Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Programový nástroj pro plánování svozných a rozvozových tras v regionu

Bc. Zuzana Karlíková

Diplomová práce
2009

University od Pardubice
Faculty of Electrical Engeneering and Informatics

Program tool for VRP (Vehicle Routing Problem) planning in district

Bc. Zuzana Karlíková

Graduation theses
2009

Prohlašuji:
Tuto práci jsem vypracovala samostatně. Veškeré literární prameny a informace, které jsem v práci využila, jsou uvedeny v seznamu použité literatury.
Byla jsem seznámena s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 18. 5. 2009

Zuzana Karlíková

PODĚKOVÁNÍ

Tímto bych chtěla poděkovat panu Doc. Ing. Josefu Volkovi, CSc. za vedení této diplomové práce, za cenné rady a podnětné připomínky v průběhu jejího zpracování. Také bych chtěla poděkovat paní Šárce Klicperové ze společnosti SmP Odpady a.s., za poskytnuté informace týkající se svozu odpadu.

Anotace

Tato práce se zabývá problematikou optimalizace svozu komunálního odpadu. Obsahuje teoretický úvod do problematiky oblasti teorie grafů, která se řešením úloh tohoto typu zabývá. Úloha vychází z požadavků na svoz komunálního odpadu v regionu Pardubice. Jsou uvedeny současné technické prostředky pro svoz odpadu.

Výsledkem diplomové práce je programový nástroj řešící úlohu svozu komunálního odpadu. Výstupem programu jsou měsíční plány tras pro jednotlivá vozidla na základě dat zadaných do aplikace.

Klíčová slova

teorie grafů, problém obchodního cestujícího, okružní jízdy, svoz odpadu

Annotation

This thesis deals with optimization of waste collection. It contains a theoretical introduction of graph theory and describes mainly the VRP – Vehicle Routing Problem. The work is based on the requirements for waste collection in the region of Pardubice. Further there are given the technical means for waste removal.

The result of this thesis is a tool, which solves problem of waste removal. The outputs of the program are monthly plans routes for each vehicle based on the data entered in the application.

Key words

Graph Theory, Travelling Salesman Problem, Vehicle Routing Problem, Waste Collecting

Stručný obsah

Obsah

1. Úvod

Hlavními cíly Evropské unie jsou ekonomický růst a blahobyt. Ekonomický růst má však často i velmi negativní vedlejší účinek - produkuje větší množství odpadů. Se zvyšující se produkcí továren a narůstající spotřebou se zvyšuje i množství vyprodukovaného odpadu. Otázka využívání nebo odstraňování odpadu je prvořadý úkol jak z hlediska ochrany životního prostředí, tak i z hlediska ekonomického.

Ochrana životního prostředí je v dnešní době velice diskutovaný pojem. Česká republika, stejně jako Evropská unie, podporuje různé formy ekologického chování. Podle slov ministra Martina Bursíka se: „tato vláda stala nejzelenější vládou v historii ČR“. Ekologický přístup se hodně uplatňuje i v oblasti nakládání s odpady. V minulosti skončila větší část vyprodukovaného odpadu na skládkách nebo ve spalovnách. Oba tyto způsoby nakládání s odpady znečišťují životní prostředí. Skládky znečišťují vzduch, podzemní vody a půdu - do ovzduší uvolňují oxid uhličitý a metan a do půdy a podzemích vod chemikálie a pesticidy. To má škodlivý vliv nejen na zdraví lidí, ale i rostlin a zvířat. Dnešní společnost si naštěstí uvědomuje dopady svého chování na životní prostředí a tak se stále více snaží minimalizovat množství směsného odpadu. Z průzkumů vyplývá, že více než dvě třetiny obyvatel České republiky pravidelně třídí odpad, čímž umožní jeho další zpracování – recyklaci.

Třídění odpadů není ale prospěšné jen pro životní prostředí. Z hlediska ekonomiky státu je recyklace odpadů klíčová. Vytříděný odpad se dále využívá pro další zpracování. Z recyklovaných materiálů se vyrábí mnoho výrobků, proto mají o tříděný odpad zpracovatelé velký zájem. Opětovné využití materiálů nejen, že šetří přírodu, ale navíc značně snižuje náklady firem. Navíc odpad, který už se znovu zpracovat nedá, se někdy používá jako palivo a slouží tak například při výrobě elektrické energie nebo se jím vytápí.

Firmy zabývající se svozem odpadu by se měly, stejně jako řada jiných firem, otázkou ekologie také zabývat. Důležitou součástí ekologického chování je soustředit se na minimalizaci nákladů spojených s provozem jejich vozového parku. Vzhledem k tomu, že stále stoupá produkce odpadu a také proto, že firmy zabývající se svozem

odpadu, díky zavedení plošných poplatků za svoz odpadu, sváží odpad i z míst, která jsou na samotách nebo více vzdálená od centra vesnice, je nutné trasy svozu odpadu optimalizovat. Problém optimalizace svozu je možné řešit za pomocí metod operačního výzkumu.

Cílem této práce je nastudovat a rozebrat problematiku svozně-rozvozných úloh a na základě těchto znalostí vytvořit softwarový nástroj pro optimalizaci svozu komunálního odpadu. Tento nástroj bude navržen na základě informací o realizaci svozu odpadu v regionu Pardubice.

Práce bude rozdělena do několika částí. V první části bude rozebrán teoretický aparát teorie grafů vhodný pro řešení úloh svozu odpadu. Další část bude věnovaná analýze současného stavu řešení svozu komunálního odpadu v České republice. Za analýzou následuje fáze návrhu konkrétního řešení. Dále bude popsána počítačová implementace vybraného řešení. Celá práce bude zakončena zhodnocením dosažených výsledků.

2. Teoretický aparát

2.1. Základní pojmy z teorie grafů

Graf – je základní objekt teorie grafů. Skládá se z vrcholů a hran. Hrana vždy spojuje dva vrcholy. Hrana může být orientovaná (rozlišujeme počáteční a koncový vrchol) nebo neorientovaná. Hrana spojující vrchol se sebou samým se nazývá smyčkou. Graf může vyjadřovat souvislosti mezi objekty, návaznosti, spojení nebo toky.

Incidence – přiřazuje každé hraně grafu G (ne)uspořádanou dvojici vrcholů. Je-li incidence hrany $h \in X$: $p(h) = (u,v)$ říkáme, že hrana h inciduje s vrcholy u a v.

Stupeň vrcholu – číslo, rovnající se počtu hran incidujících s daným vrcholem.

2.1.1. Rozdělení grafů

Podle orientace hran

- Orientovaný graf – trojice $G = (V, X, p)$ tvořená neprázdnou konečnou množinou V, jejíž prvky nazýváme vrcholy, konečnou množinou X jejíž prvky nazýváme orientovanými hranami a zobrazením p, které zobrazuje množinu X na množinu V^2. Toto zobrazení přiřadí každé hraně $h \in X$ uspořádanou dvojici vrcholů (x,y). První z nich nazýváme počátečním vrcholem hrany h a druhý z nich nazýváme koncovým vrcholem hrany h. Zobrazení p se nazývá incidence grafu G.
- Neorientovaný graf – trojice $G = (V, X, p)$ tvořená neprázdnou konečnou množinou V, jejíž prvky nazýváme vrcholy, množinou konečnou X, jejíž prvky nazýváme neorientovanými hranami a zobrazením p, které zobrazuje množinu X na množinu V^2. Toto zobrazení přiřadí každé hraně $h \in X$ jedno- nebo dvou prvkovou množinu vrcholů Zobrazení p se nazývá incidence grafu G.

Podle četnosti hran

- Prostý graf - nemá rovnoběžné hrany, tzn. mezi libovolnými dvěma vrcholy vede nejvýše jedna hrana.

- Multigraf - mezi dvěma vrcholy může grafu vést libovolný počet hran.
- Obyčejný graf – je prostý graf neobsahující smyčky.

Podle souvislosti

- Souvislý graf – pro každou dvojici vrcholů *u* a *v* existuje cesta *m(u,v)*.
- Nesouvislý graf – neexistuje cesta mezi každou dvojicí vrcholů.

Podle existence kružnice v grafu

- Cyklický graf
- Acyklický graf

2.1.2. Definice potřebných pojmů z oblasti teorie grafů

Kružnice (cyklus) – uzavřená posloupnost propojených vrcholů.

Most – hrana grafu, jejímž odstraněním se graf rozpadne na dvě komponenty.[5]

Artikulace – vrchol grafu, jehož odstraněním z grafu (včetně incidujících hran) se zvýší počet komponent alespoň o jednu.[5]

Hamiltonovská kružnice – je podgraf grafu, který je kružnicí a obsahuje všechny vrcholy grafu. Hamiltonovskou kružnici můžeme rovněž definovat jako souvislý pravidelný graf druhého stupně, který obsahuje všechny vrcholy grafu.[5]

Hamiltonovský graf – je graf, kteým lze projít tak, že každý jeho uzel, kromě uzlu počátečního, bude navštíven právě jednou. Obsahuje kružnici, která prochází všemi vrcholy grafu – hamiltonovskou kružnici.

Kostra grafu - takový podgraf souvislého grafu *G* na množině všech jeho vrcholů, který je stromem.

Strom - neorientovaný graf, který je souvislý a neobsahuje žádnou kružnici.

Minimální kostra grafu – je kostra grafu, která má nejmenší součet ohodnocení hran mezi všemi kostrami.

Sled – sled délky k grafu G je posloupnost vrcholů a hran ($v_0, h_1, v_1, ..., h_k, v_k$) taková, že $e_i = \{v_{i-1}, v_i\}$ pro $i = 1, ..., k$.[24]

Tah - tah délky k grafu G je sled délky k grafu G takový, že platí $h_i \neq h_j$ pro každé $i \neq j, i, j \in \{1, ..., k\}$. Jedná se o sled, ve kterém se neopakují hrany. [24]

Eulerovský tah – označuje tah, který obsahuje každou hranu grafu právě jednou.[13]

Uzavřený eulerovský tah – takový tah, u kterého je počáteční a koncový uzel totožný.

Neuzavřený eulerovský tah – takový tah, který nemá totožný počáteční a koncový uzel.

Eulerův graf – graf, ve kterém existuje uzavřený eulerovský tah. Tento graf lze nakreslit „jedním tahem“.

Vlastnosti:

- Neorientovaný graf je eulerovský právě tehdy, je-li souvislý a každý jeho vrchol má sudý stupeň.
- Neorientovaný graf je eulerovský právě tehdy, je-li souvislý a má-li právě dva vrcholy lichého stupně (eulerovský tah bude otevřený).
- Neorientovaný graf je eulerovský právě tehdy, je-li souvislý a lze jej rozložit na hranově disjunktní cykly.
- Orientovaný graf je eulerovský právě tehdy, je-li souvislý a každý jeho vrchol má vstupní stupeň rovný výstupnímu.[13]

2.2. Složitost algoritmu

Při řešení úloh pomocí výpočetní techniky musíme mít nástroj, kterým dokážeme porovnat efektivitu a rychlost vykonávání jednotlivých algoritmů. Pro tyto účely existuje pojem složitost algoritmu. [6]

Pod složitostí algoritmu rozumíme časovou složitost algoritmu (dobu prováděného algoritmu) a prostorovou složitost algoritmu (rozsah použité operační paměti). Složitost závisí na velikosti vstupních dat, proto ji můžeme popsat jako funkci $f(n)$, kde číslo n udává velikost vstupních dat.

Horní odhad složitosti algoritmu nás zajímá nejvíce, protože udává složitost algoritmu v nejhorším případě. Zapisujeme ji jako $f(n) = O(g(n))$. Například časová složitost O(n) (tzv. lineární) říká, že doba trvání práce algoritmu se zvýší přibližně tolikrát, kolikrát se zvýší velikost vstupu. Někdy ovšem algoritmus dosahuje této horní složitosti jen ve vzácných případech, v takovém případě pak o složitosti algoritmu vypovídá lépe složitost průměrná.

Složitost v průměrném případě (očekávaná složitost) se počítá jako střední hodnota náhodné složitosti $T(n)$ při nějakém rozložení vstupních dat. Někdy může být i řádově lepší než složitost v nejhorším případě. Zapisujeme ji jako $f(n) = \Xi(g(n))$.

Dolní odhad složitosti nám určuje ideální složitost daného algoritmu (nejrychlejší možné provedení), která nastává jen pro určité případy vstupních dat. Zapisujeme ji jako $f(n) = \Omega(g(n))$.

Uveďme ještě tabulku ukazující délku výpočtu v závislosti na rozsahu dat pro jednotlivé složitosti. Předpokládejme dále, že vykonávané elementární operace trvají jednu milisekundu.

	Rozsah dat			
Složitost	n = 20	n = 40	n = 100	n = 1000
n	20 ms	40 ms	0,1 s	1 s
$n \cdot \log_2 n$	85 ms	0,24 s	0,66 s	10 s
n^2	0,44 s	1,6 s	10 s	16 min. 40 s
n^3	8 s	64 s	16 min. 40 s	11dní 13 hodin
2^n	17 min. 28 s	34 let 307 dní	…	…

Tabulka 1 – závislost délky výpočtu na složitosti algoritmu [zdroj: Karlíková]

Z tabulky je zřejmé, že u algoritmů, jejichž složitost je dána rychle rostoucí funkcí, se výrazně prodlužuje zpracování dat většího rozsahu.

2.2.1. Třídy složitosti

Algoritmy můžeme roztřídit do tříd podle hodnoty $f(n)$. Následující tabulka udává nejčastěji používané třídy složitosti. Hodnoty v tabulce jsou seřazeny od nejrychlejších k nejpomalejším:

Označení	*Název*
O(1)	konstantní
O(log n)	logaritmická
O(n)	lineární
O(n·log n)	lineárnělogaritmická
$O(n^2)$	kvadratická
$O(n^3)$	kubická
$O(n^x)$	polynomiální
$O(x^n)$	exponenciální
O(n!)	fatoriálová

Tabulka 2 – třídy složitosti[6]

2.2.2. Třídy složitosti P a NP

Na složitost problémů lze nahlížet různými způsoby. Jako únosné lze označit takové problémy, jejichž stavový prostor je polynomiálně velký. Neúnosné jsou pak takové, jejichž stavový prostor je nadpolynomiální. U některých existuje únosně dlouhá cesta stavovým prostorem k řešení, pokud víme, kudy jít. U některých ani to ne.

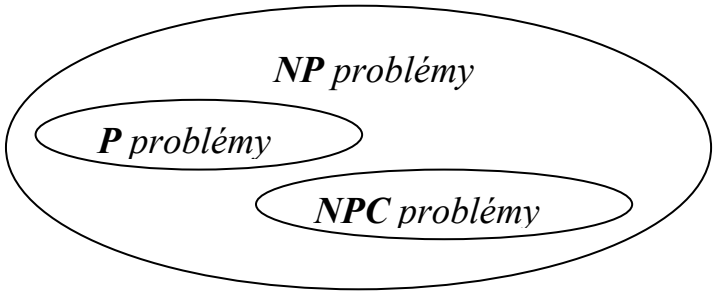
Turingův stroj je teoretický model počítače popsaný matematikem Alanem Turingem. Skládá se z procesorové jednotky, tvořené konečným automatem, programu ve tvaru pravidel přechodové funkce a pravostranně nekonečné pásky pro zápis mezivýsledků. Využívá se pro modelování algoritmů v teorii vyčíslitelnosti.

Rozhodovací problém patří do **třídy P**, jestliže pro něj existuje program pro deterministický Turingův stroj, který jej řeší v čase $O(n^k)$, kde n je velikost instance a k je konečné číslo (s polynomiální časovou složitostí).

Rozhodovací problém patří do **třídy NP,** jestliže pro něj existuje program pro nedeterministický Turingův stroj a jestliže každý jeho kladný výsledek lze verifikovat v čase $O(n^k)$, kde n je délka vstupních dat a k je konečné číslo (v polynomiálním čase).

Důležitou roli v této diskuzi hrají **NP-úplné** problémy (NPC problémy), které se nachází uvnitř třídy NP. Třídu NP-úplných problémů tvoří nejtěžší úlohy. Je to

třída problémů, pro něž se zatím nepovedlo najít polynomiální algoritmus a jejichž časová složitost je pravděpodobně exponenciální. Pokud by byl nalezen deterministický polynomiální algoritmus pro nějakou NP-úplnou úlohu, znamenalo by to, že všechny nedeterministicky polynomiální problémy jsou řešitelné v polynomiálním čase, tedy že třída NP se „zhroutí“ do třídy P (NP = P). Otázka, zda nějaký takový algoritmus existuje, zatím nebyla rozhodnuta, předpokládá se však, že NP ≠ P (je však zřejmé, že P ⊆ NP).



Obrázek 1 – NP problémy [zdroj: Karlíková]

Vztah mezi P a NP je jedním ze sedmi problémů tisíciletí, které vypsal Clay Mathematics Institute 24. května 2000, za rozhodnutí vztahu nabízí odměnu 1 000 000 dolarů.[7]

3. Východiska řešení problematiky

3.1. Metody řešení

3.1.1. Exaktní (deterministické) metody

Exaktní, deterministcké nebo také přesné metody dávají vždy optimální řešení. Jsou založeny na prozkoumání všech existujících variant. Porovnáním účelové funkce jednotlivých možností lze najít globální extrém (maximum nebo minimum). Procházení všech variant možného řešení je u rozsáhlých úloh, z hlediska časové náročnosti, značnou nevýhodou. U velmi rozsáhlých úloh není možné získat optimální řešení v reálném čase.[8]

3.1.2. Heuristické metody

Heuristickým algoritmem nazýváme postup získání řešení problému, které však nemusí být optimální a které není možné matematicky dokázat. Podstatou těchto metod je kombinace exaktních metod s lidskou inteligencí a s postupy získanými zkušenostmi a intuicí. Správnost těchto algoritmů je potvrzena dlouhodobou zkušeností a ověřením výsledků na mnoha zpracovaných příkladech.

Heuristické algoritmy neprozkoumají všechny možnosti, ale pouze vybranou část, proto jsou rychlejší a jsou dobře použitelné u složitých a rozsáhlých problémů, kde exaktní metody dosahují exponenciální složitosti a tím i nereálnou dobu výpočtu. Řešení získané těmito algoritmy je pouze suboptimální. Pro zpřesnění výsledků nalezených heuristickými algoritmy, můžeme u některých metod opakovat algoritmus.

3.2. Metody pro nalezení minimální cesty

Nejkratší cesta mezi vrcholy *u* a *v* v grafu G = *(V, X, p)* je cesta $m^*(u, v) \in M$ pro kterou platí: $\sum_{h \in m^*(u.v)} o(h) = \min_{m(u,v) \in M} \{\sum_{h \in m(u.v)} o(h)\}$, kde *M* je množina všech cest *m(u,v)* z vrcholu *u* do vrcholu *v*.

Množina sousedů – Nechť $G = (V, X, p)$ a $u \in V$. Množinou sousedů $\Gamma(u)$ vrcholu u nazýváme podmnožinu vrcholů definovanou vztahem: $\Gamma(u) = \{v \in V: \exists h \in X\ (p(h) = (u, v))\}$

3.2.1. Dijkstrův algoritmus

Algoritmus poprvé popsal nizozemský informatik Edsger Dijkstra. Je to algoritmus sloužící k nalezení nejkratší cesty v grafu mezi dvěma vrcholy nebo z daného vrcholu do všech ostatních. Je konečný, protože v každém průchodu cyklu se do množiny navštívených uzlů přidá právě jeden uzel, průchodů cyklem je tedy nejvýše tolik, kolik má graf vrcholů. Funguje nad hranově ohodnoceným, souvislým grafem $G = (V, X, p)$ kde V je množina vrcholů $V = \{v_0, v_1, \ldots, v_n\}$ a $o(h)$ je ohodnocení hrany h.[1]

Složitost algoritmu

Nejjednodušší implementace - pro uložení sousedních vrcholů používá pole a nalezení vrcholu s minimálním ohodnocením je lineární prohledávání všech vrcholů v tomto poli. V tomto případě je asymptotická časová složitost $O(|V|^2+|E|)$, kde $|V|$ je počet vrcholů a $|E|$ počet hran.

Složitější implementace (používaná pro řídké grafy - grafy s počtem hran mnohem menším než $|V|^2$) - pro uložení sousedních vrcholů používá binární nebo fibonacciho haldu. Nalezení vrcholu s minimálním ohodnocením má asymptotickou časovou složitost $O(\log_2 n)$. Asymptotická časová složitost algoritmu je v tomto případě $O((|E| + |V|)\log |V|)$.

Algoritmus

1. Zvolíme počáteční vrchol cesty u a koncový vrchol cesty v.
2. Všem vrcholům přiřadíme ohodnocení t. Pro vrchol u: $t_0 = 0$, pro ostatní vrcholy $t_i = \infty$ pro $i = 1,2, \ldots N$
3. V grafu hledáme dvojici přilehlých vrcholů $v_i, v_j \in V$ pro něž platí: $t_j - t_i > o(v_i,v_j)$

   3. a) Jestliže dvojice vrcholů existuje, potom ohodnocení t_j nahradíme ohodnocením $t'_j = t_i + o(v_i,v_j)$ a ohodnocení $t_j = t'_j$.

   3.b) Jestliže dvojice vrcholů neexistuje, přejdeme k bodu 4.

4. Hodnota t_n udává délku nejkratší cesty $m^*(u,v) \in M$: $t_n = \sum_{h \in m^*(u,v)} o(h)$

Rekonstrukce cesty:

5. Rekonstrukci zahájíme v koncovém vrcholu cesty *v*. Označíme si jej a zařadíme do množiny vrcholů již zařazených do cesty. Tuto množinu nazveme *U*.

6. Určíme množinu sousedů vrcholu *v*, nazveme ji *V* a vyloučíme z ní vrcholy již zařazené v množině *U*. Z množiny *V* vybereme vrchol pro který platí: $t_{k_i} - t_{k_{i+1}} = o(v_{k_{i+1}}, v_{k_i})$, to znamená, že rozdíl ohodnocení vrcholů se rovná ohodnocení hrany incidující s oběma vrcholy.

7. Zjistíme, jestli se předchůdce koncového vrcholu v_{k_i} shoduje s počátečním vrcholem cesty *u*. Pokud ano, přejdeme k bodu 8. Pokud ne, zařadíme vrchol v_{k_i} do množiny *U* a považujeme ho za vrchol koncový. Přejdeme k bodu 6.

8. Uvedeným postupem vznikne posloupnost vrcholů: $v = v_n = v_{k_0}$, $v_{k_1}, \dots, v_{k_r} = v_0 = u$ z níž vytvoříme hledanou minimální cestu.

3.2.2. Floydův algoritmus – algoritmus na určení distanční matice

Metoda hledání minimální cesty tímto algoritmem je vhodná pro nalezení nejkratší cesty pro všechny dvojice vrcholů. Výpočtová složitost Floydova algoritmu je vyšší než u Dijkstrova algoritmu, protože hledá nejkratší cesty mezi všemi vrcholy grafu během jednoho použití algoritmu.

Složitost algoritmu

Asymptotická časová složitost algoritmu je $O(n^3)$ – kubická složitost. Při počítání *k-té* úrovně můžeme přepsat informace vytvořené *(k - 1).* úrovní. Asymptotická paměťová složitost algoritmu je potom $O(n^2)$ – kvadratická složitost.

Algoritmus

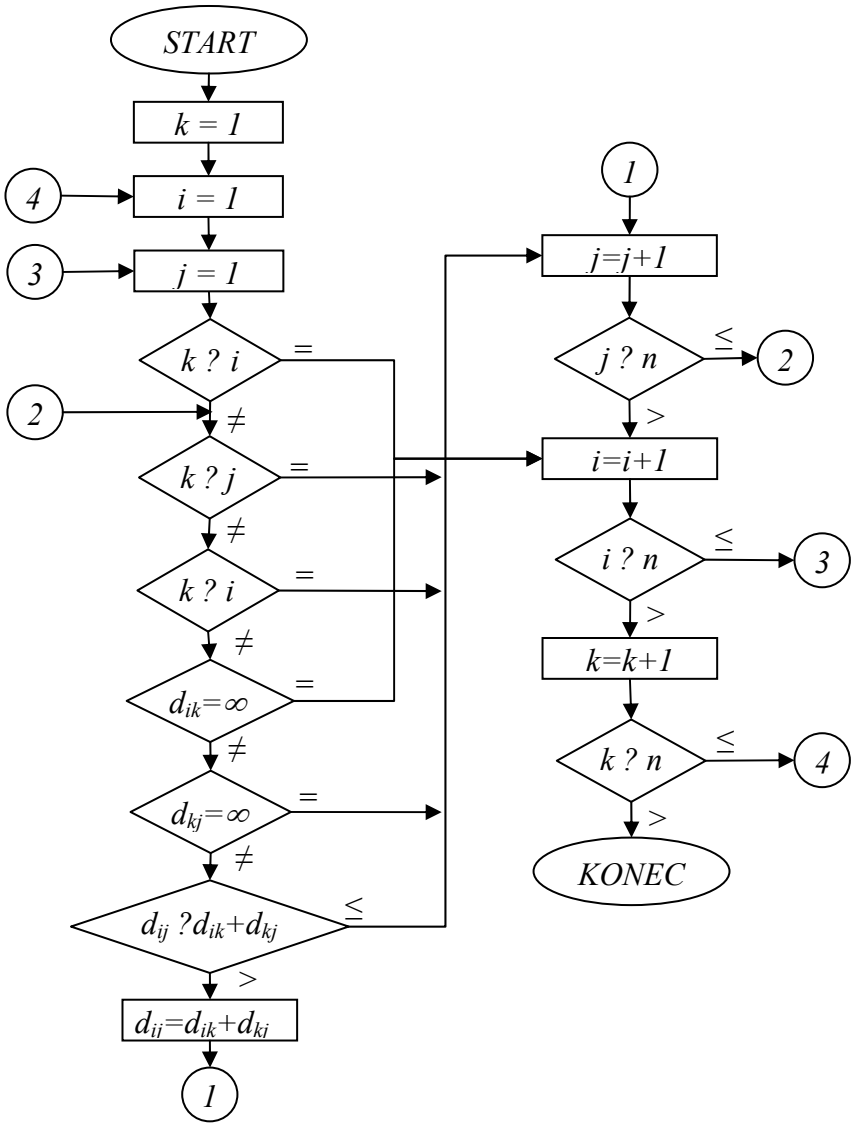
Grafu *G* přiřadíme čtvercovou matic*i* $D = (d_{ij})_{i,j=1}^{n}$, rozměru $n \times n$, kterou nazýváme maticí přímých vzdáleností. Prvky d_{ij} mohou nabývat hodnot:

a) $d_{ij} = o(h)$, jestliže $\exists h \in X$, pro kterou $p(h) = (v_i, v_j)$, $i \neq j$,

b) $d_{ij} = \infty$, jestliže $\nexists h \in X$, pro kterou $p(h) = (v_i, v_j)$, $i \neq j$,

c) $d_{ij} = 0$, pro $i = j$.

