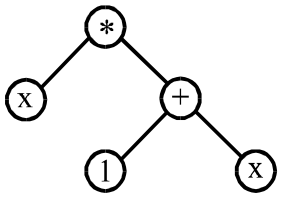
5. kapitola

Genetické programovanie (GP)

5.1 Úvodné poznámky

Na prelome 80-tých a 90-tých rokov americký informatik John Koza [1,2] (Stanford University) navrhol originálnu modifikáciu genetického algoritmu, ktorú nazval *genetické programovanie*. V tomto prístupe sú chromozómy - znakové reťazce nahradené zložitejšími štruktúrami - funkciami. Čo sa rozumie pod pojmom "funkcia"? V najjednoduchšej verzii genetického programovania sa funkcie rovnajú výrazom obsahujúcim premenné, konštanty, základné aritmetické operácie a elementárne funkcie. Jednoduchá funkcia "$x*(1+x)$" je reprezentovaná pomocou syntaktického stromu (parse tree)



kde horný vrchol (priradený operácii násobenia) sa nazýva koreň stromu. Interpretácia tohto syntaktického stromu je jednoduchá, postupujúc zdola-nahor, od koncových vrcholov ku koreňu stromu, postupne sa vykonáva výpočet funkcie "$x*(1+x)$". Prvý problém, ktorý musíme riešiť v rámci genetického programovania, je spôsob vyhodnotenia chromozómu - stromu. Toto vyhodnotenie je formálne chápané ako funkcia $y=t(x)$ priraďujúca nezávislej premennej (vstup) x závislú premennú (výstup) y. V našom príklade je táto funkcia určená výrazom $t(x)=x*(1+x)$. Nech $A=\{(x_i,y_i);\ i=1,2,...,p\}$ je tréningová množina obsahujúca p bodov (x_i,y_i). Naším cieľom je nájsť takú funkciu $t(x)$ (reprezentovanú syntaktickým stromom), ktorá minimalizuje rozdiel (jeho kvadrát alebo absolútnu hodnotu) vypočítaných a zadaných hodnôt y z tréningovej množiny. V prípade, že je tento súčet nulový, funkcia $t(x)$ presne vystihuje body z tréningovej množiny. Nájsť takú funkciu sa nám najskôr nepodarí, no môžeme sa pokúsiť nájsť takú funkciu, pre ktorú bude tento súčet čo najmenší. Hľadanú funkciu vyjadrujeme pomocou syntaktických stromov obsahujúcich predpísané typy vrcholov. Takto formulovaná úloha je veľmi blízka regresnej analýze, kde pre danú tréningovú množinu bodov a pre daný typ funkcie hľadáme také parametre - koeficienty funkcie, ktoré minimalizujú účelovú funkciu. V tomto prípade je "priestor" funkcií podstatne ohraničený, ich tvar sa mení len v dôsledku zmeny parametrov funkcie, ktorej "funkcionálny" tvar je fixný. John Koza nazval takýto zovšeobecnený prístup k regresii *symbolická regresia.*

Symbolická regresia je len jedna možná interpretácia genetického programovania. Iný pohľad na genetické programovanie je chápanie funkcie $t(x)$ ako "programu" pre správanie sa nejakého objektu v prostredí. Tak napríklad, pohyb robota v prostredí môže byť určený funkciou $y=t(x)$, kde x sú vstupné informácie (napr. z TV kamery) o jeho blízkom okolí a y sú príkazy pre jeho motorické zariadenie. Naším cieľom je zostrojiť takú funkciu $t(x)$, ktorá čo najlepšie popisuje pohyb robota (umelého živočícha) a jeho úspešnosť v prostredí. V tomto prípade už nemôžeme hovoriť o tréningovej množine A ako o množine bodov vopred zadanej. Úspešnosť danej funkcie $t(x)$ je daná úpešnosťou robota v prostredí po určitom vopred danom počte krokov (napr. počtom kúskov potravy, ktoré nazbieral pri pohybe v prostredí po n krokoch). Vyššie uvedený postup pre ohodnotenie funkcie $f(x)$ je vhodný pre riešenie širokej triedy problémov adaptácie pomocou genetického programovania. Môžeme teda konštatovať,

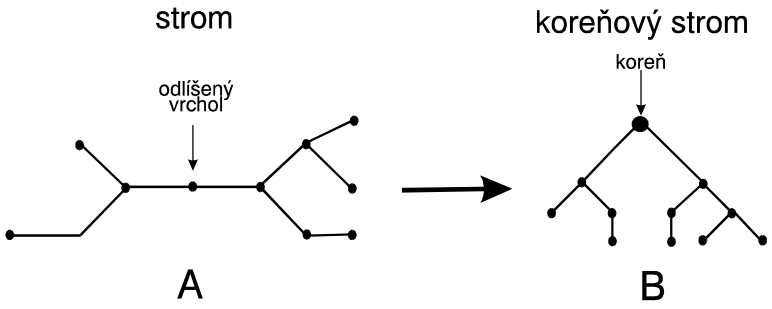
že prístup genetického programovania je vhodný k riešeniu dvoch tried problémov, a to symbolickej regresie a adaptácie (alebo učenia).

Pre úspešnú a hlavne efektívnu implementáciu genetického programovania hrá významnú úlohu metóda kódovania stromových štruktúr. John Koza implementoval genetické programovanie v jazyku LISP (funkcionálny logicky orientovaný jazyk), ktorý má silné programovacie prostriedky pre kódovanie stromov a manipuláciu s nimi. Napriek jednoduchosti napísaných programov v LISP-e, ich efektívnosť je neobyčajne malá, reálne aplikácie vyžadujú pracovné stanice riscovského typu. Na prvý pohľad zdá sa, že genetické programovanie je tak zložitá aplikácia, ktorá a-priori vyžaduje pracovné stanice riscovského typu a ktorá na počítačoch triedy PC nie je (alebo len veľmi obtiažne) realizovateľná. V ďalšej časti tejto kapitole ukážeme jednoduchý spôsob kódovania stromových štruktúr pomocou tzv. Readovho lineárneho kódu a naznačíme jeho implementáciu v pseudopascale. Jeho výhodou pred funkcionálnymi jazykmi je, že umožňuje implementovať genetické programovanie pomocou štandardných procedurálnych jazykov (Pascal, C/C++, Fortran, atď.) na počítačoch triedy PC, pričom výsledný kód je dostatočne efektívny, takže aj netriviálne aplikácia sú realizovateľné.

5.2 Koreňové stromy a Readov lineárny kód

Nech ***G***=(*V*,*E*) je *strom* (súvislý acyklický graf [3]), kde $V=\{v_1,v_2,...,v_p\}$ je neprázdna *vrcholová množina* a $E=\{e_1,e_2,...,e_q\}$ je *hranová množina*, pozri obr. 5.1. Kardinality týchto dvoch množín sú navzájom vztiahnuté formulou

$$|V| = |E| + 1 \tag{5.1}$$



Obrázok 5.1 . Diagramatická reprezentácia *stromu* (A) a *koreňového stromu* (B). Ak jeden vrchol v strome je odlíšený od ostatných vrcholov, potom strom sa nazýva *koreňový strom* a odlíšený vrchol sa nazýva *koreň*.

Nech $v \in V$ je vrchol stromu ***G***, valentnosť tohto vrcholu, označená *val*(*v*), je nezáporné celé číslo, ktoré určuje počet hrán incidentných s vrcholom *v*. Suma valencií všetkých vrcholov spĺňa podmienku [3]

$$\sum_{v \in V} val(v) = 2q = 2(p-1) \tag{5.2}$$

Koreňový strom je strom, ktorý ma jeden vrchol špeciálne odlíšený od ostatných vrcholov, tento vrchol sa nazýva *koreň* (pozri obr. 5.1, diagram B). Koreňový strom je formálne určený ako usporiadaná trojica

$$\boldsymbol{T} = (V, E, v) \tag{5.3}$$

kde $v \in V$ je koreň. Dva koreňové stromy $\boldsymbol{T} = (V, E, v)$ a $\boldsymbol{T}' = (V', E', v')$ sú izomorfné [3] vtedy a len vtedy, ak existuje také 1-1 zobrazenie $\phi: V \to V'$ ($\boldsymbol{T} \approx \boldsymbol{T}'$) , ktoré zachováva susednosť vrcholov a zobrazuje koreň stromu ***T*** na koreň stromu ***T',*** t.j. $\phi(v)=v'$.

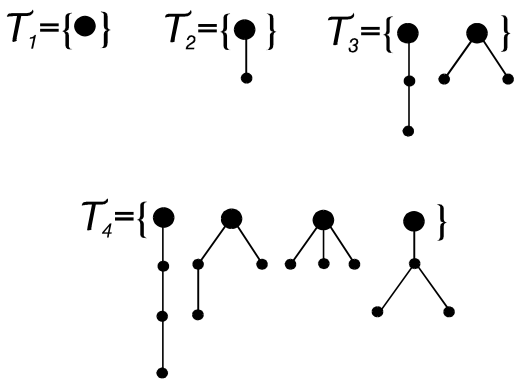
Príklad 5.1*. Dokážte vzťahy (5.1.) a (5.2).*

Nech T_i je množina obsahujúca všetky možné koreňové stromy, ktoré sú zložené z *i* vrcholov, pozri obr. 5.2. Zjednotenie týchto množín

$$T = T_1 \cup T_2 \cup \ldots \tag{5.4}$$

obsahuje všetky možné neizomorfné koreňové stromy. Nech koreňový strom z $\mathcal{T}$ je ohodnotený reálnym číslom

$$t: \mathcal{T} \to R \tag{5.5}$$



Obrázok 5.2. Prvé štyri množiny $\mathcal{T}_1$-$\mathcal{T}_4$, ktoré obsahujú všetky možné neizomorfné koreňové stromy s počtom vrcholov 1 až 4.

Táto funkcia ohodnotí každý koreňový strom reálnym číslom, ktoré nazývame *index*, vyjadruje v určitom priblížení "topológiu" koreňového stromu. Požadovaná hodnota indexu nech je označená t_{req} , potom môžeme definovať účelovú funkciu takto

$$f(\boldsymbol{T}) = |t(\boldsymbol{T}) - t_{req}| \tag{5.6}$$

Účelová funkcia určuje odchýlku vypočítaného indexu $t(\boldsymbol{T})$ od požadovanej hodnoty t_{req} . Ak je hodnota účelovej funkcie nulová, potom index $t(\boldsymbol{T})$ je rovný požadovanej hodnote . Nulová hodnota účelovej funkcie odpovedá globálnemu minimu nad priestorom všetkých možných koreňových stromov, odpovedajúci koreňový strom ako riešenie nasledujúceho minimalizačného problému

$$\boldsymbol{T}_{opt} = \arg \min_{\boldsymbol{T} \in \mathcal{T}} f(\boldsymbol{T}) \tag{5.7}$$

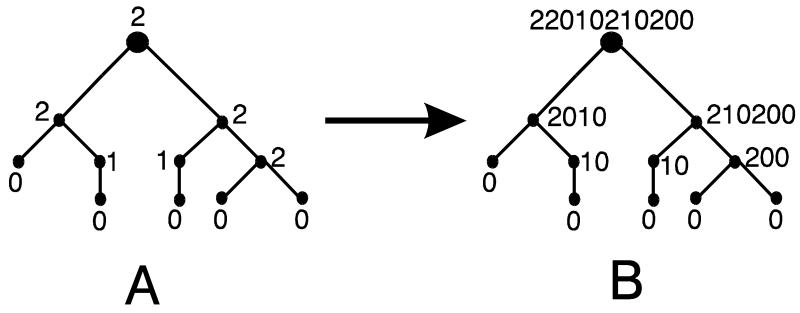
kde výsledný koreňový strom $\boldsymbol{T}_{opt}$ odpovedá takému koreňovému stromu, ktorý minimalizuje účelovú funkciu nad celým priestorom koreňových stromov $\mathcal{T}$.

Metóda Readovho lineárneho kódovanie koreňových stromov je často používaná v teórii grafov pre konštruktívnu enumeráciu stromových štruktúr [4,5]. V poslednej dobe bola tiež použitá pre kódovanie stromov pre potreby takej ich konštrukcie, aby mali požadované vlastnosti [6].

Readov kód pre koreňové stromy je reťazec (sekvencia) celých čísel, ktoré sú priradené buď valencii koreňa alebo valencii zníženej o jednotku pre ostatné vrcholy (pozri obr. 5.3). Lineárny kód koreňového stromu $\boldsymbol{T}$ bude označený code($\boldsymbol{T}$); ak strom $\boldsymbol{T}$ obsahuje jeden vrchol (pozri koreňový strom na obr. 3 patriaci do množiny $\mathcal{T}_1$) , potom jeho kód je code($\boldsymbol{T}$)= '0'. Študujme koreňový strom obsahujúci dva alebo viac vrcholov a predpokladajme, že jeho koreň je incidentný s r hranami, odstránením koreňa zo stromu dostaneme tzv. stromy prvej generácie. Tieto stromy prvej generácie sú znovu uvažované ako koreňové stromy, koreň je identifikovaný s vrcholom, ktorý bol susedný s koreňom pôvodného stromu. Nech vzniklé koreňové stromy sú označené $\boldsymbol{T}_1$, $\boldsymbol{T}_2$, ..., $\boldsymbol{T}_r$, potom Readov kód pôvodného stromu $\boldsymbol{T}$ je definovaný vzťahom

$$\text{code}(\boldsymbol{T}) = '\,r\,' + \text{code}(\boldsymbol{T}_1) + \text{code}(\boldsymbol{T}_2) + \ldots + \text{code}(\boldsymbol{T}_r) \tag{5.8}$$

kde '+' reprezentuje operáciu zreťazenia. Ak pravá strana (5.8) obsahuje koreňový strom obsahujúci dva alebo viac vrcholov, potom jeho kód je určený formulou analogickou formule (5.8), táto procedúra sa rekurzívne opakuje až do vtedy, že vyskytujúce sa koreňové stromy obsahujú len jeden vrchol (ich kódy sú jednoducho určené reťazcami '0'). Výsledný kód code($\boldsymbol{T}$) má tvar reťazcu - postupnosti $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_p)$, jeho dĺžka je označená $|\boldsymbol{\alpha}| = p$.



Obrázok 5. 3. Konštrukcia Readovho lineárneho kódu . Diagram A znázorňuje koreňový strom na ktorom je každý vrchol ohodnotený celým nezáporným číslom, ktoré je buď valenciou vrcholu alebo valenciou zníženou o jednotku pre ostatné vrcholy. Výsledný Readov kód je zostrojený na diagrame B tak, že idúc postupne zdola-nahor spájame ohodnotenie daného vrcholu a kódy nižších vrcholov. Koreňový strom je reprezentovaný kódom, ktorý je priradený koreňu.

