VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

OPTIMALIZACE POMOCÍ MRAVENČÍCH ALGORITMŮ

OPTIMALIZATION USING ANT ALGORITHMS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **MATĚJ VÁLEK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. MICHAL BIDLO, Ph.D.**
SUPERVISOR

BRNO 2018

Abstrakt

Tato bakalářská práce se zabývá různými aplikacemi optimalizací mravenčí kolonií. Zejména algoritmus ant colony system bude použit pro optimalizaci problému obchodního cestujícího a návrh pravidel pro vývoj celulárních automatů. Výsledky budou statisticky analyzovány. Kromě toho byla vytvořena GUI aplikace, která umožňuje interaktivně sledovat vývoj algoritmu ant colony system pro vzdělávací účely.

Abstract

This bachelor thesis will deal with various applications of ant colony optimisation. In particular, Ant Colony System will be applied on the optimisation of traveling salesman problem and the design of rules for the development of cellular automata. The obtained results will be statistically analysed. Moreover, a GUI-based application has been developed which allows to interactively observe the progress of Ant Colony System for the educational purposes.

Klíčová slova

optimalizace mravenčí kolonií, problém obchodního cestujícího, celulární automat, problém synchronizace

Keywords

ant colony optimalization, traveling salesman problem, celular automatas, synchronization problem

Citace

VÁLEK, Matěj. *Optimalizace pomocí mravenčích algoritmů.* Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michal Bidlo, Ph.D.

Optimalizace pomocí mravenčích algoritmů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Michala Bidla. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Matěj Válek
13. května 2018

Poděkování

Děkuji Ing. Michalu Bidlovi za pomoc při vedení bakalářské práce, cenné rady, věcné připomínky a vstřícnost při konzultacích. Tato práce byla podpořena Ministerstvem školství, mládeže a tělovýchovy z podpory Velkých infrastruktur pro výzkum, experimentální vývoj a inovace v rámci projektu IT4Innovations národní superpočítačové centrum - LM2015070.

Obsah

Kapitola 1

Úvod

Mravenčí algoritmy byly poprvé představeny v roce 1992 v doktorandské práci Marca Doriga [3]. Přesto, že to je téměř 25 let starý nápad, doteď se stále algoritmy tohoto typu modifikují a používají. Na poli umělé inteligence se s nimi hojně pracuje na řešení výpočetních problémů. Původní algoritmus byl vytvořen pro hledání co nejvíce optimálního řešení problému obchodního cestujícího (Traveler salesman problem TSP).

Název těchto algoritmů nebyl vymyšlen náhodně. Dorigo se při vývoji úplně prvního algoritmu tohoto typu silně inspiroval přírodou. Konkrétně mravenců a jejich sociálního chování při hledání potravy. Při zkoumání jejich způsobu hledání a následného nošení jídla do mraveniště si uvědomil, že tyto postupy mravenců by se dali použít pro nalezení řešení některých složitých problémů, které se v té době nalézaly ve světě umělé inteligence. Tyto algoritmy byly nakonec použity na veliké spektrum úkolů v několika oblastech jako například:

- vehicle routing problem a jeho varianty, například v aplikacích

  Job-shop scheduling problem (JSP) [9].

  Open-shop scheduling problem (OSP) [11].

- scheduling problem a jeho varianty, například v aplikacích

  Capacitated vehicle routing problem (CVRP) [16].

  Multi-depot vehicle routing problem (MDVRP) [13].

- assignment problem a jeho varianty, například v aplikacích

  Quadratic assignment problem (QAP) [14].

  Generalized assignment problem (GAP) [12].

Mravenčí algoritmy tedy byly v uplynulých letech aplikovány na různé typické problémy z oblasti optimalizace. Prvním vyvinutým algoritmem byl Ant system. V dalších letech vznikly různé pokročilejší varianty, z nichž asi nejznámější je algoritmus Ant colony system.

Tato práce je rozdělena na tři části. První část je implementace algoritmu z rodiny Ant colony optimalization, konkrétně Ant colony system, který je pokročilejší variantou původního Dorigova algoritmu Ant system z roku 1992. Cílem bude aplikovat tento pokročilejší algoritmus na problém obchodního cestujícího a zaznamenat statistická data při hledání co nejlepšího řešení pro různá zadání a různou konfiguraci. Tímto se myslí, že daný problém bude mít různý počet měst určených k prozkoumání. Konkrétně se jedná o rozsah 15-ti až

194 měst. Bude také provedeno testování, jak přesný je tento algoritmus a jak se chová pro daný set měst.

Druhý cíl práce je spjat s využitím ant colony onptimalization (ACO) algoritmů v oblasti simulací a návrhů celulárních automatů. Konkrétně se tato práce bude zajímat o výzkum a modifikaci Ant colony system algoritmu u problému, který se nazývá problém sychronizace. Tento problém je založený na nalezení takové transformační funkce, aby po určitém konečném počtu kroků CA procházel periodicky se opakujícími stavy homogenní 1 a homogenní 0. Cílem tedy bude zkoumání využití grafu pro nalezení transformační funkce, zkonstruování co nejlepší matematické funkce pro ohodnocení postupných výsledků iterací a také následné vyhodnocení statistických údajů a zkoumání chování aplikace při dané konfiguraci.

Třetí a poslední cíl je vytvoření GUI aplikace pro výukové účely. Výsledný program bude co nejjednodušeji a nejpřehledněji demonstrovat krok po kroku chování Ant colony system algoritmu na problému obchodního cestujícího. Aplikace bude znázorňovat postup výpočtu jak na grafu, tak i matematicky.

Kapitola 2

Mravenčí algoritmy

Mravenčí algoritmy představují podmnožinu tzv. algoritmů kolektivní výpočetní inteligence, která čerpá hlavní inspiraci z chování společenstev mravenců v přírodě. Hlavním smyslem je napodobit mravence při hledání potravy, kdy procházením různých cest a značkováním nadějných tras feromony jsou ostatními mravenci upřednostňovány efektivnější trasy (např. ty kratší nebo ty, které vedou k bohatším zdrojům potravy), díky tomu mravenci dokáží proces obstarávání potravy přirozeně optimalizovat. V této kapitole podrobně popíšeme analogii mravenčích algoritmů a jejich aplikace na řešení vybraných technických úloh.

2.1 Od živých mravenců po ty umělé

Mravenčí kolonie v reálném světě jsou velice sofistikované společnosti hmyzu. I když se může zdát, že to jsou velice primitivní živočichové, představují vysoce strukturovanou společenskou organizaci. Účelem této organizace je řešení problémů jako je například nalezení potravy, či chránění svého teritoria. Tuto vlastnost organizace, která má za následek koordinované chování mravenců, je možné využít pro chování umělých agentů, kteří řeší výpočetní problémy. I když jsou mravenci téměř slepí, komunikují se sebou pomocí takzvanému feromonu. Je to forma nepřímé komunikace, kdy mravenec vypouští na zem po cestě biologickou látku, která má určité vlastnosti. Biologové ukázali, že mnoho koloniálních skupenství živočichů se chová na základě stigmergie (nepřímého dorozumívání). Jinými slovy se dá říci, že lze dosáhnout vysokého stupně organizace i za pomoci takto primitivního prostředku.

Jako lidská rasa spoléháme hlavně na našich 5 hlavních smyslů. Čich, sluch, zrak, hmat a chuť. Ty nám dávají velikou výhodu. Je pro nás nepředstavitelné, abychom pracovali pouze s jediným hlavním smyslem. Na světe existuje mnoho druhů mravenců, ale jedno mají všichni společné. Dorozumívají se pomocí feromonu zvaný *trail pheromone* [2] [15]. Je to speciální druh feromonu, pomocí kterého mravenci značí cesty například od zdroje jídla k mraveništi. Jednotlivci si vybírají cestu podle toho, jak silná tato stopa je. Uvedeme si příklad.

Mravenec jde hledat potravu. Při hledání vypouští po trase průměrnou hodnotu feromonu až dokud nenajde jídlo. Na zpáteční cestě do mraveniště vypouští už jiné množství této látky, a to na základě dvou atributů. Jaké množství potravin našel a jak je to daleko od mraveniště. Řekněme, že jeden mravenec najde výrazně lepší zdroj potravy než druhý, tak poté vypustí značně více feromonu na cestě zpět než ten druhý. Ostatní mravenci toto poznají a dají se cestou, na které je větší koncentrace feromonu. Čím více kvalitní zdroj potravy, tím více mravenců k němu půjde a tím lepší feromonová stopa. Pokud ale tento

zdroj vytěží, každý mravenec, který došel k tomuto dotěženému zdroji, přestane (nebo úplně minimálně) vypouštět feromon při zpáteční cestě.

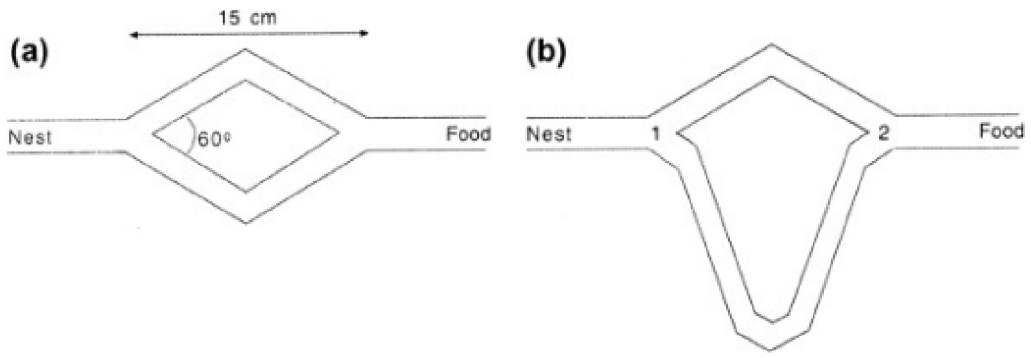
Tato chemikálie má vlastnost, že po nějaké době začne vyprchávat. Tudíž už další mravenci touto cestou přestanou postupně chodit. Chování mravenců při výběru cest, kterými se vydají, můžeme rozdělit na dva typy. "exploration", jinými slovy nikde není žádná stopa dostatečně silná tak, aby to mravence zaujalo a vyrážejí tedy náhodnými směry hledat potravu. Druhý typ se nazývá "exploitation", tedy vykořisťování. To znamená, že mravenec našel kvalitní zdroj potravy a většina mravenců se vydává za tímto zdrojem tou samou cestou.

2.2 Experiment dvou mostů

Takzvaný Double bridge experiment uskutečnil v roce 1989 pan Deneubourg a jeho kolegové (Deneubourg, Aron, Goss, and Pasteels, 1990; Goss et al., 1989) [2]. Experiment spočíval v tom, že izolovali mravenčí hnízdo a propojili jej se zdrojem jídla pomocí dvou oddělených cest. Na tento experiment aplikovali různý poměr délky obou cest

$$r = \frac{l_l}{l_s} \tag{2.1}$$

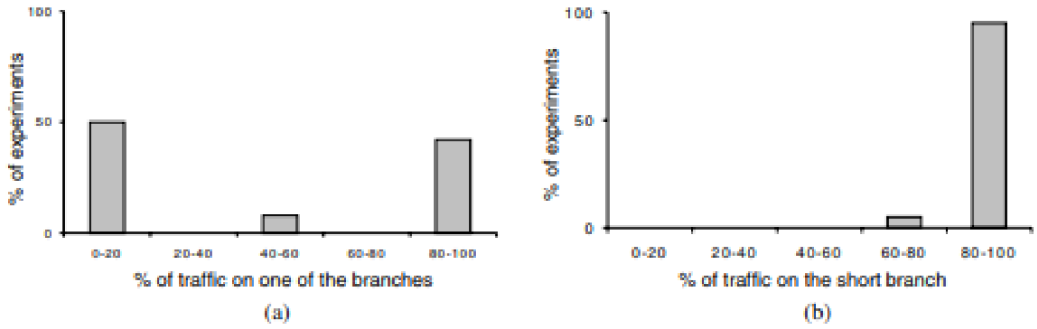
kde l_l je délka delší cesty a l_s délka kratší cesty. První experiment, který provedli s hodnotou $r = 1$, tedy obě cesty měly stejnou délku, je znázorněn na obr 2.1(a).



Obrázek 2.1: Ukázka poměru délky cest u experimentu dvojitého mostu. (a). obě cesty jsou stejně dlouhé, (b). Rozdílná délka cest. [15]

Výsledkem nastavení rovnosti délek obou cest bylo, že ze začátku si mravenci volili cestu náhodně. Ovšem postupem času začali mravenci favorizovat pouze jednu z cest i přesto, že obě cesty jsou stejně dlouhé. Na počátku experimentu měly obě cesty hodnotu feromonu rovnu 0. Mravenci tedy s pravděpodobností 0.5 pro každou cestu vybírali svou pouť čistě náhodně (exploration). Ve skutečnosti se však v průběhu experimentu ukázalo, že počet mravenců na jedné cestě lehce převažoval, což je důsledek pravděpodobnostní fluktuace. Mírná převaha mravenců na jedné cestě mělo za následek větší míru vypuštěného feromonu

na dané stezce. To bylo důvodem, proč se daná cesta stala pro mravence dominantní (exploitation). V druhém pokusu byl nastaven poměr na hodnotu $r = 2$. Tedy jedna cesta byla dvakrát delší než druhá. Mravenci po velice krátké době porozuměli tomu, že jedna cesta je výhodnější a začali využívat lepší cestu.



Obrázek 2.2: Dalším aspektem (téměř okamžité) preference kratší cesty je, že jí za jednotku času stihne projít více mravenců, což má za následek rychlý nárůst koncentrace feromonů a následný výběr této cesty ostatními mravenci [15].

2.3 Umělí mravenci

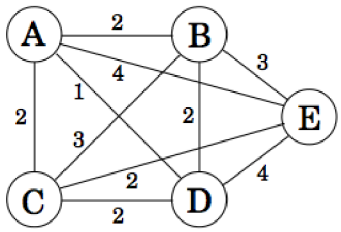
Experiment dvojitého mostu jasně ukázal, že kolonie mravenců mají optimalizační potenciál. Ovšem pro využití ve výpočetních problémech je zapotřebí změnit pár věcí, což obvykle vychází z požadavků implementace umělých mravenců na počítači. Hlavní rozdíly mezi umělými a živými mravenci jsou následující:

- Mravenci chodí v reálném/spojitém prostředí asynchronně, umělí mravenci se pohybují v diskrétním prostředí synchronně.
- Živí mravenci se řídí feromonovými stopami při návratu do mraveniště, umělí mravenci se vrací do mraveniště v každém cyklu po stejných cestách, kterými se v této iteraci pohybovali od mraveniště.
- Živí mravenci vypouští feromon pořád, umělí mravenci vytváří pachovou stopu pouze při návratu do mraveniště.
- Chování živých mravenců je založeno na implicitním vyhodnocení cest. To spočívá v tom, že pohyb po kratších cestách trvá mravencům kratší dobu, proto mohou tyto cesty opakovat častěji a tím se množství feromonu na těchto cestách zvyšuje. Umělí mravenci vyhodnocují cesty explicitně, a to při návratech do mraveniště podle kvality a délky cesty.
- Skuteční mravenci jsou téměř slepí, umělí mravenci jej mají.
- Umělí mravenci mají paměť.
- Umělí mravenci se nazývají agenti. [1]

[1] `https://www.fit.vutbr.cz/study/courses/IZU/private/1718izu_4.pdf`

2.4 Problém obchodního cestujícího

Podstatu a implementaci základního mravenčího algoritmu (Ant system) si popíšeme na řešení problému obchodního cestujícího (TSP), na který byl původně tento algoritmus sestrojen. U TSP si klademe následující otázku. Při daném počtu měst a množin cest, které propojují každé město s každým, jaká je nejkratší možná cesta, kterou urazí cestující, jenž navštívil všechna města z této dané množiny právě jednou s návratem do města, ze kterého startoval? Jedná se o jeden z NP těžkých problémů v oblasti kombinatorické optimalizace. Pro potřeby této práce bude problém TSP reprezentován jako úplný ohodnocený neorientovaný graf. Města jsou reprezentovány uzly a cesty hranami grafu. Délka hrany zastupuje váhu hrany. Pro tento problém existují dva různé přístupy k typu grafu. Symetrický a nesymetrický. Při symetrickém grafu vzdálenost x mezi oběma městy i a j je definována jako: $x_{ij} = x_{ji}$. Při tomto přístupu je graf neorientovaný. Druhý přístup značí formule $x_{ij} \neq x_{ji}$ a tento graf je orientovaný. V této práci se u TSP bude využívat neorientovaný graf a u problému synchronizace orientovaný.



Obrázek 2.3: Ukázka grafu pro TSP s pěti městy, kde je vidět ohodnocení hran (vzdálenosti), úplnost grafu (každé město je spojeno se všemi ostatními městy) a neorientovanost (hrany bez šipek) [2]

2.5 Ant system

Ant system (AS) [6] je první algoritmus rodiny ACO a byl navržen pro řešení Problému obchodního cestujícího. Hlavní myšlenka je mít několik mravenců (agentů). Agenti pracují skrz feromony a globální komunikaci. V každé iteraci se mravenci jako první náhodně rozmístí do měst. Poté každý mravenec vytváří iterativně částečné řešení problému metodou zkoumání (exploration) nebo aplikačně specifickou heuristikou (exploitation). Každý agent vygeneruje kompletní cestu vybíráním měst pomocí pravděpodobnostní funkce, která se nazývá pravidlo přechodů, neboli *state transition rule*. Mravenci preferují cesty, které jsou krátké a obsahují vysokou hodnotu feromonů. Jakmile všichni mravenci dokončí svou cestu, aplikuje se tzv. globální aktualizační pravidlo, anglicky *global pheromone updating rule*, přičemž se na všech hranách aktualizují feromonové hodnoty. Je třeba ale uvést, že zlomek feromonových hodnot na hranách se vypaří. Celý cyklus se opakuje. Po ukončení předem daného počtu iterací se nalezne výsledek.

Hlavní charakteristikou AS je, že v každé iteraci jsou hodnoty feromonů na hranách grafu upravovány všemi mravenci, kteří vytvořili nějaké řešení [5]. Pro pochopení algoritmu je potřeba si určit definici některých prvků. Hlavní cyklus se značí c. Určuje, kolik iterací

[2] Zdroj obrázku: `https://www.quora.com/How-do-I-solve-the-traveling-salesman-problem-using-dynamic-programming`

samotného algoritmu program provede. Intenzita feromonů na hranách grafu mezi městy i a j se značí τ_{ij}. Hodnoty feromonů se aktualizují pomocí globálního aktualizačního pravidla viz 2.2 [4].

$$\tau_{ij} \leftarrow (1-\rho)\cdot\tau_{ij} + \sum_{k=1}^{m}\Delta\tau_{ij}^{k} \tag{2.2}$$

Kde konstanta $\rho \in <0,1>$ značí míru vypařování feromonů a m je počet mravenců. $\Delta\tau_{ij}^{k}$ je množství feromonu přidaného k hraně (i,j) mravencem k. a je dáno vztahem 2.3

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{Q}{L_k} & \text{pokud mravenec k prošel hranou}(i,j) \\ 0 & jinak \end{cases} \tag{2.3}$$

kde Q je konstanta a L_k je součet délky kompletní cesty mravence k. Při vytváření řešení si mravenci vybírají následující město, do kterého se přesunou pomocí stochastického mechanismu. Mravenec, nacházející se v městě i, volí svůj další krok do města j na základě pravděpodobností vypočtených funkcí 2.4 pro každou hranu $(i,l) \in N(s_k)$ vedoucí z města i do města l z množiny $N(s_k)$ což je množina dosud nenavštívených měst mravence k [4].

$$p_{ij}^{k} = \begin{cases} \frac{\tau_{ij}^{\alpha}\cdot\eta_{ij}^{\beta}}{\sum_{c_{il}\in N(S_k)}\tau_{ij}^{\alpha}\cdot\eta_{ij}^{\beta}} & \text{jestliže } c_{ij} \in N(s_k), \\ 0 & jinak, \end{cases} \tag{2.4}$$

Konstanty α a β kontrolují relativní význam feromonů a heuristickou informaci η_{ij}, která určuje viditelnost mezi městy i,j a vypočítá se pomocí vztahu $\eta_{ij} = \frac{1}{d_{ij}}$, kde d_{ij} je vzdálenost mezi danými městy. Je také nutno dodat podmínku $\alpha > 0, \beta > 0$.

2.6 Ant colony system

I když AS byl úspěšný v hledání dobrého či optimálního řešení pro malé TSP problémy (max. 30 měst), čas který k tomu byl zapotřebí nebyl optimální. Pro větší problémy byly uskutečněny tři hlavní změny za účelem vylepšení výkonu. Tyto změny vedly k definici ACS, tedy Ant colony system. ACS se liší oproti algoritmu Ant system v těchto hlavních oblastech [5]:

1. Pravidlo přechodů zde algoritmu umožňuje specifikovat dobrou rovnováhu mezi zkoumáním nových cest (exploration) a využíváním heuristické informace, tj. vzdálenosti mezi městy (exploitation) pomocí nově zavedeného parametru.

2. Zavádí globální aktualizační pravidlo, které je aplikováno pouze na cestu mravence, který za danou iteraci vyzkoumá nejlepší řešení.

3. Při konstruování řešení mravenci je aplikováno další tzv. lokální aktualizační pravidlo.

Neformálně tedy je ACS velice podobný algoritmu AS: m mravenců je inicializačně umístěno do n měst vybraných nějakým inicializačním pravidlem (např. náhodně). Každý mravenec opět vykonstruuje řešení aplikováním pravidla přechodů. Během hledání řešení ale upravuje hladinu feromonů na hranách, které při své cestě navštívil lokálním aktualizačním pravidlem. Poté, co všichni mravenci vytvoří nějaké řešení, se spočítají jejich délky. Mravenec s nejkratší cestou aplikuje globální aktualizační pravidlo. Podobně jako v AS

jsou mravenci ovlivněni délkou hran a hodnotami jejich feromonů. Hrany s velikých množstvím feromonů jsou pro ně velice lákavé. Aktualizační pravidla jsou zde nastavená tak, aby přidávala velké množství feromonů na hrany, kterými mravenci procházejí častěji.

2.6.1 Rozhodování funkce

Pravidlo přechodů se nazývá náhodně proporcionální pravidlo a jak už bylo řečeno, udává nám, s jakou pravděpodobností se vydá mravenec k z města r do místa s, viz rovnice (2.4). Nyní se na základě této pravděpodobnosti musí rozhodnout, kam se tedy mravenec doopravdy vydá. V algoritmu ACS se mravenci rozhodují dvojím způsobem [5] (volba následujícího města z uzlu, ve kterém se právě nacházíš, je dána vztahem (2.5)

$$s = \begin{cases} \arg\, max_{u \in j_k(r)}[\tau(r,s)] \cdot [\eta(r,s)]^{\beta}, & \text{jestliže } q \leq q_0 (\text{exploitation}) \\ \text{jinak pravděpodobnostní výběr dle 2.4.} \end{cases} \tag{2.5}$$

kde q je náhodně vygenerované reálné číslo uniformním náhodným generátorem v rozsahu $[0...1]$. q_0 je předem zadaný parametr programu. Jinými slovy, parametr q_0 udává pravděpodobnost, že mravenec se přesune do města, do něhož vede hrana s nejvýhodnější kombinací feromonů a heuristické informace (hrana s nejvyšší pravděpodobností přesunu). Jak už bylo řečeno, větší pravděpodobnost výběru následující destinace mravence pomocí tohoto principu mají kratší hrany s větší hodnotou feromonu. Ale pořád můžeme korigovat průzkum mravenců pomocí konstanty q_0, kdy čím menší číslo, tím více se mravenci nebudou ubírat nejlepší cestou a budou prozkoumávat i jiné možnosti. Konstanta q_0 je aplikačně závislá a její hodnota je volena experimentálně či na základě doporučení z literatury. Při špatné hodnotě nemusí algoritmus dojít ke kvalitním výsledkům. Implementovaný způsob rozhodování mravenců v této práci je vidět v algoritmu 3.

2.6.2 Globální aktualizační pravidlo

Po ukončení cest všech mravenců se aplikuje globální aktualizační pravidlo. To znamená, že pouze mravenec, který ušel nejkratší vzdálenost může aktualizovat feromonové hodnoty na hranách, kterými prošel. Feromonová hladina se mění na základě pravidla (2.6). Feromonový úbytek značí konstanta $0 < a < 1$. Konstantou L_{gb} se označuje délka nejlepší cesty v dané iteraci [5].

$$\Delta\tau(r,s) = (1-a) \cdot \tau(r,s) + a \cdot \Delta\tau(r,s) \tag{2.6}$$

kde

$$\Delta\tau(r,s) = \begin{cases} (1/L_{gb}), & \text{if } (r,s) \in \text{Je hranou nejlepší cesty} \\ 0, & jinak \end{cases} \tag{2.7}$$

2.6.3 Lokální aktualizační pravidlo

Kromě globálního pravidla v Ant system se začalo aplikovat i tzv. lokální aktualizační pravidlo. Pokaždé, co mravenec při své cestě projde nějakou hranou, aplikuje na feromonovou hodnotu pravidlo (2.8), kde konstanta $0 < \rho < 1$ také značí feromonový úbytek [5].

$$\Delta\tau(r,s) = (1-\rho) \cdot \tau(r,s) + \rho \cdot \Delta\tau(r,s) \tag{2.8}$$

Výraz $\Delta\tau(r,s)$ je na počátku nastaven na vhodnou počáteční hodnotu r_0, kde r_0 je iniciační hodnota feromonů. Základní princip algoritmu ACS implementovaný v této práci

je popsán v algoritmu(1).

Algoritmus 1: Ant colony system

Input: $(c, \alpha, \beta, \rho, r_0, m, q_0, a)$

```
1: nastav hodnoty feromonů na všech hranách na hodnotu r_0
2: for i = 0 to c do
3:     Náhodně umísti mravence
4:     for j = 0 to Number of cities do
5:         for k = 0 to m (Number of ants) do
6:             State_transition_rule(k,α,β,q_0)
7:             Lokal_update_rule(k,r_0,ρ)
8:         end for
9:     end for
10:    Find_best_route()
11:    Global_update_rule(a)
12: end for
```

Kapitola 3

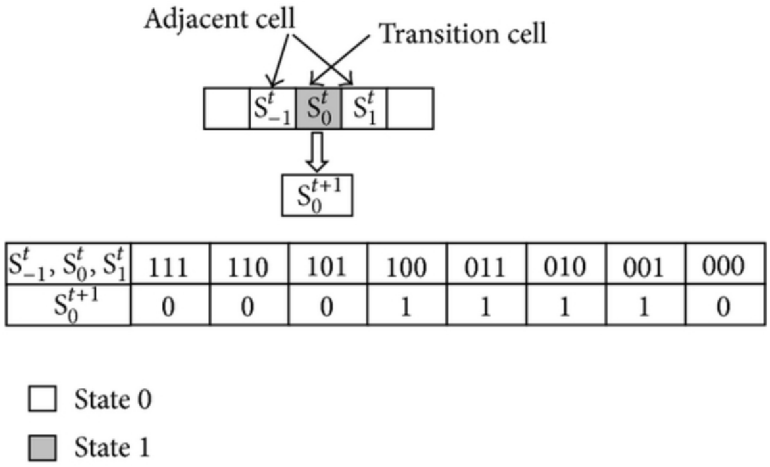
Aplikace mravenčích algoritmů pro návrh celulárních automatů

3.1 Úvod do celulárních automatů

Za celou dobu vývoje výpočetní techniky se vyvinulo mnoho systémů, jejichž akce jsou jednoduché, ale mají veliký výpočetní potenciál. Celulární automaty (CA) byly oficiálně koncipovány Johnem von Neumanem za účelem zjištění fungování komplexních systémů [17]. První důležitým faktorem popularizace celulárních automatů byla hra Life od britského matematika Johna Hortona Conwaye [7]. CA jsou dynamické diskrétní systémy. Diskrétní se myslí jejich prostor a čas. V dnešní době se používají hlavně v simulacích velkého spektra problémů jako například šíření virů, plynů do okolí a spousty dalších například fyzikálních jevů.

