TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta přírodovědně-humanitní
a pedagogická

Optimalizace trasy závodu – aplikace simulovaného žíhání

Bakalářská práce

Studijní program: B1101 Matematika
Studijní obory: Matematika se zaměřením na vzdělávání
Tělesná výchova se zaměřením na vzdělávání

Autor práce: **Josef Šůma**
Vedoucí práce: Mgr. Čeněk Jirsák
Katedra aplikované matematiky

Liberec 2021

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta přírodovědně-humanitní
a pedagogická

Zadání bakalářské práce

Optimalizace trasy závodu – aplikace simulovaného žíhání

Jméno a příjmení: **Josef Šůma**
Osobní číslo: P18000282
Studijní program: B1101 Matematika
Studijní obory: Matematika se zaměřením na vzdělávání
Tělesná výchova se zaměřením na vzdělávání
Zadávající katedra: Katedra aplikované matematiky
Akademický rok: **2020/2021**

Zásady pro vypracování:

Cílem práce je navhrnout a implemetovat algoritmus simulovaného žíhání na zobecnění úlohy obchodního cestujícího a vyřešit modelový problém inspirovaný horolezeckým závodem Jizerský Qaker.
Student se seznámí s grafovou teorií, speciálně s klasickou NP-těžkou úlohou obchodního cestujícího. Dále se seznámí se stochastickým optimalizačním algoritmem simulovaného žíhání. V praktické části student aplikuje simulované žíhání na plánování trasy závodu – zobecnění úlohy obchodního cestujícího. Algoritmus implemetuje v programovacím jazyce R.

Rozsah grafických prací:
Rozsah pracovní zprávy: 40
Forma zpracování práce: tištěná/elektronická
Jazyk práce: Čeština

Seznam odborné literatury:

[1] V. Černý, Thermodynamical approach to the traveling salesman problem: An effcient simulation algorithm, Journal of Optimization Theory and Applications, 45 (1985), pp. 4151.
[2] P. Fajgl, O. Simm, and M. Vrkoslav, Jizerské hory: Horolezecký průvodce, NH SAVANA,2009.
[3] J. Matoušek and J. Nešetřil, Kapitoly z diskrétní matematiky, Karolinum, V Praze, 2010.
[4] M. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, Optimization by simulated annealing, SCIENCE, 220 (1983), pp. 671 -680.

Vedoucí práce: Mgr. Čeněk Jirsák
Katedra aplikované matematiky

Datum zadání práce: 8. ledna 2021
Předpokládaný termín odevzdání: 1. května 2022

L.S.

prof. RNDr. Jan Picek, CSc.
děkan

doc. RNDr. Miroslav Koucký, CSc.
vedoucí katedry

V Liberci dne 26. dubna 2021

Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Jsem si vědom toho, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má bakalářská práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

15. července 2021 Josef Šůma

Optimalizace trasy závodu - aplikace simulovaného žíhání

Abstrakt

Práce řeší aplikaci algoritmu simulovaného žíhání pro nalezení optimální kružnice v úplném grafu. Motivací pro úlohu je optimalizace trasy závodu Jizerský Qaker. Úkolem závodníků je v omezeném čase 22 hodin zdolat co nejvíce vylosovaných skal. Tyto skály jsou obodovány dle oblasti, ve které se nachází. Vyhrává dvojice, která v tomto časovém úseku jako první nasbírá co nejvíce bodů. Matematicky tato úloha vede k nalezení optimální kružnice v úplném grafu. Optimální v našem případě znamená, že nepřekročí daný čas a má maximalní hodnocení.

Klíčová slova: Simulované žíhání, problém obchodního cestujícího, teorie grafů, diskrétní matematika

Race route optimization - application of simulated annealing

Abstract

This thesis uses the application of the simulated annealing method to find an optimal circle in a graph. The motivation for this task is the optimization of the route of the competition Jizerský Qaker. Competitors have to climb on as many selected rocks as possible within a time frame of 22 hours. These rocks are rated according to the area where they lie. The winning team is the team that earns the most points in this time limit. Mathematically this task leads to the finding of an optimal cycle in a graph. An optimal cycle means that the cycle will not exceed the given time limit and will have the most points.

Keywords: simulated annealing, Travelling salesman problem, Graph theory, discrete mathematics

Poděkování

Rád bych poděkoval především vedoucímu mé práce, panu magistru Čeňku Jirsákovi za jeho ochotu a čas, který mi věnoval, a tím výraznou mírou přispěl k tvorbě této práce. Dále bych chtěl poděkovat Petru Svobodovi, organizátorovi závodu Jizerský Qaker, za jeho rady a pomoc s objasněním pravidel.

Obsah

1 Úvod

Tato práce se věnuje navržení a implementování algoritmu simulovaného žíhání na úlohu podobnou problému obchodního cestujícího a vyřešení modelového problému inspirovaného horolezeckým závodem Jizerský Qaker. Cílem této práce je najít ideální trasu, kterou by se měl člověk vydat, pokud chce vyhrát závod. Tento závod dvojic se koná v Jizerských horách. Cílem závodu je za daný časový interval nasbírat co nejvíce bodů. Je vylosováno 50 skal a jsou obodované dle oblasti, kde leží od 1 do 3 bodů. Více o tomto závodě se dočtete v kapitole 5.

Našim cílem je nalézt za pomoci algoritmu simulovaného žíhání nejlepší možnou kružnici v grafu. Nejlepší možná kružnice pro náš problém znaména, že nasbírá co největší počet bodů a její čas bude co nejmenší, ale vždy musí být menší než časový limit závodu 22 hodin. Tento náš úkol je variací na problém obchodního cestujícího, který je jedním z NP-těžkých problémů. Problém obchodního cestujícího spočívá v nalezení nejkratší hamiltonovské kružnice v grafu, to je taková, která prochází všemi body grafu. Obchodník má z daného města objet všechna daná města, mezi kterými vedou silnice a známe jejich délky. Naše variace spočívá v tom, že kružnice nemusí nutně procházet všemi body graf.

Nalézt optimální trasu v problému obchodního cestujícího pro malý počet měst není až tak těžké, neboť si vypíšeme všechna možná řešení a vzájemně je mezi sebou porovnáme. Problém však nastává při zvyšování počtu měst, tedy vrcholů (bodů), které musíme propojit hranami. Pokud přidáme k n městům ještě jedno, které chceme projít, je možností $(n + 1)$krát více. To znamená, že složitost roste faktoriálně, protože musíme vyzkoušet všechny možnosti. Největším problémem tedy není nalézt nejkratší hamiltonovskou kružnici, protože to by pro kterékoli konečné číslo bylo možné a snadné, ale bude to trvat dlouho. Například pro 50 měst by bylo 49! možností všech hamiltonovských kružnic. Pro představu tento faktoriál je přibližně roven $6 \cdot 10^{62}$, což by znamenalo pro současný plánovaný nejvýkonnější počítač, který má být dokončen roku 2023 a zvládne za jednu sekundu vyřesit $2 \cdot 10^{18}$operací - $1,9 \cdot 10^{37}$ let práce.

Jde tedy o to, za jak dlouho k tomuto výsledku dojdeme. Problém tedy spočívá v nalezení časově efektivního algoritmu, který by nám toto umožnil. Právě díky neznalosti časově efektivního algoritmu používáme pro přibližné řešení numerické metody. My v této práci budeme používat algoritmus simulovaného žíhání, jehož název je odvozen z metalurgie a je pro náš výpočet ideální. Simulované žíhání má hlavní výhodu v tom, že dokáže v „rozumném“ čase nalézt „rozumné“ suboptimální řešení. Tento algoritmus je blíže popsán v kapitole 4.

Dalším výstupem této práce je implementace algoritmu v programovacím jazyce

R. Tento algoritmus je přílohou této práce.

Základní pojmy shrnuje kapitola 2. Kapitola 3 se věnuje problému obchodního cestujícího, jeho vymezení a stručné historii. Dále zde definujeme a objasníme pojem výpočetní složitost a P versus NP problém. Obecnému algoritmu simulovaného žíhání se věnuje kapitola 4. V poslední 5 kapitole si představíme konkrétní závod, náš použitý algoritmus, postup při jeho tvorbě a porovnání s výsledky závodu z roku 2017.

2 Graf a jeho náležitosti

V této kapitole si popíšeme základní názvosloví, které následně budeme v celé práci používat. Grafy je určen:

- množinou bodů
- spojnic mezi některými dvojicemi bodů

Množiny bodů mohou být jak konečné, tak i nekonečné. My však v naší práci budeme pracovat jen s konečnými množinami bodů. Pomocí různých grafů lze vyjádřit mnoho věcí nejen v matematice, informatice, ale i jiným kupříkladu praktických úlohách. Jedním z nejznámnějších příkladů jsou mapy, kdy křižovatky představují vrcholy a silnice mezi nimi hrany. Dalším příkladem mohou být rodokmeny, kdy se nám vždy spojují příbuzní z jedné a druhé strany. Vrcholy těchto grafů je možné nazývat i **uzly**.

2.1 Teorie grafu

Definice 1 *Graf G je uspořádaná dvojice (V,E), kde V je nějaká neprázdná množina a E je množina dvouprvkových podmnožin množiny V.*

Písmeno V značí vrcholy z anglického slova vertex. Stejně tak E pochází z angličtiny ze slova edge a značí hrany.

V případě, že chceme na daném grafu, řekněme G, vyznačit jeho vrcholy V a hrany E, píšeme $G = (V, E)$. Chceme-li konkrétně poukázat na množinu vrcholů, zapíšeme V(G), obdobně na množinu hran V(E). Nyní, když máme definovaný graf, můžeme si ukázat, jak budeme nazývat speciální typy grafů. My budeme uvažovat jen jednoduchý, neboli obyčejný graf, který je typický tím, že nemá vícenásobné hrany ani smyčky. Tedy každý vrchol je v grafu právě jednou a každá hrana maximálně jednou.

Definice 2 Úplný graf *je graf značený K_n, kde $n \geq 1$, ve kterém*

$$V = \{1, 2, \ldots, n\}, \quad E = \{(i, j); i, j \in V \wedge i \neq j\} \tag{2.1}$$

V tomto grafu nalezneme hrany mezi všemi vrcholy.

Definice 3 *Řekneme, že graf H je podgrafem grafu G, jestliže $V(H) \subseteq V(G)$ a $E(H) \subseteq E(G)$.*

Definice 4 *Nechť G = (V,E) je graf. Posloupnost* $(v_0, e_1, v_1, e_2, \ldots, e_n, v_n)$ *nazveme* **sled** *grafu G.*

Definice 5 Cesta P_n *je speciální sled, ve kterém se neopakují vrcholy ani hrany.*

Každý bod je zde spojen hranou pouze se dvěma dalšími body. Cesta je tedy posloupnost vrcholů a hran.

Ve sledu se narozdíl od cesty mohou vrcholy i opakovat. V grafu G existuje cesta z vrcholu i do vrcholu j právě když v G existuje sled z v_i do v_j. Každá cesta tvoří sled. Sled z v_i do v_j, který má nejmenší možnou délku je vždy cesta.

Definice 6 Souvislým grafem *nazveme takový graf, který má každé dva vrcholy spojené cestou.*

Definice 7 *Speciálním příkladem sledu je* **kružnice** C_n*. Kružnice je uzavřená posloupnost vrcholů spojených hranami. Každý bod je spojen hranou pouze se dvěma dalšími a do počátečního bodu se dostaneme z bodu koncového.*

Kružnici, která prochází všemi vrcholy grafu se říká **Hamiltonovská**. Graf je tedy hamiltonovský, právě tehdy když obsahuje kružnici, která hranami spojuje všechny jeho vrcholy.

Název je odvozen od Williama Rowana Hamiltona, který byl irským fyzikem, matematikem a astronomem. Hamilton vymyslel hlavolam, ve kterém bylo za cíl projít po hranách vrcholy pravidelného dvanáctistěnu.

2.2 Matice vzdáleností

Definice 8 *Nechť G = (V,E) je souvislý graf. Pro vrcholy v,v' definujeme číslo* $d_G(v, v')$ *jako délku nejkratší možné cesty z v do v' v grafu G. Číslo* $d_G(v, v')$ *se nazývá vzdálenost vrcholů v a v' v grafu G.*

Funkce $d_G : V \times V \to \mathbb{R}$, kterou nazýváme metrika grafu G má tyto vlastnosti:

1. $d_G(v, v') \geq 0$ a $d_G(v, v') = 0$ právě když v = v'
2. (symterie) $d_G(v, v') = d_G(v', v)$ pro každou dvojici vrcholů v, v'
3. (trojúhelníková nerovnost) $d_G(v, v'') \leq d_G(v, v') + d_G(v', v'')$ pro každou trojici v, v', v'' vrcholů z V

Každému zobrazení $V \times V$ do $\mathbb{R}$ s vlastnostmi 1-3 se říká metrika na množině V.

Za splnění předešlých podmínek nazveme dvojici (V,d_G) **metrickým prostorem**.

V našem případě máme orientovaný graf, který má ohodnocené hrany dle jejich délky. Vzdálenost mezi vrcholy je minimum součtu ohodnocení hran při cestě z jednoho vrcholu do druhého. Tyto hodnoty si zapíšeme do matice. Této matici se říká matice vzdáleností. Nejprve si vrcholy očíslujeme $V = 1, 2, \ldots, n$. Nyní můžeme

vytvořit matici o rozměrech $n \times n$, do které budeme vkládat jednotlivé vzdálenosti (délky hran) mezi vrcholy. Platí zde, že na pozici A(j,k) je buď kladné číslo, pokud spojení mezi body existuje, nebo nekonečno, pokud dané dva body nelze spojit. Matice vzdáleností má na hlavní diagonále nuly, tedy $A(j,k) = 0 \Leftrightarrow j = k$.