***Príklad 5.2**. Zostrojte Readov kód pre niekoľko náhodne vybraných koreňových stromov.*

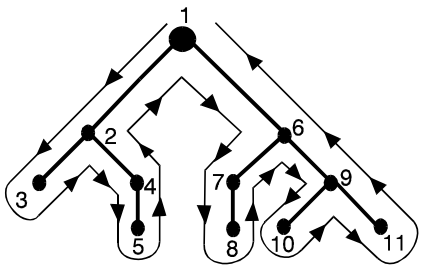
Základný problém pre efektívnosť Readovho kódu je tzv. analýza (parsing) kódu, ktorá spočíva v jednoznačnom preklade kódu do tvaru v ktorom vrcholy a hrany grafu sú identifikované. K tomuto bude použitý jednoduchý *back-track* algoritmus, ktorý zostrojí postupne tzv. *celkovú vychádzku* (spanning walk). Táto celková vychádzka navštívi každú hranu dvakrát (v opačných smeroch), pričom počiatočný a konečný bod tejto vychádzky je koreň, pozri algoritmus 5.1 a obr. 5.4.

Niekoľko poznámok k jednoznačnosti Readovho kódu. Ako vyplýva z vyššie uvedenej konštrukcie kódu, táto môže byť realizovaná mnohými spôsobmi, t.j. koreňový strom ***T*** je reprezentovaný mnohými kódmi. Ináč povedená, dva rôzne kódy môžu reprezentovať koreňové stromy, ktoré sú izomorfné. Toto ohraničenie Readovho kódu je jednoducho odstrániteľné tak, že formula (5.8) je aplikovaná tak, že príslušné "podkódy" sú usporiadané pred ich zreťazením do nerastúcej alebo neklesajúcej postupnosti. Týmto jednoduchým spôsobom získame tzv. *kanonický Readov kód*, ktorý má tú vlastnosť, že ak dva kanonické kódy sú rôzne, potom odpovedajúce koreňové stromy sú neizomorfné. V našich ďalších úvahách nebudeme používať kanonický Readov kód, pre potreby genetického programovania je plne postačujúci obyčajný Readov kód.

***Príklad 5.3.** Pre koreňové stromy z príkladu 5.2 realizujete indexovanie vrcholov pomocou algoritmu 5.1.*

```
procedure Parsing_Code(input:α; output:V,E);
begin E:=∅; V:={1};
      index0:=1; branch0:=α1; d:=0; i:=1;
      while d≥0 do
      if branchd>0 then
      begin branchd:=branchd-1; i:=i+1; d:=d+1;
            branchd:=αi; indexd:=i;
            V:=V∪{indexd};
            E:=E∪{[indexd-1,indexd]};
      end else d:=d-1;
end;
```

Algoritmus 5.1. Pseudopascalovská implementácia analýzy Readovho lineárneho kódu, ktorý vyhovuje podmienkam (5.9a-b), t.j. k analyzovanému kódu existuje koreňový strom. Výstupné parametre V a E sú vrcholová resp. hranová množina. Hĺbka back-track algoritmu je určená celočíselnou premennou d. Algoritmus postupne navštívi všetky vrcholy koreňového stromu, pozri obr. 5.4. Po ukončení algoritmu množiny V a E obsahujú všetky vrcholy resp. hrany, pričom koreň je indexovaný 1.



Obrázok 5.4. Back-track konštrukcia celkovej vychádzky koreňového stromu, ktorá začína a končí v korene. Vrcholy sú indexované postupne tak, ako sú navštevované vychádzkou.

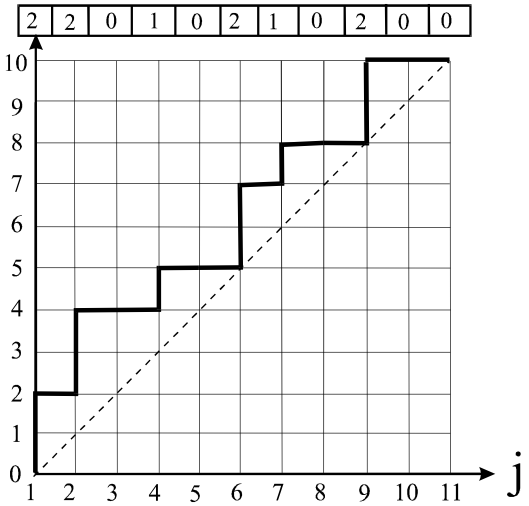
Obráťme teraz našu pozornosť na problém za akých podmienok postupnosť *p* nezáporných celých čísel je priradená Readovmu kódu. Takáto postupnosť sa nazýva *grafová*, ak existuje koreňový strom obsahujúci *p* vrcholov, ktorého Readov kód je totožný s postupnosťou.

Veta 5.1. [4,5]. Nutné a postačujúce podmienky k tomu, aby postupnosť nezáporných celých čísel $(\alpha_1, \alpha_2, \ldots, \alpha_p) \in \{0,1,2,\ldots\}^p$ bola grafová (t.j. existuje koreňový strom s rovnakým kódom) sú

$$\sum_{i=1}^{j} \alpha_i \geq j \qquad (j = 1,2,\ldots,p-1) \tag{5.9a}$$

$$\sum_{i=1}^{p} \alpha_i = p - 1 \tag{5.9b}$$

Druhá vlastnosť (5.9b) môže byť jednoducho dokázaná použitím vzťahu (5.2). Táto veta má veľký význam pre náhodnú generáciu koreňových stromov a pre hľadanie podstromov v koreňových stromoch, čo sú dôležité operácie pre jednoduchú implementáciu mutácia a kríženia nad stromami.



Obrázok 5.5. Diagramatické znázornenie podmienok (5.9a-b), jednotlivé komponenty kódu (22010210200) sú interpretované ako dĺžky vertikálnych častí hrubej lomenej čiary. Prerušovaná čiara vyjadruje nerovnosti (5.9a-b), hrubá čiara musí ležať nad touto čiarou.

```
Procedure Generation_Code(output:p,α);
begin p:=p_min+random(p_max-p_min+1);
      α_1:=1+random(p-1); d_1:=p-1;
      for j:=2 to p-1 do
      begin d_j:=d_j-1-α_i=1;
            if d_j=p-j then  α_j:=1+random(d_j)
            else α_j:=random(d_j+1);
      end;
      α_p:=0;
end;
```

Algoritmus 5.2. Pseudopascalovská implementácia náhodnej generácie Readovho lineárneho kódu dĺžky p, ktorý je automaticky grafový, t.j. vyhovuje podmienkam (5.9a-b). Funkcia `random(I)` je náhodný generátor celých čísel z uzavretého intervalu `[0,I-1]`. Algoritmus je založený na podmienkach (5.10-11). Dĺžka generovaného kódu je ohraničená podmienkami $p_{min} \le p \le p_{max}$.

Príklad 5.3. *Pomocou obr. 5.5 náhodne vygenerujte niekoľko kódov a priradte im koreňové stromy.*

Príklad 5.4. *Náhodne vygenerujte kódy pomocou nerovností (5.11) a verifikujte ich pomocou grafu na obr. 5.5.*

Definujme rekurentne tieto veličiny

$$d_1 = p - 1 \tag{5.10a}$$

$$d_j = d_{j-1} - \alpha_{j-1} \qquad (j = 2,3,\ldots,p) \tag{5.10b}$$

ktoré ohraničujú zhora komponenty kódu $(\alpha_1, \alpha_2, \ldots, \alpha_p)$

$$1 \le \alpha_j \le d_j \qquad (\text{if } d_j = p - j) \tag{5.11a}$$

$$0 \le \alpha_j \le d_j \qquad (\text{if } d_j < p - j) \tag{5.11b}$$

pre j=1,2,...,p. Tieto ohraničujúce podmienky sú veľmi výhodné pre náhodnú generáciu Readových kódov tak, aby podmienky (5.9a-b) boli splnené, pozri obr. 5.5. Implementácia náhodnej generácie Readových lineárnych kódov je vyjadrená pomocou algoritmu 5.2.

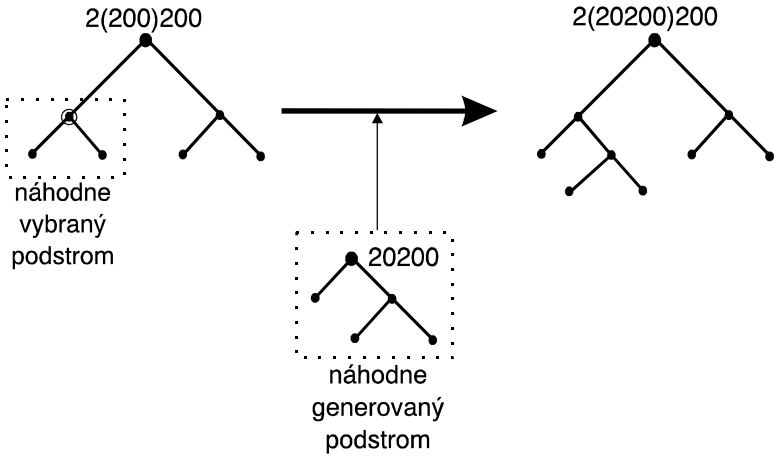
Pre možnosť použitia kódovania koreňových stromov pomocou Readovho prístupu v rámci evolučných algoritmov je potrebné implementovať v rámci tohto prístupu operácie mutácie a kríženia [7-9]. Operácia mutácie priradí stochasticky koreňovému stromu iný koreňový strom, pričom oba stromy môžu mať rôzny počet vrcholov (obvykle ohraničené intervalom [p_{min},p_{max}]). Ak starý a nový lineárny kód sú označené $\boldsymbol{\alpha}$ a $\boldsymbol{\alpha}'$, potom mutáciu formálne píšeme ako operátor transformujúci reťazec $\boldsymbol{\alpha}$ na reťazec $\boldsymbol{\alpha}'$

$$\boldsymbol{\alpha}' = O_{mut}(\boldsymbol{\alpha}) \tag{5.12}$$

kde $|\boldsymbol{\alpha}|, |\boldsymbol{\alpha}'| \in [p_{min}, p_{max}]$. Podľa Kozu [1] mutácia koreňového stromu je realizovaná tak, že v prvom kroku sa náhodne vyberie nekoreňový vrchol a príslušný podstrom sa nahradí iným podstromom náhodne generovaným, pozri obr. 5.6.

Alternatívna reprezentácia procesu mutácie môže byť formulovaná nasledujúcim spôsobom. Nech kód $\boldsymbol{\alpha}$ je rozdelený na tri podkódy, $\boldsymbol{\alpha} = (\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \boldsymbol{\alpha}_3)$, kde podkód $\boldsymbol{\alpha}_2$ odpovedá nejakému podstromu stromu vyjadrenému kódom $\boldsymbol{\alpha}$**,** pozri obr. 5.6 a 5.7. Podkód $\boldsymbol{\alpha}_2$ je nahradený novým náhodné generovaným podkódom $\boldsymbol{\alpha}_2'$, potom dostaneme nový kód interpretovaný ako mutácia kódu $\boldsymbol{\alpha}$, $\boldsymbol{\alpha}' = (\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2', \boldsymbol{\alpha}_3)$. Dĺžka podkódov $\boldsymbol{\alpha}_2$ a $\boldsymbol{\alpha}_2'$ je ohraničená nerovnosťami

$$p_{min} - |\boldsymbol{\alpha}| + |\boldsymbol{\alpha}_2| \le |\boldsymbol{\alpha}_2'| \le p_{max} - |\boldsymbol{\alpha}| + |\boldsymbol{\alpha}_2| \tag{5.13}$$



Obrázok 5.6. Ilustratívny príklad mutácie koreňového stromu. V prvom kroku je náhodne vybraný nekoreňový vrchol, ktorý je zakrúžkovaný na ľavom koreňovom strome. Príslušný koreňový podstrom je nahradený iným náhodne generovaným koreňovým podstromom. Počiatočný a konečný kód príslušných koreňových stromov sú uvedené pri koreňových vrcholoch.

Podkód $\boldsymbol{\alpha}_2$ kódu $\boldsymbol{\alpha}$ je určený tak, že najprv náhodne určíme tzv. mutačný bod $2 \le \tau \le |\boldsymbol{\alpha}| - 1$, určuje tú polohu v kóde $\boldsymbol{\alpha}$ od ktorej začína podkód $\boldsymbol{\alpha}_2$. Ako efektívne stanovíme dĺžku podkódu $\boldsymbol{\alpha}_2$? K tomuto účelu je vhodné použiť nerovnosti (5.9a-b), hovoríme, že podkód $\boldsymbol{\alpha}_2$ má dĺžku *r* ak nasledujúce podmienky sú splnené

$$\sum_{i=1}^{j} \alpha_{i+\tau-1} \ge j \qquad (j = 1,2,\ldots,r-1) \tag{5.14a}$$

$$\sum_{i=1}^{r} \alpha_{i+\tau-1} = r-1 \tag{5.14b}$$

Pseudopascalovská implementácia mutácie Readovho lineárneho kódu je uvedená v algoritme 5.3.

```
procedure Mutation(input:p,α;output:p',α');
begin τ:=2+random(p-1);
      r:=length of a subcode α2 of α that
         starts at τ;
      r':=pmin-p+r+random(pmax-pmin+1);
      Generation_Code(r',α2');
      α'=(α1,α2',α3); p':=p-r+r';
end;
```

Algoritmus 5.3. Implementácia mutačného procesu aplikovaného na kód **α,** pričom výsledný kód je **α**'. Algoritmus je inicializovaný náhodným generovaním mutačného bodu τ, dĺžka podkódu, ktorý začína v bode τ je vyjadrená premennou `r`. Dĺžka nového náhodne generovaného podkódu je označená `r'`.

```
procedure Crossover(input:p,p̃,α,β;output p',p̃',α',β');
begin τ:=2+random(p-1);  τ̃:= 2 + random(p̃ − 1)
      α′ = (α1,β2,α3);  β′ = (β1,α2,β3);
      p':=p-|α2|+|β2|;  p̃':= p̃ -|β2|+|α2|;
end;
```

Algoritmus 5.4. Implementácia operátora kríženia dvoch rodičovských lineárnych kódov $\boldsymbol{\alpha}$ a $\boldsymbol{\beta}$, ktoré sú modifikované na nové lineárne kódy - potomkov $\boldsymbol{\alpha}'$ a $\boldsymbol{\beta}'$. Náhodný výber bodov kríženia τ a $\tilde{\tau}$ je realizovaný tak, že dĺžky nových lineárnych kódov vyhovujú nerovnostiam $p_{min} \le |\boldsymbol{\alpha}'| \le p_{max}$ a $p_{min} \le |\boldsymbol{\beta}'| \le p_{max}$.

