Obsah

[1 Úvod 3](#_Toc237241180)

[1.1 Motivácia 3](#_Toc237241181)

[1.2 Cieľ práce 3](#_Toc237241182)

[1.3 Štruktúra práce 3](#_Toc237241183)

[2 Problematika generovania cestovných poriadkov 4](#_Toc237241184)

[2.1 Taktové cestovné poriadky 4](#_Toc237241185)

[2.2 Obmedzujúce podmienky 5](#_Toc237241186)

[2.2.1 Predpoklady 5](#_Toc237241187)

[2.2.2 Formulácia obmedzujúcich podmienok 7](#_Toc237241188)

[2.3 Periodic Event Scheduling Problem 10](#_Toc237241189)

[2.4 Fázy plánovania a PESP 10](#_Toc237241190)

[2.5 Známe algoritmy 11](#_Toc237241191)

[2.6 Existujúce systémy 12](#_Toc237241192)

[2.7 Prípadová štúdia 12](#_Toc237241193)

[2.7.1 Berlínske metro 12](#_Toc237241194)

[3 Formulácia riešeného problému 14](#_Toc237241195)

[3.1 Vstup 14](#_Toc237241196)

[3.2 Modifikácia vstupu 14](#_Toc237241197)

[3.2.1 Viacúrovňové zadávanie údajov 14](#_Toc237241198)

[3.3 Obmedzujúce podmienky 15](#_Toc237241199)

[3.4 Požadovaný výstup 15](#_Toc237241200)

[3.5 Minimalizačná funkcia 15](#_Toc237241201)

[3.6 Generovanie a optimalizácia 16](#_Toc237241202)

[4 Navrhnuté algoritmy 17](#_Toc237241203)

[4.1 Náhodné generovanie s vylepšovaním 17](#_Toc237241204)

[4.2 Generovanie s použitím „discrete set“ 19](#_Toc237241205)

[4.2.1 Varianty algoritmu 23](#_Toc237241206)

[4.3 Možné využitie a obmedzenia 23](#_Toc237241207)

[5 Užívateľská dokumentácia 24](#_Toc237241208)

[5.1 Inštalácia a spustenie programu 24](#_Toc237241209)

[5.2 Popis programu 24](#_Toc237241210)

[5.3 Základné používanie programu 25](#_Toc237241211)

[5.4 Načítanie vstupných údajov 26](#_Toc237241212)

[5.5 Editovanie položiek liniek 27](#_Toc237241213)

[5.6 Zoznam staníc 28](#_Toc237241214)

[5.7 Editovanie položiek staníc 28](#_Toc237241215)

[5.8 Editovanie spojení 29](#_Toc237241216)

[5.8.1 Zmena položiek spojenia 29](#_Toc237241217)

[5.8.2 Zmena trasy spojenia 29](#_Toc237241218)

[5.9 Generovanie cestovných poriadkov 31](#_Toc237241219)

[5.9.1 Výber algoritmu 31](#_Toc237241220)

[5.9.2 Generovanie 31](#_Toc237241221)

[5.10 Prezentácia výsledných CP 32](#_Toc237241222)

[6 Implementácia 33](#_Toc237241223)

[6.1 Technológie 33](#_Toc237241224)

[6.2 Dátové štruktúry 33](#_Toc237241225)

[6.2.1 Triedy reprezentujúce hlavné entity 33](#_Toc237241226)

[6.2.2 Triedy reprezentujúce „cache“ 34](#_Toc237241227)

[6.2.3 Pomocné triedy 34](#_Toc237241228)

[6.3 Hľadanie najkratších vlakových spojení 34](#_Toc237241229)

[6.4 Navrhnuté Algoritmy 35](#_Toc237241230)

[6.4.1 Randomized with local search 35](#_Toc237241231)

[6.4.2 Discrete Set Algorithm 35](#_Toc237241232)

[6.4.3 Varianty algoritmu 35](#_Toc237241233)

[7 Analýza výsledkov 37](#_Toc237241234)

[7.1 Testovanie na vstupných údajoch 37](#_Toc237241235)

[7.2 Porovnanie algoritmov 37](#_Toc237241236)

[7.3 Porovnanie vygenerovaných CP 37](#_Toc237241237)

[8 Záver 38](#_Toc237241238)

[8.1 Nedostatky a možné rošírenia 38](#_Toc237241239)

[Literatúra 39](#_Toc237241240)

[Prílohy 39](#_Toc237241241)

# Úvod

## Motivácia

TODO: úvod, motivácia, prečo?

