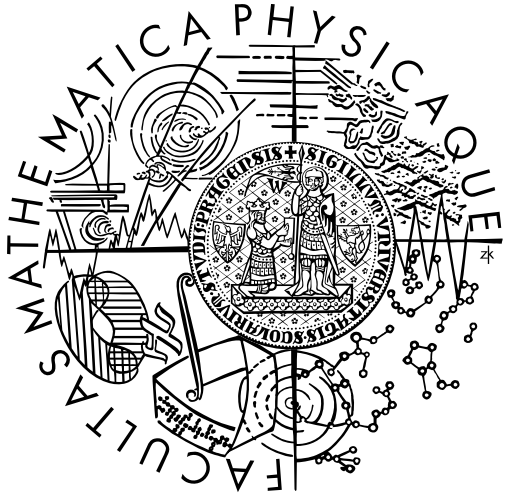
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁRSKA PRÁCA



Andrej Kalický

Generování taktových jízdních řádů na železnici

Katedra aplikované matematiky

Vedúci bakalárskej práce: Mgr. Robert Babilon

Študijný program: Správa počítačových systémov, Informatika

2009

Rád by som poďakoval Mrg. Robertovi Babilonovi za odborné vedenie a venovaný čas, taktiež za jeho cenné rady a pripomienky. Ďalej by som chcel poďakovať mojej rodine a priateľke, ktorí ma pri mojom štúdiu podporovali.

Prehlasujem, že som svoju bakalársku prácu napísal samostatne a výhradne s použitím citovaných prameňov. Súhlasím so zapožičiavaním práce.

V Prahe dňa 6.8.2009 Andrej Kalický

Obsah

Názov práce: Generování taktových jízdních řádů na železnici

Autor: Andrej Kalický

Katedra (ústav): Katedra aplikovanej matematiky

Vedúci bakalárskej práce: Mgr. Robert Babilon

e-mail vedúceho: babilon@kam.ms.mff.cuni.cz

Abstrakt: V predloženej práci sa autor zaoberá problematikou taktových cestovných poriadkov na železnici. Cieľom bakalárskej práce je navrhnúť časové posuny vlakových liniek voči sebe tak, aby prestupy medzi linkami vychádzali čo najlepšie. Autor formuluje riešený problém a jeho odlišnosť od štandardného procesu konštrukcie taktových cestovných poriadkov na železnici. Autor navrhuje a naprogramuje 2 algoritmy na optimalizáciu stávajúceho cestovného poriadku a v závere práce porovná výsledné štatistické údaje spojené s výslednými taktovými cestovnými poriadkami.

Kľúčové slová: taktový cestovný poriadok, železnice, generovanie, optimalizácia, obmedzujúce podmienky

Title: Construction of Periodic Railway Timetables

Author: Andrej Kalický

Department: Department of Applied Mathematics

Supervisor: Mgr. Robert Babilon

Supervisor's e-mail address: babilon@kam.ms.mff.cuni.cz

Abstract: In the propounded thesis author studies the periodic railway timetable. The aim of this thesis is to construct time shifts among train lines so that transfers between lines would be planned in the best possible way. Author discusses the problem to solve, and its differences from standard processing of periodic railway timetable construction. Author propounds and code 2 algorithms for optimization of given timetable. At the end of thesis the solutions and algorithms are compared based on statistical data related with the process of periodic railway timetable construction.

Keywords: periodic railway timetable, construction, optimization, constraints

1 Úvod

1.1 Motivácia

Taktové cestovné poriadky (ďalej pod skratkou CP) sa v dnešnej dobe stávajú trendom pre každú národnú železničnú spoločnosť v Európe, poskytujúcu osobnú dopravu. Zavedením taktových cestovných poriadkov získavajú spoločnosti lepšie možnosti pri regulácii a koordinácii železničnej siete, v neposlednom rade ovplyvňujú komfort cestovania. Taktové CP sa používajú aj pre iné dopravné siete ako železnice, napríklad pre systém integrovanej dopravy v mestách zahrňujúci metro, električky a autobusy, ktoré chodia v kratších časových intervaloch ako vlaky.

Taktový CP sa vyznačuje tým, že vlaky chodia v pravidelne opakujúcich sa intervaloch. Ak je v taktovom CP dobrý prípoj (s rozumným časom na prestup), tak je rovnako dobrý po celý deň. Vzhľadom k rozľahlosti železničnej siete a prepojenosti vlakových liniek nejde zaručiť dobrý prípoj všade. Z tejto myšlienky vyplýva úloha naplánovať čo najkvalitnejšie prípoje z pohľadu prestupujúcich pasažierov. Touto problematikou sa zaoberá táto bakalárska práca.

1.2 Cieľ práce

Cieľom práce je skonštruovať cestovný poriadok zo zadaných liniek tak, aby bolo čo najviac cestujúcich spokojných pri prestupoch. T.j. optimalizovať čas na prestupoch v závislosti od počtu cestujúcich, očakávaných na danom prestupe. Súčasťou zadania práce je navrhnúť algoritmy na generovanie a optimalizáciu cestovných poriadkov na železnici.

Výsledkom práce je program PTG, ktorého názov je skratka odvodená od Periodic Timetable Generation. Vstupom pre program je jeden konkrétny cestovný poriadok a ďalšie údaje, ktoré sa dajú zadávať na viacerých úrovniach. Výstupom programu sú viaceré cestovné poriadky, vygenerované rôznymi algoritmami a štatistické údaje pre porovnanie kvality vstupných dát, ktoré máme k dispozícii.

1.3 Štruktúra práce

V druhej kapitole je načrtnutá problematika generovania cestovných poriadkov a popísaná základná modelovacia technika, ktorá sa používa pri navrhovaní, plánovaní a optimalizácii CP. Tretia kapitola je zameraná na formuláciu riešeného problému v tejto bakalárskej práci.

Vo štvrtej kapitole sú popísané navrhnuté algoritmy. Piata kapitola obsahuje užívateľskú dokumentáciu, ktorá objasňuje postup, ako modifikovať vstupné údaje na viacerých úrovniach a ovplyvniť výsledky generovania.

V šiestej kapitole sú popísané detaily spojené s implementáciou. V siedmej kapitole sú vzájomne porovnané výsledky vygenerované navrhnutými algoritmami nad rôznymi kolekciami vstupných dát. Výsledky sú porovnané aj so skutočným cestovným poriadkom. V závere sú zhrnuté výsledky porovnania algoritmov a kriticky spomenuté nedostatky.

2 Problematika generovania cestovných poriadkov

V tejto kapitole sa rovnako ako aj v celej práci zameriavam na problematiku cestovných poriadkov (ďalej len pod skratkou „CP“) na železnici. Taktové CP sa používajú aj pre iné dopravné siete ako železnice, ako napríklad systém integrovanej dopravy v mestách zahrňujúci metro, električky a autobusy, ktoré chodia v kratších časových intervaloch ako vlaky. Preto sa v texte obmedzím na pojem taktové cestovné poriadky.

Špeciálne pre CP na železnici sa berie do úvahy aj infraštruktúra železničných tratí a rôzne iné obmedzenia z toho vyplývajúce, ktoré budú vysvetlené neskôr.

2.1 Taktové cestovné poriadky

Myšlienka taktových cestovných poriadkov spočíva v tom, že vlaky, prípadne vlakové spojenia, jazdia na danom úseku v pravidelných intervaloch, v tzv. taktoch. Vlakové súpravy jednej vlakovej linky sú z určitej stanice vypravované vždy v rovnaký čas v každej perióde intervalu, napríklad každú hodinu.

Taktové CP prinášajú viaceré výhody. Svojou transparentnosťou sú pre cestujúcich ľahko zapamätateľné, a to najmä pre ich pravidelnosť. Pre odchod vlaku si stačí zapamätať v ktorej minúte odchádza zo stanice. Koncept zachováva rovnaký čas potrebný na prestup počas celého dňa. Ak Vám teda uchádza prípoj len o pár minút, bude Vám pre periodicitu CP uchádzať rovnako po celý deň. Optimalizácia časov potrebných na prestupoch v závislosti od počtu cestujúcich je predmetom tejto bakalárskej práce.

Z hľadiska plánovania je výhodné, že pri generovaní stačí uvažovať iba jeden interval. Zásadnou požiadavkou je, že situácia na konci periódy musí odpovedať situácii na začiatku periódy. V prípade zhody sa základ vygenerovaného CP pre daný interval skopíruje do celého rozsahu dňa, a tým skomponuje výsledný celodenný CP. Vlaky chodia v každej perióde rovnako až na posun.

Pre prispôsobenie taktového CP potrebám obyvateľstva a dopytu po cestovaní sa narúša periodicita. V praxi to vyzerá tak, že v prípade dopravnej špičky sa hodinový interval môže skrátiť na polhodinu, naopak mimo špičky a pri nočných spojoch natiahnuť na dvojhodinový interval.

V skutočnom CP sa pridávajú spoje aj v dôsledku zvýšenia počtu cestujúcich pred víkendom a po víkende. Počas víkendov sa interval zväčšuje. Príkladom pridaných neperiodicky sa opakujúcich spojov môžu byť školské linky a linky zabezpečujúce dopravu pre industriálne zóny.

km	SŽDC, státní organizace / ČD, a.s. Vlak	5863 ⊖♿1 🚲1 80	5865 ⊖♿1 🚲1 80	5867 ⊖♿1 🚲1 80		5869 ⊖♿1 🚲1 80	5871 ⊖♿1 🚲1 80	5873 ⊖♿1 🚲1 16 80	5875 ⊖♿1 🚲1 10 80
	Praha Masarykovo nádraží 011	*12 15*	*13 15*	*14 15*		*15 15*	*16 15*	*17 15*	*18 15*
	Ze stanice								
0	**Poříčany** 011 🛏1 ⇔4	13 11	14 11	15 11		16 11	17 11	18 11	19 11
3	Třebestovice ⇔4	×13 14	×14 14	×15 14		×16 14	×17 14	×18 14	×19 14
5	Sadská ⇔5	13 17	14 17	15 17		16 17	17 17	18 17	19 17
9	Hořátev	×13 22	×14 22	×15 22		×16 22	×17 22	×18 22	×19 22
13	**Nymburk** město 061 o	13 26	14 26	15 26		16 26	17 26	18 26	19 26
	Nymburk město 061	13 26	14 26	15 26		16 26	17 26	18 26	19 26
15	**Nymburk** hl.n. 061,071,231 o	13 30	14 30	15 30		16 30	17 30	18 30	19 30
	Do stanice								

Obrázok 1: Ukážka taktového cestovného poriadku.

2.2 Predpoklady

V tejto sekcii uvedieme predpoklady a požiadavky na CP vyplývajúce z rôznych faktorov. Následne v sekcii 2.3 ukážeme, ako sa dajú takéto požiadavky formálne zapísať. Predpoklady pre model CP môžu byť rôzne v závislosti od infraštruktúry, vlakových spojení a iných požiadaviek na CP. Predpoklady a požiadavky sú vopred zadané.

Železničná infraštruktúra

Železničnú infraštruktúru tvoria uzly a trate.

- **Uzly** reprezentujú miesta v železničnej sieti, kde dochádza k vetveniu železničnej siete, respektíve tratí alebo koľají. Príkladom uzla je vlaková stanica, kríženia, výhybky a zoraďovacie koľajisko.

- **Trate** sú spojenia z uzla do najbližšieho uzla, po ktorých sa presúvajú vlaky. Medzi dvojicou uzlov môže existovať aj viac paralelne položených tratí - koľají, a každému vlaku je vopred určená voľná koľaj po ktorej sa má presúvať. Trate môžu byť:

    - Jednokoľajové.

    - Dvojkoľajové, pri ktorých je každá koľaj využívaná jedným smerom.

    - Viackoľajové, (napr.: 2 koľaje pre každý smer) na ktorých môže byť riešené predbiehanie v tom zmysle, že pre koľaje v jednom smere sú pridelené vlaky podľa rýchlosti (jedna koľaj pre IC a rýchliky, druhá pre osobné a nákladné vlaky).

    - Banalizované viackoľajové, kde všetky koľaje môžu byť využívané pre oba smery podľa potreby.

- **Stanica** je uzol určený pre zastavenie vlaku na určitý čas, pre nástup a výstup cestujúcich, ale taktiež miesto pre križovanie vlakov. Pri pohľade na stanicu ako uzol, je skrytá jej vlastná infraštruktúra. Uvažovanie a začlenenie staničnej infraštruktúry by viedlo k veľkému a komplikovanému modelu, preto v našom modeli stanice chápeme ako čierne skrinky. Pri tomto prístupe môže nastať prípad, že po vygenerovaní CP nebude realizovateľný pre obmedzenie staničnej infraštruktúry. Existujú modely a algoritmy, ktoré zahŕňajú aj staničnú infraštruktúru (koľaje a nástupištia).

Vlaky

Jednotlivé vlaky sú uvažované vo forme vlakových liniek.

- **Vlaková linka** je priame vlakové spojenie medzi počiatočnou a konečnou stanicou po určitej, vopred definovanej trase. Každá linka jazdí v pravidelnom intervale, t.j. pravidelne sú vypravované vlaky v každej perióde intervalu. Pre každú linku je čas medzi dvomi stanicami vopred určený a fixný.

Cestujúci

Vzhľadom k počtu cestujúcich, ich zámeru odkiaľ, kam, a v akom čase cestovať, je optimalizovaný celý železničný model. Podľa toho sú navrhované vlakové linky, ich periodicita, či použitie konkrétnych vlakových súprav.

Špeciálne požiadavky

CP musia spĺňať viacero požiadaviek, ktoré sú kladené na bezpečnostnú reguláciu, na dosiahnutie určitého štandardu poskytovaných služieb a v neposlednom rade musia byť realizovateľné samotné CP.

- **Pobyt vlaku na stanici** - je časový interval počas ktorého vlak zastaví na stanici. Spodná hranica predstavuje minimálny potrebný čas na nástup a výstup pasažierov. Na druhej strane horná hranica obmedzuje dobu státia, po uplynutí ktorej už vlak nevyťažuje kapacitu stanice. Započítava sa do celkovej doby cestovania.

- **Prípojové vlaky** - dva vlaky môžeme označiť ako prípojové, ak je medzi nimi plánovaný prestup. Dôležitým faktorom je príchod prvého nasledovaný odchodom druhého vlaku zo stanice, respektíve sa uvažuje rozdiel týchto dvoch. Rozdiel by mal zohľadňovať minimálny čas na prestup a nemal by byť príliš veľký.

- **Spájanie vlakov** - spájanie môže vzniknúť medzi dvomi vlakmi, ktoré majú v určitom úseku spoločnú trasu. Takáto kombinácia dvoch vlakov si vyžaduje prítomnosť oboch v stanici, kde sú spájané do jednej vlakovej súpravy. Optimalizuje sa tým veľkosť posádky (na spoločnej trase postačí jedna) a šetrí sa kapacita tratí (dva vlaky by museli mať medzi sebou bezpečnostné časové rozstupy).

- **Synchronizácia vlakov** sa používa, ak dve vlakové linky majú spoločný úsek na ich trasách. Synchronizácia časov odchodu prebieha už od prvého spoločného uzla. Napríklad pri rovnakej perióde majú dve vlakové linky frekvenciu jedna, po synchronizácii je poskytovanie prepravy osôb na spoločnom úseku trasy s frekvenciou dva.

- **Otáčanie vlakovej súpravy** v konečnej stanici býva v dôsledku jej využitia pre vlakovú linku v opačnom smere. Čas strávený v tomto uzle je naplánovaný tak, že započítava dobu prepojovania súpravy, prípadné overenie technického stavu a taktiež aj čas slúžiaci na zmenšovanie alebo absorbovanie meškaní.

- **Fixované príchody a odchody** sa vyskytujú napríklad pri vlakových linkách, ktoré sú zároveň medzinárodnými linkami. Čas príchodu na hranice štátu je vymedzený vzájomnou dohodou susediacich železničných spoločností a nie ako výsledok plánovacieho procesu.

- **Bezpečnostná regulácia** spočíva v tom, že dva vlaky využívajúce rovnakú trať v určitom smere sú od seba oddelené bezpečnostnými časovými rozstupmi. Časový rozdiel medzi nimi musí byť dodržaný rovnako v

počiatočnom aj v koncovom uzle trate. Bezpečnostné opatrenia taktiež nepovoľujú stretávanie a predbiehanie vlakov na jednej koľaji.

2.3 Obmedzujúce podmienky

Obmedzujúce podmienky sa používajú pre modelovanie vzťahu medzi dvomi udalosťami. Pomocou nich sa dajú sformulovať všetky možné vzťahy, požiadavky a predpoklady, ktoré majú byť obsiahnuté vo výsledne vygenerovanom CP.

2.3.1 Formulácia obmedzujúcich podmienok

V tejto sekcii si ukážeme ako sa dajú predchádzajúce požiadavky zo sekcie 2.2 pretransformovať do obmedzujúcich periodických podmienok. Väčšina známych algoritmov na generovanie CP je založená na splňovaní takéhoto modelu s obmedzujúcimi podmienkami. Podkladom pre nasledujúci súhrn formulácií obmedzujúcich podmienok nám bola použitá literatúra [1].

