VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA PODNIKATELSKÁ
ÚSTAV MANAGEMENTU

FACULTY OF BUSINESS AND MANAGEMENT
INSTITUTE OF MANAGEMENT

VYUŽITÍ PROSTŘEDKŮ UMĚLÉ INTELIGENCE PRO PODPORU ROZHODOVÁNÍ V PODNIKU

THE USE OF MEANS OF ARTIFICIAL INTELLIGENCE FOR THE DECISION MAKING SUPPORT IN THE FIRM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Ing. ŠTĚPÁN ROSA
AUTHOR

VEDOUCÍ PRÁCE — prof. Ing. PETR DOSTÁL, CSc.
SUPERVISOR

BRNO 2012

Vysoké učení technické v Brně
Fakulta podnikatelská

Akademický rok: 2011/2012
Ústav managementu

ZADÁNÍ DIPLOMOVÉ PRÁCE

Rosa Štěpán, Ing.

Řízení a ekonomika podniku (6208T097)

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách, Studijním a zkušebním řádem VUT v Brně a Směrnicí děkana pro realizaci bakalářských a magisterských studijních programů zadává diplomovou práci s názvem:

Využití prostředků umělé inteligence pro podporu rozhodování v podniku

v anglickém jazyce:

The Use of Means of Artificial Intelligence for the Decision Making Support in the Firm

Pokyny pro vypracování:

Úvod
Vymezení problému a cíle práce
Teoretická východiska práce
Analýza problému a současné situace
Vlastní návrhy řešení, přínos návrhů řešení
Závěr
Seznam použité literatury
Přílohy

Seznam odborné literatury:

ALIEV, A., ALIEV, R. Soft Computing and Its Applications. World Scientific Pub. Ltd, 2002. 444 s. ISBN 981-02-4700-1.
DAVIS, L. Handbook of Genetic Algorithms. Int. Thomson Com. Press, 1991. 385 s. ISBN 1-850-32825-0.
DOSTÁL, P. Pokročilé metody analýz a modelování v podnikatelství a veřejné správě. 1. vyd. Brno : CERM, 2008. 340 s. ISBN 978-80-7204-605-8.
DOSTÁL, P. Advanced Decision Making in Business and Public Services. Brno : CERM, 2011. 168 s., ISBN 978-80-7204-747-5.
THE MATHWORKS. MATLAB – User's Guide. The MathWorks, Inc. 2010.
THE MATHWORKS. MATLAB – Global Optimization Toolbox - User's Guide. The MathWorks, Inc. 2010.

Vedoucí diplomové práce: prof. Ing. Petr Dostál, CSc.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2011/2012.

L.S.

PhDr. Martina Rašticová, Ph.D.	doc. RNDr. Anna Putnová, Ph.D., MBA
Ředitel ústavu	Děkan fakulty

V Brně, dne 20.05.2012

Abstrakt

Diplomová práce se zaměřuje na využití genetických algoritmů pro úlohy vycházející z problému obchodního cestujícího. Na základě teoretických poznatků a analýzy problému poskytuje návrh řešení, které s ohledem na omezující podmínky sestaví denní plán tras pro servisní techniky. Případová studie ukazuje, že navržené řešení v porovnání s plánováním dle zkušeností umožňuje snížit náklady na dopravu.

Abstract

The diploma thesis focuses on the use of genetic algorithms for tasks related to the travelling salesman problem. Based on theoretical knowledge and problem analysis a proposal of the solution is provided. This creates a daily route plan for service technicians with regard to constraints. The case study shows that the proposed solution in comparison with manual scheduling by experience enables to reduce transportation costs.

Klíčová slova

Genetické algoritmy, problém obchodního cestujícího, optimalizace, plánování tras

Key words

Genetic algorithms, traveling salesman problem, optimization, route planning

Bibliografická citace

ROSA, Š. *Využití prostředků umělé inteligence pro podporu rozhodování v podniku.* Brno : Vysoké učení technické v Brně, Fakulta podnikatelská, 2012. 88 s. Vedoucí diplomové práce prof. Ing. Petr Dostál, CSc.

Čestné prohlášení

Prohlašuji, že předložená diplomová práce je původní a zpracoval jsem ji samostatně. Prohlašuji, že citace použitých pramenů je úplná, že jsem ve své práci neporušil autorská práva (ve smyslu Zákona č. 121/2000 Sb., o právu autorském a o právech souvisejících s právem autorským).

V Brně dne 25. května 2012 ……..

Poděkování

Děkuji prof. Ing. Petru Dostálovi, CSc., vedoucímu své diplomové práce, za cenné rady a připomínky při tvorbě této práce a za čas, který mi věnoval.

Dále děkuji panu Janu Pavelcovi ze společnosti VÝTAHY, s.r.o. za informace a konzultace, které mi poskytl.

Obsah

1 Úvod

Dnes a denně se podniky, které mají co do činění s logistikou a přepravou, zabývají otázkou, jak minimalizovat své cestovní náklady při zachování kvality služeb. Společnost, která dokáže pokrýt potřeby zákazníka a zároveň udržet náklady v rozumné míře bude mít jistě konkurenční výhodu.

Tato diplomová práce hledá řešení problému plánování tras servisním technikům. Vychází při tom ze známého problému obchodního cestujícího, který chce každé město ze seznamu měst navštívit právě jednou a vrátit se domů. Otázkou je, v jakém pořadí má daná města navštívit, když chce minimalizovat své cestovní náklady. I když se tento problém zdá na první pohled jednoduchý, pro jeho obecný případ stále nebyla nalezena efektivní metoda řešení [29].

Jedním z přístupů, kterým je možné tento problém řešit, jsou genetické algoritmy. Ty jsou založené na přímé analogii s procesy v živé přírodě a používají se tam, kde by systematické hledání optimálního řešení trvalo téměř nekonečně dlouho. Není garantováno, že se jejich použitím nalezne optimální řešení, nicméně se ukazuje, že jsou obecně vhodné k nalezení přijatelného řešení v přijatelném čase [7, 14].

Po analýze problému plánování tras a požadavků na aplikaci pro podporu rozhodování vznikl návrh a byla vyvinuta aplikace s využitím programu MATLAB, který mimo jiné obsahuje knihovnu právě pro práci s genetickými algoritmy. Jelikož MATLAB představuje poměrně nákladnou investici a ne každý podnik jej vlastní, bylo využito možnosti exportu programu napsaného v MATLABu do javovské komponenty. Tato komponenta pro svůj běh potřebuje tzv. MATLAB Compiler Runtime, který je zdarma ke stažení. V Javě bylo pro komponentu vybudováno přívětivé grafické uživatelské rozhraní a vznikla tak výsledná aplikace umožňující pohodlné denní plánování tras s ohledem na omezující podmínky. V případové studii se ukázalo, že v porovnání s ručním plánováním dle zkušeností umožňuje tato aplikace snížit náklady na dopravu.

Struktura práce je tato: Po úvodní části následuje formulace cílů práce a stručný popis metod a postupů použitých při zpracování diplomové práce. Bezprostředně poté je uvedena kapitola analyzující zvolený podnikatelský subjekt, kde čtenář nalezne

základní informace o podniku, předmětu jeho podnikání, organizační struktuře, historii a provedené SWOT analýze. Další kapitolu tvoří teoretická východiska práce, je tu detailně popsán problém obchodního cestujícího, jeho varianty, výpočetní složitost, aproximační algoritmy a praktické aplikace, ve kterých nachází uplatnění, dále je pak podrobně zpracována teorie týkající se genetických algoritmů a jejich aplikace na řešení problému obchodního cestujícího. Pátá kapitola ve stručnosti představuje použité nástroje. Jedná se jednak o program MATLAB, jeho knihovnu pro genetické algoritmy a nástroj pro vytvoření Java komponenty MATLAB Builder JA, jednak o programovací jazyk Java a vývojové prostředí NetBeans. V šesté kapitole je detailně analyzována problematika plánování tras servisních techniků ve zvolené společnosti a provedena klasifikace s ohledem na problém obchodního cestujícího. Těžiště práce je v další kapitole, kde jsou uvedeny vlastní návrhy řešení a kde je popsána implementace podpůrné aplikace. V této kapitole se také nachází případová studie srovnávající trasy naplánované ručně a trasy naplánované pomocí vytvořeného softwaru. V závěrečné kapitole shrnuji dosažené výsledky ve vztahu ke stanoveným cílům.

2 Cíle práce, metody a postupy zpracování

Hlavním cílem diplomové práce je vytvoření nástroje pro podporu rozhodování managementu v oblasti plánování tras servisním technikům. Tento nástroj by měl rozdělit místa, která mají být v daný den navštívena, mezi týmy servisních techniků a naplánovat jim trasy tak, aby celková ujetá vzdálenost byla minimální s ohledem na časové možnosti a omezení vyplývající mimo jiné z potřebné techniky.

S tímto cílem se pojí důkladná analýza problému, dále pak prostudování literatury a dostupných informací k této problematice. Na základě získaných znalostí je možné udělat si představu, jak daný problém řešit. Poté je třeba hledat nástroje, pomocí kterých je možné řešení realizovat. Jakmile jsou tyto kroky splněny, je možno přistoupit k návrhu a implementaci aplikace. Na závěr je nutné aplikaci otestovat na reálných datech, aby byla ověřena její funkčnost.

3 Charakteristika podnikatelského subjektu

V této kapitole budou uvedeny základní charakteristiky podnikatelského subjektu VÝTAHY, s.r.o., pro který tato diplomová práce řeší problém plánování tras servisním technikům. Po krátkém přestavení společnosti bude věnována pozornost předmětu podnikání a organizační struktuře. Následovat bude zmínka o historii podniku, po ní SWOT analýza.

3.1 O společnosti

Společnost VÝTAHY s.r.o. se zabývá výrobou a montáží výtahů, průmyslových a garážových vrat, posuvných bran a mostových jeřábů. Dále také nabízí generální opravy, rekonstrukce a revizní zkoušky na tato zařízení. V jejím areálu je též prodejna hutních materiálů, která provádí i dělení, ohýbání, případně svařování zakoupeného zboží [30].

Obrázek 3.1: Logo společnosti (Zdroj: [30])

3.2 Základní údaje

Název společnosti:	VÝTAHY, s.r.o.
Právní forma:	společnost s ručením omezeným
Sídlo:	Vrchovecká 216, 594 29 Velké Meziříčí
IČO:	46342354
Datum vzniku:	27. března 1992
Počet zaměstnanců:	cca. 170 osob
Roční obrat (rok 2010):	353 miliónů Kč

[28]

3.3 Předmět podnikání

Dle výpisu z obchodního rejstříku [24] je předmětem podnikání:

- výroba, obchod a služby neuvedené v přílohách 1 až 3 živnostenského zákona
- projektová činnost ve výstavbě
- zámečnictví, nástrojařství
- silniční motorová doprava, a sice nákladní vnitrostátní provozovaná vozidly o největší povolené hmotnosti do 3,5 tuny včetně, nákladní vnitrostátní

provozovaná vozidly o největší povolené hmotnosti nad 3,5 tuny, nákladní mezinárodní provozovaná vozidly o největší povolené hmotnosti do 3,5 tuny včetně, nákladní mezinárodní provozovaná vozidly o největší povolené hmotnosti nad 3,5 tuny

- montáž, opravy, revize a zkoušky zdvihacích zařízení
- hostinská činnost

3.4 Organizační struktura

„Podnik je rozdělen na 8 úseků. Jsou to úseky technický, obchodní, projekce, výroba, montáž, ekonomický úsek, úsek kvality a řízení jakosti a sekretariát. Vedoucí těchto úseků jsou přímo podřízeni řediteli podniku.“ [24]

3.5 Historie společnosti

Výroba a montáž výtahů byla ve Velkém Meziříčí zahájena již v roce 1957. Zanedlouho bylo založeno i učňovské středisko, které vychovávalo mechaniky zdvihacích zařízení pro Moravu a Slovensko. V tehdejším Československu bylo učiliště jedním ze dvou, díky tomu se provozovna zapsala do povědomí.

Provozovna původně sídlila v bývalém Podhradském mlýně, který byl vyvlastněn původnímu majiteli. Několikrát změnila název i majitele. V roce 1992 byla budova v restituci vrácena původnímu majiteli. Zbytek podniku byl zprivatizován a stal se základem nové společnosti VÝTAHY, s.r.o., která přechodně sídlila v pronájmu, nicméně vzhledem k jejímu poměrně rychlému růstu (z původních 60 zaměstnanců stav vzrostl na 160), bylo rozhodnuto o stavbě nového sídla. Z rozestavěného rybářského areálu, který společnost koupila a přestavěla tak v roce 1997 vzniklo nové sídlo.

K zavedeným oborům činnosti, tedy výrobě, montáži a servisu výtahů a průmyslových vrat přibyla výroba mostových jeřábů, garážových vrat a posuvných bran a byla otevřena prodejna hutních materiálů.

Společnost se profilovala jako převážně výrobní a stala se největším výrobcem výtahů v České republice s českým vlastnictvím [11].

3.6 SWOT Analýza

Tabulka 3.1: SWOT Analýza

Silné stránky	Slabé stránky
➢ Dlouholetá praxe v oboru ➢ Hustá síť smluvních servisních partnerů ➢ Působnost na celém území České republiky (i na Slovensku) ➢ Detailní přizpůsobení konkrétnímu místu i požadavkům zákazníka ➢ Vlastní konstrukční a vývojový tým ➢ Držitel řady certifikátů	➢ Softwarové vybavení ➢ Není bezplatná zákaznická linka ➢ Jednojazyčný web (na vícejazyčném se však pracuje) ➢ Obchodní zastoupení pouze v České republice a na Slovensku ➢ Nepřítomnost na sociálních sítích ➢ Sponzorství mimo region vyjímečně
Příležitosti	**Hrozby**
➢ Poptávka po modernizaci výtahů ➢ Zvyšování podílu na českém a slovenském trhu ➢ Vstup na nové zahraniční trhy ➢ Budování povědomí o firmě v zahraničí	➢ Politická a ekonomická situace u nás i v zahraničí ➢ Zvyšování cen energií ➢ Vstupní bariéry na zahraniční trhy ➢ Neakceptování českých certifikátů v zahraničí ➢ Nedostatek kvalifikovaných pracovníků ➢ Vstup silných zahraničních konkurentů na trh

4 Teoretická východiska práce

4.1 Problém obchodního cestujícího (traveling salesman problem, TSP)

Obchodní cestující začíná a končí svou cestu v domovském městě, přičemž chce navštívit každé město z dané množiny měst právě jednou [17]. Otázkou je v jakém pořadí má daná města navštívit, když chce minimalizovat své náklady na cestování – ať už z pohledu času, vzdálenosti či jiných nákladů.

Tento na pohled jednoduše znějící problém je ve skutečnosti jedním z nejintenzivněji zkoumaných problémů ve výpočetní matematice a pro obecný případ stále nebyla nalezena efektivní metoda řešení [29].

4.1.1 Klasifikace TSP

V zásadě se rozlišují tři typy TSP

- symetrický problém obchodního cestujícího
- asymetrický problém obchodního cestujícího
- problém více obchodních cestujících

4.1.1.1 Symetrický problém obchodního cestujícího (symmetric traveling salesman problem, sTSP)

Nechť $V = \{v_1, v_2, \dots, v_n\}$ je množina měst, $A = \{(r,s): r,s \in V\}$ je množina hran a $d_{rs} = d_{sr}$ jsou náklady přiřazené hraně $(r,s) \in A$. Poté sTSP je problém nalezení uzavřené cesty (hamiltonovské kružnice) s minimální délkou takové, že každé město je navštíveno právě jednou [6].

4.1.1.2 Asymetrický problém obchodního cestujícího (asymmetric traveling salesman problem, aTSP)

Pokud platí, že $d_{rs} \neq d_{sr}$ minimálně pro jednu hranu (r,s), jedná se o aTSP [6]. Jednosměrné ulice, objížďky, uzavírky v pozemní dopravě jsou příkladem porušení této symetrie.

4.1.1.3 Problém více obchodních cestujících (multiple traveling salesman problem, mTSP)

Nechť je v množině daných uzlů jeden speciální uzel – základna, ostatní uzly nazvěme přechodné uzly. Pak mTSP sestává z nalezení cest pro všech m obchodních cestujících takových, že všichni začínají a končí svoji cestu v základně a přechodné uzly jsou navštíveny právě jednou a náklady spojené s navštívením všech uzlů jsou minimální.

Možné variace problému jsou následující:

Jedna základna vs. více základen – V případě jedné základny se všichni obchodní cestující vrací do této základny, zatímco v druhém případě se mohou vrátit buď do své původní základny, nebo do jakékoliv jiné za předpokladu, že jejich počet je v každé základně po návratu stejný jako předtím.

Počet obchodních cestujících – Počet obchodních cestujících může být pevná či vázaná proměnná.

Náklady – V případech, kdy počet obchodních cestujících není pevný, jsou obvykle zadány fixní náklady spojené s využitím daného obchodního cestujícího. V tomto případě se cílem také stává minimalizace potřebných obchodních cestujících.

Časový rámec – Toto rozšíření mTSP přidává podmínku, že některé uzly potřebují být navštíveny v určitých časových úsecích nazývaných časová okna. Aplikace této variace nazývané multiple traveling salesman problem with specified timeframe (mTSTTW) lze dobře vidět v problémech plánování letecké dopravy.

Další omezení – Jedná se o omezení týkající se počtu navštívených uzlů, minimální či maximální uražené vzdálenosti daného obchodního cestujícího atd. [6].

4.1.2 Výpočetní složitost

Význam TSP spočívá mj. v tom, že je zástupcem větší třídy kombinatorických optimalizačních úloh známých jako NP - úplných problémů, tedy problémů řešitelných nedeterministickým Turingovým strojem v polynomiálním čase, v případě deterministického Turingova stroje se jedná o exponenciální časovou složitost. Důležité je, že pokud je možné nalézt efektivní algoritmus (tj. algoritmus, u kterého bude zaručeno, že nalezne optimální řešení v polynomiálním počtu kroků) pro problém

obchodního cestujícího, pak lze nalézt efektivní algoritmy také pro další problémy z třídy NP – úplných problémů [12]. Příkladem problémů v této třídě může být SAT problém – problém splnitelnosti booleovských formulí, Knapsack a řada grafových problémů [5].

4.1.2.1 Výzva

Do dnešního dne nikdo nenalezl algoritmus s polynomiální časovou složitostí pro TSP. Objevení takovéhoto algoritmu anebo důkazu, že takový algoritmus neexistuje, by objeviteli přineslo milión amerických dolarů od Clay Mathematics Institute a vyřešilo otázku rovnosti tříd složitosti P a NP [2].

4.1.2.2 Počet možných cest

Zvolme výchozí město, z tohoto města existuje $n-1$ možností pro druhé město, $n-2$ možností pro třetí město atd. Vynásobením těchto čísel dohromady získáme $(n-1)! = (n-1)(n-2)\ldots 3.2.1$ přípustných řešení. V případě symetrické varianty můžeme počet možností dělit dvěma tedy $(n-1)!/2$ [27].

Pro zajímavost uveďme tabulku počtů měst, cest a času potřebného pro výpočet na počítači. Rychle rostoucí hodnoty vylučují možnost řešení hrubou silou tedy vyhodnocením všech možností.

Tabulka 4.1: Výpočetní náročnost problému obchodního cestujícího (Zdroj: [19])

Počet měst	Počet cest	Čas
5	12	12 µs
8	2520	2,5 ms
10	181440	0,18 s
12	19958400	20 s
15	87178291200	12,1 h
18	177843714048000	5,64 let
20	60822550204416000	1927 let

4.1.3 Aproximační algoritmy

Během dlouhé historie TSP byly navrženy různé heuristiky a aproximační algoritmy, které poskytují dobré řešení v krátkém čase. Moderní metody jsou schopné nalézt řešení pro extrémně velké problémy (milióny měst) v přiměřené době. Tato řešení jsou

s velkou pravděpodobností vzdáleny od optimálního řešení pouze 2-3 %. Tyto algoritmy můžeme rozdělit do několika kategorií

- Konstruktivní heuristiky
- Iterativní heuristiky
- Náhodná zlepšení

4.1.3.1 Konstruktivní heuristiky

Tyto algoritmy pomocí jistých konstrukčních pravidel postupně budují řešení, přičemž jeho nalezené části již nemění. Jakmile naleznou řešení, ukončí svůj běh, nikdy se nepokouší řešení vylepšit. Uvádí se, že konstruktivní heuristiky nalézají řešení 10-15 % vzdálené od optimálního, pro některé aplikace jsou užitečné, nicméně obecně nejsou vyhovující [6, 17].

Někteří zástupci této skupiny

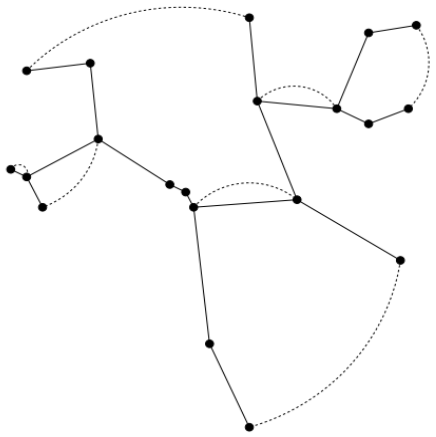
- Heuristika nejbližšího souseda
- Heuristika vkládání
- Christofidova heuristika
- Úsporné heuristiky

Heuristika nejbližšího souseda – Jedná se o nejjednodušší TSP heuristiku. Náhodně se zvolí výchozí město a poté se do cesty přidává vždy „nejbližší“ město ze zbývajících. Složitost této metody je polynomiální $O(n^2)$.

Heuristika vkládání – Další z intuitivních heuristik začíná s cykly o určitém počtu uzlů (většinou třemi) vybraných z množiny měst. Podle určitého kritéria se vybere další město např. město s nejkratší vzdáleností ke kterémukoliv městu již obsaženému v cyklu. Toto město je připojeno ke stávajícímu řešení např. tak, že se najde hrana taková, že když se mezi její uzly vloží uzel představující vybrané město, náklady budou minimální. Takto se pokračuje, dokud zbývají nějaká města. Složitost je polynomiální $O(n^2)$ [6].