***Príjklad 5.5**. Pre kódy vygenerované v príklade 5.3, určite pomocou nerovností 5.14 podkódy, vykonajte grafickú interpretáciu podkódu.*

Nech $\boldsymbol{\alpha}$ a $\boldsymbol{\beta}$ sú dva (rodičovské) lineárne kódy koreňových stromov, aplikovaním operácie kríženie dostaneme dva nové lineárne kódy (potomkov) $\boldsymbol{\alpha}'$ a $\boldsymbol{\beta}'$

$$(\boldsymbol{\alpha}', \boldsymbol{\beta}') = O_{cross}(\boldsymbol{\alpha}, \boldsymbol{\beta}) \tag{5.15}$$

Táto operácia je podľa Kozu [1,2] realizovaná tak, že v rodičovských kódoch - chromozómoch $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ náhodne vyberieme dva body kríženia a príslušné podkódy, ktoré začínajú v týchto bodoch sa vymenia, pozri obr. 5.7.

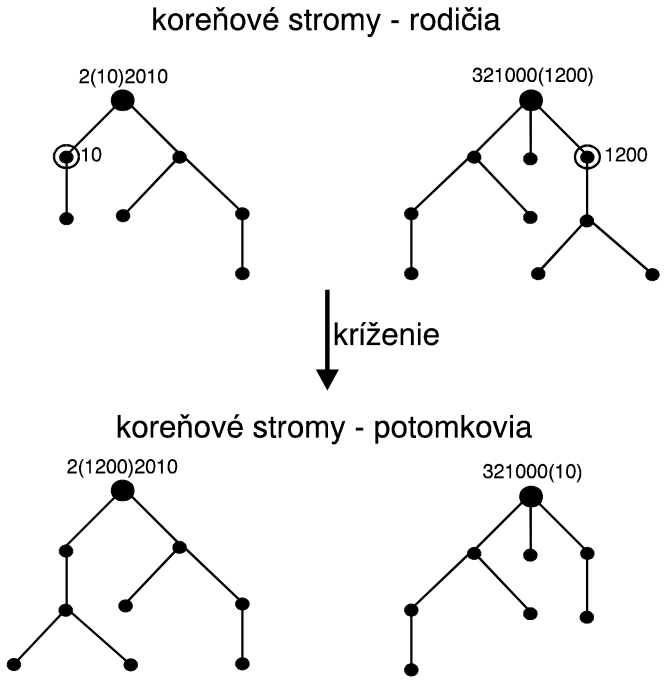
Alternatívna reprezentácia operátora kríženia je nasledovná. Nech rodičovské kódy $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ koreňových stromov majú tvar $\boldsymbol{\alpha} = (\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \boldsymbol{\alpha}_3)$ resp. $\boldsymbol{\beta} = (\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \boldsymbol{\beta}_3)$, kde počiatočné body podkódov $\boldsymbol{\alpha}_2$ a $\boldsymbol{\beta}_2$ boli náhodne generované. Aplikácia operácie kríženia na rodičovské kódy $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ spočíva vo výmene podkódov $\boldsymbol{\alpha}_2$ and $\boldsymbol{\beta}_2$, potomkovia majú potom tvar

$$\boldsymbol{\alpha}' = (\boldsymbol{\alpha}_1, \boldsymbol{\beta}_2, \boldsymbol{\alpha}_3) \text{ a } \boldsymbol{\beta}' = (\boldsymbol{\beta}_1, \boldsymbol{\alpha}_2, \boldsymbol{\beta}_3) \tag{5.16}$$

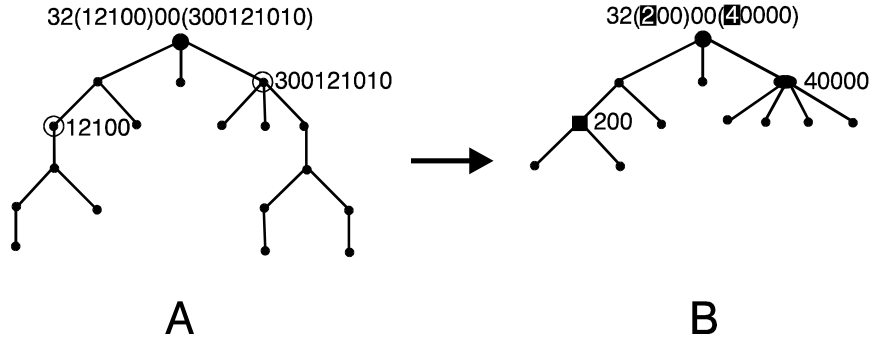
Dĺžka potomkov je určená vzťahmi $|\boldsymbol{\alpha}'|=|\boldsymbol{\alpha}|-|\boldsymbol{\alpha}_2|+|\boldsymbol{\beta}_2|$ a $|\boldsymbol{\beta}'|=|\boldsymbol{\beta}|-|\boldsymbol{\beta}_2|+|\boldsymbol{\alpha}_2|$, pričom tieto dĺžky sú ohraničené nerovnosťami $p_{min} \le |\boldsymbol{\alpha}'| \le p_{max}$ resp. $p_{min} \le |\boldsymbol{\beta}'| \le p_{max}$. Pseudopascalovksá implementácia operátora kríženia je uvedená v algoritme 5.4.

Ako posledný problém, ktorý budeme študovať v tejto kapitole, je tzv. kompresia Readovho kódu, ktorá úzko súvisí s ADF (Automatically Defined Function) prístupom používaným Kozom pre zvýšenie efektívnosti genetického programovania [2]. Nech koreňový strom $\boldsymbol{T}$ je reprezentovaný kódom $\boldsymbol{\alpha}$. Ak vyberieme v tomto strome podstrom, potom kód $\boldsymbol{\alpha}$ môže byť vyjadrený rozkladom $\boldsymbol{\alpha} = (\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \boldsymbol{\alpha}_3)$, kde $\boldsymbol{\alpha}_2$ odpovedá kódu vybraného podstromu, pozri obr. 5.8. Proces kompresie spočíva v substitúcii podkódu $\boldsymbol{\alpha}_2$ tzv. komprimovaným kódom typu (n00...0), kde n je počet terminálových vrcholov v podstrome (alebo počet komponent '0' v podkóde $\boldsymbol{\alpha}_2$).

$$\boldsymbol{\alpha} = (\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \boldsymbol{\alpha}_3) \xrightarrow{kompresia} \boldsymbol{\alpha}_{compress} = (\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_{2,compress} = (n00\ldots0), \boldsymbol{\alpha}_3) \tag{5.17}$$

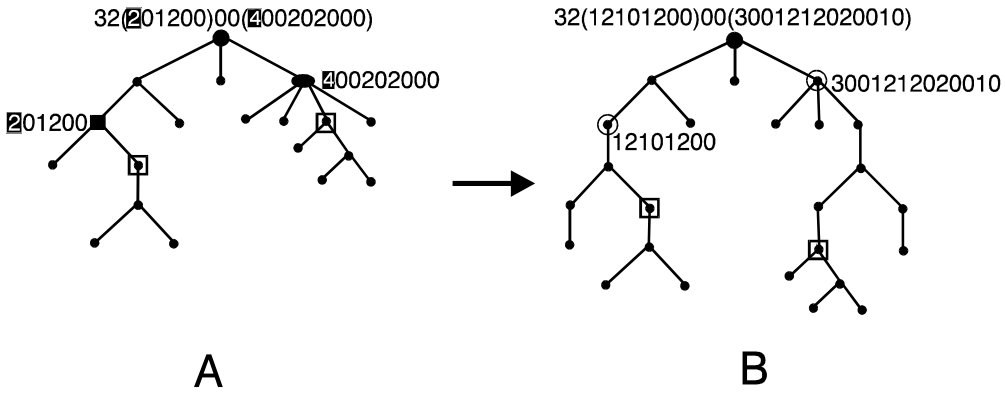


Obrázok 5.7. Ilustratívny príklad operácie kríženia dvoch rodičovských koreňových stromov, ktoré sú modifikované na dva nové koreňové stromy - potomkov. Zakrúžkované vrcholy označujú náhodne generované body kríženia. Operácia kríženia spočíva v tom, že podstromy sa medzi koreňovými stromami vymenia.



Obrázok 5.8. Ilustračný príklad kompresie Readovho lineárneho kódu. Koreňový strom reprezentovaný diagramom A je komprimovaný na menší koreňový strom reprezentovaný diagramom B. Tento proces sa vykoná tak, že vybrané podstromy (ich koreňové vrcholy sú zakrúžkované) v diagrame A sú nahradené novými vrcholmi (štvorcovým alebo oválnym), ktoré sú incidentné len s terminálovými vrcholmi, pričom ich počet je rovnaký ako počet terminálových vrcholov v pôvodných podstromoch.

Ako realizovať inverzný proces ku kompresii ? Nech kód **β** obsahuje vrchol interpretovaný ako "komprimovaný" vrchol. Potom kód **β** môže byť dekomprimovaný tak, že odpovedajúci podstrom priradený komprimovanému vrcholu je nahradený koreňovým podstromom, ktorý indukuje daný komprimovaný vrchol, pričom niektoré terminalové vrcholy môžu byť substituované podstromami (pozri obr. 5.9).



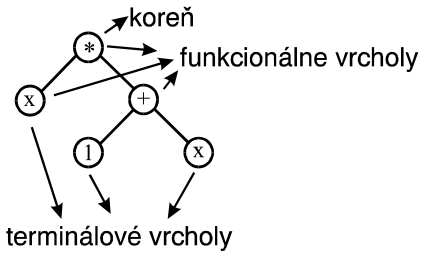
Obrázok 5.9. Ilustratívny príklad dekompresie Readovho lineárneho kódu. Koreňový strom A obsahuje dva komprimované vrcholy reprezentované štvorcovým a oválnym vrcholom, pričom vrcholy označené štvorcom (pôvodne terminalové) boli substituované reťazcami 1200 resp. 20200. Ak komprimované vrcholy sú "dekomprimované" odpovedajúcimi koreňovými podstromami (pozri obr. 5.7), potom pôvodný strom A je dekomprimovaný na väčší strom B, ktorý už obsahuje len štandardné vrcholy.

Príklad 5.6*. Podrobne preštuduj príklady kompresie a dekompresie znázornené na obrázkoch 5.8 a 5.9. Navrhni iné prípady kompresie a dekompresie.*

5.3 Symbolická regresia

Symbolická regresia patrí medzi základné aplikácie genetického programovania. Ako už bolo spomenuté v úvodnej kapitole 5.1, symbolická regresia spočíva v hľadaní takej funkcie reprezentovanej syntaktickým stromom s predpísanými operáciami, ktorá čo najlepšie aproximuje údaje z treningovej množiny.

Čo je to syntaktický strom? Pod syntaktickým stromom *t* budeme rozumieť koreňový strom *T*, ktorého vrcholy sú ohodnotené symbolmi aritmetických (alebo iných) operácií a celý strom môže byť ohodonotený reálnym číslom reprezentujúcim hodnotu funkcie, ktorá je priradená stromu pre danú vstupnú hodnotu nezávislej premennej (alebo nezávislých premenných). Vrcholy v syntaktických stromoch klasifikujeme takto (pozri obr. 5.10).



Obrázok 5.10. Klasifikácia vrcholov syntaktického stromu na funkcionálne a terminálové.

(1) *Terminálne vrcholy*, tieto vrcholy odpovedajú buď *nezávislým premenným* x,y,... nezáporným celočíselným *konštantám* 0,1,2,....

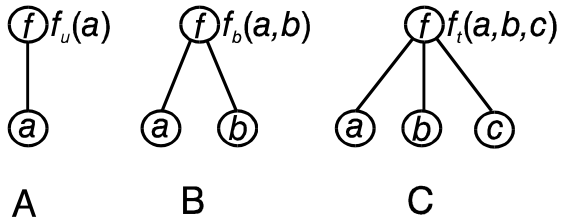
(2) *Funkcionálne vrcholy* odpovedajú jednoduchým operáciam, ktoré sú unárne, binárne, ternárne,... .

Formálne môžeme syntaktický strom vyjadriť ako usporiadanú dvojicu

$$\boldsymbol{t} = (\boldsymbol{T}, \phi) \tag{5.18}$$

kde ***T*** je koreňový strom určujúci štruktúru syntaktického stromu ***t*** a ϕ je zobrazenie vrcholov koreňového stroma na "aritmetické" symboly, premenné, alebo konštanty

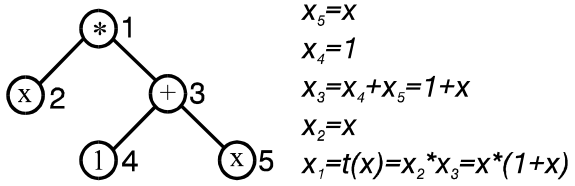
$$\phi: V \rightarrow \{*,+,-,\ldots,x,y,\ldots,1,2,\ldots\} \tag{5.19}$$



Obrázok 5.11. Klasifikácia funkcionálnych vrcholov na unárne (A), binárne (B) a terciárne (D). Unárny vrchol má jednoho predchodcu, ktorého hodnota *a* sa zobrazuje funkčnú hodnotu $f_u(a)$. Binárny vrchol má dvoch predchoscov, ich funkčné hodnoty *a* a *b* sú zobrazené na $f_b(a.b)$. Ternárny vrchol má troch predchodcov, hodnoty *a*, *b* a *c* sú zobrazené na $f_t(a,b,c)$. Vo všeobecnosti platí, že binárne, unárne, ... vrcholy nie sú "komutatívne", t.j. napr. pre binárny vrchol plati $f_t(a,b) \neq f_t(b,a)$.

Každému syntaktickému stromu ***t*** priradíme funkciu *t(x,y,....)*, kde *x,y,....* sú nezávislé premenné nasledujúcim rekurzívnym spôsobom: Nech každý vrchol syntaktického stromu je označený indexom, toto indexovanie je realizovateľné pomocou algoritmu 5.1 (pozri obr. 5.4). Potom *i*-temu vrcholu môže byť predpísaná "aktivita" x_i tak, že terminálové vrcholy tieto aktivity sú rovné ohodonoteniam týchto vrcholov (t.j. konštantám alebo nezávislým premenným) , postupujúc zdola nahor tieto aktivity postupne počítame tak, že na základe topológie syntaktického stromu počítame aktivity tých vrcholov, ktoré majú predchodcov s už známymi hodnotami aktivít. Tento proces končí v korene stromu, jeho aktivita sa rovná hodnote funkcie *t*(*x,y*,...) priradenej syntaktickému stromu ***t*** (pozri obr. 5.12)

$$x_1 = t(x,y,\ldots) \tag{5.20}$$



Obrázok 5.12. Ilustratívny príklad interpretácie jedno-duchého syntaktického stromu, jednotlivé aktivity sú počítané "zpätným šírením" po strome, začína sa od terminálových vrcholov a končí sa v korene stromu, jeho aktivita priamo odpovedá hodnote funkcie.