3.1.1 Celulární automaty a transformační funkce

Celulární automat je pole buněk, kde tyto buňky nabývají stavu z nějaké konečné množiny. Stavy buněk jsou aktualizovány v každé iteraci, která se provádí, jak už bylo řečeno, v diskrétním čase. Následující stav buňky se určuje pomocí stavu buňky, která se má transformovat a stavů buněk v jejím okolí. Celkový prostor automatu může nabývat n dimenzí, kde n je celé číslo. V praxi se nejčastěji používají hodnoty $n = 1, 2, 3$. Jednotná pravidla pro transformace stavů buňky je možno definovat jako konečný stavový automat, kde pravidla jsou například stanovená ve formě tabulky nebo pole. Soubor těchto pravidel se nazývá **transformační funkce**. Jelikož v této práci se bude často používat slovo index, je potřeba si vysvětlit, co zde znamená. Index zde bude mít stejný význam jako v programování. Transformační funkci pro jednodimenzionální automat si můžeme představit jako datové pole, jehož položky obsahují stavy, na který se má transformovaná buňka transformovat. Index tedy bude ukazovat do paměti na nějakou položku v poli transformační funkce. Příklad můžeme vidět na obrázku 3.1.



S^t_{-1}, S^t_0, S^t_1	111	110	101	100	011	010	001	000
S^{t+1}_0	0	0	0	1	1	1	1	0

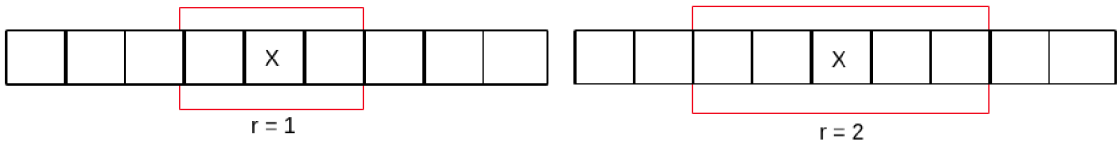
Obrázek 3.1: Ukázka transformační funkce pro jednodimenzionální automat s okolím $r = 1$, Výsledky transformační funkce pro dané indexy jsou zapsané v poli. Kombinace hodnot buňky a okolí nám dá dohromady index do pole transformační funkce, který obsahuje budoucí hodnotu stavu buňky S_0[1].

Vstup této funkce jsou hodnoty stavů buněk v předem definovaném okolí a hodnota stavu buňky připravené k transformaci. Při shodě kombinací vstupních hodnot s pravidlem (indexem) v tabulce se nalezne výsledná hodnota transformace, jenž dané pravidlo představuje. Pravidla transformační funkce kryjí všechny možné kombinace vstupů. V obrázku 3.1 označení S_0 reprezentuje buňku, která má být transformována a buňky označené jako S_{-1}, S_1 jsou součástí okolí buňky S_0.

Celulární automaty z hlediska transformační funkce můžeme rozdělit na dva druhy. Uniformní a neuniformní. V této práci se budeme zabývat uniformními automaty. To znamená, že pro všechny buňky v automatu platí stejná transformační pravidla. Existuje tedy pouze jedna transformační funkce pro celý automat.

3.1.2 Buňka a její okolí

Tato práce bude pracovat výhradně s jednorozměrnými CA, kde okolí každé buňky zahrnuje r jejích sousedů bezprostředně nalevo a napravo od této buňky a danou buňkou samotnou. Parametr r se nazývá rádius a v této implementaci CA použijeme rádius $r = 2$. Každá buňka v celém celulárním automatu tedy má $2r + 1$ buněk, které patří do jejího okolí (pokud tedy obsahuje cyklické okrajové podmínky) viz obrázek 3.2 [1]:



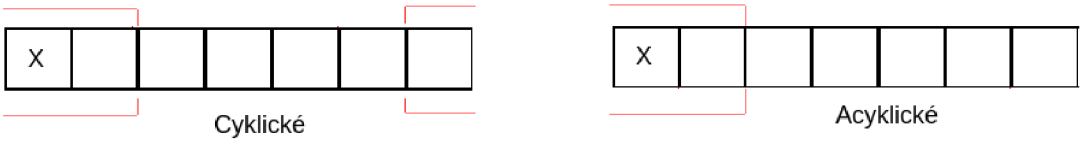
Obrázek 3.2: Ukázka okolí pro 1D celulární automat, $r = 1$ má celkem 3 buňky v okolí. Při $r = 2$ má buňka celkem 5.

[1]Zdroj: https://www.researchgate.net/figure/Example-of-the-transition-rule-in-the-one-dimensional-cellular-automata-system-The_fig6_276856575

3.1.3 Okrajové podmínky celulárních automatů

Představme si ale buňku, která se nachází na samotném okraji prostoru CA. Celulární automat se ale chápe jako neomezený prostor. Jak tedy ohodnotit buňky, které prakticky ani v našem programu neexistují? S tímto problémem se dá vypořádat několika způsoby. Rozdělujeme je na spojité, adiabatické, reflexní, absorbující a určené hodnotou [8].

- Spojité – U 1D automatu se jedná o smyčku, tedy sousední buňka té, která leží na prvním nebo posledním indexu je buňka z opačného pólu. u dimenze $n = 2$ potom se toto okolí nazývá anuloid
- Absorbující – předpokládá se, že žádné buňky za hranicí automatu nejsou

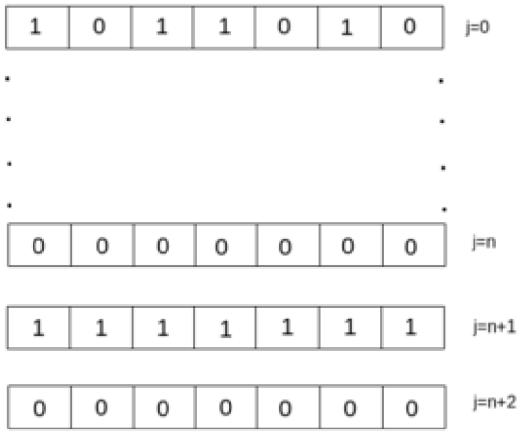


Obrázek 3.3: Ukázka okolí pro spojité (cyklické) a absorbující (acyklické) okrajové podmínky

3.2 Problém synchronizace

Pro potřeby této práce budeme definovat problém synchronizace v CA následovně. Je dán takový celulární automat, kde jeho buňky mohou nabývat pouze stavu 1 nebo 0 s konečným počtem buněk, vhodně zvolenou počáteční konfigurací a nějakým buněčným okolí. Řešením problému synchronizace je nalezení takové přechodové funkce CA, podle které z dané počáteční konfigurace CA dospěje za konečný počet kroků do střídajících se homogenních stavů 0 a 1 (tj. v jednom okamžiku jsou všechny buňky ve stavu 1, v dalším ve stavu 0 atd.), přičemž jako první může nastat libovolný z homogenních stavů [10]. Příklad, kdy v celulárním automatu nastane po nějakém počtu iterací stav synchronizace ilustruje obr 3.4.

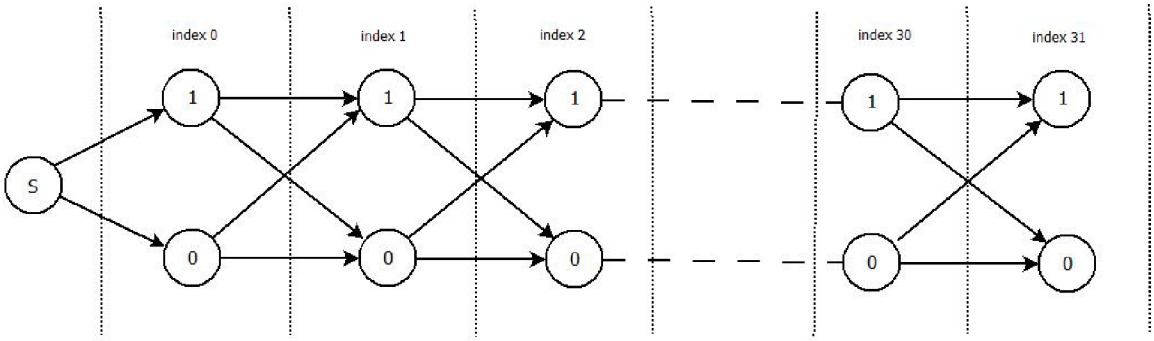
I když v této práci bude předmětem zkoumání pouze binární automat s okolím $r = 2$, jeho transformační funkce celkově obsahuje $2^5 = 32$ prvků pole, tedy 32 indexů ukazujících do pole transformační funkce nám dává $2^{32} = 4,294,967,296$ možných transformačních funkcí. Při náročnějším zadání by bylo tedy obtížné počítat tento problém hrubou silou a s rostoucím okolím či množinou hodnot stavů buněk výpočetní náročnost tohoto problému roste. Cílem tohoto výzkumu bude ověření schopnosti návrhu transformační funkce CA pomocí mravenčího algoritmu pro výše formulovaný problém synchronizace.



Obrázek 3.4: Ilustrační ukázka synchronizace v celulárním automatu pro šířku CA 7 s nespecifikovaným okolím. J označuje číslo iterace transformací, tedy kolikrát už proběhla transformace celého pole.

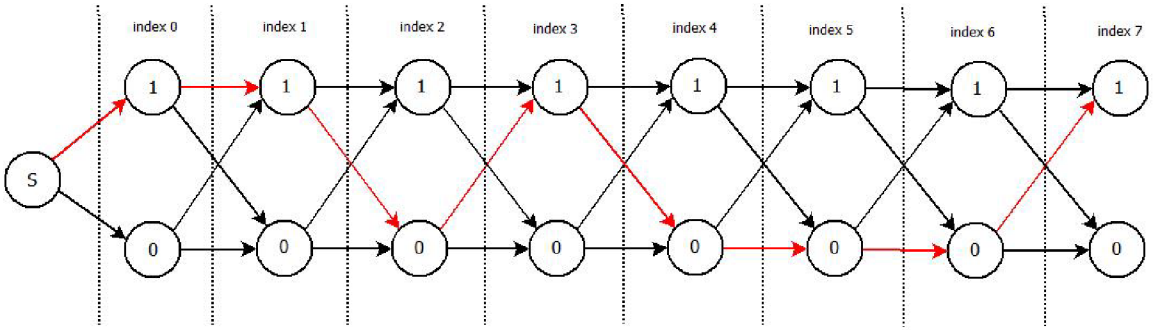
3.2.1 Graf pro nalezení transformační funkce

Jak už bylo řečeno, agenti (mravenci) u TSP hledají řešení v úplném grafu, jehož hrany mají nějakou váhu. Jelikož mravenčí algoritmy nebyly k návrhu CA určeny, je potřeba jejich koncept poněkud upravit. Graf, kterým mravenci procházejí a hledají řešení transformační funkce, nebude úplně propojený (každý uzel nebude spojen se všemi ostatními uzly). Kvůli okolí $r = 2$ bude graf rozdělen na 33 částí, kdy první část je startovací bod a zbylých 32 částí reprezentuje prvky v poli transformační funkce pro binární CA. Každá část je pak rozdělena na dva uzly, které zastupují hodnotu daného stavu, tedy jeden uzel nabývá hodnoty 1 a druhý 0. Graf je orientovaný, tudíž mravenec se nemůže vrátit a musí pouze postupovat do části reprezentující vyšší hodnotu indexu než se právě nachází viz obr. 3.5.



Obrázek 3.5: Graf pro nalezení všech hodnot prvků transformační funkce pro okolí $r = 2$ pomocí mravenčího algoritmu.

na obr 3.6. je uveden příklad průchodu mravence grafem pro okolí $r = 1$. Agent začíná ve startovacím bodě S. Následně postupuje po indexech až do konce. na konci se z jeho cesty vytvoří transformační funkce. Konkrétně v tomto případě agent vytvořil transformační funkci v tabulce(3.1)



Obrázek 3.6: Ukázka průchodu mravence grafem pro okolí $r = 1$, tedy transformační funkce tvoří pole o $2^3 = 8$ prvcích.

S_{-1}, S_0, S_1	000	001	010	011	100	101	110	111
S_0	1	1	0	1	0	0	0	1

Tabulka 3.1: Příklad transformační funkce vytvořené mravencem průchodem grafem 3.6

3.2.2 Aplikace ACS na problém synchronizace

Pro hledání transformačních funkcí CA byl použit modifikovaný algoritmus Ant colony system (s využitím konceptu popsaného na straně 9), jehož pseudokód je popsán v algoritmu 1. Na rozdíl od grafu použitého u TSP, hrany obsahují pouze jeden atribut a to feromon. Vzdálenost je v tomto případě irelevantní. V každém kroku postupu grafem se agent rozhoduje mezi dvěma uzly. To nám značně zjednodušuje v jistém ohledu rozhodovaní, kudy se má mravenec vydat. Opět je využita pravděpodobnostní funkce(2.4) z ACS, ale mírně poupravená. Modifikovaná verze pravděpodobnostní funkce pro tento problém je uvedena v rovnici (3.1).

$$p_k(r, s_x) = \frac{\tau(r, s_x)}{\tau(r, s_x) + \tau(r, s_y)}, \text{kde } x \in \{0, 1\}, y = \neg x \tag{3.1}$$

Kde $\tau(r, s_x)$ a $\tau(r, s_y)$ značí hodnotu feromonů na hranách ulzů, do kterých se může aktuálně mravenec přesunout. Tak jako u ACS, pouze mravenec s nejlepším výsledkem může zapsat přírůstek feromonů na cestu, kterou prošel. Po ukončení předem daného počtu iterací pro konkrétní transformační funkci nad konkrétním zadáním celulárního automatu je potřeba zjistit, který mravenec k se dostal k nejlepšímu výsledku. Jelikož hrany grafu již neobsahují vzdálenost, musel být vymyšlen úplně nový způsob, jak ohodnotit řešení jednotlivých mravenců. Za tímto účelem byla v této práci vyvinuta tzv. nekonečná norma, jejíž vyjádření je uvedeno v rovnici(3.2).

$$kvocient = \frac{max(\sum A, \sum B)}{n} \cdot \frac{\sum_{i=1}^{n} |a_i + b_i|}{n} \tag{3.2}$$

kvocient dává jako výsledek číslo v rozsahu 0 až 1, kdy čím blíže se toto reálné číslo přiblíží číslu jedna, tím je výsledek blíže synchronizaci. Na druhou stranu čím blíže k 0, tím horší řešení agent nalezl. Synchronizace nastává v bodě, kdy výsledek je roven jedné. A reprezentuje hodnoty buněk automatu v předposlední iteraci a B reprezentuje hodnoty

buněk automatu v poslední iteraci. n je počet buněk automatu. Proměnné a, b reprezentují hodnotu stavu buněk na indexu i. a v předposlední iteraci, b v poslední.

Je nutné také popsat, jakým způsobem se aktualizují hodnoty feromonů. Za tímto účelem je zde využita rovnice (2.6) pro globální aktualizaci feromonů z ACS. Ovšem u této rovnice se výpočet $\Delta\tau(r, s)$ realizoval pomocí vzdálenosti. Jelikož u problému synchronizace nám vzdálenost odpadá, bylo potřeba vymyslet zcela nový způsob, jak šetrně aktualizovat feromony tak, aby algoritmus fungoval co nejlépe. Použitý postup je popsán v rovnici (3.3).

$$\Delta\tau(r,s) = \begin{cases} \frac{kvocient}{Q} & \text{if } (r,s) \in \text{Kvocient s nejvyšší hodnotou} \\ 0, & jinak \end{cases} \tag{3.3}$$

Výsledek ohodnocování transformačních funkcí pomocí nekonečné normy(3.2) nám dává i dobrý přehled, jak blízko jsme se ke kýženému výsledku dostali. Proto je velice výhodné použít tuto informaci i pro výpočet aktualizací feromonových hodnot. Použití jenom samotného kvocientu ale nestačí, jelikož je to poměrně veliké číslo, mohlo by to mít za následek, že v počátku algoritmu budou preferované hrany vedoucí buďto ne k optimálnímu, ale možná i k úplně špatnému výsledku. Proto toto číslo je potřeba zmenšit dělením vhodnou konstantou Q. Díky tomuto kroku budou mravenci zpočátku algoritmu více provádět průzkum(Exploration), kde se postupem času propracují k řešením blížících se k výsledku. Modifikovaný postup ACS použitý pro řešení problému synchronizace v CA je uveden v popisu algoritmu (2), kde vstupní parametr c je počet iterací, r_0 je počáteční hodnota feromonů na všech hranách a m je počet mravenců.

Algoritmus 2: ACS pro problém synchronizace

Input: (c, r_0, m)

```
1: nastav hodnoty feromonů na všech hranách na hodnotu r_0
2: for i = 0 to c do
3:     umísti všechny mravence do startovacího bodu S
4:     for j = 0 to 32 do
5:         for k = 0 to m do
6:             State_transition_rule(k)
7:         end for
8:     end for
9:     for k = 0 to m do
10:        nastav stavy buněk v celulárním automatu na iniciační hodnoty
11:        for j = 0 to x (počet iterací aplikace transformační funkce) do
12:            Aplikuj transformační funkci nalezenou mravencem k na celulární
               automat
13:        end for
14:        vyhodnoť řešení aplikací rovnice 3.2
15:        if kvocient == 1 then
16:            return transformační funkci
17:        end if
18:    end for
19:    var t = Find_best_kvocient();
20:    Global_update_rule(t)
21: end for
```

Kapitola 4

Experimentální výsledky pro vybrané case studies

V této kapitole se zaměříme na testování algoritmu ACS a vyhodnocení statistických údajů. Algoritmus bude řešit problém obchodního cestujícího a problém synchronizace. Tato kapitola tedy bude rozdělena na dvě hlavní části. Důležitou součástí experimentů u TSP bude nejprve najít co nejlepší konfiguraci vstupních parametrů programu. Najít rovnováhu v nastavení algoritmu je velice důležité, protože daná konfigurace má velice významný vliv na přesnost výpočtů. Následně aplikujeme ACS na dva různé sety zadání. Aplikace ACS na problém synchronizace je výzkumnou částí této práce a budeme především zkoumat úspěšnost, rychlost konvergence k řešení a také celkové chování aplikace.

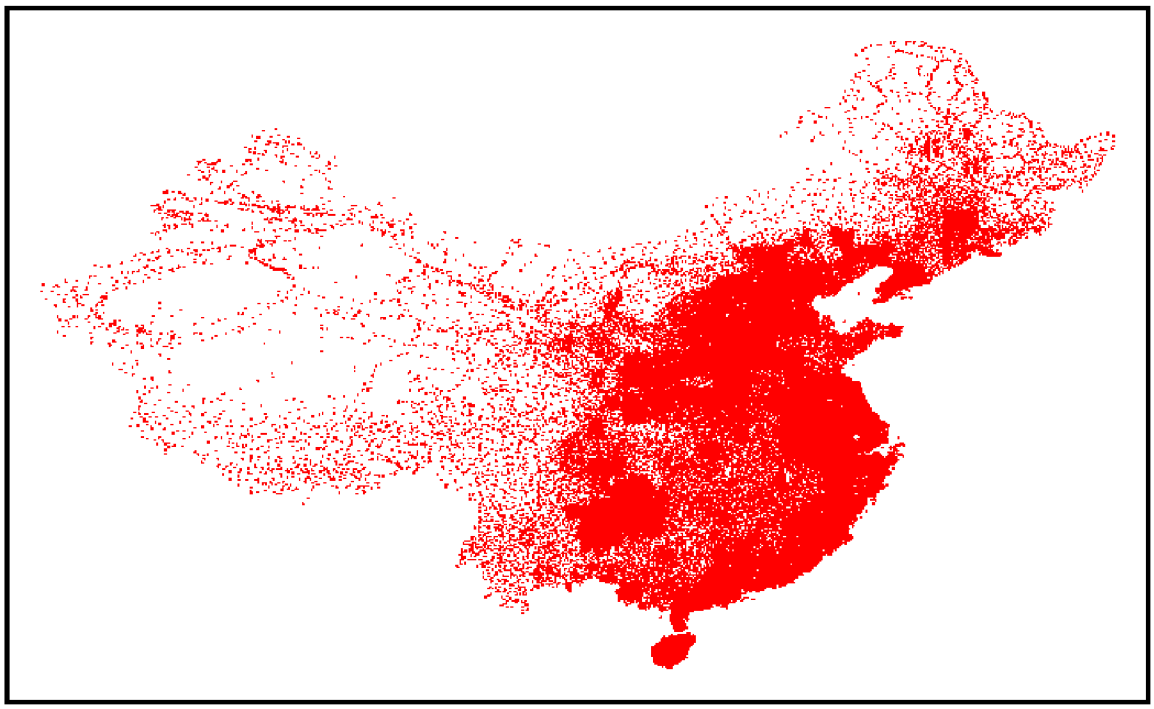
4.1 Experimentální výsledky pro TSP

U TSP bude algoritmus testován na benchmarky získané z knihovny TSPLIB. Co nás bude hlavně zajímat, je způsob chování a efektivita algoritmu vůči různému počtu měst. Tento počet bude v rozmezí 15-ti až 194 městy. Efektivnost nás bude zajímat hlavně proto, protože ACS je stochastický algoritmus.

4.1.1 Knihovna TSPLIB

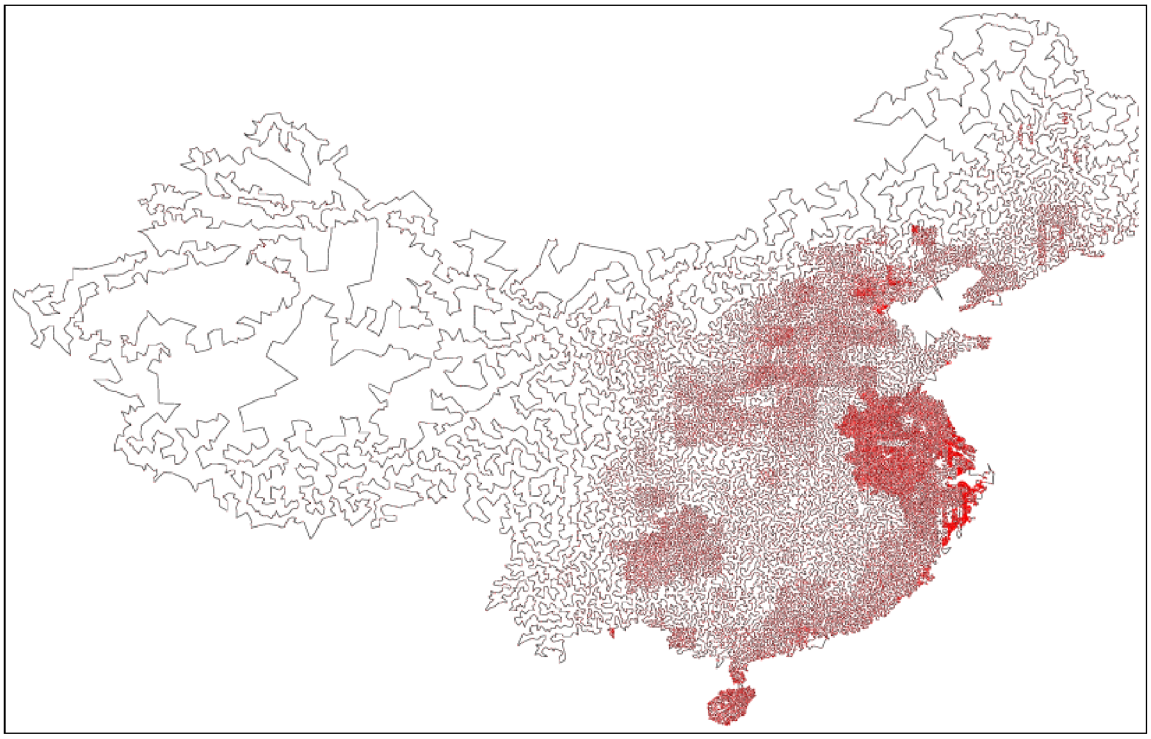
Pro testování algoritmu bylo potřeba najít kvalitní sety zadání se zveřejněnými výsledky. Tyto kritéria splnila knihovna TSPLIB. Co to tedy vůbec je? TSPLIB je knihovna vzorových zadání pro problém obchodního cestujícího, ale i jiných problémů z různých zdrojů a různých typů. Množina zadání, ze kterých tato práce čerpá, patří do třídy mezinárodních datasetů. To znamená, že města v konkrétním setu na naší planetě opravdu existují a každý dataset reprezentuje rozložení měst v nějakém státu. Knihovna obsahuje zadání v rozsahu 29-ti měst v západní Sahaře až po Čínu se 71 tisíci městy. Každý dataset obsahuje několik prvků.

První takový prvek je point set, kde jsou jednotlivá města zakreslena v Eukleidovském prostoru dle jejich souřadnic a tudíž jsou zde obrazně vykreslené vzdálenosti mezi městy.



Obrázek 4.1: Ukázka pointsetu měst v Číně.

Point set také obsahuje informace o optimální cestě uvedenou v kilometrech. Je potřeba vzít v potaz, že optimální cesta nemusí být úplně přesná a knihovna uvádí s jakou nepřesností můžeme počítat. U každého zadání se jedná ale pouze o velmi malá procenta, která se pohybují v řádech setin, či tisícin. Důvodem je, že výpočty, ze kterých výsledky vyšly, zaokrouhlovaly vzdálenosti mezi městy na celá čísla. V tomto algoritmu se ovšem tomuto vyhneme a algoritmus bude počítat s floating point aritmetikou.



Obrázek 4.2: Ukázka vykreslené optimální cesty v čínském datasetu, kdy nepřesnost je odhadována pouze na hodnotu 0,028%. Prozatím nejlepší nalezená optimální cesta v tomto případě má vzdálenost 4,566,563 kilometrů.

