Gymnázium, Praha 5, Nad Kavalírkou 1

PŘEDMATURITNÍ ROČNÍKOVÁ PRÁCE

Programování podle Darwina

Vypracoval: Daniil Barabashev

Třída: 6. B

Školní rok: 2016/2017

Předmět: ICT

Místo zpracování: Praha

Způsob ověření v praxi: Implementace do vlastních programů

Konzultant: Mgr. Pavel Petrášek

Souhlasím se zapůjčením své práce ke studijním účelům.

V Praze dne: ……

Daniil Barabashev

Čestné prohlášení

Prohlašuji, že jsem předkládanou práci vypracoval/a sám/sama pod vedením konzultanta a za použití zdrojů a literatury v ní uvedených.

V Praze dne: ……

Daniil Barabashev

Poděkování

Rád bych vyslovil své díky panu profesorovi Mgr. Pavlu Petráškovi za vedení a pomoc při zpracování práce. V neposlední řadě děkuji Václavu Volhejnovi za nápad a podporu v jeho realizaci.

Obsah

Anotace

Tato práce se bude zabývat problematikou evoluční informatiky.

V první části práce pod názvem *Úvod do problematiky genetických algoritmů* ve zkratce popisuje, co to vlastně evoluční algoritmy jsou a jejich teoretické využití.

V druhé části jsou hloubkově popsané základní principy fungování takových algoritmů. Součástí je i praktické pozorování algoritmů při řešení nejkratší cesty v prostoru (*Smart Rockets*) a porovnání výkonu takových algoritmů s algoritmy konkurenčními. K porovnání je použit známý NP-těžký (viz *Úvod do problematiky genetických algoritmů*) problém obchodního cestujícího (Travelling salesmans problem – TSP).

Ve třetí části je vysvětlena důležitost tohoto oboru informatiky, budoucnost vůči neuronovým sítím a můj osobní názor.

Klíčová slova: genetický algoritmus; evoluční postupy; heuristika;

1 Úvod do problematiky genetických algoritmů

Genetické algoritmy se zakládají na myšlence evoluce – předávání DNA a vývoje jedinců k nejideálnějšímu přizpůsobení daným podmínkám. V biologii základ této teorie přisuzujeme Charlesi Darwinovi, který v roce 1859 publikoval svoji revoluční myšlenku v knize *O vzniku druhů přirozeným výběrem čili zachováním vhodných odrůd v boji o život.* V dnešní době si už daný proces umíme představit bez problémů: evoluce je dlouhodobý a samovolný proces, kdy se jedinci mezi sebou páří a předávají své geny svým potomkům, kteří pak jsou zdatnější než jejich rodiče. Umíme však v informatice nastavit objektu DNA? Může ji pak předat další generaci? Jsme schopni vytvořit umělou evoluci?

Tyto otázky si vědci informačních technologií začali pokládat už v 50. letech minulého století, avšak výkon tehdejších počítačů jim neumožňoval tento postup využít v plné míře. Během vývoje teoretické informatiky se začala objevovat řada úloh, které nedokážeme vyřešit v polynomiálním čase[1]. U těchto úloh se často vzdáváme řešení deterministickými postupy a využíváme postupy heuristické (založené na náhodě a odhadu, nemusí však být přesné). Jedním z takových neobvyklých postupů jsou právě dědičné algoritmy. Průkopníkem v oblasti se stal John Holland (1919–2008), který pozoroval mentální procesy v populacích. Na základě svých poznatků navrhl evoluční algoritmus, jenž se zakládal přímo na biologických procesech v přírodě. Nejsilnější jedinci se rozmnožují nejvíce, a tak mají nejlepší příležitost předat svoje geny dál. Oproti tomu slabší jedinci se reprodukce účastní méně, nebo dokonce umírají bez potomstva. Tímto způsobem v každé další generaci dostáváme silnější jedince, protože svoje geny předali pouze ti nejsilnější. Budeme tedy předpokládat, že schopnost jedince budou určovat pouze jeho geny a nic jiného (v biologii – *darwinismus*).

1.1 Využití genetických algoritmů

Je důležité pochopit, že genetické algoritmy nejsou schopny řešit jakýkoliv problém, ba dokonce některé nezvládnou vyřešit vůbec. Základní indicie, že genetický algoritmus bude daný problém řešit jsou:

- Prohledávaný prostor je obrovský (může být i vícerozměrný) a řešení není na první pohled zřejmé.

[1] V nejhorším případě čas výpočtu bude $O(k^n)$, kde n je počet vstupů.

- Ideální řešení není potřeba a problém potřebujeme vyřešit v přijatelném čase.
- Můžeme numericky ohodnotit výkonnost jedince.
- Prohledávaný prostor je komplikovaný.

V dnešní době se genetické algoritmy běžně využívají v praxi, nejčastěji je můžeme nalézt při hledání nejkratší cesty na mapě (nejrychlejší spoj z bodu A do bodu B), optimalizace modelování, simulace a začínají vznikat roboti, kteří se v budoucnu dokážou přizpůsobit pod naše potřeby a nahradí naše každodenní činnosti jako praní nebo vaření (*Internet of Things)*. Genetické algoritmy jsou běžně užívány při řešení NP – těžkých úloh.

V dnešní teoretické informatice existují dvě základní třídy složitosti algoritmů a zavádí pojem tzv. Turingova stroje.

