UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

DIPLOMOVÁ PRÁCE

2020 **Bc. Arsen Khalafian**

UNIVERZITA PARDUBICE

Fakulta elektrotechniky a informatiky

Srovnání různých přístupů řešení dopravního problému

Bc. Arsen Khalafian

Diplomová práce

2020

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2019/2020

ZADÁNÍ DIPLOMOVÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Arsen Khalafian**
Osobní číslo:	**I18331**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Téma práce:	**Srovnání různých přístupů řešení dopravního problému**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování

Diplomová práce by měla obsahovat srovnání jednotlivých metod, vhodných pro řešení vybraného dopravního problému – například problém obchodního cestujícího (okružní) nebo jiný dle výběru diplomanta. Dle rozsahu lze vybrat i dvě různé úlohy a srovnat je mezi sebou. Diplomová práce by měla obsahovat jak srovnání klasických metod při řešení vybraného problému (například pro problém obchodního cestujícího to jsou metoda výhodnostních čísel, metoda Habrových frekvencí, spojovací metoda, metoda ztrát a další), tak i genetický přístup a implementovaný genetický algoritmus. Práce by měla obsahovat i metaheureistické metody, hlavně metodu Tabu search. Výstupem by měla být aplikace, umožňující vyřešení konkrétního dopravního problému optimální metodou z diplomantem implementovaných metod a jejich srovnání. Práce by měla být doplněna i o grafické výstupy, vzniklé při optimalizaci řešených úloh.

Rozsah pracovní zprávy: 50 – 70 stran
Rozsah grafických prací:
Forma zpracování diplomové práce: **tištěná**

Seznam doporučené literatury:

DOSTÁL, Petr. Optimalizační metody. Kunovice: Evropský polytechnický institut, 2008, 44 s. ISBN 978-80-7314-136-3.
KUČERA, Petr. Metodologie řešení okružního dopravního problému. Praha: ČZU, 2009, 122s.
MATOUŠEK, Jiří a Jaroslav NEŠETŘIL. Kapitoly z diskrétní matematiky. 4., upr. a dopl. vyd. V Praze: Karolinum, 2009, 442 s. ISBN 978-80-246-1740-4.

Vedoucí diplomové práce: **Mgr. Alena Pozdílková, Ph.D.**
Katedra matematiky a fyziky

Datum zadání diplomové práce: **5. listopadu 2019**
Termín odevzdání diplomové práce: **15. května 2020**

Univerzita Pardubice Česká republika

Ing. Zdeněk Němec, Ph.D.
děkan

prof. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 15. listopadu 2019

PROHLÁŠENÍ

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

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

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

V Pardubicích dne 21. 5. 2020

_____

Bc. Arsen Khalafian

PODĚKOVÁNÍ

Chtěl bych poděkovat svým rodičům za jejich podporu a pomoc po celou dobu mého studia. Zvláštní poděkování bych chtěl vyjádřit vedoucí práce paní Mgr. Ph.D. Aleně Pozdílkové a panu Mgr. Ph.D. Jaroslavu Markovi za odborné vedení práce a cenné rady, kteří pomohli tuto práci zkompletovat. Také bych chtěl poděkovat celému pedagogickému týmu Fakulty informačních technologií Univerzity Pardubice za úžasnou profesionalitu a seznámení s nejnovějšími a současnými technologiemi ve světě IT.

ANOTACE

Tato práce je zaměřena na srovnání různých způsobů řešení vybraného dopravního problému, známého jako problém obchodního cestujícího. V prvních kapitolách práce je představen teoretický základ této problematiky a jsou uvedeny základní pojmy teorie grafů, které jsou potřebné k vyřešení problému obchodního cestujícího. Třetí kapitola práce je věnována podrobnému popisu metod, používaných k řešení tohoto problému. Jsou zde popsány algoritmy klasických, evolučních i metaheuristických metod. Ve čtvrté části je uvedena srovnávací analýza implementovaných algoritmů, včetně porovnání jejich výsledků na reálných geodatech. V poslední části práce je popsána architektura vytvořené aplikace, včetně uživatelské dokumentace a vizualizace získaných výsledků na reálné mapě.

KLÍČOVÁ SLOVA

Problém obchodního cestujícího, teorie grafů, evoluční metody, metaheuristické metody, optimalizace, cesta.

TITLE

Comparison of different techniques to solving the transport problem.

ANNOTATION

This work is focused on the comparison of different techniques to solving a selected transport problem, known as the traveling salesman problem. The first chapters of the work introduce the theoretical basis of this issue and present the basic concepts of graph theory, which are needed to solve the travelling salesman problem. The third chapter is devoted to a detailed description of the methods used to solve this problem. Algorithms of classical, evolutionary and metaheuristic methods are described here. The fourth part presents a comparative analysis of implemented algorithms, including a comparison of their results on real geodata. The last part of the work describes the architecture of the created application, including user documentation and visualization of the obtained results on a real map.

KEYWORDS

Travelling Salesman Problem, graph theory, evolutionary methods, metaheuristic methods, optimization, route.

OBSAH

SEZNAM ZKRATEK

AC	Ant colony
B&B	Branch and bound
DP	Dynamic programming
GA	Genetický algoritmus
HTTP	HyperText Transfer Protocol
KML	Keyhole Markup Languag
MVC	Model-View-Controller
NP	Non-deterministic polynomial
SA	Simulated annealing
TS	Tabu search
TSP	Travelling salesman problem
UML	Unified Modeling Language
XML	eXtensible Markup Language

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM KÓDŮ

ÚVOD

Známý problém obchodního cestujícího byl formulován ve 30. letech 20. století a je stále jednou z nejpopulárnějších a nejdůležitějších úloh kombinatoriky. Důvod spočívá v jejím praktickém významu. Úlohou je najít nejvýhodnější cestu procházející všemi zadanými městy právě jednou s návratem do původního města. Optimalizace této úlohy patří mezi NP-těžké problémy. Tato úloha také patří k transvýpočetním úlohám: již při relativně malém počtu měst (více než 65) bude výpočet řešení metodou jednotného vyhledávání trvat několik miliard let.

Mnoho z existujících problémů lze formulovat jako problém obchodního cestujícího: problém s dopravou, vrtání desek s plošnými spoji, rentgenová krystalografie, počítačová síť atd. Během posledních 50 let fyzici stále více využívají problém obchodního cestujícího (TSP), zejména stochastické verze problému, kde jsou případy náhodně vybírány z určitého množství problémů. Motivací bylo vždy najít vlastnosti, které lze aplikovat na velkou třídu neuspořádaných systémů, a to buď prostřednictvím dobrých přibližných metod, nebo pomocí přesných analytických způsobů. Na druhou stranu, všechny NP-těžké problémy mohou být transformovány jeden do druhého v polynomickém čase, takže řešení TSP je velmi teoreticky zajímavé.

Cílem diplomové práce je srovnání různých metod pro řešení problému obchodního cestujícího, vytvoření aplikace schopné vizualizovat řešení a také vybrat nejvhodnější cestu.

1. PROBLÉM OBCHODNÍHO CESTUJÍCÍHO

Problém obchodního cestujícího (nebo TSP z angl. Travelling salesman problem) je jedna z nejznámějších úloh kombinatorické optimalizace, jejímž cílem je nalezení nejziskovější cesty procházející uvedenými městy alespoň jednou a poté návrat do původního města. V podmínkách úlohy se uvádí kritérium ziskovosti cesty (nejkratší, nejlevnější, souhrnné kritérium atd.) a odpovídající matici vzdáleností, nákladů a podobně. Zpravidla se uvádí, že cesta musí projít každým městem pouze jednou - v tomto případě se vybírá mezi hamiltonovskými cykly. Existuje několik zvláštních případů obecné formulace problému, zejména geometrický problém obchodního cestujícího (nazývaný také planární nebo euklidovský, kdy vzdálenostní matice odráží vzdálenosti mezi body v rovině), metrický problém obchodního cestujícího (kdy v je nákladové matici splněna trojúhelníková nerovnost), symetrický a asymetrický problém obchodního cestujícího. Existuje také zobecnění problému, tzv. zobecněný problém obchodního cestujícího. V této kapitole jsem vycházel z [BRENDON, 1993], [COOK, 2012], [KUČERA, 2009].

Optimalizační formulace problému patří do třídy NP-těžkých problémů, stejně jako většina jeho zvláštních případů. Verze „decision problem“ (tj. verze, ve které je položena otázka, zda existuje cesta, která není delší, než stanovená hodnota k) patří do třídy NP-úplných problémů. Problém obchodního cestujícího patří do transvýpočetních úloh: již při relativně malém počtu měst (66 a více) ji nelze vyřešit vyčíslením možností žádnými teoreticky myslitelnými počítači po dobu kratší než několik miliard let.

1.1 Grafické znázornění

Aby bylo možné problém vyřešit pomocí matematického nástroje, měl by být prezentován ve formě matematického modelu. Problém obchodního cestujícího je možné znázornit jako grafický model, tj. pomocí vrcholů a hran mezi nimi. Vrcholy grafu tedy odpovídají městům a hrany (i, j) mezi vrcholy i a j jsou komunikační cesty mezi těmito městy. Každá hrana (i, j) může být spojena s kritériem ziskovosti cesty $c_{ij} \geq 0$, což lze chápat např. jako vzdálenost mezi městy, čas nebo náklady na cestu.

Hamiltonovský cyklus je cesta, která obsahuje každý vrchol grafu právě jednou.

Za účelem zjednodušení problému a zajištění existence cesty se obvykle předpokládá, že graf, modelující daný problém, je souvislý, to znamená, že mezi libovolným párem vrcholů existuje hrana.

V případech, kdy takováto hrana neexistuje, toho lze dosáhnout zadáním hran s maximální délkou. Vzhledem k velké délce se taková hrana nikdy nestane součástí optimální cesty, pokud taková existuje.

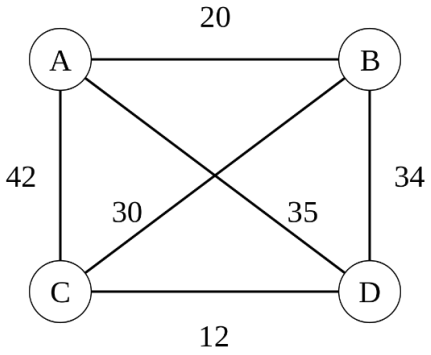
To znamená, že řešením problému obchodního cestujícího je problém nalezení nejkratší hamiltonovské kružnice v úplném neorientovaném ohodnoceném grafu. Existují různé verze problému obchodního cestujícího, z nichž nejběžnějšími jsou symetrické a metrické problémy.

1.2 Asymetrické a symetrické problémy

Obecně platí, že asymetrický problém obchodního cestujícího se od symetrického liší v tom, že je reprezentován pomocí orientovaného grafu. To znamená, že je třeba vzít v úvahu orientaci jednotlivých hran.

V případě symetrického problému pro všechny dvojice vrcholů platí, že hrany mezi nimi mají stejnou délku v obou orientacích, tj. pro hranu (i, j) platí, že $c_{ij} = c_{ji}$.

V symetrickém případě je počet možných cest poloviční než u asymetrického případu. Symetrický problém je modelován pomocí neorientovaného grafu.



Obrázek 1: Symetrický problém pro čtyři města

Ve skutečnosti může být problém obchodního cestujícího v případě skutečných měst buď symetrický, nebo asymetrický, v závislosti na konkrétní modifikaci (směr pohybu atd.).

1.3 Metrický problém

Symetrický problém obchodního cestujícího se nazývá *metrický*, pokud délky hran splňují trojúhelníkovou nerovnost. Jinak řečeno, u takových problémů jsou obchvatové cesty delší než přímé, to znamená, že hrana od vrcholu i do vrcholu j nikdy není delší než cesta přes mezilehlý vrchol $k : c_{ij} \leq c_{ik} + c_{kj}$.

Tato vlastnost délek hran určuje měřitelný prostor na množině hran a míru vzdálenosti, odpovídající intuitivnímu porozumění vzdálenosti.

Běžné funkce, reprezentující vzdálenosti v praxi jsou také metriky, splňující trojúhelníkovou nerovnost:

- Euklidovská metrika v euklidovském problému obchodního cestujícího,
- Manhattanská (kvartální) metrika obdélníkového problému obchodního cestujícího, ve kterém vzdálenost mezi vrcholy mřížky se rovná součtu vzdáleností na svislé a vodorovné ose (na ose x a y),
- Maximální metrika, která určuje vzdálenost mezi vrcholy jako maximální hodnotu vzdálenosti na svislé a vodorovné ose (ose x a y).

Poslední dvě metriky se používají například při vrtání otvorů v deskách plošných spojů, kdy stroj musí vyvrtat více otvorů v co nejkratším čase a může pohybovat vrtákem v obou směrech a přemísťovat ho od jednoho otvoru k dalšímu. Manhattanská metrika odpovídá případu, kdy k pohybu v obou směrech dochází postupně, a maximální matrika odpovídá případu, kdy k pohybu v obou směrech dochází synchronně, a celkový čas se rovná maximálnímu času pohybu podél svislé a vodorovné osy (osy x a y).

Nemetrický problém obchodního cestujícího může nastat například v případě minimalizace délky pobytu za podmínek možnosti volby dopravních prostředků v různých směrech. V tomto případě může být obchvatová cesta letadlem kratší než přímá cesta vozidlem.

1.4 Algoritmická složitost

Jelikož obchodní cestující v každém městě čelí volbě dalšího města z těch, která ještě nenavštívil, existuje $(n-1)!$ cest pro asymetrický problém a $(n-1)!/2$ cest pro symetrický problém obchodního cestujícího. Proto velikost prohledávaného prostoru přímo závisí na počtu měst.

Je také známo, že za podmínek $P \neq NP$ neexistuje žádný algoritmus, který by pro některý polynom p vypočítal taková řešení problému obchodního cestujícího, která by se lišila od optimálního maxima o součinitel $2^{p(n)}$.

V praxi není vždy nutné hledat zcela optimální cestu. Existují algoritmy pro nalezení přibližných řešení pro metrický problém v polynomickém čase, případně nalezení cesty maximálně dvakrát delší než je cesta optimální. Dosud není znám žádný algoritmus s polynomickým časem, který by zaručoval lepší přesnost než 1,5 optima.

Podle předpokladu $P \neq NP$ existuje (neznámá) konstanta $c > 0$, taková, že žádný algoritmus s polynomickým časem nemůže zaručit přesnost $1 + c$.

Ale data mohou mít vlastnosti, které mohou pomoci problém vyřešit. Například města nejsou umístěna náhodně, ale podle terénu nebo dokonce podél již dávno nalezené optimální obchodní cesty. Další informace a heuristiky umožňují najít přijatelná řešení v přijatelném čase.

2. TEORIE GRAFŮ

Teorie grafů je obor diskrétní matematiky, který se zabývá studiem vlastností grafů.

Teorie grafů se používá k analýze fungování složitých systémů, jako jsou silniční sítě, telefonní a počítačové sítě a závlahové systémy. Tato teorie je tradičně účinným nástrojem pro formalizaci úloh ekonomické a plánovací výrobní praxe, používá se v automatizaci řízení výroby, v kalendářním a síťovém plánování. V této kapitole jsem vycházel z [MATOUŠEK et al., 2009], [PLOTNIKOV, 1998], [BONDY, 2005].

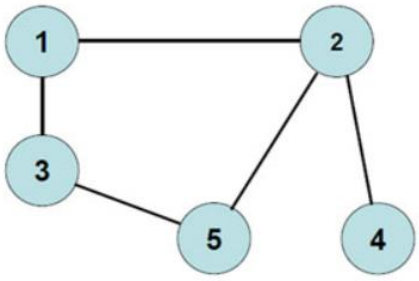
Pro reprezentaci grafů se často používá maticová forma. Existují různé typy matic grafů, ale všechny většinou zcela vyjadřují základní vlastnosti grafů. Maticová forma přiřazení grafu je dostatečně přesná pro jakýkoli stupeň složitosti grafu, a co je nejdůležitější, umožňuje automatizovat zpracování informací prezentovaných z hlediska teorie grafů - do počítače lze načíst jakoukoli grafovou matici.

V současné době se aktivně rozvíjí část teorie grafů týkající se řešení cest splňujících zvláštní omezení: Eulerovy a Hamiltonovy cykly; cesty vyhýbající se zakázaným hranám; samoneprotínající se a neprotínající se cesty; obousměrné dvojité obchvaty atd.

2.1 Základní pojmy

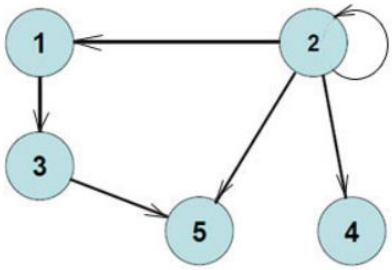
Graf je základní struktura, tvořena množinami vrcholů a hran.

Neorientovaný graf je uspořádaná dvojice $G := (V, E)$, kde V je neprázdná množina a E (může být i prázdná) je množina dvouprvkových podmnožin množiny V. Prvky množiny V nazýváme vrcholy a prvky množiny E nazýváme hrany grafu G.



Obrázek 2: Neorientovaný graf

Orientovaný graf je uspořádaná dvojice $G := (V, A)$, kde V je množina vrcholů a A je množina hran, přičemž hrana $a \in A$ je uspořádaná dvojice dvou navzájem různých vrcholů.



Obrázek 3: Orientovaný graf

Orientovaná hrana je uspořádaná dvojice vrcholů (v, w), kde vrchol v je začátek a w je konec této orientované hrany. Tedy orientovaná hrana $v \to w$ vede z vrcholu v k vrcholu w.

Vrcholy grafu, které nepatří k žádné hraně, se nazývají izolované vrcholy.

Graf, který se skládá pouze z izolovaných vrcholů, se nazývá nulový graf.

Graf, ve kterém je každá dvojice vrcholů spojena hranou, se nazývá úplný graf.

Stupeň vrcholu je počet hran, které z daného vrcholu vycházejí.

