VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

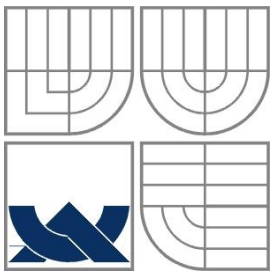
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

POROVNANIE METÓD NA RIEŠENIE PROBLÉMU OBCHODNÉHO CESTUJÚCEHO

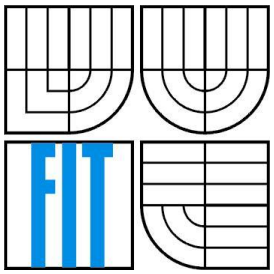
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE LUCIA ŠUŠOVÁ
AUTHOR

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

SROVNÁNÍ METOD PRO ŘEŠENÍ PROBLÉMU OBCHODNÍHO CESTUJÍCÍHO

COMPARISON OF METHODS FOR TRAVELLING SALESMAN PROBLEM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

LUCIA ŠUŠOVÁ

VEDOUCÍ PRÁCE
SUPERVISOR

ING. JAROSLAV ROZMAN

BRNO 2011

Abstrakt

Tato práce se zabývá srovnáním metod řešení problému obchodního cestujícího (traveling salesman problem). Pro řešení tohoto NP-úplného problému existuje celá řada algoritmů, kdy není jednoduché vybrat ten správný. Hlavní přínos této práce tkví v experimentálním srovnání jednotlivých metod mezi sebou. Čtenář se tak dozví, jaké výsledky pří hledání cesty může očekávat při použití konkrétního algoritmu.
První část práce se zabývá teoretickým základem, kdy jsou popsány všechny potřebné informace pro správně pochopení problému. Druhá část se zabývá popisem jednotlivých heuristik a metod řešení rozdělených do kategorií podle principu činnosti. Dále práce obsahuje experimentální srovnání metod. Toto porovnávání bylo prováděno na základě vlastní implementace jednotlivých heuristik, část práce se věnuje také samotné implementaci metod a popisu programu. Na závěr jsou uvedeny možnosti dalšího vývoje projektu a nechybí ani zhodnocení výsledků.

Abstract

This work is about comparison of methods for solving the traveling salesman problem. There are many algorithms for finding solution of this NP complete problem but it is not easy to choose the right one. Main goal of this thesis is experimental methods comparison between each other. Reader is going to learn what result she can expect if she chooses certain algorithm for finding the path.
First part is focused on theoretical basics where is described all needed information for understanding the problem. Second part describes single heuristics and methods for solving these problems. The methods are divided into groups by principle of working. Next part contains experimental comparison of methods. This comparison was done based on own implementation of single heuristics. The following part of this work contains information about this implementation and also describes comparison application. Next possible steps of this project are described in conclusion.

Klíčová slova

Problém obchodního cestujícího, NP-úplný problém, teorie grafů, heuristiky

Keywords

Traveling salesman problem, NP-complete problem, graph theory, heuristics

Citace

Šušová Lucia: Porovnanie metód na riešenie problému obchodného cestujúceho, bakalářská práce, Brno, FIT VUT v Brně, 2011

Porovnanie metód na riešenie problému obchodného cestujúceho

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně pod vedením Ing. Jaroslava Rozmana.
Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

……
Lucia Šušová
Datum 18.5.2011

Poděkování

Rada by som poďakovala vedúcemu Ing. Jaroslavovi Rozmanovi za vedenie práce, odborné rady a konzultácie.

Obsah

1 Úvod

Problém obchodného cestujúceho (*TSP* z angl. *Traveling Salesman Problem*) patrí v súčasnosti medzi najznámejšie optimalizačné úlohy. Je to aj vďaka veľkému počtu jeho praktických aplikácií. Ide o problémy súvisiace s vytvorením optimálnej postupnosti cyklických technologických operácií. Napríklad: vŕtanie dier do matičnej dosky (posun vrtáka po matičnej doske), vykreslenie plošných spojov (mechanický plotter zakresľuje všetky spoje na doske, je potrebné minimalizovať čas vykreslenia týchto spojov), osadzovanie súčiastok plošného spoja a pod. Optimálne riešenie ušetrí čas a s ním spojené peniaze. Nájsť ho však nie je veľmi jednoduché.

TSP patrí medzi NP-úplné úlohy. To znamená, že súbor možných riešení je konečný, ale so zväčšujúcim sa množstvom vstupných údajov čas potrebný na výpočet exponenciálne rastie. Z toho vyplýva neriešiteľnosť úlohy algoritmom v polynomiálnom čase.

Z týchto dôvodov sa hľadali skôr približné riešenia, ktorých výsledky by síce mali istú odchýlku od optimálnosti, ale dosahovali by prakticky použiteľných časov. Túto podmienku splňujú heuristické algoritmy riešenia TSP, ktoré zaručujú pomerne malé odchýlky od optimálneho riešenia v prijateľnom čase.

Práve tieto heuristiky sú predmetom tejto práce. Mojou úlohou bolo nájsť metódy riešenia problému obchodného cestujúceho, poskytnúť ich prehľad, ich vzájomné porovnanie a vyhodnotenie. Porovnanie je založené na výsledkoch testu naprogramovaných heuristík.

2 Teória grafov

Aby sme boli schopní pochopiť základné myšlienky heuristík, ktoré riešia problém obchodného cestujúceho, a mali lepší prehľad o danej problematike, je potrebné oboznámiť sa so základnými pojmami teórie grafov. Definície a pojmy v tejto kapitole sú prevzaté z [1], [2], [3]. Podrobnejšie informácie o teórii grafov je možné nájsť v spomínanej literatúre.

2.1 Základné definície a pojmy

Pri riešení problému obchodného cestujúceho sa nezaobídeme bez práce s grafmi. *Graf* je definovaný ako usporiadaná dvojica:

- $G = \{V, H\}$

kde V je neprázdna množina vrcholov a H je množina hrán. Graf je základným objektom *teórie grafov*.

Definícia 2.1.1. Graf G sa skladá z množiny V *vrcholov (uzlov)* a množiny H *hrán* tak, že každá hrana $h \in H$ je priradená neusporiadanej dvojici (tj. dvojprvkovej množine) vrcholov $u, v \in V$. Ak existuje jediná hrana $h \in H$ priradená dvojici vrcholov $u, v \in V$, píšeme $h \equiv \{u, v\}$. Keď $h = \{u, v\}$, o hrane h sa hovorí, že je incidentná s vrcholmi u a v alebo *spája* vrcholy u a v.

Jednotlivé vrcholy môžu byť navzájom spojené hranou. Hrana $h \in H$, ktorá spája vrcholy v_i a v_j sa označuje (v_i, v_j) a dané vrcholy v_i a v_j spojené touto hranou sa nazývajú *susedné*. Obecne môže byť jednej dvojici vrcholov priradených viac hrán, tieto hrany sa potom nazývajú *násobné*. Ak hovoríme o hrane, ktorá začína a končí v tom istom vrchole $h = \{u, u\}$ hovoríme o *slučke.*

V teórii grafov poznáme rôzne druhy grafov. Základ problému obchodného cestujúceho je postavený na *neorientovanom grafe*.

Neorientovaný graf je usporiadaná trojica:

- $G = \{V, H, \rho\}$

kde V je neprázdna konečná množina všetkých vrcholov, H je konečná množina všetkých hrán a ρ je *incidenčná relácia*. Presnejšie: ρ je zobrazenie množiny H do množiny V také, že platí:

- $1 \leq |\rho(h)| \leq 2 \qquad \forall h \in H$

Ako sme už spomínali, poznáme rôzne typy grafov a každý typ grafu je niečím charakteristický. V tejto práci sa zaoberáme úzkym okruhom grafov, a preto si následne vysvetlíme tie typy grafov, s ktorými budeme pri riešení TSP pracovať.

Neorientovaný graf o n vrcholoch, ktorý obsahuje práve jednu hranu medzi každou dvojicou vrcholov je *kompletný graf*, označovaný ako K_n. V kompletnom grafe s počtom vrcholov $n \geq 3$ máme zaručenú existenciu hamiltonovského cyklu, zatiaľ čo vo všeobecnom grafe je nájdenie

hamiltonovského cyklu NP-ťažký problém. Z tohto dôvodu budeme pri riešení TSP pracovať s kompletnými grafmi.

Pre neorientovaný graf $G = \{V, H\}$ môžeme ďalej definovať *hranové ohodnotenie* c:

- $G = \{V, H, c\}$,

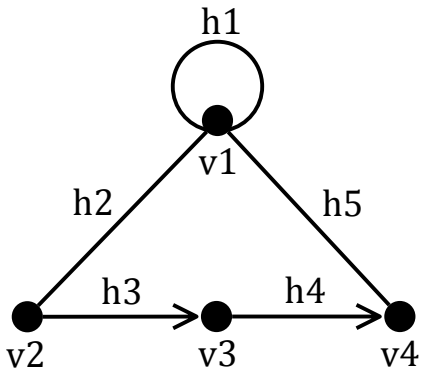
kde c je reálnou funkciou:

- $c: E \to \mathbb{R}$,

ktorá každej hrane (v_i, v_j) priradí reálne číslo c_{ij} nazývané *ohodnotenie hrany* (cena hrany). V konkrétnych prípadoch ohodnotenie hrán nazývame rôzne: dĺžka hrany, priepustnosť hrany a pod. V našom prípade ohodnotenie hrany predstavuje vzdialenosť medzi mestami.

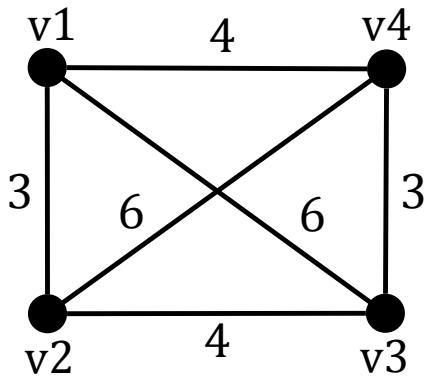
Definícia 2.1.2 *Stupeň vrcholu* v neorientovanom grafe G je rovný počtu hrán s ním incidentných, pričom každá slučka sa počíta dvakrát. Stupeň vrcholu v sa označuje $\deg(v)$. Ak má daný graf G všetky vrcholy rovnakého stupňa, nazýva sa *pravidelný*.

Pre ľahšiu orientáciu sú použité pojmy znázornené na obrázkoch 2.1 a 2.2.



v_1 - vrchol
h_1 - slučka
h_2, h_5 - neorientované hrany
h_3, h_4 - orientované hrany
$\rho(h_2) = \{v_1, v_2\}$

Obrázok 2.1: Názorný graf



$\{v_1, v_2\}$ - ohodnotená hrana
$\deg(v_1)$ - vrchol stupňa 3

Obrázok 2.2: Kompletný súvislý graf

Postupnosť uzlov a hrán neorientovaného grafu $G = \{V, H, \rho\}$ tvaru:

- $S = (v_0, h_1, v_1, h_2, \dots, v_{n-1}, h_n, v_n)$, kde $h_i \in H$, $\rho(h) = \{v_{i-1}, v_i\}$, $i = 1, \dots, n$,

sa nazýva *sled* v grafe G medzi uzlami v_0, v_n. Číslo n sa nazýva *dĺžka sledu* S.

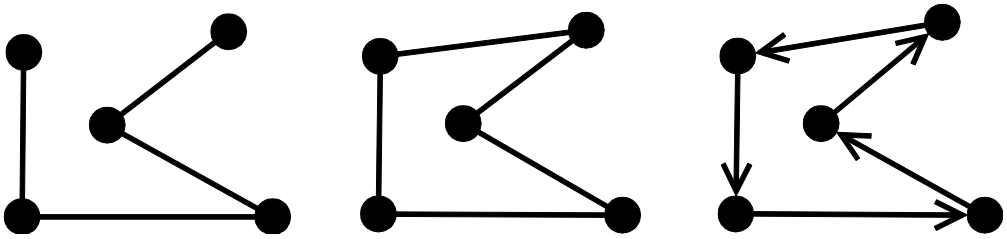
Ak pre hrany sledu platí, že $h_i \neq h_j$, pre $i \neq j$, hovoríme o *ťahu*. Niektoré heuristiky riešenia TSP používajú tzv. *eulerovský ťah*. Tento termín zaviedol L. Euler v roku 1736. Grafy, ktoré obsahujú uzavretý eulerovský ťah nesú názov *eulerovské grafy* a dokážeme ich nakresliť jedným ťahom.

Ak pre vrcholy sledu platí $v_i \neq v_j$, hovoríme o *ceste*. V prípade, že pre počiatočný a koncový vrchol cesty platí $v_0 = v_n$, hovoríme o *kružnici*. Kružnica je pravidelný graf, ktorý má všetky vrcholy stupňa 2.

TSP je z matematického hľadiska problém nájdenia *hamiltonovského cyklu*. *Cyklus* je orientovaný uzavretý sled ($v_0 = v_n$), v ktorom sa žiadny vrchol okrem $v_0 = v_n$ nevyskytuje 2-krát. Každý cyklus je zároveň aj kružnicou, preto ďalej budeme hovoriť len o cykle.

Algoritmy, ktoré riešia problémy najkratšej cesty medzi dvoma ľubovoľnými vrcholmi grafu a eulerovského ťahu sú popísané v [2].

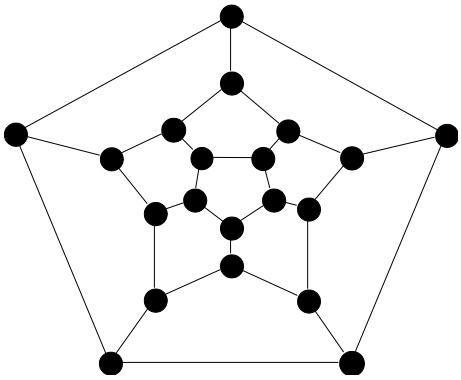
Graf G nazveme *súvislý* ak medzi jeho ľubovoľnými dvoma vrcholmi existuje sled.



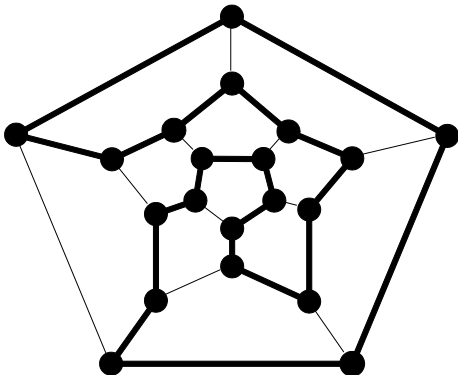
Obrázok 2.3: Súvislé grafy - cesta, kružnica a cyklus

2.1.1 Hamiltonovské grafy

V roku 1859 sa slávny írsky matematik W. R. Hamilton zaoberal hrou „cesta okolo sveta“, ktorá je zobrazená na obrázku 2.4. Úlohou bolo precestovať všetkými hlavnými mestami (vrcholmi pravidelného dvanásťstena) a vrátiť sa naspäť domov (počiatočný vrchol cesty). Takto vzniklo niekoľko problémov. Jedným z nich bolo nájsť „okružnú cestu“ – t.j. cyklus, ktorý obsahuje všetky vrcholy grafu. Jedno z možných riešení je na obrázku 2.5.



Obrázok 2.4



Obrázok 2.5

Hoci hra pre svoju jednoduchosť nemala veľký úspech, na počesť svojho objaviteľa sa cyklus, ktorý obsahuje všetky vrcholy grafu nazýva *hamiltonovský*.

Na základe týchto znalostí môžeme problém obchodného cestujúceho formulovať ako problém nájdenia hamiltonovského cyklu, ktorého c - dĺžka (súčet dĺžok prejdených hrán) je minimálna.

