EVT - Evoluční výpočetní techniky

Obecný postup EA (Obecný evoluční cyklus):

1. **Vymezení parametrů** (řídící, ukončovací), **stanovení účelové funkce**/fitness (=normovaná hodnota účelové funkce) → slouží k vyhodnocení kvality jedince
2. **Generace náhodné prvopočáteční populace jedinců**. Jedinec = vektor čísel.
3. **Ohodnocení** všech jedinců přes účelovou funkci
4. **Výběr rodičů** podle jejich kvality (občas lze i jinak, např. náhodně)
5. **Křížení rodičů** ⇒ tvorba potomků
6. Každý potomek je **zmutován** pomocí vhodného náhodného procesu
7. Každý nový jedinec je **opět ohodnocen** přes účelovou funkci
8. **Výběr nejlepších** jedinců (zpravidla z generace rodičů a jejich potomků)
9. **Likvidace staré**, nevhodné **generace**, na její místo **přichází nová generace** z vybraných nejlepších jedinců
10. Opakování od kroku 4.

Holy trinity for EAs: Křížení, Mutace, Výběr

Stručná historie EVT

- Darwin - Theory of Evolution - 1859
    - survival of the fittest (fitness)
        - ti co mají dobré vlastnosti přežijí, ti co mají horší postupně vymizí
    - not specific about how the traits are passed on
    - ⇒ inspiroval výběr v EVT
- Mendel - Founder of genetics
    - cca ve stejne dobe jako Darwin, ale prace byla “objevena” az ve 20. st.
    - vysvětlil jak funguje dědičnost - geny
    - ⇒ inspiroval křížení v EVT
- Hugo de Vries
    - znovu objevil Mendelovu praci (zkopiroval ho bez citace :D)
    - mutace
- Turing
- Barricelli ??

Shittalk about algoirthms

- Každý je dobrý na něco jiného ⇒ NFL
- mezi EA patří např
    - Ant Colony Optimization
        - dobrý na kombinatoriku → traveling salesman problem
    - Immunology System Method
    - Scatter Search
    - Particle Swarm
- Na každý optimalizační problém se dá nahlížet jako geometrický problém (hledání min max na ploše)
- multi-modalita = víc než jeden extrém
- Společné rysy EAs
    - Jednoduchost - easy to implement

- Hybridnost - mohou pracovat s celými čísly, reálnými čísly, nebo diskrétními hodnotami (vybrané z nějaké konečné množiny, např. {–5, 2, 3, 8}.)
- Použití dekadických čísel - no need for binary or Gray Code
- Rychlost - rychlejší než klasické metody no kiding
- Schopnost “nalézt jehlu v kupce sena”
- Schopnost nalézt vícenásobné řešení

Základní koncepty EVT

- Individual = jedinec
  - jedinec je dán několika parametry v závislosti na zkoumaném problému
  - Fitness jedince - speciální parametr; počítán zpravidla pomocí účelové funkce (+ normalizace); tento parametr se však přímo neúčastní evolučního procesu
- Population
  - Skupina více jedinců
  - zpravidla znázorněná jako matice M x N, kde M je počet jedinců v populaci a N je počet jejich parametrů
  - specimen = vzorový jedince
    - udává typ a hranice jednotlivých parametrů, používá se například při korekci parametrů po vytvoření potomka
    - *zajišťuje, že všechny parametry jedinců budou náhodně generovány uvnitř povolených hranic prostoru možných řešení*
  - prvotní generace většinou náhodně, každá nová vzniká výběrem ze staré a jejích potomků
  - populace se v čase nahrazuje
- Objective function - asi funkce co analyzujeme, ergo účelová funkce
- Representation of individuals
  - reprezentace jedince může být různá
    - historicky nejstarší reprezentace je **binární řetězec**
      - popř Gray Code (další bod)
    - reálná, celá čísla
    - diskrétní čísla (pouze omezený počet čísel)
    - nenumerické hodnoty(je potřeba speciální techniky)
    - strom

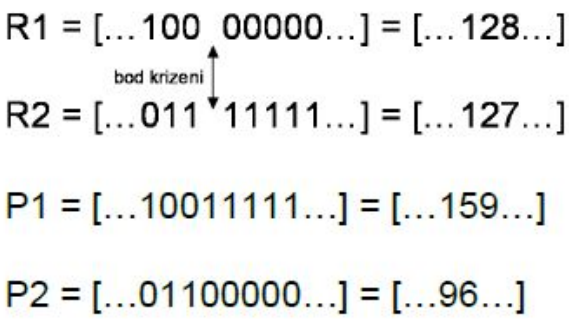
NFL, No free Lunch

- Ve zkratce - neexistuje “bůh” mezi algoritmy, tj univerzální algoritmus na vše
- U algoritmů hledající extrémy platí, že si jsou ekvivalentní pokud je zprůměrujeme pro všechny možné problémy. To neznamená, že je jejich výkon stejný.
- Žádný algoritmus není shit → funguje úžasně na některé skupině a na všech ostatních může být shit :D.

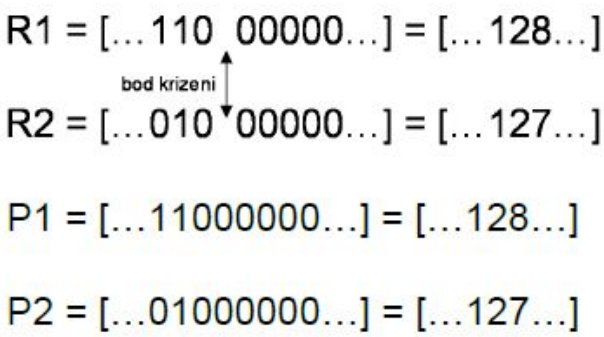
Gray code (Grayův kód)

- V případě, že používáme reprezentaci jedinců pomocí binárního řetězce, mohou zde vznikat velké skoky. Např 15,16,17 ⇒ 01111, 10000, 10001 přechod z 15 a 16 znamená inverzi 5ti bitů, přechod mezi 16 a 17 inverze jednoho bitu, při křížení 15 a 16 pak můžou vzniknout kraviny jako potomek 0 a 31 při křížení na prvním bitu

- proto vznikl speciální kód, ve kterém se tyto skoky nenacházejí; technik na převod je více ale vždycky se rozšiřuje kód o další výpočetní krok, křížení pak dopadá tak jak by se očekávalo
- např pomocí XORu - další číslo je vždy předchozí převedené číslo XOR momentální
- 0 = 00
- 1 = 00 XOR 01 ⇒ 01 (GC předchozího XOR binarni hodnota current ⇒ GC current)
- 2 = 01 XOR 10 ⇒ 11, etc.



Obr. 6.10 Křížení jedinců ve standartním binárním kódování (R1, R2 – rodiče; P1, P2 – potomci).

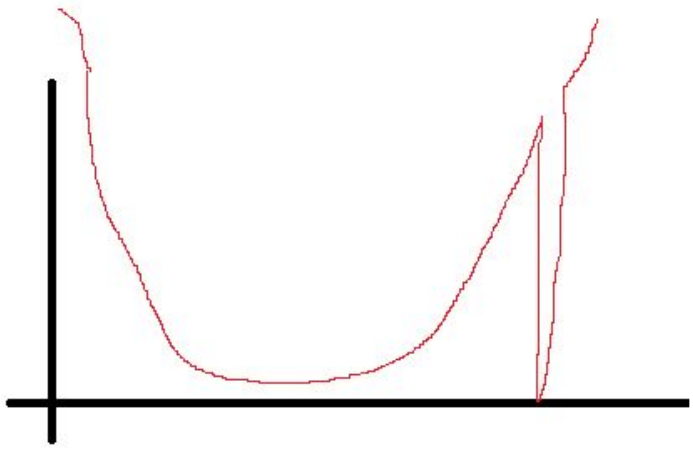


Obr. 6.11 Křížení jedinců v Grayově kódování (R1, R2 – rodiče; P1, P2 – potomci).

