Katedra kybernetiky a umelej inteligencie
Fakulta elektrotechniky a informatiky
Technická univerzita v Košiciach

Riešenie úloh rozvrhovania logickým programovaním ohraničení

Doktorandská dizertačná práca

Košice, máj 1997 Ing. Ján Paralič

OBSAH

Zoznam obrázkov

Zoznam tabuliek

Zoznam použitých skratiek a symbolov

CLP (Constraint Logic Programming) logické programovanie ohraničení

CSP (Constraint Satisfaction Problems) úlohy spĺňania ohraničení

FCSP (Finite Constraint Satisfaction Problems) konečné úlohy spĺňania ohraničení

:: operátor v jazyku ECL^iPS^e slúžiaci na stanovenie konečnej domény premennej

znak, ktorý sa používa v jazyku ECL^iPS^e ako prvý znak numerických ohraničení pracujúcich s premennými s konečnými doménami.

#>= relačný operátor "väčší alebo rovný" v jazyku ECL^iPS^e reprezentuje numerické ohraničenie pre premenné s konečnými doménami

disjunctive/3 jednoznačné označenie predikátu v logickom programovaní, obsahuje názov predikátu a počet argumentov

Start(A) premenná reprezentujúca čas začiatku operácie *A*

Koniec(A) premenná reprezentujúca čas ukončenia operácie *A*

start_min(A) najskorší možný čas začiatku operácie *A* (minimum domény premennej *Start(A)*)

koniec_max(A) najpozdnejší možný čas ukončenia operácie *A* (maximum domény premennej *Koniec(A)*, resp. maximálna hodnota domény premennej *Start(A)* zväčšená o dĺžku operácie *A*, t.j. *trvanie(A)*)

trvanie(A) čas vykonávania (dĺžka trvania) operácie *A*

Ω množina operácií zdieľajúcich ten istý zdroj

start_min(Ω*)* najmenší z najskorších časov začiatku operácií z množiny Ω

koniec_max(Ω*)* najväčší z najpozdnejších časov ukončenia operácií z Ω

trvanie(Ω*)* súčet časov vykonávania operácií z Ω

$K_{(A,B)}$ interval úloh (množina operácií *C*, ktoré sa celé zmestia do intervalu <*start_min(A)*, *koniec_max(B)*>, t.j. pre ktoré platí *start_min(A)* $\leq$ *start_min(C)* a *koniec_max(C)* $\leq$ *koniec_max(B)).*

O(f(n)) zložitosť algoritmu vzhľadom na veľkosť vstupu *n*, kde *f(n)* je nejaká matematická funkcia závislá od *n*

$A \prec B$ operácia *A* musí byť vykonaná pred operáciou *B*

$A \succ B$ operácia *A* musí byť vykonaná po operácii *B*

$A \prec \Omega$ operácia *A* musí byť vykonaná pred všetkými operáciami z množiny Ω

$A \succ \Omega$ operácia *A* musí byť vykonaná až po všetkých operáciách z množiny Ω

r_i	čas povolenia výroby výrobku V_i, t.j. čas kedy sa môže začať jeho výroba.
d_i	čas dodávky výrobku V_i, t.j. čas kedy by mal byť V_i hotový.
a_i	dovolená doba výroby výrobku V_i: $a_i = d_i - r_i$
W_{ik}	doba čakania pri výrobe V_i pred operáciou k
W_i	celková doba čakania (prestojov) pri výrobe V_i: $W_i = \sum_{k=1}^{m} W_{ik}$
C_i	skutočný čas ukončenia výroby V_i
F_i	čas ktorý strávi výrobok V_i vo výrobe, $F_i = C_i - r_i$
L_i	oneskorenie, $L_i = C_i - d_i$
T_i	omeškanosť, $T_i = max\{L_i,0\}$
E_i	včasnosť, $E_i = max\{-L_i,0\}$

K a p i t o l a 1

ÚVOD

V dizertačnej práci sa zaoberám riešením úloh časového rozvrhovania, najmä ich kritickej zložky - disjunktných ohraničení - v prostredí logického programovania ohraničení (CLP z anglického Constraint Logic Programming).

Úlohy časového rozvrhovania predstavujú veľmi častú triedu reálnych aplikácií. Cieľom týchto úloh je nájsť podľa určitého kritéria optimálny rozvrh pre realizáciu presne danej skupiny úloh tak, aby boli splnené všetky ohraničenia.

Tieto ohraničenia možno obyčajne rozdeliť do dvoch základných skupín. Prvú skupinu tvoria ohraničenia precedenčné, ktoré stanovujú postupnosti medzi jednotlivými úlohami. Druhú skupinu tvoria ohraničenia disjunktné, ktoré vyplývajú zo zdieľania obmedzeného počtu zdrojov viacerými úlohami, pričom v danom okamžiku môže jeden zdroj slúžiť len jednej úlohe.

Z hľadiska zložitosti úlohy majú rozhodujúci vplyv ohraničenia druhej skupiny, nakoľko každá disjunkcia predstavuje dve alebo viac rovnocenných alternatív, ktoré je nutné v etape riešenia preskúmať. S narastajúcim počtom disjunkcií narastá exponenciálne zložitosť úlohy.

Dizertačná práca sa zaoberá metódami, ako sa vysporiadať s narastajúcou zložitosťou úloh časového rozvrhovania v prostredí CLP. CLP je totiž vďaka svojej deklaratívnosti a flexibilite veľmi vhodným prostredím pre reprezentáciu a riešenie rozvrhovacích úloh. V tejto práci sú popísané a analyzované existujúce a navrhnuté nové postupy a algoritmy vedúce k výraznému zvýšeniu efektívnosti CLP pri riešení úloh časového rozvrhovania.

Dizertačná práca je rozdelená do deviatich kapitol. Prvá kapitola prináša neformálny úvod do problematiky súvisiacej s náplňou dizertačnej práce. Druhá kapitola predstavuje zhrnutie poznatkov súvisiacich s náplňou dizertačnej práce. Najskôr prináša prehľad základných pojmov týkajúcich sa rozvrhovania a CLP. Súčasťou tejto kapitoly je aj prehľad ostatných prístupov k riešeniu rozvrhovacích úloh, stručný popis rôznych prístupov k riešeniu disjunktných ohraničení a optimalizačných postupov používaných v CLP.

Tretia kapitola obsahuje popis cieľov tejto dizertačnej práce, ako aj odkazy, kde v práci je popísané splnenie jednotlivých cieľov. Štvrtá kapitola prináša klasifikáciu a porovnanie ostatných prístupov k riešeniu rozvrhovacích úloh, ako aj vymedzenie postavenia CLP medzi nimi.

Piata kapitola popisuje jednotlivé spôsoby reprezentácie a propagácie disjunktných ohraničení, spôsoby akým som rôzne poznatky o riešení disjunktných ohraničení upravil a spracoval do podoby implementovateľnej v CLP jazyku ECL^iPS^e

(t.j. ako symbolické ohraničenia). V kapitole sú uvedené aj výsledky porovnaní implementovaných metód pre skupinu testovacích úloh.

Testované úlohy samotné sú podrobnejšie popísané v nasledujúcej, šiestej kapitole. Ide jednak o náhodne generované úlohy typu job-shop, ako aj jednu reálnu aplikáciu (návrh časového harmonogramu výstavby päťsegmentového mosta).

Siedma kapitola popisuje prínosy v oblasti riešenia optimalizácie v prostredí CLP.

Poznatky získané v priebehu práce sú zhrnuté v ôsmej kapitole do podoby metodológie riešenia úloh časového rozvrhovania v prostredí CLP. Navrhnutá metodológia bola aplikovaná na riešenie reálnej aplikácie rozvrhovania brám do narážacích pecí v závode VSŽ Oceľ s.r.o., ktorej popis a zhodnotenie výsledkov možno tiež nájsť v tejto kapitole.

Deviata kapitola podáva stručné zhrnutie dosiahnutých výsledkov a načrtáva ďalšie možné perspektívy v študovanej problematike.

1.1 Úlohy rozvrhovania

Cieľom tejto triedy problémov je rozvrhnúť danú množinu aktivít (úloh) pri zohľadnení existujúcich zdrojov a ďalších technologických ohraničení tak, aby výsledný rozvrh spĺňal všetky zadané ohraničenia a prípadne aj aby bol tento rozvrh optimálny z hľadiska nejakého kritéria.

Pri úlohách rozvrhovania je nutné najprv uvažovať charakteristiky daných aktivít a zdrojov, ktoré je potrebné rozvrhnúť. Pritom je možné rozlíšiť tri široké skupiny rozvrhovacích úloh.

1. **Čisté rozvrhovacie úlohy** (napr. job-shop) sú charakteristické tým, že kapacita každého zdroja je definovaná počas veľkého počtu časových intervalov (iným typom sú potom zdroje s obmedzenou kapacitou, napr. rôzne suroviny na sklade) a problém pozostáva z umiestnenia aktivít vyžadujúcich tieto zdroje v čase, bez toho aby sa vyčerpala dostupná kapacita.
2. **Čisté rozmiestňovacie úlohy** (napr. rozmiestňovanie personálu na vlaky alebo lietadlá) zase charakterizuje to, že požiadavka na každý typ zdroja je známa dopredu a problém spočíva v rozmiestnení dostupných zdrojov v čase tak, aby ponuka zdrojov v každom okamihu bola rovná alebo prevyšovala požiadavku na zdroje.
3. **Zmiešané úlohy** sú charakteristické tým, že existuje voľnosť ako pri určovaní toho, ktoré aktivity a kedy majú byť vykonané, tak aj v tom, ktoré zdroje dať k dispozícii a kedy.

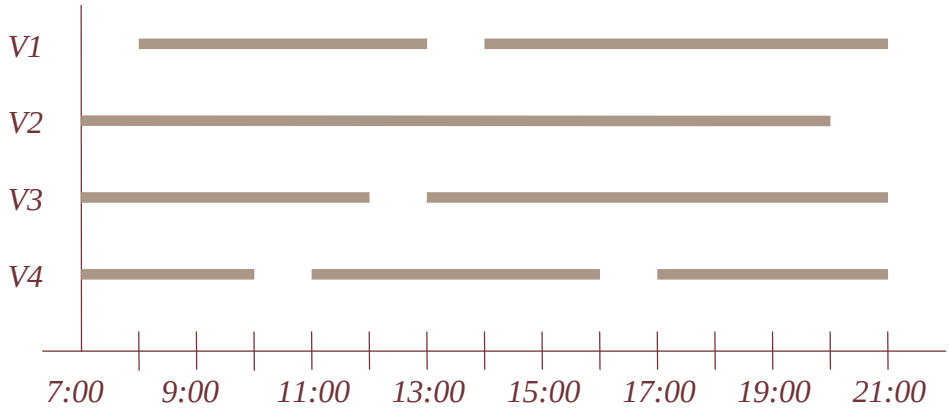
Predmetom tejto dizertačnej práce sú úlohy z prvej skupiny, avšak mnohé postupy a závery je možné viac či menej využiť aj pre ostatné skupiny úloh rozvrhovania. Pre lepšiu názornosť a za účelom poukázania na základné charakteristiky tejto skupiny úloh uvádzam nasledujúci príklad [Tsang 95].

PRÍKLAD 1.

Úlohou je navrhnúť rozvrh výroby pre štyri rôzne výrobky V1 až V4. Každý z týchto výrobkov má presne definované poradie operácií, ktoré sa realizujú na niektorých, alebo na všetkých pracoviskách P1 až P4. Na každom z týchto pracovísk možno spracovávať len jeden výrobok v danom čase a každé z pracovísk je k dispozícii len v určitom časovom intervale. Každý výrobok musí prejsť jednotlivými operáciami (každá operácia na zadanom pracovisku) presne v zadanom poradí a časy trvania jednotlivých operácií pre rôzne výrobky môžu byť rôzne. Spracovávanie výrobku na danom pracovisku nesmie byť prerušené (jedna operácia nesmie byť prerušená inou) a každý výrobok má špecifikovaný najneskorší čas, kedy musí byť hotový. Všetky požiadavky pre túto úlohu sú zhrnuté v tabuľke 1 a dostupnosť pracovísk na obrázku 1.

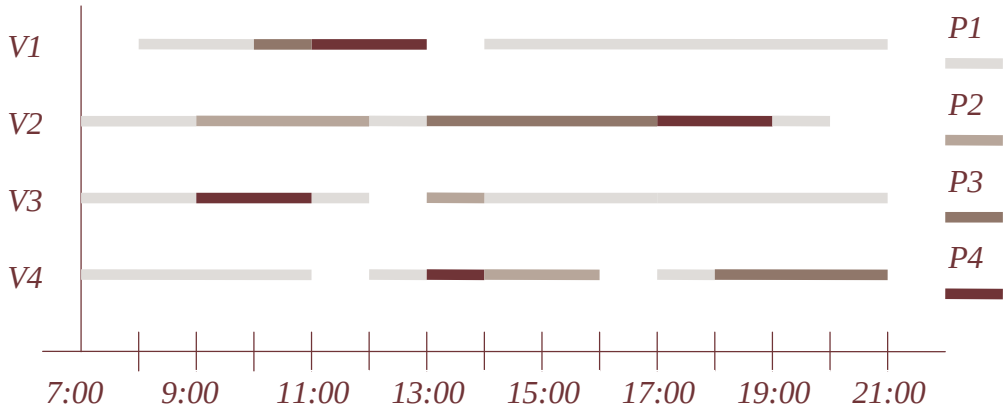
Výrobok	Najskorší čas začiatku výroby	Poradie operácií na jednotlivých pracoviskách a ich trvanie	Najneskorší čas ukončenia výroby
V1	7:00	P1(2),P2(1),P3(3),P4(1)	20:00
V2	9:00	P2(3),P3(1),P4(2)	18:00
V3	9:00	P1(1),P2(4),P4(3)	21:00
V4	8:00	P3(2),P1(2),P4(1),P2(2)	19:00

Tabuľka 1 Výrobné požiadavky.



Obrázok 1 Dostupnosť jednotlivých pracovísk.

Na obrázku 2 je ukážka rozvrhu, ktorý spĺňa všetky ohraničenia dané v tomto príklade. Avšak mnohé úlohy vyžadujú optimalizáciu, t.j. nájdenie takého rozvrhu, ktorý minimalizuje (prípadne maximalizuje) nejaké kritérium. Napríklad nájsť taký rozvrh, v ktorom by boli všetky výrobky hotové tak skoro, ako je to len možné.



Obrázok 2 Ukážka rozvrhu, ktorý spĺňa všetky ohraničenia definované v príklade 1.

1.2 Použitie CLP pre riešenie rozvrhovacích úloh

Logické programovanie ohraničení (CLP) je zhruba desať rokov stará technológia programovania, ktorá vychádza z logického programovania s tým, že podstatne rozširuje jeho schopnosti a efektívnosť zásluhou zovšeobecnenej logickej premennej a algoritmov propagácie a riešenia ohraničení [Paralič 95].

CLP si pritom ponecháva deklaratívny prístup k formulácii úlohy, ktorá je zároveň vykonateľným programom. Je to veľmi vhodný nástroj na riešenie úloh spĺňania ohraničení (CSP - z anglického Constraint Satisfaction Problems). CSP sú definované pomocou množiny premenných a množiny ohraničení medzi týmito premennými. Úlohou je nájsť také priradenie hodnôt premenným, aby boli splnené všetky ohraničenia.

Do skupiny CSP možno zaradiť aj rozvrhovacie úlohy. Riešenie CSP za pomoci CLP sa skladá z troch krokov.

1. **Voľba premenných** a ich počiatočných domén (množín, z ktorých premenné môžu nadobúdať svoje hodnoty).
2. **Stanovenie ohraničení** medzi týmito premennými (definícia podmienok, ktoré musí spĺňať riešenie úlohy).
3. **Odštartovanie (prípadne aj voľba spôsobu) prehľadávania** v priestore prehľadávania, ktorý je dynamicky orezávaný v priebehu riešenia zásluhou algoritmov propagácie ohraničení. Do tohto bodu je možné zahrnúť aj prípadnú požiadavku na optimalizáciu.[1]

Skúsme teraz aplikovať uvedený postup na príklad 1. Ako premenné možno zvoliť časy kedy sa začne spracovávať daný výrobok na danom pracovisku (začiatok vykonávania jednej operácie). Takže pre príklad 1 to znamená celkovo 14 premenných (po 4 pre *V1*, *V4* a po 3 pre *V2*, *V3*).

[1] Optimalizácia je v jazykoch CLP obvykle realizovaná zabudovanými predikátmi, ktoré stačí len vhodne použiť a zastrešiť tak prehľadávanie. Podrobnejšie viď. 2.4.

Nech T_{XY} reprezentuje začiatok spracovávania výrobku V_X na pracovisku P_Y a D_{XY} trvanie tohoto spracovania. Potom jednotlivé typy ohraničení z definície úlohy možno reprezentovať nasledovne.

- Požiadavku najskoršieho času započatia výroby napríklad pre výrobok *V1* možno reprezentovať nerovnicou

$$T_{11} \geq 7.$$

- Požiadavku presného poradia pracovísk pri spracovávaní daného výrobku možno vyjadriť opäť ako nerovnice, kde sa zohľadňuje trvanie jednotlivých operácií. Napr. operácia s časom začiatku T_{11} trvá 2 hodiny a T_{12} môže začať až po skončení T_{11}, takže

$$T_{11} + 2 \leq T_{12}$$

- Aby sa zabezpečilo, že v danom okamžiku môže na jednom pracovisku prebiehať len jedna operácia, musí sa definovať množina dvojíc disjunktných ohraničení, z ktorých zakaždým bude platiť práve jedno. Vo všeobecnosti pre pracovisko P_Y a výrobky V_X a V_{X*} ktoré majú byť na tomto pracovisku spracovávané musí platiť práve jedna z nasledujúcich dvoch nerovníc.

$$\forall\ X* \neq X : T_{XY} + D_{XY} \leq T_{X*Y} \quad \text{alebo}$$

$$T_{X*Y} + D_{X*Y} \leq T_{XY}$$

  Ak platí prvá nerovnica, potom pracovisko P_Y spracováva výrobok V_X pred výrobkom V_{X*}. Naopak ak platí druhá nerovnica, potom pracovisko P_Y spracováva výrobok V_{X*} pred výrobkom V_X.

  Disjunktné ohraničenia sú hlavným zdrojom komplexnosti rozvrhovacích úloh, z ktorých sú väčšina NP-úplné problémy. Preto je v práci venovaná veľká pozornosť práve spôsobu spracovania disjunktných ohraničení.

Metódy a systémy založené na spĺňaní ohraničení umožňujú návrh presných, flexibilných, efektívnych a rozšíriteľných rozvrhovacích aplikácií.

Presné a flexibilné sú tieto aplikácie preto, že môžu brať do úvahy každé ohraničenie, ktoré je možné vyjadriť v danom jazyku spĺňania ohraničení. Zároveň systémy CLP ponúkajú stále širšiu množinu ohraničení (nielen numerické, ale aj rôzne symbolické ohraničenia), ako aj nástroje na definovanie nových ohraničení, podľa potrieb používateľa.

Efektívne sú tieto aplikácie do tej miery, nakoľko efektívne algoritmy šírenia a spĺňania ohraničení sú k dispozícii v tom ktorom systéme.

Rozšíriteľnosť týchto aplikácií je daná tým, že požiadavka na nový typ ohraničenia obvykle vedie iba k definícii tohto nového ohraničenia a prípadne spôsobu jeho propagácie, pričom nie je nutné meniť existujúci program.

Ďalšími dôležitými parametrami, ktoré je nutné brať do úvahy pri riešení úloh rozvrhovania, sú celkový časový interval, pre ktorý sa robí rozvrh a časová presnosť rozvrhu (elementárny časový interval, ktorý sa pri rozvrhu uvažuje). Ak ich vzájomný

pomer je malý (napr. jeden deň v hodinách), je vhodnejšie voliť diskrétnu reprezentáciu času, ktorá dovoľuje priamejší prístup k reprezentovaným údajom. Na druhej strane, ak je tento pomer veľký (napr. jeden mesiac v sekundách), najlepšie je voliť reprezentáciu so spojitou reprezentáciou času.

Sila tejto technológie CLP je v aktívnej úlohe definovaných ohraničení, ktoré zužujú domény premenných na ktoré sú aplikované a tým aj priestor, ktorý v ďalšom bude potrebné prehľadať. Pritom je dôležité si uvedomiť, že takéto pružné prostredie pre vyjadrenie rôznych druhov ohraničení dáva výborný priestor pre prototypovanie a to aj v prípade, keď výsledný systém bude naprogramovaný v inom programovacom jazyku. V tom je výhoda v porovnaní s jednoúčelovými (síce výkonnými ale málo flexibilnými) programovacími prostriedkami.

K a p i t o l a 2

SÚČASNÝ STAV PROBLEMATIKY

V tejto kapitole sú zhrnuté poznatky z oblasti riešenia rozvrhovacích úloh, najmä tie ich aspekty, ktoré boli ďalej v dizertačnej práci rozvinuté. Kapitola je rozčlenená do štyroch častí. Prvá podáva základné definície a popis pojmov súvisiacich s problémovou oblasťou, t.j. úlohy rozvrhovania (konkrétne typu job-shop) a CLP.

Druhá časť poskytuje stručný prehľad metód na riešenie rozvrhovacích úloh, ktoré som sa potom v práci pokúsil porovnať a vymedziť postavenie CLP medzi nimi. Tretia časť sa potom zameriava na existujúce postupy a metódy na riešenie disjunktných ohraničení, ako hlavného zdroja zložitosti úloh. Záverečná časť pojednáva o základných princípoch optimalizácie v CLP.

2.1 Základné pojmy a definície

2.1.1 Úlohy rozvrhovania typu job-shop

Uvediem základné definície a označenia, ktoré budem ďalej používať v súvislosti s rozvrhovacími úlohami typu job-shop (názov z angličtiny).

Terminológia teórie rozvrhovania typu job-shop pochádza z pôvodnej a veľmi častej aplikačnej domény v priemyselnej výrobe. Preto sa tu používajú pojmy ako výrobky a stroje. To však neznamená, že by sa aj mnoho nevýrobných úloh nedalo formulovať ako job-shop, skôr naopak. Nasledujúce definície sú prebrané z [French 82].

Job-shop

Predpokladajme, že máme n **výrobkov** $\{V_1, V_2, \ldots, V_n,\}$, ktoré majú byť vyrobené za pomoci m **strojov** $\{P_1, P_2, \ldots, P_m,\}$. Spracovanie výrobku na danom stroji sa nazýva **operácia**. Operáciu pri výrobe i-teho výrobku na j-tom stroji budeme označovať o_{ij}. **Technologické ohraničenia** požadujú, aby každý výrobok bol spracovávaný na jednotlivých strojoch v presne zadanom poradí[2].

Na vykonanie každej operácie o_{ij} je potrebný určitý, tzv. **výrobný čas**, ktorý sa označuje p_{ij}. Podľa dohody v tomto čase je už zahrnutý aj tzv. **prestavovací čas**, t.j. všetky časové nároky spojené s prípravou stroja na danú operáciu. Hodnoty všetkých výrobných časov p_{ij} sú konštantné a vopred známe. Predpokladá sa, že stroje sú stále k dispozícii, čo však už nemusí platiť o výrobkoch. Každý stroj môže spracovávať v

[2] U úloh typu job-shop má každý výrobok svoju vlastnú postupnosť operácií. Dôležitým špeciálnym prípadom je však tzv. **flow-shop**, kde majú všetky výrobky tú istú postupnosť operácií pri výrobe.

každom okamžiku maximálne jednu operáciu. Operáciu nemožno prerušiť. Pre každý výrobok V_i je známy tzv. **čas povolenia výroby** r_i, t.j. čas, kedy sa môže začať jeho výroba.

Úlohou je nájsť rozvrh (poradie), podľa ktorého majú výrobky prechádzať jednotlivými strojmi a ktorý bude

- spĺňať všetky technologické ohraničenia (tzn. **prípustný** rozvrh),
- **optimálny** s ohľadom na zvolené kritérium.

Kritéria pre výber optimálneho rozvrhu môžu byť rôzne. Na tomto mieste uvediem len tie najčastejšie sa vyskytujúce. Pre ich presnejšie vymedzenie a formalizáciu je potrebné definovať ďalšie pojmy [French 82]:

d_i je **čas dodávky** výrobku V_i, t.j. čas kedy by mal byť V_i hotový.

a_i je **dovolená doba výroby** V_i: $a_i = d_i - r_i$

W_{ik} je **doba čakania** pri výrobe V_i pred operáciou k

W_i je **celková doba čakania** (prestojov) pri výrobe V_i: $W_i = \sum_{k=1}^{m} W_{ik}$

C_i je skutočný **čas ukončenia výroby** V_i

F_i je **čas** ktorý strávi výrobok V_i **vo výrobe**, t.j. $F_i = C_i - r_i$

L_i je **oneskorenie**, t.j. $L_i = C_i - d_i$

T_i je tzv. **omeškanosť**, t.j. $T_i = max\{L_i,0\}$

E_i je **včasnosť**, t.j. $E_i = max\{-L_i,0\}$

X_{max} bude označovať maximálnu hodnotu veličiny X spomedzi všetkých hodnôt X_i (pre jednotlivé výrobky $i = 1$ až n) a $\overline{X}$ zase priemernú hodnotu tejto veličiny vzhľadom na všetky výrobky. S využitím týchto označení možno teraz definovať jednotlivé v praxi najčastejšie používané kritériá optimálnosti a rozdeliť ich do nasledovných skupín:

1. Kritériá založené na časoch ukončenia výroby:
   - minimalizovať maximálny čas pobytu výrobku vo výrobe (F_{max}) je vhodné najmä ak náklady sú priamo úmerné najdlhšie vyrábanému výrobku.
   - minimalizovať maximálny čas ukončenia výroby[3] (C_{max}). C_{max} sa zvykne nazývať aj **celkový čas výroby** a je veľmi častým kritériom, vyjadruje situáciu, keď záleží na celkovej dobe ukončenia výroby všetkých výrobkov.
   - minimalizovať priemerný čas pobytu výrobku vo výrobe ($\overline{F}$).

[3] V prípade že časy povolenia výroby sú 0 pre všetky výrobky, potom $C_{max} = F_{max}$. V opačnom prípade môžu byť výsledky dosť odlišné.

- minimalizovať priemerný čas ukončenia výroby[4] ($\overline{C}$). Posledné dve kritériá sú vhodné, keď sú náklady priamo úmerné priemernej dobe potrebnej na výrobu jedného výrobku.

2. Kritériá založené na časoch dodávok

   - minimalizovať priemernú hodnotu oneskorenia ($\overline{L}$).
   - minimalizovať maximálnu hodnotu oneskorenia (L_{max}). Použitie týchto dvoch kritérií je vhodné najmä vtedy, keď je odmena tým väčšia, čím skôr sa ukončí výroba výrobku.
   - minimalizovať priemernú hodnotu omeškanosti ($\overline{T}$).
   - minimalizovať maximálnu hodnotu omeškanosti (T_{max}). Použitie posledných dvoch kritérií je vhodné najmä vtedy, keď skoré ukončenie výroby neprináša väčšiu odmenu, len za oneskorenú výrobu sú pokuty.
   - niekedy sa používa ako kritérium minimalizovať počet oneskorených výrobkov n_T, t.j. takých u ktorých sa nestihol termín dodávky.

3. Kritéria založené na skladových a spotrebných nákladoch

   - minimalizovať priemerný počet výrobkov čakajúcich na nejaký stroj.
   - minimalizovať počet nedokončených výrobkov. Obe spomínané kritériá sa vzťahujú na medzivýrobné skladové náklady.
   - minimalizovať priemerný počet ukončených výrobkov je dôležité pri znižovaní skladových nákladov pre hotové výrobky.
   - minimalizovať dobu prestojov jednotlivých strojov (či už priemernú dobu, alebo maximálny prestoj).

Ďalšou významnou vlastnosťou optimalizačných kritérií je ich regulárnosť [French 82]. **Regulárne kritérium** R je také, ktoré je neklesajúcou funkciou času ukončenia výroby, t.j. ak

$$C_1 \le C_1', C_2 \le C_2', \ldots, C_n \le C_n' \Rightarrow R(C_1, C_2, \ldots, C_n) \le R(C_1', C_2', \ldots, C_n')$$

t.j. ak máme dva rozvrhy, pričom v prvom z nich je každý výrobok dokončený aspoň tak skoro ako v druhom, potom vzhľadom na nejaké regulárne kritérium je prvý rozvrh aspoň taký dobrý ako druhý.

$\overline{C}, C_{\max}, \overline{F}, F_{\max}, \overline{L}, L_{\max}, \overline{T}, T_{\max}$ a n_T (t.j. všetky z kritérií uvedených v prvých dvoch skupinách) sú regulárne kritériá optimálnosti.

[4] Minimalizovať $\overline{C}$ je ekvivalentné minimalizovaniu $\overline{F}$, t.j. v oboch prípadoch je optimálny ten istý rozvrh.

2.1.2 Logické programovanie ohraničení

V tejto časti možno nájsť stručný popis konceptu CLP, základných metód ktoré používa, ako aj niektorých súčasných systémov. Omnoho podrobnejší prehľad všetkých podstatných aspektov CLP možno nájsť v mojej rigoróznej práci [Paralič 95].

Už názov "logické programovanie ohraničení" sa snaží vyjadriť dve základné zložky tejto technológie. A síce **logické programovanie** a **riešenie ohraničení**.

Logické programovanie (LP) používa matematickú logiku na vyjadrenie problému a dedukciu na jeho riešenie [Lloyd 84]. Výsledkom je deklaratívny programovací jazyk, kde v princípe programátorovi stačí vyjadriť **čo** sa má urobiť bez toho aby sa musel starať o to, **ako** sa má tento cieľ dosiahnuť. V tom sa zásadne líši od klasických procedurálnych programovacích jazykov ako sú *Pascal* alebo *C*. Typickým predstaviteľom skupiny logických programovacích jazykov je *Prolog* [Bratko 86].

To, čo je však z programátorského hľadiska výhodou, ukázalo sa pri riešení reálnych úloh ako určitá prekážka. Logické programovanie totiž narába iba s objektmi ktoré sú neinterpretované štruktúry (t.j. symboly bez vlastného významu) a zhoda medzi nimi sa realizuje jedine syntakticky, a to algoritmom nazývaným unifikácia. Každý sémantický objekt musí byť takto explicitne zakódovaný v podobe termov. Napríklad číselná rovnosť 3 = 1 + 2 bude unifikačným algoritmom zamietnutá, nakoľko ľavá a pravá strana rovnosti sú syntakticky rozdielne (3 je konštanta, zatiaľ čo 1 + 2 je binárna štruktúra +(1,2)), napriek tomu že sa sémanticky zhodujú.

Druhým problémom je práve jednotné, ale pomerne jednoduché výpočtové pravidlo, a síce prehľadávanie do hĺbky, ktorého výsledkom je stratégia **generuj a testuj** s dobre známymi problémami s výkonnosťou pri kombinatoricky náročných úlohách.

Oba tieto problémy pomáha preklenúť integrácia metód na **riešenie (spĺňanie) ohraničení**. Obmedzenia totiž predstavujú vyjadrenie vzťahov medzi sémantickými objektmi z nejakej výpočtovej oblasti (napr. prirodzené alebo reálne čísla). Táto špecifikácia umožňuje využitie efektívnejších algoritmov, takpovediac šitých na mieru tej ktorej výpočtovej domény. Tento koncept teoreticky sformulovali Joxan Jaffar a Jean-Louis Lassez v [JafLas 87]. Jeho základnou myšlienkou je nahradiť unifikáciu ako základnú operáciu LP všeobecnejším procesom riešenia ohraničení na vopred špecifikovanej výpočtovej doméne *X*. Tento koncept, označovaný CLP(*X*), bol následne realizovaný na doméne reálnych čísel v programovacom jazyku nazývanom CLP($\Re$) [JafMich 87], [Heinze a kol. 92].

Avšak v tej istej dobe sa objavili v Európe prologovsky orientované programovacie jazyky ohraničení, ako *CHIP* [Dincbas 88], [AggBel 91] a *Prolog III* [Benhamou 93]. *CHIP* predstavil okrem domény reálnych čísel aj doménu boolovských premenných a tzv. konečné domény (premenné môžu nadobúdať hodnotu z konečnej množiny vopred definovaných hodnôt). *Prolog III* bol podstatne odlišný, jeho doménou boli zoznamy.

Široké aplikačné využitie priniesli so sebou konečné domény a značne ovplyvnili celý nasledujúci vývoj ďalšej generácie CLP jazykov. Najväčším prínosom sú techniky

lokálnej propagácie (šírenia) ohraničení, ktoré pôvodný prístup LP charakterizovaný vyššie ako **generuj a testuj** nahradzujú prístupom **zadaj ohraničenia a generuj**. Pri tomto prístupe sa najskôr spomínaným procesom lokálnej propagácie ohraničení značne zredukujú pôvodné domény premenných. Niekedy je už výsledkom tohoto procesu nájdenie jednoznačného riešenia. Ak nie, potom sa odštartuje zabudovaný mechanizmus prehľadávania, ktorý postupne generuje konkrétne riešenia. Naviac je etapa prehľadávania tiež sprevádzaná propagáciou ohraničení, čo tiež veľmi výrazne zmenšuje prehľadávaný priestor.

Ak si uvedomíme, že mnoho úloh technickej praxe možno formulovať ako CSP, t.j. popísať pomocou množiny premenných, ktorých hodnoty nás zaujímajú a pomocou množiny ohraničení, ktoré musia byť medzi týmito premennými vo výslednom riešení splnené (či už ide o technologické, ekonomické, prípadne iné ohraničenia), možno tušiť, aké široké možnosti využitia tejto technológie sa ponúkajú.

Napriek tomu, že CLP je ešte stále predmetom intenzívneho výskumu [JafMah 94], táto technológia začiatkom deväťdesiatych rokov prekročila hranice výskumných ústavov a v súčasnej dobe existuje vo svete už niekoľko komerčných CLP systémov [Cras 93], ktoré už majú za sebou celý rad úspešných aplikácií v praxi. Prakticky všetky tieto systémy pochádzajú z krajín západnej Európy, najmä z Francúzska, kolísky *Prolog*u. Okrem komerčnej verzie *CHIP*u, ktorý pôvodne vznikol v Mníchovskom ECRC (European Computer-Industry Research Centre) [5], sú to najmä produkty ktoré vznikli transferom tejto technológie do jeho troch podielnických firiem. Ide o systémy *DECISIONPOWER* (anglický ICL), *CHARME* (francúzsky BULL, resp. jeho centrum umelej inteligencie CEDIAG) a *SNI Prolog* (nemecký Siemens-Nixdorf Information Systems) [Cras 93]. Z ďalších systémov je na trhu komerčná verzia *Prolog III*. Celkom iný prístup zvolili tvorcovia *ILOG Solver* [Puget 94], kde algoritmy typické pre CLP sú dodávané vo forme knižnice *C++* funkcií. A samozrejme je tu aj celý rad viac-menej akademických CLP systémov, ako *ECL^{i}PS^e* [MeiBri 95], *CLP(BNR), CAL, clp(FD)* [DiazCod 93] a ďalšie (u niektorých z nich sa už pripravuje ich komerčná verzia).

Pre zápis častí programov v celej dizertačnej práci budem používať syntax jazyka *ECL^{i}PS^e* [Brisset a kol. 94], ktorý je veľmi podobný prologovskej, má však niektoré podstatné rozšírenia pre účely CLP, ktorých význam je vždy na príslušnom mieste v popise programu vysvetlený.

Základné metódy CLP

Na rozdiel od LP je potrebné v CLP naviac v každom kroku zabezpečovať konzistentnosť aktuálnej množiny ohraničení. V zásade sú možné dva prístupy [Mayoh 94].

Prvým z nich je použiť efektívny inkrementálny algoritmus, ktorý je schopný zabezpečiť konzistentnosť riešením ohraničení a ich uchovávaním v nejakej efektívnej podobe (tzv. normálna forma). Jeho inkrementálnosť spočíva v tom, že ak v niektorom

[5] *CHIP V4* je vyvíjaný ďalej francúzskou firmou COSYTEC.

kroku pribudnú nové ohraničenia, algoritmus je schopný zabezpečiť konzistentnosť tejto rozšírenej množiny ohraničení s menšími nákladmi (t.j. napr. bez toho, aby musel prebehnúť celý výpočet odznova). Typickým príkladom je algoritmus Simplex[6], resp. jeho upravené varianty, používané na doméne reálnych (racionálnych) čísel [Lim 94] pre riešenie sústavy lineárnych rovníc a nerovníc. Preto sa táto technológia nazýva **inkrementálne lineárne riešenie**.

