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

GLOBÁLNÍ PLÁNOVÁNÍ CEST V NEURČITÝCH PROSTŘEDÍCH S VYUŽITÍM METOD UMĚLÉ INTELIGENCE

GLOBAL PATH PLANNING IN NONDETERMINISTIC ENVIRONMENT USING ARTIFICIAL INTELLIGENCE

DISERTAČNÍ PRÁCE

DISSERTATION THESIS

AUTOR PRÁCE — Ing. Petr Šoustek

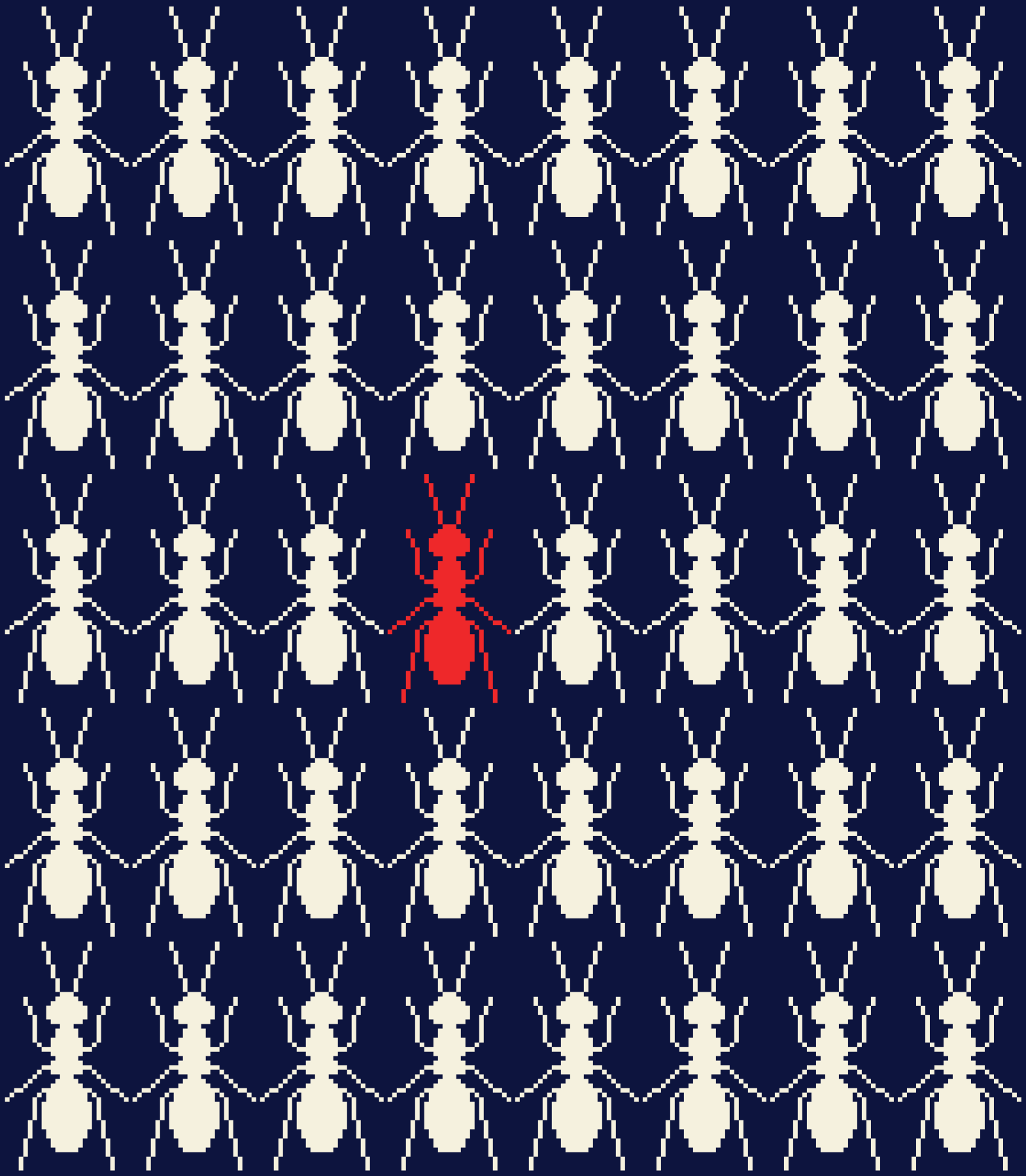
AUTHOR

ŠKOLITEL — prof. Ing. Radomil Matoušek, Ph.D.

SUPERVISOR

BRNO 2024

GLOBÁLNÍ PLÁNOVÁNÍ CEST V NEURČITÝCH PROSTŘEDÍCH S VYUŽITÍM METOD UMĚLÉ INTELIGENCE



ABSTRAKT

Tato disertační práce se věnuje problematice globálního plánování cest v neurčitých prostředích s využitím metod umělé inteligence. Vlastnímu řešení implementovaných algoritmů předchází seznámení se základní problematikou teorie grafů, plánování cest a jejich metodám. Část práce se věnuje popisu stěžejního problému kanadského cestujícího, který vychází z problému obchodního cestujícího. Práce se rovněž detailně zabývá rešerší mravenčích metod, a to od základních, až po pokročilé mravenčí metody. Mravenčí metody jsou zcela novým přístupem k řešení problému kanadského cestujícího. Vytvořený software napsaný v jazyce C++ je určen pro řešení tohoto problému, ale i dalších problémů, jako je problém obchodního cestujícího a problém kanadského obchodního cestujícího. Software umožňuje použití navržených mravenčích metaheuristik a heuristických metod. Experimenty jsou podloženy výsledky na syntetických i reálných problémech kanadského cestujícího. Využití mravenčích metod se jeví perspektivně, což je podpořeno prezentovanými výsledky v závěru práce.

ABSTRACT

This dissertation deals with the issue of global travel planning in uncertain environments using artificial intelligence methods. Getting to know the basic issues of graph theory, path planning and their methods precedes the actual solution of the implemented algorithms. Part of the work is devoted to the description of the core problem of the Canadian Traveller, which is based on the Travelling Salesman problem. The work also deals in detail with the research of ant methods, from basic to advanced ant methods. Ant methods are a completely new approach to solving the Canadian Traveller problem. The created software is written in C++ language and is intended for solving the Canadian Traveller problem, but also other problems such as the Travelling Salesman problem and the Canadian Traveling Salesman problem. It allows the use of ant metaheuristics and other heuristic methods. Experiments are supported by results on synthetic and real cases of Canadian Traveller problems. The use of ant methods appears promising, which is supported by the results presented in the end of the dissertation.

KLÍČOVÁ SLOVA

Problém kanadského cestujícího, problém obchodního cestujícího, optimalizace, heuristika, metaheuristika, mravenčí algoritmy

KEYWORDS

Canadian Traveler Problem, Traveling Salesman Problem, optimization, heuristic, metaheuristic, ant algorithms

BIBLIOGRAFICKÁ CITACE

ŠOUSTEK, Petr. Globální plánování cest v neurčitých prostředích s využitím metod umělé inteligence [online]. Brno, 2024 [cit. 2024-02-28]. Dostupné z: https://www.vut.cz/studenti/zav-prace/detail/154868. Disertační práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky. Školitel práce prof. Ing. Radomil Matoušek, Ph.D.

PODĚKOVÁNÍ

Děkuji svému školiteli Radomilu Matouškovi za vedení při studiu, za spolupráci při výzkumu a za všechen čas mně věnovaný. Za cenné připomínky děkuji též Jiřímu Dvořákovi a Sabině Frýzové. Dále bych chtěl poděkovat za podporu a spolupráci mým kolegům. V neposlední řadě chci vyjádřit svůj vděk a ze srdce poděkovat své rodině a nejbližším za podporu nejen při studiu.

„Thanks for the company and the fish, you know who you are.“

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že tato práce je mým původním dílem, vypracoval jsem ji samostatně pod vedením školitele 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 následku porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestně právních důsledků.

V Brně dne 29. 2. 2024 ……

Petr Šoustek

„There are two ways of getting home; and one of them is to stay there.“

G.K. Chesterton, The Everlasting Man

OBSAH

1 ÚVOD

Plánování cesty v neurčitém prostředí představuje významnou výzvu v oblasti autonomní mobility, logistiky, či robotiky, přičemž doménově spadá do oblastí umělé inteligence a operačního výzkumu. Tento problém zahrnuje identifikaci optimální nebo přijatelné trasy pro agenta z počátečního bodu do cílového bodu v prostředí, které může obsahovat neznámé nebo dynamicky se měnící překážky. Neurčitost může vznikat z nedokonalých nebo neúplných informací o prostředí, z nepředvídatelných změn v prostředí nebo z omezení senzorů a aktuátorů agenta.

Předložená práce se věnuje problematice plánování cest v neurčitém prostředí pomocí vybraných metod *umělé inteligence* (*Artificial Inteligence*), jejichž užití v ohledu mravenčích algoritmů je zcela inovativní. Při plánování cest se nejčastěji setkáváme s použitím metod, které na základě heuristického nebo metaheuristického přístupu hledají optimální, či dostatečně dobrá řešení.

V celé práci se v ohledu řešených úloh bude vždy jednat o grafovou interpretaci problému. Tato strategie odpovídá globálnímu plánování trasy tak, jak je například běžně v navigačních prostředcích u automobilů (*gps navigace).* Vzhledem k cílovému měřítku předkládaných úloh je uvedená interpretace tedy zcela opodstatněná a v reálných aplikacích běžná. Pojem neurčité prostředí tedy v grafové reprezentaci představuje třídu úloh, kde nemusí být předem známo ohodnocení hran grafu nebo kde bude toto ohodnocení časově proměnné. Zásadní řešená úloha této práce, označená jako *problém kanadských cestujícího* (*Canadian Traveller Problem*, CTP), je zajímavý a významný problém v teorii grafů a algoritmů, který rozšiřuje klasický problém nejkratší cesty tím, že zavádí prvek neurčitosti do definovaného prostředí. Tento problém zahrnuje mnoho variant, které byly v rámci řešení práce analyzovány a dále klasifikovány pro potřebu řešení, resp. návrhu a implementace řešících algoritmů. Motivací a výzvou úlohy CTP je způsob, jak efektivně plánovat cestu v prostředí, kde jsou některé informace o dostupnosti cest známy pouze lokálně a v reálném čase. To vyžaduje nové strategie, které mohou rychle reagovat na nově objevené blokace či dočasnéí uzavírky a najít alternativní trasy, aniž by tyto výrazně zvyšovaly celkovou dobu cesty nebo vzdálenost. Uveďme, že CTP problém patří do kategorie NP-těžkých problémů. To znamená, že neexistuje známý polynomiální časový algoritmus, který by jej řešil pro všeobecný případ.

Jak prokázala předložená disertační práce, mezi množinu metod, které je možné úspěšně využít pro řešení problému CTP patří, krom diskutovaných a nově modifikovaných i implementovaných heuristik, metody umělé inteligence, resp. metaheuristiky založené na mravenčích koloniích (*Ant Colony Optimization, ACO*). Jde o metody řazené do kategorie tzv. hejnové či rojové inteligence (*Swarm Intelligence*), které jsou inspirovány chováním samo-organizujících se organismů, jako jsou například včely, mravenci nebo ptáci. Rozebírané mravenčí metody (*Ant Colony Optimization*) spolu s vlastním CTP problémem tvoří základní kámen řešení disertační práce.

1.1 Cíle práce

Řešení disertační práce představovalo netriviální sled činností zahrnujících důkladnou rešerši studovaného problému, který patří v oblasti kombinatorické optimalizace k méně diskutovaným, analýzu stávajících heuristik a jejich rozšíření, zcela nově pojatou analýzu mravenčích algoritmů ve smyslu jejich implementace, vytvoření testovacích korpusů, přizpůsobení možných testovacích korpusů třídy TSP, adaptaci plnohodnotných reálných map pro potřebu

testování, vlastní vyhodnocení a pochopitelně rozsáhlou programátorskou činnost v ohledu implementace řešících algoritmů a podpůrných metod.

Tyto cíle byly dekomponovány následovně:

- Rešerše problematiky CTP úloh a všech jejich variant.
- Rešerše existujících heuristik, jejich analýza, případné rozšíření a implementace.
- Rešerše metaheuristik na bázi mravenčích algoritmů vhodných pro řešení CTP.
- Návrh a implementace heuristik (problémově závislá algoritmizace).
- Návrh a implementace metaheuristik třídy ACO (problémově závislá algoritmizace).
- Objektový návrh a implementace vlastního řešiče a podpůrných metod CTP.
- Vlastní experimenty realizované na umělých i reálných problémech CTP.
- Závěry a zhodnocení.

1.2 Popis kapitol

V úvodní 2. kapitole jsou rozbírány vybrané pojmy z oblasti navigace robotů, či obecněji agentů a plánování cest. Jsou popisovány typy diskretizace prostoru a je uveden přehled použití metod umělé inteligence pro plánování cesty využívajících grafových struktur.

Následující kapitoly jsou věnovány problému obchodního cestujícího, jako základní typové úloze – 3. kapitola a intenzivně *problému kanadského cestujícího* (*Canadian Traveller Problem*, CTP) – 4. kapitola, který modeluje reálnou úlohu v neurčitém prostředí. Pro tuto stěžejní úlohu je provedena rešerše jednotlivých variant problému a existujících způsobů – heuristik pro nalezení řešení. Na tyto stati navazuje v 5. kapitole podrobná rešerše mravenčích metod, a to od jejich biologické inspirace až po jednotlivé základní i pokročilé mravenčí metody.

V prakticky orientované 6. kapitole je popsáno vlastní řešení problému kanadského cestujícího, včetně algoritmizace implementovaných metod. Nejprve jsou uvedeny heuristické metody pro řešení CTP a dále modifikace mravenčích algoritmů určené pro tento problém. Přičemž zdůrazněme, že právě využití mravenčích metod je zásadní a vzhledem k problému CTP zcela novým a perspektivním přístupem.

V 7. kapitole je popsána komplexní struktura navrženého software, který vznikl rovněž jako zásadní část řešení této práce. Stručně je diskutována architektura, vývojová platforma C++, vlastnosti, GUI i rozhraní příkazové řádky a souhrnně vlastní princip práce s navrženým software vč. stručné dokumentace.

Důležitou částí předložené práce uzavírá 8. kapitola, kde jsou uvedeny výsledky experimentů při řešení umělých i praktických (reálných) úloh CTP. Jsou zde prezentována a komparována řešení dosažená s využitím vytvořených řešičů na definovaných testovacích korpusech, z nichž některé jsou získány z reálných mapových podkladů.

Na závěr, tedy v 9. kapitole, je uvedeno shrnutí a věcná diskuze k výsledkům, které byly dosaženy v rámci řešení předložené disertační práce. Jsou uvedeny návrhy pro možná rozšíření a využití dosažených výsledků, které naplnily netriviální cíle této disertační práce a bezesporu obohatily jejího autora.

2 VYBRANÉ POJMY A ALGORITMY

2.1 Navigace

Navigací mobilních robotů označujeme postupy, jak stanovit jejich polohu v prostoru a nalézt nejvhodnější cestu z počátečního bodu do cílového bodu. Navigaci můžeme postupně rozdělit do tří částí [1]:

- *Učení mapy*,

  jedná se o proces, při kterém se zpracovávají data získaná ze senzorů robotu.

- *Lokalizace*,

  určuje aktuální pozici robotu na mapě pracovního prostoru.

- *Plánování cesty*,

  provádí výběr cesty, po které se robot dostane z počáteční pozice do cílové pozice tak, aby nedošlo k případné kolizi s překážkou.

Učení mapy a lokalizace jsou navzájem spojené procesy, v literatuře označované jako problém simultánní lokalizace a mapování (SLAM) [2, 3], jelikož pro lokalizaci robotu na mapě je nutno znát mapu prostoru a pro vytvoření mapy je zase nutno znát pozici tohoto robotu v prostoru. Proces plánování cesty je závislý na výsledcích předcházejících dvou procesů. K samotnému plánování je nutno znát jak mapu prostředí, ve kterém se robot nachází, tak i jeho pozici na této mapě. Nejčastěji přístupy k plánování cesty můžeme rozdělit do dvou kategorií:

- *globální plánování cesty*,
- *lokální plánování cesty*.

Globální plánování cesty se snaží nalézt takovou posloupnost kroků (stavů) robotu (obecně mobilního prostředku), která jej dovede z počátečního bodu do cílového bodu tak, aby tato nalezená cesta nekolidovala s překážkami, a to ještě před započetím pohybu robotu. Poznamenejme, že globální plánování se majoritně interpretuje na ohodnoceném grafu. Mapa je nejprve převedena do grafové podoby a pohyb v tomto grafu je úlohou plánování cest v grafu. V případě lokálního plánování robot reaguje na překážky zjištěné senzory a upravuje svůj pohyb tak, aby nedošlo k případné kolizi. V případě reálného robotu probíhá jeho pohyb ve spojitém prostředí. V takovém spojitém prostředí je pak hranice mezi dvěma stavy omezena jen lidským vnímáním a nikoli prostředím jako takovým. Pak lze definovat toto spojité prostředí jako nekonečnou množinu všech stavů, a to i pro prostor s konečnou velikostí. Mapu, která obsahuje souřadnice objektů, jež se vztahují k danému systému, nazýváme metrickou mapou [1]. Plánování cesty nad touto mapou není zcela časově efektivní, jelikož množina všech stavů robotu je nespočetná. Pro případné plánování cesty ve spojitém prostoru lze však použít metodu potenciálových polí, která je stručně popsána dále v textu. Abychom mohli použít další metody plánováni cesty, je nutno z metrické mapy vytvořit topologickou mapu pomocí diskretizace spojitého prostoru. Diskretizací spojitého prostředí o konečné velikosti získáme diskrétní prostředí, které je pak definováno jako konečná množina stavů a přechodů mezi těmito stavy. Dále při plánování cesty robotu je nutno zohlednit, jaká omezení jsou kladena na jeho pohyb. Dle těchto omezení rozlišujeme dva základní druhy, a to holonomní a neholonomní. Holonomní robot má stejný počet řiditelných stupňů volnosti, jako je jeho celkový počet stupňů volnosti. Neholonomní robot má počet řiditelných stupňů volnosti menší, než je jejich celkový počet. Vzhledem k tomu, že je jako modelový aplikační hardware používán holonomní robot, jsou dále v textu popisovány algoritmy vhodné pro tento typ robotu. Dodejme, že volba měřítka dané úlohy může

i neholonomní mobilní prostředek dobře aproximovat na holonomní. Příkladem je globální plánování pohybu automobilu z města A do města B.

2.2 Pracovní a konfigurační prostor

Podle autora knihy [4] je prostor, ve kterém robot plní úkoly, nazýván pracovním prostorem W, který lze reprezentovat jako n-rozměrný euklidovský prostor E^n. V tomto prostoru se nachází překážky O $(obstacles)$ a robot R. V případě překážek, které omezují pohyb robotu, se může jednat jak o statické překážky, které nemění svou polohu, tak o dynamické překážky, které mění svoji polohu v průběhu času. Robot tak na tyto dynamické překážky reaguje až v případě setkání. Pracovní prostor W je reprezentován n-rozměrným euklidovským prostorem E^n, ve kterém se robot nachází v nějaké konfiguraci q. Tato konfigurace je pro planární pohyb dána vektorem $q = (x, y, \alpha)$, jehož složky obsahují souřadnice polohy a orientaci robotu. Všechny konfigurace robotu v daném pracovním prostoru W tvoří konfigurační prostor C [4]. Ve vztahu k reálnému prostředí můžeme následně konfigurační prostor C rozdělit na prostor překážek C_{obs} a volný konfigurační prostor C_{free}. Pro prostor překážek pak platí:

$$C_{obs} \subseteq C \tag{2.1}$$

Jako překážku můžeme označit jakýkoliv objekt, který omezuje pohyb robotu, nebo konfiguraci robotu, která by neměla nastat. Pro volný konfigurační prostor platí:

$$C_{free} = C / C_{obs} \tag{2.2}$$

Pro pracovní prostor $W = E^2$ obsahující překážky O, $O \subset W$ a robot R, $R \subset W$, jenž má konfiguraci: $q = (x, y, \alpha)$ a $q \in C$, lze pak vyjádřit prostor překážek jako:

$$C_{obs} = \{q \in C | R(q) \cap O \neq \emptyset\} \tag{2.3}$$

2.3 Plánování cesty

Samotnou úlohu plánování cesty můžeme nejčastěji rozdělit na dvě části, přičemž v té první dochází ke zpracování mapy. Zde se provádí diskretizace pracovního prostoru a jeho následné převedení do konfiguračního prostoru. Poté je volný konfigurační prostor reprezentován grafovou strukturou. Druhou částí plánování cesty je pak použití některé z klasických metod nebo metod umělé inteligence pro nalezení vhodné cesty v této grafové struktuře.

2.3.1 Metody zpracování pracovního prostoru

Zpracování pracovního prostoru můžeme provést mnoha způsoby. Jelikož se jedná o velmi studovanou oblast, uvádíme zde stručně pouze výběr základních metod. Podrobně se tomuto tématu věnují například autoři v následující literatuře [1, 3, 4].

Metody rozkladu do buněk

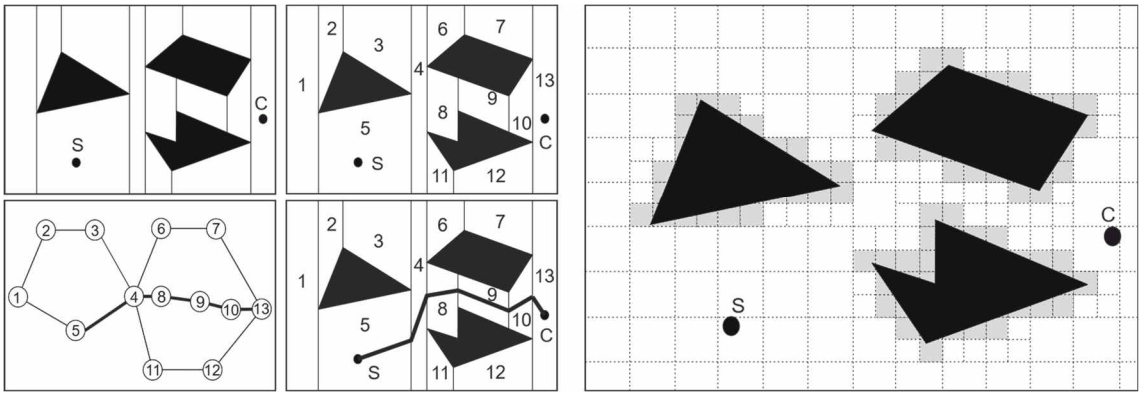
V případě metod rozkladu do buněk je pracovní prostředí rozloženo na jednotlivé buňky. Dochází tak k postupné dekompozici spojitého prostoru. Jednotlivé buňky mohou mít různou velikost a tvar. Podle toho, jakým způsobem rozložíme pracovní prostor, můžeme rozklad do buněk rozdělit na tzv.

- *exaktní rozklad* a
- *aproximativní rozklad*.

Poté, co rozložíme pracovní prostor na jednotlivé buňky, jsou označeny právě ty buňky, které obsahují překážku. Z takto zpracovaného prostoru je vytvořena grafová struktura, ve které jednotlivé buňky tvoří vrcholy grafu a hrany tvoří spojnice mezi těmito vrcholy.

Exaktní rozklad

Metody exaktního rozkladu rozdělují volný konfigurační prostor C_{free} do navzájem se nepřekrývajících buněk. Tvar těchto buněk je nejčastěji volen jako trojúhelníkový nebo lichoběžníkový, a to z důvodu snížení výpočetní náročnosti. Jedna buňka obsahuje startovní pozici a jedna buňka cílovou pozici. Jednotlivé buňky sdílí společné hrany. Vrcholy grafové struktury jsou tvořeny jednotlivými buňkami a hrany mezi vrcholy tvoří přechody mezi těmito buňkami. V této grafové struktuře je pak vyhledána cesta mezi startovním a cílovým bodem. Pokud cesta mezi těmito body existuje, je vždy nalezena.



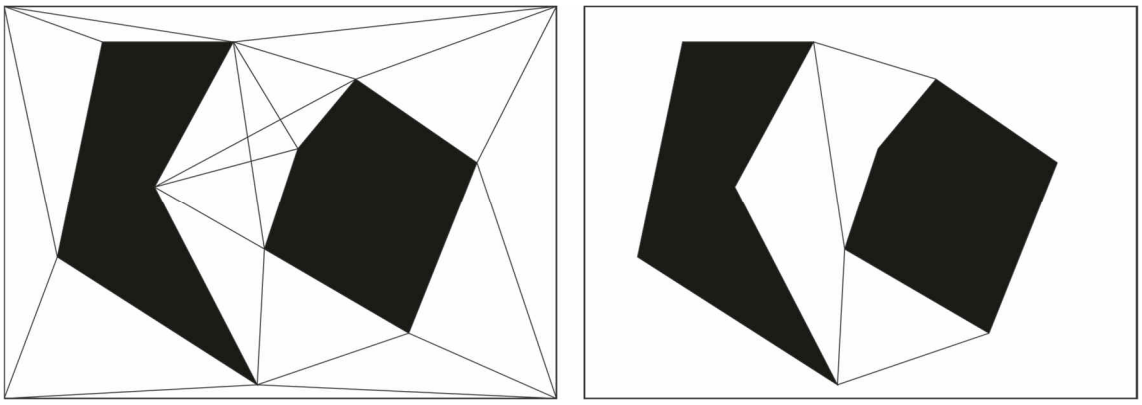
Obr. 1. Ukázka metod exaktního (vlevo) a aproximativního (vpravo) rozkladu

Aproximativní rozklad

Tento algoritmus rozkládá pracovní prostor na buňky, a to buď o stejné, nebo rozdílné velikosti. Každá buňka je označena, zda je prázdná, nebo obsazená překážkou. V případě, že rozkládáme pracovní prostor na buňky o stejné velikosti, je buňka, jejíž část obsahuje překážku, označena jako obsazená. V případě, kdy používáme různé velikosti buněk, lze takovou buňku dále dělit, dokud nedosáhneme dostatečného rozlišení pro vyhledání cesty.

2.3.2 Mapy cest

Cestovní mapy zachycují volný konfigurační prostor C_{free} do grafové struktury takovým způsobem, že propojují významné body v tomto prostoru pomocí spojnic. Tyto body tvoří jednotlivé vrcholy překážek. K vytvoření propojení dochází mezi vrcholem jedné překážky a vrcholem druhé překážky, aniž by došlo ke kolizi s některou z dalších překážek. Množina všech těchto propojení spolu se startovním a cílovým bodem tvoří cestovní mapu. Tato cestovní mapa je zároveň grafová struktura, ve které následně hledáme vhodnou cestu. U méně složitých cestovních map tak můžeme použít například *Dijkstrův algoritmus*. V případě těchto metod je vytvořená mapa vhodná pro bodovou reprezentaci robotu. Pro mapu použitelnou reálným robotem je proto nutné provést tzv. rozšíření překážek ještě před započetím tvorby spojnic v prostoru. Do kategorie cestovních map patří plánovací metody, jako je *graf viditelnosti*, *Voroného diagramy*, *metoda dálniční sítě* [5] nebo *metoda siluet* [6].



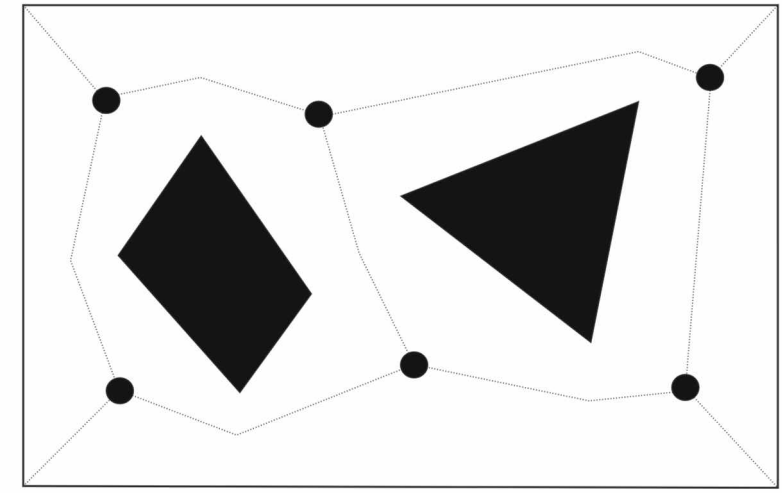
Obr. 2. Graf viditelnosti a graf tečen

Graf viditelnosti

Jednotlivé vrcholy grafu tvoří vrcholy překážek, startovní a cílová pozice. Hrany grafu jsou vytvořeny ze spojnic mezi těmito vrcholy. Vrcholy, které propojují tyto spojnice, jsou navzájem viditelné, tudíž spojnice neprotínají žádnou překážku. Tento graf lze vytvořit jenom v prostředí, které obsahuje pouze polygonální překážky. Pokud obsahuje i nepolygonální překážku, je nutno z takovéto překážky vytvořit polygon. V komplexních prostředích může dojít k vytvoření velkého množství hran a zvětšení časové náročnosti prohledávání grafu. Časovou náročnost lze snížit konstrukcí redukovaného grafu viditelnosti neboli grafu tečen, kdy hrany tvoří tečny mezi navzájem viditelnými vrcholy. Pro graf tečen může prostředí obsahovat i nepolygonální překážky.

Voroného diagram

Voroného diagram představuje jednu z nejdůležitějších struktur v počítačové geometrii. Byl pojmenován po ruském matematikovi Georgiji Fedosjeviči Voroném (1868-1908). Pro danou množinu navzájem disjunktních polygonů Voroného diagram rozděluje rovinu do buněk tak, že body v každé Voroného buňce jsou blíže ke konkrétnímu polygonu než ke všem ostatním polygonům. Pro každý bod roviny uvažujeme nejbližší vrchol nebo hranu polygonu. Voroného vrcholy jsou v tomto případě body stejně vzdálené od nejbližších vrcholů či hraná tří (či více) různých polygonů. Vrcholy jsou spojeny souvislými řetězci Voroného oblouků. Jestliže je oblouk stejně vzdálený od dvou nejbližších vrcholů nebo od dvou nejbližších hran polygonu, jedná se o úsečkový segment. Když je oblouk stejně vzdálený od vrcholu polygonu a neincidentní hrany polygonu, jedná se o parabolický oblouk. Pro jakýkoli bod roviny je hodnota odstupu vzdálenost bodu od nejbližšího polygonu. Voroného diagram lze použít k nalezení cest s maximálním odstupem od překážek. Podrobněji se tomuto tématu věnuje například práce [7].



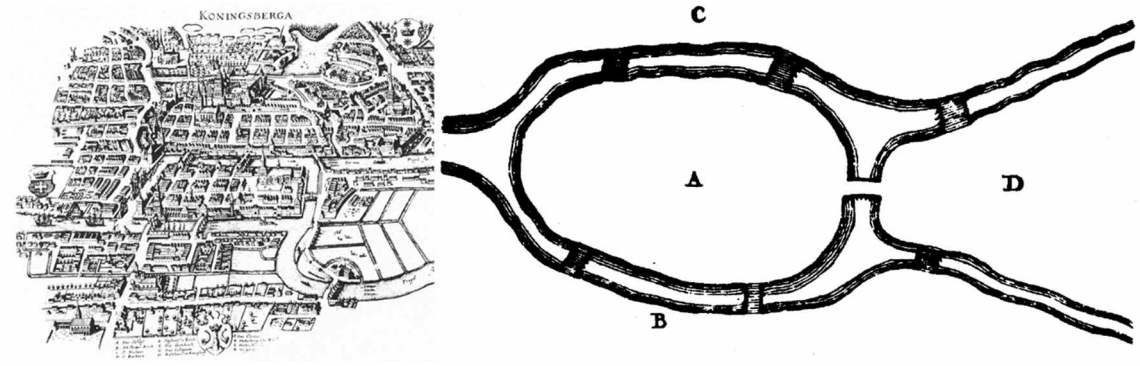
Obr. 3. Voroného diagram

2.4 Metody plánování cesty

Druhou fází plánování cesty je nalezení cesty ve vytvořeném grafu. Pro nalezení cesty můžeme použít některou z klasických metod nebo metod umělé inteligence. V této podkapitole jsou používány obecné poznatky a definice z teorie grafů, se kterými se pracuje v této práci. Jelikož je téma teorie grafů velmi obsáhlé, lze čtenáři doporučit pro další informace knihy, jako je například *Graph theory* od Richarda Dienstela [8] nebo *Graphs, Networks and Algorithms* od Dietera Jungnickela [9] a další, které jsou zmiňovány na následujících řádcích.

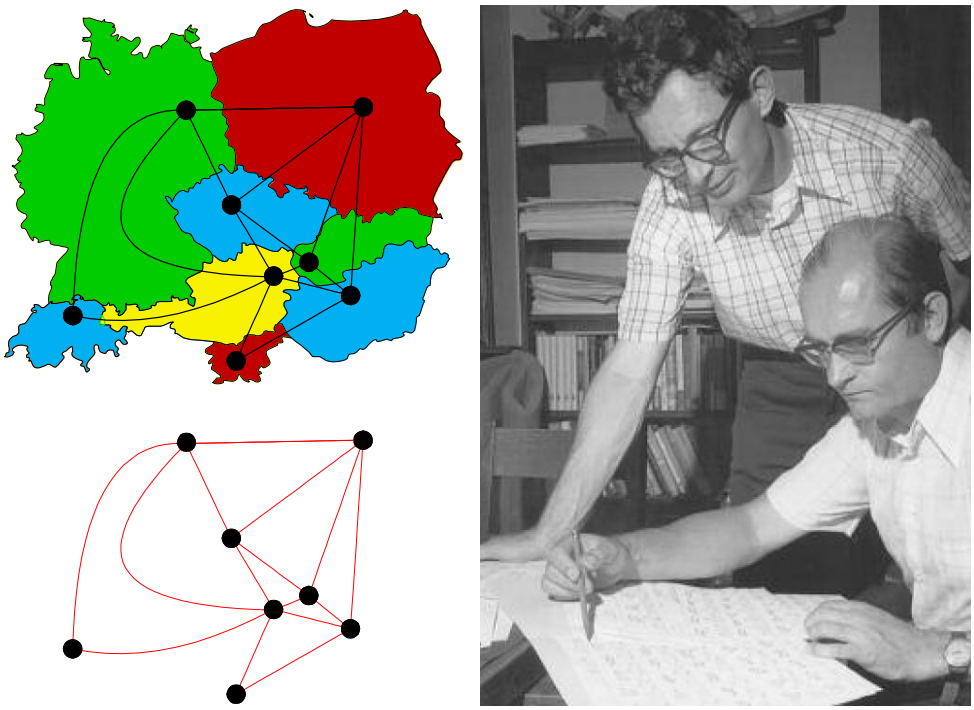
2.4.1 Historie teorie grafů

Základní kámen teorie grafů položil v roce 1736 Leonard Euler článkem *Solutio problematis ad geometriam situs pertinentis* [10] kde řešil problém sedmi mostů v Královci, dnešním Kaliningradu. Euler byl dotázán, zda je možné přejít všech sedm mostů přes řeku Pregel pod podmínkou, že každý z mostů lze přejít pouze jedenkrát (dnes tomuto způsobu průchodu říkáme Eulerovský tah). Euler ve svém článku uvádí částečné podmínky pro existenci takovéhoto tahu a dokazuje, že úloha v podobě, jaká je v Královci, je neřešitelná [11]. Kompletní důkaz, zda a kdy existuje taková cesta, která splňuje všechny podmínky, uvedl v roce 1871 Carl Hierholzer svému kolegovi, načež brzy zemřel. Článek [12] ve kterém byl tento důkaz publikován, tak vyšel až posmrtně v roce 1873. O téměř sto let později, co vyšel Hierholzerův článek, publikoval v roce 1960 Mei-Ko Kwan článek nejprve v čínštině a o dva roky později i v angličtině s názvem „*Programming method using odd or even points*“ [13, 14]. V něm navazuje na Eulerův problém sedmi mostů v Královci a formuluje problém čínského listonoše, kterým ostatně i jeden čas byl. Cílem problému čínského listonoše je nalezení uzavřeného Eulerovského tahu (listonoš se musí vrátit zpět na poštu) s co nejmenší délkou cesty. O minimalizaci délky této cesty do té doby nikdo neuvažoval, cílem pro Eulera i Hierholzera byla pouze její existence. To dělá z problému čínského listonoše již optimalizační úlohu.



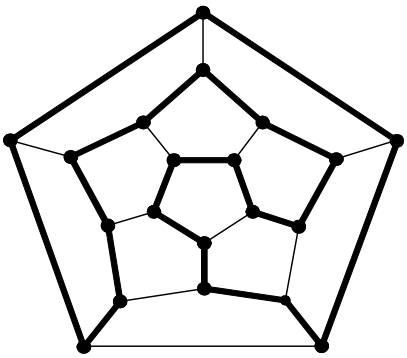
Obr. 4. Historická mapa Královce [15] a nákres rozložení jeho mostů, upraveno dle [16]

V 19. století se poprvé objevuje další ze známých problémů v teorii grafů, a tím je problém barvení grafu. Roku 1852 se snažil Francis Guthrie vybarvit mapu anglických hrabství různými barvami, přičemž si povšimnul, že mu postačují pro tento úkol pouze čtyři barvy [17]. Guthrie vyslovil hypotézu, že na vybarvení jakékoliv mapy tak, aby dvě části spojené hranou neměly stejnou barvu, mu postačují právě čtyři barvy. Platnost tohoto tvrzení se mu nepodařilo prokázat. Započala tak více než stoletá snaha tuto hypotézu potvrdit matematickým důkazem. V průběhu této doby se vyskytlo množství možných důkazů, ale vždy se potvrdila jejich neplatnost. Až v roce 1976 zveřejnili autoři Appel a Haken [18, 19] důkaz, ve kterém platnost tohoto výroku potvrzují.



Obr. 5. Problém barvení grafu spolu s jeho zobrazením do grafu, autoři kompletního důkazu Appel a Haken [11, 20]

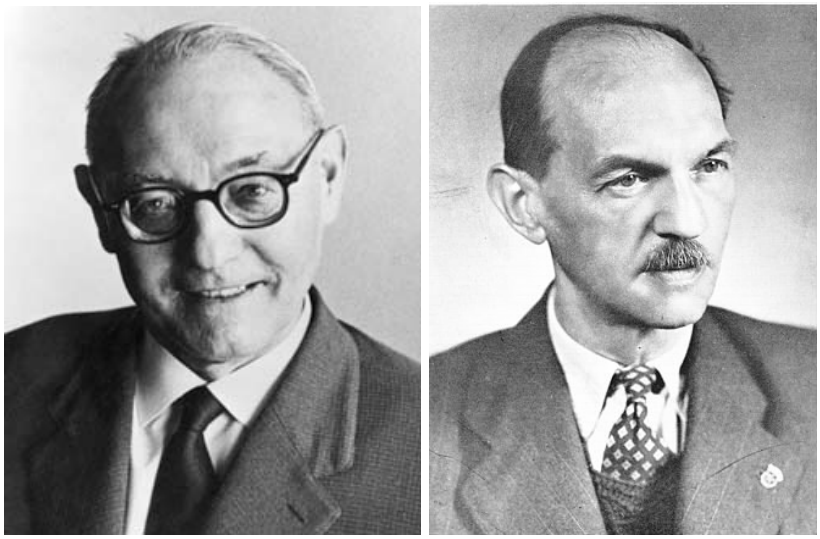
O pár let později, co Francis Guthrie vybarvil mapu anglických hrabství pomocí čtyř barev, vymyslel William Rowan Hamilton hru, jejímž cílem je nalezení posloupnosti vrcholů, která v pravidelném dvanáctistěnu prochází jeho vrcholy pouze a právě jednou. Hru pojmenoval „*Icosian game*“ po pravidelném dvanáctistěnu neboli ikosaedru. Souvislý neorientovaný graf, jehož každý vrchol inciduje právě se dvěma hranami, označujeme jako kružnici. Kružnice, která splňuje podmínky *„Icosian game“, je* dnes označována jako hamiltonovská kružnice nebo také hamiltonovský cyklus.



Obr. 6. Hamiltonovský cyklus v Icosian game

Jestliže libovolný podgraf hranami spojuje každý vrchol původního grafu a neobsahuje žádnou kružnici, bude se jednat o kostru grafu. S rozvojem elektrifikace na konci 19. a začátku 20. století vznikla potřeba efektivně plánovat budování elektrické sítě. Podobně tomu tak bylo i zde v Brně, kde pracovník Západomoravských elektráren Jindřich Saxel požádal svého přítele Otakara Borůvku, zda jim nemůže s tímto problémem pomoci. Zadání bylo prosté, navrhnout co nejlevnější, a tedy nejkratší elektrickou síť spojující města v západní části Moravy. Načež Borůvka v roce 1926 publikoval článek „*O jistém problému minimálním*“ v časopise moravské přírodovědecké společnosti a ve zjednodušené podobě v časopise Elektrotechnický obzor [21]. Tyto články

obsahují popis algoritmu pro nalezení minimální kostry grafu a je dnes označován jako Borůvkův algoritmus. V průběhu dalších let byl algoritmus několikrát znovuobjevován, například Steinhausem a Sollinem [22, 23]. V 50. letech se abstrakt Borůvkova článku dostal do rukou Josepha Kruskala v Princetonu ve Spojených státech, kde mu posloužil jako inspirace pro jeho algoritmus [24].



Obr. 7. Otakar Borůvka a Vojtěch Jarník [25]

Na Borůvkův článek zareagoval v roce 1930 Vojtěch Jarník v příspěvku „*O jistém problému minimálním. (Z dopisu panu O. Borůvkovi)*“ [26]. Zde Jarník navrhuje jednodušší způsob nalezení kostry grafu. Podobně jako Borůvkův algoritmus je jeho postup stejně tak znovuobjevován, nejprve Robertem Primem v roce 1957 [27] a o dva roky později Edsgerem Dijkstrou [28]. Jarníkův algoritmus je tak v zahraničí často označován jako Primův, případně jako DJP neboli Dijkstra-Jarník-Prim algoritmus.

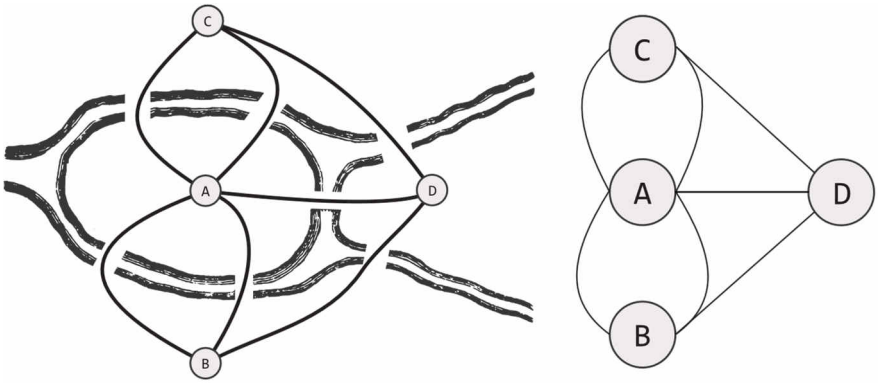
Jen několik let před vypuknutím druhé světové války vyšla monografie věnovaná zcela teorii grafů. Monografie Dénese Kőniga „*Theorie der endlichen und unendlichen Graphen*“ se snažila zahrnout veškeré dostupné informace o grafech a v podstatě tak odstartovala éru teorie grafů jako samostatné matematické disciplíny [29]. S poválečným rozvojem kybernetiky a výpočetní techniky se zájem o teorii grafů začal rapidně zvyšovat. V roce 1958 vychází druhá kniha o teorii grafů a jejích aplikacích ve francouzštině od Clauda Bergeho s názvem „*Théorie des Graphes et ses Applications*“. Načež o pár let později vychází první knihy v angličtině, a to ve Spojených státech. První z nich byla „*Theory of Graphs*“, jejímž autorem je norský matematik Øystein Ore z Yaleovy univerzity. Druhou z nich byla kniha Franka Hararyho „*Graph Theory*“ vydaná roku 1969. V této době publikují své práce o teorii grafů lidé jako Paul Erdős, který byl studentem Dénese Kőniga a spolupracoval i s Otakarem Borůvkou, Edsgerem W. Dijkstrou nebo Williamem Tuttem. Jejich práce a samozřejmě i mnoha dalších pomohla dostat teorii grafů do popředí nejen teoretického výzkumu, ale i praktického využití [30].

Například v matematické optimalizaci, kam spadá i téma této práce, tvoří aparát teorie grafů důležitý prostředek pro vizualizaci a analýzu problémů. Mezi těmito problémy rozpoznáváme několik kategorií problémů, jako je například hledání optimálních cest v grafu, síťová analýza, toky v dopravních sítích, anebo konstrukční úlohy. V případě hledání optimálních cest v grafech můžeme zařadit do této kategorie problémy jako je hledání nejkratší cesty, nebo hledání nejspolehlivější cesty či cesty s maximální kapacitou. U síťové analýzy se zaměřujeme na analýzu projektů a nalezení kritických činností, které mohou způsobit opoždění celého projektu. Do

kategorie konstrukčních úloh patří problém hledání kostry grafu, eulerovského tahu a okruhu, jako je tomu v případě problému čínského listonoše. Sem také patří i úloha hledání hamiltonovské kružnice a její varianta hledání minimální hamiltonovské kružnice, jež je označována jako problém obchodního cestujícího. Teorie grafů spolu s algoritmy matematické optimalizace má tak široké využití při řešení reálných problémů. Zde lze uvést například navigaci pomocí systému GPS a jemu podobných, která nachází řadu využití od armády přes osobní dopravu až po výběr mýta nákladních aut.

2.4.2 Grafy

Vrátíme-li se k Eulerovu problému v Královci, při řešení tohoto problému si Euler uvědomil, že podoba území, která mosty spojují, není důležitá. Co však je důležité, je počet mostů a odkud kam vedou mezi jednotlivými částmi města. To nám umožňuje vytvořit zjednodušenou grafickou reprezentaci tohoto problému, jak je zobrazena na následujícím obrázku.



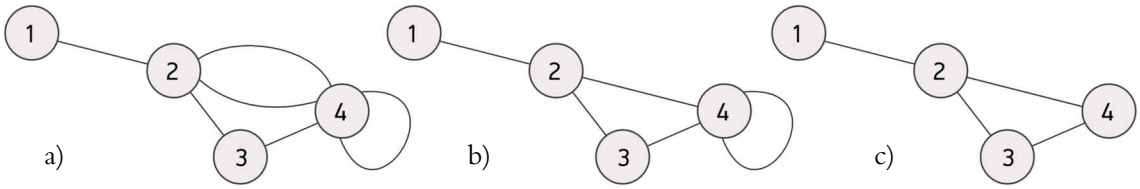
Obr. 8. Reprezentace grafu v mapě mostů v Královci (upraveno podle [16]), samostatné zobrazení tohoto grafu

Plochy, jež spojují mosty, jsou reprezentovány jako body, v teorii grafů je označujeme termínem *uzly* nebo také *vrcholy*. Samotné mosty mají podobu spojů mezi těmito vrcholy a označujeme jako *hrany*. Celou tuto grafickou reprezentaci pak označujeme jako graf. Jeho definice je následující:

$$G = (V, E);\ \forall e \in E;\ e = \{u, v\}, u, v \in V. \tag{2.4}$$

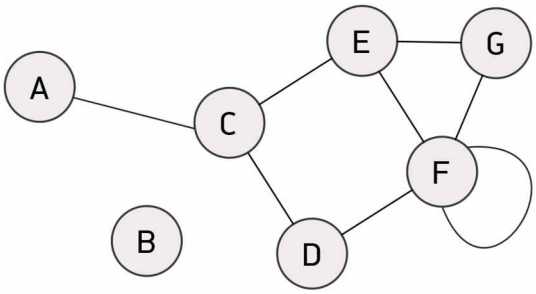
Graf je dvojice množin $G = (V, E)$, kde V je konečná neprázdná množina vrcholů a množina hran E sestává z dvouprvkových podmnožin z množiny V. Jsou-li dva vrcholy spojeny dvěma nebo více hranami, jako je tomu na obrázku výše, jedná se o takzvaný multigraf. Tyto hrany označujeme jako násobné. V případě, že hrana je reprezentována jediným vrcholem, mluvíme o vrcholu se smyčkou. Hrany grafu mohou mít ohodnocení neboli váhu, takovémuto grafu pak říkáme ohodnocený graf či vážený graf. Ohodnocení hran můžeme ilustrovat na příkladu železniční soustavy mezi evropskými městy, které tvoří vrcholy našeho grafu a tratě jsou jednotlivé hrany. Ohodnocení těchto hran pak může označovat délku tratě mezi dvěma městy, délku jízdy nebo kapacitu takovéto tratě.

