**UNIVERZITA PARDUBICE**

Fakulta elektrotechniky a informatiky

**Využití optimalizace v logistice**

Lukáš Míšek

Bakalářská práce

2020

Zde bude sken první stranky

*Prohlaseni zde bude – nutny SKEN zadani (v praci ve skrince)*

**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**

*Optimilization, 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, palling …). In teoretical part student will describe chosen problém, existing methods, design and realize experiments (programm in chosen programming language) and analyse results.*

**KEYWORDS**

*microcontroller, programming, control system.*

[1 Rozbor zadání 10](#_Toc35203773)

[1.1 Podobné problémy 10](#_Toc35203774)

[1.2 Návrh řešení 11](#_Toc35203775)

[1.3 Blokové schéma 12](#_Toc35203776)

[1.4 Úvod do problematiky 13](#_Toc35203777)

[2 Genetické algoritmy a programování 14](#_Toc35203778)

[2.1 Princip genetických algoritmů 14](#_Toc35203779)

[2.2 Fitness funkce 16](#_Toc35203780)

[2.3 Selekce 17](#_Toc35203781)

[2.4 Křížení 18](#_Toc35203782)

[2.5 Mutace 18](#_Toc35203783)

[3 Návrh algoritmu 19](#_Toc35203784)

[3.1 Úloha, problematika 19](#_Toc35203785)

[3.2 Rozbor zadání 20](#_Toc35203786)

[3.2.1 Vnitřní jedinec 21](#_Toc35203787)

[3.2.2 Vnitřní generace 21](#_Toc35203788)

[3.2.3 Vnější jedinec 21](#_Toc35203789)

[3.2.4 Vnější generace 21](#_Toc35203790)

[3.3 Možné výstupy algoritmu 22](#_Toc35203791)

[3.3.1 Města jsou nevhodně rozděleny na úrovni vnějšího členu 22](#_Toc35203792)

[3.3.2 Nebyla dosažena nejkratší vzdálenost 22](#_Toc35203793)

[3.3.3 Odhad optimální řešení 23](#_Toc35203794)

[4 Řešení vlastní práce 24](#_Toc35203795)

[4.1 Krok Výpočet nejkratší vzdálenosti 24](#_Toc35203796)

[4.1.1 Matice nejkratších vzdalenosti 24](#_Toc35203797)

[4.1.2 Pythagorova veta 25](#_Toc35203798)

[4.1.3 Reálná vzdálenost vs vzdušná čára 25](#_Toc35203799)

[4.2 Krok Programování vnitřního jedince 26](#_Toc35203800)

[4.2.1 Vhodná reprezentace vnitřního jedince 26](#_Toc35203801)

[4.2.2 Reprezentace vnitřní generace 26](#_Toc35203802)

[4.2.3 Zlepšení populace 27](#_Toc35203803)

[4.2.4 Fitness jedince 27](#_Toc35203804)

[4.2.5 Zobrazeni jedince 27](#_Toc35203805)

[4.3 Krok Programovani Vnejsiho jedince 28](#_Toc35203806)

[4.3.1 List<string> - Orders | InvidualOrders | PopulationOrder 28](#_Toc35203807)

[4.3.2 SuperInvidual 28](#_Toc35203808)

[4.4 Pomocne tridy 29](#_Toc35203809)

[4.4.1 Functions 29](#_Toc35203810)

[4.4.2 InputOutput 29](#_Toc35203811)

[4.4.3 Cities 30](#_Toc35203812)

[4.4.4 Controller 30](#_Toc35203813)

[5 Statistika 31](#_Toc35203814)

[5.1 Stare reseni full (Ok výsledky?) 31](#_Toc35203815)

[5.2 Stare reseni bez opravy (Spatny výsledky?) 31](#_Toc35203816)

[5.3 Nove reseni full (Top výsledky?) 31](#_Toc35203817)

[5.4 Nove reseni vnejsi jedinec 50+ vnitrni 20- 31](#_Toc35203818)

[5.5 Nove reseni vnejsi jedince 20- vnejsi 50+ 31](#_Toc35203819)

[5.6 Pouze vnitrni jedinec 31](#_Toc35203820)

[6 Zaver 32](#_Toc35203821)

[7 Seznam literatury: 33](#_Toc35203822)

Nenalezena položka seznamu obrázků.

**Seznam zkratek:**

Txt … textový soubor

Csv … textový soubor s oddělovači

* 1. 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.

* 1. Podobné problémy

Podobným problémem se zabýval Aleš Horký 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 1 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 4 učiteli, 40 žáky, 1 místností a 4 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. Nelze zároveň dát přednost méně prioritním předmětům, které je možné učit v pozdějších hodinách.

Řešení je stále poměrně jednoduché. Co se ale stane, když máme školu o 15 učitelích, 20 třídách, kde má každá 20-30 žáků a k dispozici bude pouze 12 učeben? Zároveň zde budou požadavky na předměty, které mohou být vyučovány pouze v konkrétních třídách. Například hodiny chemie, tělocvik, nebo hudební výchova. Aleš Horký tuto problematiku podrobně popisuje v jeho práci.

* 1. Návrh řešení

Řešením by tedy měl být program, který je schopen vyřešit daný problém. Poskytnuté řešení by mělo vyhovět požadavkům zadaným zadavatelem. Dále by mělo poskytnou řešení v reálném čase a 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.

