VYSOKÁ ŠKOLA EKONOMICKÁ V PRAZE

FAKULTA INFORMATIKY A STATISTIKY

DIPLOMOVÁ PRÁCE

2010 Michal Škopek

VYSOKÁ ŠKOLA EKONOMICKÁ V PRAZE

FAKULTA INFORMATIKY A STATISTIKY

Název diplomové práce:

Problém obchodního cestujícího a metoda GENIUS

Autor: **Michal Škopek**

Katedra: **Katedra ekonometrie**

Obor: **Matematické metody v Ekonomii**

Vedoucí práce: **prof. RNDr. Jan Pelikán, CSc.**

Prohlášení:

Prohlašuji, že jsem diplomovou práci na téma „Problém obchodního cestujícího a metoda GENIUS“ zpracoval samostatně. Veškerou použitou literaturu a další podkladové materiály uvádím v seznamu použité literatury.

V Praze dne 16. srpna 2010 .....

Michal Škopek

Poděkování:

Rád bych na tomto místě poděkoval prof. RNDr. Janu Pelikánovi, CSc. za vedení mé diplomové práce a za podnětné návrhy, které ji obohatily.

Abstrakt

Název práce: Problém obchodního cestujícího a metoda GENIUS
Autor: Michal Škopek
Katedra: Katedra ekonometrie
Vedoucí práce: prof. RNDr. Jan Pelikán, CSc.

Cílem diplomové práce je vysvětlit Problém obchodního cestujícího a vytvořit program, který bude počítat speciální metodu GENIUS. Problém obchodního cestujícího je popsán z několika hledisek. Nejprve z hlediska historického k objasnění souvislostí s určitými metodami a následně je popsán z hlediska výpočetních metod. Pro popis těchto metod byly vybrány zástupci jak exaktních metod tak i heuristických. Stěžejní částí diplomové práce je popis heuristiky GENIUS, ke které je vytvořen speciální počítačový program. Tento program pracuje nejprve s algoritmem GENI a následně s post-optimalizačním algoritmem US. Program je popsán z uživatelského pohledu a je k němu vytvořen manuál. Program je otestován na dvou základních příkladech. Výsledky, dané výpočtem pomocí programu pracujícím s heuristikou GENIUS, jsou srovnány s výsledky získanými pomocí exaktních algoritmů.

Klíčová slova: GENIUS, heuristiky, Problém obchodního cestujícího, TSP

Abstract

Title: Travelling Salesman Problem and method GENIUS
Author: Michal Škopek
Department: Department of Econometrics
Supervisor: prof. RNDr. Jan Pelikán, CSc.

The target of this thesis is to explain the Travelling Salesman Problem and also create a special program, which will be able to make calculations using the heuristics GENIUS. The Travelling Salesman Problem will be described from two different points of view. The first one is the historical description of the idea of the Travelling Salesman Problem and later will be the problem will be described with some of the very wide number of the calculation methods. For the explanation of the methods, in the thesis there has been chosen some of the algorithms which belong to that methods. The heuristics and also the exact algorithms will be explained. The focus of this thesis is on the heuristics called GENIUS and also in the creation of the program which can calculate it. The program works first with the GENI algorithm and after that it works with US post-optimization algorithm. The program will be described from the point of view of the user and the manual will be written as well. The program will be tested on two different examples and will be compared with the exact algorithm.

Keywords: GENIUS, heuristics, Travelling Salesman Problem, TSP

Obsah

Úvod

Problém obchodního cestujícího nás provází na každém kroku. Není spojen pouze s pracovním procesem, ale v konečném důsledku se dotýká téměř všech našich činností. Nechceme neefektivně ztrácet čas, například špatným naplánováním cesty, neefektivními zajížďkami, vracením se na již navštívená místa. Hledáme tedy způsoby jak nejlépe naplánovat a jak nejlépe náš čas investovat. Znamená to, že se snažíme vykonat nějakou činnost v kratším čase nebo s menším úsilím než dříve. Zajímalo mě tedy, jak pomocí použití matematických metod dojde k efektivnímu využití především pracovního času. Z vlastní zkušenosti vím, že v současné době tento problém řada firem neřeší, často nevyhodnocuje efektivnost nejen při převozu zboží ale i při pohybu svých zaměstnanců.

V této práci je rozpracována jedna z úloh Okružních problémů (obecně zabývají způsoby a možnostmi, jak přepravovat lidi, zboží nebo jakékoliv různé komodity do určitých bodů, lokalit nebo měst) a to Problémem obchodního cestujícího (Travelling Salesman Problem). Úkolem bude projet všechna města tak, aby vzdálenost, která bude ujeta, byla co nejmenší. Pod Problémem obchodního cestujícího je představen člověk (obchodní zástupce), který má za úkol objet několik měst v určitém regionu. Tato města jsou od sebe různě vzdálená. Obchodní zástupce tak musí objet všechna města tak, aby celková vzdálenost byla co nejkratší. Obchodní zástupce vyjíždí na svou trasu vždy z jednoho města, které se nazývá výchozí město. Toto výchozí město je například to, kde je továrna nebo kanceláře firmy. Z tohoto města obchodní cestující vyjede na svou okružní cestu, objede všechna města a vrací se zpátky do města, ze kterého původně vyjel. Úloha je však komplikována faktem, že každým městem smí projet právě jednou. Nesmí se tak stát, že by obchodní zástupce více než jednou projížděl některým z měst, které má vybrané.

Problém obchodního cestujícího má velké množství využití a existují různé variace úlohy. Typické úlohy jsou například [9]: Problém obchodního cestujícího s časovými okny (časové omezení úlohy z hlediska hodin strávených v určitých městech), Vehicle Routing Problem (modifikovaná úloha obchodního cestujícího s několika trasami a více než jedním vozidlem), síťování počítačů (propojení všech počítačí přes ethernet síť), výroba mobilních telefonů (ovládání robota) a kombinatorická analýza dat (přerovnávání řádkou a sloupců matic dat). Takových úloh je celá řada a často jsou mezi sebou propojeny a navazují na sebe i při řešení.

Formulace Problému obchodního cestujícího uvedená na začátku práce zní velmi jednoduše. Její obtížnost se projevuje až při jejím matematickém vyjádření a následným hledáním optimálního řešení. Proto patří do skupiny úloh NP-Complete [15]. U těchto úloh se dají velmi obtížně použít obecné algoritmy výpočtu úloh lineárního programování. Dokonce lze říci, že ve většině případů se musí upravovat známé algoritmy pomocí dalších omezujících podmínek a tvoří se tak velmi složité a velmi dlouhé výpočty. V době, kdy

se poprvé řešili, nebyla dostupná tak výkonná technika, jaká je dnes. Proto se začali vyhledávat možnosti, jak tyto algoritmy zjednodušit. Dnes se algoritmy pro výpočet Problému obchodního cestujícího rozdělují do dvou základních typů. Prvním typem jsou exaktní algoritmy, které řeší velké množství úloh, ale jsou časově a výpočetně náročné. Druhým typem jsou heuristiky, které sice nemusí dát optimální řešení, ale mohou nás optimálnímu řešení velmi přiblížit. Jejich výhoda je hlavně v kratším výpočetním čase. Rozsáhlé úlohy lze vyřešit za zlomek času při srovnání s exaktními algoritmy.

Exaktní algoritmy jako takové nejsou velmi často využívané. A to i přes skutečnost, že jejich největší pozitivum, nalezení optimálního řešení úlohy, převažuje nad jejich negativem, dlouhým výpočetním časem. V dalších kapitolách této práce uvádím základní popis dvou metod pro výpočet, a to metody větvení a mezí [5] a metody řezných nadrovin [14].

Dále se budu zabývat heuristikami. Tyto metody podávají výsledky blízké optimum za zlomek výpočetního času. Typů heuristik lze v dnešní době najít velké množství, každá z nich vychází z jiných předpokladů úlohy. Rozdělují se do třech základních skupin podle jejich aplikace na heuristiky:

1. heuristiky, které spojují body na základě přidávání jednoho bodu v každém kroku
2. heuristiky, které upravují již vytvořený cyklus na základě záměn bodů
3. kompozitní algoritmy, které kombinují dvě předešlé heuristiky.

Heuristiky popíšu, uvedu jejich algoritmy a na závěr zmíním jejich výhody a nevýhody. Stěžejní bude heuristika GENIUS (Generalized Insertion Procedure Unstringing and Stringing). Heuristika GENIUS [8] je tvořená ze dvou částí. První je GENI, což je základní algoritmus na vytvoření cyklu a druhá je US, což je post-optimalizace, která se v každém kroku snaží zkrátit celkovou vzdálenost odebíráním a následným přidáváním bodů do cyklu. O této heuristice následně uvedu matematické formulace a grafy, na kterých ukážu, jakým způsobem tato heuristika řeší úlohy.

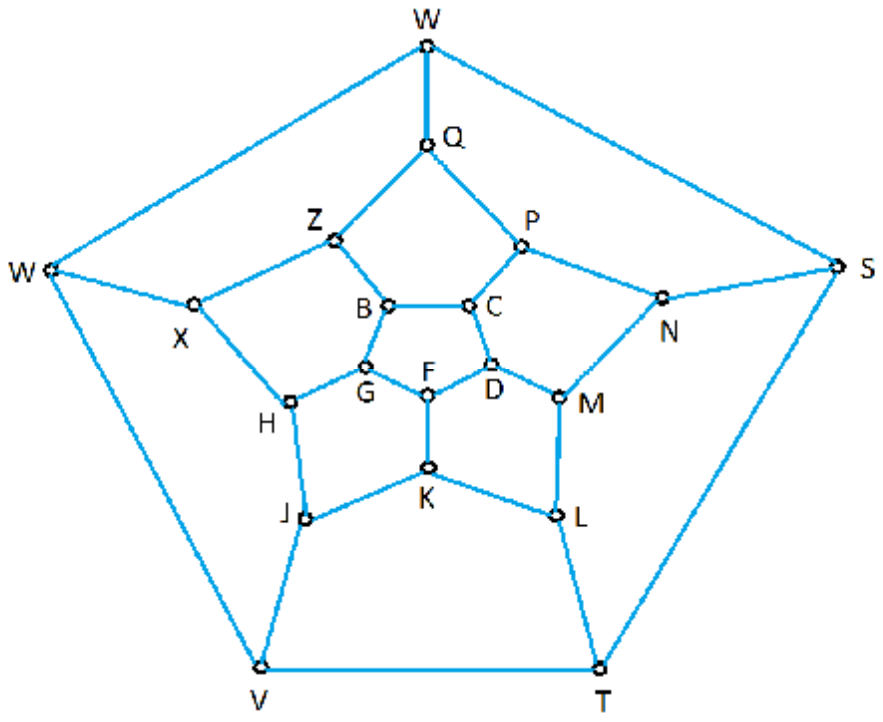
Ve druhé části práce se zaměřím na praktické využití heuristiky GENIUS. Praktické využití budu demonstrovat na programu vytvořeném v programu napsaném v C++ a za využití programovacích nástrojů pro tento jazyk. Zaměřím se na důvody volby tohoto programového prostředí, popíšu grafické rozhraní a ovládání, a nakonec se zaměřím na testování programu pomocí výpočtů tras na dvou reálných příkladech, pro které jsem vybral data z atlasu České Republiky. Výsledky pak budu srovnávat s výsledky exaktního algoritmu, který si vypočítám pomocí programu LINGO, a tím zjistím efektivnost výpočetního algoritmu a správné naprogramování.

1 Historie Problému obchodního cestujícího

Historie Problému obchodního cestujícího sahá až do 18. století [9]. V tomto období se začaly objevovat první náznaky Problému obchodního cestujícího. Především se jednalo o počátky teorie grafů, ze které tento problém vychází. Teorie grafů je součástí diskrétní matematiky. Jako první jí uvedl v roce 1736 Leonhard Euler, který publikoval práci „Solutio problemas ad geometriam situs pertinensis“. V této práci se zmínil o modelu sedmi mostů ve městě Královec. Tento model bude popsán v následující kapitole Teorie grafů. Další a neméně známá úloha, kterou řešil L. Euler, je „Úloha jezdce“. Tato úloha je založena na konceptu šachovnice o 64 polích. Úkolem jezdce (koněm) je projít všechna pole právě jednou a vrátit se do původního místa. Jezdec tak měl jedno pole (uzel) označené jako výchozí a druhé pole (uzel) jako koncové. Tyto dva uzle byly spojeny hranou právě tehdy, když se jezdec přesunul z jednoho pole do druhého. Jezdec takovýmto způsobem musel objet všechna pole na šachovnici a nakonec se musel vrátit do výchozího pole. V úloze tedy šlo o nalezení Hamiltonova cyklu, který je popsán v následující kapitole. Lze tedy říci, že tato úloha byla prvním krokem k Problému obchodního cestujícího.

Po uvedeních těchto myšlenek Leonharda Eulera, se v matematice neobjevila žádná další zmínka o Teorii grafů více než sto let. Až v polovině 19. století se objevily náznaky obnovení této teorie. Přispěl k tomu G. Kirschhoff v roce 1847, tím, že se zabýval výpočtem v té době neznámých proudů v elektrických sítích. Jeho přínosem však bylo nejen řešení úloh pomocí soustav algebraických rovnic, ale přispěl i k prvním zmínkám o teorii grafů a stavbě stromů řešení úlohy.

Další důležitý krok v teorii grafů a Problému obchodního cestujícího udělal sir William Roawn Hamilton, který vytvořil hru zvanou „Icosian Game“, někdy také nazývanou „Hamilton´s Icosian“. Hra vychází z několika teorií, z klasické geometrie a ze studie dvanáctistěnu a měla za úkol, co nejefektivněji vytvořil cyklus přes 20 bodů použitím specifických hran. Hamilton byl tak inspirovaný využitím geometrie a algebry, že vytvořil strukturu pro uskutečnění spojení mezi 20 ti body. Tyto body mohly být spojeny pouze podél stěn dvanáctistěnu. Jako pomoc při řešení vytvořil obrázek uvedený níže.



Obrázek 1-1: Hamilton´s Icosian

Na obrázku 1-1, spojnice představují hrany a kruhy přestavují body neboli vrcholy. Úkolem této úlohy je spojit všechny body tak, aby ve výsledku tvořili jeden cyklus. Na tomto příkladě se může dokázat, že při vytvoření prvního cyklu z libovolných pěti bodů, je možné vždy spojit zbylých 15 bodů. Více informací o této úloze lze najít v knize „Graph Theory 1736-1936 od N. L. Biggse, E. K. Lloyda a R. J. Wilsona, Clarendon Press, Oxford, 1976“.

Ve druhé polovině 19. století (1878) se díky Jamesi Josephovi Sylvestrovi podařilo objevit teorii grafů, resp. graf jak ho známe dnes. Nejznámější úlohou, kterou se v 19. století matematici zabývali, byl tzv. „problém čtyř barev“, který je podrobněji popsán v Teorii grafů, v následující kapitole.

Po Jamesi Josephovi Sylvestrovi, se matematici začali více zabývat Teorií grafů a v druhé polovině 20. století vycházeli knihy specializované na tuto teorii. Důležité je zmínit knihu francouzského matematika Clauda Bergeho, který napsal (1958) „Théorie des Graphes et ses Applications“ a knihu norského matematika Oysteina Oreho (1962) „Theory of Graphs“ a neposledně knihu „Graph Theory“ od amerického matematika Franka Hararyho.

Jak bylo na začátku kapitoly řečeno, Problém obchodního cestujícího vychází z Teorie grafů. První zmínka byla již v 18. století a to Leonhardem Eulerem, který řešil „Úlohu jezdce“. Od té doby nebylo moc informací o pokračování práce, kterou začal Euler. Až v roce 1920 byla publikována studie zvaná „Botenproblem“ matematikem a ekonomem Karlem Mengerem. Později se ve svých studiích zmínili o této úloze Hassler Whitney a Merrill Flood z univerzity v Princetonu, kteří využili poznatky při řešení optimalizace v zemědělství a při

výzkumu pro firmu RAND. Detailní popis úlohy a její propojení mezi modely, které publikovali Karl Menger a Hassler Whitney, je možné najít v knize, kterou napsal Alexander Schrijver – „On the history of combinatiorial optimization till 1960“.

Zlom však přišel s ekonomy Georgem Dantzigem, Ray Fulkersonem a Selmerem Johnsonem, kteří v roce 1954 publikovali knihu s metodami na řešení Úloh obchodního cestujícího a ukázali možnost řešení úlohy se 49 městy [5]. Tato úloha byla vytvořena vhodným zvolením jednoho města z každého státu v USA a přidáním hlavního města, Washington D. C.. Mezi každým městem byly změřeny vzdálenosti, ty se měřili po silničních cestách. Úloha však byla následně snížena na 42 měst, byla vypuštěna města na západním pobřeží. Zvláštností úlohy je to, že i při vypuštění těchto měst, nejkratší cyklus byl vypočten tak, že vedl i přes tato velká vypuštěná města. Podařilo se tak vypočítat řešení jak pro zkrácenou úlohu se 42 městy, tak i pro úlohu, která již obsahuje všech 49 hlavních měst Spojených Států Amerických.

Pokud se vezmou v úvahu všechny vzdálenosti, které jsou lineární mezi každou dvojicí měst, máme k dispozici celkem 861 hran pro redukovaný model se 42 městy. Počet hran se dá jednoduše získat z výpočtu $n(n-1)/2$, kde písmeno n udává počet měst. Tyto hrany byly vloženy do matice vzdáleností vypadající stejně jako ta, kterou najdeme například v Autoatlase České Republiky. Na základě matice vzdáleností, vytvořili stejně velkou matici, do které zapisovali jedničky a nuly. Jednička v případě, že bude vytvořeno spojení (hrana) mezi jedním a druhým městem, nula v případě, že nebude tato hrana využita. Pro výpočet této úlohy byly využity metody lineárního programování a celkem bylo pro výpočet využito čtyř kroků, které pomohly vyřešit tuto úlohu:

1. Jako základní předpoklad se bral neorientovaný graf, který zjednodušil popis cyklů a zkrátil výpočetní čas úlohy.
2. Cesta není popsaná úplným výčtem lineárních omezení, ale za lineární byly pouze považovány vzdálenosti mezi městy. To nám zajistí minimalizování celkové vzdálenosti a zjednodušení výpočtu délky mezi dvěma body. Tento postup vychází z předpokladů I. Hellera a H. Kuhna, kteří definovali lineární vzdálenosti, díky kterým je úloha tvořena jednoduchými a nikoliv složitými soustavami rovnic.
3. K urychlení celkového výpočtu a v některých případech může dojít k přidání nových omezujících podmínek nebo využití subjektivních změn cyklu pomocí vizuální kontroly mapy.
4. Jakmile jsou spojena všechna města, řešení se blíží optimálnímu a následně při využití kombinatorických metod dojde k eliminaci vše cyklů, které nebyly eliminovány předchozími podmínkami. Počet těchto cyklů může být velmi nízký vzhledem k složitému vzájemnému propojení omezení.

Abychom zamezili tvorbě cyklů, tato metoda obsahuje i určité lineární podmínky. Tyto podmínky zajišťují, aby při výpočtu nedocházelo k zacyklení, a aby se nevracelo několikrát

do jednoho bodu. Takto byli poprvé definované podmínky pro výpočet úlohy se 42 městy. Následně podle výzkumu a testech G. Dantziga, R. Fulkersona a S. Johnsona, se daly využít tyto podmínky i pro kalkulaci úlohy obsahující 49 měst. Více informací lze čerpat z díla „Solution of large-scale travelling-salesman problem“ napsaného výše uvedenými autory v Rand Corporation, Santa Monica, California, 9. 8. 1954.