Platí, že každý kompletný graf K_n, kde $n \geq 3$, obsahuje hamiltonovskú kružnicu. Preto pri riešení TSP budeme pracovať s kompletnými grafmi v 2D rovine, pri ktorých platí trojuholníková nerovnosť pre ktorékoľvek tri vrcholy i, j a k:

- $c_{ij} \leq c_{ik} + c_{kj}$

a kde vzdialenosť medzi jednotlivými vrcholmi sa rovná euklidovskej vzdialenosti [2] daných bodov. Takéto grafy sa nazývajú *euklidovskými grafmi*.

2.1.2 Stromy a kostry grafov

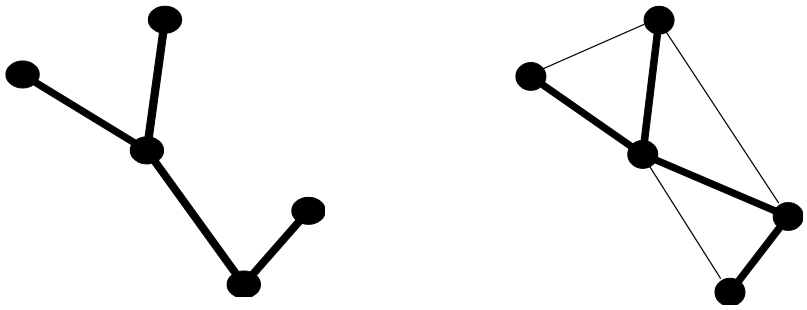
V teórii grafov sa ako *strom* (obrázok 2.6 vľavo) označuje súvislý neorientovaný graf G, ktorý neobsahuje žiadnu kružnicu a platí:

- $|V| = |H| + 1$,

kde V je množina vrcholov a H množina hrán grafu G. Pri niektorých úlohách z praxe je dôležité k danému grafu G nájsť jeho podgraf, ktorý obsahuje všetky vrcholy, je súvislý a nemá kružnice (tzn. je stromom). V tomto prípade hovoríme o *kostre grafu* G (obrázok 2.6 vpravo). Kostra súvislého grafu $G = \{V, H\}$ je ľubovoľný strom:

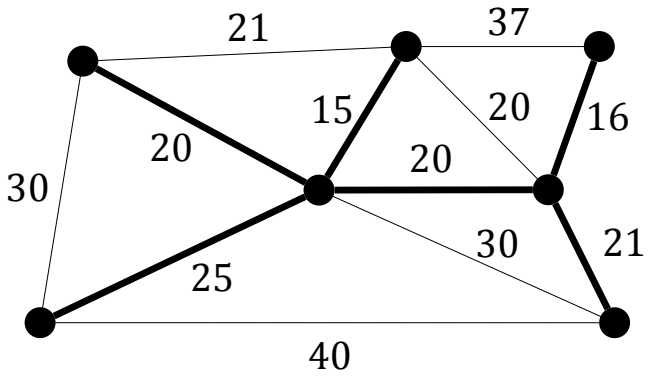
- $T = \{V, H'\}$,

kde $H' \leq H$. Platí, že každý súvislý graf má kostru. Pri hľadaní kostry v hranovo ohodnotenom grafe sa dostávame k *úlohe o minimálnej kostre*.



Obrázok 2.6: Strom a kostra grafu

V danom grafe G, kde každá hrana $h \in H(G)$ má reálne ohodnotenie (cenu) $c(h)$, treba nájsť minimálnu kostru, tj. takú kostru T grafu G, že jej cena $c(T)$ (súčet cien jednotlivých hrán kostry) je minimálna. Na obrázku 2.7 je príklad ohodnoteného grafu a jeho minimálnej kostry.



Obrázok 2.7

S úlohou nájsť minimálnu kostru sa možno stretnúť aj v praxi, napr. niektoré heuristiky riešiace TSP pracujú na základe nájdenia minimálnej kostry, pri zostavovaní najlacnejšej súvislej siete prenosných liniek pre sieť televíznych vysielačov, pri rozvode elektriny či plynu. Tu všade je nutné nájsť najlacnejší súvislý faktor kompletného grafu, kde cena hrany $\{u, v\}$ vyjadruje predpokladané náklady na vybudovanie linky z u do v. Hľadáme teda minimálnu kostru kompletného grafu.

Daný problém je dobre známy a zaoberalo sa ním mnoho matematikov (O. Borůka, V. Jarník, J. B. Kruskal). Jeho jednotlivé metódy riešenia sú dopodrobna popísané v [2].

3 Úvod do problematiky

Táto kapitola sa zameriava na popis a priblíženie zložitosti problému obchodného cestujúceho, ktorý patrí medzi najznámejšie NP-úplné problémy v súčasnosti, ďalej vysvetľuje pojem výpočtová zložitosť v teoretickej informatike triedy NP a podáva stručný opis problému N verzus NP [4]. Posledná časť kapitoly je venovaná formulácii TSP [2] na konkrétnom príklade. Pre podrobnejší prehľad v danej problematike odporúčam spomínanú literatúru.

3.1 Výpočtová zložitosť

Dôvod, prečo na problém obchodného cestujúceho ešte nebol nájdený algoritmus pracujúci v polynomiálnom čase, je jeho *časová zložitosť* (koľko krokov je potrebných na vyriešenie daného problému). TSP patrí do triedy NP-úplných problémov čo znamená, že so vzrastajúcim počtom miest exponenciálne rastie čas riešenia. Pre kompletný graf K_n, kde $n \geq 3$, počet všetkých rôznych hamiltonovských kružníc určuje vzťah: $\frac{1}{2}(n-1)!$. Zložitosť algoritmu, ktorý by počítal hodnotu všetkých hamiltonovských kružníc v kompletnom grafe, je $O(n!)^2$. Napríklad, ak hovoríme o kompletnom grafe K_4, tak pri návšteve týchto štyroch miest máme k dispozícii tri potenciálne cesty. Pri návšteve 10 miest rastie počet možných riešení na 181 440. Návšteva 15 miest znamená nárast potenciálnych ciest na viac ako 43 miliárd a pri 16 mestách sa toto číslo šplhá na viac ako 653 miliárd. Je zrejmé, že takýto algoritmus nepracuje v polynomiálnom čase.

Súčasťou modernej teórie grafov je stanovenie zložitosti riešenej úlohy a výpočtovej zložitosti daného algoritmu. Výpočtovou zložitosťou sa zaoberá *teória zložitosti*, ktorá rozdelila existujúce problémy do dvoch skupín:

- **„rozhodnuteľné“ problémy** (lineárna, logaritmická a polynomiálna zložitosť)
- **„nerozhodnuteľné“ problémy** (zložitosť väčšia než je polynomiálna, napr. exponenciálna)

Výpočtová zložitosť má dve základné miery, a to *časovú* (koľko krokov je potrebných na vyriešenie problému) a *pamäťovú* (koľko pamäti je potrebnej na vyriešenie problému) *zložitosť*. Minimalizáciou jednej alebo obidvoch mier dosiahneme optimalizáciu algoritmu.

Pre „rozhodnuteľné“ problémy poznáme algoritmy, ktorých časová náročnosť sa dá z hora ohraničiť polynómom. Bohužiaľ, v mnohých a veľmi praktických problémoch (napr. TSP) nepoznáme algoritmus, ktorý by pracoval v polynomiálne obmedzenom čase.

Všetky problémy sú rozdelené podľa svojej zložitosti do jednotlivých tried. Bližšie si popíšeme najznámejšie *triedy zložitosti P (polynomial)* a *NP (non-deterministic polynomial)*.

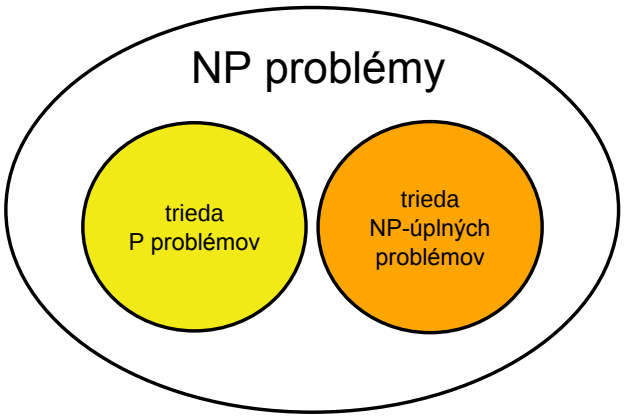
Ak hovoríme o *triede zložitosti P* ide o skupinu úloh, pre ktoré existuje algoritmus pre *deterministický Turingov stroj* [4] riešiaci tento problém v polynomiálnom čase. Deterministický Turingov stroj pre praktické účely takmer zodpovedá modelu súčasných počítačov.

V prípade, že je daná úloha týmto spôsobom neriešiteľná, ide o *triedu zložitosti NP* (non-deterministic polynomial). Do triedy NP patria úlohy riešiteľné v polynomiálnom čase na *nedeterministickom Turingovom stroji* [4], ktorý na n krokov môže preveriť exponenciálny počet

možností. Ide teda o problémy, ktorých riešenie môžeme v polynomiálnom čase overiť, ale nevieme či ho môžeme v polynomiálnom čase taktiež nájsť.

Medzi týmito dvoma typmi stojí *trieda NP-úplných problémov* (do tejto skupiny patrí aj TSP), na ktoré je možné v polynomiálnom čase redukovať ľubovoľný NP problém. Všetky problémy z tejto triedy sú rovnocenné v tom zmysle, že nájdenie „rýchleho" algoritmu pre riešenie ktoréhokoľvek z týchto úloh zaručuje existenciu efektívneho algoritmu pre vyriešenie ostatných problémov. Nezodpovedanou otázkou ostáva, či platí rovnosť $P = NP$. Podrobnejšie sa týmto problémom zaoberáme v nasledujúcej podkapitole.

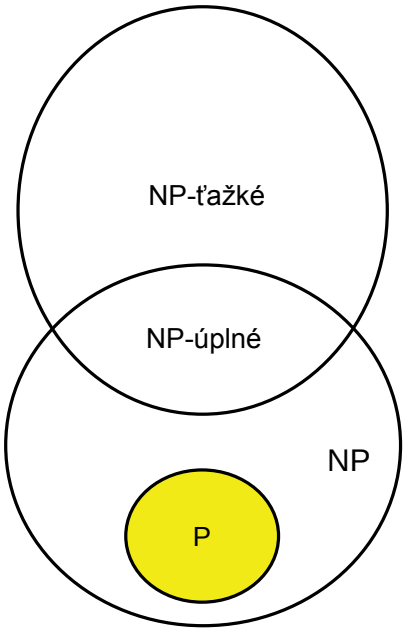
Vzťahy medzi popísanými triedami zložitosti sú graficky znázornené na obrázku 3.1.



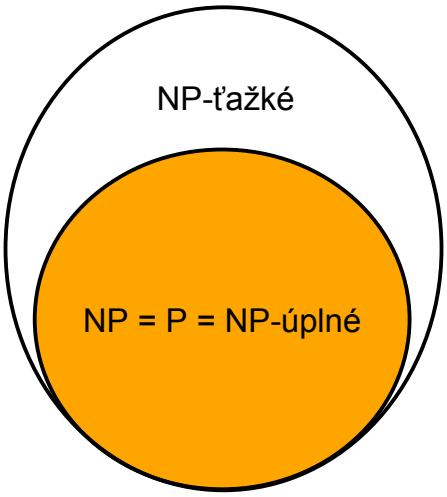
Obrázok 3.1

3.2 Problém „P vs. NP"

Ako *problém P verzus NP* sa v teoretickej informatike označuje otázka či platí rovnosť $P = NP$. Považuje sa za najdôležitejší otvorený problém tohto oboru a je zaradený medzi sedem tzv. problémov tisícročia. Je zrejmé, že $P \subseteq NP$, doteraz sa však s istotou nevedelo či aj všetky problémy z NP nepatria do P. Vzťahy medzi triedami zložitosti pre obidva prípady sú znázornené na obr. 3.2 a 3.3.



Obrázok 3.3: $P \neq NP$



Obrázok 3.2: $P = NP$

Ak by platilo $P = NP$, malo by to ďalekosiahle následky pre všetky NP-úplné problémy. Znamenalo by to, že existujú deterministické polynomiálne („rýchle“) algoritmy na ich riešenie. To by malo zásadný dopad na teoretickú informatiku, ale hlavne na kryptografiu. Tá totiž vo väčšine metód predpokladá, že sa šifry nedajú prelomiť dostatočne rýchlo. Teda zložitosť prelomenia rady moderných šifier závisí na predpoklade, že platí $N \neq NP$.

Podľa nedávno zverejneného dôkazu V. Deolalikara [5] platí $P \neq NP$. To automaticky znamená, že žiadny NP-úplný problém sa nemôže nachádzať v triede P, a teda byť riešiteľný v polynomiálnom čase na súčasných počítačoch. Medzi NP-úplnými problémami je rada praktických problémov. Ak sa potvrdí dôkaz V. Deolalikara, dané problémy, a teda aj problém obchodného cestujúceho, nie je možné riešiť v polynomiálnom čase. Pretože podľa dôkazu V. Deolalikara neexistuje polynomiálny algoritmus na riešenie problémov z NP a riešenie týchto úloh metódou hrubej sily (úplného prehľadávania všetkých možností) je v praxi už pre relatívne malý rozmer úlohy nezvládnuteľné, musíme sa uspokojiť s heuristikami, ktoré nám v rozumnom čase nájdu prijateľné riešenie.

3.3 Formulácia problému obchodného cestujúceho

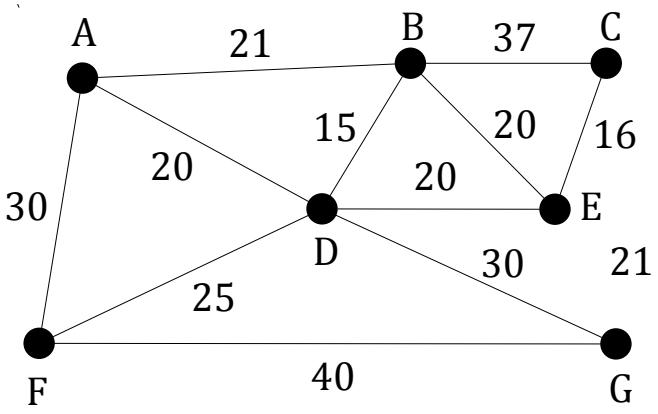
V predchádzajúcich kapitolách sme si priblížili všetky pojmy, ktoré sú potrebné na pochopenie problému obchodného cestujúceho a jeho následné riešenie. Preto sa teraz zameriame na jeho presnú formuláciu uvedenú na príklade. Na začiatku problému obchodného cestujúceho máme súbor vrcholov $\{v_1, v_{2,} \cdots, v_N\}$, ktoré predstavujú jednotlivé *mestá* a pre každý pár $\{v_i, v_j\}$ rôznych miest *vzdialenosť* $d(v_i, v_j)$. Našou úlohou je nájsť také usporiadanie π miest, ktoré bude mať najmenšiu veľkosť:

$$\sum_{i=1}^{N-1} d\left(v_{\pi(i),} v_{\pi(i+1)}\right) + d\left(v_{\pi(N),} v_{\pi(1)}\right)$$

Táto veľkosť je uvádzaná ako *dĺžka okružnej cesty*. To je cesta, ktorú obchodník prejde pri svojom cestovaní mestami, pričom sa na konci cesty vracia do počiatočného mesta. Zameriame sa na riešenie tzv. *symetrického* TSP, v ktorom dĺžky splňujú $d(v_i, v_j) = d(v_j, v_i)$ pre $1 \leq i, j \leq N$.

