VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMATIZACE A INFORMATIKY
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

PROBLÉM OBCHODNÍHO CESTUJÍCÍHO S ČASOVÝMI OKNY
TRAVELING SALESMAN PROBLEM WITH TIME WINDOWS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Dávid Pavlovič**
AUTHOR

VEDOUCÍ PRÁCE — **RNDr. Jiří Dvořák, CSc.**
SUPERVISOR

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav automatizace a informatiky
Student:	**Bc. Dávid Pavlovič**
Studijní program:	Strojní inženýrství
Studijní obor:	Aplikovaná informatika a řízení
Vedoucí práce:	**RNDr. Jiří Dvořák, CSc.**
Akademický rok:	2020/21

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Problém obchodního cestujícího s časovými okny

Stručná charakteristika problematiky úkolu:

Klasický problém obchodního cestujícího (TSP) spočívá v tom, že agent navštíví všechna zadaná místa, přičemž každé místo navštíví právě jednou a vrátí se do výchozí pozice. Tento problém má řadu variant. Jednou z nich je problém TSP s časovými okny, kde agent musí navštívit každé místo v předem daném časovém okně. Cílem může být minimalizace času dokončení cesty nebo nákladů na provedení cesty.

Cíle diplomové práce:

1. Analyzovat problém obchodního cestujícího s časovými okny.
2. Implementovat vybrané metody řešení tohoto problému.
3. Provést a vyhodnotit ověřovací a srovnávací experimenty.

Seznam doporučené literatury:

KONA, H., BURDE, A. A Review of Traveling Salesman Problem with Time Window Constraint. IJIRST – International Journal for Innovative Research in Science & Technology, vol. 2, issue 1, 2015, pp. 253-256.

ROBERTI, R., WEN, M. The Electric Traveling Salesman Problem with Time Windows. Transportation Research. Part E: Logistics and Transportation Review, vol. 89, 2016, pp. 32-52. https://doi.org/10.1016/j.tre.2016.01.010.

KAABACHI, I., JRIJI, D., MADANY, F., KRICHEN, S. A Bi-criteria Ant Colony Optimization for Minimizing Fuel Consumption and Cost of The Traveling Salesman Problem With Time Windows. Procedia Computer Science, vol. 112, 2017, pp. 886-895, ISSN 1877-0509, https://doi.org/10.1016/j.procs.2017.08.105.

AMGHAR, K., CORDEAU, J., GENDRON, B. A General Variable Neighborhood Search Heuristic for the Traveling Salesman Problem with Time Windows under Completion Time Minimization. 2019, CIRRELT-2019-29.

BRETIN, A., DESAULNIERS, G., ROUSSEAU, L. The traveling salesman problem with time windows in postal services. Journal of the Operational Research Society, 2020, DOI: 10.1080/01605682.2019.1678403.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2020/21

V Brně, dne

L. S.

doc. Ing. Radomil Matoušek, Ph.D. ředitel ústavu	doc. Ing. Jaroslav Katolický, Ph.D. děkan fakulty

Abstrakt

Táto práca sa zaoberá riešením problému obchodného cestujúceho s časovými oknami. Problém spočíva v tom, že obchodný cestujúci musí prejsť každým definovaným miestom práve raz, a nakoniec sa vrátiť do pôvodného miesta za čo najnižšiu cenu trasy. Časové okná v tomto probléme definujú určité časové úseky, v ktorých môže byť každé konkrétne miesto navštívené, alebo sa môže stať, že v istom časovom úseku nebude existovať cesta medzi niektorými miestami. Práca sa zaoberá prehľadom tohto problému a problémov jemu podobných. Ďalej sa zaoberá popisom rôznych metód, ktorými tento problém možno riešiť. V rámci tejto práce bola tiež vytvorená aplikácia v programovacom jazyku Python, ktorá slúži na testovanie vybraných metód riešenia. Na záver sú vyhodnotené dané experimenty a porovnaná efektivita daných stratégií.

ABSTRACT

This thesis deals with the Travelling salesman problem with time windows. The problem is that the travelling salesman must pass through each defined location exactly once and finally return to the original place for the lowest possible price. The time windows in this problem are that each place can only be visited in a given time range, or it can happen that in a certain period of time there will be no path between some places. The thesis deals with an overview of this problem and problems similar to it. It also deals with the description of various methods by which this problem can be solved. As part of this thesis, an application in the Python programming language was also created, which is used to test selected methods for finding solutions. Finally, the given experiments are evaluated and the effectiveness of the given strategies is compared.

Kľúčové slová

Problém obchodného cestujúceho s časovými oknami, algoritmy umelej inteligencie, kombinatorická optimalizácia, plánovanie

KEYWORDS

Travelling salesman problem with time windows, artificial intelligence algorithms, combinatorial optimization, planning

2021

BIBLIOGRAFICKÁ CITÁCIA

PAVLOVIČ, Dávid. *Problém obchodního cestujícího s časovými okny*. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky, 2021, 73 s. Diplomová práca. Vedúci práce: RNDr. Jiří Dvořák, CSc.

Vyhlásenie autora o pôvodnosti diela

Prehlasujem, že táto diplomová práca je mojím pôvodným dielom, vypracoval som ju samostatne pod vedením vedúceho práce RNDr. Jiřího Dvořáka, CSc. a s použitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry.

Ako autor uvedenej práce ďalej prehlasujem, že v súvislosti s vytvorením tejto práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a som si plne vedomý následkov porušenia ustanovení § 11 a následného autorského zákona č. 121/2000 Sb., vrátane možných trestnoprávnych dôsledkov.

V Brne dňa 21. 5. 2021 .....

Dávid Pavlovič

Poďakovanie

Chcel by som týmto poďakovať svojej rodine, ktorá pri mne stála vždy – v dobrom aj v zlom – a podporovala ma v mojich rozhodnutiach. Tiež by som sa rád poďakoval vedúcemu tejto práce, pánovi RNDr. Jiřímu Dvořákovi, CSc. za jeho odborné pripomienky a trpezlivosť. V neposlednom rade sa chcem tiež poďakovať môjmu dobrému kamarátovi Ondrejovi Č. za užitočné rady pri tvorbe tejto práce a počas celého štúdia.

Obsah

1 Úvod

Táto práca se venuje Problému obchodného cestujúceho s časovými oknami (ďalej iba TSP-TW – z anglického Travelling salesman problem with time windows). V úvode si popíšeme tento problém a pohľad naň. Tiež si stručne popíšeme obsah jednotlivých kapitol jeto práce.

1.1 Úvod do problému obchodného cestujúceho s časovými oknami

Problém obchodného cestujúceho (ďalej iba TSP – z anglického Travelling salesman problem) je známy kombinatorický problém, ktorý je intenzívne študovaný v odbore optimalizácie a má tiež veľký význam v operatívnom výskume. Riešenie tohto problému má veľkú škálu využití, ale obecne sa najviac využíva v doprave.

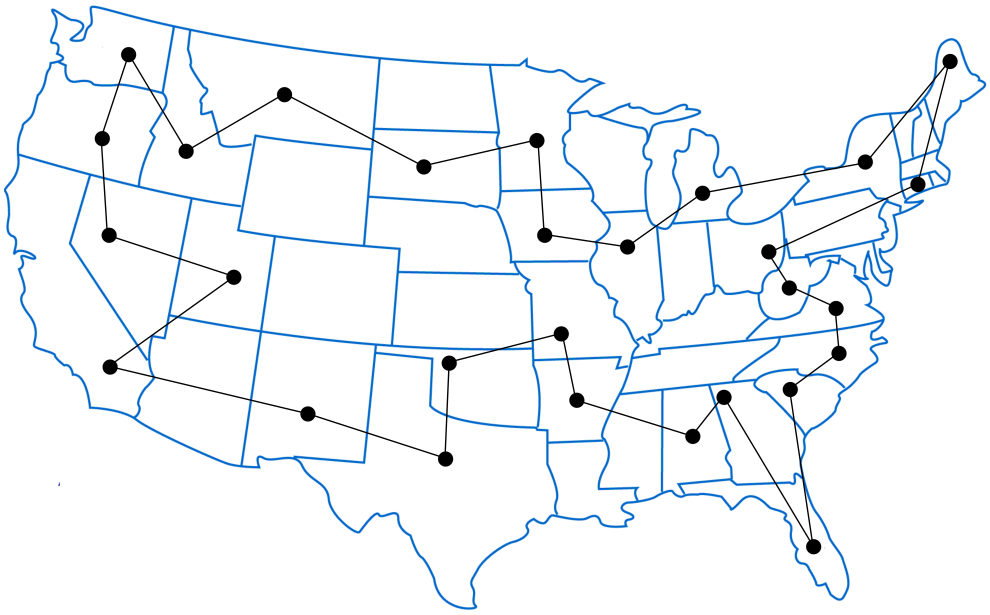
TSP môžeme definovať nasledovne: obchodný cestujúci sa snaží nájsť najkratšiu trasu začínajúcu v určenom meste, pričom každým mestom nachádzajúcim sa na trase musí prejsť práve jeden raz a cestu musí ukončiť v meste, z ktorého pôvodne vyrazil. Tento problém možno demonštrovať aj na grafe. Predstavme si, že obchodný cestujúci pozná graf $G(V, E)$, v ktorom uzly V korešpondujú s určitými miestami na mape (napríklad mestami) a hrany E indukujú prepojenia medzi týmito miestami (napríklad diaľnice medzi mestami). Každá hrana E je pritom asociovaná s konkrétnou reálnou nezápornou cenou, ktorú môžeme interpretovať ako dĺžku trasy, cenu za pohonné hmoty potrebné na prejdenie tejto trasy, časové ohodnotenie trasy a pod. Obchodný cestujúci sa teda snaží nájsť najkratší existujúci Hamiltonovský ťah v tomto grafe – ťah, ktorý bude obsahovať každý uzol práve jeden raz a začína aj končí v tom istom uzle. TSP je označený ako NP-ťažký problém. Znamená to, že je ľahko opísateľný, ale nájsť jeho riešenie je náročné. Zložitosť tohto problému je $O(n!)$ pre nájdenie riešenia s n mestami (uzlami). Znamená to, že s pribúdajúcim počtom miest sa nájdenie riešenia tohto problému stáva náročnejším.

Tento problém bol prvý raz načrtnutý v roku 1930, odkedy nepretržité výskumné práce venujúce sa TSP prišli s mnohými riešeniami znížujúcimi čas potrebný na vyriešenie TSP. Tento problém leží už dlhé roky v jadre riadenia distribúcie. Tomuto problému čelia dennodenne tisíce spoločností a organizačných skupín po celom svete, ktoré zabezpečujú a riešia distribúciu tovaru, prepravu materiálu alebo ľudí. Väčšina softvérových algoritmov sa v tejto oblasti zameriava na obmedzený počet prototypových problémov, ale budovaním dostatočnej flexibility v optimalizácii je možné ich prispôsobiť rôznym praktickým kontextom [1].

Táto práca sa bude konkrétnejšie zaoberať typom TSP, ktorý sa nazýva Problém obchodného cestujúceho s časovými oknami. Má niekoľko dôležitých praktických využití vrátane plánovania cesty. Tento typ problému je považovaný za NP-ťažký problém a teda použitie tradičných optimalizačných algoritmov pri jeho riešení nemusí viesť k optimálnym výsledkom, obzvlášť pri ich aplikácii na riešenie úloh väčšieho rozsahu. Z uvedeného dôvodu bola riešeniu tohto typu problémov venovaná zvýšená pozornosť.

Podproblém TSP-TW môžeme definovať nasledovne: každé mesto (alebo uzol, zákazník a pod.) musí byť navštívené vo vopred stanovenom časovom úseku a celková trasa má byť absolvovaná v čo najkratšom čase. Cestujúci teda musí v danom mieste počkať, ak sa doň dostal s predstihom a tento čas sa započíta do ceny celkovej cesty.

V tejto práci sa budeme venovať spôsobom hľadania čo najlepšej (najkratšej) možnej trasy, pričom navštívené miesta budú reprezentované skutočnými svetovými letiskami, trasy budú reprezentované jednotlivými letmi a časové okná budú zastúpené nasledovne: každý deň bude z každého z n letísk lietať x rôznych letov na ostatné letiská. Každý deň existuje z každého letiska maximálne jeden let smerujúci do iného letiska, avšak môže sa stať, že v daný deň z daného letiska do iného letiska nebude let existovať. Cestujúci musí každý deň absolvovať práve jeden let.



Obr. 1: Príklad riešenia TSP-TW demonštrovaného na mape Spojených štátov amerických. Námet na obrázok prevzatý z: [38].

1.2 Zhrnutie obsahu jednotlivých kapitol práce

V úvode sme teda nastolili a definovali TSP a TSP-TW. V druhej kapitole sa budeme venovať podrobnejšiemu popisu TSP-TW, ktorým sa primárne táto práca zaoberá. Stručne si predstavíme aj ďalšie podtypy problému TSP, ktoré sa podtypu TSP-TW podobajú. V tretej kapitole sa budeme venovať jednotlivým druhom metód riešenia, ktoré je možné aplikovať na TSP a TSP-TW. Budú tu rozobrané z obecného hľadiska, a teda ako nimi môžeme riešiť hlavne klasický TSP. V štvrtej kapitole si podrobnejšie rozoberieme stratégie, ktorými sme priamo riešili TSP-TW. Ku každej stratégii si popíšeme aj to, aké modifikácie museli byť vykonané, aby nimi bolo možné riešiť TSP-TW. V piatej kapitole si podrobnejšie popíšeme implementáciu vybraných stratégií na reálnych dátach. Popíšeme si tiež aplikáciu navrhnutú v programovacom jazyku Python, ktorá ako vstupné dáta využíva údaje o reálnych letoch a pozíciach reálnych letísk, na ktoré aplikuje vybrané stratégie. Výstupom budú jednotlivé riešenia. V šiestej kapitole si predstavíme výsledky testov jednotlivých implementovaných stratégií a vyhodnotíme, ktoré z nich boli najúspešnejšie z rôznych hľadísk. V závere si zhrnieme výsledky našej práce.

2 TSP-TW a podtypy TSP jemu podobné

V tejto kapitole sa budeme venovať podrobnejšie TSP-TW, ale predstavíme si aj niektoré podproblémy TSP, ktoré sú TSP-TW podobné a tiež je na ne možné aplikovať časové okná.

2.1 Problém obchodného cestujúceho s časovými oknami

Problém obchodného cestujúceho s časovými oknami pozostáva z nájdenia cesty s minimálnou cenou, pričom počiatočný a koncový uzol je ten istý a cesta musí byť Hamiltonovskou kružnicou, teda cesta musí obsahovať každý uzol práve raz. Každý z týchto uzlov má byť navyše navštívený v dopredu danom časovom okne. Toto časové okno zväčša pozostáva zo spodnej a hornej časovej hranice, v ktorých rozmedzí má byť daný uzol navštívený. Obchodnému cestujúcemu teda nie je povolené navštíviť daný uzol po uplynutí hornej časovej hranice pre tento uzol. Naopak, ak obchodný cestujúci príde do uzlu pred spodnou časovou hranicou pre daný uzol, musí v ňom vyčkať až do spodnej časovej hranice. Tento vyčkávací čas sa pričíta k celkovej cene trasy. Ide teda o časovo obmedzený problém a po grafe sa pohybuje práve jeden obchodný cestujúci.

Tento typ problému je obsiahnutý v širokej aplikačnej škále od priemyslu až po služby. Využitie nachádza napríklad v bankovom sektore, poštových dodávkach, v plánovaní cesty pre školský autobus, plánovaní cesty lietadiel a pod. TSP-TW tak predstavuje dôležitý element vysoko komplexných problémov s plánovaním cesty.

Formálne môžeme tento problém definovať nasledovne: predpokladajme, že máme množinu N uzlov ktoré musia byť navštívené, $N = \{1, 2, ..., n\}$. Ďalej máme duplikovaný počiatočný uzol o, ktorý je zároveň aj koncovým uzlom d. Platí $V = N \cup \{o, d\}$. Ďalej asociujeme každému uzlu $i \in V$ časové okno $[a_i, b_i]$. Konštanta a_o označuje najskorší možný čas, kedy môže obchodný cestujúci vyraziť z počiatočného uzlu. Množinu hrán označíme ako A a každej hrane v súbore priradíme nezáporné časové trvanie t_{ij} a jej cenu c_{ij}. Ďalej predpokladáme, že prípadná doba obsluhy zákazníka (alebo prestoja na letisku, či doba státia autobusu na určitej zastávke a pod.) v uzle i je už započítaná v premennej t_{ij}, pre každé $i \in N \cup \{o\}$. Nakoniec, hrana (i, j) je definovaná v množine A, ak platia následovné podmienky:

$$a_i + t_{ij} \leq b_j, \quad i \in N \cup \{o\}, \quad j \in N \cup \{d\}, \quad i \neq j. \tag{1}$$

Formulácia pomocou matematického programovania uvedená nižšie, obsahuje dva typy premenných: množinu binárnych tokov premenných $X = (X_{ij}, (i, j) \in A)$ a množinu časových premenných $T = (T_i, i \in V)$. Premenná X_{ij} je rovná jednej, ak je hrana (i, j) použitá v optimálnej trase, v opačnom prípade je jej hodnota nulová. Vzhľadom na to, že obchodnému cestujúcemu je povolené prísť do uzlu pred tým, než čas dosiahol spodnú hranicu časového okna pre daný uzol, v ktorom ale v takomto prípade musí obchodný cestujúci počkať až dokým neuplynie čas do spodnej hranice časového okna, premenná T_i nám špecifikuje začiatok prípadnej obsluhy zákazníka (alebo prestoj auta, lietadla a pod.) v uzle i, kde $i \in N \cup \{o\}$, zatiaľ čo premenná T_d nám určuje čas príchodu do počiatočného, respektíve koncového uzlu.

Keďže cesta s minimálnou cenou začína v rozpätí časového intervalu $[a_o, b_o]$ v počiatočnom uzle, navštívenie každého uzla z množiny N práve raz v rozpätí ich časových okien a ukončenie cesty v počiatočnom – koncovom uzle pred uplynutím časového limitu b_d môže byť formulované nasledovne:

$$\text{Minimalizovanie} \sum_{(i,j)\in A} c_{ij} X_{ij} \tag{2}$$

v závislosti od:

$$\sum_{j\in N\cup\{d\}} X_{ij} = 1, \quad \forall i \in N \tag{3}$$

$$\sum_{j\in N} X_{o,j} = 1, \tag{4}$$

$$\sum_{i\in N\cup\{o\}} X_{ij} - \sum_{i\in N\cup\{d\}} X_{ij} = 0, \quad \forall j \in N \tag{5}$$

$$\sum_{i\in N} X_{i,d} = 1, \tag{6}$$

$$X_{ij}(T_i + t_{ij} - T_j) \leq 0, \quad \forall (i, j) \in A \tag{7}$$

$$a_i \leq T_i \leq b_i, \quad \forall i \in V \tag{8}$$

$$X_{ij} \geq 0, \quad \forall (i, j) \in A \tag{9}$$

$$X_{ij} \text{ binárne}, \quad \forall (i, j) \in A \tag{10}$$

Táto formulácia dôverne replikuje stanovený fixný harmonogram, o ktorom sa môžeme dočítať viac v druhej kapitole práce [3]. Ide o nelineárny program s počtom premenných $O((n+2)^2)$ a s počtom obmedzení $O((n+2)^2)$, kde objektívna

funkcia (2) reprezentuje celkovú cenu trasy. Podproblém definovaný obmedzeniami (2) až (5) a (9) je problémom s minimálnymi nákladmi (v skutočnosti ide o problém priradenia s (n+2) riadkami a (n+2) stĺpcami). Obmedzenia (7) a (8) zabezpečujú uskutočniteľnosť časového harmonogramu.