V našem případě bude možno se dostat z každého vrcholu do všech ostatních, a tak v matici vzdáleností budou jen kladá čísla a na hlavní diagonále nuly. Do tabulky tedy budeme vkládat vzdálenosti mezi jednotlivými vrcholy grafu.

3 Historie problému obchodního cestujícího

Problém obchodního cestujícího je klasickou úlohou, která se řeší již řadu let. Následující historický přehled čerpá z webové stránky [1].

Tento problém vychází již z 18. století, kdy se začali objevovat jeho náznaky při formulování teorie grafů, která je součástí diskrétní matematiky. V roce 1736 Leonhard Euler provedl nejspíše první zmínku ve svém díle „Solutio problemas ad geommetriam situs pertinensis“. Variace problému zde obsahovala 7 mostů ve městě Královec. Uprostřed města protékala řeka, město tedy leželo na dvou březích a mezi nimi, uprostřed řeky, byly ještě dva ostrovy. Město mělo tedy 4 části spojené sedmi mosty. Eulerův úkol spočíval v nalezení cesty, při které měl projít všemi částmi města za využití všem mostů právě jednou. Euler tento problém vyřešil tím, že prohlásil, že tato úloha nemá řešení, označil ji jako neřešitelnou a zformuloval obecné podmínky pro řešení obdobných úloh [9]. Euler řešil i další úlohu zvanou „Problém jezdce“, ve kterém má jezdec (kůň) na šachovnici za úkol projet všechna pole a vrátit se zpět na své místo odkud začínal.

Po těchto myšlenkách Eulera nepřišel žádný vědec více než 100 let na další důležité poznatky z oblasti teorie grafů. Až v roce 1847 německý fyzik Gustav Kirschhoff při výpočtu proudů v elektrických sítích začal řešit úlohy pomocí soustav algebraických rovnic a přispěl k dalším zmínkách o teorii grafů.

Roku 1878 se James Joseph Sylvester objevil teorii grafů, která přetrvala dodnes. Vědci se zabývali problémem čtyř barev, kterým se zabýval především Francis Guthrie. Problém spočíval v ověření, že každou mapu je možné nabarvit čtyřmi barvami tak, že žádný ze sousedních států nebude mít stejnou barvu. To se mu však nepovedlo ověřit, a tak problém v matematice přetrvával. Problém se podařilo vyřešit až v roce 1976, kdy Kenneth Appel a Wolfgang Haken vyřešili tento problém s pomocí počítačového programu. Tento počítačový program po 1200 procesorových hodinách vykázal, že je opravdu možné jakoukoli mapu obarvit čtyřmi barvami podle zadání. Důkaz mnoho matematiků neuznávalo, neboť nebyl nikdo schopen ho přímo zkontrolovat.

Po Sylvesterovi se začalo daleko více matematiků o teorii grafů zajímat a začaly vycházet knihy právě o této teorii. Například Claude Berge, francouzský matematik, roku 1958 napsal knihu „Théorie des Graphes et ses Applications“, norský matematik Oystein Ore „Theory of Graphs“ [17] a Frank Harary „Graph Theory“ [11]. [14]

Nyní zpátky k samotnému problému obchodního cestujícího. Po Eulerově úloze jezdce v 18. století se téměř 200 let nepodařilo nikomu udělat nějaký větší pokrok v této úloze. Až roku 1930 rakouský matematik Karl Menger publikoval svou studii „Botenproblem“. O této úloze se následně zmiňovali například Hassler Whitney

a Merrill Flood, kteří ji využili při optimalizaci v zemědělství během výzkumu pro firmu RAND. Detailní informace o jejich studii jsou zaznamenány v knize „On the history of combinatiorial optimization till 1960“ [18] od Alexandera Schrijvera.

Největší pokrok v tomto problému mají na svědomí Gaorge Dantzig, Ray Fulkerson a Selmer Johnson. Tito američtí ekonomové v roce 1954 vydali knihu, ve které uvedli metody řešení úloh obchodního cestujícího a předvedli to na úloze se 49 městy. Města byla vhodně zvolena z každého státu v USA a navíc bylo přidáno hlavní město Washington D. C.. Mezi všemi těmito městy byly změřeny vzdálenosti po silnicích. Zajimavostí je, že z původního záměru nalézt nejkratší cestu mezi městy bylo vypoštěno sedm měst na západním pobřeží, ale i přes to výsledná cesta vedla přes ně. Takže tito ekonomové dokázali vyřešit úlohu pro 42 měst, ale výsledná cesta vedla přes sedm dříve vypuštěných měst. Dalo by se tedy říct, že se jim podařilo vyřešit úlohu pro 49 měst.

Na jejich dílo se podíváme blíže. V případě, že budeme uvažovat všechny možné cesty (hrany) mezi každými dvěma městy, dostaneme 861 hran pro model s 42 městy, který řešili. Obecně počet hran můžeme vypočítat pomocí vzorce $n \cdot (n-1)/2$, kde n vyjadřuje počet měst. Tyto cesty (hrany) vložili do matice vzdálenosti, kterou můžeme znát například z autoatlasů. Stejně jako tuto matici vytvořili i další, kam ovšem zapisovali pouze 1 a 0. Jedničku pokud hranu využijí a nulu pokud nikoli. Ve výpočtu využili metody lineárního programování a čtyřech kroků, které jim v něm pomohly.

1. Základním předpokladem byl neorientovaný graf, který zjednodušil popis cyklů a tím zkrátil výpočetní čas.

2. Za lineární byly považovány pouze vzdálenosti mezi městy, to vede k minimalizování celkové vzdálenosti a zjednodušení výpočtu vzdálenosti mezi dvěma body. Tento krok vychází z definice lineární vzdálenosti, díky které je úloha tvořena jednoduchými soustavami rovnic.

3. V některých případech a pro urychlení výpočtu může dojít k přidání dalších omezujících podmínek nebo změně cyklu pomocí vizuální kontroly mapy.

4. Po spojení všech měst se již řešení blíží tomu hledanému – optimálnímu. Ještě se využijí kombinatorické metody k eliminaci cyklů, které neeliminovaly předchozí podmínky.

Tato metoda obsahuje i jisté lineární podmínky tak, aby se zamezilo tvorbě cyklů, které by se při výpočtu vraceli stále do jednoho bodu. Podrobné informace o jejich postupu jsou uvedeny v knize [3].

Roku 1955 vymysleli George Morton a Ailsa Land výpočetní metodu 3-Opt. O rok později, roku 1956, publikoval Merrill Meeks Flood dílo „The travelling salesman problem“ [7]. V této analýze popsal Flood heuristické metody, které měly pomoci vypočítat úlohu ve velmi krátkém čase. Jednou z jeho nejvýznmnějších metod patří Metoda nejbližšího souseda 2-Opt.

V průběhu 50. let 20 století se rozšířil zájem o využívání výpočetní techniky pro řešení matematických úloh, a tak F. Bocks v roce 1958 vytvořil matici s 10 městy

a vypočítal úlohu za pomoci počítače IBM 650 pomocí metody 3-Opt. S postupem času se heuristiky využívaly a testovaly na různých počítačích.

Roku 1964 Robert L. Karg společně s Geraldem L. Thompsonem za pomoci heuristických metod vypočítali úlohu s 57 městy. O rok později Shen Lin využil těchto metod a dokázal vypočíst tuto úlohu pro 105 měst, což bylo v dané době téměř neuvěřitelné.

V 70. letech 20. století publikovali R. M. Karp a M. Held dílo s názvem „The travelling salesman problem and minimum spanning trees“ [15], kde předvedli možnosti výpočtu Problému obchodního cestujícího pomocí 1-tree relaxace, kam dokázali i přiřadit různé váhy jednotlivým uzlům. Tuto metodu využili pro řešení úlohy s 42 městy a 57 městy. V obou případech byli úspěšní.

Roku 1976 nastal další pokrok ve výpočtech, kdy P. Miliotis dokázal využít možnosti celočíselného lineárního programování a poté na Problém obchodního cestujícího aplikoval metodu větvení a mezí, při které využíval omezujících podmínek s nerovnostmi v podgrafech.

Od této doby se problémem stále zabývalo spousta matematiků. Vyvíjeli a zlepšovali heuristické metody, metody na výpočet celočíselných úloh, hlavně metody řezných nadrovin a lineárních relaxací. P. Miliotis ve své práci s názvem „Using cutting planes to solve the symmetric travelling salesman problem“ [16] využil právě tyto poznatky. Na konci 20. století napsali M. Grötschel a O. Holland „Solution of large scale symmetric travelling salesman problems“ [10], což bylo jedno z nejvýznamnějších děl.

Tabulka 3.1: Vývoj počtu měst [1]

Rok	Počet měst	Řešitelé
1954	49	G. Dantzig, R. Fulkerson, S. Johnson
1971	64	M. Held, R. M. Karp
1975	67	P. M. Camerini, L. Fratta, F. Maffioli
1977	120	M. Grötschel
1980	318	H. Crowder, M. W. Padberg
1987	532	M. Padberg, G. Rinaldi
1987	666	M. Grötschel, O. Holland
1987	2392	M. Padberg, G. Rinaldi
1994	7397	D. Applegate, R. Bixby, V. Chvátal, W. Cook
1998	13509	D. Applegate, R. Bixby, V. Chvátal, W. Cook
2001	15112	D. Applegate, R. Bixby, V. Chvátal, W. Cook
2004	24978	D. Applegate, R. Bixby, V. Chvátal, W. Cook, K. Helsgaun

3.1 Výpočetní složitost

Výpočetní složitnost (nebo také asymptotická složitost) je způsob rozřazení algoritmů podle jejich časové náročnosti. Určuje se zde operační náročnost jednotlivých

algoritmů tak, že se pozoruje jak se chová algoritmus při změně (zvětšení) počtu vstupních dat. Tato klasifikace algortimů do tříd říká, že algoritmus jakékoli třídy bude vždy pomalejší než ten třídy předchozí bez ohledu na výkonnost počítače.

Značí se různě, ale základními typy jsou velké O nebo též Omicron notace, Omega notace a Théta notace. Omicron notace $O(N)$ vyjadřuje složitost, se kterou se výpočet provádí vzhledem k rostoucímu vstupu. $N \in \mathbb{N}$ a vyjadřuje velikost vstupu.

Definice 9 *Mějme reálnou funkci f(x), O(f(x)) je třída funkcí, pro kterou platí*

$$g(x) \in O(f(x)) \Leftrightarrow \exists x_0 > 0, c > 0, \forall x > x_0 : g(x) < c \cdot f(x). \tag{3.1}$$

Omicron notace popisuje horní hranici rychlosti růstu dané funkce.

Existují různé třídy funkcí. Každá vyšší třída obsahuje funkce z nižší třídy. Třídy funkcí jsou například následující

$$O(1) \subset O(log(N)) \subset O(N) \subset O(Nlog(N) \subset N^k \subset l^N \subset N! \subset N^N, \tag{3.2}$$

kde $k, l \in \mathbb{R}$ a $k, l > 1$.

Problémem je v případě n^{100} a 2^n, kdy bude pro $n = 10\,2^{10}$ menší než 10^{100}. Proto nutně neznamená, že vždy musí algoritmus z třídy vyšší rychlejší než algoritmus ze třídy nižší.

Omega notace popisuje spodní hranici rychlosti růstu dané funkce. Je tedy opakem Omicron notace. V definici 9 pouze změníme znaménka nerovnosti na pravé straně na $g(x) > c \cdot f(x)$, jinak je definice naprosto stejná jako pro Omicron notaci. Kombinací obou zmíněných tříd je notace Théta Θ .

$$g(x) \in \Theta(f(x)) \Leftrightarrow \exists x_0 > 0, c_1 > 0, c_2 > 0, \forall x > x_0 : c_1 \cdot f(x) < g(x) < c_2 \cdot f(x) \tag{3.3}$$

Základními typy jsou například $O(N)$, které se říká lineární a ta určuje, že doba výpočtu algoritmu se zvyšuje lineárně s rostoucími daty na vstupu. Tedy kolikrát se zvětší vstup, tolikrát se prodlouží doba výpočtu. Další třídou složitosti je $O(N^2)$ nazývaná kvadratickou. Tady se doba výpočtu zvětší kvadraticky oproti zvětšení dat. Nejjednodušší třídou je $O(1)$, kde nezáleží na velikosti vstupu. Tento algoritmus potřebuje pro výpočet pořád stejnou dobu bez ohledu na množství vstupních dat.

Třída složitosti P je tvořena algoritmy $O(n^k)$, kde $k \in \mathbb{R}$. Tyto algoritmy též nazveme polynomiálně omezenými.

Složitější problémy nazveme NP. Ovšem pro nalezení řešení tohoto problému nemusí existovat polynomiálně omezený algoritmus. Tyto problémy definují **třídu složitosti NP**. Třída NP se dělí na NP-těžké a NP-úplné problémy. Stále nevyřešenou otázkou je rovnost či nerovnost tříd P a NP.

3.1.1 P versus NP

Jedná se o velmi důležitý problém v teoretické informatice. Problém spočívá v otázce, zda jsou třídy složitosti P a NP totožné. Tedy jednoduše řečeno, zdali každý problém, jehož správnost dokáže počítač rychle vyřešit, dokáže počítač samotnou úlohu rychle vyřešit. Tento problém je jedním ze sedmi „Millenium Prize Problems“, což jsou

problémy vybrané Clayovým matematickým institutem a za vyřešení každého z nich, tedy i tohoto nabízí tento institut 1 000 000$. Termín rychle, který je použit v této definici vyjadřuje existenci vyřešení algoritmu v polynomiálním čase.

