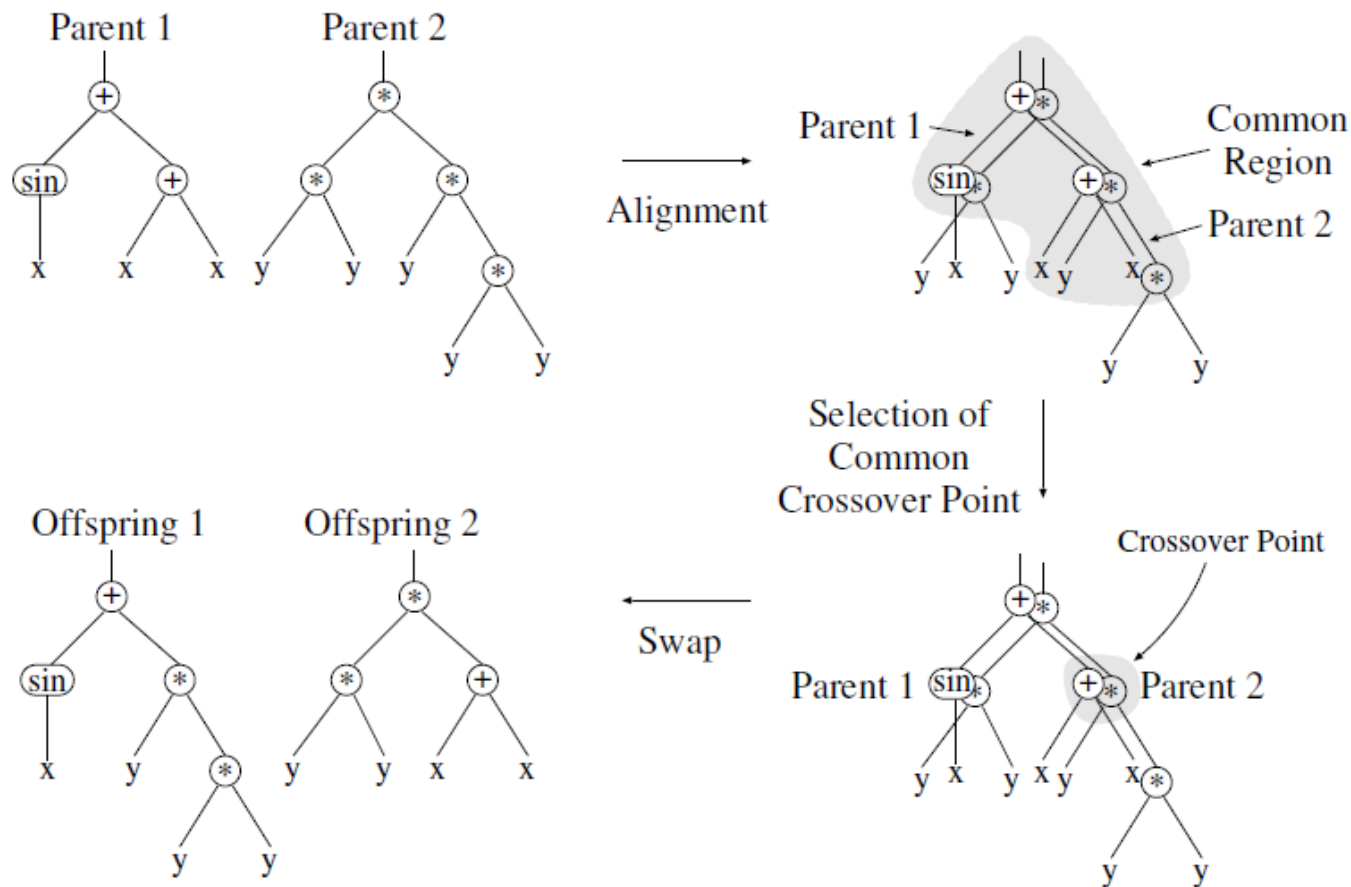
Ephemereal random constants

- Tražimo $f(x) = 0.5 * x^2 + 1.3$
- Slučajna početna vrijednost (prilikom inicijalizacije populacije ili prilikom mutacije)
- Kasnije nepromjenjive, osim za prilagođenu mutaciju (npr. Gaussova)
- Korisno u simboličkoj regresiji!