Druhou možnosťou je neposkytnúť síce úplné riešenie v každom kroku (čo by bolo časovo veľmi náročné), ale uspokojiť sa s určitým rizikom nekonzistentnosti v priebehu riešenia, profitujúc z neskorších zjednodušení ohraničení (napr. naviazaním premenných v priebehu výpočtu). To je práve prípad techník lokálnej propagácie, používaných najmä pre konečné domény (preto aj jej názov **doménová technológia**), kde je určenie konzistentnosti NP-úplným problémom. Algoritmy na riešenie ohraničení sa tu aktivujú prostredníctvom zmien v doménach premenných, ktorých sa toto ohraničenie týka. Pri tomto prístupe sa najskôr propagáciou ohraničení značne zredukujú pôvodné domény premenných. Už samotná propagácia ohraničení vedie často k takému zredukovaniu domén jednotlivých premenných, ktoré s dostatočnou presnosťou popisujú riešenia úlohy, prípadne nájdu jediné existujúce riešenie.

Vo väčšine prípadov je však potrebné odštartovať prehľadávanie tak, že sa jednotlivým premenným postupne priradzujú hodnoty z ich redukovaných domén. Pritom každé takéto priradenie môže vyvolať ďalšie zmeny v doménach niektorých iných premenných. To vyvoláva opätovnú propagáciu prostredníctvom definovaných ohraničení aj na ostatné premenné, čo vedie v konečnom dôsledku k postupnej redukcii priestoru prehľadávania. Pre tento proces je charakteristická aktívna úloha definovaných ohraničení.

2.2 Prehľad metód na riešenie úloh rozvrhovania

Táto časť je venovaná prehľadu súčasných metód využívaných na riešenie úloh rozvrhovania. Ide jednak o techniky vyvinuté v oblasti operačného výskumu (lineárne programovanie, branch-and-bound, tabu search) a techniky vyvinuté v rámci umelej inteligencie (spĺňanie ohraničení, hill climbing, simulované žíhanie, genetické algoritmy, neurónové siete a expertné systémy).

Možno tu nájsť nielen stručný popis všetkých významných skupín metód ktoré už boli úspešne použité pre riešenie rozvrhovacích úloh, ale aj ich výhody a nevýhody, ktoré je potrebné brať do úvahy pri voľbe vhodnej metódy na riešenie konkrétnej aplikácie. Tieto metódy budú potom v kapitole 4 rozdelené do skupín a na základe zvolených kritérií porovnané.

[6] Aj keď tento algoritmus má v najhoršom prípade exponenciálnu zložitosť, v praxi je s obľubou používaný pre svoju dobrú priemernú zložitosť.

Lineárne programovanie

Lineárne programovanie je použiteľné na riešenie úloh, ktoré sa dajú vyjadriť ako konjunktívna množina lineárnych rovníc alebo nerovníc [Tsang 95]. Úlohou je pritom optimalizovať danú lineárnu funkciu. Aby bolo možné použiť lineárne programovanie, je nutné najprv rozvrhovaciu úlohu reprezentovať nasledujúcim spôsobom.

Postup ilustrujem na príklade 1, ktorý bol definovaný v úvodnej kapitole. Reprezentácia úlohy popísaná v časti 1.2 pre účely riešenia v CLP v zásade vyhovuje aj požiadavkám pre lineárne programovanie. Takže len stručne zrekapitulujem.

Premenné označujú začiatočné časy jednotlivých operácií, ktorých je dokopy 14. T_{XY} označuje začiatočný čas operácie na výrobku V_X vykonávanej na pracovisku Py a D_{XY} je trvanie tejto operácie. Potom platia všetky tri skupiny ohraničení uvedených v časti 1.2. Avšak posledná skupina ohraničení sú disjunktné, a síce pre každú dvojicu operácií zdieľajúcich ten istý zdroj (rovnaké pracovisko) máme disjunktné ohraničenie s dvoma alternatívami, z ktorých len jedna bude platiť vo výslednom rozvrhu.

Tieto disjunktné ohraničenia sa obyčajne spracúvajú vymenovaním všetkých kombinácií nerovníc. Takto dostaneme príslušný počet konjunktívnych množín nerovníc, z ktorých každá sa musí riešiť samostatne.

K úplnosti formulácie úlohy ešte chýba optimalizačná funkcia. Nakoľko táto nebola v zadaní požadovaná (chcem len nájsť rozvrh spĺňajúci všetky zadané ohraničenia), stačí zvoliť nejakú funkciu, napr. $\sum T_{XY}$.

Simplexová metóda a modifikovaná simplexová metóda sú najčastejšie používané algoritmy na riešenie úloh lineárneho programovania. Dôležitou vlastnosťou lineárneho programovania je, že ak riešenie existuje, vždy nájde optimálne riešenie úlohy. Avšak v prípade veľkého počtu disjunktných ohraničení, čo je dosť časté pri úlohách rozvrhovania, aplikovateľnosť tejto metódy je obmedzená vzhľadom na kombinatorickú explóziu.

Branch-and-bound (vetvenie a orezávanie)

Ide o pomerne dobre známy algoritmus z operačného výskumu pre účely optimalizácie [Tsang 93]. Branch-and-bound je úplná metóda prehľadávania. V zásade ide o prehľadávanie do hĺbky plus použitie vyhodnocovacej funkcie pre orezanie priestoru prehľadávania.

Branch-and-bound prehľadáva priestor stavov (každý stav predstavuje uzol v priestore prehľadávania). V rozvrhovaní môže byť stavom priradenie hodnôt určitej podmnožine premenných. Napr. premennými môžu byť počiatočné časy operácií z príkladu 1 (úvodná kapitola). Funkcia susednosti definuje štruktúru priestoru prehľadávania (t.j. ktoré stavy sú priamo prístupné z ktorých - uzol reprezentujúci stav sa takto rozvetvuje). Napr. potomok daného stavu môže byť získaný priradením hodnoty premennej, ktorej ešte nebola priradená hodnota v rodičovských uzloch.

Postup začína koreňovým uzlom, v ktorom nie je žiadne priradenie hodnôt premenným. Tento uzol sa ďalej vetví (pre každú možnú hodnotu prvej zvolenej premennej jedna vetva). Algoritmus v každom kroku preskúma jeden uzol (jedného potomka aktuálneho uzla, resp. jeho súrodenca, ak už preskúmal všetkých potomkov), pričom postupuje smerom do hĺbky, až kým nepreskúma všetky vetvy. Naviac si algoritmus branch-and-bound pamätá doposiaľ najlepšie riešenie a skôr než začne skúmať ďalší uzol (t.j. všetkých jeho potomkov definovaných funkciou susednosti), využije doménové znalosti vo forme vyhodnocovacej funkcie, aby odhadol hodnotu optimálneho riešenia pod týmto uzlom. Ak táto odhadnutá hodnota je horšia než doposiaľ najlepšie nájdené riešenie, zvolený uzol nebude preskúmaný. Pre korektnosť tejto metódy je preto nevyhnutné, aby vyhodnocovacia funkcia nikdy nepodhodnotila (nenadhodnotila) skutočné optimálne hodnoty v maximalizačnej (minimalizačnej) úlohe.

Branch-and-bound možno použiť v kombinácii s lineárnym programovaním popísaným v predchádzajúcom bode. Stavom je v takomto prípade konjunkcia nerovníc. Potomok uzla k tejto množine pridáva ďalšiu nerovnicu. Listový uzol obsahuje kompletnú konjunkciu nerovníc, ktorých splnenie definuje úplný rozvrh.

Efektívnosť tejto metódy je veľmi ovplyvňovaná (1) kvalitou vyhodnocovacej funkcie, ktorá robí odhad optimálneho riešenia pod daným uzlom (čím presnejší odhad, tým viac vetiev je možné orezať). Ďalším dôležitým faktorom určujúcim počet stavov, ktorých prehľadávanie sa môže orezať, je (2) poradie v ktorom sú vetvy prehľadávané (čím skôr sa nájde kvalitnejšie riešenie, tým účinnejšie orezávanie). A nakoniec nemenej dôležitým je aj (3) spôsob reprezentácie úlohy, ktorý determinuje veľkosť priestoru prehľadávania (viď. príklad o rozvrhovaní rezania kusov dreva podrobne popísaný v [Dincbas a kol. 88b] kde sa veľkosť priestoru prehľadávania znížila z pôvodných 4^{72}, čo je cca 10^{43} na 7^{42}, čo je už len cca 10^{7} len zmenou spôsobu reprezentácie úlohy).

Spĺňanie ohraničení

Tento smer v umelej inteligencii priniesol množstvo algoritmov predovšetkým na riešenie úloh splniteľnosti [Dechter 92], [ParSab 95] (t.j. pre rozvrhovanie to znamená nájdenie nejakého riešenia spĺňajúceho všetky riešenia). Tieto algoritmy sú určené na všeobecne formulované tzv. konečné úlohy spĺňania ohraničení (z anglického finite constraint satisfaction problem - FCSP).

FCSP zahrňuje množinu premenných, z ktorých každá má konečnú doménu (množinu prípustných hodnôt) a množinu ohraničení, ktoré rozličným spôsobom obmedzujú hodnoty ktoré môžu premenné nadobúdať [ParSab 95]. Úlohou je priradiť každej premennej hodnotu z jej domény tak, aby boli splnené všetky ohraničenia. Ako už bolo spomínané pre tento typ úloh bolo vyvinutých množstvo algoritmov ktoré na tomto mieste nebudem uvádzať. Podrobne spracovaný prehľad týchto metód možno nájsť v [ParSab 95].

Na rozdiel od lineárneho programovania pri spĺňaní ohraničení nemusia byť len numerické premenné, môžu to byť aj enumeračné premenné s ľubovoľnou konečnou

množinou prípustných symbolov. Takisto nie je žiadne obmedzenie na typ prípustných ohraničení. Táto flexibilita znamená, že spĺňanie ohraničení má široké spektrum použitia.

Metódy z tejto skupiny sú využívané aj vnútri CLP systémov. Podrobne možno nájsť popis používaných algoritmov a techník v mojej rigoróznej práci [Paralič 95].

Vráťme sa opäť pre účely porovnania k príkladu 1. Existuje viacero spôsobov ako ho vyjadriť formou CSP. Jedným z nich je aj spôsob popísaný v časti 1.2 úvodnej kapitoly, t.j. použiť numerické premenné pre počiatočné časy operácií. Počiatočné domény budú dané dostupnosťou zdroja, na ktorý je tá ktorá operácia viazaná. Napr. operácia T_{22} môže byť inicializovaná doménou $\langle 7..20 \rangle$ (čo znamená interval celých čísel od 7 po 20) čo je dostupnosť na túto operáciu požadovaného pracoviska *P2* od 7. do 20. hodiny. Nakoľko však operácia T_{22} trvá 3 hodiny, jej počiatočný čas musí byť z intervalu $\langle 7..17 \rangle$. Okrem toho ďalšie ohraničenie udáva, že najskorší možný čas pre začiatok výroby výrobku *V2* je 9 hodín a najpozdnejší termín ukončenia jeho výroby 18 hodín, takže doména T_{22} sa zúži na $\langle 9..15 \rangle$.

Popísaný postup zodpovedá algoritmu uzlovej konzistentnosti [ParSab95], ktorý zužuje doménu premennej na základe unárnych (týkajúcich sa len tejto premennej) ohraničení, ktoré musí táto premenná spĺňať.

Ohraničenia v tejto úlohe môžu byť definované ako funkcie, ktoré vrátia hodnotu “pravda”, ak aktuálna kombinácia priradených hodnôt spĺňa dané ohraničenie a “nepravda” v opačnom prípade. Tieto funkcie spravidla opäť šíria ohraničenia do určitej miery. Napríklad ak sú známe hodnoty všetkých premenných okrem jednej, zredukuje sa jej doména tak, že sa z nej vyradia všetky neprípustné hodnoty (tzv. forward checking - dopredná kontrola). Vo všeobecnosti čím viac času sa obetuje na propagáciu ohraničení, tým menší priestor prehľadávania. Tu je ale opäť nutné hľadať rozumný kompromis medzi mierou propagácie a časom stráveným prehľadávaním. Oba procesy sú obvykle úzko spojené (detaily viď. [ParSab 95]). Aj táto skupina metód naráža na kombinatorickú explóziu.

Techniky spĺňania ohraničení sú pružnejšie než lineárne programovanie a majú širšiu oblasť použiteľnosti. Avšak väčšina metód nerieši optimalizačný problém. Niektoré metódy môžu však byť integrované do branch-and-bound na dosiahnutie lepšej efektívnosti.

Efektívnosť výslednej aplikácie opäť veľmi ovplyvní (1) spôsob formulácie problému (vo všeobecnosti je zložitosť priamo úmerná súčinu počtu premenných a veľkosti ich domén, takže čím menej premenných a čím menšie sú ich domény, tým lepšie). Ďalšími faktormi sú (2) poradie premenných v akom im budú priradzované hodnoty v priebehu prehľadávania a (3) poradie hodnôt z aktuálnej domény, v akom budú brané ako alternatívne hodnoty pre danú premennú.

Hill climbing

Touto metódou začína popis stochastických metód, alebo metód lokálneho prehľadávania (okrem hill climbing sem patrí aj simulované žíhanie a tabu search). Hill climbing [Tsang 93] je ich najjednoduchšou verziou, ale princíp je u nich rovnaký. Všetky sú totiž používané pre optimalizačné úlohy, kde je akceptovateľné aj suboptimálne riešenie. Tieto suboptimálne riešenia sú v praxi dosť často akceptovateľné (najmä ak priestor prehľadávanie je priveľký pre úplné metódy prehľadávania uvedené v predchádzajúcich bodoch).

Hill climbing vyžaduje funkciu susednosti, ktorá mapuje každý stav na skupinu ďalších - "susedných" stavov v priestore prehľadávania. Kvalita definície tejto funkcie (to si vyžaduje doménovo závislé znalosti) výrazne ovplyvňuje efektívnosť algoritmu.

Základný postup u hill climbing je veľmi jednoduchý. Počínajúc z náhodne (alebo heuristicky) generovaného stavu sa prejde do takého susedného stavu, ktorý je "lepší" vzhľadom na optimalizačnú funkciu. Tento proces sa opakuje až do chvíle, kým žiadny ďalší lepší prechod už neexistuje. Heuristika, ktorá vyberá z lepších susedných stavov môže byť rôzna (napr. vyber ľubovoľný, alebo vyber najlepší).

Hill climbing je dôležitou metódou pre riešenie úloh, u ktorých použitie úplných metód prehľadávania (viď. predchádzajúce body) už zlyháva v dôsledku kombinatorickej explózie. Obyčajne nie je zložité vyvinúť stratégiu pre hill climbing, ale nájsť dobré riešenia (blízke optimálnemu) je vždy dosť ťažké. Hlavným nedostatkom hill climbing je jeho náchylnosť uviaznuť v lokálnych optimách prípadne v oblastiach, kde sa nemení kvalita susedných riešení (rovinkách).

Simulované žíhanie

Simulované žíhanie [Kirkpatrick a kol. 83], [Tsang 95], [Crabtree 95] je rozšírením metódy hill climbing s cieľom vyviaznuť z lokálnych optím. Znamená to, že je tu aj určitá pravdepodobnosť, že smer postupu od jedného stavu k nasledujúcemu už nemusí byť len jedným smerom (od horšieho k lepšiemu), ale s určitou pravdepodobnosťou je možný aj ľubovoľný iný prechod.

Metóda vznikla v prvej polovici osemdesiatych rokov [Kirkpatrick 83]. Bola inšpirovaná procesom eliminácie defektov kryštálovej mriežky kryštálov ich ohriatím s nasledovným pomalým ochladzovaním na nízku teplotu.

Pri tomto algoritme teplota vystupuje ako riadiaci parameter, ktorý určuje, ktoré nové stavy sú akceptovateľné a ktoré nie. Vychádzajúc z preddefinovanej počiatočnej teploty, ktorá sa postupne znižuje (podľa tzv. plánu ochladzovania), majú aj slabšie susedné riešenia šancu byť vybrané. Pravdepodobnosť vybratia horšieho riešenia (vzhľadom na optimalizačnú funkciu) je priamo úmerná aktuálnej teplote. Ak teplota klesne na 0, prehľadávanie sa správa presne tak, ako hill climbing.

Algoritmus pracuje nasledovne. Prehľadávanie začína podobne ako u hill climbing zo stavu, ktorý môže byť generovaný náhodne (prípadne heuristicky). V každej iterácii je preskúmané náhodne vybraté susedné riešenie. Ak je toto riešenie lepšie ako aktuálne,

potom sa stáva novým aktuálny stavom. Ak je horšie vzhľadom na optimalizačnú funkciu, potom je akceptované ako aktuálne riešenie s pravdepodobnosťou priamo úmernou vyššie spomínanej teplote. Ak je toto riešenie odmietnuté, potom sa preskúma iné susedné riešenie podobným spôsobom. Pri tomto postupe je pravdepodobnosť dosiahnutia lepšieho riešenia vyššia než u hill climbing.

Podobne ako u metódy hill climbing, aj u simulovaného žíhania je efektívnosť tejto metódy silne závislá od definície funkcie susednosti. Naviac veľmi dôležitú úlohu zohráva plán ochladzovania. Ak je teplota znižovaná príliš rýchlo, nemusí sa tým veľmi zvýšiť pravdepodobnosť nájdenia lepšieho riešenia. Na druhej strane čím pomalšie ochladzovanie, tým dlhší čas je potrebný na ukončenie behu programu.

Tabu search

Aj keď bol tento postup vyvinutý v rámci komunity operačného výskumu, metóda tabu search [Jánošíková 94] je veľmi podobná hill climbing. Je taktiež používaná na riešenie optimalizačných úloh, u ktorých je dostatočné suboptimálne riešenie. Podobne ako simulované žíhanie, aj tabu search sa snaží vyviaznuť z lokálnych optím.

Metóda hill climbing končí dosiahnutím lokálneho optima, ktoré spravidla nie je globálnym. Metóda tabu search prekonáva toto obmedzenie a po dosiahnutí lokálneho optima pokračuje v hľadaní lepšieho riešenia. Inými slovami povolí prechod k novému riešeniu, ktoré je vzhľadom na zadanú optimalizačnú funkciu horšie, ako aktuálne. Prechodom k horšiemu riešeniu je však nutné zabrániť tomu, aby sa v nasledujúcom kroku metóda vrátila k predchádzajúcemu, lepšiemu riešeniu. Aby sa zabránilo zacykleniu metódy, teda návratu k preskúmaným riešeniam, tento zákaz sa musí vzťahovať nielen na posledný prechod (transformáciu), ale na t posledných prechodov ($t \in N$). To znamená, že z okolia aktuálneho riešenia sa vylúčia tie riešenia, ku ktorým by sa dospelo zakázanými (tabu) prechodmi. Podmienkou ukončenia algoritmu môže byť napríklad vyčerpanie všetkých možných prechodov z najlepšieho aktuálneho stavu, alebo prekročenie maximálneho povoleného počtu iterácií pre jeho aktualizáciu.

Tabu search je potrebné vidieť ako triedu algoritmov, ktoré sú charakteristické tým, že sa v nich určitým spôsobom definuje a spravuje zoznam tabu, ktorý obsahuje popis zakázaných prechodov. Môže to byť napr. zoznam do daného okamžiku už preskúmaných uzlov, alebo zoznam zakázaných smerov postupu a pod. Po každom prechode je upravený aj zoznam tabu. Rôzne algoritmy prehľadávania tabu môžu využívať rôzne stratégie manipulácie so zoznamom tabu.

Efektívnosť prehľadávania tabu v porovnaní s hill climbing závisí len od spôsobu, akým je definovaný a spracovávaný zoznam tabu. Keďže v tomto smere nie sú žiadne obmedzenia, tabu search je takto veľmi všeobecnou stratégiou riešenia.

Genetický algoritmus

Táto metóda vychádza z Darwinovej evolučnej teórie, uvažujúc sexuálnu reprodukciu kombinovanú s náhodnou mutáciou [Mach 96], [AndMach 96].

Potenciálne riešenie je reprezentované ako jedno indivíduum populácie. V používanej terminológii je nazývané chromozómom a jednotlivé časti (parametre) riešenia sú gény. Chromozóm kóduje riešenie špecifického problému jednoduchou štruktúrou, najčastejšie ako reťazec binárnych hodnôt. Klasická podoba algoritmu (základný cyklus je vyobrazený aj na obrázku 3) vyzerá nasledovne:

1. *Náhodné generovanie počiatočnej populácie*
2. *Určenie vhodnosti každého indivídua populácie*

opakuj

3. *Určenie pravdepodobnosti výberu každého indivídua*
4. *Výber subpopulácie indivíduí pre reprodukciu*
5. *Vznik nových indivíduí náhodnou reprodukciou*
6. *Náhodná mutácia nových indivíduí*
7. *Určenie vhodnosti nových indivíduí*
8. *Vytvorenie novej aktuálnej populácie*

pokiaľ podmienka ukončenia

Pri výpočte pravdepodobnosti výberu indivídua sa najprv určí priemerná vhodnosť populácie a na základe tejto sa normalizuje vhodnosť každého indivídua. Proporcionálne takto vzniklej hodnote sa stanoví hľadaná pravdepodobnosť. Tým sa dosahuje, že sľubnejšie indivíduá získavajú lepšie možnosti reprodukcie.

Z vybraných indivíduí sa náhodne zvolia dvojice rodičov a ich rekombinovaním vznikajú dvojice potomkov (veľkosť populácie ostáva konštantná). Mutácii sa prikladá rádovo menší význam ako kríženiu. Používa sa však preto, že pri krížení nevzniká nový genetický materiál, iba sa distribuuje, a práve mutácia môže zaistiť jeho tvorbu. Zvyčajne je realizovaná náhodnou inverziou náhodného bitu reťazca chromozómu.

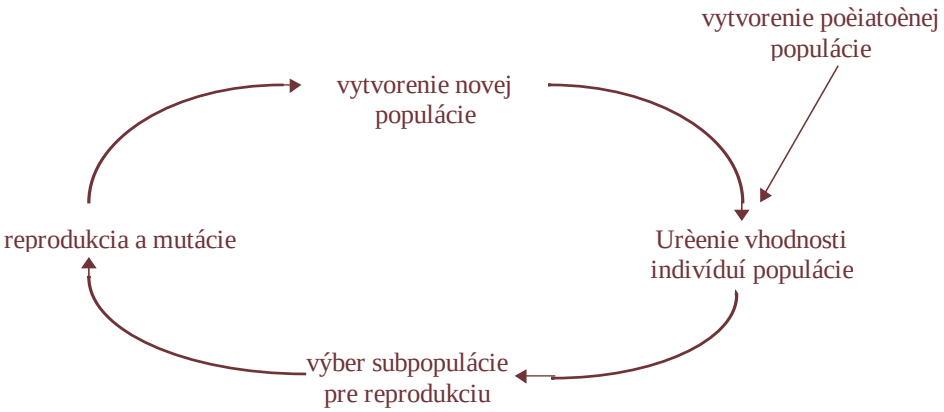
V praxi sa zaužívali dva spôsoby vytvárania novej populácie. Jeden z nich (generatívny) nahrádza všetky indivíduá starej aktuálnej generácie novými indivíduami. Pri druhom sa nová generácia tvorí z novovzniknutých indivíduí a z indivíduí starej generácie výberom indivíduí s najvyššou vhodnosťou.

Podmienka ukončenia algoritmu je zvyčajne odvodená z priemernej vhodnosti celej populácie. Ako populácia konverguje, priemerná vhodnosť populácie sa blíži vhodnosti najlepšieho indivídua. Nie je však žiadna garancia nájdenia globálneho optima.

Aby bolo možné použiť genetický algoritmus pre riešenie úloh rozvrhovania, musí sa najprv nájsť reťazec, ktorý reprezentuje možné rozvrhy. Všeobecne používané sú binárne reťazce. Napr. pre príklad 1 z úvodnej kapitoly by bolo možné reprezentovať počiatočné časy jednotlivých operácií, t.j. napr. binárny reťazec veľkosti 14 x 5 bitov pre reprezentáciu 14 operácií. Pomocou 5 bitov možno reprezentovať číslo veľkosti 0 až 31, takže asi najvýhodnejšie bude reprezentovať posunutie začiatku danej operácie oproti jej najskoršiemu možnému začiatku. Napr.: 00011 01001 01010 … 00111 možno považovať za reprezentáciu rozvrhu, v ktorom operácia T_{11} začne v čase 10=7+3(00011), T_{12}=9+8(01000), atď. Parameter zvaný vhodnosť reťazca bude u rozvrhovania daný kvalitou rozvrhu, ktorý reprezentuje (t.j. vyjadruje správnosť rozvrhu, ako aj optimalizačné kritérium). Hneď teraz je dôležité si uvedomiť, že

niektoré reprezentácie môžu byť lepšie ako iné a voľba správnej reprezentácie často znamená obrovskú pomoc pri prehľadávaní.

V rozvrhovaní vygenerovaný reťazec nemusí reprezentovať prípustný rozvrh. Jedným spôsobom ako sa vyrovnať s týmto problémom je pridať penalizačnú funkciu k parametru vhodnosti, ktorá bude vyjadrovať závažnosť porušenia ohraničení pri danom rozvrhu. Iná možnosť je "opraviť" každý reťazec generovaný algoritmom (napr. pomocou hill climbing).



Obrázok 3 Základný cyklus genetického algoritmu.

Vo všeobecnosti sa od genetických algoritmov očakáva, že majú väčšiu šancu preskúmať väčšiu časť priestoru prehľadávania než hill climbing. Avšak ako to vyplýva z ich podstaty, vyžadujú vopred nejasný počet iterácií aby našli riešenie dobrej kvality, takže vo všeobecnosti pre nájdenie riešenia je potrebný netriviálny čas[7].

Aj genetické algoritmy majú problém konvergencie, ktorý je do istej miery analogický problému uviaznutia v lokálnych optimách pri metóde hill climbing. Pretože stavebné bloky vhodnejších reťazcov majú väčšiu šancu, že prežijú v ďalšej generácii, je tu určité nebezpečenstvo, že všetky reťazce budú mať tie isté stavebné bloky. Preto je nutné nájsť určitú rovnováhu medzi mutáciami v algoritme a väčšími šancami pre vhodnejšie reťazce stať sa rodičmi.

U genetického algoritmu je nutné zvoliť rad parametrov ako veľkosť populácie, veľkosť množiny rodičov, počet križencov generovaných každým rodičovským párom, atď. Rovnako je nutné zvoliť operátory, ktoré genetický algoritmus použije na výber rodičov, rekombináciu, mutácie atď. Naviac je možné zmeniť aj vyššie popísanú riadiacu stratégiu.

[7] Týmto pojmom budem označovať také algoritmy, ktorých časová zložitosť sa nedá vyjadriť ako funkcia veľkosti vstupu.

Neurónové siete

Neurónové siete sa ukázali ako nástroj vhodný na riešenie úloh spĺňania ohraničení, vrátane optimalizačných úloh [Tsang 93]. Použitím veľkého počtu jednoduchých procesorov sa dosahuje schopnosť generovať rozvrhy rýchlejšie než je to možné ktoroukoľvek z ostatných spomínaných metód. Avšak jedným dôležitým obmedzením pri tomto prístupe je, že vybudovanie špeciálnej siete pre riešenie konkrétnej aplikácie je obvykle drahé.

Pri tomto prístupe je problém reprezentovaný ako sieť. Spôsob činnosti jednotlivých uzlov v sieti, ako aj spôsob ich vzájomného prepojenia sú kľúčom k úspechu tejto metódy.

Ako príklad možno uviesť systém Genet [Tsang 93] ktorý sa ukazuje ako veľmi sľubný. Aby ho bolo možné použiť, musí sa najprv úloha formulovať ako úloha spĺňania ohraničení. V systéme Genet každá hodnota premennej je reprezentovaná ako jeden (hodnotový) uzol a každé nebinárne ohraničenie tiež ako uzol (ohraničenia). Binárne ohraničenia sú reprezentované priamou inhibítorovou väzbou medzi hodnotovými uzlami. Každé *n*-árne ($n > 2$) ohraničenie je reprezentované už spomínaným uzlom ohraničenia, ktorý je spojený s každým relevantným hodnotovým uzlom.

Množina pravidiel je navrhnutá tak, aby zabezpečila, že sieť sa ustáli v určitom stave. Jednoduchý mechanizmus učenia (typu reinforcement) je použitý na vyviaznutie z lokálnych optím.

Pre riešenie binárnych úloh spĺňania ohraničení (premenné s konečnými doménami a ohraničenia len unárne a binárne) je postup veľmi jednoduchý. Neurónová sieť sa inicializuje priradením váh -1 všetkým hranám (všetky uzly sú v tomto prípade hodnotové). Uzly reprezentujúce rôzne hodnoty tej istej premennej tvoria samostatnú podmnožinu (cluster). Pre každú takúto podmnožinu sa jeden náhodne vybratý uzol vybudí (t.j. priradí sa mu váha 1), všetky ostatné majú váhu 0. Spustí sa výpočet v sieti, až do ustálenia, pričom v každej podmnožine bude vybudený ten uzol, ktorý má najväčšiu hodnotu na vstupe.

Dosiaľ vykonané testy na rôznych úlohách [Tsang 95] ukazujú, že Genet má vyššiu úspešnosť v nájdení riešenia pre testované riešiteľné úlohy než najlepšie známe algoritmy hill climbing vyvinuté pre tieto úlohy. Odhady hovoria, že systémom Genet by sa mali dať riešiť pomerne veľké úlohy rádovo v sekundách.

Expertné systémy

Expertné systémy majú dlhú históriu použitia pre účely rozvrhovania. Jeden z najznámejších expertných systémov pre rozvrhovanie je ISIS [Fox 87]. Väčšina týchto systémov je pravidlovo orientovaných a ich prínosy sú spravidla špecifické pre ich doménu použitia, na ktorú sú tieto systémy priam “ušité”. Preto môžu byť expertné systémy použité v princípe na ľubovoľný typ úloh, ktoré sú riešiteľné.

Architektúra takéhoto expertného systému je obyčajne veľmi jednoduchá. V zásade ide o množinu pravidiel (báza znalostí) a inferenčný mechanizmus, ktorý riadi spôsob a poradie použitia jednotlivých pravidiel. Sila expertných systémov teda tkvie v kvalite znalostí z danej domény použitia reprezentovaných v báze znalostí.

Zásluhou jasne formulovaných pravidiel, ktoré sú základom bázy znalostí takéhoto expertného systému, sú tieto systémy pomerne dobre zrozumiteľné pre používateľov. Tieto pravidlá sa získavajú od experta pre daný rozvrhovací problém a to je práve najkritickejšie miesto tejto metódy (získať a korektne formulovať vedomosti experta formou pravidiel je veľmi zložité).

Tiež voľba inferenčného mechanizmu a stratégie riešenia konfliktov medzi pravidlami (ak sú v danom okamžiku aplikovateľné viaceré pravidlá naraz), môže byť ďalším problémom. Štandardné inferenčné mechanizmy ponúka celá rada komerčných "shell-ov" pre návrh expertných systémov s pomocou ktorých môže byť vývoj aplikácie dosť urýchlený.

Systémy na programovanie ohraničení

Niektoré z metód stručne popísaných v predchádzajúcich bodoch boli zabudované do niekoľkých komerčných programovacích systémov ohraničení. Vynikajúci prehľad týchto systémov možno nájsť v [Cras 93]. Ide vlastne o rozličné programovacie jazyky, ktoré umožňujú efektívne vyjadrenie a riešenie úloh spĺňania ohraničení. Väčšina týchto systémov sú jazyky CLP, iná skupina sú potom objektovo orientované jazyky.

Tieto systémy poskytujú účinné techniky spĺňania ohraničení bez toho, aby ich musel používateľ poznať. Aj keď na druhej strane ich znalosť môže požívateľovi pomôcť zlepšiť efektívnosť vyvíjanej aplikácie.

Napríklad CLP jazyky (*CHIP* [BelCon 94], *Prolog III* [Benhamou 93], *ECL^{i}PS^e* [MeiBri 95] a iné) poskytujú na riešenie numerických ohraničení pre premenné s reálnymi doménami lineárne programovanie, pre premenné s konečnými doménami algoritmy spĺňania ohraničení a pre účely optimalizácie branch-and-bound. Naviac boli vyvinuté nové, špecializované algoritmy a prístupy (napr. [AggBel 93] pre *CHIP*). Podrobnejší opis niektorých prístupov zameraných na riešenie disjunktných ohraničení v jazyku *ECL^{i}PS^e* možno nájsť v nasledujúcich kapitolách tejto dizertačnej práce.

Jazyky s objektovo orientovaným prístupom, ako napr. *ILOG solver* [Puget 94] ponúkajú podobné techniky ako CLP, naviac sú flexibilnejšie čo sa týka riadiacich stratégií, nakoľko je možné použiť aj neúplné metódy prehľadávania, čo však na druhej strane vyžaduje experta na metódy riešenia rozvrhovacích úloh, ktorý by vedel využiť túto flexibilitu.

2.3 Spôsoby reprezentácie a propagácie disjunktných ohraničení

Ako už bolo spomínané v predchádzajúcich častiach, disjunktné ohraničenia sú veľmi často používané na vyjadrenie skutočnosti, že dve aktivity (povedzme *A* a *B*)

vyžadujú ten istý zdroj, a teda nemôžu prebiehať súčasne, ale buď *A* predchádza *B*, alebo naopak.

Propagácia, teda šírenie disjunktných ohraničení spočíva v zužovaní domén zainteresovaných premenných na základe informácie vyťaženej z ohraničení. Najcennejším v tomto smere je však určenie prípadov, keď je možná už len jedna alternatíva usporiadania dvoch aktivít v čase, čo vedie obvykle k najväčším zúženiam domén. Výsledkom je potom úprava dolných a horných časových hraníc oboch aktivít (t.j. zúženie ich domén). Existuje viacero prístupov ako riešiť tento problém. Jednotlivé prístupy možno rozdeliť do štyroch skupín.

Prvé tri skupiny sa zameriavajú len na disjunktné ohraničenie s dvoma alternatívami a sú teda použiteľné pre ľubovoľný typ disjunktných ohraničení. Tieto prístupy sú výsledkom aplikácie rôznych algoritmov spĺňania a propagácie ohraničení pochádzajúcich z komunity umelej inteligencie. Na rozdiel od týchto, prístupy z poslednej skupiny (napr. algoritmus Carlier a Pisonov, alebo intervaly úloh) vychádzajú z doménových znalostí rozvrhovacích úloh typu job-shop berúc do úvahu celé množiny úloh zdieľajúcich jeden zdroj.

1. Disjunkcie ako body výberu

Ide o najjednoduchší spôsob riešenia disjunktných ohraničení, ktorý uvažuje jednotlivé alternatívy nezávisle, a teda nevie využiť informáciu o celej disjunkcii, len o jednej jej alternatíve. Pri tomto postupe sa najskôr vyberie jedna alternatíva a ak sa neskôr v priebehu propagácie ohraničení, resp. vo fáze prehľadávania ukáže, že pre takúto voľbu riešenie neexistuje, vyberie sa nasledujúca alternatíva v poradí. Nevýhodou takéhoto riešenia je veľmi veľký počet alternatív u väčších úloh.

Predtým než uvediem ďalšie dve skupiny metód riešenia disjunktných ohraničení, najskôr ešte zavediem definície dvoch typov hranovej konzistentnosti ohraničení podľa [BapPap 95], ktoré sú ich základom.

Ohraničenie $c(v_1, v_2,\ldots, v_n)$ je **hranovo konzistentné** vtedy a len vtedy ak pre každú premennú v_i $(i = 1\ldots n)$ a každú hodnotu val_j z jej domény existujú také hodnoty $val_1, val_2,\ldots val_{i-1}, val_{i+1}, \ldots val_n$ v doménach premenných $v_1, v_2,\ldots v_{i-1}, v_{i+1}, \ldots v_n$, že platí ohraničenie $c(v_1, v_2,\ldots, v_n)$.

Hranová B-konzistencia (B podľa anglického bounds, teda hranice) naproti tomu zaručuje len to, že hodnoty $val_1, val_2,\ldots val_{i-1}, val_{i+1}, \ldots val_n$ existujú len pre val_j rovné dolnej alebo hornej hranici domény premennej v_i.

2. Disjunkcia pomocou úplnej hranovej konzistencie

Pri tomto prístupe sa využívajú disjunktné ohraničenia omnoho aktívnejším spôsobom než v predchádzajúcom prípade. Tento postup býva v literatúre často označovaný aj pojmom **konštruktívna disjunkcia** [JouSol 93], [WueMue 95].

V tomto prípade propagácia zúži domény premenných zainteresovaných v disjunktnom ohraničení takým spôsobom, že pre každú hodnotu jednotlivo (t.j. aj pre

hodnoty z vnútra domény) existuje taká hodnota z domény druhej zainteresovanej premennej, pre ktorú by bolo dané disjunktné ohraničenie splnené.

