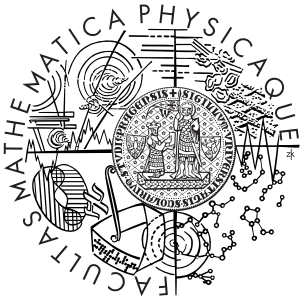
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Pavol Jusko

Použití metod evolučního modelování ve fyzice

Katedra fyziky povrchů a plazmatu

Prof. RNDr. Rudolf Hrach, DrSc.

Studijní program: Obecná fyzika

2007

Ďakujem Prof. RNDr. Rudolfovi Hrachovi, DrSc. za vedenie práce a cenné pripomienky.

Prohlašuji, že jsem svou bakalářskou práci napsal(a) samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne Pavol Jusko

Obsah

1 Úvod **5**
1.1 Prehľad kapitol 5
1.2 Algoritmus inšpirovaný prírodou 5
1.3 Analýza obrazu 6

2 Evolučný a genetický algoritmus **7**
2.1 Princípy a názvoslovie 7
2.2 Zakódovanie a hodnotenie 8
2.3 Výber 8
2.4 Kríženie a mutácia 10
2.5 Ukončenie algoritmu 11

3 Hľadanie najkratšej cesty **12**
3.1 Odveký problém 12
3.2 Zakódovanie, fitness, výber 12
3.3 Mutácie a heuristika 13
3.4 Rýchlosť algoritmu 15
3.5 Implementácia 17

4 Použitie vo fyzike **20**
4.1 Usporiadanie povrchu 20
4.2 Najkratšia spojnica a QC v závislosti na D 20
4.3 Nedeterministickosť evolučného algoritmu 21

5 Záver **26**
5.1 Porovnanie metód QC a dĺžka najkratšej spojnice 26
5.2 Možnosti metódy najkratšej spojnice 26
5.3 Ďalšie využitie genetických algoritmov 28

Literatúra **29**

A CD-ROM **30**

Název práce: Použití metod evolučního modelování ve fyzice
Autor: Pavol Jusko
Katedra (ústav): Katedra fyziky povrchů a plazmatu
Vedoucí bakalářské práce: Prof. RNDr. Rudolf Hrach, DrSc.
e-mail vedoucího: Rudolf.Hrach@mff.cuni.cz

Abstrakt: V tejto práci sme sa zamerali na možnosť použitia evolučných algoritmov vo fyzike. Ako problém, ktorý je aplikovateľný na konkrétnu fyzikálnu úlohu a zároveň je skúmaný metódami počítačovej fyziky, sme zvolili problém obchodného cestujúceho. Jeho riešenie, teda nájdenie najkratšej uzavretej spojnice množiny bodov, ktorá prechádza každým bodom práve raz, využijeme pri klasifikácii usporiadanosti povrchových štruktúr. Výsledky tejto metódy porovnávame s dobre známou metódou Quadrat Counts ako aj s difúznou zónou, teda parametrom ovplyvňujúcim štruktúry pri ich vytváraní pomocou metódy hard disk. Pozornosť venujeme aj optimalizácii použitého algoritmu.

Klíčová slova: Evolúcia, najkratšia spojnica, Quadrat Counts, hard disk, analýza tenkých vrstiev

Title: Application of evolutionary proramming in physics
Author: Pavol Jusko
Department: Department of surface and plasma science
Supervisor: Prof. RNDr. Rudolf Hrach, DrSc.
Supervisor's e-mail address: Rudolf.Hrach@mff.cuni.cz

Abstract: In the present work we study a possibility of using evolutionary programming in physics. We have chosen a travelling salesman problem as a problem, which can be on one hand applied in physics and on the other hand is studied extensively in computer science. The solution of the problem of the shortest flowline connecting each point in a set and passing through each point exactly once, would then be used as a layout classification of these points. We use Quadrat Counts method as a refference. The shortest flowline method is being also compared to the difussion zone used in a hard disk method the process of structure generation. We pay attention to algorithm optimalisation as well.

Keywords: Evolution, the shortest flowline, Quadrat Counts, hard disk, film structure analysis

Kapitola 1

Úvod

1.1 Prehľad kapitol

Stručná charakteristika jednotlivých kapitol:

V prvej kapitole načrtneme problematiku evolučných a genetických algoritmov ako aj možnosti ich využitia v rôznych problémoch nie nutne súvisiacich s fyzikou. Ďalej sa stručne venujeme problematike analýzy tenkých vrstiev, na ktorú sa snažíme v Kap. 4. aplikovať evolučný algoritmus.

Druhá kapitola pojednáva o filozofii evolučného algoritmu ako aj o jeho prednostiach a nevýhodách voči klasickým prístupom. Na jednoduchých príkladoch ilustrujeme pojmy používané v genetických algoritmoch, ako aj základné postupy aplikovateľné na širšie spektrum problémov.

Nasledujúca tretia kapitola pojednáva o probléme hľadania najkratšej spojnice bodov. Uvádzame možné prístupy k riešeniu tejto úlohy. Analyzujeme jednotlivé parametre evolučného algoritmu, aby sme boli schopní dosiahnuť optimálnu rýchlosť ako aj kvalitu riešenia. V závere kapitoly zhŕňame v jednoduchom teste, ktorá architektúra procesorov si s týmto algoritmom poradí najlepšie.

V štvrtej kapitole sa venujeme analýze tenkých vrstiev a to metódou QC. Venujeme pozornosť aj metóde Hard disk, ktorú ako jedinú využívame pri generovaní štruktúr. Aplikujeme dĺžku najkratšej krivky ako ďalší možný štruktúrny parameter.

V piatej kapitole posudzujeme, či je metóda najkratšej spojnice metódou vhodnou na popis štruktúry povrchových útvarov (načrtneme aj možnosť získania príznaku), ako aj miesto evolučných algoritmov vo fyzikálnych aplikáciách.

1.2 Algoritmus inšpirovaný prírodou

V šesťdesiatych rokoch minulého storočia sa intenzívne začalo so skúmaním živých bytostí za účelom napodobniť ich vývoj v algoritmoch. Pri napodobňvaní

prírodných postupov sa využili najmä teórie Ch. R. Darwina a J. B. Lamarka. Algoritmy ktoré vzišli z tohto úsilia nachádzajú dnes uplatnenie najmä v optimalizačných úlohách, kde nevieme nájsť najlepšie riešenie (prípadne jeho nájdenie je časovo náročné), ale vystačíme si s riešením, ktoré sa k najlepšiemu približuje. Príkladom môžu byť úlohy rozdelenia práce určitému počtu strojov, navigácia robota v priestore s prekážkami, hľadanie extrémov funkcií viacerých premenných atď.

Jednoduchosť týchto algoritmov tkvie aj v tom, že nepotrebujeme poznať všetky aspekty problému, stačí ak vieme porovnať dve možné riešenia a rozhodnúť, ktoré vyhovuje viac. Avšak nutno dodať, že ide o stochastické (pravdepodobnostné) algoritmy, to znamená, že výsledné riešenia získané takýmto algoritmom nemusia byť rovnaké. Výrazným rozdielom oproti štandardným algoritmom je aj to, že nepracujeme s jedným riešením, ktoré postupne zlepšujeme. Miesto toho používame celú populáciu, kde vznikajú lepšie ale aj horšie riešenia.

Označenie jednotlivých typov algoritmov podľa spôsobu vytvorenia novej generácie:

- **Genetický algoritmus** využíva kríženie aj mutácie.
- **Evolučný algoritmus** (alebo evolučné programovanie) využíva len mutácie prípadne heuristiky.

Toto značenie však nie je jednotné a medzi jednotlivými autormi sa líši, čo do počtu kategórií aj jednotlivých názvov [2], [7].

1.3 Analýza obrazu

Pri spracovaní obrazu (napr. u tenkých vrstiev) sa uplatňujú metódy založené na teórii matematickej morfológie (napr. radiálna distribučná funkcia) ako aj metódy založené na integrálnych transformáciach (napr. waveletové funkce). Analýza obrazu vyššej úrovne popisuje danú štruktúru výstupným údajom, ktorým je príznak. Príznak môžeme z danej morofologickej metódy získať rôznymi spôsobmi:

- výstup metódy je priamo príznakom (napr. číslo u metódy Quadrat Counts QC)
- výstupom metódy je napr. funkcia, u ktorej následne volíme príznak ako nejakú jej špecifickú vlatnosť (napr. radiálna distribučná funkcia, príznakom môže byť polomer v ktorom sa dostávame do polovice výšky prvého maxima)

K danej štruktúre vieme teda jednoznačne priradiť príznaky, ktorými ju budeme charakterizovať [8]. Metódy obrazovej analýzy sa dajú rozlíšiť aj podľa toho či pracujú s bodovými objektmi (prípadne iba ich stredmi) alebo len s plošnými objektmi. Nami používaný QC patrí do prvej skupiny.

