**UNIVERZITA PARDUBICE**

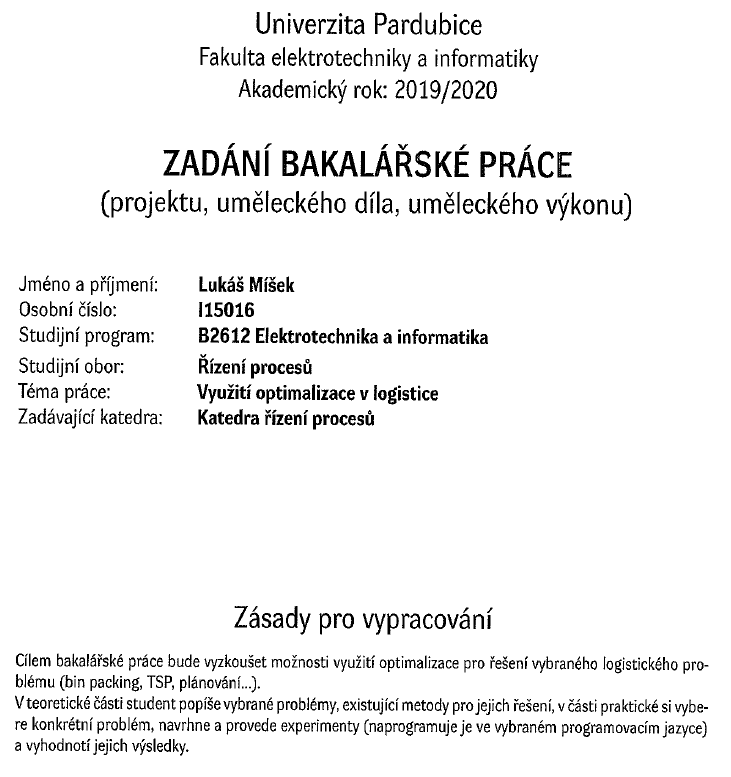
Fakulta elektrotechniky a informatiky

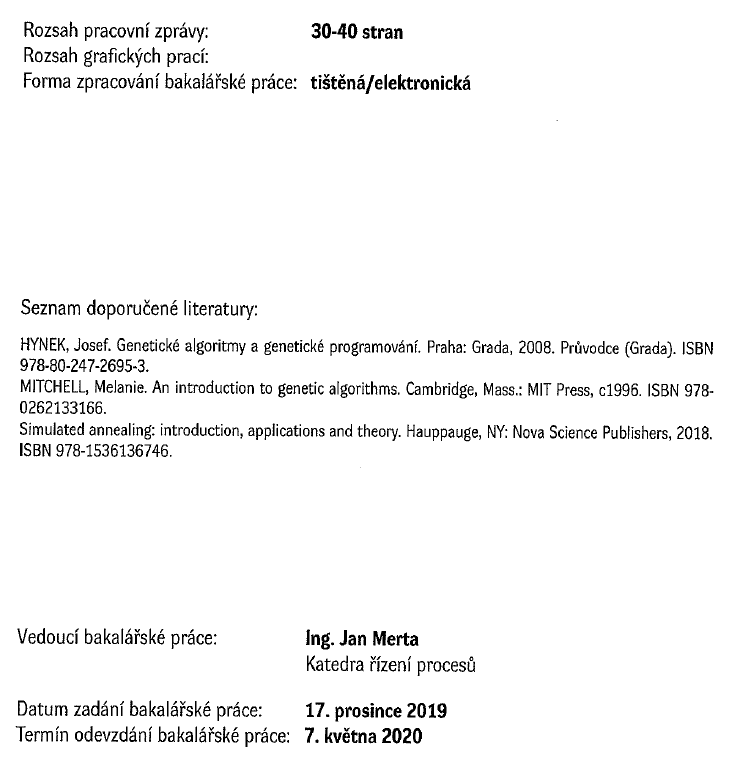
**Využití optimalizace v logistice**

Lukáš Míšek

Bakalářská práce

2020





**Prohlášení**

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.



V Pardubicích dne 15. 03. 2020

Lukáš Míšek

**ANOTACE**

*Cílem bakalářské práce bude vyzkoušet možnosti využití optimalizace pro řešení vybraného logistického problému (bin packing, TSP, plánování...). V teoretické části student popíše vybrané problémy, existující metody pro jejich řešení, v části praktické si vybere konkrétní problém, navrhne a provede experimenty (naprogramuje je ve vybraném programovacím jazyce) a vyhodnotí jejich výsledky.*

**KLÍČOVÁ SLOVA**

*Optimalizace, obchodní cestující, umělá inteligence, genetické programování, logistika.*

**TITLE**

*Optimization, travelling seller, artificial inteligence, genetic programming, logistics.*

**ANNOTATION**

*Goal of this work is to try use of the optimilization for solving in logistic problém (bin packing, TSP, planing etc). In teoretical part student will describe chosen problem, existing methods, design and realize experiments (programm in chosen programming language) and analyse results.*

**KEYWORDS**

*microcontroller, programming, control system.*

**OBSAH**

[SEZNAM ILUSTRACÍ 10](#_Toc39592980)

[ÚVOD 12](#_Toc39592981)

[1 SEZNÁMENÍ SE S PROBLEMATIKOU 13](#_Toc39592982)

[1.1 Rozbor zadání 14](#_Toc39592983)

[1.2 Rešerše literatury 14](#_Toc39592984)

[1.3 Třídy složitosti NP a P 15](#_Toc39592985)

[1.4 TSP (Traveling salesman problem) 15](#_Toc39592986)

[1.5 VRP (Vehlice routing problem) 16](#_Toc39592987)

[1.6 Řešení VRP pomocí GA 16](#_Toc39592988)

[2 GENETICKÉ ALGORITMY 17](#_Toc39592989)

[2.1 Genetický algoritmus 17](#_Toc39592990)

[2.2 Reprezentace jedince 18](#_Toc39592991)

[2.3 Fitness jedince 20](#_Toc39592992)

[2.4 Selekce 21](#_Toc39592993)

[2.5 Genetické operátory 23](#_Toc39592994)

[2.6 Křížení 24](#_Toc39592995)

[2.7 Mutace 25](#_Toc39592996)

[2.8 Příklad skládání puzzle 26](#_Toc39592997)

[2.9 Problém N dam 26](#_Toc39592998)

[3 NÁVRH ALGORITMU 28](#_Toc39592999)

[3.1 Úloha, problematika 28](#_Toc39593000)

[3.2 Rozbor zadání 29](#_Toc39593001)

[3.3 Návrh řešení 29](#_Toc39593002)

[3.4 Blokové schéma 31](#_Toc39593003)

[3.5 Komponenty algoritmu 32](#_Toc39593004)

[3.5.1 Vnitřní algoritmus 32](#_Toc39593005)

[3.5.2 Vnitřní jedinec 32](#_Toc39593006)

[3.5.3 Vnitřní populace 32](#_Toc39593007)

[3.5.4 Vnější algoritmus 33](#_Toc39593008)

[3.5.5 Vnější jedinec 33](#_Toc39593009)

[3.5.6 Vnější populace 33](#_Toc39593010)

[3.6 Pseudokód algoritmu 34](#_Toc39593011)

[3.7 Možné výstupy algoritmu 35](#_Toc39593012)

[3.7.1 Města jsou nevhodně rozdělena na úrovni vnějšího jedince 35](#_Toc39593013)

[3.7.2 Nebyla dosažena nejkratší vzdálenost 36](#_Toc39593014)

[3.8 Odhad optimálního řešení 36](#_Toc39593015)

[4 ŘEŠENÍ VLASTNÍ PRÁCE 37](#_Toc39593016)

[4.1 Krok Výpočet nejkratší vzdálenosti 37](#_Toc39593017)

[4.1.1 Matice vzdáleností 37](#_Toc39593018)

[4.1.2 Pythagorova věta 38](#_Toc39593019)

[4.1.3 Reálná vzdálenost oproti vzdušné čáře 39](#_Toc39593020)

[4.2 Krok Programování vnitřního jedince 41](#_Toc39593021)

[4.2.1 Vhodná reprezentace vnitřního jedince 41](#_Toc39593022)

[4.2.2 Reprezentace vnitřní generace 42](#_Toc39593023)

[4.2.3 Zlepšení populace 43](#_Toc39593024)

[4.2.4 Fitness jedince 43](#_Toc39593025)

[4.2.5 Zobrazeni jedince 44](#_Toc39593026)

[4.2.6 List<string> – Orders | InvidualOrders | PopulationOrder 44](#_Toc39593027)

[4.2.7 SuperInvidual 44](#_Toc39593028)

[4.3 Pomocné třídy 47](#_Toc39593029)

[4.3.1 Functions 47](#_Toc39593030)

[4.3.2 InputOutput 48](#_Toc39593031)

[4.3.3 Cities 48](#_Toc39593032)

[4.3.4 Controller 49](#_Toc39593033)

[5 METODIKA A VYHODNOCENÍ POKUSŮ 50](#_Toc39593034)

[5.1 Zkoumání vlivu počtu generací vnitřního a vnějšího jedince 52](#_Toc39593035)

[5.1.1 Shrnutí měření 52](#_Toc39593036)

[5.1.2 Grafy s naměřenými daty 53](#_Toc39593037)

[5.2 Zkoumání vlivu ceny za kilometr a ceny za auto 55](#_Toc39593038)

[5.2.1 Shrnutí měření 55](#_Toc39593039)

[5.2.2 Grafy s naměřenými daty 56](#_Toc39593040)

[5.3 Zkoumání vlivu umístění cílových měst 59](#_Toc39593041)

[5.3.1 Shrnutí měření 59](#_Toc39593042)

[5.3.2 Grafy s naměřenými daty 60](#_Toc39593043)

[5.4 Vliv počtu generací na nejkratší vzdálenost 61](#_Toc39593044)

[5.5 Vliv ceny na fitness jedince 61](#_Toc39593045)

[5.6 Vliv rozmístění měst na nejkratší vzdálenost 61](#_Toc39593046)

[6 ZÁVĚR 62](#_Toc39593047)

[SEZNAM LITERATURY 63](#_Toc39593048)

[PŘÍLOHY 64](#_Toc39593049)

**SEZNAM ZKRATEK**

array datový typ pole se statickým rozměrem

bmp soubor s obrázkem ve formátu bmp

csv textový soubor s oddělovači

double datový typ reálné číslo

char datový typ znak

GA genetický algoritmus

int datový typ celé číslo

jpg soubor s obrázkem ve formátu jpg

list datový typ pole s dynamickým rozměrem

pdf soubor ve formátu pdf

string datový typ řetězec znaků

TSP travelling salesman problem

txt textový soubor

VRP vehlice routing problém

xls excel soubor

# SEZNAM ILUSTRACÍ

[Obr. 2.1 – Tvorba nové populace 17](file:///C:\Users\cen65065\Desktop\Aktualni%20prace_BP\MisekL1.docx#_Toc39593052)

[Obr. 2.2 – Reprezentace jedince 1 18](#_Toc39593053)

[Obr. 2.3 – Reprezentace jedince 2 18](#_Toc39593054)

[Obr. 2.4 – Reprezentace jedince 3 18](#_Toc39593055)

[Obr. 2.5 – Nová reprezentace jedinců 19](#_Toc39593056)

[Obr. 2.6 – Nová reprezentace jedince 1 19](#_Toc39593057)

[Obr. 2.7 – Nová reprezentace jedince 2 19](#_Toc39593058)

[Obr. 2.8 – Nová reprezentace jedince 3 19](#_Toc39593059)

[Obr. 2.9 – Zkrácený zápis jedinců 20](#_Toc39593060)

[Obr. 2.10 – Příklad ohodnocení jedinců 20](#_Toc39593061)

[Obr. 2.12 – Varianty selekce podle rulety 22](file:///C:\Users\cen65065\Desktop\Aktualni%20prace_BP\MisekL1.docx#_Toc39593062)

[Obr. 2.11 – Selekce ruletou 22](file:///C:\Users\cen65065\Desktop\Aktualni%20prace_BP\MisekL1.docx#_Toc39593063)

[Obr. 2.13 – Selekce pomocí turnaje 23](file:///C:\Users\cen65065\Desktop\Aktualni%20prace_BP\MisekL1.docx#_Toc39593064)

[Obr. 2.14 – Nadbytečné informace v chromozomu 23](#_Toc39593065)

[Obr. 2.15 – Příklad křížení jedinců 24](#_Toc39593066)

[Obr. 2.16 – Příklad křížení jedinců 24](#_Toc39593067)

[Obr. 2.17 – Získání nového genu mutací 25](#_Toc39593068)

[Obr. 2.18 – Příklad mutace jedince 25](#_Toc39593069)

[Obr. 2.19 – Příklad N dam 27](file:///C:\Users\cen65065\Desktop\Aktualni%20prace_BP\MisekL1.docx#_Toc39593070)

[Obr. 3.1 – Blokové schéma algoritmu 31](file:///C:\Users\cen65065\Desktop\Aktualni%20prace_BP\MisekL1.docx#_Toc39593071)

[Obr. 3.2 – Blokové schéma logiky algoritmu 31](file:///C:\Users\cen65065\Desktop\Aktualni%20prace_BP\MisekL1.docx#_Toc39593072)

[Obr. 3.3 – Vnitřní populace 32](file:///C:\Users\cen65065\Desktop\Aktualni%20prace_BP\MisekL1.docx#_Toc39593073)

[Obr. 3.4 – Vnější populace 33](file:///C:\Users\cen65065\Desktop\Aktualni%20prace_BP\MisekL1.docx#_Toc39593074)

[Obr. 3.5 – Pseudokód algoritmu 34](#_Toc39593075)

[Obr. 3.6 – Neoptimální cesta 35](file:///C:\Users\cen65065\Desktop\Aktualni%20prace_BP\MisekL1.docx#_Toc39593076)

[Obr. 3.7 – Optimální cesta 35](file:///C:\Users\cen65065\Desktop\Aktualni%20prace_BP\MisekL1.docx#_Toc39593077)