## 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, z ktorého sa načítajú potrebné údaje. Tie sa dajú modifikovať na viacerých úrovniach, čím sa do značnej miery dá ovplyvniť výsledok generovania. Výstupom programu sú viaceré cestovné poriadky, vygenerované rôznymi algoritmami pre porovnanie.

## Štruktúra práce

V druhej kapitole je načrtnutá problematika generovania cestovných poriadkov a popísané známe algoritmy, ktoré riešia daný problém. 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 vzájomne porovnané výsledky vygenerované navrhnutými algoritmami nad rôznymi kolekciami vstupných dát. Výsledky sú porovnané aj so skutočným cestovným poriadkom. Siedma kapitola je zameraná na implementáciu a  popisuje zaujímavosti, ktoré sa vyskytli pri implementácii. V závere sú zhrnuté výsledky porovnania algoritmov a kriticky spomenuté nedostatky.

# Problematika generovania cestovných poriadkov

V tejto kapitole a rovnako ako aj celej práci sa 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.

## 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.

TODO: ukážka taktového cestovného poriadku

## Obmedzujúce podmienky

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

V tejto sekcii popíšem ako sa dajú modelovať obmedzujúce podmienky pre taktové CP na železnici. Najskôr uvediem predpoklady a požiadavky vyplývajúce z rôznych faktorov, následne ukážem ako sa formulujú periodické podmienky na základe rôznych požiadaviek.

### Predpoklady

Predpoklady pre model môžu byť rôzne v závislosti od infraštruktúry, vlakových spojení a iných požiadaviek na CP. Spomenuté predpoklady a požiadavky z nich vyplývajúce sú vopred zadané.

#### Železničná infraštruktúra

Železničnú infraštruktúru tvoria uzly a trate.

* **Uzly** reprezentujú miesta v železničnej sieti, kde sa môžu vlaky vzájomne ovplyvňovať, preto sa v nich vyžaduje koordinácia. 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.

#### 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. pravidelné 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 a 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. Taká situácia nastane ak medzi dvomi stanicami neexistuje priame vlakové spojenie, čiže nutnosť prestupu.
* **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, v ktorej 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.

### Formulácia obmedzujúcich podmienok

V tejto sekcii si ukážeme ako sa dajú predchádzajúce požiadavky 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 (TODO:).

#### 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:

čas príchodu vlaku do uzla

čas odchodu vlaku z uzla

Kde parameter je interval cestovného poriadku v minútach. Rozhodujúce premenné a nadobúdajú celočíselné hodnoty z domény . Vlak podľa CP príde do stanice v čase :25, pobudne v stanici jednu minútu a odíde v čase :26, zapíšeme pomocou obmedzujúcej podmienky takto:

Inými slovami udalosť nastane jednu minútu pred udalosťou , a naopak udalosť nastane jednu minútu po .

Periodicita obmedzujúcich podmienok sa dosiahne počítaním v modulo T (pri perióde T). Pre zovšeobecnenie myšlienky uvažujme ako príchod vlaku do uzla , ako odchod iného vlaku z uzla , a ich vzťah vytvoríme pomocou časového okna  namiesto fixovanej hodnoty. Časové okno predstavuje interval s dolnou a hornou hranicou pre rozdiel dvoch rozhodujúcich premenných.

Ekvivalentne sa predchádzajúci zápis pre prehľadnosť skracuje na:

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

* **Pobyt vlaku na stanici** , ktorý má byť minimálne 2 minúty a maximálne 10 minút sa dá vyjadriť
* **Prípojové vlaky -** plánovaný vzájomný prestup medzi linkami a v stanici , sa definuje dvojicou obmedzujúcich podmienok. Požadovaný čas na prestup od 2 do 10 minúť medzi oboma navzájom.
* **Spájanie dvoch vlakov** a , respektíve spájanie ich vlakových súprav do jednej (napr. ) na spoločnej trase ohraničenej stanicami a si vyžaduje, aby boli v oboch hraničných staniciach obidva vlaky prítomné. Presnejšie v stanici , kde sa vlaky spájajú do , príchod vlaku musí predchádzať odchodu vlaku . Druhá obmedzujúca podmienka vyplýva analogicky pre stanicu, kde sa rozpájajú:
* **Otáčanie vlakov** v konečnej stanici pre použitie vlakovej súpravy na linke v opačnom smere vyjadruje obmedzujúca podmienka

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.