Grafy můžeme klasifikovat dle různých parametrů. Jedním z parametrů je obsah násobných hran a smyček. Grafy pak rozdělujeme následovně: multigraf, prostý graf a obyčejný graf.



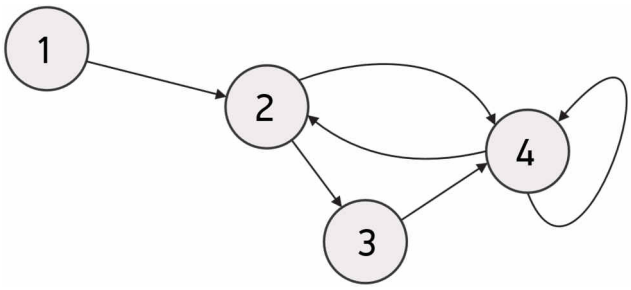
Obr. 9. a) multigraf, b) prostý graf, c) obyčejný graf

Multigraf obsahuje násobné hrany a zároveň může mít vrcholy obsahující smyčky, naproti tomu prostý graf obsahuje smyčky, ale ne násobné hrany. Obyčejný graf, je takový, který neobsahuje vícenásobné hrany a ani smyčky. Každému vrcholu v grafu G, lze určit jeho stupeň značený $deg(v)$. Stupeň vrcholu je dán počtem hran, které jsou s tímto vrcholem spojeny. Má-li vrchol smyčku, zvyšuje se jeho stupeň o dva. Více na následujícím obrázku:



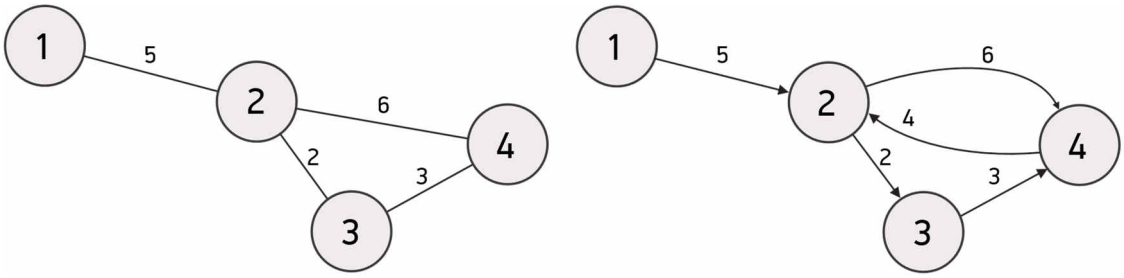
Obr. 10. Stupně vrcholů, $\mathrm{d(V)} = \{(\mathrm{A},1),(\mathrm{B},0),(\mathrm{C},3),(\mathrm{D},2),(\mathrm{E},3),(\mathrm{F},5),(\mathrm{G},2)\}$

Doposud jsme pracovali s neorientovaným grafem, jehož hrany jsou obousměrné. V teorii grafů existují i grafy, které obsahují orientované hrany, pomocí kterých vyjadřujeme jejich směr. Definice orientovaného grafu je následující: graf je dvojice $G = (V, E)$ dvou množin, V je konečná neprázdná množina vrcholů a množina hran E sestává z uspořádaných dvouprvkových podmnožin z množiny V. Každá orientovaná hrana má svůj počáteční a koncový vrchol. Podobně jako u neorientovaného grafu určujeme pro daný vrchol v jeho stupeň, který se dělí na výstupní stupeň, značený $deg^{-}(v)$ pro hrany jež vychází z daného vrcholu. Pro příchozí hrany určujeme vstupní stupeň, značený $deg^{+}(v)$.



Obr. 11. Orientovaný graf

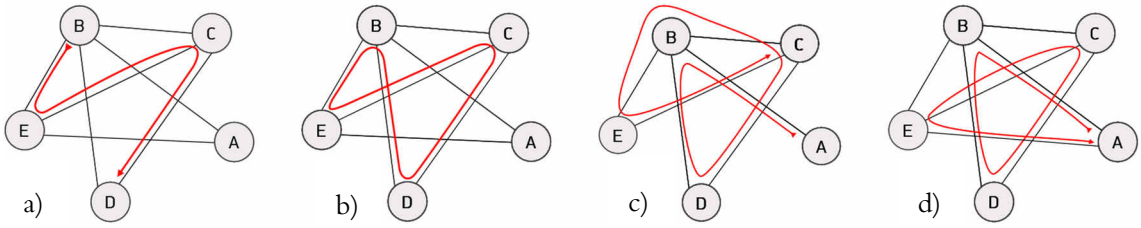
Jak u neorientovaného, tak i u orientovaného grafu mohou mít hrany ohodnocení, které může vyjadřovat například vzdálenost mezi dvěma vrcholy, jež tato hrana spojuje. U takovéhoto ohodnoceného grafu může mít každá hrana spojující dva stejné vrcholy různé ohodnocení, případně můžeme mít i více ohodnocení pro každou z hran.



Obr. 12. Ohodnocený neorientovaný a orientovaný graf

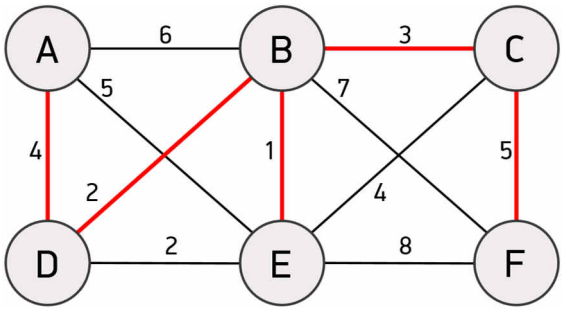
Základní operací, se kterou se setkáváme u grafu, je jeho procházení, například z vrcholu x do vrcholu y. Při průchodu grafem rozlišujeme způsob zápisu takovéhoto procházení. Prvním způsobem průchodu je sled, jde o následující posloupnost $v_0, e_1, v_1, e_2, \ldots, v_k$ vrcholů grafu v_i a hran grafu e_i pro $1 \leq i \leq k$ s počátečním vrcholem v_{i-1}a koncovým vrcholem v_i. Ve sledu se můžou jak vrcholy, tak i hrany opakovat, jeho délku měříme počtem hran. Oproti tomu tah je varianta sledu, v němž se žádná hrana neopakuje. Eulerův tah je takový tah, jenž obsahuje všechny hrany v grafu G, eulerovský okruh pak uzavírá tento tah zpět do startovního vrcholu. Cesta je varianta tahu, v němž se neopakují žádné vrcholy.

V případě naší úlohy sedmi mostů v Královci je jejím cílem nalezení právě eulerovského tahu. Jak bylo uvedeno dříve, Euler a poté i definitivně Hierholzer dokázal, že nezbytnou podmínku pro splnění této úlohy je, aby každý vrchol měl sudý stupeň.

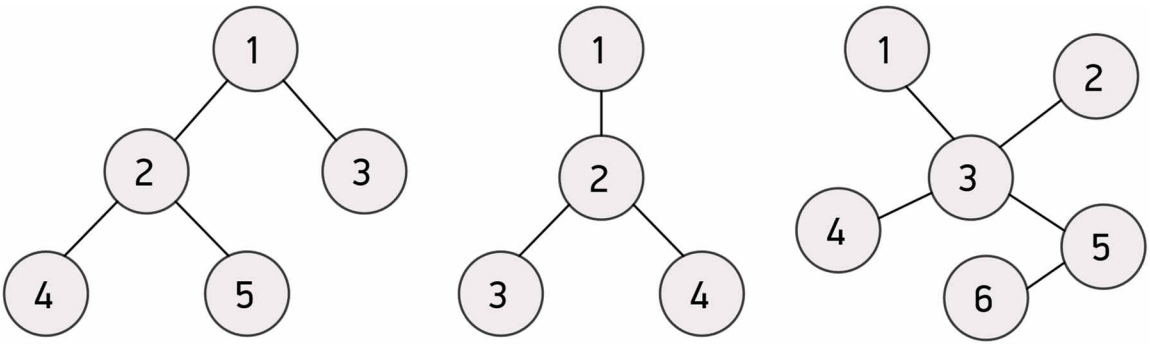


Obr. 13. a) cesta, b) cyklus, c) eulerovský tah, d) hamiltonovský cyklus

Dalším problémem, na němž si můžeme ilustrovat některé pojmy z teorie grafů, je hledání minimální kostry grafu. Kostru grafu G tvoří strom obsahující všechny vrcholy z tohoto grafu. Stromem grafu pak nazýváme souvislý podgraf grafu G, který neobsahuje kružnice. Minimální kostra ohodnoceného grafu je kostra s minimálním součtem hranových ohodnocení. Ve svých článcích Otakar Borůvka a Vojtěch Jarník uvádí algoritmy pro její nalezení.

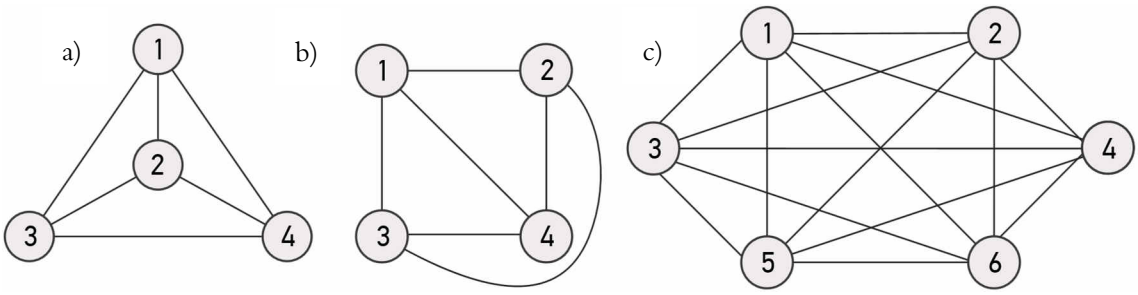


Obr. 14. Minimální kostra grafu



Obr. 15. Podoby stromových grafů

Grafy také můžeme klasifikovat dle jejich podoby, například regulární graf má stejný stupeň pro všechny vrcholy grafu. Rovinný (planární) graf je takový graf, v jehož zobrazení se žádné hrany nekříží. Oproti tomu jsou u úplného grafu všechny vrcholy navzájem pospojovány hranami. S úplným grafem se například můžeme setkat u problému obchodního cestujícího, který bude popsán v následující podkapitole.



Obr. 16. a) regulární graf, b) rovinný graf a c) úplný graf

2.4.3 Klasické metody prohledávání grafu

V předchozí části této práce jsme uvedli řadu problémů, které využívají grafových struktur pro jejich řešení. Mezi takové problémy patří průchod grafem při splnění určitých omezujících podmínek, ať už se jedná o problém čínského listonoše, hledání minimální kostry grafu a další. Problém nalezení nejkratší cesty v grafu (*SPP – Shortest path problem*) je základním problémem, se kterým se setkáváme. Rozpoznáváme následující tři podoby tohoto problému. Prvním je hledání nejkratší cesty (*shortest path*) mezi dvěma vrcholy v grafu, kde máme zadán počáteční vrchol s a cílový vrchol t. Druhým případem je nalezení nejkratší cesty z počátečního vrcholu s do všech zbylých vrcholů $v \in V$ v grafu (*single source shortest path*). Tento případ má i svoji opačnou úlohu, a to nalezení nejkratší cesty pro všechny vrcholy v grafu $v \in V$ do jednoho cílového vrcholu t. Poslední variantou je nalezení nejkratší cesty pro každý pár vrcholů v grafu (*all pairs shortest path*).

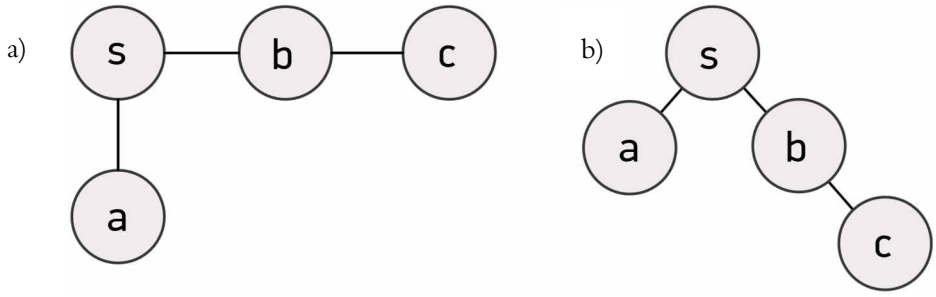
Metody prohledávání grafu, které patří do skupiny klasických algoritmů, rozdělujeme na*neinformované* a *informované*. Nejkratší cestu můžeme hledat jak v orientovaných, neorientovaných tak i ohodnocených a neohodnocených grafech a máme k dispozici řadu algoritmů pro její nalezení. Máme-li najít nejkratší cestu v neohodnoceném grafu, je délka cesty dána počtem hran mezi startovním vrcholem u a cílovým vrcholem v. Pro nalezení nejkratší cesty v neohodnoceném grafu můžeme použít algoritmus prohledávání grafu do šířky (*BFS – Breadth-first search*). Spolu s algoritmem prohledávání grafu do hloubky (*DFS – Depth-first search*) patří k základním algoritmům pro procházení grafu. Autorem algoritmu hledání do šířky je E. F. Moore,

který jej použil pro nalezení nejkratší cesty z bludiště v 50. letech 20. století [31]. Jelikož u těchto metod dochází k rozsáhlému prohledání stavového prostoru, jsou tyto metody pro plánování cesty nevhodné.

Algoritmus 1 Hledání do šířky (BFS)

```
BFS(G:graph, s:start)
 1  for each vertex v ∈ V_G
 2        distance[v] ← ∞
 3        parent[v] ← nil
 4
 5  distance[s] ← 0
 6  Q ← s
 7
 8  while Q ≠ ∅
 9        u ← DEQUEUE(Q)
10        for each edge e = (u, v)
11              if distance[v] == ∞
12                    distance[v] ← distance[u] + 1
13                    parent[v] ← u
14                    Q ← v
```

V prvotní inicializační části se nastaví hodnoty pro všechny vrcholy u v grafu G na ∞ a vytvoří se fronta Q, do které se vloží startovní vrchol s. Pokud není fronta Q prázdná, vyjme se vrchol u z fronty a pro všechny jeho sousední vrcholy v , které nejsou zároveň již označeny jako navštívené, se nastaví nová hodnota vzdálenosti a ukazatel na předchozí vrchol. Dále se tyto vrcholy přidají do fronty Q a dokud není tato fronta prázdná, opakujeme celý postup od výběru vrcholu z fronty. Jestliže je fronta Q prázdná, je prohledávání grafu ukončeno, všechny vrcholy byly již prohledány. Po dokončení algoritmu budeme mít zjištěny délky nejkratších cest ze startovního vrcholu s do všech vrcholů grafu G, které jsou dosažitelné z tohoto vrcholu. Cestu z vrcholu u do vrcholu v, získáme zpětným průchodem pomocí odkazů na předchozí vrchol z vrcholu v. Časová složitost prohledávání do šířky je $\mathcal{O} = (|V| + |E|)$, kdy složitost $\mathcal{O} = (|E|)$ je závislá na hustotě grafu a může být $\mathcal{O} = (1)$ až $\mathcal{O} = (|V^2|)$.



Obr. 17. a) vstupní graf, b) postup prohledávání do hloubky

U ohodnoceného grafu platí pro nejkratší cestu $p^*(u, v)$ mezi vrcholy u a v v grafu G následující:

$$\sum_{e \in p^*(u,v)} w(e) = min_{p(u,v) \in P} \left\{ \sum_{e \in p(u,v)} w(e) \right\}, \tag{2.5}$$

kde P je množina všech cest z vrcholu u do vrcholu v, $p(u,v)$ je cesta z vrcholu u do vrcholu v, $p(u,v) \in P$. Ohodnocení hran $w(e) = w(v_i, v_j)$ může mít jak kladné i záporné hodnoty.

V případě informovaných prohledávacích metod jsou využívány znalosti o stavovém prostoru k omezení jeho nadbytečného prohledávání. Mezi tyto znalosti o stavovém prostoru patří heuristická funkce, která ohodnocuje každý stav dle daných kritérií. Ohodnocení je využito při prohledávání pro výběr expandovaného vrcholu. Jedním z nejznámějších a nejčastěji používaných algoritmů pro kladně ohodnocené grafy je Dijkstrův algoritmus [28]. Kořeny tohoto algoritmu sahají až do roku 1956, kdy Edsgera W. Dijkstru napadla myšlenka zjistit, jaká je nejkratší cesta z Rotterdamu do Gronigenu [32]. Během dopolední kávy přišel na první variantu algoritmu pro nalezení nejkratší cesty. Rok poté algoritmus upravil pro řešení praktické úlohy nalezení nejkratší délky kabeláže pro počítač na univerzitě v Rotterdamu. Shodou okolností tak znovuobjevil Jarníkův a zároveň Primův algoritmus, což ovšem v té době netušil. Dijkstra tento algoritmus publikoval v roce 1959, dva roky po Primově [27] a téměř třicet let po Jarníkově [26] publikaci tohoto algoritmu. Dijkstrův algoritmus však není zcela stejný jako Jarníkův nebo Primův algoritmus. Rozdíl mezi nimi je tvořen odlišnou relaxační funkcí, která se stará o vylepšování ohodnocení. V případě Dijkstrova algoritmu má relaxační část podobu $distance[v] = distance[u] + weight(u,v)$, kdežto u Jarníkova / Primova algoritmu je relaxační funkce $distance[v] = weight(u,v)$. To má za následek, že Dijkstrův algoritmus nalezne vždy nejkratší cestu z daného vrcholu do všech ostatních vrcholů grafu, pokud daná cesta existuje. Dijkstrův algoritmus funguje jak na neorientovaných, tak i orientovaných grafech, oproti tomu Jarníkův nebo Primův algoritmus pracuje na neorientovaných grafech, jelikož hledání minimální kostry grafu se provádí pouze na nich. Dalším rozdílem mezi těmito algoritmy je v obsahu negativně ohodnocených hran (*negativní váhový cyklus*). Obsahuje-li graf takto ohodnocené hrany, nelze na něj použít Dijkstrův algoritmus. V takovémto případě lze použít *Bellman-Fordův algoritmus*, který zvládá negativní ohodnocení hran v grafu, jeho nevýhodou je menší rychlost. V následujícím pseudokódu je uveden Dijkstrův algoritmus pro nalezení nejkratších cest v grafu.

Algoritmus 2 Dijkstrův algoritmus

```
DIJKSTRA(G:graph, s:start)
 1  // Inicializace
 2  for each vertex v ∈ V_G
 3       distance[v] ← ∞
 4       parent[v] ← nil
 5
 6  distance[s] ← 0
 7  Q ← V_G
 8
 9  while Q ≠ ∅
10       u ← EXTRACT-MIN(Q)
11       for each edge e = (u, v)
12            if distance[v] > distance[u] + weight[e]
13                 distance[v] ← distance[u] + weight[e]
14                 parent[v] ← u
15
16  return distance[], parent[]
```

Samotný algoritmus je následující. Máme zadán graf $G = (V, E)$ a startovní vrchol s. Dále je pro každý vrchol vytvořena proměnná $distance[v]$, jež obsahuje délku nejkratší dosud nalezené cesty z vrcholu s do v. Druhou proměnnou je $parent[v]$ obsahující ukazatel na bezprostředního předchůdce vrcholu v na této dosud nejkratší cestě. Tato proměnná je určená k vytvoření cesty z cílového vrcholu do startovního. Na počátku algoritmu probíhá inicializace, kdy pro každý

vrchol $v \in V$ se nastaví hodnota $distance[v]$ na takovou hodnotu, která představuje nekonečno, krom startovního vrcholu, jehož hodnota je nastavena na nulu. Spolu s touto hodnotou se nastavuje i hodnota $parent[v]$ na hodnotu, která neodkazuje na ani jeden z vrcholů v grafu. Dalším krokem je vytvoření množiny Q, která bude obsahovat všechny vrcholy z grafu včetně startovního vrcholu s. Z této množiny získáme vrchol s nejmenší hodnotou $distance[u]$ dle řádku č. 10 v pseudokódu a pro všechny jeho nenavštívené sousední vrcholy zjistíme ohodnocení do nich vedoucích hran. K ohodnocení těchto hran přičteme ohodnocení $distance[u]$ a v případě, že je tento součet menší než aktuální hodnota $distance[v]$, je tato hodnota nahrazena tímto novým ohodnocením a také hodnota $parent[v]$ je nastavena na ukazatel na u. Tento postup od získání vrcholu s nejmenší hodnotou $distance[u]$ z množiny Q opakujeme, dokud není tato množina prázdná. Po dokončení budeme mít délku nejkratší cesty pro každý z vrcholů $v \in V$ v grafu, uloženu v $distance[v]$. Ke zjištění samotné cesty z vybraného vrcholu do startovního pak pokračujeme zpětným průchodem, kdy za pomoci ukazatelů uložených v $parent[v]$ se dostaneme až na startovní vrchol. Efektivita Dijkstrova algoritmu závisí na zvolené implementaci grafu a prioritní fronty, kdy mezi nejjednodušší varianty prioritní fronty patří implementace pomocí pole nebo seznamu. Získání vrcholu s minimálním ohodnocením se tak provádí pomocí lineárního prohledávání a časová složitost Dijkstrova algoritmu je v takovém případě $\mathcal{O}(|E| + |V|^2) = \mathcal{O}(|V|^2)$. Dalším faktorem ovlivňujícím časovou složitost je hustota grafu. Například pro řídké grafy, tj. grafy jejichž počet hran je menší než n^2, kde n je počet vrcholů v grafu, je vhodnější použít například binární nebo Fibonacciho haldu. V prvním případě je pak časová složitost $\mathcal{O}((|E| + |V|^2) \log|V|)$ a v případě Fibonacciho haldy $\mathcal{O}(|E| + |V| \log|V|)$.

Bellman-Fordův algoritmus můžeme použít v případě grafu obsahujícího záporně ohodnocené hrany. Podobně jako u Dijkstrova algoritmu, používá se zde relaxační funkce, při které je hodnota vzdálenosti postupně upravována. Avšak oproti Dijkstrově algoritmu nedochází k uzavření vrcholu po projití všech jeho následovníků, kdy hodnota vzdálenosti se již neupravuje. U Bellman-Fordova algoritmu se prochází vrcholy několikrát, přesněji v počtu $|V| - 1$ opakování a relaxací je postupně vzdálenost upravována. Navíc algoritmus umožnuje detekci záporně ohodnocených cyklů, při jejichž existenci je úkol nalezení nejkratší cesty zbytečný.

Poprvé byl tento algoritmus navrhnut Shimbelem v roce 1955 a během následujících pěti let byl nezávisle publikován Richardem Bellmanem. Lesterem Fordem a Edwardem F. Moorem. Proto se také někdy označuje jako Bellmanův-Fordův-Moorův algoritmus [33].

Algoritmus 3 Bellmanův-Fordův algoritmus

```
BELLMAN-FORD(G:graph, s:start)
 1  for each vertex v ∈ V_G
 2      distance[v] ← ∞
 3      parent[v] ← nil
 4
 5  distance[s] ← 0
 6
 7  for i ← 1 to |V_G| − 1
 8      for each edge e = (u, v)
 9          if distance[v] > distance[u] + weight[e]
10              distance[v] ← distance[u] + weight[e]
11              parent[v] ← u
12
13  for each edge e = (u, v) ∈ E_G
14      if distance[v] > distance[u] + weight[e]
15          error "Graph contains a negative-weight cycle"
16
17  return distance[], parent[]
```

Dalším významným algoritmem je algoritmus A* (A star), který byl publikován v článku *„A Formal Basis for the Heuristic Determination of Minimum Cost Paths"* autory Hartem, Nilssonem a Raphaelem v roce 1968 [34]. Algoritmus A* navazuje na Dijkstrův algoritmus a rozšiřuje jej o část heuristického odhadu. Hlavní myšlenkou tohoto heuristického odhadu či chceme-li heuristické funkce je, že směr hledání by měl být orientován k cíli. Samotný heuristický odhad je pak součástí následující rovnice:

$$f(x) = g(x) + h(x), \tag{2.6}$$

kde délka cesty $g(x)$ představuje vzdálenost mezi startovním vrcholem a aktuálním vrcholem x. Heuristický odhad $h(x)$ je odhadovaná vzdálenost z aktuálního vrcholu x do cílového vrcholu. Pro tento odhad platí podmínka přípustnosti dle rovnice (2.7), podle níž nesmí nadhodnocovat vzdálenost k cíli nad skutečnou vzdálenost a kde $\mathrm{h}^*(\mathrm{x})$ je vzdálenost do cíle.

$$h(x) \leq h^*(x) \tag{2.7}$$

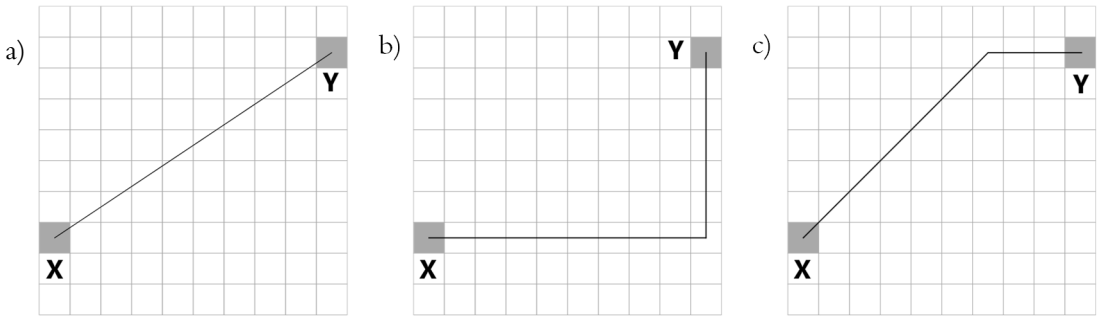
Jestliže je tato podmínka splněna, algoritmus garantuje nalezení nejkratší cesty, pokud taková cesta existuje [34, 35]. V případě, že heuristická funkce není přípustná, nemusí být nalezené řešení optimální. Jako heuristickou funkci při hledání nejkratší cesty lze použít například vybrané metriky z teorie metrických prostorů [36]. Příkladem je euklidovská metrika, která představuje vzdálenost mezi dvěma stavy $X = (x_1, x_2)$, $Y = (y_1, y_2)$ a můžeme ji zapsat v $\mathbb{R}^2$:

$$\rho_1(X, Y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}. \tag{2.8}$$

Mezi další významné metriky patří např. manhattanská, maximální (Čebyševova) a octile metrika. Manhattanská metrika je inspirovaná ulicemi na Manhattanu v New Yorku, které mají pravoúhlý systém. Její zápis v $\mathbb{R}^2$ je následující:

$$\rho_2(X, Y) = |x_1 - y_1| + |x_2 - y_2|. \tag{2.9}$$

Octile metrika oproti manhattanské pracuje s diagonálním směrem pohybu, viz následující obrázek. Výběr metriky závisí na podobě prohledávaného prostoru, zda například se lze pohybovat je v kardinálním nebo i v diagonálním směru.



Obr. 18. Grafická reprezentace metrik a) Euklidovská b) Manhattanská c) a Octile metrika

Průběh A* je zobrazen na následujícím pseudokódu. Podobně jako u Dijkstrova algoritmu nejprve vytvoříme prioritní frontu, zde označenou *open*, dosud nenavštívených stavů (vrcholů v grafu), které jsou seřazeny dle nejnižší hodnoty $f(v)$. V cyklu, který probíhá, dokud není fronta *open* prázdná nebo nenarazíme na cílový stav, vyjmeme v každém kroku stav u z této fronty. Pro všechny sousední stavy určíme nové hodnoty $f(v)$, tedy součtu vzdálenosti od startu a heuristického odhadu vzdálenosti do cíle. Pokud není daný sousední stav ve frontě *open*, je do ní přidán. Je-li stav již ve frontě *open*, ale jeho nová hodnota $f(v)$ je nižší než stávající, je pak tato hodnota upravena na novou. Aktuální stav u je na závěr přesunut do seznamu *closed* a pokračujeme v cyklu while.

Algoritmus 4 A* algoritmus

```
A-STAR(G:graph, s:start, t:target)
 1  open ← start
 2  closed ← ∅
 3
 4  while open ≠ ∅
 5      u ← EXTRACT-MIN(open)
 6      if u == t break
 7
 8      for each edge e = (u, v)
 9          distance ← g[u] + weight[e]
10
11          if v ∉ open or distance < g[v]
12              g[v] ← distance
13              h[v] ← DISTANCE-TO-TARGET(v, t)
14              f[v] ← g[v] + h[v]
15              open ← v
16
17      closed ← u
```

Časová složitost algoritmu se odvíjí od použité heuristiky, která má hlavní vliv na výběr vrcholů k prohledání. V nejhorším případě může být $\mathcal{O}(b^d)$, kde b je faktor větvení, tj. průměrný počet hran pro každý vrchol, a d je počet vrcholů ve výsledné cestě. S lepší heuristikou bude klesat počet navštívených vrcholů. Oproti tomu, v optimálním případě, v němž má prohledávaný graf podobu stromu, může dosahovat polynomiální časové složitosti.

Po publikování algoritmu A* vzniklo množství jeho variant a rozšíření, jako jsou například algoritmy IDA* (*Iterative-Deepening A**) [37], D* [38], D* lite [39], LPA* (*Lifelong planning A**) [40] nebo Theta* [41]. Mezi další algoritmy, patří Floyd-Warshallův, ale také Beam search [42], Fringe search [43], Johnsonův algoritmus [44], SMA* [44, 45], JPS [46, 47] a SubGoal [48, 49]. Další metody lze naleznout například zde [50].

Metody umělé inteligence

Pro plánování cesty lze také použít metod umělé inteligence. Skupina těchto metod je velmi bohatá na způsoby řešení problémů. Můžeme se tak setkat nejenom s metodami, které se inspirují chováním živočichů, jako jsou mravenci, včely a další druhy, ale také s metodami, které jsou založeny na evoluci nebo na struktuře nervového systému. Na následujících řádcích lze nalézt jen velmi stručné informace o některých z těchto metod, zejména těch, jež patří do stejné skupiny jako metoda mravenčího roje, která byla vybrána pro řešení cílů disertační práce. Metodě mravenčí kolonie je věnována samostatná kapitola. Podrobné informace lze o uvedených metodách naleznout v odkazované literatuře.

Genetické algoritmy

Genetické algoritmy (*GA*) patří do skupiny evolučních metod, které napodobují evoluční procesy známé z biologie, jako je křížení, dědění a další. Přípustná řešení tvoří populaci, přičemž každé řešení je jeden jedinec této populace. Tento jedinec je reprezentován chromozómem, který tvoří zakódované informace o daném řešení. Nejčastěji jsou chromozómy tvořeny binárními řetězci nebo se jedná o pole hodnot. Na počátku simulace jsou náhodně vybrána řešení, která tvoří populaci. Pro přechod do nové generace jsou jednotliví jedinci ohodnoceni fitness funkcí, která vyjadřuje, jak kvalitní řešení daný jedinec reprezentuje. Podle této kvality jsou vybíráni jedinci, jež jsou pak modifikováni pomocí genetických operátorů, jako jsou křížení a mutace. Z těchto modifikací dále vzniká nová generace. To se opakuje, než dojde ke splnění ukončovacích podmínek simulace. Algoritmy lze použít na řešení velké škály problémů. Jen pro plánování cesty robotů existuje velké množství metod jak pro diskrétní, ale i spojité prostředí [51, 52]. Například v práci [51] je prezentována paralelní implementace GA pro plánování cesty holonomního robotu, v níž jsou jednotlivé cesty kódovány jako seznamy pohybů robotu.

Rojové inteligence

Rojová inteligence napodobuje nejen kolektivní chování živočichů, jako jsou mravenčí kolonie, hejna ptáků nebo ryb, ale je i inspirována gravitací. Algoritmy rojové inteligence často tvoří roj jednoduchých agentů, v němž se jejich chování navzájem ovlivňuje, a to díky specifické formě komunikace mezi těmito agenty, které je převzato ze svých přírodních vzorů. Příkladem je zanechávání feromonové stopy podél nalezené cesty u mravenčí kolonie nebo včelího tanečku u včelích kolonií. Díky tomuto lokálnímu ovlivňování dochází k vzniku složitého globálního chování a není tak potřeba centrálního řízení. Mezi výhody těchto algoritmů je schopnost se dynamicky přizpůsobovat změnám v prostoru a celková rychlost výpočtu. Do skupiny algoritmů rojové inteligence patří velké množství algoritmů. Mezi nejznámější metody rojové inteligence patří metoda rojení částic, mravenčí kolonie a včelí kolonie. Podrobněji je tématu rojové inteligence věnována samostatná kapitola této práce.

Metoda rojení částic

Tento algoritmus se inspiruje chováním ptačích a rybích hejn [53]. Jednotliví agenti jsou označování jako částice a reprezentují kandidáty řešení daného problému. Tito agenti pak dohromady tvoří hejno. V něm je každá částice definována svým vektorem polohy, vektorem rychlosti a pamětí předchozích úspěchů při hledání. Částice se pohybují v prohledávaném prostoru a jsou naváděny dle své zkušenosti spolu se zkušenostmi okolních částic. Pro řešení problému hledání cesty robotu byla tato metoda použita ať již samostatně [54] nebo v kombinaci s jiným algoritmem [55], zde například spolu s Dijkstrovým algoritmem.

Včelí algoritmy

Mezi další rojové metody patří i včelí algoritmy, které napodobují chování včel medonosných. Mezi nejznámější z nich patří Včelí algoritmus (*Bees Algorithm - BA*) publikovaný v roce 2005 [56] a algoritmus umělého včelího roje (*Artificial Bee Colony Algorithm - ABC*) publikovaný v témže roce [57]. Obě tyto metody se inspirují chováním včel při hledání potravy. Existují však i další varianty včelích algoritmů, například některé z nich se inspirují pářícím chováním včel [58] nebo jejich chováním při hledání nového hnízda [59]. Včelí algoritmus (*BA*) rozděluje včelí roj na dvě části, na včely průzkumnice a vyčkávající včely. Na počátku algoritmu jsou průzkumnice náhodně umístněné v prohledávaném prostoru. V dalším kroku algoritmu je vybrán určitý počet nejlepších včel a jsou označeny jako elitní včely. Poté se k těmto elitním včelám přidružují vyčkávající včely a provádí lokální prohledávání. Pokud bylo nalezeno lepší řešení, je daným průzkumníkem aktualizováno. K vytvoření nové populace pro další iteraci je použito m vybraných průzkumníků z předchozí iterace a n-m nových průzkumníků. To se opakuje, dokud nejsou splněny ukončovací podmínky. V článku [60] je pro řešení problematiky plánování cest pro více robotů použita varianta umělého včelího roje, zde uvedená jako metoda chaotického umělého včelího roje. ABC algoritmus je tak obohacen o dynamiku chaosu, čímž autoři dosáhli vylepšení řešení.

Mezi ostatní metody umělé inteligence, které lze použít pro plánování cesty, patří neuronové sítě [61], případové usuzování [62] nebo posilované učení [63]. V literatuře tak můžeme narazit na použití různých druhů neuronových sítí, které jsou pro tento problém vhodné. Dále se můžeme setkat s neuronovými sítěmi, které při řešení plánování cesty často využívají kombinace s genetickými algoritmy, případovým učením, anebo fuzzy logikou [64]. Případové usuzování řeší problémy při plánování cesty tím způsobem, že se snaží adaptovat známá řešení problémů, které se již v minulosti vyskytly. Například v práci [65] je popsána metoda případového učení pro plánování cesty jak v diskrétním, tak i spojitém prostoru.

3 PROBLÉM OBCHODNÍHO CESTUJÍCÍHO (TSP)

Jedním z nejznámějších optimalizačních problémů je *problém obchodního cestujícího*. Základní podobu problému můžeme obecně definovat následovně. V dané množině měst existuje ohodnocené spojení mezi každými dvěma městy z této množiny. Cílem je nalezení trasy s nejmenším ohodnocením při projití všemi městy a návratem do startovního města.

Obr. 19. Optimální cesta skrz 532 měst v USA [66]

Přesnější definice TSP je následující: mějme množinu měst $\{c_1, c_1, \dots, c_N\}$ a pro každý pár $\{c_i, c_j\}$ rozdílných měst je dána vzdálenost $d(c_i, c_j)$. Cílem je nalézt takové pořadí průchodu π městy včetně návratu do počátečního města, které minimalizuje hodnotu délky cesty:

$$\sum_{i=1}^{N-1} d\left(c_{\pi(i)}, c_{\pi(i+1)}\right) + d\left(c_{\pi(N)}, c_{\pi(1)}\right). \tag{3.1}$$

V případě, že vzdálenosti splňují následující podmínku $d(c_i, c_j) = d(c_j, c_i)$, kdy $1 \leq i, j \leq N$ se jedná o symetrický TSP, v opačném případě jde o asymetrický TSP [67].

Samotný pojem *problém obchodního cestujícího* se ve vědecké publikaci objevil poprvé až v roce 1949 s názvem „On the Hamiltonian Game (A Traveling Salesman Problem)" autorky Julie Robinsonové [68]. Historie tohoto problému je však mnohem delší. Základy problému položil v 19. století W. R. Hamilton se svojí *„Icosian game"*. Jejím cílem bylo nalezení Hamiltonovského cyklu v grafu, jak již bylo uvedeno v předchozí části této práce. Hamilton však nebyl sám, kdo se tomuto problému věnoval. Jak je patrné ze samotného názvu tohoto problému, minimalizovat strávenou dobu na cestách bylo cílem lidí pracujících jako obchodní cestující. A právě pro ně vyšla v roce 1832 příručka „Der Handlungsreisende", jež obsahuje nejen definici tohoto problému, ale i několik doporučených tras po Německu. Tato příručka se objevila ve vědeckých kruzích až v 80. letech 20. století, díky práci Heinera Müller-Merbacha [69], kterému se ji podařilo nalézt v archívech. Studiu tohoto problému se ve 20. letech následujícího století věnoval matematik a ekonom Karl Menger, který jej ve Vídni zpopularizoval mezi kolegy. O několik let později se tak tento problém znovu objevuje na druhé straně oceánu, nejprve na Harvardské univerzitě, kde byl na počátku 30. let Menger na studijním pobytu a následně na univerzitě v Princetonu. A právě zde se tomuto problému věnují v následujícím desetiletí Mahalanobis [70], Jessen [71]. V stejné době problém zpopularizoval v RAND Corporation na západním pobřeží USA Merill Flood, kde později publikovala svůj článek o TSP i již zmíněná Julie Robinsonová. V této společnosti získalo TSP postavení náročného problému kombinatorické optimalizace, jehož řešení zkoušením všech možností je neefektivní. Pokrok v řešení nastal v roce 1954, kdy George Dantzig, Ray Fulkerson a Selmer Johnson z RAND Corporation publikovali práci [72] popisující řešení pro instanci problému obsahujících 49 měst, což byl na tu dobu nevídaný počet. Autorům se podařilo dokázat, že pomocí metod lineárního programování nalezli optimální trasu pro tuto instanci.

O více jak dvacet let později Martin Grotschel nalezl podobným způsobem optimální cestu pro 120 měst v západním Německu. Od konce osmdesátých let dvacátého století začala nová éra řešení tohoto problému, zvláště v roce 1987 došlo k publikování několika významných řešení. V AT&T se podařilo Padbergovi a Rinaldimu najít optimální trasu pro 532 telefonních ústředen ve Spojených státech. Následně Ollaf Holland spolu s Martinem Grotschelem nalezli optimální trasu pro 666 měst rozprostřených na celé planetě, Holland tento výsledek nejprve publikoval ve své disertační práci. Později toho roku se Paddberovi a Rinaldimu podařilo nalézt optimální trasu spojující 2392 bodů na tištěném spoji firmy Tektronics Incorporated. Na počátku 90. let vzniká knihovna testovacích úloh pod názvem TSPLIB, vytvořil ji Gerhard Reinelt, která se používá dodnes pro porovnání úspěšnosti optimalizačních technik [73]. O pár let později, v roce 1994, vyřešili Applegate, Bixby, Chvátal a Cook úlohu se 7397 městy, čímž začala úspěšná série řešení tohoto týmu. V průběhu dvaceti let se jim podařilo nalézt optimální řešení úloh o stále větším počtu měst. Například: rok 1998, úloha o 13 509 městech, rok 2004 a úloha o 24 978 městech a prozatím poslední a největší úloha je z roku 2006 o velikosti 85 900 měst [66]. Instance o daleko větší velikosti se v současnosti daří řešit s výsledky, jež jsou velmi blízko optimálnímu řešení.

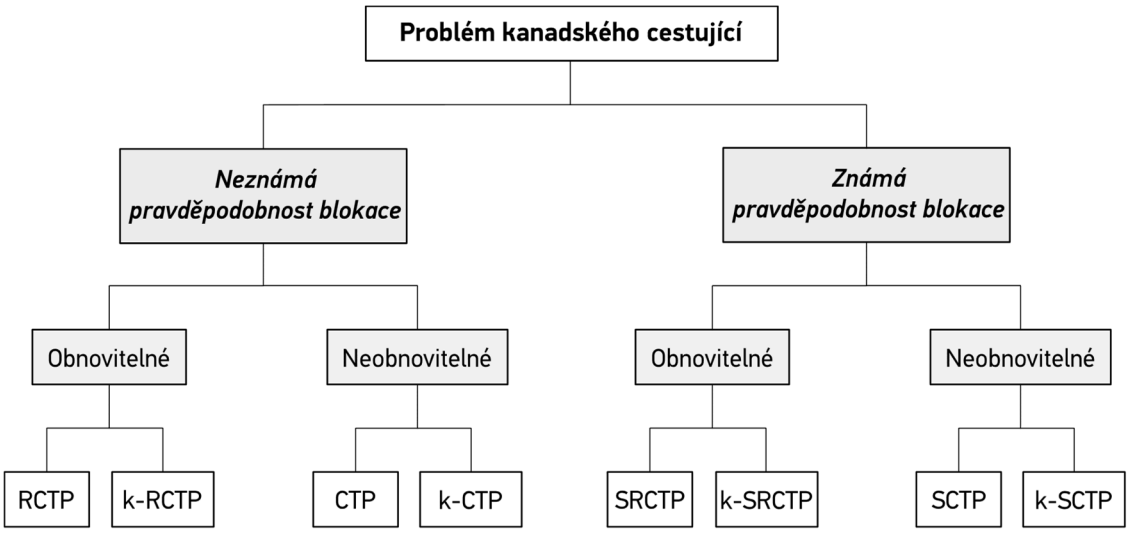
Pro úlohu obchodního cestujícího existuje několik modifikací, mezi které patří například Úloha obchodního cestujícího s časovými okny (TSP with Time Windows). V této variantě je pro každé město přiřazen časový interval (časové okno), ve kterém je možno dané město navštívit. Jako příklad lze uvést letecké spoje mezi letišti. Více informací o této variantě lze naleznout například zde [74–76]. Další variantou TSP je Úloha s více návštěvami (TSP with multiple visits). Tato varianta se od původního TSP odlišuje, jak už název napovídá tím, že můžeme navštívit města opakovaně. Existuje i rozšíření této varianty, kdy je specifikováno, kolikrát se města mají navštívit, každé město se však musí navštívit alespoň jedenkrát. Cílem je opět minimalizovat délku cesty při projití všemi městy s návratem do startovní pozice. Více informací k této variantě je uvedeno v [77, 78]. Další varianty, modifikace a řešení TSP lze naleznout v knize „*The Traveling Salesman Problem and Its Variations*“ z roku 2007 [79]. Poznamenejme, že existuje mnoho variant TSP, včetně symetrického TSP, kde vzdálenost mezi dvěma městy je stejná v obou směrech a asymetrického TSP, kde se vzdálenosti liší v závislosti na směru.

Metody pro řešení TSP lze rozdělit na exaktní, heuristické, aproximační a stochastické. Exaktní metody zaručují nalezení optimálního řešení. Příkladem jsou metody založené na principu větví a mezí (*branch & bound*), jež jsou založeny na větvení úlohy na jednodušší podúlohy. V průběhu větvení dochází k prořezání podúloh, a to o takové, které neobsahují optimální řešení. Tímto způsobem se zamezí zbytečnému prohledávání těchto neperspektivních větví. Další příkladem jsou metody kombinující princip větví a mezí s principem sečné nadroviny (*cutting plane*), jež tvoří princip větvení a řezů (*branch & cut*). Právě princip větví a řezů využívá Concorde, který je jedním z nejlepších exaktních řešičů [80]. Metody patřící do kategorie konstrukčních heuristik postupně tvoří a optimalizují hamiltonovský cyklus dle dané heuristiky. Mezi takové heuristiky patří například heuristika nejbližšího souseda (*nearest neighbour*) nebo heuristika vkládání (*insertition heuristic*). Do této kategorie patří i zlepšovací heuristiky (*improving heuristic*), jež jsou založeny na lokálním prohledávání. Sem patří metoda *2-opt* navržená G. A. Croesem v roce 1958, která se snaží na zvolené permutaci měst vylepšovat délku trasy tak, že odstraní dvě hrany a opětovně je spojí v jiném pořadí, nejlépe tak, aby vznikl kratší cyklus. Další metodou je *k-opt*, jde o zobecnění metody *2-opt* v rámci, kde se mění *k* hran, nejznámějším zástupcem je varianta *3-opt*. Podrobně se těmto metodám věnuje článek [67]. Zástupcem ve skupině aproximačních metod je Christofidesův algoritmus z roku 1976, který kombinuje metody hledání minimální kostry grafu a nejlevnějšího perfektního párování. Více o aproximačních metodách je uvedeno zde [79]. Do poslední kategorie stochastických metod patří horolezecký algoritmus, simulované žíhání nebo mravenčí metody, kterým se věnuje třetí kapitola této práce.