Dataset také obsahuje mapu dané země získanou z databáze map (World FactBook Map) organizace CIA. Dále samozřejmě surové data souřadnic měst, které jsou uložena ve speciálním formátu.

```
NAME: dj38
COMMENT : 38 locations in Djibouti
COMMENT : Derived from National Imagery and Mapping Agency data
COMMENT : This file is a corrected version of dj89, where duplications
COMMENT:  have been removed.  Thanks to Jay Muthuswamy and others for
COMMENT:  requesting data sets without duplications.
TYPE: TSP
DIMENSION: 38
EDGE_WEIGHT_TYPE: EUC_2D
NODE_COORD_SECTION
1 11003.611100 42102.500000
2 11108.611100 42373.888900
3 11133.333300 42885.833300
4 11155.833300 42712.500000
5 11183.333300 42933.333300
```

Obrázek 4.3: Ukázka části datového souboru, kde označení DIMENSION značí počet měst, EDGE_WEIGHT_TYPE způsob výpočtu vzdáleností a v sekci NODE_COORD_SECTION každý řádek zastupuje jedno město, kdy první číslo je označení města, druhé číslo x souřadnice a třetí číslo y souřadnice.

Výpočet vzdálenosti mezi městy závisí na typu zadání. U TSP se objevuje pouze jediný typ dat a to EUC_2D. Je to jednoduchý Eukleidovský prostor, není tedy třeba při výpočtu používat například zaoblení země. Implementace výpočtu je zobrazeno na obr. 4.4

```
xd = x[i] - x[j];
yd = y[i] - y[j];
dij = nint( sqrt( xd*xd + yd*yd) );
```

Obrázek 4.4: Kód v jazyce C pro výpočet vzdáleností pro EUC_ 2D typ dat pomocí jednoduché Pythagorovy věty.

Poslední sekce, kterou dataset nabízí je log. soubor výpočtů pro dané zadání. Obsahuje například informace o výpočetním čase, na jakém procesoru výpočet běžel, aplikovaný algoritmus či nastavení daného algoritmu. Zajímavostí je, že u opravdu složitých problému, jako je zadání Číny, není optimální cesta zatím známá. Pořád stále probíhají výpočty na různých strojích s různými algoritmy a historie nalezení nového optimálního řešení je rovněž přítomna na stránkách TSPLIB [1].

4.1.2 Vliv nastavení na ACS algortimus

U ACS algoritmu máme mnoho možností jak změnit charakter chování celkového algoritmu. Na jeho běh má vliv velké množství konstant a proměnných, jejichž hodnoty nám dávají představu například o tom, jakým způsobem se budou mravenci rozhodovat, jestli spíše budou preferovat hledání nových nebo procházení již kvalitně ohodnocených cest. Dále určitě způsob ohodnocení výsledků, tedy jak budou aktualizovány hodnoty feromonů na hranách grafu. Toto nastavení je velice důležité pro hledání co nejlepších výsledků. V neposlední řadě můžeme definovat, jak moc je pro nás důležitý poměr hodnot feromonů a vzdáleností na hranách při aplikaci pravidla přechodů. Správná kombinace těchto prvků nám dopomůže k tomu, aby tento algoritmus hledal řešení blížící se co nejvíce tomu optimálnímu. O jaké prvky se tedy jedná?

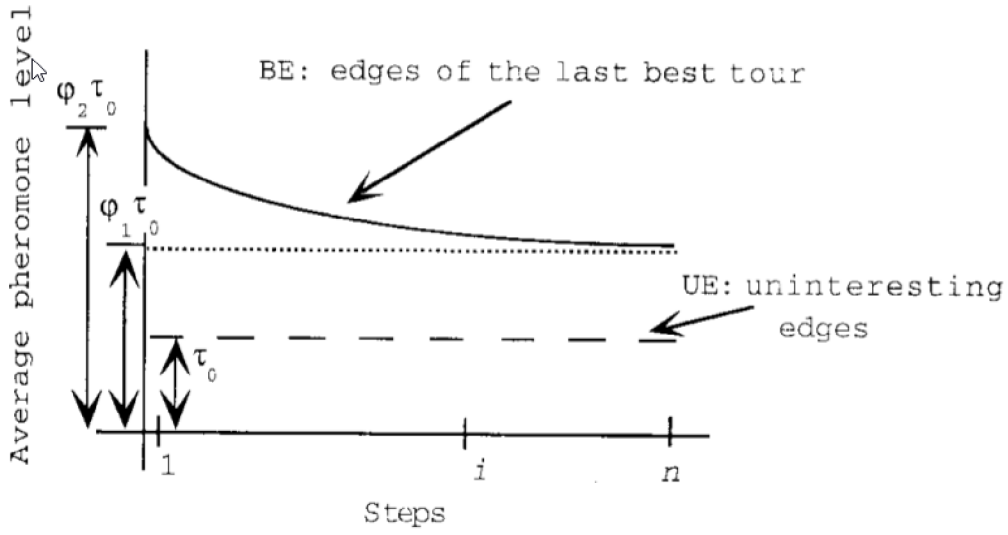
Jako první je potřeba zvážit, kolik iterací by měl samotný algoritmus provést, tedy nastavení konstanty c. Pokud by jsme toto číslo nastavili příliš malé, mohlo by dojít k tomu, že program nestihne konvergovat k přesnosti, kterou vyžadujeme. V opačném případě, pokud hodnotu zvolíme příliš vysokou, algoritmus zbytečně spotřebuje mnoho výpočetního času. Abychom dostali dobrý přehled o tom, jak algoritmus funguje při daném nastavení parametrů, bude konstanta c nastavena podle literatury na hodnotu 1000. Víme, že výpočetní náročnost algoritmu roste exponenciální řadou úměrně s počtem měst. Ovšem při 15-ti městech nebude výpočet nějak náročný, tudíž si toto nastavení můžeme dovolit. Při testování složitějších zadání bude potřeba tuto hodnotu upravit a postačíme si například s hodnotou 500 či 200.

Dále si rozebereme nastavení heuristických konstant α a β. Tyto konstanty mají velký vliv na rozhodování mravenců během hledání částečného řešení v dané iteraci. Konkrétně α udává, jakou váhu budou mít feromony při aplikaci Pravidla přechodů. Na druhé straně to samé samozřejmě platí u konstanty β, jenž je ve výpočtu mocnitel vzdáleností. O této konstantě můžeme obrazně říci, že je protiváhou feromonů. Obecně se v literatuře u Ant system algoritmu udává, že hodnota α i β mají nabývat hodnot $< 1,5 >\in \mathbb{N}$. Nastavení

[1] http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

hodnoty $\alpha > 1$ zní jako logický krok, protože chceme, aby mravenci navštěvovali i hrany, které oplývají větší hodnotou feromonové stopy. Samozřejmě z počátku algoritmu by tato konstanta neměla sebemenší vliv, protože hodnoty feromonů jsou na všech hranách velice podobné, ne-li stejné. Postupem času by ale toto nastavení mělo za následek velikou převahu některých hran, které by nemusely vést ke kvalitnímu řešení. Proto se u ACS upustilo od preferování feromonů a α bude vždy nastavena na hodnotu $\alpha = 1$. Na druhou stranu β už je v tomto ohledu zajímavější. Umocňuje hodnoty, které jsou po celou dobu algoritmu neměnné a je žádané, aby mravenci vybírali krátké cesty. V této práci tedy budeme testovat chování algoritmu při hodnotách $\beta = 1, 2, 3a4$.

Další proměnná, kterou je potřeba dobře nastavit, je počet mravenců řešící dané zadání. Pro tento výpočet je důležitý obr. 4.5 a následující úvaha. Nechť $\varphi_2 tau_0$ je průměrná hodnota feromonů na hranách z BE (*best edges*) množiny hned po aplikaci globálního aktualizačního pravidla(2.6) a $\varphi_1 tau_0$ je průměrná hodnota feromonů na hranách z BE množiny před aplikaci globálního aktualizačního pravidla. ($\varphi_1 tau_0$ je taktéž úrovní feromonové hodnoty na hranách TE (*test edges*) množiny na začátku cyklu algoritmu).



Obrázek 4.5: Změna průměrné hladiny feromonů na hranách, které jsou součástí BE množiny. Průměrná hodnota feromonů na BE hranách začíná na hodnotě $\varphi_2 tau_0$ a jejich hodnota se zmenšuje pokaždé, když je hrana navštívena. Na konci algoritmu každá hrana byla navštívena průměrně $q_0 \cdot m$ a finální hodnota feromonů je $\varphi_1 tau_0$. [5]

Předpokládejme, že hodnoty $\varphi_2 tau_0$ a $\varphi_1 tau_0$ známe, pak se vhodný počet mravenců počítá následovně. Lokální aktualizační pravidlo má vztah lineární recidivy prvního řádu k této rovnici:

$$T_z = T_{z-1}(1 - \rho) + \tau_0 \rho \tag{4.1}$$

Která má uzavřenou formu popsanou v následující rovnici:

$$T_z = T_0(1 - \rho)^z - \tau_0(1 - \rho)^z + \tau_0 \tag{4.2}$$

Tedy, když víme, že těsně před aplikací Globálního aktualizačního pravidla platí $T_0 = \varphi_2 tau_0$ a následně co všichni mravenci vykonstruují své řešení v dané iteraci platí $T_z = \varphi_1 tau_0$, poté můžeme odvodit následující rovnici:

$$\varphi_1 = \varphi_1(1-\rho)^z - \tau_0(1-\rho)^z + 1 \tag{4.3}$$

Když si uvědomíme, že nejlepší hrany se vybírají s pravděpodobností q_0, pak můžeme zkusit aproximovat počet mravenců, kteří navštíví nejlepší hrany z pomocí vztahu $z = m \cdot q_0$. Posléze dosazením do formule (4.3) získáme rovnici pro výpočet optimálního počtu mravenců.

$$m = \frac{\log(\varphi_1 - 1) - \log(\varphi_2 - 1)}{q_0 \cdot \log(1-\rho)} \tag{4.4}$$

Bohužel se doposud nepodařilo zjistit předpis funkcí φ_2 a φ_1. Ovšem experimenty ukázaly, že ACS pracují kvalitně při poměru $(\varphi_1 - 1)/(\varphi_2 - 1) \approx 0,4$ tedy $\frac{m}{n}$, kde m je počet mravenců a n počet měst je rovno zlomku přibližně $\frac{2}{5}$ [5].

Další konstanta, která bude řešena je počáteční hodnota feromonů r_0. Podle literatury [5] je ideální použít pro výpočet této hodnoty heuristiku nejbližšího souseda. Tato heuristika je značně jednoduchá a funguje následovně. Umístíme jednoho agenta nějakým způsobem do jednoho města (např. náhodně). Daný agent se přesouvá do dalších měst na základě vzdáleností k ostatním městům. Vždy vybírá město, které ještě nenavštívil, a které je mu nejblíže. Po navštívení všech měst, tedy poté co dokončí řešení, spočítáme celkovou vzdálenost L, kterou mravenec ušel a vynásobíme ji počtem měst m. Výsledná rovnice pro výpočet vypadá následovně: $r_0 = 1/L \cdot m$. Pro nejsložitější zadání, které v této práci budeme testovat a které má 194 měst, se ale této heuristice vyhneme a parametr r_0 nastavíme na hodnotu 0.1 z toho důvodu, protože heuristika by u takového počtu měst našla až příliš malé číslo, které by mohlo být při výpočtech nežádoucí.

Jednou z nejdůležitějších konstant pro správné fungování algoritmu je nastavení hodnoty q_0, která v podstatě definuje chování mravenců. Tato konstanta nám říká, jestli mají mravenci preferovat chamtivou heuristiku (exploitation) nebo prohledávat jiné cesty, tedy jiné možnosti řešení (exploration). Jelikož se jedná o pravděpodobnostní konstantu, může q_0 nabývat hodnot $< 0,1 > \in \mathbb{R}$. Nastavení je velice citlivá záležitost a testování hodnot

bude předmětem zkoumání této práce. Implementovaný princip rozhodování v této práci je možné vidět v algoritmu 3

Algoritmus 3: ACS způsob rozhodování agentů

Input: (q_0)

```
1: x = Generate_random_real_number(0,1)
2: if x < q0 then
3:     return město s nejvyšší pravděpodobností
4: else
5:     y = Generate_random_real_number(0,1)
6:     total = 0
7:     for pro každé město i které může mravenec navštívit do
8:         if y >= total and y <= pravděpodobnost(i) + total then
9:             return město i
10:        else
11:            total + = Prob(i)
12:        end if
13:    end for
14: end if
```

Poslední zkoumaná konstanta zastupuje v algoritmu vypařování feromonů. V ACS algoritmu rozlišujeme dva typy vypařovací konstanty. První ω figuruje při výpočtu globálního aktualizačního pravidla a druhá konstanta a u lokálního aktualizačního pravidla. Obě tyto konstanty mají veliký vliv na vytváření průběžných mezivýsledků a i tedy na celkové řešení. Hodnoty obou konstant jsou nastaveny dle doporučení v literatuře na hodnotu $\omega = a = 0.1$ [5].

4.1.3 Hledání nastavení algoritmu

V sekci 4.1.2 jsme si popsali vliv parametrů na chování algoritmu. V této sekci se zaměříme na hledání kvalitního nastavení těchto parametrů pro ACS algoritmus. Konkrétně tedy budeme zkoumat vliv konstant q_0 a β. Program bude spuštěn s různými kombinacemi hodnot hledaných argumentů paralelně s pevně nastavenými ostatními argumenty. Vybraná úloha pro testování obsahuje 15 měst a to z důvodu velkého počtu kombinací a tím pádem výpočetní náročností. Každé různé nastavení bude spuštěno přesně stokrát, přičemž tolik iterací by mělo být dostačující pro zajištění kvalitních statistických údajů. Sledovány budou tyto informace. Nejlepší nalezená cesta, průměrná nalezená cesta, nejhorší nalezená cesta a medián délky nalezených cest. Poté se zaměříme přímo na chování algoritmu analýzou chováním mravenců při vytváření řešení. Jak už bylo řečeno, pevné parametry programu jsou nastaveny následovně: $\alpha = 1, m = 7, \omega = a = 0.1$. Zkoumat se budou hodnoty $\beta = 1, 2, 3, 4$ a $q_0 = 0.1, 0.3, 0.5, 0.7, 0.9$.

β	1			
$q_0 = 0.1$	Nejlepší 2601.2638	Průměr 2668.4311	Nejhorší 2788.3177	Medián 2666.9388
$q_0 = 0.3$	Nejlepší 2601.2638	Průměr 2602.0178	Nejhorší 2606.1524	Medián 2601.2638
$q_0 = 0.5$	Nejlepší 2601.2638	Průměr 2601.2638	Nejhorší 2601.2638	Medián 2601.2638
$q_0 = 0.7$	Nejlepší 2601.2638	Průměr 2601.3038	Nejhorší 2603.2449	Medián 2601.2638
$q_0 = 0.9$	Nejlepší 2601.2638	Průměr 2656.1147	Nejhorší 3071.2550	Medián 2620.0329

Tabulka 4.1: Tabulka statistických údajů pro kombinace hodnot q_0 a hodnoty $\beta = 1$.

V tabulce 4.1 je možno si všimnout, že všechny kombinace nastavení nalezly optimální cestu, to ale ostatně kvůli jednoduchosti zadání také všechny ostatní. Je potřeba si uvědomit, že pokud hodnota β nabývá hodnoty jedna, dáváme vzdálenostem na hranách stejnou váhu jako feromonům. Na rozdíl od ostatních tabulek je zde poznat určitý rozdíl v nalezení nejhorších cest a u mediánu. Zatímco u ostatních výsledků se hodnota nejhorší nalezené vzdálenosti většinou úměrně zvyšuje s výší hodnoty q_0, zde je nalezení špatných cest vázáno spíše k extrémům hodnot q_0. Taktéž můžeme stejnou hypotézu konstatovat u mediánu. Ovšem tato skutečnost nepatří mezi žádané chování algoritmu a jeho chování se spíše jeví jako nahodilé či chaotické, což by u složitějších zadání mohlo vyústit k horším výsledkům.

β	2			
$q_0 = 0.1$	Nejlepší 2601.2638	Průměr 2601.2638	Nejhorší 2601.2638	Medián 2601.2638
$q_0 = 0.3$	Nejlepší 2601.2638	Průměr 2601.2638	Nejhorší 2601.2638	Medián 2601.2638
$q_0 = 0.5$	Nejlepší 2601.2638	Průměr 2603.2312	Nejhorší 2791.1524	Medián 2601.2638
$q_0 = 0.7$	Nejlepší 2601.2638	Průměr 2605.6493	Nejhorší 2722.6427	Medián 2601.2638
$q_0 = 0.9$	Nejlepší 2601.2638	Průměr 2638.1670	Nejhorší 2995.7047	Medián 2603.2449

Tabulka 4.2: Tabulka statistických údajů pro kombinace hodnot q_0 a hodnoty $\beta = 2$.

Na rozdíl od tabulky 4.1 můžeme v tabulce 4.2 pozorovat očekávané chování algoritmu velice zřetelně. Se zvyšující se hodnotou q_0 algoritmus nalezl v několika iteracích poměrně špatné řešení. Toto je ale přirozený výsledek při vysoké hodnotě konstanty q_0. Spolu s těmito hodnotami je samozřejmě normální, že průměrná hodnota taktéž vzroste. Co je ale důležité, že se jedná pouze o několik iterací s horším výsledkem. Medián v tomto případě ukazuje buď na optimální hodnotu nebo na hodnotu jí velice blízkou.

β	3			
$q_0 = 0.1$	Nejlepší 2601.2638	Průměr 2601.2638	Nejhorší 2601.2638	Medián 2601.2638
$q_0 = 0.3$	Nejlepší 2601.2638	Průměr 2601.3132	Nejhorší 2606.1524	Medián 2601.2638
$q_0 = 0.5$	Nejlepší 2601.2638	Průměr 2602.2729	Nejhorší 2669.8463	Medián 2601.2638
$q_0 = 0.7$	Nejlepší 2601.2638	Průměr 2610.5674	Nejhorší 2831.8575	Medián 2601.2638
$q_0 = 0.9$	Nejlepší 2601.2638	Průměr 2633.1249	Nejhorší 2973.8061	Medián 2601.2638

Tabulka 4.3: Tabulka statistických údajů pro kombinace hodnot q_0 a hodnoty $\beta = 3$.

V tabulce 4.3 figurují podobné výsledky jako v tabulce 4.2 a taktéž vykazují prvky očekávaného chování algoritmu. Oproti tabulce 4.2 tyto výsledky dokonce lépe reflektují očekávaný průměr a nejhorší nalezené výsledky vzhledem k hodnotě q_0. Podle mého názoru jsou rozdíly výsledků mezi hodnotami $\beta = 2$ a $\beta = 3$ minimální a jedná se nejspíše o důsledek stochastičnosti algoritmu.

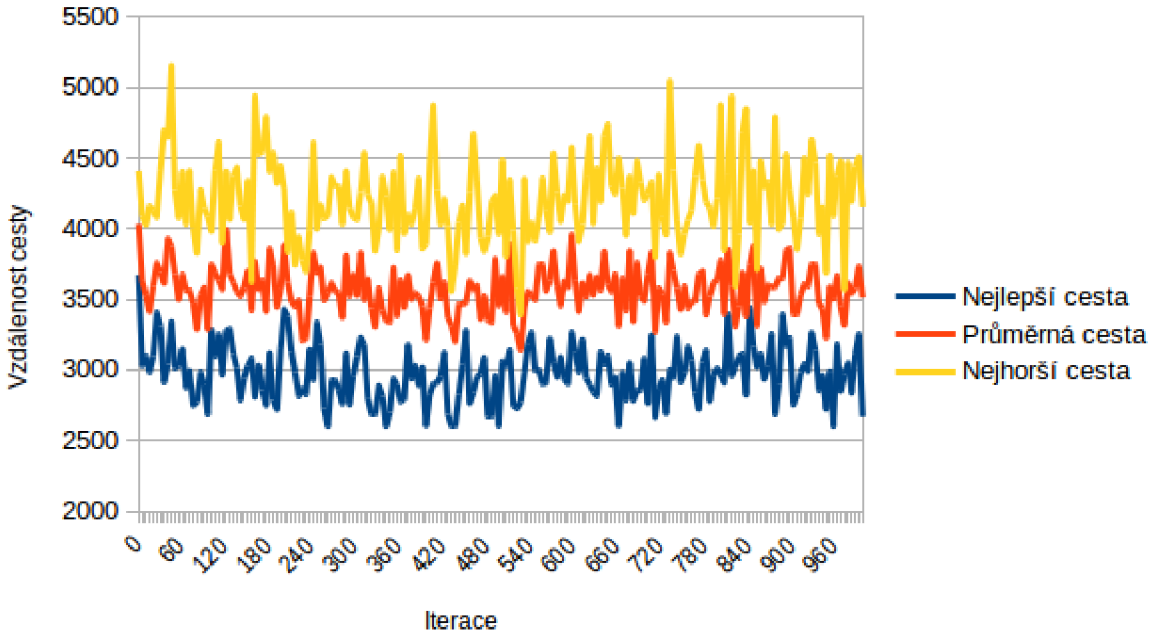
β	4			
$q_0 = 0.1$	Nejlepší 2601.2638	Průměr 2601.4119	Nejhorší 2606.1524	Medián 2601.2638
$q_0 = 0.3$	Nejlepší 2601.2638	Průměr 2602.3035	Nejhorší 2669.8463	Medián 2601.2638
$q_0 = 0.5$	Nejlepší 2601.2638	Průměr 2610.5501	Nejhorší 2863.6683	Medián 2601.2638
$q_0 = 0.7$	Nejlepší 2601.2638	Průměr 2618.4053	Nejhorší 2806.9611	Medián 2601.2638
$q_0 = 0.9$	Nejlepší 2601.2638	Průměr 2618.6865	Nejhorší 2734.3181	Medián 2601.2638

Tabulka 4.4: Tabulka statistických údajů pro kombinace hodnot q_0 a hodnoty $\beta = 4$.

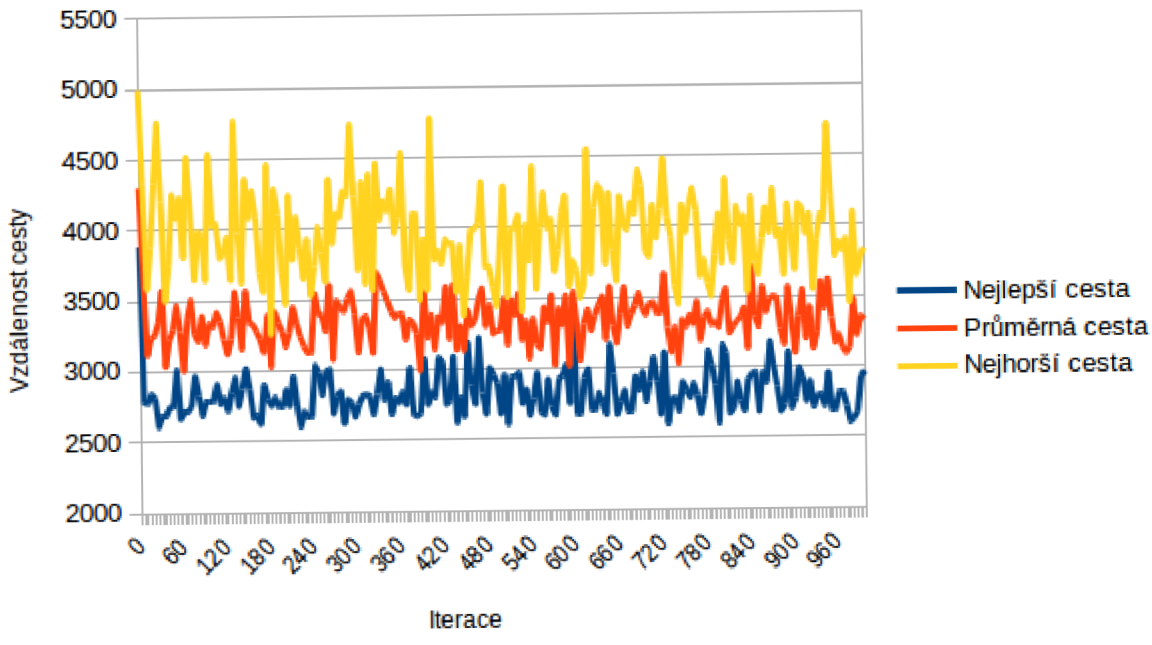
I když se jedná už o poměrně extrémní hodnotu, výsledky pro konstantu $\beta = 4$ v tabulce 4.4 nevykazují takové náhodné chování jako v tabulce 4.1. Můžeme si povšimnout, že medián i průměrná hodnota vykazují poměrně přijatelná data. To už ale neplatí pro nalezení nejhorších výsledků.

Dá se tedy říci, že extrémy hodnot konstanty β 1 a 4, mohou mít na chování algoritmu nežádoucí účinky. U hodnoty $\beta = 1$ je poměrně pravděpodobné, že mravenci budou tíhnou k náhodným řešením, které nemusí být kvalitní. Feromonová stopa bude mít ve výpočtu velikou váhu a tím pádem může převýšit důležitost vzdáleností. U opačného extrému pak dáváme až přílišnou výhodu vzdálenosti, kdy na feromonech nebude téměř vůbec záviset. Tím aktualizační pravidla přestanou mít nejspíše žádanou účinnost a u obsáhlejších problémů by výsledky nemusely být vůbec kvalitní. Výsledky pro $\beta = 2$ a $\beta = 3$, se ukázaly jako poměrně lepší varianty.