Linearno skaliranje

- Problem u simboličkoj regresiji: funkcija ima dobar oblik, ali je pomaknuta u odnosu na podatke
- Tražimo $f(x) = 0.5 * x^2 + 1.3$
- Teško pogoditi konstante
- pojednostavimo kao: $f'(x) = a * f(x) + b$
- konstante **a** i **b** tražimo minimizacijom srednje kvadratne pogreške (izvan GP-a)
 - tada GP treba samo skužiti (x^2)

Homologna križanja



Tipovi u GP-u

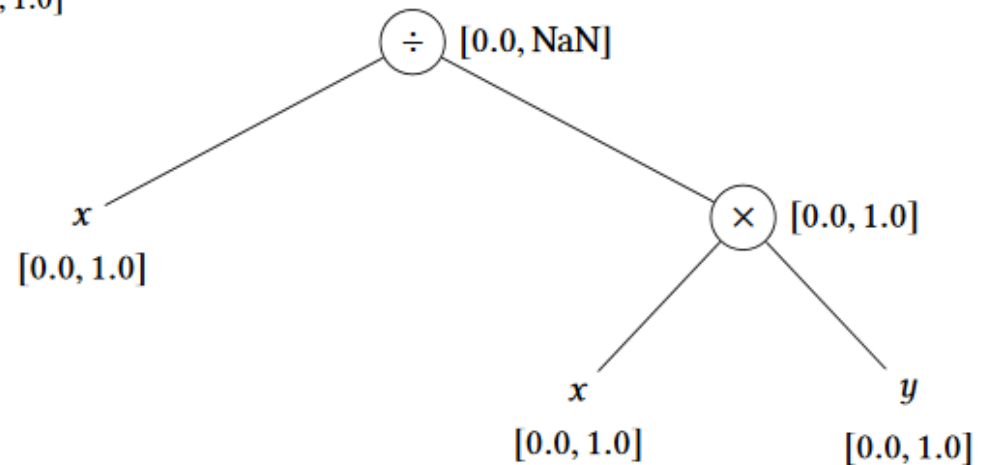
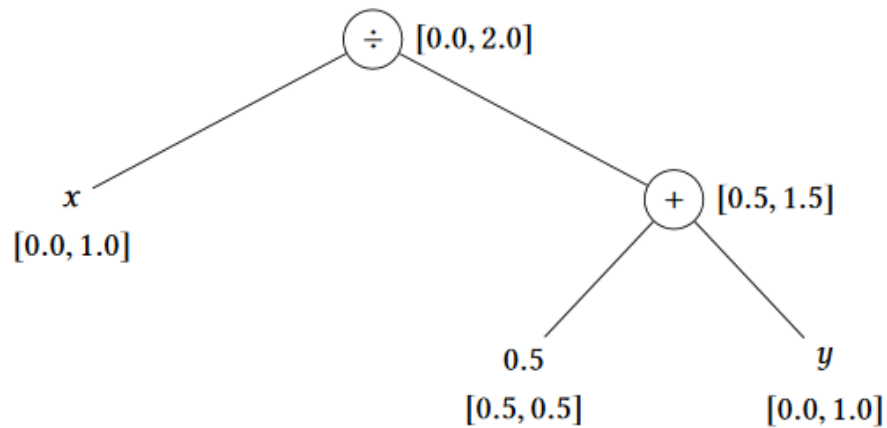
STGP: Strongly typed GP

- podijeliti terminale i funkcije po tipu (double, int, bool, vektor, matrica...)
- dozvoliti samo sintaksno ispravne programe!
- većina GP aplikacija ipak 'typeless'
- srodna ideja: semantička (a ne sintaksna) ispravnost!
- DAGP: *dimensionally aware GP*
 - npr: ako je t vrijeme, ne zbrajati t i t^2 !
 - studentski rad: <http://bib.irb.hr/prikazi-rad?&rad=408164>