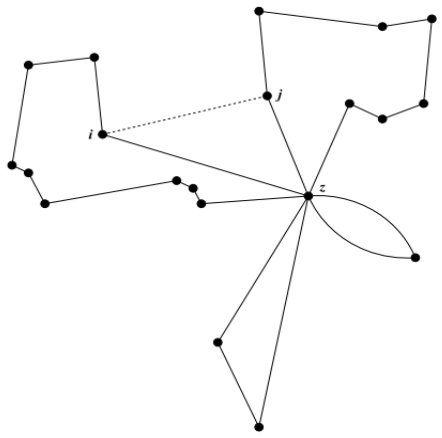
Christofidova heuristika – Tato heuristika je zástupcem heuristik používajících minimální kostru jako základ pro generování hamiltonovské kružnice. Uzly s lichým počtem hran se po dvojicích spojí hranami tak, aby součet jejich ohodnocení byl

minimální, každý uzel je obsažen pouze v jedné hraně. Tyto hrany přidá k minimální kostře. Z vzniklého eulerovského cyklu se vytvoří hamiltonovská kružnice. Složitost tohoto přístupu je $O(n^3)$ [6, 17].



Obrázek 4.1: Christofidova heuristika (Zdroj: [17])

Úsporné heuristiky – Heuristiky původně vyvinuté pro směrování vozidel (vehicle routing problem) aplikovatelné i na TSP postupují tak, že se vybere základní uzel a vytvoří se $n - 1$ hran vedoucích z tohoto uzlu do všech zbývajících. Poté se pro každé dvě podcesty spočítá úspora, které by se dosáhlo při odstranění jedné hrany obsahující základní uzel z každé podcesty a propojením otevřených konců jednou hranou. Spojí se takové dvě podcesty, které poskytují největší úsporu. Složitost je polynomiální $O(n^3)$ [17].



Obrázek 4.2: Úsporná heuristika (Zdroj: [17])

V testu provedeném v [17] dopadly z konstruktivních heuristik nejlépe úsporné heuristiky. Pokud je požadováno podstatně rychlejší, ale ne tak kvalitní řešení, může být použita heuristika nejbližšího souseda. Pro geometrické problémy lze s úspěchem využít heuristiky využívající minimální kostru, např. rychlou variantu Christofidovy heuristiky.

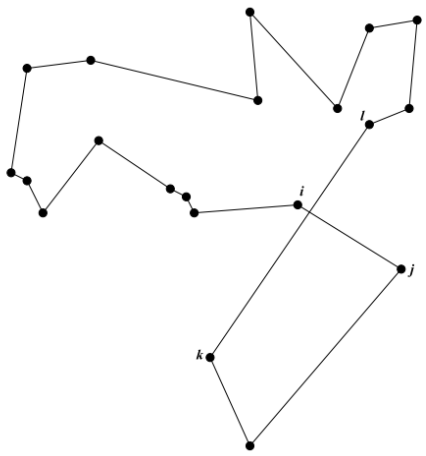
4.1.3.2 Iterativní heuristiky

Jedná se o heuristiky, které iterativně modifikují a pokouší se vylepšit nějaké počáteční řešení (generované např. konstruktivní heuristikou).

Někteří zástupci této skupiny

- 2-opt výměna
- 3-opt výměna
- Lin-Kernighan

2-opt výměna – Při euklidovských problémech bylo vypozorováno, že pokud se hamiltonovská kružnice někde kříží, může být její délka jednoduše zkrácena a to tak, že odstraníme dvě křížící se hrany a nahradíme je hranami, které se nekříží. Iterativně se hledají nejlepší 2-opt výměny, dokud dochází ke zlepšení.



Obrázek 4.3: 2-opt výměna (Zdroj: [17])

3-opt výměna – Tento algoritmus pracuje podobně jako předchozí. Odeberou se tři hrany a části se opět spojí nejlepší způsobem tak, aby tvořili hamiltonovskou kružnici.

Počet kombinací pro odebrání tří hran je $\binom{n}{3}$, pro opětovné spojení osm (za předpokladu, že každá část obsahuje nejméně jednu hranu).

Lin-Kernighan – Tato heuristika je založena na pozorování, že modifikace lehce zvyšující délku cyklu může občas otevřít nové možnosti pro dosažení značného zlepšení později. Základním principem je vytváření složitých modifikací složených z jednodušších, kde ne všechny jednoduché změny musí nutně snižovat délku cyklu. Existuje mnoho variant tohoto principu [17].

Shrnutí testů provedených v [17] je následující. Pro dosažení velmi dobrých výsledků nejsou jednoduché metody se základními změnami dostatečné. Nicméně když už se použijí, doporučuje se začínat na rozumných výchozích řešeních, protože tyto metody nejsou dosti silné na to, aby si poradily s libovolnou počáteční konfigurací. Výsledky získané konstruktivní heuristikou jsou vhodné jako vstup pro jednoduché iterativní heuristiky. Pro získání řešení vzdáleného 1 % od optimálního je nutné použít Lin-Kernighanovu heuristiku, jelikož se dokáže vyhnout špatným lokálním minimům, avšak při aplikaci na rozsáhlejší problémy je třeba věnovat značné úsilí její efektivní implementaci [17].

4.1.3.3 Náhodná zlepšení

Algoritmy popsané v této části se snaží vyhnout lokálním minimům nebo se s nimi vypořádat. Základním prvkem v tomto boji je použití náhody či náhodného hledání na rozdíl od předchozích čistě deterministických heuristik.

Někteří zástupci této skupiny

- Simulované žíhání
- Genetické algoritmy
- Tabu hledání
- Neuronové sítě
- Mravenčí kolonie

Simulované žíhání – Tato metoda se inspiruje jevy nastávajícími ve fyzice při ochlazování kapalin. Při ochlazování kapaliny až do jejího bodu mrazu s cílem získat uspořádanou krystalickou strukturu se postupuje pozvolna a v každém kroku se systému

nechá čas na relaxaci do stavu s minimální energií. V analogii odpovídají přípustná řešení stavům systému. Hodnota účelové funkce se podobá energii systému. Relaxace při určité teplotě je modelována umožněním náhodných změn v aktuálním řešení v závislosti na úrovni teploty. Čím vyšší je teplota, tím větší změny jsou dovoleny. Teplota také přímo úměrně ovlivňuje pravděpodobnost akceptace změn zvyšujících délku cyklu. Obvykle se pro modifikace používají heuristiky 2-opt a 3-opt [17].

Genetické algoritmy – Vývoj tohoto přístupu byl motivován přírodou, jelikož velmi dobrá řešení složitých problémů lze nalézt v ní samotné. Základní genetické algoritmy začnou s náhodně vygenerovanou populací přípustných řešení. Každý jedinec je ohodnocen tzv. fitness funkcí, která určuje, jak dobré řešení jedinec představuje (vhodné jsou např. celková vzdálenost, náklady). Někteří jedinci (nebo všichni) jsou vybráni ke genetické rekombinaci, u některých dojde k mutaci. Následuje výběr jedinců do další populace a celý proces se opakuje, dokud nenastane jedna z ukončujících podmínek [6, 17]. Genetickým algoritmům je věnována podkapitola 274.2.

Tabu hledání – Předchozí heuristiky dovolovaly změny zvyšující délku cyklu, takže bylo možné se v průběhu výpočtu dostat z lokálního minima. Nicméně nebyla použita žádná opatření, aby se heuristice zabránilo v opakovaném navštívení lokálního minima. Tento nedostatek byl začátkem pro vývoj tabu hledání, které obsahuje mechanismus, který tomuto nežádoucímu jevu zabraňuje.

Neuronové sítě – Tento přístup se snaží napodobovat fungování lidského mozku. V podstatě se vytvoří množina neuronů propojených jistých způsobem. Na základě vstupů, které neuron obdrží, je vypočítán jeho výstup, který se šíří do dalších neuronů. Výsledek vypočtený neuronovou sítí se objeví buďto explicitně jako výstup sítě, anebo je dán stavem neuronů. Výpočetní výsledky toho přístupu na TSP zatím nejsou přesvědčivé [17].

Mravenčí kolonie – Metoda, jež se snaží imitovat pohyb mravenců. Tento nápad byl poměrně úspěšně aplikován na TSP a pro malé problémy získal optimální řešení rychle. Mravenci za sebou při zkoumání nových oblastí zanechávají stopu feromonů. Tato stopa by měla vést další mravence k možnému zdroji potravy. Při optimalizaci pomocí mravenčí kolonie se začíná typicky s množstvím kolem dvaceti mravenců. Jsou

náhodně rozmístěni v městech, která chceme navštívit a jejich úkol je navštívit další města. Jednotlivec nesmí vstoupit do města, které již navštívil. Výjimku tvoří jeho výchozí město při dokončení jeho trasy. Mravenec, který při svém putování zvolil nejkratší cestu, zanechá na cestě stopu feromonů nepřímo úměrnou délce trasy. Tato stopa bude brána v úvahu při rozhodování dalších mravenců, do kterého města se mají vydat. Nejpravděpodobnější cesta je s největším množstvím feromonů. Tento proces je opakován, dokud není nalezena dostatečně krátká cesta [6].

Přístupy uvedené v této kapitole mají velkou výhodu v tom, že jsou obecně aplikovatelné na kombinatorické optimalizační úlohy a další typy problémů. Mohou být implementovány běžně s malou znalostí o struktuře problému. Pokud máme k dispozici dostatek výpočetního výkonu, mohou být aplikovány (po určitém čase stráveném laděním parametrů) na velké problémy a existuje velká šance na nalezení řešení, která budou blízko řešením optimálním [17].

4.1.4 Praktické aplikace

TSP nachází využití přirozeně v přepravních a logistických aplikacích, ale nejen v nich. I když nejpřirozenějším prostředím pro TSP jsou dopravní aplikace, jednoduchost modelu vedla k mnoha zajímavým aplikacím i v jiných oblastech [3].

Některé z důležitých aplikací

- Vrtání desek plošných spojů
- Krystalografie pomocí rentgenových paprsků
- Kompletování objednávek ve skladech
- Směrování vozidel
- Jiné

4.1.4.1 Vrtání desek plošných spojů

Problém vrtání je aplikace TSP, která hraje důležitou ekonomickou roli ve výrobě desek plošných spojů. Je třeba vrtat díry různých průměrů, změna vrtáku je časově náročná. Proto se nejprve vrtají díry určitého průměru, změní se vrták, vrtají se díry jiného průměru atd. Problém je tedy rozdělen na sekvenci TSP, každá aplikace pro jeden průměr. V analogii města představují polohy děr a náklady jsou čas potřebný k pohybu vrtací hlavy mezi polohami. Cílem je minimalizovat čas pohybu hlavy.

4.1.4.2 Krystalografie pomocí rentgenových paprsků

Jedná se o významnou aplikaci TSP při analýze struktury krystalů. K získání informací o struktuře krystalického materiálu se používá rentgenový difraktometr. Detektorem se měří intenzita odražených rentgenových paprsků na různých pozicích. Zatímco měření je velmi rychlé, čas na umístění detektoru představuje značnou režii vzhledem k tomu, že pro některé experimenty je třeba vyhodnotit až stovky tisíc pozic. Ačkoliv výsledek experimentu není závislý na pořadí, ve kterém jsou dané pozice nastavovány, čas pro vykonání experimentu ano. Čas potřebný pro přesun z jedné pozice na druhou lze přesně vypočíst. Problém sestává z nalezení sekvence pozic, při které je minimalizován celkový čas nastavení těchto pozic.

4.1.4.3 Kompletování objednávek ve skladech

Na sklad dorazí objednávka některých položek. Je třeba posbírat všechny položky objednávky. Umístění položek ve skladu odpovídá uzlům v grafu. Vzdálenost mezi dvěma uzly je dána časem potřebným pro přesun z jednoho místa na druhé. Nalezení nejrychlejší cesty může být řešeno jako TSP.

4.1.4.4 Směrování vozidel

Předpokládejme, že v nějakém městě je třeba každý den za určitý časový úsek, např. 1 hodinu, vyprázdnit n poštovních schránek. Problémem je najít minimální počet vozidel a minimální čas, za který je schopný tento počet vozidel daný úkol splnit.

Jako další příklad může posloužit situace, kdy n zákazníků požaduje určité množství komodit a dodavatel musí uspokojit jejich poptávku s danou flotilou nákladních automobilů. Problémem je přiřadit zákazníky k nákladním automobilům a zpracovat plán pro každý vůz tak, že není překročena jeho kapacita a celková najetá vzdálenost je minimální.

Tyto problémy lze řešit jako mTSP, pokud nejsou zadaná kapacitní a časová omezení a pokud je počet vozidel fixní [17].

4.1.4.5 Další

Jako další příklady aplikací z praxe, kde TSP nachází využití, jmenujme výpočet DNA sekvencí, generální opravy turbínových motorů, výrobu desek s tištěnými spoji,

klastrování datových polí, rozvoz jídel, kontrolu pohybu výrobních robotů, směrování školních autobusů, plánování tisku atd. [2, 3, 6, 12, 17].

4.2 Genetické algoritmy

Genetické algoritmy se používají tam, kde by systematickým prohledáváním prostoru řešení trvalo téměř nekonečně dlouho přesné řešení nalézt. Nelze garantovat, že se jejich použitím nalezne optimální řešení, nicméně se ukazuje, že jsou obecně vhodné k nalezení přijatelného řešení v přijatelném čase. Průkopníkem v této oblasti se stal John Holland, jehož genetický algoritmus byl založen na přímé analogii s procesy, které Charles Darwin odhalil v živé přírodě.

V evolučním vývoji se prosazují „silnější“ jedinci, kteří jsou lépe geneticky vybavení a mají tak větší šanci na přežití a vlastní reprodukci. Vhodnou kombinací rodičovských vlastností, lze navíc docílit toho, že potomek bude mít lepší vlastnosti než kterýkoliv z rodičů. U zrodu genetických algoritmů stála myšlenka, že by pro získání lepších řešení složitých problémů bylo možné kombinovat části existujících řešení [7, 14].

4.2.1 Princip

Genetický algoritmus pracuje s populací jedinců, kteří reprezentují vhodným způsobem zakódovaná řešení. Každý jedinec je ohodnocen jistou funkcí, která určuje, jak dobré řešení daný jedince představuje a také jeho reprodukční předpoklady. Imitací přirozeného výběru prostřednictvím operátorů křížení, mutace a inverze je jedincům umožněna reprodukce. V závislosti na použité strategii vzniká z původní populace a jejího potomstva populace nová. Po určitém počtu generací obvykle vznikne populace s jedním nebo i více jedinci, kteří odpovídají přijatelnému někdy dokonce i optimálnímu řešení [14].

4.2.2 Terminologie

Jedince nazveme fenotypem a jeho reprezentaci genotypem. Každá buňka organizmu daného druhu obsahuje určitý počet chromozomů. Chromozom se dělí na jednotlivé lineárně uspořádané geny. Každý gen řídí dědičnost jednoho nebo několika znaků, jeho pozice v chromozomu se nazývá locus a hodnoty, kterých nabývá, označujeme alely. V genetických algoritmech jsou jedinci často charakterizováni pouze jedním

chromozomem. V těchto případech je zjednodušení, kdy se chromozom považuje za genotyp korektní [7, 14].

4.2.3 Algoritmus

V literatuře lze nalézt různé definice genetického algoritmu, které se v některých detailech liší, nicméně obecně lze popsat algoritmus následujícími kroky:

1. Algoritmus začne vytvořením náhodné prvotní populace.
2. Poté algoritmus vytváří sekvence nových populací. V každém kroku použije jednotlivce současné generace pro vytvoření generace nové. Při vytváření nové populace si algoritmus počíná následovně:
    a. Ohodnotí každého člena současné populace vyhodnocením fitness funkce.
    b. Vybere členy, zvané rodiče, na základě jejich hodnoty fitness funkce.
    c. Z rodičů vytvoří potomky kombinací páru rodičů (křížení) anebo provedením náhodných změn v jednom rodiči (mutace).
    d. Vytvoří novou populaci na základě původní populace a množiny nově vzniklých potomků.
3. Algoritmus končí, je-li splněno jedno z ukončujících kritérií, jinak pokračuje krokem 2 [7, 13, 14].

4.2.4 Kódování jedinců

Způsob, jakým jsou jedinci představující řešení daného problému zakódováni, je velmi důležitý pro úspěch genetického algoritmu.

Většina algoritmů pracuje s původním tzv. **binárním kódováním**, kde je chromozom reprezentován jako binární řetězec např. 010010110. Pokud gen nabývá více než dvou hodnot, lze tyto hodnoty zapsat více bity. Každých l bitů umožňuje vyjádřit 2^l různých alel. Zde však dochází k problému, který může výrazným způsobem ovlivnit chování genetického algoritmu. Většinou se totiž předpokládá, že nepatrná změna vlastností jedince se projeví jen nepatrnou změnou v jeho chromozomu a naopak. Tento předpoklad však u klasického binárního kódování není splněn. Proto byl zaveden tzv. **Grayův kód**, který má tu vlastnost, že dvě sousední hodnoty jsou zakódovány

binárními řetězci tak, že se liší právě v jednom bitu. Ukázka zakódování osmi celých čísel oběma způsoby je uvedena v tabulce 4.2.

Tabulka 4.2: Porovnání binárního a Grayova kódu (Zdroj: [14])

Číslo	Binární kód	Grayův kód
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

V mnohých případech je však lepší použít přímo **celá**, nebo dokonce **reálná čísla**. Při řešení různých kombinatorických a plánovacích úloh se velmi často používá tzv. **permutační kódování**. V tomto kódování je jedinec reprezentován permutací několika čísel. Tato permutace určuje pořadí jednotlivých objektů v řešení daného problému [14].

4.2.5 Počáteční populace

Počáteční populace se většinou získává náhodným generováním. Byly provedeny také pokusy nasadit do počáteční populace kvalitní řešení a urychlit tak hledání lepšího řešení. V tomto případě je třeba dát pozor na zvýšenou pravděpodobnost konvergence do lokálních optim. Experimenty naznačují, že co se týče velikosti, je v mnohých případech dostačující populace čítající kolem sta jedinců, resp. mezi n a $2n$, kde n je délka binárního řetězce [7].

4.2.6 Fitness Funkce

Fitness funkce bývá často přímo účelová funkce či její vhodná modifikace, která slouží k ohodnocení kvality řešení, jež daný jedinec reprezentuje. Musí být definována pro všechny jedince a konstruována tak, aby její hodnota byla tím vyšší, čím lepší je hodnota účelové funkce. Je třeba, aby fitness funkce byla správně navržená, jinak by algoritmus konvergoval do nevhodných řešení, anebo nekonvergoval vůbec. Jelikož se

v průběhu genetického algoritmu bude vyhodnocovat mnohokrát, její výpočet musí být rychlý. Preferují se jedinci s vyšší hodnotou fitness funkce [7, 8, 14].

4.2.7 Selekční mechanizmus

Imituje proces přirozeného výběru. Z dané populace vybírá jedince vhodné k další reprodukci. Důležité je, aby selekční mechanizmus na jedné straně preferoval jedince s dobrou hodnotou fitness funkce, ale na straně druhé udržoval populaci dostatečně různorodou, aby nedocházelo k předčasné konvergenci. Tento fakt lze popsat tzv. selektivním tlakem. Čím vyšší je hodnota selektivního tlaku, tím rychleji algoritmus konverguje, ale tím více hrozí nebezpečí předčasné konvergence. Nejčastější formou realizace selekce je tzv. ruletový mechanizmus. Jedinci s vyšším ohodnocením mají na pomyslné ruletě přiřazenou větší výseč a existuje tedy větší pravděpodobnost, že budou vybráni [7, 14].

Metody selekce

- Proporcionální selekce
- Oříznutá selekce
- Lineární pořadí
- Exponenciální pořadí
- Turnajová selekce

Proporcionální selekce – Pravděpodobnost výběru i-tého jedince v závislosti na hodnotě fitness funkce f_i je $p_i = \frac{f_i}{\sum_{j=0}^{n} f_j}$. Nevýhoda této selekce spočívá v problému tzv. předčasné konvergence nastávajícím obvykle při přítomnosti jednoho či více silných jedinců. Klesá tak rozmanitost populace a algoritmus snadno uvízne v lokálním optimu [7, 14].

Oříznutá selekce – Celá populace jedinců se setřídí sestupně dle hodnoty fitness funkce. Do další generace je vybrán určitý počet nejlepších jedinců. Při této selekci opět dochází k poklesu rozmanitosti [7].

Lineární pořadí – Tato metoda selekce rovněž vychází ze setříděné populace. Nejhorší jedinec je označen indexem 1 a nejlepší indexem N. Pravděpodobnost výběru i-tého

jedince je $p_i = \frac{1}{N}\left(\eta^- + (\eta^+ - \eta^-).\frac{i-1}{N-1}\right)$, $i \in \{1, 2, \dots, N\}$, kde $\frac{\eta^-}{N}$ resp. $\frac{\eta^+}{N}$ představuje pravděpodobnost výběru nejhoršího resp. nejlepšího jedince dané populace. Tato metoda potlačuje preferenci nadprůměrně ohodnocených jedinců [7, 14].

Exponenciální pořadí – Princip je podobný jako v předchozí metodě s tím rozdílem, že pravděpodobnost výběru jedince je rozložena exponenciálně $p_i = \frac{c^{N-i}}{\sum_{j=1}^{N} c^{N-j}}$, $i \in \{1, 2, \dots, N\}$, kde se konstanta c volí v rozsahu $c \in (0, 1)$ [7].

Turnajová selekce – Z populace se náhodně vybere skupina t jedinců, $t \geq 2$, ze kterých je vybrán jedinec s nejlepším ohodnocením [14].

4.2.8 Operátor křížení

Pro dva rodiče a dva potomky lze operátor křížení symbolicky zapsat ve tvaru $O_c: X^2 \to X^2$. Je to tedy operátor, který dvěma rodičovským jedincům $x_1, x_2 \in X$ přiřazuje náhodně nový pár potomků $(x_1', x_2') = O_c(x_1, x_2)$. Noví jedinci obvykle vznikají výměnou částí rodičů mezi sebou a obsahují tak genetické informace z obou rodičů. Ke křížení dochází obvykle s 95% pravděpodobností [7], někdy totiž může být potřebné a žádoucí dát jedincům možnost přejít do další populace beze změny. Existuje velké množství různých způsobů křížení obecných i závislých na zvolené reprezentaci [14].