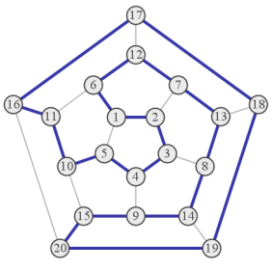
Graf, který lze v rovině znázornit tak, že se jeho hrany protínají pouze ve vrcholech, se nazývá rovinný graf.

Cyklus je cesta, ve které se počáteční a koncový bod shodují.

Jednoduchý cyklus je cyklus, který neprochází žádným z vrcholů grafu více než jednou.

Hamiltonovský cyklus je cyklus obsahující všechny vrcholy grafu právě jednou.

Hamiltonovský graf je graf, který obsahuje hamiltonovský cyklus.



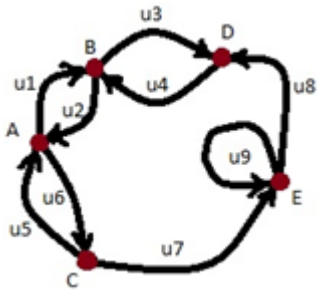
Obrázek 4: Hamiltonovský graf

Dva vrcholy A, B v grafu se nazývají sousední, pokud existuje cesta vedoucí z vrcholu A do B. Strom je souvislý graf, který neobsahuje cykly.

2.2 Způsoby zadávání grafů

2.2.1 Latinská matice

Graf lze zadat různými způsoby: graficky, vyjmenováním vrcholů a hran (orientovaných nebo neorientovaných) atd. Ve velké většině případů se graf zadává maticí. Pro počítačové výpočty je to nejčastěji používaný způsob. U této metody je směr orientovaných hran určen podle pořadí písmen v jejich názvu. To nám ilustruje například graf zobrazený na obr. 5. Příslušná matice má pro něj podobu uvedenou v tabulce 1.



Obrázek 5: Orientovaný graf

	A	B	C	D	E
A		AB	AC		
B	BA			BD	
C	CA				CE
D		DB			
E				ED	EE

Tabulka 1: Reprezentace orientovaného grafu maticí

Pokud je graf neorientovaný, pak se v matici odpovídající buňka v tabulce šrafuje.

2.2.2 Incidenční matice

Další maticí spojenou s grafem G, ve kterém jsou označeny vrcholy a hrany, je incidenční matice $B = (b_{ij})$. V této matici typu $p \times q$ platí, že $b_{ij} = 1$, pokud v_i a x_j jsou incidentní, a v jiných případech $b_{ij} = 0$ (jsou incidentní, pokud se nachází na stejné hraně). Stejně jako matice sousednosti určuje incidenční matice graf G až na izomorfismus.

Pro neorientovaný graf jsou prvky této matice definovány podle následujícího pravidla: $b_{ij} = 1$, pokud je vrchol v_i incidentní s hranou x_j, a roven nule, pokud v_i a x_j nejsou incidentní.

Pro neorientovaný graf se prvky této matice zadávají následujícím způsobem:

$$b_{ij} = \begin{cases} 1, & pokud\ \ i-ty\ vrchol\ je\ incidentni\ s\ \ j-tou\ hranou \\ 0, & jinak \end{cases}$$

Pro orientovaný graf se prvky matice zadávají následujícím způsobem:

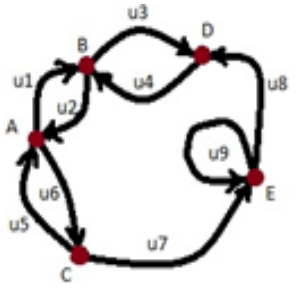
$$b_{ij} = \begin{cases} 1, & pokud\ \ i-ty\ vrchol\ je\ pocatecnim\ vrcholem\ j-te\ orientovane\ hrany \\ -1, & pokud\ \ i-ty\ vrchol\ je\ poslednim\ vrcholem\ j-te\ orientovane\ hrany \\ 0, & jinak \end{cases}$$

Řádky incidenční matice se nazývají incidenční vektory grafu G. Incidenční matice také jednoznačně určuje strukturu grafu. Pro incidenční matici platí věta podobná pro matici sousednosti.

Věta: Grafy jsou izomorfní právě tehdy, pokud jsou jejich incidenční matice získány jedna z druhé libovolnými permutacemi řádků a sloupců.

Důkaz: Prozkoumáme dva grafy G_1 a G_2, které se od sebe liší pouze číslováním vrcholů. To znamená, že v těchto grafech existuje permutace **s** na množině vrcholů, které zachovává jejich incidenci.

Této permutaci (dva řádky a dva sloupce se stejnými čísly jsou přemístěny současně jako jediná operace) skutečně odpovídá přečíslování vrcholů grafu, což zjevně vede k izomorfnímu grafu.



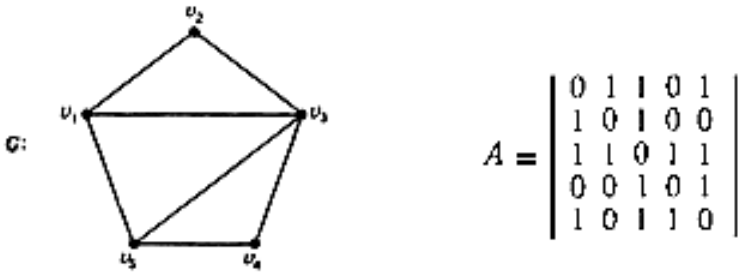
$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{cccccccc} u_1 & u_2 & u_3 & u_4 & u_5 & u_6 & u_7 & u_8 \end{array} \\ \begin{pmatrix} -1 & 1 & 0 & 0 & 1 & -1 & 0 & 0 \\ 1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{pmatrix} \end{array}$$

Obrázek 6: Příklad grafu a incidenční matice

2.2.3 Matice sousednosti

Maticí sousednosti $A = \|a_{ij}\|$ grafu G s vrcholy p se nazývá matice typu $p \times p$, ve které $a_{ij} = 1$, pokud vrchol u_i sousedí s u_j, a v ostatních případech $a_{ij} = 0$. Řádky a sloupce této matice odpovídají vrcholům grafu a její prvek na pozici (i, j) se rovná počtu násobných hran spojujících vrcholy v_i a v_j (nebo směrovaných z vrcholu v_i do vrcholu v_j pro orientovaný graf).

Existuje tedy vzájemná jednoznačné zobrazení mezi grafy s vrcholy p a symetrickými binárními maticemi typu $p \times p$ s nulami na úhlopříčce. Na obrázku 7 je uveden graf G a jeho matice sousednosti A. Je snadno vidět, že součty prvků matice A v řádcích jsou rovny stupňům vrcholů G.



Obrázek 7: Příklad grafu a matice sousednosti

Matice sousednosti neorientovaného grafu je vždy symetrická a matice orientovaného grafu je obecně asymetrická. Neorientovaným hranám odpovídají dvojice nenulových prvků symetrických hlavní úhlopříčce matice, orientovaným hranám odpovídají nenulové prvky matice a smyčkám odpovídají nenulové prvky hlavní úhlopříčky. Ve sloupcích a řádcích odpovídajícím izolovaným vrcholům jsou všechny prvky rovny nule. Prvky matice jednoduchého grafu jsou rovny 0 nebo 1 a všechny prvky hlavní úhlopříčky jsou nulové (jednoduchý graf je graf, který obsahuje prosté hrany bez smyček).

Prvky matice jsou definovány následujícím způsobem:

Pro neorientovaný graf

$$a_{ij} = \begin{cases} 1, & pokud\ \ i-ty\ a\ \ j-ty\ vrcholy\ jsou\ sousedni \\ 0, & jinak \end{cases}$$

Pro orientovaný graf:

$$a_{ij} = \begin{cases} 1, & pokud\ z\ \ i-teho\ vrcholu\ do\ \ j-teho\ \ vrcholu\ vede\ orientovana\ hrana \\ 0, & jinak \end{cases}$$

Věta: Grafy jsou izomorfní právě tehdy, když jsou jejich matice sousednosti získány jedna od druhé párovými permutacemi stejných řádků a sloupců.

V mé aplikaci je matice uložena ve formě matice sousednosti. Je to proto, že algoritmy často potřebují zkontrolovat existenci hrany mezi dvěma vrcholy. V matici sousednosti je algoritmicky složité získat délky mezi dvěma vrcholy $O(1)$, což příznivě ovlivňuje rychlost algoritmů.

3. PŘEHLED STÁVAJÍCÍCH METOD PRO ŘEŠENÍ PROBLÉMU OBCHODNÍHO CESTUJÍCÍHO

Problémy obchodního cestujícího jsou řešeny použitím různých přístupů a metod odvozených z teoretických výpočtů a výzkumu. Všechny účinné metody (výrazně omezující rozsáhlé vyhledávání) patří do skupiny heuristických metod. Výsledkem většiny heuristických metod není nejziskovější cesta, ale pouze její hrubý odhad. Často jsou zapotřebí takzvané any-time algoritmy, které postupně zlepšují některé aktuální přibližné řešení. V této kapitole jsem vycházel z [DOSTÁL, 2003], [APPLEGATE, 2011], [KOLESNIKOV et al., 2011].

Rozlišují se následující skupiny metod pro řešení problémů obchodního cestujícího:

Klasické (přesné) metody se vyznačují vysokou časovou složitostí - vyhledávací prostor roste exponenciálně, proto, když počet měst není malý, používají se jiné diskrétní optimalizační metody.

Metaheuristické metody umožňují najít nejvýhodnější řešení různých optimalizačních problémů v přijatelném čase. Metaheuristika je účinná a velmi oblíbená skupina optimalizačních metod, která umožňuje najít řešení pro širokou škálu úloh z různých aplikací. Síla metaheuristik spočívá v jejich schopnosti řešit složité úlohy bez znalosti vyhledávacího prostoru, a proto tyto metody umožňují řešit nevyřešitelné optimalizační úlohy.

Evoluční (genetické) metody - stochastické vyhledávací metody, které se úspěšně používají v mnoha reálných a složitých aplikacích (epistatické, multimodální, víceúčelové a velmi omezené úlohy). Úspěch těchto algoritmů při řešení složitých optimalizačních problémů přispěl k výzkumu v oblasti známé jako evoluční výpočty.

3.1 Klasické metody

Klasické metody umožňují garantovat optimálnost nalezeného řešení. Klasické metody se vyznačují vysokou složitostí, která často neumožňuje jejich použití při řešení skutečných úloh. Tato skupina zahrnuje různé verze metody Branch and Bound a algoritmus dynamického programování.

3.1.1 Algoritmus Branch and Bounds

Algoritmus Branch and Bounds byl poprvé navržen v roce 1960 Landem a Doigem v jejich práci věnované problému celočíselného lineárního programování. Tato práce však neměla znatelný přímý dopad na vývoj diskrétního programování.

Ve skutečnosti je „znovuzrození“ této metody spojeno s prací autorů Little, Murthy, Sweeney a Karel, která byla zaměřena na problém obchodního cestujícího. V této práci byl poprvé navržen obecně přijímaný název metody - „Branch and Bound“ – česky Metoda větví a hranic. Od této chvíle se objevuje velmi velké množství prací věnovaných metodě větví a hranic a jejich různým úpravám. Takový velký úspěch (a to i s ohledem na problém klasického obchodního cestujícího) je vysvětlen tím, že Little, Murthy, Sweeney a Karel byli první, kdo věnovali pozornost šíři možností této metody, zaznamenali důležitost využití specifik problému a sami tuto specifiku velmi úspěšně využili. V této kapitole jsem čerpal z [BRUSCO et al., 2005] a [STRUCHENKOV, 2016].

Popis algoritmu

Nechť je $x = (i_1, i_2 ... i_n, i_1)$ libovolná přípustná cesta obchodního cestujícího, kde čísla $i_1, i_2 ... i_n, i_1$ označují pořadí měst na této cestě, kde $(i_1, i_2), (i_2, i_3), ..., (i_n, i_1)$ jsou hrany cesty. D je množina všech přípustných cest obchodního cestujícího, a $l(x)$ je délka cesty x, která se rovná $c_{i_1 i_2} + c_{i_2 i_3} + ... + c_{i_n i_1}$.

1. Výpočet odhadu pro množinu D

Předpokládejme, že $\alpha_i = \min_j c_{ij}$. Pak $c'_{ij} = c_{ij} - \alpha_i \geq 0$ pro jakékoli j a

$$l(x) = \sum_{i=1}^{n} \alpha_i + (c'_{i_1 i_2} + c'_{i_2 i_3} + ... + c'_{i_{n-1} i_n} + c'_{i_n i_1}).$$

Předpokládejme, že $\beta_j = \min_i c'_{ij}$. Pak $c''_{ij} = c'_{ij} - \beta_j \geq 0$ pro jakékoli i a

$$l(x) = \sum_{i=1}^{n} \alpha_i + \sum_{j=1}^{n} \beta_i + (c''_{i_1 i_2} + c''_{i_2 i_3} + ... + c''_{i_{n-1} i_n} + c''_{i_n i_1}). \quad (1)$$

Matice $C'' = (c''_{ij})_{n \times n}$ se nazývá *redukovaná*, operace jejího vytvoření se nazývá *redukce* matice C, a hodnoty α_i, β_j se nazývají *redukční konstanty.*

Všimneme si, že v každém řádku a v každém sloupci matice C'' je alespoň jeden nulový prvek. Předpokládejme, že

$$\gamma(D) = \sum_{i=1}^{n} \alpha_i + \sum_{j=1}^{n} \beta_i .$$

Z (1) vyplývá, že pro každou přípustnou cestu platí $l(x) \geq \gamma(D)$. Je zřejmé, že po redukci jsou délky všech cest sníženy o stejnou hodnotu $\gamma(D)$. Proto je optimální cesta nalezená pomocí redukované matice optimální i pro původní úlohu. Kromě toho pro délky optimálních cest úlohy s maticí C a úlohy s maticí C'' platí následující rovnost:

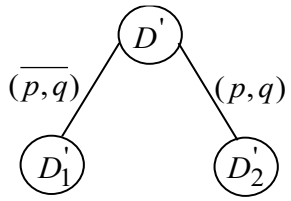
$$l_C^* = l_{C''}^* + \gamma(D). \qquad (2)$$

2. Výběr množiny pro větvení

V prvním kroku nechť je touto množinou množina D. Její minimální hodnota je $\varphi(D) = \gamma(D)$. Ve zbývajících krocích se z množiny kandidátů na větvení (z množiny visících vrcholů stromu větvení) vybere množina D' s nejnižší hodnotou.

3. Rozdělení množiny D' na podmnožiny (větvení)

Písmenem E označíme redukovanou maticí odpovídající množině D'**.** Způsobem popsaným níže v odst. 5 je vybrána hrana (p,q). Množina D' je rozdělena na dvě podmnožiny D_1' a D_2' a je vyloučena z počtu kandidátů na větvení. Podmnožina D_1' se skládá ze všech cest množiny D', které neobsahují hranu (p,q); podmnožina D_2' se skládá z cest, které obsahují hranu (p,q) (obr. 8).



Obrázek 8: Rozdělení množiny D' na podmnožiny (větvení)

4. Přeměna matice nákladů a výpočet odhadů

Matice E_1 odpovídající množině D_1' je získána z E záměnou $e_{pq} = \infty$. Pro množinu D_2' je odpovídající matice E_2 získána z E odstraněním p - tého řádku a q - tého sloupce. Kromě toho je zpravidla vyžadován zákaz některého prvku za účelem odstranění z D_2' uzavřené cesty procházející hrany již zahrnutými do cesty, ale nikoli úplné (tj. neobsahující všechny n měst).

Odhady pro nově vytvořené množiny se vypočítají následovně:

$$\varphi(D_1^{'}) = \varphi(D') + \gamma(D_1^{'}),$$
$$\varphi(D_2^{'}) = \varphi(D') + \gamma(D_2^{'}),$$

kde $\gamma(D_1^{'})$ a $\gamma(D_2^{'})$ jsou součty konstant redukce matic E_1 a E_2.

Odsud lze odvodit, že

$$\gamma(D_1^{'}) = \min_{j: j \neq q} e_{pj} + \min_{i: i \neq p} e_{iq}. \qquad (3)$$

5. Výběr hrany pro větvení

Ať E je redukovaná matice odpovídající množině D'.

Budeme analyzovat pouze hrany (k,l), pro které platí, že

$$e_{kl} = 0. \quad (4)$$

Dále mezi hrany splňujícími (4) vybereme hranu (p,q), pro který bude hodnota $\gamma(D_1^{'})$ maximální. Za tímto účelem vypočítáme pomocí (3) funkci

$$\Delta(k,l) = \min_{j: j \neq l} e_{kj} + \min_{i: i \neq k} e_{il}$$

pro každou hranu (k,l), aby $e_{kl} = 0$. Jako hranu (p,q) si vybereme ten, pro který

$$\Delta(p,q) = \max_{(k,l): e_{kl} = 0} \Delta(k,l).$$

3.1.2 Algoritmus Dynamic programming

Problém obchodního cestujícího lze v zásadě optimálně řešit způsobem úplného probírání možných variant s výpočtem hodnoty optimalizovaného kritéria pro každou z těchto variant. Počet variant je omezený, ale jejich počet je zpravidla i při relativně malé dimenzi problémů, které mají být vyřešeny, tak velký, že není možné provést úplný výpočet. Použití schématu dynamického programování umožňuje určitým způsobem uspořádat vyhledávání, čímž se významně sníží počet zvažovaných variant. V této kapitole jsem čerpal z [SESEKIN, 2005].

Algoritmus Dynamic programming (DP) řeší problémy, které splňují princip sekvenční optimalizace: řešení původního optimalizačního problému velké dimenze je nahrazeno řešením sekvence optimalizačních problémů malé dimenze.

Z tohoto důvodu je hlavní podmínkou použitelnosti metody DP možnost rozdělení postupu rozhodování do řady podobných kroků nebo fází, z nichž každá je plánována samostatně, ale s přihlédnutím k výsledkům dosaženým v jiných krocích.