Subcesty sú eliminované obmedzeniami (7). Ak použijeme veľkú hodnotu konštanty M, tieto obmedzenia môžu byť linearizované a prepísané nasledovne:

$$T_i + t_{ij} - T_j \leq M(1 - X_{ij}), \quad \forall (i,j) \in A. \tag{11}$$

Všimnime si, že veľká hodnota konštanty M môže byť nahradená ako $M_{ij} = \max\{b_i + t_{ij} - a_j, 0\}$, $(i, j) \in A$. Ak $b_i + t_{ij} \leq a_j$, tieto obmedzenia sú vždy splnené pre všetky hodnoty T_i, T_j a X_{ij}. Potom formulácia vyžaduje existenciu obmedzení (7) alebo (11) iba pre hrany $(i, j) \in A$, také že $M_{ij} > 0$.

Vlastnosť nelineárnej integrity môžeme popísať nasledovne. Nelineárna formulácia (2) až (10) má zaujímavú vlastnosť. Ak je problém realizovateľný, požiadavky (10) môžu byť eliminované z vyššie spomenutej formulácie. Aby sme si ukázali, že sa tak môže stať, predpokladajme že (X^*, T^*) je optimálne riešenie. Ak X^* nie je celé číslo, definujeme $A^* = \{(i, j) \in A \mid X^*_{ij} > 0\}$. Predpokladajme ďalej, že problém minimálnej ceny trasy na subsieti získame iba pomocou hrán z A^* keď existuje celočíselné optimálne riešenie $\overline{X}$. Je potom jednoduché overiť že ($\overline{X}$, T^*) je optimálne celočíselné riešenie pre (2) až (9). Uvedomme si však, že linearizovaný predpis (11) nemusí nutne prinášať celočíselné riešenie. Preto musia byť obmedzenia (10) dodržané v linearizovanej formulácii.

Výskum ohľadom TSP-TW bol však nedostatočný. Bolo dokázané, že aj keď nájdeme uskutočniteľné riešenie pre TSP-TW, stále to bude NP-úplny problém. Na základe toho boli navrhnuté heuristické algoritmy založené na koncepcii k-výmeny. Toto zahŕňa výmenu k hrán z momentálneho riešenia za k iných hrán. Pre TSP, proces jednej k-výmeny môže byť vykonaný v konštantom čase pre ľubovolné k. Avšak pre TSP-TW, k-výmeny budú vyžadovať $O(n)$ času. Takáto k-výmena by totiž mohla posunúť štart z určitého uzlu, čo by v konečnom dôsledku posunulo štart zo všetkých následujúcich uzlov na trase. Preto je pre každú takúto výmenu nutné otestovať nie len jej prispenie k zníženiu ceny cesty, ale aj jej uskutočniteľnosť. Bolo dokázané, že pre dvoj-výmeny a obmedzené troj-výmeny môže byť proces jednej výmeny stále vykonaný v konštantom čase s cieľom minimalizovať celkový čas cesty.

Aj napriek vysokej náročnosti TSP-TW sa niekoľko ďalších autorov sústredilo na vytvorenie exaktných algoritmov, ktoré majú minimalizovať celkovú cenu cesty alebo celkovú dobu jej trvania. Cieľ (2) nám opisuje prvý prípad; minimalizovanie hodnoty $T_d - T_o$ zodpovedá druhému prípadu. Pre tento cieľ je tiež jednoduché definovať premenné opisujúce vyčkávací čas $W_i \geq 0$, $i \in N \cup \{d\}$ pre každý uzol, ktorý má byť navštívený a pre počiatočný-konečný uzol. Ich vplyvom na obmedzenia

(7) alebo (11) a zámenou znamienka nerovnosti za znamienko rovnosti získavame výraz:

$$\text{Minimalizovanie } T_d - T_o = \sum_{(i,j)\in A} t_{ij} X_{ij} + \sum_{i \in N \cup \{d\}} W_i. \tag{12}$$

Formulácie (2) až (10) alebo model s objektívnou funkciou (2) nahradenou (12) môže byť použitý v schéme vetiev a medzí pre determinovanie optimálnej cesty obmedzenej časovými oknami. Napríklad je možné uvoľniť obmedzenia (7) a (8) a použiť spodnú hranicu poskytnutú riešením priradzovacieho problému. Vetvenie na premenných X_{ij} môže poskytnúť veľmi dobrý algoritmus. Ak berieme do úvahy linearizovanú verziu, ďalšia spodná hranica je daná riešením lineárneho rozvolňovania. Táto hranica je minimálne tak dobrá ako hranica získaná priradením, ale výpočet bude časovo oveľa náročnejší.

Pomocou predchádzajúcich formulácií sa dá navrhnúť mnoho ďalších rozvolňovaní a rozkladov. V ľubovoľnom prehľadávacom strome je vetvenie na tokových premenných X_{ij} $(i, j) \in A$ ľahko implementovateľné. Avšak ak najdôležitejší komponent špecifickej aplikácie TSP-TW obsahuje obmedzenia časovými oknami, vetvenie na časových premenných T_i je jednoznačne oveľa vhodnejšie. Toto platí aj pre všetky časovo obmedzené problémy s plánovaním trasy riešené vetvením a obmedzovaním [3].

2.2 Časovo závislý problém obchodného cestujúceho

Časovo závislý problém obchodného cestujúceho (ďalej len TD-TSP – z anglického Time dependent travelling salesman problem) je verziou klasického TSP, v ktorom predpokladáme, že cena cesty medzi dvoma uzlami je závislá od toho, kedy je uzol navštívený. Znamená to, že cena cesty z uzlu i do uzlu j sa líši v závislosti na tom, v ktorom časovom období sa z daného uzlu i do uzlu j snažíme dostať. Problém môže byť formulovaný nasledovne: predpokladajme že máme určitý graf $G(V,E)$, ktorého uzly sú situované ako $V = 1, 2, ..., n$. Pre každú hranu $(i, j) \in E$, cena c_{ij}^t za prejdenie cez danú hranu v čase t je známa, kde $t = 1, 2, ..., n$. Riešenie TD-TSP teda pozostáva z nájdenia Hamiltonovskej kružnice v grafe G [2].

$$\xi = (i_1 = 1,\ i_2,\ ...,\ i_n,\ i_{n+1} = i_1 = 1), \tag{13}$$

kde cena cesty ξ je

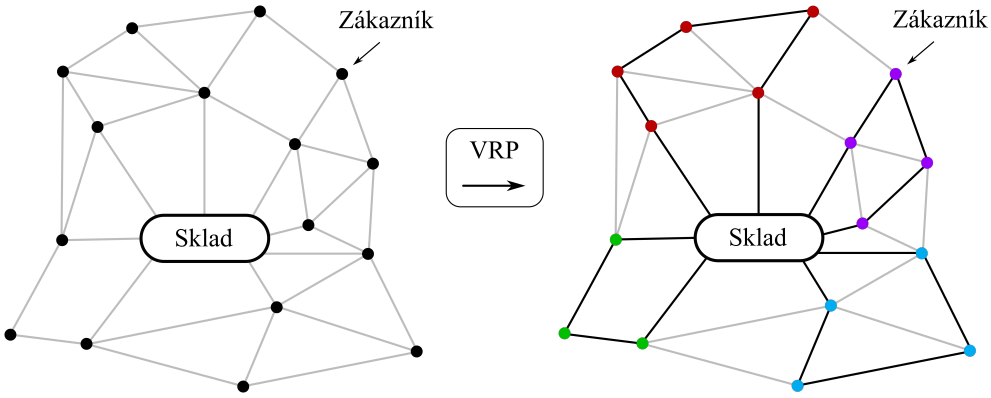
$$c_\xi = c^1_{i_1 i_2} + c^2_{i_2 i_3} + ... + c^n_{i_n i_1}. \tag{14}$$

2.3 Úloha s viacerými obchodnými cestujúcimi

Ďalším typom TSP je úloha s viacerými cestujúcimi (ďalej iba m-TSP – z anglického Multiple travelling salesman problem). Uvažujeme v nej m obchodných cestujúcich (alebo vozidiel, ktoré zabezpečujú rozvoz tovaru). Všetci títo cestujúci majú jedno depo. Predpokladáme, že každý cestujúci musí opustiť depo (kde napríklad naberá tovar k rozvozu) a každý zákazník alebo miesto musí byť navštívené práve jedným cestujúcim. Cieľom je minimalizovať súčet cien ciest jednotlivých cestujúcich. Obchodným cestujúcim sa zväčša nezadávajú žiadne podmienky súvisiace s kapacitou, ktorú môžu prepravovať. Tento problém môže nájsť aplikačné využitie v spoločnostiach, ktoré sa zaoberajú rozvozom tovaru alebo poskytovaním servisu [4].

2.4 Rozvozný problém

Rozvozný problém (ďalej iba VRP – z anglického Vehicle routing problem) vzniká, ak je k obsluhe miest (alebo zákazníkov) k dispozícii viacero obchodných cestujúcich. VRP je obdobou m-TSP, kde každému zákazníkovi je dopredu priradený dopyt po konkrétnom tovare (alebo službe). Rozvážajúce vozidlá v tomto type problému – na rozdiel od m-TSP – zväčša majú určenú obmedzenú kapacitu, ktorá sa môže líšiť v závislosti na jednotlivých obchodných cestujúcich. Obmedzený môže byť aj počet zákazníkov, ktorých môže jeden obchodný cestujúci obslúžiť alebo môže byť obmedzená maximálna trasa, ktorú môže každý obchodný cestujúci prejsť. Súčet dopytu na trase pre konkrétneho obchodného cestujúceho nesmie prekročiť danú kapacitu konkrétneho obchodného cestujúceho. Okrem vzdialenosti sa môžu v tomto type problému optimalizovať viaceré prvky. Môže to byť tiež napríklad počet obchodných cestujúcich [5].



Obr. 2: Príklad počiatočného zadania VRP (naľavo) a jeho následného riešenia (napravo). Vidíme, že vo výsledku sú farebne odlíšené uzly. Jednou farbou sú vždy označené uzly, ktorými prejdú jednotliví obchodní cestujúci. Tieto uzly spájajú tmavo vyznačené hrany, ktoré bude nutné pre každého daného obchodného cestujúceho prejsť, aby bolo splnené zadanie. Námet na obrázok prevzatý z: [39].

3 Prehľad stratégií riešiacich TSP a TSP-TW

V tejto kapitole si predstavíme možné stratégie použiteľné na získanie riešenia TSP a TSP-TW. Popíšeme si metódy informovaného prehľadávania, ktoré sa budú deliť na jednoduché heuristiky a metaheuristiky. Druhé menované sa ešte ďalej rozdelia na genetické algoritmy a metódy rojovej inteligencie.

3.1 Informované metódy prehľadávania

Informované metódy prehľadávania sú založené na hodnotiacej funkcii. Hodnotiaca funkcia nám pre každý uzol stromu určuje jeho ohodnotenie. Tieto hodnoty sa následne použijú ako kritérium pre výber ďalšieho uzlu, ktorý bude expandovaný. Ak hodnotiaca funkcia dobre vystihuje vlastnoti a charakter úlohy, budú vždy expandované najperspektívnejšie uzly a zabráni sa takprehľadávaniu ciset, ktoré nevedú k cielu. Čím kvalitnejšie heuristické znalosti o danej úlohe sa v hodnotiacej fukncii využívajú, tým efektívnejšie bude prehladávanie.

3.2 Jednoduchšie heuristiky

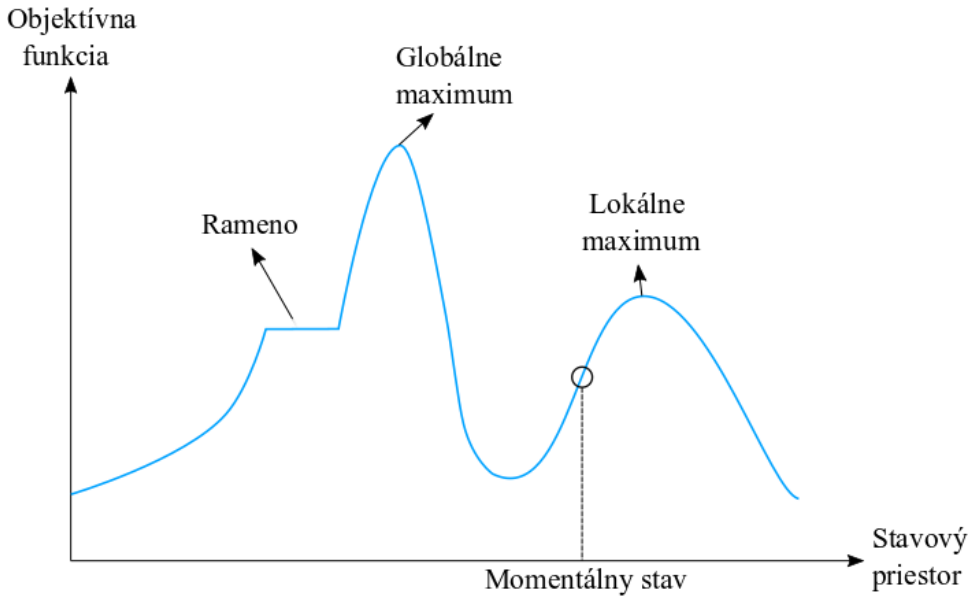
Obecne všetky heuristiky spadajú pod informované metódy prehľadávania, čo znamená, že sú založené na hodnotiacej funkcii. Hodnotiaca funkcia slúži na určenie kvality riešenia. Heuristika (z gréckeho slova heuriskó – nájsť, objaviť) nám charakterizuje skúšobné riešenie problému, pre ktorý nepoznáme exaktný algoritmus alebo exaktnú metódu. Takéto riešenie je často iba približné, založené na učení sa z predchádzajúcich výsledkov (metóda pokus – omyl). Po prvom riešení môže často dôjsť k lepšiemu výsledku pri ďalších riešeniach, ale heuristické metódy nám nikdy nemôžu zaručiť, že nájdené riešenie je optimálne. Výhoda tejto metódy je však jej univerzálnosť, jednoduchosť a rýchlosť.

V tejto podkapitole sa budeme venovať jednoduchším heuristikám, ktoré zväčša používajú iba hodnotiacu funkciu, narozdiel od metaheuristík (popisovaných v ďalšej podkapitole), ktoré okrem hodnotiacej funkcie používajú aj iné mechanizmy slúžiace k nájdeniu lepšieho riešenia.

3.2.1 Hill-Climb algoritmus

Hill-Climb algoritmus patrí medzi gradientné algoritmy, čo znamená, že každá ďalšia voľba uzlu na expandovanie nejakým spôsobom závisí na gradiente. Funkciu Hill-Climb algoritmu môžeme popísať nasledovne: na začiatku sa nachádzame v počiatočnej pozícii, teda v počiatočnom uzle. Algoritmus náhodne vyberie niektorého zo susedov expandovaného uzlu. Vypočíta hodnotu hodnotiacej funkcie tohto rozhodnutia a ďalej počíta hodnotiace funkcie pre ostatné susedné uzly. Nakoniec vyberie práve ten uzol, ktorý má najlepšiu hodnotu hodnotiacej funkcie. Vybraný uzol následne expanduje. Ďalej algoritmus skontroluje, či je práve expandovaný uzol zároveň aj cieľovým uzlom. Ak áno, algoritmus sa ukončí a vráti riešenie. Ak nie, algoritmus pokračuje v prvom kroku až dovtedy, dokým sa nedostane ku koncovému uzlu. Algoritmus zároveň prechádza vždy do uzlov, ktoré ešte neboli expandované. Pracuje sa vždy iba s práve expandovaným uzlom. Susedia predchádzajúceho uzlu sú vždy pri expanzii nového uzlu zabudnutí.

Nevýhodou tohto algoritmu je možnosť zaseknutia sa v lokálnom extréme. Algoritmus nemá ako zistiť, či ním nájdené riešenie je iba lokálnym extrémom alebo globálnym. Tiež môže dôjsť k zacykleniu – pohyb po nekonečne dlhej ceste (napríklad pri konštantnom ohodnotení hrán v grafe) [6].

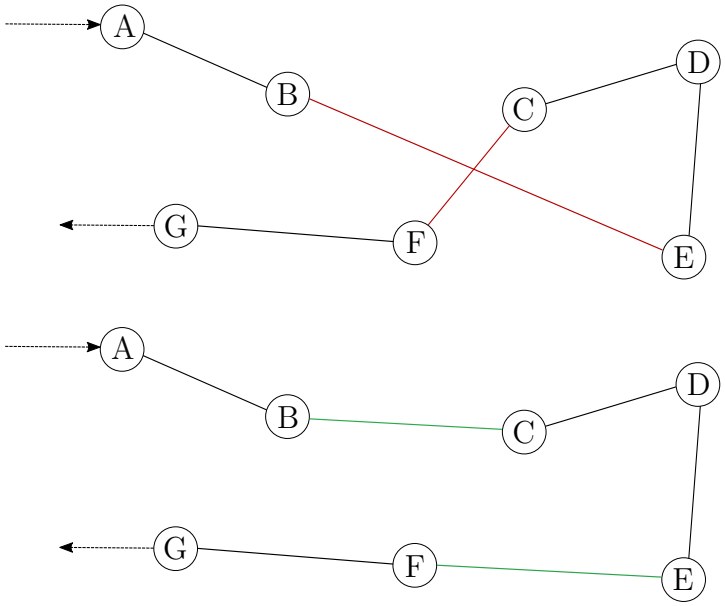


Obr. 3: Príklad chodu algoritmu Hill-Climb. Algoritmus si vždy podľa gradientu vyberie najstrmší spád (vertikálne smerom hore, ak hľadáme globálne maximum a vertikálne smerom dolu, ak hľadáme lokálne minimum), po ktorom sa vyberie. Námet na obrázok prevzatý z: [41].

3.2.2 2-opt algoritmus

Algoritmus 2-opt je jednoduchý lokálny prehľadávač, ktorý má pre prípad TSP veľmi podstatné využitie a môže citeľne zlepšiť vyhľadávanie riešení tohto problému. Jeho hlavnou úlohou je nájsť cestu, ktorá sa kríži a prehodiť poradie uzlov tak, aby sa ďalej nekrížila.

Beh tohto algoritmu vyzerá nasledovne: algoritmus 2-opt zoberie riešenie získané nejakým iným vyhľadávacím algoritmom. Jednu trasu (riešenie) rozdelí na tri subtrasy. Prvá a tretia subtrasa sa nemenia, zatiaľ čo v druhej subtrase algoritmus náhodne prehodí poradie uzlov. Tieto tri subtrasy potom spojí a otestuje, či má takto nová vytvorená trasa lepšiu cenu riešenia, ako bola cena predošlého riešenia. Pokiaľ áno, algoritmus sa ukončí, pokiaľ nie, algoritmus pokračuje v slučke až dokým nenájde také riešenie, ktoré bude mať lepšiu cenu riešenia ako bola pôvodná cena [9].



