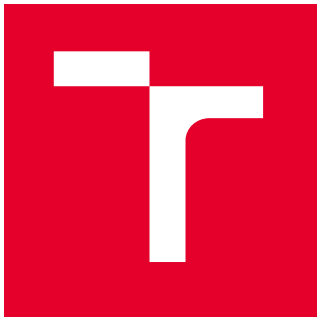


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ

FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMATIZACE A INFORMATIKY

INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

NÁVRH OPTIMÁLNÍ TURISTICKÉ TRASY

TOURIST TRIP DESIGN PROBLEM

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE

AUTHOR

Bc. Tomáš Benda

VEDOUCÍ PRÁCE

SUPERVISOR

Ing. Jakub Kůdela, Ph.D.

BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav automatizace a informatiky
Student:	**Bc. Tomáš Benda**
Studijní program:	Aplikovaná informatika a řízení
Studijní obor:	bez specializace
Vedoucí práce:	**Ing. Jakub Kůdela, Ph.D.**
Akademický rok:	2021/22

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:

Návrh optimální turistické trasy

Stručná charakteristika problematiky úkolu:

Pro plánování turistických tras je kromě výběru vhodných míst také důležitým parametrem to, že posloupnost zvolených míst tvoří praktickou trasu. Diplomová práce se bude zabývat plánováním optimální turistické trasy dle osobních preferencí a omezujících podmínek, jako je například její časová nebo finanční náročnost. Jako podklad práce poslouží vhodné mapové API.

Cíle diplomové práce:

Popsání a rozbor návrhu optimální turistické trasy.
Rešerše používaných přístupů a algoritmů.
Implementace vybraného algoritmu.

Seznam doporučené literatury:

WORNDL, W., HEFELE, A., HERZOG, D. Recommending a sequence of interesting places for tourist trips. Information Technology & Tourism. 2017, 17, 31-54.

GAVALAS, D., et al. A survey on algorithmic approaches for solving tourist trip design problems. Journal of Heuristics. 2014, 20, 291-328.

TLILI, T., KRICHEN, S. A simulated annealing-based recommender system for solving the tourist trip design problem. Expert Systems With Applications. 2021, 186, 115723.

ABSTRAKT

Tato diplomová práce se zabývá návrhem optimální turistické trasy. Jedná se o netriviální optimalizační problém, jehož výsledkem je personalizovaná turistická trasa. V praktické části se implementují vybrané algoritmy řešící problém návrhu optimální turistické trasy na reálných datech. Implementace vybraných algoritmů jsou využity při tvorbě mobilní aplikace.

ABSTRACT

This master's thesis deals with the Tourist Trip Design Problem. Designing a tourist route is non-trivial optimization problem which results in personalised tourist tour. The algorithms which solve the Tourist Trip Design Problem are implemented to work with real world data. These algorithms were used during development of mobile application.

KLÍČOVÁ SLOVA

Návrh optimální turistické trasy, team orienteering problem with time windows, iterované lokální prohledávání, ant colony system.

KEYWORDS

Tourist Trip Design Problem, team orienteering problem with time windows, iterated local search, ant colony system.

ÚSTAV AUTOMATIZACE A INFORMATIKY

2022

BIBLIOGRAFICKÁ CITACE

BENDA, Tomáš. *Návrh optimální turistické trasy.* Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky, 2022, 73 s. Diplomová práce. Vedoucí práce: Ing. Jakub Kůdela, PhD.

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že tato práce je mým původním dílem, vypracoval jsem ji samostatně pod vedením vedoucího práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury.

Jako autor uvedené práce dále prohlašuji, že v souvislosti s vytvořením této práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků.

V Brně dne 20. 5. 2022 .....

Tomáš Benda

PODĚKOVÁNÍ

Děkuji mnohokrát vedoucímu této práce, Ing. Jakubu Kůdelovi Ph.D. za jeho odborné rady a věnovaný čas při vedení této práce.

OBSAH

1 ÚVOD

Návrh optimální turistické trasy je problém, který turista řeší při návštěvě nové lokality. Najít informace o zajímavých místech v dané lokalitě nebývá problém. Existují služby jako Google Places, TripAdvisor či Foursquare, které slouží jako dobrý zdroj o místech zájmu. Tyto služby navíc nabízejí možnost svým uživatelům ohodnotit navštívená místa podle svých zkušeností a turistovi, který se snaží sestavit plán výletu poskytují náhled na to, která místa jsou zajímavá mezi uživateli.

Ačkoliv turista získá informace o místech zájmu, musí řešit problém, jaká místa si vybrat a jak sestavit logický plán jejich návštěvy. Zde vstupují do problému další předpoklady a omezení. Návrh turistické trasy musí dozajista splňovat určitá časová omezení. Turista chce například strávit procházením památek pouze určitý čas dne. Místa zájmu, ať už to jsou muzea, kostely či hrady mají různou otevírací dobu. V České republice jsou mnohá místa během sezony uzavřeny v pondělky, turista tak musí pracně dávat dohromady informace o otevíracích dobách. Z dalších podmínek lze uvést například daný peněžní rozpočet pro výlet.

Návrh optimální turistické trasy má za úkol řešit výše zmíněné potíže. Výsledkem návrhu by měla být logická posloupnost návštěvy míst zájmu, které neporušují zejména časová omezení nebo omezení ve formě peněžního rozpočtu. Návštěva míst by navíc měla být personalizovaná a maximalizovat užitek turisty, tj. návrh trasy by měl nabízet místa, která jsou pro danou osobu zajímavá.

V práci jsou zmíněné modely, podle kterých lze na návrh optimální turistické trasy nahlížet (viz kapitola 3). Návrh turistické trasy můžeme jednoduše rozdělit na návrh turistické trasy pro jeden či více dnů. Pro obě varianty jsou zmíněné optimalizační problémy, které se přibližují návrhu optimální turistické trasy.

V kapitole 4 jsou připomenuta již existující vytvořená řešení návrhu optimální turistické trasy. Jedná se zejména o aplikace demonstrující návrh optimální turistické trasy pracující s reálnými daty. Rozebrány jsou různé algoritmy, které řeší různá pojetí tohoto optimalizačního problému. Jmenovitě jsou zmíněny algoritmy GRASP, KSA, rozšířený Dijkstrův algoritmus, Iterované lokální prohledávání (ILS) a jeho rozšíření o simulované žíhání (SAILS) a použití hejnové inteligence Ant Colony System (ACS) pro model Team Orienteering Problem with Time Windows (TOPTW).

V praktické části byly implementovány alogritmy, které řeší návrh optimální turistické trasy. Algoritmy, které byly implementovány jsou ILS, SAILS a vlastní návrh Ant Colony Sytem. V rámci testování byl vyzkoušeno rozšíření algoritmu SAILS o vytvoření počátečního řešení algoritmem ACS.

Výsledkem práce je mobilní aplikace, která na základě uživatelských dat sestaví návrh turistické trasy, která maximalizuje užitek turisty dle jeho preferencí

(viz kapitola 7). Vytvořená aplikace má reálné využití pro turistu, jenž potřebuje vytvořit posloupnost návštěvy zajímavých míst ve zvoleném městě, když cestuje tzv. po vlastní ose. Výsledná trasa by kromě maximalizace užitku turisty měla zajistit dodržení časových oken míst zájmu ve formě otevírací doby.

V závěru jsou zhodnoceny výsledky práce. Algoritmy byly testovány na konkrétním případě sestavení návrhu turistické trasy ve městě Brno. Dále krátce zhodnocena vhodnost využití použitých prostředků pro tvorbu aplikace.

2 OPTIMALIZACE A GRAFY

Návrh optimální turistické trasy je optimalizační problém, proto je vhodné nejdříve vzpomenout, jak optimalizační úloha obecně vypadá. Jelikož se v návrhu trasy pracuje s místy zájmu a vzdálenosti mezi nimi, které lze reprezentovat pomocí grafu, je nutné nejdříve pojem graf zavést. V neposlední řadě se tato kapitola zabývá problémem obchodního cestujícího, jelikož návrh optimální turistické trasy je možné modelovat některými generalizacemi tohoto problému (viz refkap:ttdp).

2.1 Optimalizace

Základní úlohou optimalizace je minimalizace popřípadě maximalizace účelové funkce za podmínek danými omezujícími podmínkami. Pro vytvoření optimalizačního modelu je nutné určit rozhodovací proměnné, tedy proměnné, které reprezentují rozhodnutí, jež by se měla provést.

Jestliže je určena účelová funkce problému, rozhodovací proměnné a omezující podmínky, je možné určit model. Standardní model vypadá dle (1). Funkce $f, g_1, ..., g_m$ jsou dané funkce rozhodovacích proměnných $x_1, ..., x_m$ a $b_1, ..., b_m$ jsou konstanty.

Optimalizační modely můžeme rozdělit na lineární a nelineární. Optimalizační model je lineární, jestliže účelová funkce f ve standardním modelu (1) a omezující funkce $g_1, ..., g_m$ jsou lineární (funkce je lineární, pakliže se jedná o součet rozhodovacích proměnných násobených konstantou).

V této práci se předpokládá model celočíselného programování. Rozhodovací proměnné v celočíselném optimalizačním modelu nabývají pouze celočíselné hodnoty. Často touto hodnotou bývá binární hodnota, viz modely návrhu optimální turistické trasy (kapitola 3).

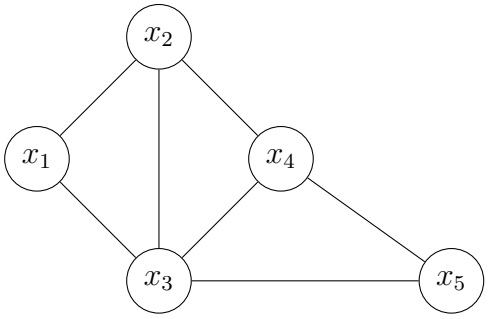
V práci se objevuje také pojem vicekriteriální (konkrétně bikriteriální) optimalizační model. Jedná se o model, kdy cílem je maximalizace či minimimalizace alespoň dvou účelových funkcí (u bikriteriálního modelu to jsou dvě účelové funkce) [10].

$$\begin{aligned} &\max \text{ nebo } \min f(x_1, ..., x_n) \\ &\text{za podmínek } g_i(x_1, ..., x_n) \leq \text{nebo} \geq \text{nebo} = b_i \end{aligned} \tag{1}$$

2.2 Graf

Grafem se v teorii grafů rozumí objekt daný dvojicí množin vrcholů a hran spojující vrcholy. Prostý graf lze tedy zapsat jako $G = (V, H)$, kde V je množina vrcholů a H je množina hran mezi vrcholy [5, 18]. Obecný graf pak zapíšeme jako $G = (V, H, \epsilon)$, kde V je množina vrcholů, H je množina hran a ϵ je zobrazení incidence. Obecný graf byl zaveden z toho důvodu, že mezi dvěma vrcholy z množiny vrcholů V může existovat více hran [18].

Graf znázorňujeme tak, že vrcholy jsou reprezentovány kroužky popřípadě body a hrany jsou čáry mezi vrcholy respektive šipky naznačující směr, pakliže hovoříme o tzv. orientovaném grafu [18]. Znázornění grafu lze vidět na obr. **??**.



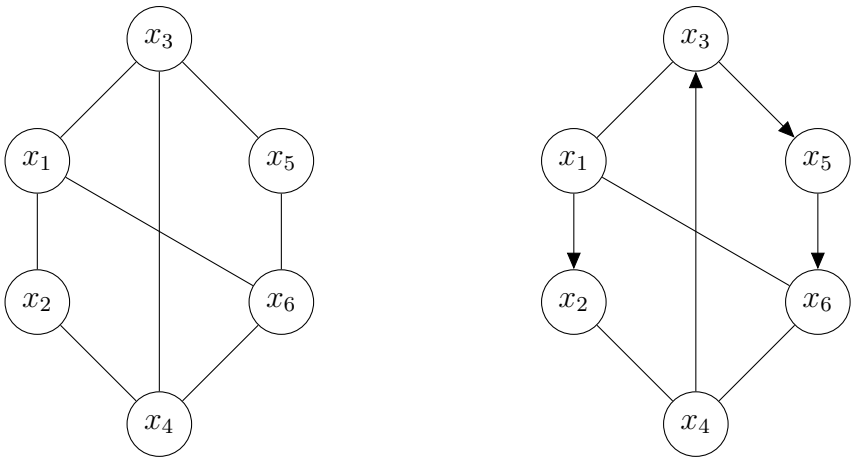
Obr. 1: Zobrazení grafu [vlastní zpracování]

Orientovaný graf je ten graf, kde hrana mezi vrcholy v množině hran H je seřazenou dvojicí vrcholů. Každá hrana má pak danou orientaci [5]. Orientovaný graf je naznačen na obr. 2b). Naopak u neorientovaného grafu na pořadí dvojice vrcholů v množině vrcholů nezáleží. Neorientovaný graf je vidět v obr. 2a.

Jestliže pro všechny vrocholy platí, že každý vrchol z množiny vrcholů V je spojen se všemi ostatními vrcholy, pak mluvíme o úplném grafu. Úplný orientovaný graf (viz obr. 3b) je tedy graf, jehož množina hran V je tvořena všemi uspořádanými dvojicemi vrcholů grafů. Úplný neorientovaný graf je graf, jehož každé dva vrcholy jsou spojeny hranou se všemi ostatními vrcholy grafu (viz obr. 3a) [18].

2.3 Problém obchodního cestujícího

Problém obchodního cestujícího neboli Travelling salesman problem (TSP) je optimalizační úloha, kterou lze popsat následovně. Máme n měst a známe vzdálenosti mezi městy. Cílem řešení problému obchodního cestujícího je nalezení takové okružní cesty, která začíná v jednom městě a postupně projde všechna města právě jednou. Konec cesty je opět v počátečním městě. Jedná se tedy o nalezení Hamiltonovské uzavřené cesty [18] nebo také Hamiltonovské kružnice [15].



(a) Neorientovaný graf (b) Orientovaný graf

Obr. 2: Orientace grafu [vlastní zpracování]

(a) Úplný neorientovaný graf (b) Úplný orientovaný graf

Obr. 3: Úplný orientovaný a neorientovaný graf [vlastní zpracování]

Problém obchodního cestujícího může být symetrický a asymetrický. Jestliže vzdálenost mezi uzlem i a kterýmkoliv dalším uzlem j je stejná, jako vzdálenost, kterou je nutné urazit při přechodu z uzlu j do uzlu i, hovoříme o symetrickém TSP.

Účelová funkce celočíselného modelu lineárního programování symetrického problému obchodního cestujícího lze vyjádřit pomocí (2). Rozhodovací proměnná $x_{ij} = 1$ jestliže uzel i předchází v cestě uzlu j, jestliže ne, pak $x_{ij} = 0$. Omezení (3) zaručuje, že se uzel i objeví v řešení pouze jednou. Omezení (4) vylučuje rozdělení řešení na více menších cest. Podmnožina S je podmnožina uzlů, které je nutné navštívit, přičemž $|S| \geq 3$ [10].

$$\min \sum_{i} \sum_{j>i} d_{ij} x_{ij} \tag{2}$$

$$\sum_{j<i} x_{ji} + \sum_{j>i} x_{ij} = 2 \tag{3}$$

$$\sum_{i \in S} \sum_{j \notin S, j > i} x_{ij} + \sum_{i \notin S} \sum_{j \in S, j > i} x_{ij} \geq 2 \tag{4}$$

Pro řešení problému obchodního cestujícího existují metody, které lze rozdělit na exaktní a heuristické. Mezi exaktní metody lze zařadit metodu celočíselného lineárního programování, metodu hrubé síly nebo metodu větví a mezí (branch-and-bound) [15].

Problém obchodního cestujícího je NP-úplná úloha. To znamená, že složitost exponencionálně roste se zvyšujícím se rozsahem úlohy. Proto jsou exaktní metody vhodné pouze pro malý počet měst.

Pro problém obchodního cestujícího se proto často využívá aproximativních nebo heuristických metod. Mezi heuristické metody můžeme zařadit genetický algoritmus (genetic algorithm GA), simulované žíhání (simulated annealing SA), zakázané prohledávání (tabu-search TS) nebo horolezecký algoritmus (hill climbing HC) [18].

Výhodou heuristických metod oproti exaktním je především znatelně kratší doba výpočtu za cenu nalezení téměř optimálního řešení a nikoliv optimálního řešení. Heuristické metody by měly mít odchylku od optimálního řešení 2-3 % [15].

3 NÁVRH OPTIMÁLNÍ TURISTICKÉ TRASY

Turista, jenž přijede do země či města s cílem navštívit body zájmu (POI) v dané lokalitě, řeší netriviální problém sestavení plánu výletu. Výsledný plán by měl vyhovovat zvoleným zájmům a referencím turisty a zároveň dodržovat zvolená kritéria jako je časové omezení nebo cenový limit výletu. Problematikou nalezení personalisovaného plánu turistické trasy se obecně zabývá Návrh optimální turistické trasy či Tourist Trip Design Problem (TTDP).

Vstupem pro modelování TTDP je zejména seznam bodů zájmů se svými specifickými atributy jako je kategorie daného místa (muzeum, galerie atp.), geografická lokace či otevírací doba. Pro řešení dané úlohy je důležité sestavení grafu a tedy znalost délky cest mezi jednotlivými body zájmu. U jednotlivých bodů zájmu je nutné znát profit daného místa, který turista získá při jeho návštěvě. Z uživatelských dat vstupujících do řešení jsou to zejména počet dnů, které chce turista strávit v dané lokalitě. Tato informace říká, kolik personalisovaných tras je nutné sestavit k uspokojení uživatele. Z časových hledisek, které je třeba brát v úvahu, jsou to dále délka setrvání v daném bodě zájmu a celkový čas turistické trasy pro jednotlivý den [2].