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 obsahuje nejmenší množství konfliktů. Uživatel by potom mohl řešit problém nestandardním řešením.

Výstup aplikace bude řešen pomocí csv, nebo excel 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 zvláštní systémy.

V neposlední řadě je důležité, aby byla aplikace prezentovatelné široké veřejnosti. Aplikace by neměla být příliš složitá, aby ji mohl ovládat pouze specializovaný pracovník, nebo být tak specializovaná, že ji je možné použít k řešení jenom několika málo problémů. Aplikace tedy bude rozhodně obsahovat grafické rozhraní a bude realizována v programovacím jazyce C#.

* 1. Blokové schéma

Vstupem aplikace budou soubory s požadavky ve formátu txt, csv, nebo xls. Soubory představují jednotlivé kapacity, které se musí sejít. Mějme tedy Kamion, který představuje nákladový prostor. Každý kamion má místo na 32 palet. Potom máme palety, které jsou dále předměty. Třetím souborem budou skladníci. Skladníci neboli pracovníci mohou nakládat do kamiónů pouze určité kamiony. Tyto tři soubory představují náš vstup.

Dále je aplikace samotná. Ta se skládá z vyhodnocení vstupu, samotného algoritmu a výstupu. Výstup v tomhle případě je kombinace skladníků, kamiónů a palet. Program pomocí algoritmu bude generovat velké množství výstupu. Tyto výstupy musí být ohodnoceny, aby bylo možné rozlišit, které jsou nevyhovující a které vyhovují. Program si bude nejlepších 20% výstupů pamatovat. Tohle bude dalším vstupem do algoritmu.

Budou zde tedy paměťové blogy. Blog k vyhodnocení výstupu a blok k uchování výstupu, který bude prezentován uživateli. V neposlední řadě bude mít program blok s kontrolním nastavením. Zde budou vstupy, které nepřímo ovlivňují výpočet. Například počet generací, které budou generovány, velikost populací, procento generací, které je označeno za úspěšné a uživatelské nastavení, jako formát výstupu.

* 1. Úvod do problematiky

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.

Jedním z příkladu řešení je takzvaná hrubá síla. To znamená, že zkouším náhodné kombinace a vyhodnocuji, jestli jsou tyto kombinace vyhovující. Tento postup přesto, že funguje, tak není optimální, protože čas pro nalezení vhodné kombinace je exponenciálně zvyšuje s rostoucími vstupy.

Tyto vstupy jsou konkrétní požadavky zadavatelů. Aleš Horký v jeho práci zmiňuje dva druhy omezení. Takzvané Tvrdé podmínky a Měkké podmínky. Tvrdé podmínky musí být splněny vždy. Měkké podmínky nemusí být nutně splněny. Nicméně jejich splnění pozitivně ovlivní ohodnocení kombinace pomocí hodnotící funkce.

Mezi tvrdé podmínky platí 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.

Měkké podmínky jsou takové, které nemusí být splněny. Nicméně jejich splnění pozitivně ovlivní 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 přiřazeno takzvané fitness skóre. Toto fitness skóre udává, jak vhodné je řešení. 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.

1. Genetické algoritmy a programování

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.

Tato kniha slouží 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 kapitole 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.

* 1. Princip genetických algoritmů

Libovolnou reálnou úlohu je možné matematickým, nebo symbolickým způsobem popsat. Řekněme, že obchodník potřebuje navštívit 3 místa. Řešením této úlohy je posloupnost 3 měst. Tyto města označíme jako A, B, C. Řešení potom může vypadat následovně:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | A | B | C | |  |
|  |  |

Tato posloupnost měst je vhodným řešením a zároveň je nejkratší vzdáleností mezi těmito 3 body. Tuto posloupnost nazveme jedincem, kde každý jeho prvek bude genem. Zde jsou všechna možná řešení pro tuto úlohu:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| A | B | C |  | A | C | B |
|  |  |  |  |  |  |  |
| B | C | A |  | B | A | C |
|  |  |  |  |  |  |  |
| C | A | B |  | C | B | A |

Problém nastane ve chvíli, když počet měst se zvýší. Potom nemusí být zjevné, co je nejkratší vzdáleností. V tom případě vygenerujeme skupinu jedinců o konečném počtu, kde každý jedinec bude obsahovat náhodné geny. Tuto skupinu nazveme generací.

Pro nalezení nejvhodnějšího řešení budeme kombinovat jedince v aktuální generaci. Z generace 1 vybereme 2 jedince, kteří budou rodiči. Tyto 2 rodiče zkombinujeme a vytvoříme tím nového jedince. Tento nový jedinec bude potomkem součástí další generace. Tedy generace 2.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **Rodič 1** | A | B | C | D | E | |  |  |  |  |  |  | | **Rodič 2** | D | A | C | B | E | | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  |  |  |  |  |  | | **Potomek** | A | B | C | B | E | |  |  |  |  |  |  | |
|  |  |

Zde jsme zkombinovali 2 jedince (rodiče) z generace 1 a vytvořili jsme z nich 1 jedince (potomka) z generace 2. Podobným způsobem naplníme celou generaci 2 kombinováním náhodných rodičů. Tento postup se opakuje pro daný počet generací.