NP-úplné problémy

NP-úplné problémy jsou problémy z části NP. Mají tři hlavní vlastnosti. Lze jej převést na jiný NP-úplný, nedá se řešit v polynomiálním čase a výsledek se dá snadno ověřit. Jejich řešení však lze ověřit v polynomiálním čase. Toto plyne z definice NP-úplného problému, která říká, že každý NP problém je možno rychle zredukovat na NP-úplný problém. Příkladem NP-úplného problému může být rozklad na prvočísla, kdy získání řešení zabere dlouho dobu. Tato doba se prodlužuje s rostoucí hodnotou čísla. Ověření tohoto řešení je ovšem velmi rychle – stačí čísla mezi sebou vynásobit.

Můžeme si zde uvést i jeden příklad, obyčejné sudoku. Pokud by někdo sudoku neznal, jedná se o to, že hráč dostane částečně vyplněnou mřížku čísel, typicky 9×9 políček rozdělenou ještě na 9 částí 3×3 políčka. Hráč má za úkol vepsat do mřížky čísla 1 až 9 tak, aby se v žádném řádku, sloupci ani buňce neopakovala. Jakékoli řešení, ke kterému hráč dojde je velmi snadno ověřitelné a se zvětšující se mřížkou čas na kontrolu roste polynomiálně (pomalu). Oproti tomu všechny dodnes známé a vymyšlené algoritmy potřebují se zvětšující se mřížkou exponenciálně více času. Sudoku tedy spadá do třídy NP, protože ho lze rychle zkontrolovat, ale nezdá se, že by bylo P, tedy rychle řešitelné.

NP-těžké problémy

Tyto problémy se zakládají na velké složitosti výpočtu a také složitosti ověření správnosti výsledku. Za předpokladu, že $P \neq NP$, potom tyto NP-těžké problémy nemohou být vyřešeny v polynomiálním čase. Příkladem NP-těžkého problému je například problém obchodního cestujícího, kdy nalezení nejkratší hamiltonovské kružnice nejspíše nelze vyřešit v polynomiálním čase. Ani ověření výsledku nelze vyřešit v polynomiálním čase, neboť musíme provést všechny dílčí výsledky znovu, abychom mohli říci, že jsme našli nejkratší hamiltonovskou kružnici.

NP = P

Odpověď na otázku, že $P \neq NP$, by znamenala, že existují problémy, které je těžší vypočítat než ověřit jejich správnost. K této variantě se přiklání většina vědců, avšak stále se najdou zastánci možnosti $P = NP$ a snaží se tuto rovnost dokázat. V případě vyřešení tohoto problému, ať už rovnost či nerovnost, by to byl obrovský pokrok a mělo by to obrovské důsledky nejen v matematice, ale třeba i v teorii her, umělé inteligenci, ekonomii, zpracování multimédií, a tak dále.

Vědci také zjistili, že pokud by se přišlo na řešení nějakého NP problému, dalo by se ho využít i k řešení dalších NP problémů. Tedy pokud by se našlo jedno řešení, mohli bychom prohlásit $P \neq NP$. Bohužel prozatím se na nic takového nepřišlo i přes to, že tento problém je celkem známý už řádku let.

3.1.2 Historie

Ikdyž k definici tohoto problému došlo až v 80. letech 20. století, již předtím existovaly náznaky souvisejících problémů, obtížnosti dokazování a možných důsledků. Roku 1955 matematik John Nash napsal dopis národní bezpečnostní agentuře USA, ve kterém spekuloval, že prolomení dostatečně složitého kódu bude vyžadovat časově exponenciální délku klíče. Pokud by se tato spekulace potvrdila, znamenalo by to, že $P \neq NP$, protože navrhovaný klíč jde snadno ověřit v polynomiálním čase.

V roce 1956 napsal Kurt Gödel Johnu von Neumannovi dopis, ve kterém se ho ptal, zda může být důkaz řešen v kvadratickém nebo lineárním čase. Přitom zdůraznil jeden z nejdůležitějších důsledků, že pokud by tomu tak bylo, mohl by být automatizován objev matematických důkazů.

Přesně problém P versus NP vyjádřil Stephen Cook, americko-kanadský počítačový vědec a matematik v roce 1971 ve své seminární práci „Složitost postupů prokazování věty“. Stejné vyjádření měl i Leonid Levin, sovětsko-americký matematik a informatik, bez závislosti na Cookovi v roce 1973. Proto se tento problém občas přezdívá Cook-Levinova věta.

William Gasarch od roku 2002 provedl tři výzkumy mínění vědců souvisejících s problémem P versus NP. Výzkum prokázal, že díky stále delší době, kdy se nikomu nepodařilo složit důkaz rovnosti, či nerovnost, roste podpora názoru $P \neq NP$. Zatímco v roce nerovnosti věřilo v roce 2002 60% tázaných, v roce 2012 už to bylo 83% a v roce 2019 dokonce 88%, pričemž pokud se berou jen odborníci, je důvěra v $P \neq NP$ 99% [12].

3.2 Náš problém

V našem případě se budeme věnovat variaci na problém obchodního cestujícího, neboť nebudeme mít za úkol nalézt Hamiltonovskou kružnici, ale optimální (suboptimální) cestu grafem takovou, abychom dosáhli co nejlepších výsledků. Naším úkolem je nalézt takovou trasu, ve které dvoučlenný tým nasbírá za daný čas co nejvíce bodů a stihne se vrátit do startovacího místa – Hausmanka u Kozy. Tedy naše kružnice nutně nemusí obejít všechny vrcholy, protože je možné, že by to s naší rychlostí přesunu ani nebylo možné.

My budeme skály uvažovat jako vrcholy v grafu a je tedy naším úkolem nalézt takovou kružnici v grafu, kterou stihneme obejít v daném čase a nasbíráme v ohodnocených vrcholech co nejvíce bodů. Projití této kružnice také musí zabrat co nejmenší čas, protože při shodě bodů mezi soupeřícími týmy rozhoduje právě čas. Počátečním a zároveň cílovým bodem je Hausmanka U Kozy.

4 Algoritmus simulovaného žíhání

Algoritmus simulovného žíhání je pravděpodobnostní optimalizační algoritmus[5] pro hledání globálního extrému funkce, v našem případě kružnice a je založen na Boltzmannově rozdělení [13]. Algoritmus simulovaného žíhání dokáže v nějakém pro nás přijatelném čase nalézt vyhovující suboptimální řešení. Tento algoritmus tedy nezaručuje nalezení optimálního řešení, protože narozdíl od metody hrubou silou neprohledává všechny možné způsoby. Výhoda algoritmu spočívá v tom, že pokud budeme tento algoritmus provádět dostatečně pomalu, tak velice pravděpodobně nalezneme optimální řešení [4].

Definice 10 Optimální řešení *je takové řešení, jehož čas je menší než limit a zároveň má co největší bodové ohodnocení. Mezi řešeními se stejným bodovým ohodnocením je optimální to, které má nejmenší čas.*

Tento algoritmus je jeden z mnoha používaných pro řešení problému obchodního cestujícího. Algoritmus nejprve vytvoří náhodnou cestu, ve které následně podle zvolené návrhové funkce změní cestu na navrhovanou. V našem případě prohazuje hrany mezi náhodnými vrcholy. Tímto prohozením získáme navrhovanou cestu, kterou následně buďto přijmeme, nebo nikoli a postupujeme znovu stejně. Algoritmus řídí zejména parametr teploty T, který se mění. Na počátku algoritmu je nastaveno T tak, aby se přijímala různá řešení, tedy i horší. Jak algoritmus postupuje, T se snižuje tak, aby se stále zmenšovala pravděpodobnost přijetí horšího řešení. Dále má velkou roli v algoritmu právě návrhová hustota a také žíhací schéma.

Výhoda tohoto algoritmu spočívá v tom, že nepřijímá jen lepší řešení. Pokud je hodnota navrhované cesty lepší (má větší hodnocení) přijmeme ji vždy. Přijme i v dané chvíli horší řešení s danou pravděpodobností rovnicí (4.2). Toto přijetí někdy horšího řešení má předcházet od uvíznutí v lokálním extrému.

Definice 11 *Nechť Ω je prostor všech cest, potom pravděpodobnost, že se nacházíme v konkrétní cestě je dána vzorcem udávaným Boltzmannovým rozdělením:*

$$P(X = x) = \frac{1}{Z(\beta)} e^{-\beta H(x)} \tag{4.1}$$

Toto rozdělení se někdy též označuje jako Gibsovo rozdělení.

Tento vzorec udává pravděpodobnost, že se nacházím ve stavu x. Parametr β je kladný parametr Boltzmannova rozdělení, který ve fyzice udává převrácenou hodnotu teploty a v našem případě ho budeme brát také tak. Normalizační konstanta je

$Z(\beta) = \sum_{x \in \Omega} e^{-\beta H(x)}$. Rovnice pochází z fyziky, kde námi používané $H(x)$ se říká energii stavu x.

Název žíhání již napovídá, že se jedná o odvozeninu z metalurgie, kde se ochlazuje zahřátý kov pomalu tak, aby vznikly velké krystaly s malými defekty. V zahřátém kovu jsou totiž atomy volné a nahodile se pohybují ve stavech s vyšší energií. Během ochlazování dochází k tomu, že se atomy postupně zpomalují a koncentrují se v místech s nízkými energiemi.

V algoritmu tyto energie nahrazují funkce, v našem případě funkce hodnocení. Při každém kroku algoritmu dochází k tomu, že se vybírají stavy z blízkého okolí a porovnává se jejich hodnota a s určitou pravděpodobností, která je závislá na daném rozdílu a parametru teploty, dojde ke změně stavu. Při každém ochlazení se zmenší pravděpodobnost přijetí horšího stavu.

4.1 Popis algoritmu

Tato úloha je variací na problém obchodního cestujícího, protože v problému obchodního cestujícího se musí projít všechny body grafu. V našem případě ideálně bychom také chtěli projít všechny, ale může se stát, že to nebude možné díky časovému omezení. Proto je naším úkolem maximalizovat funkci, jejímž definičním oborem jsou všechny možné kružnice začínající a končící v daném bodě (Hausmanka U Kozy). Pro správné fungování našeho algoritmu potřebujeme následující.

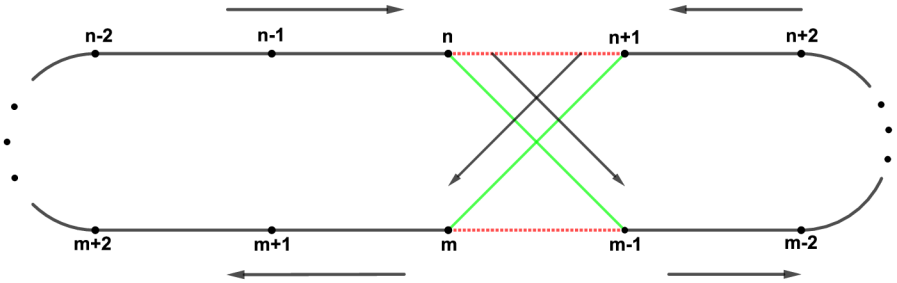
- Funkce, kterou chceme minimalizovat, v našem případě ohodnocení cesty.
- Návrhová funkce – v našem případě používáme prohazování hran v grafu.
- Žíhací schéma neboli posloupnost teplot, při kterých se algoritmus provádí a počtu kroků (kolikrát se vykoná).
- Pravděpodobnost přijetí horší cesty je udána Boltzmannovým faktorem.

Mějme graf o k vrcholech. Algoritmus probíhá následovně:

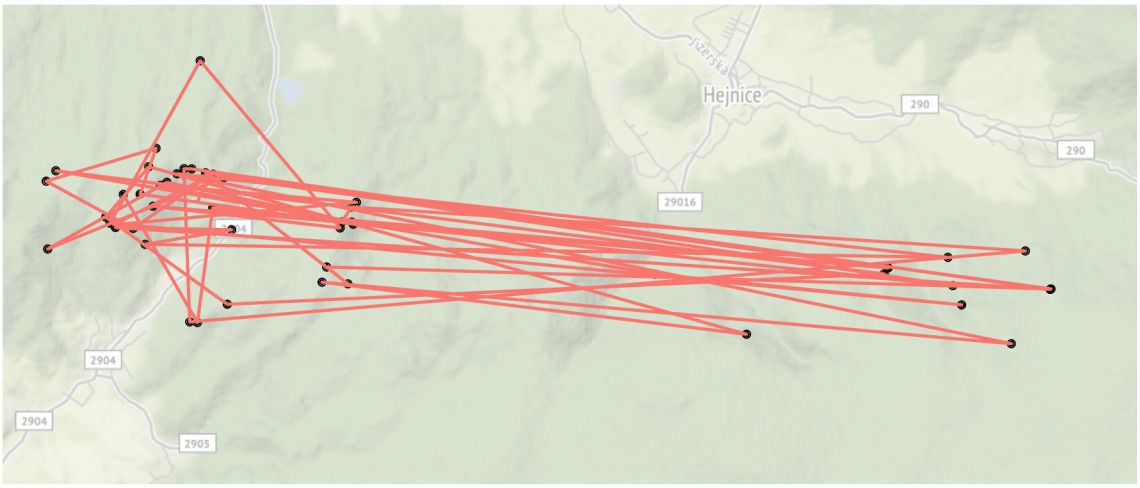
1. Vygenerujeme náhodnou cestu. Tuto cestu si na začátku označíme jako aktuální, v tuto chvíli c_a.
2. Náhodně zvolíme dva vrcholy n a vrchol m z cesty o k vrcholech. Předpokládejme, že platí $n < m$. Navrhneme novou cestu c_i, kde i vyjadřuje určitý krok v našem simulovaném žíhání.