Podle výsledného počtu tras rozdělit TTDP na dvě základní varianty. Pakliže bude výsedná trasa jedna, pak hovoříme o **Single Tour TTDP**. Při dvou a více trasách se jedná o **Multiple Tour TTDP**.

3.1 Single Tour Tourist Trip Design Problem

Single Tour TTDP lze modelovat jako Travelling Salesman Person with Profits (TSPP) respektive dalšími variantami tohoto optimalizačního problému.

TSPP je definován jako bikriteriální generalizace klasického Travelling Salesman Problem (TSP). Úkolem TSPP je nalezení cesty grafem z počátečního uzlu do koncového tak, aby byl maximalizován celkový zisk z navštívených uzlů a aby byla zároveň minimalizována cena cesty (reprezentována délkou trasy nebo časem stráveným na cestě) [13].

TSPP lze rozdělit do tří základních variant na základě toho, jak je přistupováno k maximalizaci profitu při minimalizaci ceny cesty. Jsou jimi Profitable Tour Problem (PTP), Prize Collecting Travelling Salesman Problem (PCTSP) a Orienteering Problem (OP) [2].

3.1.1 Profitable Tour Problem

V optimalizační úloze Profitable Tour Problem je účelovou funkcí maximalizace získaného profitu s odečtením ceny cesty. Proto je nutné, aby jak profit tak cena cesty byly ve stejných jednotkách. Pakliže toto není splněno, je nutné provést konverzi.

PTP lze definovat následovně. Předpokládejme, že máme sadu uzlů $N = 1, 2, ..., |N|$. Každému uzlu přísluší nezáporný profit P_i. Počáteční uzel je 1 a koncový $|N|$. Počáteční uzel se může, ale nemusí shodovat s koncovým uzlem.

Hlavním cílem PTP je nalezení takové cesty grafem, jež maximalizuje získaný profit s odečtenými náklady na cestu, přičemž každý uzel může být navštíven maximálně jednou (s výjimkou počátečního uzlu, pakliže je shodný s koncovým).

PTP je model celočíselného programování s rozhodovacími proměnnými $x_{ij} = 1$, jestliže uzel j je navštíven bezprostředně po uzlu i, jinak $x_{ij} = 0$. Účelová funkce problému je (5). Omezení (6) zajišťuje, že cesta začíná v počátečním uzlu 1 a končí v koncovém uzlu $|N|$. Omezení (7) zaručí celistvost cesty a to, že každý uzel je navštíven maximálně jednou. Omezení popsané v (8) a (9) společně zajišťují, že se řešení nerozdělí do několika menších cest [13].

$$\max \sum_{i=1}^{|N|-1} \sum_{j=2}^{|N|} (P_i x_{ij} - t_{ij} x_{ij}) \tag{5}$$

$$\sum_{j=2}^{|N|} x_{1j} = \sum_{i=1}^{|N|-1} x_{i|N|} = 1 \tag{6}$$

$$\sum_{i=1}^{|N|-1} x_{ik} = \sum_{j=2}^{|N|} x_{kj} \leq 1; \forall k = 2, ..., (|N| - 1) \tag{7}$$

$$2 \leq u_i \leq |N|; \forall i = 2, ..., |N| \tag{8}$$

$$u_i - u_j + 1 \leq (|N| - 1)(1 - x_{ij}); \forall i, j = 2, ..., |N| \tag{9}$$

3.1.2 Prize Collecting Travelling Salesman Problem

Optimalizační úloha Prize Collecting Travelling Salesman Problem (PCTSP) se liší od PTP tím, že účelovou funkcí je minimalizace nákladů na cestu při získání alespoň takového profitu, který je požadován. Profit tedy vstupuje do problému jako omezení, není součástí účelové funkce.

PCTSP lze definovat následujícím způsobem. Mějme sadu uzlů $N = 1, 2, ..., |N|$. a každému uzlu z této sady přiřaďmě nezáporný profit P_i. Počáteční a koncový uzel je určen jako 1 respektive $|N|$. Opět jako u PTP můžeme mít počáteční uzel shodný s koncovým, není to ovšem podmínkou a uzly se mohou lišit.

Řešením úlohy je nalezení takové cesty, která minimalizuje cestovní náklady a zaručí celkový profit větší či rovný minimálnímu požadovanému profitu P_{min}. Každý uzel lze navštívit pouze jednou a profit uzlu lze přičíst k celkovému. Nezápornou cenu trasy mezi uzly i a j předpokládejme t_{ij}.

PCTSP lze formulovat jako model celočíselného programování s rozhodovacími proměnnými $x_{ij} = 1$, jestliže v řešení uzel i bezprostředně předchází uzlu j, jinak $x_{ij} = 0$. Proměnné u_i určují v omezeních (14) respektive (15) pozici navštíveného uzlu v cestě.

Účelová funkce minimalizace nákladů cesty je popsána v (10). Omezení (11) zajišťuje, že cesta začíná v počátečním uzlu 1 a končí v koncovém uzlu N. Celistvost cesty a to, že každý uzel je navštíven maximálně jednou lze popsat pomocí (12). Minimální profit, který je nutné získat během cesty, je dán pomocí (13). Opatření rozdělení řešení na menší cesty zajišťují rovnice (14) a (15).

$$\min \sum_{i=1}^{|N|-1} \sum_{j=2}^{|N|} t_{ij} x_{ij} \tag{10}$$

$$\sum_{j=2}^{|N|} x_{1j} = \sum_{i=1}^{|N|-1} x_{i|N|} = 1 \tag{11}$$

$$\sum_{i=1}^{|N|-1} x_{ik} = \sum_{j=2}^{|N|} x_{kj} \leq 1; \forall k = 2, ..., (|N| - 1) \tag{12}$$

$$\sum_{i=1}^{|N|-1} \sum_{j=2}^{|N|} P_i x_{ij} \geq P_{min} \tag{13}$$

$$2 \leq u_i \leq |N|; \forall i = 2, ..., |N| \tag{14}$$

$$u_i - u_j + 1 \leq (|N| - 1)\,(1 - x_{ij})\,; \forall i, j = 2, ..., |N| \tag{15}$$

3.1.3 Orienteering Problem

Název Orienteering Problem (OP) je odvozen od orientačního běhu, ve kterém je hlavním úkolem soutěžících navštívit co nejvíce kontrolních stanovišť v daném časovém intervalu, přičemž každé stanoviště je ohodnocené a návšťěvou stanovišť se zvyšuje celkové skóre soutěžícího.

Cílem OP je tedy nalezení takové cesty grafem, při které je maximalizovaný zisk z navštívených uzlů přičemž nesmí být porušeno omezení ve formě maximální ceny cesty [2].

Formálně lze OP formulovat následovně. Mějme sadu uzlů $N = 1, 2, ..., |N|$. Každému uzlu $i \in N$ je přiřazen nezáporný užitek P_i. Počáteční a koncový uzel

jsou dány jako 1 respektive $|N|$ přičemž počáteční i koncový uzel může ale nemusí být totožný. Užitek počátečního a koncového uzlu je nulový $P_1 = P_{|N|} = 0$. Řešením problému je nalezení podmnožiny N maximalizující celkový zisk při dodržení omezení ceny cesty T_{max}. Předpokládá se, že každý uzel lze navštívit pouze jednou (výjimkou je pouze počáteční respektive koncový uzel, pakliže jsou tyto uzly totožné). Nezáporná cena cesty mezi uzlem i a j je t_{ij}.

Matematicky lze popsat OP jako model celočíselného programování s rozhodovacími proměnnými $x_{ij} = 1$, jestliže uzel j je navštíven po uzlu i a jinak $x_{ij} = 0$. Účelová funkce je zapsána v (16). Omezení (17) zajišťuje, že cesta začíná v uzlu 1 a končí v uzlu $|N|$. Omezení (18) zaručuje, že každý uzel je navštíven maximálně jednou a zajišťuje celistvost cesty. Celková cena cesty musí být nižší než maximální cena cesty T_{max}, což vyjadřuje (19). Kombinace omezení (20) a (21) zajišťují, že se řešení nerozdělí na více menších cest. Proměnné u_i reprezentují v (20) a (21) pozici navštíveného uzlu v cestě [13].

$$\max \sum_{i=2}^{|N|-1} \sum_{j=2}^{|N|} P_i x_{ij} \tag{16}$$

$$\sum_{j=2}^{|N|} x_{1j} = \sum_{i=1}^{|N|-1} x_{i|N|} = 1 \tag{17}$$

$$\sum_{i=1}^{|N|-1} x_{ik} = \sum_{j=2}^{|N|} x_{kj} \leq 1; \forall k = 2, ..., (|N| - 1) \tag{18}$$

$$\sum_{i=1}^{|N|-1} \sum_{j=2}^{|N|} t_{ij} x_{ij} \leq T_{max} \tag{19}$$

$$2 \leq u_i \leq |N|; \forall i = 2, ..., |N| \tag{20}$$

$$u_i - u_j + 1 \leq (|N| - 1)\,(1 - x_{ij})\,; \forall i, j = 2, ..., |N| \tag{21}$$

3.1.4 Orienteering Problem with Time Windows

Orienteering Problem with Time Windows (OPTW) je varianta OP, kdy každý uzel grafu lze navštívit pouze v časovém okně, které je dané pro daný uzel. Časové okno je dané začátkem služby a nejpozdějším okamžikem začátku služby.

Matematický model je obdobný jako u OP. Máme uzly $N = 1, 2, ..., |N|$ a každý uzel $i \in N$ má nezáporný užitek P_i. Počáteční a koncový uzel uvažujeme 1 respektive $|N|$ a stejně jako u OP může být počáteční i koncový uzel totožný. Pro počáteční a koncový uzel uvažujeme $P_i = 0$. Nadstavbou oproti OP je přiřazení časových oken $[O_i, C_i]$. Uzel může být pak navštíven pouze jednou a jenom v tomto

časovém okně. Nezáporná cena cesty mezi dvěma uzly i a j je t_{ij}. Řešením je podmnožina N maximalizující celkový zisk při dodržení omezení ceny cesty T_{max}. Toto omezení je přiřazené počátečnímu a koncovému uzlu ve formě jejich časových oken $[O_i, T_{max}]$. Doba trvání služby se považuje jako součást ceny cesty mezi uzly.

Rozhodovací proměnné OPTW jsou $x_{ij} = 1$, pakliže je uzel i následován uzlem i a jinak $x_{ij} = 0$, s_i je začátek služby v uzlu i a L je vysoká konstanta. Účelová funkce je (22). Omezení zaručující, že cesta začne v uzlu 1 a skončí v uzlu $|N|$ je (23). Omezení (24) zaručuje, že každý uzel je navštíven maximálně jednou a zároveň je zajištěna spojitost cesty. Omezení (25) a (26) říkají, že začátek služby musí být v rámci časového okna. Poslední omezení (27) zajišťuje správnost časové osy cesty [13].

$$\max \sum_{i=2}^{|N|-1} \sum_{j=2}^{|N|} P_i x_{ij} \tag{22}$$

$$\sum_{j=2}^{|N|} x_{1j} = \sum_{i=1}^{|N|-1} x_{i|N|} = 1 \tag{23}$$

$$\sum_{i=1}^{|N|-1} x_{ik} = \sum_{j=2}^{|N|} x_{kj} \leq 1; \forall k = 2, ..., (|N| - 1) \tag{24}$$

$$O_i \leq s_i; \forall i = 1, ..., |N| \tag{25}$$

$$s_i \leq C_i; \forall i = 1, ..., |N| \tag{26}$$

$$s_i + t_{ij} - s_j \leq L\,(1 - x_{ij})\,; \forall i, j = 1, ..., |N| \tag{27}$$

3.2 Multiple Tour Tourist Trip Design Problem

Návrh optimální turistické trasy pro dvě a více trasy lze modelovat jako varianty tzv. Vehicle Routing Problem with Profits[2]. U této úlohy je předem daný seznam zákazníků s přiděleným profitem získaným při návštěvě daného zákazníka. Cílem úlohy je nalézt předem daný počet cest tak, aby zisk z návštěvy zákazníků byl co nejvyšší, přičemž nemusí být obslouženi všichni zákazníci ale jen někteří[11].

VRPP lze dále rozdělit na **Prize Collecting Vehicle Routing Problem** (PCVRP), ve kterém je účelovou funkcí lineární kombinace minimalizace celkové vzdálenosti cest, minimalizace použitých vozidel a maximalizace získaného profitu. V **Capacitated Profitable Tour Problem** (CPTP) je účelovou funkcí rozdíl mezi celkovým profitem a celkovou délkou tras. Další optimalizační úlohou spadající do

kategorie VRPP je **Vehicle Routing Problem with Profits and deadlines** (VRPP-TD). Účelová funkce je stejná jako u CPTP, rozdíl je ovšem v přidání časových omezení jednotlivým zákazníkům. [2] Další variantou, kterou lze modelovat TTDP, je Team Orienteering Problem (TOP) a varianty této úlohy. V diplomové práci je uvažován tento model.

3.2.1 Team Orienteering Problem

Team Orienteering Problem (TOP) se vyznačuje tím, že cílem je nalezení maximálního užitku ne pro jednu cestu jako u OP, ale pro několik cest. Při plánování turistické trasy je možné si rozdíl představit jako plánování výletu na jeden den (OP) oproti plánování výletu rozloženého do několika dní (TOP).

Obdobně jako u OP je předpoklad sady uzlů $N = 1, 2, ..., |N|$. Každý uzel $i \in N$ má nezáporný užitek P_i. Počáteční uzel je 1 a koncový uzel $|N|$, oba uzly můžou ale nemusí být totožné. Cílem je nalézt M cest tak, aby byl maximální užitek ze všech cest co nevyšší. Každá cesta m je omezená časovým omezením T_{max}. Délka cesty mezi uzlem i a uzlem j je t_{ij} a je nezáporná.

TOP lze popsat jako model celočíselného programování s rozhodovacími proměnnými $x_{ijm} = 1$, pakliže návštěva uzlu i je následována návštěvou uzlu j v cestě m a jestliže uzel j nenásleduje za uzlem j v cestš m, pak $x_{ijm} = 0$. Účelová funkce problému je (28). To, že každá cesta v M začíná v uzlu 1 a končí v uzlu $|N|$ je vyjádřeno omezením (29). Omezení (30) zaručuje, že každý uzel je navštíven maximálně jednou. Celistvost cest je vyjádřena omezením (31). Každá cesta je omezena časem T_{max}, což vyjadřuje omezení (32). Kombinace omezení (33) a (34) zajišťují, že se výsledná cesta nerozdělí na několik menších cest [13].

$$\max \sum_{m=1}^{M} \sum_{i=2}^{|N|-1} P_i y_{im} \tag{28}$$

$$\sum_{m=1}^{M} \sum_{j=2}^{|N|} x_{1jm} = \sum_{m=1}^{M} \sum_{i=1}^{|N|-1} x_{i|N|m} = M \tag{29}$$

$$\sum_{m=1}^{M} y_{km} \leq 1; \forall k = 2, ..., (|N| - 1) \tag{30}$$

$$\sum_{i=1}^{|N|-1} x_{ikm} = \sum_{j=2}^{|N|} x_{kjm} = y_{km}; \forall k = 2, ..., (|N| - 1); \forall m = 1, ..., M \tag{31}$$

$$\sum_{i=1}^{|N|-1} \sum_{j=2}^{|N|} t_{ij} x_{ijm} \leq T_{max}; \forall m = 1, ..., M \tag{32}$$

$$2 \leq u_{im} \leq |N|; \forall i = 2, ..., |N|; \forall m = 1, ..., M \tag{33}$$

$$u_{im} - u_{jm} + 1 \leq (|N| - 1)(1 - x_{ijm}); \forall i, j = 2, ..., |N|; \forall m = 1, ..., M \tag{34}$$

3.2.2 Team Orienteering Problem with Time Windows

Team Orienteering Problem with Time Windows (TOPTW) je rozdílný od TOP přidáním časových oken do problému, což zvyšuje náročnost řešení. Každý uzel v každé cestě může být navštíven pouze v časovém okně specifickém pro každý uzel.

Stejně jako u předchozích modelů mějme uzly $N = 1, 2...|N|$ a ke každému uzlu $i \in N$ přiřaďme nezáporný užitek P_i. Počáteční uzel a koncový uzel definujme jako 1 respektive $|N|$, přičemž počáteční a koncový uzel může a nemusí být stejný. Řešením je nalezení M cest takových, že jejich maximální užitek je co nejvyšší. Každá cesta m je omezená časovým omezením T_{max} a návštěva každého uzlu musí proběhnout během časového okna $[O_i, C_i]$, které je přiřazeno každému uzlu. Počátečnímu respektive koncovému uzlu pak přiřaďme časové okno $[0, T_{max}]$. Délku cesty mezi dvěma uzly i a j předpokládejme nezápornou ve tvaru t_{ij}.