Periodická obmedzujúca podmienka

Cestovný poriadok pozostáva z časov príchodu a odchodu pre všetky linky v každej stanici, ktorou prechádzajú. Pri modelovaní sa používajú rozhodovacie premenné pre príchody a odchody nasledovne:

$$a_n^t \in \{0,..,T-1\} \text{ čas príchodu vlaku } t \text{ do uzla } n$$
$$d_n^t \in \{0,..,T-1\} \text{ čas odchodu vlaku } t \text{ z uzla } n$$

Kde parameter T je interval cestovného poriadku v minútach. Rozhodujúce premenné a_n^t a d_n^t nadobúdajú celočíselné hodnoty z domény $\{0,..,T-1\}$. Situáciu, keď vlak t príde do stanice s, pobudne v stanici jednu minútu a odíde, zapíšeme pomocou obmedzujúcej podmienky takto:

$$d_s^t - a_s^t = 1$$

Inými slovami udalosť a_s^t nastane jednu minútu pred udalosťou d_s^t, a naopak udalosť d_s^t nastane jednu minútu po a_s^t.

Periodicita obmedzujúcich podmienok sa dosiahne počítaním v modulo T (pri perióde T). Pre zovšeobecnenie myšlienky uvažujme a_n^t ako príchod vlaku t do uzla n, $d_m^{t'}$ ako odchod iného vlaku t' z uzla m, a ich vzťah vytvoríme pomocou časového okna $[l,u]$ namiesto fixovanej hodnoty. Časové okno predstavuje interval s dolnou a hornou hranicou pre rozdiel dvoch rozhodujúcich premenných.

$$a_n^t - d_m^{t'} + Tp \in [l,u] \qquad p \in \mathbb{Z}$$

Ekvivalentne sa predchádzajúci zápis pre prehľadnosť skracuje na:

$$a_n^t - d_m^{t'} \in [l,u]_T$$

Pre jednotlivé špeciálne požiadavky spomenuté v 2.2 sa formulujú obmedzujúce podmienky nasledovne (všetky konkrétne čísla použité v obmedzujúcich podmienkach sú uvedené ako príklad).

- **Pobyt vlaku t na stanici s**, ktorý má byť minimálne 2 minúty a maximálne 10 minút sa dá vyjadriť

$$d_s^t - a_s^t \in [2,10]_{60}$$

- **Prípojové vlaky** - plánovaný vzájomný prestup medzi linkami t_1 a t_2 v stanici s, sa definuje dvojicou obmedzujúcich podmienok. Požadovaný čas na prestup od 2 do 10 minúť medzi oboma navzájom.

$$d_s^{t_2} - a_s^{t_1} \in [2,10]_{60}$$
$$d_s^{t_1} - a_s^{t_2} \in [2,10]_{60}$$

- **Spájanie dvoch vlakov** t_1 a t_2, respektíve spájanie ich vlakových súprav do jednej (napr. t_1) na spoločnej trase ohraničenej stanicami s_1 a s_2 si vyžaduje, aby boli v oboch hraničných staniciach obidva vlaky prítomné. Presnejšie v stanici s_1, kde sa vlaky spájajú do t_1, príchod vlaku t_2 musí predchádzať odchodu vlaku t_1. Druhá obmedzujúca podmienka vyplýva analogicky pre stanicu, kde sa rozpájajú:

$$d_{s_1}^{t_1} - a_{s_1}^{t_2} \in [5,10]_{60}$$
$$d_{s_2}^{t_2} - a_{s_2}^{t_1} \in [5,10]_{60}$$

- **Otáčanie vlakov** v konečnej stanici pre použitie vlakovej súpravy na linke v opačnom smere vyjadruje obmedzujúca podmienka

$$d_s^{t_2} - a_s^{t_1} \in [20,50]_{60}$$

kde je požadovaný minimálny čas 20 minúť predtým ako vlaková súprava opustí konečnú stanicu a 50 minút je maximálny čas, ktorý v nej môže pobudnúť.

- **Fixované príchody a odchody** sa vyskytujú pri medzinárodných linkách. Vlak vstupuje na územie štátu o :25 a opúšťa územie o :34. Vstup odpovedá odchodu medzinárodného vlaku od uzla na hranici, ktorý je podľa dohody napríklad v rozpätí :23 a :27. Opúšťanie odpovedá príchodu vlaku do uzla hranice v rozpätí :32 a :36.

$$d_b^t = 25$$
$$a_b^t = 34$$
$$d_b^t \in [23,27]$$
$$a_b^t \in [32,36]$$

Obmedzujúce podmienky nie sú ale periodické. Vyjadrujú presne hodnotu v minútach. Pre zapísanie do obecného tvaru potrebujeme pridať pomocnú premennú

$$\beta \in \{0, \dots, T-1\}$$

Dostávame upravené periodické obmedzujúce podmienky:

$$d_b^t - \beta = [25]_{60}$$
$$a_b^t - \beta = [34]_{60}$$
$$d_b^t - \beta \in [23,27]_{60}$$
$$a_b^t - \beta \in [32,36]_{60}$$

Každá obmedzujúca podmienka má dve rozhodujúce premenné. Ak nejaký cestovný poriadok spĺňa všetky obmedzujúce podmienky, tak pridaním m minút pre každý čas príchodu a odchodu dostaneme nový cestovný poriadok, rovnako spĺňajúci všetky podmienky. Oba CP sú v podstate rovnaké, až o posunutie o m minút. Ak pri zvolenom posunutí sú všetky pomocné premenné $\beta = 0$, tak všetky obmedzujúce podmienky pre fixné odchody a príchody sú splniteľné.

- **Synchronizáciu dvoch vlakov** t_1 a t_2, ktoré majú značnú časť trasy spoločnú a frekvenciu jedna v perióde cestovného poriadku, skonštruujeme nasledovne: chceme posunúť odchod vlaku t_2 voči odchodu vlaku t_1 o 30 minút s chybou 2 minúty. Synchronizácia sa vzťahuje na celú spoločnú trasu so stanicami $s_i, i \in \{1, \dots, n\}$.

$$d_{s_i}^{t_2} - d_{s_i}^{t_1} \in [28,32]_{60} \qquad i \in \{1, \dots, n\}$$

Po synchronizácii vlakov o 30 minút je poskytovaná osobná doprava na spoločnej trase s frekvenciou dva. Podobným spôsobom sa dajú synchronizovať viac ako dva vlaky.

- **Bezpečnostná regulácia** rozstupov (napr. 3 minúty) dvoch po sebe idúcich vlakov v jenom smere po tej istej trati znamená, že ak vlak t_1 opustí stanicu s v určitom čase, tak vlak t_2 nesmie opustiť stanicu do 3 minút pred ani po odchode vlaku t_1. Keďže bezpečnostný rozstup sa týka vlakov po celej dĺžke trate medzi stanicami s_1 a s_n, tak dostávame dve podmienky:

$$d_{s_1}^{t_2} - d_{s_1}^{t_1} \in [3,57]_{60}$$
$$a_{s_n}^{t_2} - a_{s_n}^{t_1} \in [3,57]_{60}$$

2.4 Periodic Event Scheduling Problem

Táto sekcia opisuje Periodic Event Scheduling Problem (PESP) zaoberá naplánovaním periodicky sa opakujúcich udalostí v určitom časovom intervale. PESP bol pôvodne sformulovali Serafini a Ukovich v roku 1989 [2]. Súčasťou ich formulácie bola definícia PESP stavajúca na obmedzujúcich podmienkach a navrhnutá metóda pre nájdenie rozumného riešenia založená na technike *branch and bound*. Problém splniteľnosti zadaných obmedzujúcich podmienok je NP-úplný [2].

Definícia PESP

Nech je daná N množina udalostí, množina $A \subseteq N \times N$, časová perióda T, a časové okná $[l_{ij}, u_{ij}]$ pre všetky $(i,j) \in A$. PESP má nájsť periodický rozvrh $v_i \in [0,T)$, $i \in N$, ktorý spĺňa

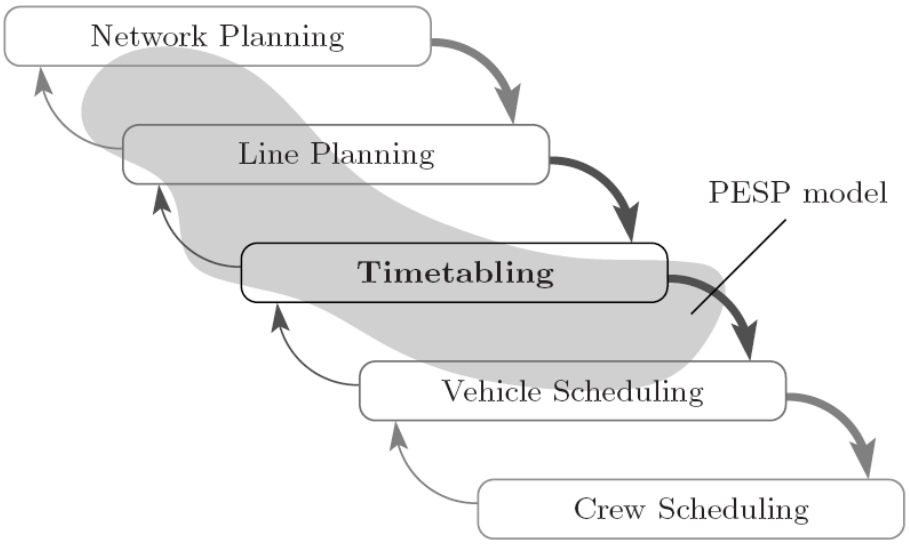
$$(v_j - v_i)\ mod\ T \in [l_{ij}\, u_{ij}] \quad pre\ všetky\ (i,j) \in A$$

alebo rozhodne, že vstupné podmienky sú nesplniteľné a rozvrh neexistuje.

PESP poskytuje akýsi framework pre podmienky typu popísané v sekcii 2.3. Hľadanie riešenia spočíva v iteratívnom fixovaní rozhodujúcich premenných a testovaní takto upravených podmienok na splniteľnosť. V prípade detekcie nesplniteľnosti podmienok je použitý backtracking.

2.5 Fázy plánovania a PESP

Komplexný model plánovania zahŕňa fázy: plánovanie infraštruktúry, plánovanie liniek, generovanie CP, plánovanie vlakových súprav a plánovanie personálu. Proces plánovania je ovplyvnený vzájomnou interakciou dvoch na seba nadväzujúcich fáz.



Obrázok 2: Fázy plánovania. Zdroj: [3]

Model PESP pôvodne predstavoval fázu generovania cestovného poriadku a čiastočne zahŕňa plánovanie vlakových súprav nasadzované na vlakové linky.

PESP však nie je limitované len pre generovanie CP, dá sa rozšíriť aj na ostatné fázy plánovacieho procesu. Problematiku rozšírenia rozoberajú Liebchen a Möhring v práci [3]. Demonštrujú modelovaciu silu PESP a jeho schopnosť rozšírenia aj na iné fázy plánovania, konkrétne plánovanie liniek a čiastočne plánovanie infraštruktúry. Spätná väzba na plánovanie infraštruktúry odzrkadľuje požiadavky na aktuálny stav plánovania liniek.

2.6 Známe algoritmy

Ďalší výskum problematiky generovania a optimalizácie taktových cestovných poriadkov vychádzal z modelu PESP, ktorý uviedli Serafini a Ukovich. Postupne bol doplňovaný o nové obmedzujúce podmienky zohľadňujúce dodatočné požiadavky na CP. Boli navrhované nové metódy vyhľadávania riešení obohatené o rôzne heuristiky. Prehľad známych algoritmov a prístupov k riešeniu danej problematiky sme čerpali z prác [1] a [4].

Voorhoeve [5], ako prvý, uvažoval PESP model v súvislosti s taktovými cestovnými poriadkami. Vyvinul algoritmus založený na propagácii obmedzujúcich podmienok a *backtrackingu* na nájdenie rozumného taktového CP.

Schrijver and Steenbeek [6-7], vyvinuli algoritmus založený na *constraint programmingu* a nadviazali tak na prácu od Voorhoeve. Algoritmus v prípravnej fáze eliminuje nadbytočné rozhodujúce premenné a obmedzujúce podmienky. Vyvinuli metódu na post optimalizáciu získaného CP.

Odijk [8] uviedol pre riešenie problému splniteľnosti algoritmus, ktorý bol založený na technike *cutting plane*. Použil model PESP pre náhodné generovanie množiny železničných cestovných poriadkov, ktoré potom použil na testovanie voči navrhovaným zmenám v infraštruktúre železničnej siete, konkrétne ku zmenám kapacity železničných staníc (koľaje a nástupištia).

Hassin [9] sformuloval *Network Synchronization Problem* (NSP), ktorý predstavuje optimalizovanú formuláciu PESP. Časové okná nepredstavujú striktné obmedzenia, ale namiesto toho sú hodnoty mimo požadovaných časových okien vysoko penalizované. Ďalej uviedol, že penalizovanie časových okien s veľkým intervalom hodnôt zapríčiňuje problémy.

Nachtigall [10-11] použil obmedzujúce podmienky PESP. Pri generovaní CP zobral do úvahy minimalizáciu doby čakania cestujúcich. Okrem toho uviedol optimalizáciu pre dvoj-kritéria. Napríklad dodatočné náklady na infraštruktúru verzus prínos vylepšenia synchronizovanosti v CP.

Nachtigall a Voget [12] zobrali do úvahy problém minimalizácie doby čakania cestujúcich. Heuristicky vygenerovali počiatočný CP, využijúc myšlienky z praktizovaného manuálneho konštruovania CP. Na vylepšenie počiatočného CP použili *genetické algoritmy*.

Lidner [13] zobral PESP ako súčasť problému, ktorý konštruuje CP optimalizovaný voči nákladom na vozový park.

2.7 Existujúce systémy

DONS

Desgin of network Schedules (DONS), vyvinutý Holanďanmi, je systém asistujúci pri smerovaní a plánovaní železničnej dopravy. Systém sa používa pri generovaní kostry vlakových spojení v Holandsku (IC vlakov). Pozostáva z troch častí: užívateľského rozhrania, databázy a inteligentných modulov.

Modul *CADANS*, ktorý pri generovaní neberie podrobnú infraštruktúru železničnej siete, t.j. neuvažuje udalosti odohrávajúce sa v rámci infraštruktúry staníc. Druhý modul *STATION* skúma pre jednotlivé stanice a ich infraštruktúru, či priraďovanie nástupíšť a smerovanie vlakov je v súlade s CP vygenerovaným modulo *CADANS*.

2.8 Prípadová štúdia

Nasledujúca sekcia stručne popisuje situáciu ako bol použitý matematický optimalizačný model v praxi, t.j. pri generovaní CP pre sieť liniek metra [14-15].

2.8.1 Berlínske metro

Sieť berlínskeho metra

Sieť berlínskeho metra má dĺžku všetkých tratí 144 km s počtom 170 staníc, kde 19 z nich je prestupných. Priemerná doba cestovania je 6 km alebo 8 staníc. Počas jedného dňa metro prepraví okolo 1,3 milióna cestujúcich. Z úvodných informácií sa dá usúdiť, že ide o rozľahnú sieť, ktorá je len časťou siete integrovanej dopravy v Berlíne. Pre porovnanie pražské metro má 59 km s počtom 57 staníc, z toho sú 3 prestupné.

Počas dopravnej špičky a v pracovných dňoch sú intervaly na linkách kratší 5 minút. Intervaly vo večerných a nočných hodinách a počas víkendov sú 10 minút. Predmetom ich štúdie bola prevádzka berlínskeho metra mimo dopravnej špičky pri intervale T=10.

Požiadavky

Cieľom ich štúdie bolo vylepšiť aktuálny CP metra a predovšetkým redukovať dobu čakania cestujúcich. Kritéria, ktoré sa brali do úvahy, sa týkali počtu použitých vlakových súprav, priemernej doby potrebnej na prestup, priemernej rýchlosti vlakov, a počtu plánovaných prestupov.

Ďalšími kritériami boli vyrovnanosť medzi všetkými prestupmi (neexistujú príliš zlé časy na prestupoch), a stabilita CP, schopnosť absorbovať v určitej miere odchýlenie sa od plánovaného rozvrhu.

Optimalizácia

Pri modelovaní použili PESP model s určitými doplneniami. Výslednú úlohu lineárneho programovania riešili CPLEX[1] MIP-Solverom. Skonštruovali CP, ktorý podstatne vylepšil ten predchádzajúci vo všetkých zadaných kritériách.

[1] http://www.cplex.com

Výsledok

Prvý výsledný CP zredukoval vážený priemer čakacej doby cestujúcich z 34% doby intervalu na menej ako 17%. Pre 24 najdôležitejších prestupov sa priemerná doba čakania znížila z 30% na 4% doby intervalu. Zvýšil počet potrebných vlakových súprav zo 71 na 78. Spoločnosť BVG, prevádzkujúca berlínske metro, ho zamietla.

Nasledujúca spolupráca medzi vývojovým tímom generovania CP a BVG[2] viedla k vzniku viacerých CP. Pri každom z nich sa vyskytli situácie, ktoré viedli k zmene pôvodných podmienok alebo sa pridávali dodatočné nové požiadavky. Konečný akceptovaný CP bol v porovnaní s pôvodným CP z pohľadu vylepšenia doby čakania len o niečo lepší, ale výrazne však zlepšoval vyrovnanosť a stabilitu.

