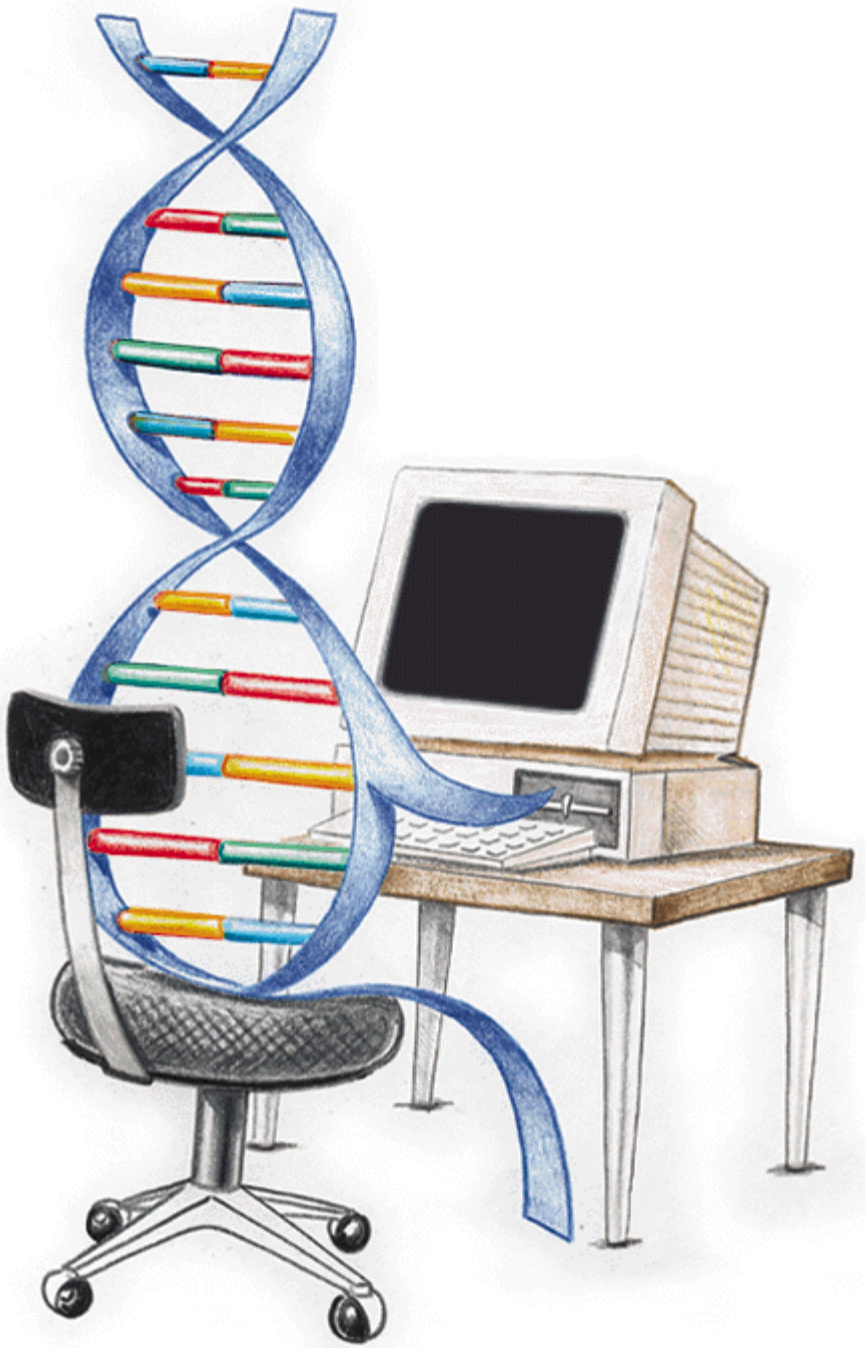
Evolucijsko računanje

prof. dr. Marko Robnik Šikonja
oktober 2015



Osnove

- ideje iz biologije: uporaba teorija evolucije kot algoritma
- napredek (učenje, optimizacija) kot tekmovanje razvijajočih se struktur (rešitev, programov)
- delujejo na populacijah (paralelizacija)
- prilagodljivost, robustnost

