UNIVERZITA PALACKÉHO V OLOMOUCI
PŘÍRODOVĚDECKÁ FAKULTA
KATEDRA MATEMATICKÉ ANALÝZY A APLIKACÍ MATEMATIKY

DIPLOMOVÁ PRÁCE

Řešení VRP metodami typu tabu search

Vedoucí diplomové práce: RNDr. Pavel Ženčák, Ph.D.
Vypracovala: Bc. Michaela Kubiczková
Studijní program: N1103 Aplikovaná matematika
Studijní obor: Aplikace matematiky v ekonomii
Forma studia: prezenční
Rok odevzdání: 2015

BIBLIOGRAFICKÁ IDENTIFIKACE

Autor: Bc. Michaela Kubiczková

Název práce: Řešení VRP metodami typu tabu search.

Typ práce: Diplomová práce

Pracoviště: Katedra matematické analýzy a aplikací matematiky

Vedoucí práce: RNDr. Pavel Ženčák, Ph.D.

Rok obhajoby práce: 2015

Abstrakt: Rozvozní problém (VRP) je velmi známý a náročný kombinatorický optimalizační problém, který se používá k vytvoření optimálních rozvozních tras z jednoho nebo více centrálních skladišť, kde je daný počet vozidel, do mnoha geograficky různorodě rozptýlených míst, kde sídlí zákazníci. V nejjednodušší verzi VRP takzvané kapacitní VRP, kde všechny vozidla jsou identické a kapacity vozidel jsou dané, je cílem minimalizovat celkovou délku (nebo náklad) všech tras. Tato diplomová práce je zaměřena na tento problém a řeší ho obzvláště pomocí Granulární tabu search metody. Metoda je prezentována skrze výpočetní proceduru, kterou jsem vytvořila v softwarovém prostředí Matlab.

Klíčová slova: rozvozní problém (VRP), λ – výměny, tabu prohledávání, heuristika Clarka a Wrighta, 2 – opt heuristika, granulární tabu search metoda (GTS), tabu list, granulární mez, řídký graf, Matlab

Počet stran: 63

Počet příloh: 1

Jazyk: český

BIBLIOGRAPHICAL IDENTIFICATION

Author: Bc. Michaela Kubiczková

Title: Solution of VRP using Tabu search method.

Type of thesis: Diploma thesis

Department: Department of Mathematical Analysis and Application of Mathematics

Supervisor: RNDr. Pavel Ženčák, Ph.D

The year of presentation: 2015

Abstract: Vehicle Routing Problem (VRP) is well-known and extremely difficult combinatorial optimization problem used to designing optimal delivery routes from one or several depots, where are standing a given number of vehicles, to number of geographically scattered places or customers. In the basic version of the problem, known as capacitated VRP, all vehicles are identical, capacities of vehicles is given and the objective is to minimize the total routing length (or cost) of the routes. This thesis deals with this problem especially by Granular Tabu Search method. Method is presented via a computational procedure, which I created in Matlab.

Key words: Vehicle Routing Problem (VRP), λ – exchanges, Tabu search, Clark and Wright heuristics, 2 – opt heuristics, granular tabu search method (GTS), tabu list, granular threshold, sparse graph, Matlab

Number of pages: 63

Number of appendices: 1

Language: Czech

Prohlášení

Prohlašuji, že jsem diplomovou práci zpracovala samostatně pod vedením RNDr. Pavla Ženčáka, Ph.D. a všechny použité zdroje jsem uvedla.

V Třinci dne 3. května 2015. Podpis ……

Obsah

Poděkování

Zde bych chtěla poděkovat svému vedoucímu diplomové práce a to panu RNDr. Pavlu Ženčákovi, Ph.D. za jeho cenné rady a připomínky při vypracování této práce. Také bych chtěla vyjádřit svůj velký dík mým rodičům a mému příteli, kteří mě ve studii plně podporují.

Úvod

Cílem této práce je popsat řešení problematiky VRP metodami typu tabu search, implementovat vybrané varianty algoritmů užitím vhodného programovacího jazyka a provést výpočetní porovnání efektivity jednotlivých variant vytvořeného algoritmu. Vehicle Routing Problem neboli rozvozní problém je velice známé a velmi často studované téma. Existuje mnoho metod pro řešení tohoto problému, já se budu v této práci věnovat hlavně přístupu tabu search metod.

Nejdříve se v první kapitole zabývám rozvozním problémem jako takovým, což znamená, že je potřeba nadefinovat tento problém, jak obecně tak matematicky. Druhá kapitola je zaměřena na metody řešení rozvozního problému obecně. Stěžejním metodám tabu search je věnována třetí kapitola, kde jsou obecně popsány různé aplikace tabu search metod pro rozvozní problém. Ve čtvrté kapitole je pak podrobně popsána konkrétní metoda pro tabu search a to granulární tabu search (GTS) metoda, která je použitá v implementaci v softwaru Matlab a je důlžitá i pro pátou kapitolu, kde jsou testovány parametry potřebné pro tuto metodu. Obrázky v této práci jsou vytvořeny v programu Geogebra nebo v Matlabu, tabulky jsou vytvořeny v Excelu.

Klíčová literatura pro tuto práci se skládá zejména z odborných článků zaměřených na metody typu tabu search obecně, ale také na konkrétní metody zvlášť. Nejvíce jsem tedy využívala článek o granulárním tabu search algoritmu [5] a to i pro samotnou implementaci, dále pak internetovou stránku o rozvozním problému [8], odkud jsou stáhnuty ukázkové příklady s optimálními hodnotami a optimálními trasami využívané i pro testování.

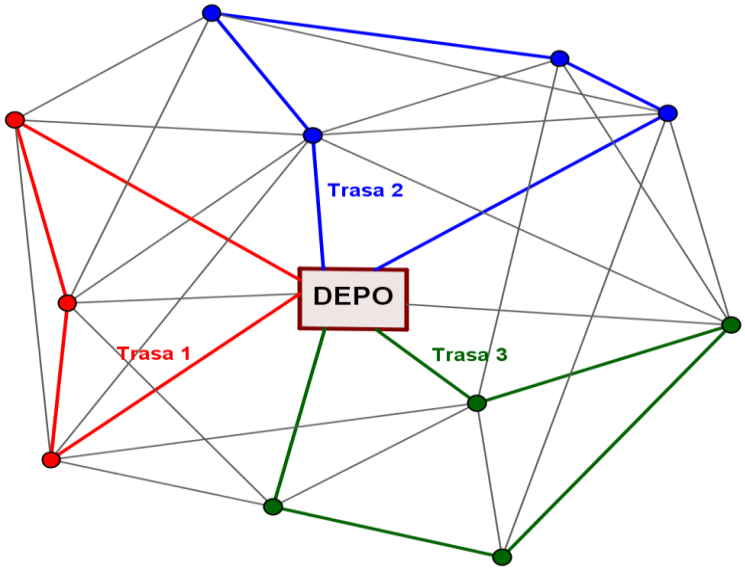
1. Rozvozní problém

Při tvorbě této kapitoly jsem použila tyto zdroje [1], [3], [4], [5], [8], [10], [11].

1.1. Popis a historie

Rozvozní problém neboli z angličtiny VRP, kde zkratka VRP označuje slova Vehicle – dopravní prostředek, Routing – trasa, Problem - problém, je známou úlohou logistiky tedy nauky, která se zabývá fyzickými toky zboží či jiných druhů zásob od dodavatele k odběrateli (zákazníkovi), toky však mohou být i informačního charakteru v písemné či ústní podobě.

Definice 1.1: Vehicle Routing Problem je popsán jako problém navržení optimálních dodávek nebo tras z jednoho nebo více centrálních skladišť (neboli depo) do mnoha geograficky různorodě rozptýlených míst, kde sídlí zákazníci. Úkolem je nalézt optimální plán přepravy tedy ten, který minimalizuje přepravní náklady.



Obrázek 1 – příklad tras

Poprvé byla tato úloha zformulována v roce 1959 matematiky G. B. Dantzigem a R. H. Ramserem v článku The Truck Dispatching Problem (rozvozní problém s nákladními auty), který publikovali v časopise Management Science. Úloha vznikla převážně proto, že v některých firemních sektorech tvoří náklady na dopravu velkou část výdajů, které se

zásadně promítají ve výsledné ceně produktu, a tím pádem je velmi výhodné snížit náklady na dopravu, co nejvíce.

Rozvozní problém vlastně zobecňuje známější úlohu a to problém obchodního cestujícího, pro jehož řešení existuje řada metod.

Definice 1.2: Problém obchodního cestujícího je diskrétní optimalizační problém, jedná se o úlohu nalezení nejkratší možné cesty procházející všemi body (vrcholy) v grafu (mapě) a vracející se nazpět do výchozího bodu.

1.2. Matematická formulace VRP

Definice 1.3: Nechť $G = (V, H)$ je orientovaný graf, kde $V = \{v_0, v_1, \dots, v_n\}$ je množina vrcholů grafu reprezentující místa, celkový počet míst je $n + 1$, a $H = \{(v_i, v_j) | v_i, v_j \in V; i \neq j\}$ je množina orientovaných hran grafu. S každou hranou (v_i, v_j) je spojena hodnota nezáporné matice vzdáleností mezi místy $C = (c_{ij})$. V některých případech může být c_{ij} interpretováno jako cestovní náklady nebo jako čas strávený na cestě. Jestliže je matice vzdáleností C symetrická, pak je vhodné množinu H nahradit množinou $E = \{(v_i, v_j) | v_i, v_j \in V; i < j\}$, která je složena z neorientovaných hran. Vrchol v_0 označuje depo neboli centrální skladiště, ve kterém je umístěno k identických vozidel. Potom se ***Vehicle Routing Problem*** definuje jako konstrukce množiny tras s minimálními náklady, počet tras označíme jako K, s těmito podmínkami:

1) každá trasa vozidlem začíná a končí v depu;
2) každé místo z $V \setminus \{v_0\}$ je navštíveno právě, jednou právě jedním vozidlem.

Dále může být požadováno splnění některých dalších podmínek, čímž dostáváme speciální VRP úlohy. Mohou to být například následující:

1) ***omezení na celkovou délku jedné cesty (Distanced - constrained VRP = DVRP)*** – celková délka jakékoliv cesty nesmí převýšit přednastavenou mez L, tato délka může být tvořena součtem vzdáleností mezi místy zahrnutých v jedné cestě nebo také časem stráveným na cestě a časem potřebným k zastavení v jednotlivých místech, aniž by se změnil přístup k řešení úlohy. A existují i úlohy, kde jsou využity oba přístupy jak omezení na čas, tak na délku.

2) ***kapacitní omezení vozidla (Capacitated VRP = CVRP)*** – s každým místem (zákazníkem) je spojena nezáporná poptávka $p_i (p_0 = 0)$ s tím, že celková poptávka každé cesty vozidlem nesmí převýšit kapacitu vozidla P;
3) ***vetší množství centrálních skladišť (Multiple VRP = MVRP)*** – vozidla mohou vyjíždět z více skladišť a pokud jsou zákazníci nashromážděni okolo skladišť, je možné rozvozní problém modelovat jako množinu nezávislých VRP s jedním depem;
4) ***časová okna (VRP with Time Windows = VRPTW)*** – město i musí být navštíveno v čase z časového intervalu $\langle a_i,\ b_i \rangle$;
5) ***uvolnění na podmínku 2) ve VRP Definici 1.2 (Periodic VRP =PVRP)*** – je povoleno obsloužení stejného zákazníka různými vozidly, pokud to sníží celkové náklady. Toto uvolnění je velice důležité v případě, že poptávky zákazníků jsou tak velké jako kapacita vozidla.
6) ***přednostní vztahy mezi dvojicemi měst*** – místo i musí být navštíveno před místem j.
7) ***omezení na počet vozidel*** – hodnota k je buď fixní konstanta, nebo je ohraničena shora hodnotou $\tilde{k}$.

2. Metody řešení rozvozního problému

K vytvoření této kapitoly bylo využito těchto zdrojů: [1], [2], [4], [5], [7], [8], [9], [12], [13], [14], [15].

V této kapitole jsou nejdříve představeny základní metody řešení VRP. Ve druhé části kapitoly podrobněji představím vybrané metody, které budu dále používat v své práci.

Od té doby, co byla úloha VRP zformulována, lze zaznamenat značný vývoj v konstrukci řešení tohoto problému, a to jak v exaktních (přesných) nebo aproximativních (přibližných) algoritmech řešení.

2.1. Exaktní metody

Tyto metody jsou navrženy tak, aby postupně omezovaly množinu přípustných řešení ovšem, tak aby odstranily pouze řešení, která nejsou optimální. Tyto metody tedy hledají optimální řešení a patří mezi ně tyto metody:

- Přímé stromové prohledávání
- Dynamické programování
- Celočíselné lineární programování
- Algoritmus větví a mezí

O těchto metodách lze více nalézt zde [4].

Žádný známý exaktní algoritmus však není schopný v rozumném čase nalézt optimální řešení pro jakoukoliv úlohu zahrnující více než 50 zákazníků.

2.2. Heuristiky

Heuristiky patří mezi aproximativní metody a výsledky jimi dosažené jsou tedy určitou aproximací, uplatňují se hlavně v praxi.

Definice 2.1: Heuristika (heuristický algoritmus) je v informatice chápána jako postup získání řešení problému, které však není zcela přesné, ale poskytuje v krátkém čase dostatečné a dosti spolehlivé výsledky.

Mezi VRP heuristiky patří:

1) Konstruktivní heuristiky

Tyto heuristiky postupně jednoduchými kroky tvoří rozumné přípustné řešení. Tudíž se často stávají počátečním řešením pro další heuristiky, které jsou založeny na tom, že vycházejí z nějakého počátečního řešení, které zlepšují.

Mezi tyto metody patří například:

- Modifikovaná heuristika savings viz [8]
- Heuristika Clarka a Wrigtha (tzv. savings)

Heuristika Clarka a Wrighta je určena pro VRP s jedním centrálním skladištěm a je velmi známou koncepcí, jak se rychle a docela snadno dostat k přijatelným výsledkům. V této práci ji využívám, a proto je níže popsána podrobněji.

2) Dvoufázové heuristiky

V dvoufázových heuristikách se problém rozloží do dvou přirozených složek (fází). Jsou dva různé typy těchto metod:

- Nejprve jsou vrcholy seskupeny do shluků a teprve potom jsou podle těchto shluků konstruovány přípustné trasy.
- Na začátku je pomocí metod pro problém obchodního cestujícího zkonstruována jedna trasa procházející všemi vrcholy a ta je pak rozdělena do více tras podle toho, jak jsou vrcholy shlukovány.

3) Zlepšovací heuristické metody

Zlepšovací heuristické metody vycházejí z řešení zkonstruovaného jiným postupem a pokoušejí se ho zlepšit. Mohou například využívat heuristiky pro problém obchodního cestujícího nebo zákaznické výměny hran, případně míst, mezi různými trasami. Tyto metody jsou založeny na algoritmech lokálního prohledávání.

Definice 2.2: Algoritmy lokálního prohledávání začínají z počátečního přípustného řešení a postupně se iterační procedurou dostávají k novému lepšímu řešení prohledáváním okolí aktuálního řešení, dokud se nenajde další vylepšení, tj. až do té doby, kdy aktuální řešení je lokálním optimem s ohledem na aktuální strukturu okolí.

Definice 2.3: Okolí řešení je množina jiných řešení, které mohou být získány nějakou jednoduchou modifikací. Posunutí se z jednoho řešení k jinému nazveme ***přechod***, tedy v případě VRP nějak pozměníme aktuální, případně počáteční řešení, například zákaznickými výměnami.

- k – opt zlepšovací heuristiky

k – opt je nejrozšířenější heuristická metoda pro problém obchodního cestujícího, kde v každém kroku algoritmu k hran aktuální trasy je zaměněno za k jiných hran tak, aby bylo dosaženo zkrácení trasy. Tyto metody se u VRP používají jako vylepšení na každé aktuální trase. V praxi se většinou používají 2 – opt (viz Algoritmus 2.2) a 3 – opt heuristiky, které mají výpočetní náročnost $O(n^2)$ a $O(n^3)$.

Poznámka 2.1: Velké O v zápisu $O(\dots)$ se nazývá Landaův symbol a je používaný v komplexní teorii, počítačové vědě a matematice k popsání asymptotického chování funkcí. V podstatě nám říká, jak rychle funkce roste nebo klesá. Například při analýze nějakého algoritmu můžeme dojít k závěru, že čas potřebný k ukončení problému o velikosti n je dán takto $T(n) = 4n^2 - 2n + 2$. Potom, pokud budeme ignorovat konstanty pomaleji rostoucí ostatní členy, je možné říci, že „funkce $T(n)$ roste v souladu s n^2“a zapisuje se jako $T(n) = O(n^2)$.