V roce 1955 G. Morton a A. H. Land publikovali dílo, které obsahovalo jejich vlastní verzi metody výpočtu zvanou 3-Opt. O rok později Merill M. Flood publikoval dílo s názvem „The travelling salesman problem“ v Operation Research 4, strana 61-75. V tomto díle Flood popisoval heuristické metody, které mohly vypočítat úlohu ve velmi krátkém čase. Mezi nejvýznamnější patří Metoda nejbližšího souseda a 2-Opt.

V 50. letech 20. století rostl zájem o využití výpočetní techniky pro řešení matematických úloh. V roce 1958 F. Bocks využil dílo „An algorithm for solving travelling salesman and related network optimisation problems“ v Operation Reserach Society of America. V tomto díle byl popsán 3-Opt algoritmus a schéma pro výpočet optimálního řešení. F. Bocks využil informace z tohoto díla, vytvořil si matici s 10 městy a vypočítal úlohu pomocí počítače IBM 650.

V dalších letech byly tyto heuristiky hojně využívány a testována na různých počítačích. V roce 1964 byly tyto heuristiky R. L. Kargem a G. L. Thompsonem využity pro výpočet úlohy s 57 městy a rok poté Shen Lin využil tyto metody pro výpočet úlohy obsahující na tu dobu neuvěřitelných 105 měst.

V 70. letech R. M. Karp a M. Held publikovali dílo „The travelling salesman problem and minimum spanning trees“, ve kterém ukázali možnosti výpočtu Problému obchodního cestujícího pomocí 1-tree relaxace, kde navíc využili možnost přiřadit váhy jednotlivým uzlům. Autoři využili tuto metodu pro řešení úlohy 42 měst (Dantzig, Fulkerson, Johnson, 1954), 57 měst (Karg a Thomspon) a byli úspěšní.

V roce 1976 se uskutečnil další pokrok ve výpočtech. P. Miliotis využil možnosti celočíselného lineárního programování a následně na Problém obchodního cestujícího aplikoval metodu větvení a mezí, která využívala omezujících podmínek s nerovnostmi v podgrafech.

V následujícím období pokračoval vývoj heuristik a možnosti využití i další metody na výpočet celočíselných úloh, především metody řezných nadrovin a lineárních relaxací. Jeden z matematiků, který využil tyto poznatky, byl výše uvedený P. Miliotis ve své práci „Using cutting planes to solve the symmetric travelling salesman problem“, Mathematical Programming 15, 177-188.

Na konci 20. století mezi nejvýznamnější díla patřilo „Solution of large scale symmetric travelling salesman problems“ od autorů M. Grötschela a O. Hollanda.

Na základě vývoje informačních technologií a způsobů výpočtu, docházelo k možnosti řešení sofistikovanějších a tím pádem složitějších úloh. V tabulce lze pozorovat vývoj mezi lety 1954 a 2004, resp. od řešení úlohy se 49 městy až po velmi složité úlohy o velikosti 24 978 měst.

Rok	Výzkumný tým	Velikost projektu	Název projektu
1954	G. Dantzig, R. Fulkerson, and S. Johnson	49 měst	dantzig42
1971	M. Held and R.M. Karp	64 měst	64 náhodných bodů
1975	P.M. Camerini, L. Fratta, and F. Maffioli	67 měst	67 náhodných bodů
1977	M. Grötschel	120 měst	gr120
1980	H. Crowder and M.W. Padberg	318 měst	lin318
1987	M. Padberg and G. Rinaldi	532 měst	att532
1987	M. Grötschel and O. Holland	666 měst	gr666
1987	M. Padberg and G. Rinaldi	2,392 měst	pr2392
1994	D. Applegate, R. Bixby, V. Chvátal, and W. Cook	7,397 měst	pla7397
1998	D. Applegate, R. Bixby, V. Chvátal, and W. Cook	13,509 měst	usa13509
2001	D. Applegate, R. Bixby, V. Chvátal, and W. Cook	15,112 měst	d15112
2004	D. Applegate, R. Bixby, V. Chvátal, W. Cook and K. Helsgaun	24,978 měst	sw24798

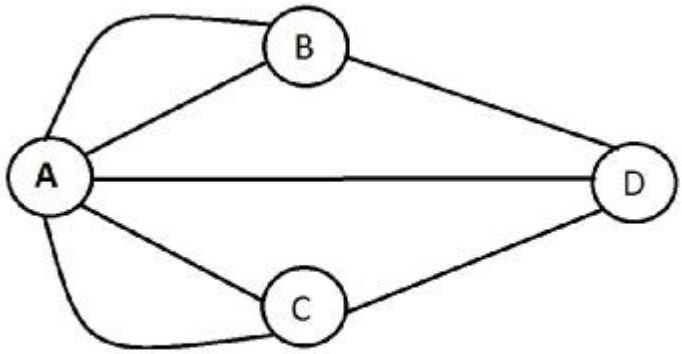
Tabulka 1-1: Milníky výpočtu úlohy obchodního cestujícího [3]

2 Teorie grafů

Teorie grafů je vědní disciplína, která zkoumá speciální matematické či technologické struktury, které se nazývají grafy. Graf je definován jako dvojice množin $G = (V, A)$, kde V jsou vrcholy (uzly, města) a A jsou hrany (spojnice). Hrana vždy spojuje dva uzly.

V roce 1736 matematik Leonhard Euler formuloval a řešil úlohu sedmi mostů ve městě Královec. Město bylo rozděleno řekou, rozkládalo se tedy na dvou březích a uprostřed řeky Pregel byly dva ostrovy. Město se tedy rozdělilo na čtyři samostatné části, které byly propojeny sedmi mosty. Euler měl navrhnout, jak projít všechny čtyři části města tak, aby se každým mostem prošlo právě jednou. Euler se snažil úlohu vyřešit a nalézt optimální řešení. Bohužel Euler nebyl při hledání řešení úspěšný a tak úlohu definoval jako neřešitelnou a formuloval obecné podmínky pro řešení úloh tohoto typu.

Úlohu také definoval graficky, kdy části města označil A, B, C, D a mosty označil jako spojnice těchto bodů.



Graf 2-1: Mosty ve městě Královec

Takto zakreslenou úlohu nazýváme graf, body (A, B, C, D) jsou uzly a spojnice mezi uzly jsou hrany.

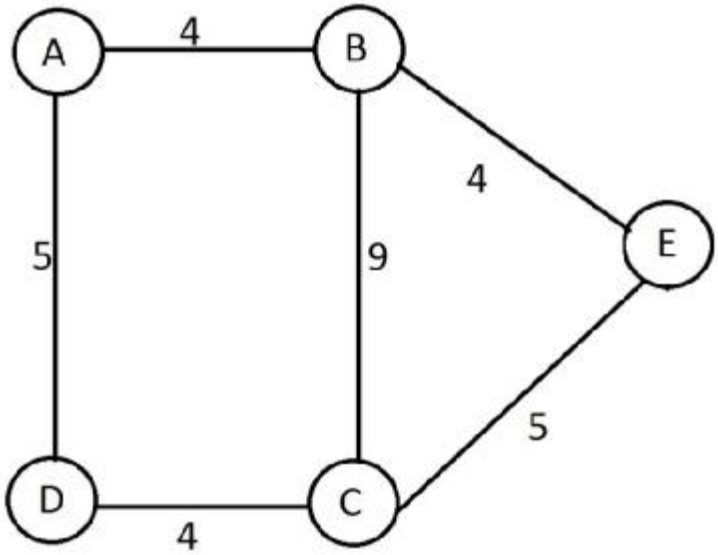
Na teorii grafů je velmi zajímavý fakt, že i přestože Euler definoval základy teorie grafů, matematici neměli velký zájem tuto teorii jako o vědní disciplínu. K této teorii se vrátili o více než 100 let později. Nakonec se několik matematiků pokusilo tuto teorii znovu obnovit. Jednalo se o Gustava Kirchhoffa, který využil vlastnosti této teorie pro výpočty napětí a proudu v elektrických obvodech a Francise Guthrieho, který se zabýval především problémem čtyř barev. V problému čtyř barev se snažil ověřit, že každou mapu lze nabarvit

čtyřmi barvami tak, aby žádný ze sousedních států neměl stejnou barvu. Bohužel, se mu to nepodařilo ověřit a tak tento problém trápil matematiky velmi dlouhou dobu. Až s vývojem moderních výpočetních technologií v roce 1976 se podařilo problém čtyř barev vyřešit. Úspěch s řešením patřil dvěma osobám, Kennethovi Applovi a Wolfgangovi Hakenovi. Za datum vzniku teorie grafů jako samostatné vědní disciplíny můžeme považovat až rok 1936, kdy vyšla kniha s názvem „Teorie konečných a nekonečných grafů“ od D. Königa.

2.1 Dělení grafů

2.1.1 Neorientovaný graf a úloha obchodního cestujícího

Neorientovaný konečný graf je definován jako graf, který se skládá z hran a uzlů, $G = (V, A)$. V je množina všech uzlů $V = \{v_1, v_2, \dots, v_n\}$ a A je množina všech hran $A = \{a_{ij}\}$. Množina hran je podmnožinou všech možných dvojic uzlů v grafu $\{v_i, v_j\}$, $A \subset V\ x\ V$. U hran není přesně definovaný směr, takže na základě výpočtů lze vybrat směr z bodu A do bodu B nebo i opačný.



Graf 2-1: Neorientovaný graf

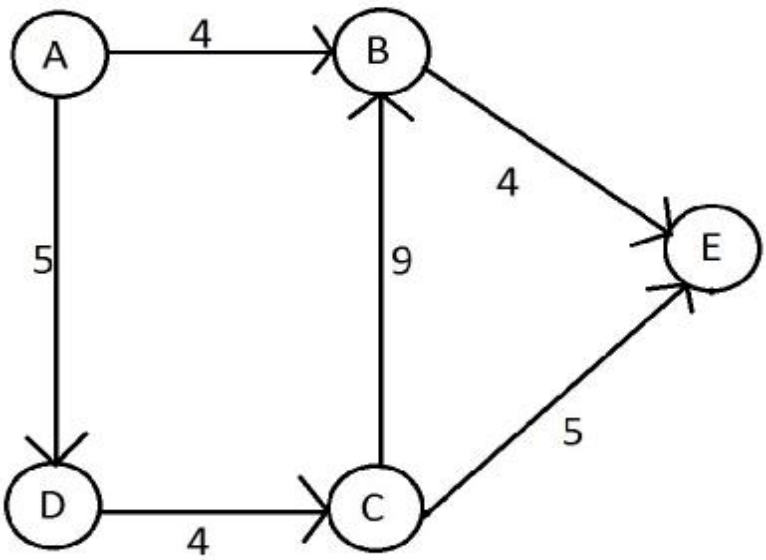
Zobecnění definice neorientovaného grafu viz. [6] pro úlohu obchodního cestujícího:

1. Stupeň uzlu $\deg(v)$ je roven počtu hran, které tento uzel obsahuje.
2. Regulární graf je takový, ve kterém mají všechny uzly stejný stupeň $\deg(v)$.
3. Podgraf grafu G je takový graf, který obsahuje část uzlů a hran původního grafu. Mějme výchozí graf $G = (V, A)$ a graf $G' = (V', A')$, pak platí $V' \subset V, A' \subset A$. V případě, že by v množině A´ ležely všechny hrany grafu G, které mají oba uzly v množině V´, jedná se o úplný podgraf. Úplný podgraf je určen jednoznačně množinou svých uzlů.

4. Sled je možné definovat tak, že máme posloupnost uzlů $v_1, v_2, \dots, , v_k$ takovou, že dva po sobě následující uzly(v_i, v_{i+1}) jsou spojeny hranou.
5. Cesta je takový sled, který prochází různými uzly.
6. Cyklus je uzavřená cesta mezi všemi uzly.
7. Acyklický graf je takový graf, který neobsahuje cykly.
8. Souvislý graf je takový, kdy mezi každými dvěma uzly existuje cesta, která je spojuje.
9. Strom je graf souvislý a acyklický. Pokud odstraníme jakoukoliv hranu, stane se z něj graf nesouvislý.
10. Úplný graf je graf, ve kterém je každá dvojice uzlů spojena hranou. Pokud bychom přidali hranu, vytvořily by se tak paralelní hrany nebo smyčky.
11. Graf je hranově ohodnocený. Každé hraně je tak přiděleno číslo, které může vyjadřovat například vzdálenost nebo cenu.
12. Hamiltonův cyklus je základní kámen pro úlohu obchodního cestujícího. Hamiltonův cyklus je takový cyklus, který pouze v souvislém grafu prochází všemi uzly. Tento cyklus prochází každým uzlem právě jednou.

2.1.2 Orientovaný graf a úloha obchodního cestujícího

Orientovaný graf vychází z definice neorientovaného grafu, která je však modifikována. Máme množinu uzlů a hran, $G = (V, A)$. V je množina uzlů a A je množina uspořádaných dvojic uzlů $\{v_i, v_j\}$. Hrany jsou vždy orientovaný od počátečního uzlu k finálnímu, kde v uspořádané dvojici je první uzel počátečný a druhý koncový. V grafu se vždy směry zobrazují šipkou.



Graf 2-3: Orientovaný graf

Vlastnosti orientovaného grafu, popsané v [6], jsou stejné jako u neorientovaného grafu až na následující výjimky:

1. Orientovaný graf má dva různé typy stupně hran. $\deg^+(v)$ a $\deg^-(v)$.
2. Orientovaný sled z uzlu v_1 do uzlu v_2 je konečná posloupnost hran (v_1, v_2), $(v_2, v_3), \dots, (v_{k-1}, v_k)$ taková, že dvě po sobě následující orientované hrany mají společný uzel.
3. Cyklus je uzavřená cesta mezi všemi uzly s orientovanými hranami.
4. Pro řešení problému obchodního cestujícího by platilo $\deg^+(v) = \deg^-(v) = 1$. Takovou podmínkou zajistíme, že do každého uzlu bude vstupovat právě jedna hrana a vystupovat také právě jedna hrana - tím vytvoříme Hamiltonův cyklus.

3 Typy úloh

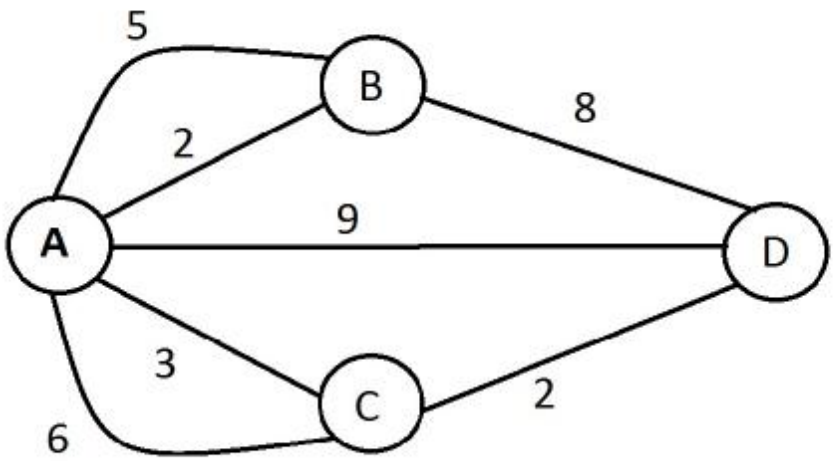
U obchodního cestujícího si na základě teorie grafů můžu určit dva různé typy úloh. Každá z nich má svá specifika a různé základní matice vzdáleností. Typy úloh se dělí na symetrické a asymetrické. Symetrii úloh lze ukázat na základní matici vzdáleností.

	Město 1	Město 2	Město 3	Město 4
Město 1	d_{11}	d_{12}	d_{13}	d_{14}
Město 2	d_{21}	d_{22}	d_{23}	d_{24}
Město 3	d_{31}	d_{32}	d_{33}	d_{34}
Město 4	d_{41}	d_{42}	d_{43}	d_{44}

Tabulka 3-1: Matice vzdáleností

V matici jsou v řádcích a sloupcích vzdálenosti mezi městy. Vzdálenosti si označím d_{ij}. Tato vzdálenost udává nejkratší vzdálenost mezi městem i a městem j. Vzdálenosti kdy $i = j$, respektive mezi dvěma stejnými městy, jsou nulové.

Na obrázku je vidět jedna z možností zakreslení čtyř měst (A, B, C, D), kde všechna města jsou mezi sebou spojena ohodnocenou hranou. Ohodnocení hrany označuje vzdálenost jednoho města od druhého.

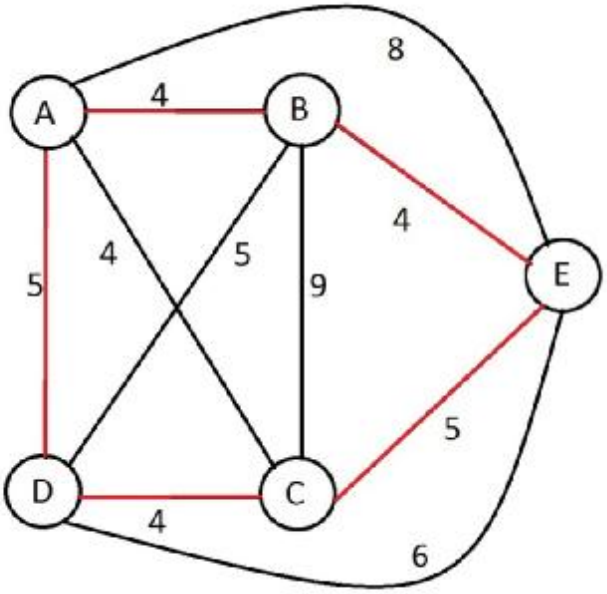


Graf 3-1: Úlohy obchodního cestujícího pro čtyři města

3.1 Symetrický typ

Je to taková úloha, která využívá vlastnosti neorientovaného grafu. To znamená, že vzdálenost například mezi bod A a B je stejná, jako vzdálenost mezi B a A. To znamená, že v případě zápisu této úlohy do matice by byla vytvořena matice symetrická podle hlavní diagonály. Takto definovanou úlohu lze využít například při výpočtu cesty mezi městy při využití letadla, vlaku nebo jakékoliv jiné dopravy.

Na obrázku můžeme vidět červeně optimální neorientovanou cestu mezi všemi uzly. Každým uzlem se prochází právě jednou a je tvořen Hamiltonův cyklus.



Graf 3-2: Symetrická úloha obchodního cestujícího

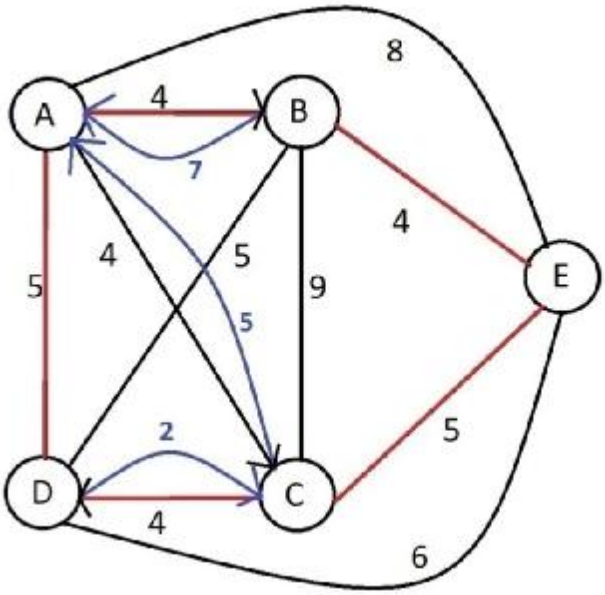
Matice symetrického typu grafu vycházející z obrázku by vypadala následovně:

	A	B	C	D	E
A	-	4	4	5	8
B	4	-	9	5	4
C	4	9	-	4	5
D	5	5	4	-	6
E	8	4	5	6	-

Tabulka 3-2: Symetrická matice vzdáleností

3.2 Asymetrický typ

Vychází z vlastností orientovaného grafu. To znamená, že vzdálenost mezi body A a B je jiná, než mezi body B a A. Výpočty takovéto úlohy jsou většinou komplikovanější, než symetrického typu. S takovou situací se můžeme například setkat, pokud budeme cestovat uvnitř města, ve kterém jsou jednosměrné ulice.