Některé typy binárních křížení

- Jednobodové křížení
- k-bodové křížení
- Uniformní křížení

Jednobodové křížení – Jedná se o nejjednodušší způsob křížení. Náhodně se zvolí bod křížení c, od kterého se zbylé části rodičovských chromozomů mezi sebou zamění. c by mělo být z množiny $\{1, \dots, l-1\}$, kde l je délka chromozomu, jinak by mezi rodiči nedošlo k výměně žádné informace a potomci by byli přímou kopií svých rodičů. Na obrázku 4.4 je bod křížení c reprezentován znakem | a nachází se na čtvrté pozici.

Rodiče		Potomci
(**1, 0, 0, 0, | 1, 1, 1, 1**)	→	(**1, 0, 0, 0,** | 0, 0, 0, 0)
(0, 0, 1, 1, | 0, 0, 0, 0)	→	(0, 0, 1, 1, | **1, 1, 1, 1**)

Obrázek 4.4: Jednobodové křížení (Vlastní zpracování dle: [14])

k-bodové křížení – Je zobecněnou variantou jednobodového křížení. V tomto případě se vygeneruje více bodů křížení a části rodičovských chromozomů se vymění způsobem naznačeným na obrázku 4.5 [14].

Rodiče		Potomci
(**1, 0, | 0, 0, | 1, 1, 1, 1**)	→	(**1, 0, |** 1, 1, | **1, 1, 1, 1**)
(0, 0, | 1, 1, | 0, 0, 0, 0)	→	(0, 0, | **0, 0,** | 0, 0, 0, 0)

Obrázek 4.5: k-bodové křížení (Vlastní zpracování dle: [14])

Uniformní křížení – V případě tohoto křížení, které je zobecněním k-bodového křížení, se při vzniku potomků postupuje tak, že se prochází současně oba rodičovské chromozomy o délce n genů a s pravděpodobností p_{uc} dojde k výměně příslušných genů na aktuální pozici. Uniformní křížení je dobré pro zamezení předčasné konvergence, nicméně nezachovává ucelenější části rodičovských genů. Princip křížení je znázorněn na obrázku 4.6 [7, 14].

Rodiče		Potomci
(**1, 0, 0, 0, 1, 1, 1, 1**)	→	(**1, 0,** 1, **0,** 0, **1,** 0, 0)
(0, 0, 1, 1, 0, 0, 0, 0)	→	(0, 0, **0,** 1, **1,** 0, **1, 1**)

Obrázek 4.6: Uniformní křížení (Vlastní zpracování dle: [14])

Dosud uvedené operátory lze aplikovat také na reprezentace využívající celá či reálná čísla. Reprezentace pomocí reálných čísel navíc dovoluje zavést další typy křížení, kde nová alela může nabývat hodnot např. aritmetického průměru rodičovských alel, odmocniny součinu alel rodičů atd. [14] Některé z operátorů pro permutační reprezentaci budou uvedeny v kapitole 4.3.2.

4.2.9 Operátor mutace

Tento operátor lze symbolicky zapsat ve tvaru $O_m: X \rightarrow X$. Každému chromozomu $x \in X$ přiřazuje náhodně chromozom $x' = O_m(x)$ takový, že $x' \in X$. Tento operátor většinou jednoduchým způsobem náhodně mění hodnotu jednotlivých genů a nastává s malou pravděpodobností, obvykle 0,001 – 0,05. Slouží jako zdroj nových genetických informací v populaci, který brání předčasné konvergenci. Příliš častá mutace může vyvolat nestabilitu, nedostatečná naopak nemusí přinést potřebnou variabilitu. Nejznámějším případem mutace pro binární reprezentaci je **bitová negace**, jež náhodně mění hodnotu jednotlivých genů (obrázek 4.7 nahoře). Další případem může být např. **výměnná mutace** (obrázek 4.7 uprostřed), která prohodí dva náhodně zvolené geny či **inverzní mutace** (obrázek 4.7 dole), která invertuje pořadí bitů mezi dvěma náhodně zvolenými body chromozomu [7, 14].

Před mutací		Po mutaci
(1, 0, **0**, 0, 1, 1, 1, 1)	→	(1, 0, **1**, 0, 1, 1, 1, 1)

Před mutací		Po mutaci
(1, 0, **0**, 0, 1, **1**, 1, 1)	→	(1, 0, **1**, 0, 1, **0**, 1, 1)

Před mutací		Po mutaci
(1, 0, **0, 0, 1, 1,** 1, 1)	→	(1, 0, **1, 1, 0, 0,** 1, 1)

Obrázek 4.7: Bitová negace, výměnná mutace a inverzní mutace (Vlastní zpracování dle [18])

V případě reprezentace jedinců reálnými hodnotami může být mutace provedena např. nahrazením náhodně vybraného genu náhodně vygenerovanou hodnotou, součtem či vynásobením původní alely s náhodně vygenerovanou hodnotou ve vhodně zvoleném intervalu [14].

U permutačního kódování je třeba vzít v úvahu omezení, vyplývající z této reprezentace, tedy dát pozor na to, aby z původní permutace nezmizely některé hodnoty anebo se některé hodnoty po aplikování operátoru mutace nevyskytovaly vícekrát [14]. Ukázky vybraných mutací pro permutační kódování jsou v podkapitole 4.3.2.

4.2.10 Vytvoření nové populace

Existuje několik možností vytvoření nové populace. Jedním z přístupu je, že původní generace zcela vymře a nová generace se bude skládat pouze z jedinců, kteří vznikli jako potomci oné původní generace. Zde hrozí nebezpečí, že slibní jedinci mohou být navždy ztraceni, pokud neprojdou procesem selekce nebo budou změněni pomocí operátorů křížení a mutace. Tato nevýhoda může být eliminována, pokud se určitému počtu nadějných jedinců dovolí přejít do další populace v nezměněné podobě. Existují dva takové přístupy. První z nich, tzv. **elitizmus** umožňuje několika nejlepším jedincům přežít a automaticky je zařadí do nové populace, druhý tzv. **setrvalý stav** naopak uchovává většinu populace v původní podobě a pouze několik nejhůře ohodnocených jedinců je v každé generaci nahrazeno nově vytvořenými potomky. Dochází tak k uchování nejlepší části aktuální generace, ale také k pomalému vývoji.

V některých případech jsou podmínky pro vstup nových potomků do vznikající generace ještě více zpřísněny a jsou akceptováni pouze jedinci, kteří mají vyšší ohodnocení než aktuálně nejslabší jedinec v populaci. Ten pak uvolní místo novému jedinci. Jindy je zase žádoucí mít co možná nejrozmanitější populaci, proto pokud se v populaci vyskytuje více stejných jedinců, je ponechán pouze jeden z nich bez ohledu na jeho ohodnocení. V některých případech se různými způsoby omezuje i množství podobných jedinců [14].

4.2.11 Ukončující kritéria

Ukončujícím kritériem může být např. počet generací, časový limit běhu algoritmu, hodnota fitness funkce či nemožnost získání lepšího řešení po určitý počet generací nebo zvolenou dobu [13].

4.2.12 Omezující podmínky

V reálném světě má naprostá většina optimalizačních úloh určitá omezení. Tato omezení pak dělí prostor řešení na množinu přípustných a množinu nepřípustných řešení. Existuje několik přístupů, jak se s omezujícími podmínkami vyrovnat. Jedním hojně využívaným způsobem je penalizace jedinců představujících nepřípustné řešení. Další skupinu tvoří tzv. opravné algoritmy, které dokážou nahradit nepřípustná řešení přípustnými, anebo využití speciálních genetických operátorů, které zajišťují, že jejich aplikací na přípustné jedince vzniknou opět přípustní jedinci. Poslední skupina obsahuje

metody, jež využívají tzv. dekodéry. Namísto prohledávání původního prostoru všech řešení se prohledává jiný prostor. Chromozom v tomto jiném prostoru není chápán jako zakódované řešení, nýbrž jako informace, na základě které je dekodér schopný získat přístupné řešení v původním prostoru [14].

4.2.13 Paralelní genetické algoritmy

Díky tomu, že genetický algoritmus pracuje s populacemi řešení, se nabízí možnost paralelizace, pomocí které lze zkrátit celkový čas potřebný pro běh algoritmu. Některé způsoby používají paralelizaci na úrovni fitness funkce. V tomto případě se na řídicím procesoru vykonává vlastní genetický algoritmus, provádí se zde selekce i aplikace operátorů a pouze výpočet fitness funkce je distribuován na další procesory. Další metody využívají pomocné procesory i pro genetické operátory anebo selekci. Zajímavější možnosti zpracování však nabízí paralelizace na úrovni populace. Zde jsou na jednotlivé procesory distribuovány celé subpopulace, které mohou být nezávislé anebo spolu mohou komunikovat a vyměňovat si vybrané jedince. Jednotlivé subpopulace se vyvíjejí samostatně, lze tedy na ně aplikovat různé parametry, odlišné způsoby selekce či reprodukce [7, 14].

4.3 Genetické algoritmy a TSP

4.3.1 Reprezentace TSP

Pro problém obchodního cestujícího bylo použito mnoho různých reprezentací. Nejpřirozenější a v současnosti nejpoužívanější je reprezentace cesty [18], pro kterou budou uvedeny i některé z operátorů.

4.3.1.1 Binární reprezentace

V binární reprezentaci TSP s n městy je každé město zakódováno jako řetězec $\lceil \log_2 n \rceil$ bitů. Např. při TSP s 6 městy jsou města reprezentována jako 3bitové řetězce (viz tabulka 4.3) [18]. Tato reprezentace se z praktických důvodů neprosadila. Mezi nejvážnější nedostatky lze zařadit nepřirozený charakter binární reprezentace, značný destrukční vliv klasických genetických operátorů a z toho důvodu nutnost použití časově náročných opravných algoritmů [14].

Tabulka 4.3: Binární reprezentace TSP s 6 městy (Zdroj: [18])

i	Město i	i	Město i
1	000	4	011
2	001	5	100
3	010	6	101

Ukázka: cestu

$$1 - 2 - 3 - 4 - 5 - 6$$

zakódujeme jako

$$(000\ 001\ 010\ 011\ 100\ 101)$$

Pozn. V tomto případě existují 3bitové řetězce, jež nepředstavují žádné město (110, 111).

4.3.1.2 Sousedská reprezentace

V této reprezentaci se číslo i vyskytuje na j-té pozici právě tehdy, když cesta reprezentovaná tímto vektorem obsahuje přesun z města j do města i. Každá okružní cesta je reprezentována jedním vektorem, ale ne všechny vektory obsahující n čísel představují přípustná řešení [14]. Je vidět, že klasický operátor křížení může produkovat nepřípustná řešení. Z tohoto důvodu byly vyvinuty speciální operátory a opravné algoritmy [18]. I přes tuto skutečnost se sousedská reprezentace neukázala jako dostatečně efektivní pro použití k řešení TSP [14].

Ukázka: Vektor

$$(3, 6, 4, 2, 1, 5)$$

reprezentuje okružní cestu

$$1 - 3 - 4 - 2 - 6 - 5 - 1$$

4.3.1.3 Ordinální reprezentace

Cesta je kódována jako seznam n měst, kde i-té číslo v seznamu může nabývat hodnoty $j \in \{1, \dots, (n - i + 1)\}$. K interpretaci tohoto seznamu složí referenční seznam měst, ve

kterém jsou města uložena v daném pořadí [14]. Výhodou tohoto přístupu je fakt, že může být použitý klasický operátor křížení. Je vidět, že se částečná cesta vlevo od bodu křížení nemění, zatímco částečná cesta vpravo je změněna náhodně. Experimentální výsledky při aplikaci na TSP jsou díky tomu obecně nízké [18].

Ukázka: referenční seznam

$$R = (1, 2, 3, 4, 5, 6)$$

cesta

$$1 - 3 - 4 - 2 - 6 - 5$$

reprezentace chromozomem

$$(1, 2, 2, 1, 2, 1)$$

Tento chromozom je interpretován pomocí referenčního seznamu tak, že při vyhodnocování i-tého genu s hodnotou k se vyjme k-tý prvek z referenčního seznamu a přidá se k postupně vytvářené cestě. Ve výše uvedeném příkladě první gen s hodnotou 1 znamená, že cesta bude začínat v prvním prvku referenčního seznamu, tedy ve městě číslo 1, přidáme toto město do vznikající cesty a vyjmeme jej z referenčního seznamu. Druhý gen s hodnotou 2 představuje druhý prvek upraveného referenčního seznamu, tedy město číslo 3. Pokračujeme tímto způsobem, až nakonec získáme finální okružní cestu [14]

$$(1 - 3 - 4 - 2 - 6 - 5 - 1)$$

4.3.1.4 Reprezentace cesty

Jedná se o pravděpodobně nejpřirozenější a nejúspěšnější způsob kódování. Cesta je reprezentována jako seznam n měst. V tomto seznamu jsou města uvedena v pořadí, ve kterém se mají navštívit. Tedy když je město i j-tým prvkem seznamu, má být navštíveno jako j-té. Jelikož klasické operátory pro tuto reprezentaci nejsou vhodné, byly vytvořeny speciální [18]. Některým z nich je věnována podkapitola 4.3.2.

Ukázka: vektor

$$(5, 6, 3, 1, 4, 2)$$

reprezentuje cestu

$$5 - 6 - 3 - 1 - 4 - 2 - 5$$

4.3.1.5 Maticová reprezentace

Bylo učiněno několik pokusů, jak použít binární maticovou reprezentaci. Jedna reprezentace kóduje cestu tak, že prvek na řádku i a sloupci j je 1, právě když město i je navštíveno před městem j. Např. cesta 2 – 3 – 1 – 4 je pak zakódována maticí

$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

Pro tuto reprezentaci byly vytvořeny operátory křížení průnikem a sjednocením.

Další reprezentace definovala, že prvek matice na i-tém řádku a j-tém sloupci je 1, právě když je město j navštíveno hned po městě i. Z toho vyplývá, že legální cesta je reprezentována maticí, která v každém řádku a v každém sloupci obsahuje právě jednu 1. Je nutno podotknout, že ne každá matice, která splňuje tuto podmínku, představuje legální cestu. Cesta 2 – 3 – 1 – 4 je v tomto případě reprezentována maticí

$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}.$$

Byly použity různé způsoby křížení a mutací. Po aplikování těchto operátorů mohou vzniknout matice reprezentující nelegální cesty (obsahující několik podcest). Jedni ihned použili opravné algoritmy, druzí za určitých podmínek tyto matice povolili (pokud podcesty obsahovali více než n měst, n je parametr) a po skončení genetického algoritmu deterministicky nalezené řešení převedli na přípustné řešení [18].

4.3.2 Vybrané genetické operátory pro reprezentaci cesty

V této podkapitole budou uvedeny některé z úspěšných operátorů aplikovaných na TSP.

4.3.2.1 Operátor křížení s částečným přiřazením (Partially Mapped Crossover)

Tento operátor zachovává úseky cest a částečně i pořadí měst. Postup vypadá takto: Nejprve se náhodně s uniformním rozdělením pravděpodobnosti vyberou dva body křížení (v našem případě za třetím a šestým prvkem).

(1 2 3 | 4 5 6 | 7 8)
(3 7 5 | 1 6 8 | 2 4)

Podřetězce mezi body křížení představují mapování (v našem případě 4 ↔ 1, 5 ↔ 6, 6 ↔ 8). Mapovací sekce prvního rodiče se zkopíruje do druhého potomka a obdobně mapující sekce druhého rodiče do prvního potomka.

(x x x | 1 6 8 | x x)
(x x x | 4 5 6 | x x)

Dále se postupně kopírují prvky mimo mapovací sekci z i-tého rodiče do i-tého potomka. Pokud už se město v potomkovi vyskytuje, je nahrazeno městem dle příslušného pravidla. V našem případě by prvním městem prvního potomka mělo být město označené číslem 1. Jelikož je však již toto město v prvním potomkovi obsaženo, aplikuje se mapování 1 ↔ 4, a prvním městem se tak stane město číslo 4. U druhého, třetího a sedmého města tento problém nenastává, zkopírují se tedy hodnoty 2, 3, 7. Poslední město by mělo mít číslo 8. To je již v řetězci přítomno, proto se aplikuje pravidlo 8 ↔ 6, avšak město 6 je rovněž obsaženo, použije se tedy další pravidlo 6 ↔ 5 a doplní se číslo 5. Stejným způsobem se postupuje při vytváření druhého potomka [18].

(4 2 3 | 1 6 8 | 7 5)
(3 7 8 | 4 5 6 | 2 1)

4.3.2.2 Cyklický operátor křížení (Cycle Crossover)

Cyklický operátor křížení vytváří jednoho potomka ze dvou rodičů takovým způsobem, že každý prvek potomka je prvek z prvního, nebo druhého rodiče na dané pozici. Když vezmeme v úvahu opět rodiče

(1 2 3 4 5 6 7 8)
(2 4 6 8 7 5 3 1)

postupuje se následovně. Začne se prvním prvkem potomka. V našem případě může nabývat hodnoty 1 nebo 2. Předpokládejme, že bylo vybráno město 1.

(1 - - - - - - -)

Nyní se zaměříme na poslední prvek řetězce. Jelikož tento prvek může nabývat hodnot 8 nebo 1, musí být vybráno město 8, v opačném případě by vznikla nedovolená cesta. Podobně i čtvrtý a druhý prvek nemají na výběr a dostáváme tak

(1 2 - 4 - - - 8).

Pozice doposud vybraných prvků tvořily cyklus. Třetí prvek může být z kteréhokoliv rodiče tedy hodnota 3 nebo 6. Předpokládejme, že byla vybrána hodnota 6. Tato volba způsobila, že pátý, šestý a sedmý prvek musí být vybráni z druhého rodiče, jelikož tvoří další cyklus. Výsledný potomek je

(1 2 6 4 7 5 3 8).

Absolutní pozice v průměru poloviny prvků z obou rodičů zůstala zachována [18].

4.3.2.3 Operátor křížení se zachováním pořadí (Order Crossover)

Využívá toho, že v reprezentaci cestou je důležité pořadí měst, nikoliv jejich pozice. Každý z potomků obsahuje část cesty jednoho z rodičů (každý potomek z jiného) při zachování relativního pořadí měst druhého z rodičů. Uvažujme, že byly vygenerovány body křížení za druhým a pátým prvkem

(1 2 | 3 4 5 | 6 7 8)
(2 4 | 6 8 7 | 5 3 1).

Postupuje se tak, že části řetězců mezi body křížení se zkopírují do potomků, získáváme tak

(- - | 3 4 5 | - - -)
(- - | 6 8 7 | - - -).

Poté se zbytek měst počínaje druhým bodem křížení jednoho rodiče zkopíruje do druhého potomka v pořadí, ve kterém jsou města v druhém rodiči od druhého bodu křížení s tím, že se přeskakují města, která jsou již v potomkovi obsažena. Když se dosáhne konce rodičovského chromozomu, pokračuje se dále od první pozice. Výslední potomci jsou dle [18]

(8 7 | 3 4 5 | 1 2 6)
(4 5 | 6 8 7 | 1 2 3).

4.3.2.4 Operátor křížení s rekombinací hran (Edge Recombination Crossover)

Tento operátor ze dvou rodičů vytváří jednoho potomka takovým způsobem, že každý úsek cesty v novém jedinci pochází z některého rodiče. Při tvorbě potomka používá tzv. hranovou tabulku, která každému městu přiřazuje seznam sousedů vyskytujících se pro dané město v jeho rodičích [14].

Algoritmus

1. Z jednoho z rodičů se vybere počáteční město. Toto město může být vybráno náhodně anebo dle bodu 4. Počáteční město je nyní aktuálním městem.
2. Ze seznamu sousedů se aktuální město odstraní.
3. Pokud má aktuální město ještě nějaké sousedy, pokračuje se bodem 4, v opačném případě bodem 5.
4. Ze sousedů aktuálního města se zvolí město s minimálním počtem sousedů, pokud je takových více, vybere se z nich náhodně. Vybrané město se stává aktuálním městem a pokračuje se krokem 2.
5. Pokud již nezbývá žádné nenavštívené město, algoritmus končí. V opačném případě se některé náhodně vybere, stává se aktuálním městem a výpočet se přesouvá na bod 2.

Uvažujme rodiče

(1 2 3 4 5 6)
(2 4 3 1 5 6).

Hranová tabulka na začátku výpočtu vypadá následovně

Tabulka 4.4: Hranová tabulka (Zdroj: [18])

Město	Sousedé	Město	Sousedé
1	2, 6, 3, 5	4	3, 5, 2
2	1, 3, 4, 6	5	4, 6, 1
3	2, 4, 1	6	1, 5, 2

V našem případě vzniká potomek následujícím způsobem: První město může být město 1 nebo 2, jelikož obě obsahují čtyři sousedy, předpokládejme, že jsme náhodně vybrali město 2. Toto město má sousedy 1, 3, 4, 6. Po odstranění města 2 ze seznamu sousedů má město 3, 4 a 6 dva sousedy, město 1 tři. Vybírat tedy budeme mezi městy 3, 4 a 6. Náhodně jsme zvolili město 3, které má sousedy 4 a 1. Jelikož město 4 má méně sousedů, je vybráno. Město 4 má nyní pouze jednoho souseda, a sice město 5, pokračujeme tedy městem 5. Město 5 obsahuje 2 sousedy, města 6 a 1, oběma z nich zbývá už jen jeden soused. Náhodně zvolíme město 6, další město musí být město 1. Výsledná cesta je

(2 3 4 5 6 1).

Operátor byl později ještě vylepšen tím, že se zohlednila skutečnost, že některá města sousedila vícekrát se stejným městem, jelikož oba rodiče tento přechod obsahovali [14, 18].

4.3.2.5 Výměnná mutace (Exchange mutation)

Tento operátor náhodně vybere dva geny jedince a jednoduše je mezi sebou vymění. Postup ilustruje obr. 4.8, kde byla náhodně vybrána města 2 a 6.