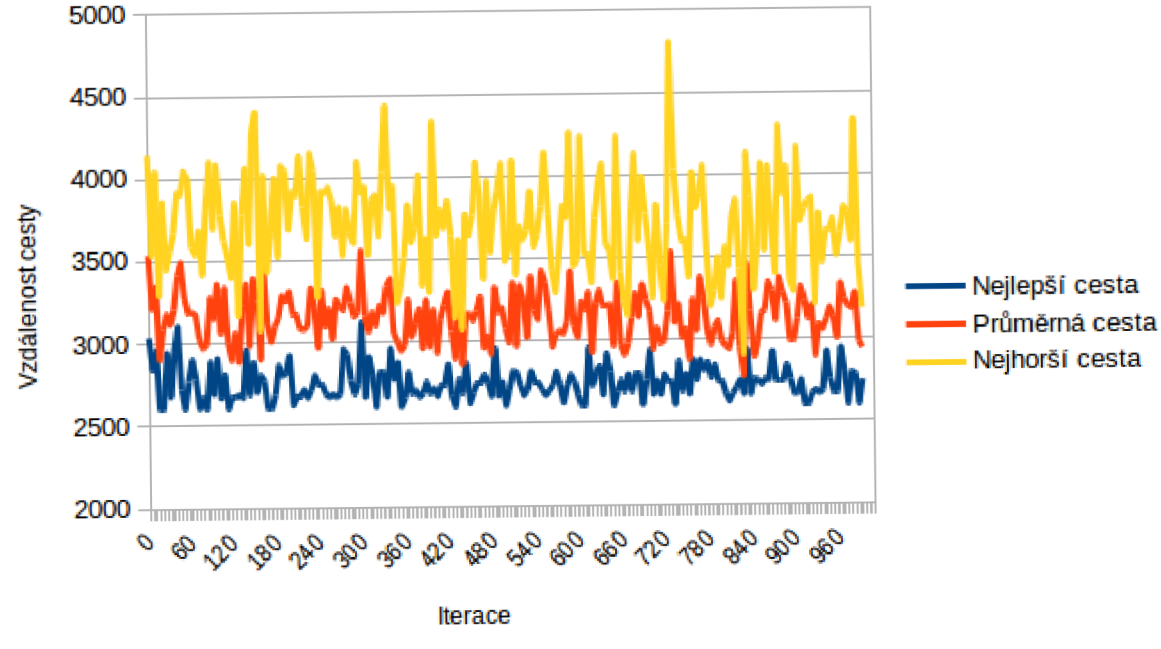
U konstanty q_0 je to velice podobné, ovšem výsledky v předchozích tabulkách nám moc neukázaly, jaký vliv má tato hodnota na výsledky algoritmu, proto je potřeba posoudit vliv této konstanty na samotném chování agentů při vytváření výsledků. Následné grafy ukazují příklad vývoje jednoho spuštění algoritmu s postupem iterací, kde zobrazují nejlepší, průměrné a nejhorší nalezené řešení mravenců v jednotlivých iteracích pro různé kombinace parametrů β a q_0.



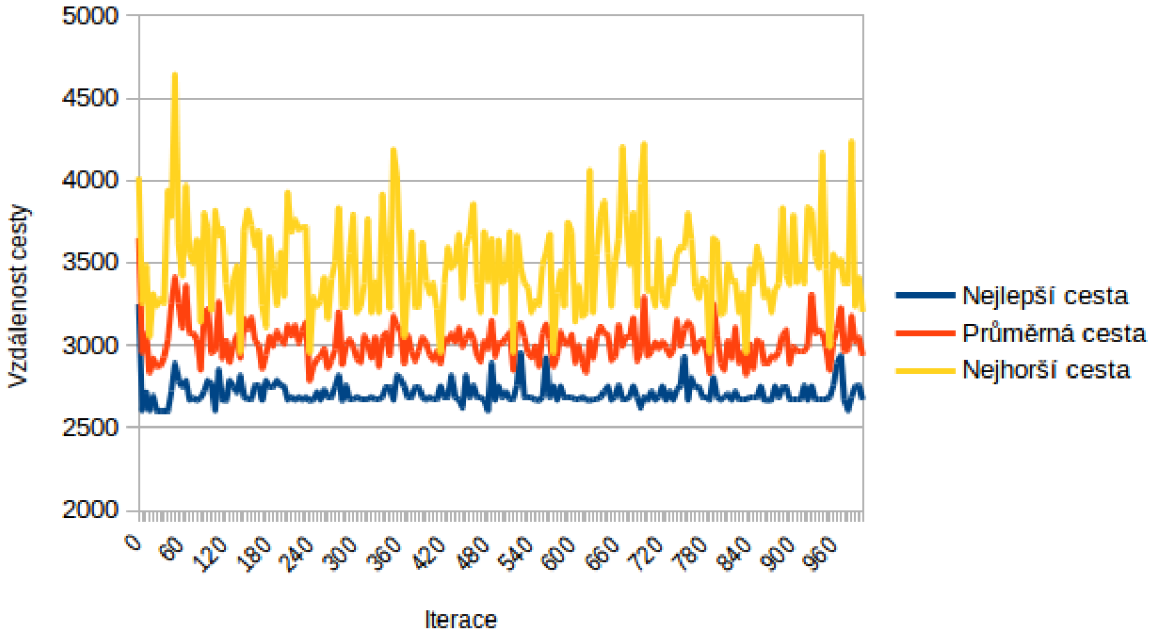
Obrázek 4.6: $\beta = 1, q_0 = 0.1$



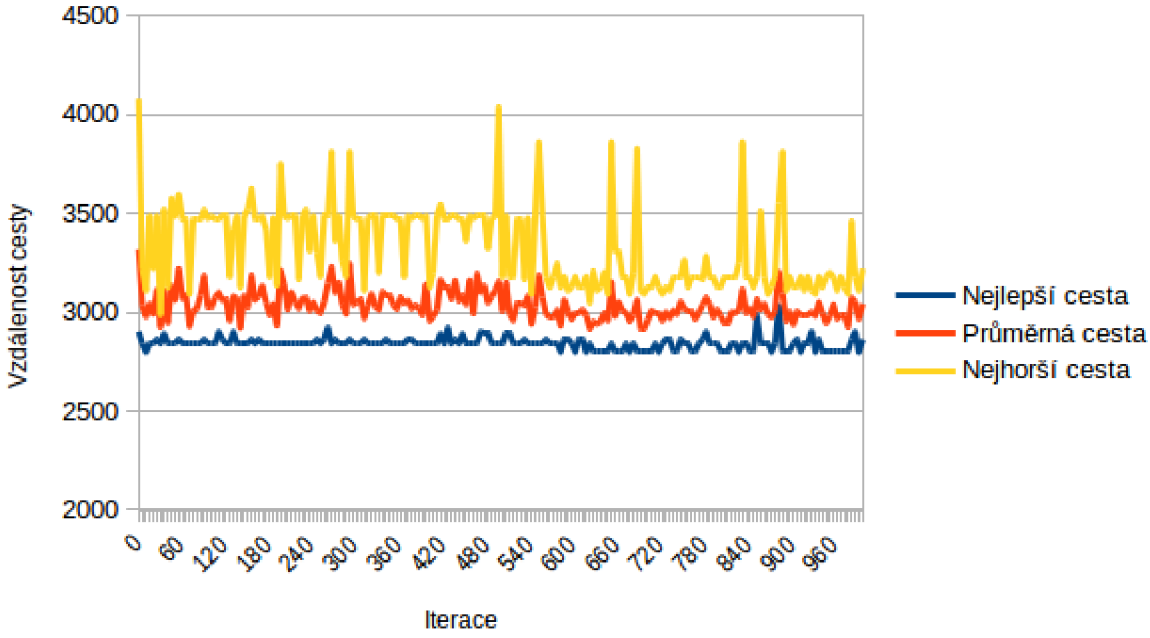
Obrázek 4.7: $\beta = 1, q_0 = 0.3$



Obrázek 4.8: $\beta = 1, q_0 = 0.5$



Obrázek 4.9: $\beta = 1, q_0 = 0.7$



Obrázek 4.10: $\beta = 1, q_0 = 0.9$

V grafech na obrázcích 4.6 - 4.10 jsou ukázky chování mravenců pro různé hodnoty q_0 a $\beta = 2$. Z grafů je dobře vidět, že čím větší hodnota q_0 je, tím více má algoritmus tendenci usadit se na nějaké hladině nejlepších řešení. Takové chování je ale předpokládané. Opět si lze povšimnout, že obě extrémní hodnoty $Q_0 = 0.1, q_0 = 0.9$ nepřinesly poměrně

kvalitní výsledky. Při první zmíněné variantě algoritmus spíše nalézal horší řešení a míra náhodnosti v rozhodování při výběru dalšího města je až příliš veliká. Tuto nahodilost lze vysledovat z rozptylu hodnot nejlepší nalezené cesty v jednotlivých iteracích, kdy křivka grafu vykresluje veliké výkyvy délky nejlepších cest. V druhém případě je problém opačný. Program velice rychle konvergoval k nějaké hladině řešení, které v tomto případě je poměrně přesné, ale při složitých úkolech toto chování může být spíše ke škodě, kdy u většiny případů můžou výsledky rychle konvergovat k horším nebo úplně špatným řešením. Grafy pro ostatní hodnoty *beta* jsou uvedeny v příloze na straně 47. U ostatních grafů A.1 - A.15 pro jiné hodnoty než $beta = 2$ si lze povšimnout, že i vyšší hodnota *beta* má jistý vliv na rychlost konvergence k nálezu nějakých nejlepších nalezených hodnot. Konečné hodnoty, vybrané na základě těchto výsledků, které se budou dále v této práci aplikovat jsou $beta = 2, q_0 = 0.7$.

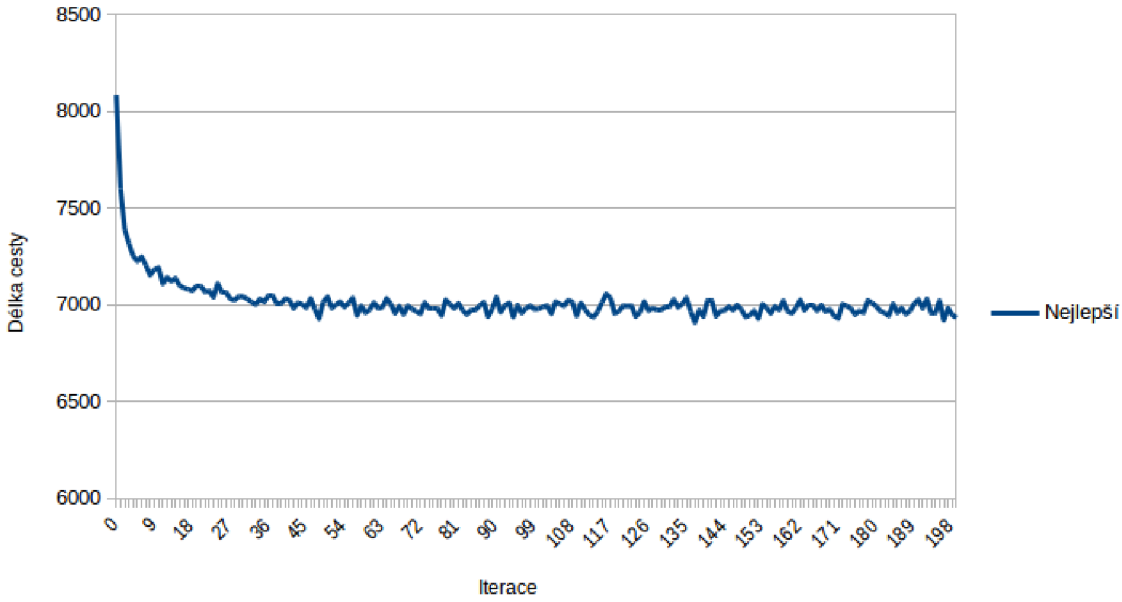
4.1.4 Výsledky pro zadání dj38

Jméno dj38 označuje v knihovně zadání s 38-mi městy ve státě zvaném Djibouty. Jak práce uvádí v sekci na straně 9, Ant system byl navržen spíše pro hledání optimálního řešení na zadáních do 30-ti měst. V této sekci otestujeme ACS algoritmus na setu s o něco větším počtem měst. Parametry programu byly nastaveny na základě informací v sekcích na stranách 24 a 21. Předmětem zkoumání bude opět hlavně přesnost algoritmu a dále také jeho chování. U zadání typu dj38 knihovna TSPLIB uvádí, že optimální řešení má 6656 km. Je potřeba ale vzít v potaz, že výsledek byl počítán pouze se zaokrouhlenými hodnotami na celá čísla. V této práci implementovaný algoritmus pracoval s floating point aritmetikou, a proto budeme počítat také odchylku od původního řešení.

dj38	Nejlepší nalezená cesta	Nejhorší výsledek	Průměr	Medián
Délka cesty v km	6659,4315	6831.3748	6696.0567	6674.8595

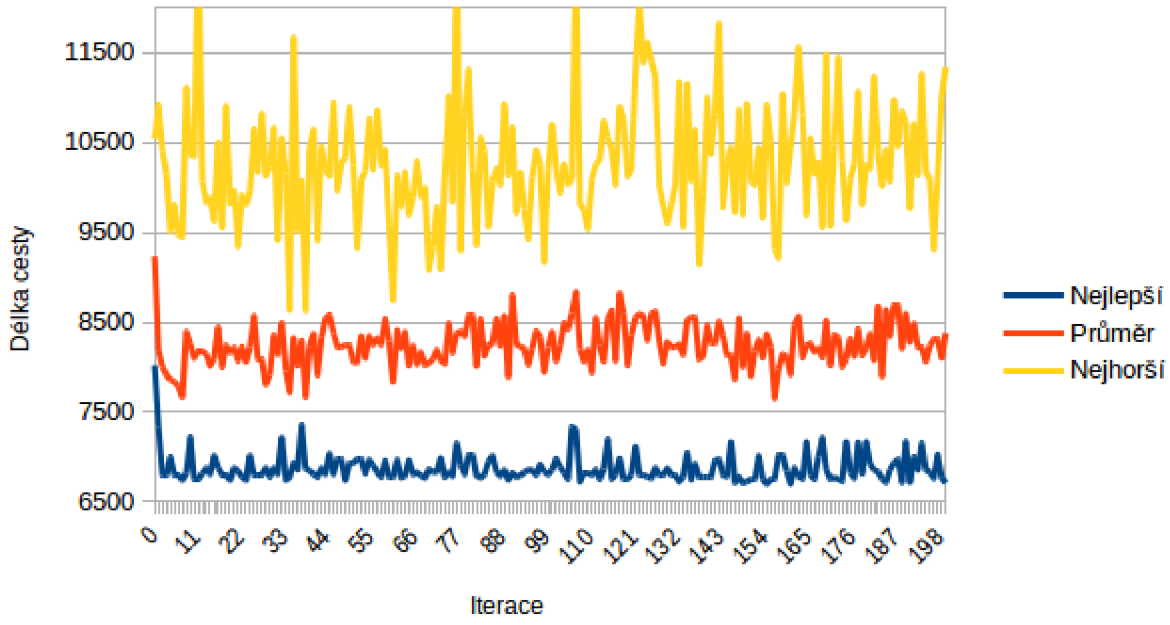
Tabulka 4.5: Tabulka obsahující statistické výsledky nalezených řešení programu.

V tabulce 4.5 je vidět, že nejlepší nalezený výsledek je velice kvalitní. Odchylka je v tomto případě rovna pouze 0.052%. Celkově tedy algoritmus zaznamenal malé zhoršení výsledků oproti sekci 4.1.3, kde hlavně medián a průměr se při kvalitním nastavení blížil mnohem více nalezené optimální hodnotě než je tomu zde. Ovšem toto chování bylo předpokládané z důvodu výpočetně těžšího zadání. Důležité je také vyzdvihnout rozdíl mezi nejlepším a nejhorším nalezeným výsledkem, kdy tento rozdíl je spíše zanedbatelný a každé spuštění programu našlo poměrně kvalitní řešení, kdy odchylka nejhoršího nalezeného řešení je rovna číslu 2,567%.

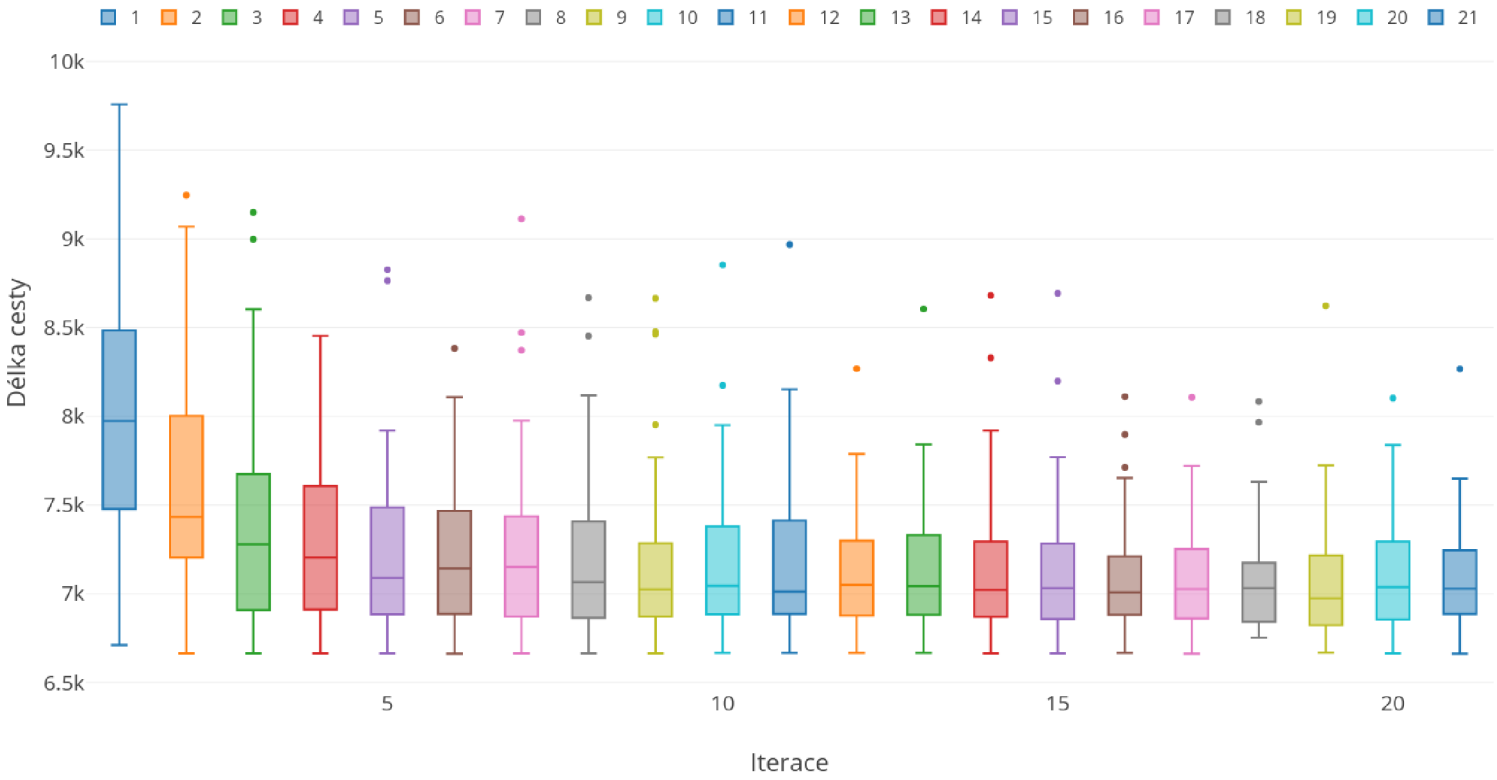


Obrázek 4.11: Graf znázorňující průměrné nejlepší nalezené hodnoty délky cest v každé iteraci všech spuštění programu

Výsledky v grafu 4.11 nám mají ukázat vývoj výsledků algoritmu, tedy jinými slovy, křivka v grafu vykresluje průměr nejlepších nalezených řešení v dané iteraci v rámci všech spuštění programu. Tento graf má demonstrovat chování programu s postupem výpočtů v iteracích. Algoritmus tedy očekávaně při prvních pokusech nalézal spíše horší hodnoty, kdy postupně nachází lepší a lepší výsledky a poté se usadí na nějaké hladině. V prvních iteracích algoritmu je znát, že čím vyšší číslo iterace, tím menší rozdíly v délkách nalezených cest. Po nějaké době se křivka už ustálí a s malými výkyvy se mihotá kolem nějaké hodnoty. Je ovšem mít na paměti, že v grafu 4.11 je uveden průměr všech měření. V každém samotném spuštění se může program chovat více či méně odlišným způsobem, jenž je znázorněný v grafu 4.12 a je to zapříčiněno mírou náhodnosti v algoritmu. Zhodnocení konvergence algoritmu je možné vidět také na obrázku 4.13, který ukazuje statistické údaje prvních 21 iterací ACS.



Obrázek 4.12: Graf znázorňující vývoj jednoho spuštění programu



Obrázek 4.13: Boxplot znázorňující výsledky nalezených nejlepších cest v iteracích 1-21 tedy úplného začátku algoritmu

Obrázek 4.13 doplňuje statistické údaje z grafu 4.11, kde střední část krabicového grafu (obdélník mezi kvartily) se se stoupající hodnotou iterace přibližně zmenšuje a posouvá níže. Tato část krabicového grafu nám reprezentuje většinu nalezených hodnot v dané iteraci a tudíž čím níže se boxpolot nachází, tím lepší výsledky algoritmus v dané iteraci nacházel. Důležitým prvkem je zde i hladina mediánu, kdy tato hodnota nám rozděluje nalezené výsledky na polovinu. Ve vyšších iteracích se hladina mediánu nachází blíže k prvnímu kvartilu, tedy veliká část nalezených cest se většinou přibližovaly nalezenému minimu v dané iteraci.

4.1.5 Výsledky pro zadání qa194

Poslední zadání, na kterém se bude testovat ACS algoritmus je množina měst ve státu Quatar, kde tento set obsahuje 194 měst. Jak už bylo dříve řečeno, výpočetní náročnost a složitost zadání roste exponenciální řadou se zvětšujícím se počtem měst. Pro představu si uvedeme strojový čas potřebný pro výpočet u jednotlivých zadání. Pro výpočet TSP s 15-ti městy potřeboval počítač s parametry popsanými v tabulce 4.6 přibližně pár sekund. U Djibouty setu s 38-mi městy se tento čas pohyboval v rozmezí již 30-ti minut. Jelikož první zadání mělo kalibrační účel a druhé ještě nepředstavovalo velikou výpočetní zátěž, byl algoritmus implementován v pythonu kvůli výraznému zjednodušení implementace a manipulace s algoritmem. Ovšem pro výpočetní potřeby tak velikého zadání jako je qa194

byl algoritmus přepsán do jazyka C++. Zadání bylo testováno na superpočítači Anselm v Ostravě, který podporuje paralelní spuštění procesů. Pro vyhodnocení řešení této instance problému bylo použití superpočítače prakticky nezbytné, protože jeden optimalizační běh ACO trval přibližně 17 hodin. Výsledky lze vidět v tabulce 4.7.

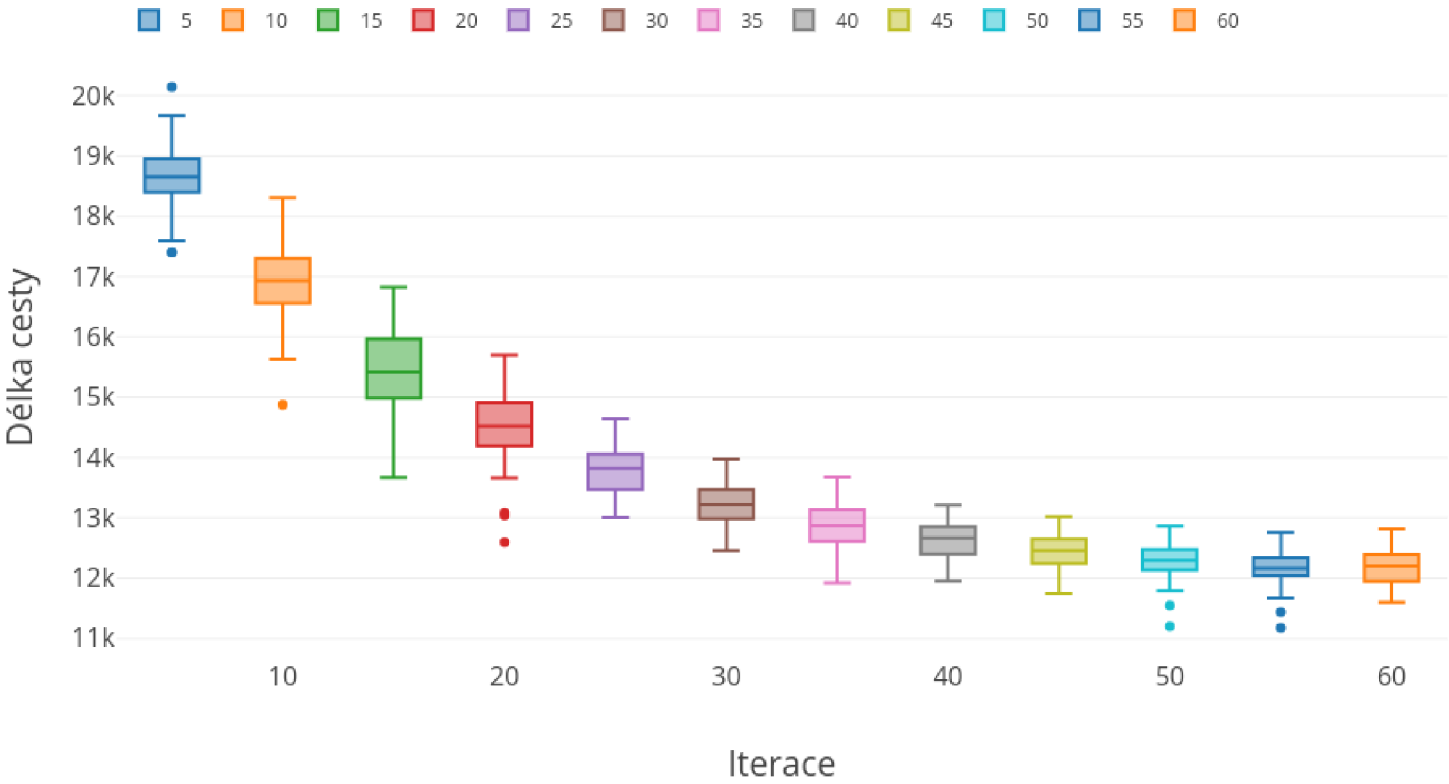
Dell inspiration 15 7000	Parametry
Procesor	8×Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz
RAM paměť	8 Gb
Disk	SSH

Tabulka 4.6: Parametry PC na kterém byli testovány sety s 15-ti a 38-mi městy.

qa194	Nejlepší nalezená cesta	Nejhorší výsledek	Průměr	Medián
Délka cesty v km	10542.9321	11313.9645	11109.0283	11126.5378

Tabulka 4.7: Tabulka obsahující statistické výsledky nalezených řešení programu.

Knihovna TSLIB uvádí optimální cestu 9352km, kdy se tedy jedná o poměrně velkou odchylku o 11,295%. Ovšem vzhledem k vlastnostem ACS algoritmu a náročnosti zadání se jedná o poměrně dobrý výsledek. Rozdíl ve výpočetní náročnosti lze také vypozorovat v grafu 4.14, který zobrazuje statistická data průběhu výpočtu algoritmu. Na rozdíl od zadání dj38 trvá algoritmu mnohem delší dobu, než se usadí na nějaké hladině řešení, kdy zde klesání trvalo přibližně 55 iterací oproti předchozímu zadání, kde konvergence přestala přibližně v desátém cyklu.



Obrázek 4.14: Boxplot znázorňující výsledky nalezených nejlepších cest do šedesáté iterace z 500-ti iterací, kdy data jsou zpracována z 96-ti vzorků pro každou znázorněnou iteraci.