Tuto kombinaci jedinců nazvěme křížením. Výběr dvou rodičů nazvěme selekcí. Celkovou vzdálenost mezi městy zobecníme a nezveme ji fitness funkcí. Celý algoritmus by potom fungoval tak, že by každý další generace byla tvořena potomky, kteří byli vybrání z rodičů podle fitness funkce. Právě díky fitness funkci by se do další generace dostaly geny pouze těch nejlepších jedinců.

* 1. Fitness funkce

Každý jedinec je reprezentací řešení. Jak dobré toto řešení je, popisuje fitness funkce neboli ohodnocení. Fitness funkce může být součet vzdáleností mezi městy, potom by menší fitness funkce byla lepším a vhodnějším řešením neboli silnějším jedincem.

Jiným příkladem fitness funkce může být počet konfliktů v rozvrhu. Jedinec by byl posloupností předmětů vyučovaných ve škole a každý konflikt by se započítal do fitness funkce. Fitness funkce také může být získanými body žákem. Každý žák by odpověděl na testové otázky jiným způsobem a získané body by byla právě fitness funkce.

V některých případech by vyšší fitness funkce mohla být negativní a jindy by mohla být pozitivní. Na základě této fitness funkce by potom byli vybrání ti nejlepší jedinci a z nich by byli vytvořeni jedinci další generace. Tento proces se nazývá selekce a může vypadat následovně.

Zde máme 2 generace. Obě generace mají celkem 12 jedinců. Číslice reprezentují fitness funkce každého jedince. V tomto případě vyšší fitness značí silnějšího jedince.

* Žlutě jsou označení nejsilnější jedinci z generaci 1. Tito jedinci mají největší šanci na vytvoření potomstva.
* Modře byl označen jedinec, který je silnější, než všichni z předchozí generace. Tento jedinec má nyní největší pravděpodobnost, aby vytvořil potomstvo do další generace.
* Červeně byl označen jedinec, který „se nepovedl“. Tento jedinec během selekce „odpadne“.

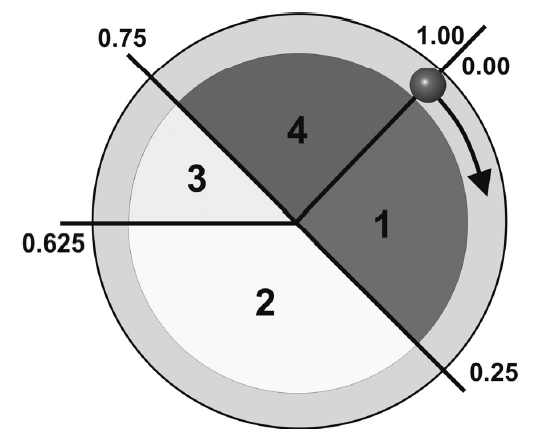
|  |  |
| --- | --- |
| **Generace 1:** | **Generace 2:** |
| |  |  |  | | --- | --- | --- | | 93 | 38 | 1 | | 40 | 5 | 65 | | 33 | 1 | 89 | | 32 | 34 | 57 | | |  |  |  | | --- | --- | --- | | 63 | 26 | 29 | | 58 | 0 | 67 | | 95 | 87 | 92 | | 21 | 19 | 63 | |

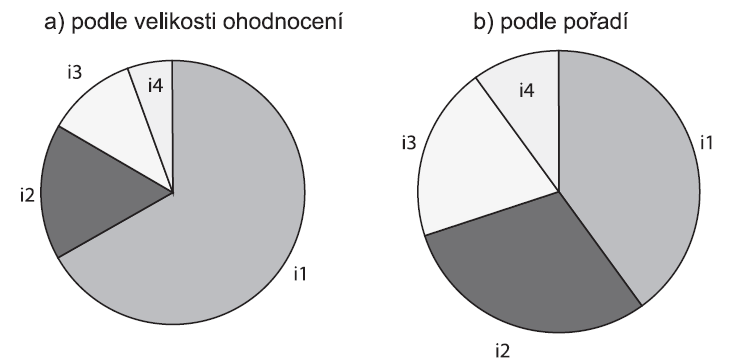
* 1. Selekce

Selekce je způsob, kterým jsou vybírání jedinci do další generace. Většinou se používá fitness funkce, podle které jsou označeni ti nejvhodnější. Vzhledem k tomu, že je algoritmus do určité částí řízen náhodou, tak nelze ale 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 funkci, 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 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. Zde je fitness funkce přímo úměrná ploše na ruletě. V tomto případě je velmi vysoká pravděpodobnost, že jedinec s vysokou fitness funkcí (2) se neztratí a jedinec s nízkou fitness funkcí (3) se ztratí. Nevýhodou je, že slabý jedinec (3) může obsahovat důležitý gen, který nemusí obsahovat jiný jedinec z populace.

Ruletu je možné pozměnit, aby plocha nebyla přímo úměrná velikosti fitness funkce (ohodnocení), ale pořadí. Potom bude ruleta vypadat takto. Zde má slabší jedinec vyšší šanci na podílení se tvorby nové generace. Zároveň jsou silnější jedinci stále zvýhodněni díky jejich vlastnostem.

Dalším způsobem selekce je selekce pomocí turnaje. Selekce pomocí turnaje funguje tak, že jsou náhodně vybráni 2 dvojice jedinců a mezi z těchto dvojic jsou vybráni 2 vítězové, kteří se 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.