Graf 3-3: Asymetrická úloha obchodního cestujícího

Matice u asymetrické úlohy obchodního cestujícího by vypadala následovně:

	A	B	C	D	E
A	-	4	4	5	8
B	7	-	9	5	4
C	5	9	-	4	5
D	5	5	2	-	6
E	8	4	5	6	-

Při zkoumání matic symetrické a asymetrické úlohy si lze všimnout jednoho základního rozdílu. Hodnoty pod hlavní diagonálou jsou různé od hodnot nad hlavní diagonálou, to znamená, že například vzdálenost z A do B je 4 jednotky, ale vzdálenost z B do A je 7 jednotek. Na diagonále jsou vzdálenosti nulové (nebo označené pro potřeby výpočtu pomocí exaktních algoritmů jako velké číslo M), jsou různé od hodnot nad diagonálou.

4 Základní terminologie

V této kapitole jsou definovány základní termíny, označení a proměnné, které budou využity na vysvětlení exaktních a heuristických metod. Tyto zkratky pak budou využity v celém zbytku diplomové práce, ve které budeme pracovat se symetrickou maticí vzdáleností.

Jako první je využití teorie grafů pro úlohu obchodního cestujícího. Bude využit popis symetrického grafu, kdy graf $G = (V, A)$ je tvořen V městy a A hranami. Jednotlivá města patřící do množiny měst $V = \{v_1, v_2, \dots, v_n\}$ budou označena $v_1, v_2, \dots, v_n$, kde města jsou označena od 1 do n, resp. máme až n měst. Město, které je pojmenováno jako výchozí, tedy to město, ze které začínáme vytvářet cyklus a ze kterého vychází hrana, je označeno v_i. Město cílové, respektive to město, do kterého vstupuje hrana, je označeno v_j. V úloze se dále označují spojnice měst, resp. hrany. Hrany jsou označeny $A = \{a_{ij}\}$, kdy každá hrana je označena dvěma indexy. Indexem i a indexem j, kde index i označuje výchozí město v_i a index j označuje cílové město v_j.Nové město, které se bude nově vkládat do úlohy nebo do již vytvořeného cyklu, bude označeno písmenem v.

Dále je třeba, hlavně u exaktních algoritmů, definovat proměnné. První proměnnou je x_{ij}. Tato proměnná nabývá hodnot 1 v případě, že hrana mezi městy i a j bude vybrána a zařazena do cyklu a hodnotu 0 v případě, že nebude. Proměnná c_{ij} označuje matici vzdáleností, kdy i označuje výchozí město a j cílové město. Tato proměnná může nabývat libovolných hodnot, většinou se hodnoty pohybují v intervalu $\langle 0, \infty)$, protože záporná vzdálenost by neměla smysl v eukleidovském prostoru použitém v Problému obchodního cestujícího (byla by totiž vždy vybrána jako nejlepší). Matice vzdáleností je zadávána stejně, jako v předchozí kapitole, ale pro ilustraci je uvedena v tabulce 4-1.

	A	B	C	D	E
A	-	4	4	5	8
B	4	-	9	5	4
C	4	9	-	4	5
D	5	5	4	-	6
E	8	4	5	6	-

Tabulka 4-1: Symetrická matice vzdáleností

Pro heuristiku GENIUS se navíc bude využívat dvourozměrný prostor, ve kterém budeme uvažovat města (body), které budou mít vždy souřadnice podle osy x a osy y. Body tedy budou zapsány ve stylu: město (x, y). Popis eukleidovského prostoru lze najít v mnoha literaturách, např.: v [4].

5 Metody řešení úloh

Metody řešení úloh lze rozdělit na dvě hlavní skupiny, na exaktní algoritmy a heuristické metody. Každá z metod řešení problému obchodního cestujícího se dále dělí na různé výpočetní algoritmy.

Exaktní algoritmy jsou algoritmy, které vypočtou problém obchodního cestujícího a jejich výsledkem je optimální řešení úlohy. Pro výpočty pomocí exaktních algoritmů se využívají *metody větvení a mezí [5]* a *metody řezných nadrovin [14]*. Vzhledem k náročnosti obou úloh a řešení problému obchodního cestujícího za podmínek celočíselnosti se tyto algoritmy často využívají na řešení relativně malých úloh. Řešení i relativně malé úlohy může zabrat relativně delší čas, protože například při využití metody větvení a mezí může dojít k prohledání všech větví, než se nalezne optimální řešení. Proto lze říci, že exaktní metody sice vypočítají optimální řešení, ale u většího rozměru úloh (viz následující kapitoly), je vidět náročnost výpočetního času a také velké množství iterací.

Heuristické metody patří do skupiny metod, které budou také popsány v této práci. Tyto metody, oproti exaktním metodám, nevycházejí z předpokladu získání optimálního řešení úlohy. Nejdůležitějším faktorem heuristik je výpočetní jednoduchost a hlavně výpočetní čas úlohy. Heuristiky jsou nejvíce využívány pro malé a střední úlohy, kde většinou nedávají, ale mohou dát, optimální řešení úlohy, ale za velmi krátký čas. Využívat heuristiky ovšem má smysl i pro rozsáhlejší úlohy, kde je nejdůležitějším faktorem úspora času. Takové úlohy, které řeší exaktní algoritmy klidně i desítky minut, heuristické algoritmy mohou vypočítat úlohu za zlomek tohoto času. Některá srovnání jsou provedeny v dalších částech diplomové práce.

6 Exaktní algoritmy

Exaktní algoritmy pro řešení Úlohy obchodního cestujícího jsou většinou založené na smíšeném celočíselném programování. To znamená, že některé proměnné nabývají celočíselných hodnot a některé proměnné mohou nabývat libovolných hodnot. U všech proměnných platí podmínky nezápornosti. Exaktní algoritmy jsou většinou postaveny na jednom ze dvou algoritmů pro celočíselné úlohy, a to na metodě Větvení a mezí nebo na metodě Řezných nadrovin. Informace o těchto metodách budou uvedeny v této kapitole, ale z důvodu zaměření práce na heuristiku GENIUS, budou uvedeny pouze základní informace. Více informací lze najít např. v [5], [2], [14], [18], atd.

6.1 Větvení a mezí

Metoda větvení a mezí [18] je metodou pro výpočet úloh celočíselného nebo smíšeného lyneárního programování. Tuto metodu lze však využít i pro řešení speciálních úloh, mezi které patří i Problém obchodního cestujícího. Algoritmy, které jsou založené na této metodě jsou nepolynomiální a tudíž výpočet výsledku může probýhat i ve velmi dlouhém čase i u relativně malé úlohy. Často se u větších úloh po určitém čase výpočet zastavuje a metoda tak dává řešení s určitým odhadem odchylky optimálního řešení.

Pro ilustraci metody větvení a mezí se uvažuje úloha celočíselného smíšeného programování:

$$(MIP)\ \max\{cx + dy : Ax + Dy \leq b, x \in Z_+^n, y \in R_+^p\}.$$

Takto zapsanou úlohu lze ještě přepsat do následující formy:

$$\mathrm{z} = cx + dy \rightarrow max, \quad (6.1)$$

$$Ax + Dy \leq b, \quad (6.2)$$

$$x \in Z_+^n, y \in R_+^p. \quad (6.3)$$

kde rovnice (6.1) je účelovou funkcí, kterou budeme maximalizovat, (6.2) jsou omezující podmínky a (6.3) jsou omezení pro jednotlivé proměnné.

$$S = \{(x, y), x \in Z_+^n, R_+^p, Ax + Dy \in b\}. \quad (6.4)$$

Metoda větvění a mezí spočívá ve vyhledání optimálního neceločíselného řešení a následně jeho dělení množiny přípustných řešení S uvedené v (6.4) na podmnožiny $S^a, a \in A$. Na těchto podmnožinách se následně hledá optimální řešení. Pokud se v podmnožině S^a nepodaří nalézt optimální řešení, opakuje se rozdělení této podmnožiny.

Podmnožiny S^a se nazývají větve. Při řešení úloh metodou větvení a mezí lze získat velké množství větví, resp. podmnožin. Na každé z těchto větvích se hledá optimální řešení:

$$(\mathrm{MIP}^{\mathrm{a}}) \quad \max\{z = cx + dy;\ (x, y) \in S^a\}$$

Úlohu řešíme a každá nalezená optimální řešení v podmnožinách je třeba mezi sebou porovnat, hledá se největší hodnota účelové funkce u maximalizační úlohy, resp. nejmenší hodnota účelové funkce u úlohy minimalizační. Při výpočtu se na každé podmnožině zjišťuje aktuální hodnota horního odhadu účelové funkce. Tato hodnota se následně srovnává s jinými hodnotami účelových funkcí na ostatních větvích. Pokud by hodnota na jedné větvi byla nižší než na jiné, je výhodné opustit větev s nižší hodnotou účelové funkce a pokračovat ve výpočtu s větví, která tuto hodnotu má větší.

Kompletní informace o výpočtu hodnoty horního odhadu účelové funkce a o přesném matematickém popisu metody větvení a mezí lze nalézt na [18].

6.2 Problém obchodního cestujícího

Pro výpočet Problému obchodního cestujícího existuje mnoho formulací, zde bude uvedena pouze jedna formulace, která bude později využita při výpočtu příkladů.

$$\sum_{\mathrm{i}=1}^{\mathrm{n}} \sum_{\mathrm{j}}^{\mathrm{n}} c_{ij} x_{ij} \rightarrow min \tag{6.1}$$

$$\sum_{\mathrm{j}=1}^{\mathrm{n}} x_{ij} = 1 \quad \mathrm{i} = 1,2,\dots,\mathrm{n}, \tag{6.2}$$

$$\sum_{\mathrm{i}=1}^{\mathrm{n}} x_{ij} = 1 \quad \mathrm{j} = 1,2,\dots,\mathrm{n}, \tag{6.3}$$

$$\mathrm{v_i} - \mathrm{v_j} + \mathrm{n}\mathrm{x_{ij}} \leq \mathrm{n} - 1 \quad i = 1,2,\dots,\mathrm{n}, \mathrm{j} = 1,2,\dots,\mathrm{n},\ i \neq j, \tag{6.4}$$

$$x_{ij} \in \{0,1\} \quad i = 1,2,\dots,\mathrm{n},\ \mathrm{j} = 1,2,\dots,\mathrm{n}. \tag{6.5}$$

Rovnice (6.1) je účelová funkce, která se bude minimalizovat. Minimalizací účelové funkce se zajistí minimální délka celkového cyklu. Omezení (6.5) omezuje proměnnou x_{ij} tak, že bude nabývat pouze dvě hodnoty, a to 0 v případě, že nebude vytvořeno spojení (hrana) mezi městem i a městem j a naopak 1, v případě, že bude vytvořeno spojení. Rovnice (6.2) a (6.3) jsou lineární rovnice, které zabezpečí, že v matici výsledků, která má stejné rozměry jako matice vzdáleností, bude v každém řádku a každém sloupci právě jedna jednička. Omezení (6.5) je takzvaná smyčková podmínka [17], která slouží k odstranění případných podcyklů. Tato smyčková podmínka je nazvaná podle autorů Miller-Tucker-Zemlin. Takto formulovaná úloha se využije při výpočtu pomocí programu LINGO.

7 Heuristiky

Heuristiky jsou metody výpočtu komplikovaných úloh, které za využití krátkého výpočetního času, umožňují najít řešení, blížící se optimálnímu. Tyto metody jsou velmi efektivní pro řešení úloh malého nebo středního rozsahu, kde se teoreticky může (s minimální pravděpodobností) podařit najít i optimální řešení. Pro řešení úloh většího rozsahu mají samozřejmě také význam a při srovnání s exaktními metodami dosahují za zlomek času výsledku. Dnes existuje velké množství různých heuristik, které se liší rychlostí výpočtu, přesností případně složitostí programování. Heuristiky lze rozdělit do třech základních skupin podle jejich aplikace [11]:

1. Na heuristiky, které tvoří cyklus tak, že postupují iterativním způsobem. To znamená, že spojují body na základě přidávání jednoho bodu v každém kroku výpočtu,
2. Na heuristiky, které upravují již vytvořený cyklus na základě záměn bodů,
3. Na kompozitní algoritmy, které kombinují dvě předešlé heuristiky.

Nyní uvedu příklady heuristik ke každé ze tří skupin aplikace a následně některé jejich zástupce vysvětlím. Prvním typem heuristik jsou například Vkládací algoritmy, mezi které patří Metoda nejbližšího souseda, Greedy nebo GENI. Heuristiky, které upravují již vytvořený Hamiltonový cyklus je například US, 3-Opt, 2-Opt. Posledním typem jsou kompozitní algoritmy, které kombinují oba dva předešlé. Kombinací může být metoda GENIUS, CCAO, která kombinuje obě předešlé.

7.1 Vkládací algoritmy

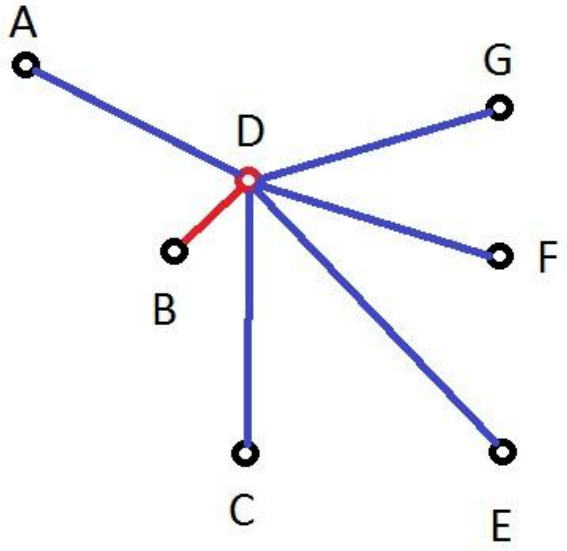
Vkládací algoritmy patří mezi první vyvinuté heuristiky pro Problém obchodního cestujícího. První typy se objevovali v 60. a 70. letech 20. století a sloužily k řešení rozsáhlých úloh a pro vyhledávání řešení blízkého optimálnímu. Popis těchto metod lze najít v mnoha literaturách a pro svou jednoduchost jsou hojně využívány.

Vkládací algoritmy začínají výběrem města nebo bodu, který označí jako výchozí. Následně k němu vyhledávají nejbližší město (hranu s nejmenším ohodnocením), které přidají a vytvoří tak cyklus. Vložení tohoto města se prodlouží celkový cyklus, který se snažíme minimalizovat. Postupným vkládáním dalších měst postupuje až do momentu, dokud nevytvoříme cyklus přes všechna města. Pro vybrání následujících měst, která mají být vložena do cyklu, využiji následující možnosti:

7.1.1 Metoda nejbližšího souseda

Metoda nejbližšího souseda vychází z velice jednoduchého algoritmu. Bude k dispozici matice nebo seznam měst a jejich vzdáleností mezi sebou. Ideální je mít pro výpočet symetrickou matici. Výpočet začíná zvolením jednoho města jako výchozího a následným výběrem dalšího města, které je k němu nejblíže. Takto vložené město se následně bere jako výchozí a hledá se k němu znova nejbližší město. Algoritmus se opakuje až do momentu, kdy jsou všechna města spojena. Je třeba ale mít při výpočtu na paměti, že každé město může být navštívenou právě jednou.

Pro ilustraci si lze představit metodu nejbližšího souseda podle následujícího obrázku. Jako výchozí bod se zvolí město D. K němu se hledá takové město, které je nejblíže. Srovnává se tedy každé město s městem D. V tomto případě, se jedná o město B, které bylo zvoleno jako nejbližší a od něj se dále pokračuje s výpočtem a hledá se k němu znovu nejbližší město.



Obrázek 7-1: Metoda nejbližšího souseda

7.1.2 Algoritmus Greedy

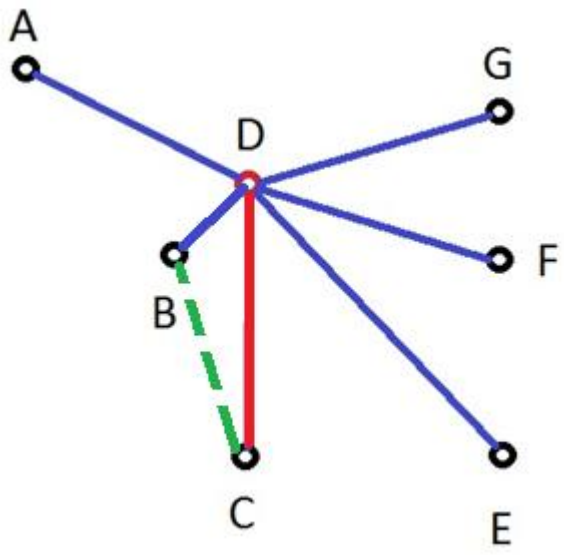
Algoritmus Greedy [12] je velmi podobný metodě nejbližšího souseda a mnozí autoři ho s touto heuristikou zaměňují. Vychází se ze stejného předpokladu jako metoda nejbližšího souseda. Předpoklad je, že jsou známa všechna města a jejich vzájemné vzdálenosti. První krok výpočtu je vybrání celkové nejkratší vzdálenosti ze všech vzdáleností, která jsou zadaná v matici vzdáleností. Jakmile se určí nejkratší spojení, zaznamenávají se města, která se spojila a vyhledávají se další nejkratší vzdálenosti mezi zbylými městy. Při postupu

se musí dávat pozor, aby se prošlo každým městem právě jednou. Snahou je vytvořit cyklus přidáním vždy jednoho města podle nejkratší vzdálenosti. Algoritmus opakuje až do momentu, kdy jsou všechna města spojena.

Pokud se srovná tento algoritmus s Metodou nejbližšího souseda, je zde vidět podobnost ve vyhledávání vždy nejkratší hrany, avšak rozdíl je ve způsobu vyhledávání nejbližší vzdálenosti, resp. nejkratší hrany. Zatímco Metoda nejbližšího souseda vyhledává vždy nejkratší vzdálenost z nově přidaného města, metoda Greedy vždy vyhledává nejkratší vzdálenost ze všech měst.