Celý algoritmus je znázorněn na obrázku 2.



Obrázek 2 – Floydův algoritmus[5]

3.3. Metody pro stanovení tras

3.3.1. Nalezení eulerovského tahu

Věta: Nechť graf G je souvislý. Pak v grafu G existuje (ne)orientovaný uzavřený eulerovský tah právě tehdy, když pro každý vrchol v platí $d^+(v) = d^-(v)$.[14] Tedy

počet hran, které do vrcholu vchází musí být roven počtu hran, které z vrcholu vychází.

Algoritmus

1. Začněme z libovolného vrcholu v_0 grafu G. Procházejme hranami grafu libovolně, ale tak, abychom žádnou hranou nešli dvakrát. Takto pokračujeme, dokud je to možné. Díky vlastnostem grafu G skončíme ve vrcholu v_0. Nalezli jsme uzavřený tah.[14]
2. Je-li tah eulerovský, algoritmus končí. Není-li tah eulerovský, pokračujeme krokem 3.
3. V grafu G existuje hrana, která v tahu neleží. Mezi v_0 a touto hranou existuje cesta a někde na této cestě existuje vrchol v_1, kterým tah prochází a z něhož vychází nějaká nepoužitá hrana. V tomto vrcholu tah rozpojíme a začneme opět procházet nepoužitými hranami a tím tah prodlužovat. Toto prodlužování opět končí ve vrcholu v_1. Navážeme novou část tahu na původní, čímž získáme uzavřený tah.[14] Přejdeme na krok 2.
4. Tento postup opakujeme, dokud existuje hrana, která v tahu neleží.

3.3.2. Problém čínského pošťáka (listonoše)

Cílem úlohy je nalézt trasu listonoše tak, aby prošel každou ulicí právě jednou, vrátil se na výchozí místo a náklady (délka cesty) byly co nejmenší.

V teorii grafů to znamená nalézt nejlevnější eulerovský tah na souvislém, neorientovaném, hranově ohodnoceném grafu.

Eulerovský tah nemusí existovat v každém grafu. Pokud takový tah v grafu neexistuje, je nutné některými hranami procházet dvakrát nebo i vícekrát. Lze však dokázat, že nejkratší sled prochází každou hranou pouze jedenkrát nebo dvakrát. Hledáme tedy sled, v němž je nejmenší součet délek hran, které jsou procházeny opakovaně.[14]

Klíčem k řešení úlohy je rozdělit vrcholy s lichým stupněm do dvojic, a to tak, aby součet délek nejkratších cest mezi vrcholy ve dvojicích byl nejmenší. Z toho vyplývá, že hledáme nejlevnější perfektní párování v (pomocném) úplném grafu K,

jehož vrcholy jsou všechny vrcholy lichého stupně původního grafu *G* a délky hran grafu *K* jsou délky nejkratších cest v grafu *G*. [14]

Algoritmus

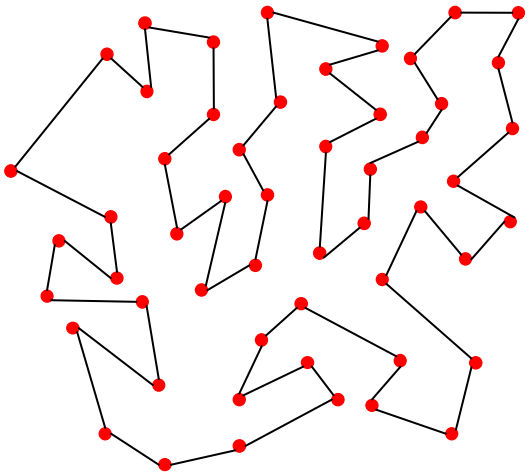
1. V grafu *G* najdeme množinu vrcholů *L* s lichým stupněm.
2. Pro každou dvojici vrcholů *(u, v)* z množiny *L* spočítáme délku nejkratší cesty z vrcholu *u* do vrcholu *v*. Tuto délku označíme *l(u, v)*.
3. Na množině *L* definujeme úplný graf *K*. Hrany tohoto grafu jsou ohodnoceny délkami *l(u, v)*.
4. V tomto grafu najdeme nejlevnější perfektní párování *P* (tzv. přiřazovací problém – řešení např. maďarským algoritmem viz. níže).
5. Pro každou dvojici vrcholů *u* a *v*, která je spojena hranou z párování *P*, přidáme ke grafu *G* kopie hran, které tvoří nejkratší cestu mezi vrcholy *u* a *v*. Získáme tak graf *G‘,* který má všechny vrcholy sudého stupně.
6. V grafu *G‘* sestrojíme eulerovský tah. Tento tah prochází všemi přidanými hranami. Nahradíme-li přidané hrany původními hranami, získáme nekratší hledaný sled, který pokrývá všechny hrany grafu.

3.3.3. Problém obchodního cestujícího

Problém obchodního cestujícího (TSP – **T**ravelling **S**alesman **P**roblem) je obtížný diskrétní optimalizační problém, matematicky vyjadřující a zobecňující úlohu nalezení nejkratší možné cesty procházející všemi zadanými body na mapě.[2]

3.3.3.1. Formulace úlohy obchodního cestujícího

Existuje ohodnocený souvislý neorientovaný graf, v němž vrcholy představují nějaká města a hrany silniční spoje mezi nimi. Ohodnocení vyjadřuje délku příslušného silničního spojení. Obchodní cestující má vyjet z jednoho města, projet všemi ostatními městy právě jednou a vrátit se do výchozího místa, tak aby délka jeho cesty byla co nejkratší (náklady na cestu byly co nejmenší). Situaci demonstruje obrázek číslo 3, kde je znázorněná trasa obchodního cestujícího.[4] Zabývat se budeme symetrickou variantou úlohy, kdy náklady na cestu z bodu A do bodu B jsou stejné jako náklady na cestu z bodu B do bodu A.



Obrázek 3 – trasa obchodního cestujícího [zdroj: Karlíková]

Cílem této úlohy je najít tu hamiltonovskou kružnici (cyklus), jejíž délka, která je vyjádřena součtem ohodnocení hran v ní, je nejmenší ze všech hamiltonovských kružnic v grafu. Tato hamiltonovská kružnice je zároveň i nejkratší trasou pro obchodního cestujícího a je tedy řešením úlohy obchodního cestujícího.

Nalézt hamiltonovskou kružnici v daném grafu je obecně velice obtížné a efektivně zvládnutelné jen pro malé grafy. Čím více má graf při daném počtu uzlů hran, tím větší je naděje, že bude hamiltonovský. Nejjednodušší situace je u úplného grafu - úplný graf (s alespoň třemi uzly) je vždy hamiltonovský.

Dosud totiž není známa žádná jednoduchá nutná a postačující podmínka k tomu, aby graf byl hamiltonovský.[18]

Nutnou podmínkou pro existenci hamiltonovské kružnice je, že nesmí obsahovat most, artikulaci a vrcholy stupně 1. Splnění nutné podmínky není postačující k existenci hamiltonovské kružnice.[5]

Postačující podmínky existence hamiltonovské kružnice (nejsou nutnými podmínkami, co znamená, že z jejich neplatnosti nevyplývá pro existenci hamiltonovské kružnice žádné tvrzení):[5]

- Uvažujme graf $G = (V, X)$, kde $|V| = p \geq 3$, pro jehož dva různé vrcholy $u, v \in V$ platí vztah: $st(u) + st(v) \geq p$, potom graf $G = (V, X)$ obsahuje hamiltonosvkou kružnici, je hamiltonovský.[5]

- Nechť v obyčejném grafu G = (V, X) s $p \geq 3$ pro každý vrchol $v \in V$ platí: $st(v) \geq p/2$, potom graf $G = (V, X)$ obsahuje hamiltonosvkou kružnici, je hamiltonovský.[5]

3.3.3.2. Složitost algoritmu

Asymptotická časová složitost přesného řešení problému obchodního cestujícího (či jiného NP-úplného problému) je $O(2^n)$.

3.3.3.3. Milníky v řešení problému obchodního cestujícího

První úlohu obchodního cestujícího vyřešil v r. 1954 G. Dantzig spolu s R. Fulkersonem a S. Johnsonem, kteří vyřešili problém pro 49 měst USA. Počet vrcholů, pro které byl tento problém úspěšně vyřešen, se v květnu roku 2004 vyšplhal až na 24 978 švédských měst. Optimální trasa obchodního cestujícího je v tomto případě přibližně 72 500 km. Ostatní milníky jsou uvedeny v tabulce 3.

Rok	***Řešitelé***	***Počet vrcholů***
1954	G. Dantzig, R. Fulkerson, S. Johnson	49
1971	M. Held, R.M. Karp	64
1975	P.M. Camerini, L. Fratta, F. Maffioli	67
1977	M. Grötschel	120
1980	H. Crowder, M.W. Padberg	318
1987	M. Padberg, G. Rinaldi	532
1987	M. Grötschel, O. Holland	666
1987	M. Padberg, G. Rinaldi	2 392
1994	D. Applegate, R. Bixby, V. Chvátal, W. Cook	7 397
1998	D. Applegate, R. Bixby, V. Chvátal, W. Cook	13 509
2001	D. Applegate, R. Bixby, V. Chvátal, W. Cook	15 112
2004	D. Applegate, R. Bixby, V. Chvátal, W. Cook, K. Helsgaun	24 978

Tabulka 3 – milníky v řešení úlohy obchodního cestujícího[20]

3.3.3.4. Metody řešení

Exaktních či heuristických metod pro řešení problému obchodního cestujícího existuje celá řada. V této kapitole bude popsáno několik základních metod.

Výpočet hrubou silou

Výpočet hrubou silou je nejznámější a nejjednodušší exaktní metoda, která dává přesné řešení. Spočívá v procházení všech permutací vrcholů a výpočtu délky trasy pro každou permutaci. Nejkratší z těchto permutací je řešením problému obchodního cestujícího. Pro graf se čtyřmi vrcholy je těchto možností *P(4) = 24*. Pokud nám nezáleží na směru cesty, můžeme počet možností zredukovat na polovinu. V případě takto malého počtu vrcholů je výpočet snadno realizovatelný. Složitost výpočtu ale stoupá s každým přidaným vrcholem. Například pro 10 vrcholů je počet možností již *P(10) = 3 628 800*. Toto číslo můžeme opět zredukovat na polovinu, pokud nezáleží na směru cesty obchodního cestujícího. Pro 15 vrcholů je počet možností P(15) = 1 307 674 368 000. Z výsledků je vidět, že nárůst počtu možností je velice dramatický a pro grafy s větším počtem vrcholů je tento postup nerealizovatelný.

Lineární programování

Lineární programování (LP) patří také mezi exaktní optimalizační metody. Je založeno na hledání maxima či minima jedné cílové funkce v prostoru přípustných řešení, jenž je vymezen množinou omezujících podmínek. Pokud jsou cílová funkce i omezující podmínky lineární, jedná se o lineární programování. Úloha LP se skládá z následujících tří částí:

- Účelová funkce: $\min_{x \in M} c^T x$, kde $c^T x$ označuje skalární součin vektorů *c* a *x*, *M* je množina přípustných řešení.
- Omezující podmínky: $Ax \leq b$, kde *A* je matice ve tvaru *m*×*n*, *b* je *m*-rozměrný vektor, *x* jsou *n*-rozměrné vektory.
- Podmínky nezápornosti: $x \geq 0$.

Nejznámější algoritmus pro řešení úloh lineárního programování je simplexová metoda.

Celočíselné programování

Celočíselné programování je speciálním typem lineárního programování, které doplníme o podmínky celočíselnosti: $x \in Z$.

Model úlohy obchodního cestujícího se dá zapsat jako model úlohy celočíselného programování, kde *n* je počet obsluhovaných vrcholů, c_{ij} jsou náklady na cestu z vrcholu *i* do vrcholu *j* a x_{ij} je bivalentní proměnná, která vyjadřuje spojení vrcholů v_i a v_j. Preměnná $x_{ij} = 1$, pokud obchodní cestující jde z vrcholu v_i do vrcholu v_j (hrana (v_i, v_j) je zařazena do cesty, jinak $x_{ij} = 0$).

$$\min \sum_{i=1}^{n}\sum_{j=1}^{n} x_{ij} \cdot c_{ij}$$

$$\text{za podmínek:} \quad \sum_{i=1}^{n} x_{ij} = 1; j = 1, 2, \dots, n$$

$$\sum_{j=1}^{n} x_{ij} = 1\,; i = 1, 2, \dots, n$$

$$x_{ij} \in \{0,1\}$$

První omezující podmínka znamená, že obchodní cestující vstoupí do každého uzlu právě jednou, druhá podmínka znamená, že obchodní cestující vystoupí z každého uzlu právě jednou.

Pro úlohu obchodního cestujícího je nutné stanovit ještě podmínku zaručující, že hamiltonovská kružnice nebude uzavřena dříve, než bude obsahovat všechny vrcholy:

$$u_i - u_j + (n-1) \cdot x_{ij} \leq n - 2;\ i, j = 1, 2, \dots, n;\ i \neq j$$

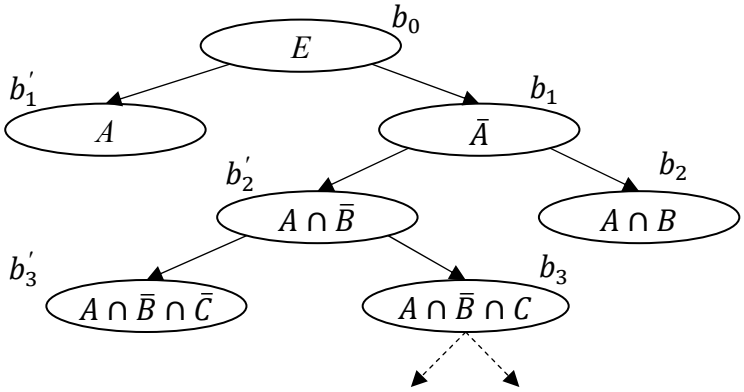
Proměnná $u_i = q$ vyjadřuje, že obchodní cestující navštívil uzel *i* po *q-tém* přesunu.[8]

Metoda Větve a hranice (Branch & Bound)

Tato exaktní metoda je založena na postupném rozkladu množiny přípustných řešení $E = \{S_1, S_2, \dots, S_n\}$ účelové funkce *f* vždy na dvě disjunktní podmnožiny *A* a $\bar{A}$ za pomocí nějaké vlastnosti P_A, přičemž $A \cap \bar{A} = \{\}$, $A \cup \bar{A} = E$.

Pro množinu *E* nalezneme dolní hranici b_0 účelové funkce *f*. Dále stanovíme i spodní hranice $b_1 \geq b_0$ a $b_1' \geq b_0$ funkce *f* na podmnožině *A*, resp. $\bar{A}$. Podle těchto hodnot určujeme, která podmnožina bude spíše obsahovat hledané optimální řešení, tedy ve kterém z visících vrcholů budeme pokračovat ve větvení.

Množinu přípustných řešení *E* tvoří kořen stromu řešení. Postupně vytváříme strom řešení, kde každá podmnožina tvoří vrchol stromu. Tímto způsobem vytváříme tzv. *prastrom řešení*. Příklad prastromu řešení je znázorněn na obrázku 4.



Obrázek 4 – prastrom řešení [zdroj: Karlíková]

Littlův algoritmus (vyhledání minimální hamiltonovské kružnice)

Exaktní metoda sloužící pro nalezení minimálních (maximálních) hamiltonovských kružnic grafu. Přesnou metodu optimalizace publikoval v roce 1963 Little, který v algoritmu využívá principu metody větve a hranice.

Littlův algoritmus můžeme použít na orientované nebo neorientované hranově ohodnocené grafy. Graf musí být souvislý, obyčejný a musí obsahovat alespoň tři vrcholy. Vychází z matice sazeb. Matice sazeb $V = (v_{ij})_{i,j=1}^{n}$, kde hodnoty v_{ij} odpovídají ohodnocení hrany $h=(v_i,v_j)$. Ohodnocení hrany může nabývat hodnot $v_{ij} = d$, kde *d* je délka hrany mezi vrcholy v_i a v_j, nebo $v_{ij} = \infty$, pokud mezi vrcholy v_i a v_j neexistuje hrana.

1. V každém řádku matice *V* vyhledáme minimální prvek. Tento prvek odečteme od všech prvků v uvedeném řádku. Výsledkem tohoto postupu je vytvoření matice V', kde $v'_{ij} = v_{ij} - min_j\{v_{ij}\}$, pro *i = 1,2, ..., n*. (vznikne nám alespoň jedna nula v každém řádku).

2. V každém sloupci matice V', v němž se nenachází žádná nula, vyhledáme minimální prvek a ten odečteme od všech prvků v příslušném sloupci. Výsledkem je vytvoření matice V'', kde $v''_{ij} = v'_{ij} - min_i\{v'_{ij}\}$, pro $j = 1,2, \ldots, n$. (vznikne nám alespoň jedna nula v každém sloupci).

3. Pokud se jedná o první průchod algoritmu, vytvoříme kořen prastromu řešení úlohy E a přiřadíme mu hodnotu b_0. Hodnota b_0 je dolní odhad účelové funkce, který vypočítáme jako součet všech hodnot, o které jsme snižovali vzdálenosti v příslušných řádcích, resp. sloupcích: $b_0 = \sum_{i=1}^{n} min_j\{v_{ij}\} + \sum_{j=1}^{n} min_i\{v'_{ij}\}$. Dolní odhad účelové funkce udává hodnotu, pod kterou hodnota účelové funkce určitě neklesne, tedy, že žádná hamiltonovská kružnice grafu nebude mít menší hodnotu než b_0.
4. Ohodnotíme každou nulu v matici V'' číslem γ_{ij}tak, že v příslušném řádku a sloupci vyhledáme minimální ze zbylých prvků a obě hodnoty sečteme (právě ohodnocovanou nulu do výpočtu nezahrnujeme): $\gamma_{ij} = \min_{r \neq j}\{v''_{ir}\} + \min_{s \neq i}\{v''_{sj}\}$.
5. Vyberme nulu s maximálním ohodnocením (existuje-li v matici více nul se stejným maximálním ohodnocením, vybereme libovolnou z nich): $\gamma_{kl} = \min_{ij}\{\gamma_{ij}\}$. Vybrané pole (v_k,v_l) znamená, že vznikající Hamiltonovská kružnice bude obsahovat hranu (v_k,v_l).
6. V aktuální matici zrušíme ohodnocení zbylých nul. Rozšíříme prastrom o vrchol s vlastností $\bar{P}_{kl}$ (znázorňuje situaci, kdy hamiltonovská kružnice nebude obsahovat hranu (v_k,v_l)). Tento vrchol ohodnotíme tak, že k ohodnocení předchůdce přičteme γ_{kl}. Prastrom rozšíříme o vrchol s vlastností P_{kl} (hamiltonovská kružnice bude obsahovat hranu (v_k,v_l)).
7. Redukujeme matici V'' vyloučením *k-tého* řádku a *l-tého* sloupce. Prvky, které by umožnily vznik nepřípustného řešení, tedy vznik hamiltonovské kružnice menší délky než počet prvků, položíme rovny ∞.
8. Opakujeme 1. a 2. krok.
9. Opakujeme 3. krok.
10. Pokud má aktuální matice rozměr 1x1 algoritmus končí.
11. Z visících vrcholů vybereme vrchol s nejmenším ohodnocením (je-li těchto vrcholů více, vybereme libovolný z nich).

12. Jestliže vybraný vrchol odpovídá posledně uvažované vlastnosti P_{kl}, přejdeme na 4. krok.
13. Mohou nastat dvě možnosti:
    a) Vybraný vrchol odpovídá vlastnosti $\bar{P}_{ij}$. V matici odpovídající této vlastnosti změníme hodnotu v''_{ij} na ∞. V *i-tém* řádku určíme minimální prvek, který odečteme od všech prvků v daném řádku. Stejný postup opakujeme i pro *j-tý* sloupec. Přejdeme na 4. krok.
    b) Vybraný vrchol odpovídá vlastnosti P_{ij}. Pokračujeme 4. krokem s maticí odpovídající této vlastnosti.

Algoritmus nejbližšího souseda

Algoritmus nejbližšího souseda (Nearest neighbor search) je heuristický optimalizační problém hledání nejbližšího bodu. Byl jedním z prvních algoritmů používaných k nalezení řešení problému obchodního cestujícího (nalezení hamiltonovské kružnice grafu).

Tento algoritmus je rychlý a lze ho velice snadno implementovat. Nedává ale velmi dobré výsledky. Vzdálenosti na začátku cesty budou krátké, ale v závěru se mohou velice prodlužovat. Má několik variant např.: Algoritmus k-nejbližších sousedů, Approximate nearest neighbor (Přibližně nejbližší soused), Nearest neighbor distance ratio (Nejbližší soused s ohledem na poměr vzdáleností). Pro problém obchodního cestujícího se ale využívá nejjednodušší varianty algoritmu, kde hledáme nejbližšího souseda ze všech vrcholů, které ještě nebyly zařazeny do cesty.

Algoritmus

Pracuje se dvěma množinami uzlů. Množina $U = \{v_1, v_2, ..., v_k\}$ je množina uzlů, které jsou do trasy zařazeny. Množina W = V – U je množina uzlů, které do trasy ještě nejsou zařazeny. Uzel v_q je do množiny již zařazených vrcholů zařazen na základě podmínky: $c_{kq} = \min_{j \in W}\{c_{kj}\}$, do množiny V je tedy zařazen ten vrchol, který je nejblíže posledně zařazenému vrcholu v_k. Na začátku algoritmu je nutné zvolit prvotní uzel.

Násobný algoritmus nejbližšího souseda

Tento algoritmus je založen na n opakování algoritmu nejbližšího souseda pokaždé s jiným počátečním uzlem. Je třeba si pamatovat nejlepší nalezenou trasu a její ohodnocení.

Algoritmus postupného zvětšování trasy

Heuristický algoritmus, který vychází z postupného zvětšování prvotní trasy složené z vrcholů v_0, v_1, v_2, v_0. V každém iteračním kroku je tato trasa zvětšena o další uzel, který je vsunut mezi dva uzly, které jsou již v trase zařazené. V trase hledáme uzel v_s, pro který bude výraz $c(v_s, v_j) + c(v_j, v_{s+1}) - c(v_s, v_{s+1})$ minimální pro všechny uzly v_j, které dosud nejsou zařazeny v trase (uzly jsou do trasy zařazovány na to místo mezi uzly v_s a v_{s+1}, kde vsunutím uzlu v_j vznikne co nejmenší prodloužení trasy).

Na začátku algoritmu je nutné rozhodnout, které vrcholy budou vybrány do prvotní trasy.

3.3.4. Okružní jízdy (vícenásobný problém obchodního cestujícího)

Problematika okružních jízd (VRP – **V**ehicle **R**outing **P**roblem) je zejména v silniční dopravě velmi aktuální. Společnosti zabývající se svozem nebo rozvozem chtějí optimalizovat jízdy jednotlivých vozidel s cílem minimalizovat náklady na dopravu.

3.3.4.1. Formulace svozně rozvozných úloh

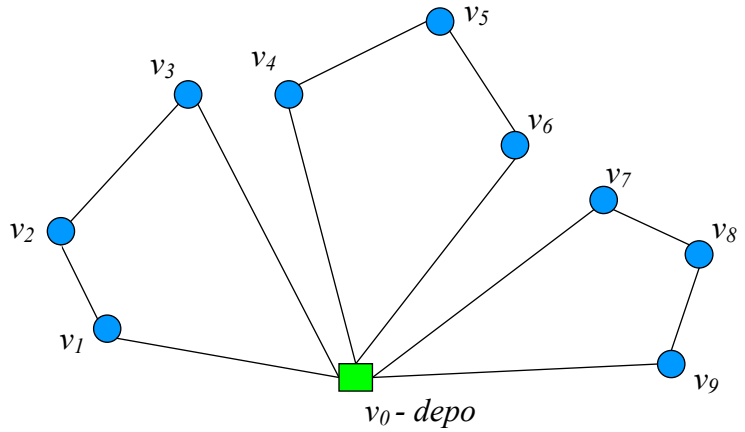
Formulace svozné úlohy – jde o úlohu, kdy je třeba z ostatních vrcholů sítě $v_1, v_2,...,v_n$ (obsluhované vrcholy) svézt požadované množství výrobků, materiálu apod. do centrálního depa (skladu) v_0.