* 1. Křížení

Křížení je operace, která vezme 2 jedince a z jejich genů vytvoří nového jedince. 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. Po křížení se může výsledný potomek podstatně lišit od jeho rodičů.

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.

Zároveň je důležité, aby nebylo křížení zvoleno nerozumně. Například by neměl být křížen jedinec v poměru 1:9, kde rodič 1 by předal potomku 10% svých genů a rodič 2 by předal 90% svých genů. Z by mohlo pravidelně docházet ke ztrátě genů.

* 1. Mutace

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

K mutaci na rozdíl od křížení by mělo docházet velmi zřídka. Pokud by byla pravděpodobnost mutace příliš vysoká, tak by byl algoritmus příliš náhodný a nikdy by nemusel dojít do konečného řešení.

1. 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.

* 1. 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 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. Tímto způsobem je možné dosáhnout rychle výsledku a zároveň je pravděpodobné, že náhodným způsobem se poprvé netrefíme a bude nutné skládat rozvrh několikrát, než bude dosaženo přijatelného řešení.

Další možností je možné zkoušet poskládat rozvrh hrubou silou a zkusit všechny možnosti, které jsou. Nicméně se zvyšujícím se počtem podmínek, vstupů a výstupu se počet kombinací exponenciálně zvyšuje. Tohle je zvané 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.

Genetické programování 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.

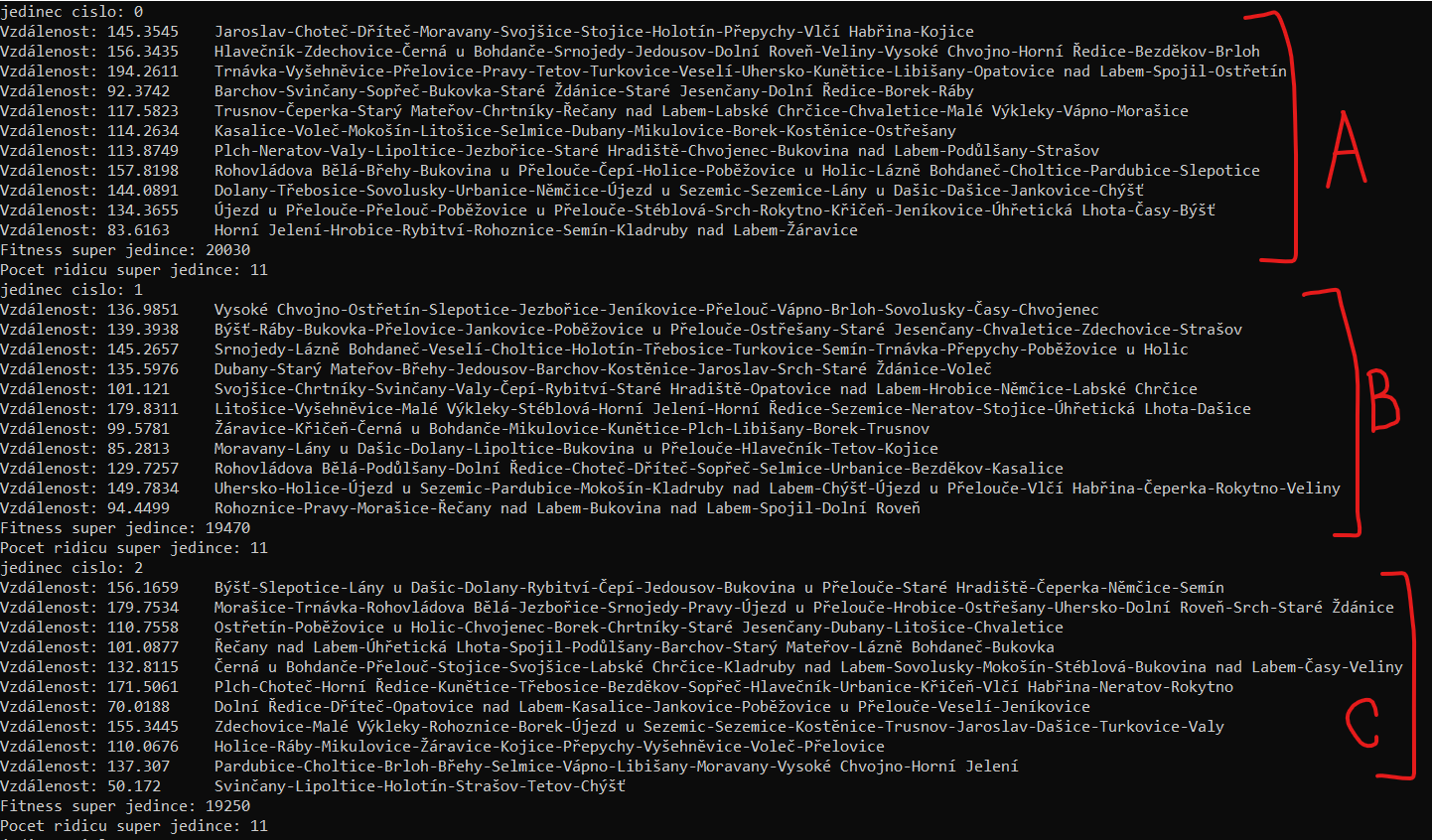
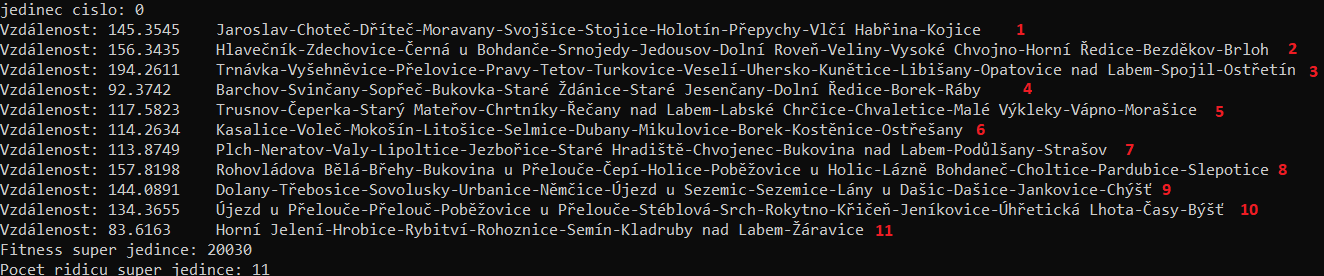
* 1. Rozbor zadání