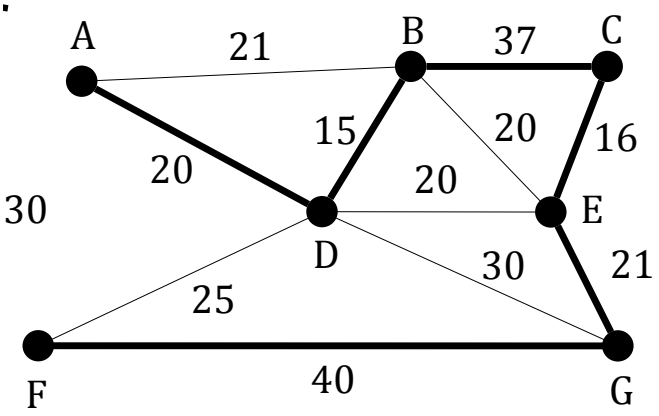
V terminológii teórie grafov jednotlivé mestá predstavujú vrcholy a prepojenia (cestná sieť) medzi mestami sú hrany. V kompletnom hranovo ohodnotenom grafe K_n hľadáme taký hamiltonovský cyklus, ktorý začína a končí vo vrchole v_0, a ktorého dĺžka c je minimálna.

Uvediem jeden príklad. Predstavme si situáciu, ktorú ilustruje obrázok 3.4.



Obrázok 3.4

Ide o neorientovaný graf, ktorý reprezentuje mapu oblasti, ktorou musí obchodný cestujúci prejsť. Vrcholy grafu predstavujú mestá, ktoré musí obchodný cestujúci navštíviť a hrany, spájajúce jednotlivé vrcholy, možné cesty. Ceny hrán zodpovedajú dĺžke jednotlivých úsekov ciest. Za počiatočný bod cesty si zvolíme mesto A. Úlohou je určiť poradie miest tak, aby dĺžka cesty bola čo najmenšia. Riešenie úlohy je zobrazené na obrázku 3.5.



Obrázok 3.5

4 Grafy v teoretickej informatike

Základným prostredím pri riešení TSP sú grafy. Vo všeobecnom grafe je dôkaz existencie hamiltonovského cyklu NP-ťažký problém. Z tohto dôvodu si všetky grafy prevedieme do kompletného grafu s kladne ohodnotenými hranami. Popis úpravy grafov je obsahom tejto kapitoly. Zdrojom informácií boli [1], [2].

4.1 Úprava vstupných grafov

Pri experimentálnom testovaní heuristík pracujeme s kompletnými grafmi, ktoré majú kladné ohodnotenie hrán. Je to z toho dôvodu, že u kompletného grafu máme zaručenú existenciu riešenia, zatiaľ čo u všeobecného grafu je hľadanie hamiltonovského cyklu NP-ťažký problém.

Akýkoľvek nekompletný súvislý graf $G = \{V, H\}$ môžeme pre našu potrebu upraviť na kompletný, pridaním chýbajúcich hrán do pôvodného grafu. Každá z týchto novo pridaných hrán bude mať vysoké kladné ohodnotenie M (napr. $M \geq \sum_{h \in H} c_h$). Takto upravený graf je možné riešiť heuristikami TSP pre kompletné grafy. Ak nájdený hamiltonovský cyklus neobsahuje žiadnu hranu veľkosti M, tak potom je tento cyklus optimálnym riešením aj pôvodného grafu. V opačnom prípade riešenie v pôvodnom grafe neexistuje.

Nevýhodou použitia heuristík je skutočnosť, že nám nezaručujú nájdenie optimálneho riešenia, napriek jeho existencii v pôvodnom grafe.

Ďalšou podmienkou je kladné ohodnotenie hrán. To zaručíme jednoduchým pripočítaním dostatočne veľkej konštanty K k všetkým hodnotám hrán v prípade, že sa v pôvodnom grafe nachádza ohodnotenie $c_h < 0$. Táto úprava nijakým spôsobom neovplyvní riešenie úlohy. Hamiltonovský cyklus obsahuje n hrán, pričom n je počet vrcholov daného grafu. V tomto prípade je ku každej z týchto n hrán pripočítaná aj konštanta K. Z uvedeného vyplýva, že odpočítaním hodnoty $n * K$ od výsledného cyklu získame hodnotu hamiltonovského cyklu pôvodného grafu.

4.2 Reprezentácia grafu pomocou matice

Vstupné grafy musíme pre naše potreby upraviť na formát matíc. Vychádzali sme z *matice susednosti*. Je to štvorcová matica $A_{(G)}$ grafu G typu $\{0,1\}^{n \times n}$, pre ktorú platí:

- $A_{i,j} = \begin{cases} 0 & v_i v_j \notin H_G \\ 1 & v_i v_j \in H_G \end{cases}$

Pretože pracujeme s hranovo ohodnotenými grafmi museli sme túto maticu upraviť. Maticu sme modifikovali na *maticu vzdialenosti* a to tým spôsobom, že namiesto 1 je na príslušnej pozícii číslo reprezentujúce ohodnotenie c_h danej hrany.

5 Heuristiky

Pri riešení úlohy obchodného cestujúceho sa stretávame s niekoľkými problémami. Ako sme už spomínali, ide o úlohu, ktorá patrí medzi NP-úplné problémy a riešenie metódou hrubej sily je v praxi už pre relatívne malý počet miest nezvládnuteľné. Z tohto dôvodu sa vedci skôr zamerali na nájdenie aproximačných algoritmov pracujúcich v polynomiálnom čase. Sú to algoritmy, ktoré nájdu prijateľné riešenie a toto riešenie sa blíži k optimálnemu. Ide o tzv. heuristiky, ktorých výsledky síce majú istú odchýlku od optimálnosti, no riešenie nájdu v prijateľnom čase. V niektorých prípadoch sa dokonca podarilo dokázať, že nájdené riešenie bolo aj riešením optimálnym.

Táto kapitola je venovaná popisu jednotlivých heuristík, pričom postupujeme systematicky od tých najjednoduchších (heuristika najbližšieho suseda) až po tie zložitejšie (neurónové siete). Začíname konštrukčnými heuristikami, ktoré sa pokúšajú o nájdenie prijateľného riešenia v jedinom pokuse. Do tejto kategórie patria aj heuristiky založené na hľadaní minimálnej kostry grafu. Po týchto základných heuristikách sa dostávame k heuristikám iteratívnym, ktoré sú založené na opakovanom zlepšovaní už nájdených riešení. Prehľad ukončujú stochastické heuristiky (metaheuristiky).

Pri každej heuristike je uvedené na ako princípe pracuje, jej bližší popis pomocou pseudokódu a názorný príklad na obrázku. Názvy heuristík sú uvedené v slovenskom jazyku a keďže väčšina odbornej literatúry popisujúca TSP je v anglickom jazyku, uvádzame aj ich anglické názvy.

Všetky uvedené heuristiky pracujú s grafmi. Z dôvodov uvedených v predchádzajúcej kapitole sú to kompletné kladne ohodnotené grafy $G(V, H, c)$. Základné informácie o jednotlivých heuristikách sú z [6], [7].

5.1 Konštrukčné heuristiky

Konštrukčná heuristika (constructive heuristic) je algoritmus, ktorý určuje cestu podľa daných konštrukčných pravidiel, ale nesnaží sa o vylepšenie tejto cesty. Vznikajúci hamiltonovský cyklus teda optimalizuje na základe určitého lokálneho minima. Výsledky najlepších konštrukčných heuristík sa obvykle pohybujú medzi 10 – 15% od optimálnosti.

Heuristiky, ktoré si do detailov popíšeme sú:

- Heuristika najbližšieho suseda
- Heuristika vkladania
- Metóda zdvojenia kostry
- Christofidesova metóda kostry a párenia.

Prvé tri heuristiky poskytujú rozumný mechanizmus na nájdenie počiatočnej cesty v procedúre, ktorá pracuje na lokálnom vyhľadávaní. Posledná reprezentuje to najlepšie čo môžu konštrukčné heuristiky dokázať a je preto hodnotným orientačným bodom.

5.1.1 Heuristika najbližšieho suseda

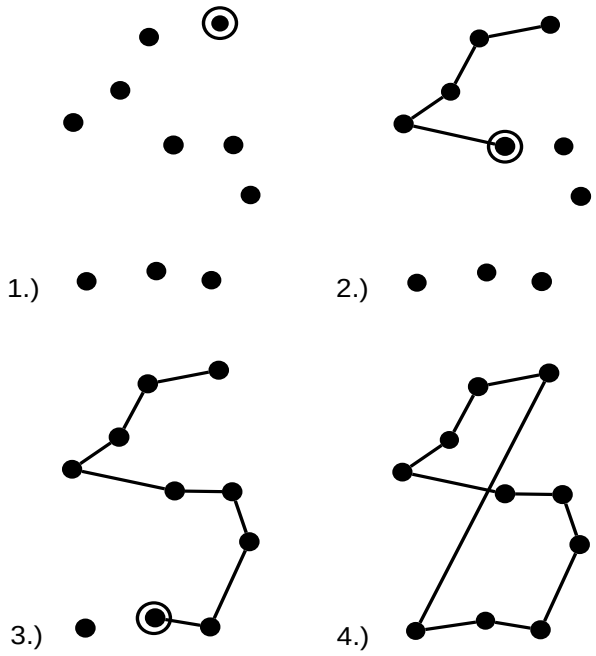
Najprirodzenejšou heuristikou pre TSP je *heuristika najbližšieho suseda* (*NN* z angl. *Nearest Neighbor*). Tento algoritmus napodobuje cestovateľa, ktorého pravidlom je navštíviť vždy najbližšie nenavštívené mesto. Na základe toho zostavíme poradie miest $v_{\pi(1)}, \cdots, v_{\pi(N)}$ s počiatočným mestom $v_{\pi(1)}$, ktoré si ľubovoľne zvolíme. Vo všeobecnosti pre mesto $v_{\pi(i+1)}$ vybrané v poradí ako mesto v_k platí $\{d(v_{\pi(i)}, v_k) : k \neq \pi_{(j)}, 1 \leq j \leq i\}$. Keď cestujúci prejde všetkými mestami, vracia sa po

návšteve mesta $v_{\pi(N)}$ do počiatočného mesta $v_{\pi(1)}$. Pretože v tejto chvíli boli navštívené všetky mestá práve raz, bolo nájdené prípustné riešenie tejto úlohy.

NAJBLIŽŠÍ_SUSED

1. Vyberte ľubovoľný počiatočný vrchol $v_j \in V$, položte $k = j$ a $W = \{1, \cdots, N\} - \{j\}$.
2. Pokiaľ $W \neq \emptyset$ opakujte:
   a. Vyberte vrchol v_j, $j \in W$ tak, že platí $cena_{kj} = min\{cena_{ki} \mid i \in W\}$.
   b. Spojte vrcholy v_k a v_j, položte $W = W - \{j\}$ a $k = j$.
3. Na uzavretie hamiltonovského cyklu spojte v_k s počiatočným vrcholom v_j, ktorý ste vybrali v prvom kroku.

Spomínaný postup je znázornený na obrázku 5.1. Zvýraznený vrchol bol práve vybraný algoritmom ako najbližší sused predchádzajúceho vrcholu (okrem počiatočného bodu, ktorý je vybraný náhodne).



Obrázok 5.1: Princíp činnosti heuristiky najbližšieho suseda

Táto štandardná verzia algoritmu beží v čase $O(n^2)$ a výsledok sa mnohokrát drží 24% od *Held-Karpovej dolnej hranice* (*Held-Karp lower bound*) [7]. Heuristika najbližšieho suseda poskytuje v celku uspokojivé výsledky pre úlohy menších rozmerov, no v porovnaní s ostatnými metódami ďaleko zaostáva pri riešení väčších úloh. Aj napriek tomuto nedostatku sa veľmi často využíva vďaka svojej jednoduchosti. Na začiatku pracuje veľmi efektívne, keď do cyklu zapája naozaj tie najvhodnejšie vrcholy. V priebehu zostavovania cyklu však niektoré vrcholy zostanú „zabudnuté“ a sú pripojené až neskôr na pozíciu oveľa horšiu než je optimálna. Vo väčšine prípadov je veľmi „drahé“ aj spojenie posledného vrcholu s prvým, za účelom získania hamiltonovského cyklu. To je dobre viditeľné aj na ukážkovom príklade. Vytvorený cyklus však obsahuje aj dostatok vhodne zvolených úsekov, preto sa často používa na vytvorenie počiatočného cyklu pre iteratívne (zlepšovacie) heuristiky.

Ďalší možný variant tohto algoritmu je tzv. *dvojstranná heuristika najbližšieho suseda*. V tomto prípade pripájame vrcholy na výhodnejší koniec cesty.

Hlavným problémom algoritmu je drahé zapájanie vrcholov na konci cyklu. Túto nevýhodu môžeme minimalizovať spojením niekoľkých vrcholov naraz. Na začiatku spojíme 10 najbližších vrcholov [6] do jedného podgrafu H a ten definujeme ako jeden bod k. V nasledujúcej iterácii pokračujeme v pôvodnom grafe zmenšenom o body podgrafu H a novo pridaným bodom h. Takto pokračujeme až kým v pôvodnom grafe nezostane menej než 20 vrcholov, ktoré následne zapojíme štandardným spôsobom. V porovnaní s priemernou kvalitou 24% štandardnej verzie má modifikovaný algoritmus priemernú kvalitu 18,6%.

Treťou možnosťou ako vylepšiť heuristiku najbližšieho suseda je využiť citlivosť algoritmu na výber počiatočného vrcholu cesty tým spôsobom, že spustíme algoritmus pre každý z možných počiatočných bodov, a potom vyberieme ten najlepší. V tomto prípade môže byť nájdené riešenie až o 10% lepšie, avšak časová náročnosť algoritmu vzrastie na $O(n^3)$.

5.1.2 Heuristika vkladania

Ďalšia z radu intuitívnych prístupov k danému problému je *heuristika vkladania* (*Insertion heuristic*). Na rozdiel od predchádzajúcej heuristiky začína s malou podskupinou vrcholov, ktoré tvoria hamiltonovský cyklus. Priberaním zostávajúcich vrcholov sa počiatočný cyklus zväčšuje, až kým nezapojí všetky vrcholy grafu.

HEURISTIKA_VKLADANIA

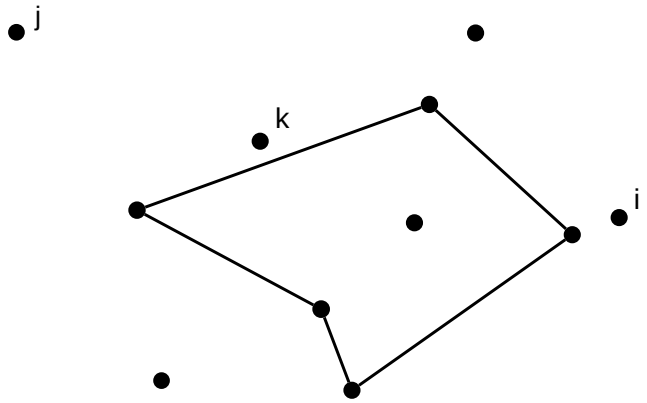
1. Vyberte počiatočný cyklus k vrcholov $v_1, v_2, \cdots, v_k$ $(k \geq 1)$ a $W = V - \{v_1, v_2, \cdots, v_k\}$.
2. Pokiaľ $W \neq \emptyset$ opakujte:
   a. Vyberte vrchol v_j, $j \in W$ podľa niektorého kritéria.
   b. Vložte j na niektorú pozíciu v cykle a položte $W = W - \{j\}$.

Pri implementácii môžeme vychádzať z niekoľkých možností. Hlavným rozdielom medzi nimi je kritérium na výber vrcholu j v kroku $(2.a)$. Počiatočný cyklus môže pozostávať z troch ľubovoľných vrcholov grafu alebo v degenerovaných prípadoch zo slučky $(k = 1)$, či hrany $(k = 2)$. Vrchol, vybraný v kroku $(2.a)$, sa obyčajne vkladá na miesto, kde spôsobí najmenší nárast dĺžky cyklu.