7.1.3 Algoritmus Clarka-Wrighta

Tento algoritmus [12] byl původně využíván pro Rozvozní problém. Pro Problém obchodního cestujícího byl tento algoritmus upraven. Algoritmus vychází z předpokladu, že jedno město je označeno jako výchozí bod. Výchozí bod je spojen spojnicemi (hranami) s ostatními městy a obchodní cestující se po navštívení města vždy vrací do výchozího bodu. Po určení výchozího stavu úlohy, přichází moment optimalizace úlohy. Ta vychází ze dvou předpokladů: zkrácení celkové vzdálenosti a zajištění, že každým městem projdeme právě jednou. Aby se zajistilo, že obchodní cestující vždy projde každým městem právě jednou, tak algoritmus v momentě, že se dostane obchodní cestující mimo výchozí město, zjišťuje, jestli lze celkový cyklus zkrátit, pokud bude procházet přes jiné město využitím algoritmu Greedy. Pokud se nalezne kratší vzdálenost, upraví se cyklus a zapíše se nová celková vzdálenost. Takto se bude pokračovat až do momentu, kdy budou všechna města spojena a bude vytvořen Hamiltonův cyklus.



Obrázek 7-2: Clark-Wright heuristika

Na obrázků je vidět výchozí situace. Bod D je označen jako výchozí bod a z něj jsou vytvořeny spojnice do ostatních měst - modrou barvou. Obchodní cestující vyšel z bodu D a došel do bodu C. Z něj se vyhledává, jestli existuje nějaké řešení, při kterém by se mohl vrátit do původního bodu D, projít ještě nějakým bližším a zkrátit celkovou vzdálenost. Algoritmus zvolil bod B, přes který se vrací do výchozího bodu. Následně obchodní cestující znova odejde z výchozího bodu např. do bodu E a situace se znova opakuje, dokud nenajde řešení úlohy a dokud nevznikne Hamiltonův cyklus.

Heuristiku GENI, která patří také do vkládacích algoritmů, vysvětlím v následující kapitole.

7.2 Heuristiky upravující cykly

7.2.1 k-Opt algoritmus

Heuristiky založeny na vyhledávání k-opt řešení [13]. V názvu heuristiky se objevuje písmenko *k*, které označuje, kolik spojení (hran) se bude z již vytvořeného cyklu vyjímat a nahrazovat jinými. Tyto heuristiky rozdělím na dva nejvíce využívané typy, a to na 2-Opt (Croes 1958) a 3-Opt (Lin 1965) heuristiky pro symetrický Problém obchodního cestujícího. Pro využití této heuristiky se musí splnit podmínka, že je již vytvořený nějaký cyklus, který má určitou vzdálenost a jsou v něm zahrnuta všechna města. Na základě zvolení vhodného *k*, se bude určovat, kolik spojení (hran) se z již vytvořeného cyklu vyjme a bude nahrazeno jinými. Následně po vytvoření nového cyklu, dojde výpočtu celkové délky cyklu a srovnání s původní délkou cyklu. Pokud je nové řešení lepší, než původní, vybere se nové řešení a pokračuje se s tímto řešením při optimalizaci. Pokud je ovšem nové řešení horší než původní, vybere se původní řešení jako výchozí, zvolí se jiné hrany a pokračuje se s výpočtem. Výpočet se opakuje až do momentu, kdy nelze vybrat spojení (hrany) tak, aby nové řešení mělo lepší hodnotu účelové funkce, resp. aby se dostala menší celkovou vzdálenost. V tom momentě lze konstatovat, že se dostalo řešení k-Opt.

7.2.2 Metoda Lin-Kernighana

Metoda Lin-Kernigham [10] je jedna z nejefektivnějších metod pro vyhledávání řešení blízkého optimálnímu. Nevýhoda tohoto algoritmu je v obtížnosti implementace a vytvoření softwaru, který ho dokázal řešit. Metoda Lin-Kerninghamova vychází z výše uvedeného k-Opt algoritmu. Metoda je iterativní a v každé iteraci se řeší následující situace:

1. Určí se náhodně celý cyklus a zvolí se výchozí bod pro optimalizaci.
2. Zvolí se výchozí *k* pro řešení k-Opt úlohy, většinou je zvoleno $k = 2$ a vypočte se celková délka. Pokud je nová délka trasy kratší než původní, zvolí se toto jako nové řešení. Pokud ne, pokračuje se v dalším kroku.

3. Zvolí se vyšší hodnota k o jednotku. Pokud v prvním kroku bylo $k = 2$, v dalším kroku je to $k = 3$. Většinou se volí pouze hodnoty $k = 2$ a $k = 3$, v některých situacích byly zvoleny i hodnoty $k = 4$ a $k = 5$, ale tyto hodnoty se považují za minimálně využívané. Pokud se nalezne nové řešení, které je kratší než původní, zapíše se a pokračuje se s tímto řešením ve výpočtu. Vrací se do předchozího kroku. Pokud ani při k=3 nelze zkrátit cyklus, vrací se do předchozího bodu a zvolí se jiné spojnice (hrany).
4. Pokud již nelze zvolit žádné spojnice (hrany) tak, aby se zkrátil cyklus, nalezlio se řešení blízké optimálnímu.

Na základě těchto čtyř bodů lze vidět složitost této metody. Složitost vyplývá hlavně ve využití dvou metod úprav již vytvořených cyklů, a to 2-Opt a 3-Opt metody. Oproti této složitosti však Lin-Kernighamova heuristika dává velmi dobré výsledky (Keld Helsgaun, An Effective Implementation of the Lin-Kernighan Travelling Salesman Heuristic) v porovnání s ostatními heuristikami, které jsem uvedl na začátku této kapitoly. Je to hlavně díky využití optimalizace již vytvořených cyklů.

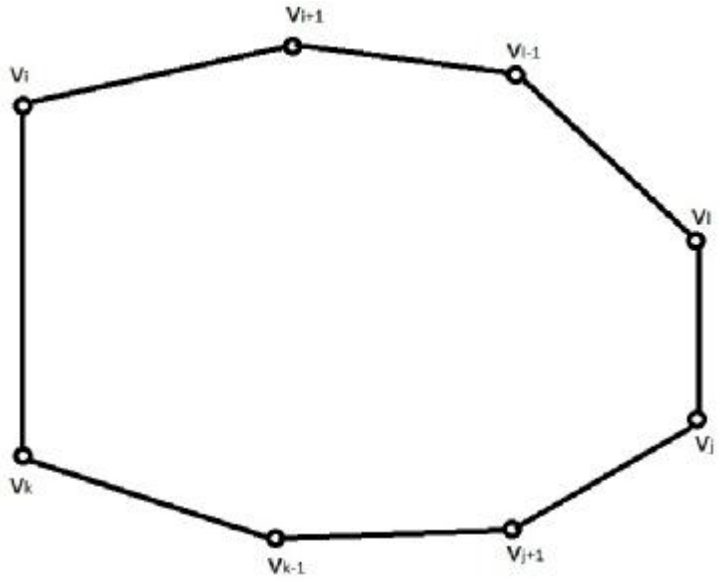
8 GENIUS

Heuristika GENIUS, vysvětlená a popsaná v [8], je jedna z metod výpočtu Problému obchodního cestujícího, která byla uvedena matematiky a ekonomy Michaelem Gendreauem, Hertzem a Laportem (1992). Tato heuristika je používána především pro symetrický problém obchodního cestující, teda pro úlohu, kde je matice vzdáleností symetrická. Algoritmus GENIUS je rozdělený do dvou částí. První část je GENI, Generalized Insertion, což je heuristika pro výpočet základního řešení úlohy. Druhý část je postoptimalizace zvaná US, Unstringing and Stringing.

8.1 Generalized Insertion (GENI)

Generalized Insertion Procedure, zkratka GENI, je heuristika, která slouží k řešení Problému obchodního cestujícího. Lze ji využít na řešení jak symetrické, tak i asymetrické úlohy, ale z důvodu zjednodušení, se bude uvažovat pouze symetrická úloha**.** GENI je heuristika, která slouží k vytvoření Hamiltonova cyklu, tedy vytvoření spojení všech měst tak, že z výchozího bodu se projdou všechna měst tak, že každým se projde právě jednou, návrat je do výchozího bodu. Metoda začíná výběrem prvních tří měst, která vytvoří cyklus. V dalším kroku metoda vkládá v každé iteraci právě jedno město na základě dvou typů vkládacího algoritmu. Tyto typy vkládacího algoritmu nazveme typ 1 a typ 2.

Oba dva typy vkládacího algoritmu vycházejí ze stejného algoritmu. Označí se město v jako město, které se bude vkládat do již vytvořeného cyklu. Toto město se bude vkládat mezi další dvě města, která se označí v_i a v_j. Pro výpočty se budou případně uvažovat ještě další dvě města v_k a v_l. Město v_k je takové město, které při vytvoření cyklu je následující město po městech v_i a v_j. Pokud budeme uvažovat pořadí měst opačné, půjde se nejprve přes město v_j a až následně přes město v_i. Město které se bude nacházet po městě v_i se označí v_l. Pro jakékoliv město v_h, $h = i, j, k, l$ platí, že v_{h-1} je jejich předchůdce a v_{h+1} je jejich následovník. Pro ilustraci se může využít obrázek pod textem, kde jsou označena města v_i, v_j, v_k a v_l.

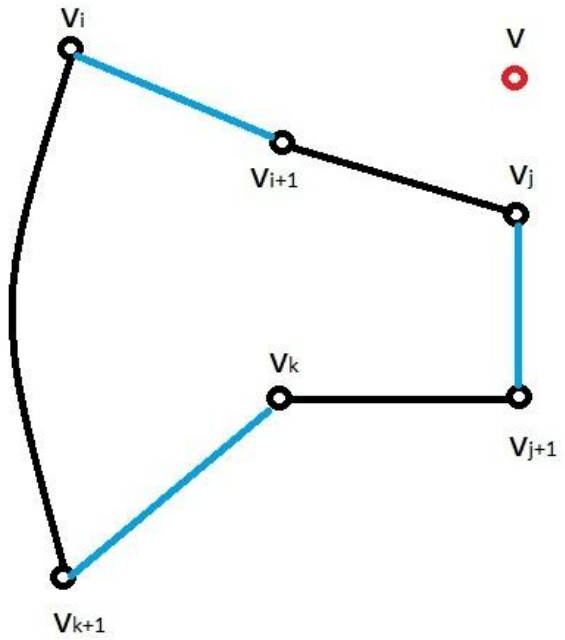


Obrázek 8-1: Zobrazení vrcholů při úloze 8 měst

8.1.1 Vložení vrcholu typ 1

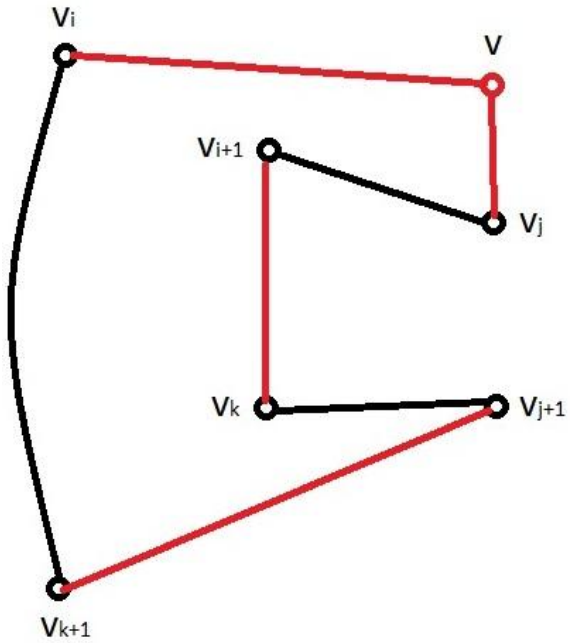
Na základě předchozích informací, se bude uvažovat jako první typ 1 vložení nového města do cyklu. Při aplikaci vložení tohoto typu, se musí vycházet ze dvou kritérií: $v_k \neq v_i$ a $v_k \neq v_j$. Město v je vloženo mezi města v_i a v_j tak, že v již vytvořeném cyklu dojde v prvním kroku k odstranění již vytvořených hran a nahrazení novými hranami. V tomto případě dojde k odebrání hran (v_i, v_{i+1}), (v_j, v_{j+1}) a hrany (v_k, v_{k+1}). Tyto hrany jsou nahrazeny novými hranami následujícími hranami (v_j, v), (v, v_j), (v_{j+1}, v_k) a (v_{j+1}, v_{k+1}). Vzhledem k této změně, jsou cykly $(v_{i+1}, \dots, v_j)$ a $(v_{j+1}, \dots, v_k)$ navzájem opačné. Jako speciální případ nastává situace, kdy $j = i + 1$ a $k = j + 1$, kdy se dostane shodný výsledek jako u standardního vkládacího algoritmu.

Na obrázku 8-2 je vidět aktuální Hamiltonův cyklus, kde černě jsou označeny hrany, které se nebudou měnit a modře hrany, které se budou zaměňovat. Navíc je zde již zobrazen bod v, který bude zařazen do Hamiltonova cyklu.



Obrázek 8-2: Vkládací algoritmus typ 1 před aplikací

Na obrázku 8-3 je vidět záměna hran na základě aplikace prvního typu vkládacího algoritmu heuristiky GENI. Hrany, které byly na prvním obrázku označeny modře, byly odebrány a nahrazeny hranami, které mají červenou barvu. Začleněn byl také vrchol v a byl tak vytvořen cyklus, který je delší než předchozí, ale již obsahuje nové město v.

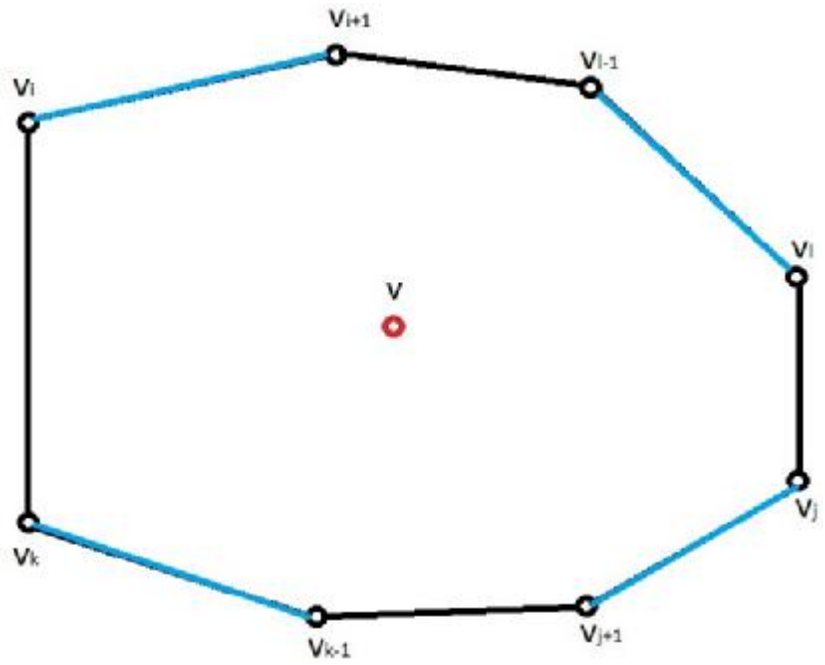


Obrázek 8-3: Vkládací algoritmus typ 1 po aplikaci

8.1.2 Vložení vrcholu typ 2

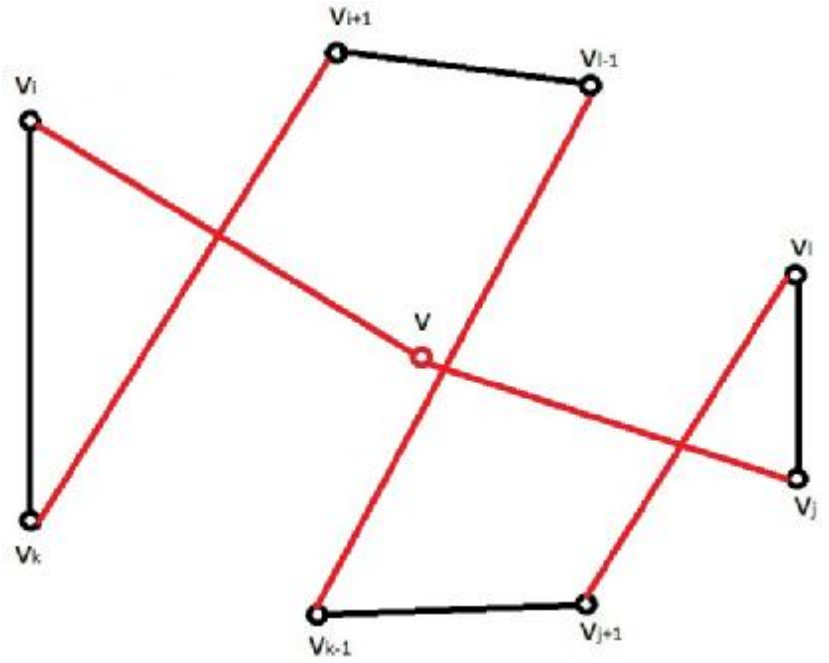
Druhý typ vkládacího algoritmu vychází z prvního typu. Bude se vycházet z předpokladu již vytvořeného Hamiltonova cyklu. Jako kritéria pro užití druhého typu vkládacího algoritmu se bude uvažovat $v_k \neq v_j$ a $v_k \neq v_{j+1}$, $v_l \neq v_i$ a $v_l \neq v_{i+1}$. Za těchto kritérií se bude do vytvořeného cyklu vkládat město v mezi města v_i a v_j. Na základě tohoto vložení dojde k odebrání hran (v_i, v_{i+1}), (v_{l-1}, v_l), (v_j, v_{j+1}) a (v_{k-1}, v_k). Tyto hrany jsou nahrazeny novými hranami (v_i, v), (v, v_j), (v_l, v_{j+1}), (v_{k-1}, v_{l-1}) a (v_{i+1}, v_k). Vzhledem k této změně, jsou cykly $(v_{i+1}, \dots, v_{l-1})$ a $(v_l, \dots, v_j)$ navzájem opačné.

Na obrázku 8-4 lze vidět aktuální Hamiltonův cyklus, kde černě jsou označeny hrany, které se nebudou měnit a modře hrany, které se budou zaměňovat. Navíc je zde již zobrazen bod v, který bude zařazen do Hamiltonova cyklu.



Obrázek 8-4: Vkládací algoritmus typ 2 před aplikací

Na obrázku 8-5 je vidět záměna hran na základě aplikace druhého typu vkládacího algoritmu heuristiky GENI. Hrany, které byly na prvním obrázku označeny modře, byly odebrány a nahrazeny hranami, které mají červenou barvu. Začleněn byl také vrchol v a byl tak vytvořen cyklus, který je delší než předchozí, ale již obsahuje nové město v.



Obrázek 8-5: Vkládací algoritmus typ 2 po aplikací

Na základě vysvětlení způsobu vkládání vrcholů v heuristice GENI, je možné jednoduše zjistit, že lze touto metodou řešit jak symetrický, tak asymetrický problém obchodního cestujícího, na základě možnosti orientace tvorby cyklu jako „po směru“ hodinových ručiček tak i po směru opačném. Limitace tohoto algoritmu ale spočívá v množství řešení pro $v_i,\ v_j,\ v_k\ a\ v_l$, kde maximální počet řešení úlohy je n^4.

Pro jakékoliv město $v\ \in V$ se určí jeho p-sousední města $Np(v)$ jako množinu všech p měst na cyklu, která jsou nejblíže městu v na základě matice vzdáleností. Pokud má město v méně než p sousedních měst v cyklu, všechna budou patřit do $Np(v)$. Pro zadaný parametr hodnoty p tak se jako první vybere $v_i, v_j\ \in Np(v)$, $v_k \in Np(v_{i+1})$ a $v_l \in Np(v_{j+1})$. Také se bude uvažovat vložení města v mezi dva po sobě jdoucí města v_i a v_{i+1}, pokud $v_i \in Np(v)$. V praxi se často potkává s relativně malou hodnota p.