Najlepšie to bude asi ukázať na príklade. Uvažujme disjunktné ohraničenie *(Start(A) + 5 ≤ Start(B) ∨ Start(B) + 7 ≤ Start(A))*, pričom aktuálna doména oboch premenných *Start(A), Start(B)* je interval prirodzených čísel <1…10>. T.j. v terminológii rozvrhovacích úloh to znamená, že operácia *A* trvajúca 5 časových jednotiek musí predchádzať operácii *B* trvajúcej 7 časových jednotiek, alebo naopak (teda operácie *A* a *B* sa nesmú prekrývať v čase). Obe operácie pritom môžu začať len v intervale 1 až 10 časových jednotiek. Ak bude platiť prvá (ľavá) alternatíva, môže premenná *Start(A)* nadobúdať len hodnoty od 1 do 5 a *Start(B)* zase len hodnoty 6 až 10. Pre druhú (pravú) alternatívu zase *Start(A)* môže nadobúdať hodnoty 8 až 10 a *Start(B)* hodnoty 1 až 3. Z toho ale vyplýva, že hodnoty 6 a 7 pre začiatok operácie *A* (hodnota premenne *Start(A)*) a hodnoty 4, 5 pre začiatok operácie *B* (hodnota premenne *Start(B)*) neprichádzajú do úvahy a možno ich teda vybrať z jednotlivých domén premenných *Start(A)* a *Start(B)*. Tým sa môže neskôr podstatne zmenšiť priestor prehľadávania.

3. Disjunkcia pomocou hranovej B-konzistencie

Pri tomto prístupe sa využívajú disjunktné ohraničenia aktívnejším spôsobom než u disjunkcie ako nezávislé alternatívy, ale nezískava sa toľko informácie ako v predchádzajúcej metóde založenej na úplnej hranovej konzistencii. Hranová B-konzistencia je však menej náročná na výpočet.

Uvažujme disjunktné ohraničenie pri úlohách časového rozvrhovania v nasledujúcej podobe [BapPap 95]:

$$Koniec(A) \leq Start(B) \vee Koniec(B) \leq Start(A)$$

kde *Start(A), Koniec(A)*[8]*, Start(B), Koniec(B)* sú ohraničované premenné označujúce počiatočný čas, resp. čas ukončenia príslušnej operácie vo výslednom rozvrhu, ktorých hodnoty chceme určiť.

V tomto prípade vedie propagácia ohraničení k zužovaniu domén premenných nasledovným spôsobom: akonáhle je najmenšia možná hodnota premennej *Koniec(A)* (t.j. najskorší možný čas ukončenia operácie *A*) väčšia ako najväčšia možná hodnota premennej *Start(B)* (t.j. najneskorší možný čas začiatku operácie *B*), potom *A* už nemôže byť pred *B* a hranice domén premenných pre operácie *A*, resp. *B* môžu byť upravené s ohľadom na nové ohraničenie *Koniec(B) ≤ Start(A)*. Podobne to platí aj v druhom prípade, ak najskorší možný čas ukončenia operácie *B* presiahne hodnotu najskoršieho možného času začiatku operácie *A*, potom operácia *B* nemôže byť vykonávaná pred *A*.

[8] Pri konštantnom trvaní operácie *X* (označované *trvanie(X)*) stačí jedna premenná, obvykle *Start(X)*. Potom *Koniec(X) = Start(X) + trvanie(X)*.

Pre porovnanie s úplnou hranovou konzistenciou by v príklade uvedenom v predchádzajúcom bode nedošlo k žiadnej redukcii domén premenných *Start(A)*, resp. *Start(B)*.

Tento spôsob propagácie vynucuje hranovú B-konzistenciu (t.j. hranovú konzistentnosť vzhľadom na dolné a horné hranice domén premenných). To je zároveň najčastejší spôsob propagácie vyžívaný v CLP systémoch pre konečné domény (viac o ostatných typoch konzistentnosti možno nájsť v [ParSab95]).

4. Komplexnejšie metódy

Táto skupina metód je založená na určovaní situácií, kedy daná operácia musí, môže, resp. nemôže byť prvá (prípadne posledná) medzi danou množinou operácií, ktoré vyžadujú ten istý zdroj [CarPin 89], [CasLab 94].

Majme operáciu A a množinu operácií Ω, všetky zdieľajúce ten istý zdroj, takže akákoľvek súbežnosť medzi jednotlivými operáciami A a Ω je vylúčená. Nech $start\_min(A)$ označuje najskorší čas začiatku operácie A, $koniec\_max(A)$ najpozdnejší čas ukončenia operácie A a $trvanie(A)$ označuje čas vykonávania operácie A. Nech $start\_min(\Omega)$ označuje najmenší z najskorších časov začiatku operácií z množiny Ω, $koniec\_max(\Omega)$ najväčší z najpozdnejších časov ukončenia operácií z Ω a $trvanie(\Omega)$ označuje súčet časov vykonávania operácií z Ω. Nech $A \prec B$ $(A \succ B)$ znamená že operácia A musí byť vykonaná pred (po) B a $A \prec \Omega$ $(A \succ \Omega)$ znamená že operácia A musí byť vykonaná pred všetkými operáciami (po všetkých operáciách) z množiny Ω. Potom platia nasledovné dve pravidlá:

$$\begin{bmatrix} koniec\_max(\Omega) - start\_min(\Omega) < trvanie(A) + trvanie(\Omega) \\ koniec\_max(A) - start\_min(\Omega) < trvanie(A) + trvanie(\Omega) \end{bmatrix} \Rightarrow [A \prec \Omega]$$

$$\begin{bmatrix} koniec\_max(\Omega) - start\_min(\Omega) < trvanie(A) + trvanie(\Omega) \\ koniec\_max(\Omega) - start\_min(A) < trvanie(A) + trvanie(\Omega) \end{bmatrix} \Rightarrow [A \succ \Omega]$$

Na základe takýchto zistení možno hneď vydedukovať nové časové hranice pre doménu premennej A. Ak totiž A predchádza operácie z Ω, potom čas ukončenia A musí byť menší alebo rovný $koniec\_max(\Omega) - trvanie(\Omega)$. Ak A nasleduje až po operáciách z Ω, potom počiatočný čas operácie A musí byť väčší nanajvýš rovný $start\_min(\Omega) + trvanie(\Omega)$.

Pri tomto postupe je nutné si uvedomiť, že ak n operácií zdieľa ten istý zdroj, potom je nutné uvažovať $n*2^{n-1}$ dvojíc (A, Ω), nakoľko na mieste Ω môžu byť všetky podmnožiny operácií z toho istého zdroja a spolu s operáciou A je ich n zdieľajúcich ten istý zdroj. Algoritmus, ktorý vykonáva všetky takéto úpravy časových hraníc v čase[9] $O(n^2)$ bol publikovaný v [CarPin89]. Tento algoritmus je popísaný podrobnejšie v časti 5.3.

[9] Ide o zložitosť algoritmu vzhľadom na veľkosť vstupu n, ktorú budem označovať zaužívaným spôsobom pomocou $O(f(n))$, kde $f(n)$ je funkcia veľkosti vstupu n.

Iná technika založená na tom istom princípe, navrhnutá v [CasLab94] sa nazýva intervaly úloh. Aj túto metódu možno nájsť popísanú podrobne v časti 5.4.

Jednotlivé skupiny metód riešenia disjunktných ohraničení som uviedol v poradí so stúpajúcou zložitosťou a tým aj nárokmi na výpočet, aj keď samozrejme stúpa aj šanca, že sa podstatne zredukujú domény jednotlivých premenných a tým aj priestor prehľadávania. Ide tak vlastne o dva protichodné aspekty riešenia disjunktných ohraničení. Čím viac času venujeme na redukciu domén premenných pomocou dokonalejších ohraničení, tým menej času strávime v etape prehľadávania a naopak. Predkladaná práca sa snaží nájsť základné odporúčania pre použitie jednotlivých metód pre riešenie rozvrhovacích aplikácií v rámci CLP.

Za týmto účelom som implemetoval niektoré z vyššie uvedených metód v prostredí CLP a porovnával na skupine náhodne generovaných rozvrhovacích úloh, ako aj na jednej reálnej rozvrhovacej aplikácii. Podrobnejší popis konkrétne implementovaných algoritmov možno preto nájsť v kapitole 5.

2.4 Optimalizácia

Dôležitým aspektom rozvrhovacích úloh je optimalizácia, t.j. hľadanie takého riešenia, ktoré nielen že spĺňa všetky technologické ohraničenia, ale je aj optimálne podľa daného kritéria (podrobnejšie viď. časť 2.1.1).

V prostredí CLP sa na tento účel prakticky výlučne používa adaptovaná verzia algoritmu branch-and-bound zastrešená vhodným predikátom s príslušným počtom argumentov.

Branch-and-bound používa pre orezávanie tzv**. hornú a dolnú hranicu nákladov**, t.j. predpokladá, že optimálne riešenie leží niekde medzi týmito hranicami. Na tomto mieste si je dôležité uvedomiť, že okrem spôsobu a efektívnosti prehľadávania samotného branch-and-bound môže celkový čas potrebný na nájdenie riešenia výrazne ovplyvniť aj voľba týchto hraníc na začiatku prehľadávania. Čím užší je inicializačný interval, tým menší priestor prehľadávania musí spracovať branch-and-bound.

Efektívna voľba týchto hraníc je silne závislá od konkrétnych znalostí o riešenej rozvrhovacej úlohe. Pre rozsiahle úlohy je obvykle nevyhnutné venovať určitý čas na výpočet čo možno najlepšieho odhadu pre dolnú a hornú hranicu. Tento čas sa obvykle mnohonásobne vráti v ďalšej etape výpočtu, keď nastúpi branch-and-bound. Niektoré postupy pre výpočet týchto hraníc sú uvedené v kapitole 7.

Dôvod, prečo sa v CLP jazykoch (ako napr. *CHIP* [AggBel 91], *ELC^{i}PS^{e}* alebo *cc(FD)* [Hentenryck a kol. 93]) používa práve tento prístup, je jednoduchý. **Branch-and-bound** je úplná metóda prehľadávania a výborne sa hodí do prostredia kde sú možné návraty (z anglického backtracking), ktoré sú typickým sprievodným javom prehľadávania do hĺbky u logických programovacích jazykov.

Stručný popis tejto stratégie bol uvedený v časti 2.2, na tomto mieste uvádzam len to najnutnejšie z pohľadu CLP. Branch-and-bound hľadá riešenie úlohy a po jeho nájdení pridá ďalšie ohraničenie, že každé nové riešenie musí byť lepšie podľa daného

kritéria, než doteraz najlepšie nájdené riešenie. V princípe existujú dve stratégie (sú implementované napr. V CLP jazykoch *CHIP* a *ELCiPSe*) [MudPre 95]:

MINMAX[10] Vychádzajúc zo známej počiatočnej hodnoty hornej a dolnej hranice nákladov, ktoré sa požadujú od riešenia $C^{\max}$ a $C^{\min}$ používa ohraničenie $C^{\min} \le C \le C^{\max}$ (ide o celé čísla) na orezávanie priestoru prehľadávania. Akonáhle nájde riešenie s nákladmi C_n, prehľadávanie zastaví a začne opäť odznova, ale s novým ohraničením $C^{\min} \le C \le C_n - 1$. Ak sa nenájde žiadne ďalšie riešenie, alebo ak $C_n = C^{\min}$, potom posledne nájdené riešenie je optimálne.

MINIMIZE[11] pracuje veľmi podobne s tým rozdielom, že po nájdení riešenia nezačína prehľadávať od začiatku, ale pokračuje na mieste, kde sa práve nachádza.

Na jednej strane MINIMIZE v porovnaní s predchádzajúcim MINMAX nemusí odznova prehľadávať tú časť priestoru, ktorú už predtým prehľadal do nájdenia posledného riešenia. Na druhej strane sa pri tejto metóde objavuje u mnohých úloh jav v angličtine nazývaný trashing v prípade, že množstvo riešení s podobnými nákladmi je topologicky blízko seba v priestore prehľadávania.

Tento postup je možné vylepšiť dvoma spôsobmi.

1. Ak sa uspokojíme so **suboptimálnym riešením v rámci vopred zadanej presnosti** E (číslo od 0 do 1)[12], potom je možné oba vyššie uvedené postupy upraviť nasledovne.

Po nájdení nového riešenia C_n sa novou hornou hranicou stane $C_n(1-E)-1$ (namiesto pôvodnej $C_n - 1$). Ak sa nenájde žiadne lepšie riešenie, potom vieme, že optimálne riešenie leží v intervale $\langle C_n(1-E), C_n \rangle$. Tento prístup čiastočne predchádza javu nazvanému vyššie trashing.

2. **Paralelným prehľadávaním** priestoru prehľadávania. Ide o tzv. paralelizmus typu OR. Ďalšie vylepšenia boli dosiahnuté využitím paralelizmu na prehľadávanie s viacerými hranicami pre náklady súčasne, ako boli navrhnuté a testované v [MudPre95].

Táto stratégia predchádza tak trashingu u MINIMIZE ako aj zbytočnému opätovnému prehľadávaniu častí priestoru u MINMAX.

[10] Táto stratégia je implementovaná v CLP jazyku *ELC^{i}PS^e* v rámci zabudovaného predikátu min_max/2.
[11] Táto stratégia je implementovaná v CLP jazyku *ELC^{i}PS^e* v rámci zabudovaného predikátu minimize/2.
[12] V CLP jazyku *ELC^{i}PS^e* tejto stratégii zodpovedajú zabudované predikáty min_max/5, resp. minimize/5.

K a p i t o l a 3

CIELE DIZERTAČNEJ PRÁCE

V tejto kapitole sú do piatich hlavných okruhov zhrnuté základné ciele dizertačnej práce. V zátvorke pri každom cieli možno nájsť odkaz na tú časť práce, kde je popísané splnenie príslušného cieľa.

1. Porovnať rôzne metódy riešenia úloh rozvrhovania pomocou vytipovaných kritérií a zhodnotiť postavenie CLP medzi ostatnými metódami.
   a) Na základe základných charakteristík najčastejšie používaných metód na riešenie úloh rozvrhovania (2.2) ich rozdeliť do skupín podľa príbuznosti (4.1).
   b) Vytipovať základné kritériá dôležité pri výbere najvhodnejšej metódy na riešenie konkrétnej rozvrhovacej aplikácie (4.2).
   c) Zhodnotiť postavenie CLP medzi ostatnými metódami na riešenie rozvrhovacích úloh (4.3).
2. Analýza viacerých metód na riešenie disjunktných ohraničení, ich implementácia v špeciálnom tvare symbolických ohraničení v prostredí CLP, otestovanie a vzájomné porovnanie (kapitoly 5 a 6).
   a) Disjunkcia ako nezávislé alternatívy (5.1).
   b) Disjunkcia na základe hranovej konzistencie - úplná hranová konzistencia a hranová B-konzistencia (5.2).
   c) Spracovať základné poznatky z algoritmu Carlier a Pinsona pre úlohy typu job-shop, upraviť ich a navrhnúť na ich základe symbolické ohraničenie v CLP jazyku ECL^iPS^e (5.3).
   d) Spracovať základné poznatky z algoritmu intervalov úloh pre úlohy typu job-shop a navrhnúť na ich základe symbolické ohraničenie v CLP jazyku ECL^iPS^e (5.4).
3. Implementovať prostriedkami CLP viaceré rozvrhovacie úlohy vrátane reálnej aplikácie s cieľom otestovať rôzne prístupy k riešeniu disjunktných ohraničení a vypracovať základné doporučenia pre ich použitie (kapitola 6).
   a) Úlohy typu job-shop (6.1).
   b) Návrh časového harmonogramu výstavby päťsegmentového mostu (6.3).
4. Návrh a otestovanie nových postupov pre riešenie optimalizácie v CLP (kapitola 7).

a) Otestovať a porovnať viaceré heuristiky pre nájdenie suboptimálneho riešenia a navrhnúť novú heuristiku s lepšími vlastnosťami (7.1).

b) Navrhnúť nové postupy pre hľadanie optimálneho riešenia a vyhodnotiť ich na množine testovacích úloh (7.2).

5. Návrh metodológie riešenia rozvrhovacích úloh prostriedkami CLP a jej aplikovanie pri riešení reálnej rozvrhovacej aplikácie (kapitola 8).

   a) Navrhnúť a popísať základné kroky v procese formulácie rozvrhovacej úlohy v CLP jazyku (8.1).

   b) Popísať základné aspekty optimalizácie v CLP (8.2).

   c) Aplikovať navrhnutú metodológiu pre riešenie reálnej úlohy rozvrhovania brám do narážacích pecí v závode VSŽ Oceľ s.r.o. (8.3).

K a p i t o l a 4

POROVNANIE METÓD NA RIEŠENIE ÚLOH ROZVRHOVANIA

4.1 Rozdelenie metód do skupín podľa príbuznosti

Z analýzy popísaných metód riešenia úloh rozvrhovania vyplýva, že v zásade existujú tri skupiny metód.

1. **Úplné metódy**, ktoré zaručujú nájdenie (optimálneho) riešenia ak existuje (lineárne programovanie, branch-and-bound, spĺňanie ohraničení).
2. **Neúplné metódy** (hill climbing, simulované žíhanie, tabu search, genetické algoritmy), ktoré neprehľadávajú celý priestor prehľadávania, iba jeho časť s tým že zaručujú iba nájdenie suboptimálneho riešenia.
3. Iné, **neštandardné metódy**, kam možno zahrnúť neurónové siete a expertné systémy.

Aj keď je druhá skupina metód často veľmi účinná pre rozsiahle úlohy, kde metódy z prvej skupiny narazia na problém kombinatorickej explózie, je potrebné si uvedomiť, že stochastické (neúplné) metódy fungujú dobre len pre úlohy, kde podobné (susedné) riešenia majú aj približne rovnaké náklady. Ak totiž nie je vzťah medzi podobnými riešeniami a ich nákladmi, potom niet dôvodu, prečo by mali byť stochastické metódy lepšie než štandardný backtracking (t.j. najjednoduchší algoritmus pre úplné prehľadávanie).

Pojem “podobnosti” medzi riešeniami je zachytený v operátoroch definovaných pre danú úlohu. U hill climbing a simulovaného žíhania ide o funkcie susednosti, ktoré generujú nové (susedné) riešenia, u genetických algoritmov zase operátor rekombinácie, ktorý generuje nové riešenie ako vhodnú kombináciu dvoch už nájdených riešení.

4.2 Kritériá pre výber najvhodnejšej metódy

Aby sa riešiteľ rozvrhovacej aplikácie mohol rozhodnúť, ktorú metódu použiť, potrebuje do hĺbky poznať tak riešený problém, ako aj jednotlivé metódy. Pri tejto analýze je nutné zodpovedať niektoré základné otázky z odpovedí na ne by mali vyplynúť základné doporučenia z hľadiska použiteľnosti jednotlivých metód. Existujú štyri základné kritériá:

1. Splniteľnosť alebo optimalizácia

Najprv je dôležité si uvedomiť, či je cieľom nájsť akékoľvek riešenie spĺňajúce všetky zadané ohraničenia (úlohy splniteľnosti), alebo je potrebné nájsť riešenie, ktoré je optimálne z hľadiska nejakého kritéria (optimalizačné úlohy).

Lineárne programovanie (len pre úlohy, kde sú ohraničenia aj kriteriálna funkcia lineárne), branch-and-bound, hill climbing, simulované žíhanie, tabu search a genetický algoritmus sú určené najmä pre riešenie optimalizačných úloh. Spĺňanie ohraničení na úlohy splniteľnosti. Expertné systémy a neurónové siete sa dajú vybudovať na dosiahnutie akéhokoľvek potrebného cieľa.

Tu je nutné zmieniť sa ešte o jednom dôležitom a častom fenoméne, a síce preferenčných ohraničeniach. Často totiž v rozvrhovaní sú definované ohraničenia, ktoré musia byť splnené (**tvrdé, alebo technologické ohraničenia**), ale aj ohraničenia, ktorých splnenie by bolo síce vítané, ale nie je nevyhnutné (**mäkké, alebo preferenčné ohraničenia**).

V takomto prípade sú možné v zásade dva prístupy:

- považovať preferenčné ohraničenia za tvrdé a problém sa stáva problémom splniteľnosti. Ak riešenie neexistuje, potom je možné vybrať niektoré preferenčné ohraničenia a uvolniť ich (nebrať do úvahy).
- porušenie preferenčného ohraničenia zarátať do nákladov a celý problém riešiť ako optimalizačný s tým, že sa minimalizujú náklady (a teda počet porušených preferenčných ohraničení).

2. Výpočtový čas verzus optimálnosť riešenia

Pre nájdenie zaručene optimálneho riešenia je nutné použiť niektorú z úplných metód. Avšak tie narážajú na kombinatorickú explóziu. Čas výpočtu totiž exponenciálne narastá s veľkosťou úlohy (veľkosť úlohy je daná počtom množín disjunktných operácií a ich veľkosťou).

Stochastické metódy si poradia aj s rozsiahlejšími úlohami, ale zaručujú len suboptimálne riešenie. Preto je nutné zvážiť, ktorá požiadavka je dôležitejšia. Napríklad pri dlhodobom plánovaní ktoré narába s drahými zdrojmi, nie je až taký rozhodujúci čas výpočtu, ale práve optimálnosť riešenia. V takom prípade je potrebné použiť niektorú z úplných metód.

Naopak sú zasa situácie (napr. v mimoriadnych stavoch), kedy je úplne postačujúce suboptimálne riešenie, ktoré je ale čo možné získať čo najrýchlejšie. V takom prípade má najväčšie šance neurónová sieť nasledovaná metódou hill climbing. Simulované žíhanie a tabu search budú asi potrebovať viac času, čo je cena za nájdenie lepšieho riešenia.

3. Špecifikácia problému

Každá z vyššie popísaných metód si vyžaduje svoju reprezentáciu úlohy. Lineárne programovanie narába s množinou lineárnych nerovníc a kriteriálnou funkciou.

Branch-and-bound možno použiť na každý optimalizačný problém, ak je k dispozícii spôsob, ako ohodnotiť kvalitu čiastočného riešenia. To je málokedy problémom, ale efektívnosť prehľadávania závisí od presnosti tohoto ohodnotenia.

Spĺňanie ohraničení sa používa najmä pre úlohy s konečnými doménami, aj keď niektoré z nich sú aplikovateľné aj na reálne domény.

Hill climbing, simulované žíhanie a tabu search možno aplikovať na široké spektrum úloh. Ich kvalita záleží od funkcií susednosti, ktoré systém používa pre nájdenie nového riešenia (u simulovaného žíhania je naviac veľmi dôležitý plán ochladzovania a u tabu search spôsob vytvárania a narábania s tabu zoznamom).

Efektívnosť genetických algoritmov veľmi záleží na tom, ako sú reprezentovaní kandidáti na riešenie a ako je definovaná vyhodnocovacia funkcia, čo si vyžaduje expertízu riešiteľa rozvrhovacej aplikácie.

Bolo ukázané, že neurónové siete sú použiteľné na riešenie úloh ktoré možno formulovať ako konečné úlohy spĺňania ohraničení, a to bez, aj s požiadavkou na optimalizáciu.

4. Voľba algoritmu a implementácia

Algoritmy lineárneho programovania, branch-and-bound a spĺňania ohraničení sú dobre definované v literatúre a ich implementácia je pomerne priamočiara, aj keď nie vždy jednoduchá. U spĺňania ohraničení je naviac potrebné vybrať si z veľkého množstva existujúcich algoritmov (v [ParSab 95] možno nájsť pomerne rozsiahly prehľad existujúcich algoritmov a početné odkazy na literatúru) čo si vyžaduje značné vedomosti.

Podobne aj základný algoritmus pre hill climbing je dobre definovaný, ale efektívnosť tejto metódy je silne závislá od kvality funkcie susednosti. Jej definícia leží na riešiteľovi danej rozvrhovacej úlohy (nájsť nejakú nie je obvykle až taký problém, ale nájsť takú, ktorá bude efektívne prehľadávať priestor, je ťažká úloha).

Simulované žíhanie naviac oproti hill climbing vyžaduje aj definíciu plánu ochladzovania, ktorý je kritickou zložkou algoritmu. Náročnosť implementácie týchto algoritmov je daná hlavne zložitosťou použitej funkcie susednosti. U tabu search ešte naviac aj zložitosťou tabu zoznamu a jeho manipulačného mechanizmu.

Expertné systémy môžu byť vytvorené s použitím existujúcich shell-ov relatívne ľahko, ale nájdenie efektívnych pravidiel je obvykle veľmi zložité.

Veľká výhoda systémov na programovanie ohraničení je v tom, že riešiteľ rozvrhovacej aplikácie môže využívať metódy lineárneho programovania, branch-and-bound, algoritmy spĺňania ohraničení a prípadne ďalšie metódy bez toho, aby sa ich musel učiť a sám implemetoval.

Členenie metód vzhľadom na uvedené kritériá je zhrnuté v nasledujúcej tabuľke 2.

metóda	všeobecné zásady	zásady špecifické pre danú metódu
lineárne programovanie	• splniteľnosť aj optimalizácia • úplné	• problém musí byť zadaný formou množiny rovníc a nerovníc
branch-and-bound	• optimalizácia • úplné	• vyžaduje heuristiky na orezávanie • dôležité je poradie prehľadávania vetiev
spĺňanie ohraničení	• splniteľnosť • úplné aj neúplné	• existuje veľké množstvo algoritmov • vhodné najmä pre netriviálne ohraničenia
hill climbing	• splniteľnosť a optimalizácia, ak stačí suboptimum • hill climbing môže uviaznuť v lokálnych optimách • simulované žíhanie a tabu search sa z nich snažia vyviaznuť	• vyžaduje funkciu susednosti, ktorá je rozhodujúca pre efektívnosť
simulované žíhanie		• funkcia susednosti rozhoduje o efektívnosti • plán ochladzovania je kritický
tabu search		• efektívnosť závisí najmä od stratégie manipulácie s tabu zoznamom
genetické algoritmy	• optimalizácia • neúplné	• rozhodujúca je reprezentácia • efektívnosť môže byť citlivá na voľbu hodnôt parametrov a operátorov
neurónové siete	• splniteľnosť alebo optimalizácia • rýchly výpočet	• zostavenie siete a mechanizmus zmeny hodnôt sú rozhodujúce • vytvorenie špeciálnej siete môže byť veľmi drahé
expertné systémy	• šité na mieru • sila je v doménovo závislých znalostiach	• nutné je získanie vedomostí od experta a to môže byť zložité

Tabuľka 2 Zásady, ktoré je potrebné brať do úvahy pri výbere metódy na riešenie úloh rozvrhovania.

4.3 Zaradenie CLP do systému metód

Keď sa pozrieme na CLP z pohľadu metód popísaných v prehľadovej časti dizertačnej práce (časť 2.2) a ich členenia v časti 4.1, môžeme tvrdiť, že CLP v sebe integruje množinu metód z prvej skupiny.

CLP totiž integruje v sebe algoritmy lineárneho programovania (pre reálne prípadne racionálne premenné), branch-and-bound (hľadanie optimálneho riešenia) a spĺňanie ohraničení (pre celočíselné a enumeračné premenné). Naviac je omnoho pružnejšie v reprezentácii netypických ohraničení, s ktorými sa často stretávame v konkrétnych aplikáciách. V prostredí CLP je možné implementovať veľmi prirodzene aj expertné systémy.

Ako už bolo spomínané, vďaka svojej deklaratívnosti tak ponúka CLP veľmi účinný prostriedok pre riešenie úloh rozvrhovania, ako aj experimentovanie s rôznymi postupmi bez toho, aby bolo nutné programovať špeciálne algoritmy. A to všetko pri podstatne menšom rozsahu zdrojového kódu, než je tomu napr. u tradičných procedurálnych programovacích jazykov.

Vďaka týmto vlastnostiam som si vybral práve CLP ako nástroj pre implementáciu a testovanie rôznych prístupov k riešeniu disjunktných ohraničení, ako aj optimalizačných postupov a ich vyhodnotenie.

K a p i t o l a 5

RIEŠENIE DISJUNKTNÝCH OHRANIČENÍ V PROSTREDÍ CLP

V prehľadovej časti 2.3 boli uvedené 4 základné spôsoby reprezentácie a propagácie disjunktných ohraničení. Keď sa však na nich bližšie pozrieme, prídeme na to, že prvé tri skupiny (disjunkcia ako nezávislé alternatívy, úplná hranová konzistencia a hranová B-konzistencia) sú použiteľné naozaj pre ľubovoľné disjunktné ohraničenia (B-konzistencia len pre premenné s usporiadanými doménami). Metódy poslednej skupiny sú viazané len na úlohy časového rozvrhovania, kde disjunktné ohraničenia reprezentujú operácie zdieľajúce ten istý zdroj, ktoré sa preto nesmú v čase prekrývať. Všetky skupiny metód sú však zaujímavé z pohľadu mojej témy dizertačnej práce.

Ďalší dôležitý fakt, ktorý je potrebné si uvedomiť je, že samotná reprezentácia ako aj spôsob propagácie disjunktných ohraničení obvykle v rôznej miere zmenšuje priestor, ktorý sa v ďalšej etape riešenia prehľadáva. Aj samotné prehľadávanie riadené priradzovaním hodnôt premenným z ich aktuálnych domén je prekrývané propagáciou ohraničení. Z toho vyplývajú dva protichodné dôsledky pre efektivitu výsledného riešenia.

(1) Jednotlivé spôsoby reprezentácie je možné ľubovoľne kombinovať za účelom dosiahnutia lepšej propagácie ohraničení, čiže zmenšovania domén zainteresovaných premenných, a teda v konečnom dôsledku zmenšovania tej časti priestoru prehľadávania, ktorá musí byť preskúmaná. Takto sa skracuje čas potrebný na prehľadávanie.

(2) Čím lepšiu propagáciu sa snažíme dosiahnuť, tým viac času pri nej spotrebujeme.

Z celkového pohľadu na riešený problém je ale nutné brať do úvahy súčet časov potrebných na propagáciu ohraničení a samotné prehľadávanie. Aby sme minimalizovali úhrnný čas potrebný na vyriešenie celej úlohy, je potrebné nájsť rovnováhu medzi oboma protichodnými požiadavkami.

Vo svojej práci som sa pokúsil preskúmať a objasniť tieto základné aspekty súvisiace s reprezentáciou a propagáciou disjunktných ohraničení za účelom efektívneho riešenia rozvrhovacích úloh.

Za týmto účelom som upravil pre podmienky CLP a implementoval viacero algoritmov z vyššie uvedených skupín, konkrétne v CLP jazyku *ECL^{i}PS^e*.[13] Algoritmy som potom zvlášť a v kombináciách testoval na náhodne generovaných úlohách typu

[13] *ECL^{i}PS^e* je skratka z anglického ECRC (European Computer-Industry Research Centre) Common Logic Programming System.

job-shop, ako aj na reálnej rozvrhovacej aplikácii. Cieľom bolo nájsť čo možno najefektívnejší spôsob spracovania disjunktných ohraničení v prostredí CLP, podať základné doporučenia pre použitie rôznych implementovaných disjunktných ohraničení a vypracovať metodológiu riešenia rozvrhovacích úloh v prostredí CLP.

Pokiaľ je mi známe, takéto komplexné porovnanie nebolo zatiaľ publikované a pokladám ho za dôležité pre efektívnejšie využitie CLP na účely riešenia úloh rozvrhovania.

V tejto kapitole sú postupne uvedené všetky testované základné spôsoby reprezentácie a propagácie disjunktných ohraničení, popis ich implementácie v jazyku *ECL^{i}PS^e* a výsledky dosiahnuté pri ich testovaní na skúmanej množine príkladov, ktorých podrobnejší popis možno nájsť v nasledujúcej kapitole 6.

5.1 Disjunkcia ako nezávislé alternatívy

Ako už bolo uvedené v časti 2.3, najjednoduchší spôsob reprezentácie dvoch vylučujúcich sa alternatív v prostredí CLP je reprezentácia pomocou dvoch klauzúl. Z operačného hľadiska logického programu to znamená, že v priebehu výpočtu v mieste, kde sa vyskytne viac ako jedna alternatívna klauzula, vznikne bod výberu s toľkými alternatívami, koľko klauzúl pre splnenie daného cieľa program obsahuje.

Najskôr sa vyberie prvá klauzula. Ak sa však neskôr v priebehu výpočtu dôjde do slepej uličky, t.j. nie je možné splniť takto vybrané klauzuly, dôjde k návratu (anglicky backtracking) do predchádzajúceho bodu výberu a vyberie sa nasledujúca, alternatívna klauzula.

Pre úlohy rozvrhovania majú disjunktné ohraničenia charakter dvoch alternatívnych poradí usporiadania dvoch úloh zdieľajúcich ten istý zdroj vo výslednom rozvrhu.

Ak totiž máme rozvrhnúť dve úlohy (povedzme so začiatočnými časmi SI, resp. SJ a dĺžkami trvaní TI, resp. TJ), ktoré musia zdieľať spoločný zdroj, potom máme dve možnosti. Alebo bude v rozvrhu najprv prvá úloha, a potom druhá nemôže začať skôr, kým neskončí prvá, alebo opačne. Túto skutočnosť možno reprezentovať nasledovným nedeterministickým predikátom s dvoma klauzulami:

```
disjunktne(SI, SJ, TI, TJ) :-
        SJ #>= SI + TI.
disjunktne(SI, SJ, TI, TJ) :-
        SI #>= SJ + TJ.
```

V tomto predikáte “#>=” znamená v jazyku *ECL^{i}PS^e* ohraničenie s významom väčší alebo rovný, ktoré ale naviac predpokladá, že jeho argumentmi môžu byť okrem číselných hodnôt aj premenné s konečnými doménami, na ktoré je možné aplikovať algoritmy hranovej konzistencie. Obvykle z dôvodov efektívnosti ide u väčšiny CLP systémov o hranovú B-konzistenciu (definícia viď. časť 2.3).

To v praxi znamená, že výberom jednej z uvedených klauzúl sa dané ohraničenie stáva aktívnou súčasťou aktuálneho zásobníka ohraničení a kedykoľvek sa zmení (dolná alebo horná) hranica niektorej zainteresovanej premennej (SI, resp. SJ, TI a TJ sú totiž obyčajne konštantné), toto ohraničenie sa automaticky aktivuje, t.j. testuje sa, či sa nezmenila jeho platnosť. Naviac tento test môže viesť k ďalšiemu zúženiu domén zainteresovaných premenných o tie hodnoty, pre ktoré aktuálne ohraničenia nemôžu byť splnené.

Na tomto mieste je potrebné uviesť, že z operačného hľadiska možno vyššie uvedenú dvojicu klauzúl definujúcich predikát disjunktne/4 v rámci CLP využiť aj iným spôsobom. V [Dincbas a kol. 90] boli v CLP jazyku *CHIP* testované a vzájomne porovnávané tri rôzne operačné stratégie narábania s takýmito disjunkciami.

Programátorsky sa to dosiahlo vždy len jednoduchou deklaráciou, ktorá predpisovala, aký druh propagácie ohraničení sa má pre danú definíciu predikátu použiť. Napr. forward disjunktne(d,g,d,g).[14] v jazyku *CHIP* predpisuje použiť dopredu kontrolu pre propagáciu ohraničenia disjunktne/4. Ak sa v programe nenachádzala žiadna deklarácia, potom sa použil tradičný prístup výberom nezávislých alternatív.

- Okrem **nezávislých alternatív** to boli aj ďalšie dve stratégie, a síce:
- dopredná kontrola (anglicky **forward checking**) a
- dopredný pohľad (anglicky **lookahead**).

Dopredná kontrola (podrobnejšie viď. [Hentenryck 89]) je taký mechanizmus, kde sa ohraničenie aktivuje až v okamihu, akonáhle ostane zo všetkých zainteresovaných premenných nenaviazaná už len jedna, pričom z jej domény odstráni všetky hodnoty, ktoré sú nekonzistentné s aktuálnym zásobníkom ohraničení, takže ostanú len hodnoty spĺňajúce dané ohraničenia.

Dopredný pohľad (podrobnejšie viď. [Hentenryck 89]) sa aktivuje pri každej zmene v doméne niektorej zo zainteresovaných premenných a z domény každej z nich sú odstránené hodnoty, ktoré nemôžu splniť ohraničenie. V našom prípade budú okamžite vyskúšané všetky kombinácie hodnôt premenných SI a SJ z ich aktuálnych domén (čo zodpovedá úplnej hranovej konzistencii).

Ako sa ukázalo [Dincbas a kol. 90], z uvedených troch možností jednoznačne najefektívnejšia pre rozvrhovacie aplikácie sú nezávislé alternatívy. Dôvodom je veľká časová náročnosť algoritmov doprednej kontroly a dopredného pohľadu najmä pre väčšie domény premenných a veľké množstvo disjunktných ohraničení.