   Pro novou (navrhovanou) cestu potom platí, že posloupnost vrcholů je oproti původní následující (první cesta je cesta původní a druhá je změněná s prohozenými hranami mezi vrcholy.

$$\{1, 2, \ldots, n, n+1, n+2, \ldots, m-2, m-1, m, m+1, \ldots, k\}$$
$$\{1, 2, \ldots, n, m-1, m-2, \ldots, n+1, m, m+1, m+2, \ldots, k\}$$



Obrázek 4.1: Prohození hran



Obrázek 4.2: Graf po náhodném spojení vrcholů

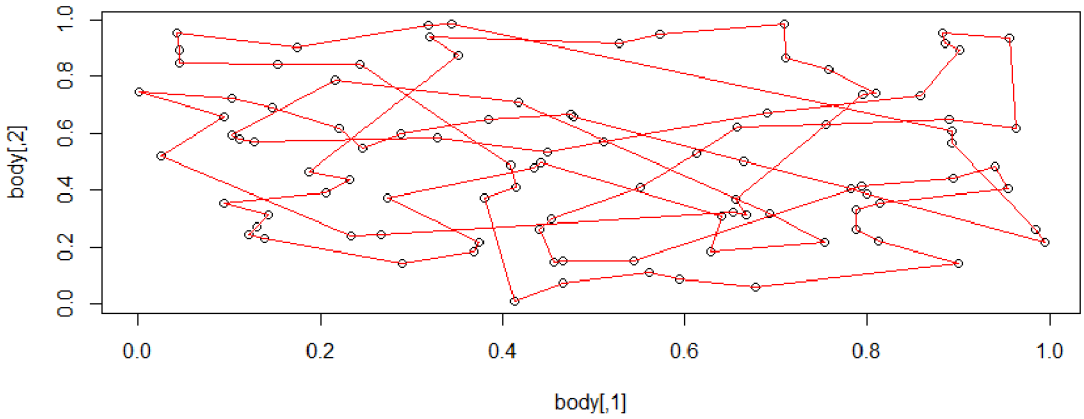
3. Pro tuto novou (navrhovanou) cestu si spočítáme ohodnocení $h(c_i)$.

   Pokud má navrhovaná cesta lepší ohodnocení, než původní aktuální cesta, přijmeme ji a dále budeme jako aktuální cestu uvažovat tuto navrhovanou. Tedy $c_a = c_i$.

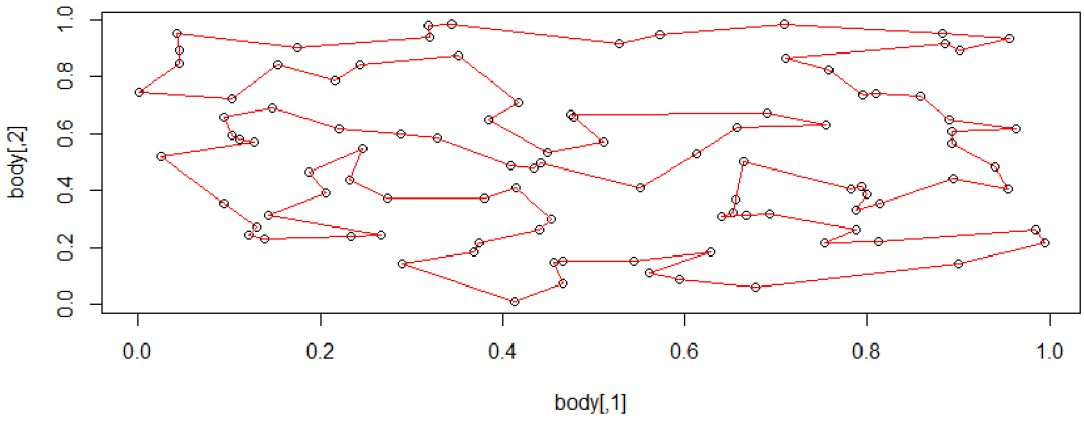
   Pokud však bude mít ohodnocení $h(c_i)$ horší, přijmeme ji právě s pravděpodobností udanou vzorcem podílu dvou pravděpodobností:

$$p(c_a, c_i) = e^{\frac{h(c_i)-h(c_a)}{T}} \tag{4.2}$$

   V této rovnici $p(c_a, c_i)$ vyjadřuje pravděpodobnost přechodu navrhované cesty na aktuální, $h(c_i)$ ohodnocení navrhované cesty a $h(c_a)$ ohodnocení aktuální cesty. Pravděpodobnosti ze vzorce (4.2) vychází z Boltzmannova rozdělení (definice 11). Do vzorce nemusíme vkládat $Z(\beta)$, protože se při podílu zkrátí.



Obrázek 4.3: Graf během práce algoritmu



Obrázek 4.4: Výsledek algoritmu

4. Nyní přecházíme zpět do bodu 2 a celý postup opakujeme podle počtu kroků.

5. Pokud jsme dosáhli počtu kroků v dané teplotě, přesouváme se do nižší teploty a celý postup se od bodu 2 opakuje.

Výsledkem algoritmu je tedy cesta s nejvyšším hodnocením, kterou jsme potkali za celou dobu.

Jak jsme si již ukázali, ve vzorci figuruje teplota. Teplota je daná žíhacím schématem, kde máme postupné snižovaní teploty s příslušnými počty kroků. Pro vytvoření žíhacího schéma neexistuje rigorózní návod, proto se vždy musí přizpůsobit dané úloze.

4.2 Praktické problémy při implementaci algoritmu

Problémy, které jsem při implementaci řešil jsou tři. Jsou to věci, které musíme navrhnout. Prvním problémem je správné sestavení žíhacího schéma tak, aby nebylo moc dlouhé, ale zároveň abychom se nedostali do lokálních extrémů, které nejsou hledanými lokálními. Při chybném sestavení žíhacího schéma, které by ochlazovalo moc rychle, sice dojdeme rychleji k výsledku algoritmu, avšak tento výsledek bývá často zkreslený. Naopak, pokud bychom ochlazovali velmi pomalu, sice bychom měli téměř zajištěno nalezení optimálního řešení, ale tento algoritmus by trval velmi dlouho a blížil by se spíše metodě hrubou silou. Proto je velice důležité zvolit vhodné schéma, které nám umožní v „rozumném“ nalézt „rozumné“ suboptimální řešení, které bude blízko tomu optimálnímu. Při navrhování žíhacího schéma je dalším problémem časová náročnost, kdy se musí vždy nechat algoritmus provádět a až na základě výsledků upravovat a zlepšovat do ideální podoby.

Druhým problémem je navržení vhodné návrhové funkce. Nakonec jsme se rozhodli pro použití návrhové funkce prohazování hran. Prohazovaní hran je jednou z nejpoužívanějších metod při řešení problému obchodního cestujícího. Této metodě se říká 2-opt, o které se poprvé zmínil M. M. Flood roku 1956 [8], ale poprvé ji použil o dva roky déle G. A. Croes [2].

5 Aplikace algoritmu

Výsledkem této práce je program, který určí ideální trasu, kterou pokud závodník v tomto závodě poběží, vyhraje. Jedná se o sportovní soutěž spojující horolezení a turistiku.

Závod vždy začíná v pátek, kdy je sraz v 19:00 v Hausmance U Kozy, nacházející se mezi Oldřichovem v Hájích a Raspenavou. Zde se sejdou všechny závodící dvojice a provádí se losování. Ze předem určených skal (skalních věží) se náhodně vylosuje 50 věží, které jsou obodovány od 1 do 3 bodů dle toho, kde se nacházejí. Losuje se z jedné losovací nádoby. V této nádobě jsou umístěny názvy skalních věží ze 4 oblastí Jizerských hor, konkrétně Srázy a Poledník, Stržový vrch a Kopřivník (tyto skály jsou ohodnoceny jedním bodem), Malý a Velký Štolpich (2 body), Polední kameny s Hlídači (3 body). Soupis všech skalních věží je obsažen v příloze této práce jako „Seznam skal“.

Na seznamu losovaných skal nejsou všechny skály z těchto oblastí, neboť se jedná o legální závod, a tak mají pořadatelé jisté zákazy od Lesů ČR a ochránců přírody. Úkolem každého týmu je za 22 hodin nasbírat co nejvíce bodů a stihnout se vrátit na startovní místo. Start závodu je v pátek ve 20 hodin, tedy závod probíhá přes noc. Závod končí v sobotu v 18:00. Kdo dorazí po časovém limitu, je diskvalifikován. Plánování a seřazení skal je čistě na výběru závodníků. Týmy mají k dispozici soupis vylosovaných skal a papírového horského průvodce [6], neboť na některých místech Jizerských hor není signál, a tak by si nevystačili s mobilními telefony. Kvůli používání papírových průvodců se také stává, že některé skály, které jsou již zaznamenány na webu horosvaz.cz [20], nejsou zařazeny do losování, protože papírový průvodce je vydáván jednou za 10 let. Horosvaz.cz je oficiální webovou stránkou českého horolezeckého svazu, který na tyto stránky umisťuje aktuality a je zde i databáze skal s informacemi o nich. Tým si při zdolání skalní věže zaznamená na seznam vylosovaných skal k dané skále čas jejího zdolání.

Do algoritmu vstupuje časový limit závodu (22 hodin), vrcholy (skalní věže) s jejich konkrétním ohodnocením (1 až 3 body) a matice vzdáleností mezi jednotlivými vrcholy. Pro správné fungování je jen zapotřebí zadat vylosované skály a algoritmus může pracovat.

Soupis skal jsem sepsal podle jejich souřadnic uvedených na webové stránce [19]. Do tohoto seznamu jsem zaznamenal všechny skály, které jsou uvedeny v papírové průvodci [6]. Zároveň jsem ke skalám do dalšího sloupce napsal jejich bodové ohodnocení, které, jak jsem se již zmínil vyplývá z oblasti, kde se nachází. Skály jsem importoval v souboru csv do algoritmu v programovacím jazyce R, kde jsem využil funkce distm z balíčku geosphere, která počítá vzdálenosti mezi dvěma souřadnicově

zadanými vrcholy.

5.1 Parametry vstupující do algoritmu

Do algoritmu vstupují následující čtyři parametry a v následujících odstavcích si představíme, jak jsem k těmto parametrům došel.

5.1.1 Odhad vzdálenosti mezi vrcholy

Vzdálenosti mezi jednotlivými vrcholy vzdušnou čarou neodpovídají skutěčné vzdálenosti. Skutečná vzdálenost je daleko složitější, protože jde o to, jestli mezi vrcholy vede cesta (nebo je houští). Celý problém jsem se rozhodl vyřešit tak, že jsem provedl měření na 20 vrcholech, mezi nimiž je cesta po označené cestě je circa o jednu čtvrtinu delší, a tak tuto vzdálenost ještě násobím koeficientem 1,25.

5.1.2 Odhad časové náročnosti pohybu terénem

Prováděl jsem měření v terénu v těžším kopcovitém terénu a vyšla mi průměrná rychlost chůze 4,5 kilometru za hodinu, avšak s rostoucí kilometrovou zátěží se mi tempo značně zpomaluje, proto jsem usoudil, že tempo, které jsem schopný vydržet celých 22 hodin (nepočítáme spánek), je 3,5 kilometru za hodinu. Také do času započítávám jednotlivé skalní věže, kdy beru, že jejich výstup zabere circa 10 minut, které k času přičítám.

5.1.3 Hodnota cesty

Algoritmus běží podle algoritmu popsaného v kapitole 4.1. Nejprve si necháme vytvořit náhodnou cestu. Cesta je vektor vrcholů v posloupnosti, v jaké je procházíme. Ohodnocení se skládá ze dvou složek. První složkou je hodnota každého navštíveného vrcholu a hodnota c_i je součtem hodnot všech vrcholů, které naše cesta c_i projde.

Abychom rozlišili cesty se stejným hodnocením, budeme brát vpotaz i čas. Čas se vypočítá jako vzdálenost mezi jednotlivými vrcholy děleno rychlostí přesunu. K vypočtení času a hodnocení cesty využíváme pomocné funkce doba cesty a hodnota cesty. K času přidáme koeficient 0,015. Tato hodnota je zvolena tak, aby hodnota cesty měla větší váhu v celkovém hodnocení. K tomuto koeficientu jsem došel několika pokusy. Celkové ohodnocení je tak udáno vzorcem:

$$h(c_i) = hodnota(c_i) + cas(c_i) \cdot 0,015 \tag{5.1}$$

Je velmi pravděpodobné, že bude čas náhodné cesty vyšší, než je limit závodu, protože v této fázi bereme vpotaz cestu skrze všech 50 vrcholů. Existuje mnoho možností, jak se s tímto problémem vypořádat. Nejjednodušším řešením je vypustit vždy poslední vrchol. Prohledat všechny možnosti je velmi časově náročné, a tak jsme se rozhodli, že buďto vezmeme souvislou podmnožinu cesty a nebo odebereme

souvislou podmnožinu cesty. Po každém pokusu ověříme čas a pokračujeme dokud nám cesta časově nevyhovuje.

Tuto cestu si následně zvolíme jako prozatimní nejlepší a její čas si uložíme do proměnné cas_min, hodnotu do hodnota_max a posloupnost vrcholů do proměnné cesta_min. Všechny tyto proměnné potom ještě uložím jako aktuální, se kterými budu dále pracovat, konkrétně čas, aktuální hodnota a cesta.