Prípad optimalizácie ukázal, že výsledný CP je významne lepší ako manuálne skonštruovaný v splniteľnosti kritérií. Prípad demonštroval úspešné uvedenie prvého, matematicky optimalizovaného, taktového CP v praxi.

[2] Spoločnosť prevádzkujúca metro v Berlíne, http://www.bvg.de

3 Formulácia riešeného problému

Táto kapitola obsahuje formuláciu riešeného problému, ktorý vychádza zo zadania bakalárskej práce. Definuje vstup a požadovaný výstup, popisuje minimalizačnú funkciu v závislosti od ktorej prebieha optimalizácia problému.

V porovnaní s druhou kapitolou, ktorá obsahuje komplexný pohľad na danú problematiku, táto kapitola opisuje ako sa nami riešený problém odlišuje od vyššie spomenutého problému. Odlišnosť vyplýva už zo zadania.

3.1 Vstup

Vstupnými dátami sú rozplánované vlakové linky, ktoré majú definovanú pevnú trasu, číslo linky, časový interval a obsahujú zoznam zastávok. Zastávka obsahuje názov stanice, čas príchodu a odchodu do stanice, a na akom kilometri od počiatku trasy sa nachádza. Časovanie vlakových liniek je tak kompletne zadané.

Parametrom vstupu je aj zoznam navzájom previazaných liniek, ktorým môžeme definovať vzťahy pre jednokoľajnú trať alebo prekladané linky. Prekladané linky zvyšujú frekvenciu poskytovaných prepravných služieb v danom úseku. Vzájomne previazané linky tvoria triedu ekvivalencie. Ak posuniem odchod linky o m minút, posunú sa rovnako o m minút odchody všetkých liniek, ktoré sú s ňou previazané.

Posledným dôležitým parametrom vstupu je počet pasažierov pre konkrétny prestup. Zadávanie počtu pasažierov cez užívateľské rozhranie jednotlivo je zdĺhavá záležitosť. Preto zvolíme alternatívny spôsob, ako si ich vygenerovať v závislosti od osídlenia.

Pomocou Newtonovho gravitačného zákona si vyjadríme počet očakávaných cestujúcich medzi dvomi stanicami. Do vzorca dosadíme počty obyvateľov aglomerácie spadajúcich pod danú stanicu a vzdialenosť medzi týmito stanicami.
Pre vypočítanie vzdialenosti medzi dvomi mestami použijeme zadané údaje vlakových liniek a algoritmus pre vyhľadávanie najkratšej (najrýchlejšej) cesty. Tento algoritmus nám vygeneruje vlakové spojenia, z ktorých získame všetky možné prestupy. Pre každý špecifický prestup (linka, linka, stanica) sa sčíta hodnota cestujúcich z príslušných prestupov obsiahnutých vo vlakových spojeniach. A tým dostávame potrebný vstupný údaj o počte očakávaných cestujúcich na prestupoch.

3.1.1 Dodatočný vstup

Pre spomínané generovanie počtu cestujúcich na prestupoch budeme potrebovať zadať ďalšie údaje. Dodatočným vstupom je zoznam staníc spolu s počtom obyvateľov a minimálnym časom na prestup, ktorý je špecifický pre každú stanicu. Pre jednoduchosť si stanice rozdelíme do kategórií, kde pre každú z nich bude pripadať určitý počet obyvateľov.

Každú stanicu špecifikuje aj názov mesta v ktorom sa nachádza a počet obyvateľov aglomerácie spadajúcich pod danú stanicu. Položka mesto nie je pre stanicu povinná, slúži pre potreby optimalizácie. Nie sú vyhľadávané vlakové spojenia medzi dvomi stanicami jedného mesta, inak by nezohľadňovali realitu. Obyvatelia mesta preferujú MHD, ktorá poskytuje frekventovanejšiu osobnú dopravu. Okrem toho by došlo

k skresleniu výsledkov, pri staniciach v krátkej vzdialenosti s veľkou hustotou osídlenia.

3.1.2 Viacúrovňové zadávanie údajov

Procesu optimalizácie predchádza možnosť viacúrovňového zadávania, respektíve modifikácie vstupných údajov. Možnosť modifikácie je na úrovni vlakových liniek, ich zastávok a staníc. Po vygenerovaní vlakových spojení medzi všetkými stanicami, sa dajú modifikovať údaje o vlakových spojeniach medzi jednotlivými stanicami. Takéto viac úrovňové zadávanie údajov ovplyvňuje výsledné vygenerované taktové cestovné poriadky.

3.2 Optimalizácia

Cieľom optimalizácie CP je nájsť vzájomné posunutie zadaných vlakových liniek v čase tak, aby bolo čo najviac cestujúcich spokojných. T.j. minimalizovať časy v závislosti od počtu cestujúcich na prestupoch. Kvalita CP je urečnená minimalizačnou funkciou, ktorá je definovaná v nasledujúcej sekcii.

Sformulovaný problém má exponenciálnu zložitosť (prehľadaním všetkých možností), nemusí nájsť riešenie v rozumnom čase.

3.2.1 Minimalizačná funkcia

Navrhnuté algoritmy počítajú s minimalizačnou funkciou, ktorá je navrhnutá ako súčet cez všetky prestupy z množiny P súčinu váhy a času maximálnej doby prestupu. Formálne zapísané ako:

$$\sum_{(t_i,t_j,s)\in P} w_s^{ij}\left(d_s^{t_j} - a_s^{t_i}\right)$$

$(t_i, t_j, s) \in P$	je prestup medzi dvomi linkami v stanici s, z linky t_i na linku t_j
$P = \{(t_i, t_j, s)\}$	je množina existujúcich prestupov v CP
$a_s^{t_i}$	je čas príchodu linky t_i do prestupnej stanice s
$d_s^{t_j}$	je čas odchodu linky t_j z prestupnej stanice s
w_s^{ij}	je počet očakávaných cestujúcich na prestupe (t_i, t_j, s)
N	je počet vlakových liniek
$i, j \in \{1, \dots, N\}$	
S	je množina všetkých staníc
$s \in S$	je prestupná stanica pre linky t_i a t_j

3.3 Požadovaný výstup

Výstupom sú skonštruované taktové cestovné poriadky. Sú vygenerované rôznymi navrhnutými algoritmami optimalizované vzhľadom k minimalizačnej funkcii. Pre štatistické účely porovnania sú súčasťou výstupu pre jednotlivé CP aj výsledná hodnota minimalizačnej funkcie a počet progresívnych zmien, ktoré vylepšili aktuálnu hodnotu minimalizačnej funkcie.

3.4 Doplňujúce poznámky

3.4.1 Obmedzujúce podmienky

Keďže kompletné časovanie vlakových liniek je zadané priamo na vstupe, riešený problém sa zužuje len na použitie obmedzujúcich podmienok pre prípojové vlaky. Ostatné podmienky zo sekcie 2.3 pri našom probléme nebudeme uvažovať. Okrem toho zavedieme obmedzujúcu podmienku pre previazané linky.

Previazané linky

Konfliktu, pri ktorom dva vlaky používajú jednu koľaj v rovnakom čase, sa dá predísť zapamätaním si relatívneho posunu v čase medzi takýmito dvomi linkami. Predpokladá sa, že vo vstupnom CP nie je konflikt, a križovanie je načasované v uzle, kde je to umožnené. Týmto nám vzniká formulácia podmienky pre previazané linky.

Pre zachovanie vzťahu križovania dvoch protichodných liniek na jednokoľajovej trati používame podmienky typu pre previazané linky:

$$d^{t_2} - d^{t_1} \in [m]_{60} \qquad m \in \{0, \dots, T-1\}$$

Odchody dvoch previazaných liniek sú tak od načítania vstupných údajov pevne fixované.

Previazané linky sú také linky, že keď zmením čas odchodu jedného vlaku z jeho počiatočnej stanice o m minút neskôr, rovnako o m minút sa mi zmení aj čas odchodu druhého vlaku z jeho počiatočnej stanice. Navzájom previazané linky tvoria triedy ekvivalencie.

Previazanými linkami by sa dali riešiť aj niektoré špeciálne požiadavky zo sekcie 2.2, konkrétne spájanie a synchronizácia vlakov. Špeciálna požiadavka na pobyt vlaku na stanici by sa dala riešiť rozdelením linky v tejto stanici na dve.

3.4.2 Ostatní cestujúci

Cestujúcich bez prestupov nebudeme uvažovať. Ich doba čakania závisí iba na veľkosti intervalu linky, ktorou cestujú.

3.4.3 Zdroj vstupných údajov

Konkrétny existujúci CP (rozplánované vlakové linky) získame z IDOSu[3].

[3] http//www.idos.cz

4 Navrhnuté algoritmy

4.1 Náhodné generovanie s vylepšovaním

Prvý z navrhnutých a implementovaných algoritmov je založený na vygenerovaní náhodného CP. Jeho modifikáciou sa ho postupe snažíme vylepšovať. Na vylepšovanie použijeme metódu lokálneho prehľadávania.

Pseudokód algoritmu

Randomized Generation Algorithm with local search

```
- definitions
AllLines : is a set of all used lines
AvailableLines : is a set of lines
StableLines : is a set of lines
selectedLine : is a line
ConnectedLines : is set of lines connected with actual selectedLine
state : actual state of timetable

- initialization
createRandomizedTimetable(AllLines)
AvailableLines := AllLines
StableLines := 0

- main loop
while AvailableLines ≠ 0 do
    improved := false
    selectedLine := chooseRandomlyFrom(AvailableLines)

    for all possible shifts {0,..T − 1} in period of selected line do
        change start time for selectedLine and its ConnectedLines
        calculate all transfers` rating value of current shift

        if newRatingValue > oldRatingValue then
            revert changes
        else if newRatingValue < oldRatingValue then
            remember state
            improved
        fi
    od

    if improved then
        increment progressive changes field of selectedLine
        StableLines := 0
        AvailableLines := AllLines
    fi
    StableLines := StableLines ∪ selectedLine ∪ ConnectedLines
    AvailableLines := AvailableLines \
                        (selectedLine ∪ ConnectedLines)
od
```

Popis algoritmu

Algoritmus je navrhnutý tak, že si v inicializačnej časti vygeneruje náhodný CP, a potom sa ho snaží vylepšovať. Súčasťou inicializácie je aj nastavenie množiny stabilných liniek na prázdnu a do množiny dostupných liniek, s ktorými ešte môžeme posúvať, sa pridajú všetky zadané linky. Množina stabilných liniek tvorí doplnok množiny dostupných liniek a naopak (každá linka musí byť práve v jednej z týchto množín).

Vylepšovanie prebieha tak, že algoritmus si náhodne vyberie linku z množiny dostupných liniek, ktorá ešte môže byť vylepšená. Vybranou linkou algoritmus posúva v tom zmysle, že nastavuje čas odchodu z počiatočnej stanice prechádzaním celej množiny periódy $\{0, \dots, T-1\}$. Množina reprezentuje minúty, o ktoré môže byť posunutý aktuálny CP linky. Algoritmus pri vybranej linke zistí, či je previazaná s inými linkami. Ak áno tak algoritmus pri každom posune v čase posúva celým komponentom rovnako. Komponent tvorí vybraná linka a s ňou previazané linky.

Pre každé posunutie si spočíta minimalizačnú funkciu pre všetky prestupy obsiahnuté v CP. Prejdením celej množiny overí, či je možné posunutím vybranej linky, prípadne celého jej komponentu vylepšiť aktuálny CP. Ak sa nedá žiadnym posunutím CP vylepšiť, algoritmus ďalej pokračuje s predchádzajúcim najlepším CP (t.j. v tejto iterácii nenastali žiadne zmeny na CP).

Ak sa dá CP vylepšiť, algoritmus si zapamätá aktuálny CP ako doteraz najlepší z pohľadu optimalizácie. Súčasne sa množina stabilných liniek nastaví na prázdnu, a množina dostupných liniek sa rozšíri opäť na všetky linky, podobne ako v inicializačnej časti algoritmu.

V oboch prípadoch, či už je CP vylepšený alebo nie z pohľadu minimalizačnej funkcie, sa vybraná linka, prípadne celý komponent previazaných liniek pridá do množiny stabilných liniek a súčasne odoberie z množiny dostupných liniek.

Konečnosť algoritmu

Algoritmus určite skončí po konečnom množstve krokov. V každom kroku sa z pohľadu hodnoty minimalizačnej funkcie aktuálne riešenie zlepší alebo ostane rovnaké. Začíname s hodnotou minimalizačnej funkcie pôvodne vygenerovaného CP, ktorý je inicializovaný náhodnými hodnotami. Hodnota môže naďalej už len klesať.

$$\sum_{P} w_s^{ij} \left(d_s^{t_j} - a_s^{t_i} \right) \leq \sum_{P} w_s^{ij} \, T_l \leq \; 120 \sum_{P} w_s^{ij}$$

V každom kroku algoritmu (jedna iterácia cyklu `while`), ak sa hodnota minimalizačnej funkcie zlepší, tak poklesne minimálne o 1. Maximálny počet operácií (iterácií cyklu `for`) medzi dvomi po sebe nasledujúcimi zlepšeniami je $O(L \times T_l)$. Kde L je počet liniek a T_l je perióda linky.

Množina hodnôt minimalizačnej funkcie je konečná a zdola obmedzená nulou. Ak pri jednej iterácii vonkajšieho cyklu uvažujeme náhodný výber linky, ktorá obsahuje

previazané linky, posúvame s nimi naraz, celým komponentom. To nám počet operácií môže len znížiť, celý komponent sa pridá do množiny stabilných liniek.

Zložitosť algoritmu

Celkový počet operácií v podstate závisí na nájdenom počiatočnom riešení, respektíve na jeho hodnote minimalizačnej funkcie. V najhoršom prípade algoritmus vykoná $O(L \times T_l \times W)$ iterácií cyklu `for`. Kde L je počet liniek, T_l je perióda linky a W je hodnota minimalizačnej funkcie počiatočného riešenia. V skutočnosti algoritmus pobeží rýchlejšie, vylepšenie je o viac ako o jedna.

4.2 Generovanie s použitím „discrete set“

Pri návrhu tohto algoritmu sme vychádzali z propagačného algoritmu podmienok navrhnutým v práci od Voorhoeve [5]. Algoritmus je založený na propagácii obmedzujúcich podmienok.

Podmienka sa vzťahuje na dve udalosti (príchod alebo odchod vlaku), ktoré sú obmedzené určitou diskrétnou množinou, t.j. nastanú po sebe s časovým rozdielom patriacim do tejto diskrétnej množiny. V našom prípade sú to podmienky typu formulované na prestupoch (pre prípojové vlaky) a podmienky pre previazané linky.

Navrhnutý algoritmus sa dá rozdeliť na viacero ucelených častí, ktoré podrobne popíšeme v nasledujúcich sekciách:

- `Propagation part` – propagačná časť algoritmu
- `Search part` – časť algoritmu vyhľadávajúca riešenie
- `Propagation method` – metóda upravujúca propagačnú maticu, je spoločná pre obe časti

4.2.1 Propagačná časť

Pseudokód propagačnej časti

Propagation part

```
- definitions
- transfers : defined as from line to line on specified station
- constraints : derived from transfer, contains discrete set
- I : set of {1,…,N} where N is number of lines used in constraints
- matrix : matrix of discrete set

- initialization
constraints := createConstraints(transfers)
constraints := createSetAndMinimizationFactor(constraints)
constraints := mergeEquivalentConstraints(constraints)

- matrix initialization
for i,j ∈ I do
    if i = j then
        matrix[i,j] := {0}
    else
        matrix[i,j] := {0,..T − 1}
    fi
od
foreach constraint on i,j ∈ I do
    matrix[i,j] := constraint.set
    matrix[j,i] := {0} – matrix[i,j]
od

- main part
propagate(matrix)
```

Popis propagačnej časti

V propagačnej časti sú vytvorené obmedzujúce podmienky, pre každý naplánovaný prestup jedna podmienka. Potom je pre každú podmienku inicializovaná diskrétna množina.

Ak podmienka vychádza z prestupu z linky l_1 na linku l_2 v stanici s, a minimálny čas na prestup sú 3 minúty a maximálne 10 minút, podmienku označíme $c(1 \longrightarrow 2)$ a bude mať tvar:

$$(t_2 + x_2) - (t_1 + x_1) \in \{3, \dots, 10\}_T$$

Kde:

- t_1 – je odchod linky l_1 z jej počiatočnej stanice
- x_1 – je príchod linky l_1 do prestupnej stanice s
- t_2 – je odchod linky l_2 z jej počiatočnej stanice
- x_2 – je odchod linky l_2 z prestupnej stanice s
- T – interval taktového cestovného poriadku

V tejto fáze rozšírime pôvodný algoritmus. Vzhľadom na našu minimalizačnú funkciu si okrem diskrétnej množiny musíme zapamätať aj prírastok k minimalizačnej funkcii pre jednotlivé prvky diskrétnej množiny. Je nutné zaviesť mapovanie, ktoré vyzerá nasledovne:

$$3 \dashrightarrow 3w_s^{1,2}$$
$$4 \dashrightarrow 4w_s^{1,2}$$
$$\dots$$
$$10 \dashrightarrow 10w_s^{1,2}$$

Kde $w_s^{1,2}$ je počet očakávaných pasažierov. Mapovanie je nutné kvôli následnému upravovaniu podmienok a tým aj diskrétnej množiny. Podmienky sú zlučované nasledujúcim spôsobom. Ak existuje podmienka $c(i \longrightarrow j)$ alebo $c(j \longrightarrow i)$, nová výsledná podmienka $c(i \longrightarrow j)$ bude mať diskrétnu množinu M_{ij}:

$$M_{ij} = \left(\bigcap_{c(i \longrightarrow j)} \{u_1, \dots, u_n\}_T\right) \cap \left(\bigcap_{c(j \longrightarrow i)} \{-v_m, \dots, -v_1\}_T\right)$$

Môže sa stať, že výsledná množina nemusí byť intervalom v množina celých čísel Pri týchto množinových operáciách je nutné prepočítať podobným spôsobom aj mapovanie, pri prieniku sa hodnoty prírastku pre rovnaký prvok z oboch množín sčítajú.