Turingův stroj (TS) je teoretický model počítače popsaný matematikem Alanem Turingem. Skládá se z procesorové jednotky, tvořené konečným automatem, programu ve tvaru pravidel přechodové funkce a pravostranně nekonečné pásky pro zápis mezivýsledků. Využívá se pro modelování algoritmů v teorii vyčíslitelnosti. [6]

Třída složitosti P je množina všech problémů, které můžou být vyřešeny na deterministickém TS v polynomiálním čase, oproti tom NP třída jsou problémy řešitelné na nedeterministickém TS. Kvůli složitosti NP třídy nám nezbývá nic jiného, než se pustit do heuristických postupů. Do třídy NP patří právě Problém obchodního cestujícího, který si ukážeme v kapitole *Výkon genetických algoritmů.*

2 Jak na genetické algoritmy

Každý genetický algoritmus obsahuje *populaci*. Jedná se o seznam *jedinců*, kteří mají svou jedinečnou DNA, na základě které můžeme určit jejich výkonost (dále označovanou jako *fitness*) neboli jejich schopnost řešení zadané úlohy. Dále musí obsahovat *selekci, křížení* a *mutaci.*

Obecný pseudokód genetických algoritmů [1]:

1. Vygenerování náhodné populace *P (0)* nastavení *t = 0*
2. Výpočet *fitness* všech jedinců
3. Pomocí *selekce* výběr dvojic a vytvoření jejich potomků *P ‘ (0)*
4. Ohodnocení *fitness P ‘ (0)*
5. Provedení *mutace* s pravděpodobností *m*
6. Vytvoření nové populace *P(t+1)* z *P ‘ (0)* a *P (0)*
7. *t = t + 1*
8. Ověření, zda algoritmus již nevyřešil zadaný úkol nebo jestli nebylo dosaženo jiné podmínky (maximální počet generací, stejné *fitness* v průběhu *n* generací)

Pokud necháme algoritmus projít mnoha generacemi, silnější jedinci se zkříží a postupně dosáhneme *populace*, jejíž jedinci odpovídají optimálnímu řešení úlohy. Protože genetické algoritmy jsou heuristické (tudíž v nich hraje značnou roli náhodnost) nemůžeme zaručit, že algoritmus nalezne ideální řešení v přijatelném čase nebo zda ho vůbec nalezne. Obecně pro ně platí, že jsou vhodné k *nalezení přijatelného řešení v přijatelném čase.*

2.1 Selekce

V této části využijeme darwinovského principu selekce. Zde musíme porovnat jedince, abychom určili ty, co se dostanou do další generace. Samotný proces může být rozdělen na dva kroky:

1. Určení *fitness*
2. Výběr jedinců na základě *fitness*

Pro ohodnocení je potřeba vytvořit speciální funkci, která by měla být úzce svázaná s konkrétním problémem. Funkce ve vstupu dostane DNA jedince a vrátí

numerickou hodnotu schopnosti řešit daný problém (např.: čím kratší cesta v labyrintu, tím vyšší hodnota *fitness*).

Pro výběr existují dva základní principy: ruletový mechanizmus a turnajová selekce. Tento krok je zásadní pro celý algoritmus, protože určuje jeho efektivitu. Problém spočívá v balancování mezi rozmanitostí DNA a přitom silnější jedinec musí mít větší šanci se dostat dál. Jelikož tento proces napodobuje přirozený výběr a značnou roli hraje náhoda, ne vždy silnější individuum bude vybráno. Pokud ale jedinec bude mít až moc velkou šanci být vybrán, ztrácí se rozmanitost a algoritmus pak může najít pouze suboptimální řešení.

Ruletový mechanizmus spočívá v přiřazení každému jedinci část výseče na pomyslné *ruletě*. K vytvoření kola stačí pouze spočítat procentuální zastoupení jednotlivých hodnocení vůči celkovému součtu. Pokud uvažujeme o populaci velikosti *n* a ohodnocení označíme jako f_i, tak pro každého jedince *i* je pravděpodobnost vyjádřena vztahem:

$$p_i = \frac{f_i}{\sum_{i=1}^{N} fi} \cdot 100\ \%$$

Rovnice 1 - Pravděpodobnost vybrání jedince v ruletovém mechanizmu

Druhý velmi využívaný princip vybíraní jedince se nazývá turnajová selekce. I v tomto případě se čerpala inspirace v přírodě, kde se jedinci často utkávají v přímém souboji o právo účastnit se reprodukce. Proces probíhá tak, že se vybere část jedinců o velikosti *m* a ten s nejvyšší hodnotou *fitness* předá svoje geny dál. Kvůli zavržení slabších jedinců se vytváří poměrně velký selekční tlak a tím mizí rozmanitost populace, proto obvykle *m* nebývá větší než 5.

2.2 Křížení

Nejčastější způsob vytvoření potomků z vybraných rodičovských párů je takzvané jednobodové křížení. V DNA vybereme náhodný bod, tam jedince rozpůlíme a prohodíme jejich druhé části. Pokud máme dostatečně dlouhé DNA můžeme využít i dvoubodové křížení, které funguje obdobně, s tím, že se prohazuje prostřední část obou. Vetší počet bodů se nevyužívá.

Křížení ale nepoužíváme ve všech případech, občas je totiž dobré zachovat jedince pro další generace a tím vznikají přímé kopie rodičovských párů. Pravděpodobnost

křížení by ale neměla klesnout pod 70 % a stoupnout nad 90 % [3]. Další nástroj, který se často využívá pro zachování potenciálně správného řešení je elitismus. Principem je, že z každé generace se vybere 3–5 nejlepších jedinců, kteří se beze změny pošlou do další generace. Tímto způsobem nenastane situace, kdy genom nejlepších jedinců bude ztracen.

2.3 Mutace

Tento operátor do algoritmu přidáváme za účelem zachování rozmanitosti a brání zjednodušování vlastností. Mutace s jistou pravděpodobností *p* změní hodnotu genu. Tato pravděpodobnost se obvykle volí mezi 5 % až 15 % [4].

2.4 Nastavení parametrů genetických algoritmů

Jak jsem se už zmiňoval výše, základem funkčnosti genetického algoritmu je jeho správné nastavení. Pokud nastane situace, kdy budou zvoleny nevhodné hodnoty, může se stát, že algoritmus nenalezne ani suboptimální řešení, dokonce nemusí nalézt řešení žádné. Každý případ se musí posuzovat individuálně a existuje pouze pár základních pomůcek, jak postupovat při řešení problému.

Pro simulaci základního genetického algoritmu jsem použil myšlenku, kterou v roce 2009 zpracoval Jer Thorp, pod názvem Smart Rockets [7]. Ve svém blogu chtěl podotknout, jak NASA používá evoluční postupy od řešení designu radiových antén satelitů po konfiguraci odpalu raket a v budoucnosti plánuje vytvořit sondy, které se dokáží adaptovat nečekaným podmínkám bez zásahu ze strany člověka.

