ŽILINSKÁ UNIVERZITA V ŽILINE
FAKULTA RIADENIA A INFORMATIKY

VEHICLE ASSIGNMENT PROBLEM A PROGRAMOVANIE S OBMEDZUJÚCIMI PODMIENKAMI

Diplomová práca

24/2012

Študijný program: Aplikovaná informatika

Pracovisko (katedra/ústav): Katedra dopravných sietí, Fakulta riadenia a informatiky

Vedúci záverečnej práce/školiteľ: doc. Ing. Ľudmila Jánošíková, PhD.

Žilina 2013 **Bc. Miroslav Mačura**

Abstrakt

Vehicle Assignment Problem je časť úlohy evakuácie ľudí z ohrozeného územia a zaoberá sa rozhodnutím o tom, koľko vozidiel z jednotlivých dopravných parkov sa zúčastní na evakuácii každej obce. V práci sa zaoberáme týmto problémom, jeho matematickým modelom a spôsobmi riešenia, a to hlavne pomocou metódy Constraint Programming. Pozrieme sa na túto metódu bližšie, na jej výhody a nevýhody a na programovacie jazyky založené na tejto metóde, špeciálne jazyk MiniZinc. Cieľom tejto práce je navrhnutie algoritmu pre Vehicle Assignment Problem založeným na metóde Constraint Programming, porovnanie s algoritmom založeným na metóde vetiev a hraníc. Následne vyhodnotíme tieto metódy na základe testovacích príkladov a experimentov. Ďalším cieľom práce je zakomponovanie algoritmu založeným na metóde Constraint Programming do existujúceho softvérového nástroja na podporu rozhodovania.

Abstract

Vehicle Assignment Problem is part of the problem of people evacuation from endangered area and deals with decision about how many vehicles from various transport parks will take part on evacuation of every community. In this thesis we deal with this problem, its mathematical model and solving methods, mainly by means of Constraint Programming method. We look closer to this method, to its advantages and disadvantages and to programming languages based on this method, especially language MiniZinc. The aim of this thesis is designing algorithm for Vehicle Assignment Problem based on Constraint Programming method, comparing with algorithm based on branch and bounds method. After that we evaluate these methods on the basis of test examples and experiments. Next aim of this thesis is integration of algorithm based on Constraint Programming method into the existing decision support tool software.

Prehlásenie:

Prehlasujem, že som diplomovú prácu spracoval samostatne pod odborným vedením vedúceho záverečnej práce a že som uviedol všetky použité pramene a literatúru, z ktorých som čerpal.

V Žiline, dňa 1. mája 2013

Podpis: .....

Poďakovanie

Úprimne ďakujem vedúcej diplomovej práce doc. Ing. Ľudmile Jánošíkovej, PhD. za jej odbornú pomoc, cenné pripomienky a usmernenia, ktoré mi poskytla pri vypracovaní záverečnej práce.

Moje poďakovanie patrí taktiež Ing. Ľubomírovi Tomanovi za jeho softvérový nástroj na podporu rozhodovania, pomoc a výsledky algoritmu založenom na metóde vetiev a hraníc.

OBSAH

ÚVOD

Evakuácia je dnes často používaným pojmom a to tým viac, čím sa zvyšuje výskyt prírodných alebo ľudsky zapríčinených katastrof celosvetovo, alebo len na území Slovenskej republiky. Jedná sa o javy ako záplavy, zosuvy pôdy či priemyselná havária. Pod úlohou evakuácie rozumieme presun obyvateľstva, zvierat prípadne vecí z ohrozeného územia. Na úlohu evakuácie sa môžeme pozerať z viacerých uhlov pohľadu. Mikroskopický pohľad je zameraný na evakuáciu ľudí z budov (pracoviská, obchodné centrá, paneláky, domy atď.), zatiaľ čo makroskopický pohľad sa zameriava na evakuáciu ľudí z ohrozeného, geograficky rozľahlého územia. Úloha evakuácie ľudí z ohrozeného územia, ktorej sa budeme venovať v tejto práci, je typická obmedzeným množstvom dopravných prostriedkov, ktoré môžeme použiť na presun ľudí z ohrozeného územia do útočísk, ktoré predstavujú bezpečné miesta. Dopravné prostriedky majú obmedzenú kapacitu danú počtom osôb, ktoré môžu súčasne prevážať. Útočiská majú taktiež kapacitné obmedzenia čiže počet ľudí, ktoré dokážu prijať. Jednou z podúloh tejto úlohy, ktorá ale patrí medzi jej kľúčové časti z hľadiska operačného výskumu, je úloha prideľovania dopravných prostriedkov, ktorej cieľom je vhodne prideliť resp. prerozdeliť obmedzený počet dopravných prostriedkov, ktoré sú k dispozícii v dopravných parkoch, k jednotlivým ohrozeným miestam (najčastejšie obciam), v ktorých sa nachádzajú ľudia tak, aby boli všetci ľudia zo všetkých ohrozených miest evakuovaní za minimálny čas na bezpečné miesta.

V práci sa bližšie zaoberáme touto podúlohou evakuačnej úlohy, presnejšie využitím programovania s obmedzujúcimi podmienkami na jej riešenie. Programovanie s obmedzujúcimi podmienkami nie je veľmi známy spôsob programovania. V práci skúmame možnosti jeho použitia pre danú úlohu, jeho efektívnosť a vypočítané riešenie porovnávame s riešením získaním pomocou matematického programovania. Pozrieme sa na výhody tejto metódy a na programy, ktoré sa zaoberajú programovaním s obmedzujúcimi podmienkami. Na záver si porovnáme získané výsledky pomocou experimentov na reálnej sieti.

1 POPIS PROBLÉMU EVAKUÁCIE

Popis problému evakuácie čerpáme z práce Ľ. Tomana [1].

Úloha evakuácie ľudí z ohrozeného územia alebo skrátene len úloha evakuácie patrí do oblasti operatívneho riadenia verejných obslužných systémov. Cieľom úlohy evakuácie je v prípade výskytu alebo hrozby mimoriadnej udalosti evakuovať resp. presunúť ľudí nachádzajúcich sa v ohrozených miestach na bezpečné miesta za minimálny čas, ktorý zohráva pri jej riešení dôležitú úlohu. Či už je to čas odozvy na vzniknutú resp. vznikajúcu situáciu, ktorý súvisí s časom potrebným na jej výpočet, alebo čas potrebný na realizáciu samotného rozhodnutia, ktorý je priamo ovplyvnený kvalitou získaného riešenia, v každom prípade treba konať rýchlo, aby mohla byť krízová situácia efektívne zvládnutá. Výsledkom riešenia tejto úlohy je vypracovanie návrhu evakuačného plánu, podľa ktorého môže byť vykonaná evakuácia resp. ktorý môže podporiť rozhodnutia týkajúce sa samotného priebehu evakuácie.

V úlohe evakuácie vystupuje množina ohrozených obcí, ktoré sa nachádzajú v území ohrozenom nejakou mimoriadnou udalosťou, napr. prírodná katastrofa (záplavy, zosuvy pôdy atď.), priemyselná havária (požiar v chemickej továrni s následným únikom nebezpečnej chemikálie), vojnový konflikt a iné. V každej ohrozenej obci sa nachádza určitý počet obyvateľov, ktorých treba evakuovať na bezpečné miesta nazývané útočiská. Na prevoz ľudí z obcí do útočísk sú použité vozidlá, ktoré sú k dispozícii v dopravných parkoch, pričom v každom takomto parku sa môže nachádzať rôzny počet vozidiel s rôznou kapacitou, ktorá je daná počtom ľudí, ktorí môžu byť vo vozidle súčasne prepravovaní. Cieľom úlohy evakuácie je použiť jednotlivé vozidlá tak, aby bol celkový čas evakuácie minimálny. Za tento celkový čas evakuácie považujeme časový interval, ktorý začína momentom vyslania vozidiel z dopravných parkov a ktorý končí v momente, keď je posledný obyvateľ z ohrozených obcí dopravený do niektorého z útočísk.

Načrtnutá úloha evakuácie pozostáva z viacerých častí. Prvá z nich spočíva vo vypracovaní predbežných plánov, ktoré obsahujú vytipované útočiská, dopravné parky a ohrozené obce pre rôzne prípady krízových situácií. V prípade pretrhnutia konkrétnej hrádze je možné do určitej miery určiť, ktoré obce nachádzajúce sa v blízkosti hrádze budú ohrozené, ktoré nebudú ohrozené, a teda môžu slúžiť ako útočiská a v ktorých miestach v blízkom okolí sa môžu nachádzať dopravné prostriedky použiteľné na evakuáciu. Riešenie resp. spracovanie tejto časti spadá do kompetencie pracovníkov krízového

manažmentu, ktorých úlohou je príprava takýchto predbežných plánov tvoriacich vstupné údaje pre riešenie ostatných častí úlohy evakuácie.

Ďalšou časťou evakuačnej úlohy je rozdelenie ohrozených obcí na menšie časti (čo sa týka počtu obyvateľov), ktoré možno evakuovať nezávisle od ostatných častí obce (napríklad vozidlami z iného dopravného parku alebo do iného útočiska). Tieto časti obcí budeme nazývať *komunity*. Pre každú komunitu určíme útočisko, do ktorého budú jej obyvatelia evakuovaní tak, aby sa neprekročili kapacity týchto útočísk. Kapacita útočiska je daná počtom ľudí, ktorých môže dané útočisko dočasne prichýliť na nevyhnutnú dobu do pominutia dôvodu, pre ktorý bola vykonaná evakuácia. Rozdelenie obcí na komunity je nevyhnutné preto, lebo by sa mohlo stať, že je potrebné evakuovať obec s veľkým počtom obyvateľov, pričom ani jedno z útočísk nemá dostatočnú kapacitu, ktorá by pokryla celú požiadavku obce.

Rozdeľovanie ohrozených obcí na komunity sa môže riešiť viacerými spôsobmi. Napríklad ako vybilancovanú dopravnú úlohu (s prípadným vytvorením fiktívnej ohrozenej obce v prípade, že je suma obyvateľov obcí menšia ako suma kapacít útočísk), v ktorej sú počty obyvateľov ohrozených obcí brané ako kapacity skladov a kapacity útočísk brané ako požiadavky odberateľov. Iná možnosť je riešiť túto úlohu ako dopravnú úlohu (ale nie nutne vybilancovanú), v ktorej sú na rozdiel od predchádzajúceho prípadu počty obyvateľov ohrozených obcí brané ako požiadavky odberateľov a kapacity útočísk brané ako kapacity skladov. Je možné použiť rôzne modifikácie modelu popisujúceho úlohu s cieľom eliminovať vznik komunít s veľmi malým počtom obyvateľov, menším než je priemerná kapacita dopravných prostriedkov použitých na evakuáciu.

Ako bolo uvedené, úlohu rozdeľovania ohrozených obcí na počtom obyvateľov menšie komunity je možné previesť na klasickú dopravnú úlohu, ktorú nezaraďujeme medzi NP ťažké úlohy vďaka vlastnosti celočíselnosti. Táto úloha teda nie je veľmi zaujímavá z hľadiska operačného výskumu, avšak jej riešenie nemôžeme podceniť ani zanedbať, nakoľko jej výsledok vytvára východiskovú pozíciu pre riešenie záverečnej fázy úlohy evakuácie.

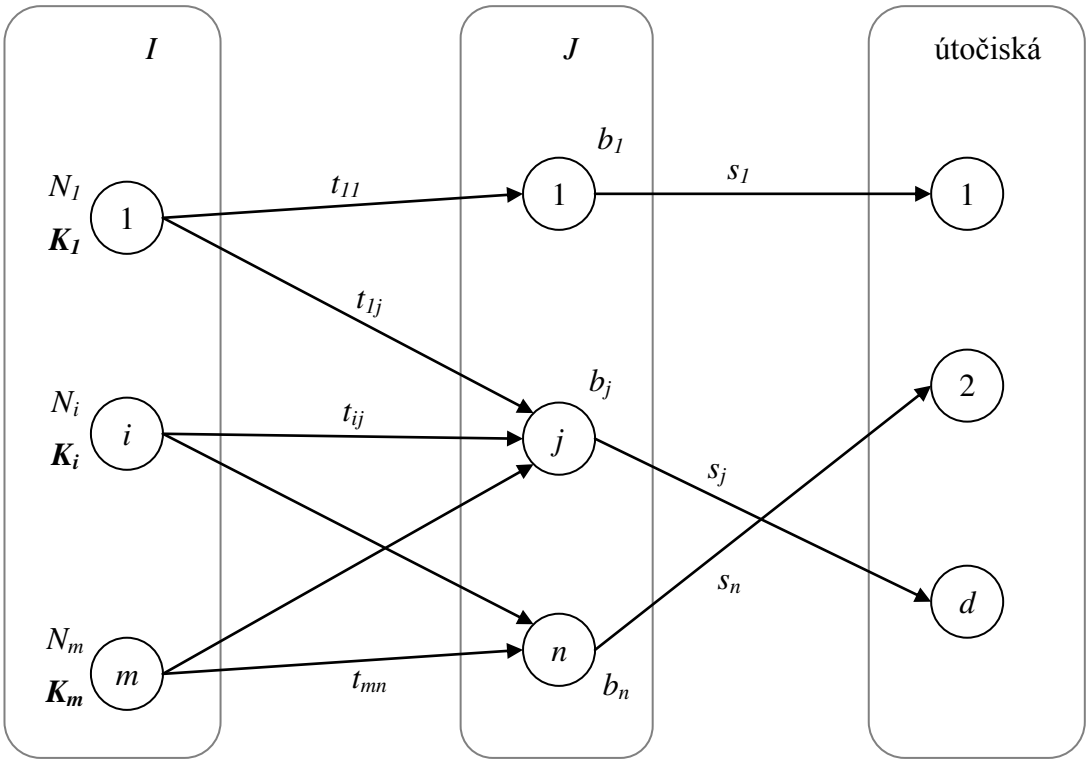
Poslednou časťou úlohy evakuácie je úloha prideľovania dopravných prostriedkov alebo VAP (Vehicle Assignment Problem). VAP spočíva v stanovení trasy pre každé vozidlo použité na evakuáciu. Úloha patrí medzi NP ťažké a predstavuje zložitú kombinatorickú úlohu. Určením týchto trás pre jednotlivé vozidlá je skompletizovaný evakuačný plán, nakoľko ten potom obsahuje množinu komunít, ktorá vznikla delením ohrozených obcí na počtom obyvateľov menšie, samostatne evakuovateľné celky a tiež

jednoznačné pridelenie komunít k útočiskám s ohľadom na kapacity útočísk, čím je jednoznačne určené, kam budú obyvatelia jednotlivých komunít evakuovaní. Evakuačný plán ďalej obsahuje umiestnenia jednotlivých vozidiel na začiatku evakuácie ako aj stanovené trasy pre tieto vozidlá, ktorých dodržanie pri realizácii samotného presunu obyvateľov garantuje určitú kvalitu evakuácie danú celkovým časom potrebným na jej vykonanie.

1.1 Formulácia úlohy VAP

V tejto kapitole definujeme úlohu prideľovania dopravných prostriedkov.