8.1.3 Kroky výpočtu algoritmu GENI

Výpočet pomocí algoritmu GENI se skládá ze třech jdoucích kroků. Jsou uvedeny v následujících třech bodech:

1. Vytvoří se počáteční cyklus zvolením libovolných třech měst. Vyhledají se nejbližší města k těmto třem na základě definice p-sousedních měst $Np(v)$.
2. Libovolně se zvolí takové město v, které ještě netvoří cyklus s již obsazenými městy. Pro vložení tohoto města se bude využívat metody vložení města s nejmenším růstem celkové délky cyklu a budou se uvažovat obě možnosti orientace cyklu. Upraví se seznam p-sousedních měst $Np(v)$ s ohledem na fakt, že město v je nyní součástí cyklu.

3. Pokud jsou všechny vrcholy zahrnuty v cyklu, dokončil se výpočet pomocí algoritmu GENI. Pokud některý vrchol není zahrnutý, vrací se k předchozímu bodu.

Ve druhém kroku se musí brát v úvahu $O(p^4)$ výběrů z $v_i,\ v_j,\ v_k\ a\ v_l$. Vrchol v je tak vložen do cyklu na základě nejmenšího růstu celkové vzdálenosti. Celkový cyklus je upraven a pokračuje se s výpočtem. Celé tato operace zabere $O(n)$ výpočetního času. Druhý krok výpočtu je tedy spuštěn $(n-3)$ krát, takže celkový potřebný výpočetní čas pro algoritmus GENI je $O(np^4 + n^2)$.

Zajímavostí na heuristice GENI, je využívání kroků lokální optimalizace uvnitř vkládacího algoritmu. Podobné nápady měli i Steiglitz a Wiener, kteří využili metodu 3-Opt a upravili ji o dynamické prvky. Pokud by se důkladně prozkoumaly na oba dva vkládací algoritmy GENI, tak první vkládací algoritmus je vlastně 3-Opt metoda řešení a druhý typ vkládacího algoritmu je 4-Opt metoda. Výhoda metody GENI je ovšem v počtu vrcholů, které mohou být vloženy do cyklu. Tento počet vrcholů je omezen na p-sousedních, neboli nejbližší vrcholů ke každému z nich.

8.2 Unstringing and Stringing (US)

Na začátku kapitoly jsem uvedl, že se budu zabývat heuristikou GENIUS. V první části jsem uvedl algoritmus GENI a nyní se podívám na post-optimalizační část této heuristiky, na část US, neboli Unstringing and Stringing. Tento algoritmus vychází z algoritmu GENI. Je založen na odebírání a následným vkládáním měst do cyklu. Algoritmus začíná na vytvořeném cyklu, např. pomocí algoritmu GENI, a má uloženou celkovou délku cyklu. V první fázi odebere jedno z měst z cyklu. V druhém kroku na základě metody vložení města s nejmenším růstem celkové délky cyklu, toto město znovu vloží do cyklu a bude zkoumat, jestli je celková délka cyklu kratší (stejná) nebo větší. V případě, že je délka cyklu kratší nebo stejná, zapamatuje si řešení a pokračuje s tímto řešením dál ve výpočtu. Pokud je délka cyklu větší, nebude brát v úvahu toto řešení, vrátí se k původnímu a zvolí pro další část výpočtu jiné město.

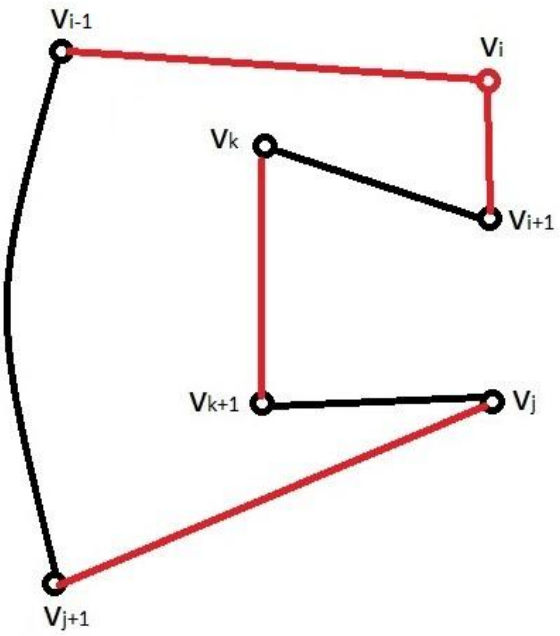
Část výpočtu s přidáváním města zpátky do cyklu je identická s druhým krokem výpočtu algoritmu GENI. Samozřejmě při tvorbě cyklu vložením města je třeba uvažovat dva směry výpočtu. Stejně jako u algoritmu GENI se budou uvažovat dva typy odebírání a vkládání vrcholů.

8.2.1 Odebírání vrcholu typ 1

První typ odebírání vrcholu vychází z předpokladu, že $v_j \in Np(v_{i+1})$. Dále je třeba určit si směr, jakým se budou odebírat hrany. Bude se uvažovat směr $(v_{i+1}, \dots, v_{j-1})$ za předpokladu, že $v_k \in Np(v_{i-1})$. Na základě určení směru výpočtu se teď lze zaměřit na to, které hrany se vyjmou a za které se vymění. V prvním kroku se vyjmou hrany (v_{i-1}, v_i),

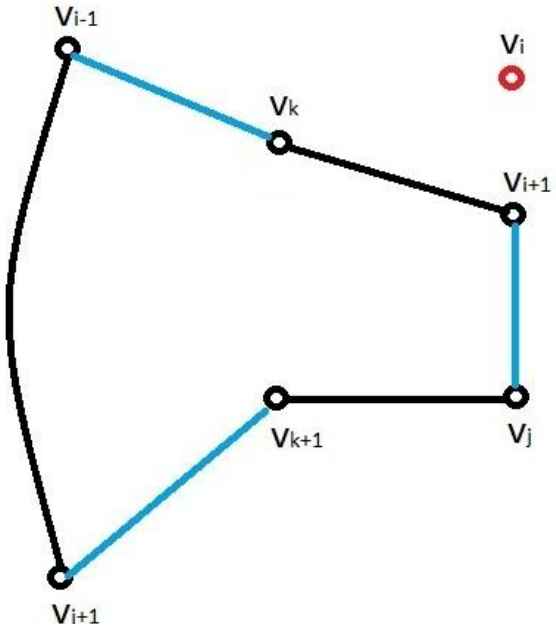
(v_i, v_{i+1}), (v_k, v_{k+1}) a (v_j, v_{j+1}) a tím se zajistí, že bude možno vyjmout město v z cyklu. Tyto hrany se následně nahradí hranami (v_{i-1}, v_k), (v_{i+1}, v_j) a (v_{k+1}, v_{j+1}). Vzhledem k této změně, jsou cykly $(v_{i+1}, \dots, v_j)$ a $(v_{k+1}, \dots, v_j)$ navzájem opačné.

Na obrázku 8-5 je vidět Hamiltonův cyklus vytvořený výpočtem například metodou GENI. Červeně jsou označeny hrany, které se budou odebírat z již vytvořeného cyklu. Dojde tak k vyjmutí města v_i z vytvořeného cyklu tak, že se vytvoří nový cyklus bez tohoto města a červeně označené hrany se nahradí jinými.



Obrázek 8-5: Odebírací algoritmus typ 1 před aplikací

Na obrázku 8-6 je vidět výsledek po odebrání města v_i z Hamiltonova cyklu. Původní hrany jsou nahrazeny novými modře znázorněnými.

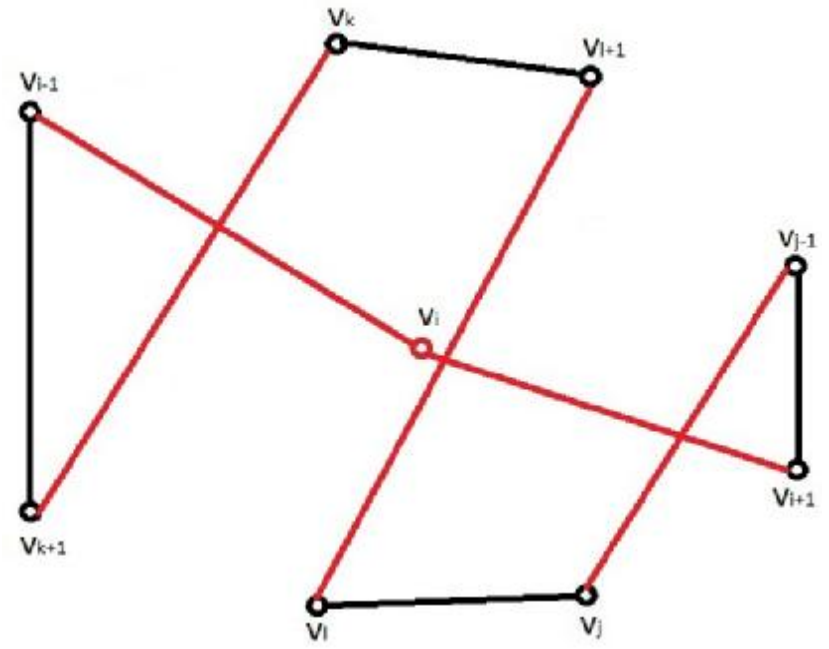


Obrázek 8-6: Odebírací algoritmus typ 1 po aplikaci

8.2.2 Odebírání vrcholu typ 2

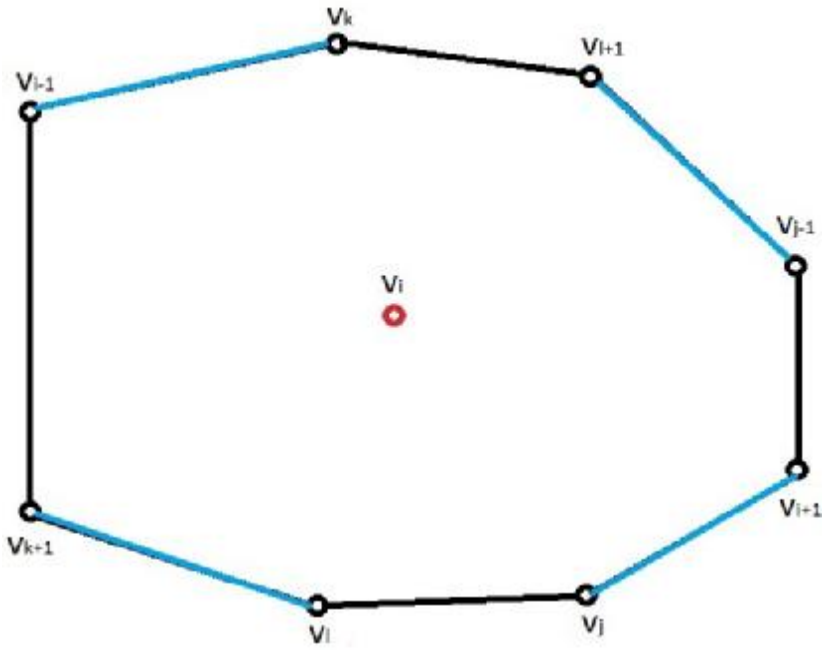
Druhý typ odebírání vrcholu taktéž vychází z algoritmu GENI. Vychází se z předpokladu, že $v_j \in Np(v_{i+1})$. Stejně jako u US typu 1 je třeba dávat pozor na směr cyklu, respektive směr odebírání hran. Bude se uvažovat $v_k \in Np(v_{i-1})$ jako město na cyklu $(v_{i+1}, \dots, v_{j-1})$ a také $v_l \in Np(v_{k+1})$ jako město cyklu $(v_j, \dots, v_{k-1})$. V prvním kroku se budou odebírat hrany (v_{i-1}, v_i), (v_i, v_{i+1}), (v_{j-1}, v_j), (v_l, v_{l+1}) a (v_k, v_{k+1}), abychom mohli vyjmout město v_i z cyklu. Tyto hrany se pak nahradí hranami (v_{i-1}, v_k), (v_{l+1}, v_{j-1}), (v_{i+1}, v_j) a (v_l, v_{k+1}). Vzhledem k této změně, jsou cykly $(v_{i+1}, \dots, v_{j-1})$ a $(v_{l+1}, \dots, v_k)$ navzájem opačné.

Na obrázku 8-7 je vidět Hamiltonův cyklus vytvořený výpočtem například metodou GENI. Červeně jsou označeny hrany, které se budou odebírat z již vytvořeného cyklu. Dojde tak k vyjmutí města v_i z vytvořeného cyklu tak, že se vytvoří nový cyklus bez tohoto města a červeně označené hrany se nahradí jinými.



Obrázek 8-7: Odebírací algoritmus typ 2 před aplikací

Na obrázku 8-8 je vidět výsledek po odebrání města v_i z Hamiltonova cyklu. Původní hrany jsou nahrazeny novými modře znázorněnými.



Obrázek 8-8: Odebírací algoritmus typ 2 po aplikaci

8.2.3 Algoritmus US

Výpočet algoritmu US se rozdělí do dvou základních kroků, stejně, jako je rozdělen u algoritmu GENI.

1. Uvažuje se vytvořená cesta po výpočtu metodou GENI. Bude se uvažovat původní cyklus τ a celkovou délku cyklu z. Dále se bude využívat $\tau^* := \tau$ jako optimalizovaný cyklus a $z^* := z$ jako optimalizovanou celkovou vzdálenost. Písmenem t se bude označovat iterace. První iterace je $t = 1$ a na ní následuje iterace $t + 1$, $t + 2$, atd.
2. Ve druhém kroku se začíná s vytvořeným Hamiltonovým cyklem τ. Aplikuje se algoritmus US, hrany vyjmou se a zase zpátky vloží do cyklu na základě typu 1 nebo typu 2 algoritmu. Samozřejmě se budou uvažovat dva možné směry výpočtu úlohy, „po směru“ hodinových ručiček nebo naopak. Úpravou tak cyklus označený $\tau' := \tau$ s hodnotou celkové délky cyklu $z' := z$.
   2.1. Pokud $z < z^*$, tak $\tau^* := \tau$, $z^* := z$ a $t = 1$. Pokud nebylo nalezeno lepší řešení po provedení algoritmu US, algoritmus se znova vrací do bodu dva a řeší se úloha s původním řešením.
   2.2. Pokud $z \geq z^*$, tak $t := t + 1$. Pokud bude hodnota optimalizované délky cyklu menší, našlo se nové řešení a pokračuje se dále ve výpočtu s tímto řešením.
   2.3. Pokud $t = n + 1$, tak výpočet končí. Nejlepším řešení úlohy tak je τ^* s optimalizovanou délkou cyklu z^*. Pokud to tak není, vrací se zpátky do bodu dva a pokračuje se v post-optimalizaci.

Je třeba poznamenat, že druhý krok je použit k poslednímu, ne vždy k nejlepšímu řešení úlohy s nejkratším cyklem. Může se tak stát, že celková délka cyklu, po dvou po sobě nalezených momentálně nejlepších řešeních, může vzrůst. Město, které bude takto odebráno z cyklu, nemusí být umístěno přesně mezi dvěma dalšími vrcholy, které obsahují p-sousedních měst. Nicméně vložení města zpátky do původní pozice, ve které se nacházelo před jeho odebráním, není v této metodě povoleno. Výhodou metody zůstává, že nejlepší nalezený cyklus je vždy uchován v paměti.

9 Program pro výpočet algoritmu GENIUS

Hlavním cílem diplomové práce bylo vytvoření programu pro počítače, který bude vypočítávat Problém obchodního cestujícího na základě heuristiky GENIUS. Tento program jsem musel naprogramovat v nějakém počítačovém jazyce. V počítačovém světě existuje velké množství programovacích jazyků, jak již jazyků, pomocí kterých se mohou vytvořit spouštěcí soubory, které budou fungovat na všech počítačích, tak i jazyky, pomocí kterých lze vytvořit webovou stránku, která bude takovou úlohu také řešit.

Při volbě jazyka se musely brát v úvahu různé faktory. Prvním byla využitelnost programu na všech počítačích přes všechny platformy a operační systémy. Druhá byla složitost programování, ale také možnosti využívat dynamické proměnné a čerpání dat z jiných souborů.

Jako první byl na výběr Microsoft Excel a jeho Visual Basic. V tomto programu bylo napsáno velké množství programů, avšak makra jako taková neposkytují uživateli volnost při programování. Dále to byla JAVA, která je dnes velmi oblíbená, ale její velká nevýhoda je nutnost ji mít nainstalovanou a tak na některých počítačích by nemusela fungovat správně. Také je JAVA relativně pomalá při iterativních výpočtech, takže také nepřipadala v úvahu. Programovací jazyk PHP, který se používá na tvorbu webových aplikací, byl jednou z ideálních možností. Problém ale nastává v momentě, kdy uživatel chce využít program v tomto jazyce napsaný. Musí mít buď doma server, na kterém běží Apache a webová aplikace a nebo musí být stále připojen k internetu.

Poslední možnost, která připadala v úvahu, byl jazyk C++. Tento jazyk je relativně starý, protože vychází z jazyku C, který vznikal v 90. letech 20. století. Jazyk C++ byl standardizován od roku 1998. Díky vývoji jazyka C++, lze tedy zajistit zpětnou kompatibilitu i se staršími počítači, operačními systémy a tudíž lze jej využít i napříč platformami. Lze je provozovat jak na počítačích s operačním systémem Microsoft Windows tak i na systémech postavených na Linuxovém jádru. Dále tento jazyk nabízí absolutní volnost ve využívání proměnných a dokonce obsahuje knihovny, ve kterých jsou popsány funkce, které program bude využívat. Proto při programování se definuje, jaké knihovny bude program využívat, a tak bude zajištěno, že bude využívat jen ty, které jsou potřeba. Tím je zajištěna minimalizace výpočetního času. Existuje zde také možnost využívat i nestandardní knihovny a tím se rozšiřuje využitelnost jazyka na maximum. Poslední zajímavostí o jazyku C++ je ta, že z něj vychází některé dnes velmi používané jazyky, například JAVA, .NET, C#, atd.

9.1 INFORMACE O PROGRAMU

Program, jak jsem uvedl na začátku kapitoly, je napsaný v počítačovém jazyce C++. Program je součástí diplomové práce, je omezený na výpočet minimálně 10 měst a maximálně 20 měst. Testy programu ale probíhali i se 40 a 50 městy a tak lze program případně rozšířit i na výpočet velmi rozsáhlých úloh. Program je umístěn na přiloženém CD, kde je umístěn pod názvem GENIUS.exe.

Spustitelný soubor byl testován na počítačích s operačním systémem Windows XP 32bit, Windows 7 32bit a Windows 7 64bit. Na ostatních platformách nebyl testován, ale vzhledem k původu programovacího jazyku C++ a jeho zpětné kompatibilitě a funkčnosti na nejnovějších operačních systémech, by zpětná kompatibility měla být zaručena.

V následujících částech kapitoly popíšu, jak program funguje a jak je naprogramován. Popíšu, jakým způsobem je třeba zadávat vstupní data a jak budou zobrazena výstupní data. V poslední části kapitoly ukážu část zdrojového kódu, který i popíšu.