Kapitola 2

Evolučný a genetický algoritmus

2.1 Princípy a názvoslovie

Evolúcia v živočíšnej ríši funguje nasledovným spôsobom. Ak máme určitú populáciu jedincov, lepší jedinci v nej majú väčšiu pravdepodobnosť, že sa dožijú reprodukcie, teda vytvorenia novej populácie. Toto je jediný prípad, kedy môžu aspoň časť informcie, ktorú nesú, podať ďalej. Tento postup sa aj so spontánnou mutáciou opakuje milióny generácií. Genetický algoritmus pracuje na množine prvkov, ktoré nazývame jedinci (angl. chromosome). Samotná množina sa potom nazýva populácia (angl. population). Jedinec je prvkom z priestoru všetkých možných riešení problému. Každému jedincovi musíme vedieť priradiť hodnotenie (angl. fitness), t.j. mieru, akou vyhovuje požiadavkom [2]. Následne môže dôjsť za pomoci mutácie a kríženia k vytvoreniu novej populácie. Postupnosť populácií nazývame generácie. V ďalšom texte je gén synonymom pre slovo jedinec. Avšak v programe je pod pojmom jedinec chápaná zlozitejšia štruktúra obsahujúca aj samotný gén. Pojmom chromozóm chápeme časť informácie, ktorú nesie jedinec. Schéma jednoduchého genetického algoritmu:

1. Vytvorenie populácie náhodne vytvorenými jedincami.
2. Ohodnotenie každého jedinca.
3. Vytvorenie novej populácie (tento krok opakujeme, kým nová populácia nemá požadovanú veľkosť):
   Výber jedinca zo starej populácie.
   - Možné kríženie s iným vybraným jedincom.
   - Možná mutácia jedinca.

   Vloženie jedinca do novej populácie.
4. Je splnená konečná podmienka?
   – Nová populácia sa stáva starou a vraciame sa do bodu 2.
   \+ Máme požadované riešenie.

V nasledujúcich podkalpitolách podrobnejšie preskúmame jednotlivé časti algoritmu.

2.2 Zakódovanie a hodnotenie

Zakódovaním nazývame konkrétnu dátovú štruktúru, ktorá úplne popisuje vlastnosti jedinca. Niektoré používané typy kódovania:

- **binárne** každý bit génu reprezentuje jeden chromozóm.
- **floatové** číslo reprezentuje jeden chromozóm, napr. súradnica.
- **Grayovo** reprezentácia číselnej hodnoty, jeho výhoda je v blízkosti susedných čísel.

Často môže byť gén reprezentovaný kombináciou uvedených kódovaní. V konkrétnych aplikáciách sa však používajú kódovania najlepšie vyhovujúce danému problému, v iných prípadoch sú nevhodné.

Hodnotiaca funkcia (fitness, f) je funkcia, ktorá určuje ako dobre splňuje jedinec požiadavky naň kladené. Ak X je množina všetkých možných jedincov, požadujeme aby mala nasledujúce vlastnosti:

- $\forall x \in X, \exists! f(x)$
- $\forall a, b \in X, f(a) < f(b) \Leftrightarrow a$ je považovaný za horšieho jedinca ako b
- prípadne môžme požadovať normalizovanie f

Ako jednoduchý príklad môžme uvažovať sadu čísel (každé číslo je jedinec, a zároveň gén, ktorý má jediný chromozóm $\Rightarrow$ napr. floatové kódovanie). Budeme sa snažiť nájsť maximum funkcie $f(x), x \in \mathbb{R}$. Keďže funkcia f spĺňa ňaše požiadavky, máme zvolené zakódovanie aj hodnotiacu funkciu.

2.3 Výber

Výber jedinca, ktorý vstúpi do ďalšej populácie a bude tak mať možnosť ovplyvňovať ďalšie dianie je najdôležitejšou časťou evolučného algoritmu (analógiou v živočíšnej ríši je prežitie jedinca do reprodukcie). Ak ho nezvolíme správne, môžeme neúmerne zvýhodňovať nadaných jednicov, a takto degenerovať novú populáciu alebo naopak spomaliť konvergenciu. Na rozdiel od živej prírody má ale každý jedinec nenulovú pravdepodobnosť výberu.
Niektoré používané typy výberu:

- **Ruleta** "spravodlivý" spôsob výberu, čo sa týka fitness. Pravdepodobnosť, že jedinec bude vybraný je úmerná jeho fitness. Jednoducho implementovateľný algoritmus, avšak je nevyhovujúci pre populácie s rádovými odchylkami fitness.
- **Rank** "spravodlivý", čo sa týka poradia. Pravdepodobnosť, že jedinec bude vybraný je úmerná jeho poradiu (najlepší je najvyššie v poradí, má teda najväčšiu pravdepodobnosť výberu). Nediskriminuje málo nadaných, ale naopak v populácii s rovnomerne rozloženou fitness uprednosťuje vyvinutejších.

- **Turnaj** "spravodlivý", čo sa týka "sily". Z populácie je náhodne vybraná malá skupinka jedincov, najlepší (môže ich byť aj viac) postupujú ďalej. Tento spôsob výberu je prirodzene elitistický.

- **Elitizmus** najlepší jedinec (prípadne niekoľko najlepších jedincov) postupuje do novej generácie priamo. Takto zachováme najlepšie riešenie.

Príklad rulety a rank výberu uvádza Tab.2.2, kde je uvedený príklad pre populáciu z Tab. 2.1. Zvolíme si náhodne číslo $x, x \in < 1, K >$. Toto číslo je indexom do tabuľky rulety, kde jednoznačne určuje jedného (vybraného) jedinca. Najjednoduchší príklad rank výberu je podobný rulete, s výnimkou toho, že pole naplníme poradovým číslom, ktoré prislúcha danému jedincovi v poradí od najhoršieho po najlepšieho. Konkrétne pravdepodobnosti výberu z rulety a ranku sú uvedené v Tab. 2.3.

Tabuľka 2.1: Príklad populácie

Jedinec	1	2	3	4
Fitness	4	1	9	6

Tabuľka 2.2: Príklad rulety a ranku

Index

1 2 3 4 5 6 7 8 9 A B C D E F G H I J K

Ruleta

1	2	3	4

Rank

1	2	3	4

Tabuľka 2.3: Fitness a pravdepodobnosť výberu v rulete a ranku

Jedinec		1	2	3	4
Fitness	[%]	20	5	45	30
Pravdepodobnosť výberu v rulete	[%]	20	5	45	30
Pravdepodobnosť výberu v ranku	[%]	20	10	40	30

Teda ak pre našu modelovú populáciu "padnú" čísla 6, C, J, 4 vyberieme postupne týchto jedincov:

- z rulety 3, 3, 4, 1

- z ranku 2, 3, 4, 1

Tabuľka 2.4: Kríženie v binárnom kódovaní

Rodič 1.	**0**	**1**	**1**	**0**	**0**	0	1	1	1	0	1	0
Rodič 2.	0	1	0	1	1	**1**	**0**	**0**	**1**	**1**	**0**	**1**
	1.					2.						
Potomok	0	1	1	0	0	1	0	0	1	1	0	1

Tabuľka 2.5: Mutácia v binárnom kódovaní

Rodič	0	1	1	0	0	0	1	1	**1**	0	1	0
Potomok	0	1	1	0	0	0	1	1	**0**	0	1	0

2.4 Kríženie a mutácia

Kríženie a mutácia sú operácie, ktoré menia jedincov pri vytváraní novej populácie. Ich účelom je:

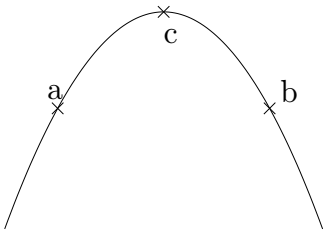
- rozšírenie prehľadávaného priestoru.
- nájdenie lepšieho jedinca, ako bol pôvodný.

Pod pojmom prehľadávaný priestor je myslená tá časť priestoru všetkých možných riešení, ktorá je obsiahnutá v danej generácii.

Vysvetlíme si operátor kríženia na jedincovi s floatovým zakódovaním a ako hodnotiacu funkciu budeme používať $\forall x \in \mathbb{R}, f(x) = -x^2$. Uvažujme dvoch jedincov, $a = -2, b = 2$ (potomka budeme nazývať c). Ak zvolíme aritmetický priemer za operátor kríženia potom:

$$c = a \oplus b = \frac{a+b}{2} = 0$$

Z dvoch jedincov sa nám takto podarilo získať jedného s lepšou fitness, ako boli predchádzajúci dvaja (viď Obr. 2.1). Príklad mutácie v binárnom zakódovaní je uvedený v Tab. 2.5



Obr. 2.1: Mutácia v $\mathbb{R}$

Operátor mutácie má názorný význam ak v populácii vznikne nový jedinec, ktorý nesie informáciu, ktorú nemal žiadny z predchádzajúcich členov populácie. Takýto jedinec sa môže vyskytnúť v ešte neprebádanej oblasti prehľadávaného priestoru, a takto nám zabrániť v konvergencii do lokálnych maxím f.