Při prezentaci tohoto algoritmu se budeme držet terminologie odpovídající dané typické interpretaci problému.

Popis algoritmu

Nechť optimální (nejkratší) cesta z bodu v_1 je $[v_1, v_k, v_{k+1}, v_{k+2}, ..., v_{k+n}, v_1]$.

Subcesta $[v_k, v_{k+1}, v_{k+2}, ..., v_{k+n}, v_1]$ musí být nejkratší cestou od v_k do v_1, která právě jednou navštíví všechny vrcholy.

Podobně pro $[v_{k+1}, v_{k+2}, ..., v_{k+n}, v_1]$, $[v_{k+2}, ..., v_{k+n}, v_1]$ a tak dále až do $[v_{k+n}, v_1]$.

Problém je v tom, že nevíme, jaké permutace vrcholů $[v_1, v_k, v_{k+1}, v_{k+2}, ..., v_{k+n}, v_1]$ tvoří nejkratší cestu.

Proto je nutné projít všechny možnosti:

$[v_k, v_{k+1}, v_{k+2}, ..., v_{k+n-2}, v_{k+n-1}, v_{k+n}]$

$[v_k, v_{k+1}, v_{k+2}, ..., v_{k+n-1}, v_{k+n-2}, v_{k+n}]$

…

$[v_k, v_{k+2}, v_{k+1}, ..., v_{k+n-2}, v_{k+n-1}, v_{k+n}]$

$[v_{k+1}, v_k, v_{k+2}, ..., v_{k+n-2}, v_{k+n-1}, v_{k+n}]$

$[v_{k+2}, v_k, v_{k+1}, ..., v_{k+n-2}, v_{k+n-1}, v_{k+n}]$

…

$[v_k, v_{k+1}, v_{k+2}, ..., v_{k+n-2}, v_{k+n}, v_{k+n-1}]$

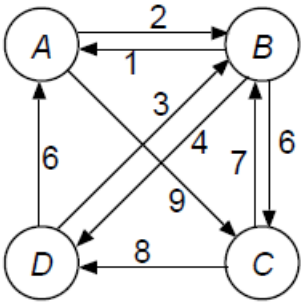
$[v_k, v_{k+1}, v_{k+2}, ..., v_{k+n}, v_{k+n-2}, v_{k+n-1}]$

…

A všechny ostatní permutace.

Příklad

Následující obrázek ukazuje příklad grafu a jemu odpovídající matici sousednosti, kde ∞ označuje chybějící hrany.



	a	b	c	d
a	0	2	9	∞
b	1	0	6	4
c	∞	7	0	8
d	6	3	∞	0

Obrázek 9: Příklad grafu a sousední matice pro DP

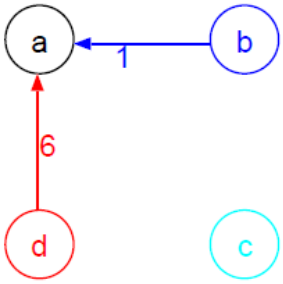
Nechť C(v, w) je délka hrany (v, w) a D(u, v) je cesta z (u) do (v), procházející několika vrcholy.

Krok 1: Vypočítáme všechny možné cesty, jak se dostat k bodu (a), skládající se ze dvou měst.

$$D[b,a] = C[b,a] = 1$$

$$D[c,a] = C[c,a] = \infty$$

$$D[d,a] = C[d,a] = 6$$



Obrázek 10: První krok algoritmu DP

Krok 2: Vypočítáme všechny možné cesty, jak se dostat k bodu (a), skládající se ze tří měst. V tomto případě budeme používat údaje získané v první fázi.

$$D[c,b,a] = C[c,b] + D[b,a] = 7 + 1 = 8$$

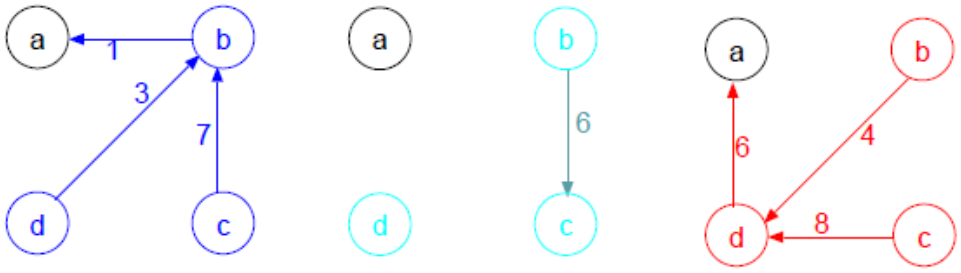
$$D[d,b,a] = C[d,b] + D[b,a] = 3 + 1 = 4$$

$$D[b,c,a] = C[b,c] + D[c,a] = 6 + \infty = \infty$$

$$D[d,c,a] = C[d,c] + D[c,a] = \infty + \infty = \infty$$

$$D[b,d,a] = C[b,d] + D[d,a] = 4 + 6 = 10$$

$$D[c,d,a] = C[c,d] + D[d,a] = 8 + 6 = 14$$



Obrázek 11: Druhý krok algoritmu DP

Krok 3: Vypočítáme všechny možné cesty, jak se dostat k bodu (a), skládající se ze čtyř měst. V tomto případě budeme používat údaje získané v druhé fázi.

$$D[d,c,b,a] = C[d,c] + D[c,b,a] = \infty + 8 = \infty$$

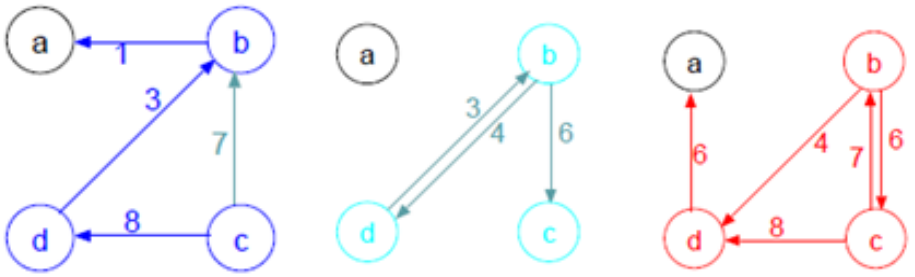
$$D[c,d,b,a] = C[c,d] + D[d,b,a] = 8 + 4 = 12$$

$$D[d,b,c,a] = C[d,b] + D[b,c,a] = 3 + \infty = \infty$$

$$D[b,d,c,a] = C[b,d] + D[d,c,a] = 4 + \infty = \infty$$

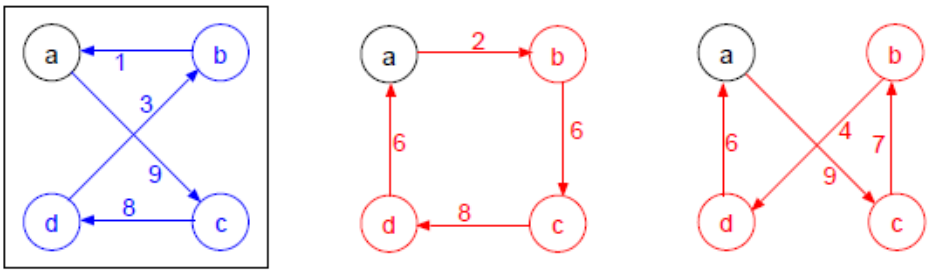
$$D[c,b,d,a] = C[c,b] + D[b,d,a] = 7 + 10 = 17$$

$$D[b,c,d,a] = C[b,c] + D[c,d,a] = 6 + 14 = 20$$



Obrázek 12: Třetí krok algoritmu DP

Krok 4: V poslední fázi je nutné spojit poslední bod s výchozím bodem a vybrat nejkratší cestu.



Obrázek 13: Čtvrtý krok algoritmu DP

$$
\begin{aligned}
D[a,\ldots,a] = MIN\ & (C[a,c] + D[c,d,b,a]),\\
& (C[a,b] + D[b,c,d,a]),\\
& (C[a,c] + D[c,b,d,a])\\
= MIN & (9+12,\ 2+20,\ 9+17)\\
= 21 &
\end{aligned}
$$

Dostáváme cestu:

$[a,[c,\ldots,a]] = [a,[c,[d,\ldots,a]]] = [a,[c,[d,[b,a]]]].$

Algoritmická složitost je stále exponenciální, ale zatím není polynomická.

3.2 Metaheuristické metody

Metaheuristiky jsou strategie, které řídí proces prohledávání k nalezení optimálních řešení.

Cílem metaheuristiky je účinné prozkoumání vyhledávacího prostoru a nalezení optimálních řešení. V této kapitole jsem čerpal z [SERGIENKO et al., 2003] a [CORMEN, 2009].

Metaheuristické algoritmy se pohybují od jednoduchých lokálních vyhledávacích postupů ke složitým procesům učení. Metaheuristické algoritmy jsou přibližné a zpravidla nedeterministické. Metaheuristické algoritmy mohou zahrnovat mechanismy zabraňující uvěznění v omezené oblasti vyhledávacího prostoru.

Metaheuristika může být popsána na abstraktní úrovni (tj. není určena k řešení specifických problémů).

Moderní metaheuristiky využívají zkušenosti s hledáním řešení pro správu vyhledávání uložené v paměti.

Každá metaheuristika má své vlastní chování a vlastnosti. Všechny metaheuristiky však mají řadu základních komponent a provádějí operace v rámci omezeného počtu kategorií.

Inicializace. Metoda pro nalezení počátečního řešení.

Okolí. Každému řešení x odpovídá jeho okolí (okolí budeme brát podle standardní definice. Podrobněji je popsáno v [BRENDON, 1993]) a k němu přidružené přechody: $\{N_1, N_2, ..., N_q\}$.

Kritérium pro výběr okolí je potřeba stanovit, pokud existuje více než jedno okolí. Toto kritérium musí uvádět nejen vybírané okolí, ale také podmínky pro jeho výběr. Alternativy mohou vypadat například: „při každé iteraci“ (např. genetické metody) nebo „za daných podmínek“.

Výběr kandidátů. Okolí může být velmi velké. Pak se obvykle zvažuje pouze podmnožina z množiny všech možných přechodů v každé iteraci. Odpovídající seznam kandidátů $C(x) \in N(x)$ může být konstantní a aktualizovaný z iterace na iteraci (například metoda Tabu search), nebo může být vybrán při každé nové iteraci (například genetické metody). Ve všech případech kritérium výběru určuje, jak lze vybrat kritérium pro zařazení do seznamu kandidátů.

Akceptační kritérium. Přechody se odhadují pomocí funkce $g(x, y)$ v závislosti na parametrech, jako je hodnota cílové funkce, pokuty za porušení určitých omezení atd. Nejlepší řešení je vybráno s ohledem na následující kritérium: $x = \arg opt\{g(x, y);\ y \in C(x)\}$ s ohledem na potřebu zabránit opakování.

Kritéria pro zastavení. Metaheuristika může být zastavena na základě různých kritérií: doba výpočtu, počet iterací, rychlost vylepšení řešení atd. Pro řízení různých fází vyhledávání lze definovat více než jedno kritérium.

3.2.1 Algoritmus Simulated annealing

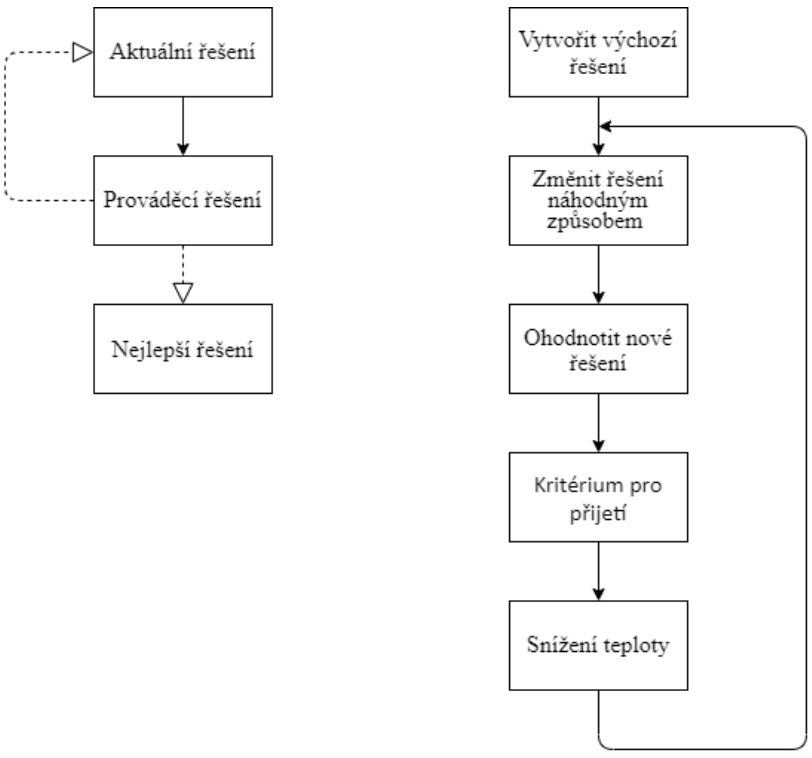
Algoritmus Simulated annealing (česky algoritmus simulovaného žíhání) je algoritmus pro řešení různých optimalizačních problémů. Je založen na modelování skutečného fyzikálního procesu, ke kterému dochází během krystalizace látky z kapaliny do pevného stavu, například během žíhání kovů.

Uvažujme systém popsaný n-rozměrným stavem $x = (x_1, x_2, ..., x_n)$, kde x_i patří do nějakého intervalu. Stavy tohoto systému jsou ohodnoceny funkcí f, která přiřadí každému bodu hodnotu $y = f(x)$. Cílem algoritmu je najít takový stav $x_{\min}$, ve kterém nabývá funkce f globálního minima.

Během činnosti algoritmu znázorněného na obrázku 14 je uloženo aktuální řešení, které je lokálním minimem. A po ukončení činnosti algoritmu dojde k nalezení globálního minima.

Hlavní kroky algoritmu:

1. Výběr výchozího řešení a výchozí teploty
2. Posouzení výchozího řešení
3. Náhodná změna aktuálního řešení
4. Posouzení změněného řešení
5. Kritérium pro přijetí
6. Snížení teploty a v případě, že teplota překročí určitou prahovou hodnotu, pak přechod k třetímu kroku



Obrázek 14: Hlavní kroky algoritmu SA

1. Výběr výchozího řešení a výchozí teploty

Jednou ze strategií je náhodný výběr výchozího řešení. Jako primární řešení lze také využít řešení získané jinými metodami.

2. Posouzení výchozího řešení

Tato fáze je zcela závislá na vlastnostech úlohy. Jediným požadavkem je odvození odhadu reálného čísla, které bude popisovat, do jaké míry je vypočítané řešení optimální. Toto číslo v algoritmu simulovaného žíhání se nazývá energie. Pokud je výběr takového čísla obtížný, pak možná je vhodné navrhovanou metodu odmítnout.

3. Náhodná změna aktuálního řešení

Tato fáze je velmi závislá na konkrétní úloze. Změny se provádí pouze lokálně. Například pro problém obchodního cestujícího se používá výměna dvou náhodných měst při jejich aktuálním sledování. V důsledku této změny dostaneme dvě řešení: aktuální a změněné.

4. Kritérium pro přijetí

Pro jistotu předpokládáme, že optimalizace spočívá v minimalizaci energie. Ve většině případů je tento přístup spravedlivý. Kritériem pro přijetí je kontrola a případně náhrada aktuálního řešení novým.

Pokud má změněné řešení méně energie, považuje se za aktuální. Pokud má změněné řešení větší energii, pak se bere s největší pravděpodobností

$$P = \exp(-\delta \mathrm{E}/\mathrm{T}),$$ kde:

P - pravděpodobnost přijetí změněného řešení,

$\delta \mathrm{E}$ - modul rozdílu mezi energií optimálního řešení a energií změněného řešení,

T - aktuální teplota.

5. Snížení teploty

Důležitou součástí algoritmu je snižování teploty. Při vysoké teplotě je pravděpodobnost výběru méně optimálního řešení vysoká. Během fungování algoritmu však teplota klesá a pravděpodobnost výběru méně optimálního řešení se snižuje. Výběr způsobu snížení teploty se může lišit a musí být zvolen experimentálně. Důležité je, aby teplota monotónně klesala k nule. Dobrou strategií je vynásobit teplotu v každém kroku součinitelem o něco menším než jedna.

6. Výběr počáteční a prahové teploty

Tento výběr by měl být také proveden experimentálně. Doporučením může být výběr prahové teploty blízké nule a poměrně vysoké výchozí teploty.

Oblasti aplikace algoritmu

1. Vytvoření cesty
2. Rekonstrukce obrazu
3. Zadání a plánování úkolů
4. Umístění sítě
5. Globální směrování
6. Detekce a rozpoznávání vizuálních objektů
7. Vývoj speciálních digitálních filtrů

3.2.2 Algoritmus Tabu search

Algoritmus Tabu search neboli algoritmus zakázaného vyhledávání je meta-algoritmus vyhledávání, který používá metody lokálního vyhledávání používané pro matematickou optimalizaci. Algoritmus byl vytvořen Fredem W. Gloverem v roce 1986 a formalizován v roce 1989.

Lokální vyhledávání přijímá potenciální řešení úlohy a kontroluje své bezprostřední sousedy (tj. řešení, která jsou podobná, s výjimkou několika velmi malých detailů) v naději, že najde vylepšené řešení. Metody místního vyhledávání mají tendenci uvíznout v suboptimálních oblastech nebo plošinách, kde je řada stejně vhodných řešení.

Algoritmus Tabu search zlepšuje výkon lokálního vyhledávání zeslabením jeho základního pravidla. Nejprve může být přijato zhoršení v každém kroku, pokud nedochází ke zlepšení (podobné případu, kdy vyhledávání skončilo v lokálním minimu). Kromě toho se zadávají zákazy (tabu), aby se zabránilo hledání již navštívených řešení.