5.1.4 Návrh žíhacího schématu

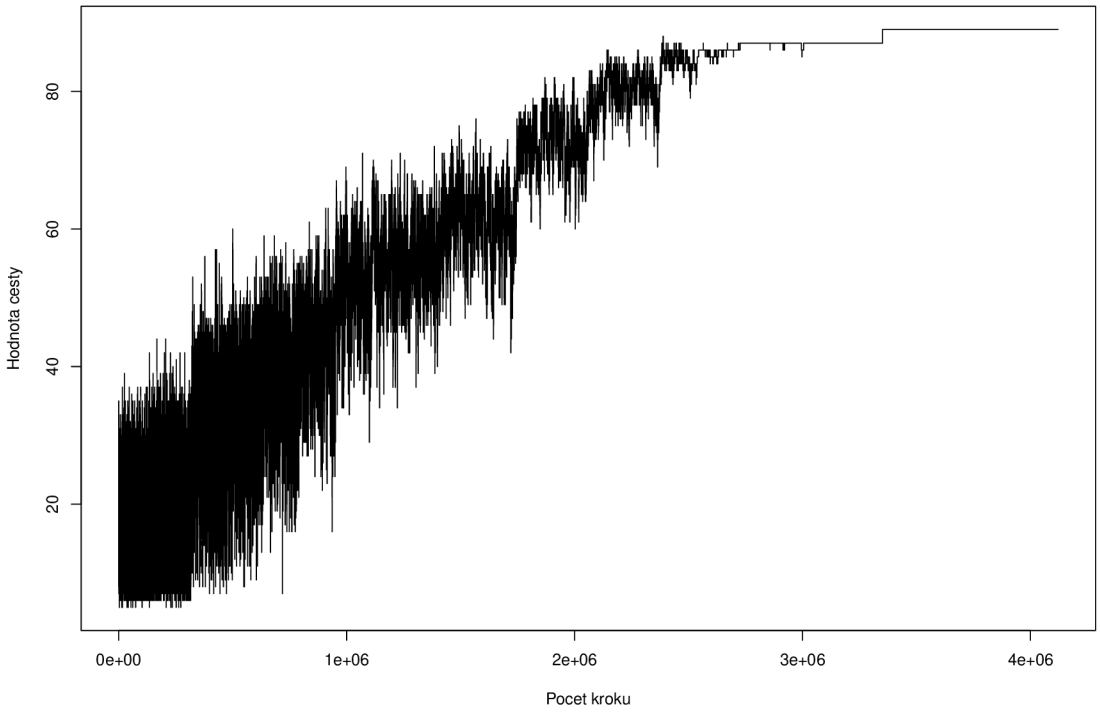
Nyní v algoritmu vytváříme podklady pro simulované žíhání a to konkrétně žíhací schéma. Potíže při sestavování žíhacího schématu jsou popsány v kapitole 4.2. Pro náš konkrétní problém jsem sestavil žíhací schéma následovně. Schéma bylo sestaveno na základě pokusů a osvědčilo se následující:

Tabulka 5.1: Žíhací schéma

Teplota	**Počet kroků**
10	25 000
5	25 000
2	25 000
1,75	25 000
1,5	25 000
1,25	25 000
1	25 000
0,87	50 000
0,75	50 000
0,5	50 000
0,38	50 000
0,25	25 000
0,145	25 000
0,1	50 000
0,05	75 000
0,043	25 000
0,025	50 000
0,01	25 000
0,005	25 000
0,001	25 000

Pro naši úlohu jsme zvolili počet kroků 25000. Samozřejmě s více kroky bychom dostali lepší aproximaci řešení, ale algoritmus už by pro nás nebyl tak využitelný, neboť jedním z požadavků je právě rychlost algoritmu. Na druhou stranu, pokud zmenšíme počet kroků, sice dostaneme výsledek rychle, ale aproximace řešení bude daleko horší. Proto jsme zvolili právě tento počet kroků, při kterém algoritmus nalezne vhodné řešení v přijatelném čase.

Během hledání pro náš příklad optimálního žíhacího schéma jsem pracoval s grafem, který nám znázorňuje aktuální energii algoritmu.



Obrázek 5.1: Graf vývoje aktuálního hodnocení cesty

Toto schéma testoval i s více kroky, abych měl možnost vidět, že při navýšení počtu kroků stoupá ohodnocení téměř pravidelně a dochází ke stejným nebo velmi podobným výsledkům jako při provádění s naším počtem kroků, které lze vykonat v relativně krátkém čase.

Nyní, když máme správně sestavené žíhací schéma, si přejdeme k další části algoritmu. Konkrétně si představíme cyklus, který se pro každou teplotu provádí počtem kroků udaným v žíhacím schéma. V každém cyklu si nejprve zavedeme navrhovanou cestu jako cestu (aktuální). Nyní použijeme naší návrhovou funkci. Necháme algoritmus náhodně zvolit dva vrcholy z naší cesty a prohodíme hrany mezi nimi (jako na obrázku 4.1). Hned poté si necháme vypočítat navrhovaný čas této cesty a navrhované ohodnocení této cesty.

V tento okamžik v cyklu přichází na řadu přijetí navrhované cesty nebo její zamítnutí. Přijetí navrhované cesty je udáno následujícím vzorcem a vychází z rovnice (4.2):

$$p < e^{\frac{h(c_i)-h(c_a)}{teplota}}.$$

Kde p má rovnoměrné rozdělení od 0 do 1.

Vzorec vychází z Boltzmannova faktoru a je zde zastoupeno jak hodnocení cesty, tak její čas (podle 5.1).

Pokud je aktuální hodnota větší než maximální hodnota, uloží se aktuální proměnné do minimálních (potažmo maximálních v případě hodnocení) proměnných. Pokud však nastane rovnost aktuální hodnoty s maximální, porovnáme jejich časy. Pokud bude čas menší než je doposud minimální čas, uložíme si aktuální proměnné obdobně jako v případě většího ohodnocení.

Takto se cyklus provádí při každé teplotě tolikrát, kolikrát je v žíhacím schéma udáno počtem kroků. Teplota se tedy postupně snižuje, s ní se snižuje i pravděpodobnost přijetí horší cesty. Ve chvíli, kdy se provede daný počet kroků v nejnižší teplotě, algoritmus končí a vrací výsledky, kterými jsou výsledná hodnota cesty, konkrétní cesta, její čas a graf.

5.2 Příprava algoritmu

Seznam skal jsem získal z průvodce [6]. K těmto skalám jsem nalezl pomocí Mapy.cz [19] jejich GPS souřadnice a zapsal je do tabulky o čtyřech sloupcích. Tabulka skal je také přílohou této práce.

Díky možnostem využití balíčku pro programovací jazyk R jsem snadno implementoval tabulku s vrcholy a nechal spočítat vzdálenosti mezi jednotlivými vrcholy. Jak již bylo v práci zmíněno, tato vzdálenost byla vynásobena koeficientem $1,25$ viz kapitola 5.1.2.

V našem problému jde o to, že máme zadaných 50 skal a my hledáme optimální kružnici právě mezi nimi. Tedy některé skalní věže při určitém vylosovaní nás nebudou až tak zajímat. Může se stát, že naše cesta bude vést kolem, ale jelikož skála nebyla vylosována, je bez bodového ohodnocení.

Algoritmus si vybírá jen určité položky. Žíhací schéma jsem upravoval na základě grafů, které jsem si nechal vykreslovat, a do kterých jsem si v každém kroku uložil aktuální hodnocení cesty, abych viděl, jak se mi hodnota mění. Příklad tohoto grafu na obrázku 5.1.

Tyto grafy jsem vždy po vykonání algoritmu prohlížel a hledal, kde nastává nějaký skok. Pokud jsem viděl, že najednou hodnota rapidně vzroste, doplnil jsem nějaké teploty mezi původní tak, abych zamezil těmto skokům mezi hodnotami. Na tomto grafu je také hezky vidět, jak funguje algoritmus simulovaného žíhání, kdy ze začátku přijímá téměř všechna možná řešení, ale čím je vykonáno více kroků, tím je nižší teplota a v poslední třetině grafu algoritmus přijímá jen lepší řešení.

V tuto chvíli je algoritmus hotový a funkční.

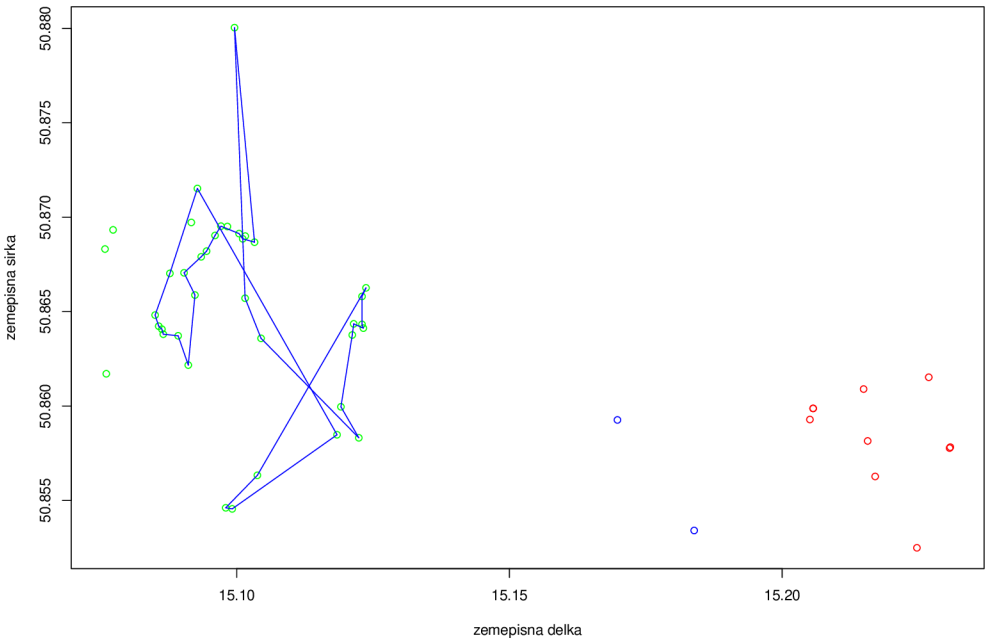
6 Prezentace výsledků

V této kapitole se podíváme již na konkrétní aplikované příklady, k jejichž řešení jsem využil vytvořeného algoritmu. Prvním příkladem je vymyšlený příklad, ve kterém je vidět funkce algoritmu a druhý příklad je spojen se závodem a lze v něm tak pozorovat shodu se skutečností.

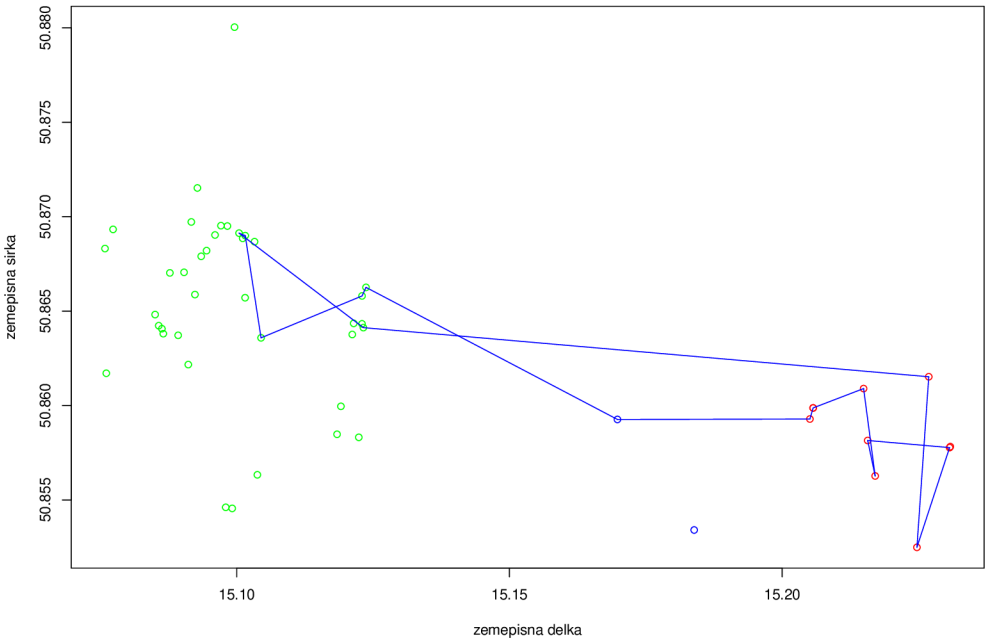
6.1 Modelový příklad

V této kapitole si představíme příklad, kdy je omezený čas (850 minut), aby algoritmus nestihl projít veškeré zadané skály, které jsou vybírány tak, aby bylo na algoritmu rozhodnout, jakou cestou se vydat. Na obrázcích je vidět postup, jakým se výpočet ubíral. Složitost tohoto konkrétního příkladu spočívá v tom, že pouhým pohledem nelze jednoznačně určit právě nejlepší cestu. Proto je tento příklad optimální k prezentaci funkčnosti tohoto algoritmu. Tento postup je znázorněn na obrázku 6.5. Zajímavostí na tomto příkladě je rozložení vrcholů. U startu (Hausmanka U Kozy) se nachází většina vrcholů v jednobodových oblastech a můžeme pozorovat, jak algoritmus pracuje.