2.5 Ukončenie algoritmu

V prípade, že poznáme aspoň kvantitatívne riešenie úlohy (teda najlepšiu možnú hodnotu fitness), môžeme ako ukončovaciu podmienku použiť dosiahnutie tejto hodnoty.

Avšak vo väčšine prípadov takéto šťastie nemáme. Musíme sa uspokojiť s podmienkou typu:

- za definovaný čas t_{end} nedôjde k zvýšeniu fitness.
- konštantný čas výpočtu t_{max}.
- zmena fitness po určitom počte generácii klesne pod definovanú hodnotu Δf_{min}.

V prvých dvoch podmienkach nemusíme uvažovať čas t, môžeme napríklad zvoliť počet generácii. Podmienky môžeme taktiež kombinovať.

Kapitola 3

Hľadanie najkratšej cesty

3.1 Odveký problém

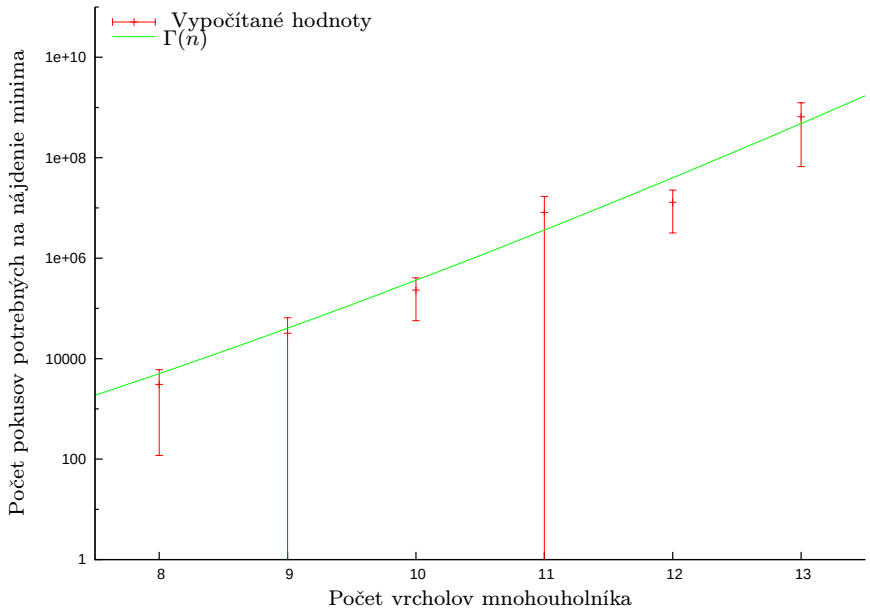
Tento problém sa často nazýva aj problémom obchodného cestujúceho (angl. TSP – travelling salesman problem). Obchodník musí navštíviť všetky miesta práve raz (a vrátiť sa na začiatok cesty), ale chcel by zároveň uraziť čo možno najkratšiu cestu. Pri malom počte miest, môže skúsiť vyrátať najkratšiu spojnicu, tak že skúsi všetky možné kombinácie. Počet možných kombinácií sa však s každým ďalším mestom faktoriálne zvyšuje, a teda tadiaľ cesta nevedie. Iným prístupom môže byť metóda Monte Carlo. Avšak podľa Obr. 3.1 si ani tu veľmi nepomôžeme, zelenou farbou je uvedená vhodne škálovaná gamma funkcia, aby sme získali prehľad o zložitosti. Veľký rozptyl nameraných hodnôt bol zapríčinený samotnou podstatou algoritmov pracujúcich s náhodnými faktormi, kde môže dôjsť k nájdeniu optimálneho riešenia s nenulovou pravdepodobnosťou v prvom kroku, ako aj za extrémne dlhý čas. Možnosťou ostávajú aproximačné algoritmy ako napr. algoritmus založený na trojuholníkovej nerovnosti v [1], prípadne algoritmy postavené na neurónových sieťach študované v [6]. Dĺžka je vo všetkých prípadoch uvádzaná bezrozmerne, avšak keďže sme pracovali s diskrétnou podložkou, je touto bezrozmernou veličinou myslená najkratšia vzdialenosť dvoch susedných miest podložky, nazývaná pixel.

3.2 Zakódovanie, fitness, výber

Medzi najčastejšie používané kódovania v kombinatorických problémoch patria kódovania náhodným kľúčom a permutačné kódovanie. Ich aplikácia na problém najkratšej spojnice:

- jedinečný **náhodný kľúč** z intervalu $(0, 1)$ priradíme každému bodu a ako gén používame postupnosť týchto čísel:

N	:	1	2	3	4	5	6
gén	:	0.14	0.98	0.33	0.72	0.26	0.51
krivka	:	1	5	3	6	4	2



Obr. 3.1: Rýchlosť konvergencie metódy Monte Carlo na mnohouholníku

Poradové číslo v zozname určuje o ktorý bod sa jedná. Po zotriedení zoznamu dostávame poradie podľa ktorého môžme zostrojiť krivku [7].

- **permutačné kódovanie** ako gén využíva priamo poradie bodov, v ktorom sú spojené pri vytvorení jednej konkrétnej krivky:
  Ide o častejšie používaný spôsob [7].

N	:	1	2	3	4	5	6
gén	:	1	5	3	6	4	2

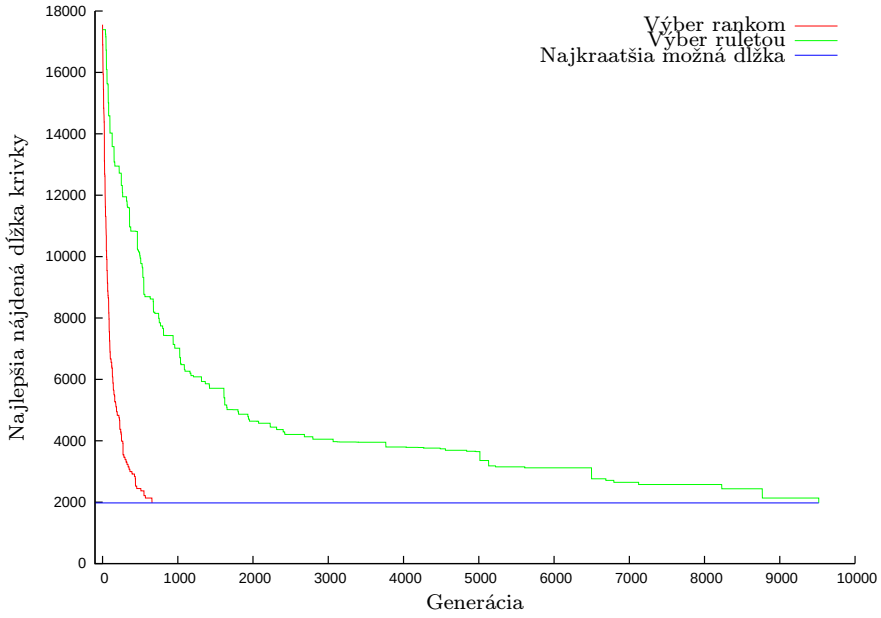
N reprezentuje poradové číslo bodu, gén reprezentuje, v akom poradí body spojíme. Krajné body génu považujeme taktiež za susedné. Najmä kôli jednoduchosti implementácie som zvolil permutačné kódovanie.

Keďže sa snažíme nájsť najkratšiu spojnicu bodov, nemôžeme ako fitness f použiť $l(\gamma)$, kde funkcia l znamená dĺžku krivky, pretože táto má so zlepšujúcim sa jedincom klesajúci charakter. Preto som použil $f = c.l^{-1}(\gamma)$, kde c je vhodne zvolená škálovacia konštanta. Nepoužíval som normovanie f, keďže použitá implementácia rulety ani rank výberu ho nevyžaduje. Pri použití ruletového výberu, som sa venoval aj fitness tvaru $f = c.l^{-x}(\gamma), x \in \{\frac{1}{2}, \frac{1}{3}, \frac{2}{3}, 2\}$, avšak k znateľnému zrýchleniu konvergencie nedošlo.

Implementoval som výber ruletou a rankom, viac sa osvedčil výber rankom (viď Obr. 3.2, kde je hľadaný pravidelný 50 uholník), preto ak nie je uvedené, o aký typ výberu sa jedná, je použitý tento typ výberu.