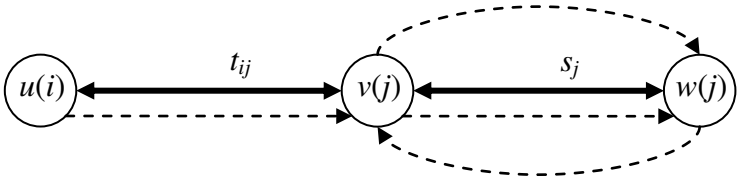
Na území ohrozenom nejakou mimoriadnou udalosťou sa nachádza množina komunít J. V každej komunite $j \in J$, ktorá leží v grafe modelujúcom cestnú sieť vo vrchole $v(j)$, sa nachádza b_j obyvateľov, ktorých treba evakuovať. Každá komunita $j \in J$ je priradená práve jednému útočisku, do ktorého budú evakuovaní jej obyvatelia. Toto útočisko, ku ktorému je priradená komunita j, sa nachádza v grafe vo vrchole $w(j)$. K jednému útočisku môže byť priradených aj viacej komunít. Svojou kumulovanou požiadavkou, t.j. súčtom počtov svojich obyvateľov však nesmú prekročiť kapacitu daného útočiska. Ďalej sa v cestnej sieti nachádza množina homogénnych dopravných parkov I. Homogénny dopravný park je dopravný park, ktorý má k dispozícii určitý počet vozidiel, pričom všetky tieto vozidlá majú rovnakú kapacitu. Homogénny park $i \in I$, ktorý sa nachádza vo vrchole $u(i)$, má k dispozícii N_i vozidiel s rovnakou kapacitou K_i. Kapacita je daná počtom obyvateľov, ktorí môžu byť vo vozidle prepravovaní súčasne. Pre účely riešenia úlohy sú nehomogénne dopravné parky, v ktorých sa nachádzajú vozidlá s rôznymi kapacitami, rozdelené na viac homogénnych dopravných parkov. Symbolom t_{ij} označíme čas potrebný na presun vozidla z parku i do komunity j, teda na presun z vrcholu $u(i)$ do vrcholu $v(j)$. Symbol s_j označuje čas potrebný na presun vozidla medzi komunitou j a útočiskom, ku ktorému je komunita pridelená, t.j. na presun medzi vrcholmi $v(j)$ a $w(j)$. Uvedená situácia je schematicky znázornená na obr. 1.



Zdroj: [1]

Obr. 1. Úloha prideľovania dopravných prostriedkov

Ak je vozidlo z parku *i* pridelené na evakuáciu komunity $j \in J$, potom už nie je použité na evakuáciu žiadnej inej komunity $j' \in J$, pre $j \neq j'$. Cieľom úlohy prideľovania dopravných prostriedkov je rozhodnúť o trase každého vozidla, ktoré bude použité na evakuáciu. Vzhľadom na predchádzajúcu požiadavku zjednodušujúcu úlohu, ktorá umožňuje prideliť vozidlo najviac jednej komunite, bude trasa vozidla, ktorú vozidlo vykoná počas evakuácie, po pridelení komunite jednoznačne daná. Táto trasa znázornená na obr. 2 začína v dopravnom parku *i*. Odtiaľ vozidlo prejde do komunity *j*, kde vezme časť obyvateľov a potom ich dopraví do útočiska, ku ktorému je komunita pridelená. Vozidlo môže navštíviť komunitu aj viackrát opakovaným prejazdom medzi ňou a útočiskom, a tak evakuovať väčší počet jej obyvateľov. Počet návštev však musí byť najviac taký, aby celková doba jazdy vozidla nepresiahla celkovú dobu evakuácie.



Zdroj: [1]

Obr. 2. Trasa vozidla s dvoma návštevami v komunite j

Úloha prideľovania dopravných prostriedkov rozhoduje o tom, koľko vozidiel z každého parku $i \in I$ pôjde do jednotlivých komunít $j \in J$ tak, aby bol celkový čas potrebný na vykonanie evakuácie minimálny. Čas evakuácie meriame od momentu vyslania vozidiel z dopravných parkov do chvíle, keď je aj posledný obyvateľ presunutý do útočiska. Aj napriek predchádzajúcemu zjednodušeniu predstavuje úloha prideľovania dopravných prostriedkov zložitú kombinatorickú úlohu, avšak jej riešenie musíme nájsť v krátkom čase, lebo ide o krízovú situáciu.

1.2 Matematický model úlohy prideľovania dopravných prostriedkov

Uvedieme si matematický model úlohy prideľovanie dopravných prostriedkov, v ktorom y je čas evakuácie, ktorý minimalizujeme (6). Koeficienty $P_{ij}(y)$ pre $i \in I$ a $j \in J$ udávajú maximálny počet návštev vozidla z parku i do komunity j do času y a sú vyjadrené v podmienkach (1). Potom evakuačná kapacita $a_{ij}(y)$ je kapacita, ktorú je jedno vozidlo z parku i schopné poskytnúť komunite j počas evakuácie, ktorej priebeh trvá čas y. Môžeme ich vyjadriť pomocou podmienok (2).

Ak označíme x_{ij} ako rozhodovaciu premennú, ktorá bude predstavovať počet vozidiel priradených z parku i komunite j, potom podmienky (3) zabezpečujú, aby nebolo zo žiadneho homogénneho dopravného parku vyslaných viac vozidiel, než má park k dispozícii. Podmienky (4) zabezpečia, že priradenie vozidiel z parkov ku komunitám bude také, aby boli evakuovaní všetci obyvatelia všetkých komunít. Obligatórne podmienky (5) vyžadujú celočíselnosť premenných x_{ij}, nakoľko počet vozidiel bude malé celé číslo (rádovo jednotky).

$$P_{ij}(y) = \left\lfloor \frac{y - t_{ij} + s_j}{2s_j} \right\rfloor \qquad pre\ i \in I,\ j \in J \tag{1}$$

$$a_{ij}(y) = K_i P_{ij}(y) \qquad pre\ i \in I,\ j \in J \tag{2}$$

$$\sum_{j \in J} x_{ij} \leq N_i \qquad pre\ i \in I \tag{3}$$

$$\sum_{i \in I} a_{ij} x_{ij} \geq b_j \qquad pre\ j \in J \tag{4}$$

$$x_{ij} \in Z_0^+ \qquad pre\ i \in I,\ j \in J \tag{5}$$

$$\mathrm{f} = \min y \tag{6}$$

Pretože evakuačná kapacita a_{ij} je nelineárnou funkciu času y, úlohu VAP klasifikujeme ako nelineárnu úlohu zmiešaného matematického programovania. Takúto úlohu nevieme riešiť exaktne, preto si ukážeme metódu ako riešiť daný problém, a to pomocou iteratívneho postupu.

1.3 Iteratívny postup

Pri použití tohto postupu transformujeme komplexnú úlohu prideľovania dopravných prostriedkov, pri riešení ktorej by sme sa snažili prideliť vhodné množstvo vozidiel z parkov $i \in I$ na evakuáciu komunít $j \in J$ tak, aby bol celkový čas evakuácie minimálny, na časovo obmedzenú redukovanú úlohu, v ktorej sa snažíme vykonať evakuáciu do vopred stanoveného maximálneho času, ktorý označíme symbolom T^{max}. Pri riešení redukovanej úlohy prideľovania dopravných prostriedkov (RVAP) nás zaujíma iba, či sme evakuáciu do daného času schopný vykonať alebo nie. Ak sme schopní evakuáciu do času T^{max} vykonať, znížime tento čas, ktorý tvorí hornú hranicu hodnoty času optimálneho riešenia, a riešime RVAP pre novú hodnotu tohto času. Ak evakuáciu nie sme schopní vykonať, čas T^{max} tvorí dolnú hranicu hodnoty času optimálneho riešenia VAP. V tomto prípade čas T^{max} zvýšime a znova riešime RVAP. Pri jej riešení nám stačí nájsť iba prípustné riešenie (netreba nájsť optimálne), nakoľko nás zaujíma iba informácia, či je možné evakuáciu do času T^{max} vykonať alebo nie. Týmto iteratívnym spôsobom zmenšujeme interval daný dolnou a hornou hranicou času, v ktorom sa nachádza čas optimálneho riešenia VAP.

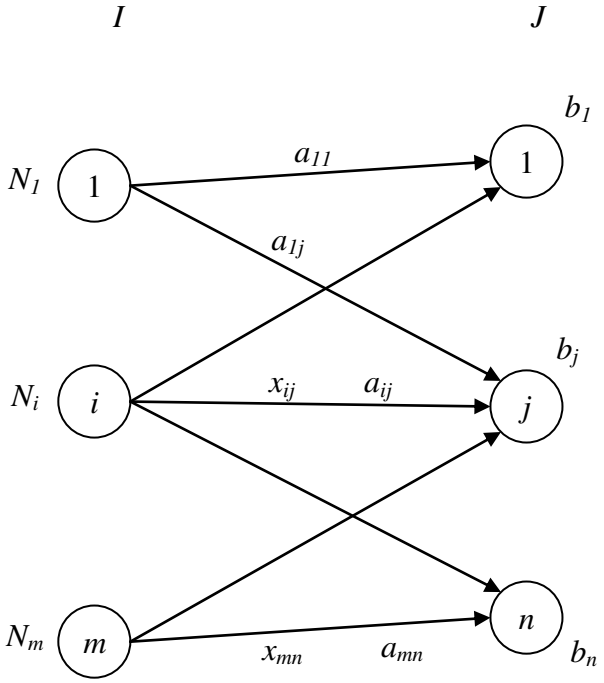
Ak je maximálny čas evakuácie T^{max} daný, môžeme pomocou (7) vyjadriť hodnotu koeficientov $P_{ij}(T^{max})$.

$$P_{ij}\left(T^{\max}\right) = \left\lfloor \frac{T^{\max} - t_{ij} + s_j}{2 s_j} \right\rfloor \qquad pre\ i \in I,\ j \in J \tag{7}$$

Ak je hodnota $P_{ij}(T^{max})$ menšia ako 1, potom vozidlo z parku *i* nemôže byť použité na evakuáciu komunity *j*, nakoľko by nestihlo do času T^{max} navštíviť komunitu ani jedenkrát. Vtedy je komunita *j* z parku *i* nedosiahnuteľná resp. medzi parkom a komunitou neexistuje evakuačná hrana (viď obr. 3), ktorá predstavuje využiteľné spojenie.

A evakuačnú kapacitu $a_{ij}(T^{max})$ môžeme vyjadriť ako pomocou (8).

$$a_{ij}\left(T^{\max}\right) = K_i P_{ij}\left(T^{\max}\right) \qquad pre\ i \in I,\ j \in J \tag{8}$$



Zdroj: [1]

Obr. 3. Grafický model RVAP

Podmienky (3)-(5) z predchádzajúcej podkapitoly sú totožné pre RVAP, a teda úloha spočíva v nájdení prípustného riešenia vyhovujúcemu podmienkam (3)-(5) alebo dokázať, že také riešenie neexistuje.

Iným postupom ako sa vysporiadať s nelinearitou úlohy VAP je metóda Constraint Programming, na ktorú sa pozrieme bližšie v ďalšej kapitole, a takisto sa pozrieme na jej implementáciu pre úlohu VAP.

2 CONSTRAINT PROGRAMMING

Constraint Programming (ďalej len CP) alebo tiež programovanie s obmedzujúcimi podmienkami je metóda určená pre riešenie rozhodovacích problémov, kde cieľom je odpovedať na otázku, či existuje riešenie daného problému (akékoľvek prípustné), prípadne toto riešenie nájsť. Princíp metódy spočíva v tom, že problém sa formuluje deklaratívnym spôsobom pomocou premenných s konečným a diskrétnym definičným oborom a pomocou obmedzujúcich podmienok. Podmienky sa nemusia zapísať v tvare lineárnych rovníc alebo nerovníc; môžu byť logické a nelineárne. Algoritmus (založený na stratégii backtracking) postupne prideľuje hodnoty premenným tak, aby boli splnené obmedzujúce podmienky. Ak sa podarí prideliť hodnoty všetkým premenným, riešenie problému existuje.

CP možno použiť aj na riešenie optimalizačných úloh, kedy hľadáme najlepšie riešenie spomedzi prípustných riešení vzhľadom k určitému kritériu. Metóda riešenia optimalizačného problému využíva iteratívny postup, ktorý bol popísaný v predchádzajúcej kapitole a spočíva v opakovanom riešení rozhodovacieho problému a aktualizovaní dolnej (LB) a hornej hranice (UB) účelovej funkcie (ÚF). Jednou z verzií tohto iteratívneho postupu je dichotomický algoritmus, ktorý možno popísať takto:

2.1 Dichotomický algoritmus pre riešenie optimalizačnej úlohy

1. Vypočítaj LB a UB.
2. Polož D = (LB + UB)/2.
3. Do rozhodovacieho problému pridaj podmienku ÚF ≤ D. Rieš výsledný CSP. Ak existuje riešenie, nastav UB na hodnotu ÚF, inak polož LB = D + 1.
4. Ak LB ≥ UB, koniec (algoritmus našiel optimálne riešenie, ktorého hodnota účelovej funkcie je rovná poslednej hodnote UB). Inak, vráť sa na krok 2.

2.2 Krátka história a použitie

Koncept podmienok bol použitý už v roku 1963 v práci I. Sutherlanda na interaktívnom kresliacom systéme Sketchpad [4]. V sedemdesiatych rokoch boli navrhnuté rôzne experimentálne jazyky, ktoré použili myšlienku podmienok a spoliehali sa na koncept riešenia podmienok.

Koncept úlohy splňovania podmienok bol formulovaný v sedemdesiatych rokoch výskumníkmi umelej inteligencie. Taktiež určili hlavné myšlienky lokálnej konzistencie a algoritmu, ktorý umožňuje ich dosiahnutie. Viaceré metódy hľadania boli definované nezávisle od seba. Niektoré z nich ako backtracking môžeme vystopovať až do 19-tého storočia, kým ostatné ako metóda vetiev a hraníc boli definované v kontexte kombinatorickej optimalizácie. Príspevok CP bol v zistení rôznych nových foriem hľadania, kde kombinuje známe techniky s rôznymi algoritmami propagácie podmienok.

V 80-tych rokoch bol navrhnutý a implementovaný prvý CP jazyk. Najvýznamnejšie jazyky boli založené na paradigme logického programovania (*logic programming*). To viedlo k rozvoju odboru *constraint logic programming*, čo je rozšírenie logického programovania konceptom podmienok. Tým vznikol programovací pohľad, ktorý viedol k identifikácii tzv. *constraint store* ako hlavného konceptu. Constraint store je určitá množina podmienok, ktoré vyhovujú riešeniu. Propagácia podmienok a rôzne formy riešenia sú zvyčajne štandardne zabudované v týchto jazykoch.

V neskorších 80-tych a 90-tych rokoch sa uskutočnila forma syntézy medzi vývojom umelej inteligencie a vývojom constraint logic programming. Výskumníci našli viaceré nové použitia CP, obzvlášť v odbore operačného výskumu a numerickej analýzy. Pokrok bol často dosahovaný identifikovaním dôležitých nových typov podmienok a nových algoritmov propagácie podmienok. Taktiež si uvedomili, že ďalší pokrok môže vychádzať z kombinácie techník z umelej inteligencie, operačného výskumu, počítačovej algebry a matematickej logiky. To nasadilo CP do zaujímavej hybridnej oblasti, v ktorej teoretická práca je často poháňaná uplatnením a v uplatneniach vedie k novým výzvam implementácie CP.

Prístup CP na riešenie problémov je obzvlášť vhodný na riešenie nelineárnych vzťahov podmienok na spojitých premenných.