Realizace algoritmu Tabu search používá struktury, které popisují navštívená řešení nebo vlastní sady pravidel. Pokud bylo potenciální řešení navštíveno během krátké doby nebo pokud to porušuje pravidlo, je označeno jako „tabu“, takže algoritmus nebude znovu prověřovat toto řešení.

Algoritmus Tabu search je meta-algoritmus, který lze použít k řešení úloh kombinatorické optimalizace (úlohy, kde je třeba najít optimální uspořádání a výběr možností).

Základní principy algoritmu

Algoritmus Tabu search využívá postup lokálního vyhledávání, které se bude iterativně posouvat z jednoho potenciálního řešení x na vylepšené řešení x' v sousedství x, dokud nedosáhneme splnění ukončovacího kritéria (obvykle to je počet iterací nebo prahová hodnota cílového odhadu). Algoritmy lokálního vyhledávání se často zasekávají v oblastech se špatným hodnocením optima nebo v oblastech, kde hodnocení tvoří plošinu (plochý vodorovný povrch). Aby se těmto variantám zabránilo a bylo možné prozkoumat oblasti vyhledávacího prostoru, které by zůstaly nezkoumány jinými vyhledávacími postupy, algoritmus Tabu search v průběhu vyhledávání prozkoumá okolí každého řešení. Řešení rozpoznaná novými sousedy $N*(x)$ jsou určena pomocí paměťových struktur. Při použití těchto struktur postupuje hledání iterativním přesunem z aktuálního řešení x do vylepšeného řešení x' ze seznamu $N*(x)$.

Tyto struktury tvoří tzv. tabu-seznamy, řadu pravidel a označených řešení používaných pro filtrování, která řešení od sousedů $N*(x)$ zpracovávat při vyhledávání. V nejjednodušší podobě je tabu-seznam krátkodobý soubor řešení, která byla navštívena během posledních n iterací, kde n se rovná počtu zapamatovaných řešení (toto číslo se nazývá doba platnosti). Tabu-seznam se častěji skládá z řešení, která byla změněna v procesu přechodu z jednoho řešení na druhé. Pro zjednodušení je vhodné chápat „řešení“ jako objekt zakódovaný a reprezentovaný některými atributy.

Druhy paměti

Struktury paměti používané při zakázaném vyhledávání lze zhruba rozdělit do tří kategorií:

Krátkodobé: Seznam nedávno prozkoumaných řešení. Pokud se potenciální řešení objeví v seznamu zakázaných, nemůže být podruhé navštíveno, dokud nedosáhneme vypršení platnosti zákazu.

Střednědobé: Pravidla zesílení určená k posunu vyhledávání směrem ke slibným oblastem vyhledávacího prostoru.

Dlouhodobé: Pravidla rozšíření, která vyhledávají nové oblasti.

Krátkodobé, střednědobé a dlouhodobé seznamy se mohou překrývat. Uvnitř těchto kategorií se paměť může dále rozlišovat podle kritérií, jako je frekvence nebo působení provedených změn. Jedním příkladem střednědobé struktury je zákaz nebo podpora řešení, která obsahují určité atributy (například řešení, která obsahují nežádoucí nebo žádoucí hodnoty určitých definovaných proměnných) nebo strukturu paměti, která zabraňuje nebo generuje určité pohyby (například na základě četnosti výskytu příznaků ve slibných řešeních nalezených

dříve). V krátkodobé paměti jsou vybrané atributy v nedávno navštívených řešeních označeny jako „tabu-aktivní“. Použití řešení s tabu-aktivními prvky je zakázáno. Ke změně stavu tabu-řešení se používá kritérium vymazání, včetně vyloučených řešení v seznamu povolených řešení (řešení je „dost dobré“ podle kvality). Jednoduchým a běžně používaným kritériem vymazání je použití řešení, která jsou lepší než v současnosti nalezené nejlepší řešení (lokální optimum).

Samotná krátkodobá paměť může být dostačující k získání řešení, které je lepší než u běžných metod místního vyhledávání, ale střednědobé a dlouhodobé struktury jsou často nezbytné k řešení složitějších problémů. Algoritmus Tabu search je často porovnáván s jinými meta-heuristickými metodami, jako je simulované žíhání, genetické algoritmy, mravenčí algoritmy, citlivé vyhledávání nebo hladový algoritmus. Mimo jiné je algoritmus Tabu search někdy kombinován s jinými meta-heuristikami za účelem vytvoření hybridních metod. Nejběžnější hybridní vyhledávání se zákazy vzniká kombinací s algoritmem Scatter Search. Jedná se o skupinu algoritmů, které mají společné vyhledávání zakázaných oblastí a které se často používají k řešení velkých nelineárních optimalizačních úloh.

Pseudokód

Následující pseudokód je zjednodušenou verzí algoritmu Tabu search se zákazy, jak je popsáno výše. Tato realizace má nejjednodušší verzi krátkodobé paměti a neobsahuje střednědobé ani dlouhodobé struktury. Termín "fitness" se týká výpočtu cílové funkce pro kandidáta na řešení.

```
 1 sBest ← s0
 2 bestCandidate ← s0
 3 tabuList ← []
 4 tabuList.push(s0)
 5 while (not stoppingCondition())
 6     sNeighborhood ← getNeighbors(bestCandidate)
 7     for (sCandidate in sNeighborhood)
 8             if ( (not tabuList.contains(sCandidate)) and
(fitness(sCandidate) > fitness(bestCandidate)) )
 9                     bestCandidate ← sCandidate
10             end
11     end
12     if (fitness(bestCandidate) > fitness(sBest))
13             sBest ← bestCandidate
14     end
15     tabuList.push(bestCandidate)
16     if (tabuList.size > maxTabuSize)
17             tabuList.removeFirst()
18     end
19 end
20 return sBest
```

Kód 1: Pseudokód Tabu Search

Řádky 1–4 provádějí výchozí přiřazení a vytvářejí výchozí řešení (získané metodami náhodného vyhledávání), nastavují výsledné řešení jako první zobrazené a inicializují tabu-seznam tímto řešením. V tomto příkladu je tabu-seznam krátkodobá struktura, která obsahuje záznamy již navštívených prvků.

Hlavní cyklus začíná od řádku 5. Tento cyklus pokračuje v hledání optimálního řešení, dokud neobdrží uživatelem definované ukončovací kritérium (dva příklady tohoto kritéria jsou časový limit nebo hodnota fitness funkce). Sousední řešení jsou kontrolována na jejich přítomnost v tabu seznamu v řádku 8. Kromě toho algoritmus ukládá nejlepší přípustná řešení v sousedství.

Cílová fitness funkce je obvykle matematická funkce, která vrací cílové hodnocení nebo kritérium - například nalezení nového vyhledávacího prostoru lze považovat za cílové kritérium. Pokud má nejlepší lokální potencionální řešení vyšší hodnotu fitness funkce, pak je aktuální hodnota lepší (řádek 12), nyní je přijata jako nejlepší (řádek 13). Místní lokální řešení je vždy přidáno do tabu-seznamu (řádek 15) a pokud je tabu-seznam plný (řádek 16), odstraňujeme z tabu seznamu první prvek (řádek 17). Obvykle jsou prvky ze seznamu odstraněny v pořadí, v jakém jsou v něm zadány. Postup vybere nejlepšího lokálního kandidáta i přesto, že je hodnota fitness funkce horší než sBest (proměnná sBest uchová nejlepší řešení), aby vyskočil z lokálního optima.

Tento postup pokračuje, dokud nedosáhneme uživatelsky definovaného ukončovacího kritéria, výstupem algoritmu je nejlepší řešení, které se v postupu vyskytlo (řádek 20).

Použití metody Tabu Search k řešení problému obchodního cestujícího

Úloha obchodního cestujícího se někdy používá k ukázce toho, jak vyhledávání pracuje se zákazy. Cílem této úlohy je nalezení nejkratší cesty k návštěvě všech měst, pokud je uveden seznam měst. Například pokud jsou město A a město B blízko sebe a město C je daleko od nich, celková délka cesty bude kratší, pokud poprvé navštívíme A a B a poté přejdeme do města C. Protože je nalezení optimálního řešení NP-obtížné, pro získání řešení blízkého optimálnímu jsou užitečné přibližné metody založené na heuristice (jako je lokální vyhledávání). Pro získání dobrého řešení úlohy obchodního cestujícího je důležité prostudovat strukturu grafu. Důležitost zkoumání struktury úlohy je opakujícím se tématem v meta-heuristických metodách a pro tento účel je vhodná metoda Tabu search. Skupina strategií souvisejících se zakázaným vyhledáváním, nazývaných metody získávání řetězců (ejection chain methods) umožňují účinně získat vysoce kvalitní řešení úlohy obchodního cestujícího.

Na druhé straně lze jednoduché vyhledávání se zákazy použít k nalezení uspokojivého řešení pro problém obchodního cestujícího, tj. řešení, které splňuje kritérium použitelnosti

(kritériem použitelnosti může být vzdálenost definovaná uživatelem). Hledání začíná výchozím řešením, které lze získat náhodně nebo libovolným algoritmem, například metodou nejbližšího souseda. Pro vytvoření nových řešení se pořadí, ve kterém jsou města navštěvována, zaměňuje. Celková vzdálenost cesty mezi všemi městy slouží k porovnání různých řešení. Aby se předešlo zacyklení, tj. opětnému získání určité sady řešení a uvíznutí v lokálním optimu, je řešení přidáno do seznamu zakázaných řešení. Nová řešení se vytváří, dokud nedosáhneme ukončovacího kritéria, například určitého počtu iterací. Jakmile se algoritmus Tabu search zastaví, vrací nejlepší řešení nalezené během průběhu algoritmu.

3.3 Evoluční metody

Evoluční metody patří mezi iterativní metody, které stochastické operátory používají pro skupinu jednotlivců, tvořících populaci. Každý jednotlivec v populaci je kódovanou verzí navrhovaného řešení. Zpočátku jsou tyto populace generovány náhodně. Vyhodnocovací funkce odpovídá hodnotě vhodnosti pro každého jednotlivce a vyhodnocuje jeho vhodnost pro danou úlohu. V této kapitole jsem čerpal z [BUONTEMPO, 2019], [DORIGO, 2004], [CHJO MU KHAN, 2010], [KOLESNIKOV et al., 2011], [SKIENA, 2010].

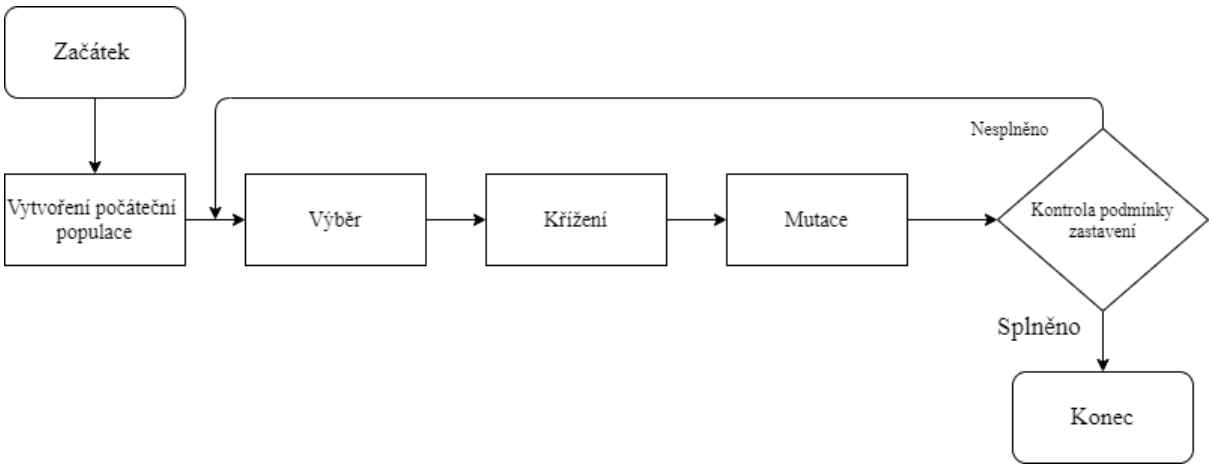
Evoluční algoritmy zahrnují genetické algoritmy, evoluční strategie a metodu Ant colony. Evoluční algoritmy používají náhodně generovanou populaci řešení. Počáteční populace se zlepšuje přirozeným evolučním procesem. S každou generací procesu je celá populace (nebo její část) nahrazena nově generovanými jedinci (obvykle lepšími než předchozí).

```
Generate(P( 0));
t := 0;
while not Termination-Criterion(P(t)) do
Evaluate( P( t ));
P'(t) := Selection(P(t));
P'(t) := ApplyReproduction-Ops(P'( t));
P(t + 1) := Replace(P(t), P'(t));
t := t + 1;
endwhile
```

Kód 2: Pseudokód evolučních algoritmů

3.3.1 Genetic algorithm (GA)

Genetické algoritmy patří do skupiny evolučních metod a napodobují evoluční procesy biologických organismů. V biologii byly přírodní populace studovány po mnoho generací a ukázalo se, že se vyvíjejí v souladu se zásadami přirozeného výběru a přežití „dobře přizpůsobených jedinců“, kteří jsou pro reprodukci nejvhodnější. Genetické algoritmy napodobují tento proces při řešení problémů s optimalizací. Podle tohoto paradigmatu se populace řešení (obvykle kódovaných ve formě bitových nebo celočíselných řetězců nazývaných chromozomy) vyvíjí z jedné generace na druhou použitím operátorů podobných těm, které existují v přírodě: výběr, genetické páření a mutace. V procesu výběru budeme nejlepší řešení považovat za rodiče k vytvoření potomstva. Proces křížení používá dvě vybraná rodičovská řešení a kombinuje jejich nejžádanější vlastnosti k získání jednoho nebo více lepších řešení-potomků.



Obrázek 15: Hlavní kroky GA

Rozebereme kroky GA. V prvním kroku je obvykle náhodně generována počáteční populace jednotlivců (soubor řešení). Poté je modelováno rozmnožování jednotlivců. Za tímto účelem je vybráno několik párů jednotlivců, v každém páru se provádí jejich kříženi a získaní nové jednotlivci (potomci) jsou umístěny do populace nové generace. Tyto kroky se provádějí, dokud nebude splněno ukončovací kritérium, např. počet generací, limit fitness funkce (algoritmus se zastaví, když nejlepší hodnota fitness funkce není větší než nastavená hodnota), počet generací beze změn (zastavení, pokud nedochází v rámci zadaného počtu generací ke zlepšení fitness funkce) atd.

Před řešením úlohy GA je nutné vyvinout metodu kódování řešení. Pro TSP byly vyvinuty nejméně dva způsoby, jak reprezentovat cesty, z nichž každá definuje soubor přijatelných operací křížení a mutace:

1. pořadové znázornění kóduje cestu se seznamem n měst, ve kterém je i-tý prvek číslo od 1 do n-i-1 (číslo města v seznamu dosud nenavštívených měst), např. cesta 1–2–4–3–8–5–9–6–7 bude znázorněna jako seznam (1, 1, 2, 1, 4, 1, 3, 1, 1). Hlavní výhoda pořadového znázornění je, že pro něj aplikujeme pouze klasický jednobodový crossover;

2. „cestovní“ znázornění - nejpřirozenější znázornění cesty. Např. cesta 5-1-7-8-9-4-6-2-3 je znázorněna jako (5, 1, 7, 8, 9, 4, 6, 2, 3). Pro znázornění cesty existují tři operace crossoveru: částečně zobrazený, pořadový a cyklický crossover.

Tato znázornění mají své výhody a nevýhody pro konkrétní úlohy. Může se ukázat, že v počáteční fázi práce GA je jedna operace crossoveru efektivnější a pokud již existují dostatečně dobrá řešení, bude mít jiná operace větší účinnost. Nejlepší výsledky se získávají použitím dynamické adaptace algoritmu. Podstata této adaptace spočívá v následujícím: algoritmus používá všechna znázornění a operace. Zpočátku jsou jednotlivci generováni v konkrétním znázornění se stejnou pravděpodobností. U jednotlivce se zaznamenává informace o typu znázornění a operace, kterou byl získán. Pravděpodobnost použití znázornění a operací se dále mění v závislosti na počtu jedinců jednoho nebo druhého druhu v populaci. Aby některé typy operací během výpočtu nezmizely, měla by zůstat pravděpodobnost použití kterékoli z implementovaných operací. Takovýto mechanismus komplikuje implementaci algoritmu, protože vyžaduje postupy pro převod chromozomů na různé typy reprezentace, avšak umožňuje algoritmu přizpůsobit se konkrétní úloze a získat přesnější řešení.

3.3.2 Algoritmus Ant colony

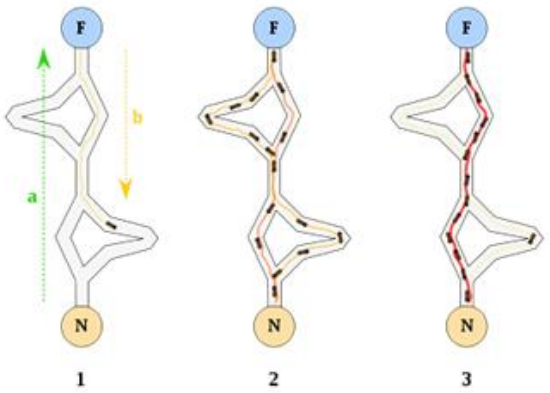
Algoritmus Ant colony (algoritmus optimalizace imitací mravenčí kolonie, angl. ant colony optimization, AC) je jedním z nejúčinnějších polynomických algoritmů pro nalezení přibližných řešení úloh obchodního cestujícího a pro řešení podobných úloh při hledání cest v grafech. Podstatou tohoto přístupu je analýza a použití modelu chování mravenců hledajících cesty od kolonie ke zdroji potravy. Jedná se o metaheuristickou optimalizaci.

První verze algoritmu, kterou navrhl Dr. Marco Dorigo v roce 1992, byla zaměřena na nalezení optimální cesty v grafu.

Řešení není přesné a může být dokonce jedním z nejhorších, avšak díky dobře zvoleným heuristikám dává iterační aplikace algoritmu obvykle výsledek téměř optimální.