Obr. 4: Príklad využitia 2-opt algoritmu. Pôvodná trasa A→B→E→D→C→F→G sa zmenila na trasu s poradím uzlov A→B→C→D→E→F→G, ktorá má lepšie ohodnotenie ako trasa pôvodná. Námet na obrázok prevzatý z: [42].

3.2.3 A* algoritmus

Hodnotiaca funkcia algoritmu A^* má nasledujúcu podobu:

$$f(i) = g(i) + h(i). \tag{15}$$

Krok, v ktorom sa pre každého následovníka počíta hodnota hodnotiacej funkcie, aby sa následne mohol vybrať najlepší kandidátny uzol na expandovanie, má nasledujúcu podobu: expanduje sa stav i; pre každého následovníka j stavu i sa vypočíta hodnota hodnotiacej funckie ako:

$$f(j) = g(j) + h(j), \tag{16}$$

pričom:

$$g(j) = g(i) + c(i, j), \tag{17}$$

kde $c(i,\ j)$ je cena prechodu od stavu i do stavu j [15]. Pre prípad TSP bude v algoritme nutné vykonať isté modifikácie. Po týchto modifikáciach bude chod algoritmu pre riešenie TSP vyzerať nasledovne [36]:

1. Vloženie koreňového uzlu do zoznamu navštívených uzlov.
2. Ak je zoznam navštívených uzlov prázdny, algoritmus sa ukončí.
3. Vyberie prvý uzol v zozname navštívených uzlov, označíme ho ako uzol i.
4. Ak je uzol i koncovým uzlom, algoritmus vráti výsledok a ukončí sa.
5. Vloží všetky dcérske uzly uzlu i do zoznamu navštívených uzlov. Všetky položky v zozname navštívených uzlov zoradí vo vzostupnom poradí podľa hodnoty hodnotiacej funkcie.
6. Vráti sa ku kroku číslo 2.

3.3 Metaheuristiky

Metaheuristiky sú zložitejšie heuristické metódy, ktoré okrem hodnotiacej funkcie využívajú aj ďalšie mechanizmy slúžiace k nájdeniu lepšieho riešenia. Tieto mechanizmy slúžia zväčša na to, aby algoritmom nájdený lokálny extrém nebol považovaný za extrém globálny a neukončilo sa tak vyhľadávanie predčasne. Vyznačujú sa väčšou zložitosťou ako jednoduchšie heuristiky a často sú aj časovo náročnejšie na výpočet.

3.3.1 Simulated annealing

Algoritmus Simulated annealing (ďalej iba SA) je inšpirovaný reálnym procesom ohrevu materiálu na určitú teplotu, pri ktorej je materiál (kov) ľahko tvarovateľný. Postupným ochladzovaním vplyvom okolitého prostredia sa jeho tvarovateľnosť znižuje. Takto nahriaty materiál sa tvaruje tým ľahšie, čím je teplota vyššia. Postupne

začne teplota klesať vplyvom chladnejšieho prostredia okolia (buď vplyvom izbovej teploty, vody alebo oleja). Čím bude teplota materiálu nižšia, tým ťažšie sa materiál bude tvarovať, až sa dostane na takú teplotu, že stuhne úplne a už ho nebude možné tvarovať vôbec.

Tab. 1: Vzťah medzi termodynamickou simuláciou a optimalizáciou [14].

Termodynamická simulácia	**Optimalizácia**
stav systému	prípustné riešenie
energia	hodnota účelovej funkcie
zmena stavu	prechod k susednému riešeniu
teplota	riadiaci parameter
zmrazený stav	heuristické riešenie

Algoritmus SA je prezentovaný ako algoritmus pre kombinatorické optimalizačné problémy. Je často volenou metaheuristikou pre riešenie diskrétnych a spojitých optimalizačných úloh. Kľúčová vlastnosť tohto algoritmu je schopnosť uniknúť z lokálneho extrému tým, že povoluje Hill-Climbovské pohyby, aby sa našiel globálny extrém.

Jednou z nevýhod tohto algoritmu je to, že má pomerne veľa atribútov, ktoré môžeme upravovať a optimálne hodnoty týchto atribútov sa môžu líšiť pre rôzne typy problémov. Existujú dva druhy spôsobov úpravy týchto atribútov: online a offline prístup. Pri online prístupe sa hodnoty parametrov upravujú dynamicky počas chodu algoritmu. Pri offline prístupe sa hodnoty atribútov zvolia ako konštanty ešte pred spustením algoritmu a po celý čas sa nemenia. Docieliť optimálne naladenie parametrov nie je jednoduché a metóda pokus-omyl je väčšinou nepostačujúca. Tieto atribúty sú konkrétne: teplota, mraziaca teplota a koeficient α. Hodnota teploty sa nastaví na konštantnú veľkosť a po každej iterácii sa bude násobiť koeficientom α pričom $\alpha < 1$. Mraziaca teplota má funkciu spodnej hranice. Ak hodnota teploty klesne pod hodnotu mraziacej teploty, algoritmus sa ukončí.

Chod algoritmu môžeme popísať nasledovne: na začiatku sa vygeneruje jedno riešenie (môže byť napríklad náhodné alebo heuristické pomocou *greedy search* a pod.). Ďalej sa budú generovať náhodné riešenia zo susedstva aktuálneho riešenia, ktoré buď algoritmus SA prijme alebo nie. Ak je nájdené riešenie lepšie ako aktuálne riešenie, je automaticky prijaté ako nové riešenie. Ak je nové nájdené riešenie

horšie ako aktuálne riešenie, tak to, či ho algoritmus prijme záleží na premennej p – pravdepodobnosti prijatia nového riešenia. Pre túto pravdepodobnosť platí:

$$p = e^{\frac{-(f(y)-f(x))}{t}}, \tag{18}$$

kde $f(y)$ je hodnota hodnotiacej funkcie nového riešenia y, $f(x)$ je hodnota hodnotiacej funkcie aktuálneho riešenia x a t je aktuálna teplota. Celkovo teda pre pravdepodobnosť prijatia nového riešenia p platí:

$$p = \begin{cases} 1, & \text{ak} f(y) \leq f(x), \\ e^{\frac{-(f(y)-f(x))}{t}}, & \text{inak.} \end{cases} \tag{19}$$

Tento cyklus sa opakuje, pokiaľ hodnota teploty neklesne pod hodnotu mraziacej teploty. V takom prípade sa algoritmus ukončí a vráti najlepšie nájdené riešenie. Všimnime si, že použitie tohto algoritmu nám nedáva úplnú istotu, že bude nájdené najlepšie riešenie. Akceptovaním aj horších riešení (v závislosti od momentálnej teploty a od rozdielu od aktuálneho riešenia) sa zvyšuje pravdepodobnosť toho, že algoritmom nájdený lokálny extrém nebude považovaný za extrém globálny a neukončí sa tak vyhľadávanie predčasne [10]. Jedným z možných použití algoritmu SA na TSP-TW je jeho modifikácia Stlačené žíhanie. Tento druh žíhania je doplnený o multiplikátor variabilného trestu ktorý sa nazýva tlak a dopĺňa parameter teploty. V súvislosti s TSP-TW teplota riadi pravdepodobnosť prechodu na horšiu trasu, zatiaľ čo tlak riadi pravdepodobnosť prechodu na nerealizovateľnú cestu s ohľadom na časové okná. Viac o využití tohto druhu algoritmu pre TSP-TW je možné dočítať sa v práci: [11].

3.3.2 Tabu search

Algoritmus Tabu Search (ďalej len TS) je metaheuristická lokálna vyhľadávacia metóda používaná pre matematickú optimalizáciu. Tak ako pri SA, jeho hlavnou výhodou oproti ostatným lokálnym vyhľadávacím algoritmom je veľká pravdepodobnosť, že neuviazne v lokálnom extréme. TS teda kombinuje lokálne hľadanie založené na množine elementárnych pohybov a heuristiky, ktoré majú zabrániť výskytu cyklov a uviaznutiu postupu v suboptimálnom riešení [12]. Základom TS je využívanie foriem flexibilnej pamäte. Táto pamäť môže byť dvojakého druhu [14]:

- *krátkodobá pamäť* – atribúty pohybov, ktoré sú po určitý počet iterácií zakázané; sú to atribúty pohybov inverzných k vykonaným pohybom,
- *dlhodobá pamäť* – informácie o frekvencii výskytu atribútov v doterajšom procese hľadania; využitie v procesoch intenzifikácie a diverzifikácie.

Chod algoritmu môžeme popísať nasledovne: na začiatku sa vygeneruje náhodné riešenie (môže sa ich vygenerovať aj viac a z nich sa na začiatok vyberie to najlepšie) a zoznam so zakázanými riešeniami je prázdny. Ďalej sa vytvorí slučka, v ktorej sa budú kroky opakovať, až pokiaľ nie je splnená podmienka ukončenia (vyčerpanie vopred stanoveného času alebo počtu iterácií, zistenie, že po stanovenom počte iterácií algoritmus nedospel k lepšiemu riešeniu – predpoklad nájdenia globálneho extrému). K momentálnemu riešeniu sa vytvorí zoznam kandidátnych susedných riešení. Z tohto zoznamu sa vyberie najlepšie riešenie a vykoná sa kontrola, či sa nachádza v zozname zakázaných riešení. Ak áno, riešenie je vymazané a z kandidátneho zoznamu sa vyberie ďalšie najlepšie riešenie. Ak sa riešenie v zozname zakázaných riešení nenachádza, vykoná sa porovnanie, či je nové riešenie lepšie ako doposiaľ najlepšie nájdené riešenie. Viac o implementácii algoritmu TS pre VRP-TW (z anglického Vehicle routing problem with time windows – Rozvozný problém s časovými oknami), veľmi podobnému TSP-TW, je možné dočítať sa v práci: [13].

3.3.3 Genetické algoritmy

Genetické algoritmy (ďalej iba GA) sú adaptívne heuristické prehľadávacie metódy inšpirované evolučnými mechanizmami prirodzeného výberu a genetiky. Proces v GA spočíva v simulovaní procesov prírodných systémov potrebných pre evolúciu, konkrétne systémov založených na princípoch prežitia najsilnejších, prvý krát predstavených Charlesom Darwinom. Tieto systémy predstavujú inteligentné využitie náhodného prehľadávania v rámci definovaného priestoru na vyriešenie problému. Aj napriek intenzívnym štúdiám matematikov, počítačových vedcov a ďalších výskumných pracovníkov v posledných 50 rokoch zostáva otázka možností zostrojenia efektívneho riešenia založeného na GA nezodpovedaná [32].

Realizovateľné riešenia TSP môžu predstavovať chromozómy pozostávajúce z génov. Každý gén v chromozóme predstavuje nejaké miesto. Hodnoty týchto génov a ich pozícia v *reťazci génu* hovorí GA, aké riešenie toto indivíduum reprezentuje. Ďalej môžeme definovať matematický model. Predpokladajme, že $C = \{1, 2, ..., i, .., n\}$ sú miesta, ktoré musia byť navštívené. Každý chromozóm môže byť potom reprezentovaný nasledovne [32]:

$$X = (X_1, X_2, ..., X_i, ..., X_n), \tag{20}$$

pre TSP, tento chromozóm reprezentuje trasu:

$$X_1 \rightarrow ... \rightarrow X_i ... \rightarrow X_n \rightarrow X_1. \tag{21}$$

Aby bola zaistená validita chromozómu, musia byť splnené nasledovné podmienky:

$$X_i \in C, 1 \leq i \leq n \tag{22}$$

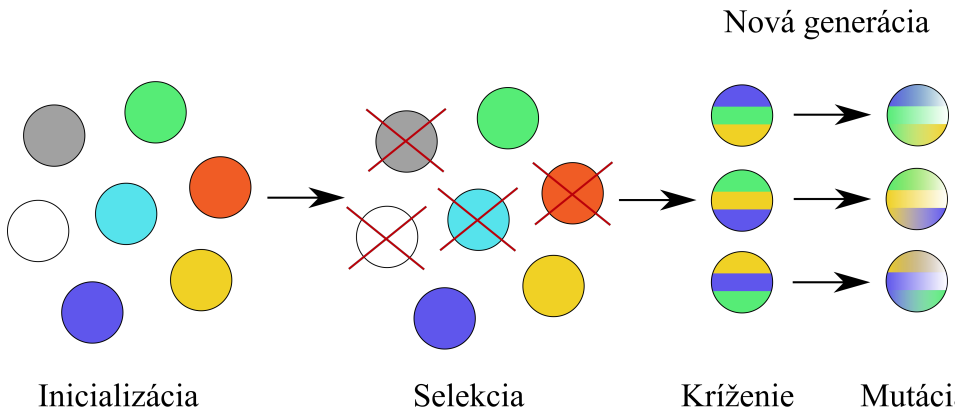
$$X_i \neq X_j, i \neq j, 1 \leq i, j \leq n \tag{23}$$

Fitness funkcia v tomto algoritme ohodnocuje kvalitu každého indivídua. Keďže cieľom TSP je nájsť najkratšiu trasu, fitness funkcia $f(X)$ každého choromozómu X môže byť definovaná nasledovne:

$$f(X) = \sum_{i=1}^{n-1} D(X_i, X_{i+1}) + D(X_n, X_1), \tag{24}$$

kde:

- n – celkový počet miest,
- X_i – číslo miesta v pozícii i,
- $D(X_i, X_j)$ – vzdialenosť z miesta X_i do X_j,
- $D(X_i, X_j) = D(X_j, X_i)$.



Obr. 5: Príklad procesu genetického algoritmu. Námet na obrázok prevzatý z: [43].

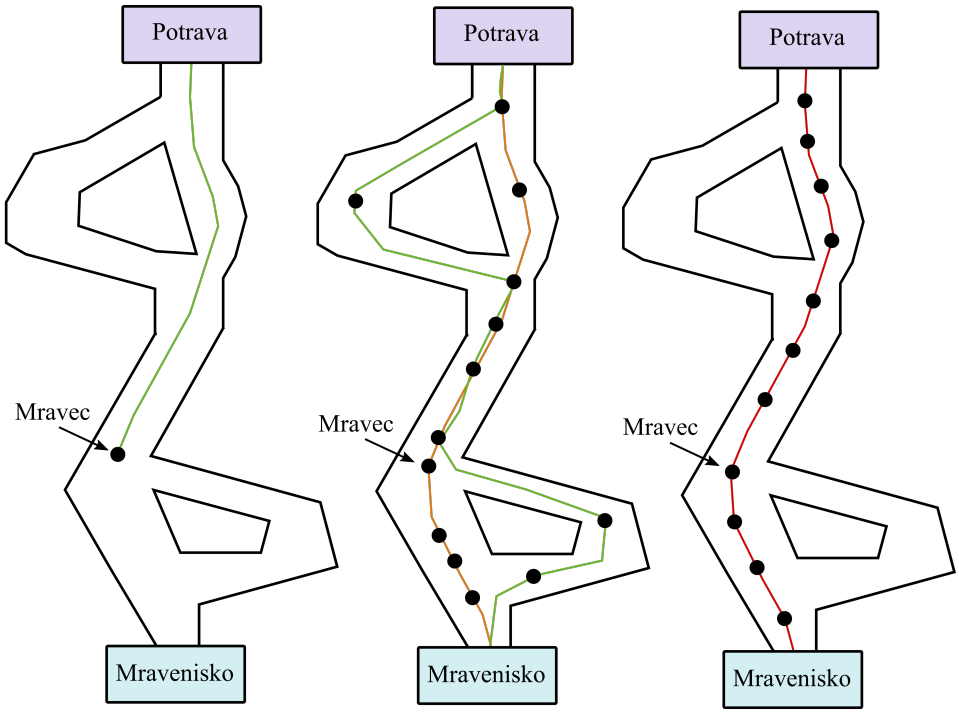
Viac o implementácii GA pre TSP-TW je možné dočítať sa v práci: [34].

3.3.4 Metódy rojovej inteligencie

Metódy rojovej inteligencie patria tiež pod metaheuristiky, konkrétnejšie pod metódy inšpirované živou prírodou. Sú inšpirované chovaním roju určitého druhu živočíchov (napríklad mravcov, včiel, mačiek, vlkov a pod.). Tieto spôsoby chovania sú prenesené do optimalizácie s cieľom využiť ich pri hľadaní optimálneho riešenia. Tento druh metaheuristiky je pomerne novým prístupom k riešeniu optimalizačných problémov.

Ant Colony Optimization

Ant colony optimization (ďalej iba ACO) je inšpirované správaním niektorých druhov mravcov pri hľadaní potravy. Tieto mravce zanechávajú na trase stopy feromónu, aby označili najlepšiu trasu, ktorú budú ostatné mravce z kolónie nasledovať. Optimalizácia mravcov v prírode slúži ako príklad pre aplikáciu tohto prístupu k riešeniu matematickej optimalizácie. Túto stratégiu možno implementovať pre nájdenie riešenia TSP a s istými modifikáciami aj pre nájdenie riešenia TSP-TW.



Obr. 6: Proces hľadania najkratšej trasy mravcov v prírode, ktorým je algoritmus ACO inšpirovaný. Zelenou je znázornená slabá, oranžovou stredne silná a červenou najsilnejšia feromónová stopa. Námet na obrázok prevzatý z: [22].

Pre jednotky, ktoré reprezentujú mravce v matematickej optimalizácii, platí:

- majú pamäť, čo znamená že nenavštívia uzly, ktoré už predtým navštívili,
- dopredu presne vedia, aká bude vzdialenosť dvoch uzlov a teda môžu si vybrať cestu kratšiu,
- ak sa pri rozhodovaní mravca stane to, že dve a viac ciest budú mať rovnaké ohodnotenie, mravec sa zvyčajne rozhodne pre tú cestu, ktorá bude obsahovať silnejšiu stopu feromónu (tak ako to robia reálne mravce v prírode).

Vždy, keď sa mravec dostane na rozcestie (teda má na výber minimálne dve ďalšie subtrasy), každej z nich sa priradí hodnota p_{ij}^k, čo je pravdepodobnosť s akou si ju mravec vyberie ako ďalšiu subtrasu, kde i je práve expandovaný uzol, j je ďalší možný expandovaný uzol a k je označenie daného mravca. Pre pravdepodobnosť výberu danej subtrasy platí:

$$p_{ij}^k = \frac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{s \in povolene_k} \tau_{is}^{\alpha} \cdot \eta_{is}^{\beta}}, \tag{25}$$

kde:

- τ_{ij} - intenzita feromónovej stopy medzi uzlami i a j,
- α - parameter regulujúci veľkosť vplyvu τ_{ij},
- η_{ij} - viditeľnosť uzlu j z uzlu i, ktorá je rovná $\frac{1}{d_{ij}}$, kde d_{ij} je vzdialenosť medzi uzlami i a j,
- β - parameter regulujúci veľkosť vplyvu η_{ij},
- $povolene_k$ - množina uzlov, ktoré doteraz neboli navštívené mravcom k.

Pre pravedpodobnosť voľby danej subtrasy teda obecne platí:

$$p_{ij}^k = \begin{cases} \frac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{s \in povolene_k} \tau_{is}^{\alpha} \cdot \eta_{is}^{\beta}}, & j \in povolene_k \\ 0, & inak. \end{cases} \tag{26}$$

V skratke, pre uzly, ktoré ešte neboli navštívené mravcom k, sa pravdepodobnosť ich navštívenia cez hranu (i, j) vypočíta vzrocom (25). Pre uzly, ktoré už mravcom k navštívené boli, bude pravdepodobnosť automaticky nulová [17, 18].