Zasnova evolucijskega računanja

generiraj populacijo struktur

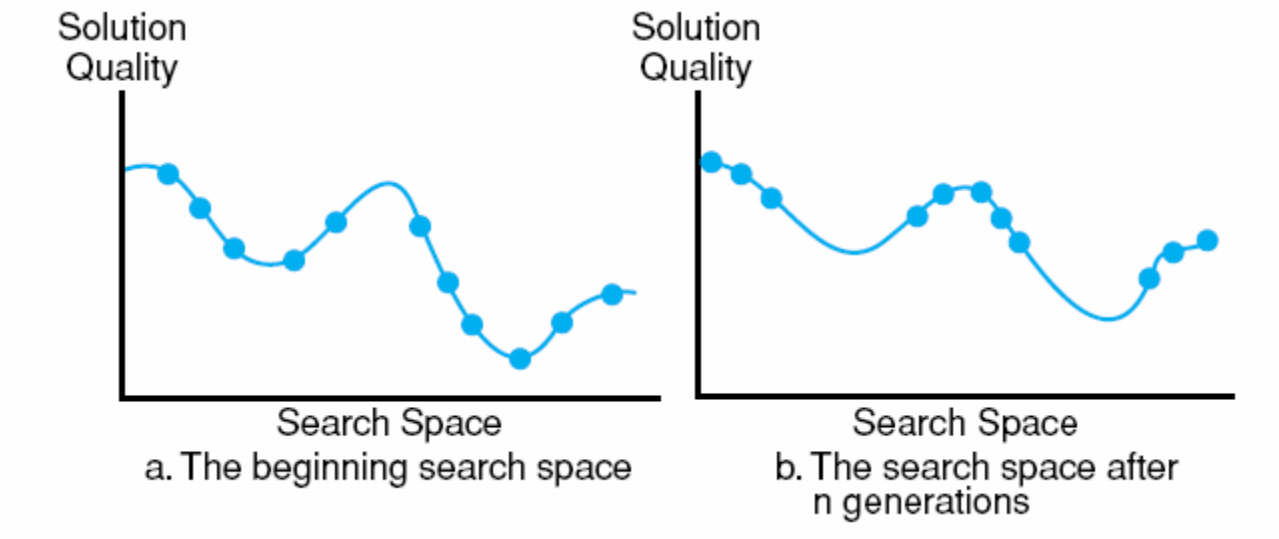
do {

izračunaj kvaliteto struktur
izberi strukture za reprodukcijo glede na kvaliteto
ustvari nove inačice struktur
zamenjaj stare strukture z novimi

} while (not zadovoljen)

- ogromno različnih možnosti

Primer uspešne evolucije



Analiza prednosti in slabosti

- robustnost, prilagodljivost in splošnost
- potrebno le šibko znanje o problemu (samo ocena kvalitete rešitve)
- več alternativnih rešitev
- možni hibridizacija in paralelizacija

- neoptimalne rešitve
- nastavitev številnih parametrov
- računska zahtevnost

- no-free-lunch teorem

Poglavitne tehnike

- genetski algoritmi (fiksna dolžina kode)
- genetsko programiranje (fleksibilna dolžina kode, npr. drevesa)
- metode roja (roji delcev, kolonija mravelj)
- samo-organizirajoča polja
- ...

Biološke vzporednice

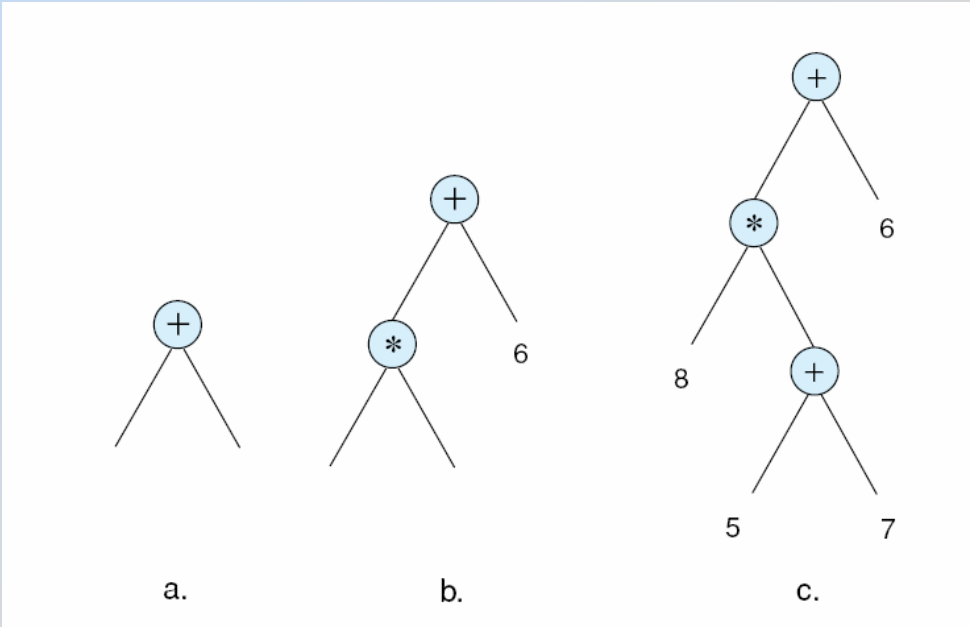
- gen opisuje eno od lastnosti
- alel je vrednost (pomen) lastnosti, npr. barva oči
- evolucija je variacija frekvence alelov v populaciji skozi čas
- reprodukcija, variacija (mutacija, križanje), selekcija

Ključni pojmi evolucijskega računanja

- Predstavitev: podatkovna struktura in operacije na njej
- Uspešnost (fitness): hevristična ocena
- Spremenljivost populacije
- lokalni in globalni ekstremi
- koevolucija, spremenljivost funkcije uspešnosti

Predstavitve

- bitna: osebek kot zaporedje bitov
- vektorska: vektor realnih ali celih števil
- nizovna: zaporedje znakov (besedilo ali oznake)
- permutacije
- drevesna: funkcije, izrazi, programi
- ...



Križanje (crossover)

- eno, dvo ali večmestno,
- naj bi ohranilo strukturo osebka
  - bitna predstavitev