Před mutací		Po mutaci
(1 **2** 3 4 5 **6** 7 8)	→	(1 **6** 3 4 5 **2** 7 8)

Obrázek 4.8: Výměnná mutace (Vlastní zpracování dle [18])

4.3.2.6 Mutace vložením (Insertion mutation)

Mutace vložením náhodně vybere gen, vyjme ho z chromozomu a vloží ho na náhodně vybrané místo. Např. na obrázku 4.9 bylo vybráno město 3.

Před mutací		Po mutaci
(1 2 **3** 4 5 6 7 8)	→	(1 2 4 5 6 **3** 7 8)

Obrázek 4.9: Mutace vložením (Vlastní zpracování dle [18])

4.3.2.7 Mutace přemístěním (Dispacement mutation)

Tento typ mutace náhodně vybere část cesty jedince, vyjme ji a vloží na náhodně zvolené místo. Předpokládejme, že byl náhodně vybrán úsek (3 4 5) a náhodně pozice za městem 6, situaci ilustruje obr. 4.10.

Před mutací		Po mutaci
(1 2 **3 4 5** 6 | 7 8)	→	(1 2 6 | **3 4 5** 7 8)

Obrázek 4.10: Mutace přemístěním (Vlastní zpracování dle [18])

4.3.2.8 Mutace jednoduchou inverzí (Simple inversion mutation)

Vyberou se náhodně dva dělící body. Podřetězec jimi ohraničený se invertuje. Tato mutace je základem 2-opt heuristiky pro řešení TSP. Na obr. 4.11 je zobrazena situace, kdy byly dělící body vybrány za městem 2 a 6.

Před mutací		Po mutaci
(1 2 | 3 4 5 6 | 7 8)	→	(1 2 | 6 5 4 3 | 7 8)

Obrázek 4.11: Mutace jednoduchou inverzí (Vlastní zpracování dle [18])

4.3.2.9 Mutace inverzí (Inversion mutation)

Náhodně vybere část cesty, odebere ji a invertovanou znovu vloží na náhodně zvolenou pozici. Na obr. 4.12 byla náhodně vybrána část cesty (6 7 8) a náhodně pozice za městem 1 [18].

Před mutací		Po mutaci
(1 | 2 3 4 5 **6 7 8**)	→	(1 | **8 7 6** 2 3 4 5)

Obrázek 4.12: Výměnná inverzí (Vlastní zpracování dle [18])

4.3.3 mTSP

Problém více obchodních cestujících (mTSP) je podobný problému obchodního cestujícího (TSP) s rozdílem, že je k dispozici více než jeden obchodník pro navštívení n měst. Pokud pro mTSP zvolíme vhodnou reprezentaci, mohou být při řešení aplikovány operátory vyvinuté pro TSP [4].

Vhodné metody reprezentace mTSP:

- Dvojchromozomová
- Jednochromozomová
- Chromozom se dvěma částmi

4.3.3.1 Dvojchromozomová metoda

První chromozom představuje permutaci měst, zatímco druhý uvádí přiřazení obchodních cestujících městům na dané pozici v prvním chromozomu. Tato reprezentace tedy potřebuje dva chromozomy délky n. Na obrázku 4.13 by byla města 5, 14, 10 a 15 navštívena v tomto pořadí prvním obchodním cestujícím, města 2, 8, 12 a 9 druhým atd. Když m je počet obchodních cestujících, existuje $n!\,(m^n)$ možných řešení, kde mnoho z nich je redundantních.

Města

2	5	14	6	1	11	8	13	4	10	3	12	15	9	7

Obchodníci

2	1	1	3	4	3	2	4	4	1	3	2	1	2	3

Obrázek 4.13: Ukázka dvojchromozové metody (Zdroj: [4])

4.3.3.2 Jednochromozomová metoda

Modeluje mTSP jako TSP pomocí jednoho chromozomu prodlouženého o $m-1$ pozic při počtu m obchodních cestujících. Tyto nové pozice představují fiktivní města reprezentující změnu obchodníků. Na obr. 4.14 jsou zobrazena pomocí záporných čísel. Tedy první obchodní cestující by v tomto případě navštívil města 2, 5, 14, 6 v tomto pořadí, druhý města 1, 11, 8, 13 atd. Výsledná délka chromozomu je pak $n+m-1$. Při této reprezentaci existuje $(n+m-1)!$ možných řešení, kde mnohá z nich jsou opět redundantní.

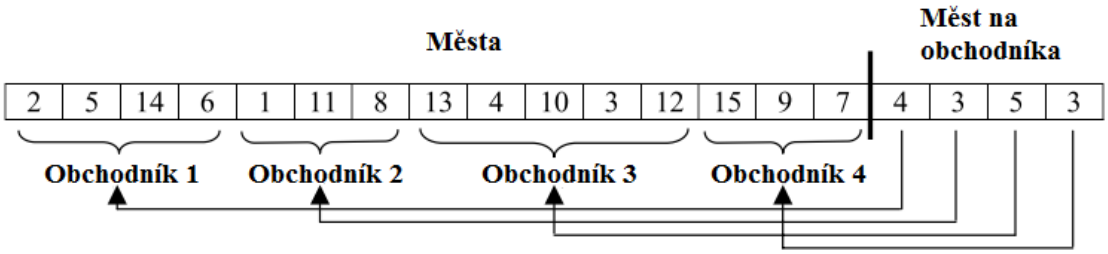
Města

2	5	14	6	-2	1	11	8	13	-3	4	10	1	-1	12	15	9	7

Obrázek 4.14: Ukázka jednochromozové metody (Zdroj: [4])

4.3.3.3 Chromozom se dvěma částmi

Tato technika redukuje redundantní reprezentace stejného řešení (ne všechny). První část chromozomu je permutace n měst. Druhá část chromozomu délky m představuje počet měst přiřazených každému z m obchodníků. Suma hodnot přiřazených každému z obchodníků m se musí rovnat počtu měst $|n|$. Na obrázku 4.15 by první obchodník navštívil města 2, 5, 14, 6 v tomto pořadí, druhý města 1, 11, 8 atd.



Obrázek 4.15: Ukázka reprezentace chromozomem se dvěma částmi (Zdroj: [4])

Porovnání – Při testování výše uvedených reprezentací, kde kritériem byla minimální celková vzdálenost se nenašla metoda, která by předčila ostatní ve všech kombinacích počtu měst a počtu obchodníků. Jednochromozomová metoda si vedla nejlépe pro malé počty obchodníků (do 3 obchodníků ve všech případech), zatímco metoda chromozomu se dvěma částmi poskytovala nejlepší výsledky při vyšším počtu obchodníků [4].

5 Použité nástroje

V této kapitole bude uveden popis nástrojů, kterých bylo použito při zpracování diplomové práce. První podkapitola pojednává o MATLABu, sadě nástrojů Global Optimization a MATLAB Builder JA. Druhá obsahuje informace o Javě a freewarovém vývojovém prostředí NetBeans.

5.1 MATLAB

MATLAB je jednak programovacím jazykem čtvrté generace, jednak interaktivním programovým prostředím pro vývoj algoritmů, vizualizaci a analýzu dat a numerické výpočty. Poskytuje nejen mocné grafické a výpočetní nástroje, ale i rozsáhlé specializované knihovny, které jej činí využitelným v širokém spektru aplikací, včetně zpracování signálů a obrazu, návrhů řídicích a komunikačních systémů, finanční analýze a modelování, výpočetní biologii, ale i v dalších.

MATLAB umožňuje řešit početně náročné úlohy i bez znalosti matematické podstaty problémů a za jeho nejsilnější stránku je považováno mimořádně rychlé výpočetní jádro s optimálními algoritmy, které jsou prověřeny léty provozu na špičkových pracovištích po celém světě. MATLAB čítá více než milión uživatelů z akademických i průmyslových řad a byl implementován na všech významných platformách (Windows, Linux, Solaris, Mac). Jeho název vznikl zkrácením slov MATrix LABoratory [20, 21, 23].

5.1.1 Global Optimization Toolbox

Tato sada nástrojů poskytuje metody, které hledají globálně optimální řešení problémů, jež obsahují lokální extrémy. Mezi tyto metody patří globální vyhledávání, start z několika výchozích bodů, vyhledávání vzorů, genetické algoritmy a simulované žíhání.

Konkrétněji se budeme zabývat genetickými algoritmy, jež lze použít na řadu optimalizačních problémů, které nejsou vhodné pro standardní optimalizační algoritmy, včetně problémů, kde je účelová funkce nespojitá, nediferencovatelná, stochastická anebo vysoce nelienární. Genetické algoritmy umožňují nastavit vlastní funkce pro vytvoření populace, změnu měřítka hodnot fitness funkce, výběru rodičů, křížení a mutaci.

Optimalizaci genetickými algoritmy lze provést

- voláním funkce ga z příkazové řádky
- použitím optimalizačního nástroje

5.1.1.1 Funkce ga

Volání funkce ga má v základu následující syntaxi

```
[x fval] = ga(@fitnessfun, nvars, options)
```

kde *@fitnessfun* je handle fitness funkce; *nvars* je počet nezávislých proměnných fitness funkce; *options* je struktura obsahující nastavení pro genetické algoritmy, pokud není tento argument předán, je použito výchozí nastavení.

Návratové hodnoty funkce jsou *x* a *fval*, kde *x* je bod, ve kterém je nalezeno řešení, *fval* je hodnota fitness funkce v nalezeném řešení. Funkci lze volat i s dalšími výstupními parametry (*exitflag* – celočíselná hodnota odpovídající důvodu ukončení algoritmu, *output* – struktura obsahující informace o výkonu algoritmu v každé generaci, *population* – poslední populace, *scores* – konečné výsledky).

Tento způsob je vhodný, když genetické algoritmy chceme volat ze souboru (ať už ze souboru, ve kterém je definována funkce nebo ze souboru obsahujícího skript).

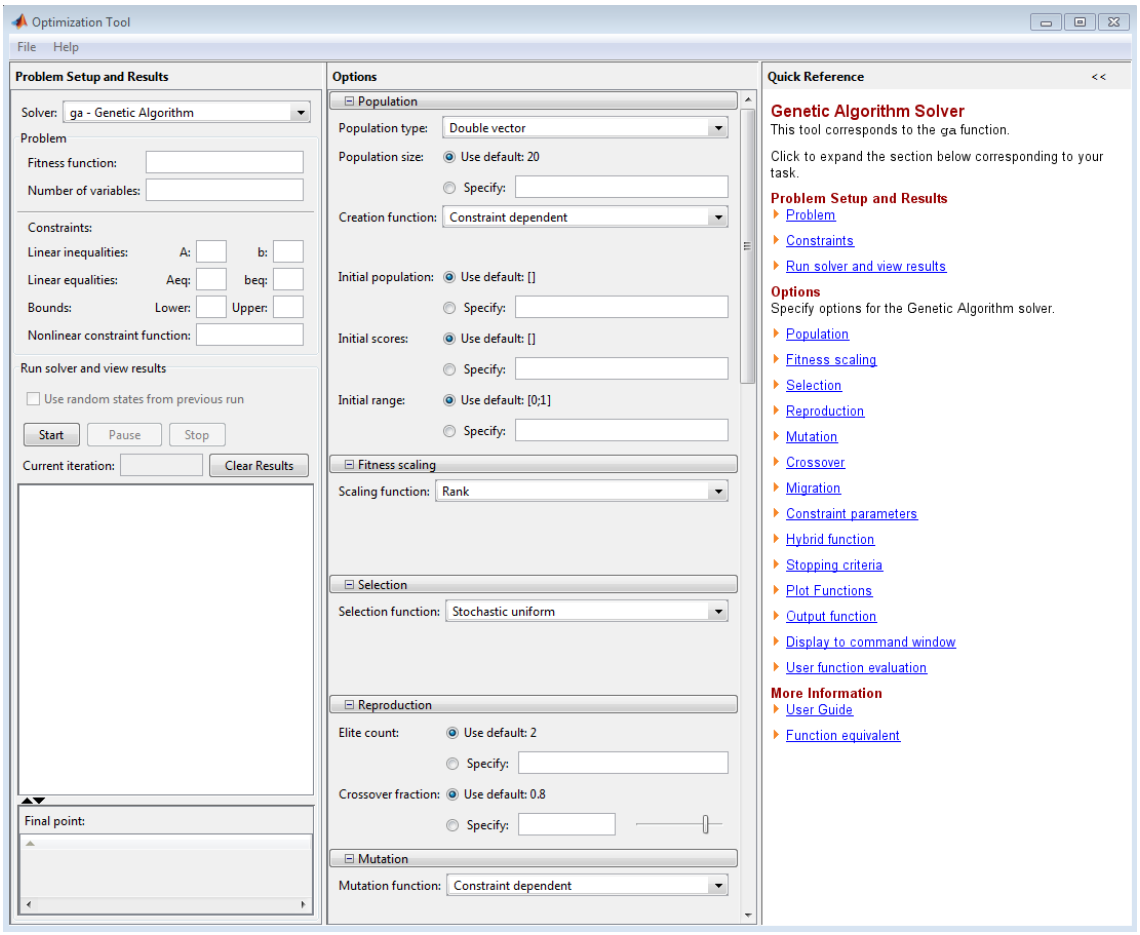
5.1.1.2 Optimalizační nástroj

Optimalizační nástroj lze otevřít buď voláním

```
optimtool('ga')
```

z příkazové řádky, nebo přes **Start** – **Toolboxes** – **Optimization** – **Optimization Tool** a výběrem **ga – Genetic Algorithm** jako řešitele (**Solver**).

Pro práci s tímto nástrojem je třeba zadat informace o fitness funkci (ve tvaru *@fitnessfun*, kde *fitnessfun.m* je soubor obsahující její výpočet), počtu proměnných – délce vstupního vektoru předávaného do fitness funkce, volitelně je možné zadat omezující podmínky a změnit nastavení v panelu **Options**. Poté stačí kliknout na tlačítko **Start**, výsledky optimalizace se objeví v panelu **Run solver and view results** [10].



Obrázek 5.1: Optimalizační nástroj

5.1.1.3 Nastavení genetických algoritmů

V případě volání funkce ga lze parametry zadat pomocí struktury vytvořené funkcí *gaoptimset*

```
options = gaoptimset('Param1', value1, 'Param2', value2, ...);
```

Při použití optimalizačního nástroje se dané položky nastaví výběrem ze seznamu nebo vyplněním příslušného textového pole.

Některá nastavení genetických algoritmů

- Plot
- Population
- Fitness Scaling
- Selection

- Reproduction
- Mutation
- Crossover
- Stopping Criteria
- Output Function
- Vectorize and Parallel Option

Plot – V průběhu výpočtu je možné vykreslovat různá data (např. *best fitness* - nejlepší hodnoty fitness funkce v každé generaci, *score diversity* - histogram hodnot fitness funkce dané generace, *range* - minimální, maximální a střední hodnoty fitness funkce každé generace, *custom* – vlastní funkce a řada dalších), parametr *plot interval* udává počet generací mezi voláním funkce pro vykreslení.

Population – Zde se nastavují parametry populace, např. typ populace (*double* – desetinné číslo, *bit string* – řetězec bitů, *custom* - vlastní), velikost populace a funkce pro vytvoření populace (*uniform* – pomocí rovnoměrného rozdělení, *feasible population* - náhodná splňující podmínky, *custom* - vlastní). Dále je také možné nastavit počáteční populaci, počáteční skóre a interval ohraničující hodnoty počáteční populace.

Fitness Scaling – Zde se vybírá funkce, která přede hodnoty vrácené fitness funkcí na hodnoty v rozsahu, který je vhodný pro výběrovou funkci. K dispozici jsou *rank* – založena na pořadí, *proportional* – proporcionálně odpovídají hodnotám fitness, *top* – top nejlepších je ohodnoceno kladně stejnou hodnotou, zbytek nulou, *shift linear* – nejlepší jedinec bude ohodnocen součinem zadané konstanty a průměrného skóre, *custom* – vlastní.

Selection – Toto nastavení udává, jakým způsobem genetické algoritmy vyberou rodiče další generace (*stochastic uniform* – lze si představit jako přímku, kde je jeden rodič za druhým a délka každého z nich odpovídá proporcionálně jeho změněné fitness, náhodně se zvolí výchozí bod menší než velikost kroku a pak se daným krokem prochází přímka a vybírají se rodiče, kteří obsahují daný bod, *remainder* – v prvním kroku jsou rodiče určeni deterministky z celé části své upravené fitness hodnoty, v druhém stochasticky ruletově na základě desetinné části, *uniform* – pomocí očekávání a počtu rodičů,

roulette – Simuluje ruletu, výseče odpovídají proporcionálně očekávání, *tournament* – náhodně vybere daný počet jedinců a vybere nejlepšího, *custom* – vlastní).

Reproduction – Určuje, jak jsou vytvářeni potomci, *elite count* – počet nejlepších jedinců, kteří automaticky postupují do další generace, *crossover fraction* – podíl jedinců, kteří budou vytvořeni křížením mimo elitní jedince.

Mutation – Nastavuje, jakým způsobem bude probíhat mutace (*Gaussian* – na základě Gaussova rozdělení mění geny jedince, *uniform* – v prvním kroku pomocí uniformního rozdělení vybere geny, v druhém provede mutaci, *adaptive feasible* – adaptivní změny s ohledem na omezení, *custom* – vlastní).

Crossover – Zde volíme způsob křížení (*scattered* – na základě náhodně vygenerovaného binárního řetězce se daný gen potomka vybere z jednoho či druhého rodiče, *single point* – jednobodové, *two point* – dvoubodové, *intermediate* – na základě váženého průměru rodičů, *heuristic* – potomek leží na přímce procházející oběma rodiči, malou vzdálenost od lepšího ve směru pryč od horšího, *arithmetic* – vážený aritmetický průměr rodičů s ohledem na omezení a hranice, *custom* – vlastní).

Stopping Criteria – Určuje, kdy algoritmus končí (počet generací, časový limit, limit fitness funkce, nemožnost získání lepšího řešení po určitý počet generací nebo zvolenou dobu).

Output Function – Zde je možné nastavit funkci, kterou genetický algoritmus bude volat v každé generaci.

Vectorize and Parallel Option – Zde se nastavuje, zda jsou fitness funkce a omezující funkce vyhodnocovány postupně, paralelně anebo vektorově [9].

5.1.2 MATLAB Builder JA

MATLAB Builder JA umožňuje programy napsané v MATLABu „zabalit“ do javovské třídy. Tyto třídy mohou být importovány do programů napsaných v jazyce Java a bez poplatku nasazeny na běžné počítače nebo webové servery, které nemají MATLAB nainstalovaný. Ke svému chodu používají tzv. MATLAB Compiler Runtine, který je zdarma ke stažení. MATLAB Compiler Runtine obsahuje kompletní sadu sdílených

knihoven potřebných pro spuštěný komponent založených na MATLABu a poskytuje tak plnou podporu všech rysů jazyku MATLAB a většiny toolboxů.

V součinnosti s MATLAB Compiler (kompilátorem) vytvoří Builder komponenty, které zpřístupňují uživatelům Javy výpočty, vizualizace a grafické uživatelské rozhraní založené na MATLABu.

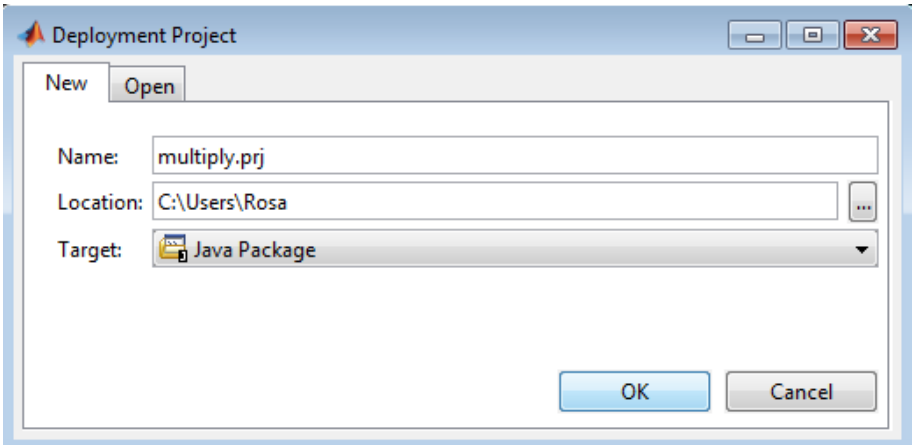
Builder šifruje funkce napsané v MATLABu a vytváří nad nimi obálku tzv. Java wrapper tak, že se pak chovají jako všechny ostatní třídy v Javě. Java třídy takto vytvořené jsou přenositelné a spustitelné na všech platformách podporovaných MATLABem [31].

5.1.2.1 Vytvoření Java komponenty

Předpokládejme, že máme funkci multiply uloženou v souboru multiply.h, jejíž obsah je

```
function y = multiply(x)
y = 10 * x;
```

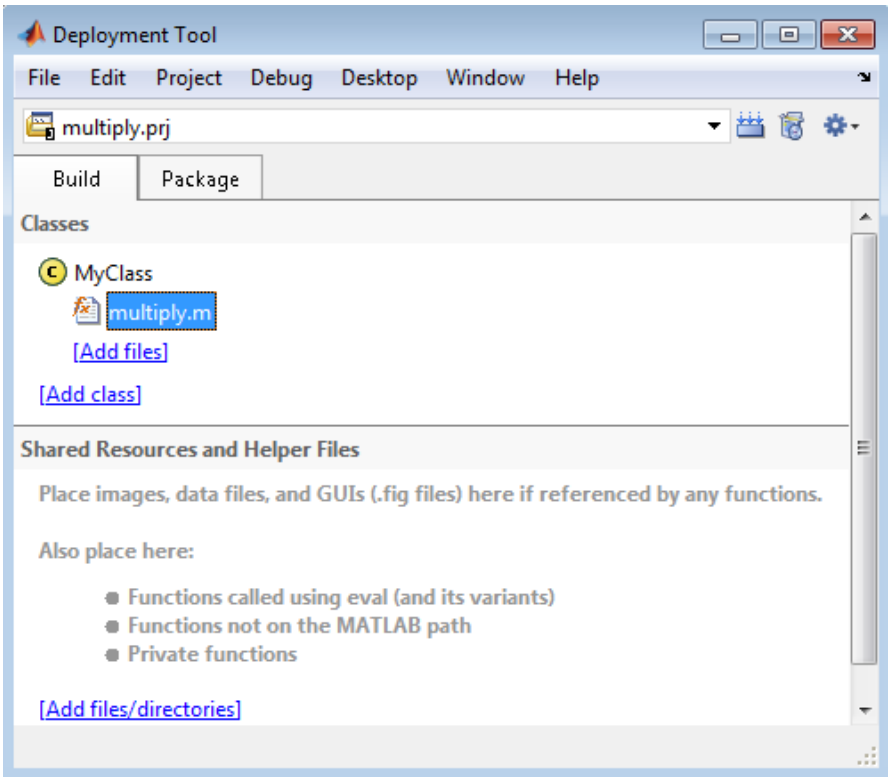
Příkazem `deploytool` v příkazové řádce MATLABu spustíme uživatelské rozhraní příslušného nástroje (viz obr. 5.2). Vyplníme jméno projektu, zvolíme jeho umístění a jako cíl (**Target**) vybereme **Java Package**, klikneme na **OK**.