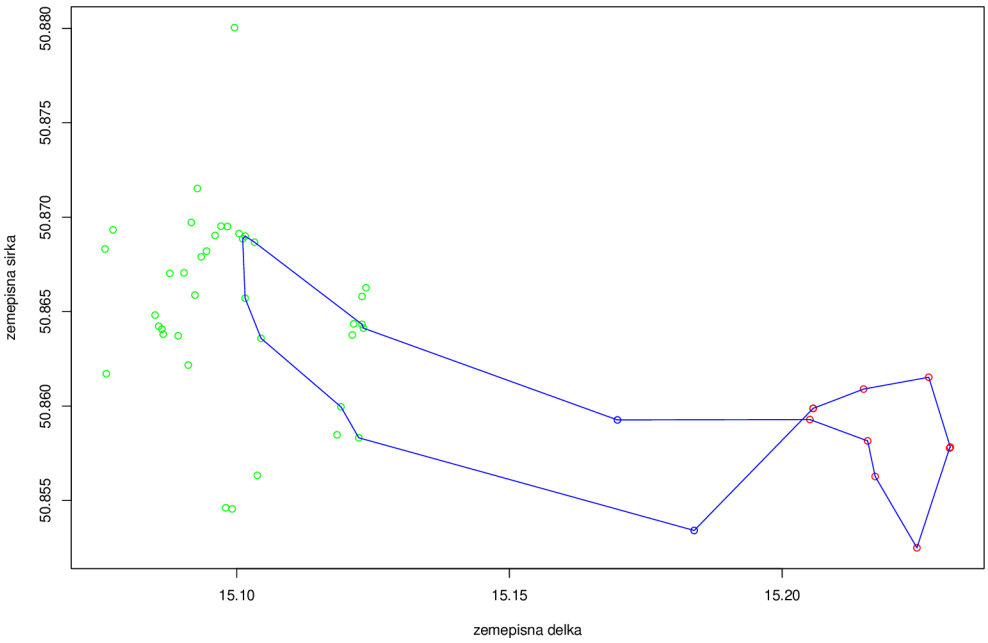
Během práce algoritmu jsem pozoroval, že se původně snažil projít všechny jednobodové vrcholy, ale se snižující se teplotou cesta přesunula a začala do své zvolené cesty zahrnovat tříbodové věže. Konečná cesta pak vede skrze vzdálenější vrcholy. Cesta je znázorněna na obrázku 6.5 a postup jejího vykreslování je znázorněn na obrázcích 6.1, 6.2, 6.3, 6.4. Postup je znázorněn ve 2D měřítku a jen umístěných bodech podle souřadnic nikoli v mapě, aby obrázek byl přehlednější a lépe čitelný.



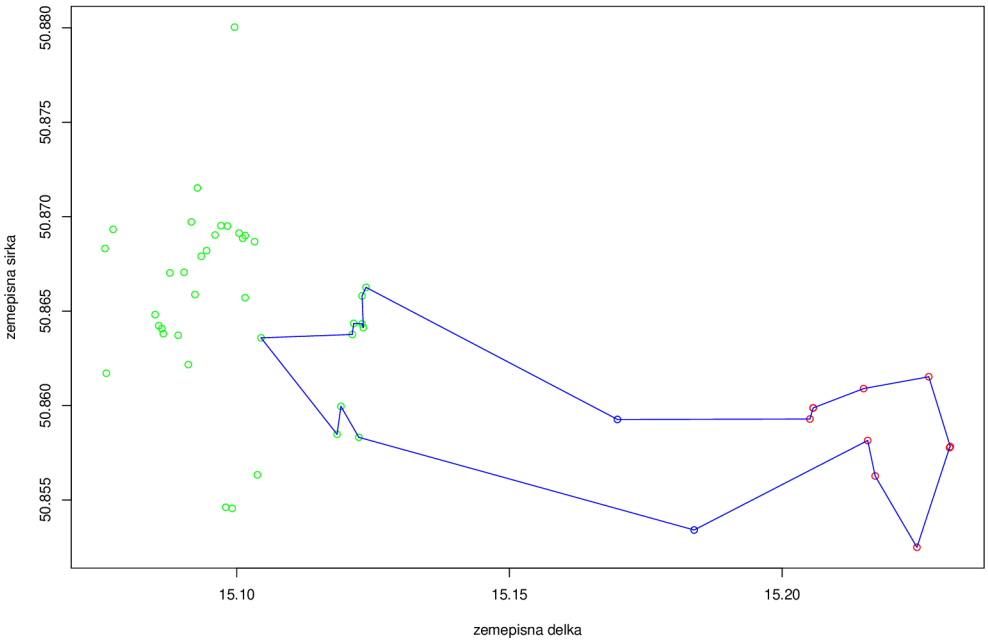
Obrázek 6.1: Zvolená cesta algoritmu při teplotě 2



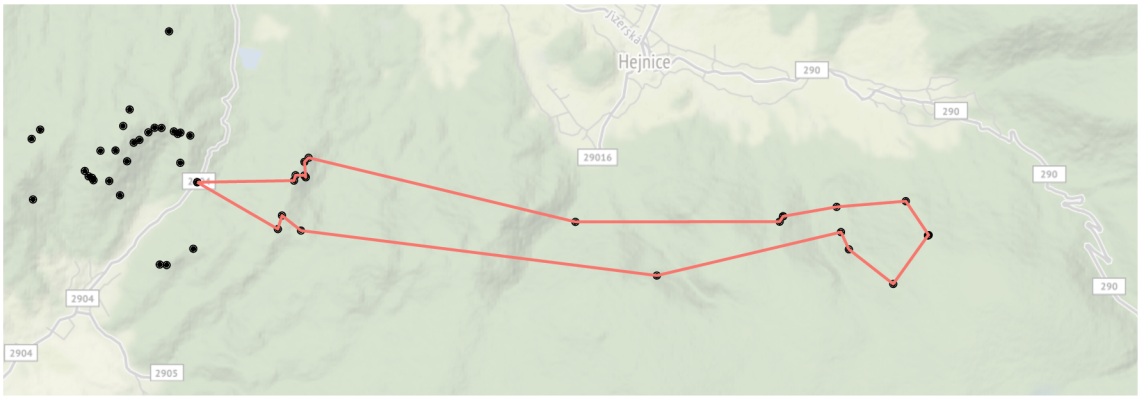
Obrázek 6.2: Zvolená cesta algoritmu při teplotě 1



Obrázek 6.3: Zvolená cesta algoritmu při teplotě 0,25



Obrázek 6.4: Zvolená cesta algoritmu při teplotě 0,001



Obrázek 6.5: Cesta nalezená algoritmem skrze vybrané body pro čas 850 minut

6.2 Porovnání s vítězem závodu 2017

Díky organizátoru Petru Svobodovi, čímž mu ještě jednou za jeho pomoc děkuji, jsem dostal seznam skal vítězné dvojice s názvem týmu Rozvodové řízení. Vzal jsem tedy skály, které byly v roce 2017 vylosované a implementoval jsem je do našeho algoritmu. Tento příklad slouží spíše pro zajímavost, ale i lehkou ukázku, že v jistých pasážích je shodný s postupem vítězů. Berme omezený čas 1320 minut a počet bodů 50.

Jen pro představu, jak vypadá onen záznamový list, který dostanou dvojice na začátku, na Obrázku 6.6 můžeme vidět právě onen list vítězného týmu z roku 2017.

Náš algoritmus dokázal vykreslit cestu, která během časového limitu projde 48 skal z 49 (nedokázal jsem najít souřadnice jedné skalní věže). Jediná skalní věž, kterou algoritmus nezahrnul, je Skalní brána, která se nachází podle soutěžících 10 minut od věže PIC 60. výročí VŘSR. Takže teoreticky by soutěžící měl být schopný obejít všechny skály, za předpokladu průměrné rychlosti přesunu 3,5 kilometru za hodinu, 10 minutám věnovaným každé věži a cestě mezi věžemi o čtvrtinu delšími než je vzdálenost vzdušnou čarou.

V tabulce 6.1 máme vypsané skalní věže v pořadí, v jakém je prošel algoritmus a vítěžná dvojice. Je možné zde zahlédnout právě ono prohození některých cest od dvojice. Toto zapřičiňuje i počítání vzdáleností mezi skalami, kdy předpokládáme, že dvojice vybrala podle nich nejlepší možnou trasu. Ovšem v určitých pasážích se naprosto shodujeme. Na Obrázku 6.7 vidíme kružnici, kterou nalezl náš algoritmus a na Obrázku 6.8 kružnici, kterou se vydala vítězná dvojice.

Vítězové také měli nejspíše dvouhodinovou pauzu v noci, kdy si odpočali a pak pokračovali dále.

Tabulka 6.1: Porovnání algoritmu se skutečným vítězem

Pořadí skal	**Dle algoritmu**	**Dle vítězů**
1	Pic 60. výročí VŘSR	Pic 60. výročí VŘSR
2	Malá frýdlantská věž	Skalní brána
3	Komínové věže	Hradby
4	Stržová plotna	Uhlířova čapka
5	Plotna u Kovadliny	Dvojitá věž
6	Dvojitá věž	Plotna u Kovadliny
7	Uhlířova čapka	Stržová plotna
8	Hradby	Komínové věže
9	Stržová věžička	Malá Frýdlantská věž
10	Rozeklaná skála	Šolcova stěna
11	Koňská plotna	Šolcův jehlan
12	Viklanová stěna	Březová věžička
13	Jizerský kostelík	Lokomotivka
14	Sviní kámen	Emilova věž
15	Netopýří věžička	Kazatelna

16	Štolpišská plotna	Novoroční věž
17	Divoký okraj	Mufloní plotna
18	Jeřábova skála	Dolní hlídač
19	Spárová věž	Střední hlídač
20	Strážce Divé Máří	Malý jezdec
21	Divá Máří	Horní hlídač
22	Malá Máří	Kostková skála
23	Podzimní stěna	Hranatá skála
24	Kapucín	Kapucín
25	Hranatá skála	Podzimní stěna
26	Kostková skála	Stoletá stěna
27	Emilova věž	Zahradní věž
28	Lokomotivka	Zahradníkův učeň
29	Březová věžička	Zahradník
30	Šolcova stěna	Bivak
31	Šolcův jehlan	Přední stěna
32	Kazatelna	Štolpišská plotna
33	Novoroční věž	Sviní kámen
34	Mufloní plotna	Netopýří kámen
35	Dolní hlídač	Viklanová stěna
36	Malý jezdec	
37	Horní hlídač	
38	Střední hlídač	
39	Zahradníkův učeň	
40	Zahradní věž	
41	Zahradník	
42	Věž pod Ořešníkem	
43	Bivak	
44	Přední stěna	
45	Čihulova jehla	
46	Homole cukru	
47	Ostrý roh	
48	Hláska	

6.3 Funkčnost algoritmu

Celý algoritmus je rozdělen do třech hlavních částí. V první části dochází ke sběru a zpracování vstupních dat. Stačí do této sekce vložit vylosované vrcholy a algoritmus si z nich následně vybere ty, které potřebuje. Přebytečná data nechá mimo tabulku a dále s nimi nepracuje, aby ho to zbytečně nezpomalovalo. Právě vylosované vrcholy jsou jediná věc, která je zapotřebí algoritmu doložit, pokud budeme souhlasit s průměrnou rychlostí chůze a dobou potřebebnou na vylezení skalní věže.

V druhé části jsou definovány funkce, které se v průběhu algoritmu používají. Funkce jsme nakonec vložili do jedné funkce nazvané „hodnota cesty“. Tato funkce postupně upravuje cestu tak, aby pokud možno prošla vrcholy s co nejvíce body. Z funkce nakonec vychází seznam, na kterém jsou cesta, kterou algoritmus nakonec zvolil, její časová náročnost a také její bodové ohodnocení. Těchto hodnot využíváme dále v algoritmu k jeho správné funkci.

V poslední části dochází k navrhování a přijímání nové cesty. Důležitým faktorem je snižování teploty, kdy jak jsme si již popsali algoritmus snižuje teplotu a pracuje s čím dál, tím nižší teplotou.

V algoritmu jsme tedy navrhli žíhací schéma. Díky tomu můžeme dosahovat takovýhto výsledků v rozumném čase.

Zvolený počet kroků konkrétně 25000 je také zvolen na základě našich zkušeností s kódem. Několikrát jsem měnil cestu, ve které měl algoritmus za úkol nalézt nejkratší a sledoval jsem změny mezi jednotlivými počty kroků. Samozřejmě při vyšším počtu kroků je algoritmus přesnější, ale právě při námi zvolených 25000 je algoritmus funkční a relativně rychlý. Například jeho vykonání ve verzi programu R Studio 1.3.1093 na notebooku s procesorem Intel Core i7-6500U o frekvenci 2.50GHz a pamětí RAM 8 GB trvá provedení algoritmu 15 minut a 32 sekund. V tomto případě jsme hledali zlatý střed mezi dobou trvání a přesností výsledku. Jak jsem již zmínil, jednalo se o neustále pokusy, kdy jsem vysledoval, že při vyšším počtu kroků je výsledek téměř neměnný.