Formulace rozvozné úlohy – jde o úlohu, kdy je třeba z centrálního skladu v_0 rozvézt požadované množství výrobků, materiálu, substrátu apod. do ostatních vrcholů sítě $v_1, v_2,...,v_n$ (obsluhované vrcholy).

Formulace svozně – rozvozné úlohy - jde o kombinaci dvou předchozích problémů, tedy o úlohu, kdy je třeba z centrálního skladu v_0 rozvézt požadované množství výrobků, materiálu, substrátu apod. do ostatních vrcholů sítě $v_1, v_2,...,v_n$ (obsluhované vrcholy), a zároveň je třeba z ostatních vrcholů sítě $v_1, v_2,...,v_n$ (obsluhované vrcholy) svézt požadované množství výrobků, materiálu apod. do centrálního depa (skladu) v_0.

K dispozici je určitý počet nákladních vozidel m se známou kapacitou c. Každé vozidlo vyjíždí z centrálního skladu (depa) a po obsluze jednoho nebo více pobočných skladů se vrací zpět do depa v_0. Uskladňovací kapacitu vrcholu v_j označíme c_j. Rozvoz (svoz) do (z) pobočných skladů bude realizován tak, aby jízdy jednotlivých automobilů netvořily navzájem se prolínající cykly, to znamená, aby jízdy jednotlivých vozidel, které tvoří cykly začínající a končící ve vrcholu v_0, byly vrcholově a hranově disjunktní množiny. Cílem řešení úlohy je stanovit plán rozvozu (svozu) tak, aby celkové náklady na rozvoz (svoz) byly minimální. To znamená určit trasu pro každé z vozidel tak, aby se minimalizovaly náklady, byla uspokojena poptávka v každém vrcholu a současně nebyla překročena povolená kapacita vozidel (u svozně – rozvozné úlohy je třeba dbát na to, aby množství nakládaného zboží a množství zboží, jež je potřeba rozvést, nepřesáhlo kapacitu vozidla).

Situace je znázorněna na obrázku 5. Předpokládáme kapacitu nákladních aut $c = 3$ a kapacitu ve vrcholech $c_i = 1$ pro $i=1,...,n$.



Obrázek 5 – okružní jízdy [zdroj: Karlíková]

3.3.4.2. Klasifikace úloh okružních jízd

Problém okružních jízd je velice rozsáhlý a komplexní. K úloze je možné vytvořit několik různých modifikací.

Rozdělení dle počtu dep

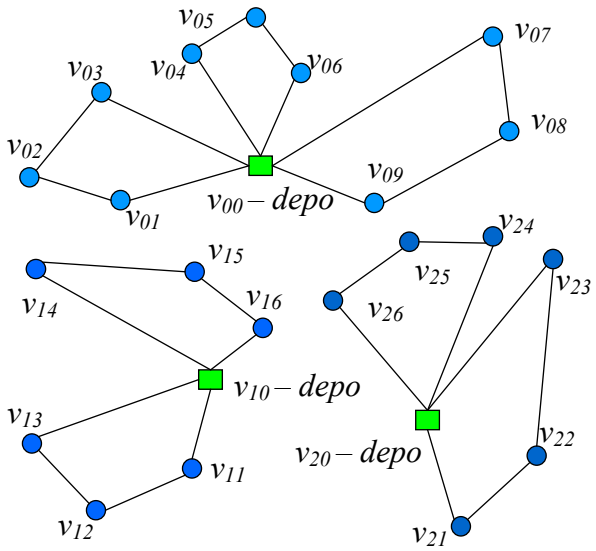
- Jedno depo

Pevně daný počet vozidel stejné kapacity vyjíždí z jednoho depa a musí obsloužit *n* vrcholů. Účelem je minimalizovat počet vozidel a počet najetých kilometrů, přičemž celková kapacita každé trasy nesmí přesáhnout kapacitu vozidla, které trasu obsluhuje. Situace je znázorněna na obrázku 5.

- Více dep

Společnosti mohou mít více dep, ze kterých vykonávají obsluhu svých zákazníků. Při řešení tohoto problému se předpokládá, že jednotliví zákazníci jsou přiřazeni k jednotlivým depům. V každém depu je umístěn určitý počet vozidel a každé vozidlo vyjíždějící z jednoho depa, obslouží jednoho nebo více zákazníků a vrací se zpátky do původního depa. Účelem je opět minimalizovat počet aut a počet najetých kilometrů.

Situace je nastíněna na obrázku 6. Počet dep $m = 3$. Kapacitu nákladních aut $c_i = 3$ kde $i = 1, 2, 3$. Kapacitu ve vrcholech $c_{ij} = 1$ pro $i=1,2,3; j=1, \ldots, n$.



Obrázek 6 - okružní jízdy - více dep [zdroj: Karlíková]

Rozdělení dle požadavků

- Jeden typ požadavků

Zákaznící (vrcholy), kteří mají být obslouženi, mají stejný typ požadavku. Všechna vozidla jsou také stejného typu. Každý z vrcholů může být obsloužen libovolným vozidlem.

- Více typů požadavků

Zákaznící (vrcholy), kterří mají být obslouženi, mají různé typy požadavků. Pro přepravu požadavku typu *T* je třeba použít vhodné vozidlo – vozidlo typu *T*. Kapacita vozidel pro přepravu stejného typu požadavku je stejná. Požadavky v jednotlivých vrcholech jsou uspokojeny pouze jedním vozidlem a každý vrchol s požadavkem *T* je navštívem vozidlem typu *T* právě jednou.

Rozdělení dle frekvence obsluhy

- Stejná frekvence obsluhy požadavků

V tomto případě je frekvence obsluhy všech zákazníků stejná. Tato frekvence bývá obvykle jeden den. Každý den musí být obslouženi všichni zákazníci.

- Různá frekvence obsluhy požadavků

U každého zákazníka je frekvence obsluhy požadavku jiná. Při plánování okružních jízd je třeba brát v potaz, jakou frekvenci obsluhy má každý zákazník. Zde rozlišujeme dva typy úloh. Úlohy kde je frekvence pevně daná, např.: denně, týdně, měsíčně a úlohy kdy si frekvenci určuje sám zákazník, není pevně daná.

Speciální typy úloh

Rozvozní (svozní) úloha s časovými okny – každý zákazník má určeno časové období, kdy je nutné provést obsluhu požadavku.

3.3.4.3. Metody řešení

Celočíselné programování

Model úlohy okružních jízd se dá zapsat jako úloha celočíselného programování. Model úlohy s jedním střediskem obsluhy, kde n je počet obsluhovaných vrcholů, c_{ij} jsou nálady na cestu mezi vrcholy v_i a v_j. Proměnná x_{ij} určuje zda je hrana (v_i, v_j) zařazena do trasy. Pokud ano $x_{ij} = 1$ jinak $x_{ij} = 0$.

$$\min \sum_{i=0}^{n} \sum_{j=0}^{n} x_{ij} \cdot c_{ij}$$

$$\text{za podmínek:} \quad \sum_{i=1}^{n} x_{i0} = A$$

$$\sum_{j=1}^{n} x_{0j} = A$$

$$\sum_{i=1}^{n} x_{ij} = 1; i = 1,2,\dots,n$$

$$\sum_{j=1}^{n} x_{ij} = 1; j = 1,2,\dots,n$$

První dvě omezující podmínky zabezpečují, že z depa vyjede a zpátky do depa vjede právě *A* vozidel. Druhé dvě podmínky zabezpečují, že do každého uzlu vjíždí a z každého uzlu vyjíždí právě jedno vozidlo.

Pro velkou náročnost při řešení úloh většího rozsahu se tato úloha metodami lineárního programování neřeší.

Výpočet hrubou silou

Spočívá (stejně jako u úlohy obchodního cestujícího) v procházení všech permutací vrcholů. Na rozdíl od úlohy obchodního cestujícího, musí být tyto permutace ještě rozděleny do tras podle kapacity. Pro každou takto rozdělenou permutaci se spočítají náklady. Permutace s nejmenšími náklady je řešením. Tento způsob řešení ale není vhodný pro úlohy většího rozsahu (viz. kapitola 3.3.3.).

Clark Wright algoritmus – Metoda určování okružních jízd

Nejznámější heuristický algoritmus, který se používá pro řešení okružních jízd. Složitějším případem okružních jízd je situace, kdy máme k dispozici více dep s různým počtem vozidel. K řešení těchto úloh s více depy je pak výhodné použít algoritmus autorů Tillmana a Caina.

Algoritmus předpokládá síť vrcholů P_0 až P_n. P_0 je centrální sklad (depo), kde je umístěn dostatečný počet vozidel. P_1 až P_n jsou obsluhované vrcholy. Všechna vozidla mají stejnou kapacitu *c*. Každé vozidlo vyjíždí z depa a po obsluze vrcholů se

vrací zpět do depa. Každý vrchol je obsloužen jedním vozidlem. Náklady na přepravu z vrcholu P_i do P_j jsou d_{ij}. Požadavek na obsluhu vrcholu P_i je c_i. Každý vrchol je obsloužen jedním vozidlem.

Výchozí situace: Každý vrchol je obsluhován jedním vozidlem vyjíždějícím z depa a vracejícím se do depa zpět. Tato výchozí situace je dále upravována a obsluhované vrcholy jsou podle hodnoty úspor spojovány do tras. Tím se snižují náklady na obsluhu vrcholů a zároveň se snižuje potřebný počet vozidel. Při každém spojení tras je nutné kontrolovat, zda požadavky ve vrcholech nepřevyšují kapacitu vozidla.

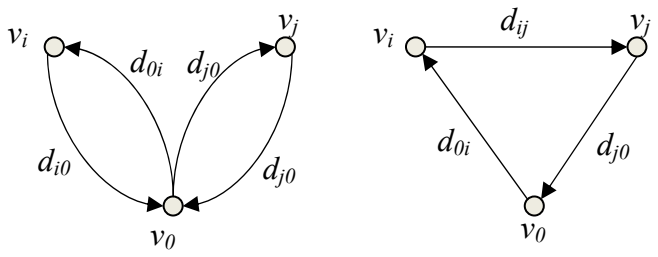
Tento algoritmus patří do kategorie heuristických metod. Ty oproti exaktním metodám, které spočívají v prověření všech variant řešení, vypočtení hodnoty kritéria pro každou z nich a výběru optimálního řešení, se u metod heuristických nezjišťují všechny varianty řešení. Obzvláště u složitých úloh pak může dojít k situaci, že získané řešení není optimální, ale pouze suboptimální, většinou je však optimálnímu řešení velmi blízké. Pro složité úlohy by ale bylo použití exaktních postupů neefektivní, proto se obvykle dává přednost heuristickým metodám a postupům.

Složitost algoritmu

Algoritmus vyniká dobrou asymptotickou časovou složitostí $O(n^2 \cdot log(n))$.

Algoritmus

1. Pro všechna $i,j = 1,2, \ldots,n$ určit hodnoty $\lambda_{ij} = d_{i0} + d_{0j} - d_{ij}$. Hodnoty λ_{ij} vyjadřují úspory vzniklé spojením vrcholů v_i a v_j do jedné trasy. Situaci demonstruje obrázek číslo 7, kde v levé části obrázku vidíte výchozí řešení, pro které jsou náklady rovny $d_{i0} + d_{0i} + d_{j0} + d_{0j}$. V pravé části je zobrazeno spojení dvou vrcholů do jedné trasy. Náklady jsou rovny $d_{0i} + d_{ij} + d_{j0}$. Prvky d_{ij} vyjadřující délku nejkratší cesty mezi vrcholem v_i a v_j, kterou získáme z distanční matice.



Obrázek 7 – znázornění úspor sloučením dvou tras do jedné[23]

Hodnoty λ_{ij} sestavíme do přehledné tabulky.

	v_0	v_1	v_2	v_3	...	v_n
v_0		d_{01}	d_{02}	d_{03}	...	d_{0n}
v_1	d_{10}		λ_{12}	λ_{13}	...	λ_{1n}
v_2	d_{20}	λ_{21}		λ_{23}	...	λ_{2n}
v_3	d_{30}	λ_{31}	λ_{32}		...	λ_{3n}
...	...	...	...	...		...
v_n	d_{n0}	λ_{n1}	λ_{n2}	λ_{n3}	...	

Tabulka 4 – tabulka úspor[22]

V první iteraci hodnoty v tabulce odpovídají výchozímu řešení, kdy je každý vrchol obsluhován jedním vozidlem. Řešení obsahuje *n* triviálních cyklů: $\{\{v_0, v_1, v_0\}, \{v_0, v_2, v_0\}, \dots, \{v_0, v_n, v_0\}\}$.

2. Označíme *S* množinu všech orientovaných hran současného řešení. Pro současné řešení určíme hodnoty γ_i:
   - $\gamma_i = 0$, jestliže hrana $[v_0, v_i] \wedge [v_i, v_0]$ nepatří do současného řešení
   - $\gamma_i = 1$, jestliže hrana $[v_0, v_i] \vee [v_i, v_0]$ patří do současného řešení

   Hodnota γ_i udává, zda vrchol v_i leží uprostřed nebo na kraji trasy. Má-li vrchol hodnotu $\gamma_i = 1$, nelze k němu připojit další vrchol, protože tento vrchol již leží uprostřed trasy. Pro výchozí řešení $\gamma_i = 1$, pro $i = 1,2,\dots,n$.

   Označíme $\{v_0, v_{k_1}, v_{k_2}, \dots, v_k, v_0\}$ množinu skladů obsluhovaných jedním automobilem v aktuálním řešení. Celkovou kapacitu v tomto cyklu označíme $Q\{v_0, v_{k_1}, v_{k_2}, \dots, v_k, v_0\}$. Nechť sklady v_i a v_j patří do dvou různých cyklů daných množinami: $\{v_0, v_i, v_{k_2}, \dots, v_k, v_0\}$ a

$\{v_0, v_j, v_{l_2}, \dots, v_l, v_0\}$.[22] K vytvoření nové, delší okružní jízdy spojením dvou kratších je zapotřebí splnění podmínek:

- *α)* $\gamma_i = 1$ a $\gamma_j = 1$
- *β)* $Q\{v_0, v_i, v_{k_2}, \dots, v_{k_r}\} + Q\{v_0, v_j, v_{l_2}, \dots, v_{l_s}\} \leq c$

Za předpokladu $\lambda_{ij} > 0$ a splnění podmínek α a β mohou nastat 3 situace:

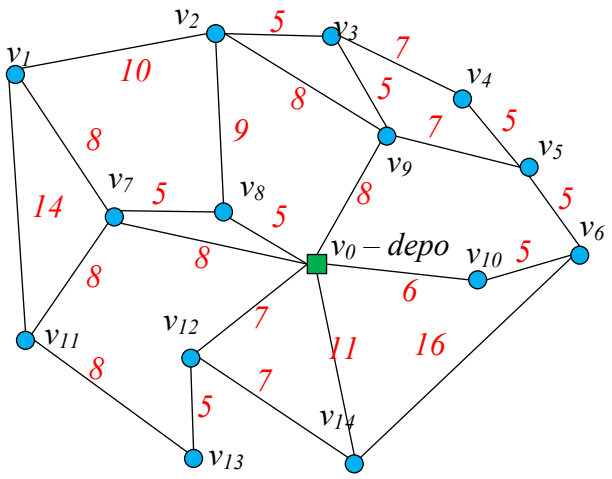
a) Sklad v_i je obsluhován samostatnou jízdou vozidla. Cyklus bude dán následujícími množinami vrcholů: $\{v_0, v_i, v_j, v_{l_2}, \dots, v_{l_s}, v_0\}$ nebo $\{v_0, v_{l_s}, \dots, v_{l_2}, v_j, v_i, v_0\}$.

b) Sklad v_j je obsluhován samostatnou jízdou vozidla. Cyklus bude dán následujícími množinami vrcholů: $\{v_0, v_j, v_i, v_{k_2}, \dots, v_{k_r}, v_0\}$ nebo $\{v_0, v_{k_r}, \dots, v_{k_2}, v_i, v_j, v_0\}$.

c) Sklady v_j a v_j jsou obsluhovány dvěma různými vozidly. Cyklus bude dán následujícími množinami vrcholů: $\{v_0, v_{l_s}, \dots, v_{l_2}, v_j, v_i, v_{k_2}, \dots, v_{k_r}, v_0\}$ nebo $\{v_0, v_{k_r}, \dots, v_{k_2}, v_i, v_j, v_{l_2}, \dots, v_{l_s}, v_0\}$.

3. Zjistíme zda existuje ještě nějaké $\lambda_{ij} > 0$, které dosud nebylo vybráno. Pokud existuje, přejdeme na krok 4. Pokud neexistuje, pokračujeme krokem 6.
4. Vybereme maximum z λ_{ij}, které ještě nebylo vybráno. Pro vybrané vrcholy testujeme platnost podmínek α a β. Pokud jsou tyto podmínky splněny, označíme toto λ_{ij} jako použitelné a pokračujeme krokem 5. Pokud alespoň jedna z podmínek α nebo β není splněna, označíme příslušné jako nepoužitelné a pokračujeme krokem 3.
5. Sjednotíme trasu do v_i a do v_j do jedné, určíme příslušná γ a pokračujeme krokem 3.
6. Obdržené řešení je (sub)optimálním řešením okružních jízd na síti.

Celý algoritmus bude demonstrován na jednoduchém příkladu. Je dán graf, který je souvislý, vrcholově i hranově ohodnocený. Ohodnocení vrcholů vyjadřuje kapacitu, ohodnocení hran jejich délku v km. Uvažujme homogenní vozový park s kapacitou vozidel $C = 25$, která jsou všechna umístěna v jednom depu. Na obrázku 8 je znázorněn graf, v příloze 1 je vypočtená matice nejkratších vzdáleností mezi jednotlivými vrcholy.



Obrázek 8 – vzorový graf [zdroj: Karlíková]

Náklady výchozího řešení, tedy řešení, kde je každý vrchol obsluhován samostatnou jízdou jsou 326 km.

Pro každý vrchol vypočteme hodnoty $\lambda_{ij} = d_{i0} + d_{0j} - d_{ij}$. Matice úspor je zobrazena v tabulce 5. Tato matice je souměrná podle hlavní diagonály.

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}
v_1	0													
v_2	20	0												
v_3	14	22	0											
v_4	14	22	26	0										
v_5	7	15	17	31	0									
v_6	0	5	7	21	22	0								
v_7	16	8	2	1	1	0	0							
v_8	8	10	4	4	-4	0	8	0						
v_9	6	14	16	2	17	7	0	0	0					
v_{10}	0	0	0	11	12	12	0	0	0	0				
v_{11}	18	8	2	12	0	0	16	8	0	0	0			
v_{12}	0	0	0	0	0	0	0	0	0	0	10	0		
v_{13}	6	0	0	0	0	0	4	0	0	0	27	14	0	
v_{14}	0	0	0	10	1	1	0	0	0	0	7	11	11	0

Tabulka 5 – tabulka úspor [zdroj: Karlíková]

Nastavíme $\gamma_i = 1$ pro $i = 1, \ldots, n$. Nalezneme maximální $\lambda_{ij} > 0$, které ještě nebylo vybráno (pokud existuje).

- $\lambda_{4,5} = 31$ - Protože $c_4+c_5 = 19 < C$, sloučíme vrchol v_4 a v_5 do jedné trasy $\{v_0, v_4, v_5, v_0\}$. Hodnoty γ_i se nezmění. Do kapacit vrcholů v_4 a v_5 přiřadíme jejich součet. $\lambda_{4,5}$ označíme jako prošlé.
- $\lambda_{11,13} = 27$ - Protože $c_{11}+c_{13} = 14 < C$, sloučíme vrchol v_{11} a v_{13} do jedné trasy $\{v_0, v_{11}, v_{13}, v_0\}$. Hodnoty γ_i se nezmění. Do kapacit vrcholů v_{11} a v_{13} přiřadíme jejich součet. $\lambda_{11,13}$ označíme jako prošlé.
- $\lambda_{3,4} = 26$ - Protože $c_4+c_3 = 28 > C$, označíme $\lambda_{11,13}$ jako prošlé a pokračujeme dál.
- $\lambda_{2,3} = 22$ - Protože $c_2+c_3 = 19 < C$, sloučíme vrchol v_2 a v_3 do jedné trasy $\{v_0, v_2, v_3, v_0\}$. Hodnoty γ_i se nezmění. Do kapacit vrcholů v_2 a v_3 přiřadíme jejich součet. $\lambda_{2,3}$ označíme jako prošlé.
- $\lambda_{2,4} = 22$ - Protože $c_2+c_4 = 38 > C$, označíme $\lambda_{2,4}$ jako prošlé a pokračujeme dál.
- Stejně tak součet kapacit vrcholů (v_5, v_6) a (v_4, v_6) je větší než kapacita vozidla.
- $\lambda_{1,2} = 20$ - Protože $c_1+c_2 = 25 = C$, sloučíme vrchol v_1 a v_2 do jedné trasy. Jelikož je vrchol v_2 součástí trasy $\{v_0, v_2, v_3, v_0\}$ vznikne trasa $\{v_0, v_1, v_2, v_3, v_0\}$. Do $\gamma_2 = 0$, do kapacit všech vrcholů vzniklé trasy přiřadíme součet kapacit v_1 a v_2 a $\lambda_{2,3}$ označíme jako prošlé.
- (v_1, v_{14}) , (v_3, v_5), (v_5, v_9), (v_1, v_7) a (v_3, v_9) nesplňují kapacitní podmínku.
- $\lambda_{7,11} = 16$ - Protože $c_7+c_{11} = 22 < C$, sloučíme vrchol v_7 a v_{11} do jedné trasy. Jelikož je vrchol v_{11} součástí trasy $\{v_0, v_{11}, v_{13}, v_0\}$ vznikne trasa $\{v_0, v_7, v_{11}, v_{13}, v_0\}$. Do $\gamma_{11} = 0$, do kapacit všech vrcholů vzniklé trasy přiřadíme součet kapacit v_7 a v_{11} a $\lambda_{7,11}$ označíme jako prošlé.
- (v_2, v_5), (v_1, v_3), (v_1, v_4), (v_2, v_9), (v_{12}, v_{13}) a (v_{10}, v_5) nesplňují kapacitní podmínku.
- $\lambda_{10,6} = 12$ - Protože $c_{10}+c_6 = 20 < C$, sloučíme vrchol v_{10} a v_6 do jedné trasy $\{v_0, v_{10}, v_6, v_0\}$. Hodnoty γ_i se nezmění. Do kapacit vrcholů v_{10} a v_6 přiřadíme jejich součet. $\lambda_{4,5}$ označíme jako prošlé.
- (v_{11}, v_4) a (v_{10}, v_4) nesplňují kapacitní podmínku.
- $\lambda_{12,14} = 11$ - Protože $c_{12}+c_{14} = 20 < C$, sloučíme vrchol v_{12} a v_{14} do jedné trasy $\{v_0, v_{12}, v_{14}, v_0\}$. Hodnoty γ_i se nezmění. Do kapacit vrcholů v_{12} a v_{14} přiřadíme jejich součet. $\lambda_{12,14}$ označíme jako prošlé.

- (v_4, v_{13}) , (v_2, v_8), (v_{12}, v_{11}), (v_4, v_{14}), (v_2, v_7), (v_1, v_8), (v_7, v_8), (v_2, v_{11}), (v_8, v_{11}), (v_1, v_5), (v_3, v_6), (v_6, v_9), (v_{11}, v_{14}), (v_1, v_5), (v_2, v_6) a (v_3, v_8) nesplňují kapacitní podmínku.
- $\lambda_{4,8} = 4$ - Protože $c_4+c_8 = 24 < C$, sloučíme vrchol v_4 a v_8 do jedné trasy. Jelikož je vrchol v_4 součástí trasy $\{v_0, v_4, v_5, v_0\}$ vznikne trasa $\{v_0, v_8, v_4, v_5, v_0\}$. Do $\gamma_4 = 0$, do kapacit všech vrcholů vzniklé trasy přiřadíme součet kapacit v_4 a v_8 a $\lambda_{4,8}$ označíme jako prošlé.
- (v_7, v_{13}), (v_3, v_7), (v_4, v_9), (v_3, v_{11}), (v_4, v_7), (v_5, v_7), (v_5, v_{14}) a (v_6, v_{14}) nesplňují kapacitní podmínku.

Neexistuje žádné $\lambda_{ij} > 0$, které ještě nebylo vybráno. Algoritmus končí.