4 PROBLÉM KANADSKÉHO CESTUJÍCÍHO (CTP)

Podobným problémem jako je problém obchodního cestujícího (TSP) je problém kanadského cestujícího (*Canadian Traveller Problem – CTP*). Jedná se o optimalizační problém, jehož cílem je najít nejvhodnější cestu mezi dvěma městy při neurčitých podmínkách. Tento problém vychází z reálné situace, se kterou se setkávají řidiči v Kanadě. Velká část Kanady je díky své poloze pokryta po několik měsíců v roce sněhem a na mnoha jejich místech hrozí časté sněhové bouře, kdy napadne i několik metrů sněhu a teploty padají pod -30 °C. Z těchto důvodů dochází přes zimu velmi často k situacím, kdy dojde k pokrytí úseku silnice sněhem, ať už ze sněhové bouře či laviny. Při velikosti Kanady a řídké hustotě její silniční sítě pak nastává situace, kdy řidiči nemají kompletní informace o stavu silnic. Ty mohou získat až v průběhu jízdy a musí se rozhodnout, zda budou čekat na zprovoznění dané silnice nebo se vydají jinou trasou do cíle. Samozřejmě uvedená situace se neváže pouze na Kanadu, ale s její obecnou podobou se můžeme setkat na celé Zemi. Řešení tohoto problému má využití všude tam, kde dochází k podobným situacím, ať už jde o navigaci v dopravě nebo navigaci robotu.

Problém kanadského cestujícího byl poprvé popsán Papadimitriouem a Yannakakisem [81] v roce 1991 v článku *„Shortest paths without a map"*. V tomto článku definovali právě obecnou podobu tohoto problému, kdy pracujeme s dynamicky se měnícím prostředím a nemáme tak kompletní informace o stavu tohoto prostředí. Na tuto jejich práci navazují autoři Bar-Noy a Schieber [82] článkem *„The Canadian Traveller Problem"*, ve kterém jsou definovány hlavní varianty tohoto problému. Můžeme je rozdělit do dvou hlavních kategorií, a to se známou pravděpodobnosti blokací a neznámou pravděpodobností blokací. V následující části jsou pak obě kategorie spolu s jejich podkategoriemi popsány.

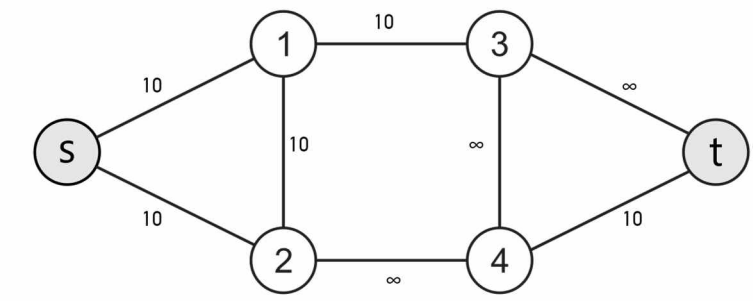


Obr. 20. Typy problému kanadského cestujícího

4.1 CTP s neznámou pravděpodobností blokací

Nejdříve si uvedeme případ, kdy dopředu neznáme pravděpodobnosti blokací. Jedná se o situaci, kterou popsali právě Papadimitriou s Yannakakis. V tomto případě zjistíme, zda je daná hrana průjezdná až ve vrcholu, který je s touto hranou incidenční. Formálně jej můžeme definovat následujícím způsobem.

Mějme neorientovaný graf $G = (V, E)$ s dopředu vybranými dvěma vrcholy $s, t \in V$ pro startovní a cílový vrchol. Každá hrana $e \in E$ má přiřazenu funkci $w: E \rightarrow \mathbb{R}_{\geq 0}$, která určuje její ohodnocení. Tato funkce pro podmnožinu $E' \subseteq E$ zablokovaných hran vrací ohodnocení s hodnotou hodnotu ∞. Jakoukoliv hranu $e \in E$ lze projít až po zjištění stavu o její průchodnosti.



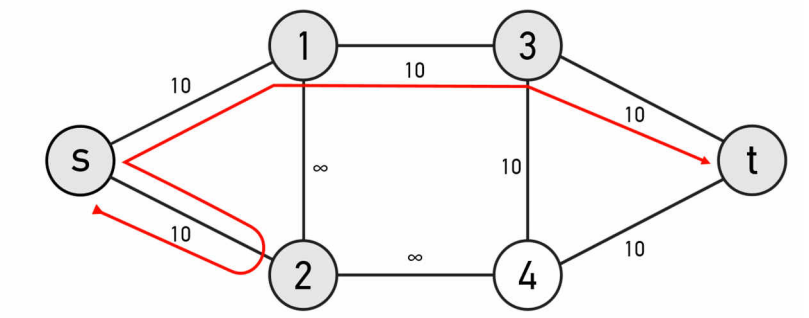
Obr. 21. Graf problému kanadského cestujícího s viditelným stavem hran

Jak již bylo uvedeno, stav hrany zjistíme až ve vrcholu, se kterým je spojena. Zde ještě rozdělujeme problém dle toho, zda zablokované hrany zůstávají zablokované, nebo dochází po určitém čase k jejich odblokování. Papadimitriou a Yannakakis pracují s tím, že v průběhu průchodu agenta se již stav hrany nemění. Dále je předpokládáno, že je graf do cíle průchozí i s rozmístěnými blokacemi. Cílem agenta je nalézt optimální strategii tak, aby výsledná cesta byla nejkratší. Ve svém článku tuto úlohu formulují jako hru dvou hráčů. První hráč je agent procházející prostředím a druhý hráč nastavuje podmínky v tomto prostředí. Tím ovlivňuje umisťování blokací na hrany grafu. Cílem prvního hráče je minimalizovat poměr mezi nejkratší cestou a nalezenou cestou s překážkami. Protihráčovým cílem je tento poměr maximalizovat. Autoři se snažili navrhnout vhodnou cestovní strategii bez znalosti budoucích zablokovaných hran. Podařilo se jim dokázat, že navrhnutí takového online algoritmu, jenž má ohraničený kompetitivní poměr, je PSPACE kompletní úloha [81]. Tuto variantu kanadského cestujícího rozvádí Bar-Noy a Schieber [82] zavedením omezení počtu blokovaných hran. Maximum blokovatelných hran je dáno parametrem k a tato varianta problému je označována jako *k-CTP*. Problém kanadského cestujícího, jak jej definovali Papadimitriou a Yannakakis, tak můžeme brát jako speciální případ *k-CTP,* kdy parametr $k = |E|$, tedy je roven počtu hran v grafu. Autoři uvádí, že nejčastěji je hodnota k daleko menší než počet hran. To zdůvodňují tím, že v reálné situaci je průměrně zablokován jen malý počet silnic a také se odkazují na podobné zavedení parametrizace u problému byzantských generálů [82]. Podobně jako autoři předchozího článku dokazují, že navrhnutí online algoritmu pro řešení *k-CTP* je PSPACE kompletní úloha. Oproti Papadimitriouovi a Yannakakisovi však nepracují s kompetitivním poměrem, ale navrhují strategii, jež zaručuje nejkratší *worst-case* délku cesty mezi dvěma vrcholy grafu. Kvalitu online algoritmů poměřujeme dvěma způsoby. Prvním je *kompetitivní poměr* algoritmu a druhým je *worst-case výkon* algoritmu.

Kompetitivní poměr je poměr mezi výkoností *online* algoritmu k výkonu *offline* algoritmu na stejné instanci problému.[1]

K problému *k-CTP* autoři definovali také jeho duální podobu, tu označují *k*-Vital Edges Problem (*k*-VEP) spolu s důkazem o jeho NP složitosti. Stejně jako u *k*-CTP máme ohodnocený graf $G = (V, E)$ s vybraným startovním vrcholem s a cílovým vrcholem t a spolu s těmito vrcholy máme definován parametr k. Cílem této úlohy je najít k hran, jejichž odstranění z grafu G maximalizuje zvětšení délky nejkratší cesty mezi startovním a cílovým vrcholem.

Na předchozí dvě práce navazují v roce 2008 nejprve Stephan Westphal s článkem *„A note on the k-Canadian Traveller Problem"* a jen o něco později Yinfeng Xu a kolektiv s článkem *„The canadian traveller problem and its competitive analysis"*. Stephan Westphal ve svém článku [83] dokazuje, že pro *k-CTP* neexistuje žádný deterministický algoritmus s menším kompetitivním poměrem než $2k + 1$. Dále zde navrhuje adaptivní online algoritmus pro tuto variantu s poměrem rovným $2k + 1$. Pojmenovává ho jako *Backtrack*, později též označovaný jako *Reposition*. V jeho případě se cestovatel (agent) vydává po nejkratší nalezené cestě (například pomocí Dijkstrova algoritmu) bez blokací. Pokud je na této trase zjištěna blokace, cestovatel se se vrací do startovního vrcholu. Ve startovním vrcholu je opět nalezena nejkratší cesta, tentokrát již bez zablokované hrany. Následně se opakuje předchozí proces, dokud cestovatel nedorazí do cíle.



Obr. 22. Průběh algoritmu Backtrack

Yinfeng Xu s kolektivem ještě ve stejném roce doplňují Westphalovu práci o další dvě strategie, *Greedy a Comparison* [84]. Algoritmus *Greedy* [2] [3] je podobný algoritmu *Backtrack*, stejným způsobem je nalezena nejkratší cesta v grafu. Po této nejkratší cestě se vydává cestující a v případě, že narazí na zablokovanou hranu, se nevrací na startovní vrchol s jako u *Backtrack* algoritmu, ale hledá novou nejkratší cestu z vrcholu, ve kterém se právě nachází. Cestující se pak vydává po této nově nalezené cestě. Tento postup se opakuje do té doby, než dorazí do cíle. Tato strategie má kompetitivní poměr $2^{k+1} - 1$. Druhý algoritmus nese jméno *Comparison* a kombinuje předchozí dvě strategie. Opět je nalezena nejkratší cesta v grafu bez ohledu na blokované vrcholy. Po této trase se vydává cestující a v případě, že narazí na zablokovanou hranu, dojde k rozhodovací situaci. Cestující je nucen si vybrat mezi Backtrack a Greedy strategií dle následující podmínky o tom, jak bude postupovat:

$$C_{BT}(s, t|E_i) < C_{GR}(s_i, t|E_i), \tag{4.1}$$

[1] Offline algoritmus zná dopředu podobu grafu na rozdíl od online algoritmu, který ji zjišťuje až v průběhu jeho procházení.
[2] Autoři jej označují i jako Myopic – krátkozraký.
[3] Sahin jej v [85] označuje jako Optimism – OMT algoritmus.

kde $C_{BT}(s,t|E_i)$ je délka cesty ze startovního vrcholu s do cílového vrcholu t po odstranění zablokovaných hran E_i z grafu G pomocí strategie Backtrack[4] a $C_{GR}(s_i,t|E_i)$ pro Greedy strategii. Hodnota s_i odpovídá aktuálnímu vrcholu v němž byla zjištěna blokace hrany. Je-li tedy splněna podmínka (4.1) volí se strategie Backtrack, v opačném případě se volí Greedy. Po vybrání strategie se vydává cestující po nově zvolené trase a celý proces se tak opakuje, dokud nedorazí do cíle.

Další přístup k řešení CTP uvádí autoři Hrdina a Matoušek v článku *„Canadian traveller problem: A note about Game Theory approach"* [86] z roku 2014. Zde je uveden způsob pomocí teorie her. Navazují tak na Papadimitrioua a Yannakakise kteří, jak je uvedeno v předchozí části, popisují CTP jako hru dvou hráčů. Tento přístup pak porovnávají s řešením pomocí mravenčích algoritmů. Ve výsledku jsou hodnoty získané z obou rozdílných metod dle autorů velmi podobné. Poslední strategie byla publikována v článku „*Canadians Should Travel Randomly*" [87, 88] a nese název *Greedy & Reposition Randomized Algorithm* neboli GRR. Zde se nejprve vybere množina nejkratších cest v grafu, která je dána omezující podmínkou. Tato množina cest je následně reprezentována pomocí stromové struktury označované jako *apex tree*. V té pak agent vybere náhodně cestu a tu prochází, dokud nenarazí na zablokovanou hranu. Načež se vybere další cesta z množiny, kterou bude agent procházet. Tento postup se opakuje do té doby, než se podaří agentovi dorazit do cíle. GRR strategie má kompetitivní poměr $\left(1+\frac{\sqrt{2}}{2}\right)k+1$ v pseudo-polynomiálním čase. V průběhu let vyšlo několik článků, které řeší *k-CTP* z pohledu speciálních případů grafů. Například v roce 2013 Marco Bender a Stephen Westphal publikovali článek *„An optimal randomized online algorithm for the k-Canadian Traveller Problem on node–disjoint paths"* [89]. Řeší zde variantu grafu s vrcholově nesouhlasnými cestami pro *k*-CTP. V článku navazují na Westphalovu předchozí práci a navrhují náhodný online algoritmus vycházející z *Backtrack* algoritmu. Zde je nejprve vybráno $k+1$ nejkratších uzlově nesouhlasných cest v grafu mezi vrcholy *s-t*. Pro každou z těchto cest je spočtena pravděpodobnost zablokování. Dle toho se vybere nejvhodnější cesta pro cestujícího. V případě, že dojde k zablokování, se cestující vrací na start a pokračuje další cestou v pořadí. Tento algoritmus má kompetitivní poměr $2k+1$. V roce 2015 Chan a kolektiv [90] navazují na Bendera s Westphalem a studují variantu, kde všechny hrany v grafu mají stejné ohodnocení *k*-EWCTP. Ve svém článku pak dokazují, že neexistuje deterministický online algoritmus s kompetitivním poměrem menším jak $2k+1$ a pro náhodný online algoritmus je pak poměr nejméně $k+1$. Dalším případem je zajímavé použití více agentů a vzájemné komunikace mezi nimi. Ten je uveden v práci „*The k-Canadian Traveller Problem with communication*" [91]. Autoři zde pracují se dvěma úrovněmi komunikace, plnou a omezenou. Pro otestování obou způsobů komunikace byla navržena *Retrace-Alternating* strategie (RAS) zasazená do městského prostředí. Z výsledků těchto testů je patrné, že nižší spodní hranici konkurenčního poměru dosahuje větší zastoupení agentů s úplnou komunikací. Avšak větší úroveň zastoupení úplné komunikace nemusí vždy vést k výraznému zlepšení konkurenčního poměru oproti omezené komunikaci, jelikož větší vliv na něj má samotná struktura sítě.

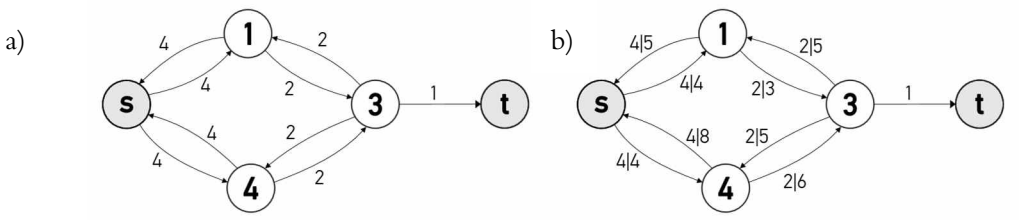
Podobně, jako autoři předchozího článku, použili Liao a Huang ve svém článku "*The Covering Canadian Traveller Problem*" [92] dva algoritmy *Greedy* a *Backtrack/Reposition* pro speciální případ dvojitě ohodnoceného grafu. V článku popisují kombinaci těchto dvou algoritmů do jednoho, pojmenovaného jako GR, *Greedy-Reposition*. V tomto článku pak také kombinují problém kanadského cestujícího s problémem obchodního cestujícího. Takto vzniklý problém označují pak jako *Covering CTP* a pro jeho řešení navrhují algoritmus pojmenovaný *Cyclic routing* (CR).

[4] Autoři ve svém článku označují Backtrack algoritmus jako Reposition.

Doposud uvedené metody řeší problém kanadského cestujícího ve variantě *k-CTP* za předpokladu, že vznikne-li blokace na hraně grafu, je hrana trvale zablokována. V následující části je popsán případ, kdy blokace hrany není trvalá.

4.2 CTP s neznámou pravděpodobností blokací a obnovitelnými hranami

Druhou variantou CTP s neznámou pravděpodobností blokací je případ, kdy zablokované hrany jsou po určité době opět průchozí. Poprvé ji popsali autoři Bar-Noy a Schieber [82], kteří ji pojmenovali *Recoverable-CTP*. Zde má každý vrchol definovánu hodnotu penalizace či ceny obnovení. Pod penalizací si můžeme představit například dobu potřebnou ke znovuotevření blokované hrany. V případě jejich článku je hodnota penalizace stejná pro všechny zablokované hrany, jež z daného vrcholu vycházejí. Stejně jako v předchozí části můžeme tuto variantu CTP rozdělit dle omezení maximálního počtu blokací, ta je označována jako *k*-RCTP. Bar-Noy a Schieber navrhli strategii právě pro případ *k*-RCTP, kdy jsou penalizace menší než délky hran. Tato strategie zaručuje nejkratší *worst-case* délku cesty mezi cílovým vrcholem a ostatními vrcholy grafu.



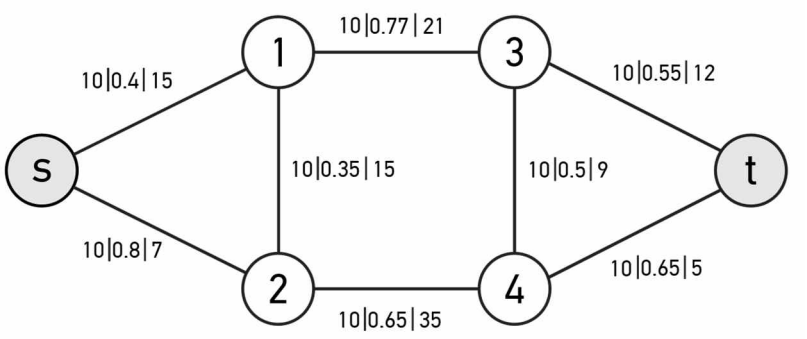
Obr. 23. a) Recoverable-CTP graf s penalizačními časy ve vrcholech,
b) graf s ohodnocením a penalizačními časy na hranách

O pár let později došlo k úpravě, kdy již není hodnota penalizace vázána k danému vrcholu, ale každá hrana má vlastní hodnotu penalizace spolu s ohodnocením vzdálenosti. V této práci se pracuje pouze s touto úpravou *Recoverable-CTP*. Liao a Huang [88] označují graf jako dvojitě ohodnocený (*Double valued graph*), kde hrana $e \in E$ má hodnotu ohodnocení či vzdálenosti značenou $w(e)$ a hodnotu penalizace $r(e)$. S touto variantou *k*-RCTP pak pracují následující články [88, 93, 94], kde jejich autoři navrhují několik strategií pro její řešení. V prvním článku jsou popsány algoritmy *Waiting* a *Greedy*. U *Waiting* algoritmu se vydává cestující-agent po nejkratší nalezené cestě, která byla nalezena v grafu bez započtení blokací. Stejně jako u *k*-CTP je pro nalezení nejkratší cesty použit *Dijkstrův* algoritmus. V případě, že cestující v průběhu najde zablokovanou hranu, setrvá na místě a počká na odstranění překážky. Poté pokračuje přes hranu po výše zmíněné nejkratší cestě až do cíle. Cestující u algoritmu *Greedy* se stejně jako u *Waiting* algoritmu vydává po nejkratší nalezené cestě. Pokud narazí na zablokovanou hranu, má dvě možnosti, buď vyčká na odstranění blokace a pokračuje po cestě dále, nebo najde novou nejkratší cestu z vrcholu, ve kterém se nachází a vydává se po této nově nalezené cestě. Z těchto dvou možností si cestující vybírá vhodnější. Ve třetím z uvedených článků [94] je popsán případ speciálního grafu, kde vrcholy jsou uspořádány v sérii a mezi sousedními vrcholy se vyskytuje několik paralelních hran. Strategie autorů Su a Bing je pojmenována *Comparison* a jedná se o upravenou verzi jejich stejnojmenné strategie pro řešení *k*-CTP. Zde agent po zjištění zablokovaní hrany určí horní mez čekání a porovná ji s hodnotou penalizace. Pokud je penalizace menší nebo rovna horní mezi, agent vyčká na odblokování hrany. V opačném případě agent hledá novou nejkratší cestu z daného vrcholu do cíle. Poslední strategií z této trojce je *Greedy-Reposition* (GR), již navrhli Liao a Huang ve svém článku „*Generalized Canadian traveller problem*“ [95].

V něm jsou kombinovány dvě strategie *Greedy* a *Reposition*[5], přičemž strategie *Greedy* je použita opakovaně. V prvním případě *Greedy* nebere v potaz blokaci hrany a v druhém s touto blokací již počítá. Tyto strategie tvoří sub-strategie a dle daných podmínek se pak vybírá ta nejvhodnější pro průchod agenta.

4.3 CTP se známou pravděpodobností blokací

Druhou částí již uvedeného rozdělení problémů CTP je situace, kdy dopředu známe jak graf G, tak i pravděpodobnost blokací v prohledávaném grafu. Tento typ je označován jako stochastická varianta problému kanadského cestujícího (SCTP). Formální definice je oproti předchozí verzi rozšířená o hodnotu pravděpodobnosti zablokování $p(e)$ pro každou hranu $e \in E$. Stejně jako u CTP s neznámou pravděpodobností blokace se stochastická verze CTP dále dělí na dvě podskupiny. První, kdy jsou zablokované hrany permanentně uzavřeny, tj. SCTP a *k*-SCTP, a druhou, kdy dojde k jejich odblokování, a tedy obnovitelnou variantu SRCTP. Na následujícím obrázku je ukázka takovéhoto grafu pro úlohu SRCTP, kdy každá hrana má jak své ohodnocení, tak i pravděpodobnost blokace a penalizaci.



Obr. 24. SRCTP graf, hrana je definována:
ohodnocením | pravděpodobností blokace | hodnotou penalizace

V roce 2009 vyšel článek *„High-Quality Polices for the Canadian Traveler's Problem"* [96], ve kterém bylo publikováno několik algoritmů pro řešení SCTP a *k*-SCTP. Pro nastavení základní hodnoty pro porovnání používají autoři jednoduchou strategii *„Optimism"*. Jedná se o stejnou strategii jako je již dříve popisovaná *Greedy*. Pomineme-li tuto strategii, je první autory z navržených strategií *Hindsight optimization* (HOP). Ta spolu se strategií Optimistic Rollout (ORO) řeší SCTP způsobem, kdy se pro výběr vhodné cesty použije řada deterministických řešení. Přesněji agent při průchodu grafem, nacházející se v aktuálním vrcholu, si pro své možné následníky vygeneruje množství deterministických instancí grafu. Zde jsou dle daných pravděpodobností již pevně určeny stavy hran, tj. zda jsou průchozí, či ne. Tyto vygenerované instance označují autoři jako „*rollouts*"[6]. Na těchto vygenerovaných instancích se poté hledá nejkratší cesta z daného vrcholu do cíle (SPP) a z úspěšných průchodů se vypočte průměrná odhadovaná délka cesty. Agent poté pokračuje po takové hraně, která má minimální odhadovanou průměrnou délku cesty. Množství rolloutů je dáno parametrem N a autory používaná hodnota je 10000, která je kompromisem mezi výpočtovou náročností a kvalitou výsledků. Čím větší je počet takovýchto instancí, tím je výpočet náročnější. Další nevýhodou velkého počtu rolloutů je častá konvergence k suboptimálnímu řešení.

[5] Jde o jiné pojmenování strategie *Backtrack*.

[6] Termín „rollout" nemá český ekvivalent, proto je ponecháno jeho anglické znění.

Proto byla navržena strategie *Optimistic Rollout* (ORO), která modifikuje Hindsight strategii. Její úprava spočívá v prohledávání rolloutů pomocí *„Optimism"* strategie, ze které je jako u HOP spočtena průměrná odhadovaná délka cesty. Po výpočtu odhadovaných délek cest u rolloutů se agent vydává po cestě s nejmenším ohodnocením.

U dalších dvou strategií využívají autoři jako základ pro řešení algoritmus UCT neboli „*Upper Confidence bounds applied to Trees*". Tento algoritmus byl publikovaný v článku „*Bandit based Monte-Carlo planning*" [97] autorů Kocsise a Aksakalliho. Strategie pro CTP založené na UCT algoritmu využívají také rolloutů, avšak oproti předchozím strategiím HOP a ORO jsou tyto rollouty na sobě závislé.

Pro dosažení vzájemné závislosti rolloutů je použita stromová struktura, jak již ostatně napovídá sám název UCT. Kořen takovéhoto stromu tvoří aktuální pozice agenta v grafu a rollouty jsou podstromy. Každý vrchol v tomto stromu obsahuje informace jak o vrcholu grafu, tak i doplňující hodnoty, jako je například celkový počet návštěv vrcholu či suma odhadovaných cen.

Výběr následujícího vrcholu v grafu lze rozdělit do několika kroků, které se opakují, dokud nejsou splněny ukončující podmínky, jako je například předem daný počet opakování N nebo limit výpočetního času t_{dec}. V prvním kroku se pro každou pozici agenta, která není cílová, nejprve vytvoří instance grafu. Jde o vygenerování podoby grafu, kde se určí stavy hran dle daných pravděpodobností, tj. zda jsou průchozí nebo zablokované.

Druhým krokem je vytvoření sekvence stavů. Zde se pro vygenerovanou instanci začíná tvořit sekvence stavů, tedy rollout. Výběr stavů je postaven na maximalizaci UCT rovnice, která je velmi podrobně popsána zde [98]. Pro každý vybraný vrchol se provede expanze, při níž je vrchol spolu s jeho potomky přidán do UCT stromu. Tento výběr se opakuje do té doby, než je nalezen cílový stav v rolloutu nebo poslední stav v sekvenci, kterého bylo dosaženo, pokud již není možné dosáhnout cílového vrcholu.

Třetím krokem je výpočet očekávané ceny. Sečtou se délky hran pro danou sekvenci. Pokud dojde k situaci, že poslední stav v sekvenci není cílový vrchol, je zbytek cesty dopočítán pomocí OMT strategie. Posledním krokem v iteraci je zpětná propagace, kdy dochází k aktualizaci hodnot stavů z rolloutu v UCT stromu, které jsou následně využity v nové iteraci. V okamžiku ukončení hledání následníka je vybrán nejslibnější potomek kořene z UCT stromu a ten je nastaven jako následník aktuálního vrcholu agenta.

První strategií založenou na algoritmu UCT je *Blind UCT* (UCTB). Tato strategie odpovídá obecnému popisu, jenž jsme si zde uvedli. Její pojmenování Blind, tedy slepá, vychází z toho, že nevyužívá žádných specifických informací o CTP problému. Proto tato strategie potřebuje často velký počet iterací k tomu, aby dosahovala dostatečných výsledků.

Druhou strategií je *Optimistic UCT* (UCTO), která rozšiřuje UCTB o několik úprav. Nejdůležitější úpravou je využití OMT strategie pro ohodnocení délky u nenavštívených vrcholů během rolloutů. Cílem UCTO je směřovat první rollouty ke slibným částem stavového prostoru. Sebastián Filip ve své diplomové práci [98] navrhuje další varianty založené na UCT algoritmu, jedná se o UCTO2 a UCTP – Pruning UCT. Základem těchto variant je úprava UCT algoritmu, ve které se odhadují ceny všech potomků podobným způsobem jako ve strategiích HOP a ORO. Nejprve tak dochází k prohledávání do šířky, a až následně jsou slibné vrcholy expandovány a prohledávány do hloubky. Filip tuto variantu pojmenovává UCT s rozvojem do šířky a strategie UCTO2 je pak implementací UCTO strategie pro ni. Druhou variantou je Pruning UCT, která zavádí prořezávání neperspektivních větví a jejím cílem je dosažení rychlejší konvergence ke kvalitní strategii.

Dále se v roce 2008 Nikolova a Karger [99, 100] ve svých dvou článcích věnují řešení pro orientovaný acyklický graf (DAG) a typ grafu, jenž obsahuje vrcholově nesouhlasné cesty[7] (DPG) mezi počátečním vrcholem s a cílovým vrcholem t. Pro řešení jsou zde využity Markovské rozhodovací procesy a pro oba problémy jsou navrženy strategie řešící CTP v polynomiálním čase.

U orientovaných acyklických grafů je výhodou nemožnost vrátit cestujícího do předchozího vrcholu, čímž je zmenšen stavový prostor. Dojde tak ke zjištění délky ze startovního vrcholu do cílového vrcholu jako u Backtrack strategie. A v druhém případě je zjištěna délka z aktuálního vrcholu opět do cílového vrcholu. V obou případech je nová cesta vyhledána po odstranění všech zablokovaných hran z grafu. Následně dojde k porovnání výsledků a je vybrána strategie s nižší délkou cesty, po které pokračuje náš cestující.

4.4 Speciální případy CTP

Existují však i speciální případy CTP, které nejdou zahrnout do uvedených kategorií. Mezi takové patří například: CTP s opakováním (CTP-rep). V této variantě prochází grafem postupně n agentů. Průchod agenta začíná v momentě, kdy první agent dorazí do cíle. Agent, který dokončil svou cestu, může informovat následujícího o stavech hran v okamžiku jeho průchodu. V článku *„Repeated-task Canadian Traveler Problem“* [101] byla uvedena strategie, jež minimalizuje souhrnnou cenu cest agentů pomocí úpravy UCT na speciálních grafech DPG. CTP se vzdáleným snímáním (CTP with Remote sensing) je varianta, kdy agent má schopnost dopředu zjistit stav hrany. Tato schopnost má však svoji cenu. Cílem u této úlohy je minimalizovat cenu cesty včetně ceny za využití schopnosti zjišťující stav hran. Více informací o této úloze lze najít v článku *„Canadian Traveler Problem with Remote Sensing“* [102].

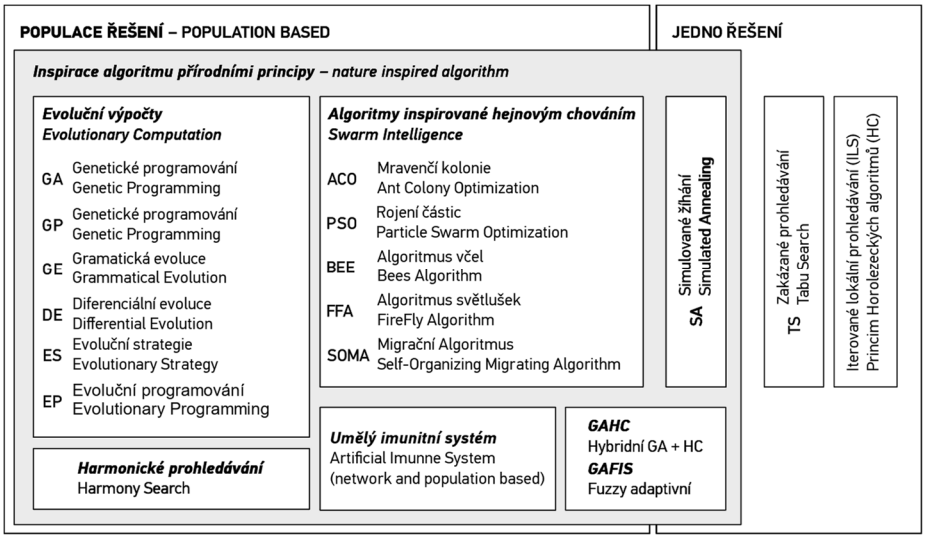
[7] V anglické terminologii *„node–disjoint paths“*.

5 ANT COLONY OPTIMIZATION

V předchozích kapitolách byly popsány dva náročné optimalizační problémy, a to úloha obchodního cestujícího a problém kanadského cestujícího. Právě na prvním z těchto problémů byly počátkem devadesátých let poprvé použity algoritmy inspirované chováním reálných mravenců. Obecně se takové algoritmy označují názvem Ant Colony Optimization – ACO, tedy optimalizace pomocí mravenčích kolonií. ACO spolu s ostatními optimalizačními technikami, jež se inspirují kolektivně žijícími živočichy, řadíme do kategorie Rojových metod. V této práci jsou použity právě tyto algoritmy pro řešení druhého z uvedených optimalizačních problémů, tedy problému kanadského cestujícího. Z tohoto důvodu se věnuje tato kapitola přehledu optimalizace pomocí mravenčích kolonií.

5.1 Swarm intelligence – Rojová inteligence

Swarm intelligence, neboli rojová inteligence, je součástí metod umělé inteligence. Využívá studia chování kolektivně žijících živočichů pro řešení široké skupiny úloh, od optimalizačních úloh, úloh řízení až po robotiku, kdy roboty se chovají jako členové roje. Mezi takto žijící živočichy patří například mravenci, včely, ptáci, ryby a mnoho dalších. Jejich chování se z makroskopického pohledu zdá jako inteligentní. Budeme-li ale sledovat chování jednotlivých jedinců z mikroskopického úhlu pohledu, bude jejich chování vypadat často podivně a chaoticky. Je to dáno tím, že řízení roje není centralizované, ale je založeno na samo-organizaci. Chování roje vychází z lokální komunikace jedinců. A chování jedince je založeno na vrozených jednoduchých pravidlech, kterými se řídí [103].



Obr. 25. Rozdělení metaheuristik, vytvořeno dle [104]

Do kategorie metod patřících do rojové inteligence mimo mravenčích algoritmů patří i například známá metoda Particle Swarm Optimization – PSO neboli metoda rojení částic, včelí algoritmy, algoritmus světlušek a mnohé další. PSO je metoda, jež se inspiruje chováním hejna ptáků či hejna ryb. Autory Kennedyho a Eberharta zaujala schopnost těchto živočichů zůstat v hejnu v případě ptáků a v hejnu ryb bez toho, aby je vedl nějaký jasný vůdce [53]. Výsledky svého studia publikovali v roce 1995, kde ukázali, že pomocí simulací využívajících celulární automaty lze dosáhnout takovéhoto globálního chování pomocí několika pravidel použitých v lokální interakci jednotlivých jedinců [53]. V případě včelích algoritmů lze inspiraci rozdělit do dvou skupin, do první patří metody inspirované sháněním potravy, například algoritmus Bee Colony Optimization – BCO a Artificial Bee Colony Algorithm – ABC [57, 105]. Ve druhé

skupině metody napodobují chování včel při páření, viz algoritmus Honeybee Mating Optimization Algorithm – HBMO [106]. Optimalizace hejnem světlušek neboli Glowworm Swarm Optimization – GSO je další populární metoda a byla publikována v roce 2005 [107]. Skupina metod rojové inteligence je bohatá na množství přístupů, jimiž se inspiruje. Mimo již zmíněných metod se můžeme setkat s metodami inspirovanými nejen chováním hmyzu, ptáků, ryb a dalších druhů živočichů [108–111]. Přehledu a popisu nejznámějších metod rojové inteligence je věnována kniha [112] a vybraným novým metodám knihy [113, 114] a dále přehledové články novějších metod [115, 116].

5.2 Bio-inspirace

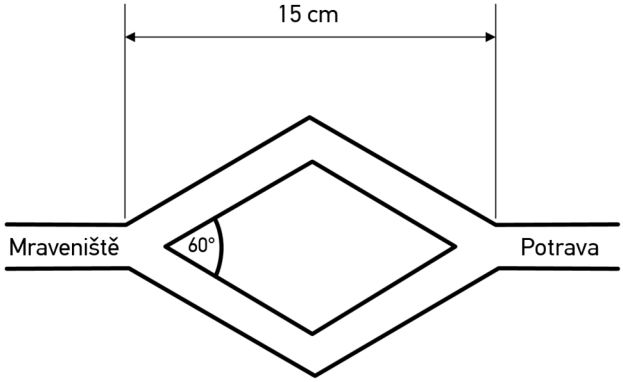
Mravenci jsou sociálně žijící hmyz, který tvoří kolonie o různých velikostech. Dle druhů, kterých je dosud popsáno více než 13 tisíc, mají kolonie velikost od sotva několika jedinců žijících pod kamenem až po stovky milionů žijících v decentralizovaných mraveništích tvořících tzv. superkolonie. Pro přežití jakkoli veliké kolonie mravenců, ale i jakýchkoli v roji žijících živočichů je nutná nějaká forma komunikace. V rojové inteligenci rozlišujeme tři základní typy komunikace, nepřímou, přímou interaktivní a přímou komunikaci. Nejrozšířenějším způsobem komunikace u mravenců je nepřímý typ označovaný jako stigmergie. Jedná se o způsob komunikace, kdy mravenci zanechávají v prostoru feromonovou stopu a ovlivňují tak okolo procházející mravence. Tento způsob komunikace je využíván i v mravenčích algoritmech, proto si ho v následující části popíšeme podrobněji. S přímou interaktivní komunikací se lze setkat u některých druhů mravenců, jako je například rod *Oecophylla Ionginoda*. Pokud se setkají dva mravenci tohoto druhu, dojde k výměně informací opět v podobě feromonových stop dotykem tykadel obou mravenců. Jako příklad posledního způsobu komunikace lze uvést včely. Přímá komunikace je zde je řešena *„včelím tanečkem“,* který je předváděn ostatním včelám [117]. Stigmergie je označení, které poprvé použil francouzský entomolog Pierre-Paul Grassé ve svém článku [118] z roku 1959. Bylo to vyvrcholení jeho více něž dvacetiletého výzkumu chování termitů. Grassé popisuje typ stigmergie, jež je vlastní právě termitům rodu *Bellicositermes natalensis*. Ti zpočátku roznáší hrudky bahna po termitišti nahodile. V případě, že mají ve svém dosahu umístěnou hrudku bahna, vzniká v nich přirozená potřeba ji rozšiřovat. Jedná se o jejich jedinou formu komunikace a postupně tak dochází k rozšiřování či opravě termitiště. Chování termitů lze shrnout do podmíněného chování, vidí-li termití dělník rozpracovanou práci, je stimulován v ní pokračovat.

Další podobu stigmergie lze uvést na případě mravenců [119]. Mnoho druhů používá pro komunikaci tzv. feromonovou stopu, kterou zanechávají za sebou například při návratu od zdroje potravy zpět do mraveniště. Feromonová stopa stimuluje ostatní mravence v její blízkosti k jejímu následování. Pokud feromonovou stopu následují a narazí na potravu, vydávají se s ní nazpět do mraveniště. Při cestě zpět již položenou feromonovou stopu posilují. Čím kratší trasa, tím rychleji se může mravenec vrátit k potravě a zpět do mraveniště. Postupně dochází k nalezení co nejkratší trasy k potravě. Tento způsob používá například u nás velmi rozšířený mravenec obecný (*Lasius Niger*), nebo silně invazivní druhy jako mravenec faraon (*Monomorium pharaonis*) a mravenec argentinský (*Linepithema humile* dříve označovaný jako *Iridomyrmex humilis*). Právě mravenec argentinský (*Linepithema humile*) je nejen invazivní, ale i velmi agresivní druh, který vytlačuje všechny ostatní druhy žijící v jeho okolí. Z tohoto důvodu patří k jedněm z nejvíce studovaných druhů mravenců a stali se inspirací pro mravenčí algoritmy. Poprvé byl popsán v roce 1866 v Argentině rakouským entomologem Gustavem Mayrem [120].

Obr. 26. Mravenec argentinský (Linepithema humile), královna a dělnice [121]

Vyskytuje se hlavně v teplých částech všech světadílů mimo Antarktidu. Jedná se o druh, který tvoří již dříve zmiňované superkolonie, například evropská superkolonie je rozprostřena na pobřeží středozemního moře v délce několika tisíc kilometrů. V článku *„Intercontinental union of Argentine ants"* [122] uvádí autoři výsledky pokusu, kdy mravenci z evropské, kalifornské a japonské superkolonie nebyli vůči sobě agresivní. Vzájemnou neagresivitu autoři přisuzují silné genetické příbuznosti mezi zástupci těchto superkolonií a hovoří o mezikontinentální superkolonii. Vůči členům nespřízněných kolonií nebo jiným druhům byli jednotliví zástupci silně agresivní.

S koncem 80. a počátkem 90. let byly publikovány dva stěžejní články o chování mravenců argentinských při sběru potravy. V obou článcích jsou popsány experimenty s takzvaným dvojitým mostem. V prvním článku *„The self-organizing exploratory pattern of the argentine ant"* Deneubourg a kolektiv [123] vytvořili dvě arény, jednu obsahující mraveniště, a druhou se zdrojem potravy. Arény jsou propojené dvojitým mostem stejného tvaru s délkou 15 centimetrů a šířkou 1 centimetr. Podoba experimentu je zobrazena na následujícím obrázku.



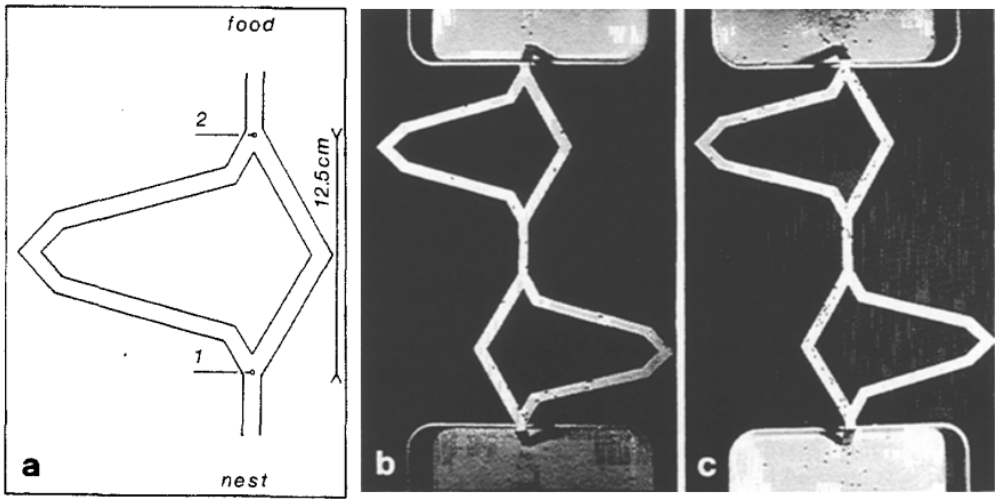
Obr. 27. Experiment „dvojitý most", vytvořeno dle [124]

Mraveniště bylo umístěno do první arény a potrava do druhé. Mravenci zpočátku chodili skrze oba mosty ve stejných počtech a feromonová stopa byla stejná na obou větvích mostu. Jakmile došlo i k mírné nerovnováze v úrovni feromonové stopy na jedné straně mostu, způsobené například zpožděním několika mravenců, došlo k preferování větve mostu se silnější feromonovou stopou většinou mravenců. To vedlo autory experimentu k hypotéze a sestavení pravděpodobnostní rovnice (5.1), dle které se mravenci rozhodují:

$$P_A = \frac{(k+A_i)^n}{(k+A_i)^n+(k+B_i)^n}, \quad (P_\mathrm{A} + P_\mathrm{B} = 1), \tag{5.1}$$

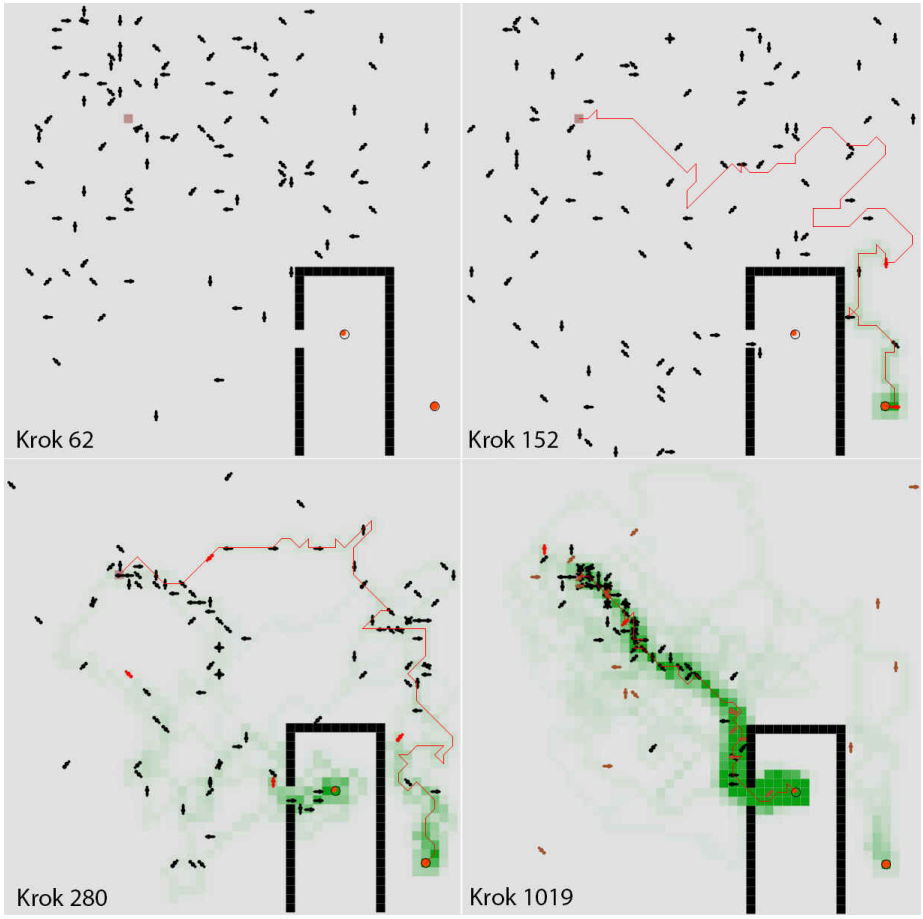
kde P_A a P_B jsou pravděpodobnosti přechodu větve A a větve B mostu, A_i a B_i jsou počty mravenců, kteří prošli danou větví. Parametry k a n jsou hodnoty vycházející z pozorování

chování mravenců. Parametr n je citlivost na rozdílné hodnoty feromonů na větvích mostu. S vyšší hodnotou n se zvyšuje šance výběru dané větve i při malém rozdílu feromonů. Hodnota k je atraktivita neoznačené větve, čím větší je hodnota k, tím je nutná větší hodnota položeného feromonu pro nenáhodný výběr větve. V této rovnici se nepracuje přímo s hodnotou feromonů, nýbrž s počtem mravenců v dané větvi. Jedním z důvodů, proč je tomu tak, je ignorování odpařování feromonové stopy. Doba nutná k jejímu odpaření je odhadována na 30 minut, přičemž k ustálení hodnot, a tedy k vybrání jedné z větví mostu, došlo mezi 10. a 25. minutou experimentu. Rovnice byla následně použita v Monte Carlo simulaci, která potvrdila výsledky z pozorování živých mravenců. Na tento experiment navazují autoři dalším článkem s názvem „Self-organized Shortcuts in the Argentine Ant" [125]. Opět je zde experiment s dvojitým mostem, jeho rozdílnou podobu lze vidět na následujícím obrázku.



Obr. 28. Experiment „dvojitý most" [125]

Jak je patrné, experiment obsahuje kratší a delší větve mostu. Mravenci se musí navíc rozhodovat i uprostřed mostu, po které z větví se vydají. Průběh experimentu byl podobný jako u prvního, ze začátku se vydávali mravenci jak po delších, tak i kratších větvích mostu ve stejném poměru. Avšak k vybrání kratších větví došlo dříve než v předchozím experimentu. Mravenci vydávající se po kratších větvích byli schopni se mnohem dříve vrátit s potravou do mraveniště a zpět k potravě. Koncentrace jejich feromonové stopy byla silnější než mravenců chodících po delší cestě. Stejně jako u předchozího experimentu byla opět provedena Monte Carlo simulace, která potvrdila chování reálných mravenců. Díky těmto experimentům se podařilo poodhalit samo-organizační mechanismus chování mravenců při sběru potravy. A právě výsledky těchto experimentů stály na počátku mravenčích algoritmů, které budou popsány v následující podkapitole.