Pro návrh aplikace pro své účely jsem použil podklady Daniela Shiffmana [2], jehož původní simulace je psána v JavaScriptu. Algoritmus je převedený do jazyka Processing, který se zakládá na Javě, ale je zjednodušený pro výukové účely. Obsahuje nový datový typ PVecor, který v sobě nese informaci o pozici, směru a velikosti vektoru a lze s ním jednoduše pracovat. Dále jsem ho upravil, aby odpovídal mým požadavkům a aby mohl nastínit, jak ho správně nastavit. Uživatelské rozhraní je doplněno o textboxy v levé dolní části a informace o generaci jsou přeloženy do češtiny. Textboxy umožňují úpravu velikosti populace, pravděpodobnost mutace, počet náhodných jedinců v jedné populaci a maximální zrychlení. Nastavení zrychlení nemá žádný vliv na výkon, oproti tomu zbytek parametrů úzce souvisí s efektivitou algoritmu. Do programu jsou taky přidány nové funkce. Po zmáčknutí mezerníků se objeví překážky (pokaždé stejné), které

rakety musí překonat. Pokud by mnou nastavené překážky nevyhovovaly, lze opětovným zmačknutím mezerníku překážky vymazat a tažením myši nakreslit vlastní.[2]

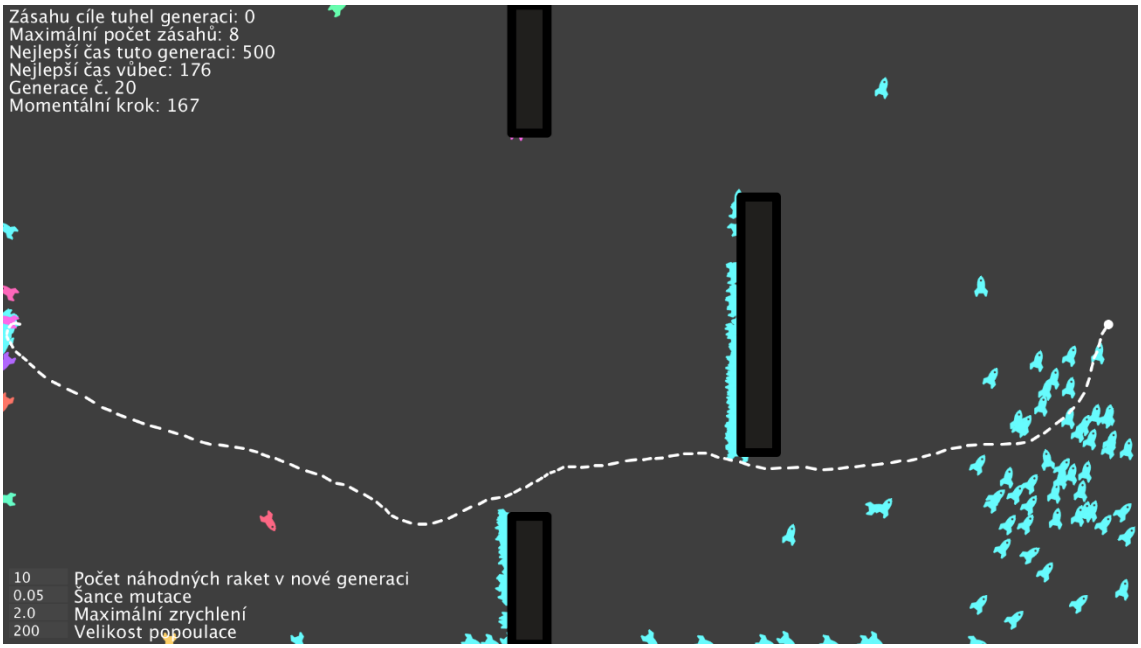
Obrázek 1 - Stav programu během první generace

Pro řešení úlohy je využita standardní selekce pomocí rulety, kdy každý jedinec se dostane do rulety n-krát podle podílu jeho *fitness* s největším *fitness* celé generace vynásobeného 100. Rozmanitost genotypu je zajištěna mutací o pravděpodobnosti *m* a počtem *r* náhodných jedinců v každé iteraci (počet potomků předešlé generace je pak *nová populace* = *velikost populace* - *r*). Hodnota *r* určuje, kolik naprosto nových jedinců bude přidáno do každé generace, kde *r* je z intervalu <0; *velikost populace*>. Genetická informace je vytvořena z pole PVectorů, který udává, jakým směrem a jakou velikostí se má na raketu působit. Křížení probíhá vybráním dvou jedinců z rulety a podle náhodného bodu *p* se jejich geny překříží a vznikne tak jejich potomek, který se přidá do další generace. Následně pro každý gen potomka proběhne mutace s pravděpodobností *m*, po které se daný gen zamění na nový náhodně vygenerovaný PVector. Nejproblémovější částí celého programu byla správná implementace ohodnocující funkce. Nakonec je řešena způsobem, že raketa spočítá vzdálenost *d* od cíle a pak hodnotu naškáluje pomocí funkce *map()*[3]. Tímto způsobem raketa, která má vzdálenost od cíle 0, dostane nejvyšší možné ohodnocení a naopak. Pokud raketa zasáhla překážku, její *fitness*

[2] Kompletní zdrojový kód je dostupný online (viz *Přílohy*). Obsahuje podrobný popis každé funkce.

[3] Přepočítá číslo z jednoho rozsahu do druhého.

se ještě zmenšil. Tím bylo dosaženo, že rakety, které se snažily vyhnout, měli větší zastoupení v populaci (Obrázek 2).