Pre potreby propagácie je vytvorená matica M diskrétnych množín, kde štandardne sú na diagonálne inicializované množiny $\{0\}$ a inde $\{0, .. T-1\}$. Inak sú pre každú podmienku $c(i \longrightarrow j)$ na do matice pozíciu i, j dosadená diskrétna množina M_{ij}, čiže $M[i, j] = M_{ij}$. Nakoniec v propagačná časti je zavolaná propagačná metóda.

Propagačnú časť ešte spomenieme v sekcii 4.2.4, v ktorej uvedieme jej modifikáciu, čím vzniknú rôzne varianty algoritmu.

4.2.2 Propagačná metóda

Pseudokód propagačnej metódy

Propagate method

```
- parameters
matrix : matrix of discrete set

- definitions
changed : boolean
I : set of {1,…,N} where N is number of train lines used in
constraints

- initialization
changed := true

- main loop
while changed do
    changed := false
    for i,j,k ∈ I and i ≤ j and k≠i,j do
        A := matrix[i,k] + matrix[k,j]
        if matrix[i,j] ⊈ A then
            matrix[i,j] := matrix[i,j] ∩ (A)
            matrix[j,i] := {0} – matrix[i,j]
            changed := true
        fi
    od
od
```

Popis propagačnej metódy

Propagačná metóda upravuje maticu M diskrétnych množín, ktoré vzišli z podmienok postupnými inicializačnými úpravami popísanými v predchádzajúcej sekcii 4.2.1. Tieto diskrétne množiny sú navzájom previazané. Majme tri previazané množiny a ich prvky $a \in M_{ij}, b \in M_{ik}, c \in M_{kj}$, ktoré predstavujú časové rozdiely medzi nimi, tak musí platiť $a = b + c$. Prvky a, b, c predstavujú časové rozdiely medzi dvomi udalosťami. Propagačná metóda eliminuje tie hodnoty, ktoré nemôžu vzniknúť týmto súčtom.

Definícia stabilnej matice

Matica *B* je stabilná práva vtedy, keď platí:

$$B[i,j] \subseteq \{0\}$$
$$B[i,j] = -B[j,i]$$
$$B[i,j] \subseteq B[i,k] + B[k,j]$$

Propagačná metóda vyrobí z matice *M* maximálnu stabilnú maticu *B*, pre ktorú platí $S_{ij} \subseteq M_{ij}$. Zo stabilnej matice už metóda nič neeliminuje. Ak nám po propagácii vznikne niekde v matici prázdna množina, je rozpropagovaná do celej matice. V tomto prípade môžeme povedať, že vstupné obmedzujúce podmienky boli príliž striktné, nie sú splniteľné a neexistuje pre ne riešenie.

Keďže ide o stabilnú maticu, stačilo by sa pri iterácií cyklu `for` obmedziť na podmienku $i < j$. My sme použili podmienku $i \leq j$ preto, aby sa propagovanie prázdnej množiny dostalo aj na diagonálu. Môžeme sa obmedziť aj na $k \neq i,j$. Ak $i = j$, tak $M[i,i] \subseteq M[i,k] + M[k,i]$ je určite splnené, a ak $i = k$, tak $M[i,j] \subseteq M[i,i] + M[i,j]$ je taktiež splnené.

Sčítanie diskrétnych množín je definované ako $U + V = \{u + v | u \in U, v \in V\}_T$.
Unárna operácia je definovaná ako $-M = \{0\} - M$.

4.2.3 Časť vyhľadávajúca riešenia

Pseudokód časti vyhľadávajúcej riešenie

Search part (Backtracking) with specific best searcher

```
- paramaters
matrix : matrix of discrete set

- main loop
while (true) do
    propagate(matrix)
    if matrix is not valid then
        return
    fi

    bestRecord := choose best record from Set
    if bestRecord was not found then
        solution found
        end of algoritm
    fi

    fixOnePotentialOfSet(matrix, bestRecord)
    search(matrix)
    removedFixedPotentialOfSet(matrix, bestRecord)
od
```

Vyhľadávacia časť fixuje v každom kroku jednu množinu tak, že z nej zostane na danej pozícii v matici len jednoprvková množina. Pri kroku fixovania vyberieme vhodnú diskrétnu množinu, s najväčším rozsahom prírastkov k minimalizačnej funkcii pre jednotlivé prvky. Vo vybranej množine zafixujeme prvok s minimálnym prírastkom. (Táto časť výberu je predmetom vytvárania variant algoritmu rozoberaných v sekcii 4.2.4)

Po fixovaní aplikuje na propagačnú maticu propagačnú metódu. Ak je potom matica stabilná, pokračuje vo fixovaní výberom inej množiny. Ak po propagácii nie je matica stabilná, predchádzajúci proces fixovania vráti spať, s tým že fixovaný prvok vyškrtne. Algoritmus pokračuje ďalším krokom.

Pri každom úspešnom fixovaní sa vyeliminuje viacero prvkov z diskrétnych množín, ktoré boli previazané s týmto prvkom (tvorili spoločný súčet popísaný v popise propagačnej metódy v 4.2.2).

Riešenie algoritmu

Riešenie algoritmu predstavuje jeden riadok upravenej matice, ktorá v časti algoritmu vyhľadávajúcej riešenie má po eliminácii všade jednoprvkové množiny. Hodnoty jednoprvkových množín dvoch riadkov sa voči seba líšia len o časový posun.

```
Construction of Timetables
```

```
- parameters
matrix : matrix of discrete with singletons only

- solution
matrix[0] represents solution
```

Konečnosť algoritmu

Algoritmus je konečný. Horná hranica všetkých prehľadávaní je obmedzená TL^2, kde T je perióda CP a L je počet liniek tvoriacich obmedzujúcu podmienku (linky, ktoré nie sú viazané žiadnou podmienok môžeme z propagačnej matice vynechať).

4.2.4 Varianty algoritmu

Druhý navrhnutý algoritmus má viaceré varianty, ktoré vznikli až vo fáze implementácie. Boli navrhnuté rôzne postupy, ktoré pozmenili hlavné časti algoritmu `propagate part` a `search part`. Pri jednotlivých možnostiach diverzifikácie uvedieme aj anglický názov ako identifikátor, ktorým potom sformulujeme výsledné varianty.

Možnosti diverzifikácie

V súvislosti s vytváraním diskrétnych množín pre obmedzujúce podmienky vznikli možnosti:

`SameMaxTransferTime`

- Diskrétne množiny pre všetky dvojice udalostí majú iniciálny tvar $\{0, \dots, M\}$, t.j. všetky obmedzujúce podmienky pre prestupy zaručujú rovnaký maximálny čas M.

`AlphaTMaxTransferTime`

- Diskrétne množiny pre prestup (t_1, t_2, s) majú tvar $\{0, \dots, \alpha \cdot T\}_T$, kde $T = min\{t_1.T, t_2.T\}$ a $t_i.T$ je perióda linky t_i. Obmedzujúce podmienky pre

prestupy tak zaručujú čas $\alpha \cdot T$. Pre dve rôzne inštancie diskrétnych množín to môže vyzerať napríklad takto: $\{0, \dots, 30\}_{120}$ a $\{0, \dots, 15\}_{60}$.

`FullDiscreteSets`

- Diskrétne množiny inicializuje všetky $\{0, \dots, T-1\}_T$, kde T je perióda taktového CP.

Propagačnú časť sme modifikovali tak, že sme pridali jeden voľný parameter M (identický ako v prvom bode predchádzajúceho odseku), ktorý predstavuje variabilnú veľkosť množiny. Vznikli nasledujúce možnosti:

`BisectionPropagator`

- Metódu polenia intervalu sme aplikovali na parameter M, a tak vytvorili minimálnu maticu diskrétnych množín, ktorá je po propagácii ešte stabilná.

`SimplePropagator`

- Druhý variant uchováva propagačnú časť algoritmu v nezmenenú.

V súvislosti s časťou algoritmu vyhľadávajúcou riešenie sme rozlíšili možnosti pri výbere vhodnej diskrétnej množiny a kandidáta z nej.

`DeterministicSearcher`

- Si vyberá kandidáta na fixovanie z množiny s najväčším rozsahom prírastkov, ktorá má absolútne najväčší rozsah hodnôt diskrétnej množiny.

`ProbablisticSearcher`

- Vyberá množinu z Top10 najvhodnejších množín (t.j. s najväčším rozsahom) s určitou pravdepodobnosťou v závislosti od hodnoty rozsahu. Kandidát

Kombináciou týchto možností sme vytvorili nasledujúce varianty algoritmu.

Variant A

Variant A používa nasledujúce možnosti:

- `BisectionPropagator`
- `SameMaxTransferTime`
- `DeterministicSearcher`

Variant B

Variant B používa nasledujúce možnosti:

- `BisectionPropagator`
- `AlphaTMaxTransferTime`
- `DeterministicSearcher`

Variant C

Variant C používa nasledujúce možnosti:

- `SimplePropagator`
- `FullDiscreteSets`
- `DeterministicSearcher`

Variant D

Variant D používa nasledujúce možnosti:

- `SimplePropagator`
- `FullDiscreteSets`
- `ProbablisticSearcher`

5 Užívateľská dokumentácia

Súčasťou tejto bakalárskej práce je programu Periodic Timetable Generation (ďalej len pod skratkou PTG). V tejto kapitole je uvedený popis procesu jeho inštalácie, samotného programu, jeho základné použitie a následne popis jednotlivých funkcií programu.

5.1 Inštalácia a spustenie programu

Program PTG je určený pre platformu Windows, verzie operačného systému XP/Vista/7. Má vlastný inštalátor, ktorý užívateľ spustí súborom **setup.exe**. Inštalátor pred procesom samotnej inštalácie programu PTG overí, či je na cieľovom nainštalovaný.NET Framework 3.5, potrebný pre spustenie programu. Ak nie je nainštalovaný, prípadne je prítomná len nevyhovujúca verzia, inštalátor sa snaží pripojiť na Internet, konkrétne na servery spoločnosť Microsoft, a doinštalovať potrebné súbory.

Po skončení inštalačného procesu **PTG.exe**, ktorý je umiestnený v adresári zvolenom užívateľom, inštalácia pridá zástupcu pre spustenie do ponuky štart a na plochu. Po spustený sa otvorí okno programu.

5.2 Popis programu

Tento program bol vytvorený s cieľom vygenerovať taktové cestovné poriadky, umožniť užívateľovi modifikovať vstupné údaje požadovaným spôsobom, a porovnať výsledky a efektívnosť v závislosti od použitého algoritmu. Program implementuje dva navrhnuté algoritmy na generovanie, celkovo v piatich verziách.

Užívateľské rozhranie

Užívateľské rozhranie programu predstavujú dve okná:

- Prvé poskytuje užívateľovi načítať vstupné údaje a modifikovať ich vo viacerých úrovniach, po každej fáze spracovania dát.
- Druhé riadi proces samotnej generácie a slúži na zobrazenie výsledných taktových CP.

Pojmy

Program je v angličtine, preto pre vysvetlenie uvedieme význam základných pojmov, ktoré sa vyskytujú v ďalšom texte užívateľskej dokumentácie. Ostatné pojmy, sú vysvetlené v nasledujúcom texte užívateľskej dokumentácie, kde sú zmienené podrobnejšie v súvislosti s ich použitím.

- **Train Line** – vlaková linka.

- **Connected Line** – previazaná linka. Medzi dvomi previazanými linkami je taký vzťah. Že keď posunieme čas odchodu z počiatočnej stanice o m minút, rovnako sa posunú m minút aj všetky linky s ňou previazané.

- **Train Station** – vlaková stanica.

- **Train Connection –** vlakové spojenie z medzi dvomi stanicami v jednom smere.

- **Route** – dráha, predstavuje sled použitých vlakových liniek pri vlakových spojeniach. Dráha, rovnako aj potrebné údaje k jej inicializácii, vznikne zoskupením viacerých vlakových spojení, ktoré sú majú rovnaké zoznamy konkrétne použitých vlakových liniek. (Pre potreby generovania je zaujímavý hlavne údaj o počte očakávaných pasažierov)

- **Transfer** – prestup z jednej linky na druhú v určitej stanici. Obsahuje údaj o počte očakávaných cestujúcich.

- **Timetable** – cestovný poriadok. Inštancia CP obsahuje konkrétne časy odchodov vlakových liniek z množiny $\{0,..,T-1\}$, kde T je perióda vypravovania vlakových súprav pre danú linku. Každý vygenerovaný CP má pre potreby porovnania uchovanú hodnotu minimalizačnej funkcie, spočítanú v procese generovania.

- **Line's Timetable –** linkový cestovný poriadok, je tvorený zoznamom staníc, v ktorých vlak zastavuje. Pre každú stanicu obsahuje čas odchodu počas dňa.

- **Station's Timetable** – staničný cestovný poriadok je tvorený časmi príchodu a odchodu vlakov konkrétnych liniek vzťahujúcich sa k príslušnej stanici. Pre niektoré vlaky je stanica počiatočná (konečná), preto obsahuje iba údaj o čase odchodu (príchodu).

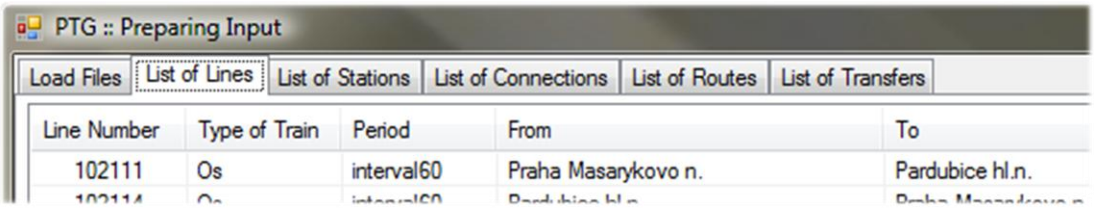
5.3 Základné používanie programu

Beh programu je lineárny. Naviguje užívateľa prejsť sekvenčne cez všetky taby jednotlivých okien. Každý tab predstavuj fázu programu, ktorá spracováva dáta a dáva možnosť modifikovať vstupné údaje. Niektoré taby majú len jednoducho prezentačný význam, nedochádza k modifikácii.

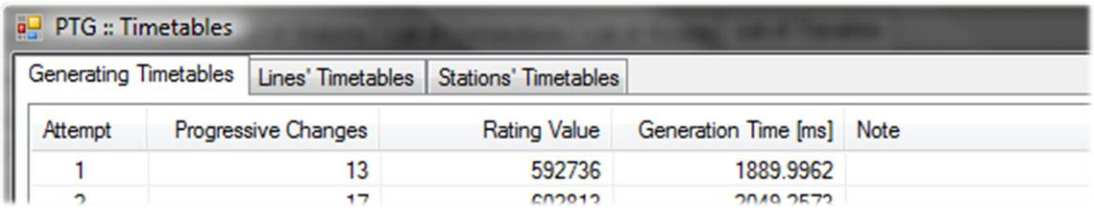
Do ďalšieho tabu sa úžívateľ dostane len stlačením tlačidla **Next**, ktoré sa nachádza v každom tabe okrem posledného. Pri prechádzaní tabmi v opačnom smere si môže užívateľ kliknúť na ľubovoľný tab, ktorý už navštívil: T.j. príslušná už fáza prebehla a tab je sprístupnený užívateľovi.

Spätná navigácia nachádza uplatnenie, či už v dôsledku porovnávania výsledku zo vstupnými údajmi, alebo vrátenie sa do určitej fázy programu za účelom modifikácie a opätovného spustenia následných fáz.

Prvé okno **PTG :: Preparing Input** je zobrazené po spustení programu. Po prechode všetkými tabmi prvého okna si užívateľ zvolí algoritmus (viď), a následne sa zobrazí druhé okno **PTG :: Timetables**. Taby oboch okien sú zobrazené na Obrázok 3 a Obrázok 4.



Obrázok 3: Taby v okne Preparing Input.



Obrázok 4: Taby v okne Timetables.

5.4 Načítanie vstupných údajov

Vstupné údaje ako vlakové linky, detaily vlakových staníc a previazané linky sa načítavajú z textových súborov.

Vlakové linky

Hlavnými vstupnými údajmi programu sú už existujúce linkové cestovné poriadky. Jedna linka je načítaná z jedného vybraného súboru. Pre jednotlivé linky sú obsahom súboru informácie o linke: číslo linky, perióda, typ vlaku a prevádzkovateľ, z toho povinnými údajmi sú prvé dve. Za týmito informáciami nasleduje zoznam zastávok, kde každý záznam obsahuje názov stanice, čas príchodu a odchodu zo stanice a počet kilometrov od počiatočnej zastávky.