Classification of optimization methods

- Classification of optimization methods (enumerative, deterministic, ..., heuristic) and their choice in the knowledge of the objective function, principles of design objective function.
- first option
  - **Enumerative**
    - Algoritmus provede výpočet všech kombinací
    - vhodný pro problémy, kde jsou argumenty účelové funkce diskrétního charakteru a nabývají malých hodnot.
  - **Deterministic**
    - založeno na matematických metodách
    - většinou potřebují nějaké omezení typu problém je lineární, konvexní, účelová funkce má jen jeden extrém, …
    - výsledek je pak jediné řešení
    - **Greedy, Horolezecký, Depth first, Broadth first**
  - **Stochastické (Náhodné)**
    - náhodné hledání, pomalé, dobré pro hrubý odhad
    - **Tabu search, Monte Carlo, Simulované žíhání, Stochastický horolezecký, Random walk, Evoluční strategie**

    - **Smíšené**
        - vykazují prvky **deterministických** + **stochastických**
        - **evoluční algoritmy**
        - efektivní, robustní, schopné více řešení a schopné práce s blackboxem
        - **Diferenciální evoluce, SOMA, GA, Particle swarm, Scatter Search,**
- second option
    - rozdělení na **tradiční** (dávají přesný výsledek) a **heuristické** (přibližný výsl.)
    - **tradiční** se dělí na analytické a konstruktivní řešení
        - analytické = lineární programování, lokální prohledávání
        - konstruktivní = divide and conquer, dynamické programování
    - **Heuristické** → **deterministické** a **pravděpodobnostní** metody
        - Deterministické
            - Tabu search
        - Pravděpodobnostní (Stochastické)
            - Zabývají se body
                - Simulované žíhání
                - Stochastický horolezecký algoritmus
            - Zabývají se populací
                - EA
                - Genetické Algoritmy / Genetické prog.
                - Evoluční programování
                - Evoluční strategie
- Visualization and transfer minima search to finding maxima and vice versa.
    - well we can always multiply the objective function by -1
    - Visualisation of what? of that transfer? Well the function will be inversed o.O
- Explain the concept of misleading function.
    - Did not find anywhere
    - Probably ment Deceptive Test Function
        - its global extreme is elsewhere than it seems
        - art by MarKay(WOW, AMAZINK ART M8)



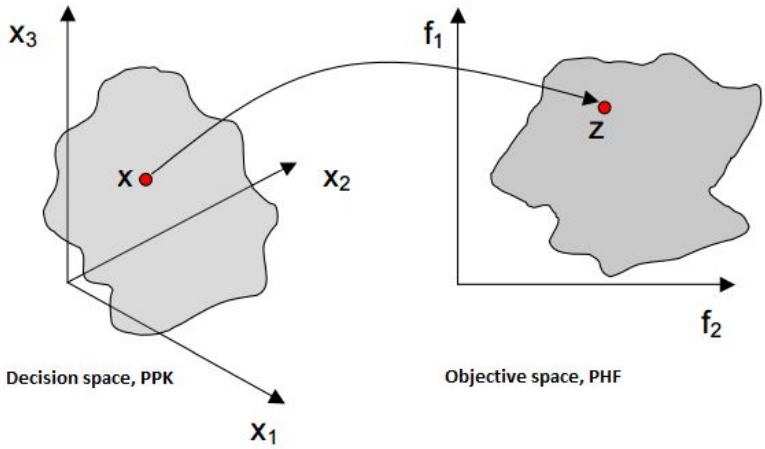
Paretova množina, Paretova hranice

- Example of objective function for multi-purpose optimization.
- Define a multi-purpose optimization Explain the concept of Pareto set, what is the Pareto frontier.

- Give examples for borders: min-min, max-min, max-min, max-max problem. Explain the concept of local Pareto frontier.
- Explain the concept of dominant and subdominant solutions, give an example.
- What is the relation must be the individual components of the objective function to the Pareto set was / was not the point.

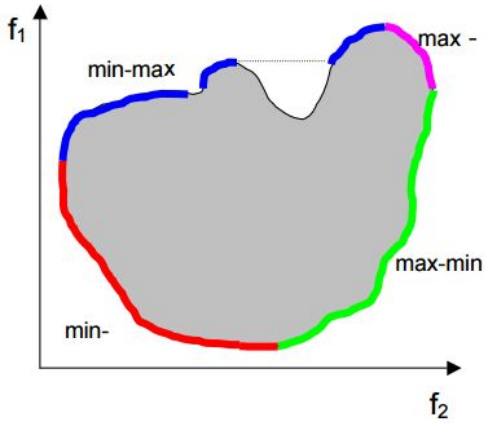
Víceúčelová optimalizace

- Optimalizace dvou a více funkcí
- V obecné formě se jedná o soustavu nerovnic
- Jelikož optimalizujeme vše najednou, výsledkem je množina bodů, které jsou přípustné pro všechny funkce zároveň → množina přípustných řešení
    - tento prostor je zpravidla nesouvislý, může vznikat více izolovaných množin
- dva způsoby jak značit výsledek víceúčelové optimalizace



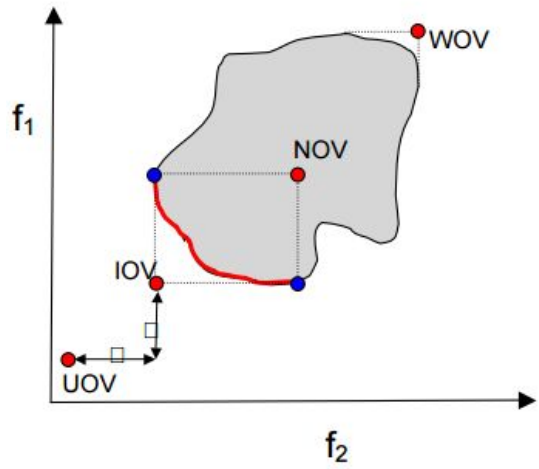
    - **Decision space** = PPK = Prostor přípustných kombinací
        - osy jsou parametry jedince
        - prostor je prostor přípustných řešení pro obě funkce
    - **Objective space** = PHF =Prostor hodnot funkcí
        - osy jsou **hodnoty účelové funkce** pro každou fci zvlášť
        - plocha jsou hodnoty účelové funkce vhodné pro obě optimalizované fce

- **Paretova hranice**



    - **Jedná se o nejlepší možné řešení optimalizovaného problému**
    - Aby vznikla Paretova hranice, musí být hledané funkce v “konfliktu”, jinak se kardinalita Paretovy hranice rovná 1 ⇒ jediné optimální řešení
    - většinou (u minimalizace) je hranice definována body nejvíce vlevo-dole
    - typy a pozice

        - min-min (u obou funkcí hledáme minima) - vlevo dole
        - min-max (u první min u druhé max) - vlevo nahore
        - max-min (u první max u druhé min) - vpravo dole
        - max-max (u první max...i give up) - vpravo nahore
    - vektory
        - IOV - Ideal Objective Vector
            - ideální stav, kterého však nejde dosáhnout
        - NOV - Nadir Objective Vector
            - geometrická inverze IOV, spolu s IOV vyznačuje paretovu množinu (přerušovaný obdélník)
        - UOV - Utopian Objective Vector
            - nositel informace o striktně lepších hodnotách než IOV, vzniká přičtením malé kostanty ε k vektoru IOV
        - WOV - Worst Objective Vector