TOPTW je model celočíselného programování s rozhodovacími proměnnými $x_{ijm} = 1$, jestliže uzel j následuje uzel i v cestě m a pakliže ne, tak $x_{ijm} = 0$. Jestliže se uzel i nachází v cestě m, pak $y_{im} = 1$, jinak $y_{im} = 0$. L je velká konstanta. s_{im} je začátek služby v uzlu i cesty m. Účelová funkce problému maximalizovat celkový profit popisuje (35). Každá cesta musí začínat v uzlu 1 a končit v uzlu $|N|$ dle omezení (36). Omezení (37) zajistí, že každý uzel lze navštívit maximálně jednou. Splnění časových omezení a celistvost cest ukazuje kombinace omezení (38) a (39). Celkové trvání cesty nesmí překročit T_{max} dle omezení (39). Kombinace omezení (40) a (41) zaručuje, že každý uzel ve všech cestách je navštíven během svého časového okna [13].

$$\max \sum_{m=1}^{M} \sum_{i=2}^{|N|-1} P_i y_{im} \tag{35}$$

$$\sum_{m=1}^{M} \sum_{j=2}^{|N|} x_{1jm} = \sum_{m=1}^{M} \sum_{i=1}^{|N|-1} x_{i|N|m} = M \tag{36}$$

$$\sum_{m=1}^{M} y_{km} \leq 1; \forall k = 2, ..., (|N| - 1) \tag{37}$$

$$\sum_{i=1}^{|N|-1} x_{ikm} = \sum_{j=2}^{|N|} x_{kjm} = y_{km}; \forall k = 2, ..., (|N| - 1); \forall m = 1, ..., M \tag{38}$$

$$s_{im} + t_{ij} - s_{jm} \leq L\,(1 - x_{ijm})\,; \forall i, j = 1, ..., |N|; \forall m = 1, ..., M \tag{39}$$

$$O_i \leq s_{im}; \forall i = 1, ..., |N|; \forall m = 1, ..., M \tag{40}$$

$$s_{im} \leq C_{im}; \forall i = 1, ..., |N|; \forall m = 1, ..., M \tag{41}$$

3.2.3 Hierarchic Team Orienteering Problem with Time Windows

Hierarchic Team Orienteering Problem with Time Windows (HTOPTW) byl zaveden pro použití algoritmu hejnové inteligence Ant Colony System k řešení TOPTW (viz. 4.4).

Definujme HTOPTW obdobně jako TOPTW. Mějme graf $G = \{V, H\}$. Každému uzlu přiřaďme nezáporným profit P_i. Počáteční uzel je 1, koncový pak $|N|$. Každý uzel má časové okno $[O_i, C_i]$, ve kterém musí proběhnout návštěva. Výsledkem výpočtu HPTOPTW je sada elementárních cest $\mathcal{M} = (m_1, m_2, ..., m_{|\mathcal{M}|})$, kde každá cesta $m_k \in \mathcal{M}$ je definována jako posloupnost uzlů začínajících v počátečním uzlu 1 a končící v koncovém uzlu $|N|$, tak, aby $O_i \leq v_i \leq C_i$, kde v_i je čas návštěvy uzlu i. Účelová funkce problému je pak v (42). L je velká konstanta větší než optimální řešení OPTW na stejném grafu. A je sada orientovaných hran, pro které platí $\forall\{i, j\} \in V, (i, j), (j, i) \in H$ [8].

$$\max \left(\sum_{m_k \in \mathcal{M}, k \leq l} \left(\sum_{(i,j) \in A} P_i x_{ij}^{m_k} \right) + \sum_{m_k \in \mathcal{L}, k > l} \left(L^{l-k} \sum_{(i,j) \in A} P_i x_{ij}^{m_k} \right) \right) \tag{42}$$

4 EXISTUJÍCÍ ŘEŠENÍ NÁVRHU OPTIMÁLNÍ TURISTICKÉ TRASY

V této kapitole bude představeno několik vytvořených systémů pro řešení návrhu optimální turistické trasy. Podoba návrhu optimální turistické trasy není pevně zadána, představená řešení se proto snaží nalézt optimální řešení pro různé modely návrhu optimální trasy.

4.1 City Trip Planner

City Trip Planner je expertní systém, který byl vytvořen pro návrh personalizovaných výletů pro pět měst v Belgii (konkrétně Antwerpy, Brugy, Ghent, Leuven a Mechlin).

Autoři vytvořili databázi s místy zájmu, ve které byly obsaženy informace jako GPS souřadnice, otevírací doba daného místa či typ místa zájmu (hrad, kostel apod.). Zájmová místa byla posléze roztříděna do kategorií.

K tomu, aby byl návrh výletu personalizovaný podle preferencí uživatele, systém potřebuje získat uživatelská data. Uživatel si vybírá město, datum příjezdu a odjezdu. Dále je zvolen startovní a koncový bod. Uživatel má možnost zahrnout přestávku na oběd v průběhu výletu. V neposlední řadě je nutno zvolit kategorie míst, která chce uživatel navštívit.

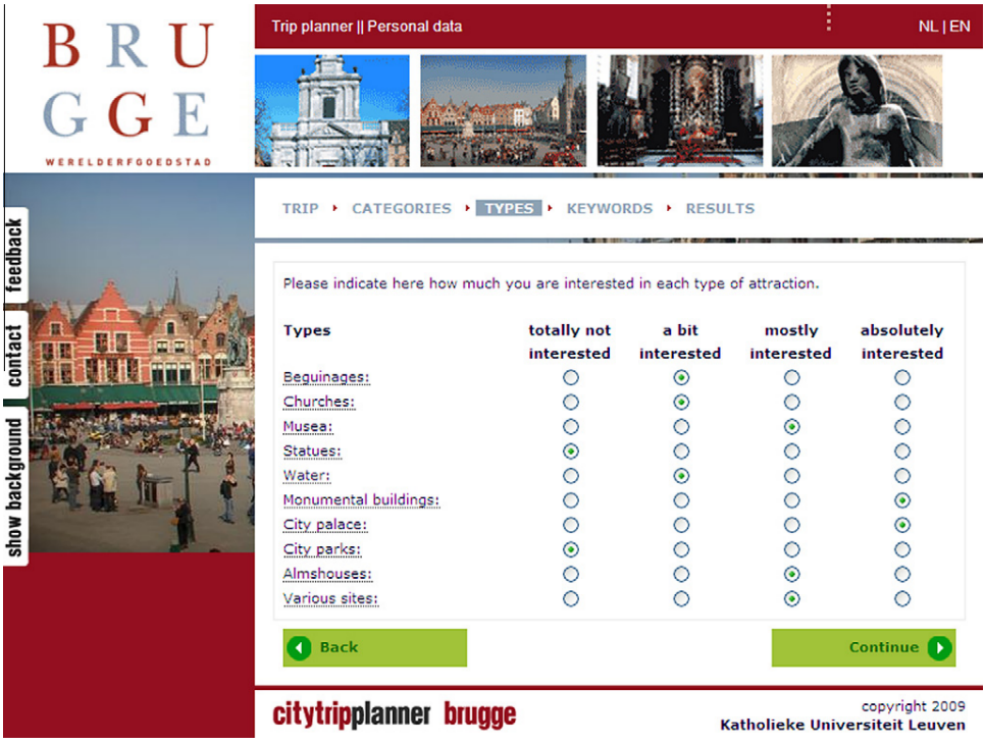
Systém následně vybere a ohodnotí místa zájmu dle osobních preferencí uživatele dle jeho profilu [14].

4.1.1 GRASP algoritmus pro návrh optimální turistické trasy

Algoritmus navržený autory vychází z tzv. Greedy Randomised Adaptive Search Procedure (GRASP). Algoritmus běží v iteracích, dokud nejsou splněny ukončovací podmínky. Na začátku každé iterace je zvolen parametr hladovosti (greediness) rovnoměrně rozložený v intervalu $(0, 1)$. Počáteční řešení se skládá pouze ze startovních a koncových uzlů. Následně se vytvoří seznam kandidátů z míst, která lze navštívit. Dle (43), kde $Shift_j$ je celková časová náročnost vložení uzlu j do řešení a $InterestScore_j^2$ je druhá mocnina ohodnocení daného místa, se spočítá heuristická hodnota pro všechna místa ze seznamu kandidátů.

$$h_j = \left| \frac{InterestScore_j^2}{Shift_j} \right| \tag{43}$$

Rozdílem mezi maximální a minimální heuristickou hodnotou ze seznamu kandidátů spočítáme prahovou hodnotu. Na základě toho, zda jsou heuristické hod-



Obr. 4: City Trip Planner [14]

noty míst ze seznamu kandidátů větší či menší než prahová hodnota se filtrují místa zájmu tak, že místa s větší heuristickou hodnotou než prahová jsou vložena do nového seznamu omezených kandidátů. Ze seznamu omezených kandidátů je náhodně zvoleno jedno místo zájmu. Iterace je ukončena, pakliže je list kandidátů prázdný. Jesliže je aktuálně vytvořené řešení lepší než předchozí nejlepší, pak je toto řešení přijato jako nejlepší.

Autoři aplikují pro zlepšení výkonu operátory $MaxShift$, $Wait$ a $Arrival$. Při kontrole, zda zvolené místo může být vloženo do řešení, dochází k velké časové náročnosti algoritmu při kontrole dodržení všech časových oken následující vložení. Z tohoto důvodu je zaveden operátor $Wait$, který určuje čekací dobu, pakliže je parametr $Arrival$ menší než čas otevření časového okna daného místa (dané místo lze navštívit nejdříve na začátku časového okna a proto je nutné čekat určitou dobu, jestliže je místo navštíveno dříve). $MaxShift$ určuje maximální zpoždění, které je možné aplikovat na řešení, aniž by byla narušena realizovatelnost dané trasy. $MaxShift$ uzlu i je rovna, jesltliže není omezena vlastním časovým oknem, součtu $MaxShift$ a $Wait$ uzlu $i + 1$.

Kvůli integraci přestávky na oběd autoři zavádějí tzv. virtuální místo zájmu. Od klasického místa zájmu se liší tím, že nemá GPS souřadnice a tudíž má nulové vzdálenosti k dalším bodům zájmu [14].

```
Vstup: T_0, α, x, popSize
Výstup: x*
while nebylo dosaženo ukončovacího kritéria do
    řešení = prázdné;
    hladovost = U(0,1);
    seznamNavštívených = GenerováníMožnýchNávštěv(řešení);
    while seznamNavštívených není prázdný do
        foreach navšívený in seznamNavštívených do
            SpočítejHeuristickouHodnotu(navštívený);
        end
        UrčiPrahovouHodnotu();
        omezenýSeznamNavštívených =
         OmezeníSeznamuNavštívených(seznamNavštívených);
        náhodnýNavštívený = random navštívený from
         omezenýSeznamNavštívených;
        řešení = insert náhodnýNavštívený;
        seznamNavštívených = GenerováníMožnýchNávštěv(řešení);
    end
end
return nejlepšíNalezený;
```

Algoritmus 1: GRASP algoritmus pro TTDP

4.2 StayPlan Recommender System

StayPlan Recommender System je systém pro návrh optimální turistické trasy založený na algoritmu KSA, který kombinuje shlukovou analýzu k-means [7] a algoritmus simulovaného žíhání [6].

StayPlan Recommender System se skládá ze tří modulů. Prvním je modul uživatelských preferencí, kde si uživatel mimo jiné volí oblíbené kategorie míst, startovní a koncové místo pro každý den a druh dopravy.

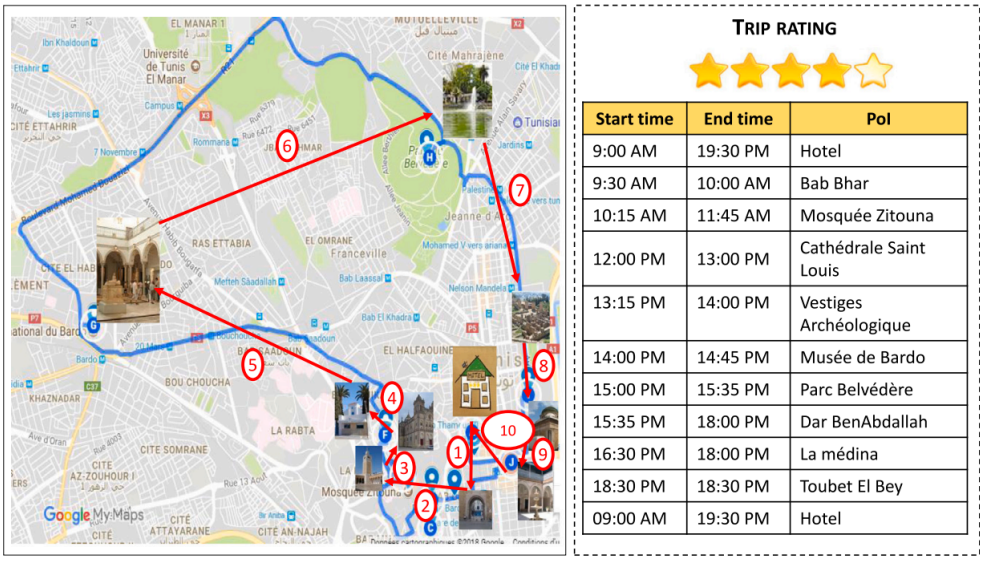
Druhý modul získává seznam zajímavých míst na základě uživatelových preferencí ze služby TripAdvisor. Pro daná místa jsou důležité zejména parametry hodnocení a počet hodnocení, na jejichž základě je dané místo ohodnoceno dle (44).

$$score = rating * \log_2 votesNumber + 1 \tag{44}$$

Třetí modul je modul pro řešení návrhu optimální turistické trasy a optimalizace využívající KSA algoritmus [12].

4.2.1 KSA algoritmus

KSA algoritmus se skládá ze dvou částí. Nejdříve je na sadu nalezených míst zájmu aplikováno k-means shlukování.



Obr. 5: StayPlan Recommender System [12]

K-means je algoritmus shlukové analýzy, který na základě euklidovské vzdálenosti vytvoří obecně m shluků pro sadu bodů, které je potřeba roztřídit.

Algoritmus pracuje tak, že na začátku je vytvořeno m středu shluků s náhodnými souřadnicemi. Pro každý bod ze zadané sady se spočítají vzdálenosti ke všem středům a bod je připojen k tomu středu, ke kterému je vzdalenost nejnižší.

Po přiřazení všech bodů ze zadané sady se provede přepočet souřadnic středů. Nové souřadnice středu se získají tak, že se spočítá těžiště pro body, které patří k danému středu.

Algoritmus pokračuje tak dlouho, dokud nejsou ve dvou po sobě následujících iteracich souřadnice všech středů stejné. Výsledkem je tedy m shluků s rozdělenými body.

U KSA algoritmus se shlukování pomocí k-means provádí tak, že počet dní návštěvy určuje počet shluků m. Vstupní sadou míst, která vstupují do shlukové analýzy je sada míst zájmu, jejichž poloha je dána geografickými souřadnicemi. Pakliže je plánovaný výlet jednodenní, pak shlukovani nemá význam a přechází se rovnou k druhé části algoritmu KSA.

Pro každý shluk bodů zájmu vytvořených pomocí k-means algoritmu se vytvoří optimální posloupnost bodů zájmu pomocí algoritmu simulovaného žíhání tak, aby se maximalizoval užitek uživatele.

Simulované žíhání je biologicky inspirovaný algoritmus, který si bere základní ideu z procesu žíhání oceli. Jedná se o optimalizační metodu prohledávání stavového prostoru. Hlavní výhodou tohoto algoritmu je ten fakt, že s určitou pravděpodobností je přijato k dalšímu prohledávání i aktuálně horší řešení. Díky tomu je možné uniknout lokálnímu minimu.

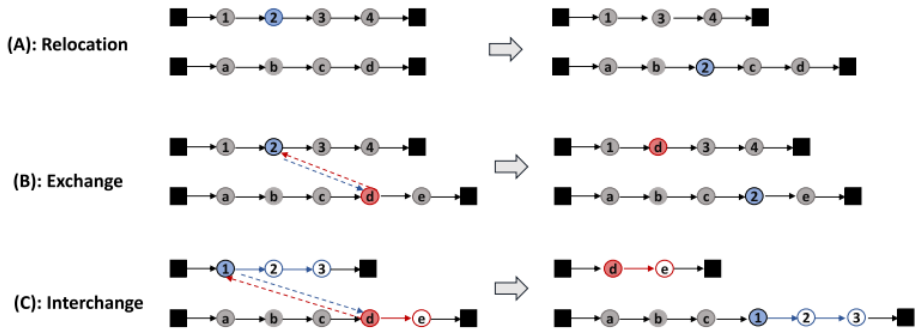
Vstup: $T_0, \alpha, x, popSize$
Výstup: x^*
while *časový limit nebyl překročen* **do**
  for $i = 1$ **to** $popSize$ **do**
    vytvoř r $U(0,1)$;
    if $r \leq \frac{1}{3}$ **then**
      vytvoř nové řešení x^* pomocí operátoru přemístění bodu;
    else
      vytvoř nové řešení x^* pomocí operároru výměny bodu;
    end
    vytvoř nové řešení x^* pomocí operátoru výměny cesty;
    spočítej δT;
    if $\delta T < 0$ **then**
      x^* je nové řešení;
    else
      vytvoř p $U(0,1)$;
    end
    if $p < e^{\frac{\delta T_0}{T}}$ **then**
      x^* je nové řešení;
    end
  end
  $T = \alpha T_0$;
end

Algoritmus 2: Simulované žíhání - KSA algoritmus [12]

Simulované žíhání u algoritumu KSA běží, dokud není dosaženo daného časového limitu. V každé iteraci je získáno náhodně vygenerované číslo r, které je v intervalu $(0, 1)$ a které je použito při výběru **operátoru sousedství**. Pakliže je $r \leq \frac{1}{3}$, nové řešení x^* je získáno pomocí **operátoru přemístění bodu** (point relocation operator). Jestliže je $\frac{1}{3} < r \leq \frac{2}{3}$, pak je nové řešení vytvořeno pomocí **operátoru výměny bodu** (point exchange operator). Při $r > \frac{2}{3}$ je použit **operátor vzájemné výměny cest** (route interchange operator). [12]

Jesliže hodnota účelové funkce $f(x)$ je větší či stejná jako předchozí nejlepší řešení, je řešení x^* přijato jako nové řešení. Jestliže je hodnota účelové funkce $f(x)$ menší, pak je aktuální řešení x^* přijato, jestliže $p < e^{\frac{\delta T_0}{T}}$, kde p je normálně distribuované náhodné číslo v intervalu $(0, 1)$, $\delta = f(x^*) - f(x)$ je rozdíl hodnot účelových funkcí stávajícího řešení a doposud nejlepšího nalezeného řešení, T_0 je parametr představující původní teplotu, T je aktuální teplota.