Train Line format

```
TypeOfTrain#Os
TrainLineNumber#0709568
Period#120
Provider#CD
#
Mladá Boleslav hl.n.##9:51#0
Mladá Boleslav-Debř#9:57#9:58#5
Bakov n.Jizerou#10:03#10:04#9
Bakov n.Jizerou m.#10:06#10:07#10
Mnichovo Hradiště#10:13#10:14#16
Březina n.Jizerou##10:20#21
Loukov u Mn.Hradiště#10:22#10:23#23
Příšovice#10:26#10:27#26
Turnov#10:33##30
```

Súbory s vlakovými linkami pre potreby načítania sa dajú pridávať, respektíve odoberať, v tabe **Load Files** kliknutím na **Add Files**, respektíve označením a kliknutím na **Remove Files**.

Detaily vlakových staníc

Vstupnými údajmi sú aj detaily vlakových staníc. Údaje pre všetky stanice sú načítané z jedného súboru. Detail stanice obsahuje názov stanice, pre ktorú sú načítané dodatočné údaje, ako odhad počtu obyvateľov spadajúcich do aglomerácie stanice, a názov mesta, v ktorom sa stanica nachádza.

Odhad počtu obyvateľov slúži na generovanie počtu očakávaných pasažierov medzi každými dvomi mestami, z ktorého sa získa údaj počtu prestupujúcich cestujúcich na jednotlivých prestupoch.

Update Stations List format

```
Praha Masarykovo n.#200000#Praha
Praha-Vysočany#100000#Praha
Praha-Hor.Počernice#10000#Praha
Zeleneč#8000
Mstětice#2000
Čelákovice#5000
```

Detaily vlakových staníc sa načítavajú v tabe **List of Stations** kliknutím na **Update Categories** a následným vybraním príslušného súboru.

Previazané linky

Vzťahy medzi dvomi linkami sú načítané dodatočne. V jednom súbore sú uložené dvojice navzájom previazaných liniek. U týchto liniek platí symetria aj tranzitivita. Navzájom previazané linky tvoria triedy ekvivalencie, čiže nie je potreba explicitne zadávať všetky vzťahy liniek z jednej triedy.

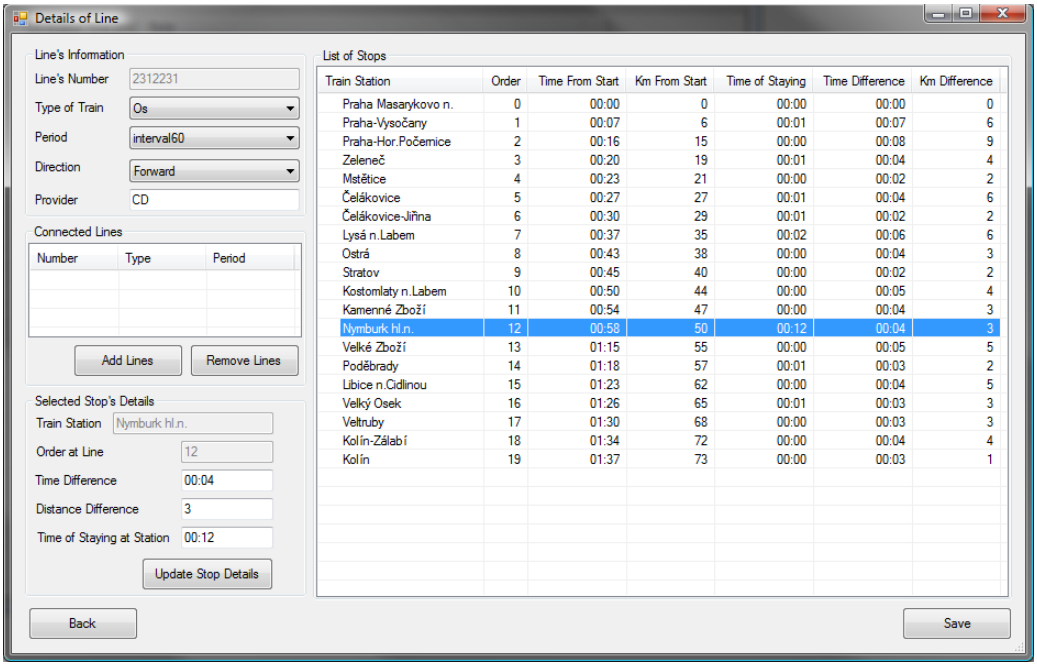
Connected Lines List format

```
0716050#0716051
2312226#2312231
2319442#2319445
```

Dvojice navzájom previazaných liniek sa načítavajú v tabe **List of Lines** kliknutím na **Update Connected Lines**, a následným vybraním textového súboru s uloženými dvojicami.

5.5 Editovanie položiek liniek

Načítané vlakové linky, respektíve detailné informácie o nich, sa dajú zobraziť z tabu **List of Lines** dvoj-kliknutím na vybranú linku alebo jej označením a kliknutím na **Details**. Na Obrázok 5 je screenshot okna detailu vlakovej linky **Details of Line**, ktoré sa následne zobrazí. Editovateľné sú položky: **Type of Train**, **Period**, **Direction** a **Provider.**



Obrázok 5: Detaily vlakovej linky

Previazané linky

Modifikovať zoznam previazaných liniek pre určitú linku sa dá v okne **Details of Line** (pre ilustráciu zobrazené na Obrázok 5). Pridávanie sa uskutoční kliknutím na tlačidlo **Add Lines,** následne sa otvorení okno **Add Lines** s ponúkanými možnosťami vlakových liniek (previazať je možné dve akékoľvek linky).

Odoberanie sa uskutoční označením linky v zozname **Connected Lines** a kliknutím na tlačidlo **Remove Lines**.

5.6 Zoznam staníc

Po preklikaní sa do tabu **List of Stations**, sa zobrazí zoznam všetkých staníc aj s príslušnými detailnými informáciami. Výberom vlakovej linky z rozbaľovacej ponuky **Select Line** sa zoznam staníc obmedzí iba na stanice, ktorými vybraná linka prechádza.

5.7 Editovanie položiek staníc

Vlakové spojenia sú vygenerované medzi všetkými dvojicami staníc v oboch smeroch. Pre možnosť modifikácie jednotlivých spojení je nutné v tabe **List of Connections** kliknúť na **Details.** Pre vlakové spojenie sú editovateľné tieto položky:

- **Category** – kategória vlakovej stanice vyjadruje zaradenie stanice v závislosti od počtu obyvateľov, respektíve potencionálnych cestujúcich, spadajúcich do jej aglomerácie.
- **Inhabitation** – vyjadruje počet obyvateľov, respektíve potencionálnych cestujúcich spadajúcich do aglomerácie stanice.
- **Town –** názov mesta, v ktorom sa stanica nachádza. Táto položka slúži pre potreby optimalizácie pri generovaní spojení. Spojenie medzi dvomi

stanicami, ktoré sa nachádzajú v jednom meste, sa neuvažuje. Predpokladá sa použitie alternatívnej mestskej dopravy.

- **Minimal Transfer Time** – minimálny čas na prestup medzi dvomi vlakovými linkami na stanici. Hodnota môže byť špecifická pre jednotlivé stanice v závislosti od viacerých faktorov (infraštruktúra, rozľahlosť, ...).

Poznámka:
Položky **Category** a **Inhabitation** sú zadávané komplementárne. Podľa zvolenej kategórie sa priradí do druhej položky príslušný preddefinovaný počet obyvateľov, a naopak. Pri zadaní, resp. modifikácii oboch súčasne, má vyššiu prioritu položka **Inhabitation**, a kategória sa nastaví zohľadňujúc zadaný počet obyvateľov.

5.8 Editovanie spojení

5.8.1 Zmena položiek spojenia

Pre vygenerovaní jednotlivých vlakových spojení je možné pre každé zmeniť očakávaní počet cestujúcich, a tým prispôsobiť modelované spojenie realite.

Napríklad dané spojenie je vyťaženejšie (čo do počtu očakávaných cestujúcich) v dôsledku infraštruktúry lokality staníc, ktoré sa nachádzajú v industriálnej, obchodnej alebo obytnej zóne so zvýšením dopytom po osobnej preprave.

Pre zobrazenie detailných informácii o vlakovom spojení je nutné v tabe **List of Connections** dvoj-kliknutím na vybrané spojenie otvoriť okno **Details of Connection**. Okno sa dá otvoriť aj označením spojenia a kliknutím na tlačidlo **Details**. V tomto okne sa dá modifikovať spomínaná položka **Passengers.**

5.8.2 Zmena trasy spojenia

Pre vygenerované vlakové spojenie sa dá meniť navrhnutá trasa. Nová trasa je pridávaná postupne sekvenčne po úsekoch. V každom kroku program navrhne úsek do ďalšieho uzla. Ak sa cieľová stanica posledného úseku bude zhodovať s cieľovou stanicou spojenia, program upozorní užívateľa, že nová trasa je korektná, a môže tak zameniť tú pôvodnú.

V tabe **List of Connections** treba označiť vybrané spojenie a kliknúť na tlačidlo **Edit Path.** V následne otvorenom okne **Edit Path of Connection** môže užívateľ ľubovoľne pridávať úseky kliknutím na **Add Stage** a v okne **Add Stage** zvoliť jeden z ponúkaných úsekov. Situácia je zobrazená na Obrázok 8. V každom kroku je pridaný jeden úsek, po ktorom program overí či aktuálna trasa postavaná s novo pridaných úsekov je platná. Výsledok zobrazí užívateľovi v položke **Path's Validity** (**is not valid** alebo **is valid**). Nová trasa sa dá uložiť len v platnom prípade.

Český Brod	Čelákovice-Jiňna	01:04	63	12	102114->2312231	Praha Masarykovo n.
Český Brod	Čelákovice	01:02	61	12	102114->2312231	Praha Masarykovo n.
Český Brod	Mstětice	00:58	55	5	102114->2312231	Praha Masarykovo n.

Obrázok 6: Pôvodne vygenerované spojenie z Českého Brodu do Čelákovic

Pre lepšie predstavenie si danej situácie uvádzam ilustračný príklad.

Príklad:

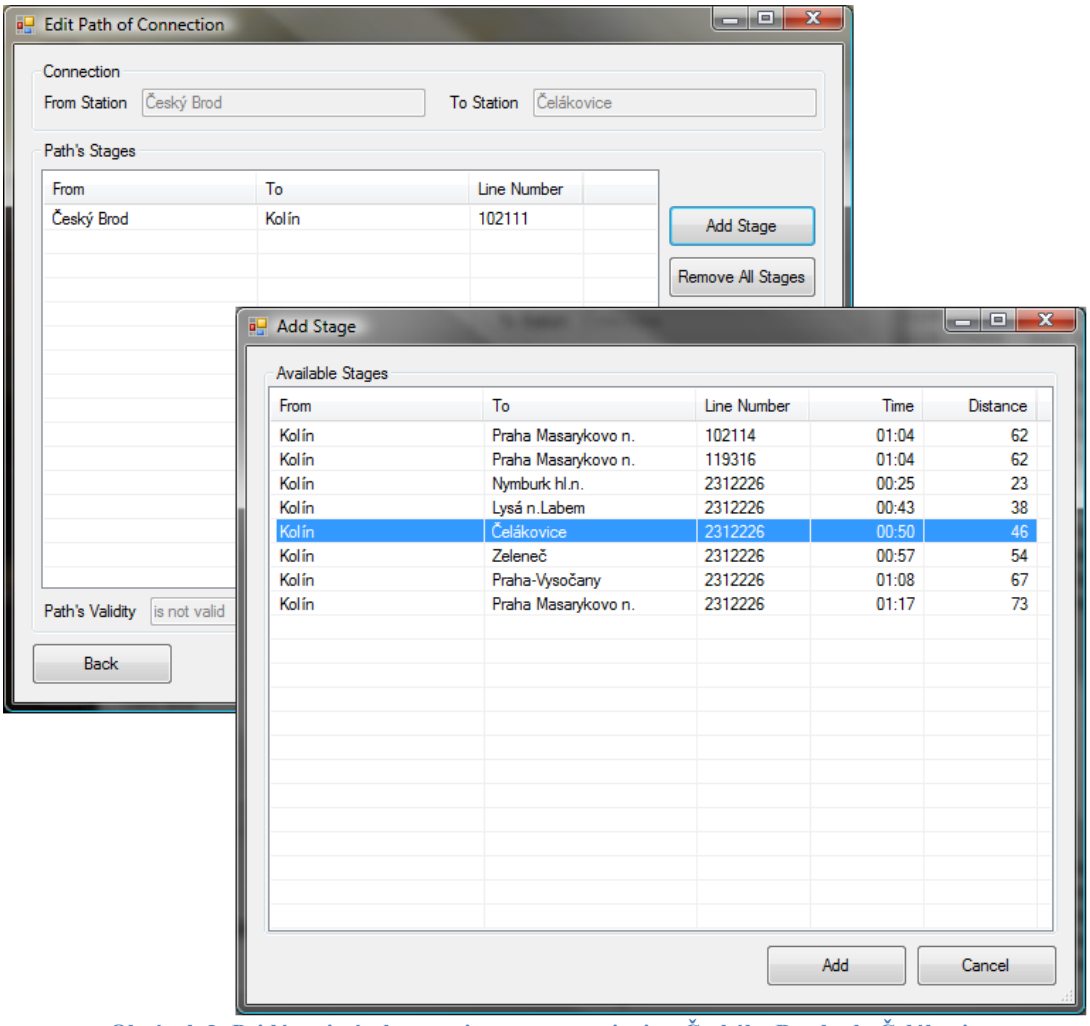
Program PTG vygeneroval vlakové spojenie zo stanice *Český Brod* do stanice Čelákovice s prestupom v stanici *Praha Masarykovo n.*, použijúc linky *102114* a *2312231*. Situácia na Obrázok 6.

Užívateľ sa rozhodol , že chce trasu spojenia modifikovať použijúc spojenie linkami *102111* a *231226* s prestupom v stanici *Kolín*. Situácia na Obrázok 7.

Do celkové súčtu pasažierov pri vytváraní prestupov, ktoré budú použité vo fáze generovania, tak nebude zarátaná hodnota 12 (očakávaných cestujúcich) na prestupe z linky *102114* na *2312231*. Namiesto toho bude z nového spojenia zarátaná hodnota 7, na prestupe z *102111* na *231226*.

Český Brod	Čelákovice-Jiňna	01:04	63	12	102114->2312231	Praha Masarykovo n.
Český Brod	Čelákovice	01:18	74	7	102111->2312226	Kolín
Český Brod	Mstětice	00:58	55	5	102114->2312231	Praha Masarykovo n.

Obrázok 7: Upravená trasa spojenia z Českého Brodu do Čelákovic



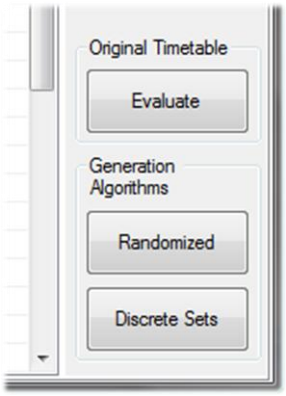
Obrázok 8: Pridávanie úseku novej trasy pre spojenie z Českého Brodu do Čelákovic.

5.9 Generovanie cestovných poriadkov

Fáza generovania CP sa skladá z výberu algoritmu a spustenia samotného generovania.

5.9.1 Výber algoritmu

Užívateľ má na výber z viacerých implementovaných algoritmov, ktoré si volí kliknútím na príslušné tlačidlo algoritmu v okne **PTG :: Preparing Input** na tabe **List of Transfers** v skupine tlačidiel **Generation Algorithms**. Následne sa otvorí okno **PTG :: Timetables**. Ďalší postup sa už líši v závislosti od zvoleného algoritmu. Okrem algoritmov má užívateľ možnosť zvoliť si ohodnotenie originálneho CP minimalizačnou funkciou, a porovnať s vygenerovanými CP . Situácia možnosti výberu je na Obrázok 9.