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

Tato cesta má reálnou vzdálenost a vhodným řešením se rozumí takové, které se přibližuje k nejkratší vzdálenosti. V případě obchodního cestujícího vhodným řešením je takové řešení, kde se cestující zbytečně nevrací a města jsou seskupena do skupin.

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

Každý algoritmus bude mít několik fází.

1. Zboží je nahodile rozděleno mezi několik cestujících. Je vytvořeno několik variant tohoto rozdělení. Každá varianta se nazývá vnější jedinec a tito vnější jedinci tvoří vnější generaci. Na obrázku jsou generace označeny jako písmena A, B a C.
2. Každý vnější jedinec obsahuje posloupnosti měst. Tyto posloupnosti měst představují pořadí, ve kterém cestující města navštíví.
3. Je vypočtena vzdálenost mezi těmito městy. Tyto posloupnosti představují vnitřní jedince a tvoří vnitřní generaci.
4. Součet vzdáleností vnitřních jedinců je označeno jako Fitness vnějšího jedince.
5. Každý vnější jedinec ve vnější generaci je podroben zlepšení. Je zde 80% pravděpodobnost ke křížení a 5% pravděpodobnost k mutaci.
6. Pokud je dosažena ukončující podmínka, tak je algoritmus ukončen.
7. Pokud ukončující podmínka není splněna, tak algoritmus pokračuje krokem 3.
   * 1. Vnitřní jedinec

Vnitřní jedinec představuje posloupnost měst, kterou musí navštívit jeden obchodník. Pořadí těchto měst je pořadím, ve kterém obchodník města navštíví. Celková vzdálenost, kterou obchodník projde je Fitness funkcí.

* + 1. Vnitřní generace

Vnitřní generace obsahuje konečný počet vnitřních jedinců. Tito vnitřní jedinci jsou podrobeni genetickému algoritmu a vnitřní jedinec s nejnižší fitness funkcí (nejkratší vzdáleností) je označen za vítěze.

* + 1. Vnější jedinec

Vnější jedinec obsahuje několik posloupností, které jsou tvořeny vítězi z vnitřních generací. Fitness funkce vnějšího jedince obsahuje několik částí. Tyto části jsou sečteny.

První část je cena za vzdálenost. Cena za vzdálenost je vypočtena ze součtu vzdáleností vnitřních jedinců a je vynásobena cenou za kilometr.

Druhou částí je cena za řidiče. Je sečten počet vnitřních jedinců ve vnějším jedinci a tento počet je vynásoben cenou za řidiče.

* + 1. Vnější generace

Vnější generace obsahuje konečný počet vnějších jedinců. Tito vnější jedinci navzájem soupeří a ten nejvhodnější z nich je označen jako vítěz a zároveň výstup algoritmu.

* 1. 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. Jsou 4 hlavní scénáře, které budou řešeny.

* Města nejsou pohromadě
* Města jsou pohromadě
* Cesta není tou nejkratší
* Cesta je tou nejkratší
  + 1. Města jsou nevhodně rozděleny na úrovni vnějšího členu

Pokud by nebyly města pohromadě, tak by nejkratší vzdálenosti mohly vypadat takto. V obou případech je nejkratší vzdálenost nalezena, ale v případě nalevo je tato vzdálenost mnohem vyšší, než vzdálenost v případě napravo.

|  |  |
| --- | --- |
|  |  |

* + 1. Nebyla dosažena nejkratší vzdálenost

V tomhle případě se může stát, že řidič bude mít vhodně určenou, 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 obrázku vlevo jsou zvoleny ty nejkratší vzdálenost a na obrázku vpravo jsou vzdálenosti o něco delší.

|  |  |
| --- | --- |
|  |  |

* + 1. Odhad optimální řešení

Z předchozích kapitol je vidět, že špatný výpočet nejkratší vzdálenosti, i špatné rozdělení měst obchodníkovi může mít zásadní dopad na výslednou cestu. Problémem také může být předčasné ukončení algoritmu a to jak vnitřního, tak vnějšího.

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.