Na konci iterace dochází ke snížení teploty $T = \alpha T_0$, kde α je parametr ochlazování [12].



Obr. 6: Operátory sousedství [12]

4.3 Webová aplikace používající rozšířený Dijkstrův algoritmus

Autoři aplikace se zaměřili na jiný modelový příklad než u aplikací City Trip Planner a StayPlan Recommender System. Zatímco u zmíněných aplikací si uživatel zvolil počet dní a preference návštěvy míst, u této aplikace využívající rozšířený Dijkstrův algoritmus si uživatel volí startovací a koncové místo výletu ve městě. Aplikace poté dle zvolených preferencí uživatele navrhne seznam míst poblíž trasy mezi startovním a koncovým bodem tak, aby byl celkový užitek uživatele co největší.

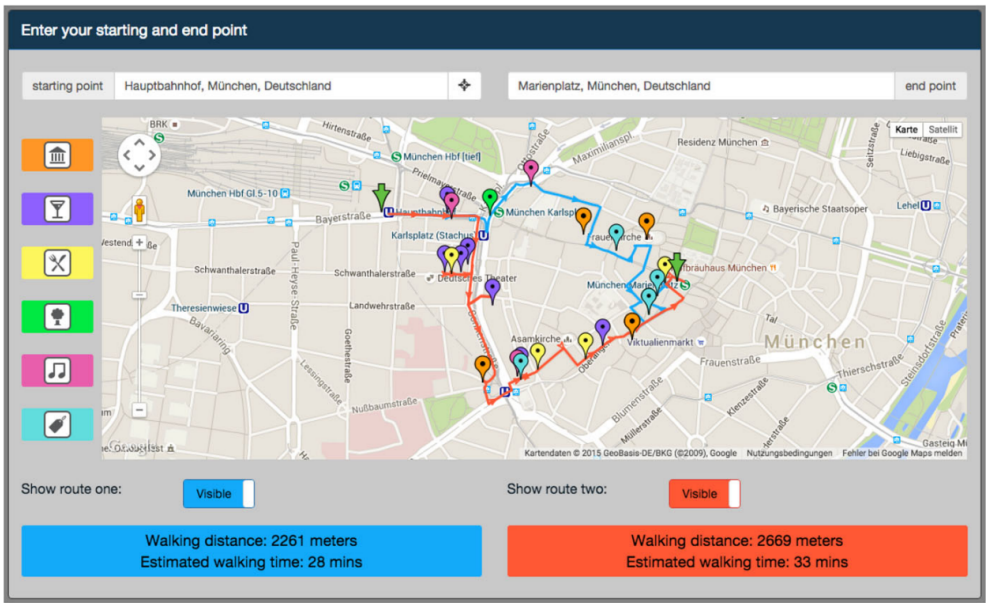
Zdrojem dat pro tuto aplikaci je Foursquare API. Foursquare je služba, ve které uživatelé mohou ohodnotit různé atrakce a zájmová místa na základě svých zkušeností. Vývojáři poté mohou využívat tuto službu jako zdroj dat o místech zájmu.

Uživatel si v aplikaci zvolí, jaké kategorie ho zajímají, tím, že danou kategorii ohodnotí na stupnici od 0-5. Místa zájmu získaná voláním Foursquare API jsou poté ohodnocena na základě příslušnosti ke kategorii, hodnocení a počtu hlasů dle rovnice (45). Lze si povšimnout, že rovnice je téměř totožná s rovnicí (43) pro ohodnocení míst zájmu u StayPlan Recommender System. I zde autoři použili logaritmickou stupnici.

$$score = rating * \log_2 ammountofvotes + 1 \tag{45}$$

Při určení strávené doby na určitém místě autoři volí hodnoty, jejichž základní hodnotu uživatel nemůže ovlivnit. Pro restaurace to je například čas 45 min., pro místa zájmu dalších kategorií pak 60 min. Hodnoty jsou ovšem ovlivněny hodnocením kategorie. Při ohodnocení 3, tedy průměrné hodnoty, je offset základní hodnoty nulový. Při hodnocení 2 je offset -5 min. a u ohodnocení 1 (nejhorší ohodnocení) je offset -15 min. Naopak pro hodnocení 4, tj. lepší než průměrné, je offset +5 min. a pro nejlepší hodnocení 5 je offset +15 min. Autoři argumentují tím, že na místě

kategorie s vyšším ohodnocením uživatel chce trávit více času než na místě s nízkým hodnocením [16].



Obr. 7: Webová aplikace pro návrh turistických tras [16]

4.3.1 Algoritmus bez omezujících podmínek

Tento algoritmus bez omezujících podmínek (constraint-free algorithm) je založený na Dijkstrově algoritmu [1]. Dijkstrův algoritmus je algoritmus pro nalezení nejkratších cest v grafu z počátečního uzlu ke všem ostatním uzlům.

Na začátku algoritmu se vytvoří úplný graf míst s ohodnocenými hranami, kde ohodnocení hrany je vzdálenost mezi danými uzly. Pomocí Dijkstrova algoritmu se následně nalezne nejlepší cesta grafem z počátečního bodu do koncového. Místo minimalizace vzdálenosti je cílem algoritmu maximalizace podílu $\frac{zábava}{vzdálenost}$, kde *zábava* je součet všech ohodnocení míst zájmu na trase. Parametr *vzdálenost* je celková vzdálenost takové trasy, tedy součet vzdálenosti mezi body dané posloupnosti uzlů.

Jestliže je v poměru k ostatním kategoriím více nalezených bodů zájmu u kategorie, kterou uživatel ohodnotil nízko a tudíž místa daně kategorie nechce navštěvovat, může se stát, že maximalizací $\frac{zábava}{vzdálenost}$ vyjde posloupnost s velkým počtem bodů zájmu nežádoucí kategorie. Autoři proto navrhují spočítat Pearsonův korelační koeficient pro datasety uživatelských preferencí kategorií a počtu míst zájmu daně kategorie. Následně se maximalizuje $\frac{r \cdot zábava^2}{vzdálenost^2}$ kde r je příslušný Pearsonův korelační koeficient. Druhé mocniny u *zábava* a *vzdálenost* jsou z toho důvodu, aby se upřednostnily tyto parametry oproti koeficientu r [16].

4.3.2 Algoritmus s omezujícími podmínkami

Obdobně jako předchozí algoritmus bez omezujících podmínek je i algoritmus s omezujícími podmínkami (constraint-based algorithm) založený na Dijkstrově algoritmu pro hledání nejkratší cesty grafem. Rozdíl je, že při dosažení uzlu v grafu se nejdříve zjistí pro každou možnou cestu, jestli přidáním nového uzlu nedojde k porušení časových či cenových omezení zadaných uživatelem. Další podmínkou určenou autory je, že výsledná cesta nesmí mít více než jeden uzel z kategorie noční život popřípadě obsahovat více než jednu restauraci. Pakliže existuje více cest k aktuálnímu uzlu neporušující daná kritéria, porovnává se součin $r \cdot zábava$ těchto cest. r je Pearsonův korelační koeficient a *zábava* je součet ohodnocení uzlů obdobně jako u předchozího algoritmu. Největší z těchto násobků je přijat jako nejlepší dosavadní cesta do daného uzlu [16].

4.4 Ant Colony System pro TOPTW

Ačkoliv se nejedná o použití konkrétně pro návrh optimální turistické trasy, postup využití hejnové inteligence pro řešení TOPTW je aplikovatelný na návrh optimální turistické trasy, pakliže návrh optimální turistické trasy je modelován právě jako TOPTW.

Ant colony system (ACS) je algoritmus hejnové inteligence, který si bere inspiraci ve fungování kolonie mravenců při hledání potravy. Základem algoritmu je jednoduchý matematický model, mravenec, který při hledání řešení pokládá tzv. feromonovou stopu stejně tak jako mravenec v přírodě. Velikost feromonové stopy se odvíjí od toho, jak je dané řešení kvalitní. Mravenci v další iteraci poté berou v potaz hladinu feromonové stopy při rozhodování, jakou cestou se vydat [8].

4.4.1 Ant Colony System

Pro použití ACS pro TOPTW je zavedený nový model Hierarchic Team Orienteering Problem with Time Windows (viz 3.2.3). Tento model je zaveden z důvodu uchovávání předchozích fragmentů řešení pro lokální prohledávání nalezených řešení. Nevýhodou tohoto modelu je, že se musí optimalizovat $(m + 1)$ posledních řešení, která následně nejsou výsledkem hledaného optimální posloupnosti uzlů.

V modelu je startovní a cílový uzel sloučený do jednoho. Odchozí časy odpovídají startovnímu uzlu a příchozí časy odpovídají koncovému uzlu. Takto vytvořený sloučený uzel je rozložen na začátku řešení do cesty tolikrát, kolik je v grafu uzlů. Mravenec poté vytváří jenom jednu jedinou velkou cestu - autoři tak přibližují řešení k řešení TSP, ačkoliv u TOPTW respektive HTOPTW není nutností navštívit všechny uzly narozdíl od TSP.

První fází algoritmu je konstrukční fáze, ve které každý agent vytváří řešení, které neporušuje zadaná časová okna. Každý mravenec je vyslán sekvenčně z uzlu 1 a postupně prochází všemi uzly v grafu, dokud nejsou všechny uzly navštíveny. K přechodu mezi uzly se používá tzv. pravidlo přechodu. Pravidlo přechodu závisí na dvou parametrech. Prvním je τ_{ij}, což je feromonová stopa daného přechodu mezi uzlem i a j. Uzel j se musí nacházet v množině uzlů $F(i)$, což je podmnožina všech uzlů, které ještě nebyly navštíveny a jejichž návštěvou nedojde k porušení časových oken daného uzlu j, který má být navštíven. Druhý parametr η_{ij} je heuristická informace hrany mezi uzlem i a j a spočítá se dle (46).

$$\eta_{ij} = \frac{p_j}{max\{t_{ij}, O_j - v_i - s_i\} \cdot (C_j - v_j - s_j - t_{ij}) + 1} \tag{46}$$

Pro výběr uzlu j, který má následovat po uzlu i, je nutné spočítat parametr $q_0 = \frac{\hat{n}}{n}$, kde $\hat{n}$ je počet uzlů, které by se měli vyskytovat v řešení konstrukční fáze. S pravděpodobností q_0 je pak navštíven ten uzel, pro který platí $j = argmax_{l \in F(i)}\{\tau_{il} \cdot \eta_{il}\}$. S pravděpodobností $(1 - q_0)$ se zvolí náhodný uzel s pravděpodobností $p_{ij_{j \in F(i)}}$, která se počítá podle (47).

$$p_{ij_{j \in F(i)}} = \frac{\tau_{ij} \cdot \eta_{ij}}{\sum_{l \in F(i)} (\tau_{il} \cdot \eta_{il})} \tag{47}$$

Během přechodu mravence z uzlu i do uzlu j dochází k evaporaci feromonu a tedy ke snížení hodnoty feromonu na dané hraně. Tato lokální úprava hodnot feromonu je z toho důvodu, aby si další agenti vybrali spíše jinou cestu a během jedné iterace bylo nalezeno více rozdílných řešení, které lze porovnat. Úprava hladiny feromonu při přechodu mravence z uzlu i do uzlu j je v (48), kde ψ je parametr udávající úbytek feromonu na hraně a τ_0 je počáteční hodnota feromonových stop.

$$\tau_{ij} = (1 - \psi) \cdot \tau_{ij} + \psi \cdot \tau_0 \tag{48}$$

U ant colony system pokládá feromonovou stopu při globální úpravě hladiny feromonu pouze mravenec, který v dané iteraci dosáhl nejlepšího řešení. V případě HPTOW je to mravenec, který získal největší profit během cesty grafem. Úprava feromonových stop na hranách cesty, kterou se vydal nejlepší mravenec, je v (49), kde $Profit_{Best}$ je největší profit posbíraný nejlepším mravencem. ρ je parametr ovlivňující globální úpravu feromonů.

$$\tau_{ij} = (1 - \rho) \cdot \tau_{ij} + \rho \cdot Profit_{Best} \tag{49}$$

Na každé řešení, které je získáno pomocí konstrukční fáze, je následně aplikováno lokální prohledávání. Autoři používají operátor CROSS výměny, tzn. výměny dvou řetězců uzlů. Jeden řetězec může být i prázdný, proto může dojít k vložení

řetězce uzlů do jiného řešení. Maximální délka řetězce je dána jako proměnná, která se snižuje podle počtu iterací, kdy nedojde k zlepšení řešení [8].

Vstup: N
Výstup: $NejlepšíCesta$
while *nebylo dosaženo ukončovacích podmínek* **do**
  foreach *mravenec* **do**
    Inicializace cesty;
    for $i = 0$ **to** *počet uzlů v sadě uzlů* N **do**
      Vlož uzel do cesty;
    end
    Lokální aktualizace feromonů;
  end
  Globální aktualizace feromonů nejlepšího mravence;
  Lokální prohledávání;
  if $profit\ nejlepšího\ mravence > profit\ NejlepšíCesta$ **then**
    $NejlepšíCesta \leftarrow nejlepší\ mravenec$;
  end
end
return $NejlepšíCesta$

Algoritmus 3: Ant Colony System pro TOPTW[8]

4.4.2 Enhanced Ant Colony System

Úpravou konstrukční fáze výše zmíněného ACS dochází ke snížení časové náročnosti algoritmu. V přechodovém pravidlu je při přechodu z uzlu i zvolen s pravděpodobností q_0 uzel j takový, který se nachází v nejlepším dosavadním řešení po uzlu i. Pakliže tento uzel je nedosažitelný ve stávajícím řešení (porušuje časová okna, v nejlepším dosavadním řešení se nenachází), je použito přechodové pravidlo z původního ACS.

Druhou změnou je aplikace lokálního prohledávání s pravděpodobností $\frac{cnt-itr}{cnt}$, kde parametr *cnt* je nastaven na záčátku na 0 a s každou iterací se zvyšuje o 1. Parametr *itr* je parametr udávající iteraci, ve které naposledy došlo k lokálnímu prohledávání [9].

4.5 Iterovanné lokální prohledávání

Vstupem do algoritmu iterovaného lokálního prohledávání (Iterated Local Search), dále ILS, je sada uzlů N a sada cest (posloupností uzlů) M, která obsahuje m prázdných posloupností.

Na začátku iterovaného lokálního prohledávání (Iterated Local Search), dále ILS, je vytvořeno počáteční řešení pomocí heuristiky hladového vytvoření (Greedy Construction Heuristic).

Na začátku konstrukce počátečního řešení je vytvořena sada možných možných uzlů F, která je dána trojicí informací $\langle n, p, m \rangle$: uzlem n, který lze vložit do cesty m na pozici p. Pro každé $\langle n, p, m \rangle$ je určen poměr $ratio_{n,p,m}$ s jakou pravděpodobností budou vybrány do další fáze algoritmu. Obecně je přijímáno do dalšího výberu pouze f trojic $\langle n, p, m \rangle$ s největší hodnotou $ratio_{n,p,m}$. $ratio_{n,p,m}$ se spočítá dle (50), kde $Diff_{n,p,m}$ je rozdíl celkové doby strávené na cestě před a po vložení daného uzlu n do cesty m na pozici p a u^2 je druhá mocnina ohodnocení uzlu.

$$ratio_{n,p,m} = \frac{u^2}{Diff_{n,p,m}} \tag{50}$$

Pro f nejlepších F je spočítána pravděpodobnost $prob_{n,p,m}$ podle 51. Na základě pravděpodobnosti $prob_{n,p,m}$ je zvoleno jedno $\langle n, p, m \rangle$ pomocí Roulette Wheel Selection [17] a přidáno do cesty $m \in M$. Zároveň je uzel n přidán do N^*, což je seznam již rozvržených uzlů, a odebrán ze seznamu N', což je sada nerozvržených uzlů (platí $N^* \cup N' = N$). Konstrukce počátečního řešení je ukončena, když $F = \emptyset$.

$$prob_{n,p,m} = \frac{ratio_{n,p,m}}{\sum_{\langle i,j,k \rangle \in F} ratio_{i,j,k}} \tag{51}$$

Po vytvoření počátečního řešení se v iteracích opakují lokální prohledávání a perturbace. Algoritmus je ukončen, když je dosaženo časového limitu určeného pro výpočet.

Perturbace je aplikována na aktuální řešení tak, aby bylo možno uniknout z lokálního minima. Skládá se z operátoru **ExchangePath**, kdy jsou vyměněny pozice po sobě jdoucích cest v řešení. Operátor Move lokálního prohledávání začíná vždy výběrem první cesty v seznamu cest, proto díky výměně pozic je operátor Move postupně aplikován na odlišné cesty. Druhou částí perturbace je **Shake**, kdy je postupně ze všech cest odstraněno určité množství uzlů.

Lokální prohledávání se skládá ze šesti operátorů. Prvním je **Swap1**, při kterém jsou vyměněny uzly v rámci jedné cesty, která má nenižší zbývající čas (rozdíl času trvání cesty oproti koncovému časovému oknu koncového uzlu).

Operátor **Swap2** vyměňuje uzly v rámci dvou cest s nejnižšími zbývajícími časy.