Obecný výklad

Ve skutečném světě mravenci zpočátku chodí v náhodném pořadí a poté, co najdou jídlo, se vracejí do své kolonie a vytváří stopy s feromony. Pokud ostatní mravenci takové stopy najdou, pravděpodobně je sledují. Namísto sledování řetězu jej posilují, když se vrátí, pokud nakonec najdou zdroj potravy. Postupem času se feromonová stopa začíná odpařovat, čímž se snižuje její atraktivita. Čím déle trvá cesta k cíli a zpět, tím více se feromonová cesta vypařuje. Na krátké cestě bude pro srovnání průchod rychlejší a v důsledku toho zůstává hustota feromonů vysoká. Odpařování feromonů má také tu vlastnost, že se vyhýbá touze po místně optimálním řešení. Pokud by se feromony neodpařovaly, byla by cesta vybraná jako první nejefektivnější. V tomto případě by byl výzkum prostorových řešení omezen. Když tedy jeden mravenec najde (například krátkou) cestu z kolonie ke zdroji potravy, je pravděpodobné, že tuto cestu budou následovat další mravenci, a pozitivní recenze nakonec povedou všechny mravence na jednu nejkratší cestu.



Obrázek 16: Pohyb mravenčích kolonií při hledání nejkratší cesty

Původní myšlenka vychází z pozorování mravenců v procesu nalezení nejkratší cesty z kolonie ke zdroji potravy.

1. První mravenec najde zdroj potravy (F) jakýmkoli způsobem (a) a poté se vrací do hnízda (N) a zanechává za sebou stopu feromonů.
2. Poté si mravenci vyberou jednu ze čtyř možných cest, pak ji zesílí a učiní atraktivní.
3. Mravenci volí nejkratší cestu, protože feromony z delších cest se rychleji vypařují.

Mezi experimenty s výběrem mezi dvěma cestami nerovnoměrné délky vedoucími od kolonie ke zdroji potravy si biologové všimli, že mravenci zpravidla používají nejkratší cestu. Model tohoto chování spočívá v následujícím:

- Mravenec (tzv. „Blitz“) prochází náhodně z kolonie
- Pokud najde zdroj potravy, vrátí se do hnízda a zanechá stopu feromonu

- Tyto feromony přitahují další mravence v okolí, kteří, s největší pravděpodobností, budou následovat tuto cestu
- Po návratu do hnízda posílí feromonovou cestu
- Pokud existují 2 cesty, pak kratší cestou projde za stejnou dobu více mravenců než delší cestou
- Krátká cesta se stane atraktivnější
- Dlouhé cesty nakonec zmizí v důsledku odpařování feromonů

Mravenci používají okolní prostředí jako prostředek komunikace. Během své „práce“ si nepřímo vyměňují informace prostřednictvím feromonů. Výměna informací je místní povahy, o nich se mohou dozvědět pouze ti mravenci, kteří jsou v bezprostřední blízkosti, kde feromony zůstaly. Takový systém se nazývá stigmergie a platí pro mnoho společenských zvířat (byl studován pro stavbu sloupů v termitních hnízdech). Tento mechanismus pro řešení problému je velmi složitý a je dobrým příkladem samoorganizace systému. Takový systém je založen na pozitivní (jiní mravenci posilují feromonovou cestu) a negativní (odpařování feromonové stopy) zpětné vazbě. Teoreticky, pokud se počet feromonů v průběhu času na všech cestách nezmění, nebude možné zvolit cestu. Díky zpětné vazbě však malé výkyvy povedou k posílení jedné z cest a systém se stabilizuje a nalezne nejkratší cestu.

4. SROVNÁVACÍ ANALÝZA IMPLEMENTOVANÝCH ALGORITMŮ

V této kapitole jsou představeny výsledky činnosti algoritmů realizovaných v rámci diplomové práce. Pro optimalizaci činnosti některých algoritmů je také třeba vzít v úvahu klíčové parametry. V průběhu praktického testování implementovaných algoritmů byly odvozeny optimální parametry algoritmů.

Jako testovací data byly použity soubory se souřadnicemi na osách (x, y), soubory knihovny TSP95 a reálná geodata používaná prostřednictvím služby Google Maps.

Jako hlavní ukazatele pro hodnocení účinnosti algoritmů byly vybrány: vzdálenosti, které jsou konečným výsledkem činnosti algoritmu, procentuální odchylka výsledku od optimálního výsledku, jakož i čas strávený algoritmem k nalezení konečného výsledku.

Měření probíhalo na testovacím prostředí, kterým byl:

notebook Dell Inspiron 15-7559

Processor: Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz

RAM: 16.0 GB

OS: Microsoft Windows 10 Home Single Edition.

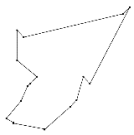
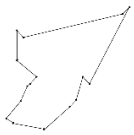
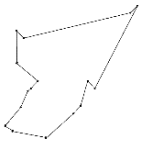
4.1 Branch and bound

Výsledky této metody závisí především na konkrétní implementaci algoritmu, protože v této metodě neexistují žádné přenášené parametry.

Vzhledem k tomu, že mnou realizovaná implementace Branch and bound je schopna efektivně zpracovat maximálně 20 měst, jako testovací data budou použity soubory se souřadnicemi na osách (x, y) s počtem bodů: 10, 15, 16, 17, 18, 19, 20.

File name			coords10	coords15	coords16	coords17
Optimal tour length			2526,55950702768	2567,97034796402	2644,47646803748	2646,07660946633
Number of cities			10	15	16	17
Branch and bound	Tour length	avg	2526,5595070276 (+ 0%)	2591,32874286324 (+ 0,909%)	2667,83486293671 (+0,883%)	2734,00386416624 (+3,322%)
		min	2526,55950702768 (+ 0%)	2591,32874286324 (+ 0,909%)	2667,83486293671 (+0,883%)	2734,00386416624 (+3,322%)
		max	2526,55950702768 (+ 0%)	2591,32874286324 (+ 0,909%)	2667,83486293671 (+0,883%)	2734,00386416624 (+3,322%)
	Time (sec)	avg	0,012	0,165	0,532	1,872
		min	0,01	0,16	0,51	1,84
		max	0,02	0,17	0,57	1,9
	Tour					
Number of iterations			100	100	100	100

Tabulka 2: Výsledky činnosti algoritmů (1). B&B.

File name			coords18	coords19	coords20
Optimal tour length			2647,99954653625	2649,27761535269	2649,35397500336
Number of cities			18	19	20
Branch and bound	Tour length	avg	2654,0632773507 (+ 0,228%)	2655,34134616714 (+ 0,23%)	2655,41770581781 (+ 0,3%)
		min	2654,0632773507 (+ 0,228%)	2655,34134616714 (+ 0,23%)	2655,41770581781 (+ 0,3%)
		max	2654,0632773507 (+ 0,228%)	2655,34134616714 (+ 0,23%)	2655,41770581781 (+ 0,3%)
	Time (sec)	avg	3,63	19,346	47,82
		min	3,52	18,72	46,76
		max	3,82	19,95	51,24
	Tour				
Number of iterations			100	100	100

Tabulka 3: Výsledky činnosti algoritmů (2). B&B.

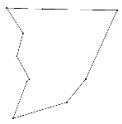
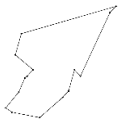
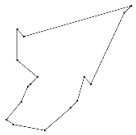
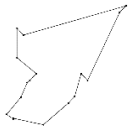
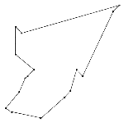
Obrázek 17: Graf závislosti doby činnosti algoritmu na počtu měst. B&B.

Metoda Branch and Bound je jednou z přesných metod pro nalezení nejkratší cesty s ohledem na problém obchodního cestujícího. Jak vidíme z grafu závislosti času na počtu měst, tento algoritmus velmi efektivně nachází cestu pro úlohy s malým počtem měst. Nicméně, s rostoucím počtem měst tato metoda není příliš efektivní vzhledem k delší době provedení.

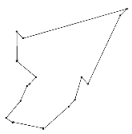
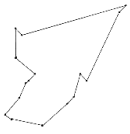
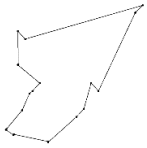
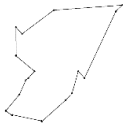
4.2 Dynamic programming

Výsledky této metody závisí především na konkrétní implementaci algoritmu, protože v této metodě neexistují žádné přenášené parametry, jako v předchozí metodě.

Vzhledem k tomu, že mnou realizovaná implementace algoritmu dynamického programování je schopna efektivně zpracovat maximálně 23 měst, jako testovací data budou použity soubory se souřadnicemi na osách (x, y) s počtem bodů: 10, 15, 16, 17, 18, 19, 20, 21, 22, 23.

File name			coords10	coords15	coords16	coords17	coords18
Optimal tour length			2526,55950702768	2567,97034796402	2644,47646803748	2646,07660946633	2647,99954653625
Number of cities			10	15	16	17	18
Dynamic programming	Tour length	avg	2526,5595070276 (+ 0%)	2567,97034796402 (+ 0%)	2644,47646803748 (+ 0%)	2646,07660946633 (+ 0%)	2647,99954653625 (+ 0%)
		min	2526,55950702768 (+ 0%)	2567,97034796402 (+ 0%)	2644,47646803748 (+ 0%)	2646,07660946633 (+ 0%)	2647,99954653625 (+ 0%)
		max	2526,55950702768 (+ 0%)	2567,97034796402 (+ 0%)	2644,47646803748 (+ 0%)	2646,07660946633 (+ 0%)	2647,99954653625 (+ 0%)
	Time (sec)	avg	0,016	0,032	0,065	0,152	0,418
		min	0,01	0,02	0,05	0,11	0,36
		max	0,03	0,05	0,08	0,21	0,52
	Tour						
Number of iterations			100	100	100	100	100

Tabulka 4: Výsledky činnosti algoritmů (1). DP.

File name			coords19	coords20	coords21	coords22	coords23
Optimal tour length			2649,27761535269	2649,35397500336	2649,4904176735	2661,17626985874	2662,79556235854
Number of cities			19	20	21	22	23
Dynamic programming	Tour length	avg	2649,27761535269 (+ 0%)	2649,35397500336 (+ 0%)	2649,4904176735 (+ 0%)	2661,17626985874 (+ 0%)	2662,79556235854 (+ 0%)
		min	2649,27761535269 (+ 0%)	2649,35397500336 (+ 0%)	2649,4904176735 (+ 0%)	2661,17626985874 (+ 0%)	2662,79556235854 (+ 0%)
		max	2649,27761535269 (+ 0%)	2649,35397500336 (+ 0%)	2649,4904176735 (+ 0%)	2661,17626985874 (+ 0%)	2662,79556235854 (+ 0%)
	Time (sec)	avg	1,01	2,68	6,152	14,412	32,548
		min	0,91	2,5	5,99	13,89	27
		max	1,1	2,8	6,31	15,24	36,5
	Tour						
Number of			100	100	100	100	100

Tabulka 5: Výsledky činnosti algoritmů (2). DP.

Obrázek 18: Graf závislosti doby činnosti algoritmu na počtu měst. DP.

Jak vyplývá ze získaných dat, metoda dynamického programování je velmi přesná. Funguje rychleji než předchozí metoda. Jak je však patrné z grafu závislosti času na počtu měst, s velkým počtem měst není metoda dynamického programování dostatečně efektivní vzhledem k delší době provedení.

4.3 Simulated annealing

V algoritmu SA jsou klíčovými parametry, které ovlivňují výsledek algoritmů, počáteční teplota a rychlost chlazení. Na základě provedených zkoušek byly odvozeny parametry, které poskytují optimální výsledek.

Algorithm	Parameter	TSP
Simulated annealing	T_0	$500 * n$
	α	$1-(5E-3)/n^2$

Tabulka 6: Parametry metody SA.

Parametru „počáteční teplota“ T_0 v kódu odpovídá proměnná **temperature**.

n - počet měst.

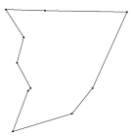
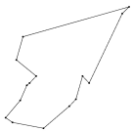
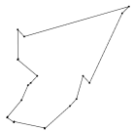
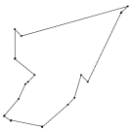
Parametru „rychlost chlazení“ α v kódu odpovídá proměnná **coolingRate.**

Při každé iteraci teplota klesne o $1-0{,}005/n^2$ krát. To znamená, že v případě, že počet měst je n = 100, je součinitel 0,9999995.

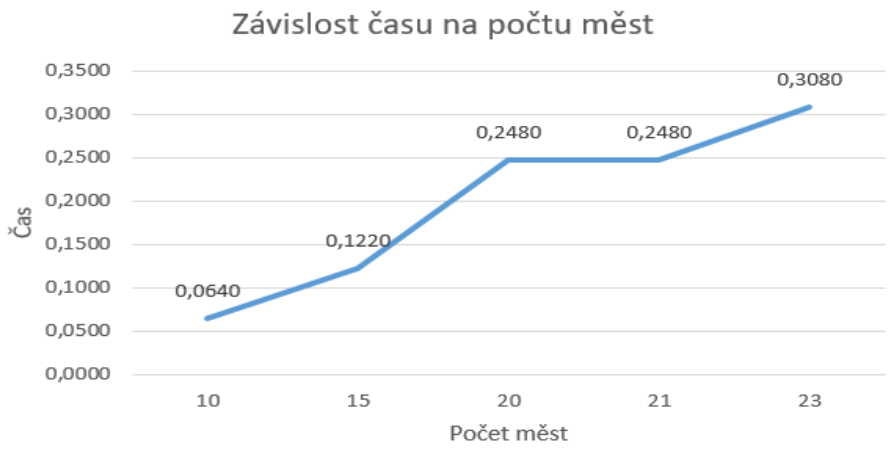
```
public Solver()
{
        temperature = n => 500 * n;
        coolingRate = n => 1 - (5E-3) / (n * n);
}
```

Kód 3: Inicializace proměnných pomocí delegátů. Algoritmus SA.

Jako testovací data budou použity soubory se souřadnicemi na osách (x, y) s počtem bodů: 10, 15, 20, 21, 23.

File name			coords10	coords15	coords20	coords21	coords23
Optimal tour length			2526,55950702768	2567,97034796402	2649,35397500336	2649,4904176735	2662,79556235854
Number of cities			10	15	20	21	23
Simulated annealing	Tour length	avg	2526,55950702768 (+ 0%)	2567,97034796402 (+ 0%)	2649,35397500336 (+ 0%)	2649,4904176735 (+ 0%)	2662,79556235854 (+ 0%)
		min	2526,55950702768 (+ 0%)	2567,97034796402 (+ 0%)	2649,35397500336 (+ 0%)	2649,4904176735 (+ 0%)	2662,79556235854 (+ 0%)
		max	2526,55950702768 (+ 0%)	2567,97034796402 (+ 0%)	2649,35397500336 (+ 0%)	2649,4904176735 (+ 0%)	2662,79556235854 (+ 0%)
	Time (sec)	avg	0,064	0,122	0,248	0,244	0,308
		min	0,05	0,07	0,2	0,2	0,28
		max	0,09	0,17	0,28	0,3	0,34
	Tour						
Number of iterations			100	100	100	100	100

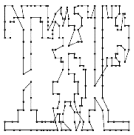
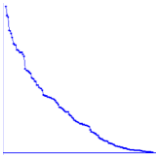
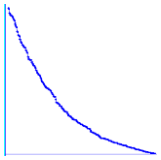
Tabulka 7: Výsledky činnosti algoritmů (1). SA.



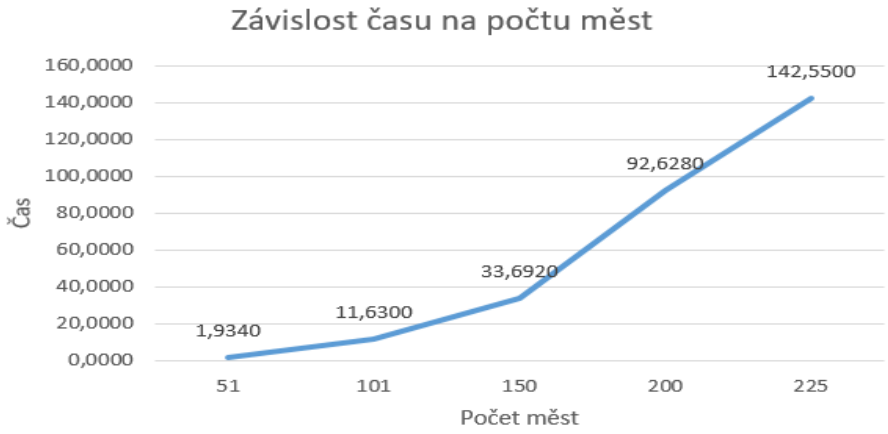
Obrázek 19: Graf závislosti doby činnosti algoritmu na počtu měst (1). SA

Vzhledem k tomu, tento algoritmus je velmi účinná metaheuristická metoda pro řešení diskrétních optimalizačních úloh, k testování algoritmu jsou také použity soubory knihovny TSP95 s velkým počtem měst ve srovnání s předchozími algoritmy.

Při zpracování souborů s velkým počtem měst můžeme, kromě zobrazení cesty, také vizualizovat závislost změny délky cesty na době činnosti algoritmu. Získaná data se vyhladí pomocí „klouzavého průměru“ a zobrazí se v podobě grafu. Tato vizualizace je uvedena v řádku „Optimization process“.

File name			eil51	eil101	kroB150	kroA200	tsp225
Optimal tour length			426	629	26130	29368	3919
Number of cities			51	101	150	200	225
Simulated annealing	Tour length	avg	449,311 (+ 5,47%)	679,5 (+ 8,02%)	28749 (+ 10,02)	32636,02 (+11,12%)	4224,65 (+7,9%)
		min	440,523 (+ 3,4%)	650,89 (+ 3,48%)	27150,62 (+ 3,9%)	30512,75 (+ 3,89%)	4019,75 (+2,57%)
		max	456,72 (+ 7,211%)	695,7 (+ 10,61%)	29582,25 (+ 13,2 %)	33452,23 (+13,9%)	4350,55 (+11,01%)
	Time (sec)	avg	1,934	11,63	33,692	92,628	142,55
		min	1,7	10,7	29	85,71	119,56
		max	2,3	12,5	35,38	110,17	181,7
	Tour						
	Optimization process						
Number of iterations			100	100	100	100	100

Tabulka 8: Výsledky činnosti algoritmů (2). SA.