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

Dostávame upravené periodické obmedzujúce podmienky:

Každá obmedzujúca podmienka má dve rozhodujúce premenné. Ak nejaký cestovný poriadok spĺňa všetky obmedzujúce podmienky, tak pridaním 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 minút. Ak pri zvolenom posunutí sú všetky pomocné premenné , tak všetky obmedzujúce podmienky pre fixné odchody a príchody sú splniteľné.

* **Synchronizáciu dvoch vlakov**  a , ktoré majú značnú časť trasy spoločnú a frekvenciu jedna v perióde cestovného poriadku, skonštruujeme nasledovne: chceme posunúť odchod vlaku voči odchodu vlaku o 30 minút s chybou 2 minúty. Synchronizácia sa vzťahuje na celú spoločnú trasu so stanicami .

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 opustí stanicu v určitom čase, tak vlak nesmie opustiť stanicu do 3 minút pred ani po odchode vlaku . Keďže bezpečnostný rozstup sa týka vlakov po celej dĺžke trate medzi stanicami a , tak dostávame dve podmienky:

## Periodic Event Scheduling Problem

Táto sekcia opisuje Periodic Event Scheduling Problem (PESP), ktorý pôvodne sformulovali Serafini a Ukovich v roku 1989 (1). 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 .

––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––

Definícia PESP

––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––

Nech je daná množina udalostí, množina , časová perióda , a časové okná pre všetky . PESP má nájsť periodický rozvrh , ktorý spĺňa

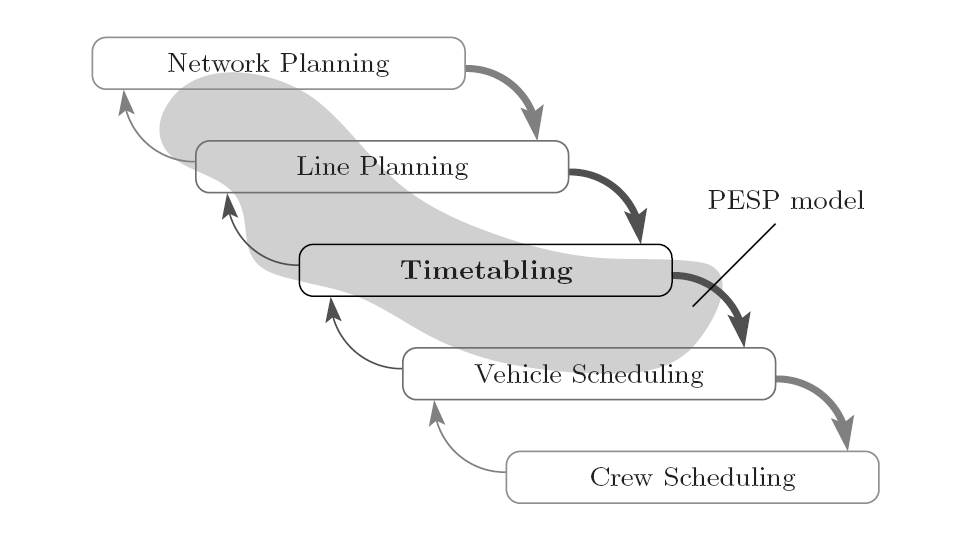
alebo rozhodne, že vstupné podmienky sú nesplniteľné a rozvrh neexistuje.

––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––

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

## 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 : Fázy plánovania. Zdroj:

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 plánovacie procesu. Problematiku rozšírenia rozoberajú Liebchen a Möhring v . 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.

## 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

TODO: cite

Voorhoeve (1993), 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 (1993, 1994) 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 (1996) 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).

Hurkens (1996) uviedol algoritmus založený na technike *branch and cut* na riešenie problému PESP.