Starši: 1101011100 0111000101

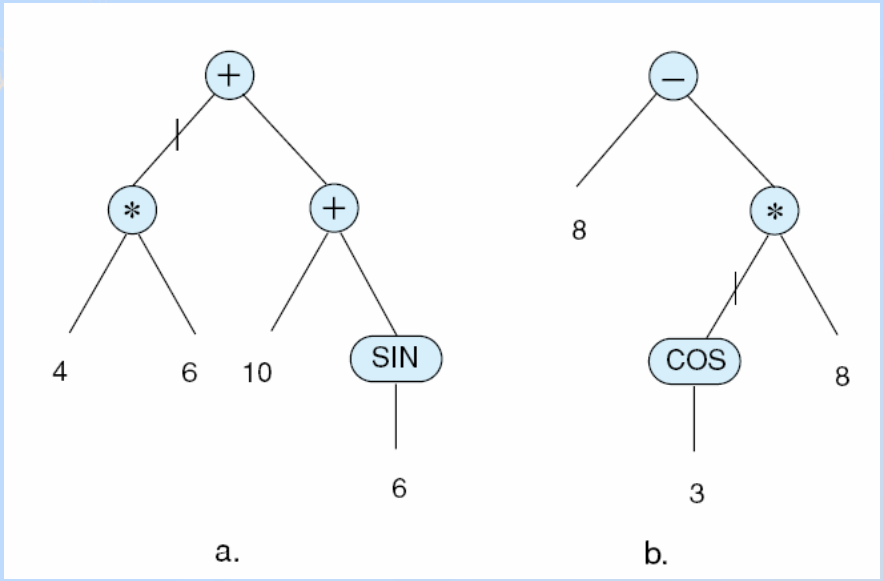
Potomci:1101010101 0111001100

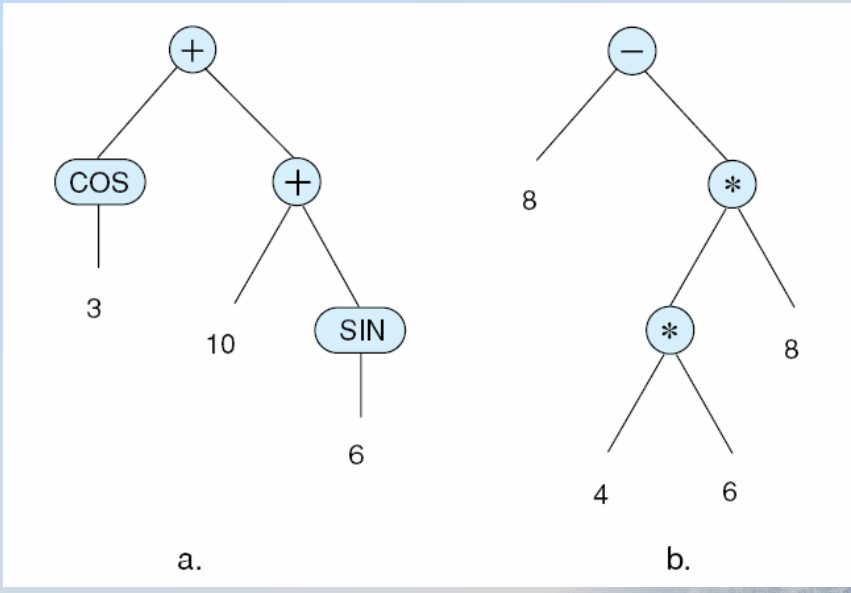
- Vektorska predstavitev:

Starši: (6.13, 4.89, 17.6, 8.2) (5.3, 22.9, 28.0, 3.9)

Potomci: (6.13 , 22.9, 28.0, 3.9) (5.3, 4.89, 17.6, 8.2)

Križanje (drevesa)





Primer: trgovski potnik

- 9 mest: 1,2 ..9
- bitna predstavitev s 4 biti
  - 0001 0010 0011 0100 0101 0110 0111 1000 1001
  - kako realizirati križanje? dobimo lahko neveljavne predstavitve
- predstavitev s permutacijo in urejeno križanje
  - ohrani cele sekvence
  - pri ostalih ohrani vrstni red (od drugega reza izpiši vrstni red, odstrani že obstoječe)

1 9 2 | 4 6 5 7 | 8 3 → x x x | 4 6 5 7 | x x ↘ 2 3 9 | 4 6 5 7 | 1 8

4 5 9 | 1 8 7 6 | 2 3 → x x x | 1 8 7 6 | x x ↗ 3 9 2 | 1 8 7 6 | 4 5

Grayevo kodiranje binarnih števil

- Ohranjanje lokalnosti pri binarnem zapisu
- Zaporedna števila imajo razliko le na enem mestu
- Zaželena lastnost: velikost spremembe v kodi ustreza velikosti spremembe števila

Binary	Gray
0000	0000
0001	0001
0010	0011
0011	0010
0100	0110
0101	0111
0110	0101
0111	0100
1000	1100
1001	1101
1010	1111
1011	1110
1100	1010
1101	1011
1110	1001
1111	1000

Adaptivno križanje

- poskus prilagoditve križanja različnim fazam evolucije
- uporaba nastavkov, ki so podvrženi evoluciji
- 0 – uporabimo istega starša, 1 uporabimo drugega
- evolucija nastavkov lahko sledi drugačni dinamiki

	Gene	Template
Parent 1	1.2 3.4 5.6 4.5 7.9 6.8	010101
Parent 2	4.7 2.3 1.6 3.2 6.4 7.7	011100
Child 1	1.2 2.3 5.6 3.2 7.9 7.7	010100
Child 2	4.7 3.4 1.6 4.5 6.4 6.8	011101

Mutacija

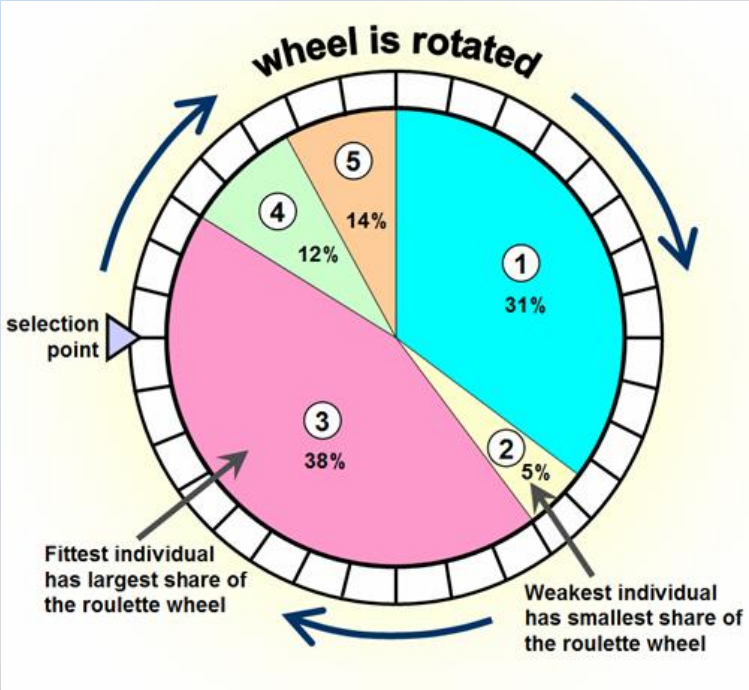
- doda novo informacijo
- pretirana verjetnost mutacije vodi v naključno iskanje
- Osebek: 0111001100 --> 0011001100
- enotočkovna, večtočkovna, uniformna verjetnostna
- Lamarckovska (izbira najboljše mutacije izmed vseh možnih)