        -

Účelová funkce, restrikce

- proč? - protože normální metody trvají dlouho
- **Účelová funkce**
    - taková funkce, jejíž optimalizace (nalezení min/max) vede k nalezení optimálních hodnot jejích atributů
    - na každou tuto funkci lze nahlížet jako na plochu v N+1 rozměrném prostoru (N = počet parametrů jedince), přičemž “+1tý argument” je výsledek účelové funkce
    - účelová funkce může být unimodální (jeden globální extrém), nebo multimodální (<2, nekonečně mnoho> stejně kvalitních řešení - extrémů)
    - **tvorba:**
        - jeden z nejkritičtějších bodů
        - nelze říct univerzální postup
- **Restrikce (Omezení) a ošetření krizových stavů**
    - dva typy
        - funkční omezení
            - penalizace hodnot účelové funkce
                - penalizované hodnoty představují fyzicky nesmyslné hodnoty (záporná tloušťka, záporná pravděpodobnost)
                - hard-constraints → nevyhovující oblasti jsou přímo zakázány → jedinec je zrušen a nahrazen jedincem ležícím v povolené oblasti
                - soft-constraints → jedinec je znevýhodněn modifikací účelové funkce

            - výhodou soft oproti hardconst je to, že prostor řešení zůstává souvislý, nevznikají nám nepřirozené lokální extrémy na hranicích. Ne vždy však softconst lze použít.
        - omezení argumentů
            - když překročí, tak se nahradí novým náhodným jedincem
            - ~~nebo se hodnoty posunou na hranici (bad idea)~~
    - **DSH (Práce s celočíselnými a diskrétními hodnotami)**
        - ošetření celočíselných hodnot může být dvojího typu:
            - argument účelové fce (parametr jedince) se zaokrouhlí přímo v populaci → parametry jedince se nahradí
            - zaokrouhlí se pouze na vstupu do účelové funkce, ale jejich parametry zůstavaji puvodni, nezaokrouhlené
        - práce s diskrétními hodnotami probíhá tak, že si je nahradíme za celočíselný index reprezentující jednu diskrétní hodnotu s tím, že při výpočtu účelové fce se nahradí zpátky.
- Restrictions on the utility function - problem formulation.
- Restrictions on individual parameters - parameters during treatment leaving a space of possible solutions.
- Types of penalties and jejch impact on the geometry of the objective function and its continuity.
- Using real, integer and discrete parameters of the individual. Technology DSH.
- Describe creation of objective function

Specifické algoritmy

Formulate and explain the principle algorithms depth-first search, best-first search, greedy algorithm. Formalise mathematical notation, a description of the algorithm in pseudo-code. Provide control and stop the algorithm parameters.

- 404 not found

Formulate and explain the principle of **local search** algorithm method, **blind algorithm**, **climbing algorithm**, **simulated annealing**. The difference between these algorithms. The effect of algorithm parameters on its activities. Describe the selected algorithm in pseudo-code. Provide control and stop the algorithm parameters.

Metoda Lokálního Hledání / Local Search Method / Neighbour Search Method

- stochastický (náhodný) algoritmus
- vybere se první bod(většinou náhodně), vygeneruje se okolí, pokud je v okolí lepší, zaměníme a pokračujeme, pokud ne, ukončíme algoritmus
- **parametry** LS = (M,X0, N, F)
    - M = množina řešení
    - x0 = počáteční bod
    - N = množina sousedů bodu, většinou pomocí odchylky
    - f = objective function, účelová fce
- **vylepšení**
    - čím větší bude prohledávané sousedství, tím se omezí deadlock
    - opakovat LS několikrát pro různé počáteční řešení a zaznamenávat nejlepší řešení
    - připustit i kroky, po nichž dojde ke zhoršení hodnoty účelové funkce → odbočení do jiné oblasti prostoru řešení
    - RLS (repeated local search) , musí se přidat parametr obsahující počet iterací celého algoritmu (t_max) → ukončovací kritérium

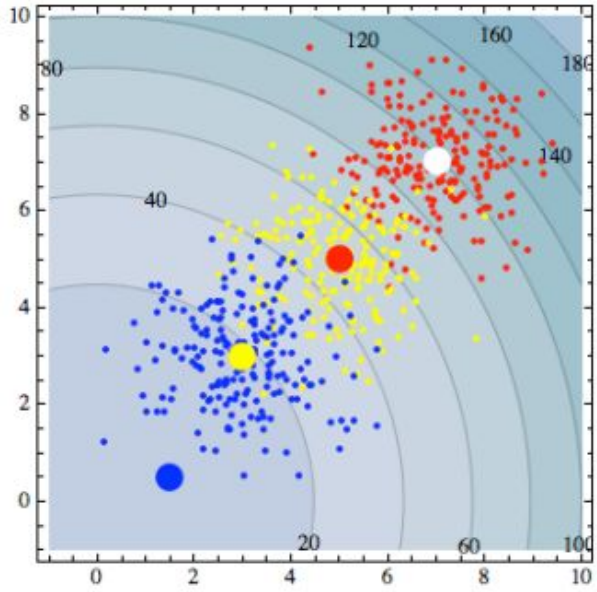
Slepý Algoritmus / Blind Search / Random walk

- stochastický (náhodný) algoritmus
- BS = (M, n, t_max , f)
    - v prezentaci chybí tak si vymýšlím :D
    - M = prohledávaný prostor
    - n = počet generovaných bodů v populaci
    - t_max = počet iterací
    - f = účelová fce

- **algořiťmus**
    1. Vygenerujeme **n náhodných jedinců** dle fce f(M) resp vybereme n náhodných jedinců z M
    2. Vybereme nejlepšího jedince z populace + dosud nejlepšího
    3. Opakujeme dokud nedosáhneme t_max cyklů

Horolezecký Algoritmus / Hill Climbing

- HC = (M, X0, N, F, T_max) - same as RLS
- podobné LS a RLS, ale umožňuje i vzít horší výsledek než je globálně nejlepší nalezený, respektive neporovnává okolí s nejlépe nalezeným
    - navíc se zastaví až po dokončení stanoveního počtu iterací, LS se zastaví, když nenašel lepšího.
- opět problematické u multimodálních fcí
- https://www.youtube.com/watch?v=oSdPmxRCWws
- **algořiťmus:**
    1. vezmeme náhodný bod
    2. vygenerujeme okolí pro tento bod a vezmeme nejlepší z tohoto okolí (bez ohledu na rodiče)
    3. tento bod se stane novým středem a opakujeme dokud t < t_max
- **vylepšení**
    - stochaistic hill climbing - přidá náhodný element do generace sousedů
    - *Velikost **uživatelsky přijatelné odchylky** se může implementovat do algoritmu tak, že např. se sleduje posledních 10 iterací a pokud se hodnota účelové funkce neliší o více, než je nastaveno právě v parametru odchylky, tak se algoritmus zastaví. A to i přes to, že ještě není vyčerpaný zadaný počet iterací.*
    - Climbing with learning - not described how exactly
    - paralelní prohledávání