Intervalna aritmetika

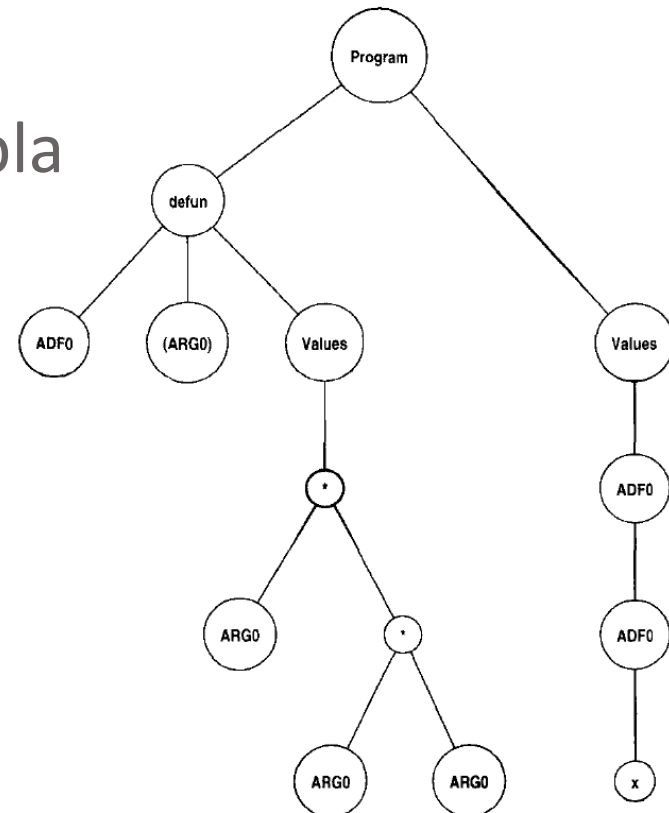
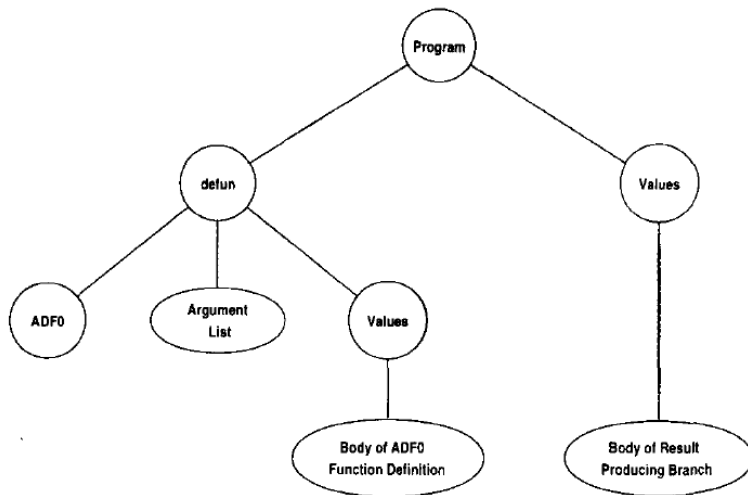
- uobičajeno: 'zaštićeno' dijeljenje ($x \% y$)
 - $\text{division} = \text{fabs}(\text{second}) > \text{MIN} ? \text{first} / \text{second} : 1.;$
 - $\text{division} = \text{fabs}(\text{second}) > \text{MIN} ? \text{first} / \text{second} : \text{first};$
- bolje: provjera mogućeg raspona vrijednosti svakog čvora (podstabla)
 - samo za simboličku regresiju
 - potrebno poznavati skup ispitnih primjera
 - podstabla nedefinirane vrijednosti se ne evaluiraju