- Multi – trasové zlepšovací heuristiky

Tyto zlepšovací algoritmy usilují o vylepšení známého přípustného řešení. Jedním ze způsobů vylepšení je provedení posloupnosti takzvaných λ – výměn.

Takzvané λ – výměny spočívají ve výměnách maximálně λ zákazníků nebo hran mezi dvěma trasami. Existuje více způsobů definování λ – výměn. Například takto jsou definovány v článku [1]:

Definice 2.4: $\boldsymbol{\lambda}$ ***– výměny*** jsou definovány jako přesun maximálně λ zákazníků mezi dvěma trasami. Hodnota λ je často stanovena na 1 nebo 2, aby se snížilo množství výměn a tím i výpočetní nároky algoritmu.

Takové výměny lze popsat dvojicemi (λ_1, λ_2), kde $\lambda_1 \leq \lambda$ a $\lambda_2 \leq \lambda$. λ_1 reprezentuje zákazníky přesunuté z trasy 1 do trasy 2 a λ_2 zákazníky přesunuté z trasy 2 do trasy 1. Tudíž, za předpokladu symetrické matice C, v případě 2 – výměny jsou možné tyto přesuny: $(2,2), (2,1), (2,0), (1,1), (1,0)$.

Alternativní definice pro λ – výměny je přes hrany, kdy vyměňujeme místo zákazníků hrany, viz Definice 3.1.

λ – výměny také určují strukturu okolí, ve kterém provádím prohledávání. Jedná se o rozšíření k – opt zlepšovací heuristiky na dvě vybrané trasy, proto mají λ – výměny podobnou výpočetní náročnost.

Provádíme pouze takové λ – výměny, pomocí kterých dochází ke zlepšení, tedy rozdíl mezi novou délkou vybraných dvou tras po provedení výměny a starou délkou těchto tras je menší než rozdíl získaný v předchozí iteraci. Pomocí této metody dosahujeme lokálního minima.

4) Metaheuristiky

Metaheuristiky byly navrženy pro VRP v posledních 15 letech a patří do široké rodiny heuristických metod, které prozkoumávají úplný prostor řešení a přechodně povolují zhoršení aktuálního řešení nebo povolují řešení neuskutečnitelné. Umožňují tak opustit lokální extrémy.

Gendreau, Laporte a Potvin v roce 2002 v článku [1] identifikovali 6 skupin metaheuristik pro VRP a to:

- Simulované žíhání
- Deterministické žíhání
- Tabu search (prohledávání)
- Genetické algoritmy
- Mravenčí systémy
- Nervové sítě

Úspěch každé určité metody závisí na její konkrétní implementaci, uvádí se, že pro VRP tabu search metody ostatní překonávají. Toto je podloženo rozsáhlými výpočetními experimenty prováděnými nezávisle v rámci několika výzkumů.

2.2.1. Heuristika Clarka a Wrighta (savings)

Tato heuristika je určena pro VRP s jedním centrálním skladištěm a je velmi známou koncepcí, jak se velmi rychle a docela snadno dostat k přijatelným výsledkům. Základní myšlenka je velmi jednoduchá. Uvažujeme depo v_0 a n vrcholů, tedy míst, do kterých vozíme zboží. Na začátku předpokládáme, že se řešení VRP skládá z použití n vozidel, kdy každé jedno vozidlo rozváží zboží do jednoho z n míst. Celková délka všech tras toho řešení je pak $2\sum_{i=1}^{n} c(v_0, v_i)$. Pokud však použijeme jedno vozidlo na obsloužení dvou míst v_i a v_j najednou, celková délka je pak snížena o takzvanou úsporu.

***Definice 2.5**: Úspora (savings)* $\boldsymbol{s(v_i, v_j)}$, která je výsledkem vložení míst v_i a v_j do jedné trasy, je definovaná takto:

$$\boldsymbol{s(v_i, v_j)} = 2c(v_0, v_i) + 2c(v_0, v_j) - \left[c(v_0, v_i) + c(v_i, v_j) + c(v_0, v_j)\right]$$
$$= \boldsymbol{c(v_0, v_i) + c(v_0, v_j) - c(v_i, v_j)}.$$

Čím je větší $s(v_i, v_j)$, tím více je žádané zkombinovat místa v_i a v_j do jedné trasy. Nicméně místa v_i a v_j nemohou být zkombinovány v jedné trase, jestliže nevyhovují některé z podmínek VRP, tedy v případě CVRP, kterým se v práci hlavně zabývám, nesmíme přesáhnout danou kapacitu vozidla.

Algoritmus 2.1: Heuristika Clarka a Wrigtha

1) Nejdříve vypočteme matici euklidovských vzdáleností C.
2) Vypočítáme úsporu $s(v_i, v_j)$ pomocí vzorce uvedeného v Definici 2.5 pro každou dvojici míst (v_i, v_j).
3) Seřadíme všechny úspory $s(v_i, v_j)$ sestupně. Tím vytvoříme tzv. „seznam úspor“. Proces úsporného algoritmu pak začíná od první úspory v seznamu, tedy od té největší.
4) Pro konkrétní úsporu $s(v_i, v_j)$ zahrneme spojení neboli hranu (v_i, v_j) do jedné trasy, pokud nebudou porušeny nějaké podmínky pro VRP, a pokud:
   a) *Ani* v_i ani v_j nejsou ještě přiděleny do trasy, v tomto případě je vytvořena nová trasa zahrnující v_i i v_j.

b) *Právě* jedno z míst v_i a v_j už bylo zahrnuto do existující trasy a zároveň takové místo není takzvané vnitřní v této trase, tj. není spojené hranou s depem, v tomto případě hranu (v_i, v_j) přidáme do té samé trasy.

c) *Obě* místa v_i a v_j už byly zahrnuty do dvou různých existujících tras a žádné z nich není vnitřní ve své trase, v tomto případě jsou tyto trasy spojeny do jedné.

5) Pokud nebyl seznam úspor vyčerpán, vrátíme se na krok 3 a prověřujeme další úsporu v sestupném pořadí. Pokud je seznam vyčerpán, zastavíme algoritmus a řešení VRP se skládá z tras vzniklých v kroku 3. Jestliže zůstane nějaké místo bez toho, aby bylo zařazeno do trasy během kroku 3, musí být zahrnuto do trasy, která začíná v depu, pak pokračuje přímo do nezařazeného místa, a odtud se vrací zase do depa.

Poznámka 2.2: Existují dvě varianty tohoto algoritmu. Pokud chceme minimalizovat součet délek všech tras, uvažujeme ze seznamu úspor pouze kladné hodnoty. Jestliže chceme minimalizovat počet vozidel, vybíráme ze seznamu úspor i záporné hodnoty prodlužující součet délek všech tras.

Poznámka 2.3: Euklidovská vzdálenost bodů (vrcholů) v_i a v_j se souřadnicemi (i_1, i_2) a (j_1, j_2) je číslo

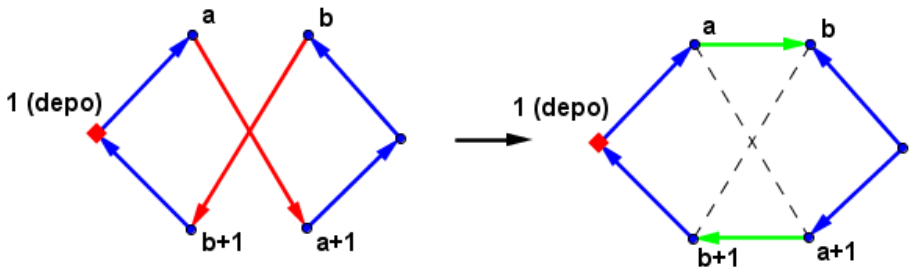
$$c(v_i, v_j) = \sqrt{(i_1 - j_1)^2 + (i_2 - j_2)^2}.$$

2.2.2. Zlepšovací heuristika 2 – opt

Jedná se o jednoduchou a rychlou zlepšovací heuristiku pro problém obchodního cestujícího. Pokud však uvažujeme VRP tak, vlastně každá trasa představuje takovýto problém.

Algoritmus 2.2: 2 – opt zlepšovací heuristika

2 – opt algoritmus v podstatě odstraňuje dvě hrany z cesty, čímž vzniknou dvě různé části cest a tyto části opět spojuje do nové cesty. Zkoumáme, zda nahrazením hran $(a, a+1)$ a $(b, b+1)$ hranami (a, b) a $(a+1, b+1)$ nedojde ke zkrácení délky trasy a opakujeme to, tak dlouho, dokud takové hrany existují.



Obrázek 2 – 2-opt heuristika

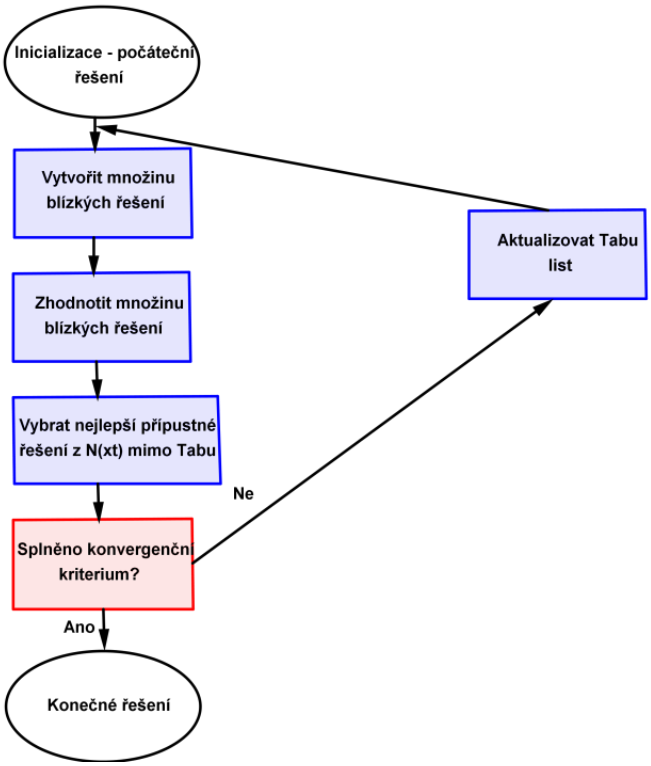
3. Tabu search (Tabu prohledávání)

V této kapitole bylo využito těchto zdrojů: [1], [2], [5], [6].

3.1. Popis a historie

Jednou z největších složek úspěchu metaheuristik, a obzvláště tabu search metod, je jednoduchost jejich základní struktury. Tyto metody rozšiřují a vylepšují takzvané techniky lokálního prohledávání, které byly velmi rozšířeny na začátku 60. let 19. století. Protože kvalita nalezeného lokálního optima nemusí být moc dobrá z hlediska optimálního řešení, používáme specifické algoritmy tabu search (ale také další metaheuristiky založené na lokálním prohledávání). Abychom se dostali z lokálního optima a mohli pokračovat v prohledávání směrem k možnému lepšímu řešení, přidáváme tzv. tabu hrany, tedy zakázané hrany, zabraňující návratu do předchozích aktuálních řešení. Tabu search algoritmy řešení byly navrženy v roce 1986 profesorem F. Gloverem a jsou jedněmi z nejrozšířenějších a nejefektivnějších heuristik dostupných pro řešení kombinatorických optimalizačních problémů.

Tabu search metody prozkoumávají prostor řešení přechodem z řešení x_t dosaženého v iteraci t k nejlepšímu řešení x_{t+1} v podmnožině prstencového okolí $N(x_t)$ řešení x_t. Na rozdíl od zlepšovacích heuristik tabu search metody připouštějí dočasné zhoršení řešení v následujících iteracích proto, abychom se mohli dostat z lokálního optima, pokud bylo dosaženo. Nicméně k vyhnutí se zacyklení, které může být způsobené návratem do lokálního optima v další iteraci, používáme tzv. mechanismus krátkodobé paměti, tedy tzv. tabu list, kde uchováváme zakázané hrany z přechozích iterací, viz kapitola 3.2.2. Běžně je tabu search algoritmus zastaven po daném počtu iterací celkových nebo těch, které už neznamenají zlepšení. Obecné schéma těchto metod je zobrazeno v diagramu na obrázku 3.



Obrázek 3 – diagram obecného tabu search algoritmu

3.2. Hlavní vlastnosti tabu search algoritmů

3.2.1. Struktura okolí

Vlastnost struktura okolí má velký vliv na rychlost výpočtu a kvalitu nalezeného řešení. Existují dva hlavní způsoby, jak definovat prstencové okolí $N(x_t)$ v tabu search algoritmu pro VRP, a to λ – výměny a ejection chains.

3.2.1.1. λ – výměny

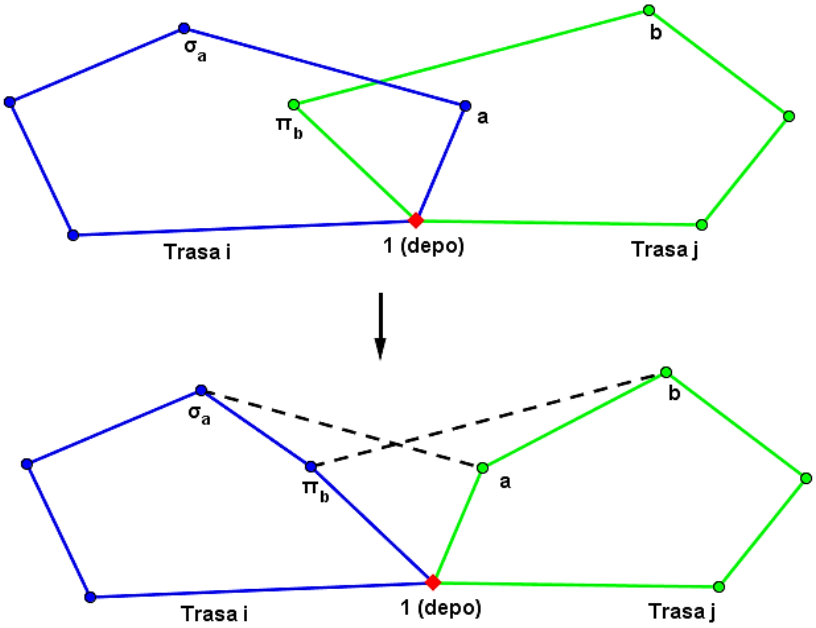
U tabu search algoritmů lze využívat stejné λ – výměny jako u zlepšovacích heuristik, zde ale vybíráme nejlepší výměnu v okolí, která není tabu, i kdyby zhoršila řešení. Nyní si představíme konkrétní příklady λ – výměn. Kromě slovního popisu přidám i příklady výměn mezi dvěma trasami, trasy jsou v obrázku označeny modrou a zelenou barvou. Tyto výměny následně využívám v GTS algoritmu v kapitole 4 a 5. Zde je využita alternativní definice λ – výměn z článku o GTS algoritmu [5].

Definice 3.1: λ – výměny jako struktura okolí je tvořena všemi přechody získanými pomocí škrtnutí λ hran použitých v aktuálním řešení a jejich nahrazení jinými λ hranami, které definují nové řešení.

V seznamu níže jsou λ – výměny pojmenovány podle Definice 3.1. V následujících obrázcích různých výměn je využito toto označení:

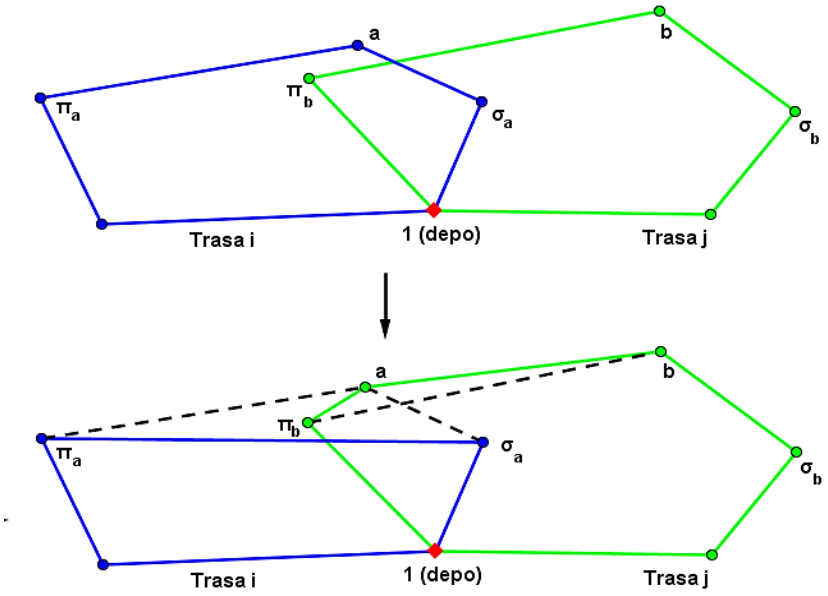
- a … index jednoho z vrcholu z hrany (a, b) přidané po provedení výměny
- b … index jednoho z vrcholu z hrany (a, b) přidané po provedení výměny
- π_i … index vrcholu předcházející daný vrchol $i \in V$
- σ_i … index vrcholu následující po daném vrcholu $i \in V$
- **- - -** … čárkové hrany představují škrtnuté hrany po provedení výměny

1) **2 – *výměna*** vytvořená ze všech získaných přechodů, které získáme zrušením dvou hran z různých dvou tras používaných v aktuálním řešení a nahrazením jinými dvěma hranami, které definují nové řešení. Tato výměna je podle Definice 2.4 typu (1,1), tj. prohodí si místo vždy jeden vrchol z každé trasy. Počet vrcholů v trasách účastnících se této výměny je nejméně tři (včetně depa na začátku a na konci trasy), tedy můžeme brát v úvahu všechny trasy. Aby výměna byla uskutečnitelná vrchol a smí probíhat počínaje depem přes všechny vrcholy až do předposledního vrcholu v jedné ze dvou tras účastnících se výměny. K tomu bude vrchol b probíhat od druhého vrcholu následujícího hned po depu, přes všechny vrcholy až do posledního vrcholu druhé trasy účastnící se výměny. Tento případ je symetrický a tedy, když prohodíme zelenou a modrou trasu z obrázku, výsledky se nezmění.