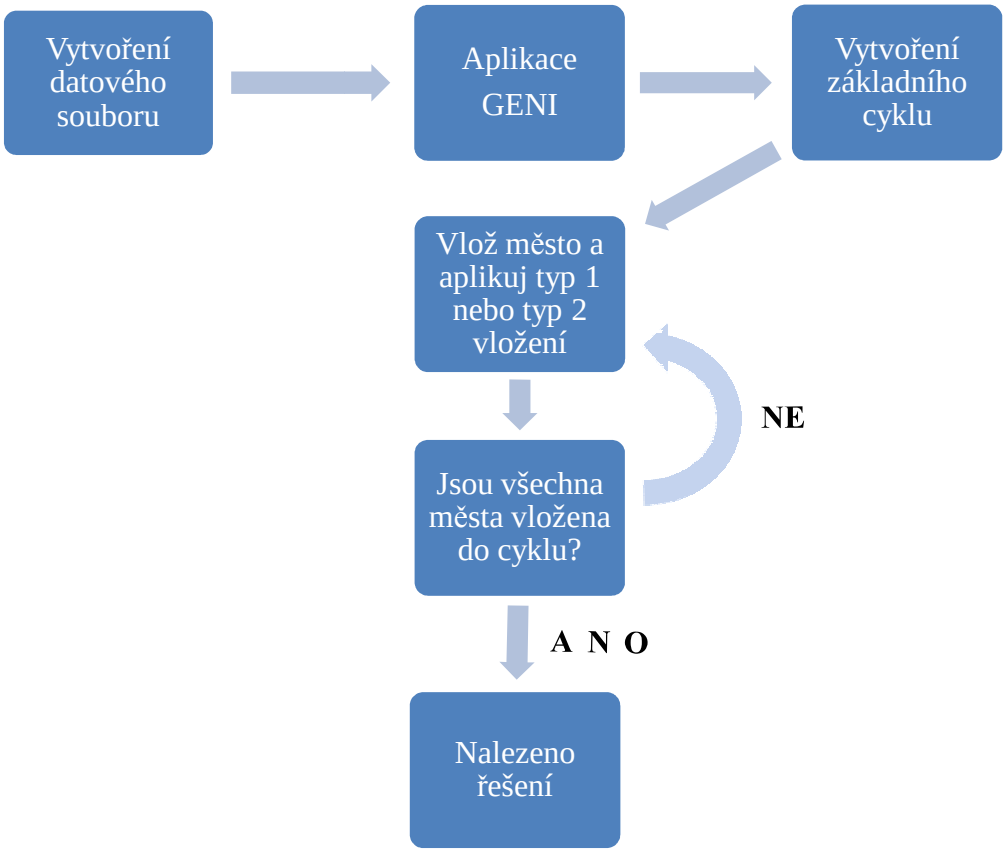
9.1.1 Vývojový diagram

Vývojový diagram mi slouží k zobrazení, jakým způsobem bude program postupovat při výpočtu úlohy. Vývojový diagram je rozdělen do dvou částí. První část je pro vkládací algoritmus GENI a druhý je pro post-optimalizaci US.

9.1.1.1 GENI

Pro vkládací algoritmus GENI je vývojový diagram zobrazen na obrázku 9-1. Vývoj začíná vytvořením základního souboru dat, který bude obsahovat výpočetní data. V dalším kroku bude definován algoritmus GENI, který obsahuje oba dva typy vložení vrcholů, tedy typ 1 a typ 2.

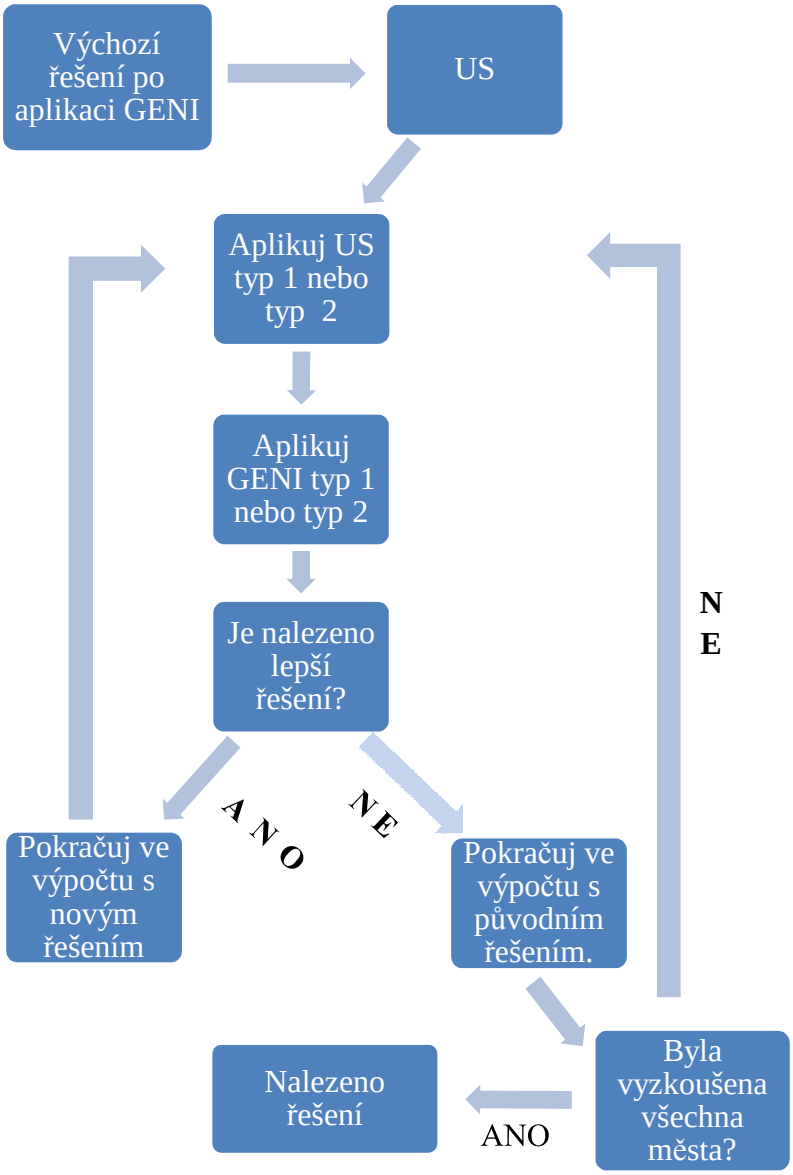
Ve třetím kroku program vybere 3 města a vytvoří z nich základní cyklus a vypočte celkovou délku. V každém dalším kroku pak bude vkládat postupně města a při každém vložení bude vyhodnocovat, jestli je výhodnější využít vkládací algoritmus typu 1, anebo typu 2. Nově nalezený cyklus si zapamatuje a následně vkládá další město a znova se vrací do čtvrtého kroku a zkoumá, jak nejlépe ho může vložit. Samozřejmostí je, že při vkládacím algoritmu vyhodnocuje cyklus po směru i proti směru hodinových ručiček. V momentě, kdy program již nemá žádné další město, která by se dalo vložit do cyklu, ukončí výpočet a zobrazí výsledek.



Obrázek 9-1: Vývojový diagram GENI

9.1.1.2 US

V momentě, kdy je dokončený algoritmus GENI, nastupuje post-optimalizační část US. Program si po aplikaci metody GENI pamatuje její výsledek a v každém kroku vybere jedno město, to pomocí US typu 1 nebo typu 2 odebere z cyklu a následně pomoci GENI typu 1 nebo typu 2 ho zpátky vloží. V případě, že nalezne lepší řešení, tak si toto řešení zapamatuje a počítá s ním, pokud ne, tak pokračuje s výpočtem podle původního řešení. Program odebírá a přidává body do takového momentu, dokud lze zlepšovat řešení, resp. dokud nevyzkouší všechna města a nezjistí, že již nelze zlepšovat řešení. V tomto momentě, kdy již nelze zlepšovat řešení, výpočet post-optimalizace končí a program zapíše výsledek.



Obrázek 9-2: Vývojový diagram post-optimalizace US

9.1.2 Vstupní data

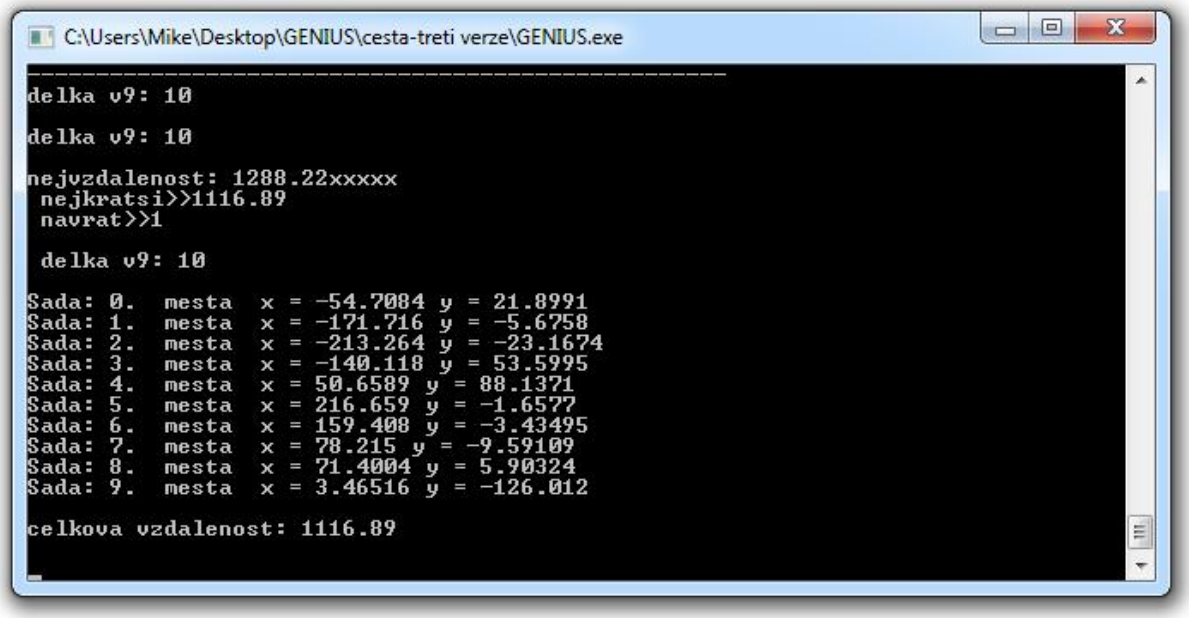
Program jako vstupní data využívá standardizovaný zápis. V první fázi je třeba vytvořit soubor, který se bude jmenovat *input.txt*, do kterého zapisuji všechna data. Do souboru se zapisují pouze souřadnice měst. Tyto souřadnice vycházejí z prostoru o dvou souřadnicích, prostoru 2D dimenzí. Budeme tedy uvažovat souřadnice x a y. Po vytvoření tohoto souboru zapisujeme do prvního sloupce všechny hodnoty souřadnice x, druhý sloupec je prázdný a třetí sloupec obsahuje hodnoty souřadnice y. Souřadnice mohou mít maximální délku 255 znaků a mohou obsahovat desetinná čísla a záporná čísla. Při psaní desetinných čísel je třeba využívat desetinnou tečku. Vstupní data tak mohou mít následující tvar.

159.40785 -3.43495
3.46516 -126.01189
71.40042 5.90324
216.65947 -1.65770
50.65890 88.13706
-213.26432 -23.16742
-140.11773 53.59949
78.21501 -9.59109
-171.71636 -5.67580
-54.70841 21.89906

Tabulka 10-1: Vstupní data do souboru myfile.txt

9.1.3 Výstupní data

Výstupní data jsou v základu ve formátu výstupu na základní obrazovce programu a také v souboru *output.txt*. Do souboru se ukládají výsledné souřadnice seřazených měst a celková vzdálenost. Na obrazovce lze vyčíst u názvu *nejvzdalenost* nejlepší řešení nalezené algoritmem GENI a pod *nejkratsi* je zobrazeno nejlepší dosažené řešení po dokončení post-optimalizačního algoritmu US. Pod těmito údaji je konečné řazení měst, resp. informace o vytvořeném cyklu. Dostanu tedy přesně seřazená města podle výpočetního algoritmu. Města jsou řazena s celkovou minimální vzdáleností. Jako poslední informaci, kterou program udává, je celková délka cyklu uvedená na konci pod názvem *celková vzdálenost*.



Obrázek 9-3: Výstupní data

10 Příklad

Pro ověření správného naprogramování, využiji několik příkladů, na kterých ukážu výsledky úlohy pomocí heuristiky GENIUS a také pomocí jednoho z Exaktních algoritmů, konkrétně budu vycházet z algoritmu uvedeného Dantzigem, Fulkersonem a Johnsonem (1954), kteří využili svůj algoritmus pro výpočet 42, resp. 49 měst. Tento algoritmus upravím tak, že použiju jinou smyčkovou podmínku, konkrétně Miller-Tucker-Zemlinovu podmínku. Pro výpočet pomocí tohoto algoritmu využiju program LINGO.

10.1 Příklad 1

V prvním příkladě využiji jako zdroj pro matici vzdáleností mapu České Republiky [16], ze které zvolím na začátek 10 měst. Města a jejich vzdálenosti jsou uvedeny v níže uvedené tabulce. Vzhledem k tomu, že uvažuju Symetrický problém obchodního cestujícího, bude tak i matice symetrická. Proto vzdálenost z Brna do Českých Budějovic bude stejná, jako když bych zvolil vzdálenost z Českých Budějovic do Brna.

	Brno	České Budějovice	Havlíčkův Brod	Hodonín	Hradec Králové	Cheb	Chomutov	Jihlava	Karlovy Vary	Kladno
Brno	0	194	96	58	151	373	302	85	331	218
České Budějovice	194	0	137	252	221	232	239	127	207	155
Havlíčkův Brod	96	137	0	154	84	287	216	25	245	132
Hodonín	58	252	154	0	209	431	360	143	389	276
Hradec Králové	151	221	84	209	0	287	216	109	245	132
Cheb	373	232	287	431	287	0	95	293	42	165
Chomutov	302	239	216	360	216	95	0	222	53	97
Jihlava	85	127	25	143	109	293	222	0	251	138
Karlovy Vary	331	207	245	389	245	42	53	251	0	123
Kladno	218	155	132	276	132	165	97	138	123	0

Tabulka 10-1: Tabulka vzdáleností v km

Pro výpočet pomocí exaktního algoritmu musím tuto matici upravit. Pokud bych uvažoval nulové vzdálenosti mezi dvěma stejnými městy, došlo by k nesprávnému výpočtu, protože metoda vždy vyhledává nejkratší vzdálenost a mohla by tak využít nulové vzdálenosti. Proto místo nulových hodnot dosadím nějaké velké číslo, v matematice se označuje např. M, my tam zadáme hodnotu např. 100 000km.

	Brno	České Budějovice	Havlíčkův Brod	Hodonín	Hradec Králové	Cheb	Chomutov	Jihlava	Karlovy Vary	Kladno
Brno	100 000	194	96	58	151	373	302	85	331	218
České Budějovice	194	100 000	137	252	221	232	239	127	207	155
Havlíčkův Brod	96	137	100 000	154	84	287	216	25	245	132
Hodonín	58	252	154	100 000	209	431	360	143	389	276
Hradec Králové	151	221	84	209	100 000	287	216	109	245	132
Cheb	373	232	287	431	287	100 000	95	293	42	165
Chomutov	302	239	216	360	216	95	100 000	222	53	97
Jihlava	85	127	25	143	109	293	222	100 000	251	138
Karlovy Vary	331	207	245	389	245	42	53	251	100 000	123
Kladno	218	155	132	276	132	165	97	138	123	100 000

Tabulka 10-2: Upravená tabulka vzdáleností v km pro výpočet

Pro výpočet souřadnicí jednotlivých měst je možné využít některý z širokého spektra programů, například program PCO [1], Microsoft Excel, který umí pracovat s eukleidovskou metrikou a převede nám matici vzdáleností na souřadnice měst x a y, které také potřebujeme k výpočtu.

Tytou souřadnice podle os x a y jsou uvedené v tabulce 10-3, kde je máme seřazené stejně, jako v tabulce 10-2. Souřadnice následně využiji v programu pro výpočet pomocí heuristiky GENIUS.

	Souřadnice			**Souřadnice**	
Města	**x**	**y**	**Města**	**x**	**y**
Brno	159.40785	-3.43495	Cheb	-213.26432	-23.16742
České Budějovice	3.46516	-126.01189	Chomutov	-140.11773	53.59949
Havlíčkův Brod	71.40042	5.90324	Jihlava	78.21501	-9.59109
Hodonín	216.65947	-1.65770	Karlovy Vary	-171.71636	-5.67580
Hradec Králové	50.65890	88.13706	Kladno	-54.70841	21.89906

Tabulka 10-3: Tabulky s (x,y) souřadnicemi měst pro výpočet

Pokud bych si chtěl ověřit správné vypočítání vzdálenosti, lze využít vzorec na výpočet vzdáleností podle eukleidovské metriky, tedy podle vzorečku:

$$d_{ij} = ((x_2 - x_1)^2 + (y_2 - y_1)^2)^{1/2}$$

kde x_1 jsou souřadnice prvního města a x_2 jsou souřadnice druhého města na ose x, a y_1 resp. y_2 jsou souřadnice prvního resp. druhého bodu na ose y.

Pro výpočet byl využit počítač Intel Core i7 930, 6GB RAM. Pro srovnání výpočetní náročnosti exaktního algoritmu bude spuštěna stejná úloha na počítači AMD Athlon 1GHz, 768MB RAM v programu LINGO. U exaktního algoritmu totiž uvidíme přesnou délku trvání výpočtu a tak si dokážeme, že exaktní algoritmy jsou velmi dlouhé a musí se provést velké množství iterací, než se dospěje k optimálnímu řešení. Oproti tomu ukážu, že heuristiky vypočítají úlohu za zlomek času.

Pro výpočet pomocí exaktního algoritmu využiji níže uvedené rovnice a metodu větvení a mezí.

$$\sum_{i=1}^{n}\sum_{j}^{n} c_{ij}x_{ij} \rightarrow min \qquad (10.1)$$

$$\sum_{j=1}^{n} x_{ij} = 1 \quad i = 1,2,\dots,n, \qquad (10.2)$$

$$\sum_{i=1}^{n} x_{ij} = 1 \quad j = 1,2,\dots,n, \qquad (10.3)$$

$$v_i - v_j + nx_{ij} \leq n - 1 \quad i = 1,2,\dots,n, j = 1,2,\dots,n, \; i \neq j, \qquad (10.4)$$

$$x_{ij} \in \{0,1\} \quad i = 1,2,\dots,n, \; j = 1,2,\dots,n. \qquad (10.5)$$

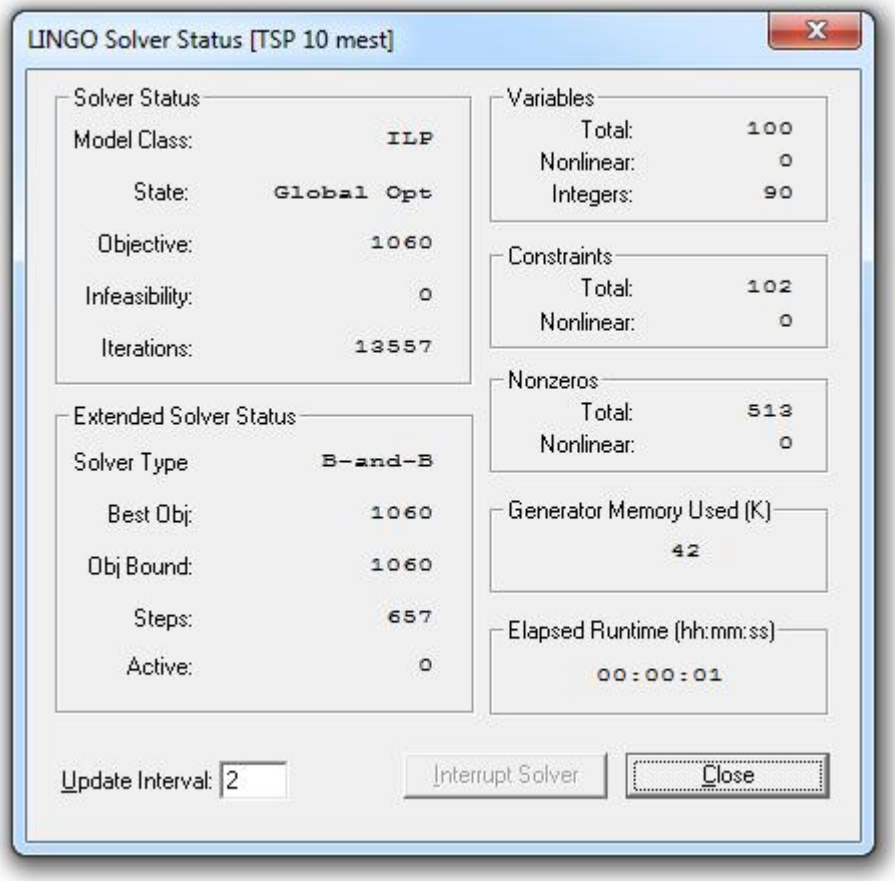
Rovnice (10.1) je účelová funkce, kterou budu minimalizovat. Rovnice (10.2) a (10.3) jsou lineární rovnice, které zabezpečí, že v matici výsledků, která má stejné rozměry jako matice se vzdálenostmi, bude v každém řádku a každém sloupci právě jedna jednička. Společně s omezením (10.4) a (10.5) budu mít zajištěnu tvorbu Hamiltonova cyklu za podmínek, že každým městem projedu právě jednou a vrátím se zpátky do výchozího bodu.