Evolucijski model

- način razmnoževanja
- kako ohraniti dobre osebke, kako zagotoviti raznolikost, preprečiti prezgodnjo konvergenco

Proporcionalna izbira

- verjetnost izbire za razmnoževanje je proporcionalna uspešnosti
- postopek
  - na podlagi funkcije uspešnosti ustvari verjetnostno distribucijo $p_i = f_i / \Sigma f_j$
- vzorčenje na podlagi te distribucije
  - implementacija: ruletno kolo
- vnaprej določeno število potomcev $n_i = n\, p_i$



Rangovna izbira

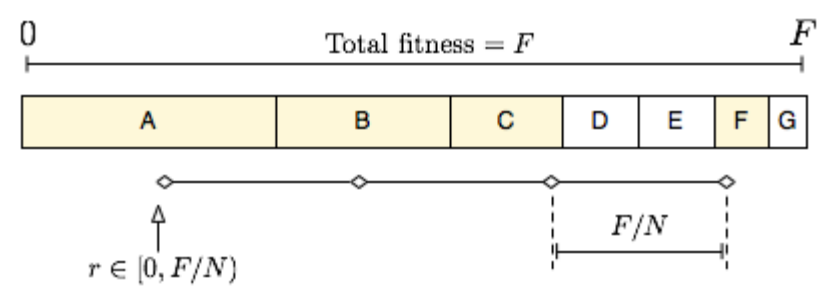
- osebke sortiramo glede na kvaliteto od najslabšega do najboljšega in vsakemu priredimo rang r_i
- postopek
  - na podlagi rangov ustvari verjetnostno distribucijo $p_i = r_i / \Sigma r_j$
- vzorčenje na podlagi te distribucije

Turnirska izbira

1. določi t=velikost turnirja, p=verjetnost izbire
2. za turnir izberi naključno t osebkov iz populacije
3. izberi najboljšega z verjetnostjo p
4. izberi drugega najboljšega z verjetnostjo $p(1-p)$
5. izberi tretjega najboljšega z verjetnostjo $p(1-p)^2$
6. ...

Stohastično univerzalno vzorčenje

- stohastic universal sampling (SUS)
- nepristrano, manjše odstopanje kot ruleta
- osebke naključno razvrstimo, vsak dobi delež proporcionalen kvaliteti F = Σf_j, izberemo N naslednikov
- naključno izberemo r ∈ [0, F/N]
- izberemo osebke, ki ležijo r + i*N, i ∈ 0, 1,..., N-1]



SUS učinkovita implementacija

```
// Vhod: verjetnostna distribucija osebkov P,
         število naslednikov N

// Vrača vektor c = (c1, c2, ..., cm ),kjer ci predstavlja število
   naslednikov osebka i, Σci = N

int[]  SUS(double[] P, int N) {

   naključno izberi r ∈ [0, 1/N]
   sum = 0.0;
   for (i = 1 ; i <= m ; i++) {
      ci = 0 ;
      sum += Pi;
      while (r < sum) {
           ci ++ ;
           r += 1/N ;
      }
   }
   return c ;

}
```

Nadomeščanje osebkov

- zamenjaj vse, zamenjaj glede na kvaliteto (ruleta, rang, turnir, naključno)
- elitizem: ohrani določen del najboljših
- lokalni elitizem: otroci zamenjajo starše, če so boljši

Enoturnirska izbira

1. naključno razbij populacijo na majhne skupine
2. dva najboljša osebka iz vsake skupine se križata in njuna potomca nadomestita najslabša osebka v skupini

- prednosti: pri skupinah velikosti *g*, preživi v naslednjo generacijo *g-2* najboljših (ne izgubljamo prehitro dobrih, maksimalna kvaliteta populacije ne pada)
- ne glede na kvaliteto ima še tako dober osebek le dva potomca (ne izgubljamo raznolikosti populacije)

Velikost populacije

- premajhna populacija nima zadosti različnih osebkov
- velika populacija je računsko zahtevna, na začetku vsebuje mnogo šibkih osebkov, ki za izboljšanje potrebujejo čas

Nišna specializacija

- osebki si lahko postanejo preveč podobni in se izboljšujejo le v določeni okolici (evolucijska niša)
- če lahko definiramo podobnost osebkov (npr. med vektorji v $\Re^n$) definiramo radij podobnosti r
- funkcijo kvalitete zmanjšamo glede na število osebkov znotraj radija podobnosti

$f'_i = f_i / q(r,i)$ $q(r,i) = \{ 1;\ sim(i) <= 4,\ sim(i)/4\ sicer\}$

- tako rešitve, ki jih najde več osebkov, postanejo manj privlačne

Kdaj končati?