Obr. 29. Simulace chování mravenců, vytvořeno pomocí [126]

5.3 ACO

V roce 1992 Marco Dorigo obhájil disertační práci s názvem *„Optimization, Learning and Natural Algorithms“* [127], kde popisuje nový přístup k řešení kombinatorických optimalizačních problémů. Ant Colony Optimization neboli ACO vychází z pozorování mravenců a jejich chování. Dorigo ve své práci řešil tímto způsobem problém obchodního cestujícího. Ve své práci ACO formuluje jako tzv. metaheuristiku, tedy jako předpis či chceme-li základní kostru pro použití heuristické metody vhodné pro daný problém. S metaheuristikami se setkáváme u náročných optimalizačních problémů, u kterých je problematické najít optimální řešení v polynomiálním čase. Jedná se o metody, které negarantují nalezení optimálního řešení. Mezi další příklady metaheuristických metod lze uvést například simulované žíhání, včelí algoritmy a mnohé další. Pomocí ACO lze řešit širokou škálu optimalizačních úloh, od směrových, okružních a rozvozních úloh po přiřazovací či plánovací úlohy a spoustu dalších úloh.

Tab. 1 Přehled vybraných optimalizačních problémů řešených pomocí ACO

Typ problému	Jméno	Autoři	Rok	Reference
Směrovací	Traveling salesman	Dorigo et al.	1991	[128, 129]
		Dorigo, Gambardellla	1996	[130, 131]
		Stützle, Hoos	2000	[132]
	TSP s časovým oknem	López-Ibáñez et al	2010	[75]
	VRP	Gambardellla et al.	1999	[133]
		Reimann et. Al	2004	[134]
		Fountas	2005	[135]
Přiřazovací	QAP	Maniezzo et al	1999	[136]
		Stützle, Hoos	2000	[132]
	Barvení grafu	Costa, Herz	1997	[137]
Plánovací	Job scheduling	Colorni et al	1994	[138]
	Open shop	Pfahringer	1996	[139]
	Flow shop	Stützle	1998	[140]
Strojové učení	Klasifikační pravidla	Parpinelli et al	2002	[141]
		Martens et al	2007	[142]
	Bayesovské sítě	de Campos et al	2002	[143]
Routovací		Schoonderwoerd	1997	[144]
		Di Caro, Dorigo	1998	[145]
		Varela, Navarro	1999	[146]
		Okdem	2009	[147]

Ant Colony Optimization je metaheuristikou, kde kolonie umělých mravenců spolupracuje při vyhledání co nejlepšího možného řešení pro daný optimalizační problém. Stejně jako u živých mravenců se zde využívá jednoduchých agentů, jejichž způsob komunikace je založen na stigmergii. Použitím jak heuristické informace o daném problému, tak i stigmergie, jsou ACO algoritmy schopny dosáhnout kvalitního řešení v efektivním čase. Virtuální mravenci mají několik vlastností, které je od svých živých předobrazů odlišují [148]. Reální mravenci se při hledání potravy rozhodují dle intenzity feromonu ve svém okolí. Oproti tomu u virtuálních

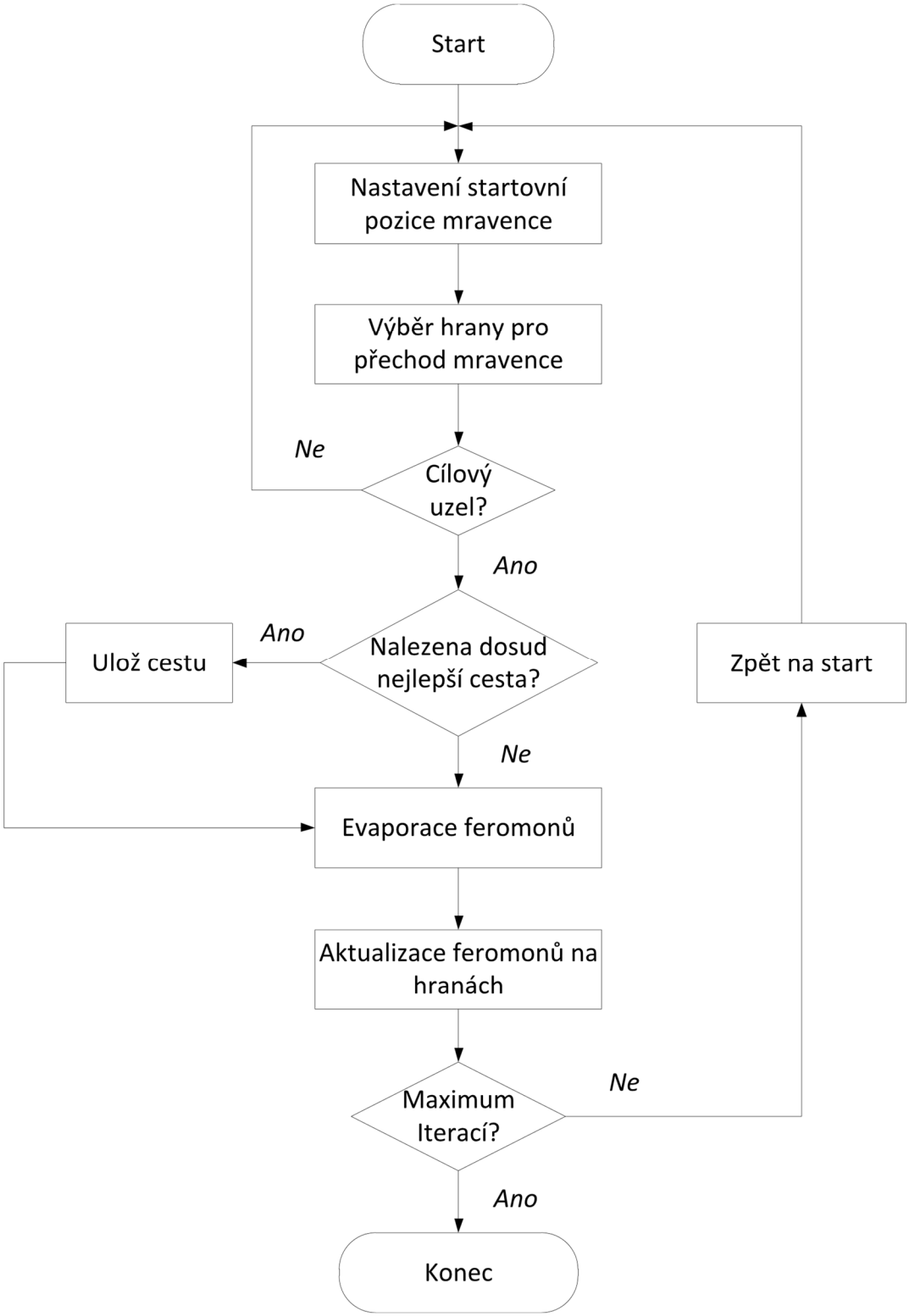
mravenců je rozhodování ovlivněno heuristickou informaci spolu s intenzitou feromonové stopy. Jednou z odlišností je způsob pokládání feromonové stopy. U virtuálních mravenců je pokládána až po dokončení průchodu grafem a často dle určitého kvalitativního ohodnocení. U reálných mravenců je feromonová stopa často pokládána jak při cestě k potravě, tak i nazpět. Dále je u virtuálních mravenců přidána paměť pro zaznamenání jejich cesty skrze graf. Cesta k potravě a nazpět do mraveniště je tak u umělých mravenců vždy stejná, naopak v případě reálných mravenců stejná být nemusí.

Pro použití mravenčí metaheuristiky musíme nejprve přeformulovat vstupní problém na problém prohledávání grafu. Načež mravenčí kolonie, které tvoří jednodušší agenti, prochází stochasticky grafem a vyhledávají řešení problému. Kolonie mravenců opakovaným hledáním postupně zpřesňuje svou představu o globálním optimálním řešení. To se děje, dokud není splněna ukončovací podmínka. Na následujícím pseudokódu je uvedena obecná podoba mravenčí metaheuristiky.

Algoritmus 5 Ant colony optimization metaheuristic

```
setParameters()
initializePheromoneTrails()
while (!terminationCondition) do
    constructAntSolutions()
    updatePheromones()
    updateStatistics()
end
```

Nejprve dochází k inicializační fázi, kdy se nastavují vstupní parametry mravenčích algoritmů spolu s nastavením základní úrovně feromonové stopy na hranách grafu. Hlavní částí metaheuristiky je cyklus, ve kterém probíhají tři části. Nejprve jsou ve funkci *constructAntSolutions* mravenci umístěni do grafu a hledají řešení. Při průchodu grafem se mravenci rozhodují dle pravděpodobnostní funkce. Její hodnota je dána jak feromonovou stopou, tak problémově danou heuristikou a případnými dalšími omezeními. Jakmile všichni mravenci dorazí do cíle, provedou se dodatečné kroky jako je kvalitativní vyhodnocení nalezených řešení. V závislosti na problému se může provést lokální prohledávání a úprava již nalezených cest z předchozího kroku. Další částí je aktualizace feromonové stopy. Nejprve je nutno provést evaporaci části feromonové stopy na všech hranách grafu. Evaporací feromonu se snažíme podpořit další prohledávání a zabránit tak uvíznutí v lokálním minimu. Načež je provedena aktualizace stopy pro hrany patřící mezi úspěšná řešení. Počet řešení, kde se bude feromonová stopa posilovat spolu s její intenzitou, je specifický pro konkrétní mravenčí algoritmus. Poslední částí metaheuristiky je funkce *updateStatistic*, zde se provádí dodatečné akce, ať už jde o aktualizaci dosud nejlepšího řešení, dalších globálních hodnot či uplynulého času. Cyklus se opakuje do doby, než jsou splněny ukončovací podmínky, například nalezení řešení, které se již nezlepšuje, vypršení časového intervalu, proběhnuvší maximální počet iterací nebo případně vyčerpání zdrojů.



Obr. 30. Vývojový diagram obecného ACO algoritmu

5.3.1 Mravenčí algoritmy

V již zmiňované práci z úvodu této podkapitoly [124, 149] Marco Dorigo popsal tři mravenčí algoritmy: ANT-density, ANT-quantity a ANT-cycle, které tvoří komplex Ant System [124, 145]. Všechny tyto algoritmy mají společnou strukturu pravděpodobnostního přechodového pravidla pro průchod grafem. Odlišuje je, jakým způsobem a kdy dochází aktualizaci feromonové stopy. V případě ANT-quantity a ANT-density dochází k úpravě feromonové stopy již v okamžiku, kdy mravenec vybere následující vrchol. Oproti tomu u ANT-cycle dochází k aktualizaci, až když všichni mravenci dorazí do cíle. Z těchto tří metod právě ANT-cycle dosahoval nejlepších výsledků. Postupem času došlo k tomu, že z ANT-cycle se stal pouze Ant System a ostatní dvě metody zůstaly nevyužity. Ant System tak tvoří základní mravenčí algoritmus. Proto je v následující části popsán spolu s dalšími mravenčími algoritmy v kontextu řešení problému obchodního cestujícího. Jak již bylo popsáno v podkapitole 0, cílem problému obchodního cestujícího je najít nejkratší cestu mezi navzájem propojenými městy a to tak, že je projdeme pouze jedenkrát (s výjimkou startovního města, do kterého se musí cestující vrátit). Problém lze reprezentovat neorientovaným grafem $G = (V, E)$, kde V je množina všech vrcholů grafu, jež reprezentují města a E je množinou všech hran odpovídajících všem přímým cestám mezi městy. V následující části popíšeme ty nejdůležitější ACO metody.

Tab. 2 Přehled nejznámějších ACO algoritmů pro řešení TSP

ACO algoritmus	Reference	Rok publikování
Ant System	[128, 129]	1991
Elitist AS	[128, 129]	1992
Ant Colony System	[130]	1996
MMAS	[132]	1996
Ranked AS	[150]	1997
Best-Worst AS	[151, 152]	2000
Population-based ACO	[153]	2002
Beam ACO	[75, 75]	2004
ACO+C	[154, 155]	2008

5.3.2 Ant System

Je prvním mravenčím algoritmem a je předlohou pro mnohé další algoritmy, které z něj vycházejí. Prvním krokem algoritmu je inicializace hodnot feromonů na hranách. Tato hodnota by měla být mírně vyšší než hodnoty očekávané v prvních iteracích prohledávání grafu. Zamezíme tak ovlivnění řešení v počátku prohledávání, kdy při nízké úrovni feromonu τ_0 by mohlo dojít k rychlému zaměření pouze na určité části prohledávaného prostoru. Oproti tomu při velké počáteční hodnotě feromonu τ_0 by trvalo několik iterací, než by došlo k jeho odpaření z neperspektivních částí a mravenci mohli ovlivnit prohledávání. Dorigo [124] pro její odhad doporučuje použít rovnici:

$$\tau_0 = \frac{m}{C^{nn}}, \quad \forall (i,j) \in L, \tag{5.2}$$

kde τ_0 je základní úroveň feromonové stopy na všech hranách grafu, m počet mravenců a C^{nn} je délka trasy dle „*nearest neighbour*“ [148] heuristiky.

Po inicializační fázi následuje hlavní cyklus, ve kterém jsou provedeny nejdůležitější části ACO metaheuristiky. Nejprve je nutné vytvořit mravenčí řešení (*constructAntSolutions*). Proto jsou mravenci náhodně umístěni do startovního vrcholu, ze kterého se postupně vydávají na cestu po grafu, dokud nenarazí na cílový vrchol. Každý mravenec prochází grafem samostatně a výběr vrcholu, do kterého přejde, je realizován náhodně dle pravděpodobnosti p_{ij}^k. Pro každého mravence k v aktuálním vrcholu i platí, že si zvolí vrchol j z množiny přípustných vrcholů $\mathcal{N}_i^k$ a tedy $j \in \mathcal{N}_i^k$. Množina přípustných vrcholů $\mathcal{N}_i^k$ je množina neobsahující již navštívené vrcholy, které má mravenec uloženy v paměti. Pro volbu následujícího vrcholu je použita pravděpodobnostní rovnice přechodu označovaná také jako náhodné proporcionální pravidlo [124]:

$$p_{ij}^k = \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in \mathcal{N}_i^k} [\tau_{il}]^\alpha [\eta_{il}]^\beta}, \quad if\ j \in N_i^k, \tag{5.3}$$

kde τ_{ij} je intenzita feromonů na hraně mezi vrcholy i a j, dále η_{ij} je ohodnocení hrany. Hodnoty α a β jsou parametry, pomocí kterých se ovlivňuje důležitost feromonové stopy nebo heuristické informace. Parametr α posiluje význam feromonů a β posiluje důležitost heuristické informace. V případě, že nastavíme hodnotu $\alpha = 0$ a omezíme tak vliv feromonové stopy, změní se z mravenčího algoritmu na algoritmus hladový. V opačném případě, kdy $\beta = 0$ a mravenci se spoléhají pouze na feromonovou stopu, dochází k brzkému uvíznutí v lokálním minimu a výsledky hledání jsou suboptimální. V literatuře [124] jsou pro problém TSP doporučovány následující hodnoty $\alpha = 1$ a $\beta \in \langle 2{,}5 \rangle$. Jakmile dokončí mravenci svůj průchod grafem, lze použít pro nalezená řešení vybraný algoritmus pro lokální prohledávání. Například algoritmus 2-opt [156] nebo 3-opt [157].

Aktualizace feromonové stopy

Dalším krokem je provedení aktualizace feromonové stopy. V případě Ant System algoritmu je provedena aktualizace všemi mravenci. Nejprve se část feromonů nechá vyprchat dle rovnice:

$$\tau_{ij} = (1 - \rho)\tau_{ij}, \quad \forall (i,j) \in L, \tag{5.4}$$

kde ρ je konstanta vypařování. Ta je spolu s parametry α a β pevně nastavena před spuštěním algoritmu.

Po proběhlém odpaření se provede položení nových feromonů dle rovnice:

$$\tau_{ij} = \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}, \quad \forall (i,j) \in L, \tag{5.5}$$

a

$$\Delta\tau_{i,j}^{k} = \begin{cases} 1/C^{k}, & if\ hrana(i,j) \in T^{k}; \\ 0, & jinak; \end{cases} \tag{5.6}$$

kde m je počet mravenců, $\Delta\tau_{i,j}^{k}$ je množství feromonu, jež bude pokládat mravenec k na hrany, skrze které prošel a C^{k} je celková délka trasy každého mravence.

5.3.3 Elitist Ant System

Je první úpravou Ant System algoritmu [124, 128, 129]. EAS modifikace je založena na dodatečném posílení feromonové stopy na hrany, které patří na dosud nejlepší nalezenou cestu grafem. Tu značíme T^{bs} (*best-so-far tour*) a C^{bs} je délka této nejlepší cesty. Upravená rovnice pro aktualizaci feromonové stopy je následující:

$$\tau_{ij} = \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{i,j}^{k} + e\Delta\tau_{i,j}^{bs}, \quad \forall (i,j) \in L. \tag{5.7}$$

Přičemž $\Delta\tau_{i,j}^{k}$ má stejnou definici jako rovnice (5.6) a $\Delta\tau_{i,j}^{bs}$ je definována:

$$\Delta\tau_{i,j}^{bs} = \begin{cases} 1/C^{bs}, & if\ hrana(i,j) \in T^{bs}; \\ 0, & jinak; \end{cases} \tag{5.8}$$

Evaporace feromonové stopy zůstává stejná jako v případě Ant System algoritmu.

5.3.4 Ranked Ant System

Jedná se o další úpravu Ant System algoritmu, která byla publikována v článku „*A new rank-based version of the Ant System: A computational study*“ [150] z roku 1999 autorů Bullnheimer, Hartl a Strauß. Ranked Ant System (AS_{rank} případně RAS) podobně jako v případě EAS upravuje způsob aktualizace feromonové stopy. V tomto případě se množství feromonu, které bude mravenci položeno, odvíjí dle jejich úspěšnosti v nalezení nejkratší cesty. Také je omezen počet mravenců, kterým je povoleno aktualizovat feromonovou stopu. Proto je nutné před samotnou aktualizací setřídit mravence dle délky nalezené cesty a přiřadit jim ohodnocení r. Z tohoto žebříčku se vybere $w-1$ nejlepších mravenců spolu s mravencem, který nalezl dosud nejlepší řešení T^{bs}. Pro tuto skupinu vybraných se provede aktualizace feromonové stopy dle následující rovnice:

$$\tau_{ij} = \tau_{ij} + \sum_{r=1}^{w-1} (w-r)\Delta\tau_{i,j}^{r} + w\Delta\tau_{i,j}^{bs}, \quad \forall (i,j) \in L, \tag{5.9}$$

přičemž $\Delta\tau_{i,j}^{r} = \frac{1}{C^{r}}$ a C^{r} je délka cesty mravence dle jeho ohodnocení. Hodnota $\Delta\tau_{i,j}^{bs} = \frac{1}{C^{bs}}$ a C^{bs} je délka cesty dosud nejlepšího mravence.

5.3.5 MaxMin Ant System

Byl publikován v článku „*The MAX-MIN ant system and local search for the traveling salesman problem*“ v roce 1997 autory Thomasem Stützlem a Holgerem Hoosem [130]. Oproti předchozím modifikacím Ant System algoritmu, MaxMin Ant system ($\mathcal{MM}$AS) zavádí několik výrazných změn. Například aktualizaci feromonové stopy provádí pouze nejlepší mravenec, ať už se jedná o nejlepšího mravence v dané iteraci nebo mravence s dosud nejlepší nalezenou cestou. Použití této strategie by vedlo k rychlému ustálení nalezeného řešení díky hromadění feromonu pouze na jedné trase. Nemuselo by tak dojít k nalezení kvalitnějších výsledků. Proto autoři zavedli protiopatření v podobě omezení hodnot feromonu na hranách. Je dána dolní mez τ_{min} a horní mez τ_{max} hodnot feromonu. Pokud dojde k překročení jedné z těchto mezí, je hodnota feromonu nastavena na příslušnou mezní hodnotu. Další změnou oproti předchozím modifikacím je nastavení základní úrovně feromonu na jeho horní mez τ_{max}. V případě, že v průběhu algoritmu dojde k stagnaci nalezeného řešení, nebo po n iterací nedojde ke změně v nalezeném řešení, dochází k re-inicializaci, kdy je feromonová stopa opět nastavena na jeho horní mez τ_{max}.

Postup aktualizace feromonové stopy u $\mathcal{MM}$AS algoritmu vychází opět z AS algoritmu, kdy nejprve dochází k evaporaci feromonu a následně je aktualizován dle rovnice:

$$\tau_{ij} = \tau_{ij} + \Delta\tau_{i,j}^{best}, \quad \forall(i,j) \in L, \tag{5.10}$$

kde $\Delta\tau_{i,j}^{best} = \frac{1}{C^{best}}$. Pokud je jako $\Delta\tau_{i,j}^{best}$ vybráno dosud nejlepší nalezené řešení, pak $\Delta\tau_{i,j}^{best} = \frac{1}{C^{bs}}$, v opačném případě $\Delta\tau_{i,j}^{best} = \frac{1}{C^{ib}}$ odpovídá nejlepšímu nalezenému řešení v iteraci, přičemž C^{ib} je délka příslušné cesty. Dorigo ve své knize [124] uvádí, že pro malé instance problému obchodního cestujícího je vhodné použít pro aktualizaci feromonu řešení nejlepšího mravence z dané iterace $\Delta\tau_{i,j}^{ib}$. Oproti tomu u velkých instancí je pak výhodnější nejlepší dosud nalezené řešení $\Delta\tau_{i,j}^{bs}$. Pro aktualizaci feromonové stopy je také nutno nastavit mezní hodnoty feromonu. Horní mezní hodnota je stanovena dle $\tau_{max} = \frac{1}{\rho C^{bs}}$ a při změně nové nejlepší hodnoty je horní mez aktualizována. Dolní mez τ_{min} je nastavena dle $\tau_{min} = \frac{\tau_{max}}{a}$, kde a je parametr.

5.3.6 Ant Colony System (ACS)

Ant Colony System neboli ACS [131, 131] je algoritmus, který se podobně jako $\mathcal{MM}$AS výrazněji odlišuje od Ant System algoritmu. Dorigo a Gambaredella jej publikovali v článku s názvem *„Ant Colony System: A cooperative learning approach to the traveling salesman problem“* v roce 1997 [131]. Tento algoritmus nahrazuje starší algoritmus Ant-Q [133], který se snaží o kombinaci mravenčích algoritmů s posilovaným učením. První odlišností od ostatních ACO algoritmů je úprava rovnice pro výpočet inicializační hodnoty feromonu:

$$\Delta\tau_0 = \frac{1}{nC^{nn}}. \tag{5.11}$$

kde n je počet měst a C^{nn} je stejně jako u Ant System algoritmu délka cesty dle „*nearest neighbour*“ heuristiky. Dále se oproti ostatním algoritmům odlišuje úpravou rozhodovacího pravidla. Pomocí parametru q_0 se podařilo přidat přímý způsob, jak ovlivnit výběr mezi prohledáváním nových hran a využitím zkušeností získaných mravenci.

Tento parametr je použitý v pravidlu pro výběr následujícího vrcholu, u ACS algoritmu jej označujeme jako pseudonáhodné proporcionální pravidlo:

$$j = \begin{cases} \text{argmax}_{\ l \in \mathcal{N}_i^l} \left\{ \tau_{ij} \left[\eta_{il} \right]^{\beta} \right\}, & if\ q \leq q_0 \\ p_{AS}, & jinak; \end{cases}, \tag{5.12}$$

kde q odpovídá náhodné hodnotě s uniformním rozložením v $\langle 0,1 \rangle$ a q_0 $(0 \leq q_0 \leq 1)$ je zmiňovaný parametr ovlivňující výběr dosud nejlepší nalezené cesty, pro $q > q_0$ bude použito stejné pravidlo přechodu (5.3) z Ant System algoritmu. Další výrazná odlišnost algoritmu ACS spočívá v aktualizaci feromonové stopy, ta je rozdělena na globální a lokální aktualizaci. Globální aktualizace feromonu je umožněna pouze mravenci s dosud nejlepším nalezeným řešením T^{bs}. Nejprve tak dojde k odpaření feromonu na hranách patřících do T^{bs} a pak k aktualizaci dle rovnice:

$$\tau_{ij} = (1-\rho)\tau_{ij} + \rho \Delta \tau_{ij}^{best}, \forall (i,j) \in T^{bs}, \tag{5.13}$$

kde $\Delta \tau_{i,j}^{best} = \frac{1}{C_{best}}$.

Oproti ostatním ACO algoritmům obsahuje ACS i lokální aktualizaci feromonové stopy, ke které dochází ihned po přechodu hrany (i,j) mravencem. Lokální aktualizace je dána vztahem:

$$\tau_{ij} = (1-\xi)\tau_{ij} + \xi \tau_0, \tag{5.14}$$

kde ξ, $0 < \xi < 1$ a τ_0 jsou dva předem dané parametry. Hodnota τ_0 je shodná s inicializační hodnotou feromonové stopy. Parametr ξ je síla, s jakou dojde k odpařování. Dorigo [124] doporučuje experimentálně [130, 131] zjištěnou hodnotu $\xi = 0{,}1$. Výraz $\xi\tau_0$ zajišťuje, aby se hodnota feromonu co nejvíce přibližovala inicializační hodnotě. Lokální aktualizace feromonu je u ACS způsob, jak zamezit stagnaci ve vyhledávání řešení. Budou-li mravenci preferovat pouze nejlepší řešení, dojde k rychlému snížení atraktivity takovéto cesty, a tím ke zvýšení prohledávání neprozkoumaných hran.

V průběhu téměř třiceti let od první publikace o mravenčích algoritmech vzniklo množství dalších mravenčích algoritmů a jejich variant. Pro řešení problému obchodního cestujícího lze uvést několik následujících. Best-Worst Ant System [151, 151] je z roku 2000. U této varianty dochází k aktualizaci feromonové stopy pouze na nejlepší nalezené cestě T^{bs} *a* také současně nejhoršímu nalezenému řešení v dané iteraci T^{cw}. V případě T^{bs} dojde k pozitivnímu posílení a u T^{cw} dojde k negativnímu posílení feromonu. Podobně jako $\mathcal{MM}$AS umožňuje re-inicializaci feromonové stopy. Další variantou je Population-based ACO [153] z roku 2002. Zde se udržuje populace nejlepších řešení z několika iterací T^{ib}. Dokud není nalezeno k těchto řešení, nedochází odpařování feromonové stopy a pouze T^{ib} tuto stopu aktualizuje. Jakmile je dosaženo limitu k, je nejhorší řešení z této populace odstraněno a spolu s ním dojde i k odpaření feromonu z hran tohoto řešení. Beam ACO je hybridní algoritmus z roku 2004 [75, 158], který kombinuje mravenčí algoritmy s klasickou metodou paprskového prohledávání. Metoda ACO+C publikovaná v roce 2008 [154] je modifikace, která se snaží napodobit rozdělení do skupin, či chceme-li kast v mravenčí kolonii. Kolonii tvoří nejčastěji královna, samečci a nejpočetnější skupinou jsou dělnice. V případě některých druhů, jako je například *Atta cephalotes* [159], se dělnice s velkými kusadly člení do kasty vojáků [160]. Publikovaná úprava napodobuje kastovní systém takovým způsobem, že rozděluje virtuální mravence do skupin s rozdílnými parametry α a β. To umožňuje vytvořit skupinu mravenců kladoucí větší důraz na prohledávání neprozkoumaných

hran a zároveň mít další skupinu mravenců více ovlivněnou feromonovou stopou. Autoři tímto způsobem upravují algoritmus $\mathcal{MM}$AS označený pak jako $\mathcal{MM}$AS+C, samotná úprava ale není na tento algoritmus vázána a lze ji použít i na ostatní mravenčí algoritmy. Podobný způsob je popsán také v článku z roku 2011 [155], kde je upraven algoritmus Ant Colony System – ACS a jeho parametr q_0, který ovlivňuje výběr mezi prohledáváním nových hran a využitím zkušeností získaných mravenci. Další způsobem, jak zlepšit vlastnosti mravenčích algoritmů je hybridizace ACO s jinou metaheuristikou. V literatuře se tak lze setkat s příklady kombinování s rojem částic (PSO) [161], zakázaným hledáním (Tabu search) [162] či simulovaným žíháním (Simulated annealing) [163].

Výše uvedené mravenčí algoritmy lze použít nejen pro řešení problému obchodního cestujícího, ale i pro další optimalizační problémy. Ať už se jedná o problémy z kategorie směrových, do které patří i problém obchodního cestujícího, nebo dalších kategorií jako například přiřazovacích problémů, plánovacích problémů a dalších. ACO lze použít nejen na diskrétní problémy, ale také i na kontinuální optimalizační problémy. Podrobnému popisu stavu mravenčích algoritmů a jejich použití se věnují přehledové články [151, 164].

6 IMPLEMENTOVANE ALGORITMY PRO CTP

Praktická část realizované disertační práce je vystavěna na vlastním návrhu a pokročilé implementaci výpočetního software pro řešení uvedené velmi specifické třídy kombinatorických problémů – problém kanadského cestujícího (*Canadian Traveller Problem – CTP*). Poznamenejme, že navržený software je rovněž schopen řešit problém obchodního cestujícího (*Traveling Salesman Problem – TSP*) a kombinaci problému kanadského cestujícího a obchodního cestujícího označovanou jako CTSP (*Canadian Traveling Salesman Problem*).

Připomeňme, že standardní problém kanadského cestujícího popisuje situaci, kdy agent při průchodu prohledávaným grafem G zjistí, že daná hrana vycházející z vrcholu, kde se agent právě nachází je zablokována, a tím pádem je neprůchozí. Cílem agenta je pak zvolit strategii, buď čekat na odblokování, pokud je reálné, nebo najít novou vhodnou trasu, která minimalizuje očekávané cestovní náklady při cestě ze startovního vrcholu do cílového vrcholu.

Jak bylo uvedeno v teoretické části práce, problém CTP je variantou plánování cesty v dynamicky se měnícím prostředí, které je simulováno pomocí pravděpodobností a celkovou nepředvídatelností ohodnocení grafu. V ohledu definice i řešících algoritmů má CTP významné odlišnosti od TSP. Zásadní odlišnosti v definici úlohy si vyžádaly nové definice řešičů na bázi heuristik a mravenčích kolonií. Zdůrazněme, že v ohledu použití mravenčích algoritmů jde o zcela nový přístup k řešení CTP problému. Vytvořený software implementuje pro řešení problému kanadského cestujícího dvě kategorie řešících algoritmů, a to mravenčí a heuristické algoritmy. V této kapitole představíme všechny diskutované a implementované metody řešiče CTP.

6.1 Implementované metody

Z kategorie heuristických metod jsou implementovány vybrané algoritmy včetně modifikací pro řešení obnovitelné varianty CTP. Pro řešení problému obchodního cestujícího (TSP) a problému kombinujícího oba předchozí problémy do problému kanadského obchodního cestujícího (CTSP) jsou implementovány pouze základní mravenčí algoritmy a nejsou v následující části dále diskutovány.

Heuristické algoritmy	**Mravenčí (ACO) algoritmy**
• Backtrack • Greedy • Comparison • Comparison With Penalization • Recovery Backtrack • Recovery Greedy • Recovery Comparison • Recovery Comparison With Penalization • Dijkstra-Waiting • Dijkstra-SRCTP	• Ant System • Ranked Ant System • Elistist Ant System • MaxMin Ant System • Ant Colony System • Restarted MaxMin Ant System • Global Parameter Learning Ant System • Global Parameter Learning MaxMin Ant System • ACO+C

Každá metoda z uvedeného výčtu je rovněž doplněna o variantu tzv. mravenčích kast, označená jako ACO+C. Tento přístup je inspirován biologickým jevem, kdy různé „kasty“ mravenců (například dělníci, vojáci) mají specifické role v kolonii. V kontextu ACO, metoda ACO+C přidává další úroveň specializace do tradičního modelu tím, že přiřazuje různé skupiny „virtuálních mravenců“ k různým úkolům nebo fázím řešení problému. Tyto skupiny mohou být optimalizovány pro specifické aspekty problému, jako je průzkum nových oblastí řešení, intenzifikace hledání kolem slibných řešení, nebo diverzifikace, aby se zabránilo uvíznutí v lokálních optimech. Dále jsou v implementaci zahrnuty nové experimentální metody Global Parameter Learning Ant System vycházející z metody Ant System a metody Restarted Ant System, označované dále zkratkou GPL uvedenou jako prefix základní metody.

6.2 Heuristické metody pro neznámou pravděpodobnost blokace

V této části se budeme zabývat popisem heuristických metod pro neznámou pravděpodobnost blokace. Jedná se o algoritmy, které dopředu neznají stav hran grafu. Daný stav hrany se zjistí až ve vrcholu incidentním s touto hranou. Přesto, že podrobný popis algoritmů se již nachází v kapitole 4.1, pro ucelenost a konzistenci výkladu je připomenut i zde na následujících pseudokódech, a to jak pro algoritmus *Backtrack*, tak i pro algoritmy *Greedy* a *Comparison*. Vstupy algoritmu tvoří graf $G(V, E)$, startovní vrchol s a cílový vrchol t. Výstupem je pak nalezená cesta, pojmenována jako *tour,* z počátečního do koncového vrcholu $s - t$.

6.2.1 Heuristické metody pro neobnovitelnou variantu CTP (*k-CTP*)

Prvním implementovaným algoritmem je *Backtrack* od Stephena Westphala [83]. Nejprve je nalezena nejkratší cesta ze startovního vrcholu do cílového pomocí Dijkstrova algoritmu, aniž by se zjišťoval aktuální stav hran. Na nalezenou nejkratší cestu *sp* (shortest path) se ve funkci *PassThrough* vydává agent, který při svém průchodu grafem bere v potaz aktuální stav hran nacházející se na nalezené cestě *sp*. Pokud je nadcházející hrana z vrcholu, kde se agent nachází, zablokována, vrací se agent do startovního vrcholu, tento návrat zpět na start je zahrnutý do výsledné cesty. Opět dochází k hledání nové *sp* pomocí Dijkstrova algoritmu, přičemž zablokovaná hrana zůstává zablokována a při hledání nejkratší cesty není uvažována. Cyklus se opakuje, dokud agent nedorazí do cílového vrcholu t.

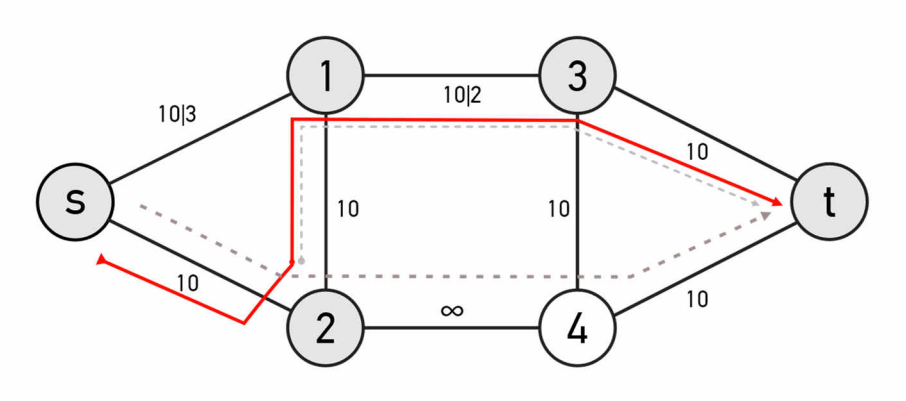
Algoritmus 6 k-CTP Backtrack

```
1:  function BACKTRACK(G, s, t)
2:      tour ← list()
3:      while true do
4:          sp ← DIJKSTRA(G, s, t)
5:          solutionFound, path ← PASSTHROUGH(G, sp)
6:          if solutionFound then
7:              return tour + path
8:          else
9:              RETURNTOSTART(path, tour)
10:         end if
11:     end while
12: end function
```

Další uvedený algoritmus se nazývá Greedy a pochází od Yinfenga Xu a jeho kolektivu [84]. Podobně jako u předchozího algoritmu Backtrack se nejprve pomocí Dijkstrova algoritmu nalezne nejkratší cesta *sp* ze startovního vrcholu, na kterou následně vyráží agent. Pokud však v jeho průchodu dojde k zablokování některé hrany, agent se nevrací zpět na start, jak tomu bylo u algoritmu Backtrack. Vrchol, ze kterého zablokovaná hrana vychází, je použit jako startovní vrchol pro nalezení nové nejkratší cesty *sp*, za pomocí Dijkstrova algoritmu. Tento postup se opakuje, dokud agent nedorazí do cílového vrcholu.

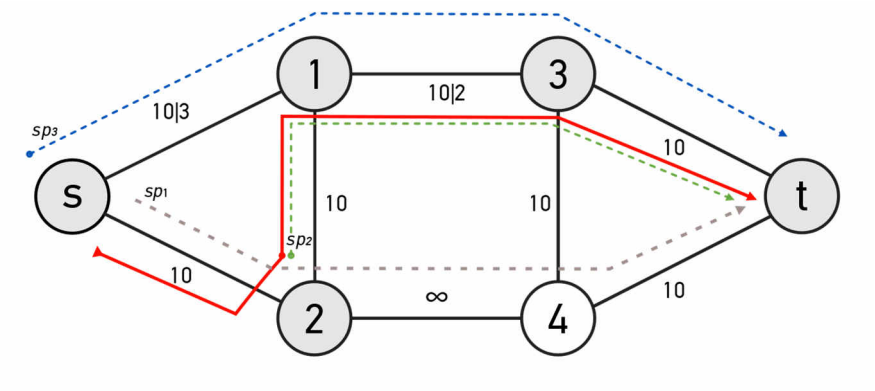
Algoritmus 7 k-CTP Greedy

```
1: function GREEDY(G, s, t)
2:     tour ← list()
3:     start ← s
4:     while true do
5:         sp ← DIJKSTRA(G, start, t)
6:         solutionFound, path, lastNode ← PASSTHROUGH(G, sp)
7:         if solutionFound then
8:             return tour + path
9:         else
10:            tour ← tour + path
11:            start ← lastNode
12:        end if
13:    end while
14: end function
```



Obr. 31. Průběh algoritmu *Greedy*

Ve stejném článku, jako byl publikován algoritmus *Greedy,* byl publikován i algoritmus *Comparison*. Ten specificky kombinuje způsob vyhledávání nové nejkratší cesty *sp* dvou předchozích algoritmů *Backtrack a Greedy*. Po nalezení nejkratší cesty $s - t$ Dijkstrovým algoritmem, se na tuto cestu ze startovního do cílového vrcholu vydává agent. V případě, že v průběhu dojde k zablokování některé z hran, vyhledá se nová nejkratší cesta *spFromLast* (na následujícím obrázku uvedena jako sp_2) z vrcholu, ve kterém se agent nachází, jako to bylo u algoritmu *Greedy*. Také se znovu vyhledá nejkratší cesta *spFromStart* (sp_3) ze startovního vrcholu (včetně započtení návratu zpět do startu), což je převzato od algoritmu *Backtrack,* kde blokovaná hrana je zanedbána. Následně dojde k porovnání délek těchto nejkratších cest, ze kterých je vybrána ta výhodnější. Po zvolené cestě pak pokračuje agent v průchodu grafem, dokud nedorazí do cílového vrcholu t.



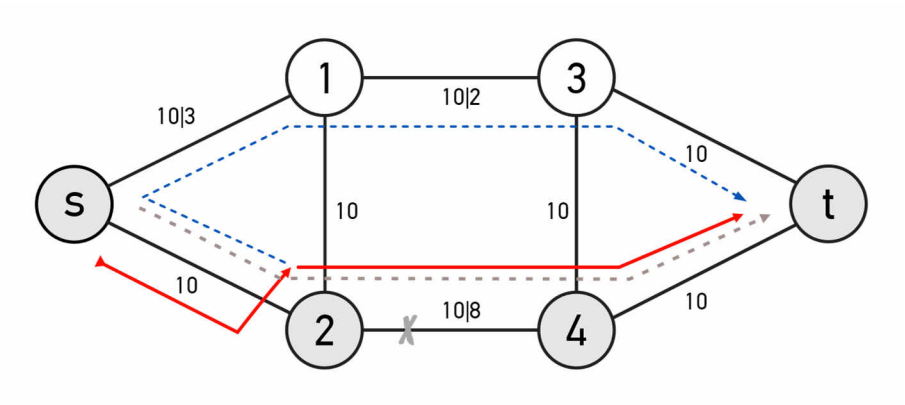
Obr. 32. Průběh algoritmu *Comparison*

Algoritmus 8 k-CTP Comparison

```
1:  function COMPARISON(G, s, t)
2:      tour ← list()
3:      start ← s
4:      while true do
5:          sp ← DIJKSTRA(G, start, t)
6:          solutionFound, path, lastNode ← PASSTHROUGH(G, sp)
7:          if solutionFound then
8:              return tour + path
9:          else
10:             spFromStart ← DIJKSTRA(G, start, t)
11:             spFromLast ← DIJKSTRA(G, lastNode, t)
12:             lengthFromStart ← LENGTH(spFromStart, path)
13:             lengthFromLast ← LENGTH(spFromLast)
14:             if lengthFromStart < lengthFromLast then
15:                 tour ← tour + path + REVERSE(path)
16:             else
17:                 tour ← tour + path
18:                 start ← lastNode
19:             end if
20:         end if
21:     end while
22: end function
```

6.2.2 Heuristické metody pro obnovitelnou variantu CTP (*r-CTP*)

Pro řešení obnovitelné varianty CTP bylo navrženo několik vlastních algoritmů, které vychází z předchozích algoritmů pro *k-CTP*. Prvním takovým je *r-CTP Backtrack*. Stejně jako u *k-CTP* varianty se agent vydává po nejkratší cestě $s - t$ nalezené pomocí Dijkstrova algoritmu. Rozdíl mezi *r*-CTP a *k*-CTP spočívá v penalizaci. V případě, že při průchodu dojde k zablokování některé z hran, je zjištěna délka cesty z aktuálního vrcholu do cílového vrcholu se započtením časové penalizace u zablokované hrany *lengthWaiting*. Dále se zjistí délka cesty zpět do startovního vrcholu *lengthBackToStart*. V případě, že je odhadovaná délka cesty do cíle menší, než zpět na start, vyčká agent na obnovení zablokované hrany a pokračuje po cestě *sp* do cílového vrcholu *t*.



Obr. 33. Příklad průběhu algoritmu *Backtrack* pro obnovitelnou variantu CTP

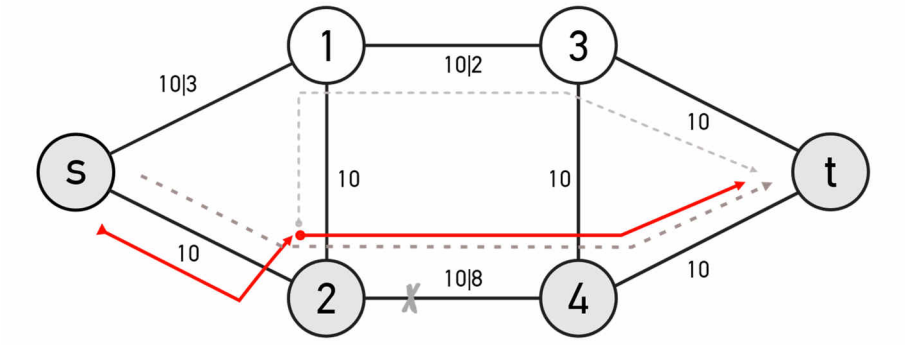
Algoritmus 9 r-CTP Backtrack

```
1:  function BACKTRACK(G, s, t)
2:      tour ← list()
3:      while true do
4:          sp ← DIJKSTRA(G, s, t)
5:          solutionFound, path, lastNode, blockedEdge ← PASSTHROUGH(G, sp)
6:          if solutionFound then
7:              return tour + path
8:          else
9:              edgeCost ← GETEDGECOST(blockedEdge)
10:             lengthWaiting ← LENGTH(sp, path, blockedEdge)
11:             lengthBackToStart ← LENGTH(path)
12:             if lengthWaiting ≤ lengthBackToStart then
13:                 tour ← tour + path + edgeCost
14:                 start ← GETNEXTNODE(lastNode, blockedEdge)
15:             else
16:                 RETURNTOSTART(path, tour)
17:             end if
18:         end if
19:     end while
20: end function
```

U algoritmu *r*-CTP *Greedy* se v případě zablokování hrany při průchodu agenta zjistí zbývající délka cesty se započtením časové penalizace takovéto hrany *lengthWaiting*. Oproti předchozímu algoritmu *r*-CTP *Backtrack* se nezjišťuje délka cesty zpět na startovní vrchol, ale je nalezena nová nejkratší cesta z aktuálního vrcholu do cílového. Délky cest jsou porovnány a pokračuje se po výhodnější z nich. Postup se opakuje, dokud agent nedorazí do cílového vrcholu t.

Algoritmus 10 r-CTP Greedy

```
1:  function GREEDY(G, s, t)
2:      tour ← list()
3:      start ← s
4:      while true do
5:          sp ← DIJKSTRA(G, start, t)
6:          solutionFound, path, lastNode, blockedEdge ← PASSTHROUGH(G, sp)
7:          if solutionFound then
8:              return tour + path
9:          else
10:             lengthWaiting ← LENGTH(sp, path, blockedEdge)
11:             spFromLast ← DIJKSTRA(G, lastNode, t)
12:             lengthFromLast ← LENGTH(spFromLast)
13:             if lengthWaiting ≤ lengthFromLast then
14:                 tour ← tour + path + GETEDGECOST(blockedEdge)
15:                 start ← GETNEXTNODE(lastNode, blockedEdge)
16:             else
17:                 tour ← tour + path
18:                 start ← lastNode
19:             end if
20:         end if
21:     end while
22: end function
```

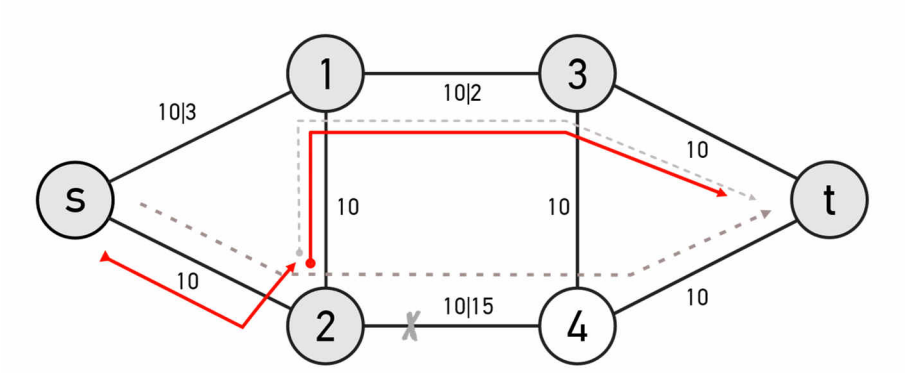


Obr. 34. Průběh algoritmu *r*-CTP *Greedy*

Algoritmus *r*-CTP *Greedy With Penalization* se od předchozího algoritmu *r*-CTP *Greedy* odlišuje tím, že v místě blokace je nalezena nová nejkratší cesta *spFromLast* pomocí funkce *DijkstraWithPenalization*. Rozdíl od běžné varianty Dijkstrova algoritmu spočívá v tom, že jsou započteny i případné penalizace u hran s nenulovou pravděpodobností blokace. Agent následně pokračuje po nové nejkratší cestě *sp a* v případě zablokované hrany je penalizace započtena k ohodnocení hrany.