Vzniklé trasy:

- $\{v_0, v_8, v_4, v_5, v_0\}$ – náklady 5 + 21 + 5 + 16 = 47 km
- $\{v_0, v_{12}, v_{14}, v_0\}$ – náklady 7 + 7 + 11 = 25 km
- $\{v_0, v_{10}, v_6, v_0\}$ – náklady 6 + 5 + 11 = 22 km
- $\{v_0, v_7, v_{11}, v_{13}, v_0\}$ = náklady 8 + 8 + 8 + 12 = 36 km
- $\{v_0, v_1, v_2, v_3, v_0\}$ – náklady 16 + 10 + 5 + 13 = 44 km
- $\{v_0, v_9, v_0\}$ – náklady 8 + 8 = 16 km

Součtem nákladů na jednotlivé trasy získáme celkové náklady získaného řešení, tedy 47 + 25 + 22 + 36 + 44 + 16 = 190 km.

4. Analýza současného stavu řešení

4.1. Průzkum situace

Společnostem zabývajícím se svozem odpadu v ČR byl poslán e-mail (viz. příloha 2) s dotazem na používané SW nástroje pro plánování tras svozu odpadu. Bylo dotazováno 18 firem. Pouze 12 z nich odpovědělo. Výsledky došlých odpovědí jsou znázorněny v tabulce 6.

Počet dotazovaných firem	18
Počet odpovědí	12
Počet firem používajících software pro plánování tras vozidel	1
Počet firem pořádajících výběrové řízení	1
Počet firem, které uvažují o pořízení software pro plánování tras vozidel	2
Počet firem, které nepoužívají žádný software pro plánování tras vozidel	8

Tabulka 6 – počty firem využívajících SW pro plánování tras [zdroj: Karlíková]

Z výsledků vyplývá, že většina firem nepoužívá k plánování tras vozidel žádný speciální softwarový nástroj. Tato situace může být způsobená složitostí úlohy svozu odpadu. Také může být způsobena tím, že existujících softwarových řešení na českém trhu není mnoho a většina z nich nabízí komplexní řešení jak optimalizace tras, tak monitoring vozidel včetně vazby na ERP systém s návazností na objednávky svozu. Tato řešení jsou většinou drahé systémy, ve kterých by většina funkcí nebyla v případě optimalizace svozu využita.

4.2. Současný stav vedení tras svozu komunálního odpadu – Pardubice

O svoz a likvidaci odpadu v Pardubicích se stará firma SmP – Odpady a.s. (Služby města Pardubice – Odpady). Nabízí svoz a likvidaci celé škály odpadů od domovního svozu komunálního odpadu, přes svoz separovaného a biodegradabilního odpadu až po výsyp odpadkových košů nebo přistavení velkoobjemových kontejnerů.

V současné době není pro plánování tras vozidel používán žádný software. K plánování tras je využíván lidský faktor. Informace o místech sběru odpadu jsou vedeny v excelovských tabulkách. Pardubice jsou rozděleny do rajónů. Každému z rajónů je přiděleno vozidlo, které má na starost svoz komunálního odpadu v tomto rajónu. Řidičům vozidel je k dispozici seznam ulic, které jsou součástí jejich rajónu. Trasy jsou určovány na základě zkušeností osoby, která se stará o rozdělení rajónů a zkušeností řidičů, kteří danou trasu jezdí.

Frekvence svozu komunálního odpadu se řídí dle předplatného jednotlivých majitelů (jednou týdně, jednou za 14 dní nebo jednou za měsíc). V případě separovaného odpadu se frekvence řídí aktuálními potřebami. Při nedostatečné kapacitě kontejneru se frekvence výsypu zvyšuje a naopak. U biodegradabilních odpadů je situace stejná jako u komunálního odpadu.

Depo vozidel se nachází v sídle společnosti. Parkují zde všechna vozidla zajišťující svoz odpadu.

Vozidla se jezdí vysýpat víckrát za den – průměrně dvakrát až třikrát. Komunální odpad se neodváží na skládky, které jsou 35 – 40 km daleko, ale na překladiště odpadu, které se nachází v Dražkovicích. Doba potřebná k vyprázdnění vozidla (cesta do Dražkovic, výsyp vozidla, cesta zpět na původní trasu) je asi 30 minut.

4.3. Existující SW řešení

Na českém trhu není v oblasti optimalizace svozu a rozvozu mnoho aplikací. Následující přehled produktů je pouze výčet nejznámějších z nich. Zahrnuje popis produktu, jeho výhody a nevýhody nasazení v oblasti svozu komunálního odpadu.

4.3.1. Logicon Partner s.r.o. – PLANTOUR

LOGICON Partner, s.r.o. poskytuje logistické poradenství středním a velkým společnostem ve formě projektů, studií, konzultací a vzdělávání v oblasti logistiky, včetně dílčí SW podpory pro optimalizaci vybraných logistických procesů.[15]

Popis produktu

Program PLANTOUR – strategická a operativní optimalizace tras, vozidel a nákladů.[15]

PLANTOUR umožňuje na základě každodenního zpracování objednávek navrhovat optimální trasy pro závoz dodacích míst včetně zohlednění zpětných svozů. Trasy jsou navrhovány na základě aktuálních objednávek a vozového parku dynamicky tak, aby byly optimální z hlediska nákladů.[15]

PLANTOUR přináší možnost plné každodenní kontroly nad náklady na distribuci až na úroveň dodacího místa s možností jejich dalšího snižování pomocí optimalizace tras. Přímých úspor je možno dosáhnout redukcí tras, nákladů, počtu vozidel, ujetých kilometrů. Úspora přímých dopravních nákladů 10 - 30% je ověřena desítkami implementací systému v ČR i zahraničí.[15]

Navržené trasy jsou prezentovány několika způsoby. Jedná se o tiskové sestavy, dispečerský přehled, itineráře pro řidiče. Veškeré informace o trasách se zobrazují v přehledném datovém seznamu, mapovém okně a na časové ose. Využitím analytických možností produktu je možno definovat potřeby vozového parku, optimální rozložení dep, přiřazení dodacích míst depům, apod. [15]

Výhody

- redukce nákladů na svoz
- optimální využití vozového parku

Nevýhody

- napojení aplikace na ERP systém společnosti, návaznost na přijaté objednávky – v řešení svozu odpadu toto napojení není nutné
- komplexní nástroj – vysoká cena

4.3.2. TRANIS s.r.o. – Kilometrovník PRO

Firma TRANIS s.r.o. se zabývá prováděním statistických šetření a podrobných analýz zejména v oblasti silniční nákladní dopravy.

Popis produktu

Plánovač tras Kilometrovník je PC software pro plánování a výpočet tras s možností vytvoření a tisku podrobného itineráře. Plánovač tras obsahuje podrobné digitální mapy ČR a Evropy. Výpočet nákladů na přepravu pomocí Plánovač tras zahrnuje výpočet ceny za PHM, a to podle ceny daného paliva v každé zemi, poplatky za mýtné, dálnice, mosty a tunely. Navigace GPS umožňuje využívat Plánovač tras jako navigační systém.[16]

Program dále umožňuje výpočet trasy ze zadaného výchozího do cílového místa se zadáním až 50-ti tranzitních míst. Program dokáže optimalizovat okružní trasu a vypočtené trasy se graficky zobrazit na mapě. Umožňuje výpočet nákladů na přepravu a zahrnuje výpočet ceny za pohonné hmoty, poplatky za použití trajektů, silnic, dálnic, (tunelů a mostů). [16]

Výhody

- optimalizace okružní jízdy

Nevýhody

- zaměřen na plánování tras na velké vzdálenosti
- není použitelný na úlohu svozu komunálního odpadu, dokáže pouze optimalizovat danou okružní jízdu.
- komplexní nástroj – vysoká cena

4.3.3. Telefónica O2 Business Solutions – LOGISmart

Společnost Telefónica O2 Business Solutions vznikla integrací Telefónica O2 Services a společnosti DELTAX Systems a je dnes jedním z největších tuzemských hráčů na poli ICT a největším dodavatelem služeb pro český stát.[17]

Popis produktu

LOGISmart je systém plánování, monitorování a vyhodnocování pohybu a provozu vozidel. Skládá se ze dvou logicky souvisejících, vzájemně provázaných „částí“ Monitoring a Logistika, které pracují on-line a tvoří jediný komplexní celek.[17]

Monitoring zahrnuje v on-line režimu vlastní evidenci pohybu vozidla a také sledování různých parametrů spotřeby a výkonu vozidla.[17]

Logistika umožňuje činnosti plánovat a na základě on-line informací z Monitoringu také vyhodnocovat. Uživatel má k dispozici nejen obrazové a tiskové informace o plánu a skutečnosti plnění jednotlivých tras a svozů, ale současně i vyhodnocení plnění plánovaných tras na mapovém podkladu. Na mapovém podkladu je zobrazován jak plán, tak skutečnost vybrané trasy a dále také označení ulic a objektů, umístění jednotlivých technických nádob, které jsou předmětem svozu/odvozu, atd.[17]

Výhody

- úspora nákladů na jízdy vozidel
- monitoring jízd
- identifikace rozdílů mezi skutečnou a naplánovanou trasou

5. Analýza konkrétního reálného systému aplikace

5.1. Popis projektu

Zadáním projektu bylo vytvořit komplexní programový nástroj umožňující automatickou sestavu měsíčních plánů tras svozu komunálního odpadu s podrobnou mapou vybraného regionu a možností online sledování vozidel. Práce byla rozdělena na dvě části (mezi dva studenty):

1. část – výpočet plánů tras pro jednotlivá vozidla, sestava měsíčních plánů.
2. část – zpracování mapových podkladů a online sledování vozidel,

5.1.1. Sběr odpadu

Třídění, sběr a svoz komunálního odpadu organizuje většinou obec ve spolupráci se svozovou firmou.

5.1.1.1. Druhy odpadů

Existuje několik druhů odpadů. Následující tabulka poskytuje přehledný výčet s konkrétními příklady.

Druh odpadu	***Co sem patří:***	***Co sem nepatří:***
tříděný odpad		
o papír	noviny, časopisy, karton	kopíráky, znečištěný papír, tvrdá vazba z knih
o barevné sklo	barevné skleněné lahve, tabulové sklo	čiré skleněné lahve, drátěné sklo, automobilové sklo
o bílé sklo	čiré skleněné lahve	barevné skleněné lahve, tabulové sklo, drátěné sklo, automobilové sklo
o plasty	vše, co je vyrobeno z plastů, igelitové (polyethylenové) a mikrotenové obaly, PET lahve, kelímky od jogurtů, od domácí drogerie	bakelit, guma, PVC, linoleum, pneumatiky, plastové obaly od chemikálií olejů a barev
o nápojové kartony	nápojové krabice bez hliníkové vrstvy (krabice od džusů, mléka)	ostatní papírové krabice

o kovy	železo a barevné kovy	
o textil	veškeré nepotřebné ošacení, ložní prádlo, bytové textilie, atd.,	koberce, matrace, plasty, kůže, syntetické materiály, molitan, impregnované textilie
Komunální odpad	předměty denní spotřeby z domácností	
Nebezpečný odpad	nebezpečné složky komunálního odpadu, které ohrožují životní prostředí - fritovací olej, zbytky starých barev, ředidla včetně obalů, baterie, léky, atd.,	televizory, ledničky, zářivky a další elektrozařízení
elektrozařízení	elektrospotřebiče	elektrické bojlery
velkoobjemový odpad	nábytek, matrace, koberce, linolea, lyže, sáňky, kola, dveře, WC, umyvadla	stavební suť, komunální odpad, který se vejde do klasické popelnice, bojlery, pračky, ždímačky
stavební suť	odpad ze stavby – cihly, beton, omítka	
autovraky	staré vraky z osobních a nákladních automobilů	
biodegradabilní odpad	zbytky jídla, produkty zemědělství a zahrádkaření, zvířecí exkrementy, skořápky vajec a ořechů, potraviny s prošlou trvanlivostí	jedlé oleje, kosti, maso, uhynulá zvířata, znečištěné pilin,
pneumatiky	staré pneumatiky z osobních aut.	

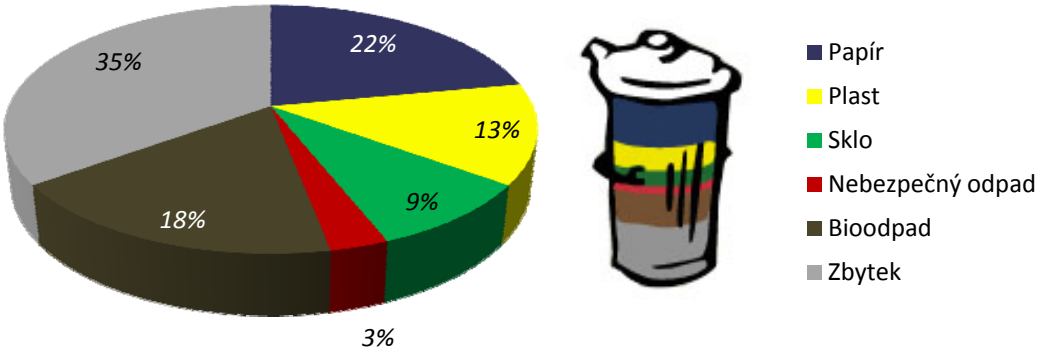
Tabulka 7 – druhy odpadů[25]

5.1.1.2. Způsoby třídění odpadu

Rozlišujeme dva druhy třídění odpadu – hromadné třídění komunálního odpadu, kdy je odpad všeho druhu vyhazován do jedné sběrné nádoby, třídění tohoto odpadu probíhá až na skládkách. Dalším druhem je třídění odpadu před uložení do sběrných nádob. Tomuto druhu třídění se dává přednost hlavně z finančních důvodů.

Každý průměrný občan ČR vyhodí za rok asi 150 - 200 kg odpadů. Pokud jsou odpady tříděny už doma a odkládány do barevných kontejnerů, je možná

recyklace více než třetiny tohoto množství. Za rok tak může každý vytřídit až 30 kg papíru, 25 kg plastů, 15 kg skla.



Obrázek 9 – Skladba domovního odpadu v % hmotnosti[27]

Třídění odpadu umožňuje jejich další zpracování. Odpady, které jsou roztříděny do barevných kontejnerů, odveze svozové auto na dotřiďovací linku. Odtud putují do zpracovatelských firem, kde z odpadů vznikají nové výrobky.[27]

5.1.1.3. Sběrné nádoby

Pro uložení směsného komunálního odpadu slouží nejčastěji popelnice o objemu 110l z pozinkovaného plechu nebo plastové nádoby o objemu 120l. Na sídlištích se používají pojízdné kontejnery z plastu nebo pozinkované oceli o objemu 1,1 m^3.

Pro tříděný odpad se nejčastěji používají barevně označené kontejnery z plastu nebo pozinkované oceli o objemu 1,1 m^3. V poslední době se začínají využívat podzemní kontejnery, které nahrazují klasické nevzhledné kontejnery a navíc šetří místo. Nevýhodou jsou zatím vysoké náklady na pořízení spojené hlavně s vybudováním podzemní šachty pro umístění kontejneru. Barevné označení kontejnerů je přehledně popsáno v tabulce

Obrázek 10 – barevně označené kontejnery na tříděný odpad o objemu 1,1 m^3 [zdroj: http://www.psas.cz/main.cfm?path=1,56]

Sběr bioodpadu je prováděn pomocí speciálních plastových hnědých nádob, tzv. Compostainerů o objemu 120 nebo 240 litrů.

Modrá	Papír
Zelená	Barevné sklo
Bílá	Bílé sklo
Žlutá	Plasty
Hnědá	Bioodpad

Tabulka 8 – barevné označení kontejnerů pro tříděný odpad [zdroj: Karlíková]

5.1.1.4. Svoz a přeprava

Odpad je svážen pomocí speciálních nákladních automobilů. Tyto automobily můžeme dělit podle typu sběrných nádob, které sváží. Rozlišujeme automobily např.: pro svoz komunálního odpadu, pro svoz nebezpečného odpadu nebo automobily pro odvoz velkoobjemových kontejnerů apod.

Svoz probíhá ve dvou (až třech) fázích. V první fázi občan odnese odpad do sběrné nádoby. Každý zákazník má vlastní odpadovou nádobu, kterou plní odpadem. Podle nastavených pravidel svozu ji v definovaném čase připraví k vysypání. Četnost odvozu záleží na přání zákazníka a provozních podmínkách v dané lokalitě. Nádoby by měly v den svozu být připraveny nejdále 15 metrů od okraje silnice.

Na základě přepravní vzdálenosti mezi sběrným místem a skládkou, může být druhá fáze rozdělena do dvou etap. Pokud je tato vzdálenost přijatelná, je odpad odvážen přímo na skládku. V případě větší vzdálenosti, je odpad odvážen na překladiště, ze kterého je ve speciálních velkoobjemových nádobách, které jsou až několikanásobně větší než nákladní automobily svážející odpad, převážen na

skládku. V případě dlouhých vzdáleností mezi sběrnými místy a skládkou, dochází k úspoře nákladů na odvoz odpadu na skládku.[27]

5.2. Využití telematiky

Pomocí nástrojů telematiky se řeší online zobrazování vozidel na mapových podkladech.

5.2.1.1. Mapové podklady

Hlavním poskytovatelem mapových podkladů v České republice je společnost CEDA. Tyto mapové podklady jsou ve formátu ESRI shapefile. Dělí se na vektorové a rastrové. Ceny těchto mapových podkladů se pohybují od 9000 Kč za vektorový mapový podklad okresního města v měřítku 1 : 10 000 až po sta tisíce korun za podrobné vektorové mapy celé České republiky v měřítku 1:10 000. Ceny rastrových map se pohybují řádově od 1 000 Kč za mapu okresního města do 25 000 Kč za rastrový mapový podklad České republiky v měřítku 1 : 50 000.

5.2.1.2. Sledování vozidel

Sledování vozidel můžeme rozdělit do dvou skupin:

- aktivní systémy - umožňují sledování vozidla v reálném čase, informace o pohybu vozidla vysílá aktivní systém pomocí sítě GSM,
- pasivní systémy – zaznamenávají pohyb vozidla pomocí GPS systému, zobrazení a vyhodnocení dat o jízdě vozidla ze provádí po skončení cesty vozidla.

Tato diplomová práce se zabývá realizací 1. části a zaměřuje konkrétně na optimalizaci svozu komunálního odpadu v regionu Pardubice s cílem redukce nákladů na svoz odpadu – nižší spotřeba paliva na základě optimalizace tras svozu odpadu.

5.3. Požadavky

- interaktivní grafické prostředí,
- mapový podklad – práce s mapou (přiblížení, oddálení, posun),
- přidávání, odebírání a úprava sběrných míst,

- import a export sběrných míst,
- správa typů kontejnerů a správa jednoduchého vozového parku
- na základě informací o sběrných místech, typech kontejnerů a kapacit vozidel naplánovat optimální trasy pro svoz odpadu,
- zobrazení tras na mapě,
- výstup - měsíční plány tras pro jednotlivá vozidla, statistiky počtu svezeného odpadu a počtu ujetých kilometrů a spotřeba paliva,
- online zobrazení vozidel,
- online sledování vozidel.

5.4. Řešení projektu

Výstupem projektu bude aplikace řešící problém optimalizace svozu komunálního odpadu v regionu Pardubice. Mapovým podkladem bude podrobná mapa Pardubic. Program bude umožňovat přibližovat, oddalovat a posouvat mapu, zobrazovat sběrná místa, vozidla a trasy vozidel. Dále program umožní kliknutím na mapu přidávat, upravovat nebo odebírat sběrná místa. Sběrná místa bude možné exportovat a importovat do formátu XML. K dispozici bude možnost správy typů kontejnerů a správa jednoduchého vozového parku. Hlavní funkčností aplikace bude výpočet optimálních tras svozu komunálního odpadu, jejich zobrazení na mapě a sestavení plánů pro zadaná období, které bude možno zobrazit buď v okně aplikace, nebo vyexportovat do formátu HTML, kde se zobrazí ve formě přehledných tabulek. Dále bude možné zobrazit statistiky počtu ujetých kilometrů a počtu svezeného odpadu s následnou možností exportu do přehledných tabulek do formátu HTML. U statistik bude umístěna jednoduchá kalkulačka výdajů za pohonné hmoty, na které si uživatel, na základě spotřeby vozidel a ceny za pohonné hmoty, vypočítá celkovou cenu za pohonné hmoty. Aplikace bude dále umožňovat sledování vozidel v reálném čase. Aplikace bude číst hodnoty polohy vozidel z databáze a tyto hodnoty zobrazovat na mapě, což umožní kontrolu plnění naplánovaných tras.

6. Návrh metody řešení

Byla vytvořena softwarová aplikace pro plánování tras svozu komunálního odpadu, vycházející ze základních principů sběru komunálního odpadu v regionu Pardubice.

6.1. Výběr algoritmu

Výchozí předpoklady:

- jedno depo – všechna vozidla vyjíždějí z jednoho místa, každé vozidlo sváží odpad na určité trase, všechna vozidla se na konci směny vrací zpět do stejného depa,
- více vozidel – heterogenní vozový park – pro svoz odpadu jsou k dispozici různé typy vozidel svážející různý typ odpadu,
- heterogenní poptávka ve vrcholech – různé typy kontejnerů – ve vrcholech může být více kontejnerů různého typu (různý obsah kontejneru a různá frekvence výběru)
- vstup – distanční matice.

Pro řešení dané úlohy je vhodné použít některou z metod teorie grafů. Po zhodnocení těchto předpokladů byl pro výpočet jednotlivých okružních jízd zvolen Clark-Wrightův algoritmus. Hlavním důvodem pro toto rozhodnutí byl fakt, že máme k dispozici více vozidel různého typu. Pro tato vozidla je třeba naplánovat svozné trasy tak, aby byl každý vrchol obsloužen správným typem vozidla a to právě jednou.

Hodnotícím kritériem je minimalizace dopravních nákladů, tedy minimalizace počtu najetých kilometrů. Výchozím bodem je matice distančních vzdáleností, ze které je počítána matice úspor.

Při plnění tras je důležité, aby doba trvání svozné trasy nepřesáhla pracovní dobu posádky vozidla. Tato doba je stanovena na základě průměrného počtu výsypů vozidla. Každý výsyp vozidla trvá průměrně půl hodiny. V případě průměrného počtu výsypů 2 (jeden výsyp vozidla v průběhu trasy a jeden výsyp vozidla na konci směny) – je od hodnoty 8 hodin odečteno 2 krát půl hodina a tato doba se nastaví do maximální vymezené doby.

Doba trvání trasy se počítá jako součet časů jednotlivých úseků trasy. Podle vzorce $t = \frac{d}{v}$, kde t je čas v hodinách, d je délka úseku trasy v kilometrech a v je průměrná rychlost v kilometrech za hodinu, se spočítá čas potřebný na projetí jednoho úseku trasy. Při výpočtu je průměrná rychlost závislá na délce projížděného úseku, od rychlosti 5 km/h pro úseky do 100 m až po 30 km/h pro úseky nad 1 km. K délce trasy je zároveň připočítána průměrná doba obsluhy vrcholu, která je nastavena na minutu a půl.

6.2. Modifikace algoritmu

Vzhledem k tomu, že se jedná primárně o svoz komunálního odpadu, byl algoritmus upraven pro tyto potřeby. V základní verzi algoritmu jsou vrcholy spojovány do tras na základě úspor. V modifikované verzi jsou vrcholy, které mají být spojeny do tras hledány nejen podle hodnoty úspor, ale zároveň algoritmus testuje, zda vrcholy, které mají být spojeny, leží na stejné komunikaci.

Při hledání vrcholů pro spojení do jedné trasy se bude v matici úspor hledat prvek, jehož vrcholy se nachází na stejné komunikaci a z těchto prvků bude vybrán ten, jehož úspory budou maximální.

6.3. Popis řešení

6.3.1. Správa sběrných míst, kontejnerů a vozidel

V programu je vedena správa sběrných míst, což je seznam sběrných míst, kde index prvku odpovídá indexu vrcholu. Každé sběrné místo má svého majitele a seznam kontejnerů, nacházejících se v daném místě. Každý kontejner je určitého typu a je mu nastavena určitá frekvence obsluhy, která musí být dodržena. Konkrétní typ kontejneru může být obsloužen jen určitým typem vozidla. Proto jsou v programu zavedeny správa typů kontejnerů a správa typů vozidel.

Správa kontejnerů obsahuje seznam typů kontejnerů, které mohou být sváženy. Typ kontejneru obsahuje typ odpadu a obsah.

Správa typů vozidel obsahuje seznam typů vozidel. Každý typ vozidla uchovává vlastnosti společné pro vozidla svážející určitý typ kontejneru. Obsahuje

kapacitu vozidla, typ kontejneru, který sváží a průměrný počet výsypů. Na základě typů vozidel je tvořen jednoduchý vozový park.

Typy kontejnerů, typy vozidel i vozový park si definuje uživatel. U všech typů správ má k dispozici funkce přidej, odeber a uprav. Nejprve musí definovat typy kontejnerů, které budou sváženy, na základě typů kontejnerů budou definovány typy vozidel a na základě těchto typů vozidel bude definován jednoduchý vozový park.

6.3.2. Rozdělení vrcholů

Clark-Wrightův algoritmus lze použít pouze v případě homogenní poptávky ve vrcholech a homogenního vozového parku, proto bylo nutné problém rozdělit na několik dílčích podproblémů, které budou řešitelné Clark-Wrightovým algoritmem.

Vrcholy (sběrná místa) byly rozděleny do skupin dle typu kontejneru, který se v daném místě nachází. K těmto vrcholům byla přiřazena distanční matice jejich vzdáleností. Zároveň byl každé skupině přiřazen typ vozidla, jehož kapacita je použita při výpočtu algoritmu.