Hassin (1996) 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 (1994, 1996) 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 (1996) 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 (2000) zobral PESP ako súčasť problému, ktorý konštruuje CP optimalizovaný voči nákladom na vozový park.

## Existujúce systémy

CADANS

TODO:dpln systemy a linky

## Prípadová štúdia

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

### 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 aspoň 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]](#footnote-2) MIP-Solverom. Skonštruovali CP, ktorý podstatne vylepšil ten predchádzajúci vo všetkých zadaných kritériách.

#### 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]](#footnote-3) 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 význačne lepší ako manuálne skonštruovaný v splniteľnosti kritérií. Prípad demonštroval úspešné uvedenie prvého, matematicky optimalizovaného, taktového CP v praxi.

# 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.

## Vstup

Vstupom je existujúci cestovný poriadok získaný z IDOSu[[3]](#footnote-4), ktorý slúži pre potreby načítania vstupných dát. 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.

Ďalším parametrom vstupu je zoznam staníc. Každú stanicu špecifikuje názov stanice, mesto 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.

Parametrami vstupu aj zoznam navzájom prepojených liniek. Vzájomne prepojené linky tvoria triedu ekvivalencie. Ak posuniem odchod linky o  minút, posunú sa rovnako o minút odchody všetkých liniek, ktoré sú s ňou prepojené.

## Modifikácia vstupu

Sekcia opisuje, že predmetom vstupných údajov sú informácie o linkách, staniciach a o prepojených linkách. Pre samotný proces generovania je nutné vytvoriť aj súhrnné informácie o prestupoch.

Najskôr sa vygenerujú vlakové spojenia, ktorým sa pomocou Newtonovho gravitačného zákona vypočíta počet očakávaných cestujúcich medzi dvomi stanicami. Pre každý špecifický prestup (linka, linka, stanica) sa sčíta hodnota cestujúcich z príslušných prestupov obsiahnutých vo vlakových spojeniach.

### Viacúrovňové zadávanie údajov

Procesu optimalizácie predchádza možnosť viac úrovňového 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.

## Obmedzujúce podmienky

Keďže kompletné časovanie vlakových liniek je zadané priamo na vstupe, riešený problém sa zužuje len na použitie obmedzujúcich podmienok pre prípojové vlaky zo sekcie .

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:

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  minút neskôr, rovnako o minút sa mi zmení aj čas odchodu druhého vlaku z jeho počiatočnej stanice. Navzájom previazané linky tvoria triedy ekvivalencie.

## 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.

## Minimalizačná funkcia

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

je množina prestupov v CP

je prestup medzi dvomi linkami v stanici , z linky na linku

je čas príchodu linky do prestupnej stanice

je čas odchodu linky z prestupnej stanice

je počet očakávaných cestujúcich na prestupe z linky na linku v stanici

je počet vlakových liniek

je množina všetkých staníc

je prestupná stanica pre linky a

## Generovanie a optimalizácia

Výsledkom generovania a konštrukcie CP je nájsť vzájomné posunutie zadaných vlakových liniek v čase tak, aby výsledný CP zohľadňoval problém optimalizácie. T.j. minimalizovať časy na prestupoch tak, aby bolo čo najviac cestujúcich spokojných. Minimalizačná funkcia je definovaná v sekcii .

# Navrhnuté algoritmy

## Náhodné generovanie s vylepšovaním

Prvý z navrhnutých a implementovaných algoritmov je založený na vygenerovaní náhodného CP a jeho modifikáciou sa ho postupe snaží vylepšovať. Vylepšovanie využíva 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*

T := createRandomizedTimetable(AllLines)

AvailableLines := AllLines

StableLines := 0

- *main loop*

**while** AvailableLines 0 **do**

improved := false

selectedLine := chooseRandomlyFrom(AvailableLines)

**for** all possible shifts 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.

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 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 vybranú linky, prípadne 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.

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 . Kde je počet liniek a 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 iterácií cyklu for. Kde je počet liniek, je perióda linky a je hodnota minimalizačnej funkcie počiatočného riešenia.