4.1.6 Shrnutí řešení TSP

Z uvedených výsledků je patrné, že zvolený mravenčí algoritmus je schopen úspěšně řešit různé instance TSP. Také se ukázalo, že přesnost algoritmu a rychlost konvergence k výsledkům je závislá na náročnosti zadání, což je ovšem logický důsledek vlastností problému obchodního cestujícího. Dle očekávání je nejlepší výsledek pro 190 měst více vzdálen předpokládanému optimu a to z důvodu vysoké výpočetní náročnosti. Na druhou stranu pro jednodušší zadání byl algoritmus schopen nalézt i dané optimální řešení.

4.2 Experimentalní výsledky pro problém synchronizace

Jelikož se v této sekci budeme zabývat výzkumnou částí této práce, důraz bude kladen na vlastní zkoumání chování algoritmu při daném nastavení a hlavně výsledků aplikace ACS na nalezení transformační funkce řešící problém synchronizace pro CA. Co tedy budeme sledovat? U problému tohoto typu má smysl například sledovat rychlost konvergence k požadovanému řešení a úspěšnost nalezení plně funkčního řešeni v rámci daného počtu iterací, například ze 100 běhů. Na druhou stranu nemá smysl zavádět toleranci odchylky od optima, protože tato úloha (i z pohledu fitness) má čistě diskrétní charakter a požadované chování CA je přesně dáno.

4.2.1 Vliv konstanty q_0 na chování algoritmu

Konstanta q_0 bude mít v tomto případě podstatně jiný vliv na chování algoritmu než měla u TSP, což ovšem neznamená, že se nejedná o důležitou hodnotu. Důvodem rozdílu vlivu této konstanty u problému synchronizace a u TSP je ten, že oba tyto problémy se aplikují na zásadně rozdílné grafy. Zatímco TSP grafy se různí, tj. tvar, velikost atd. závisí čistě na zadání a tudíž pro každou sadu měst vypadá graf pokaždé jinak. U problému synchronizace v 1D celulárním automatu je graf pokaždé principiálně stejný. Jediná věc, která má na graf potencionální vliv, je šířka buněčného okolí celulárního automatu, která určuje délku grafu.

V této práci se ale pouze zaměříme na jednotnou hodnotu okolí a tím pádem graf bude při všech výpočtech stejný. Vztah hodnoty q_0 k vlastnostem grafu je následující. U TSP se mravenci rozhodovali na základě pravidla přechodů, které počítalo pravděpodobnosti přesunu agenta do následujícího města. Konstanta hrála velikou roli u chování mravenců v tom, jestli spíše vyhledávali nové cesty (možné výsledky), nebo preferovali již nějaké nalezené řešení. Jelikož agenti měli většinou na výběr z velkého počtu měst, bylo velice důležité najít rovnováhu, aby se algoritmus choval co nejlépe. U problému synchronizace se agent v každém kroku průchodu grafem rozhoduje pouze ze dvou možných variant kam se vydá, tedy jestli půjde do uzlu reprezentující binární hodnotu 0 nebo 1. Tento zásadní rozdíl bude mít nejspíše za následek to, že konstanta q_0 bude mít jiný vliv na chování algoritmu než u TSP.

Experimenty proběhly následovně. Bylo spuštěno pět verzí programu, každý s jinými hodnotami parametru q_0 a se stejně zadanými parametry c tedy počet iterací (pokusů) pro nalezení transformační funkce byl nastaven na hodnotu 200, počet mravenců m je 60, okolí celulárního automatu je nastaveno na $r = 2$, feromonová hodnota ρ na všech hranách grafu je nastavená na 0.1 a konstanta reprezentující vypařování feromonů obsahuje hodnotu 0.05.

Každý program s různou hodnotou q_0 byl spuštěn 100-krát. Na začátku spuštění bylo vhodně vybráno číslo z rozsahu 0 až 1023 (celulární automat o deseti buňkách může binárně zakódovat pouze čísla z tohoto rozsahu), které reprezentovalo iniciační stav automatu, a každá varianta parametrů hledala řešení pro ten samý počáteční stav automatu. Cílem tedy nebylo obecné řešení problému synchronizace, ale nalezení přechodové funkce pouze pro jediný počáteční stav. Důvodem je ověření konceptu použití MA v této úloze, jelikož předchozí pokusy s obecným řešením tohoto problému pomocí MA nebyly úspěšné. Pokud program nalezne transformační funkci, tak daný cyklus, ve kterém se tak stalo, se ukončil a vypsala se hláška o úspěšnosti výsledku či iteraci, ve které výsledek nalezl a běh MA byl považován za úspěšný. V opačném případě po doběhnutí všech sta cyklů vypsal pouze číslo pokusu.

q_0	0.1	0.3	0.5	0.7	0.9
Počet nálezů	142	100	64	40	18
Procentuální úspěšnost	70.647%	49.751%	31.841%	19.9%	8.9552%

Tabulka 4.8: Tabulka obsahující procentuální úspěšnost nalezených výsledku z 200 pokusů programu.

Z výsledků v tabulce 4.8 je jasně vidět, že čím menší hodnota q_0 tím lepší výsledky programu jsme dostali. Je to z toho důvodu, že konkrétně u tohoto problému je velká míra zkoumání velice žádoucí. Pokud se agent rozhoduje na základě algoritmu 3 a pravděpodobnost se vypočítá pomocí feromonových hodnot dle rovnice (3.1), pak při rozhodování mezi

dvěma místy při nízké hodnotě q_0 je i tak zaručen kvalitní poměr hledání nových cest a upřednostňování těch cest s vyšší intenzitou feromonů.

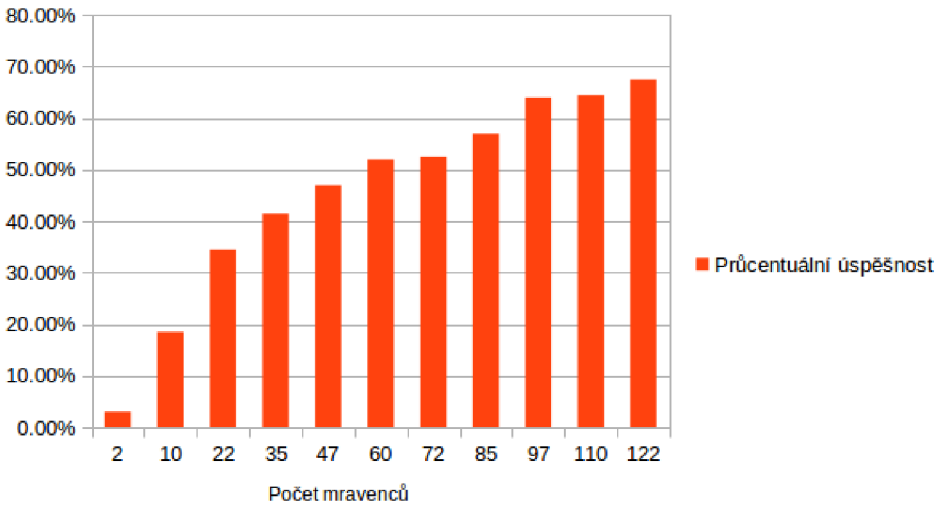
q_0	Nejlepší iterace	Nejhorší iterace	Medián	Průměr	Počet nálezů
0.1	2	65	11.5	14.486	142
0.3	1	72	9	12.31	100
0.5	2	89	7	10.781	64
0.7	2	83	6.5	10.2	40
0.9	1	45	3	7.0556	18

Tabulka 4.9: Tabulka obsahující statistické údaje pro ukázku konvergence programu k výsledku.

Z tabulky 4.9 lze vyčíst několik zajímavých údajů. Každé nastavení nalezlo řešení alespoň v jednom pokusu již při první nebo druhé iteraci, což je dáno spíše náhodou, protože program byl nastaven na 60 mravenců, tak v každé iteraci je testováno až 60 různých transformačních funkcí, tudíž je i poměrně dobrá šance, že při některé z 200 spuštění se náhodně trefí již při první iteraci. Ovšem na toto má také veliký podíl omezení velikosti transformační funkce i celulárního automatu. Další věc, které je si třeba všimnout je, že čím větší hodnota q_0 je, tím méně má algoritmus úspěšnost, ale dříve konverguje k řešení. Hodnoty mediánu a průměru se zmenšují ekvivalentně s rostoucí hodnotou q_0, což je dáno chováním algoritmu při daných hodnotách. Jako u TSP algoritmu s vyšší hodnotou konstanty q_0 konverguje rychle k nějakému řešení. Ovšem v hodně případech konvergoval k špatnému řešení a to vyústilo v malou procentuální úspěšnost.

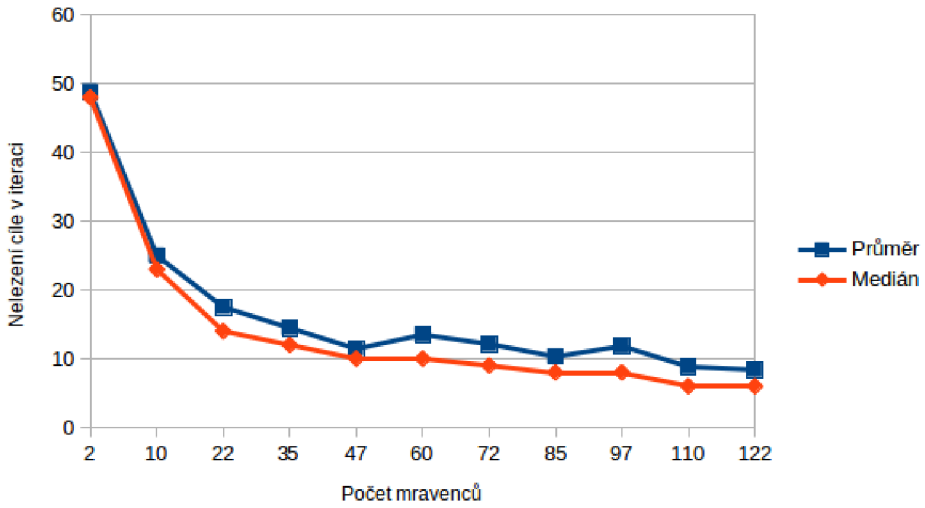
4.2.2 Poměr úspěšnosti a výpočetní náročnosti při různém počtu mravenců

Další možnost, jak významně ovlivnit algoritmus, je měnit počet agentů hledajících řešení. Jak už bylo řešeno v sekci 4.2.1, počet mravenců znamená minimální kvantitu testovaných transformačních funkcí v jedné iteraci. Teoreticky by tedy větší počet mravenců měl znamenat rychlejší konvergenci k cíli a větší procentuální úspěšnost programu. Ovšem s větším počtem agentů také přichází výpočetní náročnost. Program tedy bude spuštěn s různým počtem mravenců a to konkrétně $m = 2, 10, 22, 35, 47, 60, 72, 85, 110, 122$. Tato sekce si klade za úkol zjistit vztah mezi vybranými parametry a výpočetní náročností algoritmu při řešení problému synchronizace v CA.



Obrázek 4.15: Tabluka znázorňující úspěšnost algoritmu při daném počtu mravenců

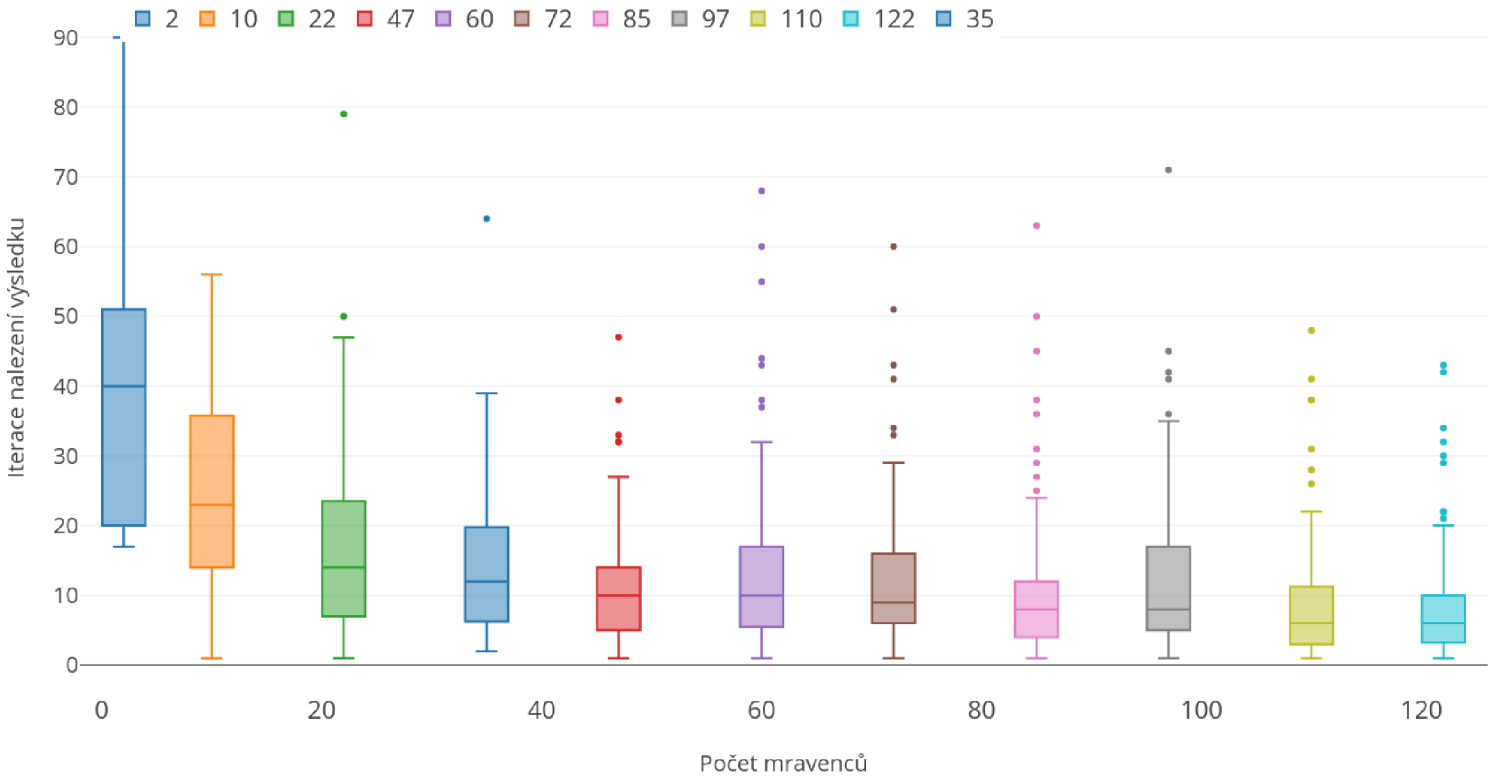
Jak je v grafu 4.15 vidět, potvrdila se hypotéza, že čím větší počet mravenců je, tím větší má algoritmus úspěšnost nalezení řešení. Výsledek grafu přibližně připomíná průběh logaritmické funkce. Tedy po nějakém určitém počtu agentů už nemá moc smysl zvětšovat jejích počet, s větším počtem mravenců přichází větší výpočetní náročnost, ale výsledky se zlepšují minimálně.



Obrázek 4.16: Tabluka ukazující konvergenci algoritmu k cíli při daném počtu mravenců

Graf 4.16 ukazuje, kolik iterací je průměrně (a medián) potřeba pro nalezení korektního řešení pro daný počet mravenců, kdy v prvotní fázi (tedy malý počet mravenců) je vidět veliké zlepšení při konvergenci k cíli, ale postupně hladina ustane a hodnoty zůstávají přibližně na stejné hladině. Rozdíl je při vyšších počtech mravenců opět minimální. Statistické výsledky k testování konvergence doplňuje boxplot4.17, kde je opět vidět zlepšení výsledků s počtem mravenců, kdy s větším počtem mravenců se střední část krabicového

grafu přibližně zmenšuje a úsečka mediánů se přibližuje k minimu. Rozdíly při malém počtu mravenců a velkého počtu mravenců jsou zde ovšem taktéž zjevné.



Obrázek 4.17: Boxplot znázorňující konvergenci algoritmu k výsledkům

4.2.3 Vliv velikosti celulárního automatu na algoritmus

Doposud se práce zaobírala experimentováním nad celulárním automatem s pevně danou šířkou, tedy počet buněk byl roven deseti. Předmětem zkoumání v této sekci bude chování algoritmu pokud tento prostor rozšíříme. Cíl bude zhodnocení statistických hodnot, především opět úspěšnost algoritmu a také jeho tendenci konvergovat ke kýženému cíli. Nastavení algoritmu bude vycházet z informací na stranách 37 a 36. parametry byly zvoleny následně:

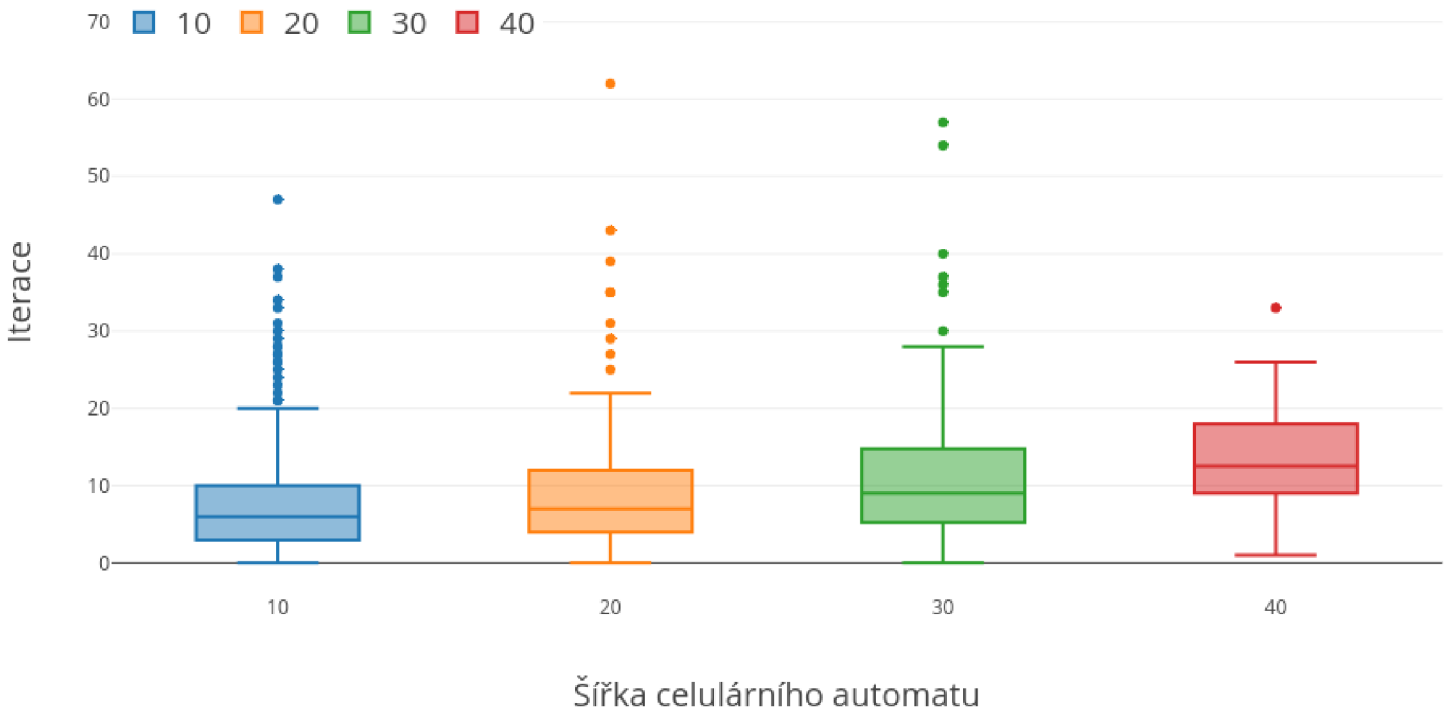
m	n	q_0	ω	c	s
120	5	0.3	0.05	100	10
120	5	0.3	0.05	100	20
120	5	0.3	0.05	100	30
120	5	0.3	0.05	100	40

Tabulka 4.10: s značí šířku celulárního automatu a n velikost buněčného okolí.

Počet buněk	10	20	30	40	Počet pokusů
Procentuální úspěšnost	68.25%	45.5%	24.75%	9.5%	400

Tabulka 4.11: Tabulka obsahující statistické údaje zpracovaných ze 400 spuštění programu pro ukázku konvergence programu k výsledku.

V tabulce 4.11 je vidět poměrně veliký pokles úspěšnosti se zvyšující se šířkou automatu. Ovšem je třeba si uvědomit, že větší šířka celulárního automatu dělá nalezení takové transformační funkce, která řeší problém synchronizace pro daný iniciační stav podstatně složitější. Dost podstatné je také zdůraznit poměr velikosti celulárního automatu a jeho buněčného okolí, které je v obou případech nastaveno na 5 buněk ($r = 2$). Při velikosti celulárního automatu rovné desíti je velice pravděpodobné, že pro nějaký iniciační stav celulárního automatu existuje mnoho transformačních funkcí, které řeší problém synchronizace. Se zvětšující se šířkou automatu, tato pravděpodobnost klesá a je stále obtížnější nalézt transformační funkce s okolím $r = 2$. Ovšem při zvětšení okolí na hodnotu $r = 3$ bude okolí každé buňky obsahovat 7 buněk. Sedmimístné binární číslo může zakódovat čísla v rozsahu 0 - 127 a tudíž by počet potencionálních transformačních funkcí vzrostl na hodnotu 2^{127}, což už je obrovské číslo. Je tedy možné, že existuje přímá závislost mezi šířkou CA a velikostí jeho buněčného okolí na úspěšnost nalezení řešení. Rychlost konvergence k výsledku je se zvyšující se šířkou automatu horší, ale jedná se zde o poměrně malé zhoršení. Výsledky jsou k vidění v grafu 4.18



Obrázek 4.18: statistické zpracování počtu iterací potřebných k nalezení řešení pro vybrané šířky CA.

Je zřejmé, že mravenčí algoritmus dokázal poskytnout s rozumnou četností řešení dané úlohy pro různé velikosti CA a to pouze s uvažováním pevně daného buněčného okolí i pro poměrně široké celulární automaty.

Obrázek 4.19: Bitmapový obrázek který vykresluje průběh transformace celulárního automatu v jednom z pokusů se šířkou CA $s = 40$, který nalezl takovou transformační funkci, která vedla pro daný iniciační stav k synchronizaci. černé body reprezentují binární hodnotu 0 a bílé 1.

4.2.4 Shrnutí řešení pro problém synchronizace

Jelikož tato sekce nabývala výzkumného charakteru, zhodnotíme zde vyzkoumané výsledky vzhledem k navrhnutým postupům popsaných v sekci 3.2.2. Pokud se podíváme na celkovou úspěšnost algoritmu při kvalitním nastavení, algoritmus došel k velice dobrým výsledkům a ty nejlepší dosáhly hodnot nad 70%. Ovšem je potřeba konstatovat, že u tohoto problému a při tom, jak jsme na něj pohlíželi, pro nějaký iniciační stav automatu existuje větší počet transformačních funkcí u buněčného okolí $r = 2$, které vedou k synchronizaci, jinak by takové úspěšnosti nejspíše algoritmus nedosáhl. Statistické údaje nám přiblížily chování algoritmu, ze kterých jsme mohli kvalitně nastavit algoritmus tak, aby splnil námi chtěné požadavky. Otázkou nastává, jaký vliv má buněčné okolí na úspěšnost algoritmu. Pokud by jsme okolí zvětšili na hodnotu $r = 3$, pak by se počet potenciálních transformačních funkcí extrémně zvýšil a to by mohlo mít neblahé důsledky pro hledání synchronizace. Na druhou stranu počet transformačních funkcí vedoucí k synchronizaci pro jeden daný iniciační stav by byl nejspíše taktéž navýšen a výsledek by tedy závisel na poměru navýšení hodnot u obou pojmů.

Kapitola 5

Grafické rozhraní

Součástí zadání této bakalářské práce je vytvoření grafického uživatelského rozhraní. Toto rozhraní má plnit výhradně výukové a demonstrační účely. Hlavní úkol tedy tohoto gui bude seznámit uživatele s principy algoritmů optimalizace mravenčích kolonií. Konkrétně chování a průběh ACS algoritmu. Výsledné gui je rozděleno na 3 části (nebo také vrstvy) a to na hlavní nabídku, hlavní okno programu a podpůrná okna. Pro implementaci byl vybrán jazyk java a jeho grafická odnož javaFx kvůli její multiplatformnosti, širokou paletou implementačních a grafických prvků a celkově pěkného vzhledu aplikací naprogramovaných právě v této třídě javy.

Co tedy aplikace obsahuje? Jako první je třeba zmínit popsání obou problematik, tedy jak Ant colony system, tak i problému obchodního cestujícího, který tento algoritmus v aplikaci řeší. Je to jakási průprava nebo příprava uživatelů, kteří například příliš nevědí o čem tato problematika pojednává, nebo aby se upřesnilo s jakou verzí algoritmu gui pracuje. Samotný program tedy obsahuje mnoho funkcí. Jako první, Uživatel si vytvoří vlastní TSP sadu na které algoritmus poběží, ovšem tato sada z výukových účelů je limitována na daný počet měst. Při vysokém počtu měst by se mohlo stát že graf a doplňkové informace jsou velice nepřehledné, a při malém počtu měst by zase pozbýval algoritmus některé své vlastnosti.

Po vytvoření datové sady se v hlavní programové ploše objeví 4 základní informativní sektory viz popis obrázku A.17. Jako první stojí za zmínku animační graf, který mění stav (vzhled) vzhledem k danému kroku ve kterém se algoritmus nachází. Je to hlavní demonstrační nástroj tohoto gui, který obrazně nejlépe vystihuje chování algoritmu. Graf například zobrazuje výsledky iterací, rozhodování mravence při hledání místa na přesun, nebo rozhodnutí do jakého města se mravenec nakonec vydal.

Další sektor, který má dopomoci uživateli porozumět algoritmu je popis kroku. Každý krok, který algoritmus vykonal a je znázorněn v grafu je slovně popsán. je to doplňující nástroj funkce k obraznému zobrazení grafem. Tyto dva prvky se komplementárně doplňují. Další co může uživatel je sledovat doplňkové informace algoritmu pomocí podpůrných oken. Jedno podpůrné okno obsahuje například tabulku s hodnotami vzdáleností, které se během fungování algoritmu logicky nemění. Dále tabulku s hodnotami feromonů, které jsou již proměnné a aktualizují se při určitých krocích, tedy po provedení lokálního aktualizačního pravidla nebo globálního aktualizačního pravidla. Poslední část demonstrační plochy programu je vyhrazena pro zobrazení paměti mravenců při vytváření řešení. Celkový vzhled a popis různých částí výsledného grafického uživatelského rozhraní je více popsán v příloze 54.