- kriterij za kvaliteto rešitve
- fiksno število generacij
- izraba proračuna (npr. računskega časa)
- najboljši osebek se več generacij ne izboljša
- kombinacija

Zakaj evolucijski pristopi delujejo?

- hipoteza zidakov (building blocks hypothesis)
- zidak je nizkonivojska predstavitev (shema), ki prispeva k nadpovprečni kvaliteti osebka
- hipoteza: genetski pristopi implicitno in učinkovito identificirajo in sestavljajo zidake
- sestavljanje rešitve iz delnih nizkonivojskih podrešitev
- nepotrjena, kritizirana hipoteza, ki ji nasprotujejo nekateri poskusi z mutacijami
- evolucijski algoritmi vzorčijo rešitve (usmerjajo preiskovanje) z večjo verjetnostjo iz podprostorov, ki so bolj perspektivni

Tipične aplikacije

- izredno širok spekter aplikacij v problemih, kjer želimo globalno optimizacijo
- razporejanje opravil in urniki
- optimizacija parametrov v problemih, kjer iščemo globalni optimum
- bioinformatika, strojno učenje
- načrtovanje
- večkriterijsko optimiranje
- ...

Uporabnost EA

- problemi z mnogimi lokalnimi ekstremi
- problemi, kjer je na voljo le funkcija kvalitete, ne pa tudi njen odvod
- kjer ne obstajajo specializirane metode za reševanje
- robustna metoda
- možnost kombiniranja z drugimi pristopi, npr. lokalno optimizacijo

Številne orodjarne in knjižnice

- CIlib – computational intelligence library
- EO (C++) - evolutionary computation library
- ECJ (Java)
- EvA2 (Java),
- JAGA (Java)
- ECF- Evolutionary Computation Framework (C++)
- Matlab, …
- R: Rfreak,ppso, numDeriv,…

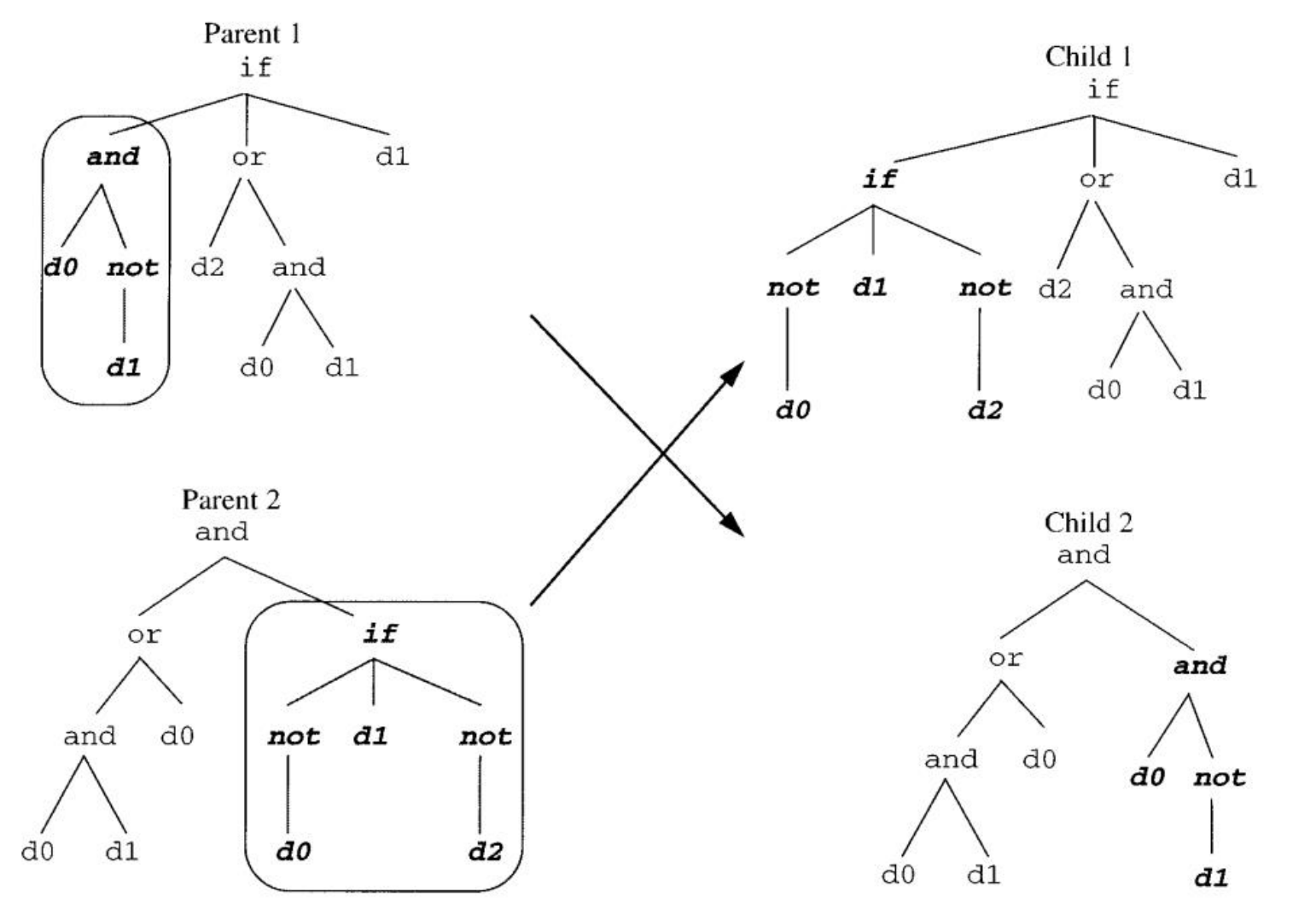
Genetsko programiranje

- poskus evolucije zahtevnejših podatkovnih struktur (funkcij, programov, izraznih dreves)
- tudi po uporabi genetskih operatorjev (križanje, mutacija) mora ostati struktura veljavna