Pre daný súbor funcionálnych a terminálnych vrcholov môžeme definovať množinu $\mathcal{T}$ obsahujúcu všetky možné neekvivalentné syntaktické stromy, ktoré majú počet vrcholov z predpísaného intervalu a ktoré sú ohodnotené symbolmi z predpísaných súborov

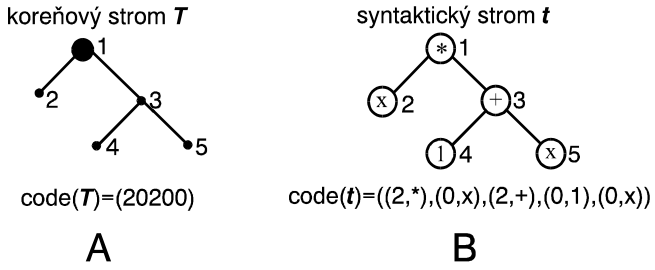
$$\mathcal{T} = \{\boldsymbol{t}_1, \boldsymbol{t}_2, \ldots\} \tag{5.21}$$

Budeme používať nasledujúcu konvenciu, každý syntaktický strom $\boldsymbol{t} \in \mathcal{T}$ je identifikovaný s jej interpretáciou - funkciou *t(x,y,...)*. To znamená, že množina $\mathcal{T}$ môže byť tiež alternatívne chápaná ako množina funkcií, ktorých tvar je ohraničený podmienkami kladenými na syntaktické stromy.

Ako kódovať syntaktické stromy? Podľa (5.18) syntaktický strom je definovaný ako koreňový strom, ktorého vrcholy sú ohodnotené zobrazením (5.19), $\boldsymbol{t}=(\boldsymbol{T},\phi)$. Nech Readov lineárny kód stromu ***T*** je code(***T***)=$\boldsymbol{\alpha}=(\alpha_1,\alpha_2,\ldots,\alpha_p)$, potom lineárny kód syntaktického stromu ***t*** môže byť zostrojený tak, že Readov kód koreňového stromu je rozšírený o zobrazenie ϕ

$$\text{code}(\boldsymbol{t}) = \left((\alpha_1,\phi_1),(\alpha_2,\phi_2),\ldots,(\alpha_p,\phi_p)\right) \tag{5.22}$$

kde ϕ_i je ohodnotenie *i*-teho vertexu buď funkcionálnym alebo terminálovým symbolom. Týmto spôsobom sme dostali jednoduchý prostriedok pre kódovanie syntaktických stromov pomocou Readovho lineárneho kódovania koreňových stromov, pozri obr. 5.13.



Obrázok 5.13. Tvorba kódu syntaktického stromu ***t*** pomocou Readovho lineárneho kódu priradeného koreňového stromu ***T***. Kód syntaktického stromu získame z pôvodného kódu koreňového stromu tak, že každá jeho komponenta je doprevádzaná tiež aj ohodnotením príslušného vrcholu.

Majme tréningovú množinu obsahujúcu *n* bodov (regresnú tabulku)

$$A_{train} = \{x_i \ / \ y_i; i = 1,2,...,n\} \tag{5.23}$$

Cieľom štandardne regresnej analýzy je nájsť také optimálne parametre modelovej funkcie $G(x;w)$, kde **w** sú parametre funkcie *G*, také, že nasledujúca účelová funkcia je minimalizovaná

$$E(w) = \sum_{i=1}^{n} \left| G(x_i;w) - y_i \right| \tag{5.24}$$

Táto funkcia má minimum v bode

$$w_{opt} = \arg\min_{w} E(w) \tag{5.25}$$

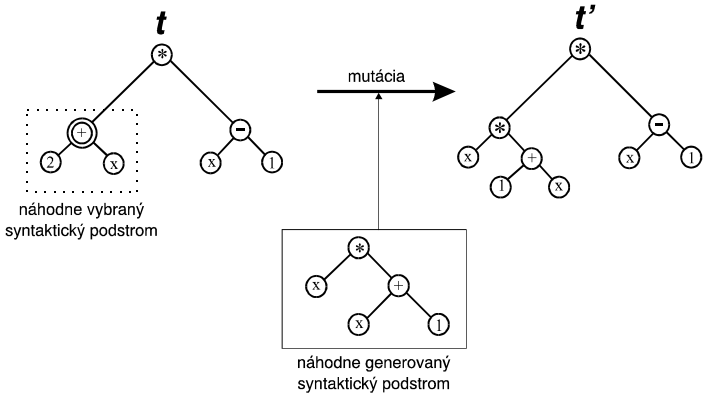
Hovoríme, že adaptovaná funkcia $G(\mathbf{x},w_{opt})$ modeluje tréningovú množinu A_{train}

Symbolická regresia ide ďalej ako štandardná regresná analýza, hladá v množine $\mathcal{T}$ takú funkciu, že nasledujúca účelová funkcia (funkcionál) je minimalizovaná

$$E(t) = \sum_{i=1}^{n} \left| t(x_i) - y_i \right| \tag{5.26}$$

pričom táto účelová funkcia má minimum v "bode"

$$t_{opt} = \arg\min_{t \in T} E(t) \tag{5.27}$$



Obrázok 5.14. Ilustračný príklad mutácie syntaktického stromu ***t*** na iný syntaktický strom ***t'*** . V prvej etape mutácie sa náhodne vyberie podstrom a tento sa v druhej etape zamení za háhodne vygenerovaný podstrom. Formálne môžeme povedať, že stochastickou operáciou mutácia sa funkcia *t(x)* priradená pôvodnému stromu zamenila za novú funkciu *t'(x)* .

Symbolická regresia je hlavným cielom Kozovho genetického programovania s reštrikciou, že povolené funkcie sú len tie, ktoré môžu byť reprezentované syntaktocým stromom obsahujúcim povolené fukcionálne a terminálové vrcholy.

Riešenie minimalizačného problému (5.27) sa realizuje pomocou genetického algoritmu prezentovaného v predchádyajúcej kapitole. Stochastické operácie mutácie a kríženia sú definované podobným spôsobom ako pre koreňové stromy. Operácia mutácie zmení syntaktický strom $\boldsymbol{t}$ na iný syntaktický strom $\boldsymbol{t}'$, formálne

$$\boldsymbol{t}' = O_{mut}(\boldsymbol{t}) \tag{5.28}$$

určitá časť syntaktického stromu (t.j. syntaktický podstrom) je nahradená náhodne generovaným syntaktickým podstromom (pozri obr. 5.14)

Každému syntaktickému stromu môžeme jednoznačne priradit funkciu *t(x,y,...)*, potom mutáciu môžeme formálne chápať ako stochastickú transformáciu funkcie na inú funkciu, v prípade ilustračného príkladu na obr. 5.14 tieto funkcie majú tvar

$$\begin{aligned} t(x) &= (2+x)*(x-1) \\ t'(x) &= \big(x*(1+x)\big)*(x-1) \end{aligned} \tag{5.29}$$

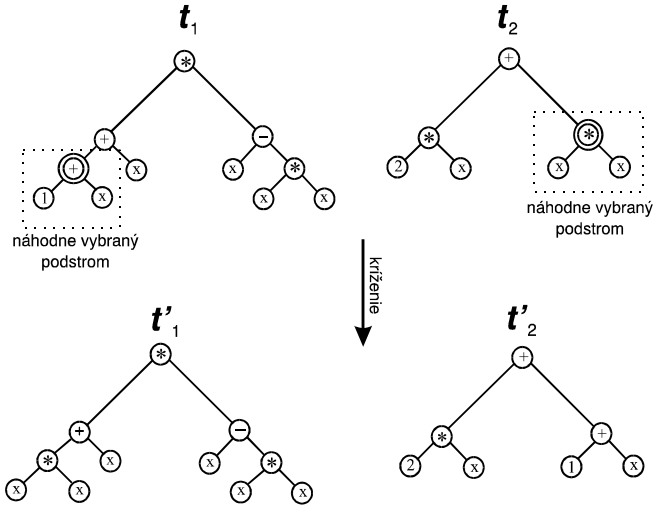
Operácia kríženia pre dva syntaktické stromy je definovaná analogickým spôsobom ako pre koreňové stromy. Majme dva syntaktické stromy $\boldsymbol{t_1}$ a $\boldsymbol{t_2}$, stochastickou operáciou kríženia tieto dva stromy sú transformované na dva nové stromy $\boldsymbol{t}'_1$ a $\boldsymbol{t}'_2$

$$(\boldsymbol{t}'_1, \boldsymbol{t}'_2) = O_{cross}(\boldsymbol{t}_1, \boldsymbol{t}_2) \tag{5.30}$$

Táto operácia sa realizuje tak, že v stromoch náhodne vyberieme dva podstromy a tieto navzájom vymeníme (pozri obr. 5.15).

Podobne ako v predchádzajúcom príklade mutácie aj pre kríženie môžeme hovoríť, že stochastická operácia kríženia nám transformovala východzie funkcie $t_1(x)$ a $t_2(x)$ na nové funkcie $t'_1(x)$ a $t'_2(x)$. Pre prípad syntaktických stromov na obr. 5.15 tieto funkcie majú tvar

$$\begin{aligned} t_1(x) &= (1+2x)*\left(x-x^2\right) \\ t_2(x) &= 2x+x^2 \\ t'_1(x) &= \left(x^2+x\right)*\left(x-x^2\right) \\ t'_2(x) &= 2x+(1+x) \end{aligned} \tag{5.31}$$



Obrázok 5.15. Ilustračný príklad kríženia dvoch syntaktických stromov. V prvej etape v každom strome sa náhodne vyberie podstrom, v druhej etape sa tieto podstromy vzájomne zamenia. Formálne môžeme povedať, že operáciou kríženia sa dve funkcie $t_1(x)$ a $t_2(x)$ pretransformovali na nové funkcie $t'_1(x)$ a $t'_2(x)$.

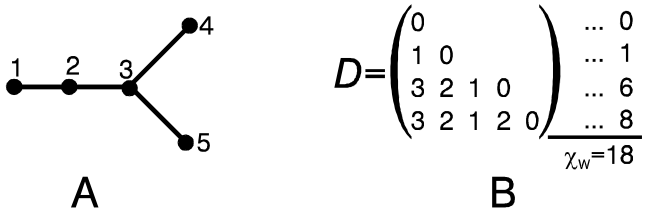
***Príklad 5.7**. Implementuje genetický algoritmus (pozri algoritmus 3.5) tak, že chromozómy budú určené ako Readove lineárne kódy s ohodnotením vrcholov podľa (5.18-19). Toto ohodnotenie nech jhe vykonané tak, že syntakticé strumy budú odpovedať buď racionálnym polynómom (t.j. funkčné vrcholy sú +,-,*, a /, terminálové vrcholy sú celočíselné konštanty alebo premenná x) alebo Boolovským funkciám (t.j. funkčné vrcholy sú operácia and, or, xor, a*

not, a terminálové vrcholy sú jednotlivé Boolovské premenné). Účelová funkcia (5.24) nech je rozšírená o pokutový clen, ktorý bude preferovať syntaktické stromy s menším poštom vrcholov

$$E(t) = \sum_{i=1}^{n} \left| t(x_i) - y_i \right| + \alpha |t|$$

kde α je malá kladná konštanta a symbol |t| vyjadruje počet vrcholov v strome t. Modifikujete napísanú implementáciu tak že bude odpovedať horolezeckému algoritmu alebo evolučnému programovaniu. Porovnajte efektívnosť týchto troch rozdieľných metód pri konštrukcii funkcií, ktoré reprodukujú predpísanú regresnú tabulku.

5.4 Rekonštrukcia stromov s požadovanou vlastnosťou



Obrázok 5.16. Schematické znázornenie výpočtu Wienerovho indexu pre jednoduchý strom znázornený diagramom A. Diagram B obsahuje maticu vzdialeností ***D*** , suma jej zkožiek pod diagonálou tvorí Wienerov index.

Pred nedávnom autori tejto publikácie vypracovali [6] verziu genetického programovania, ktorá je schopná rekonštruovať stromy (súvislé acyklické grafy) s požadovanými vlastnosťami. Pod vlastnosťou budeme rozumieť reálne číslo, ktoré je priradené stromu ***G*** z množiny prípustných stromov $\mathcal{T}$, formálne $t: \mathcal{T} \to R$. Úloha rekonštrukcie spočíva v tom, že hľadáme v množine prípustných stromov $\mathcal{T}$ taký strom ***G***, ktorého vlastnosť $t(\boldsymbol{G})$ je blízka požadovanej vlastnosti t_{req}. Definujme účelovú funkciu

$$E(\boldsymbol{G}) = \left| t(\boldsymbol{G}) - t_{req} \right| \tag{5.32}$$

Optimálny strom, ktorého vlastnosť minimalizuje funkcionál (5.32) je určený ako riešenie nasledujúceho minimalizačného problému

$$\boldsymbol{G}_{opt} = \arg \min_{\boldsymbol{G} \in \mathcal{T}} E(\boldsymbol{G}) \tag{5.33}$$

Riešenie tohto optimalizačného problému sa realizuje pomocou genetického algoritmu nad populáciou stromov, ktoré sú kódované pomocou Readovho lineárneho kódu a operácie mutácie a kríženia sa vykonávajú spôsobom popísaným nižšie.

Akým spôsobom popísať vlastnosti grafu ***G***=(*V*,*E*)? Najjednoduchší spôsob je pomocou tzv. *topologických indexov* $\chi_1(\boldsymbol{G})$, $\chi_2(\boldsymbol{G})$,...., ktoré sú charakterizované ako funkcie - zobrazenia priraďujúce grafu reálne číslo. Medzi najznámejšie topologické indexy patria tieto topologické indexy:

(1) *Wienerov topologický index* [10]

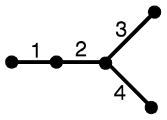
$$\chi_W(\boldsymbol{G}) = \sum_{i<j} d_{ij} \tag{5.34}$$

kde sumácia obsahuje všetky rôzne dvojice vrcholov a symbol d_{ij} je vzdialenosť medzi *i*-tým a *j*-tým vrcholom v grafe ***G*** (pozri obr. 5.16).

(2) *Randičov topologický index* [xx]

$$\chi_R(\boldsymbol{G}) = \sum_{[v,v'] \in E} \frac{1}{\sqrt{\mathrm{val}(v) \cdot \mathrm{val}(v')}} \tag{5.35}$$

kde sumácia obsahuje všetky hrany $[v,v'] \in E(\boldsymbol{G})$ (pozri obr. 5.17).