V minulosti bol CP úspešne použitý na také rôzne typy úloh ako:

- časové a priestorové rozvrhy (*scheduling, routing*)
- priradenie zdrojov (*resource allocation*)
- plánovanie (*planning*)
- konfigurácia siete (*network configuration*)
- časové rozvrhy (*timetabling*)
- bioinformatika

Ďalej bol CP použitý špecificky na problémy ako:

- interaktívne grafické systémy (na vyjadrenie geometrickej konzistentnosti v prípade analýz scén)
- molekulárna biológia (DNA riadenie, konštrukcia 3D modelov proteínov)
- biznis použitie (možnosť obchodovania)
- elektroinžinierstvo (detekcia chýb v obvode, výpočet usporiadania obvodu, testovanie a kontrola návrhu)
- numerické výpočty (riešenie polynomiálnych podmienok so zaručenou presnosťou)
- prirodzené spracovanie jazyka (konštrukcia výkonných parserov)
- počítačová algebra (riešenie a zjednodušenie rovníc cez rôzne algebrické štruktúry)

Rastúcu dôležitosť tejto oblasti nám môže dosvedčiť aj fakt, že v súčasnosti sa konajú každoročné konferencie a kurzy pre CP a jeho použitie, ktoré priťahujú viac ako stovku účastníkov. V roku 1996 sa začal vydávať časopis Constraints a tiež sa objavili špeciálne výtlačky časopisov z oblasti počítačovej vedy venované predmetu CP. Avšak tento odbor je stále veľmi mladý a len pár kníh venovaných CP bolo doteraz vydaných. Pre osvetu a spojenie nadšencov vznikla aj asociácia pre CP – Association for constraint programming (http://4c.ucc.ie/a4cp/), čo je celosvetové združenie pre CP, ktorého cieľom je propagácia CP v každom aspekte vedeckého sveta, a to podporením teoretického a praktického vývoja, vyučovaním v akademických inštitúciách, prijatím v priemyselnom svete a použitím v aplikačných oblastiach. Je to nezisková organizácia, ktorej zisk z organizovaných udalostí sa používa na podporu budúcich udalostí alebo aktivít. Formálny začiatok bol v marci 2005, ale už od roku 1995 existovala organizácia neformálnym spôsobom. Každoročne od roku 1995 organizuje konferenciu v rôznych častiach sveta, okrem toho poskytuje služby ako: informačný bulletin, letné školy, web stránka, vzťahy medzi komunitami, atď.

2.3 Problém splňovania podmienok

Problém splňovania podmienok (Constraint Satisfaction Problem, CSP) definujeme pomocou:

- množiny premenných $X = \{x_1,...,x_n\}$
- každá premenná x_i môže nadobúdať hodnoty z množiny D_i, ktorú nazývame doménou premennej x_i

- množiny podmienok, ktoré určujú, aké hodnoty môžu premenné súčasne nadobúdať, pričom každá podmienka je definovaná na podmnožine X. Množinu podmienok označíme $C = \{c_1,...,c_m\}$.

CSP je potom trojica (X,D,C), napríklad:

- premenné: x_1,x_2,x_3
- domény: $\{0,1\}$ pre x_1, $\{1\}$ pre x_2, $\{0,1,2\}$ pre x_3
- podmienky: $x_1 \neq x_2$ a zároveň $x_2 \neq x_3$

Čiastočné priradenie premenných $(d_1,...,d_k)$, kde $k < n$ nazývame vtedy, ak niektoré premenné majú priradenú hodnotu, napríklad:

- $x_1 = 1, x_2 = 1$

Úplne priradenie premenných $(d_1,...,d_n)$ nazývame, ak všetky premenné majú priradenú hodnotu, napríklad:

- $x_1 = 1, x_2 = 1, x_3 = 0$

Konzistentné priradenie je také, ktorého priradené hodnoty premenných spĺňajú všetky podmienky množiny C. V uvedených príkladoch je priradenie nekonzistentné, keďže priradenie $x_1 = 1$, $x_2 = 1$ nevyhovuje podmienke $x_1 \neq x_2$. Konzistentné priradenie a riešením CSP je napríklad:

- $x_1 = 0, x_2 = 1, x_3 = 2$

Pri CSP môžeme hľadať:

- práve jedno riešenie (je jedno, aké)
- všetky riešenia
- optimálne riešenie, ak je definovaná účelová funkcia (ÚF) (v tomto prípade sa problém splňovania podmienok stáva optimalizačným problémom), napríklad:
    - ÚF definovaná na premenných z X: $f = x_2 + x_3$
    - optimálne riešenie s minimálnou hodnotou ÚF bude: $x_2 = 1, x_3 = 0$

Dôvod pre výber riešenia úlohy pomocou CSP radšej ako matematickým programovaním je dvojnásobný:

- Zobrazenie pomocou CSP je často oveľa bližšie k originálnemu problému: CSP premenné priamo zodpovedajú entitám úlohy, taktiež podmienky môžeme vyjadriť bez potreby transformácie na lineárne nerovnosti. To robí formuláciu úlohy jednoduchšou a riešenie ľahšie na pochopenie

- Hoci sú CSP algoritmy v podstate veľmi jednoduché, môžu nájsť riešenie rýchlejšie ako metódy celočíselného programovania.

Riešenie problému splňovania podmienok sa najčastejšie hľadá tak, že sa systematicky prehľadávajú možné priradenia hodnôt premenným. Obvykle sa pri prehľadávaní stromu riešení používa stratégia prehľadávania do hĺbky. Čiastočné priradenie hodnôt premenným sa v jednom kroku rozšíri o ďalšiu premennú, ak jej možno priradiť hodnotu z jej domény tak, aby sa neporušila žiadna podmienka medzi touto premennou a už naplnenými premennými. Ak sa podarí rozšíriť čiastočné riešenie na všetky premenné, našli sme riešenie problému. Ak neexistuje prípustná hodnota pre aktuálnu premennú, vrátime sa k predchádzajúcej premennej a pokúsime sa priradiť jej inú hodnotu. Ak sa vrátime až ku koreňu stromu riešení a nevychádza z neho ďalšia nepreskúmaná vetva, problém nemá riešenie. Tento spôsob prehľadávania priestoru riešení je známy pod názvom backtracking.

Bactracking má dve hlavné nevýhody:

- Konflikt (nesplnenie podmienky) nevieme predpovedať, odhalí sa až vtedy, keď nastane (keď sa pridelili hodnoty všetkým premenným v podmienke).
- Premennou, ktorá konflikt spôsobila, nemusí byť posledná premenná, ale môže ležať oveľa vyššie v strome riešení. Preto sa opakujú rovnaké neúspešné kroky v rôznych častiach stromu riešení.

Ak vieme dopredu predpovedať, či pridelenie určitej hodnoty premennej spôsobí nesplnenie nejakej podmienky v budúcnosti, môžeme eliminovať spomenuté nedostatky backtrackingu, a tým obmedziť počet prehľadávaných vetiev v strome riešení. Počet konfliktov v budúcnosti obmedzíme, ak po priradení hodnoty aktuálnej premennej zaistíme konzistenciu budúcich premenných s touto premennou tým, že z ich domén vylúčime hodnoty, ktoré nemôžu nadobúdať, ak majú byť splnené podmienky. Tento proces sa nazýva propagácia podmienok (*constraint propagation*)

2.4 Techniky konzistencie

Techniky konzistencie boli prvýkrát predstavené pri zlepšení účinnosti programov na rozpoznávanie obrazu, a to výskumníkmi umelej inteligencie. Rozpoznávanie obrazu zahŕňa označovanie všetkých čiar v obraze konzistentným spôsobom, teda takým, ktorý vyhovuje všetkým podmienkam. Počet možných kombinácií môže byť obrovský, kým len zopár je konzistentných. Techniky konzistencie efektívne vylúčili mnoho nekonzistentných

označovaní vo veľmi skorom štádiu, a tak prerušili pátranie po konzistentnom označovaní. Tieto techniky sa osvedčili ako efektívne na širokom spektre úloh.

Techniky konzistencie sú deterministické na rozdiel od hľadania, ktoré je nedeterministické. Teda deterministický výpočet je vykonávaný čo najskôr a nedeterministický výpočet počas hľadania je použitý, iba ak už nie je možná žiadna ďalšia propagácia podmienok. Napriek tomu, techniky konzistencie sú zriedka použité na vyriešenie CSP úplne (aj keď to dokážu).

2.4.1 Konzistencia vrcholu

Budeme predpokladať, že problém splňovania podmienok je binárny, t.j. v každej podmienke sa vyskytujú len dve premenné. Príkladom binárneho problému splňovania podmienok je problém *k*-zafarbenia grafu. Každému vrcholu grafu zodpovedá jedna premenná s doménou {1, 2, ..., *k*}. Všetky podmienky sú binárne a vyjadrujú, že premenné reprezentujúce susedné vrcholy nemôžu mať rovnakú hodnotu.

Binárny problém môžeme reprezentovať grafom *G*, v ktorom vrcholy zodpovedajú premenným a hrany podmienkam.

Najjednoduchšia technika konzistencie je známa ako konzistencia vrcholu. Vrchol *v* reprezentujúci premennú *x* v grafe je vrcholovo konzistentný, ak pre každú hodnotu *a* v aktuálnej doméne premennej *x*, každá unárna podmienka pre *x* je splnená.

Ak doména *D* premennej *x* obsahuje hodnotu *a*, ktorá nespĺňa unárnu podmienku pre *x*, potom priradenie hodnoty *a* premennej *x* vždy skončí okamžitým zlyhaním pri hľadaní prípustného riešenia. Nekonzistentnosť vrcholu môže byť odstránená jednoduchým odobratím tých hodnôt z domény *D* pre premennú *x*, ktoré nespĺňajú unárnu podmienku pre *x*.

Algoritmus pre zaistenie konzistentnosti vrcholu môžeme v pseudokóde zapísať takto:

Algoritmus KV

```
begin
    Pre ∀ premenné x
        Pre ∀ hodnoty a v doméne D premennej x
            Ak a nevyhovuje niektorej unárnej podmienke pre x, potom
                Odober a z D
end;
```

2.4.2 Konzistencia hrany

Ak graf je vrcholovo konzistentný, potom unárne podmienky môžu byť odobraté, pretože sú všetky splnené.

Hranu (x_i,x_j) nazveme konzistentnou, ak pre každú hodnotu a z aktuálnej domény D_i existuje nejaká hodnota b z D_j taká, že priradenie $x_i = a$ a $x_j = b$ vyhovuje binárnej podmienke medzi x_i a x_j. Koncept konzistencie hrany je orientovaný, t.j. ak hrana (x_i,x_j) je konzistentná, neznamená to automaticky, že (x_j,x_i) je tiež konzistentná.

Očividne, hranu (x_i,x_j) môžeme urobiť konzistentnou jednoduchým odstránením tých hodnôt z domén premennej x_i, pre ktoré neexistuje príslušná hodnota v doméne D_j taká, že binárna podmienka medzi x_i a x_j je splnená (odstránenie takých hodnôt nevylúči žiadne riešenie z originálneho CSP). Presne to robí nasledujúci algoritmus Revise.

```
Procedure Revise(xi,xj)
begin
    Delete := false;
    Pre ∀ hodnoty a v doméne Di
        Ak neexistuje také b v Dj pre ktoré (a,b) je konzistentné potom
            Odober a z Di
            Delete := true;
    Return Delete;
end;
```

Na to aby bol graf úplne hranovo-konzistentný, nepostačuje vykonať Revise pre každú hranu iba raz. Ako náhle Revise redukuje domény niektorej premennej x_i, potom každá predchádzajúca revidovaná hrana (x_j,x_i) musí byť zrevidovaná znova, pretože niektoré členy domény x_j nemusia byť viac kompatibilné s niektorými ostávajúcimi členmi zrevidovanej domény premennej x_i. To robí nasledujúci algoritmus, známy ako AC-3. Q predstavuje zoznam hrán.

procedure AC-3

begin

Q ← ∀ dvojice (x_i,x_j), kde $i \neq j$

Kým nie je Q prázdne

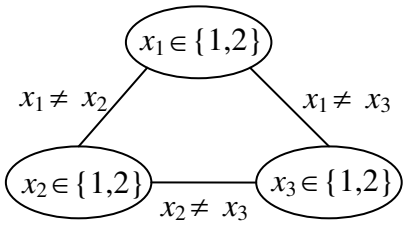
Vyber a odober ľubovoľný oblúk (x_k,x_m) z Q;

Ak Revise(x_k,x_m) potom

Q ← Q zjednotenie {(x_i,x_k) také že $i \neq k$, $i \neq m$}

end;

Udržiavanie konzistencie hrany odstráni mnoho nekonzistencie z grafu, ale nezaručuje, že vznikne riešenie pre CSP. Ak v doméne pre každú premennú zostane len jedna premenná, potom CSP má práve jedno riešenie, ktoré vznikne tak, že každej premennej priradíme jedinú možnú hodnotu v jej doméne. Inak riešenie nemožno určiť. Nasledujúci príklad ukazuje prípad, kde graf je hranovo-konzistentný, domény nie sú prázdne, ale stále neexistuje riešenie spĺňajúce všetky podmienky.



Zdroj: [2]

Obr. 4. Graf podmienok

2.4.3 *K* – konzistencia (Konzistencia cesty)

Za predpokladu, že konzistencia hrany nestačí na odstránenie potreby backtrackingu, naskytá sa otázka, či existuje nejaký silnejší stupeň konzistencie, ktorý môže odstrániť potrebu hľadania. Graf je *k*-konzistentný, ak platí:

Vyber hodnoty niekoľkých *k*-1 premenných, ktoré spĺňajú všetky podmienky medzi týmito premennými a vyber niektorú *k*-tú premennú. Potom existuje hodnota pre *k*-tú premennú, ktorá spĺňa všetky podmienky medzi týmito *k* premennými.

Graf je silne *k*-konzistentný, ak je *j*-konzistentný pre všetky $j <= k$.

Konzistencia vrcholu je vlastne ekvivalentná silnej 1-konzistencii a konzistencia hrany je ekvivalentná silnej 2-konzistencii. Existujú taktiež algoritmy pre zhotovenie grafu

silne k-konzistentného pre k>2, ale v praxi sa používajú zriedkavo kvôli výpočtovej náročnosti. Výnimku tvorí algoritmus 3-konzistencie, ktorý je označovaný ako konzistencia cesty.

Vrchol reprezentujúci premennú x_i je slabo cestovo-konzistentný, ak je hranovo-konzistentný, t.j. všetky hrany z tohto vrchola sú hranovo-konzistentné, a tiež platí: Pre každú hodnotu a v doméne D_i premennej x_i existuje práve jedna hodnota b z domény náhodnej premennej x_j taká, že existuje hodnota c v doméne premennej x_k, pre ktorú dvojica (a,c) je prípustná binárnou podmienkou medzi x_i a x_k, a (c,b) je prípustná binárnou podmienkou medzi x_k a x_j.

Algoritmus pre zhotovenie slabo cestovo-konzistentného grafu odstráni viac nekonzistentných hodnôt ako algoritmus konzistencie hrany, avšak neodstráni potrebu hľadania vo všeobecnosti. Samozrejme, ak graf podmienok obsahujúci n uzlov je silne nekonzistentný, potom riešenie pre CSP môžeme nájsť bez hľadania. Najhoršia zložitosť pre algoritmus na získanie n-konzistentného grafu je exponenciálna. Ak je graf silno k-konzistentný pre $k < n$, potom vo všeobecnosti sa nevyhneme backtracking-u, pretože stále existujú nekonzistentné hodnoty.