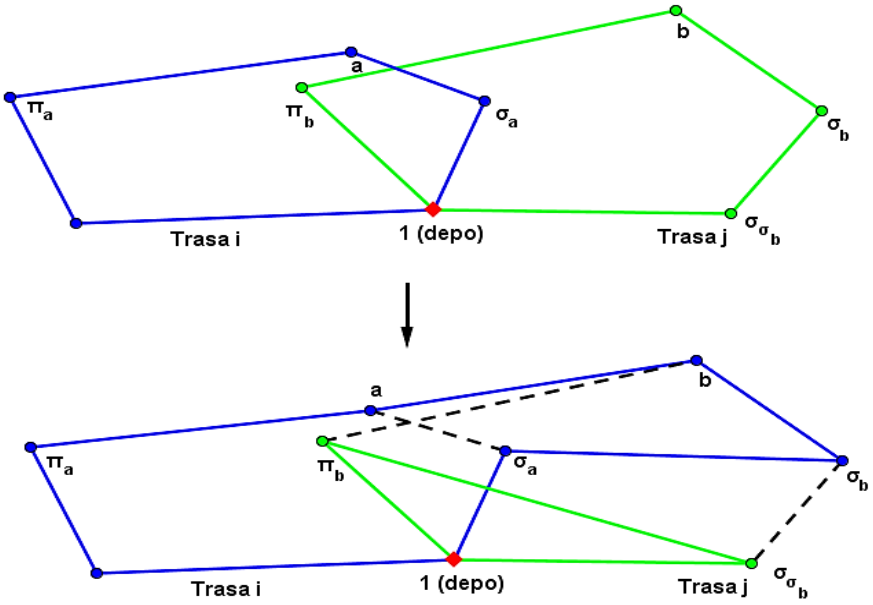
Obrázek 4 – 2 - výměna

2) **3 – *výměna*** spočívá ve zrušení tří hran používaných v aktuálním řešení a nahrazení jinými třemi hranami, které definují nové řešení. Tento případ oproti 2 – výměny není symetrický. Výměnu lze uskutečnit různě, my si tady ukážeme dva případy:
   a) V tomto případě vezmeme vrchol a z modré trasy a vložíme ho do zelené trasy s tím, že vrchol a je z modré trasy odstraněn, tj. podle Definice 2.4 je to výměna $(1,0)$. Pro počty vrcholů v trasách účastnících se výměny platí to samé jako u 2 – výměny. Vrchol a bude probíhat od dvou, vrchol následující hned po depu, přes všechny vrcholy až do předposledního vrcholu modré trasy a vrchol b potom od dvou přes všechny vrcholy až do posledního vrcholu zelené trasy účastnící se výměny.



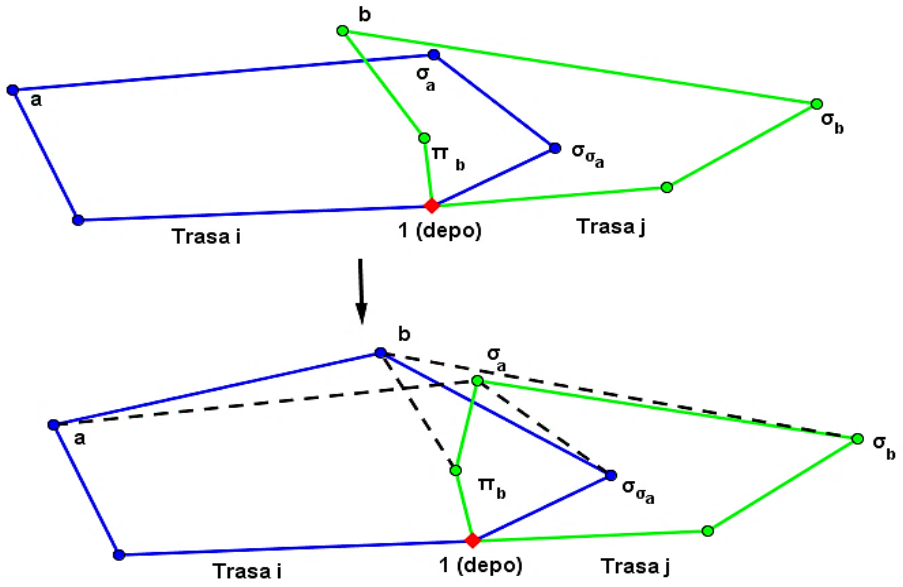
Obrázek 5 – 3 - výměna a)

b) Tímto způsobem se dvojice vrcholů spojená hranou (b, σ_b) přehodí i s touto hranou ze zelené do modré trasy. Tedy podle Definice 2.4 je to výměna $(0,2)$. Modrá trasa by měla obsahovat nejméně tři vrcholy, tedy vlastně můžeme použít všechny trasy, zelená trasa však musí obsahovat čtyři a více vrcholů. Vrchol a bude probíhat počínaje depem přes všechny vrcholy až do předposledního vrcholu, a vrchol b pak od dvou přes všechny vrcholy až do předpředposledního vrcholu zelené trasy účastnící se výměny.



Obrázek 6 – 3 - výměna b)

3) **4** – ***výměna*** vytvořená zrušením čtyř hran používaných v aktuálním řešení a nahrazením jinými čtyřmi hranami, které definují nové řešení. Tato výměna je podle Definice 2.4 výměna typu (1,1), tedy prohodíme jeden vrchol ze zelené trasy s jedním vrcholem z modré trasy. V tomto případě stačí, aby počty vrcholů v trasách účastnících se výměny byly rovny tři, což znamená, že můžeme použít všechny trasy. Pokud však v obou trasách účastnících se výměny je počet vrcholů roven tři, pak se jen tyto trasy prohodí a nic se nemění, proto takové výměny nepovolujeme. Vrchol a bude probíhat počínaje depem přes všechny vrcholy až do předpředposledního vrcholu modré trasy účastnící se výměny a vrchol b pak bude probíhat od druhého vrcholu následujícího po depu přes všechny vrcholy až do předposledního vrcholu. Podobně jako u 2 – výměny je tento případ také symetrický.



Obrázek 7 – 4 - výměna

Všechny možnosti výměn popsané výše jsou uskutečnitelné, ale v některých případech se může stát, že se vymění hrany a nic se nezmění, trasy se ani nezkrátí ani neprodlouží. V praktické implementaci přiložené na CD je to ošetřeno tak, aby nulový rozdíl nových délek tras účastnících se výměny a starých délek tras účastnících se výměny nebyl brán v úvahu.

3.2.1.2. Ejection chains (stěhování řetězců)

Další možností, jak definovat strukturu okolí jsou ejection chains, kde se změny dějí typicky na více než dvou trasách zároveň. Tento způsob se skládá z identifikace množiny tras a dále z přesunu zákazníků z trasy 1 do trasy 2, z trasy 2 do trasy 3, atd. Tudíž, zobecňuje λ – výměny mezi více než dvěma trasami. Tento způsob v této práci však využíván není.

3.2.1.3. Přípustné řešení

Další záležitost související se strukturami okolí je to, že přechodně povolíme během prohledávání nepřípustné (nemožné) řešení. V algoritmu tedy povolíme například posloupnosti typu (x_t, x_{t+1}, x_{t+2}), kde x_t a x_{t+2} jsou přípustné, ale x_{t+1} není, a x_{t+2} je lepší než x_t, jsou v algoritmu dovoleny. Jeden způsob, jak se vypořádat s nepřípustností řešení, je pomocí penalizované účelové funkce typu:

$$c'(x) = c(x) + \alpha P(x) + \beta D(x),$$

kde $c(x)$ jsou náklady na trasy pro řešení x, $P(x)$ představuje celkové překročení poptávky zákazníků, $D(x)$ celkové překročení délky všech tras a α, β jsou samoregulační parametry. Na začátku prohledávaní jsou tyto parametry určeny jako 1 a dále jsou periodicky měněny, buďto sníženy nebo zvýšeny, pokud posledních μ řešení bylo přípustných (případně nepřípustných), kde μ je uživatelem určený parametr. Tento postup je vhodný spíše pro okolí s jednoduchou strukturou, v případě složitějších se příliš nepoužívá například v případě ejection chains, protože prohledávací proces je schopen se dostat z řešení x_t k řešení x_{t+2} přímo bez toho, abychom museli jít přes nepřípustné řešení x_{t+1}.

3.2.2. Krátkodobá paměť

Krátkodobá paměť zabraňuje zacyklení prostřednictvím vytvoření tabu listu, což znamená, že některé atributy (například některé hrany) posledně prozkoumaných řešení jsou přechodně zakázány k použití v dalších několika iteracích neboli označeny jako tabu.

Definice 3.2: Tabu list je tedy paměťový seznam, který uchovává některé atributy identifikující přechody, jež vytvořily řešení v posledních krocích.

Během prohledávání aktuálního okolí všechny přechody, které mají atributy stejné jako ty, co jsou v tabu listu, jsou považovány za tabu a tedy vyřazeny. Přechod je považován za tabu, tedy je zakázán k provedení, pokud byla hrana tohoto přechodu smazána v jednom z předchozích přechodů. To znamená, že hrany vymazané v přechodu nemohou být použity v následném přechodu v další iteraci. Tyto hrany pak zůstávají tabu po několik dalších iterací, toto nazýváme jako tabu držení označené T a pro každý uskutečněný přechod je určeno jako náhodná celočíselná hodnota z rovnoměrně rozloženého intervalu $\langle T_{min}, T_{max} \rangle$. Čím větší je hodnota T, tím déle zůstávají přechody tabu. Tabu list tedy zabraňuje návratu do předchozího řešení, které může být lokálním minimem, a proto se může stát, že aktuální řešení na nějakou dobu zhoršíme například z hlediska celkové délky tras, abychom neuvízli v lokálním minimu.

3.2.3. Dlouhodobá paměť

Dlouhodobá paměť poskytuje další informaci o tom, kolikrát byl tabu element přechodu (hrana) uložen do tabu listu a slouží k diverzifikaci řešení.

3.2.3.1. Diverzifikace

Některé tabu search algoritmy penalizují často prováděné přechody, a to přidáním penalizačního členu k nákladům na trasy $c(x_{t+1})$ řešení x_{t+1}. Penalizační člen je součinem kombinace tří faktorů:

i. faktor měřící předchozí frekvenci;
ii. faktor velikosti konkrétního problému (např. $\sqrt{n}$);
iii. uživatelem kontrolovaný škálovací faktor, umožňující mu řídit velikost vlivu penalizačního členu.

Diverzifikace zajišťuje, že prohledávací proces není omezen na limitovanou část prostoru řešení. V praxi je tento mechanismus efektivní a výpočetně nenáročný.

3.2.4. Zesilování neboli zintenzivňování (Intenzifikace)

Zintenzivňování klade důraz na prohledávání v blízkosti kvalitních známých řešení. Algoritmus znovu spustíme v bodě, kde předchozí iterace docílila nejlepšího řešení.

Jeden ze způsobů zesilování může být periodické zlepšování tras ve VRP použitím algoritmů pro problém obchodního cestujícího.

Jiný efektivní zesilovací nástroj pro tabu search je takzvaná adaptivní paměť, která je populací dobrých řešení dosažených během procedury prohledávání. Základní myšlenka pro konstrukci nového řešení je podobně jako v genetických algoritmech založena na kombinaci jednotlivých prvků těchto dobrých řešení. V adaptivní paměti pak zůstávají řešení s lepšími výsledky.

3.3. Přehled nejpoužívanějších Tabu search metod

3.3.1. Osmanův algoritmus

V roce 1991 Osman navrhl koncept takzvaných λ-výměn mezi trasami, viz konkrétní příklady λ – výměn v kapitole 3.3.1. Ve svých implementacích používá $\lambda = 2$, tudíž povoluje kombinaci přesunů jednoho nebo dvou vrcholů v rámci jedné trasy nebo mezi více trasami. Osman testoval dvě strategie pro výběr struktury okolí, ve kterém hledáme řešení. První strategie nazvaná nejlepší přípustná vybírá nejlepší řešení, jehož části nejsou v tabu listu. Druhá strategie zvaná první nejlepší přípustná vybírá první přípustné řešení, které představuje zlepšení, pokud takové řešení existuje, jinak je nejlepší přípustné řešení zachováno. Pomocí empirického testování Osman zjistil, že se zvoleným stejným konvergenčním kritériem druhá zmíněná strategie konstruuje o něco lepší řešení než první strategie, avšak je o hodně pomalejší. Osmanův tabu search algoritmus používá fixní dobu držení kroků v tabu listu, nepoužívá mechanismus dlouhodobé paměti a mechanismus zintenzivňování.

3.3.2. Gendreau, Hertz a Laporte – Taburoute algorimus

V tomto algoritmu z roku 1994 jsou aktuální řešení v konkrétním okolí získávána přemístěním vrcholu z aktuální trasy r do jiné trasy s obsahující jednoho z nejbližších sousedů přemisťovaného vrcholu. Vložení vrcholu do trasy s je provedeno souběžně s lokální reoptimalizací užitím Geni procedury pro problém obchodního cestujícího, viz článek [2]. Toto může mít za následek odstranění trasy nebo vytvoření nové trasy. Abychom omezili velikost okolí, náhodně vybíráme podmnožinu vrcholů uvažovaných pro znovu vložení do jiných tras. V tomto algoritmu byl představen koncept penalizované účelové funkce $c'(x)$, který jsme si popsali v odstavci 3.2.1.3 Parametry α a β jsou nastavované každých 10 iterací: jestliže 10 předchozích řešení bylo přípustných s ohledem na kapacitu (případně délku trasy), pak α, β jsou děleny dvěma; jestliže byly všechny

nepřípustné, pak α, β jsou násobeny dvěma; jinak α, β zůstávají neměnné. Jakémukoli vrcholu odstraněnému z trasy r v iteraci t je zabráněno znovu vložení do této trasy až do iterace $t + \theta$, kde θ je náhodně vybráno z intervalu $\langle 5{,}10 \rangle$, pokud by však takové přemístění neposkytlo nové současné řešení. Trasy jsou periodicky aktualizovány užitím US post-optimalizace pro problém obchodního cestujícího, viz článek [2]. Jsou zde používány mechanismy zintenzivňování, ale i diverzifikace.

3.3.3. Taillardův algoritmus

Algoritmus byl publikován v roce 1993, ale navrhnut byl ve stejném období jako taburoute. Používá λ-výměn s $\lambda = 1$ bez lokální reoptimalizace a bez povolení nepřípustnosti. Toto schéma je mnohem jednodušší než taburoute a provádí mnohem více iterací za stejný čas. Tabu a dlouhodobý paměťový mechanismus jsou zde stejné jako u taburoute. Trasy jsou periodicky reoptimalizovány použitím přesných algoritmů pro problém obchodního cestujícího. Procedura prohledávání využívá dekompozičního schématu, které vede k uplatnění paralelního výpočtu. V rovinných problémech je množina zákazníků rozdělena na oblasti, a to na oblasti soustředěné okolo depa a také na oblasti složené ze soustředných kruhů. Prohledávání je uskutečňováno v každé oblasti různými procesory. Hranice těchto oblastí jsou periodicky měněny, aby bylo dosaženo diverzifikačního efektu. V ostatních problémech jsou oblasti definovány přes výpočet nejkratšího rozpětí stromu trasy začínající v depu. Tento algoritmus je stále jedním z nejefektivnějších tabu search algoritmů pro VRP.