Obrázek 5.2: Vytvoření nového projektu pro export programu v MATLABU do Java komponenty

V záložce Build (obrázek 5.3) klikneme na **Add class** a napíšeme jméno třídy, kterou chceme vytvořit např. MyClass. Kliknutím na **Add files** přidáme funkce napsané v MATLABu, které chceme využívat (v našem případě soubor multiply.m). Tyto

funkce se stanou metodami vytvářené třídy. Podpůrné soubory lze přidat kliknutím na **Add files/directories**. Komponentu vytvoříme kliknutím na tlačítko **Build** ( ) [22].



Obrázek 5.3: Vytvoření Java komponenty

5.2 Java

Java je objektově orientovaný jazyk, který vyvinula a v roce 1995 vydala společnost Sun Microsystems. Je jedním z nejpoužívanějších a také nejpopulárnějších programovacích jazyků na světě. Jeho syntaxe vychází z jazyků C a C++, oproti C++ má jednodušší objektový model. Aplikace napsané v Javě jsou zpravidla kompilovány do tzv. byte kódu (class soubor), který je nezávislý na počítačové architektuře. Byte kód je interpretován pomocí Java Virtual Machine (JVM), která již závislá je. Java nachází využití v desktopových aplikacích (Java Standard Edition), podnikových a webových aplikacích běžících na serverech (Java Enterprice Edition), v aplikacích pro přenosná a vestavěná zařízení (Java Micro Edition), aplikacích pro čipové karty (Java Card). Java umožňuje psaní vysoce spolehlivého a bezpečného kódu, podporuje vícevláknové aplikace a má automatickou správu paměti [15, 16].

5.2.1 NetBeans

NetBeans je open-source projekt zaměřený na poskytování produktů pro vývoj softwaru. Má rozsáhlou uživatelskou základnu a komunitu vývojářů a partnerů po celém světě. Pod open-source licencí byl uvolněn v roce 2000 společností Sun Microsystems, která zůstala hlavním sponzorem až do roku 2010, kdy se stala dceřinou společností společnosti Oracle. V rámci projektu existují dva hlavní produkty. Jedná se o vývojové prostředí NetBeans (NetBeans IDE) a vývojovou platformu NetBeans (the NetBeans Platform), přičemž oba produkty je možné bezplatně využívat pro komerční i nekomerční účely.

Vývojové prostředí je napsáno v Javě, takže jej lze spustit na operačních systémech Windows, Linux, Mac OS, Solaris a dalších platformách podporujících kompatibilní Java Virtual Machine. Poskytuje podporu vývoje všech typů Java aplikací ale také dalších programovacích jazyků, jako např. PHP, JavaFX, C/C++, JavaScript atd. Existuje celá řada přídavných modulů, které toto vývojové prostředí rozšiřují [1, 25, 26].

5.2.2 Použití komponenty vytvořené pomocí MATLAB Builder JA

Použití komponenty vytvořené v kapitole 5.1.2.1 ilustruje ukázka kódu uvedena dále.

Pro práci s komponentou je třeba

- mít nainstalovaný MATLAB nebo MATLAB Compiler Runtime a řádně nastavené systémové proměnné
- importovat nejen třídy naší komponenty, ale i knihovny MATLABu
- vytvořit instanci třídy komponenty pomocí klíčového slova *new*
- uvolnit nepotřebné zdroje (není nutné, ale je to dobrým zvykem).

Metoda naší komponenty očekává vstupní parametr ve formátu interního pole MATLABu. Je možné využít automatické konverze a metodě předat datový typ Javy nebo manuálně tento parametr konvertovat do standardního typu MATLABu pomocí tříd *MWArray*, obsažených v balíku *com.mathworks.toolbox.javabuilder*. Více informací lze nalézt v nápovědě [22], ze které tato kapitola vychází.

```
// import potrebnych baliku
import com.mathworks.toolbox.javabuilder.*;
import multiply.*;

/**
 * Trida example spocita nasobek cisla 10 a celeho cisla
 * zadaneho z prikazove radky.
 */
class example {

    public static void main(String[] args) {
        MWNumericArray x = null;    // vstup
        Object[] result = null;     // vystup
        MyClass myClass = null;     // ukazkova trida

        try {
            // kontrola
            if (args.length != 1) {
                System.out.println("Chyba: Zadejte číslo,"
                        + " které chcete vynásobit");
                return;
            }

            // konverze vstupni hodnoty
            int i = Integer.valueOf(args[0]);
            x = new MWNumericArray(i, MWClassID.INT32);

            // vytvoreni objektu
            myClass = new MyClass();

            // vynasobeni a tisk vysledku
            result = myClass.multiply(1, x);
            System.out.println("10 * " + i + " = " + result[0]);

        } catch (Exception e) {
            System.out.println("Exception: " + e.toString());
        } finally {
            // uvolneni nativnich zdroju
            MWArray.disposeArray(x);
            MWArray.disposeArray(result);
            if (myClass != null) {
                myClass.dispose();
            }
        }
    }
}
```

6 Analýza problému a současné situace

V této kapitole bude provedena analýza problému plánování tras pro servisní techniky společnosti VÝTAHY, s.r.o., která bude sloužit jako základ pro návrh a implementaci podpůrné aplikace.

6.1 Analýza problému

Jak bylo uvedeno výše, společnost mimo jiné provádí pravidelný servis výtahů v lokalitách zahrnujících celou Českou republiku. Každý den vyjíždí na cestu několik týmů servisních techniků, kteří mají za úkol navštívit zadaná místa, provést potřebné práce a vrátit se zpět. Společnost Výtahy, s.r.o. nepoužívá v současné době žádný specializovaný podpůrný software a plánování tras provádí podle zkušeností. Následující podkapitola pojednává o tom, co je třeba při plánování vzít v úvahu a další pak převádí tento problém na instanci problému obchodního cestujícího.

6.1.1 Fakta o plánování tras

V daný den je třeba navštívit x míst, jejichž seznam má osoba zodpovědná za plánování k dispozici. Musí se rozhodnout, kolik techniků bude potřebovat a kdo kam pojede. Její snahou je minimalizovat cestovní náklady, které byly stanoveny na 13 Kč/km.

Na některé výtahy potřebují technici speciální vybavení, kterých má společnost k dispozici omezený počet, třeba i jeden kus. To plánovači práci ulehčí, protože bude vědět, že tým, který bude mít toto vybavení, bude muset jet na dané místo. Je jistě výhodné, aby tento tým navštívil i místa vyskytující se v nejbližším okolí.

Zároveň je však žádoucí, aby týmy strávily v terénu přibližně stejnou dobu, aby v extrémních případech nedošlo k situaci, že jeden tým pojede pouze na jedno místo, zatímco druhý bude pracovat do půlnoci. Je třeba vzít v úvahu i fakt, že servis jednoho výtahu nemusí nutně trvat stejně dlouho jako servis jiného.

Není nutné, aby servisní technici dojížděli každý den do sídla společnosti. Firemní vozy mohou parkovat v místě bydliště.

Někdy může být rozhodování velmi snadné, např. když je nutné navštívit 6 míst v Jihlavě, 6 míst v Brně, žádná speciální technika není třeba a jeden tým by práci nestihl. V jiných případech nemusí být plánování vůbec jednoduché.

6.1.2 Paralela s problémem obchodního cestujícího

Dle fakt uvedených v předchozí kapitole lze usoudit, že se jedná o aplikaci problému více obchodních cestujících. Kritériem při plánování není pouze minimální celková ujetá vzdálenost, ale zohledňuje se také čas (doba jízdy a doba servisu) a potřebné vybavení. Vzhledem k tomu, že cesta z jednoho místa na druhé nemusí být nutně stejně dlouhá i ve směru opačném, budeme na problém nahlížet jako na asymetrický. Jelikož je technikům povoleno parkovat firemní vozy v místě bydliště a nemusí nutně každý den dojíždět do sídla společnosti, jedná se navíc o vícezákladnové rozšíření.

7 Vlastní návrhy řešení

V této kapitole jsou uvedeny požadavky na podpůrnou aplikaci tak, aby byla využitelná v praxi, umožňovala definovat potřebné parametry optimalizace a poskytovala relevantní výstupy. Následuje její návrh a implementace v MATLABu a v Javě. V případové studii je srovnání plánů poskytnutých společností VÝTAHY, s.r.o., které byly vytvořeny bez specializovaného softwaru, a plánů vytvořených navrženou aplikací při stejných vstupech.

7.1 Požadavky na aplikaci

V podpůrné aplikaci by mělo být možné zadat

- místa, která je nutné v daný den navštívit
- odhadovanou dobu servisu
- předpokládaný počet týmů
- základny jednotlivých týmů
- omezující podmínky
- váhu optimalizačních kritérií (minimální celková vzdálenost či podobná časová náročnost)

Aplikace by měla zobrazovat

- celkový přehled (ujeté kilometry, časovou náročnost)
- přehledy a trasy pro jednotlivé týmy

7.2 Návrh řešení

Fakta o plánování tras byla uvedena v kapitole 6.1.1. Ze závěru kapitoly 6.1.2 pak vyplývá, že je tento problém možné řešit jako asymetrický problém více obchodních cestujících s více základnami, kde náklady na cestování budou tvořeny váženým průměrem celkové vzdálenosti a časové vyrovnanosti týmů. Je nutné počítat s omezujícími podmínkami kvůli vybavení a také možností různých servisních dob různých výtahů.

Vzhledem k výpočetní složitosti TSP (kapitola 4.1.2) se jako elegantní možnost řešení instance tohoto problému nabízejí genetické algoritmy (kapitola 4.2). Z poznatků v kapitole 4.3 a požadavků kladených na aplikaci (kapitola 7.1) navrhuji to řešení.

Implementace bude provedena ve vývojovém prostředí MATLAB (kapitola 5.1) za pomoci Global Optimization Toolboxu (kapitola 5.1.1) a řešitele genetických algoritmů. Problém bude zakódován reprezentací cesty (kapitola 4.3.1.4) modifikací jednochromozomového způsobu (kapitola 4.3.3.2). Jako operátor křížení bude využita zcela postačující mutace jednoduchou inverzí (kapitola 4.3.2.8), operátorem mutace bude výměnná mutace (kapitola 4.3.2.5). S omezujícími podmínkami (kapitola 4.2.12) se vypořádáme penalizací ve fitness funkci. Implementace tohoto návrhu je blíže popsána v kapitole 7.3.

Řešení v MATLABu se použije jako základ pro aplikaci, jež pro svůj běh nebude potřebovat MATLAB, nýbrž pouze MATLAB Compiler Runtime, který je bezplatně k dispozici. Toto řešení bude implementováno pomocí programovacího jazyka Java (kapitola 5.2). Nejprve se pomocí MATLAB Builder JA (kapitola 5.1.2) původní řešení exportuje do Java komponenty. Poté bude vytvořeno grafické uživatelské rozhraní, které bude využívat funkcí této komponenty. Grafické uživatelské rozhraní bude navrženo s ohledem na intuitivní ovládání aplikace. Popis implementace v jazyce Java je uveden v kapitole 7.4.

7.3 Implementace v MATLABu

Tato kapitola detailně popisuje některé funkce, které jsou stěžejní pro genetické algoritmy. Nejprve je vysvětlena zvolená reprezentace chromozomu, následuje ukázka generování populace, křížení, mutace a hlavní část, která zpracuje uživatelské vstupy, nastaví parametry optimalizace a provede výpočet.

7.3.1 Reprezentace chromozomu

Jedná se o modifikaci jednochromozomové metody reprezentace chromozomu. Uvažujme, že základna prvního týmu je ve městě 1, druhý a třetí tým mají stejnou základnu umístěnou ve městě 2 (obrázek 7.1).

Základny

1	2	2

Obrázek 7.1: Základny

Potom chromozom (obrázek 7.2)

Města

16	5	7	8	-18	11	4	17	13	9	-19	6	12	10	15	14

Obrázek 7.2: Navržená reprezentace chromozomu

reprezentuje následující cesty. Cesta prvního týmu je 1 – 16 – 5 – 7 – 8 – 1, druhého 2 - 11 – 4 – 17 – 13 – 9 – 2, třetího 2 – 6 – 12 – 10 – 15 – 14. Záporná čísla znamenají změnu týmu, čísla měst v chromozomu začínají od čísla i, kde $i = t + 1$ a t je počet týmů.

7.3.2 Generování populace

Generování populace probíhá s ohledem na zvolenou reprezentaci. Nejprve je vygenerovaná náhodná permutace o délce $n + t - 1$, kde n je počet měst a t je počet týmů. Čísla, která jsou větší, než je počet měst n, jsou vynásobena -1, k číslům, jež jsou menší nebo rovna počtu týmů t je přičtena hodnota n. Tímto vznikne reprezentace popsaná v kapitole 7.3.1.

```
function population = MTSP_create(genomeLength, fitnessFcn, options, teams)
    totalPopulationSize = sum(options.PopulationSize);
    % novy pocet mest z duvodu reprezentace MTSP
    newGenomeLength = genomeLength + teams - 1;
    population = cell(totalPopulationSize, 1);

    for i = 1:totalPopulationSize
        % nahodna permutace 1:n
        genome = randperm(newGenomeLength);
        % zmeny tymu
        genome(genome > genomeLength) = genome(genome > genomeLength).*-1;
        % vytvoreni mista pro zakladny
        genome(genome <= teams) = genome(genome <= teams) + genomeLength;
        population{i} = genome;
    end
end
```

7.3.3 Křížení

Křížení probíhá pomocí jednoduché inverzní mutace jednoho rodiče popsané v kapitole 4.3.2.8. Tento způsob je rychlý a pro naše účely dostačující.

```
function xoverKids = MTSP_crossover(parents, options, nvars, FitnessFcn,
unused, thisPopulation)
idx = 1;
countKids = length(parents) / 2;
xoverKids = cell(countKids, 1);

for i = 1:countKids
   parent = thisPopulation{parents(idx)};
   idx = idx + 2;
   i1 = ceil((length(parent) - 1) * rand);
   i2 = i1 + ceil((length(parent) - i1 - 1) * rand);
   kid = parent;
   % krizeni pomoci jednoduche inverzni mutace
   kid(i1:i2) = fliplr(kid(i1:i2));
   xoverKids{i} = kid;
end
end
```

7.3.4 Mutace

Jedná se o výměnnou mutaci popsanou v kapitole 4.3.2.5.

```
function mutationChildren = MTSP_mutation(parents, options, nvars, ...
                            FitnessFcn, state, thisScore, thisPopulation)
mutationChildren = cell(length(parents), 1);
for i = 1:length(parents)
    parent = thisPopulation{parents(i)};
    ind = ceil(length(parent) * rand(1,2));
    child = parent;
    child(ind(1)) = parent(ind(2));
    child(ind(2)) = parent(ind(1));
    mutationChildren{i} = child;
end
end
```

7.3.5 Fitness funkce

Fitness funkce pracuje níže popsaným způsobem:

Pokud je počet týmů dva a více, potom pro každého jedince ověřuje, zda hranice není v jeho krajních genech a zda dvě hranice neleží hned vedle sebe. Kdyby tomu tak bylo, nebyl by některý z týmů vůbec využit, což by bylo hrubé porušení zadání. V takovém případě by byl jedinec ohodnocen maximální hodnotou představující penalizaci za nepřípustné řešení a pokračovalo by se dalším jedincem.

Pokud k porušení zadání nedošlo, pokračuje se tak, že se pro každý tým vytáhne z jedince korespondující cesta a v případě, že jsou zadány omezující podmínky týkající

se přiřazení či zákazu návštěvy servisního místa danému týmu, provede se test, zda jsou splněny. Pokud ne, opět nemá cenu pokračovat, je vrácena maximální hodnota.

V případě, že nedošlo k penalizaci, je přistoupeno k samotnému ohodnocení jedince. Postupně se pro každý tým prochází jeho trasa a sčítají se délky tras a časy jízd a dob servisních prací na daných místech. Nakonec se dle zvoleného poměru provede vážený aritmetický průměr součtu délky tras všech týmů a nevyváženosti časové náročnosti jednotlivých tras. Tento výsledek je ohodnocením daného jedince.

```
function scores = MTSP_fitness(input, distances, times, teamsN,
strongConditions, ratio, serviceTime)
% max. pokuta za nedodrzeni tezkych podminek, poctu tymu
MAX_PENALTY = 100000000000;

solutionsCount = size(input,1);
scores = zeros(solutionsCount, 1);

for i = 1:solutionsCount
    finish = false;
    solution_i = input{i};

    % nastaveni indexu, kde je zmena tymu
    teamBounds = find(solution_i < 0);

    % PRVNI TEST - pocet tymu, jinak receno 2 hranice nemuzou byt vedle
    % sebe
    if ~isempty(teamBounds)  % pokud je vice tymu nez jeden
        % kontrola hranic, delic nemuze by na zacatku nebo na konci
        if min(teamBounds) == 1 || max(teamBounds) == length(solution_i)
            finish = true;            % dal neni treba testovat
            scores(i) = MAX_PENALTY;  % nastav max penalty
        end

        % vice nez dva tymy, 1 delic, delice nesmi byt vedle sebe
        if ~finish && length(teamBounds) > 1
            for j=1:length(teamBounds)-1
                if abs(teamBounds(j+1) - teamBounds(j)) == 1
                    finish = true;            % dal neni treba testovat
                    scores(i) = MAX_PENALTY;  % nastav max penalty
                    break;
                end
            end
        end
    end

    % pokud nesouhlasi pocet tymu, nema cenu pokracovat
    if finish
        continue;
    end

    % vytazeni cest pro jednotlive tymy
    teamKpath = cell(teamsN, 1);
    teamBoundsExt = [0, teamBounds, length(solution_i) + 1];
    for j=1:teamsN
        teamKpath{j} = solution_i(teamBoundsExt(j)+1:teamBoundsExt(j+1)-1);
    end

    % DRUHY TEST - tezke podminky, prirazeni tymu
```

```
    if ~finish && ~isempty(strongConditions) % pokud jsou
        for k=1:size(strongConditions, 1)
            cond = strongConditions{k};
            t = cond(1);
            c = cond(2);
            z = cond(3);
            if z == 1 % musi byt
                if isempty(find(teamKpath{t} == c, 1))
                    finish = true;            % dal neni treba testovat
                    scores(i) = MAX_PENALTY;  % nastav max penalty
                    break;
                end
            else     % nesmi byt
                if ~isempty(find(teamKpath{t} == c, 1))
                    finish = true;            % dal neni treba testovat
                    scores(i) = MAX_PENALTY;  % nastav max penalty
                    break;
                end
            end
        end
    end

    % pokud byla porusena tezka podminka, nema cenu pokracovat
    if finish
        continue;
    end

    % VYPOCET FITNESS
    % spocitame vzdalenosti a casy pro jednotlive tymy
    teamKscoreDist = zeros(teamsN, 1);
    teamKscoreTime = zeros(teamsN, 1);
    for l=1:teamsN
        % na zacatek a na konec pridame vychozi mesto
        teamKpathExt = [1, teamKpath{l}, 1];
        % projdeme cestu, pripocitavame vzdalenosti a casy
        for j = 2:length(teamKpathExt)
            teamKscoreDist(l) = teamKscoreDist(l) + distances(teamKpathExt(j-
1), teamKpathExt(j));
            ttt = serviceTime(teamKpathExt(j));
            teamKscoreTime(l) = teamKscoreTime(l) + times(teamKpathExt(j-1),
teamKpathExt(j)) + ttt;
        end
    end

    % prumerny cas vsech tymu
    averageTime = mean(teamKscoreTime);
    % casova nevyvazenost
    scoreTime = sum(abs(teamKscoreTime - averageTime)) /teamsN;
    % celkova vzdalenost
    scoreDist = sum(teamKscoreDist);
    % vahovani
    score = scoreDist * (1-ratio) + scoreTime * ratio;
    % vysledek
    scores(i) = score;
end
end
```

7.3.6 Hlavní funkce

Funkce MTSP_solver zpracuje vstupní parametry, načte potřebné matice, nastaví parametry optimalizace a handly na vlastní funkce pro křížení, mutaci, fitness funkci a

vykreslování dat. Potom spustí optimalizaci a z nalezeného chromozomu s nejlepším ohodnocením připraví výslednou strukturu, obsahující celkové informace a údaje o trasách jednotlivých týmů.