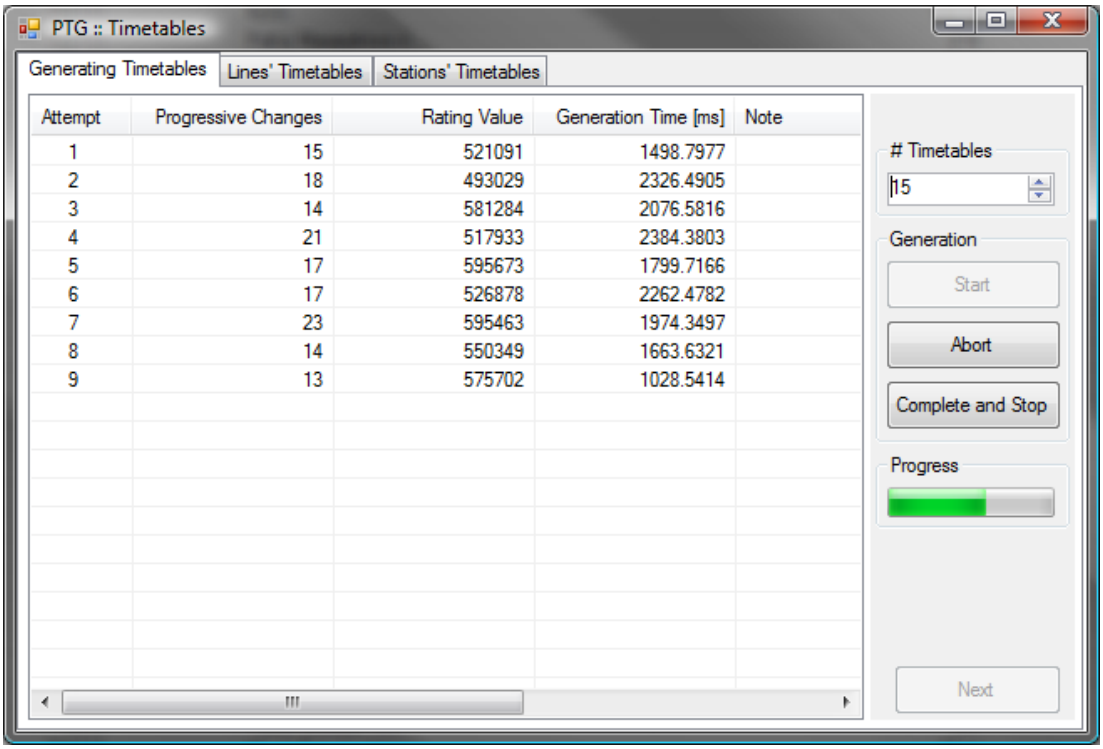


Obrázok 9: Možnosti výberu

Poznámka:
Okien **PTG :: Timetables**, kde prebieha generovanie pomocou navrhnutého algoritmu a prezentácia výsledných CP môže byť otvorených viacero. Užívateľ tak má lepšie možnosti na porovnanie algoritmov.

5.9.2 Generovanie

V tabe **Generating timetables** má užívateľ spočiatku možnosť iba nastaviť položku **#Timetable** a kliknúť tlačidlo **Start,** a tým spustiť proces generovania. V procese generovania je užívateľ informovaný o priebehu progres barom, a súčasne môže proces prerušiť kliknutím na **Abort**. Prerušením sa zobrazia len doteraz kompletne vygenerované CP. Priebeh generovania je zobrazený na Obrázok 10. Pre každý záznam predstavujúci CP sú uvedené informácie z procesu generovania.



Obrázok 10: Priebeh generovania CP zvoleným algoritmom.

Randomized with local search

Pre tento zvolený algoritmus si môže užívateľ nastaviť počet CP v položke **#Timetables**, ktoré budú vygenerované náhodne a následne vylepšené.

Discrete Set Algorithm

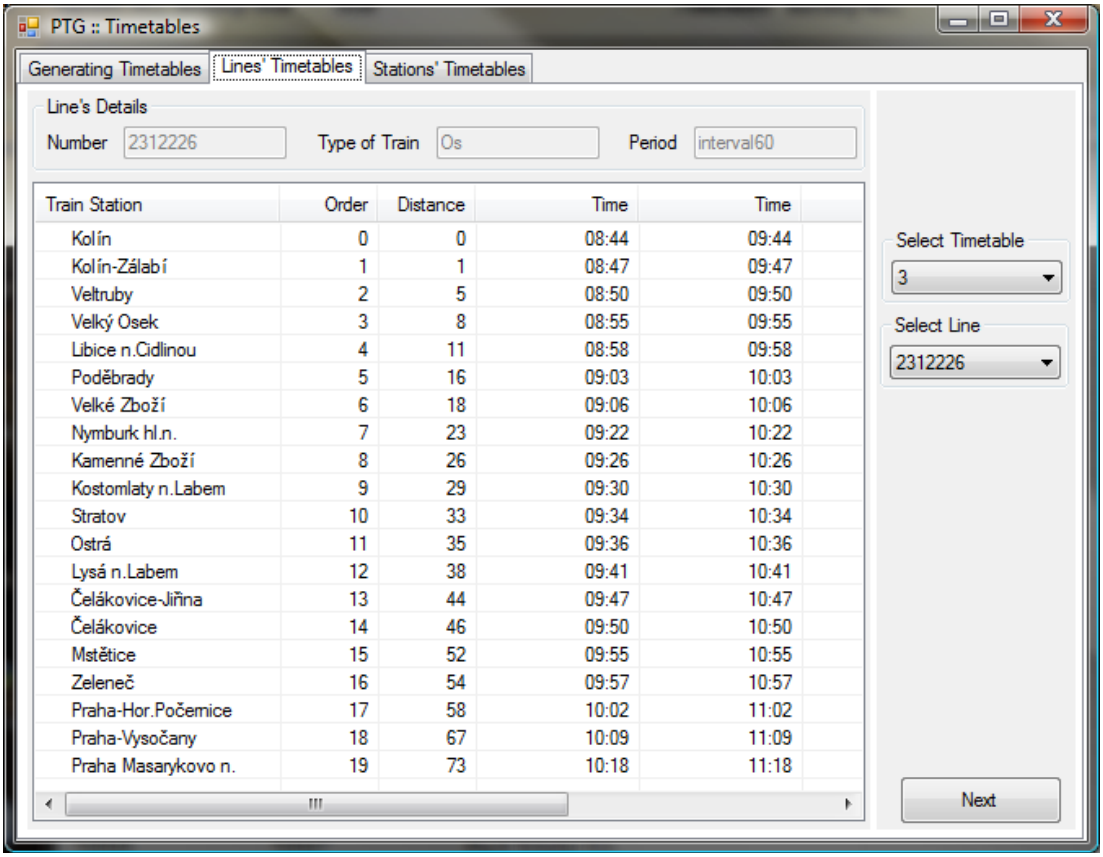
Tento zvolený algoritmus má 4 varianty, ktoré sú podrobnejšie popísané v sekcii 4.2.4. Každý variant nájde jedno riešenie, taktový CP, a proces algoritmu je tým ukončený. Jednotlivé výsledné záznamy o CP sú odlíšené v zozname poznámkou **Note**, t.j. akým variantom algoritmu boli vygenerované.

5.10 Prezentácia výsledných CP

Vygenerované taktové CP sú pre potreby zobrazenia konkrétne vypočítané od istej hodiny dňa. Obecne je stanovená na 08:00. Štruktúra taktových CP je pre každú periódu rovnaká. Preto je rozpočítanie časovania od istej konkrétnej hodiny pre jednotlivé linky urobené pre komfortnejšie porovnanie výsledkov užívateľom.

Linkové CP

V tabe **Lines' Timetables** je pre užívateľa dostupný taktový CP z pohľadu liniek. CP si užívateľ zvolí z vygenerovaných podľa ID v rozbaľovacej ponuky **Select Timetable**,a konkrétnu linku z rozbaľovacej ponuky **Select Line**. Situácia je zobrazená na Obrázok 11.



Obrázok 11: Prezentácia linkového CP.

Staničné CP

V tabe **Stations' Timetable** je pre užívateľa dostupný taktový CP z pohľadu staníc. CP si užívateľ zvolí z vygenerovaných podľa ID v rozbaľovacej ponuky **Select Timetable**, a konkrétnu stanicu z rozbaľovacej ponuky **Select Station.** Zobrazenia sa príchody a odchody vlakových liniek vzťahujúce sa na vybranú stanicu. V zobrazovacom okne so záznamami je možnosť triediť zoznam podľa rôznych stĺpčekov.

6 Implementácia

6.1 Technológie

Ako cieľová platforma programu PTG bol zvolený operačný systém Windows. Program je naprogramovaný v jazyku C# s použitím platformy .NET Framework 3.5. Ako vývojové prostredie bolo použité Microsoft Visual Studio 2008. Na vytvorenie užívateľského rozhrania bola použitá knižnica WinForms.

6.2 Dátové štruktúry

V tejto sekcii popisujem dátové štruktúry, funkcie jednotlivých tried a vzťahy medzi nimi. Pre detailnejší popis tried, ich položiek a metód s podrobnými komentármi, odporúčame pozrieť zdrojové súbory v projekte **PTG.sln**, ktorý je priložený na DVD nosiči. Rovnako na nosiči sa v priečinku **Documentation** nachádza detailná dokumentácia projektu vygenerovaná programom Doxygen[4].

6.2.1 Triedy reprezentujúce hlavné entity

V tejto sekcii sú popísané hlavné triedy celého programu reprezentujúce entity. Nasledovný zoznam popisuje hlavné triedy a ich dôležité položky. V položkách sú odkazy na inštancie príslušných tried.

`TrainLine`

- obsahuje detailné informácie o vlakovej linke, zoznam príslušných previazaných liniek `TrainLine` a zoznam zastávok `TrainStop`.

`TrainStop`

- obsahuje informácie o príchode, odchode vlaku určitej linky do stanice `TrainStation`, dobu státia v stanici , poradie zastávky v rámci linky a na akom kilometri sa od počiatočnej zastávky nachádza.

`TrainStaiton`

- stanica obsahuje minimálny čas na prestup, názov stanice, mesto v ktorom sa nachádza, kategóriu v závislosti od mesta `TownCategory`, počet obyvateľov spadajúcich na stanicu, zoznam liniek `TrainLine` prechádzajúcich stanicou, zoznam prestupov `Transfer` na tejto stanici.

`TrainConnection`

- obsahuje informácie o počiatočnej a konečnej stanici `TrainStation`; zoznam úsekov `Stage` jednotlivých liniek, z ktorých je spojenie zostavené, počet očakávaných pasažierov.

`Transfer`

- obsahuje linky `TrainLine`, medzi ktorými je naplánovaný; stanicu `TrainStation`, v ktorej je naplánovaný; počet očakávaných cestujúcich.

[4] http://www.stack.nl/~dimitri/doxygen/

`Timetable`

- obsahuje informácie z priebehu generovania, a zoznam vlakových liniek `TrainLineVariable` tvoriaci konkrétny vygenerovaný CP.

`TrainLineVariable`

- trieda reprezentuje entitu pre konkrétny CP, obsahuje odkaz na príslušnú `TrainLine`; čas odchodu linky v počiatočnej stanice, ktorá je výsledkom generovania; a zoznam príslušných previazaných liniek `TrainLineVariable`.

6.2.2 Triedy reprezentujúce „cache“

K jednotlivým entitám je nutné pristupovať z viacerých miest programu, narábať s nimi ako s databázou údajov. Preto sme zvolili návrhový vzor využívajúci `Singleton` s vnorenou privátnou statickou triedou `SingletonHolder` [16]. „Cache“ je implementovaná pre hlavné entity dát, kde sú uložené všetky ich inštancie.

Triedy implementujúce „cache“:

- `TrainLineCache`
- `TrainStationCache`
- `TrainConnectionCache`
- `FinalInput` – obsahuje zoznam skupín spojení `GroupsOfConnection` a zoznam všetkých prestupov `Transfers`

6.2.3 Pomocné triedy

Pomocné triedy sú v návrhu odlíšené koncovkou `Util`. Implementujú statické metódy týkajúce sa triedy s rovnomenným názvom (Napr. `Stage` a `StageUtil`). Takýto spôsob návrhu bol zvolený preto, lebo jednotlivé entity sú navzájom poprepájané a niektoré metódy sa tak dajú volať bez nutnosti vytvorenia inštancie triedy. Statické obsluhujúce metódy, tak sú separované.

Špecifickými statickými triedami, ktoré implementujú pomocné metódy sú: `IOUtil`, `LogUtil`, `MergeSort` a `NewtonFormula`.

6.3 Hľadanie najkratších vlakových spojení

Na vyhľadávanie najkratších ciest sme použili algoritmus Floyd-Warshallov algoritmus, špeciálne pre nájdenie vlakových spojení medzi všetkými dvojicami staníc. Primárnym kritériom bol však čas (najrýchlejšia cesta), sekundárnym vzdialenosť (najkratšia cesta).

Implementácia

Algoritmus pre vyhľadávanie najkratších vlakových je obsluhovaný triedami:

`ShortestPathAlgorithm`

- implementuje metódu `generateAllConnections`, pre vyhľadanie všetkých spojení

- a optimalizačnú metódu `optimisePath`, ktorá upravuje cestu vygenerovaného spojenia (napríklad ak vznikne spojenie L1–L2–L1, tak ho nahradí cestou použijúc len L1, ak taká medzi koncovými stanicami existuje)
- trieda implementuje konverziu z liniek `TrainLine` na hrany a potom spätne z hrán skonštruuje vlakové spojenia `TrainConnection`

`FloydWarshall`
- implementuje samotný algoritmus metódou `calculateShortestPaths`

6.4 Navrhnuté Algoritmy

Oba navrhnuté algoritmy obsiahnuté v tejto práci spracovávajú rovnaké údaje a generujú štruktúrne rovnaké výsledky. Preto sme navrhli pre ne spoločné rozhranie a tak presne definovali metódy, vlastnosti a udalosti pre ich obsluhovanie. Popis a pseudokód oboch navrhnutých algoritmov sa nachádza v kapitole 4.

`IGenerationAlgorithm`
- metóda `generateTimetables` pre spustenie generovania CP
- vlastnosť `Timetables` vracajúca zoznam CP
- udalosť `OnProgressChanged` signalizujúca priebeh algoritmu

6.4.1 Randomized with local search

Prvý navrhnutý algoritmus je reprezentovaný triedou implementujúc spoločný interface pre generovacie algoritmy:

```
GenerationAlgorithmRandomized : IGenerationAlgorithm
```

6.4.2 Discrete Set Algorithm

Druhý navrhnutý algoritmus je implementovaný triedou:
`GenerationAlgortihmDSA : IGenerationAlgorithm`

Pomocné statické metódy sú pre tento algoritmus implantované v triede:
`GenerationAlgorithmDSAUtil`
- metóda `constructTimetables` vytvára pre konkrétne riešenia nájdené algoritmom inštancie taktových cestovných poriadkov `Timetable`
- metóda `createDiscreteSetMatrix` vytvárana maticu diskrétnych množín na základe vstupných obmedzujúcich podmienok `Constraint`

Algoritmus je založený na splňovaný podmienok, ktoré reprezentuje trieda:
`Constraint`
- je abstratkná trieda
- modeluje vzťah (podmineku) medzi dvomi udalosťami, respektíve medzi dvomi `TrainLine`
- obsahuje diskretnú množinu `Set`, špecifickú pre túto podmienku
- pre inicializačnú časť algoritmu, kde treba zlučovať ekvivalentné podmienky medzi sebou, sú implementované metódy `mergeConstraintWith`, `reverseConstraint`, `equivalentWith`, vykonávajúce operácie s nimi

Triedy pre konkrétne typy podmienok:
`TransferConstraint : Constraint`

- je vytvorený z konkrétneho prestupu `Transfer`
- implementuje tri rôzne varianty vytvárania diskrétnych množín pre prílušnú podmienku

`ConnectedLineConstraint : Constraint`

- implementuje obmedzujúce podmienky pre previazané linky

Diskrétna množina použitá v podmienkach je implementovaná triedou:
`Set`

- obsah diskrétnej množiny je uložený v generickej dátovej štruktúre `HashSet<int>`
- mapovanie pre prírastok k minimalizačnej funkcii pre príslušné prvky diskrétnej množiny je použitá generická dátová štruktúra `IDictionary<int,int>`
- pre manipuláciu s touto triedou sú implementované metódy `addition`, `intersectWith`, `unionWith`, `mergeWith`, `isSubsetOf`, `reverse`

6.4.3 Varianty algoritmu

Konkrétne varianty algoritmu sa od seba odlišujú v propagačnej časti. K tomuto účelu bolo vytvorené rozhranie `IConstraintPropagator`. V propagačnej časti sa vytvárajú diskrétne množiny špecifickým spôsobom, ktorý tiež predstavuje odlišnosť medzi týmito variantmi. K tomuto cieľu bolo vytvorené rozhranie.
`IConstraintSetCreator.`

Konkrétne typy pre propagačnú časť:
`SimplePropagator : IConstraintPropagator`

- implementuje jednoduchú propagačnú časť algoritmu

`BisectionPropagator : IConstraintPropagator`

- metódou polenia intervalu prispôsobuje veľkosť diskrétnych množín, pre potreby propagácie, hľadá tak minimálne množiny, po ktorých propagáciou vznikne platná matica

Konrétne typy odlišujúce spôsob vytvárania množín:
`SameTransferTime : IConstraintSetsCreator`

- diskrétna množina je vytváraná spôsob, že zaručuje (zhora obmedzuje veľkosť diskrétnej množiny) rovnaký čas na všetkých prestupoch

`AlfaTTransferTime : IConstraintSetsCreator`

- diskrétna množina je vytváraná spôsob, že zaručuje (zhora obmedzuje veľkosť diskrétnej množiny) čas na prestupoch v závislosti od periód liniek v podmienke

`FullDiscreteSet : IConstraintSetsCreator`

- pre všetky podmienky je vytvorenú diskrétne množiny s veľkosťou periódy CP

V druhej časti algoritmu, vyhľadávajúcej riešenie, sa od seba varianty algoritmu odlišujú spôsobom, akým vyberajú kandidáta, t.j. diskrétnu množinu, ktorej fixujú prvok, pre ktorý je prírastková hodnota minimalizačnej funkcie najmenšia. Pre tento účel bolo vytvorené rozhranie:

`IBestChoiceSearcher`

- definuje metódu `chooseBestRecord` na vybranie vhodného kandidáta z diskrétnych množín `Set`