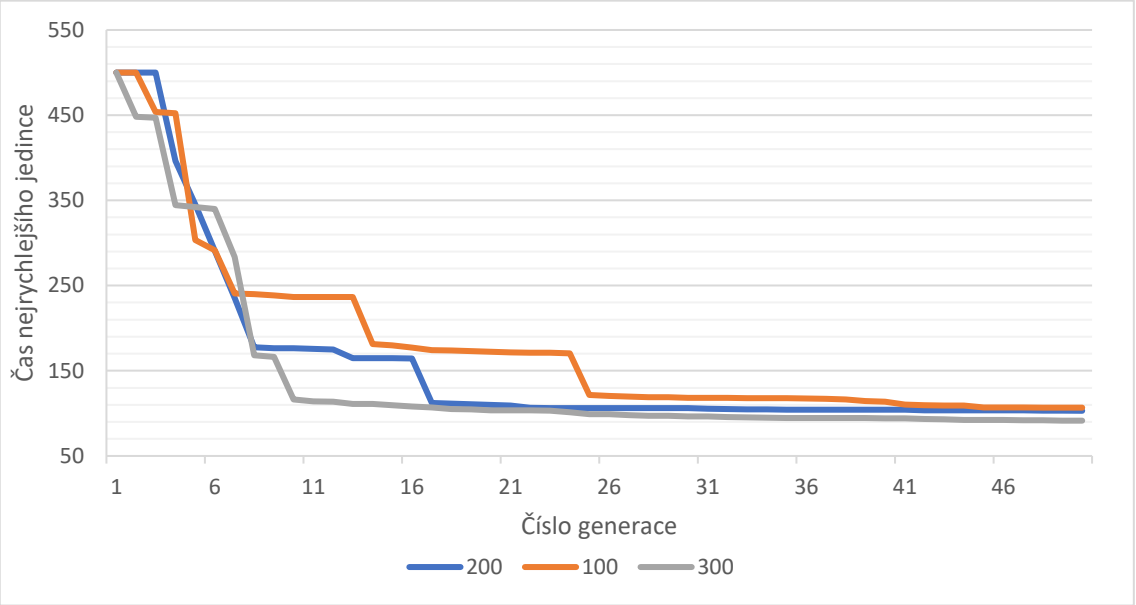
Obrázek 2 - Stav programu během 20. generace

V následujících krocích chci zjistit, jaký vliv na efektivitu mají hodnoty velikosti populace, šance mutace a počet náhodných jedinců. Různá nastavení budu porovnávat podle počtu kroků (genů) které byly potřeba, aby se raketa dostala do cíle. Tento způsob umožnil zanedbat výkon hardwaru a optimalizaci aplikace. Pro základní nastavení jsem vybral tyto parametry:

- Velikost populace: 200 jedinců
- Pravděpodobnost mutace: 5 %
- Počet nových jedinců: 10

Vždy bude testován jeden z parametrů, dva zbývající budou nastaveny právě takto. Je to z důvodu zachování měřitelnosti. Budu provádět vždy 7 testů, výsledky zprůměruji a s těmi pak budu pracovat.

2.4.1 Velikost populace

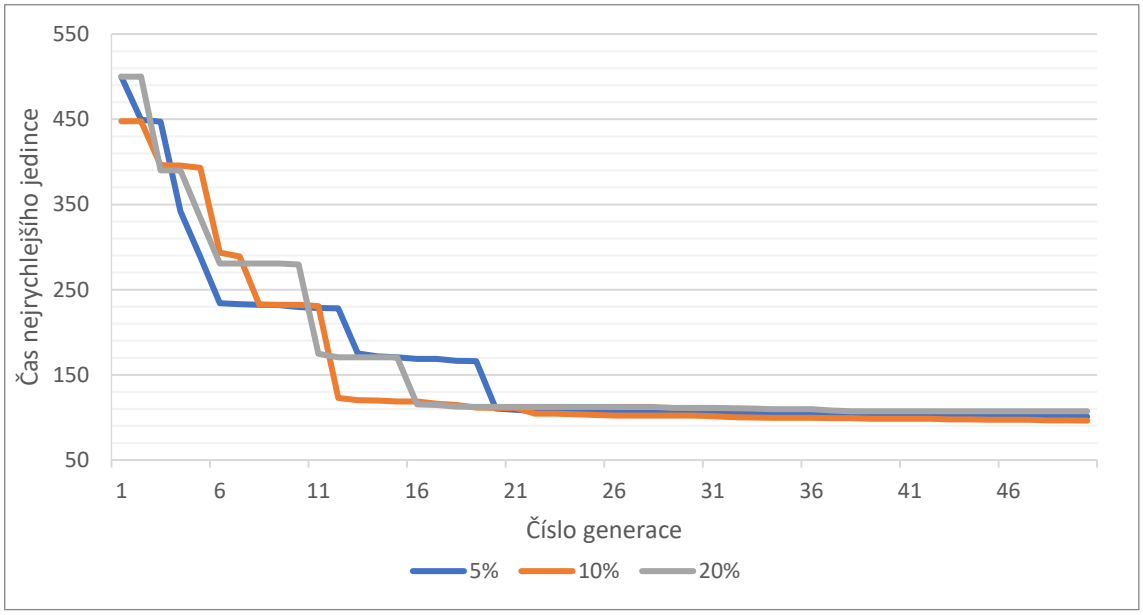


Graf 1 - Grafické porovnání rychlosti řešení populací s různou velkostí populace

Pro zdokumentování důležitosti velikosti populace jsem použil hodnoty 100, 200 a 300. Jak lze vyčíst z grafu, čím větší populace je, tím rychleji algoritmus nalezne řešení. Kvůli tomu, že se celý proces ze značné míry zakládá na náhodě, nelze to v žádném případě považovat za přímou úměrnost. Rozdíl je značně viditelný již po desáté generaci, kde rozdíl mezi řešením se 100 jedinci a 300 je celých 236 kroků (Graf 1). Sice výsledná řešení se od sebe tolik neliší, 26 generací, které byly potřeba 100 jedincům nalézt alespoň přibližně stejné řešení, je značný rozdíl. Důvodem pro takto výrazný rozdíl je, že 300 individuí má třikrát větší šanci nalézt řešení. Tak velký počet dokáže lépe pokrýt hledaný prostor, a tak nemá problém nalézt řešení už v prvních iteracích. Další problém, který nastal s populací o 100 jedincích, byly její nekonstantní výsledky. Odchylka měření se pohybuje až na 20 %, oproti 12 % u populace s 300 jedinci. V tuto chvíli se ale dostáváme do problému s optimalizací. Při řešení komplikovanějších úloh bychom měli brát v potaz, zda se nám vyplatí vytvářet tak velké populace. Záleží to především na tom, jak optimální řešení chceme nalézt. Pokud porovnáme výsledky 200 a 300 jedinců, můžeme si všimnout, že čas se už u 20. generace liší o velmi malou hodnotu. V této situaci by 200 jedinců mohlo najít řešení rychleji, protože výpočet jedné generace o 300 se může výrazně zpomalit při počítání dráhy jedinců, kteří nenesou perspektivní genetický kód. Bohužel bez přístupu ke kvalitní výpočetní technice a programu navrženém pro tento problém, svou hypotézu nijak nemůžu potvrdit.