2.5 Propagácia podmienok

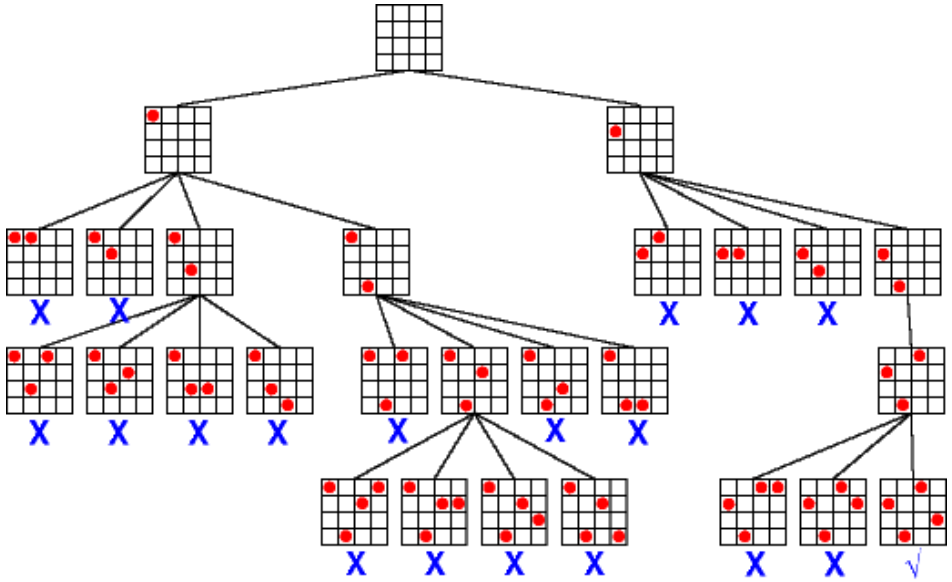
V predchádzajúcej časti sme uviedli techniky konzistencie ako spôsob riešenia CSP. Druhý spôsob riešenia je pomocou backtracking-u, ktorý si vysvetlíme bližšie. A tretí používaný spôsob je vloženie algoritmu konzistencie do algoritmu backtracking-u. Základná kostra je jednoduchý algoritmus backtracking, ktorý sa postupne snaží rozširovať čiastočné riešenie, ktoré fixuje hodnoty pre niektoré z premenných, ku úplnému riešeniu, opakovaním vyberaním hodnoty pre ďalšiu premennú konzistentnú s hodnotami v aktuálnom čiastočnom riešení. Po vybratí hodnotí premennej, je použitá niektorá z techník konzistencie. V závislosti od stupňa techniky konzistencie dostaneme rôzne algoritmy pre propagáciu podmienok.

2.5.1 Backtracking

Dokonca aj jednoduchý backtracking vykonáva určitý druh techniky konzistencie, aj keď to je len zlomok konzistencie hrany. Algoritmus testuje konzistenciu hrany medzi aktuálnou premennou a fixovanými premennými, t.j. algoritmus kontroluje platnosť

podmienok vzhľadom na čiastočné riešenie. Ak nejaká hodnota aktuálnej premennej nie je konzistentná, algoritmus vykoná nové priradenie.

Ukážeme si algoritmus na príklade úlohy 4 kráľovien (obr. 5), ktorá spočíva v rozmiestnení štyroch kráľovien na šachovnici 4x4 a to tak, aby sa navzájom neohrozovali, čiže žiadne dve kráľovné sa nemôžu nachádzať na rovnakom riadku, stĺpci či diagonále.



Zdroj: [2]

Obr. 5. Úloha 4 kráľovien a backtracking

Algoritmus môžeme ľahko rozšíriť o návrat ku konfliktnej premennej, a tak vykonať určitý druh spätného prechodu alebo tzv. inteligentného backtracking-u. To nám však pridáva ďalšie náklady ku algoritmu, a teda sa zdá, že zabránenie možných budúcich konfliktov je viac opodstatnené ako zotavovanie sa z nich.

2.5.2 Forward checking

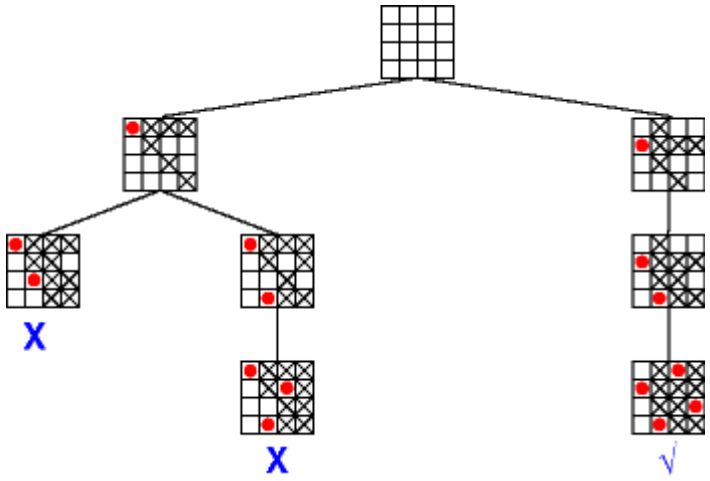
Forward checking je najľahší spôsob ako zabrániť budúcim konfliktom. Namiesto kontroly konzistencie hrany k priradeným premenným, vykoná obmedzenú formu konzistencie hrany k ešte nepriradeným premenným. Obmedzenú v tom, že skontroluje iba podmienky medzi aktuálnou premennou a budúcimi premennými, ktoré ešte nemajú priradenú hodnotu a sú spojené spoločnou podmienkou s aktuálnou premennou. Keď sa priradí hodnota aktuálnej premennej, hocijaká hodnota v doméne budúcej premennej, ktorá je konfliktná s týmto priradením je (dočasne) odstránená z domény budúcej premennej. Výhoda je teda v tom, že ak sa doména budúcej premennej stane prázdnou, vieme

okamžite, že aktuálne čiastočné priradenie je nekonzistentné. Forward checking umožňuje orezať vetvy stromu riešenia, v ktorých nie je prípustné riešenie skôr ako jednoduchý backtracking. Taktiež vždy, keď sa ide fixovať nová premenná, všetky jej ostávajúce hodnoty majú zaručenú konzistenciu s minulými premennými, takže kontrola priradenia oproti minulým premenným nie je viac nutná. Procedúra vyzerá nasledovne, Q je zoznam hrán, cv predstavuje aktuálnu premennú:

```
procedure FC(cv);
begin
        Q ← ∀ dvojice (x_i,x_cv), kde i > cv
        consistent := true;
        kým Q nie je prázdne & consistent
                vyber a odober ľubovoľnú hranu (x_k,x_m) z Q;
                ak Revise(x_k,x_m) potom
                        consistent := not D_k empty;
        return consistent;
end;
```

Tým, že algoritmus zistí nekonzistenciu skôr ako backtracking, umožní skôr orezať vetvy stromu riešenia, ktoré by viedli k zlyhaniu. To zmenší strom riešenia a dúfajme aj prácu, pretože forward checking urobí viac práce pri každom priradení pridanému k čiastočnému riešeniu. Forward checking je skoro vždy lepšia voľba ako backtracking.

Na obr. 6 je ilustrovaný algoritmus forward checking pre úlohu 4 kráľovien.



Zdroj: [2]

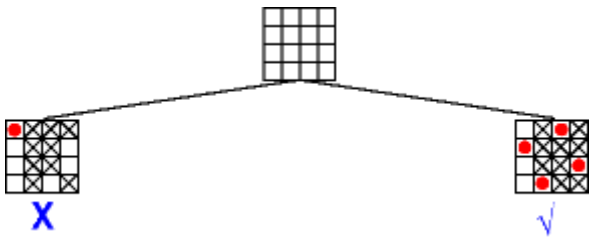
Obr. 6. Úloha 4 kráľovien a forward checking

2.5.3 Look ahead

Forward checking kontroluje len podmienky medzi aktuálnou premennou a budúcimi premennými. Look ahead je prístup, kedy sa snažíme vykonať plnú konzistenciu hrany, ktorá ešte viac zmenší domény a odstráni možné konflikty. Výhoda look ahead je, že odhalí viac konfliktov medzi budúcimi premennými, a preto umožní orezať neproduktívne vetvy stromu riešenia ešte skôr ako forward checking. Taktiež ako pri forward checking, vždy keď prechádzame ku novej premennej, všetky jej ostávajúce hodnoty majú zaručenú konzistenciu s minulými premennými, a preto kontrola priradenia oproti minulým premenným nie je potrebná. Procedúra vyzerá nasledovne:

```
procedure LA(cv);
begin
    Q ← ∀ dvojice (x_i,x_cv), kde i > cv
    consistent := true;
    kým Q nie je prázdne & consistent
        vyber a odober ľubovoľný oblúk (x_k,x_m) z Q;
        ak Revise(x_k,x_m) potom
         Q ← Q zjednotenie {( x_i,x_k), take že i ≠ k, i ≠ m, i > cv}
        consistent := not D_k empty;
    return consistent;
end;
```

Na obr. 7 je ilustrovaný algoritmus look ahead pre úlohu 4 kráľovien.

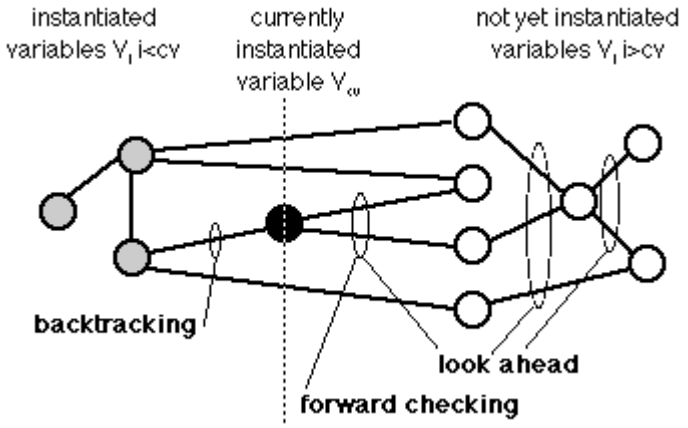


Zdroj: [2]

Obr. 7. Úloha 4 kráľovien a look ahead

2.5.4 Porovnanie a zhrnutie

Nasledujúci obrázok ukazuje, ktoré podmienky sú testované, keď sú použité vyššie spomenuté techniky propagácie podmienok.



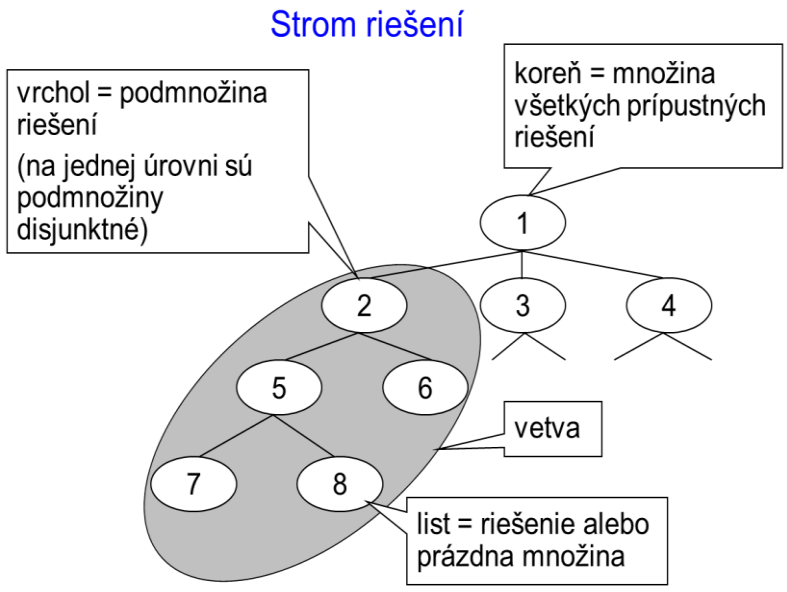
Zdroj: [2]

Obr. 8. Porovnanie algoritmov propagácie podmienok

Viac propagácie podmienok pri každom vrchole stromu riešení bude mať za následok menej vrcholov v strome riešení, ale celková cena (doba výpočtu) môže byť vyššia, pretože spracovanie pri každom vrchole bude drahšie. V jednom extréme, získanie silnej *n*-konzistencie pre pôvodnú úlohu by mohlo úplne odstrániť potrebu hľadania, čo je však obyčajne drahšie ako jednoduchý backtracking. V skutočnosti, v niektorých prípadoch dokonca aj plný look ahead môže byť drahší ako jednoduchý backtracking. Preto sa najčastejšie v praxi používa forward checking a backtracking.

2.6 Branch and Prune

Kombinácia algoritmu backtracking a propagácie podmienok sa označuje ako metóda Branch and Prune. Metóda prehľadáva strom riešení (obr. 9), pričom využíva propagáciu podmienok, čo je filtrovací algoritmus, ktorý odstráni z domén premenných neprípustné hodnoty, a to tie, ktoré nevyhovujú podmienkam úlohy, čím sa zmenší priestor pre hľadanie prípustného riešenia.



Obr. 9. Strom riešení

Algoritmus Branch and Prune

1. Preskúmaj koreň stromu (vykonaj propagáciu podmienok) a vlož ho do zoznamu.
2. Pokiaľ existuje nepreskúmaný následník prvého vrcholu v zozname (nenastavená, nefixovaná premenná), opakuj:
   a) Preskúmaj následníka.
      - Vyber niektorú premennú.
      - Priraď jej nejakú hodnotu z jej domény.
      - Vyvoď dôsledky tohto priradenia (vykonaj propagáciu podmienok).
   b) Ak sa nejaká doména stala prázdnou množinou, potom doterajšie priradenie hodnôt premenným je neprípustné. Inak, vlož následníka do zoznamu.
3. Ak si našiel riešenie (všetky premenné sú fixované), koniec.
4. Vyber prvý vrchol zo zoznamu.
5. Ak je zoznam prázdny, koniec (riešenie neexistuje). Inak, vráť sa na krok 2.

Propagácia podmienok sa udeje vždy po priradení hodnoty aktuálnej premennej (krok 2a) a znamená to dočasné vyhodenie hodnôt z domén budúcich premenných (minulé premenné sú už fixované), a to tých, ktoré sú v rozpore s týmto priradením. Samozrejme, môže nastať situácia, kedy sa doména niektorej budúcej premennej stane prázdnou (priradenie nie je prípustné), a teda musíme skúsiť priradiť aktuálnej premennej inú hodnotu, alebo sa vrátiť v strome riešení, ak žiadna hodnota aktuálnej premennej nie je

prípustná. Propagácia podmienok teda oreže tie vetvy stromu riešení, ktoré neobsahujú prípustné riešenia.

Poradie premenných pri vetvení v kroku 2a) **Vyber niektorú premennú** môže byť určené rôznym spôsobom, a to :

- statickým – poradie je určené dopredu podľa určitého kľúča, napr. podľa zadeklarovania, podľa výskytu v podmienkach od najčastejšej premennej, podľa prírastu k ÚF;
- dynamickým – poradie premenných sa v priebehu hľadania mení, napr. vyberá sa premenná s najmenším počtom zostávajúcich hodnôt (najmenšou doménou). Táto heuristika (známa ako princíp first-fail) je založená na myšlienke, že ak aktuálne čiastočné riešenie nevedie k výsledku, potom čím skôr sa na to príde, tým lepšie, t.j. tým menej neproduktívnych vetiev v strome riešení sa prehľadáva.