[Obr. 3.8 – Optimální skupina měst s optimální cestou 36](file:///C:\Users\cen65065\Desktop\Aktualni%20prace_BP\MisekL1.docx#_Toc39593078)

[Obr. 3.9 – Optimální skupina měst s neoptimální cestou 36](file:///C:\Users\cen65065\Desktop\Aktualni%20prace_BP\MisekL1.docx#_Toc39593079)

[Obr. 4.1 – Výpočet vzdálenosti mezi městy 38](#_Toc39593080)

[Obr. 4.3 – Nejrychlejší cesta 40](file:///C:\Users\cen65065\Desktop\Aktualni%20prace_BP\MisekL1.docx#_Toc39593081)

[Obr. 4.2 – Nejkratší cesta 40](file:///C:\Users\cen65065\Desktop\Aktualni%20prace_BP\MisekL1.docx#_Toc39593082)

[Obr. 4.4 – Programová reprezentace vnitřního jedince 41](#_Toc39593083)

[Obr. 4.5 – Vnitřní jedinec, třída Invidual 42](#_Toc39593084)

[Obr. 4.6 – Původní podoba vnějšího jedince, třída InvidualOrders 45](#_Toc39593085)

[Obr. 4.7 – Nová podoba vnějšího jedince, třída SuperInvidual 46](#_Toc39593086)

[Obr. 4.8 – Třída Functions 47](#_Toc39593087)

[Obr. 4.9 – Třída Cities 48](#_Toc39593088)

[Obr. 4.10 – Třída Controller 49](#_Toc39593089)

[Obr. 5.1 – Ukázka výstupního souboru 51](#_Toc39593090)

[Obr. 5.2 – Průběh parametru fitness vnějších jedinců během měření 52](#_Toc39593091)

[Obr. 5.3 – Vývoj fitness nejlepšího jedince během měření Vnejsi20Vnitrni50 53](#_Toc39593092)

[Obr. 5.4 – Vývoj fitness nejlepšího jedince během měření Vnejsi50Vnitrni20 53](#_Toc39593093)

[Obr. 5.5 – Vývoj fitness nejlepšího jedince během měření Vnejsi50Vnitrni50 54](#_Toc39593094)

[Obr. 5.6 – Srovnání měření fitness pro nejlepší jedince ze všech měření 54](#_Toc39593095)

[Obr. 5.7 – Průběh parametru Fitness během měření s vyrovnanou cenou 56](#_Toc39593096)

[Obr. 5.8 – Parametr Fitness pro měření s vysokou cenou za kilometr a nízkou za auto 57](#_Toc39593097)

[Obr. 5.9 – Parametru Fitness pro měření s nízkou cenou za kilometr a vysokou za auto 58](#_Toc39593098)

[Obr. 5.10 – Vývoj parametru Fitness během měření s městy daleko od sebe 60](#_Toc39593099)

# ÚVOD

Práce se zabývá problematikou optimalizace v logistice. Logistikou je možné rozumět proces, kdy je nutné přemístit předmět z jednoho místa do dalšího. Tento proces je možné optimalizovat hned v několika ohledech. Tato práce se zabývá zejména optimalizací celkové ujeté vzdálenosti.

Kapitoly v této práci jsou strukturovány do podoby, která umožní čtenáři proniknout do problematiky, představí variantu možného řešení, provede čtenáře analýzou problému, aplikuje navržené řešení, provede měření a zhodnotí výsledky.

Očekává se, že čtenář bude mít základní znalosti oblasti logistiky a základům objektově orientovaného programování. Vybraným jazykem pro realizaci aplikace je programovací jazyk C#.

První kapitola je obecným pohledem na problémy, které se mohou v logistice objevit. Zároveň jsou zde obecný úvod do problematiky. Druhá kapitola se zabývá rozborem genetických algoritmů, které je možné k řešení optimalizace použít a byly v tomto případě použity k řešení problému.

Třetí kapitola je zaměřena na analýzu zadání logistického problému a je zde navržen model a algoritmus, který bude použit k vyřešení logistického problému, který bude následně optimalizován. Čtvrtá kapitola popisuje programové řešení celé aplikace.  V této kapitole jsou popsány metody, které byly použity. Také je zde zdůvodněno, proč tyto metody byly použity.

Pátá kapitola obsahuje popis měření, která byla použita pro ověření navržené aplikace. Zároveň je možné zde nalézt grafy, které shrnují výsledky měření. Poslední částí je závěr, kde je celá práce zhodnocena.

# SEZNÁMENÍ SE S PROBLEMATIKOU

Problematika hledání vhodné kombinace řešení, ať už je to rozložení palet do kamionu, tvorba rozvrhu pro školu, nebo dosazení doktorů na služby, je v dnešní době řešena specializovanými pracovníky. Tito pracovníci používají rozličné metody k nalezení těchto řešení.

Jednou takovou metodou je takzvaná hrubá síla. V tomhle případě jsou systematicky generovány vhodné kombinace, které jsou následně vyhodnocovány. Tento postup přesto, že funguje, tak není optimální z hlediska časové náročnosti. Zejména protože čas pro nalezení vhodné kombinace se exponenciálně zvyšuje s rostoucími vstupy.

Tyto vstupy jsou konkrétní požadavky zadavatelů. Aleš Horký v jeho práci zmiňuje dva druhy omezení. Takzvaná tvrdá omezení a měkká omezení. Tvrdá omezení musí být splněna vždy. Měkká omezení nemusí být nutně splněna. Nicméně jejich splnění pozitivně ovlivní ohodnocení kombinace pomocí hodnotící funkce.

Mezi tvrdá omezení patří základní podmínky, jako na jedno paletové místo musí být dosazena pouze jedna paleta, kamion nemůže obsahovat více palet, než je jeho kapacita, a skladník nemůže nakládat více než jeden kamion v daný okamžik. Splněním těchto požadavků dosáhneme správného řešení. Toto řešení může být jediné, zároveň jich může existovat více než jedno a zároveň nemusí být ani jedno.

Měkká omezení jsou taková, která nemusí být splněna. Nicméně jejich splnění může negativně penalizovat ohodnocení výsledné kombinace. Může to být například zvláštní požadavek skladníka, který chce nakládat pouze některé kamiony, palety, které mohou být naloženy pouze do některých kamiónů. Některé kamiony mohou být prioritní a je nutné je naložit dříve a podobně.

Vyhodnocením těchto vstupů získáme kombinaci, která může být správným řešením. Pokud kombinace vyhoví všem tvrdým podmínkám, tak se jedná o správné řešení. Pokud nevyhoví, tak se jedná o nesprávné řešení. Zároveň je kombinace pomocí ohodnocující funkce hodnocena a je jí přiřazena hodnota, která udává, jak vhodná je kombinace. Vyšší fitness skóre představuje vhodnější řešení.

Kombinaci správných řešení získáme náhodným vygenerování. Po vyhodnocení kombinace je možné hledání dalších kombinací ukončit, nebo pokračovat. V případě pokračování je další generace nových kombinací vytvořena s ohledem na ty nejúspěšnější kombinace. Zároveň jsou kombinace náhodným způsobem nepatrně pozměněny. Tato změna za účelem nezaseknutí se v lokálním extrému.

## Rozbor zadání

Úkolem práce je vyzkoušet možnosti a využití optimalizace pro řešení vybraného logistického problému. Logistickým problémem je možné rozumět proces, kde je nutné přesunout věci z místa A do místa B, případně libovolného množství dalších míst. Hlavním účelem optimalizace potom může být snížit náklady, zvýšit kapacitu, urychlit plánování, nebo zjednodušit proces přesunu.

Hlavním předmětem této práce bude pokusit se optimalizovat řešení VRP (vehicle routing problem). To znamená snížit celkové náklady potřebné pro přepravu předmětů. Hlavními parametry budou celková ujetá vzdálenost a počet dopravních prostředků. Vedle těchto dvou parametrů existuje mnoho dalších, které ovlivňují celý proces.

Parametry jako časová okna, kdy může být předmět dodán, nebo typ dopravního prostředku, který je schopen předmět přepravit, nebudou brány v úvahu. Výstupem této práce je algoritmus, nebo postup, který je možné aplikovat v reálné situaci.

## Rešerše literatury

Podobným problémem se zabýval Aleš Horký (Horký, 2011), v práci „Tvorba rozvrhů pomocí genetických algoritmů“. Úkolem této práce bylo vytvořit program, který umožní uživateli vytvořit rozvrhy pro školu. Tyto rozvrhy jsou tvořeny s ohledem na mnohé požadavky. Například v jedné třídě nemůže být více vyučovaných hodin současně, jeden učitel v danou chvíli může učit pouze jednu třídu, žáci nemohou mít více než jednu hodinu současně a podobně.

Důvodem ke vzniku tohoto problému je komplexnost řešení. Pro požadavky, které jsou omezeny pouze několika podmínkami je řešení poměrně triviální. Například škola se čtyřmi učiteli, čtyřiceti žáky, jednou místností a čtyřmi vyučovacími předměty, kde každý učitel vyučuje právě jeden předmět po dobu jedné hodiny. Zde jsou žáci rozmístěni do jedné třídy a postupně je učí jeden učitel. Zde existuje mnoho správných řešení.

Problém nastane ve chvíli, kdy učitelé potřebují vyučovat hodiny v konkrétních třídách a mohou učit pouze v omezeném časovém úseku. Potom nelze předměty rozmístit libovolně jako v předchozím příkladu. Nemůžeme použít třídu, ve které je vyučován konkrétní předmět k výuce jiných předmětů. Řešení je stále poměrně jednoduché. Problém nastane ve chvíli, kdy vzroste počet vyučovaných hodin, učitelů, učeben a vzniknou zvláštní podmínky, za kterých mohou být předměty vyučovány. Aleš Horký tuto problematiku podrobně popisuje v jeho práci (Horký, 2011).

## Třídy složitosti NP a P

Informace v kapitole jsou založeny na článku „The Aged P versus NP Problem“ z webu „Towards data science“ (Hledání nejkratší cesty v grafu, 2020). Každé řešení úlohy je možné rozdělit na několik fází. Zaměřme se na fáze nalezení řešení a ověření řešení. Nalezení řešení i ověření řešení jsou vlastně algoritmy. Tyto algoritmy jsou posloupnosti instrukcí, které řešitel provede, aby dosáhl výsledku. Přitom doba k provedení algoritmu se může lišit. Z toho důvody byly zavedeny třídy složitosti.

Třídou složitosti P je možné rozumět, že algoritmus je možné vyřešit v polynomiálním čase. Příkladem takového problému je sčítání, násobení, případně nalezení největšího čísla v řadě.

Třída složitosti NP je potom algoritmem, který je možné vyřešit v polynomiálním čase nedeterministickým počítačem. Jinak řečeno je možné NP problém převést na P problém, pokud máme k dispozici nekonečné množství počítačů, které budou problém počítat paralelně. Příkladem jsou problémy zvané TSP a VRP.

## TSP (Traveling salesman problem)

Informace v této kapitole vychází z práce Ondřeje Míči (Míča, 2016), která pojednává o problému obchodního cestujícího. Tento problém se anglicky nazývá Travelling salesman problém a dále je na něj odkazováno zkratkou TSP.

Jedná o úlohu, kde je hledána nejkratší vzdálenost mezi několika body na mapě. Vstupem této úlohy je množina měst a očekávaným výstupem je posloupnost měst, která představuje pořadí, ve kterém budou města navštívena.

Zásadním problémem úloh typu TSP je doba k nalezení řešení. Počet vstupních měst může být v řádu jednotek až v řádu stovek či tisíců. Doba k nalezení řešení způsobem vygenerování všech možných pořadí a následně jejich vyhodnocení je přímo úměrná faktoriálu počtu vstupních měst mínus jedna. Tedy když písmeno *n* bude označovat počet měst, tak celkový počet kombinací bude (*n*-1)!. Těm nejmodernějším počítačům v dnešní době by trvalo nalezení řešení tímto způsobem velmi dlouho, potenciálně až několik let. Jedná se tedy o problém o NP složitosti. Příklad řešení takového algoritmu je uveden v příručce Google OR-Tools (Traveling Salesman Problem, 2020). Řešení spočívá ve využití matice vzdáleností pro vstup celkem třinácti měst. Výstupem je nejkratší vzdálenost mezi těmito městy. Reálnějším příkladem je nalezení nejkratší vzdálenosti pro vrták, který vrtá díry do desky plošných spojů ze stejné příručky.

## VRP (Vehlice routing problem)

Tato kapitola byla vytvořena na základě informací v příručce Google OR-Tools (Traveling Salesman Problem, 2020). Vehlice routing problem zkráceně VRP je velice podobný problému TSP. Pokud by v případě neexistovaly omezující podmínky, tak by bylo možné problém zjednodušit na TSP, protože by jeden řidič rozvezl všechny předměty. Optimalizace v tomto případě znamená eliminovat ty nejdelší cesty z tras řidičů.

Příkladem může být logistické depo, které potřebuje rozvést určitý počet předmětů do měst, které jsou nahodile rozmístěny na mapě, a depo by bylo uprostřed. Nejdelší cesty by potom byly ty, které by vedly ze severu na jih přes depo. Vhodnější by bylo poslat jednoho řidiče do severních měst a jednoho do jižních měst. Odpadla by potom dlouhá cesta ze severu na jih.

## Řešení VRP pomocí GA

Hlavním úskalím pro řešení TSP je nalezení trasy s nejkratší vzdálenosti spojující několik měst. Na rozdíl od TSP v případě VRP bude těchto tras dva a více. Jako vybraným prostředkem pro řešení VRP je v této práci genetický algoritmus (GA).

Řešení VRP pomocí GA bude popsáno v následujících kapitolách včetně finálních výsledků. Jako testovací vzorek poslouží souřadnice reálných měst a jejich vzorek zároveň bude simulovat reálný požadavek dopravní společnosti, která se zabývá rozvážkou zboží.