A

$$\chi_R = \underbrace{\frac{1}{\sqrt{1\cdot 2}}}_{e_1} + \underbrace{\frac{1}{\sqrt{2\cdot 3}}}_{e_2} + \underbrace{\frac{1}{\sqrt{1\cdot 3}}}_{e_3} + \underbrace{\frac{1}{\sqrt{1\cdot 3}}}_{e_4}$$

B

Obrázok 5.17. Výpočet Randičovho topologického indexu pre strom znázornený diagramom A, kde jednotlivé hrany stromu sú indexované. V diagrame B je priradený ku každej hrane odpovedajúci výraz $1/\sqrt{\mathrm{val}(v)\cdot \mathrm{val}(v')}$.

Pre lineárne stromy (t.j. taký strom, ktorý neobsahuje tzv. vetviace vrcholy, každý vrchol je buď valencie 1 alebo 2) jednoduchými úvahami (matematickou indukciou) je možné zostrojiť explicitné výrazy pre Wienerov a Randičov topologický index

$$\chi_W = \frac{p\left(p^2-1\right)}{6} \qquad \left(\text{pre } p \geq 1\right) \tag{5.36a}$$

$$\chi_R = \sqrt{2} + \frac{p-3}{2} \qquad \left(\text{pre } p \geq 3\right) \tag{5.36b}$$

Postulujme, že vlastnosť $t(\boldsymbol{G})$ grafu $\boldsymbol{G}$ je popísaná ako konvexná kombinácia Wienerovho a Randičovho indexu

$$t(\boldsymbol{G}) = \omega\chi_R(\boldsymbol{G}) + (1-\omega)\chi_W(\boldsymbol{G}) \tag{5.37a}$$

pre $0 \leq \omega \leq 1$. Táto "vlastnosť" pre lineárne grafy má tvar

$$t(\boldsymbol{G}) = \omega\left(\sqrt{2} + \frac{p-3}{2}\right) + (1-\omega)\left(\frac{p\left(p^2-1\right)}{6}\right) \tag{5.37b}$$

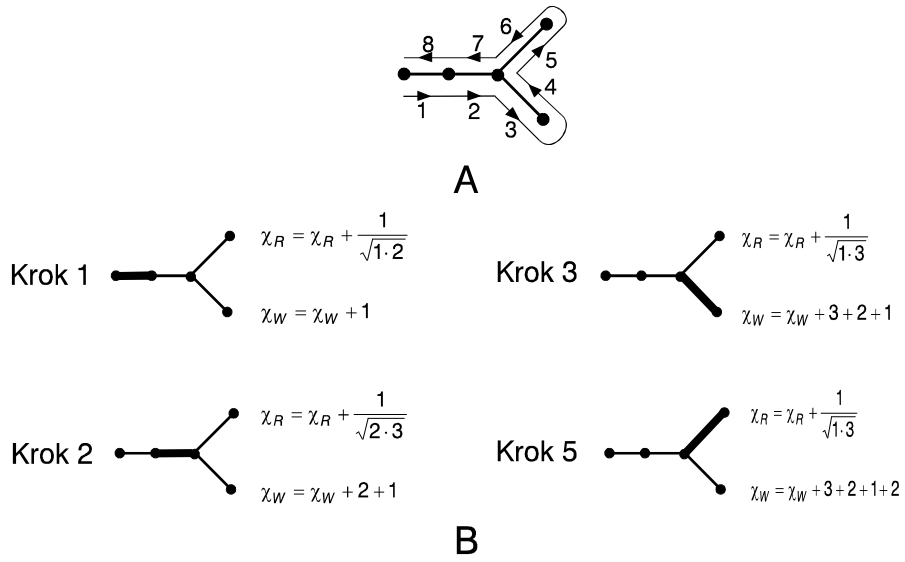
Jeden z hlavných problémov popisovanej rekonštrukcie stromov v rámci genetického programovania je výpočet Wienerových a Randičových topologických indexov pre stromy, ktoré sú reprezentované Readovým lineárnym kódom. Tieto topologické indexy môžu byť jednoducho spočítané v rámci back-track analýzy kódu vyjadreného Algoritmom 5.1, tieto jeho modifikácie sú vyjedrené pascalovskými pseudokódmi v Algoritmoch 5.5 a 5.6 (pozri obr. 5.18)

```
function Randic_topological_index(α):real;
begin χ:=0; branch0:=α1; val0:=0; d:=0; i:=1;
      while d≥0 do
      if branchd>0 then
      begin branchd:=branchd-1; i:=i+1; d:=d+1;
            branchd:=αi; vald:=αi+1;
            χ:=χ+1/sqrt(vald-1*vald)
      end else d:=d-1;
      Randic_topological_index:=χ;
end;
```

Algoritmus 5.5. Pseudopascalovská implementácia výpočtu Randičovho indexu stromu zadaného pomocou Readovho lineárneho kódu, tento algoritmus je jednoduchou modifikáciou back-track analýzy kódu vyjadreného Algoritmom 5.1.

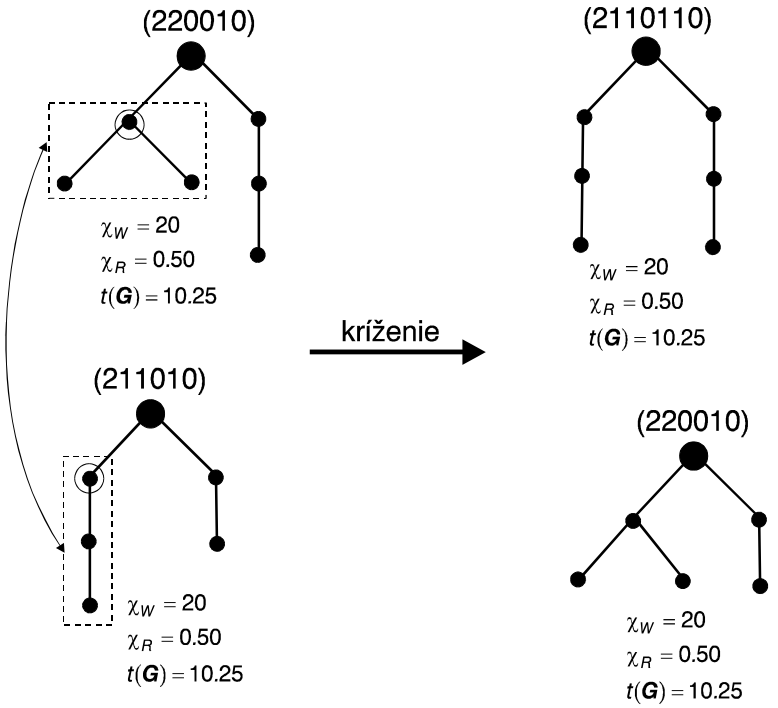
```
function Wiener_topological_index(α):real;
begin χ:=0; branch0:=α1; index0:=1; d11:=0; d:=0; i:=1;
      while d≥0 do
      if branchd>0 then
      begin branchd:=branchd-1; i:=i+1; d:=d+1;
            branchd:=αi; indexd:=i;
            for j:=1 to i-1 do
             begin dij:=dindexd-1,j+1; dij:=dji; χ:=χ+dij end;
      dii:=0;
      end else d:=d-1;
      Wiener_topological_index:=χ;
end;
```

Algoritmus 5.6. Pseudopascalovská implementácia výpočtu Wienerovho indexu stromu zadaného pomocou Readovho lineárneho kódu, tento algoritmus je jednoduchou modifikáciou back-track analýzy kódu vyjadreného Algoritmom 5.1. V priebehu analýzy kódu sa postupne zostrojí matica vzdialeností (d_{ij}).

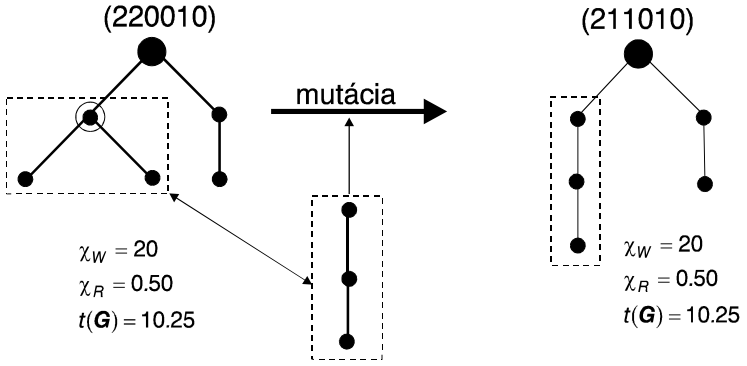


Obrázok 5.18. V priebehu analýzy koreňového stromu (diagram A) sú identifikované hrany grafu. V priebehu každého kroku tejto identifikacie vykoná sa čiastočný výpočet Randičovho a Wienerovho indexu (diagram B).

Operácie mutácie a kríženia pre problém rekonštrukcie koreňových stromov sú identické s podobnými operáciami uvedenými v kap. 5.2. Pre jednoduchosť predpokladajme, že parameter ω=0.5, potom vlastnosť $t(\boldsymbol{G})$ je určená ako aritmetický priemer topologických indexov $\chi_W(\boldsymbol{G})$ a $\chi_R(\boldsymbol{G})$. Na obrázkoch 5.19-20 sú uvedené jednoduché ilustračné príklady pre mutáciu a kríženie a odpovedajúce zmeny vlastnosti $t(\boldsymbol{G})$.



Obrázok 5.20. Ilustračný príklad kríženia dvoch koreňových stromov. V každom strome je náhodne vybraný vrchol (zakrúžkovaný), príslušné podstromy si koreňvé stromy medzi sebou vymenia. Príslušné hodnoty toipologických indexov a odpovedajúcich vlastností sú uvedené pod grafdmi.

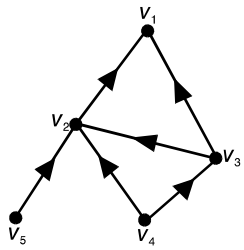


Obrázok 5.19. Ilustračný príklad mutácie koreňového stromu, podstrom susedný s náhodne vybraným vrcholom (zakrúžkovaný) sa vymení za náhodne vygenerovaným podstromom. Hodnoty topologických indexov a príslušná vlastnosť sú uvedené pod grafmi.

Príklad 5.8. *Genetický algoritmus z predchádzajúceho príkladu 5.7 modifikujte tak, že účelová funkcia (5.24) bude nahradená účelovou funkciou (5.32), pričom špecifikuje požadovanú vlastnosť stromu nejakým zvoleným jednoduchým spôsobom*

5.5 Kódovanie funkcií pomocou acyklických orientovaných grafov

Uvedieme alternatívny prístup ku kódovaniu funkcií pomocou acyklických orientovaných grafov [12,13], ktoré v tejto súvislosti môžu byť chápané ako zovšeobecnenie koreňových stromov, ktoré boli podrobne diskutované v kapitole 5.2.



Obrázok 5.21. Orientovaný graf obsahujúci päť vrcholov a šesť orientovaných hrán. Množina vrcholov V obsahuje päť vrcholov, $V=\{v_1,v_2,v_3,v_4,v_5\}$, množina hrán obsahuje šesť orientovaných hrán, $E=\{e_1,e_2,e_3,e_4,e_5,e_6\}$. Orientovaná $e\in E$ je určená pomocou usporiadanej dvojice dvoch vrcholov $v,v'\in E$, $e=(v,v')$. Hovoríme, že orientovaná hrana e začína v o vrchole v (hrana e vzchádza z vrcholu v) a končí vo vrchole v' (hrana e vchádza do vrcholu v'). Graf je acyklický, neobsahuje cyklickú cestu, ktorá by sa skladala z postupnosti rovnako orientovaných hrán.

Študujme orientovaný graf [3] (hrany grafu sú orientované) $\boldsymbol{G}=(V,E)$, kde $V=\{v_1,v_2,...,v_p\}$ je neprázdna množina vrcholov a $E=\{e_1,e_2,...,e_q\}$ je množina hrán, pozri obr. 5.21. Orientovaný graf sa nazýva *acyklický* vtedy, ak neexistuje orientovaná cyklická cesta obsahujúca postupnosť rovnako orientovaných hrán.

Indexovanie vrcholov orientovaného grafu G je realizované pomocou zobrazenia φ, ktoré priradí 1-1-značne každému vrcholu celé číslo

$$\varphi : V \to \{1,2,\ldots,p\} \tag{5.38}$$

Celé číslo $\varphi(v)$ sa nazýva index vrcholu v. Nech orientovaný graf $\boldsymbol{G}$ je indexovaný pomocou zobrazenia φ, potom matica susednosti $\boldsymbol{A}=(A_{ij})\in\{0,1\}^{p\times p}$ je určená takto

$$A_{ij} = \begin{cases} 1 & \left(\text{pre}\left(v_i,v_j\right)\in E\right) \\ 0 & (\text{inák}) \end{cases} \tag{5.39}$$

matica susednosti orientovaného grafu z obr. 5.21 má tento tvar

$$\boldsymbol{A} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix} \tag{5.40}$$

Pomocou tohto jednoduchého ilustračného príkladu môžeme uzavrieť, že vhodne indexované orientované acyklické grafy sú určené pomocou matíc susednosti, ktoré obsahujú nenulové elementy len pod hlavnou diagonálou (nad diagonálou a včítane diagonály sú len nulové elementy).

Veta 5.2 [3]. Orientovaný graf ***G***=(*V*,*E*) je acyklický vtedy a len vtedy, ak jeho vrcholy môžu byť indexované tak, že platí

$$\forall (v,v') \in E: \quad \varphi(v) > \varphi(v') \tag{5.41}$$

Indexovanie φ, ktoré vyhovuje podmienke (5.41) sa nazýva *kanonické indexovanie*. Podľa vety 1, každý orientovaný acyklický graf môže byť kanonicky indexovaný. Jednoduchý ilustračný príklad kanonického indexovania orientovaného acyklického grafu je uvedený na obr. 5.19, kde vrcholy sú už tak označené, aby podmienka (5.41) boľa splnená. Vrcholy kanonicky indexovaného grafu môžu byť rozdelené na tri disjunktné množiny:

(1) *Vstupné vrcholy*, tieto vrcholy susedia len s vychádzajúcimi hranami,

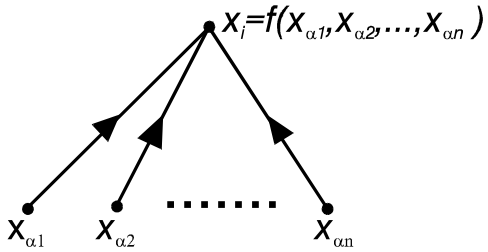
(2) *Prechodné vrcholy*, tieto vrcholy súčasne susedia tak s vzchádzajúcim, ako aj s vchádzajúcou hranou.