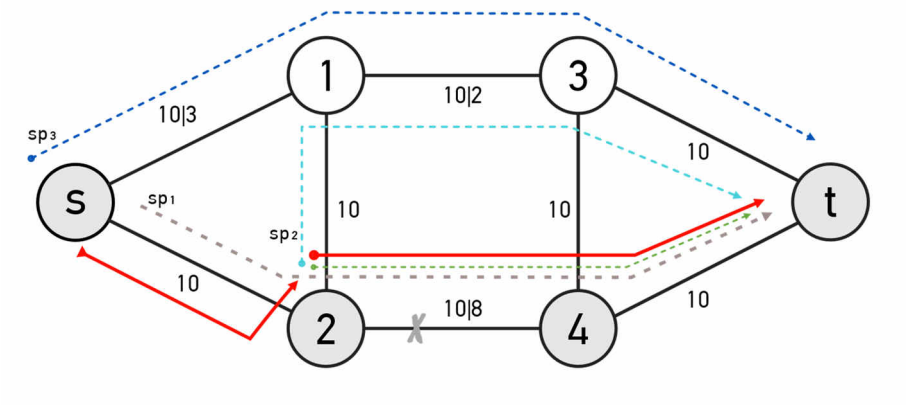
Algoritmus 11 r-CTP Greedy With Penalization

```
1:  function GreedyWithPenalization(G, s, t)
2:      tour ← list()
3:      start ← s
4:      while true do
5:          sp ← Dijkstra(G, start, t)
6:          solutionFound, path, lastNode, blockedEdge ← PassThrough(G, sp)
7:          if solutionFound then
8:              return tour + path
9:          else
10:             spFromLast ← DijkstraWithPenalization(G, lastNode, t)
11:             nextNode, edge ← getNextNode(spFromLast, lastNode)
12:             tour ← tour + path + getEdgeCost(edge)
13:             start ← nextNode
14:         end if
15:     end while
16: end function
```



Obr. 35. Průběh algoritmu r-CTP Greedy with Penalization

Algoritmus *r*-CTP *Comparison* vychází se stejnojmenného algoritmu pro *k*-CTP. Průběh je na počátku stejný jako u předchozích variant a po nalezené *sp* cestě se vydává agent. Jestliže agent narazí na zablokovanou hranu ve vrcholu, z kterého hrana vychází, zjišťuje následující parametry: zbývající délku cesty do cílového vrcholu *lenghtWaiting* se započtenou penalizací, délku nové nejkratší cesty *lenghtFromLast* (na následujícím obrázku označena jako sp_2) do cílového vrcholu a nejkratší cestu *spFromStart* (sp_3) ze startovního vrcholu do cílového vrcholu včetně započtení návratu zpět do startovního vrcholu. Z těchto možností vybere tu nejvýhodnější.

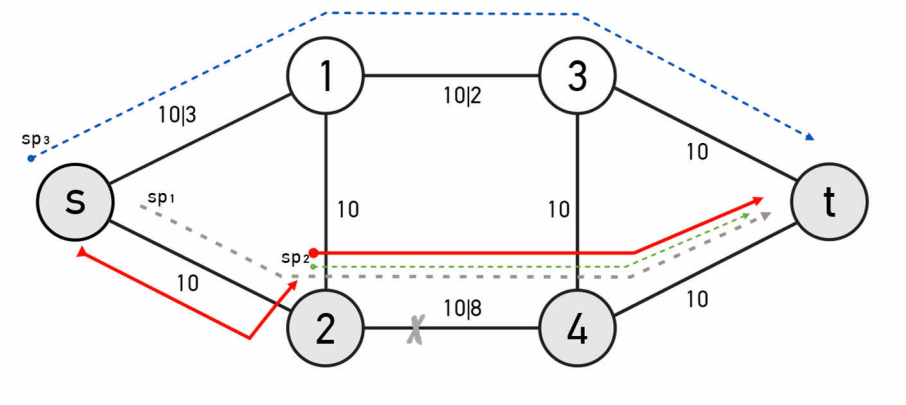


Obr. 36. Průběh algoritmu *Comparison* pro obnovitelnou variantu CTP

Algoritmus 12 r-CTP Comparison

```
1:  function COMPARISON(G, s, t)
2:      tour ← list()
3:      start ← s
4:      while true do
5:          sp ← DIJKSTRA(G, start, t)
6:          solutionFound, path, lastNode, blockedEdge ← PASSTHROUGH(G, sp)
7:          if solutionFound then
8:              return tour + path
9:          else
10:             edgeCost ← GETEDGECOST(blockedEdge)
11:             lengthWaiting ← LENGTH(sp, path) + edgeCost
12:             spFromLast ← DIJKSTRA(G, lastNode, t)
13:             lengthFromLast ← LENGTH(spFromLast)
14:             spFromStart ← DIJKSTRA(G, start, t)
15:             lengthFromStart ← LENGTH(spFromStart, path)
16:             if lengthFromLast < lengthWaiting then
17:                 if lengthFromLast ≤ lengthFromStart then
18:                     tour ← tour + path
19:                     start ← lastNode
20:                 else
21:                     tour ← tour + path + REVERSE(path)
22:                     start ← s
23:                 end if
24:             else
25:                 if lengthWaiting ≤ lengthFromStart then
26:                     tour ← tour + path + edgeCost
27:                     start ← GETNEXTNODE(lastNode, blockedEdge)
28:                 else
29:                     tour ← tour + path + REVERSE(path)
30:                     start ← s
31:                 end if
32:             end if
33:         end if
34:     end while
35: end function
```

Algoritmus *r*-CTP *Comparison With Penalization* kombinuje přístup algoritmu *Greedy With Penalization* s předchozím algoritmem *r*-CTP *Comparison*. Pokud agent narazí na zablokovanou hranu, je nalezena nová nejkratší cesta *spFromLast* (sp_2 na následujícím obrázku) pomocí funkce *DijkstraWithPenalization*, která započte možné penalizace. Délka této nové cesty je pak porovnána s délkou nově nalezené nejkratší cesty ze startovního vrcholu *spFromStart* (sp_3), do které je zahrnuta i cesta zpět. Agent pak pokračuje po výhodnější trase, dokud nedorazí na další zablokovanou hranu nebo do cílového vrcholu.



Obr. 37. Průběh algoritmu *Comparison With Penalization* pro obnovitelnou variantu CTP

Algoritmus 13 r-CTP Comparison With Penalization

```
1:  function ComparisonWithPenalization(G, s, t)
2:      tour ← list()
3:      start ← s
4:      while true do
5:          sp ← Dijkstra(G, start, t)
6:          solutionFound, path, lastNode, blockedEdge ← PassThrough(G, sp)
7:          if solutionFound then
8:              return tour + path
9:          else
10:             spFromLast ← DijkstraWithPenalization(G, lastNode, t)
11:             lengthFromLast ← length(spFromLast)
12:             spFromStart ← Dijkstra(G, start, t)
13:             lengthFromStart ← length(spFromStart, path)
14:             if lengthFromStart < lengthFromLast then
15:                 tour ← tour + path + reverse(path)
16:                 start ← s
17:             else
18:                 nextNode ← getNextNode(spFromLast, lastNode)
19:                 tour ← tour + path
20:                 start ← nextNode
21:             end if
22:         end if
23:     end while
24: end function
```

6.3 Heuristické metody pro známou pravděpodobnost blokace

Pro řešení varianty CTP se známou pravděpodobností blokace a obnovitelnými hranami (SRCTP) byl navržen heuristický algoritmus pojmenovaný *Dijkstra-SRCTP*. Nejprve je nalezena nejkratší cesta sp ze startovního vrcholu s do cílového vrcholu t pomocí Dijkstrova algoritmu, kde pro ohodnocení hran je použita následující rovnice (6.1) vyjadřující střední dobu trvání:

$$weight(e) = w(e) + p(e)r(e), \tag{6.1}$$

kde $w(e)$ je ohodnocení hrany, $p(e)$ odpovídá pravděpodobnosti blokace a $r(e)$ označuje časovou penalizaci neboli dobu čekání. Agent se následně vydává ze startovního vrcholu s a prochází grafem G do cílového vrcholu t po nalezené cestě sp, přičemž ohodnocení hrany je dáno rovnicí:

$$weight(e) = w(e) + b(e)r(e), \tag{6.2}$$

kde $b(e)$ odpovídá stavu hrany a nabývá binárních hodnot 0 pro nezablokovanou hranu a 1 pro zablokovanou.

Algoritmus 14 Dijkstra-SRCTP

```
1:  function DijkstraSRCTP(G, s, t)
2:      tour ← list()
3:      cost ← 0
4:      node ← s
5:      sp ← DijkstraMean(G, start, t)
6:      while node ≠ t do
7:          nextNode, edge ← GetNextNode(G, sp, node)
8:          tour ← tour + nextNode
9:          cost ← cost + getEdgeCost(edge)
10:         node ← nextNode
11:     end while
12: end function
```

6.4 Mravenčí metody pro neznámou pravděpodobnost blokace

Nasazení mravenčích algoritmů na řešení problémů s neurčitostí, ať už v podobě problému kanadského cestujícího (CTP), nebo problému kanadského obchodního cestujícího (CTSP), je zcela nový přístup k jejich řešení. Z tohoto důvodu bylo nutné provést několik dílčích změn pro použití mravenčí metaheuristiky na jejich řešení. Tyto úpravy jsou společné pro všechny mravenčí algoritmy. Základní podoba metaheuristiky však zůstává stejná, jak ji definoval Marco Dorigo [124]. Příkladem praktické implementace je následující výpis algoritmu 15. Zdůrazněme, že v následujících částech práce jsou použity zjednodušené výpisy zdrojového kódu v jazyce C++, a to z důvodu použití některých nestandardních konstrukcí, které by bylo obtížné vytvořit v běžném pseudokódu.

Výpis 1 Ant Colony Optimization

```
void search()
{
     initSearch();

     while (!terminationCondition())
     {
          constructSolutions();

          localSearch();

          #if PPL
               parallel_invoke
               (
                    [&]{searchControlAndStatistics();},
                    [&]{updatePheromoneTrails();}
               );
          #else
               searchControlAndStatistics();
               updatePheromoneTrails();
          #endif
     }
}
```

První úprava se nachází ve funkci *initSearch*, která připravuje vše potřebné k samotnému hledání řešení. V rámci vlastního vývoje bylo nutné nejprve definovat, jakým způsobem budou mravenčí metody pracovat s neurčitostí daného problému. Jedním z parametrů je způsob, jakým jsou zadány pravděpodobnosti blokace p_b, se kterými pracuje zvolený mravenčí algoritmus.

Výpis 2 Init Search

```
void initSearch()
{
     initTrails();
     initSearchSettingsAndResults();

     if (useLogger)
          log→initLogger();

     if (!generateInIteration)
          if(probabilityMode)
               generateProbabilities();
          generateBlockades();
};
```

Tento způsob je dán proměnnou *probabilityMode a* jedná se výčtový typ *probabilityGen_e*, který rozlišuje tři možnosti uvedené v následujícím výpisu.

Výpis 3 Probability settings

```
enum probabilityGen_e
{
     PROB_GEN_OFF    = 0,
     PROB_GEN_RANDOM = 1,
     PROB_GEN_STATIC = 2,
};
```

Zde *prob_gen_off* určuje, že rozsah pravděpodobností je dán mapou řešeného problému, *prob_gen_random* nastavuje náhodně jejich hodnotu v rozsahu zadaném uživatelem a poslední hodnota *prob_gen_static* nastavuje pevně danou hodnotu pro pravděpodobnost blokace.

Dalším z parametrů je způsob, jakým bude docházet k případné rekonfiguraci hodnot pravděpodobnosti blokací a následnému určení stavu hran. Jejich stav není předem pro mravence znám, dokud nedorazí při průchodu grafem do vrcholu, ze kterého hrana vychází. V řešení existují pouze dvě možnosti určené hodnotou příznaku *generateInIteration*. První možností je generovat tyto parametry v inicializaci hledání řešení (funkce *initSearch*) nebo v každém volání funkce *constructSolutions*. Ta se věnuje vyhledávání řešení pomocí jednotlivých mravenců. Na následujícím výpisu je patrná už zmiňovaná úprava. Zároveň je zde provedena i inicializace jednotlivých mravenců, kteří jsou na počátku hledání umístění do startovního vrcholu.

Výpis 4 Construct Solutions

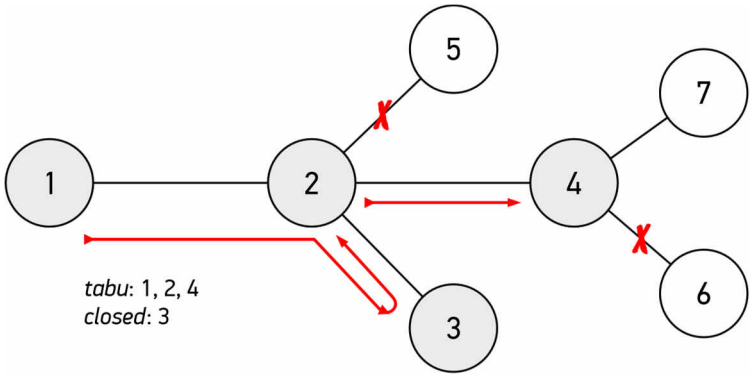
```
void constructSolutions()
{
	if (generateInIteration)
		if (probabilityMode)
			generateProbabilities();
		generateBlockades();

	resetAnts();

	parallel_for(0, AntsCount, [&](int i)
	{
		while (!isAntFinished(Ants[i]))
		{
			selectNext(Ants[i]);
		}
	});
}
```

Následně dochází ke spuštění paralelního průchodu mravenců grafem a tím i ke hledání řešení. Výběr následujícího vrcholu je proveden ve funkci *selectNext*. V případě neobnovitelné varianty problému (*k*-CTP/*k*-SCTP) může nastat situace, kdy cestovatel dorazí do vrcholu a zjistí, že došlo k zablokování všech hran a nemá tak kam přejít dál. K tomu může dojít v okamžiku, kdy je vektor pravděpodobností *selectionProbabilities* prázdný. Proto bylo nutné vytvořit způsob, který umožňuje cestovateli, tj. mravenci, návrat zpět a výběr nového následujícího vrcholu.

Výpis 5 Select Next Node

```
void selectNext(Ant<V> *ant)
{
      random = getRandom();
      actualVertexId = ant->getVertexId();
      neighbours = getNeighbours(actualVertexId, ant);

      selectionProbabilities = calcProbabilities(ant, neighbours);

      if (!selectionProbabilities.size())
      {
            ancestor = getAncestorVertex(ant);

            actualVertexId = ancestor->idx;
            neighbours = getNeighbours(actualVertexId, ant);
            selectionProbabilities = calcProbabilities(ant, neighbours);
      }

      selectedID = 0;
      probability = selectionProbabilities[selectedID];

      while (probability < random)
      {
            probability += selectionProbabilities[selectedID++];
      }

      nextNode = neighbours[selectedID]
      ant->updateTour(nextNode, getEdge(actualVertexId, nextNode));
};
```

Řešením této situace je funkce *getAncestorVertex*. Ta využívá množinu navštívených vrcholů, která je vlastní pro každého mravence a je pojmenována Dorigem jako *tabu* a pomocnou množinu s názvem *closed*. Na následujícím obrázku je příklad situace, kdy se mravenec dostal do vrcholu číslo 3, ze kterého nevychází žádné další hrany, nebo jsou zablokovány a nelze z tohoto vrcholu pokračovat v cestě. Mravenec nejprve ukládá zablokovaný vrchol do množiny *closed* a vrací se na předposlední vrchol v *tabu*. Zde jsou zjištěny dostupné sousední vrcholy, po kterých lze pokračovat dál v cestě. V případě existence přechodového vrcholu či vrcholů je aktualizován stav mravence a jeho množina *tabu*. Výsledkem funkce je vrchol, do kterého se mravenec vrátil. Pokud takovéto přechodové vrcholy neexistují, pokračuje mravenec na další předchozí vrchol v *tabu*.



Obr. 38. Ukázka průběhu algoritmu *getAncestorVertex*

Výpis 6 Get Ancestor Vertex

```
V* getAncestorVertex(Ant<V> *ant)
{
      tabu = ant→getTabu();
      ant→appendClosed(ant→last);
      node = getNextToLastNode(tabu);

      while (node)
      {
            neighbours = getNeighbours(node, ant);

            if (neighbours.size())
            {
                  updateAnt(ant, node);
                  return node;
            }

            if (node == ant→startNode) resetAnt();

            ant→appendClosed(node);
            node = getNextToLastNode(tabu, node);
      }
}
```

Restarting *MAX—MIN* Ant System (ReMMAS)

Algoritmus *MaxMin Ant System* patří mezi jedny z nejlepších mravenčích algoritmů. Důležitou jeho vlastností je re-inicializace feromonové stopy při stagnaci nalezeného řešení, jak je uvedeno v popisu tohoto algoritmu v teoretické části. Při implementaci a testování byla navržena úprava, která využívá právě této stagnace a nejenže dochází k re-inicializaci feromonové stopy, ale i k náhodnému přenastavení hodnot parametrů α a β v další iteraci. Jak je patrné, hodnoty těchto parametrů jsou generovány z daného rozsahu, který vychází z experimentálního testování. Tato úprava je pojmenována jako *Restarting MaxMin Ant System (ReMMAS)*.

Výpis 7 Restarting MaxMin Ant System

```
void updatePheromoneTrails()
{
     globalEvaporation();
     depositMaxMinPheromones();

     checkPheromoneTrailLimits();

     if (stagnationIterations ≥ maxStagnationIterations)
     {
           initTrails();
           stagnationIterations = 0;
           lastReInitIteration = Iteration;

           updateAlphaBetaLevels();
     }
}

void updateAlphaBetaLevels()
{
     Alpha = rand.genRandInRange(minReAlpha, maxReAlpha);
     Beta  = rand.genRandInRange(minReBeta, maxReBeta);
}
```

Global Parameter Learning MaxMin Ant System (*GPL MAX—MIN* Ant System)

Madjid Khichane a kolektiv v článku [165] s názvem „*An ACO-based Reactive Framework for Ant Colony Optimization: First Experiments on Constraint Satisfaction Problems*“ publikovali algoritmus *AS*($\mathcal{GPL}$) neboli *Ant-Solver with Global Parameter Learning* pro řešení problému splňování podmínek. Tento algoritmus rozšiřuje původní verzi *Ant-Solver* o způsob dynamické adaptace parametrů α a β. *Global Parameter Learning MaxMin Ant System* je nový algoritmus adaptovaný na *MaxMin Ant System* algoritmus pro řešení problému kanadského cestujícího. Dále byla vytvořena i varianta pro základní algoritmus *Ant System* (*GPL – Ant System*). Algoritmus zavádí několik nových parametrů, například množiny I_α a I_β, které obsahují hodnoty parametrů α a β. Hodnoty v uvedených množinách je vhodné zvolit dle experimentálních výsledků na řešeném problému, ale lze je mít i zcela náhodné. Dalšími parametry jsou $\tau_\alpha(i)$, kde $i \in I_\alpha$ a $\tau_\beta(j)$, kde $j \in I_\beta$, jedná se o hodnoty kvality parametrů α a β při řešení problému. Autoři je označují jako sílu feromonové stopy jednotlivých parametrů. Pro hodnoty $\tau_\alpha(i)$ a $\tau_\beta(j)$ jsou zavedeny hranice minimální $\tau_{min_{\alpha\beta}}$ a maximální $\tau_{max_{\alpha\beta}}$ hodnoty. Na počátku hledání je jejich hodnota nastavena na $\tau_{max_{\alpha\beta}}$. Posledním parametrem je pak hodnota odpařování $\rho_{\alpha\beta}$ feromonové stopy. Algoritmus upravuje dvě části mravenčí metaheuristiky, inicializaci mravenců a aktualizaci feromonové stopy. K inicializaci mravenců, viz výpis 8, dochází v každé iteraci hledání řešení a algoritmus ji doplňuje o výběr parametru α z množiny $i \in I_\alpha$ a β z $j \in I_\beta$. Výběr je založen na pravděpodobnostní rovnici (6.3), která se inspiruje samotnou mravenčí metaheuristikou. Rovnice pro výpočet p_β je stejná, jen s parametry pro β hodnoty. Hodnota pravděpodobnosti pro zvolenou hodnotu α či β je proporcionální k jejich ohodnocení.

$$p_a = \frac{\tau_\alpha(i)}{\sum_{j \in I_\alpha} \tau_\alpha(j)} \tag{6.3}$$

Výpis 8 Global Parameter Learning MaxMin Ant System - ResetAnts

```
virtual void resetAnts() override
{
      for (ant : Ants)
            ant→resetAnt();

      alphaProbabilieties = computeProbability(alphaPheromones);
      id = 0;
      random = rand.getRand();
      probability = alphaProbabilieties[id];

      while (probability < random)
      {
            probability += alphaProbabilieties[ ++id];
      }
      Alpha = gplAlphaList[id];

      betaProbabilieties = computeProbability(betaPheromones);
      id = 0;
      random = rand.getRand();
      probability = betaProbabilieties[id];

      while (probability < random)
      {
            probability += betaProbabilieties[ ++id];
      }
      Beta = gplBetaList[id];
};
```

Druhou úpravou je pak funkce pro aktualizaci feromonové stopy. Jak je patrné z dalšího výpisu 9, byla tato část doplněna nejprve o odpaření feromonové stopy pro všechny hodnoty $\tau_\alpha(i)$ a $\tau_\beta(j)$. Následně jsou posíleny feromonové stopy pro aktuálně zvolené hodnoty parametrů α a β. Velikost posílení je proporcionální k nalezenému řešení. Pokud zvolené parametry α a β povedou k nalezení lepšího řešení, zvýší se i síla jejich feromonové stopy.

Výpis 9 Global Parameter Learning MaxMin Ant System - Pheromone update

```
virtual void updatePheromoneTrails() override
{
      globalEvaporation();
      depositMaxMinPheromones();
      checkPheromoneTrailLimits();

      if (stagnationIterations ≥ maxStagnationIterations)
      {
            reInitTrails();
            stagnationIterations = 0;
            lastReInitIteration  = Iteration;
      }

      // evaporace a aktualizace self-adapt feromonu
      for (i = 0; i < alphaPheromones.size(); ++i)
            alphaPheromones[i] *= (1 - gplEvaporationRate);

      for (i = 0; i < betaPheromones.size(); ++i)
            betaPheromones[i] *= (1 - gplEvaporationRate);

      // aktualizace feromonu
      depositGplPheromone();
}

void depositGplPheromone()
{
      alphaIdx = getParamIdx(Alpha, gplAlphaList);
      betaIdx  = getParamIdx(Beta, gplBetaList);

      deltaTau = Q / bestIterationResults.tourLength;

      alphaPheromones[alphaIdx] += deltaTau;
      betaPheromones[betaIdx] += deltaTau;

      // korekce hodnot feromonu
      checkParametersLimits(alphaIdx, alphaPheromones);
      checkParametersLimits(betaIdx, betaPheromones);
}
```

ACO+C

Mravenčí kastovní systém označovaný jako *ACO+C* je modifikace, u níž dochází k rozdělení mravenců na jednotlivé skupiny. Každá skupina má své vlastní nastavení parametrů α a β. Jak už bylo zmíněno v teoretické části, že jedna skupina, označovaná jako kasta, má více posílenou hodnotu parametru α a tím více klade důraz na ohodnocení hran. Druhá skupina naopak upřednostňuje feromonovou stopu a má tak větší hodnotu parametru β. Provedené úpravy mají dvě části, první je vytvoření jednotlivých mravenců a inicializace jejich parametrů podle předem zadaných kast, více ve výpisu 10.

Výpis 10 Init Ant Castes

```
void initAntsCastes(const std::vector<acoCastes>& castes)
{
     for (i = 0; i < castes.size(); ++i)
     {
          ants  = castes[i].numOfAntsInCaste;
          alpha = castes[i].alpha;
          beta  = castes[i].beta;

          for (j = 0; j < ants; ++j)
          {
               Ant<V>* ant = new AntCast<V>(Cities, startNode, alpha, beta);
               Ants.push_back(ant);
          }
     }
}
```

Další částí je samotný výpočet pravděpodobnosti přechodu v závislosti na aktuální hodnotě parametrů α a β daného mravence. Tato modifikace je dostupná pro všechny základní mravenčí algoritmy.

6.5 Mravenčí metody pro známou pravděpodobnost blokace

Pro řešení problému, kdy je dopředu známa pravděpodobnost blokace a pravděpodobnost obnovitelné hrany, bylo nutné k předchozím úpravám ještě změnit způsob hodnocení hran. A to do podoby střední doby trvání (6.1) jako je tomu u stochastické heuristiky *Dijkstra-SRCTP*. Podoba této úpravy je pak patrná na následujícím výpisu. Díky této úpravě lze použít mravenčí algoritmy i na řešení problému SRCTP.

Výpis 11 SRCTP Get Edge Weight

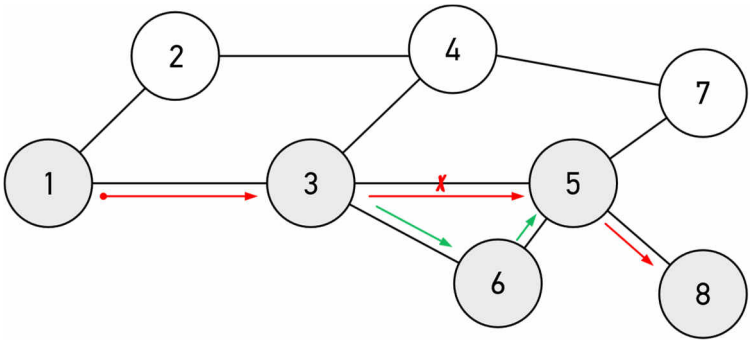
```
size_t getEdgeWeight(const E* edge)
{
      distance = 0;

      switch (edgeWeightingType)
      {
      case EDGE_WEIGHT:
      {
            if (ctpMode == CTP_MODE_K)
                  distance = edge→distance;
            else
                  distance = edge→blocked == true ? edge→weight + edge→penalty : edge→weight;
      }
      break;
      case EDGE_WEIGHT_SRCTP:
            distance = edge→weight + (edge→probability * edge→penalty);
            break;
      }

      return distance;
}
```

6.6 Verifikační strategie – porovnání mravenčích metod s heuristickými metodami

Pro porovnání efektivity heuristických a mravenčích algoritmů bylo nutné navrhnout způsob, který bude objektivizovat nalezená řešení. Jelikož výsledkem mravenčích algoritmů je mravenci nalezená nejkratší trasa ze startovního do cílového vrcholu, byl jako vhodný způsob zvolen proces simulace vícenásobného průchodu agenta nalezenou cestou. Pro tento účel objektivizace výsledků byly vy-tvořeny dva způsoby vyhodnocení, prvním je tzv. „cestovní agent“ a druhým „feromonový agent“. První způsob využívá mravenci nalezené nejkratší trasy a druhý způsob pak feromonovou stopu zanechanou mravenci na hranách grafu.

Cestovní agent

Ověřovací algoritmus pracuje následujícím způsobem. Na mravenci nalezenou trasu se vydávává cestovní agent. Stav jednotlivých hran je dopředu neznámý, dokud nedorazí do vrcholu, ze kterého hrana vychází. Pokud dojde v průběhu cesty agenta k zablokování některé z hran, je tato situace pro variantu k-CTP řešena pomocí Dijkstrova algoritmu. Ten nalezne nové spojení mezi aktuálním a některým z následujících vrcholů na mravenci nalezené trase. Pokud dojde k situaci, že jsou zablokované všechny vrcholy, vrací se agent na předchozí vrchol, ze kterého je nalezena cesta do dalšího vrcholu ležícího na mravenčí trase.

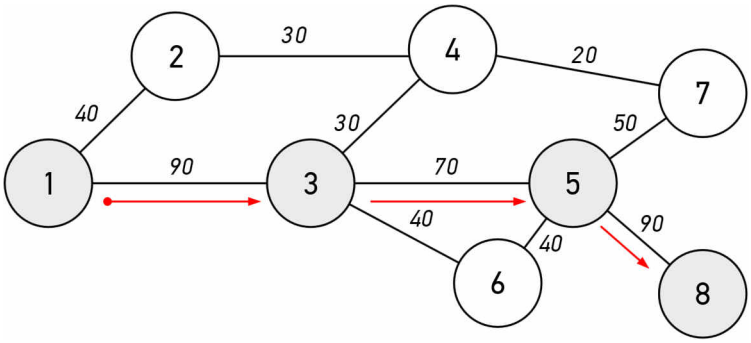


Obr. 39. Ukázka průběhu algoritmu Cestovní agent

V případě, že dojde k zablokování některé z hran u varianty *r-CTP,* je k ohodnocení připočtena časová penalizace a cestovní agent pokračuje dále po trase.

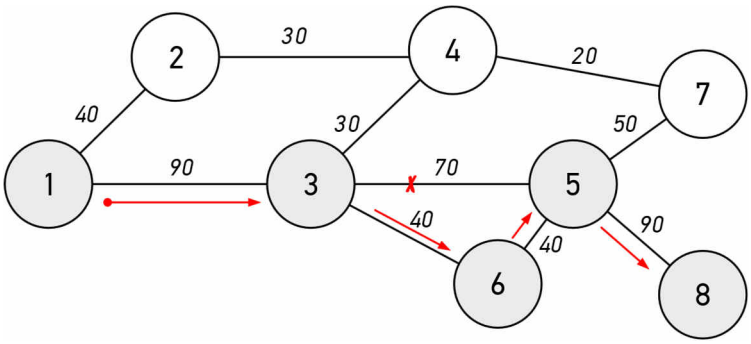
Feromonový agent

Druhý způsob ověřovacího agenta, který místo nalezené nejkratší mravenčí trasy spoléhá na feromo-novou stopu zanechanou mravenci na procházeném grafu. Agent si přechodovou hranu vybírá dle síly feromonové stopy a pokračuje po hraně s nejsilnější stopou. Průchod agenta je pak patrný na následujícím obrázku, kde na hranách je uvedena síla feromonové stopy. Tento způsob spoléhá na kolektivní řešení nalezené všemi mravenci.



Obr. 40. Ukázka průběhu algoritmu Feromonový agent

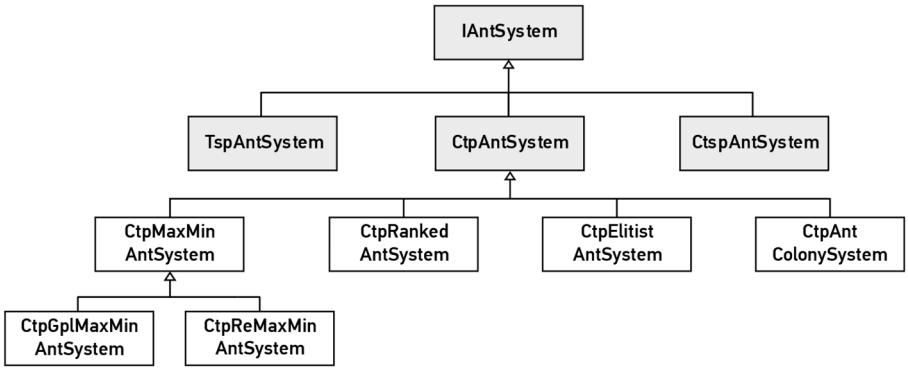
Pokud dojde při průchodu agenta grafem k zablokování některé z hran, je tato situace u varianty *k-CTP* řešena výběrem následující feromonově nejsilnější hrany. V případě varianty *r-CTP* je opět přičtena časová penalizace a agent pokračuje dále.



Obr. 41. Ukázka průběhu algoritmu Feromonový agent se zablokovanou hranou

7 VÝPOČETNÍ SOFTWARE PRO ŘEŠENÍ CTP

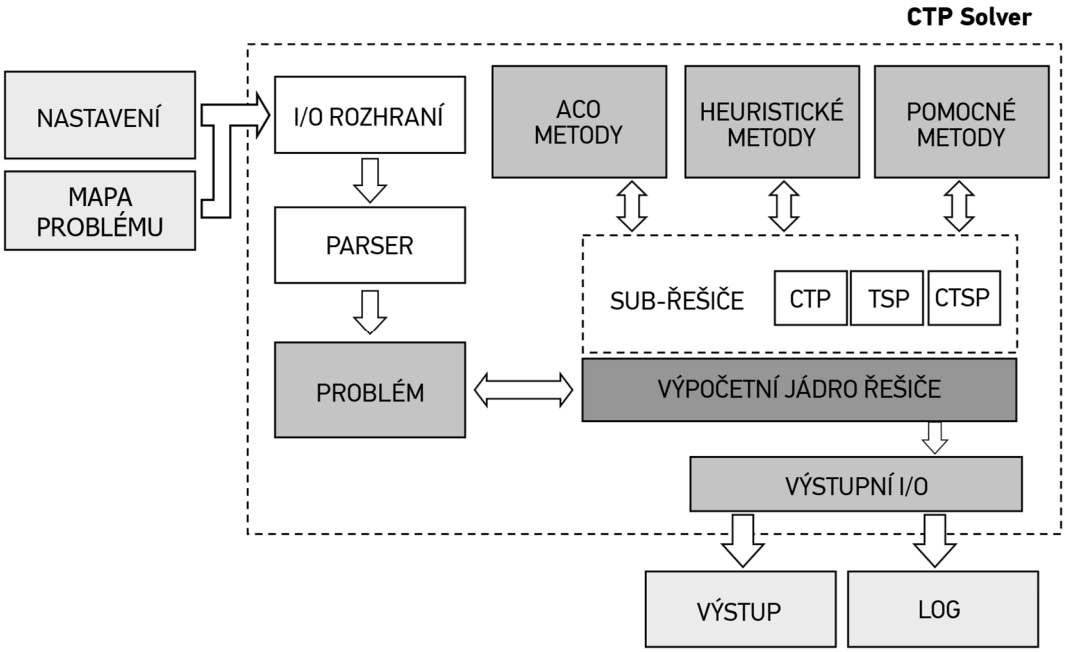
Tato kapitola je věnována vlastní implementaci řešiče, jeho částem a jejich popisu. Pro řešení problému kanadského cestujícího a dalších bylo vytvořeno vlastní softwarové řešení. Samotná implementace byla vytvořena v jazyce C++ 11 (a pozdější verze standardu) a je koncipována jako multiplatformní a modulární. Software byl vyvíjen pomocí objektového návrhu (*OOP*) při dodržení návrhových principů SOLID tak, aby bylo možné jednoduchým způsobem přidávat nové algoritmy pro řešení problémů a další funkcionalitu. Výpočetní software podporuje běh v příkazové řádce neboli CLI (*command line interface*) a také má vlastní grafické rozhraní pro pohodlnější ovládání a spouštění experimentů.



Obr. 42. Základní objektový návrh a vazba tříd pro řešení CTP pomocí ACO

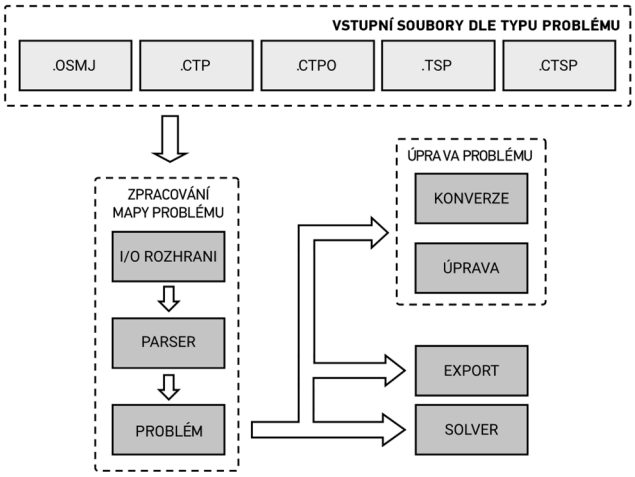
Koncepce

Vytvořenou aplikaci *CTP Solver* můžeme rozdělit do několika částí dle jejich funkcionality. Hlavní část tvoří společné výpočetní jádro a jednotlivé sub-řešiče (*solvery*). Výpočetní jádro se stará o spouštění paralelních výpočtů – testů a zajišťuje komunikaci s vybraným řešičem a pomocnými metodami. Řešič je rozdělen na několik částí. Ty tvoří sub-řešiče, které jsou rozděleny dle optimalizačního problému a dle typu výpočetní metody. Sub-řešiče implementují jednotlivé metody, ať už jde o mravenčí nebo heuristické algoritmy.



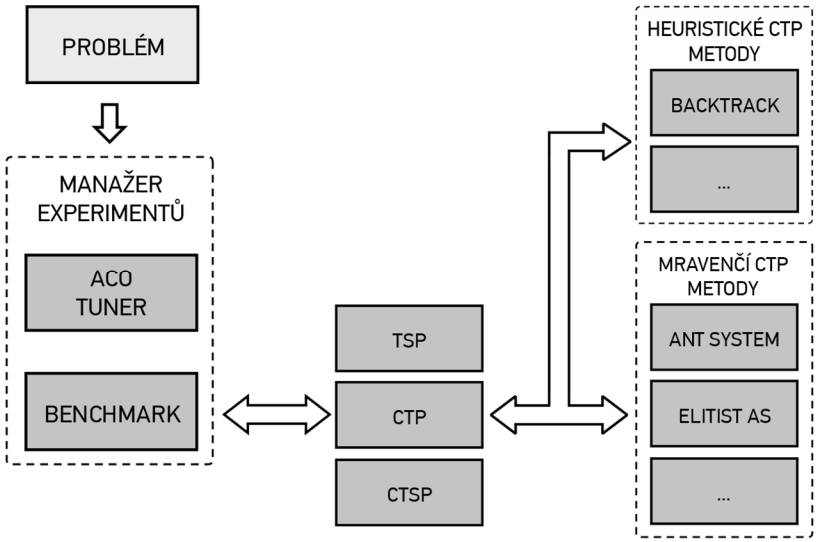
Obr. 43. Architektura CTP řešiče

Další částí je vstupní rozhraní, které se spolu s parserem stará o zpracování vstupních dat a jejich přípravu pro manažera experimentů a samotné výpočetní jádro. Vstupem do vytvořené aplikace je konfigurační soubor ve formátu JSON[8] obsahující nastavení a parametry pro jednotlivé výpočetní metody a jméno vstupního souboru obsahujícího data problému, viz. obr 44. Vstupní soubor problému popisuje podobu grafu problému a jeho vlastnosti, jako je rozložení hran mezi jednotlivými vrcholy grafu, jejich ohodnocení a pravděpodobnosti blokace. Pro problémy vycházející z reálných mapových podkladů je použit vlastní formát popisu mapy, opět ve formátu JSON. Podrobně je popisu vstupních souborů věnována jedna z následující podkapitol.



Obr. 44. Architektura zpracování vstupních souborů problému

Vytvořený software dále umožňuje generování a úpravu testovacích problémů. Také podporuje konverzi z formátu tsp pro problém obchodního cestujícího do formátů ctsp a ctp. Ve všech těchto variantách lze nastavovat řadu parametrů, ať jsou to pravděpodobnosti blokací jednotlivých hran, hodnoty penalizací a další. Spouštění experimentů je řešeno pomocí modulu manažera experimentů. Tento modul v závislosti na nastavení daného experimentu dynamicky spouští jednotlivé výpočetní metody a zpracovává výsledná data.



Obr. 45. Architektura spouštění experimentů

[8] JSON - JavaScript Object Notation

7.1 Paralelizace výpočetního jádra

Mravenčí algoritmy a obecně metaheuristiky se často používají na výpočetně náročné problémy. Z toho důvodu bylo nutné použít některý ze způsobů pro paralelizaci prováděného kódu. Koncepčně bylo na výběr mezi použitím více vláknové paralelizace (*multithreadingu*) na více jádrových procesorech (*multicore cpu*) a výpočtů na grafické kartě (*GPGPU*). Vzhledem k experimentální povaze úprav mravenčích algoritmů pro řešení problémů jako je CTP a CTSP bylo rozhodnuto o použití právě více vláknové paralelizace. V případě hejnových algoritmů, a specificky pak mravenčích algoritmů, jsou některé části koncipovány jako přirozeně paralelní. Jako příklad lze uvést průchod mravenců grafem. Na následujícím výpisu je ukázka běžné neparalelní podoby průchodu mravenců grafem. V takovém případě prochází mravenci grafem postupně a je pouze využíváno jedno jádro procesoru.

Výpis 12 Single Thread Code

```
for (size_t i = 0; i < m_AntsCount; ++i)
{
      while (!isAntFinished(m_Ants[i]))
      {
            selectNext(m_Ants[i]);
      }
}
```

Oproti tomu při využití více vláken lze dosáhnout paralelního průchodu mravenců grafem a dosáhnout využití více jader procesoru a zkrátit tak čas výpočtu. Výpis 13 obsahuje ukázku paralelní verze průchodu mravenců grafem řešenou pomocí vláken v moderní podobě jazyka C++.

Výpis 13 Modern C++ std::threads

```
std::vector<std::thread> threadPool;

auto threadFunc = ([&](size_t i)
{
      while (!isAntFinished(m_Ants[i]))
      {
            selectNext(m_Ants[i]);
      }
});

for (size_t i = 0; i < m_AntsCount; ++i)
{
      threadPool.push_back(std::thread(threadFunc, i));
}

for (auto &t : threadPool)
{
      if (t.joinable()) t.join();
}
```

Další možností je použití některé z knihoven pro rozšiřující jazyk C++ o paralelní konstrukce, jako je například Parallel Patterns Library (PPL) [166] nebo multiplatformní oneAPI Threading Building Blocks (TBB) [167]. Tyto knihovny obsahují konstrukce jako je například *parallel_for* nebo *parallel_invoke* a další, které zjednodušují tvorbu více vláknového kódu, jak je patrné z výpisu 14.

Výpis 14 Parallel for

```
parallel_for(0, m_AntsCount, [&](int i)
{
	while (!isAntFinished(m_Ants[i]))
	{
		selectNext(m_Ants[i]);
	}
});
```

Ve vytvořeném výpočetním software je použito několik přístupů, které lze použít v závislosti na cílové platformě, pro kterou bude kompilována. Pomocí podmíněné kompilace lze nastavit zda se použije paralelizace pomocí knihovny TBB nebo pomocí vláken *std::thread* ze standardní knihovny jazyka C++. Poznamenejme, že paralelizaci kódu lze také vypnout pro případnou analýzu a ladění navržených algoritmů.

7.2 Vstupní soubory

Vytvořená aplikace podporuje několik formátů vstupních dat. Ať už se jedná o vstupní data, která reprezentují vlastní podobu grafu, tak i podrobné nastavení pro řešič. Pro popis podoby grafu je podporovaných několik formátů. Jedná se o vlastní formáty *ctsp, ctp a ctp-old* vycházející z formátu *tsp* z knihovny problémů TSPLIB [73] pro řešení problému obchodního cestujícího. Tuto knihovnu tvoří textové soubory, které obsahují informace o mapě, např. název mapy, velikost grafu a samotná data reprezentující graf. Ta jsou zde uložena jako seznam hodnot a oddělují se pomocí klíčových slov. Pro řešení CTP problému byla navržena úprava, která rozšiřuje formát použitý v TSPLIB o informace jako je seznam sousedů pro vytvoření hran v grafu, pravděpodobnosti zablokování definovaných hran, penalizace a další. Tyto soubory mají koncovku *.ctp a .ctpo* a jednotlivá klíčová slova jsou uvedena v následující tabulce. Formát *ctp-old* je první verzí formátu pro popis CTP grafů a je považován za zastaralý.

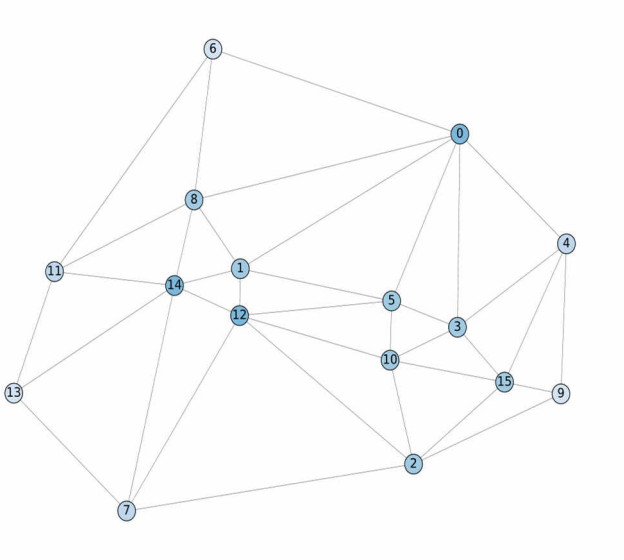
Dalším z vytvořených formátů vstupního souboru je *osmj*[9]. Jde o vlastní formát pro reprezentování grafu vycházejícího z reálných mapových podkladů projektu OpenStreetMap [168]. A to pomocí JSON formátu, který je široce využíván a je vhodný pro zápis dat organizovaných v objektech. Reálné mapové podklady jsou výrazně rozsáhlejší, než je tomu u map vycházejících z knihovny TSPLIB, proto bylo nutné zvolit vhodněji strukturovaný formát. Na obrázku 46 pro příklad formátu na bázi TSPLIB pro CTP je odstín pozadí vrcholu dán stupněm vrcholu.