5.2 Disjunkcia pomocou hranovej konzistencie

V predchádzajúcej časti bol popísaný prístup, kedy sú jednotlivé alternatívy disjunkcie brané nezávisle. V tejto časti budú popísané také prístupy a spôsoby

[14] V tejto deklarácii "d" označuje doménové premennú a "g" naviazanú (anglicky ground) hodnotu na mieste príslušných argumentov predikátu disjunktne/4.

implementácie, kedy je disjunkcia využívaná aktívne. Znamená to, že dvojica disjunktných ohraničení sa berie do úvahy viac či menej súčasne, pričom sa snažíme získať viac informácií, a teda lepšiu propagáciu ohraničení ako v predchádzajúcom prípade.

V jazyku *ECL^{i}PS^e* existuje viacero spôsobov, ako to dosiahnuť. Najjednoduchší z hľadiska programátora je postup využívajúci zabudovanú knižnicu zovšeobecnenej propagácie Propia (podrobnejšie som sa venoval princípom zovšeobecnenej propagácie vo svojej rigoróznej práci [Paralič 95]).

Podstata použitia je veľmi podobná v predchádzajúcej časti 5.1 popísanému postupu pre voľbu spôsobu propagácie v jazyku *CHIP*. Pri použití knižnice Propia je možné voliť si mieru propagácie daného ohraničenia tiež iba jednoduchou deklaráciou. Táto deklarácia predpíše, koľko informácie sa má vyťažiť z definície (ktorá môže prirodzene pozostávať z viacerých klauzúl) daného predikátu (ohraničenia).

K dispozícii sú tieto typy propagácie:

- infers most - je najsilnejším ohraničením, nakoľko sa snaží vyťažiť maximum informácie z definície predikátu. Jeho použitie však môže byť náročné na výpočtový čas (zodpovedá úplnej hranovej konzistencii - viď. 5.2.1).

- infers ground - je veľmi jednoduchý spôsob vyhodnotenia. Akonáhle totiž ešte existujú dve (alebo viac) odpovedí na zadaný cieľ, vyhodnotenie ohraničenia sa pozastaví až do doby, kým nebude k dispozícii viac informácií. Propagácia je úspešná len v prípade, keď vedie k jednoznačnému priradeniu hodnoty premennej.

- infers consistent - je ešte jednoduchší spôsob vyhodnotenia. Ide len o test, či vôbec existuje nejaká odpoveď na zadaný cieľ. Ak odpoveď neexistuje, dôjde k neúspechu. Ak odpoveď existuje, toto ohraničenie testuje, či nie je už cieľ splnený.

- infers size(N) - získa všetky informácie, ktoré možno vyjadriť doménou veľkosti menšej alebo rovnej N. Ak ohraničenie zabezpečuje takéto zúženie domény, potom sa vykoná. V opačnom prípade sa ohraničenie pozastaví až do doby, kým nebude k dispozícii viac informácií.

5.2.1 Úplná hranová konzistencia

Za pomoci knižnice Propia bolo definované aj ohraničenie disjunction/5, ktoré využíva maximálnu mieru propagácie v rámci definície disjunktného ohraničenia (infers most) a zodpovedá teda úplnej hranovej konzistencii popísanej v časti 2.3. Samotný výpis infers most je už pred používateľom v poslednej verzii *ECL^{i}PS^e v3.5.2* skrytý (viď. nasledujúce príklady).

```
disjunction(SI, SJ, TI, TJ, Flag) :-
        disjunct(SI, SJ, TI, TJ, Flag) infers most.

disjunct(SI, SJ, TI, TJ, 1) :-
        SJ #>= SI + TI.
disjunct(SI, SJ, TI, TJ, 2) :-
        SI #>= SJ + TJ.
```

disjunction(Start1,Trvanie1,Start2,Trvanie2,Flag) predpokladá, že Start1 a Start2 sú doménové premenné reprezentujúce počiatočné časy operácií 1 a 2, ktoré zdieľajú ten istý zdroj. Trvanie1 a Trvanie2 sú celočíselné hodnoty reprezentujúce časy trvania týchto operácií a Flag je príznak ktorého hodnota určuje vzájomné usporiadanie tejto dvojice operácií vo výslednom rozvrhu. Vo všeobecnosti je to teda tiež doménová premenná s dvoma možnými hodnotami. Ak nadobudne hodnotu 1, potom operácia 1 musí byť vykonaná pred operáciou 2 a naopak Flag=2 ak musí byť vykonaná najprv operácia 2 a potom operácia 1.

Toto ohraničenie sa aktivuje kedykoľvek dôjde k zmene aspoň niektorej z domén Start1 alebo Start2. A to pri akejkoľvek zmene, nielen pri zmene hraníc (ako tomu je u hranovej B-konzistencie). Toto ohraničenie je teda schopné propagovať aj zmeny vo vnútri domén (diery). Naviac sa samozrejme budí aj pri nastavení príznaku Flag na celočíselnú hodnotu 1 alebo 2 a podľa toho vnúti zodpovedajúce ohraničenie (z danej disjunktnej dvojice) o usporiadaní týchto dvoch operácií.

Uvediem príklad ktorý poukazuje na správanie sa ohraničenia disjunction/5. Použime tie isté hodnoty ako v časti 2.3 pre operáciu A trvajúcu 5 a B trvajúcu 7 časových jednotiek, pričom obe premenné budú mať rovnakú počiatočnú doménu 1 až 10. Potom zodpovedajúce volanie v jazyku ECL^iPS^e bude:

```
[eclipse 6]: [A, B]::1..10, disjunction(A, 5, B, 7, F).

  A = A{[1..5, 8..10]}
  B = B{[1..3, 6..10]}
  F = F{[1, 2]}

  Delayed goals:
disjunction(A{[1..5, 8..10]}, 5, B{[1..3, 6..10]}, 7, F{[1, 2]})
  yes.
```

Výsledok naznačuje, že prebehla propagácia zodpovedajúca úplnej hranovej konzistencii popísanej v časti 2.3. Domény počiatočných časov operácií A a B sa totiž zúžili (A má doménu 1 až 5 a 8 až 10, operácia B zase 1 až 3 a 6 až 10). Zároveň však samotné ohraničenie zostalo uspané (delayed goals).

Skúsme teraz vyskúšať, čo sa stane, keď bude vnútené nejaké usporiadanie operácií tým, sa nastaví hodnota príznaku Flag povedzme na 1.

```
[eclipse 7]: [A, B]::1..10, disjunction(A, 5, B, 7, 1).

  A = A{[1..5]}
  B = B{[6..10]}

  Delayed goals:
disjunction(A{[1..5]}, 5, B{[6..10]}, 7, 1)
B{[6..10]} - A{[1..5]}#>=5   % toto ohraničenie bolo vnútené na základe
                             % nastavenia príznaku Flag na 1
  yes.
```

V tomto prípade vidno, že ohraničenie vnútilo usporiadanie operácií A a B podľa daného príznaku tak, že najprv musí byť vykonaná A až potom B. To sa prejavilo jednak v zúžení domén začiatočných časov operácií A na 1 až 5 a B na 6 až 10, ako aj ďalším uspaným ohraničením B - A #>= 5 (viď delayed goals). To je nutné, nakoľko ešte stále hodnoty A, B nie sú celkom presne známe, je tam niekoľko možností výsledku.

V prípade že sa hodnota príznaku Flag nastaví na 2 (t.j. najprv sa musí vykonať operácia B, až potom A):

```
[eclipse 7]: [A, B]::1..10, disjunction(A, 5, B, 7, 2).

  A = A{[8..10]}
  B = B{[1..3]}

  Delayed goals:
disjunction(A{[8..10]}, 5, B{[1..3]}, 7, 2)
A{[8..10]} - B{[1..3]} #>=7  % toto ohraničenie bolo vnútené na základe
                             % nastavenia príznaku Flag na 2
  yes.
```

Vnútené usporiadanie operácií v poradí B, A sa prejavilo jednak v zúžení domén začiatočných časov operácií A na 8 až 10 a B na 1 až 3, ako aj ďalším pozastaveným ohraničením A - B #>= 7 (viď. delayed goals).

Skúsme teraz použiť ten istý príklad, ale počiatočné domény premenných A, B zúžime na 1 až 5.

```
[eclipse 8]: [A, B]::1..5, disjunction(A, 5, B, 7, F).

  no (more) solution.
```

V tomto prípade bola hneď odhalená nekonzistentnosť, teda že neexistujú také hodnoty A, B z ich počiatočných domén, aby bolo možné splniť dané disjunktné ohraničenie.

5.2.2 Hranová B-konzistencia

Hranová B-konzistencia (popísaná podrobnejšie v časti 2.3) sa sústreďuje iba na propagáciu horných a dolných hraníc jednotlivých doménových premenných, teda nie ich vnútorných prvkov, ako tomu bolo u úplnej hranovej konzistencie.

Aby som dosiahol tento spôsob propagácie, využil som tie nástroje jazyka *ECL^{i}PS^e*, ktoré umožňujú naprogramovať priamo spôsob, akým sa má definované ohraničenie správať v jednotlivých situáciách (kedy sa bude budiť, ako bude testovať a upravovať domény zainteresovaných premenných atď.). Za týmto účelom bolo vytvorené ohraničenie disjunctionB/5, ktoré realizuje hranovú B konzistenciu. Jeho argumenty sú presne také isté ako u ohraničenia disjunction/5 popísaného

v predchádzajúcej časti 5.2.1. Podrobne okomentovaný zdrojový text tohoto ohraničenia v jazyku *ECL^{i}PS^e* možno nájsť v programovej prílohe P1.

Ďalším implementovaným ohraničením s rovnakou funkcionalitou ako disjunctionB/5 je disjunction_choose/5, ktoré bolo vytvorené kvôli komplexnejšiemu typu ohraničenia disjunctive/3 (viď. ďalej v časti 5.3.2) v jazyku *C*. Zastrešuje ho len krátka časť programu v *ECL^{i}PS^e*, ktorá zabezpečuje prípravu argumentu príznaku a správne nastavenie podmienok aktivovania tohoto ohraničenia.

```
disjunction_choose(X1,D1,X2,D2,F) :-
        Temp1 :: 1..2,
        F = Temp1, % potrebné kvôli volaniu funkcie v C
        disjunction_choose_1(X1,D1,X2,D2,F).

disjunction_choose_1(X1,D1,X2,D2,F) :-
        disjunction_choose_(X1,D1,X2,D2,F,List), % volanie funkcie v C
        handle_requests(List), % spracovanie požiadaviek na zmeny
        suspend(disjunction_choose_1(X1,D1,X2,D2,F),4, [X1,X2] -> min),
        suspend(disjunction_choose_1(X1,D1,X2,D2,F),4, [X1,X2] -> max),
        suspend(disjunction_choose_1(X1,D1,X2,D2,F),4, F -> inst),
        % uspanie cieľa s prioritou 4 a zároveň aj stanovenie, kedy sa má tento
        % cieľ budiť, t.j. pri zmene dolnej (hornej) hranice domén X1, X2,
        % a pri naviazaní premennej F.
```

Správanie sa oboch uvedených verzií hranovej B-konzistencie pre disjunktné ohraničenia je prirodzene rovnaké a možno ho porovnať s úplnou konzistenciou na tom istom príklade, ktorý som uviedol v predchádzajúcej časti 5.2.1. Tento krát budeme používať volanie disjunction_choose/5.

```
[eclipse 5]: [A, B]::1..10, disjunction_choose(A, 5, B, 7, F).

  A = A{[1..10]}
  B = B{[1..10]}
  F = F{[1, 2]}

  Delayed goals:
disjunction_choose_1(A{[1..10]}, 5, B{[1..10]}, 7, F{[1, 2]})
  yes.
```

V prvom prípade na rozdiel od úplnej hranovej konzistencie nedochádza tento krát k redukcii vnútri domén, takže disjunction_choose/5 v prvom prípade neodvodí žiadnu dodatočnú informáciu a vedie len k pozastaveniu ohraničenia.

```
[eclipse 6]: [A, B]::1..10, disjunction_choose(A, 5, B, 7, 1).

A = A{[1..5]}
B = B{[6..10]}

Delayed goals:
```

```
A{[6..10]} - B{[1..5]}#>=5  % toto ohraničenie bolo vnútené na základe
                            % nastavenia príznaku Flag na 1
disjunction_choose_1(A{[1..5]}, 5, B{[6..10]}, 7, 1)
  yes.
```

V druhom prípade však už dochádza k výsledku zhodnému s úplnou hranovou konzistenciou, nakoľko vnútené usporiadanie operácií A a B prostredníctvom 1 na mieste príznaku Flag vedie k zmene hornej hranice domény A a dolnej hranice domény B. Rovnako ako predtým pribúda aj uspané ohraničenie, ktoré definuje toto usporiadanie (A - B #>=5 je jeho vnútorný tvar).

```
[eclipse 8]: [A, B]::1..5, disjunction_choose(A, 5, B, 7, F).

no (more) solution.
```

Analogicky v poslednom prípade hranová B-konzistencia postačuje na odhalenie nekonzistentnosti.

5.2.3 Výsledky testov

Implementované algoritmy hranovej konzistencie pre disjunktné ohraničenia som testoval na skupine náhodne generovaných úloh typu job-shop rôznej zložitosti (viď. podrobnejší opis v časti 6.1), ako aj na reálnej úlohe návrhu časového harmonogramu výstavby päťsegmentového mostu (viď. 6.2).

Pre každú z implementovaných metód som porovnával dosiahnutý čas v sekundách potrebný na nájdenie optimálneho riešenia (stĺpce s označením čas0 až čas2) a počet návratov, ktoré boli na to potrebné (stĺpce s označením PN0 až PN2).

Pre jednotlivé úlohy som testoval tri rôzne verzie programu. Východiskom bol program, ktorý implementoval disjunkciu len ako body výberu. Výsledky dosiahnuté týmto najjednoduchším postupom sú uvedené v prvých dvoch stĺpcoch tabuľky 3 (čas0 a PN0).

K tomuto základnému spôsobu reprezentácie disjunkcií som pridal najprv úplnú hranovú konzistenciu (disjunction/5). Dosiahnuté výsledky sú zachytené v stĺpcoch čas1 a PN1 tabuľky 3. Hrubo vyznačené sú najlepšie dosiahnuté hodnoty. Všetky testy uvedené v tejto práci boli vykonané na počítači PC s procesorom AMD5x86-P75 133 MHz so 16 MB RAM pod operačným systémom Linux.

Nakoniec posledné dva stĺpce (čas2 a PN2) tabuľky 3 zodpovedajú hranovej B-konzistencii implementovanej pomocou algoritmu disjunction_choose/5.

	len body výberu		**disjunction/5**		**disjunction_choose/5**	
úloha	čas0 (s)	PN0	čas1 (s)	PN1	čas2 (s)	PN2
5_5_0	**1.43**	1102	19.54	**19**	2.80	**19**
5_5_1	**1.59**	623	26.70	**8**	3.80	**8**
5_5_2	4.29	7326	25.78	**252**	**4.01**	**252**

5_5_3	**1.28**	1206	16.72	**12**	2.35	**12**
5_5_4	**1.48**	997	19.36	**0**	2.73	**0**
5_5_5	3.28	7131	17.61	**0**	**2.50**	**0**
5_5_6	**2.71**	3779	24.26	**130**	3.78	**130**
5_5_7	**2.11**	1725	24.85	**65**	3.69	**65**
5_5_8	**1.67**	1569	20.17	**8**	2.86	321
5_5_9	**3.34**	4116	32.98	**181**	4.84	**181**
8_8_0	-	-	1966.12	**29424**	**318.38**	**29424**
8_8_1	-	-	-	-	**7570.03**	**868961**
8_8_2	-	-	-	-	**12802.50**	**1163518**
8_8_3	-	-	1752.52	**36541**	**370.89**	**36541**
8_8_4	-	-	766.29	**41957**	**168.52**	**41957**
8_8_5	-	-	7541.67	**236349**	**1740.16**	**236349**
8_8_6	-	-	946.41	**7820**	**136.12**	**7820**
8_8_7	-	-	5283.74	**381805**	**1350.06**	**381805**
8_8_8	-	-	2502.23	**80372**	**551.53**	**80372**
8_8_9	-	-	4999.74	**80263**	**980.23**	**80263**
most	92.25	124008	74.18	**2232**	**19.77**	**2232**

Tabuľka 3 Výsledky testov disjunkcie ako nezávislých alternatív a hranovej konzistencie na úlohách typu job-shop a úlohe návrhu časového harmonogramu výstavby mostu.

Z výsledkov vyplýva niekoľko dôležitých záverov. Pre úlohy menšieho rozsahu (napr. úlohy typu job-shop pre 5 strojov a 5 výrobkov) je použitie dodatočných spôsobov reprezentácie disjunktných ohraničení z časového hľadiska zbytočné. Čas potrebný na propagáciu ohraničení sa vyrovná času potrebnému na prehľadanie väčšieho priestoru u najjednoduchšieho spôsobu reprezentácie disjunkcií ako bodov výberu.

Naproti tomu použitie hranovej konzistencie sa výraznejšie prejavuje s nárastom zložitosti úloh. Napríklad u úloh typu job-shop pre 8 strojov a 8 výrobkov sa už pri použití najjednoduchšieho spôsobu reprezentácie optimálneho riešenia nedočkáme ani za niekoľko hodín. U týchto úloh strednej zložitosti obvykle pomôže použiť reprezentáciu disjunkcií pomocou hranovej konzistencie.

Použitie hranovej konzistencie, ako to vyplýva z dosiahnutých výsledkov, sa v podstate vyplatí iba pre hranovú B-konzistenciu. Tá bola dostatočná pre nájdenie optimálnych riešení u všetkých testovaných úloh job-shop 8 x 8. Avšak u väčších úloh ako boli job-shop 10 x 10 už ani použitie tohoto typu reprezentácie nebolo dostatočné pre nájdenie optimálneho riešenia.

Prekvapujúcim je aj zistenie, že použitie úplnej hranovej konzistencie pre testované úlohy neprinieslo s výnimkou jednej úlohy (5_5_8) žiadne zmenšenie priestoru prehľadávania. Z toho vyplýva, že redukcie vo vnútri domén premenných sa pri úlohách rozvrhovania výraznejšie neprejavujú, a teda čas potrebný na najpresnejšiu

propagáciu ohraničení (pomocou úplnej hranovej konzistencie) sa v konečnom dôsledku neoplatí investovať. Naopak hranová B-konzistencia sa ukázala rovnako efektívna v zmenšovaní priestoru prehľadávania, pričom však potrebuje omnoho menej času na propagáciu.

Zaujímavé sú aj výsledky pre úlohu nájdenia harmonogramu výstavby päťsegmentového mosta. Tu sa v podstate potvrdzujú všetky doteraz urobené závery. Avšak časy uvedené v tabuľke sú len časy do okamžiku nájdenia optimálneho riešenia (na rozdiel od údajov k úlohám typu job-shop v tabuľke 3, ktoré zahrňujú aj dôkaz optimálnosti, t.j. prehľadanie zvyšnej časti priestoru s podmienkou lepšej hodnoty optimalizačnej funkcie). Tieto sú veľmi povzbudivé. Avšak na to, aby sa aj dokázala optimálnosť nájdeného riešenia, už nestačil ani jeden z týchto spôsobov reprezentácie disjunktných ohraničení pomocou hranovej konzistencie. Optimálnosť sa podarilo dokázať až ďalším implementovaným algoritmom, ktorý je popísaný v nasledujúcej časti 5.3.

5.3 Postup vychádzajúci z práce Carlier a Pinson

V tejto časti sa budem venovať tým myšlienkam z najznámejšieho algoritmu pre riešenie úloh typu job-shop pochádzajúcemu z oblasti operačného výskumu [CarPin 89], ktoré sú využiteľné v prostredí CLP.

Hneď v úvode by som rád upozornil, že algoritmus popísaný v [CarPin 89] vychádza z reprezentácie úlohy typu job-shop v podobe disjunktného grafu (operácie sú uzly, orientované hrany reprezentujú precedencie) a presne popisuje algoritmus branch-and-bound. Z pohľadu CLP majú najväčší význam poznatky, ktoré boli odvodené pre operácie zdieľajúce ten istý zdroj. Takéto operácie totiž vytvárajú v grafe kliku a identifikovaním podmnožiny jej potenciálnych vstupov (resp. výstupov) možno znížiť počet vetiev v priestore prehľadávania. Tento prístup k disjunktným ohraničeniam spadá do skupiny 4 spomínanej v prehľadovej časti práce (2.3).

Tieto poznatky bolo možné využiť aj v prostredí CLP. Najskôr ich bolo nutné preformulovať z ich pôvodnej (grafovej) podoby do podoby prezentovanej v teoretickej časti 5.3.1. Potom bolo potrebné implementovať ich do podoby symbolického ohraničenia disjunctive/3 v CLP jazyku *ECL^{i}PS^e* (podrobnejší popis implementácie je uvedený v časti 5.3.2)

Počas môjho desaťmesačného štipendijného pobytu na ECRC (European Computer-Industry Research Centre) v Mníchove som sa venoval práve problematike implementácie základných poznatkov z [CarPin 89] do CLP jazyka *ECL^{i}PS^e* vo forme zabudovaných symbolických ohraničení. Výsledkom bolo zavŕšenie tohoto procesu, ukončenie implementácie, dôkladné testovanie a vyladenie vytvorených ohraničení zásluhou niektorých vylepšení, ako aj vypracovanie dokumentácie k týmto ohraničeniam, ktoré sa objavili ako súčasť poslednej verzie *v3.5.2* systému *ECL^{i}PS^e*.

5.3.1 Teoretické východiská

Carlier a Pinson popísali v roku 1990 prístupy, pomocou ktorých sa podarilo nájsť optimálne riešenie dlho odolávajúceho testovacieho príkladu rozvrhovacej úlohy typu job-shop pre 10 výrobkov a 10 strojov, popísaného v [MuтTho 63].

Ich algoritmus uvažuje oddelene jednotlivé stroje, pre ktoré je schopný odhaliť čiastkové usporiadania operácií zdieľajúcich ten istý stroj a následne podľa nich upraviť dolné a horné hranice týchto operácií. To je prístup, ktorý spadá do štvrtej skupiny metód na riešenie disjunkcií popísaných v časti 2.3.

Vychádzajúc z článku [CarPin 89] možno tam uvedené poznatky preformulovať nasledujúcim spôsobom. Použijem nasledovné označenia:

$$start\_min(\Omega) = min(s_1, s_2, \ldots, s_n),\ s_i = start\_min(o_i)$$

$$koniec\_max(\Omega) = max(e_1, e_2, \ldots, e_n),\ e_i = koniec\_max(o_i),$$

$$trvanie(\Omega) = \sum_{i=1}^{n} Trvanie(o_i)$$

kde $\Omega = \{o_i : i \in I\}$ označuje množinu operácií, o_i a o jednotlivé operácie, ktoré zdieľajú ten istý zdroj. $start\_min(o_i)$ je najskorší možný čas začiatku, $koniec\_max(o_i)$ najneskorší možný čas ukončenia a $trvanie\ (o_i)$ trvanie operácie o_i.

Carlier a Pinson definovali nasledujúce tri podmienky, ktoré sú predpokladom ďalších odvodení o čiastočnom usporiadaní operácií. U všetkých predpokladáme, že operácie o a Ω zdieľajú ten istý zdroj.

Podmienka 1.

$start\_min(o) + trvanie(o) + trvanie(\Omega) > koniec\_max(\Omega)$ $\quad pre\ \{o\} \cap \Omega = \varnothing$

Podmienka 2.

$start\_min(\Omega) + trvanie(\Omega) + trvanie(o) > koniec\_max(o)$ $\quad pre\ \{o\} \cap \Omega = \varnothing$

Podmienka 3.

$start\_min(\Omega) + trvanie(\Omega) + trvanie(o) > koniec\_max(\Omega)$ $\quad pre\ \{o\} \cap \Omega = \varnothing$

Vychádzajúc z týchto troch podmienok platia nasledujúce tri vety.

Veta 1.

Ak platí podmienka 1, potom operácia o nemôže byť vykonaná pred množinou operácií Ω.

Veta 2.

Ak platí podmienka 2, potom operácia o nemôže byť vykonaná po množine operácií Ω.

Veta 3.

Ak platí podmienka 3, potom operácia *o* môže byť vykonaná len pred alebo po množine operácií Ω.

Spojením týchto tvrdení (keďže operácia *o* môže byť vykonávaná len pred, v priebehu alebo po vykonaní operácií z množiny Ω) dostávame nasledujúce dve vety.

Veta 4.

Ak platia podmienky 1 a 3, potom operácia *o* musí byť vykonaná až po množine operácií Ω. Možno teda upraviť hranice trvania *o* podľa nasledujúceho vzťahu:

$$start\_min(o) \geq start\_min(\Omega) + trvanie(\Omega)$$

Veta 5.

Ak platia podmienky 2 a 3, potom operácia *o* musí byť vykonaná pred množinou operácií Ω. Možno teda upraviť hranice trvania *o* podľa nasledujúceho vzťahu:

$$koniec\_max(o) \leq koniec\_max(\Omega) - trvanie(\Omega)$$

Naviac platia ešte dve ďalšie vety pre úpravu hraníc trvania operácie o.

Veta 6.

Nech je množina operácií Ω usporiadaná vzostupne podľa počiatočných časov: $\Omega = \{o_1, o_2, \ldots, o_n\}$. Ak operácia *o* musí byť vykonaná pred množinou operácií Ω, potom:

$$koniec\_max(o) \leq Min_{i=1}^{n}\{koniec\_max(o_i) - \sum_{j=1}^{i} trvanie(o_j)\}$$

Veta 7.

Nech je množina operácií Ω usporiadaná vzostupne podľa počiatočných časov: $\Omega = \{o_1, o_2, \ldots, o_n\}$. Ak operácia *o* musí byť vykonaná po množine operácií Ω, potom:

$$start\_min(o) \geq Max_{i=1}^{n}\{start\_min(o_i) + \sum_{j=i}^{n} trvanie(o_j)\}$$

Z uvedeného vyplýva, že vymenovaním všetkých dvojíc (o, Ω) a testovaním splnenia podmienok 1 až 3 možno niektoré operácie usporiadať pred, resp. po množine iných operácií. To vedie následne k úprave dolných, resp. horných hraníc niektorých z nich, čím sa redukuje priestor prehľadávania.

Avšak všetkých dvojíc (o, Ω) je $n*2^{n-1}$ (podľa binomickej vety počet všetkých podmnožín *n* prvkovej množiny je 2^n, t.j. ak jeden zdroj je zdieľaný *n* operáciami a zároveň dvojica o, Ω musí byť disjunktná, potom pre každý z *n* prvkov máme 2^{n-1} možných podmnožín Ω). Autori spomínajú algoritmus zložitosti $o(n^4)$, ktorý nájde všetky potrebné dvojice. Pre samotnú implementáciu bol použitý algoritmus zložitosti

$o(n^3)$[15], ktorý vychádza z toho, že stačí generovať len najväčšie maximálne podmnožiny Ω. Nie je potrebné skúmať podmnožiny Ω_k ($\Omega_k \subset \Omega$) také, u ktorých $start\_min(\Omega) = start\_min(\Omega_k)$ a $koniec\_max(\Omega) = koniec\_max(\Omega_k)$. Postup je nasledovný:

1. Usporiadaj operácie lexikograficky vzostupne vzhľadom na *(start_min(o), koniec_max(o))* a vlož do zoznamu: zložitosť $o(n*log(n))$
2. Usporiadaj všetky hodnoty *koniec_max(o)* do zoznamu: zložitosť $o(n*log(n))$
3. Generuj maximálne podmnožiny vloženým prezeraním vygenerovaných dvoch zoznamov: zložitosť $o(n^2)$

Keďže tento postup je potrebné zopakovať pre všetkých n strojov, dostávame výslednú zložitosť algoritmu $o(n^3)$.

Na tomto mieste by som rád podotkol, že najcennejšie na tomto algoritme sú práve redukcie domén premenných, teda úprava ich dolných, resp. horných hraníc. Tu sa však dá vyťažiť ešte o čosi viac ak si uvedomíme, že čiastočné usporiadanie operácie voči množine iných operácií môže viesť nielen k redukcii domény tejto operácie, ale prejavuje sa aj v stanovení poradia medzi dvojicami operácií. Za týmto účelom sa sleduje aj zoznam príznakov, ktoré definujú precedenčné vzťahy medzi každou dvojicou operácií zdieľajúcich ten istý zdroj.

5.3.2 Popis implementácie

Základné poznatky z algoritmu Carlier a Pinson popísané podrobnejšie v predchádzajúcej časti boli implementované v podobe ohraničenia disjunctive/3 do systému ECL^iPS^e.

Toto ohraničenie má tri argumenty. Prvým je zoznam doménových premenných reprezentujúcich počiatočné časy všetkých operácií zdieľajúcich ten istý zdroj. Druhý argument predstavuje zoznam trvaní všetkých týchto operácií (prirodzené čísla). Nakoniec tretí argument obsahuje zoznam dvojhodnotových príznakov (spolu ich je n^2), ktoré reprezentujú čiastočné usporiadania jednotlivých dvojíc spomínaných operácií (ak je na mieste príznaku týkajúceho sa operácií *I* a *J* jednotka, znamená to, že vo výslednom rozvrhu bude operácia *I* vykonaná pred operáciou *J*. Ak je na tomto mieste 2, potom naopak operácia *J* bude vykonaná pred *I* vo výslednom rozvrhu).

Toto ohraničenie implementuje všetky posuny hraníc vyplývajúce z viet 4 až 7 uvedených v predchádzajúcej časti. Množina príznakov usporiadaní jednotlivých dvojíc operácií sa využíva jednak na zaznamenanie relatívnej pozície daných dvoch operácií v aktuálnom rozvrhu, ale aj na ovplyvnenie a modifikáciu čiastkových rozvrhov tým, že sa vnúti usporiadanie dvojice operácií nastavením ich príznaku. Nasleduje podrobnejší popis programu, ktorý túto činnosť realizuje.

[15] Na tomto mieste je však nutné spomenúť aj fakt, že medzičasom boli autormi [CarPin89] nájdené už aj efektívnejšie algoritmy, posledný z nich má zložitosť už len $o(n*log(n))$ [CarPin94].

Zastrešujúca časť disjunctive/3 je implementovaná v *ECL^{i}PS^e*, avšak rozhodujúca časť algoritmu bola naprogramovaná v jazyku *C* a je volaná (*ECL^{i}PS^e* ponúka pomerne jednoduchý spôsob rozhrania k jazyku *C*) zo spomínanej zastrešujúcej časti programu:

```
disjunctive(Starts,Durations,Flags) :-
        length(Starts,Ls),
        length(Durations,Ld),
        Ls = Ld,
        % nasleduje vytvorenie ďalej používaných štruktúr
        SStarts =.. [starts|Starts], % pole počiatočných časov
        SDurations =.. [durations|Durations], % pole trvaní
        SS is Ls * Ls,
        functor(SFlags,flags,SS), % pole príznakov usporiadaní
        setup_disjunctions(Starts,Durations,Flags,Sflags),
            % elementárne disjunkcie monitorujúce dvojice operácií
            % (nižšie možno nájsť presný popis)
        disjunctive(SStarts,SDurations,Flags,SFlags).
            % ohraničenie realizujúce úpravy podľa viet 4 až 7
```

Samotné volanie disjunctive/4 zastrešuje jadro funkcie v *jazyku C* a obhospodaruje zmeny hraníc a príznakov, ktoré spustenie algoritmu Carlier a Pinson prinesie.

```
disjunctive(Starts,Durations,Flags,SFlags) :-
        disjunctive_(Starts,Durations,SFlags,List), % volanie funkcie v C
        handle_requests(List), % spracuj požiadavky na zmeny
        suspend(disjunctive(Starts,Durations,Flags,SFlags),4,Flags -> inst).
            % pozastavenie cieľa s prioritou 4 a predpísanie, kedy sa má
            % tento cieľ budiť, t.j. pri naviazaní niektorého z príznakov Flags
```

Ako vyplýva z uvedeného zdrojového textu programu v *ECL^{i}PS^e*, po vyhodnotení ohraničenia disjunctive/4 a vykonaní ním navrhnutých zmien je nutné tento cieľ s upravenými argumentmi opäť pozastaviť s tým, že ohraničenie sa opäť vyvolá v okamžiku, keď dôjde k nejakému ďalšiemu čiastkovému usporiadaniu (teda keď sa naviaže niektorý z príznakov Flags).

Volanie setup_disjunctions/4 slúži na sledovanie elementárnych disjunkcií medzi dvojicami operácií zdieľajúcich ten istý zdroj. Informácia o ich vzájomnom usporiadaní sa prenáša prostredníctvom príznakov (Flags).

```
setup_disjunctions(Starts,Durations,Flags,SFlags) :-
        length(Starts, Arity), % zistenie dĺžky zoznamu Starts
        setup_disjunctions(Starts,Durations,Flags,SFlags,0,1,Arity).

setup_disjunctions([],[],[],_,_,_,_).
setup_disjunctions([S|Ss],[D|Ds],Flags,SFlags,I,J,Arity) :-
        setup_disjunctions(Ss,S,Ds,D,Flags,FlagsT,SFlags,I,J,Arity),
        I1 is I + 1,
        J1 is J + 1,
        setup_disjunctions(Ss,Ds,FlagsT,SFlags,I1,J1,Arity).
```

```
setup_disjunctions([],_,[],_,Flags,Flags,_,_,_,_).
setup_disjunctions([S|Ss],SS,[D|Ds],DD,[F|Fs],FsO,SFlags,I,J,A) :-
        F :: 1..2,
        disjunction_choose(SS,DD,S,D,F), % elementárna disjunkcia
        P is I*A + J + 1,  % pozícia zodpovedajúceho príznaku v poli
        arg(P,SFlags,F), % vlož príznak na správne miesto v poli
        J1 is J + 1,
        setup_disjunctions(Ss,SS,Ds,DD,Fs,FsO,SFlags,I,J1,A)
```

V tejto časti je podstatným volanie disjunction_choose/5 pre každú dvojicu operácií zdieľajúcich ten istý zdroj a nastavenie im zodpovedajúceho príznaku (F) aj do zoznamu všetkých príznakov (SFlags). Jadro funkcie disjunction_choose/5 je opäť realizované v *C* (podrobnejšie viď predchádzajúca časť 5.2.2).

5.3.3 Výsledky testov

Implementovaný algoritmus disjunctive/3 vychádzajúci z algoritmu Carlier a Pinson som testoval na skupine náhodne generovaných úloh typu job-shop rôznej zložitosti[16] (viď. podrobnejší opis v časti 6.1.1), ako aj reálnej úlohe návrhu časového harmonogramu výstavby päťsegmentového mostu (viď. 6.2).

Ohraničenie disjunctive/3 som porovnával s najlepším z algoritmov hranovej konzistencie testovaných v predchádzajúcej časti 5.2.3, a síce hranovej B-konzistencie realizovanej ohraničením disjunction_choose/5. Pre oba prípady som opäť porovnával dosiahnutý čas potrebný na nájdenie optimálneho riešenia (stĺpce s označením čas0, čas1) a počet návratov, ktoré boli na to potrebné (stĺpce s označením PN0 a PN1). Výsledky sú zhrnuté v tabuľke 4.

Posledné dva stĺpce tabuľky 4 reprezentujú pomer dosiahnutých časov (čas0/čas1) a počtov návratov (PN0/PN1) medzi algoritmom s využitím ohraničenia disjunction_choose/5 a algoritmom s využitím disjunctive/3.

[16] Príklady zodpovedajú tým, ktoré boli testované v predchádzajúcej časti 5.2.3, na rovnakom počítači.

úloha	disjunction_choose/5 čas0 (s)	PN0	disjunctive/3 čas1 (s)	PN1	čas0 / čas1	PN0 / PN1
5_5_0	**2.80**	19	3.89	**2**	0.72	9.5
5_5_1	**3.80**	8	5.47	**0**	0.69	-
5_5_2	**4.01**	252	4.02	**11**	0.99	22.3
5_5_3	**2.35**	12	3.29	**1**	0.71	12.0
5_5_4	**2.73**	0	4.15	**0**	0.66	-
5_5_5	**2.50**	0	4.01	**0**	0.62	-
5_5_6	**3.78**	130	4.19	**13**	0.90	10.0
5_5_7	**3.69**	65	4.46	**0**	0.83	-
5_5_8	**2.86**	321	4.40	**2**	0.65	160.5
5_5_9	**4.84**	181	6.07	**24**	0.80	7.5
8_8_0	318.38	29424	**69.10**	**72**	4.61	408.7
8_8_1	7570.03	868961	**662.54**	**33140**	11.42	26.2
8_8_2	12802.50	1163518	**275.12**	**5010**	46.53	232.2
8_8_3	370.89	36541	**152.38**	**5038**	2.43	7.3
8_8_4	168.52	41957	**87.63**	**3629**	1.92	11.6
8_8_5	1740.16	236349	**141.15**	**7757**	12.33	30.5
8_8_6	136.12	7820	**113.78**	**246**	1.20	31.8
8_8_7	1350.06	381805	**127.46**	**923**	10.59	413.6
8_8_8	551.53	80372	**75.72**	**732**	71.44	109.8
8_8_9	980.23	80263	**171.54**	**2303**	5.71	34.8
10_10_0	-	-	**838.72**	**8930**	-	-
10_10_1	7600.33	555882	**630.62**	**10152**	12.05	54.7
10_10_2	-	-	**1358.61**	**20230**	-	-
10_10_3	-	-	**1268.20**	**19271**	-	-
10_10_4	-	-	**2490.90**	**1114045**	-	-
10_10_5	9356.55	1009329	**490.45**	**8450**	19.08	119.4
10_10_6	454.43	33507	**163.60**	**1425**	2.78	23.5
10_10_7	-	-	**7728.37**	**99100**	-	-
10_10_8	-	-	**735.09**	**27043**	-	-
10_10_9	-	-	**7863.94**	**110430**	-	-
most	19.77	2232	**2.92**	**624**	6.77	3.6

Tabuľka 4 Výsledky testov reprezentácie disjunkcie pomocou hranovej konzistencie a ohraničením disjunctive/3.