(3) *Výstupné vrcholy*, tieto vrcholy susedia len s vchádzajúcimi hranami.

Tieto tri dôležité pojmy môžu byť ilustrované grafom z obr. 5.21, vrcholy v_1, v_2 sú vstupné vrcholy v_3, v_4 sú prechodné, zatiaľ čo vrchol v_5 je výstupný.

Kanonicky indexovaný orientovaný acyklický graf sa nazýva *syntaktický graf*. Tento druh orientovaných grafov má veľký význam pre implementáciu symbolickej regresie, pretože tieto grafy môžu slúžiť ako efektívna reprezentácia funkcií - programov. Študujme znovu orientovaný graf na obr. 5.21. Každý prechodný alebo výstupný vrchol je ohodnotený reálnym číslom, ktoré je určené ako hodnota funkcie priradenej vrcholu, argumenty tejto funkcie sú funkčné hodnoty vrcholov spojené s uvažovaným vrcholom, pozri obr. 5.22. Podľa tejto konvencie, jednotlivé vrcholy grafu na obr. 5.19 sú ohodnotené takto

$$x_5 = c_1,\ x_4 = c_2,\ x_3 = f_3(x_4),\ x_2 = f_2(x_3, x_4, x_5),\ \text{and}\ x_1 = f_1(x_2, x_3) \tag{5.42}$$



Obrázok 5.22. Každý prechodný a výstupný vrchol syntaktického grafu je ohodnotený funkčnou hodnotou s argumnetami odpovedajúcim funkčným hodnotám vrcholov, ktoré sú incidentné s hranami vychádzajúcimi z nich a vchádzajúcimi do uvažovaného vrcholu. Vrcholy, ktoré sú vstupné sú ohodnotené konštantami.

Pre syntaktické grafy, idúc postupne z vstupných vrcholov (ohodnotené "vstupnými" konštantami) cez prechodné vrcholy do výstupného vrcholu počítajú sa rekurentne funkčné hodnoty vrcholov, koncová funkčná hodnota výstupného vrcholu je chápaná ako nejaké ohodnotenie grafu ako takého. Inými slovami, syntaktické grafy realizujú zobrazenie vstupných konštánt na konečnú funkčnú hodnotu výstupného vrcholu

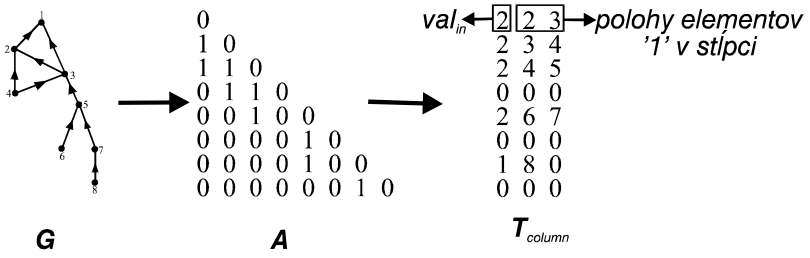
$$x_1 = \boldsymbol{G}(c_1, c_2, \ldots) \tag{5.43}$$

Ako kódovať syntaktické grafy? Ako už bolo naznačené v prvej časti tejto podkapitoly, matica susednosti vyjadrená dolnou trojuholníkovou maticou reprezentuje univerzálny prostriedok pre kódovanie syntaktických grafov. Z vety 1 a z vyššie uvedeného komentáru vyplýva nasledujúca veta.

Veta 5.3. Graf ***G*** je *syntaktický graf* vtedy a len vtedy, ak matica susednosti ***A*** je dolno-trouholníkovou maticou, pričom každý riadok až na prvý obsahuje aspoň jeden jednotkový element '1'.

Podmienka, že v každom riadku až na prvý je aspoň jeden element '1' odpovedá podmienke, že syntaktický graf obsahuje práve jeden výstupný vrchol. Ak v stĺpci pod diagonálou sú len nulové elementy '0', potom v odpovedajúci vrchol je vstupný (t.j. neobsahuje predchodcov). Počet orientovaných hrán je určený pomocou *vstupných* (počet hrán, ktoré vchádzajú do vrcholu) a *výstupných valencií* (počet hrán, ktoré vzchádzajú z vrcholu) všetkých vrcholov grafu

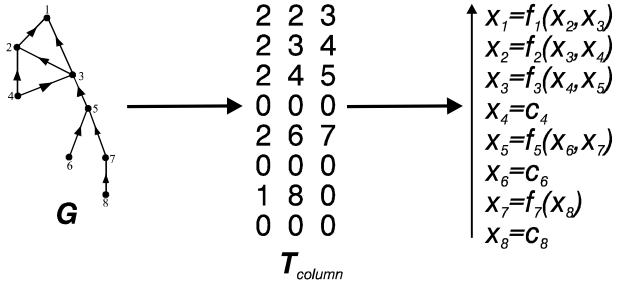
$$q = \sum_{v \in V} val_{in}(v) = \sum_{v \in V} val_{out}(v) \tag{5.44}$$



Obrázok 5.23. Ilustračný príklad kódovania syntaktického grafu pomocou stĺpcovej tabuľky. Syntaktický graf *G* je kódovaný maticou susednosti *A*, táto matica je v ďalšom kroku "kondenzovaná" do tvaru stĺpcovej tabuľky T_{column}.

Použitie matíc susednosti pre kódovanie acyklických orientovaných grafov má jedno vážne ohraničenie, a to skutočnosť, že matica susednosti obsahuje potrebnú informáciu o "topológii" grafu ***G*** vo veľmi "zriedenej forme", t.j. dominantná časť jej elementov sú nulové elementy '0'. Preto obrátime našu pozornosť na "kondenzovanú" maticu susednosti reprezentovanú pomocou stĺpcovej tabuľky $\boldsymbol{T}_{\text{column}}$, pozri obr. 5.23.

Stĺpcová tabuľka $\boldsymbol{T}_{column}$ špecifikuje polohy elementov '1' v stĺpcoch matice susednosti ***A***. V *i*-tom riadku tejto tabuľky sú uvedené polohy elementov '1' v odpovedajúcom *i*-tom stĺpci matice susednosti. Z tejto definície stĺpcovej tabuľky priamo vyplýva jej ekvivalentnosť s maticou susednosti. Syntaktický graf ***G*** určený stĺpcovou tabuľkou $\boldsymbol{T}_{column}$ umožňuje jednoduchý rekurentný výpočet funkčnej hodnoty výstupného vrcholu, pozri obr. 5.24 a algoritmus 5.5.



Obrázok 5.24. Ilustračný príklad výpočtu funkčných hodnôt syntaktického grafu ***G*** určeného pomocou stĺpcovej tabuľky $\boldsymbol{T}_{\text{column}}$**.** Postupujúc zdola-hore cez všetky riadky tabuľky , funkčné hodnoty sú rekurentne počítané pomocou predchádzajúcich funkčných hodnôt.

Pre potreby symbolickej regresie, jednotlivé vrcholy syntaktického grafu musia byť ešte ohodnotené funkciame v súhlase s ich vstupnou valenciou (aritou), t.j. *i*-tému vrcholu je priradená funkcia f_i. Táto požiadavka sa jednoducho realizuje tak, že stĺpcová tabuľka ***T*** je rozšírená o nový tzv. 0-tý stĺpec. Tento prístup bude ilustrovaný syntaktickým grafom na obr. 5.24, pričom jeho vrcholy budú tera ohodnotené jednoduchými algebraickými operáciami, pozri obr. 5.25.

Týmto spôsobom stĺpcové tabuľky sú plne špecifikované, predstavujú jednoduchý a efektívny prístup ku kódovaniu syntaktických grafov. Možno povedať, že predstavujú významné zovšeobecnenie prístupu syntaktických stromov pre kódovanie jednoducho vypočitateľných funkcií - procedúr. Na záver tejto podkapitoly ukážeme jednoduchý postup, ako použiť syntaktické grafy pre potreby symbolickej regresie realizovanej pomocou evolučných algoritmov. Zovšeobecnený pojem stĺpcovej tabuľky bude použitý ako "chromozóm" - elementárna informačná jednotka evolučného algoritmu a budú ukázané elemntárne operácie mutácie a krížemia nad týmito entitami.

```
function Eval_Table(input : i) : real;
begin if Ti1=0 then
      begin {input vertex}
            Eval_Table:=ci
      end else
      if Ti1=1 then
      begin {intermediate or output unary vertex}
            Eval_Table:=fi(Eval_Table(Ti2))
      end else
      if Ti1=2 then
      begin {intermediate or output binary vertex}
            Eval_Table:=fi(Eval_Table(Ti2),Eval_Table(Ti3))
      end;
end;
```

Algoritmus 5.5. Implementácia v pseudopascale výpočtu funkčnej hodnoty výstupného vrcholu pomocou procedúry - funkcie s rekurziou, ktorá pôsobí nad stĺpcovou tabuľkou $\boldsymbol{T}=(T_{ij})$. Funkcia je inicializovaná príkazom `Eval_Table(1)`, jej aktivácia končí na vstupnom vrchole, t.j. $T_{i1}=0$. Predpokladáme, že v syntaktickom grafe každý vrchol má maximálne dvoch predchodcov ($val_{in}^{max}=2$) , tabuľka $\boldsymbol{T}$ má tri stĺpce a p riadkov. Ak je táto funkcia aplikovaná na tabuľku z obr. 5.22, ako výstup dostaneme rovnakú funkčnú hodnotu výstupného vrcholu, ako je uvedená na obrázku. Funkcie f_1, f_2, ...sú priradené jednotlivým vrcholom, reprezentujú požadované funkčné operácie (súčet, plus, krát, zmena znamienka, ...). V prípade vstupných vrcholov, tieto funkcie odpovedajú daným koštantám.

Obrátme našu pozornosť v prvom kroku na taký spôsob určenia stĺpcovej tabuľky, aby jej generovanie bolo dostatočne stochastické a pritom nevyžadovalo komplikovaný opravný proces transformácie náhodne generovanej tabuľky na semanticky korektný tvar. Budeme predpokladať, že počet riadkov v tabuľke je určený číslom (zadanou konštantou) p_{max}, a maximálna vstupná valencia vrcholov syntaktického grafu je v_{in}^{max} . Potom stĺpcová tabuľka je matica typu $p_{max} \times \left(val_{in}^{max}+2\right)$, t.j. obsahuje p_{max} riadkov a $\left(val_{in}^{max}+2\right)$ stĺpcov (ktoré sú indexované 0,1,..., $(val_{in}^{max}+1)$) . Použijeme rovnakú konvenciu ako v obr. 5.23, kde 0-tý stĺpec numericky kóduje typy funkcií priradených jednotlivým vrcholom, 1-vý stĺpec vyjadruje vstupnú valenciu vrcholu, t.j. počet vrcholov - predchodcov v syntaktickom grafe. Konečne, 2-hý a 3-tí stĺpec obsahujú indexy predchodcov, ich počet (včítane nuly) je určený elementami 1-vého stĺpca (vstupnými valenciami). Pretože vrcholy syntaktického grafu musia byť indexované kanonicky, pre elementy i-tého riadku (T_{i1}, T_{i2}) musia vyhovať podmienkam

$$i+1 \le T_{i1} \le p_{max} \text{ , } i+1 \le T_{i2} \le p_{max} \text{ a } T_{i1} \neq T_{i2} \qquad (5.45)$$

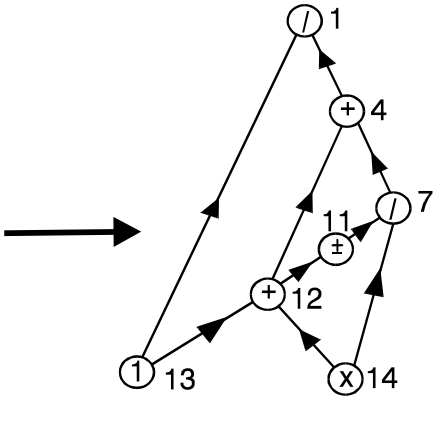
Graf určený takto definovanou tabuľkou nevyhovuje základnej podmienke pre určenie syntaktického grafu, aj keď je orientovaný acyklický graf, môže obsahovať viac ako jeden výstupných vrcholov.

```
procedure Gener_Syntactic_Graph(input : p_max, val_in^max; output T);
begin for i:=1 to p_max do
      begin T_i1:=random(val_in^max+1);
            case T_i1 of
              0 : begin {input vertex}
                        T_i0:=gener_type_function(0);
                  end;
              1 : begin {unary intermediate/output vertex}
                        T_i0:=gener_type_function(1);
                        T_i2:=i+1+random(p_max-i);
                  end;
              2 : begin {binary intermediate/output vertex}
                        T_i0:=gener_type_function(2);
                        repeat
                        T_i2:=i+1+random(p_max-i);
                        T_i3:=i+1+random(p_max-i);
                        until T_i1=T_i2;
                  end;
              .
              .
            end {of case};
      end {of for};
end;
```

Algoritmus 5.6. Implementácia náhodnej generácie stĺpcovej tabuľky obsahujúcej p_{max} riadkov a (val_{in}^{max}+2) stĺpcov. Funkcia `random(n)` je náhodný generátor celých čísel z intervalu [0,*n*-1] s rovnomernou pravdepodobnosťou. cyklus repeat-until je aplikovaný tak dlho, až náhodne generované elementy T_{i2}, T_{i3} vyhovujú podmienke (5.42). Funkcia `genera_type_function(val_in)` náhodne generuje typ funkcie i-tého vrcholu v závislosti od hodnoty vstupnej valencie určenej $val_{in}=T_{i1}$. V prípade, že T_{i1}=0, potom táto funkcia určuje náhodne "vstupnú konštantu".