Tab. 3 Formát vstupního souboru pro popis *ctp* mapy

Klíčové slovo	**Popis**
NAME	Název problému
TYPE	Typ problému
GRAPH	Typ grafu
DIMENSION	Počet vrcholů v grafu
PROBABILITY_COUNT	Procento hran s nenulovou pravděpodobnosti blokace
EDGE_WEIGHT_TYPE	Typ ohodnocení hran
NODE_COORD_SECTION	Seznam souřadnic vrcholů grafu
EDGE_DIRECTION_SECTION	Seznam sousedů
EDGE_WEIGHT_SECTION	Seznam ohodnocení hran
PROBABILITY_SECTION	Seznam pravděpodobnosti zablokovaní hran

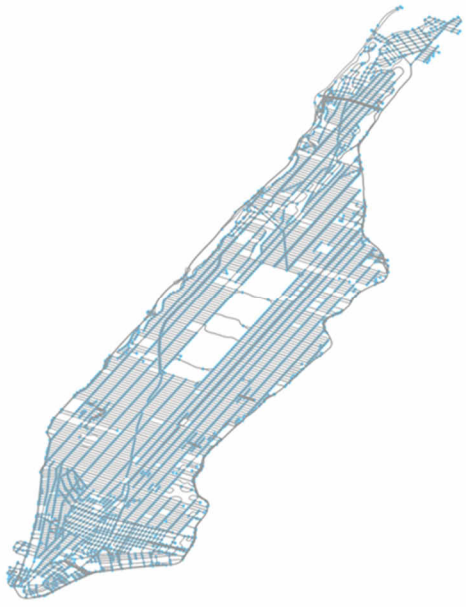
[9] OSMJ – Open Street Map in Json

```
NAME : ulysses16
COMMENT : Odyssey of Ulysses
COMMENT : Modified for CTP problem.
TYPE : CTP
GRAPH : UNDIRECTED
PROBABILITY_COUNT : 100
DIMENSION : 16
EDGE_WEIGHT_TYPE : GEO
NODE_COORD_SECTION
1 38.24 20.42
2 39.57 26.15
3 40.56 25.32
…
EDGE_DIRECTION_SECTION
2 3 4 5 6 7
1 5 6 8 10
1 4 5 9
…
PROBABILITY_SECTION
0.28 1.0 0.25 0.87 1.0 0.69
1.0 0.3 0.52 1.0 1.0
1.0 0.23 1.0 0.96
…
PENALTY_SECTION
8 10 10 8 2 8
8 8 6 2 3
10 8 8 10
EOF
```



Obr. 46. Příklad formátu na bázi TSPLIB pro CTP

```
{
  "Name": "manhattan",
  "Filename": "manhattan.osmj",
  "Directed": false,
  "ProbabilityCount": 0,
  "Dimension": 4486,
  "Nodes": [
    {
      "id": 0,
      "uid": 1773060097,
      "lat": 40.7140611,
      "lon": -73.9975944,
      "edges": [
        {
          "uid": 42437559,
          "length": 11.237,
          "penalty": 0,
          "probability": 0
        },
        {
          "uid": 1773060099,
          "length": 51.557,
          "penalty": 0,
          "probability": 0
        }
      ]
    }, {…},
}
```



Obr. 47. Příklad formátu na bázi formátu OSMJ pro reálné mapové podklady

Nastavení řešiče je uloženo v konfiguračním souboru, jehož podoba je opět ve formátu JSON (příklad lze vidět na následujícím obrázku). V tomto souboru jsou uloženy podrobnosti nutné pro provedení experimentu. Je zde uveden název souboru s mapovými podklady, počet opakování experimentu, startovní a cílový vrchol, vybrané řešící metody, specifické parametry mravenčích metod a také nastavení parametrů pro CTP problém.

```
{
        "Type": "Tour",
        "File": "maps\\delunai-50.ctp",
        "Repeats": 100,
        "AgentRepeats": 200,
        "StartNode": 0,
        "EndNode": -1,

        "Heuristic-Algorithms": [
                "DJK-BACKTRACK",
                "DJK-GREEDY",
                "DJK-COMPARSION",
                "DJK-REC-BACKTRACK",
                "DJK-REC-GREEDY",
                "DJK-REC-GREEDY-WP",
                "DJK-REC-COMPARSION"
                "DJK-REC-COMPARSION-WP",
                "DJK-WAITING",
        ],

        "AcoA-lgorithms": [
                "KCTP-AS",
                "KCTP-EAS",
                "KCTP-RAS",
                "KCTP-MMAS",
                "KCTP-REMMAS",
                "RCTP-AS",
                "RCTP-EAS",
                "RCTP-RAS",
                "RCTP-MMAS",
                "RCTP-REMMAS",
        ],

        "Ants": 300,
        "Iterations": 200,

        "Alpha": 0.8,
        "Beta": 1.8,
        "Rho": 0.7,
        ...
```

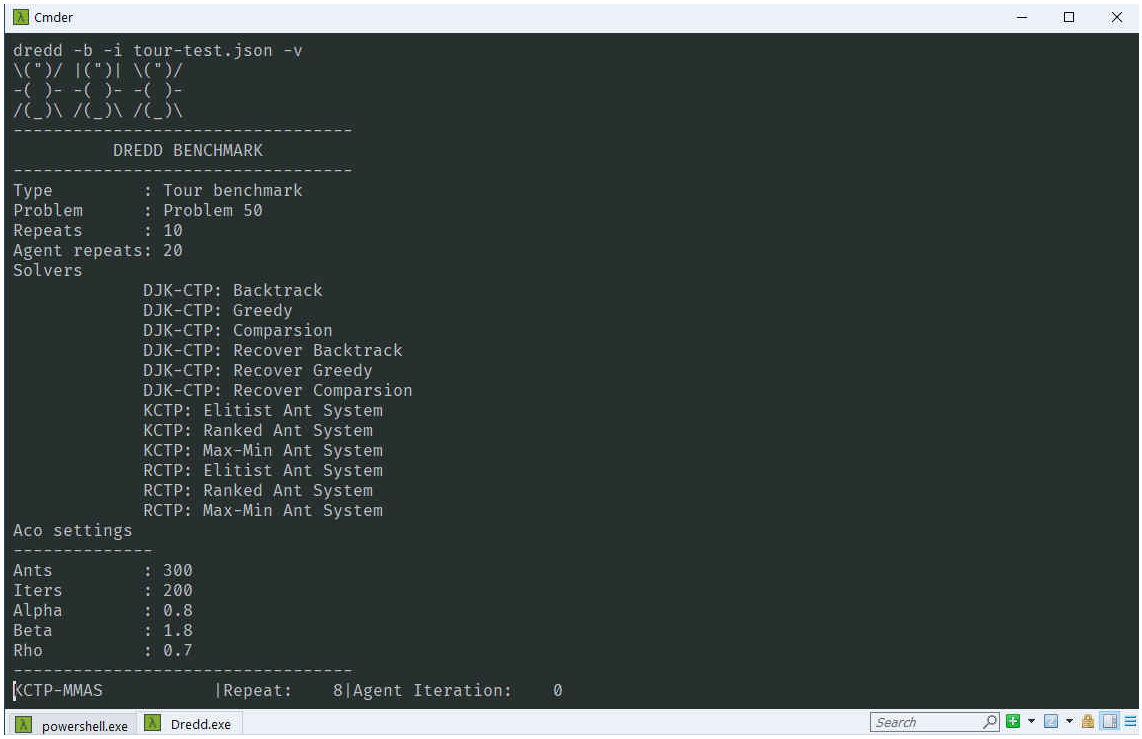
```
...
        "RandomAntPosition": false,
        "ElitistCoef": 200,
        "RankedAnts": 10,
        "Q": 1,
        "Q0": 0.5,
        "Xi": 0.1,
        "MaxStagnationCount": 5,

        "UseCastes": false,
        "Castes": [
                {
                        "Ants": 100,
                        "Alpha": 5,
                        "Beta": 1
                },
                {
                        "Ants": 100,
                        "Alpha": 1,
                        "Beta": 6
                },
                {
                        "Ants": 100,
                        "Alpha": 2,
                        "Beta": 9
                }
        ],

        "OutDir": "Tests",
        "Verbose": true,
        "Log": false,
        "LogDir": "Tests",

        "GenBlockades": true,
        "GenProbabilities": 0,
        "MinProbabilities": 0.0,
        "MaxProbabilities": 0.6,
        "StaticProbabilities": 0.5,
        "UseDefaultInitPheromone": false,
        "EdgeWeighting": 0,
        "AgentMethod": 2,
```

Obr. 48. Příklad formátu s parametry pro řešič

Výsledky aplikace jsou vypočtená data, která se ukládají do výstupního souboru, opět ve formátu JSON. Tento soubor pak obsahuje jak nastavení experimentu, tak podrobné výsledky z každého jeho opakování. Dále lze uložit logovací soubor pro možnou pokročilou analýzu funkčnosti vytvořené aplikace.

7.3 Konzolová aplikace

Vlastní aplikace je rozdělena do konzolové aplikace a aplikace s grafickým rozhraním. Jednou z výhod konzolové aplikace je možnost jejího spouštění přímo pomocí skriptů, které nastavují aplikaci parametrizovaným způsobem a jsou schopny vypočtená data dále zpracovávat. Další výhodou je možnost provozu na systémech, které nemají grafické rozhraní, jako je například výpočetní cluster.



Obr. 49. Ukázka běžícího testu v rozhraní konzolové aplikace

Aplikace umožňuje nastavovat veškeré operace pomocí parametrů, ať už se jedná o samotné spuštění výpočtových testů, generování map a další operace. Podoba vstupních parametrů aplikace pro běh experimentu je uvedena na následujícím obrázku.

```
Options:

  -h, --help            Show this help message and exit

  -i FILE, --input=FILE

                        Read problem source from FILE

  -o FILE, --output=FILE

                        Save to FILE

  -v, --verbose         Set verbose mode, default = 0


  Start benchmark using setting file...:

    -b, --benchmark     Run benchmark
```

Obr. 50. Základní parametry pro spuštění testu

Mimo spouštění řešiče, a tedy provedení experimentu, lze pomocí vytvořené aplikace provádět i další úkony. Jedním z nich je schopnost generovat mapové podklady. Jak je patrné z obrázku 51, generování map má celou řadu volitelných parametrů. Generátor map umožňuje vygenerovat mapy jak v novém (*.ctp*), tak i starším (*.ctpo*) formátu, přičemž starší formát již není doporučován. Lze nastavit jednotlivé parametry, jako je počet vrcholů, procentuální hodnota počtu hran, které budou mít nenulovou hodnotu pravděpodobnosti blokace, včetně rozsahu této pravděpodobnosti. Dalšími parametry jsou ohodnocení hran včetně velikosti penalizace, kde lze opět nastavit rozsah těchto hodnot a v případě penalizací i procentuální hodnotu z ohodnocení hran.

```
Options:

Generate problem source file:

    -g, --generate      Generate problem source file

    --old               Generate CTP-O problem

    --new               Generate CTP-N problem

    -d, --directed      Create directed graph

    --dimension=INT     Problem dimension, default = 50

    -p INT, --block-percent=INT

                        Set % of blockable edges, default = 75

    --min-weight=INT    Minimum edge weight, default = 10

    --max-weight=INT    Maximum edge weight, default = 50

    --min-probability=FLOAT

                        Block probability minimum, default = 0.1

    --max-probability=FLOAT

                        Block probability maximum, default = 1

    --min-penalty=INT   Minimum edge penalty, default = 1

    --max-penalty=INT   Maximum edge penalty, default = 10

    --penalty-percent=INT

                        Use edge distance as edge penalty

    --min-penalty=INT   Minimum edge penalty, default = 1

    --max-penalty=INT   Maximum edge penalty, default = 10

    --min-edges=INT     Minimum edges from node, default = 3

    --max-edges=INT     Maximum edges from node, default = 10

    --max-nearest=INT   Maximum nearest nodes to select, default = 15

    --triangulate       Use Delaunay triangulation

    --map-width=INT     Maximum map width.

    --map-height=INT    Maximum map heigth.
```

Obr. 51. Parametry pro generování map

Mezi další parametry patří rozsah hodnot pro hodnotu stupně vrcholu. Způsob rozložení vrcholů lze zvolit buď náhodný, nebo vytvořený pomocí Delaunyho triangulace. Posledním parametrem je pak velikost mapy. Generování map není jedinou možností tvorby mapových podkladů, nové mapy lze vytvořit i konverzí map z knihovny TSPLIB resp. mapových podkladů v tomto formátu. U vytvořených map lze také měnit nastavení parametrů hran v grafu. Tímto způsobem můžeme měnit procento hran s nenulovou pravděpodobností blokace. Další funkcionalitou je možnost exportování vygenerovaných map do formátu *dot* a *gml* pro jejich následnou vizualizaci.

```
Options:
  Convert TSP to CTP/CTSP problem source file:
    -c, --convert       Convert TSP problem to CTP/CTSP
    --ctp               Convert to CTP
    --ctsp              Convert problem to CTSP
    -d, --directed      Create directed graph
    -p INT, --block-percent=INT
                        Set % of blockable edges, default = 75
    --min-penalty=INT   Minimum edge penalty, default = 1
    --max-penalty=INT   Maximum edge penalty, default = 10
    --penalty-percent=INT
                        Use edge distance as edge penalty
    --min-penalty=INT   Minimum edge penalty, default = 1
    --max-penalty=INT   Maximum edge penalty, default = 10
    --min-edges=INT     Minimum edges from node, default = 3
    --max-edges=INT     Maximum edges from node, default = 10
    --max-nearest=INT   Maximum nearest nodes to select, default = 15
```

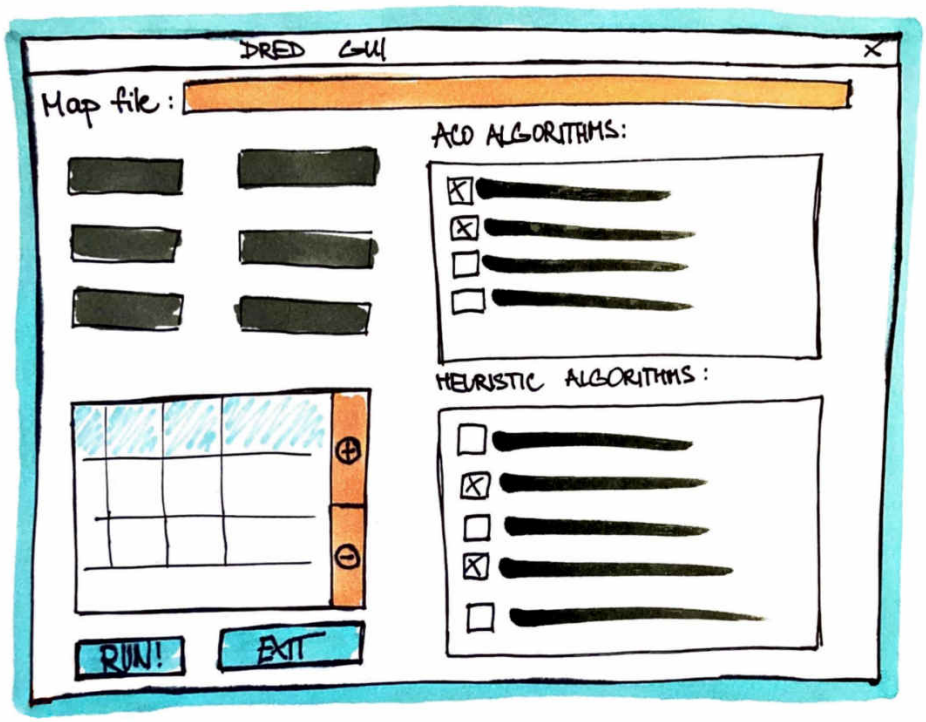
Obr. 52. Parametry pro konverzi map

```
Options:
Adjust CTSP/CTP/CTP-O problem source file:
    -a, --adjust        Adjust problem parameters.

    -r, --replace       Replace problem parameters.

    -p INT, --block-percent=INT

                        Set % of blockable edges

    --min-probability=FLOAT

                        Block probability minimum, default = 0

    --max-probability=FLOAT

                        Block probability maximum, default = 1

    --min-penalty=INT   Minimum edge penalty, default = 1

    --max-penalty=INT   Maximum edge penalty, default = 10

    --penalty-percent=INT

                        Use % of edge weight as edge penalty
```

Obr. 53. Parametry pro úpravu nastavení map

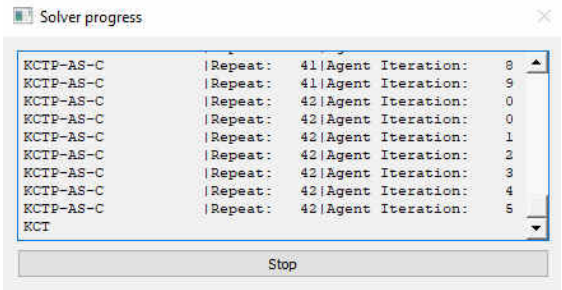
7.4 Grafické rozhraní aplikace

Pro uživatelsky pohodlné a přívětivé ovládání bylo vytvořeno grafické rozhraní. To umožňuje jednoduše nastavovat parametry pro spuštění experimentů. Rozhraní bylo vytvořeno pomocí jazyka Python ve verzi 3 a grafického toolkitu *PyQT* ve verzi 5. Rozložení grafické aplikace bylo vytvářeno pro zajištění jednoduchého a funkčního ovládání a vychází z předběžného návrhu, jak je patrné z následujícího obrázku.

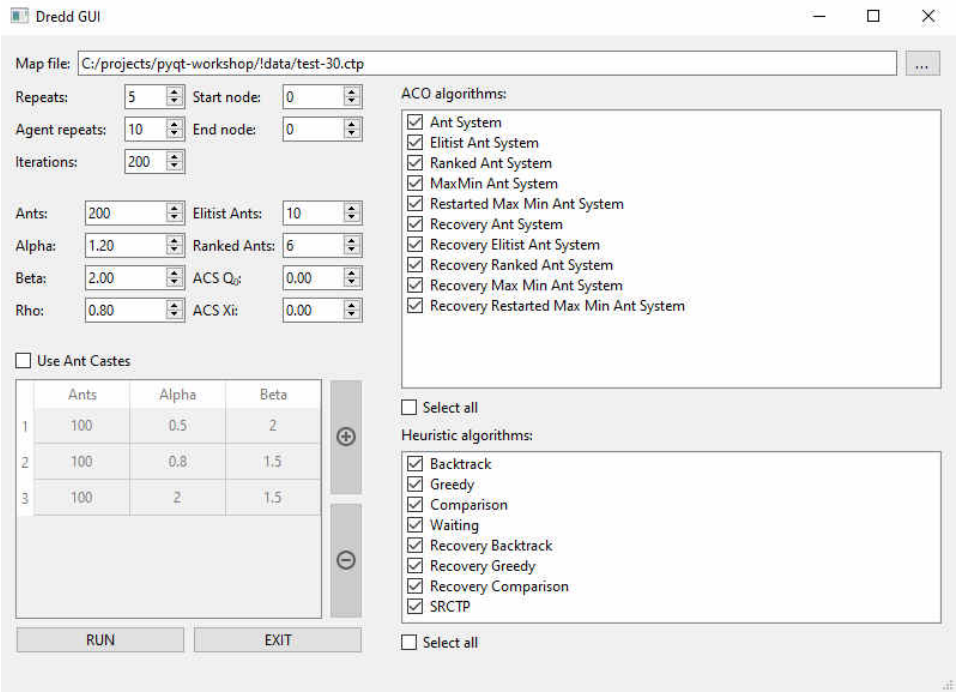


Obr. 54. Ideový návrh (skica) grafického rozhraní

Konečná podoba grafického rozhraní pak podporuje výběr mapy problému a umožňuje nastavovat nejdůležitější parametry pro spuštění experimentu, ať se jedná o výběr konkrétních metod nebo o jednotlivé parametry. Jde například o počet opakování experimentu, počty mravenců, parametry α, β, ρ a další. Lze také nastavit specifické parametry pro konkrétní mravenčí algoritmy, například počet elitních mravenců pro algoritmus *Elitist Ant System* nebo počet hodnotících mravenců u *Ranked Ant System* algoritmu. V neposlední řadě může uživatel elegantně měnit počet a nastavení mravenčích kast, tedy skupin mravenců s různorodým nastavením jejich počtu a parametrů α, β.

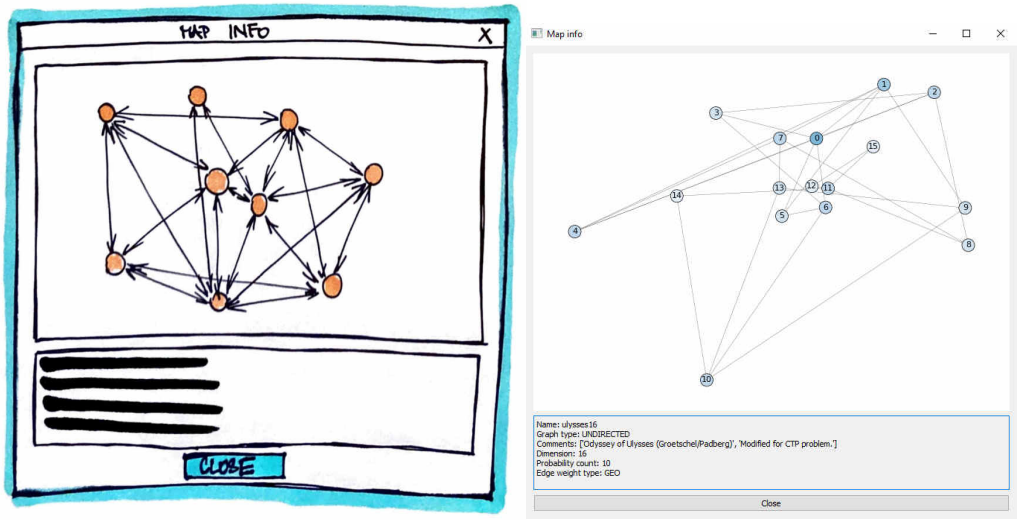


Obr. 55. Grafické rozhraní aplikace – spuštěný experiment



Obr. 56. Grafické rozhraní aplikace

Po provedení výběru konkrétní mapy problému se uživateli zobrazí okno s podrobnostmi a vizualizací zvoleného problému. Z této části grafického rozhraní vznikla později i samostatná aplikace *Map Info*, která slouží pouze k zobrazování informací o mapě. Po nastavení vybraných parametrů se provede spuštění experimentu zmáčknutím tlačítka „*Run*“.



Obr. 57. Ideový návrh a realizace grafického rozhraní s informacemi o mapě problému

8 EXPERIMENTY

V této kapitole byla provedena sada experimentů navrhovaných řešení CTP problémů s neznámou a známou pravděpodobností blokace na úlohách z vytvořeného testovacího korpusu. Ten tvoří řada map s různorodým počtem vrcholů. Dále byly použity reálné mapy měst Piedmont, Manhattan z projektu OpenStreetMap, což podtrhuje praktickou využitelnost prezentovaného řešení. Cílem experimentů bylo porovnat výsledky dosažené pomocí mravenčích algoritmů a s hodnotami zjištěnými heuristickými metodami. Experimenty byly opakovány stokrát pro zajištění lepší objektivizace výsledků.

V rámci řešení problému kanadského cestujícího vznikla knihovna map problémů pojmenovaná *CTPlib*, která je inspirovaná již zmiňovanou knihovnou TSPLIB. Vytvořená knihovna bude zveřejněna a dostupná pro zájemce o řešení problému CTP.

Výsledky uvedených algoritmů (modifikované heuristiky a mravenčí algoritmy) jsou obecně silně závislé na definici daného problému. Silnější statistické závěry nejsou vzhledem k povaze dat možné. Pravděpodobnostní rozdělení nemají povahu normálního rozdělení a jsou diskrétní, navíc s velkou četností stejných hodnot. Z tohoto důvodu se upustilo od neparametrických variant ANOVA testu typu Kruskal-Wallis a zvolila se metoda hodnocení založená na robustních charakteristikách mediánu a kvartilů. Z dále uvedených výsledků pro diskutované algoritmy byly většinově vyňaty ty algoritmy, které neměly vůči ostatním ani průnik v mezikvartilovém rozpětí, jinak řečeno, po vizuální stránce neměly odpovídající box-grafy ani průnik. Tam kde jsou box-grafy v průniku, považujeme výsledky za obdobné. Z hlediska nejlepšího výsledku uvažujeme medián, případně mezikvartilové rozpětí jako příznak robustnosti algoritmu vzhledem k menšímu rozptylu hodnot.

V průběhu vývoje výpočetního software byly experimentálně zjištěny následující hodnoty pro nastavení parametrů mravenčích algoritmů, a to $\alpha = 1{,}8;\ \beta = 3{,}8;\ \rho = 0{,}7$. Tyto hodnoty byly použity jako základní hodnoty parametrů pro následující experimenty. Algoritmy vybrané pro experimenty jsou následující:

Heuristické algoritmy	**Mravenčí (ACO) algoritmy**
• Backtrack	• Ant System
• Greedy	• Ranked Ant System
• Comparison	• Elistist Ant System
• Comparison With Penalization	• MaxMin Ant System
• Recovery Backtrack	• Restarted MaxMin Ant System
• Recovery Greedy	• Global Parameter Learning Ant System
• Recovery Comparison	• Global Parameter Learning MaxMin Ant System
• Recovery Comparison With Penalization	

8.1 Neznámá pravděpodobnost blokace

8.1.1 *k*-CTP syntetické mapy

Nejprve byly provedeny experimenty na variantě *k-CTP*. V tomto případě, když dojde k zablokování hrany, je *hrana zablokována navždy a nejde ji obnovit*. Experimenty byly provedeny na mapách delaunay-50, test-100, test-350.

Mapa delaunay-50

Tab. 4 Parametry experimentu pro mapu delaunay-50

Parametry Experimentu	Hodnota
Vrcholů	50
Opakování experimentu	100
Průchodů agenta	100
Mravenců	200
Iterací	200
α	1,8
β	3,8
ρ	0,7

Mapa test-100

Tab. 5 Parametry experimentu pro mapu test-100

Parametry Experimentu	Hodnota
Vrcholů	100
Opakování experimentu	100
Průchodů agenta	100
Mravenců	300
Iterací	400
α	1,8
β	3,8
ρ	0,7

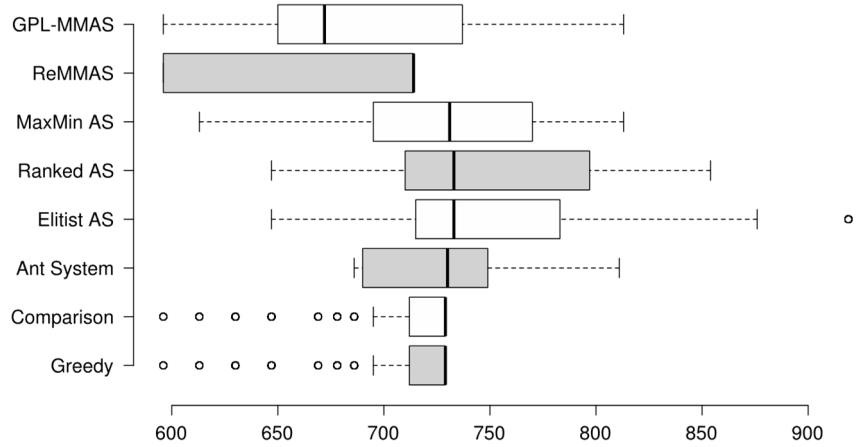
Mapa test-350

Tab. 6 Parametry experimentu pro mapu test-350

Parametry Experimentu	Hodnota
Vrcholů	350
Opakování experimentu	100
Průchodů agenta	100
Mravenců	1000
Iterací	1000
α	1,8
β	3,8
ρ	0,7

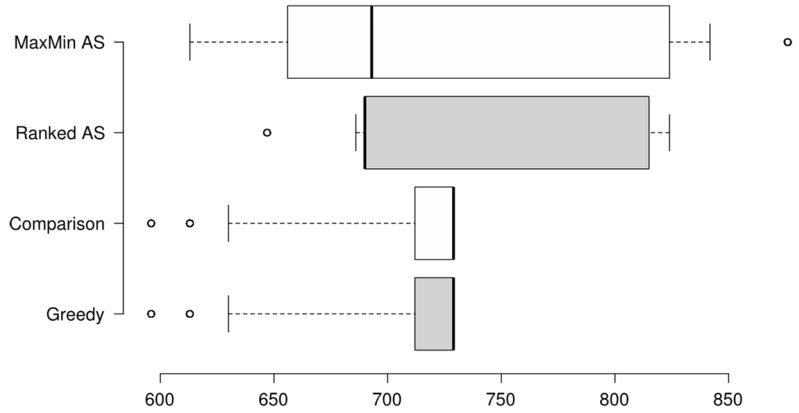
Mapa delaunay-50

Pravděpodobnost blokací 15 %



	Greedy	Comp.	AS	EAS	RAS	MMAS	ReMMAS	GPLMMAS
3 kvartil	729	729	749	783	797	770	714	737
Medián	729	729	730	733	733	731	714	672
1 kvartil	712	712	690	715	710	695	596	650

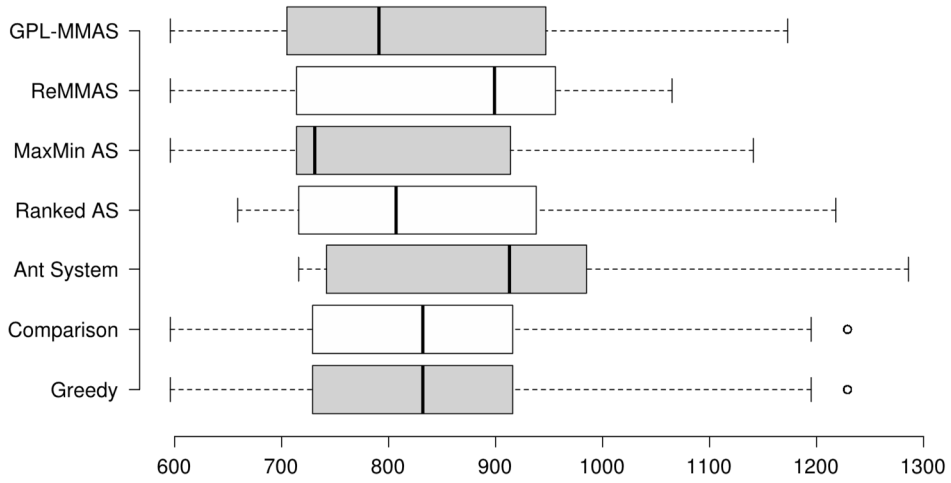
Obr. 58. Výsledky pro cestovního agenta, delaunay-50 s 15 % prav. blokací



	Greedy	Comp.	RAS	MMAS
3 kvartil	729	729	815	824
Medián	729	729	690	693
1 kvartil	712	712	690	656

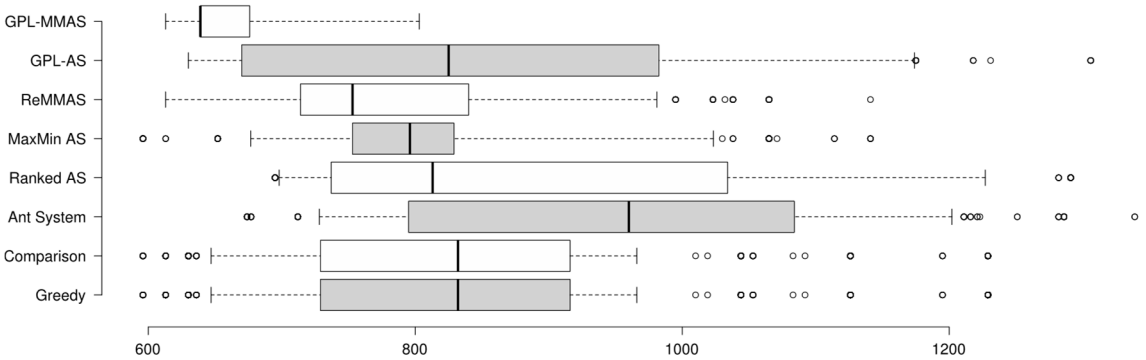
Obr. 59. Výsledky pro feromonového agenta, delaunay-50 s 15 % prav. blokací

Pravděpodobnost blokací 30 %



	Greedy	Comp	AS	RAS	MMAS	ReMMAS	GPL-MMAS
3 kvartil	916	916	985	938	914	956	947
Medián	832	832	913	807	731	899	791
1 kvartil	729	729	742	716	714	714	705

Obr. 60. Výsledky pro cestovního agenta, delaunay-50 s 30 % prav. blokací

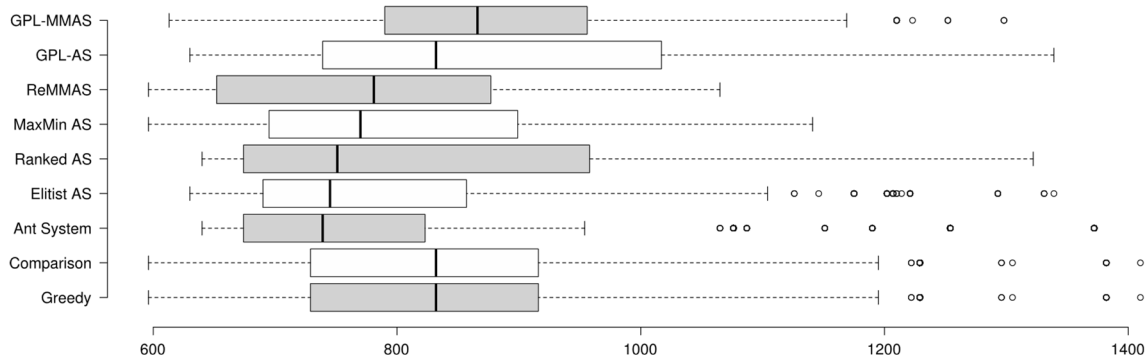


	Greedy	Comparison	AS	RAS
3 kvartil	916	916	1084	1034
Medián	832	832	960	813
1 kvartil	729	729	795	737

	MMAS	ReMMAS	GPL-AS	GPL-MMAS
3 kvartil	829	840	982	676
Medián	796	753	825	639
1 kvartil	753	714	670	639

Obr. 61. Výsledky pro feromonového agenta, delaunay-50 s 30 % prav. blokací

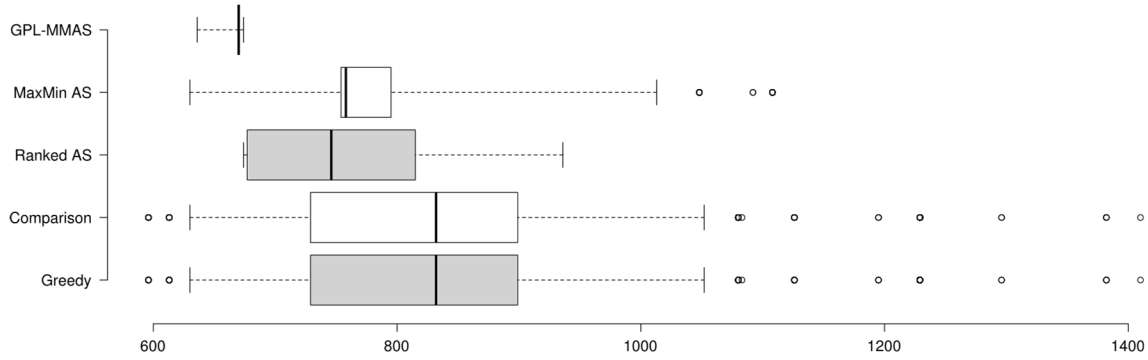
Pravděpodobnost blokací 45 %



	Greedy	Comp.	AS	EAS
3 kvartil	916	916	823	857
Medián	832	832	739	745
1 kvartil	729	729	674	690

	RAS	MMAS	ReMMAS	GPL-AS	GPL-MMAS
3 kvartil	958	899	877	1017	956
Medián	751	770	781	832	866
1 kvartil	674	695	652	739	790

Obr. 62. Výsledky pro cestovního agenta, delaunay-50 s 45 % prav. blokací



	Greedy	Comparison	RAS	MMAS	GPL-MMAS
3 kvartil	899	899	815	795	670
Medián	832	832	746	758	670
1 kvartil	729	729	677	754	670

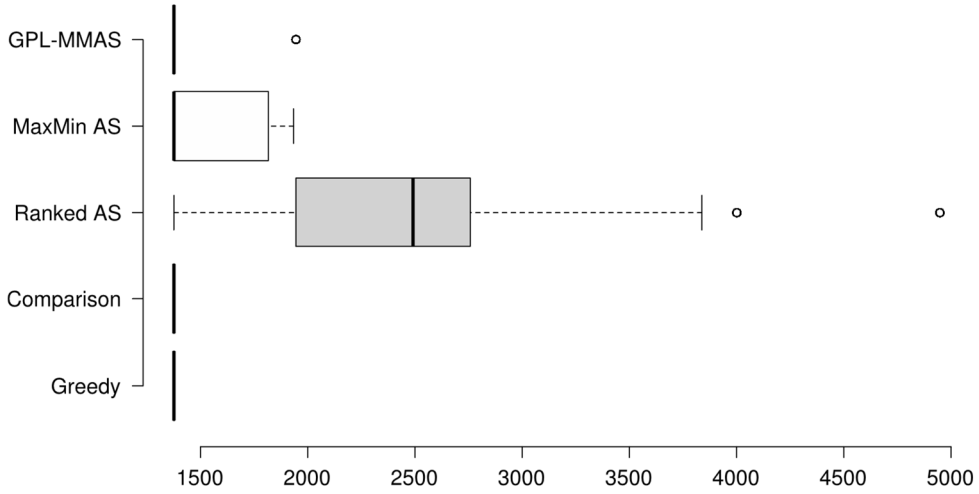
Obr. 63. Výsledky pro feromonového agenta, delaunay-50 s 45 % prav. blokací

Mapa test-100

Pravděpodobnost blokací 15 %

	Greedy	Comparison	MMAS	ReMMAS	GPL-MMAS
3 kvartil	1376	1376	1376	1376	1376
Medián	1376	1376	1376	1376	1376
1 kvartil	1376	1376	1376	1376	1376

Obr. 64. Výsledky pro cestovního agenta, test-100 s 15 % prav. blokací



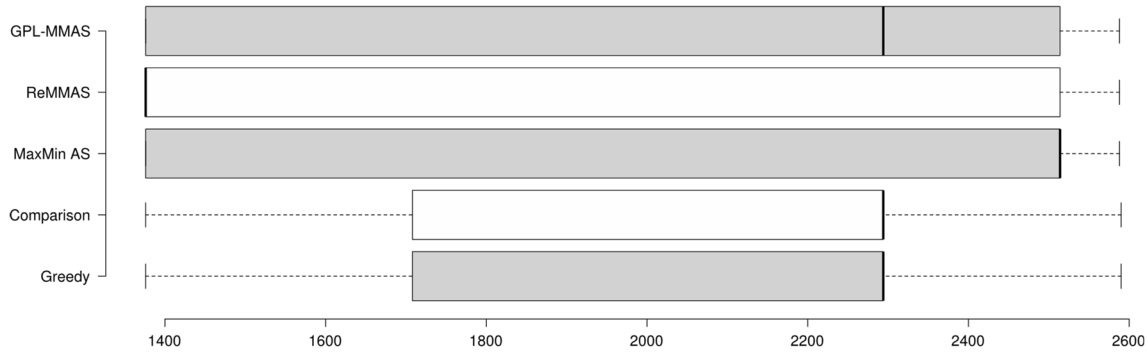
	Greedy	Comparison	RAS	MMAS	GPL-MMAS
3 kvartil	1376	1376	2758	1816	1376
Medián	1376	1376	2491	1376	1376
1 kvartil	1376	1376	1945	1376	1376

Obr. 65. Výsledky pro feromonového agenta, test-100 s 15 % prav. blokací

	MMAS	GPL-MMAS
3 kvartil	1376	1376
Medián	1376	1376
1 kvartil	1376	1376

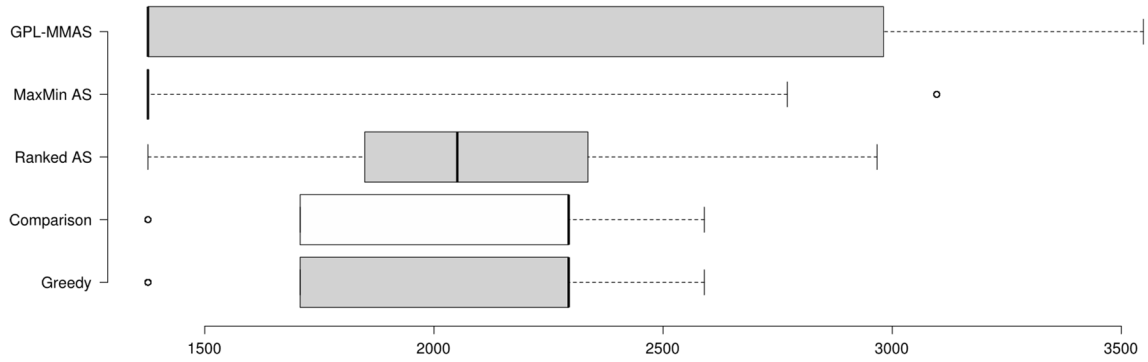
Obr. 66. Výsledky pro feromonového agenta s počtem mravenců 800, test-100 s 15 % prav. blokací

Pravděpodobnost blokací 30 %



	Greedy	Comparison	MMAS	ReMMAS	GPL-MMAS
3 kvartil	2294	2294	2514	2514	2514
Medián	2294	2294	2514	1376	2294
1 kvartil	1708	1708	1376	1376	1376

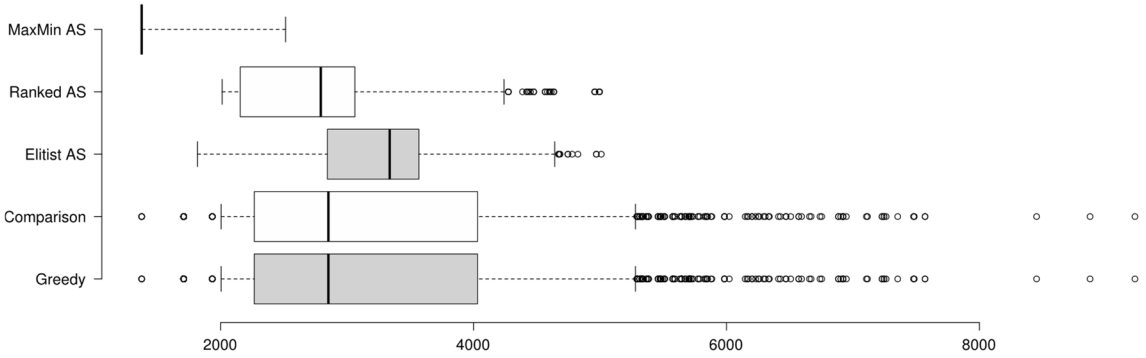
Obr. 67. Výsledky pro cestovního agenta, test-100 s 30 % prav. blokací



	Greedy	Comparison	RAS	MMAS	GPL-MMAS
3 kvartil	2294	2294	2336	1376	2981
Medián	2294	2294	2051	1376	1376
1 kvartil	1708	1708	1849	1376	1376

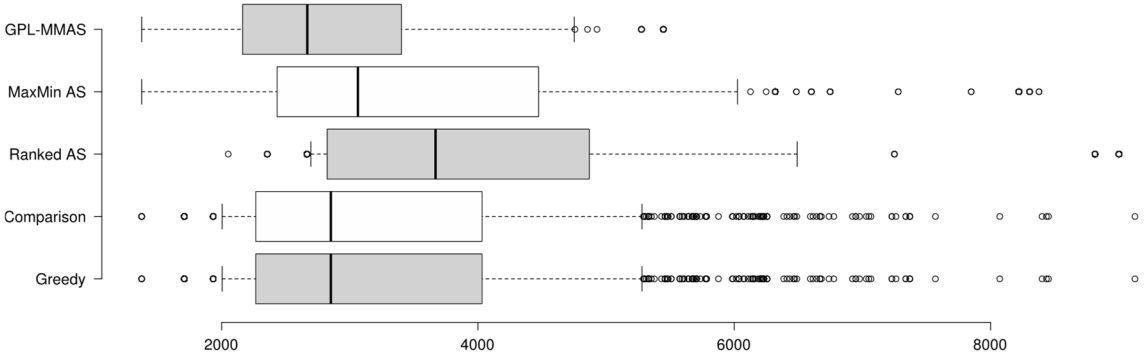
Obr. 68. Výsledky pro feromonového agenta, test-100 s 30 % prav. blokací

Pravděpodobnost blokací 45 %



	Greedy	Comparison	EAS	RAS	MMAS
3 kvartil	4032	4032	3568	3062	1376
Medián	2852	2852	3337	2792	1376
1 kvartil	2266	2266	2844	2155	1376

Obr. 69. Výsledky pro cestovního agenta, test-100 s 45 % prav. blokací

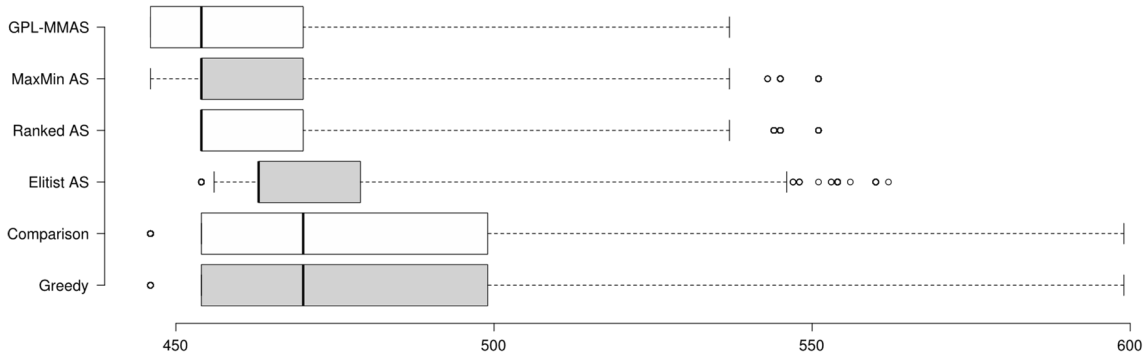


	Greedy	Comparison	RAS	MMAS	GPL-MMAS
3 kvartil	4032	4032	4869	4473.50	3403
Medián	2852	2852	3669	3063	2668
1 kvartil	2266	2266	2823	2432.50	2164

Obr. 70. Výsledky pro feromonového agenta, test-100 s 45 % prav. blokací

Mapa test-350

Pravděpodobnost blokací 30 %



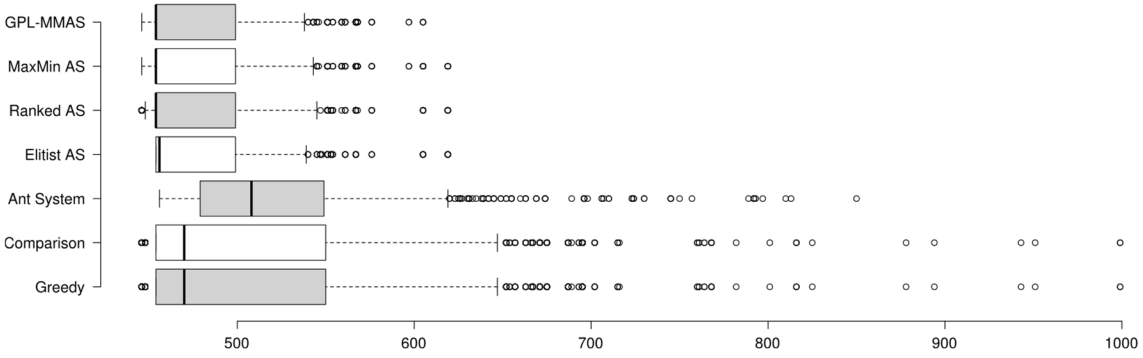
	Greedy	Comparison	EAS	RAS	MMAS	GPL-MMAS
3 kvartil	499	499	479	470	470	470
Medián	470	470	463	454	454	454
1 kvartil	454	454	463	454	454	446

Obr. 71. Výsledky pro cestovního agenta, test-350 s 30 % prav. blokací

	Greedy	Comparison	MMAS	GPL-MMAS
3 kvartil	446	446	449	446
Medián	446	446	449	446
1 kvartil	446	446	449	446

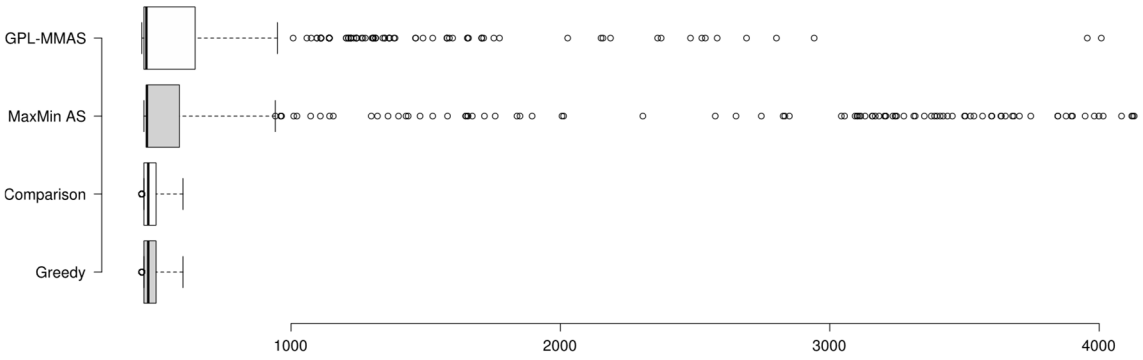
Obr. 72. Výsledky pro feromonového agenta, test-350 s 30 % prav. blokací

Pravděpodobnost blokací 45 %



	Greedy	Comparison	AS	EAS	RAS	MMAS	GPL-MMAS
3 kvartil	550	550	549	499	499	499	499
Medián	470	470	508	456	454	454	454
1 kvartil	454	454	479	454	454	454	454

Obr. 73. Výsledky pro cestovního agenta, test-350 s 45 % prav. blokací



	Greedy	Comparison	MMAS	GPL-MMAS
3 kvartil	499	499	586	644
Medián	470	470	465	463
1 kvartil	454	454	463	454

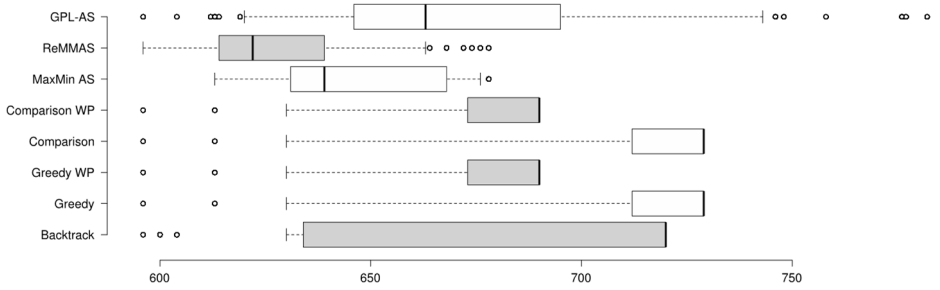
Obr. 74. Výsledky pro feromonového agenta, test-350 s 45 % prav. blokací

8.1.2 *r*-CTP syntetické mapy

U varianty *Recoverable-CTP* v případě, že dojde k zablokování hrany, není hrana zablokována navždy a lze ji obnovit. Experimenty byly provedeny na mapách delaunay-50, test-100, test-350.

