REPREZENTÁCIE V EVOLUČNOM DIZAJNE

JURAJ PLAVČAN

2007

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY
KATEDRA APLIKOVANEJ INFORMATIKY

REPREZENTÁCIE V EVOLUČNOM DIZAJNE

(Diplomová práca)

JURAJ PLAČAN

Štúdijný odbor: Informatika
Diplomový vedúci: Mgr. Pavel Petrovič

Bratislava, 2007

Čestne prehlasujem, že som diplomovú prácu vypracoval samostatne s použitím uvedenej literatúry a s odbornou pomocou diplomového vedúceho.

.....

Juraj Plavčan

POĎAKOVANIE

V prvom rade by som chcel vyjadriť veľkú vďaku môjmu diplomovému vedúcemu Pavlovi Petrovičovi za jeho cenné rady, návrhy a pripomienky, ktorými mi ukazoval správny smer pri tvorbe tejto práca. Bez jeho pomoci pri získavaní výpočtových zdrojov by som nemohol vykonať potrebné experimenty v takom rozsahu.

Ďalej sa chcem poďakovať Marcovi Schoenauerovi, ktorý nám dal do pozornosti článok Marca Ebnera, ktorý sa stal inšpiráciou a východiskom tejto práce.

Pri náročných experimentoch mi poskytnutím výpočtového výkonu svojich počítačov výraznou mierou pomohli: môj brat Jožo, bratranci Martin, Laco a Mišo, ďalej kamaráti Rado, Palo a Renáta. Patrí im za to moja vďaka.

Špeciálne sa chcem poďakovať Tomášovi Grajcárovi za jeho podporu v posledných týždňoch tvorby tejto práce. Svojimi odbornými komentármi a pripomienkami zároveň pomohol zlepšiť kvalitu textu.

V neposlednom rade by som sa chcel poďakovať rodine za oporu a zázemie pri písaní práce.

Abstrakt

Evolučný dizajn hľadá riešenie optimalizačného problému návrhu (dizajnu) tvaru predmetu so žiadanými vlastnosťami – využíva pritom evolučné algoritmy. Dôležitou úlohou v evolučnom dizajne je navrhnúť vhodnú reprezentáciu tvaru trojrozmerného objektu, ktorá musí byť na jednej strane dostatočne všeobecná, na strane druhej musí byť vhodná pre evolúciu.

Táto práca nadväzuje na experimenty Marca Ebnera [1], ktorý skúmal nepriame reprezentácie v podobe scénických grafov. Experimenty predstavovali evolúciu tvaru listu veternej vrtule. Najskôr reprodukujeme jeden z jeho experimentov a potvrdíme jeho výsledky – vhodnosť danej reprezentácie pre evolúciu. Následne navrhneme priamu a novú nepriamu reprezentáciu využívajúcu alternatívnu konštrukciu modelu vrtule pomocou krivej plochy. Pre tieto reprezentácie navrhneme vhodné genetické operátory. Fitness funkcia ohodnocuje jedince v simulácii prúdenia vzduchu cez vrtuľu v jej fyzikálnom modeli. Výsledná kinetická energia vrtule určí fitness jedinca. Výsledky experimentov poukážu na výhody a nevýhody tej ktorej reprezentácie.

Časová náročnosť evolučného výpočtu vyžaduje implementáciu špecializovaného distribuovaného systému. Architektúra *Master-Slave* je vhodná pre evolučné výpočty a umožní vysoké využitie paralelizmu. Vďaka distribuovanému evolučnému výpočtu získame výsledky experimentov v pomerne krátkom čase.

Kľúčové slová: evolučný dizajn, reprezentácie trojrozmerných objektov (priama, nepriama), fyzikálny model a simulácia, distribuovaný evolučný výpočet, genetické programovanie, genetický algoritmus

Obsah

Zoznam obrázkov

Zoznam tabuliek

1 Úvod

Od formulácie problému vedie veľa ciest k riešeniu. Ľudia vyskúšali a overili mnohé z nich, ale neprestali hľadať nové. Inšpiráciu hľadali aj v prírode. Tak vytvorili teóriu *evolučných algoritmov* podľa princípov Darwinovej teórie evolúcie a modernej genetiky. Evolučné algoritmy boli úspešne overené pri štandardných optimalizačných problémoch (v umelej inteligencii), neostali však obmedzené iba na ne. Okrem problému obchodného cestujúceho alebo symbolickej regresie sa ukázali byť aplikovateľnými aj v iných doménach pri zložitejších problémoch.

Práve využitie evolučných algoritmov pri zložitejších úlohách sa stalo našou motiváciou, aby sme preskúmali problematiku *komplexných reprezentácií* v evolučných algoritmoch. Oblasť *evolučného dizajnu* sa stala prostredím nášho bádania. Evolučný dizajn hľadá riešenie optimalizačného problému návrhu (dizajnu) tvaru predmetu so žiadanými vlastnosťami, aplikuje pritom evolučné postupy.

Nadviažeme na prácu Marca Ebnera [1], ktorý skúmal dve *nepriame* reprezentácie tvarov trojrozmerných objektov vzhľadom na vhodnosť ich použitia v procesoch evolúcie. Zameral sa na evolúciu listu vrtule veterného mlyna. Zrekonštruujeme jeho experiment s jeho úspešnejšou reprezentáciou, navrhneme novú *priamu* a *nepriamu* reprezentáciu tvaru listu vrtule a pokúsime sa ich porovnať. Dôležité je poznamenať, že našou ambíciou nie je nájsť optimálny tvar vrtule využiteľný napr. v energetike, ale nájsť vhodnú *reprezentáciu* tvaru vrtule pre evolučný dizajn. Hľadanie dobrého tvaru je cieľom našich evolučných výpočtov, nie tejto práce.

Modely trojrozmerných objektov zakódujeme do reprezentácie pre evolučné algoritmy. Navrhneme genetické operátory kríženia, mutácie a selekcie. Vyberieme vhodné evolučné metódy z rodiny evolučných algoritmov, pomocou ktorých sa pokúsime nájsť čo najvhodnejší tvar listu vrtule v čo najkratšom čase.

V práci priblížime známy pojem počítačovej simulácie a ukážeme jej využitie v evolučnom dizajne. Evolúcia tvaru listu vrtule vyžaduje modelovanie a simuláciu prúdenia vzduchu cez vrtuľu veterného mlyna.

Výsledok simulácie nám číselne určí kvalitu tvaru listu. Hodnota výsledku simulácie je použitá v procese evolúcie ako *fitness* jedinca.

Evolučné výpočty použité pri riešení zložitejších problémov bývajú časovo náročné. Vďaka možnosti zapojiť viacero počítačov paralelne do výpočtu môžeme túto prekážku v užitočnom nasadení evolučných metód prekonať. Pre potreby tejto práce sme navrhli a implementovali *distribuovaný systém*, ktorý koncentroval výkon stovky počítačov na riešenie jedného problému.

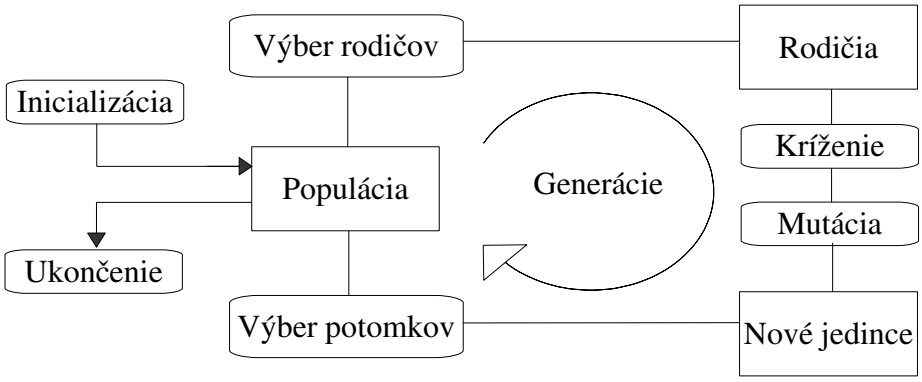
V závere práce zhrnieme výsledky experimentov a zhodnotíme prínos distribuovaného systému.

2 Evolučné algoritmy

Evolučné algoritmy je pojem zastrešujúci viacero metód riešení problémov v umelej inteligencii, ktoré majú isté spoločné vlastnosti. Patria sem *genetické algoritmy* (predstavené Hollandom [4] v 1975 – Ann Harbor, MI), *evolučné programovanie* (L. Fogel [5] – 1965 – San Diego, CA), *evolučné stratégie* (Rechenberg, Schwefel [6] – 1964 – Berlin, Nemecko), *genetické programovanie* (Koza [7] – 1989 – Palo Alto, CA) a *classifier systems* (Holland – 1975).

V pozadí týchto metód je Darwinova teória evolúcie, ktorá sa zakladá na téze prirodzeného výberu, podľa ktorej v prostredí prežívajú s veľkou pravdepodobnosťou len najlepšie prispôsobené jedince populácie [3].

Evolučné algoritmy simulujú evolúciu jedincov v populácii pomocou operátorov selekcie, mutácie a rekombinácie (obr. 2.1). Použitie operátorov je ovplyvnené úspešnosťou (kvalitou) konkrétneho jedinca v prostredí, v ktorom sa nachádza. Prostredie poskytuje len obmedzené zdroje (napr. v prírode: potrava, priestor), a preto tie jedince, ktoré najlepšie vedia súťažiť o tieto zdroje, majú najväčšie šance na prežitie.



Obr. 2.1: evolučný algoritmus

Evolučné algoritmy pracujú s populáciou jedincov, ktorá je modifikovaná selekciou (výberom), krížením (rekombináciou) a mutovaním jedincov tejto populácie. Každému jedincovi je priradená fitness, ktorá číselne vyjadruje jeho kvalitu v danom prostredí. Evolučné algoritmy sú navrhované tak, aby jedince s vysokou hodnotou fitness boli pri reprodukcii preferované. Operátory kríženia a mutácie slúžia na prehľadávanie priestoru všetkých možných jedincov. Mutácia predstavuje tú zložku evolúcie, ktorá umožňuje objavovanie nových vlastností jedincov, ktoré by sa z vlastností existujúcich jedincov nedali získať len za pomoci kríženia. To zabraňuje zároveň konvergenciám jedincov (riešení) v lokálnom optime. V priebehu evolúcie dochádza k zachovávaniu tých vlastností jedincov, ktoré majú tendenciu zvyšovať ich fitness.

Všeobecná schéma evolučného algoritmu (preložené z [10]):

```
t = 0;
Inicializuj_populáciu P(t);
Ohodnoť_populáciu P(t);
pokiaľ nie je splnená ukončovacia podmienka opakuj:
  {
      P'(t) := vyber_rodičov_z P(t);
      rekombinuj P'(t);
      mutuj P'(t);
      ohodnoť P'(t);
      P(t+1) := vyber_novú_generáciu_z P,P'(t);
      t = t + 1;
  };
```

2.1 Reprezentácie v evolučných algoritmoch

Rôzne metódy evolučných algoritmov (spomenuté vyššie a opísané nižšie) historicky vznikli rôznosťou reprezentácií jedincov. V genetických algoritmoch sú jedince reprezentované binárnymi reťazcami, v evolučných stratégiách vektormi reálnych čísel, v evolučnom programovaní konečno-stavovými automatmi a v genetickom programovaní majú jedince stromovú štruktúru. Tieto reprezentácie nie sú jedinými, nad ktorými je možné využiť evolučné metódy. Pri riešení konkrétneho problému je potrebné zvoliť vhodnú reprezentáciu a nad ňou zodpovedajúce operátory.

Reprezentáciu jedinca v evolučnom algoritme nazývame *genotyp* a prislúchajúcu reprezentáciu tohto jedinca v prostredí (problému) nazývame *fenotyp*. Genotyp (doména evolučného algoritmu) determinuje fenotyp (problémová doména). Napriek rozlišovaniu reprezentácií jedincov v algoritme a v prostredí, pri konkrétnych problémoch a algoritmoch sa niekedy genotyp zhoduje s fenotypom (najmä pri genetickom programovaní).

Práve vzťah medzi genotypom a fenotypom jedinca je základom rozlišovania *priamej* a *nepriamej reprezentácie jedinca* v evolučných algoritmoch. Pri priamej reprezentácii ide o priamočiare mapovanie (zobrazenie) genotypu na fenotyp. Najjednoduchšou priamou reprezentáciou je reprezentácia rovnaká pre genotyp aj fenotyp. Pri nepriamej reprezentácii genotyp je predpisom pre vytvorenie fenotypu (príkladom z prírody môže byť genetický kód človeka vo forme DNA chromozómov, ktorý predurčuje fenotypické vlastnosti človeka), ale zobrazenie genotypu na fenotyp nie je priame. Toto zobrazenie (preklad) môže byť v evolučných algoritmoch realizované konečným automatom, prípadne gramatikou.

Závisí od problému, či bude lepšie použiť priamu alebo nepriamu reprezentáciu. Väčšinou je možné použiť obe a rozhodnutie je na riešiteľovi problému. V tejto práci ukážeme riešenie konkrétneho problému pomocou oboch prístupov.

2.2 Riešenie problémov pomocou evolučných algoritmov

Pri optimalizačných problémoch nie je náročné nájsť ľubovoľné riešenie ale riešenie optimálne alebo blízke optimálnemu. Väčšina metód riešení optimalizačných problémov je založená na (slepom alebo cielenom) prehľadávaní priestoru riešení.

V evolučných algoritmoch, ktoré zaraďujeme medzi prehľadávacie metódy, jedinec väčšinou predstavuje nejaké riešenie problému. V populácii máme mnoho rôznych jedincov (z priestoru genotypov). V procese evolúcie sa vytvárajú a zachovávajú schopnejšie jedince, nekvalitné jedince zanikajú a evolúcia prichádza ku kvalitnejším riešeniam problému (v priestore fenotypov). *Fitness funkcia* sa definuje nad priestorom fenotypov. Každému jedincovi priraďuje číselnú hodnotu[1] – mieru výkonnosti, kvality jedinca v prostredí, v ktorom sa nachádza.

1 ide o tzv. skalárnu fitness. Tiež sa používajú aj vektorové fitness, ktoré určujú kvalitu jedinca pri optimalizačných problémoch viacerých premenných (*multi-objective optimization problems*)

Pri *selekcii rodičov* sa zväčša využíva fitness jedincov, aby do procesu reprodukcie vstupovali schopnejšie jedince. Často používanou selekčnou metódou je "turnaj". Pri turnaji veľkosti k sa náhodne vyberie k jedincov, navzájom sa porovnajú a jedinec s najväčšou fitness sa pridá do vytváranej množiny rodičov. Ak chceme získať množinu l rodičov, opakujeme turnaj l-krát. Turnaj sa používa aj pri selekcii nasledovníkov – jedincov pre novú generáciu (pozri nižšie).

Pri *reprodukcii* vznikajú z rodičov nové jedince. Počas *kríženia* (rekombinácie) dochádza k vzájomnej výmene častí genotypov zúčastnených rodičov. Na novovzniknutých jedincov sa aplikuje operátor *mutácie*, ktorý zväčša v malej miere ovplyvní časti genotypu. Operátor mutácie poskytuje nástroj na prekonanie lokálnych extrémov tým, že pôsobí proti konvergenčným tendenciám operátora kríženia (je málo pravdepodobné, že dvaja rodičia z blízkeho okolia lokálneho extrému len pomocou kríženia vytvoria potomka, ktorý sa nebude nachádzať v okolí toho istého extrému).

Na vytvorenie novej generácie jedincov, je potrebné vykonať výber spomedzi nových potomkov (*selekcia potomkov – nasledovníkov*), ku ktorým môžu byť pridaní aj ich rodičia. Výber je uskutočnený často na základe fitness hodnoty. Niektoré algoritmy jednoducho nahrádzajú všetkých rodičov novými potomkami (*Generational replacement* – generačná výmena). Je dôležité zabezpečiť, aby počet potomkov bol rovný počtu rodičov, aby populácia neprerástla alebo nevyhynula. Iné prístupy kombinujú generačnú výmenu so zachovaním najsilnejších rodičov (jedného alebo viacerých) – *elitizmus*.

Evolučný algoritmus býva *inicializovaný* náhodným výberom populácie jedincov z priestoru všetkých genotypov. Proces evolúcie sa vykonáva, kým nie je splnená *ukončovacia podmienka*. Medzi najpoužívanejšie kritéria ukončenia evolúcie patria:

- dosiahnutie maximálneho počtu generácií
- dosiahnutie dostatočného pomeru aktuálnej a cieľovej celkovej hodnoty fitness všetkých jedincov
- dosiahnutie predvolenej minimálnej miery rôznosti jedincov
- dosiahnutie istého počtu po sebe idúcich generácií bez zlepšenia celkovej fitness populácie.

2.3 Výhody a nevýhody evolučných algoritmov

Evolučné algoritmy sú vo všeobecnosti považované za robustnú metódu riešenia (nielen) optimalizačných problémov. Medzi ich výhody patrí ich univerzálnosť – široká paleta problémov, na ktoré je táto metóda aplikovateľná. Taktiež nie sú kladené nároky na dôkladnú znalosť priestoru riešení [2], stačí ak vieme nájsť reprezentáciu genotypu, definovať operátory kríženia a mutácie, inicializovať populáciu a definovať funkciu na výpočet fitness jedinca.

K nevýhodám patrí výpočtová náročnosť – potrebných je veľa[2] generácií, aby evolúcia dospela k dobrým výsledkom. Tiež nie je jednoduché nájsť vhodnú reprezentáciu – genotyp a prislúchajúce operátory, určiť parametre evolúcie (veľkosť populácie, selekčné metódy, pravdepodobnosti aplikovania genetických operátorov, najlepšiu ukončovaciu podmienku). Na rozdiel od problémovo špecifických algoritmov, evolučné algoritmy vo všeobecnosti negarantujú nájdenie optimálneho riešenia problému.

2.4 Genetické algoritmy

V genetických algoritmoch je jedinec reprezentovaný chromozómom, reťazcom nad abecedou (lineárne usporiadaným informačným obsahom jedinca). Názov reprezentácie pochádza z biologickej evolúcie, kde DNA vo forme chromozómov tvorí genetický materiál živých organizmov.

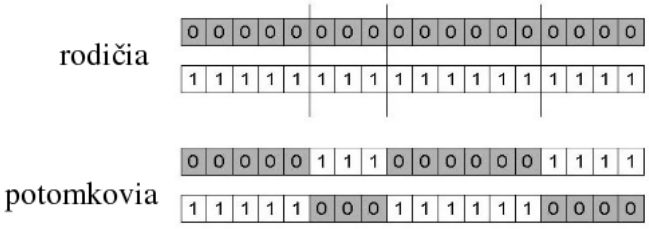
Hollandov [4] pôvodný genetický algoritmus – SGA (Simple genetic algorithm) mal nasledovné atribúty:

Reprezentácia jedinca	binárne reťazce
Rekombinácia	N-bodová alebo uniformná
Mutácia	bitová negácia – bit po bite s konštantnou mierou pravdepodobnosti
Selekcia rodičov	podľa fitness
Selekcia nasledovníkov	generačná výmena
Špecifickosť	dôraz na operáciu kríženia

Tab. 2.1: genetické algoritmy

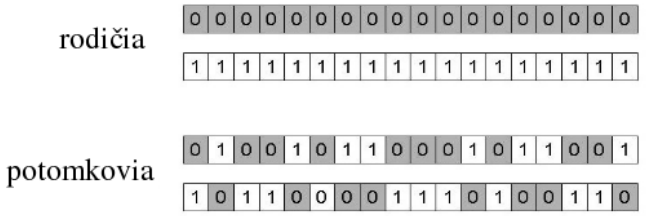
[2] počet generácií zavisí od problému. Platí však, že čím viac generácií, tým väčšia pravdepodobnosť dosiahnutia optimálneho riešenia (alebo blízkeho optimálnemu)

Pri N-bodovom krížení sa náhodne vyberie N bodov (indexov) reťazca. Pri vytváraní nových dvoch jedincov sa postupne kopírujú reťazce z prvého rodiča do prvého potomka, z druhého rodiča do druhého potomka a vo vybraných bodoch si rodičia vymieňajú poradie (obr. 2.2).



Obr. 2.2: príklad N-bodového kríženia pre N=3

Pri uniformnom krížení si prvý potomok náhodne volí rodiča pre každú pozíciu v chromozóme. Druhý potomok kopíruje na každej pozícii toho rodiča, ktorého si nezvolil prvý (obr. 2.3).



Obr. 2.3: príklad uniformného kríženia

Pri bitovej negácií každý bit s pravdepodobnosťou p_M preklopí svoju hodnotu.

Rodičia sa vyberajú na základe ich fitness hodnoty, s cieľom pripúšťať do procesu kríženia schopnejších jedincov. Jedna z často používaných metód, ktoré využívajú hodnotu fitness na výber rodičov, sa nazýva *Ruleta (Roulette Wheel Selection [2])*. Každému jedincovi je priradený na rulete úsek s relatívnou dĺžkou priamo úmernou k jeho fitness hodnote. Pri každom výbere jedného rodiča sa roztočí ruleta a pri zastavení, vyberáme toho jedinca, na ktorého úsek ukazuje šípka. Po výbere dvoch rodičov nastáva kríženie rodičov a mutácia potomkov. Vzniknuté jedince sa pridajú do novej populácie. Tento proces sa opakuje, až kým vytváraná populácia nemá rovnaký počet jedincov ako rodičovská populácia.

Klasický SGA bol neskôr doplnený mnohými inými algoritmami s podobnými vlastnosťami, ktoré prekonávali isté obmedzenia (binárna reprezentácia), a ktoré boli vhodnejšie na konkrétne problémy. Chromozómy môžu obsahovať celočíselné alebo reálne hodnoty namiesto pôvodných dvoch hodnôt (0, 1). Tieto reprezentácie vyžadujú aj nové genetické operátory, ktorých je nad priestorom reálnych alebo celých čísel viac (pravdepodobnostné mutácie hodnôt podľa normálneho, rovnomerného rozdelenia; pri krížení rodičov vzniká potomok aritmetickým priemerov hodnôt rodičov v jednej alebo viacerých pozíciach; využitie váženého priemeru medzi rodičmi zohľadňujúc fitness hodnoty).

2.5 Evolučné stratégie

Evolučné stratégie sa používajú hlavne pri numerických optimalizačných problémoch (najmä nad reálnymi číslami). Ich zvláštnosťou je auto-adaptácia (evolúcia) vnútorných parametrov algoritmu v priebehu evolučného výpočtu. Pôvodne boli evolučné stratégie navrhnuté s populáciou obsahujúcou iba jedného jedinca, neskôr sa zovšeobecnili na populáciu viacerých jedincov. Pri jednom jedincovi v populácii sa používal iba operátor mutácie a počas výpočtu sa uchovávali najviac dva jedince – rodič a potomok.

Ich základné charakteristické vlastnosti sú zhrnuté v tabuľke 2.2

Reprezentácia jedinca	vektory reálnych čísel
Rekombinácia	diskrétna alebo priemerom
Mutácia	podľa normálnej distribúcie
Selekcia rodičov	rovnomerne náhodná
Selekcia nasledovníkov	(μ, λ) alebo $(\mu + \lambda)$ metóda
Špecifickosť	auto-adaptácia veľkosti mutácie

Tab. 2.2: evolučné stratégie

Treba poznamenať, že pri evolučných stratégiach sa rekombinácia často nepoužíva. Dôraz sa kladie hlavne na mutáciu. Operátor mutácie môžeme zapísať nasledovne:

$$\vec{\mathbf{x}}' = \vec{\mathbf{x}} + \mathbf{N}(0, \sigma)$$

kde $\mathbf{N}(0, \sigma)$ je vektor nezávislých náhodných čísel s normálnou distribúciou pravdepodobnosti s nulovou strednou hodnotou a štandardnou odchýlkou σ,

$\vec{x}$ je rodič (vektor reálnych čísel) a $\vec{x}'$ je potomok. Parameter σ nazývame aj veľkosť mutácie a v priebehu evolúcie dochádza k jeho modifikácii podľa *pravidla pätinovej (1/5) úspešnosti* – každých *k* iterácií (generácií) sa σ upraví:

$$\sigma = \sigma / c \qquad \text{if } p_S > 1/5$$

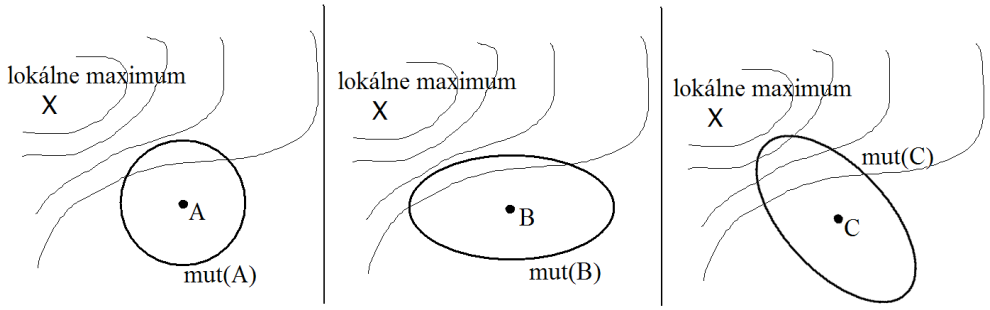
$$\sigma = \sigma \cdot c \qquad \text{if } p_S < 1/5$$

$$\sigma = \sigma \qquad \text{if } p_S = 1/5$$

kde p_S je koeficient úspešnosti definovaný ako pomer počtu úspešných mutácií k celkovému počtu mutácií v priebehu posledných *k* iterácií, $0.8 \leq c \leq 1$

Pravidlo 1/5 úspešnosti sa používa na zvýšenie efektívnosti prehľadávania priestoru jedincov. Pri akceptácií viac ako jednej pätiny potomkov bude prehľadávanie prebiehať vo väčších krokoch (σ je väčšia), pri neúspešnosti sú kroky menšie (σ je menšia). Pôvodne bolo pravidlo vytvorené pre konkrétny problém, neskôr sa však ukázalo, že dáva dobré výsledky aj vo všeobecnosti.