Kapitola 6

Závěr

Tato práce měla několik základních cílů. Ovšem tyto cíle měly jednu věc společnou a tou bylo použití mravenčích algoritmů. První úkol byl použít jeden z rodiny mravenčích algoritmů (byl zvolen Ant colony system) na problém obchodního cestujícího a v podstatě ověřit funkčnost konvergence tohoto algoritm k optimálnímu výsledku. Principy k této problematice použité v této práci byly popsány v sekcích 2.5 a 2.6. Algoritmus byl testován na třech různých instancích problému s různou výpočetní náročností. Výsledky, které testování přineslo spolu s hledáním kvalitního nastavení, se ukázaly jako poměrně kvalitní, i když algoritmus nebyl pro nejnáročnější uvažované zadání (TSP s 194 městy) schopen v omezeném čase nalézt známé optimum, ale našel řešení s odchylkou cca 11% od optima.

Dalším cílem této práce byl výzkum v oblasti celulárních automatů, konkrétně ověření, zda je mravenčí algoritmus schopen návrhu transformačních funkcí řešících problému synchronizace v jednorozměrném celulárním automatu. Prvotně bylo potřeba navrhnout, jakým způsobem by se daly mravenčí algoritmy aplikovat na tento problém a jak tyto algoritmy modifikovat, aby postup algoritmu vyhovoval řešení tohoto typu problému. Nakonec bylo vymyšleno a aplikováno několik nových modifikací algoritmu ACS, které byly popsány v kapitole 3. Za zmínku především stojí koncept konstrukčního grafu, který byl v rámci této práce navržen pro řešení tohoto problému, a dále způsob vyhodnocování výsledků problému synchronizace pomocí nekonečné normy, kdy se tyto principy ukázaly velice efektivní a to především v úspěšnosti a rychlostí konvergence při aplikaci mravenčích algoritmů na tento problém.

Poslední část této práce si kladla za úkol vytvoření grafického uživatelského rozhraní, jenž má plnit výukové a demonstrační účely a bude sloužit především studentům Fakulty informačních technologií na Vysokém učením technickém v Brně, kteří v navazujícím magisterském studiu si zvolí volitelný předmět Aplikované evoluční algoritmy (EVO). Vlastnosti a principy výsledného GUI můžeme vyčíst v kapitoje 5 a podrobný popis jeho funkcionality a vzhledu v příloze v sekci A.2.

Mravenčí algoritmy se tedy ukázaly jako velice silný optimalizační nástrojem s obrovským potenciálem, který má velice široké spektrum uplatnění a flexibility, kde jeho principy lze upravovat pro potřeby daných problémů. Tato práce ukázala, jak lze například přenést tyto principy do světa simulací, tedy konkrétně pro řešení problému na poli celulárních automatů. Zásadní nevýhodou těchto postupů však zůstává nutnost precizního návrhu hlavních částí mravenčího algoritmu pro řešení daného problému, což je úloha netriviální a vyžaduje zkušeného návrháře.

Literatura

[1] Bidlo, M.: *Biologií inspirovaný vývin jako technika evolučního návrhu*. 2018.

[2] Deneubourg, J. L.; Aron, S.; Goss, S.; aj.: The self-organizing exploratory pattern of the argentine ant. *Journal of Insect Behavior*, ročník 3, č. 2, Mar 1990: s. 159–168, ISSN 1572-8889, doi:10.1007/BF01417909.
URL `https://doi.org/10.1007/BF01417909`

[3] DORIGO, M.: *Optimization, learning and natural algorithms*. Dizertační práce, 1992.
URL `https://ci.nii.ac.jp/naid/10000136323/en/`

[4] Dorigo, M.; Birattari, M.; Stützle, T.: Ant Colony Optimization. ročník 1, 12 2006: s. 28–39.

[5] Dorigo, M.; Gambardella, L. M.: Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, ročník 1, č. 1, Apr 1997: s. 53–66, ISSN 1089-778X, doi:10.1109/4235.585892.

[6] Dorigo, M.; Maniezzo, V.; Colorni, A.: Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, ročník 26, č. 1, Feb 1996: s. 29–41, ISSN 1083-4419, doi:10.1109/3477.484436.

[7] Gardner, M.: Mathematical games-The fantastic combinations of John Conway's new solitaire game, Life, 1970. *Scientific American, October*: s. 120–123.

[8] Kulis, A.: Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies. *Scalable Computing: Practice and Experience*, ročník 10, č. 4, 2009.

[9] Martens, D.; Backer, M. D.; Haesen, R.; aj.: Classification With Ant Colony Optimization. *IEEE Transactions on Evolutionary Computation*, ročník 11, č. 5, Oct 2007: s. 651–665, ISSN 1089-778X, doi:10.1109/TEVC.2006.890229.

[10] Moshe, S.: *Evolution of Parallel Cellular Machines*. In *Lecture Notes in Computer Science*, 1997.

[11] Pfahringer, B.: Multi-agent search for open shop scheduling: Adapting the Ant-Q formalism. *Vienna, Austrian Research Institute for Artificial Intelligence*, 1996.

[12] Ramalhinho Lourenco, H.; Serra, D.: Adaptive search heuristics for the generalized assignment problem. *Mathware & soft computing. 2002 Vol. 9 Núm. 2 [-3]*, 2002.

[13] Salhi, S.; Sari, M.: A multi-level composite heuristic for the multi-depot vehicle fleet mix problem. *European Journal of Operational Research*, ročník 103, č. 1, 1997: s. 95–112.

[14] Stutzle, T.: MAX-MIN ant system for quadratic assignment problems. *Germany: Intellektik Group, Department of Computer Science, Darmstadt University of Technology (Report No. AIDA-97-04)*, 1997.

[15] Stützle, T.; Dorigo, M.: Ant Colony Optimization. 01 2004.

[16] Toth, P.; Vigo, D.: Models, relaxations and exact approaches for the capacitated vehicle routing problem. *Discrete Applied Mathematics*, ročník 123, č. 1-3, 2002: s. 487–512.

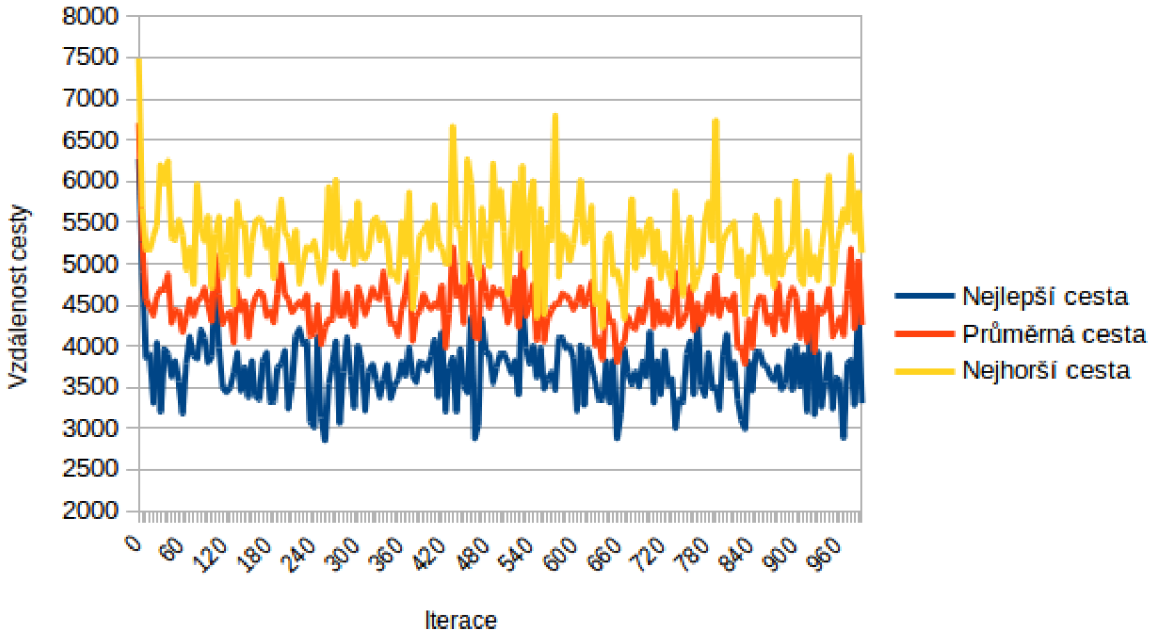
[17] Von Neumann, J.; Burks, A. W.; aj.: Theory of self-reproducing automata. *IEEE Transactions on Neural Networks*, ročník 5, č. 1, 1966: s. 3–14.

Příloha A

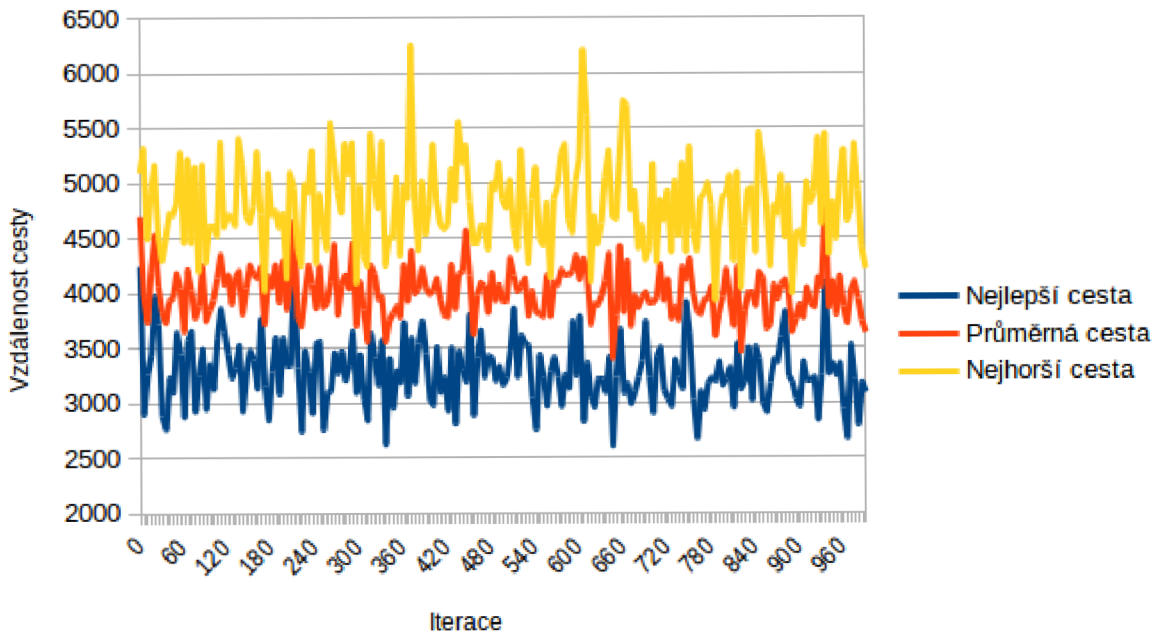
Přílohy

A.1 Doplňující Grafy k sekci 4.1.3

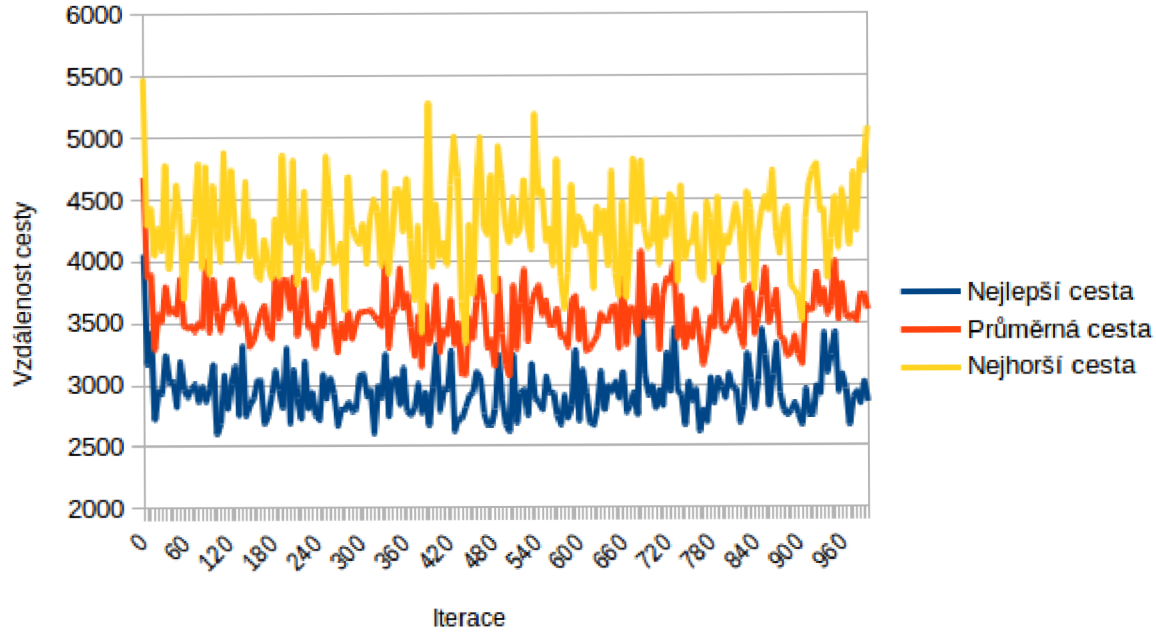
Následující sekce bude věnována ostatním grafům pro hodnoty $\beta = 1, 3$ a 4. Každý graf znázorňuje ukázku průběhu řešení jednoho spuštění programu.



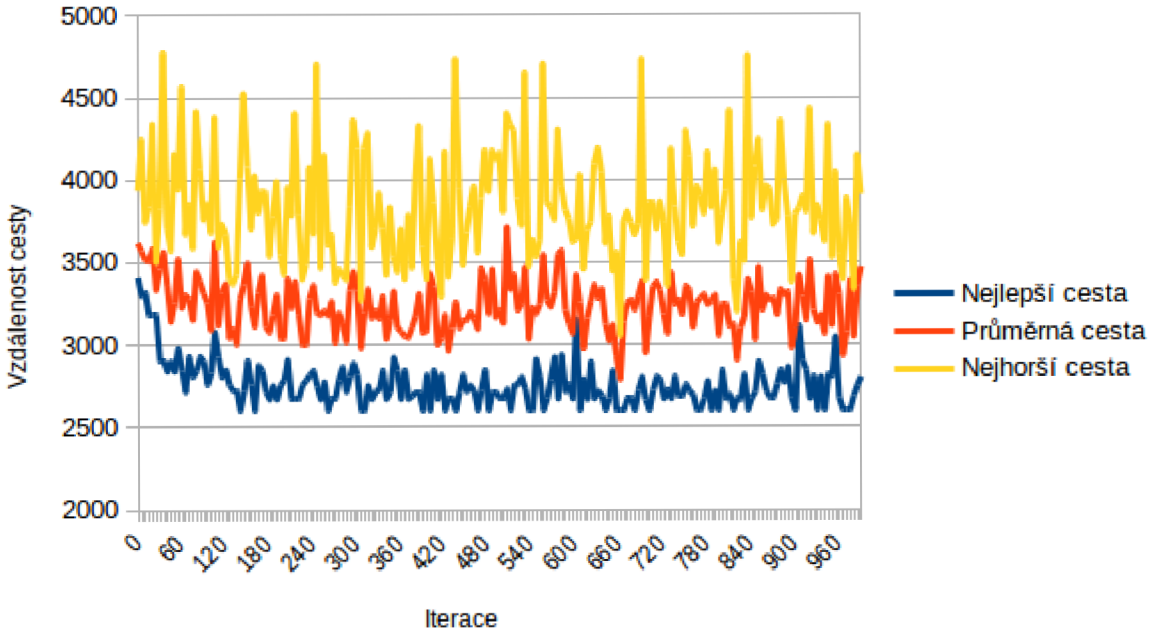
Obrázek A.1: $\beta = 1, q_0 = 0.1$.



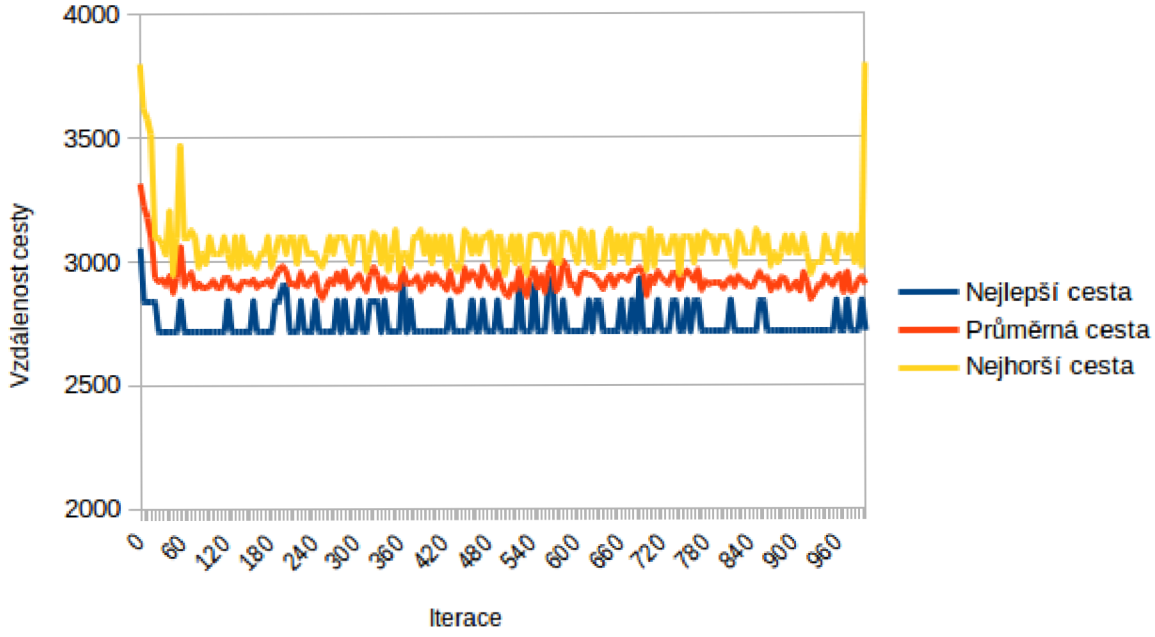
Obrázek A.2: $\beta = 1, q_0 = 0.3$.



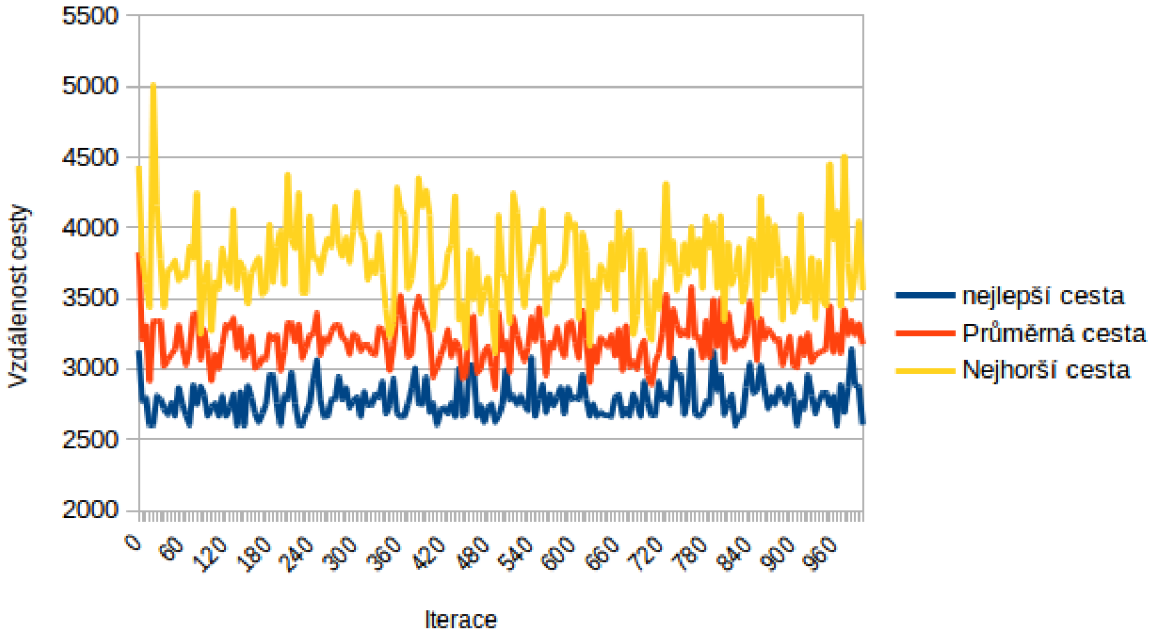
Obrázek A.3: $\beta = 1, q_0 = 0.5$.



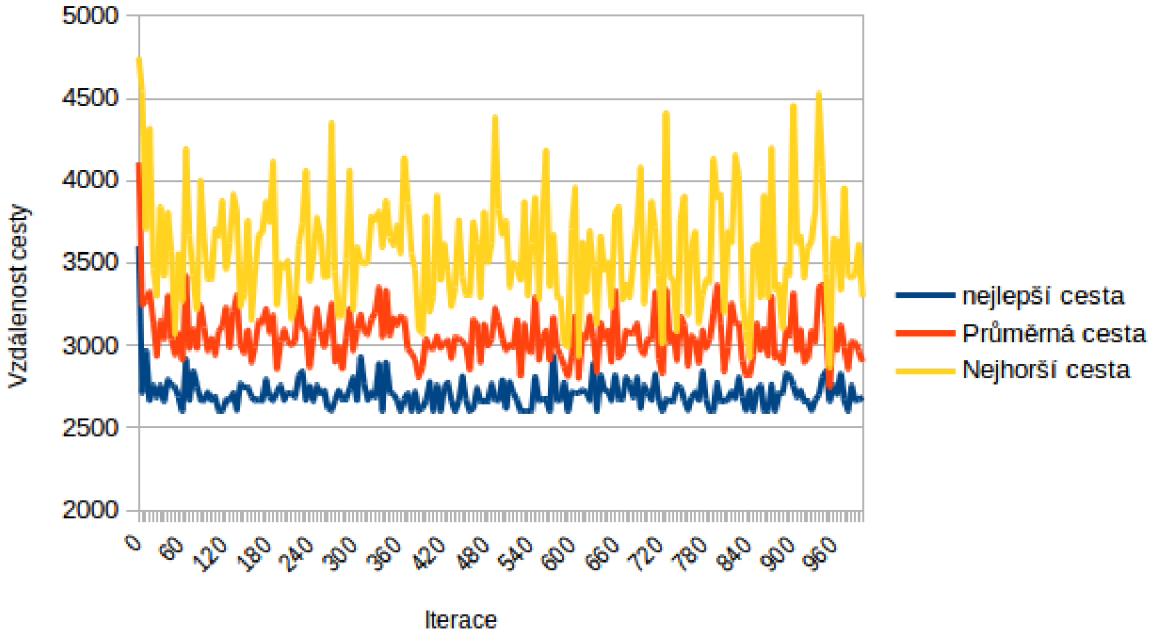
Obrázek A.4: $\beta = 1, q_0 = 0.7$.



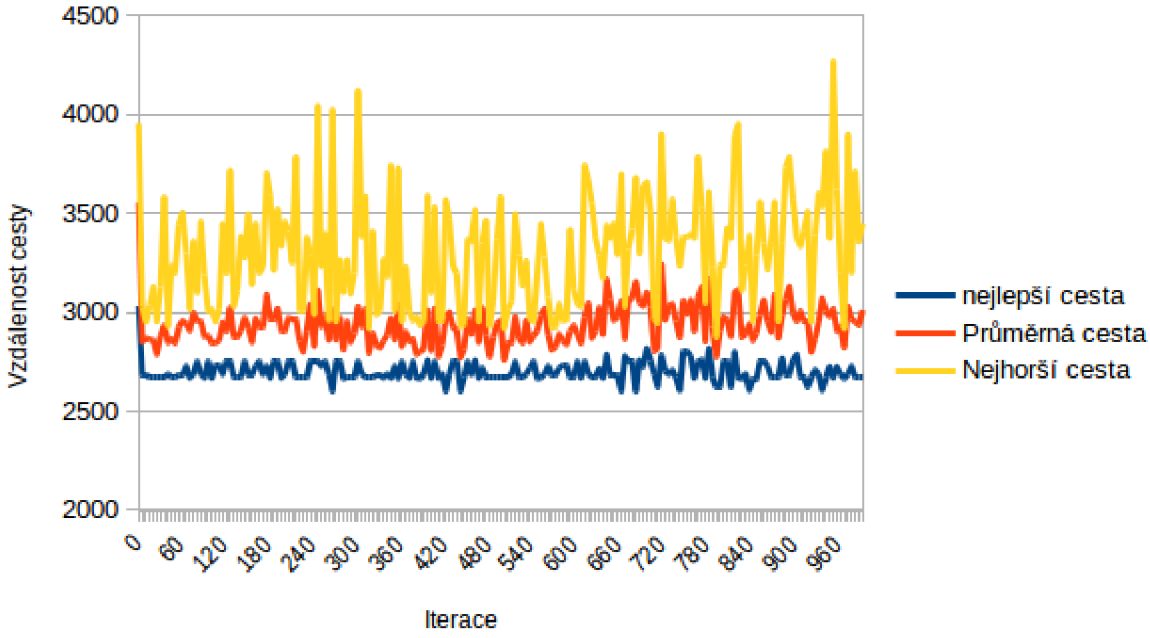
Obrázek A.5: $\beta = 1, q_0 = 0.9$.



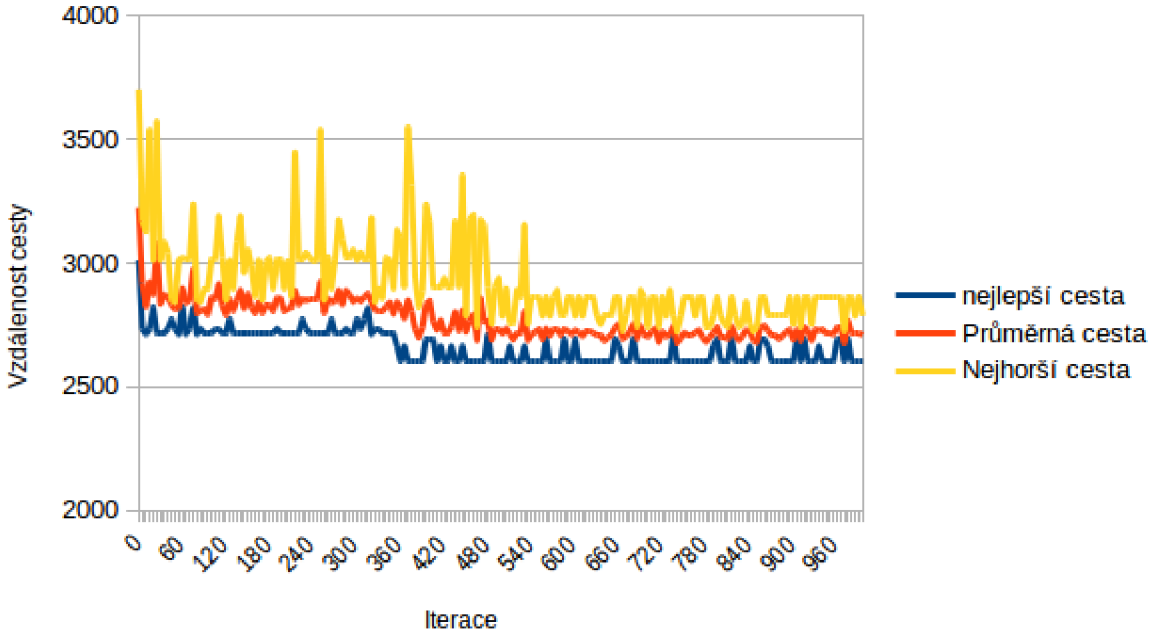
Obrázek A.6: $\beta = 3, q_0 = 0.1$.