2.4.2 Pravděpodobnost mutace

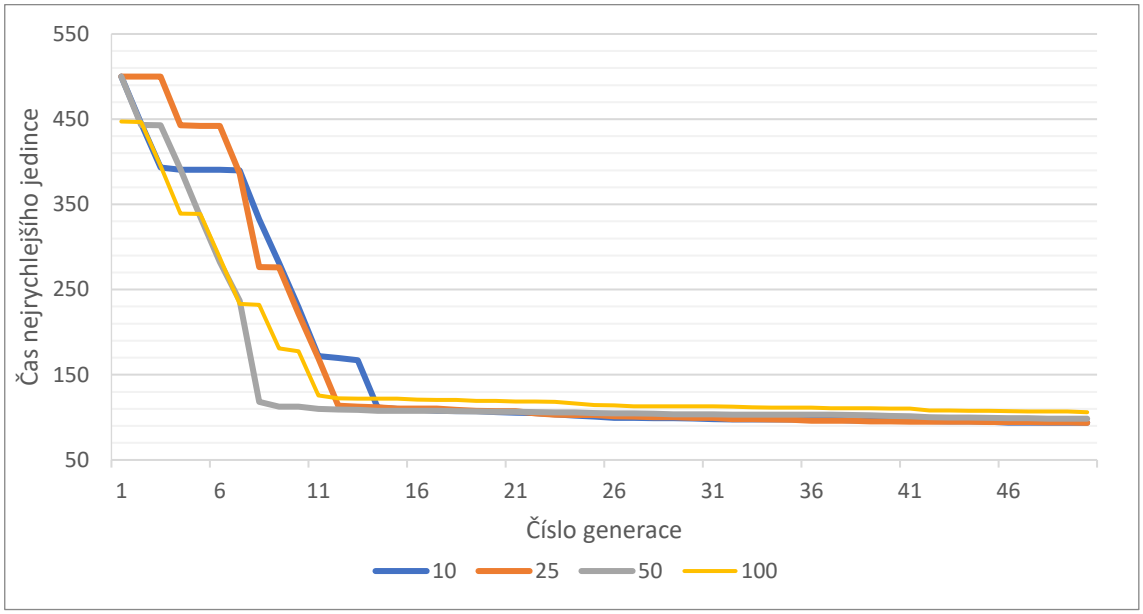
V tomto případě nebudou rozdíly tak drastické, jelikož to tolik neovlivňuje prohledávaný prostor. Pokud se ale zaměříme na důvod, proč mutace je obsažena v kódu, můžeme vyvodit, že vyšší mutace nedovolí algoritmu se zaseknout v suboptimálním řešení. Nastává však problém, jak moc vysoká šance by měla být, aby neovlivňovala počáteční prohledávání, kdy správná permutace genů je velmi důležitá.



Graf 2 - Grafické porovnání rychlosti řešení s různou pravděpodobností mutace

Pro hodnoty mutace jsem testoval šanci 5 %, 10 % a 20 %. Je zajímavé, že i při vysoké šanci mutace genu algoritmus tolik nezaostával za ostatními, a dokonce i nalezl přibližně stejné řešení po 20 generacích (Graf 2). Musím ale poukázat na to, že algoritmus s 10 % to dokázal nalézt skoro dvakrát rychleji. Ukázalo se, že 10 % je dostatečné na to, aby v průběhu nedocházelo k stagnaci jedinců a program se nezasekl u suboptimálního řešení a nedocházelo k rozbití permutací již vytvořených. Protože se ale evoluce zakládá na náhodě, nemá cenu určovat konkrétní číslo, protože při dalším testování se může ukázat, že o jednu jednotku větší číslo si vedlo lépe, stačí nám tedy číslo jen přibližné. 10 % také mělo nejmenší časovou odchylku, pohybující se kolem 16 % oproti 26 % a 20 % zbylých dvou. Největší odchylku pak měl algoritmus s 5 % (26 %), který by na první pohled měl mít odchylku nejmenší. Je to způsobeno tím, že program často nedokázal najít cestu kvůli tomu, že se zasekl v prohledávání a zdokonalování nevýhodné cesty.