Operátor **2-Opt** vyměňuje pořadí mezi dvěma uzly v rámci cesty s nejnižším zbývajícím časem. Všechny kombinace dvou uzlů jsou brány v potaz a operace je úspěšná, pakliže se zvýší zbývající čas a nejsou porušené časové podmínky. Operace je ukončena, když je nalezena výměna pořadí uzlů splňující tyto podmínky.

Operátor **Move** provádí přesun uzlu z jedné cesty do jiné. Nejdříve se zkouší přesunout uzel z první cesty v sadě cest do dalších cest pomocí stejného postupu, jako je generování F v konstrukci počátečního řešení. Pakliže uzel nelze přesunout, postupuje se k dalšímu uzlu. Operace je ukončena, jestliže se podaří některý uzel

Vstup: N, M
Výstup: S^*
$S_0 \leftarrow Construction(N, M)$;
$S_0 \leftarrow LocalSearch(S_0, N, N^*, M)$;
$S_0^* \leftarrow S_0$;
$NoImpr \leftarrow 0$;
while *časový limit nebyl překročen* **do**
  $S_0 \leftarrow Perturbation(S_0, N^*, N', M)$;
  $S_0 \leftarrow LocalSearch(S_0, N^*, N', M)$;
  if S_0 *lepší než* S^* **then**
    $S_0^* \leftarrow S_0$;
    $NoImpr \leftarrow 0$;
  else
    $NoImpr \leftarrow NoImpr + 1$;
  end
  if $(NoImpr + 1)\, Mod\, Threshold1 = 0$ **then**
    $S_0 \leftarrow S_0^*$;
  end
end
return S^*

Algoritmus 4: Iterované lokální prohledávání [4]

z některé cesty úspěšně přesunout nebo když je dosaženo posledního uzlu v poslední cestě.

Předposledním operátorem je **Insert**. Obdobně jako u vytváření počátečního řešení se vytvoří sada F trojic $\langle n, p, m \rangle$ z uzlů, které nejsou ještě umístěné do některé z cest. Vkládání pokračuje, dokud $F = \emptyset$.

Posledním operátorem je **Replace**. Vybere se cesta s nejvyšším zbývajícím časem a postupně jsou uzly v této cestě nahrazovány uzlem ze sady nerozvržených uzlů, který má největší ohodnocení. Pakliže je operace úspěšná, přejde se k dalšímu nerozvrženému uzlu s druhým nejvyšším ohodnocením. Operace je ukončena v případě, že neexistuje žádná další možná náhrada [4].

Algoritmus ILS již byl také testován na reálných datech. Jednalo se o vytvořený dataset 50 míst zájmu reprezentujících reálnou situaci [3].

4.5.1 SAILS

SAILS je rozšířením algoritmu ILS. Jedná se o hybridizaci iterovaného lokálního prohledávání s algoritmem simulovaného žíhání. Díky implementaci simulovaného žíhání a tedy základní ideje tohoto algoritmu, tj. přijímání i horších řešení za určité pravděpodobnosti, lze uniknout lokálnímu minimu [4].

Vstup: N, M
Výstup: S^*
$S_0 \leftarrow Construction(N, M)$;
$S_0^* \leftarrow S_0$;
$S_0' \leftarrow S_0$;
$Temp \leftarrow T_0$;
$NoImpr \leftarrow 0$;
while *časový limit nebyl překročen* **do**
  $InnerLoop = 0$;
  while $InnerLoop < MaxInnerLoop$ **do**
    $S_0 \leftarrow Perturbation(S_0, N^*, N', M)$;
    $S_0 \leftarrow LocalSearch(S_0, N^*, N', M)$;
    $\delta \leftarrow$ *hodnota* účelové $funkce\ S_0 -$ *hodnota* účelové $funkce\ S'$;
    if $\delta > 0$ **then**
      $S' \leftarrow S_0$;
      if S_0 *lepší než* S^* **then**
        $S^* \leftarrow S_0$;
        $NoImpr \leftarrow 0$;
      else
        $NoImpr \leftarrow NoImpr + 1$;
      end
    else
      $r \leftarrow$ *náhodné číslo* $[0, 1]$;
      if $r < exp\,(\delta / Temp)$ **then**
        $S' \leftarrow S_0$;
      else
        $S_0 \leftarrow S'$;
      end
      $NoImpr \leftarrow NoImpr + 1$;
    end
    $InnerLoop \leftarrow InnerLoop + 1$;
  end
  $Temp \leftarrow Temp \times \alpha$;
  if $NoImpr > Limit$ **then**
    $S0 \leftarrow S^*$;
    $S' \leftarrow S_0$;
    $NoImpr \leftarrow 0$;
  end
end
return S^*

Algoritmus 5: SAILS [4]

5 PROSTŘEDKY PRO VÝVOJ APLIKACE

Pro vývoj aplikace řešící návrh optimální trasy byla nutná zejména volba mapového API a API, které by bylo možné najít vhodná ohodnocená místa zájmu ideálně dle kategorií. Jako vývojová platforma byla zvolena platforma .NET.

5.1 Mapové API

Základním požadavkem na mapové API bylo, aby bylo možné získat reálné vzdálenosti v jednotkách času mezi místy zájmu. V dnešní době je na výběr několik mapových API, které vývojáři mohou bezplatně či za určitý poplatek využívat pro své komerční i nekomerční aplikace. Během přípravy byly brány v potaz některé mapové API jako Google Maps, TomTom, Here, OpenStreetMap a Bing Maps. Nakonec bylo rozhodnuto o využití Bing Maps pro vývoj a testování aplikace.

5.1.1 Bing Maps

Bing Maps je mapová platforma společnosti Microsoft. Tato služba funguje přes dvacet let, v roce 2020 došlo k navázání užší spolupráce mezi Bing Maps a společností TomTom, jedním z předních výrobců GPS navigačních systémů. Díky rozšíření základních mapových sad od společnosti TomTom by Bing Maps měly poskytovat aktuální a kvalitní mapové podklady.

Pro využívání Bing Maps se stačí zaregistrovat a získat tzv. Basic Key. Ten je součástí Developer licence určené pro menší společnosti, vývoj aplikací či studijní účely. Podle účelu aplikace nabízí licence v základu 125 000 bezplatných volání API ročně.

Výhodou Bing Maps oproti konkurenci je tzv. Distance Route Matrix API. Jedná se o API, kde na základě matice startovních a koncových bodů služba vrátí spočítané vzdálenosti mezi těmito body. Tuto službu nabízejí i ostatní mapová API. Výhodou Bing Maps je, že nepočítá volání API jako každý výpočet cesty mezi body, ale výsledný počet volání API je počet spočítaných hran mezi body podělený čtyřmi. Pro představu lze uvést, že pro sestavení úplného grafu s 50 místy zájmu je nutné mít 2450 ohodnocených hran.

Bing Maps dále nabízí kromě jiného SDK pro vývoj webových aplikací či mobilních aplikací pro operační systémy Android a iOS.

5.2 API s informacemi o místech zájmu

Druhým důležitým prostředkem pro vytvoření reálné aplikace pro návrh optimální turistické trasy ve městě je získání ohodnocených bodů zájmů. Opět existuje několik

služeb, které vývojář může využít. Přikladem jsou služby Google Places, TripAdvisor nebo FourSquare. Využita byla nakonec poslední zmíněná služba.

5.2.1 Foursquare

Foursquare je služba, která nabízí informace o více než 100 milionech míst zájmu po celém světě. Informace o daném místě zájmu jsou kromě GPS souřadnic například jméno daného místa, místní název, otevírací doba, přiřazení do více než 1100 kategorií apod.

Pro využití v této aplikaci je důležitá informace hodnocení místa zájmu. Foursquare provozuje službu Foursquare City Guide, která na základě parametrů uživatele vyhledává místa zájmu v zadané lokalitě. Uživatel poté může danému místu napsat recenzi a ohodnotit jej. Foursquare ukládá hodnocení a vývojaři mohou informace o průměrném hodnocení či počtu hodnocení využívat ve své aplikaci.

Základní licence je zdarma. Vývojář dostává rozpočet 200$ na jeden měsíc, při překročení rozpočtu přestává služba fungovat, popřípadě je nutné za každé další využití zaplatit.

5.3 Vývojová platforma .NET

.NET je open sourcová, multiplatformní vývojová platforma společnosti Microsoft. Platforma .NET je zaměřená na to, aby byla co nejvíce využitelná pro různé druhy projektů. Vývojáři mohou vytvářet webové aplikace (ASP.NET), desktopové aplikace pro operační systém Windows (WPF, WinForms, UWP, Xamarin), multiplatformní aplikace (Xamarin pro vývoj aplikací pro iOS, Android a UWP).

Jako programovací jazyk převládá jazyk C#. Jedná se o opensourcový programovací jazyk podporující objektově orientované programování (OOP). C# vychází z rodiny jazyků C a je pevně spjatý právě s platformou .NET. Kromě jazyka C# lze využít ještě jazyky F a Visual Basic.

Aplikace pro návrh optimální turistické trasy má největší potenciál jako webová či mobilní aplikace, .NET nabízí prostředky pro vývoj takových typů aplikací. Platforma .NET byla vybrán zejména kvůli osobním preferencím autora.

6 IMPLEMENTACE ALGORITMŮ

Pro řešení návrhu optimální turistické trasy je zpočátku nutné, jak na problém pohlížet a jaký model problému vybrat. Single Tour TTDP popsaný v podkapitole 3.1 je model, ve kterém se předpokládá vytvoření jedné turistické trasy. V reálné aplikaci to znamená vytvoření trasy pro jeden den. Oproti tomu Multiple Tour TTDP (viz podkapitola 3.2) popisuje reálnější případ, kdy si uživatel chce rozdělit návštěvu města do několika dní.

Vytvořená vzorová aplikace tedy uvažuje model Multiple Tour TTDP, konkrétně TOPTW, který je popsán v podkapitole 3.2.2. Odůvodnění výběru tohoto modelu je, že popisuje poměrně věrně realitu návrhu optimální trasy s časovými omezeními.

Vhodné algoritmy pro řešení TOPTW jsou algoritmy ze skupiny heuristik a metaheuristik s ohledem na to, že TOPTW je NP těžká úloha. Druhým důvodem pro výběr heuristických či metaheuristických algoritmů je čas výpočtu dané úlohy. V reálných aplikacích je nutné, aby byl výpočet co nejkratší a tedy uživatelsky přívětivější. I s tímto ohledem je vhodnější uživateli aplikace předložit řešení blížící se optimu, jehož výpočet je kratší a výsledky jsou téměř optimální.

Z výše uvedených důvodů byly vybrány pro řešení úlohy návrhu optimální turistické trasy algoritmy ILS, SAILS a ACS.

6.1 Implementace ILS

Algoritmus je popsán vpodkapitole 4.5. Jedná se o algoritmus lokálního prohledávání, který je schopný najít dobrá téměř optimální řešení při krátkém času výpočtu. Autoři algoritmu již ILS využili i přímo pro řešení TTDP na konkrétních reálných datech. Z těchto důvodů se jedná o algoritmus, jehož implementace je pro danou úlohu dobrou volbou.

V rámci vytváření alogritmu byly vytvořeny pomocné třídy. Místo zájmu tak představuje třída POI. Pro vytvoření návrhu optimální trasy jsou důležité zejména parametry $Rating$ (ohodnocení místa zájmu), $Duration$ (délka trvání návštěvy daného místa zájmu), seznam $Neigbours$ (seznam sousedů), seznam $Distance$ (vzdálenost k patřičným sousedům v sekundách) a slovník $Opening$ (slovník, kde klíč je číslo dne v týdnu a hodnota třída $Opening$ s parametry $OpenHour$ a $CloseHour$ udávající otevírací a zavírací dobu v sekundách. Ostatní parametry jako $Name$, Tel apod. jsou využity později při interpretaci výsledků (viz kapitola 7).

Pro reprezentaci trasy výletu byla vytvořena pomocná třída $Route$. $Route$ je třída s parametry $Length$ (délka trasy v sekundách), $Score$ (celkové ohodnocení všech míst zájmu v dané trase), $DayNum$ (číslo dne v týdnu) a $RouteList$ (se-

znam míst zájmu třídy POI). Metoda $ComputeRoute$ s parametry $POI\ startNode$, $POI\ endNode$, $int\ dayNum$ vrací délku trasy uložené v dané třídě $Route$. Pakliže trasa porušuje časová kritéria, vrací -1.0. Přetížení metody $ComputeRoute$ s parametry $List\langle POI\rangle\ route$, $POI\ startNode$, $POI\ endNode$, $int\ dayNum$ počítá délku trasy ne v uložené třídě $Route$, ale délku trasy v předaném parametru $route$.

Poslední pomocná třída pro návrhu optimální turistické trasy je třída NPM. Ta slučuje parametry M (cesta, do které má být vložen uzel N na pozici P), N (uzel v cestě M na pozici P), P (pozice uzlu N v cestě M), $Ratio$ (poměr z výpočtu (50)) a $RouteLength$ (délka nové trasy M po přidání uzlu N na pozici P).

V rámci OOP je vytvořena třída ILSBase. Jelikož algoritmy ILS a SAILS sdílí stejné metody a parametry, je vhodné vytvořit třídu ILSBase, jež bude rodičovskou třídou pro třídy ILS a SAILS.

$ILSBase$ obsahuje metody $LocalSearch$ provádějící lokální prohledávání a příslušné metody s operátory lokálního prohledávání: $Swap1$, $Swap2$, $Opt-2$, $Move$, $Insert$ a $Replace$. Dále $ILSBase$ obsahuje metodu $Perutrbation$ a související metody $ExchangePath$ a $Shake$. Další metodou, která je stejná jak pro algoritmus ILS tak pro algoritmus SAILS, je metoda $Select$.

Vstup: timeLimit
Výstup: S_star
$S0 \leftarrow StartConstruction()$;
$S0 \leftarrow LocalSearch()$;
$S\_0star \leftarrow S0$;
$noImprovement \leftarrow 0$;
while $timeLimit > elapsedSeconds$ **do**
  $S0 \leftarrow Perturbation(noImprovment)$;
  $S0 \leftarrow LocalSearch()$;
  if $S0\_score > S\_star\_score$ **then**
    $S\_star \leftarrow S0$;
    $NoImprovement \leftarrow 0$;
  else
    $NoImprovement \leftarrow NoImprovement + 1$;
  end
  if $(NoImpr + 1)\ Mod\ threshold1 = 0$ **then**
    $S0 \leftarrow S\_star$;
  end
end
return S_star

Algoritmus 6: Metoda StartILS

Vstupem do implementace algoritmu ILS je již vytvořený úplný graf $Graph\ g$ reprezentovaný listem sousedů. Uzly grafu jsou instancí třídy POI. Konstruktor třídy ILS dále přijímá parametry $days$ (seznam čísel dnů v týdnu), $threshold1$ (parametr pro rozhodnutí, zda dále prohledávat nejlepší dosavadní řešení či aktuální

```
Výstup: S0
npm = ∅;
F ← UpdateF();
noImprovement ← 0;
while počet prvků v F > 0 do
    npm ← Select(F);
    S0 ← npm;
    N = N \ npm.N;
    N_star ← N ∪ npm.N;
end
return S0
```

Algoritmus 7: Metoda StartConstruction

řešení), *ithreshold*2 a *threshold*3 (parametry pro zjištění, jaký druh perturbace provést), f (parametr udávající kolik trojic $\langle NPM \rangle$ vybrat pro další práci viz 4.5), *startNode* a *endNode* (počáteční respektive koncový bod cest).

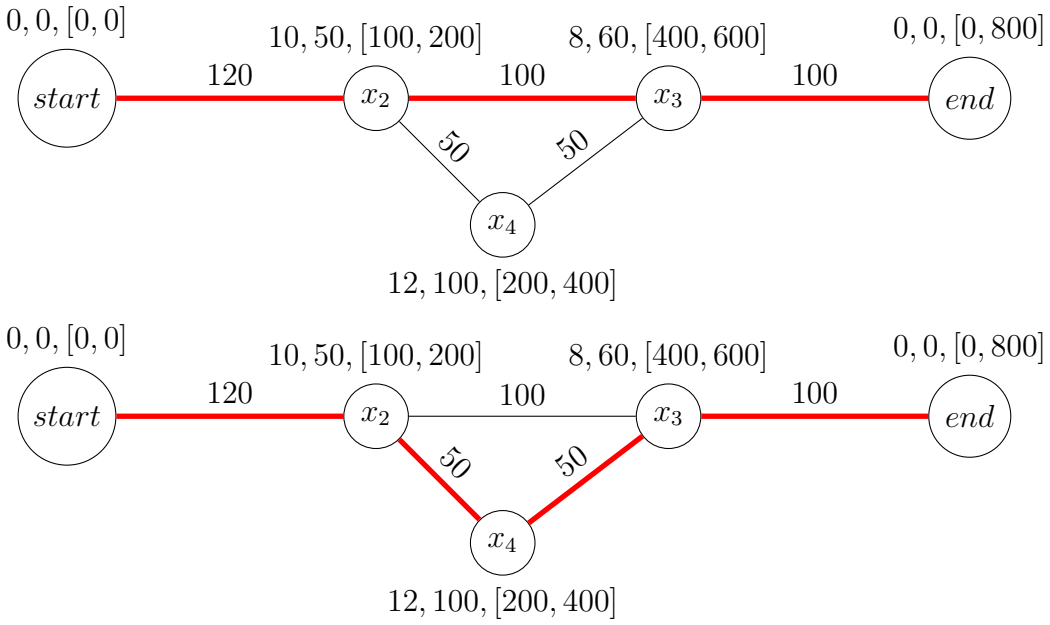
```
Výstup: F
F = ∅;
new_distance ← 0;
ratio ← 0;
for i = 0 to počet nerozvržených uzlů do
    for j = 0 to počet dní do
        for k = 0 to počet možných pozic v cestě j do
            if uzel i je otevřený v den j then
                new_distance ← TryInsert(N[i], startNode, endNode, k);
                if new_distance > 0 then
                    ratio ← spočítej ratio;
                    F ← ⟨n, p, m⟩ = ⟨uzel i, pozice k, cesta j⟩;
                end
            end
        end
    end
end
F ← f prvků z F s největším parametrem ratio;
return F
```

Algoritmus 8: Metoda UpdateF

Samotný výpočet ILS začíná voláním metody *StartILS*, které je nutné předat parametr *timeLimit* (maximální časový limit výpočtu algoritmu v sekundách). Pseudokód metody je v algoritmu 6.