Teraz si bližšie popíšeme niektoré varianty rozšírenia aktuálneho cyklu. Hovoríme, že vrchol je *vrcholom cyklu*, ak je obsiahnutý v čiastočnom hamiltonovskom cykle. Pre $j \in W$ definujeme $d_{min}(j) = min\{cena_{ij} \mid i \in V - W\}$.

- **NAJBLIŽŠIE VKLADANIE:** Vloží vrchol, ktorý má najkratšiu vzdialenosť od vrcholov cyklu, t.j. vyberie $j \in W$ kde $d_{min}(j) = min\{d_{min}(l) \mid l \in W\}$.
- **NAJVZDIALENEJŠIE VKLADANIE:** Vloží vrchol, ktorého najkratšia vzdialenosť od vrcholov cyklu je najdlhšia, t.j. vyberie $j \in W$ kde $d_{min}(j) = max\{d_{min}(l) \mid l \in W\}$.
- **NAJLACNEJŠIE VKLADANIE:** Vloží vrchol, ktorý zväčší dĺžku cyklu čo najmenej.
- **NÁHODNÉ VKLADANIE:** Vrchol vyberie náhodne a vloží ho na najlepšiu možnú pozíciu

Princíp výberových kritérií si vysvetlíme na nasledujúcom obrázku 5.2.



Obrázok 5.2

V situácii, ktorú ilustruje obrázok 5.2 si najbližšie vkladanie vyberie vrchol i, najvzdialenejšie vrchol j a najlacnejšie by vybralo vrchol k.

Všetky vyššie popísané heuristiky, okrem najlacnejšieho vkladania, bežia v čase $O(n^2)$. Nie sú vhodné pre úlohy väčších rozmerov z dôvodu veľkej pamäťovej náročnosti. Najlacnejšie vkladanie má časovú náročnosť $O(n^2 \log n)$. Pre úlohy TSP splňujúce trojuholníkovú nerovnosť platí, že dĺžka hamiltonovských cyklov vypočítaných heuristikou najbližšieho a najlacnejšieho vkladanie je menšia než dvojnásobok dĺžky optimálneho cyklu (Rosenkratz, Stearns a Lewis (1977)).

Výsledky majú priemernú kvalitu 20,0%, 9,9% a 11,1% pre najbližšie, najvzdialenejšie a náhodné vkladanie v tomto poradí. Výkon heuristiky vkladania nezávisí tak veľmi na počiatočnom zoskupení ako je to u heuristiky najbližšieho suseda. Môžeme očakávať odchýlku cca 6% pre variantu náhodného vkladania a 7 – 8% pre ostatné varianty.

5.2 Heuristiky založené na minimálnej kostre

Heuristiky v tejto skupine generujú hamiltonovský cyklus z minimálnej kostry grafu. Obidve nižšie popísané heuristiky boli pôvodne navrhnuté pre grafy splňujúce trojuholníkovú nerovnosť, no v princípe ich je možné použiť aj na všeobecné grafy.

Skôr než budú uvedené heuristiky riešiace TSP je nutné si priblížiť algoritmus, ktorý z ľubovoľného sledu vytvorí hamiltonovský cyklus. Tento hamiltonovský cyklus nie je dlhší než táto cesta. Majme sled vrcholov $v_{i0}, v_{i1}, \cdots, v_{ik}$ (vrátane opakovania), cesta začína vo vrchole v_{i0} a končí v tom istom vrchole $v_{ik} = v_{i0}$.

VYTVORENIE_CYKLU

1. Položte $T = \{v_{i_0}\}$, $v = v_{i_0}$ a $l = 1$.
2. Pokiaľ $|T| < n$ opakujte:
   a. Ak $v_{i_l} \notin T$ potom $T = T \cup \{v_{i_l}\}$, spojte v s v_{i_l} a položte $v = v_{i_l}$.
   b. Položte $l = l + 1$.
3. Na vytvorenie hamiltonovského cyklu spojte v s v_{i0}.

Ak v grafe platí trojuholníková nerovnosť, potom každé algoritmom vytvorené spojenie vrcholov je hranou cyklu alebo jej skratkou, ktorá spája dva koncové vrcholy. Z toho vyplýva, že výsledný hamiltonovský cyklus nemôže byť dlhší než pôvodný sled.

Obidve heuristiky sú si veľmi podobné a začínajú s minimálnou kostrou grafu. Líšia sa len spôsobom vytvárania výsledného cyklu.

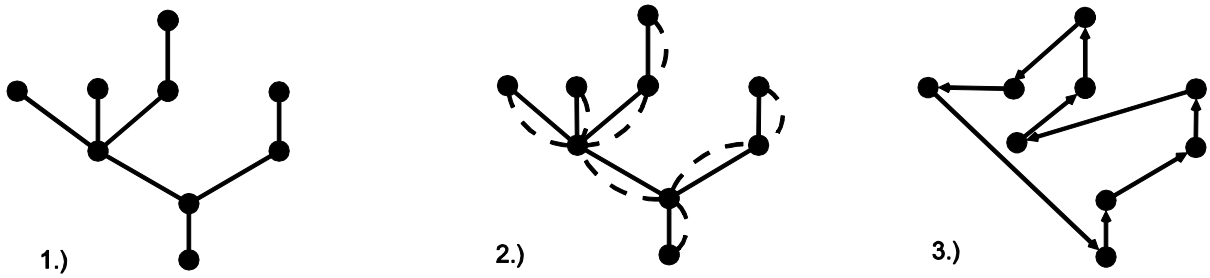
5.2.1 Metóda zdvojenia kostry

Metóda zdvojenia kostry (*Doubletree heuristic*) je založená na zdvojnásobení hrán minimálnej kostry grafu. V prvom kroku je potrebné nájsť minimálnu kostru daného grafu, na jej výpočet použijeme Kruskalov algoritmus [2].

ZDVOJENIE_KOSTRY

1. Vypočítajte minimálnu kostru grafu.
2. Zdvojnásobte všetky hrany kostry a vytvorte eulerovský graf.
3. Nájdite eulerovský ťah.
4. Použite VYTVORENIE_CYKLU na získanie konečného hamiltonovského cyklu.

Časová náročnosť algoritmu je $O(n^2)$ a je podmienená časovou náročnosťou pre výpočet minimálnej kostry grafu.



Obrázok 5.3: Princíp činnosti heuristiky zdvojenia kostry

Obrázok 5.3 nám znázorňuje činnosť algoritmu. V prvom kroku je nutné nájsť minimálnu kostru grafu – obrázok 1). Nasleduje zdvojenie hrán kostry, ktoré ilustruje obrázok 2). Jeden z možných výsledkov, ak procedúru VYTVORENIE_CYKLU iniciujeme z najspodnejšieho vrcholu, vidíme na obrázku 3). Táto metóda má najhorší možný výsledok:

$$\frac{c(MZK)}{c(OPT)} < 2$$

Kde $c(MZK)$ je dĺžka hamiltonovského cyklu získaného metódou zdvojenia kostry a $c(OPT)$ je dĺžka najkratšieho hamiltonovského cyklu v grafe. Dôkazom je získanie najlacnejšej kostry po vypustení najdlhšej hrany z hamiltonovského cyklu. Priemerná kvalita výsledku sa pohybuje okolo 38,08%.

5.2.2 Christofidesova metóda kostry a párenia

Nasledujúca heuristika – *Christofidesova metóda kostry a párenia* (*Christofides*), nesie názov podľa svojho autora, ktorým je Christofides (1976). Christofides, v porovnaní s predchádzajúcou heuristikou, má omnoho sofistikovanejší prístup k problematike, napriek tomu, že obidve heuristiky vychádzajú z najlacnejšej kostry. Dosiahol zlepšenie aproximácie tým, že namiesto zdvojenia

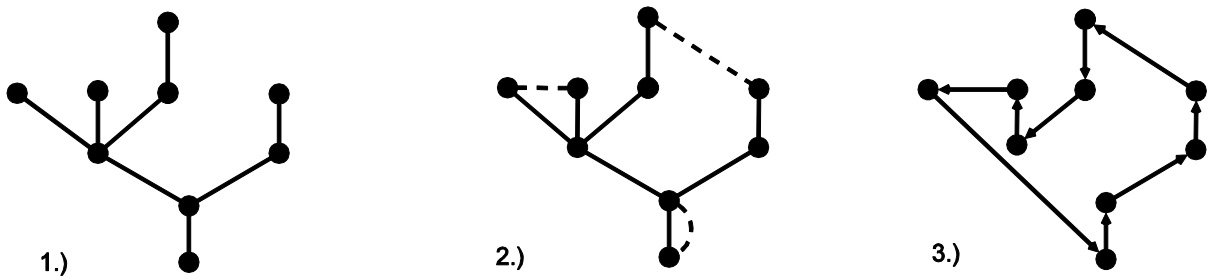
všetkých hrán kostry, pridá len tie hrany, ktorú sú nevyhnutne potrebné k zaručeniu eulerovského ťahu. Pridané hrany sú ďalej podrobené minimalizačným požiadavkám.

CHRISTOFIDES

1. Vypočítajte minimálnu kostru grafu.
2. Vypočítajte systém najlacnejších hrán pre páry vrcholov nepárneho stupňa a získané hrany pridajte ku kostre.
3. Nájdite eulerovský ťah.
4. Použite VYTVORENIE_CYKLU na získanie konečného hamiltonovského cyklu.

Táto metóda zaberie značne viac času než predchádzajúca. Jej časová náročnosť je $O(n^3)$.

Činnosť heuristiky ilustruje obrázok 5.4. Plné čiary zodpovedajú hranám minimálnej kostry grafu, čiarkovane sú zobrazené najlacnejšie hrany medzi nepárnymi vrcholmi grafu, ktoré boli doplnené v druhom kroku tejto metódy. Konečný výsledok znázorňuje obrázok 3).



Obrázok 5.4: Princíp činnosti Christofidesovej metódy

Pre úlohy spĺňajúce trojuholníkovú nerovnosť nedáva Christofidesova metóda horší výsledok než:

$$\frac{c(MKP)}{c(OPT)} < \frac{3}{2}$$

Kde $c(MKP)$ je dĺžka hamiltonovského cyklu získaného metódou kostry a párenia a $c(OPT)$ je dĺžka najkratšieho hamiltonovského cyklu v grafe. Nie je známy žiadny polynomiálny algoritmus, pre ktorý by bol zaručený lepší pomer $c(MKP)/c(OPT)$ než $3/2$.

5.3 Iteratívne heuristiky

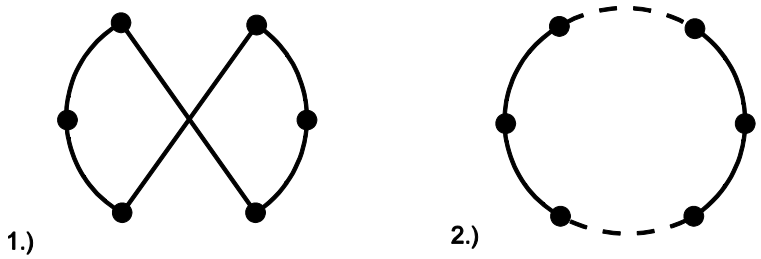
Doteraz sme sa zaoberali heuristikami, ktoré konštruovali hamiltonovský cyklus zo zadaného grafu. Takto získané riešenie však nijak ďalej neupravovali. Dosahované výsledky boli preto len miernej kvality, a hoci môžu byť použiteľné v niektorých aplikáciách, vo všeobecnosti nie sú moc uspokojivé.

V tejto časti sa budeme zaoberať otázkou ako tieto známe cykly vylepšiť. Ide o algoritmy, ktoré zlepšujú už známe riešenia, získané heuristikami v predchádzajúcej podkapitole. Rozširujúce informácie nájdete v [6], [7] a [8], z ktorých sme čerpali.

5.3.1 2-optimalizácia

Metóda *2-optimalizácie* (*2-opt*) je založená na lokálnom prehľadávaní. Ide o jednoduchú heuristiku, ktorá začína s náhodne vygenerovanou permutáciou miest (cestou) T a snaží sa ju ďalej vylepšovať.

Ak hamiltonovský cyklus v istej časti križuje sám seba, výmenou zapojenia je možné získať cyklus kratší. Za týmto účelom sa definuje okolie cesty T tak, že do tejto množiny patria všetky cesty, ktoré sa od T líšia vzájomnou výmenou dvoch priamo nenadväzujúcich hrán z cesty T. Táto operácia ja nazývaná 2-výmenou a je znázornená na obrázku 5.5.



Obrázok 5.5: Ukážka 2-zámeny

Pokiaľ sa v okolí cesty T nenachádza cesta T' s nižšími celkovými nákladmi, potom T' nahradí T a pokračuje sa ďalej rovnakým spôsobom. Pokiaľ sa v okolí cesty T už žiadna cesta s nižšími nákladmi nenachádza, cesta T je 2-optimálna (čo, ale nie je to isté ako cesta optimálna) a algoritmus končí.

Pre urýchlenie činnosti algoritmu sa obvykle negeneruje v jednotlivých iteráciách celé okolie cesty T, ale tento proces končí okamžite po nájdení prvej cesty T', ktorá má nižšie celkové náklady než cesta T.

2_OPTIMALIZÁCIA

1. Nech T je ľubovoľný hamiltonovský cyklus.
2. Pre každý vrchol i, až kým sa neohlási chyba, opakujte:
   a. Vyberte vrchol i.
   b. Preskúmajte všetky 2-optimalizačné kroky zahrňujúce hranu medzi vrcholom i a jeho nástupcom (pravý sused) v cykle. Zo všetkých možných riešení vyberte to najlepšie a zapojte ho. Ak neexistuje žiadne skrátenie deklarujte chybu pre i.
3. Ako výsledok vráťte nové T.

Hoci nájdenie najlepšieho riešenia (vylepšenia) v jednom kroku si žiada $O(n^2)$ operácií, počet všetkých operácií algoritmu nie je možné ohraničiť polynomiálnou funkciou. Je však zaručená jeho konečnosť, keďže existuje konečný počet prípustných cyklov a v každom kroku vyberáme lacnejší od aktuálneho.

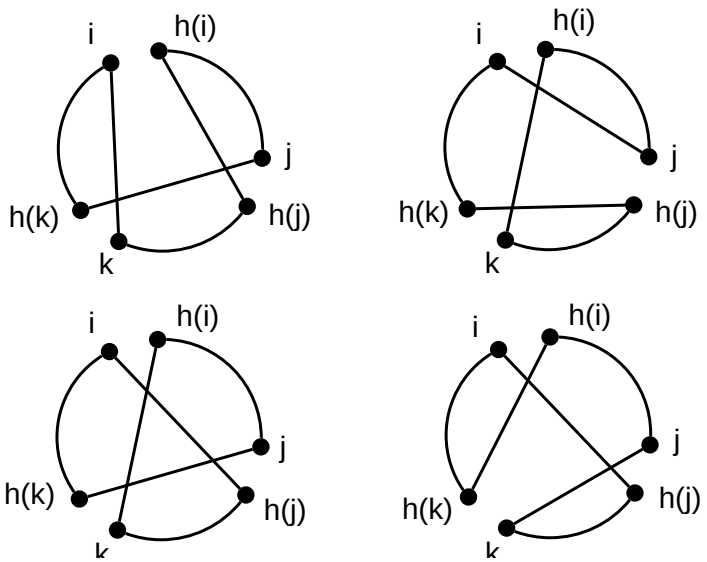
5.3.2 3-optimalizácia

Na základe predchádzajúceho princípu pracuje i heuristika *3-optimalizácia* (*3-opt*), ktorou získame flexibilnejších zmien. Podobne ako 2-optimalizácia pracuje so zmenami cyklu pomocou kombinovaného prepojenia voľných vrcholov, ktoré získame eliminovaním troch nesusedných hrán. Na rozdiel od 2-optimalizácie, kde bolo možné len jediné znovu zapojenie, v čistej 3-optimalizácii (bez zdegenerovaných 2-optimalizačných krokov) môžeme vrcholy zapojiť do cyklu štyrmi rôznymi spôsobmi (obrázok 5.6, v ktorom sú znázornené možnosti 3-optimalizačného kroku pre vrcholy

i, j, k a ich pravých susedov $h(i), h(j), h(k)$). Výsledná cesta s nižšími celkovými nákladmi ako bola pôvodná je 3-optimálna.