3.3 Mutácie a heuristika

Zvolili sme evolučný algoritmus, teda na vytvorenie nového jedinca nepoužívame kríženie. Jediný spôsob ako rozširujeme prehľadávaný priestor je mutácia. Na zlep-



Obr. 3.2: Porovnanie rýchlosti konvergencie pri výbere rankom a ruletou

šovanie jedincov používame mutáciu ako aj jednoduchú heuristiku. Ak neuvádzam pravdepodobnosť niektorej z mutácií, tak je použitá štandardná hodnota 10%, u heuristiky je to 1%. Štandardná veľkosť populácie je 100 jedincov.

Používané spôsoby mutácie:

- Výmena dvoch chromozómov - dve náhodne zvolené chromozómy si vymenia pozíciu.

Rodič	:	0	1	2	**3**	4	5	6	7	**8**	9
Potomok	:	0	1	2	**8**	4	5	6	7	**3**	9

- Presunutie - časť génu (či už konštantnej alebo náhodnej dĺžky) sa presunie na iné miesto v géne.

Rodič	:	0	1	2	3	4	5	**6**	**7**	8	9
Potomok	:	0	1	**6**	**7**	2	3	4	5	8	9

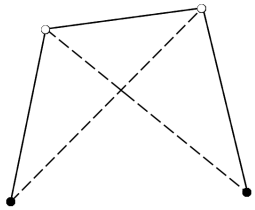
- Obrátenie poradia - zvolia sa dve náhodné chromozómy v géne a medzi nimi dôjde k obráteniu poradia chromozómov.

Rodič	:	0	1	2	**3**	**4**	**5**	**6**	7	8	9
Potomok	:	0	1	2	**6**	**5**	**4**	**3**	7	8	9

Okrem vyššie uvedených mutácii som použil aj jednoduchú heuristiku, ktorá rieši úlohu pre 4 body. Definícia pojmu heuristika je značne obtiažna, zväčša ide o empiricky zistený postup, ktorý vedie rýchlo k dosiahnutiu aspoň nejakého výsledku, ale nič nám negarantuje [9]. V géne daného jedinca si zvolí po sebe idúce

štyri body, krajné dva považuje za konce krivky, ktorú sa snaží minimalizovať. Takto ale ostávajú už len dve možnosti zostrojenia krivky, takže sme schopní okamžite rozhodnúť, ktorú variantu máme do vylepšeného génu vybrať (viď Obr. 3.3).

Pokročilejšie heuristiky sa nesústreďujú na jedno konkrétne miesto génu, ale napr. zvolia tri náhodné body v géne, ohodnotia všetky možné permutácie týchto troch bodov a najlepšiu variantu vyberú ako výsledok.



Obr. 3.3: Konštrukcia lepšieho génu heuristikou

3.4 Rýchlosť algoritmu

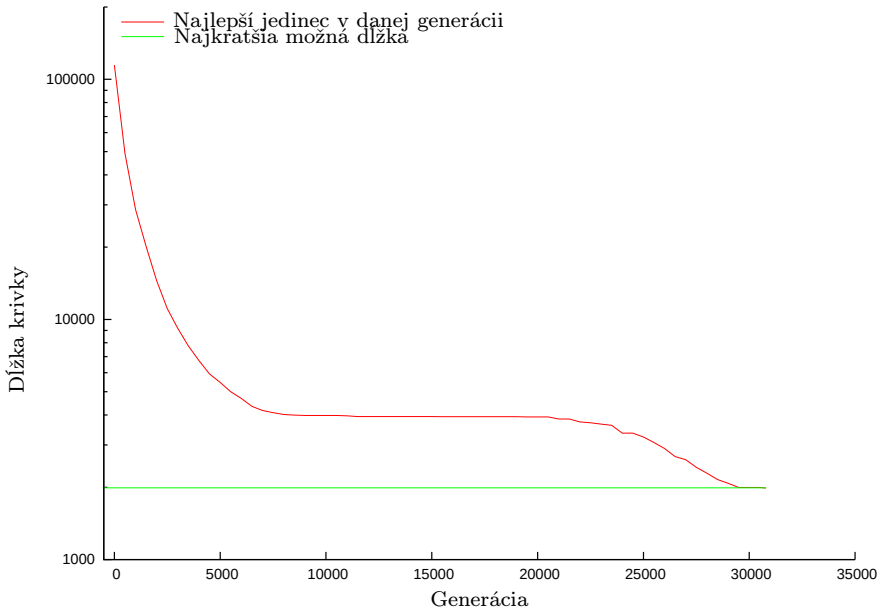
Pri pozorovaní rýchlosti algoritmu sme potrebovali štruktúru, u ktorej by sme vedeli analyticky vyrátať dĺžku najkratše krivky γ. Zvolili sme pravidelný mnohouholník.

Konkrétny príklad ako algoritmus našiel riešenie 300-uholníku je uvedený na Obr. 3.4, dlhá konštantná časť zodpovedá prípadu, keď sú vrcholy spojené nasledujúcim spôsobom: začneme vo vrchole a, v jednom smere obiehania sa spájame s každým druhým najbližším vrcholom, až kým sa nedostaneme k najbližšiemu susedovi a z opačnej strany, ako sme vyrazili. Tu zmeníme smer obiehania a vrátime sa do bodu a obdobným postupom. Takáto a podobné konfigurácie zodpovedajú zhruba dvojnásobku najlepšieho riešenia, a predstavujú lokálne minimum, v ktorom sa algoritmus zdrží.

Ako sa zvyšuje náročnosť algoritmu s pribúdajúcim počtom vrcholov, môžeme vidieť v Obr. 3.6. Je v ňom zobrazená aj vhodne škálovaná exponenciála, takže sa dá predpokladať, že takýto algoritmus má exponencionálnu asymptotickú zložitosť. Veľký rozptyl nameraných hodnôt, zapríčinený typom konečnej podmienky a nedeterminističnosťou algoritmu, je dobre viditeľný taktiež u metódy Monte Carlo. Konkrétne pri pozorovaní rýchlosti konvergencie u 500-uholníka dochádzalo k odchylkám v počte generácií väčším ako priemer tejto hodnoty. Grafický prehľad o konvergencii dáva Obr. 3.7. Dĺžky najlepších jedincov v danej generácii sú uvedené v Tab. 3.1.

Tabuľka 3.1: Dĺžky γ z Obr. 3.7

Generácia	0	1000	3000	najkratšia spojnica
Dĺžka γ	52216	10299	3287	2826



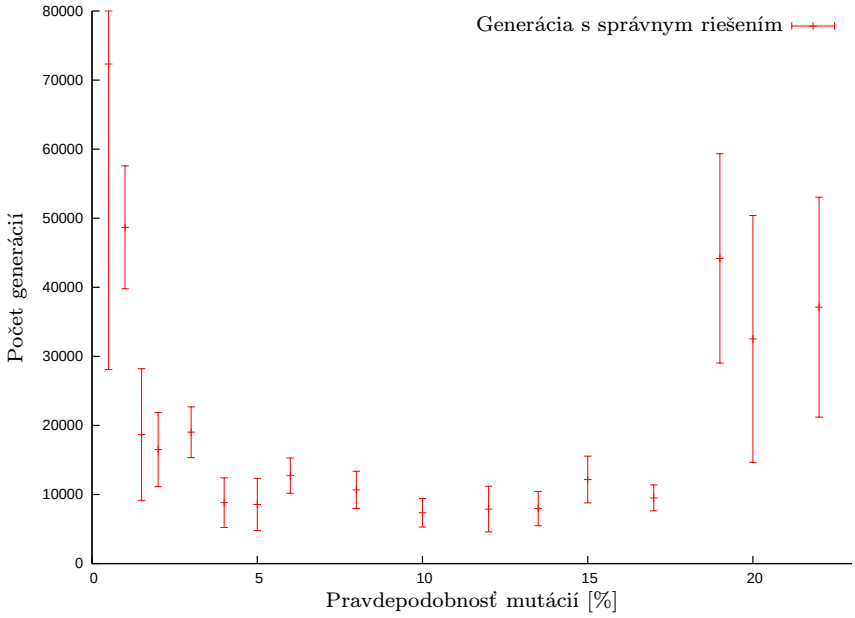
Obr. 3.4: Konvergencia algoritmu na pravidelnom 300-uholníku

Použitie elitizmu malo výrazný vplyv, čo sa týka kvality výsledku. Bez elitizmu začala dĺžka najlepšieho jedinca oscilovať na hodnotách $\approx$ 1.5-násobku najlepšieho riešenia a algoritmus takmer prestal konvergovať. Vhodný počet elitistických jedincov som neskúmal, ale najmä kvôli vyššie uvedenému poznatku som používal elitizmus v rozsahu 2.5% z veľkosti populácie.

Použitie danej heuristiky s ľubovoľnou pravdepodobnosťou nemalo pozorovateľný vplyv na rýchlosť konvergencie. Tento stav je zrejme spôsobený jednoúčelovosťou danej heuristiky, ktorá sa už po niekoľkých desiatkach generácií nemá kde uplatniť. Keďže som sa zameriaval najmä na evolučný algoritmus, nevenoval som ďalším typom heuristík pozornosť.