Križanje dreves

- sintaktične omejitve zahtevajo pravilne vrednosti (terminali) v listih in samo funkcijske operatorje v notranjih vozliščih ter pravo število argumentov (poddreves)
- poenostavitev – zaprtje (closure): vse funkcije vračajo isti podatkovni tip, kar zmanjša število sintaktičnih omejitev
- preprosta rešitev: zamenjava dveh naključno izbranih poddreves
- potrebne omejitve glede globine dreves ali števila vozlišč v drevesu
- tipizirano križanje: šele po izbiri prvega poddrevesa, izberemo ustrezni tip vozlišča v drugem drevesu

Križanje izraznih dreves: primer



Genetsko programiranje: uporaba

- učenje struktur pri načrtovanju izdelkov, topologije nevronskih mrež, gramatik
- težava: računska zahtevnost zaradi ogromnega prostora stanj

Druge biološko navdahnjene metode

- značilnosti:
  - fiksna populacija, ki se usmerjeno spreminja
  - populacija med seboj komunicira
- inteligenca roja (particle swarm optimization)
  - kolektivno obnašanje: roji in jate rib, ptic
  - vsak posameznik v roju ima določeno avtonomijo
  - vsak zase išče hrano, ko jo najde, obvesti ostale
- kolonija mravelj (ant colony)

Inteligenca roja

- osebek (delec) je predstavljen z dvema vektorjema
  - lokacijski vektor $x = (x_1, x_2, \ldots)$, ki ustreza genetskemu zapisu pri genetskih algoritmih
  - vektor hitrosti $v = (v_1, v_2, \ldots)$ predstavlja hitrost in smer potovanja delca v časovnem koraku
  - če sta $x(t-1)$ in $x(t)$ lokaciji v času t-1 in t velja

$$\vec{v} = \vec{x}(t) - \vec{x}(t-1)$$

- vsak delec je na začetku inicializiran z naključno lokacijo in naključnim (majhnim) vektorjem hitrosti, (npr. polovica vektorja razdalje do naključne druge točke, naključna majhna vrednost, nič)

Izmenjava informacij v roju

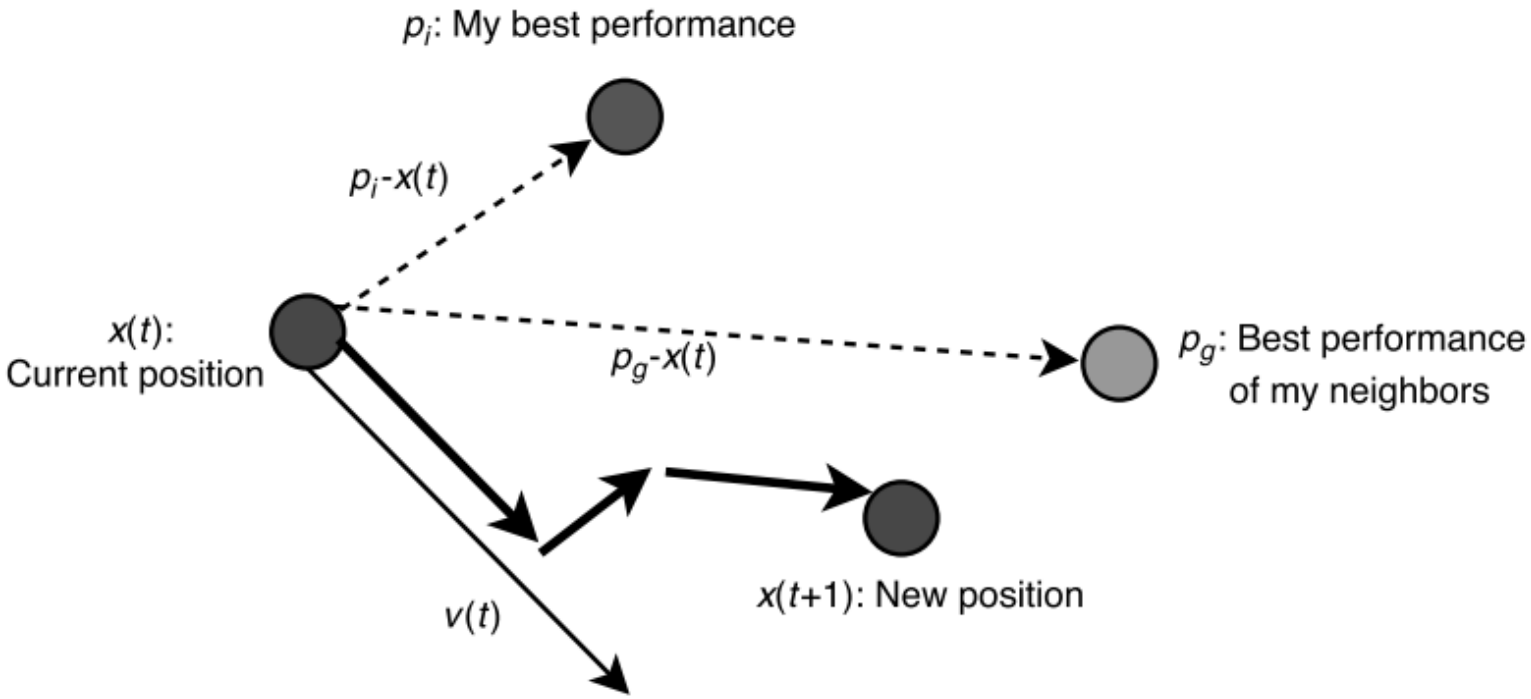
- tekom iskanja vsak delec pozna dosedanjo
  - svojo najboljšo lokacijo x^*
  - najboljšo lokacijo x^+ svojih informatorjev (npr. nekaj delcev iz soseščine ali nekaj naključnih delcev)
  - skupno najboljšo lokacija $x^!$