3_OPTIMALIZÁCIA

1. Nech T je ľubovoľný hamiltonovský cyklus.
2. Pre každý vrchol i, až kým sa neohlási chyba, opakujte:
   a. Vyberte vrchol i.
   b. Preskúmajte všetky 3-optimalizačné kroky zahrňujúce hranu medzi vrcholom i a jeho nástupcom (pravý sused) v cykle. Zo všetkých možných riešení vyberte to najlepšie a zapojte ho. Ak neexistuje žiadne skrátenie deklarujte chybu pre i.
3. Ako výsledok vráťte nové T.



Obrázok 5.6

Heuristiky 2-optimalizácia a 3-optimalizácia je možné veľmi ľahko zovšeobecniť na *k*-opt. So zvyšujúcou sa hodnotou parametra *k* je príslušný algoritmus schopný aplikovať sofistikovanejšie modifikácie cesty a tým smerovať k dokonalejšiemu riešeniu úlohy. Na druhú stranu, s rastúcou hodnotou parametru *k* sa exponenciálne zväčšuje mohutnosť okolia pôvodnej cesty a zodpovedajúcim spôsobom rastie i čas potrebný k spracovaniu tejto množiny. Z tohto dôvodu sa v praxi len výnimočne používa *k*-opt algoritmus s hodnotou $k > 3$.

5.3.3 Lin-Kernighan

Na využití *k*-opt stratégie je založený i zrejme najúspešnejší heuristický algoritmus, ktorý je v literatúre známy ako *Lin-Kernighanov algoritmus* (*Lin-Kernighan algorithm*). Prvým zreteľným rozdielom oproti vyššie uvedeným heuristikám je skutočnosť, že Lin a Kernighan vylepšili výmenu hrán tak, že parameter *k* nie je pevne stanovený pre celý beh algoritmu , ale mení sa postupne iteráciu po iterácii. Navyše sa v tomto prípade algoritmus neuspokojí s prvým nájdeným vylepšením cesty T, ale hľadá sa také vylepšenie, ktoré prinesie najväčšie možné zníženie nákladov v danej iterácii. Viac informácii k tejto heuristike nájdete v [9], [10].

5.4 Stochastické heuristiky

Ďalšou veľkou oblasťou algoritmov na riešenie TSP sú algoritmy založené na náhodnom (stochastickom) prehľadávaní. Ich najväčšou výhodou je schopnosť opustiť lokálne minimum, a tým s určitou pravdepodobnosťou nájsť riešenie s lepšou hodnotou účelovej funkcie ako malo dané nájdené lokálne minimum. Nasledujúce informácie vychádzajú predovšetkým z [6], [7], [11].

Heuristiky založené na analógii s prírodnými procesmi tvoria hranicu medzi operačným výskumom a umelou inteligenciou, ktorá sa aplikuje na problémy optimalizácie. Medzi najznámejšie patria:

- Simulované žíhanie
- Zakázané hľadanie
- Kolónia mravcov
- Neurónové siete

5.4.1 Simulované žíhanie

Metóda *simulovaného žíhania* (*Simulated Annealing*) je založená na vzťahu medzi procesom hľadania optimálneho riešenia pre kombinatorickú optimalizáciu a fenoménmi vyskytujúcimi sa vo fyzike. Inšpiráciou metódy je fyzikálny dej prebiehajúci pri žíhaní pevného telesa, ktorý sa využíva k odstráneniu vnútorných defektov.

Jej hlavnou výhodou je úspešné riešenie uviaznutia v lokálnom minime, s ktorým si predchádzajúce metódy nevedeli poradiť. Nepripúšťali totiž možnosť zväčšenia dĺžky cyklu, a to ani v prípade, kedy by v nasledujúcich krokoch došlo k výraznému zlepšeniu. Metóda simulovaného žíhania však s istou mierou pravdepodobnosti prijíma i riešenie horšie, než bolo vychádzajúce.

Fyzikálna realizácia žíhania bola nahradená numerickou simuláciou pomocou Metropolisovho algoritmu [11]. Hlavný algoritmus simulovaného žíhania vyzerá nasledovne:

SIMULOVANÉ_ŽÍHANIE

1. Nech T je ľubovoľný hamiltonovský cyklus, vyberte počiatočnú teplotu ϑ a faktor opakovania r.
2. Kým nie sú naplnené podmienky ukončenia opakujte:
   2.1. r-krát opakujte:
      2.1.1. Urobte náhodnú zmenu cyklu T na cyklus T' a vypočítajte rozdiel ich celkových dĺžok $\Delta = c(T) - c(T')$.
      2.1.2. Ak $\Delta < 0$ potom $T = T'$. Inak vyberte náhodné číslo x, $0 \leq x \leq 1$ a $T = T'$, ak $x < e^{-\frac{\Delta}{\vartheta}}$.
   2.2. Aktualizujte ϑ a r.
3. Ako výsledok vráťte najlepšie dosiahnuté T.

Táto formulácia nám dáva istú mieru voľnosti pri implementácii. Preto uvádzame niektoré konkrétne body nastavenia: v kroku (1) je r inicializované ako počet vrcholov grafu, v kroku (2.1.1) sme modifikáciu cyklu T na cyklus T' realizovali pomocou 2-optimalizačných krokov, zníženie teploty ϑ v kroku (2.2) nastavíme položením $\vartheta = \gamma\vartheta$, kde γje reálne číslo blízko 1 a faktor opakovania r upravíme podobne $r = \alpha r$, kde α je ľubovoľné reálne číslo z intervalu (1,2). Po dosiahnutí určitého

počtu opakovaní faktor r už ďalej nezvyšujeme a procedúra končí, ak po viacerých úpravách teploty ϑ nedôjde k zmene cyklu T.

Počas celého behu procesu simulovaného žíhania sa môžeme dostať do rôznych lokálnych miním, ale rovnako ich môžeme aj opustiť. Preto je dobré pamätať si najlepšie riešenie dosiahnuté počas simulácie a ako výstup vrátiť práve toto riešenie.

5.4.2 Zakázané hľadanie

Metóda *zakázaného hľadania* (*Tabu Search*) je vylepšenou verziou horolezeckého algoritmu, informácie prevzaté z [11]. Jej vylepšenie spočíva v tom, že do horolezeckého algoritmu je zavedená tzv. krátkodobá pamäť, ktorá znižuje nebezpečenstvo návratu k už preskúmanému riešeniu tak, že každé riešenie je po istú dobu zaznamenané v zozname zakázaných riešení (tabu list). V princípe heuristika pracuje nasledovne:

TABU_SEARCH

1. Nech T je ľubovoľný hamiltonovský cyklus, začnite s prázdnym tabu listom $\mathcal{L}$.
2. Kým nie sú naplnené podmienky ukončenia, opakujte nasledujúce kroky:
   a. Vyberte najlepší krok, ktorý nie je zakázaný v $\mathcal{L}$.
   b. Aktualizujte tabu list $\mathcal{L}$.
3. Ako výsledok vráťte najlepšie nájdené T.

Tak ako v predchádzajúcom prípade máme niekoľko možností ako heuristiku implementovať. Na začiatku je potrebné nastaviť dĺžku tabu listu (tabu list $<$ počet transformácií v rámci susedstva), v ktorom je uvedených posledných n zmien ohodnotení, konkrétne dvojica (premenná, hodnota). Empirické skúsenosti s algoritmom ukazujú, že veľkosť tabu listu je dôležitým parametrom pre prekonanie lokálneho minima. Ak je dĺžka zoznamu malá, môže sa vyskytnúť zacyklenie algoritmu, rovnako ako u klasického horolezeckého algoritmu, ale po viacerých krokoch. Ak je však dĺžka tabu listu veľká, potom s veľkou pravdepodobnosťou dôjde k preskočeniu možných ciest vedúcich k optimálnemu riešeniu. Pri každom novom ohodnotení premennej nejakou hodnotou je táto dvojica zapísaná na začiatok tabu listu a posledná (najstaršia) dvojica je z neho odobraná. Vždy, keď sa vyberá hodnota pre nejakú premennú, zisťuje sa či sa táto dvojica nenachádza v tabu liste. Pokiaľ áno, je zakázané túto hodnotu premennej nastaviť. Táto technika teda zabraňuje opätovnému nastaveniu rovnakej hodnoty rovnakej premennej v krátkom časovom intervale, určenom dĺžkou tabu listu.

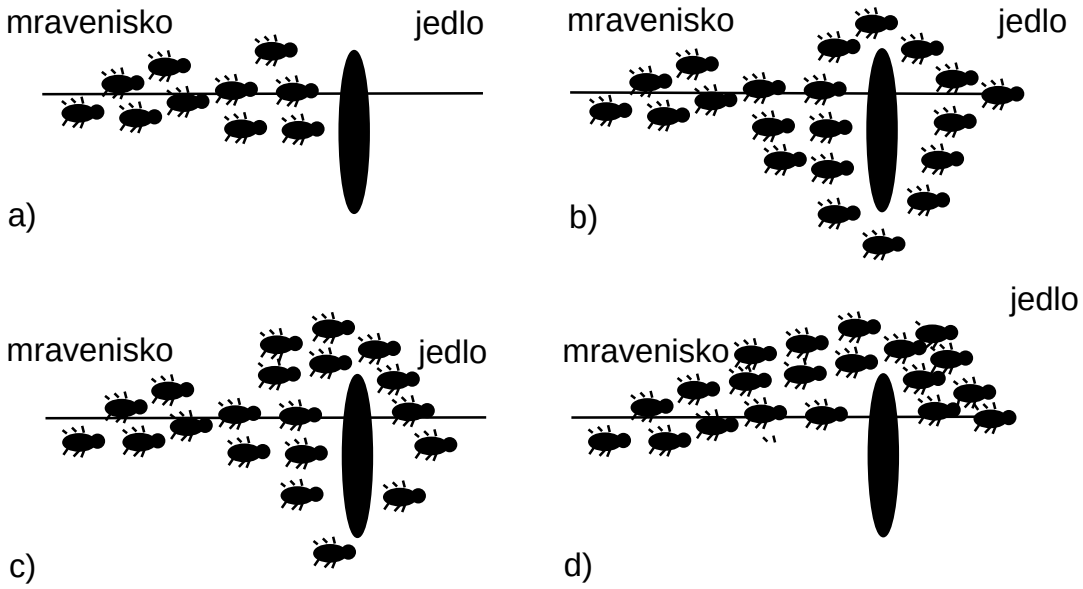
Často bývajú zavedené ešte dodatočné (aspiračné) kritériá, ktoré nám hovoria, kedy môže byť zákaz ohodnotenia danej premennej danou hodnotou porušený. Je to napríklad v tom prípade, keď nastavením tejto hodnoty získame doposiaľ najlepšie nájdené riešenie. Ako najlepšia hodnota dĺžky tabu listu býva často uvádzané číslo medzi 10 a 50, i keď táto hodnota veľmi závisí na štruktúre riešeného problému.

Iný prístup je založený na tzv. „dlhodobej pamäti“. Využíva metódy intenzifikácie a diverzifikácie algoritmu zakázaného hľadania k nájdeniu globálneho optima. Využíva možnosti pokutovania transformácií, ktoré nepatria do krátkodobej pamäti, ale často sa vyskytovali v doterajšom procese hľadania. Prípadne, do dlhodobej pamäti zaznamenávame frekvenciu výskytu určitých atribútov doposiaľ získaných riešení. Intenzifikačná stratégia podporuje transformácie vedúce k riešeniu s „dobrými“ vlastnosťami. Oproti tomu diverzifikačná stratégia znamená podporu riešení, obsahujúcich významne odlišné atribúty od tých, ktoré sa vyskytli v priebehu doterajšieho hľadania.

5.4.3 Kolónia mravcov

Ako väčšina stochastických optimalizačných algoritmov aj optimalizácia pomocou *kolónie mravcov* (*Ant Colony*) má svoj základ v biológii. Metóda je založená na simulácii správania sa mravcov v kolónii.

Princíp riešenia problémov pomocou kolónie mravcov prvýkrát použil Marco Dorigo a kol. [12] na riešenie NP-úplných problémov na grafoch. Cieľom mravcov je nájdenie potravy. Každý mravec pri svojej ceste za potravou vylučuje chemickú látku – feromón. Ostatné mravce sa riadia touto stopou a vyberajú si cestu s najväčšou koncentráciou feromónu. Po určitej dobe mravce nájdu optimálnu (najkratšiu) cestu od mraveniska k cieľu a túto trasu potom preferujú. Tento princíp znázorňuje nasledujúci obrázok 5.7.



Obrázok 5.7: Princíp hľadania kratšej cesty kolóniou mravcov

Na začiatku príde mravec k rázcestiu (obrázok a)), ktoré ešte nebolo označené feromónom a cestu si vyberie náhodne (obrázok b)). Vybranú cestu značkuje feromónom. Po nájdení potravy sa vracia do mraveniska cestou, ktorou prišiel a značkuje ju po druhý raz. Mravce, ktoré si zvolia kratšiu z ciest sú pri potrave skôr – cesta je častejšie označovaná feromónom a ostatné mravce ju budú preferovať (obrázok c). Po určitej dobe je intenzita feromónu na kratšej z ciest tak silná, že sa stane preferovanou voľbou aj pre ostatné mravce (obrázok d)) a zosilňovanie prebieha aj naďalej.

Na cestách sa však feromóny aj vyparujú. Ak niektorá križovatka nebola dlhšie navštívená, rozdiely v množstve feromónov sa znižujú, až kým si mravec môže opäť náhodne vybrať, ktorou cestou pôjde.

Pre účely implementácie optimalizačných algoritmov je vytváraný umelý mravec, ktorého správanie je inšpirované správaním skutočných mravcov. Oproti nim však boli niektoré vlastnosti umelých mravcov upravené tak, aby došlo k zlepšeniu výsledkov algoritmov riešiacich konkrétne problémy. Disponujú pracovnou pamäťou M_k, ktorá slúži na uloženie už navštívených miest (na začiatku každej novej cesty je táto pamäť prázdna a je aktualizovaná po každom kroku – navštívení nového mesta) a na rozdiel od skutočných mravcov je množstvo zanechaného feromónu funkciou kvality nájdeného riešenia. Virtuálny feromón umožňuje udržať v pamäti dobré riešenia, ktoré môžu slúžiť k nájdeniu ešte lepších riešení. Je potrebné vyhnúť sa tomu, aby niektoré dobré, ale nie dosť dobré riešenia viedli k predčasnému uviaznutiu algoritmu. Preto je implementovaná záporná spätná

väzba spočívajúca vo vyparovaní feromónu, čím je do algoritmu zahrnutá časová mierka. Podobnosti so skutočnými mravcami sú predovšetkým v kolónii spolupracujúcich mravcov, v použití feromónovej stopy a v pravdepodobnostnom rozhodovaní.

Mravce použité v kolóniách fungujú ako stochastické procedúry vytvárajúce nové riešenia interaktívnym pridávaním komponent (feromónová stopa) do čiastočného riešenia. Behom algoritmu každý mravec k, buduje cestu vykonaním n krokov. Iterácie sú indexované pomocou t, pričom t_{max} je užívateľom definovaný maximálny počet povolených iterácií. Mravce, ktoré pracujú v danej iterácii dobre, ovplyvňujú výskum mravcov v budúcich iteráciách, pretože tie využívajú feromónovú cestu ako sprievodcu pre svoje výskumy.