Intervalna aritmetika

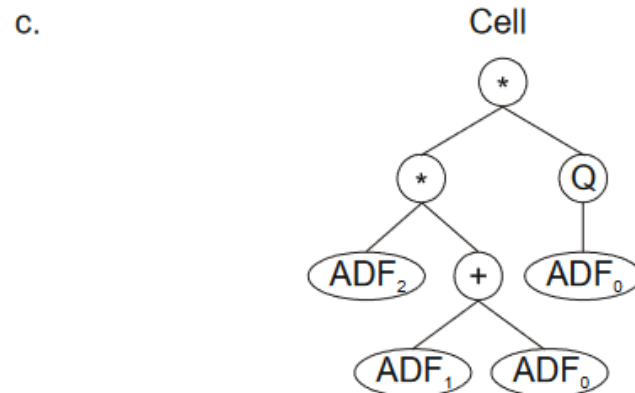
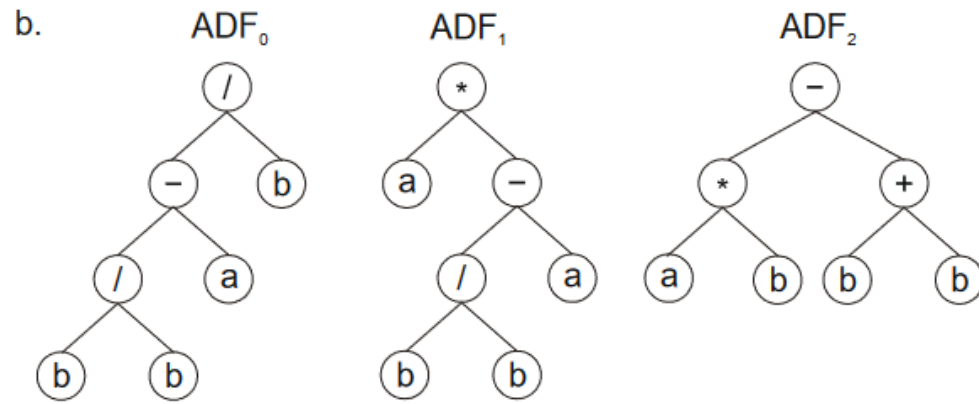


Automatski definirane funkcije

- Potprogrami implementirani kao dodatna stabla
- Posebni skup čvorova
- Poziva se kao funkcija glavnog stabla



Automatski definirane funkcije



Prikazi u genetskom programiranju

Drugi oblici prikaza rješenja

- CGP – Cartesian GP
 - studentski rad: <http://bib.irb.hr/prikazi-rad?&rad=519001>
- GEP – gene expression programming
 - niz znakova
 - glava niza: funkcije i terminali (zadajemo duljinu)
 - rep niza: do maksimalne duljine (ovisno o broju i n-arnosti funkcija)
 - križanje, mutacija, *transpozicija*
 - studentski rad: <http://bib.irb.hr/prikazi-rad?&rad=518985>
- grammatical evolution

CGP

- Čvorovi raspoređeni rešetku dimenzija $n \times m$ (proizvoljan broj stupaca i redaka)
- Može biti više izlaza
- Čvorovi međusobno povezani
- Neki čvorovi se ne moraju uzimati u obzir pri računanju izlaza
- Rješenje zapisano kao niz cijelih brojeva
 - Brojevi određuju ulaze u pojedini čvor i operaciju koju čvor predstavlja