Výsledky sú do istej miery analogické výsledkom z predchádzajúcej časti 5.2.3. Zložitejší spôsob reprezentácie disjunkcií priniesol určitú časovú stratu u najjednoduchších úloh, kde čas potrebný na propagáciu ohraničení je príliš dlhý.

Naopak pri väčších úlohách sa informácia získaná propagáciou ohraničenia disjunctive/3 mnohonásobne vráti a umožní vyriešiť aj väčšie úlohy typu job-shop pre 10 strojov a 10 výrobkov, ktoré sa nepodarilo vyriešiť pomocou hranovej konzistencie.

Vo všetkých prípadoch je však zjavné zmenšenie priestoru prehľadávania, ktoré sa prejavuje v počte návratov. Vo všetkých testovaných úlohách bol počet návratov pri použití algoritmu disjunctive/3 niekoľkokrát menší (pre úlohy job-shop 7,3 až 413,6 krát menej návratov, pre úlohu stavby mostu 3,6 krát menej - viď. tabuľka 4) než pri použití algoritmu disjunction_choose/5).

Pre úlohu návrhu časového harmonogramu výstavby päťsegmentového mostu sú v tabuľke 4 opäť kvôli porovnaniu uvedené len časy potrebné pre nájdenie optimálneho riešenia, nie však dôkaz optimálnosti. Ten bolo možné vykonať len s použitím ohraničenia disjunctive/3, ktorý na to potreboval celkovo 28821.2 sekúnd (t.j. 8 hodín a 21 minút).

5.4 Intervaly úloh (Task Intervals)

Metódu spracovanú v tejto časti možno zaradiť do tej istej skupiny ako predchádzajúcu, založenú na [CarPin 89]. Postup nazvaný intervaly úloh po prvýkrát navrhli Caseau a Laburthe v [CasLab 93]. Jedna z ich posledných prác [CasLab 95] popisuje kompletný systém na riešenie rozvrhovacích úloh s disjunktnými ohraničeniami naprogramovaný v jazyku *CLAIRE*, ktorý je však bližší skôr *C++*.

Z pohľadu CLP sú však najzaujímavejšie práve redundantné ohraničenia vyplývajúce zo špeciálnej štruktúry zvanej interval úloh. Tieto je totiž možné implementovať v prostredí CLP v podobe symbolického ohraničenia, podobne ako tomu bolo u disjunctive/3. V nasledujúcej časti popisujem podrobne tie ohraničenia, ktoré boli implementované v jazyku *ECL^{i}PS^e* a zastrešené opäť jedným symbolickým ohraničením task_intervals/3.

5.4.1 Teoretické východiská

Francúzski autori Caseau a Laburthe v snahe vyhnúť sa generovaniu obrovského množstva dvojíc (o,Ω) potrebných podľa prístupu definovaného v [CarPin 89], navrhli iný prístup [CasLab 95], v ktorom každej dvojici operácií (A, B) zdieľajúcich ten istý zdroj priradili množinu takých operácií $K_{(A,B)}$ ktoré budú určite rozvrhnuté medzi najskorším možným časom začiatku A (t.j. *start_min(A))* a najneskorším možným časom ukončenia B (t.j. *koniec_max(B)):*

$$K_{(A,B)} = \{C \,/\, start\_min(A) \leq start\_min(C)\,,\; koniec\_max(C) \leq koniec\_max(B)\}$$

$K_{(A,B)}$ sa nazýva **interval úloh** (odtiaľ aj názov celej metódy) a uvažuje sa len v prípade, že operácie A a B patria do $K_{(A,B)}$. Potom pre každú operáciu C ležiacu v $K_{(A,B)}$ existuje niekoľko pravidiel, ktoré určujú, kedy C musí byť vykonaná pred (alebo po) všetkých ostatných operáciách v $K_{(A,B)}$.

Podotýkam, že pre n operácií a m strojov existuje najviac $m*n^2$ rôznych intervalov úloh. Vyplýva to z faktu, že pre jeden stroj, ktorý zdieľa maximálne n operácií,

existuje maximálne *n* navzájom rôznych dolných hraníc intervalov (najskoršie možné časy začiatkov jednotlivých operácií) a *n* horných (najpozdnejšie možné časy ukončenia jednotlivých operácií), teda ich kombinácií je maximálne *n*n*. Pri *m* rôznych strojoch to teda činí maximálne spomínaných $m*n^2$ rôznych intervalov úloh.

Použité označenia sú rovnaké ako v predchádzajúcich častiach, naviac ešte pribudne $\Delta(K_{(A,B)})$, čo bude označovať voľnú časť intervalu úloh $K_{(A,B)}$, t.j.

$$\Delta(K_{(A,B)}) = koniec\_max(K_{(A,B)}) - start\_min(K_{(A,B)}) - trvanie(K_{(A,B)})$$

V ďalšom uvediem pravidlá, podľa ktorých je možné propagovať (šíriť) zmeny v doménach jednotlivých premenných prostredníctvom skupiny ohraničení aj na domény ďalších premenných. Pravidlá možno rozdeliť do štyroch skupín:

1. **Základné pravidlo** odhaľuje nekonzistentnosť celého intervalu úloh, a to v prípade, ak nie je splnené nasledovné ohraničenie:

$$koniec\_max(K_{(A,B)}) - start\_min(K_{(A,B)}) \geq trvanie(K_{(A,B)})$$

2. Druhá skupina pravidiel sú tzv. **pravidlá usporiadania**, nakoľko upravujú hranice domén začiatkov jednotlivých operácií podľa daných, resp. v priebehu prehľadávania zistených precedenčných vzťahov medzi dvojicami operácií. Nasledujúca dvojica ohraničení platí pre každú dvojicu operácií zdieľajúcich ten istý zdroj (plnia funkciu hranovej B-konzistencie):

$$\forall A, B, (A \prec B \wedge start\_min(B) < start\_min(A) + trvanie(A)) \Rightarrow$$
$$start\_min(B) = start\_min(A) + trvanie(A)$$
$$\forall A, B, (A \prec B \wedge koniec\_max(A) > start\_min(B) + trvanie(B)) \Rightarrow$$
$$koniec\_max(A) = start\_min(B) + trvanie(B)$$

3. Ďalšiu skupinu tvoria **pravidlá zisťovania hrán**, ktoré určujú či daná operácia z intervalu úloh môže byť prvá, resp. posledná medzi operáciami z tohto intervalu.

$$\forall C, K_{(A,B)}, (C \in K_{(A,B)} \wedge koniec\_max(K_{(A,B)}) - trvanie(K_{(A,B)}) < start\_min(C))$$
$$\Rightarrow start\_min(C) \geq \min\{start\_min(o_i) + trvanie(o_i), o_i \in K_{(A,B)} - \{C\}\}$$
$$\forall C, K_{(A,B)}, (C \in K_{(A,B)} \wedge koniec\_max(K_{(A,B)}) - trvanie(C) > koniec\_max(C))$$
$$\Rightarrow koniec\_max(C) \leq \max\{start\_min(o_i) + trvanie(o_i), o_i \in K_{(A,B)} - \{C\}\}$$

Tieto pravidlá majú však jednu nevýhodu, a síce že vyžadujú časovo náročný výpočet výrazu $op\{start\_min(o_i) + trvanie(o_i), o_i \in K_{(A,B)} - \{C\}\}$ (kde *op* znamená min, resp. max). Naviac, častokrát tento výpočet ešte nemusí viesť k úprave domény premennej pre operáciu *C*. Aby sa tento výpočet nemusel robiť zbytočne, zvykne sa pridať ešte jeden predpoklad, a síce, že

$start\_min(C) < start\_min(A) + trvanie(A)$ v prvom pravidle, resp.

$koniec\_max(C) > koniec\_max(B) - trvanie(B)$ v pravidle druhom.

4. Posledná skupina pravidiel sa snaží usporadúvať operácie voči celým intervalom úloh, t.j. rozpoznať prípady kedy daná operácia, ktorá nepatrí do intervalu úloh ale

zdieľa ten istý zdroj, musí byť vykonaná pred, resp. po tomto intervale úloh. Ide o tzv. **vylučovacie pravidlá**.

$$\forall C, K_{(A,B)}, (C \notin K_{(A,B)} \wedge$$
$$koniec\_max(K_{(A,B)}) - trvanie(K_{(A,B)}) - trvanie(C) < start\_min(C)) \Rightarrow$$

ak

$$koniec\_max(K_{(A,B)}) - start\_min(K_{(A,B)}) > trvanie(K_{(A,B)}) + trvanie(C) \vee$$
$$start\_min(C) + trvanie(C) > \max\{koniec\_max(o_i) + trvanie(o_i), o_i \in K_{(A,B)}\}$$

potom

$$start\_min(C) \geq start\_min(K_{(A,B)}) + trvanie(K_{(A,B)}) \wedge$$
$$\forall o_i \in K_{(A,B)}, koniec\_max(o_i) \leq koniec\_max(C) - trvanie(C)$$

ináč

$$start\_min(C) \geq \min\{start\_min(o_i) + trvanie(o_i), o_i \in K_{(A,B)}\}$$

Analogicky by sa dalo napísať symetrické pravidlo pre prípad, že daná operácia $C \notin K_{(A,B)}$ musí byť vykonaná až po všetkých operáciách z intervalu úloh $K_{(A,B)}$.

Pre správnu a čo možno najefektívnejšiu propagáciu uvedených ohraničení je ešte dôležité stanoviť, kedy majú byť aktivované. Základné pravidlo sa aktivuje pri každej zmene intervalu úloh $K_{(A,B)}$ (vznik, pridanie alebo ubranie úlohy). Pravidlá usporiadania sa aktivujú pre každej zmene *start_min(C)*, resp. *koniec_max(C)*, ako aj pri zistení usporiadania dvoch operácií $A \prec B$. Pravidlá zisťovania hrán sa aktivujú pri zmenách *start_min(C)*, *koniec_max*($K_{(A,B)}$), ale aj *start_min*($K_{(A,B)}$). Rovnako sa aktivujú aj pravidlá vylučovacie, tie sú naviac ešte citlivé na zmeny počtu operácií v $K_{(A,B)}$.

Intervaly úloh si teda vyžadujú definíciu a správu nových údajových štruktúr. Zmeny v doménach počiatočných časov jednotlivých operácií preto vedú nielen k aktivácii spomínaných ohraničení, ale môžu spôsobiť aj zmeny samotných intervalov úloh. Pri zmene dolnej hranice domény niektorej operácie, t.j. ak sa zväčší *start_min(C)* z hodnoty h_1 na h_2, je potrebné dodržať nasledovný postup úpravy zainteresovaných intervalov úloh:

- deaktivovať tie intervaly úloh $K_{(C,B)}$, pre ktoré $h_2 > start\_min(B)$
- odstrániť operácie *D* z aktívnych intervalov úloh $K_{(C,B)}$, ak $start\_min(D) < h_2$
- vytvoriť nové aktívne intervaly úloh $K_{(A,C)}$, ak $h_1 < start\_min(A) \leq h_2$ a $koniec\_max(A) < koniec\_max(C)$
- pridať operáciu *C* do aktívnych intervalov úloh $K_{(A,B)}$, ak $h_1 < start\_min(A) \leq h_2$ a $koniec\_max(C) < koniec\_max(B)$

Až po vykonaní týchto úprav v množine aktívnych intervalov úloh môžeme spustiť propagáciu vyššie uvedených pravidiel.

5.4.2 Poznámky k implementácii

Na tomto mieste nebudem uvádzať presný popis implementácie, ktorý možno nájsť v [Schmotzer 97], len niekoľko dôležitých poznámok k existujúcemu programu.

Intervaly úloh boli implementované (podobne ako v predchádzajúcich prípadoch) vo forme symbolického ohraničenia task_intervals/3 v jazyku *ECL^{i}PS^e*. V práci [CasLab 95] ide totiž o kompletný systém na riešenie úloh typu job-shop, z ktorého som vybral len časť zaoberajúcu sa redundantnými ohraničeniami na báze štruktúry intervalu úloh, ktoré vedú k lepšej propagácii ohraničení a tým k redukcii priestoru prehľadávania.

Na rozdiel od implementácie prístupov z algoritmu Carlier a Pinson nebola tento krát využitá väzba na jazyk C a celý program je na úrovni jazyka *ECL^{i}PS^e*. To samozrejme prináša menšiu efektívnosť programu než by pravdepodobne bolo možné dosiahnuť implementáciou na úrovni jazyka C. Pre zodpovedajúce porovnanie s ohraničením disjunctive/3 by bolo potrebné preniesť podobným spôsobom rozhodujúcu časť programu narábajúcu so špeciálnymi štruktúrami do jazyka C a v *ECL^{i}PS^e* potom už len zabezpečiť správnu aktiváciu ohraničenia. Napriek tejto skutočnosti je možné urobiť niektoré závery porovnaním dosiahnutých výsledkov (viď. nasledujúca časť 5.4.3)

5.4.3 Výsledky testov

V tabuľke 5 sú uvedené výsledky testov pre tie isté náhodne generované úlohy typu job-shop a reálnu úlohu pre nájdenie časového harmonogramu výstavby päťsegmentového mostu ako v predchádzajúcich častiach. Pre porovnanie som použil výsledky dosiahnuté použitím ohraničenia disjunctive/3, pričom som testoval samostatné použitie ohraničenia task_intervals/3, ako aj súčasné použitie disjunctive/3 a task_intervals/3. Sledoval som opäť dva ukazovatele, a síce čas výpočtu a počet návratov, ktoré dávajú informáciu o veľkosti tej časti priestoru prehľadávania, ktorá musela byť preskúmaná.

Pre optimalizáciu bol použitý algoritmus LOGARITMICKÝ MINIMIZE a odhady hraníc podľa postupov navrhnutých v kapitole 7.

úloha	disjunctive/3 čas1 (s)	disjunctive/3 PN1	task_intervals/3 čas2 (s)	task_intervals/3 PN2	disjunctive/3 + task_intervals/3 čas3 (s)	disjunctive/3 + task_intervals/3 PN3
5_5_0	**0.81**	**104**	5.99	110	4.08	**104**
5_5_1	**0.35**	**0**	1.22	**0**	1.08	**0**
5_5_2	**1.04**	64	5.72	63	3.78	**61**
5_5_3	**0.83**	**104**	5.82	116	3.52	**104**
5_5_4	**0.51**	**100**	3.62	**100**	2.39	**100**
5_5_5	**0.46**	**100**	2.48	**100**	2.01	**100**
5_5_6	**0.86**	**59**	6.45	109	3.11	**59**
5_5_7	**0.35**	**0**	1.38	**0**	1.12	**0**
5_5_8	**0.52**	**52**	3.39	55	2.41	**52**
5_5_9	**1.19**	185	8.83	190	7.08	**183**
8_8_0	**10.25**	746	76.10	1993	33.44	**702**
8_8_1	**5.89**	319	65.71	543	22.92	**313**
8_8_2	**32.65**	1238	267.89	2277	82.65	**1195**
8_8_3	**9.99**	744	156.81	944	42.01	**735**
8_8_4	**5.47**	**676**	38.16	783	22.31	**676**
8_8_5	**21.51**	873	231.75	1979	67.86	**784**
8_8_6	**3.41**	**452**	28.3	454	17.61	**452**
8_8_7	**11.01**	755	131.07	1122	37.98	**751**
8_8_8	**12.50**	604	145.81	898	40.24	**585**
8_8_9	**30.94**	1073	458.42	3375	102.72	**914**
10_10_0	**108.28**	2130	3373.45	15507	229.23	**1952**
10_10_1	**32.65**	883	893.72	136634	106.09	**875**
10_10_2	**351.28**	8037	6320.07	32384	890.66	**7021**
10_10_3	**289.85**	4851	5133.50	21733	574.53	**4098**
10_10_4	**156.26**	4506	4002.98	27303	434.08	**3894**
10_10_5	**56.30**	3286	615.56	4221	170.33	**3092**
10_10_6	**18.68**	1043	396.22	3461	75.86	**1042**
10_10_7	**8932.44**	167274	-	-	17866.80	**131630**
10_10_8	**89.90**	2577	1541.93	6763	326.85	**2443**
10_10_9	**1228.24**	29234	23631.30	173603	2097.65	**22994**
most	32730.90	1427795	2669.87	55035	**1717.67**	**39130**

Tabuľka 5 Výsledky testov pre disjunkcie pomocou ohraničenia disjunctive/3, task_intervals/3 a ich kombinácie.

Z výsledkov možno vidieť, že samotné intervaly úloh implementované vo forme ohraničenia task_intervals/3 nepriniesli zmenšenie priestoru prehľadávania v porovnaní s ohraničením disjunctive/3. Rovnako časová náročnosť výpočtu bola podstatne väčšia, čo sa však dalo očakávať vzhľadom na skutočnosť, že ohraničenie

task_intervals/3 bolo implementované kompletne na úrovni jazyka *ECL^{i}PS^e*, kým jadro ohraničenia disjunctive/3 bolo implementované v jazyku C.

Jedinou výnimkou je reálna úloha rozvrhu prác na výstavbe mosta, kde bola zaznamenaná výrazná redukcia priestoru prehľadávania, ako aj času výpočtu. Ukázalo sa, že v tejto úlohe ohraničenie task_intervals/3 výrazne napomohlo v poslednej fáze riešenia, t.j. pri dôkaze optimálnosti.

Kombinácia oboch ohraničení priniesla prakticky vo všetkých prípadoch zmenšenie priestoru prehľadávania (nikdy nie zväčšenie). Avšak čas výpočtu sa tým zakaždým predĺžil vzhľadom na čas strávený propagáciou ohraničení. Opäť výnimkou bola reálna úloha rozvrhu prác na výstavbe mostu, kde sa touto kombináciou dosiahol najlepší výsledok, keď celkový výpočet vrátane dôkazu optimálnosti sa skrátil 19 krát na necelú polhodinu. Preskúmaný priestor prehľadávania sa zmenšil 36 krát oproti použitiu samotného ohraničenia disjunctive/3. Podrobnejšie sú výsledky pre túto úlohu rozobrané v časti 6.2.3.

K a p i t o l a 6

PRAKTICKÁ REALIZÁCIA A OVERENIE

6.1 Typické úlohy rozvrhovania “job-shop”

Úlohy typu job-shop boli podrobnejšie popísané v časti 2.1.1. Pre účely testovania a vyhodnocovania úspešnosti jednotlivých prístupov k spracovaniu a propagácii disjunktných ohraničení som používal množinu náhodne vygenerovaných úloh typu job-shop rôznej zložitosti.

V tejto časti bude podrobnejšie popísaná reprezentácia týchto úloh, ako aj program, ktorý bol použitý na ich riešenie.

6.1.1 Popis použitých úloh typu job-shop

Zvolil som tri skupiny úloh rôzneho rozmeru po 10 úlohách z každej skupiny. Prvá skupina boli úlohy rozmeru 5 x 5 (teda 5 výrobkov a 5 strojov), druhá skupina bola tvorená úlohami 8 x 8 a nakoniec tretiu skupinu tvorili úlohy rozmeru 10 x 10 (čiže 10 výrobkov na 10 strojov).

Ďalej sa budem odkazovať na jednotlivé úlohy jednoznačným kódom *X_Y_*0 až *X_Y_*9, kde čísla *X* (počet výrobkov) a *Y* (počet strojov) reprezentujú rozmer úlohy, teda v našom prípade *X* = *Y* budú mať hodnotu 5, 8 alebo 10. Ako som spomínal, z každého rozmeru bolo vygenerovaných 10 navzájom rôznych úloh, čomu zodpovedá posledná cifra (0 až 9).

Reprezentáciu úlohy vysvetlím na jednoduchšom príklade rozmeru 5 x 5 (konkrétne úloha s označením 5_5_0). Každý súbor s testovacími údajmi začína faktom typu test_data(TypUlohy, C1, C2, PocetStrojov, PocetVyrobkov) popisujúcim danú inštanciu riešenej úlohy, napr.

```
test_data(job_shop, 527556884, 1343124817, 5, 5).
```

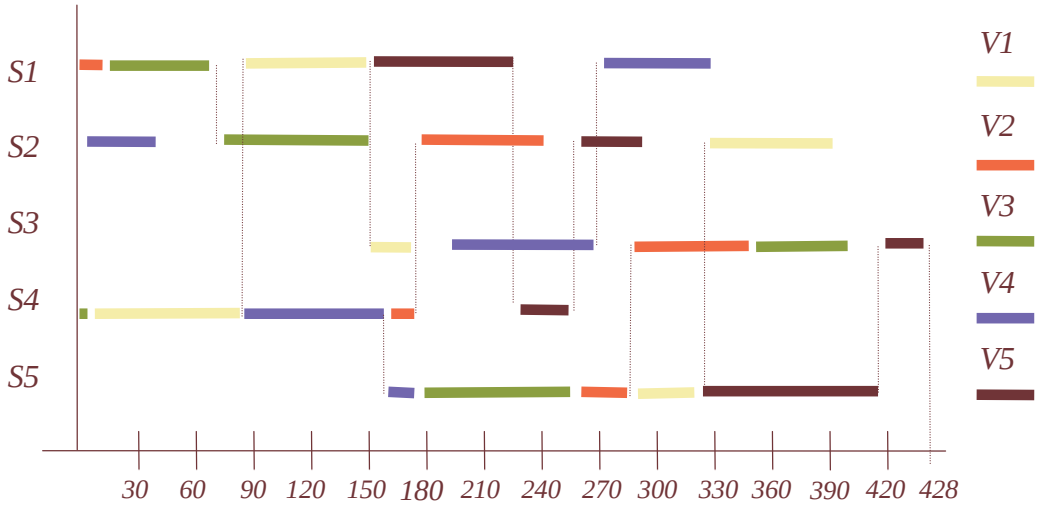
Kde C1 a C2 sú náhodne vygenerované postupnosti cifier. Potom nasleduje rozpis jednotlivých výrobkov na operácie. Ide o zoznam všetkých operácií, pričom každá operácia je reprezentovaná štruktúrou tvaru t(Cvyr, Cop, Cstr, Trv), kde:

Cvyr - číslo výrobku
Cop - číslo operácie
Cstr - číslo stroja, na ktorom má byť operácia vykonaná
Trv - trvanie operácie.

Pre vybranú úlohu *5_5_0* vyzerá táto štruktúra nasledovne:

job_shop_data(5, 5, [t(1, 1, 4, 85), t(2, 1, 1, 7), t(3, 1, 4, 1), t(4, 1, 2, 45), t(5, 1, 1, 80), t(1, 2, 1, 64), t(2, 2, 4, 14), t(3, 2, 1, 74), t(4, 2, 4, 76), t(5, 2, 4, 15), t(1, 3, 3, 31), t(2, 3, 2, 69), t(3, 3, 2, 70), t(4, 3, 5, 13), t(5, 3, 2, 45), t(1, 4, 5, 44), t(2, 4, 5, 18), t(3, 4, 5, 90), t(4, 4, 3, 98), t(5, 4, 5, 91), t(1, 5, 2, 66), t(2, 5, 3, 68), t(3, 5, 3, 60), t(4, 5, 1, 54), t(5, 5, 3, 10)]).

Pre lepšiu názornosť na tomto mieste uvádzam aj riešenie tejto konkrétnej úlohy job-shop 5_5_0 vo forme časového diagramu na obrázku 4. Na horizontálnej osi *x* v tomto diagrame je čas, na osi *y* sú jednotlivé stroje (*S1* až *S5*). Jednotlivé operácie toho istého výrobku sú označené tou istou farbou (viď legenda obrázku 4 vpravo). Každá úsečka teda reprezentuje časové trvanie jednej operácie s jej presným umiestnením vo výslednom optimálnom rozvrhu. Ako je zrejmé z obrázku, optimálne riešenie tejto úlohy má trvanie 428 časových jednotiek (najkratší možný čas, kedy sa dá stihnúť výroba pri splnení všetkých zadaných ohraničení). Operácie, ktoré na seba naväzujú sú naviac na svojich hranách spojené prerušovanou čiarou.



Obrázok 4 Časový diagram znázorňujúci optimálny rozvrh pre úlohu job-shop *5_5_0*.

Podrobnejší popis programu implementujúceho riešenie úloh typu job-shop s jednotlivými spôsobmi riešenia disjunktných ohraničení je uvedený v programovej prílohe P.2.

6.1.2 Výsledky testov

Výsledky dosiahnuté pre jednotlivé spôsoby reprezentácie disjunktných ohraničení možno nájsť v tabuľkách 3, 4, 5 a 11. Na tomto mieste iba uvediem základné závery, ktoré boli naznačené už v častiach 5.2.3 (tabuľka 3 - algoritmy hranovej konzistencie), 5.3.3 (tabuľka 4 - ohraničenie disjunctive/3 na báze Carlier a Pinson) a 5.4.3 (tabuľka 5 - ohraničenie task_intervals/3 na báze intervalov úloh), ako aj ďalšie aspekty, ktoré je nutné brať do úvahy pri úlohách typu job-shop.

Pre úlohy typu job-shop menšieho rozmeru (t.j. cca do 5 stroj a 5 výrobkov)[17], je úplne postačujúce použitie základného spôsobu reprezentácie disjunkcií ako nezávislých alternatív. So stúpajúcou zložitosťou úloh je potrebné pridať inteligentnejšie spôsoby reprezentácie a propagácie disjunktných ohraničení, v nasledujúcom poradí účinnosti:

1. úplná hranová konzistencia (disjunction/5),
2. hranová B-konzistencia (disjunction_choose/5),
3. intervaly úloh (task_intervals/3),
4. algoritmus disjunctive/3 (implementácia algoritmu Carlier a Pinson).

Pritom pre úlohy stredného rozsahu (cca 8 výrobkov a 8 strojov) je ešte postačujúca hranová B-konzistencia, pre väčšie úlohy (napr. rozmer 10 x 10 a viac) už len task_intervals/3, resp. disjunctive/3, prípadne ich kombinácia. Použitie úplnej hranovej konzistencie, ako to vyplynulo z testov sa však v porovnaní s hranovou B-konzistenciou sa nevyplatí, nakoľko je časovo náročnejšie a neprináša výraznejšie zmenšenie priestoru prehľadávania (z testovaných úloh len v jednom prípade).

Výsledky testov uvedené v tabuľkách 3 a 4 boli dosiahnuté pre volanie

```
:- jobshop(1000).
```

To ale znamenalo pre jednotlivé úlohy rôznu vzdialenosť od optimálneho riešenia (viď. tabuľka 10). Táto skutočnosť sa čiastočne podpísala na predĺžení výpočtového času u úloh, ktorých optimum ležalo výraznejšie pod hranicou 1000 časových jednotiek.

S cieľom znížiť počet iterácií algoritmu MINMAX (popis možno nájsť v 2.4), boli navrhnuté algoritmy LOGARITMICKÝ MINMAX a LOGARITMICKÝ MINIMIZE (podrobnejšie viď. 7.2), ktoré vychádzajúc z heuristicky nájdených odhadov dolnej a hornej hranice intervalu v ktorom leží optimálne riešenie, postupujú jeho delením na polovice, čím sa počet nevyhnutných iterácií podstatne zníži.

Tieto metódy priniesli zároveň najlepšie výsledky, ktoré sú zhrnuté v tabuľke 11. V drvivej väčšine úloh je efektívnosť tohoto postupu obrovská. Zostávajú však úlohy s veľmi zložitým dôkazom optimálnosti (ak čas strávený dôkazom optimálnosti zaberá rozhodujúcu časť celkového času výpočtu) u ktorých tento postup nie je oveľa lepší ako klasický MINMAX (alebo môže byť dokonca o niečo horší, ako tomu bolo pri úlohe most).

Avšak aj v týchto prípadoch je nutné uviesť, že algoritmus v každom kroku dáva presné vyčíslenie maximálnej vzdialenosti doposiaľ nájdeného riešenia od optima (v percentách), čo umožňuje kedykoľvek ukončiť výpočet, keď je používateľ už spokojný s dosiahnutou kvalitou riešenia a nechce čakať na lepšie, resp. na ukončenie dôkazu optimálnosti nájdeného riešenia.

Pri testovaní ohraničenia task_intervals/3 už bola použitá metóda LOGARITMICKÝ MINIMIZE (tabuľka 5). Na základe vykonaných testov nemožno doporučiť samostatné použitie tohoto ohraničenia v súčasnej podobe, iba jeho kombináciu

[17] Samozrejme tu je potrebné brať do úvahy aj výkonnosť počítača a efektívnosť konkrétneho CLP systému.

s ohraničením disjunctive/3, kedy prináša zmenšenie preskúmanej časti priestoru prehľadávania, ale predlžuje výpočet.

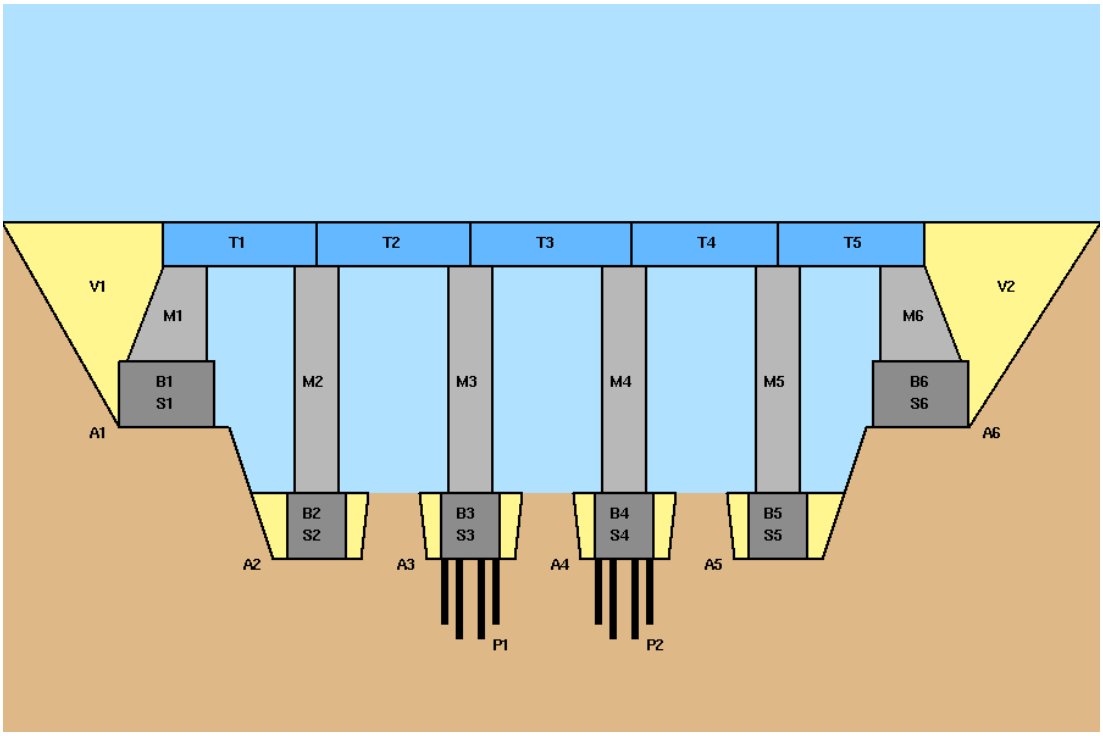
6.2 Návrh časového harmonogramu výstavby mostu

Ďalšou reálnou úlohou bude rozvrhovanie prác potrebných na výstavbu päťsegmentového mosta.

6.2.1 Popis úlohy

V tejto časti je uvedený popis úlohy pre nájdenie minimálneho časového rozvrhu prác na stavbe päťsegmentového mosta (viď. obrázok 5), ktorá bola uvedená v Bartuschovej dizertačnej práci v roku 1983 a jej popis som prebral z [Hentenryck 89].

Tento projekt pozostáva zo 42 úloh (pre každú je definované jej trvanie a zdroje potrebné pre jej vykonanie - viď tabuľka 6) a množiny ohraničení. Pre stavbu je k dispozícii po jednom z každého z definovaných zdrojov, t.j. jedno rýpadlo, baranidlo, žeriav, miešačka betónu a pásové vozidlo, ako aj jedna tesárska a jedna murárska čata.



Obrázok 5 Schéma päťsegmentového mosta s vyznačením jednotlivých úloh.

Okrem tradičných **precedenčných ohraničení** (ako napr. že výkop pre daný pilier musí predchádzať betonárskym prácam na nej) musia byť splnené aj **disjunktné ohraničenia** vyplývajúce z obmedzeného počtu zdrojov, ktoré musia byť zdieľané pri viacerých úlohách (napríklad že jediná betonárska čata ktorá je k dispozícii, môže v jednom okamihu pracovať maximálne na betónovaní jedného piliera) a ešte nasledujúca skupina **dodatočných ohraničení**:

1. Medzi ukončením daného debnenia a zodpovedajúcim betónovaním základov môžu uplynúť nanajvýš 4 dni.
2. Nanajvýš 3 dni môžu uplynúť medzi ukončením výkopu a začatím osadzovania debnenia v ňom.
3. Práce na debnení môžu začať najskôr 6 dní po začiatku montáže dočasného plášťa.
4. Odstránenie dočasného plášťa môže začať dva dni pred ukončením posledných murárskych prác.
5. Dodávka vyhotovených nosníkov sa uskutoční presne 30 dní po začiatku projektu.

Nasleduje tabuľka 6 obsahujúca popis všetkých úloh (prác) na projekte, s uvedením časov trvania a potrebných zdrojov. Posledný stĺpec obsahuje čas začiatku jednotlivých úloh pri optimálnom rozvrhu (posledná, fiktívna úloha Koniec s trvaním 0 symbolizuje ukončenie celej stavby).

č.	názov	popis	trvanie	zdroje	zač.
1	A1	výkop (podpera 1)	4	rýpadlo	3
2	A2	výkop (pilier 1)	2	rýpadlo	1
3	A3	výkop (pilier 2)	2	rýpadlo	7
4	A4	výkop (pilier 3)	2	rýpadlo	24
5	A5	výkop (pilier 4)	2	rýpadlo	28
6	A6	výkop (podpera 2)	5	rýpadlo	38
7	P1	základy piliera 2	20	baranidlo	9
8	P2	základy piliera 3	13	baranidlo	29
9	UE	montáž dočasného plášťa	10	-	0
10	S1	debnenie (podpera 1)	8	tesárska čata	10
11	S2	debnenie (pilier 1)	4	tesárska čata	6
12	S3	debnenie (pilier 2)	4	tesárska čata	29
13	S4	debnenie (pilier 3)	4	tesárska čata	42
14	S5	debnenie (pilier 4)	4	tesárska čata	33
15	S6	debnenie (podpera 2)	10	tesárska čata	46
16	B1	betónový základ (podpera 1)	1	miešačka betónu	18
17	B2	betónový základ (pilier 1)	1	miešačka betónu	10
18	B3	betónový základ (pilier 2)	1	miešačka betónu	33
19	B4	betónový základ (pilier 3)	1	miešačka betónu	46
20	B5	betónový základ (pilier 4)	1	miešačka betónu	37
21	B6	betónový základ (podpera 2)	1	miešačka betónu	56
22	AB1	doba tvrdnutia betónu (pilier 1)	1	-	19
23	AB2	doba tvrdnutia betónu (podpera 1)	1	-	11
24	AB3	doba tvrdnutia betónu (podpera 2)	1	-	34
25	AB4	doba tvrdnutia betónu (podpera 3)	1	-	47
26	AB5	doba tvrdnutia betónu (podpera 4)	1	-	38
27	AB6	doba tvrdnutia betónu (pilier 2)	1	-	57
28	M1	murárske práce (podpera 1)	16	murárska čata	20
29	M2	murárske práce (pilier 1)	8	murárska čata	12
30	M3	murárske práce (pilier 2)	8	murárska čata	36
31	M4	murárske práce (pilier 3)	8	murárska čata	52
32	M5	murárske práce (pilier 4)	8	murárska čata	44
33	M6	murárske práce (podpera 2)	16	murárska čata	60
34	DE	dodávka vyhotovených nosníkov	2	žeriav	30
35	T1	osadenie nosníka 1	12	žeriav	36
36	T2	osadenie nosníka 2	12	žeriav	48
37	T3	osadenie nosníka 3	12	žeriav	88
38	T4	osadenie nosníka 4	12	žeriav	60
39	T5	osadenie nosníka 5	12	žeriav	76
40	UA	odstránenie dočasného plášťa	10	-	74
41	V1	násyp 1	15	pásové vozidlo	48
42	V2	násyp 2	10	pásové vozidlo	88
43	Koniec	ukončenie projektu	0	-	100

Tabuľka 6 Údaje o jednotlivých úlohách v probléme stavby päťsegmentového mosta.