3.3.4. Xu a Kelly algoritmus

Xu a Kelly ve svém tabu search algoritmu z roku 1996 definují okolí pro prohledávání pomocí přepínání mezi ejection chains a výměnami vrcholů, viz kapitola 3.2.1.1. Ejection chains hledá řešení pomocné úlohy minimalizující toky v sítích. Jednotlivé trasy jsou periodicky reoptimalizovány ve smyslu 2 – opt operace viz kapitola 2.2.2. Průběžná nepřípustná řešení s ohledem na kapacitu jsou během prohledávání povolena, jsou zde použita fixní tabu držení T a mechanismus dlouhodobé paměti. Algoritmus používá paměť nejlepších známých řešení. Tento algoritmus vyprodukoval několik dobrých řešení pro CVRP, ale je docela komplikovaný a vyžaduje nastavení několika parametrů.

3.3.5. Granulární tabu search algoritmus

Byl navržen v roce 1998 Tothem a Vigem. Granulární koncept tohoto algoritmu není aplikován pouze na VRP nebo tabu search algoritmus, ale i na různé diskrétní optimalizační problémy obecně. Hlavní myšlenkou je nebrat v úvahu „dlouhé“ hrany pro řešení, protože tyto hrany mají velmi malou šanci patřit k optimálnímu řešení. V této práci se zabývám touto metodou nejpodrobněji a to v kapitole 4. A také implementace v Matlabu je zaměřena na tuto metodu a to z toho důvodu, že za relativně krátký čas dokáže dosáhnout velmi dobrých výsledků.

4. Granulární (granular) tabu search (GTS) metoda

V této kapitole byly využity tyto zdroje: [5], [8].

Tato metoda patří mezi konstruktivní heuristiky a je založena na použití drasticky omezené struktury okolí, která neobsahuje hrany, u kterých je nepravděpodobné, že by patřily k dobrým možným řešením. Tato omezená struktura okolí se nazývá granulární (zrnitá) a je vnímána jako efektivní implementace strategií založených na tabu listu. Tato koncepce může být použita na širokou škálu kombinatorických optimalizačních problémů například problém obchodního cestujícího, rozvozní problém a další podobné síťové problémy, ve kterých celkové náklady na řešení jsou definované jako součet nákladů vybraných prvků. Jeden z důvodů, proč různé metody typu tabu search pro VRP vyžadují velmi dlouhý výpočetní čas je ten, že tyto metody běžně potřebují provést i několik tisíc iterací k obdržení vysoce kvalitních řešení. Každá iterace se skládá z prohledávání zmiňovaných λ – výměn určujících strukturu okolí a vyžaduje tím pádem nejméně $O(n^2)$ času.

Byly navrženy různé způsoby, jak zkrátit výpočetní čas prohledávání okolí. V této metodě je toho dosaženo použitím granulární struktury okolí, která může být prohledávána za mnohem kratší čas než ty tradičně používané s tím, že to ovšem příliš neovlivní kvalitu nalezeného řešení. Taková struktura okolí zajišťuje vyřazení velkého množství málo pravděpodobných přechodů a skutečné prohledávání je provedeno pouze pro malou podmnožinu přechodů, která obsahuje ty nejvíce pravděpodobné hrany.

4.1. Jednoduchá granulární struktura okolí pro VRP

Rozvozní problém je definován na úplném grafu. Je ovšem zřejmé, že „dlouhé“ hrany mají menší pravděpodobnost stát se součástí vysoce kvalitního řešení. Tudíž možný způsob, jak zrychlit průběh výpočtu je prohledávání omezeného okolí se snahou vyřadit přechody, které by vkládaly pouze „dlouhé“ hrany do aktuálního řešení.

Definice 4.1: Vycházíme z původního úplného grafu $G = (V, H)$ a z něho vytvoříme nový takzvaný ***řídký graf*** $G' = (V, H')$, který zahrnuje jen některé a to všechny „krátké“ hrany společně se všemi hranami spojenými s depem a s těmi, co patří k nejlepšímu doposud nalezenému řešení.

Prohledávání granulární struktury okolí znamená, že jsou generovány pouze přechody zahrnující aspoň jednu „dobrou“ hranu z H'. To však nemá za následek, že „dlouhé“ hrany nejsou nikdy vloženy do aktuálního řešení, ale přechody obsahující jen „dlouhé“ hrany nebereme v úvahu. Používá se jednoduché filtrující pravidlo pro definování „krátkých“ hran v grafu G.

Definice 4.2: Hrana je ***krátká,*** tudíž patří do grafu G', pokud náklad na tuto hranu není větší než hodnota tzv. ***granulární meze***

$$\vartheta = \beta \cdot \frac{x'}{n + K},$$

kde β je vhodný kladný parametr řídkosti; x' je hodnota řešení (celková délka všech tras) získaná nějakou rychlou tradiční heuristikou, například heuristikou Clarka a Wrighta; n je počet vrcholů tedy míst, do kterých rozvážíme zboží a K představuje počet tras vytvořených pomocí úsporné heuristiky.

V některých konkrétních případech může být použito i jiné filtrující pravidlo například, pokud máme v grafu vrcholy seskupené do shluků.

4.2. Efektivní prohledávání a diverzifikace

Máme tedy množinu hran H' obsahující všechny „krátké“ hrany menší než granulární mez a dále množinu I skládající se z dalších důležitých hran jako jsou hrany do depa a z depa nebo hrany patřící k nejlepšímu dosavadnímu řešení. Pak

$$H' = \{(v_i, v_j) \in H : c_{ij} \leq \vartheta\} \cup I.$$

Hrany z H' jsou pak přímo používány jako generátory přechodů a ke stanovení dalších hran zahrnutých do konkrétního přechodu původního okolí. Když je hrana (a, b) z λ – výměn vybrána do přechodů, všechny ostatní hrany zahrnuté v přechodu jsou implicitně definovány. Tudíž užitím nějaké efektivní datové struktury k uchování řídkého grafu celkový počet přechodů vytvořených v prohledávání granulárního okolí je $O(|H'|)$, kde $|H'| << n^2$.

Řídký graf spolu s granulárním okolím poskytuje jednoduchý způsob, jak zařadit do GTS další možné strategie, které vylepšují algoritmus, jako jsou zintenzivňování nebo diverzifikace. Tyto strategie lze zavést pomocí modifikace parametru β, který ovlivňuje

počet hran, aktuálně začleněných v řídkém grafu, změněný. GTS algoritmus se může měnit mezi dlouhými zintenzivňovacími kroky, spojenými s malými hodnotami β, a krátkými diverzifikačními kroky, ve kterých je β podstatně zvyšováno. Jestliže se aktuální nejlepší řešení nezlepšuje po n_b iterací, potom je parametr řídkosti β zvětšen na β_d, je zkonstruován nový řídký graf a tento řídký graf s tímto parametrem je využit po n_d iterací. Po těchto iteracích je však hodnota parametru znovu změněna na původní.

4.3. Algoritmus GTS metody

V tomto algoritmu je jako počáteční řešení využita heuristika Clarka a Wrighta, dále jsou zde využity λ – výměny pro tabu search metody popsané kapitole 3.2.1.1. Využíváme zde efektivní strategii tabu listu z kapitoly 3.2.2.

Algoritmus 4.1: Granulární tabu search pro CVRP

1) Nejdříve vypočteme matici euklidovských vzdáleností C.
2) Inicializace počátečního řešení – Savings algoritmus z kapitoly 2.2.1., ale také může být využita i jiná konstruktivní heuristika.
3) Provádění λ – výměn, které určují základní strukturu okolí, viz konkrétní příklady λ – výměn v kapitole 3.2.1.1. s tím, že tyto výměny mohou aktuální řešení na nějakou dobu i zhoršit. Pokud pomocí výměn nenajdeme žádnou změnu, zvětšíme několikanásobně β a prohledáváme větší okolí, abychom nějakou změnu našli a dostali se z lokálního minima.
4) Určení granulární struktury okolí, která je získaná z hran v řídkém grafu definovaném v Definici 4.1. Jako vstupní parametr musíme určit parametr řídkosti grafu. Po každé iteraci jsou nově vložené hrany aktuálního řešení přidány do řídkého grafu.
5) Tvorba a průběžná aktualizace tabu listu, kde jsou uchovávány hrany vyškrtnuté v provedených λ – výměnách. Zde musíme na vstupu určit horní a dolní mez intervalu $\langle T_{min}, T_{max} \rangle$, ze kterého náhodně vybíráme celé číslo pro tabu držení T.
6) Na začátku uvažujeme jako nejlepší řešení počáteční řešení z kroku 2 a také ho v první iteraci bereme jako aktuální řešení, které se mění v každé iteraci. Pokud se aktuální řešení zlepší natolik, že tedy celková délka tras aktuálního řešení bude menší než celková délka tras nejlepšího řešení, pak si toto řešení uložíme a změníme jej na nejlepší řešení.

7) Algoritmus se poté vrací ke kroku 3. Ukončen je po daném celkovém počtu iterací, který lze zadat jako vstupní parametr.

V každé iteraci se tedy oproti zlepšovacímu heuristickému algoritmu, nemusí aktuální řešení zlepšovat, ale může se nějakou dobu i zhoršovat. Tento algoritmus může být vylepšen například pomocí 2 – opt zlepšovací heuristiky nebo nějakou zlepšovací heuristikou metodou.

5. Implementace v Matlabu

V této kapitole jsem využila tyto zdroje: [5], [7], [8], [12], [15], nápověda v Matlabu.

Na přiloženém CD jsou mnou vytvořené funkce v softwarovém prostředí Matlab pro řešení CVRP pomocí úsporné heuristiky, zlepšovacího algoritmu, GTS algoritmu a navíc jsou tam také funkce pro vykreslení grafu výsledného řešení, ale také pro průběžné vykreslování aktuálních řešení.

Ke každé uvedené funkci je vytvořena nápověda v Matlabu, kterou lze otevřít pomocí příkazu help, například help granular_tabu_search.

5.1. Seznam funkcí

5.1.1. Základní funkce

Základní důležité funkce volané uživatelem pro zobrazení výsledku daného datového souboru. Funkce jsou uvedeny v pořadí, v jakém jsou postupně spouštěny pro konkrétní datové soubory.

1) **`nacti_vrp`**

Funkce je určena k otevření a načtení souborů s příponou vrp a je převzata od vedoucího práce s menší modifikací.

Vstupní proměnné:

`cesta` … do této vstupní proměnné vložíme `'A-VRP/'` tedy název složky souborů s příponou vrp uloženou v adresáři, ze kterého spouštíme všechny M soubory v Matlabu

`nazev` …do toho vstupu vložíme `'A-n16-k8.vrp'` tedy název konkrétního souboru s příponou vrp ze složky

Výstupní proměnné:

`n` … celkový počet míst

`kapacita` … kapacita vozidla

`poptavka` … matice s množstvím zboží požadovaným místy v druhém řádku plus

řádek s očíslováním míst

`mista` … matice souřadnic zákaznických míst se třemi sloupci, v prvním sloupci očíslování míst a ve druhém a třetím sloupci souřadnice míst

`mista_souradnice` … matice pouze souřadnic zákaznických míst bez sloupce očíslování míst

`opt_hodnota` … optimální hodnota, která je v datovém souboru s příponou vrp označena jako Best value nebo Optimal value

Volání této funkce:

```
[n,kapacita,poptávka,mista,mista_souradnice,opt_hodnota]=nacti_vrp
(cesta,nazev)
```

2) **pdist2**

Funkce vytvoří matici euklidovských vzdáleností C mezi zákaznickými místy. Tato funkce je v Matlabu předdefinovaná.

Vstupní proměnné:

`mista_souradnice` … matice pouze souřadnic míst bez sloupce očíslování míst

Výstupní proměnné:

`vzdalenosti` … matice euklidovských vzdáleností C mezi jednotlivými místy

Volání této funkce:

```
vzdalenosti=pdist2(mista_souradnice,mista_souradnice)
```

3) **nahodne**

Funkce je určena ke konstrukci tras, které jsou vytvořeny náhodně tak, aby nepřesáhla žádná trasa kapacitu vozidla.

Vstupní proměnné:

`n` ... celkový počet míst

`vzdalenosti` ... matice euklidovských vzdáleností C mezi jednotlivými místy

`poptavka` ... matice s množstvím zboží požadovaným místy v druhém řádku plus

řádek s očíslováním míst

`kapacita` ... kapacita vozidla

Výstupní proměnné:

`cesty` ... vytvořena z pole buněk, kde jsou ve sloupci uloženy trasy ve formě vektoru z depa (=1) do příslušných měst a zpátky do depa, po provedení funkce `nahodne`, lze použít jako počáteční řešení pro GTS algoritmus

`pocet_tras` ... počet všech vytvořených tras (vektorů) funkcí `nahodne` v sloupci pole cesty (číslo)

`naklady_c` ... vektor nákladů na jednotlivé trasy

`delka_tras` ... vektor délek jednotlivých tras

`celkova_delka_tras` ... celková délka všech tras (číslo)

`okruh` ... vektor počtu míst v jednotlivých trasách

Volání této funkce:

```
[cesty,pocet_tras,naklady_c,delka_tras,celkova_delka_tras,okruh=…
nahodne(n,vzdalenosti,poptavka,kapacita)
```

4) **`savings`**

Funkce vytváří počáteční řešení pro GTS, tedy heuristickým algoritmem Clarka a Wrighta vytváří jednotlivé trasy, viz kapitola 2.2.1.

Vstupní proměnné:

`n` ... celkový počet míst

`vzdalenosti` ... matice euklidovských vzdáleností C mezi jednotlivými místy

`poptavka` ... matice s množstvím zboží požadovaným místy v druhém řádku plus řádek s očíslováním míst

`kapacita`...kapacita vozidla

Výstupní proměnné:

`cesty` ... vytvořena z pole buněk, kde jsou ve sloupci uloženy trasy ve formě vektoru z depa (=1) do příslušných míst a zpátky do depa, vytvořené pomocí algoritmu Savings

`pocet_tras` ... počet všech vytvořených tras (vektorů) funkcí nahodne v sloupci pole cesty (číslo)

`naklady_c` ... vektor nákladů na jednotlivé trasy

`delka_tras` ... vektor délek jednotlivých tras

`celkova_delka_tras` ... celková délka všech tras (číslo)

`okruh` ... vektor počtu míst v jednotlivých trasách

Volání této funkce:

```
[cesty,pocet_tras,naklady_c,delka_tras,celkova_delka_tras,okruh]=…
savings(n,vzdalenosti,poptavka,kapacita)
```

5) **`zlepsovaci`**

Funkce vytváří jednotlivé trasy do proměnné cesty použitím zlepšovacího algoritmu.

Vstupní proměnné:

`c` .... pomocné pole buněk pro jednotlivé trasy z počátečního řešení

`n` ... celkový počet míst

`vzdalenosti` ... matice euklidovských vzdáleností C mezi jednotlivými místy

`poptavka` ... matice s množstvím zboží požadovaným místy v druhém řádku plus řádek s očíslováním míst

`kapacita` ... kapacita vozidla

`tol` ... tolerance pro nulový rozdíl počítaný v tomto algoritmu

Výstupní proměnné:

`vylepsene_cesty` ... zlepšené trasy po provedení zlepšovacího algoritmu

`delka_tras` ... vektor zlepšených délek tras po provedení zlepšovacího algoritmu

`celkova_delka_tras` ... zlepšená celková délka tras po provedení zlepšovacího algoritmu (číslo)

Volání této funkce:

```
[vylepsene_cesty,delka_tras,celkova_delka_tras]=zlepsovaci(cesty,…
n,vzdalenosti,poptavka,kapacita,tol)
```

6) **granular_tabu_search**

Funkce je určena k provedení GTS algoritmu, viz kapitola 4.3.