Po tom, čo každý mravec vykoná n iterácií, je získaná kompletná trasa. Feromónové stopy na cestách sú aktualizované takým spôsobom, že kratšie cesty budú obsahovať výraznejšiu stopu feromónu ako dlhšie cesty. Tak ako v prírode, feromón zanechávaný na cestách mravcami po čase vyprchá. Čím je teda cesta kratšia a ide po nej viacero mravcov, tým je na tejto ceste stopa feromónu silnejšia, pretože ňou

mravci, ktorí po nej cestujú, chodia s väčšou frekvenciou a teda feromónová stopa sa neustále obnovuje. Predpis, ktorým sa feromón aktualizuje, je nasledovný:

$$\tau_{ij}(t+1) = \rho \ \cdot \ \tau_{ij}(t) + \Delta\tau_{ij}, \tag{27}$$

kde:

$$\Delta\tau_{ij} = \sum_{k=1}^{l} \Delta\tau_{ij}^{k}, \tag{28}$$

pričom:

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{Q}{L_k}, & \text{ak mravec } k \text{ cestuje po hrane } (i,\ j) \\ 0, & \text{inak.} \end{cases} \tag{29}$$

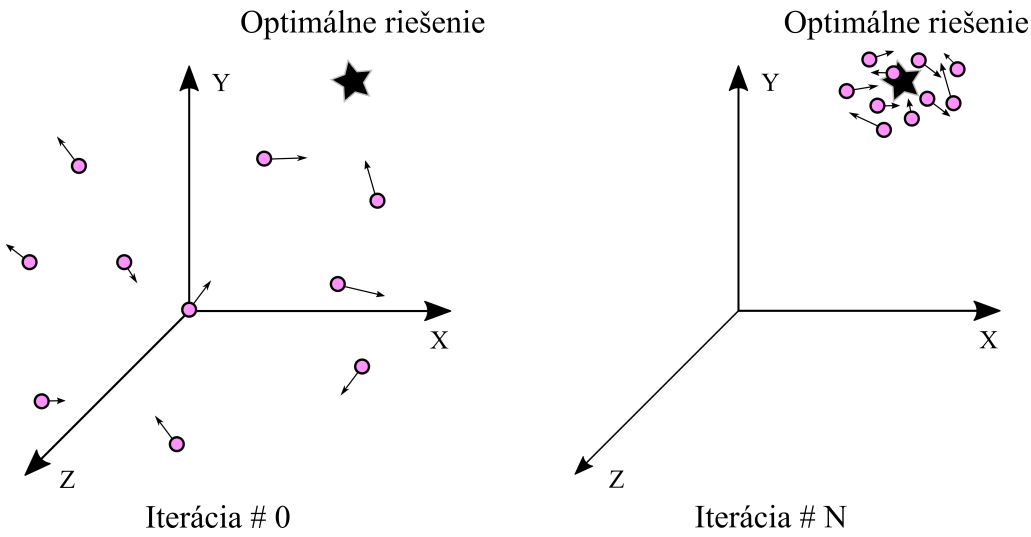
Definície premenných a konštánt sú nasledovné:

- Q – konštanta,
- L_k – dĺžka trasy,
- t – počítadlo iterácií,
- $\rho \in \ [0, 1]$ – evaporačný faktor, ktorý reguluje redukciu feromónu,
- $\Delta\tau_{ij}$ – celková zmena množstva feromónu na hrane $(i,\ j)$.

Pre potreby TSP-TW vznikla modifikácia algoritmu ACO. Viac o využití tohto druhu algoritmu ACO pre TSP-TW je možné dočítať sa v práci: [19].

Particle Swarm Optimization

Particle swarm optimization (ďalej iba PSO) je stochastická optimalizačná metóda navrhnutá Eberhartom a Kennedym v roku 1995 [20]. Vychádza zo správania ekosystému, konkrétnejšie zo sociálneho správania živočíchov žijúcich v skupinách, napríklad rýb alebo vtákov letiacich v skupinách. Pri riešení optimalizačných problémov sa táto metóda spolieha na množinu individuí, pôvodne usporiadaných náhodne, ktoré nazývame *častice*. Tieto častice sa pohybujú v prehľadávacom priestore. Každá z nich reprezentuje riešenie problému. Navyše, každá z nich má pridelenú pozíciu X_{id} a rýchlosť V_{id}. Každá častica má tiež v pamäti zapísanú svoju doposiaľ najlepšiu dosiahnutú pozíciu a doposiaľ najlepšiu dosiahnutú globálnu pozíciu všetkých častíc [21].



Obr. 7: Príklad chodu PSO algoritmu. V súradnom systéme sú ružovými kruhmi vyznačené častice a hviezdicou je znázornené optimálne riešenie, ktoré sa algoritmus snaží nájsť. Šípky znázorňujú rýchlosť daných častíc. Námet na obrázok prevzatý z: [33].

Každej častici je v každej iterácii priradená jej rýchlosť podľa nasledujúceho predpisu:

$$V_{id} = V_{id} + \alpha \cdot (P_{id} - X_{id}) + \beta \cdot (P_{gd} - X_{id}), \tag{30}$$

kde:

- X_{id} – momentálna pozícia častice,
- P_{id} – najlepšia pozícia častice,
- P_{gd} – globálne najlepšie riešenie všetkých častíc,
- α a β – koeficienty určujúce váhu vplyvu P_{id} a P_{gd} kde α, $\beta \in [0, 1]$.

Po výpočte rýchlosti V_{id} novú pozíciu do ďalšej iterácie dostaneme nasledovne:

$$X_{id} = X_{id} + V_{id}. \tag{31}$$

PSO je evolučná výpočtová metóda pretože má nasledujúce atribúty:

- má inicializačný proces, v ktorom sa vytvorí istá populácia (v tomto prípade pozostávajúca z častíc) a každému prvku sa priradí náhodné riešenie,
- hľadá nové lepšie riešenia v prehľadávacom priestore tak, že produkuje nové, lepšie generácie,
- produkcia novej generácie je založená na predchádzajúcej generácii.

Pre ďalšiu úvahu budeme uvažovať PSO priamo pre riešenie TSP. Predpokladajme postupnosť riešenia TSP s n uzlami:

$$S = (a_i),\ i = 1,\ ...,\ n. \tag{32}$$

Tu definujeme *swap operátor* $SO(i_1, i_2)$ ako zámenu uzlov a_{i1} a a_{i2} v riešení S. Ďalej definujeme S' $= S + SO(i_1, i_2)$ ako nové riešenie, na ktoré bol použitý operátor $SO(i_1, i_2)$. Takže znamienko „+“ tu nadobúda nový význam. Pre názornosť uvedieme príklad: predpokladajme, že existuje riešenie TSP, obsahujúce päť uzlov $S = \{1, 3, 5, 2, 4\}$. Swap operátor je $SO(1, 2)$, potom:

$$S' = S + SO(1,\ 2) = (1,\ 3,\ 5,\ 2,\ 4) + (1,\ 2) = (3,\ 1,\ 5,\ 2,\ 4). \tag{33}$$

Ďalej si vysvetlíme nový význam znamienka „-“. Predpokladajme, že existujú dve riešenia A, B a našou úlohou je zostrojiť základnú swap sekvenciu SS, ktorá dokáže za pomoci riešenia B dostať riešenie A. Definujeme $SS = A - B$ (v tomto prípade znamieno „-“ tiež nadobúda nového významu). Môžeme potom zamienať uzly v B na základe A zľava doprava, aby sme dostali SS. Takže musí existovať rovnosť $A = B + SS$. *Swap sekvencia* SS je vytvorená z jedného alebo viacero swap operátorov:

$$SS = (SO_1, SO_2, ..., SO_n), \tag{34}$$

SO_1, SO_2, ..., SO_n sú swap operátory, ktorých poradie v SS je tu veľmi dôležité, pretože swap sekvencia vždy iniciuje swap operátory v poradí, v ktorom sa nachádzajú v swap sekvencii. Môžeme to demonštrovať nasledovne:

$$S' = S + SS = S + (SO_1, SO_2, ..., SO_n) = ((S + SO_1) + SO_2) + ... + SO_n. \tag{35}$$

Rôzne swap sekvencie, pôsobiace na rovnaké riešenie, môžu vytvoriť rovnaké nové riešenie. Všetky tieto swap sekvencie sú pomenované ako ekvivalentná množina swapových sekvencií. V tejto množine sa sekvencia s najmenej swap operátormi tiež nazýva ako *základná swap sekvencia*.

V skratke môžeme potom chod algoritmu PSO popísať nasledovne: na začiatku sa každej častici priradí náhodné riešenie. Následne sa zaháji cyklický algoritmus, ktorý bude ukončený po splnení stanovenej podmienky (napríklad prekročí sa stanovený maximálny počet iterácií alebo čas, a pod.). Pre každú časticu sa vypočíta nová pozícia. Vypočíta sa rozdiel medzi P_{id} a X_{id}, ďalej $A = P_{id}$ - X_{id} a $B = P_{gd}$ - X_{id} kde A, B sú základné swap sekvencie (operátor mínus má ten význam, že príslušná sekvencia pôsobí na druhý argument s cieľom získať prvý argument). Ďalej sa vypočíta V_{id} podľa predpisu (30) a V_{id} sa pretransformuje na základnú swap

sekvenciu. Vypočíta sa nové riešenie podľa predpisu (31). Ak sa nájde nové lepšie riešenie, aktualizuje sa P_{id}. Na konci cyklického výpočtu sa aktualizuje hodnota P_{gd}, ak sa našlo nové najlepšie riešenie. Po ukončení cyklického výpočtu PSO vráti najlepšie globálne riešenie P_{gd} [23]. Pre potreby TSP-TW vznikla modifikácia algoritmu PSO s názvom Genetický PSO. Viac o využití tohto druhu algoritmu PSO pre TSP-TW je možné dočítať sa v práci: [24].

Artificial Bee Colony

Artifical Bee Colony (ďalej iba ABC) je algoritmus inšpirovaný správaním roja včiel. Vo využití ABC pre optimalizáciu môžeme včely rozdeliť na tri skupiny:

- *včely medonosné* – ich úlohou je náhodne prehľadávať ohraničený priestor, hľadať v ňom nektár a následne zdielať informácie získané pri hľadaní nektáru s nasledovníkmi,
- *nasledovníci* – včely, ktoré sa snažia nájsť čo najlepší nektár za použitia informácií získaných včelami medonosnými, tieto včely majú určitú vlastnú selekčnú stratégiu,
- *skauti* – túto skupinu tvoria včely medonosné, ktoré opustili nektár, pretože ho zhromažďovali už vopred určený počet cyklov alebo zhromažďujú iba malé množstvo medu; skauti neskôr začnú skúmať nové zdroje nektáru.

Tab. 2: Vzťah medzi ABC a TSP.

Artificial Bee Colony	**TSP**
včely medonosné	iterácie možnými trasami
hodnota fitness	dĺžka trasy
skauti	nájdenie novej trasy
nasledovníci	ladenie trasy

Ďalej si popíšeme fázy, ktorými ABC prechádza. Sú konkrétne dve. Prvou je počiatočná fáza - predpokladajme m medonosných včiel, m nasledovníkov a D-dimenzionálny vektor $X_{ij}(j = 1, 2, 3, ..., D)$ reprezentujúci lokáciu i-tého zdroju medu. Predpokladajme ďalej, že pri každom zdroji medu operuje práve jedna včela medonosná, teda máme m zdrojov medu. Inicializácia vyzerá následovne:

$$X_{ij} = X_{min\ j} + \varphi(X_{max\ j} + X_{min\ j}) \tag{36}$$

kde:

- X_{ij} – momentálna pozícia kde $i \in \{1, 2, ..., m\}$,
- $X_{max\ j}$ a $X_{min\ j}$ – maximálne a minimálne hodnoty v j-tej dimenzii,
- φ – náhodné číslo pričom $\varphi \in [0, 1]$.

Druhá fáza sa nazýva ťažiaca fáza. Po tom, čo včely medonosné nájdu zdroj medu, kvalita nektáru v ňom je ohodnotená ako fitness, ktorý je daný nasledovne:

$$F_1 = \begin{cases} \frac{1}{1+f_i}, & f_i \geq 0 \\ 1 + |f_i|, & f_i < 0, \end{cases} \tag{37}$$

O tom, či bude cesta aktualizovaná sa nasledovníci rozhodnú na základe nasledujúceho predpisu:

$$P_i = \frac{F_i}{\sum_{j=1}^{m} F_i}, \tag{38}$$

kde definície premenných vo vyššie spomenutých výrazoch:

- f_i – adaptibilita i-teho zdroja medu,
- F_i – fitness hodnota i-teho zdroja medu.

Včely medonosné a nasledovníci aktualizujú hodnoty pozicií nasledujúcim predpisom:

$$V_{ij} = X_{ij} + \varphi(X_{ij} - X_{kj}), \tag{39}$$

kde V_{ij} označuje novú pozíciu, pričom $k \in \{1, 2, ..., m\}$ a $k \neq i$ je náhodné číslo.

Chod algoritmu môžeme popísať nasledovne. Na začiatku sa inicializujú základné parametre a vypočíta sa počiatočná pozícia včiel podľa predpisu (36). Ďalej sa vytvorí cyklus, v ktorom sa nasledovne vypočíta fitness hodnota nektáru, nájdeného včelami medonosnými, podľa predpisu (37). Porovná sa s ostatnými hodnotami a uloží sa optimálna hodnota. Ďalším krokom je to, že nasledovníci vyberú včely medonosné podľa predpisu (38) a aktualizujú pozíciu nového zdroja medu podľa predpisu (39). Porovná sa fitness hodnota aktualizovanej pozície a uloží sa optimálna hodnota. Ak existuje aspoň jedna skautská včela, počiatočná pozícia sa reviduje podľa (36) a vykoná sa aktualizácia optimalizácie. Ak skautská včela neexistuje a počet vykonaných iterácií prekonal prednastavený počet maximálneho počtu iterácií, cyklus sa ukončí a program vráti najlepšiu nájdenú hodnotu. Ak počet vykonaných iterácií ešte nedosiahol maximum, cyklický výpočet pokračuje [26]. V súvislosti s problémom TSP-TW bola vytvorená modifikácia algoritmu ABC s názvom Diskrétny algoritmus ABC. Viac o využití tohto druhu algoritmu ABC pre TSP-TW je možné dočítať sa v práci: [27].

4 Vybrané stratégie riešiace TSP-TW

V tejto kapitole si popíšeme implementáciu vybraných stratégií priamo na náš prípad TSP-TW. Popíšeme si, aké modifikácie museli byť urobené oproti použitiu na klasický TSP. Pre lepšie pochopenie si ale v úvode popíšeme, ako vyzerajú vstupné dáta a celkovo, ako vyzeral náš prípad TSP-TW a čo ho reprezentovalo.

Vstupné dáta boli reálne lety medzi existujúcimi letiskami. Uzly boli teda zastúpené letiskami, hrany jednotlivými letmi a časové okno pozostávalo z toho, že nie každý deň existoval z mesta x let do všetkých ostatných miest. Jednotlivým letiskám boli pre každý deň pridelené konkrétne lety na iné letiská. Zohľadnená teda musela byt možnosť, že nie vždy existuje z uzla x v i-tý deň cesta do všetkých ostatných uzlov. Tomuto museli byť prispôsobené všetky stratégie, aby tak nedochádzalo k slepým uličkám. Pre implementáciu boli vybraté jedna stratégia z neinformovaných metód a päť stratégií z metód informovaných. Konkrétne jedna z jednoduchších heuristík a po dve stratégie z ostatných podskupín informovaných metód prehľadávania okrem GA. Boli teda vybrané:

- neinformované metódy:
  - Random Search
- jednoduché heuristiky:
  - Hill-Climb
- metaheuristiky:
  - Simulated Annealing
  - Tabu Search
- metódy rojovej inteligencie:
  - Particle Swarm Optimization

Pred tým ako si začneme popisovať jednotlivé použité stratégie, popíšeme si fungovanie 2-opt algoritmu, ktorý bol pri stratégiách použitý ako pomocné lokálne prehľadávanie.

4.1 2-opt

Tento algoritmus nám sám o sebe riešenie nenájde. Je však veľmi dobre použiteľný na optimalizáciu riešenia nájdeného nejakou zložitejšou stratégiou. Pri implementácii bol použitý pre všetky stratégie okrem Hill-Climb algoritmu.

Popis chodu 2-opt algoritmu môžeme popísať nasledovne. Na vstup vždy dostane zoznam riešení, ktoré vygenerovala nejaká zložitejšia metóda. Algoritmus

2-opt vyberie m najlepších riešení zo zoznamu. Na týchto m riešení sa ďalej počas i iterácií tento algoritmus sám aplikuje (skontroluje, či sa niektoré hrany v riešení krížia a ak áno, vytvorí nové spojenia hrán tak, aby sa nekrížili, až pokiaľ nenájde také riešenie, ktoré je lepšie ako pôvodné a zároveň je korektné, teda neobsahuje slepú uličku). To znamená, že po prvej iterácii vytvorí zoznam A s počtom nových riešení m. V druhej iterácii sa algoritmus 2-opt znova aplikuje, tentokrát na všetkých m riešení v zozname A, a novo vytvorené – lepšie – riešenia zapíše do zoznamu B. Takto sa cyklus opakuje a po dosiahnutí i-tej iterácie sa z konečného zoznamu vyberie najlepšie riešenie, ktoré je celkovým riešením.

4.2 Random Search

Metóda Random search je jediným zástupcom neinformovaných metód prehľadávania medzi implementovanými metódami. Fungovanie tohto algoritmu je veľmi jednoduché, riešenia sú čisto náhodné (avšak korektné) a algoritmus neobsahuje žiaden mechanizmus zamedzujúci uviaznutiu v lokálnom extréme. Algoritmus si pri expanzii nových miest vždy najprv zistí, ktoré lety je v daný deň z daného mesta možné absolvovať. Následne sa náhodne rozhodne pre jeden z nich a za jeho využitia expanduje nové mesto. Ak sa stane, že z určitého mesta už neexistuje let do doposiaľ nepreskúmaného miesta, riešenie sa zahodí a ide sa odznova. V takomto prípade sa iterácia nepočíta – počítajú sa vždy iba iterácie, v ktorých je nájdené korektné riešenie. Keďže ide o neinformovanú metódu (respektíve určuje sa vždy až kvalita konečného riešenia pomocou externej funkcie, nie medzikrokov), pravdepodobnosť nájdenia najlepšieho riešenia je úplne rovnaká ako pre prvú, tak aj pre poslednú vykonanú iteráciu. Všetky výsledky sa ukladajú do zoznamu riešení a tento zoznam nakoniec slúži ako vstupné dáta pre 2-opt algoritmus. Z 2-opt algoritmu vzíde finálne riešenie.