Keďže jedince populácie predstavujú body v mnohorozmernom priestore nad poľom reálnych čísel, je vhodné, aby sa v priebehu evolúcie hodnoty v istých súradniciach menili inou mierou ako v iných súradniciach. Je prirodzené predpokladať, že v istých súradniciach sa jedinec nachádza pri optime, v iných je od neho ďaleko. Preto sa pri mutácií jedinca nepoužíva len jedna hodnota σ, ale pre každú súradnicu jedinca sa uchováva vlastná hodnota σ_i. $\boldsymbol{\sigma}$ je teda vektor, ktorý dopĺňa vektor **x**, a ktorý nesie informáciu o distribúcii pre novú mutáciu v každej súradnici. Pre úplnosť, aby potomkovia smerovali k optimu tak rýchlo ako sa dá, uchovávajú si jedince aj vektor $\boldsymbol{\alpha}$, ktorý obsahuje uhly otočenia jednotlivých osí súradnicového systému. Operátor mutácie si môžeme predstaviť ako okolie jedinca, ktoré určuje pravdepodobných potomkov jedinca. Rozšírenie jedného parametra σ na vektor $\boldsymbol{\sigma}$, ktorý tvorí súčasť evolvovaného jedinca, a následné pridanie vektora $\boldsymbol{\alpha}$ na orientovanie elipsy v priestore tak, aby hlavná os smerovala k optimu (takto bude najviac potomkov vytvorených v dobrom smere) možno lepšie pochopiť z obr. 2.4.



Obr. 2.4: operátor mutácie

A: s jednou hodnotou σ pre každú súradnicu jedinca
B: s vektorom **σ**
C: s vektorom **σ** a vektorom **α**

Pri takto rozšírených evolučných stratégiach sa nepoužíva pravidlo 1/5 úspešnosti. Pre zjednodušený prípad (neuvažujeme vektor **α**) uvedieme pravidlá, podľa ktorých sa mutuje jedinec. Každá smerodajná odchýlka je mutovaná podľa vzťahu

$$\sigma'_i = \sigma_i \cdot exp(\mathbf{N}(0, \Delta\sigma_0))$$

kde $\Delta\sigma_0$ je smerodajná odchýlka pre distribúciu normálneho rozdelenia pravdepodobnosti pre smerodajné odchýlky uchovávané v jedincovi. Je pevná pre všetky smerodajné odchýlky a v priebehu evolúcie sa nemení. Novovytvorená smerodajná odchýlka sa použije hneď na vygenerovanie nového potomka:

$$x'_i = x_i + \mathbf{N}(0, \sigma'_i)$$

Prípad uvažujúci aj natočenie súradnicových osí (vektor **α**) možno nájsť v [3].

Operátor rekombinácie je pri evolučných stratégiách najčastejšie realizovaný diskrétnym krížením alebo krížením priemerom. Pri krížení priemerom potomok vzniká ako aritmetický priemer rodičov. Nemusí ísť len o dvoch rodičov a rodičia môžu byť zvolení nezávisle pre každú súradnicu potomka. Pri diskrétnom krížení si v každej súradnici potomok náhodne vyberie jedinca z množiny rodičov a jeho hodnotu v príslušnej súradnici prevezme. Na rozdiel od genetických algoritmov, operátor rekombinácie štandardne vytvára iba jedného potomka.

Pri vytvorení novej generácie z μ rodičov a λ potomkov sa používa (μ, λ) metóda (ČIARKA) alebo (μ + λ) metóda (PLUS). Pri metóde ČIARKA (λ >

μ) sa do novej generácie vyberá μ najlepších z λ potomkov, bez ohľadu na kvalitu rodičov – ide o generačnú výmenu. Pri metóde PLUS sa do novej generácie vyberá μ najlepších z množiny rodičov a potomkov. V metóde PLUS je zabudovaný *elitizmus* – kvalitný rodičia nevymierajú. Metóda ČIARKA býva preferovaná, pretože má tendenciu ľahšie opustiť lokálne optimum: jedince (rodičia) v lokálnom optime majú väčšinou vyššiu fitness ako priami potomkovia v okolí tohto optima. Preto sú rodičia pri metóde PLUS zachovávaní v populácií na úkor potomkov. Títo rodičia vymrú, až keď sa nájde lepšie lokálne optimum (ale to hľadajú potomkovia a tých je menej). Metóda ČIARKA býva doplnená *miernejším elitizmom (weak elitism)*, ktorý zabezpečí zachovanie najlepšieho rodiča, ak je lepší ako všetci potomkovia. Tým sa dosiahne, že v priebehu evolúcie fitness najlepšieho jedinca neklesá.

2.6 Evolučné programovanie

Evolučné programovanie je veľmi blízke evolučným stratégiam, ale kvôli rozdielnemu historickému pôvodu sa považujú stále za dva evolučné algoritmy. Spoločnou črtou evolučného programovania a evolučných stratégií, ktorá ich odlišuje od genetických algoritmov, je reprezentácia jedinca [3]. Tá sa podriaďuje problému a pre evolučné programovanie neexistuje preddefinovaná reprezentácia. Genotypom býva štruktúra vektorov reálnych čísel.

Pôvodne bolo evolučné programovanie vytvorené na simuláciu evolúcie na populácii súťažiacich algoritmov – konečno stavových automatov. Cieľom bolo predpovedať stavy automatu alebo budúce vstupné symboly. Evolučné programovanie rovnako ako evolučné stratégie kladú dôraz na podobnosť správania sa rodičov a ich potomkov, ako aj na napodobňovanie genetických operátorov podľa vzoru prírody [3]. Rekombinácia sa nepoužíva a jediným evolučným operátorom je mutácia. Jedným z vysvetlení býva uvažovanie o jedincovi ako o (živočíšnom) druhu a medzidruhové kríženie sa nepripúšťa.

Operátor mutácie sa aplikuje na každého rodiča a z každého rodiča vznikne jeden potomok – pri selekcii rodičov sa nevyužíva fitness jedinca. Mutácia je problémovo-špecifická. Pri konečno-stavových automatoch používal Fogel [5] až 5 rôznych mutácií – zmena počiatočného stavu, zmena výstupného symbolu v prechodovej funkcii, zmena výstupného stavu v prechodovej funkcii, pridanie a odobranie stavu. V každej iterácií sa náhodne vybrala jedna z nich. Vo všeobecnosti operátor mutácie využíva zväčša normálne rozdelenie pravdepodobnosti na modifikáciu genotypu jedinca – ako to je pri evolučných stratégiach. Ďalšou podobnou vlastnosťou je auto-adaptácia operátora mutácie.

Pri selekcii nasledovníkov pre ďalšiu generáciu metódou ČIARKA sa zvykne používať turnaj na množine μ rodičov a μ potomkov.

Charakteristiky evolučného programovania sú zhrnuté v tabuľke 2.3.

Reprezentácia jedinca	viaceré vektory reálnych čísel
Rekombinácia	žiadna
Mutácia	podľa normálnej distribúcie
Selekcia rodičov	deterministická – každý jedinec
Selekcia nasledovníkov	(μ + μ) metóda
Špecifickosť	auto-adaptácia veľkosti mutácie

Tab. 2.3: evolučné programovanie

2.7 Genetické programovanie

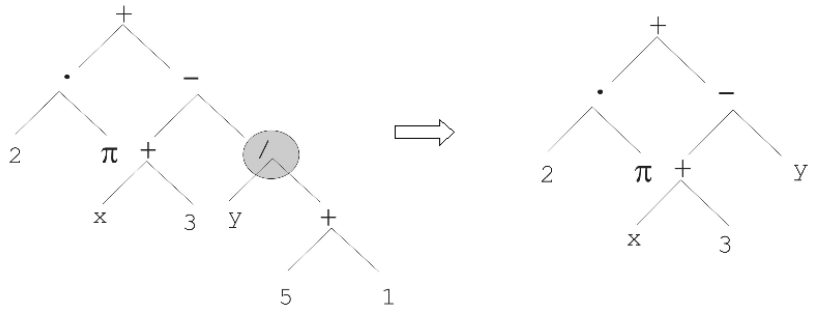
Aritmetické výrazy, logické formuly, počítačové programy a iné môžeme zapísať do stromovej štruktúry. V genetickom programovaní je jedinec reprezentovaný stromom. Všetky vyššie uvedené algoritmy pracovali s lineárnymi štruktúrami (genotyp) konštantnej veľkosti. V genetickom programovaní môžu mať jedince, reprezentované ako stromy, rôznu veľkosť (šírku, hĺbku).

Každý strom (jedinec) definuje symbolický výraz, ktorý môže byť z množiny terminálnych symbolov T alebo z množiny funkčných symbolov F (funkcie rôznej arity). Korektný symbolický výraz je definovaný rekurzívne:

- Každý $t \in T$ je korektný výraz
- $f(e_1, e_2, \ldots e_n)$ je korektný výraz, akk $f \in F$, $arita(f) = n$ a $e_1, e_2, \ldots e_n$ sú korektné výrazy
- Neexistujú ďalšie iné korektné výrazy

Pri inicializácii jedinca (na začiatku evolúcie alebo pri generovaní náhodného podstromu pri mutácií) sa určí maximálna počiatočná hĺbka stromu D_{MAX}. Strom sa generuje tak, že sa uzly v hĺbke $< D_{MAX}$ náhodne vyberú z množiny F a uzly v hĺbke D_{MAX} z množiny T. Možné je túto metódu modifikovať a uzly v hĺbke $< D_{MAX}$ náhodne vyberať z množiny $F \cup T$. Tým získame stromy s rôzne dlhými vetvami. V praxi sa najčastejšie používa metóda RHH (Ramped-Half-and-Half) [11] – polovica populácie sa inicializuje prvou uvedenou metódou a zvyšok populácie druhou uvedenou metódou.

Pri genetických algoritmoch sme uviedli, že v každej iterácii rekombináciou rodičov vznikajú potomkovia, ktorí sú následne modifikovaní s istou pravdepodobnosťou p_M pomocou operátora mutácie (sekvenčná aplikácia genetických operátorov). Pri genetickom programovaní sa miesto postupnej aplikácie oboch operátorov náhodne zvolí, či sa v danej iterácii uskutoční rekombinácia alebo mutácia (paralelná aplikácia genetických operátorov). Najbežnejšou mutáciou je výber náhodného podstromu a jeho nahradenie náhodne vygenerovaným stromom (obr. 2.5).

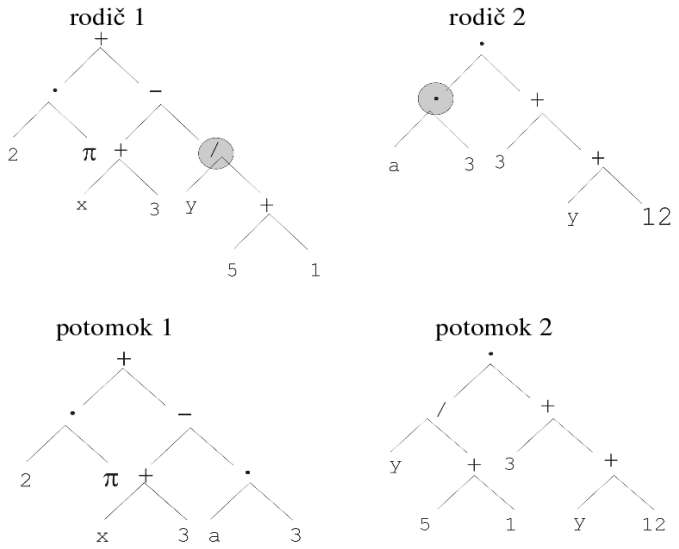


Obr. 2.5: stromová mutácia

V prípade, že sa v danej iterácií zvolí operátor rekombinácie, vyberú sa na základe fitness dvaja rodičia a nastane kríženie. Tým býva najčastejšie vzájomná výmena náhodne vybraných podstromov oboch rodičov (obr. 2.6).

Pri oboch operátoroch môže vzniknutý potomok byť väčší ako rodič – je dôležité mať definované obmedzenia na veľkosť (hĺbku) stromu, alebo stromom s nežiadúcou veľkosťou priradiť nízku fitness (prípadne nulovú).

Pre genetické programovanie je typické pracovať s veľkými populáciami jedincov. Optimalizácia pri selekcii rodičov v podobe *Pareto princípu* sa ukázala byť výhodná z hľadiska úspešnosti evolúcie. Všetky jedince sa usporiadajú podľa fitness a rozdelia sa do 2 skupín. Do prvej patrí najlepších x% z populácie a do druhej zvyšok (100-x)%. Potom 80% výberov rodičov do procesu kríženia alebo mutácie je z prvej skupiny a 20% výberov je z druhej skupiny. Podľa [2] hodnota x závisí od veľkosti populácie a napr. pre populáciu s 1000, 2000, 4000, resp. 8000 jedincami je najlepšie zvoliť x rovné 32%, 16%, 8%, resp. 4%.



Obr. 2.6: stromové kríženie

Alternatívou ku bežnej generačnej výmene pri selekcii nasledovníkov môže byť *elitizmus*, ktorý kvalitných rodičov preferuje na úkor slabších potomkov pri vytváraní novej generácie. Základné znaky genetického programovania môžeme zhrnúť v tabuľke 2.4.

Reprezentácia jedinca	stromová štruktúra
Rekombinácia	výmena podstromov
Mutácia	náhodná zmena v strome
Selekcia rodičov	podľa fitness
Selekcia nasledovníkov	generačná výmena
Špecifickosť	rôzna veľkosť jedincov

Tab. 2.4: genetické programovanie

2.8 Návrh konkrétnych algoritmov

Na riešenie konkrétneho problému pomocou evolučných algoritmov je potrebné vykonať nasledovné:

1. Nájdenie vhodnej reprezentácie jedincov
2. Určenie mapovania genotypu na fenotyp

3. Vytvorenie fitness funkcie na ohodnotenie jedinca
4. Návrh vhodných operátorov mutácie
5. Návrh vhodných operátorov rekombinácie
6. Určenie metódy selekcie rodičov do procesu rekombinácie
7. Určenie spôsobu vytvorenia novej generácie (selekcia potomkov)
8. Definovanie inicializácie evolúcie
9. Definovanie ukončenia evolúcie

Po návrhu a implementácii evolučného algoritmu pre daný problém je potrebné ho spustiť viackrát a na rôznych inštanciách problému. Tak nájdeme vhodné parametre pre jednotlivé operátory. Keďže evolúcia je stochastický proces, eliminujeme tým zároveň nepravdepodobné výpočty. Z jedného výpočtu nie je možné vyvodiť dôveryhodné dôsledky.

2.9 Evolučné algoritmy a knižnica EO

Evolučné algoritmy si od ich prvého predstavenia v 60. rokoch našli uplatnenie v mnohých oblastiach vedy a priemyslu. Vedci z mnohých oblastí však pri problémoch, ktoré chceli riešiť evolučnými algoritmami stáli pred otázkou, či si vytvoria vlastnú implementáciu alebo použijú existujúce knižnice pre evolučné výpočty. Výhodou vlastnej implementácie je možnosť navrhnúť problémovo-špecifický nástroj pre daný problém bez funkčných obmedzení. Medzi nevýhody patrí potreba detailného štúdia teórie evolučných algoritmov, zvýšené sú finančné a časové nároky na implementáciu vlastných nástrojov.

Pre tých, ktorí radšej využijú univerzálne evolučné nástroje vytvorené špecialistami v oblasti, vzniklo viacero balíkov (knižníc) ako napr. GA-lib [29], EO, OpenBEAGLE [30]. K tým rozsiahlejším a univerzálnejším patri **EO (Evolving objects)** [8,9] vytvorená skupinou okolo Marca Schoenauera, Maartena Keijzera, Jeroena Eggermonta a Sebastiena Cahona.

EO je univerzálna open-source platforma na evolučné výpočty, implementovaná v C++ využívajúca objektovo-orientovaný prístup a šablóny (*templates*). Obsahuje hierarchiu tried, ktorá pokrýva všetky najpoužívanejšie spomenuté evolučné metódy. Táto *univerzálnosť* umožňuje relatívne jednoduchú implementáciu evolučného riešenia nad ľubovoľnou doménou objektov – neobmedzuje sa iba na objekty zakódovateľné do binárnych alebo reálno-číselných reťazcov (chromozómov). Ľubovoľný objekt, pre ktorý

navrhneme rekombinačné operátory a účelovú funkciu môžeme začleniť do hierarchie tried v EO. Popri uvedenej *flexibilite* ponúka EO aj preddefinované triedy pre často používané prístupy. Napríklad pre genetické programovanie EO poskytuje triedy pre inicializáciu stromových štruktúr, ich kríženie a niekoľko mutačných operátorov. Na používateľovi zostáva iba definovanie triedy pre vnútorné uzly a listy stromu a účelovú funkciu na vyhodnotenie daného stromu. Vďaka šablónam a dedičnosti používateľ nemusí vo väčšine prípadov modifikovať (na úrovni zdrojového kódu) poskytnutý evolučný mechanizmus (inicializácia populácie, selekcia rodičov, selekcia potomkov, zastavenie evolúcie) a s ním súvisiace komponenty (pozastavovanie evolúcie, pokračovanie staršej evolúcie, štatistiky...). Parametre týchto komponentov sa upravujú externým konfiguračným súborom.

Pre potreby distribúcie[3] evolučného výpočtu sme museli niektoré časti zdrojového kódu upraviť a pridať komunikáciu s databázou.

3 Podrobnejšie v kapitole *Distribuovaný systém*

3 Evolučný dizajn

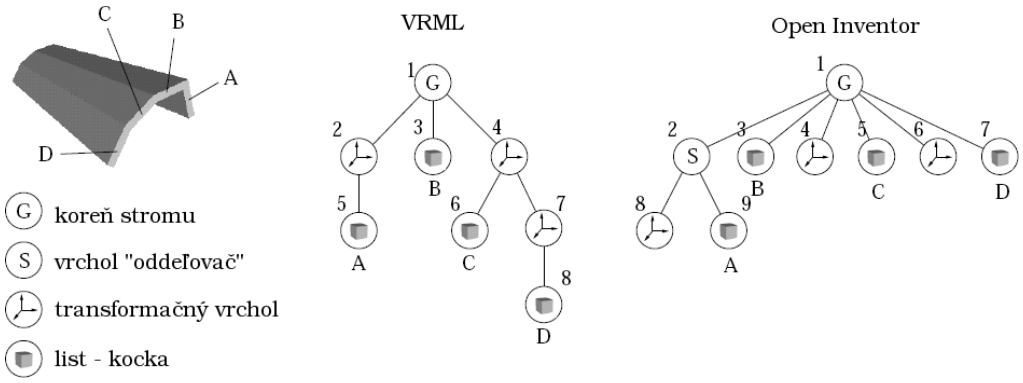
Evolučné algoritmy nachádzajú využitie pri hľadaní štruktúr, ktoré sa približujú k optimálnym pre daný problém. Evolučný dizajn uplatňuje tieto algoritmy na hľadanie vhodných tvarov – tvarov objektov pre priemyselné využitie, konštrukciu rôznych zariadení, súčiastok, pre umelecké predmety a iné.

V evolučnom dizajne jedinec reprezentuje určitý tvar predmetu. Operátory kríženia a mutácia menia tieto tvary. V priebehu evolúcie vznikajú a zachovávajú sa tvary, ktoré viac vyhovujú ako riešenie daného problému. Výsledkom tejto evolúcie je populácia najvhodnejších tvarov. Medzi hlavné výzvy evolučného dizajnu patrí nájdenie vhodnej reprezentácie objektu a evolučných operátorov nad touto reprezentáciou. Potrebné je ich definovať tak, aby bolo možné vytvoriť ľubovoľné tvary (prehľadávanie čím väčšieho priestoru riešení) a aby boli zachované súvislosti medzi jedincami – rodičmi – a jedincami – potomkami – ktoré vznikli krížením a mutáciou z týchto rodičov. Predpokladom v evolučnom dizajne je totiž korelácia veľkosti zmeny genotypu objektu a veľkosti prislúchajúcej zmeny fitness objektu – fitness potomoka by sa nemala výrazne odlišovať od rodičov. Je preto potrebné zabezpečiť (vhodnými evolučnými operátormi), aby v priebehu evolúcie pri reprodukcii rodičovských jedincov vznikali väčšinou rodičom podobné jedince. Inak by bol evolučný výpočet vzhľadom na komplexnosť problémov evolučného dizajnu náhodným prehľadávaním. Keďže fitness funkcia je pre daný problém pevne daná, túto požiadavku je možné splniť len vhodným výberom reprezentácie genotypu. Preto sú reprezentácie pre evolučný dizajn predmetom štúdia tejto práce.

3.1 Ebnerove experimenty

Vytvorených bolo viacero reprezentácií trojrozmerných (3D) objektov najmä v oblasti počítačovej grafiky a virtuálnej reality. Boli skúmané aj v súvislosti s evolučným dizajnom. Marc Ebner [1] sa vo svojej práci zamerial na scénické grafy a porovnal dve existujúce reprezentácie, patriace do triedy scénických

grafov. Jednu používanú v 3D grafickej knižnici Open Inventor a druhú používanú vo VRML – jazyku na modelovanie virtuálnej reality.



Obr. 3.1: príklad zápisu kompozície objektu vo VRML a Open Inventor scénickom grafe (prevzatý z [1])

Obe reprezentácie majú stromovú štruktúru, vnútorné vrcholy predstavujú transformačné operácie a listy stromu jednoduché objekty ako guľa, kapsula[4] alebo kváder. Pri VRML reprezentácií je výsledná transformácia listového objektu definovaná postupným (v danom poradí) aplikovaním transformácií na ceste z koreňa stromu do príslušného listu. Pri Open Inventor reprezentácii transformačné vrcholy môžu byť aj listami stromu a navyše existujú špeciálne vrcholy "oddeľovače". Pri určovaní výsledných pozícií a rotácií elementárnych objektov sa strom prehľadáva do hĺbky a na daný objekt sú aplikované všetky po ceste pozbierané transformácie. Vrcholy "oddeľovače" slúžia na prerušenie zbierania týchto transformácií a predstavujú lokálny akumulátor, do ktorého idú všetky transformácie v podstrome tohto oddeľovača. Transformácie v podstrome oddeľovača sú aplikované len na listy tohoto podstromu (obr. 3.1).

Ako príklad, na ktorom Ebner porovnával vhodnosť spomenutých reprezentácií, použil evolúciu tvaru listu vrtule veterného mlynu. Vytvoril počítačový model trojlistovej vrtule s identickými listami. List vrtule bol zložený z mnohých elementárnych objektov (guľa, kapsula, kocka), ktoré boli umiestnené v priestore podľa transformácií uvedených v scénickom grafe. Počas simulácie modelu[5] častice vzduchu lietali cez vrtuľu istou rýchlosťou,

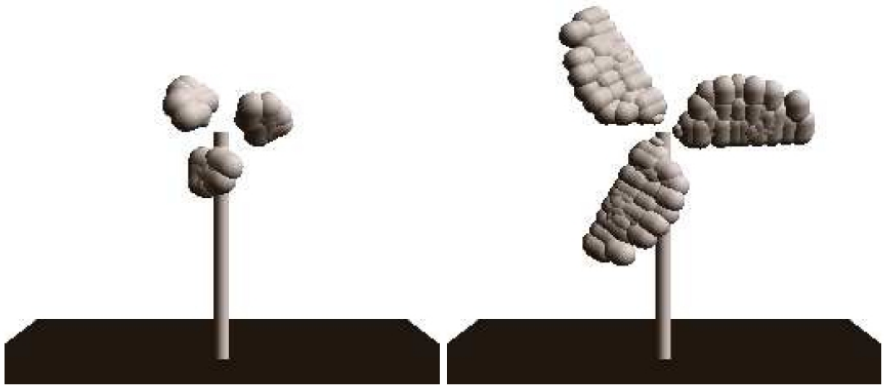
4 kapsula je valec s polguľami na oboch koncoch, polomer polgúľ je rovnaký ako polomer podstavy valca

5 podrobnejšie v kapitole Fyzikálna simulácia

dochádzalo k zrážkam častíc s listami vrtule a prevodu kinetickej energie častíc na rotačnú kinetickú energiu vrtule. Kvalita tvaru listu bola reprezentovaná touto rotačnou energiou.

Na evolúciu scénických grafov – stromových štruktúr použil genetické programovanie. Populácia obsahovala 50 jedincov, selekčnou metódou bol turnaj s veľkosťou 7. Pri inicializácii použil RHH metódu. Vnútorné parametre vnútorných vrcholov stromov (vektor posunutia, os a uhol otočenia) ako aj listov (rozmery umiestnených objektov) boli náhodne zvolené pri inicializácii a počas evolúcie sa nemenili. Štandardné[6] stromové kríženie a mutácia boli aplikované s pravdepodobnosťou 50%. Evolúcia trvala 200 generácií.

Vzhľadom na značnú časovú náročnosť vykonal len 10 evolučných výpočtov pre každú reprezentáciu.



Obr. 3.2: výsledné fitness a tvary listov vrtule (vľavo Open Inventor, vpravo VRML) (prevzatý z [1])

Vo svojej práci ukázal, že VRML scénický graf je lepší ako Open Inventor scénický graf z hľadiska dosahovanej kvality listu vrtule v evolučnom výpočte (obr. 3.2). Svoje závery zdôvodnil tým, že pri VRML reprezentácii podstrom zodpovedá konkrétnym častiam listu vrtule a teda pri rekombinácií a mutácií dochádza k zmenám iba častí listu. Pri Open Inventor reprezentácii mutácia v danom strome môže ovplyvniť transformácie v iných podstromoch a tým rôzne časti listu vrtule – najmä ak nie sú používané uzly "oddeľovače". Rozdiel vo výsledkoch bol značný (aj štatisticky podložené t-testom (t = 2,413)).

V kapitole Experimenty a výsledky – podkapitola Nepriama reprezentácia I – opíšeme náš experiment, ktorý reprodukuje (s istými odlišnosťami) Ebnerov

6 ako uvedené v kapitole *Evolučné algoritmy*

experiment s VRML reprezentáciou. Iné spôsoby dizajnu tvaru vrtule budú motiváciou pre experimenty opísané v podkapitolách Priama reprezentácia a Nepriama reprezentácia II.

4 Fyzikálna simulácia

V tejto kapitole predstavíme základy počítačovej simulácie modelu fyzikálneho sveta, ktorá je hlavnou zložkou fitness funkcie v evolučnom algoritme.

Keďže v práci sledujeme evolúciu tvarov objektov reálneho sveta, konkrétne listov vrtule, je nevyhnutné, aby sme vhodnosť evolvovaného tvaru posudzovali vzhľadom na reálne podmienky prostredia. Jedným z prístupov k tomuto problému by bola fyzická konštrukcia danej vrtule a jej následné otestovanie v skutočnom prostredí. Tento veľmi nepraktický prístup bol vo svete počítačov nahradený *počítačovými simuláciami modelov* sveta. Počítačová simulácia je program, ktorý simuluje skutočný systém prostredníctvom abstraktného modelu. V ďalšom budeme pod pojmom simulácia rozumieť počítačovú simuláciu (program).

Rôznosť systémov, ktoré sú predmetom modelovania a simulácie, je dôvodom vzniku rôznych modelov sveta (fyzikálny model, model spoločenstiev organizmov, model trhu v ekonomike, model počítačových sieti, atď). My sa budeme zaoberať jedným z fyzikálnych modelov (existuje viacero v závislosti od predpokladov, obmedzení modelu a uvažovaných fyzikálnych zákonov).