Obrázek 20: Graf závislosti doby činnosti algoritmu na počtu měst (2). SA.

Jak vyplývá ze získaných dat, metoda založená na činnosti algoritmu simulovaného žíhání je dobrým nástrojem pro nalezení nejkratší cesty. Tato metoda je schopna efektivně zpracovávat soubory jak s malým, tak i s velkým počtem měst v relativně krátké době.

4.4 Tabu Search

V algoritmu Tabu search jsou následující klíčové parametry: velikost seznamu kandidátů a velikost tabu seznamu.

Algorithm	Parameter	TSP
Tabu search	N_C	50
	N_T	12

Tabulka 9: Parametry metody TS.

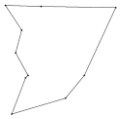
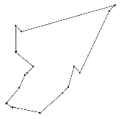
Parametru „velikost seznamu kandidátů“ N_C v kódu odpovídá pole int[,] candidateList.

Parametru „velikost tabu seznamu“ N_T v kódu odpovídá proměnná int tabuListLength.

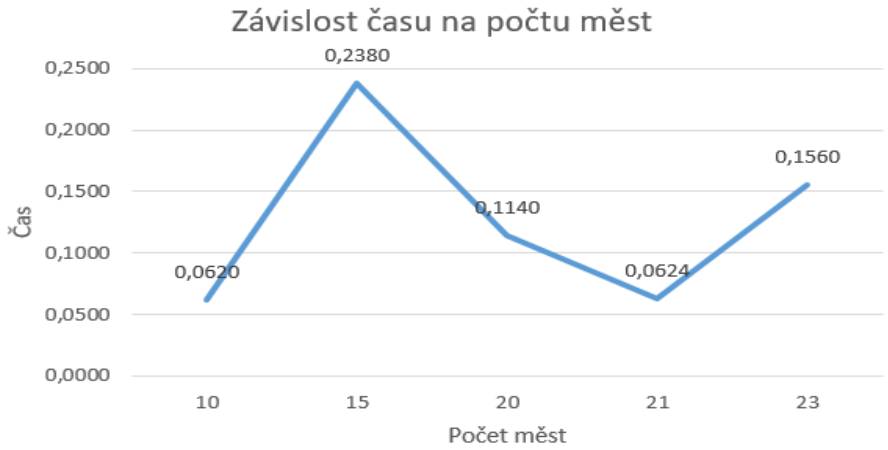
```
public Solver()
        {
            candidateList = n => 50;
            tabuListLength = n => 12;
        }
```

Kód 4: Inicializace proměnných pomocí delegátů. TS.

Jako testovací data budou použity soubory se souřadnicemi na osách (x, y) s počtem bodů: 10, 15, 20, 21, 23.

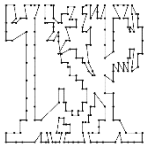
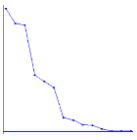
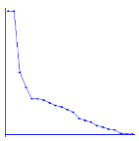
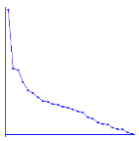
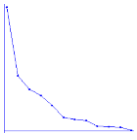
File name			coords10	coords15	coords20	coords21	coords23
Optimal tour length			2526,55950702768	2567,97034796402	2649,35397500336	2649,4904176735	2662,79556235854
Number of cities			10	15	20	21	23
Tabu search	Tour length	avg	2526,55950702768 (+ 0%)	2567,97034796402 (+ 0%)	2649,35397500336 (+ 0%)	2649,4904176735 (+ 0%)	2662,79556235854 (+ 0%)
		min	2526,55950702768 (+ 0%)	2567,97034796402 (+ 0%)	2649,35397500336 (+ 0%)	2649,4904176735 (+ 0%)	2662,79556235854 (+ 0%)
		max	2526,55950702768 (+ 0%)	2567,97034796402 (+ 0%)	2649,35397500336 (+ 0%)	2649,4904176735 (+ 0%)	2662,79556235854 (+ 0%)
	Time (sec)	avg	0,062	0,238	0,114	0,0624	0,156
		min	0,03	0,03	0,09	0,012	0,14
		max	0,1	0,92	0,15	0,1	0,17
	Tour						
Number of iterations			100	100	100	100	100

Tabulka 10: Výsledky činnosti algoritmů (1). TS.

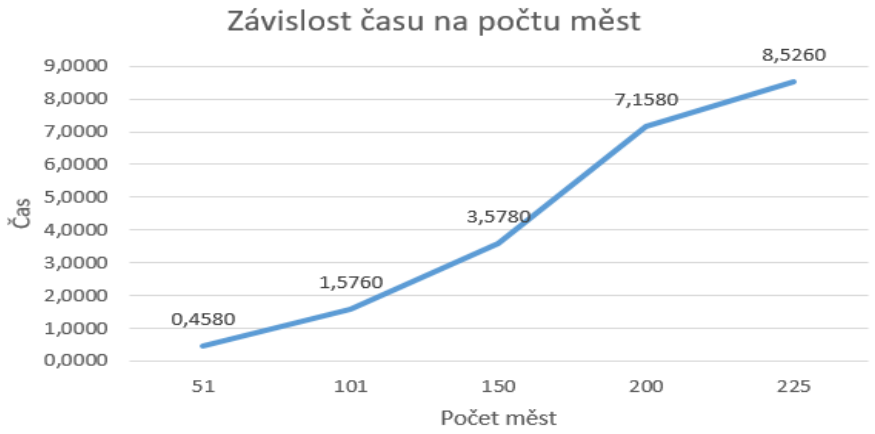


Obrázek 21: Graf závislosti doby činnosti algoritmu na počtu měst (1). TS.

Jako testovací data byly pro činnost algoritmu, stejně jako v předchozí metodě, použity soubory knihovny TSP95.

File name			eil51	eil101	kroB150	kroA200	tsp225
Optimal tour length			426	629	26130	29368	3919
Number of cities			51	101	150	200	225
Tabu search	Tour length	avg	428,83 (+ 0,66%)	645,63 (+ 2,64%)	26332,84 (+0,776%)	29755,47 (+1,319%)	3961,03 (+1,073%)
		min	427,12 (+ 0,26%)	635,89 (+ 1,09%)	26191,57 (+0,236%)	29412,14 (+0,15%)	3921,485 (+0,063%)
		max	430,21 (+ 0,98%)	650,003 (+ 3,33%)	26571,43 (+1,689%)	29994,46 (+2,133%)	3986,453 (+1,721%)
	Time (sec)	avg	0,458	1,576	3,578	7,158	8,526
		min	0,37	1,37	3,12	6,5	7,12
		max	0,56	1,82	4,7	7,9	9,8
	Tour						
	Optimization process						
Number of iterations			100	100	100	100	100

Tabulka 11: Výsledky činnosti algoritmů (2). TS.



Obrázek 22: Graf závislosti doby činnosti algoritmu na počtu měst (2). TS.

Jak je zřejmé z tabulek, metoda založená na činnosti algoritmu Tabu search je velmi výkonným nástrojem pro nalezení nejkratší cesty. Tato metoda je schopna efektivně zpracovat soubory jak s malým, tak i s velkým počtem měst v minimálním čase.

4.5 Genetic algorithm

Klíčové parametry jsou: pravděpodobnost mutace, konstantní hodnota pro metodu turnajového výběru, velikost populace a hodnota pro zastavení algoritmu.

Algorithm	Parameter	TSP
Genetic Algorithm	M	0,01
	N_S	5
	N_P	50
	Q	0,001

Tabulka 12: Parametry GA

Parametru M v kódu odpovídá proměnná **mutation**. Tato hodnota udává pravděpodobnost mutace v posloupnosti měst uvnitř cesty.

Parametru N_S v kódu odpovídá proměnná **tournamentSize.** Tato hodnota udává konstantu pro metodu turnajového výběr, která obsahuje počet náhodných cest a výběr nejkratší z nich.

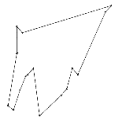
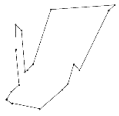
Parametru N_P v kódu odpovídá proměnná **sizePopulation.** Tato hodnota udává velikost populace.

Parametru Q v kódu odpovídá proměnná **qualityPercent.** Tato hodnota ovlivňuje zastavení algoritmu, a sice: pokud se během předchozí iterace cesta nezlepší o 0,001 (0,1%) ve srovnání s předchozí, algoritmus se zastaví.

```
public Solver()
        {
            qualityPercent = n => 0.001;
            mutation = n => 0.01;
            tournamentSize = n => 5;
            sizePopulation = n => 50;
        }
```

Kód 5: Inicializace proměnných pomocí delegátů. GA.

Jako testovací data budou použity soubory se souřadnicemi na osách (x, y) s počtem bodů: 10, 15, 20, 21, 23.

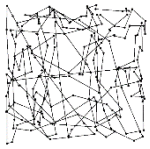
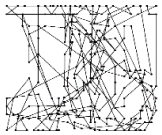
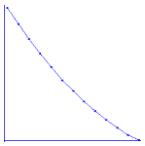
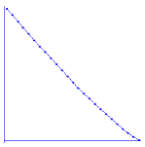
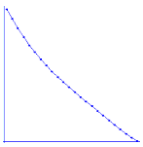
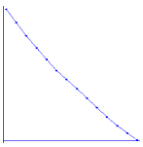
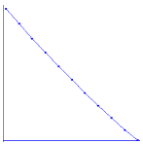
File name			coords10	coords15	coords20	coords21	coords23
Optimal tour length			2526,55950702768	2567,97034796402	2649,35397500336	2649,4904176735	2662,79556235854
Number of cities			10	15	20	21	23
Genetic algorithm	Tour length	avg	2526,56 (+ 0%)	2673,01 (+ 4,09%)	2719,74 (+ 2,65%)	2764,23 (+ 4,33)	2868,81 (+ 7,73%)
		min	2526,56 (+ 0%)	2567,97 (+ 0%)	2649,35 (+ 0%)	2649,49 (+ 0%)	2662,79 (+ 0%)
		max	2526,56 (+ 0%)	2719,2 (+ 5,88%)	2860,32 (+ 7,96%)	2860,46 (+ 7,96)	3388,12 (+ 27,23%)
	Time (sec)	avg	10,25	11,68	15,2	14,6	24,83
		min	9,89	11,42	12,1	14,05	14,96
		max	10,56	11,92	21,7	22,7	32,37
	Tour						
Number of iterations			100	100	100	100	100

Tabulka 13: Výsledky činnosti algoritmů (1). GA.

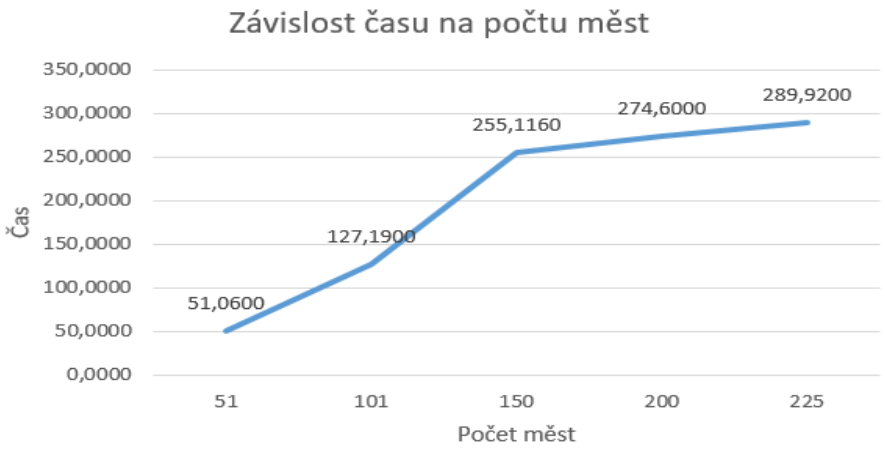
Obrázek 23: Graf závislosti doby činnosti algoritmu na počtu měst (1). GA.

Za použití metody založené na Genetickém algoritmu je možné docela efektivně najít optimální délku cesty při práci s malým počtem bodů.

Jako testovací data byly pro činnost algoritmu, stejně jako v předchozí metodě, použity soubory knihovny TSP95.

File name			eil51	eil101	kroB150	kroA200	tsp225
Optimal tour length			426	629	26130	29368	3919
Number of cities			51	101	150	200	225
Genetic Algorithm	Tour length	avg	497,86 (+ 16,86%)	769,42 (+ 22,32%)	36558,79 (+40,02%)	44794,19 (+52,52%)	7257,7 (+85,19%)
		min	460,55 (+ 8,11%)	710,75 (+ 12,99%)	30717,73 (+17,55%)	33887,4 (+15,38%)	4572,12 (+16,66%)
		max	520,25 (+ 22,12%)	809,08 (+ 28,62%)	40802,23 (+56,15%)	55600,92 (+89,32%)	8892,3 (+126,9%)
	Time (sec)	avg	51,06	127,19	255,116	274,6	289,92
		min	34,26	88,75	179,13	192	225,1
		max	59,52	155,46	283,38	312	323,5
	Tour						
	Optimization process						
Number of iterations			100	100	100	100	100

Tabulka 14: Výsledky činnosti algoritmů (2). GA.



Obrázek 24: Graf závislosti doby činnosti algoritmu na počtu měst (2). GA.

Jak vyplývá ze získaných dat, tato implementace genetického algoritmu je účinná při testování na malých objemech dat, avšak s velkým počtem měst tato metoda funguje nepřesně a po dlouhou dobu. Zejména ve srovnání s jinými uvažovanými metodami diskrétní optimalizace.

4.6 Ant colony

Klíčové parametry v algoritmu jsou: počet mravenců a hodnota (q) k zastavení algoritmu.

Algorithm	Parameter	TSP
ACO	N_α	n
	Q	0,01

Tabulka 15: Parametry metody AC.

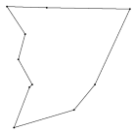
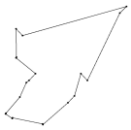
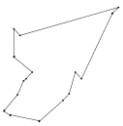
Parametru N_α v kódu odpovídá proměnná **numberAnt**. Tato hodnota udává počet mravenců v kolonii. V případě, že proměnná numberAntFunction je n, počet mravenců se rovná počtu měst v souboru.

Parametru Q v kódu odpovídá proměnná **qualityPercent.** Tato hodnota ovlivňuje zastavení algoritmu, a sice: pokud se během předchozí iterace cesta nezlepší o 0,01 (1%) ve srovnání s předchozí, algoritmus se zastaví.

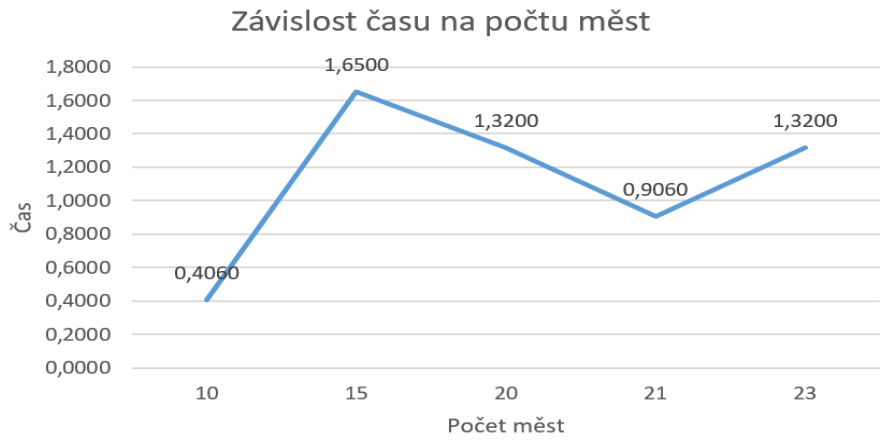
```
public Solver()
    {
      qualityFunction = n => 0.01;
      numberAntFunction = n => n;
    }
```

Kód 6: Inicializace proměnných pomocí delegátů. AC

Jako testovací data budou použity soubory se souřadnicemi na osách (x, y) s počtem bodů: 10, 15, 20, 21, 23.

File name			coords10	coords15	coords20	coords21	coords23
Optimal tour length			2526,55950702768	2567,97034796402	2649,35397500336	2649,4904176735	2662,79556235854
Number of cities			10	15	20	21	23
Ant colony	Tour length	avg	2526,55 (+ 0%)	2567,97 (+ 0%)	2663,69 (+ 0,54%)	2662,06 (+ 1,186%)	2671,85 (+ 0,34%)
		min	2526,55 (+ 0%)	2567,97 (+ 0%)	2649,35 (+ 0%)	2649,49 (+ 0%)	2662,79 (+ 0%)
		max	2526,55 (+ 0%)	2567,97 (+ 0%)	2683,86 (+ 1,3%)	2680,91 (+ 1,18%)	2695,82 (+ 1,24%)
	Time (sec)	avg	0,406	1,65	1,32	0,906	1,32
		min	0,02	0,06	0,07	0,11	0,2
		max	0,63	2,09	4,93	2,9	3,2
	Tour						
Number of iterations			100	100	100	100	100

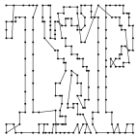
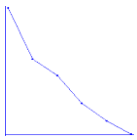
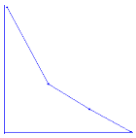
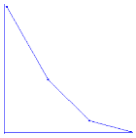
Tabulka 16: Výsledky činnosti algoritmů (1). AC.