Během tvorby řešení nebudou zohledněny nereálné situace, nebo výjimečné stavy, které běžně nenastávají, nebo k nim nemůže dojít. Například požadavek zákazníka na přepravu objemu stovek až tisíců palet se zbožím, přeprava kombinovaná s námořní, nebo leteckou dopravou. Stěžejní pro práci bude vyzkoušet aplikaci GA na VRP.

# GENETICKÉ ALGORITMY

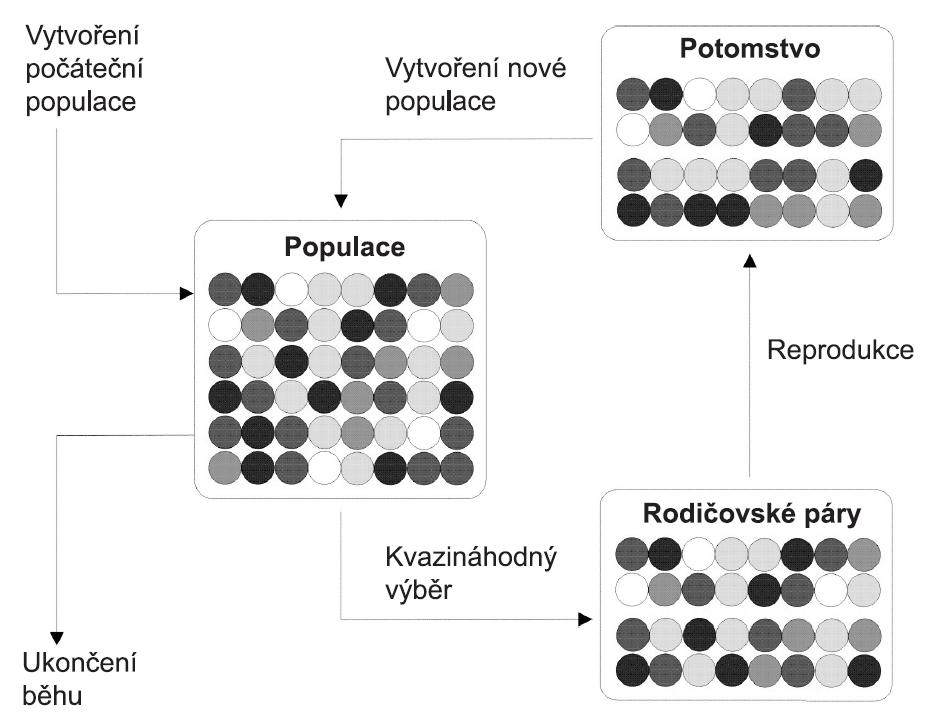
V této kapitole jsou popsány genetické algoritmy, jejich návrh, programování a princip. Tato kapitola se opírá o již vydané publikace. Zejména o publikaci s názvem Genetické algoritmy a genetické programování od Josefa Hynka (Hynek, 2008).

Tato kniha posloužila jako dobrý nástroj pro úvod do problematiky a nastiňuje jejich aplikaci. V knize jsou podrobně popsány postupy a principy, které se používají při tvorbě genetických algoritmů.

V této práci jsou popsány pouze ty postupy, které byly použity v realizaci algoritmu. Obsáhlejší popis genetických algoritmů, kódování a operací je popsán v knize Genetické programování a genetické algoritmy, ze které tato práce vychází.

## Genetický algoritmus

Genetický algoritmus vychází z myšlenky Charlese Darwina (Hynek, 2008), že populace živočichů a rostlin se vyvíjela po mnoho generací. Každá populace je tvořena jedinci, kde každý jedinec má své charakteristické vlastnosti. Tito jedinci soupeří navzájem o prostředky a k rozmnožovacímu procesu se dostanou pouze ti nejsilnější. Tito nejsilnější jedinci se stávají rodiči a vytvoří novou generaci, která ponese jejich vlastnosti. Pro tuto novou generaci potomků se proces opakuje a do každé nové generace se dostávají vlastnosti těch nejsilnějších. Proces tvorby nových generací je zobrazen na obr. 2.1.



Obr. ‑ – Tvorba nové populace (Hynek, 2008, s. 14)

## Reprezentace jedince

Každý jedinec v populaci může být reprezentován jeho specifickými vlastnostmi (Hynek, 2008). Každý jedinec má své vlastnosti, které ho reprezentují. V případě TSP, nebo VRP by vlastnosti jedinců byly sekvence měst, které řidič musí projet. Příklad může být vidět na obr. 2.2 a obr. 2.3.

{  
 "Název": "Rodič\_1",  
 "Sekvence":  
 [ "Pardubice", "Hradec Králové", "Přelouč", "Kolín", "Chrudim" ]  
}

Obr. 2‑2 – Reprezentace jedince 1

{  
 "Název": "Rodič\_2",  
 "Sekvence":  
 [ "Hradec Králové", "Přelouč", "Kolín", "Pardubice", "Chrudim" ]  
}

Obr. 2‑3 – Reprezentace jedince 2

Tito dva jedinci se následně dostali do rozmnožovacího procesu a stali se rodiči. Nyní v další generaci byl vytvořen další jedinec tedy potomek. Potomek nese vlastnosti jeho rodičů. Zde může být vidět potomek, jehož vlastností je kombinace vlastností jeho rodičů. Potomek dvou rodičů je zobrazen na obr. 2.4.

{  
 "Název": "Potomek",  
 "Sekvence":  
 [ "Pardubice", "Hradec Králové", "Kolín", "Pardubice", "Chrudim" ]  
}

Obr. 2‑4 – Reprezentace jedince 3

Výčet vlastností každého jedince se nazývá chromozom (Hynek, 2008). Přitom každá jednotlivá vlastnost je genem. Pro přehlednost jednotlivé geny budou označeny číslicemi, které budou následně použity dále. Zaveďme reprezentaci jako na obr. 2.5.

"Hradec Králové": 1 "Rodič\_1": 1  
"Přelouč": 2 "Rodič\_2": 2  
"Chrudim": 3 "Potomek": 3  
"Kolín": 4  
"Pardubice": 5

Obr. 2‑5 – Nová reprezentace jedinců

Původní jedinci budou tedy vypadat jako na obr. 2.6, obr. 2‑7 a obr. 2‑8.

{  
 "Název": "1",  
 "Sekvence":  
 [ 5, 1, 2, 4, 3 ]  
}

Obr. 2‑6 – Nová reprezentace jedince 1

{  
 "Název": "2",  
 "Sekvence":  
 [ 1, 2, 4, 5, 3]  
}

Obr. 2‑7 – Nová reprezentace jedince 2

{  
 "Název": "3"  
 "Sekvence":  
 [ 5, 1, 4, 5, 3 ]  
}

Obr. 2‑8 – Nová reprezentace jedince 3

Všechny tři jedince, kteří aktuálně tvoří populaci, je možné zapsat jako na obr. 2‑9.

Jedinec Chromozom

1 {5, 1, 2, 4, 3}  
 2 {1, 2, 4, 5, 3}  
 3 {5, 1, 4, 5, 3}

Obr. 2‑9 – Zkrácený zápis jedinců

## Fitness jedince

Další otázkou je, jak vybrat ty nejsilnější jedince. Z tohoto důvodu je nutné jedince ohodnotit. Každému genu v chromozomu jedince bude přiděleno „skóre“, které bude určovat jak silný jedinec ve skutečnosti je (Hynek, 2008). Na obr. 2.10 je příklad takového ohodnocení.

Jedinec Chromozom Fitness

1 {5, 1, 2, 4, 3} 20  
 2 {1, 2, 4, 5, 3} 25  
 3 {5, 1, 4, 5, 3} 30  
 4 {1, 3, 2, 1, 1} 20  
 5 {5, 4, 3, 2, 1} 15  
 6 {2, 2, 2, 2, 2} -5  
 7 {1, 3, 2, 5, 4} 20

Obr. 2‑10 – Příklad ohodnocení jedinců

Z obr. 2.10 je vidět, že jedinec 3 dosáhl nejvyššího ohodnocení a jedinec 6 dosáhl nejnižšího ohodnocení. Pokud by se rozmnožovacího procesu měli účastnit pouze ti nejsilnější jedinci, tak by se jednalo o jedince 3 a 2, kteří by se podíleli na tvorbě nové generace.

Ohodnocení jedince může být počítáno mnoha způsoby. V příkladu byla použita jako fitness vzdálenost mezi městy a navíc počet duplicit. Z toho důvodu jedinec 6 dopadl jako nejhorší. Otázkou také je, jak velký význam na fitness budou mít jednotlivé položku. Pokud budou mít duplicity příliš vysokou váhu, tak se může stát, že jedinec s duplicitou, který má jinak dobré řešení bude naprosto vyřazen.

## Selekce

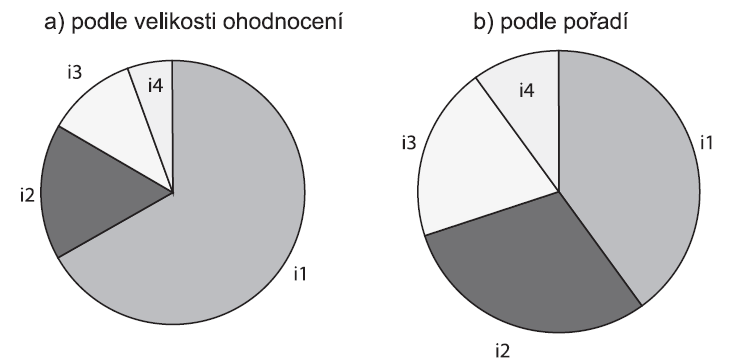
Selekce je způsob, kterým jsou vybírání jedinci do další generace (Hynek, 2008). Většinou se používá fitness funkce, podle které jsou označeni ti nejvhodnější. Vzhledem k tomu, že je algoritmus do určité částí řízen náhodou, tak nelze vyloučit, že se novou generaci budou vytvářet slabší jedinci.

Tvorba nové generace ze slabších jedinců není ve skutečnosti negativním jevem. Silný jedinec může obsahovat hodně pozitivních genů, díky kterým má vysokou fitness skóre, ale nemusí obsahovat pozitivní gen slabšího jedince. Pokud by byl slabý jedinec vyřazen z procesu tvorby nové generace, tak by se tato vlastnost ztratila.

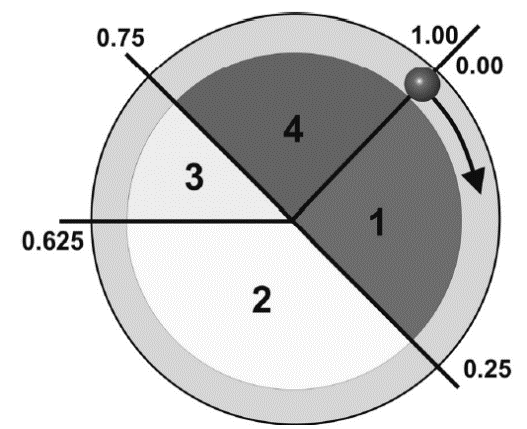
Ve skutečnosti by to znamenalo, že by silný jedinec uvízl v lokálním extrému. Proto existují způsoby, které dávají šanci i těm slabším jedincům. Různé způsoby selekce mají různé výhody a nevýhody.

Existuje například selekce pomocí rulety, jako je na obr. 2.11. Zde je fitness skóre přímo úměrné ploše na ruletě. V tomto případě je velmi vysoká pravděpodobnost, že jedinec s vysokým fitness skóre (2) se neztratí a jedinec s nízkým fitness skóre (3) má také určitou šanci na úspěch. Nevýhodou je, že příliš silný jedinec (2) může zabrat dominantní postavení a žádný jiný jedinec nebude připuštěn k rozmnožovacímu procesu.

Ruletu je možné pozměnit, aby plocha nebyla přímo úměrná velikosti fitness skóre, ale pořadí. Potom bude ruleta vypadat jako na obr. 2.12. Zde má slabší jedinec vyšší šanci na podílení se na tvorbě nové generace. Zároveň jsou silnější jedinci stále zvýhodněni díky jejich dobrým genům.



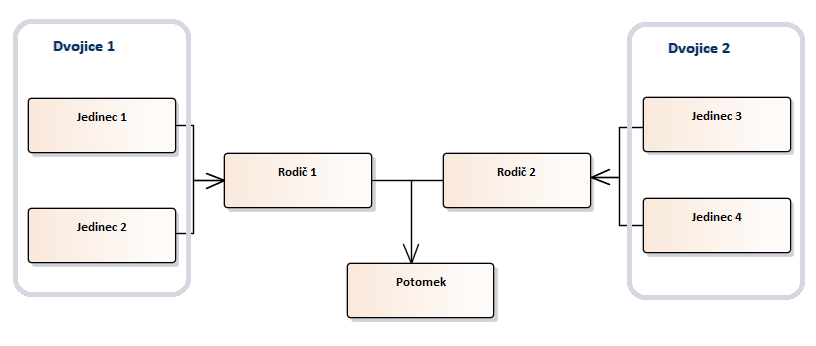
Obr. ‑ – Varianty selekce podle rulety (Hynek, 2008, s. 47)



Obr. ‑ – Selekce ruletou   
(Hynek, 2008, s. 23)

Dalším způsobem selekce je selekce pomocí turnaje. Příklad selekce pomocí turnaje je náhodné vybráni 2 dvojic jedinců a z těchto dvojic jsou vybráni 2 vítězové. Vítězi se následně budou podílet na tvorbě nové generace. Pokud tedy vybereme dvojici 2 slabších a 2 silnějších jedinců, tak rodiči se stane 1 silný a 1 slabý jedinec. Nevýhodou je, že některý silný jedinec nemusí být vybrán do turnaje právě kvůli náhodě.

Příklad turnaje je zobrazen na obr. 2.13.



Obr. ‑ – Selekce pomocí turnaje

## Genetické operátory

Genetické operátory jsou funkcemi, které lze použít k práci s jedinci (Hynek, 2008). Návrh těchto operací je velice důležitý. V případě této práce se jedná o křížení dvou jedinců a o mutace jedince.