4.1 Fyzikálna simulácia v počítači

Fyzikálny simulátor uvažuje model, ktorý zahŕňa fyzikálne zákonitosti akými sú napríklad vzájomné pôsobenie objektov prítomných v modeli a simuluje dynamiku tejto interakcie. Objekty majú isté vlastnosti – pozícia v priestore, orientácia, rýchlosť, hmotnosť, objem, tvar – a pôsobia silami pri interakcii s inými objektami. Vzhľadom na zložitosť skutočného prostredia, simulátor abstrahuje od mnohých zanedbateľných atribútov a zameriava sa len na tie, ktoré majú podstatnejší vplyv na simulovanú úlohu (v našom prípade napríklad neuvažujeme vzájomné pôsobenie medzi časticami vzduchu, a tiež gravitačné sily nemajú význam pre nás, a preto sa pre jednoduchosť a rýchlosť simulácie zanedbávajú).

Fyzikálna simulácia má zväčša (a tak je to v našom prípade) *spojitý charakter* – stav objektov simulovaného modelu sa mení spojite. Naopak pri modeli bankových transakcií môžeme hovoriť o *diskrétnej simulácii*. Stav modelu sa mení v istých udalostiach (*events*) – oddelených časových okamihoch (výber, vklad hotovosti). Spojitý charakter býva vyžadovaný napr. pri simulácií letu lietadla, pri modelovaní tepelných vlastnosti skúmaného materiálu. V mnohých prípadoch sa spojitý charakter modelu aproximuje v počítačovej simulácii diskrétnym modelom s malými časovými krokmi. Systém sa vyvíja v čase prostredníctvom reprezentácie, v ktorej sa stav premenných mení v okamihoch medzi krátkymi časovými intervalmi (matematicky by sme mohli povedať, že systém sa mení iba v konečnom počte časových okamihov – udalostí). Veľkosť týchto intervalov môže byť konštantná (*fixed-increment time advance*), alebo variabilná – určená významnými udalosťami v modeli (*next-event time advance*). Pri simuláciach v našich experimentoch používame prvý prístup.

Simulácia môže byť *deterministická* alebo *stochastická*. Pri stochastickej simulácií je v systéme zahrnutý komponent, ktorý generuje (pseudo)náhodnú informáciu potrebnú v simulácii. V deterministických modeloch je výsledok simulácie určený vstupom, ale napriek tomu deterministická simulácia môže byť jedinou metódou, ako určiť tento "vypočítateľný" výstup. V našej práci používame (pseudo)náhodný generátor a teda ide o schochastickú simuláciu.

Simulácia sa vykonáva, kým nie je splnená ukončujúca podmienka (zohľadňujúca stav premenných modelu), alebo kým sa nedosiahne vopred definovaný počet (časových) krokov.

Pri predstavovaní simulácií treba uviesť aj iný prístup k riešeniu problému, ktoré buduje model skúmaného problému. Ide o *analytické riešenie* numerickými metódami. Tento prístup je možný zväčša len pri dostatočne jednoduchých problémoch, ktoré sú opísateľné, dostupnými metódami riešiteľnými, rovnicami.

4.2 Minimalistické simulácie

Spomenuli sme, že počítačové simuláciu pomáhajú pri problémoch, ktoré by inak vyžadovali konštrukciu reálneho modelu. Napríklad pri experimentoch [15] s evolúciou tvaru robota a jeho správania v určitých podmienkach sa využívajú počítačové simulácie na ohodnotenie jeho výkonu v počítačovom modeli. Problémom avšak stále zostáva nájdenie vhodného modelu a procesu simulácie tak, aby model čím vernejšie reprezentoval modelovaný systém a

zároveň, aby rýchlosť simulácie bola dostatočná, aby celé modelovanie a simulácia mali vôbec zmysel (ak by bolo časovo veľmi náročné simulovať robota, napr. by simulácia trvala dlhšie ako činnosť skutočného robota v reálnom svete, tak prínos modelovania a simulácie by bol otázny). Vzhľadom na komplexnosť skutočného prostredia a obmedzeného výkonu súčasných počítačov, model systému musí abstrahovať od mnohých detailov.

Ako vybrať atribúty sveta, ktoré nebudú v modeli prítomné? Ohodnotíme vplyvy jednotlivých atribútov a odstránime z modelu tie, ktoré majú zanedbateľnú hodnotu. V našom prípade sme model vzduchu aproximovali časticovým modelom (vzhľadom na možnosti použitého simulátora sme nemodelovali aerodynamiku). V tomto modeli sme neuvažovali vzájomné gravitačné pôsobenie medzi jednotlivými časticami vzduchu, nedochádzalo k turbulentnému prúdeniu častíc. Zanedbávali sme gravitačné pôsobenie Zeme. Pri zrážke častice s listom vrtule sme v mieste zrážky aplikovali jednoduchý Coulombov model trenia (pozri nižšie). Materiál listu vrtule bol neforemný.

Týmito abstrakciami a aproximáciami vytvárame zjednodušený model – *minimalistický model* systému. Simulácie minimalistického modelu nazývame *minimalistické simulácie*.

4.3 ODE

Existuje viacero fyzikálnych simulátorov ako komerčných, tak aj zo sveta open-source. Nebudeme sa venovať porovnaniu viacerých dostupných nástrojov (*Newton Dynamics*™ [16], *Havok Physics*™ [17], ...), ale pozrieme sa detailnejšie na najpopulárnejší open-source projekt – *Open Dynamics Engine (ODE)* [13], ktorý použil aj Marc Ebner vo svojej práci. ODE sa používa hlavne na interaktívnu simuláciu v reálnom čase ako napr. v počítačových hrách a robotických simulátoroch (*SimRobot* [31], *UCHILSIM* [32], Ubersim [33], *Webots* [34]). Uprednostňuje rýchlosť a stabilitu pred fyzikálnou presnosťou a korektnosťou, čo má pozitívny (rýchlosť) aj negatívny (fyzikálne nekorektný model, nepresnosti pri výpočtoch) vplyv na našu úlohu.

Tento produkt sme využili na simulovanie prúdenia častíc vzduchu okolo listov vrtule, ich kolíziu s listami vrtule a jej následne roztočenie.

Simulátor ODE pozostáva z dvoch hlavných komponentov – *Podsystém detekcie kolízií* (*Collision Detection Subsystem*) a *Podsystém fyzikálnej dynamiky* (*Physics Engine Subsystem*). Druhý uvedený pracuje s telesami (*rigid body*) – objektami s atribútmi pozícia, orientácia, rýchlosť

(lineárna/uhlová), hmotnosť, pozícia ťažiska. Telesá môžu byť izolované, alebo môžu byť spojené s iným telesom (pomocou spojenia/kĺbu (*joint*)) a opakovaným spájaním vytvárať zložené objekty. Telesá môžu byť v danom časovom kroku „vypnuté" a nezúčastňovať sa simulácie až do doby, keď sú znova „zapnuté".

Funkciami môže užívateľ aplikovať na telesá sily, ktoré sú pridané do silového akumulátora telesa (*force accumulators*), vyhodnotené v danom kroku simulácie a spôsobujú pohyb/rotáciu telies. Po každom kroku simulácie sú akumulátory vynulované. Na výslednú silu pôsobiacu na teleso spojené s iným telesom a jeho následnú zmenu pozície, orientácie a/alebo rýchlosti vplývajú obmedzenia (*constraints*) podľa použitých spojení medzi telesami.

Podsystém detekcie kolízií má za úlohu sledovať vzájomnú pozíciu telies a zabezpečiť, aby telesa do seba neprenikali, alebo sa cez seba nepohybovali. Na to je potrebné telesám priradiť ďalšie atribúty ako objem, tvar, mäkkosť, pružnosť. Geom reprezentuje teleso (alebo jeho časť) a definuje jeho objem a tvar. Mäkkosť a pružnosť sa určuje pri konkrétnom dotyku (zrážke) dvoch geomov. Pri dotyku vzniká v danom kroku simulácie špeciálny spoj (*contact joint*). Tento spoj má životnosť jedného časového kroku a predstavuje tzv. kolízne obmedzenie (*collision constraint*), ktoré v dynamickom podsystéme dopĺňa užívateľom definované vzájomné obmedzenia medzi telesami (*joint constraint*). Tieto dve triedy obmedzení určujú vzájomnú interakciu telies zúčastnených v kolízii.

V ODE simulátore môžu mať telesá (geomy) tvary jednoduché – kváder, guľa, kapsula[7] alebo zložitejšie – definované krivou plochou (triangulovanou mriežkou - *TriMesh*). Spojenia medzi telesami môžu mať viac stupňov voľnosti a rozlišujeme napr. guľový spoj, pánt, posuvný spoj, pevný spoj.

ODE používa jednoduchý Coulombov model trenia v kontaktoch dvoch telies (pri našich experimentoch išlo o kolízie častíc vzduchu s listami vrtule):

$$F_T = \mu F_N$$

kde μ je Coulombov koeficient, F_T je maximálna tangenciálna sila pôsobiaca proti pohybu telesa a F_N je normálová sila pôsobiaca kolmo na dotykovú plochu telies. Tento model je pre efektívnosť zjednodušený a abstrahuje od veľkosti normálovej sily – maximálna trecia sila je určená len parametrom μ. Pri našich simuláciách sme použili $\mu = 10$. Pri menších hodnotách dochádzalo k nerealistickým odrazom častice od listu vrtule. Pri väčších hodnotách častica odovzdávala málo svojej kinetickej energie vrtuli, lebo zrážka spôsobovala

[7] valec s polguľami na oboch koncoch, polomer polgúľ je rovný polomeru podstáv valca

príliš veľkú rotáciu častíc, keď nedopadli na list vrtule kolmo – tiež nerealistické.

Typický proces ODE simulácie možno zhrnúť do nasledovnej postupnosti krokov:

1. Vytvorenie „sveta dynamiky“ (*dynamics world*)
2. Vytvorenie telies vo svete a definovanie ich stavov (atribútov)
3. Vytvorenie spojení medzi telesami vo svete a nastavenie ich parametrov
4. Vytvorenie „kolízneho sveta“ (*collision world*)
5. Vytvorenie geomov a ich asociácia s telesami zo sveta dynamiky
6. Cyklus simulácie – v jednom kroku sa vykoná:
    - Aplikuj užívateľom definované sily na telesá
    - Uprav parametre spojov, aby sa zachovali obmedzenia medzi telesami
    - Podsystém detekcie kolízií zistí dotyky telies
    - Vytvorenie kontaktných spojov pre každý bod dotyku
    - S definovanými silami a spojmi uprav pozície a orientácie telies
    - Zrušenie kontaktných spojov
7. Zrušenie sveta dynamiky a kolízneho sveta a ich objektov

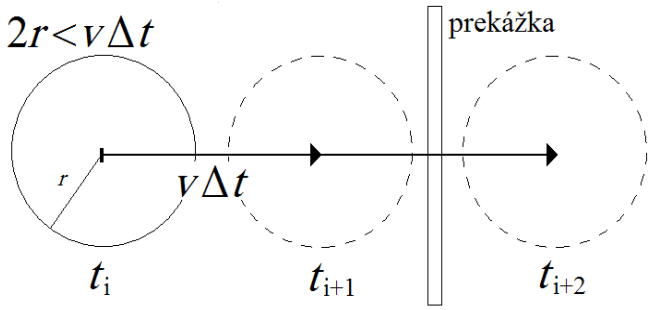
4.3.1 Nedostatky ODE

ODE bolo vytvorené ako základný systém simulácie fyzikálnych telies pre širokú škálu aplikácií. Hlavným cieľom bola rýchla simulácia v reálnom čase, hlavne pohybujúcich sa zložených telies v meniacom sa prostredí virtuálnej reality. Simulátor je robustný a stabilný s prepracovaným ošetrovaním chýb vznikajúcich pri simulácii. Tieto vlastnosti – rýchlosť a stabilita – sú vyvážené fyzikálnou presnosťou.

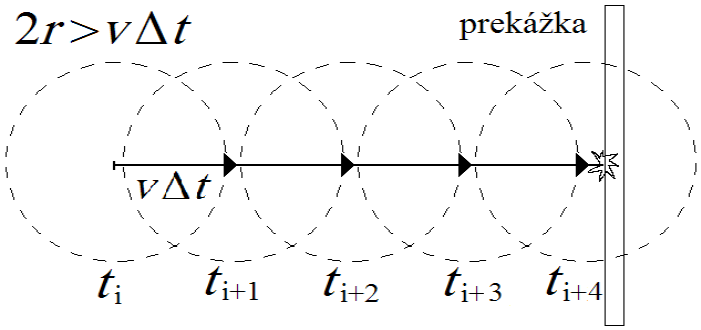
Simulátor používa diskrétny model simulácie. Pozície a pohyb telies sú vyhodnocované a určované v časových krokoch konštantnej veľkosti. Môžme povedať, že telesá sa “teleportujú” v časových krokoch. Pri simulácii telies, ktorých veľkosť je relatívne veľká v porovnaní s dráhami prekonanými[8] v jednom časovom kroku, sú tieto teleportácie (skoky) nepozorovateľné (obr.

[8] skokom

4.2). Ak telesá majú rozmery rádovo porovnateľné s veľkosťou časového kroku vynásobeného okamžitou rýchlosťou telesa[9], tak dochádza k nežiadúcim javom – telesá môžu preskočiť prekážku, nedôjde k očakávanej kolízii (obr. 4.1).



Obr. 4.1: "prelet" častice prekážkou



Obr. 4.2: let častice a následná kolízia

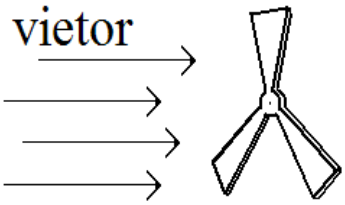
Na prekonanie tohto nedostatku sme použili dostatočne malý časový krok (pre danú veľkosť častice a jej maximálnu rýchlosť), aj keď to nepriaznivo ovplyvnilo rýchlosť simulácie.

4.4 Model vrtule

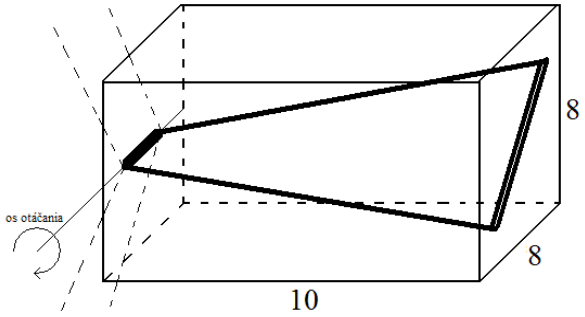
Modelovali sme trojlistovú vrtuľu s horizontálnou osou otáčania (obr. 4.4). List vrtule bol vytvorený na základe konkrétnej reprezentácie a buď bol určený množinou rôzne umiestnených a natočených kapsúl alebo množinou bodov v

9 lepšie povedané: veľkosť telesa je menšia ako ním prekonaná dráha v jednom časovom kroku

priestore pokrytých krivou plochou (triangulovanou mriežkou). Veľkosť listu bola ohraničená – list vrtule nesmel presahovať pomyslený kváder veľkosti 10x8x8 (obr. 4.3). Listy boli upevnené v ložisku vrtule (os otáčania) a navzájom pootočené o uhol 120˚.



Obr. 4.4: model vrtule



Obr. 4.3: vymedzujúci kváder 10x8x8

Ložiskom vrtule bola kapsula s hlavnou osou identickou s osou otáčania vrtule. Ložisko vrtule bolo umiestnené do „vzduchu“, ukotvené o tzv. statické prostredie pomocou kĺbu (pántu), ktorý umožňoval otáčanie okolo horizontálnej osi. Bez ujmy na realistickosti sme zanedbali trenie v ložisku[10].

Pri vrtuli nás zaujímala maximálna dosiahnuteľná uhlová rýchlosť ω_{MAX} (pri danej rýchlosti vetra), hmotnosť listu vrtule *m* a rozloženie hmoty v liste vrtule (dôležitá je vzdialenosť ťažiska od osi otáčania *r*). Na základe hodnôt týchto veličín bola vypočítaná maximálna dosiahnuteľná kinetická (rotačná) energia vrtule podľa vzťahu[11]:

$$E_{\mathrm{MAX}} = \frac{1}{2} I \omega_{MAX}^2 = \frac{1}{2} 3 m r^2 \omega_{\mathrm{MAX}}^2$$

4.5 Model vetra

Prúdenie vzduchu – vietor – bolo vymedzené fiktívnym veterným koridorom, ktorý má tvar valca s podstavami pred a za vrtuľou kolmými na os otáčania vrtule. Nositeľmi energie vetra boli častice tvaru gule s polomer 0,1 jednotky. Počas simulácie vznikali pred vrtuľou v rovine podstavy veterného koridoru a

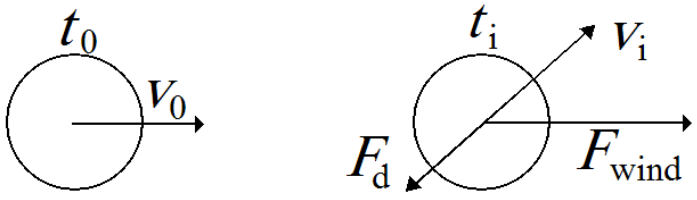
10 neznamená to, že vrtuľa by sa mohla rozkrútiť na nekonečnú rýchlosť pri neustálom vetre. Pre daný tvar (listu) vrtule a rýchlosť vetra existuje maximálna uhlová rýchlosť bez ohľadu na odpor v ložisku – pozri nižšie podkapitolu *Simulácia*

11 $I = 3mr^2$ je moment zotrvačnosti 3-listovej vrtule

bola im udelená počiatočná rýchlosť v_0 = 10. V každom kroku simulácie pôsobili na časticu 2 sily (obr. 4.5). Sila vetra F_{wind} = 0,3 pôsobí v smere prúdenia vzduchu rovnobežne s osou otáčania vrtule. Odporová sila F_{d} pôsobí proti pohybu letiacej častice a jej veľkosť je priamoúmerná rýchlosti častice. Výsledné silové pôsobenie na časticu popisuje vzťah

$$\vec{F} = \vec{F}_{\text{wind}} + \vec{F}_d = \vec{F}_{\text{wind}} - \alpha \,.\, \vec{v}$$

kde α = 0,02 je koeficient odporu v je vektor rýchlosti častice. Zo vzťahu vyplýva, že silové pôsobenie na časticu je nulové ak $\vec{F}_{\text{wind}} = \alpha \,.\, \vec{v}$ a teda na základe uvedených hodnôt ak vektor rýchlosti je súhlasne orientovaný ako vektor sily vetra a veľkosť sily je rovná 10. Z toho môžeme ďalej vyvodiť: častica po štarte (v_0 = 10) letí rovnomerným priamočiarym pohybom až kým nepreletí celým veterným koridorom a na podstave veterného koridoru za vrtuľou nezanikne, alebo kým nenarazí na prekážku – list vrtule. Vtedy odovzdá časť svojej kinetickej energie vrtuli a neodovzdanú energiu využije na pohyb od listu vrtule podľa zákona odrazu. Vektor rýchlosti v sa zmení a tým aj silové pôsobenie na časticu. Nenulová pôsobiaca sila bude smerovať časticu do smeru vetra a udelí jej v konečnom čase pôvodnú maximálnu rýchlosť v_0 (ak medzitým nedôjde k ďalšej zrážke, alebo neopustí veterný koridor).



Obr. 4.5: inicializácia a aktualizácia častice počas simulácie

Parametrami modelu sú taktiež veľkosť veterného koridoru a počet častíc vzduchu. Marc Ebner použil konštantne veľký veterný koridor nezávislý od veľkosti vrtule a počet častíc stanovil na hodnotu 100. Častice po opustení veterného koridoru boli reštartované v novej (náhodnej) počiatočnej pozícii v rovine podstavy veterného koridoru pred vrtuľou.

Keďže sme sa v našich experimentoch snažili vytvoriť exaktnejší model vetra, použili sme menší časový krok 0,01 (Ebner mal 0,1) a vietor bol tvorený väčším počtom častíc. Model sme zároveň optimalizovali premenne veľkým veterným koridorom, ktorý mal konštantný polomer podstavy 10,5 a dĺžku závislú od hĺbky vrtule. Predná podstava bola vo vzdialenosti 0,1 pred

najprednejším bodom vrtule a zadná podstava na úrovni najzadnejšieho bodu vrtule. Počet častíc v kubickej jednotke veterného koridoru bol konštantný[12] (0,7) a prakticky to znamenalo, že počet častíc bol v rozsahu 500 – 2000. Pre rôzne hlboké vrtule bol veterný koridor a teda počet častíc rôzny. Mohlo by sa zdať, že hlboké vrtule boli zvýhodnené oproti plytkým vrtuliam, lebo na ne mohlo pôsobiť viac častíc. Ak by sme pre plytké častice predĺžili veterný koridor na veľkosť pre hlboké vrtule, zvýšili by sme počet častíc v simulácii. No častice v priestore, ktorý sme pridali, na plytkú vrtuľu v danom okamihu simulácie aj tak nepôsobia a len spomaľujú simuláciu tým, že ich musí dynamický podsystém ODE spracovávať.

Ďalšou úlohou bolo vytvorenie čo najrovnomernejšieho vetra počas celej simulácie. Vyskúšali sme 2 možné prístupy:

- v každom kroku simulácie boli aktívne (“zapnuté”) všetky častice a v prípade opustenia veterného koridoru boli reštartované v rovine podstavy veterného koridoru pred vrtuľou (vstupovali do veterného koridoru). Tento prístup dosahuje takmer konštantnú energiu vetra (súčet kinetických energií prítomných častíc) počas celej simulácie. Nevýhodou je veľmi premenlivý počet reštartovaných častíc v každom kroku. To spôsobuje “nárazy vetra” a negatívne ovplyvňuje okrem iného aj zastavenie simulácie (pozri nižšie – kritérium zastavenia simulácie).
- v každom kroku bol reštartovaný konštantný počet častíc, presnejšie povedané každá častica, ktorá opustila veterný koridor zanikla a pred vrtuľou vznikol konštantný počet nových častíc. Konštanta definujúca počet nových častíc v každom kroku bola získaná experimentálne tak, aby počet častíc v kubickej jednotke veterného koridoru bol približne 0,7. Aj napriek snahe o dosiahnutie konštantnej hustoty častíc v priestore, práve hustota bola najväčším nedostatkom tohoto prístupu – celkový počet častíc v simulácií značne kolísal a s tým aj celková energia vetra. Naviac počet častíc bol veľmi závislý od tvaru vrtule (nielen veľkosti) a energia vetra nebola porovnateľná pre rôzne vrtule.

Nakoniec sme zvolili strednú cestu medzi oboma prístupmi. Celkový počet častíc vo veternom koridore bol určený ako v prvom prípade (pomocou hustoty častíc v priestore). Definovali sme limit na maximálny počet vstupujúcich častíc v jednom kroku *max_restarts*. Vytvorili sme zároveň zásobník častíc, do ktorého sme ukladali tie častice, ktoré opustili veterný koridor. V každom kroku simulácie sme zo všetkých častíc, ktoré opustili veterný koridor v

[12] konštanta získaná na základe pozorovaní

danom kroku, najviac *max_restarts* reštartovali a zvyšné sme pridali na zásobník. Ak pritom nebol naplnený limit *max_restarts*, tak sa vybrali častice zo zásobníka a tie doplnili množinu reštartovaných častíc. Ak nebol dostatok častíc ani v zásobníku, tak v danom kroku vstupovalo do veterného koridoru menej častíc. Limit *max_restarts* bol dynamicky upravovaný podľa stavu zásobníka. Pri zväčšovaní zásobníka (nad hranicu bezpečnej rezervy[13]) rástol priamo úmerne od veľkosti zásobníka. Pri zmenšovaní zásobníka (alebo pri prázdnom zásobníku) klesal konštantne.

Poznámka: Kvôli niektorým konvexným tvarom vrtúľ dochádzalo počas simulácie k "uviaznutiu" častíc v konvexnej oblasti, čo výrazne spomalilo simuláciu (spôsobovalo to veľa kolízií v každom kroku) a narušilo rovnomernosť vetra. Preto častice, ktoré počas jedného letu veterným koridorom narazili na vrtuľu viac ako 16-krát[14], boli klasifikované ako častice mimo veterného koridoru a buď boli reštartované alebo pridané na zásobník.

Uvedená implementácia rovnomerného vetra sa ukázala ako dostatočne spoľahlivá pre rôzne tvary vrtúľ – energia vetra kolísala v závislosti od tvaru vrtule a času simulácie minimálne a "nárazy vetra" boli tlmené.

4.6 Simulácia

Po vytvorení scény v prostredí ODE simulátora – konštrukcia vrtule a jej umiestnenie – nasleduje samotná simulácia. Po štarte v každom kroku vzniká 20 nových častíc vzduchu až kým sa nedosiahne žiadaný počet častíc vo veternom koridore[15]. Umiestnené sú rovnomerne náhodne v podstave veterného koridoru pred vrtuľou. Počas simulácie dochádza ku kolíziám častíc vzduchu a listov vrtule, častice odovzdávajú časť svojej kinetickej energie vrtuli, čo sa prejaví jej rozkrútením.

Simulácia sa vykonáva, kým nedôjde ku konvergencii uhlovej rýchlosti vrtule, minimálne však 500 krokov a maximálne 2000 krokov. Kritérium konvergencie uhlovej rýchlosti je definované tak, že konvergencia nastáva ak sa priemerná uhlová rýchlosť 4 rôznych úsekov histórie uhlovej rýchlosti líši o menej ako δ. Pri danej rýchlosti vetra existuje pre každú vrtuľu[16] konečná maximálna uhlová rýchlosť v_{MAX}, ktorú vie dosiahnuť v konečnom čase pri pôsobení daného vetra. Cieľom simulácie je vlastne dosiahnutie tejto rýchlosti,

13 rezerva predstavuje žiadúci objem zásobníka (veľkosť rezervy je 20 častíc)
14 konštanta získaná na základe pozorovaní
15 pozri podkapitolu Model vetra
16 pripomíname, že neuvažujeme trenie v ložisku vrtule a teda odpor proti otáčaniu vrtule

ktorá spolu s hmotnosťou a rozložením hmoty listov vrtule určí, koľko energie je schopná získať z daného vetra. Treba poznamenať, že v dôsledku stochastického charakteru simulácie sa pri viacerých simuláciách tej istej vrtule dosahujú výsledné uhlové rýchlosti z okolia teoretickej v_{MAX}. Vhodnými parametrami kritéria zastavenia sme sa snažili zmenšiť toto okolie čo najviac. Na druhej strane kritérium nesmelo byť príliš prísne (malé δ), aby čas simulácie nebol neúnosne dlhý.