Vstupní proměnné:

`c` ... pomocné pole buněk pro jednotlivé trasy z počátečního řešení

`n` ... celkový počet míst

`mista` ... matice souřadnic zákaznických míst se třemi sloupci, v prvním sloupci očíslování míst a ve druhém a třetím sloupci souřadnice míst

`vzdalenosti` ... matice euklidovských vzdáleností C mezi jednotlivými místy

`poptavka` ... matice s množstvím zboží požadovaným místy v druhém řádku plus řádek s očíslováním míst

`kapacita` ... kapacita vozidla

`iterace` ... počet celkových iterací

`beta` ... volitelný parametr pro výpočet granulární meze pro krátké hrany

`Tmin` ... dolní mez pro vygenerování náhodných čísel pro matici tabu_iterace, která uchovává doby tabu držení T, tedy po kolik iterací bude hrana tabu

`Tmax` ... horní mez pro vygenerování náhodných čísel pro matici tabu_iterace, která uchovává doby tabu držení T, tedy po kolik iterací bude hrana tabu

`cislo` ... nepovinný parametr, při nezadání nastaven na hodnotu 10, kolik iterací povolujeme zvětšení beta pro zvětšení okolí, abychom se mohli dostat z lokálního minima

`tol` ... nepovinný parametr, při nezadání nastaven na hodnotu 0.000001, pro toleranci pro nulový rozdíl počítaný ve všech výměnách

`var` ... nepovinný parametr, jakou variantou jsou počítány všechny výměny

- `var` $= 1$ ... varianta s funkcemi pro výměny vymena_dva, vymena_tria, vymena_trib, vymena_ctyri, kde podmínka pro nové hrany je taková, že musí být aspoň jedna krátká
- `var` $= 2$ … varianta s funkcemi pro výměny vymena_dva2, vymena_tria2, vymena_trib2, vymena_ctyri2, kde podmínka pro nové hrany je taková, že musí být hrana (a,b) krátká

`stats` ... nepovinný parametr pro to, co má tato funkce všechno vypisovat, určený pro ladění, při nezadání nastaven na hodnotu 1

- `stats`= 0 ... žádné výpisy proměnných ani vykreslení grafu
- `stats`= 1 ... výpisy aktuálních délek tras, nejlepších dosavadních délek tras, použité výměny a použité beta pro každou iteraci (1 řádek = 1 iterace)
- `stats`= 2 ... graf aktuálních cest spolu s grafem změny celkové délky tras nejlepších cest a aktuálních cest v závislosti na počtu iterací

  `dveopt` ... nepovinný parametr pro 2-opt zlepšovací heuristiku, při nezadání nastaven na hodnotu 1
- `dveopt`= 0 ... 2-opt je úplně vypnuto
- `dveopt`= 1 ... 2-opt (funkce dve_opt) je zapnuto a zlepšuje každé dvě cesty vybrané pomocí výměn, matice no_tabu je zde průběžně aktualizována
- `dveopt`= 2 ... 2-opt (funkce dve_opt2) je zařazena mezi výměny jako další způsob

Výstupní proměnné:

`nejlepsi_cesty` ... pole buněk pro jednotlivé zlepšené trasy po provedení GTS algoritmu

`delka_tras` ... zlepšená délka tras (vektor) po provedení GTS algoritmu

`celkova_delka_tras` ... zlepšená celková délka tras po provedení GTS algoritmu

`graf_cest` ... vykresluje aktuální cesty do grafu pro všechny iterace

`jednotlive_delky` ... matice, kde je v prvním sloupci celková délka tras nejlepších cest a ve druhém celková délka aktuálních tras

Volání této funkce:

```
[nejlepsi_cesty,delka_tras,celkova_delka_tras,jednotlive_delky]=…
granular_tabu_search(cesty,n,mista,vzdalenosti,poptavka,kapacita,…
iterace,beta0,Tmin,Tmax,cislo,tol,var,stats,dveopt)
```

7) **`vykresleni`**

Funkce slouží k vykreslení zákaznických míst (plus je v závorce poptávka zákazníků) do grafu s barevně rozlišeným vyznačením tras a legendou s celkovou délkou všech tras, délkou jednotlivých tras a náklady na jednotlivé trasy.

Vstupní proměnné:

`n` ... celkový počet míst

`cesty` ... pole buněk, kde jsou ve sloupci uloženy trasy ve formě vektoru

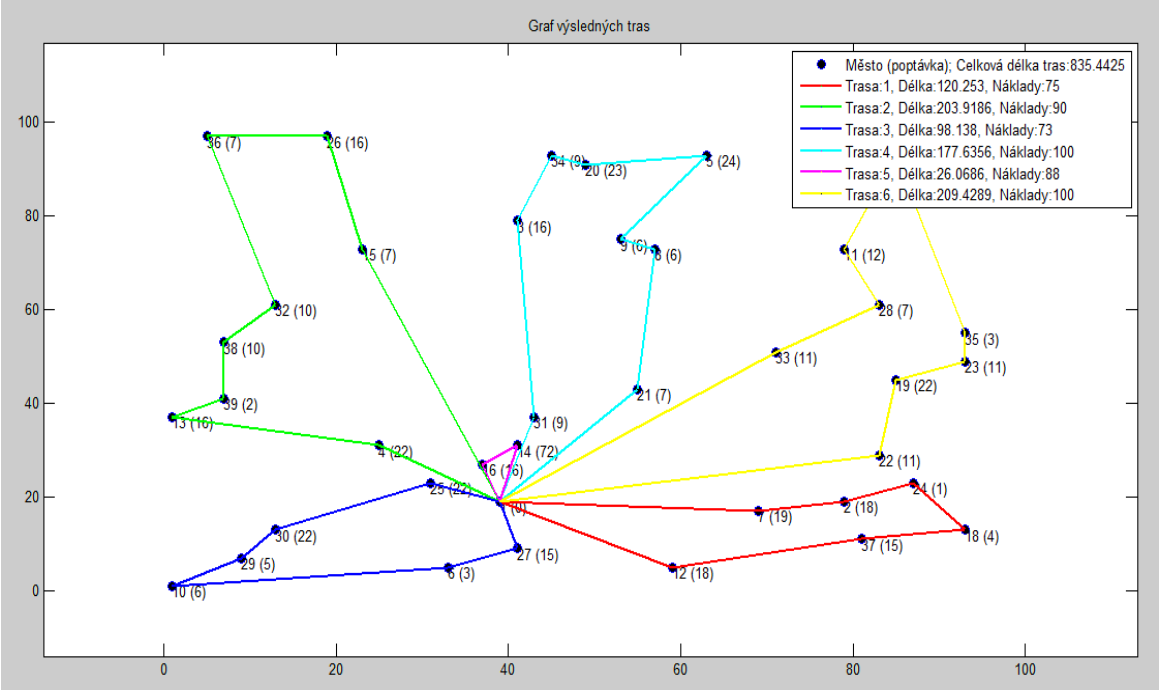
`vzdalenosti` ... matice euklidovských vzdáleností C mezi jednotlivými místy

`mista` ... matice souřadnic zákaznických míst se třemi sloupci, v prvním sloupci očíslování míst a ve druhém a třetím sloupci souřadnice míst

`poptavka` ... matice s množstvím zboží požadovaným místy v druhém řádku plus řádek s očíslováním míst

Výstupní proměnné:

`graf_cest` ... graf s vykreslením tras a legendou, viz obrázek 8



Obrázek 8 – graf s vykreslením tras a legendou

Volání této funkce:

```
graf_cest=vykresleni(n,cesty,vzdalenosti,mista,poptavka)
```

8) **`vykresleni_delek`**

Funkce je určena k vykreslení grafu změny celkové délky tras nejlepších cest a aktuálních cest v závislosti na počtu iterací.

Vstupní proměnné:

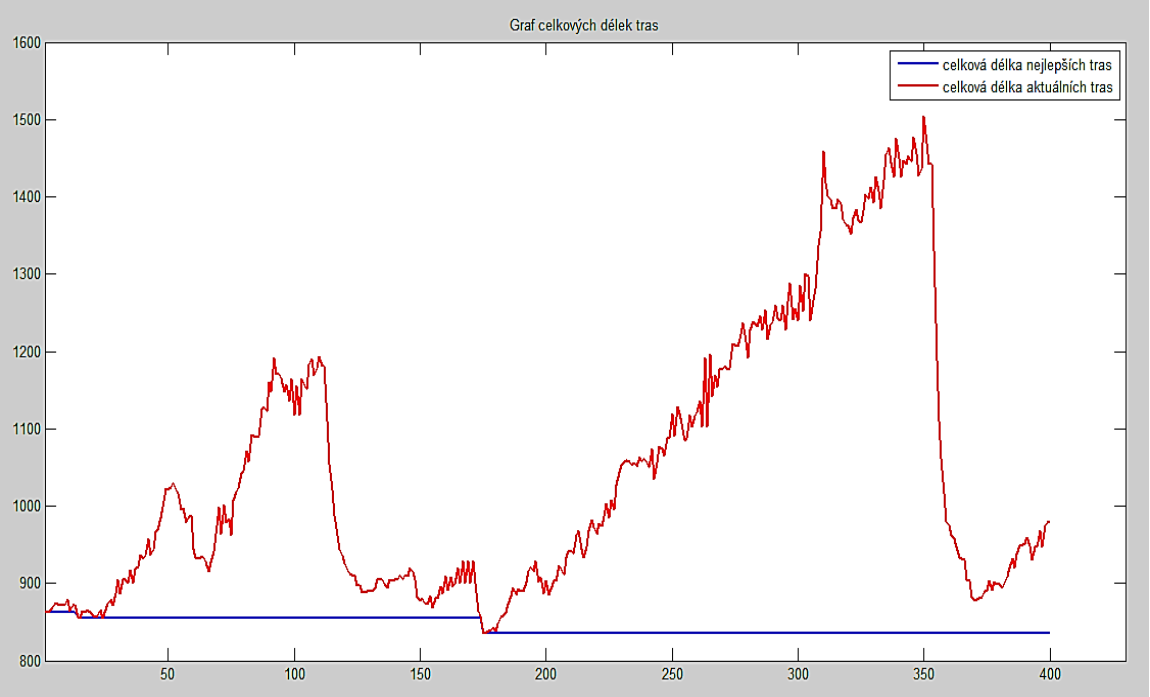
`iterace` ... počet celkových iterací

`pocet` ... aktuální iterace

`jednotlive_delky` ... matice, kde je v prvním sloupci celková délka tras nejlepších tras a ve druhém celková délka aktuálních tras

Výstupní proměnné:

`graf_celkovych_delek` ... vykreslení změny proměnné jednotlive_delky v závislosti na počtu iterací do grafu, viz obrázek 9



Obrázek 9 – graf celkových délek tras

Volání této funkce:

```
graf_celkovych_delek=vykresleni_delek(iterace,pocet,…
jednotlive_delky)
```

5.1.2. Funkce pro λ – výměny a pro 2 – opt

Tyto funkce jsou také volány ze základních funkcí, a proto je uživatel nemusí volat přímo.

1) **`vymena_dva`**

   Funkce provádí 2 – výměnu, viz kapitola 3.2.1.1. V takovéto verzi je aspoň jedna nově přidaná hrana z matice H.

2) **`vymena_dva2`**

   Funkce provádí 2 – výměnu, viz kapitola 3.2.1.1. V této verzi musí být nově přidaná hrana (a, b) z matice H.

3) **`vymena_tria`**

   Funkce provádí 3 – výměnu způsob a), viz kapitola 3.2.1.1. V takovéto verzi je aspoň jedna nově přidaná hrana z matice H.

4) **`vymena_tria2`**

   Funkce provádí 3 – výměnu způsob a), viz kapitola 3.2.1.1. V této verzi musí být nově přidaná hrana (a, b) z matice H.

5) **`vymena_trib`**

   Funkce provádí 3 – výměnu způsob b), viz kapitola 3.2.1.1. V takovéto verzi je aspoň jedna nově přidaná hrana z matice H.

6) **`vymena_trib2`**

   Funkce provádí 3 – výměnu způsob b), viz kapitola 3.2.1.1. V této verzi musí být nově přidaná hrana (a, b) z matice H.

7) **`vymena_ctyri`**

   Funkce provádí 4 – výměnu, viz kapitola 3.2.1.1. V takovéto verzi je aspoň jedna nově přidaná hrana z matice H.

8) **`vymena_ctyri2`**

   Funkce provádí 4 – výměnu, viz kapitola 3.2.1.1. V této verzi musí být nově přidaná hrana (a, b) z matice H.

9) **dve_opt**

Funkce provádí zlepšovací heuristiku 2 – opt, viz kapitola 2.2.2., pro trasy účastnící se výměn po každé iteraci.

10) **dve_opt2**

Funkce provádí zlepšovací heuristiku 2 – opt, viz kapitola 2.2.2., pro všechny trasy z počátečního řešení a hledá nejlepší změnu jako u výměn.

5.1.3. Pomocné funkce

Tyto funkce jsou volány ze základních funkcí, uživatel je nemusí volat přímo, proto si uvedeme jen jejich seznam.

1) **delka_cesty**

Funkce je určena k výpočtu délek dvou tras účastnících se výměny.

2) **kapacita_cesty**

Funkce slouží k výpočtu kapacit dvou tras účastnících se výměny.

3) **naklady_na_trasu**

Funkce, která počítá náklady na jednotlivé trasy ve formě řádkového vektoru.

4) **pocet_mest**

Funkce, která je určena ke stanovení počtu měst v jednotlivých trasách ve formě řádkového vektoru.

5) **vypocet_delek_tras**

Funkce počítá délku jednotlivých tras ve formě řádkového vektoru.

6) **vypocet_meze**

Funkce je určena k výpočtu granulární meze pro GTS algoritmus.

7) **log_matice**

Funkce je určena pro vytvoření matice H' a také pro konstrukci matice tabu listu. Dohromady tyto matice tvoří logickou matici H, kde 1 představuje povolené hrany, se kterými se mohou provádět výměny.

5.1.4. Konkrétní příklad

Nyní si ukážeme, jak funkce využívat. Příklad, jehož hlavní části si zde ukážeme, je uložen také na přiloženém CD jako skript Matlabu s názvem A_n32_k5, který si uživatel může zkusit spustit, případně si také může změnit různé parametry a podívat se, co to s výsledkem udělá.

Načítání datového souboru

Načtení datového souboru A-n32-k4 s příponou vrp:

```
[n,kapacita,poptavka,mista,mista_souradnice,opt_hodnota]=nacti_vrp
('A-VRP/','A-n32-k5.vrp');
```

Výpočet matice vzdáleností C mezi místy z datového souboru A-n32-k4:

```
vzdalenosti=pdist2(mista_souradnice,mista_souradnice);
```

Savings algoritmus

Konstrukce počátečního řešení pomocí savings algoritmu:

```
[cesty,pocet_tras,naklady_c,delka_tras,celkova_delka_tras,okruh]=…
savings(n,vzdalenosti,poptavka,kapacita);
```

Počáteční řešení může být také konstruováno i náhodně pomocí funkce `nahodne`.

```
[cesty,pocet_tras,naklady_c,delka_tras,celkova_delka_tras,okruh]=n
ahodne(n,vzdalenosti,poptavka,kapacita);
```

Nastavení tolerance pro nulový rozdíl počítaný ve výměnách:

```
tol=0.000001;
```

Zlepšovací algoritmus

Dále také můžeme počáteční řešení vylepšit zlepšovacím algoritmem, který lze spustit takto:

```
[cesty,delka_tras,celkova_delka_tras]=zlepsovaci(cesty,n,…
vzdalenosti,poptavka,kapacita,tol);
```

Granulární tabu search algoritmus

Nastavení parametrů pro funkci `granular_tabu_search`, které lze měnit:

počet celkových iterací pro GTS algoritmus,

```
iterace=700;
```

parametr pro výpočet meze pro krátké hrany,

```
beta0=1.25;
```

dolní mez pro náhodná čísla v `tabu_iterace` matici,

```
Tmin=10;
```

horní mez pro náhodná čísla v `tabu_iterace` matici,

```
Tmax=15;
```

počet iterací, po které bude beta zvětšeno beta,

```
cislo=10;
```

varianta pro výpočet výměn,

```
var=1;
```

parametr pro to, co se bude vypisovat,

```
stats=1;
```

parametr pro zapnutí nebo vypnutí 2 – opt zlepšovací heuristiky.

```
dveopt=2;
```

Samotná funkce pro GTS algoritmus:

```
[nejlepsi_cesty,delka_tras,nejlepsi_celkova_delka_tras,…
jednotlive_delky]=granular_tabu_search(cesty,n,mista,vzdalenosti,…
poptavka,kapacita,iterace,beta0,Tmin,Tmax,cislo,tol,var,…
stats,dveopt);
```

Vykreslení

vykreslení grafu změny celkové délky tras nejlepších cest a aktuálních cest v závislosti na počtu iterací,

```
pocet=iterace;
figure(1)
graf_celkovych_delek=vykresleni_delek(iterace,pocet,…
jednotlive_delky);
```

vykreslení výsledných tras do grafu s legendou.

```
figure(2)
```

```
nejlepsi_graf_cest=vykresleni(n,nejlepsi_cesty,vzdalenosti,…
mista,poptavka);
```

5.2. Testy

K provedení testů parametrů, měření času, měření odchylky od optimální hodnoty, atd. jsem použila data z této stránky [8].