4.3 Hill-Climb

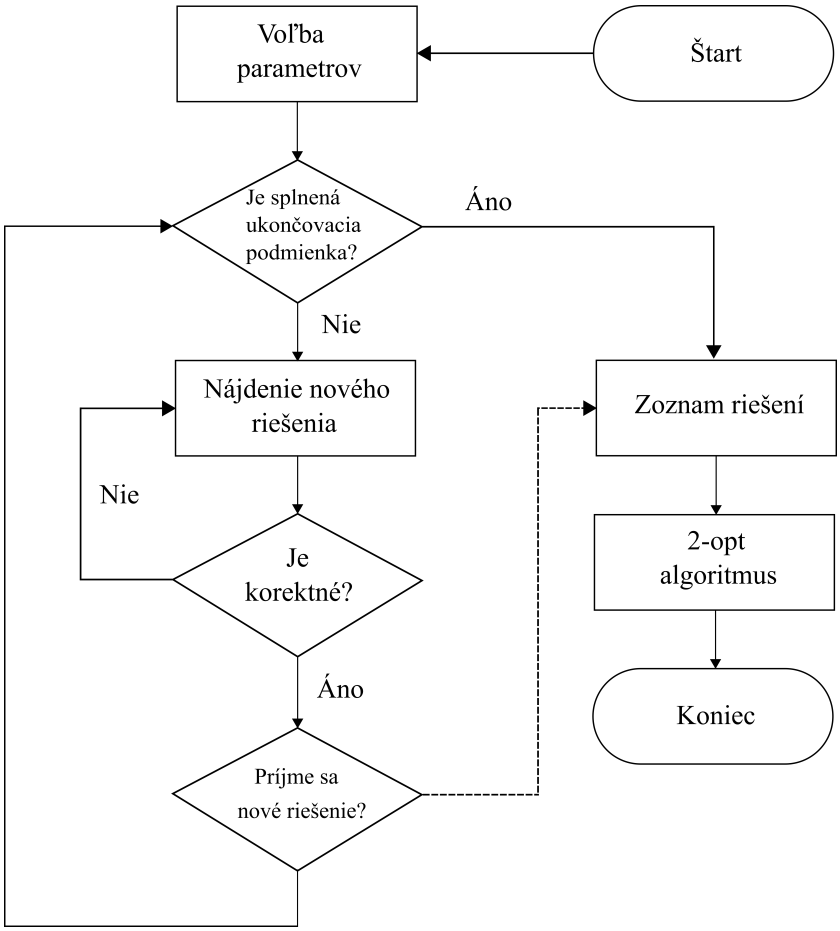
Metóda Hill-Climb je jediným zástupcom jednoduchých heuristík z implementovaných metód. Táto metóda je pomerne jednoduchá a je veľmi rýchla, pretože výpočet je realizovaný iba v jednom cykle. Do implementovaných metód bola zaradená najmä pre porovnanie s ostatnými stratégiami a jej rýchlosť. Nepredpokladá sa, že bude patriť medzi najúspešnejších riešiteľov.

Chod nami implementovaného Hill-Climbu môžeme popísať nasledovne. Pre každý deň sa vygeneruje zoznam možných subciest z práve expandovaného mesta do

ostatných miest. Z tohto zoznamu sa vždy vyberie tá, ktorá má najlepšie ohodnotenie. Takto sa pokračuje až dokým sa znovu nedosiahne počiatočného, respektíve konečného uzlu. Algoritmus priamo vráti výsledok.

4.4 Simulated Annealing

Metóda simulated annealing reprezentuje metaheuristiky. Jej vyššia zložitosť v porovnaní s jednoduchými heuristikami spočíva v tom, že SA automaticky neprijme každé riešenie, ale na základe pravdepodobnosti nové riešenie buď prijme alebo nie. Obsahuje teda určité mechanizmy, ktoré zabraňujú uviaznutiu algoritmu v lokálnom extréme. Časovo je teda náročnejšia ako dve predošlé spomenuté metódy.



Obr. 8: Schéma chodu implementovaného Simulated annealing algoritmu.

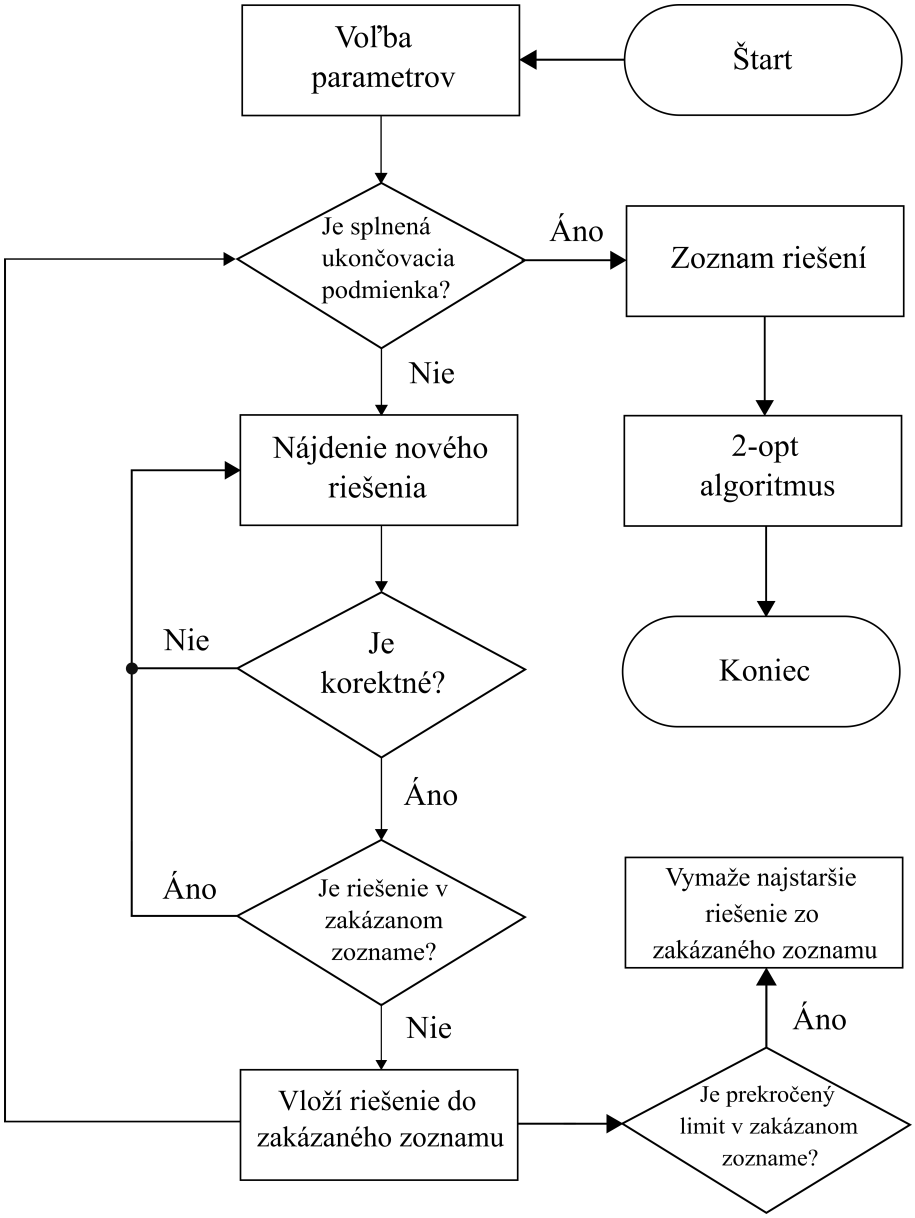
Chod SA, nami implementovanej, môžeme popísať nasledovne: na začiatku sa staticky pridelia hodnoty vstupných parametrov. Nastaví sa hodnota počiatočnej teploty, mraziacej teploty a koeficientu α. Ďalej sa spustí cyklus, v ktorom sa po každej iterácii kontroluje, či je splnená ukončovacia podmienka (v tomto prípade či

teplota klesne pod mraziacu teplotu). Ak nie je, pokračuje sa nájdením nového riešenia. Riešenie sa hľadá náhodne a tento úkon sa vykonáva až pokým nie je nájdené korektné riešenie (bez slepej uličky). Ak sa takéto riešenie nájde, posunie sa do rozhodovacieho mechanizmu. Tento mechanizmus rozhoduje o tom, či je nové riešenie prijaté a následne doplnené do zoznamu riešení, alebo je odmietnuté. Pravdepodobnosť prijatia nového riešenia závisí na vzťahu (19). Na konci iterácie sa upraví hodnota novej teploty ako súčin momentálnej teploty a koeficientu α. Ak po niektorej z iterácií bude platiť, že teplota klesla pod hodnotu mraziacej teploty, cyklus sa ukončí a zoznam riešení sa vloží do 2-opt algoritmu. Ten vyberie najlepšie z riešení v zozname, aktualizuje ich svojim mechanizmom a vráti najlepšie nájdené riešenie. Po tomto sa algoritmus ukončí.

4.5 Tabu Search

Metóda Tabu search je ďalšia z radu metaheuristík. Jej vyššia zložitosť v porovnaní s jednoduchými heuristikami spočíva v tom, že algoritmus neprijme nové riešenia, ak sa už nachádzajú v zakázanom zozname. Do zoznamu obsahujúceho zakázané riešenia sa dostane riešenie vtedy, ak v ňom ešte nie je. Tento zoznam má fixnú dĺžku a ak sa presiahne hraničná fixná veľkost tohto zoznamu, najstaršie riešenie sa odstráni.

Chod algoritmu môžeme popísať nasledovne. Na začiatku sú zvolené statické hodnoty parametrov. Určí sa počet iterácií pre opakovanie chodu algoritmu a počet iterácií, koľkokrát nezmenená hodnota riešenia znamená predpoklad, že sme našli globálny extrém. Druhý parameter slúži na to, aby sa chod algoritmu zbytočne nepredlžoval, ak je už zjavné, že sme dospeli do globálneho extrému. Oba tieto parametry slúžia ako ukončovacia podmienka. Posledným parametrom je statická veľkosť zakázaného zoznamu. Po voľbe parametrov sa skontroluje, či je ukončovacia podmienka splnená, ak nie je, pokračuje sa nájdením nového riešenia. Toto riešenie sa hľadá dovtedy, dokým nie je korektné. Nájdené korektné riešenie sa porovná s riešeniami v zozname zakázaných riešení. Ak už sa v ňom nachádza, riešenie sa ignoruje a hľadá sa nové riešenie (celý algoritmus sa posunie o krok dozadu). Ak dané riešenie naopak v zakázanom zozname nie je, pokračuje sa jeho vložením do zakázaného zoznamu a do zoznamu riešení. Prebehne tiež kontrola dĺžky zakázaného zoznamu a ak je prekročená, vymaže sa z neho najstaršie riešenie. Pokračuje sa kontrolou, či je splnená podmienka, ktorá ukončuje cyklus. Po splnení niektorej z ukončovacích podmienok sa na zoznam riešení aplikuje 2-opt algoritmus, ktorý ešte vylepší najlepšie doposiaľ nájdené riešenia a vráti z nich to najlepšie.



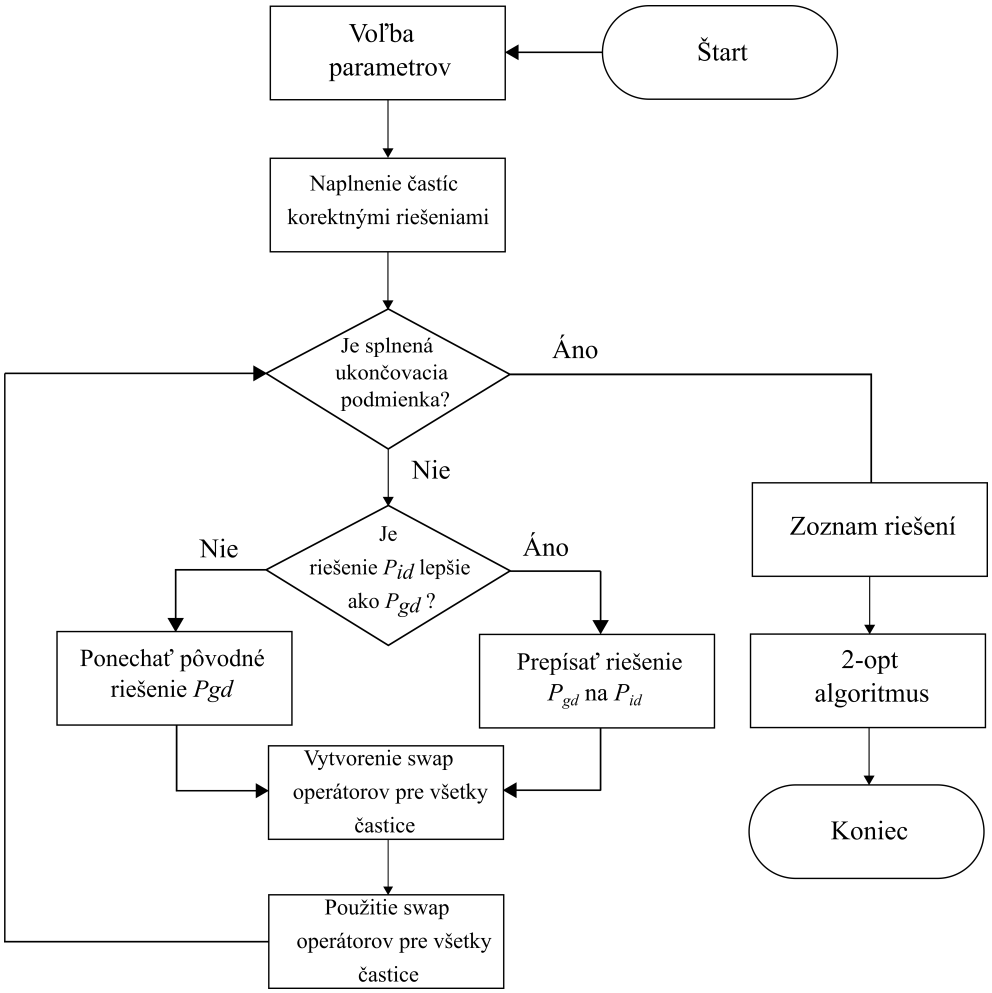
Obr. 9: Schéma chodu implementovaného Tabu search algoritmu.

4.6 Particle swarm optimization

Metóda Particle swarm optimization je metóda spadajúca pod metaheuristiky, konkrétnejšie pod metódy inšpirované živou prírodou – konkrétne chovaním roja určitého druhu živočíchov. Hlavnou komponentou PSO sú častice, ktoré sa v počiatku naplnia náhodnými riešeniami a ďalej sa upravujú. Swap operátory, ktoré umožňujú zmenu riešení v časticiach medzi generáciami, sa snažia nájsť vždy lepšie riešenia za pomoci lokálneho najlepšieho riešenia danej častice a globálneho najlepšieho riešenia všetkých častíc.

Chod algoritmu môžeme popísať nasledovne: na začiatku sa zvolia konštantné hodnoty vstupných parametrov. Vstupné parametry sú koeficienty α a β, počet iterácií, pre ktoré sa bude chod algoritmu vykonávať a veľkosť populácie, teda koľko častíc sa má na začiatku vygenerovať a naplniť náhodným riešením. Pri plnení častíc sa musí pre každú časticu vygenerovať korektné riešenie. Po naplnení častíc riešeniami sa spustí cyklus, ktorého jediná ukončovacia podmienka je počet iterácií. Vo vnútri tohto cyklu sa spustí ďalší cyklus, ktorý pre každú časticu vykoná nasledovné: najprv sa vytvoria swap operátory z porovnania momentálneho riešenia v danej častici a z najlepšieho riešenia v danej častici. Jednotlivým swap operátorom sa po porovnaní prvkov momentálneho a najlepšieho lokálneho riešenia priradí prvok, ktorý sa bude zamienať v momentálnom riešení. Swap operátorom sa tiež priradí poloha tohto prvku v najlepšom lokálnom riešení tejto častice a konštantná hodnota α. Tak isto sa v ďalšom kroku vytvoria swap operátory pre porovnanie momentálneho riešenia a globálneho najlepšieho riešenia, ale tentokrát s priradením koeficientu β. Ďalej sa tieto swap operátory aplikujú na momentálne riešenie s tým, že pravdepodobnosť každého vykonania swapu určujú koeficienty α a β, podľa toho ktorý bol akému operátoru priradený. Po vykonaní všetkých zámien nám vznikne nové riešenie, ktoré by malo byť lepšie ako riešenia z minulej generácie. Nakoniec sa vyhodnotí, či riešenie, ktoré sme dostali po vykonaní všetkých zámien je korektné, alebo ho treba upraviť. Ak algoritmus zistí, že dané riešenie neexistuje, začne náhodne zamienať dvojice uzlov v riešení, až dokým nenájde také riešenie, ktoré je korektné - neexistuje v ňom slepá ulička a zo všetkých uzlov v riešení existuje v daný deň let do ďalšieho uzlu v poradí riešenia. Pri tomto úkone hrozí, že prídeme o výhodu vyplývajúcu z toho, že ďalšia generácia by mala mať lepšie riešenie. Je to však cena za dodržiavanie podmienky časového okna. Navyše nie je vylúčené, že po náhodnej výmene dvoch uzlov nedosiahneme riešenie lepšie. V ďalšom kroku sa vykoná kontrola, či je nové riešenie lepšie ako lokálne najlepšie riešenie danej častice, a tiež, či je lepšie ako globálne najlepšie riešenie všetkých častíc. Ak áno, prepíšu sa. Po vykonaní všetkých iterácií budú najlepšie globálne riešenie spolu s niekoľkými najlepšími lokálnymi riešeniami z častíc slúžiť ako vstup do 2-opt algoritmu, ktorý

ich upraví a vyberie z nich to najlepšie, ktoré sa nakoniec vráti ako výsledné riešenie PSO algoritmu.



Obr. 10: Schéma chodu implementovaného PSO algoritmu. Zoznam riešení bude v tomto prípade obsahovať P_{gd} a niekoľko najlepších P_{id}.

5 Popis implementácie

V tejto kapitole si popíšeme, ako bola prevedená a čím bola inšpirovaná implementácia. Popíšeme si pravidlá vychádzajúce z dvoch súťaží, ktorými sa implementácia riadila. Ďalej si popíšeme vstupné dáta a vysvetlíme si tiež postup použitý pri programovaní testovacej aplikácie TSP-TW solver. Tento postup bude obsahovať najmä popis implementovaných tried v programovacom jazyku Python, v ktorom bola aplikácia naprogramovaná. Ukážeme si, ako spolu jednotlivé triedy súvisia a ako na seba nadväzujú. V ďalšej časti tejto kapitoly si opíšeme chod aplikácie TSP-TW solver a tiež popis jej užívateľského rozhrania.

5.1 Realizácia implementácie

Celá implementácia bola inšpirovaná dvoma súťažami organizovanými spoločnosťou Kiwi.com s.r.o. Úlohou súťažiacich bolo vytvoriť aplikáciu, ktorá nájde najlepšiu trasu na základe daných údajov o letoch. Obchodného cestujúceho z TSP-TW v tejto úlohe predstavoval cestujúci, ktorý sa snažil nájsť čo najlacnejšiu trasu poskladanú z letov. Cestujúci musel každý deň vykonať práve jeden let a musel navštíviť všetky definované mestá. Pri testovaní implementovaných stratégií boli zo súťaží použité nasledujúce položky:

- čiastočné pravidlá súťaží,
- vstupné testovacie datasety,
- verifikačné testovacie skripty.

Čiastočne použité pravidlá zo súťaží vyzerali nasledovne [28]:

- úlohou bolo nájsť Hamiltonovskú kružnicu v časovo závislom orientovanom grafe,
- dané bolo:
  - m letov operujúcich medzi n mestami,
  - n dní na navštívenie n miest,
  - počiatočné, respektíve konečné mesto,
- zahájenie dňom i v meste x znamená, že je možné chytiť všetky lety operujúce počas dňa i+1 z mesta x,
- všetky lety sú okamžité, čas potrebný na nástup do lietadla a výstup z lietadla sa neberie do úvahy,
- každý deň je nutné vykonať práve jeden let,

- môže sa stať, že v niektoré dni nebudú existovať lety medzi niektorými mestami,
- je garantované, že z údajov zo vstupných dát je možné zostrojiť minimálne jednu Hamiltonovskú kružnicu.