Nicméně někdy není možné vhodně navrhnout genetický operátor, protože samotný jedinec je nerozumně, nebo nedostatečně reprezentován. Například na obr. 2.14 jsou křížení dva jedinci. Ve vlastnostech jsou navíc informace, které neslouží k výpočtu fitness a jsou pouze informativního charakteru, nebo jsou použity během vývoje. Takové informace navíc mohou podstatně zpomalit chod algoritmu a musí na ně být brán ohled během křížení.

Jedinec Chromozom

1 {5, 1, 2, 4, 3, "Generace\_1", "2020-04-16", „Highscore“}  
 2 {1, 2, 4, 5, 3, "Generace\_1", "2020-04-16", „-„}

3 {5, 1, 4, 5, 3, "Generace\_2", "2020-04-17", „-„}

Obr. 2‑14 – Nadbytečné informace v chromozomu

## Křížení

Křížení je operace, která zkombinuje 2 jedince a z jejich genů vytvoří nového jedince (Hynek, 2008). Tento nový jedinec bude kombinací jeho 2 rodičů z předešlé generace. Křížení je možné realizovat mnoha způsoby. Toto je hlavní způsob, kterým jsou tvoření noví jedinci v této práci. Po křížení se může stát potomek silnějším i slabším, než jsou jeho rodiče.

Může být ke křížení použita vždy konkrétní část jedinců. Například první polovina z prvního jedince a druhá polovina z druhého jedince. Tyto poloviny mohou být vybírány náhodně. Je možné každý gen vybírat náhodným způsobem z jedinců. Například na obr. 2‑15 je zobrazeno křížení, kde potomek dědí část vlastností z každého rodiče.

Jedinec Chromozom

1 {5, 1, 2, 4, 3}  
 2 {1, 2, 4, 5, 3}  
  
 3 {5, 1, 4, 5, 3}

Obr. 2‑15 – Příklad křížení jedinců

Křížení je možné omezit pouze na některé gen uvnitř chromozomu. Kdyby dodatečné vlastnosti, které skutečně neovlivňují fitness, by bylo potřeba držet v chromozomu, tak je možné je z křížení vynechat jako na obr. 2‑16.

Jedinec Chromozom

1 {5, 1, 2, 4, 3, “Generace\_1“, “2020-04-16“, “Highscore“ }  
 2 {1, 2, 4, 5, 3, “Generace\_1“, “2020-04-16“, “-“}  
  
 3 {5, 1, 4, 5, 3, “Generace\_2“, “2020-04-17“, “-“}

Obr. 2‑16 – Příklad křížení jedinců

## Mutace

Mutace je operace, kdy se jeden nebo více genů náhodně změní (Hynek, 2008). Touto náhodnou změnou může vzniknout gen, který dříve neexistoval. Mutace je způsob, kterým je algoritmus chráněn od lokálního extrému.

Jedinec Chromozom

8 {5, 1, 1, 2, 3}  
 9 {5, 1, 1, 2, 3}  
 10 {5, 1, 1, 2, 3}

9 {5, 1, 4, 2, 3}

Obr. 2‑17 – Získání nového genu mutací

Mějme reprezentaci jedinců jako na obr. 2‑17. Jedinci mají své sekvence měst a vypadá to, že algoritmus dosáhl extrému a dál již nepokračuje. Problém je ten, že žádný z jedinců neobsahuje hodnotu 4 a křížením už tuto hodnotu není možné získat. Z tohoto extrému je možné uniknout mutací, a to právě tak, že se po přechodu do další generace náhodně změnil jeden z genů v chromozomu.

Další výhodou je, že se může objevit gen, nebo chromozom, který může vést k mnohem silnějším a vhodnějším jedincům. Po mutaci na obr. 2‑18 sice vznikl gen s duplicitní hodnotou, ale později se ukázalo, že vzdálenost mezi městy 3 a 1 je mnohem menší, než jiné kombinace.

Jedinec Chromozom

8 {5, 1, 1, 2, 3}  
 9 {5, 1, 1, 2, 3}  
 10 {5, 1, 1, 2, 3}

10 {3, 1, 1, 2, 3}

Obr. 2‑18 – Příklad mutace jedince

Zároveň je důležité, aby nebyl výskyt mutace příliš vysoký. Algoritmus by následně mohl být příliš náhodný a nikdy by nemusel dojít do extrému.

## Příklad skládání puzzle

Uvažujme, že máme puzzle, které má celkem 1000 dílků. To znamená, že máme 1000 možných pozic, na který je možné každý dílek umístit. Podle vzorce pro variace (Variace, 2006) s opakováním je to celkem 10001000 možných kombinací a pouze jedna z nich je správná.

První možností je pokusit se řešit takové puzzle „hrubou silou“. V tom případě bychom museli všechny tyto možnosti vyzkoušet. Tohle by mohlo trvat potenciálně několik desítek let. Druhou možností je náhodně rozložit dílky po ploše a pokoušet se dílky spojovat. Na začátku by byl náhodně vybrán dílek a k tomu by byl hledán dílek, který zapadne. Tento proces by se opakoval, dokud by nebylo puzzle hotové.

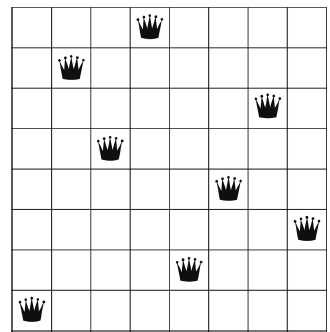
Je možné si představit řešení „hrubou silou“, jako poskládání všech 1000 dílků na plochu a vyhodnocení, jestli je puzzle správně poskládáno. Pokud by nebylo správně poskládáno, tak všechno bude přeskládáno znovu a proces by se opakoval, dokud není nalezeno správné řešení. Genetický algoritmus je jako nalezení částečně správného řešení, dílků, které k sobě patří a k těmto dílkům by byly hledány souhlasné dílky. Toto částečné správné řešení je následně použito k nalezení lepšího řešení, dokud by nebylo dosaženo správného řešení.

## Problém N dam

Dalším příkladem je problém N dam (Hynek, 2008). Mějme čtvercovou šachovnici, která konečný počet polí. Na tuto šachovnici chceme umístit figurky dámy, které nesmí být navzájem ve své dráze. Na obr. 2‑20 je příklad s šachovnicí 8x8, kde musí být rozmístěno 8 dam. Každá dáma se může pohybovat vertikálně, horizontálně, nebo po diagonále. Dámy nesmí být navzájem ve svých dráhách.

V případě 3x3 by bylo podle vzorci o variacích nutné projít 504 kombinací a tato hodnota by velice rychle rostla. Pro případ 10x10 by existovalo 62815650955529470000 kombinací.

Je jisté, že případ by nebylo možné řešit „hrubou silou“, protože by nalezení řešení trvalo opět několik let. Jedná se o typický příklad, který je možný řešit genetickým algoritmem. Podrobně popsáno v publikaci (Hynek, 2008).



Obr. ‑ – Příklad N dam (Hynek, 2008, s. 92)

# NÁVRH ALGORITMU

Tato kapitola se věnuje tomu, jak byl problém rozebrán a jak bylo navrženo řešení. Řešení je v tomto případě několik a ne všechny se ukázaly jako vhodné. Mezi hlavní parametry, kterými byl algoritmus hodnocen, jsou výsledná posloupnost měst a realizovatelnost algoritmu po programové stránce.

## Úloha, problematika

Výhodou genetického algoritmu je jeho obecnost. Tento algoritmus je možné použít pro širokou škálu úloh. Tyto úlohy mohou být hledání nejkratší vzdálenosti, rozmístění pracovníků ke strojům, naplánování rozvrhu a mnoho dalších podobných úloh.

V tomto typu úloh může existovat mnoho řešení. Například pokud je nutné vytvořit školní rozvrh, tak vždy se potýkáme s omezujícími podmínkami, které ovlivňují, kdy je možné předmět učit. Mezi tyto podmínky může patřit počet učeben, počet vyučovaných předmětů, počet vyučujících, speciální časy, ve které je možné předměty učit, možnost učit více tříd současně, nutnost učit pouze určité dny a mnoho dalšího.

Jednou z možností, jak naplánovat takový rozvrh je zkoušet předměty nahodile poskládat a následně dobré kombinace jenom ladit. Tímto způsobem je možné dosáhnout rychle výsledku. Ladění je ve skutečnosti k částečně správnému řešení zkoušet přidávat chybějící předměty.

Další možností je možné zkoušet poskládat rozvrh hrubou silou a zkusit všechny možnosti, které jsou. Tohle ve skutečnosti funguje, ale se zvyšujícím se počtem omezujících podmínek, vstupů a výstupu se počet kombinací exponenciálně zvyšuje. Tento jev je zvaný kombinatorická exploze a je to zároveň důvod, proč není vhodné tímto způsobem hledat řešení. Velice výkonný počítač by potřeboval několik let pro nalezení řešení tímto způsobem.

Genetický algoritmus umožňuje využít toho, že součástí každého řešení je část vhodného řešení. Je možné vygenerovat skupinu řešení náhodně, ohodnotit je pomocí podmínek a z těch nejlepších poskládat řešení, které jsou jejich kombinacemi. Po několika desítkách kombinací je možné tímto způsobem získat vhodné řešení, které by hrubou silou trvalo několik let.

## Rozbor zadání

Úkolem práce je navrhnout algoritmus, který přebere vstup, který reprezentuje požadavek zákazníka a jako výstup poskytne posloupnost měst. Tato posloupnost měst představuje cestu, kterou řidič musí projít.

Tato cesta má reálnou vzdálenost a vhodným řešením se rozumí takové, které se přibližuje k nejkratší vzdálenosti. V případě nalezení vhodné trasy pro řidiče je vhodným řešením takové řešení, kde se řidič zbytečně nevrací, a města jsou seskupena do skupin.

Vstupy budou omezeny na případy, kdy jeden řidič není schopen všechno zboží na jednu cestu naložit, a proto musí být zboží rozděleno mezi několik řidičů. Díky tomu existuje nekonečně mnoho možností, jak zboží rozdělit.

## Návrh řešení

Řešením by tedy měl být program, který je schopen přečíst seznam předmětů, které je nutné rozvést, tento seznam analyzovat a vytvořit vhodnou posloupnost měst, kterou vrátí jako řešení. Poskytnuté řešení by mělo vyhovět požadavkům zadaným zákazníkem. Dále by mělo poskytnuté řešení umožnit uživateli více než jednu možnost, případně předložené řešení modifikovat.

Tyto požadavky by měly být strukturovány do formátu, který je možné v elektronické formě uložit do souboru. Aplikace na základě tohoto programu stanoví požadavky uživatele. Tyto soubory by měly být navrženy takovým způsobem, že je možné je vytvořit, nebo naplnit daty běžným uživatelem, nebo analytikem.

Uživateli bude umožněno upravit dvě skupiny parametrů a vstupní data. První skupinou jsou parametry, které přímo ovlivňují výsledek.

Do první skupiny patří tyto parametry:

* Cena za kilometr.
* Cena za dopravní prostředek.

Druhá skupina představuje parametry, které ovlivňují genetický algoritmus.

* Počet jedinců v generacích.
* Počet generací.
* Ukončující podmínka.

Vstupy jsou tvořeny dvěma soubory. Tyto soubory nesou informace o:

* Seznam požadavků zákazníků
* Seznam souřadnic měst

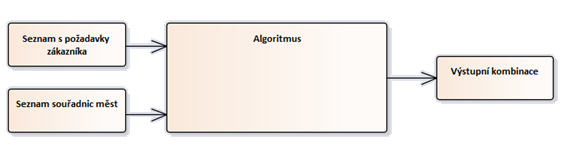
Aplikace na základě požadavků vygeneruje výslednou kombinaci. Uživatel musí mít k dispozici ovládací prvky, které mu umožní vybrat z výsledných řešení. V případě, že žádné řešení neexistuje, tak bude vrácena taková kombinace, která je nejblíže k vhodnému řešení. Uživatel by potom mohl dořešit problém nestandardním řešením.

Výstup aplikace bude řešen pomocí csv, nebo xls souboru. Tento soubor je potom dále možné použít v jiných aplikacích, které slouží k zadání samotného rozvrhu. Případně může být výstup ve formátu jpg, bmp, nebo pdf, které je možné vytisknou. Tato možnost by mohla být požadována uživateli, kteří nepoužívají žádné výpočetní programy.

Aplikace bude realizována v programovacím jazyce C#.

## Blokové schéma

Vstupem aplikace budou soubory s požadavky ve formátu txt, csv, nebo xls. První vstupním souborem je seznam s požadavky zákazníka. Druhým vstupním souborem je seznam souřadnic měst. Výstupním souborem je výstupní kombinace.



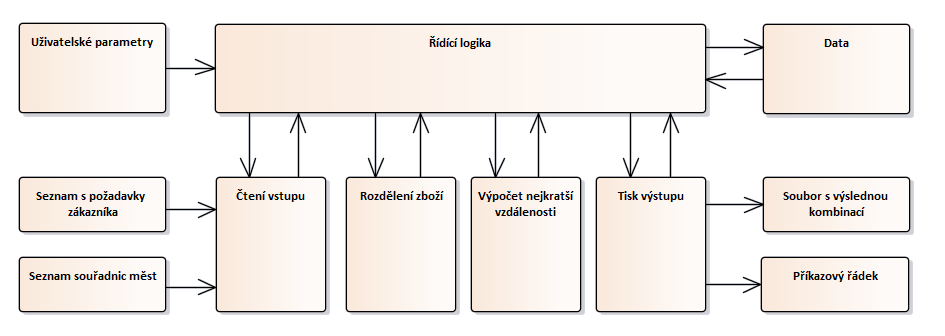
Obr. ‑ – Blokové schéma algoritmu

Samotný algoritmus může být rozdělen na několik bloků, které vykonávají konkrétní funkce. Bude potřeba číst vstup s požadavky zákazníka, rozdělit zboží na několik celků, vypočítat nejkratší vzdálenost a vytisknout posloupnost jako výstup.