Neformálny dôkaz existencie konečnej v_{MAX}:

Uvažujme ľubovoľný tvar listu vrtule (veľkosť listu je vymedzená kvádrom (obr. 4.3)). Nech má vrtuľa nulovú počiatočnú uhlovú rýchlosť. Nech pôsobenie letiacich častíc vzduchu začne roztáčať vrtuľu v jednom smere. Označme *A* množinu častíc, ktoré *v danom okamihu* zrážkou s vrtuľou prispievajú k zvýšeniu jej uhlovej rýchlosti v tomto smere. Častice množiny *A* odovzdávajú časť svojej kinetickej energie vrtuli a to spôsobuje kladné uhlové zrýchlenie v uvažovanom smere. Otáčaním vrtule dochádza k pohybu listov vrtule okolo osi otáčania. Pri pohybe listy "zrážajú" iné letiace častice vzduchu. Označme *B* množinu častíc "zrazených" listami vrtule *v danom okamihu*. Častice množiny *B* odoberajú kinetickú energiu rotujúcej vrtule a tým ju brzdia (znižujú okamžité uhlové zrýchlenie). Čím rýchlejšie bude vrtuľa rotovať, tým bude množina *B* väčšia a množina *A* menšia. Rovnovážny stav nastane, keď okamžitý energetický *zisk* vrtule bude rovný okamžitej energetickej *strate* vrtule. V rovnovážnom stave je okamžité uhlové zrýchlenie rovné nule a uhlová rýchlosť vrtule sa bez umelého zväčšenia množiny *A* (silnejší vietor) nezvýši. Teda uhlová rýchlosť dosiahne maximum – v_{MAX}.

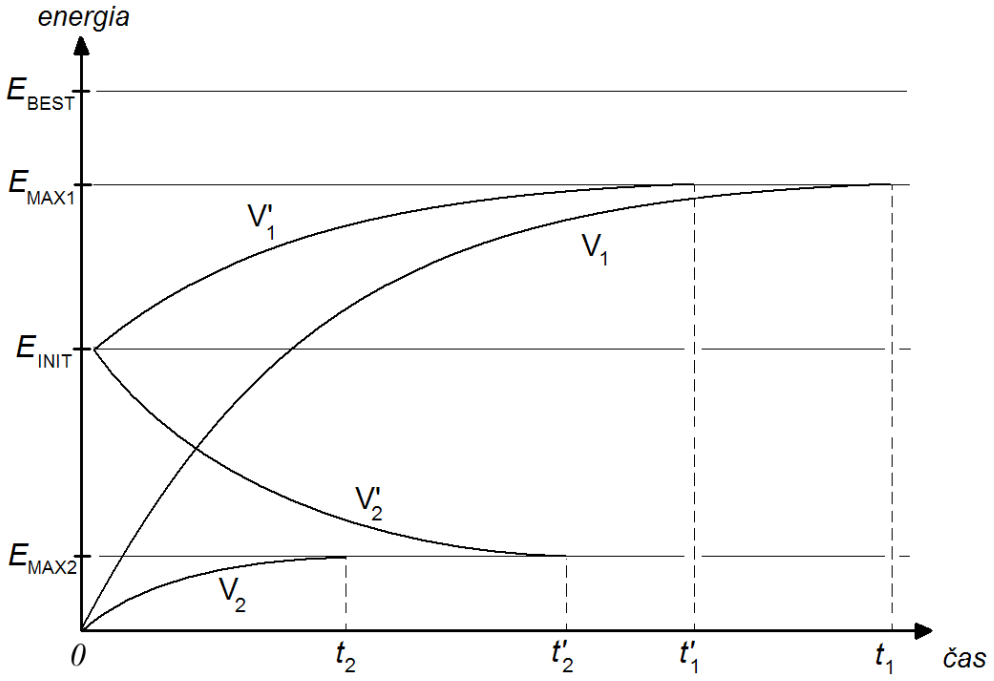
4.6.1 Optimalizácia simulácie

V priebehu evolúcie sa populácia postupne vyvíja a začínajú v nej prevládať jedince (vrtule) s vyššou fitness (kinetická energia). V simulácii pri počiatočnej nulovej kinetickej energii bude pre väčšinu vrtúľ potrebné vykonať čoraz viac krokov na dosiahnutie tejto (v evolúcii) rastúcej energie. To bude spôsobovať predlžovanie celkového času simulácie ako evolúcia smeruje k vyšším energiám.

Tým, že vrtuli udelíme nenulovú počiatočnú kinetickú energiu E_{INIT} (rotácia v žiadanom smere), pomôžeme jej "preskočiť" prvých *k* krokov simulácie počas ktorých by sa rozbiehala z 0 na túto E_{INIT} . Hodnota E_{INIT} je rovná polovici okamžitej maximálnej dosiahnutej kinetickej energie E_{BEST} (nejakou vrtuľou – jedincom) v priebehu evolúcie.

Na obr. 4.6 je zobrazený priebeh okamžitej kinetickej energie od času počas jednej simulácie. Ak vrtuli V_1 udelíme počiatočnú energiu E_{INIT}, skrátime celkový čas simulácie z t_1 na t'_1.

V prípade, že počiatočná udelená kinetická energia E_{INIT} je pre nejakú vrtuľu V_2 (V'_2) väčšia ako jej maximálna dosiahnuteľná kinetická energia E_{MAX2} (vrtuľa má nevhodný tvar listov), dôjde počas simulácie k poklesu[17] energie z E_{INIT} na E_{MAX2}. Pri niektorých vrtuliach (E_{MAX} je blízka 0) teda môže nenulová počiatočná energia spôsobiť predĺženie celkového času simulácie, avšak ide o podstatne *menej* vrtúľ ako je tých, ktorých simulačný čas sa skráti. Zároveň predĺžený čas simulácie týchto vrtúľ nie je dlhší ako čas simulácie vrtúľ, ktorých E_{MAX} je blízka E_{BEST}. V konečnom dôsledku je po zavedení nenulovej E_{INIT} maximálny čas simulácie ľubovoľnej vrtule kratší.



Obr. 4.6: optimalizácia času simulácie.

V_1 a V'_1 sú rovnaké vrtule. V_1 má nulovú počiatočnú energiu a V'_1 nenulovú. Podobne pre V_2 a V'_2.

[17] množiny A a B z neformálneho dôkazu existencie v_{MAX} (predošlá strana) majú v tomto prípade iný pomer vplyvu na rotáciu vrtule. Vplyv B prevláda nad vplyvom A a teda dochádza k brzdeniu. Rovnovážny stav nastane, keď vplyv B klesne a vplyv A narastie a navzájom sa vyrovnajú

5 Distribuovaný systém

Mnohé (nielen) vedecké problémy v súčasnosti vyžadujú veľký výpočtový výkon, ktorý nedosahujú ani tie najvýkonnejšie samostatne pracujúce počítače. Na získanie požadovanej výpočtovej kapacity sa využívajú skupiny počítačov[18] prepojené na rôznych úrovniach, ktoré tvoria *distribuovaný systém*. Distribuovaný systém ako celok rieši jeden pridelený problém. Existujú rôzne veľké distribuované systémy s rôznymi architektúrami, bežiace na rôznych platformách. Z hľadiska dostupných výpočtových zdrojov ich môžeme zaradiť do nasledovných kategórií:

- špecializované paralelné vysoko výkonné výpočtové systémy obsahujúce tisícky procesorov využívajúce špecializovaný softvér riadiaci komunikáciu medzi procesormi. Príkladom môže byť v súčasnosti najvýkonnejší "superpočítač" BlueGene [21] firmy IBM, ktorý tvorí viac ako 130 000 procesorov. Bol vytvorený na štúdium skladania bielkovín z oblasti molekulárnej biológie
- počítačové klastre, ktoré obsahujú zväčša výkonné osobné počítače s upravenou verziou operačného systému Linux so softvérom podporujúcim distribuované aplikácie – komponenty a knižnice pre *message passing, threading,* rozvrhovanie (*scheduling*), sledovanie stavu počítačov v klastri (*monitoring*). Počítače sú prepojené počítačovou sieťou využívajúcou TCP/IP. Príkladom jednoduchého klastra je CPR [22] klaster na FMFI UK, ktorý tvorí 1 dvoj-procesorový Xeon 3.06GHz server a 6 jedno-procesorových P4 2.4GHz výpočtových uzlov. Operačným systémom je Fedora Linux 4, počítače zdieľajú jeden súborový systém.
- osobné počítače patriace istej organizácií s primárnym účelom iným ako je distribuované riešenie problémov. Na svoj primárny účel nebývajú avšak využité 24 hodín denne, preto v čase nevyužitia sa stávajú súčasťou systému riadeného špecializovaným softvérom ako napr. Condor [23] alebo Q^2ADPZ [24]. Tieto nástroje spravujú dočasne dostupné výpočtové zdroje, spracúvajú požiadavky klientov (napr. pracovníci na vedeckom projekte), rozdeľujú úlohy a odovzdávajú výsledky klientom

[18] v istých prípadoch pod počítačom rozumieme procesor

- osobné počítače patriace dobrovoľníkom a nadšencom, ktorí poskytnú výpočtový čas svojho procesora na riešenie najnáročnejších vedeckých problémov. Jednotlivci si stiahnu a nainštalujú klientské programy, ktoré sa pripájajú na server spravovaný napr. univerzitou, ktorá rieši problém. Používateľ sa zaregistruje pre potreby štatistického vyhodnocovania zúčastnených užívateľov (čo zároveň zvyšuje záujem ľudí o účasť v tejto forme distribuovaného systému). Používateľ si tiež vyberá zo súboru aktuálnych projektov. Najznámejším viac-účelovým softvérom tejto kategórie je BOINC (Berkeley Open Infrastructure for Network Computing) [25], ktorý podporuje viacero projektov (Rosetta@home, Proteins@Home, SETI@Home...). Iným príkladom je Folding@Home [26], ktorý bol vytvorený na Stanfordskej univerzite a predstavuje vlastný softvér aj projekt. Mnohé projekty sú z oblasti molekulárnej biológie, matematiky, klimatológie a iných. Charakteristikou úloh týchto projektov, ktoré klientské programy riešia, je nízky pomer spracovávaných príp. generovaných dát k časovým nárokom na ich spracovanie (počítače komunikujú prostredníctvom Internetu, preto objem dát pre jednu úlohu musí byť značne ohraničený) a ich nezávislosťou na dátach a výpočtoch iných klientov. Vyriešenie jednej úlohy vyžaduje aj niekoľko hodín. Z hľadiska efektivity by nebolo výhodné distribuovať jednoduché úlohy, pretože by čas komunikácie prevyšoval čas potrebný na výpočet.

Pre potreby tejto práce bol vytvorený špecifický distribuovaný systém, ktorý by sme mohli zaradiť do tretej a štvrtej kategórie (využíval počítače v učebniach a domácnostiach, ktoré neboli momentálne využívané). Špecializovaný softvér typu Condor bol nahradený jednou MySQL databázou a skriptami pre Linux aj Windows. Komunikácia prebieha na úrovni TCP/IP prostredníctvom Internetu.

5.1 Distribuovaný evolučný výpočet

Pri reálnych problémoch, ktoré sú riešené pomocou evolučných algoritmov, časovo najnáročnejšou časťou je ohodnotenie jedinca v populácií. Inicializácia, kríženie, mutácia a selekcia sú relatívne jednoduché operácie nad genotypmi jedincov, ktorých podiel na celkovom čase evolučného výpočtu je menší ako 1%. Naopak, ohodnotenie jedinca (určenie jeho fitness), ktoré vyžaduje ohodnotenie jeho vlastností v prostredí problému – teda posudzujú sa jeho fenotypické vlastnosti, zaberie viac ako 99% času behu evolučného algoritmu.

Evolučné algoritmy pracujú síce s populáciou ako s celkom a napríklad pri selekcii (či už rodičov alebo potomkov) potrebujú mať k dispozícii celú populáciu, ohodnotenie jedinca však býva nezávislé na ostatných jedincoch. Ohodnotenie je preto veľmi vhodné implementovať paralelne (obr. 5.1). Jednoduchá myšlienka – majme k dispozícii N počítačov a celú populáciu veľkosti M. Ak $M \leq N$, tak každého jedinca pridelíme jednému počítaču – čas potrebný na ohodnotenie jednej generácie sa bude rovnať najdlhšiemu času potrebnému na ohodnotenie jedného jedinca. Ak $M > N$, tak niektoré (prípadne všetky) počítače budú postupne ohodnocovať viacero jedincov – čas na ohodnotenie jednej generácie bude rovný času výpočtu počítača najdlhšie vyhodnocujúceho jemu pridelenú skupinu jedincov.

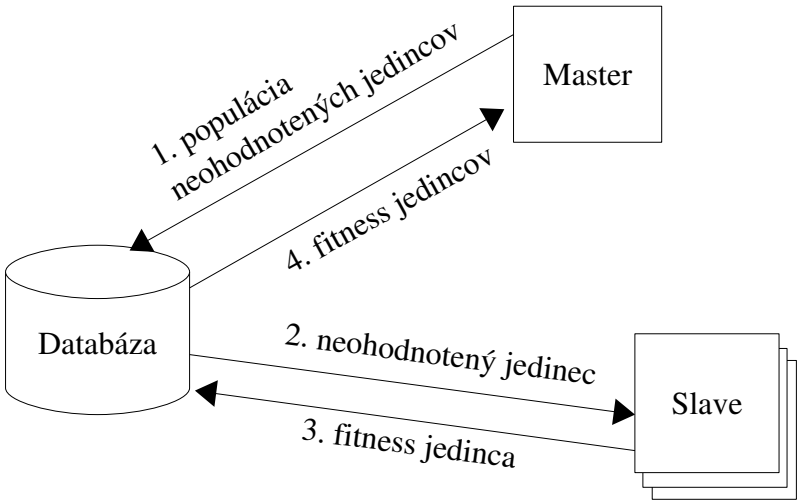
Experimenty opísané v tejto práci vyžadovali vykonanie viacero evolučných výpočtov, pričom každý predstavoval evolúciu 200 jedincov počas 200 generácií. Ohodnotenie jedinca – tvaru listu vrtule – bolo výsledkom simulácie, ktorá trvala 1-15 minút (záviselo to najmä od výkonu počítača, ale aj od zložitosti tvaru listu vrtule). K dispozícii bolo viacero počítačov, z ktorých bol vytvorený distribuovaný systém opísaný nižšie. Vďaka tomuto distribuovanému systému bol čas jednej evolúcie skrátený z približne 50 dní na 1 deň.

5.2 Architektúra systému

Evolučný algoritmus podľa schémy na obr. 2.1 bol vykonávaný na jednom počítači, ktorý nazývame *Master*[19]. Na ostatných počítačoch – nazývaných *Slave(s)* – bol spustený program, ktorý ohodnocoval jedného jedinca.

Počítače boli pripojené do siete Internet. Komunikácia medzi *Master*-om a *Slave*-ami prebiehala na aplikačnej úrovni prostredníctvom MySQL databázy. *Master* program v každej generácii odoslal množinu neohodnotených jedincov do tabuľky `evoX` v databáze. V poradí ako ich odoslal, ich potom načítaval – už aj s hodnotou fitness. Ak jedinec ešte nebol ohodnotený, tak *Master* počkal 5s a pokus o načítanie jedinca s fitness opakoval znova. Po získaní fitness všetkých jedincov z populácie, vykonal jeden evolučný cyklus (vytvorenie novej generácie, selekcia rodičov, kríženie a mutácia) a ak nebolo splnené kritérium zastavenia evolúcie, odoslal opäť novovytvorených (a teda neohodnotených) jedincov do databázy.

[19] rovnakým názvom označujeme aj počítač aj program



Obr. 5.1: Distribuovaný evolučný algoritmus

Slave programy sa v pravidelných intervaloch[20] (5s) pripájali do databázy a v prípade, že sa v tabuľke `evoX` nachádzal neohodnotený jedinec, stiahli si jeho genotyp (genóm), preložili ho na fenotyp a spustili simuláciu s týmto konkrétnym fenotypom (vrtuľou). Po ukončení simulácie odoslali výslednú hodnotu fitness do príslušného riadku tabuľky `evoX`. Pri prvom pripojení sa do databázy prebehne registrácia *slave*-a – vytvorí sa nový záznam v tabuľke `slavesX`.

Základná schéma distribuovaného systému je na obr. 5.1.

Kontrola stavu distribuovaného systému bola vykonávaná sledovaním web-stránky obsahujúcej štatistiky, ktorá zbierala informácie z tabuľky `slavesX`.

5.3 Konkurencia *slave* programov

Slave programy vykonávajú rovnaký algoritmus. Preto by sa pri výbere jedinca z databázy mohlo stať, že si všetci vezmú rovnakého jedinca, ak by vyberali iba podľa kritéria – prvý neohodnotený jedinec v poradí.

Prvé riešenie problému spočívalo v podpísaní sa pod neohodnoteného jedinca v tabuľke (pomocou jedného UPDATE príkazu). Podpísať sa dalo iba pod jedinca, ktorý ešte nemal podpis iného *slave*-a. Po podpísaní si *slave* pomocou príkazu SELECT stiahol svojho jedinca. Tento prístup zabezpečil, že každý *slave* riešil iného jedinca. Počas prvých experimentov, keď bolo k

[20] nie počas behu simulácie

dispozícii relatívne málo počítačov (10-15), toto riešenie postačovalo (pri populácii 200 jedincov). S nárastom počtu počítačov (100+) vznikol problém v situáciach, keď v tabuľke nebol k dispozícii voľný (nepodpísaný) neohodnotený jedinec. Vtedy všetky *slave* programy, ktoré ukončili simuláciu, museli čakať, kým ostatné *slave* programy dokončia simulácie svojich jedincov. Až keď všetky ukončili simulácie, *Master* program mohol získať hodnotu všetkých jedincov v populácii a do tabuľky odoslať novú generáciu jedincov. Keďže výkon počítačov – *slave*-ov – v našom distribuovanom systéme bol rôzny, zväčša tie rýchlejšie (výkonnejšie) čakali na pomalšie. Využitie paralelizmu (*utilization*) bolo znížené.

V krajnom prípade (200 *slave*-ov simuluje 200 jedincov), všetci okrem najpomalšieho čakali po ohodnotení jedného jedinca na tohto najpomalšieho *slave*-a. Najpomalší *slave* počas celej evolúcie ohodnotil rovnaký počet jedincov ako ten najrýchlejší.

Situácia pri našich experimentoch bola komplikovaná aj skutočnosťou, že popri rôznej výkonnosti dostupných *slave*-ov bola aj časová náročnosť simulácie rôznych jedincov rôzna (pri danom výkone počítača) – zložitejšie vs. jednoduchšie jedince (tvary listu vrtule). Nastávali prípady, keď sa pomalší *slave*-ovia podpísali pod zložitejšieho jedinca[21] a naopak rýchlejšie počítače simulovali jednoduchších jedincov (vrtule).

Na základe pozorovaní bol pre *slave* program navrhnutý nový prístup akvizície jedinca na ohodnotenie. Nevyžaduje sa, aby daného jedinca riešil iba jeden *slave*. Práve naopak – každý *slave,* ktorý by mal prejsť do stavu nečinnosti pri čakaní na ešte simulujúcich *slave*-ov, namiesto toho začne simulovať jedinca, ktorý už je v procese simulácie, ale ešte stále nie je ohodnotený. Je možné, že daný jedinec pôvodne riešený pomalým *slave*-om je príliš zložitý a iba rýchlejšie počítače sú schopné ho ohodnotiť v rozumnom čase. Niektoré počítače sú 5-krát rýchlejšie ako iné a v populácii sa nachádzajú jedince, ktorých simulácia týmito rýchlymi *slave*-ami trvá aj 5-6 minút. V pôvodnej verzii *slave* programu, sa mohol pod zložitého jedinca podpísať pomalý *slave,* simulácia trvala aj 30 minút a ostatní *slave*-ovia nečinne čakali. Keď je mnoho jedincov na ohodnotenie (a rýchli *slave-ovia* sú zamestnaní*)*, tak aj tento pomalý *slave* je prínosom, avšak pri zmenšení množiny neohodnotených jedincov (“koniec generácie”), je viac na škodu ako na úžitok, lebo blokuje rýchlych *slave*-ov. V novej verzii *slave* programu prínos pomalého *slave*-a na začiatku každej generácie je zachovaný, ku koncu generácie už avšak nebrzdí rýchlejších *slave*-ov.

21 zložitosť jedinca sa nedá odhadnúť vopred

Nová verzia *slave* programu využíva pole `slaves_working` v zázname jedinca v tabuľke `evoX`, aby dochádzalo k rovnomernému rozdeľovaniu jedincov. Pole obsahuje počet *slave*-ov aktuálne riešiacich daného jedinca. *Slave*, ktorý ukončil simuláciu si vyberá pre ďalšiu simuláciu jedinca, ktorého rieši najmenší počet *slave*-ov. Ak má na výber spomedzi viacerých jedincov (je viacero jedincov, ktorých rieši minimálny počet *slave*-ov), vyberá si toho, ktorý je už najdlhšie riešený (využitie poľa `start`).

Pre prístup do databázy pri akvizícii nového jedinca sa využíva transakčné spracovanie.

5.4 Databáza

Databáza obsahuje 2 tabuľky – `evoX`[22] a `slavesX` popísané nižšie:

```
mysql> describe evoX;
+----------------+------------------+------+-----+---------+----------------+
| Field          | Type             | Null | Key | Default | Extra          |
+----------------+------------------+------+-----+---------+----------------+
| ID             | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
| genome         | blob             | NO   |     |         |                |
| value          | double           | YES  |     | NULL    |                |
| slaves_working | int(10) unsigned | NO   |     | 0       |                |
| state          | int(10) unsigned | NO   |     | 0       |                |
| start          | datetime         | NO   |     |         |                |
+----------------+------------------+------+-----+---------+----------------+
```

Tab. 5.1: MySQL tabuľka evoX

genome – genotyp jedinca v ASCII znakoch

value – fitness jedinca

slaves working – koľko *slave* programov začalo ohodnocovať príslušného jedinca

state – slúži na riadenie činnosti *slave* programov užívateľom (upgrade programov)

start – čas začiatku ohodnocovania jedinca prvým *slave* programom

[22] `X` vyjadruje identifikátor konkrétnej evolúcie. Pre VRML napr. `evoVRML`, podobne pre tabuľku `slavesX`: `slavesVRML`

```
mysql> describe slavesX;
+---------------+------------------+------+-----+---------+----------------+
| Field         | Type             | Null | Key | Default | Extra          |
+---------------+------------------+------+-----+---------+----------------+
| ID            | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
| slave_id      | varchar(255)     | NO   |     |         |                |
| first_visit   | datetime         | NO   |     |         |                |
| last_visit    | datetime         | NO   |     |         |                |
| best_solution | double           | NO   |     |         |                |
| max_time      | int(10) unsigned | NO   |     |         |                |
| reported      | int(10) unsigned | NO   |     |         |                |
| completed     | int(10) unsigned | NO   |     |         |                |
| unfinished    | int(10) unsigned | NO   |     |         |                |
| useful_time   | int(10) unsigned | NO   |     |         |                |
| useless_time  | int(10) unsigned | NO   |     |         |                |
| idle_time     | int(10) unsigned | NO   |     |         |                |
+---------------+------------------+------+-----+---------+----------------+
```

Tab. 5.2: MySQL tabuľka slavesX

slave_id – čitateľný identifikátor klienta obsahujúci verziu programu, platformu, názov počítača, fyzickú adresu sieťovej karty a pracovný adresár

first_visit – dátum a čas registrácie

last_visit – dátum a čas posledného zápisu do databázy

best_solution – fitness hodnota najlepšieho jedinca, ktorého *slave* doposiaľ ohodnotil

max_time – najdlhší čas, ktorý *slave* potreboval na ohodnotenie nejakého jedinca

reported – počet ohodnotených jedincov, ktorých fitness *slave* zapísal do databázy

completed – počet ohodnotených jedincov (úspešne ukončené simulácie)

failed – počet nedokončených ohodnotení (simulácií)

useful_time – čas, ktorý *slave* strávil úspešne ukončenými simuláciami

useless_time – čas strávený pri neúspešných simuláciách

idle_time – čas, počas ktorého *slave* neohodnocoval žiadneho jedinca

5.5 *Master* program – komunikácia s databázou

V každej generácii *Master* program vykoná nasledovné.

Odoslanie neohodnotených jedincov do tabuľky:

```
START TRANSACTION;
SELECT * FROM evoX FOR UPDATE;[23]
INSERT INTO evoX (genome, value, state, slaves_working, start)
VALUES ('<jedinec1>',-1000.0, 0, 0, '1900-01-01 00:00:00');
...
INSERT INTO evoX (genome, value, state, slaves_working, start)
VALUES ('<jedinecN>',-1000.0, 0, 0, '1900-01-01 00:00:00');
COMMIT;
```

Načítanie fitness hodnoty z databázy:

```
SELECT value FROM evoX
WHERE (genome='<jedinec1>') AND (value > -1000.0);
```

ak je výsledkom uvedeného dotazu prázdna množina, opakuj ho o 5s

Master si uloží fitness jedinca a zmaže jeho záznam z tabuľky:

```
DELETE FROM evoX
WHERE (genome='<jedinec1>') AND (value > -1000.0) LIMIT 1[24];
```

následne prejde k ďalšiemu jedincovi `<jedinec2>` a pokračuje od kroku 2.