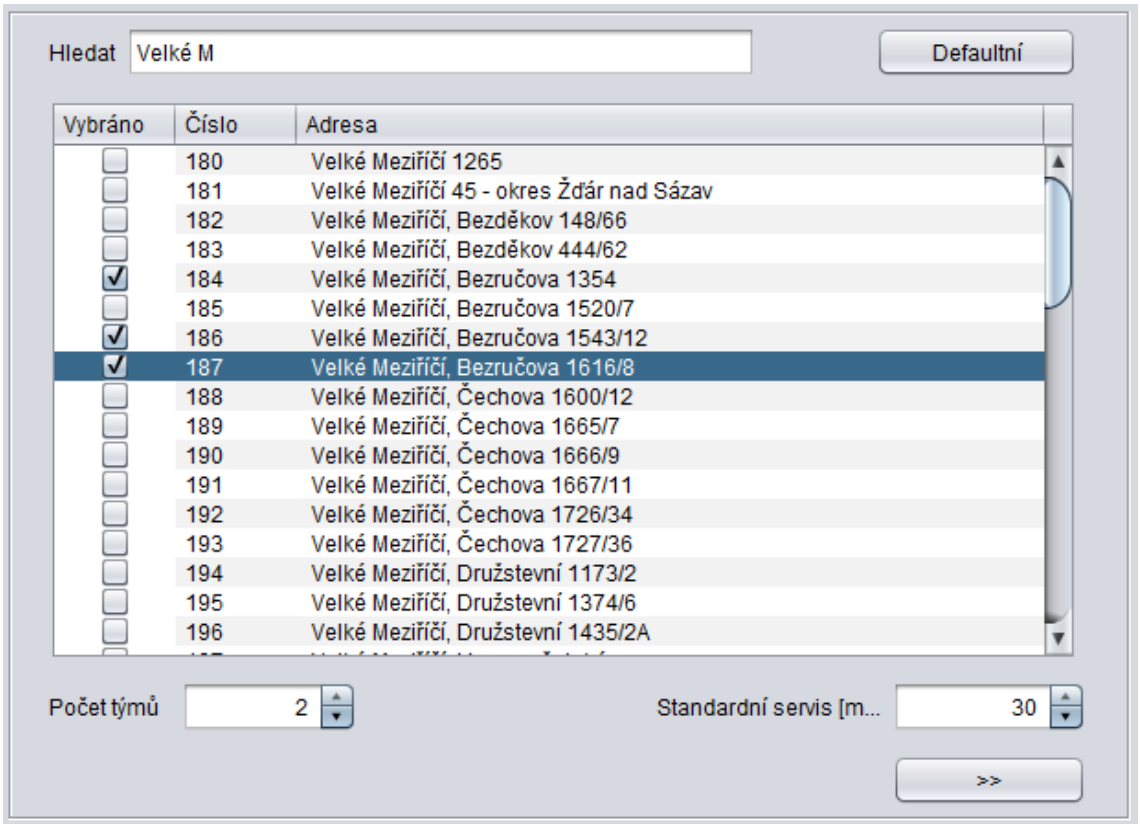
```
function [x, fval, reason, output, result] = MTSP_solver(depots, ...
    citiesSelection, numberOfTeams, strongConditions, ...
    distance2timeBalanceRatio, standardServiceTime, specialServiceTimes)

    [coordOfCities, citiesNames] = MTSP_create_cities();
    distanceMatrix = dlmread('dist_matrix');
    timeMatrix = dlmread('time_matrix');
    depotsNCities = [depots,citiesSelection];
    serviceTimes = MTSP_prepareServiceTimes(depots, citiesSelection,...
        standardServiceTime, specialServiceTimes);
    strongConditions = MTSP_transferStrongConditions(depotsNCities, ...
        strongConditions);

    % vytvoreni vyrezu ze vzdalenosti - selekce a projekce
    distancesSelection = distanceMatrix(depotsNCities,:);
    distancesSelection = distancesSelection(:,depotsNCities);

    % vytvoreni vyrezu z casu - selekce a projekce
    timesSelection = timeMatrix(depotsNCities,:);
    timesSelection = timesSelection(:,depotsNCities);

    % pocet mest
    numberOfCities = length(citiesSelection);

    % nastaveni ukazatelu na funkce
    PlotFcn = @(options, state, flag) ...
        MTSP_plot(state, coordOfCities, depotsNCities, numberOfTeams);

    CrossoverFcn = @(parents, options, nvars, FitnessFcn, unused,
thisPopulation)...
        MTSP_crossover(parents, options, nvars, FitnessFcn, unused,
thisPopulation);

    CreateFcn = @(GenomeLength, FitnessFcn, options)...
        MTSP_create(GenomeLength, FitnessFcn, options, numberOfTeams);

    FitnessFcn = @(x) MTSP_fitness(x, distancesSelection, timesSelection, ...
        numberOfTeams, strongConditions, distance2timeBalanceRatio, ...
        serviceTimes);

    % http://www.mathworks.com/help/toolbox/gads/gaoptimset.html
    options = gaoptimset('PopulationType', 'custom', ...
                         'PopInitRange', [1;numberOfCities], ...
                         'CreationFcn', CreateFcn, ...
                         'CrossoverFcn', CrossoverFcn, ...
                         'MutationFcn', @MTSP_mutation, ...
                         'Generations', 300, ...
                         'PopulationSize', 150, ...
                         'StallGenLimit', 250, ...
                         'PlotFcn', {PlotFcn;@gaplotbestf}, ...
                         'Vectorized', 'on');

    [x, fval, reason, output] = ga(FitnessFcn, numberOfCities, options);
    %x
    result = MTSP_result(x, depotsNCities, distancesSelection, ...
        timesSelection, numberOfTeams, distance2timeBalanceRatio, ...
        serviceTimesSelection);
end
```

7.4 Implementace v Javě

V Javě bylo vytvořeno grafické uživatelské rozhraní, které na základě uživatelských vstupů připravuje potřebná data pro komponentu vytvořenou z implementace v MATLABu provádějící samotný výpočet. Po provedení výpočtu komponentou navrácená data aplikace transformuje do prvků uživatelského rozhraní tak, aby uživateli byly poskytnuty přehledné informace o nalezených trasách. V následující podkapitole je demonstrována výsledná aplikace a její funkcionalita.

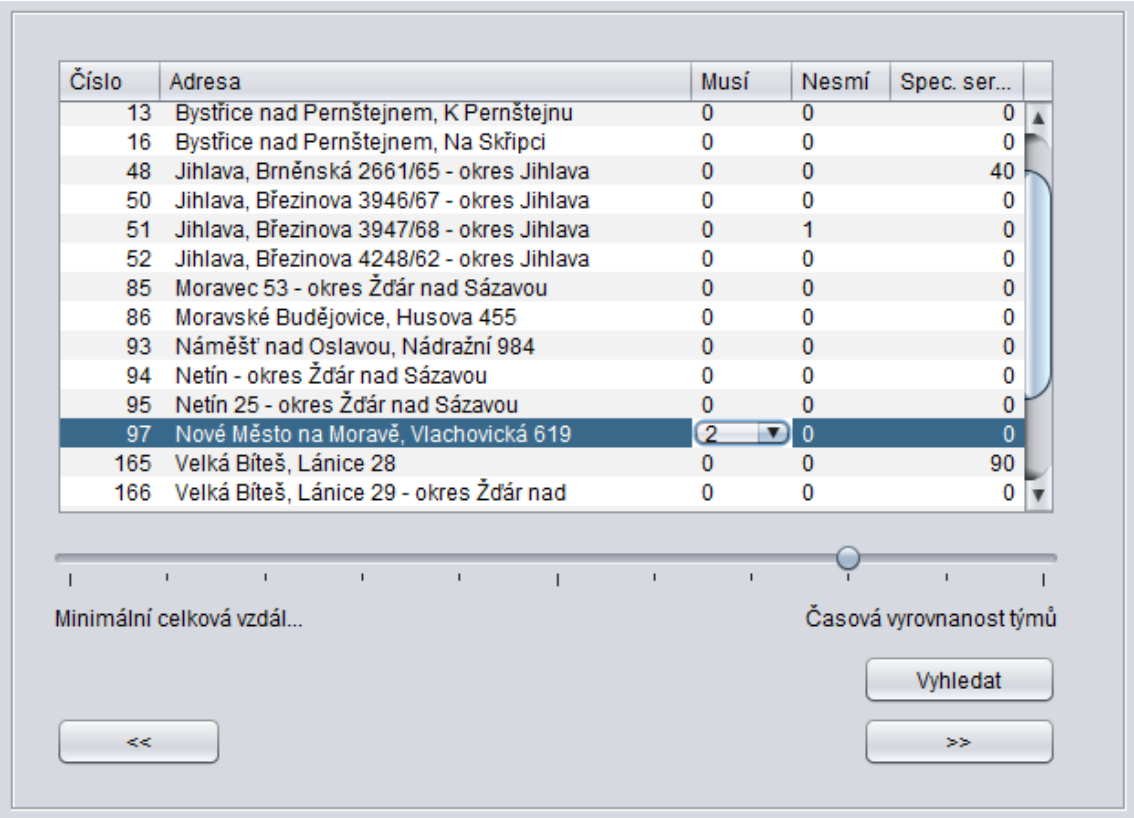
7.4.1 Výsledná aplikace

V úvodním panelu (obr. 7.3) si uživatel vybere místa, která v daný den potřebují servis (v poli hledat může vyfiltrovat z nabídky místa obsahující pouze zadané znaky), zvolí počet týmů a standardní dobu servisu. Kliknutím na tlačítko Defaultní může odznačit všechna místa a zrušit nastavení z dalšího panelu. Kliknutím na tlačítko vpravo dole potvrdí výběr a přejde na druhý panel.



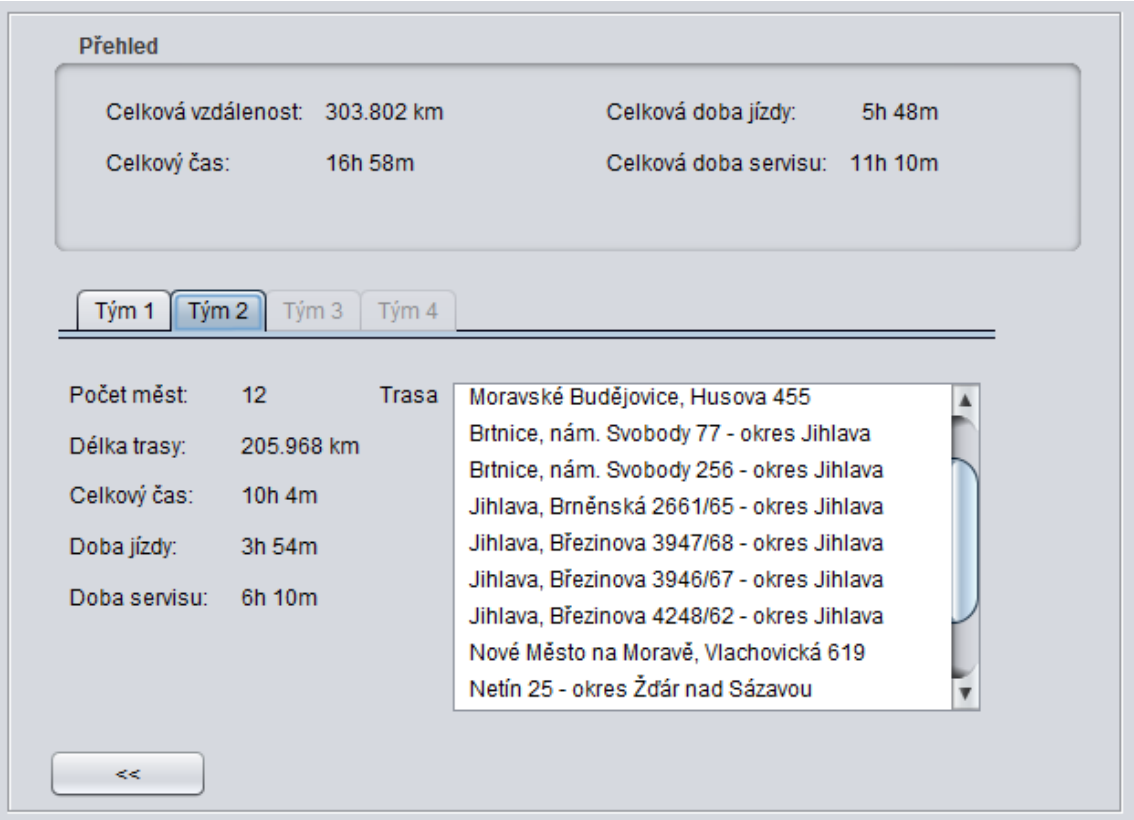
Obrázek 7.3: Aplikace - úvodní panel

V druhém panelu (obr. 7.4) se nachází seznam vybraných míst a je zde možné nastavit dodatečné parametry – vybrat tým, který musí dané místo navštívit, nebo naopak nesmí, speciální dobu servisu daného místa a váhu, která určuje, do jaké míry má vyhledávání probíhat s ohledem na minimální celkovou vzdálenost a do jaké s ohledem na vyrovnanost časových náročností tras včetně dob servisů.



Obrázek 7.4: Aplikace - druhý panel

V posledním panelu (obr. 7.5) je přehled obsahující celkovou vzdálenost, čas, dobu jízdy a dobu servisu všech týmů. Stejné údaje jsou dostupné pro jednotlivé týmy, kde je navíc uvedena nalezená trasa a počet míst, jež na této trase potřebují servis.



Obrázek 7.5: Aplikace – třetí panel

7.5 Případová studie

V této kapitole bude uvedeno srovnání plánů poskytnutých společností VÝTAHY, s.r.o., které byly vytvořeny bez specializovaného softwaru, a plánů vytvořených navrženou aplikací při stejném zadání.

7.5.1 Den 1

7.5.1.1 Původní plán

Celková délka tras byla v tomto dni 282,13 km. První a třetí tým jeli tento den do sídla společnosti. Bližší informace jsou uvedeny v tabulce 7.1.

Tabulka 7.1: Plán prvního dne (Zdroj: Vlastní zpracování dle dat poskytnutých společností)

Tým 1 108,86 km	Tým 2 89,18 km	Tým 3 84,09 km
Tasov 7 - okres Žďár nad Sázavou	Tasov 258 - okres Žďár nad Sázavou	Nový Telečkov 31 - okres Třebíč
Velké Meziříčí, Vrchovecká 216	Hrotovice, F. B. Zvěřiny 220	Velké Meziříčí, Vrchovecká 216
Kouty 62 - okres Třebíč	Hrotovice, F. B. Zvěřiny 215	Vel. Meziříčí, Hornoměstská 383/60
Kouty 1 - okres Třebíč	Hrotovice, F. B. Zvěřiny 212	Velké Meziříčí, Hornoměstská 36

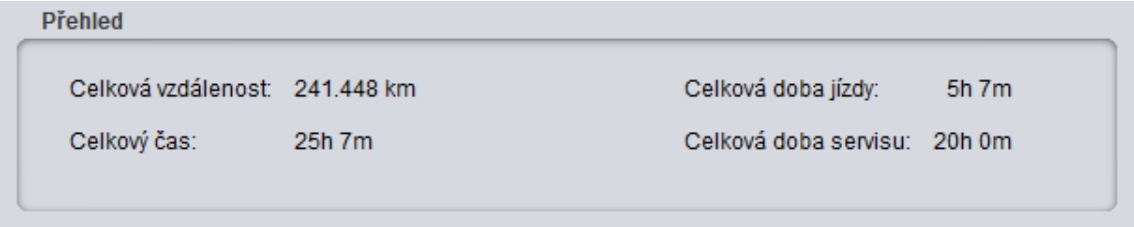
Brtnice, nám. Svobody 256	Rouchovany 187 - okres Třebíč	Velké Meziříčí, Ostrůvek 290/6
Brtnice, nám. Svobody 77	Rouchovany 407 - okres Třebíč	Velké Meziříčí, Bezděkov 148/66
Brtnice, Školní 191 - okres Jihlava	Rouchovany 36 - okres Třebíč	Velké Meziříčí, Bezděkov 444/62
Třešť, Palackého 1352/39	Rouchovany 31 - okres Třebíč	Měřín, Otínská 570 - okres Žďár n. S.
Třešť, Tovární - okres Jihlava	Mohelno 523 - okres Třebíč	Měřín, Otínská 32 - okres Žďár n. S.
Třešť, Tovární 389/1	Dukovany 2 - okres Třebíč	Velké Meziříčí, Třebíčská 1297/52
Třešť, Nerudova 1125/4 - okres Jihlava	Náměšť nad Oslavou, Masarykovo nám. 585	Velké Meziříčí, Bezručova 1354
Třešť, Nerudova 1126/6 - okres Jihlava	Náměšť nad Oslavou, Masarykovo nám.	Velké Meziříčí, Družstevní 1435/2A
Třebíč, Týn, Lavického 406 - okres Třebíč	Náměšť nad Oslavou, Nádražní 984	Velké Meziříčí, Družstevní 1173/2
Tasov 242 - okres Žďár nad Sázavou	Náměšť nad Oslavou, Nádražní 984	Velké Meziříčí, Náměstí 16/18
	Náměšť nad Oslavou, Husova 898	Nový Telečkov - okres Třebíč
	Náměšť nad Oslavou, Havlíčkova 211	
	Tasov 258 - okres Žďár nad Sázavou	
		Celkem 282,13 km

7.5.1.2 Plán vytvořený aplikací

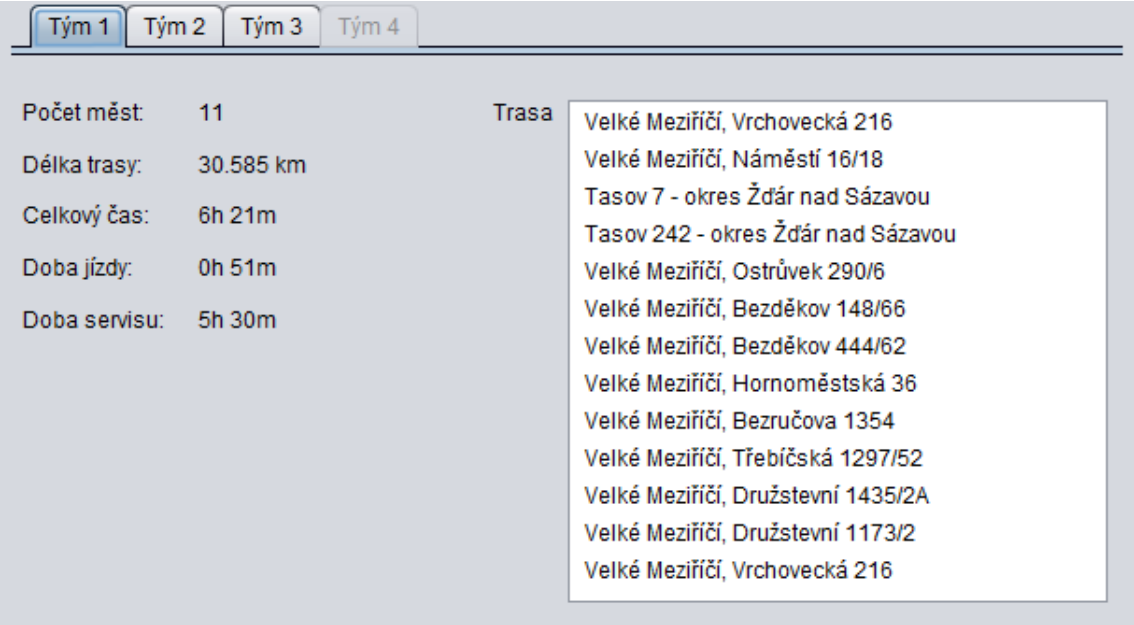
Zde bylo třeba nastavit, aby první a třetí tým navštívil sídlo společnosti. Poté, aby trasa těchto dvou týmů obsahovala počáteční místa v jejich původním plánu, bylo nutné přidat omezující podmínky, aby první tým navštívil adresu Tasov 7 a třetí tým adresu Nový Telečkov 31. Váhu bylo možné nastavit na 0,4, jelikož i v tomto případě byla časová náročnost tras poměrně vyrovnaná. Celková délka tras by činila 241,448 km, úspora zhruba 40 km.