Na začiatku je m umelých mravcov umiestnených do náhodne vybraných miest grafu. Po každom kroku pokračujú do nového mesta a modifikujú feromónovú stopu na hranách – tzv. *lokálna aktualizácia feromónu*. Keď všetky mravce dokončia cestu, mravec, ktorý vykoná najkratšiu cestu, modifikuje hrany patriace k jeho ceste – tzv. *globálna aktualizácia feromónu* (pridanie množstva koncentrácie, ktoré je nepriamoúmerné dĺžke cesty).

Je niekoľko možností ako tento princíp použiť na vyriešenie TSP. Mravec k v meste r vyberá mesto s podľa nasledujúceho pravidla:

$$s = \begin{cases} \arg\max\left\{\left[\tau_{(r,u)}\right].\left[\eta_{(r,u)}\right]^{\beta}\right\}_{u \notin M_k}, & q \leq q_0 \\ S, & \text{inak} \end{cases}$$

kde $\tau_{(r,u)}$ je koncentrácia feromónu na hrane (r,u), $\eta_{(r,u)}$ je heuristická funkcia, ktorá je rovná inverznej hodnote vzdialenosti medzi mestami r a u ($\eta_{(r,u)} = 1/d_{(r,u)}$), β je nastaviteľný parameter, ktorý reguluje relatívnu váhu feromónu, q je náhodná veličina s rovnomerným rozložením $[0,1]$, q_0 je nastaviteľný parameter $0 \leq q_0 \leq 1$ a S je mesto náhodne vybrané podľa nasledujúcej pravdepodobnostnej funkcie, ktorá uprednostňuje hrany, ktoré majú väčšiu koncentráciu feromónu:

$$p_k(r,s) = \begin{cases} \dfrac{\left[\tau_{(r,u)}\right].\left[\eta_{(r,u)}\right]^{\beta}}{\sum_{u \notin M_k}\left[\tau_{(r,u)}\right].\left[\eta_{(r,u)}\right]^{\beta}}, & s \notin M_k \\ 0, & inak \end{cases}$$

kde $p_k(r,s)$ je pravdepodobnosť, s ktorou mravec k vyberá pohyb z mesta r do mesta s.

Feromónová stopa ja aktualizovaná lokálne aj globálne. Mravec, ktorý generuje najlepšiu cestu, ako jediný môže globálne aktualizovať koncentráciu feromónu. Okamžite ako mravce dokončia svoju cestu, nejlepší z nich vloží feromón na hrany, ktoré tvorili jeho cestu (ostatné hrany ostávajú nezmenené). Hodnota feromónu $\Delta\varphi(r,s)$ uloženého na každej navštívenej hrane (r,s) najlepšieho mravca je inverzne úmerná dĺžke cesty. Kratšia cesta má väčšiu koncentráciu feromónu uloženého na hranách. Globálna aktualizácia feromónu je aplikovaná len na hrany patriace k najlepšej trase od začiatku cesty. Aktualizačné pravidlo je nasledujúce:

$$\varphi_{(r,s)} \leftarrow (1-\alpha)\cdot\varphi_{(r,s)} + \alpha\cdot\Delta\varphi_{(r,s)}$$

kde (r,s) sú hrany patriace k najlepšej ceste, $\Delta\varphi_{(r,s)} = (najkratšia\ cesta)^{-1}$ a α je parameter riadiaci rozklad feromónu.

Lokálna aktualizácia feromónu je určená na vyhýbanie sa veľmi silným hranám vybraných veľkým počtom mravcov. Koncentrácia feromónu na mravcom vybranej hrane je vždy zmenená podľa lokálneho pravidla:

$$\tau_{(r,s)} \leftarrow (1-\alpha) \cdot \tau_{(r,s)} + \alpha \cdot \tau_0$$

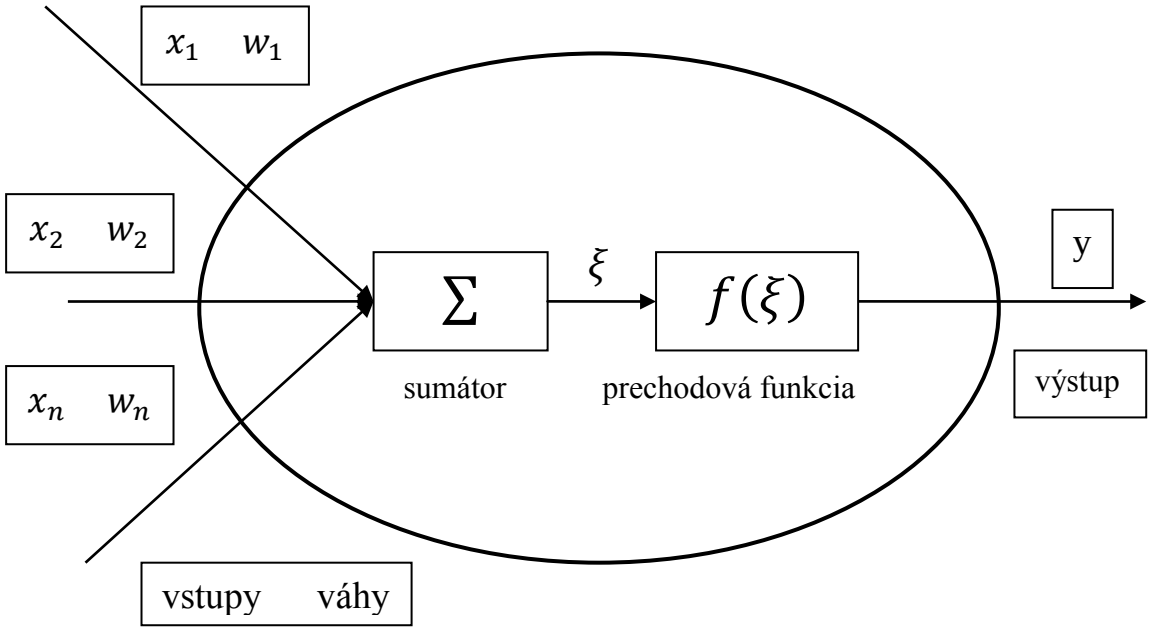
kde τ_0 je počiatočná hodnota feromónových stôp, pričom bolo experimentálne zistené, že nastavením $\tau_0 = (n.L_{nn})^{-1}$, kde n je počet miest a L_{nn} je dĺžka cesty produkovaná heuristikou najbližšieho suseda, dáva veľmi dobré výsledky.

5.4.4 Neurónové siete

Umelé neurónové siete (*neural network*) [11] pôvodne vznikli ako modely mozgu resp. nervového systému a jeho činnosti. Postupne sa však rozvinuli do nástrojov strojového učenia, používaných napr. na automatické rozpoznávanie priestorových objektov alebo časových postupností signálov, zhlukovú analýzu a pod.

Neurónová sieť je v podstate súborom jednoduchých prvkov (uzlov, neurónov) vykonávajúcich jednoduché operácie. Spojenia neurónov sú ohodnotené reálnymi číslami – váhami.

Predstavíme si umelý model neurónu – *perceptrón* (z angl. percept), viz obrázok 5.8.



Obrázok 5.8: Umelý model neurónu

Na vstup do neurónu privádzame vstupný vektor x (vzor) zložený z binárnych alebo reálnych čísel cez synaptické spojenia ohodnotené synaptickými váhami, ktoré tvoria váhový vektor w. Neurón má $n+1$ vstupov a hodnotu vstupu x_0 berieme vždy ako 1 a váhu w_0 považujeme za prah neurónu ϑ, môže byť aj nulový. Vnútorný potenciál neurónu je určený ako skalárny súčin vektorov w a x:

$$\xi = wx = \sum_{i=0}^{n} w_i x_i = \sum_{i=1}^{n} w_i x_i - \vartheta$$

Výstup (aktivita) z neurónu je potom určený pomocou *prechodovej* (aktivačnej) *funkcie* f, ktorú aplikujeme na vnútorný potenciál neurónu:

- $y = f(\xi)$

Prechodou funkciou je často funkcia skoková:

- $y = \begin{cases} 1\ pre\ \ \xi \geq 0 \\ 0\ pre\ \ \xi < 0 \end{cases}$

Táto funkcia však nie je diferencovateľná, a preto sa nedá použiť pri vstupoch, kedy sa parametre neurónu nastavujú pomocou derivácie prechodovej funkcie. Z tohto dôvodu sa často používa funkcia v tvare sigmoidy:

- $y = \frac{1}{1+e^{-k\xi}}$

Konštanta k udáva strmosť sigmoidy, zvyčajne sa používa hodnota 1.

Kohonenova sieť (Kohonen network)

Kohonenova SOM (SamoOrganizujúca sa Mapa) [11] patrí do kategórie sietí, ktoré sa učia bez učiteľa (unsupervised learning). To znamená, že algoritmus učenia nemá informáciu, aké váhy pri jednotlivých prvkoch siete sú správne, na rozdiel od sieti s dopredným šírením [11]. Základnou črtou Kohonenových sietí je schopnosť realizovať zobrazenie zachovávajúce topológiu tréningovej množiny dát. To znamená, že body (vstupné vektory), ktoré sa podobali svojimi hodnotami, sa premietnu na uzly, ktoré budú „ležať“ blízko seba. Termín „ležať blízko seba“ bude závislý od topológie siete, ktorá je väčšinou buď lineárna (uzly sú formálne usporiadané do reťazca) alebo mriežková (uzly sú usporiadané v dvojrozmernej pravouhlej mriežke). Susednosť v takýchto sieťach sa meria iba počtom uzlov, cez ktoré treba prejsť, aby sme sa dostali od jedného uzla k druhému, nejde teda o euklidovskú vzdialenosť. Uzly v reťazci majú dvoch najbližších susedov – naľavo a napravo, uzly v pravouhlej mriežke (uprostred mysleného štvorca) majú ôsmich najbližších susedov – hore, dolu, naľavo, napravo (uprostred myslených hrán štvorca) a naprieč na uhlopriečkach (na myslených rohoch štvorca). Každý z týchto bodov je ohodnotený vektorom (spočiatku náhodne vygenerovaným) rovnakého typu, ako sú vstupné vektory. Pri vkladaní vstupných dát sa pre každý vstupný vektor nájde „víťazný“ uzol, ktorého vektor má od daného vstupu najmenšiu (väčšinou euklidovskú) vzdialenosť. Hodnoty vektora tohto uzla sa potom zmenia podľa tzv. Hebbovho pravidla učenia tak, aby euklidovská vzdialenosť od daného vstupu bola menšia. Podobne sa danému vstupnému vektoru môžu priblížiť aj hodnoty vektorov uzlov, ktoré sú s „víťazným“ uzlom susedné. Pri lineárnej topológii to budú susedné uzly vľavo a vpravo. Môžeme uvažovať aj širšie okolie zahrňujúce viac uzlov vľavo a vpravo od víťazného uzla, no čím vzdialenejšie sú uzly, tým menej by sa hodnoty vektorov týchto uzlov mali blížiť hodnotám vstupného vektora. To je v pseudokóde zohľadnené funkciou $h(I_{min}, i)$, kde táto funkcia určuje, v akom okolí víťazného uzla a do akej miery budú adaptované ostatné uzly.

KOHONEN_NETWORK

1. Inicializujte váhy w_{ij}, kde vektor $w_i = (w_{i1}, w_{i2})$ je vektorom priradeným uzlu i.
2. Priraďte mestám topológiu cyklu.
3. Nastavte veľkosť učenia α.
4. Nastavte veľkosť okolia.
5. Kým nie je splnená ukončovacia podmienka opakute:
   - 5.1. $j = 1$ a n-krát opakujte:
     - 5.1.1. $i = 1$ a n-krát opakujete:
       - 5.1.1.1. $d(i) = \left(w_{i1} - x_j\right)^2 + (w_{i2} - y_j)^2$
     - 5.1.2. $I_{min} = i$, kde $d(i)$ je minimálne
     - 5.1.3. Pre víťazný uzol I_{min} a jeho okolie upravte váhy.
     - 5.1.4. $w_{i1}(\text{nová}) = w_{i1}(\text{stará}) + \alpha h(I_{min}, i)\left[x_j - w_{i1}(stará)\right]$
     - 5.1.5. $w_{i2}(\text{nová}) = w_{i2}(\text{stará}) + \alpha h(I_{min}, i)\left[y_j - w_{i2}(stará)\right]$
   - 5.2. Upravte rýchlosť učenia α.
   - 5.3. Zmenšite okolie uzla pri splnení zadanej podmienky.

Váhy v bode $(1.)$ sú generované náhodne v rozmedzí maximálnych a minimálnych hodnôt súradníc (x, y) miest, pre $i \in (1, \cdots, n)$, kde n je počet miest. Topológiu v bode $(2.)$ priradíme nasledovne: mesto i susedí naľavo s mestom $i - 1$ a napravo s mestom $i + 1$, mesto 1 susedí naľavo s mestom n a opačne, mesto n teda susedí napravo s mestom 1. V bode $(3.)$ nastavujeme hodnotu učenia α, napr. bude nastavená na $0{,}5$ a bude klesať na $0{,}4$. Veľkosť okolia v bode $(4.)$ na začiatku nastavíme na 1, tzn. že okrem víťazného uzla upravujeme vždy váhy aj jeho ľavého a pravého suseda. Rýchlosť učenia $(5.2.)$ môžeme upraviť nasledovne : $\alpha = \alpha(0{,}4/0{,}5)^{0{,}005}$. Najjednoduchšia používaná funkcia je:

$$h(I_{min}, i) = \begin{cases} 1, & d_M(I_{min}, i) \leq 1 \\ 0, & inak \end{cases}$$

kde $d_M(I_{min}, i)$ je vzdialenosť typu Manhattan medzi neurónmi I_{min} a i, teda počet hrán, po ktorých treba v danej topológii siete prejsť, aby sme sa dostali od neurónu I_{min} k neurónu i. Niekedy sa pre vzdialenejšie okolie volí dokonca „záporné“ učenie, teda že vektory uzlov vzdialených cez viac hrán od víťazného uzla sa „vzďaľujú“ od hodnôt vstupného vektora, zatiaľ čo ešte vzdialenejšie uzly zostávajú bez zmeny. Tento proces učenia sa opakuje stále dookola. Veľkosť okolia a parameter učenia (teda miera priblíženia vektora víťazného uzla vstupnému vektoru) sa znižujú s časom v priebehu algoritmu.

6 Implementácia

Porovnanie heuristík prebiehalo na základe vlastnej implementácie vybraných algoritmov. Na implementáciu bolo zvolené platforma .NET a jazyk C#. Implementovaný program ponúka grafické užívateľské rozhranie napísané vo Windows communication foundation. Toto vývojové prostredie bolo zvolené najmä z dôvodu rýchleho vývoja aplikácie, a pretože heuristiky boli porovnávané medzi sebou, nie je zvolenie tejto platformy problém ani z dôvodu horšieho výkonu oproti programu napísanému napr. v jazyku C. Program (ovládanie programu viz. Príloha 1.), okrem samotného riešenia úlohy, vizualizuje výsledok, umožňuje tvorbu vlastných grafov a užívateľ si môže kedykoľvek vizualizáciu grafu uložiť do formátu PNG.

Vstup programu môže byť zadaný užívateľom tak, že pomocou myši vytvorí rozmiestnenie vrcholov kompletného grafu, nad ktorými potom prebieha výpočet. Druhý spôsob spočíva v načítaní vstupných dát zo súboru. Program spracováva formát typu TSPLIB [13], body musia byť vo formáte euklidovských vzdialeností.

Vyššie popísané algoritmy umožňujú rôzne implementácie. Množina naprogramovaných heuristík bola volená tak, aby sa medzi sebou dali porovnávať rôzne skupiny založené na rozdielnych princípoch.

Heuristika najbližšieho suseda potrebuje určiť prvý bod, od ktorého sa má začať zostavovať hľadaný cyklus. Program vyberie vždy prvý načítaný bod zo súboru (prípadne prvý vytvorený bod užívateľom).