CGP

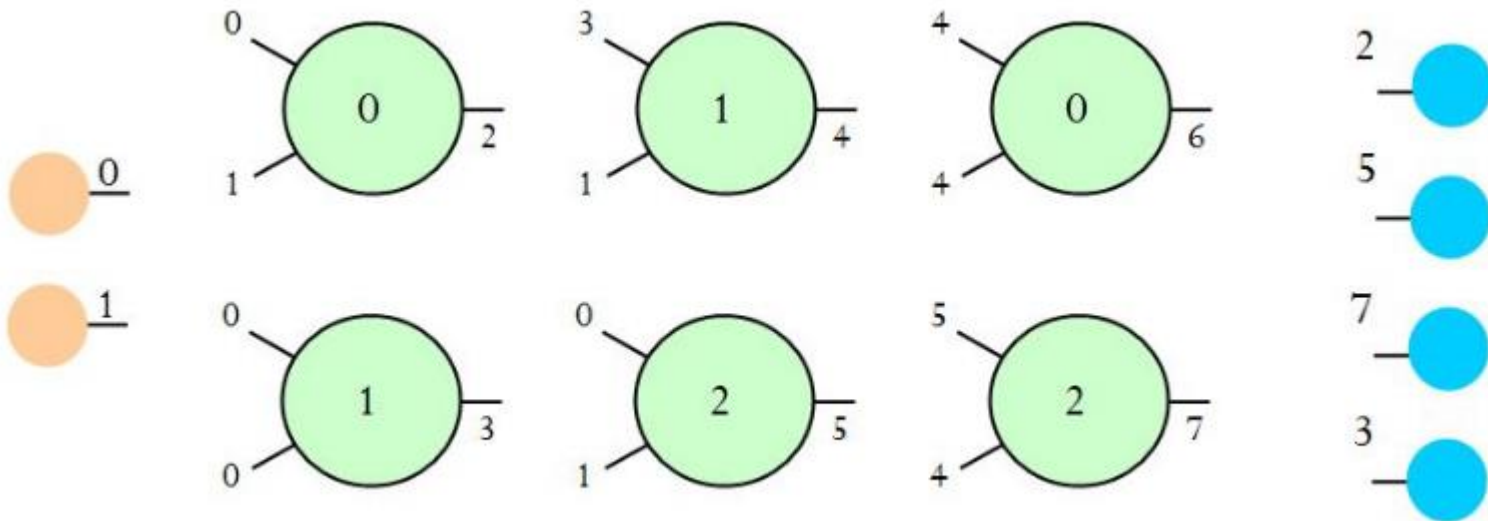
Genotip

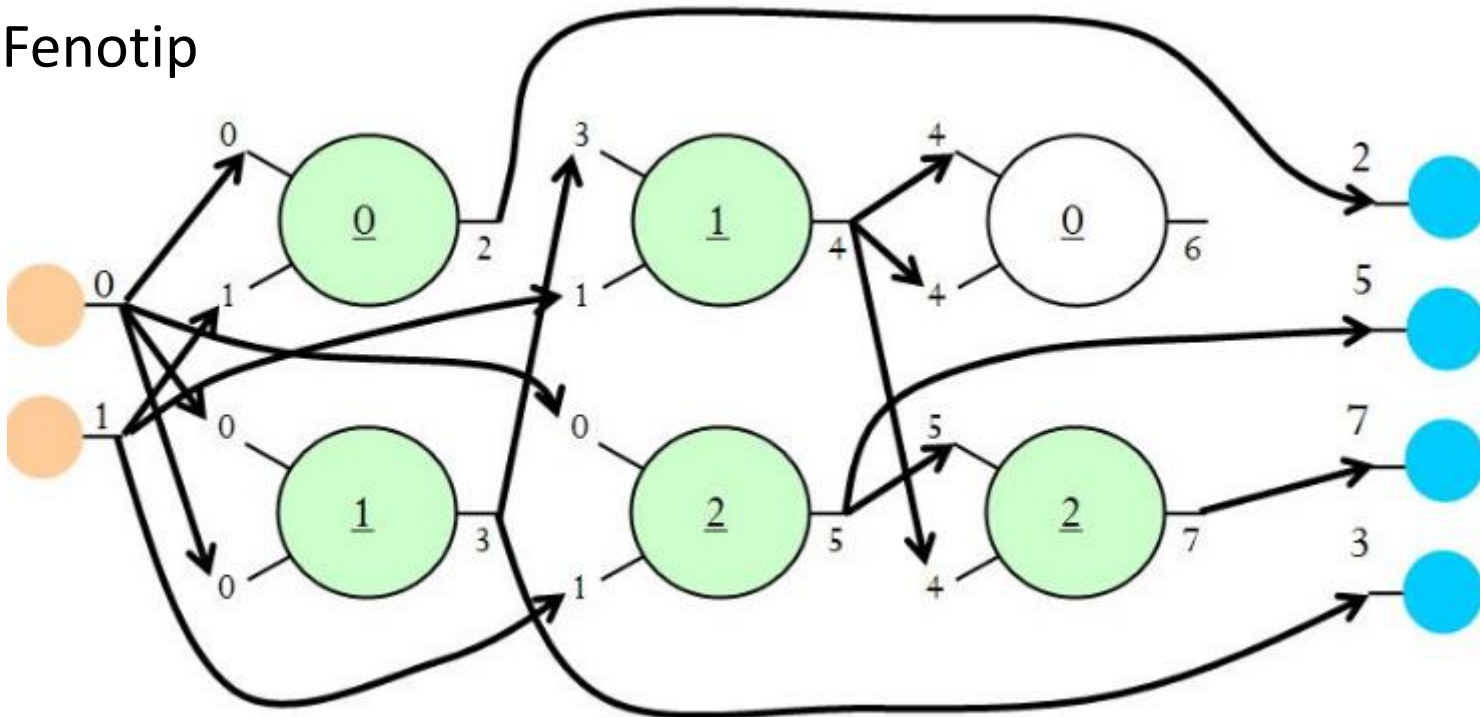
0 0 1 1 0 0 1 3 1 2 0 1 0 4 4 2 5 4

2 5 7 3

Indeks	Operator
0	+
1	-
2	*

Fenotip





CGP - operatori

- Mutacija simbola
- Križanje
 - Jednom točkom prekida
 - Često se uopće ne koristi
- Koriste se male populacije i $1+\lambda$

CGP – primjena

- Razvoj sklopova
- Evolucija neuronskih mreža
- Sve što se je rješavalo GP-om

GEP

- Čvorovi zapisani u obliku niza
- Niz se sastoji od više dijelova koje zovemo geni
- Svaki gen se sastoji od glave i repa
 - U glavi se nalaze bilo koji čvorovi
 - U repu se nalaze samo terminalni čvorovi
- Iz niza se gradi stablo koje predstavlja izraz

GEP

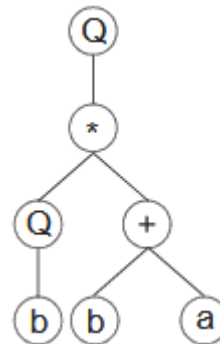
- Veličinu glave definira korisnik: h
- Veličina repa računa se kao: $t = h * (n - 1) + 1$
 - n maksimalni broj argumenata svih funkcijskih čvorova
- Osigurano da se ne može izgraditi neispravan izraz