Mapa delaunay-50

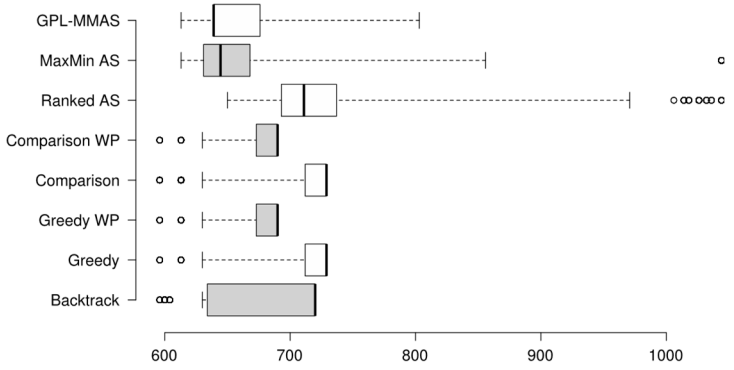
Pravděpodobnost blokací 15 %



	Backtrack	Greedy	Greedy WP	Comp.	Comp. WP
3 kvartil	720	729	690	729	690
Medián	720	729	690	729	690
1 kvartil	634	712	673	712	673

	MMAS	ReMMAS	GPLAS
3 kvartil	668	639	695
Medián	639	622	663
1 kvartil	631	614	646

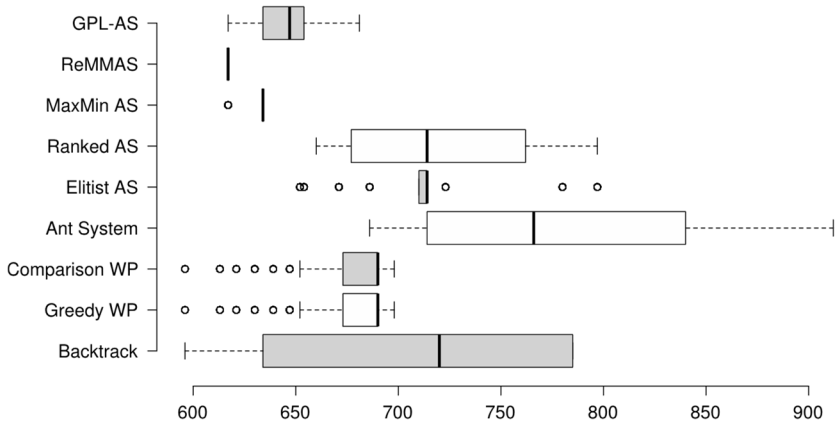
Obr. 75. Výsledky pro cestovního agenta, delaunay-50 s 15 % prav. Blokací



	Backtrack	Greedy	Greedy WP	Comp.	Comp. WP	RAS	MMAS	GPLAS
3 kvartil	720	729	690	729	690	737	668	676
Medián	720	729	690	729	690	711	644	639
1 kvartil	634	712	673	712	673	693	631	639

Obr. 76. Výsledky pro feromonového agenta, delaunay-50 s 15 % prav. blokací

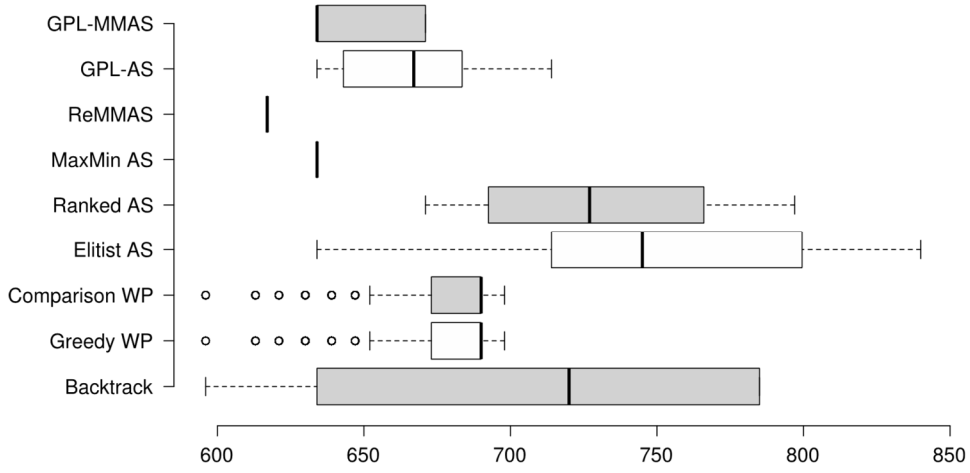
Pravděpodobnost blokací 30 %



	Backtrack	Greedy WP	Comp. WP	AS
3 kvartil	785	690	690	840
Medián	720	690	690	766
1 kvartil	634	673	673	714

	EAS	RAS	MMAS	ReMMAS	GPL-AS
3 kvartil	714	762	634	617	654
Medián	714	714	634	617	647
1 kvartil	710	677	634	617	634

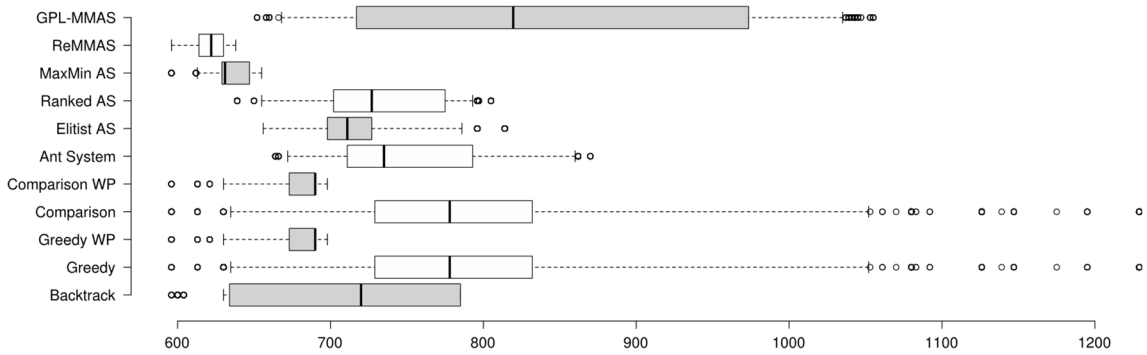
Obr. 77. Výsledky pro cestovního agenta, delaunay-50 s 30 % prav. blokací



	Backtrack	Greedy WP	Comp. WP	EAS	RAS	MMAS	ReMMAS	GPLAS	GPL-MMAS
3 kvartil	785	690	690	799	766	634	617	683	671
Medián	720	690	690	745	727	634	617	667	634
1 kvartil	634	673	673	714	692.50	634	617	643	634

Obr. 78. Výsledky pro feromonového agenta, delaunay-50 s 30 % prav. blokací

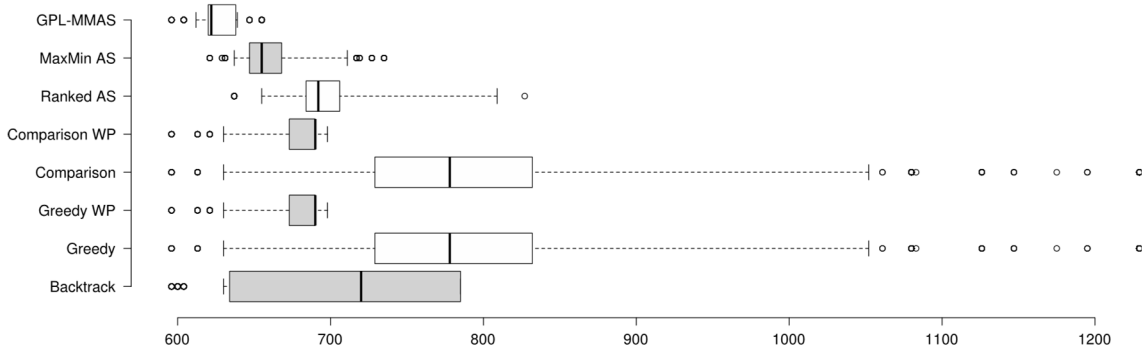
Pravděpodobnost blokací 45 %



	Backtrack	Greedy	Greedy WP	Comp.	Comp. WP	AS
3 kvartil	785	1052	698	1052	698	860
Medián	720	778	690	778	690	735
1 kvartil	634	729	673	729	673	711

	EAS	RAS	MMAS	ReMMAS	GPLAS
3 kvartil	786	793	655	638	1035
Medián	711	727	631	622	819
1 kvartil	698	702	629	614	717

Obr. 79. Výsledky pro cestovního agenta, delaunay-50 s 45 % prav. blokací



	Backtrack	Greedy	Greedy WP	Comp.	Comp. WP	RAS	MMAS	GPLAS
3 kvartil	785	832	690	832	690	706	668	638
Medián	720	778	690	778	690	692	655	622
1 kvartil	634	729	673	729	673	684	647	620

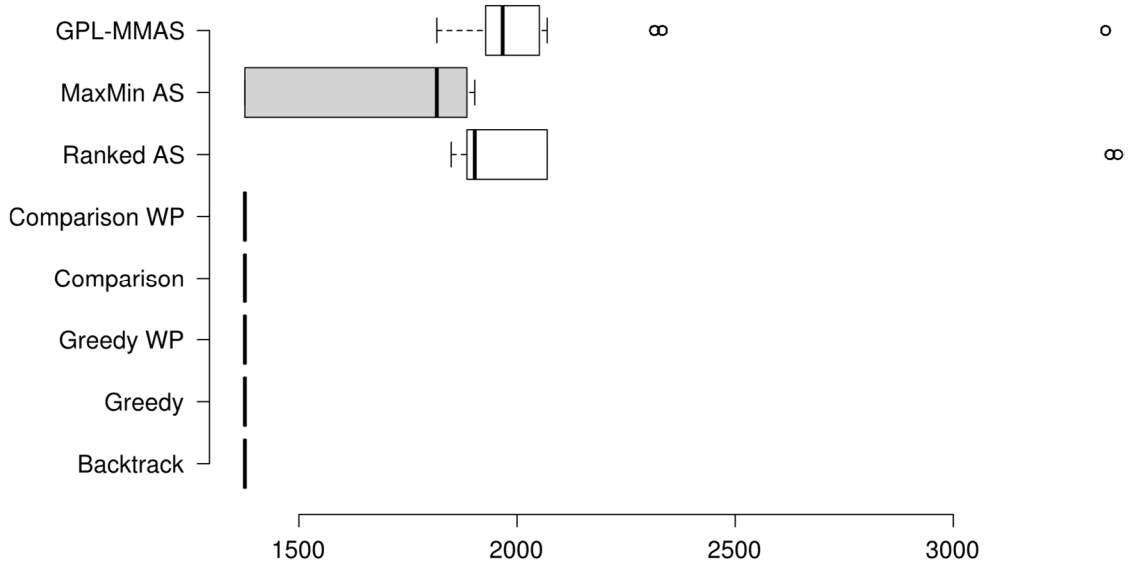
Obr. 80. Výsledky pro feromonového agenta, delaunay-50 s 45 % prav. blokací

Mapa test-100,

Pravděpodobnost blokací 15 %

	Backtrack	Greedy	Greedy WP	Comp.	Comp. WP	MMAS	Re MMAS	GPL-MMAS
3 kvartil	1376	1376	1376	1376	1376	1376	1376	1376
Medián	1376	1376	1376	1376	1376	1376	1376	1376
1 kvartil	1376	1376	1376	1376	1376	1376	1376	1376

Obr. 81. Výsledky pro cestovního agenta, test-100 s 15 % prav. blokací



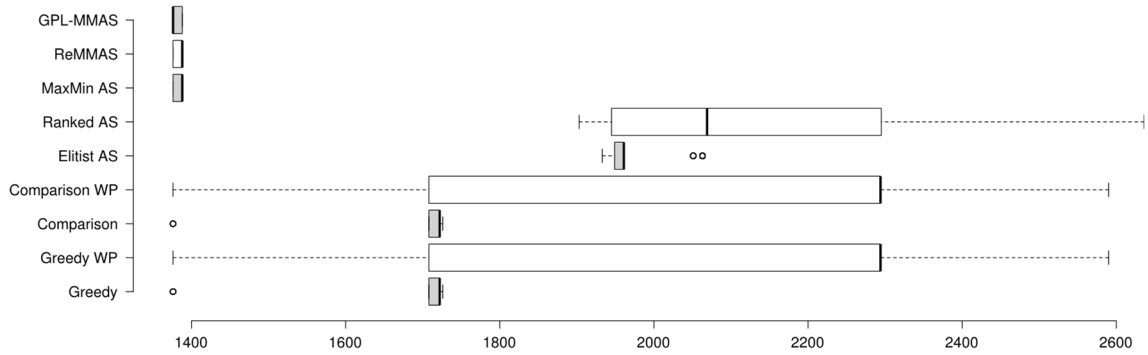
	Backtrack	Greedy	Greedy WP	Comp.	Comp. WP	RAS	MMAS	GPL-MMAS
3 kvartil	1376	1376	1376	1376	1376	2069	1885	2051
Medián	1376	1376	1376	1376	1376	1903	1816	1967
1 kvartil	1376	1376	1376	1376	1376	1885	1376	1928

Obr. 82. Výsledky pro feromonového agenta, test-100 s 15 % prav. blokací

	MMAS	GPL-MMAS
3 kvartil	1376	1376
Medián	1376	1376
1 kvartil	1376	1376

Obr. 83. Výsledky pro feromonového agenta, s počtem mravenců 800, test-100 s 15 % prav. blokací

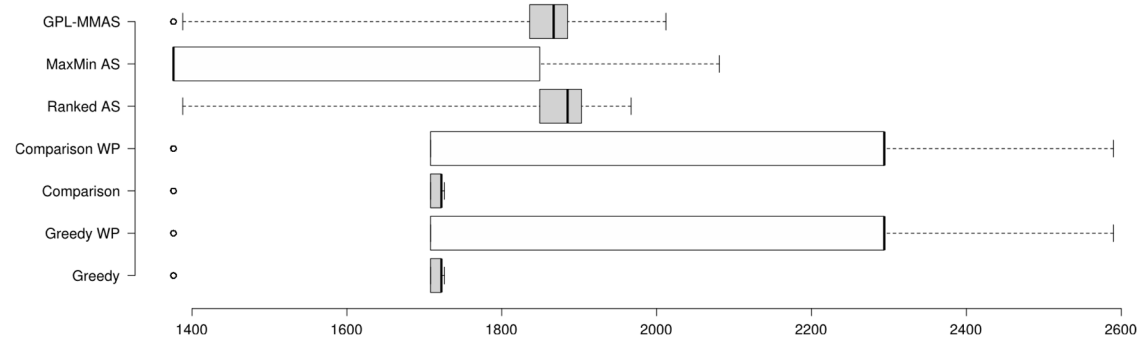
Pravděpodobnost blokací 30 %



	Greedy	Greedy WP	Comp.	Comp. WP
3 kvartil	1722	2294	1722	2294
Medián	1722	2294	1722	2294
1 kvartil	1708	1708	1708	1708

	EAS	RAS	MMAS	ReMMAS	GPL-MMAS
3 kvartil	1961	2295	1388	1388	1388
Medián	1961	2069	1388	1388	1376
1 kvartil	1949	1945	1376	1376	1376

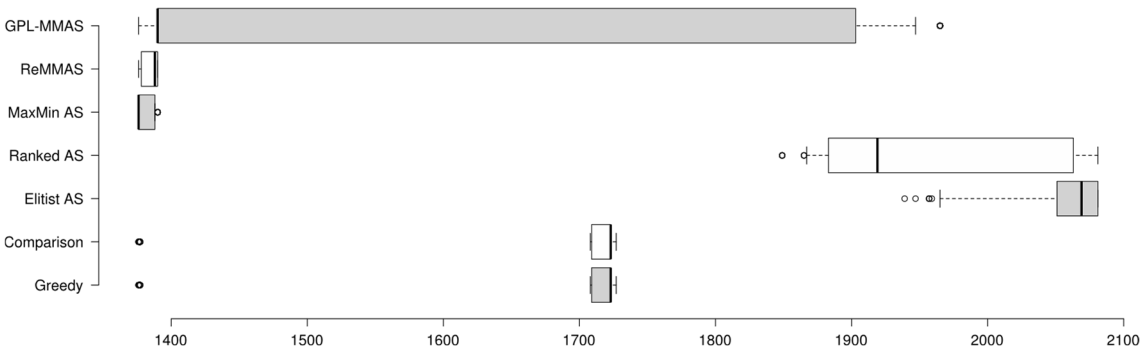
Obr. 84. Výsledky pro cestovního agenta, test-100 s 30 % prav. blokací



	Greedy	Greedy WP	Comp.	Comp. WP	RAS	MMAS	GPL-MMAS
3 kvartil	1722	2294	1722	2294	1903	1849	1885
Medián	1722	2294	1722	2294	1885	1376	1867
1 kvartil	1708	1708	1708	1708	1849	1376	1836

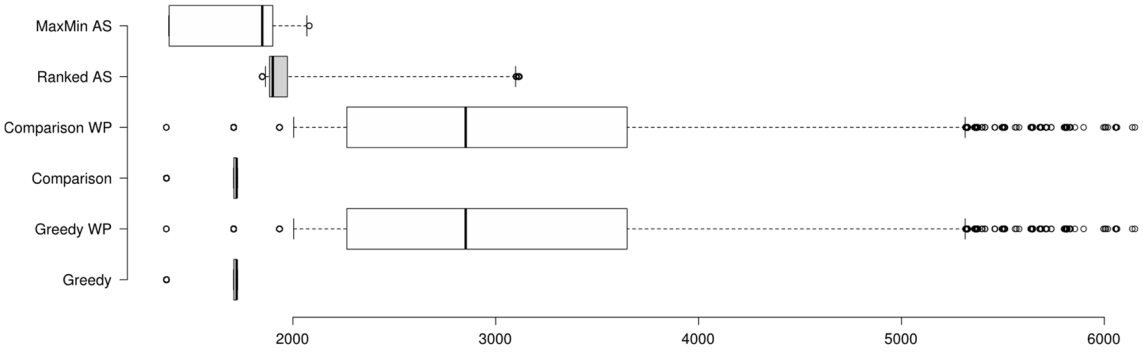
Obr. 85. Výsledky pro feromonového agenta, test-100 s 30 % prav. blokací

Pravděpodobnost blokací 45 %



	Greedy	Comp.	EAS	RAS	MMAS	ReMMAS	GPL-MMAS
3 kvartil	1723	1723	2081	2063	1388	1390	1903
Medián	1723	1723	2069	1919	1376	1388	1390
1 kvartil	1709	1709	2051	1883	1376	1378	1390

Obr. 86. Výsledky pro cestovního agenta, test-100 s 45 % prav. blokací

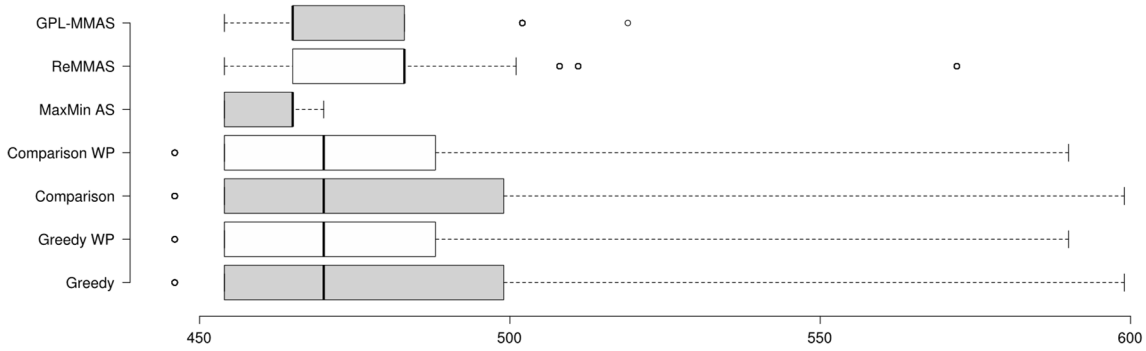


	Greedy	Greedy WP	Comp.	Comp. WP	RAS	MMAS
3 kvartil	1723	3647.75	1723	3647.75	1973	1901
Medián	1723	2852	1723	2852	1901	1849
1 kvartil	1709	2266	1709	2266	1885	1390

Obr. 87. Výsledky pro feromonového agenta, test-100 s 45 % prav. blokací

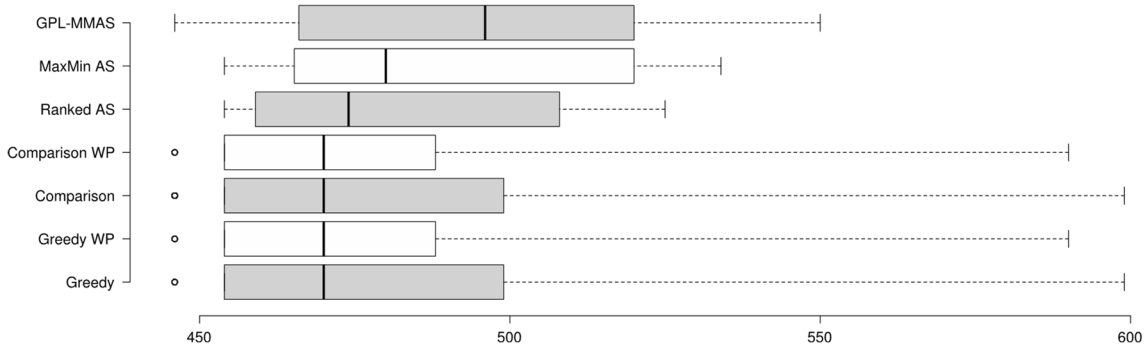
Mapa test-350

Pravděpodobnost blokací 30 %



	Greedy	Greedy WP	Comp.	Comp. WP	MMAS	ReMMAS	GPL-MMAS
3 kvartil	499	488	499	488	465	483	483
Medián	470	470	470	470	465	483	465
1 kvartil	454	454	454	454	454	465	465

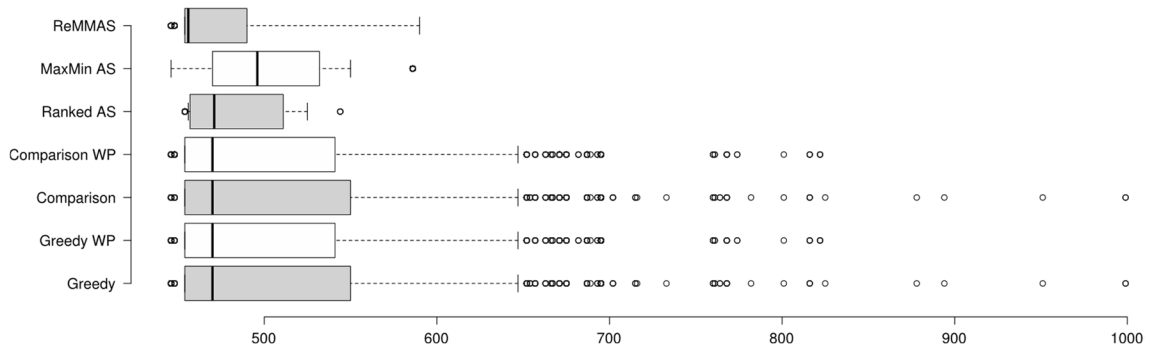
Obr. 88. Výsledky pro cestovního agenta, test-350 s 30 % prav. blokací



	Greedy	Greedy WP	Comp.	Comp. WP	RAS	MMAS	GPL-MMAS
3 kvartil	499	488	499	488	508	520	520
Medián	470	470	470	470	474	480	496
1 kvartil	454	454	454	454	459	465.25	466

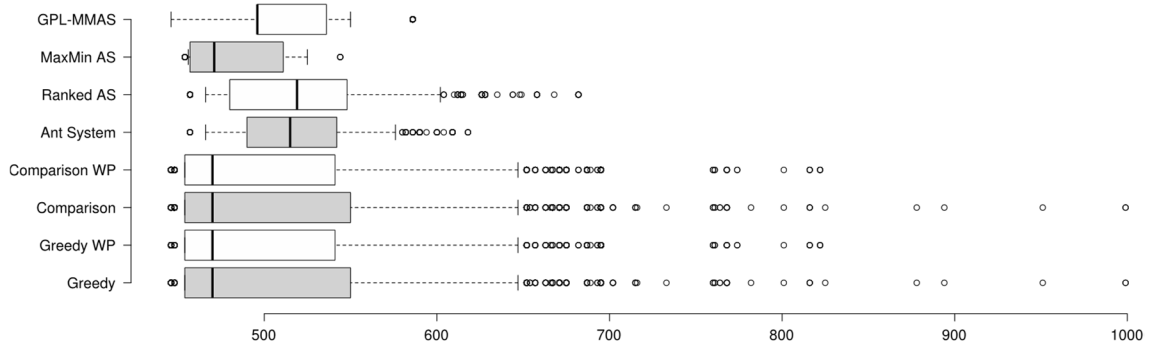
Obr. 89. Výsledky pro feromonového agenta, test-350 s 30 % prav. blokací

Pravděpodobnost blokací 45 %



	Greedy	Greedy WP	Comp.	Comp. WP	RAS	MMAS	ReMMAS
3 kvartil	550	541	550	541	511	532	490
Medián	470	470	470	470	471	496	456
1 kvartil	454	454	454	454	457	470	454

Obr. 90. Výsledky pro cestovního agenta, test-350 s 45 % prav. blokací



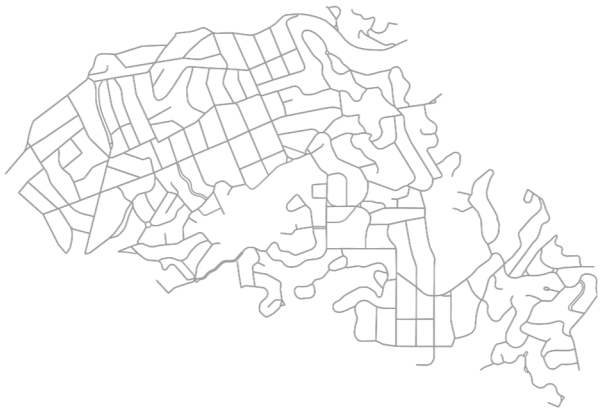
	Greedy	Greedy WP	Comp.	Comp. WP	AS	RAS	MMAS	GPL-MMAS
3 kvartil	550	541	550	541	542	548	511	536
Medián	470	470	470	470	515	519	471	496
1 kvartil	454	454	454	454	490	480	457	496

Obr. 91. Výsledky pro feromonového agenta, test-350 s 45 % prav. blokací

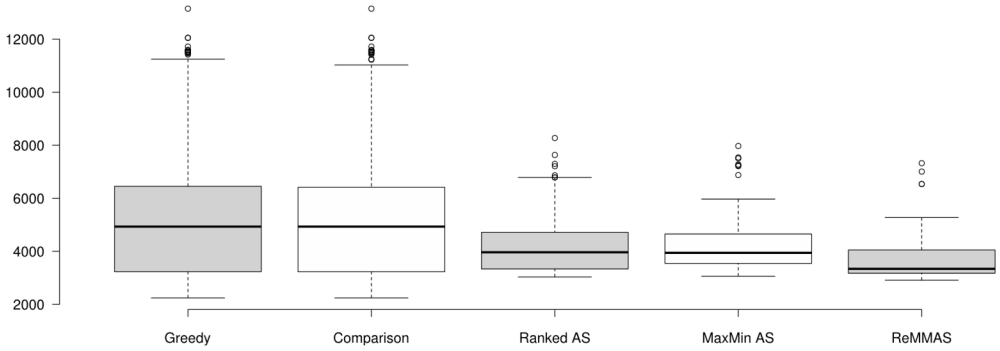
8.1.3 Reálné mapové podklady

Pro další experimenty byly zvoleny reálné mapové z projektu OpenStreet Maps[168] a to pomocí knihovny OsmNx [169]. Tato knihovna byla použita jak pro získání, tak i předzpracování map pro samotný experiment. Testování probíhalo na dvou rozdílně velkých mapách. První mapou je město Piedmont, které se nalézá nedaleko San Francisca v Kalifornii. Graf této mapy tvoří 346 vrcholů, počet hran je 938, průměrný stupeň vrcholu v grafu je 5,4 a počet křižovatek je 312. Celková délka všech hran je 59,6 km a průměrná délka hrany je 121 m. Mravenčí algoritmy mají následující parametry: počet mravenců: 500 a maximální počet iterací: 300. Hodnoty parametrů, α, β, ρ jsou stejné jako u předchozích experimentů, samotný počet opakování experimentu je 50.

```
import osmnx as ox
G = ox.graph_from_place('Piedmont,
CA, USA', network_type='drive')
ox.plot_graph(G)
```

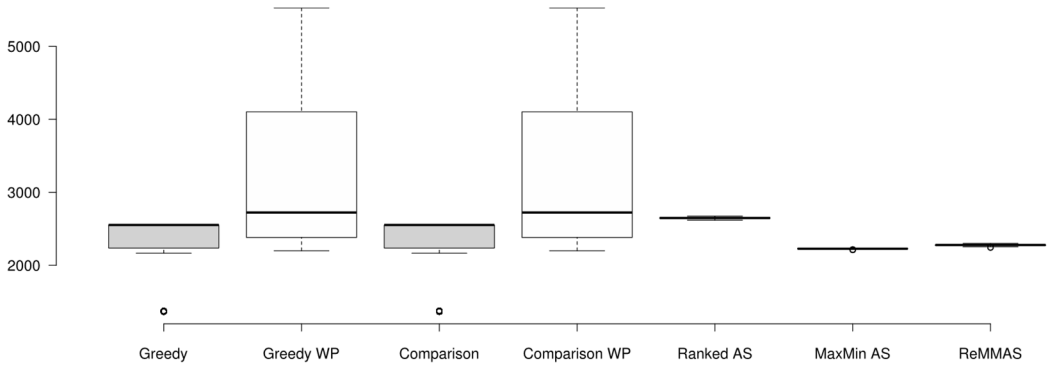


Obr. 92. Příklad importu dat města Piedmont pomocí knihovny OsmNx



	Greedy	Comparison	RAS	MMAS	ReMMAS
3 kvartil	6453	6417	4716	4653	4050
Medián	4932	4932	3965	3943	3340
1 kvartil	3230	3230	3337	3545	3173

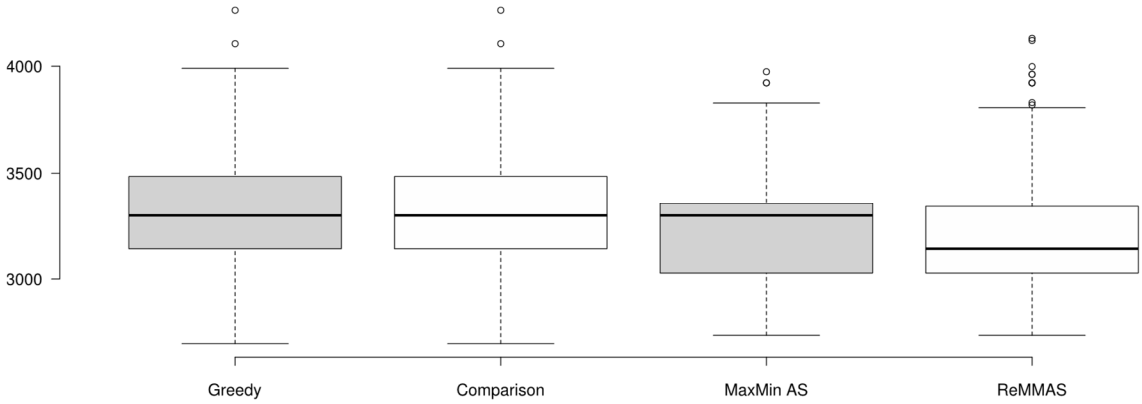
Obr. 93. Výsledky pro cestovního agenta *k*-CTP, Piedmont s 25 % prav. Blokací



	Greedy	Greedy WP	Comp.	Comp. WP	RAS	MMAS	ReMMAS
3 kvartil	2552	4103	2552	4103	2656	2232	2282
Medián	2552	2723	2552	2723	2648	2227	2278
1 kvartil	2236	2381	2236	2381	2640	2226	2270

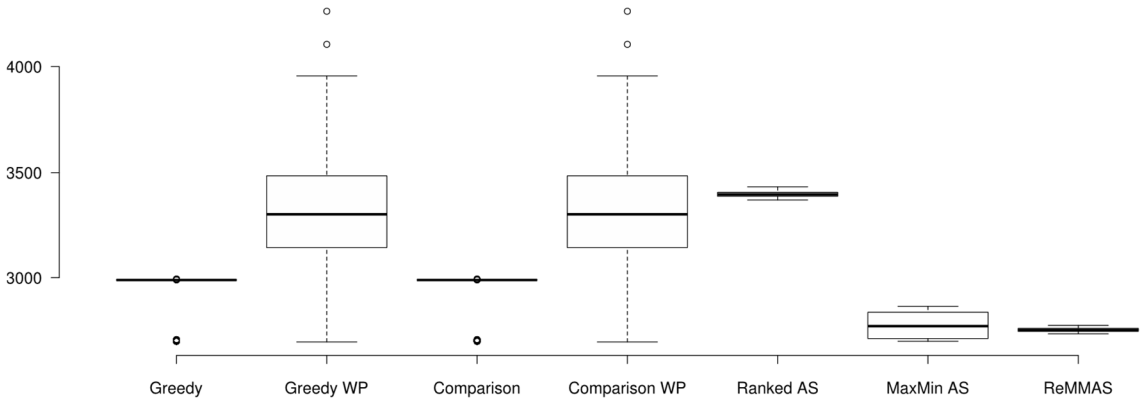
Obr. 94. Výsledky pro cestovního agenta *r*-CTP, Piedmont s 25 % prav. blokací

Druhý test byl proveden na mapě ostrova Manhattan v New Yorku. Graf této mapy tvoří 4486 vrcholů s počtem hran 9751 s celkovou délkou 963,8 km a průměrnou délkou 117 metrů. Průměrný stupeň vrcholu je pak 4,3 a počet křižovatek je 4384. Mravenčí algoritmy mají následující parametry: počet mravenců: 300, $\alpha = 0{,}8$; $\beta = 1{,}8$; $\rho = 0{,}7$ a maximální počet iterací: 300. Počet opakování experimentu: 50.



	Greedy	**Comparison**	**MMAS**	**ReMMAS**
3 kvartil	3485	3485	3356	3342
Medián	3299	3299	3299	3142
1 kvartil	3142	3142	3028	3028

Obr. 95. Výsledky pro cestovního agenta *k*-CTP, Manhattan 25 % prav. blokací



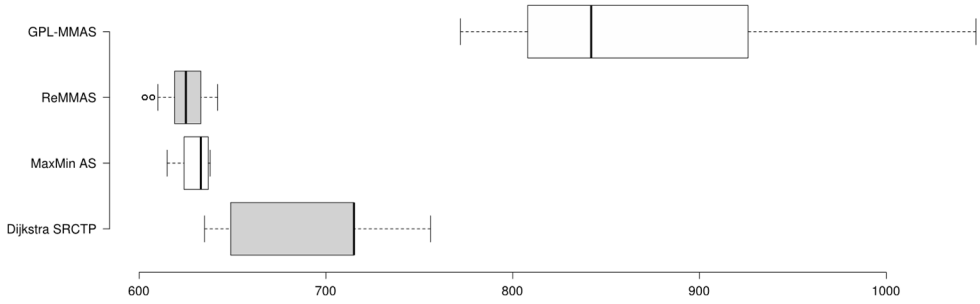
	Greedy	**Greedy WP**	**Comp.**	**Comp. WP**	**RAS**	**MMAS**	**ReMMAS**
3 kvartil	2989	3485	2989	3485	3407	2837	2761
Medián	2989	3299	2989	3299	3397	2771	2753
1 kvartil	2989	3142	2989	3142	3389	2713	2747

Obr. 96. Výsledky pro cestovního agenta *r*-CTP, Manhattan 25 % prav. blokací

8.2 Známá pravděpodobnost blokace

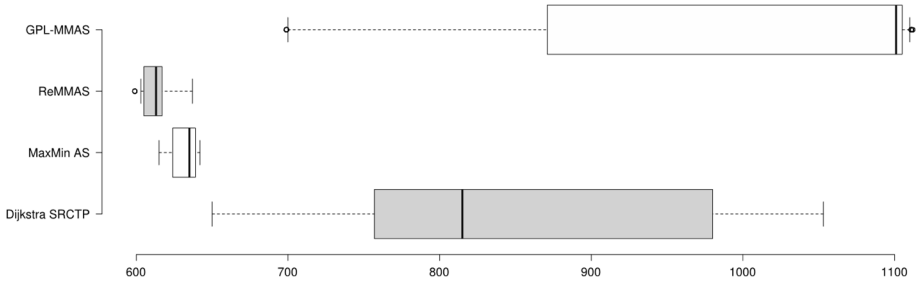
Pro případ, kdy je dopředu cestovateli známa pravděpodobnost blokace byly provedeny experimenty na stejných mapách jako v případě neznámé pravděpodobnosti blokace. Nastavení pro mravenčí algoritmy bylo stejné jako u předchozích syntetických map. Jako verifikační strategie byla zvolena strategie „Cestovního agenta".

Mapa delaunay-50



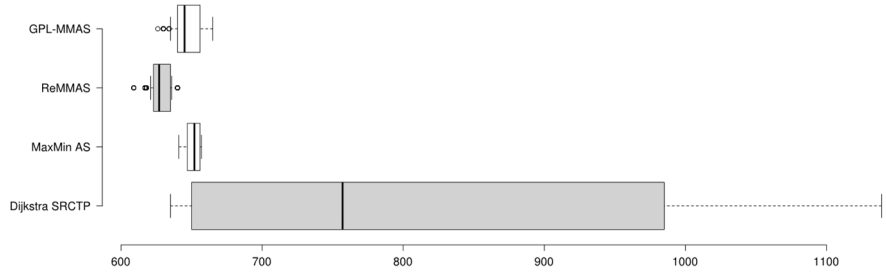
	Dijkstra SRCTP	MMAS	ReMMAS	GPL-MMAS
3 kvartil	715	637	633	926
Medián	715	633	625	842
1 kvartil	649	624	619	808

Obr. 97. Výsledky pro cestovního agenta, delaunay-50 s 15 % prav. blokací



	Dijkstra SRCTP	MMAS	ReMMAS	GPL-MMAS
3 kvartil	980	639	617	1105
Medián	815	635	613	1101
1 kvartil	757	624	605	871

Obr. 99. Výsledky pro cestovního agenta, delaunay-50 s 30 % prav. blokací



	Dijkstra SRCTP	MMAS	ReMMAS	GPL-MMAS
3 kvartil	985	656	635	656
Medián	757	652	627	645
1 kvartil	650	647	623	640

Obr. 100. Výsledky pro cestovního agenta, delaunay-50 s 45 % prav. blokací

Mapa test-100

	Dijkstra SRCTP	MMAS	ReMMAS
3 kvartil	1376	1376	1376
Medián	1376	1376	1376
1 kvartil	1376	1376	1376

Obr. 101. Výsledky pro agenta, test-100 s 15 % prav. blokací

	Dijkstra SRCTP	MMAS	ReMMAS
3 kvartil	2771	1389	1376
Medián	2771	1389	1376
1 kvartil	2771	1389	1376

Obr. 102. Výsledky pro agenta, test-100 s 30 % prav. blokací

	Dijkstra SRCTP	MMAS	ReMMAS
3 kvartil	5976	1393	1387
Medián	5714	1388	1381
1 kvartil	4885	1386	1380

Obr. 103. Výsledky pro agenta, test-100 s 45 % prav. blokací

Mapa test-350

	Dijkstra SRCTP	MMAS	ReMMAS
3 kvartil	446	449	446
Medián	446	449	446
1 kvartil	446	449	446

Obr. 104. Výsledky pro agenta, test-350 s 15 % prav. blokací

	Dijkstra SRCTP	MMAS	ReMMAS
3 kvartil	447	469	460
Medián	447	462	459
1 kvartil	447	449	450

Obr. 105. Výsledky pro agenta, test-350 s 30 % prav. blokací

	Dijkstra SRCTP	MMAS	ReMMAS
3 kvartil	511	470	471
Medián	496	463	467
1 kvartil	450	457	464

Obr. 106. Výsledky pro agenta, test-350 s 45 % prav. blokací

9 ZÁVĚR

Předložená disertační práce se věnuje problematice plánování cest v neurčitém prostředí, přičemž vyzdvihuje mravenčí algoritmy jako zcela nový přístup pro řešení problému CTP. Složitost problému kanadského cestujícího (CTP) spočívá v několika klíčových aspektech, které činí tento problém významně náročnějším než klasické problémy plánování cesty. V úloze CTP je cestovatel (agent) konfrontován s prostředím, ve kterém mohou být některé cesty neprůchodné kvůli nepředvídatelným překážkám, jako je špatné počasí nebo stavební práce. Tyto překážky jsou známy pouze při dosažení bodu před nimi, což vyžaduje dynamické přizpůsobení plánu cesty. Tato kombinatorická úloha je třídy NP-hard (konkrétně PSPACE complete, #P-hard), což znamená, že pro ni neexistuje známý algoritmus, který by poskytoval přesné řešení v polynomiálním čase pro všechny možné instance.

V rámci teoretické části práce je proveden vhled do studované problematiky jak vlastní optimalizační úlohy CTP, tak intenzivně do pokročilých řešících heuristik a metaheuristik, vč. vlastní implementace. Realizace kapitol 2 až 6 si vyžádala velmi intenzivní rešeršní činnost. komplexní rešerše tohoto problému a jeho současného stavu poznání. Rešeršní část a state-of-the-art dané problematiky uvádí vlastní kapitola věnována mravenčím metaheuristikám (Ant Colony Optimization). V rámci rešerší byly vybrány a jednotně klasifikovány všechny varianty CTP úloh a pro ně navržených heuristik, které však existovaly pouze pro kategorii k-CTP (neobnovitelné blokace). Zásadní je rovněž skutečnost, že pro CTP problémy třídy r-CTP (obnovitelné blokace) řešící heuristiky neexistovaly a byly nově navrženy – konkrétně metody heuristiky greedy a comparison s inspirací v *k*-CTP a heuristiky backtrack, greedy wp a comparison wp jako zcela nové.

Jak bylo uvedeno, implementace mravenčích algoritmů byla pro řešení diskutovaných variant CTP zcela nová a dosud nepublikovaná, což je zásadní přínos předložené práce. V práci jsou uvedeny pseudokódy či části kódu modifikovaných algoritmů, popis a vlastnosti. V rámci provedených experimentů jde majoritně o algoritmy:

- AS Ant System,
- RAS Ranked Ant System,
- EAS Elistist Ant System,
- MMAS MaxMin Ant System,
- ReMMAS Restarted MaxMin Ant System,
- GPL-AS Global Parameter Learning Ant System,
- GPL-MMAS Global Parameter Learning MaxMin Ant System.

Textová část práce dále obsahuje adekvátní popis vytvářeného software pro řešení problému kanadského cestujícího a obecně i některých dalších kombinatorických optimalizačních problémů daného typu. Ve vytvořeném software, resp. řešiči jsou implementovány diskutované heuristické i metaheuristické ACO metody, včetně několika vlastních modifikací. Realizované softwarové prostředí bylo vyvinuto v jazyce C++ (11 a výše), s důrazem na více vláknové řešení a profilaci rychlosti. Navržený systém je využitelný jako knihovna, zahrnuje benchamarkovací nástroj a z uživatelského hlediska vhodné GUI i příkazovou řádku pro vlastní experimenty.

Důležitou a praktickou částí předložené práce jsou výsledky experimentů při řešení umělých i praktických (reálných) úloh CTP. Uvedená řešení, dosažená s využitím vytvořených řešičů a jejich parametrizací, jsou realizována na definovaném testovacím korpusu. Ten je složen jak z vygenerovaných – tedy syntetických, tak i reálných map. Tyto modelové mapy s definicí CTP tvoří vytvořenou knihovnu pojmenovanou *CTPlib*. Tato knihovna bude zveřejněna v průběhu roku 2024 a bude dostupná zájemcům o problematiku CTP úloh pro možné srovnání dosažených výsledků při užití nově navržených nebo jinak parametrizovaných algoritmů.

Z rozsáhlé experimentální činnosti v ohledu řešících algoritmů a testovacích map CTP problémů vznikla 8. kapitola, prezentující dosažené výsledky v ohledu heuristiky vs. mravenčí algoritmy. V uvedeném ohledu bylo využito srovnání algoritmů, resp. dosažených výsledků na základě mediánu a mezikvartilových rozpětí, což je objasněno v uvedené kapitole. K porovnání a objektivizaci výsledků byly navrženy dvě verifikační strategie testovacího agenta označené jako cestovní agent a feromonový agent. Uveďme, že pokud jde o verifikační strategie, tak z provedených experimentů se potvrdilo, že strategie cestovního agenta je lepší než strategie feromonového agenta. V případě této druhé strategie se ukázalo, že s rostoucím počtem vrcholů v mapě je nutný i větší počet mravenců pro nalezení řešení konkurence schopného s heuristickými metodami. Vytvořený software umožňuje velmi bohatou parametrizaci

Z provedených experimentů je patrný jasný vzor, kdy mravenčí algoritmus MMAS (*MaxMin Ant System*) a jeho modifikace (*ReMMAS* a *GPL–MMAS*) jsou schopny naleznout velmi kvalitní řešení, které většinově překonává heuristické metody, nebo se jím nalezeným řešením vyrovná. Z provedených experimentů na dané třídě CTP problémů lze konstatovat, že při blokacích nad 20 % hran, byly mravenčí strategie vždy úspěšnější a jejich úspěšnost s procentem blokací rostla. Mezi mravenčími strategiemi je pak zřejmá problémová závislost, vzhledem k definované CTP mapě, přičemž varianty MMAS patřily k nejlepším.