Premikanje delcev

- v vsakem časovnem koraku se izvedejo naslednje operacije

1. izračunamo kvaliteto vsakega delčka in po potrebi popravimo x^*, x^+ in $x^!$
2. določimo spremembo delčka
   - vektor hitrosti spremenimo tako, da v njem upoštevamo smeri proti x^*, x^+ in $x^!$
   - smeri dodamo tudi nekaj naključnega šuma (vsaki dimenziji prištejemo različne naključne vrednosti)
3. delček premaknemo v smeri vektorja hitrosti

Ilustracija premikanja



PSO - parametri

- α - delež ohranitve dosedanjega vektorja hitrosti v
- β - delež najboljše lastne vrednosti x*, ki ga upoštevamo
  prevelika vrednost teži k svojemu maksimumu, ne pa k odkrivanju globalnega optimuma (dobimo skupino požrešnih individualnih iskalcev, ne pa skupinskega iskanja)
- γ - delež najboljše vrednosti informatorjev x^+, ki ga upoštevamo
  učinek med β in δ, odvisno tudi od števila informatorjev: več informatorjev daje večji povdarek globalnemu maksimumu, manj lokalnemu (če so informatorji izbrani naključno).
- δ - delež najboljše globalne vrednosti x^l, ki ga upoštevamo
  če je vrednost velika, se delci pomikajo proti globalnemu maksimumu in dobimo eno samo požrešno iskanje, namesto več več ločenih iskanj; (marsikdaj postavimo vrednost kar na 0)
- ε - hitrost premikanja delcev
  velika vrednost povzroči, da delci napredujejo v velikih skokih (nevarnost preskoka), in se hitro usmerijo proti najboljšim področjem, a težko podrobno optimizirajo vrednost Ponavadi se vrednost nastavi na 1.
- swarmsize – velikost roja

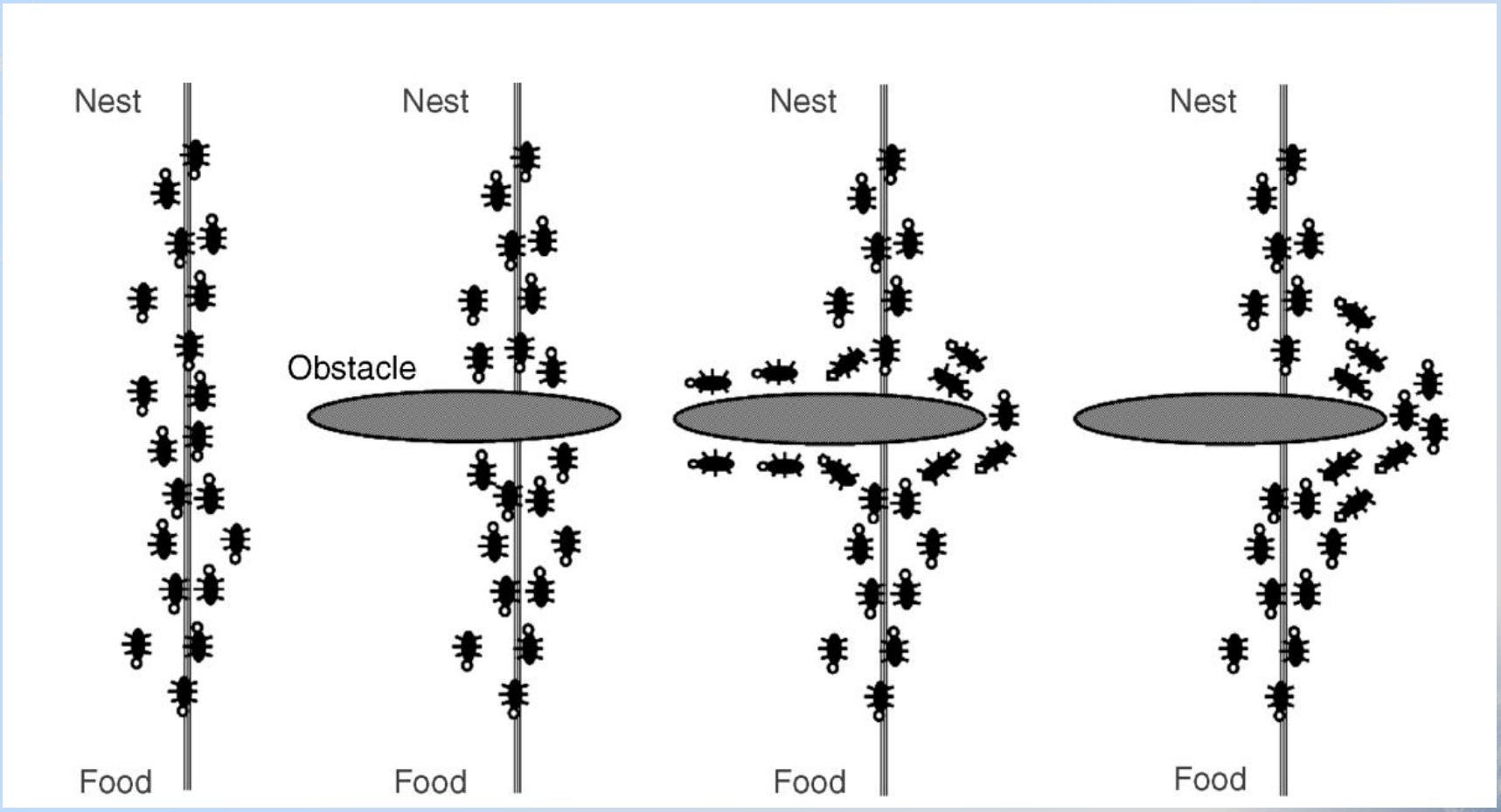
PSO implementacija

```
P = []
for (i=0 ; i < swarmsize ; i++)
    P_i = nov delec z naključno pozicijo x in naključno hitrostjo v
best = null
do {
   for  (i=0 ; i < swarmsize ; i++) {
      določi kvaliteto(P_i)
      if  ( kvaliteta(P_i) >  kvaliteta(best) )
          best = P_i
   }
   for (i=0 ; i < swarmsize ; i++)  {
      x* = lokacija z dosedaj največjo kvaliteto x_i
      x+ = lokacija z dosedaj največjo kvaliteto informantov x_i
      x! = lokacija z dosedaj največjo kvaliteto med vsemi delci
      for (j=0; j < steviloDimenzij; j++) {
         b = naključno število med 0 in β
         c = naključno število med 0 in γ
         d = naključno število med 0 in  δ
         v_j = αv_j + b(x*_j − x_j)  + c(x+_j − x_j) + d(x!_j − x_j)
      }
      x_i = x_i + ε·v
} while (best ni optimalen AND čas ni iztekel)
return best
```