Při rozdělování vrcholů bylo nutné brát v úvahu i frekvenci výběru jednotlivých sběrných míst. Podle frekvence výběru a data posledního výběru je zjištěno, zda má nebo nemá být sběrné místo zařazené do dané skupiny vrcholů.

Příklad: Pokud má vrchol typ kontejneru – komunální odpad, byl by zařazen do skupiny vrcholů se stejným typem kontejneru. Ale uvažujme, že vrchol má frekvenci výběru jednou týdně. Datum spuštění algoritmu je 22. 4. 2009 a datum posledního výběru tohoto kontejneru je 24. 4. 2009, to znamená, že kontejner ještě nemá být vybrán. Tento kontejner nebude 24. 4. 2009 zařazen do žádné skupiny, protože frekvence jeho výběru je jednou týdně. Pokud bude algoritmus spuštěn 28.4.2009 toto sběrné místo bude zařazeno do skupiny podle typu kontejneru, který obsahuje a datum posledního výběru se nastaví na 28.4.2009. Vzhledem k tomu, že datum 22. 4. 2009 je úterý, je vše v pořádku. Situace se komplikuje v případě, že by datum posledního výběru, v případě frekvence jednou týdně, nebylo úterý. Pokud by datum posledního výběru bylo nastavené na 21. 4. 2009, vrchol by byl vybrán také až 28. 4. 2009.

To je způsobeno tím, že všechny vrcholy jsou podle typu frekvence vybírány jen v určité dny:

frekvence výběru	*dny*
denně	pondělí až pátek
obden	pondělí, středa, pátek
jednou týdně	každé úterý
jednou za 14 dní	druhý a čtvrtý čtvrtek v měsíci
jednou měsíčně	první čtvrtek v měsíci

Tabulka 9 – frekvence výběru [zdroj: Karlíková]

Hlavním důvodem tohoto nastavení je zajistit, aby se pro vrcholy s určitou frekvencí výběru uskutečňoval svoz odpadu ve stejný den. Nemůže potom nastat situace, kdy by část vrcholů s danou frekvencí výběru byla svážena v jeden den a druhá část v jiný den, což by v případě malého množství přidaných sběrných míst mohlo vést k neefektivnímu plánování svozných tras.

Situace, kdy je datum posledního výběru nastaveno na jiný den, než je plánovaný den pro danou frekvenci výběru může nastat pouze v případě přidávání sběrného místa. Do data posledního výběru se při přidání vrcholu, nebo přidání kontejneru do vrcholu, uloží datum v závislosti na frekvenci. Pokud je frekvence výběru denně, do data posledního výběru se uloží datum o jeden den nižší než je datum přidání sběrného místa. V případě frekvence obden, se do data posledního výběru uloží datum o dva dny nižší, než je datum přidání sběrného místa atd. Při prvním výběru se nastaví datum na správný den.

6.3.1. Trasy

Každá skupina vrcholů je uchovávána zvlášť. Pro každou skupinu je udržován seznam původních indexů vrcholů, distanční matice a vypočítané trasy. Každá trasa obsahuje seznam indexů vrcholů, které odpovídají indexům v distanční matici. Tyto indexy jsou platné pouze pro danou supinu vrcholů, pro zobrazení tras je nutné tyto indexy převést pomocí seznamu původních indexů.

6.3.2. Popis algoritmu

Po rozdělení vrcholů do skupin dle typu kontejneru, je na každou skupinu vrcholů aplikován modifikovaný Clark-Wrightův algoritmus.

Nejprve jsou inicializovány hodnoty, potřebné pro běh algoritmu. Každý vrchol obsahuje několik hodnot nutných k výpočtu algoritmu. Tyto hodnoty jsou uvedeny v tabulce 10.

hodnota	*popis*	*výchozí nastavení*
`gama[i]`	určuje, zda je vrchol na konci nebo uprostřed trasy	1
`obsluhovanSamostatne[i]`	určuje, zda je vrchol obsluhován samostatně, nebo je součástí nějaké trasy	`true`
`cisloTrasy[i]`	určuje, číslo trasy, které je vrchol součástí	-1

Tabulka 10 – hodnoty vrcholů nutné pro výpočet Clark-Wrightova algoritmu [zdroj: Karlíková]

V prvním kroku je vytvořena matice úspor z distanční matice. Matice úspor je v programu implementována jako seznam typu `ArrayList`, v němž jsou prvky setříděny nejprve podle toho, zda se nacházejí na stejné silnici a následně podle hodnot úspor. Základem algoritmu je cyklus, který:

1) vybere maximální neprozkoumanou hodnotu z matice úspor, přičemž mezi maximálními hodnotami úspor hledá vrcholy, které se nacházejí na stejné komunikaci. Vybranou hodnotu nastaví jako prozkoumanou. Tím, že je matice úspor uložená v seznamu typu `ArrayList`, je složitost při vybírání prvku s maximální hodnotou úspor $f(n) = O(1)$, protože prvek je vybrán ze začátku seznamu a rovnou ze seznamu odstraněn. Čas potřebný pro výpočet tras se tak snížil ze 6 s (s použitím matice o rozměrech $n \times n$) na 0,018 s.
2) pro vybrané vrcholy, testuje platnost podmínek α a β.
3) pokud jsou podmínky splněny, zjistí, zda jsou vrcholy zařazeny do nějaké trasy nebo zda jsou obsluhovány samostatnou jízdou vozidla. Mohou nastat tři situace:
    - oba vrcholy jsou obsluhovány samostatnou jízdou vozidla – tyto vrcholy spojí do jedné trasy, nastaví hodnotu

`obsluhovanSamostatne` na `false` a nastaví vrcholům číslo trasy.

- Jeden z vrcholů je obsluhován samostatnou jízdou vozidla, druhý je součástí nějaké trasy – vrchol, který je obsluhován samostatnou jízdou vozidla přidá do trasy druhému vrcholu. Nastaví hodnotu `obsluhovanSamostatne` prvního vrcholu na `false`, hodnotu `gama` druhého vrcholu na 0 a do čísla trasy vrcholu, který byl obsluhován samostatnou jízdou vozidla, přiřadí číslo trasy vrcholu, který již byl zařazen v nějaké trase.
- Oba vrcholy jsou součástí nějaké trasy – spojí tyto dvě trasy do jedné, aktualizuje čísla trasy u vrcholů, které byly součástí původních tras a oběma vrcholům nastaví hodnotu `gama` na 0.

Ve všech třech případech zároveň testuje, zda délka vzniklé trasy nepřesahuje pracovní dobu vymezenou pro vozidlo.

Cyklus končí, pokud již není v matici úspor žádné kladné neprozkoumané číslo, tedy pokud je ArrayList, obsahující hodoty úspor, prázdný. Po ukončení cyklu je třeba zjistit, jestli neexistuje vrchol, který nebyl zařazen do žádné trasy. Pokud takový vrchol existuje, vytvoří se pro něj samostatná trasa.

6.3.1. Tvorba plánů

Při počítání tras se datum posledního výběru kontejneru nastaví na den, kdy byl kontejner vybrán. Pokud ale chceme vytvořit plány pro následující měsíc, musíme datum posledního reálného výběru kontejneru uchovat. Proto je nutné uchovávat dvě informace. Jeden datum, označený `posledniVyběr`, který bude uchovávat datum posledního reálného výsypu kontejneru a druhý, označený `vyberPlany`, který bude sloužit pro výpočet plánů. Při výpočtu tras pro měsíční plánování, se vychází z data posledního výběru kontejneru. Pro výpočet tras pro dané období se potom počítá s datem `vyberPlany`, který se v průběhu výpočtu pro jednotlivé dny mění. Na konci výpočtu, je tento datum změněn zpět na dnešní datum.

7. Počítačová implementace

7.1. Použité nástroje

Diplomová práce byla vytvářena pod operačním systémem Windows XP(na jiných platformách není ověřena jeho správná funkčnost). Pro implementaci byl zvolen programovací jazyk C# ve vývojovém prostředí MS Visual Studio 2008. Do vývojového prostředí MS Visual byl integrován nástroj pro správu verzí Visual SVN, který zajistí ukládání jednotlivých verzí programu a zároveň podporuje práci více programátorů na jednom projektu.

7.2. Definice rozhraní

Jelikož byla tato aplikace tvořena jako práce dvou studentů, bylo nutné vymezit si rozhraní mezi těmito dvěma částmi.

Pro potřeby vykreslování objektů a tras na mapě bylo využito rozhraní `IZobrazení`, které poskytuje metody pro přidávání a odebírání bodů z mapového podkladu a metody pro vykreslování a mazání čar na mapovém podkladu. Toto rozhraní slouží i pro práci se sběrnými místy. Obsahuje metodu, která pro vložené prvky vrací jejich distanční matici, vypočtenou na základě nejkratších cest. Tato matice byla použita jako do Clark-Wrightova algoritmu.

7.3. Definice vstupů a výstupů

Jak již bylo řešeno, vstupem do algoritmu je distanční matice, kde prvky matice reprezentují vzdálenosti mezi jednotlivými vrcholy v metrech, a pole sběrných míst, které uchovává všechny informace o sběrných místech. Indexy v poli sběrných míst odpovídají indexům v distanční matici.

V Každém sběrném místě může být několik kontejnerů různého typu každý s různou frekvencí výběru. Každému typu kontejneru odpovídá typ vozidla, který daný kontejner sváží. Dále je k dispozici seznam vozidel, kde každé vozidlo je určitého typu.

Na základě těchto informací jsou vrcholy rozděleny do skupin (princip rozdělování vrcholů do skupin popsán v předchozí kapitole) a jsou pro ně spočítány

trasy, které tvoří výstup z programu. Trasy jsou přehledně zobrazovány v programu jako seznam po sobě jsoucích názvů sběrných míst, který je tvořen z názvu ulice a čísla popisného, z čehož je na první pohled zřejmé, kudy trasa přesně vede.

7.4. Model tříd

Kompletní UML diagram modelu tříd je v příloze 3. Jelikož je model tříd na popisování jako celek rozsáhlý, pro potřeby popisu tříd bude rozdělen do 4 menších oblastí. Takto rozdělený UML diagram je v příloze 4.

První oblastí jsou třídy, reprezentující výpočet Clark-Wrightova algoritmu, jehož hlavní princip byl popsán v předchozí kapitole. Celý výpočet algoritmu probíhá ve třídě `VypoctovaMatice`. Rozdělení vrcholů do skupin zajišťuje `SpravaVypocetnichPrvku`, která se stará o rozdělení vrcholů podle typu kontejneru a frekvence výsypu do skupin. Obsahuje seznam typu `VypocetniPrvek`, který reprezentuje jednu skupinu vrcholů. Každá instance tohoto typu obsahuje distanční matici, pole původních indexů a seznam tras pro danou skupinu. UML diagram tříd týkajících se výpočtu je v příloze 5.

Dále jsou zde třídy, které obsahují informace o typech kontejnerů a typech vozidel. Jsou reprezentovány třídami `TypKontejneru`, `SpravaTypKontejneru`, `TypVozidel`, `SpravaTypuVozidel`. Každý kontejner má svůj název a obsah. Každý typ vozidla má kapacitu a obsahuje instanci třídy `TypKontejneru`. UML diagram těchto tříd je v příloze 6.

Další část modelu tvoří třídy související se sběrnými místy. Každé sběrné místo je realizováno instancí třídy `SberneMisto`. Seznam těchto instancí je uložen ve třídě `SpravaSbernychMist`. Každé sběrné místo má svého majitele – třída `majitelKontejneru` a obsahuje kontejnery, které jsou v daném sběrném místě. Tyto kontejnery představuje třída `Kontejner`. Třídy týkajících se sběrného místa jsou znázorněny v UML diagramu v příloze 7.

Poslední oblastí jsou třídy týkající se vypočtených tras. Vypočtené trasy se pro každou skupinu vrcholů ukládají do instance třídy `SpravaTras`, která obsahuje seznam instancí třídy `Trasa` a důležité metody pro tvorbu tras. Každá trasa obsahuje seznam vrcholů a instanci třídy `Vozidlo`, které je dané trase přiřazeno. Pro práci

s vozidly je k dispozici pouze rozhraní `IVozidlo`, poskytující metody pro práci s vozidly. Třídy týkajících se sběrného místa jsou v UML diagramu v příloze 7.

V následující části jsou popsány jednotlivé třídy a rozhraní a popsány jejich důležité datové složky, vlastnosti a metody.

7.4.1. Třídy pro výpočet algoritmu

7.4.1.1. Třída `prvekMaticeUspor`

Třída, která reprezentuje hodnotu úspor (hodnotu lamba).

Datové složky a vlastnosti

`int[] indexy` – pole indexů vrcholů, které mají být spojeny do trasy.

`double uspory` - hodnota úspor, které vzniknou spojením dvou vrcholů do jedné trasy.

`bool naStejneSilnici` – datová složka, která je nastavena na `true`, pokud jsou vrcholy, které mají být spojeny do trasy na sejné silnici.

`public double Uspory`, `public int[] Indexy`, `public bool NaStejneSilnici` – veřejné vlastnosti poskytující všechny datové složky.

7.4.1.2. Třída `porovnaniUspor : IComparer`

Tato třída implementuje rozhraní `IComparer`. Rozhraní IComparer je např. parametrem jedné z metod `Sort` tříd `Systém.Array` a `Systém.Collections.ArrayList`. [26]

Metody

`public int Compare(object prvek1, object prvek2)` – předefinovaná metoda, slouží k porovnání dvou prvků typu `prvekMaticeUspor`.

- Vrací zápornou hodnotu, pokud je hodnota `NaStejneSilnici` instance `prvek1` `true` a hodnota `NaStejneSilnici` instance `prvek2` `false`.
- Vrací kladnou hodnotu, pokud jsou hodnoty naopak.

- Pokud je hodnota `naStejneSilnici` obou instancí `prvek1` i `prvek2` `true` (nebo `false`), potom:
    - vraci kladnou hodnotu, pokud je hodnota `uspory` instance `prvek1` menší než hodnota `uspory` instance `prvek2`,
    - nulu, pokud hodnota `uspory` instance `prvek1` rovna hodnotě `uspory` instance `prvek2`,
    - zápornou hodnotu, pokud je hodnota `uspory` instance `prvek1` větší než hodnota `uspory` instance `prvek2`.

7.4.1.3. Třída `VypoctovaMatice`

Třída, ve které jsou prováděny hlavní výpočty Clark-Wrightova algoritmu. Obsahuje datové složky a vlastnosti potřebné pro výpočet algoritmu.

Datové složky a vlastnosti

`double[,] matice` – datová složka reprezentující náklady na přepravu v kilometrech.

`ArrayList maticeUspor` – datová složka reprezentující matici úspor, obsahuje prvky typu `prvekMsticeUspor`, prvky jsou v seznamu seřazeny pomocí třídy porovnání úspor.

`public double[,] Matice`, `public ArrayList MaticeUspor` – veřejné vlastnost poskytující dané datové složky.

`double[] kapacityVrcholu` – pole kapacit pro jednotlivé vrcholy.

`bool[] gamaVrcholu` – pole hodnot gama pro jednotlivé vrcholy.

`int[] cisloTrasy` – hodnota na i-tém indexu udává číslo trasy, vrcholu `i`.

`bool[] obsluhovanSamostatne` – hodnota `true` na i-tém indexu znamená, že vrchol `i` není zařazen do žádné trasy, hodnota `false` znamená, že vrchol `i` je zařazen do nějaké trasy.

`string[] nazevSilnice` – pole názvů silnic odpovídající jednotlivým vrcholům.

`int rozmer` – rozměr všech matic a polí.

`double maximalniDoba` – určuje maximální dobu jízdy vozidla.

Metody

`private void SpocitejLambda()` – projde matici nákladů a pro každý prvek spočítá úspory, které uloží do `MaticeUspor`. Po dokončení seznam setřídí, k setřídění používá třídu `porovnaniUspor`.

`private bool podminkaAlfa(int i, int j)` – testuje platnost podminky α (hotnota `gama` vrcholů s indexem `i` a `j` musí být 1).

`private bool podminkaBeta(int i, int j, double kapacita)` - testuje platnost podminky β (součet kapacit ve spojovaných trasách musí být menší než kapacita vozidel).

`private bool podminkaStejneTrasy(int i, int j)` – testuje, zda jsou vrcholy zařazeny ve stejné trase, pokud ano vraci `false`.

`private double Prepocti_KM_CAS(double km)` – vrací přepočtené kilometry na čas. Podle hodnoty proměnné `km` se liší průměrná rychlost:

- vzdálenost km je menší než 100 m - průměrná rychlost vozidla je 5 km/h,
- vzdálenost km je větší než 100 m ale menší než 250 m - průměrná rychlost vozidla je 10 km/h,
- vzdálenost km je větší než 250 m ale menší než 500 m - průměrná rychlost vozidla je 20 km/h,
- vzdálenost km je větší než 500 m ale menší než 1 km - průměrná rychlost vozidla je 25 km/h,
- vzdálenost km je větší než 1 km - průměrná rychlost vozidla je 30 km/h.

`private int[] NajdiMax()` – vrací indexy prvního prvku v `maticeUspor`, tento prvek ze seznamu vymaže.

`private void NastavMaxDobu(TypVozidla vozidlo)` - nastaví hodnotu datové složky `maximalniDoba` v závislosti na hodnotě `vozidlo.pocetVysypu`.

`private ISpravaTras ClarkWrightAlgoritmus(TypVozidla vozidlo)` – metoda pro výpočet Clark-Wrightova algoritmu (princip algoritmu popsán v kap. 6.2.3.).

`public ISpravaTras SpocitejTrasy(TypVozidla vozidlo)` – metoda, která nejprve volá metodu `SpocitejLambda`, potom metodu `NastavMaxDobu(vozidlo)` a nakonec metodu `ISpravaTras ClarkWrightAlgoritmus(vozidlo)`.

7.4.1.4. Třída `VypocetniPrvek`

Objekt této třídy reprezentuje jednu skupinu vrcholů se stejným typem kontejneru. Obsahuje distanční matici těchto vrcholů, pole původních indexů a seznam tras pro tyto vrcholy.

Datové složky a vlastnosti

`IVypocet matice` – matice pro výpočet Clark-Wrightova algoritmu.

`List<int> poleSbernychMist` – protože algoritmus počíta s lokálními indexy vrcholů, toto pole uchovává původní indexy vrcholů nutné k převedení indexů zpět na globální indexy vrcholů.

`ISpravaTras spravaTras` – trasy pro daný typ vozidla

`public IVypocet Matice`, `public List<int> PoleSbernychMist`, `public ISpravaTras spravaTras` – veřejné vlastnosti poskytující dané datové složky.

Metody

`public ISpravaTras Spocitej()` – volá metodu `SpocitejTrasy(TypVozidla vozidlo)` třídy `VypoctovaMatice` která vrací trasy vypočtené Clark-Wrightovým algoritmem.

7.4.1.5. Rozhraní `ISpravaVypoctu`

Rozhraní pro třídu `SpravaVypocetnichPrvku`. Poskytuje všechny metody pro výpočet tras.

7.4.1.6. Třída `SpravaVypocetnichPrvku : ISpravaVypoctu`

Třída, která zajišťuje rozdělení vrcholů do skupin dle typu a frekvence výběru kontejneru. Tyto skupiny vrcholů uchovává v seznamu prvků typu `VypocetniPrvek`.

Datové složky a vlastnosti

`List<VypocetniPrvek> prvky` – uchovává seznam prvků typu `VypocetniPrvek`, pro které spouští výpočet tras.

`DateTime datumVyberu` – uchovává datum výpočtu tras. V případě spuštění výpočtu podruhé ve stejný den se trasy nemohou počítat jako reálný výběr ale jako plánování, jelikož pro daný den již byly trasy algoritmem spočteny. Datum výběru jednotlivých kontejnerů, které měly být daný den vybrány, bylo přepsáno. Toto datum se při prvním výpočtu v daný den uloží do souboru.

Metody

`public SpravaVypocetnichPrvku()` – konstruktor, inicializuje datovou složku `prvky` a načte `datumVyberu` ze souboru. Pokud soubor neexistuje do `datumVyberu` přiřadí včerejší datum - `DateTime.Now.AddDays(-1)`.

`private void PridejPrvek(VypocetniPrvek p)`, `private void OdeberPrvek(VypocetniPrvek p)` – soukromé metody pro práci se seznamem `prvky`.

`public double SpocitejKM(int[] trasa, double[,] distancniMatice)` – metoda, která slouží k výpočtu celkového počtu kilometrů trasy.

`public bool RozdelVypocet(ISpravaSbernychMist sbernaMista, double[,] distancniMatice, SpravaTypKontejneru typyKontejneru, SpravaTypuVozidel typyVozidel, DateTime datum, bool plan)` – metoda, která rozděluje vrcholy do skupid dle typu kontejneru a dle frekvence výběru kontejneru. Nejprve testuje zda `datum` není sobota nebo neděle, v tyto dny se odpad nesváží. Dále načte ze souboru

`datumVyberu` a testuje, pouze v případě, že `plan` je `false`, jestli `datum` není rovno `datumVyberu`. Pokud ano, metoda končí a vrací `false`, protože trasy pro tento den již byly spočteny. Pokud ne, metoda pokračuje. Nejprve se do skupin rozdělí vrcholy a pro každou skupinu vrcholů se vypočítá distanční matice. Dále se danému typu kontejneru, přiřadí typ vozidla, který tento typ kontejneru sváží. Ze všech získaných atributů se vytvoří instance třídy `VypocetniPrvek`, který se vloží do seznamu `prvky`.

`private void PrevedIndexy()` – převede indexy vrcholů v trase každého prvku seznamu `prvky` na indexy, které mají vrcholy v globálním seznamu vrcholů.

`public List<ISpravaTras> SpocitejTrasy()` – pro každý prvek seznamu `prvky` volá metodu `Spocitej()`. Po spočtení tras volá metody `PrevedIndexy()`. Následně vrací seznam vypočtených tras.

7.4.2. Třídy pro realizaci Typů kontejnerů a typů vozidel

7.4.2.1. Třída `TypKontejneru`

Datové složky a vlastnosti

`String nazev` – název kontejneru.

`double obsah` – obsah kontejneru.

`double dobaObsluhy` – průměrná doba obsluhy.

`bool prirazen` – určuje, zda je typ kontejneru přirazen nějakému typu vozidla.

Dále obsahuje veřejné vlastnosti, které poskytují a nastavují dané datové složky.

7.4.2.2. Třída `SpravaTypKontejneru`

Datové složky a vlastnosti

`List<TypKontejneru> seznamTypuKontejneru` – seznam typů kontejnerů.

`public List<TypKontejneru> SeznamTypuKontejneru` – veřejná vlastnost poskytující datovou složku `seznamTypuKontejneru`.

`public int Pocet` – veřejná vlastnost poskytující počet prvků v seznamu.

Metody

`public bool PridejKontejner(TypKontejneru kontejner)` – pokud v seznamu `seznamTypuKontejneru` není, `přidá` kontejner do seznamu a vrací `true`, jinak vrací `false`.

`public void OdeberKontejner(TypKontejneru kontejner)` – odebere kontejner `kontejner` ze `seznamTypuKontejneru`.

`public void OdeberKontejner(int index)` – odebere kontejner na indexu `index` ze `seznamTypuKontejneru`.

`public bool UpravKontejner(TypKontejneru kontejnerOld, TypKontejneru kontejnerNew)` – najde v seznamu `kontejnerOld`, do jeho datových složek přiřadí datové složky `kontejnerNew` a vrací `true`, pokud nebyl `kontejnerOld` nalezen, vrací `false`.

`public TypKontejneru DejKontejner(int index)` – vrací typ kontejneru na indexu `index`.

`public void UlozDoSouboru()` – serializuje `seznamTypuKontejneru` do souboru.

`public void NactiZeSouboru()` – deserializuje `seznamTypuKontejneru` ze souboru.

7.4.2.3. Třída `TypVozidla`

Datové složky a vlastnosti

`List<TypKontejneru> typKont` – seznam typů kontejnerů, který daný typ vozidla sváží.

`double kapacita` – kapacita vozidla.

`String pocetVysypu` – průměrný počet výsypů vozidla.

`String nazev` – název vozidla.

`public List<TypKontejneru> TypKont, public double Kapacita, public String PocetVysypu, public String Nazev` – veřejné vlastnosti, které poskytují a nastavují dané datové složky.

7.4.2.4. Třída `SpravaTypuVozidel`

Datové složky a vlastnosti

`List<TypVozidla> typyVozidel` – seznam typů vozidel.

`public List<TypVozidla> TypyVozidel` – veřejná vlastnost poskytující satovou složku `typyVozidel`.

`public int Pocet` – veřejná vlastnost poskytující počet prvků seznamu.

Metody

`public bool PridejTypVozidla(TypVozidla vozidlo)` – pokud v seznamu `typyVozidel` není, přidá typ vozidla do seznamu a vrací `true`, jinak vrací `false`.

`public void OdeberTypVozidla(TypVozidla vozidlo)` – odebere typ vozidla `vozidlo` ze seznamu `typyVozidel`.

`public void OdeberTypVozidla(int index)` – odebere typ vozidla na indexu `index` ze seznamu `typyVozidel`.

`public List<TypVozidla> OdeberTypVozidla(TypKontejneru tk)` – odebere ze seznamu `typyVozidel` typy vozidel, typ kontejneru `tk`, pokud typ vozidla obsahuje pouze tento typ kontejneru je vozidlo odebráno, vrací seznam odebraných prvků.