## Generovanie s použitím „discrete set“

Pri návrhu tohto algoritmu sme vychádzali z propagačného algoritmu podmienok navrhnutým v práci od Voorhoeve (TODO: ref). Algoritmus je založený propagácií obmedzujúcich podmienok.

Podmienka sa vzťahuje na dve udalosti (príchod alebo odchod vlaku), ktoré sú obmedzené určitou diskrétnou množinu, 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:

Propagation part – propagačná časť algoritmu

Search part – časť algoritmu vyhľadávajúca riešenie

Propagation method – spoločná metóda pre obe časti

#### 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] :=

**else**

matrix[i,j] :=

**fi**

**od**

**foreach** constraint on i,j *I* **do**

matrix[i,j] := constraint.set

matrix[j,i] := – 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 na linku v stanici . Ak minimálny čas na prestup sú 3 minúty a maximálne 10 minút, podmienku označíme a bued mať tvar:

Kde:

– je odchod linky z jej počiatočnej stanice

– je príchod linky do prestupnej stanice

– je odchod linky z jej počiatočnej stanice

– je odchod linky z prestupnej stanice

– 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:

Kde je počet očakávaných pasažierov. Mapovanie je nutné kvôli následnému upravovaniu podmienok a tým aj diskrétnej množiny. Podmienky sú zlučované nasledujúcim spôsobom. Ak existuje podmienka alebo , nová výsledná podmienka bude mať diskrétnu množinu :

Môže sa stať, že množina nebude mať súvislý jeden beh, ale viacero. Preto dôkladne diskrétnu množinu nahradíme množinou . 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 diskrétnych množín , kde štandardne sú na diagonálne inicializované množiny a inde . Inak sú pre každú podmienku na do matice pozíciu dosadená diskrétna množina , čiže . Nakoniec v propagačná časti je zavolaná propagačná metóda.

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

#### 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** ki,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] := – matrix[i,j]

changed := true

**fi**

**od**

**od**

––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––

#### Popis propagačnej metódy

Ak nám po propagácii vznikne nikde v matici prázdna mnosžina, je rozpropagovaná do celej matice. V tomto prípade môže povedať, že vstupné obmedzujúce podmienky boli príliž striktné, nie sú splniteľné a neexistuje pre ne riešenie.

––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––

Definícia stabilnej matice

––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––

Matica M je stabilná práva vtedy, keď platí:

––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––

Keďže ide o stabilnú maticu, stačilo by sa pri iterácií cyklu for obmedziť na podmienku *i j*. My sme použili podmienku *i j* preto, aby sa propagovanie prázdnej množiny dostalo aj na diagonálu. Môžeme sa obmedziť aj na *ki,j*. Ak *i = j*, tak je určite splnené, a ak *i =k*, tak je taktiež splnené.

Sčítanie diskrétnych množín je definované ako .

#### 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**

––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––

TODO: este nie je doimplementovane

––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––

**Construction of Timetables**

––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––

*- parameters*

matrix : matrix of discrete set

*- solution*

matrix[0] represents solution

––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––

#### Pseudokód algoritmu

#### Konečnosť algoritmu

#### Zložitosť algoritmu

### Varianty algoritmu

Druhý navrhnutý algoritmus má viaceré varianty, ktoré vznikli až vo fáze implementácie. Boli navrhnuté rôzne postupy, ktoré pozmenili hlavné časti algorimtu propagate a search part.

#### Variant A

Bisection algorithm with same max transfer time

Deterministic searcher

#### Variant B

Bisection algorithm with alpha\*T max transfer time

Deterministic searcher

#### Variant C

Propagation with full discrete set

Deterministic

#### Variant D

Propagation with full discrete set

Probabilistic

## Možné využitie a obmedzenia

# 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.

## Inštalácia a spustenie programu

Program PTG je určený pre platformu Windows (verzie win ??). 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 **PeriodicTimetableGeneration.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.

## 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.

#### 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  minút, rovnako sa posunú 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 , kde 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).

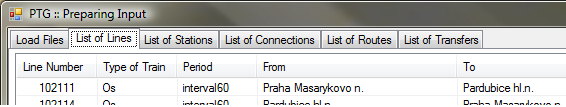
## Základné používanie programu

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.