Na posudzovanie vplyvu pravdepodobnosti mutácie som použil pravidelný 200-uholník a populáciu o veľkosti 200 jedincov. Pri pravdepodobnosti mutácií (všetky mutácie mali rovnakú pravdepodobnosť) pod 0.1% algoritmus prakticky prestal fungovať. Na Obr. 3.5 je graficky znázornený počet generácií potrebných na nájdenie správneho riešenia pri rôznych pravdepodobnostiach mutácie. Charakter tejto závislosti bol až na škálovanie obdobný aj pre iné počty bodov. Ako štandardnú pravdepodobnosť mutácií som zvolil hodnotu 10%, v ktorej daná závislosť dosahovala svojho minima. Z Obr. 3.5 je však patrné, že voľba tohto parametru nie je kritická a rovnaká rýchlosť by sa dala očakávať s pravdepodobnosťou mutácie v rozmedzí $5 - 15\%$.

Aby sme získali rámcový prehľad, ako dlho môže jeden výpočet reálne trvať na súčasnom hardware uvádzam v Tab. 3.2 časy potrebné na prejdenie 10^4 generáciami. Každá populácia bola zložená z 200 jedincov. Časy sú uvedené pre neparalelizovaný program, takže pri meraní využíval len jedno jadro. Získanie ďalej analyzovaných dát na počítači s procesormi Opteron 880 a využití 4 jadier trvalo zhruba 48 hodín.



Obr. 3.5: Rýchlosť algoritmu v pravidelnom 200-uholníku v závislosti na pravdepodobnosti mutácie

Tabuľka 3.2: Čas potrebný na prejdenie 10^4 generáciami na rôznych procesoroch s využitím len jedného vlákna aj u viacjadrových procesorov

procesor	f [GHz]	cache [Mb]	t [s]	δt [s]
Intel Pentium M	1.3	1	20.1	0.2
Intel Pentium 4	3.2	0.5	15.3	0.3
AMD Opteron 880	2.4	1	11.8	0.2
Intel Core2 Duo	2.0	4	8.5	0.3
Intel Core2 Duo	2.4	4	8.4	0.2

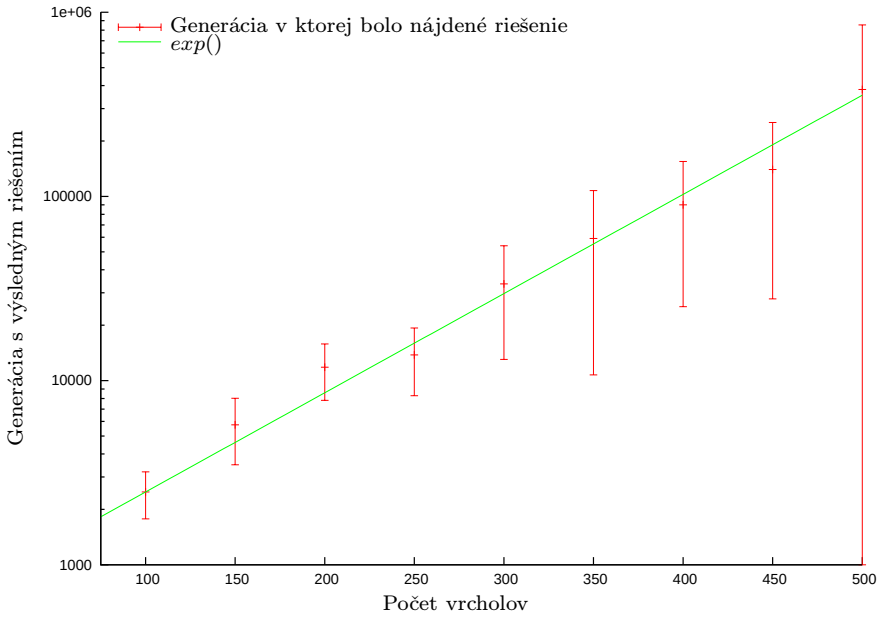
3.5 Implementácia

Celý algoritmus je napísaný v jazyku C. Na paralelizáciu je využívana knižnica pthread, grafický výstup bol vytvorený pomocou knižnice gtk-2.

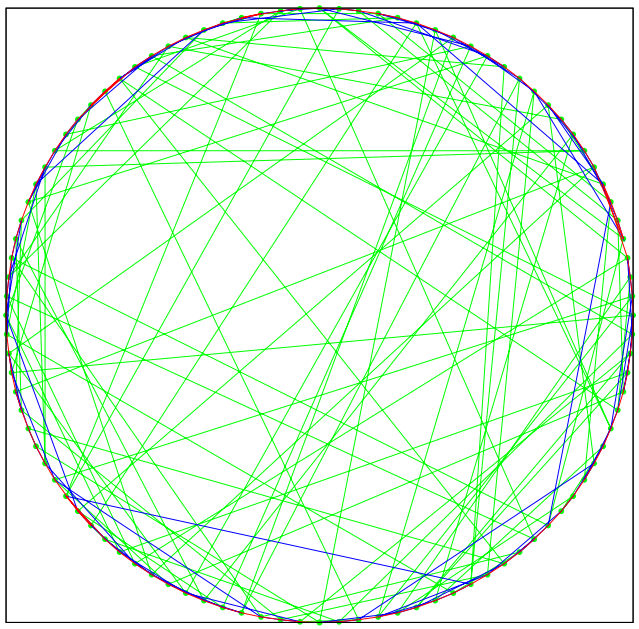
Niektoré detaily implementácie:

- Vzdialenosti medzi jednotlivými bodmi sú vyrátané len raz. Ak ich potrebujeme, máme ich uložené v pamäti.
- Ruletový výber bol implementovaný rovnakým spôsobom ako je uvedené v kap. 2.2.
- Rank výber je založený na znalosti získať z poradového čísla v ranku pravdepodobnosť výberu. Ak a-tý jedinec má pravdepodobnosť výberu $a/\sum_{k=1}^{n} k$, vieme túto formulu invertovať a z náhodného čísla $x, x \in < 0, \sum_{k=1}^{n} k >$ takto získať poradové číslo i zvoleného jedinca:

$$i = \left[-\frac{1}{2} + \sqrt{\frac{1}{4} + 2x} \right]$$



Obr. 3.6: Počet generácii potrebných na nájdenie správneho riešenia v pravidelnom mnohouholníku



Obr. 3.7: Riešenia v 100-uholniku po 0 (znáz. zelene), 1000 (znáz. modro), 3000 (znáz. červeno) generáciách

- Paralelizácia sa odohráva na úrovni delenia práce riadiacim vláknom, ktoré môže výpočetnému vláknu/am prideliť úlohu vo forme "priraď fitness danej časti populácie" alebo "vytvor danú časť novej populácie".
- Jednotlivé typy mutácie sú vložené do vlastnej rulety (ruleta je normovaná, takže súčet všetkých pravdepodobností nesmie presiahnuť maximálnu pravdepodobnosť), do ktorej vhadzujeme náhodné čísla. Ak nie vybraná žiadna z mutácií, s jedincom sa nedeje nič a postupuje priamo do novej poulácie.
- Výber rôznych generátorov náhodných čísel nemal vplyv na kvalitu výsledkov.

Kapitola 4

Použitie vo fyzike

4.1 Usporiadanie povrchu

Ako metódu analýzy povrchu som zvolil Quadrat Counts a dĺžku krivky som porovnával aj s difúznou zónou.

- Quadrat Counts (QC) študuje počet objektov nájdených v štvorcoch (môžu sa použiť i iné útvary) náhodne umiestňovaných na povrchu. Táto metóda je sama sebe príznakom, výstupom je len jedno číslo a to podiel rozptylu ($\sigma^2(x)$) a priemeru ($< x >$) počtu útvarov x nájdenych v štvorci.

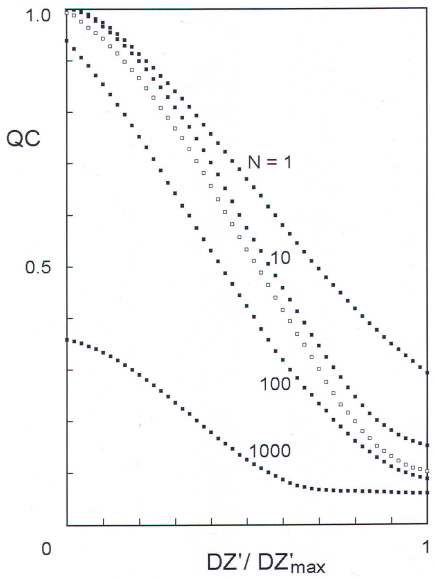
$$QC = \frac{\sigma^2(x)}{< x >}$$

  Pre dostatočný počet bodov platí, že ak hodnota $QC = 0$, štruktúra je úplne usporiadaná, prípad $QC \approx 1$ zodpovedá štruktúre úplne náhodnej. V prípade menšieho počtu bodov však ide o rozdelenie binomické a dostávame aj hodnoty väčšie ako 1. Používali sme štvorec o strane 300 pixelov, čo zodpovedá $x \approx 45$ bodov vo štvorci. Štvorec sme umiestňovali 10^6 krát. Kalibračná krivka QC je uvedená na Obr. 4.1 (Naším parametrom metódy zodpovedajú biele štvorčeky). Metóda QC je podrobne študovaná v [8]. Jeden z prezentovaných záverov hovorí, že s rastúcim usporiadaním štruktúry klesá rozptyl, druhý, že chyba tejto metódy taktiež rastie s veľkosťou testovacieho štvorca.