Tabulka 1 – Přehledná tabulka s datovými soubory

Název úlohy	Počet míst – n	Kapacita vozidla	Minimální počet vozidel – k	Optimální hodnota	Nejlepší dosažená hodnota
A-n32-k5	32	100	5	784	
A-n33-k5	33	100	5	661	
A-n33-k6	33	100	6	742	
A-n34-k5	34	100	5	778	
A-n36-k5	36	100	5	799	
A-n37-k5	37	100	5	669	
A-n37-k6	37	100	6	949	
A-n38-k5	38	100	5	730	
A-n39-k5	39	100	5	822	
A-n39-k6	39	100	6	831	
A-n44-k6	44	100	6	937	
A-n45-k6	45	100	6	944	
A-n45-k7	45	100	7		1146
A-n46-k7	46	100	7	914	
A-n48-k7	48	100	7		1073
A-n53-k7	53	100	7	1010	
A-n54-k7	54	100	7	1167	
A-n55-k9	55	100	9	1073	
A-n60-k9	60	100	9		1408
A-n61-k9	61	100	9	1034	
A-n62-k8	62	100	8		1290
A-n63-k9	63	100	9		1634
A-n63-k10	63	100	10		1315
A-n64-k9	64	100	9		1402
A-n65-k9	65	100	9	1174	
A-n69-k9	69	100	9		1168
A-n80-k10	80	100	10		1764

Pro testování jsem použila 20 datových soborů z tabulky 1, pro které lze dohledat optimální výsledky i s konkrétními trasami. Datové soubory mají název ve tvaru například

A-n53-k7, kde 53 znamená počet míst včetně depa a 7 minimální počet dostupných vozidel. Z těchto 20 souborů jsem pak ještě pro konkrétní testy vybírala 10, až na test celkového počtu iterací, kde jsem použila všechny. Parametr `stats` je ve všech testech nastaven na 0, aby byl výpočet rychlejší.

5.2.1. Testování celkového počtu iterací

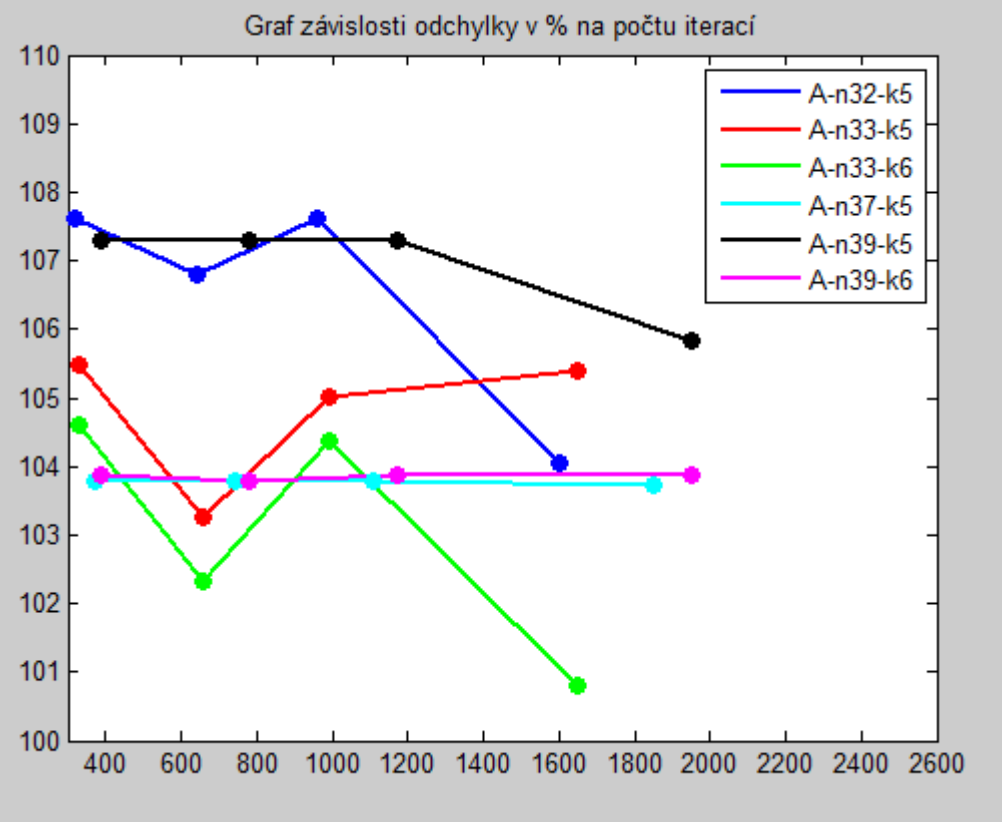
V tomto případě jsem testovala všech 20 souborů, parametry byly nastaveny na doporučené hodnoty z článku [5]:

`beta0` = 1,25; `Tmin`= 5; `Tmax`= 10;

další parametry: `cislo`= 10; `tol`= 0,000001; `var`= 1.

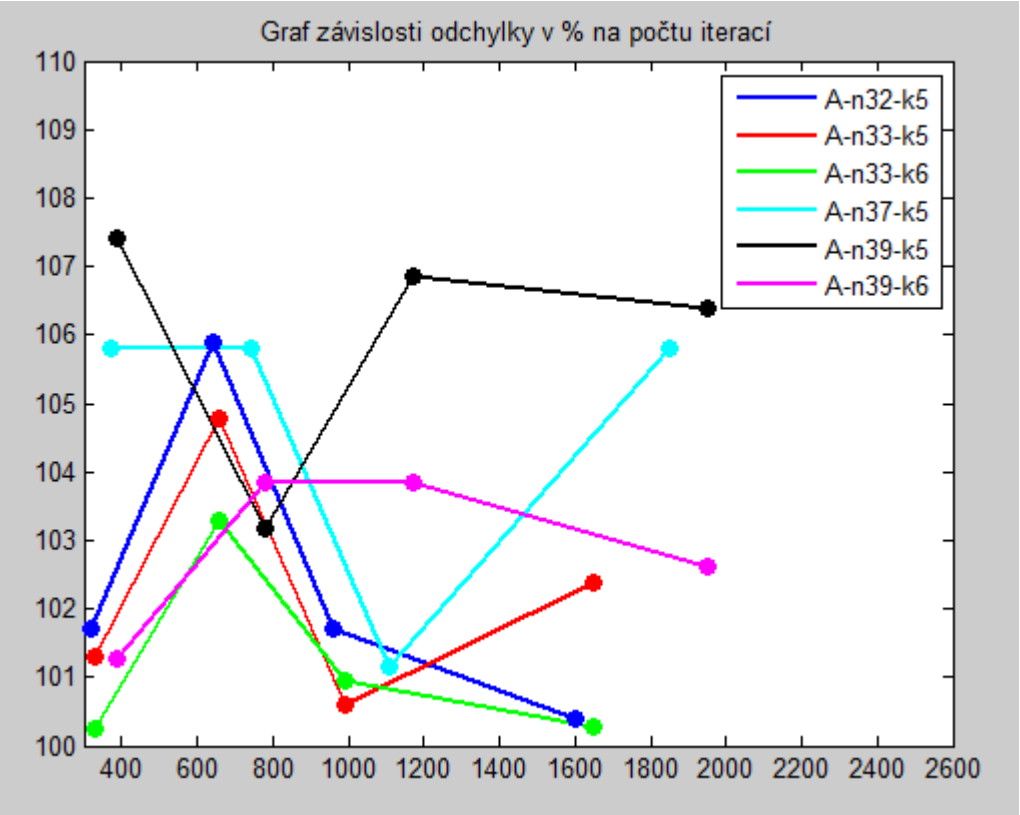
Testovala jsem čtyři možnosti počtu celkových iterací:

`iterace` $= n * 10$, `iterace` $= n * 20$, `iterace` $= n * 30$, `iterace` $= n * 50$. Pro každý z těch počtů iterací proběhl test znovu, a proto mohlo vlivem náhodných faktorů dojít k tomu, že pro vyšší počet iterací je v některých případech řešení horší.



Obrázek 10 – vykreslení závislosti odchylky v % na počtu iterací

Nejdříve jsem testovala variantu bez 2 – opt jako výměny, tedy `dveopt` $= 0$. V obrázku 10 je vykreslena závislost odchylky od optimální hodnoty celkové délky tras v procentech na celkovém počtu iterací pro vybrané datové soubory uvedené v legendě obrázku.



Obrázek 11 – vykreslení závislosti odchylky v % na počtu iterací

Dále jsem testovala variantu s 2 – opt jako výměnou, tedy `dveopt` $= 2$. V obrázku 11 je vykreslen graf závislosti odchylky od optimální hodnoty celkové délky tras v procentech na celkovém počtu iterací pro případy jako v obrázku 10.

Pro další testování jsem se rozhodla na základě výše uvedených obrázku 10 a 11, použít celkový počet iterací rovný $n * 25$, takže něco mezi testovanými $n * 20$ a $n * 30$, protože méně iterací nezajistí, tak dobré výsledky, ale na druhou stranu velký počet iterací znamená značné zvýšení nároků na čas.

5.2.2. Testování parametrů β, T_{min} a T_{max}

Na základě testu celkového počtu iterací jsem se rozhodla u testů parametrů β, T_{min} a T_{max} použít variantu algoritmu s 2 – opt jako výměnou, tedy `dveopt`$= 2$, protože 2 – opt je efektivním zesilovacím nástrojem. Dostaneme se tak k dobrým výsledkům mnohem

rychleji za menší počet iterací než jako u verze bez 2 – opt. V GTS algoritmu se vyskytuje náhoda, proto jsou pro všechny hodnoty parametrů algoritmy spuštěny desetkrát a v tabulkách jsou uvedeny průměrné (Průměr GTS), minimální (Minimum GTS) a maximální hodnoty (Maximum GTS) celkové délky tras a také průměrný čas, za který dostaneme GTS algoritmem, kde počáteční řešení je ze savings algoritmu Clarka a Wrighta (Celková délka tras savings), výsledky. V tabulkách jsou také uvedeny v prvním sloupci konkrétní úlohy z tabulky 1 (Úloha), v druhém celkové počty iterací (Celkové iterace), v posledním sloupci optimální hodnoty získané z tabulky 1 (Opt. hodnota) a průměrná odchylka průměrné hodnoty celkové délky tras provedením GTS algoritmu od optimální hodnoty v procentech (Průměrná % odchylka), navíc zde uvádím i minimální odchylku od optimální hodnoty a také maximální hodnotu od optimální hodnoty.

Test parametru β

V tomto testování je vybráno 10 datových soborů pro testování a testované hodnoty parametru β (ve funkci `granular_tabu_search` je to `beta0`) jsou rovné 1,1; 1,25; 1,3; 1,4; 1,6; 1,8; 5. Ostatní parametry jsou nastaveny na doporučené hodnoty z článku [5]:

`Tmin`= 5; `Tmax`= 10;

další parametry: `cislo`= 10; `tol`= 0,000001; `var`= 1.

V tomto testu záleží, pro různé hodnoty β, na času výpočtu, protože se zvětšující hodnotou parametru β prohledáváme širší strukturu okolí a provádíme více výměn, což může trvat o něco déle.

Tabulka 2 - $\beta = 1,1$

Úloha	Celkové iterace	Čas	Celková délka tras savings	Průměr GTS	Minimum GTS	Maximum GTS	Opt. hodnota
A-n32-k5	800	21,069	843,688	787,154	787,082	787,202	784
A-n33-k6	825	25,391	776,256	744,326	742,830	746,459	742
A-n34-k5	850	25,713	810,411	791,100	788,796	794,501	778
A-n36-k5	900	29,243	828,472	804,869	804,869	804,869	799
A-n38-k5	950	36,133	768,132	758,474	734,617	765,277	730
A-n39-k5	975	37,297	901,990	856,243	846,563	864,380	822
A-n44-k6	1100	56,290	976,038	948,708	948,708	948,708	937
A-n46-k7	1150	61,229	939,744	926,214	926,214	926,214	914
A-n48-k7	1200	71,655	1112,822	1098,866	1098,866	1098,866	1073
A-n54-k7	1350	101,901	1201,197	1188,450	1188,450	1188,450	1167
			Průměrná % odchylka	1,803514	Max 4,165834	Min 0,313459	

Tabulka 3 - $\beta = 1,25$

Úloha	Celkové iterace	Čas	Celková délka tras savings	Průměr GTS	Minimum GTS	Maximum GTS	Opt. hodnota
A-n32-k5	800	20,990	843,688	798,612	787,082	816,505	784
A-n33-k6	825	26,077	776,256	745,689	743,004	748,341	742
A-n34-k5	850	26,716	810,411	790,952	785,405	792,919	778
A-n36-k5	900	31,120	828,472	816,145	806,168	818,639	799
A-n38-k5	950	38,712	768,132	747,862	747,862	747,862	730
A-n39-k5	975	37,561	901,990	850,618	836,582	867,016	822
A-n44-k6	1100	57,052	976,038	947,772	944,026	948,708	937
A-n46-k7	1150	61,536	939,744	926,214	926,214	926,214	914
A-n48-k7	1200	72,744	1112,822	1098,866	1098,866	1098,866	1073
A-n54-k7	1350	104,118	1201,197	1185,757	1185,757	1185,757	1167
			Průměrná % odchylka	1,86037	Max 3,481457	Min 0,497189	

Tabulka 4- $\beta = 1,3$

Úloha	Celkové iterace	Čas	Celková délka tras savings	Průměr GTS	Minimum GTS	Maximum GTS	Opt. hodnota
A-n32-k5	800	19,921	843,688	798,255	787,202	816,953	784
A-n33-k6	825	24,815	776,256	745,621	743,004	750,253	742
A-n34-k5	850	25,289	810,411	791,344	782,515	793,986	778
A-n36-k5	900	29,009	828,472	813,022	806,783	818,639	799
A-n38-k5	950	36,520	768,132	747,862	747,862	747,862	730
A-n39-k5	975	38,126	901,990	842,848	830,358	847,662	822
A-n44-k6	1100	57,137	976,038	948,708	948,708	948,708	937
A-n46-k7	1150	62,883	939,744	926,214	926,214	926,214	914
A-n48-k7	1200	72,328	1112,822	1095,970	1084,387	1098,866	1073
A-n54-k7	1350	103,282	1201,197	1185,609	1185,609	1185,609	1167
			Průměrná % odchylka	1,708065	Max 2,536275	Min 0,487966	

Tabulka 5 - $\beta = 1,4$

Úloha	Celkové iterace	Čas	Celková délka tras savings	Průměr GTS	Minimum GTS	Maximum GTS	Opt. hodnota
A-n32-k5	800	20,528	843,688	787,688	787,688	787,688	784
A-n33-k6	825	25,444	776,256	746,658	742,693	750,889	742
A-n34-k5	850	26,107	810,411	790,604	780,976	798,045	778
A-n36-k5	900	29,549	828,472	818,195	816,416	818,639	799
A-n38-k5	950	37,155	768,132	749,862	747,862	755,862	730
A-n39-k5	975	39,500	901,990	844,203	837,915	854,261	822
A-n44-k6	1100	59,891	976,038	948,708	948,708	948,708	937
A-n46-k7	1150	64,135	939,744	926,214	926,214	926,214	914
A-n48-k7	1200	75,234	1112,822	1098,866	1098,866	1098,866	1073

A-n54-k7	1350	106,958	1201,197	1185,609	1185,609	1185,609	1167
			Průměrná % odchylka	1,713361	Max 2,720886	Min 0,470378	

Tabulka 6 - $\beta = 1,6$

Úloha	Celkové iterace	Čas	Celková délka tras savings	Průměr GTS	Minimum GTS	Maximum GTS	Opt. hodnota
A-n32-k5	800	21,420	843,688	814,885	787,082	830,922	784
A-n33-k6	825	26,701	776,256	745,288	743,472	752,554	742
A-n34-k5	850	27,094	810,411	794,301	791,463	795,010	778
A-n36-k5	900	30,829	828,472	818,025	815,569	818,639	799
A-n38-k5	950	38,800	768,132	747,862	747,862	747,862	730
A-n39-k5	975	40,307	901,990	857,351	842,647	861,028	822
A-n44-k6	1100	62,447	976,038	947,042	947,042	947,042	937
A-n46-k7	1150	66,702	939,744	926,214	926,214	926,214	914
A-n48-k7	1200	77,664	1112,822	1098,866	1098,866	1098,866	1073
A-n54-k7	1350	111,634	1201,197	1185,609	1185,609	1185,609	1167
			Průměrná % odchylka	2,201976	Max 4,300659	Min 0,443142	