Simulated Annealing

- stejně jako HC připouští i kroky, po nichž dojde ke zhoršení ⇒ lze tedy na něj opět nahlížet jako rozšíření LS
- od HC se liší tím, že je schopný (alespoň ze začátku) se vyhrabat z lokálního extrému (když je teplota vysoká. šance akceptování horšího řešení se snižuje s tím, jak se snižuje teplota)
- inspirace přišla z metalurgie, konrétně z žíhání kovu - zahřívání a pomalé chlazení kovu
- T = teplota, T0 = startovní teplota, Tf = finální teplota
- α = ochlazovací funkce
    - nejčastější *const* * T
    - *const* je většinou pojmenovaná jako redukční faktor a je většinou 0,8 - 0,99
- nT = počet opakování Metropolis algorithmu, často se volí počet sousedů
- SA = (M, x0, N, f, T0, Tf, α, nT)
- **algořiťmus**
    1. vygenerování náhodného počátečního bodu x0
    2. for → vybrání náhodného souseda počátečního bodu (Metropolis algorithm)
        a. resp pro 1 teplotu projedu vícero sousedy
    3. porovnání hodnoty souseda a počátečního bodu
        a. lepší → zaměníme
        b. **horší → zaměníme pokud rand() < $e^{-\Delta f/T}$ (Metropolisovo kriterium)**
    4. ochladíme a zkontrolujeme porovnáme teploty → končíme nebo opakujeme
- **vylepšení**
    - Parallel Simulated Annealing - začíná se z více bodů
    - Simulated Annealing Alogithm with elitism - uchovává si nejlepší nalezené řešení

Tabu Search

- vylepšený horolezecký algoritmus
- pamatuje si transformace, podle kterých byl spočítan aktuální střed ⇒ nedochází k zacyklení ⇒ neuvízne tak snadno v lokálním extrému
- krátkodobá paměť **TL (tabu list)** - fronta ***k*** předchozích výsledků - ty označuje jako “tabu” nepovoluje ignoruje tyto body, fronta má omezenou délku, pořád se může stát, že se přejde k původním hodnotám
- **k** musí být menší než **θ**, respektive k musí být menší než počet generovaných sousedů
- **aspirační kritérium** - pokud by provedení zakázané transformace zlepšilo hodnotu nejlepšího řešení → povolí ji

- TS = (M, x0, θ, f, t_{max}, TL, k)
    - θ = množina přípustných transf. generující okolí (theta)
    - TL = seznam tabu transformací
    - k = kapacita krátkodobé paměti (size of TL)
- algořiťmus
    1. vygenerování náhodného počátečního bodu
    2. vygenerování okolí kolem počátečního bodu dle θ
    3. pro každý bod z okolí se **zkontroluje** zda-li **je lepší než počáteční bod** a (**jestli není v TL** nebo **jestli není lepší než nejlepší řešení** (asp. kriterium))
    4. nejlepší takto nalezený bod se přidá do TL (v případě překročení *k* se odebere první prvek z *TL* ← fronta)
    5. repeat till t_{max}
- vylepšení
    - adaptace délky zakázaného seznamu (k)
        - malé k ⇒ velká tendence spadnout do lokálního extrému
        - velké k ⇒ hrozí přeskočení nadějných lokálních extrému
    - dlouhodobá paměť - uchovává četnosti použitých transformací
        - čím větší četnost → tím větší penalizace

Formulate and explain the principle of **genetic algorithm**. Terminology GA. Principle of operation, operators GA. Explain the concept of "schemes". Hybrid GA, messy GA, GA parallel, migration and diffusion model. The difference between the versions of the GA. The effect of algorithm parameters on its activities. Describe the selected algorithm in pseudo-code. How are mutations, roulette selection. Provide control and stop the algorithm parameters.

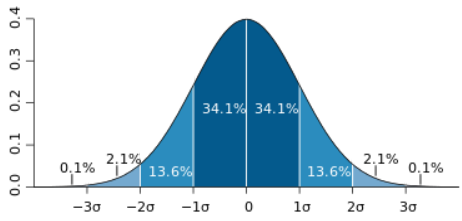
Genetic Algorithms

- patří mezi EVT (**E**volční **v**ýpočetní **t**echniky) → EA != GA
- terminologie
    - **geny** = parametry jedinců (typicky binární podoba)
        - **chromozom** = konkrétní řetězec složený ze všech parametrů = jedinec
    - **genotyp** = sada všech parametrů jedince (genů) e8rgo jedinec?
    - **fenotyp** = vhodnost (fitness)
- **Vhodnost/Fitness**
    - jeden z parametrů jedince → Vhodnost/Fitness (normalizovaná hodnota účelové fce, např. na intervalu [0,1]) ⇒ znázorňuje jakýsi procentuální podíl vhodnosti použití tohoto jedince pro výběr za rodiče
- **Výběr rodičů**
    - Podle Darwinovy evoluční teorie **pouze nejlepší přežívají**
        - na druhou stranu i horší jedinec může přinést vhodné geny pro budoucí vývoj
    - Výběr může probíhat více způsoby, vždy se vybírají dva rodiči, rodičem se jedinec může stát vícekrát
    - Výběr Ruleta
        - všechny fitness dávají dohromady hodnotu 1 → můžeme použít random fci pro výběr konkrétního rodiče, ti s větším fitness budou mít větší šanci
    - Výběr Rank
        - pokud jsou velké rozdíly mezi fitness, je lepší dát nejhoršímu hodnotu 1, lepšímu 2,atp.
        - ve výsledku má zase lepší fitness lepší šanci, ale nedochází k extrémům
- **Elitismus**
    - x nejlepších jedinců z populace se automaticky zkopíruje do nové populace, zamezí se tak jejich ztrátě. Tito jedinci mohou být stále zvoleni za rodiče.
- **Proces / Reprodukce**

    - **Křížení**
        - dojde k “rozseknutí” chromozomu v určitém bodě a prohození částí
        - můžou se použít dva body → vyměňuje se prostřední část
        - může se použít i více bodů = uniformní křížení = náhodně vybrané úseky se prohazují
    - **Mutace**
        - 1 až n náhodně zvolených bitů se invertuje = n-bodová mutace
        - prevence před stagnací v lokálním extrému.
    - Parametry křížení a mutace ⇒ procentuální šance, že k dané změně dojde
        - křížení zpravidla 80 - 95%; mutace 0,5 - 1%
- **Schémata**
    - Kromě číslic 1 a 0 se zavede *, což může znamenat obojí
        - 11*0 =1110 x 1100
    - Hodnota účelové fce = průměr všech možných řetězců
    - pokud najdeme jedince s daným vzorcem
- **Verze**
    - **Hybridní genetický algoritmus**
        - kombinace GA a lokálních optimalizačních technik
        - GA najde vhodnou oblast a lokální prohledávač pak najde extrém
    - **Messy genetický algoritmus**
        - umožňuje práci s různě dlouhými řetězci, každý gen (bit) je tuple → pozice + hodnota
        - může se stát, že budou přeurčené, nebo podurčené
            - přeurčené = na stejné pozici jsou dva různé bity → platí kdo je v řetězci první
            - podurčené = některé pozice jsou prázdné → je uchováván nejlepší řetězec, pozice se doplňují z něj
        - vykazuje dobrý výkon u klamavých multimodálních funkcí
    - **Parallelní genetický algoritmus**
        - několik nezávislých populací najednou, každá v separátním procesu → vyměňují si jedince navzájem
        - 3 metody
            - master-slave
                - jeden proces je master a posílá účelovou funkci slaveům
                - synchronní → hodně čekání, nejjednodušší
            - sub-populace
                - každý proces má vlasní sub-populaci, čas od času si prohodí nejlepší výsledky
            - buněčné GA
                - malé sub-populace (někdy jen jeden jedinec) každý se může integrovat s někým jiným z jiné populace

Evoluční Strategie / Evolution Strategies

- ES se od GA liší:
    - Reprezentace jedinců je **z oboru reálných čísel**
    - ES **nepoužívaly křížení**, **používaly** pouze **selekci** a **mutaci**
- U značení ES se používá syntaxe: “**+**” a “**,**” ⇒ (1+1)-ES; (1,1)-ES
    - **+** ⇒ do nové populace se vybírají nejlepší **z rodičů i potomků**
    - **,** ⇒ do nové populace se vybírají nejlepší **z potomků only**
- (1+1)-ES:
    - nejjednoduší verze ES, pracuje poze s jedním jedincem → rodičem
        - k rodiči se přičte náhodné číslo vygenerované normálním rozdělením **N(0, σ)**