Takto sa to opakuje až po jedinca `<jedinecN>`

5.6 *Slave* program – komunikácia s databázou

Registrácia (ak sa ešte nenachádza príslušný záznam v tabuľke) :

```
INSERT INTO slavesX(slave_id, first_visit, last_visit,
idle_time, useful_time, useless_time, completed, unfinished,
reported, best_solution, max_time)
VALUES('<version-hostname-...>',NOW(),NOW(),0,0,0,0,0,0,0.0,0)
```

Načítanie (genotypu) jedinca z databázy:

```
START TRANSACTION;
SELECT genome,state,slaves_working,id FROM evoX
WHERE (value=-1000)
ORDER BY slaves_working, start, id
LIMIT 1 FOR UPDATE
```

[23] tým *Master* získa write-lock na celú tabuľku

[24] môže sa nachádzať aj viacero rovnakých jedincov, ak je daná evolúcia spustená paralelne viackrát

ak je výsledkom uvedeného dotazu prázdna množina, ukončí transakciu príkazom `ROLLBACK`, upraví pole `idle_time`:

```
ROLLBACK;
UPDATE slavesX SET idle_time=idle_time+5,last_visit=NOW()
WHERE (slave_ID='<verzia-hostname-...>')
```

a program zastaví (skript, ktorý ho spustil v tomto prípade spúšťa znova *slave* program[25] po uplynutí 5s)
inak uloží genotyp a id jedinca a zvýši počet *slave*-ov riešiacich príslušného jedinca o 1:

```
UPDATE evoX SET slaves_working=slaves_working+1 {,
start=NOW()}
WHERE (id=<id_jedinca>);
COMMIT;
```

Každých 30s počas simulácie kontroluje, či už jeho jedinec nebol ohodnotený iným bežiacim *slave*-om:

```
SELECT ID FROM evoX WHERE (ID=<id_jedinca>) AND (value=-1000)
```

ak je výsledkom prázdna množina – čo znamená, že niektorý *slave* riešiaci toho istého jedinca už zapísal jeho fitness (`value`) do tabuľky – zastaví simuláciu, upraví polia `unfinished` a `useless_time`:

```
UPDATE slavesX
SET unfinished=unfinished+1, last_visit=NOW(),
useless_time=useless_time+<doba vykonávania simulácie>
WHERE (slave_ID='<verzia-hostname-...>')
```

a program končí (a nasleduje ďalší cyklus skriptu)
inak simulácia pokračuje ďalej

Po zastavení simulácie (splnením kritérií zastavenia simulácie[26]) program zapíše hodnotu fitness do príslušného riadku tabuľky:

```
UPDATE evoX
SET value=<fitness>
WHERE (value=-1000) AND (id=<genome_id>)
```

ak uvedený UPDATE zmení jeden riadok tabuľky (program je prvým, ktorý ukončil simuláciu daného jedinca), upravia sa polia `reported`, `completed`, `max_time`, `best_solution` a `useful_time`:

```
UPDATE slavesX
SET reported=reported+1, completed=completed+1,
useful_time=useful_time+<doba simulácie>, best_solution=<...>,
max_time=<...>, last_visit=NOW()
```

25 ktorý má ten istý identifikátor

26 maximálny počet krokov, konvergencia uhlovej rýchlosti vrtule (pozri kapitolu *Fyzikálna simulácia*)

```
WHERE (slave_ID='<verzia-hostname-...>')
```

inak (pri poslednej kontrole stavu jedinca – bod 3. – jedinec ešte nebol ohodnotený, ale kým program ukončil simuláciu, už niektorý iný *slave* ohodnotil tohoto jedinca a zapísal jeho fitness do tabuľky) sa upravia len polia `completed, max_time, best_solution` a `useful_time`:

```
UPDATE slavesX
SET completed=completed+1, useful_time=useful_time+<doba
simulácie>, best_solution=<...>, max_time=<...>,
last_visit=NOW()
WHERE (slave_ID='<verzia-hostname-...>')
```

a program končí (nasleduje ďalší cyklus skriptu).

5.7 Nasadenie *slave*-ov

Pre jednoduchšie spúšťanie *slave* programov, sme upravili distribúciu Linuxu Knoppix Live CD [27] pre naše potreby tak, aby sa po nabootovaní z CD spustil skript, ktorý stiahne z pevne danej url adresy bash skript a začne ho vykonávať. Tento skript zistí počet procesorov v systéme (zväčša jeden alebo dva) a toľkokrát spustí v samostatnom adresári ďalší stiahnutý bash skript. Tento tretí skript v nekonečnom cykle spúšťa (a čaká na jeho ukončenie) skompilovaný program, ktorý si stiahne pri prvom cykle, alebo ak prostredníctvom databázy (na ktorú sa tento program pripája) obdrží signál UPGRADE. Nekonečný cyklus môže byť taktiež prerušený špeciálnym signálom QUIT, tým *slave* ukončí svoju činnosť.

Signál UPGRADE sa využíval počas vývojového štádia na opravovanie chýb v *slave* programe. Neskôr slúžil hlavne na "presunutie" pracujúcich *slave-ov* na novo spustenú evolúciu (keď sa mení simulačné prostredie a teda aj kód *slave* programu – napr. keď sa po ukončení evolúcie s priamou reprezentáciou spúšťa evolúcia s nepriamou reprezentáciou[27]).

Okrem *slave* programov spúšťaných pri bootovaní pomocou Live CD boli k dispozícií (a aj využívané) tiež Windows a Linux verzie *slave* programov spúšťané užívateľom. Tie vyžadovali manuálne stiahnutie si bash skriptu (Linux) alebo batch súboru (Windows).

27 pozri kapitolu *Experimenty*

6 Experimenty

Pri skúmaní reprezentácií v problémoch evolučného dizajnu sme sa pokúsili reprodukovať Ebnerov experiment opísaný v kapitole Evolučný dizajn. O detailoch tejto reprodukcie budeme pojednávať v podkapitole *Nepriama reprezentácia I*. V ďalšej časti s názvom *Priama reprezentácia* uvedieme alternatívny spôsob dizajnu tvaru listu vrtule a opíšeme podrobnosti experimentu, ktorý overil vlastnosti nami navrhnutej reprezentácie. V časti *Nepriama reprezentácia II* uvedieme nepriamu reprezentáciu tvaru listu inšpirovanú Ebnerovou reprezentáciou, avšak vychádzajúcu z našej priamej reprezentácie.

Všetky experimenty sú realizované evolučným algoritmom, ktorý na ohodnotenie jedinca populácie (tvaru vrtule) používa simuláciu modelu prúdenia vzduchu cez vrtuľu. Podrobnosti o modeli, jeho nastaveniach a parametroch a o samotnej simulácii sú uvedené v kapitole Fyzikálna simulácia. Vzhľadom na naše ambície porovnať reprezentácie z viacerých hľadísk, model vetra, všeobecný model vrtule a priebeh simulácie bol rovnaký v každej úlohe.

6.1 Nepriama reprezentácia I

6.1.1 Štruktúra jedinca

Genotypom jedinca je strom s veľkosťou vetvenia 2-3. Maximálna počiatočná hĺbka stromu je 4. Maximálna veľkosť stromu je 100 vrcholov (vnútorných vrcholov a listov). Vrcholy stromu sú vyberané z množiny F.

F = {R_0, T_0, R_2, T_2, R_3, T_3}, kde

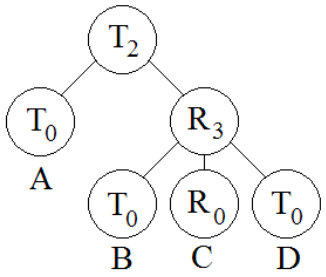
$$R_i(x, y, z, \alpha) \in (0;3)\,x\,(0;1)\,x\,(0;1)\,x\,(0;2\pi),\ \ i = 0,2,3$$

$$T_i(dx, dy, dz) \in (0;3)\,x\,(0;1)\,x\,(0;1),\ \ i = 0,2,3$$

R_i reprezentuje otočenie *(rotate)* okolo osi *(x, y, z)* o uhol α

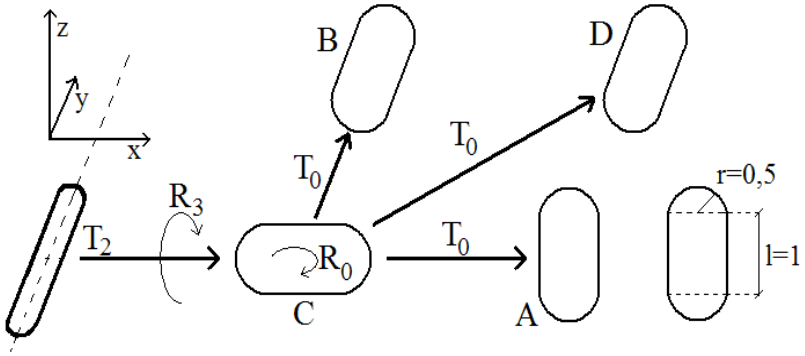
T_i reprezentuje posunutie (*translate*) podľa vektora *(dx, dy, dz)*
R_0, resp. T_0 sú listami stromu a reprezentujú finálne otočenie, resp. posunutie kapsule (stavebnej jednotky listu vrtule). Vrcholy R_2, T_2 majú dvoch synov, R_3, T_3 troch. Vnútorné parametre vrcholov sú rovnomerne náhodne zvolené pri inicializácii jedinca počas vytvárania počiatočnej populácie a pri vytváraní nových vrcholov počas mutácie.

Pri vytváraní modelu vrtule (fenotypu) podľa uvedeného genotypu je strom prehľadávaný do hĺbky. Pre každý list stromu sa do scény ODE simulátora umiestni kapsula a aplikujú sa na ňu transformácie uložené vo vrcholoch na ceste z listu do koreňa stromu (v tomto poradí). Tie kapsule, ktoré nie sú pripojené na kapsulu v osi otáčania (ložisko) priamo alebo prostredníctvom iných kapsúl, sú zrušené. Ak výsledný list vrtule má rozmery väčšie ako pomyslený kváder[28] veľkosti 10x8x8, tak príslušný jedinec má fitness 0. Ak je veľkosť listu vrtule v norme, spustí sa simulácia a kinetická energia vrtule po zastavení simulácie je fitness jedinca.



T2 1 0 0
(T0 1 0 0)
(R3 1 0 0 $\frac{\pi}{2}$
(T0 0 0 1)
(R0 0 0 1 $\frac{\pi}{2}$)
(T0 1 0 1)
)

Obr. 6.1: príklad genotypu jedinca



Obr. 6.2: príklad fenotypu prislúchajúceho jedincovi z obr. 6.1

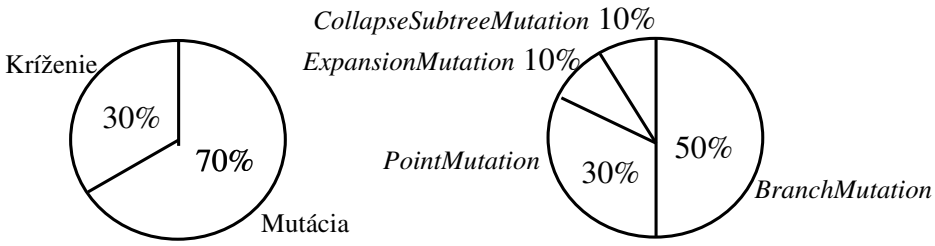
[28] pozri *Model vrtule* v kapitole *Fyzikálna simulácia*

Na obr. 6.1 je uvedený príklad genotypu jedinca a na obr. 6.2 zodpovedajúca konštrukcia listu vrtule. Strom má 4 listy – do scény v simulátore umiestnime 4 kapsule (A,B,C,D) do osi otáčania (ich počiatočná orientácie je ako pri kapsuli na obrázku vpravo dole). Všetky posunieme v smere osi *x* o úsek dĺžky 1 (T2 1 0 0). Kapsulu A posunieme ďalej v smere osi *x* o úsek dĺžky 1 (T0 1 0 0) a to je výsledná pozícia kapsule A. Ďalšie 3 kapsule otočíme o uhol $\frac{\pi}{2}$ okolo osi *x* (R3 1 0 0 $\frac{\pi}{2}$) – zároveň otáčame ich súradnicový systém. Kapsulu C otočíme o uhol $\frac{\pi}{2}$ okolo osi *z* (R0 0 0 1). Kapsulu B, resp. D posunieme v smere osi *z* o úsek dĺžky 1 (T0 0 0 1), resp. v smere osi *x* a osi *z* o úseky dĺžky 1 (T0 1 0 1) – pripomíname, že tieto kapsule sa posúvajú už v otočenom súradnicovom systéme.

6.1.2 Genetické operátory

Kríženie je implementované pomocou štandardnej operácie (*SubtreeXover*). V oboch rodičoch náhodne zvolí vrchol. Podstromy pod zvolenými vrcholmi sa navzájom vymenia. Ak je to potrebné, je novovytvorený strom orezaný na maximálnu veľkosť stromu.

Počas evolúcie jedinec môže byť predmetom 4 rôznych druhov mutácií. Pri *BranchMutation* náhodne zvolený podstrom je nahradený podstromom nového náhodne vytvoreného stromu. Pri *PointMutation* náhodne zvolený vrchol stromu je nahradený novým vrcholom s rovnakou aritou. Pri *CollapseSubtree-Mutation* náhodne zvolený podstrom je nahradený novým listom. Pri *ExpansionMutation* náhodne zvolený list stromu je nahradený podstromom nového náhodne vytvoreného stromu.



Obr. 6.3: pravdepodobnosti operácií kríženia a mutácie (NR)

Používame klasický[29] paralelný prístup pri vytváraní nového jedinca – pre každého (selekciou zvoleného) jedinca sa rozhodne, či sa zúčastní kríženia alebo mutácie (práve jedna operácia). Operácia kríženia nastávala s pravdepodobnosťou 0,3 a operácia mutácie s pravdepodobnosťou 0,7. V prípade, že je zvolená mutácia, aplikuje sa práve jedna z uvedených 4 mutácií. Pravdepodobnosti *BranchMutation, Point-Mutation, CollapseSubtreeMutation,* resp. *ExpansionMutation* sú 0,5; 0,3; 0,1 resp. 0,1 (obr. 6.3).

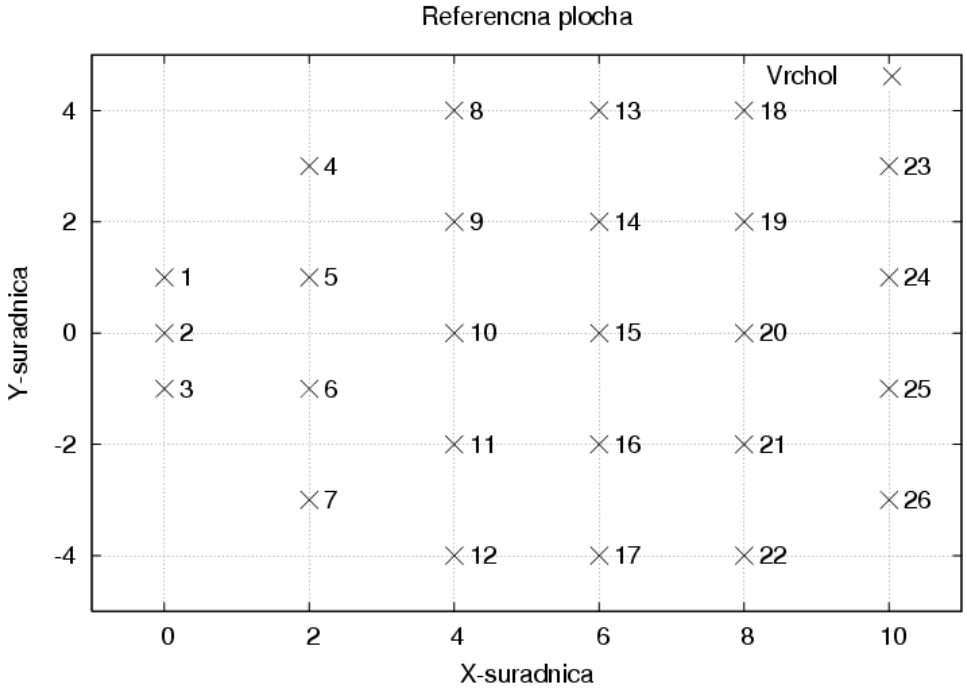
6.1.3 Evolúcia

Počiatočná populácia bola vytvorená metódou *Ramped-Half-and-Half (RHH)*[30] [11]. Obsahovala 200 jedincov. Pri selekcii rodičov sa používal turnaj veľkosti 2. Krížením a mutáciou vznikla nová populácia, ktorá nahradila rodičovskú populáciu – *generačná výmena*. Najlepší rodič nahradil v novej populácii najhoršieho jedinca v prípade, že tento rodič bol lepší ako všetky nové jedince – *mierny elitizmus*. Evolúcia skončila po dosiahnutí 200. generácie.

29 pre genetické programovanie (pozri kapitolu *Evolučné algoritmy*)
30 pozri časť *Genetické programovanie* (kapitola *Evolučné algoritmy*)

6.2 Priama reprezentácia

Skôr ako prejdeme k formálnej definícii genotypu, neformálne opíšeme našu priamu reprezentáciu tvaru listu vrtule. Pri definovaní tvaru vychádzame z tzv. *referenčnej plochy*, ktorá obsahuje 26 vrcholov. Vrcholy majú vopred určené súradnice (dva rozmery), ktoré sú rovnaké pre všetky vrtule a počas evolúcie sa nemenia. Počet vrcholov referenčnej plochy a ich rozloženie (súradnice) na ploche sme zvolili odhadom (obr. 6.4).



Obr. 6.4: rozloženie vrcholov referenčnej plochy v rovine XY (PR)

Genotyp jedinca určuje *vyvýšenie* vrcholov vzhľadom na referenčnú plochu. Toto vyvýšenie je ohraničené – maximálne +4 a minimálne -4. Genotyp tiež pre každý vrchol určí, či je *prítomný* vo výslednej krivej ploche, alebo je ignorovaný. Tvar listu je potom určený krivou plochou, ktorá obsahuje už *vyvýšené prítomné* vrcholy referenčnej plochy. Poslednou informáciou obsiahnutou v genotype je *uhol* naklonenia celého listu vrtule (ide vlastne o naklonenie referenčnej plochy).

Pri inicializácií a mutáciách jedinca sa kontrolujú rozmery listu, aby nepresahoval pomyslený kváder[31] veľkosti 10x8x8. V prípade presahovania je vyvýšenie príslušného vrcholu upravené na najbližšiu povolenú hodnotu.

6.2.1 Triangulácia

Na definovanie plochy potrebujeme okrem množiny vrcholov aj množinu trojuholníkov nad množinou vrcholov. Triangulációu množiny vrcholov (dvojrozmerný prípad) je množina trojuholníkov, ktoré pokrývajú dané vrcholy, navzájom sa nepretínajú a tvoria konvexný obal množiny vrcholov. Pre každú množinu vrcholov (s minimálne 4 vrcholmi) existuje viacero triangulácií, avšak nie každá má dobré vlastnosti.

V našom experimente sme používali Delaunayovu trianguláciu, ktorá je spomedzi všetkých triangulácií "najrovnomernejšia" – minimalizuje obvody trojuholníkov a polomer opísanej kružnice ľubovoľného trojuholníka. Podrobnejšie o Delaunayovej triangulácii možno nájsť v prílohe A. Delaunayovu trianguláciu sme hľadali nad množinou *prítomných* vrcholov referenčnej plochy *bez vyvýšenia*, keďže algoritmus na hľadanie triangulácie uvažuje dvojrozmerný priestor (teda pri hľadaní triangulácie sme z-súradnicu nezohľadňovali). Po získaní množiny trojuholníkov sme vrcholom pridali tretí rozmer (vyvýšenie), tým sa trojuholníky roztiahli do trojrozmerného priestoru. Výsledkom bola krivá (triangulovaná) plocha reprezentujúca tvar listu.

Na základe získanej množiny trojuholníkov a množiny vrcholov boli v ODE simulátore vytvorené listy vrtule (využijúc ODE objekt *TriMesh*), naklonené o uhol uvedený v genotype. Hrúbka listu vrtule (krivej plochy) bola 0,1.

6.2.2 Štruktúra jedinca

Genotypom jedinca je vektor (chromozóm)

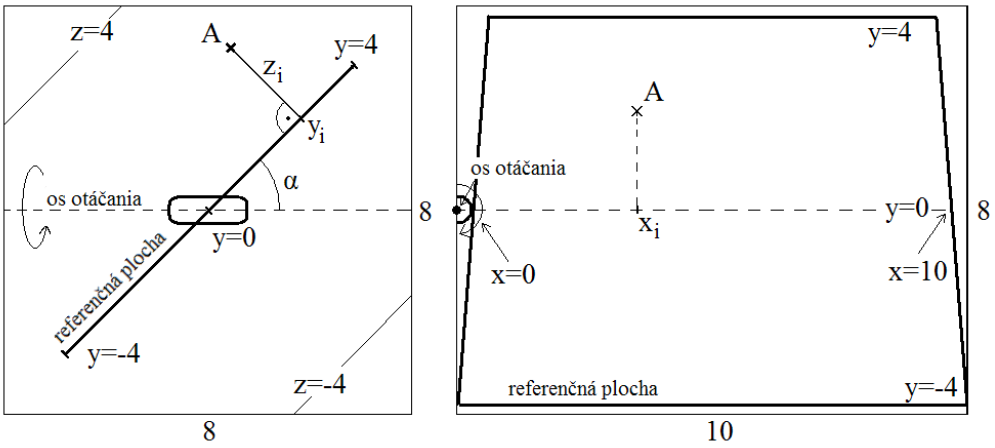
$$(\alpha, (z_1, b_1), (z_2, b_2), ..., (z_{26}, b_{26})), \alpha \in \mathbb{R}(0,1), z_i \in \mathbb{R}(-4,4), b_i \in \{0,1\}$$

- α reprezentuje uhol naklonenia referenčnej plochy (listu vrtule) (obr. 6.5 vľavo). Interval (0, 1) zodpovedá intervalu (0°, 90°) – pri uhle 0° je rovina horizontálna, pri uhle 90° je vertikálna
- z_i určuje vyvýšenie i-teho vrcholu vzhľadom na referenčnú plochu (obr. 6.5). Súradnice i-teho vrcholu v referenčnej ploche sú (x_i, y_i), ktoré sú pevne dané (obr. 6.4)

[31] pozri *Model vrtule* v kapitole *Fyzikálna simulácia*

- b_i určuje, či je *i*-ty vrchol *prítomný* v krivej ploche listu vrtule

Výsledná plocha listu vrtule je tvorená vhodnou[32] množinou trojuholníkov nad množinou *prítomných* vrcholov $\{(x_i, y_i, z_i) \mid b_i = 1, i = 1, 2, \ldots 26\}$.



Obr. 6.5: Referenčná plocha listu vrtule z profilu a z prednej strany. Príklad umiestenia bodu A(xi, yi, zi), ktorý je jedným z vrcholov prítomných v liste (v krivej ploche) – (xi, yi, 0) je vrcholom jedného alebo viacerých trojuholníkov Delaunayovej triangulácie

Po uvedení štruktúry genotypu a prekladu informácie jedinca do výsledného tvaru listu je dobré sa pozastaviť pri otázke, či ide naozaj o priamu reprezentáciu (z hľadiska vzťahu medzi genotypom fenotypom). Napriek nie práve jednoduchej definícii genotypu vieme každej dvojici (z_i, b_i) priradiť vrchol z plochy listu (a naopak – každému vrcholu z plochy listu vieme jednoznačne priradiť dvojicu (z_i, b_i) z vektora genotypu). Zároveň vieme podľa indexu vrchola v chromozóme povedať, kam (topologicky) na list vrtule daný vrchol patrí. Vrcholy blízke v chromozóme sú blízko aj na liste vrtule.

Pri nepriamej reprezentácii z predošlej časti vieme danú stavebnú jednotku (kapsulu) umiestniť až po prečítaní všetkých vrcholov po ceste z koreňa do listu. Avšak ani prečítanie tejto informácie ešte negarantuje prítomnosť danej kapsule v liste. To sa ukáže až po umiestnení iných kapsúl[33]. Pri nepriamej reprezentácií sú fenotypické vlastnosti (umiestnenie kapsúl) globálne organizované na základe stromovej štruktúry. Pri priamej reprezentácií sú fenotypické vlastnosti určené lokálne na základe časti vektora genotypu (chromozómu).

32 Delaunayova triangulácia je vhodnou množinou trojuholníkov

33 pripomíname, že kapsule musia byť pripojené k ložisku vrtule priamo alebo prostredníctvom iných pripojených kapsúl

6.2.3 Genetické operátory

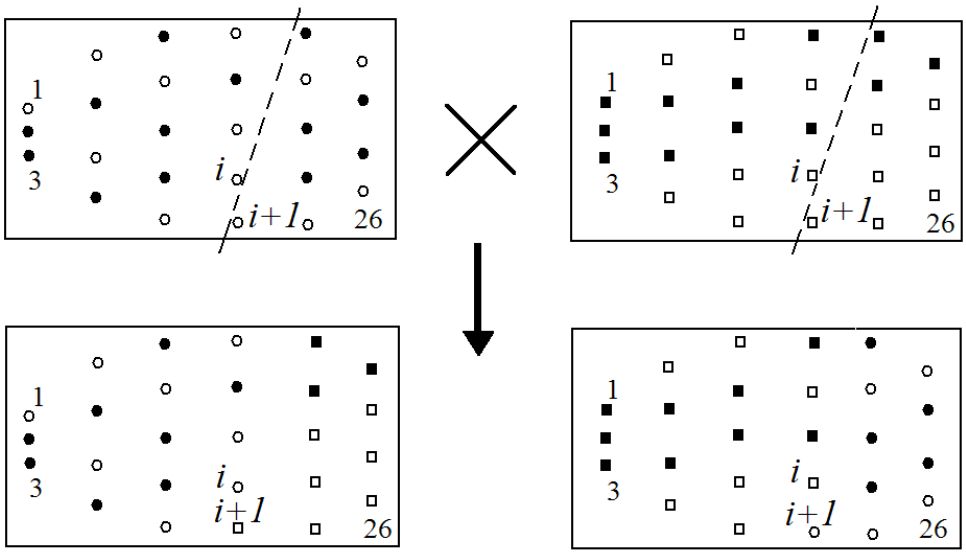
Používame dva druhy kríženia jedincov. Pri *AngleCrossover* majú nové jedince uhol rovný priemeru uhlov rodičov – kríženie priemerom:

$$(\alpha,(z_{1,}b_1),\ldots(z_{26,}b_{26}))\ \boldsymbol{X}\ (\tilde{\alpha},(\tilde{z_{1,}}\tilde{b_1}),\ldots(\tilde{z_{26,}}\tilde{b_{26}}))$$
$$\downarrow$$
$$(\frac{(\alpha+\tilde{\alpha})}{2,}(z_{1,}b_1),\ldots(z_{26,}b_{26})),(\frac{(\alpha+\tilde{\alpha})}{2,}(\tilde{z_{1,}}\tilde{b_1}),\ldots(\tilde{z_{26,}}\tilde{b_{26}}))$$

Pri *CutCrossover* sa náhodne zvolí index i a rodičia si vymenia časti chromozómu za týmto indexom:

$$(\alpha,(z_{1,}b_1),\ldots(z_i,b_i),(z_{i+1},b_{i+1}),\ldots(z_{26,}b_{26}))$$
$$\boldsymbol{X}$$
$$(\tilde{\alpha},(\tilde{z_{1,}}\tilde{b_1}),\ldots(\tilde{z_i},\tilde{b_i}),(\tilde{z_{i+1}},\tilde{b_{i+1}}),\ldots(\tilde{z_{26,}}\tilde{b_{26}}))$$
$$\downarrow$$
$$(\alpha,(z_{1,}b_1),\ldots(z_i,b_i),(\tilde{z_{i+1}},\tilde{b_{i+1}}),\ldots(\tilde{z_{26,}}\tilde{b_{26}})),$$
$$(\tilde{\alpha},(\tilde{z_{1,}}\tilde{b_1}),\ldots(\tilde{z_i},\tilde{b_i}),(z_{i+1},b_{i+1}),\ldots(z_{26,}b_{26}))$$

Keďže vrcholy referenčnej plochy sú topologicky usporiadané v smere osi x (po dĺžke listu vrtule), pri *CutCrossover* dochádza k výmene súvislých častí listov (obr. 6.6).