Model rovnice (10.1) - (10.5) jsem přepsal do LINGA. V první části je označení MODEL, to znamená, že se jedná o popis matematického modelu. V sekci SETS jsem značil uzly písmenem u a graf si definoval jako matici, ve které jsem si označil písmenem c matici vzdáleností a písmenem x matici, která bude obsahovat nuly nebo jedničky. Nulu, pokud nebude obsahovat hranu mezi dvěma městy a jedničku, pokud bude obsahovat hranu.

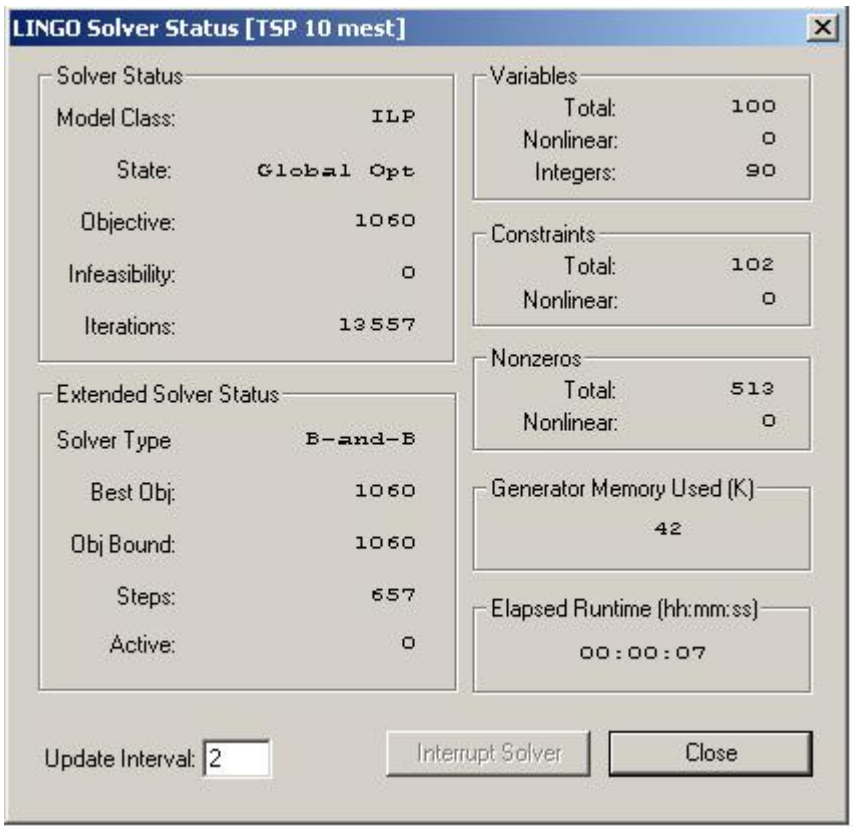
V další části modelu v sekci DATA jsem zapsal matici vzdáleností c a určil jsem si počet měst n. Nakonec v modelu zapisuju rovnice přepsané pomocí příkazů programu LINGO, které samozřejmě obsahují smyčkovou podmínku, abych zajistil, že každým městem projdu právě jednou.

```
MODEL:
SETS:
uzel/1..10/:u;
graf(uzel,uzel):c,x;
ENDSETS
DATA:
c=
100000	194	96	58	151	373	302	85	331	218
194	100000	137	252	221	232	239	127	207	155
96	137	100000	154	84	287	216	25	245	132
58	252	154	100000	209	431	360	143	389	276
151	221	84	209	100000	287	216	109	245	132
373	232	287	431	287	100000	95	293	42	165
302	239	216	360	216	95	100000	222	53	97
85	127	25	143	109	293	222	100000	251	138
331	207	245	389	245	42	53	251	100000	123
218	155	132	276	132	165	97	138	123	100000
;
n=10;
ENDDATA
min=@sum(graf(i,j):c(i,j)*x(i,j));
@for(uzel(j):@sum(uzel(i):x(i,j))=1);
@for(uzel(i):@sum(uzel(j):x(i,j))=1);
@for(graf(i,j)|i#ge#1#AND#j#gt#1#AND#i#ne#j:v(i)-v(j)+n*x(i,j)<=n-1);
@for(graf(i,j)|i#eq#j:x(i,j)=0);
@for(graf(i,j):@bin(x(i,j)));
END
```

Jako první výsledek uvedu výpočet pomocí exaktního algoritmu. Na následujícím obrázku je výstup z programu LINGO. Model Class označuje, že se jedná o ILP (Integer Linear Programming), neboli celočíselné lineární programování. State označuje, že jsem našel optimální řešení a Objective hodnotu účelové funkce, celkovou délku cyklu. Iterations označuje počet iterací, které program musel provést, aby vypočítal toto optimální řešení. Variables označuje počet proměnný, Constraints počet omezení. Elapsed time je měřič délky výpočetu úlohy.



Obrázek 10-1: Výpočetní výsledek pomocí programu LINGO na Intel Core i7 930 pod Windows 7



Obrázek 10-2: Výpočetní výsledek pomocí programu LINGO na Athlon 1GHz pod Windows XP

Z obrázku 10-1 a 10-2 je vidět rozdíl ve výpočetní náročnosti úlohy na základě využití různých počítačů a operačních systémů**.** Na obrázcích jsou zobrazeny počty iterací, které jsou třeba k úspěšnému výpočtu úlohy. Jedná se celkem o 13 557 výpočetních kroků, které vedly k výpočtu optimálního řešení, které je 1 060 km. Dalším zajímavým výsledkem z těchto obrázků je celková doba výpočtu, která je 1 sekunda, resp. 7 sekund. Na to, že je rozsah úlohy relativně malý, 10 měst, a v prvním případě byl využit na dnešní dobu velmi výkonný stroj, je výpočetní čas relativně krátký. Pokud ale budu uvažovat druhý stroj, který je 8 let starý, tak už výpočetní čas je dokonce sedminásobně vyšší. Před více než 20-ti lety, kdy ještě nebyly počítače tak výkonné, tak mohl výpočet této úlohy trvat několikasetnásobně více než na těchto strojích.

	Brno	České Budějovice	Havlíčkův Brod	Hodonín	Hradec Králové	Cheb	Chomutov	Jihlava	Karlovy Vary	Kladno
Brno	0	0	0	0	0	0	0	1	0	0
České Budějovice	0	0	0	1	0	0	0	0	0	0
Havlíčkův Brod	0	0	0	0	1	0	0	0	0	0
Hodonín	1	0	0	0	0	0	0	0	0	0
Hradec Králové	0	0	0	0	0	0	0	0	0	1
Cheb	0	1	0	0	0	0	0	0	0	0
Chomutov	0	0	0	0	0	0	0	0	1	0
Jihlava	0	0	1	0	0	0	0	0	0	0
Karlovy Vary	0	0	0	0	0	1	0	0	0	0
Kladno	0	0	0	0	0	0	1	0	0	0

Tabulka 10-4: Tabulka s výsledkem pomocí exaktního algoritmu

V tabulce je zobrazen výsledek této úlohy řešení exaktním algoritmem. Žlutě jsou označeny hrany, které budou využity při vytvoření Hamiltonova cyklu. Optimální řešení úlohy pomocí exaktního algoritmu je:

Hodonín - Brno - Jihlava - Havlíčku Brod - Hradec Králové - Kladno - Chomutov - Karlovy Vary - Cheb - České Budějovice - **Hodonín**

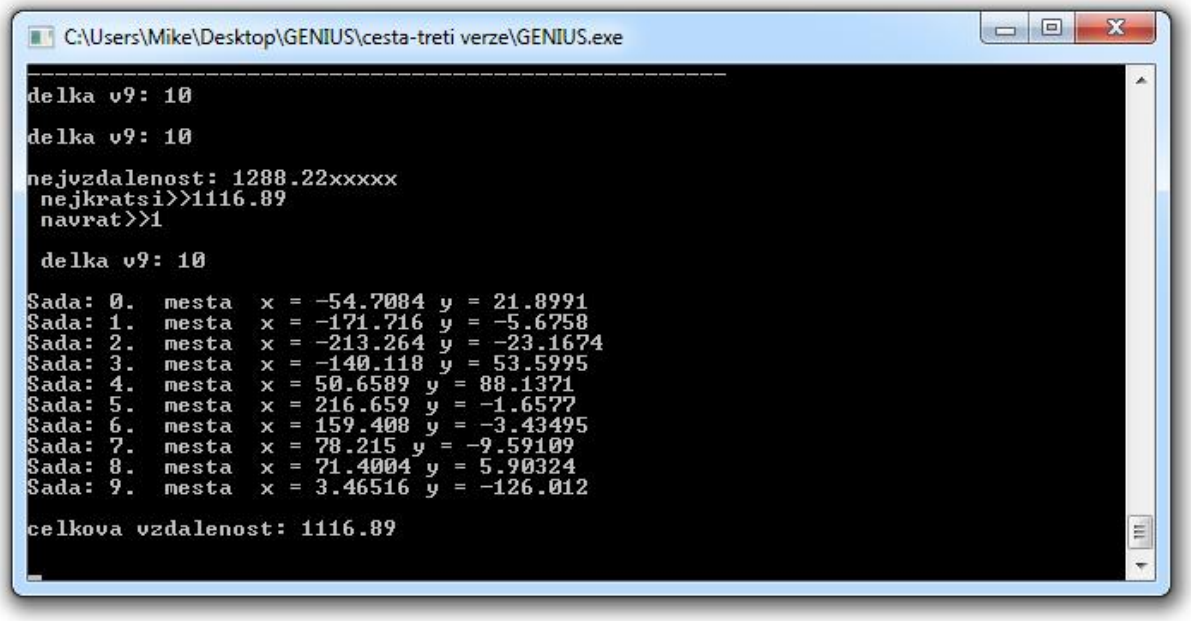
Nyní se podívám, jakým způsobem budu řešit stejnou příklad pomocí heuristiky GENIUS. Jako data pro výpočet pomocí této heuristiky, využiji souřadnice měst podle os x a y, které jsou uvedeny v tabulce 10-3. Tyto data si připravím do souboru, který si pojmenuju *input.txt* a data zařadím tak, že první sloupec bude obsahovat souřadnice x, druhý sloupec bude prázdný a ve třetím sloupci budou hodnoty souřadnice y. Pro ilustraci budou data souboru zadána způsobem zobrazeným v tabulce 10-5:

159.40785 -3.43495
3.46516 -126.01189
71.40042 5.90324
216.65947 -1.65770
50.65890 88.13706
-213.26432 -23.16742
-140.11773 53.59949
78.21501 -9.59109
-171.71636 -5.67580
-54.70841 21.89906

Tabulka 10-5: zadané hodnoty v souboru input.txt

Pokud zadám správně data, program bude umět tato data přečíst a následně vypočítá řešení úlohy. Na CD je uložen soubor *intput10.txt*, který obsahuje výpočetní data.

Jakmile spustím program, začne výpočet příkladu, který trvá řádově několik vteřin. Po výpočtu řešení dá program výsledek, kde jsou zobrazena města seřazená tak, aby byla celková trasa co nejmenší. Výsledek řešení je vidět na následujícím obrázku:



Obrázek 10-3: výpočet pomocí heuristiky GENIUS

Na obrázku 10-3 je vidět výstup z programu pro výpočet pomocí heuristiky GENIUS. Pro ulehčení práce je možné najít výsledek úlohy také v souboru *output.txt* v adresáři, kde se nalézá program GENIUS.exe. Tento soubor je také přiložen na CD s programem a je pojmenován *output10.txt*. Celková vzdálenost je 1 116,89 km a města jsou seřazena:

Kladno - Karlovy Vary - Cheb - Chomutov - Hradec Králové - Hodonín - Brno - Jihlava - Havlíčkův Brod - České Budějovice - Kladno

Pro srovnání s exaktním algoritmem mi sice heuristika GENIUS nedala optimální výsledek, ale pokud spustím program, tak výpočet touto heuristikou trval jednu vteřinu, kdežto exaktní algoritmus trval desítky vteřin. Pro porovnání, výpočet na počítači Intel Core i7 930, 6GB RAM trval stejně dlouhou dobu jako na počítači Athlon XP 1GHz, 768MB RAM, tedy 1 vteřinu. Výpočetní čas byl měřen pomocí programu na měření výpočetního času, který je součástí Microsoft Windows. Z toho důvodu lze konstatovat, že výpočetní náročnost pomocí této heuristiky je minimální a lze ji využívat na i na starších počítačích a tak získat v krátkém čase výsledky velmi blízké optimálnímu.

10.2 Příklad 2

Stejně jako v prvním příkladě využiju jako zdroj pro matici vzdáleností mapu České Republiky [16], ze které zvolíme na začátek 20 měst. Města a jejich vzdálenosti jsou uvedeny v níže uvedené tabulce. Vzhledem k tomu, že uvažuju Symetrický problém obchodního cestujícího, bude tak i matice symetrická. Proto vzdálenost z Brna do Českých Budějovic bude stejná, jako když bychom zvolili vzdálenost z Českých Budějovic do Brna.

Pro výpočet pomocí exaktního algoritmu i algoritmu GENIUS musím tuto matici upravit. Pokud bych uvažoval nulové vzdálenosti mezi dvěma stejnými městy, došlo by k nesprávnému výpočtu, protože metoda vždy vyhledává nejkratší vzdálenost a mohla byt tak využít nulové vzdálenosti. Proto místo nulových hodnot dosadím nějaké velké číslo, v matematice se označuje např. M, my tam zadáme hodnotu např. 100 000km.

Stejně jako v předchozím případě využiji nejprve exaktní algoritmus, který budu řešit metodou větvení a mezí.

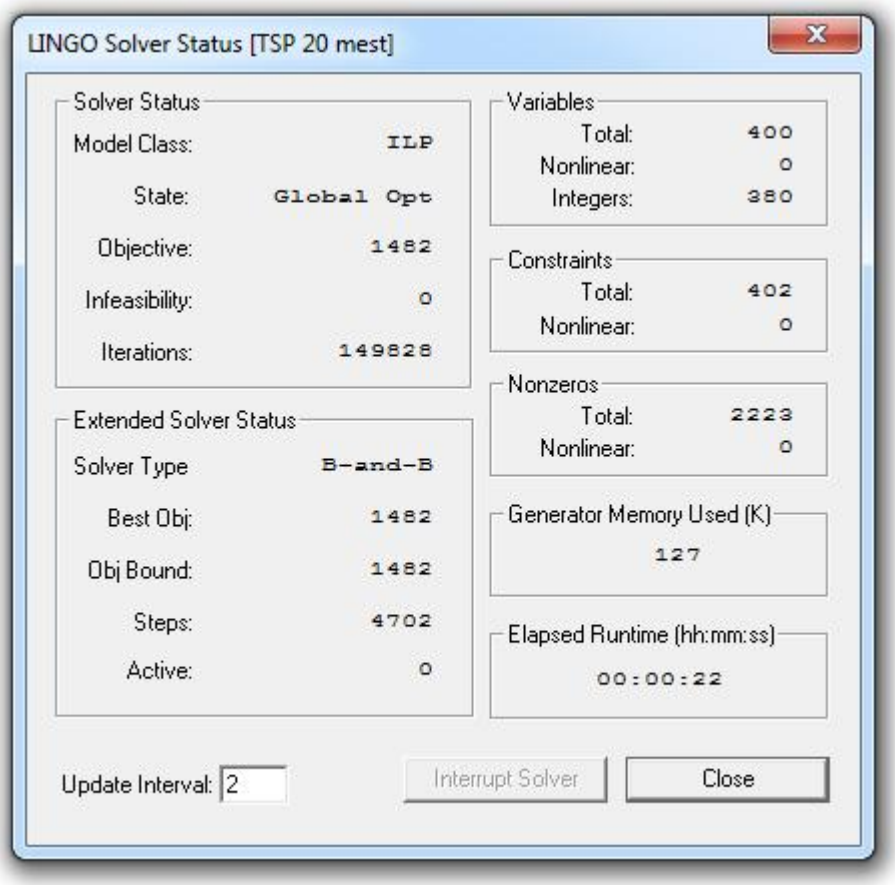
Zadání tohoto problému obchodního cestujícího do programu LINGO je stejné, jako v předchozím případě, proto ho již zde uvádět nebudu. Jediný rozdíl oproti Příkladu 1 je v počtu měst, kterých budu uvažovat celkem 20. Matici c s ohledem na velikost zde zadávat nebudu, lze ji jednoduše získat z matice vzdáleností uvedené v příloze 1.

Pro výpočet souřadnic z matice vzdáleností lze využít stejné programové vybavení, jako jsem uváděl v prvním příkladě. Takto se vypočtou souřadnice měst x a y, které využiju pro výpočet pomocí heuristiky GENIUS. V tabulce 10-7 jsou uvedeny souřadnice těchto měst.