Volání metody komponenty vypadalo takto:

```
MTSP_solver([234,131,234], [5,6,7,29,37,38,39,72,73,77,78,82,89,90,91,
92,93,98,99,108,109,110,111,129,133,146,153,154,157,159,182,183,184,
194,196,199,202,215,220,230], 3, {[3,99,1];[1,133,1]}, 0.40, 30, {});
```

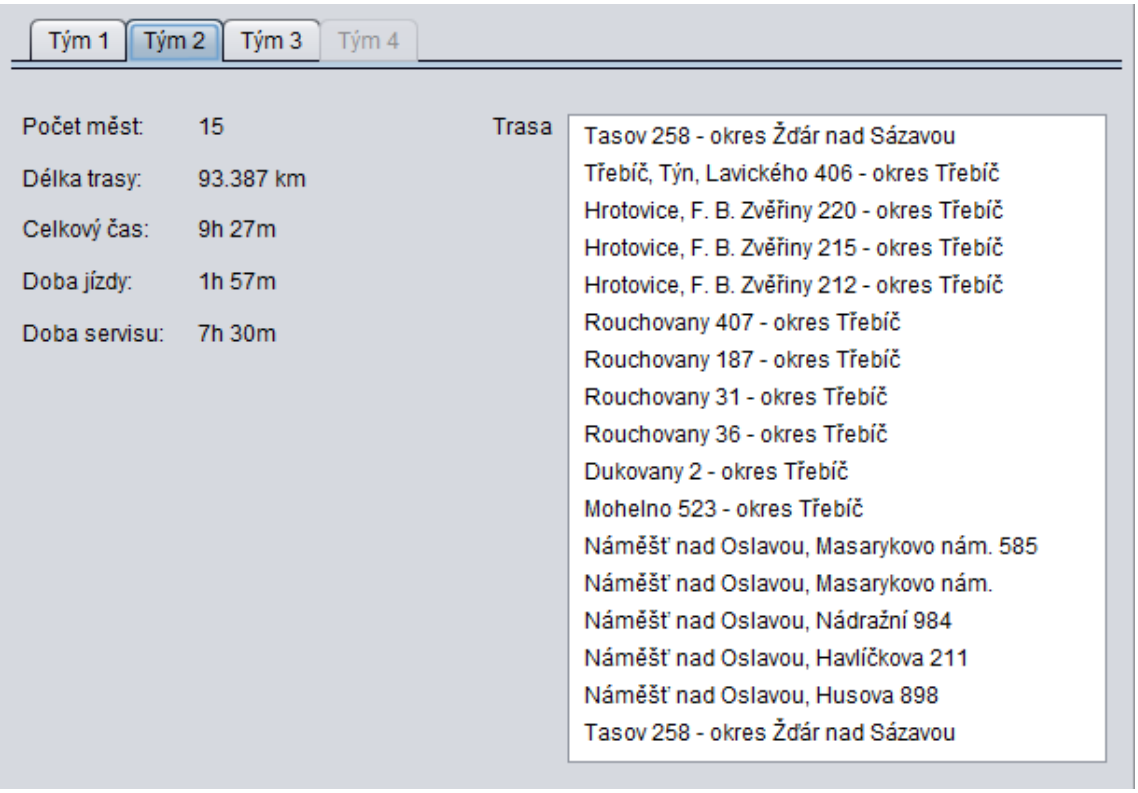


Obrázek 7.6: Plán vytvořený aplikací - den 1 - přehled

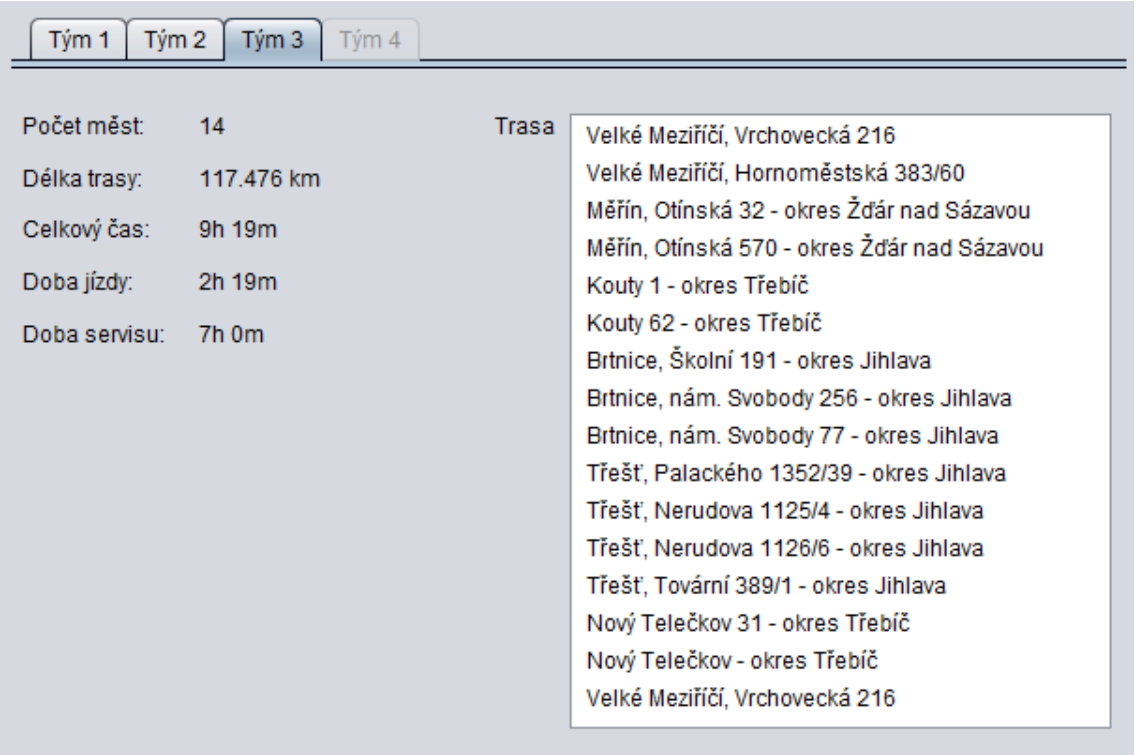


Obrázek 7.7: Plán vytvořený aplikací - den 1 - tým 1

Pozn. k týmu 1 (obr. 7.7): Trasa ve vytvořeném plánu sice začíná na Vrchovecké 216, protože byla nastavena jako základna, avšak vzhledem k tomu, že trasa je uzavřená, může začínat i na adrese Tasov 7 (stejná situace je u třetího týmu). Původní plán nepočítal se zakončením ve výchozí adrese, bylo by tedy možné od celkové délky trasy odečíst vzdálenost mezi místy Tasov 7 a Tasov 242, ale vzhledem k tomu, že tato místa jsou velmi blízko, není nutné se tím zabývat.



Obrázek 7.8: Plán vytvořený aplikací - den 1 - tým 2



Obrázek 7.9: Plán vytvořený aplikací - den 1 - tým 3

7.5.2 Den 2

7.5.2.1 Původní plán

Celková délka tras byla v tomto dni 418,57 km. Žádný z týmů nejel do sídla společnosti. Bližší informace jsou uvedeny v tabulce 7.2.

Tabulka 7.2: Plán druhého dne (Zdroj: Vlastní zpracování dle dat poskytnutých společností)

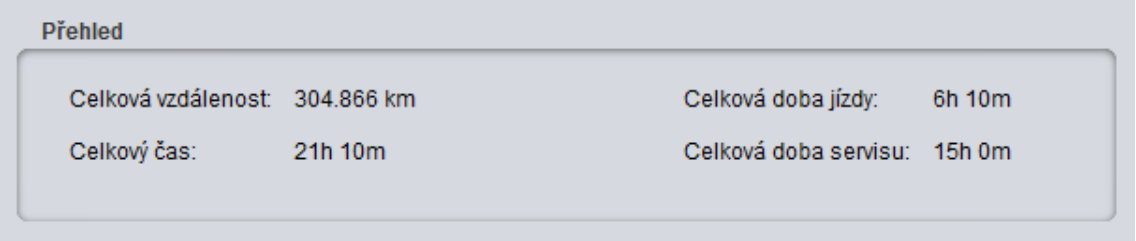
Tým 1 **138,55 km**	**Tým 2** **142,32 km**	**Tým 3** **137,7 km**
Tasov 258 - okres Žďár nad Sázavou	Tasov 6 - okres Žďár nad Sázavou	Nový Telečkov - okres Třebíč
Dobronín, Za Školou 71/4 - okres Jihlava	Třešť, Nerudova 1124/2 - okres Jihlava	Velké Meziříčí, Poštovní 1832/16
Dobronín, Za Školou 163/6a - okres Jihlava	Třešť, Nerudova 1125/4 - okres Jihlava	Vír 162 - okres Žďár nad Sázavou
Dobronín, Za Školou 320/18 - okres Jihlava	Telč, Telč-Staré Město, Hradecká 235	Jimramov, Benátky, Ubušínská 20
Polná, Varhánkova 263 - okres Jihlava	Třešť, Palackého 1352/39 - okres Jihlava	Jimramov, Benátky, Ubušínská 7
Velká Bíteš, U Stadionu 574	Třešť, Nerudova 1124/2 - okres Jihlava	Jimramov, Padělek - okres Žďár nad Sázavou
Velká Bíteš, Lánice 29	Třešť, Luční 31/23 - okres Jihlava	Jimramov, Padělek 313 - okres Žďár nad Sázavou
Velká Bíteš, Vlkovská 217	Třebíč, Vnitřní Město, Karlovo nám. 147/17	Velké Meziříčí, Kostelní 93/11
Velká Bíteš, Vlkovská 212	Třebíč, Vnitřní Město, Karlovo nám. 21/15	Velké Meziříčí, Kostelní 476/13
Tasov 258 - okres Žďár nad Sázavou	Tasov 242 - okres Žďár nad Sázavou	Velké Meziříčí, Zdenky Vorlové
Tasov 249 - okres Žďár nad Sázavou		Velké Meziříčí, Novosady 1147/77
Velké Meziříčí, Třebíčská 1297/52		Velké Meziříčí, Třebíčská 342/10
Velké Meziříčí 1265		Nový Telečkov - okres Třebíč
Tasov 258 - okres Žďár nad Sázavou		
		Celkem **418,57 km**

7.5.2.2 Plán vytvořený aplikací

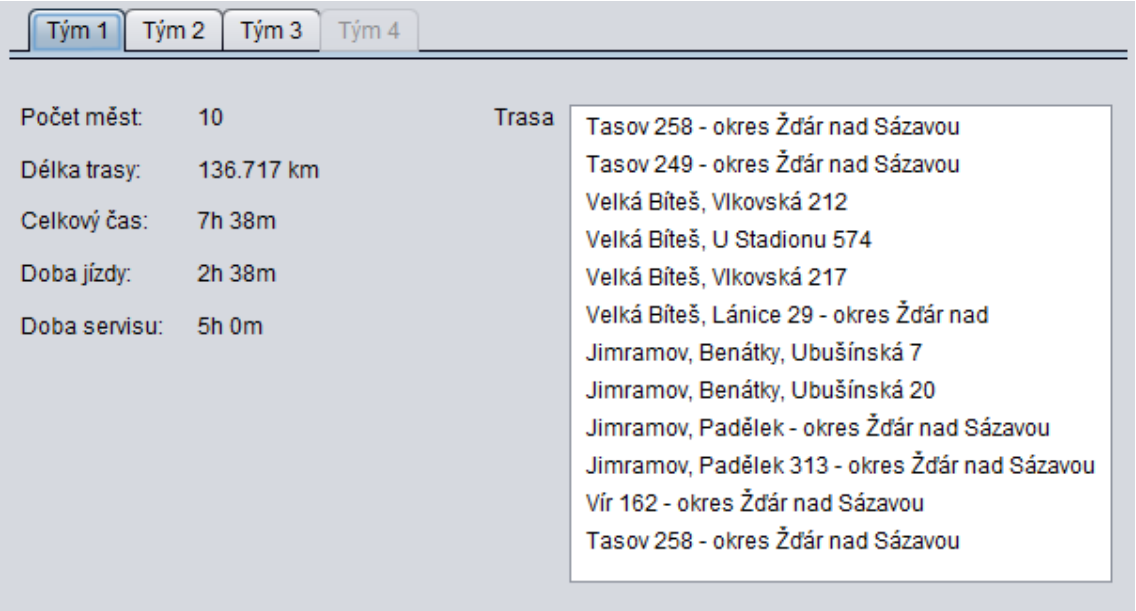
V tomto případě nebylo nutné nastavovat žádná omezení. Celková délka tras by činila 304,866 km, úspora zhruba 113 km.

Volání metody komponenty vypadalo takto:

```
MTSP_solver([131,132,98],    [26,27,28,64,65,66,67,107,135,147,148,151,
152,153,157,166,174,176,177,180,208,209,218,223,230,231,236,238],    3,
{}, 0.80, 30, {});
```



Obrázek 7.10: Plán vytvořený aplikací - den 2 - přehled



Obrázek 7.11: Plán vytvořený aplikací - den 2 - tým 1

Obrázek 7.12: Plán vytvořený aplikací - den 2 - tým 2

Obrázek 7.13: Plán vytvořený aplikací - den 2 - tým 3

7.5.3 Den 3

7.5.3.1 Původní plán

Celková délka tras byla v tomto dni 341,22 km. Všechny týmy jely tento den do sídla společnosti. Bližší informace jsou uvedeny v tabulce 7.3.

Tabulka 7.3: Plán třetího dne (Zdroj: Vlastní zpracování dle dat poskytnutých společností)

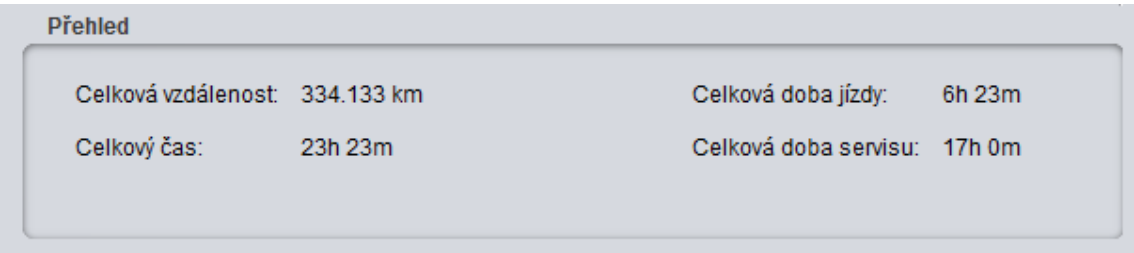
Tým 1 107,93 km	Tým 2 134,19 km	Tým 3 102,97 km
Tasov 242 - okres Žďár nad Sázavou	Tasov 249 - okres Žďár nad Sázavou	Nový Telečkov 31 - okres Třebíč
Velké Meziříčí, Vrchovecká 216	Stránecká Zhoř 35 - okres Žďár nad Sázavou	Velké Meziříčí, Vrchovecká
Velké Meziříčí, Mostiště	Stránecká Zhoř 55 - okres Žďár nad Sázavou	Velké Meziříčí, Vrchovecká 216
Jihlava, Březinova 3946/67	Velké Meziříčí, Vrchovecká 216	Nové Město na Moravě, Vlachovická 619
Jihlava, Březinova 3947/68	Velké Meziříčí, Vrchovecká	Nové Město na Moravě, Vlachovická
Jihlava, Březinova - okres Jihlava	Velká Bíteš, Vlkovská 433	Bystřice nad Pernštejnem, Luční 764
Jihlava, Březinova 4248/62	Žďár nad Sázavou, Žďár nad Sázavou 1	Bystřice nad Pernštejnem, Nová čtvrť 638
Jihlava, 17. listopadu 5259/8	Sviny 13 - okres Žďár nad Sázavou	Bystřice nad Pernštejnem 492
Jihlava, Vrchlického 2078/12	Tasov 258 - okres Žďár nad Sázavou	Bystřice nad Pernštejnem, Příční 744
Jihlava, Mahenova 1699/3 - okres Jihlava		Bystřice nad Pernštejnem 492
Jihlava, Mahenova 2434/24 - okres Jihlava		Bystřice nad Pernštejnem, Luční 764
Jihlava, Brněnská - okres Jihlava		Velké Meziříčí, Poštovní 1831/14
Kamenice 496 - okres Jihlava		Velké Meziříčí, Poštovní 1832/16
Kamenice - okres Jihlava		Velké Meziříčí, Čechova 1726/34
Kamenice 189 - okres Jihlava		
Kamenice 38 - okres Jihlava		
Kamenice 496 - okres Jihlava		
Tasov - okres Žďár nad Sázavou		
Tasov 242 - okres Žďár nad Sázavou		
		Celkem 341,22 km

7.5.3.2 Plán vytvořený aplikací

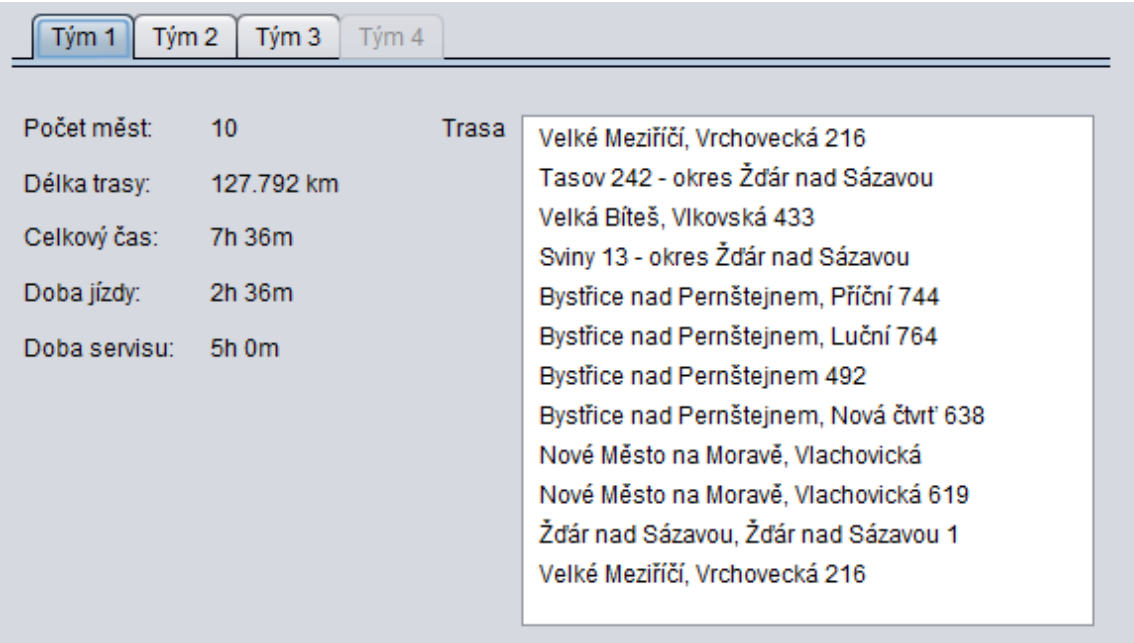
V tomto případě bylo nutné nastavit, aby všechny týmy navštívily sídlo společnosti, dále pak omezující podmínky, aby první tým navštívil adresu Tasov 242, druhý Tasov 249 a třetí Nový Telečkov 31 a Velké Meziříčí, Čechova 1726/34. Výchozí adresa třetího týmu neodpovídá jeho cílové adrese, proto je korektní od celkové délky trasy 334,133 km odečíst 9,5 km, což je vzdálenost mezi výchozím a cílovým místem. Celková délka tras by tedy po korekci činila 324,633 km, úspora zhruba 16 km.

Volání metody komponenty vypadalo takto:

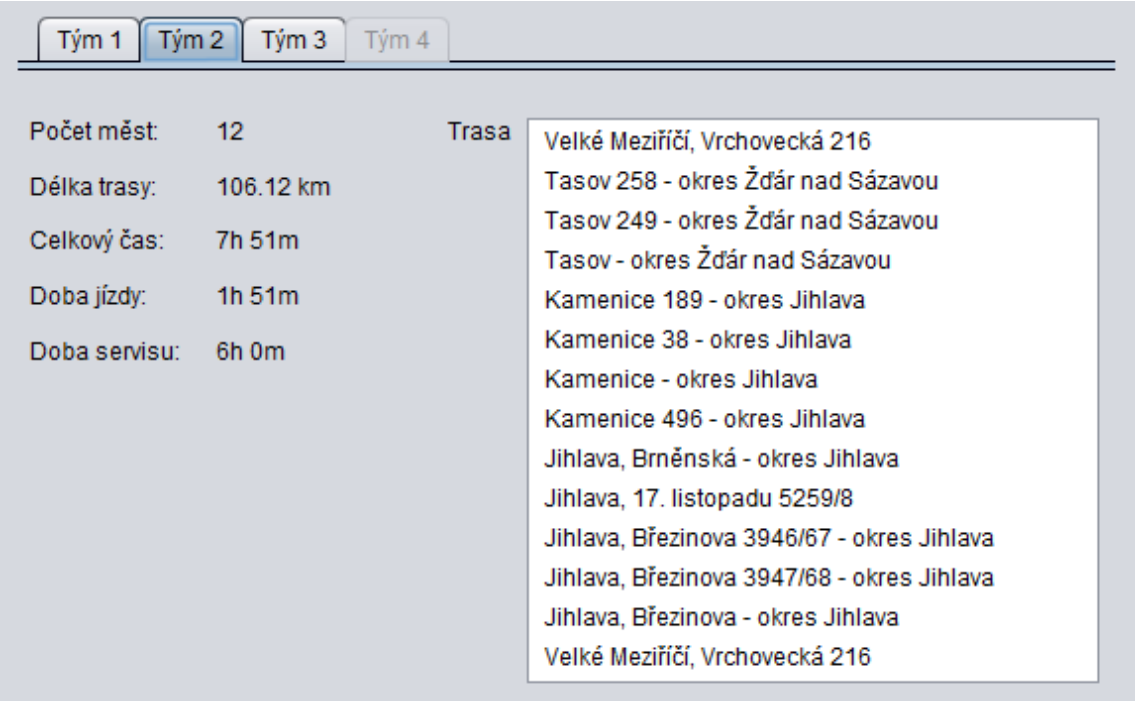
```
MTSP_solver([234,234,234],[11,14,20,23,46,47,49,50,51,52,58,59,62,68,
69,70,71,96,97,99,117,118,123,124,129,130,131,179,192,213,222,223,233,
242], 3, {[3,99,1];[1,129,1];[2,130,1];[3,192,1]}, 0.79, 30, {});
```



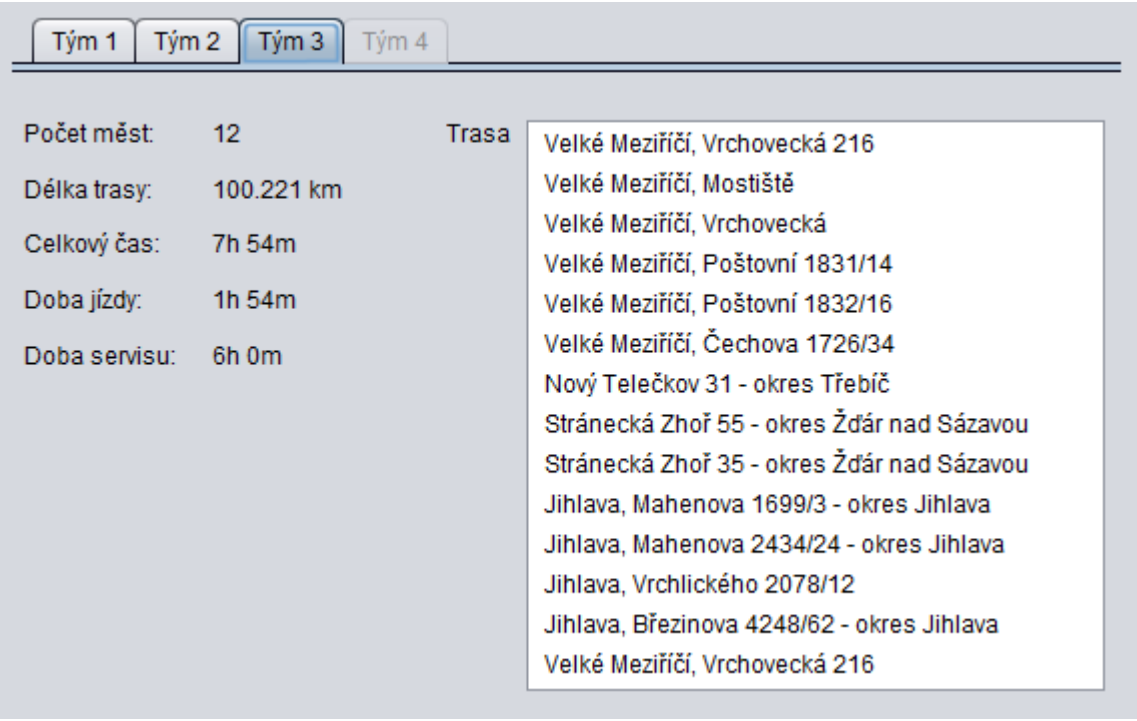
Obrázek 7.14: Plán vytvořený aplikací - den 3 - přehled



Obrázek 7.15: Plán vytvořený aplikací - den 3 – tým 1



Obrázek 7.16: Plán vytvořený aplikací - den 3 - tým 2



Obrázek 7.17: Plán vytvořený aplikací - den 3 - tým 3

7.5.4 Den 4

7.5.4.1 Původní plán

Celková délka tras byla v tomto dni 498,17 km. Pouze třetí tým jel do sídla společnosti. Bližší informace jsou uvedeny v tabulce 7.4.