Priradenie hodnoty premennej v kroku 2a) **Priraď jej nejakú hodnotu z jej domény** sa riadi znalosťou problému, a to napr. princípom succeed-first, kde preferujeme hodnoty, ktoré pravdepodobne patria k riešeniu, alebo v optimalizačnej úlohe vyberáme hodnotu s najmenším prírastkom k ÚF.

2.7 CP jazyky

V súčasnosti existuje mnoho programov, modelovacích jazykov a solverov, ktoré fungujú na metóde CP. Keďže neexistuje žiadny štandardný modelovací jazyk, väčšina solverov používa svoj vlastný. To sťažuje prácu používateľov, pretože experimentovanie s rozličnými solvermi, vyžaduje znalosť viacerých modelovacích jazykov. Uvedieme niektoré z nich.

Constraint Logic Programming je CP použitý v logickom programovaní, ktoré patrí medzi deklaratívne spôsoby programovania (napr. jazyk Prolog). V tomto odvetví existuje viacero jazykov, niektoré z nich sú:

- B-Prolog: na základe jazyka Prolog, proprietárny softvér
- CHIP V5: na základe jazyka Prolog, zahŕňa C++ a C knižnice, proprietárny softvér
- Ciao: na základe jazyka Prolog, voľný softvér s licenciou GPL (General Public License) / LGPL (Lesser General Public License)
- ECLiPSe: na základe jazyka Prolog, open source

Niektoré CP knižnice pre rôzne programovacie jazyky:

- Artelys Kalis: C++ knižnica, modul pre Xpress-Mosel, proprietárny softvér
- Choco: Java knižnica, voľný softvér s licenciou X11
- Emma: Python knižnica, proprietárny softvér
- Gecode: C++ knižnica, voľný softvér s licenciou X11
- Google CP Solver: Python, Java, C++ a .NET knižnice, licencia Apache
- IBM ILOG CP Optimizer: C++, Java, .NET knižnice, proprietárny softvér
- Turtle: voľný softvér s licenciou GPL

Niektoré jazyky, ktoré podporujú CP:

- AIMMS: algebrický modelovací jazyk s podporou pre CP
- Bertrand: jazyk na budovanie CP systémov
- G12 MiniZinc: CP systém na vysokej úrovni, licencia BSD
- Kaleidoscope: objektovo-orientovaný imperatívny CP jazyk

Môžeme vidieť, že výber jazykov je veľký a líšia sa tak v jednoduchosti, či zložitosti použitia, ako aj na čo sú určené, teda nakoľko nám to umožňuje licencia. Jedným z jednoduchších jazykov je MiniZinc, ktorý si bližšie rozoberieme v kapitole 4.

3 IMPLEMENTÁCIA METÓDY CONSTRAINT PROGRAMMING PRE VAP

Pre problém splňovania podmienok v úlohe VAP je potrebné stanoviť si premenné, ich domény a podmienky, preto:

- označíme evakuačný čas ako *T*, ktorý sa snažíme minimalizovať
- a_{ij} označíme evakuačné kapacity parku *i* komunite *j* pre čas *T*
- b_j predstavuje počet obyvateľov v obci *j*, ktorých ešte treba evakuovať
- N_i je počet vozidiel, ktoré ešte sú k dispozícii v parku *i*
- rozhodovacie premenné označíme x_{ij} a znamenajú počet vozidiel pridelených z parku i komunite j
- doména premennej x_{ij} je $\{0,1,2,\ldots D_{ij}^{\max}\}$, kde $D_{ij}^{\max} = \min\{N_i, \lceil b_j / a_{ij} \rceil\}$

Ďalej si označme množinu *J(i)* ako množinu komunít, ktoré sú dostupné z parku *i*, teda existuje evakuačná hrana medzi *i* a $j \in J(i)$, resp. evakuačná kapacita a_{ij} je väčšia ako 0, čo znamená, že je možné použiť pri evakuácii komunity *j* park *i* v čase *T*. A podobne si označme množinu *I(j)* ako množinu parkov, ktoré môžeme použiť pri evakuácii komunity *j*.

Po pridelení hodnoty premennej x_{ij} sa zníži počet obyvateľov v obci *j*, ktorých ešte treba evakuovať ($b_j = b_j - a_{ij}x_{ij}$) a počet vozidiel, ktoré zostanú v parku *i* ($N_i = N_i - x_{ij}$). Potom sa vykoná propagácia podmienok, teda zmena domén premenných x_{il} pre tie komunity *l*, ktoré sú dostupné z parku *i* ($l \in J(i), l \neq j$) a premenných x_{kj} pre tie parky *k*, z ktorých je dostupná komunita *j* ($k \in I(j), k \neq i$).

Pri propagácii podmienok využijeme tieto vedomosti o probléme:

Pre všetky komunity *l*, ktoré ovplyvní nastavenie premennej x_{ij} (sú dostupné z parku *i*), treba skontrolovať, či ich možno evakuovať pomocou zvyšných vozidiel, teda vyhodnotiť podmienku:

$$b_l \leq \sum_{k \in I(l)} N_k a_{kl} \text{ pre} l \in J(i), l \neq j \tag{9}$$

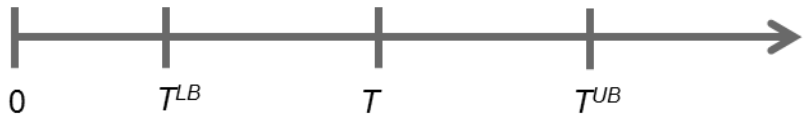
Ak premenná x_{ij} je posledným priradením pre komunitu *j*, treba skontrolovať, či je komunita *j* evakuovaná, teda:

$$b_j \leq 0 \tag{10}$$

Ak podmienka (9) nie je splnená pre niektorú komunitu alebo podmienka (10) nie je splnená pre komunitu j v prípade, že ide o posledné priradenie vozidiel komunite j, aktuálne priradenie hodnoty premennej x_{ij} nie je konzistentné a treba jej priradiť inú hodnotu, poprípade vrátiť sa v strome riešení.

Úloha VAP má prípustné riešenie vtedy, ak existuje také priradenie vozidiel, ktoré nám zabezpečí evakuáciu všetkých obyvateľov v danom čase T. Inak pre čas T neexistuje prípustné riešenie.

Riešime rozhodovaciu úlohu VAP pre čas T, ktorý chceme minimalizovať. Dichotomickým algoritmom dokážeme optimalizovať riešenie. Preto si stanovíme dolný T^{LB} a horný odhad T^{UB} pre čas evakuácie. A pomocou vzorca $T = (T^{LB} + T^{UB})/2$ získame čas T, pre ktorý spustíme algoritmus Branch and prune, ktorého úlohou bude zistiť, či v čase T existuje prípustné riešenie. Situáciu nám zobrazuje obr. 10.



Obr. 10. Dichotomický algoritmus

Stanovenie dolného odhadu času je založené na nasledujúcej úvahe. Najlepší čas evakuácie by sme dosiahli, keby sme každú obec navštívili len raz vozidlami z najbližšieho parku, preto si určíme:

$$T^{LB} = \max_{j \in J} \left\{ \min_{i \in I(j)} \{t_{ij}\} + s_j \right\}$$

3.1 Heuristiky

Horný odhad času evakuácie T^{UB} získame pomocou heuristiky, a to ako čas rýchlo nájdeného prípustného riešenia.

Algoritmus pre výpočet T^{UB}

0. Načítaj sieť podľa zadanej úlohy. Usporiadaj komunity.
1. $T = T^{LB}$
2. Vypočítaj evakuačné kapacity pri danom čase T.
3. Vyber prvú neobslúženú komunitu s a park r s voľnou kapacitou. Nastav premennú $x_{rs} = \min\{N_r, \lceil b_s / a_{rs} \rceil\}$. Aktualizuj hodnoty $N_r = N_r - x_{rs}$ a

$b_s = b_s - a_{rs}x_{rs}$. Opakuj krok 3, pokiaľ mám neobslúženú komunitu, alebo pokiaľ mám park s voľnou kapacitou.

4. Ak je riešenie konzistentné, koniec – horný odhad T^{UB} pre čas evakuácie je T.
5. Inak zvýš T o 1 a pokračuj krokom 2.

Konzistentné riešenie je vtedy, ak má každá komunita dostatok pridelených vozidiel tak, aby sa do času T^{UB} stihli evakuovať všetci jej obyvatelia.

Pri usporiadaní komunít a parkov v kroku 0 boli použité rôzne stratégie, a to:

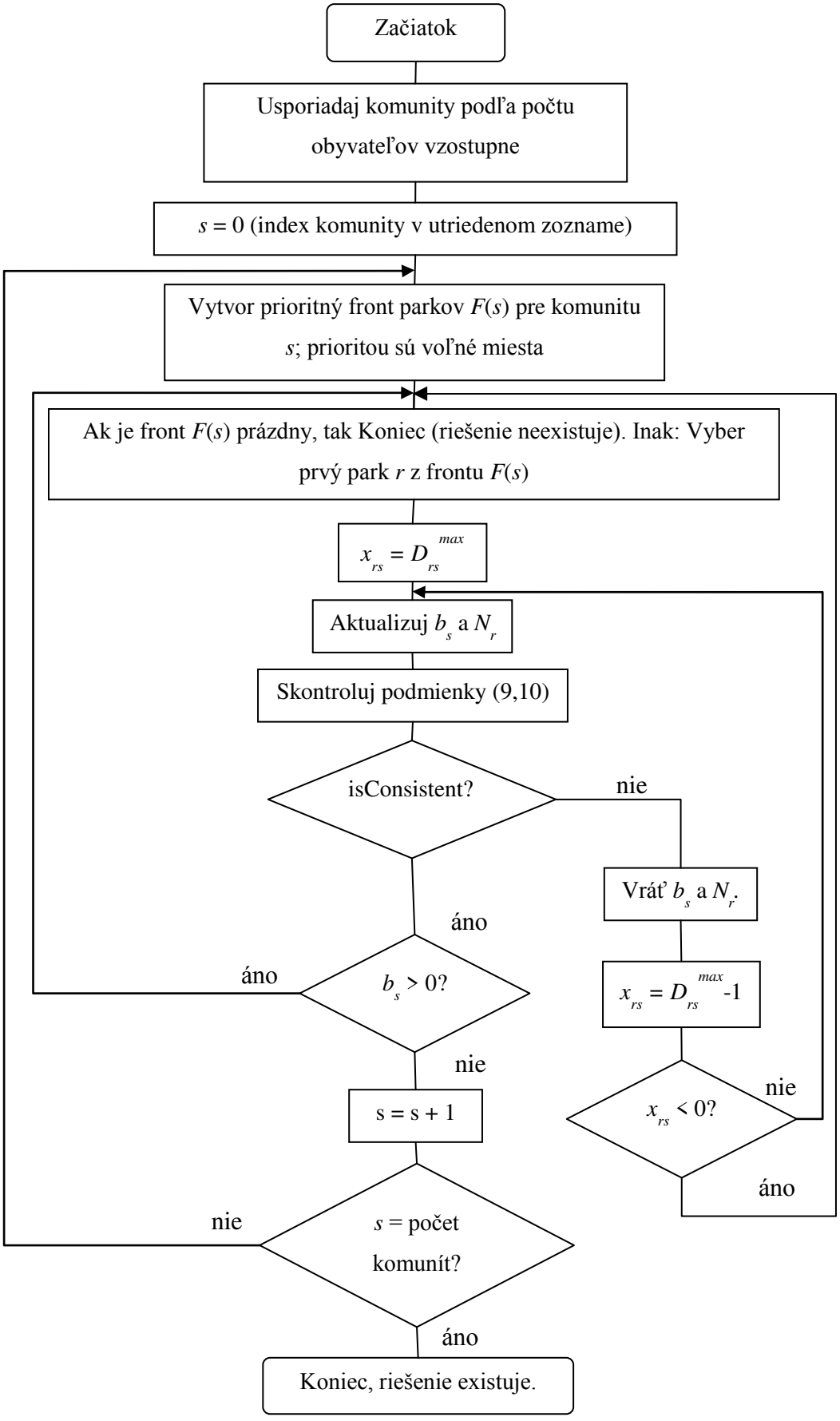
- S1 - stratégia, ktorá usporiada komunity a parky podľa ich výskytu vo vstupných súboroch. V kroku 3 zoberie prvý park v poradí a prvú komunitu, ktorej priradí potrebný počet vozidiel. Ak je potrebný počet vozidiel menší ako počet vozidiel v parku, pokračuje ďalšou komunitou. Inak prejde na ďalší park a pridelí zvyšok vozidiel k ešte celkom neobslúženej komunite. Ostatné stratégie sa líšia poradím komunít a parkov.
- S2min – stratégia, ktorá v kroku 0 zoradí komunity vzostupne podľa počtu obyvateľov, čím sa prvou komunitou stane komunita s najmenším počtom obyvateľov a potom postupne prideľuje vozidlá ako v stratégii S1.
- S2max – stratégia, ktorá v kroku 0 zoradí komunity zostupne podľa počtu obyvateľov, čím sa prvou komunitou stane komunita s najväčším počtom obyvateľov a potom postupne prideľuje vozidlá ako v stratégii S1.
- S3min – stratégia, ktorá v kroku 0 zoradí komunity vzostupne podľa počtu obyvateľov. Park r pre komunitu s v kroku 3 vyberá tak, aby strata voľných miest vo vozidlách bola čo najmenšia: $r = \arg\min_{i \in I(s)} \{\lceil b_s / a_{is} \rceil * a_{is} - b_s\}$.
- S3max – stratégia, ktorá v kroku 0 zoradí komunity zostupne podľa počtu obyvateľov a parky vyberá ako v stratégii S3min.

Samozrejme, mohli by sme vymyslieť mnoho ďalších stratégii, tieto však pre náš účel postačovali.

3.2 Algoritmus Branch and prune

Ďalším krokom je opakované spustenie algoritmu Branch and prune pre čas $T = (T^{LB} + T^{UB})/2$, až kým sa dostaneme k optimálnemu riešeniu. Ak algoritmus zistí, že v čase T existuje riešenie, nastaví $T^{UB} = T$, inak $T^{LB} = T$ a opäť spustí algoritmus Branch and prune pre nový čas T. Ak $T^{UB} - T^{LB} = 1$, máme optimálne riešenie. Týmto urýchlime výpočet, pretože ak by sme ho spúšťali pre každý čas, výpočet by trval veľmi dlho. V prípade dobrého horného odhadu evakuačného času T^{UB} je možnou alternatívou nastaviť $T = T^{UB}$ -1 a v každej iterácii ho znížiť o jednu minútu, až kým algoritmus Branch and prune nezistí, že pre čas T neexistuje prípustné riešenie. Optimálne riešenie je teda čas T + 1. Týmto zlepšením môžeme znížiť počet opakovaní algoritmu Branch and prune, čo sa v našom prípade ukázalo ako lepšia možnosť.