Obr. 6.6: príklad kríženia CutCrossover

Navrhli sme 4 operátory mutácie. Pri *AngleMutation* sa k uhlu pripočítalo náhodne zvolené δ z normálneho rozdelenia pravdepodobnosti:

$$(\alpha,(z_{1,}b_1),...(z_{26,}b_{26}))\rightarrow(\alpha+\delta,(z_{1,}b_1),...(z_{26,}b_{26})),\ \delta=\boldsymbol{N}(0,\sigma)$$

Zabezpečilo sa, aby nový uhol zostal v intervale (0,1).

Pri *GaussAllMutation* všetky vyvýšenia z_i boli upravené podľa:

$$(\alpha,(z_{1,}b_1),...(z_{26,}b_{26}))\rightarrow(\alpha,(z_1+\delta_{1,}b_1),...(z_{26}+\delta_{26,}b_{26})),$$
$$\delta_i=\boldsymbol{N}(0,\sigma),i=1,2,...26$$

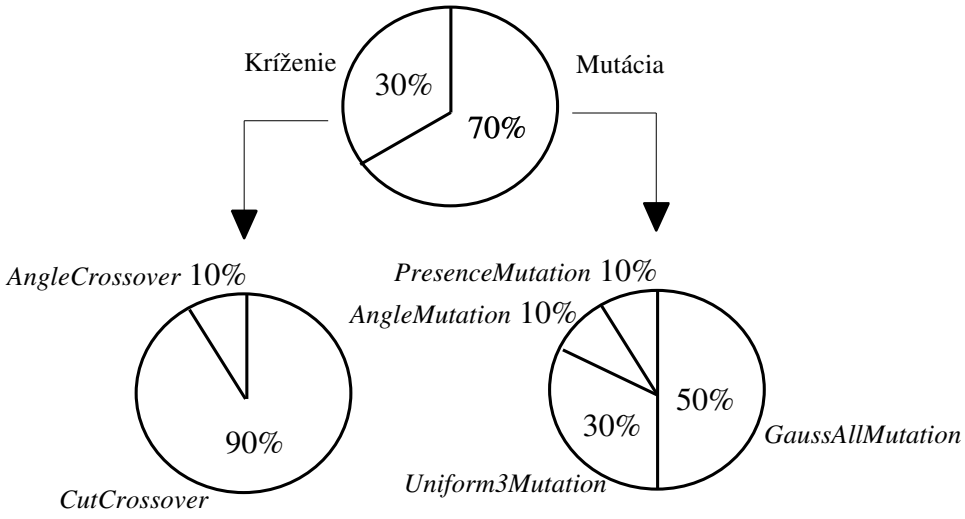
Táto mutácia modifikovala z_i tak, aby nové z_i boli z intervalu (-4, 4).

Pri *Uniform3Mutation* 3 náhodne zvolené vrcholy zmenili hodnotu z_i rovnomerne náhodne z intervalu (-4, 4):

$$(\alpha,(z_{1,}b_1),...(z_i,b_i),...(z_j,b_j),...(z_k,b_k),...(z_{26,}b_{26}))$$
$$\downarrow$$
$$(\alpha,(z_{1,}b_1),...(\tilde{z}_i,b_i),...(\tilde{z}_j,b_j),...(\tilde{z}_k,b_k),...(z_{26,}b_{26})),$$
$$\tilde{z}_i=\boldsymbol{R}(-4,4),\tilde{z}_j=\boldsymbol{R}(-4,4),\tilde{z}_k=\boldsymbol{R}(-4,4)$$

Pri *PresenceMutation* trom náhodne zvoleným vrcholom bola bitovo negovaná hodnota b_i (zmenila sa "prítomnosť" vrchola v liste):

$$(\alpha,(z_{1,}b_1),...(z_i,b_i),...(z_j,b_j),...(z_k,b_k),...(z_{26,}b_{26}))$$
$$\downarrow$$
$$(\alpha,(z_{1,}b_1),...(z_i,1-b_i),...(z_j,1-b_j),...(z_k,1-b_k),...(z_{26,}b_{26})),$$



Obr. 6.7: pravdepodobnosti operácií kríženia a mutácie (PR)

Pre každého jedinca populácie nastávala najprv operácia kríženia s pravdepodobnosťou 0,3 a následne potom operácia mutácie s pravdepodobnosťou 0,7. Pri krížení bola s pravdepodobnosťou 0,9 zvolená operácia *CutCrossover,* v zvyšných prípadoch išlo o *AngleCrossover*. V prípade, že je zvolená mutácia, aplikuje sa práve jedna z uvedených 4 mutácií. Pravdepodobnosti *GaussAllMutation, Uniform3Mutation, PresenceMutation,* resp. *AngleMutation* sú 0,5; 0,3; 0,1 resp. 0,1 (obr. 6.7).

Operátory kríženia a mutácie sú implementované tak, aby výsledný list vrtule nepresahoval rozmery 10x8x8. To znamená, že pri konštrukcii modelu vrtule v simulátore nie je potrebné kontrolovať rozmery listu vrtule. Preto fitness každého jedinca je určená až po ukončení simulácie – na rozdiel od nepriamej reprezentácie.

6.2.4 Evolúcia

Populácia obsahovala 200 jedincov. Na začiatku evolúcie boli všetky parametre inicializované rovnomerne náhodne z príslušných definičných oborov. Pri selekcii rodičov sa používal turnaj veľkosti 2. Nová generácia populácie bola vytvorená *generačnou výmenou* s *miernym elitizmom*. Evolúcia skončila po dosiahnutí 200. generácie.

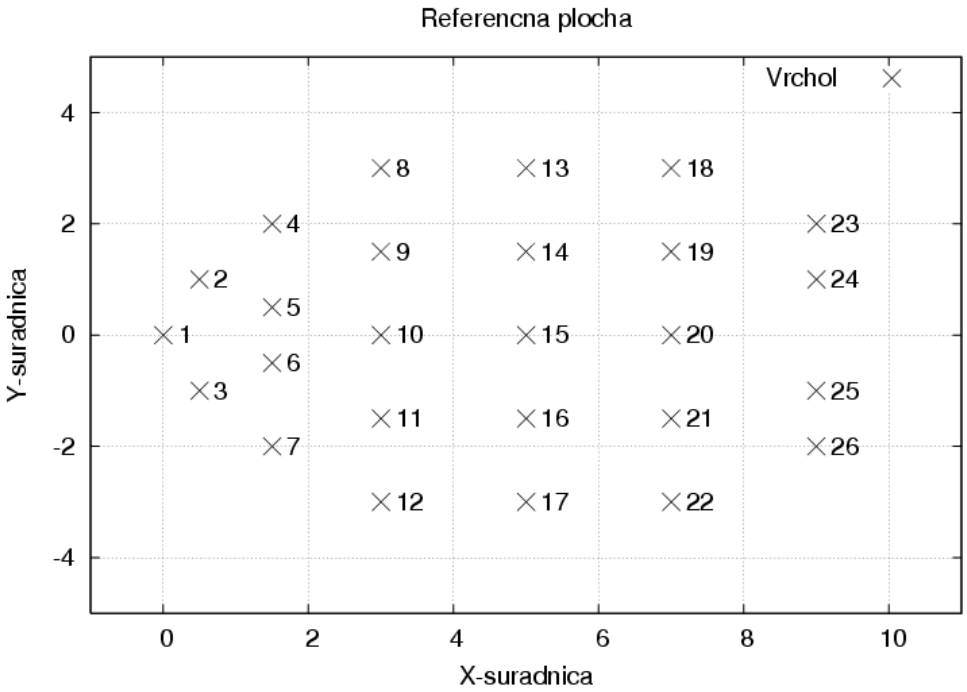
6.3 Nepriama reprezentácia II

List vrtule je vytvorený pomocou krivej (triangulovanej) plochy v trojrozmernom priestore, ktorá vychádza z referenčnej plochy podobne ako pri priamej reprezentácií. Rozloženie vrcholov referenčnej plochy je však menej roztiahnuté (obr. 6.8). Pozície vrcholov referenčnej plochy v priestore sú upravené pomocou *stromu transformácií* (podobný ako pri nepriamej reprezentácií). Vnútornými vrcholmi stromu sú transformačné operácie – posunutie a otočenie. Listami stromu sú indexy vrcholov referenčnej plochy. Transformácie na ceste z koreňa stromu do listu postupne (v danom poradí) menia pozíciu vrchola, ktorý prislúcha danému listu.

Delaunayova triangulácia určí množinu trojuholníkov nad týmito vrcholmi, ktorou definujeme krivú plochu listu vrtule. Delaunayova triangulácia je nájdená pre vrcholy referenčnej plochy ešte pred[34] aplikovaním transformácií.

[34] máme algoritmus na získanie Delaunayovej triangulácie pre dvojrozmerný priestor, preto hľadáme trianguláciu množiny vrcholov referenčnej plochy (a nie množiny vrcholov už posunutých v trojrozmernom priestore)

List každej vrtule obsahuje všetkých 26 vrcholov, a preto je Delaunayova triangulácia pre všetky vrtule rovnaká. Vrtule sa líšia v umiestnení a otočení trojuholníkov, čo sa dosiahne úpravou pozícií vrcholov trojuholníkov.



Obr. 6.8: rozloženie vrcholov referenčnej plochy v rovine XY (NR2)

Vrcholy referenčnej plochy *nie* sú roztiahnuté v obdĺžniku 10x8 tak ako pri priamej reprezentácií (porovnaj obr. 6.4 a obr. 6.8). Dôvodom je obmedzenie veľkosti listu vrtule na pomyslený kváder 10x8x8 a možnosť meniť pozíciu vrchola voľnejšie ako pri priamej reprezentácií. Vrchol nie je iba kolmo vyvýšený vzhľadom na referenčnú plochu, ale môže byť posunutý v ľubovoľnom smere. Pri roztiahnutom rozložení vrcholov referenčnej plochy by sa ľahko mohlo stať, že aj menšie transformácie okrajových vrcholov by spôsobili presahovanie daného kvádra.

6.3.1 Štruktúra jedinca

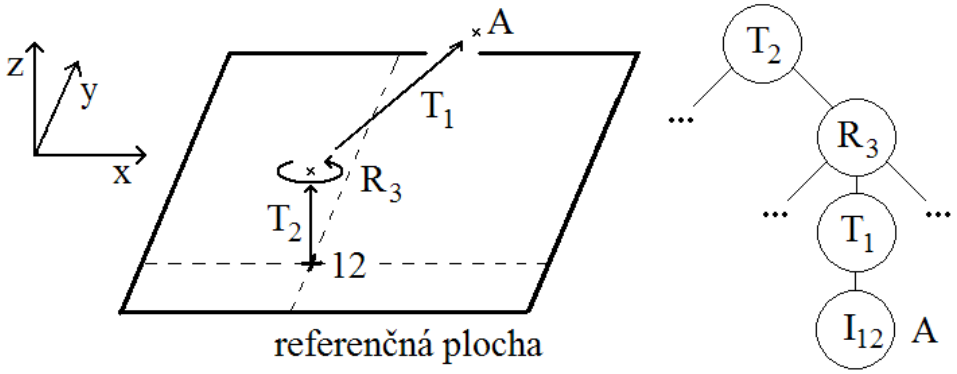
Genotypom jedinca je strom s veľkosťou vetvenia 1-3 a s 26 listami. Vrcholy stromu sú vyberané z množiny F.

$F = \{I_1, I_2, ... I_{26}, R_1, T_1, R_2, T_2, R_3, T_3\}$, kde

$$R_i(x, y, z, \alpha) \in (-0{,}5; -0{,}5)\,x\,(-0{,}5; 0{,}5)\,x\,(-1; 1)\,x\,(0; \pi),\ \ i = 1, 2, 3$$

$$T_i(dx, dy, dz) \in (-0{,}5; 0{,}5)\,x\,(-0{,}5; 0{,}5)\,x\,(-1; 1),\ \ i = 1, 2, 3$$

R_i reprezentuje otočenie *(rotate)* okolo osi *(x, y, z)* o uhol α
T_i reprezentuje posunutie (*translate*) podľa vektora *(dx, dy, dz)*
I_1, I_2, ... I_{26} sú listami stromu. I_i prislúcha *i*-temu vrcholu referenčnej plochy. Každý I_i sa nachádza v strome práve raz. Postupné aplikovanie transformácií uložených vo vrcholoch na ceste z koreňa stromu do listu I_i na *i*-ty vrchol referenčnej plochy určí výslednú pozíciu príslušného vrcholu listu vrtule (obr. 6.9). Vrcholy R_i, T_i majú *i* synov. Parametre v transformačných vrcholoch sa rovnomerne náhodne inicializujú z príslušných intervalov pri vytváraní počiatočnej populácie. K ich zmene dochádza aj počas mutácií jedincov.



Obr. 6.9: výsledné umiestnenie vrcholu A plochy listu vrtule prislúchajúceho vrcholu referenčnej plochy s indexom 12 (podľa uvedeného stromu transformácií)

Pri vytváraní modelu vrtule (fenotypu) podľa uvedeného genotypu sa vypočítajú súradnice 26 vrcholov v trojrozmernom priestore, ktoré sú vrcholmi trojuholníkov určujúcich krivú plochu listu vrtule. Referenčná plocha je naklonená o 18° z horizontálnej polohy, aby sa naznačil žiadaný smer rotácie vrtule. Overí sa, či rozmery listu vrtule nepresahujú rozmery 10x8x8. Ak presahujú, simulácia sa nespúšťa a jedincovi je priradená nulová fitness.

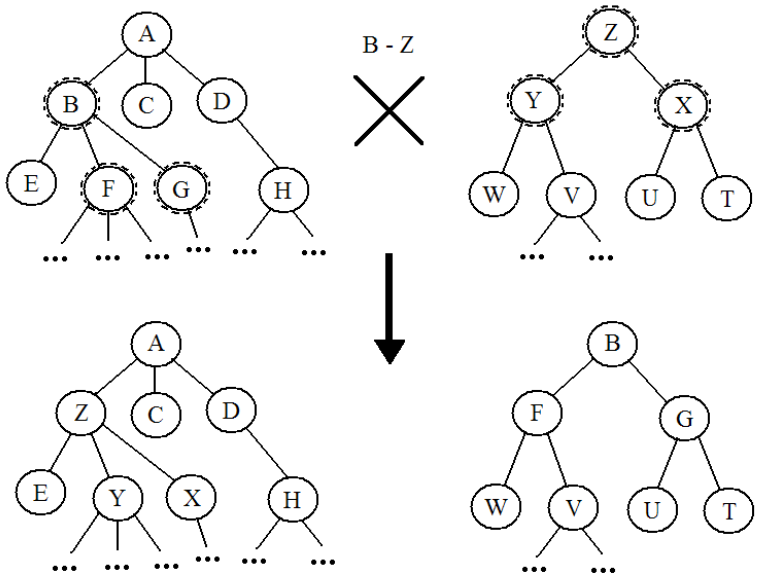
6.3.2 Genetické operátory

Z definície genotypu jedinca vyplýva, že vypísaním listov (I_1, I_2, ... I_{26}) po prehľadaní ľubovoľného stromu do hĺbky získame jednu z 26! permutácií. Táto vlastnosť stromov musí byť zachovaná počas celej evolúcie. Pri inicializácií

počiatočnej populácie sa konštruujú náhodné stromy s 26 rôznymi listami. Pri štandardnom[35] stromovom krížení a mutácii by však mohli vzniknúť stromy s viac alebo menej listami a/alebo duplicitnými alebo chýbajúcimi listami – výpis listov stromu by nebol permutáciou. Pri kombinatorických optimalizačných problémoch [12], kde jedinec predstavuje jednu z permutácií objektov[36], sa preto používajú operátory kríženia a mutácie, ktoré:

- vytvárajú nové jedince tak, aby nenarušili podmienku zachovania permutácie
- vytvoria nové jedince pomocou štandardných operátorov a následne aplikujú opravné procedúry, ktoré z novovytvorených jedincov vytvoria permutácie

My sme pri návrhu operátorov zvolili prvý uvedený spôsob. Evolučný algoritmus aplikoval jeden druh kríženia – *SwapCrossover*. V stromoch oboch rodičov sa náhodne zvolilo po jednom vrchole. Tieto vrcholy si vymenili svoj obsah – transformačnú informáciu. Následne si čo najviac ich transformačných synovských vrcholov podobne vymenilo svoj obsah (obr. 6.10).



Obr. 6.10: príklad kríženia SwapCrossover

35 pozri *Genetické programovanie* v kapitole Evolučné algoritmy
36 napríklad TSP – problém obchodného cestujúceho [12]

Na mutáciu jedinca boli k dispozícií 4 rôzne operátory. Pri *ContentGauss-Mutation* sa v náhodne zvolenom vrchole stromu upravili parametre transformácie podľa normálneho rozdelenia pravdepodobnosti (σ=0,1):

$$\mathrm{R}_j(x,y,z,\alpha) \rightarrow \mathrm{R}_j(x+\mathbf{N}(0,\sigma), y+\mathbf{N}(0,\sigma), z+\mathbf{N}(0,2\sigma), \alpha+\mathbf{N}(0,\sigma))$$

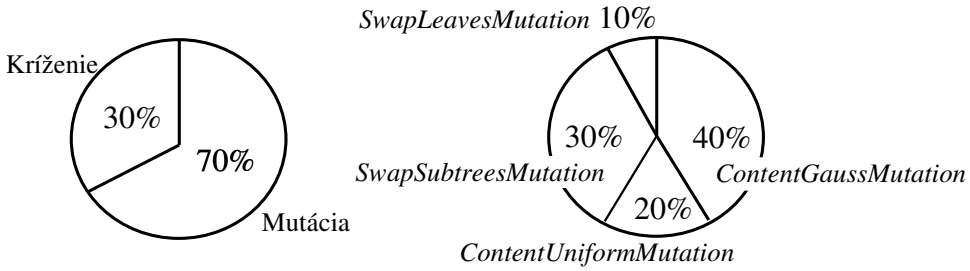
$$\mathrm{T}_k(dx,dy,dz) \rightarrow \mathrm{T}_k(dx+\mathbf{N}(0,\sigma), dy+\mathbf{N}(0,\sigma), dz+\mathbf{N}(0,2\sigma))$$

Pri *ContentUniformMutation* sa v náhodne zvolenom vrchole upravili parametre transformácie rovnomerne náhodne (ako pri inicializácii):

$$\mathrm{R}_j(x,y,z,\alpha) \rightarrow \mathrm{R}_j(\mathbf{R}(-0{,}5;0{,}5), \mathbf{R}(-0{,}5;0{,}5), \mathbf{R}(-1;1), \mathbf{R}(0,\pi))$$

$$\mathrm{T}_k(dx,dy,dz) \rightarrow \mathrm{T}_k(\mathbf{R}(-0{,}5;0{,}5), \mathbf{R}(-0{,}5;0{,}5), \mathbf{R}(-1;1))$$

Pri *SwapSubtreesMutation* sú náhodne zvolené 2 vrcholy stromu, ktoré sa spolu s ich podstromami navzájom vymenia. Jeden zo zvolených vrcholov nesmie byť potomkom druhého. Je to jediná mutácia, ktorá mení štruktúru stromu. *SwapLeavesMutation* je špeciálnym prípadom predošlej mutácie – zvolia sa náhodne 2 listy, ktoré sa vymenia. Táto mutácia vlastne predstavuje výmenu celých postupností transformácií aplikovaných na vrcholy referenčnej plochy prislúchajúce daným listom.



Obr. 6.11: pravdepodobnosti operácií kríženia a mutácie (NR2)

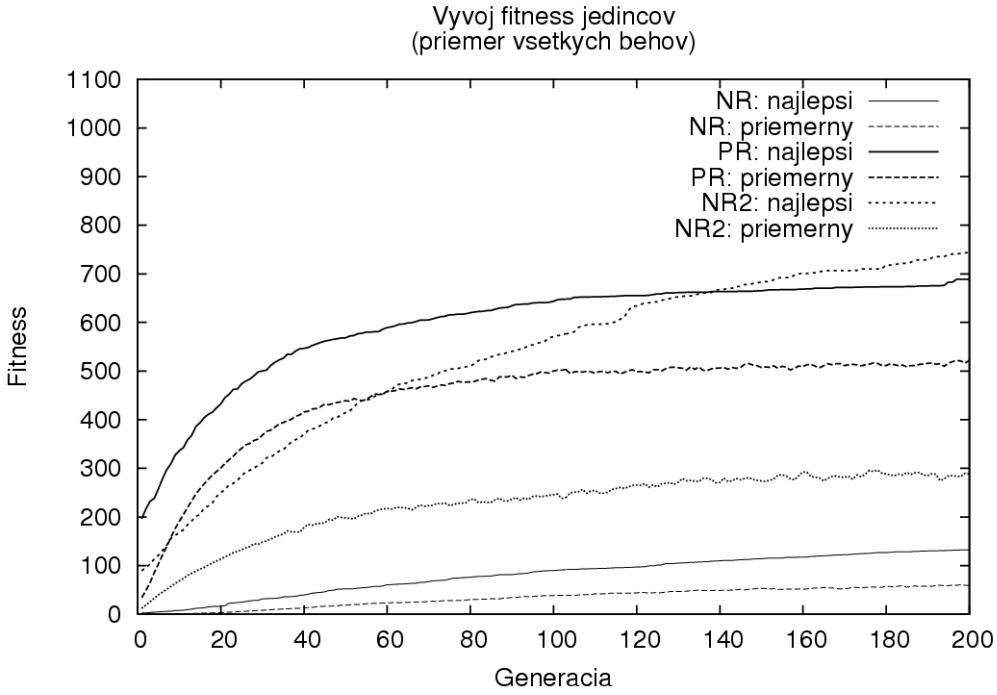
V každej iterácii evolučného algoritmu každý jedinec v populácii podstúpil buď kríženie, alebo mutáciu (paralelný prístup). Operácia kríženia nastávala s pravdepodobnosťou 0,3 a operácia mutácie s pravdepodobnosťou 0,7. V prípade, že je zvolená mutácia, aplikuje sa práve jedna z uvedených 4 mutácií. Pravdepodobnosti *ContentGaussMutation, ContentUniformMutation, Swap-SubtreesMutation,* resp. *SwapLeavesMutation* sú 0,4; 0,2; 0,3 resp. 0,1 (obr. 6.11).

6.3.3 Evolúcia

Počiatočná populácia (200 jedincov) bola vytvorená generovaním náhodných stromov s 26 listami. Ďalšie parametre evolučného algoritmu boli rovnaké ako pri predošlých reprezentáciách.

7 Výsledky

Pre každú reprezentáciu sme vykonali 11 evolučných výpočtov pri pevne daných nemenných parametroch. Každá evolúcia trvala 200 generácií[37], veľkosť populácie bola 200 jedincov.



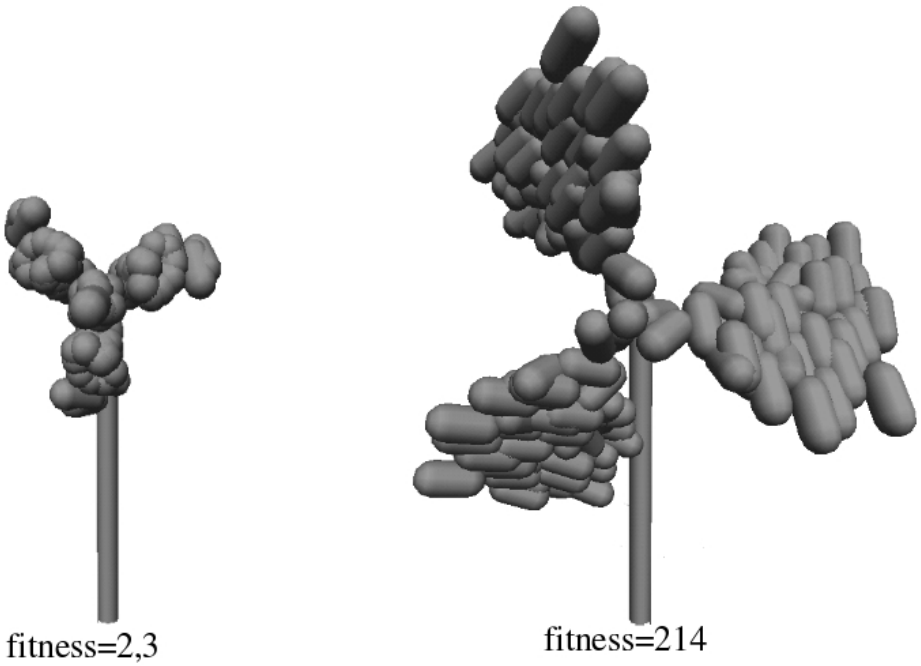
Obr. 7.1: vývoj fitness najlepšieho jedinca a priemernej fitness populácie (v závislosti od počtu generácií)

Nepriama reprezentácia I (NR) vytvárajúca vrtule pomocou stavebných jednotiek (kapsúl) dosahovala priemernú najlepšiu fitness (154,69) podstatne menšiu ako priama reprezentácia (PR) (fitness = 688,77) a nepriama reprezen-

[37] s výnimkou piatich evolúcií pri priamej reprezentácií, ktoré skončili medzi 193. a 200. generáciou

tácia II (NR2) (fitness = 743,89), ktoré použili krivú plochu na definovanie tvaru listu vrtule (obr. 7.1).

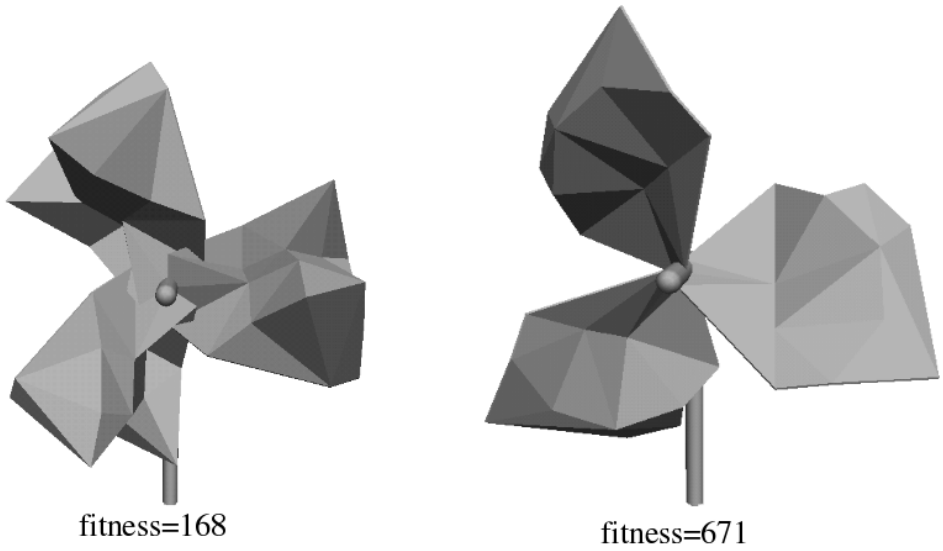
Ukážku niektorých z najlepších vrtúľ prvej a poslednej generácie vybraných evolúcií možno vidieť na obr. 7.2 – 7.4 a v prílohe B. Čitateľ si určite všimne, že naevolvované vrtule sa nepodobajú skutočným veterným vrtuliam, ale skôr vyzerajú ako lodné vrtule. Vzhľadom na použitý fyzikálny model nezohľadňujúci aerodynamické a materiálové vlastnosti danej vrtule by bolo trúfalé očakávať nájdenie vrtule podobnej skutočnej veternej vrtuli. To ani nebolo cieľom. Cieľom bolo preskúmať rôzne reprezentácie objektov z hľadiska evolučných výpočtov.