- **σ = směrodatná odchylka**

- Mělo by platit **“slavné” pravidlo jedné pětiny**
  - p_s = poměr úspěšných mutací ke všem mutacím by měl být 1/5
  - úspěšná mutace = potomek má lepší fitness než rodič

$$\sigma^{t+1} = \begin{cases} c_d\sigma^t & jestli \quad p_s < 1/5 \\ \sigma^t / c_d & jestli \quad p_s > 1/5 \\ \sigma^t & jestli \quad p_s = 1/5 \end{cases} \text{ kde } c_d = 0{,}817.$$

- Vícečlená ES: (μ+λ)-ES a (μ, λ)-ES
  - **λ** = populace potomků
  - **μ** = populace rodičů
  - už pracuje s populacemi, ne jen jedním jedincem
  - po vytvoření nových potomků (stejné jako u 1+1 ES), se sjednotí **λ** a **μ**
  - dojde k výběru nejlepších řešení ze sjednocení. pokud obsahuje lepší než stanovenou FV (fitness value, vhodnost, pri kterem ES končí.), tak kaniec filma.
  - **AKA ES s elitismem** ⇒ do nové rodičovské populace jsou vybírani jak rodiče, tak potomci na základě dosažéné fitness.
  - protože **(μ+λ)-ES** na některých problémech stagnovala ⇒ **(μ, λ)-ES AKA ES bez elitismu** (nedochází ke sjednocení → vybírano pouze z potomků)
  - existují různé kombinace, většinou s využití další proměnné - life span, který dovolí použít rodiče, ale tento rodič má omezený life span a později z populace vymizí
- Rekombinační a Adaptivní ES
  - další dvě ES
  - Rekombinační - před vlastní mutací je vytvořen rekombinant z více rodičů - takže prakticky zde dochází ke “křížení”
    - parametr ρ (rho) udává počet rodičů
    - průměrová rekombinace (vezme se průměr atributů vybraných rodičů)
    - diskrétní rekombinace (parametr náhodně vybraný z jednoho z vybraných rodičů)
  - Adaptivní ES
    - dochází k adaptaci směrodatné odchylky mezi iteracemi
    - existují tři různé adaptace - izotropní, neizotropní a korelační

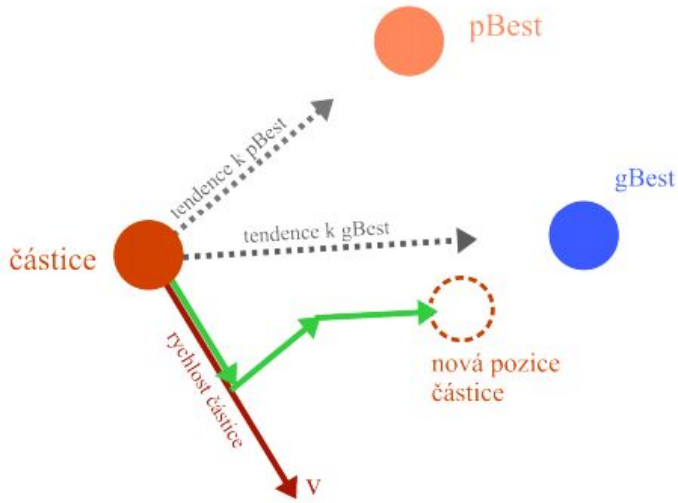
Rojení částic / Particle Swarm Optimization / PSO

- inspirováno pohybem ptačích/rybích hejn
- **nemá žádné evoluční operátory (křížení, mutace)**
- potenciální řešení (jedinci) následují v řešeném prostoru trajektorie těch částic, která mají nejlepší fitness

- **algořiťmus**

1. PSO je inicializován **náhodně vygenerovanou populací**
2. Každému jedinci je vygenerován **vektor rychlosti ⇒ udává směr**, kterým se jedinec v příštím kroku chce vydat
3. Z pozice jedinců je vypočítána hodnota účelové funkce → jedinec s nejlepší hodnotou uloží tuto pozici do společné paměti populace, takže každý jedinec ví, kde se nachází doposud nejlepší nalezené řešení (**gBest**), zároveň si každý jedinec pamatuje svou nejlepší pozici (**pBest**) [hádám globalBest, personalBest]
4. Dle kombinace **vektoru rychlosti, pBest a gBest** vzniká nový vektor rychlosti
    a. **vNow + c_p * rand * (pBest - pNow) + c_g * rand * (gBest - pNow)**
    b. c = učící faktory, pNow = momentální pozice, v = momentální v rychlosti
5. Dle nového vektoru rychlosti se vypočte nová pozice
    a. pNow + vNewVektor
6. vypočte se nové fitness a upraví se pBest popř gBest

- podle **pBest** a **gBest** pak má jedinec tendenci postupovat třemi směry:
    - **Individuálně** → pokračuje svou vlastní cestou (podle vektoru rychlosti)Tfa
    - **Konzervativně** → vrací se na svou dosud nejlepší pozici (k pBest)
    - **Přizpůsobivě** → následuje jedince s nejlepším řešením (k gBest)



skutečný pohyb

- **nedostatky:**
    - Vygenerovaná příliš velká rychlost jedinců → ti se pak vzdalují od nejlepšího dosavadního řešení ⇒ vyřešeno zavedením parametru **V_max** (max rychlost)
    - U multimodálních problémů má sklon k předčasné konvergenci
    - Mnoho nastavitelných parametrů:
        - Dimenze → daná problémem, který optimalizujeme
        - Rozsah → velikost prohledávaného problému
        - Počet částic → čím víc částic, tím hustěji bude prostor prohledán (good), ale výpočetní náročnost narůstá a bude větší čas pro nalezení řešení (bad) ⇒ doporučený počet je **10*D**
        - Vmax → maximální rychlost částic.
            - příliš malá hodnota ⇒ prohledávají důkladněji, ale malou oblast
            - příliš vysoká hodnota ⇒ moc se vzdalují, překračují meze → stává se náhodným (při překročení meze se totiž vygeneruje nová pozice pro danou částici)
        - Učící faktor c1 a c2 → ovlivňují který směr následovat (původní směr (**c1**), pBest, gBest (**c2**)) okořeněno o rand(), aby to nebylo tak jisté…
- **Vylepšení PSO**
    - **Sousedství** → částice jsou rozdělené do skupin (sousedství)
        - Geografické sousedství
            - částice se nacházejí ve stejné oblasti

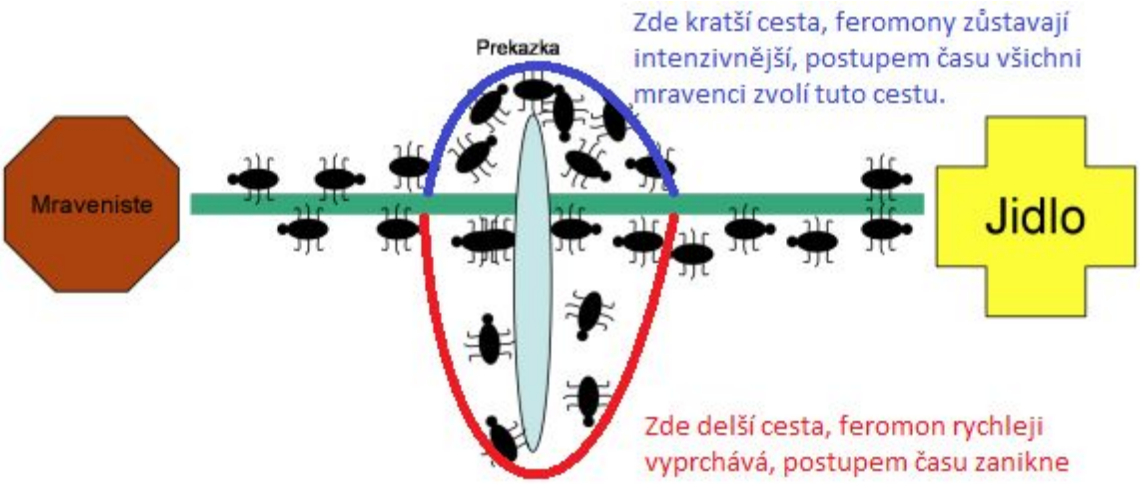
        - problém definice hranic a vzdáleností, kdy jedinec patří do nějakého sousedství a kdy už do jiného
    - Sociální sousedství
        - nejčasteji používané → jedinci jsou sousedi nehledě na to, kde se nacházejí → o tom rozhoduje pořadové číslo jedince