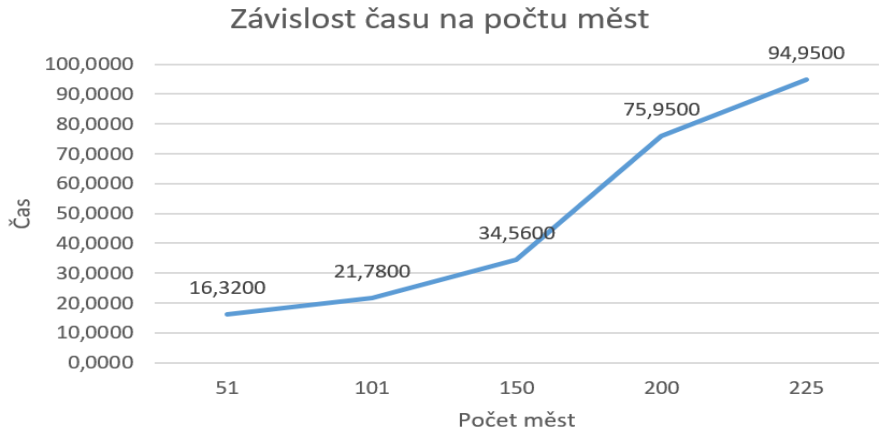
Obrázek 25: Graf závislosti doby činnosti algoritmu na počtu měst (1). AC

Jak je vidět z tabulky, algoritmus Ant colony dokáže poměrně přesně a rychle najít optimální cestu při práci s malým počtem bodů.

Jako testovací data byly pro činnost algoritmu, stejně jako v předchozí metodě, použity soubory knihovny TSP95.

File name			eil51	eil101	kroB150	kroA200	tsp225
Optimal tour length			426	629	26130	29368	3919
Number of cities			51	101	150	200	225
Ant colony	Tour length	avg	451,2 (+ 5,9%)	703,34 (+ 11,82%)	29274,04 (+ 12,03%)	33623,95 (+ 14,4%)	4347,61 (+ 10,93%)
		min	441,7 (+ 3,93%)	688,1 (+ 9,39%)	28188,88 (+ 7,87%)	32226,77 (+ 9,73%)	4299,8 (+ 9,71%)
		max	458,46 (+7,62 %)	715,89 (+ 13,81%)	29832,73 (+ 14,17%)	35382,53 (+ 20,48%)	4450,6 (+ 13,56%)
	Time (sec)	avg	16,32	21,78	34,56	75,95	94,95
		min	2,34	4,5	4,12	5,64	16,34
		max	32,5	56,71	90,28	139,29	175
	Tour						
	Optimization process						
Number of iterations			100	100	100	100	100

Tabulka 17: Výsledky činnosti algoritmů (2). AC.



Obrázek 26: Graf závislosti doby činnosti algoritmu na počtu měst (2). AC

Jak je vidět z tabulky, algoritmus Ant colony dokáže najít optimální vzdálenost při práci s velkým počtem měst. Velikost maximální průměrné odchylky při práci se soubory TSP činí 14,4% při počtu měst rovném 200. Z tabulky je také vidět, že algoritmus má velkou časovou volatilitu. Je to podmíněno vlastnostmi metody Ant colony a konkrétní implementací algoritmu.

4.7 Porovnání výsledků činnosti algoritmů na reálných geodatech

Pro porovnání výsledků činnosti algoritmů s reálnými geodaty byl vytvořen soubor s příponou KML, který ukládá seznam čtrnácti největších měst České republiky: Praha, Brno, Ostrava, Plzeň, Olomouc, Liberec, České Budějovice, Hradec Králové, Ústí nad Labem, Pardubice, Havířov, Zlín, Kladno, Most.

File name			cities.kml	Tour visualization
Number of cities			14	
Algorithm	BB	avg Distance	1 219 477,241 (+5,45%)	
		avg Time (sec)	0,24	
	DP	avg Distance	1 156 468,477 (+ 0%)	
		avg Time (sec)	0,52	
	SA	avg Distance	1 156 472,574 (+ 0,00035%)	
		avg Time (sec)	0,239	
	TS	avg Distance	1 156 468,477 (+ 0%)	
		avg Time (sec)	0,115	
	GA	avg Distance	1 156 472,574 (+ 0,00035%)	
		avg Time (sec)	20,2	
	AC	avg Distance	1 156 472,574 (+ 0,00035%)	
		avg Time (sec)	3,5	
Number of iterations	100			

Tabulka 18: Výsledky algoritmů při práci s reálnými geodaty

Jak je vidět z tabulky, všechny algoritmy poměrně efektivně vypořádali s problémem obchodního cestujícího při práci s reálnými geodaty.

4.8 Porovnání výsledků činnosti všech algoritmů

V následujících tabulkách jsou uvedeny výsledky činnosti všech algoritmů.

File name		coords10	coords15	coords20	coords21	coords23
Optimal tour length		2526,55950702768	2567,97034796402	2649,35397500336	2649,4904176735	2662,79556235854
Number of cities		10	15	20	21	23
BB	avg Distance	2526,5595070276 (+ 0%)	2591,32874286324 (+ 0,909%)	2655,41770581781 (+ 0,3%)	-	-
	avg Time (sec)	0,012	0,165	47,82	-	-
DP	avg Distance	2526,5595070276 (+ 0%)	2567,97034796402 (+ 0%)	2649,35397500336 (+ 0%)	2649,4904176735 (+ 0%)	2662,79556235854 (+ 0%)
	avg Time (sec)	0,016	0,032	2,68	6,152	32,548
SA	avg Distance	2526,55950702768 (+ 0%)	2567,97034796402 (+ 0%)	2649,35397500336 (+ 0%)	2649,4904176735 (+ 0%)	2662,79556235854 (+ 0%)
	avg Time (sec)	0,064	0,122	0,248	0,244	0,308
TS	avg Distance	2526,55950702768 (+ 0%)	2567,97034796402 (+ 0%)	2649,35397500336 (+ 0%)	2649,4904176735 (+ 0%)	2662,79556235854 (+ 0%)
	avg Time (sec)	0,062	0,238	0,114	0,0624	0,156
GA	avg Distance	2526,56 (+ 0%)	2673,01 (+ 4,09%)	2719,74 (+ 2,65%)	2764,23 (+ 4,33)	2868,81 (+ 7,73%)
	avg Time (sec)	10,25	11,68	15,2	14,6	24,83
AC	avg Distance	2526,55 (+ 0%)	2567,97 (+ 0%)	2663,69 (+ 0,54%)	2662,06 (+ 1,186%)	2671,85 (+ 0,34%)
	avg Time (sec)	0,406	1,65	1,32	0,906	1,32
Number of iterations		100	100	100	100	100

Tabulka 19: Porovnání výsledků činnosti všech algoritmů (1)

File name		eil51	eil101	kroB150	kroA200	tsp225
Optimal tour length		426	629	26130	29368	3919
Number of cities		51	101	150	200	225
SA	avg Distance	449,311 (+ 5,47%)	679,5 (+ 8,02%)	28749 (+ 10,02)	32636,02 (+11,12%)	4224,65 (+7,9%)
	avg Time (sec)	1,934	11,63	33,692	92,628	142,55
TS	avg Distance	428,83 (+ 0,66%)	645,63 (+ 2,64%)	26332,84 (+0,776%)	29755,47 (+1,319%)	3961,03 (+1,073%)
	avg Time (sec)	0,458	1,576	3,578	7,158	8,526
GA	avg Distance	497,86 (+ 16,86%)	769,42 (+ 22,32%)	36558,79 (+40,02%)	44794,19 (+52,52%)	7257,7 (+85,19%)
	avg Time (sec)	51,06	127,19	255,116	274,6	289,92
AC	avg Distance	451,2 (+ 5,9%)	703,34 (+ 11,82%)	29274,04 (+ 12,03%)	33623,95 (+ 14,4%)	4347,61 (+ 10,93%)
	avg Time (sec)	16,32	21,78	34,56	75,95	94,95
Number of iterations		100	100	100	100	100

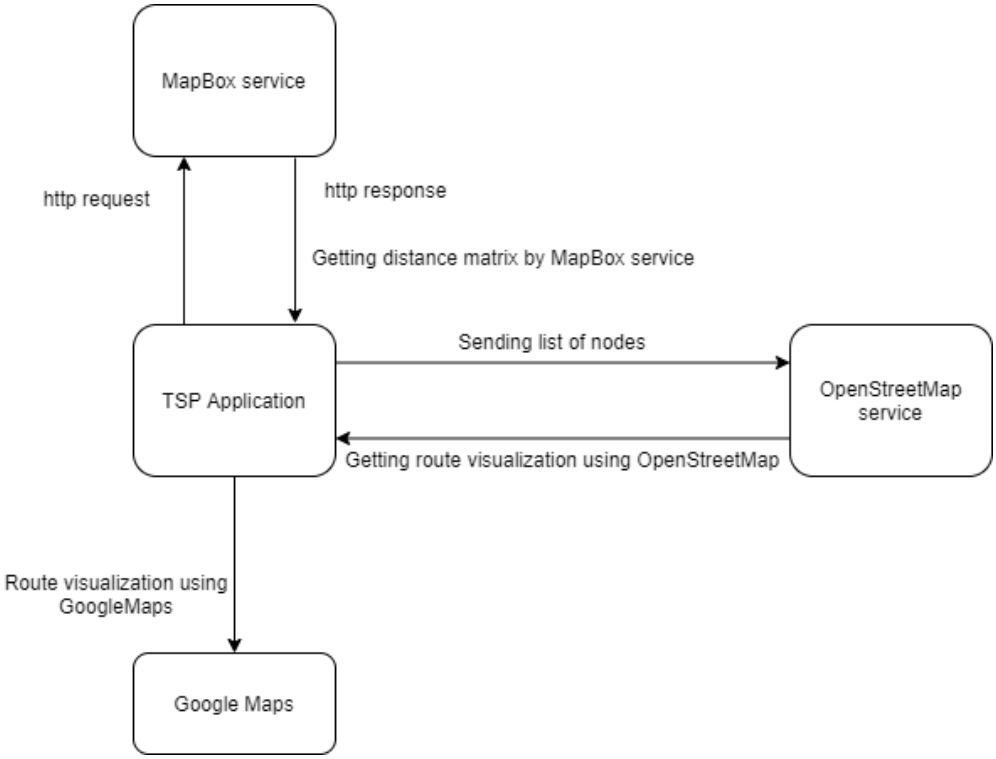
Tabulka 20: Porovnání výsledků činnosti všech algoritmů (2)

Jak je vidět ze srovnávacích tabulek, metaheuristické metody, a to metody Tabu search a Simulated annealing, jsou nesporným lídrem jak v přesnosti nalezení optimální cesty, tak v době trvání činnosti algoritmu. Klasické metody, a to Branch and bound a Dynamic programming, jsou také schopny najít výsledek blízký optimu, ale pouze u souborů s malým počtem měst. Evoluční metody, a to Genetický algoritmus a algoritmus Ant colony, nacházejí poměrně přesný výsledek při práci s malým počtem bodů, avšak při práci s velkým počtem bodů mají tyto metody velkou dočasnou volatilitu a významnou chybu vzhledem k optimální cestě.

5. APLIKACE

V rámci diplomové práce byla vytvořena aplikace, která umožňuje řešit problém obchodního cestujícího s další vizualizací dat v podobě cesty a vytvoření grafu závislosti změny délky cesty na čase. Výsledek fungování algoritmů, a sice konečná vzdálenost a doba trvání činnosti algoritmů se zapisuje do Textboxu. Data se také ukládají do souboru ve formátu txt. Aby bylo možné algoritmus spustit několikrát a získat průměrný výsledek, je možné zvolit počet spuštění algoritmů s dalším zaznamenáním přijatých dat do souboru.

Aplikace může zpracovávat formáty souborů: .txt, .tsp, .kml. Soubory s příponou .kml je formát souboru, který se používá k zobrazení geografických dat v geobrowserech (geoprohlížečích), jako je Google Earth, Mapy Google. KML je založen na standardu XML a používá strukturu založenou na značkách (tagech) s vnořenými prvky a atributy.



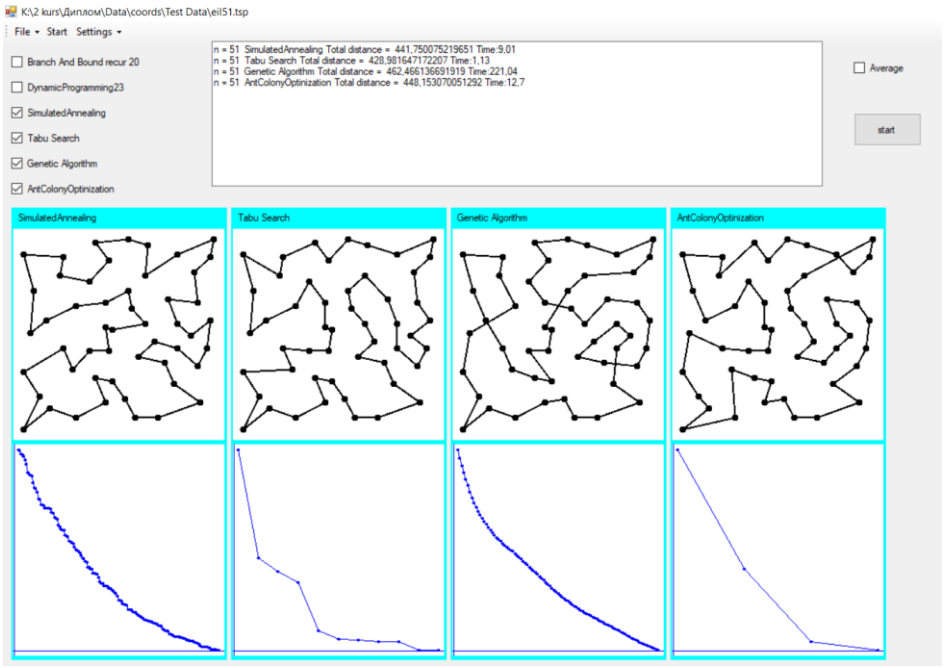
Obrázek 27: Součinnost aplikace se službami pro vizualizaci geodat.

Služba MapBox načte do projektu matici vzdálenosti. Poté začnou fungovat algoritmy realizované v diplomové práci. Výsledkem fungování algoritmů je posloupnost měst, která je nutné navštívit. Pomocí služby OpenStreetMap aplikace přijímá vizualizaci cesty a výsledná vizualizovaná cesta se zobrazí na pictureBox-e aplikace pomocí map GoogleMaps.

Zdrojový kód metody odpovědné za součinnost s externími službami viz Příloha A.

5.1 Uživatelská dokumentace

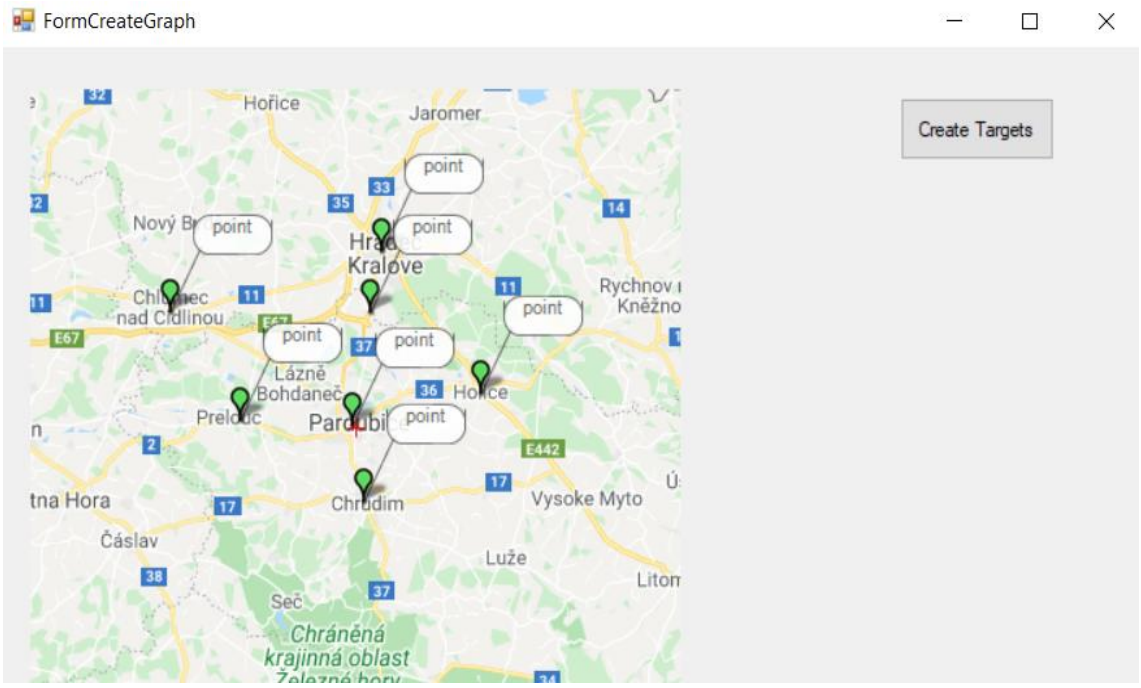
Aplikace poskytuje intuitivní grafické uživatelské rozhraní. Zaškrtnutím požadovaných algoritmů na levé straně formuláře, se uživatel rozhodne, kterou metodu je třeba použít k vyřešení konkrétního problému. Při výběru požadovaných algoritmů se ve spodní části formuláře objeví okna, která slouží k další grafické vizualizaci výsledků fungování algoritmu v podobě cesty a grafu závislosti změny délky cesty na době trvání činnosti algoritmu. Výsledky fungování algoritmu ve formě vzdálenosti nejkratší cesty a času…potřebného k použití metody se zobrazí v Textboxu ve střední části formuláře. Po výběru požadovaných algoritmů je nutné kliknout na tlačítko Start, které je umístěno na horním panelu.