Náhodne generovanie stĺpcových tabuliek tak, aby boli splnené podmienky (5.42) je

	0	1	2	3
1*	/	2	13	4
2	x	0		
3	*	2	7	15
4*	+	2	12	7
5	1	0		
6	()²	1	15	
7*	/	2	14	11
8	x	0		
9	-	2	15	14
10	±	1	15	
11*	±	1	12	
12*	+	2	13	14
13*	1	0		
14*	x	0		
15	3	0		

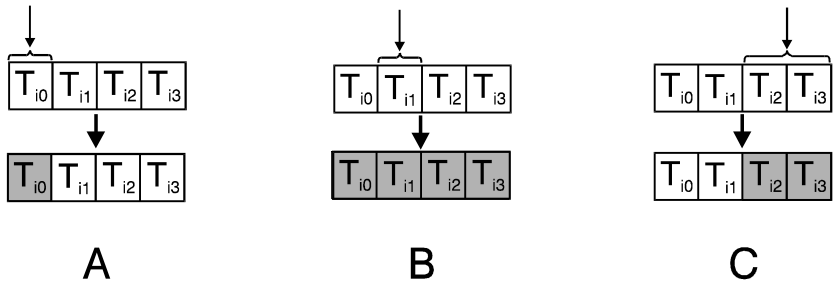


Obrázok 5.26. Ilustratívny príklad náhodne generovanej stĺpcovej tabuľky obsahujúcej 15 riadkov a 4 stĺpce (p_{max}=15 a val_{in}^{max}=2). Hviezdičkou označené vrcholy sú aktívne pri konštrukcii syntaktického grafu obsahujúceho výstupný vrchol '1'. Ostatné vrcholy sa podieľajú na konštrukcii iného (alebo iných) syntaktických grafov, ktorých výstupné vrcholy sú iné ako vrchol '1'. Nakreslený syntaktický graf tiež vznikne aktiváciou funkcie `Eval_Table(1)` z algoritmu 5.5. Z týchto dôvodov môžeme pokladať náhodne generovanú tabuľku za korektný prístup k určeniu syntaktických funkcií, aj keď je nutné poznamenať, že veľká časť tabuľky môže byť nevyužitá pri konštrukcii syntaktickej funkcie.

algoritmicky implementované algoritmom 5.6. Takto generovaná tabuľka tiež určuje syntaktický graf, ak bereme v úvahu len podgraf vytvorený rekurentným riešením vychádzajúceho z výsupného vrcholu '1', pozri obr. 5.26. V takto zovšeobecnenom prístupe ku konštrukcii a interpretácii stĺpcovej tabuľky môže nastať situácia, že väčšia časť tabuľky nie je využitá pre konštrukciu syntaktických funkcií. Ináč povedaná, redundancia informácie v náhodne generovaných stĺpcových tabuľkách je väčšia ako tých, ktoré sú založené na priamej analógii s maticou susednosti (kde je redundancia nulová).

```
procedure Mutation_Table(input : T; output : T');
begin for i:=1 to p_max do
      if random<P_mut then
      begin
            mutation_of_row(i);
      end else
      begin {copy of whole row}
            for j:=0 to T_i1+1 do
            T'_ij:=T_ij;
      end;
end;
```

Algoritmus 5.7. Jedna z možných implementácií mutácie tabuľky **T** na novú tabuľku **T**'. Parameter P_{mut} určuje pravdepodobnosť zmenz riadku v tabuľke, jednotlivé možnosti tejto elementárnej transformácie riadku sú znázornené na obr. 5.27. Takto implementovaná stochastická transformácia - mutácia tabuľky ***T*** vyhovuje podmienke (5.44), t.j. ak pravdepodobnosť P_{mut} sa blíži k nule, potom táto transformácia prestáva mutovať vstupnú tabuľku.



Obrázok 5.27. Schematické znázornenie mutácie vybraného riadku (s pravdepodobnosťou P_{mut}, pozri algoritmus 5.7). Rozlišujú sa tri rôzne prípady vybrané s rovnakou pravdepodobnosťou. V prípade A sa mutuje 0-tý element riadku, ktorý popisuje typ funkcie, t.j. náhodne sa generuje pomocou funkcie `gener_type_function(T_i1)` z algoritmu 5.6. Ostatné elementy v riadku zostávajú nezmenené. V prípade B mutuje 1-vý element riadku popisujúci vstupnú valenciu odpovedajúceho vrcholu. V tomto prípade je potrebné zmutovať aj ostatné zostávajúce elementy riadku. V poslednom prípade C sa mutujú elementy riadku, ktoré popisujú predchodcov vrcholu.

Mutácia tabuľky je chápaná ako operácia, ktorá stochasticky zmení vstupnú tabuľku na novú - výstupnú tabuľku

$$\boldsymbol{T}' = O_{mut}(\boldsymbol{T}) \tag{5.46}$$

pričom stochastičnosť tejto transformácie je vyjadrená *pravdepodobnosťou* P_{mut} pričom nasledujúca podmienka je splnená

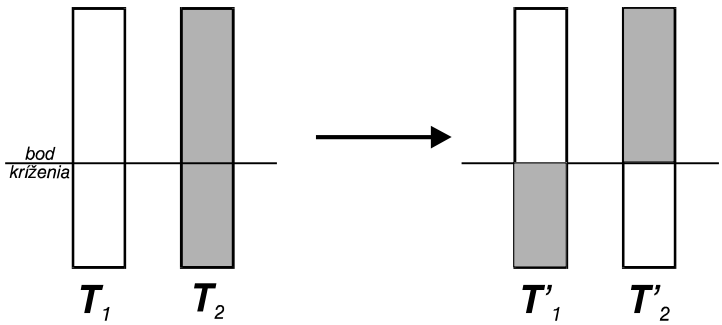
$$\lim_{P_{mut}\to 0} O_{mut}(\boldsymbol{T}) = \boldsymbol{T} \tag{5.47}$$

Možná implementácia mutácie, ktorá vyhovuje podmienke (4.44), je vyjadrená algoritmom 5.7 a obr. 5.27.

Kríženie dvoch tabuliek $\boldsymbol{T}_1$ a $\boldsymbol{T}_2$ je stochastická transformácia, ktorá vytváre dve nové tabuľky $\boldsymbol{T}'_1$ a $\boldsymbol{T}'_2$

$$(\boldsymbol{T}'_1, \boldsymbol{T}'_2) = O_{cross}(\boldsymbol{T}_1, \boldsymbol{T}_2) \tag{5.48}$$

Stochastičnosť tejto operácie v tom, že tzv. bod kríženia sa vyberie náhodne a od tohto bodu si tabuľky vymenia svoje riadky, pozri obr. 5.28.



Obrázok 5.28. Schematické znázornenie operácie kríženia medzi dvoma tabuľkami. Bod kríženia sa náhodne vyberie, potom si tabuľky medzi sebou vymenia riadky, ktoré ležia pod bodom kríženia.

Je dôležité poznamenať, že oprácia kríženia nemení charakter tabuliek, t.j. ak vstupné tabuľky $\boldsymbol{T}_1$ a $\boldsymbol{T}_2$ sú interpretovateľne syntaktickými grafmi, potom aj výstupné tabuľky $\boldsymbol{T'}_1$ a $\boldsymbol{T'}_2$ majú túto dôležitú vlastnosť.

Príklad 5.9. *Modifikujte implementáciu z príkladu 5.7 tak, aby miesto lineárneho kódu sa požívali stĺpcové tabulky.*

5.6 Ilustratívne príklady symbolickej regresie

Symbolickú regresiu budeme ilustrovať pomocou dvoch rôznych výpočtov, a to konštrukcie boolovskej funkcie pre tzv. problém parity a symetrie a konštrukcie 1-rozmernej funkcie reálnej premennej, ktoré sú zadané pomocou regresnej tabulky.

Parita a symetria binárnych vektorov

Nech $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ je n-rozmerný binárny vektor, hovoríme, že tento vektor je párnej (nepárnej) parity ak obsahuje párny počet (nepárny počet) jednotiek, párna parita je definotoricky použitá aj pre binárny vektor obsahujúci samé nuly. Táta charakteristika binárnych vekltorov môže byť chápaná ako boolovská funkcia, ktorá zobrazuje nezávislé binárne premenné $\alpha_1, \alpha_2, \ldots, \alpha_n$ na binárnu závislú premennú (1 pre párnu paritu a 0 pre nepárnu paritu)

$$y = \mathit{parity}(\alpha_1, \alpha_2, \ldots, \alpha_n) \tag{5.49}$$

Pre n=2,3,4 je funkcia parity všetkých možných binárných vektorov určená regresnými tabulkami uvedenými v Tab. 5.1. Parita binárneho vektora môže byť formálne vyjadrená takto

$$\mathit{parity}(\boldsymbol{\alpha}) = 1 + \left(\sum_{i=1}^{n} \alpha_i \right)_{mod\,2} \tag{5.50}$$

To znamená, že parita binárneho vektora $\boldsymbol{\alpha}$ je určená ako jeden plus modulo 2 počtu jednotiek v binárnom vektore $\boldsymbol{\alpha}$. Tak napríklad, ak $\boldsymbol{\alpha}$ obsahuje párny počet jednotiek, potom modulo 2 tohto počtu je nula, čiže podľa vyššie uvedenej definície parita je rovná parity($\boldsymbol{\alpha}$)=1+0=1.

Symetria binárneho vektora $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ je definovaná nasledujúcim spôsobom

$$\mathit{symmetry}(\boldsymbol{\alpha}) = \begin{cases} 1 & (\text{ak } \alpha_i = \alpha_{n-i+1} \text{ pre } i = 1,2,\ldots k) \\ 0 & (\text{ostatné prípady}) \end{cases} \tag{5.51}$$

kde $n=2k$ alebo $n=2k+1$. Verbálne formulované, binárny vektor $\boldsymbol{\alpha}$ ma jednotkovú symetriu vtedy, ak má "stred symetrie". Pre n=2,3,4 je funkcia symetrie všetkých možných binárných vektorov určená regresnými tabulkami uvedenými v Tab. 5.1.

Boolovskú funkciu *parity* môžeme vyjadriť pomocou negácie (*not*) a vylučovacieho alebo (exclusive or, *xor*) (pozri Tab.5.2 pre definíciu týchto funkcií) nasledujúcim jednoduchým spôsobom

$$\begin{aligned} parity(\alpha_1,\alpha_2,\ldots,\alpha_n) &= not(\alpha_1\ xor\ \alpha_2\ xor\ \ldots xor\ \alpha_n) \\ &= (\alpha_1\ xor\ \alpha_2\ xor\ \ldots xor\ \alpha_{n-1})\ xor\ (not\ \alpha_n) \end{aligned} \tag{5.52}$$

Pre $n \geq 2$. Pre n=2,3,4 dostaneme

$$\begin{aligned} parity(\alpha_1,\alpha_2) &= not(\alpha_1\ xor\ \alpha_2) \\ &= \alpha_1\ xor\ not\ \alpha_2 \end{aligned} \tag{5.53a}$$

$$\begin{aligned} parity(\alpha_1,\alpha_2,\alpha_3) &= not(\alpha_1\ xor\ \alpha_2\ xor\ \alpha_3) \\ &= (\alpha_1\ xor\ \alpha_2)\ xor\ (not\ \alpha_3) \end{aligned} \tag{5.53b}$$

$$\begin{aligned} parity(\alpha_1,\alpha_2,\alpha_3,\alpha_4) &= not(\alpha_1\ xor\ \alpha_2\ xor\ \alpha_3\ xor\ \alpha_4) \\ &= (\alpha_1\ xor\ \alpha_2 xor\ \alpha_3)\ xor\ (not\ \alpha_4) \end{aligned} \tag{5.53c}$$

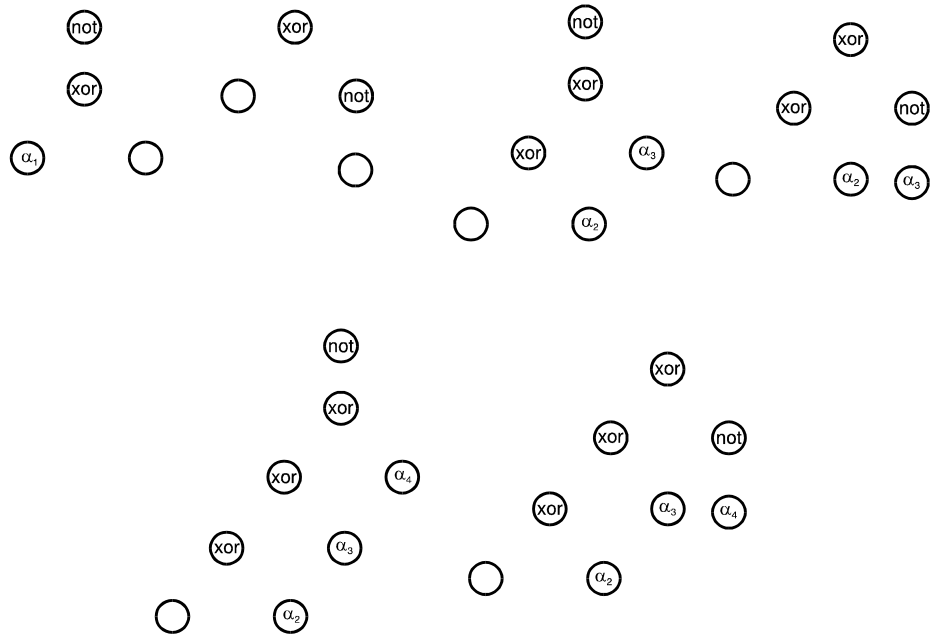
Tabulka 4.1. Regresné tabulky pre paritu a symetriu binárnych vektorov dimenzie n=2,3,4

No.	n=2				n=3					n=4					
	α_1	α_2	y_{par}	y_{symm}	α_1	α_2	α_3	y_{par}	y_{symm}	α_1	α_2	α_3	α_4	y_{par}	y_{symm}
1	0	0	1	1	0	0	0	1	1	0	0	0	0	1	1
2	0	1	0	0	0	0	1	0	0	0	0	0	1	0	0
3	1	0	0	0	0	1	0	0	1	0	0	1	0	0	0
4	1	1	1	1	0	1	1	1	0	0	0	1	1	1	0
5					1	0	0	0	0	0	1	0	0	0	0
6					1	0	1	1	1	0	1	0	1	1	0
7					1	1	0	1	0	0	1	1	0	1	1
8					1	1	1	0	1	0	1	1	1	0	0
9										1	0	0	0	0	0
10										1	0	0	1	1	1
11										1	0	1	0	1	0
12										1	0	1	1	0	0
13										1	1	0	0	1	0
14										1	1	0	1	0	0
15										1	1	1	0	0	0
16										1	1	1	1	1	1

Tabulka 5.2. Hodnoty unárnej funkcie *not* a binárnych funkcií *or*, *xor* a *and*.

x_1	x_2	*not* x_1	x_1 *or* x_2	x_1 *xor* x_2	x_1 *and* x_2
0	0	1	0	0	0
0	1	1	1	1	0
1	0	0	1	1	0
1	1	0	1	0	1

Diagramatická vizualizácia týchto výrazov pomocou syntaktických stromov je znázornená na obr. 5.29.



Obrázok 5.29. Syntaktické stromy pre Boolovskú funkciu *parity* vyjadrenú výrazmi (5.53a-b).