6.4 Zhodnocení algoritmu

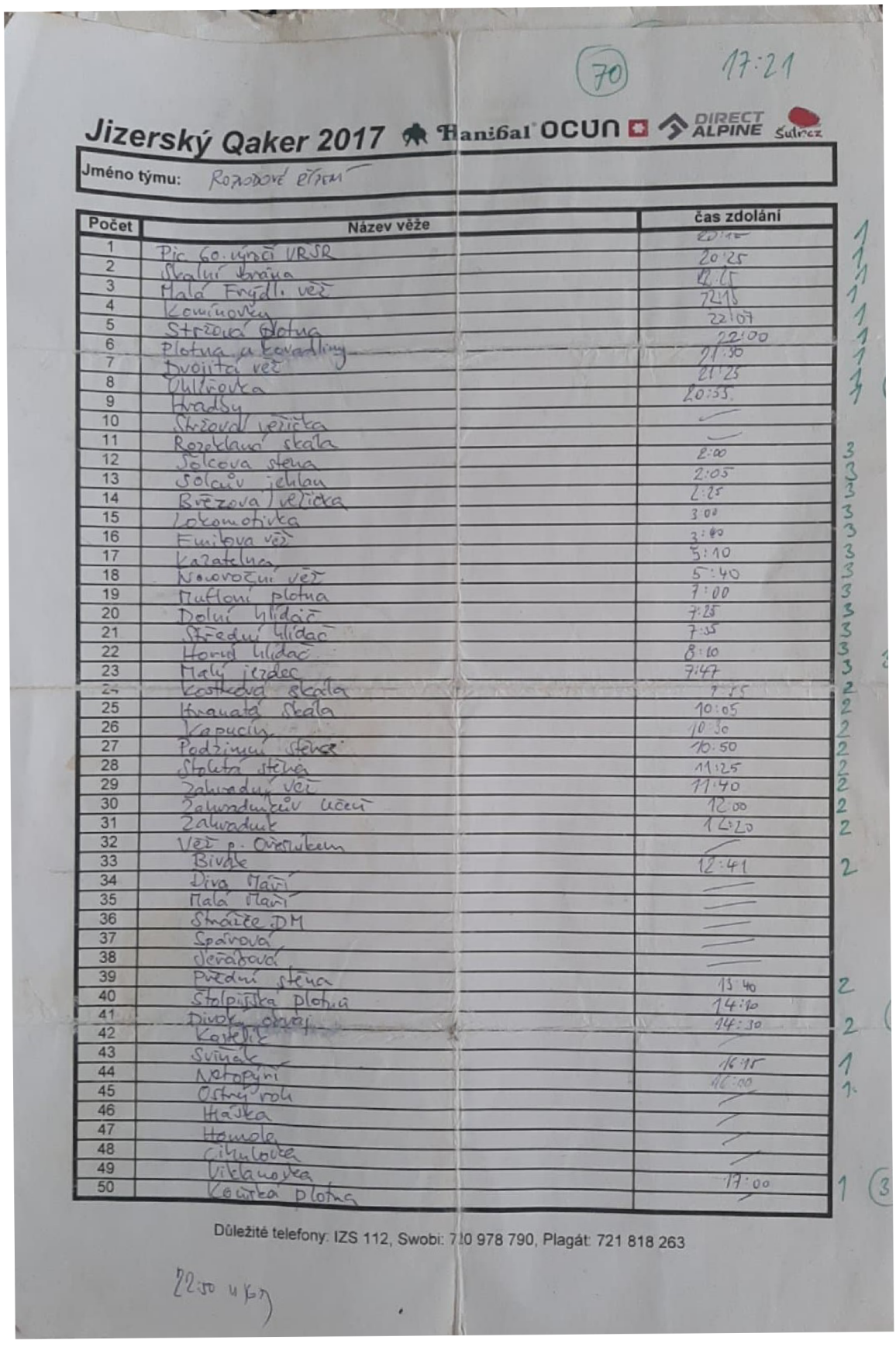
Algoritmus dokáže pracovat s různě zadanými vrcholy. Algoritmus bude předán pořadateli závodu, aby ho mohl vyzkoušet v reálných podmínkách, jak moc dobře funguje. Teoreticky funguje velmi dobře, avšak v reálné situaci je zde řada nedokonalostí.

Algoritmus byl prověřen řadou příkladů. Nejlepším ukazatelem správné funkce je především porovnání s vítězi jednoho z ročníku. V tomto příkladu lze pozorovat zkušenost vítězů a znalost Jizerských hor, kdy jsou schopni hledat spojení mezi věžmi tak, aby posbírali co nejvíce bodů, avšak také občas udělali nějakou tu chybu. Jinak v několika pasážích se jejich postup shoduje s tím, který nalezl algoritmus.

Jednou z chyb je počítání vzdáleností mezi vrcholy vzdušnou čarou, kterou násobíme koeficientem, který jsem dostal díky pokusům počítání jednotlivých vzdáleností a jejich zprůměrováním. Reálně tedy může nastat situace, kdy je jeden vrchol blíže

jinému, než je uvedeno v naší tabulce vzdáleností, avšak může být situace i naprosto obráceně. K této chybě se váže i další, kdy my bereme v potaz vzdušné spojení mezi jednotlivými skalními věžmi, avšak ve skutečnosti znalec Jizerských hor může nalézt lepší cestu díky znalosti jednotlivých cest mezi nimi a díky tomu mít třeba kratší cestu. Pro tyto případy by bylo zapotřebí vytvořit podmnožinu všech cest, což by bylo velmi náročné pro celkové množství 170 vrcholů.

Další chybou je udání průměrné rychlosti chůze $3,5\ \frac{km}{h}$, která se u jednotlivých závodníků může lišit. Tuto rychlost jsem bral z vlastních zkušeností, kdy jsem zkoušel chodit složitým terénem a dále jsem tempo trochu zmírnil vzhledem k fyzické náročnosti závodu. Ani doba potřebná ke zdolání jedné skály není nikterak pevně daná, ani stejná pro všechny vrcholky, avšak 10 minut je průměrná doba, kterou zabere právě vylezení na nějakou ze skalních věží. Zde by bylo zapotřebí ke každé skále přiřadit časovou náročnost jejího zdolání, která pokud by se zaznamenala do naší tabulky, nebylo by dále těžké s ní počítat dále.



70 17:21

Jizerský Qaker 2017 Hannibal OCUN DIRECT ALPINE Sulnex

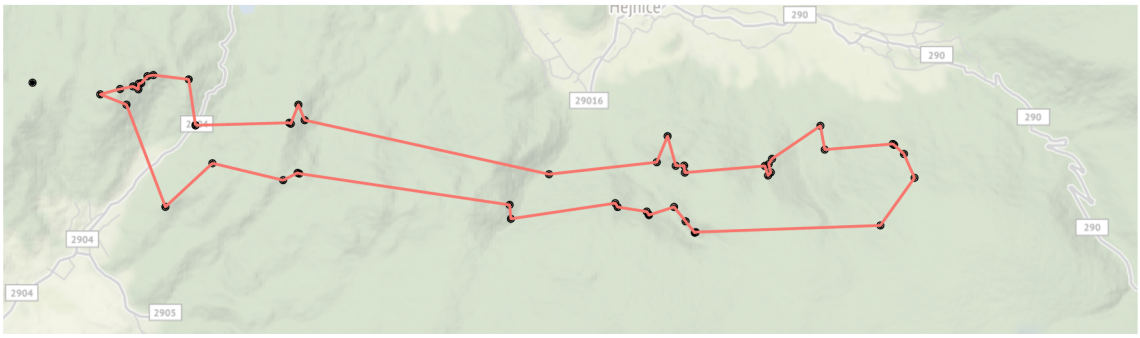
Jméno týmu: Rozbořené ořechy

Počet	Název věže	čas zdolání	
1	Pic 60. výročí VRJR	20:25	1
2	Skalní brána		1
3	Malá Frýdl. věž		1
4	Komínovka	22:07	1
5	Strážová plotna	22:00	1
6	Plotna u kovadliny	21:30	1
7	Dvojitá věž	21:25	1
8	Ullrichova	20:55	1
9	Hradby		
10	Strážová věžička		
11	Rozhledná skála		
12	Šolcova stěna	2:00	3
13	Šolcův jehlan	2:05	3
14	Březová jehlička	2:25	3
15	Lokomotiva	3:00	3
16	Emilova věž	3:00	3
17	Kazatelna	5:10	3
18	Novoroční věž	5:40	3
19	Mufloní plotna	7:00	3
20	Dolní hlídač	7:25	3
21	Střední hlídač	7:35	3
22	Horní hlídač	8:10	3
23	Malý jezdec	7:47	3
24	Kostková skála	9:55	2
25	Hraniatá skála	10:05	2
26	Kapucín	10:30	2
27	Podzimní stěna	10:50	2
28	Skalní stěna	11:25	2
29	Zahradní věž	11:40	2
30	Zahradníkův učeň	12:00	2
31	Zahradník	12:20	2
32	Věž p. Ullrichem		
33	Bivák	12:41	2
34	Dívčí Mauří		
35	Malá Mauří		
36	Smáčka DM		
37	Spárová		
38	Jeřábová		
39	Přední stěna	13:40	2
40	Stolpišťská plotna	14:10	
41	Divoký ořech	14:30	2
42	Kostelíč		
43	Svinák	16:15	1
44	Netopýří	16:00	1
45	Ostrý roh		
46	Hájka		
47	Homole		
48	Cihlová		
49	Viklanová		
50	Kočičí plotna	17:00	1

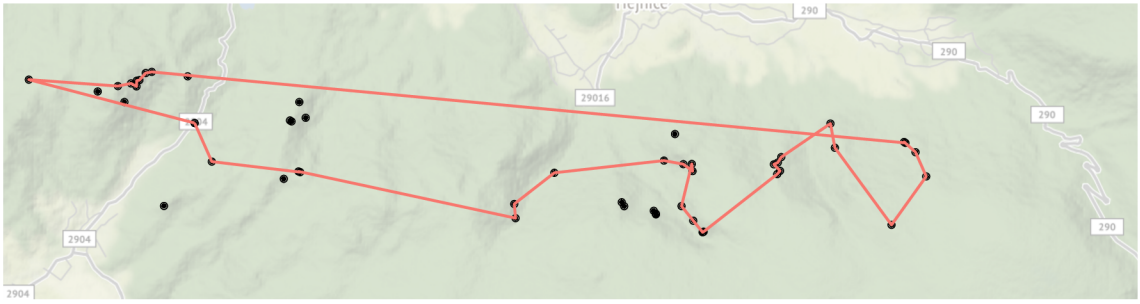
Důležité telefony: IZS 112, Swobi: 710 978 790, Plagát: 721 818 263

22:30 u kol

Obrázek 6.6: Záznamový list vítězné dvojice Jizerský Qaker 2017



Obrázek 6.7: Kružnice grafem vytvořená algoritmem pro ročník 2017



Obrázek 6.8: Kružnice grafem vítězné dvojice Jizerský Qaker 2017

Závěr

V práci jsme navrhli a implementovali algoritmus simulovaného žíhání pro závod Jizerský Qaker. Nalezli jsme tedy optimální trasu, kterou se závodník má vydat v tomto závodě. Algoritmus i přes veškerá zjednodušení, které jsme provedli funguje algoritmus relativně dobře v porovnání s realitou. Provedli jsme například přibližné měření vzdálenosti a času, ale i tak se výsledek algoritmu v příkladě z roku 2017 shoduje v jistých pasážích s trasou skutečných vítězů.

Algoritmus je vytvořen na základě matematického modelu, takže je přirozené, že trpí jistými nedostatky. Můžeme zde pozorovat jako například vynechání jedné věže v případě konkrétní úlohy, ale i přes to funguje relativně dobře. Algoritmus ale dokáže nalézt rozmuné suboptimální řešení této úlohy, což znamená, že z 50 vylosovaných skal se zadaným časovým limitem dokáže nalézt řešení. Časová náročnost cesty, kterou algoritmus nalezne jako řešení vyhovuje našemu omezenému času a také nasbírá přijatelný počet bodů.

Pro tvorbu této práce jsem se nejprve seznámil s teorií grafů, které jsem věnoval velkou pozornost při tvorbě této práce a její hlavní body potřebné pro řešení této úlohy jsem shrnul a vysvětlil v kapitole 2. Také jsem se seznámil a získal znalosti o výpočetní složitosti, konkrétně NP-těžkých úloh, kam spadá i problém obchodního cestujícího i náš problém.

Přínosy práce byly různé od matematiky, kdy jsem se dozvěděl a pochopil souvislosti spojené s teorií grafu a simulovaným žíháním, přes programátorské zkušenosti, kdy jsem algoritmus vytvářel v programovacím jazyce R až po seznámení a získání spousty zkušeností s programem LaTeX, který je vhodný pro tvorbu matematických vzorců.

Seznam obrázků

Seznam tabulek

Seznam přiložených souborů

1. vysledny_algoritmus.r
2. seznam_skal_zkraceny.csv

Literatura

[1] W. Cook, *History of the tsp*, 2007.

[2] G. A. Croes, *A method for solving traveling-salesman problems*, Online, (1958), pp. 791--812.

[3] G. Dantzig, R. Fulkerson, and S. Johnson, *Solution of a Large-Scale Traveling-Salesman Problem*, Journal of the Operations Research Society of America, 1954.

[4] J. Demel, *Grafy a jejich aplikace*, Nakladatelství Academia, 2002.

[5] V. Černý, *Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm*, Journal of Optimization Theory and Applications, 45 (1985), pp. 41--51.

[6] P. Fajgl, O. Simm, and M. Vrkoslav, *Jizerské hory: Horolezecký průvodce*, NH SAVANA, 2009.

[7] M. M. Flood, *The Traveling-Salesman Problem*, INFORMS, 1956.

[8] M. M. Flood, *The traveling-salesman problem*, Operations Research, 4 (1956), pp. 61--75.

[9] J. L. Gross and J. Yellen, *Handbook of Graph Theory*, CRC Press, 2003.

[10] M. Grötschel and O. Holland, *Solution of largescale symmetric travelling salesman problems*, Mathematical Programming, 1991.

[11] F. Harary, *Graph Theory*, Addison Wesley series in mathematics, Addison-Wesley, 1971.

[12] L. A. Hemaspaandra, *Sigact news complexity theory column 100.* online.