Na obr. 11 je vývojový diagram algoritmu Branch and prune, ktorý zistí, či existuje alebo neexistuje prípustné riešenie pre zadané vstupy: vektory $\boldsymbol{N}$, $\boldsymbol{b}$, matice $\boldsymbol{a}$, $\boldsymbol{D}^{\max}$. Algoritmus predpokladá, že nutná podmienka (9) pre existenciu prípustného riešenia pre zadané T je splnená pre všetky komunity. Pri jeho realizácii nám pomohla heuristika S3min, z ktorej sme sa inšpirovali technikou ako vyberať komunity a parky, čo pomohlo k urýchleniu nájdeniu prípustného riešenia.



Obr. 11. Algoritmus Branch and prune

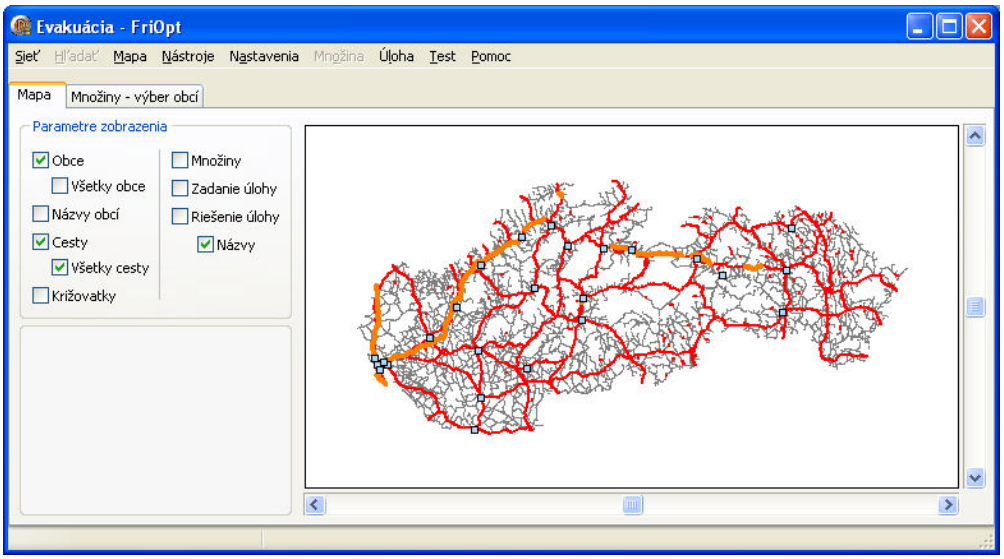
Ako vidíme samotný algoritmus nie je zložitý, čo je aj výhodou metódy Constraint Programming, a môže sa skončiť dvoma prípadmi – nájdením prípustného riešenia, alebo ukončením bez toho, aby našiel prípustné riešenie v čase *T*, teda vieme povedať, že riešenie v čase *T* neexistuje.

3.3 Pamäťová spotreba

V praxi sme obmedzení či už časovými alebo pamäťovými zdrojmi, a preto môže nastať situácia, kedy algoritmus Branch and prune neskončí v rozumnom čase, alebo počítaču nevystačia pamäťové zdroje. Vtedy nevieme povedať, či riešenie v čase *T* existuje alebo nie. Pri predčasnom ukončení výpočtu sa z predtým exaktnej metódy stane heuristická. K tomuto kroku sme sa museli rozhodnúť aj my, a to už z vyššie spomenutých pamäťových dôvodov. Obmedzili sme počet priradení, ktoré algoritmus v priebehu výpočtu vyskúša, na hodnotu 1 000 000. Vo výsledkoch výpočtových experimentov je počet priradení označovaný ako *počet krokov*.

3.4 Použitý nástroj na podporu rozhodovania

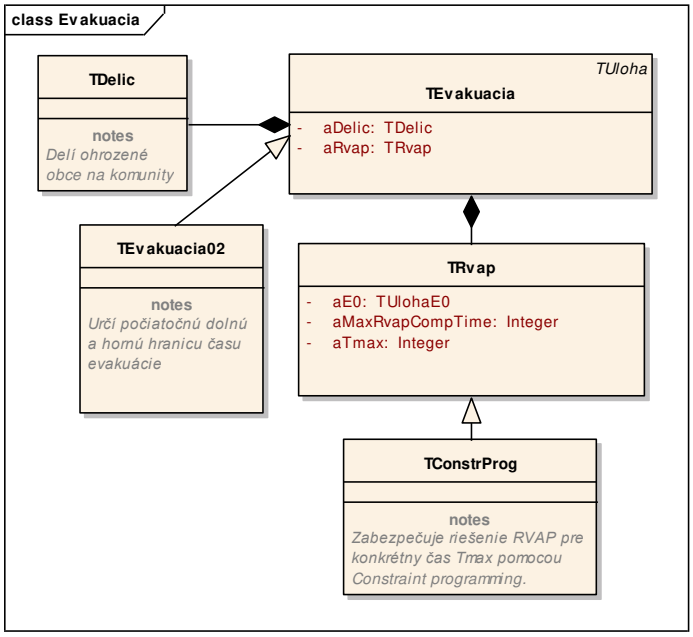
Pre implementáciu bol použitý špecializovaný nástroj, ktorý bol prevzatý a doplnený o metódu Constraint Programming. Nástroj umožňuje všetky potrebné funkcie pre prácu s mapou a je špeciálne upravený pre potreby evakuačnej úlohy. Samotný nástroj nebol predmetom tvorby tejto diplomovej práce, ale bol prevzatý s dovolením Ing. Tomana. Je vytvorený v jazyku Delphi. Jeho grafické rozhranie možno vidieť na obr. 12.



Zdroj: [1]

Obr. 12. Grafické používateľské rozhranie špecializovaného nástroja na podporu rozhodovania

Pre metódu CP bola v nástroji vytvorená nová trieda TConstrProg a trieda TEvakuacia02, ktoré môžeme vidieť na obr. 13, čo je pre pochopenie zjednodušený diagram tried. Trieda TEvakuacia02 je potomkom triedy TEvakuacia, ktorá riadi celú úlohu evakuácie. Trieda TEvakuacia02 má modifikované metódy pre určenie dolnej a hornej hranice času evakuácie. Trieda TConstrProg je potomkom triedy TRvap, ktorá riadi úlohu prideľovania vozidiel. Trieda TConstrProg obsahuje metódu CP, všetky potrebné procedúry a funkcie.



Obr. 13. Zjednodušený diagram tried

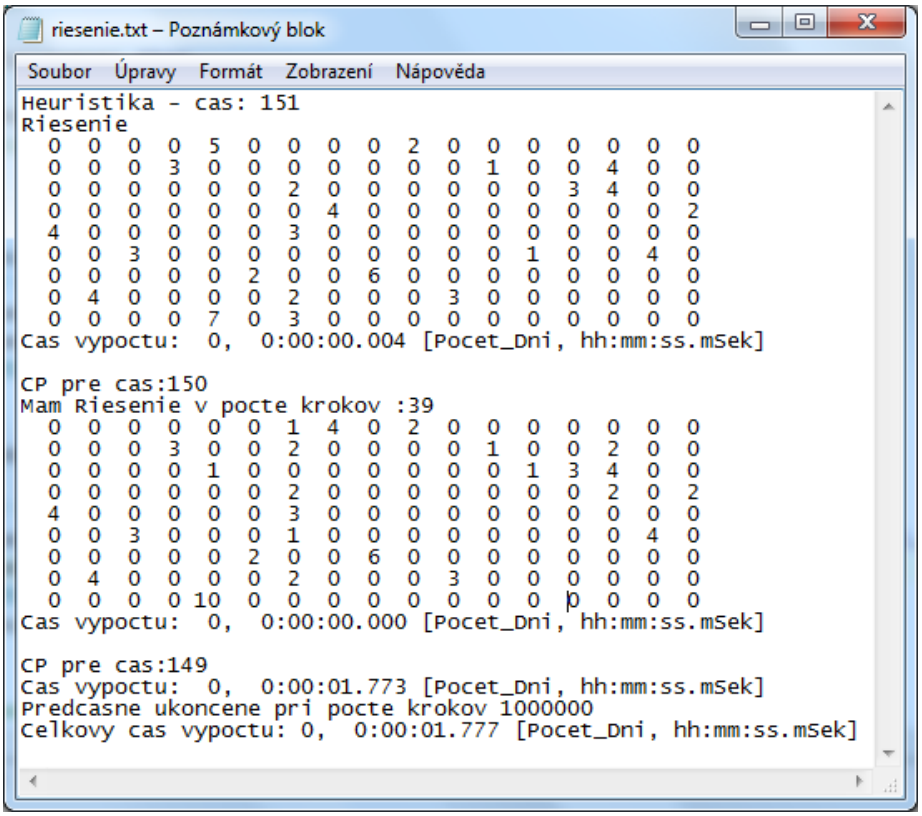
Nástroj neposkytuje zobrazenie riešenia graficky na mape, a to z dôvodu neprehľadnosti, preto sa pri riešení úlohy vytvorí protokol, v ktorom sa nachádza výpis riešenia.

3.5 Protokol riešenia

Protokol, ktorý sme vytvorili pri riešení úlohy metódou CP vyzerá nasledovne. V protokole sa najprv nachádza riešenie nájdene pomocou heuristiky, a to:

- evakuačný čas riešenia v minútach
- priradenie vozidiel, pričom riadky zodpovedajú jednotlivým parkom, stĺpce zodpovedajú komunitám a číslo vyjadruje počet vozidiel priradených z parku ku komunite
- výpočtový čas

Nasleduje výpis riešenia CP algoritmu s rovnakými informáciami pre každé volanie metódy Branch and prune. Na konci protokolu sa nachádza informácia o tom, ako riešenie skončilo, pričom môžu nastať dva stavy: riešenie pre daný čas evakuácie neexistuje, alebo riešenie sa nenašlo do zadaného počtu krokov. Celkový výpočtový čas je posledná informácia.



Obr. 14. Protokol riešenia úlohy

4 MOŽNOSTI VYUŽITIA VOĽNE DOSTUPNÉHO SOFTWARU MINIZINC

MiniZinc je jazyk určený pre zápis optimalizačných a rozhodovacích úloh s celočíselnými a spojitými premennými. Model MiniZinc neurčuje ako riešiť úlohu, aj keď môže obsahovať formulácie, ktoré slúžia na riadenie solvera. Snahou vývojárov tohto programu je, aby jazyk MiniZinc tvoril štandard pre CP. MiniZinc je vytvorený ako rozhranie pre rozličné solvery. To sa uskutočňuje pomocou transformácie vstupného MiniZinc modelu a dát do modelu FlatZinc. FlatZinc je nízkoúrovňový jazyk, ktorý slúži ako vstup pre solvery. FlatZinc modely obsahujú deklaráciu premenných a definíciu podmienok, taktiež aj definíciu optimalizačnej funkcie. Preklad z MiniZinc do FlatZinc je špecifický pre rôzne solvery, ktoré môžu určovať konečnú formu podmienok.

Špecifikácia úlohy v jazyku Minizinc má dve časti: model, ktorý opisuje štruktúru triedy úloh a dáta, ktoré špecifikujú jednu osobitnú úlohu v tejto triede. Model a dáta môžu byť v rozličných súboroch.

4.1 Príklad MiniZinc modelu

Každý MiniZinc model je sekvencia položiek, ktoré sa môžu objaviť v ľubovoľnom poradí. Uvedieme príklad MiniZinc modelu a dát pre Job-shop scheduling problem, ktorý nám pomôže pochopiť základné princípy jazyka (obr. 15).

Riadok 0 je komentár začínajúci znakom **%**.

Riadky 1-5 sú deklarácie premenných. Riadok 1 deklaruje **size** ako celočíselný parameter, teda fixnú premennú v modeli. Fixnej premennej môžeme priradiť hodnotu len raz, a to buď v dátovom súbore, alebo priamo v modeli ako v riadku 3. Riadok 4 deklaruje **s** ako 2-rozmerné pole rozhodovacích premenných, ktoré môžu nadobúdať hodnoty v intervale od 0 do **total**. Na riadku 5 je deklarovaná celočíselná premenná s obmedzeným rozsahom. Rozhodovacie premenné sa vyznačujú predponou *var*.

Riadky 7-8 obsahujú používateľom definovaný predikát *predicate no_overlap*, ktorý zaistí, že dve operácie dané začiatočným časom a trvaním sa neprekrývajú v čase.

Riadky 10-17 predstavujú podmienku, ktorá používa vstavaný *forall* cyklus pre všetky úlohy a obmedzenie, že (riadok 12) operácie jednej úlohy budú naplánované v zadanom poradí, (riadok 13) úlohy skončia pred časom trvania celej zákazky **end**, a (riadky 14-16) že žiadne dve operácie na rovnakom stroji sa neprekrývajú v čase.

Riadok 19 obsahuje príkaz *solve*, ktorým sa volá solver. Každý model ho musí obsahovať práve raz. V tomto prípade nám slúži na minimalizovanie času trvania celej

zákazky. Môžeme taktiež maximalizovať účelovú funkciu alebo hľadať hocijaké prípustné riešenie (solve satisfy).

```
0   % (square) job shop scheduling in MiniZinc
1   int: size;                                  % size of problem
2   array [1..size,1..size] of int: d;          % task durations
3   int: total = sum(i,j in 1..size) (d[i,j]);  % total duration
4   array [1..size,1..size] of var 0..total: s; % start times
5   var 0..total: end;                          % total end time
6
7   predicate no_overlap(var int:s1, int:d1, var int:s2, int:d2) =
8       s1 + d1 <= s2 \/ s2 + d2 <= s1;
9
10  constraint
11      forall(i in 1..size) (
12          forall(j in 1..size-1) (s[i,j] + d[i,j] <= s[i,j+1]) /\
13          s[i,size] + d[i,size] <= end /\
14          forall(j,k in 1..size where j < k) (
15              no_overlap(s[j,i], d[j,i], s[k,i], d[k,i])
16          )
17      );
18
19  solve minimize end;
```

Zdroj: [4]

Obr. 15. Príklad MiniZinc modelu

MiniZinc obsahuje mnoho ďalších funkcií, kľúčových slov a operátory, ktoré neboli spomenuté, ako aj globálne podmienky v knižniciach.

4.2 Možnosti použitia

G12 MiniZinc distribúcia obsahuje spustiteľné súbory pre G12 MiniZinc-to-FlatZinc konvertor a G12 FlatZinc prekladač. Taktiež obsahuje zdrojové kódy pre G12 MiniZinc-to-FlatZinc konvertor, parser pre FlatZinc a mnoho príkladov modelov a dokumentáciu. Táto distribúcia je licencovaná pod jednoduchou BSD licenciou.

MiniZinc je podporovaný mnohými CP systémami ako Gecode, Eclipse Prolog, Sicstus Prolog, JaCoP, solvermi G12 skupiny. Tieto solvery môžu urýchliť výpočet. Preto samotné práva používania závisia od týchto systémov.