Parametr $S0$ je aktuálně prohledávání řešení reprezentované jednodimenzionálním polem M tříd *Route*, kde M je délka seznamu dní a tedy počet dní výletu. Počáteční řešení se získá voláním metody *StartConstruction* (algoritmus 7). Získání počátečního řešení je popsáno v 4.5. V průběhu psaní kódu došlo ke změně rovnice

(50) použité v algoritmu 8 při počítání parametru *ratio*. Na základě velikosti poměru $ratio_{n,p,m}$ je zvoleno f nejlepších trojic $\langle n, p, m \rangle$. Rozdíl celkové doby strávené na trase před a po vložení uzlu n do trasy m na pozici p ($Diff_{n,p,m}$) ovšem může být nulový, jak je demonstrováno ve vzorové ukázce v obrázku 8.



Obr. 8: Vložení uzlu neměnící délku trasy [vlastní zpracování]

Předpokládejme první cestu z uzlu *start* do uzlu *end* přes uzly x_2 a x_3. Celková doba trasy je 560, celkový profit pak 18. Při výběru dalšího uzlu do trasy je možné vložit uzel x_4 mezi uzel x_2 a x_3, jak je ukázáno ve druhém příkladu. Při vložení uzlu x_4 do trasy dojde ke zvýšení celkového užitku na 30, celková doba času stráveného na trase ovšem zůstává 560, jelikož uzel x_4 byl vložen do trasy v době čekání na začátek časového okna x_3. Rozdíl $Diff_{n,p,m} = 0$. Aby nedocházelo k dělení nulou v rovnici (50), byla použita rovnice (52).

$$ratio_{n,p,m} = \frac{u^2}{Diff_{n,p,m} + 0.001} \tag{52}$$

Počáteční řešení $S0$ vrácené metodou *StartConstruction* je nejlepší dosavadní řešení S_star. Algoritmus nadále pokračuje ve smyčce, dokud čas výpočtu není vyšší než maximální čas výpočtu *timeLimit*. Ve smyčce se provádí postupně metoda *Perturbation* a následně *LocalSearch*.

Metoda *Perturbation* se aplikuje, aby se uniklo lokálním řešením. Na základě parametrů *threshold*2 a *threshold*3 se rozhoduje, zda se aplikuje operátor *ExchangePath* nebo *Shake*.

ExchangePath mění pořadí cest v aktuálním řešení $S0$ z toho důvodu, že operátor lokálního prohledávání *Move* pracuje vždy nejdříve s cestou aktuálního

řešení $S0$, která je první v pořadí. Díky změně pořadí a aplikací operátoru $Move$ na jinou cestu je možné uniknout lokálnímu minimu. $ExchangePath$ vymění pořadí cesty dané parametrem $exchangeInd$ s cestou s indexem $exchangeInd - 1$ (pakliže je $exchangeInd = 0$, pak se vymění cesta s indexem 0 s poslední cestou a $exchangeInd$ se změní na poslední index řešení $S0$). Na začátku je parametr $exchangeInd$ nastaven na poslední index řešení $S0$ a po aplikaci operátoru $ExchangePath$ je $exchangeInd = exchangeInd - 1$.

Operátor $Shake$ odstraní ze všech cest aktuálního řešení $S0$ uzly. Počet uzlů k odstranění je daný parametrem *cons*. *cons* se zvýší o jedna při každé druhé iteraci algoritmu. Na začátku algoritmu je *cons* inicializován na hodnotu 1, pakliže je *cons* > *počet uzl*ů *nejdel*ší *trasy* $S0$, pak $cons \leftarrow 1$. Odstranění uzlů začíná na pozici dané parametrem *post* a postupně se odstraňují další uzly cesty. Při dosažení posledního uzlu cesty řešení se začne odstraňovat od prvního uzlu. Odstraňování uzlů cesty končí, pokud je z cesty odstraněno *cons* uzlů, nebo je daná cesta prázdná. Parametr *post* se zvýší o *cons* při každé aplikaci operátoru $Shake$. Jesliže je *post* je menší než počet uzlů nejkratší trasy $S0$, pak nová hodnota *post* je snížena právě o počet uzlů nejkratší trasy $S0$. Parametr *post* je na začátku algoritmu nastaven na 0.

Lokální prohledávání $LocalSearch$ postupně aplikuje operátory lokálního prohledávání na aktuální řešení $S0$ (viz 4).

Při dosažení určitého počtu iterací bez zlepšení výsledků dojde k $S0 \leftarrow S\_star$ a tedy dalšímu prohledávání dosavadního nejlepšího řešení. Určení počtu iterací bez zlepšení výsledku se nastaví parametrem $threshold1$.

Po uplynutí časového limitu $timeLimit$ je vráceno nejlepší dosavadní řešení S_star, které je řešením problému.

6.2 Implementace SAILS

Algoritmus SAILS je hybridizace algoritmu ILS a simulovaného žíhání. Většina operátorů a metod je totožná s algoritmem ILS, zásadní rozdíl je ve vnitřní smyčce algoritmu, kdy na základě určité pravděpodobnosti je přijato i horší řešení jako nové aktuální řešení.

Implementace algoritmus SAILS je instancí třídy $SAILS$, jejíž rodičovskou třídou je $ILSBase$. Konstruktor třídy přijímá stejné parametry jako konstruktor třídy ILS vyjma parametru $threshold1$, který je nahrazený parametrem *limit*. Dále jsou navíc nutné některé parametry související se simulovaným žíháním. Jedná se o parametr počáteční teploty $t0$ a parametr udávající ochlazování *alpha*. Posledním parametrem je parametr $maxInnerLoop$ udávající maximální počet iterací vnitřní smyčky algoritmu.

Hledání optimální turistické trasy začíná voláním metody $StartSAILS$ se vstupním parametrem $timeLimit$. Počáteční řešení $S0$ je nalezeno pomocí metody $StartConstruction$. Na toto řešení je aplikováno lokální prohledávání $LocalSearch$. Původní řešení je nejlepší dosud nalezené řešení, proto $S\_star \leftarrow S0$. Pro vnitřní cyklus se zavede nová proměnná S' a inicializujeme ji na $S' \leftarrow S0$.

Algoritmus běží, dokud je čas výpočtu menší než hodnota času uložená v proměnné $timeLimit$. Na začátku tohoto vnějšího cyklu se nastaví proměnná $innerLoop \leftarrow 0$.

Vnitřní cyklus algoritmu běží, dokud $innerLoop < maxInnerLoop$. Na aktuální řešení se použijí metody $Perturbation$ a $LocalSearch$. Dále se zjistí, zda je proměnná $delta = S0\_score - S'\_score$ větší než nula. Pakliže je $delta > 0$, pak $S' \leftarrow S0$. Jesliže je účelová funkce $S0$ větší než hodnota účelové funkce S_star, pak $S\_star \leftarrow S0$.

Jestliže proměnná $delta \leq 0$, tak je nutné zjistit, zda na základě pravděpodobnosti přijmout toto řešení. Pakliže platí, že $r < e^{\frac{delta}{temp}}$, kde r je náhodně vygenerovaná veličina s normálním rozložením v intervalu $(0, 1)$, tak platí $S' \leftarrow S0$, jinak $S0 \leftarrow S'$.

Po skončení vnitřního cyklu se sníží teplota $temp \leftarrow temp \cdot alpha$. Jesliže nedošlo po určitou dobu ke zlepšení účelové funkce, pak se stává nejlepší dosavadní řešení S_star aktuálním řešením $S0$, $S0 \leftarrow S\_star$ a zároveň $S' \leftarrow S0$.

Vstup: $timeLimit$
Výstup: S_star
$S0 \leftarrow Construction();$
$S\_star \leftarrow S0;$
$S0' \leftarrow S0;$
$temp \leftarrow t0;$
$NoImpr \leftarrow 0;$
while $timeLimit > elapsedSeconds$ **do**
  $innerLoop = 0;$
  while $innerLoop < maxInnerLoop$ **do**
    $S0 \leftarrow Perturbation();$
    $S0 \leftarrow LocalSearch();$
    $delta \leftarrow$ *hodnota* účelové $funkce\ S0 -$ *hodnota* účelové $funkce\ S';$
    if $delta > 0$ **then**
      $S' \leftarrow S0;$
      if *hodnota* účelové $funkce\ S0\ >$ *hodnota* účelové $funkce\ S\_star$ **then**
        $S\_star \leftarrow S0;$
        $noImpr \leftarrow 0;$
      else
        $noImpr \leftarrow noImpr + 1;$
      end
    else
      $r \leftarrow random;$
      if $r < e^{delta/temp}$ **then**
        $S' \leftarrow S0;$
      else
        $S0 \leftarrow S';$
      end
      $noImprovement \leftarrow noImprovement + 1;$
    end
    $innerLoop \leftarrow innerLoop + 1;$
  end
  $temp \leftarrow temp \times alpha;$
  if $noImprovement > limit$ **then**
    $S0 \leftarrow S\_star;$
    $S' \leftarrow S0;$
    $noImprovement \leftarrow 0;$
  end
end
return S_star

Algoritmus 9: Metoda StartSAILS

6.2.1 Konstrukce pomocí ant colony system

V rámci dobrých výsledků implementace ant colony system byl pokus o vylepšení vytvořené implementace SAILS právě o zmíněný algoritmus ACS. Počáteční konstrukční fáze takového algoritmu SAILS je nahrazena konstrukcí řešení pomocí ACS.

Toto řešení SAILS+ACS bylo vyzkoušeno a otestováno, více viz. Zhodnocení a diskuze (kapitola 8).

6.3 Implementace Ant Colony System

Obdobně jako u předchozích implementací byly vytvořeny pomocné třídy AntRoute a AntNPM. Třída AntRoute reprezentuje jednu trasu řešení. AntNPM je obdobně jako třída NPM v předchozích implementacích ILS a SAILS využita pro lokální prohledávání. Třída POI reprezentující místo zájmu je totožná s třídou POI v implementaci algoritmů ILS a SAILS.

Dále byla vytvořena třída Ant reprezentující model mravence, tedy výpočetního agenta. Konstruktor třídy přijímá následující parametry. $startNode$ a $endNode$ (implementacd třídy POI reprezentující startovní a koncové místo výletu), $days$ (seznam celočíselných údajů o zvolených dnech výletu), N (seznam všech míst zájmu), $relPOIDay$ (pole seznamů míst zájmu relevantních pro dané dny - na patřičném indexu se místo nenachází, jestliže ten den nemá otevírací dobu) a parametr $initPheromone$ (hodnota počátčního nastavení feromonových stop). Dva druhy seznamů míst zájmu jsou z toho důvodu, že pole seznamů míst zájmu $relPOIDay$ je použito pro vytváření řešení a kompletní seznam míst zájmu N se využívá pro lokální prohledávání.

Implementace altoritmu Ant Colony System je reprezentována třídou ACO. Konstruktor třídy přijímá parametry $tau0$ (parametr počátečního nastavení feromonových stop), rho (parametr nutný pro globální aktualizaci feromonové stopy), psi (parametr určující lokální aktualizaci feromonových stop) a parametr $q0$ (parametr určující, jakým způsobem vybrat uzel pro vložení do řešení).

Výpočet algoritmu začíná voláním metody StartACO se vstupním parametrem $timeLimit$. Ukončující podmínka ve formě časového limitu byla zvolena z důvodu porovnání algoritmů.

Algoritmus dále běží ve smyčce, dokud není překročen časový limit. Pro každého mravence se vytvoří řešení, provede se lokální úprava feromonů a provede se lokální prohledávání. Z mravenců se vybere ten agent, který má největší profit. Pouze nejlepší mravenec provádí globální aktualizaci feromonové stopy. Jestliže je profit nejlepšího mravence lepší, než doposud největší nalezený profit, pak je řešení tohoto mravence přijato jako doposud nejlepší.

```
Vstup: timeLimit
Výstup: BestRoutes
actBestScore;
bestAnt;
BestScore ← 0;
while timeLimit > elapsedSeconds do
    actBestScore ← 0 foreach Antant in Ants do
        ant.Init(N);
        ant.BuildSolution(rho, psi, q0);
        atn.LocalSearch();
        for i = 0 to počet uzlů v sadě uzlů N do
            Vlož uzel do cesty;
        end
        ant.LocalUpdate();
        if ant.Score > actBestScore then
            BestAnt ← ant;
        end
    end
    GlobalUpdate(bestAnt);
    if bestAnt.Score > BestScore then
        BestScore ← bestAnt.Score;
        BestRoutes ← bestAnt.Routes;
    end
end
return BestRoutes
```

Algoritmus 10: Metoda StartACO

Vytvoření řešení metodou *BuildSolution* je obdobné jako u ACS pro TOPTW (viz 4.4) s tím rozdílem, že bylo zkoušeno vytváření cest podobně jako u ILS či SAILS místo vytvoření jedné velké cesty. Výběr místa zájmu, který se vloží do řešení závisí na velikosti náhodně vygenerovaného čísla z intervalu $(0, 1)$. Jestliže je toto náhodné číslo menší než parametr $q0$, pak se vybírá uzel dle $j = argmax_{l \in N}\{tau_{il} \cdot \eta_{il}\}$. Parametr η_{il} se spočítá dle (53) kde p_l^2 je profit uzlu l. Proměnná t_{il} je čas cesty mezi uzlem i a uzlem l. Umocnění profitu na druhou je z toho důvodu, aby se upřednostnil význam profitu oproti času stráveném na cestě mezi uzly.

$$\eta_{il} = \frac{p_l^2}{t_{il}} \tag{53}$$

Jestli je náhodně vygenerované číslo větší než paramter $q0$, pak se volí pomocí metody *ChoosePOI*. V metodě je na základě poměru (53) zvoleno místo zájmu pomocí Roulette Wheel Selection.

Lokální aktualizace feromonů se provádí stejně, jako je popsáno ve vzorci (47). Globální aktualizace feromonových stop je totožná s rovnicí (48). Globální aktualizaci provádí pouze nejlepší mravenec v dané iteraci.

```
Vstup: rho, psi, q0
actPOI;
actN;
lastNode while dayInd > početcest do
    actN ← relPOIDay[dayInd];
    actN ← actN \ N_star;
    while actN.Count > 0 do
        actPOI ← StateTransition(rho, q0);
        if actPOI = endNode then
            LocalUpdate(lastNode, endNode, psi);
            break;
        else
            Routes[dayInd] ← Routes[dayInd] ∪ actPOI;
            actN ← actN \ actPOI;
            N ← N \ actPOI;
            N_star ← N_star ∪ actPOI;
            Routes[dayInd].Score ←
              Routes[dayInd].Score + actPOI.Rating;
        end
    end
    dayInd ← dayInd + 1;
end
```

Algoritmus 11: Metoda BuildSolution

Lokální prohledávání využívá stejné operátory, jako jsou popsané v algoritmu ILS (viz 6.1). Postupně se aplikují operátory Swap1, Swap2, 2-Opt, Move, Insert a Replace.

Jakmile je dosaženo časového limitu vyhrazeného pro běh algoritmu, aktuálně nejlepší řešení je přijato jako nejlepší řešení.

7 MOBILNÍ APLIKACE

V rámci diplomové práce byla vytvořena vzorová mobilní aplikace (obr **??**, ve které je na základě údajů zadaných uživatelem vytvořen personalizovaný návrh turistické trasy.

Mobilní aplikace pro telefony s operačním systémem Android byla vytvořena pomocí frameworku Xamarin. Xamarin je multiplatformní opensourcová vývojová platforma pro vývoj aplikací pro mobilní zařízení s operačním systémem Android a iOS a pro platformu Universal Windows Platform (UWP). Kód je psaný v jazyce C# a k popisu grafického rozhraní je používaný značkovací jazyk XAML.

Pro personalizovaný návrh optimální turistické trasy jsou nutná některá data od uživatele. Aplikace je navržená tak, aby byl uživatel proveden postupně zadáním svých zájmů, preferencí a dalších údajů potřebných pro vytvoření trasy.

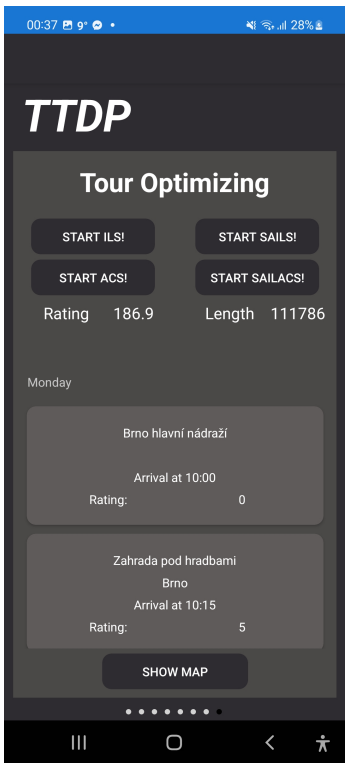
Uživatel nejdříve zadá město ve kterém by měl návrh turistické trasy proběhnout. Zadání města je řešené tak, že textový řetězec zadaný uživatelem je předán Foursquare Autocomplete API a vrácené výsledky hledání předloženy uživateli. Ten vybere z vrácených výsledků město, díky čemuž jsou získány souřadnice pro další práci.