Tabulka 7.4: Plán čtvrtého dne (Zdroj: Vlastní zpracování dle dat poskytnutých společností)

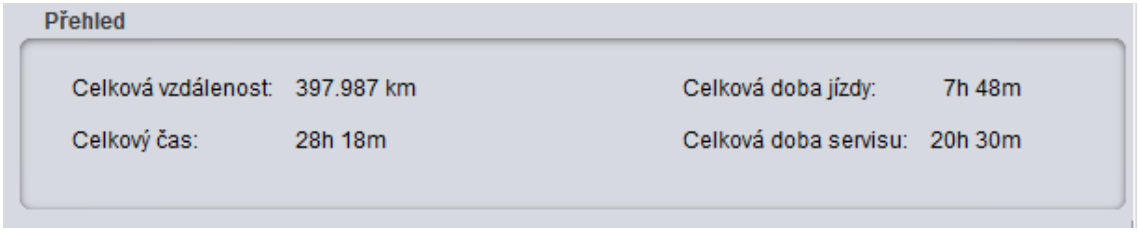
Tým 1 142,45 km	Tým 2 156,84 km	Tým 3 198,88 km
Tasov 212 - okres Žďár nad Sázavou	Tasov 7 - okres Žďár nad Sázavou	Nový Telečkov 6 - okres Třebíč
Rozsochy 146 - okres Žďár nad Sázavou	Dačice, Dačice III, Strojírenská 160	Velké Meziříčí, Bezručova 1543/12
Rozsochy 2 - okres Žďár nad Sázavou	Dačice, Dačice I, Neulingerova 151	Velké Meziříčí, Vrchovecká 216
Rozsochy 10 - okres Žďár nad Sázavou	Třešť, Wolkerova 673/16 - okres Jihlava	Velké Meziříčí, Zdenky Vorlové 2001
Žďár nad Sázavou, Žďár nad Sázavou 1	Třešť, Nerudova 1271/20 - okres Jihlava	Velké Meziříčí, Oslavická 1969/62
Žďár nad Sázavou, Žďár nad Sázavou 3	Třešť, Nerudova 1322/26 - okres Jihlava	Bystřice nad Pernštejnem, K Pernštejnu
Polná, Janovice 66 - okres Jihlava	Třešť, Nerudova 1126/6 - okres Jihlava	Bystřice nad Pernštejnem, Nádražní
Velké Meziříčí, Sokolovská 272/30	Jihlava, Masarykovo náměstí - okres Jihlava	Bystřice nad Pernštejnem, Nádražní 491
Velké Meziříčí, Sokolovská 269/32	Jihlava, Masarykovo náměstí 1096/32	Havlíčkův Brod, Havlíčkovo náměstí 1963
Velká Bíteš, Masarykovo náměstí 4	Tasov 249 - okres Žďár nad Sázavou	Velké Meziříčí, Hornoměstská 864/39
Velká Bíteš, Hrnčířská 128	Tasov 242 - okres Žďár nad Sázavou	Velké Meziříčí, Hornoměstská 357/25
Velká Bíteš, Vlkovská 433		Velké Meziříčí, Hornoměstská
Velká Bíteš, Vlkovská 429		Velké Meziříčí, Kostelní 93/11
Velká Bíteš, Vlkovská 212		Velké Meziříčí, Kostelní 476/13
Velká Bíteš, Lánice 28		Nový Telečkov - okres Třebíč
Velká Bíteš, Lánice 29		
Velká Bíteš, Janovice, Návrší 290		
Tasov 258 - okres Žďár nad Sázavou		
		Celkem 498,17 km

7.5.4.2 Plán vytvořený aplikací

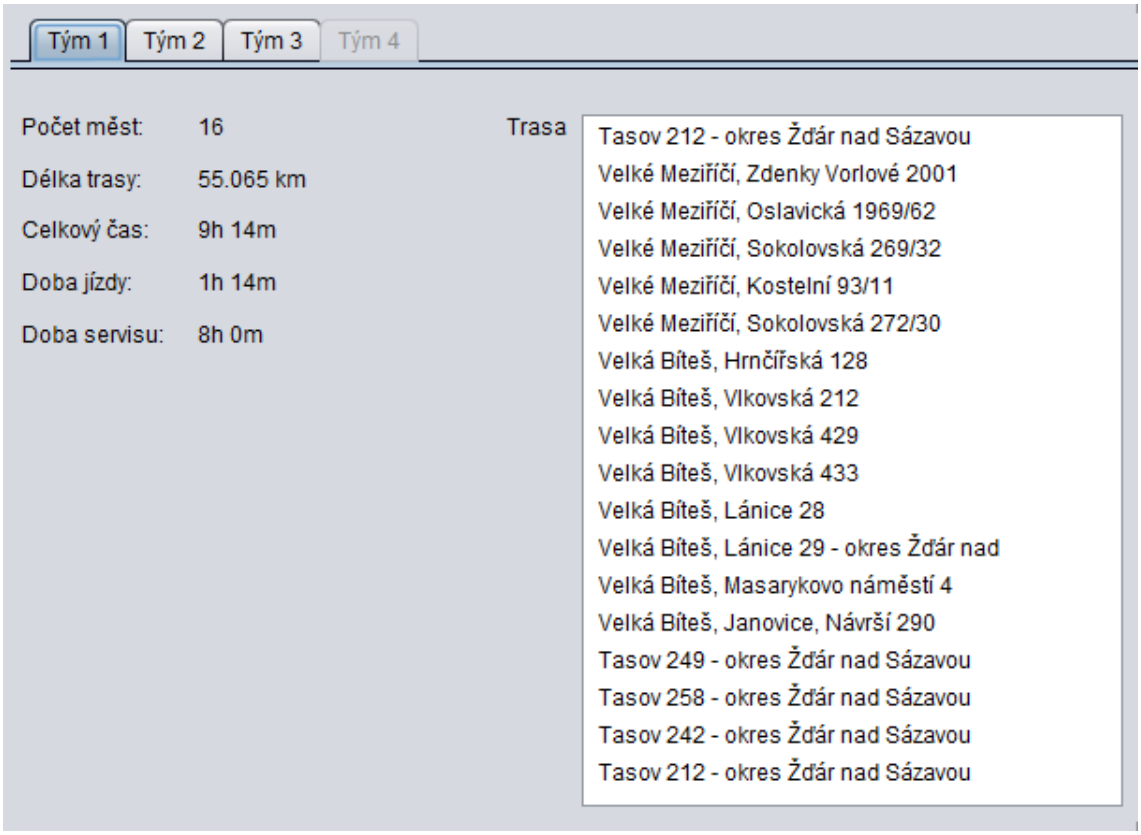
Zde bylo nutné prvnímu týmu nastavit omezující podmínku, aby navštívil sídlo společnosti. Při váze nastavené na hodnotu 0,8 měla celková vzdálenost hodnotu 351,455 km, ale časová nevyrovnanost týmů byla značná. Proto se váha zvýšila na hodnotu 0,86 a celková vzdálenost by pak byla 397,987 km, úspora zhruba 100 km.

Volání metody komponenty vypadalo takto:

```
MTSP_solver([128,133,100],[13,17,19,24,25,30,60,61,98,106,112,113,114,
129,130,131,154,155,156,160,163,164,165,166,168,176,178,179,186,197,
198,204,208,209,219,227,228,234,237,242,243],  3,  {[3,234,1]},  0.86,
30, {});
```



Obrázek 7.18: Plán vytvořený aplikací - den 4 - přehled



Obrázek 7.19: Plán vytvořený aplikací - den 4 - tým 1

Obrázek 7.20: Plán vytvořený aplikací - den 4 - tým 2

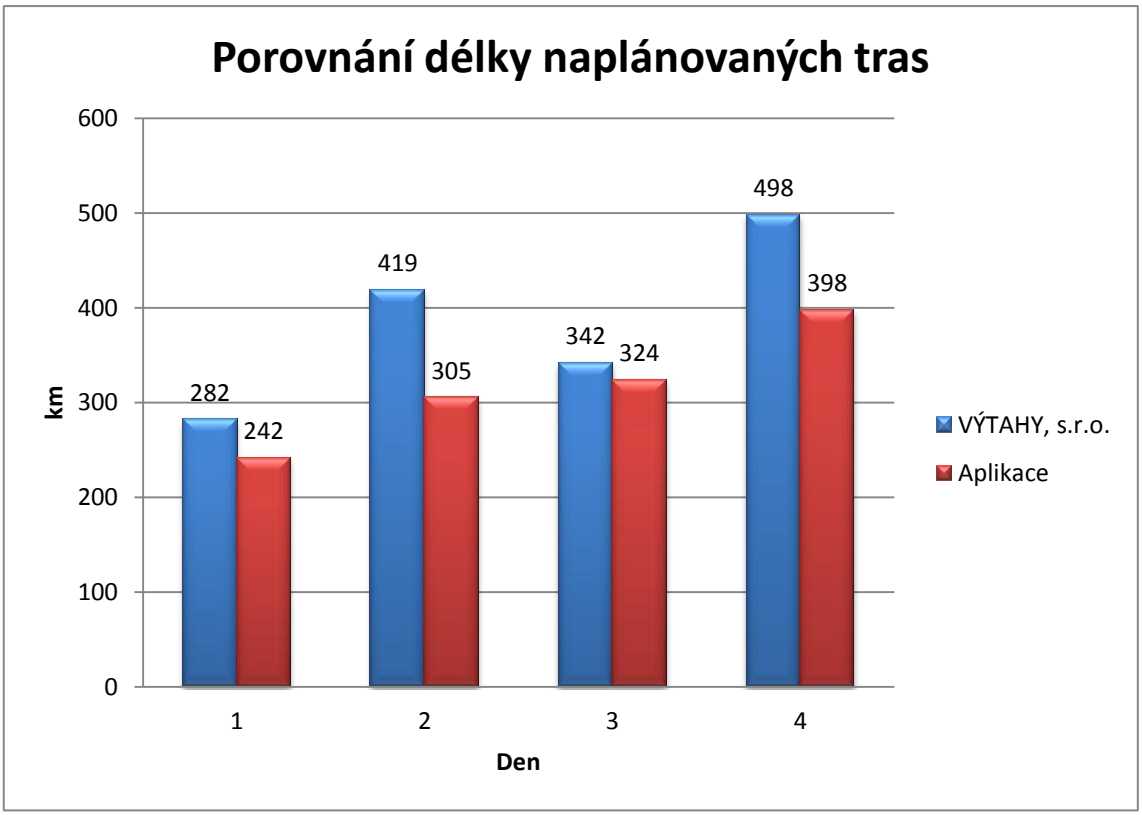
Obrázek 7.21: Plán vytvořený aplikací - den 4 - tým 3

7.6 Zhodnocení ekonomických přínosů

Ve všech dnech aplikace dokázala nalézt kratší trasy, konkrétní hodnoty jsou uvedeny v tabulce 7.5 a na grafu 7.1. Nalezené řešení, a tedy i velikost úspor se odvíjí od omezujících podmínek, dob servisu a především velikosti parametru, který udává, zda se více orientujeme na minimální celkovou vzdálenost anebo časovou vyrovnanost týmů. V testech byla zvolena průměrná doba servisu 30 minut. Je třeba vzít v úvahu, že velikost úspor byla vypočtena na základě celkové vzdálenosti, a to v ideálním případě, kdy by na silnicích nebyly žádné uzavírky a technici nemuseli měnit trasu.

Tabulka 7.5: Porovnání plánů

původní plán [km]	plán aplikace [km]	úspora	úspora [Kč]
282	242	14%	520
419	305	27%	1482
342	324	5%	234
498	398	20%	1300



Graf 7.1: Porovnání plánů

8 Závěr

Hlavním cílem diplomové práce bylo vytvoření nástroje pro podporu rozhodování managementu v oblasti plánování tras servisním technikům.

Po podrobné analýze problému a prostudování dostupných informací se ukázalo, že se jedná o problém více obchodních cestujících s více základnami a omezujícími podmínkami. Vzhledem k výpočetní složitosti tohoto problému při hledání optimálního řešení, bylo nutné použít heuristické metody či metody náhodného zlepšení. Jako nejvhodnější se jevily genetické algoritmy, které umožňovaly pokrýt všechny požadavky kladené na výslednou aplikaci a jejichž implementace je dostupná v softwaru MATLAB. Byl tedy proveden návrh a implementace aplikace v tomto programu. Jelikož ne každá společnost produkt MATLAB vlastní a protože se jedná o nezanedbatelnou investici, bylo využito možnosti exportu programu do komponenty v Javě. Ta pro svůj běh vyžaduje tzv. MATLAB Compiler Runtime, který je poskytován zdarma. V Javě bylo následně implementováno také grafické uživatelské rozhraní usnadňující práci s výslednou aplikací.

Aplikace umožňuje vybrat místa, která mají být v daný den navštívena, zvolit počet týmů, nastavit základny pro jednotlivé týmy, předpokládanou dobu servisu pro všechna místa současně i jednotlivě. Dále je také možné nastavit omezující podmínky v tom smyslu, že některá místa musí nebo naopak nesmí navštívit vybraný tým. Velký vliv na výsledný plán má nastavení poměru mezi minimální celkovou ujetou vzdáleností a časově vyrovnanou vytížeností týmů. Na základě vstupních parametrů aplikace vyhledá trasy pro jednotlivé týmy. U každé trasy je zobrazena její délka v kilometrech, počet servisních míst, čas strávený na cestě, servisem a celkový čas. V celkovém přehledu jsou pak uvedeny součty těchto údajů pro všechny týmy. V případové studii bylo ověřeno, že výsledná aplikace oproti ručnímu plánování tras bez specializovaného softwaru umožňuje snížení nákladů na dopravu.

Seznam použité literatury

[1] An Introduction to NetBeans. *NetBeans* [online]. © 2012 [cit. 2012-05-12]. Dostupné z: http://netbeans.org/about/index.html

[2] APPLEGATE, D. L., R. E. BIXBY, V. CHVÁTAL a W. J. COOK. *The Traveling Salesman Problem*: *A Computational Study* [online]. 2007 [cit. 2012-03-18]. Dostupné z: http://press.princeton.edu/chapters/s8451.pdf

[3] Applications of the TSP. *Traveling Salesman Problem* [online]. 2007 [cit. 2012-04-22]. Dostupné z: http://www.tsp.gatech.edu/apps/index.html

[4] CARTER, A. E. *Design and Application of Genetic Algorithms for the Multiple Traveling Salesperson Assignment Problem* [online]. Virginia Polytechnic Institute and State University, 2003 [cit. 2012-03-15]. Dostupné z: http://scholar.lib.vt.edu/theses/available/etd-04252003-123556/unrestricted/Dissertation.pdf

[5] ČEŠKA, M., T. VOJNAR a A. SMRČKA. *Teoretická informatika*: *Studijní opora* [online]. 2011 [cit. 2012-03-03]. Dostupné z: http://www.fit.vutbr.cz/study/courses/TIN/public/Texty/oporaTIN.pdf

[6] DAVENDRA, D. *Traveling Salesman Problem, Theory and Applications* [online]. InTech, 2010 [cit. 2012-03-10]. ISBN 978-953-307-426-9. Dostupné z: http://www.intechopen.com/books/traveling-salesman-problem-theory-and-applications

[7] DOSTÁL, P. *Pokročilé metody analýz a modelování v podnikatelství a veřejné správě*. Vyd. 1. Brno: Akademické nakladatelství CERM, 2008, 340 s. ISBN 978-80-7204-605-8.

[8] Fitness function. *Wikipedia*: *the free encyclopedia* [online]. 2011 [cit. 2012-04-22]. Dostupné z: http://en.wikipedia.org/wiki/Fitness_function

[9] Genetic Algorithm Options: Options Reference. *MathWorks - MATLAB and Simulink for Technical Computing* [online]. © 1984-2012 [cit. 2012-05-12]. Dostupné z: http://www.mathworks.com/help/toolbox/gads/f6174dfi10.html

[10] Global Optimization Toolbox: Documentation. *MathWorks - MATLAB and Simulink for Technical Computing* [online]. © 1984-2012 [cit. 2012-05-12]. Dostupné z: http://www.mathworks.com/help/toolbox/gads/gads_product_page.html

[11] Historie společnosti. *Výtahy s. r. o. - Modernizace a výroba výtahů* [online]. © 2010 [cit. 2012-04-25]. Dostupné z: http://www.vytahy.com/historie-spolecnosti.html

[12] HOFFMAN, K. a M. PADBERG. Traveling Salesman Problem. *Karla Hoffman* [online]. [cit. 2012-03-15]. Dostupné z: http://iris.gmu.edu/~khoffman/papers/trav_salesman.html

[13] How the Genetic Algorithm Works: Using the Genetic Algorithm. *MathWorks - MATLAB and Simulink for Technical Computing* [online]. © 1984-2012 [cit. 2012-04-22]. Dostupné z: http://www.mathworks.com/help/toolbox/gads/f6187.html

[14] HYNEK, J. *Genetické algoritmy a genetické programování*. 1. vyd. Praha: Grada, 2008, 182 s. ISBN 978-80-247-2695-3.

[15] Java (programming language). *Wikipedia*: *the free encyclopedia* [online]. 2012 [cit. 2012-05-22]. Dostupné z: http://en.wikipedia.org/wiki/Java_(programming_language)

[16] Java (programovací jazyk). *Wikipedie*: *otevřená encyklopedie* [online]. 2012 [cit. 2012-05-12]. Dostupné z: http://cs.wikipedia.org/wiki/Java_(programovac%C3%AD_jazyk)

[17] JÜNGER, M., G. REINELT a G. RINALDI. *THE TRAVELING SALESMAN PROBLEM* [online]. 1994 [cit. 2012-04-22]. Dostupné z: www.iasi.cnr.it/reports/R375/R375.pdf

[18] LARRAÑAGA, P., C. M. H. KUIJPERS, R. H. MURGA, I. INZA a S. DIZDAREVIC. Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators. In: KUIJPERS, I. INZA. *The Artificial intelligence review* [online]. 2. vyd.: Springer Netherlands, 1999 [cit. 2012-05-22]. ISSN 0269-2821. DOI: 10.1023/A:1006529012972. Dostupné z: http://dx.doi.org/10.1023/A:1006529012972

[19] LETCHFORD, A. N. *The Travelling Salesman Problem* [online]. 2010. vyd. [cit. 2012-04-22]. Dostupné z: www.lancs.ac.uk/staff/letchfoa/talks/TSP.pdf

[20] MATLAB - Jazyk pro technické výpočty. *Humusoft*: *Technické výpočty, řídicí technika, simulace* [online]. © 1991 - 2012 [cit. 2012-05-12]. Dostupné z: http://www.humusoft.cz/produkty/matlab/matlab/

[21] MATLAB - The Language of Technical Computing. *MathWorks - MATLAB and Simulink for Technical Computing* [online]. © 1994-2012 [cit. 2012-05-12]. Dostupné z: http://www.mathworks.com/products/matlab/

[22] MATLAB Builder JA: Documentation. *MathWorks - MATLAB and Simulink for Technical Computing* [online]. © 1984-2012 [cit. 2012-05-12]. Dostupné z: http://www.mathworks.com/help/toolbox/javabuilder/

[23] MATLAB. *Wikipedie*: *otevřená encyklopedie* [online]. 2012 [cit. 2012-05-12]. Dostupné z: http://cs.wikipedia.org/wiki/MATLAB

[24] *Ministerstvo spravedlnosti České republiky*: *Obchodní rejstřík a Sbírka listin* [online]. © 2012 [cit. 2012-04-25]. Dostupné z: https://or.justice.cz/ias/ui/rejstrik-$

[25] NetBeans. *Wikipedia*: *the free encyclopedia* [online]. 2012 [cit. 2012-05-12]. Dostupné z: http://en.wikipedia.org/wiki/NetBeans

[26] NetBeans. *Wikipedie*: *otevřená encyklopedie* [online]. 2012 [cit. 2012-05-12]. Dostupné z: http://cs.wikipedia.org/wiki/NetBeans

[27] Number of Tours. *Traveling Salesman Problem* [online]. 2005 [cit. 2012-05-22]. Dostupné z: http://www.tsp.gatech.edu/problem/pcb3cnt.html

[28] Profil společnosti VÝTAHY s.r.o. *Výtahy s. r. o. - Modernizace a výroba výtahů* [online]. © 2010 [cit. 2012-04-22]. Dostupné z: http://www.vytahy.com/o-spolecnosti.html

[29] The Problem. *Traveling Salesman Problem* [online]. 2007 [cit. 2012-04-22]. Dostupné z: http://www.tsp.gatech.edu/problem/index.html

[30] Úvod. *Výtahy s. r. o. - Modernizace a výroba výtahů* [online]. © 2010 [cit. 2012-04-22]. Dostupné z: http://www.vytahy.com/

[31] MATLAB Builder JA (for Java language). *MathWorks - MATLAB and Simulink for Technical Computing* [online]. © 1994-2012 [cit. 2012-04-23]. Dostupné z: http://www.mathworks.com/products/javabuilder/

Seznam použitých zkratek

TSP problém obchodního cestujícího (traveling salesman problem)

sTSP symetrický problém obchodního cestujícího (symmetric traveling salesman problem)

aTSP asymetrický problém obchodního cestujícího (asymmetric traveling salesman problem)

mTSP problém více obchodních cestujících (multiple traveling salesman problem)

Seznam obrázků

Seznam tabulek

Seznam grafů