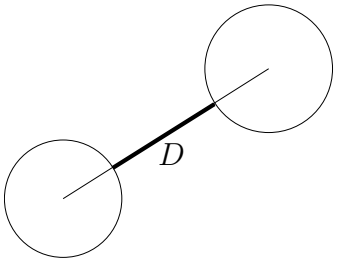
- Difúzna zóna D je minimálna vzdialenosť najbližšie susediaceho bodu v štruktúre. Je vstupným parametrom metódy hard disk, ktorú som použil pri generovaní štruktúr. Nevýhodou tejto metódy je, že nie je schopná generovať štruktúry vysokého stupňa usporiadania ($QC < 0.1$). S rastúcim D rastie stupeň usporiadania. Bližšie sa metóde hard disk venuje [3],[4]. Pri generovaní štruktúr ešte vyššieho stupňa usporiadania by sa dal s výhodou použiť soft disk model [3].

4.2 Najkratšia spojnica a QC v závislosti na D

Štúdium týchto dvoch metód bolo vykonané na 385 štruktúrach (pre každú zvolenú D 20 štruktúr, $D = 37$ len 5 štruktúr). Veľkosť podložky bola 1000x1000.



Obr. 4.1: Kalibračná krivka QC [5]



Obr. 4.2: Definícia difúznej zóny

Bol použitý výber rankom, parametre evol. algoritmu boli nasledujúce:

Tabuľka 4.1: Parametre evolučného algoritmu

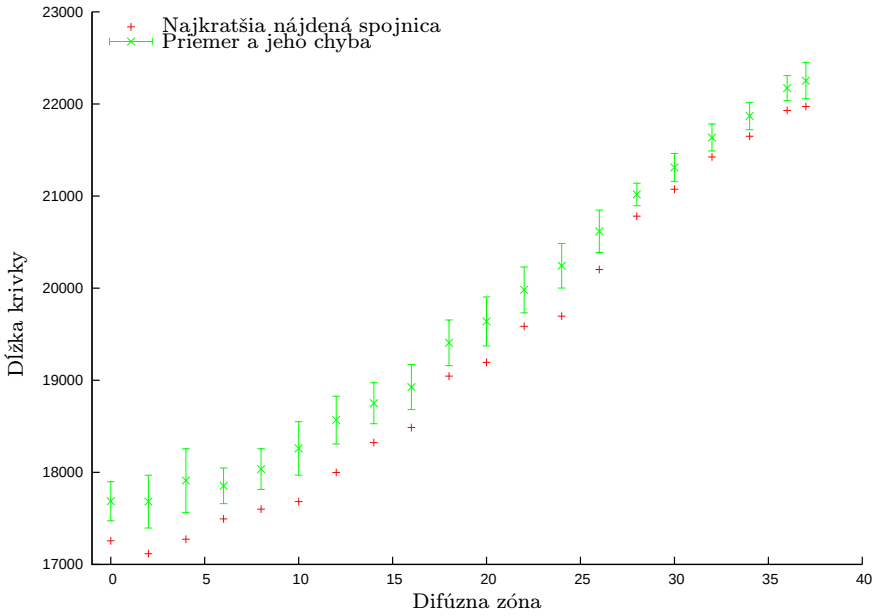
Veľkosť populácie	Prav. mutácie [%]	Prav. heuristiky [%]	Konečná podmienka
200	10	1	120 s bez lepšieho riešenia

Výsledné závislosti dĺžky najkratšej krivky a QC sú uvedené v Obr. 4.3 a Obr. 4.4. Štruktúry s $D < 37$ sa vždy podarilo vytvoriť za menej ako 60 s. U štruktúry s $D = 37$ bol maximálny čas vytvárania štruktúry stanovený na 180 s, no aj s touto podmienkou sa podarilo vytvoriť len 6 štruktúr z 20 pokusov, ktoré obsahovali 500 bodov.

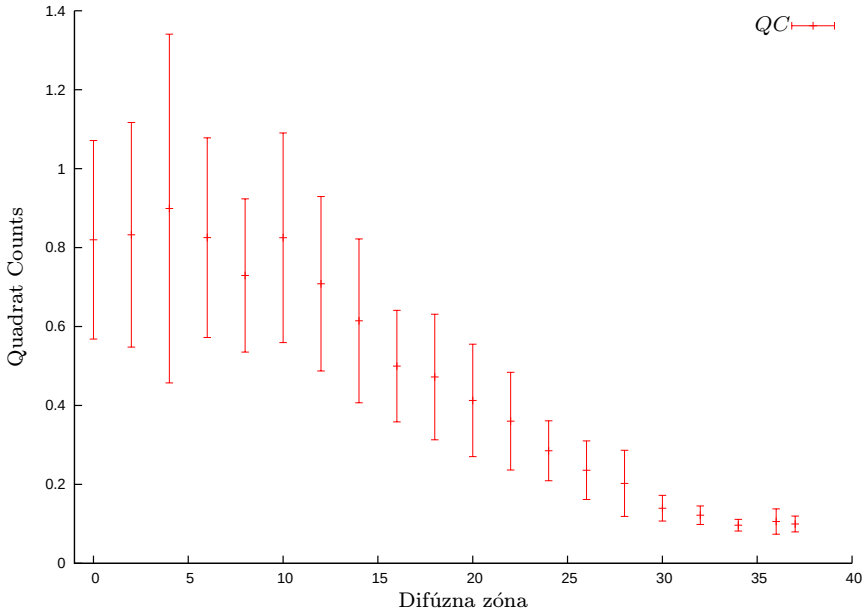
Príklad jedného z možných riešení uvádzam pre tri rôzne D v Obr. 4.5 - 4.7. Pre zaujímavosť je na Obr. 4.8 uvedený príklad štruktúry veľkosti 1000 bodov.

4.3 Nedeterministickosť evolučného algoritmu

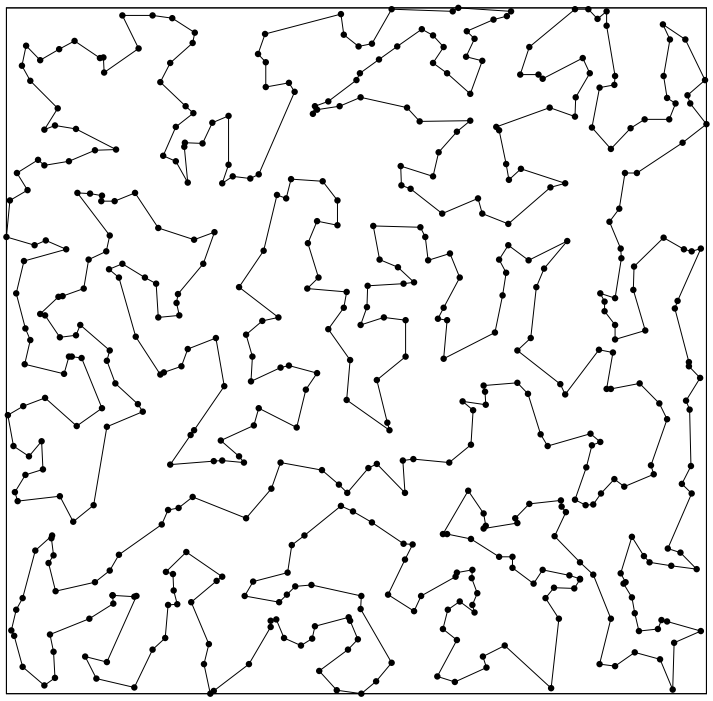
Nevýhodou evolučného algoritmu je jeho nedeterministickosť. Pri jednoduchých mnohouholníkových štruktúrach algoritmus vždy našiel správne riešenie. Avšak