V rámci aplikace je dále od uživatele požadovaný výběr startovního a koncového bodu výletu. Zde je důležité, aby již bylo určené město. Zadaný řetězec je využit pro hledání pomocí Foursquare Places API. Hledání zadaného řetězce je dáno v určitém poloměru v okolí souřadnic zvoleného města. Jesliže město pro výlet nebylo zadáno, uživatel je upozorněn a hledání počátečního respektive koncového bodu se neprovede. Uživateli je nabídnuta volba vybrat koncový bod jiný než startovní, popřípadě se startovní a koncový bod mohou shodovat.

Dalším krokem je výběr kategorií míst zájmů, která má návrh trasy obsahovat. Uživateli jsou nabídnuty kategorie *Museum*, *Planetarium*, *PublicArt*, *ZOO*, *Aquarium*, *ArtGallery*, *SpiritualCenter* a *LandmarkandOutdoors* (věr několika kategorií z Foursquare). Uživatel si dále u každé kategorie může vybrat, jak dlouhou dobu by v místě dané kategorie chtěl trávit.

V dalším kroku uživatel zadá, pro které dny v týdnu by chtěl výsledný výlet vytvořit. Tím je dáno, kolikadenní by měl výlet být. Pro každý den si navíc uživatel vybere časový údaj začátku výletu a jeho konce. Tedy čas, ve který výlet pro daný den začíná v počátečním bodě a kdy výlet nejpozději končí v koncovém bodě.

Další krok je, že uživatel potvrdí správnost informací z předchozích kroků a zadá hledání míst zájmu. K nalezení daných míst se využívá Foursquare Places API. Jako střed hledání se využijí souřadnice zadaného města. Do parametrů volání API se zadají čísla kategorií, které si uživatel zvolil. Horní limit počtu výsledků je zadaný na 50 (jedná se o horní limit Foursquare Places API), přičemž požadavek na



Obr. 9: Vytvořená aplikace [vlastní zpracování]

službu Foursquare je, aby výsledky vracel seřazené podle hodnocení od nejlepšího po nejhorší. Výsledkem volání API je tak maximálně 50 míst zájmu s největším hodnocením v pevně zadaném poloměru okolo koordinátů středu města.

Nalezeným místům zájmu je možné změnit jejich hodnocení. Třída *POI* reprezentující místo zájmu obsahuje parametr *OriginalRating* a *Rating*. Oba parametry jsou inicializovány na stejnou hodnotu získanou pomocí Foursqare Places API. Pakliže si uživatel přeje změnit hodnocení, mění se parametr *Rating*, se kterým je i dále pracováno. Díky tomu, že je uložena hodnota originálního ohodnocení lze resetovat ohodnocení na původní hodnotu. Možnost ohodnotit místo zájmu je z toho důvodu, že ne všechna místa získaná pomocí služby Foursquare musí obsahovat hodnocení. Když je hodnocení rovno nule, pak se dané místo zájmu nemůže objevit ve výsledné trase, jelikož je na začátku odstraněno ze vstupních dat. Jestliže si uživatel přeje místo zahrnout do tvorby řešení, musí jej ohodnotit na hodnotu větší než 0.

Poté, co jsou nalezená místa zájmu zvolených kategorií v zadaném městě a uživatel si již nepřeje měnit jejich ohodnocení, uživatel volí možnost vytvoření turistické trasy. Na začátku návrhu trasy je nutné vytvoření úplného orientovaného grafu. Volba orientovaného grafu je z důvodu reálnějších časů cesty mezi jednotlivými místy zájmu. Například při volbě cestování autem může hrát roli, jestli se nachází na cestě jednosměrné ulice. Jelikož historická místa zájmu se často nachází v centru

měst, kde jsou často jednosměrné komunikace, mohly by časy cesty mezi místy zájmu být zkresleny.

Orientovaný úplný graf je vytvořený díky informacím ze služby Bing Maps. Použitá služba je Route Matrix API, které je nutné předat matici počátečních míst a matici koncových míst. Místa jsou reprezentovány geografickými souřadnicemi. Jelikož je velikost matice omezená, volá se požadavek zvlášť pro každé místo. Matice vypadá tedy tak, že matice startovních míst je jednočlenná a je v ní aktuální místo zájmu respektive počáteční místo výletu. Matice koncových míst je matice se souřadnicemi všech ostatních míst včetně koncového bodu výletu. Pro každé místo je takto vytvořený seznam sousedů a patřičný seznam vzdáleností mezi daným místem a patřičným sousedem.

Jakmile je vytvořen graf, přechází se k samotnému vytvoření turistické trasy. Všechny algoritmy přijímají jako vstupní parametr vytvořený úplný orientovaný graf, lze tak jednoduše zakomponovat do mobilní aplikace všechny algoritmy pro vytvoření návrhu optimální turistické trasy.

Po vytvoření turistické trasy pomocí algoritmu ILS, SAILS nebo ACS jsou výsledky prezentovány uživateli a zobrazena mapa trasy, respektive více tras, jestliže je zvoleno více dnů.

Bohužel z časových důvodů nebyla aplikace dokončena k dokonalosti. V rámci předložené verze aplikace se například nekontroluje připojení mobilního telefonu k internetu. Bez připojení k internetu a následném požadavku na Bing Maps API či Foursquare API aplikace spadne. Ačkoliv je aplikace funkční z hlediska výpočtu návrhu optimální trasy, finální dokončení aplikace je jistě dalším návrhem práce.

8 ZHODNOCENÍ A DISKUZE

V následující kapitole budou porovnány implementované algoritmy na vzorovém příkladě. V další části se zhodnotí výběr prostředků pro vývoj aplikace.

8.0.1 Testování aplikace

Výsledná aplikace byla testována na reálném příkladě s reálnými daty o místech zájmu získané pomocí Foursquare API a času mezi místy zájmu získanými pomocí Bing Maps.

Aplikace byla spuštěna na zařízení Samsung Galaxy A52S 5G (osmijádrový procesor Qualcomm Snapdragon 778G, až 2400 MHz, 6 GB RAM, operační systém Android 12).

Výběr parametrů

Pro implementované algoritmy jsou důležité hodnoty vstupních parametrů. V průběhu testování aplikace bylo voleno několik konfigurací parametrů pro algoritmy.

U algoritmu ILS jsou vstupními parametry $threshold1$, $threshold2$, $threshold3$ a f. Parametr $threshold1$ ovlivňuje, po kolika iteracích se přijímá aktuálně nejlepší řešení jako aktuálně prohledávané řešení. Když je aktuálně nejlepší řešení přijato jako aktuální řešení, dochází k dalšímu lokálnímu prohledávání nejlepšího řešení.

Tab. 1: Parametry algoritmu ILS

threshold1	threshold2	threshold3	f
10	20	3	5

Parametry $threshold2$ a $threshold3$ u algoritmu ILS mají vliv na operátor $Perturbation$, který má za úkol vyhnout se uvíznutí v lokálnímu minimu. Posledním parametrem je f. Tento parametr určuje, kolik uzlů vstupuje do dalšího výběru při vkládání uzlu do řešení. Pakliže je parametr nastaven na $f = 1$, pak to znamená, že pouze do dalšího výberu postupuje pouze jeden nejlepší uzel, tzn. má stoprocentní šanci být vybrán. Pro ILS byly pro závěrečné testování zvoleny parametry dle tabulky (1).

Pro algoritmus SAILS je funkce parametrů $threshold2$ a $threshold3$ stejná, jako u ILS. Parametr $limit$ je obdobou parametru $threshold1$ a určuje, po kolika iteracích zvolit dosavadní nejlepší řešení za prohledávané řešení. Funkce parametru f je totožná. Velikost vnitřní smyčky $MaxInnerLoop$ je zvolena na 50. Parametr počáteční teploty T_0 pro simulované žíhání byl zvolen jako velk0 číslo a postupem během experimentů snižován. Poslední parametr α určuje snižování teploty a byl volen jako 75. Hodnoty všech parametrů jsou v tabulce (2)

Tab. 2: Parametry algoritmu SAILS

limit	threshold2	threshold3	f	α	T_0
5	20	3	5	0.75	1000

Parametry algoritmů jsou přijaté také na základě doporučení autorů. V průběhu testování se ukázaly jako robustní.

Pro algoritmus ACS je třeba vybrat počet mravenců neboli výpočetních agentů. Při velkém počtu agentů roste časová náročnost výpočtu. Během testů se ukázalo, že větší počet mravenců nevedl k lepším výsledkům.

Počáteční hodnota feromonové stopy byla postupně snižována z velké konstanty až na hodnotu v tabulce (3). Parametr ρ a ψ algoritmu byly totožně zvoleny na 0.05. Parametr q_0 je zvolen jako 0.9 z toho důvodu, aby se do řešení s velkou pravděpodobností vložil uzel s nejlepším poměrem profit ku vzdálenosti.

Tab. 3: Parametry algoritmu ACS

mravenci	ρ	ψ	q_0	τ_0
5	0.1	0.1	0.9	5

Algoritmus SAILS s vytvořením počáteního řešení pomocí ALS má stejné parametry jako algoritmus SAILS a ACS. Čtvrtinu času přiděleného pro výpočet se vytváří počáteční řešení pomocí ACS, zbylý čas je pro výpočet SAILS.

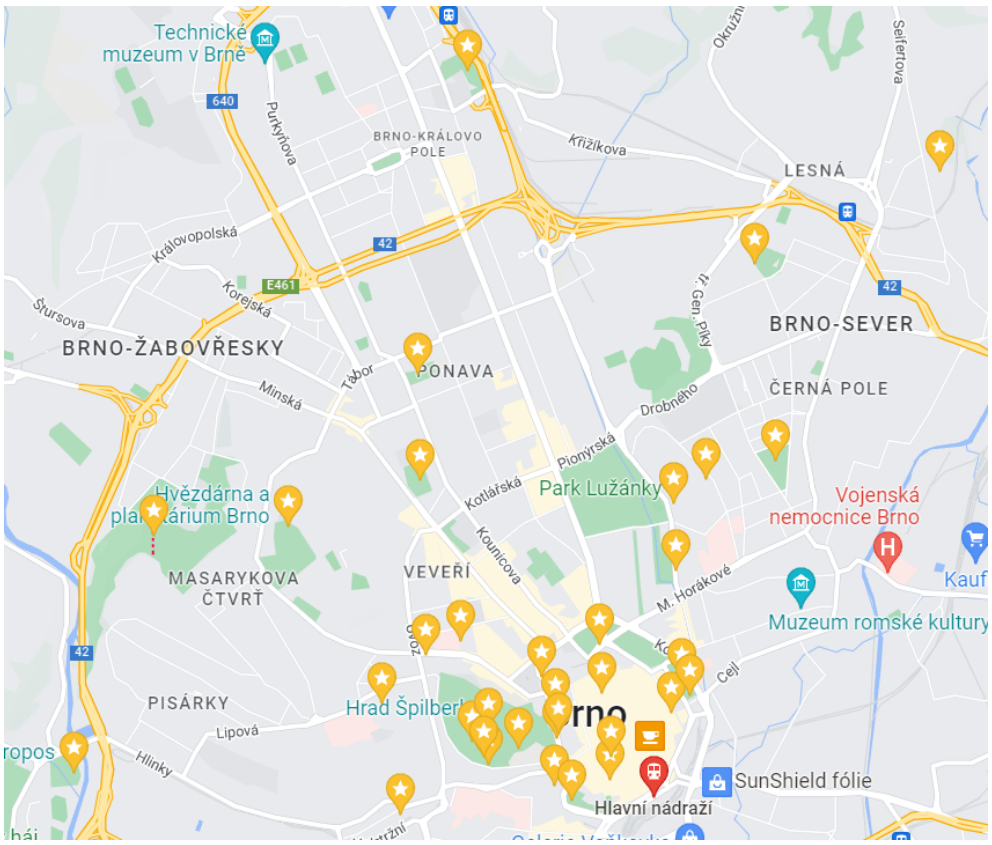
Vzorový příklad

Testovaný případ je následující. Zvolené město je Brno, jako startovní a koncový bod zvoleno Brno hlavní nádraží. Vlakové nádraží je zvoleno, jelikož se nachází téměř centru města, v okolí jsou jak hotely tak další zastávky dopravy (autobusové nádraží Zvonařka, zastávka autobusů u Grandhotelu).

Místa zájmu byla hledána v poloměru 5 km okolo počátečního místa. Zahrnuty byly všechny kategorie (Museum, Planetarium, Public Art, ZOO, Aquarium, Art Gallery, Spiritual Center, Landmark and Outdoors). Doba setrvání na všech místech byla ponechána na 60 minut.

Počet nalezených míst byl nastaven na 50. Bohužel v daném rozsahu bylo nalezeno pouze 27 ohodnocených míst. Ostatním 23 místům zájmu bylo hodnocení změněno pomocí aplikace na hodnotu 5 (nejhorší hodnocení místa zájmu službou Foursquare bylo 5.9, proto voleno hodnocení nižší).

S takto zadanými místy zájmu a počátečním respektive koncovým bodem výletu byl vytvořen graf s časovými údaji o času přechodu mezi danými místy zájmu. Zvolen byl pěší mód. Místa zájmu zobrazená na mapě lze vidět v obr. 10. Z obrázku lze usuzovat, že zvolený příklad je náročný v tom ohledu, že velké množství



Obr. 10: Zobrazení míst zájmu službou Google Maps [vlastní zpracování]

míst zájmu se nachází v centru města. Navíc má téměř polovina míst zájmu stejné hodnocení nastavené na 5.

Testování probíhalo pro různé hodnoty časového budgetu běhu algoritmů a vyhodnoceno bylo celkové hodnocení trasy. Aplikace byla vyzkoušena na sestavení výletu pro 3 a 4 dny. Výlety měly postupně začít vždy v pondělí. Každý den měl výlet trvat 8 hodin a začínat v 10:00.

Výpočet algoritmů byl postupně nastaven na 5 s, 15 s a 30 s. Každý algoritmus byl na výše popsané úloze spuštěn desetkrát. Celkové hodnocení výletů a celkové trvání výletů bylo zaneseno do tabulek.

Pro třídenní výlet jsou výsledky celkového užitku v tabulce 4. Z tabulky lze vyčíst, že nejlepší průměrnou hodnotu má pro všechny doby výpočtu algoritmus SAILS. Pouze pro čas výpočtu nastavený na 5 s vykazuje SAILS naopak nejhorší průměrné výsledky. Nejlepší výsledky pro čas 5 s má algoritmus ACS.

Jestliže porovnáme výsledky průměrné hodnoty profitu za jeden den, můžeme konstatovat, že rozdíly jsou nepatrné. Pouze pro čas výpočtu 5 s mírně vyniká hodnota algoritmu ACS. Naopak pro průměrné výsledky doby běhu algoritmů 30 s je rozdíl mezi nejmenší a největší hodnotou pouze 0.08.

Nejlepší celkový výsledek byl získán pomocí algoritmu ACS. Hodnota užitku turisty je 150.4. Celková doba takového výletu je 86 152 s. Maximální čas, který bylo možné trávit na cestě je pro třídenní výlet 86 400 s. To znamená, že pro nejlepší dosažený výsledek je téměř úplně dosažen maximální délka výletu, tedy pro každý den 8 h. Výsledný plán třídenního výletu, který maximalizuje profit turisty na vzorovém příkladě, je uveden v tabulce 7. Trasa výletu je naznačena na snímcích obrazovky aplikace na obr. 11.

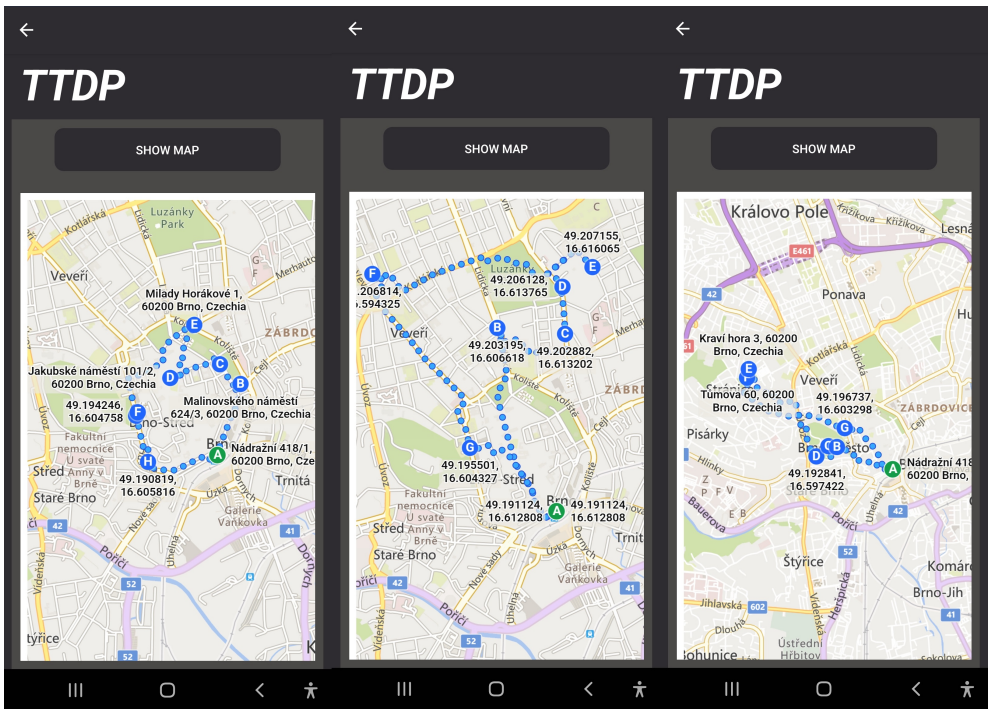
Tab. 4: Celkový profit třídenního výletu

čas výpočtu [s]	algoritmus	průměr	nejlepší	nejhorší	průměr za den
5	ILS	145.06	147.2	141	48.35
	SAILS	144.95	146.5	143	48.32
	ACS	145.92	149.9	143.2	48.64
	SAILS+ACS	145.17	147.3	142.6	48.39
15	ILS	146.62	147.4	144.5	48.87
	SAILS	146.71	147.8	145.7	48.90
	ACS	146.70	148.4	144.7	48.90
	SAILS+ACS	146.32	148.2	144.7	48.77
30	ILS	147.30	148.4	146.2	49.10
	SAILS	147.54	148.5	146.7	49.18
	ACS	147.44	150.4	146.2	49.15
	SAILS+ACS	147.39	149	145.7	49.13

Pro srovnání je ještě uvedena tabulka 5 s hodnotami celkové doby výletu skrze všechny dny. Výsledky jsou u všech algoritmů obdobné. Tabulka naznačuje, že algoritmy ve všech případech vytvořily takové cesty, které téměř naplnily celkový povolený čas výletu pro všechny dny.