6.2.2 Popis programu

V tejto časti je uvedený náčrt programu pre nájdenie minimálneho časového rozvrhu prác na stavbe päťsegmentového mosta popísanej v predchádzajúcej časti.

Pre každú úlohu I definujeme štruktúru vo forme trojice [ZdrojI, StartI, TrvanieI] označujúcej zdroj ktorý daná úloha využíva (konštanta), dĺžku jej trvania (konštanta) a deň začiatku jej vykonávania (premenná s konečnou doménou). Začiatok celého projektu bude v čase 0 a jeho ukončenie v dni Koniec (premenná, ktorej hodnota po naviazaní bude rovná času ukončenia poslednej z vykonávaných úloh). Úlohou je teda nájsť také priradenia hodnôt premenným StartI u jednotlivých úloh, aby boli splnené všetky v predchádzajúcej časti uvedené ohraničenia a aby sa minimalizovalo trvanie celého projektu Koniec. Základná schéma programu, ktorý toto zabezpečí, je nasledovná.

```
most(L, Koniec) :-
        zadanie_ulohy(L),
        definuj_domeny(L, Koniec),
        stanov_ohraničenia(L),
        minimalizuj(vyber(L), Koniec).
```

Najskôr sa načítajú vstupné údaje týkajúce sa úloh, ktoré majú byť vykonané v rámci projektu a na ich základe sa vytvorí zoznam štruktúr L jednotlivých úloh (predikát zadanie_ulohy(L)).

V ďalšom kroku (predikát definuj_domeny(L, Koniec)) sa priradia premenným označujúcim začiatok vykonávania úlohy (v štruktúre vyššie označené ako StartI), ako aj premennej Koniec východzie domény, t.j. východzí rozsah, v rámci ktorého sa určite bude vyskytovať skutočný čas začiatku tej ktorej úlohy vo výslednom rozvrhu. Túto doménu môžeme na začiatku jednoznačne obmedziť na interval ohraničený 0 (začiatok projektu) a hodnotou, ktorú dostaneme spočítaním dĺžok trvania všetkých úloh (to by zodpovedalo najjednoduchšej možnej realizácii projektu ako postupnosti všetkých úloh za sebou, bez akejkoľvek súbežnosti), zväčšenú o nutné odstupy definované v dodatočných ohraničeniach. Označme takto vypočítanú hodnotu Max_trvanie. Potom stanovenie domény bude v CLP programe zachytené zápisom

```
StartI :: 0..Max_ trvanie
```

Na takto inicializované doménové premenné možno teraz aplikovať ohraničenia z definície úlohy. V rámci predikátu stanov_ohraničenia(L) sa definujú všetky potrebné ohraničenia. V danej úlohe ide v zásade o tri druhy ohraničení potrebných pre špecifikáciu úlohy.

1. Precedenčné ohraničenia

Skutočnosť, že úloha [ZdrojI, StartI, TrvanieI] musí byť vykonaná pred úlohou [ZdrojJ, StartJ, TrvanieJ] možno v CLP vyjadriť zápisom

```
StartJ #>= StartI + TrvanieI
```

2. Dodatočné ohraničenia

Tieto ohraničenia reprezentujú v danej úlohe nutný odstup medzi niektorými úlohami (všetkých 5 takýchto ohraničení možno nájsť vymenovaných pri podrobnom popise úlohy v predchádzajúcej časti 6.2.1) možno vyjadriť veľmi podobne ako precedenčné ohraničenia.

Skutočnosť, že napr. úloha [ZdrojI, StartI, TrvanieI] môže začať až 10 dní po skončení úlohy [ZdrojJ, StartJ, TrvanieJ] možno v CLP vyjadriť zápisom:

```
StartI #>= StartJ + TrvanieJ + 10
```

Úplný zoznam a popis všetkých možných časových relácií medzi dvoma úlohami podľa [Allen 83] možno nájsť v časti 8.1.3 v kapitole venovanej metodológii.

3. Disjunktné ohraničenia:

Tieto ohraničenia, ako už bolo spomínané, vyplývajú zo zdieľania tých istých zdrojov rôznymi úlohami. Takže pre každú dvojicu úloh [ZdrojI, StartI, TrvanieI] a [ZdrojJ, StartJ, TrvanieJ] zdieľajúcich ten istý zdroj, t.j. ak ZdrojI = ZdrojJ máme dve možnosti. Alebo bude v rozvrhu najprv úloha I, a potom úloha J nemôže začať skôr kým neskončí I, alebo opačne (t.j. buď daný zdroj pridelíme najprv úlohe I a potom J, alebo naopak). Túto skutočnosť možno reprezentovať známym predikátom disjunktne/4.

```
disjunktne(StartI, StartJ, TrvanieI, TrvanieJ) :-
        StartJ #>= StartI + TrvanieI.
disjunktne(StartI, StartJ, TrvanieI, TrvanieJ) :-
        StartI #>= StartJ + TrvanieJ.
```

Predikát disjunktne/4 je súčasťou definície predikátu stanov_disjunkcie(L) vystupujúceho v definícii vyber(L, Koniec):

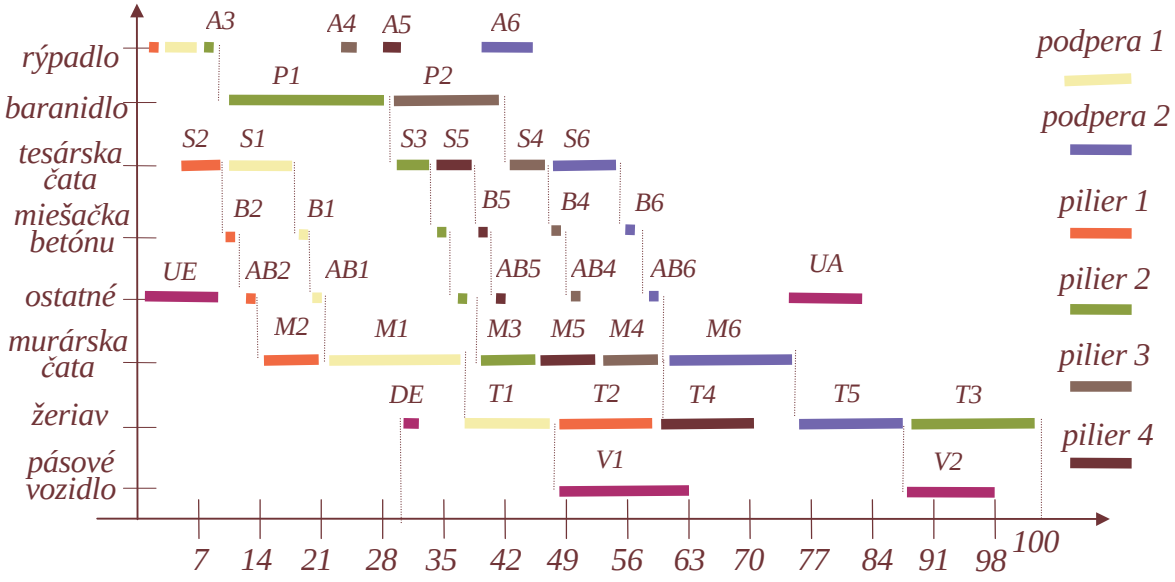
```
vyber(L) :-
        stanov_disjunkcie(L),
        zvol_hodnotu(L).
```

Predikát vyber(L, Koniec) okrem stanovenia disjunktných ohraničení naviac štartuje aj prehľadávanie tým, že zvolí hodnoty z aktuálnych domén premenných pre časy začiatku jednotlivých úloh v L (riadenie prehľadávania je v súčasných CLP systémoch podporované rôznymi zabudovanými predikátmi).

Posledným krokom v nami definovanej úlohe je nájsť optimálne riešenie, čo je v našom prípade riešenie s minimálnou dĺžkou projektu Koniec. Túto funkciu zabezpečuje predikát minimalizuj(vyber(L), Koniec) v hlavnom programe, ktorý pre daný cieľ (jeho prvý argument, t.j. v našom prípade vyber(L)) nájde také riešenie, pre ktoré hodnota druhého argumentu (v našom prípade Koniec) je minimálna. Na mieste predikátu minimalizuj/2 možno použiť zabudovaný predikát min_max/2, prípadne novo navrhnuté predikáty log_min_max/5 a log_minimize/5.

6.2.3 Výsledky

Najkratší možný harmonogram prác na výstavbe päťsegmentového mosta podľa definície úlohy v 6.2.1 je 100 dní. Časy začiatkov jednotlivých úloh pre tento optimálny rozvrh možno nájsť v poslednom stĺpci tabuľky 5. Grafický diagram optimálneho harmonogramu je uvedený na obrázku 10, pričom na horizontálnej osi je čas (v dňoch) a na vertikálnej osi sú jednotlivé zdieľané zdroje. Každej operácii zodpovedá jedna úsečka reprezentujúca časový interval, kedy má byť operácia vykonaná. Jej vertikálna poloha definuje používaný zdroj. Pre lepšiu názornosť sú práce na jednotlivých segmentoch mostu vyznačené rovnakou farbou. Úlohy, ktoré na seba naväzujú sú naviac na svojich hranách spojené prerušovanou čiarou.



Obrázok 6 Časový diagram optimálneho harmonogramu výstavby päťsegmentového mostu.

Porovnávacie výsledky pre jednotlivé spôsoby reprezentácie a propagácie disjunktných ohraničení pre túto úlohu možno nájsť na príslušných miestach v kapitole 5 (5.2.3, 5.3.3 a 5.4.3). Na tomto mieste uvediem súhrn týchto výsledkov (viď tabuľka 7).

Nájdenie optimálneho riešenia trvalo 92.25 sekúnd najjednoduchšou metódou reprezentácie disjunkcií ako nezávislých alternatív. Tento spôsob zároveň potreboval najviac návratov (124 008). Časy pre nájdenie optimálneho riešenia ďalšími spôsobmi reprezentácie a propagácie disjunktných ohraničení sa postupne skracovali, spolu s klesajúcim počtom návratov, a to v nasledovnom poradí (viď. tabuľka 7):

- úplná hranová disjunkcia (disjunction/5),
- hranová B-konzistencia (disjunction_choose/5),
- algoritmus Carlier a Pinson (disjunctive/3).

úloha	len body výberu		disjunction/5		disjunction_choose/5		disjunctive/3	
	čas0	PN0	čas1	PN1	čas2	PN2	čas3	PN3
most	92.25	124008	74.18	2232	19.77	2232	**2.92**	**624**

Tabuľka 7 Riešenie disjunkcií pre úlohu rozvrhovania prác na výstavbe päťsegmentového mosta.

Nájdenie optimálneho riešenia bolo teda pomerne jednoduché a rýchle, avšak dôkaz jeho optimálnosti sa ukázal naopak veľmi zložitý. Na potvrdenie skutočnosti, že riešenie lepšie ako 100 dní naozaj neexistuje, bolo treba čakať viac ako 8 hodín pri použití disjunctive/3 (1 098 0541 návratov) a ešte viac pri použití LOGARITMICKÉHO MINMAX, resp. LOGARITMICKÉHO MINIMIZE (viď. tabuľka 8, v ktorej sú zhrnuté výsledky pre úlohu most).

úloha	min_max/2		log_min_max/5		log_minimize/5	
	čas1 (s)	PN1	čas2 (s)	PN2	čas3 (s)	PN3
most	**28821.2**	**10980541**	29103.2	1427724	32730.9	1427795

Tabuľka 8 Výsledky optimalizácie pre úlohu rozvrhovania prác na výstavbe mosta len s použitím ohraničenia disjunctive/3.

Výraznejší pokrok v tejto úlohe prinieslo až použitie ohraničenia task_intervals/3 implementujúceho postup zvaný intervaly úloh, ktorý bol popísaný podrobne v časti 5.4 a niektoré výsledky uvedené v tabuľke 5.

Na tomto mieste uvediem podrobnejšiu analýzu výsledkov dosiahnutých použitím ohraničenia task_intervals/3, resp. jeho kombinácie s ohraničením disjunctive/3 pri použití rôznych postupov pre optimalizáciu. Výsledky sú zhrnuté v tabuľke 9. Vo všetkých testoch boli počiatočné hranice stanovené postupmi navrhnutými v časti 7.1. Pre túto úlohu to znamenalo hornú hranicu (heuristicke riešenie) s nákladmi 110 a spodnú hranicu s nákladmi 92.

úloha most	min_max/2		log_min_max/5		log_minimize/5	
	čas1 (s)	PN1	čas2 (s)	PN2	čas3 (s)	PN3
po nájdenie optima (ti+d)[18]	5.53	207	**3.58**	**36**	251.98	3663
s dôkazom optimálnosti (ti+d)	**1516.82**	**35717**	1705.41	39054	1717.67	39130
po nájdenie optima (ti)	5.41	207	**3.51**	**36**	302.21	4916
s dôkazom optimálnosti (ti)	**2438.25**	**50369**	2648.78	54959	2669.87	55035

Tabuľka 9 Podrobné výsledky pre úlohu most pri použití ohraničenia task_intervals/3 a jeho kombinácie s disjunctive/3.

Vôbec najkratší čas potrebný pre riešenie tejto úlohy vrátane dôkazu optimálnosti bol dosiahnutý klasickým postupom MINMAX s použitím kombinácie redundantných ohraničení task_intervals/3 a disjunctive/3 (19 násobné zrýchlenie v porovnaní so situáciou bez použitia ohraničenia task_intervals/3). Takisto preskúmaná časť priestoru prehľadávania bola v tomto prípade najmenšia (v tomto prípade išlo až o 307 násobné zmenšenie v porovnaní s prípadom bez4 použitia task_intervals/3).

Pri porovnaní s novými optimalizačnými postupmi možno konštatovať, že LOGARITMICKÝ MINMAX najrýchlejšie našiel optimálne riešenie (zároveň s najmenším počtom návratov) a to s použitím aj bez použitia ohraničenia disjunctive/3. Nové optimalizačné postupy sú o niečo pomalšie pri riešení s dôkazom optimálnosti, nakoľko musia vykonať niekoľko pokusov aj s hranicami nižšími ako je optimum (na rozdiel od metódy MINMAX, ktorá robí len jeden pokus s hranicou o jedna menšou než je optimum). Táto situácia nastáva v prípade, keď drvivá väčšina výpočtu je strávená dôkazom optimálnosti, čo je práve v prípade tejto úlohy veľmi výrazné.

[18] Použité ohraničenia: ti znamená task_intervals/3 a d znamená disjunctive/3.

K a p i t o l a 7

NOVÉ POSTUPY PRE RIEŠENIE OPTIMALIZÁCIE V CLP

Klasické metódy hľadania optimálneho riešenia v CLP založené na algoritme branch-and-bound boli popísané v časti 2.4. Vychádzajú zo zadanej hodnoty optimalizačnej funkcie a snažia sa nájsť lepšie riešenie v zmysle optimalizačného kritéria (napríklad s nižšími nákladmi v prípade minimalizácie). Po nájdení lepšieho riešenia sa toto stáva novou hranicou (napr. hornou v prípade minimalizácie). Tento proces môže byť dosť zdĺhavý, čo je ovplyvňované najmä:

- kvalitou zadanej počiatočnej hranice pre hodnotu optimalizačnej funkcie,
- počtom riešení na ceste od počiatočnej hodnoty optimalizačnej funkcie k optimálnemu riešeniu.

Zaoberal som sa možnosťami zefektívnenia takéhoto postupu. Cieľom môjho snaženia bolo zminimalizovať vplyv oboch spomenutých faktorov. Prvý z nich je možné ovplyvniť vhodnou voľbou heuristiky, ktorá nájde rýchlo prvé riešenie ležiace čo možno najbližšie k optimu. Tomuto problému je venovaná časť 7.1.

Druhý zo spomenutých faktorov síce súvisí najmä so špecifikami riešenej úlohy, ale aj tu možno vylepšiť postup zmenenou stratégiou postupu od aktuálne najlepšieho riešenia k optimálnemu. Riešeniu tohoto problému je venovaná časť 7.2.

7.1 Odhad hornej a dolnej hranice optimálneho riešenia

V tejto časti uvediem niekoľko heuristík pre rýchle nájdenie suboptimálneho riešenia rozvrhovacích úloh. Pri použití heuristiky pre rýchle nájdenie suboptimálneho riešenia odpadá aj nutnosť zadávať počiatočnú hodnotu optimalizačnej funkcie, prípadne sa takto zadaná hodnota porovná s hodnotou optimalizačnej funkcie riešenia vygenerovaného heuristikou a zoberie sa tá z nich, ktorá lepšia.

Význam tohoto kroku narastá so zložitosťou riešenej rozvrhovacej aplikácie. Čím zložitejšia úloha, tým väčší priestor prehľadávania a tým časovo náročnejší každý krok vedúci k skráteniu výsledného rozvrhu (znižovanie hodnoty optimalizačnej funkcie).

Z toho nutne vyplýva, že čím bližšie bude prvé vygenerované riešenie k skutočnému optimálnemu riešeniu, tým väčšiu šancu máme, že sa naozaj dočkáme jeho nájdenia, resp. že nájdeme nejaké riešenie, ktoré bude dostatočne blízke optimu.

Vygenerovať počiatočné riešenie je možné pomerne ľahko deterministickým algoritmom [CasLab 95]. Rozvrh sa vytvára postupne tak, že sa vyberajú úlohy (operácie) jedna za druhou a každá z nich začne tak skoro, ako je to len možné.

V každom kroku je k dispozícii množina úloh, z ktorých je ešte možné vybrať nasledujúcu. Na začiatku sú v tejto množine všetky úlohy (operácie). V každom kroku sa vyberie jedna z týchto úloh a priradí sa jej najskorší možný začiatok. Táto úloha sa potom vyberie zo spomínanej množiny. Celý postup sa potom opakuje tak dlho, až kým sa všetky úlohy z množiny nevyberú.

Ťažisko algoritmu teda leží v pravidle, podľa ktorého sa vyberá úloha z množiny ešte nerozvrhnutých úloh. Ja som testoval nasledovné heuristiky:

FIFO vyberaj zaradom podľa poradia (úlohy sú usporiadané podľa postupnosti v rámci jednotlivých výrobkov)

EST vyber úlohu (operáciu) s najskorším možným časom začiatku

LST vyber úlohu (operáciu) s najneskorším možným časom začiatku

EFT vyber úlohu (operáciu) s najskorším možným časom ukončenia

LFT vyber úlohu (operáciu) s najneskorším časom ukončenia

SPT vyber úlohu (operáciu) s najkratším trvaním

LPT vyber úlohu (operáciu) s najdlhším trvaním

MWR vyber úlohu (operáciu) s najdlhšou zvyškovou prácou (súčet trvaní úloh, ktoré ešte musia byť vykonané za vybranou úlohou)

Na základe dosiahnutých výsledkov som navrhol novú heuristiku, ktorá spája výhody najúspešnejších heuristík z práve uvedených, a síce EST a MWKR. Túto heuristiku budem ďalej označovať **EST+**. Táto heuristika vyberá úlohu (operáciu) s najskorším možným časom začiatku a v prípade zhody u viacerých vyberie tú, u ktorej je najdlhšia zvyšková práca. Okomentovaný výpis zdrojového textu implementácie heuristiky EST+ možno nájsť v programovej prílohe P.3.

Dosiahnuté výsledky pre jednotlivé heuristiky a všetky testované úlohy typu job-shop uvádzam v nasledujúcej tabuľke 10. Hrubo vyznačené sú najlepšie dosiahnuté odhady. Pripomínam že ide o minimalizačnú úlohu (hľadáme najskorší čas ukončenia všetkých úloh podľa rozvrhu spĺňajúceho všetky zadané ohraničenia).

Z uvedenej tabuľky vyplýva, že novo navrhnutá heuristika EST+ dosiahla z 30 testovaných úloh v 14 prípadoch najlepší odhad. Ďalšia v poradí úspešnosti je heuristika EST s 9 najlepšími odhadmi a potom nasledujú FIFO a MWKR, ktoré majú po 4 najlepšie odhady. Naviac v úlohách, kedy heuristika EST+ nedosiahla najlepší odhad, jej vzdialenosť od najlepšieho odhadu bola najviac 11.2% u úloh rozmeru 5 x 5, 7.2% u úloh 8 x 8 a 3% u úloh 10 x 10 (v priemere boli však odchýlky postupne 4.76% pre úlohy 5 x 5, 2.46% pre úlohy 8 x 8 a 2.14% pre úlohy 10 x 10). Tieto výsledky boli suverénne najlepšie spomedzi všetkých testovaných heuristík.

Testoval som aj programy, kde sa generovalo viacero suboptimálnych riešení (rôznymi heuristikami) a potom sa vybralo najlepšie z nich. Časový zisk však bol minimálny, alebo dokonca žiadny, nakoľko obvykle viac času bolo potrebného na viacnásobné hľadanie heuristickeho riešenia než na nájdenie ďalších (lepších) riešení optimalizačným postupom.

Úloha	FIFO	EST	EST+	LST	LFT	SPT	LPT	MWR	opt.	dolná
5_5_0	**486**	522	492	543	551	620	508	514	428	375
5_5_1	449	**333**	**333**	480	468	378	390	391	333	255
5_5_2	579	559	**547**	603	579	603	577	549	506	482
5_5_3	**486**	**486**	517	534	534	527	544	533	461	345
5_5_4	480	**474**	**474**	598	609	598	497	526	420	321
5_5_5	538	**481**	**481**	563	698	612	566	488	390	350
5_5_6	596	480	480	547	556	566	**479**	562	455	429
5_5_7	426	426	**392**	504	567	504	513	501	392	370
5_5_8	522	522	**513**	674	559	562	608	545	502	383
5_5_9	516	514	555	514	615	**499**	646	562	479	397
8_8_0	794	**767**	776	912	1044	1015	853	807	683	572
8_8_1	705	730	698	805	892	693	801	**692**	607	524
8_8_2	832	1040	831	1088	894	1056	1037	**811**	767	614
8_8_3	869	**770**	776	923	1097	1058	904	827	683	552
8_8_4	900	884	850	**829**	993	898	912	882	703	637
8_8_5	**810**	858	835	994	1157	1021	828	873	737	536
8_8_6	724	753	**715**	946	840	875	875	847	662	524
8_8_7	772	**708**	759	994	1074	906	951	715	654	583
8_8_8	**778**	899	791	960	848	944	902	883	701	557
8_8_9	892	978	**850**	1090	920	1017	1073	911	776	615
10_10_0	1208	1193	**1029**	-	1442	1396	1245	1100	915	826
10_10_1	1100	**1031**	1057	1398	1334	1318	1147	1114	882	723
10_10_2	988	1009	**919**	1269	987	1131	979	1193	783	599
10_10_3	1023	1064	**952**	1210	1280	1160	1145	1072	857	649
10_10_4	1122	1200	**1054**	1163	1304	1232	1203	1139	902	735
10_10_5	1110	968	974	1183	1322	1219	1183	**946**	858	739
10_10_6	977	**924**	936	1210	1137	1177	1139	1057	808	685
10_10_7	1271	1277	**1095**	1123	-	1285	1292	1184	989	770
10_10_8	1096	1065	**969**	1270	1114	1212	1164	1054	847	738
10_10_9	1218	1180	1088	1312	1244	1116	1247	**1069**	881	671

Tabuľka 10 Odhady horných hraníc za pomoci jednotlivých heuristík.

Okrem odhadu hornej hranice, kde ide vlastne o nájdenie čo možno najlepšieho suboptimálneho riešenia, môže prispieť k zúženiu priestoru prehľadávania aj dobrý odhad dolnej hranice. Tu ide o odhad doby, pod ktorú sa určite nedá už rozvrh stihnúť.

Pre odhad dolnej hranice sa používa klasický postup, ktorý spočíta dĺžky trvaní jednotlivých úloh (operácií) pre jednotlivé zdroje (stroje) a pre jednotlivé výrobky. Za dolnú hranicu sa potom vyberie najdlhší z nich. Znamená to, že rozvrh nemôže byť kratší ako súčet trvaní všetkých úloh na tom zdroji, ktorý bude najviac využívaný, resp. ako súčet trvaní všetkých operácií toho výrobku, ktorý ho má najväčší. Túto

hodnotu je možné ešte zvýšiť o najskorší možný čas začiatku spomedzi všetkých úloh (operácií) na tomto najkritickejšom zdroji, resp. výrobku (ten sa už mohol v priebehu prvotnej propagácie ohraničení zvýšiť).

Ako príklad použijem úlohu job-shop *5_5_0*, ktorá bola definovaná v časti 6.1.1. Súčet trvaní operácií, ktoré majú byť vykonané na jednotlivých strojoch *S1* až *S5* v tomto poradí sú: 279, 295, 267, 191, 256. Súčet trvaní operácií pre jednotlivé výrobky *V1* až *V5* v poradí sú: 290, 176, 295, 286, 241. Z uvedených hodnôt je maximálna 295 pre výrobok *V3*. Naviac sa však ešte pripočítava najskorší možný začiatok operácie z danej množiny pre jednotlivé stroje, resp. výrobky. V tomto prípade je táto hodnota nulová okrem stroja *S1*, kde je najskorší možný začiatok 90 a *S3*, kde je to 108. Ak teda pripočítame tieto hodnoty k príslušným súčtom, dostaneme ako maximálnu z nich práve hodnotu pre stroj *S3*, ktorá bude teraz 267 + 108 = 375, čo je zároveň náš odhad dolnej hranice (viď. tabuľka 10).

Okomentovanú časť programu realizujúcu odhad dolnej hranice možno nájsť v prílohe P.3. Takto získané hodnoty dolnej hranice pre jednotlivé testované úlohy sú uvedené v poslednom stĺpci tabuľky 10.

Každé ďalšie zvýšenie dolnej hranice je už nutne spojené s úplným prehľadávaním priestoru riešení ktoré dokáže, že riešenie týmito nákladmi neexistuje. Tento proces je zhodný s hľadaním optimálneho riešenia, preto nemá zmysel venovať viac času hľadaniu lepšej dolnej hranice, ale je potrebné sústrediť sa na skracovanie heuristicky nájdeného rozvrhu (odhad hornej hranice). Tento proces je popísaný v nasledujúcej časti.

7.2 Hľadanie optimálneho riešenia

Poslednou etapou riešenia úloh rozvrhovania je iteratívne vylepšovanie najlepšieho nájdeného riešenia tým, že sa snažíme nájsť lepšie (t.j. kratší rozvrh). Obvyklý postup implementovaný v jazykoch CLP postupne znižuje hornú hranicu buď po jednotkách (ak hľadáme optimálny rozvrh), alebo po určitých krokoch (ak hľadáme rozvrh s určitou toleranciou vzdialenosti od optimálneho riešenia). Podrobnejšie bol tento postup popísaný v časti 2.4.

Na tomto mieste uvediem dve nové metódy, ktoré boli tiež implementované v jazyku *ECL^{i}PS^e* a otestované na tej istej množine úloh. Ide o logaritmické alternatívy postupov používaných v CLP (MINMAX a MINIMIZE - viď. popis v časti 2.4). Podstata oboch metód, t.j. **LOGARITMICKÝ MINMAX** i **LOGARITMICKÝ MINIMIZE** (podrobnejší popis programov možno nájsť v [Schmotzer 97]) spočíva v delení intervalu ohraničeného aktuálnou dolnou a hornou hranicou nákladov na polovicu a v hľadaní rozvrhu s takto získanou novou hranicou. Ak rozvrh takejto dĺžky neexistuje, potom sa táto hodnota stáva novou dolnou hranicou. Ak sa takýto (prípadne kratší) rozvrh nájde, potom sa dĺžka novo nájdeného rozvrhu stáva novou hornou hranicou a postup sa opakuje až kým sa interval nezúži na jednotku. Rozdiel medzi oboma algoritmami je v tom, že LOGARITMICKÝ MINMAX po nájdení nového riešenia začína prehľadávať priestor prehľadávania odznova s novým, prísnejším

obmedzením pre hodnotu nákladov. Na druhej strane LOGARITMICKÝ MINIMIZE po nájdení nového riešenia pokračuje presne v tom mieste priestoru prehľadávania, kde skončil, len s pridaním nového prísnejšieho obmedzenia pre hodnotu nákladov.

Pre účinnejšie riešenie úloh s ťažkým dôkazom optimálnosti je naviac algoritmus LOGARITMICKÝ MINMAX vylepšený o dva typy špeciálnych situácií, kedy sa jeho stratégia mení stratégiu MINMAX.

Jednou z nich je situácia, keď sa trikrát po sebe nezmení horná hranica, potom v nasledujúcom kroku neposúva dolnú hranicu, ale len hornú zníži o 1 (pre jeden nasledujúci krok teda použije stratégiu MINMAX).Druhá situácia nastáva, keď sa veľkosť intervalu daná dolnou a hornou hranicou zníži pod hodnotu 6. Potom sa stratégia trvalo prepne na MINMAX (na zvyšných nanajvýš 5 krokov).

Takouto úpravou metódy boli dosiahnuté najlepšie výsledky a tieto sú aj uvedené v tabuľke 11.

Pre úplnosť je uvedený v prílohe P.4 popis programu prvej zo spomínaných dvoch metód hľadania optimálneho riešenia (t.j. LOGARITMICKÝ MINMAX) v CLP jazyku *ECL^{i}PS^e*. Navrhnutý predikát log_min_max(Ciel, Premenna, Minimum, Maximum, PO) nájde také riešenie cieľa Ciel, ktorý minimalizuje hodnotu doménovej premennej Premenna v rozsahu intervalu <Minimum, Maximum>. Riešenie mimo tohoto intervalu predikát nenájde. PO je povolená odchýlka od optima v percentách.

Nasledujúca tabuľka 11 udáva výsledky testov porovnávajúcich klasický postup hľadania optimálneho riešenia v CLP metódou MINMAX s novo navrhnutými metódami LOGARITMICKÝ MINMAX a LOGARITMICKÝ MINIMIZE. Opäť pre každú metódu sú v tabuľke uvedené dva údaje ku každej z testovaných úloh, a síce čas výpočtu v sekundách a počet návratov reprezentujúci veľkosť prehľadaného priestoru. Všetky algoritmy boli štartované s rovnakými počiatočnými hranicami, a síce dolná hranica podľa postupu uvedeného v predchádzajúcej časti 7.1 a hornou hranicou vypočítanou podľa novo navrhnutej heuristiky EST+ (všetky hodnoty možno nájsť v tabuľke 10).

úloha	min_max/5		log_min_max/5		log_minimize/5	
	čas1 (s)	PN1	čas2 (s)	PN2	čas3 (s)	PN3
8_8_0	30.75	**69**	11.22	80	**10.25**	746
8_8_1	40.63	384	**3.77**	**46**	5.89	319
8_8_2	98.5	1331	46.99	**720**	**32.65**	1238
8_8_3	70.34	2019	**7.65**	**67**	9.99	744
8_8_4	68.7	3654	6.42	**56**	**5.47**	676
8_8_5	114.01	6534	**17.80**	**296**	21.51	873
8_8_6	31.13	50	5.69	**4**	**3.41**	452
8_8_7	42.06	356	11.51	**113**	**11.01**	755
8_8_8	29.75	73	**9.17**	**93**	12.50	604
8_8_9	109.03	1813	**26.10**	**475**	30.94	1073
10_10_0	808.42	13517	151.65	**1570**	**108.28**	2130
10_10_1	547.48	10287	69.46	**384**	**32.65**	883
10_10_2	1064.87	20233	415.17	**6794**	**351.28**	8037
10_10_3	1010.60	19271	331.26	**3924**	**289.85**	4851
10_10_4	2307.45	1117019	157.47	**2828**	**156.26**	4506
10_10_5	382.99	8450	84.55	**877**	**56.30**	3286
10_10_6	126.64	1425	22.87	**148**	**18.68**	1043
10_10_7	8791.53	135582	**7912.14**	**116943**	8932.44	167274
10_10_8	590.05	27063	114.13	**2007**	**89.90**	2577
10_10_9	6256.24	111350	2949.03	42534	**1228.24**	**29234**
most	**28821.2**	**10980541**	29103.20	1427724	32730.90	1427795

Tabuľka 11 Porovnanie optimalizácie pre klasický postup MINMAX a nové metódy LOGARITMICKÝ MINMAX a MINIMIZE.

Dosiahnuté výsledky jednoznačne ukazujú účinnosť novo navrhnutých postupov pre hľadanie optimálneho riešenia. Vo väčšine prípadov je zrýchlenie niekoľkonásobné. Presnejšie metóda LOGARITMICKÝ MINMAX bola u úloh job-shop rozmeru 8 x 8 rýchlejšia 2.1 až 10.78 krát, v priemere 5.84 krát. Pomer počtu návratov sa pohyboval od 0.78 do 65.25 (v dvoch prípadoch bol prehľadaný priestor menší u metódy MINMAX). Pre úlohy rozmeru 10 x 10 sa zrýchlenie oproti tradičnej metóde MINMAX pohybovalo od 1.11 do 14.65, v priemere 5.19. Pomer počtu návratov sa pohyboval od 1.16 do 395.0 (ak neuvažujeme jednu extrémnu hodnotu, v priemere 9.64).

LOGARITMICKÝ MINIMIZE bol u úloh job-shop rozmeru 8 x 8 rýchlejší 2.38 až 12.56 krát, v priemere 5.67 krát. Pomer počtu návratov sa pohyboval od 0.09 do 5.4 (v štyroch prípadoch bol prehľadaný priestor menší u metódy MINMAX). Pre úlohy rozmeru 10 x 10 sa už zrýchlenie oproti tradičnej metóde MINMAX pohybovalo od 0.98 do 16.77, v priemere 7.17. Pomer počtu návratov sa pohyboval od 0.81 do 247.9

(v jednom prípade bol prehľadaný priestor menší u metódy MINMAX. Ak neuvažujeme dve extrémne hodnoty, priemer bol 4.83).

U reálnej úlohy rozvrhu prác pre stavbu päťsegmentového mostu bola situácia opačná, nové metódy boli totiž o niečo pomalšie a prehľadaný priestor väčší. Ide o úlohu, ktorá je typická tým, že má veľmi náročný dôkaz optimálnosti. U takýchto úloh (v blízkosti optima) je naozaj lepší postup zhora, teda tak, ako to robí klasický MINMAX.

Čas strávený dôkazom optimálnosti nemožno bežnými postupmi skrátiť. Jednou z možností by bol paralelný výpočet. Úloha rozvrhu prác na výstavbe päťsegmentového mostu je úplne extrémnym príkladom tohoto typu, nakoľko nájdenie optimálneho riešenia trvalo metódou LOGARITMICKÝ MINIMIZE len niekoľko sekúnd, ale dôkaz optimálnosti vyše osem hodín. Tu je potrebné si však uvedomiť, že z praktického hľadiska obvykle úplne postačuje informácia, že nájdené riešenie je niekoľko desatín percenta (resp. niekoľko málo percent) vzdialené od optima, a teda v praxi úplne postačujúce. Informáciu o maximálnej možnej vzdialenosti posledne nájdeného riešenia od optima dávajú obe navrhnuté metódy.

Naviac sa použitím symbolického ohraničenia task_intervals/3 implementovaného na základe metódy intervalov úloh (podrobnejšie viď. časť 5.4) tento čas veľmi výrazne zredukoval na výsledných zhruba 25 minút metódou MINMAX a len o 3,5 minúty viac pri použití LOGARITMICKÉHO MINMAX, resp. LOGARITMICKÉHO MINIMIZE. (podrobnejšie viď. časť 6.2.3).

Kapitola 8

METODOLÓGIA RIEŠENIA ROZVRHOVACÍCH ÚLOH V CLP

V tejto kapitole zhrniem poznatky týkajúce sa najdôležitejších aspektov riešenia rozvrhovacích úloh v prostredí CLP nadobudnuté v rámci predchádzajúcich kapitol. Výsledkom je ucelená metodológia riešenia rozvrhovacích úloh v prostredí logického programovania ohraničení (CLP).