4.3 VAP model

VAP úloha v modelovacom jazyku MiniZinc je na obr. 16. Riadky 1-9 obsahujú deklarácie parametrov, ktorým budú priradené hodnoty z dátového súboru. Riadky 11-14 obsahujú deklaráciu rozhodovacích premenných, ktorým hodnoty sa v priebehu výpočtu menia. Riadok 16 nám určuje účelovú funkciu, kde sa snažíme minimalizovať evakuačný čas. Riadky 18-22 sú podmienky, ktoré nám vypočítajú počet otočení vozidiel a evakuačné kapacity pre konkrétny evakuačný čas v priebehu výpočtu. Vzorec na výpočet počtov otočení vozidiel sme rozdelili, pretože funkcia floor nemôže mať ako argument premennú *cas*. Týmto však môže dôjsť ku strate presnosti výpočtu, a preto najlepšie nájdené riešenie bude iba horným odhadom optimálneho riešenia. Riadky 24-26 sú podmienky, ktoré nám zabezpečia evakuáciu všetkých obyvateľov. Riadky 28-30 sú podmienky, ktoré zaručia, že neprekročíme kapacitu parku. Riadky 32-34 slúžia na výpis riešenia.

```
1       int : pz;                          %pocet zdrojov
2       int : pk;                          %pocet komunit
3       set of int: I = 1..pz;
4       set of int: J = 1..pk;
5       array[I] of int: K;                %kapacity zdrojov
6       array[I] of int: N;                %vozidla zdrojov
7       array[J] of int: B;                %obyvatelia komunit
8       array[I,J] of float: T;            %casy zo zdrojov ku komunitam
9       array[J] of float: S;              %casy z komunit k utociskam
10
11      array[I,J] of var int: P;          %pocet otoceni vozidla
12      array[I,J] of var int: A;          %evakuacne kapacity
13      array[I,J] of var 0..max(N): X;    %priradene vozidla
14      var int: cas;                      %evakuacny cas
15
16      solve minimize cas;
17
18      constraint forall (i in I, j in J) (
19               P[i,j] = (cas div floor(2.0*S[j]) +
20                        floor((-T[i,j] + S[j] ) / (2.0*S[j]))) /\
21               A[i,j]=K[i]*P[i,j]
22        );
23
24      constraint      forall (j in J) (
25           sum(i in I) (X[i,j]*A[i,j]) >= B[j]
26        );
27
28      constraint      forall(i in I) (
29           sum(j in J) (X[i,j]) <= N[i]
30        );
31
32      output [ "Cas :"++show(cas)++"\n"]++
33              [show_int(3,X[i,j]) ++  if j == pk then "\n"
34                                      else " " endif | i in I, j in J ];
```

Obr. 16. MiniZinc model úlohy VAP

Dátový súbor príkladu, ktorý bol použitý na testovanie a nezodpovedá reálnej situácii v cestnej sieti vyzerá nasledovne:

```
pz=3;
pk=4;
S= [ 8.03216036, 6.99898758666667, 8.99845561466667, 9.99898758666667];
B= [ 1209, 504, 62, 105];
N= [ 4, 8, 8];
K= [ 20, 22, 25];
T= [|4.17654590866666, 2.17654590866666, 5.17654590866666, 4.17654590866666|
7.17654590866666, 5.17654590866666, 3.17654590866666, 9.37897622|
8.79317418, 6.84937994666667, 7.155174036, 9.45895614966666|];|
```

Obr. 17. MiniZinc data úlohy VAP

Výpočet modelu úlohy VAP pre konkrétne zadanie nášho testovacieho príkladu zahájime nasledovne. S G12 MiniZinc distribúciou sa spúšťa príkazom **mzn-g12fd Vap.mzn Simple.dzn**, kde fd znamená, že sa použije finite domain solver. Vap.mzn je súbor obsahujúci MiniZinc model opísaný na obr. 16. Simple.dzn je súbor obsahujúci dátovú časť z obr. 17. A príznak –a nám určuje, že sa vypíšu všetky riešenia. Na obr. 18 sa nachádza výstup riešenia danej úlohy. Dvojitou čiarou je ukončení výpis všetkých možných riešení. Opäť čas zodpovedá evakuačnému času v minútach, riadky zodpovedajú parkom a stĺpce komunitám, a číslo znamená počet vozidiel priradených komunite z parku.

```
C:\miniVap>mzn-g12fd vap.mzn simple.dzn -a
Cas :143
  0   0   0   0
  7   0   0   1
  0   2   1   0
----------
Cas :128
  0   0   0   0
  7   0   0   1
  0   3   1   0
----------
Cas :112
  0   0   0   0
  7   0   0   1
  1   3   1   0
----------
Cas :96
  0   0   0   0
  7   0   0   1
  3   3   1   0
----------
Cas :95
  1   0   0   0
  5   1   1   1
  6   2   0   0
----------
Cas :91
  2   0   0   0
  4   1   1   2
  6   2   0   0
----------
Cas :80
  4   0   0   0
  1   3   1   2
  7   1   0   0
----------
==========
```

Obr. 18. MiniZinc riešenie úlohy VAP

5 POPIS TESTOVACÍCH PRÍKLADOV

Ako vstupné údaje boli použité reálne úlohy, ktorých zadania čiastočne vychádzali z reálnej situácie v cestnej sieti. Časy t_{ij} a s_j rešpektujú existujúcu cestnú sieť Slovenskej republiky a hodnoty b_j reprezentujú počty obyvateľov existujúcich obcí. Hodnoty N_i a K_i však nevychádzajú z reálnej situácie a sú určené odhadom.

Úlohy sú označené číslom od 1 po 19 a 20. úloha je označená ako „Hradza“ a modeluje mimoriadnu situáciu, ktorá by nastala v prípade záplavovej vlny spôsobenej vyliatím sa hrádze Liptovská Mara a ktorej celkové zadanie (vrátane koeficientov N_i a K_i) rešpektuje reálnu situáciu v cestnej sieti. Zdroj údajov je z práce [1].

Vstupné dáta úloh sú uložené v nasledovných súboroch:

1. Súbor *N.txt* obsahuje počet vozidiel N_i, ktoré sú k dispozícii v homogénnom parku $i \in I$.
2. Súbor *K.txt* obsahuje kapacitu vozidiel K_i umiestnených v parku $i \in I$.
3. Súbor *B.txt* obsahuje počet obyvateľov b_j komunity $j \in J$.
4. Súbor *Sj.txt* obsahuje časy presunu s_j v minútach z komunity $j \in J$ do jej prideleného útočiska.
5. Súbor *Tij.txt* obsahuje časy presunu t_{ij} v minútach z parku $i \in I$ do komunity $j \in J$.
6. Súbor *1IdxI_.txt* obsahuje index nehomogenného dopravného parku, ku ktorému patrí homogenný dopravný park.
7. Súbor *1IdxJ_.txt* obsahuje index ohrozenej obce, ku ktorej komunita patrí.

Formát súborov je nasledovný:

1. Súbory *N.txt*, *K.txt*, *B.txt* a *Sj.txt* obsahujú ako prvé číslo počet nasledujúcich údajov a na každom ďalšom riadku je jeden údaj.
2. Súbor *Tij.txt* obsahuje ako prvé číslo počet parkov, ako druhé číslo počet komunít a na každom ďalšom riadku je jeden údaj. Údaje sú v poradí: najskôr údaje z prvého parku do všetkých komunít, potom údaje z druhého parku do všetkých komunít atď.

Vstupné dáta pre úlohy 1-19 sú taktiež vytvorené nástrojom a sú uložené v binárnom súbore. Opis formátu súboru nie je potrebný, nakoľko jeho externé vytvorenie mimo nástroja nemá praktický význam.

Každá úloha sa nachádza v samostatnom adresári, ktoré sa nachádzajú v CD prílohe v adresári „.\Zadania\TestovacieUlohy”.

6 VÝPOČTOVÉ EXPERIMENTY

Výpočtové experimenty boli vykonané pre úlohy opísané v predchádzajúcej kapitole. Experimenty pre rôzne stratégie heuristík a pre CP boli vykonané na osobnom počítači vybavenom procesorom Intel Core2 Duo 2Ghz a 2 GB RAM. Výsledky experimentov, ktoré boli prebraté, a to konkrétne v nástroji Xpress a výsledky metódy vetiev a hraníc, boli vykonané na osobnom počítači vybavenom procesorom AMD Sempron 1,8Ghz a 512 MB RAM. Pri porovnávaní výpočtových časoch dochádza teda k určitej miere skreslenia, ktorá však nebude natoľko veľká, aby sme nemohli dôjsť k záveru a porovnaniu metód.

6.1 Heuristiky

Tabuľky pre jednotlivé experimenty obsahujú stĺpce pre jednotlivé úlohy 1-20. Prvý riadok je číslo alebo označenie úlohy. Riadok „*I*“ obsahuje počet homogénnych parkov úlohy. Riadok „*J*“ obsahuje počet komunít úlohy.

Výsledky výpočtových experimentov pre heuristiky, teda horný odhad času evakuácie T^{UB}, sú zaznamenané v tabuľkách 1 a 2. Tab. 1 obsahuje evakuačné časy najlepšieho riešenia v minútach získané stratégiami S1, S2min, S2max, S3min a S3max pre jednotlivé úlohy.

Pre porovnanie sa v tabuľke nachádzajú časy získané pomocou nástroja Xpress. Na výpočet úloh pomocou nástroja Xpress bol použitý dichotomický algoritmus a iteratívny postup pre úlohu RVAP. Počiatočná dolná hranica je určená ako najvyšší čas $T \in Z_0^+$ taký, kde pre čas $T - 1$ neexistuje ani riešenie LP relaxovanej RVAP. Keďže množina celočíselných riešení je podmnožinou množiny neceločíselných riešení, neexistencia neceločíselného riešenia zaručuje aj neexistenciu celočíselného riešenia. Čas T je nájdený rýchlo pomocou bisekcie. Na riešenie LP relaxovanej RVAP je použitý algoritmus vyvinutý na Katedre dopravných sietí. Počiatočná horná hranica VAP je určená nasledovne. Po určení počiatočnej dolnej hranice je dorátaná celá úloha, avšak pre nízku hodnotu výpočtového času (0,3 sekundy), po ktorom je ukončovaná metóda vetiev a hraníc pri riešení RVAP pre čas *Tmax*. Po vyriešení úlohy pre tento nízky výpočtový čas tvorí získaný výsledný čas evakuácie počiatočnú hornú hranicu VAP. Pre ďalší postup bol použitý dichotomický algoritmus.

Riadok *Pot. XPRESS* obsahuje čas potenciálne optimálneho riešenia úlohy v minútach. Potenciálne optimálne riešenie úlohy je riešenie, ktoré by mohlo byť (resp. bolo) získané pre čas evakuácie, ktorý je o 1 minútu väčší ako je najväčší získaný čas, pre ktorý neexistuje celočíselné prípustné riešenie. Riadok *Naj. XPRESS* obsahuje najmenší čas evakuácie v minútach získaný pomocou nástroja Xpress. Riadok *Gap [min]* vyjadruje rozdiel času heuristiky a času *Naj. XPRESS*. Riadok *Gap [%]* vyjadruje koľko percent tvorí *Gap [min]* z hodnoty času z riadka *Naj.XPRESS*. Pre sprehľadnenie stĺpec *Avg* obsahuje priemer riadku *Gap [min] a riadku Gap [%]*.

Úloha	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
|I|	**10**	**7**	**11**	**7**	**9**	**9**	**9**	**10**	**10**	**28**	**12**
|J|	**9**	**13**	**20**	**16**	**9**	**17**	**17**	**16**	**17**	**55**	**22**
Pot. XPRESS [min]	396	277	67	125	61	87	141	204	88	117	119
Naj. XPRESS [min]	396	277	67	126	61	89	143	204	90	123	121
S1[min]	437	291	77	147	69	104	165	229	117	138	132
Gap [min]	41	14	10	21	8	15	22	25	27	15	11
Gap [%]	10,35	5,05	14,93	16,67	13,11	16,85	15,38	12,25	30,00	12,20	9,09
S2min[min]	451	291	75	147	68	104	163	231	108	138	138
Gap [min]	55	14	8	21	7	15	20	27	18	15	17
Gap [%]	13,89	5,05	11,94	16,67	11,48	16,85	13,99	13,24	20,00	12,20	14,05
S2max[min]	456	312	78	147	74	103	161	212	106	134	133
GAP [min]	60	35	11	21	13	14	18	8	16	11	12
Gap [%]	15,15	12,64	16,42	16,67	21,31	15,73	12,59	3,92	17,78	8,94	9,92
S3min[min]	403	280	70	135	75	97	151	210	117	126	130
GAP [min]	7	3	3	9	14	8	8	6	27	3	9
Gap [%]	1,77	1,08	4,48	7,14	22,95	8,99	5,59	2,94	30,00	2,44	7,44
S3max[min]	441	280	69	138	69	97	150	210	102	127	130
GAP [min]	45	3	2	12	8	8	7	6	12	4	9
Gap [%]	11,36	1,08	2,99	9,52	13,11	8,99	4,90	2,94	13,33	3,25	7,44

Úloha	12	13	14	15	16	17	18	19	Hradza	Avg
|I|	16	14	19	25	10	18	42	4	16	
|J|	33	20	24	37	24	14	32	15	26	
Pot. XPRESS [min]	155	117	192	430	196	409	259	60	81	-
Naj. XPRESS [min]	158	120	195	430	200	409	265	60	81	-
S1 [min]	179	132	210	473	240	416	290	68	82	-
Gap [min]	21	12	15	43	40	7	25	8	1	19,05
Gap [%]	13,29	10,00	7,69	10,00	20,00	1,71	9,43	13,33	1,23	12,13
S2min [min]	178	130	211	467	240	409	286	65	133	-
Gap [min]	20	10	16	37	40	0	21	5	52	20,9
Gap [%]	12,66	8,33	8,21	8,60	20,00	0,00	7,92	8,33	64,20	14,38
S2max[min]	180	133	214	457	242	416	293	68	133	-
Gap [min]	22	13	19	27	42	7	28	8	52	21,85
Gap [%]	13,92	10,83	9,74	6,28	21,00	1,71	10,57	13,33	64,20	15,13
S3min [min]	158	123	195	433	214	409	266	63	82	-
Gap [min]	0	3	0	3	14	0	1	3	1	6,1
Gap [%]	0,00	2,50	0,00	0,70	7,00	0,00	0,38	5,00	1,23	5,58
S3max [min]	165	125	205	456	215	416	271	68	102	-
Gap [min]	7	5	10	26	15	7	6	8	21	11,05
Gap [%]	4,43	4,17	5,13	6,05	7,50	1,71	2,26	13,33	25,93	7,47

Tabuľka 1. Výsledky heuristík

Tab. 2 obsahuje výpočtové časy potrebné pre nájdenie riešenia. Riadok *XPRESS* obsahuje výpočtový čas vo formáte [HH:MM:SS]. Riadky *S1, S2min, S2max, S3min* a *S3max* obsahujú výpočtový čas v milisekundách.

Úloha	1	2	3	4	5	6	7	8	9	10	11
|I|	10	7	11	7	9	9	9	10	10	28	12
|J|	9	13	20	16	9	17	17	16	17	55	22
XPRESS	1:09:59	0:21:56	0:44:28	0:44:35	0:09:30	1:00:07	1:40:03	1:40:02	0:23:08	1:34:57	1:31:05
S1 [ms]	7	2	1	1	0	1	1	2	1	14	2
S2min [ms]	8	2	1	2	0	2	2	2	1	12	2
S2max [ms]	8	3	6	2	0	1	2	2	1	10	2
S3min [ms]	10	3	2	2	0	1	3	5	2	19	3
S3max[ms]	10	3	1	1	1	1	3	4	2	19	3

Úloha	12	13	14	15	16	17	18	19	Hradza
|I|	16	14	19	25	10	18	42	4	16
|J|	33	20	24	37	24	14	32	15	26
XPRESS	1:31:02	1:00:11	2:00:25	1:26:54	1:08:52	-	1:44:56	0:20:01	0:00:01
S1 [ms]	7	3	8	32	6	10	33	0	1
S2min [ms]	5	2	6	29	3	7	26	0	3
S2max [ms]	5	2	6	25	3	8	23	0	3
S3min[ms]	9	4	12	55	6	17	56	0	3
S3max [ms]	7	3	9	38	4	12	43	0	3

Tabuľka 2. Výpočtové časy heuristík

Zhodnotenie. Z rôznych vyskúšaných stratégií sa k optimálnemu riešeniu najviac priblížila stratégia S3min, kde bol rozdiel 0 až 27 minút v testovacích úlohách. Priemerne vznikol rozdiel 6 minút (5,58 %) oproti najlepšiemu evakuačnému času získanému v nástroji Xpress. Rozdiely, z hľadiska výpočtového času, sú zanedbateľné, a preto sa nebrali do úvahy pri hodnotení stratégii. Pre tieto všetky spomenuté hľadiská sme stratégiu S3min využili aj pri výpočte hornej hranice T^{UB} v metóde CP.