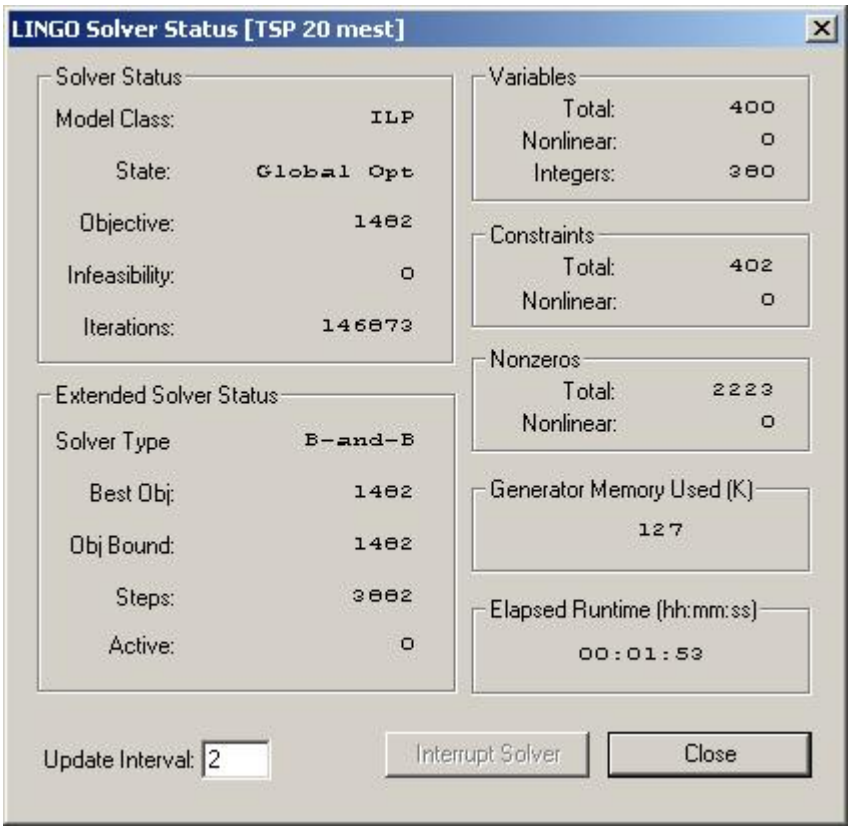
	Souřadnice			**Souřadnice**	
Města	**x**	**y**	**Města**	**x**	**y**
Brno	130.57071	-59.20612	Klatovy	-143.16628	-61.02346
České Budějovice	-53.49685	-118.69509	Liberec	-25.14516	121.58501
Havlíčkův Brod	48.54579	-23.84692	Mladá Boleslav	-34.80989	81.44962
Hodonín	163.56103	-92.88285	Olomouc	190.64315	17.23717
Hradec Králové	47.60223	74.69703	Opava	249.22116	55.49128
Cheb	-240.49001	0.12142	Ostrava	284.85214	8.19762
Chomutov	-158.38915	61.06993	Pardubice	54.52138	33.42288
Jihlava	39.95083	-55.95077	Písek	-68.74329	-84.65464
Karlovy Vary	-198.01599	12.80169	Plzeň	-142.93511	-13.01243
Kladno	-79.73642	23.78718	Praha	-64.54028	19.41146

Tabulka 10-6: Souřadnice měst

Pokud bych srovnal tabulku 10-6 s tabulkou 10-3, kde máme prvních 10 měst, lze si všimnout různých souřadnic. Je to dáno tím, že jsem vypočítával tyto souřadnice z jiného celkového počtu měst a proto jsem dostal různé souřadnice.



Obrázek 10-4: Výpočetní výsledek pomocí programu LINGO na Intel Core i7 930 pod Windows 7



Obrázek 10-5: Výpočetní výsledek pomocí programu LINGO na Athlon 1GHz pod Windows XP

Z obrázku 10-4 a 10-5 je vidět rozdíl ve výpočetní náročnosti úlohy na základě využití různých počítačů a operačních systémů. Na obrázcích jsou zobrazeny počty iterací, které jsou třeba k úspěšnému výpočtu úlohy. Jedná se celkem o 149 828, resp. 146 873 výpočetních kroků, které vedly k výpočtu optimálního řešení, které je 1 482 km. K pozastavení stojí fakt, že obě dvě úlohy mají stejné optimální řešení, ale u každé úlohy stačil různý počet iterací k výpočtu. Je to dáno metodou větvení a mezí, která zajišťuje prohledávání uzlů do doby, než nalezne optimální řešení. Proto mohly oba dva počítače volit na základě svých instrukčních sad jiné uzly, přes které se snažili vypočítat optimální řešení.

Dalším zajímavým výsledkem z těchto obrázků je celková doba výpočtu, která je 22 sekund, resp. 1 minuta a 53 sekund. Když porovnáme první a druhou úlohu, pro výpočet druhé úlohy jsem měl dvojnásobný počet měst, ale časová náročnost rostla mnohonásobně více. U prvního počítače vyrostla dvaadvaceti násobně a u druhého šestnáctinásobně. Zde je vidět, že pokud bych ještě dále rozšířil úlohu, tak na starším počítači by se mohla řešit více než několik minut.

V tabulce je zobrazen výsledek této úlohy řešení exaktním algoritmem. Žlutě jsou označeny hrany, které budou využity při vytvoření Hamiltonova cyklu. Optimální řešení úlohy pomocí exaktního algoritmu je:

Hodonín - Brno - Jihlava - Havlíčkův Brod - České Budějovice - Písek - Klatovy - Plzeň - Cheb - Karlovy Vary - Chomutov - Kladno - Praha - Mladá Boleslav - Liberec - Hradec Králové - Pardubice - Olomouc - Opava - Ostrava - **Hodonín**

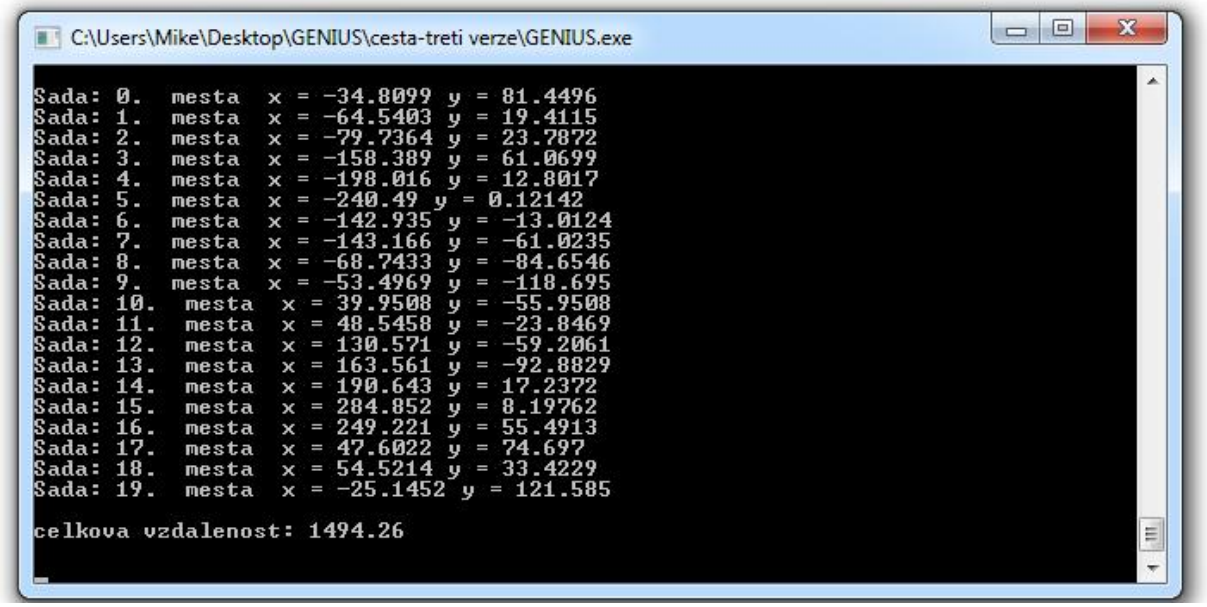
Nyní ukážu, jakým způsobem budu řešit stejný příklad pomocí heuristiky GENIUS. Jako data pro výpočet pomocí této heuristiky, využiji souřadnice měst podle os x a y, které mám uvedeny v příloze 2. Tyto data si připravím do souboru, který si pojmenuju *input.txt* a data zařadím tak, že první sloupec bude obsahovat souřadnice x, druhý sloupec bude prázdný a ve třetím sloupci budou hodnoty souřadnice y. Pro ilustraci budou data souboru zadána způsobem zobrazeným v tabulce 10-7:

130.57071 -59.20612
-53.49685 -118.69509
48.54579 -23.84692
163.56103 -92.88285
47.60223 74.69703
-240.49001 0.12142
-158.38915 61.06993
39.95083 -55.95077
-198.01599 12.80169
-79.73642 23.78718
-143.16628 -61.02346
-25.14516 121.58501
-34.80989 81.44962
190.64315 17.23717
249.22116 55.49128
284.85214 8.19762
54.52138 33.42288
-68.74329 -84.65464
-142.93511 -13.01243
-64.54028 19.41146

Tabulka 10-7: zadané hodnoty v souboru input.txt

Pokud zadám správně data, program bude umět tato data přečíst a následně vypočítá řešení úlohy. Na CD je uložen soubor *intput20.txt*, který obsahuje výpočetní data.

Jakmile spustím program, začne výpočet příkladu, který trvá řádově několik vteřin. Po výpočtu řešení nám program dá výsledek, kde jsou zobrazena města seřazená tak, aby byla celková trasa co nejmenší. Výsledek řešení je vidět na následujícím obrázku:



Obrázek 10-6: výpočet pomocí heuristiky GENIUS

Na obrázku 10-6 je vidět výstup z programu pro výpočet pomocí heuristiky GENIUS. Pro ulehčení práce je možné najít výsledek úlohy také v souboru *output.txt* v adresáři, kde se nalézá program GENIUS.exe. Tento soubor je také přiložen na CD s programem a je pojmenován *output20.txt*. Celková vzdálenost je 1 494,26 km a města jsou seřazena:

Mladá Boleslav - Praha - Kladno - Chomutov - Karlovy Vary - Cheb - Plzeň - Klatovy - Písek - České Budějovice - Jihlava - Havlíčkův Brod - Brno - Hodonín - Olomouc - Ostrava - Opava - Hradec Králové - Pardubice - Liberec - **Mladá Boleslav**

Pro srovnání s exaktním algoritmem nám sice heuristiky GENIUS nedala optimální výsledek, ale pokud spustíme program, tak výpočet touto heuristikou trvá 2 vteřiny, kdežto exaktní algoritmus trval desítky vteřin a na pomalejším počítači se výpočetní čas blížil k několika minutám. Výsledek však oproti exaktnímu algoritmu není až tolik odlišný.

Pro porovnání, výpočet na počítači Intel Core i7 930, 6GB RAM trval stejně dlouhou dobu jako na počítači Athlon XP 1GHz, 768MB RAM. Výpočetní čas byl měřen pomocí programu na měření výpočetního času, který je součástí Microsoft Windows. Z toho důvodu lze konstatovat, že výpočetní náročnost pomocí této heuristiky je minimální a lze ji využívat na i na starších počítačích a tak získat v krátkém čase výsledky velmi blízké optimálnímu.

Pokud bych srovnal výsledky první a druhého příkladu, je vidět, že heuristika GENIUS dává lepší výsledky při výpočtu většího počtu měst a dokonce se její výsledky více přibližují optimálnímu řešení. Proto je výhodnější pro menší úlohy využívat exaktní algoritmy a pro složitější úlohy využít heuristiky.

Závěr

Cílem diplomové práce bylo vysvětlit základní vlastnosti Problému obchodního cestujícího, jeho historii a výpočetní metody. Hlavní důraz byl kladen na heuristickou metodu výpočtu Problému obchodního cestujícího, a to heuristiku GENIUS. Heuristiku GENIUS jsem nejprve obecně popsal a následně rozdělil na dvě samostatné části, mezi kterými existuje propojení jejich výpočetních kroků. Tento algoritmus GENIUS jsem rozdělil na dva samostatné algoritmy. Prvním byl vkládací algoritmus GENI, který jsem nejprve rozdělil na dva typy výpočetních algoritmů a následně popsal jak matematicky, tak i pomocí Grafů. Druhý algoritmus, který jsem popisoval, byl post-optimalizační algoritmus US, který zajistil dodatečné post-optimalizační výpočty, které zajistili, že řešení se blíží optimálnímu. Algoritmus US jsem také rozdělil na dva samostatné typy odebírání měst. Algoritmus US po odebrání města následně pokračoval přidáváním měst pomocí metody GENI.

Jako hlavní přínos mojí diplomové práce vidím naprogramování programu, který využívá algoritmus GENIUS. K tomuto programu jsem vytvořil uživatelský manuál, podle kterého jakýkoliv uživatel může použít tento algoritmus a využít jeho výhody. Program jsem nakonec vyzkoušel na dvou příkladech. Výsledek výpočtu pomocí programu GENIUS jsem následně porovnal s výsledky při využití exaktních algoritmů. Při srovnání výsledků jsem zjistil, že výsledky se liší minimálně, avšak výpočetní čas úlohy byl pomocí programu GENIUS několikanásobně kratší.

Na základě možnosti otestování tohoto programu a porovnání s exaktními algoritmy, vyplývá, že heuristika GENIUS dává velmi dobré výsledky za zlomek času a její využití pro větší rozměr úloh je tedy opodstatněný.

Na základě získaných výsledků je také možné program využívat pro různé typy firem, které řeší možnosti zefektivnění procesů. Tento program jim umožní zredukovat náklady spojené převážně s odděleními obchodu a logistiky.

Program GENIUS vytvořený v jazyce C++ je součástí této diplomové práce a je přiložen na CD společně se zadáním a výsledky obou příkladů.

Literatura

[1] ANDERSON, M.J., PCO: a FORTRAN computer program for principal coordinate analysis, Department of Statistics, University of Auckland, New Zealand, 2003

[2] APPLEGATE DAVID L., BIXBY ROBERT E, CHVÁTAL VAŠEK, COOK WILLIAM J., The travelling Salesman Person, Princeton University Press, 2006, ISBN 978-0-691-12993-8

[3] COOK, WILLIAM, http://www.tsp.gatech.edu/history/index.html , Georgia Technology University, poslední aktualizace květen 2010

[4] COUFAL J., KLŮFA J., Matematika pro ekonomické fakulty 1, Ekopress 2000, ISBN 80-86119-30-0

[5] DANTZIG G, FULKERSON R, JOHNSON S, Solution of a large-scale Travelling Salesman Problem, The Rand Corporation, Santa Monica, California, (Received August 9, 1954)

[6] FIALA PETR, Řízení projektů, Nakladatelství VŠE, 2008, ISBN 978-80-245-1413-0

[7] GASS SAUL I, ASSAD ARJANG A., An Annotated Timeline of Operation Research An Informal History, Kluwer Academic Publishers 2008

[8] GENDREAU MICHEL, HERTZ ALAIN, LAPORTE GILBERT, New insertion and post-optimalization procedures for the travelling salesman problem, Operation Research. Vol. 40, No. 6, November-December 1992, strana 1086 - 1094,

[9] GUTIN G., PUNNEN A.P., (2002), The Travelling Salesman Problem and Its Variations, Kluwer Academics Publisher

[10] HELSGAUN KELD, An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic, Roskilde University

[11] CHONG, YEN NIE, Heuristic Algorithm for Routing Problems, Curtin University of Technology, September 2001

[12] JOHNSON S., MCGEOCH LYLE A., The Traveling Salesman Problem: A Case Study in Local Optimization, November 1995

[13] Journal of Statistical Software, December 2007, Volume 23, Issue 2.

[14] JÜNGER MICHEL, LIEBLING THOMAS, NADDEF DENIS a kolektiv, Integer Programming 1958 - 2008, Springer 2008, ISBN 978-3-540-68274-5

[15] KORTE BERNHARD, VYGEN JENS, Combinatorial Optimization Theory and Algorithms, Springer 2000, ISBN 3-540-43154-3

[16] MAREK MILOŠ, HLAVÁČEK RADEK, HOLKOVIČOVÁ BLANKA, ZAHAJSKÝ PAVEL, Autoatlas, Kartografie PRAHA, a.s., 2009, ISBN 978-80-7393-104-9

[17] PATAKI, G, The bad and the good-and-ugly: formulations for the travelling salesman problem, Columbia University, 2000

[18] PELIKÁN JAN, Diskrétní modely v operačním výzkumu, Professional Publishing, 2001

Přílohy

Příloha 1 - Tabulka s maticí vzdáleností pro výpočet úlohy s 20 městy

Příloha 2 - Tabulka s řešením úlohy s 20 městy

Příloha 3 - CD s programem GENIUS a vstupními daty pro výpočet řešených příkladů a výstupy řešených příkladů

Příloha 1

	Brno	České Budějovice	Havlíčkův Brod	Hodonín	Hradec Králové	Cheb	Chomutov	Jihlava	Karlovy Vary	Kladno	Klatovy	Liberec	Mladá Boleslav	Olomouc	Opava	Ostrava	Pardubice	Písek	Plzeň	Praha
Brno	0	194	96	58	151	373	302	85	331	218	286	247	234	92	166	160	126	210	280	200
České Budějovice	194	0	137	252	221	232	239	127	207	155	106	239	192	286	360	354	200	52	124	137
Havlíčkův Brod	96	137	0	154	84	287	216	25	245	132	200	180	128	143	217	236	63	124	219	114
Hodonín	58	252	154	0	209	431	360	143	389	276	344	305	292	150	224	218	184	268	338	258
Hradec Králové	151	221	84	209	0	287	216	109	245	132	236	96	83	146	205	239	21	223	194	114
Cheb	373	232	287	431	287	0	95	293	42	165	150	275	228	433	492	525	308	180	108	173
Chomutov	302	239	216	360	216	95	0	222	53	97	178	168	157	362	421	455	237	208	136	102
Jihlava	85	127	25	143	109	293	222	0	251	138	190	205	153	177	251	245	88	114	186	120
Karlovy Vary	331	207	245	389	245	42	53	251	0	123	125	221	186	391	450	484	266	155	83	131
Kladno	218	155	132	276	132	165	97	138	123	0	140	120	73	278	337	371	153	127	98	18
Klatovy	286	106	200	344	236	150	178	190	125	140	0	224	177	343	417	436	228	76	42	122
Liberec	247	239	180	305	96	275	168	205	221	120	224	0	47	242	301	335	117	211	182	102
Mladá Boleslav	234	192	128	292	83	228	157	153	186	73	177	47	0	229	288	322	104	164	135	55
Olomouc	92	286	143	150	146	433	362	177	391	278	343	242	229	0	74	93	144	267	330	250
Opava	166	360	217	224	205	492	421	251	450	337	417	301	288	74	0	34	226	341	399	319
Ostrava	160	354	236	218	239	525	455	245	484	371	436	335	322	93	34	0	237	360	433	353
Pardubice	126	200	63	184	21	308	237	88	266	153	228	117	104	144	226	237	0	187	186	106
Písek	210	52	124	268	223	180	208	114	155	127	76	211	164	267	341	360	187	0	72	109
Plzeň	280	124	219	338	194	108	136	186	83	98	42	182	135	330	399	433	186	72	0	80
Praha	200	137	114	258	114	173	102	120	131	18	122	102	55	250	319	353	106	109	80	0

Tabulka 1: Tabulka vzdáleností pro 20 měst

Příloha 2

	Brno	České Budějovice	Havlíčkuv Brod	Hodonín	Hradec Králové	Cheb	Chomutov	Jihlava	Karlovy Vary	Kladno	Klatovy	Liberec	Mladá Boleslav	Olomouc	Opava	Ostrava	Pardubice	Písek	Plzeň	Praha
Brno	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
České Budějovice	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
Havlíčkuv Brod	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Hodonín	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Hradec Králové	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
Cheb	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
Chomutov	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
Jihlava	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Karlovy Vary	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Kladno	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Klatovy	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
Liberec	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Mladá Boleslav	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
Olomouc	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
Opava	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
Ostrava	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Pardubice	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
Písek	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
Plzeň	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Praha	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

Tabulka 2: Tabulka s výsledkem pomocí exaktního algoritm