Podobne, heuristika vkladania potrebuje určiť počiatočný cyklus. V tomto prípade sú vybrané prvé tri vstupné body.

Christofidesova metóda obsahuje časť, kde je potrebné vybrať najlacnejšie párovanie nepárnych vrcholov. Pre tieto účely existuje celá rada metód, napr. Edmondsov algoritmus [2]. Implementácia týchto algoritmov je však náročná, preto bola zvolená priamočiara metóda, kedy sa pre prvý vrchol vyberie najkratšia vzdialenosť k ďalším z množiny nepárnych vrcholov, ktoré sa z množiny postupne odstraňujú. Algoritmus končí, ak je množina nepárnych vrcholov prázdna. Tento algoritmus nie je optimálny, avšak i napriek tomu poskytuje pomerne dobré výsledky a skracuje dobu behu algoritmu.

Heuristika zdvojenia kostry a tiež Christofidesova metóda potřebuju pres svoj štart minimálnu kostru. Kostra je vytvorená pomocou Kruskalovho algoritmu. Ďalej časť obidvoch heuristík využíva Eulerov ťah. Tento môže začínať v ľubovoľnom bode. Implementačne sa vyberie náhodný bod, od ktorého ťah začína. Vo výsledku to znamená, že táto skutočnosť ovplyvňuje výslednú dĺžku cyklu, preto niekoľko spustení týchto algoritmov môže zobraziť rozdielne trasy i výsledky.

Pre heuristiku 2-optimalizácie je možné vybrať vstupný cyklus z dvoch možností, a to cyklus vytvorený heuristikou najbližšieho suseda alebo metódou zdvojenia kostry. Algoritmus končí, ak optimalizácia nájde prvé lepšie riešenie než je vybraný vstupný cyklus alebo po odskúšaní všetkých možností.

Pre algoritmus simulovaného žíhania bol vybraný vstupný cyklus vytvorený metódou zdvojenia kostry. Parametre boli zvolené experimentálne tak, aby sa dosiahlo čo najlepšieho výsledku v pomere počet behov a najkratší cyklus.

Zdrojový kód je rozdelený do niekoľkých projektov. Prvý projekt obsahuje dátové typy používané pri implementácii algoritmov. Druhý projekt je samotná implementácia programu a posledný projekt je grafické užívateľské rozhranie.

7 Experimentálne porovnanie heuristík

V tejto práci sme sa venovali popisu najpoužívanejších heuristík na riešenie TSP. Mohlo by sa zdať, že ich je zbytočne veľa a stačilo by spomenúť len heuristiky, ktoré dosahujú najlepšie výsledky. No tu sa stretávame s problémom ako jednoznačne určiť víťaza. Heuristiky môžeme porovnávať v teoretickej alebo experimentálnej rovine. My sme sa zamerali na experimentálne porovnanie heuristík.

Experimentálne porovnanie sme realizovali pomocou nami naprogramovaných heuristík, na niekoľkých príkladoch z verejne dostupnej databáze a na náhodne vygenerovaných príkladoch. Výsledky, ktoré sme dosiahli týmto spôsobom môžu byť ovplyvnené kvalitou programovania (najmä čas výpočtu), ale všetky použité programy sú naprogramované jedným programátorom a v tom istom programovacom jazyku, preto môžeme dosiahnuté výsledky považovať za smerodajné.

Jednou z úloh bolo uviesť najlepšie cesty, ktoré boli nájdené jednotlivými heuristikami. Existuje voľne dostupný program *Concorde* [14] riešiaci TSP, avšak tento má implementované iné heuristiky než tie, ktoré porovnávame v tejto práci. Verejne dostupné [15] sú len optimálne cesty získané Lin-Kernighanovým algoritmom alebo jeho modifikovanou verziou od K. Helsgauna [9] *LKH*. Z týchto dôvodov nebolo možné porovnať naše riešenia s najlepšími možnými pre jednotlivé algoritmy. Porovnanie je teda založené na výsledkoch jednotlivých algoritmov medzi sebou a optimálnym riešením daného grafu uvedeným na [15]. Na porovnanie boli použité len tie vstupné dáta, u ktorých je uvedené optimálne riešenie. Tabuľku obsahujúcu vstupné dáta, ich optimálne riešenia a riešenia nájdené jednotlivými implementovanými metódami nájdete v Príloha 2.

Porovnanie heuristík pozostávalo z naprogramovania jednotlivých metód a následného testovania na príkladoch. Z dôvodu veľkej časovej náročnosti niektorých algoritmov nebolo možné tieto algoritmy testovať na úlohách väčších rozmerov, preto boli z niektorých testov vynechané.

Popri kvalite nájdeného riešenia testuje každý algoritmus aj čas potrebný na beh heuristiky. Zo získaných výsledkov sme vypočítali štatistické ukazovatele ako je priemerná dĺžka cyklu, odchýlka od optimálneho cyklu a pod. Pretože sme nemali k dispozícii známe riešenia, odchýlky sa vzťahujú na výsledok najlepšej heuristiky pre daný príklad.

Do testovania boli zahrnuté nasledujúce heuristiky (všetky boli popísané v 5.kapitole):

NN – Heuristika najbližšieho suseda
INS – Heuristika vkladania
2TREE – Metóda zdvojenia kostry
CHRF – Christofidesova metóda kostry a párenia
2OPT – 2-optimalizácia
SA – Simulované žíhanie

Na základe spôsobu, akým heuristiky vytvárajú hamiltonovský cyklus sme ich rozdelili do dvoch skupín:

1. skupina = {NN, INS, 2TREE, CHR}
2. skupina = {2OPT-NN, 2OPT-2TREE, SA-2TREE}

Do prvej skupiny sú zaradené heuristiky, ktoré konštruujú hamiltonovský cyklus, ale nijak ho neupravujú. Druhá skupina pozostáva z heuristík, ktorých vstupom je cyklus vytvorený niektorým algoritmom z prvej skupiny. To znamená, že vylepšujú už známe riešenie. Pretože sme chceli otestovať aj citlivosť heuristík na vstupný cyklus, v druhej skupine sa u 2-optimalizácie porovnávajú dve verzie. Prvou verziou je vstupný cyklus získaný heuristikou najbližšieho suseda (prípona NN) a druhou je vstupný cyklus získaný metódou zdvojenia kostry (prípona 2TREE).

Všetky údaje použité v tejto kapitole vychádzajú z konkrétnych testov, z ktorých niektoré uvádzame v prílohe.

7.1 Porovnanie 1.skupiny heuristík

Naprogramované heuristiky boli porovnávané na súbore 400 náhodne vygenerovaných príkladoch. Rozpätie generovaných príkladov bolo 25 – 200 bodov v euklidovskej rovine. Body predstavujú vrcholy kompletného grafu a dĺžky hrán boli definované ako euklidovské vzdialenosti medzi vrcholmi. V nasledujúcej tabuľke sú uvedené priemerné dĺžky cyklov testovaných heuristík.

Nearest neighbor	Insertion heuristic	Doubletree heuristic	Christofides
4081,1	4991,85	4268,35	3944,15

Tabuľka 7.1: Priemerné hodnoty dĺžky cyklov

Z tabuľky 7.1 je viditeľné, že rozdiely medzi jednotlivými heuristikami nie sú veľké, no poradie kvality je jednoznačné. Ako najlepšia heuristika vychádza Christofides, ďalej v poradí nasledujú: Nearest neighbor, Doubletree heuristic, Insertion heuristic.

Ďalším ukazovateľom kvality heuristiky je priemerná a maximálna odchýlka od optimálneho riešenia. Pretože sme nemali k dispozícii optimálne riešenia, odchýlky sa vzťahujú k najlepšiemu nájdenému riešeniu.

Odchýlka [%]	Nearest neighbor	Insertion heuristic	Doubletree heuristic	Christofides
Priemerná	7,008634	30,56033	11,10818	2,548583
Maximálna	25,37441	62,74964	25,0539	16,05124

Tabuľka 7.2: Hodnoty priemernej a maximálnej odchýlky v %

Najnižšie hodnoty u Christofidesovej heuristiky naznačujú, že práve táto metóda bolo najčastejším víťazom v našom experimentálnom porovnaní.

Ako posledným zo štatistických údajov je časová náročnosť heuristík. V tabuľke 7.3 sú údaje uvádzané v sekundách a vyjadrujú priemerný a maximálny čas strávený výpočtom jedného príkladu.

Časová náročnosť [s]	Nearest neighbor	Insertion heuristic	Doubletree heuristic	Christofides
Priemerná	0,03335	6,74485	0,036475	0,137871
Maximálna	0,2	48,722	0,1665	1,21

Tabuľka 7.3: Časová náročnosť heuristík

Najmenšiu časovú náročnosť má heuristika najbližšieho suseda. Aj napriek svojej jednoduchosti nachádza u úloh menších rozmerov prijateľné riešenia.

Aj napriek vyššej časovej náročnosti sa jednoznačným víťazom v skupine konštrukčných heuristík stala Christofidesova metóda kostry a párenia. Z testov jasne vyplýva, že bola najčastejším víťazom a ostatné výsledky heuristík boli porovnávané práve s jej výsledkom.

7.2 Porovnanie 2.skupiny heuristík

Vylepšovacie heuristiky sme testovali na rovnakej sade príkladov ako prvú skupinu. 2-optimalizačnú heuristiku sme otestovali aj na citlivosť vstupného cyklu. Experimentálne získané hodnoty priemerných dĺžok cyklov uvádzame v nasledujúcej tabuľke.

2OPT_NN	2OPT_2TREE	Simulated annealing
4072,6	4185,1	4363,7

Tabuľka 7.4: Priemerné hodnoty dĺžky cyklov

Ako je viditeľné už na prvý pohľad z tabuľky, rozdiely medzi heuristikami sú ešte menšie než v predchádzajúcom prípade. Poradie kvality heuristík od najlepšej: 2-opt_NN, 2-opt_2TREE, Simulated annealing. Tento výsledok naznačuje, že simulované žíhanie potrebuje pre svoju optimálnu činnosť značnú voľnosť a uzavretie do lokálneho minima môže nepriaznivo ovplyvniť kvalitu riešenia.

Na základe nasledujúcej tabuľky je predpoklad, že nastavený rozmer testovacích príkladov bol pravdepodobne malý a z tohto dôvodu sa nepreukázala predpokladaná sila simulovaného žíhania. Tabuľka podáva prehľad priemernej a maximálnej odchýlky od najlepšieho riešenia.

Odchýlka [%]	2OPT_NN	2OPT_2TREE	Simulated annealing
Priemerná	3,403124	5,266743	10,57996
Maximálna	27,48511	19,09568	23,57691

Tabuľka 7.5: Hodnoty priemernej a maximálnej odchýlky v %

Ako víťaz v skupine zlepšovacích heuristík vyšla v našom testovaní 2-optimalizácia. Z výsledkov ďalej vyplýva, že účinnejším variantom je 2-optimalizácia so vstupným cyklom získaným heuristikou najbližšieho suseda. Citlivosť heuristiky na vstupný cyklus však nie je veľmi veľká, čo potvrdzuje a minimálny rozdiel v kvalite riešení.

Nemenej dôležitým faktorom je aj časová náročnosť. Úspešnosť heuristiky je charakterizovaná kvalitou výsledku, ale aj časom realizácie. Priemerné a maximálne časy strávené výpočtom jedného príkladu sú uvedené v nasledujúcej tabuľke.

Časová náročnosť [s]	2OPT_NN	2OPT_2TREE	Simulated annealing
Priemerná	1,3662	0,118383	1,839875
Maximálna	24,003	1,038	31,203

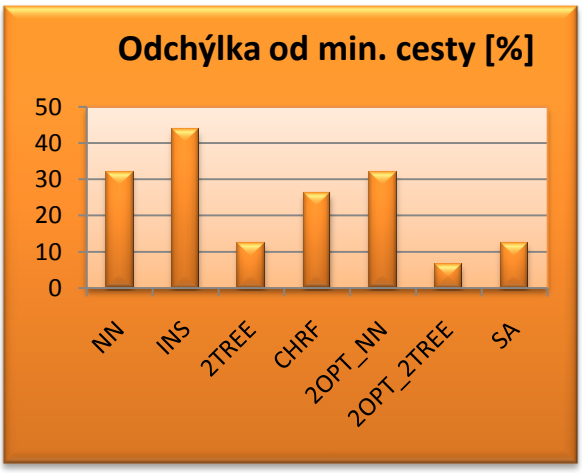
Tabuľka 7.6: Časová náročnosť heuristík

Z tabuľky je zrejmé, že najkvalitnejšie výsledky heuristiky 2-opt_NN sú vykúpené vyššou časovou náročnosťou.

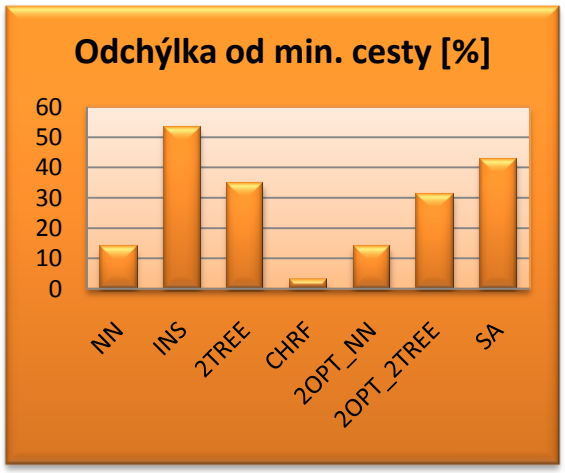
Z výsledkov druhého porovnania sa dá predpovedať, že na príkladoch väčších rozmerov budeme musieť povoliť väčší počet opakovaní na jednotlivých teplotách čo však výrazne zvýši časovú náročnosť.

7.3 Porovnanie heuristík na internetových úlohách

Posledné porovnanie heuristík sme realizovali na niekoľkých príkladoch z verejne dostupnej databázy. Výhodou týchto príkladov boli už známe riešenia, a tak sme mohli lepšie posúdiť kvalitu našich výsledkov. Najskôr sme heuristiky otestovali na troch menších príkladoch (wi29.tsp, dj38.tsp a qa194.tsp). Prehľad výsledkov tohto testu poskytujú nasledujúce grafy.



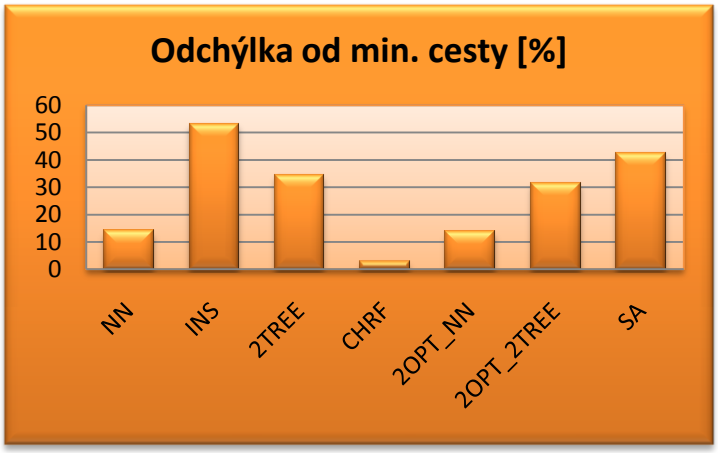
Graf 7.1: wi29.tsp



Graf 7.2: dj38.tsp

Grafy znázorňujú percentuálnu odchýlku od známeho optimálneho riešenia úloh wi29.tsp a dj38.tsp. Najúspešnejšou heuristikou sa v prvom prípade stala heuristika 2OPT_2TREE s odchýlkou 6,764%. V druhom prípade mala najkvalitnejší výsledok heuristika CHRF s odchýlkou 3,034%. Vo výsledkoch obidvoch úloh sú zreteľné rozdiely medzi heuristikami.