`public void UpravTypVozidla(TypVozidla tvOld, TypVozidla tvNew)` – najde typ vozidla `tvOld` a nahradí ho typem vozidla `tvNew`.

`public TypVozidla DejTypVozidla(int index)` – vrací typ vozidla na indexu `index`.

`public TypVozidla DejTypVozidla(string nazev)` – vrací typ vozidla s názvem `nazev`.

`public List<TypVozidla> DejTypyVozidel(TypKontejneru tk)` – vrací typy vozidel, které svážejí typ kontejneru `tk`.

`public void ZmenTypKontejneru(TypKontejneru kontejnerOld, TypKontejneru kontejnerNew)` – u všech typů vozidel obsahujících typ kontejneru `kontejnerOld`, změní tento typ kontejneru na `kontejnerNew`.

`public void UlozDoSouboru()` – serializuje `typyVozidel` do souboru.

`public void NactiZeSouboru()` – deseriallizuje `typyVozidel` ze souboru.

7.4.3. Třídy pro správu sběrných míst

7.4.3.1. Třída `MajitelKontejneru`

Obsahuje údaje o majiteli kontejneru.

Datové složky a vlastnosti

`String jmeno` – jméno majitele kontejneru.

`String prijmeni` – příjmení majitele kontejneru (v případě, že kontejner je registrován na firmu, jsou tyto pdaje prázdné).

`String nazevFirmy` – název firmy, která si objednala kontejner (v případě, že kontejner je registrován na soukromou osobu, je tento údaj prázdný).

`String ulice; int cp; String mesto; int psc` – údaje o umístění konterjneru.

Dále obsahuje veřejné vlastnosti, které poskytují a nastavují dané datové složky.

Metody

`public MajitelKontejneru(String jm, String pr, String ul, int cp, String m, int psc, bool firma)` – konstuktor, který nastaví všechny složky třídy `MajitelKontejneru` parametry konstruktoru.

7.4.3.2. Rozhraní `IKontejner`

Rozhraní pro třídu `Kontejner`.

7.4.3.3. Třída `Kontejner : IKontejner`

Datové složky a vlastnosti

`TypKontejneru typ` – typ kontejneru.

`int frekvence` – frekvence výběru kontejneru.

`DateTime posledniVyber` – datum posledního výněru kontejneru.

`DateTime vyberPlany` – datum, sloužící k výpočtu měsíčních plánů, při plánování se nastavuje toto datum a datum posledního výberu zůstává nezměněné.

Dále obsahuje veřejné vlastnosti, které poskytují a nastavují všechny datové složky třídy.

Metody

`public Kontejner(TypKontejneru typKont, int frekv)` – konstruktor, nastaví typ na na `typKont`, `vyberPlany` nastaví na `DateTime.Now`, frevnenci nastaví na `frekv` a podle frekvence `frekv` uloží datum poslendího výběru konkrétní datum. Pokud je parametr `frekv` „obden“, do data posledního výběru nastaví datum o dva dni zpět, pokud je parametr `frekv` „1xtýdně“ nastaví datum posledního výběru o 7 dní zpět atd.

7.4.3.4. Rozhraní `ISberneMisto`

Rozhraní pro třídu `SberneMisto`.

7.4.3.5. Třída `SberneMisto`

Datové složky a vlastnosti

`Point souradnice` – souřadnice sběrného místa.

`string nazev` – název sběrného místa, automaticky se vytváří z názvu ulice a čísla popisného datové složky `majitel`.

`MajitelKontejneru majitel` – majitel kontejneru v tomto sběrném místě.

`DateTime datumRegistrace` – datum registrace.

`Dictionary<TypKontejneru, IKontejner> kontejner` – seznam kontejnerů typu Dictionary. Klíčovým prvkem je typKontejneru, z čehož vyplývá, že v daném sběrném místě nemohou být dva kontejnery stejného typu.

Dále obsahuje veřejné vlastnosti, které poskytují a nastavují všechny datové složky třídy.

Metody

`public SberneMisto()` – implicitní konstruktor potřebný pro XML serializaci.

`public SberneMisto(Point souradnice)` – konstruktor, který se používá při vytváření sběrného místa kliknutím na plochu.

`public SberneMisto(string nazev)` – vytvoří sběrné místo s určitým názvem, ostatní datové složky jsou prázdné, používá se při testování, zda již sběrné místo s daným názvem neexistuje.

`public bool NastavPosledniVyber(DateTime datum, TypKontejneru tk, bool plan)` – nastavuje datum výběru všech kontejnerů, které jsou typu `tk` v dném sběrném místě. Pokud je `plan false`, podle frekvence a data posledniho vyběru, nastaví datum posledního výběru `posledníVyber`. Pokud je plan true, stejným způsobem nastavuje `vyberPlany`.

Frekvence:

- „denně“ – testuje zda byl poslení výber proveden před více jak 22 hodinami. Pokud ano, nastaví datum posledního výberu a vrací `true`, jinak vrací `false`,
- „obden“ - testuje zda byl poslení výběr proveden před více jak 46 hodinami a jestli je pondělí, středa nebo pátek. Pokud ano, nastaví datum posledního výberu a vrací `true`, jinak vrací `false`,

- „1 x týdně“ - testuje zda byl poslení výber proveden před více jak 166 ((7*24) – 2) hodinami a jestli je úterý. Pokud ano, nastaví datum posledního výberu a vrací `true`, jinak vrací `false`,
- „1x 14 dní“ - testuje zda byl poslení výber proveden před více jak 334 ((14*24) – 2)hodinami a jestli je druhý nebo čtvrtý čtvrtek v měsíci. Pokud ano, nastaví datum posledního výberu a vrací `true`, jinak vrací `false`,
- „měsíčně“ - testuje zda byl poslení výber proveden před tolika dny kolik měl měsíc z data posledního výběru a jestli je první čtvrtek v měsíci. Pokud ano, nastaví datum posledního výberu a vrací `true`, jinak vrací `false`.

Hodnoty jsou porovnávány vždy s časem o dvě hodiny menším než odpovídá dané frekvenci. Je to z důvodu rezervy pro různý čas spoustění aplikace.

`public Dictionary<TypKontejneru, IKontejner> DejKontejnery(TypVozidla tv)` – vraci kontejnery daného sběrného místa, které odpovídají typu vozidla `tv`.

`public void ReadXml(System.Xml.XmlReader reader)` – slouží ke XML deserializaci sběrného místa.

`public void WriteXml(System.Xml.XmlWriter writer)` – slouží ke XML serializaci sběrného místa.

7.4.3.6. Rozhraní `ISpravaSbernychMist`

Rozhraní pro třídu `SpravaSbernychMist`. Poskytuje metody pro správu sběrných míst.

7.4.3.7. Třída `SpravaSbernychMist`

Datové složky a vlastnosti

`List<ISberneMisto> sbernaMista` – seznam sběrných míst.

`public List<ISberneMisto> SeznamSbernychMist` – veřejná vlastnost poskytující seznam `sbernaMista`.

Metody

`public void UpravKontejnery(TypKontejneru tkOld, TypKontejneru tkNew)` – u všech sběrných míst obsahujících typ kontejneru `tkOld` změní na typ `tkNew`. Jestliže je `tkNew` null odebere ze sběrných míst `tkOld`.

`public void UlozitNastaveni()` – serializuje `sbernaMista` do souboru.

`private void NacistNastaveni()` - deserializuje `sbernaMista` ze souboru.

`public void RestartDataVyberuJedenDen()` – všem kontejnerům nastaví datum posledního výběru o jeden den dříve.

`public void RestartDataPlanu()` – všem kontejnerům nastaví `vyberPlany` na dnešní datum.

`public bool OdeberSberneMisto(ISberneMisto stanice)` – odebere sběrné místo `stanice` ze seznamu sběrných míst. Po odebrání volá metodu `UlozitNastaveni()`.

`public void PridejSberneMisto(ISberneMisto stanice)` – přidá sběrné místo `stanice` do seznamu sběrných míst. Po přidání volá metodu `UlozitNastaveni()`.

`public bool JeVSeznamu(ISberneMisto sm)` – vrací `true` pokud je sběrné místo `sm` v seznamu sběrných míst. Jinak vrací `false`.

`public void ReadXml(System.Xml.XmlReader reader)` – slouží ke XML deserializaci sběrných míst.

`public void WriteXml(System.Xml.XmlWriter writer)` – slouží ke XML serializaci sběrných míst.

7.4.4. Třídy pro práci s trasami

7.4.4.1. Rozhraní `ITrasa`

Rozhraní pro třídu `Trasa`.

7.4.4.2. Třída `Trasa`

Třída reprezentující trasu vozidla. Trasa je zde uložená jako seznam indexů vrcholů. Trasa na začátku ani na konci neobsahuje index depa. Pro správnou reprezentaci trasy je nutné cestu z depa do prvního vrcholu a z posledního vrcholu do depa přidat manuálně.

Datové složky a vlastnosti

`List<int> seznamVrcholu` – indexy vrcholů trasy.

`IVozidlo vozidlo` – vozidlo přiřazené dané trase.

`bool zobrazena` – udává, zda je trasa zobrazena.

`String nazev` – název trasy.

`Color barva` – barva, kterou je trasy vykreslena.

`double dobaJizdy` – doba jízdy.

Dále obsahuje veřejnou vlastnost, která poskytuje datovou složku `seznamVrcholu` a veřejné datové vlastnosti, které poskytují a nasstavují ostatní datové složky třídy.

Metody

`public Trasa(int[] poleVrcholu, Color barva)` – vytvoří trasu o dvou vrcholech.

`public void PridejVrchol(int index)` – přidá vrchol do seznamu vrcholů.

7.4.4.3. Rozhraní `ISpravaTras`

Rozhraní pro třídu `SpravaTras`. Poskytuje metody pro tvorbu tras a metody pro zpřístupnění tras.

7.4.4.4. Třída `SpravaTras : ISpravaTras`

Třída, reprezentující vypočtené trasy pro jeden typ kontejneru. Obsahuje seznam tras a metody pro práci s trasami.

Datové složky a vlastnosti

`Color[] barvy` – pole výrazných barev pro zobrazení tras.

`TypVozidla vozidlo` – typ vozidla obsluhující dané trasy.

`List<ITrasa> seznamTras` – seznam vypočtených tras.

`public int PocetTras` – veřejná vlastnost poskytující počet tras v `seznamTras`.

Metody

`public int PridejTrasuDvaVrcholy(int[] indexyVrcholu, int indexBarvy)` – vytvoří trasu ze dvou vrcholů, jejichž indexy jsou v poli indexyVrcholu a přidá ji do seznamu tras.

`public int PridejTrasuJedenVrchol(int indexVrcholu, double casJizdy, int indexBarvy)` – vytvoří trasu obsahující jeden vrchol (vrchol je obsluhován samostatnou jízdou vozidla) a přidá ji do seznamu tras.

`public int SpojTrasy(int trasa1, int trasa2, int[] indexyVrcholu, int indexBarvy)` – metoda pro spojení dvou tras do jedné. Parametr `indexyVrcholu` udává, přes jaké dva vrcholy mají být trasy spojeny.

`public void PridejVrchol(int indexVrcholu, int indexTrasy, int indexNavazujicihoVrcholu)` – na konec existující trasy přidá vrchol. Podle parametru `indexNavazujicihoVrcholu` se rozhoduje, na jaký konec trasy je vrchol připojen.

`public int[] DejTrasu(int index)` – vrací pole indexů vrcholů dané trasy.

`public ITrasa VratTrasu(int index)` – vrací trasu na indexu `index`.

`public ITrasa DejTrasuVozu(IVozidlo vuz)` – vrací trasu, která je přiřazená vozu `vuz`.

7.5. Práce s programem

V této kapitole budou popsány základní postupy práce s programem, ovládací prvky a funkce programu.

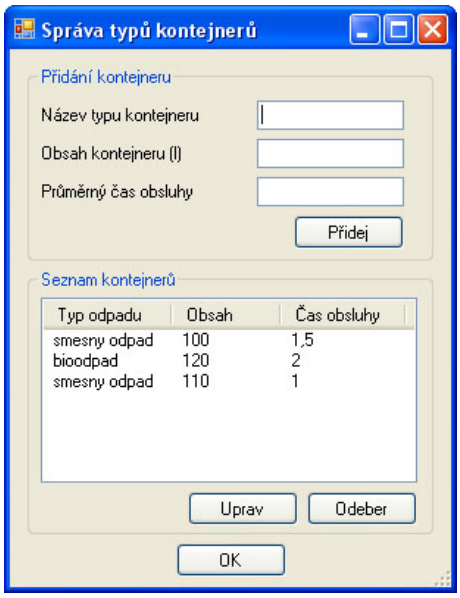
Po spuštění programu se otevře hlavní okno aplikace, jehož screenshot je v příloze 9. Uprostřed programu se zobrazuje mapa, na pravé straně se nachází informační panel, zobrazující informace o datu výpočtu a vypočtených trasách. V horní části se nachází menu a pod ním panel rychlé volby. Pro účely výpočtu tras a zobrazování plánů a statistik zde budou popsány hlavně postupy jednotlivých činností, nutných ke správné funkčnosti programu.

7.5.1. Postup přidání sběrného místa

Při přidávání sběrného místa je nutné mít nadefinované typy kontejnerů, které se potom přidávají do sběrného místa. Pro přidání typů kontejnerů slouží menu Správa ►Správa typů kontejnerů. Toto menu je zobrazeno v příloze 9 pod číslem 1.

7.5.1.1. Přidání typu kontejneru

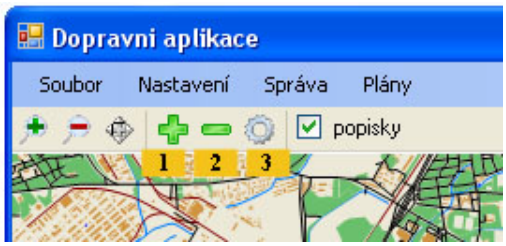
Formulář pro přidání typu kontejneru je na obrázku číslo 11. Po vyplnění názvu a obsahu kontejneru se stiskem tlačítka Přidej kontejner přidá do seznamu kontejnerů. Definované typy kontejnerů lze upravovat a odebírat. Po vybrání typu kontejneru v seznamu typů kontejnerů a stisknutí tlačítka Uprav, se vyplní pole název a obsah typu kontejneru daty vybraného typu kontejneru. Po potřebných úpravách a opětovném stisknutí tlačítka Přidej, se upravený typ kontejneru uloží. Typ kontejneru se změní v celé aplikaci. Pro vymazání kontejneru, stačí vybrat typ kontejneru ze seznamu a stisknout tlačítko Odeber. Pro ukončení dialogu stiskněte tlačítko OK.



Obrázek 11 – formulář pro přidání typu kontejneru [zdroj: Karlíková]

7.5.1.2. Přidání sběrného místa

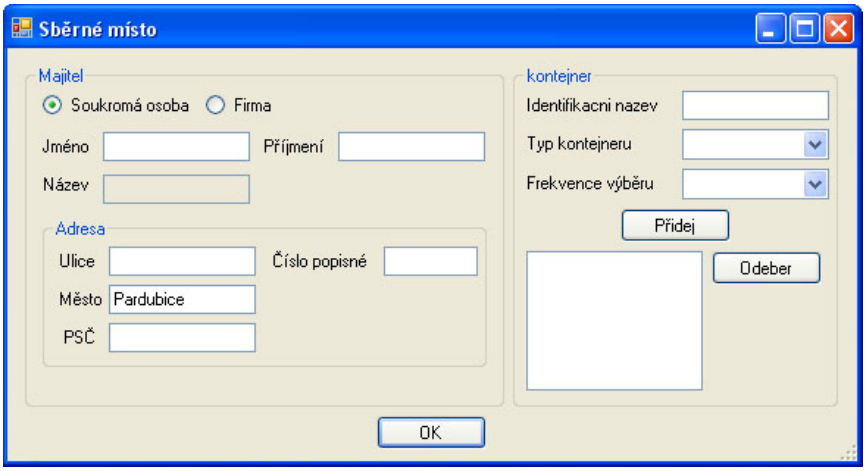
Pro přidání sběrného místa je nejprvé nutné zvoli v rychlém menu volbu pro přidávání vrcholů. Na obrázku 12 je znárorněno hlavní menu aplikace a menu rychlé volby. Pod číslem 1 se nachází volba přidání sběrného místa, pod volbou 2 se nachází odebrání sběrného místa a pod číslem tři úprava sběrného místa.



Obrázek 12 – menu a menu rychlé volby [zdroj: Karlíková]

Jako první je třeba v aplikaci vytvořit depo. Depo se vytváří vždy jako první vrchol. Po zvolení rychlé volby 1- přidání sběrného místa, stačí kliknout na mapu na místo kam chceme depo vložit. Zobrazí se dialog pro přídání sběrného místa. Tento dialog je zobrazen na obrázku 13. Je rozdělen na dvě části. První část se týká majitele daného sběrného místa, ve druhé části se vyplňují typy kontejnerů a frekvence výběru. Vpravo nahoře je umístěná kolonka Identifikační název. Vyplňuje se automaticky z údajů zadaných ve formuláři, proto je nutné tyto údaje vyplňovat správně. V případě vkládání depa, není možné vyplnit pravou stranu formuláře, která se týká kontejnerů, protože v depu žádné kontejnery nejsou.

V případě vkládání ostatních vrcholů je postup úplně stejný, jen je nutné správně vyplnit i položky formuláře týkající se informací o kontejnerech v daném sběrném místě. Pro vypnění údajů o typu kontejneru a frekvenci výběru stačí stisknout šipku a rozbalí se nabídka možností, které jsou na výběr. V kolonce typ kontejneru se zobrazí definované typy kontejnerů. Stiskem tlačítka Přidej se tento kontejner přidá ke sběrnému místu. Tlačítko Odeber slouží k odebírání již přiřazených kontejnerů. Po vyplnění všech údajů ve formuláři stačí stisknout tlačítko OK, formulář se zavře a sběrné místo se zobrazí na mapě.



Obrázek 13 – formulář pro přidání sběrného místa [zdroj: Karlíková]

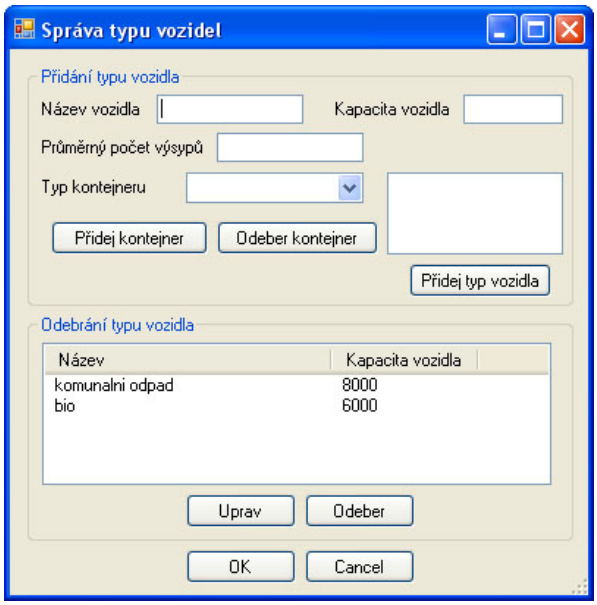
7.5.2. Postup přidání typů vozidel a tvorba vozového parku

Pro přidání typů vozidel je k dispozici menu Správa ►Správa typů vozidel. Toto menu je zobrazeno v příloze 9 pod číslem 2.

7.5.2.1. Přidání typu vozidla

Formulář pro přidání typu vozidla je na obrázku číslo 14. Je rozdělen na dvě části. Horní část obsahuje údaje o právě vkládaném typu vozidla, ve spodní části se nachází seznam již vložených typů vozidel a tlačítka pro jejich správu. Je nutné vyplnit název, kapacitu a průměrný počet výsypů vozidla a z nabídky typů kontejnerů vybrat typy kontejnerů, které toto vozidlo sváží. Typy kontejnerů se přidávají tlačítkem Přidej kontejner a odebírají tlačítkem Odeber kontejner. Tlačítkem Přidej se definovaný typ vozidla přidá do seznamu vozidel ve spodní části formuláře. Tlačítky Uprav a Odeber lze upravovat a odebírat již vložené typy vozidel. Po výběru typu vozidla ze seznamu a stisknutí tlačítka Uprav, se údaje o

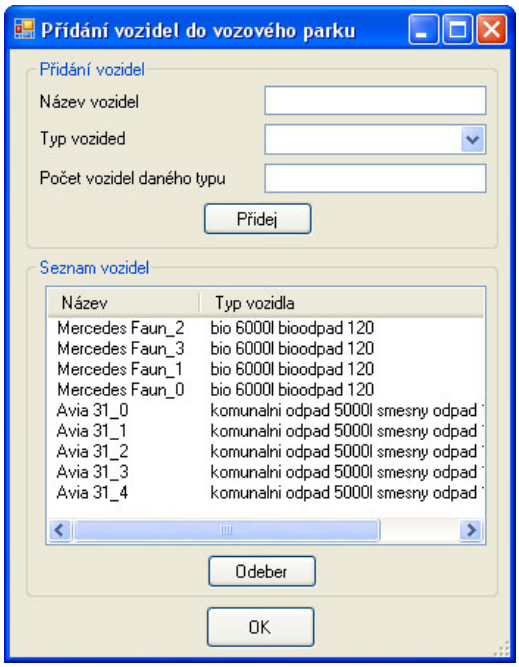
vybraném typu zobrazí v horní části formuláře. Tyto údaje mohou být upraveny a zpět uloženy do seznamu typů vozidel stiskem tlačítka Přidej.



Obrázek 14 – formulář pro přidání typů vozidel [zdroj: Karlíková]

7.5.2.2. Přidání vozidel

Pro přidání vozidel slouží volba menu Správa ►Správa vozidel. Toto menu je zobrazeno v příloze 9 pod číslem 3. Formulář pro přidání vozidla je na obrázku číslo 15. Vzhledem i funkčností se podobá formuláři pro vkládání typu vozidel. Je také rozdělen na dvě části. V horní části je nutné vyplnit údaje o vozidle, tzn. Jakého je vozidlo typu, název vozidla a pro jednoduchost je zde položka počet vozidel daného typu, která usnadňuje vkládání více vozidel stejného typu. Po přidání vozidla se toto vozidlo zobrazí ve spodní části formuláře. Rozdíl oproti formuláři pro vkládání typů vozidel je ten, že zde není tlačítko Uprav. Nachází se zde pouze tlačítko Odeber, které odebere vybrané vozidlo.



Obrázek 15 – formulář pro přidání vozidel [zdroj: Karlíková]

Tímto je nadefinované vše co je třeba pro spuštění výpočtu tras.

7.5.3. Výpočet tras

Pro výpočet a zobrazení tras slouží pravý panel v hlavním okně aplikace (viz. příloha 9). Pro výpočet tras slouží ovládací prvek kalendář nebo tlačítko Spočítej (obě tyto možnosti fungují úplně stejně, bude tedy popsán postup pomocí výběru dne v kalendáři). Po rozkliknutí tohoto prvku se zobrazí kalendář. Po výběru dne z kalendáře se automaticky spočítají trasy pro daný den. Pokud je vybraný den sobota nebo neděle, program ohlásí, že v tyto dny se odpad nesváží a trasy nespočítá.

Vypočtené trasy se zobrazí na mapě a v levém panelu ve dvou ovládacích prvcích zobrazených v příloze 9 pod čísly 4 a 5. V prvním z nich jsou zobrazena vozidla, která jsou přiřazena některé vypočteně trase, celkový počet kilometrů, který dané vozidlo najelo, čas, který jízda trvá a počet obsloužených sběrných míst. Ve druhém z nich je u každého auta zobrazen seznam sběrných míst, který daná trasa obsahuje. Každé sběrné místo je zobrazeno jako název ulice a číslo popisné místa, kde se nachází.

Toto zobrazení tras je velice přehledné, ale trasy které jsou vykreslené na mapě se mohou překrývat, proto jsou zde na výběr dvě možnosti zobrazení menšího počtu tras:

1. zobrazení tras určitého typu vozidla – pod ovládacím prvkem kalendář je možné, rozkliknutím šipky, zobrazit nabídku definovaných typů vozidel. Po vybrání jednoho z nich, se vykreslí a vypíšou pouze trasy daného typu vozidla.
2. zobrazení tras určitého vozidla – trasa, kterou jede konkrétní vozidlo, se dá zobrazit vybráním jednoho vozidla ze seznamu v ovládacím prvku, ve kterém jsou zobrazeny údaje o vozidle (v příloha 9 pod číslem 4). Po kliknutí na jedno z vozidel se zobrazí pouze trasa daného vozidla.