Rozptýlené hledání = Scatter Search = SS; Path relinking (PR)

- nové řešení vzniká lineární rekombinací dvou starších
- nejprve se pomocí heuristického procesu najdou nejlepší řešení (referenční množina)
- pak se lineární rekombinací vytvoří nové body
- nejlepší body se stanou novou referenční množinou
- algoritmus končí, jestliže nedojde v iteraci ke změně

Optimalizace mravenčí kolonií (Ant Colony Optimization)

- klasifikace jako: hledání dobré cesty grafem
- inspirováno mravenci v přírodě → random se pohybují, eventuálně najdou potravu, po cestě zpátky do mraveniště vypouštějí feromony → ostatní mravenci pak ví, kde je žrádlo → opět vypouštějí feromony, cesta je intenzivněji značkována → nejčastěji navštěvovaná cesta má nejvíc feromonové stopy, u ostatních cest feromon vyprchává (zamezení konvergence do lokálního optima)
- využití: obchodní cestující, směrovací tabulky (routování)



- **Varianty ACO:**
    - S elitismem → globální nejlepší řešení nanáší feromony spolu s ostatními mravenci
    - Max-Min Ant System (MMAS) → přidává omezení (min,max) množství feromonu, pouze globální nejlepší nebo iterační nejlepší řešení přidává feromony

SOMA (SamoOrganizující se Migrační Algoritmus)

- inteligentní jedinci, kteří kooperují na řešení společného úkolu → **soutěživě-kooperativní**
- **neprobíhá tvorba nových jedinců pomocí křížení**, ale je založena na prohledávání (migraci) prostoru → neoznačujeme iterace jako generace, ale "**migrační kolo**" (zase inspirace od mravenců, predátorů a jiných…)
- jedinci se navzájem ovlivňují během hledání nejlepšího řešení ⇒ mnohdy to vede ke vzniku skupin jedinců → ty se rozpadají a zase spojují a tak putují přes prohledávaný prostor ⇒ **SamoOrganizace**
- je závislý na správnem nastavení vstupních parametrů
- **Vstupní parametry**

- **Path Length** → určuje jak daleko se aktivní jedinec zastaví od vedoucího jedince (při 1 se zastaví na jeho pozici, při 2 se zastaví za ním, ve stejné vzdálenosti, kde startoval, pří <1 se zastaví před ním a dochazí tak k degradaci migračního procesu ⇒ konvergence k lokálním optimům). Doporučená hodnota je 3
- **Step** → jakási zrnitost, s jako bude mapována cesta aktivního jedince. v případe unimodální funkce je možné použít velkou hodnotu pro urychlení chodu algoritmu. Jestliže není známa geometrie účelové funkce, tak se doporučuje nastavit na menší hodnoty → prostor možných řešení pak bude prohledáván důkladněji. Doporučuje se nastavit aby nebyla celočíselným násobkem vzdálenosti aktivního-vedoucího jedince (0.11 je lepší než 0.1), jinak bude docházet k poklesu diverzibility populace
- **PRT [0,1]** → znamená perturbaci. podle tohoto parametru se pak tvoří PRTVector (perturbační vektor). Tento vektor určuje, zda-li se aktivní jedinec bude pohybovat směrem k vedoucímu jedinci či ne. **Jeden z nejdůležitějších parametrů s největší citlivostí!** Optimální hodnota je 0.1, při příliš vysokých hodnotách dochází ke konvergenci do lokálních optim.
- **D** → dimenze problému, počet argumentů účelové funkce
- **PopSize** → počet jedinců v populaci. Ideálka 10+, lze i 2 ale pak to bude čistě deterministické
- **Migrace** → ukončující parametr. Kolikrát proběhne migrace.
- **MinDiv** → další ukončovací parametr - maximální rozdíl mezi nejlepším a nejhorším jedincem. jestli je rozdíl < MinDiv, pak je alg ukončen. Else pokračuj.

- **Mutace**
  - v případě SOMA sa to volá perturbacia → při pohybu prostorem je pohyb jedince náhodně rušen (v nějakém směru)
  - pro každý parametr jedince se generuje náhodné číslo [0, 1] a porovnává se s PRT parametrem. if (rand < PRT) tak 1, else 0

rnd_j	PRTVector
0,231 < 0,3	= 1
0,456 > 0,3	= 0
0,671 > 0,3	= 0
0,119 < 0,3	= 1

$PRT = 0,3.$

  - je generován před každým posunem/skokem jedince, tím tak modifikuje jeho směr
  - Křížení je tedy nahrazeno u SOMA putováním jedince:

```
//vypocitat novy smer pro aktivniho jedince -> diff_points.x/y/z[j] == aktivni jedinec
var nove_pozice = [];
for (var t = 0; t < pathLength; t += step) {
    var new_x = diff_points.x[j] + (leader.x - diff_points.x[j]) * t * PRTVector.x;
    var new_y = diff_points.y[j] + (leader.y - diff_points.y[j]) * t * PRTVector.y;
    var bod = {
        x: new_x,
        y: new_y,
        z: getZAxis(new_x, new_y)
    };
    nove_pozice.push(bod);
}
```

    při PRTVector 1 je pohyb směrem k vedoucímu jedinci, při 0 je pohyb “zmražen” → díky zmražení se odkloní od Leadera a prohledává část prostoru, která by normálně prohledána nebyla → zvyšuje se diverzibilita populace.

- pokud v nove_pozice najdu lepší jedince, nahradím je:

```
//pokud naleznu lepsi, tak ho nahradim v ty generaci
for (var l = 0; l < nove_pozice.length; l++) {
    if (nove_pozice[l].z < diff_points.z[j]) {
        if (verify(nove_pozice[l].x, nove_pozice[l].y, global_from, global_to)) {
            diff_points.x[j] = nove_pozice[l].x;
            diff_points.y[j] = nove_pozice[l].y;
            diff_points.z[j] = nove_pozice[l].z;
        }
    }
}
```

- **Varianty**
  - **All-to-One** → klasika popsana nahoře, vsichni migrují k leaderovi
  - **All-to-All** → Každý zkusí migrovat ke každému, pamatuje si nejlepší pozici a pak se vrátí na svůj začátek, to samé udělají i ostatní, až všichni zjistí svou nejlepší pozici, dojde k přesunu - ez, trvá to déle a je to těžší, ale daleko spolehlivější :)
  - **All-to-all-adaptive -** zkusí migrovat ke každému, jakmile najde lepší pozici, přesune se a zkouší dál, jakmile zkusí všechny kombinace; migrace tohoto se tedy projevuje ve všech dalších migracích
  - **AllToOneRand** - Leader je určen náhodně, jinak to samé co AllToOne
  - **Svazky -** není to strategie sama o sobě, ale dá se použít k jednom z těch předchozích 4
    - pomocí takového škaredého vzorečku se rozdělí jedinci do subpopulací (svazků)
      - faktorem je “vzdálenost” - připomínají clustery z MAD, jedinec co je daleko od ostatních je svazek sám sobě
    - přesouvají se asi celé svazky dle variant výše
    - výpočetně náročné a zlepšení je nic moc, tak se to moc nepoužívá a není třeba dále dopodrobna vysvětlovat