Kolonija mravelj

- mravlje poti k hrani označijo s feromoni
- iskanje dobrih poti skozi graf, npr. trgovski potnik
- kombinatorična optimizacija
- ideja
  - na začetku mravlje iščejo hrano naključno
  - mravlja, ki najde hrano, se vrne in označi svojo pot s feromoni
  - druge mravlje, ki naletijo na feromonsko sled, ji sledijo in jo ojačajo, če najdejo hrano
  - s časom sled slabi
  - na slabših (daljših) poteh lahko izhlapi več feromonov
- izhlapevanje poskrbi, da iskanje ne konvergira k enemu samemu cilju
- primerno tudi za probleme, ki se s časom spreminjajo, npr. nove poti, nove ovire

Ilustracija iskanja mravelj



ACO: osnutek algoritma

inicializiraj feromonske sledi

do {

  foreach mravlja

    sestavi rešitev z uporabo feromonskih sledi

    popravi sledove: izhlapevanje, ojačanje

} **while** (! satisfied)

return najboljša najdena rešitev

ACO: podrobnosti

- konstrukcija rešitve: verjetnost prehoda med stanji odvisna od feromonov in cene prehoda
  - dodatni parametri kontrolirajo vpliv feromonov in cene prehoda
- ažuriranje količine feromonov

$$\tau_{i,j} = (1-\rho)\tau_{i,j} + \Delta\tau_{i,j}$$

  - izhlapevanje feromonov
    - ρ je hitrost izhlapevanja
  - obnavljanje sledi
    - ko vse mravlje sestavijo rešitev
    - med konstrukcijo
    - med konstrukcijo, ko pride mravlja na cilj
    - glede na kvaliteto drugih rešitev
    - glede na rang rešitve
    - elitistično (najboljša pot dobi največ)

$$\Delta\tau_{i,j} = \begin{cases} 1/C & \text{če mravlja potuje po povezavi i, j} \\ 0 & \text{sicer} \end{cases},$$

kjer je C cena poti i, j

- številne inačice

ACO: trgovski potnik

- vozlišča (mesta) 1,2,…,n
- cena prehoda med vozliščem i in j je $c_{i,j}$
- privlačnost prehoda med i in j je ponavadi inverz cene $\eta_{i,j} = 1/ c_{i,j}$
- verjetnost prehoda med i in j:
- α - vpliv feromonov
- β - vpliv cene povezave

$$p_{i,j} = \frac{\tau_{i,j}^{\alpha} \eta_{i,j}^{\beta}}{\sum \tau_{i,j}^{\alpha} \eta_{i,j}^{\beta}}$$

Še več idej iz biologije

- čebele
- imunski sistem
- kapljice vode
- simulirano ohlajanje
- ...

Večkriterijska optimizacija

- kriterijska funkcija kvalitete rešitve ima lahko več komponent, ki jih vse želimo optimirati, npr. cena, kvaliteta izdelka, vpliv na okolje, ...
- min $F(x)=\min (f_1(x), f_2(x), \ldots, f_n(x))$
- Pareto optimalna rešitev: ne moremo izboljšati ene od komponent, ne da bi pri tem poslabšali vsaj eno od ostalih
- iščemo nabor Pareto optimalnih rešitev, med katerimi se na koncu odločimo
- pogosto rešujemo z genetskimi algoritmi, kjer pri reprodukciji upoštevamo raznolikost rešitev po vseh kriterijih

Primer: upravljanje pametne stavbe

- preprosta inačica: imamo grelec, akumulator, sončne celice, elektriko iz omrežja
- kriterija: cena in udobje uporabnika (razlika v temperature od željene)
- kromosom določa urnik polnjenja in praznjenja baterije, delovanja grelca
- čas upravljanja (24 ur) razdelimo na npr. 15 minutne intervale

Pametna stavba: sestava kromosoma

- temperature: za vsak interval določimo željeno temperature na interval Tmin in Tmax
- baterija+: če fotovoltaične celice proizvedejo dovolj energije, določimo: 1 baterija naj se polni, 0 naj se ne polni
- baterija-: če fotovoltaične celice ne proizvedejo dovolj energije, določimo: 1 baterija naj se prazni, 0 naj se ne prazni
- porabniki: vsak porabnik ima svoj urnik, ko deluje (1) in ko ne deluje

Primer urnika

Primer vrnjenih rešitev