GEP

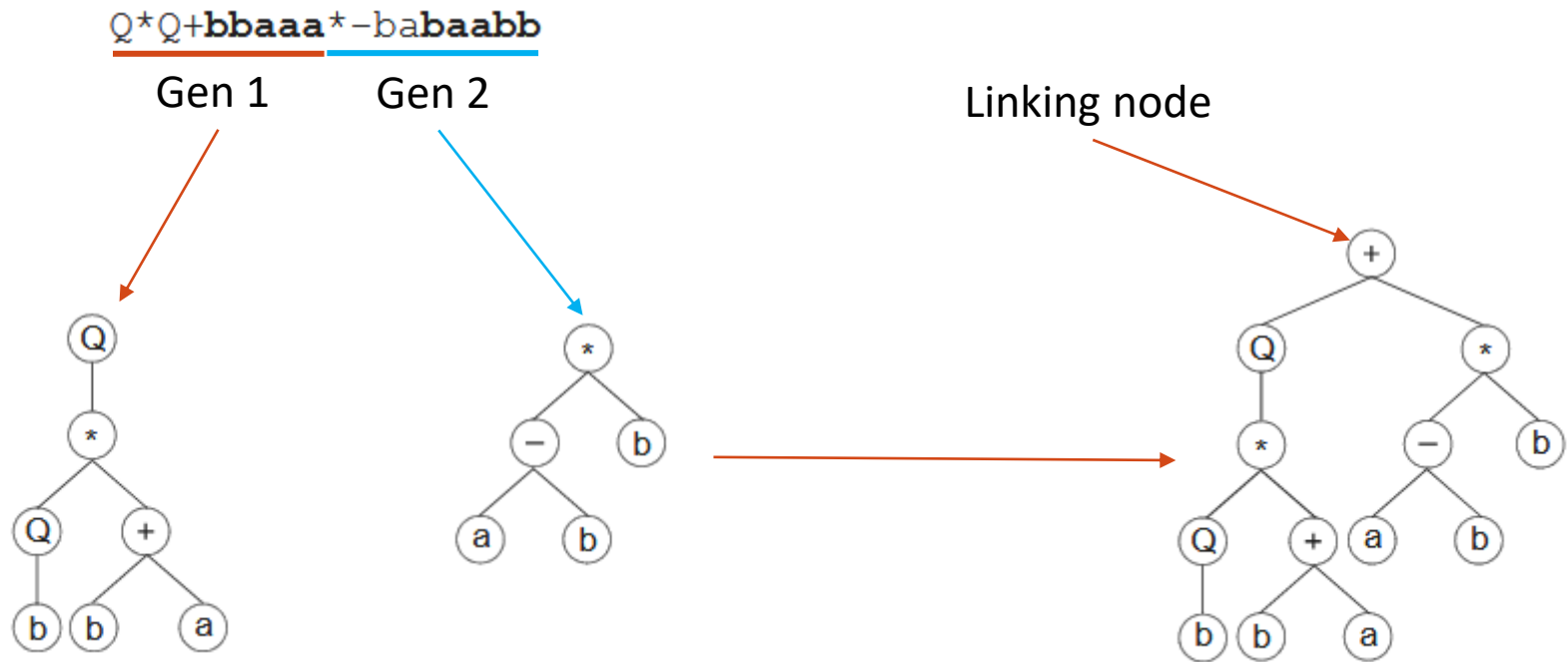
Q*Q+bbaaa

Glava gena

Rep gena



GEP



GEP - operatori

- Mutacija – promjena jednog ili više elemenata u nizu
- Križanje:
 - Jednom točkom prekida
 - Dvije točke prekida
 - Križanje gena
- Transpozicija
 - IS
 - RIS
 - Genska transpozicija

GEP - transpozicije

- IS

Prije transpozicije

+ - pt w MR age w | * PAT w w dd SL pt | w + / pt SL **dd age**
 gen 1 gen 2 gen 3

Nakon transpozicije

+ **dd age** w MR age w | * PAT w w dd SL pt | w + / pt SL dd age
 gen 1 gen 2 gen 3

GEP - transpozicije

- RIS

Prije transpozicije

+ - pt w MR age w | * PAT w w dd SL pt | w + / pt SL dd age
 gen 1 gen 2 gen 3

Nakon transpozicije

+ - pt w MR age w | * PAT w w dd SL pt | - pt w pt SL dd age
 gen 1 gen 2 gen 3

Grammatical evolution

- Gramatika koja definira program ili izraz koji se treba generirati
 - Završni i nezavršni znakovi, produkcijska pravila, početni nezavršni znak
- Prikaz u obliku niza cijelih brojeva
 - Brojevi definiraju koja se produkcijska pravila koriste