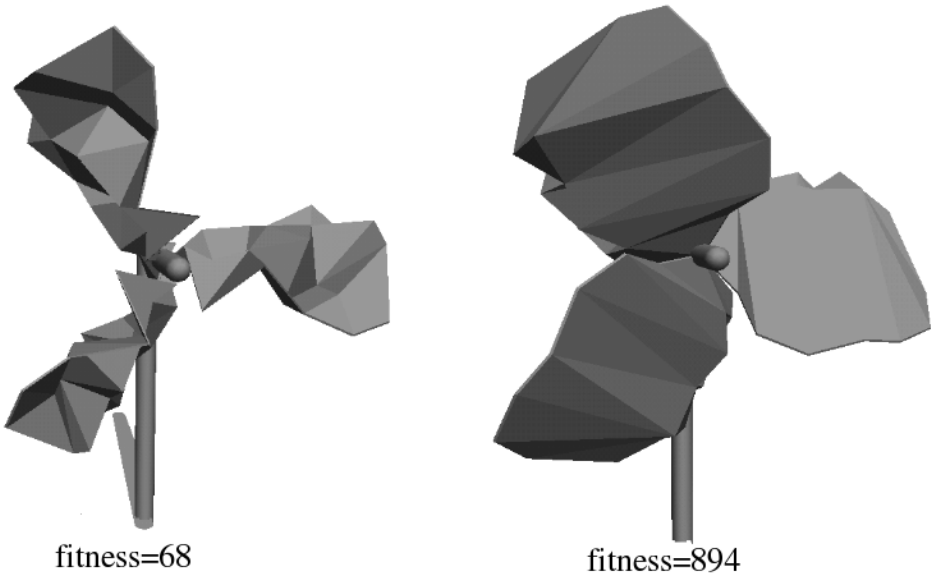
Obr. 7.2: fitness a tvar najlepšej vrtule v prvej a poslednej generácii – NR2

Nízku dosiahnutú kvalitu (fitness) vrtúľ pri NR možno vysvetliť tým, že evolúcia začínala s malými stromami. Strom určoval umiestnenie kapsúl vzhľadom na os otáčania vrtule a malé stromy vytvorili malé vrtule s malým momentom zotrvačnosti a malou schopnosťou prijať energiu vetra z celého veterného koridoru (rozkrútili sa na nízku uhlovú rýchlosť, boli ľahké, teda ich rotačná energia bola nízka). Počas evolúcie musela vrtuľa "vyrásť" na celú šírku koridoru. V tomto smere boli PR a NR2 zvýhodnené, lebo od začiatku evolúcie mohli prijať viac energie (list vrtule bol od začiatku dostatočne veľký)

a zamerali sa preto hlavne na nájdenie vhodného tvaru. Aj po roztiahnutí listu pri NR na maximálnu povolenú dĺžku nedosahuje vrtuľa vysokú kinetickú rotačnú energiu, lebo sa nedokáže rozkrútiť na dostatočnú rýchlosť.

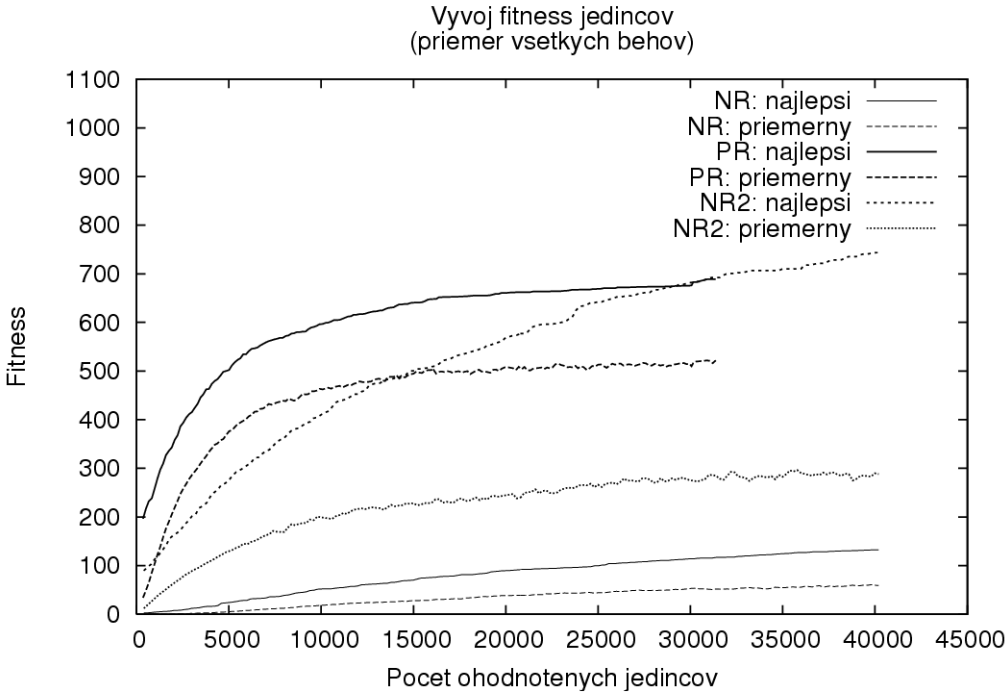


Obr. 7.3: fitness a tvar najlepšej vrtule v prvej a poslednej generácii – PR



Obr. 7.4: fitness a tvar najlepšej vrtule v prvej a poslednej generácii – NR2

Dôvodom nízkej uhlovej rýchlosti je tvar a veľkosť plochy nábežnej hrany listu, ktorá je kvôli použitiu kapsúl značne veľká a kladie veľký odpor pri rotácií vrtule.

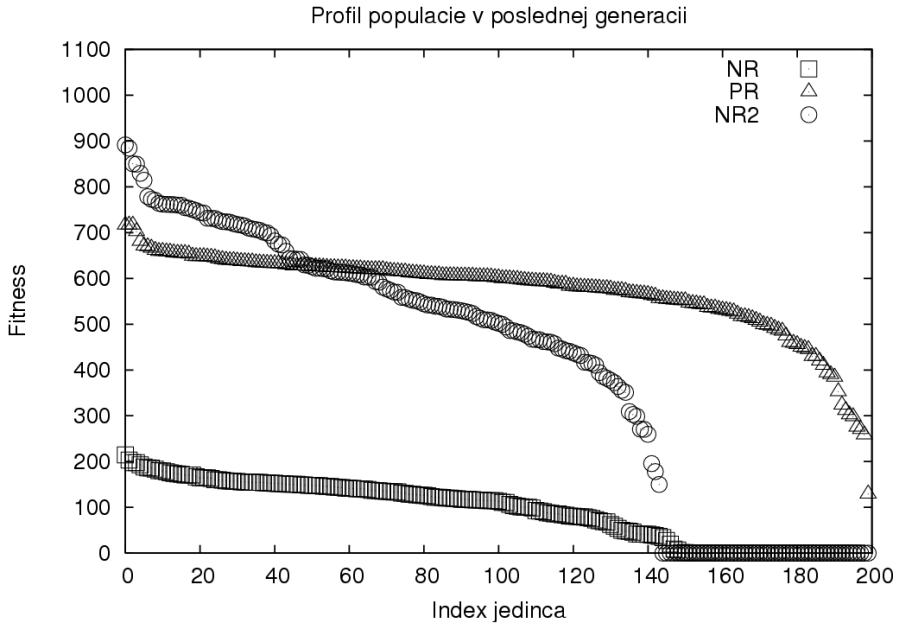


Obr. 7.5: vývoj fitness najlepšieho jedinca a priemernej fitness populácie (v závislosti od počtu ohodnotených jedincov)

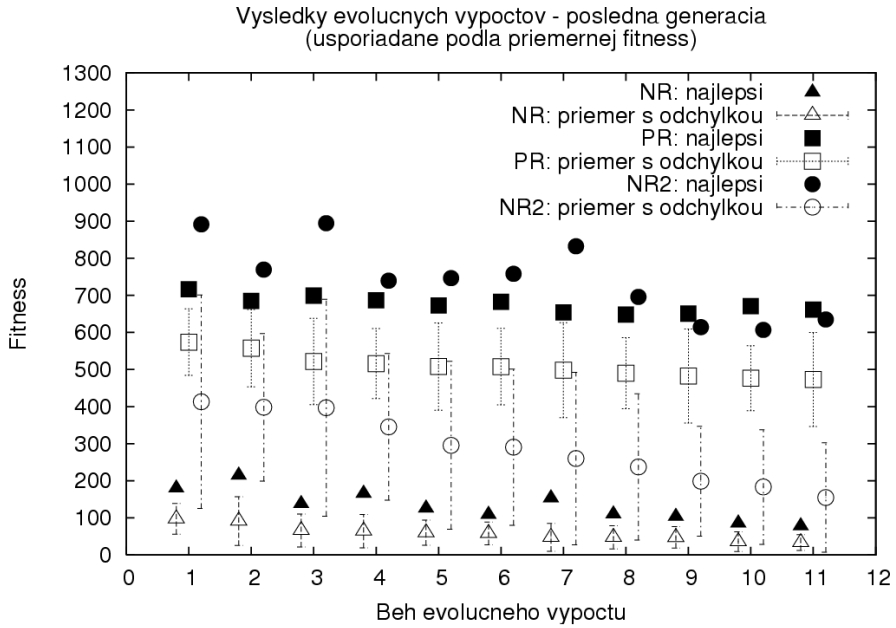
Na obr. 7.5 je zobrazený vývoj fitness v závislosti od počtu ohodnotených jedincov. Rozdiel v počte ohodnotených jedincov (graf pre PR končí skôr ako NR a NR2) je spôsobený charakterom vytvárania novej generácie populácie. Pri PR sa používa sekvenčná pravdepodobnostná aplikácia zvoleného operátora kríženia a zvoleného operátora mutácie. Pri daných pravdepodobnostiach (p = 0,3 pre kríženie a p = 0,7 pre mutáciu) boli niektoré jedince aj krížené aj mutované a iné zostali nezmenené. V každej iterácii genetického algoritmu bolo vytvorených v priemere okolo 160 nových jedincov, do počtu 200 ich doplnili jedince, ktoré už boli ohodnotené v predchádzajúcej iterácii. Pri nepriamych reprezentáciách (NR a NR2) sa používalo genetické programovanie s paralelným aplikovaním operátora kríženia a operátora mutácie. Každý jedinec bol buď krížený (p = 0,7) alebo mutovaný (p = 0,3). V každej iterácii tak vzniklo 200 nových jedincov. Mohlo by sa zdať, že v tomto smere

NR a NR2 prehľadali väčší priestor jedincov, avšak treba pripomenúť, že pri krížení a mutácii stromových štruktúr v genetickom programovaní vznikali aj jedince, ktoré nespĺňali rozmerové kritérium. Vznikali jedince, ktoré nepatrili do priestoru povolených tvarov listu vrtule. Tieto jedince boli ohodnotené nulovou fitness. Na obr. 7.6 je ukážka profilu populácie v poslednej generácii vybraných evolúcií. Jedincov s nulovou fitness pri NR a NR2 je okolo 50. Menší počet novovytvorených jedincov pri PR je kompenzovaný menším počtom legálnych z 200 novovytvorených jedincov pri NR a NR2. Výsledná veľkosť prehľadaného priestoru počas 200 generácií je teda pre všetky reprezentácie podobná.

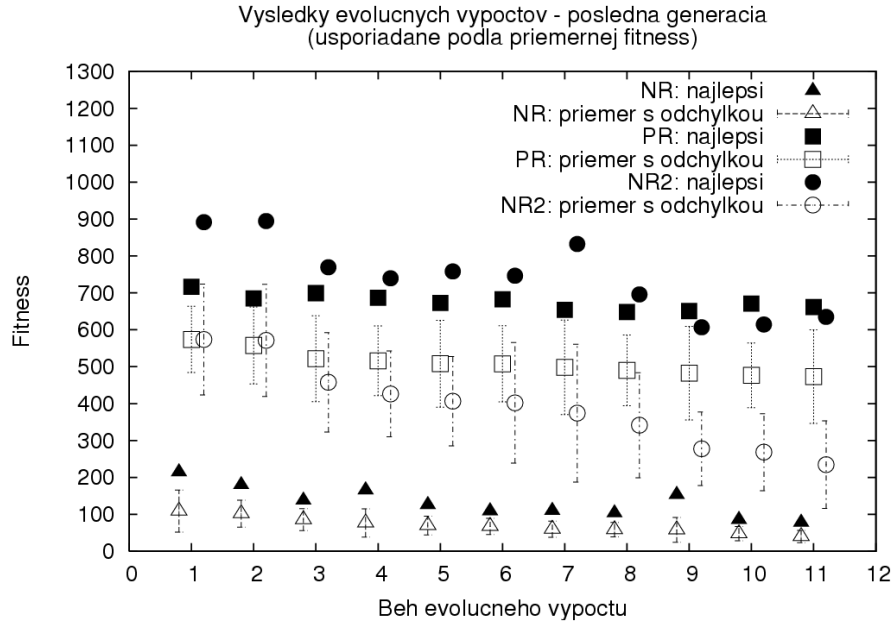
Výsledky individuálnych evolúcií sú zhrnuté v grafe na obr. 7.7. Ukazujú fitness najlepšieho jedinca v danej evolúcii a priemernú fitness populácie spolu so štandardnou odchýlkou. Graf na obr. 7.8 je modifikáciou grafu na obr. 7.7 – priemer a štandardná odchýlka sú určené iba z legálnych jedincov v populácii (s nenulovou fitness).



Obr. 7.6: profil populácie v poslednej generácii (údaje z evolúcie s najlepšou dosiahnutou priemernou fitness)



Obr. 7.7: výsledná fitness najlepšieho jedinca a priemerná fitness populácie (spolu s odchýlkou) pre všetky evolúcie



Obr. 7.8: výsledná fitness najlepšieho jedinca a priemerná fitness populácie (spolu s odchýlkou) ak zohľadňujeme len legálne jedince (s nenulovou fitness) pre všetky evolúcie

Na základe uvedených výsledkov môžeme konštatovať, že pri PR veľká časť populácie skonvergovala do optima a priemerná fitness populácie je vyššia ako pri NR2, takisto štandardná odchýlka je menšia – populácia je homogénnejšia. Táto skutočnosť je pozorovateľná aj na obr. 7.6 (sklon grafu – distribúcia jedincov s rôznou fitness).

dosiahnutie	***70% výslednej fitness***	***80% výslednej fitness***	***90% výslednej fitness***
NR	107	133	164
PR	27	42	80
NR2	82	108	142

Tab. 7.1: Rýchlosť konvergencie fitness najlepšieho jedinca – generácia, kedy sa dosiahlo % výslednej fitness

Rýchlosť konvergencie (tab. 7.1) je pri NR najnižšia. PR konverguje výrazne rýchlejšie ako NR2 – v 80. generácii dosahuje PR už 90% výslednej fitness. Na druhej strane vývoj najlepšej a priemernej fitness (obr. 7.1, resp. obr. 7.2) naznačuje, že NR2 má väčšiu tendenciu rásť aj v ďalšom pokračovaní evolúcie (po prekročení 200. generácie) na rozdiel od PR.

Z vizuálneho hľadiska (obr. 7.3 a 7.4) sú PR a NR2 podobné, pri NR2 má krivá plocha konštantne 26 vrcholov, pri PR je plocha vplyvom menšieho počtu vrcholov jednoduchšia. Pri niektorých vrtuliach je pozorovateľné "zneužitie" väčšieho počtu vrcholov prítomných v liste na vytváranie plôch v "závetrí" listu, ktoré prispievajú k zvyšovaniu hmotnosti listu a momentu zotrvačnosti vrtule. Tým sa zvyšuje kinetická energia vrtule, aj keď je uhlová rýchlosť nižšia. Voľnejšie umiestňovanie vrcholov pri NR2 umožnilo zväčšiť plochu listu vrtule v blízkosti ložiska. List vrtule pri PR bol viac obmedzený rozložením vrcholov referenčnej plochy.

Pri PR uhol naklonenia referenčnej roviny (súčasť genetickej informácie jedinca) konvergoval k hodnotám okolo 10° (pri NR2 sme používali pevný uhol naklonenia 18°).

Pri evolučných výpočtoch predchádzajúcich naše experimenty sme hľadali vhodné nastavenia parametrov evolučného algoritmu a genetických operátorov. Vyskúšali sme meniť veľkosť turnaja. Ukázalo sa, že väčšie turnaje (>3) spôsobovali predčasnú konvergenciu populácie. Turnaj veľkosti 2 sa ukázal najlepší pri pomerne malej populácii 200 jedincov – zabezpečil prehľadávanie väčšieho priestoru jedincov. Pri evolúcii NR sme v vyskúšali viaceré genotypy jedincov – stromy s väčším vetvením. Väčšie vetvenie pri ohraničenej veľkosti stromu zmenšovalo hĺbku stromov. Pozície kapsúl tvoriacich list vrtule tak boli

ovplyvnené menším počtom transformácií a kapsule sa nedostali dostatočne ďaleko od ložiska vrtule – vrtuľa nepokryla celú šírku veterného koridoru. Preto sme nakoniec obmedzili veľkosť vetvenia maximálne na 3.

7.1 Zhrnutie

Pri experimente s nepriamou reprezentáciou (NR) pomocou VRML navrhnutou Ebnerom sme potvrdili jej schopnosť evolúcie. V podstate z ničoho (len zhluk niekoľkých kapsúl koncentrovaných v osi otáčania) dokázala pomocou štandardných operátorov kríženia a mutácie vzniknúť vrtuľa pokrývajúca celú šírku veterného koridoru s tvarom umožňujúcim rozkrútiť vrtuľu na istú rýchlosť. Hlavným prínosom experimentu s touto reprezentáciou bolo overenie výsledkov Ebnerovho experimentu.

Nové reprezentácie využívajúce krivú plochu na konštrukciu listu vrtule sa ukazujú lepšie z hľadiska dosahovanej fitness vrtule. Experimenty potvrdzujú vhodnosť nových reprezentácií, navrhnutých genetických operátorov a ich parametrov. Nastavenie parametrov operátorov a parametrov evolučného algoritmu bolo výsledkom mnohých evolučných výpočtov predchádzajúcich samotné experimenty.

Napriek rôznosti v spôsobe mapovania genotypu na fenotyp (priamy vs. nepriamy), v použitom evolučnom algoritme (genetický algoritmus vs. genetické programovanie) a aplikovaných operátorov kríženia a mutácie môžeme porovnávať priamu (PR) a nepriamu (NR2) reprezentáciu. NR2 napriek väčšej odchýlke pri 11 uskutočnených evolúciách dosahuje vyššiu priemernú najlepšiu fitness. Štatisticky sme pomocou t-testu potvrdili, že v našich experimentoch NR2 dosahuje lepšie výsledky ako PR ($t = 2{,}29$).

PR poskytuje rýchlejšie relatívne dobré výsledky, ale rýchlo konverguje do lokálneho optima, z ktorého má malú šancu úniku (populácia je časom veľmi homogénna). Príčinou môže byť lokálny charakter mutácií, ktoré neumožňujú väčšie skoky pri prehľadávaní priestoru.

NR2 má väčšiu exploratívnu silu – prehľadáva širší priestor jedincov (rôznorodosť populácie v priebehu evolúcie) a to umožňuje nájdenie jedincov s vyššou fitness. Konverguje pomalšie ako PR, ale pri dlhších evolúciách nachádza lepšie riešenia. Potenciálnu výhodu tejto reprezentácie predstavuje komprimovaná informácia uložená v stromovej štruktúre. Táto informácia určujúca fenotypické vlastnosti je hierarchicky organizovaná. Operátor mutácie modifikáciou vhodnej časti stromu umožňuje veľké fenotypické zmeny

jedincov, čím sa sústreďuje na "prieskumné" prehľadávanie priestoru jedincov. Modifikáciou iných častí stromu sa vie zase zamerať aj na úpravy malých častí jedinca (listu vrtule), pričom neovplyvňuje iné. Dôsledkom je väčšia flexibilita nepriamej reprezentácie. Táto vlastnosť zabraňuje nežiadúcim konvergenciám v lokálnom optime.

Na základe uskutočnených experimentov však nemožno vo všeobecnosti povedať, že nepriama reprezentácia je pre evolučné výpočty lepšia ako priama. Vždy záleží od konkrétneho problému. Existuje veľa priamych a nepriamych reprezentácií, ktoré by sme pri danom probléme mohli použiť. Medzi nimi sú aj vhodné, aj nevhodné reprezentácie.

7.2 Prínos distribuovaného systému

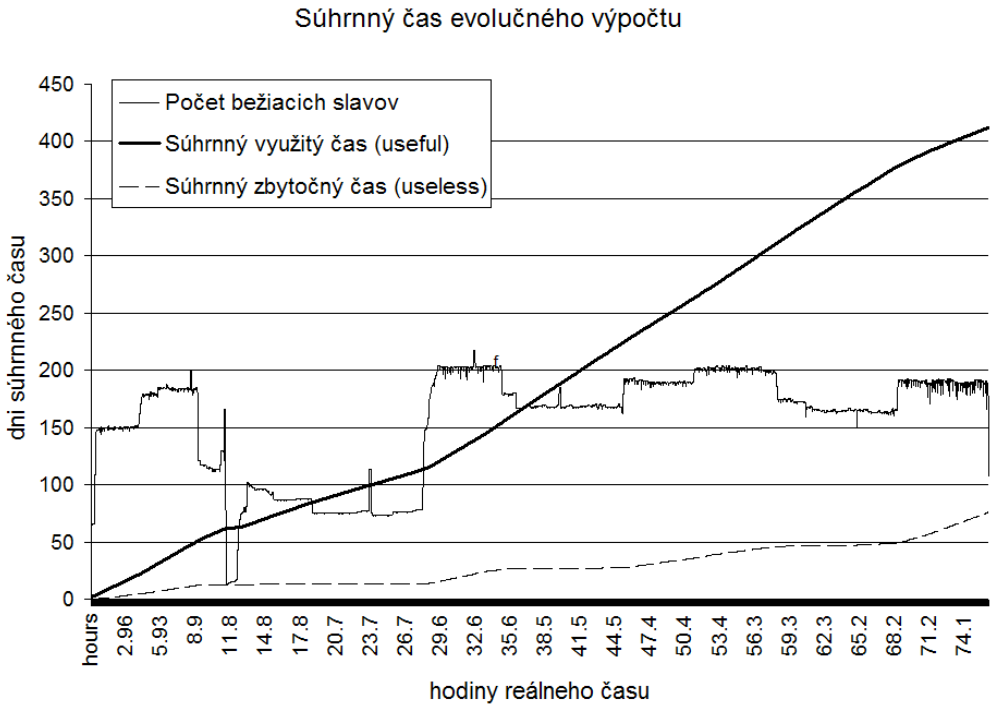
Prínos distribuovaného systému je bezpochyby veľký – umožnil praktické nasadenie evolučných algoritmov na riešenie reálneho problému.

Pri každom experimente sme mali k dispozícií desiatky počítačov (najmä v počítačových učebniach). *Slave* program mohol byť spustený alebo zastavený v ľubovoľnom okamihu, neovplyvnilo to evolučný výpočet. Počet paralelne pracujúcich *slave*-ov bol premenlivý a závisel napríklad od dennej doby. V nočných hodinách sa počet bežiacich *slave* programov vyšplhal nad 200.

Pri evolučných výpočtoch s NR[38] najlepší *slave* dokázal ohodnotiť v priemere za hodinu 42,58 jedincov, priemerný *slave* 35,13 jedincov. Jedna evolúcia vyžadovala ohodnotenie 40 000 jedincov (200 jedincov počas 200 generácií). Najlepšiemu *slave*-ovi by to trvalo vyše 39 dní, priemernému *slave*-ovi vyše 47 dní. Vďaka distribuovanému systému, ktorý zapojil do výpočtu niekedy až vyše 200 *slave*-ov, sme získali výsledky 11 evolúcií do štyroch dní.

Graf na obr. 7.9 zobrazuje počet paralelne bežiacich *slave*-ov a súhrnné časy výpočtu počas 11 paralelne vykonávaných evolúcií (NR). Krivky súhrnných časov majú väčší sklon, keď je väčší počet *slave*-ov (súhrnný čas rastie rýchlejšie, keď pracuje paralelne viac *slave*-ov). Za 75 hodín výpočtu bol celkový čas využitých výpočtov (*useful time)* všetkých *slave* programov väčší ako jeden rok.