Uvedené experimenty prokázaly opodstatněnost zcela inovativní implementace mravenčích algoritmů v problematice řešení CTP problému a jeho variant. K negativnímu faktoru při využití mravenčích algoritmů patří doba výpočtu, která byla násobná vůči nově implementovaným heuristikám. V ohledu nejkratší cesty však dominují mravenčí strategie.

Hlavním teoretickým přínosem disertační práce jsou:

- Původní modifikace mravenčích algoritmů na nový problém – CTP a jeho varianty.
- Návrh několika nových modifikací mravenčích algoritmů.
- Návrh nových heuristických metod, pro řešení CTP a jeho variant.
- Vytvoření verifikačních strategií pro porovnání mravenčích algoritmů s heuristickými metodami.

Mezi **praktické přínosy práce** jednoznačně patří vytvořený výpočetní software. Ten umožňuje řešit nejen problém CTP, ale i další jako TSP a CTSP. Dále sada provedených experimentů.

Důležité praktické přínosy:

- Implementace sady mravenčích metod.
- Implementace sady heuristických metod.
- Více vláknová paralelizace metod pro urychlení výpočtů.
- Tvorba a úprava syntetických mapových podkladů problémů, a knihovna CTPlib.
- Podpora reálných mapových podkladů problémů.

Závěrem lze konstatovat, že cíle disertační práce se podařilo bezezbytku splnit. V započatém výzkumu této problematiky lze dále pokračovat v mnoha směrech. Ať už návrhem nových mravenčích algoritmů, jejich verifikačních strategií nebo heuristických metod. Pochopitelně problém kanadského cestující je velmi bohatý a lze hledat jeho další varianty s praktickým podtextem vzhledem k aplikační oblasti. Volněji řečeno *„každý si najde ty svoje Poděbrady“*.

SEZNAM LITERATURY

[1] MEYER, Jean-Arcady a David FILLIAT. Map-based navigation in mobile robots:: Ii. a review of map-learning and path-planning strategies. *Cognitive Systems Research* [online]. 2003, **4**(4), 283–317. Dostupné z: doi:10/cxx4dv

[2] CASTELLANOS, Jose A., J. M. M. MONTIEL, José NEIRA a Juan D. TARDÓS. The SPmap: A probabilistic framework for simultaneous localization and map building. *IEEE Transactions on robotics and Automation* [online]. 1999, **15**(5), 948–952. Dostupné z: doi:10/cvxgng

[3] DISSANAYAKE, Gamini, Hugh DURRANT-WHYTE a Tim BAILEY. A computationally efficient solution to the simultaneous localisation and map building (SLAM) problem. In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)* [online]. B.m.: IEEE, 2000, s. 1009–1014. Dostupné z: doi:10/bvwpm4

[4] LATOMBE, Jean-Claude. *Robot motion planning*. B.m.: Springer Science & Business Media, 2012.

[5] BROOKS, Rodney A. Solving the find-path problem by good representation of free space. *IEEE Transactions on Systems, Man, and Cybernetics* [online]. 1983, (2), 190–197. Dostupné z: doi:10/ghbsgk

[6] BHATTACHARYYA, Aneeta, Ekta SINGLA a Bhaskar DASGPUTA. Robot path planning using silhouette method. In: *13th National Conference on Mechanisms and Machines.(NaCoMM07), Bangalore, India*. 2007.

[7] ŠEDA, Miloš a Václav PICH. Robot motion planning using generalised voronoi diagrams. In: *Proceedings of the 8th conference on signal processing, computational geometry and artificial vision* [online]. Stevens Point, Wisconsin, USA: World Scientific and Engineering Academy and Society (WSEAS), 2008, s. 215–220. ISCGAV'08. ISBN 978-960-6766-95-4. Dostupné z: https://dl.acm.org/doi/abs/10.5555/1503734.1503770

[8] DIESTEL, Reinhard. *Graph theory*. 4th ed. Heidelberg ; New York: Springer, 2010. Graduate texts in mathematics, 173. ISBN 978-3-642-14278-9.

[9] JUNGNICKEL, D. *Graphs, networks, and algorithms*. Fourth edition. Heidelberg ; New York: Springer, 2013. Algorithms and computation in mathematics, 5. ISBN 978-3-642-32277-8.

[10] EULER, Leonhard. Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae*. 1741, 128–140.

[11] HOPKINS, Brian a Robin WILSON. The Truth about Königsberg. In: *Studies in the History and Philosophy of Mathematics* [online]. B.m.: Elsevier, 2007 [vid. 2021-12-28], s. 409–420. ISBN 978-0-444-52728-8. Dostupné z: doi:10.1016/S0928-2017(07)80022-3

[12] HIERHOLZER, Carl a Chr WIENER. Ueber die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren. *Mathematische Annalen* [online]. 1873, **6**(1), 30–32. ISSN 0025-5831, 1432-1807. Dostupné z: doi:10/ddxjpf

[13] GUAN, Meigu. Graphic programming using odd and even points. *Chinese Math.* 1962, **1**, 237–277.

[14] CORBERÁN, Ángel a Gilbert LAPORTE, ed. *Arc routing: problems, methods, and applications*. Philadelphia: Society for Industrial and Applied Mathematics : Mathematical Optimization Society, 2014. MOS-SIAM series on optimization, 20. ISBN 978-1-61197-366-2.

[15] MERIAN-ERBEN. *Königsberg 1651* [online]. 1. březen 100n. l. [vid. 2016-06-02]. Dostupné z: https://commons.wikimedia.org/wiki/File:Image-Koenigsberg,_Map_by_Merian-Erben_1652.jpg

[16] EULER, Leonhard. *English: Figure 1 from the English translation of Solutio problematis ad geometriam situs pertinentis in The World of Mathematics, Volume I. It's a cleaned up copy of the original drawn by Leonhard Euler.* [online]. 11. červen 2014 [vid. 2016-06-02]. Dostupné z: https://commons.wikimedia.org/wiki/File:The_Seven_Bridges_of_K%C3%B6nigsberg,_Fig._1.png

[17] MACKENZIE, Donald A. *Mechanizing proof: computing, risk, and trust*. Cambridge, Mass. London: MIT, 2004. Inside technology series. ISBN 978-0-262-63295-9.

[18] CHARTRAND, Gary, Linda LESNIAK a Ping ZHANG. *Graphs & digraphs*. 5th ed. Boca Raton, FL: Chapman and Hall/CRC, 2011. ISBN 978-1-4398-2627-0.

[19] APPEL, Kenneth I. a Wolfgang HAKEN. *Every planar map is four colorable*. Providence, R.I: American Mathematical Society, 1989. Contemporary mathematics, v. 98. ISBN 978-0-8218-5103-6.

[20] MAIX. *A political Map of Europe in SVG format. Every country has an id which is its ISO-3116-1-ALPHA2 code in lower case for easy coloring.* [online]. 4. únor 2007 [vid. 2021-12-28]. Dostupné z: https://commons.wikimedia.org/wiki/File:Blank_map_of_Europe_(with_disputed_regions).svg

[21] BORŮVKA, Otakar. O jistém problému minimálním. 1926.

[22] FLOREK, Kazimierz, Jan \LUKASZEWICZ, Julian PERKAL, Hugo STEINHAUS a Stefan ZUBRZYCKI. Sur la liaison et la division des points d'un ensemble fini. In: *Colloquium mathematicum*. 1951, s. 282–285.

[23] SOLLIN, M. La trace de canalisation. *Programming, Games, and Transportation Networks*. 1965.

[24] KRUSKAL, Joseph B. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society* [online]. 1956, **7**(1), 48–50. ISSN 0002-9939, 1088-6826. Dostupné z: doi:10/fkqhgs

[25] ŠIŠMA, Pavel. Biografie ceskych matematiku. *Významní matematici v českých zemích* [online]. 2002 [vid. 2021-12-28]. Dostupné z: https://web.math.muni.cz/biografie/

[26] JARNÍK, Vojtěch. O jistém problému minimálním. (Z dopisu panu O. Borůvkovi). 1930.

[27] PRIM, R. C. Shortest Connection Networks And Some Generalizations. *Bell System Technical Journal* [online]. 1957, **36**(6), 1389–1401. ISSN 00058580. Dostupné z: doi:10/gmvhh7

[28] DIJKSTRA, E. W. A note on two problems in connexion with graphs. *Numerische Mathematik* [online]. 1959, **1**(1), 269–271. ISSN 0029-599X, 0945-3245. Dostupné z: doi:10/dpvk8c

[29] KONIG, Dénes. *Theorie der endlichen und unendlichen Graphen*. B.m.: American Mathematical Soc., 2001.

[30] GROSS, Jonathan L., Jay YELLEN a Ping ZHANG, ed. *Handbook of graph theory*. Second edition. Boca Raton: CRC Press, Taylor & Francis Group, 2014. Discrete mathematics and its applications. ISBN 978-1-4398-8018-0.

[31] SKIENA, Steven S. Sorting and Searching. In: Steven S. SKIENA*The Algorithm Design Manual* [online]. London: Springer London, 2012 [vid. 2021-12-28], s. 103–144. ISBN 978-1-84800-069-8. Dostupné z: doi:10.1007/978-1-84800-070-4_4

[32] FRANA, Philip L. a Thomas J. MISA. An interview with Edsger W. Dijkstra. *Communications of the ACM* [online]. 2010, **53**(8), 41–47. ISSN 0001-0782, 1557-7317. Dostupné z: doi:10/dx9rdd

[33] SCHRIJVER, Alexander. On the History of Combinatorial Optimization (Till 1960). In: *Handbooks in Operations Research and Management Science* [online]. B.m.: Elsevier, 2005 [vid. 2021-12-28], s. 1–68. ISBN 978-0-444-51507-0. Dostupné z: doi:10.1016/S0927-0507(05)12001-5

[34] HART, Peter E., Nils J. NILSSON a Bertram RAPHAEL. Correction to „A Formal Basis for the Heuristic Determination of Minimum Cost Paths". *ACM SIGART Bulletin* [online]. 1972, (37), 28–29. ISSN 0163-5719. Dostupné z: doi:10/d63jt4

[35] DECHTER, Rina a Judea PEARL. Generalized best-first search strategies and the optimality of A*. *Journal of the ACM* [online]. 1985, **32**(3), 505–536. ISSN 0004-5411, 1557-735X. Dostupné z: doi:10/bprkkt

[36] DOŠLÁ, Zuzana a Ondřej DOŠLÝ. *Metrické prostory. Teorie a příklady*. B.m.: Masarykova univerzita, 2016.

[37] KORF, Richard E. Depth-first iterative-deepening. *Artificial Intelligence* [online]. 1985, **27**(1), 97–109. ISSN 00043702. Dostupné z: doi:10/b8fmpm

[38] STENTZ, A. Optimal and efficient path planning for partially-known environments. In: *1994 IEEE International Conference on Robotics and Automation*: *Proceedings of the 1994 IEEE International Conference on Robotics and Automation* [online]. San Diego, CA, USA:

IEEE Comput. Soc. Press, 1994, s. 3310–3317 [vid. 2021-12-28]. ISBN 978-0-8186-5330-8. Dostupné z: doi:10/fsd9gj

[39] KOENIG, S. a M. LIKHACHEV. Fast replanning for navigation in unknown terrain. *IEEE Transactions on Robotics* [online]. 2005, **21**(3), 354–363. ISSN 1552-3098. Dostupné z: doi:10/crsxh5

[40] KOENIG, Sven, Maxim LIKHACHEV a David FURCY. Lifelong Planning A*. *Artificial Intelligence* [online]. 2004, **155**(1–2), 93–146. ISSN 00043702. Dostupné z: doi:10/b4xkz6

[41] DANIEL, K., A. NASH, S. KOENIG a A. FELNER. Theta*: Any-Angle Path Planning on Grids. *Journal of Artificial Intelligence Research* [online]. 2010, **39**, 533–579. ISSN 1076-9757. Dostupné z: doi:10/gfv5vf

[42] FLOYD, Robert W. Algorithm 97: Shortest path. *Communications of the ACM* [online]. 1962, **5**(6), 345. ISSN 0001-0782, 1557-7317. Dostupné z: doi:10/bcc7sw

[43] BJÖRNSSON, Yngvi, Markus ENZENBERGER, Robert C. HOLTE a Jonathan SCHAEFFER. Fringe Search: Beating A* at Pathfinding on Game Maps. *CIG*. 2005, **5**, 125–132.

[44] JOHNSON, Donald B. Efficient Algorithms for Shortest Paths in Sparse Networks. *Journal of the ACM* [online]. 1977, **24**(1), 1–13. ISSN 0004-5411, 1557-735X. Dostupné z: doi:10/dnvm2s

[45] NEUMANN, Bernd, EUROPEAN COORDINATING COMMITTEE FOR ARTIFICIAL INTELLIGENCE, a AUSTRIAN SOCIETY FOR ARTIFICIAL INTELLIGENCE, ed. *ECAI 92, 10th European Conference on Artificial Intelligence, August 3-7, 1992, Vienna, Austria: proceedings*. Chichester ; New York: Wiley, 1992. ISBN 978-0-471-93608-4.

[46] HARABOR, Daniel Damir a Alban GRASTIEN. The JPS Pathfinding System. In: *SOCS*. 2012.

[47] KLOBUŠNÍKOVÁ, Zuzana. *Plánování cesty mobilního robotu* [online]. B.m., 2018 [vid. 2022-01-02]. Vysoké učení technické v BrněFakulta strojního inženýrství. Dostupné z: http://hdl.handle.net/11012/81778

[48] URAS, Tansel, Sven KOENIG a Carlos HERNÁNDEZ. Subgoal graphs for optimal pathfinding in eight-neighbor grids. In: *Twenty-Third International Conference on Automated Planning and Scheduling*. 2013.

[49] MAŇÁKOVÁ, Lenka. *Pokročilé metody plánování cesty mobilního robotu* [online]. B.m., 2020 [vid. 2022-01-02]. Vysoké učení technické v BrněFakulta strojního inženýrství. Dostupné z: http://hdl.handle.net/11012/191857

[50] CHERKASSKY, Boris V., Andrew V. GOLDBERG a Tomasz RADZIK. Shortest paths algorithms: Theory and experimental evaluation. *Mathematical programming*. 1996, **73**(2), 129–174.

[51] AHUACTZIN, Juan Manuel, El-Ghazali TALBI, Pierre BESSIERE a Emmanuel MAZER. Using genetic algorithms for robot motion planning. In: *Workshop on geometric reasoning for perception and action*. B.m.: Springer, 1991, s. 84–93.

[52] THOMAZ, Carlos E., Marco Aurélio C. PACHECO a Marley Maria BR VELLASCO. Mobile robot path planning using genetic algorithms. In: *International Work-Conference on Artificial Neural Networks*. B.m.: Springer, 1999, s. 671–679.

[53] KENNEDY, James a Russell EBERHART. Particle swarm optimization. In: *Proceedings of ICNN'95-international conference on neural networks* [online]. B.m.: IEEE, 1995, s. 1942–1948. Dostupné z: doi:10/bdc3t3

[54] MACAŠ, Martin, Martin SASKA, Lenka LHOTSKÁ, Libor PŘEUČIL a Klaus SCHILLING. Path planning for formations of mobile robots using PSO technique. *Particle Swarm Optimization* [online]. 2009, 329. Dostupné z: doi:10/gpd5jt

[55] QIN, Yuan-Qing, De-Bao SUN, Ning LI a Yi-Gang CEN. Path planning for mobile robot using the particle swarm optimization with mutation operator. In: *Proceedings of 2004 international conference on machine learning and cybernetics (IEEE Cat. No. 04EX826)*. B.m.: IEEE, 2004, s. 2473–2478.

[56] PHAM, D. T., A. GHANBARZADEH, E. KOC, S. OTRI, S. RAHIM a M. ZAIDI. The bees algorithm. *Technical Note, Manufacturing Engineering Centre, Cardiff University, UK*. 2005, 44–48.

[57] KARABOGA, Dervis. *An idea based on honey bee swarm for numerical optimization*. B.m.: Technical report-tr06, Erciyes university, engineering faculty, computer …. 2005.

[58] SATO, Tomoya a Masafumi HAGIWARA. Bee system: finding solution by a concentrated search. *IEEJ Transactions on Electronics, Information and Systems* [online]. 1998, **118**(5), 721–726. Dostupné z: doi:10/gpd5k3

[59] KARABOGA, Dervis a Bahriye AKAY. A survey: algorithms simulating bee swarm intelligence. *Artificial intelligence review* [online]. 2009, **31**(1), 61–85. Dostupné z: doi:10/fkkjv5

[60] LIN, Jiann-Horng a Li-Ren HUANG. Chaotic bee swarm optimization algorithm for path planning of mobile robots. In: *Proceedings of the 10th WSEAS international conference on evolutionary computing*. B.m.: World Scientific and Engineering Academy and Society (WSEAS), 2009, s. 84–89.

[61] GLASIUS, Roy, Andrzej KOMODA a Stan CAM GIELEN. Neural network dynamics for path planning and obstacle avoidance. *Neural Networks* [online]. 1995, **8**(1), 125–133. Dostupné z: doi:10/d3qwgd

[62] HAIGH, Karen Zita, Jonathan Richard SHEWCHUK a Manuela M. VELOSO. Route planning and learning from execution. In: *Preprints of the AAAI 1994 Fall Symposium on Planning and Learning: On to Real Applications, New Orleans, LA*. 1994.

[63] DRUMMOND, Chris. Accelerating reinforcement learning by composing solutions of automatically identified subtasks. *Journal of Artificial Intelligence Research* [online]. 2002, **16**, 59–104. Dostupné z: doi:10/ggw2v2

[64] HUI, Nirmal Baran a Dilip Kumar PRATIHAR. A comparative study on some navigation schemes of a real robot tackling moving obstacles. *Robotics and Computer-Integrated Manufacturing* [online]. 2009, **25**(4–5), 810–828. Dostupné z: doi:10/dgkmjv

[65] KRČEK, P. a J. DVOŘÁK. Mobile robot path planning by means of genetic algorithms. In: *Book of Extended Abstracts of the National Conference with International Participation Engineering Mechanics 2007*. 2007, s. 133–134.

[66] CHVÁTAL, Vašek, William COOK, George B. DANTZIG, Delbert R. FULKERSON a Selmer M. JOHNSON. Solution of a Large-Scale Traveling-Salesman Problem. In: Michael JÜNGER, Thomas M. LIEBLING, Denis NADDEF, George L. NEMHAUSER, William R. PULLEYBLANK, Gerhard REINELT, Giovanni RINALDI a Laurence A. WOLSEY, ed. *50 Years of Integer Programming 1958-2008* [online]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010 [vid. 2021-12-28], s. 7–28. ISBN 978-3-540-68274-5. Dostupné z: doi:10.1007/978-3-540-68279-0_1

[67] JOHNSON, D. a L. A. MCGEOCH. The Traveling Salesman Problem: A Case Study in Local Optimization. *undefined* [online]. 2008 [vid. 2022-01-01]. Dostupné z: https://www.semanticscholar.org/paper/The-Traveling-Salesman-Problem%3A-A-Case-Study-in-Johnson-McGeoch/ff55976d625c5ae6234c5c767a87402a06eb3a43

[68] ROBINSON, Julia. *On the Hamiltonian game (a traveling salesman problem)*. B.m.: RAND PROJECT AIR FORCE ARLINGTON VA. 1949.

[69] MÜLLER-MERBACH, Heiner. Zweimal travelling salesman. *DGOR-Bulletin*. 1983, **25**, 12–13.

[70] MAHALANOBIS, Prasanta Chandra. A sample survey of the acreage under jute in Bengal. *Sankhyā: The Indian Journal of Statistics*. 1940, 511–530.

[71] JESSEN, Raymond James. *Statistical investigation of a sample survey for obtaining farm facts*. B.m.: Iowa State University, 1943.

[72] DANTZIG, George, Ray FULKERSON a Selmer JOHNSON. Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America* [online]. 1954, **2**(4), 393–410. Dostupné z: doi:10/bdjpbx

[73] REINELT, Gerhard. TSPLIB—A Traveling Salesman Problem Library. *ORSA Journal on Computing* [online]. 1991, **3**(4), 376–384. ISSN 0899-1499, 2326-3245. Dostupné z: doi:10/dfcwt5

[74] DUMAS, Yvan, Jacques DESROSIERS, Eric GELINAS a Marius M. SOLOMON. An Optimal Algorithm for the Traveling Salesman Problem with Time Windows. *Operations Research* [online]. 1995, **43**(2), 367–371. ISSN 0030-364X. Dostupné z: doi:10/cjpgcp

[75] LÓPEZ-IBÁÑEZ, M. a C. BLUM. Beam-ACO for the travelling salesman problem with time windows. *Comput. Oper. Res.* [online]. 2010. Dostupné z: doi:10/c9mkjp

[76] PAVLOVIČ, Dávid. *Problém obchodního cestujícího s časovými okny* [online]. B.m., 2021 [vid. 2022-01-02]. Vysoké učení technické v Brně Fakulta strojního inženýrství. Dostupné z: http://hdl.handle.net/11012/197399

[77] BERGER, André, László KOZMA, Matthias MNICH a Roland VINCZE. A time- and space-optimal algorithm for the many-visits TSP. In: *Proceedings of the 2019 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)* [online]. B.m.: Society for Industrial and Applied Mathematics, 2019 [vid. 2022-01-02], Proceedings, s. 1770–1782. Dostupné z: doi:10.1137/1.9781611975482.106

[78] KOWALIK, Łukasz, Shaohua LI, Wojciech NADARA, Marcin SMULEWICZ a Magnus WAHLSTRÖM. Many-visits TSP revisited. *Journal of Computer and System Sciences* [online]. 2022, **124**, 112–128. ISSN 0022-0000. Dostupné z: doi:10/gnx3t3

[79] PUNNEN, Abraham P. The Traveling Salesman Problem: Applications, Formulations and Variations. In: Gregory GUTIN a Abraham P. PUNNEN, ed. *The Traveling Salesman Problem and Its Variations* [online]. Boston, MA: Springer US, 2007 [vid. 2022-01-02], Combinatorial Optimization, s. 1–28. ISBN 978-0-306-48213-7. Dostupné z: doi:10.1007/0-306-48213-4_1; https://web.archive.org/web/20180609055117/https://link.springer.com/chapter/10.1007%2F0-306-48213-4_1

[80] APPLEGATE, David L., Robert E. BIXBY, Vašek CHVÁTAL, William COOK, Daniel G. ESPINOZA, Marcos GOYCOOLEA a Keld HELSGAUN. Certification of an optimal TSP tour through 85,900 cities. *Operations Research Letters* [online]. 2009, **37**(1), 11–15. ISSN 0167-6377. Dostupné z: doi:10/dz37vj

[81] PAPADIMITRIOU, Christos H. a Mihalis YANNAKAKIS. Shortest paths without a map. *Theoretical Computer Science* [online]. 1991, **84**(1), 127–150. Dostupné z: doi:10/cnkbj6

[82] BAR-NOY, Amotz a Baruch SCHIEBER. The canadian traveller problem. In: *Proceedings of the second annual ACM-SIAM symposium on Discrete algorithms*. B.m.: Citeseer, 1991, s. 261–270.

[83] WESTPHAL, Stephan. A note on the k-Canadian traveller problem. *Information Processing Letters* [online]. 2008, **106**(3), 87–89. Dostupné z: doi:10/fk87w5

[84] XU, Yinfeng, Maolin HU, Bing SU, Binhai ZHU a Zhijun ZHU. The Canadian traveller problem and its competitive analysis. *Journal of combinatorial optimization* [online]. 2009, **18**(2), 195–205. Dostupné z: doi:10/b3d6n7

[85] SAHIN, Ömer. AO* and penalty based algorithms for the Canadian traveler problem. *undefined* [online]. 2016 [vid. 2021-12-29]. Dostupné z: https://www.semanticscholar.org/paper/AO*-and-penalty-based-algorithms-for-the-Canadian-Sahin/29e529361e628ecd385efce597f33e2097f32c29

[86] HRDINA, Jaroslav a Radek MATOUSEK. Canadian traveller problem: A note about Game Theory approach. *Mendel*. 2014, **2014**, 403–406.

[87] DEMAINE, Erik D., Yamming HUANG, Chung-Shou LIAO a Kunihiko SADAKANE. Canadians should travel randomly. In: *International Colloquium on Automata, Languages, and Programming* [online]. B.m.: Springer, 2014, s. 380–391. Dostupné z: doi:10/gnxqpk

[88] HUANG, Yamming a Chung-Shou LIAO. The Canadian traveller problem revisited. In: *International Symposium on Algorithms and Computation* [online]. B.m.: Springer, 2012, s. 352–361. Dostupné z: doi:10/gnxqpw

[89] BENDER, Marco a Stephan WESTPHAL. An optimal randomized online algorithm for the k-Canadian Traveller Problem on node-disjoint paths. *Journal of Combinatorial Optimization* [online]. 2015, **30**(1), 87–96. ISSN 1382-6905, 1573-2886. Dostupné z: doi:10/f7fdbj

[90] CHAN, H., J. CHANG, H. WU a T. WU. The k-Canadian Traveller Problem on Equal-Weight Graphs. *Proc. of WCMCT*. 2015, 135–137.

[91] ZHANG, Huili a Yinfeng XU. The k-canadian travelers problem with communication. In: *Frontiers in Algorithmics and Algorithmic Aspects in Information and Management*. B.m.: Springer, 2011, s. 17–28.

[92] LIAO, Chung-Shou a Yamming HUANG. The covering Canadian traveller problem. *Theoretical Computer Science* [online]. 2014, **530**, 80–88. Dostupné z: doi:10/f6zwkj

[93] SU, B. a Y. F. XU. Online recoverable Canadian traveller problem. In: *Proceedings of the international conference on management science and engineering*. 2004, s. 633–639.

[94] SU, Bing, Yinfeng XU, Peng XIAO a Lei TIAN. A risk-reward competitive analysis for the recoverable Canadian traveller problem. In: *International Conference on Combinatorial Optimization and Applications* [online]. B.m.: Springer, 2008, s. 417–426. Dostupné z: doi:10/fnkz4j

[95] LIAO, Chung-Shou a Yamming HUANG. Generalized Canadian traveller problems. *Journal of Combinatorial Optimization* [online]. 2015, **29**(4), 701–712. Dostupné z: doi:10/f67tzw

[96] EYERICH, Patrick, Thomas KELLER a Malte HELMERT. High-quality policies for the canadian traveler's problem. In: *Twenty-Fourth AAAI Conference on Artificial Intelligence*. 2010.

[97] KOCSIS, Levente a Csaba SZEPESVÁRI. Bandit based monte-carlo planning. In: *European conference on machine learning*. B.m.: Springer, 2006, s. 282–293.

[98] FILIP, Sebastián. Řešení problému kanadského cestujícího [online]. nedatováno [vid. 2021-12-29]. Dostupné z: http://dspace.vutbr.cz/handle/11012/67982

[99] KARGER, David a Evdokia NIKOLOVA. Exact algorithms for the Canadian traveller problem on paths and trees. 2008.

[100] NIKOLOVA, Evdokia a David R. KARGER. Route Planning under Uncertainty: The Canadian Traveller Problem. In: *AAAI*. 2008, s. 969–974.

[101] BNAYA, Zahy, Ariel FELNER, Dror FRIED, Olga MAKSIN a Solomon Eyal SHIMONY. Repeated-task Canadian Traveler Problem. *AI Communications* [online]. 2015, **28**(3), 453–477. ISSN 0921-7126. Dostupné z: doi:10/f7k42b

[102] BNAYA, Zahy, Ariel FELNER a Solomon Eyal SHIMONY. Canadian traveler problem with remote sensing. In: *Twenty-First International Joint Conference on Artificial Intelligence*. 2009.

[103] CAMAZINE, Scott, Jean-Louis DENEUBOURG, Nigel R. FRANKS, James SNEYD, Guy THERAULA a Eric BONABEAU. *Self-organization in biological systems*. B.m.: Princeton university press, 2020.

[104] MATOUSEK, Radomil. *Pokročilé metody počítačové inteligence*. Brno, 2012. VUTIUM.

[105] TEODOROVIC, Dušan a Mauro DELL'ORCO. Bee colony optimization–a cooperative learning approach to complex transportation problems. *Advanced OR and AI methods in transportation*. 2005, **51**, 60.

[106] HADDAD, Omid Bozorg, Abbas AFSHAR a Miguel A. MARINO. Honey-bees mating optimization (HBMO) algorithm: a new heuristic approach for water resources optimization. *water resources management* [online]. 2006, **20**(5), 661–680. Dostupné z: doi:10/cjswsz

[107] KRISHNANAND, K. N. a Debasish GHOSE. Glowworm swarm based optimization algorithm for multimodal functions with collective robotics applications. *Multiagent and Grid Systems* [online]. 2006, **2**(3), 209–222. Dostupné z: doi:10/gnxqpx

[108] YAZDANI, Maziar a Fariborz JOLAI. Lion optimization algorithm (LOA): a nature-inspired metaheuristic algorithm. *Journal of computational design and engineering* [online]. 2016, **3**(1), 24–36. Dostupné z: doi:10/gjm5kn

[109] MIRJALILI, Seyedali, Seyed Mohammad MIRJALILI a Andrew LEWIS. Grey wolf optimizer. *Advances in engineering software* [online]. 2014, **69**, 46–61. Dostupné z: doi:10/gfxvkw

[110] BIYANTO, Totok R., Sonny IRAWAN, Henokh Y. FEBRIANTO, Naindar AFDANNY, Ahmad H. RAHMAN, Kevin S. GUNAWAN, Januar AD PRATAMA a Titania N. BETHIANA. Killer whale algorithm: an algorithm inspired by the life of killer whale. *Procedia computer science* [online]. 2017, **124**, 151–157. Dostupné z: doi:10/gnr795

[111] CHU, Shu-Chuan, Pei-wei TSAI a Jeng-Shyang PAN. Cat Swarm Optimization. In: Qiang YANG a Geoff WEBB, ed. *PRICAI 2006: Trends in Artificial Intelligence* [online]. Berlin, Heidelberg: Springer, 2006, s. 854–858. Lecture Notes in Computer Science. ISBN 978-3-540-36668-3. Dostupné z: doi:10/bvqn6w

[112] XIN-SHE, Yang. *Nature-Inspired Optimization Algorithms* [online]. B.m.: Elsevier, 2014 [vid. 2021-12-29]. ISBN 978-0-12-416743-8. Dostupné z: doi:10.1016/C2013-0-01368-0

[113] BOZORG-HADDAD, Omid. *Advanced optimization by nature-inspired algorithms*. B.m.: Springer, 2018.

[114] BANSAL, Jagdish Chand, Pramod Kumar SINGH a Nikhil R. PAL. *Evolutionary and swarm intelligence algorithms*. B.m.: Springer, 2019.

[115] LIM, Siew Mooi a Kuan Yew LEONG. A brief survey on intelligent swarm-based algorithms for solving optimization problems. *Nature-inspired Methods for Stochastic, Robust and Dynamic Optimization* [online]. 2018, **47**. Dostupné z: doi:10/ggm3sz

[116] LONES, Michael A. Mitigating metaphors: A comprehensible guide to recent nature-inspired algorithms. *SN Computer Science* [online]. 2020, **1**(1), 1–12. Dostupné z: doi:10/gnxqpz

[117] ALTHNIAN, Alhanoof a Arvin AGAH. Evolving goal-driven multi-agent communication: what, when, and to whom. *Evolutionary Intelligence* [online]. 2016, **9**(4), 181–202. Dostupné z: doi:10/gnxqph

[118] GRASSE, P.-P. La reconstruction du nid et les coordinations interindividuelles chezBellicositermes natalensis etCubitermes sp. la théorie de la stigmergie: Essai d'interprétation du comportement des termites constructeurs. *Insectes Sociaux* [online]. 2005. Dostupné z: doi:10/fj8r36

[119] SUMPTER, D. a M. BEEKMAN. From nonlinearity to optimality: pheromone trail foraging by ants. *Animal Behaviour* [online]. 2003. Dostupné z: doi:10/fqbcvt

[120] NEWELL, Wilmon. Notes on the Habits of the Argentine or "New Orleans" Ant, Iridomyrmex Humilis Mayr. *Journal of Economic Entomology* [online]. 1908, **1**(1), 21–34. ISSN 0022-0493. Dostupné z: doi:10/gnxqp4

[121] HERBST, Philip. Linepithema humile. *ANTWEB* [online]. [vid. 2020-12-29]. Dostupné z: https://www.antweb.org/description.do?rank=species&genus=linepithema&name=humile

[122] SUNAMURA, E., X. ESPADALER, H. SAKAMOTO, S. SUZUKI, M. TERAYAMA a S. TATSUKI. Intercontinental union of Argentine ants: behavioral relationships among introduced populations in Europe, North America, and Asia. *Insectes Sociaux* [online]. 2009, **56**(2), 143–147. Dostupné z: doi:10/chfrtf

[123] DENEUBOURG, J.-L., Serge ARON, Simon GOSS a Jacques M. PASTEELS. The self-organizing exploratory pattern of the argentine ant. *Journal of insect behavior* [online]. 1990, **3**(2), 159–168. Dostupné z: doi:10/fgqr8g

[124] DORIGO, M. a T. STUTZLE. Ant Colony Optimization. MIT Press, Cambridge, MA. 2004.

[125] GOSS, Simon, Serge ARON, Jean-Louis DENEUBOURG a Jacques Marie PASTEELS. Self-organized shortcuts in the Argentine ant. *Naturwissenschaften* [online]. 1989, **76**(12), 579–581. Dostupné z: doi:10/bxbhm7

[126] *AntSim v1.1 - Ant Colony Algorithms* [online]. [vid. 2021-12-29]. Dostupné z: http://www.nightlab.ch/antsim.php

[127] DORIGO, Marco. Optimization, learning and natural algorithms [Ph. D. thesis]. *Politecnico di Milano, Italy*. 1992.

[128] DORIGO, Marco, Vittorio MANIEZZO a Alberto COLORNI. Positive feedback as a search strategy. 1991.

[129] DORIGO, Marco, Vittorio MANIEZZO a Alberto COLORNI. The ant system: An autocatalytic optimizing process. 1991.

[130] DORIGO, Marco a Luca Maria GAMBARDELLA. Ant colonies for the travelling salesman problem. *biosystems* [online]. 1997, **43**(2), 73–81. Dostupné z: doi:10/ddx7t8

[131] DORIGO, M. a L.M. GAMBARDELLA. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation* [online]. 1997, **1**(1), 53–66. ISSN 1941-0026. Dostupné z: doi:10/d65j6v

[132] STUTZLE, Thomas a Holger HOOS. MAX-MIN ant system and local search for the traveling salesman problem. In: *Proceedings of 1997 IEEE international conference on evolutionary computation (ICEC'97)* [online]. B.m.: IEEE, 1997, s. 309–314. Dostupné z: doi:10/dffzvb

[133] GAMBARDELLA, Luca M. a Marco DORIGO. Ant-Q: A Reinforcement Learning approach to the traveling salesman problem. In: . B.m.: Morgan Kaufmann, 1995, s. 252–260.

[134] REIMANN, Marc, Karl DOERNER a Richard F. HARTL. D-ants: Savings based ants divide and conquer the vehicle routing problem. *Computers & Operations Research* [online]. 2004, **31**(4), 563–591. Dostupné z: doi:10/fd5bbp

[135] FOUNTAS, C. a A. VLACHOS. Ant colonies optimization (ACO) for the solution of the vehicle routing problem (VRP). *Journal of information and optimization sciences* [online]. 2005, **26**(1), 135–142. Dostupné z: doi:10/gnxqpv

[136] MANIEZZO, Vittorio a Alberto COLORNI. The ant system applied to the quadratic assignment problem. *IEEE Transactions on knowledge and data engineering* [online]. 1999, **11**(5), 769–778. Dostupné z: doi:10/c9bs7z

[137] COSTA, Daniele a Alain HERTZ. Ants can colour graphs. *Journal of the operational research society*. 1997, **48**(3), 295–305.

[138] COLORNI, Alberto, Marco DORIGO a Vittorio MANIEZZO. Distributed optimization by ant colonies. In: *Proceedings of the first European conference on artificial life*. B.m.: Paris, France, 1991, s. 134–142.

[139] PFAHRINGER, Bernhard. Multi-agent search for open shop scheduling: Adapting the Ant-Q formalism. *Vienna, Austrian Research Institute for Artificial Intelligence*. 1996.

[140] STÜTZLE, Thomas. An ant approach to the flow shop problem. In: *Proceedings of the 6th European Congress on Intelligent Techniques & Soft Computing (EUFIT'98)*. 1998, s. 1560–1564.

[141] PARPINELLI, Rafael S., Heitor S. LOPES a Alex Alves FREITAS. Data mining with an ant colony optimization algorithm. *IEEE transactions on evolutionary computation* [online]. 2002, **6**(4), 321–332. Dostupné z: doi:10/fd74f8

[142] MARTENS, D., M. DE BACKER, R. HAESEN, J. VANTHIENEN, M. SNOECK a B. BAESENS. Classification With Ant Colony Optimization. *IEEE Transactions on Evolutionary Computation* [online]. 2007, **11**(5), 651–665. ISSN 1089-778X. Dostupné z: doi:10/fsgw8v

[143] DE CAMPOS, Luis, Juan FERNÁNDEZ-LUNA, José GÁMEZ a Jose PUERTA. Ant colony optimization for learning Bayesian networks. *International Journal of Approximate Reasoning* [online]. 2002, **31**, 291–311. Dostupné z: doi:10/dn56vk

[144] SCHOONDERWOERD, R., O. HOLLAND, J. BRUTEN a L. ROTHKRANTZ. Ant-Based Load Balancing in Telecommunications Networks. *Adapt. Behav.* [online]. 1997. Dostupné z: doi:10/d4n8bb

[145] DI CARO, G. a M. DORIGO. AntNet: Distributed Stigmergetic Control for Communications Networks. *Journal Of Artificial Intelligence Research* [online]. 1998 [vid. 2021-12-29]. Dostupné z: doi:10/ggm2zv

[146] VARELA, Griselda Navarro. Ant Colony Optimisation for Virtual-Wavelength-Path Routing and Wavelength Allocation. In: *in Proceedings of the Congress on Evolutionary Computation (CEC'99* [online]. 1999 [vid. 2021-12-29]. Dostupné z: doi:10/bw5z95

[147] OKDEM, Selcuk a Dervis KARABOGA. Routing in wireless sensor networks using an ant colony optimization (ACO) router chip. *Sensors* [online]. 2009, **9**(2), 909–921. Dostupné z: doi:10/cjrgq8

[148] ROSENKRANTZ, Daniel J., Richard E. STEARNS a Philip M. LEWIS. An analysis of several heuristics for the traveling salesman problem. *SIAM journal on computing* [online]. 1977, **6**(3), 563–581. Dostupné z: doi:10/dbzmtr

[149] DORIGO, Marco a Christian BLUM. Ant colony optimization theory: A survey. *Theoretical computer science* [online]. 2005, **344**(2–3), 243–278. Dostupné z: doi:10/c662ht

[150] BULLNHEIMER, Bernd, Richard F. HARTL a Christine STRAUSS. A new rank based version of the Ant System. A computational study. 1997.

[151] CORDON, Oscar, Inaki Fernández DE VIANA, Francisco HERRERA a Llanos MORENO. A new ACO model integrating evolutionary computation concepts: The best-worst Ant System. 2000.

[152] CORDÓN GARCÍA, Oscar, Iñaki FERNÁNDEZ DE VIANA a Francisco HERRERA TRIGUERO. Analysis of the best-worst ant system and its variants on the TSP. *Mathware & soft computing. 2002 Vol. 9 Núm. 2 [-3]*. 2002.

[153] GUNTSCH, Michael a Martin MIDDENDORF. *A Population Based Approach for ACO*. 2002

[154] KOVÁŘÍK, Oleg a Miroslav SKRBEK. Ant colony optimization with castes. In: *International Conference on Artificial Neural Networks*. B.m.: Springer, 2008, s. 435–442.

[155] MELO, Leonor Albuquerque, F. PEREIRA a E. COSTA. Multi-caste Ant Colony Algorithm for the Dynamic Traveling Salesperson Problem. In: *ICANNGA* [online]. 2013. Dostupné z: doi:10/gnxqp2

[156] CROES, Georges A. A method for solving traveling-salesman problems. *Operations research* [online]. 1958, **6**(6), 791–812. Dostupné z: doi:10/fhrjbx

[157] MORTON, G. a A. H. LAND. A Contribution to the "Travelling-Salesman" Problem. *Journal of the Royal Statistical Society: Series B (Methodological)* [online]. 1955, **17**(2), 185–194. Dostupné z: doi:10/gnxqp3

[158] LÓPEZ-IBÁÑEZ, Manuel a Christian BLUM. Beam-ACO Based on Stochastic Sampling: A Case Study on the TSP with Time Windows. In: [online]. 2009, s. 59–73. ISBN 978-3-642-11168-6. Dostupné z: doi:10/fpvfwx

[159] WHEELER, William Morton. *Ants: their structure, development and behavior*. B.m.: Columbia University Press, 1910. 9.

[160] HÖLLDOBLER, Bert a Edward O. WILSON. *Cesta k mravencům*. B.m.: Academia, 1997. ISBN 978-80-200-0612-7.

[161] SHI, Chun-xue, Yingyong BU, Z. LI a J. TAN. Solving path planning problem by an aco-pso hybrid algorithm. *School of Mechanical and Electronical Engineering, Center South University, Changsha, Tech. Rep*. 2007.

[162] TALBI, E.-G., Olivier ROUX, Cyril FONLUPT a Denis ROBILLARD. Parallel ant colonies for the quadratic assignment problem. *Future Generation Computer Systems* [online]. 2001, **17**(4), 441–449. Dostupné z: doi:10/fthmvh

[163] LIU, Shao-Han, Jzau-Sheng LIN a Zi-Sheng LIN. A shortest-path network problem using an annealed ant system algorithm. In: *Fourth Annual ACIS International Conference on Computer and Information Science (ICIS'05)*. B.m.: IEEE, 2005, s. 245–250.

[164] DORIGO, Marco a Thomas STÜTZLE. Ant colony optimization: overview and recent advances. *Handbook of metaheuristics* [online]. 2019, 311–351. Dostupné z: doi:10/gnxqpt

[165] KHICHANE, Madjid, Patrick ALBERT a Christine SOLNON. An ACO-Based Reactive Framework for Ant Colony Optimization: First Experiments on Constraint Satisfaction Problems. In: Thomas STÜTZLE, ed. *Learning and Intelligent Optimization* [online]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009 [vid. 2024-02-22],

Lecture Notes in Computer Science, s. 119–133. ISBN 978-3-642-11168-6. Dostupné z: doi:10.1007/978-3-642-11169-3_9

[166] MSFT. *Parallel Patterns Library (PPL)* [online]. srpen 2021 [vid. 2024-02-27]. Dostupné z: https://learn.microsoft.com/en-us/cpp/parallel/concrt/parallel-patterns-library-ppl?view=msvc-170. Citation Key: msft_parallel_2021

[167] INTEL. *Intel® oneAPI threading building blocks* [online]. srpen 2022 [vid. 2024-02-27]. Dostupné z: https://www.intel.com/content/www/us/en/developer/tools/oneapi/onetbb.html. Citation Key: intel_tbb

[168] CONTRIBUTORS. OpenStreetMap. Planet Dump. *OpenStreetMap. Planet Dump* [online]. [vid. 2021-12-29]. Dostupné z: https://planet.osm.org/

[169] BOEING, Geoff. OSMNX: New Methods for Acquiring, Constructing, Analyzing, and Visualizing Complex Street Networks. *Computers Environment and Urban Systems* [online]. 2017, **65**, 126–139. Dostupné z: doi:10/gbvjxq

ČLÁNKY AUTORA

ŠOUSTEK, P.; MATOUŠEK, R.; DVOŘÁK, J.; BEDNÁŘ, J. Canadian traveller problem: A solution using ant colony optimization. *Mendel.* 2013, 439–444.

MATOUŠEK, R.; ŠOUSTEK, P.; DVOŘÁK, J.; BEDNÁŘ, J. ACO in Task of Canadian Traveller Problem. In *18th International Conference of Soft Computing, MENDEL 2012 Mendel Journal series*. 2012. Brno: VUT, 2012. s. 600-603. ISBN: 978-80-214-4540- 6. ISSN: 1803-3814.

ŠOUSTEK, Petr; Radomil MATOUŠEK; Jiří DVOŘÁK a Lenka MAŇÁKOVÁ. Explanation and speedup comparison of advanced path-planning algorithms presented on two-dimensional grid. online. *Mendel Journal series*, roč. 28 (2022), č. 2, s. 97–107. Dostupné z: https://doi.org/10.13164/mendel.2022.2.097.

ŠOUSTEK, P.; KRČEK, P.; DVOŘÁK, J.; ZUTH, D.; MATOUŠEK, R. Optimalizace cesty pomocí algoritmu RRT v prostředí Matlab. In *Technical Computing 2010.* Bratislava: RT Systems, 2010. s. 95-98. ISBN: 978-80-970519-0-7.

ABBADI, A.; MATOUŠEK, R.; MINÁŘ, P.; ŠOUSTEK, P. RRTs Review and Options. In *Computational Engineering in Systems Applications (*Volume II*)*. Romania: IAASAT Press, 2011. s. 194-199. ISBN: 978-1-61804-014-5.

ŠOUSTEK, P.; MATOUŠEK, R. Moderní čárové kódy. *Automa,* 2012, roč. 18, č. 5, s. 26-29. ISSN: 1210- 9592.

ŠOUSTEK, P.; MATOUŠEK, R.; ABBADI, A. Duqu – skrytá hrozba. *Automa,* 2012, roč. 18, č. 12, s. 10-12. ISSN: 1210- 9592.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

Symbol	Význam
α	faktor posílení dle ohodnocení hran u mravenčích algoritmů
β	faktor posílení dle feromonové stopy u mravenčích algoritmů
ρ	faktor odpaření fer. stopy u mravenčích algoritmů

Zkratka	Význam
ACO	Ant Colony Optimization
CTP	Problém kanadského cestujícího
TSP	Problému obchodního cestujícího
GA	Genetické algoritmy
SLAM	problém simultánní lokalizace a mapování
BA	Bees Algorithm
ABC	Artificial Bee Colony Algorithm
GR	Greedy-Repositio
CR	Cyclic routing
SCTP	Stochastická varianta problému kanadského cestujícího
SRCTP	Stoch. obnov. varianta problému kanadského cestujícího
HOP	Hindsight optimization
ORO	Optimistic Rollout
UCT	Upper Confidence bounds applied to Trees
UCTB	UCT Blind
UCTO	UCT Optimistic
UCTP	UCT Pruning
DAG	Direct acyclic graph
PSO	Particle swarm optimization
BCO	Bee colony optimization
HBMO	Honeybee mating optimization algorithm
GSO	Glowworm swarm optimization
RAS	Ranked ant system
MMAS	MaxMin Ant system
ACS	Ant colony system
ACO+C	Ant colony optimization + castes
MMAS+C	MaxMin Ant System
SA	Simulované žíhání + castes
k-CTP	Neobnovitelná varianta CTP
r-CTP	Recoverable CTP
CTSP	Problému kanadského obchodního cestujícího
ReMMAS	Restarting MaxMin Ant System

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

„He who fights with monsters should look to it that he himself does not become a monster ... when you gaze long into the abyss the abyss also gazes into you."

Friedrich Nietzsche, Beyond Good and Evil