Konkrétne triedy implementujúce rozhranie:
`DeterministicSearcher : IBestChoiceSearcher`

- metóda `chooseBestRecord` implementuje výber množiny `Set`, ktorá má absolútne najväčší rozsah hodnôt diskrétnej množiny

`ProbablisticSearcher : IBestChoiceSearcher`

- metóda `chooseBestRecord` implementuje výber množiny `Set` z Top10 najvhodnejších kandidátov s určitou pravdepodobnosťou v závislosti od rozsahu hodnôt diskrétnych množín

Volanie algoritmu pre špecifický variant algoritmu vyzerá napríklad nasledovne:

```
runSpecializedGenerationAlgorithm(
    constraints,
    new BisectionPropagator(new SameTransferTime()),
    new DeterministicSearcher(),
    timetables
    );
```

7 Analýza výsledkov

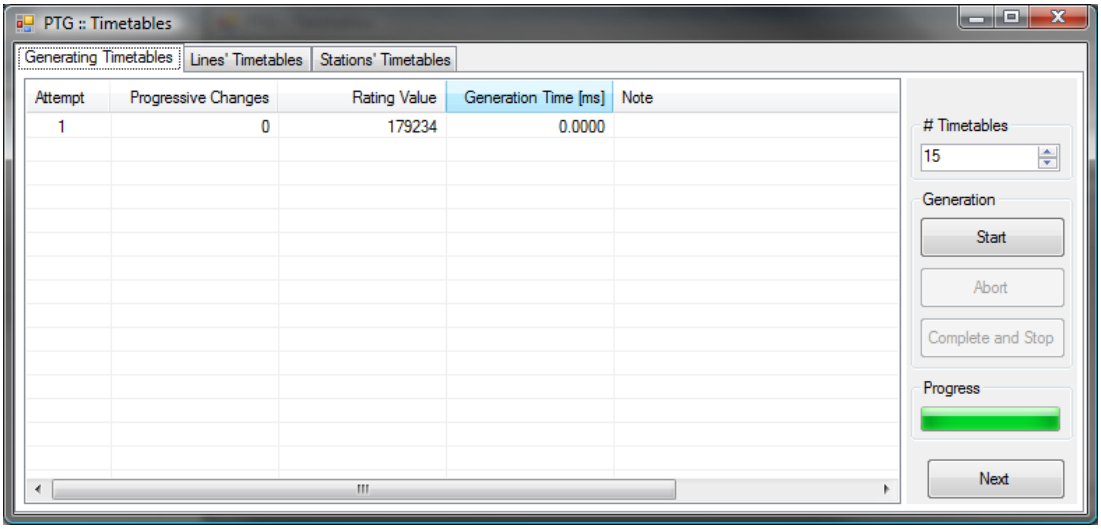
V nasledujúcich sekciách uvádzame výsledky generovania pre tri rôzne veľké železničné siete. Taktiež sme sa pokúsili o optimalizovanie CP liniek pražského metra voči našej minimalizačnej funkcii, ktorá optimalizuje cestovný poriadok vzhľadom k počtu pasažierov a doby času na prestupoch.

V sekcii, ktorá nasleduje po ukážkach výsledkov, porovnáme implementované algoritmy. Pri hodnotení algoritmov by sme sa nemali obmedziť iba na jediné testovacie dáta, preto poskytujeme aspoň štyri kolekcie dát.

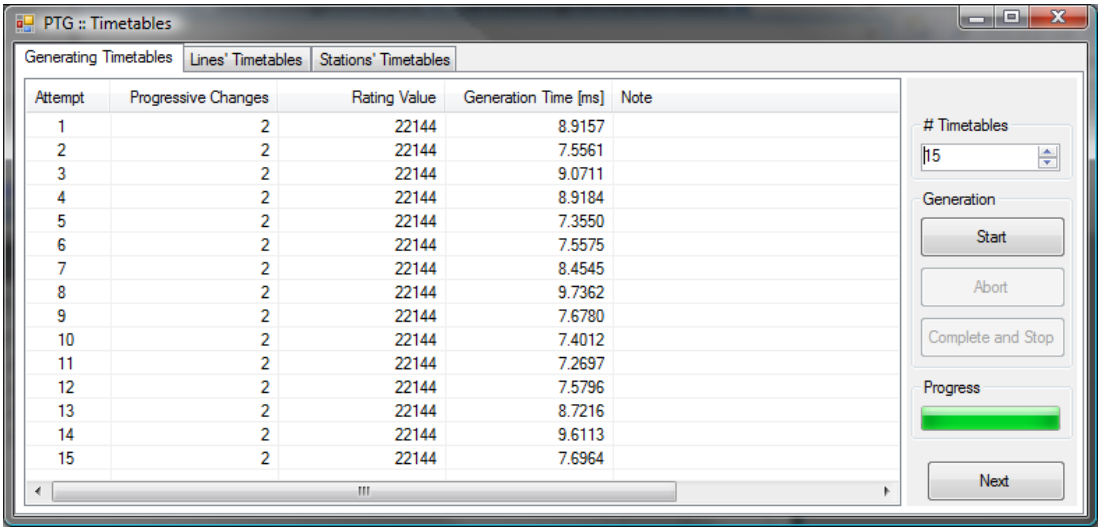
7.1 Testovanie na vstupných údajoch

7.1.1 Malá sieť

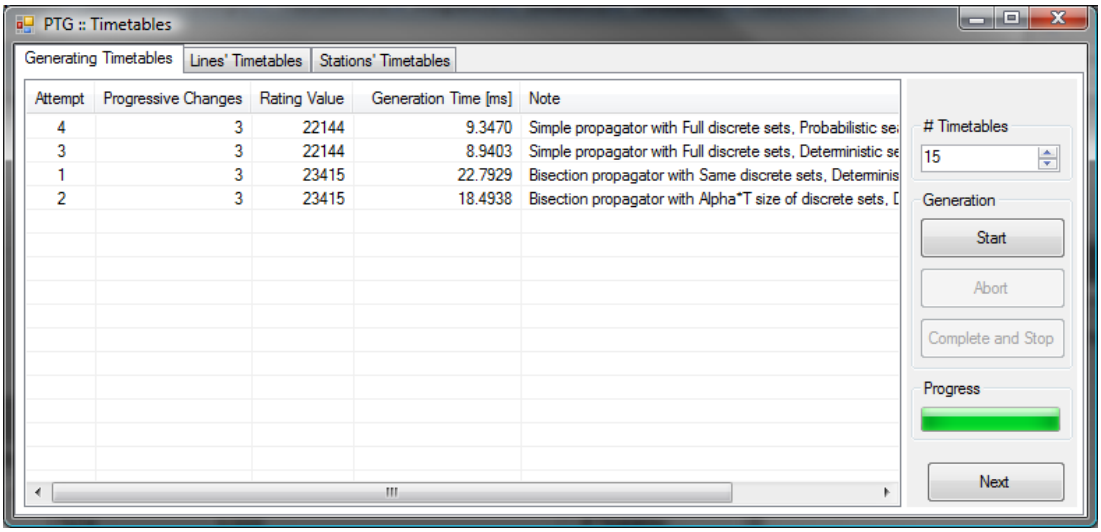
Prvé testovacie dáta - sieť pozostávajúca zo štyroch liniek medzi Prahou a Kolínom - sa nám podarilo optimalizovať až niekoľkonásobne. Je to spôsobené tým, že linky ako keby boli vytrhnuté z CP, neobsahujú tak ostatné náväznosti na iné trate a linky. Doba čakania sa znížila v priemere na 5 minút.



Obrázok 12: Ohodnotenie originálneho CP minimalizačnou funkciou pre malú sieť.



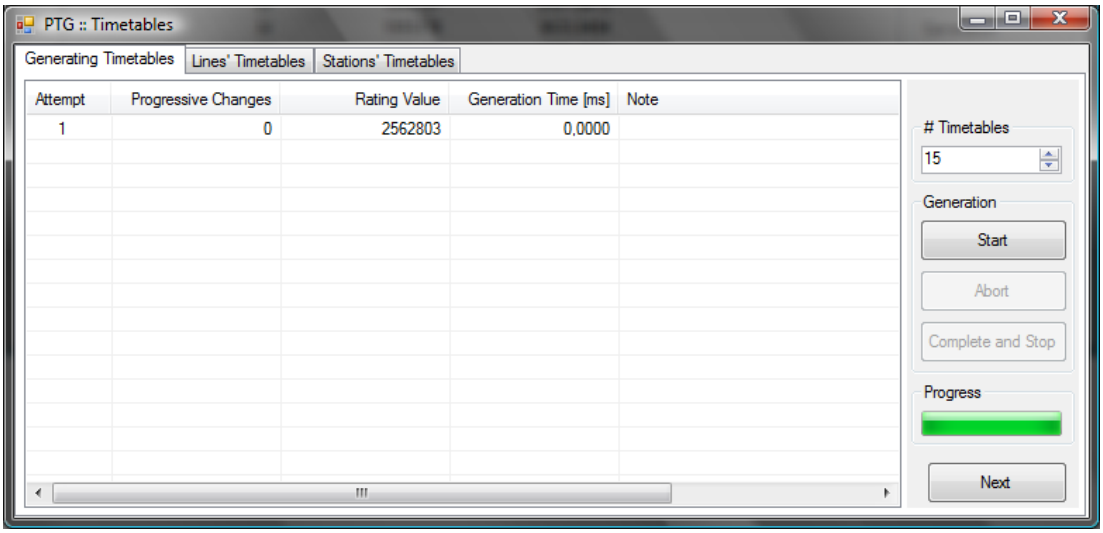
Obrázok 13: Výsledky generovania pre malú sieť náhodným algoritmom s vylepšovaním.



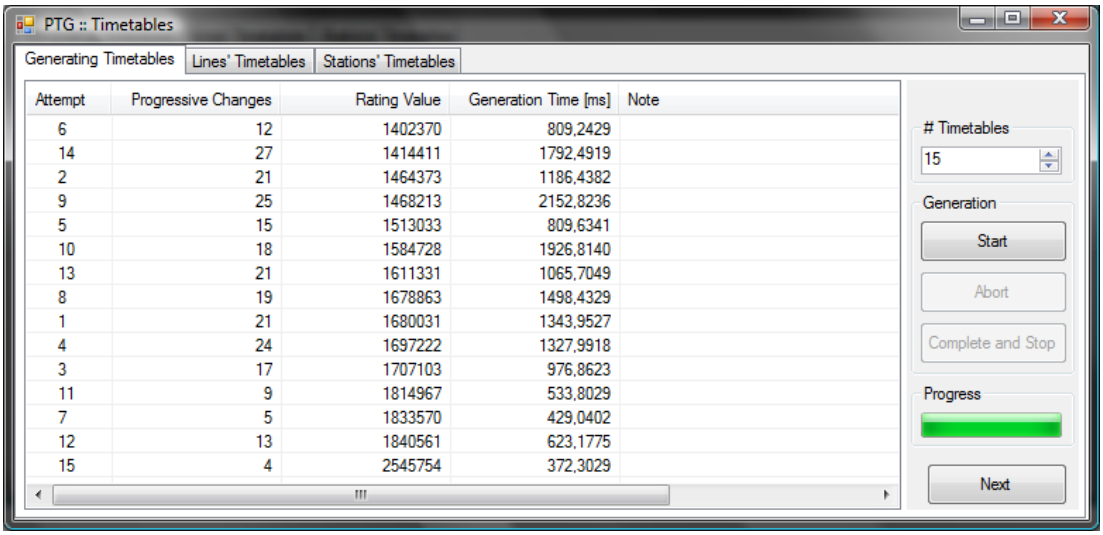
Obrázok 14: Výsledky generovania pre malú sieť algoritmom s použitím discrete set.

7.1.2 Sieť stredná

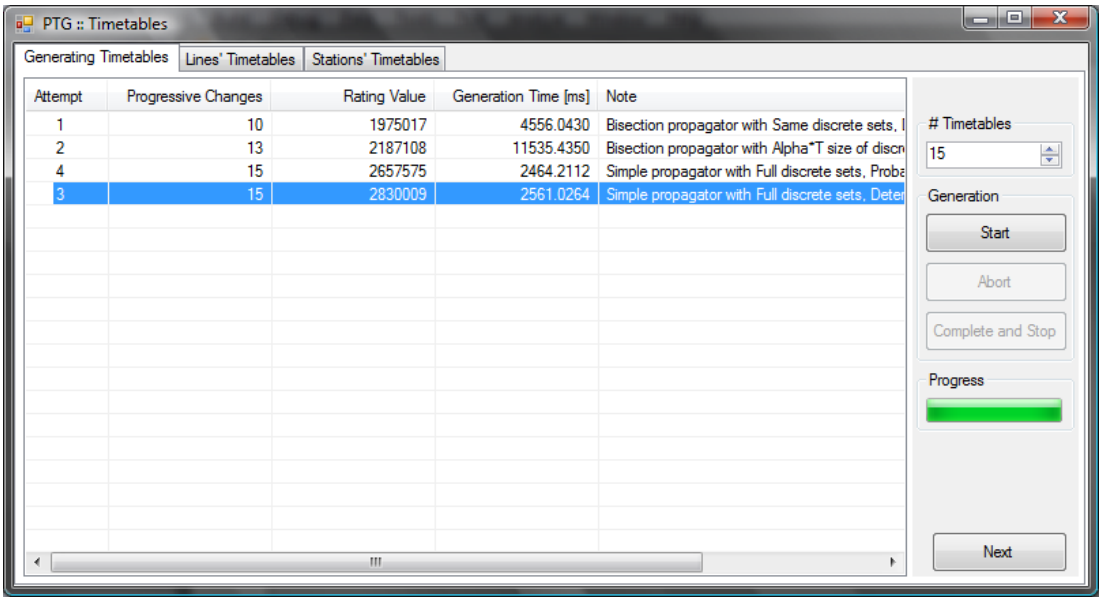
Sieť pozostáva z 18tich liniek.



Obrázok 15: Ohodnotenie originálneho CP minimalizačnou funkciou pre strednú sieť.



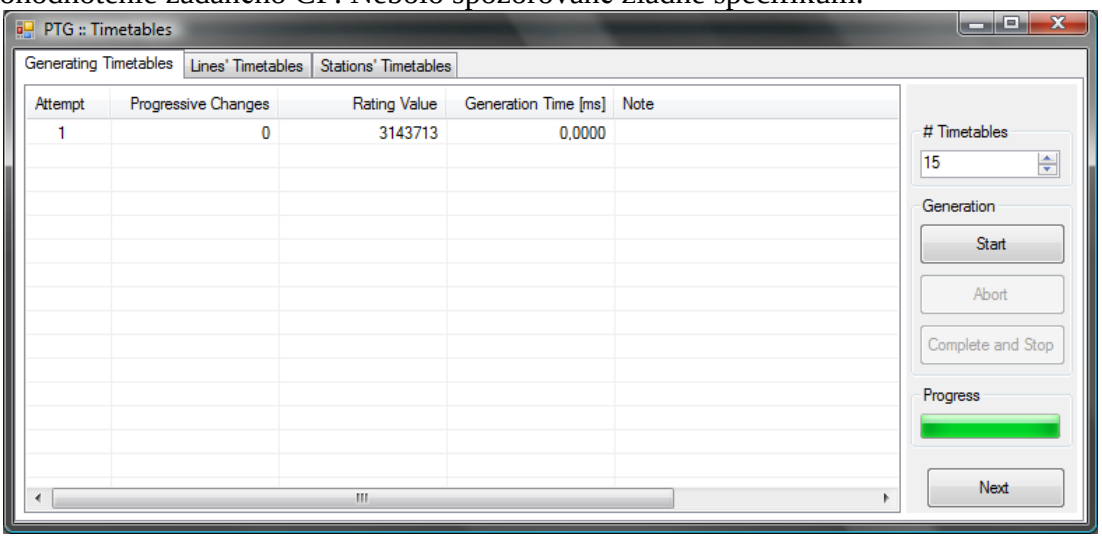
Obrázok 16: Výsledky generovania pre strednú sieť náhodným algoritmom s vylepšovaním.



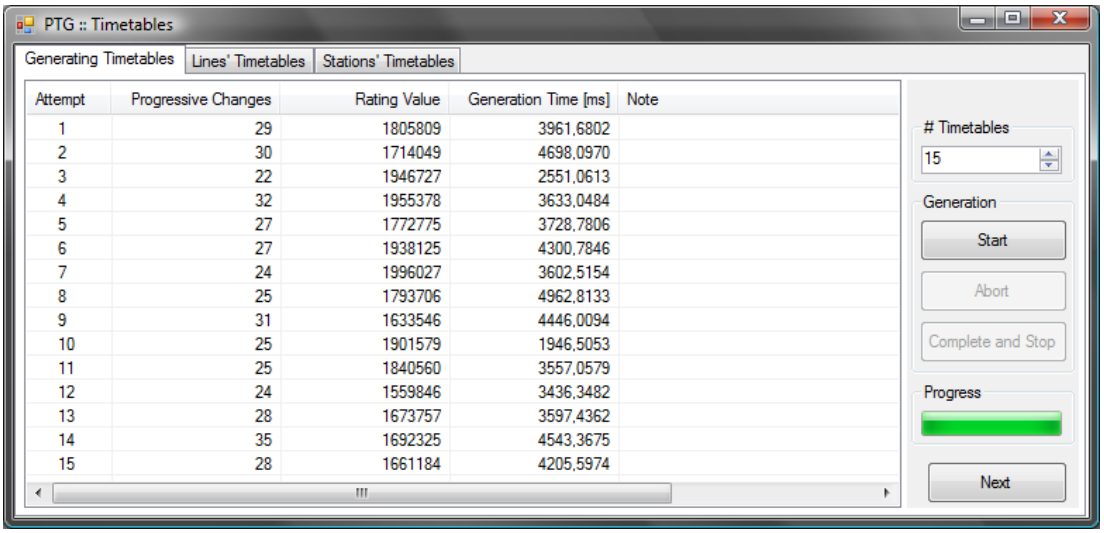
Obrázok 17: Výsledky generovania pre strednú sieť algoritmom s použitým discrete set..