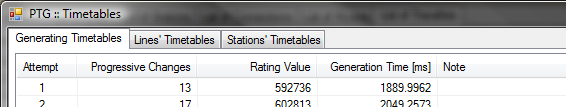
Do ďalšieho tabu sa úžívateľ dostane len stlačením tlačidla **Next**, ktoré sa nachádza v každom tabe okrem posledného. Pri prechádzaní tabmi v opačnom smere si môže užívateľ kliknúť na ľubovoľný tab,  ktorý už navštívil: T.j. príslušná už fáza prebehla a tab je sprístupnený užívateľovi.

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, a následne sa zobrazí druhé okno **PTG :: Timetables**. Taby oboch okien sú zobrazené na a o.



Obrázok : Taby v okne Preparing Input.



Obrázok : Taby v okne Timetables.

## 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

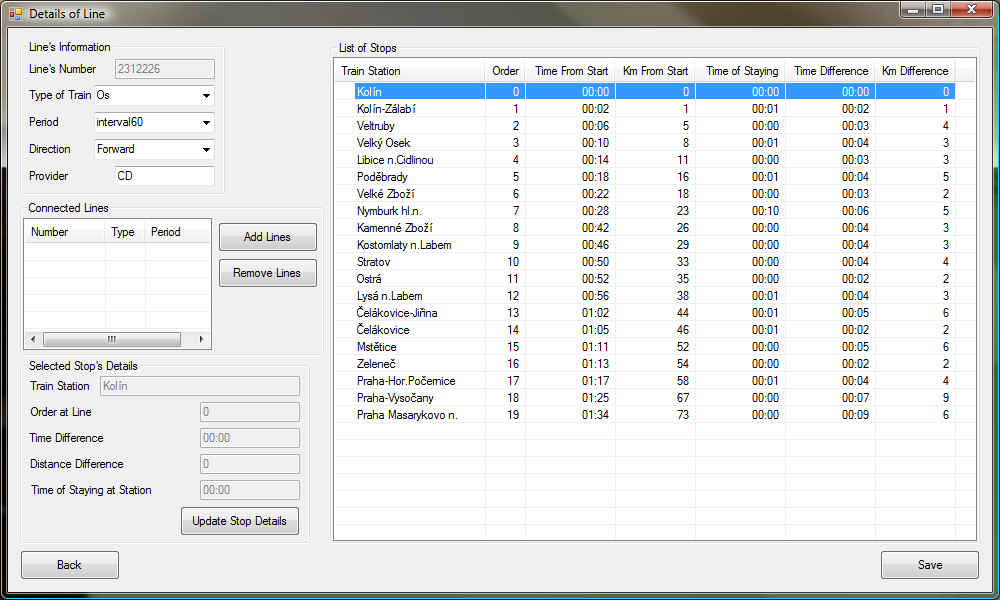
––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––

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

## 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 je screenshot okna detailu vlakovej linky **Details of Line**, ktoré sa následne zobrazí. Editovateľné sú položky: **Type of Train**, **Period**, **Direction** a **Provider.**

TODO: update pic



Obrázok : Detaily vlakovej linky

#### Previazané linky

Modifikovať zoznam previazaných liniek pre určitú linku sa dá v okne **Details of Line** (pre ilustráciu zobrazené na ). 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**.

## 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.

## 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.

## Editovanie spojení

### Zmena položiek spojenia

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

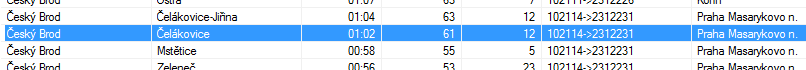
Napríklad dané spojenie je vyťaženejšie (čo do počtu očakávaných cestujúcich) v dôsledku infraštruktúry lokality staníc, ktoré sa nachádzajú v industriálnej, obchodnej alebo obytnej zóne so zvýšením dopytom po osobnej preprave.

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**.

### 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 . 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.



Obrázok : Pôvodne vygenerované spojenie z Českého Brodu do Čelákovic

Pre lepšie predstavenie si danej situácie uvádzam ilustračný príklad.

––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––

*Príklad:*

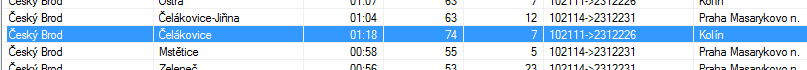
––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––

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 .