Vstupné dáta boli textové súbory s koncovkou *.txt*, ktoré obsahovali informácie o jednotlivých letoch. Prvý riadok vždy obsahoval IATA kód počiatočného letiska (IATA kód je identifikátor slúžiaci k jednoduchšiemu rozpoznaniu daného letiska a pre každé letisko je unikátny [35]). Všetky ostatné riadky už mali rovnaký formát. Na začiatku bol vždy IATA kód letiska, z ktorého let začínal. Následoval IATA kód letiska, v ktorom let končil, ďalej deň, v ktorý bolo daný let možné uskutočniť a na konci bol údaj o cene tejto trasy. Všetky tieto údaje boli v textových dokumentoch oddelené medzerou. Zhrnutie informácií o tom, ako vyzerali vstupné dáta teda vyzerá nasledovne:

- <odkiaľ, kam> – IATA kódy letísk pozostávajúce z troch písmen anglickej abecedy písané veľkými písmenami (A až Z),
- <cena> – pozostáva z kladného celého čísla ($0 < \text{Cena} \leq 65535$),
- <deň> – pozostáva z kladného celého čísla a symbolizuje, ktorý deň v poradí môže byť daný let vykonaný,
- vstupné dáta sú zoradené podľa abecedy.

Ďalej si ukážeme vzor, ako mohli vyzerať jednotlivé vstupné dáta:

```
NAP
BRQ FCO 1 40
BRQ NAP 1 510
FCO BRQ 0 641
FCO NAP 2 3
NAP BRQ 0 10
```

Pre kontrolu boli tiež použité testovacie skripty zo súťaží. Tieto skripty kontrolovali validitu výsledkov (formát, správnosť, uskutočniteľnosť). Tieto skripty sú dostupné na webovej adrese: [28]. Viac informácií o súťažiach možno nájsť na webovej adrese: [29, 30].

5.2 Voľba programovacieho jazyka

Pre implementáciu bol zvolený programovací jazyk Python. Python je interpretovaný, interaktívny, objektovo orientovaný programovací jazyk. Zahŕňa moduly, výnimky, dynamické písanie, dynamické dátové typy na vysokej úrovni a triedy. Podporuje viac paradigiem programovania presahujúcich objektové programovanie, napríklad procedurálne a funkčné programovanie. Python kombinuje pozoruhodnú silu s veľmi jasnou syntaxou. Má rozhrania s mnohými systémovými volaniami a knižnicami, ako aj s rôznymi okennými systémami a je rozšíriteľný v jazykoch C alebo C++. Je tiež použiteľný ako rozšírený jazyk pre aplikácie, ktoré potrebujú programovateľné rozhranie. Python je tiež prenosný – beží na mnohých variantoch Unixu vrátane Linuxu a macOS a na operačnom systéme Windows [37].

Hlavnou motiváciou pre voľbu tohto programovacieho jazyka bola jeho jednoduchosť a obrovské množstvo knižníc. Nevýhodou voľby tohto jazyka je však väčšia časová náročnosť pri výpočte, napríklad oproti programovaciemu jazyku C++, v ktorom je možné mechanizmy optimalizovať tak, aby boli časovo menej náročné.

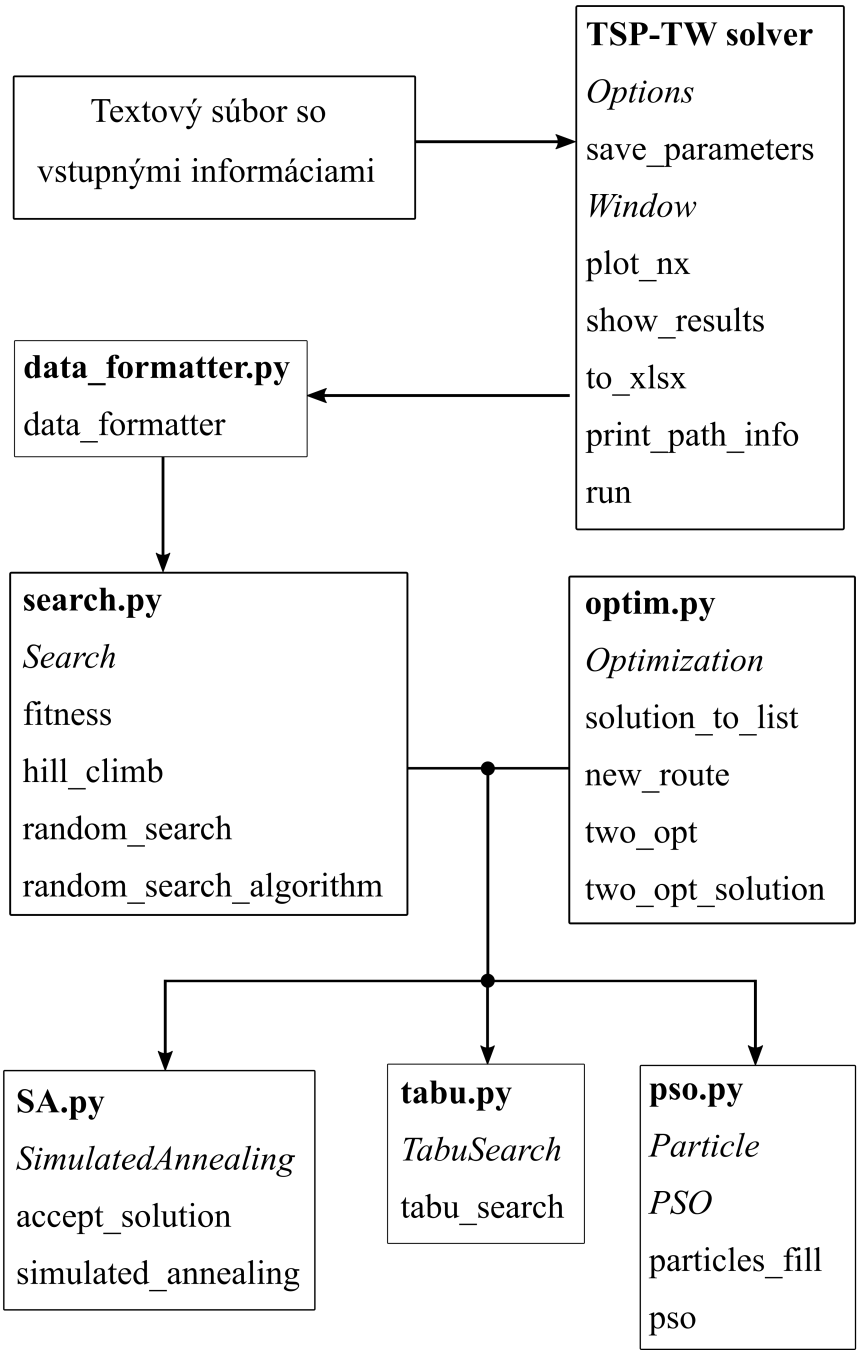
5.3 Triedy vytvorené pri implementácii

Pre prípad testovacej aplikácie TSP-TW solver bolo implementovaných viacero tried. V tejto kapitole si popíšeme ich podobu a tiež ich náväznosť na seba. Ich schému môžeme vidieť na obrázku 11.

5.3.1 Pomocné triedy a triedy obsahujúce optimalizačné prvky

Metóda data formatter

Jediná metóda, ktorá nie je priradená k žiadnej triede je metóda *data_formatter*. Z hľadiska jej unikátnej funkčnosti a lepšej štrukturalizácii bolo rozhodnuté, že bude obsiahnutá iba v samostatnom súbore a nebude pridelená pod žiadnu triedu. Túto metódu môžeme tiež nazvať akousi vstupnou bránou pre každú stratégiu. Jej úlohou je sformátovať vstupné dáta tak, aby s nimi následne mohli jednotlivé triedy pracovať. Jej vstupom je súbor obsahujúci zadanie pre TSP-TW a výstupom je zoznam možných letov (hrán v časovo orientovanom grafe), zoznam miest (uzlov v časovo orientovanom grafe) a počiatočné mesto (počiatočný uzol).



Obr. 11: Schéma vytvorených tried. Hrubým písmom sú vyznačené názvy súborov, italikou sú vyznačené triedy a klasickým písmom sú vyznačené metódy prislúchajúce jednotlivým triedam.

Trieda Search

Vstupom pre triedu *Search* sú informácie o letoch predstavujúcich hrany v časovo orientovanom grafe. Táto trieda obsahuje statickú metódu *fitness*, ktorá slúži ako fitness funkcia pre implementované stratégie. Na jej vstup sa privedie riešenie a výstupom je cena tohto riešenia. Ďalej obsahuje metódu *hill_climb*, ktorá slúži pre prevedenie Hill climb algoritmu. Jej výstupom je najlepšie nájdené riešenie tohto algoritmu spolu s cenou tohto riešenia. Ďalšia metóda, ktorú táto trieda obsahuje je metóda *random_search*. Tá je ďalej použitá v metóde *random_search_algorithm* a v niektorých ďalších.

Trieda Optimization

Vstup pre triedu *Optimization*, tak ako pre predošlú triedu, sú informácie o letoch reprezentujúcich hrany. Táto trieda obsahuje statickú metódu *solution_to_list* a nestatickú metódu *new_route*. Tieto metódy majú dôležitú úlohu pri zamieňaní uzlov a tvorení nových ciest po tom, čo sú nájdené nekorektné riešenia. Trieda ďalej obsahuje metódu *two_opt*, ktorá slúži ako základ pre metódu *two_opt_solution*. Prvá menovaná má za úlohu nájsť lepšie riešenie na základe riešenia nájdeného inou stratégiou. Druhá menovaná zase prevedie 2-opt algoritmus do viacerých iterácií a vráti najlepšie nájdené riešenie spolu s jeho cenou.

5.3.2 Triedy reprezentujúce zložitejšie heuristiky

Trieda SimulatedAnnealing

Trieda *SimulatedAnnealing* reprezentuje metaheuristický algoritmus SA. Na jej vstup sa privádza viacero parametrov. Konkrétne sú to informácie o letoch, teplota, mraziaca teplota a koeficient α. Trieda ďalej obsahuje dve metódy. Prvou je metóda *accept_solution*, ktorá rozhoduje o tom, či bude nové nájdené riešenie prijaté alebo nie (postupuje podľa vzťahu (19)). Nové riešenie sa privedie na vstup tejto metódy a tá buď zapíše toto riešenie a jeho cenu do globálneho zoznamu s riešeniami a zoznamu s cenami alebo nie. Druhou metódou je metóda *simulated_annealing*, ktorá úž priamo reprezentuje mechanizmus algoritmu SA. Zoznam s nájdenými riešeniami sa na konci chodu tejto metódy použije ako vstup pre algoritmus 2-opt. Výstupom tejto metódy je potom najlepšie nájdené riešenie spolu s jeho cenou.

Trieda TabuSearch

Trieda *TabuSearch* reprezentuje metaheuristický algoritmus TS. Vstupmi pre túto triedu sú informácie o letoch, počet iterácií chodu algoritmu, veľkosť zakázaného zoznamu a maximálny počet iterácií bez zmeny výsledku, po ktorom sa algoritmus ukončí (splní sa tak predpoklad, že algoritmus už dosiahol globálny extrém). Táto trieda obsahuje iba jednu metódu s názvom *tabu_search*, ktorá sa stará o celkový mechanizmus chodu algoritmu TS. Jej úlohou sú teda všetky úkony tohto algoritmu, konkrétne teda hľadať riešenia, prijímať alebo neprijímať ich, plniť zakázaný zoznam a tiež odstraňovať riešenia zo zakázaného zoznamu, pokiaľ je prekročená jeho fixná veľkosť. Zoznam riešení sa nakoniec privedie na vstup 2-opt algoritmu. Metóda vracia najlepšie nájdené riešenie a jeho cenu.

Triedy Particle a PSO

Algoritmus PSO je reprezentovaný triedami *Particle* a *PSO*. Trieda *Particle* reprezentuje jednotlivé častice. Vstupom je počiatočné riešenie a jeho cena, ktoré sa do častice uložia. Každá častica vždy obsahuje informácie o momentálnom riešení a jeho cene a o najlepšom lokálnom riešení danej častice a jeho cene. Každá častica tiež obsahuje premennú s názvom *rýchlosť*, v ktorej sa uchovávajú zmeny, ktoré môžu byť pre danú časticu vykonané v momentálnej iterácii. Trieda *PSO* má viacero vstupných parametrov. Konkrétne sú to informácie o letoch, počet iterácií, pre ktoré ma algoritmus PSO bežať, veľkosť populácie (teda koľko častíc bude vygenerovaných), koeficient α a koeficient β. Táto trieda ďalej obsahuje metódy $particles\_fill$ a *pso*. Prvá menovaná slúži na počiatočné naplnenie častíc korektnými riešeniami. Metóda *pso* potom obsahuje hlavné mechanizmy algoritmu PSO. Po naplnení častíc dochádza k tvorbe swap operátorov a následne k jednotlivým zámenám v časticiach. Metóda potom skontroluje korektnosť nájdených riešení a prípadne ich upraví. Na konci sa zoznam riešení aj s ich cenami použije ako vstup do algoritmu 2-opt. Metóda vracia najlepšie nájdené riešenia a jeho cenu.

5.3.3 Triedy reprezentujúce grafické rozhranie

Funkciu grafického rozhrania zabezpečujú triedy *Options* a *Window*. Trieda *Options* sa stará o funkciu okna s nastaveniami parametrov a obsahuje metódu *save_parameters*, ktorá ukladá užívateľom zvolené parametre. Trieda *Window* obsahuje viacero metód, konkrétne sú to *plot_nx* a *show_results* ktoré zabezpečujú vykresľovanie grafov, *to_xlsx*, ktorá zapisuje výsledky do súboru s príponou .xlsx, *print_path_info* ktorá vypisuje výsledky stratégií do konzoly a metódu *run*, ktorá sa stará o výpočet.

5.4 Vykresľovanie výsledkov

Vykresľovanie výsledkov, teda konkrétne vykresľovanie výsledných grafov, prebiehalo za použitia dvoch knižníc, konkrétne knižníc *NetworkX* a *matplotlib*. *NetworkX* je knižnica v programovacom jazyku Python, slúžiaca na tvorbu, manipuláciu a štúdovanie štruktúr, dynamík a funkcií komplexných sietí [44]. *Matplotlib* je obsiahla knižnica pre tvorbu statických, animovaných a interaktívnych vizualizácií [45]. Ako už v tejto práci bolo viackrát spomenuté, uzly vo výsledných grafoch reprezentovali mestá, v ktorých sa nachádzali letiská a hrany reprezentovali jednotlivé lety medzi letiskami. Jednotlivým letiskám boli priradené súradnice totožné s reálnymi (súradnice boli tvorené zemepisnou dĺžkou a šírkou). Tieto súradnice boli získané z následujúceho zdroja: [46]. Následne bol vytvorený textový súbor *airports_coords.txt*, ktorý obsahuje súradnice viac ako desaťtisíc letísk. Tento súbor sa načíta pri každom spustení aplikácie.

5.5 Aplikácia TSP-TW solver

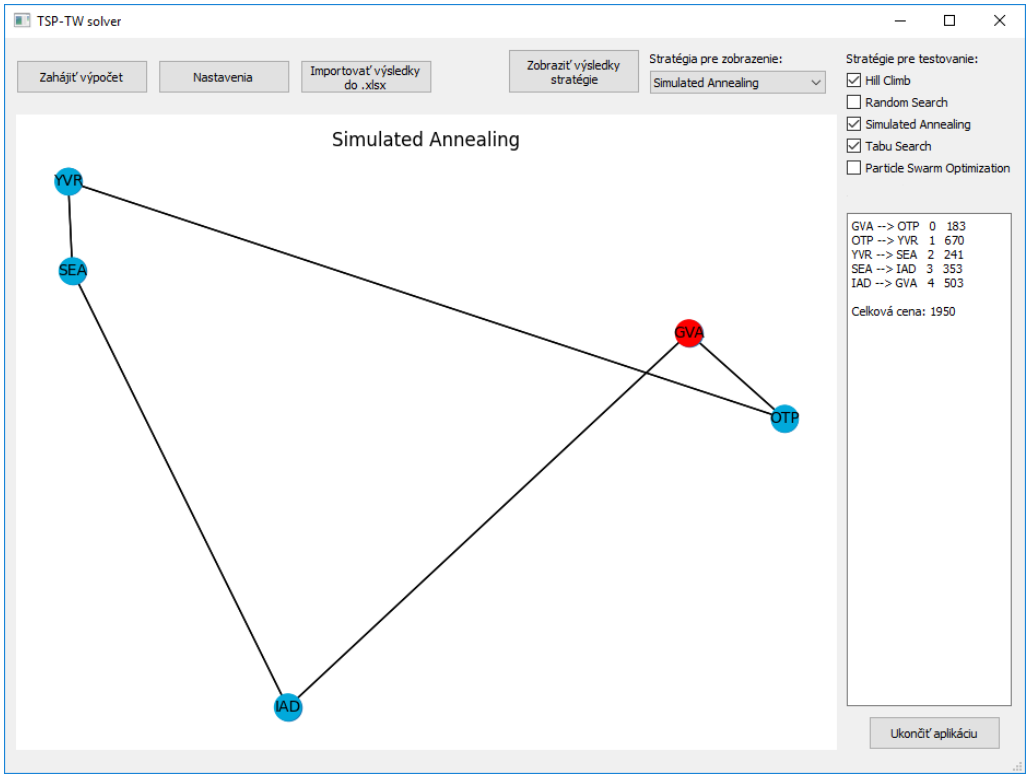
V tejto podkapitole si predstavíme vyvinutú aplikáciu *TSP-TW solver*, ktorá slúži na zobrazovanie a zápis výsledkov jednotlivých implementovaných stratégií. Ako môžeme vidieť v hlavnom okne na obrázku 12, aplikácia má viacero nastaviteľných parametrov. V hornej lište si môžeme všimnúť viacero tlačidiel. Ich funkcie, aktivované ich stlačením, sú nasledovné:

- *Zahájiť výpočet* – zaháji výpočet zvolených stratégií.
- *Nastavenia* – otvorí sa nové okno s nastaveniami (obr. 14), kde užívateľ môže zvoliť parametre jednotlivých stratégií a tiež vstupný súbor s dátami.
- *Importovať výsledky do .xlsx* – importuje výsledky všetkých testovaných stratégií do súboru s príponou *.xlsx*.
- *Zobraziť výsledky* – zobrazí výsledný graf vybranej stratégie a v malej konzole napravo zobrazí riešenie a cenu riešenia.
- *Ukončiť aplikáciu* – ukončí chod aplikácie.