Differential Evolution

- činnost a kvalita DE stejně jako u jiných EAs je ovlivněna jejími řídícími parametry
- zvláštnosti:
  - **tvorba potomků pomocí 4 rodičů (v jiných variantách DE i více)**
  - **prve mutace, pak křížení**
- **řídící parametry:**
  - **CR [0,1]** → práh křížení.
    pokud je funkce separabilní, volit **blíže** k 0.
    v opačném případě volit **blíže** k 1.
    Pokud CR = 0, mutace se nedostane do zkušebního jedince ⇒ bude čistou kopií čtvrtého rodiče.
    Pokud CR = 1, bude zkušební jedinec tvořen pouze ze 3 náhodně vybraných rodičů → DE se bude podobat náhodnému hledání.
    ⇒ CR by tedy nikdy nemělo být přesne 1 nebo 0.
  - **D** → dimenze problému, opět počet argumentů účelové funkce. dán řešeným problémem
  - **NP** → velikost populace. Neměl by být menší než 4. ideálka z [10D,100D]
  - **F** [0,2] → mutační konstanta
  - **Generace** → počet evolučních cyklů.
- **Mutace**
  - Pro každého jedince jsou náhodně vybrání 3 další nestejní jedinci (r1, r2, r3).
  - Pomocí těchto 3 jedinců se vytvoří tzv. šumový vektor. (konkrétně teda: **r3 + F * (r1 - r2)**)
- **Křížení**

- ze čtvrtého, doposud nepoužitého jedince **(r4) a šumového vektoru se vytvoří zkušební vektor**, ten se vytváří za pomocí toho CRka (opět hod kostkou).

```
var r0 = generace[j];
var r1 = generace[random3points[0]];
var r2 = generace[random3points[1]];
var r3 = generace[random3points[2]];

//SUM VEKTOR
var sum_v = {
    x: r3.x + F * (r1.x - r2.x),
    y: r3.y + F * (r1.y - r2.y)
};

//TRIAL POINT
var trial_v = {
    x: r0.x,
    y: r0.y,
    z: 0
};

if (Math.random() < CR) {
    trial_v.x = sum_v.x
}
if (Math.random() < CR) {
    trial_v.y = sum_v.y
}
trial_v.z = getZAxis(trial_v.x, trial_v.y);
```

šumový vektor

váhový diferenční vektor

diferenční vektor

- DE je ukončena pouze tehdy, provede-li se zadaný počet generací (alespoň v základní verzi algoritmu). Lze ale nakodit cokoliv → pokud se nejlepší jedinec nezměnil za posledních X generací → ukonči nebo změn řídící parametry.
- Během každé iterace (generace) se uschová hodnota účelové funkce nejlepšího jedince. Ty pak lze zanést do grafu a pozorovat průbeh evolučního procesu.
- **Stagnace:**
    - populace zůstává diverzibilní, hodnota účelové funkce se ovšem nemění (na rozdíl od klasické konvergence k lokálnímu optimu, kde i populace “stagnuje”)

10. Explain the principle of genetic programming, grammatical evolution. indicate alternatives: analytical programming, probabilistic incremental program evolution - PIPE, gene expression programming, multiexpression programming and more. Explain the symbolic representation of the solution in the form of a population of individuals. Evolutionary hardware - basic idea.

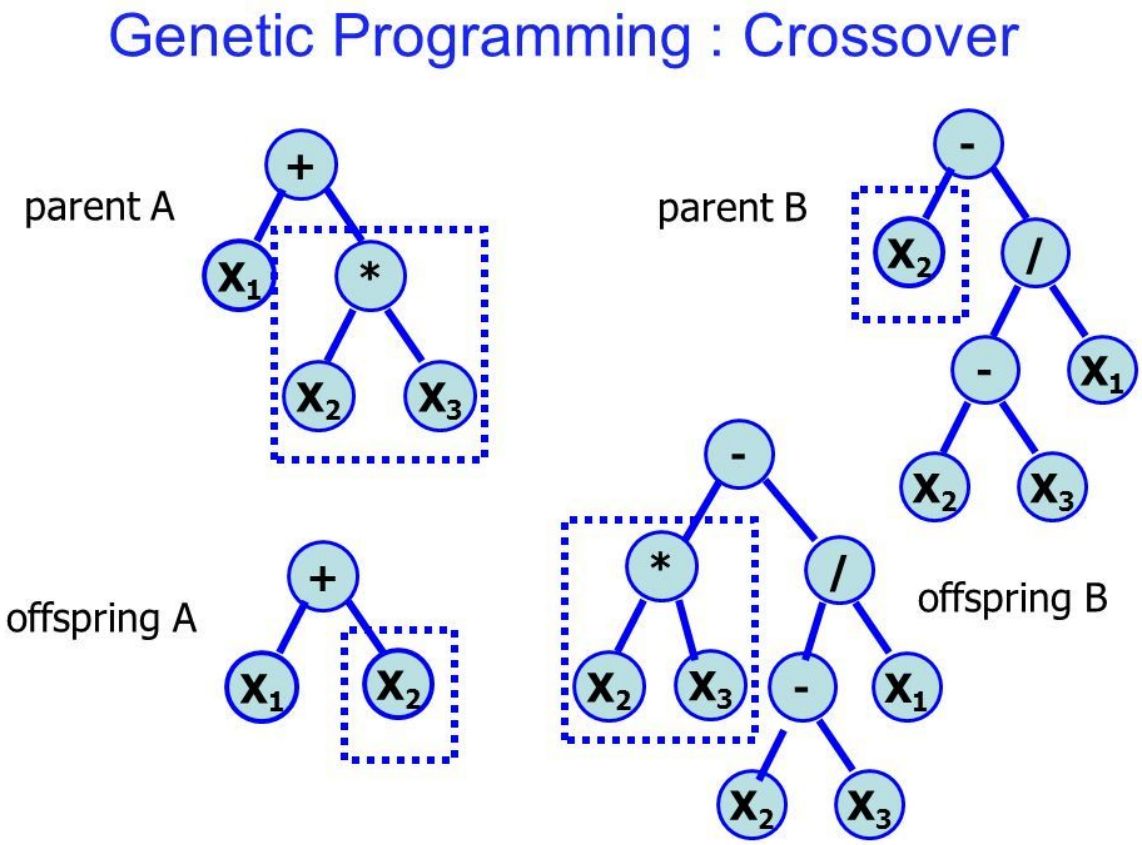
Symbolická regrese

- = skládání složitější struktury z simple elementů

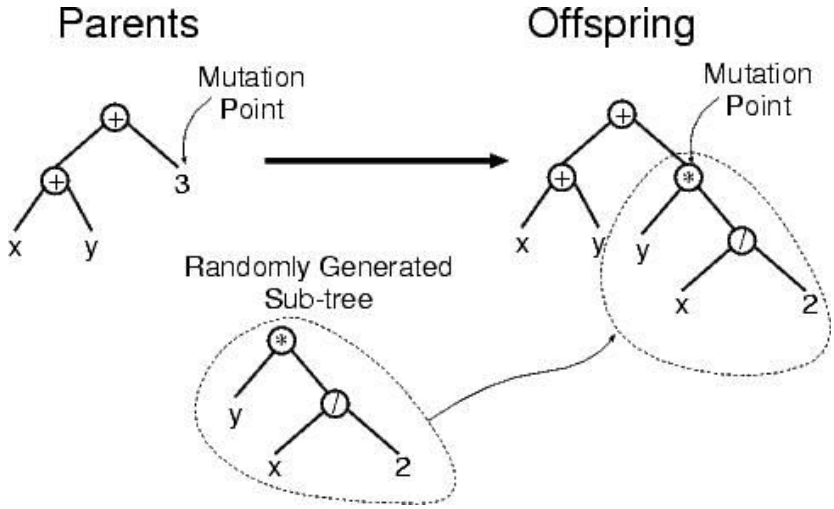
Genetické programování

- modifikace GA → nepracuje s čísly, ale se symboly(matematické fce, uživatelské programy)
- stejné názvosloví jako GA → gen, chromozom, fenotyp, genotyp
- pro snadnější zobrazení se používá stromová reprezentace
- křížení