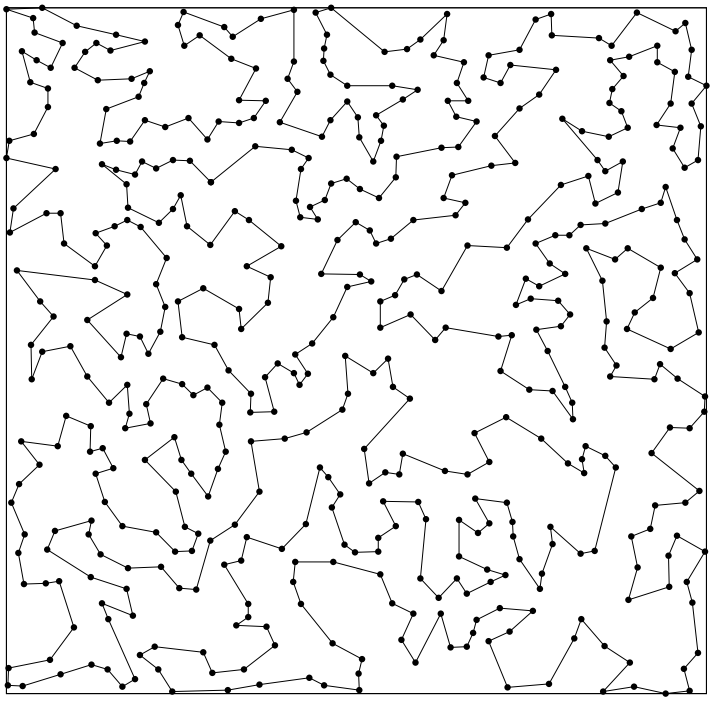
Obr. 4.3: Závislosť dĺžky najkratšej spojnice na usporiadaní objektov



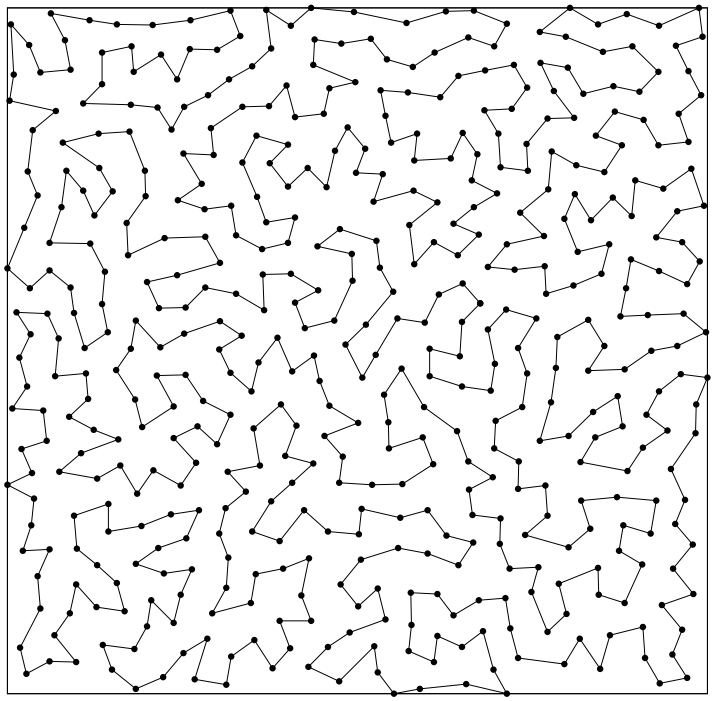
Obr. 4.4: Závislosť QC na D



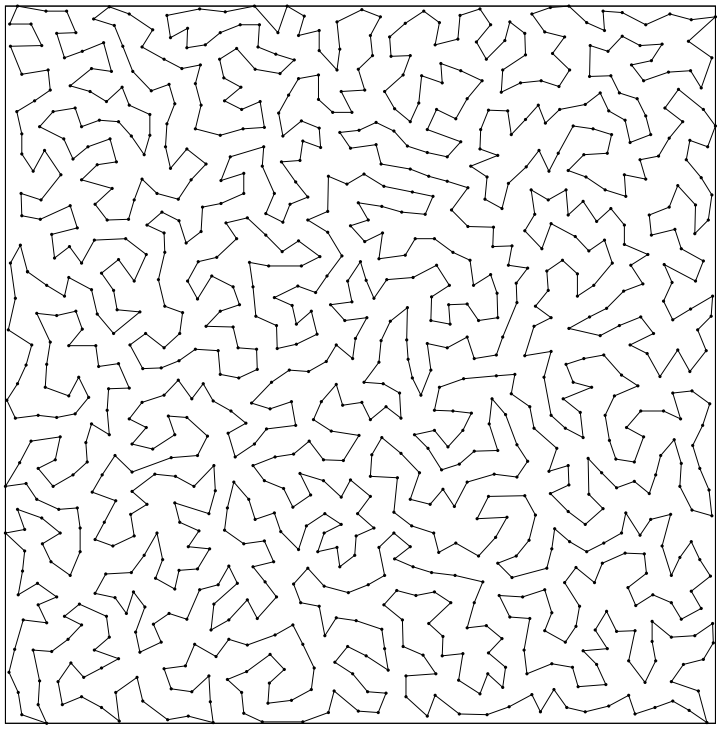
Obr. 4.5: Výsledné riešenie pre štruktúru s D = 0



Obr. 4.6: Výsledné riešenie pre štruktúru s D = 18

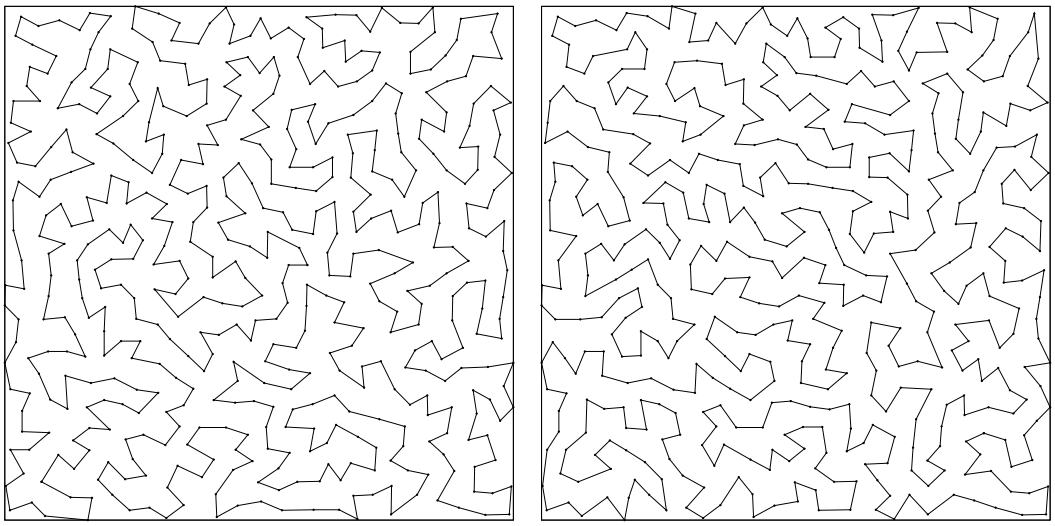


Obr. 4.7: Výsledné riešenie pre štruktúru s D = 37



Obr. 4.8: Výsledné riešenie pre 1000 bodov D = 26

pri komplikovaných štruktúrach o 500 bodoch takýto prípad nenastal. Jeden takýto rozdielny výsledok je možné vidieť na Obr. 4.9.



(a) $\gamma = 22178$ (b) $\gamma = 22058$

Obr. 4.9: Dve rôzne riešenia toho istého problému

V takomto prípade ale nastáva otázka s ako presnosťou sme schopný nájsť "správne" riešenie. Za týmto účelom bol na štruktúre s $D = 1$, a počtom objektov 500 (Rozmery 700x700 pixelov) algoritmus spustený 20 krát s koncovou podmienkou 120 s bez lepšieho jedinca a 20 krát s podmienkou 210 s bez lepšieho jedinca. Zistené údaje sú uvedené v Tab. 4.2. Paradoxne ako najlepší tak aj najhorší výsledok pochádzali z testu s konečnou podmienkou 120 s.