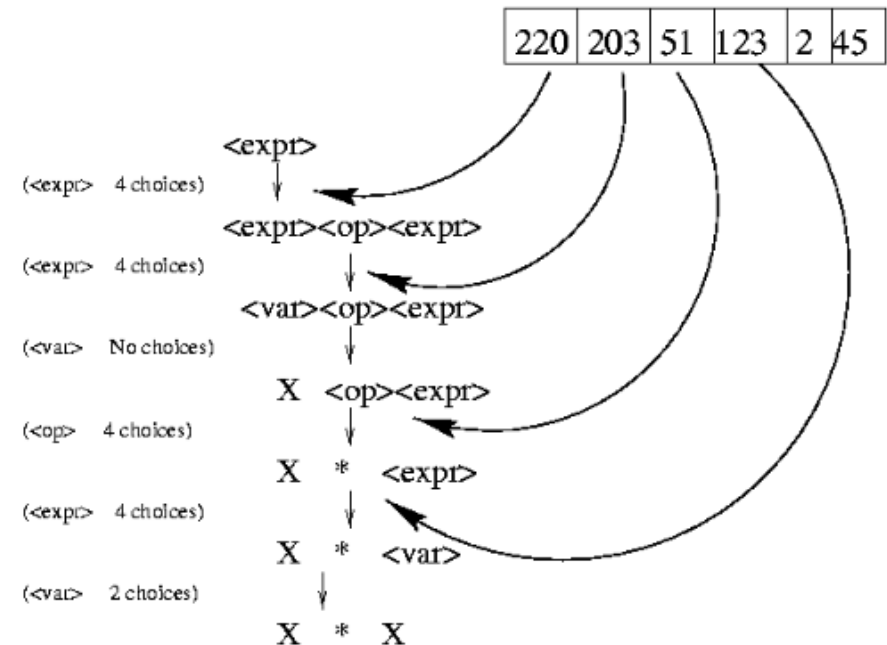
Grammatical evolution

(1) $\langle \text{expr} \rangle ::= \langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle$ (A)
 | $(\langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle)$ (B)
 | $\langle \text{pre-op} \rangle (\langle \text{expr} \rangle)$ (C)
 | $\langle \text{var} \rangle$ (D)

(2) $\langle \text{op} \rangle ::= +$ (A)
 | $-$ (B)
 | $/$ (C)
 | $*$ (D)

(3) $\langle \text{pre-op} \rangle ::= \text{Sin}$ (A)
 | Cos (B)
 | Tan (C)

(4) $\langle \text{var} \rangle ::= X$ (A)



Implementacija

Kako zapisati jedinke?

- Bitne informacije: dubina čvora, broj djece čvora, broj potomaka čvora, maksimalna dubina stabla, dubina stabla u čvoru
- **Kao stablo** (čvorovi s pokazivačima na roditelje/djecu):
 - Predizračunati sve informacije i pospremiti u čvorove
 - On-line računati informacije po potrebi
- **Kao polje**:
 - Prefiksni zapis
 - Indeksi pozicije djece

Kako ostvariti križanje?

- Kako odabrati točku prekida?
 - Random između 1 i ukupnog broja čvorova
 - Želimo favorizirati funkcijske čvorove (odrediti broj terminala!)
- Izrada nove jedinice:
 - Obavezno kopirati čvorove
 - Paziti na uvjet maksimalne dubine stabla:
 - Dubina trenutnog čvora + dubina stabla u novom čvoru < maksimalna dubina stabla!
 - Ako ne vrijedi probati ponovo odabrati točke (nekoliko puta) ili probati ponovo odabrati roditelje

Kako ostvariti mutaciju?

- Odabrati nasumično čvor (isto kao kod križanja)
- Generirati novo stablo (full ili grow metodom)
 - Paziti da se generira podstablo koje neće narušiti ograničenje maksimalne dubine cijelog stabla!
- Zamijeniti odabrano podstablo staviti novogenerirano podstablo

Zaključak

Zaključak

- GP je primjenjiv na širok spektar problema
- Dosta otvorenih problema i mogućnosti za poboljšanje
- Različiti prikazi rješenja
- Područje koje se aktivno istražuje

Što smo na FER-u radili s GP-om

- razni oblici raspoređivanja
- konstrukcija kombinatoričkih fja za kriptografiju
- problem usmjeravanja vozila (VRP)
- oblikovanje kombinatoričkih sklopova
- upravljanje robotom
- detekcija malignih tvorevina na RTG snimkama pluća
- automatsko rezanje
- dijagnosticiranje plućne embolije
- automatska paralelizacija
- primjena GP u strojnom učenju (status posla u grozdu)
- trgovanje dionicama
- strategije zamjene stranica u straničenju

Reference

- Knjige/stranice
 - [Field guide to GP](#)
 - [GP Notebook](#)
 - [Koza GP site](#)
- Alati
 - [ECJ](#), [Watchmaker](#) (Java)
 - [ECF](#), [ECF LAB](#), [ECF SRM](#) (C++)
 - [HeuristicLab](#) (C#)
 - [EO](#), [OpenBEAGLE](#) (C++)
 - [PyEvolve](#), [DEAP](#) (Phyton)