Druhá testovaná úloha je totožná s první, rozdílem je pouze požadavek na vytvoření čtyřdenního výletu. Do tabulce 6 jsou zaneseny celkové údaje o deseti testování maximalizace celkových užitků pro čtyřdenní výlet ve městě Brno. Pro nejmenší čas výpočtu bylo dosaženo nevětšího zisku profitu pomocí alogritmu ACS. Podobným rozdílem si vedl nejlépe i pro čas výpočtu 15 s. Pro nejdelší časový výpočet 30 s si vedl také nejlépe ACS, vlastní implementace algoritmu SAILS ovšem postupně vykazovala obdobně dobré výsledky.

Nejmenší průměrný profit pro výpočty 5 s a 15 s dosahovala implementace algoritmu ILS. Pro čas 30 s pak měl nejhorší průměrný výsledek algoritmus ACS+SAILS. U této navržené modifikace algoritmu SAILS je zajímavé, že pro kratší doby výpočtu návrhu optimální turistické trasy mírně převýšila průměrné výsledky vlastní implementace algoritmů ILS a SAILS.



Obr. 11: Trasa třídenního výletu [vlastní zpracování]

Výsledky průměrných profitů za jeden den pro deset testů každého algoritmu vyšly podobně. Nejlepšího průměrného profitu pro turistu dosáhl algoritmus ACS.

Porovnání hodnot celkového času výletů lze spatřiv v tabulce 8. Podobně jako u třídenního výletu jsou celkové časy strávené na výletech pro všechny algoritmy podobné. Z údajů o průměrné době výletu za jeden den lze usuzovat, že pro všechny algoritmy jsou výsledné časy blízké maximální délky trasy pro jeden den.

Maximální profit pro turistu u čtyřdenního výletu je 189.2. Celková doba strávená turistou na cestě respektive prohlížením míst zájmu se rovná 114 773 s. Obdobně jako u třídenního výletu je to doba podobná maximální možné době čtyřdenního výletu 115 200 s, tedy čtyřikrát 8 h.

Jestliže porovnáme tabulku s plánem trasy třídenního výletu (tabulka 7) s tabulkou plány čtyřdenního výletu (tabulka 9), můžeme si povšimnout, že plány pro třídenní výlet jsou obsažené v plánech čtyřdenního výletu. Konkrétně pondělní trasy jsou totožné stejně tak jako úterní. Středeční trasa třídenního výletu je přesunuta na čtvrtek. Toto přesunutí na jiný den je možné, jelikož neporušuje žádné otevírací doby míst zájmu. Čtyřdenní výlet se tak od třídenního liší pouze přidanou trasou ve středu.

S těmito informacemi lze hodnotit implementace zvolených algoritmů. Bohužel implementace algoritmů ILS a SAILS, neposkytla na daných reálných příkladech nejlepší výsledky. U těchto implementací lze spatřit problém v době výpočtu. Jak se ukázalo, výpočet turistické trasy je značně náročný. U reálných aplikací je nutné

Tab. 5: Celkový čas třídenního výletu

čas výpočtu [s]	algoritmus	průměr [s]	nejdelší [s]	nejkratší [s]	průměr za den [s]
5	ILS	83809.91	85559	82057	27936.64
	SAILS	84495.18	85923	82432	28165.06
	ACS	84921.36	86244	82336	28307.12
	SAILS+ACS	84313.82	86253	83079	28104.61
15	ILS	84070.18	85279	83018	28023.39
	SAILS	84699.36	85894	83379	28233.12
	ACS	84805.55	86110	82856	28268.52
	SAILS+ACS	85192.91	87148	83675	28397.64
30	ILS	84462.73	85590	82808	28154.24
	SAILS	84067.64	85756	82783	28022.55
	ACS	84599.55	86152	82216	28199.85
	SAILS+ACS	84426.00	85443	83383	28142.00

kontrolovat i více časových oken u míst zájmu. To odpovídá požadavku, jestli je možné navštívit místo zájmu v daný den.

Druhým faktorem ovlivňujícím čas výpočtu je určitě zvolená vývojová platforma a programovací jazyk. Interpretovaný programovací jazyk, jakým je C#, je obecně pomalejší než programovací jazyky kompilované.

Naopak navržený algoritmus využívající hejnové inteligence se ukázal jako poměrně kvalitní způsob řešení návrhu optimální turistické trasy pro menší počet míst zájmu. Poskytuje solidní výsledky při relativně krátké době výpočtu.

Vyzkoušený způsob rozšíření algoritmu SAILS o počáteční řešení algoritmem ACS se neprokázalo jako výrazné zlepšení, které by nabízeo zlepšení v oblasti návrhu optimální trasy.

8.0.2 Zvolené prostředky a výsledná aplikace

Platforma .NET a programovací jazyk C# nemohou konkurovat v rychlosti kompilovaným programovacím jazykům jako C či C++. Výběr byl ovšem spíše preferenční s ohledem na vytvoření reálně použitelné aplikace.

Volba mapové API Bing Maps je vhodná pro daný příklad. Služba Route Matrix API fungovala bez problému, odezva je i přes velký počet požadavků na server relativně rychlá. Zobrazení map s vytvořenými návrhy je vcelku přehledné, SDK pro vývoj webových aplikací je vcelku příjemné.

Služba Foursquare není v České republice tolik známá. V rámci hledání ohodnocených míst zájmu se projevil tento nedostatek. Alternativou je jistě Google Places která je v Česku populárnější. Nevýhodou je nutnost mít placený účet. I přes výhrady v rámci lokalizace v Česku je Foursquare služba vhodná pro daný druh apli-

Tab. 6: Celkový profit čtyřdenního výletu

čas výpočtu [s]	algoritmus	průměr	nejlepší	nejhorší	průměr za den
5	ILS	181.71	185	180.1	45.43
	SAILS	181.49	183.9	180.4	45.37
	ACS	185.77	186.7	185	46.44
	SAILS+ACS	183.00	186.2	180.5	45.75
15	ILS	182.12	185.5	179.1	45.53
	SAILS	182.30	184.5	181.5	45.58
	ACS	186.49	189.2	184.4	46.62
	SAILS+ACS	183.29	186.2	181.2	45.82
30	ILS	183.16	185.3	180.3	45.79
	SAILS	185.36	186.6	183.9	46.34
	ACS	186.85	187.7	185.9	46.71
	SAILS+ACS	183.08	185.5	180.5	45.77

kací. Kladně lze jistě hodnotit dokumentaci a možnost vyzkoušet službu s možností vygenerovat si kód pro dané hledání v několika programovacích jazycích.

Výsledná aplikace byla navržena tak, aby uživatel měl co největší kontrolu nad volbou vytvoření návrhu turistické trasy. Dalším návrhem pro práci může být například ukládání uživatelských preferencí, více kategorií míst či možnost zvolit si přesné datum příjezdu a odjezdu.

Tab. 7: Plán třídenního výletu maximalizující užitek turisty

den	trasa
pondělí	Brno hlavní nádraží (10:00) - Obelisk (10:08) - Městský park (11:18) - Chodník podél hřiště gymnázia na Botanické (12:49) - Björnsonův sad (13:59) - Sady Národního odboje (15:08) - Park Lužánky (16:29) - Brno hlavní nádraží (17:56)
úterý	Brno hlavní nádraží (10:00) - Žárovky (10:05) - Náměstí 28. října (11:27) - Schreberovy zahrádky (12:41) - Náměstí SNP (14:01) - Vila Tugendhat (15:20) - Vila Löw-Beer v Brně (16:29) - Brno hlavní nádraží (17:58) -
středa	Brno hlavní nádraží (10:00) - Dům umění města Brna (10:07) - Kostnice u sv. Jakuba (11:12) - park Koliště (12:18) - Janáčkův chodníček (13:27) - Moravská galerie v Brně (14:40) - Socha Masaryka (15:42) - Kryt 10-Z (16:46) - Brno hlavní nádraží (17:58)

Tab. 8: Celkový čas čtyřdenního výletu

čas výpočtu [s]	algoritmus	průměr [s]	nejdelší [s]	nejkratší [s]	průměr za den [s]
5	ILS	110716.27	113368	107938	27679.07
	SAILS	110656.27	114082	108047	27664.07
	ACS	113100.55	114008	111619	28275.14
	SAILS+ACS	111499.36	113082	109538	27874.84
15	ILS	110454.27	112930	106004	27613.57
	SAILS	110817.73	113982	109184	27704.43
	ACS	112873.91	114773	109697	28218.48
	SAILS+ACS	111783.00	114294	109877	27945.75
30	ILS	111036.91	113865	108543	27759.23
	SAILS	113003.09	114024	110903	28250.77
	ACS	112706.55	114372	111321	28176.64
	SAILS+ACS	111214.91	117362	108073	27803.73

Tab. 9: Plán čtyřdenního výletu maximalizující užitek turisty

den	trasa
pondělí	Brno hlavní nádraží (10:00) - Obelisk (10:08) - Městský park (11:18) - Chodník podél hřiště gymnázia na Botanické (12:49) - Björnsonův sad (13:59) - Sady Národního odboje (15:08) - Park Lužánky (16:29) - Brno hlavní nádraží (17:56)
úterý	Brno hlavní nádraží (10:00) - Žárovky (10:05) - Náměstí 28. října (11:27) - Schreberovy zahrádky (12:41) - Náměstí SNP (14:01) - Vila Tugendhat (15:20) - Vila Löw-Beer v Brně (16:29) - Brno hlavní nádraží (17:58) -
středa	Brno hlavní nádraží (10:00) - Moravské Zemské Muzeum (10:05) - Hrad Špilberk (11:18) - dětské hřiště (Pellicova) (12:19) - Hvězdárna a planetárium (13:44) - Park (Kraví hora) (15:06) - Uměleckoprůmyslové muzeum (16:46) - Brno hlavní nádraží (17:59)
čtvrtek	Brno hlavní nádraží (10:00) - Dům umění města Brna (10:07) - Kostnice u sv. Jakuba (11:12) - park Koliště (12:18) - Janáčkův chodníček (13:27) - Moravská galerie v Brně (14:40) - Socha Masaryka (15:42) - Kryt 10-Z (16:46) - Brno hlavní nádraží (17:58)

9 ZÁVĚR

V rámci této diplomové práce byly teoreticky rozebrány aspekty návrhu optimální turistické trasy. Jelikož se jedná o problém, který nemá jasné zadání ani řešení, existují různé pohledy jak na návrh turistické trasy nahlížet. V rešeršní práci jsou tak rozebrané optimalizační modely, které lze použít pro návrh optimální turistické trasy. Důraz je kladen zejména na Orienteering Problem, nejvíce na Team Orienteering Problem with Time Windows.

V kapitole 4 jsou připomenuty již existující řešení návrhu optimální turistické trasy. Zmíněny jsou algoritmy založené na hejnové inteligenci (ACS), algoritmy využívající lokální prohledávání (ILS, SAILS, KSA) nebo zajímavý příklad návrhu optimální trasy s řešením využívajícím rozšířený Dijkstrův algoritmus.

V praktické části byly vytvořeny implementace algoritmů pro řešení návrhu optimální turistické trasy. Jde o algoritmy iterovaného lokálního prohledávání (ILS), hybridizaci tohoto algoritmu s algorimtem simulovaného žíhání (SAILS) a vlastní návrh algoritmu Ant Colony System (ACS). V rámci vývoje bylo také předložen návrh na rozšíření SAILS právě o ACS, kdy algoritmus ACS by vytvářel počáteční řešení.

Implementace algoritmů byly využity ve vytvořené mobilní aplikaci využívající reálná mapová data a reálná data o místech zájmu. Vytvořená aplikace je funkční, nicméně nabízí prostor pro zlepšení zejména v oblasti vyladění možných chybových stavů.

Na konci závěrečné práce byly implementované algoritmy zhodnoceny. Dále byla přednesena diskuze o zvolených prostředcích a vytvořené mobilní aplikaci.

10 SEZNAM POUŽITÉ LITERATURY

[1] Dijkstra, E. W.; aj.: *A note on two problems in connexion with graphs. Numerische mathematik*, ročník 1, č. 1, 1959: s. 269–271.

[2] Gavalas, D.; Konstantopoulos, C.; Mastakas, K.; aj.: *A survey on algorithmic approaches for solving tourist trip deighn problems. J Heuristics*, ročník 20, 2014: s. 291–328, doi:10.1007/s10732-014-9242-5.

[3] Gunawan, A.; Lau, H. C.; Lu, K.: *A fast algorithm for personalized travel planning recommendation.* In *Proceedings of the 11th International Conference of the Practice and Theory of Automated Timetabling*, Udine, Italy: PATAT, 2016, s. 163–179.

[4] Gunawan, A.; Lau, H. C.; Vansteenwegen, P.; aj.: *Well-tuned algorithms for the Team Orienteering Problem with Time Windows. Journal of the Operational Research Society*, ročník 68, č. 8, 2017: s. 861–876, doi:10.1057/s41274-017-0244-1.

[5] Harris, J.; Hirst, J.; Mossinghoff, M.: *Combinatorics and Graph Theory.* Undergraduate Texts in Mathematics, Springer New York, 2010, ISBN 9781441927231.

[6] Kirkpatrick, S.; Gelatt, C.; Vecchi, M.: *Optimization by Simulated Annealing. Science (New York, N.Y.)*, ročník 220, 06 1983: s. 671–80, doi:10.1126/science.220.4598.671.

[7] MacQueen, J.; aj.: *Some methods for classification and analysis of multivariate observations.* In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, ročník 1, Oakland, CA, USA, 1967, s. 281–297.

[8] Montemanni, R.; Gambardella, L.: *An ant colony system for team orienteering problems with time windows. Foundations of Computing and Decision Sciences*, ročník Vol. 34, No. 4, 2009: s. 287–306.

[9] Montemanni, R.; Weyland, D.; Gambardella, L.: *An Enhanced Ant Colony System for the Team Orienteering Problem with Time Windows.* In *2011 International Symposium on Computer Science and Society*, 2011, s. 381–384, doi:10.1109/ISCCS.2011.95.

[10] Rardin, R.: *Optimization in Operations Research.* Pearson, 2016, ISBN 9780134384559.

[11] Speranza, M.; Archetti, C.; Vigo, D.: *Chapter 10: Vehicle Routing Problems with Profits.* 11 2014, ISBN 978-1-61197-358-7, doi:10.1137/1.9781611973594.ch10.

[12] Tlili, T.; Krichen, S.: *A simulated annealing-based recommender system for solving the tourist trip design problem. Expert Systems with Applications*, ročník 186, 2021: str. 115723, ISSN 0957-4174, doi:10.1016/j.eswa.2021.115723.

[13] Vansteenwegen, P.; Gunawan, A.: *Orienteering Problems - Models and Algorithms for Vehicle Routing Problems with Profits.* Springer Nature Switzerland AG, 2019, ISBN 978-3-030-29745-9.

[14] Vansteenwegen, P.; Souffriau, W.; Berghe, G. V.; aj.: *The City Trip Planner: An expert system for tourists. Expert Systems with Applications*, ročník 38, č. 6, 2011: s. 6540–6546, ISSN 0957-4174, doi:doi.org/10.1016/j.eswa.2010.11.085.

[15] Volek, J.; Linda, B.: *Teorie grafů - aplikace v dopravě a veřejné správě.* Pardubice: Univerzita Pardubice, 2012, ISBN 978-80-7395-225-9.

[16] Wörndl, W.; Hefele, A.; Herzog, D.: *Recommending a sequence of interesting places for tourist trips. Information Technology & Tourism*, ročník 17, 03 2017, doi:10.1007/s40558-017-0076-5.

[17] Zhong, J.; Hu, X.; Zhang, J.; aj.: *Comparison of Performance between Different Selection Strategies on Simple Genetic Algorithms.* 01 2005, s. 1115–1121, doi: 10.1109/CIMCA.2005.1631619.

[18] Šeda, M.: *Teorie grafů.* Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky, listopad 2003.

11 SEZNAM PŘÍLOH

1: Přiložený archiv 2022_DP_Benda_Tomas_200604.zip.
Zdrojový kód aplikace.
Soubor test.xlsx s výsledky testů.
Soubor Ctime.txt s dalšími pokyny.