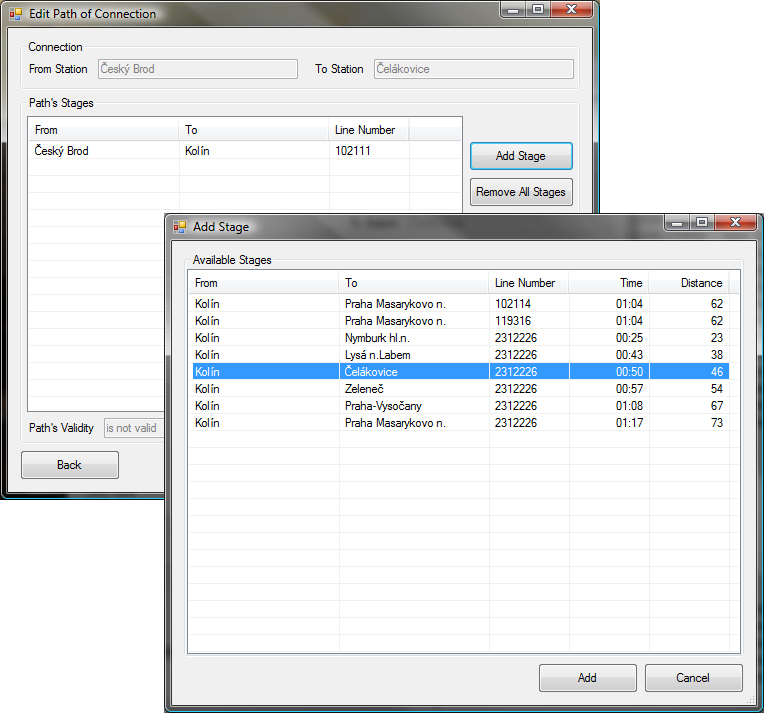
Užívateľ sa rozhodol , že chce trasu spojenia modifikovať použijúc spojenie linkami *102111* a*231226* s prestupom v stanici *Kolín*. Situácia na .

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*.

––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––



Obrázok : Upravená trasa spojenia z Českého Brodu do Čelákovic



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

## Generovanie cestovných poriadkov

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

### Výber algoritmu

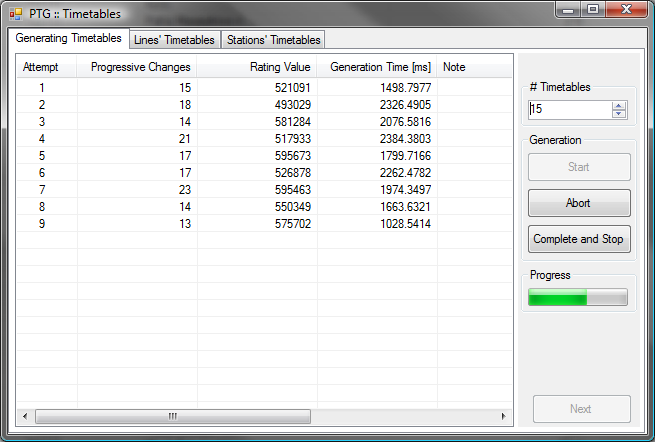
Užívateľ má na výber z viacerých implementovaných algoritmov, ktoré si volí kliknútím na príslušné tlačidlo algoritmu v okne **PTG :: Preparing Input** na tabe **List of Transfers** v skupine tlačidiel **Generation Algorithms**. Následne sa otvorí okno **PTG :: Timetables**. Ďalší postup sa už líši v závislosti od zvoleného algoritmu.

*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.

### 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 . Pre každý záznam predstavujúci CP sú uvedené informácie z procesu generovania.



Obrázok : 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 . 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.