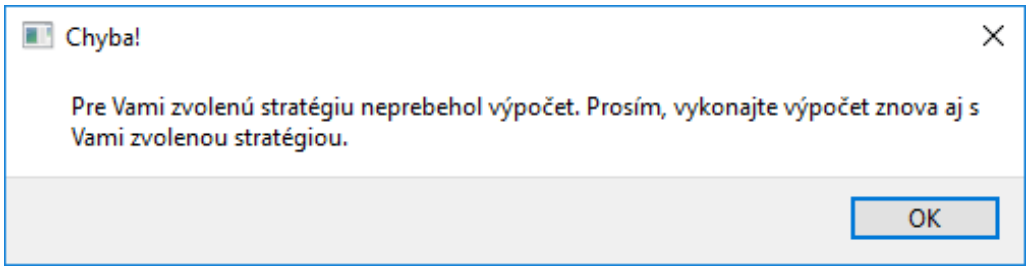
Ďalej si na obrázku 12 môžeme všimnúť možnosť výberu *Stratégia pre zobrazenie*, ktorá obsahuje rolovaciu lištu. V tejto lište si môže užívateľ zvoliť stratégiu, ktorej výsledky chce zobraziť. V pravom hornom rohu sú tiež zaškrtávacie políčka. Pomocou týchto políčiek si užívateľ zvolí, ktoré stratégie chce do výpočtu zahrnúť. Príklad použitia aplikácie môže byť nasledovný:

1. Užívateľ v pravom hornom rohu zaškrtne, ktoré aplikácie chce testovať.
2. Po kliknutí na tlačidlo *Nastavenia* zvolí parametre testovaných stratégií a vstupný súbor.
3. Užívateľ spustí výpočet stisknutím tlačidla *Zahájiť výpočet*.
4. Po úspešnom dokončení výpočtu sa užívateľovi zobrazí kontextové okno s oznamom o úspešnom dokončení výpočtu.
5. Užívateľ v rolete s názvom *Stratégia pre zobrazenie* vyberie stratégiu, ktorej výsledky chce zobraziť. V okne sa zobrazí výsledný časovo orientovaný graf a v malej konzole napravo sa zobrazí riešenie aj s ceho cenou.
6. Užívateľ môže importovať výsledky vybraných stratégií na danom datasete stlačením tlačidla *Imporotvať výsledky do .xlsx*.
7. Užívateľ môže ukončiť aplikáciu stlačením tlačidla *Ukončiť aplikáciu* alebo môže pokračovať v testoch vrátením sa ku kroku číslo 1.

Aplikácia tiež obsahuje viacero ošetrení a výnimiek. Napríklad pri pokuse o zobrazenie výsledkov stratégie, pre ktorú neprebehol výpočet, bude užívateľ upozornený v kontextovom okne (obr. 13). Tiež sa môže stať, že pri užívateľom vytvorenom vlastnom vstupnom datasete sa niektoré z letísk nebude vyskytovať v súbore s letiskami a ich súradnicami pribalenom k aplikácii. V takomto prípade sa užívateľovi tiež zobrazí kontextové okno s upozornením.

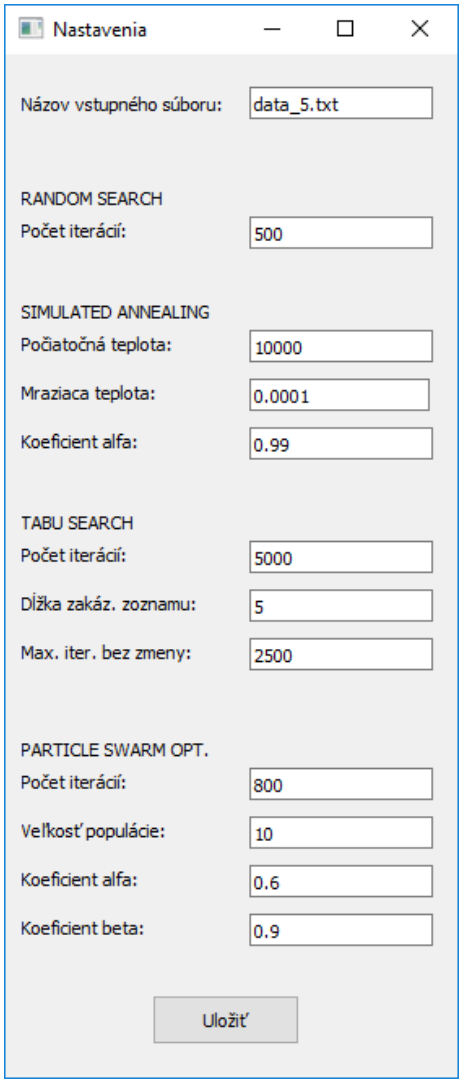


Obr. 12: Grafické rozhranie aplikácie TSP-TW solver.



Obr. 13: Príklad kontextového okna aplikácie TSP-TW solver.

Po stlačení tlačidla *Nastavenia* sa užívateľovi zobrazí okno (obr. 14), v ktorom si môže zvoliť názov vstupného súboru a parametre testovania. Ak si zvolí názov vstupného súboru, ktorý nie je správny, bude na to upozornený a zároveň bude vyzvaný, aby zvolil platný súbor. Rovnako bude upozornený pri nesprávnom vyplnení niektorého z testovacích parametrov stratégií.



Obr. 14: Nastavenia v aplikácii TSP-TW solver.

6 Výsledky testov

V tejto kapitole sa nachádza popis testovania stratégií a tiež vyhodnotenie ich efektivity. Porovnávali sme medzi sebou stratégie riešiace TSP-TW. Konkrétne teda išlo o stratégie *Random search* (zástupca neinformovaných metód prehľadávania), *Hill-Climb* (zástupca jednoduchších heuristík), *Simulated annealing* a *Tabu search* (zástupcovia klasických metaheuristík) a *Particle swarm optimization* (zástupca metaheuristík inšpirovaných správaním sa mnohopočetnej skupiny organizmov).

6.1 Parametre testovania

V tejto podkapitole si popíšeme celkové parametre testovania a parametre jednotlivých stratégií.

6.1.1 Celkové parametre testovania

Vstupnými údajmi pre každú stratégiu bol dataset reprezentujúci časovo orientovaný graf a každá stratégia našla riešenie, jeho cenu a čas, ktorý na nájdenie riešenia potrebovala. Celkovo bolo vykonaných šesť testov, každý s datasetom reprezentovaným 10, 15, 20, 30, 40, 50 uzlovými časovo orientovanými grafmi. Každý dataset bol testovaný v 15 behoch. Tento počet behov bol zvolený z dôvodu časovej náročnosti výpočtov. Testy môžeme rozdeliť do dvoch kategórií: jednoduchšie (počet uzlov ≤ 20) a zložitejšie ($30 \leq$ počet uzlov ≤ 50). Ako už bolo spomínané v predchádzajúcich kapitolách, dataset bol vždy rozpis letov zoradený abecedne. Úlohou aplikácie teda bolo z tohto datasetu vybrať si menovite všetky uzly a zistiť ich počet, ďalej z datasetu vybrať všetky hrany, teda jednotlivé lety a tieto údaje poskytnúť jednotlivým stratégiám, ktoré z nich ďalej museli poskladať riešenie. Všetky testy boli vykonané na stolnom počítači s nasledovnou hardvérovou výbavou:

- Procesor – AMD FX-8370 (4 GHz)
- Pamäť – 4x 8192 MB DDR3 (889 MHz)
- Grafické karty:
  - NVIDIA GeForce GTX TITAN X (12 GB)
  - NVIDIA GeForce GTX TITAN (6 GB)

6.1.2 Parametre jednotlivých stratégií

Jedinou stratégiou, ktorá neobsahuje žiadne parametre, ktoré bolo nutné nastaviť, bola stratégia *Hill-Climb*. Pri všetkých ostatných prebehlo *offline* ladenie parametrov. Znamená to, že parametre neboli ladené samostatne počas chodu algoritmu, ale boli ladené ručne na základe metódy pokus-omyl. Pre jednotlivé stratégie boli teda pre testovanie zvolené nasledujúce parametre:

- Random search
  - počet iterácií: 1000
- Simulated annealing
  - počiatočná teplota: 10000
  - mraziaca teplota: 0,0001
  - koeficient α: 0,999
- Tabu search:
  - maximálny počet iterácií: 3000
  - veľkosť zakázaného zoznamu: 5
  - maximálny počet iterácií bez zmeny riešenia: 1000
- Particle swarm optimization:
  - počet iterácií: 1000
  - veľkosť populácie: 10
  - koeficient α: 0,6
  - koeficient β: 0,9

6.2 Výsledky jednotlivých testov

Z pätnástich behov, ktorými prešla každá stratégia v každom datasete (okrem stratégie Hill-Climb, pri nej bol prevedený vždy iba jeden beh, keďže vracia vždy rovnaký výsledok) sa vyhodnotili výsledky. Smerodatné teda sú: aritmetický priemer cien, medián cien, maximálna a minimálna nájdená hodnota ceny a priemerný čas, za ktorý bola stratégia schopná nájsť riešenie. Ako hlavné ukazovatele kvality riešenia môžeme označiť medián cien a minimálnu nájdenú cenu riešenia. Všetky ceny sú uvedené v eurách a časy v sekundách.

6.2.1 Test č.1

Test č. 1 bol prevedený na grafe o veľkosti 10 uzlov. Dataset reprezentujúci tento graf sa nachádza v textovom súbore s názvom *data_10.txt*.

Tab. 3: Výsledky testu č.1.

Stratégia	**Aritm. priem. cien**	**Medián cien**	**Max. cena**	**Min. cena**	**Aritm. priem. časov**
Hill-Climb	5832	5832	5832	5832	0,01
Random search	5452,63	5417	5631	5375	1,33
Sim. annealing	5386,1	5375	5456	5375	8,98
Tabu search	5383,2	5375	5491	5375	3,96
PSO	5761,1	5600	6280	5375	9,42

Z výsledkov prvého testu (tab. 3) môžeme vidieť, že prvý dataset nespôsobil žiadnej stratégii výraznejšie problémy. Štyrom stratégiam sa podarilo nájsť optimálne riešenie s hodnotou 5375. Konkrétne to boli stratégie *Random search*, *Simulated annealing*, *Tabu search* a *PSO*. Tento 10 uzlový graf reprezentuje stále ešte pomerne malý prehľadávací priestor a je zrejmé, že k výrazným výkyvom vo výsledkoch jednotlivých stratégií tu nedochádza. Najmenej efektívnou stratégiou pri tomto datasete sa ukázala byť *Hill-Climb*, ktorej najlepšie nájdené riešenie malo hodnotu 5832. Zo stratégií, ktoré našli optimálne riešenie, bola najúspešnejšou stratégia *Tabu search*. *Random search* síce potrebovala na nájdenie riešenia najkratší priemerný čas (1,33 sekundy), ale *Tabu search* mala lepšiu mediánovú hodnotu cien, čo znamená, že lepšie riešenie našla pri viacerých iteráciach. Dobrý výsledok *Random search* stratégie však ukázal, že pri menších problémoch ako je tento bude rozumné použiť stratégie založené na náhodnom prehľadávaní. Tento 10-uzlový dataset môže mať až 10! možných riešení. V číselnom vyjadrení je to viac ako 3,5 milióna.

6.2.2 Test č.2

Test č.2 bol prevedený na grafe o veľkosti 15 uzlov. Dataset reprezentujúci tento graf sa nachádza v textovom súbore s názvom *data_15.txt*.

Z výsledkov druhého testu (tab. 4) môžeme konštatovať, že znova nedošlo k výrazným výchylkám pri výsledkoch stratégií a nájsť teda dobré riešenia v 15-uzlovom časovo orientovanom grafe pre stratégie znovu nebol problém. Najhorší výsledok našla opäť stratégia *Hill-Climb*, konkrétne riešenie s cenou 4801. Naopak,

najúspešnejšou bola v tomto datasete stratégia *Simulated annealing*, ktorá našla riešenie s hodnotou 4370 a hodnotu mediánu cien mala 4531. Tento 15-uzlový dataset môže mať až 15! možných riešení. V číselnom vyjadrení to môže byť až $1{,}31 \cdot 10^{12}$.

Tab. 4: Výsledky testu č.2.

Stratégia	**Aritm. priem. cien**	**Medián cien**	**Max. cena**	**Min. cena**	**Aritm. priem. časov**
Hill-Climb	4801	4801	4801	4801	0,01
Random search	4682,17	4675	4976	4398	6,95
Sim. annealing	4491,9	4531	4590	4370	56,45
Tabu search	4599,67	4578	4797	4445	19,16
PSO	4685,5	4655,5	5142	4421	48,72

6.2.3 Test č.3

Test č.3 bol prevedený na grafe o veľkosti 20 uzlov. Dataset reprezentujúci tento graf sa nachádza v textovom súbore s názvom *data_20.txt*.

Tab. 5: Výsledky testu č.3.

Stratégia	**Aritm. priem. cien**	**Medián cien**	**Max. cena**	**Min. cena**	**Aritm. priem. časov**
Hill-Climb	9889	9889	9889	9889	0,02
Random search	8642,37	8645,5	9402	7814	24,41
Sim. annealing	6615,8	6599,5	6911	6383	194,61
Tabu search	7961,57	7887	8527	7486	66,03
PSO	8059,1	8046,5	8771	7311	138,57

Z výsledkov tohto testu (tab. 5) môžeme pozorovať, že s počtom uzlov začína narastať potrebný čas pre jednotlivé stratégie na nájdenie riešenia. Najlepšie riešenie našla stretégia *Simulated annealing*, konkrétne cenu trasy 6383 s priemerným časom 194,61 sekúnd. Pre porovnanie s ňou môžeme zvoliť stratégiu *Tabu search*, ktorej najlepšie riešenie s hodnotou 7486 bolo nájdené v priemernom čase 66,03 sekúnd. *Simulated annealing* mala zároveň aj najlepšiu hodnotu mediánu cien, konkrétne 6599,5 a môžeme ju v tomto teste označiť ako najlepšiu. Naopak, najhorší výsledok našla stratégia *Hill-Climb*, konkrétne cenu trasy 9889. Tento dataset o veľkosti 20

uzlov môže mať až 20! možných riešení. Počet možných riešení sa teda môže rovnať až $2{,}4 \cdot 10^{18}$.

6.2.4 Test č.4

Test č.4 bol prevedený na grafe o veľkosti 30 uzlov. Dataset reprezentujúci tento graf sa nachádza v textovom súbore s názvom *data_30.txt*.

Tab. 6: Výsledky testu č.4.

Stratégia	**Aritm. priem. cien**	**Medián cien**	**Max. cena**	**Min. cena**	**Aritm. priem. časov**
Hill-Climb	8847	8847	8847	8847	0,07
Random search	13451,64	13538	14452	12177	113,75
Sim. annealing	8685,63	8783,5	9033	8019	2793,12
Tabu search	121513,8	12669	13575	11943	283,23
PSO	12127,4	11971	13205	10684	962,46

Z výsledkov tohto testu (tab. 6) môžeme vidieť prvé vážnejšie odchýlky vo výsledkoch jednotlivých stratégií. Ukázalo sa, že tento 30 uzlový graf reprezentoval už pomerne zložitý problém na vyriešenie. *Random search*, *Tabu search* a *PSO* prišli s oveľa horšími výsledkami ako *Hill-Climb* a *Simulated annealing*, ktorých najlepšie riešenia mali hodnotu ceny 8847, respektíve 8019. *Hill-Climb* bola z týchto dvoch stratégií rýchlejšia (výpočet jej trval 0,07 sekundy). Najlepší medián cien riešení mala *Simulated annealing*, konkrétne 8783,5. Jej negatívom však je dlhý výpočetný čas, ktorý trval v priemere až 2793,12 sekúnd, čo je približne 46 minút. Počet riešení v tomto datasete mohol dosahovať až hodnotu 30!, teda $2{,}65 \cdot 10^{32}$.

6.2.5 Test č.5

Test č.5 bol prevedený na grafe o veľkosti 40 uzlov. Dataset reprezentujúci tento graf sa nachádza v textovom súbore s názvom *data_40.txt*.

Z výsledkov piateho testu (tab. 7) môžeme vidieť pokračujúci trend exponenciálneho rastu počtu možných riešení. Tento rast spôsobil takmer u všetkých stratégií problémy nájsť riešenia aspoň podobné tomu, ktoré našli stratégie *Hill-Climb* a *Simulated annealing*. Zložitý mechanizmus v algoritme *PSO* spôsobil, že jeho priemerný výpočetný čas zabral 6988,07 sekúnd, čo sú takmer dve hodiny a priemerný výpočetný čas stratégie *Simulated annealing* trval až vyše troch hodín. Najlepšie riešenie našla stratégia *Simulated annealing*. Malo hodnotu 9076 a medián

cien riešení tejto stratégie mal hodnotu 9307,5. Na tomto konkrétnom datasete sa už veľmi jasne ukazuje, ako stúpa zložitosť problému s počtom uzlov. Počet riešení v tomto datasete mohol dosahovať až hodnotu 40!, teda $8{,}16 \cdot 10^{47}$ možných riešení.

Tab. 7: Výsledky testu č.5.

Stratégia	**Aritm. priem. cien**	**Medián cien**	**Max. cena**	**Min. cena**	**Aritm. priem. časov**
Hill-Climb	10606	10606	10606	10606	0,26
Random search	16409,68	16538	17469	15259	379,94
Sim. annealing	9340,83	9307,5	9651	9076	11709,92
Tabu search	15347,51	15537	16292	14475	1096,43
PSO	14426,4	14372	16114	13208	6988,07

6.2.6 Test č.6

Test č.6 bol prevedený na grafe o veľkosti 50 uzlov. Dataset reprezentujúci tento graf sa nachádza v textovom súbore s názvom *data_50.txt*.

Tab. 8: Výsledky testu č.6.

Stratégia	**Aritm. priem. cien**	**Medián cien**	**Max. cena**	**Min. cena**	**Aritm. priem. časov**
Hill-Climb	9209	9209	9209	9209	0,72
Random search	20694,56	20711	22313	19551	855,34
Sim. annealing	9887,88	9831	10666	9195	25740,05
Tabu search	19498,86	19783	20121	17916	2377,62
PSO	18345,33	18413	20124	16643	41847,32

Po poslednom teste (tab. 8) znova pokračoval nastolený trend, ktorý vznikol už pri 30-uzlovom grafe. Stratégie začali nachádzať veľmi zlé výsledky, čo značí, že sa im už nedarilo zvládať obrovský počet možných riešení. Časová náročnosť rapídne stúpla u všetkých stratégií a u stratégie *PSO* trval priemerný výpočet takmer 12 hodín. Všetky metódy okrem *Hill-Climb* a *Simulated annealing* sa v tomto teste ukázali ako neefektívne. Stratégia *Hill-Climb* našla najlepšie riešenie s cenou trasy 9209 za čas 0,72 sekundy a *Simulated annealing* našla najlepšie riešenie s hodnotou 9195 a jej

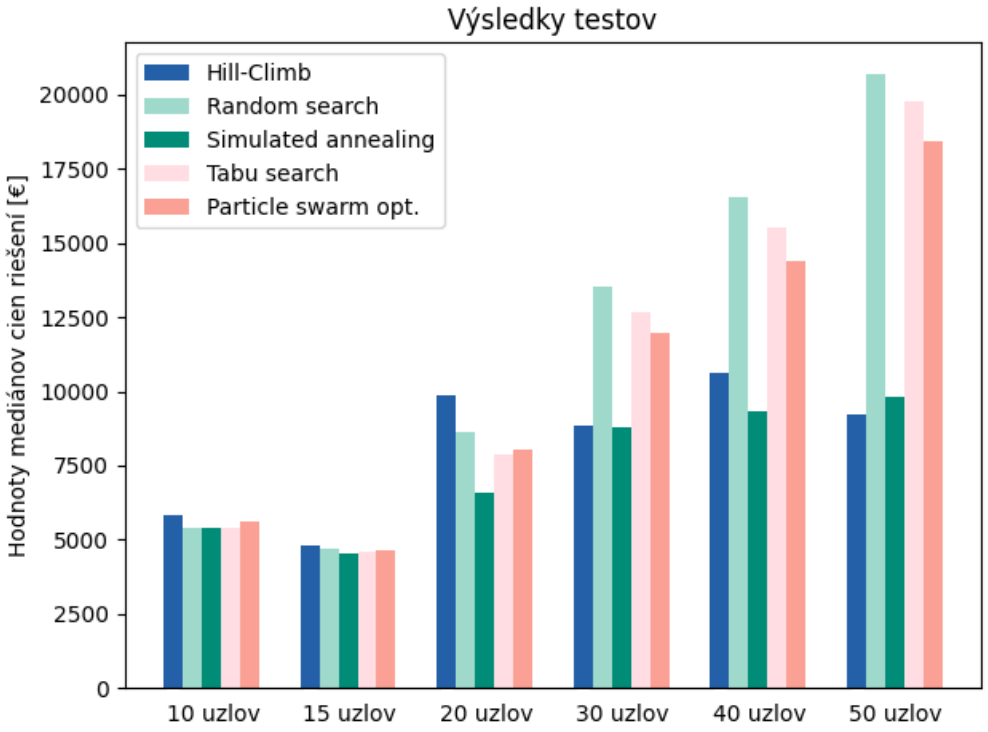
medián cien bol 9831. Počet riešení v tomto datasete mohol dosahovať až hodnotu 50!, teda $3{,}04 \cdot 10^{64}$ možných riešení.