Heuristiky sme nakoniec otestovali na úlohe qa194.tsp. Na rozdiel od predchádzajúcich dvoch úloh sú rozdiely vo výsledkoch jednotlivých heuristík minimálne.



Graf 7.3: qa194.tsp

Aj napriek minimálnym rozdielom je jednoznačným víťazom heuristika NN s odchýlkou 27,94%. Naopak najhoršie výsledky dosahuje heuristika vkladania.

Po prvej sade testov vyvstala otázka či si heuristiky zachovajú svoje kvality aj na úlohách väčších rozmerov. Z tohto dôvodu sme z verejne dostupenej databázy vybrali úlohy, ktorých počet bodov bol v rozpätí cca 600 – 2000. Priemerná odchýlka alebo priemerná dĺžka cyklov by vzhľadom na rozmanitosť úloh nebola jednoznačným ukazovateľom kvality heuristík. Preto uvádzame pomer medzi nami nájdeným riešením a optimálnym riešením.

Keďže sme boli obmedzení štruktúrami programovacieho jazyka, algoritmy CHRF a INS sme nemohli otestovať na príkladoch väčších rozmerov – sú pamäťovo veľmi náročné.

úloha	NN	2TREE	2OPT_NN	2OPT_2TREE	SA
xql662.tsp	1,294	1,386	1,293	1,391	1,400
uy734.tsp	1,312	1,373	1,312	1,379	1,391
rbu737.tsp	1,281	1,402	1,281	1,412	1,425
zi929.tsp	1,193	1,391	1,192	1,393	1,397
lim963.tsp	1,242	1,456	1,242	1,457	1,486
lu980.tsp	1,267	1,474	1,266	1,475	1,513
xit1083.tsp	1,293	1,424	1,293	1,433	1,481
dca1389.tsp	1,252	1,455	1,252	1,449	1,497
rw1621.tsp	1,285	1,524	1,285	1,534	1,550
mu1979.tsp	1,392	1,384	1,391	1,391	1,409
djb2036.tsp	1,255	1,463	1,255	1,462	1,470

Tabuľka 7.7

Z tabuľky jasne vyplýva, že až na jednu výnimku sú všetky výsledky maximálne 1,5-krát horšie než známe najlepšie riešenie. Ako jasný víťaz vyšla heuristika 2OPT_NN, keď dosiahla najlepší výsledok takmer vo všetkých testovaných úlohách. Kvalita riešenia však vplýva na čas realizácie, ktorý je druhý najvyšší spomedzi nami naprogramovaných heuristík.

Najhoršie výsledky ako v kvalite riešenia, tak i čo sa časovej náročnosti týka dosiahla heuristika simulovaného žíhania.

7.4 Zhrnutie experimentálneho porovnania

Výsledky, ku ktorým sme dospeli, nepreukázali ani jednu heuristiku jednoznačne lepšou než ostatné, no došli sme k určitým záverom. Záleží na našich prioritách či obetujeme čas výpočtu kvôli kvalitnejšiemu riešeniu, alebo si vystačíme s riešením, ktoré nie je až tak dobré, za veľmi krátky čas.

Zo skupiny konštrukčných heuristík sa však ako najlepšia javí Christofidesova metóda kostry a párenia, ktorá dáva pomerne dobré výsledky za približne rovnaký čas ako heuristika najbližšieho suseda, ktorá sa ukázala ako najrýchlejšia.

V skupine zlepšovacích heuristík sa víťaz hľadá zložitejšie, pretože na úlohách väčších rozmerov boli všetky pomerne vyrovnané.

Čo sa týka kvality riešenia, nepreukázala sa nijak výrazná citlivosť na vstupný cyklus. Pri vhodnej voľbe vstupného cyklu však môže pozorovateľne klesnúť časová náročnosť algoritmu.

Z výsledkov nášho experimentálneho testovania sme usúdili, že vhodnou kombináciou viacerých heuristík môžeme získať kvalitné riešenie za prijateľný čas.

8 Ďalší možný vývoj projektu

Na projekte je možné pokračovať rôznymi spôsobmi. V prvej rade to je implementácia ďalších heuristík (viaceré boli popísané v tejto práci).

Ďalším možným pokračovaním môže byť optimalizácia existujúcej implementácie algoritmov. Projekty v súčasnej dobe neriešia pamäťovú a časovú náročnosť (ako implementácia algoritmov, tak i porovnanie), ktoré sú však v praktickom využití rozhodne podstatné. I samotná implementácia dáva priestor k rôznym vylepšeniam, napr. výmena či optimalizácia algoritmov na hľadanie najlacnejšieho párovania nepárnych vrcholov v Christofidesovej metóde.

Aplikácie je možné vylepšovať i z hľadiska užívateľského rozhrania, kde by bolo napríklad vhodné umožniť užívateľovi ukladať vytvorené grafy a znovu je načítať, alebo voliť parametre algoritmov, napr. pri heuristike simulovaného žíhania.

Priestor na rozšírenie práce je i z hľadiska porovnania. Okrem vyššie zmieneného porovnania z hľadiska časovej a pamäťovej náročnosti, porovnanie vlastnej implementácie algoritmov s inými existujúcimi implementáciami rovnakých algoritmov, porovnanie implementácie algoritmov napísaných v rôznych programovacích jazykoch, prípadne ich behu na rôznych fyzických strojoch a pod.

9 Záver

Cieľom tejto bakalárskej práce bolo podať prehľad heuristík na riešenie problému obchodného cestujúceho, experimentálne ich porovnať a vyhodnotiť výsledky.

V prehľade sme uviedli najpoužívanejšie heuristiky rozdelené do troch skupín – konštrukčné, iteratívne a stochastické. Šesť z nich sa nám podarilo implementovať a otestovať na dostatočnom množstve úloh.

Výsledky experimentálneho testovania nám poskytli jednoznačné riešenia pre konštrukčné ako aj pre vylepšovacie algoritmy.

Tým sa nám podarilo splniť všetky vytýčené ciele. Naviac, výsledky nášho testovania umožňujú lepšie využitie heuristík, a tým aj zlepšenie výsledkov.

Rozšírením zadania tejto bakalárskej práce bolo vytvorenie aplikácie a jej grafického užívateľského rozhrania, ktorá nám poslúžila pri hľadaní implementačných chýb, no hlavne pri experimentálnom testovaní heuristík.

Literatúra

[1] BOSÁK, Juraj. *Grafy a ich aplikácie*. 2. Vydanie. Alfa, Bratislava, 1986. 176 s.

[2] PLESNÍK, Ján. *Grafové algoritmy*.VEDA, Bratislava, 1983. 344 s.

[3] HUŠEK, Roman, MAŇAS, Miroslav. *Matematické modely v ekonomii*. SNTL, Praha, 1989. 404 s. ISBN 80-03-00098-X.

[4] DELGADO, Frank Vega. *A Solution to the P versus NP Problem* [online]. Posledná modifikácia: 28. marca 2011. [cit. 2011-05-12]. Dostupný z WWW: <http://arxiv.org/ftp/arxiv/papers/1011/1011.2730.pdf>.

[5] DEOLALIKAR, Vinay. $P \neq NP$ [online]. 6. augusta 2010. [cit. 2011-05-12]. Dostupný z WWW: <http://www.scribd.com/doc/35539144/pnp12pt>.

[6] JÜNGER, M., REINELT, G., RINALDI, G. *The traveling salesman problem*. 1994. 112 s.

[7] JOHNSON, David S., MCGEOCH, Lyle A. *The Traveling Salesman Problem: A Case Study in Local Optimization*. 20. novembra 1995. 103 s.

[8] HYNEK, Josef. *Genetické algoritmy a genetické programování*. Grada Publishing, Praha, 2008. 200 s. ISBN 978-80-247-2695-3.

[9] HELSGAUN, Keld. *An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic*. 71 s.

[10] APPLEGATE, D., COOK, W., ROHE, A. Chained Lin-Kernighan for large traveling salesman problems. *INFORMS Journal on Computing*. Winter 2003, 15, 1, s. 82.

[11] KVASNIČKA, V., POSPÍCHAL, J., TIŇO, P. *Evolučné algoritmy*. STU, Bratislava, 2000. 223 s. ISBN 80-227-1377-5.

[12] DORIGO, Marco, GAMBARDELLA, Luca Maria. *Ant colonies for the traveling salesman problem*. 1996. 10 s.

[13] REINELT, Gerhard. *TSPLIB 95* [online].[cit. 2011-05-15]. Dostupný z WWW: <http://www.cs.bris.ac.uk/Teaching/Resources/COMSM0302/coursework/doc.pdf>.

[14] COOK, Wiliam. *Georgia Institute of Technology* [online]. Máj 2011 [cit. 2011-05-16]. The Traveling Salesman Problem. Dostupné z WWW: <http://www.tsp.gatech.edu/concorde/downloads/downloads.htm>.

[15] COOK, William. *Georgia Institute of Technology* [online]. Apríl 2011 [cit. 2011-05-16]. The Traveling Salesman Problem. Dostupné z WWW: <http://www.tsp.gatech.edu/world/countries.html>.

Zoznam príloh

Príloha 1. Ovládanie aplikácie
Príloha 2. Výsledky experimentálneho testovania
Príloha 3. CD nosič obsahujúci spustiteľnú verziu aplikácie, zdrojové kódy, vzorové dáta vo formáte TSPLIB a text tejto bakalárskej práce v elektronickej forme

Príloha 1.

Ovládanie aplikácie

Po spustení aplikácie sa objaví základné okno obsahujúce hlavné menu, informačný panel (v spodnej časti okna) a plochu zobrazovania grafu (viz. Obrázok 0.1). Všetky akcie sú dostupné cez hlavné menu aplikácie. Informačný panel ponúka základné informácie ako je dĺžka výsledného cyklu, čas výpočtu, zvolený režim a veľkosť grafu. Plocha zobrazovania grafu slúži na vizualizáciu grafu a manipuláciu s ním. Program pracuje v dvoch režimoch, v prvom užívateľ vytvára graf sám (insert mode), v druhom načíta vstupné dáta zo súboru (open mode).

Prvá položka menu, FILE, obsahuje obecné možnosti:

- Blank window – vymaže obsah plochy, uvoľní načítané dáta, prípadne vymaže definované body a nastaví program do insert módu
- Open file – otvára dialog pre načítanie súboru a po jeho načítaní nastavuje program do open módu
- Export to PNG – umožňuje uložiť práve zobrazený graf do formátu PNG
- Exit – ukončuje program

Položka METHODS slúži k voľbe metódy na výpočet hamiltonovského cyklu. Každá možnosť zodpovedá jednému implementovanému algoritmu.

HELP METHODS obsahuje len jednu možnosť – výpočet minimálnej kostry pomocou Kruskalovho algoritmu.

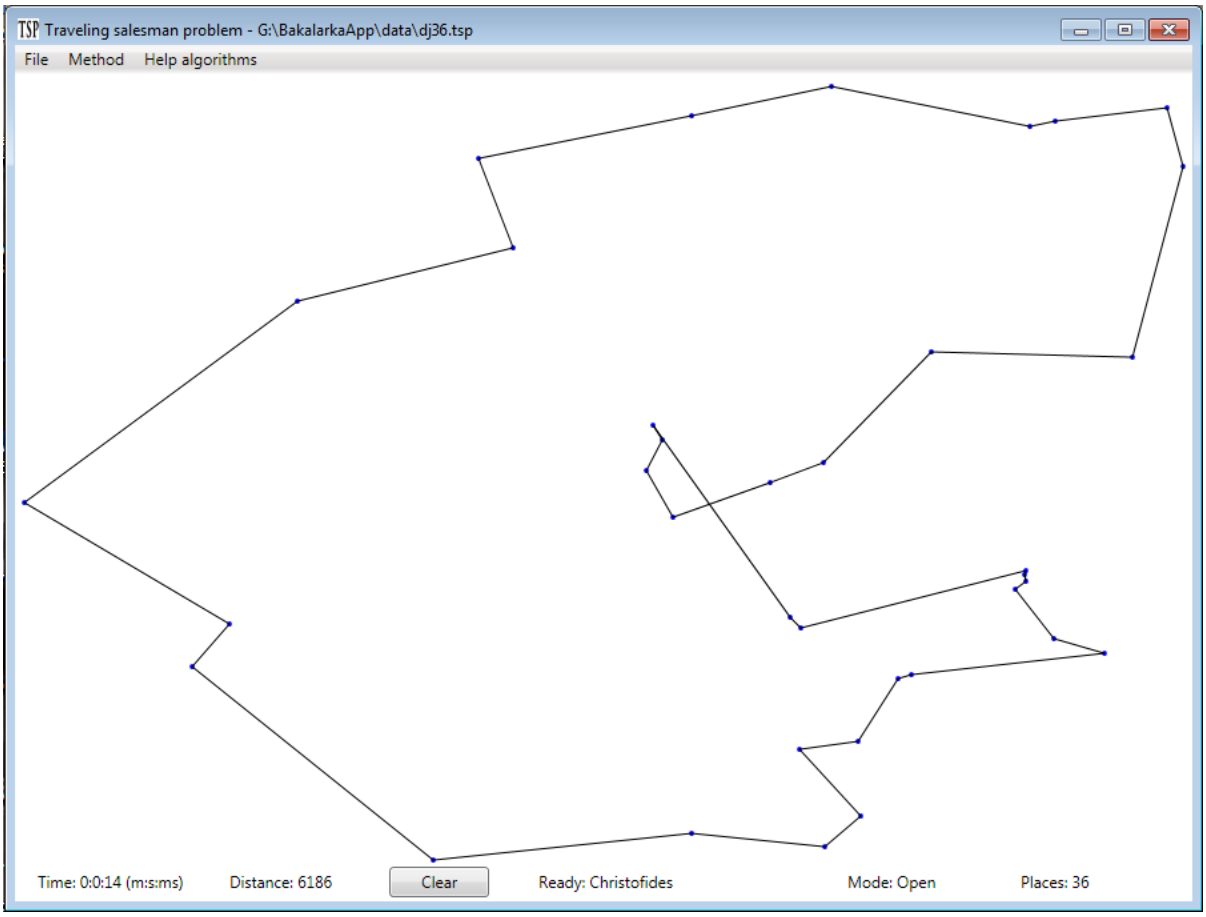
Posledná položka INSERT MODE rozhoduje o vykonávanej akcii v insert móde. Na výber sú tri možnosti:

- Insert – vkladanie bodov
- Move – pohyb bodu
- Delete – zmazanie bodu

Akcia move prebieha tak, že po kliknutí na nejaký bod, tento zmizne a užívateľ môže vložiť iný.

Tlačidlo Clear umiestnené v informačnom paneli slúži na zmazanie cesty, na ploche zostanú len definované body.

Výpočet cesty prebieha v jednom vlákne spolu s užívateľským rozhraním, preto po zahájení výpočtu až do jeho dokončenia užívateľské rozhranie neodpovedá. Čas i dĺžka cesty sa tiež objavia až po dokončení výpočtu.



Obrázok 0.1: Snímka aplikácie

Príloha 2.

Výsledky experimentálneho testovania

úloha	Min. cesta	NN	2TREE	2OPT_NN	2OPT_2TREE	SA
xql662.tsp	2513	3251	3483	3250	3495	3517
uy734.tsp	79114	103812	108657	103802	109121	110011
rbu737.tsp	3314	4245	4645	4244	4680	4723
zi929.tsp	95345	113788	132626	113653	132848	133203
lim963.tsp	2789	3465	4061	3463	4063	4144
lu980.tsp	11340	14370	16719	14351	16727	17162
xit1083.tsp	3558	4602	5065	4599	5099	5269
dca1389.tsp	5085	6368	7399	6367	7367	7614
rw1621.tsp	26051	33477	39705	33467	39959	40389
mu1979.tsp	86891	120919	120283	120908	120830	122436
djb2036.tsp	6197	7778	9064	7777	9058	9110