Blok čtení vstupu bude sloužit pouze ke čtení vstupních souborů. Tento blok bude zároveň obsahovat funkce pro práci a dekódování vstupních souborů. Dále blok výstupu bude obsahovat funkce pro zobrazení dat uživateli. Příkazový řádek bude velice důležitý ve chvíli, kdy bude algoritmus vyvíjen a tisk do souboru bude dobrý pro analýzu výsledku.

Blok rozdělení vstupu bude sloužit pro rozdělení zboží na menší celky, které budou následně přiděleny řidičům. Zde budou všechny podmínky, které budou hlídat, že není překročena kapacita řidiče, a že jsou naloženy všechny zakázky.

Výpočet nejkratší vzdálenosti bude řešen pomocí genetického algoritmu. Existují jiné a specializovanější postupy, které je možné použít, ale pro účely této práce bude použit genetický algoritmus. Rychlost tohoto výpočtu bude velice důležitá, protože během chodu algoritmu vznikne mnoho kombinací, které musí být vyhodnoceny.



Obr. ‑ – Blokové schéma logiky algoritmu

## Komponenty algoritmu

### Vnitřní algoritmus

Výstupem vnitřního algoritmu je pouze jeden nejsilnější vnitřní jedinec, který dále tvoří vnějšího jedince. Tento výstupní jedinec představuje posloupnost měst, která představuje nejkratší nalezenou trasu pro danou množinu měst. Výstupní jedinec je zároveň nejsilnějším jedincem z vnitřní populace. Vnitřní populace je přitom tvořena množinou vnitřních jedinců. Počet těchto jedinců tvořící vnitřní populaci je dán parametrem a všichni jedinci ve vnitřní populaci jsou po několik generací zlepšování, dokud není nalezen nejsilnější.

### Vnitřní jedinec

Vnitřní jedinec představuje jednu posloupnost měst. Tuto posloupnost obslouží pouze jeden řidič a pořadí měst v posloupnosti představuje pořadí, ve kterých budou navštívena. Po spojení všech měst uvnitř posloupnosti vnitřního jedince vznikne trasa, která má reálnou vzdálenost. Tato vzdálenost představuje fitness vnitřního jedinec.

### Vnitřní populace

Vnitřní populace obsahuje konečný počet vnitřních jedinců. Tito vnitřní jedinci jsou podrobeni vnitřnímu genetickému algoritmu a vnitřní jedinec s nejnižší fitness funkcí (nejkratší vzdáleností) je označen za vítěze. Vnitřní populace je zobrazena na obr. 3‑3.



Obr. ‑ – Vnitřní populace

### Vnější algoritmus

Vnější populace obsahuje určitý počet vnějších jedinců a vnější jedinci obsahují konečný počet vnitřních jedinců. Každý vnitřní jedinec je výstupem vnitřního genetického algoritmu. Výstupem vnějšího genetického algoritmu je jeden nejsilnější vnější jedinec.

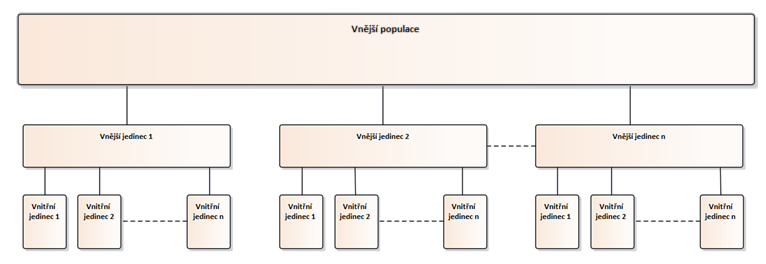
Jinými slovy lze říci, že vnější jedinec je skupina posloupností měst, kde každá posloupnost je jedním vnitřním jedincem. Tito vnější jedinci tvoří vnější populaci a jsou po generacích zlepšování vnějším genetickým algoritmem.

### Vnější jedinec

Vnější jedinec, tedy jedna skupina posloupností měst neboli vnitřních jedinců. Každý vnitřní jedinec představuje nejsilnějšího jedince, který prošel vnitřní genetickým algoritmem jako vítěz. Každý vnitřní jedinec má svou fitness a součet fitness všech vnitřních jedinců je fitness vnějšího jedince.

### Vnější populace

Vnější populace obsahuje konečný počet vnějších jedinců. Tito vnější jedinci navzájem soupeří a ten nejvhodnější z nich je označen jako vítěz a zároveň výstup algoritmu. Vnější populace je zobrazena na obr. 3‑4.



Obr. ‑ – Vnější populace

## Pseudokód algoritmu

Mějme tyto parametry:

* Počet vnitřních generací je 50
* Počet vnějších generací je 20
* Ukončující podmínka není stanovena
* Celková kapacita pro všechny předměty je vyšší než 32

Algoritmus je potom možné popsat pseudokódem na obr. 3‑5.

Načíst vstupy;  
 Načíst souřadnice;

Roztřídit předměty a vygenerovat populace;

for (1 ... 20) // Vylepšení vnější populace 20x

{

foreach (vnější jedinec ve vnější populaci)

{

Vylepšit vnějšího jedince;

{

// Vnější jedinec = skupina vnitřních jedinců

Vytvoření vnitřní populace ze vnějšího jedince;

foreach (vnitřní jedinec ve vnitřní populaci)

{

for (1... 50) // Vylepšení vnitřní populace 50x

{ Vylepšit vnitřního jedince }

}

}

Aktualizace nejlepšího vnějšího jedince;

}

Vrácení nejlepšího vnějšího jedince;

Obr. 3‑5 – Pseudokód algoritmu

## Možné výstupy algoritmu

Otázkou, která bude řešena, je, jestli je lepší hledat delší dobu tu nejkratší vzdálenost, nebo jestli je vhodnější hledat skupinu měst, která je více pohromadě. Je tedy nutné zjistit, jak dlouho má smysl optimální řešení hledat a jak velký dopad má každý scénář. Veškerá měření budou zaměřena na tyto hlavní body:

* Vliv počtu generací na algoritmy
* Vliv ceny za kilometr a řidiče na algoritmus
* Vliv rozmístění měst na algoritmus

### Města jsou nevhodně rozdělena na úrovni vnějšího jedince

Pokud by nebyly města pohromadě, tak by nejkratší vzdálenosti mohly vypadat takto. V obou případech je nejkratší vzdálenost nalezena, ale v případě na obr. 3‑6 je tato vzdálenost mnohem vyšší, než vzdálenost na obr. 3‑7.

Situace na obr. 3‑6 zjevně výrazně prodlouží celkovou trasu a nastává ve chvíli, kdy algoritmus uvízne v lokálním extrému na úrovni vnějšího jedince, ale počet generací vnitřního jedince je dostatečný.

Situace na obr. 3‑7 je optimální výstup algoritmu.

|  |  |
| --- | --- |
| Obr. ‑ – Neoptimální cesta | Obr. ‑ – Optimální cesta |

### Nebyla dosažena nejkratší vzdálenost

V tomhle případě se může stát, že řidič bude mít vhodně určenou oblast měst, ale algoritmus pro vnitřního jedince neběžel dostatečně dlouho a výsledná cesta není nejkratší. Tohle nemusí nutně vadit, protože se může jednat pouze o minimální nárůst cesty.

Na obr. 3‑8 jsou zvoleny ty nejkratší vzdálenost a na obr. 3‑9 jsou vzdálenosti o něco delší. Oba případy mohou být považovány za optimální řešení.

|  |  |
| --- | --- |
| Obr. ‑ – Optimální skupina měst s optimální cestou | Obr. ‑ – Optimální skupina měst s neoptimální cestou |

## Odhad optimálního řešení

Logistické společnosti v dnešní době rozdělují zboží, které chtějí rozvést na menší části. Například Olomoucký kraj je možné rozdělit na Šumperskou, Olomouckou, Bruntálskou, Přerovskou a Ústeckou část. Tyto části jsou blízko u sebe. To by odpovídalo případu, kdy jsou města vhodně rozdělena a jsou blízko u sebe. Nicméně pokud by logistická společnost nerozdělila zboží na části a řidiči by jeli nejkratší trasy, ale do měst, které nejsou poblíž, tak by mohlo dojít k značnému nárůstu konečné vzdálenosti.

V druhém případě by došlo pravděpodobně k mnohem většímu nárůstu konečné vzdálenosti, protože některá města mohou být od sebe desítky až stovky kilometrů. Například by to byl příklad, kdy by řidič jel z Brna do Prahy, pak do Ostravy, pak do Liberce a nakonec do Brna. Zde by došlo ke značnému nárůstu.

V prvním případě, kdyby řidiči obsluhovali pouze Pražský kraj, nebo jen Brněnský kraj, tak nejkratší vzdálenost nehrála tak zásadní roli, protože by necestovali přes celou republiku, ale udělali by si pouze zajížďku v délce několik kilometrů. Odhad je tedy takový, že důležitější vhodně rozdělit města na skupiny, které jsou poblíž a až potom hledat nejkratší vzdálenost.

# ŘEŠENÍ VLASTNÍ PRÁCE

Tato kapitola se zabývá vlastní realizací práce. Je zde popsáno, jakým způsobem byl samotný algoritmus vyvíjen. Jsou zde zároveň popsány problémy, ke kterým došlo a jejich následné řešení.

K realizaci programu byl použit programovací jazyk C# (C#, 1996). Vzhledem k charakteru úlohy nebylo nutné zpracovat pokročilé grafické rozhraní a výstup byl tisknut do příkazové řádky. Vstupem programu jsou txt soubory formátované jako csv soubory s oddělovači.

První vstupní soubor obsahoval souřadnice všech měst. Tento soubor se jmenoval Coordinates.txt a původně byl použit v projektu „souradnice-mest“ dostupného zdarma na stránkách githubu (Zemek, 2018). V souboru je každé město názvem a X a Y souřadnicemi.

Druhým vstupním souborem je soubor „ActiveOrders.txt“. Tento soubor obsahuje data, která představují požadavky zákazníků. Každý požadavek je označen unikátním identifikátorem ID a obsahuje cílové město a počet požadovaných palet.

## Krok Výpočet nejkratší vzdálenosti

Prvním krokem bylo navrhnout algoritmus pro výpočet nejkratší vzdálenosti. Vstupem tohoto algoritmu je textové pole, které obsahovaly města, mezi kterými byla nejkratší vzdálenost počítána. Prvním pokusem bylo počítat vzdálenosti pomocí Dijkstrova algoritmu (Kolář, 2000) a matice vzdáleností (Kunz, 2001).

### Matice vzdáleností

Matice vzdáleností byla použita, když byly města ještě v řádu jednotek. Matici bylo jednoduché vytvořit, ale se zvyšujícím se počtem měst se obtížnost značně zvýšila.

Tato matice má 2 hlavičky. Jedna hlavička je horizontální, druhá vertikální. Obě hlavičky obsahovaly názvy měst. Buňky, kde se města protínala, byla právě vzdálenost mezi městy. Na diagonále této matice jsou 0 a celá matice je symetrická podle diagonály.

Výhodou je, že matici je jednoduché použít. Nebylo nutné hledat v matici, kde jsou města a bylo možné rovnou číst hledanou vzdálenost. Například pro výpočet vzdálenosti mezi městem 12 a městem 23 stačilo přečíst hodnotu na 12. řádku a 23. sloupci.

Nevýhodou bylo, že pro požadovaný vstup 216 měst byla tato matice příliš velká. Nebylo pro uživatele reálné, aby tuto matici počítal v Excelu, nebo jiných programech. Nicméně není nereálné, aby byla tato matice počítána přímo programem.

### Pythagorova věta

Druhou variantou, která byla nakonec použita, je Pythagorova věta (Čučka, 2007). Pythagorova věta je metoda, kterou je možné vypočítat vzdálenost mezi 2 body. K výpočtu stačí znát souřadnice X a Y.

Výhodou tohoto výpočtu je, že není nutné tvořit žádné další matice, nebo uchovávat data o vzdálenostech v extra souborech. Zároveň je možné jednoduše vypočítat vzdálenost mezi 2 body a nejsou prováděny žádné nadbytečné operace.

Nevýhodou je, že je nutné navrhnout a optimalizovat algoritmus, který dotahuje souřadnice potřebných měst. Například pokud znám dvě města, mezi kterými počítám vzdálenost, tak musím někde získat souřadnice. Tohle je záležitost optimalizace přístupu do databáze. Bylo vyzkoušeno mnoho variant práce s městy. Nejjednodušší a nejrychlejší se ukázalo pracovat přímo s indexy měst. Výpočet vzdálenosti je na obr. 4‑1, kde tabulka allCitiesTable obsahuje souřadnice všech měst a Controller je třída, kde jsou shromažďována data.

/// <summary>

/// Vrátí vzdálenost mezi 2 městy. Vstupem jsou index měst

/// </summary>

/// <param name="city1"></param>

/// <param name="city2"></param>

/// <returns></returns>

public static double getDistance(int city1, int city2)

{

double x1 = Convert.ToDouble(Controller.allCitiesTable.Rows[city1]["X"].ToString());

double x2 = Convert.ToDouble(Controller.allCitiesTable.Rows[city2]["X"].ToString());

double y1 = Convert.ToDouble(Controller.allCitiesTable.Rows[city1]["Y"].ToString());

double y2 = Convert.ToDouble(Controller.allCitiesTable.Rows[city2]["Y"].ToString());

return Math.Round(Math.Pow(Math.Pow(x2 - x1, 2) + Math.Pow(y2 - y1, 2), 0.5), 4);

}

Obr. 4‑1 – Výpočet vzdálenosti mezi městy

Předchozí varianta byla použita, když v algoritmu byly používány celé názvy měst. Tohle se ukázalo mnohem složitější, protože se na mnoha místech musely názvy překládat a přístup do tabulky allCitiesTable byl mnohem pomalejší.