V prvej časti sa stručne dotknem problematiky reprezentácie rozvrhovacej úlohy v CLP, t.j. definovaniu premenných a ohraničení medzi týmito premennými. Najdôležitejším v tomto smere je výber spôsobu reprezentácie disjunktných ohraničení, ktorej som sa podrobne venoval v piatej kapitole.

Úlohy rozvrhovania sú optimalizačné a práve problematike optimalizácie je venovaná druhá časť tejto kapitoly. Ide jednak o voľbu optimalizačného kritéria, heuristicke nájdenie suboptimálneho riešenia a nakoniec voľbu stratégie hľadania optimálneho riešenia.

Nakoniec tretia, záverečná časť kapitoly podrobne popisuje reálnu aplikáciu rozvrhovania zavážania brám do narážacích pecí v závode VSŽ Oceľ s.r.o., ktorá bola riešená v CLP použitím tejto metodológie.

8.1 Reprezentácia úloh rozvrhovania

Ako už bolo spomínané v úvodnej kapitole pre riešenie úloh v prostredí CLP je potrebné ich formulovať ako úlohy spĺňania ohraničení, t.j. ako

- množinu premenných s počiatočnými doménami a
- množinu ohraničení medzi týmito premennými, ktoré musí spĺňať každé riešenie. Naviac je možné definovať aj
- optimalizačné kritérium, podľa ktorého sa posudzuje kvalita riešení.

V tejto časti preto zhrniem základné poznatky týkajúce sa takéhoto spôsobu reprezentácie úloh rozvrhovania.

8.1.1 Voľba premenných

Pri úlohách časového rozvrhovania je voľba premenných jednoznačná. Pre každú úlohu (operáciu) ktorej začiatok je potrebné naplánovať v čase, potrebujeme aspoň jednu premennú, ktorá bude reprezentovať čas začiatku tejto úlohy (operácie).

V prípade premenlivej dĺžky trvania niektorých úloh (operácií) potrebujeme ešte jednu premennú pre každú takúto úlohu (operáciu), ktorá bude označovať čas jej trvania, prípadne čas jej ukončenia.

Ďalšie dodatočné informácie nutné k reprezentácii danej úlohy (operácie) sa obvykle spolu s vyššie uvedenými premennými dávajú do spoločnej štruktúry. Ide najmä o informácie typu: identifikačné označenie danej úlohy, na ktorom stroji má byť vykonaná (ktorý zdroj je potrebný pre jej uskutočnenie), trvanie úlohy a pod.

Vo všeobecnosti môže byť úloha komplikovaná ďalšími skutočnosťami, ako napr. viaceré alternatívne zdroje pre vykonanie jednej úlohy (operácie). V takom prípade potrebujeme pre túto úlohu ešte jednu premennú, ktorá bude reprezentovať použitý zdroj. Jej počiatočná doména bude obsahovať množinu všetkých zdrojov, ktoré pre túto úlohu (operáciu) prichádzajú do úvahy.

Základ teda tvoria obvykle premenné reprezentujúce začiatky jednotlivých úloh (operácií), ktoré je potrebné rozvrhnúť v čase. Ich počiatočné domény sú preto závislé od požadovanej presnosti rozvrhu (hodiny, smeny, dni a pod.). Vzhľadom na túto presnosť je potom možné diskretizovať interval rozvrhu a stanoviť tak konečné domény pre jednotlivé premenné napr. ako interval $\langle 0, Limit \rangle$. *Limit* možno najjednoduchšie stanoviť ako súčet dĺžok trvaní všetkých úloh ktoré je nutné rozvrhnúť, prípadne zväčšený o predpísané časové odstupy medzi úlohami.

V niektorých prípadoch, ak je požadovaná presnosť vysoká a reprezentovaný interval dlhý, je možné použiť aj reprezentáciu pomocou reálnych premenných. Mnohé CLP jazyky obsahujú totiž aj algoritmy lineárneho programovania a sú teda schopné pomerne efektívne narábať aj s reálnymi premennými a numerickými ohraničeniami medzi nimi.[19]

Obe hranice je však možno ďalej priblížiť a tým zúžiť počiatočné domény premenných, a teda skúmanú časť priestoru prehľadávania rôznymi heuristikami. Viac o nich bolo uvedené v časti 7.1.

8.1.2 Reprezentácia ohraničení

Ohraničenia vyskytujúce sa pri úlohách rozvrhovania možno rozdeliť v zásade do dvoch skupín:

- precedenčné ohraničenia,
- disjunktné ohraničenia.

Precedenčné ohraničenia vyjadrujú jednoznačné časové relácie medzi dvojicami premenných (začiatky úloh).

Tieto je možné rozčleniť na niekoľko základných typov, ako to definoval už [Allen 83]. Prehľad jednotlivých typov časových relácií a spôsobu ich reprezentácie v CLP (na príklade jazyka *ECL^{i}PS^e*) je uvedený v nasledujúcej časti 8.1.3.

[19] V jazyku ECLiPSe sa napr. táto skutočnosť prejaví v použití numerických ohraničení, ktoré začínajú znakom *$* namiesto znaku *#* pre konečné domény.

Principiálne odlišný prístup je typický pre disjunktné ohraničenia, ktorým bola podrobne venovaná kapitola 5. Stručné zhrnutie záverov a doporučení vyplývajúcich z tejto práce možno nájsť v časti 8.1.4.

8.1.3 Precedenčné ohraničenia

Táto skupina najčastejšie sa vyskytujúcich ohraničení v rozvrhovacích úlohách má charakter časových relácií medzi dvoma úlohami (operáciami) *A* a *B*. Ide o nasledovné možnosti vzájomných časových relácií.

Ak je definovaná premenná pre čas ukončenia danej úlohy (operácie) X, potom sa priamo použije v uvedených ohraničeniach ako Koniec(X). Ak nie, potom sa nahrádza výrazom Start(X) + Trvanie(X).

- A **po** B - počiatočný čas úlohy A musí byť väčší nanajvýš rovný času ukončenia úlohy B:

  Start(A) #>= Koniec(B)

- A **pred** B - čas ukončenia úlohy A musí byť menší nanajvýš rovný počiatočnému času úlohy B:

  Koniec(A) #=< Start(B)

- A **rovná sa** B - čas začiatku aj čas ukončenia úlohy A sa musí rovnať počiatočnému času resp. času ukončenia úlohy B:

  Start(A) #= Start(B),
  Koniec(A) #= Koniec(B)

- A **je sledovaná** B - čas ukončenia úlohy A sa musí rovnať počiatočnému času úlohy B:

  Koniec(A) #= Start(B)

- A **nasleduje po** B - počiatočný čas úlohy A musí byť rovný času ukončenia úlohy B:

  Start(A) #= Koniec(B)

- A **je prekrývaná** B - počiatočný čas úlohy B musí byť menší ako počiatočný čas úlohy A a čas ukončenia úlohy B musí byť menší ako čas ukončenia úlohy B:

  Start(B) #< Start(A),
  Koniec(B) #< Koniec(A)

- A **prekrýva** B - počiatočný čas úlohy B musí byť väčší ako počiatočný čas úlohy A a čas ukončenia úlohy B musí byť väčší ako čas ukončenia úlohy A:

  Start(B) #> Start(A),
  Koniec(B) #> Koniec(A)

- A **obsahuje** B - počiatočný čas úlohy A musí byť menší ako počiatočný čas úlohy B a čas ukončenia úlohy A musí byť väčší ako čas ukončenia úlohy B:

  Start(A) #< Start(B),
  Koniec(A) #> Koniec(B)

- A **je obsiahnutá v** B - počiatočný čas úlohy B musí byť menší ako počiatočný čas úlohy A a čas ukončenia úlohy B musí byť väčší ako čas ukončenia úlohy A:

  Start(B) #< Start(A),
  Koniec(B) #> Koniec(A)

- A **uvádza** B - počiatočný čas úlohy A je zhodný s počiatočným časom úlohy B a čas ukončenia úlohy A je menší ako čas ukončenia úlohy B:

  Start(A) #= Start(B),
  Koniec(A) #< Koniec(B)

- A **je uvádzaná** B - počiatočný čas úlohy A je zhodný s počiatočným časom úlohy B a čas ukončenia úlohy A je väčší ako čas ukončenia úlohy B:

  Start(A) #= Start(B),
  Koniec(A) #> Koniec(B)

- A **ukončuje** B - čas ukončenia úlohy A je rovný času ukončenia úlohy B a počiatočný čas úlohy A je väčší ako počiatočný čas úlohy B:

  Koniec(A) #= Koniec(B),
  Start(A) #> Start(B)

- A **je ukončované** B - čas ukončenia úlohy A je rovný času ukončenia úlohy B a počiatočný čas úlohy A je menší ako počiatočný čas úlohy B:

  Koniec(A) #= Koniec(B),
  Start(A) #< Start(B)

Uvedené časové relácie medzi dvoma úlohami (operáciami) A, B môžu byť ešte doplnené o presné časové odstupy, ako tomu bolo v prípade dodatočných ohraničení pre úlohu rozvrhu prác na výstavbe päťsegmentového mosta (viď. časť 6.2.1). Napríklad ak úloha B môže začať najskôr 3 časové jednotky po skončení úlohy A, možno to zapísať ako numerické ohraničenie:

Start(B) #>= Koniec(A) + 3.

8.1.4 Disjunktné ohraničenia

Iný charakter majú ohraničenia vyjadrujúce zdieľanie zdrojov viacerými úlohami. Tieto sa používajú dvojakým spôsobom:

1. Ako nezávislé alternatívy (body výberu), ktoré sa skúmajú v priebehu prehľadávania (podrobnejšie viď. časť 5.1).

2. Ako ohraničenia zabezpečujúce väčšiu mieru propagácie. V tomto smere existuje niekoľko alternatív s rôznou mierou získanej informácie (podrobnejšie v častiach 5.2 až 5.4).

Prvý spôsob použitia je záväzný a slúži pri hľadaní riešenia vlastne tým, že vyberá postupne čiastkové usporiadania jednotlivých dvojíc úloh zdieľajúcich ten istý zdroj.

Druhý spôsob je ľubovoľný a priamo závislý od charakteru a veľkosti konkrétneho rozvrhovacieho problému, ktorý je potrebné riešiť. Na základe výsledkov mnohých testov, ktoré som vykonal s implementovanými spôsobmi reprezentácie disjunktných ohraničení (popis v častiach 5.2 až 5.4) na rozvrhovacích úlohách rôznej veľkosti a charakteru možno doporučiť nasledovné.

Pre úlohy malého rozsahu (napr. úlohy job-shop do rozmeru 5 x 5) je úplne postačujúce použiť len prvý spôsob reprezentácie disjunkcií ako nezávislých alternatív. Čas potrebný na propagáciu dômyselnejších spôsobov reprezentácie je totiž dlhší než prehľadanie celého priestoru.

S narastajúcou zložitosťou úloh sa zväčšuje aj význam druhej skupiny reprezentácií disjunktných ohraničení. Pričom pre stredné úlohy ešte úplne postačuje menej náročná propagácia ohraničení pomocou hranovej B-konzistencie (disjunction_choose/5), prípadne úplnej hranovej konzistencie (disjunction/5). Tá sa však ukázala menej účinná pre úlohy časového rozvrhovania, kde rozhodujúci význam má redukcia domén na ich hraniciach, nie vo vnútri.

Pre veľké rozvrhovacie úlohy je prakticky nevyhnutné použiť najsilnejšie propagujúce ohraničenie disjunctive/3. Naviac tu hrá už veľmi významnú rolu aj počiatočné zúženie intervalu trvania rozvrhu pomocou rôznych heuristík. Podrobnejšie o tejto problematike pojednáva časť 7.1.

Ohraničenie task_intervals/3 sa ukázalo ako účinné len pri veľmi zložitom dôkaze optimálnosti úlohy hľadania rozvrhu výstavby päťsegmentového mosta (časť 6.2). Vo všeobecnosti možno teda najviac doporučiť použitie komplexného, ale veľmi efektívneho ohraničenia disjunctive/3.

8.2 Optimalizácia

8.2.1 Optimalizačné kritérium

Vo väčšine prípadov je cieľom nájsť najkratší možný rozvrh, takže optimalizačným kritériom sa stáva čas ukončenia poslednej úlohy (operácie).

Najlepšie je pridať jednu premennú (napr. K), ktorú si možno predstaviť ako fiktívnu úlohu nulovej dĺžky, ktorá musí byť vykonaná ako posledná. Takže pre každú úlohu (operáciu) X, ktorú je potrebné rozvrhnúť, musíme pridať aj ohraničenie

```
Koniec(X) #< K
```

Potom ako optimalizačnú funkciu je možné použiť práve hodnotu K. Nakoľko ide tiež o doménovú premennú s rovnakou počiatočnou doménou ako majú všetky ostatné premenné reprezentujúce počiatočné časy úloh, snažíme sa minimalizovať jej minimálnu možnú hodnotu.

Vzhľadom na vyššie uvedené ohraničenia časov ukončenia všetkých úloh, ktoré sú viazané na hodnotu premennej K, je možné jej znižovaním priamo ovplyvňovať všetky zainteresované premenné a tým dĺžku výsledného rozvrhu.

To je aj spôsob akým sa hľadá optimálne riešenie. Ak bol nájdený rozvrh s dĺžkou L1 (K = L1), potom sa v nasledujúcom cykle hľadá riešenie s dĺžkou L2 < L1. Spôsob, akým je možné rýchle heuristicky nájsť pomerne nízku hodnotu L1 bol podrobnejšie popísaný v časti 7.1 a to najpodstatnejšie je zhrnuté v nasledujúcej časti 8.2.2.

Algoritmy, podľa ktorých je možné generovať nasledujúce L2 (a vo všeobecnosti Li+1 pre už nájdené Li) boli popísané v časti 7.2 a to hlavné je zhrnuté v 8.2.3.

8.2.2 Heuristiky pre nájdenie suboptimálneho riešenia

Význam tohoto kroku narastá so zložitosťou riešenej rozvrhovacej aplikácie. Čím zložitejšia úloha, tým väčší priestor prehľadávania a tým časovo náročnejší každý krok vedúci k skráteniu výsledného rozvrhu (znižovanie hodnoty optimalizačnej funkcie).

Z toho nutne vyplýva, že čím bližšie bude prvé vygenerované riešenie k skutočnému optimálnemu riešeniu, tým väčšiu šancu máme, že sa naozaj dočkáme jeho nájdenia, resp. že nájdeme nejaké riešenie, ktoré bude dostatočne blízke optimu.

Podľa výsledkov experimentov popísaných podrobnejšie v predchádzajúcej kapitole, možno doporučiť pre tento krok novo navrhnutú heuristiku EST+ (podrobnejšie v časti 7.1), ktorá vykazuje najlepšie priemerné správanie, t.j. v najväčšom počte testovaných úloh bol dosiahnutý najlepší odhad a zároveň v úlohách, kde táto heuristika nebola najlepšia, jej vzdialenosť od najlepšieho nájdeného odhadu bola minimálna.

Okrem odhadu hornej hranice, čo predstavuje nájdenie čo možno najlepšieho suboptimálneho riešenia, môže prispieť k zúženiu priestoru prehľadávania aj dobrý odhad dolnej hranice. Tu ide o odhad doby trvania, pod ktorú sa určite nedá už rozvrh stihnúť. Navrhnutý deterministický postup pre stanovenie dobrej dolnej hranice bol popísaný v časti 7.1 a jeho okomentovaný program možno nájsť v prílohe P.3.

8.2.3 Hľadanie optimálneho riešenia

Poslednou etapou riešenia úloh rozvrhovania je iteratívne vylepšovanie najlepšieho nájdeného riešenia tým, že sa snažíme nájsť lepšie (t.j. kratší rozvrh). Obvyklý postup implementovaný v jazykoch CLP postupne znižuje hornú hranicu buď po jednotkách (ak hľadáme optimálny rozvrh), alebo po určitých krokoch (ak hľadáme rozvrh s určitou toleranciou vzdialenosti od optimálneho riešenia). Podrobnejšie bol tento postup popísaný v časti 2.4.

Novo navrhnuté postupy (LOGARITMICKÝ MINMAX A LOGARITMICKÝ MINIMIZE), popísané podrobnejšie v časti 7.2 vedú v drvivej väčšine prípadov k omnoho rýchlejšiemu nájdeniu optimálneho riešenia. Naviac, aj u úloh s náročným dôkazom optimálnosti sú veľmi účinné v tom zmysle, že v každom kroku podávajú aj informáciu o maximálne možnej vzdialenosti najlepšieho nájdeného riešenia od optima a u oboch metód je počet medzikrokov vedúcich od počiatočného heuristicky nájdeného riešenia k optimu veľmi malý vzhľadom na delenie intervalu nákladov na polovicu

8.3 Rozvrhovanie brám do narážacích pecí

V tejto časti bude podrobnejšie popísaná reálna aplikácia rozvrhovania brám zavážaných do narážacích pecí za účelom ohrevu na požadovanú teplotu pred valcovaním na teplej širokopásovej valcovni. Pri riešení tejto reálnej rozvrhovacej úlohy som sa snažil aplikovať navrhnutú metodológiu.

Popis súvisiacej technológie a samotnej úlohy zodpovedá skutočnej situácii v závode VSŽ Oceľ s.r.o. a je prebraný z [Malindžák 96]. Za dôkladný popis úlohy, ako aj navrhnutého riadiaceho systému vďačím spolupráci s Katedrou riadenia výrobných procesov (KRVP), Fakulty Baníctva, Ekológie, Riadenia a Geotechológií (BERG) Technickej univerzity v Košiciach, najmä Doc. Ing. Dušanovi Malindžákovi, CSc., s ktorým som spolupracoval pri porovnávaní dvoch rozdielnych prístupov k riešenej problematike.

Po opise súvisiacej technológie a základných vlastností existujúceho riešenia je podrobne popísané riešenie pomocou CLP, ktoré bolo realizované v jazyku *ECL^{i}PS^e* a veľmi úspešne prezentované na 13. Európskom stretnutí pre kybernetiku a systémový výskum vo Viedni (apríl 1996), kde bol náš príspevok ocenený ako najlepší v rámci sympózia Teória a aplikácie umelej inteligencie [CsoPar 96].

Nakoniec budú popísané a zhodnotené dosiahnuté výsledky, ktoré porovnávam s riešením navrhnutým spomínaným pracoviskom na fakulte BERG (preto je v práci stručne popísané aj ich riešenie).

8.3.1 Popis problému a súvisiacej technológie

Narážacie pece (NP) zabezpečujú ohrev brám na požadovanú valcovaciu teplotu pre teplú širokopásovú valcovaciu trať (TŠP 1700) podľa valcovacieho programu - grafikonu TŠP. Súčasne sú pružnou väzbou medzi zariadeniami pre plynulé odlievanie brám (ZPO I., ZPO II.). Súčasný spôsob zavážania (t.j. vtláčania) brám do NP1 až NP4 (viď. obrázok 7) presne v tom poradí ako budú valcované na TŠP 1700 po jednej bráme, cyklicky do všetkých štyroch NP, nedáva možnosť riešiť rôzne netypické situácie, ktoré vznikajú na úseku NP - TŠP 1700 a tieto situácie vyvolávajú z hľadiska riadenia materiálového toku poruchy, najčastejšie sa prejavujúce v nedodržaní grafikonu TŠP. Tento systém tiež neumožňuje znížiť náklady na ohrev brám v NP.

V nasledujúcom je najprv podrobnejšie popísaná súvisiaca technológia a riadiaci systém navrhnutý na Katedre riadenia výrobných procesov [Malindžák 96], potom nasleduje popis riešenia časti tejto problematiky prostriedkami CLP.

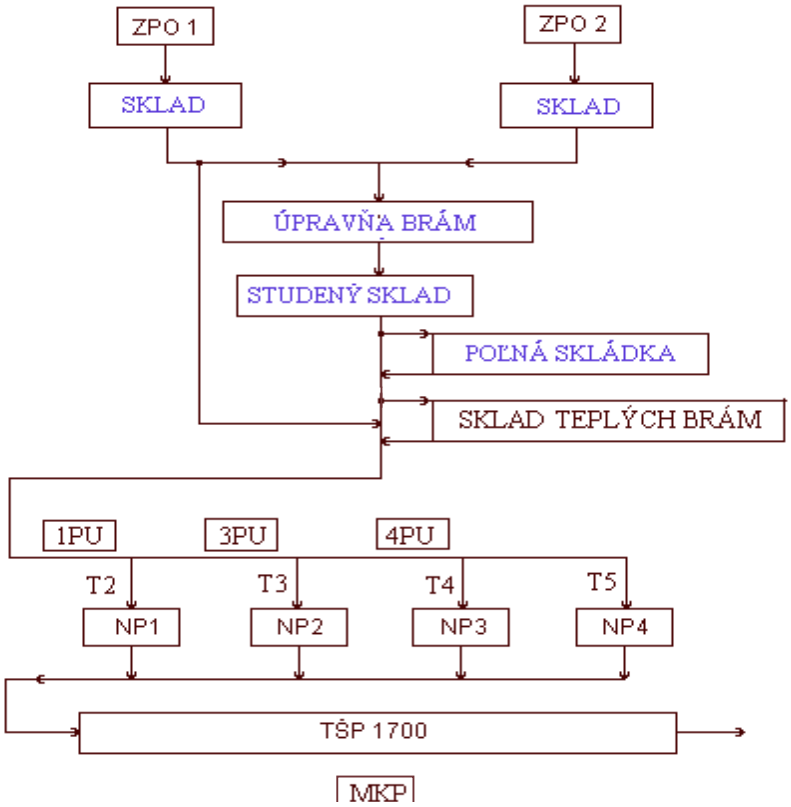
Na dvoch zariadeniach pre plynulé odlievanie (ZPO I., ZPO II.) sa odlievajú brámy požadovaných rozmerov. Odliate brámy postupujú ďalej buď do úpravne brám, alebo tzv. "priamym sledom" na ohrev do jednej zo štyroch narážacích pecí (NP1, NP2, NP3, NP4), ak sa hodia do grafikonu zavážania NP (GZNP). Ak nie, tak idú buď do skladu teplých brám (ak budú vhodné do GZNP o krátky čas), alebo do studeného skladu (obrázok 7).

V prípade požiadavky na daný typ brám sa tieto vyberú z príslušného skladu a spolu s brámami z priameho sledu (teplé brámy bez skladovania) sa zavezú na ohrev v narážacích peciach. Po ohriatí sú brámy vytláčané z narážacích pecí práve zavážanými brámami, a valníkmi sú dopravené na valcovanie na TŠP 1700.

Pohyb brám na úseku NP je riadený z operátorských kabín 1PU, 3PU, 4PU, MKP. Brámy sú privážané na valník pred NP v poradí podľa grafikonu TŠP. Obsluha kabíny 1PU ich dopravuje pred príslušnú narážaciu pec. Samotné osadenie do NP sa realizuje obsluhami kabín 3PU a 4PU pomocou "tlačiek" T2,T3,T4,T5 (viď. obrázok 7).

Signál na tlačenie kabínam 3PU a 4PU dáva obsluha kabíny MKP, ktorá podľa valcovacieho programu rozhoduje, z ktorej narážacej pece (a súčasne aj do ktorej NP), bude bráma vytlačená (zatlačená).

Celý systém NP má z hľadiska materiálového toku pomerne veľkú zotrvačnosť, nakoľko spolu v NP je okolo 100-120 kusov brám. (Brámy majú šírku od 800-1540 mm, dĺžku 6 až 8 m, hrúbku 180-200 mm; NP má dĺžku 32,5 m). Pri kadencii (interval medzi dvoma po sebe vytláčanými brámami z NP) od 60 do 120 sekúnd je možnosť operatívnej zmeny s časovým oneskorením 200-240 minút.



Obrázok 7 Schéma materiálového toku v okolí NP.

Nevýhody súčasného spôsobu riadenia a zavážania NP sú nasledovné:

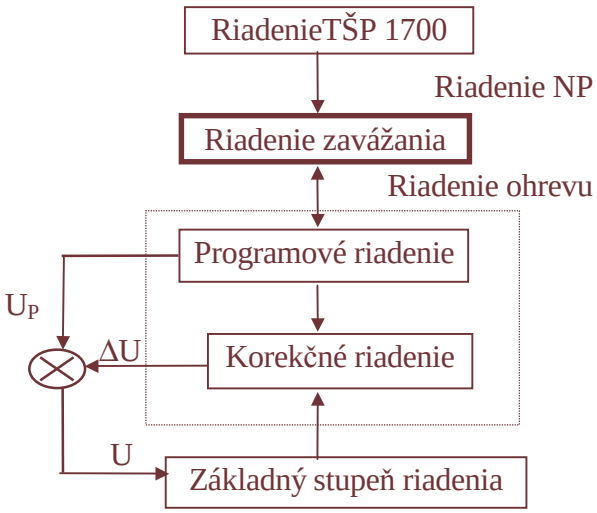
- V NP sú často za sebou pri ohreve brámy rôznych kvalít, s potrebou ohrevu na rôznu valcovaciu teplotu, pričom sa musia ohrievať na tú teplotu, ktorá je vyššia, čím sa niektoré brámy zbytočne prehrievajú a dochádza k stratám energie a prepalu kovu.
- Brámy majú tiež rôznu vstupnú teplotu (napr. 10^{o}C zo skladu studených brám resp. napr. 500^{o}C z priameho sledu). Tlak trhu núti potvrdzovať objednávky na mesiac, týždeň, čo spôsobuje zmenšenie priemerného počtu kusov rovnakých brám (položky) prichádzajúcich k narážacím peciam. Preto často v jednej zóne (zóny v NP viď. obrázok 9) narážacej pece môžu byť 2 až 3 druhy brám.
- Pevná väzba vstup - výstup z NP, obmedzuje možnosť prispôsobovaniu sa situáciám, keď sa omeškajú alebo prídu skôr brámy priameho sledu, čo spôsobí zmenu grafikonu TŠP a tým aj nedodržanie optimálnych pravidiel valcovania. Súčasne vznikajú tepelné straty vychladnutím brám, pretože sa nezavážali v momente príchodu k NP.
- Pri jednom type brám je rôzna kvalita ohrevu z dôvodu, že sa ohrievajú vo viacerých NP a tým aj rôzna kvalita valcovaného plechu pre tú istú objednávku.

- Grafikon TŠP nezohľadňuje polohu brám na skladoch ani špecifiká ohrevu brám v NP.
- Veľká zotrvačnosť dáva malú šancu pre operatívne zmeny, najmä v prípade odstávok TŠP 1700 a rôznych poruchách na úseku NP.

8.3.2 Koncepcia systému riadenia NP

Nedostatky popísané v predchádzajúcej časti z veľkej časti rieši v ďalšom stručne popísaný model situačného riadenia NP vyvinutý na spomínanej Katedre riadenia výrobných procesov [Malindžák 96]. Model situačného riadenia je súčasťou trojúrovňového systému riadenia (viď. obrázok 8):

- Riadenie skupiny narážacích pecí (riadenie zavážania).
- Optimalizácia ohrevu.
- Korekčné riadenie.



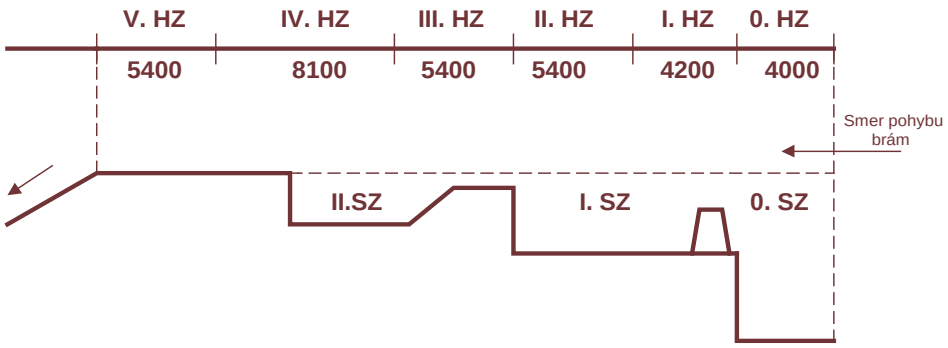
Obrázok 8 Štruktúra systému riadenia NP.

Riadenie skupiny narážacích pecí sa realizuje cez situačné vytváranie grafikonu zavážania a sledovanie materiálového toku. Pre daný grafikon zavážania brám, z ktorého vyplynú časy pobytu brám v jednotlivých zónach NP vypočíta optimalizačnú úroveň riadenia, optimálne teplotné pole a príkony v každej zóne narážacej pece tak, aby sa zabezpečil ohrev na požadovanú valcovaciu teplotu. Tým sa zabezpečí ohrev s minimálnymi nákladmi na palivo a minimálnou stratou kovu opalom.

Korekčné riadenie porovnáva skutočný teplotný režim s vopred vypočítaným optimálnym režimom (v podobe optimálnej trajektórie ohrevu) a v prípade odchýlky uskutočňuje v reálnom čase zásahy do riadenia ohrevu s cieľom maximálne priblížiť ohrev vypočítanej optimálnej trajektórii.

Ciele systému riadenia zavážania NP. Hlavným cieľom je zabezpečiť požadované množstvo brám v požadovanej sekvencii, kadencii a teplote pre TŠP 1700 :

- dávkovou organizáciou zavážania sa pripravia výhodné podmienky pre riadenie ohrevu brám v NP, čím sa zabezpečí úspora energie na ohrev,
- zväčší sa podiel zavážania teplých brám, čím sa ušetrí energia na ich ohrev,
- dávková organizácia umožní vytvoriť "situačný systém riadenia zavážania NP", čím sa vyriešia typické situácie, ako je predčasný príchod teplej vsádzky, oneskorenie teplých brám a pod. Tieto situácie takto nebudú pôsobiť ako poruchy, ale stanú sa súčasťou situačného riadenia,
- zlepší sa dodržanie grafikonov TŠP 1700 a tým aj dodržanie optimálnych podmienok valcovania.



Obrázok 9 Schéma narážacej pece s vyznačením jednotlivých zón (dĺžky jednotlivých zón NP sú udané v mm).

Dávkové zavážanie NP

Dávka je skupina brám rovnakej kvality (rozmery, typ materiálu, teplota) alebo dvoch príbuzných kvalít, s rovnakými požiadavkami na ohrev, ktoré sa v intervale kadencie zavezú do jednej NP. Dĺžka dávky, t.j. súčet šírok brám v jednej dávke je odvodená od rozmerov NP.

Dĺžka dávky bola určená z rozmerov NP a z dĺžky zón NP (viď obrázok 9). NP má 6 zón, celkovú dĺžku 32,5 m a väčšina zón má dĺžku okolo 5 m. Ohrev z plynných horákov je regulovateľný pre jednotlivé zóny. Nultá horná zóna a IV. a V. zóna nie sú vykurované. Preto bolo rozhodnuté formovať dávku brám dynamicky v závislosti od stavu v NP a šírky zavážaných brám v intervale <4,5m, 6,5m>. To je tzv. *štandardná dávka*. Zavážanie prebieha formou štandardnej dávky, kým nedôjde k žiadnym narušeniam.

Režim fungovania NP sa tým zmení z priebežného na dávkový. Tým sa činnosť NP rozdelí do dvoch období :

a) *Zavážanie* - t.j. keď sa v intervaloch kadencie zaváža niekoľko kusov brám jednej dávky do príslušnej NP a súčasne sa brámy ohrievajú (v tom čase sa mení poloha všetkých brám v NP).

b) *Ohrev* - brámy v peci stoja, nezaváža sa, prebieha ohrev, zatiaľ čo sa zavážajú dávky do ostatných narážacích pecí.

Dávkový režim zavážania NP umožňuje vedľa seba do jednej dávky umiestňovať brámy s približne rovnakými požiadavkami na ohrev.

Štruktúra a princípy situačného riadenia NP

Na základe grafikonu TŠP, údajov od operátora systému (kabína MKP), stavu pecí v aktuálnej generácii N označovanej OP(N) t.j. podľa situácií S(N) na NP sa vytvorí grafikon zavážania OP(N+1) a OP(N+2). OP(N+1) sa časovo prepočíta a informácie sa presunú do systému riadenia ohrevu, ktorý vypočíta optimálnu stratégiu ohrevov. Na základe signálov o vypadnutí brámy z NP a informácií o tlačení do NP sa postupne aktualizuje OP(N). V momente, keď sa OP(N+1) stane OP(N), vytvorí sa nová generácia OP(N+1) a OP(N+2). OP(N+1) a OP(N+2) sa vytvoria podľa situácie S(N+1). Situačné riadenie obsahuje algoritmy pre riešenie nasledovných situácií:

- Štandardná situácia (zavážanie NP štandardnou dávkou).
- Využitie neštandardnej dávky (neštandardná dávka je dávka sformovaná z brám rovnakej kvality, dĺžky od 5 do 10 m - dvojnásobná dĺžka štandardnej dávky).
- Prechod od zavážania po jednej bráme na zavážanie v dávkach.
- Prechod od zavážania v dávkach na zavážanie po jednej bráme.
- Zavážanie po jednej bráme.
- Oneskorený príchod teplej vsádzky.
- Predčasný príchod teplej vsádzky.
- Prechod z prevádzky 4 narážacích pecí na prevádzku 3 narážacích pecí.
- Prechod z prevádzky 3 narážacích pecí na prevádzku 4 pecí.
- Odstávka pece.
- Interaktívny spôsob tvorby grafikonu zavážania NP.

V prípade ak nepríde žiadna informácia od operátora systému v on-line režime, prebieha vytváranie OP(N+1), OP(N+2) algoritmom na princípe štandardnej dávky (ŠD). V momente riešenia niektorej inej situácie S(N), operátor systému vyvolá menu vyššie uvedených situácií. Kurzorom zvolí typ situácie a tým aj modul pre riešenie vybranej situácie. Modul generuje na terminál masku, kde sa vyplnia informácie potrebné pre definovanie zvolenej situácie, napr:

- čas omeškania teplých brám oproti plánovanému,
- číslo položky,
- počet brám a pod.

Každý modul prepočíta možnosť riešenia situácie vzhľadom na zotrvačnosť systému. Ak sa dá problém riešiť, vytvorí sa OP(N+1), OP(N+2).

Riešenie situácie môže byť v ľubovoľnom čase. V prípade, ak danú situáciu nevie systém riešiť, oznámi to operátorovi systému. Operátor môže vybrať iný modul, alebo môže prejsť na interaktívny režim a vytvorí OP(N+1), OP(N+2).

8.3.3 Riešenie pomocou CLP

Z popisu riešenia riadiaceho systému narážacích pecí je zrejmé, že úloha nájdenia optimálneho rozvrhu zavážania narážacích pecí (blok riadenie zavážania na obrázku 8) je samostatnou a zároveň veľmi dôležitou úlohou. Od vygenerovaného grafikonu sa totiž potom odvíja výpočet optimálneho ohrevu pre tento rozvrh (blok programové riadenie na obrázku 8) a je nakoniec prispôsobované skutočnej situácii na úseku NP (blok korekčné riadenie na obrázku 8).

Práve na otázku hľadania optimálneho rozvrhu zavážania brám do narážacích pecí som sa zameral a skúsil som riešiť túto úlohu prostriedkami CLP. Prístup popísaný v predchádzajúcej časti vedie na deterministický algoritmus, ktorý zoskupuje brámy do dávok zodpovedajúcich priemernej šírke jednotlivých zón v peci v postupnosti danej grafikonom TŠP, ktorý je požadovaný na výstupe NP.

Tento algoritmus bol realizovaný v jazyku FORTRAN 77. Program obsahuje 10 ďalších podprogramov riešiacich najčastejšie sa vyskytujúce mimoriadne situácie, ktoré podľa skúsenosti vznikajú na úseku NP. Spúšťanie jednotlivých podprogramov je riadené operátorom.

Riešenie pomocou CLP jazyka *ECL^{i}PS^e* [CsoPar 96], [ParCso 96] naproti tomu prináša prehľadný deklaratívny program pre nájdenie optimálneho rozvrhu, ktorého podrobný popis nasleduje.

Pre popis úlohy prostriedkami CLP je potrebné ju formulovať ako úlohu spĺňania ohraničení a potom použiť postup popísaný v predchádzajúcich častiach tejto kapitoly o metodológii. Prvým dôležitým krokom k takejto formulácii je definovanie premenných a ohraničení medzi nimi, ktoré dostatočne definujú prípustné riešenia.

Premenné. V našom prípade je počet premenných v danej úlohe závislý od počtu brám v zadanom grafikone TŠP ktorý máme rozvrhom dosiahnuť. Pre každú bránu z grafikonu budeme potrebovať dve premenné s konečnými doménami (FD), a síce *CisloPece* a *VstupCas* (čiže číslo pece, do ktorej bude bráma zasunutá a vstupný čas, t.j. čas kedy bude zasunutá).