2.4.3 Počet náhodných jedinců v jedné generaci



Graf 3- Grafické porovnání rychlosti řešení s různým počtem náhodných jedinců

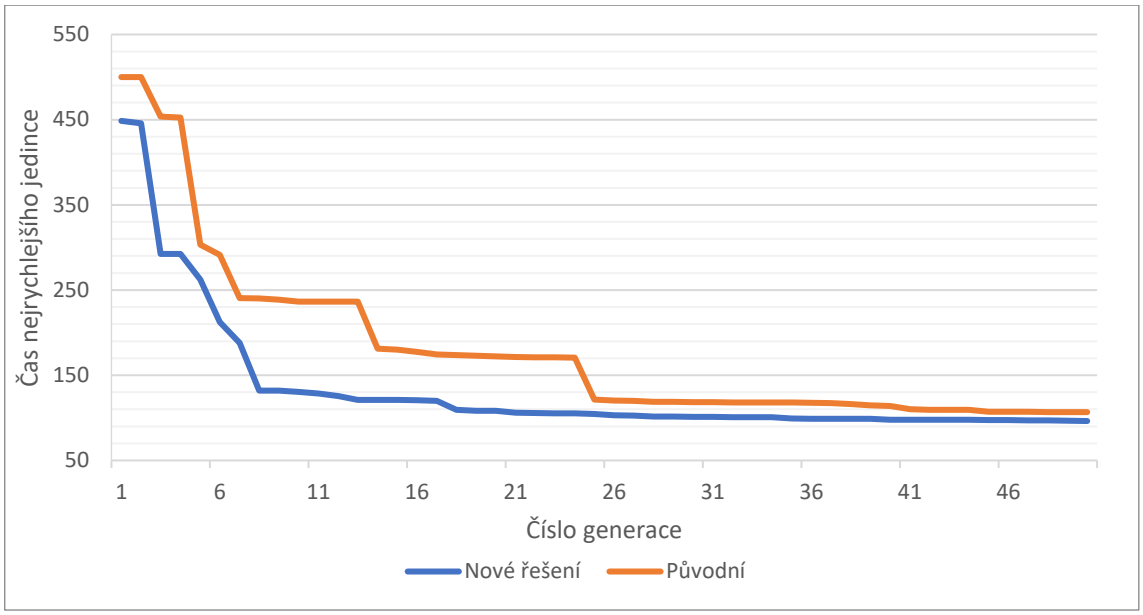
Dopad náhodných jedinců nebyl tak výrazný jako u předchozích parametrů (Graf 3). Opět se jedná o nástroj k zachování diverzity populace. Nejefektivnější počet se nachází mezi 25 až 50 jedinci (co odpovídá 12,5 % až 25 % z celé populace). 100 náhodných jedinců už bylo moc, protože na konci už algoritmus nedokázal zdokonalit cestu na úroveň ostatních.

2.4.4 Porovnání

Z testů vyplynulo, že nejlepších výsledků lze dosáhnout:

- Populace: čím větší, tím je větší šance nalézt lepší dráhu
- Mutace: kolem 10 %
- Počet náhodných jedinců: mezi 12,5 % a 25 % z populace

Výsledky jednoznačně ukazují, že nové nastavení je mnohem lepší než to, co bylo na začátku.



Graf 4 - Grafické porovnání rychlosti řešení nově navržených hodnot

3 Výkon genetických algoritmů

V této kapitole chci zjistit, jaký je potenciál pro využití evolučních postupů v reálnem čase. Jako prostředí pro simulaci reálného problému chci použít známý optimalizační problém zvaný Problém obchodního cestujícího.

Problém obchodního cestujícího (anglicky Travelling Salesman Problem – TSP) je obtížný diskrétní optimalizační problém, matematicky vyjadřující a zobecňující úlohu nalezení nejkratší možné cesty procházející všemi zadanými body na mapě. [8]

Pro řešení TSP je navržena řada algoritmů jako je Brute Force, Closest Neighbour, řešení haldou, mravenčí kolonií, Hill Climbing a Simulated Annealing. Právě posledně zmíněný algoritmus Simulated Annealing (Simulované žíhání) použiji v porovnání.

Porovnání se bude provádět v Javě (jako vývojové prostředí je použita platforma NetBeans), kde jsem se snažil oba algoritmy napsat co nejpodobněji, aby lidský faktor měl co nejmenší vliv na porovnání.[4]

3.1 Řešení pomocí simulovaného žíhaní

Postup žíhání je jeden z dalších nápadů, které jsou inspirovány přírodou. V tomto případě se nápad zakládá na prohledávání prostoru pomocí simulace žíhání oceli. Tento postup byl navržen se stejným záměrem jako genetické algoritmy: v přijatelném čase naleznout přijatelné řešení. U klasických řešení (jako Hill Climbing) se může často stát, že výsledek uvízne v lokálním minimu. V tomto případě to ale neřešíme pomocí populace, ale s určitou pravděpodobností přijetí zřejmě horšího řešení. Šance takto provedené změny záleží na *teplotě*. Čím větší je teplota, tím větší se provádějí změny. Jak postupně teplota klesá, struktura řešení je víc stálá a horší řešení se už skoro nepřijímají. Pravděpodobnost *p* přijetí horšího řešení, kde *r* je hodnota nejlepšího řešení, *n* je hodnota nového řešení a *t* je momentální teplota algoritmu.

$$p = e^{\frac{r-n}{t}} \cdot 100\ \%$$

Rovnice 2 - Pravděpodobnost přijetí horšího řešení

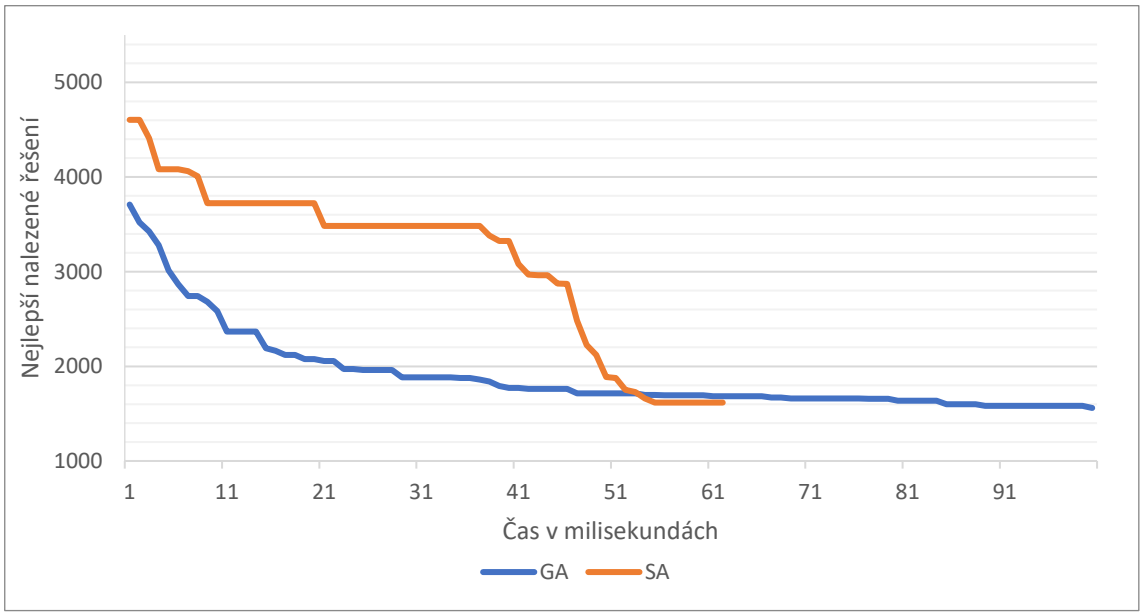
[4] Oba programy s grafickým řešením budou dostupné online

3.2 Řešení pomocí genetického algoritmu

Navržený postup je velmi podobný tomu, který je využit projektu *Smart Rockets*. *Fitness* se počítá jako celková délka cesty. Selekční mechanismus utrpěl řadu změn: byl přidán elitismus a namísto rulety je vytvořen výběr turnajem.