Tabulka 7 - $\beta = 1,8$

Úloha	Celkové iterace	Čas	Celková délka tras savings	Průměr GTS	Minimum GTS	Maximum GTS	Opt. hodnota
A-n32-k5	800	21,568	843,688	821,438	792,381	828,702	784
A-n33-k6	825	26,868	776,256	744,306	743,441	745,365	742
A-n34-k5	850	27,945	810,411	795,979	791,328	798,045	778
A-n36-k5	900	32,241	828,472	818,639	818,639	818,639	799
A-n38-k5	950	40,367	768,132	747,862	747,862	747,862	730
A-n39-k5	975	42,777	901,990	847,653	837,113	883,699	822
A-n44-k6	1100	64,365	976,038	947,042	947,042	947,042	937
A-n46-k7	1150	68,731	939,744	926,214	926,214	926,214	914
A-n48-k7	1200	78,749	1112,822	1098,866	1098,866	1098,866	1073
A-n54-k7	1350	113,074	1201,197	1185,609	1185,609	1185,609	1167
			Průměrná % odchylka	2,183588	Max 4,775228	Min 0,310816	

Tabulka 8 - $\beta = 5$

Úloha	Celkové iterace	Čas	Celková délka tras savings	Průměr GTS	Minimum GTS	Maximum GTS	Opt. hodnota
A-n32-k5	800	22,729	843,688	828,702	828,702	828,702	784
A-n33-k6	825	30,916	776,256	768,421	768,421	768,421	742
A-n34-k5	850	32,495	810,411	798,045	798,045	798,045	778
A-n36-k5	900	35,677	828,472	812,985	812,985	812,985	799
A-n38-k5	950	51,213	768,132	747,862	747,862	747,862	730
A-n39-k5	975	46,070	901,990	874,143	874,143	874,143	822

A-n44-k6	1100	73,239	976,038	947,042	947,042	947,042	937
A-n46-k7	1150	79,277	939,744	924,669	918,491	926,214	914
A-n48-k7	1200	87,941	1112,822	1098,866	1098,866	1098,866	1073
A-n54-k7	1350	127,872	1201,197	1185,609	1185,609	1185,609	1167
			Průměrná % odchylka	2,862393	Max 6,343488	Min 1,071739	

V uvedených tabulkách 2 – 8 je červeně vyznačen sloupec průměrné celkové délky tras GTS algoritmu pro jednu určitou hodnotu parametru β, zeleně je pak vyznačena hodnota průměrné procentuální odchylky od optimální hodnoty. Z tabulek vyplývá, že jako dobrá se jeví hodnota parametru β rovna 1,3; případně 1,4. Hodnoty vyšší než 1,5 nejsou příliš výhodné, neboť nezlepšují kvalitu řešení, čas výpočtu je u nich vyšší a roste s rostoucím β, neboť je zkoumáno více různých výměn vrcholů. V dalších testech jsem tedy použila $\beta = 1{,}3$.

Test parametrů T_{min}, T_{max}

V tomto testování je vybráno stejných 10 datových soborů jako u testu β a testované hodnoty parametru T_{min} jsou rovné $3;\ 5;\ 10;\ 20;\ 26$ a hodnoty parametru T_{max} jsou rovné $6;\ 10;\ 15;\ 23;\ 31$. Parametr $\beta = 1{,}3$. Ostatní parametry jsou nastaveny na: `cislo`$= 10$; `tol`$= 0{,}000001$; `var`$= 1$.

Zde nehraje roli čas, který je pro všechny dvojice T_{min}, T_{max} přibližně stejný jako u testu pro parametr $\beta = 1{,}3$, proto místo času jsou v tabulkách uvedeny konkrétní testované dvojice.

Tabulka 9 - $T_{min} = 3, T_{max} = 6$

Úloha	Celkové iterace	Tmin	Tmax	Celková délka tras savings	Průměr GTS	Minimum GTS	Maximum GTS	Opt. hodnota
A-n32-k5	800	3	6	843,688	789,455	787,082	797,451	784
A-n33-k6	825	3	6	776,256	749,645	748,605	750,253	742
A-n34-k5	850	3	6	810,411	787,170	780,936	794,482	778
A-n36-k5	900	3	6	828,472	818,639	818,639	818,639	799
A-n38-k5	950	3	6	768,132	747,862	747,862	747,862	730
A-n39-k5	975	3	6	901,990	842,584	839,703	847,662	822
A-n44-k6	1100	3	6	976,038	948,708	948,708	948,708	937
A-n46-k7	1150	3	6	939,744	923,362	918,955	926,214	914
A-n48-k7	1200	3	6	1112,822	1098,866	1098,866	1098,866	1073
A-n54-k7	1350	3	6	1201,197	1186,231	1185,609	1186,715	1167
				Průměrná % odchylka	1,664615	Max 2,504131	Min 0,69576	

Tabulka 10 - $T_{min} = 5, T_{max} = 10$

Úloha	Celkové iterace	Tmin	Tmax	Celková délka tras savings	Průměr GTS	Minimum GTS	Maximum GTS	Opt. hodnota
A-n32-k5	800	5	10	843,688	791,932	787,202	799,888	784
A-n33-k6	825	5	10	776,256	745,849	742,693	750,253	742
A-n34-k5	850	5	10	810,411	791,736	780,976	796,576	778
A-n36-k5	900	5	10	828,472	814,737	809,755	818,639	799
A-n38-k5	950	5	10	768,132	747,603	746,564	747,862	730
A-n39-k5	975	5	10	901,990	845,178	841,395	847,662	822
A-n44-k6	1100	5	10	976,038	948,708	948,708	948,708	937
A-n46-k7	1150	5	10	939,744	924,639	919,168	926,214	914
A-n48-k7	1200	5	10	1112,822	1095,970	1084,387	1098,866	1073
A-n54-k7	1350	5	10	1201,197	1185,609	1185,609	1185,609	1167
				Průměrná % odchylka	1,664552	Max 2,819728	Min 0,51869	

Tabulka 11 - $T_{min} = 10, T_{max} = 15$

Úloha	Celkové iterace	Tmin	Tmax	Celková délka tras savings	Průměr GTS	Minimum GTS	Maximum GTS	Opt. hodnota
A-n32-k5	800	10	15	843,688	797,628	787,082	815,081	784
A-n33-k6	825	10	15	776,256	747,260	743,004	750,253	742
A-n34-k5	850	10	15	810,411	794,300	792,672	795,207	778
A-n36-k5	900	10	15	828,472	816,743	809,157	818,639	799
A-n38-k5	950	10	15	768,132	747,829	747,696	747,862	730
A-n39-k5	975	10	15	901,990	839,518	831,750	847,662	822
A-n44-k6	1100	10	15	976,038	948,708	948,708	948,708	937
A-n46-k7	1150	10	15	939,744	926,214	926,214	926,214	914
A-n48-k7	1200	10	15	1112,822	1098,145	1095,262	1098,866	1073
A-n54-k7	1350	10	15	1201,197	1185,609	1185,609	1185,609	1167
				Průměrná % odchylka	1,786026	Max 2,442365	Min 0,70884	

Tabulka 12 - $T_{min} = 20, T_{max} = 23$

Úloha	Celkové iterace	Tmin	Tmax	Celková délka tras savings	Průměr GTS	Minimum GTS	Maximum GTS	Opt. hodnota
A-n32-k5	800	20	23	843,688	805,829	794,451	814,156	784
A-n33-k6	825	20	23	776,256	750,992	746,042	754,482	742
A-n34-k5	850	20	23	810,411	792,889	787,535	796,568	778
A-n36-k5	900	20	23	828,472	818,639	818,639	818,639	799
A-n38-k5	950	20	23	768,132	747,862	747,862	747,862	730
A-n39-k5	975	20	23	901,990	845,148	835,089	847,662	822

A-n44-k6	1100	20	23	976,038	948,708	948,708	948,708	937
A-n46-k7	1150	20	23	939,744	926,214	926,214	926,214	914
A-n48-k7	1200	20	23	1112,822	1098,866	1098,866	1098,866	1073
A-n54-k7	1350	20	23	1201,197	1185,609	1185,609	1185,609	1167
				Průměrná % odchylka	2,022184	Max 2,816029	Min 1,211838	

Tabulka 13 - $T_{min} = 26, T_{max} = 31$

Úloha	Celkové iterace	Tmin	Tmax	Celková délka tras savings	Průměr GTS	Minimum GTS	Maximum GTS	Opt. hodnota
A-n32-k5	800	26	31	843,688	802,753	792,528	830,664	784
A-n33-k6	825	26	31	776,256	750,617	745,376	754,482	742
A-n34-k5	850	26	31	810,411	794,397	790,370	798,045	778
A-n36-k5	900	26	31	828,472	817,673	813,808	818,639	799
A-n38-k5	950	26	31	768,132	747,862	747,862	747,862	730
A-n39-k5	975	26	31	901,990	854,122	839,522	866,620	822
A-n44-k6	1100	26	31	976,038	948,708	948,708	948,708	937
A-n46-k7	1150	26	31	939,744	926,214	926,214	926,214	914
A-n48-k7	1200	26	31	1112,822	1098,866	1098,866	1098,866	1073
A-n54-k7	1350	26	31	1201,197	1185,609	1185,609	1185,609	1167
				Průměrná % odchylka	2,094358	Max 3,9078	Min 1,161288	

V uvedených tabulkách 9 – 13 je také červeně vyznačen sloupec průměrné celkové délky tras z 10 spuštění algoritmu pro dvojici hodnot parametrů T_{min} a T_{max}, zeleně je pak vyznačena hodnota průměrné procentuální odchylky od optimální hodnoty, je zde uvedena i maximální a minimální průměrná procentuální odchylka optimální hodnoty. Tudíž z těchto hodnot je patrné, že dvojice hodnot $T_{min} = 3$ a $T_{max} = 6$; $T_{min} = 5$ a $T_{max} = 10$ doporučená článkem [5] se ukazují jako dobré pro použití i v dalších testech. Vybrala jsem však dvojici $T_{min} = 3$ a $T_{max} = 6$ z důvodu o něco menší maximální průměrné odchylky, ale určitě by se dala použít i druhá dvojice.

5.2.3. Testování různých variant algoritmu

V těchto testech jsem opět vybrala 10 datových souborů z tabulky 1 a testovala jsem různé varianty algoritmu pro 10 spuštění, kde parametry jsou nastaveny na výsledné hodnoty z předchozích testů:

`beta0` $= 1{,}3$, `Tmin` $= 3$; `Tmax` $= 6$; `iterace` $= 25 * n$;

další par1ametry: `cislo` $= 10$; `tol` $= 0{,}000001$; `var` $= 1$.

Varianta algoritmu s počátečním řešením z heuristiky Clarka a Wrighta (savings)

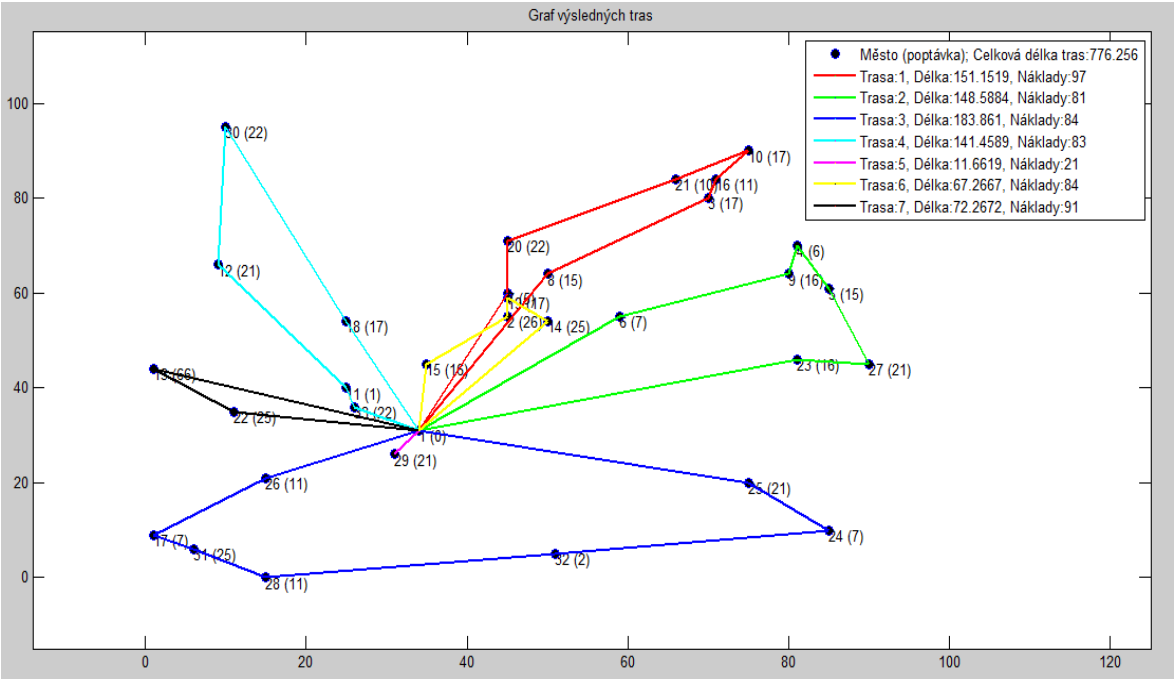
Tato varianta je klasická podoba GTS algoritmu popsaného v kapitole 4 s vylepšením pomocí 2 – opt jako výměny, tedy `dveopt` = 2. Pro tuto variantu jsou provedeny i výše zmíněné testy parametrů β, T_{min}, T_{max}.

Tabulka 14

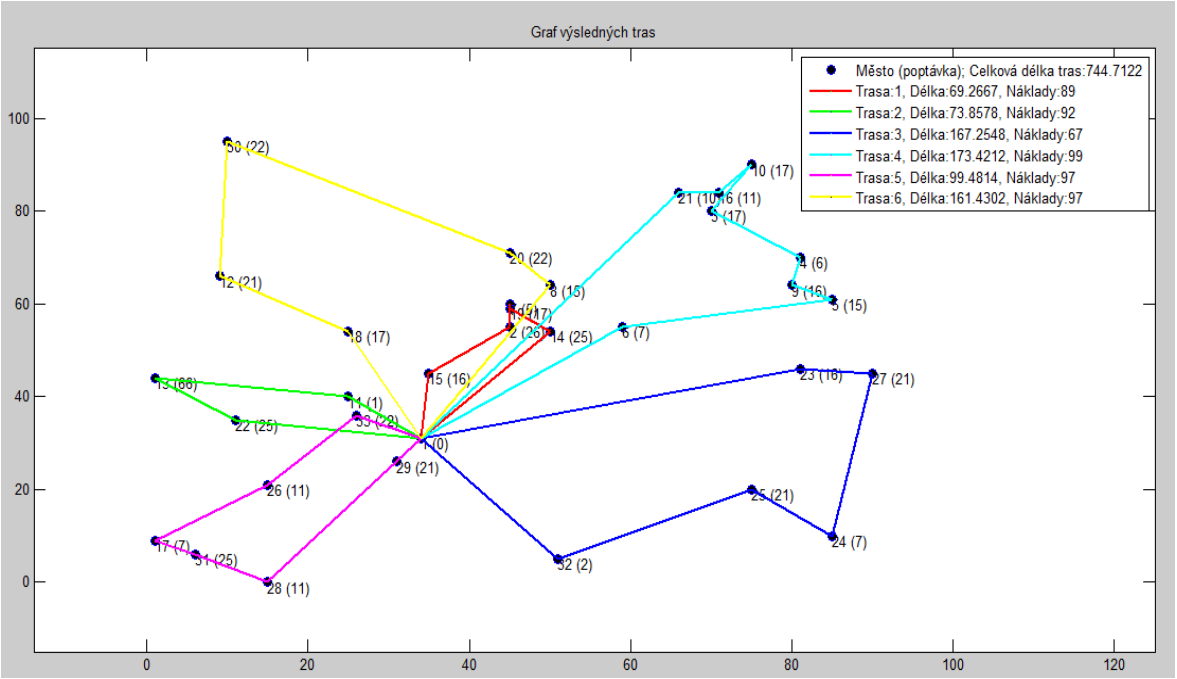
Úloha	Celk. iterace	Čas	Celková délka tras sav.	Průměr GTS	Minim. GTS	Maxim. GTS	Po. tras	Opt. hod.	Odchylka od opt. hodnoty v %
A-n32-k5	800	21,528	843,688	800,401	788,415	805,313	5	784	0,563
A-n33-k6	825	26,869	776,256	750,991	744,712	760,398	6	742	0,366
A-n34-k5	850	30,021	810,411	800,195	792,155	809,272	5	778	1,819
A-n36-k5	900	31,735	828,472	825,854	811,652	828,472	5	799	1,583
A-n38-k5	950	39,495	768,132	766,864	755,458	768,132	5	730	3,487
A-n39-k5	975	41,255	901,990	861,309	840,775	887,312	5	822	2,284
A-n44-k6	1100	62,070	976,038	962,782	962,782	962,782	6	937	2,752
A-n46-k7	1150	66,911	939,744	936,986	929,266	938,916	7	914	1,670
A-n48-k7	1200	77,631	1112,822	1110,279	1100,105	1112,822	7	1073	2,526
A-n54-k7	1350	111,452	1201,197	1186,715	1186,715	1186,715	7	1167	1,689