1. 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#. 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 33bcdd dostupného zdarma na stránkách <https://github.com/33bcdd/souradnice-mest>. 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.

* 1. 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.

* + 1. Matice nejkratších vzdalenosti

Matice nejkratších 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 byl tento soubor počítán přímo programem.

* + 1. Pythagorova veta

Druhou variantou, která byla nakonec použita, je Pythagorova věta. 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.

* + 1. Reálná vzdálenost vs vzdušná čára

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 skreslení výsledku.

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. Výsledek bude i přes tuto nepřesnost přijatelný.

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 skreslil, 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 několika úhelník. 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.

* 1. 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 bude muset projet, ohodnocením jedince.

* + 1. 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.

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.

* + 1. 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.

* + 1. 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 mutaci v 5% pravděpodobností. Populace je periodicky zlepšena 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.

* + 1. 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 jednoduší dotazy do databází, protože identifikátor je zároveň řádkem v tabulce. Tato reprezentace se velmi dobře podepsala na rychlosti algoritmu.

* + 1. Zobrazeni jedince

Poslední metodou, která byla potřeba, bylo zobrazení jedince pomocí ShowMe. Metoda ShowMe funguje podobně jako metoda GetFitness. Narozedíl od 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.

* 1. Krok Programovani Vnejsiho jedince

Dalším krokem byl vývoj Vnějšího jedince. Vnější jedinec byl mnohem problematičtější, protože pro výpočet jeho Fitness je nutné počítat fitness vnitřních jedinců.

Druhý zásadní problém byla samotná reprezentace vnějšího jedince. Vnitřní jedinec je reprezentován jako datový typ pole string. Nejprve byl vnější jedinec reprezentován jako pole string, kde každý string v sobě nesl informaci o vnitřním jedinci. Tento způsob fungoval, ale byl velmi komplikovaný pro křížení a mutaci.

* + 1. 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 ztrácely 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 měnili k jednomu podobnému výsledku a bylo obtížné dělat jakékoli změny v programu.

* + 1. 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. Vnější generace přitom funguje velice podobným způsobem jako vnitřní generace.

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.

* 1. Pomocne tridy

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.

* + 1. 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.

Tohle se ukázalo jako nepraktické a byly zvoleny unikátní identifikátory v desítkové soustavě.

* + 1. 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, nebo analyzovat v jiných aplikacích jako je Excel se ukázalo jako velmi vhodné.

* + 1. 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.

* + 1. 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 lehce 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.

1. Statistika

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.

* 1. Stare reseni full (Ok výsledky?)
  2. Stare reseni bez opravy (Spatny výsledky?)
  3. Nove reseni full (Top výsledky?)
  4. Nove reseni vnejsi jedinec 50+ vnitrni 20-
  5. Nove reseni vnejsi jedince 20- vnejsi 50+
  6. Pouze vnitrni jedinec