Pro správnou funkčnost algoritmu bylo potřeba pozměnit i mutaci. Protože pokud by metoda do řetězce měst přidávala nebo ubírala spoje, cesta by nebyla validní a celý proces by byl k zahození. Proto metoda vybere dva náhodné body z cesty a prohodí je. To samé platí i pro křížení, pokud by vznikl jedinec se dvěma stejnými městy, nedal by se správně ohodnotit.

3.3 Porovnání

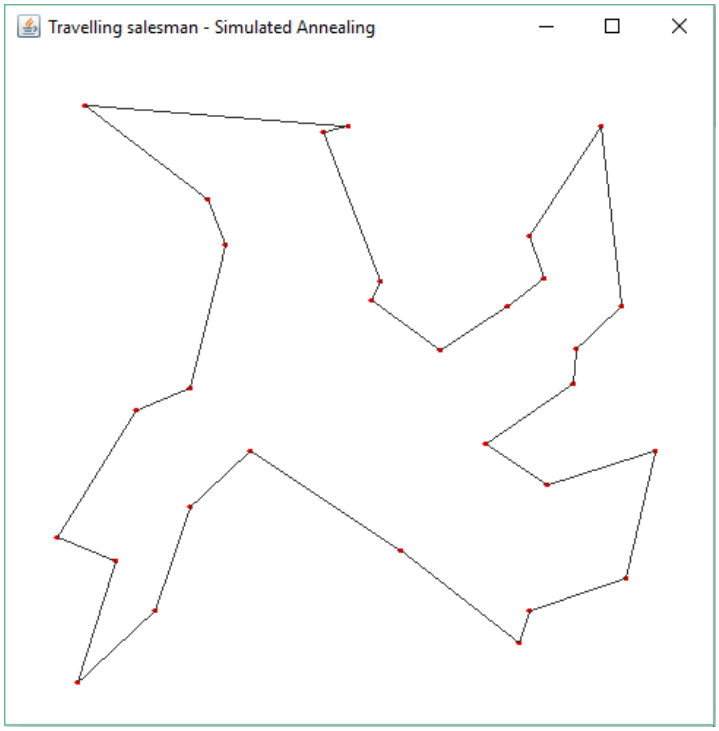


Graf 5 – Rychlost řešení TSP pomocí GA a SA

Oba algoritmy řešili TSP se 30 městy a program zapisoval nejlepší průběžnou cestu. Z výsledků je vidět, že speciálně navržený algoritmus (SA), se vypořádal lépe (nehledě na počáteční odhad, kde si genetický algoritmus náhodou našel lepší cestu z možných !30 kombinací). SA už po 61 ms nalezl své konečné řešení, oproti tomu genetický algoritmus (GA) dál pokračoval a byl zastaven po 100 ms. Je možné, že pokud by běžel dál, tak by našel i lepší řešení než SA. Otázkou pak je, zda nám to za to stojí.

Výhodou evolučních postupů je, že výsledek oproti ostatním, nezávisí na průběhu. Pokud bychom nutně potřebovali zkrátit čas u simulovaného žíhání, probíhalo by to velmi obtížně. Pokud se zaměříme na průběh, je pro simulované žíhání velmi důležitý jak začátek (upevňování základní struktury řešení), tak i konec (vylepšování struktury

krajních oblastí), kde můžeme pozorovat největší spád. Pro dosažení lepšího výsledku bychom potřebovali vylepšit samotnou logickou část programu, jinak nastavit počáteční hodnotu teploty nebo rychlost chlazení struktury. Oproti velikost cesty klesá u genetického algoritmu klesá rovnoměrně po celou dobu výpočtu a v počátečních hodnotách si vedl mnohem lépe. Jestli bychom chtěli zkrátit výpočetní čas, stačilo by pouze omezit počet generací a stejně bychom dosáhli uspokojivých výsledků. Na takto malém počtu uzlů a vzdálenostech mezi nimi rozdíl není tak znatelným, jak by byli při prohledávání mnohem komplikovanějších grafů (3D nebo závislost na čase).



Obrázek 3 - Grafické řešení TSP

4 Závěr

V této práci jsem podrobně rozebral část celého odvětví moderní informatiky. Samozřejmě, že by se dalo polemizovat, zda sem nespadají i další věci jako neuronové sítě, Deep Learning a další. Určitě všechno spolu úzce souvisí a jedno by nemohlo existovat bez druhého.

Cílem mé práce bylo seznámit sebe i čtenáře s tématem a rozšířit všeobecné povědomí o tomto oboru. Tomu bylo věnovaná první a druhá část práce. Popsal jsem nejčastěji využívané postupu v oblasti selekce, ohodnocujících funkcí, křížení a mutace. V této části byli taky otestovány schopnost algoritmu řešit problém v 2D prostoru. Byl zde představen program Smart Rockets, kde se tyto postupy využily. Na konci pak proběhlo srovnání jednotlivých parametrů algoritmu a jejich vliv na rychlost řešení.

Ve třetí části je popsán genetický algoritmus na řešení problému obchodního cestujícího. Navržený genetický algoritmus jsem porovnal se algoritmem zvaným simulované žíhání. Výsledkem testování pak bylo srovnání obou postupů, jejich plusy a mínusy.