## Prezentácia výsledných CP

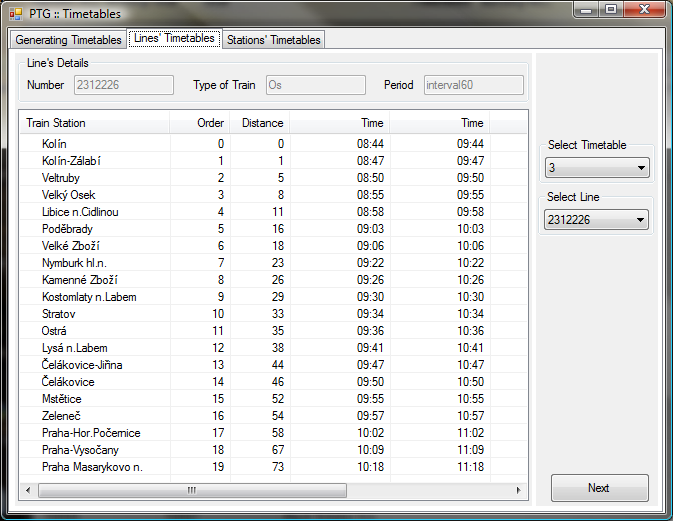
Vygenerované taktové CP sú pre potreby zobrazenia konkrétne vypočítané od istej hodiny dňa. Obecne je stanovená na 08:00. Štruktúra taktových CP je pre každú periódu rovnaká. Preto je rozpočítanie časovania od istej konkrétnej hodiny pre jednotlivé linky urobené pre komfortnejšie porovnanie výsledkov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

#### 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 robaľovacej ponuky **Select Station**. Zobrazenie



Obrázok : Prezentácia linkového CP.

# Implementácia

## Technológie

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

## 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 Trains.sln, ktorý je priložený na DVD nosiči. Rovnako na nosiči sa v priečinku (TODO: kde?) nachádzajú diagramy tried Class Diagrams a detailná dokumentácia projektu vygenerovaná programom Doxygen (TODO:).

### Triedy reprezentujúce hlavné entity

V tejto sekcii sú popísané hlavnými 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.

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.

### 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. „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

### 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.

## 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

## 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 .

IGenerationAlgorithm

* metóda generateTimetables pre spustenie generovania CP
* vlastnosť Timetables vracajúca zoznam CP
* udalosť OnProgressChanged signalizujúca priebeh algoritmu

### Randomized with local search

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

GenerationAlgorithmRandomized : IGenerationAlgorithm

### 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 diskretných množín na základe vstupných obmedzujúcich podmienok Constraint

Algoritmus je založený na splňovaný podmienok:

Constraint

* je abstratkná trieda
* obsahuje diskretnú množinu Set, špecifickú pre túto podmienku

TransferConstraint : Constraint

* je vytvorený z konkrétneho prestupu Transfer

ConnectedLineConstraint : Constraint

Set

### 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 spúsobom, ktory tiež predstavuje odlišnosť medzi týmito variantmi. K tomuto cieľu bolo vytvorené rozhranie:

IConstraintSetCreator

**BisectionPropagator : IConstraintPropagator**

**SimplePropagator : IConstraintPropagator**

**SameTransferTime : IConstraintSetsCreator**

**AlfaTTransferTime : IConstraintSetsCreator**

**FullDiscreteSet : IConstraintSetsCreator**

V druhej časti algoritmu, search, 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

DeterministicSearcher : IBestChoiceSearcher

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

ProbableSearcher : IBestChoiceSearcher

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

# Analýza výsledkov

## Testovanie na vstupných údajoch

### Sieť malá

### Sieť stredná

### Sieť velká

### Pražské metro

<http://www.dpp.cz/dp-v-datech/>

## Porovnanie algoritmov

## Porovnanie vygenerovaných CP

# Záver

## 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 nedobehol do konca.

# Literatúra

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

2. Project B15 - Service Design in Public Transport. [Online] http://www.zib.de/Optimization/Projects/TrafficLogistic/Matheon-B15/Matheon-B15long2.en.html.

3. **Leibchen, Christian a Mohring, Rolf H.** *A case Study in Periodic Timetabling.*

# Prílohy

Literatúra

[1] asdasdasd

<http://www.thushanfernando.com/index.php/2008/06/27/design-patterns-in-c-java-the-singleton/>

1. http://www.cplex.com [↑](#footnote-ref-2)
2. Spoločnosť prevádzkujúca metro v Berlíne, http://www.bvg.de [↑](#footnote-ref-3)
3. http//www.idos.cz [↑](#footnote-ref-4)