Obrázek 28: Grafické uživatelské rozhraní

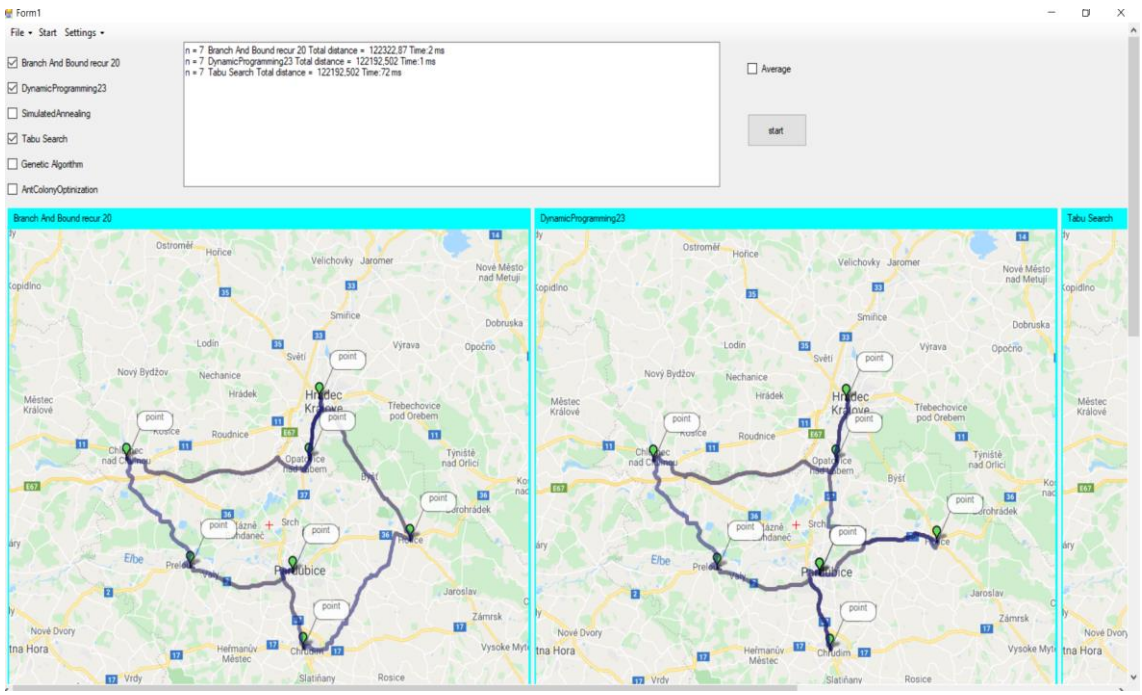
5.2 Možnost práce s mapou

Aplikace poskytuje možnost pracovat s mapou, a to zmapování vlastních bodů a získání výsledků fungování algoritmů s vizualizací na mapě. Za tímto účelem musí uživatel provést následující kroky: File - Create graph from screen. V důsledku těchto kroků aplikace zobrazí nový formulář, do kterého může uživatel přidávat body.



Obrázek 29: Přidání nových bodů na mapu

Po kliknutí na tlačítko Create Targets se vypočítá matice vzdáleností, poté uživatel klikne na Start na hlavním formuláři a obdrží výsledky fungování algoritmů.



Obrázek 30: Vizualizace výsledky algoritmů

5.3 Architektura aplikace

K oddělení aplikačních dat, uživatelského rozhraní a řídicí logiky bylo použito schéma Model-View-Controller (MVC). Projekt je rozdělen do tří hlavních bloků:

- blok Algorithm implementuje řídicí logiku, implementuje všechny algoritmy realizované v projektu. Architektura bloku aplikace popsána v jazyce UML viz Příloha B1.
- blok Data poskytuje data, reaguje na povely bloku Algorithm a mění svůj stav. Architektura bloku aplikace popsána v jazyce UML viz Příloha B2.
- blok View je zodpovědný za zobrazení dat modelu uživateli. Architektura bloku aplikace popsána v jazyce UML viz Příloha B3.

Za účelem minimalizace opakování kódu a zvýšení flexibility aplikační architektury každý algoritmus implementuje rozhraní ITspAlgorithm.

```
public interface ITspAlgorithm
  {
     string GetName();
     int[] GetTour(Graph graph);
     double GetTotalDistanse();
     List<double> GetTotalDistances();
  }
```

Kód 7: Rozhraní pro všechny implementované algoritmy

ZÁVĚR

V této diplomové práci jsou popsány klasické, metaheuristické a evoluční metody řešení problémů diskrétní optimalizace na příkladu problému obchodního cestujícího. Je provedena srovnávací analýza účinnosti implementovaných algoritmů na různých datech. Vzhledem k tomu, že aplikace vytvořená v diplomové práci umožňuje pracovat s reálnými geodaty, byla srovnávací analýza algoritmů provedena rovněž na základě reálných geodat.

Ze získaných výsledků provedené analýzy vyplývá, že kritérium výběru algoritmu pro výpočet cesty bude záviset na počtu měst N:

Je-li $N < 20$, všechny algoritmy prozkoumané v diplomové práci efektivně nachází nejkratší cestu;

Je-li $N > 20$ a $N < 101$, je vhodné hledat cestu pomocí metaheuristických a evolučních metod diskrétní optimalizace;

Je-li $N > 101$, zaručit nalezení optimální cesty v přijatelném čase mohou metaheuristické diskrétní optimalizační metody, konkrétně algoritmus Tabu Search a algoritmus simulovaného žíhání.

Studium vlastností problému obchodního cestujícího umožnilo vyvodit následující závěr: v současné době zůstává aktuální hledání přesných a přibližných metod pro řešení tohoto problému jak z teoretického, tak i praktického hlediska. Tempo moderního života navíc mění postoj člověka k času. Dnes uživatel nerad čeká a hledá způsoby, jak zkrátit čekací dobu a najít optimální řešení v co nejkratším možném čase. To vše svědčí o růstu budoucí potřeby efektivního řešení problému obchodního cestujícího a dalších souvisejících optimalizačních problémů, což by výrazně ušetřilo omezené zdroje organizací.

POUŽITÁ LITERATURA

[APPLEGATE, 2011] APPLEGATE, David. The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics), 2011. ISBN 978-0691129938.

[BONDY, 2005] BONDY, Adrian. Graph theory. – London: Springer, 2005. ISBN 978-1-84628-969-9.

[BRENDON, 1993] BRENDON, Glen. Topology and geometry. New York: Springer-Verlag, 1993. ISBN 978-1-4757-6848-0.

[BRUSCO et al., 2005] BRUSCO, Michael a Stephanie STAHL. Branch-and-Bound Applications in Combinatorial Data Analysis, 2005. ISBN 978-0-387-28810-9.

[BUONTEMPO, 2019] BUONTEMPO, Frances. Genetic Algorithms and Machine Learning for Programmers: Create AI Models and Evolve Solutions (Pragmatic Programmers), 2019. ISBN 978-1680506204.

[COOK, 2012] COOK, William. Po stopách obchodního cestujícího: matematika na hranicích možností. Praha: Argo, 2012. ISBN 978-80-7363-412-4.

[CORMEN, 2009] CORMEN, Thomas. Introduction to Algorithms, 3rd Edition (The MIT Press), 2009. ISBN 978-0262033848.

[DORIGO, 2004] DORIGO , Marco a Thomas STUTZLE. Ant Colony Optimization, 2004. ISBN 9780262042192.

[DOSTÁL, 2003] DOSTÁL, Petr. Optimalizační metody. Kunovice: Evropský polytechnický institut, 2008, 44 s. ISBN 978-80-7314-136-3.

[CHJO MU KHAN, 2010] CHJO MU KHAN, Schedule planning and passenger traffic management using a modeling environment, 2010.

[KOLESNIKOV et al., 2011] KOLESNIKOV A.V., KIRIKOV I.A. Solving the difficult tasks of a traveling salesman using functional hybrid intelligent systems, 2011. ISBN 978-5-902030-88-1.

[KUČERA, 2009] KUČERA, Petr. Metodologie řešení okružního dopravního problému. Praha: ČZU, 2009, 122s.

[MATOUŠEK et al., 2009] MATOUŠEK, Jiří a Jaroslav NEŠETŘIL. Kapitoly z diskrétní matematiky. 4., upr. a dopl. vyd. V Praze: Karolinum, 2009, 442 s. ISBN 978-80-246-1740-4.

[PLOTNIKOV, 1998] PLOTNIKOV, A.D. One Criterion of Existence of a Himiltonian Cycle. Reliable Computing 4, 199–202 (1998).

[SERGIENKO et al., 2003] SERGIENKO I.V., SHILO V.P., Discrete optimization problems: problems, solution methods, research. – Kyiv: 2003. ISBN 966-00-0114-2.

[SESEKIN, 2005] SESEKIN A.N., CHENTSOV A.G. The dynamic programming method in the traveling salesman problem: Ekaterinburg: GOU VPO, 2005.

[SKIENA, 2010] SKIENA Steven. The Algorithm Design Manual (2nd ed.). Springer Science Business Media, 2010. ISBN 1-849-96720-2.

[STRUCHENKOV, 2016] STRUCHENKOV V.I., Discrete optimization. Models, methods, algorithms for solving applied problems. Solon-Press: 2016. ISBN 978-5-91359-181-4.

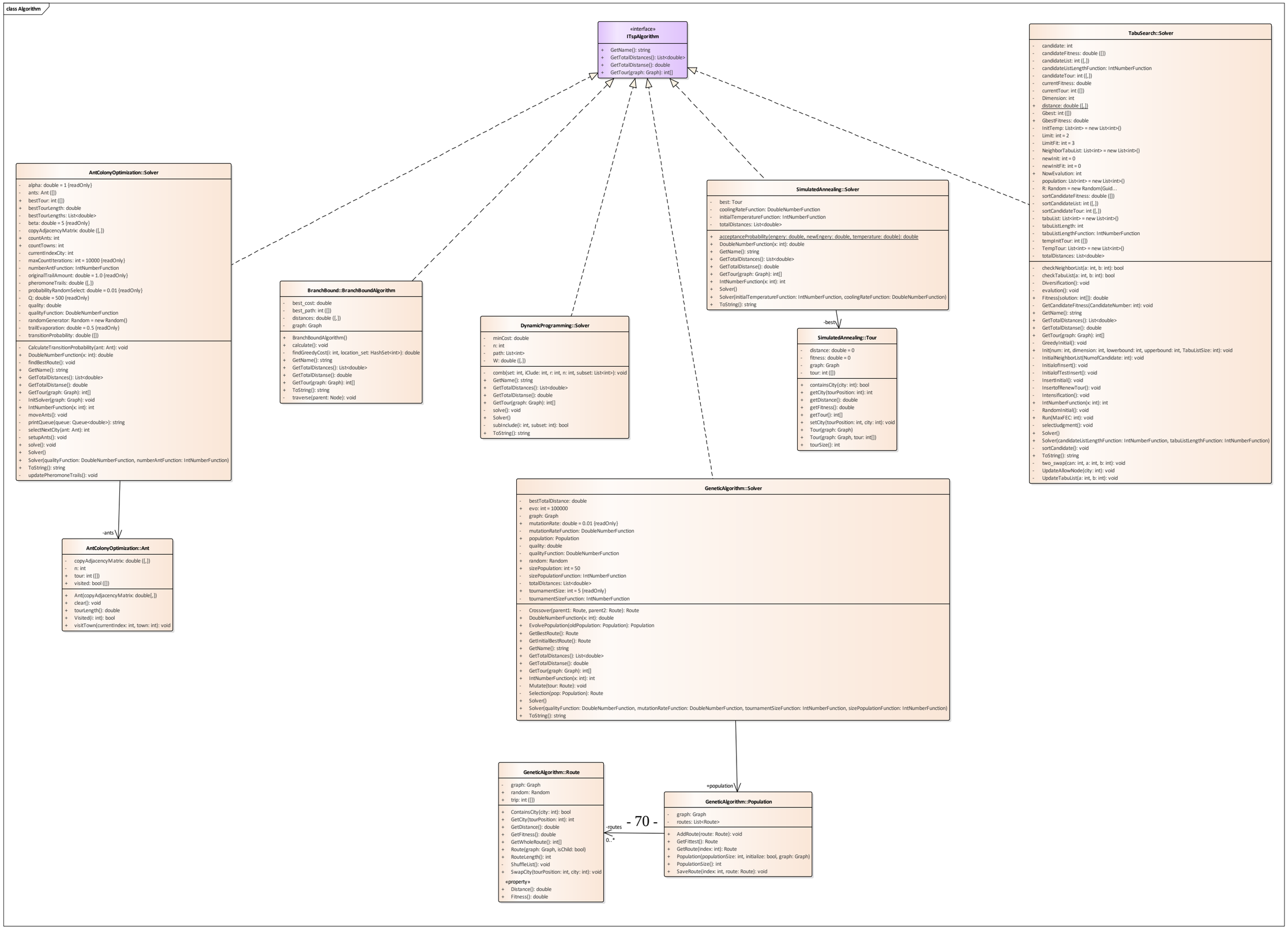
PŘÍLOHY

Příloha A: Zdrojový kód metody odpovědné za součinnost s externími službami

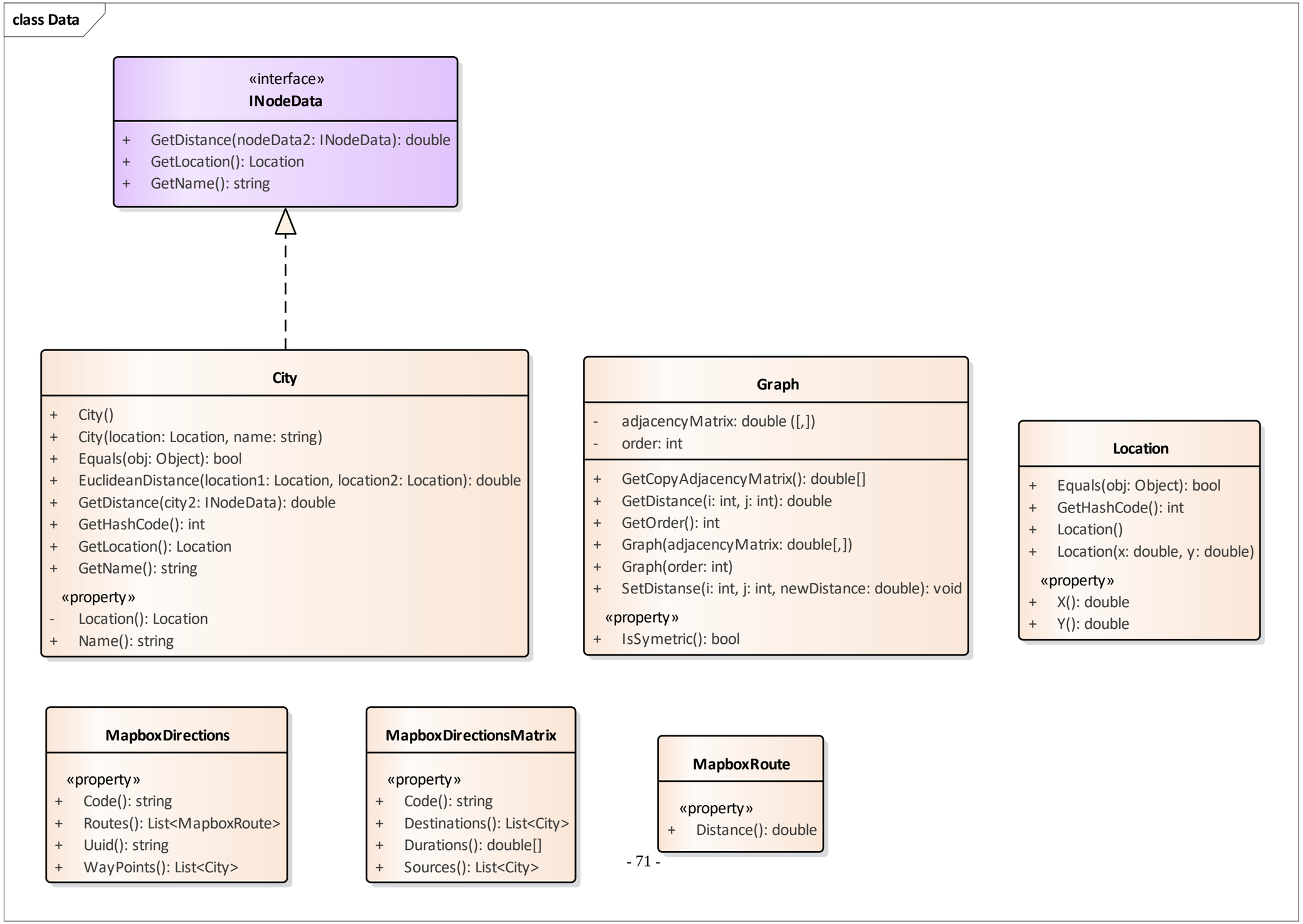
```
public void DrawMap()
    {
      if (nodes == null)
      {
        return;
      }

      OverlayRoutes.Clear();
      OverlayMarkers.Clear();
      PointLatLng point1;
      PointLatLng point2;
      Data.Location location1;
      Data.Location location2;
      for (int i = 0; i < nodes.Length; i++)
      {
        location1 = nodes[i % nodes.Length].GetLocation();
        location2 = nodes[(i + 1) % nodes.Length].GetLocation();
        point1 = new PointLatLng(location1.Y, location1.X);
        point2 = new PointLatLng(location2.Y, location2.X);
        MapRoute route2 = GMapProviders.OpenStreetMap.GetRoute(point1, point2, false, false, 10);
        GMapRoute myRoute2 = new GMapRoute(route2.Points, "route1");
        OverlayRoutes.Routes.Add(myRoute2);
        GMapMarker marker = new GMarkerGoogle(point1, GMarkerGoogleType.green_small);
        marker.ToolTipMode = MarkerTooltipMode.Always;
        marker.ToolTipText = nodes[i].GetName();
        OverlayMarkers.Markers.Add(marker);
      }
      map.ZoomAndCenterRoutes(OverlayRoutes.Id);
      map.Invalidate();
    }
```

Příloha B1: Architektura bloku Algorithm v UML



Příloha B2: Architektura bloku Data v UML



Příloha B3: Architektura bloku View v UML