7.5.4. Uložení tras a statistiky

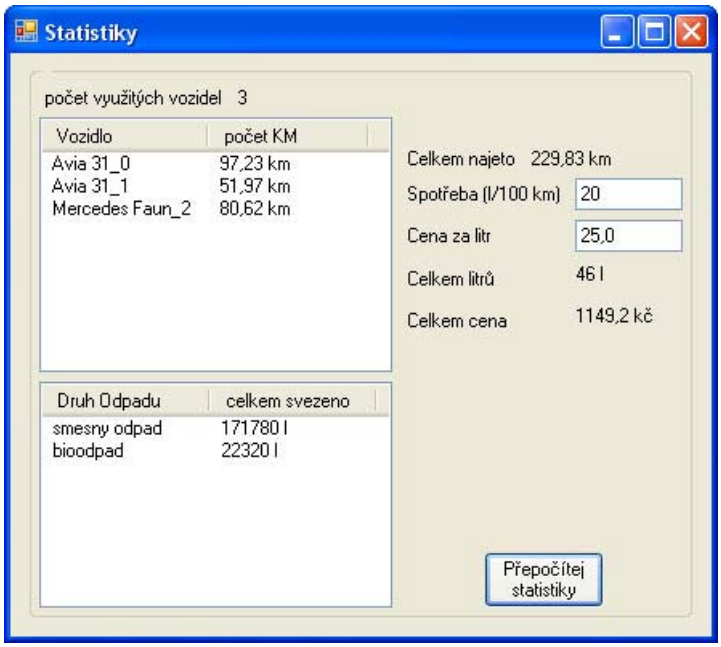
Vedle ovládacího prvku kalendář se nachází tlačítka Ulož trasy a Statistiky.

7.5.4.1. Uložení tras

Tlačítko Ulož trasy uloží trasy pro vybraný den do souboru ve formátu html. Pro soubory tohoto formátu je k dispozici css soubor `style.css`, který tento výstup zformátuje do podoby přehledných tabulek. Soubor je umístěn v adresáři Nastavení. V případě uložení souboru s trasami do jiného adresáře, je nutné soubor `style.css` přesunout do tohoto adresáře. Tabulka plánů tras pro jeden den je v příloze 11.

7.5.4.2. Statistiky

Pokud je výpočet tras počítán pro aktuální den, do adresáře statistiky se ukládá textový soubor s informacemi o trasách a vozidlech. Z těchto souborů jsou po vybrání měsíce a stisku tlačítka Statistiky tvořeny statistiky pro konkrétní měsíc. Tyto statistiky jsou zobrazeny v novém okně na obrázku 16. Na levé straně jsou pro každé vozidlo sečteny kilometry, které dané vozidlo najelo. Na pravé straně je celková ujetá vzdálenost v kilometrech, dále je zde průměrná spotřeba a cena za litr pohoných hmot. Z těchto hodnot se počítá celková spotřeba paliva a celkové měsíční náklady. Hodnoty v polích Spotřeba a Cena za litr se dají změnit. Po stisku tlačítka Přepočítej statistiky se podle nově zadaných hodnot přepočítají hodnoty – Celkem litrů a Celkem cena. Ve spodní části tohoto okna jsou dále umístěny informace o celkovém objemu svezeného odpadu. Tyto hodnoty jsou rozděleny podle typu odpadu.



Obrázek 16 – formulář pro zobrazení statistik [zdroj: Karlíková]

7.5.5. Plánování tras

Pro plánování tras je k dispozici zvlásštní dialogové okno jehož screenShot je v příloze 10. Toto okno se zobrazí stisknutí volby menu Plány ► Plány. V levé části se nachází ovládací panel a ve zbylé části tohoto dialogového okna je umístěn ovládací prvek, v němž se zobrazují naplánované trasy. Ovládací panel je rozdělen na dvě části. Horní část slouží k nastavení odbobí pro plánování tras, ve spodní části se nachází výpis statistik pro dané období.

Pro naplánování tras na určité období je nejprve nutné, v horní části okna, vybrat v kalendáři období od kdy a do kdy se mají trasy plánovat. Po stisknutí tlačítka Plánuj se trasy zobrazí v pravé části dialogového okna a zároveň se, v levé dolní části, zobrazí statistiky (obsah popsán v kapitole 7.5.4.2. Statistiky). Implicitním nastavením je zobrazení sběrných míst pod sebe. Tato volba poskytuje lepší přehled o tom která sběrná místa jsou obsažena v trase. Nad tlačítkem Plánuj je unístěna volba trasa na řádek. Po zaškrtnutí této volby bude každá trasa zobrazena na jednom řádku. Po změně této volby je nutné trasy přeplánovat – stisknout tlačítko Plánuj. Volba sběrných míst v jednom řádku poskytuje lepší přehled o trasách, jejich délce a počtu sběrných míst v trase. Pro uložení tras i statistik slouží tlačítko Uložení, které uloží naplánované trasy za zvolené období do souboru ve formatu html. Po zadání názvu souboru se vytvoří dva soubory `zadanyNazev_plany.html` a

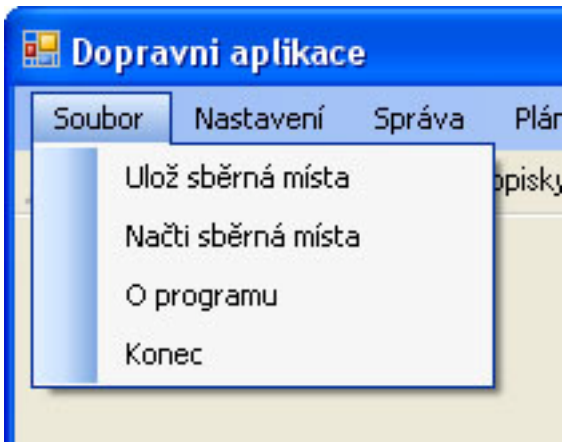
zadanyNazev_statistiky.html. Do souboru `zadanyNazev_plany.html` se uloží plány pro dané období v přehledných tabulkách a do souboru `zadanyNazev_statistiky.html` se uloží statistiky za dané období. Ukázka tabulky statistik pro jeden den je v příloze 12.

Stejně jako u ukládání tras v hlavním dialogovém okně, i zde je nutné k uloženým souborům přesunou soubor style.css, který výstup obou uložených souborů zformátuje.

7.5.6. Export a import sběrných míst

Pro snadnější zadávání sběrných míst je k dispozici volba Načti sběrná místa a Ulož sběrná místa. Tyto volby lze nalézt v menu Soubor znázorněném na obrázku 17.

Po stisknutí volby Ulož sběrná místa se všechna sběrná místa uloží do souboru formátu XML. Pro načtení sběrných míst musí být soubor XML ve stejném formátu, v jakém byl vyexportován. Po zvolení Načti sběrná místa a vybrání souboru se tato sběrná místa načtou a zobrazí na mapě. V případě, že soubor XML, ze kterého budou sběrná místa načítána bude obsahovat stejné sběrné místo, které již je načtené v aplikaci, toto sběrné místo se neuloží, přeskočí se a pokračuje se dále v načítání ostatních sběrných míst.



Obrázek 17 – import a export sběrných míst [zdroj: Karlíková]

8. Závěr

Záměrem práce bylo nastudovat a rozebrat problematiku svozně-rozvozných úloh a tuto teorii aplikovat na řešení úlohy svozu komunálního odpadu. Na základě získaných znalostí vytvořit softwarový nástroj řešící tuto problematiku.

Přínosy práce:

- teoretický přehled o metodách teorie grafů řešící problematiku svozu a rozvozu,
- vytvořený program pro optimalizaci tras svozu komunálního odpadu.

První část práce poskytuje přehled o metodách teorie grafů řešící nejen problém svozně-rozvozných úloh (okružních jízd) ale i další problémy, které s touto problematikou úzce souvisí. Nejprve jsou definovány potřebné pojmy z oblasti teorie grafů, jejichž znalost je pro danou problematiku důležitá. Dále jsou rozebrány metody pro hledání nejkratší cesty, metody čínského pošťáka a metody obchodního cestujícího. Práce se také okrajově věnuje problematice složitosti algoritmu, jelikož úlohy, zabývající se optimalizací okružních jízd se řadí mezi NP úplné problémy, což jsou problémy, pro které nelze nalézt optimální řešení v polynomiálním čase.

Po nastudování teoretického základu byla vytvořena aplikace, která na základě znalostí řešení svozně-rozvozných úloh a na základě znalostí o realizaci svozu komunálního odpadu v regionu Pardubice, optimalizuje svozné trasy. Aplikace umožňuje vypočítat trasy svozu odpadu na základě zadaných sběrných míst. Umožňuje vypočítat trasy ve zvolený den pro různé typy kontejnerů a různou frekvenci výběru jednotlivých kontejnerů a tyto trasy zobrazit a uložit ve formě přehledných tabulek. Vypočtené trasy přiřadí dostupným vozidlům.

Další důležitou funkčností programu je tvorba plánů tras pro zadané období. Takto naplánované trasy je možné zobrazit v dialogovém okně i s dalšími informace o trasách nebo je možné takto naplánované trasy uložit. Trasy se ukládají ve formátu HTML v přehledných tabulkách, kde jsou pro každý den vypsána vozidla a každému vozidlu je přiřazena trasa, která obsahuje vždy po sobě jdoucí seznam vrcholů (sběrných míst), kterými prochází. Sběrná místa jsou charakterizována vždy názvem ulice a číslem popisným, z čehož každý řidič pozná, kudy jeho trasa vede.

Program dále umožňuje počítat různé statistiky, jako jsou počet najetých kilometrů jednotlivých vozidel a z toho vyplývající spotřeba paliva, nebo objem svezeného odpadu. Tyto statistiky je možné počítat zpětně za jednotlivé měsíce nebo je možné je zobrazit při plánování tras pro dané období. Takto vytvořené statistiky potom nejsou počítány za celý měsíc, ale pouze za zvolené období.

Program vznikal jako souběžná práce dvou studentů a proto kromě plánování tras umožňuje i online sledování vozidel, obsluhujících dané trasy. Tím se z této aplikace stává nástroj, který je využitelný nejen pro optimalizaci tras, tedy snížení nákladů na svoz ale i pro kontrolu dodržování takto naplánovaných tras, což může být v dnešní době, kdy se omezuje plýtvání zdroji, velice žádanou službou.

Literatura

[1] Dijkstruv Algoritmus. *Wikipedie* [online]. 2008 [cit. 2009-03-02]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Dijkstr%C5%AFv_algoritmus>.

[2] Problém obchodního cestujícího. *Wikipedie* [online]. 2009 [cit. 2009-03-02]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Probl%C3%A9m_obchodn%C3%ADho_cestuj%C3%ADc%C3%ADho>.

[3] Hamiltonovská kružnice. *Seznam Encyklopedie* [online]. 2007 [cit. 2009-03-02]. Dostupný z WWW: <http://encyklopedie.seznam.cz/heslo/130344-hamiltonovska-kruznice>.

[4] NEČAS, Jiří. *Grafy a jejich použití*. Praha: SNTL - Nakladatelství technické literatury, 1978. 191 s.

[5] VOLEK, Josef. *Operační výzkum I.* Pardubice: Univerzita Pardubice, 2002. 111 s. ISBN 80-7194-410-6

[6] Asymptotická složitost. *Wikipedie* [online]. 2008 [cit. 2009-04-08]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Asymptotick%C3%A1_slo%C5%BEitost>

[7] NP-úplnost. *Wikipedie* [online]. 2009 [cit. 2009-04-08]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/NP-%C3%BAplnost>.

[8] KOZIOL, Jaroslav. *Optimalizace rozvozových tras*. [s.l.], 2000. 76 s. Vedoucí diplomové práce Doc. Ing. Josef Volek, CSc.

[9] Heuristické algoritmy. *Wikipedie* [online]. 2009 [cit. 2009-04-08]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Heuristick%C3%A9_algoritmy>.

[10] SCHMIDT, Jan. Problémy a algoritmy. *Wikia education* [online]. 2007 [cit. 2009-04-09]. Dostupný z WWW: <http://vzdelani.wikia.com/wiki/36PAA_Probl%C3%A9my_a_algoritmy#T.C5.99.C3.ADda_NP>.

[11] NĚMEC, Miloš. O P, NP a NPC úlohách. *Miloš Němec - osobní stránka* [online]. 30.10.2005 [cit. 2009-04-09]. Dostupný z WWW: <http://www.milosnemec.cz/clanek.php?50>.

[12] NEŠETŘIL, Jaroslav. *Teorie grafů*. [s.l.] Praha: SNTL - Nakladatelství technické literatury, 1979. 316 s.

[13] Eulerovský tah. *Wikipedie* [online]. 2009 [cit. 2009-04-15]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Eulerovsk%C3%BD_tah>.

[14] DEMEL, Jiří. *Grafy a jejich aplikace*. Praha: Academia, 2002. 160 s. ISBN 80-200-0990-6.

[15] *LOGICON Partner s.r.o. - logistika a poradenství* [online]. 2007 [cit. 2009-04-20]. Dostupný z WWW:< http://www.logicon.cz/>.

[16] *Tranis s.r.o., Tracker s.r.o.-Sledování vozidel aut Plánování trasy tras Monitorování Monitoring vozidel aut Plánovač tras trasy Kniha jízd Mýtné Mýto Výpočet mýtného Přeprava zvířat Systémy pro sledování a monitoring vozidel aut GPS trasa Navigace* [online]. 2005 [cit. 2009-04-20]. Dostupný z WWW: <http://www.tranis.cz/index.asp>.

[17] *O2 Business Solutions* [online]. [2008] [cit. 2009-04-20]. Dostupný z WWW: <http://www.o2bs.com/index.php?id=1>.

[18] Hamiltonovský graf. Wikipedie [online]. 2009 [cit. 2009-05-01]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Hamiltonovský_graf>.

[19] Travelling salesman problem. Wikipedia [online]. 2009 [cit. 2009-05-01]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Travelling_salesman_problem>.

[20] TSP : The Traveling Salesman Problem [online]. 2008 [cit. 2009-05-02]. Dostupný z WWW: <http://www.tsp.gatech.edu/index.html>.

[21] VRP Web [online]. 2007 [cit. 2009-05-02]. Dostupný z WWW: <http://neo.lcc.uma.es/radi-aeb/WebVRP/>.

[22] VOLEK, Josef. Okružní jízdy[online]. 2007 [cit. 2009-04-15].

[23] BRÁZDOVÁ, Markéta. *Využití optimalizačních metod k řešení svozných a rozvozných úloh*. Pardubice, 1998. 78 s. Vedoucí dizertační práce Josef Volek.

[24] Úvod to teorie grafů. [online]. [cit. 2009-04-08]. Dostupný z WWW: <uhk.xichtik.net/download.php?id=360&sid=69d8467499ce68694afd7511117b9102 - >.

[25] Druhy odpadů. Jablonec nad Nisou: oficiální stránky města [online]. 2005 [cit. 2009-05-06]. Dostupný z WWW: <http://www.mestojablonec.cz/cs/zivotni-prostredi/odpady/druhy-odpadu/>.

[26] GREINER, Karel. 7. přednáška – Rozhraní – Pokračování. Pardubice, 2007. 12 s. Univerzita Pardubice. Přednáška.

[27] EKOKOM [online]. c2009 [cit. 2009-05-07]. Dostupný z WWW: <http://www.ekokom.cz/>.

Seznam obrázků

Seznam tabulek

Seznam příloh

Příloha 1 – Distanční matice vrcholů

c_{ij}		v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}
	v_0	0	16	14	13	20	16	11	8	5	8	6	16	7	12	11
6	v_1	16	0	10	15	22	25	27	8	13	18	22	14	23	22	27
10	v_2	14	10	0	5	12	15	20	14	9	8	20	22	21	26	25
9	v_3	13	15	5	0	7	12	17	19	14	5	19	27	20	25	24
12	v_4	20	22	12	7	0	5	10	27	21	26	15	24	27	32	21
7	v_5	16	25	15	12	5	0	5	23	25	7	10	32	23	28	26
9	v_6	11	27	20	17	10	5	0	19	16	12	5	27	18	23	16
8	v_7	8	8	14	19	27	23	19	0	5	16	14	8	15	16	19
5	v_8	5	13	9	14	21	25	16	5	0	13	11	13	12	17	16
7	v_9	8	18	8	5	26	7	12	16	13	0	14	24	15	20	19
11	v_{10}	6	22	20	19	15	10	5	14	11	14	0	22	13	18	17
6	v_{11}	16	14	22	27	24	32	27	8	13	24	22	0	13	8	20
11	v_{12}	7	23	21	20	27	23	18	15	12	15	13	13	0	5	7
8	v_{13}	12	22	26	25	32	28	23	16	17	20	18	8	5	0	12
9	v_{14}	11	27	25	24	21	26	21	19	16	19	17	20	7	12	0

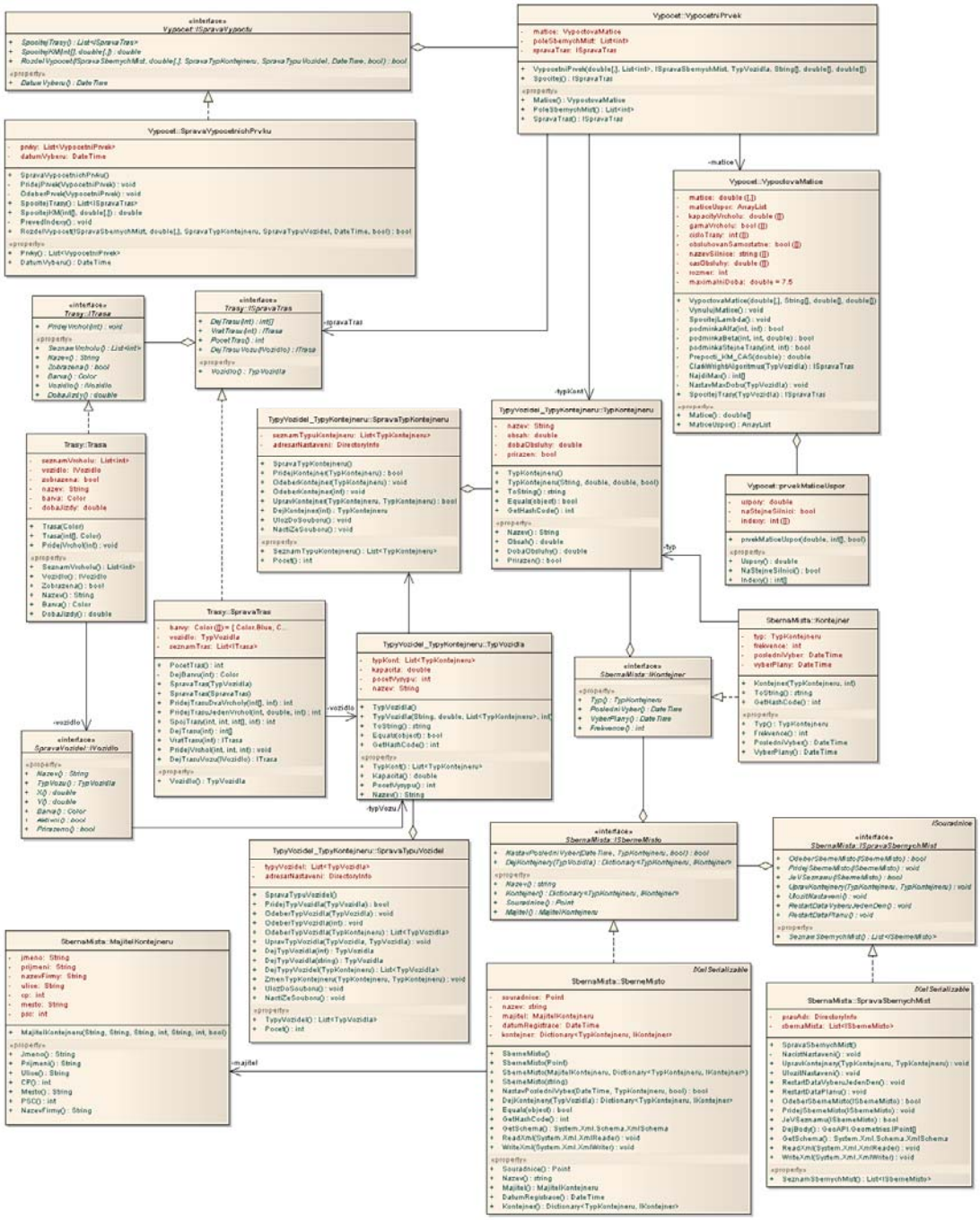
Příloha 2 – Dopis

Dobrý den,
jsem studentka Univerzity Pardubice a zpracovávám diplomovou práci týkající se svozu odpadu. Do své práce dělám malý průzkum. Zjišťuji, jaké programy používají společnosti zabývající se svozem odpadu. Proto jsem Vás chtěla požádat, zda byste mi nemohl(a) napsat, jestli používáte nějaké softwarové nástroje pro správu vozového parku, pro plánování tras svozu odpadu jednotlivých vozidel nebo pro monitoring vozidel. Pokud ano, které. Pokud ne, uvažujete do budoucna o pořízení takovéhoto produktu?

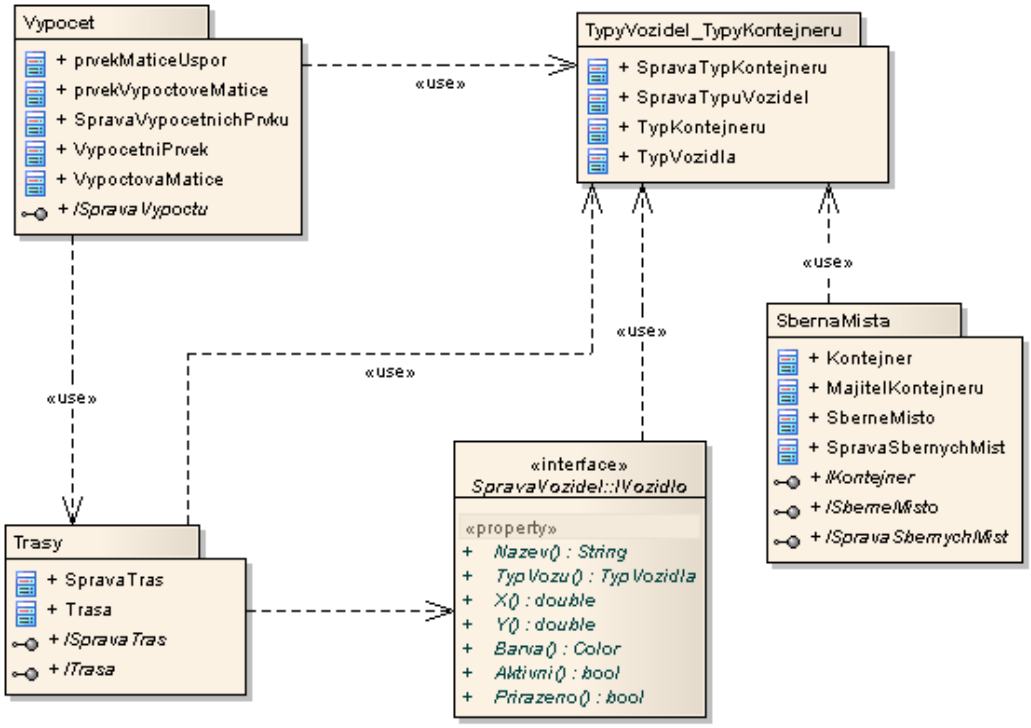
Předem děkuji za poskytnuté informace.