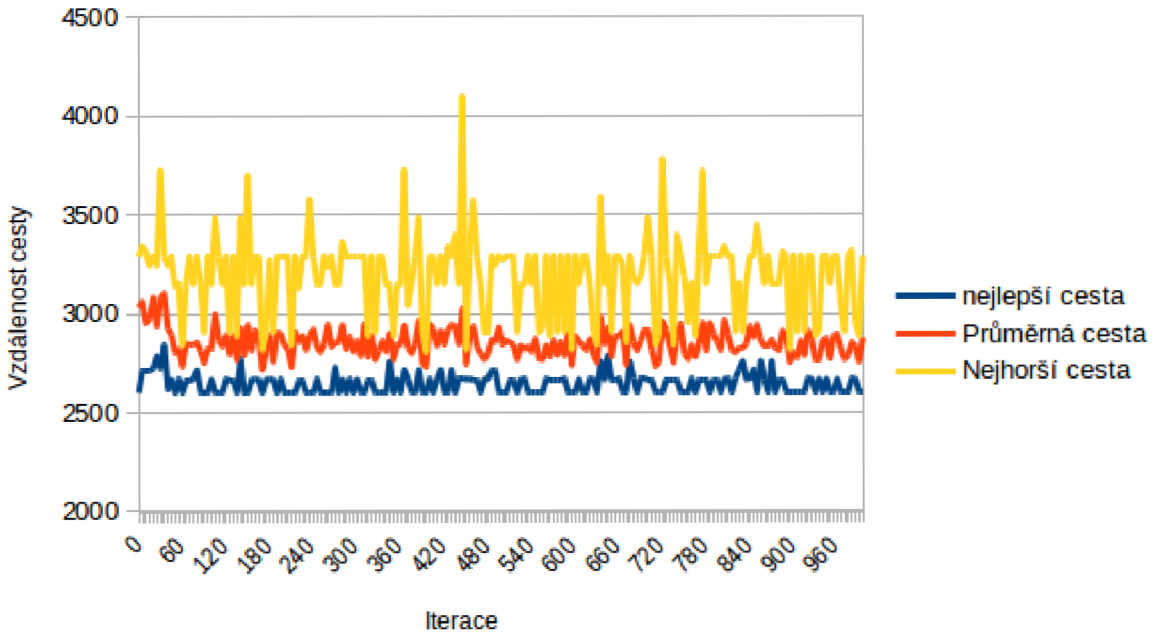
Obrázek A.7: $\beta = 3, q_0 = 0.3$.



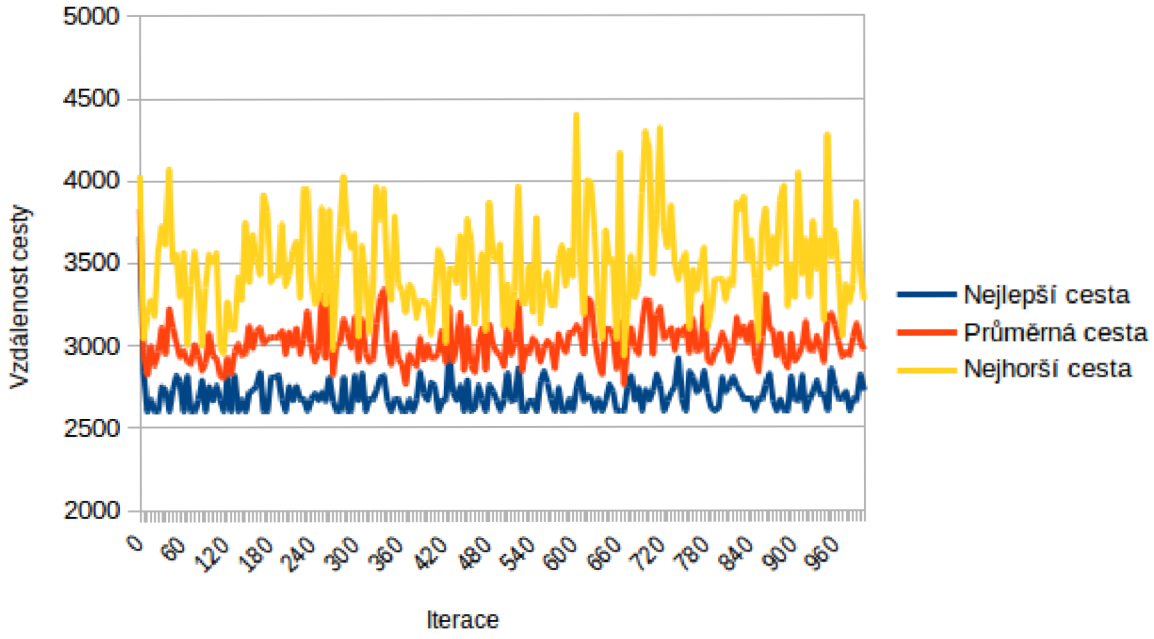
Obrázek A.8: $\beta = 3, q_0 = 0.5$.



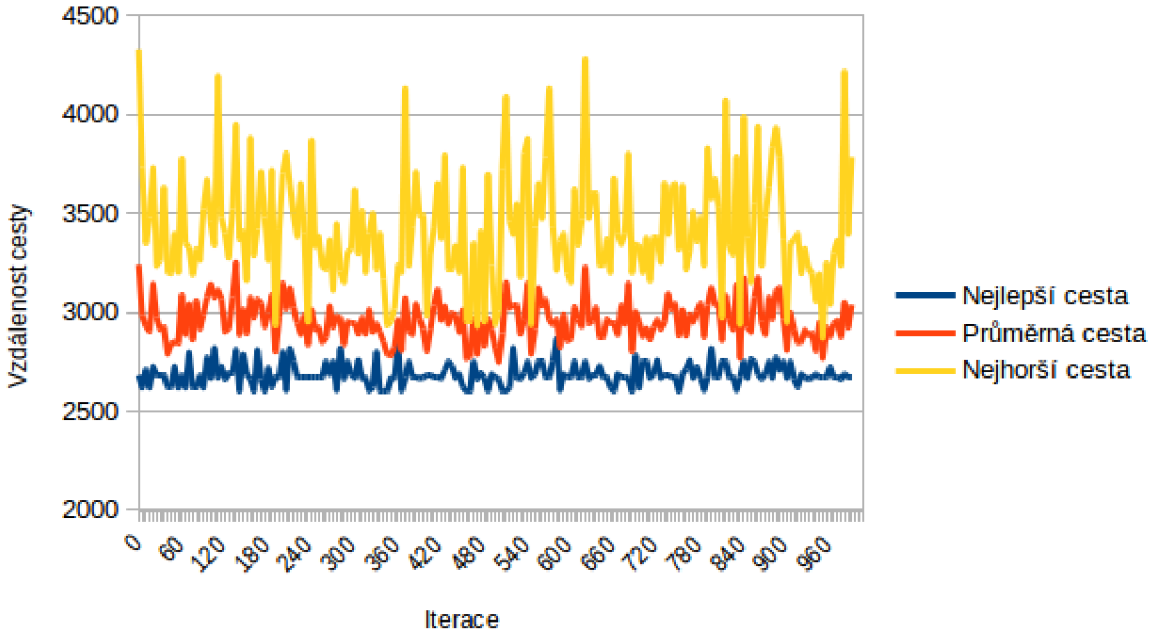
Obrázek A.9: $\beta = 3, q_0 = 0.7$.



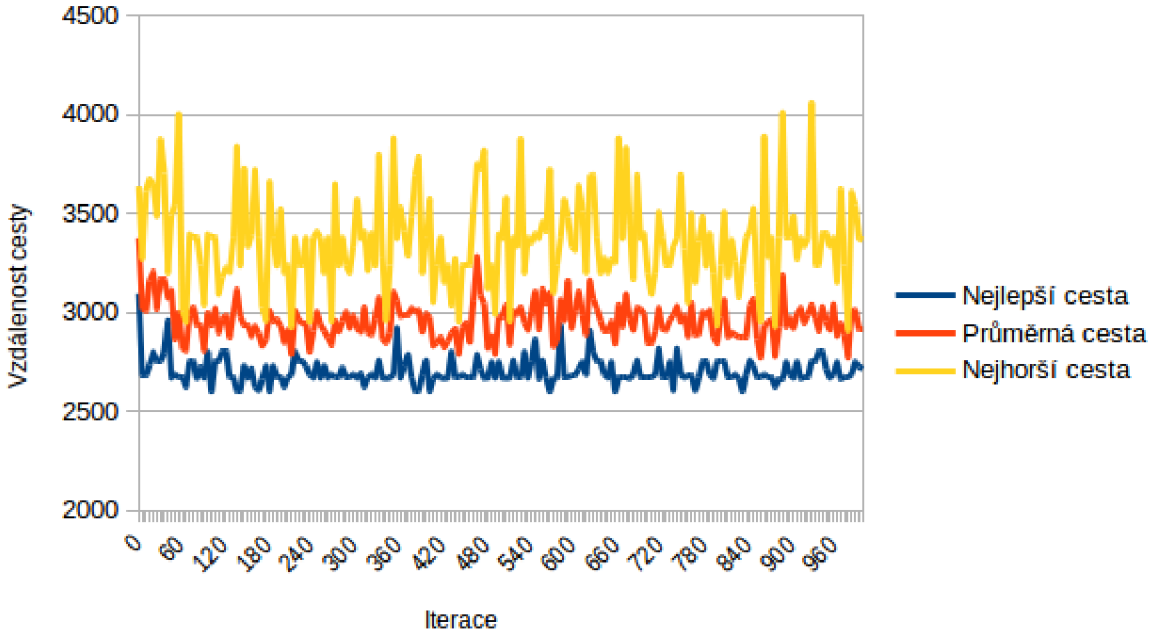
Obrázek A.10: $\beta = 3, q_0 = 0.9$.



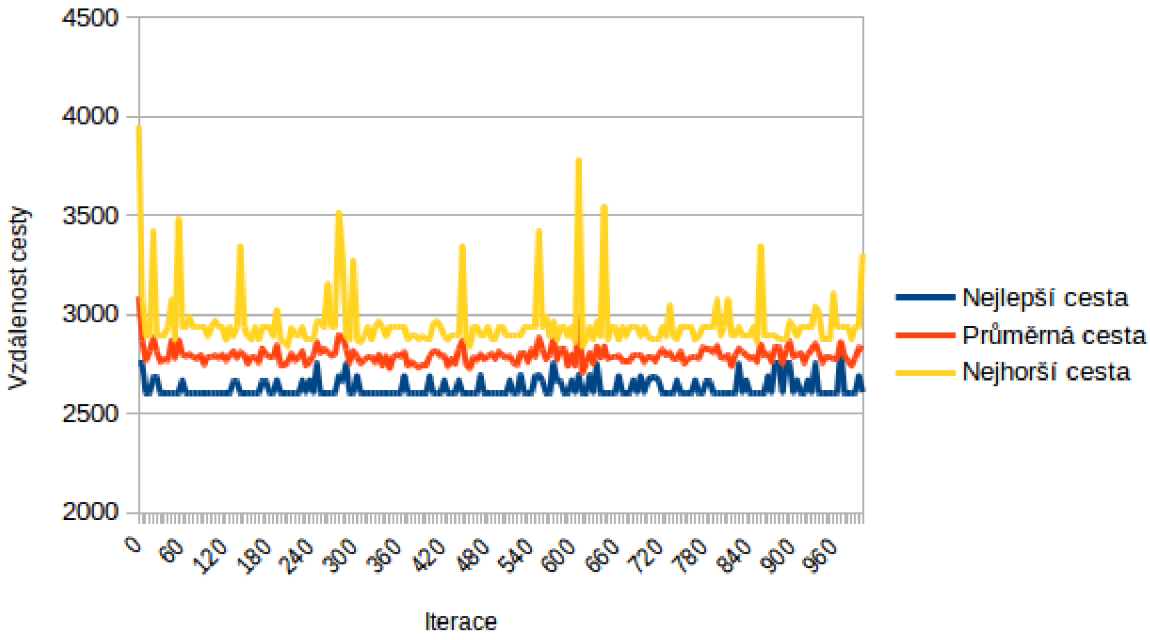
Obrázek A.11: $\beta = 4, q_0 = 0.1$.



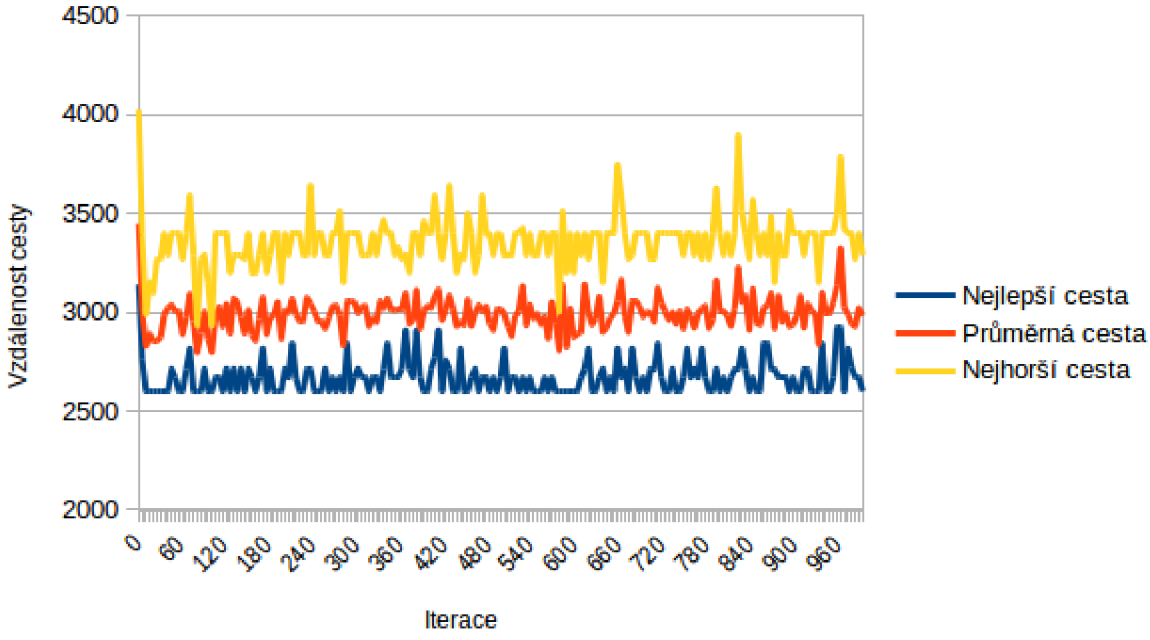
Obrázek A.12: $\beta = 4, q_0 = 0.3$.



Obrázek A.13: $\beta = 4, q_0 = 0.5$.



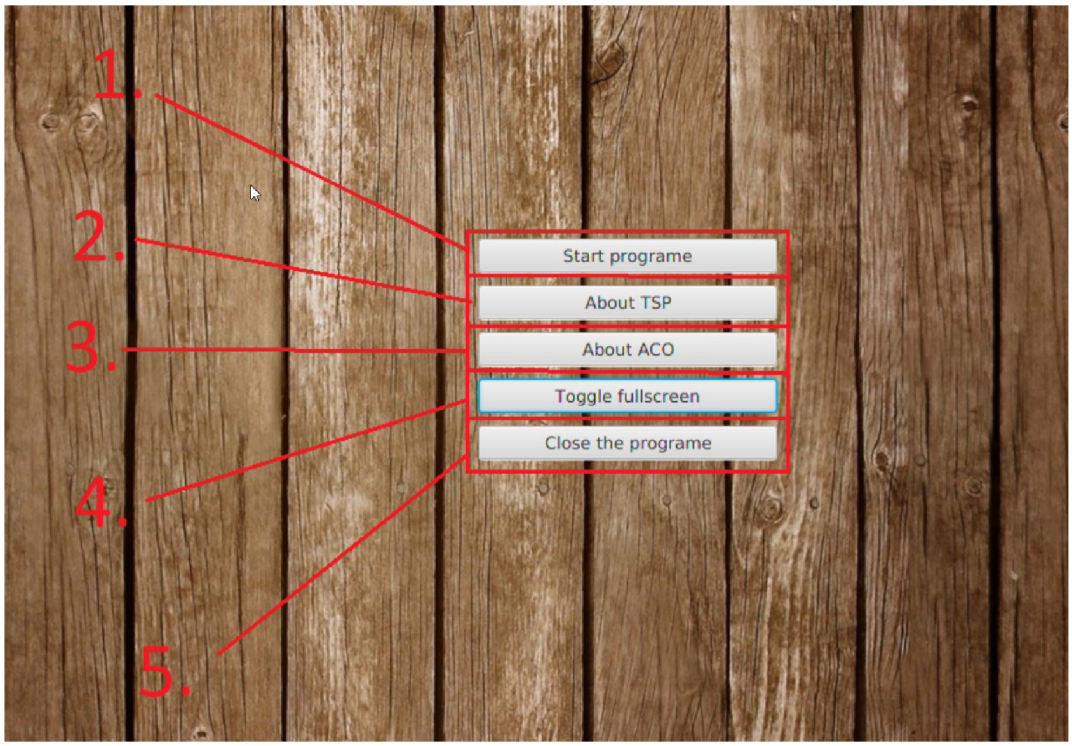
Obrázek A.14: $\beta = 4, q_0 = 0.7$.



Obrázek A.15: $\beta = 4, q_0 = 0.9$.

A.2 Popis grafického uživatelského rozhraní

V této sekci si blíže popíšeme strukturu grafického rozhraní a jeho ovládání.

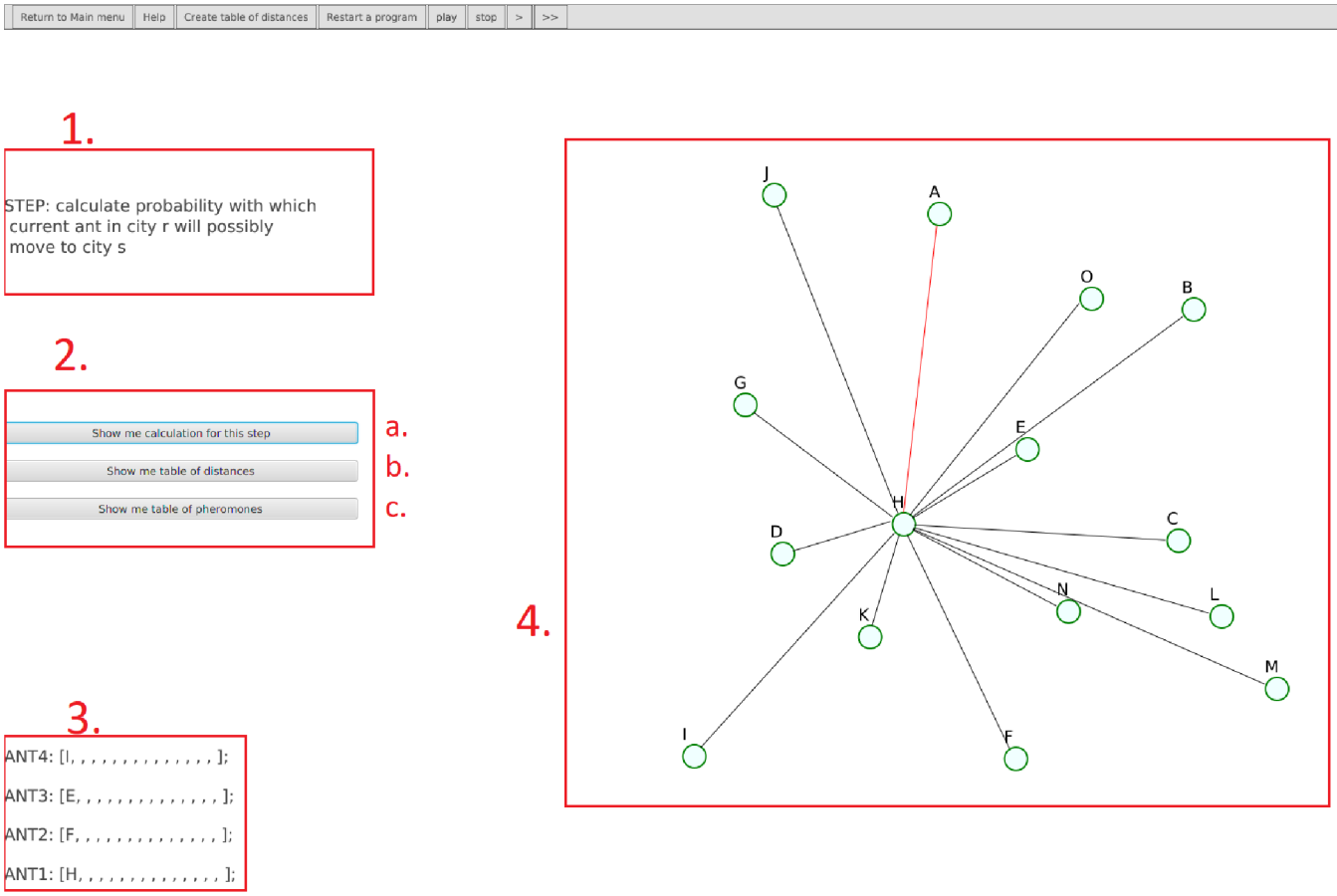


Obrázek A.16: Hlavní nabídka programu.

Hlavní nabídka programu a její tlačítka jsou znázorněny v obrázku A.16 a plní následující funkce:

1. Přejde na (otevře) hlavní programovou plochu, tedy scénu kde probíhá demonstrace algoritmu.
2. Otevře scénu, kde je popsán problém obchodního cestujícího.
3. Otevře scénu, kde je popsán algoritmus Ant colony system.
4. Umožňuje přepínat mezi módem fullscreen a windowscreen.
5. Ukončí program.

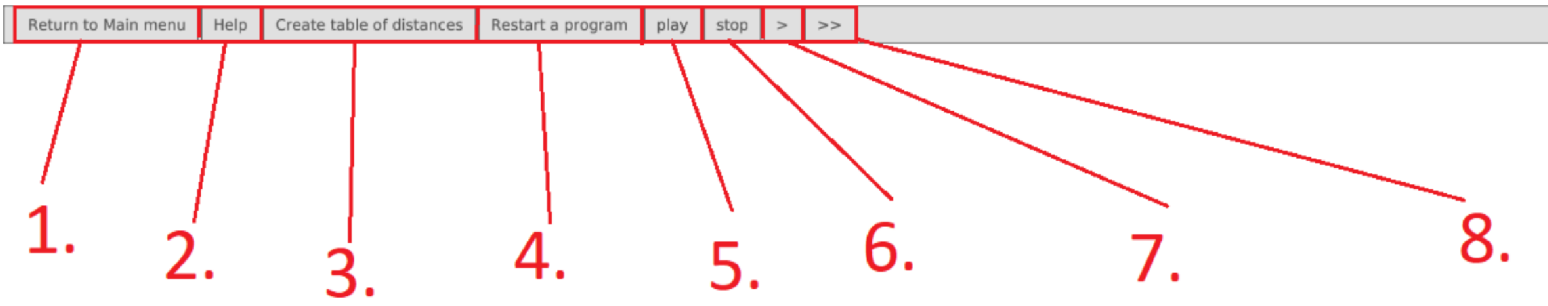
Scény s popisem problematiky TSP a ACS byly přidány hlavně pro uživatele, kteří vůbec netuší o čem tato problematika je, tedy aby úplně neznalí pochopily principy na kterých tato aplikace staví, a hlavně pro definování principů kterými se algoritmus řídí.



Obrázek A.17: Hlavní scéna programu.

Hlavní scéna programu, obr. A.17, je rozdělena na několik částí:

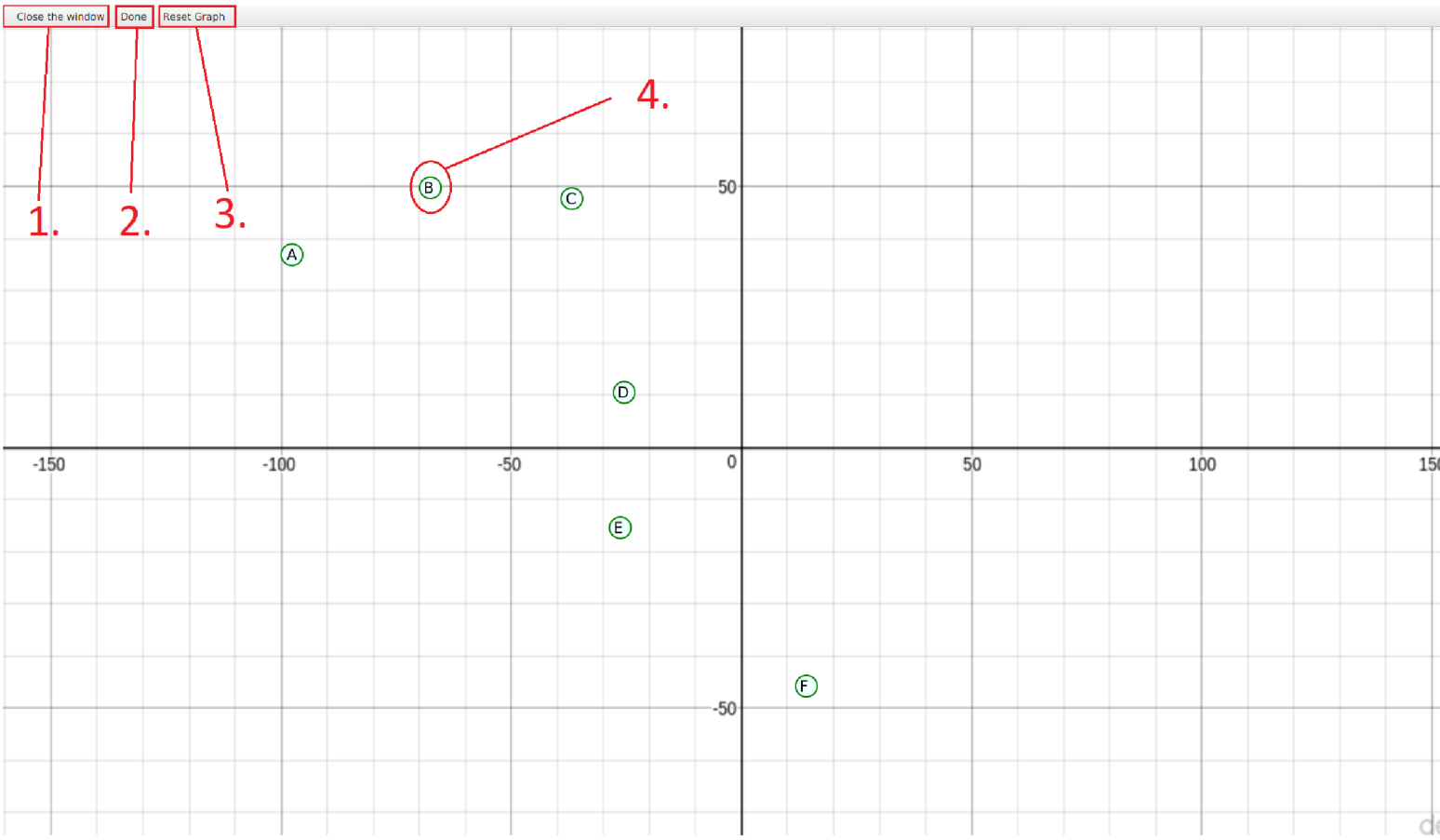
1. První oblast scény má za úkol popsat krok, který právě algoritmus provedl.
2. V druhé oblasti se nacházejí informativní tlačítka. Po stisknutí kteréhokoliv tlačítka se zobrazí informativní okno. Samozřejmě každé tlačítko má za úkol informovat o jiných důležitých informacích k danému běhu ACS algoritmu. nové okna musela být přidána z důvodu prostorové náročnosti informací. Každé informační okno je popsáno zvlášť v u obrázků A.22, A.23 a A.24.
   - 2A: Zobrazí výpočet a vzorec ve LaTeXovém formátu.
   - 2B: Zobrazí tabulku vzdáleností.
   - 2C: Zobrazí tabulku feromonů.
3. Ukazuje aktuální stav paměti mravenců kteří konstruují řešení.
4. Graf který zobrazuje animace kroků ACS algoritmu.