1. Zaver
2. Seznam literatury:
3. BARCUCH, Libor. *TRANSPORTNÍ ZAŘÍZENÍ PRO DOPRAVU KUSOVÝCH PŘEDMĚTŮ* [online]. Brno, 2017 [cit. 2019-04-22]. Dostupné z: <https://www.vutbr.cz/studenti/zav-prace/detail/100909> Vysoké učení technické Brno. Vedoucí práce Ing. Jiří Malášek
4. MICHAEL, Konečný. *Návrh robotického ramene* [online]. Brno, 2016/207 [cit. 2019-04-22]. Dostupné z: <https://www.vutbr.cz/studenti/zav-prace/detail/100537> . Vedoucí práce Tomáš Marada
5. BENDA, Tomáš. *Konstrukce a řízení manipulátoru* [online]. Pardubice, 2015 [cit. 2019-04-22]. Dostupné z: <https://portal.upce.cz/StagPortletsJSR168/PagesDispatcherServlet?pp_destElement=%23ssSouboryStudentuDivId_717&pp_locale=cs&pp_reqType=render&pp_portlet=souboryStudentuPagesPortlet&pp_page=souboryStudentuDownloadPage&pp_nameSpace=G3593&soubidno=16098> . Diplomová. Vedoucí práce Havlíček Libor
6. PROKOP, Michal. *Model třídicí linky v systému Tecnomatix Process Simulate* [online]. Brno, 2017 [cit. 2019-04-22]. Dostupné z: <https://www.vutbr.cz/studenti/zav-prace/detail/102661>. Vedoucí práce Václav Kaczmarczyk
7. VICAN, Pavel. *Třídící linka* [online]. Česká lípa, 2017 [cit. 2019-04-22]. Dostupné z: http://www1.fs.cvut.cz/stretech/2017/sbornik\_2017/pdf/16.pdf. Maturitní práce. Střední průmyslová škola, Česká Lípa. Vedoucí práce Josef Bašta.¨
8. KOUMARIS, Nick. C# Application Arduino Communication. *Educ8s.tv* [online]. [cit. 2019-04-22]. Dostupné z: <https://educ8s.tv/c-application-arduino-communication/>
9. 375668457. C# Serial Port Communication Arduino. *Instructables* [online]. [cit. 2019-04-22]. Dostupné z: <https://www.instructables.com/id/C-Serial-Communication-With-Arduino/>
10. *Historický vývoj automatizace? Poznejte 12 zásadních dat* [online]. [cit. 2019-04-21]. Dostupné z: <https://factoryautomation.cz/historicky-vyvoj-automatizace-poznejte-12-zasadnich-dat/>
11. [online]. [cit. 2019-04-21]. Dostupné z: <http://nd04.jxs.cz/419/332/3ce8c88974_74039634_o2.jpg>
12. [online]. [cit. 2019-04-21]. Dostupné z: <http://fyzweb.cz/materialy/sily/paka/kolec1.gif>
13. [online]. [cit. 2019-04-21]. Dostupné z: <http://www.j4.cz/uploads/tx_odphotogallery/thumbs/e4c6364f0a0aa08538edd5177649f77b.jpg>
14. [online]. [cit. 2019-04-21]. Dostupné z: https://www.zenavaute.cz/wp-content/uploads/vyroba.jpg
15. [online]. [cit. 2019-04-21]. Dostupné z: <http://www.newsreck.com/wp-content/uploads/2019/03/Global-Wafer-And-Integrated-Circuits-Market.jpg>
16. Maixner, L.; et al. Optoelektrické senzory binární. *Mechatronika*, 1st ed.; Computer Press: Brno, 2006; Chapter 3.2.1.6., p 49.
17. Maixner, L.; et al. Clonicí senzor rozměrů. *Mechatronika*, 1st ed.; Computer Press: Brno, 2006; Chapter 3.2.1.7, p 52.
18. Maixner, L.; et al. Clonicí senzor rozměrů. *Mechatronika*, 1st ed.; Computer Press: Brno, 2006; Chapter 3.2.1.7, p 53.
19. [online]. [cit. 2019-04-21]. Dostupné z: <http://www.riko.cz/image.php?nid=12547&oid=3490858&width=640&height=480>
20. [online]. [cit. 2019-04-21]. Dostupné z: <http://www.zakhodonin.cz/src/img/produkty/dopravniky/dopravniky/dopravnik_1.jpg>
21. [online]. [cit. 2019-04-21]. Dostupné z: <http://www.tmt.cz/cz/CMS/foto/1902.jpg>
22. [online]. [cit. 2019-04-21]. Dostupné z: <https://www.logsys.cz/files/th_8b1a7f5cccad127c10a8fd1f45573c4e.jpg>
23. [online]. [cit. 2019-04-21]. Dostupné z: <https://www.logsys.cz/files/c042868367d48ab0b6a02dfccc0a580a.jpg>
24. [online]. [cit. 2019-04-21]. Dostupné z: <https://www.feifer.cz/obrazky/1678acz.jpg>
25. [online]. [cit. 2019-04-21]. Dostupné z: <https://automatizace.hw.cz//files/styles/full/public/story_automat/11806/vyhybka_5.jpg?itok=2RpPhCv2>
26. [online]. [cit. 2019-04-21]. Dostupné z: <https://www.exasoft.cz/buddy-toys-bcr-10-robotic-arm-kit_ie561785.jpg>
27. SKALICKÝ, Tomáš. *Návrh a realizace robotického manipulátoru* [online]. Pardubice, 2015 [cit. 2017-10-15]. Dostupné z: <https://portal.upce.cz/StagPortletsJSR168/PagesDispatcherServlet?pp_destElement=%23ssSouboryStudentuDivId_745&pp_locale=cs&pp_reqType=render&pp_portlet=souboryStudentuPagesPortlet&pp_page=souboryStudentuDownloadPage&pp_nameSpace=G3593&soubidno=16238> . Vedoucí práce Havlíček Libor
28. [online]. [cit. 2019-04-21]. Dostupné z: <https://t3.ftcdn.net/jpg/01/10/03/22/240_F_110032210_m8lKqMJawvwCrlj2sD1kjB6s8P22klXN.jpg>
29. [online]. [cit. 2019-04-21]. Dostupné z: https://www.souepl.cz/wp-content/ucitele/valecka/at\_mega\_32\_popis%20(Opraveno)\_soubory/image004.jpg
30. [online]. [cit. 2019-04-21]. Dostupné z: <http://ww1.microchip.com/downloads/en/devicedoc/doc2503.pdf>