TSP je jedním z optimalizačních problému, který se často využívá v praxi. Problém se zabývá plánováním nejkratší cesty skrze několika uzly (městy). Je mnohem výhodnější při rozvozu zboží urazit nejkratší cestu, aby se náklady na cestu snížily co nejvíce. Lze zde provést přímou analogii s průmyslovým inženýrstvím, protože plánování a logistika je jedním z jeho odvětví.

Při sbírání informací jsem nabyl spoustu nových znalostí a zkušeností, které by mi v budoucnu velmi ulehčily další výzkum problematiky. Dalším krokem pro porozumění odvětví by byl rozbor a vývoj neuronových sítí. Ty však už nejsou zdaleka tak jednoduché a spíš by patřily mezi učivo vysokých škol.

Evoluční algoritmy už přežili přes 40 let a stále se využívají v inženýrství a optimalizačních problémech, ve kterých výpočetní čas není velkým problémem. Tyto postupy přináší řešení pro řadu problémů řadících se mezi NP – těžké. Další vývoj a nová rozšíření stále přibývají (jako kvantově inspirované genetické algoritmy – QIGA) a nemají problém se uchytit na trhu. Jak to ale bude vypadat v nebližší době, záleží na spoustě faktorech. Pokud opomineme čistě technické problémy, jako výkony CPU, GPU a datová uložiště, musíme se taky zeptat, zda vůbec bude poptávka po těchto

technologiích. Podle některých předpokladů se do 10 let se vyřeší nynější problémy (řeč, jazyk, překlad) a genetické analýzy budou na vrcholu své slávy.

V posledních letech se do této oblasti začaly pouštět i největší softwarové a hardwarové firmy jako Google, Nvidia, AMD a další. Jedním ze zajímavějších případů je americká Tesla, která udělala obrovské pokroky v autopilotech automobilů a parkovacích asistentech. To, že i nynější automobily dokážou samostatně jezdit za pomocích genetických postupů a neuronových sítí, jen poukazuje na rozvoj odvětví, zejména v analýzách dat a předpovědích.

Pokrok však přináší řadu morálních otázek. Genetické postupy se pomalu ale jistě dostávají do našich každodenních životů, aniž by si toho mnozí uvědomovali (vyhledávaní spojů na IDOS je velmi podobný TSP popsanému výše). Mnozí se obávají scénáře z akčního filmu Terminátor, ale má to nějaké reálné podklady? Sice se bát rozbité umělé inteligence v nejbližší době nemusíme, tento rapidní rozvoj však může zasáhnout i jiná odvětví. Modernizace výroby a rutinních zaměstnání (Amazon v roce 2018 spustil první obchod bez pokladních) může stát statisíce až miliony lidí práci. Momentálně řešení ještě nalezeno nebylo a zůstává to otevřenou otázkou.

Další věc, co mnoho lidí trápí, je, zda počítače jsou schopné rozhodovat o morálce. Pokud automobil s autopilotem naboura, kdo za to bude zodpovídat? Firma, řidič nebo vývojář? Většina lidí, co zastává názor, že umělá inteligence za volant nepatří, si ale neuvědomuje to, že čím víc zahazujeme nové technologie, tím víc lidí bude obětmi lidských faktorů (umělá inteligence nemůže být opilá nebo ospalá).

5 Zdroje

5.1 Použitá literatura

1. HYNEK, Josef. Genetické algoritmy a genetické programování. Praha: Grada, 2008. Průvodce (Grada). ISBN 978-80-247-2695-3.

5.2 Elektronické zdroje

2. Coding Challenge 29: SmartRockets [online]. Daniel Shiffman, 2017 [cit. 2018-05-10]. Dostupné z: https://github.com/CodingTrain/website/tree/master/CodingChallenges/CC_029_SmartRockets
3. První série dvacátého osmého ročníku KSP [online]. Praha: Matematicko-fyzikální fakulta Univerzity Karlovy, 2015 [cit. 2018-05-10]. Dostupné z: http://ksp.mff.cuni.cz/tasks/28/tasks1.html#task8
4. The Nature of Code [online]. Spojené: Daniel Shiffman, Shannon Fry, Zannah Marsh, 2012 [cit. 2018-05-10]. Dostupné z: http://natureofcode.com/
5. Small rocket ship silhouette free icon. In: Flaticon [online]. 2017 [cit. 2018-05-10]. Dostupné z: https://www.flaticon.com/free-icon/small-rocket-ship-silhouette_25452
6. Turingův stroj. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-05-10]. Dostupné z: https://cs.wikipedia.org/wiki/Turing%C5%AFv_stroj
7. PROJECT: Smart Rockets [online]. Jer Thorp, 2009 [cit. 2018-05-10]. Dostupné z: http://blog.blprnt.com/blog/blprnt/project-smart-rockets
8. Problém obchodního cestujícího. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-05-10]. Dostupné z: https://bit.ly/2m4xCj3

5.3 Nástroje využité během práce

- Oracle Corporation: NetBeans IDE 8.1. Dostupné z: https://netbeans.org/
- Casey Reas, Ben Fry: Processing 3.3.7. Dostupné z: https://processing.org/
- Microsoft: Microsoft Office 2016. Dostupné z: https://office.com/

6 Přílohy

6.1 Java

Travelling Salesman – Simulated Annealing. Dostupné online: https://github.com/demolice/Projects/tree/master/RP/TSP/Simulated%20annealing

Travelling Salesman – Genetic algorithm. Dostupné online: https://github.com/demolice/Projects/tree/master/RP/TSP/Genetic%20algorithm

6.2 Processing

Smart Rockets. Dostupné online: https://github.com/demolice/Projects/tree/master/RP/SmartRockets

6.3 Data z grafů

Soubor všech podkladků pro data: https://github.com/demolice/Projects/tree/master/RP/Data

6.4 Seznam grafů

6.5 Seznam rovnic

6.6 Seznam obrázků

6.7 Seznam použitých zkratek

SA	Simulated annealing
GA	Genetický algoritmus
TSP	Travelling salesman problem
NP	Nedeterministicky polynomiální
TS	Turingův stroj