    - náhodně se zvolí uzel na obou rodičích (body křížení)
    - tyto uzly a tedy i jeho child uzly se prohodí



- mutace
    - na náhodně zvolený bod mutace se připojí náhodně vygenerovaný řetězec



- stejně jako v GA zde existuje pravděpodobnost mutace a pravděpodobnost křížení, navíc existuje pravděpodobnost výběru uzlu pro mutaci
- **Bloat**
    - během evoluce dochází k lineárnímu nárustu délky řetězců (potomků)
    - je možné tomu zamezit penalizací dlouhých řešení(, uzpůsobení mutace a křížení, nebo použít vícekriteriální optimalizaci (?))

Gramatická evoluce

- může být chápána jako typ Genetického programování založeného na gramatice
- používá BNF = Backus-Naurova forma

    - gramatika definovaná zakladateli jazyka ALGOL
    - čtveřice {N,T,P,S}
    - N = konečná množina neterminálních symbolů (+,-,sin,log,...)
    - T = konečná množina terminálů (x,y,1,2,10,...)
    - S = počáteční symbol (z množiny N)
    - P = množina přepisovacích pravidel
        - pravidla v syntaxi Algolu
        - <symbol> ::= <moznost1> | <moznost2>
            - <symbol> je prakticky název
            - <moznost1> jsou retezce terminalu a neterminalu oddelene "|"
        - např <digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
        - <unisigned int> ::= <digit> | <unsigned int><digit>
        - etc.
- ve výsledku se to chová jako klasická bezkontextová gramatika
- jedinec pro gramatickou evoluci je tvořen 8mi bitovými integery (kodony) a vypadá

220	40	16	203	101	53	202	203	102	55	220	202	19	130	37	202	203	32	39	202	203	102

- tj 220,40,16,203,101,53,...
- tito číselní jedinci reprezentují při dané gramatice konkrétní řetězec složený z N a T
- mějme gramatiku následující

N = {expr, op, pre_op, var}
T = {*sin*, + , - , / , * , *x* , 1}}
S = expr

Následně jsou pak rozepsány jednotlivé možnosti substitucí – pravidla gramatiky podle Backus Naurovy formy:

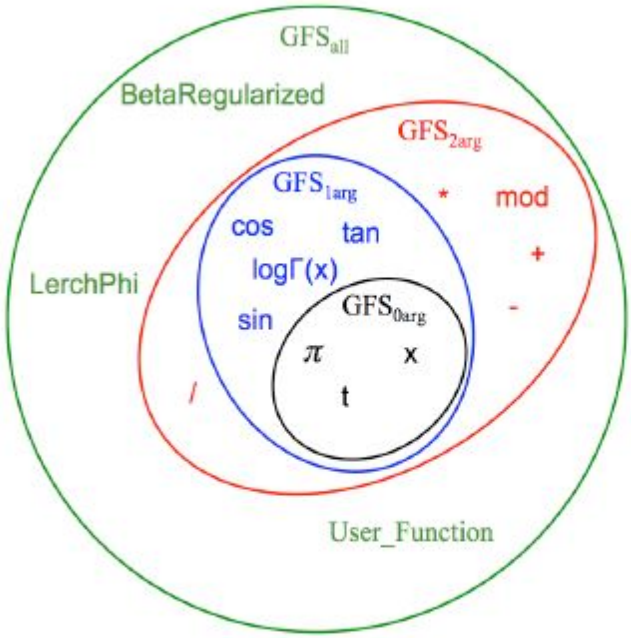
A)	<expr> : :	=	<expr> <op> <expr>	(0)
		|	(<expr> <op> <expr>)(1)	
		|	<pre-op> (<expr>)	(2)
		|	<var>	(3)
B)	<op> : :	=	+	(0)
		|	-	(1)
		|	/	(2)
		|	*	(3)
C)	<pre-op> : :	=	Sin	(0)
D)	<var> : :	=	*x*	(0)
		|	1	(1)

- jedinec teda bude to co jsem definoval výše → 220,40,16,203 dál nejedu
- začínáme od S, tedy na začátku máme expr
- přepis funguje tak, že vezmeme aktuální kodon (220) a provedeme modulo (%) a počet možných kombinací pro daný neterminál (expr má 4 možnosti)
    - 220 % 4 = 0 → vezmeme nultou možnost → <expr><op><expr>
- pokračujeme číslem 40, nahrazujeme postupně neterminály **zleva**
- první zleva je zase expr → 40 % 4 = 0 → <expr><op><expr><op><expr>
- další je zase to samé → <expr><op><expr><op><expr><op><expr>
- další je 203 → 203 % 4 je 3 → <var><op><expr><op><expr><op><expr>
- a tak dále… v případě, že narazíme na neterminál co má pouze jednu možnost, tak ji automaticky vezmeme (nepoužíváme kodon), tzn když budeme mít <pre-op> ihned to změníme na Sin
- pokud by se stalo že nám zbudou neterminály a nemáme další kodony, jedeme od začátku → **operaci provádíme tak dlouho, dokud nemáme samé terminály**
- v tomto případě nám po nějaké době vznikne 1 – sin(x)*sin(x) - sin(x)*sin(x)

- Účelová funkce je v případě gramatické evoluce rozdíl téhle vygenerované funkce oproti funkci žádoucí (hádám že dosadíme konkrétní x do obou funkcí a porovnáme?)
- Mutace i Kombinace probíhá stejně jako u Genetického Algoritmu (tohle jsou osmibitové integery → jde to převézt do binárky)

Analytické programování

- experimentální metoda, nevztahuje se k žádnému konkrétnímu algoritmu, či reprezentaci
- můžeme to chápat jako alternativní přístup k GP a GE
- nemyslím si že jsme ji brali, ale zkusím ji popsat
- je založeno na DSH (diskrétní hodnoty)
- pracuje se s funkcemi, operátory a terminály
- ty se seřadí podle počtu jejich atributů (tzn terminály 0, operátory 2, funkce 1-x)
- v praxi se to všechno hodí do speciální množiny zvané GFS - general function set
    - každá podmnožina pak obsahuje prvky se stejným počtem atributů



Obr. 9.7 Symbolicky znázorněná struktura množiny GFS.

    -
    - tady je GFS sjednocením 4 nezávislých podmnožin (obrázek je trochu nic moc tak sry)
    - všechny prvky z tohoto GFS jsou převedeny na indexy pomocí DFS
    - dejme tomu že máme jedince 1,6,7,8,9,9
    - GFS = {+,-,/,^,d/dt,sin, cos, tan, t,...}
    - 1 tedy odpovídá +, + potřebue dva argumenty a těmi jsou 6 a 7
    - máme tedy sin()+cos() → sin potřebuje jeden argument tak vezme 8, cos taky tak vezme 9
    - máme sin(cos())+cos(t)), další int v řadě je 9 = t; dosadíme t a máme finální funkci
-