### Reálná vzdálenost oproti vzdušné čáře

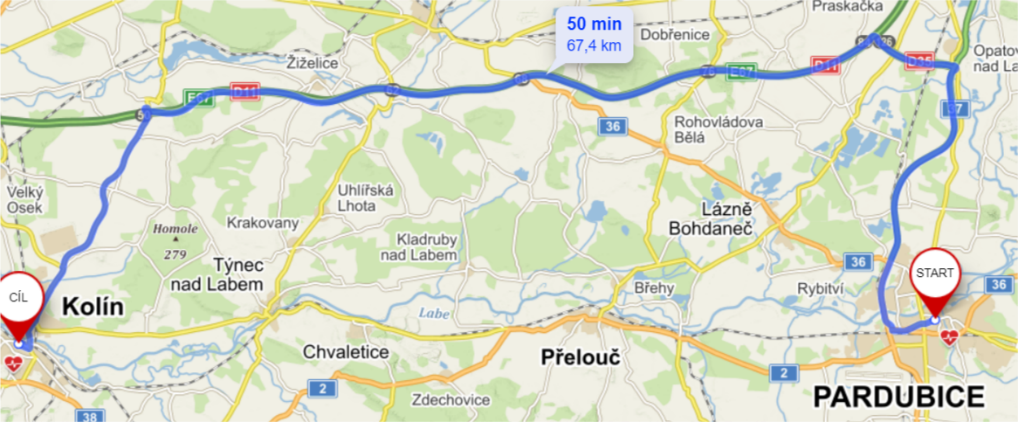
Poslední věcí, kterou je nutné promyslet je rozdíl mezi skutečnou vzdáleností a vypočtenou vzdáleností. Protože k výpočtu jsou použity X a Y souřadnice mapy, tak dochází ke zkreslení výsledku, protože silnice ve skutečnosti nevedou přímou čarou. Pro potřeby algoritmu je tohle skreslení zanedbatelné, protože rovnoměrně nastává při každém výpočtu a chyba se nenásobí mezi sebou a pro ohodnocení cesty není přesnost důležitá.

Další příčinou zkreslení je samotný zemský povrch. Protože zemský povrch se nachází na vypouklé planetě, tak musíme počítat se vzorci pro neeuklidovskou geometrii. V případě této práce i takové zkreslení je možné pominout, protože výpočty probíhají pouze na poměrně malé ploše.

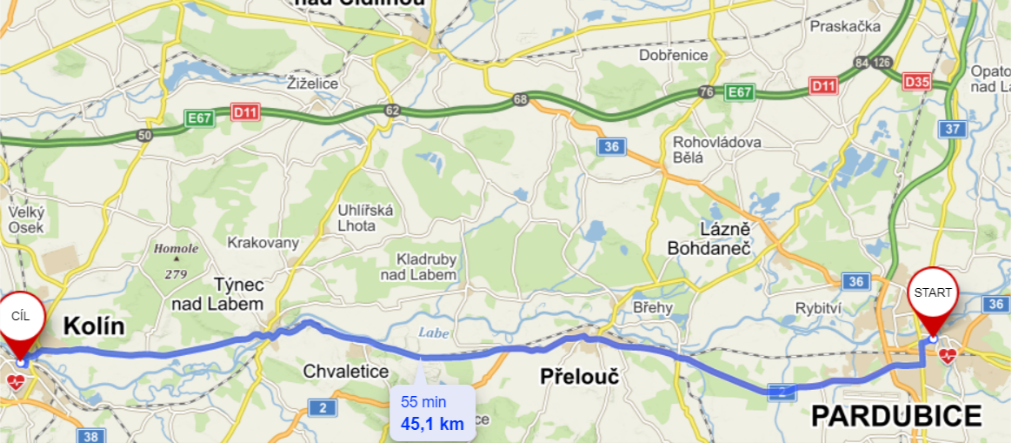
Například cesta z Ostravy do Olomouce je podle výpočtu přibližně 90 km. Nicméně ve skutečnosti je vzdálenost 95,7 km. V jiném případě je vzdálenost z Olomouce do Dubu nad Moravou 13,5 km vzdušnou čarou a 14,1 km skutečná vzdálenost po silnici. Algoritmus by v tomto případě vzdálenost trošičku zkreslil, ale finální výsledek by byl přijatelný.

Problém by byl, kdyby skutečná cesta se nepodobala přímce, ale jednalo by se o křivku jako je půlkružnice, nebo části mnohoúhelníku. Takový případ nastává pouze v případě, že by cesta musela vést přes řeku, nebo jinou překážku, kde by bylo nutné udělat velkou objížďku kvůli jejímu překročení.

Na obr. 4‑2 je ukázka nejkratší cesta a na obr. 4‑3 je nejrychlejší cesta. V drtivé většině případů je nejkratší cesta tou vhodnější, takže z toho důvod nepřesnost kvůli vzdušné čáře není brána v úvahu.



Obr. ‑ – Nejrychlejší cesta (Mapy.cz, 1996)



Obr. ‑ – Nejkratší cesta (Mapy.cz, 1996)

## Krok Programování vnitřního jedince

Vnitřní jedinec je v tomto případě posloupnost měst. Tato posloupnost měst představuje pořadí, neboli cestu, kterou řidič bude muset projet. Celková vzdálenost, kterou řidič projede, bude jeho fitness skóre.

### Vhodná reprezentace vnitřního jedince

Tato posloupnost měst je reprezentována jako pole datového typu string neboli pole řetězců. Existují další možnosti jak reprezentovat posloupnost jako je například pouze jediný řetězec, který by měl uvnitř zakódovanou informaci o městech. Tento způsob byl původně zvoleným řešením, ale ukázalo se, že algoritmus se příliš zpomalí. Zpomalení dosahovalo až pětinásobku původní doby. Na obr. 4‑4 je ukázka reprezentace jedince. Algorithm je třída s algoritmem, která obsahuje všechna nastavení, jako je délka algoritmu, počet generací a podobně. Konstanta LENGTH je zde délka algoritmu a zároveň je celkovým počtem měst.

private string[] sequence = new string[Algorithm.LENGTH];

Obr. 4‑4 – Programová reprezentace vnitřního jedince

Jedinec sám o sobě je vlastně třídou, která má vlastnost zvanou sekvence. Tato sekvence je datového typu pole string, které reprezentuje posloupnost měst. Dále obsahuje pouze metody, které jsou použity pro výpočty.

Hlavními jsou metody GetFitness, která vypočítá celkovou vzdálenost mezi městy. GetDuplicity vypočítá počet duplicit v posloupnosti, která nastává po křížení. Další důležitými metodami jsou GetSequence a ShowMe, které jsou použity pro zobrazení posloupnosti.

### Reprezentace vnitřní generace

Vnitřní generace je třída, která obsahuje instance třídy vnitřní jedinec. Počet instancí je roven počtu vnitřních jedinců ve vnitřní generaci. Vnitřní generace obsahuje metodu selekce jedinců. Tato metoda náhodně vybere 2 dvojice jedinců a pomocí jejich fitness vybere z nich lepší 2. Tito jedinci jsou následně podrobeni křížení, případně mutaci.

Další metodou je ukončovací podmínka. Tato metoda je vždy zavolána ve chvíli, kdy je populace zlepšena. Metoda zkontroluje, jestli nedošlo k dosažení ukončovací podmínky a pokud došlo, tak vrátí hodnotu True, která je datového typu Boolean.

private string[] sequence = new string[Algorithm.LENGTH];  
  
/// Konstruktor bez argumentu -> Náhodná tvorba jedince (Používá se na začátku)  
public Invidual()

/// Projde celou sekvenci a vypočítá vzdálenost mezi sekvencí měst.

public double getDistance()

/// Operátor mutace. 2 náhodné geny jsou prohozeny

public void mutate()

/// Konstruktor s argumentem -> Tvorba jedince ze 2 rodičů

public Invidual(Invidual p1, Invidual p2)

/// Funkce si vypočítá nepoužitá města

/// Následně prochází sekvenci a nahrazuje duplicity těmi nepoužitými

private void fixMe()

/// Ohodnotí jedince a vrátí počet duplicit. Stejný prvek = kladné body

public int getDuplicity()

/// Vrátí sequenci genů jako string (použito pro tisk)

public string getSequence()

Obr. 4‑5 – Vnitřní jedinec, třída Invidual

### Zlepšení populace

Zlepšení populace je řešena na úrovni vnitřního jedince. Vnitřní jedinec má ve skutečnosti 2 konstruktory a 1 metodu pro mutaci. První konstruktor nemá žádné vstupy a vytvoří jedince náhodně. Druhý konstruktor má jako vstup 2 jiné jedince, neboli rodiče a z nich je právě vytvořen křížením.

Křížení je prováděno s 80% pravděpodobností a mutace s 5% pravděpodobností. Populace je periodicky zlepšována při přechodu do každé další generace. Může se stát, že se jedinec do další generace dostane bez změny. Zároveň se do další generace vždy dostane ten nejlepší jedinec z předchozí populace.

### Fitness jedince

Fitness vnitřního jedince je jednoduchý výpočet, kdy je vypočítán součet všech vzdáleností mezi městy v posloupnosti vnitřního jedince. Tento výpočet je proveden v metodě GetFitness a celková vzdálenost není nikde uložena.

Zároveň je zde jednoduchý výpočet, kdy je proveden dotaz do databáze, která obsahuje všechny souřadnice měst. Odpovědí tohoto dotazu jsou souřadnice města. Obsluhu tohoto dotazu bylo nutné chvíli ladit, protože může značně zpomalit průběh výpočtu celého algoritmu.

Města jsou v posloupnosti reprezentována jejich unikátními identifikátory. Tato reprezentace umožňuje jednodušší dotazy do databází, protože identifikátor je zároveň řádkem v tabulce. Tato reprezentace se velmi dobře podepsala na rychlosti algoritmu.

### Zobrazeni jedince

Poslední metodou, která byla potřeba, bylo zobrazení jedince pomocí ShowMe. Metoda ShowMe funguje podobně jako metoda GetFitness. Oproti GetFitness ale pouze zobrazuje posloupnost měst do konzole.

Existují další varianty metody ShowMe jako ShowFull, nebo ShowTranslated. Tyto metody zobrazují jiným způsobem posloupnost měst a byly použity během vývoje a ladění.

### List<string> – Orders | InvidualOrders | PopulationOrder

První způsob, kterým byl vnější jedinec reprezentován, byl List<string> neboli proměnlivé pole řetězců. Tento způsob měl výhodu toho, že byl velmi intuitivní a graficky názorný. Nevýhodou byly právě zmíněné komplikované křížení a mutace. Zároveň docházelo k mnoha duplicitám a ztrácela se města. Z toho důvodu bylo nutné přidat opravnou funkci FixMe. Tato funkce opravovala vnitřní jedince ve vnějším jedinci.

Tato opravná funkce označila duplicity a nahradila je chybějícími městy. Nevyužité chybějící města potom přidala na konec poslední posloupnosti, případně jako další cesty.

Zde vznikal problém, že posloupnosti se uměle měnily k jednomu podobnému výsledku a bylo obtížné dělat jakékoli změny v programu. Příklad generování objednávek je na obr. 4‑6.

### SuperInvidual

Druhý způsob, který byl použit byla reprezentace pomocí jednoho velkého pole datového typu string. Tohle pole obsahovalo pouze informace o řidičích, kteří cestu pojedou. Protože každá buňka reprezentovala jedno město a její hodnota řidiče, tak se nikdy nestalo, že by docházelo ke ztrátě informace o městě, nebo o řidiči.

Novým problémem ale bylo to, že mohlo dojít k přetížení řidiče. Tohle přetížení nebyl zásadní problém, protože pomocí fitness funkce bylo možné přidělovat negativní body, které by vnějšího jedince penalizovaly. Tento způsob byl nakonec zvolen a parametry, které slouží k ohodnocení vnějšího jedince, jsou drženy na úrovni Vnější populace.

Poslední detail při tvorbě vnějšího jedince bylo vhodné propojení metod vnějšího jedince a vnitřního jedince. Pokud třeba je požadavek k zobrazení jedince, tak musí být vhodně zobrazeni všichni vnitřní jedinci. Pokud je požadavek k ohodnocení vnějšího jedince, tak jsou ve skutečnosti ohodnoceni všichni vnitřní jedinci.

public void generateRandomOrders(){

List<string> remainingCities = InputOutput.getUniqueColumnsValuesActiveOrders(1);

var rnd = new Random();

int a = rnd.Next(0, remainingCities.Count);

string city = remainingCities[a];

int cityCount = remainingCities.Count-1;

string newOrder = remainingCities[a];

remainingCities.RemoveAt(a);

int newOrderValue =  
 Convert.ToInt32(Controller.activeOrdersTable.Rows[a]["Count"].ToString());

for (int i = 0; i < cityCount; i++){

a = rnd.Next(0, remainingCities.Count);

if (newOrderValue +  
 Convert.ToInt32(Controller.activeOrdersTable.Rows[a]["Count"].ToString())   
 < ORDER\_CAPACITY)

{ newOrder = newOrder + "-" + remainingCities[a];

remainingCities.RemoveAt(a);

NewOrderValue = newOrderValue +  
 Convert.ToInt32(Controller.activeOrdersTable.Rows[a]["Count"].ToString());}

else

{ newOrder = newOrder + ":" + newOrderValue;

this.ordersList.Add(newOrder);

newOrder = remainingCities[a];

remainingCities.RemoveAt(a);

newOrderValue =+  
 Convert.ToInt32(Controller.activeOrdersTable.Rows[a]["Count"].ToString());}

}

if (newOrder.Length > 0){   
 newOrder = newOrder + ":" + newOrderValue;

this.ordersList.Add(newOrder);}}

Obr. 4‑6 – Původní podoba vnějšího jedince, třída InvidualOrders

public SuperInvidual(){

routes = new int[InputOutput.getActiveOrdersCount()];

List<string> remainingCities = InputOutput.getUniqueColumnsValuesActiveOrders(1);

var rnd = new Random();

int a = rnd.Next(0, remainingCities.Count);

int driver = 1;

int routeValue = 0;

for (int i = 0; i < remainingCities.Count; i++){

while (remainingCities[a] == "X") a = rnd.Next(0, remainingCities.Count);

if (routeValue +   
 Convert.ToInt32(Controller.activeOrdersTable.Rows[a]["Count"].ToString())  
 < ORDER\_CAPACITY)

{routeValue = routeValue +   
 Convert.ToInt32(Controller.activeOrdersTable.Rows[a]["Count"].ToString());

routes[a] = driver;

remainingCities[a] = "X";

}

else

{ driver = driver + 1;

routeValue =   
 Convert.ToInt32(Controller.activeOrdersTable.Rows[a]["Count"].ToString());

routes[a] = driver;

remainingCities[a] = "X";

} }

this.driverCount = driver;  
startAlgorithm();  
}

Obr. 4‑7 – Nová podoba vnějšího jedince, třída SuperInvidual

## Pomocné třídy

Vzhled ke komplexnosti úlohy byly v neposlední řadě vytvořeny pomocné třídy. Tyto třídy slouží pro zpřehlednění celé úlohy a pro jednodušší přidávání dalších funkcionalit do programu. Mezi tyto třídy patří Functions, InputOutput, Cities a Controller. Všechny tyto třídy jsou inicializovány na začátku programu a jsou průběžně volány. Některé slouží skutečně pouze k vývoji a ladění a nejsou nezbytné pro samotný chod programu.