[13] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, *Optimization by simulated annealing*, SCIENCE, 220 (1983), pp. 671--680.

[14] M. ŠKOPEK, *Problém obchodního cestujícího a metoda genius [online]*, diplomová práce, Vysoká škola ekonomická v Praze, 2009 [cit. 2021-04-20].

[15] H. Michael and R. M. Karp, *The traveling-salesman problem and minimum spanning trees*, Mathematical Programming, 1971.

[16] P. Miliotis, *Using cutting planes to solve the symmetric Travelling Salesman problem*, Mathematical Programming, 1978.

[17] O. Ore, *Theory of graphs*, Providence : AMS, 1962.

[18] A. Schrijver, *On the history of combinatorial optimization (till 1960)*, in Discrete Optimization, K. Aardal, G. Nemhauser, and R. Weismantel, eds., vol. 12 of Handbooks in Operations Research and Management Science, Elsevier, 2005, pp. 1--68.

[19] Seznam, *Mapy.cz*. Online, 2021. Dostupné online.

[20] VIZUS, *Horolezectví, lezení, hory, skály, závody lezení, skialpinismus, horosvaz - Český horolezecký svaz - Čhs*. Online, 2021. Dostupné online.

A Kód v jazyce R

```
library(geosphere)
library(plyr)
library(ggplot2)
library(ggmap)
library(mapsapi)
library(leaflet)
library(ggthemes)
library(tidyverse)

## VSTUPNÍ PARAMETRY

#počet skal, které jsou vylosované
pocet_bodu = 50

#omezený čas v minutách
omezeny_cas = 1320

#rychlost chůze
rychlost_chuze = 3.5

#koeficient cesty po cestě oproti vzdušné vzdálenosti
vzdalenost_po_ceste = 1.25

#doba výstupu na jednu skálu
cas_vystupu = 10

#Vylosované skály
cesta_min = c(1,172,166,160,152,162,155,165,27,7,33,36,37,38,40,42,43,58,
49,176,177,175,174,156,119,120,121,96,125,123,124,129,132,114,115,116,
113,112,109,106,105,87,88,89,84,83,74,79,65,62)

#Tabulka skal
data0 = read.csv(paste0(getwd(),"/seznam_skal_zkraceny.csv"),sep = ";",
header = F)

data1=data0[,c(1:5)]
colnames(data1) = c("nazev","sirka","delka","sektor","index")
data1[,"sektor_char"] = as.character(data1[,"sektor"])
```

```
#Zjištění celkového počtu bodů
celkovy_pocet_bodu=length(data1[,1])

#vytvoření samostatného vektoru hodnoty vrcholů
hodnota = data1[,4]

#matice vzdálenosti
matice_vzdalenosti = matrix(0,nrow = celkovy_pocet_bodu,ncol =
        celkovy_pocet_bodu)

for (i in 1:celkovy_pocet_bodu) {
  for (j in 1:celkovy_pocet_bodu) {

    matice_vzdalenosti[i,j] = distm(c(data1[i,2],data1[i,3]),
c(data1[j,2],data1[j,3]), fun = distHaversine)

  }
}

# matice vzdáleností s časem
matice_vzdalenosti_cas =
(((matice_vzdalenosti*vzdalenost_po_ceste)/1000)/rychlost_chuze)*60

# Vytvoření bodů ze skal
body = matrix(c(data1[,2],data1[,3]), ncol = 2)

cesta=cesta_min
## POMOCNÉ FUNKCE

unit_vect = function(n,k){
  vec = rep(0,n)
  vec[k] = 1
  vec
}


hodnota_cesty = function(cesta, hodnota, matice_vzdalenosti_cas,
omezeny_cas, cas_vystupu){

  vzdalenosti_pom = c(sapply(1:(length(cesta)-1),
function(i){matice_vzdalenosti_cas[cesta[i],cesta[i+1]]}),
matice_vzdalenosti_cas[cesta[length(cesta)],cesta[1]])

  max_hodn = 0
  min_cas = 0
  max_cesta = c(1)
```

```
  konec = F
  i=2
  j=2

  while (!konec) {

    hodnota_navrh = sum(hodnota[cesta[i:j]])
    cas_navrh = if(i<j){
      matice_vzdalenosti_cas[cesta[1],cesta[i]]+
sum(vzdalenosti_pom[(i):(j-1)])+
matice_vzdalenosti_cas[cesta[j],cesta[1]]+(j-i+1)*cas_vystupu
    } else {
        2*matice_vzdalenosti_cas[cesta[1],cesta[i]]+cas_vystupu
      }

    if(cas_navrh <= omezeny_cas){

      if((hodnota_navrh > max_hodn)|(hodnota_navrh == max_hodn &
cas_navrh < min_cas)){
        max_hodn = hodnota_navrh
        min_cas = cas_navrh
        max_cesta = c(1,cesta[i:j])

      }
      if(j==length(cesta)){konec=T} else { j = j+1}

    } else {
      if(j==length(cesta)|j<i){konec=T} else { i = i+1}
    }
  }

  konec = F
  i=2
  j=3

  while (!konec) {

    hodnota_navrh = sum(hodnota[cesta[c(1,i,j:length(cesta))]])
    cas_navrh = sum(vzdalenosti_pom[c(1:(i-1),j:length(cesta))]) +
        matice_vzdalenosti_cas[cesta[i],cesta[j]]+
        (i+length(cesta)-j)*cas_vystupu


    if(cas_navrh <= omezeny_cas){

      if((hodnota_navrh > max_hodn)|(hodnota_navrh == max_hodn &
cas_navrh < min_cas)){
        max_hodn = hodnota_navrh
```

```
        min_cas = cas_navrh
        max_cesta = cesta[c(1:i,j:length(cesta))]

      }
      if(j==length(cesta)){konec=T} else { i = i+1}

    } else {
      if(j==length(cesta)|j<i){konec=T} else { j = j+1}
    }
  }

  list(hodnota = max_hodn,
       cas = min_cas,
       cesta = max_cesta
    )

}

maximalni_mozna_hodnota = sum(hodnota)

set.seed(123)

# Data jen na graf výsledné
data_plot = data1[cesta_min,]
data_plot = rbind(data1[cesta_min,],data1[cesta_min[1],])

# Přidání dat všech vylosovaných
data_plot = rbind(data_plot,data1[hodnota_cesty(cesta_min,hodnota,
        matice_vzdalenosti_cas, omezeny_cas,cas_vystupu)$cesta,],
        data1[cesta_min[1],])

#Jednička do cesta_char --> co se dají projít
data_plot[,"cesta_char"] = as.character(1)

#dvojka do cesta_char --> co jsou všechny skály
data_plot[(pocet_bodu+2):nrow(data_plot),"cesta_char"] = as.character(2)

data_plot = data_plot[data_plot$cesta_char=="2",]

#Vykreslení mapy
jizerky_map <- get_stamenmap(
  bbox = c(left = 15.0688, bottom = 50.8140, right = 15.2724, top = 50.8862),
  maptype = "terrain",
  zoom = 13
)

ggmap(jizerky_map, darken = c(0.6, "white") )+
  geom_point(data = data_plot,
```

```
            aes(x = delka, y= sirka),
            size = 2) +
  geom_path(data = data_plot,
            aes(x = delka, y = sirka, color = cesta_char),
            size = .9,
            show.legend = F,
            ) +
  theme_map()

plot(body[cesta_min[data0[cesta_min,4]<2],c(2,1)],
        xlab = "zemepisna delka", ylab = "zemepisna sirka",col="green"
     ,xlim = c(min(data0[cesta_min,3]),max(data0[cesta_min,3])),
        ylim = c(min(data0[cesta_min,2]),max(data0[cesta_min,2])))
points(body[cesta_min[data0[cesta_min,4]==2],c(2,1)],col="blue")
points(body[cesta_min[data0[cesta_min,4]==3],c(2,1)],col="red")

cas_min = hodnota_cesty(cesta_min, hodnota, matice_vzdalenosti_cas,
        omezeny_cas,cas_vystupu)$cas

hodnota_max = hodnota_cesty(cesta_min, hodnota, matice_vzdalenosti_cas,
        omezeny_cas,cas_vystupu)$hodnota

cesta = cesta_min
aktualni_hodnota = hodnota_max
cas = cas_min

#body do grafu žhacího schéma
postup = c()
pst_prijeti = c()

kroky = 25000

zihaci_schema= matrix(c(
  10, kroky,
  5, kroky,
  2, kroky,
  1.75, kroky,
  1.5, kroky,
  1.25, kroky,
  1, kroky,
  0.87, 2*kroky,
  0.75, 2*kroky,
  0.5, 2*kroky,
  0.38, 2*kroky,
  0.25, kroky,
  0.145, kroky,
  0.1, 2*kroky,
  0.05, 3*kroky,
```

```
  0.043, kroky,
  0.025, 2*kroky,
  0.01, kroky,
  0.005, kroky,
  0.001, kroky
  ),ncol=2, byrow = T)

for(n in 1:nrow(zihaci_schema)){

  teplota = zihaci_schema[n,1]
  pocet_kroku = zihaci_schema[n,2]

  message("Teplota je ", teplota)

  for(i in 1:pocet_kroku){

    navrhovana_cesta = cesta

    vrchol = sort(sample(1:pocet_bodu,2))

    rozdil=abs(vrchol[1]-vrchol[2])

     #prohození hran
    navrhovana_cesta[(vrchol[1]+1):vrchol[2]] =
        navrhovana_cesta[vrchol[2]:(vrchol[1]+1)]

    navrhovane_ohodnoceni_cesty = hodnota_cesty(navrhovana_cesta,
        hodnota, matice_vzdalenosti_cas, omezeny_cas,cas_vystupu)$hodnota

    #do grafu se žíhacm schéma
    postup = append(postup, aktualni_hodnota)
    pst_prijeti = append(pst_prijeti, 0)

    navrhovany_cas = hodnota_cesty(navrhovana_cesta, hodnota,
        matice_vzdalenosti_cas, omezeny_cas,cas_vystupu)$cas

    #prijmu novou cestu?
    if(runif(1) < exp((- aktualni_hodnota + navrhovane_ohodnoceni_cesty +
        (cas - navrhovany_cas)*0.015)/teplota)){

      pst_prijeti[length(pst_prijeti)] = 1

      aktualni_hodnota=navrhovane_ohodnoceni_cesty
      cesta=navrhovana_cesta
      cas = navrhovany_cas

    }
```

```
  if(hodnota_max < aktualni_hodnota ){
    hodnota_max = aktualni_hodnota
    cesta_min = cesta
    cas_min = cas

  }

  #Pokud stejná hodnota, upřednostni lepší čas

  if(hodnota_max == aktualni_hodnota){


    if(cas < cas_min){

      cas_min = cas
      cesta_min = cesta
      hodnota_max = aktualni_hodnota

    }

  }

}

print(hodnota_max)
print(cesta_min)
print(cas_min)

  plot(body[cesta_min[data0[cesta_min,4]<2],c(2,1)],
      xlab = "zeměpisná délka", ylab = "zeměpisná šířka",col="green"
       ,xlim = c(min(data0[cesta_min,3]),max(data0[cesta_min,3])),
      ylim = c(min(data0[cesta_min,2]),max(data0[cesta_min,2])))
  points(body[cesta_min[data0[cesta_min,4]==2],c(2,1)],col="blue")
  points(body[cesta_min[data0[cesta_min,4]==3],c(2,1)],col="red")

  cesta_min_zkracena = hodnota_cesty(cesta_min,hodnota,
      matice_vzdalenosti_cas, omezeny_cas,cas_vystupu)$cesta

  for (i in 1:(length(cesta_min_zkracena)-1)) {
    lines(c(body[cesta_min_zkracena[i],2],
      body[cesta_min_zkracena[i+1],2]),c(body[cesta_min_zkracena[i],1],
      body[cesta_min_zkracena[i+1],1]),col="blue")
  }
  lines(c(body[cesta_min_zkracena[length(cesta_min_zkracena)],2],
      body[cesta_min_zkracena[1],2]),
      c(body[cesta_min_zkracena[length(cesta_min_zkracena)],1],
      body[cesta_min_zkracena[1],1]),col="blue")
```

```
    data_plot_2 = data1[cesta_min,]
    data_plot_2 = rbind(data1[cesta_min,],data1[cesta_min[1],])
    data_plot_2[,"cesta_char"] = as.character(1)

    data_plot = data1[cesta_min_zkracena,]
    data_plot = rbind(data1[cesta_min_zkracena,],
        data1[cesta_min_zkracena[1],])
    data_plot[,"cesta_char"] = as.character(1)

    jizerky_map <- get_stamenmap(
      bbox = c(left = 15.0688, bottom = 50.8140, right = 15.2724, top = 50.8862),
      maptype = "terrain",
      zoom = 13
    )

    ggmap(jizerky_map, darken = c(0.6, "white") )+
      geom_point(data = data_plot_2,
                 aes(x = delka, y= sirka),
                 size = 2) +
      geom_path(data = data_plot,
                aes(x = delka, y = sirka, color = cesta_char),
                size = .9,
                show.legend = F,
      ) +
      theme_map()

  }

omezeni_vect = sapply(1:1320, function(ocas){hodnota_cesty(cesta_min,
        hodnota,matice_vzdalenosti_cas, ocas, cas_vystupu)$cas})

plot(omezeni_vect)

plot(postup,  xlab = "Počet kroků", ylab = "Hodnota cesty",
        pch = 20, type = "l")

ma = sapply(0:(floor(length(pst_prijeti)/1000)-1),function(i)
        {mean(pst_prijeti[ (i*1000+1):min(length(pst_prijeti),
        (i*1000+1000))])})
plot(ma)
```