6.3 Celkový sumár testov

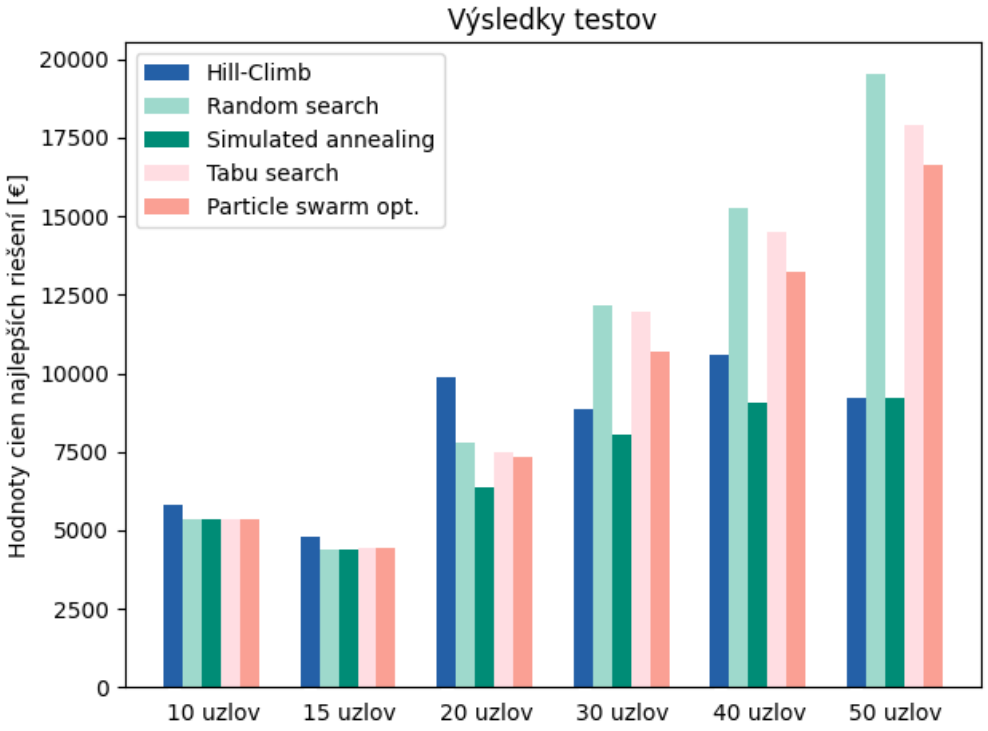
Pri menších datasetoch (veľkosť do 20 uzlov) sa pomerne dobre darilo stratégiam *PSO*, *Tabu search*, *Random search* a *Simulated annealing*, teda všetkým okrem *Hill-Climb*. Ceny ich riešení boli prijateľné. Najlepšie mediánové hodnoty cien mala *Simulated annealing*, ktorej silnou stránkou bolo aj nachádzanie najlepších riešení. Jej výpočet bol však časovo pomerne náročný. Čo sa týka väčších testov (30 a viac uzlov), najlepšie výsledky dosahovali *Simulated annealing* a *Hill-Climb*. Tieto stratégie boli jediné, ktoré pri väčších datasetoch dokázali ponúknuť dobré riešenia. Výhodou stratégie *Hill-Climb* bola jej rýchlosť, keď žiadna hodnota priemerného času potrebného na výpočet nepresiahla jednu sekundu. Lepších výsledkov u niektorých stratégií by bolo dosiahnuté lepším naladením parametrov, napríklad pri *PSO* by sa lepšie výsledky dosiahli početnejšou generáciou častíc. Sumár výsledkov môžeme vidieť aj na grafe (obr. 15), ktorý obsahuje mediánové hodnoty cien riešení a na grafe (obr. 16), ktorý obsahuje hodnoty najlepších cien riešení stratégií.

Pri našich testoch sa potvrdilo, ako zložitý problém TSP-TW naozaj je. Exponenciálny rast počtu možných riešení s pribúdajúcim počtom uzlov spôsobil problémy všetkým stratégiám. V celkovom hodnotení môžeme teda stratégie zhodnotiť nasledovne: najúspešnejšou stratégiou pri riešení menších problémov bola stratégia *Simulated annealing* a najúspešnejšie stratégie pri riešení väčších problémov boli stratégie *Hill-Climb* a *Simulated annealing*. Druhá menovaná mala o niečo lepšie výsledky, avšak *Hill-Climb* bola výrazne rýchlejšia. Naopak, najmenej úspešnou stratégiou pri menších problémoch bola stratégia *Hill-Climb* a pri väčších problémoch to bola stratégia *PSO*, ktorej priemerné výpočetné časy nekorešpondovali s kvalitou nájdených riešení. Stručne môžeme úspešnosť stratégií popísať nasledovne:

- Menšie datasety (10, 15, 20 uzlov)
  - najúspešnejšia stratégia: *Simulated annealing*
  - najmenej úspešná stratégia: *Hill-Climb*
- Väčšie datasety (30, 40, 50 uzlov)
  - najúspešnejšie stratégie: *Simulated annealing, Hill-Climb*
  - najmenej úspešná stratégia: *PSO*



Obr. 15: Stĺpcový graf s hodnotami mediánov cien riešení stratégií.



Obr. 16: Stĺpcový graf s hodnotami najlepších cien riešení stratégií.

Záver

V tejto práci sme sa zaoberali Problémom obchodného cestujúceho s časovými oknami (TSP-TW). Problém spočíva v tom, že obchodný cestujúci musí prejsť každým definovaným miestom práve raz a nakoniec sa vrátiť do pôvodného miesta za čo najnižšiu cenu. Časové okná v tomto probléme definujú určité časové úseky, v ktorých môže byť konkrétne miesto navštívené. Tiež sa môže stať, že v určitom časovom úseku nebude existovať trasa medzi niektorými miestami.

Práca sa postupne zaoberala úvodným popisom TSP a TSP-TW a ďalej boli popísané aj rôzne iné podobné podproblémy. Ďalej boli popísané stratégie, ktorými možno TSP a TSP-TW riešiť. Nasledoval popis priamo implementovaných stratégií aj s popisom modifikácií, ktoré bolo nutné kvôli TSP-TW vykonať. Ďalšiu kapitolu tvoril celkový popis implementácie a aplikácie TSP-TW solver vytvorenej v programovacom jazyku Python. Na záver boli zhodnotené výsledky testov jednotlivých stratégií. Efektivita stratégií sa merala najmä dvoma veličinami. Išlo o medián cien riešenia a najlepšie riešenie nájdené danou stratégiou (teda o riešenie s minimálnou hodnotou ceny). V krajných prípadoch už zavážil aj priemerný čas potrebný na nájdenie riešenia. Celkovo bolo vykonaných 6 testov, ktoré boli rozdelené na dve skupiny, teda na jednoduchšie problémy – 10, 15 a 20-uzlové grafy a zložitejšie problémy – 30, 40 a 50-uzlové grafy. Pri jednoduchších problémoch sa najviac darilo stratégii Simulated annealing a naopak najmenej efektívna bola stratégia Hill-Climb. Pri zložitejších problémoch sa zase najviac darilo stratégiám Simulated annealing a Hill-Climb. Najmenej efektívna bola stratégia PSO. Lepšie výsledky stratégií by mohli byť dosiahnuté voľbou vhodnejších parametrov stratégií.

7 LITERATÚRA

[1] KONA, H., BURDE, A.: *A Review of Travelling Salesman Problem with Time Window Constraint.* IJIRST – International Journal for Innovative Research in Science and Technology, 2015. 4 s.

[2] PICARD, J., C., QUEYRANNE, M.: *The Time-Dependent Traveling Salesman Problem and Its Application to the Tardiness Problem in One-Machine Scheduling*, Operations research, 1978. 86 - 110 s.

[3] DESROISERS, J., DUMAS Y., MARIUS, M., SOUMIS, F.: *Time Constrained Routing and Scheduling* Handbooks in Operations Research and Management Science, Vol 8, 1995. 35-139 s.

[4] SVESTKA, J., A., HUCKFELDT V., E.: *Computational Experience with an M-Salesman Traveling Salesman Algorithm* Management Science, 1973. 790-799 s.

[5] GOLDEN, B., L., ASSAD A., A., WASIL, E., A.: *Routing vehicles in the real world: Applications in the solid waste, beverage, food, dairy, and newspaper industries.* Philadelphia: Society for Industrial and Applied Mathematics, 2001, 245–286 s.

[6] FRONITA, M., GERNOWO, R., GUNAWAN, V.: *Comparison of Genetic Algorithm and Hill Climbing for Shortest Path Optimization Mapping* ICENIS, 2018. 1-5 s.

[7] Best First Search Algorithm in AI | Concept, Implementation, Advantages, Disadvantages, [online] [cit. 24. 2. 2021]
URL: `https://www.mygreatlearning.com/blog/best-first-search-bfs/`

[8] Introduction to A* algorithm, [online] [cit. 25. 2. 2021]
URL: `https://theory.stanford.edu/~amitp/GameProgramming/index.html`

[9] CUDA Accelerated 2-OPT Local Search for the Traveling Salesman Problem., [online] [cit. 26. 2. 2021]
URL: `https://www.intechopen.com/books/novel-trends-in-the-traveling-salesman-problem/cuda-accelerated-2-opt-local-search-for-the-traveling-salesman-problem`

[10] ZHAN, S., LIN, J., ZHANG, Z., ZHONG, Y.: *List-Based Simulated Annealing Algorithm for Traveling Salesman Problem* 2016. 1-13 s.

[11] OHLMANN, J., W., THOMAS, B., W.: *A Compressed-Annealing Heuristic for the Traveling Salesman Problem with Time Windows*, INFORMS Journal on Computing 19, 2007. 80-90 s., doi: https://doi.org/10.1287/ijoc.1050.0145

[12] ZHOU, K., WAN, W., CHEN, X., SHAO, Z., BIEGLER, L.T.: *A Parallel Method with Hybrid Algorithms for Mixed Integer Nonlinear Programming*, IEEE Computational Intelligence Magazine – vol 1., 2006. 28-39 s.

[13] HO, S., C., HAUGLAND, D.: *A tabu search heuristic for the vehicle routing problem with time windows and split deliveries*, Computers & Operations Research – Vol. 31, 2004. 1947-1964 s., ISSN 0305-0548

[14] DVOŘÁK, J., BŘEZINA, T.: *Algoritmy umělé inteligence – lekce 10*, MODERNIZACE VÝUKOVÝCH MATERIÁLŮ A DIDAKTICKÝCH METOD - CZ.1.07/2.2.00/15.0463 , Brno, 5-11 s.

[15] DVOŘÁK, J., BŘEZINA, T.: *Algoritmy umělé inteligence – lekce 3*, MODERNIZACE VÝUKOVÝCH MATERIÁLŮ A DIDAKTICKÝCH METOD - CZ.1.07/2.2.00/15.0463 , Brno, 3-8 s.

[16] DVOŘÁK, J., BŘEZINA, T.: *Algoritmy umělé inteligence – lekce 2*, MODERNIZACE VÝUKOVÝCH MATERIÁLŮ A DIDAKTICKÝCH METOD - CZ.1.07/2.2.00/15.0463 , Brno, 11 s.

[17] DORIGO, M., BIRATTARI, M., STUTZLE, T.: *Ant colony optimization*, ESCAPE, 2013. 272-273 s.

[18] YUANYUAN, L., JING, Z.: *An Application of Ant Colony Optimization Algorithm in TSP*, Fifth International Conference on Intelligent Networks and Intelligent Systems, Tianjin, China, 2012. 61 - 64 s.

[19] CHENG, CH., B., MAO, CH., P.: *A modified ant colony system for solving the travelling salesmanproblem with time windows*, Mathematical and Computer Modelling, Vol. 46, 2007. 1225 - 1235 s., ISSN 0895-7177,

[20] KENNEDY, J., EBERHART, R.: *Particle swarm optimization*, IEEE International Conference on Neural Networks Proceedings, 1995. 6 s.

[21] GHARIN, A., BENHRA, J., CHAOUQI, M.: *A performance comparison of PSO and GA applied to TSP*, International Journal of Computer Applications - vol 130 - no.15, 2015. 34 s.

[22] Ant Colony Optimization Algorithm processes, [online] [cit. 18. 5. 2021] URL: `https://figshare.com/articles/figure/_Ant_Colony_Optimization_Algorithm_processes_/1418788/1`

[23] WANG, K.-P., HUANG, L., ZHOU, C.-G., PANG, W.: *Particle swarm optimization for traveling salesman problem*, Proceedings of the 2003 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.03EX693), 2003. 1583 - 1585 s.

[24] CHENG, W., MAIMAI, Z., JIAN, L.: *Solving traveling salesman problems with time windows by genetic particle swarm optimization*, IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence), 2008. 1752 - 1755 s., doi: 10.1109/CEC.2008.4631026

[25] VEHICLE ROUTING PROBLEM, [online] [cit. 28. 2. 2021]
URL: `https://logisticsmgepsupv.wordpress.com/2015/03/24/vehicle-routing-problem/`

[26] WANG, Y.: *Improving Artificial Bee Colony and Particle Swarm Optimization to Solve TSP Problem*, International Conference on Virtual Reality and Intelligent Systems (ICVRIS), China, 2018. 179 - 182 s.

[27] KARABULUT, K., TASGETIREN, M., F.: *A Discrete Artificial Bee Colony Algorithm for the Traveling Salesman Problem with Time Windows* , WCCI 2012 IEEE World Congress on Computational Intelligence , Brisbane, Australia, 2012.

[28] github: kiwicom/travelling-salesman, [online] [cit. 1. 3. 2021]
URL: `https://github.com/kiwicom/travelling-salesman`

[29] Travelling Salesman Challenge – Recap, [online] [cit. 11. 3. 2021]
URL: `https://code.kiwi.com/travelling-salesman-challenge-recap-7956f433bc10`

[30] Travelling Salesman Challenge 2.0 wrap-up, [online] [cit. 11. 3. 2021]
URL: `https://code.kiwi.com/travelling-salesman-challenge-2-0-wrap-up-cb4d81e36d5b`

[31] Traveling with a Quantum Salesman, [online] [cit. 17. 3. 2021]
URL: `https://physics.aps.org/articles/v10/s32`

[32] WANG, L., ZHANG, J., LI, H.: *An Improved Genetic Algorithm for TSP*, International Conference on Machine Learning and Cybernetics, Hong Kong, China, 2007, 925 - 928 s.

[33] Search Algorithm Series: PSO, [online] [cit. 18. 5. 2021]
URL: `https://medium.com/@iamterryclark/swarm-intelli-eb5e46eda0c3`

[34] YULIASTUTI, G., E., MAHMUDY, W., F., RIZKI, A., M.: *Implementation of Genetic Algorithm to Solve Travelling Salesman Problem with Time Window (TSP-TW) for Scheduling Tourist Destinations in Malang City*, Journal of Information Technology and Computer Science 2, 2017, doi: 10.25126/jitecs.20172122

[35] IATA Airline and Location Codes, [online] [cit. 19. 4. 2021]
URL: https://www.iata.org/en/services/codes/

[36] Basic AI Algorithms, [online] [cit. 23. 4. 2021]
URL: https://towardsdatascience.com/basic-ai-algorithms-a7607b9ecdce

[37] General Python FAQ, [online] [cit. 23. 4. 2021]
URL: https://docs.python.org/3/faq/general.html#what-is-python

[38] PRINTABLE US BLANK MAP, [online] [cit. 24. 4. 2021]
URL: http://www.clipartbest.com/printable-us-blank-map

[39] VEHICLE ROUTING PROBLEM, [online] [cit. 24. 4. 2021]
URL: https://logisticsmgepsupv.wordpress.com/2015/03/24/vehicle-routing-problem

[40] Difference between Depth First Search and Breadth First Search, [online] [cit. 24. 4. 2021]
URL: https://dev.to/danimal92/difference-between-depth-first-search-and-breadth-first-search-6om

[41] Hill Climbing Algorithm in Artificial Intelligence, [online] [cit. 24. 4. 2021]
URL: https://www.javatpoint.com/hill-climbing-algorithm-in-ai

[42] 2-opt, [online] [cit. 24. 4. 2021]
URL: https://en.wikipedia.org/wiki/2-opt

[43] Genetic Algorithm, [online] [cit. 24. 4. 2021]
URL: https://tutorials.retopall.com/index.php/2019/03/01/genetic-algorithm

[44] NetworkX, [online] [cit. 24. 4. 2021]
URL: https://networkx.org

[45] Matplotlib: Visualization with Python, [online] [cit. 24. 4. 2021]
URL: https://matplotlib.org

[46] Airport database, [online] [cit. 24. 4. 2021]
URL: https://openflights.org/data.html

Zoznam skratiek

TSP	Travelling salesman problem - Problém obchodného cestujúceho
TSP-TW	Travelling salesman problem with time windows - Problém obchodného cestujúceho s časovými oknami
TD-TSP	Time dependent travelling salesman problem - Časovo závislý problém obchodného cestujúceho
m-TSP	Multiple travelling salesman problem - Problém obchodného cestujúceho s viacerými cestujúcimi
VRP	Vehicle routing problem - Rozvozný problém
VRP-TW	Vehicle routing problem with time windows - Rozvozný problém s časovými oknami
SA	Simulated Annealing - Simulované žihanie
TS	Tabu Search - Zakázané hľadanie
GA	Genetic algorithms - Genetické algoritmy
ACO	Ant Colony Optimization - Optimalizácia kolóniou mravcov
PSO	Particle Swarm Optimization - Optimalizácia rojom častíc
ABC	Artificial Bee Colony - Kolónia umelého včelstva
IATA	International Air Transport Association - Medzinárodné združenie leteckých prepravcov