### Functions

Jedná se o třídu se statickými metodami, které slouží k zobrazení vstupu. Těmito metodami je možné zobrazit datový typ pole string, pole Int, List string, List int a mnoho dalších datových typů. Během začátku vývoje programu bylo zároveň nutné vytvořit vhodnou reprezentaci měst, která by zjednodušila práci s velkou databází. Pro tuto reprezentaci byla zvolena hexadická soustava. Třída Functions obsahuje metody pro převod čísel. Hexadická soustava se ukázala jako nepraktická, protože převod probíhal velmi často a algoritmus byl značně zpomalen.

/// Převede znak z Dec do Hex soustavy

/// Vrátí jako char

private static char hexChar(int decValue)

/// Převede číslo z Dec do Hex soustavy

public static string toHex(int decNumber)

/// Zobrazí do konzole všechny prvky v listu

public static void showList(List<string> list)

/// Zobrazí do konzole všechny prvky v listu

public static void showList(List<int> list)

/// Zobrazi do konzole vsechny prvky v poli array

public static void showArray(string[] array)

/// Zobrazi do konzole vsechny prvky v poli array

public static void showArray(int[] array)

/// Zobrazím všechny data z tabulky allCitiesTable do Konzole

public static void showDataAllCitiesTable()

Obr. 4‑8 – Třída Functions

### InputOutput

Tato třída slouží pro manipulaci se vstupními a výstupními soubory. Vzhledem k tomu, že požadavek zákazníka a databáze se souřadnicemi měst může být rozsáhlá a měnit se, tak byly vytvořeny metody, které tyto požadavky uspokojí.

Nejdůležitějšími metodami jsou ty, které čtou soubory a ukládají je do třídy Controller. Bylo nutné vyvinout převodníky a funkce pro čtení strukturovaných souborů. Zároveň metody tisknou přečtená data do zvláštních souborů. Další funkcí této třídy je tisk dat do souborů. Tato funkcionalita byla hojně využívána během vývoje programu. Vytisknout data do souboru a zobrazovat je v jiných aplikacích jako Excel se ukázalo jako velmi vhodné.

### Cities

Tato třída slouží pro výpočty souvisejícími s městy. Původně když byla místo tabulky používána matice nejkratších vzdáleností, tak právě tato třída obsahovala všechny obslužné funkce. Mezi tyto metody patřila tvorba matice nejkratších vzdáleností, přístup do matice, mazání matice, tisknutí matice do souboru, překlad názvů měst a ukládání matice do hlavní třídy Controller. Po tom, co místo matice nejkratší vzdálenosti byla aplikována Pythagorova věta, tak se stala tato třída ve skutečnosti nadbytečná. Zůstaly zde metody, které pouze překládají názvy na identifikátory a obráceně.

/// Vypocte vzdalenost mezi 2 body. x1 a y1 je bod 1. x2 a y2 je bod 2.

/// Vysledek je zaokrouhlen na 4 desetinna mista.

public static double getDistance(int x1, int y1, int x2, int y2)

/// Vrátí vzdálenost mezi 2 městy. Vstupem je název měst

/// Výsledek zaokrouhlen na 4 desetinná místa

public static double getDistance(string city1, string city2)

/// Vrátí vzdálenost mezi 2 městy. Vstupem jsou index měst

public static double getDistance(int city1, int city2)

/// Vrátí index města, podle kterého je možné v allCitiesTable najít jeho X a Y souřadnice

public static int getIndex(string city)

/// Vrátí ID města, když dodám název města

public static string getId(string city)

/// Vrátí index města podle názvu

public static int getIndexId(string city)

Obr. 4‑9 – Třída Cities

### Controller

Třída Controller obsahuje inicializaci všech pomocných tříd. V této třídě se zároveň nachází tabulky se seznamem souřadnic měst a tabulka se seznamem požadavků zákazníků. Hlavní výhodou shromažďování dat v jedné hlavní třídě je ta, že je možné lehce řídit přistupování k datům.

Zároveň je možné lehce ovládat ostatní třídy a pouštět jednoduše algoritmus. Všechny třídy byly navrženy tak, aby je bylo možné pouštět pro různé druhy vstupů a pro různé parametry. Pro potřeby rozšíření programu, nebo vypracování celé aplikace je možné tyto třídy použít a vyvinout grafické uživatelské rozhraní.

Jednou otázkou, kterou bylo nutné vyřešit, bylo to, jestli bude možné spustit algoritmus s jinými parametry. Tohle se ukázalo jako nepotřebné, protože by byl výsledek zkreslený. Je možné nastavit parametry na začátku a spustit s nimi algoritmus, ale není nutné, aby algoritmus běžel s jinými parametry pro různé populace, nebo generace.

Posledním bodem, který bylo nutné rozhodnout, je forma zobrazení. Je možné výstup zobrazit na mapu, kde by bylo možné graficky zobrazit výsledek aplikace. Nakonec se ukázalo, že stačí výstup zobrazit pouze v příkazové řádce, nebo vytisknout do souboru. Tento soubor může být následně použit pro provedení další analýzy.

// Seznam měst, se kterých budu vybírat

public static System.Data.DataTable allCitiesTable = new DataTable("allCities");

public static string pathAllCitiesTable = "..\\..\\..\\..\\" + "Coordinates.txt";

public static string pathDistinctAreasTable = "..\\..\\..\\..\\";

// Aktivní objednávky, kolik palet, který zákazník požaduje

public static System.Data.DataTable activeOrdersTable = new DataTable("activeOrders");

public static string pathActiveOrdersTable = "..\\..\\..\\..\\" + "ActiveOrders.txt";

// Název Kraje, ze kterého budu počítat matici

public static string selectedFile = "Pardubický kraj";

public static string selectedArea = "Pardubice";

public static string pathTest = "..\\..\\..\\..\\" + "Test.txt";

public static string pathOutput = "..\\..\\..\\..\\Output\\";

Obr. 4‑10 – Třída Controller

# METODIKA A VYHODNOCENÍ POKUSŮ

Tato kapitola obsahuje naměřená data. Na základě těchto dat bylo rozhodnuto, jestli je vhodnější hledat optimální skupinu měst, které jsou blízko sebe, nebo jestli je vhodnější hledat přesnější nejkratší vzdálenost. Možných řešení je opravdu mnoho, proto je nutné provést měření mnohokrát a statistický průměr použít jako závěr.

Budou zkoumány 3 hlavní parametry algoritmu:

* Vliv počtu generací
* Vliv ceny za vzdálenost a auto
* Vliv cílových míst

Z každého měření jsou pořízeny záznamy, které obsahovaly hodnoty jednotlivých parametrů algoritmu, hodnoty jedinců a jejich fitness. Samotný název souboru nese informace o měření. Existují dva druhy záznamů.

* Záznam s nejsilnějším jedincem (výstup GA)
* Záznam se všemi jedinci (průběh GA)

Na obr. 5‑1 je ukázka výstupního souboru, průběhu GA v 15. generaci, s nastavením 20 vnějších generací, 50 vnitřních generací a naměřeného 29. 3. 2020 v 16:23. Soubor se nazývá „Test1\_Record\_Vnejsi20Vnitrni50\_Gen15-2020-03-29\_16-23-39.txt“.

Název souboru vždy obsahuje parametry jako počet generací, kde „Vnejsi20“ znamená počet vnějších generací je 20 a „Record“ znamená, že se jedná o celý průběh GA. Na rozdíl od záznamu GA (Record), záznam s nejsilnějším jedincem (výstup GA) obsahuje pouze jednoho jedince.

Parameters Invidual:  
POPULATION\_SIZE: 10 GENERATION\_COUNT: 50  
P\_CROSSOVER: 80 P\_MUTATION: 10  
GOAL\_DISTANCE: 0 DISTANCE\_COST: 10  
CAR\_COST: 10000

Parameters SuperInvidual:  
ORDER\_CAPACITY: 32 OVERLOAD\_PENALTY: 100  
CROSSIN\_POINT: 30 CAR\_COST: 500  
KM\_COST: 10

2020-03-29 16:23:39

Super Invidual: 0  
Fitness Invidual: 29.7147 Mikulovice-Lány u Dašic-Opatovice nad Labem  
Fitness Invidual: 78.1995 Labské Chrčice-Břehy-Jedousov-Borek-Ostřetín  
…  
Fitness Invidual: 47.9853 Jankovice-Brloh-Lázně Bohdaneč-Chvojenec  
Fitness Invidual: 41.1921 Litošice-Bukovina u Přelouče-Neratov-Křičeň-Čeperka  
Fitness SuperInvidual: 13400  
Inviduals: 14

…

Super Invidual: 9  
Fitness Invidual: 29.7147 Mikulovice-Lány u Dašic-Opatovice nad Labem  
Fitness Invidual: 78.1995 Ostřetín-Borek-Jedousov-Břehy-Labské Chrčice  
…  
Fitness Invidual: 47.9853 Chvojenec-Lázně Bohdaneč-Brloh-Jankovice  
Fitness Invidual: 41.1921 Litošice-Bukovina u Přelouče-Neratov-Křičeň-Čeperka  
Fitness SuperInvidual: 13430  
Inviduals: 14

Obr. 5‑1 – Ukázka výstupního souboru

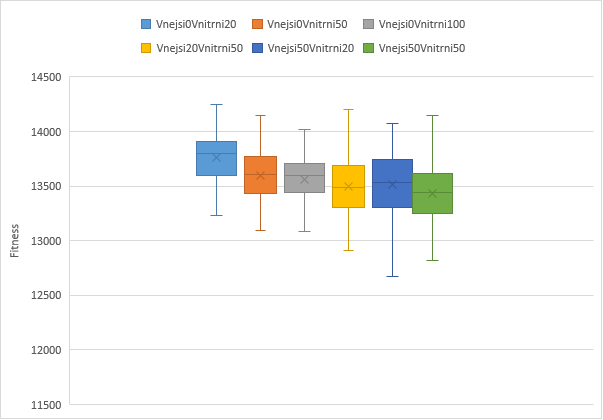
## Zkoumání vlivu počtu generací vnitřního a vnějšího jedince

Pro otestování vlivu generací vnitřního a vnějšího jedince bude provedeno několik variant měření. Otázkou je, jestli je vhodnější hledat kratší vzdálenost, nebo jestli je lepší hledat vhodnější roztřídění měst.

* Pouze vnitřní jedinec 20 generací
* Pouze vnitřní jedinec 50 generací
* Pouze vnitřní jedinec 100 generací
* Vnější jedinec 20 generací Vnitřní 50 generací
* Vnější jedinec 50 generací Vnitřní 20 generací
* Vnější jedinec 50 generací Vnitřní 50 generací

### Shrnutí měření

Bylo provedeno celkem 300 měření. Pro každou variantu bylo provedeno měření padesátkrát. Na jsou zobrazeny průměry fitness skóre vítězných jedinců, které byly výstupem měření. Měření jsou označena podle počtu generací. Například Vnejsi0Vnitrni20 je měření, kdy vnější jedinec byl nastaven na 0 generací a vnitřní jedinec na 20 generací.



Obr. 5‑2 – Průběh parametru fitness vnějších jedinců během měření

### Grafy s naměřenými daty

Obr. 5‑3 – Vývoj fitness nejlepšího jedince během měření Vnejsi20Vnitrni50

Obr. ‑ – Vývoj fitness nejlepšího jedince během měření Vnejsi50Vnitrni20

Obr. 5‑5 – Vývoj fitness nejlepšího jedince během měření Vnejsi50Vnitrni50

Obr. 5‑6 – Srovnání měření fitness pro nejlepší jedince ze všech měření

## Zkoumání vlivu ceny za kilometr a ceny za auto

Pro otestování vlivu ceny za kilometr a ceny za auto jsou provedeny tyto pokusy:

* Pouze vnitřní jedinec 20 generací. Nízká cena kilometr, vysoká za auto
* Pouze vnitřní jedinec 50 generací. Nízká cena kilometr, vysoká za auto
* Pouze vnitřní jedinec 100 generací. Nízká cena kilometr, vysoká za auto
* Vnější jedinec 20 generací Vnitřní 50 generací. Nízká cena kilometr, vysoká za auto
* Pouze vnitřní jedinec 20 generací. Vysoká cena kilometr, nízká za auto
* Pouze vnitřní jedinec 50 generací. Vysoká cena kilometr, nízká za auto
* Pouze vnitřní jedinec 100 generací. Vysoká cena kilometr, nízká za auto
* Vnější jedinec 20 generací Vnitřní 50 generací. Vysoká cena kilometr, nízká za auto
* Pouze vnitřní jedinec 20 generací. Vyrovnaná cena
* Pouze vnitřní jedinec 50 generací. Vyrovnaná cena
* Pouze vnitřní jedinec 100 generací. Vyrovnaná cena
* Vnější jedinec 20 generací Vnitřní 50 generací. Vyrovnaná cena

V těchto pokusech bude posuzováno, jak velký vliv má cena na výsledek algoritmu.