31. [online]. [cit. 2019-04-21]. Dostupné z: https://i.ebayimg.com/00/s/NDIzWDU4Mw==/z/q10AAOSwBRVae8zd/$\_35.JPG?set\_id=8800005007
32. [online]. [cit. 2019-04-21]. Dostupné z: <https://www.researchgate.net/profile/Shyam_Akashe/publication/257799438/figure/fig1/AS:341731565424653@1458486562832/MUX-graphical-symbol-a-truth-table-b.png>
33. [online]. [cit. 2019-04-21]. Dostupné z: <https://www.researchgate.net/figure/MUX-graphical-symbol-a-truth-table-b_fig1_257799438>
34. [online]. [cit. 2019-04-21]. Dostupné z: <https://assets.nexperia.com/documents/data-sheet/74HC_HCT4067.pdf>
35. [online]. [cit. 2019-04-21]. Dostupné z: <https://www.arduino.cc/en/tutorial/ShiftIn>
36. [online]. [cit. 2019-04-21]. Dostupné z: <https://www.arduino.cc/en/uploads/Tutorial/shftin_cd4021_pins.png>
37. [online]. [cit. 2019-04-21]. Dostupné z: <https://create.arduino.cc/projecthub/SAnwandter1/programming-8x8-led-matrix-23475a>
38. [online]. [cit. 2019-04-21]. Dostupné z: <http://www.lucky-light.com/Dot%20Matrix%20LED%20Displays/5x7_Dot_Matrix/KWM-50571.pdf>
39. [online]. [cit. 2019-04-30]. Dostupné z: <https://create.arduino.cc/projecthub/SAnwandter1/programming-8x8-led-matrix-23475a>
40. [online]. [cit. 2019-04-30]. Dostupné z: <https://images-na.ssl-images-amazon.com/images/I/51653PgxqgL._SX425_.jpg>
41. [online]. [cit. 2019-04-21]. Dostupné z: <https://static3.tme.eu/products_pics/5/4/d/54dcf8dd74a04d30c55078ca113eb5ac/363694.jpg>
42. [online]. [cit. 2019-04-21]. Dostupné z: <https://www.mylms.cz/text-krokovy-motor-princip/>
43. [online]. [cit. 2019-04-21]. Dostupné z: <https://www.mylms.cz/obrazky/elektronika/krokovy-motor-14.png>
44. [online]. [cit. 2019-04-21]. Dostupné z: <https://i.ebayimg.com/images/g/BXoAAOSweBFckLAn/s-l300.jpg>
45. [online]. [cit. 2019-04-21]. Dostupné z: <https://www.electricaleasy.com/2015/01/how-does-servo-motor-work.html>
46. [online]. [cit. 2019-04-21]. Dostupné z: <https://1.bp.blogspot.com/-WWEjibj3iX0/VK1LTqWI9NI/AAAAAAAABCs/FmiULNH04kg/s1600/servo%2Bmotor%2Bworking.png>
47. [online]. [cit. 2019-04-21]. Dostupné z: <https://i.ebayimg.com/images/g/DTQAAOSwFIRbxxlJ/s-l300.jpg>
48. [online]. [cit. 2019-04-21]. Dostupné z: <https://eluc.kr-olomoucky.cz/verejne/lekce/809>
49. [online]. [cit. 2019-04-21]. Dostupné z: <https://eluc.kr-olomoucky.cz/uploads/images/20238/obr.2.gif>
50. [online]. [cit. 2019-04-21]. Dostupné z: <https://i.ebayimg.com/images/g/AhEAAOSwUP5cLCBK/s-l640.jpg>
51. [online]. [cit. 2019-04-21]. Dostupné z: <https://eshop.merkurtoys.cz//data/product/178_352.jpg>
52. [online]. [cit. 2019-04-21]. Dostupné z: <https://images-na.ssl-images-amazon.com/images/I/61pnAVpZjuL._SY741_.jpg>
53. [online]. [cit. 2019-04-21]. Dostupné z: <https://pino-tech.eu/wp-content/uploads/2018/06/irlz44n.jpg>
54. [online]. [cit. 2019-04-21]. Dostupné z: <http://bildr.org/2012/03/rfp30n06le-arduino/>
55. [online]. [cit. 2019-04-21]. Dostupné z: <https://42bots.com/tutorials/bipolar-stepper-motor-control-with-arduino-and-an-h-bridge/>
56. [online]. [cit. 2019-04-21]. Dostupné z: <https://www.instructables.com/id/Arduino-How-to-Control-Servo-Motor-With-Arduino/>
57. [online]. [cit. 2019-04-21]. Dostupné z: <https://www.microchip.com/mplab/avr-support/atmel-studio-7>
58. [online]. [cit. 2019-04-21]. Dostupné z: <http://and-tech.pl/evb-4-3/>
59. [online]. [cit. 2019-04-21]. Dostupné z: <https://www.arduino.cc/en/main/software>
60. [online]. [cit. 2019-04-21]. Dostupné z: <https://www.arduino.cc/reference/en/language/functions/digital-io/digitalread/>
61. [online]. [cit. 2019-04-21]. Dostupné z: <https://www.arduino.cc/reference/en/language/functions/digital-io/digitalwrite/>
62. [online]. [cit. 2019-04-21]. Dostupné z: <https://i.ebayimg.com/images/g/FAcAAOSwLfRbPg0F/s-l640.jpg>
63. [online]. [cit. 2019-04-30]. Dostupné z: <https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcTYE4F_YMtfyQwVEHx_Ei8wSmfKx58ZBC7T0jfNmKw3UwCVjEZjFg>
64. [online]. [cit. 2019-04-21]. Dostupné z: <https://www.hackster.io/ianabcumming/arduino-simple-camera-slider-electronics-9e9d8b>
65. [online]. [cit. 2019-04-22]. Dostupné z: <https://www.arduino.cc/reference/en/language/structure/control-structure/while/>
66. [online]. [cit. 2019-04-22]. Dostupné z: <https://www.arduino.cc/reference/en/language/structure/control-structure/if/>
67. [online]. [cit. 2019-04-22]. Dostupné z: <https://www.arduino.cc/en/Tutorial/SerialEvent>
68. [online]. [cit. 2019-04-22]. Dostupné z: <https://www.arduino.cc/en/Reference/Loop?setlang=it>
69. [online]. [cit. 2019-04-22]. Dostupné z: <https://visualstudio.microsoft.com/downloads/?rr=https%3A%2F%2Fwww.google.cz%2F>