V tabulce 14 jsou uvedeny výsledky GTS algoritmu s počátečním řešením z heuristiky Clarka a Wrighta a to pro vybrané úlohy. V tabulce 14 je uvedeno: název úlohy (úloha), celkový počet iterací (Celk. iterace), průměrný čas celého výpočtu GTS algoritmu včetně počátečního řešení za 10 spuštění (Čas), celková délka tras zjištěná pomocí heuristiky Clarka a Wrighta, průměrnou celkovou délku tras provedením GTS algoritmu za 10 spuštění (Průměr GTS), minimální celkovou tras provedením GTS algoritmu za 10 spuštění (Minim. GTS), maximální tras provedením GTS algoritmu za 10 spuštění (Maxim. GTS), celkový počet tras pro minimální celkovou tras provedením GTS algoritmu za 10 spuštění (Po. tras), optimální hodnota (Opt. hod.), odchylka minimální celkové délky tras provedením GTS algoritmu od optimální hodnoty v procentech vyznačena červeně (Odchylka od opt. hodnoty v %). Podle odchylek poznáme, že pro datové soubory s 32 a 33 místy jsme dosáhli téměř optimální hodnoty. Proto jsou níže v obrázcích 12, 13 a 14, pro úlohu A-n33-k6, vykresleny trasy po provedení heuristiky savings, dále po provedení GTS algoritmu a také pro srovnání je zde vykresleno optimální

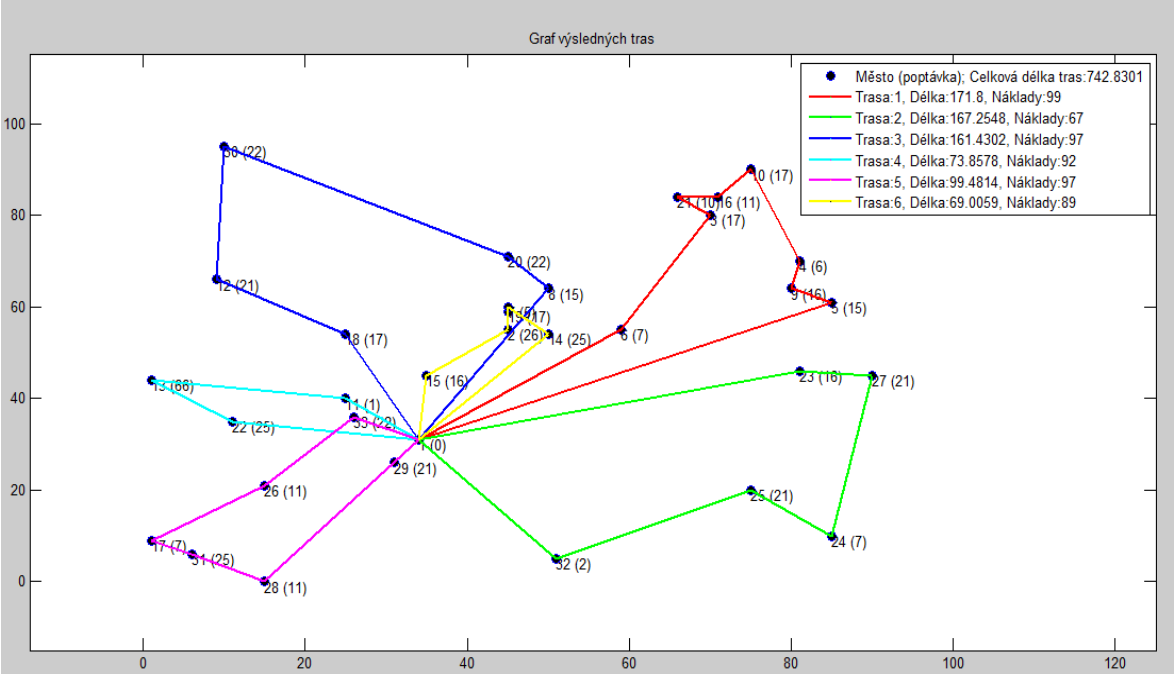
řešení. Ostatní odchylky také nejsou nijak vysoké, proto získáváme za velmi přijatelný čas dostatečně dobré výsledky.



Obrázek 12 – vykreslení tras ze savings



Obrázek 13 – vykreslení výsledných tras GTS algoritmu



Obrázek 14 – vykreslení tras optimálního řešení

Varianta algoritmu s náhodným počátečním řešením

V této variantě nejdříve rozhodíme místa do tras náhodně s ohledem na kapacitu trasy a s tímto počátečním řešením je proveden algoritmus GTS popsaný v kapitole 4 s vylepšením pomocí 2 – opt jako výměny, tedy `dveopt` = 2.

Tabulka 15

Úloha	Celk. iterace	Čas	Celková délka tras náh.	Průměr GTS	Minim. GTS	Maxim. GTS	Po. tras	Opt. hod.	Odchylka od opt. hodnoty v %
A-n32-k5	800	22,224	1929,412	813,557	791,079	847,219	5	784	0,903
A-n33-k6	825	27,677	1904,784	748,757	743,265	755,672	6	742	0,170
A-n34-k5	850	28,398	2108,770	801,404	790,570	823,193	5	778	1,616
A-n36-k5	900	32,613	2040,309	837,625	827,386	851,751	5	799	3,553
A-n38-k5	950	40,862	2179,366	778,317	748,703	806,223	5	730	2,562
A-n39-k5	975	42,102	2258,169	871,415	860,106	885,204	5	822	4,636
A-n44-k6	1100	65,995	2584,383	993,333	961,510	1026,038	6	937	2,616
A-n46-k7	1150	72,716	2767,075	959,434	930,966	984,750	7	914	1,856
A-n48-k7	1200	82,260	2818,275	1141,857	1129,511	1163,573	7	1073	5,267
A-n54-k7	1350	113,384	3178,192	1240,802	1196,861	1266,212	7	1167	2,559

Tabulka 15 je složena ze stejných sloupců jako tabulka 14 až na sloupec s celkovou délkou tras po provedení algoritmu savings, který je v tomto případě nahrazen

celkovou délkou tras po rozhození míst do trasy náhodně. Z tabulky 15 můžeme vyčíst, z jaké počáteční hodnoty celkové délky tras jsme u jednotlivých úloh vycházeli (Celková délka tras náh.), a k jaké hodnotě jsme se v jednotlivých úlohách pomocí GTS algoritmu dostali (Minim. GTS). Lze se také podívat na odchylku výsledku GTS od optimální hodnoty v posledním sloupci, kde zjistíme, že odchylka zde překvapivě není až tak velká, ale dá se říci, že se zvětšujícím počtem míst v datovém souboru se tato odchylka zvětšuje. Z časového hlediska je tato varianta nejnáročnější z toho důvodu, že počáteční řešení vypočtené náhodným algoritmem trvá o něco déle než heuristika savings.

Varianta algoritmu GTS s počátečním řešením heuristikou savings

V této variantě je GTS algoritmus popsaný v kapitole 4 vylepšen pomocí 2 – opt tentokrát tak, že se zlepšují v každé iteraci dvě trasy účastnící se výměny, tedy `dveopt` = 1.

Tabulka 16

Úloha	Celk. iterace	Čas	Celková délka tras sav.	Průměr GTS	Minim. GTS	Maxim. GTS	Po. tras	Opt. hod.	Odchylka od opt. hodnoty v %
A-n32-k5	800	22,109	843,688	793,788	787,082	797,961	5	784	0,393
A-n33-k6	825	28,353	776,256	747,269	743,074	757,225	6	742	0,145
A-n34-k5	850	29,240	810,411	792,581	787,523	793,414	5	778	1,224
A-n36-k5	900	32,762	828,472	814,587	814,430	815,995	5	799	1,931
A-n38-k5	950	41,979	768,132	751,559	749,496	756,372	5	730	2,671
A-n39-k5	975	42,795	901,990	842,637	831,055	852,494	5	822	1,102
A-n44-k6	1100	65,591	976,038	949,989	949,989	949,989	6	937	1,386
A-n46-k7	1150	69,906	939,744	923,423	919,168	923,896	7	914	0,565
A-n48-k7	1200	82,510	1112,822	1107,144	1104,328	1107,457	7	1073	2,920
A-n54-k7	1350	128,319	1201,197	1186,985	1185,578	1187,337	7	1167	1,592

V tabulce 16 jsou uvedeny úplně stejné sloupce jako u tabulky 14. Z této tabulky je patrné, že použití `dveopt` = 1 vyžaduje nejvíce času, ale výsledky takto získané jsou velmi uspokojivé podle toho, co vidíme ve sloupci s odchylkou.

Varianta GTS algoritmu s počátečním řešením z heuristiky savings vylepšené zlepšovacím algoritmem

Tato varianta je GTS algoritmem z kapitoly 4 s vylepšením pomocí 2 – opt jako výměny, tedy `dveopt` = 2 a dalším vylepšením pomocí zlepšovacího algoritmu.

Tabulka 17

Úloha	Celk. iterace	Čas	Celková délka tras sav.	Celk. délka tras zlep.	Průměr GTS	Minim. GTS	Po. tras	Opt. hod.	Odchylka od opt. hodnoty v %
A-n32-k5	800	21,136	843,688	828,702	796,540	787,082	5	784	0,393
A-n33-k6	825	25,843	776,256	776,018	745,527	742,830	6	742	0,112
A-n34-k5	850	26,061	810,411	803,865	793,212	786,437	5	778	1,084
A-n36-k5	900	30,010	828,472	819,544	817,888	813,470	5	799	1,811
A-n38-k5	950	37,442	768,132	766,225	745,812	734,185	5	730	0,573
A-n39-k5	975	38,870	901,990	886,421	847,970	831,077	5	822	1,104
A-n44-k6	1100	58,598	976,038	952,646	951,537	951,537	6	937	1,551
A-n46-k7	1150	63,122	939,744	926,214	924,069	918,955	7	914	0,542
A-n48-k7	1200	73,942	1112,822	1103,986	1097,360	1083,807	7	1073	1,007
A-n54-k7	1350	104,764	1201,197	1185,609	1185,609	1185,609	7	1167	1,595

V tabulce 17 jsou opět stejné sloupce jako v tabulce 14 až na to, že zde je přidán sloupec vylepšené celkové délky tras zlepšovacím algoritmem (Celk. délka tras zlep.) a není zde zachován sloupec s maximální celkovou délkou tras GTS algoritmu z 10 spuštění (Maxim. GTS) z důvodu nedostatečné velikosti stránky. Tato varianta se na základě nejnižších odchylek v posledním sloupci jeví jako nejlepší z uvedených s tím, že časová náročnost není až tak velká.

Závěr

Má diplomová práce je věnována tématu tabu prohledávání, které bylo již mnohokrát zpracováno a vzniklo pro něj mnoho metod. Zaměřila jsem se tedy jen na jednu metodu a to granulární tabu search metodu z toho důvodu, že použitím granulární struktury okolí pro prohledávání se stává jednou z nejrychlejších metod pro tabu search bez újmy na efektivitě získaných výsledků. V praxi totiž hledisko času bývá nejdůležitějším.

V prvních čtyřech kapitolách se věnuji teorii, jak té použité v praktické části práce tak obecné související s tématem VRP a tabu search. Pátá nejrozsáhlejší kapitola je věnována implementaci v Matlabu spolu s testováním parametrů použitých ve vytvořeném algoritmu.

Stěžejním cílem této diplomové práce bylo vytvoření algoritmu pro GTS metodu v softwarovém prostředí Matlab, který je představen a používán v páté kapitole. Tento algoritmus však měl být z hlediska časové náročnosti, co nejefektivnější, což nebyl snadný úkol. Nakonec ale vznikl rychlý algoritmus s několika variantami, který funguje velmi dobře pro datové soubory s počty míst menší než 55. Pro větší datové soubory by muselo být spuštěno mnohonásobně více iterací, než bylo testováno v kapitole 5.2.1., nebo by musel být algoritmus nějak vhodně doplněn o nějaký další zlepšovací nástroj. Jednou z možností, jak tento problém řešit, by mohlo být provedení 2 – opt zlepšovací heuristiky na výsledných trasách GTS algoritmu, případně, jak je popsáno v kapitole 4.2., po konkrétním větším počtu iterací, kdy se nejlepší řešení nemění, zvětšit parametr β na určitý počet iterací (po tomto počtu iterací parametr β vrátíme na původní hodnotu) a prohledávat tím pádem větší okolí, abychom se dostali z lokálního minima, ve kterém jsme uvízli.

Pomocí testů v kapitole 5.2., které byly provedeny na notebooku s procesorem Intel i7 – 4700 2,4 GHz, jsem doporučila určité nastavení parametrů, které uživatel musí zadat do funkce v Matlabu pro GTS algoritmus, tak, aby bylo dosaženo, co nejlepších výsledků. Toto nastavení však není pevně dané, protože v GTS algoritmu je obsažena náhoda, proto také spouštím algoritmus v testech desetkrát.

Na přiloženém CD jsou obsažené všechny funkce vytvořené v Matlabu a také pár skriptů, ze kterých si uživatel může spustit konkrétní datové soubory, podívat se na vypsané výsledky při `stats` = 1, případně se podívat na průběh celého výpočtu při `stats` = 2 a také na vykreslení výsledných tras.

CD Příloha

Na přiloženém CD můžete najít:

1) složku s funkcemi v prostředí Matlab, složku s konkrétními datovými soubory a také skripty pro ilustrativní výpočet GTS algoritmu;
2) tuto diplomovou práci ve formátu pdf.

Seznam obrázků

Seznam tabulek

Seznam literatury

[1] Cordeau J. F., Laporte G.: *Tabu Search Heuristics for the Vehicle Routing Problem.* Operations Research/Computer Science Interfaces Series Vol. 30 (2005), s. 145-163.

[2] Gendreau M., Hertz A., Laporte G.: *New insertion and post-optimization procedures for the Traveling Salesman Problem.* Operations Research Society of America 6/40 (1992), s. 1084-1094. Dostupné také z: http://www-2.dc.uba.ar/materias/metah/GENI_TSP.pdf.

[3] Gendreau M., Hertz A., Laporte G.: *A Tabu search heuristic for the Vehicle Routing Problem.* Management Science 10/40 (1994), s. 1276-1290.

[4] Laporte, G.: *The Vehicle Routing Problem: An overview of exact and approximate algorithms*. European Journal of Operational Research 3/59 (1992), s. 345-358.

[5] Toth, P., Vigo, D.: *The granular Tabu search and its application to the Vehicle Routing Problem.* Informs Journal on Computing 4/15 (2003), s. 333-346.

[6] Joubert J. W.: *An integrated and intelligent metaheuristic for constrained vehicle routing.* Ph.D. thesis: University of Pretoria, 2007.

[7] Larson R. C., Odoni A. R.: *Urban Operations Research.* Prentice – Hall, NJ, 1981, Chapter 6, Applications of Network Models. Dostupné také z: http://web.mit.edu/urban_or_book/www/book/chapter6/6.4.12.html.

[8] NEO – Networking and Emerging Optimization, Domovská stránka [online]. [cit. 2015-04-01]. Dostupné z: http://neo.lcc.uma.es/vrp/.

[9] Wikipedie, [online]. [cit. 2015-04-05]. Dostupné z: http://cs.wikipedia.org/wiki/Heuristick%C3%A9_algoritmy.

[10] Wikipedie, [online]. [cit. 2015-04-05]. Dostupné z: http://cs.wikipedia.org/wiki/Probl%C3%A9m_obchodn%C3%ADho_cestuj%C3%ADc%C3%ADho.

[11] [online]. [cit. 2015-04-05]. Dostupné z: http://neuron.csie.ntust.edu.tw/homework/94/fuzzy/homework/homework2/A9315016/ts-1.jpg.

[12] [online]. [cit. 2015-04-05]. Dostupné z: http://on-demand.gputechconf.com/gtc/2014/presentations/S4534-high-speed-2-opt-tsp-solver.pdf.

[13] [cit. 2015-04-15]. Dostupné z: http://web.mit.edu/16.070/www/lecture/big_o.pdf.

[14] [online]. [cit. 2015-04-26]. Dostupné z:
http://user.mendelu.cz/fisnarov/imt/prednasky/funkce.pdf.
[15] [online]. [cit. 2015-04-29]. Dostupné z:
http://www.akira.ruc.dk/~keld/research/LKH/KoptReport.pdf.