Boolovská funkcia *symmetry* je určená pomocou (5.51), prepis pomocou logických funkcií má tento jednoduchý tvar

$$symmetry(\alpha_1,\alpha_2,...,\alpha_n)=\begin{cases} not\Big(\big(\alpha_1\ xor\ \alpha_{2k}\big)\ or\ \big(\alpha_2\ xor\ \alpha_{2k-1}\big)\ or...or\big(\alpha_k\ xor\ \alpha_{k+2}\big)\Big) & (\text{pre } n=2k+1) \\ not\Big(\big(\alpha_1\ xor\ \alpha_{2k}\big)\ or\ \big(\alpha_2\ xor\ \alpha_{2k-1}\big)\ or...or\ \big(\alpha_k\ xor\ \alpha_{k+1}\big)\Big) & (\text{pre } n=2k) \end{cases} \tag{5.54}$$

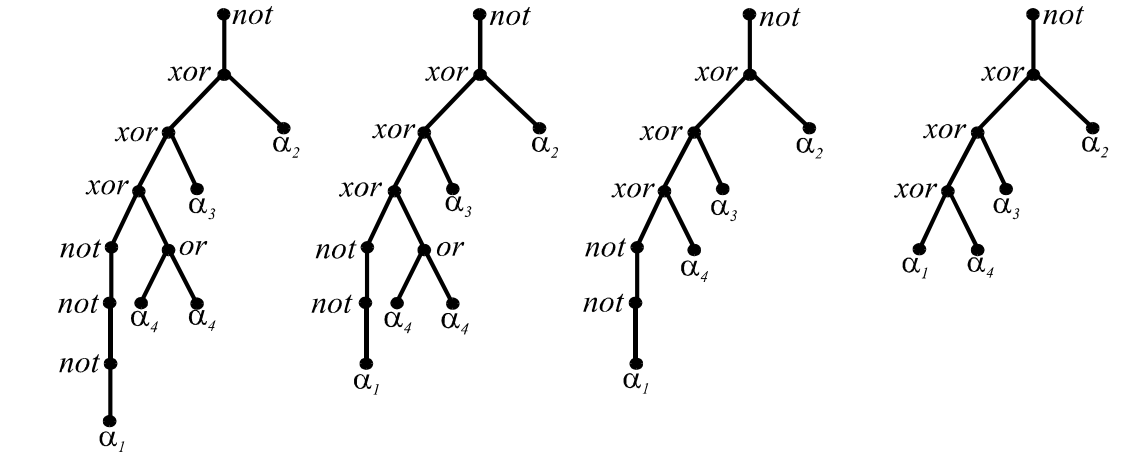
Pre n=2,3,4 táto funkcia má tvar

$$symmetry(\alpha_1,\alpha_2)=not(\alpha_1\ xor\ \alpha_2) \tag{5.55a}$$

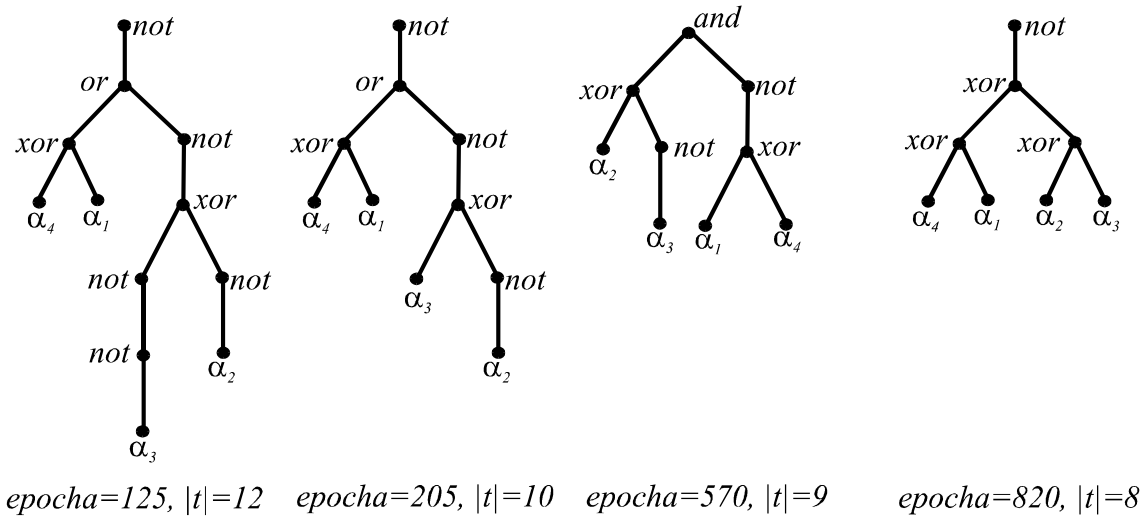
$$symmetry(\alpha_1,\alpha_2,\alpha_3)=not(\alpha_1\ xor\ \alpha_3) \tag{5.55b}$$

$$symmetry(\alpha_1,\alpha_2,\alpha_3,\alpha_4)=not\Big(\big(\alpha_1\ xor\ \alpha_4\big)\ or\ \big(\alpha_2\ xor\ \alpha_3\big)\Big) \tag{5.55c}$$

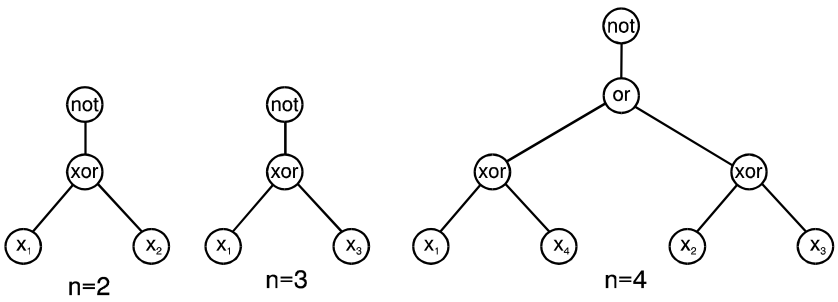
Syntaktické stromy priradené týmto výrazom majú tvar znázornený na Obr. 5.30.



Obrázok 5.31. Znázornenie syntaktických stromov produkovaných genertickým programovaním pre symbolickú regresiu parity binárnych vektorov dimenzie n=4. Všetky tieto stromy majú nulovú hodnotu účelovej funkcie (5.26), penalizačný člen z príkladu (5.7) zvýhodňuje tie stromy, ktoré majú menší počet vrcholov. Posledný strom je totožný so stromom na obr. (5.29), takže môžeme konštatovať, že v tomto jednoducho prípade symbolickej regresie, genetické programovanie poskytuje optimálny výsledok.



Obrázok 5.32. Syntaktické stromy pre paritu binárnych vektorov dimenzie n=4. Posledný strom je totožný so stromom na obr. 5.30. v tomto špeciálnom prípade, genetické programovanie opäť poskytuje správny výsledok.



Obrázok 5.30. Syntaktické stromy Boolovských výrazov vyjadrujúcich symetriu binárnych reťazcov podía výrazov (5.55).

V dôsledku toto, že binárna operácia *xor* je komutatívna a asociatívna, Boolovské funkcie (5.52 a (5.54) sú invariatné vzhľadom k ľubovolnej permutácii binárných premenných $\alpha_1, \alpha_2, ..., \alpha_n$. To znamená napríklad, že syntaktický strom pre n=4 na Obr. 5.30 definuje rovnakú Boolovskú funkciu pre všetkých 4!=24 permutácií premenných $\alpha_1, \alpha_2, \alpha_3, \alpha_4$, .

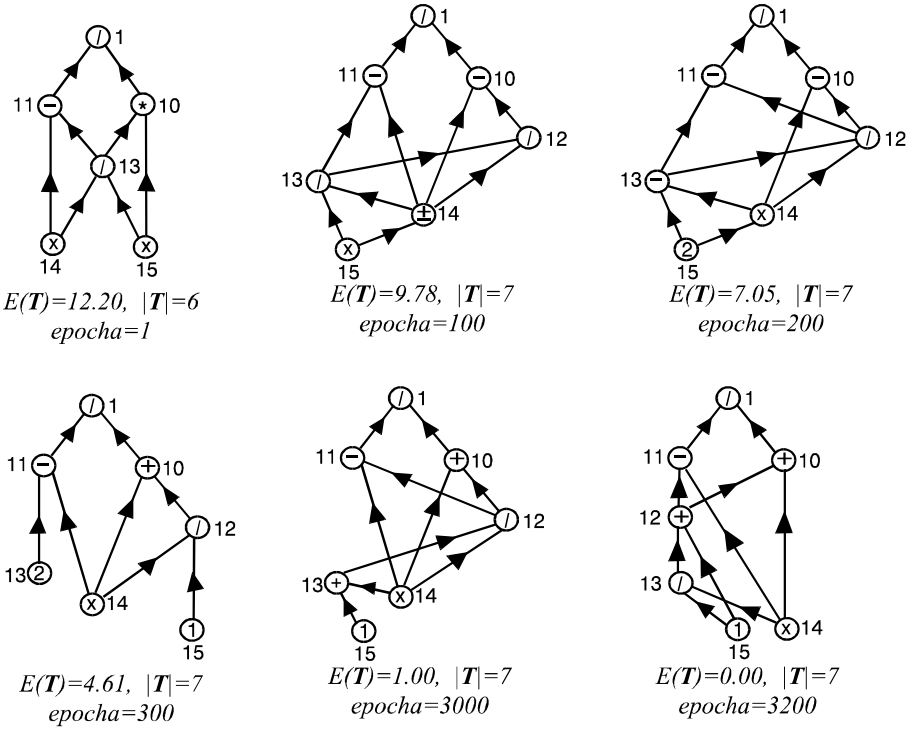
Nech množiny povolených vrcholov majú toto zloženie:

(1) terminálne vrcholy { $\alpha_1, \alpha_2, \alpha_3, \alpha_4$,} ,

(2) unárny vrchol { *not* } ,

(3) binárne vrcholy { *and* , *or* , *xor* } .

Potom nožina T obsahuje také koreňové stromy, ktorých vrcholy patria do vyššie uvedených troch množín vrcholov. Genetický algoritmus (pozri algoritmus 2.5) je teraz modifikovaný tak, ze populácia chromozómov obsahuje syntaktické stromy z množiny T , ktoré pri inicializacii algoritmu boli náhodne generované. Každý syntaktický strom - chromozóm odpovedá Boolovej funkcii, pričom hodnota účelovej funkcie (funkcionálu) (5.22) priradená tomuto syntaktickému stromu je počítané pre danú regresnú tabulku (pozri Tab. 5.1). Genetický algoritmus obsahoval populáciu zloženú z 100 chromozómov (syntaktických stromov zakódovaných pomocou Readovho kódu). Právdepodobnosť operácie kríženia P_{cross}=0.5, operácia mutácie sa aplikuje an každý strom vzstupujúci z kríženia s pravedepodobnoťou P_{mut}=0.1. Účelová funkcia mala modifikovaný tvar z príkjladu 5.7, pričom penalizačná konštanta α=0.01. Výsledky získané touto verziou genetického programovania pre boolovské funkcie parity a symetrie sú uvedené na obr. 5.31 a 5.32.



Obrázok 5.33. Ilustračné príklady syntaktických grafov z priebehu genetického algoritmu, poslednz graf má nulovú účelovú funkciu a preto presne reprodukuje regresnú tabulku.

Racionálny polynóm

V kapitole 5.5 sme popísali kódovanie funkcií pomocou stĺpcových tabuliek, pričom kódované funkcie v tomto prípade sú podstatne všeobecnejšie ako v prístupe kódovania pomocou

koreňových stromov. Budeme predpokladať, žev tomto prípade množiny povolených vrcholov majú toto zloženie:

(1) terminálne vrcholy { celé kladné čísla a premenná x} ,
(2) unárny vrchol { zmena znamienka } ,
(3) binárne vrcholy { súčet, rozdieľ, podieľ a súčin } .

Genetický algoritmus obsahoval populáciu 200 chromozómov (stĺpcových tabuliek, pričom p_{max}=15 a v_{in}^{max}=2). Regresná tabulka bola generovaná pomocou racionálnej funkcie $f(x)=(1+x-x^2)/(1+x+x^2)$ pre 40 ekvidistančných hodnôt premennej $x\in$[-10,10], pre Δx=0.5. Vybrané grafy z priebehu genetického algoritmu sú znázornené na obr. 5.33. Posledný graf na tomto obrázku odpovedá funkcii $((1/x+1)-x)/((1/x+1)+x)$, ktorá po jednoduchých úpravách poskytuje pôvodnú funkciu použitú pre generovanie regresnej tabulky.

Literatúra

[1] J. Koza: Genetic Programming. On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA, 1992.
[2] J. Koza: Genetic Programming (II). Automatic Discovery of Reusable Programs. MIT Press, Cambridge, MA, 1994.
[3] F. Harary: Graph Theory. Addison Wesley, Reading, MA, 1969.
[4] R.C. Read: Coding of Unlabeled Trees, in R.C. Read (ed.), Graph Theory and Computiong. Academic Press, New York, 1972.
[5] V. Kvasnička, J. Pospíchal: Constructive Enumeration of Acyclic Molecules. Collect. of Czech. Chem. Comm. **56**(1991), 1777.
[6] V. Kvasnička, J. Pospíchal: Simulated Annealing Construction of Molecular Graphs with Required Properties. J. Chem. Inf. Comp. Sci. 36(1996), 516.
[7] V. Kvasnička, J. Pospíchal, M. Pelikán: Read's Linear Codes and Evolutionary Computation Over Populations of Rooted Trees. Intellingent technologies (II), Proceedings of the 1-st Slovak Neural Network Symposium, Herlany, November 1996.
[8] M. Pelikán: Genetické programovanie. Práca pre ŠVOČ, CHTF STU, Máj 1996.
[9] M. Pelikán, V. Kvasnička, J. Pospíchal: Read's Linear Codes and Evolutionary Computation Over Populations of Rooted Trees. Genetic Programming '97, Stanford, July 13-15, 1997.
[10] H. Wiener: Structural Determination of Paraffin Boiling Points. J. Am. Chem. Soc. **69**(1947), 17.
[11] M. Randić: On Characterization of Molecular Branching. J. Am. Chem. Soc. **97**(1975),6609.
[12] V. Kvasnička, J. Pospíchal: Simple Implementation of Genetic Programming by Column Tables. Proceedings of Mendel '97, 3rd International Mendel Conference on Genetic Algorithms, Optimization Problems, Fuzzy Logic, Neural networks, Rough Sets. Brno, June 25-27, 1997.
[13] V. Kvasnička, J. Pospíchal: Simple Implementation of Genetic Programming by Column Tables. To be published in Lecture Notes in Computer Science, Springer Verlag, Berlin 1997.