Obrázek A.18

Na obrázku A.18 je vidět hlavní menu programu a jednotlivé tlačítka zastávají následující funkce:

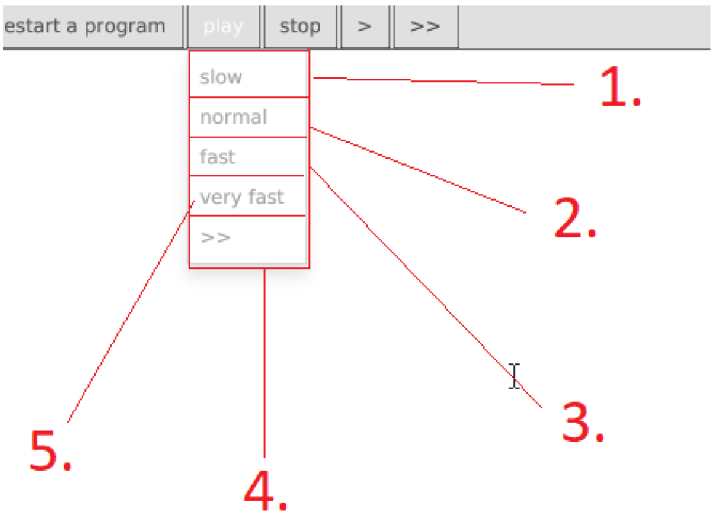
1. Přejde do hlavní nabídky programu (při návratu z hlavní nabídky do hlavního okna programu demonstrační data zůstanou tak jak byla před přesunem do hlavního menu).
2. Help zobrazí nápovědu jak s programem pracovat.
3. Zobrazí okno za účelem vytvoření grafu pro TSP, na kterém poté bude demonstrován ACS. Více je toto okno popsáno na obr. A.19.
4. Vymaže data a nastaví program do iniciačního nastavení (nerestartuje program pouze data).
5. Umožňuje přehrát funkci algoritmu. Více je tato funkce popsána u obr. A.20.
6. Zastaví přehrávání.
7. Provede jeden krok algoritmu.
8. Spustí algoritmus a zastaví se až na konci iterace ACS algoritmu, tedy kdy je už vyřešena nejlepší cesta pro danou iteraci a je také zobrazena.



Obrázek A.19

Okno pro vytvoření grafu je vyznačeno na obr. A.19. Uživatel kliká na Eukleidovský prostor a tím vytváří body, které reprezentují města.

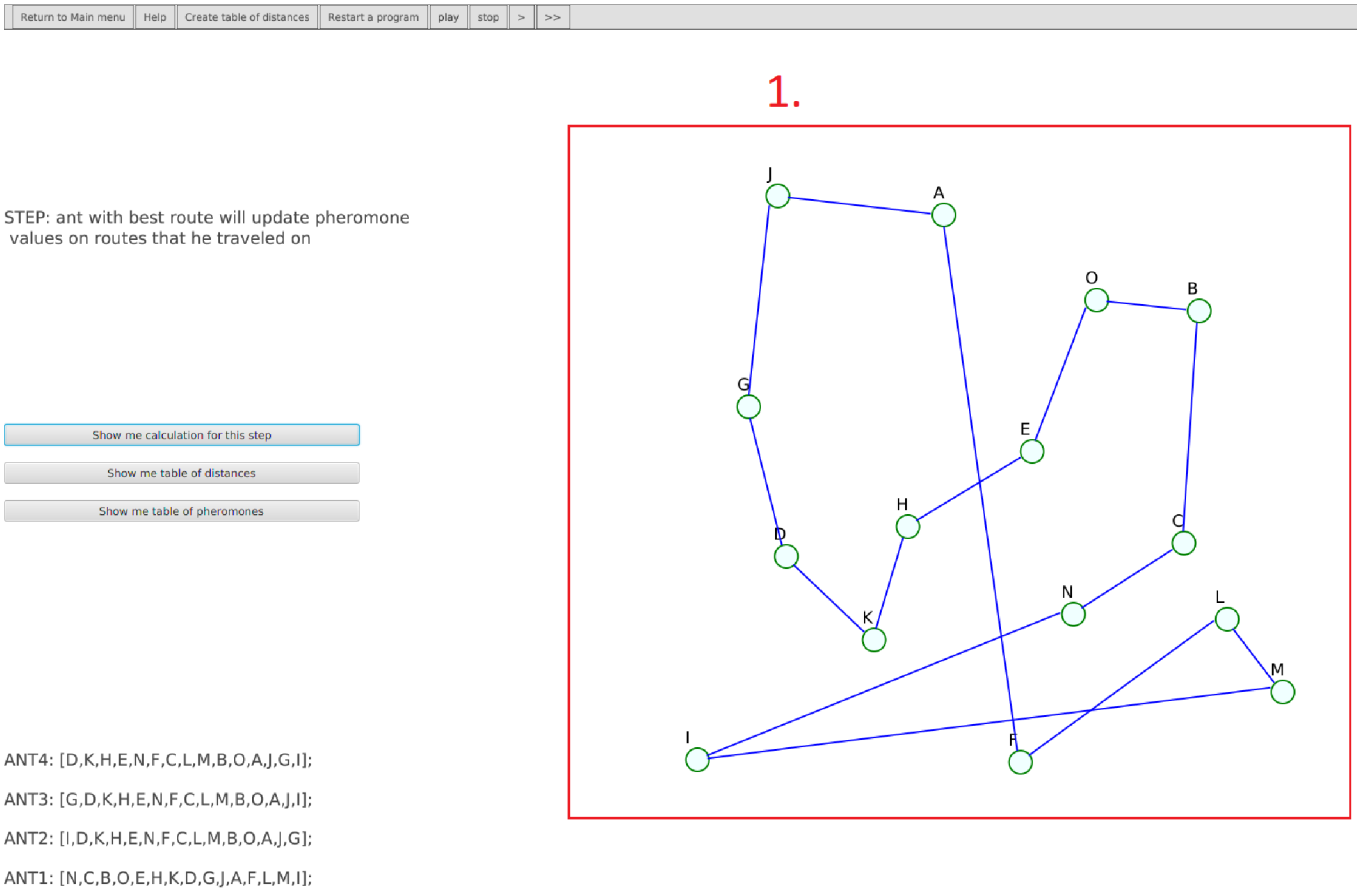
1. Zavře okno a neuloží data pro TSP.
2. Uloží zadané data pro spuštění samotného algoritmu a zavře okno.
3. Restartuje progres zadávání dat.
4. Znázorňuje zástupce vloženého města do prostoru.



Obrázek A.20

Nabídka play umožňuje uživateli plynule spustit kroky algoritmu různou rychlostí a to s následujícímy intervaly:

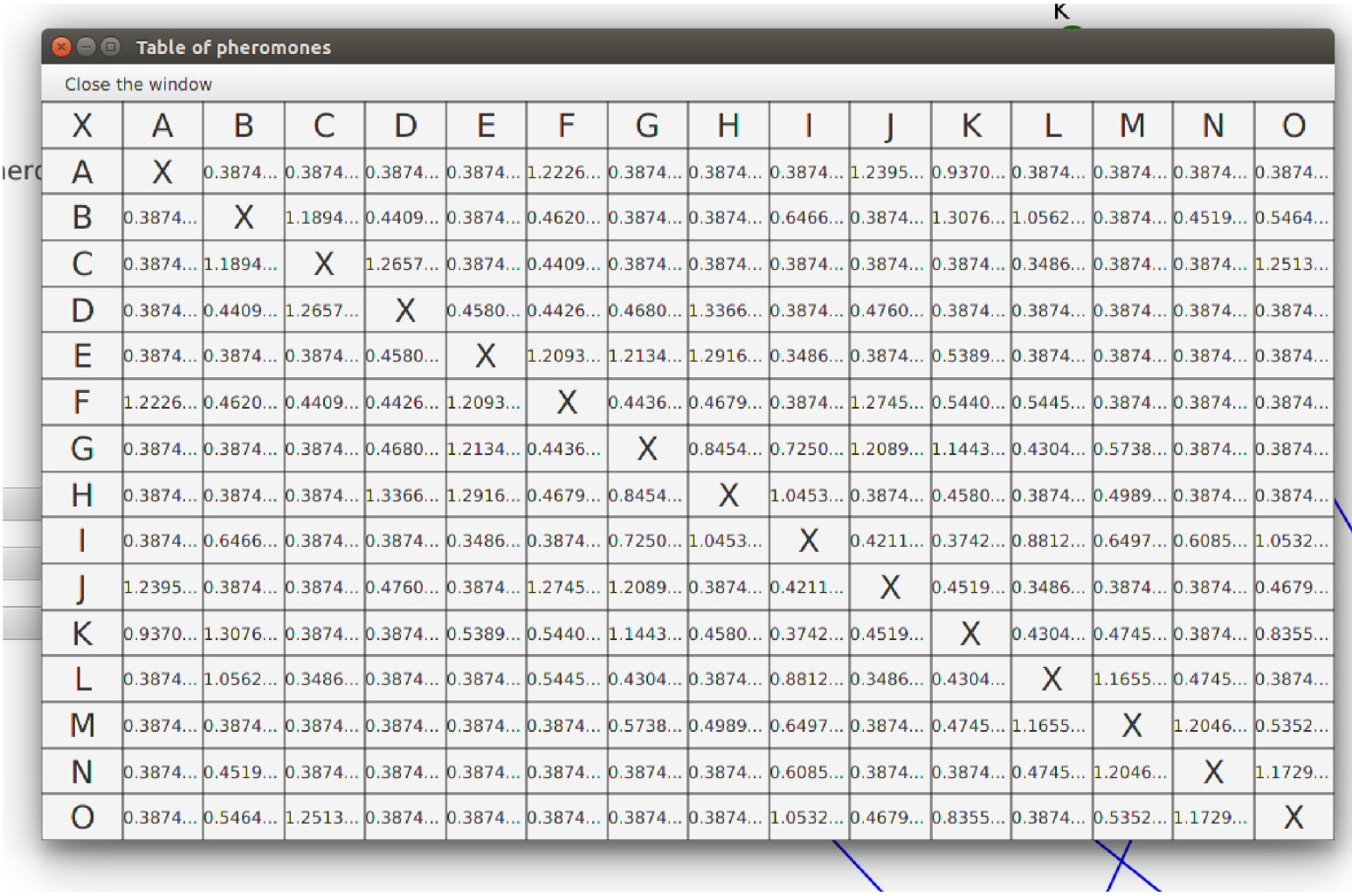
1. Pauza mezi dvěma kroky algoritmu je 3 sekundy.
2. Pauza mezi dvěma kroky algoritmu je 1 sekunda.
3. Pauza mezi dvěma kroky algoritmu je 0.1 sekundy.
4. Pauza mezi dvěma kroky algoritmu je 0.01 sekundy.
5. Algoritmus zobrazuje pouze výsledky každé iterace.



Obrázek A.21: Stav grafu po nalezení částečného řešení v jedné iteraci.

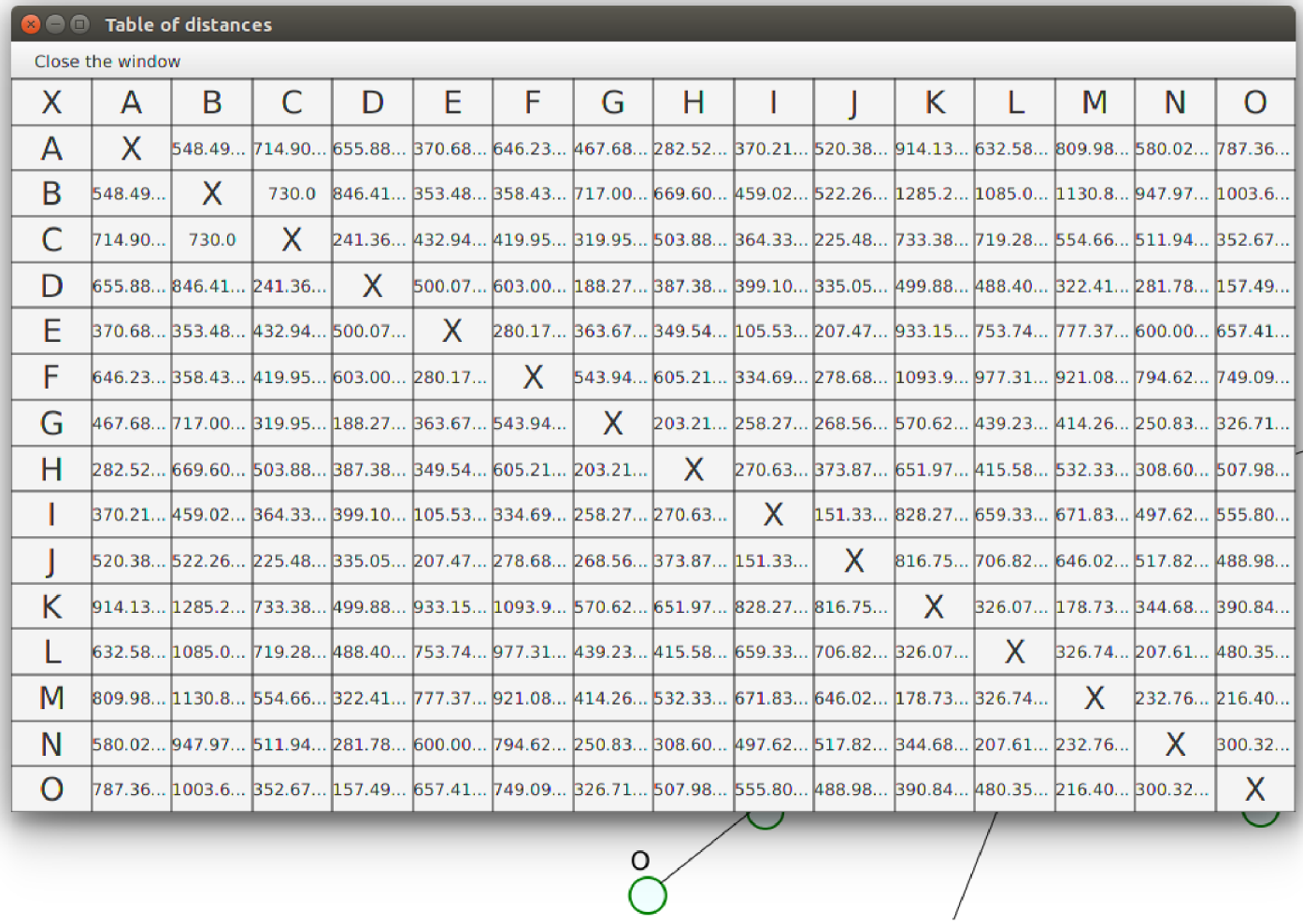
A.2.1 Informativní okna

Účel informativních oken je informovat o důležitých hodnotách či podrobnostech kroků algoritmu. tyto informace byli převedeny do externích oken z důvodu prostorové náročnosti a z důvodu přehlednosti.



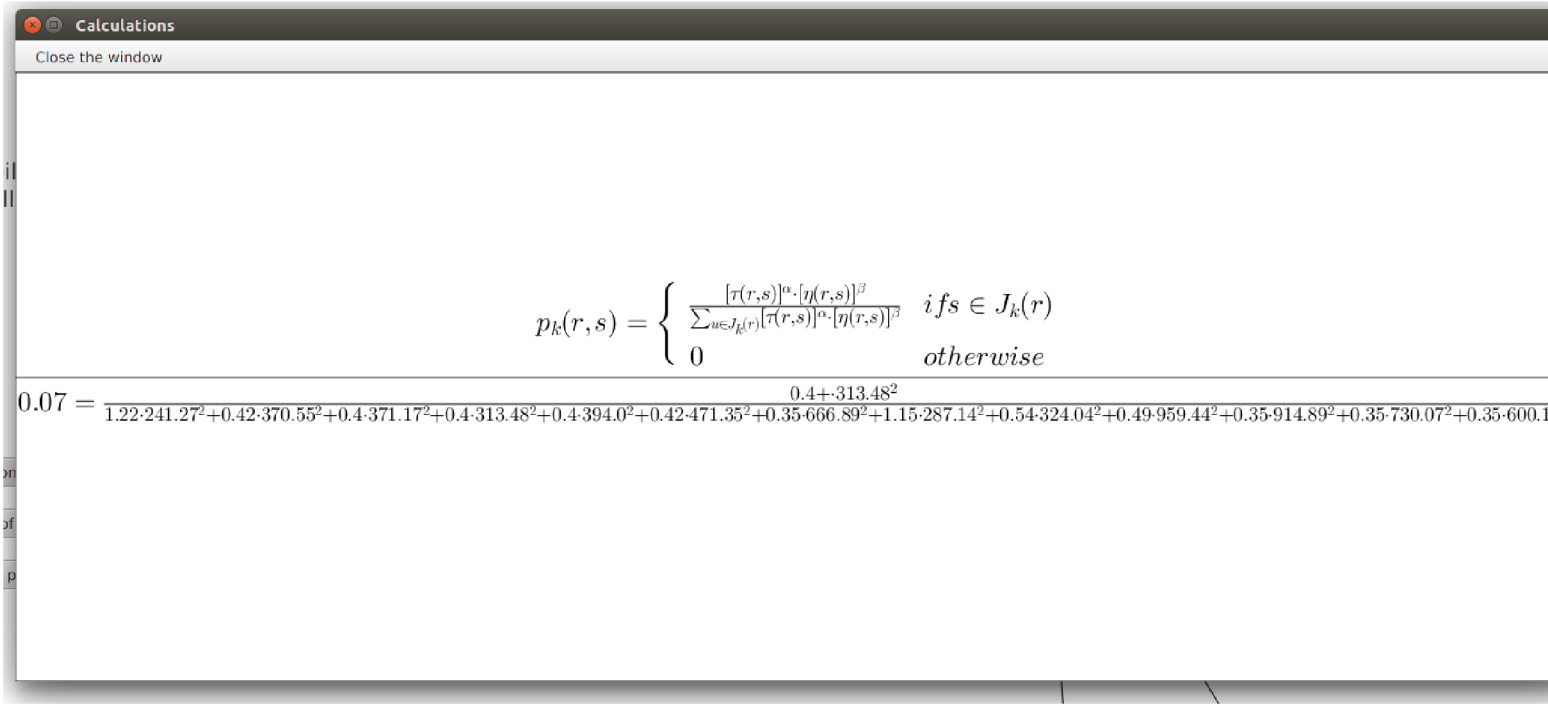
X	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
A	X	0.3874...	0.3874...	0.3874...	0.3874...	1.2226...	0.3874...	0.3874...	0.3874...	1.2395...	0.9370...	0.3874...	0.3874...	0.3874...	0.3874...
B	0.3874...	X	1.1894...	0.4409...	0.3874...	0.4620...	0.3874...	0.3874...	0.6466...	0.3874...	1.3076...	1.0562...	0.3874...	0.4519...	0.5464...
C	0.3874...	1.1894...	X	1.2657...	0.3874...	0.4409...	0.3874...	0.3874...	0.3874...	0.3874...	0.3874...	0.3486...	0.3874...	0.3874...	1.2513...
D	0.3874...	0.4409...	1.2657...	X	0.4580...	0.4426...	0.4680...	1.3366...	0.3874...	0.4760...	0.3874...	0.3874...	0.3874...	0.3874...	0.3874...
E	0.3874...	0.3874...	0.3874...	0.4580...	X	1.2093...	1.2134...	1.2916...	0.3486...	0.3874...	0.5389...	0.3874...	0.3874...	0.3874...	0.3874...
F	1.2226...	0.4620...	0.4409...	0.4426...	1.2093...	X	0.4436...	0.4679...	0.3874...	1.2745...	0.5440...	0.5445...	0.3874...	0.3874...	0.3874...
G	0.3874...	0.3874...	0.3874...	0.4680...	1.2134...	0.4436...	X	0.8454...	0.7250...	1.2089...	1.1443...	0.4304...	0.5738...	0.3874...	0.3874...
H	0.3874...	0.3874...	0.3874...	1.3366...	1.2916...	0.4679...	0.8454...	X	1.0453...	0.3874...	0.4580...	0.3874...	0.4989...	0.3874...	0.3874...
I	0.3874...	0.6466...	0.3874...	0.3874...	0.3486...	0.3874...	0.7250...	1.0453...	X	0.4211...	0.3742...	0.8812...	0.6497...	0.6085...	1.0532...
J	1.2395...	0.3874...	0.3874...	0.4760...	0.3874...	1.2745...	1.2089...	0.3874...	0.4211...	X	0.4519...	0.3486...	0.3874...	0.3874...	0.4679...
K	0.9370...	1.3076...	0.3874...	0.3874...	0.5389...	0.5440...	1.1443...	0.4580...	0.3742...	0.4519...	X	0.4304...	0.4745...	0.3874...	0.8355...
L	0.3874...	1.0562...	0.3486...	0.3874...	0.3874...	0.5445...	0.4304...	0.3874...	0.8812...	0.3486...	0.4304...	X	1.1655...	0.4745...	0.3874...
M	0.3874...	0.3874...	0.3874...	0.3874...	0.3874...	0.3874...	0.5738...	0.4989...	0.6497...	0.3874...	0.4745...	1.1655...	X	1.2046...	0.5352...
N	0.3874...	0.4519...	0.3874...	0.3874...	0.3874...	0.3874...	0.3874...	0.3874...	0.6085...	0.3874...	0.3874...	0.4745...	1.2046...	X	1.1729...
O	0.3874...	0.5464...	1.2513...	0.3874...	0.3874...	0.3874...	0.3874...	0.3874...	1.0532...	0.4679...	0.8355...	0.3874...	0.5352...	1.1729...	X

Obrázek A.22: Tabulka feromonů ukazuje hodnoty feromonů na hranách mezi všemi městy.



X	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
A	X	548.49...	714.90...	655.88...	370.68...	646.23...	467.68...	282.52...	370.21...	520.38...	914.13...	632.58...	809.98...	580.02...	787.36...
B	548.49...	X	730.0	846.41...	353.48...	358.43...	717.00...	669.60...	459.02...	522.26...	1285.2...	1085.0...	1130.8...	947.97...	1003.6...
C	714.90...	730.0	X	241.36...	432.94...	419.95...	319.95...	503.88...	364.33...	225.48...	733.38...	719.28...	554.66...	511.94...	352.67...
D	655.88...	846.41...	241.36...	X	500.07...	603.00...	188.27...	387.38...	399.10...	335.05...	499.88...	488.40...	322.41...	281.78...	157.49...
E	370.68...	353.48...	432.94...	500.07...	X	280.17...	363.67...	349.54...	105.53...	207.47...	933.15...	753.74...	777.37...	600.00...	657.41...
F	646.23...	358.43...	419.95...	603.00...	280.17...	X	543.94...	605.21...	334.69...	278.68...	1093.9...	977.31...	921.08...	794.62...	749.09...
G	467.68...	717.00...	319.95...	188.27...	363.67...	543.94...	X	203.21...	258.27...	268.56...	570.62...	439.23...	414.26...	250.83...	326.71...
H	282.52...	669.60...	503.88...	387.38...	349.54...	605.21...	203.21...	X	270.63...	373.87...	651.97...	415.58...	532.33...	308.60...	507.98...
I	370.21...	459.02...	364.33...	399.10...	105.53...	334.69...	258.27...	270.63...	X	151.33...	828.27...	659.33...	671.83...	497.62...	555.80...
J	520.38...	522.26...	225.48...	335.05...	207.47...	278.68...	268.56...	373.87...	151.33...	X	816.75...	706.82...	646.02...	517.82...	488.98...
K	914.13...	1285.2...	733.38...	499.88...	933.15...	1093.9...	570.62...	651.97...	828.27...	816.75...	X	326.07...	178.73...	344.68...	390.84...
L	632.58...	1085.0...	719.28...	488.40...	753.74...	977.31...	439.23...	415.58...	659.33...	706.82...	326.07...	X	326.74...	207.61...	480.35...
M	809.98...	1130.8...	554.66...	322.41...	777.37...	921.08...	414.26...	532.33...	671.83...	646.02...	178.73...	326.74...	X	232.76...	216.40...
N	580.02...	947.97...	511.94...	281.78...	600.00...	794.62...	250.83...	308.60...	497.62...	517.82...	344.68...	207.61...	232.76...	X	300.32...
O	787.36...	1003.6...	352.67...	157.49...	657.41...	749.09...	326.71...	507.98...	555.80...	488.98...	390.84...	480.35...	216.40...	300.32...	X

Obrázek A.23: Tabulka vzdáleností ukazuje vzdálenosti mezi všemi městy.



Obrázek A.24: Pokud si krok algoritmu vyžaduje nějaký výpočet, je popsán v tomhle okně, Je zde vždy přítomný vzorec výpočtu a většinou výpočet samotný.

A.3 Obsah CD

- Technická zpráva(Technicka_zprava.pdf)
- Zdrojové soubory(source_files.zip)
- README