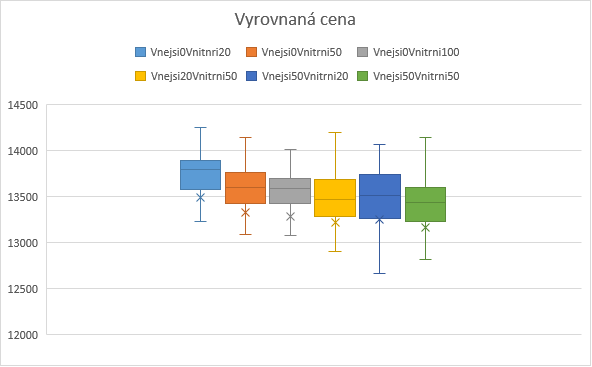
### Shrnutí měření

Bylo provedeno celkem 240 měření. Pro každou variantu bylo provedeno měření dvacetkrát.

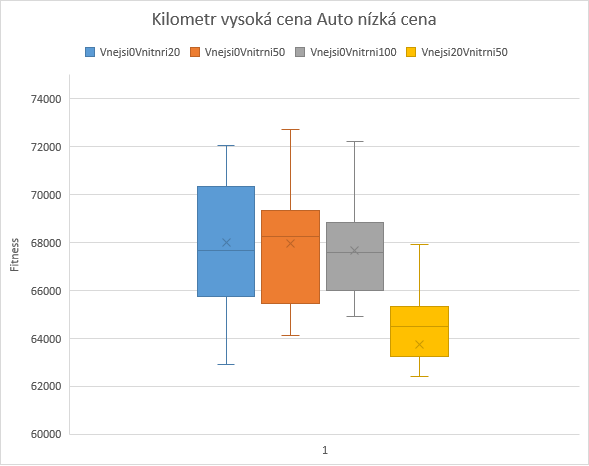
* Vysoká cena za kilometr je 100/km.
* Nízká cena za kilometr je 1/km
* Vyrovnaná cena za kilometr je 10/km
* Vysoká cena za auto je 5000/Auto
* Nízká cena za auto je 50/Auto
* Vyrovnaná cena za auto je 500/Auto

Jednotlivá měření jsou opět nazvány podle počtu generací. Tedy Vnejsi50Vnitrni20 je měření, kde vnější jedinec je nastaven na 50 generací a vnitřní jedinec je nastaven na 20 generací.

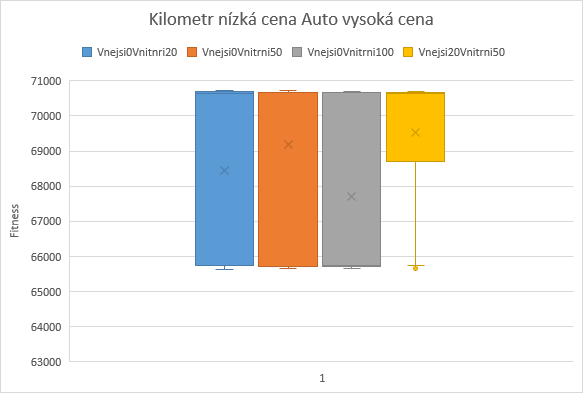
### Grafy s naměřenými daty



Obr. 5‑7 – Průběh parametru Fitness během měření s vyrovnanou cenou



Obr. 5‑8 – Parametr Fitness pro měření s vysokou cenou za kilometr a nízkou za auto



Obr. 5‑9 – Parametru Fitness pro měření s nízkou cenou za kilometr a vysokou za auto

## Zkoumání vlivu umístění cílových měst

Pro otestování vlivu umístění cílových měst jsou provedeny následující pokusy:

* Pouze vnitřní jedinec 20 generací. Města jsou z více krajů
* Pouze vnitřní jedinec 50 generací. Města jsou z více krajů
* Pouze vnitřní jedinec 100 generací. Města jsou z více krajů
* Vnější jedinec 20 generací Vnitřní 50 generací. Města jsou z více krajů
* Vnější jedinec 50 generací Vnitřní 20 generací. Města jsou z více krajů
* Vnější jedinec 50 generací Vnitřní 50 generací. Města jsou z více krajů

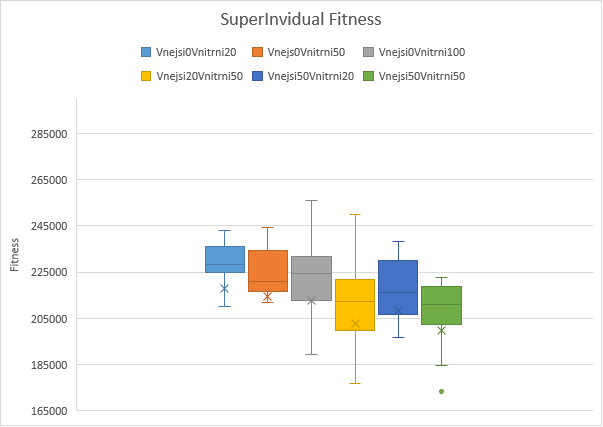
V těchto pokusech bude posuzováno, jak velký vliv má rozmístění měst na algoritmus.

### Shrnutí měření

Bylo provedeno celkem 120 měření. Pro každou variantu bylo provedeno měření dvacetkrát. Města byla vybrána celkem ze tří krajů. Tyto kraje jsou Olomoucký kraj, Pardubická kraj a kraj Vysočina. Tyto kraje jsou od sebe vzdáleny vzdušnou čarou přibližně 100 km.

Jednotlivá měření jsou opět nazvány podle počtu generací. Tedy Vnejsi50Vnitrni20 je měření, kde vnější jedinec je nastaven na 50 generací a vnitřní jedinec je nastaven na 20 generací.

### Grafy s naměřenými daty



Obr. 5‑10 – Vývoj parametru Fitness během měření s městy daleko od sebe

## Vliv počtu generací na nejkratší vzdálenost

Z obr. 5‑2 je vidět, že algoritmus s pouze vnitřním členem dosáhl viditelného snížení vzdálenosti. Toto snížení se postupně snižovalo, až začalo docházet k saturaci a vzdálenost následně klesala minimálně. Dále je vidět, že použití vnějšího členu umožnilo tuto vzdálenost ještě více snížit.

V kombinaci dvaceti generací vnitřního člena a dvaceti generací vnějšího člena se projevila náhodná mutace, kdy v patnácté generaci několik současných mutací vnějších členů negativně ovlivnilo průměrnou fitness celé vnější generace. Kombinace dvaceti generací vnějšího člena a padesáti generací vnitřního člena umožnila o trošičku kratší vzdálenost, ale čas k dokončení algoritmu se značně zvýšil.

Rozmezí maximálních a minimálních hodnot se jinak drželo přibližně ve stejném rozptylu. Průměrná vzdálenost se postupně snižovala s přibývajícími generacemi.

Na obr. 5‑4 je vidět, jak náhodně vygenerované počáteční cesty ovlivnily cílový výsledek. Pokud byla počáteční cesta nevhodně zvolena, tak došlo ke značnému snížení velmi rychle. Pokud počáteční cesta byla vygenerována vhodně, tak algoritmus moc vzdálenost trasy nesnížil. Zde je vidět důležitost vnějšího generaci z důvodu rozmanitosti řešení.

## Vliv ceny na fitness jedince

Podle obr. 5‑8 lze říci, že cena může značně ovlivnit cílovou trasu. V případě, když byla cena za kilometr vysoká, tak dosahovaly lepších výsledků algoritmy, kdy byl počet generací vnitřního jedince vyšší. Vnější jedinec zde fungoval pouze k vyhnutí se lokálnímu extrému.

Na obr. 5‑9 je na druhou stranu vidět, že cena za auto nehrála významnou roli vzhledem k tomu, že předměty musely být naloženy všechny a algoritmus přirozeně vybíral možnosti, kdy počet aut byl nižší. Nicméně pro jiná nastavení je možné, že větší počet aut je výhodnější.

## Vliv rozmístění měst na nejkratší vzdálenost

Z obr. 5‑10je zřejmé, že počet generací vnitřního jedince očekávaně snížil celkovou vzdálenost. Na druhou stranu zapojení vnějšího jedince měla mnohem větší dopad, protože města byla daleko od sebe a do výběru se mohlo dostat více sekvencí měst. V těchto sekvencích se právě mohly objevit skupiny měst, které byly blízko sebe.

# ZÁVĚR

Hlavním úkolem práce bylo se pokusit použit optimalizaci v logistice. Pomocí provedených měření se podařilo zjistit, jak dlouho má význam hledat nejkratší cestu, jak může fitness funkce ovlivnit algoritmus a jakou roli má rozdělení měst na menší celky.

Nejdůležitějším poznatkem této práce je, že vzdálenost mezi městy může znatelně ovlivnit optimální trasu. Je důležité tedy, aby byla cílová města rozdělena do menších celků, které jsou poblíž, a pro tyto skupiny byla hledána nejkratší vzdálenost.

Rozdělení měst do menších celků, které jsou poblíž, může být provedeno zvýšením počtu generací pro vnějšího jedince. Vyšší počet generací vnitřního člena přitom sníží samotnou vzdálenost, kterou řidič bude muset projet.

Samotná vzdálenost dosahuje nejznatelnějšího snížení už po dvaceti generacích, další generace se podílejí na stále menším a menším snížení této vzdálenosti. Během těchto pozdějších generací dochází sice k pokroku, ale doba k provedení algoritmu se značně zvyšuje.

V budoucnu by bylo možné na práci pokračovat zejména ve výzkumu vlivu počtu generací. Je možné algoritmus v aktuálním stavu nastavit na různé kombinace počtu generací. Například při počátečních generacích je možné nechat počet generací vnitřního jedince nízko a zvýšit je ke konci algoritmu, nebo po určitých milnících.

Zároveň je možné nastavit počty generací dynamicky. Kupříkladu kdyby fitness skóre dosáhlo zajímavé hodnoty, tak je možné pustit vnitřní algoritmus s vyšším počtem generací a přeskočit nezajímavé vnější jedince.

Další možností je optimalizace samotného algoritmu. Existují části, kde je možné algoritmus urychlit jako je přístup k datům o souřadnicích. Tato část se zásadně podílí na rychlosti algoritmu a výrazně ovlivňuje počet vstupů, které je možné zpracovat.

Poslední možností je přechod na jiné optimalizační úlohy jako je výpočet cesty pro 3D tiskárnu, vrták plošných spojů, nebo tvorbu rozvrhů. Práce celkově může posloužit jako dobrý základ pro podobné úlohy pouze s několika málo úpravami.

# SEZNAM LITERATURY

*C#* [online]. Microsoft, 1999 [cit. 2020-05-01]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/

ČUČKA, Michal. *Pythagorova věta a její důkazy*. Brno, 2007. Diplomová. VUT. Vedoucí práce Pavlína Račková.

Hledání nejkratší cesty v grafu. *ITnetwork.cz* [online]. 2020 [cit. 2020-05-01]. Dostupné z: <https://www.itnetwork.cz/navrh/algoritmy/algoritmy-grafove/hledani-nejkratsi-cesty-v-grafu>

HORKÝ, Aleš. *Tvorba rozvrhů pomocí genetických algoritmů*. Brno, 2011. Bakalářská. VUT. Vedoucí práce Miloš Minařík Ph.D.

HYNEK, Josef. *Genetické algoritmy a genetické programování*. Praha: Grada, 2008. ISBN 9788024726953.

CHONG, David. The Aged P versus NP Problem. *Towards data science* [online]. [cit. 2020-04-14]. Dostupné z: <https://towardsdatascience.com/the-aged-p-versus-np-problem-91c2bd5dce23>

KLAUS, Vladimír. ZjišŤujeme vzdálenost mezi dvěma GPS souřadnicemi, tentokrát v ASP.NET. *Vladimír Klaus BLOG* [online]. [cit. 2020-04-16]. Dostupné z: <https://www.vladimirklaus.cz/CZ/clanky-detail/33/zjistujeme-vzdalenost-mezi-dvema-gps-souradnicemi-tentokrat-v-aspnet>

KOLÁŘ, Josef. *Teoretická informatika*. 2. vyd. Praha: Česká informatická společnost, 2000. ISBN 80-900-8538-5.

KUNZ, Milan. Matice vzdáleností. *NATURA PLUS* [online]. 2001 [cit. 2020-05-01]. Dostupné z: <http://natura.baf.cz/natura/2001/1/20010104.html>

*Mapy.cz* [online]. Praha, 1996 [cit. 2020-05-01]. Dostupné z: [www.mapy.cz](http://www.mapy.cz)

MÍČA, Ondřej. *Solving Travelling salesman problem using Harmony search algorithm and other metaheuristics* [online]. QUAERE: MAGNANIMITAS, 2016 [cit. 2020-04-12]. ISBN 978-80-87952-15-3. Dostupné z: <https://dk.upce.cz//handle/10195/67047>

Traveling Salesman Problem. *Google OR-Tools* [online]. [cit. 2020-04-12]. Dostupné z: <https://developers.google.com/optimization/routing/tsp>

Variace. *Matematika.cz* [online]. Brno: Vydavatelství Nová média, 2006 [cit. 2020-03-23]. Dostupné z: <https://matematika.cz/variace>

Vehlice Routing Problem. *Google OR-Tools* [online]. [cit. 2020-04-12]. Dostupné z: <https://developers.google.com/optimization/routing/vrp>

ZEMEK, Michal. Souradnice-mest. *Github* [online]. 2018 [cit. 2020-05-01]. Dostupné z: https://github.com/33bcdd/souradnice-mest

# PŘÍLOHY

# A – CD

**Příloha A**

**Příloha k bakalářské práci**

VYUŽITÍ OPTIMALIZACE V LOGISTICE

Lukáš Míšek

**CD**

# Obsah

1 Text bakalářské práce ve formátu PDF

2 Zdrojové kódy sestavených aplikací