S pozdravem Zuzana Karlíková

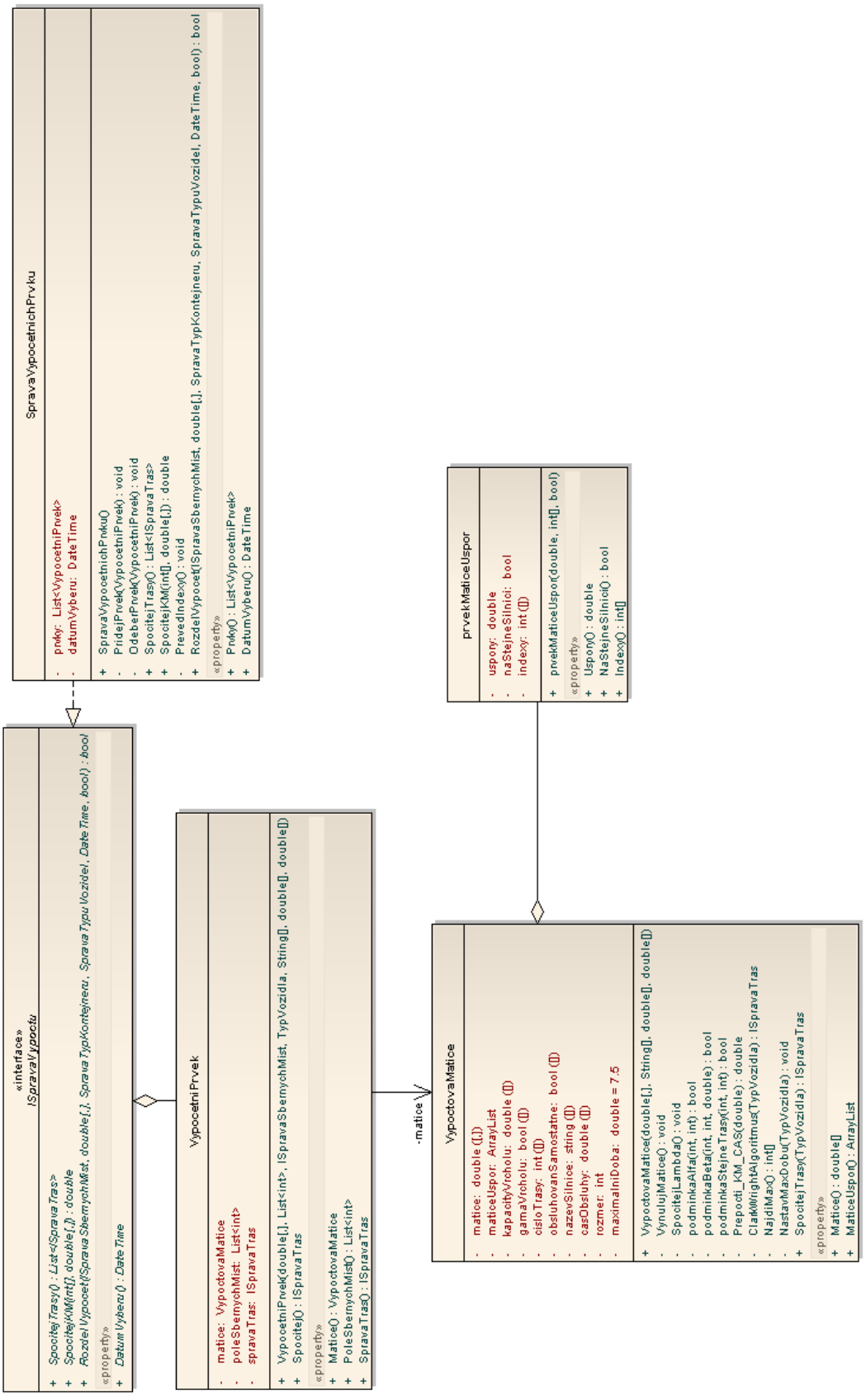
Příloha 3 – Kompletní UML diagram tříd



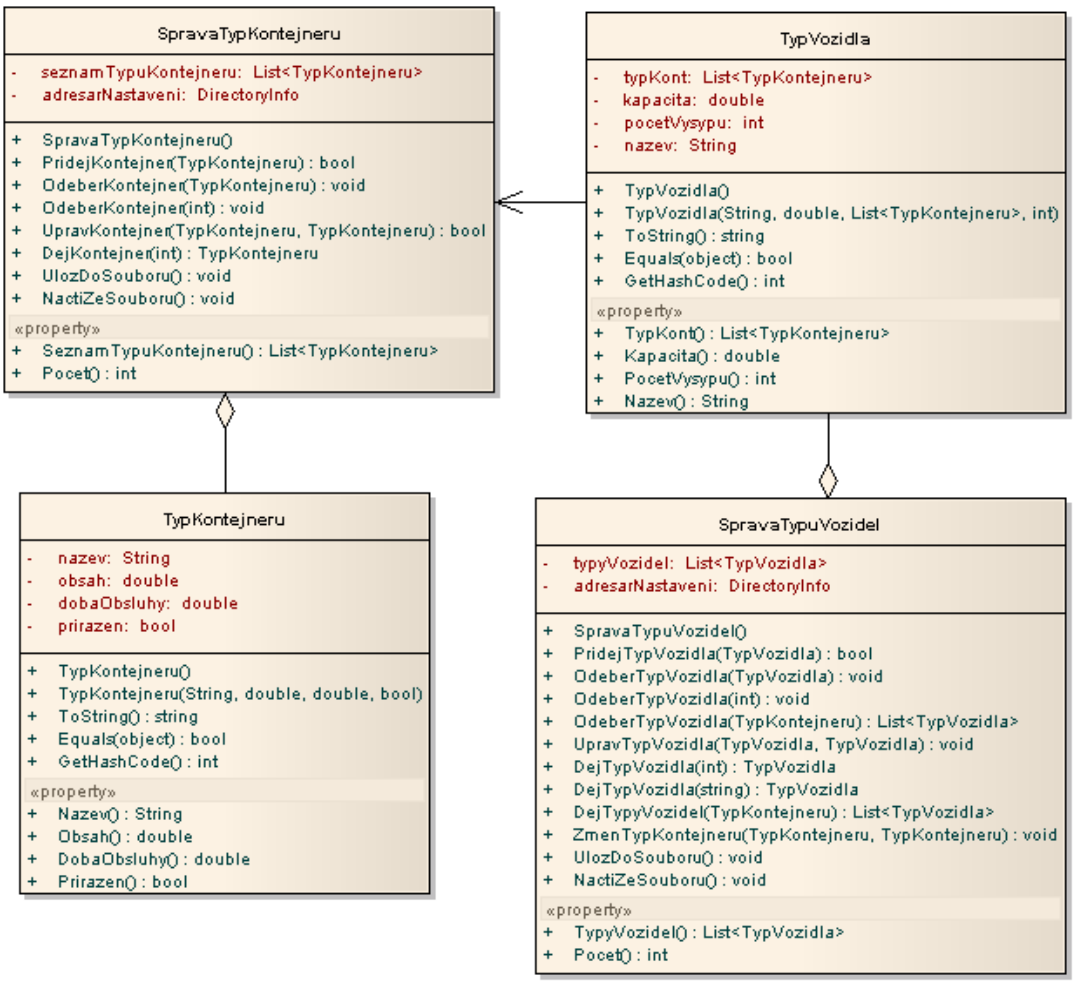
Příloha 4 - UML diagram tříd rozdělený do balíčků



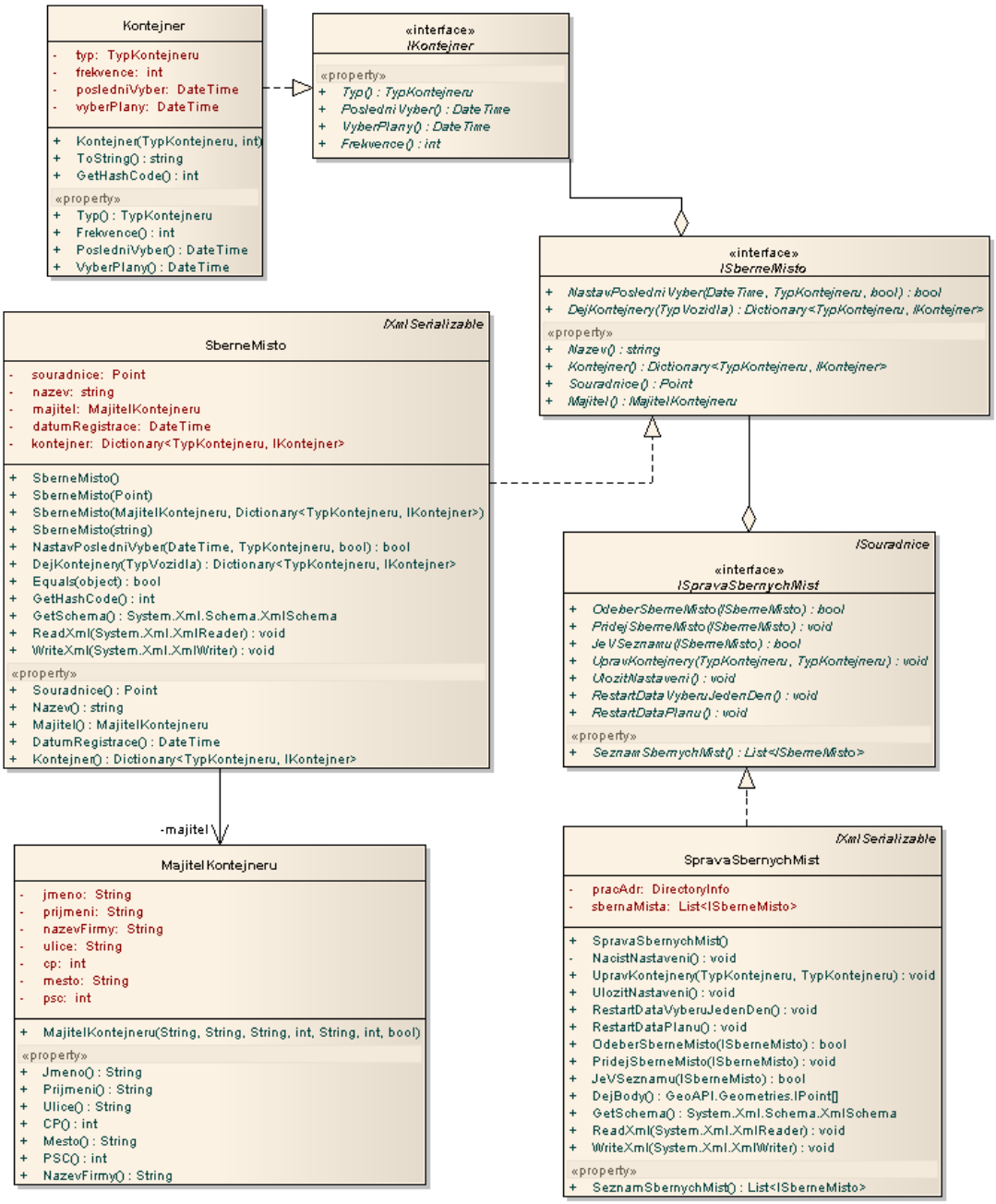
Příloha 5 – UML diagram



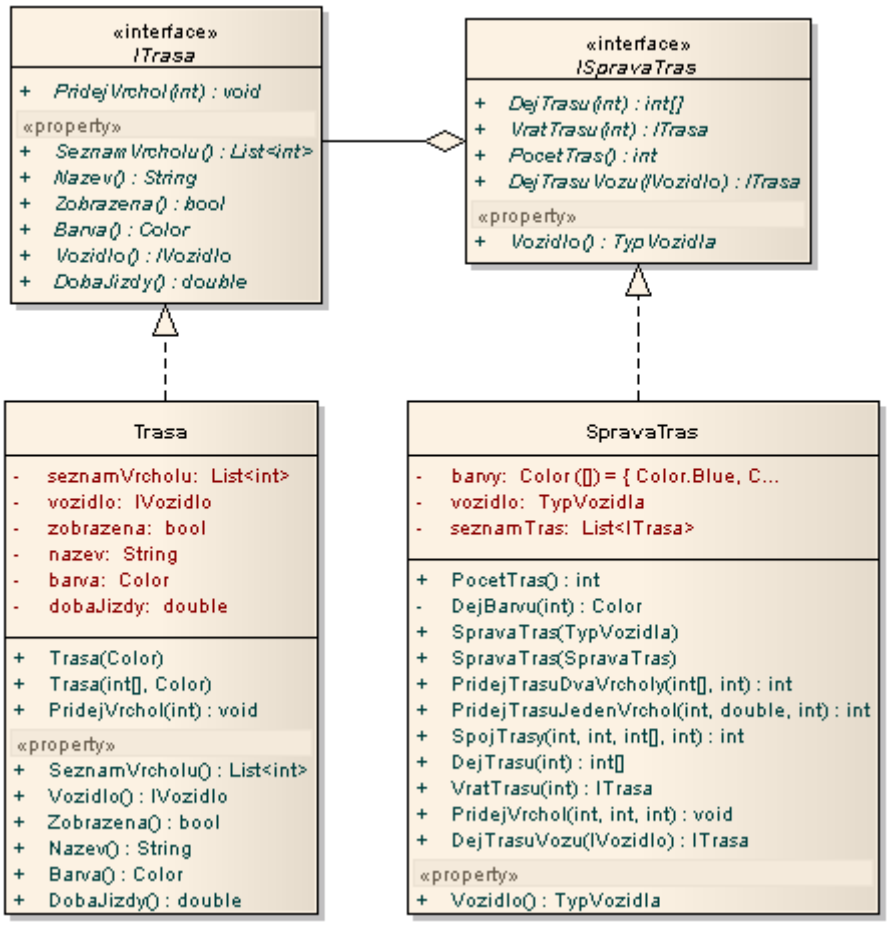
Příloha 6 - UML diagram



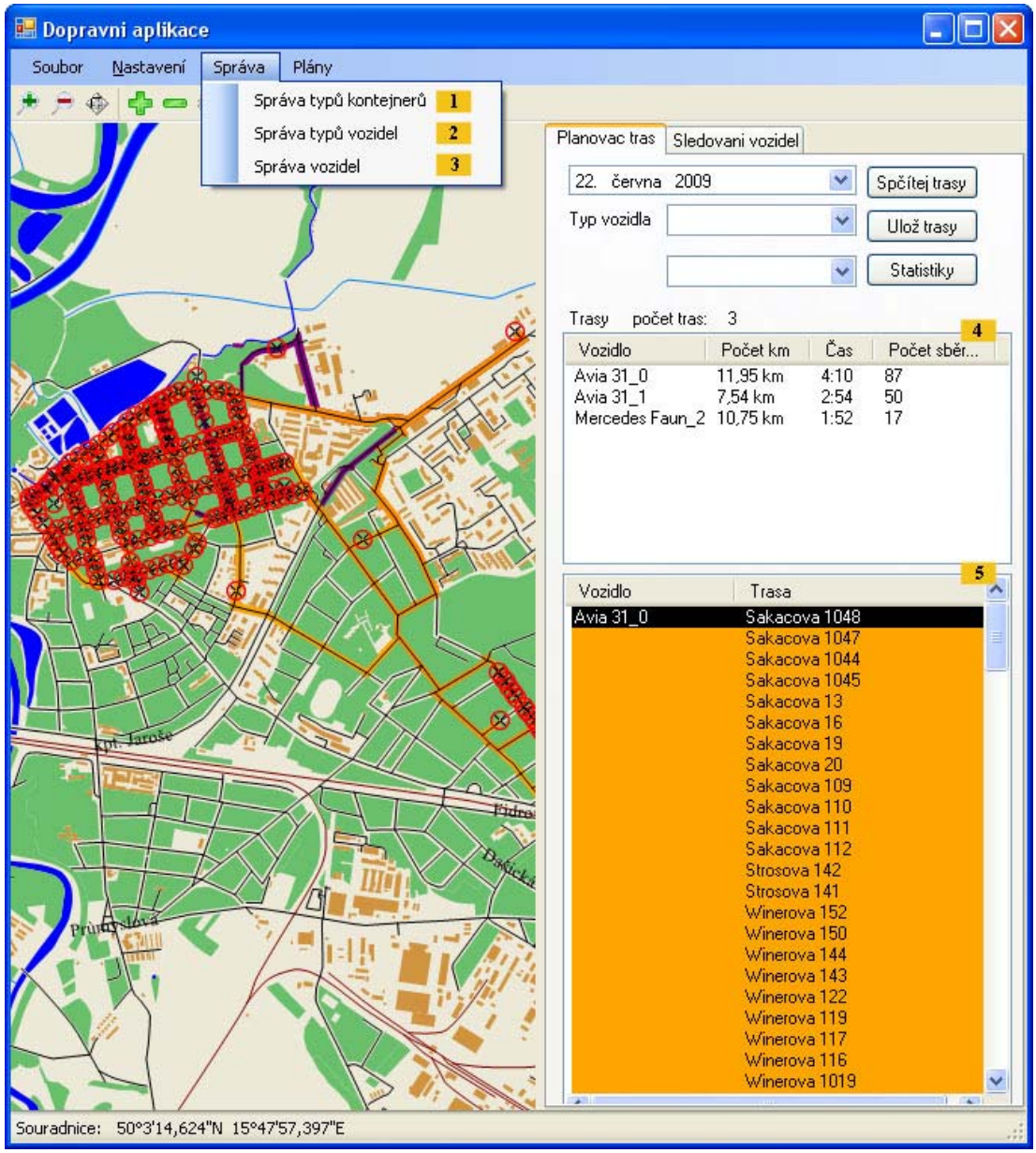
Příloha 7 – UML diagram



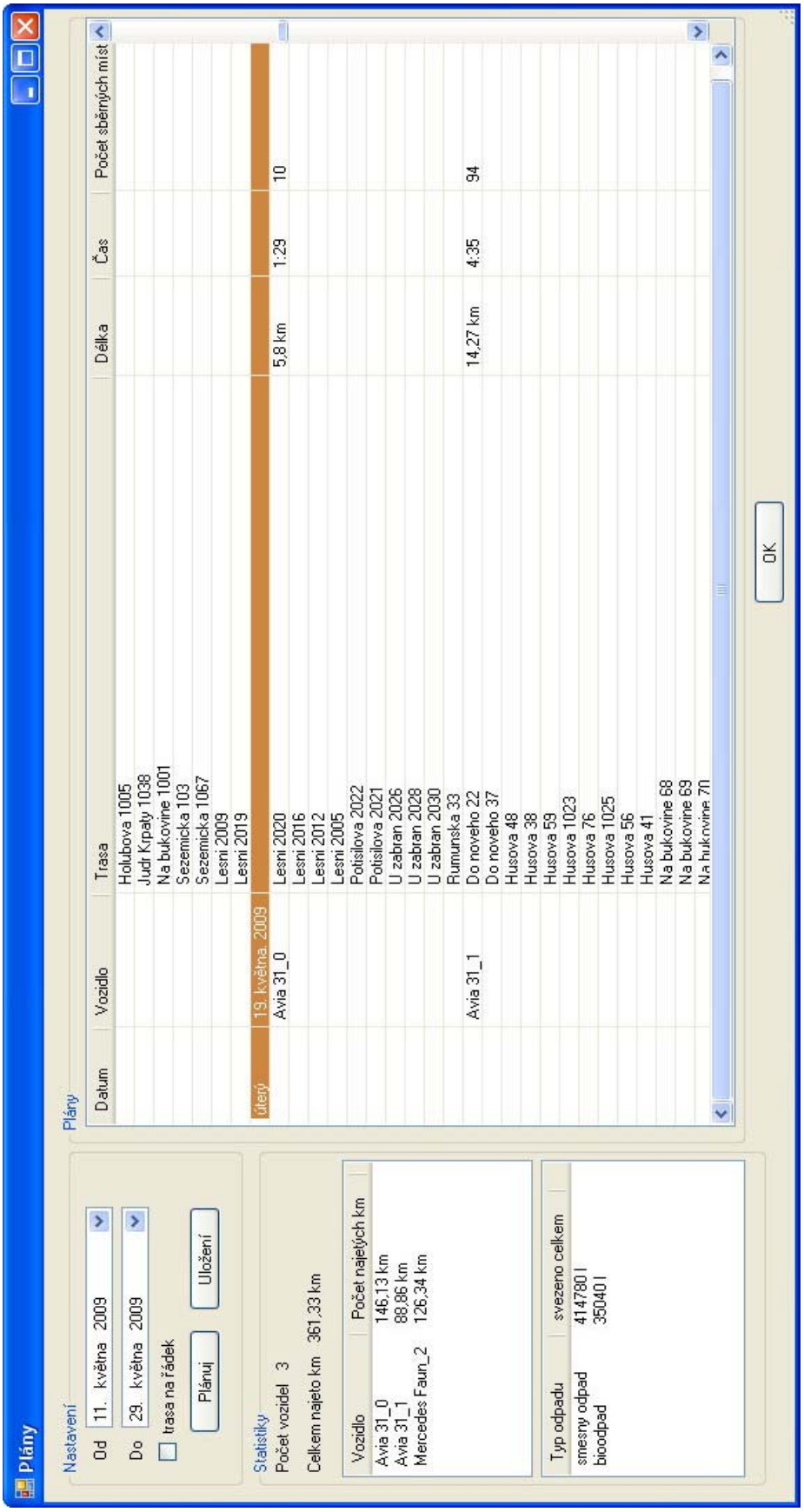
Příloha 8 – UML diagram



Příloha 9 – Hlavní okno aplikace



Příloha 10 – Dialogové okno pro plánování tras



Příloha 11 – Plány tras

středa - 1. července. 2009	
Trasa pro vozidlo Iveco_2 (komunální odpad)	
Počet kilometrů:	11,03 km
Doba jízdy:	4:03
Iveco_2	Spojilska 3112
	Spojilska 3108
	Spojilska 3107
	Lesni 2028
	Lesni 2029
	Lesni 2032
	Lesni 2031
	Lesni 2024
	Lesni 2022
	Lesni 2021
	Lesni 2019
	Lesni 2018
	Lesni 2017
	Lesni 2015
	Lesni 2014
	Lesni 2013
	Lesni 2011
	Lesni 2010
	Lesni 2009
	Lesni 2008
	Lesni 2007
	Lesni 2006
	Lesni 2002
	Lesni 2001
	Lesni 2003
	Potisilova 2021
	Potisilova 2020
	U zabran 2027
	U zabran 2029
	U zabran 3013
	U zabran 3014
	Musilkova 3018
	Musilkova 3017
	Musilkova 3015
	Musilkova 3016
	Divisova 3021
	Divisova 3024
	Lanska 3142
	Lanska 3141
	Lanska 3140

	Lanska 3138
	V lipinach 3226
	V lipinach 3228
	V lipinach 3232
	V lipinach 3231
	V lipinach 3225
	Srbkova 3036
	Srbkova 3034
	Srbkova 3032
	Ticha 3012
	Ticha 2098
	Ticha 3108
	Mandysova 3123
	Mandysova 3134
	Mandysova 3132
	Mandysova 3129
	Mandysova 3128
	Mandysova 3126
	Mandysova 3125
	Mandysova 3124
	Mandysova 3122
	Mandysova 3121
	22 cervence 3103
	22 cervence 3102
	22 cervence 3105
	22 cervence 3106
	Zajickova 3145
	Zajickova 3152
	Zajickova 3153
	Zajickova 3151
	Zajickova 3147
	Zajickova 3146
	Kratka 3221
	Kratka 3224
	Raabova 3214
	Raabova 3210
	Raabova 3211
	Raabova 3216
	Raabova 3218
	Raabova 3219
	Raabova 3220
	Hranicni 3227
	Hranicni 3224
	Hranicni 3226
	Hranicni 3228
	Hranicni 3229

	Rumunska 33
Trasa pro vozidlo Iveco_1 (komunální odpad)	
Počet kilometrů:	13,05 km
Doba jízdy:	5:25
Iveco_1	Do noveho 22
	Husova 44
	Husova 55
	Husova 57
	Husova 62
	Husova 72
	Husova 1007
	Husova 1036
	Husova 123
	Husova 74
	Husova 1022
	Husova 1021
	Husova 75
	Husova 1024
	Husova 61
	Husova 1042
	Husova 1041
	Husova 1040
	Husova 39
	Na bukovine 67
	Na bukovine 1033
	Na bukovine 1002
	Na bukovine 1001
	Na bukovine 25
	Na bukovine 23
	Na bukovine 21
	Na okrouhliku 1118
	Na okrouhliku 1115
	Na okrouhliku 1114
	Na okrouhliku 11131
	Ve lhotkach 1085
	Ve lhotkach 1084
	Ve lhotkach 1081
	Ve lhotkach 1080
	Ve lhotkach 10
	Ve lhotkach 9
	Ve lhotkach 7
	Ve lhotkach 6
	Ve lhotkach 51
	Ve lhotkach 50
	Very Junkove 1078
	Very Junkove 1080

	Very Junkove 1079
	Sakacova 1048
	Sakacova 1047
	Sakacova 1044
	Sakacova 1045
	Sakacova 13
	Sakacova 16
	Sakacova 19
	Sakacova 20
	Sakacova 112
	Sakacova 111
	Sakacova 110
	Sakacova 109
	Gebauerova 123
	Gebauerova 88
	Gebauerova 80
	Winerova 1017
	Winerova 1019
	Winerova 116
	Winerova 117
	Winerova 119
	Winerova 122
	Winerova 143
	Winerova 144
	Winerova 150
	Winerova 152
	Strosova 141
	Strosova 142
	Schvartzovo namesti 153
	Kotkova 128
	Sporilov 1089
	Sporilov 1086
	Holubova 100
	Holubova 99
	Holubova 98
	Holubova 95
	Holubova 66
	Holubova 1003
	Holubova 1004
	Holubova 1005
	Judr krpaty 77
	Judr krpaty 81
	Judr krpaty 83
	Judr Krpaty 1016
	Judr Krpaty 1014
	Judr Krpaty 1012

	Judr Krpaty 1011 Judr Krpaty 1010 Judr Krpaty 1009 Judr Krpaty 1008 Judr Krpaty 1037 Judr Krpaty 35 Judr Krpaty 1038 Judr Krpaty 31 Judr Krpaty 5 Judr Krpaty 1 Bezdickova 175 Bezdickova 177 Bezdickova 1052 Bezdickova 1050 Bezdickova 1051 Bezdickova 1053 Bezdickova 1054 Bezdickova 1079 Sezemicka 1072 Sezemicka 1071 Sezemicka 1066 Sezemicka 1064 Sezemicka 1062 Sezemicka 1060 Sezemicka 107 Sezemicka 106 Sezemicka 103 Sezemicka 102 Sezemicka 1061 Sezemicka 1063 Sezemicka 1065 Sezemicka 1067 Sezemicka 1070 Sezemicka 1075
Trasa pro vozidlo Mercedes Faun_0 (bio)	
Počet kilometrů: Doba jízdy:	10,72 km 2:26
Mercedes Faun_0	Lesni 2019 Lesni 2009 Divisova 31026 V lipinach 3226 Mandysova 3123 Mandysova 3125 Mandysova 3130 Mandysova 3122 Mandysova 3121

	Very Junkove 1079 Very Junkove 1078 Sezemicka 1067 Sezemicka 103 Holubova 1006 Holubova 1005 Judr Krpaty 1038 Na bukovine 1001 Bezdickova 175 Sakacova 1044 Sakacova 114 Strosova 141 Winerova 122 Winerova 1017 Husova 1024 Husova 48 Husova 46

Příloha 12 – Statistiky

středa - 1. července. 2009	
Vozidlo	Celkem najeto
Vozidlo Iveco_2	11,03 km
Vozidlo Iveco_1	13,05 km
Vozidlo Mercedes Faun_0	10,72 km
Druh odpadu	celkem svezeno
smesny odpad	43890 l
bioodpad	3120 l