Tabuľka 4.2: Nedeterministickosť algoritmu

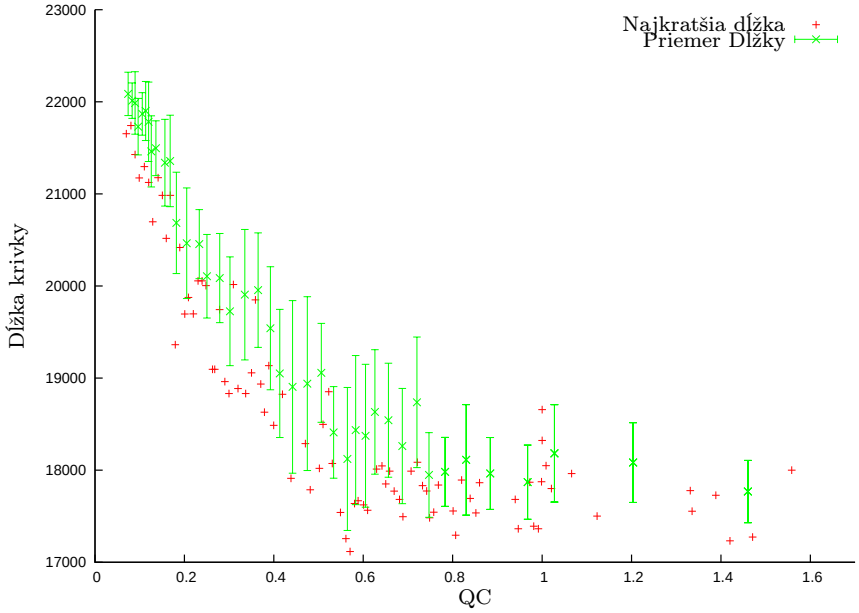
$min(\gamma)$	$max(\gamma)$	$< \gamma >$	$\Delta\gamma_{min,max}$ [%]	$\Delta\gamma_{min,<\gamma>}$ [%]
12006	12367	12230	3.0	1.8

Kapitola 5

Záver

5.1 Porovnanie metód QC a dĺžka najkratšej spojnice

Porovnanie výsledkov získaných pomocou metódy najkratšej spojnice a QC je možné vidieť na Obr. 5.1. Veľká chyba je zapríčinená chybou QC pri danej veľkosti testovacieho štvorca. Závislosť QC na D je uvedená na Obr. 4.4. Obdobná závislosť u dĺžky najkratšej spojnice je na Obr. 4.3.



Obr. 5.1: Vzťah medzi QC a dĺžkou najkratšej spojnice

5.2 Možnosti metódy najkratšej spojnice

Aby sa z tejto metódy mohol stať príznak popisujúci morfológiu museli by sme ju spraviť invariantnou voči veľkosti obrazu a počtu objektov v ňom. Prvý nedostatok sa dá vyriešiť vydelením dĺžky krivky napríklad uhlopriečkou obrazu.

Načrtnutie výsledkov takejto techniky je možné vidieť v Tab. 5.1. Použité boli štruktúry s $D = 1$ a koncová podmienka pre evolučný algoritmus bola 180 s bez lepšieho riešenia.

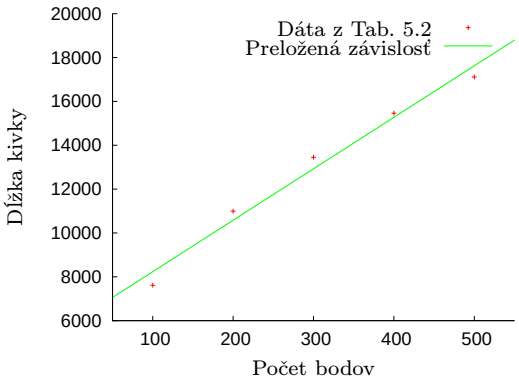
Tabuľka 5.1: Úprava dĺžky najkratšej spojnice na veľkosť podložky

Rozmer podložky	700x700	1000x1000	1200x1200
Dĺžka najkratšej krivky	12006	17116	21395
Po vydelení uhlopriečkou	12.13	12.10	12.61

Obdobné spôsob invariancie pre počet bodov je uvedený v Tab. 5.2. Konečná podmienka bola nastavená na 120 s bez lepšieho jedinca (údaj pre 500 bodov je prebratý z Tab. 5.1), u všetkých štruktúr $D = 1$. Ako vhodnejší spôsob sa ukazuje využiť všeobecnú lineárnu závislosť (Obr. 5.2).

Tabuľka 5.2: Úprava dĺžky najkratšej spojnice na počet bodov. Veľkosť podložky 1000x1000

Počet bodov	100	200	300	400	500
Dĺžka najkratšej krivky	7615	10994	13452	15462	17116
Po vydelení počtom bodov	76.15	54.97	44.84	38.66	34.232



Obr. 5.2: Náčrt zinvariantnenia metódy najkratšej krivky v záv. na počte bodov

Práca nemala za svoj cieľ dôjsť ku konkrétnemu postupu získania príznaku z metódy dĺžky najkratšej krivky, a preto som uviedol len krátky náčrt ako by sa dalo pri štúdiu tejto metódy pokračovať. Z hľadiska samotného algoritmu by bolo zaujímavé porovnať jeho rýchlosti s algoritmom využívajúcim aj kríženie, prípadne pokročilejšie spôsoby heuristík. Ďalšie úsilie by sa malo venovať aj analýze kvality výsledkov, tu ale nastáva problém s počtom štruktúr, u ktorých sme najkratšiu spojnicu schopný vytvoriť analyticky.

Nevýhodou tejto metódy však aj naďalej ostáva veľká časová náročnosť aj na súčasnom najmodernejšom hardware. Výhodu v porovnaní s metódou QC by som videl najmä u štruktúr s malým pokrytím, kde je potrebné používať u QC veľké testovacie štvorce, čo má za následok veľkú chybu metódy (viď Obr. 4.4, najmä pre malú D) [8]. U metódy najkratšej krivky bol rozptyl hodnôt v celom testovanom rozsahu D prakticky konštantný (Obr. 4.3).

5.3 Ďalšie využitie genetických algoritmov

Genetické algoritmy našli svoje uplatnenie v rôznych optimalizačných problémoch od hľadania extrémov funkcií až po odhaľovanie zneužívania kreditných kariet. Ich nespornou výhodou je ich jednoduchosť a elegancia, s akou "rieši" daný problém len za pomoci náhodných čísel. Ich hlavná nevýhoda, a to nedeterministickosť, nedovoľuje používať túto metódu exaktným oborom (napr. finančníctvo), avšak pri vhodnom aplikovaní na fyzikálne problémy, ktorých výstupy už nejakým spôsobom obsahujú určitú chybu, by som tento fakt za neprekonateľný problém nepovažoval. Ďalším príkladom, kde nedeterministickosť metódy nie je prekážkou, môžu byť simulácie technikou Monte Carlo.

Problémom ale môže byť nekorektná implementácia, kde by došlo k nájdeniu lokálnych a nie globálnych optím. Túto situáciu vo väčšine prípadov ani nie sme schopní diagnostikovať. V prípade hľadania najkratšej cesty som sa tomuto problému snažil vyhnúť tým, že som používal algoritmus nastavený tak, aby sa pri analyticky známom riešení nikdy "nepomýlil".

Aj napriek všetkým nevýhodám, ktoré genetické algoritmy majú, by som ich budúcnosť vo fyzike videl najmä pri interpretovaní a analýze veľkého množstva experimentálne získaných dát. Ich využitie môže byť úspešné aj v morfologických metódach a všade, kde je potrebná optimalizácia problému. V žiadnom prípade však netreba tieto algoritmy považovať za všeliek a nasadzovať ich bez rozmyslu.

Literatúra

[1] Cormen H. et. al.: *Introduction to Algorithms, Second Edition*, MIT Press, Cambridge, 2001 895–900.

[2] Goldberg D.: *Genetic Algorithms in Search, Optimization and Machine Learning*, Reading 1989.

[3] Haile J. M.: *Molecular dynamics simulations: Elemenatry methods*, Clemenson University, Clementon 1992 104–117.

[4] Hrach R. et. al.: *Morphological Analysis of Discontinuous and Semicontinuous Metal Films*, Thin Solid Films **317**, 1998 39–42.

[5] Hrach R.: soukromé sdělení.

[6] Malý M.: *Použití neurónových síti v analýze obrazu*, Dizertačná práca, Katedra elektroniky a vákuovej fyziky, MFF UK, Praha 2006 35–48.

[7] Mitsuo G.,Runwei Ch.: *Genetic algorithms and engineering design*, Ashikaga 1996 118–127.

[8] Šimek J.: *Moderní metódy spracování obrazu ve fyzice*, Diplomová práca, Katedra elektroniky a vákuovej fyziky, MFF UK, Praha 2002 28, 42–48, 59–60.

[9] Töpfer P.: *Algoritmi a programovací techniky*, PROMETHEUS, Praha, 1995 126.

Dodatok A

CD-ROM

Na priloženom CD-ROM disku sa nachádzajú nasledujúce dáta:

- **README** stručný návod na používanie programu evol
- **data/** obsahuje dáta získané počas výpočtu na apollo8.fzu.cz, t.j. všetky dáta použité pri spracovávaní metódy najkratšej krivky a QC.
- **evol/** obsahuje zdrojové súbory programu, použitého pri výpočtoch
- **Makefiles/** obsahuje Makefile pre rôzne typy architektúr a prekladačov

Pre samotný preklad je okrem normy POSIX vyhovujúcemu operačnému systému potrebné mať k dispozícii:

- prekladač intel c++ compiler, alebo gnu compiler collection, knižnica pthread
- v prípade grafického výstupu, knižnicu gtk+-2.8 a vyššie

Všetky výpočty sa odohrávali na rôznych distribúciách linuxu, program je však ľahko prenositeľný aj na iné operačné systémy.