7.1.3 Sieť veľká

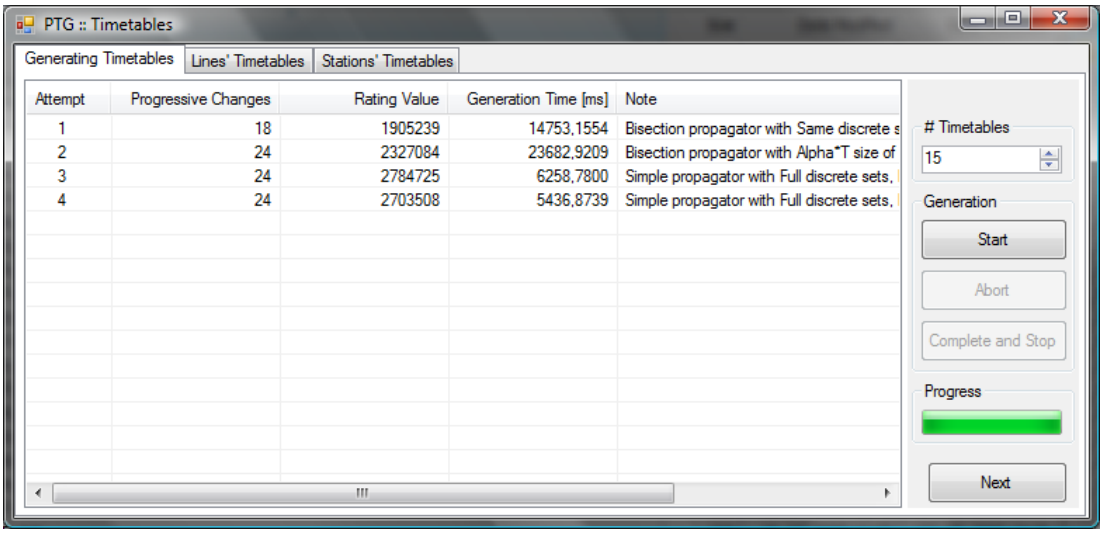
Sieť pozostáva z 29tich liniek. Pri tejto kolekcii dát oba algoritmy vylepšili pôvodné ohodnotenie zadaného CP. Nebolo spozorované žiadne špecifikum.



Obrázok 18: Ohodnotenie originálneho CP minimalizačnou funkciou pre veľkú sieť.



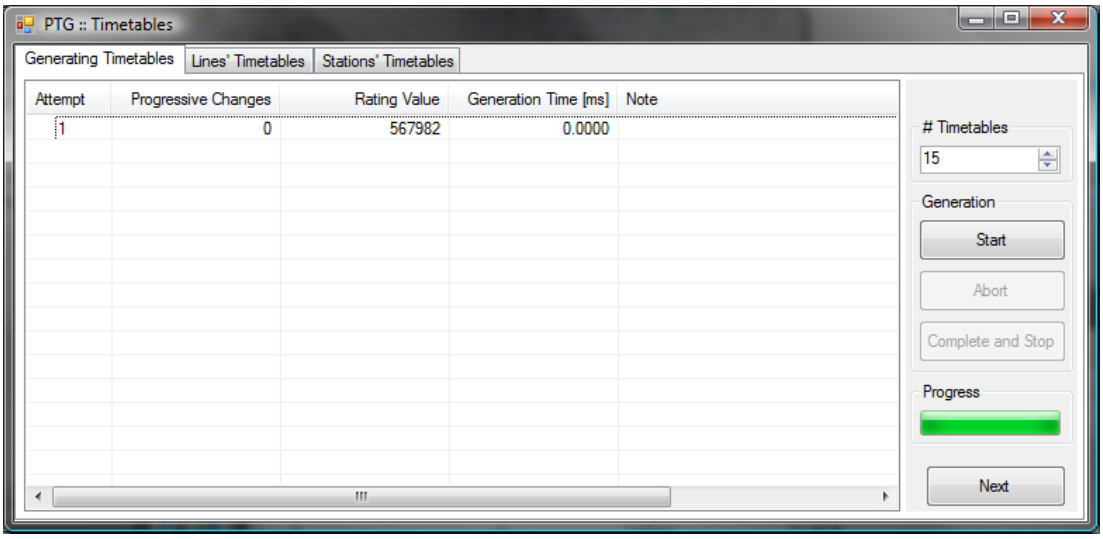
Obrázok 19 : Výsledky generovania pre veľkú sieť náhodným algoritmom s vylepšovaním.



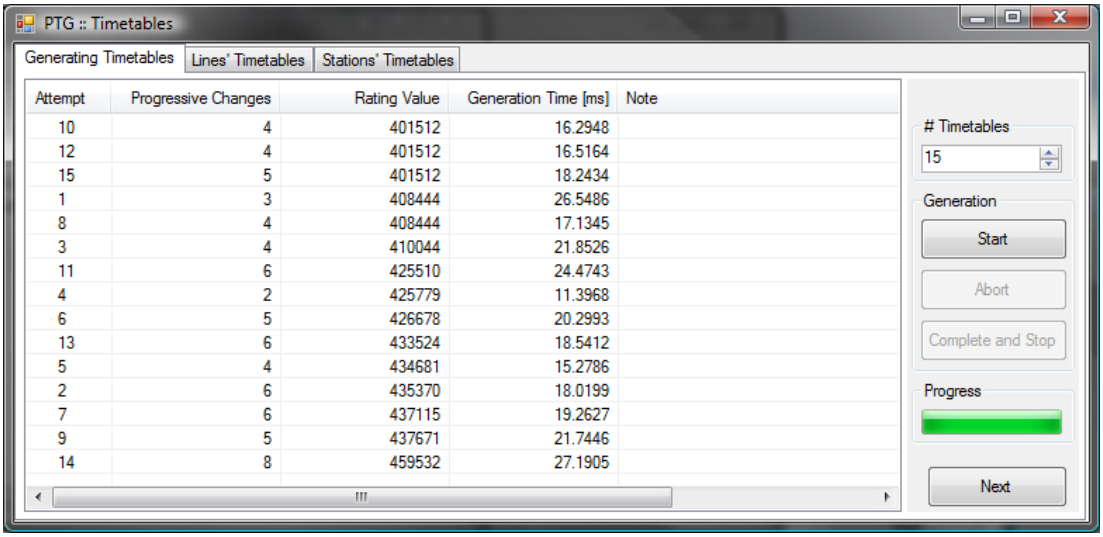
Obrázok 20: Výsledky generovania pre veľkú sieť algoritmom s použitím discrete set.

7.1.4 Pražské metro

Pôvodne bol celý projekt zameraný len na CP na železnici. Taktové CP sú však využívané aj v inej verejnej doprave, napr pri MHD. Preto sme sa pokúsili optimalizačný program PTG jemne upraviť, pridali sme periódu `interval10`, ktorý sme použili pre optimalizáciu večernej až nočnej prevádzky metra. Počas dennej prevádzky a v čase dopravnej špičky, keď sú intervaly odchodov metra 1,5 až 3 minúty, nemá optimalizácia veľký význam (už len pri uvážení minimálneho času na prestup, pre Muzeum sú to 2min, Florenc a Můstek po 3 min).



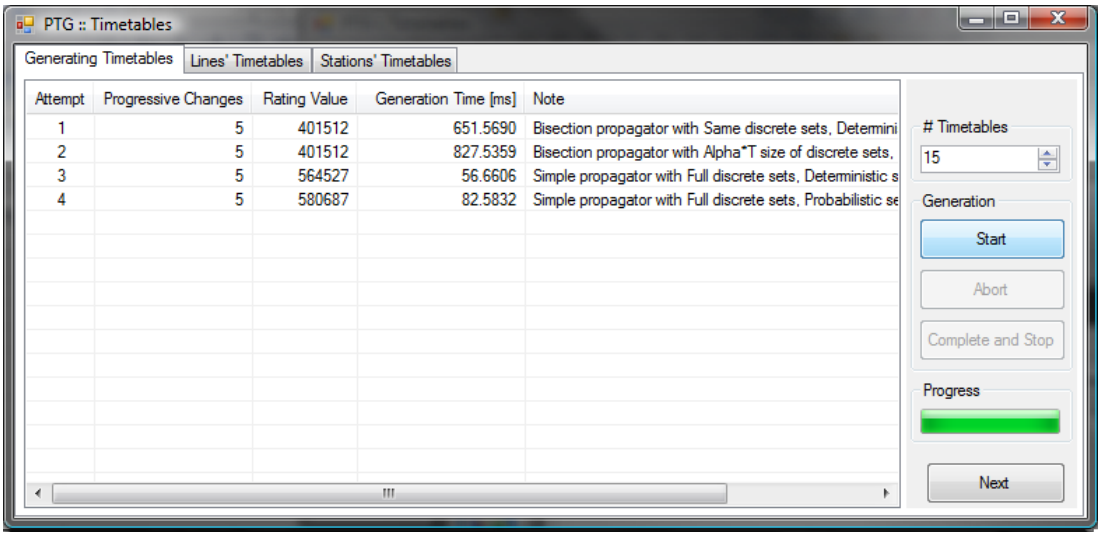
Obrázok 21: Ohodnotenie originálneho CP minimalizačnou funkciou pre pražské metro.



Obrázok 22: Výsledky generovania pre pražské metro náhodným algoritmom s vylepšovaním.

Pri porovnaní výsledkov sa nám pôvodný CP podarilo optimalizovať v závislosti od našej minimalizačnej funkcie (pri oboch našich algoritmoch až o 29% z pôvodnej hodnoty). Treba však spomenúť a zdôrazniť, že nami vytvorený model nezohľadňoval ostatné požiadavky a predpoklady, ktoré sme spomenuli v úvodnej druhej kapitole. Nezahŕňal bezpečnostnú reguláciu, otáčanie vlakových súprav a neoptimalizoval ani ich počet. Pražské metro má 3 prestupné stanice, a celkovo predstavuje 24 možných prestupov.

Naše výsledky dosahujú síce lepšie číselné hodnoty, avšak len pre nami nami definovaný model.



Obrázok 23: Výsledky generovania CP pre pražské metro algoritmom s použitím discrete set.

7.2 Porovnanie algoritmov

Algoritmus DSA (Discrete Set Algortihm) je špecifický v porovnaní s náhodným, že ak existuje riešenie pre zadanú inštanciu podmienok, nájde práve jedno riešenie – optimálne riešene (pre náš model).

Variant algoritmu s `BisectionPropagator` sa snaží vždy nájsť minimálnu maticu diskrétnych množín, ktorá je ešte stabilná, propagačná časť beží dlhšie v porovnaní s variantom so `SimplePropagator`.

`Deterministic` a `ProbalisicSearcher` nám podľa vygenerovaných štatistických údajov veľmi neovplyvňuje výsledky. Deterministický je však o čosi lepší.

Myšlienka zmeny veľkosti diskrétnych množín pri inicializácii pomocou spomínaných možností diverzity tiež našla opodstatnenie, prispela nám do štatistických údajov rôznorodosťou údajov.

Výsledky, ktoré prezentujeme, dokumentujú, že implementované algoritmy so zvolenými predpokladmi dokážu zlepšiť existujúci CP. Je paradoxné, že algoritmus DSA zvyčajne nedosiahne lepšie výsledky ako lokálne prehľadávanie (algoritmus randomized). Túto vlastnosť pripisujeme hlavne tomu, že DSA hľadá v prehľadávanom priestore iba prvé riešenie, aj keď s výrazným zacielením pomocou heuristiky výberu najlepšej množiny. To zrejme spôsobí zlepšenie existujúceho rozvrhu.

Vhodným pokračovaním a rozšírením by mohlo byť hľadanie nových heuristík, či ďalšie pokračovanie v prehľadávaní s technikami prerezávania ako je napríklad *branch and bound*. Podobným problémom trpí aj lokálne prehľadávanie. Tu sa nám poskytujú riešenia ako *simmulated annealing* alebo *random walk*. Naše riešenie spočíva v generovaní väčšieho množstva rozvrhov z náhodnej počiatočnej situácie.

8 Záver

Prínosom tejto práce pre mňa bolo pochopenie danej problematiky, ktorá je nesporne zaujímavá avšak pomerne rozsiahla. V tejto práci sme z nej obsiahli len malú časť, ktorá bola určená zadaním tejto práce. Generovanie a optimalizovanie cestovných poriadkov je určite dôležitá úloha, ktorá sa dnes rieši prakticky v celej Európe.

V porovnaní s Českou republikou majú krajiny západnej Európy, ako Nemecko či Holandsko, vyspelejšiu infraštruktúru, čo ústi do vyššej frekvencie odchodov liniek. V súčasnosti vznikajú štúdie ako môže dodatočná investícia do infraštruktúry ovplyvniť komfort cestovania a vylepšiť cestovné poriadky z pohľadu komfortu cestovania a bezpečnosti pasažierov, stability, vyrovnanosti, kapacity železničnej siete a v neposlednom celkovej rentability investovaných nákladov.

8.1 Nedostatky a možné rozšírenia

Rozšíriteľnosť druhého algoritmu je možná v rámci obmedzujúcich podmienok, keďže majú jednotnú formuláciu. Algoritmus však potrebuje prispôsobiť úvodne spracovanie podmienok, pre vstup do samotného algoritmu, a spracovanie výstupu po generovaní. Pre spracovanie podmienok je potrebné rozšíriť vstupné užívateľské rozhranie. Druhý algoritmus by mohol byť použitý aj na generovanie, ale vzhľadom k tomu, že nevyužíva dokonalejšie heuristiky, by pri väčších vstupných údajoch nedobehol v rozumnom čase.

Literatúra

[1] L. Peeters: *Cyclic Railway Timetable Optimization*, Erasmus university Rotterdam, 2003, ISBN 90-5892-042-9

[2] P. Serafini a W. Ukovich: *A mathematical model for periodic scheduling problems*, SIAM Journal on Discrete Mathematics 2, strany 550-581, 1989

[3] C. Liebchen and Rolf H. Möhring: *The Modeling Power of the Periodic Event Scheduling Problem:Railway Timetables — and Beyond*, TU Berlin, 2004

[4] J. C. Villumsen: *Construction of Timetables Based on Periodic Event Scheduling*, Kongens Lyngby, IMM-Thesis, 2006

[5] M. Voorhoeve: *Rail Scheduling with Discrete Sets*, Eindhoven University of Technology, Eindhoven, 1993

[6] Schrijver A. a Steenbeek, A.: *Timetable construction*. Technical report, CWI, Amsterdam, The Netherlands, 1993

[7] Schrijver A. a Steenbeek A.: *Timetable construction for Railned*. Technical report, CWI, Amsterdam, The Netherlands, 1994

[8] M. A. Odijk: *A Constraint Generation Algorithm for the Construction of Periodic Railway Timetables*, Transportation Research, strany 455-464, 1997

[9] R. Hassin: *A flow algorithm for network synchronization*. Operations Research, strany 570-579, 1996

[10] K. Nachtigall: *A branch and cut approach for periodic network programming*, Technical Report 29, Hildesheimer Informatik-Berichte, Hildesheim, Germany, 1994

[11] K. Nachtigall: *Cutting planes for a polyhedron associated with a periodic network*, Technical Report 17, DLR Interner Bericht, Braunschweig, Germany, 1996

[12] K. Nachtigall a S. Voget: *A genetic algorithm approach to periodic railway synchronization*, Computers and Operations Research, strany 453-463, 1996

[13] T. Lindner: *Train Schedule Optimization in Public Rail Transport*, (Ph.D. thesis), Technische Universitat Braunschweig, 2000

[14] Project B15 – Service Design in Public Transport, http://www.zib.de/Optimization/Projects/TrafficLogistic/Matheon-B15/Matheon-B15long2.en.html, 2006-2010 (aktuálne prebiehajúci)

[15] C. Liebchen and Rolf H. Möhring: *A Case Study in Periodic Timetabling*, Technische Universität□at Berlin, Institut für Mathematik, 2002

[16] http://www.thushanfernando.com/index.php/2008/06/27/design-patterns-in-c-java-the-singleton/ , august 2009

Prílohy na DVD nosiči

Adresár **Text BP**

- Text bakalárskej práce vo formáte .pdf.

Adresár **Install**

- Inštalačný súbor **setup.exe** programu PTG.

Adresár **Documentation**

- Programátorská dokumentácia vygenerovaná programom Doxygen.

Adresár **Test Data**

- Testovacie údaje pre program PTG.

Adresár **Solution**

- Zdrojový kód projektu.