6.2 Metóda CP

Tabuľky pre jednotlivé experimenty metódy CP obsahujú riadky pre jednotlivé úlohy 1-20. Prvý stĺpec je číslo alebo označenie úlohy. Stĺpec „*I*“ obsahuje počet homogénnych parkov úlohy. Stĺpec „*J*“ obsahuje počet komunít úlohy.

Nasledujú výsledky výpočtových experimentov pre metódu CP a ich porovnanie s metódu branch and bound (metóda vetiev a hraníc). Hodnoty pre metódu vetiev a hraníc sme získali z práce [9].

V tab. 3 sú evakuačné časy získané riešením matematického modelu v nástroji Xpress (stĺpec *Xpress* totožný s riadkom *Xpress* z tab. 1), pomocou metódy vetiev a hraníc (stĺpec *Branch and bound*) a pomocou metódy CP (stĺpce *CP*), kde *S3min* obsahuje čas získaný pomocou heuristiky S3min, *Best* je najlepší čas získaný pomocou metódy CP. Stĺpec *Repeat* obsahuje počet spustení algoritmu Branch and prune, čo vyjadruje aj rozdiel stĺpcov Best – S3min + 1, nakoľko čas evakuácie sa každej iterácii znížil o 1 z počiatočnej hodnoty vypočítanej heuristikou.

Stĺpce *Gap* vyjadrujú rozdiel evakuačného času vypočítaného pomocou metódy Branch and Bound, resp. CP a najlepšieho času pomocou Xpressu. Riadok *Avg* obsahuje aritmetický priemer zodpovedajúcich stĺpcov.

			Xpress	Branch and Bound	CP			Gap	
Úloha	**|I|**	**|J|**	**Best [min]**	**Best [min]**	**S3min [min]**	**Best [min]**	**Repeat**	**BaB**	**CP**
1	**10**	**9**	396	403	403	403	1	7	7
2	**7**	**13**	277	278	280	278	3	1	1
3	**11**	**20**	67	70	70	69	2	3	2
4	**7**	**16**	126	129	135	130	6	3	4
5	**9**	**9**	61	61	75	68	8	0	7
6	**9**	**17**	89	95	97	95	3	6	6
7	**9**	**17**	143	150	151	150	2	7	7
8	**10**	**16**	204	210	210	208	3	6	4
9	**10**	**17**	90	92	117	114	4	2	24
10	**28**	**55**	123	129	126	126	1	6	3
11	**12**	**22**	121	127	130	130	1	6	9
12	**16**	**33**	158	170	158	158	1	12	0
13	**14**	**20**	120	120	123	123	1	0	3
14	**19**	**24**	195	205	195	195	1	10	0
15	**25**	**37**	430	463	433	433	1	33	3
16	**10**	**24**	200	218	214	208	7	18	8
17	**18**	**14**	409	409	409	409	1	0	0
18	**42**	**32**	265	289	266	266	1	24	1
19	**4**	**15**	60	61	63	61	3	1	1
Hradza	**16**	**26**	81	81	82	82	1	0	1
Avg								7,250	4,550

Tabuľka 3. Výsledky CP a BaB

Tab. 4 obsahuje výpočtové časy pre jednotlivé metódy. Stĺpce *Xpress* a *Branch and Bound* obsahujú výpočtové časy v nástroji Xpress a pre metódu vetiev a hraníc vo formáte [HH:MM:SS]. Ak bol čas výpočtu príliš krátky, je uvedený znak pomlčka (-). Stĺpec *CP Best* obsahuje výpočtový čas najlepšieho prípustného riešenia v sekundách a stĺpec *CP Total* obsahuje celkový výpočtový čas, čo zahŕňa aj výpočtový čas algoritmu pre evakuačný čas, pre ktorý nebolo nájdené žiadne prípustné riešenie. Riadok *Avg* obsahuje aritmetický priemer zodpovedajúcich riadkov.

Úloha	|I|	|J|	Xpress	Branch and Bound	CP Best [sek]	CP Total [sek]
1	**10**	**9**	1:09:59	1:10:10	0,010	1,694
2	**7**	**13**	0:21:56	0:27:39	0,006	1,882
3	**11**	**20**	0:44:28	0:44:16	0,003	1,954
4	**7**	**16**	0:44:35	0:44:09	1,888	3,440
5	**9**	**9**	0:09:30	0:09:30	0,002	1,299
6	**9**	**17**	1:00:07	1:20:02	0,003	1,742
7	**9**	**17**	1:40:03	1:40:02	0,004	1,759
8	**10**	**16**	1:40:02	1:40:01	0,008	2,285
9	**10**	**17**	0:23:08	0:23:04	0,003	1,904
10	**28**	**55**	1:34:57	1:26:07	0,029	4,357
11	**12**	**22**	1:31:05	1:10:52	0,006	2,580
12	**16**	**33**	1:31:02	1:30:26	0,012	2,243
13	**14**	**20**	1:00:11	1:00:03	0,006	2,244
14	**19**	**24**	2:00:25	2:00:05	0,019	5,460
15	**25**	**37**	1:26:54	1:25:17	0,086	8,537
16	**10**	**24**	1:08:52	1:44:24	0,012	1,461
17	**18**	**14**	-	-	0,025	7,565
18	**42**	**32**	1:44:56	1:40:11	0,088	17,350
19	**4**	**15**	0:20:01	0:20:01	0,002	1,648
Hradza	**16**	**26**	0:00:01	-	0,001	3,229
Avg			1:00:37	0:58:49	0,111	3,732

Tabuľka 4. Výpočtové časy CP a BaB

Zhodnotenie. Obidve metódy, a to metódu CP, ktorá je predmetom tejto práce a metódu vetiev a hraníc, sme na týchto úlohách porovnali s profesionálnym nástrojom Xpress, aby sme zistili, ktorá metóda nám prinesie lepšie, a teda bližšie, riešenia k známemu riešeniu v nástroji Xpress. Metóda vetiev a hraníc sa odklonila o 0 až 33 minút a metóda CP o 0 až 24 minút. Priemerne metóda CP ukázala rozdiel 4,55 minút, čo je menej ako metóda vetiev a hraníc – 7,25 minút. Z 20 prípadov bola len 7-krát metóda CP menej úspešná v hľadaní riešenia.

Čo sa týka časového hľadiska, bola opäť metóda CP úspešnejšia, a to mnohonásobne, pretože nájdenie najlepšieho, hoci nie optimálneho riešenia, trvalo vždy len niekoľko milisekúnd. Celkový výpočtový čas, teda aj s časom nenájdeného riešenia v čase o 1 menšom ako je najlepšie riešenie, trval priemerne 3,73 sekúnd. Metóda vetiev a hraníc trvala od niekoľkých minút až po maximálne 2 hodiny. Nesmieme tiež zabúdať, že

experimenty prebiehali na rozdielnych počítačoch, ale pri takých markantných rozdieloch vo výpočtových časoch sú technické parametre procesorov nepodstatné.

Ani jedna z metód však neprekonala najlepšie nájdené riešenie pomocou nástroja Xpress, čo je možné vysvetliť použitím výkonného solvera, ktorý tento profesionálny nástroj používa.

Zlepšenie. Metóda CP v čase o 1 menšom ako bolo najlepšie nájdené riešenie nenašla riešenie v počte krokov 1 000 000. Pre väčší počet krokov nepostačovali pamäťové prostriedky počítača. V niektorých prípadoch však neviedlo k zlepšeniu ani zvýšenie počtu krokov, a preto bola snaha vytvoriť obmenu algoritmu, ktorý by rýchlejšie odhalil nekonzistentné priradenie podľa princípu fail-first, čo sa ale ukázalo ako zhoršujúce. Preto sme ponechali pôvodný algoritmus, ktorý nám síce nevráti optimálne riešenie, ale rýchlo vedie k nájdeniu veľmi dobrého riešenia. Viac orezať strom riešení, t.j. znížiť počet krokov nie je možné kvôli štruktúre modelu. Podmienky medzi premennými neumožňujú výrazne redukovať domény budúcich premenných po nastavení hodnoty aktuálnej premennej.

6.3 MiniZinc

Pre experimenty boli vytvorené dátové súbory pre MiniZinc z úloh 1-20. Ich riešenie však v žiadnom prípade neskončilo do takého času, aby sme mohli porovnať výsledky. Tento výpočet by sa dal ešte skrátiť vhodnými nastaveniami solvera, alebo využitím nepreskúmaných možností nástroja, ktoré by však vyžadovali ďalšie štúdium a experimentovanie, pričom nemusia priniesť očakávané zlepšenie. Preto môžeme skonštatovať, že úloha VAP nie je vhodná na riešenie univerzálnym solverom, a ak áno, tak len za veľmi špecifických podmienok a parametrov, ktoré vyžadujú dobrú znalosť systému.

7 ZÁVER

Evakuačná úloha a špeciálne jej časť prideľovanie vozidiel je nelineárna úloha, ktorá aj po mnohých zjednodušeniach ostáva zložitým problémom.

Cieľom tejto diplomovej práce bolo vytvoriť algoritmus na jej riešenie použitím metódy CP, porovnať ho s algoritmom založeným na metóde vetiev a hraníc a zakomponovať ho do existujúceho nástroja na podporu rozhodovania.

Rozobrali sme si metódu CP a možnosti, ktoré poskytujú programy založené na nej. Pomocou experimentov sme dospeli k záveru, že algoritmus, ktorý sme zostavili pomocou metódy CP, síce nedokáže vypočítať optimálne riešenie kvôli vysokej pamäťovej náročnosti, ale aj ako heuristický algoritmus vykazuje priemerne o niečo lepšie výsledky ako metóda vetiev a hraníc, a to, čo sa týka nájdeného prípustného riešenia v menšom evakuačnom čase, aj výpočtového času. Hlavný prínos je teda mnohonásobne zníženie výpočtového času oproti metóde vetiev a hraníc s veľmi dobrým nájdeným riešením a rýchlo nájdeným prvotným riešením pomocou heuristiky, čo môže výrazne pomôcť pri situáciách, ktoré vyžadujú okamžité a dobré riešenie, akým je aj evakuačná úloha.

CP je metóda, ktorá sa dlhodobo vyvíja a s ňou aj nástroje, ktoré ju využívajú určené pre bežného používateľa bez potreby ovládať nejaký programovací jazyk. Jedným z takýchto nástrojov, ktorý má ambície stať sa štandardom v oblasti CP jazykov je MiniZinc, a preto sme sa naňho pozreli v práci bližšie. Dospeli sme k záveru, že pre širšie a zložitejšie úlohy nie je vhodný na používanie bez potrebnej hlbšej znalosti solvera a ďalších experimentov s jeho parametrami.

8 ZOZNAM POUŽITEJ LITERATÚRY

[1] TOMAN, Ľ. Písomná práca k dizertačnej skúške: *Informatické nástroje a riešenie úloh evakuácie*, vedúci: prof. RNDr. Jaroslav Janáček, CSc. KDS-FRI-ŽU, 2011.

[2] <http://ktilinux.ms.mff.cuni.cz/~bartak/constraints/>, On-line guide to constraint programming by Roman Barták, 30.4.2013

[3] <http://www.fi.muni.cz/~hanka/konstanz09/>, Course on Constraint Programming and Scheduling, 30.4.2013

[4] BESSIERE, CH. 2007. *Principles and practice of constraint programming - CP 2007*. Berlín, Nemecko: Springer, 2007. 890 s. ISSN O302-9743

[5] ROSSI F. - BEEK P. – WALSH T. 2006. *Handbook of Constraint Programming*. Amsterdam, Holandsko: Elsevier, 2006. 978 s. ISSN 1574-6525

[6] <http://www.g12.csse.unimelb.edu.au/minizinc/downloads/doc-1.6/minizinc-tute.pdf>, MiniZinc Tutorial, 30.4.2013

[7] <http://www.cs.ubc.ca/~kevinlb/teaching/cs322%20-%202009-10/Lectures/CSP3.pdf>, CSPs: Arc Consistency, 30.4.2013

[8] APT K. 2003. *Principles of Constraint Programming*. Cambridge: Cambridge University Press, 2003. 407 s. ISBN 0-511-05616-8

[9] TOMAN, Ľ. Acceleration of evacuation problem solving by branch and bound method with assistance of rapid excluding of branches In: *Mathematical methods in economics 2011 : proceedings of the 29th international conference* : September 6-9.2011, Janská Dolina, Slovakia. - Praha: Professional Publishing, 2011. pp. 715-720.

[10] TOMAN, Ľ. *Exaktná metóda riešenia evakuačnej úlohy ako súčasť nástroja na podporu rozhodovania v krízovej situácii* : Diplomová práca. Žilina : Žilinská univerzita v Žiline, 2010. 102 s.

[11] JANÁČEK, J. *Handling of a non-linear model of the evacuation plan design by IP-solver*. In *Proceedings of the 10th international Symposium on OPERATIONAL RESEARCH*. Nova Gorica, Slovenia : Slovenian Society Informatika, 2009. ISBN 978-961-6165-30-3, s. 279-287.

[12] <http://en.wikipedia.org/wiki/Constraint_logic_programming>, 30.4.2013

[13] <http://en.wikipedia.org/wiki/Consistency>, 30.4.2013

[14] <http://en.wikipedia.org/wiki/Constraint_programming>, 30.4.2013

[15] Acton Q. 2011. *Issues in Artificial Intelligence, Robotics and Machine Learning*. Atlanta, Georgia: ScholarlyEdition, 2011. ISBN 978-1-464-96470-1

[16] <http://www.hakank.org/minizinc/>, My MiniZinc page created by Hakan Kjellerstrand, 30.4.2013

9 ZOZNAM SKRATIEK

VAP – Vehicle Assignment Problem

RVAP – Reduced VAP

CP - Constraint Programming

CSP - Constraint Satisfaction Problem

FC – Forward Checking

LA – Look Ahead

ÚF – Účelová Funkcia

UB – Upper Bound

LB – Lower Bound

10 ZOZNAM OBRÁZKOV

11 ZOZNAM TABULIEK

12 ZOZNAM PRÍLOH

Príloha č.1: CD

CD PRÍLOHA

Na CD sa nachádza táto diplomová práca v elektronickej podobe.

Priložené CD obsahuje adresáre:

- Priklady\ - testovacie príklady
- CP\ – zdrojové kódy algoritmu CP v Delphi
- MiniZinc\ – obsahuje model VAP pre MiniZinc a údaje pre testovacie príklady
- Riesenia\ – adresár obsahuje protokoly riešení pre testovacie príklady