[38] simulácia vrtule konštruovanej pomocou kapsúl bola pri väčšom počte kapsúl časovo náročnejšia ako vrtule konštruovanej pomocou krivej plochy

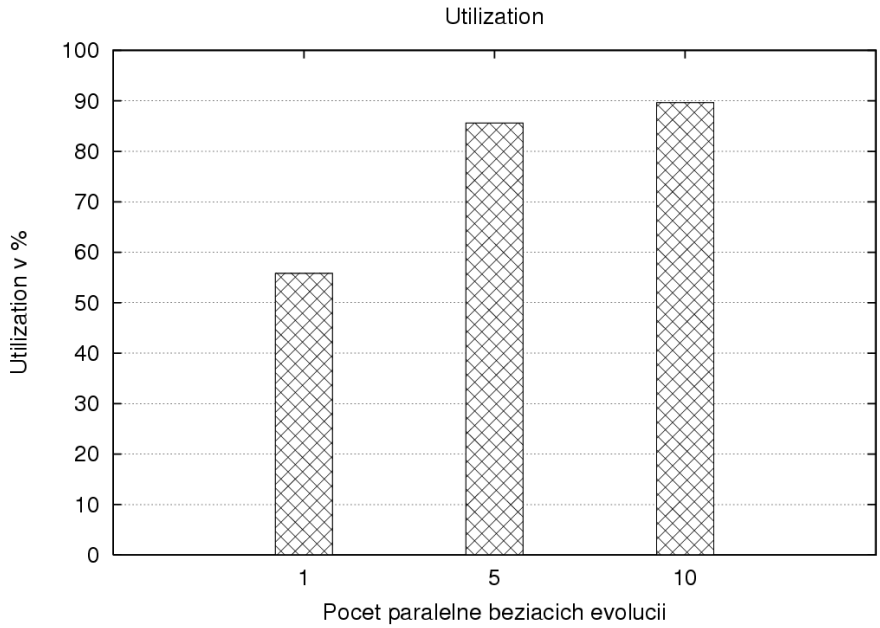


Obr. 7.9: súhrnný čas 11 evolúcií s reprezentáciou NR

Master program vykonáva evolučný algoritmus po iteráciách – generáciách. Aby vytvoril novú generáciu populácie, musí najprv získať hodnotu fitness pre každého novovytvoreného jedinca. Všetkých novovytvorených jedincov pošle do databázy a čaká na ich ohodnotenie *slave* programami. Pri veľa paralelne pracujúcich *slave*-och dochádza v každej iterácií po čase k zamestnaniu všetkých *slave*-ov na posledných pár jedincov ("koniec generácie") – výslednú fitness jedinec získa ale len od jedného *slave*-a. Čas výpočtu tohto *slave*-a sa označuje ako využitý čas (*useful time*). Čas výpočtu ostatných *slave*-ov sa označuje ako zbytočný čas (*useless time*). Po ohodnotení všetkých jedincov *master* program načíta z databázy tieto hodnoty fitness, vykoná jednu iteráciu evolučného algoritmu a odošle novú populáciu neohodnotených jedincov do databázy. Za ten čas *slave* programy čakajú – zvyšuje sa čas nečinnosti (*idle time*). Pomer

$$\frac{(\sum useful\,time)}{(\sum useful\,time + \sum useless\,time + \sum idle\,time)}$$

vyjadruje efektivitu paralelizmu (*utilization*).



Obr. 7.10: Utilization - efektívnosť paralelizmu

Graf na obr. 7.9 zobrazuje *utilization* pre 1, 5 a 10 súbežných evolúcií[39]. Paralelne vykonávané evolúcie používajú tú istú tabuľku v databáze na zverejnenie neohodnotených jedincov. Každý *slave* program tak môže ohodnocovať jedincov všetkých evolúcií. Zmyslom súbežných evolúcií je eliminácia javu "konca generácie". Ak *master* program jednej evolúcie čaká už iba na pár jedincov, nezamestnáva zbytočne veľa *slave* programov na týchto pár jedincoch. Väčšina *slave*-ov ohodnocuje v tom čase jedincov iných evolúcií. Dôsledkom je vyššia *utilization*. Pri našich experimentoch sme dosahovali *utilization* 85–90%.

[39] rovnaký typ evolúcií – napr. všetky evolúcie s NR2

8 Záver

Evolučný dizajn hľadá optimálne tvary objektov aplikovaním evolučných metód. Význačnou podúlohou v evolučnom dizajne je navrhnutie vhodných reprezentácií pre tieto objekty.

Reprodukovali sme experiment Marca Ebnera [1], implementovali sme jednu nepriamu reprezentáciu pomocou scénického grafu VRML. Podobne ako Ebner sme sa sústredili na konkrétny príklad evolúcie vrtule veterného mlyna. Navrhli a implementovali sme 2 nové reprezentácie trojrozmerných objektov využívajúce komplexnejšiu konštrukciu trojrozmerného objektu pomocou krivej plochy. Pre navrhnuté reprezentácie sme vytvorili špecifické genetické operátory. Ohodnotenie jedinca v evolučnom algoritme vyžadovalo simuláciu fyzikálneho modelu prúdenia vzduchu cez príslušnú vrtuľu.

V experimentoch sme sa snažili porovnať uvedené reprezentácie z viacerých hľadísk – vhodnosti pre evolúciu, výslednej dosahovanej fitness a rýchlosti konvergencie. Všetky tri reprezentácie potvrdili vhodnosť pre evolučné výpočty, nepriama reprezentácia pomocou krivej plochy dosahovala najvyššie hodnoty fitness jedincov (najkvalitnejšie vrtule), priama reprezentácia pomocou krivej plochy konvergovala najrýchlejšie.

Predstavením a detailným popisom uvedených experimentov sme čitateľovi priblížili postupy systematického riešenia komplexných problémov evolučnými algoritmami. Na prekonanie neúnosnej časovej zložitosti evolučných výpočtov sme implementovali distribuovaný systém.

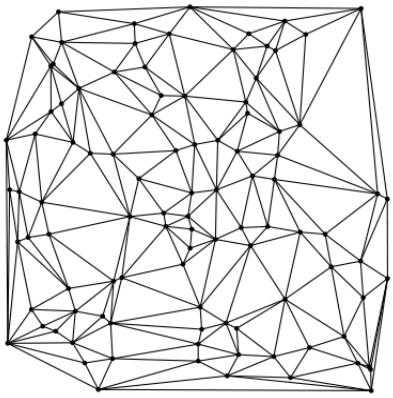
Zďaleka sme nevyčerpali problematiku priamych a nepriamych reprezentácií. Dosiahnuté výsledky s našimi reprezentáciami tiež nemožno zovšeobecniť na porovnanie priamych a nepriamych reprezentácií v evolučných algoritmoch. Táto práca však môže slúžiť ako východiskový bod pre ďalší výskum. Je možné pokračovať skúmaním modifikácií uvedených reprezentácií (iné rozloženie a rôzny počet vrcholov referenčnej plochy, možné dynamické pridávanie vrcholov na vyladenie častí listu vrtule, väčšie stromy s obmedzenými transformačnými vrcholmi a pod.), alebo navrhnutím nových reprezentácií. Ďalej by bolo zaujímavé používať detailnejší a fyzikálne korektnejší model prúdenia vzduchu cez vrtuľu opierajúci sa o teóriu

aerodynamiky a materiálové a konštrukčné vlastnosti vrtule (zrušenie nášho predpokladu absolútnej pevnosti a neforemnosti materiálu vrtule). Takýto model by však vyžadoval iný simulátor. Naša fitness funkcia – maximálna dosiahnuteľná kinetická energia pri danom vetre – by sa dala modifikovať tak, aby upustila od požiadavky neustáleho rovnomerného vetra, prípadne by uvažovala iné atribúty: množstvo použitého materiálu, zložitosť konštrukcie a pod.

Ďalším možným pokračovaním práce by mohlo byť overenie vlastností našich reprezentácií pri konštrukčných úlohách v iných doménach evolučného dizajnu (napríklad hľadanie tvaru stabilného stojanu, hľadanie tvaru optimálneho zberača objektov z plochy a iné).

Príloha A

Delaunayova triangulácia



Obr. A.1: príklad Delaunayovej triangulácie množiny 100 vrcholov (prevzatý z [28])

Triangulácia množiny vrcholov V (v dvojrozmernom priestore) je množina trojuholníkov

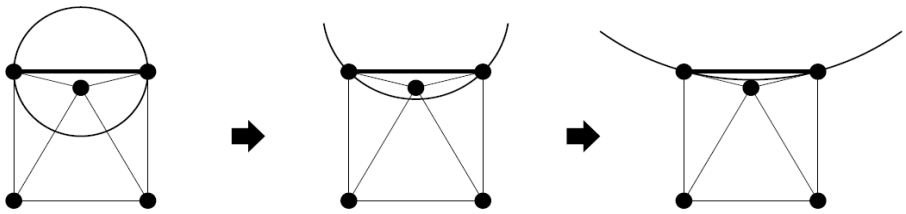
- ktorých vrcholy tvoria množinu V
- ktoré sa navzájom nepretínajú
- ktorých zjednotenie tvorí konvexný obal množiny V.

Existuje exponenciálne veľa triangulácii množiny V. V tejto kapitole opíšeme špeciálnu trianguláciu, ktorá ma dobré vlastnosti. Pre danú množinu vrcholov V existuje jediná, za predpokladu, že žiadne 4 vrcholy neležia na jednej kružnici.

Delaunayova triangulácia množiny vrcholov *V* je triangulácia, v ktorej každá hrana je delaunayovská. Hrana (*u*,*v*) je *delaunayovská*, ak existuje prázdny kruh (neexistuje vrchol vnútri kruhu), na ktorého obvode sa nachádzajú body *u* a *v*.

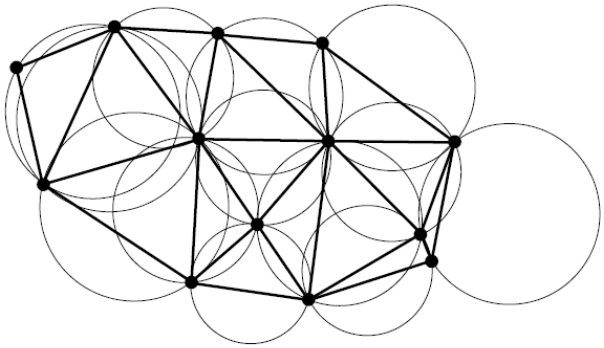
Uvedieme niekoľko liem a viet bez dôkazu (dôkazy možno nájsť v [20]), ktoré sú základom jednoduchého algoritmu, ktorý generuje Delaunayovu trianguláciu.

Veta (dvojrozmerný priestor): Ak žiadne 4 vrcholy množiny V neležia na jednej kružnici, tak množina všetkých delaunayovských hrán je rozkladom konvexného obalu množiny V a tvorí Delaunayovu trianguláciu množiny V.



Obr. A.2: hrany konvexného obalu sú delaunayovské

Lema 1: Každá hrana konvexného obalu je delaunayovská (obr. A.2).



Obr. A.3: delaunayovské trojuholníky

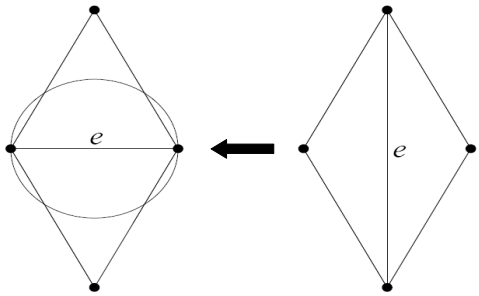
Lema 2: Pre každý trojuholník Delaunayovej triangulácie existuje prázdny kruh, na ktorého obvode ležia vrcholy trojuholníka (takýto trojuholník nazývame *delaunayovský* (obr. A.3)).

Lema 3: Každá hrana spájajúca vrchol s jeho najbližším susedom je delaunayovská.

Lema 4: Nech T je triangulácia množiny V. Ak každý trojuholník triangulácie T je delaunayovský, tak každá hrana triangulácie T je delaunayovská.

Hrana e je *lokálne delaunayovská*, ak je delaunayovská vzhľadom iba na dva trojuholníky, ktorých je spoločnou hranou.

Lema 5: Nech *e* je hrana triangulácie množiny *V*. Buď je e lokálne delaunayovská, alebo *e* je preložiteľná a hrana, ktorá vznikne preložením hrany *e* je lokálne delaunayovská.



Obr. A.4: lokálne delaunayovská hrana (vľavo)

Algoritmus (*Delaunay Flip Algorithm*):

1 Nech *V* je množina vstupných vrcholov.

2 Vytvor ľubovoľnú trianguláciu *T* množiny *V*.

3 Pokým všetky hrany triangulácie *T* nie sú delaunayovskej, opakuj:

3.1 Nájdi nedelaunayovskú hranu, ktorá je preložiteľná a prelož ju.

Lema 6: Ak *T* je trianguláciou množiny *V* so všetkými hranami lokálne delaunayovskými, tak potom *T* je delaunayovskou trianguláciou množiny *V*.

Dôsledok: Ak triangulácia obsahuje nedelaunayovskú hranu, potom obsahuje aj hranu, ktorá nie je lokálne delaunayovská. A teda algoritmus sa zastaví po konečnom počte krokov.

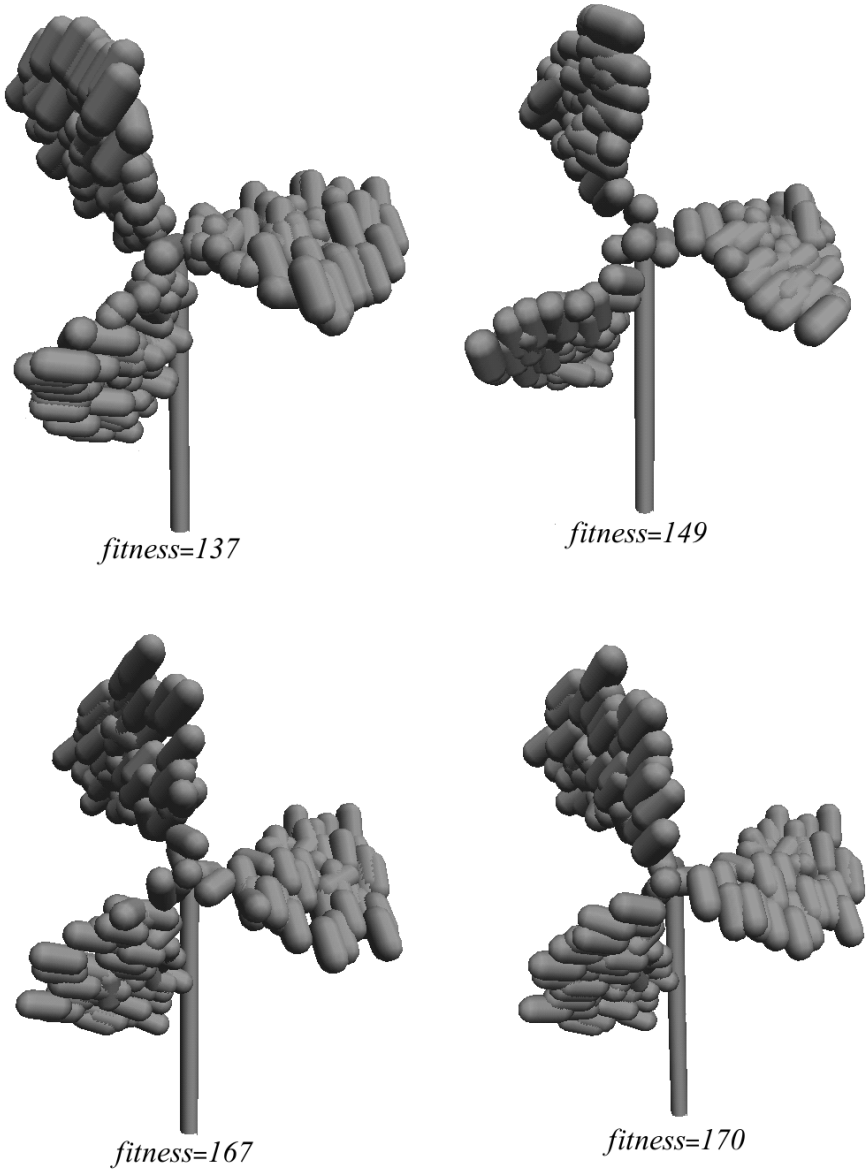
Uvedený algoritmus má časovú zložitosť $O(n^3)$. Pri implementácii sme využili triangulačnú knižnicu [19], ktorá obsahuje efektívnejší algoritmus s časovou zložitosťou $O(n \log n)$. Táto zložitosť sa dosiahne použitím metódy *Divide-and-Conquer*. Ukázali sme algoritmus, ktorý zastaví a pri zastavení všetky hrany triangulácie sú delaunayovské. Teda algoritmus generuje pre ľubovoľnú množinu vrcholov *V* Delaunayovu trianguláciu.

Medzi dobré vlastnosti Delaunayovej triangulácie patria:

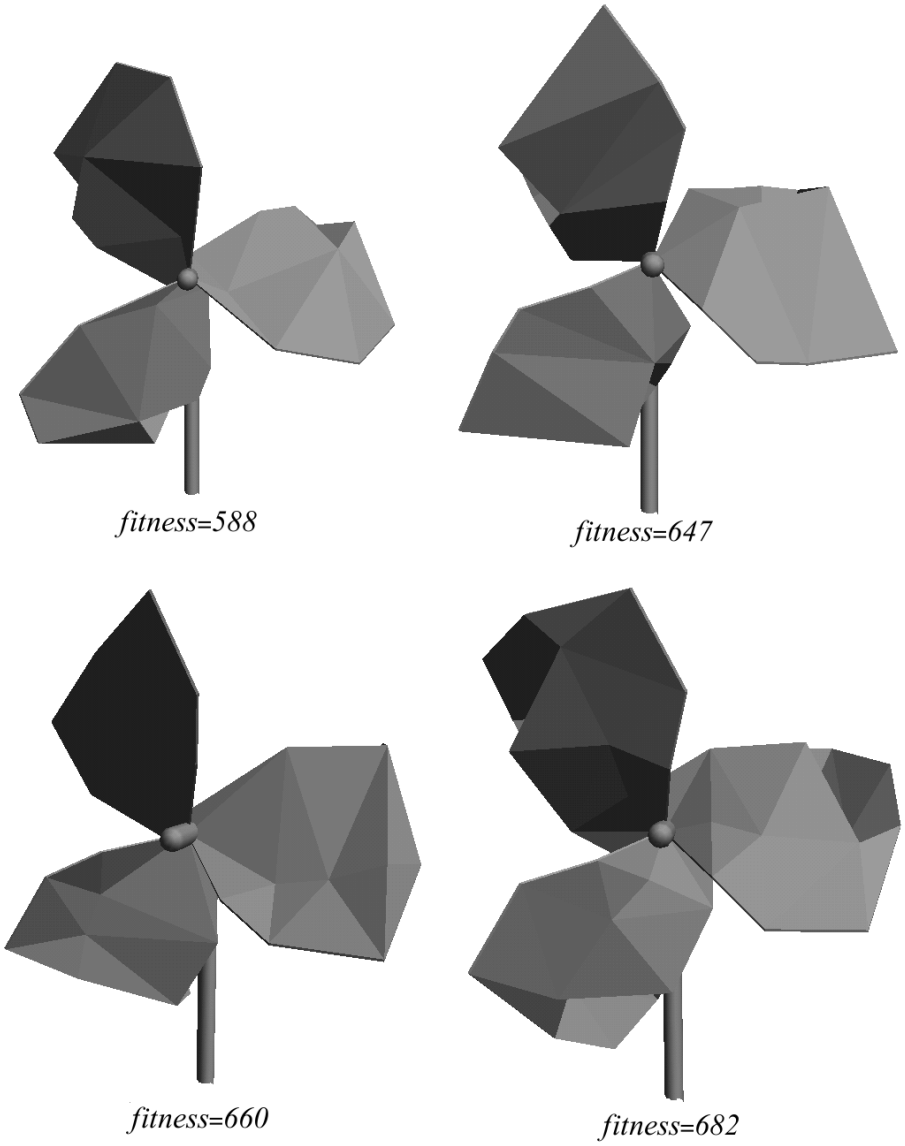
– maximalizuje minimálny uhol. V porovnaní s inými trianguláciami, najmenší uhol Delaunayovej triangulácie je väčší alebo rovný ako najmenší uhol ľubovoľnej inej triangulácie

- minimalizuje obvody trojuholníkov
- minimalizuje sa najväčšia opísaná kružnica ľubovoľného trojuholníka, vďaka tomu vyzerá triangulácia “rovnomernejšie”

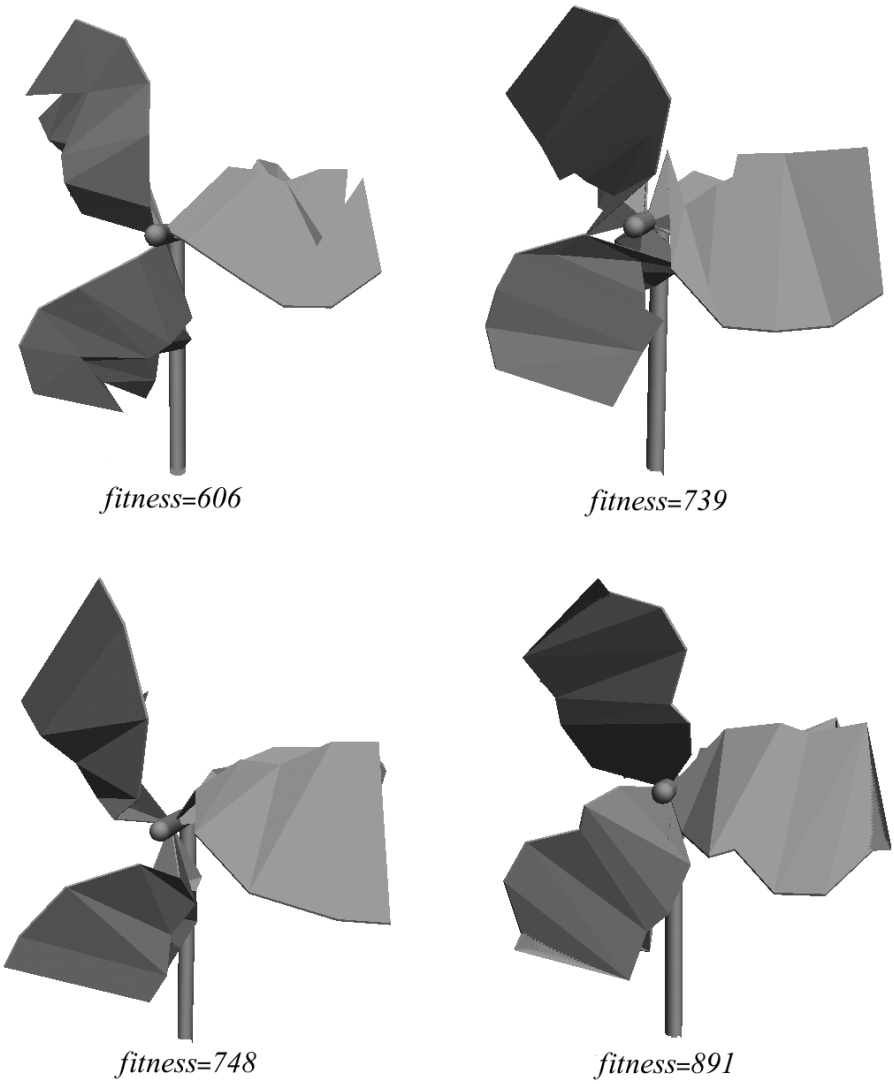
Príloha B



Obr. B.1: vrtule v poslednej generácii vybraných evolúcií - NR



Obr. B.2: vrtule v poslednej generácii vybraných evolúcií - PR



Obr. B.3: vrtule v poslednej generácii vybraných evolúcií - NR2

Literatúra

[1] Marc Ebner. *Evolutionary Design of Objects Using Scene Graphs*. Universität Würzburg.

[2] A. E. Eiben, J. E. Smith. *Introduction to Evolutionary Computing*. Springer, 2003.

[3] V. Kvasnička, J. Pospíchal, P. Tiňo. *Evolučné algoritmy*. STU Bratislava, 2000.

[4] J.H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan, MI 1975.

[5] L.J. Fogel, A.J. Owens, M.J. Walsh. *Artificial Intelligence through Simulated Evolution*. J.Wiley, New York 1966

[6] H.-P. Schwefel. *Evolution and Optimum Seeking*. J. Wiley, New York 1995

[7] J. Koza. *Genetic Programming. On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA 1992.

[8] M. Keijzer, J. J. Merelo, G. Romero, M. Schoenauer. *Evolving Objects: a general purpose evolutionary computation library.* Danish Hydraulik Institute, Universidad de Granada, CNRS and Ecole Polytechnik

[9] EO library: http://eodev.sourceforge.net

[10].Genetic algorithms FAQ : http://www.cs.cmu.edu/Groups/AI/html/faqs/ai/genetic/top.html

[11].E. Burke, S. Gustafson, G. Kendall, *Ramped Half-n-Half Initialisation Bias in GP*. University of Nottingham

[12] P. Larranaga, C.M.H. Kuijpers, R.H. Murga, I. Inza, S. Dizdarevic. *Genetic Algorithms for the Travelling Salesman Problem: A review of Representations and Operators.* University of the Basque

[13] Russell Smith. *Open Dynamics Engine*: http://www.ode.org

[14] A. M. Law. *From Simulation Modeling and Analysis, 4th Edition*. 2007. ISBN (978)-0-07-110336-8(7)

[15] Nick Jakobi. *Minimal Simulations for Evolutionary Rototics*. University of Sussex, 1998.

[16] *Newton Game Dynamics*™: http://www.newtondynamics.com

[17] *Havok Physics*™: http://www.havok.com/content/view/17/30/

[18] Pavel Petrovič. *Distributed system for Evolutionary Robotics Experiments*. Norwegian University of Science and Technology. Technical report 05/2004. ISSN 1503-416X

[19] *Triangle++*: http://www.compgeom.com/~piyush/scripts/triangle/

[20] J.R. Shewchuck. *Delaunay Refinement Mesh Generation*. Carnegie Melon 1997

[21] IBM. *The Blue Gene*: *http://www.research.ibm.com/bluegene*

[22] *CPR cluster FMFI UK*: http://ii.fmph.uniba.sk/~mnagy/index.php?file=Cluster.xml

[23] *Condor Project*: http://www.cs.wisc.edu/condor

[24] Z. Constantinescu, P. Petrovič, A. Pedersen, D. Federici. *Q²ADPZ (Quite Advanced Distributed Parallel Zystem)*: http://qadpz.idi.ntnu.no

[25] *Berkeley Open Infrastructure for Network Computing*: http://boinc.berkeley.edu

[26] *Folding@Home Distributed Computing*: http://folding.stanford.edu

[27] http://www.knoppix.net/wiki/Knoppix_Remastering_Howto

[28] *The Free encyclopedia*: http://www.wikipedia.org

[29] *GA-lib*: http://lancet.mit.edu/galib-2.4/

[30] *OpenBEAGLE*: http://beagle.gel.ulaval.ca/

[31] T. Laue, K. Spiess, T. Röfer. *SimRobot - A General Physical Robot Simulator and Its Application in RoboCup*. In A. Bredenfeld, A. Jacoff, I. Noda, Y. Takahashi (Eds.), *RoboCup 2005: Robot Soccer World Cup IX*, No. 4020, pp. 173–183, Lecture Notes in Artificial Intelligence. Springer, 2006.

[32] J. Go, B. Browning, M. Veloso. *Accurate and flexible simulation for dynamic,vision-centric robots*. In: *Proceedings of International Joint Conference on Autonomous Agents and Multi-Agent Systems* (AAMAS'04), 2004.

[33] J.C. Zagal, J.R. del Solar. *UCHILSIM: A Dynamically and Visually Realistic Simulator for the RoboCup Four Legged League*. In: *RoboCup 2004: Robot Soccer World Cup VIII*. Lecture Notes in Artificial Intelligence, Springer, 2004.

[34] O. Michel, Cyberbotics Ltd. *WebotsTM: Professional Mobile Robot Simulation*, pp. 39-42, *Int. Journal of Advanced Robotic Systems*, Vol. 1 Number 1 (2004), ISBN 1729-8806.