Ohraničenia majú rôzny charakter a sú postupne v programe definované spolu s údajovými štruktúrami, ktoré sú pre ich reprezentáciu potrebné. Detaily o jednotlivých typoch použitých ohraničení budú vysvetlené v ďalšom. Najskôr uvediem schému základného programu, v ktorom sú vyznačené 4 základné kroky, v rámci ktorých sa definujú technologické ohraničenia.

```
rozvrh_zavazania(S) :-
        pocet_peci(F),
        dlzka(L),
        bramy_vo_vnutri(S1),
        grafikon(G),
```

```
defininuj_struktury(G, F, L, S2),   % 1. krok
definuj_ohranicenia(S2),            % 2. krok
spoj(S1, S2, S),
definuj_naklady(S, Cost),
kontrola_obsadenia(S2, S),          % 3. krok
kontrola_nakladov(S, Cost),         % 4. krok
hladaj_riesenie(S2, S),
vysledok(S).
```

1. krok

Predikát defininuj_struktury/4 vytvára zoznam S2 štruktúr typu brama/4, ktorého dĺžka zodpovedá veľkosti grafikonu. Každá bráma výsledného grafikonu je v ňom reprezentovaná jednou štruktúrou. Táto štruktúra má nasledovný tvar:

```
brama(Typ, CisloPece, VstupCas, VystupCas)
```

Typ - jednoznačné typové číslo brámy (všetky údaje súvisiace s brámami tohoto typu sú reprezentované v programe štruktúrou typu typ_bramy/4). Ide o vopred známu celočíselnú konštantu.

CisloPece - číslo NP, do ktorej bude táto bráma zasunutá. Toto je premenná typu FD, môže nadobúdať hodnotu 1 až počet aktívnych pecí (F).

VstupCas - vstupný okamžik, kedy bude bráma zasunutá do NP. Opäť ide o premennú FD. Počiatočná doména je daná rozdielom času výstupu (VystupCas) a minimálnej doby potrebnej na ohrev tejto brámy (Min).

VystupCas - výstupný čas, kedy bude bráma vytlačená z NP a pôjde na valcovanie. Ide o vopred známu konštantu danú grafikonom TŠP.

Pre vstup parametrov brám daného typu slúžia fakty typ_bramy/4:

```
typ_bramy(Typ, Zdroj, Sirka, Kvalita)
```

Typ - má ten istý význam ako v štruktúrach brama/4.

Zdroj - definuje pôvod brámy s ohľadom na ich vstupnú teplotu, zatiaľ rozlišujeme brámy studené (hodnota 0) a teplé (hodnota 1).

Sirka - šírka brámy - celé číslo, skutočná šírka je jeho 20 násobkom.

Kvalita - akosť ocele z ktorej sú brámy tohto typu - celé číslo.

Pre doménové premenné je nutné definovať ich počiatočnú doménu (t.j. konečnú množinu hodnôt, ktoré tá ktorá premenná môže nadobúdať). V *ECL^{i}PS^e* na tento účel slúži preddefinovaný operátor ::/2.

```
CisloPece :: 1 .. F,
VstupCas :: 1 .. (VystupCas - Min)
```

2. krok

Po zostavení zoznamu štruktúr (S2) typu brama/4 a načítaní zoznamu takýchto štruktúr týkajúcich sa brám, ktoré už sú v peciach (zoznam S1) sa definujú ďalšie ohraničenia na jednotlivé argumenty štruktúr brama/4.

Najjednoduchšie dve sa definujú pre každú bránu osobitne a majú takýto tvar:

```
VystupCas - VstupCas #>= Min,
VystupCas - VstupCas #<= Max.
```

Prvé z nich zabezpečuje minimálny čas ohrevu (Min) potrebný na to, aby bráma dosiahla požadovanú valcovaciu teplotu. Druhý zase obmedzuje maximálny dovolený čas pobytu (Max) v peci, aby sa bráma neprehrievala.

Ďalšie ohraničenie nemá numerický, ale symbolický charakter. Ide o zabudované ohraničenie alldifferent/2 jazyka *ECL^{i}PS^e*, ktoré zabezpečuje, aby sa žiadne dve hodnoty z daného zoznamu navzájom nerovnali.

```
alldifferent(Vstupne_casy_S2)
```

pričom Vstupne_casy_S2 je zoznam premenných VstupCas zo zoznamu štruktúr S2. Teda v jednom okamžiku môže vstupovať do pece len jedna bráma.

3. krok

Pre potreby tejto úlohy sme definovali dve nové symbolické ohraničenia za pomoci knižnice FD jazyka *ECL^{i}PS^e*. Prvé z nich volané v programe:

```
kontrola_obsadenia(S2, S)
```

Toto ohraničenie kontroluje obsadenosť pece (súčet šíriek brám, ktoré sú v peci), aby sa zistilo, či sa zasunutím nenaruší požiadavka, že bráma samozrejme nesmie trčať z pece, a tiež maximálny nevyužitý priestor nepresiahne vopred zadanú hodnotu. Toto ohraničenie sa aktivuje zakaždým, keď sa v programe nejaká bráma vyberie pre zasunutie do niektorej pece.

4. krok

Posledné dôležité symbolické ohraničenie sa týka vyhodnotenia riešenia z pohľadu nákladov a slúži teda pre optimalizáciu.

```
kontrola_nakladov(S, Cost)
```

Účel tohoto ohraničenia je vyhodnotenie nákladov aktuálneho riešenia (t.j. hodnotu kriteriálnej funkcie Cost, ktorej definícia je uvedená ďalej). Akonáhle je táto hodnota väčšia ako doteraz najlepšieho nájdeného riešenia, generovanie aktuálneho rozvrhu sa zastaví a navracaním sa vráti do posledného bodu vetvenia v predikáte hladaj_riesenie/2.

Optimalizácia

Kriteriálna funkcia musí v tomto prípade obsahovať špeciálne požiadavky pre nájdenie riešenia, ktoré minimalizuje spotrebu energie potrebnej na ohrev brám, ako aj strát spôsobených opalom kovu.

Vychádzal som z predpokladu, že náklady na ohrev stúpajú s počtom situácií, kedy sú bezprostredne vedľa seba v NP brámy s rozdielnymi vstupnými teplotami. Naviac je tento vzostup priamo úmerný rozdielu teplôt susedných brám v peci.

Za týmto účelom označme brámy, ktoré budú zasunuté do tej istej pece za sebou indexmi i a j (teda $j = i+1$). Pre každú takúto dvojicu brám sa vypočíta elementárny príspevok nákladov za túto dvojicu $Naklady_{ij}$ podľa nasledovného algoritmu:

ak $Zdroj_i = Zdroj_j$
  potom $Naklady_{ij} = VstupCas_i - VstupCas_j - 1$
ináč ak $Zdroj_i > Zdroj_j$
  potom $Naklady_{ij} = 4 * (VstupCas_i - VstupCas_j)$
ináč $Naklady_{ij} = 2 * (Max - VstupCas_j - VstupCas_i)$.

Celkové náklady sú potom dané súčtom jednotlivých elementárnych nákladov pre všetky susedné dvojice a pre všetky pece. Nech N je počet brám v danej peci, potom:

$$Naklady = \sum_{\forall pece} \sum_{i=1}^{N-1} Naklady_{ij}$$

Rozvrh s najmenšími celkovými nákladmi je potom vybraný ako finálne riešenie.

Generovanie hodnôt

Definovanie premenných a stanovenie ohraničení samozrejme nestačí na nájdenie riešenia, aj keď sa môžu propagáciou ohraničení podstatne zúžiť domény premenných.

Proces generovania hodnôt (zodpovedá tomu, čo sa v angličtine označuje pojmom labelling) zabezpečuje systematické priradzovanie hodnôt premenným z ich aktuálnych domén. Tento proces je sprevádzaný opäť propagáciou ohraničení, ktoré naviazanie premennej obvykle vyvoláva. V hlavnom programe sa realizuje predikátom

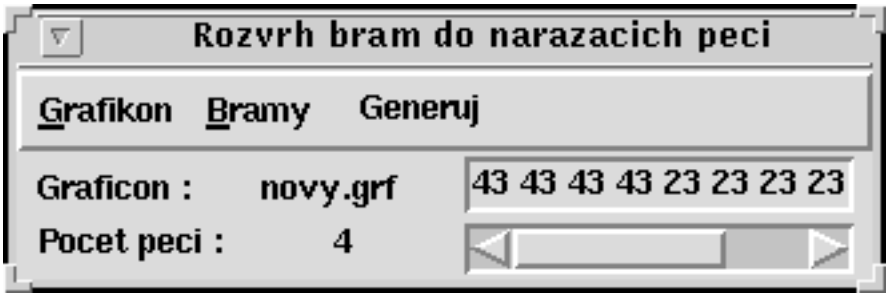
```
hladaj_riesenie(S2, S)
```

V našom programe sa postupne vyberajú brámy podľa výstupného poradia (daného grafikonom) a ich výstupným časom sa snažia priradiť zodpovedajúce brámy na vstupe, ktoré budú v danom okamžiku zatlačené. Samozrejme sa za každým takýmto výberom propaguje novo definovaná informácia prostredníctvom ohraničení podrobne popísaných vyššie. V prípade že dôjde k zisteniu nekonzistentnosti, t.j. doména niektorej z premenných sa zúži na 0, čo svedčí o tom že pre daný výber hodnôt nie je možné nájsť riešenie, dôjde k návratu do posledného bodu výberu a zvolí sa iná hodnota z aktuálnej domény premennej.

8.3.4 Zhodnotenie výsledkov

K programu popísanému podrobne v predchádzajúcej časti 8.3.3 bolo vytvorené aj grafické používateľské rozhranie [Svieženy 96] s pomocou grafickej knižnice Tcl/Tk. Prepojenie medzi Tcl/Tk a systémom *ECL^{i}PS^e* zabezpečuje knižnica ProTcl. Vytvorené používateľské rozhranie umožňuje ovládanie programu bežiaceho v systéme *ECL^{i}PS^e*, ale aj definovanie nového grafikonu, jeho zápis do súboru, prípadne načítanie už existujúceho grafikonu zo súboru. Po nájdení optimálneho riešenia poskytuje možnosť grafického modelovania pohybu brám v peci. Pohľad na základné menu je uvedený na obrázku 10. Modelovanie pohybu brám v peci pre nájdené optimálne riešenie je na obrázku 11.

Popísaný program pre nájdenie optimálneho rozvrhu brám do narážacích pecí je schopný nájsť optimálne riešenie pre ľubovoľný počet pecí ľubovoľnej dĺžky a ľubovoľný grafikon brám pre valcovanie. Avšak jeho reálna použiteľnosť (myslím tým trvanie behu programu po nájdenie optimálneho riešenia do piatich minút) sa obmedzuje napr. pre konkrétne podmienky NP pred teplou širokopásovou valcovacou traťou vo VSŽ-Oceľ s.r.o. na grafikony približne pätinovej dĺžky v porovnaní so skutočnou situáciou na úseku NP.



Obrázok 10 Pohľad na hlavné menu programu pre hľadanie optimálneho rozvrhu zavážania brám do NP.

To by síce na jednej strane nemusel byť problém, nakoľko každý grafikon je možné rozdeliť na postupnosť kratších grafikonov a robiť rozvrh pre každý z nich. To by v praxi znamenalo nutnosť generovania nového rozvrhu v kratších časových intervaloch, t.j. napr. nie každé 4 hodiny ako je tomu teraz, ale napr. každú polhodinu. Nevýhodou tohoto postupu je skutočnosť, že optimalizácia sa tak realizuje na menšom rozsahu. Výhodou by naopak mohlo byť pružnejšie reagovanie na náhle zmeny situácie na úseku NP.

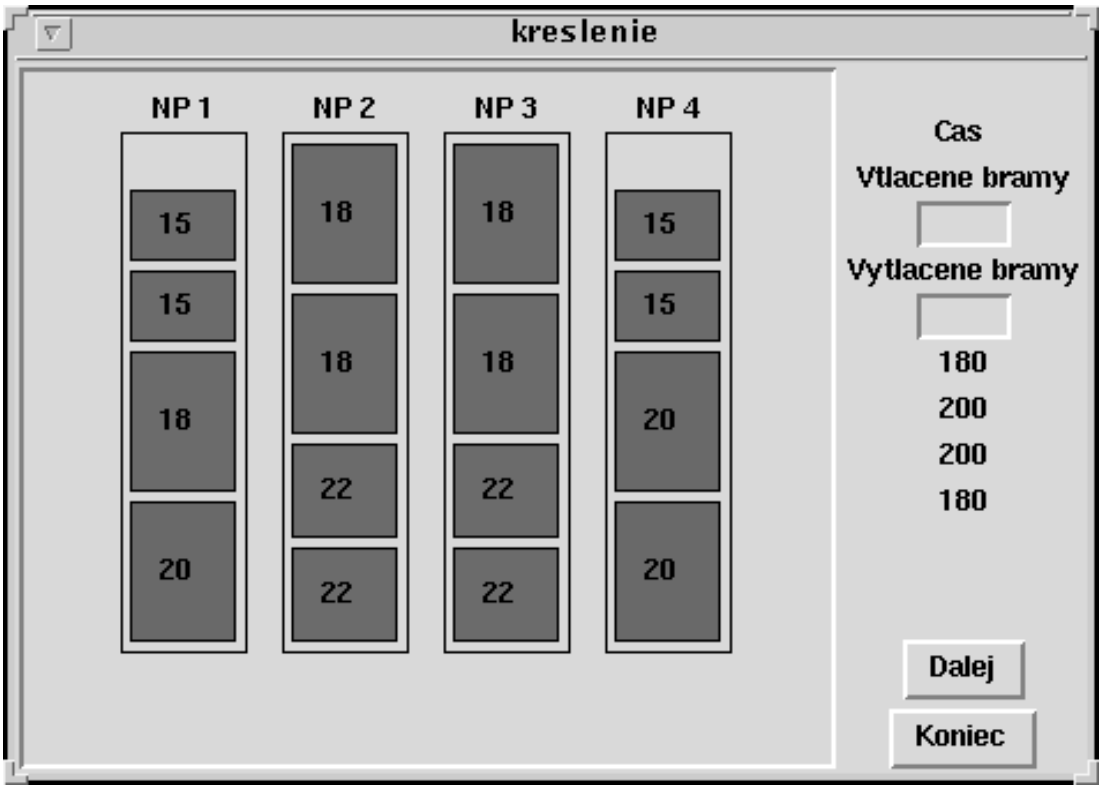
Príčinu spomínaného javu vidím najmä v dvoch hlavných dôvodoch. Prvým z nich je skutočnosť, že propagácia ohraničení sa obmedzuje najmä na základné ohraničenia stanovené v krokoch 1 a 2 (viď. predchádzajúca časť 8.3.3). Ohraničenia definované v krokoch 3 a 4 nepôsobia dopredu ale sa obmedzujú v podstate len na dodatočnú kontrolu po vygenerovaní potenciálneho riešenia. Druhým dôvodom je veľký počet

navzájom veľmi podobných riešení, ktoré sú generované a nemôžu byť včas zavrhnuté napríklad z dôvodu vyšších nákladov.

Hlavná výhoda riešenia za pomoci CLP oproti procedurálnym programovacím jazykom (ako tomu bolo aj u riešenia navrhnutého na KRVP fakulty BERG) je omnoho kratšia doba potrebná na vývoj programu, ako aj jeho neporovnateľne menší rozsah (niekoľko 100 riadkov kódu v jazyku *ECL^{i}PS^e* oproti niekoľkým tisíckam riadkov programu v jazyku FORTRAN 77). CLP sa takto potvrdil ako vhodný prototypovací nástroj. Naviac sa potvrdila aj vhodnosť jazyka *ECL^{i}PS^e* pre definovanie vlastných symbolických ohraničení (krok 3 a 4 v programe) špecifických pre danú aplikáciu.

Z pohľadu implementovaných disjunktných ohraničení je nutné konštatovať, že žiadne z nich nebolo možné použiť pre účely tejto aplikácie, nakoľko zdroj, ktorý je v tomto prípade reprezentovaný číslom NP do ktorej bude bráma zasunutá, nie je dopredu jasný (ako je tomu napr. u úloh typu job-shop). Preto nie je možné generovať disjunkcie vzhľadom na zdieľané zdroje. Tak isto optimalizácia je v tejto úlohe robená odlišným spôsobom.

Napriek týmto skutočnostiam som sa rozhodol zaradiť túto aplikáciu do dizertačnej práce, lebo poukazuje na základné črty CLP ako nástroja na riešenie reálnych rozvrhovacích úloh, a to tak jeho výhody, ako aj nevýhody, na ktoré je možno naraziť.



Obrázok 11 Pohľad na okno, v ktorom je možné modelovať pohyb brám v peci pre nájdené optimálne riešenie.

K a p i t o l a 9

VÝSLEDKY DIZERTÁCIE A KONKRÉTNE ZÁVERY

V tejto kapitole uvádzam obsiahlejší súhrn hlavných výsledkov dizertačnej práce. Najdôležitejšie prínosy dizertačnej práce sú reprezentované názvami jednotlivých častí tejto kapitoly.

V práci som sa zaoberal problematikou riešenia úloh časového rozvrhovania v prostredí logického programovania ohraničení (CLP z anglického Constraint Logic Programming). Cieľom týchto úloh je nájsť podľa určitého kritéria optimálny rozvrh pre realizáciu presne danej skupiny úloh (operácií) tak, aby boli splnené všetky zadané ohraničenia.

V práci som sa snažil venovať všetkým základným aspektom tejto problematiky (kapitola 2). Kvôli objektívnemu pohľadu som uviedol aj prehľad ostatných najčastejšie používaných metód riešenia rozvrhovacích úloh (časť 2.2).

Za prvý prínos považujem porovnanie uvedených metód na riešenie úloh rozvrhovania, v rámci ktorého som vytipoval kritériá porovnania a zhodnotil postavenie CLP v kontexte ostatných metód (kapitola 4).

Hlavný prínos však vidím v špeciálnej implementácii, otestovaní a porovnaní rôznych spôsobov reprezentácie a propagácie disjunktných ohraničení v prostredí CLP (kapitoly 5 a 6), konkrétne v jazyku *ECL^{i}PS^e*. Špecifickým prínosom sú aj nové, efektívnejšie postupy pre riešenie optimalizácie v CLP, ktoré sú zhrnuté v kapitole 7.

Naviac je v práci navrhnutá metodológia pre riešenie úloh časového rozvrhovania prostriedkami CLP a použitá pre riešenie reálnej rozvrhovacej aplikácie (kapitola 8).

V tejto záverečnej kapitole uvádzam súhrn poznatkov získaných v dizertačnej práci, ako aj náčrt ďalšieho možného vývoja v oblasti. Kapitolu som rozdelil do piatich častí. Prvá sa venuje zhrnutiu poznatkov z porovnania jednotlivých metód na riešenie úloh časového rozvrhovania a postavenia CLP medzi nimi (9.1).

Druhá časť je zameraná na súhrn výsledkov získaných úpravou, implementáciou a testovaním rôznych postupov na riešenie disjunktných ohraničení (9.2) ako najkritickejšej zložky pri úlohách rozvrhovania.

Tretia časť je venovaná prínosom dizertačnej práce v oblasti postupov pre nájdenie optimálneho riešenia v prostredí CLP (9.3).

Štvrtá časť podáva súhrn poznatkov z metodológie riešenia úloh časového rozvrhovania v prostredí CLP (9.4) a konečne piata, posledná časť, načrtáva ďalšie možné smery vývoja v predmetnej oblasti (9.5).

9.1 Porovnanie metód riešenia rozvrhovacích úloh

Pri analyzovaní desiatich v časti 2.2 popísaných metód na riešenie úloh rozvrhovania som dospel k záveru, že ich je možné v zásade rozdeliť do troch skupín navzájom príbuzných metód:

1. **Úplné metódy**, ktoré zaručujú nájdenie (optimálneho) riešenia ak existuje (lineárne programovanie, branch-and-bound, spĺňanie ohraničení).
2. **Neúplné metódy** (hill climbing, simulované žíhanie, tabu search, genetické algoritmy), ktoré neprehľadávajú celý priestor prehľadávania a zaručujú tak iba nájdenie suboptimálneho riešenia.
3. Iné, **neštandardné metódy**, kam možno zahrnúť neurónové siete a expertné systémy.

Z pohľadu tohoto členenia CLP poskytuje celú množinu algoritmov, ktorá prakticky pokrýva prvú skupinu metód. Je naviac omnoho pružnejšie v reprezentácii netypických ohraničení, s ktorými sa často stretávame v konkrétnych aplikáciách. Je výborným nástrojom najmä v etape prototypovania, t.j. v štádiu analyzovania novej úlohy a návrhu nových postupov jej riešenia, nakoľko programovanie v CLP sa vyznačuje deklaratívnosťou a podstatne menším rozsahom zdrojového kódu, než napr. u tradičných procedurálnych programovacích jazykov.

Nemožno však prehlásiť, že niektorý z uvedených postupov je univerzálne najlepší. Každú aplikáciu je nutné posúdiť osobitne a zvoliť si tú metódu, ktorá najlepšie vyhovuje požiadavkám kladeným na jej vyriešenie, ale i podmienkam a schopnostiam riešiteľa..

Aby sa riešiteľ rozvrhovacej aplikácie mohol rozhodnúť ktorú metódu použiť, potrebuje do hĺbky poznať tak riešený problém, ako aj jednotlivé metódy. Pri tejto analýze je nutné brať do úvahy nasledovné kritériá:

1. **Splniteľnosť alebo optimalizácia.**
2. **Výpočtový čas verzus optimálnosť riešenia.**
3. **Špecifikácia problému.**
4. **Voľba algoritmu a implementácia.**

Veľká výhoda systémov na programovanie ohraničení (najmä CLP) je v tom, že riešiteľ rozvrhovacej aplikácie môže využívať metódy lineárneho programovania, branch-and-bound, algoritmy spĺňania ohraničení a prípadne ďalšie metódy bez toho, aby sa ich musel učiť a sám ich implementoval.

9.2 Efektívne riešenie disjunktných ohraničení pri úlohách rozvrhovania

Disjunktné ohraničenia, vyplývajúce zo zdieľania rovnakých zdrojov rôznymi úlohami sú pôvodcom zložitosti rozvrhovacích aplikácií. Preto som sa vo svojej práci najviac zameral práve na tento ich aspekt.

Za účelom efektívneho narábania s disjunktnými ohraničeniami a vypracovania doporučení na ich použitie som spracoval a pre prostredie CLP upravil viacero algoritmov, ktoré som potom implementoval v CLP jazyku *ECL^{i}PS^e*. Algoritmy som potom zvlášť a v kombináciách testoval na náhodne generovaných úlohách typu job-shop (viď. podrobnejší opis v časti 6.1) a na jednej reálnej rozvrhovacej aplikácii (6.2).

Pokiaľ je mi známe, takéto komplexné porovnanie nebolo zatiaľ publikované a pokladám ho za dôležité pre efektívnejšie využitie CLP na účely riešenia úloh rozvrhovania.

Základným spôsobom reprezentácie disjunktných ohraničení je forma disjunkcií ako nezávislých alternatív. Táto je účinná (t.j. propagácia tohoto ohraničenia prináša zužovanie domén premenných a tým redukciu priestoru prehľadávania) až po výbere jednej alternatívy, t.j. až v priebehu prehľadávania. Nezabráni preto veľkému počtu návratov, nakoľko sa informácia o disjunkcii nevyužíva dopredu, pred začatím prehľadávania, ale až v jeho priebehu.

Tento základný spôsob je plne postačujúci pre úlohy menšieho rozsahu (napr. úlohy typu job-shop pre 5 strojov a 5 výrobkov). Pre úlohy väčšieho rozsahu je však už nedostatočný (viď. tabuľka 3). Preto je ho nutné kombinovať s ďalšími spôsobmi reprezentácie a propagácie disjunktných ohraničení.

Pri skúmaní hranovej konzistencie, základného spôsobu doprednej propagácie disjunktných ohraničení ktorá sa snaží vyťažiť dodatočnú informáciu len na základe dvojíc navzájom časovo disjunktných úloh sa ukázalo, že pre rozvrhovacie účely je úplne postačujúca hranová B-konzistencia (tento algoritmus propagácie bol implementovaný vo forme ohraničenia disjunction_choose/5). Táto síce propaguje len zmeny hraníc domén jednotlivých premenných (nie vo vnútri domén), avšak je časovo omnoho menej náročná ako úplná hranová konzistencia (implementovaná vo forme ohraničenia disjunction/5).

Najväčší efekt však prináša implementácia niektorých poznatkov z algoritmov vychádzajúcich z celých množín disjunktných úloh (nie len z dvojíc), ktoré dokážu vyťažiť viac informácie a ich propagácia je preto omnoho účinnejšia (napr. na základe algoritmu Carlier a Pinson implementované symbolické ohraničenie disjunctive/3 - viď. tabuľka 4). Druhé implementované komplexné ohraničenie na báze intervalov úloh (task_intervals/3 - viď. tabuľka 5) bolo účinné len v kombinácii s disjunctive/3. Takto viedlo k zmenšeniu preskúmanej časti priestoru riešení, avšak časovo bolo efektívnejšie len pri úlohe rozvrhu prác na výstavbe mosta, ktorá má veľmi ťažký dôkaz optimálnosti.

9.3 Vylepšenia optimalizácie v CLP

Ďalšie zvýšenie efektívnosti priniesol nový spôsob prehľadávania priestoru riešení pri hľadaní optimálneho riešenia. Tradičný algoritmus MINMAX (popis možno nájsť v časti 2.4) totiž vedie k veľkému počtu iterácií. Preto boli navrhnuté nové algoritmy LOGARITMICKÝ MINMAX a LOGARITMICKÝ MINIMIZE (podrobnejšie viď. 7.2), ktoré vychádzajúc z heuristicky nájdených odhadov dolnej a hornej hranice

intervalu v ktorom leží optimálne riešenie, postupujú jeho delením na polovicu, čím sa počet nevyhnutných iterácií podstatne zníži.

Tieto metódy priniesli zároveň najlepšie výsledky, ktoré sú zhrnuté v tabuľke 11. V drvivej väčšine úloh je efektívnosť tohto postupu vysoká. Stále však ešte zostávajú úlohy, u ktorých môže byť tento postup o niečo pomalší ako klasický MINMAX (z testovaných úloh bola takou iba reálna úloha návrhu časového harmonogramu výstavby mostu).

Avšak aj v týchto prípadoch je nutné uviesť, že algoritmy omnoho rýchlejšie konvergujú k optimálnemu riešeniu a rýchlejšie poskytujú riešenia blízke optimu, vrátane vyčíslenia maximálnej vzdialenosti doposiaľ nájdeného riešenia od optima (v percentách). Najviac času sa totiž spotrebuje pri dôkaze optimálnosti riešenia (t.j. vtedy, keď už je vzdialenosť od optimálneho riešenia minimálna).

Pre efektívne fungovanie algoritmov LOGARITMICKÝ MINMAX a LOGARITMICKÝ MINIMIZE je dôležitý kvalitný odhad najmä hornej hranice (teda heuristicke nájdenie prvého riešenia).

V tomto smere som testoval osem heuristík a navrhol jednu novú (podrobnejšie viď. v časti 7.1), ktorá je najefektívnejšia v najväčšej skupine úloh a aj v ostatných úlohách, kde iná heuristika našla lepší odhad, sa odhad pomocou novej heuristiky líši od najlepšieho v priemere najmenej (viď tabuľka 10).

9.4 Metodológia riešenia rozvrhovacích úloh

Poznatky získané v priebehu tejto práce som zhrnul do ucelenej metodológie riešenia úloh časového rozvrhovania v prostredí CLP, ktorá bola podrobnejšie popísaná v predchádzajúcej kapitole 8.

Postup pri riešení úlohy časového rozvrhovania v prostredí CLP možno rozdeliť do piatich základných bodov.

1. **Definovanie premenných a ich počiatočných domén**.

Základ tvoria obvykle premenné reprezentujúce začiatky jednotlivých úloh (operácií), ktoré je potrebné rozvrhnúť v čase. Ich počiatočné domény sú preto závislé od požadovanej presnosti rozvrhu (hodiny, smeny, dni a pod.). Vzhľadom na túto presnosť je potom možné diskretizovať interval rozvrhu a stanoviť tak konečné domény pre jednotlivé premenné napr. ako interval *<0, Limit>*. *Limit* možno najjednoduchšie stanoviť ako súčet dĺžok trvaní všetkých úloh, ktoré je nutné rozvrhnúť.

2. **Definovanie precedenčných časových ohraničení**.

Precedenčné ohraničenia vyjadrujú jednoznačné časové relácie medzi dvojicami premenných (začiatky úloh). Tieto je možné rozčleniť na niekoľko základných typov, ako to definoval už [Allen 83]. Prehľad jednotlivých typov časových relácií a spôsobu ich reprezentácie v CLP (na príklade CLP jazyka ECL^iPS^e) je uvedený v časti 8.1.3.

3. **Voľba spôsobu reprezentácie disjunktných ohraničení**.

Ohraničenia vyjadrujúce zdieľanie zdrojov viacerými úlohami sa používajú dvojakým spôsobom.

- Ako nezávislé alternatívy (body výberu), ktoré sa skúmajú v priebehu prehľadávania (podrobnejšie viď. časť 5.1).
- Ako ohraničenia zabezpečujúce väčšiu mieru propagácie. V tomto smere existuje niekoľko alternatív s rôznou mierou získanej informácie, (podrobnejšie v častiach 5.2 až 5.4).

Zhrnutie doporučení pre použiteľnosť jednotlivých spôsobov reprezentácie disjunktných ohraničení bolo uvedené v predchádzajúcej časti 9.2.

4. **Heuristicke nájdenie prvého riešenia (odhad hornej hranice) a odhad dolnej hranice.**

Čím bližšie bude prvé vygenerované riešenie k skutočnému optimálnemu riešeniu, tým väčšiu šancu máme, že sa naozaj dočkáme jeho nájdenia, resp. že nájdeme nejaké riešenie, ktoré bude dostatočne blízke optimu.

Na základe ôsmych testovaných heuristík som navrhol novú (tzv. EST+), ktorá má najlepšie priemerné správanie a najväčšie percento najlepších odhadov. V prípade najzložitejších úloh, kde je nevyhnutné nájsť najlepší odhad hornej hranice, je možné nájsť odhady pomocou troch metód, ktoré jediné mali najlepšie odhady pri najväčších úlohách a vybrať najlepší nájdený odhad (podrobnosti boli uvedené v časti 7.1).

5. **Úplné prehľadávanie so znižujúcou sa hornou hranicou až po nájdenie globálneho optima a dôkaz optimálnosti riešenia**.

Obvyklý postup implementovaný v jazykoch CLP postupne znižuje hornú hranicu buď po jednotkách (ak hľadáme optimálny rozvrh), alebo po určitých krokoch (ak hľadáme rozvrh s určitou toleranciou vzdialenosti od optimálneho riešenia). Novo navrhnuté algoritmy - LOGARITMICKÝ MINMAX a LOGARITMICKÝ MINIMIZE v drvivej väčšine prípadov zrýchľujú tento proces a vždy dávajú presnú informáciu o maximálnej možnej vzdialenosti nájdeného riešenia od optima v percentách (podrobnosti boli uvedené v časti 7.2).

9.5 Ďalšie smery vývoja v sledovanej oblasti

V zásade je možné predpokladať, že prostriedky CLP budú naďalej, a to v stúpajúcej miere využívané na riešenie reálnych úloh časového rozvrhovania. Dokonca aj v prípade, že sa pre cieľovú implementáciu vyberie iný programovací prostriedok, môže byť CLP veľmi účinným prototypovacím nástrojom.

Toto tvrdenie potvrdzuje aj komerčná úspešnosť niekoľkých systémov takéhoto charakteru, ako napr. *CHIP* alebo *ILOG Solver* [Cras 93]. Zároveň je však možné vybadať, že pre účinné riešenie zložitých reálnych aplikácií je nevyhnutné ďalej zdokonaľovať špecializované algoritmy akým je napr. Carlier a Pinsonov pre úlohy typu job-shop. Tieto sa po efektívnej adaptácii do prostredia CLP v podobe globálnych symbolických ohraničení stávajú veľmi účinným a ľahko použiteľným nástrojom (ako to ukázal aj príklad implementovaného ohraničenia disjunctive/3).

Ďalším dôležitým pokrokom by bola možnosť využitia niektorých neúplných metód prehľadávania v prostredí CLP. Niektoré prvé pokusy v tomto smere už boli urobené [CasLab 95]. Tieto by boli využiteľné najmä v etape približovania sa k optimálnemu riešeniu. Pre dôkaz optimálnosti je totiž nevyhnutné ponechať úplné prehľadávanie, ktoré prostriedky CLP poskytujú.

Nakoľko spektrum praktických aplikácií úloh časového rozvrhovania je veľmi široké (čiastočne to možno vidieť už aj z úloh realizovaných v rámci tejto dizertačnej práce), naďalej je potrebné rozširovať škálu ohraničení, ktorých reprezentácia je v jazyku CLP efektívne podporovaná, ako aj rôzne heuristiky využiteľné v procese prehľadávania. Na základe týchto prostriedkov si totiž potom používateľ môže lepšie vyladiť finálnu aplikáciu a využiť jej špecifiká na zlepšenie efektívnosti.

No a v neposlednom rade je tu aj otázka grafického rozhrania, ktoré by umožňovalo riešiteľovi sústrediť sa na problém samotný, nakoľko generovanie mnohých typov ohraničení, resp. spôsobov prehľadávania je možné zautomatizovať. Takýto prostriedok by mal poskytovať aj účinné nástroje na ladenie programov, čo nie je jednoduchý problém. Jazyk *ECL^{i}PS^e* je v tomto smere prvým, ktorý poskytuje grafické prostredie pre ladenie CLP aplikácií vyvinutých pomocou knižnice konečných domén. Komerčné systémy naproti tomu poskytujú skôr grafické nástroje pre ľahší vývoj koncovej aplikácie, ako aj rozhrania pre priemyselne používané databázové systémy.

LITERATÚRA

[AggBel 91] Aggoun, A. and Beldiceanu, N.: Overview of the CHIP Compiler System, in *Proc. of the 8th International Conference on Logic Programming*, Paris, June 1991, pp. 775-789.

[AggBel 93] Aggoun, A. and Beldiceanu, N.: Extending CHIP in Order to Solve Complex Scheduling and Placement Problems, *Math. Comut. Modelling*, Vol. 17, No. 7, pp. 57-73, 1993.

[Allen 83] Allen, J.F.: Maintaining Knowledge about Temporal Intervals. *Communications ACM*, Vol. 26, No. 11, November 1983, pp. 832-843.

[AndMach 96] Andrássyová, E. and Mach, M.: Genetické algoritmy a ich aplikácie, *Lékář a technika*, 1996.

[BapPap 95] Baptiste, P. and Le Pape, C.: A Theoretical and Experimental Comparison of Constraint Propagation Techniques for Disjunctive Scheduling. *In Proc. of the 14th International Conference on Artificial Intelligence*, Montreal, Quebec, 1995, pp. 600-606.

[Bellone a kol. 95] Bellone, J., Chamard, A. and Fischler, A.: Constraint Logic Programming Decision Support Systems for Planning and Scheduling Aircraft Manufacturing at Dassault Aviation. In *Proc. of the 3rd Int. Conf. on PAP*, Paris, April 1995, pp. 63-67.

[BenCol 93] F. Benhamou, A. Colmerauer (eds.): *Constraint Logic Programming, Selected Research*, MIT Press 1993.

[BelCon 94] Beldiceau, N. and Contejean, E.: Introducing Global Constraints in CHIP, *Math. Comut. Modelling*, Vol. 20, No. 12, 1994, pp. 97-123.

[Benhamou 93] Benhamou, F.: Boolean Algorithms in Prolog III, in *Constraint Logic Programming, Selected Research*, ed. by Benhamou, F., Colmerauer, A., MIT Press 1993, pp. 307-325.

[Bratko 86] Bratko, I.: *Prolog Programming for Artificial Intelligence*, Addison Wesley, 1986.

[Brisset a kol. 94] Brisset, P., Fruewirth, T., Lim, P., Meier, M., Le Provost, T., Schimpf, J. and Wallace, M.: *ECLiPSe* 3.4 - Extensions User Manual, ECRC, 1994.

[CarPin 89] Carlier, J. and Pinson, E.: An Algorithm for Solving the Job-Shop Problem. *Managment Science*, Vol. 35, No. 2, 1989, pp. 164-176.

[CasLab 94] Caseau, Y. and Laburthe, F.: Improved CLP Scheduling with Task Intervals. In *Proc. of the International Conference on Logic Programming*, 1994, pp. 369-383.

[CasLab 95] Caseau, Y. and Laburthe, F.: Disjunctive Scheduling with Task Intervals. LIENS Technical Report 95-25, Laboratoire d'Informatique de I'Ecole Normale Superieure, 1995.

[Crabtree 95] Crabtree I.B.: Resource Scheduling - Comparing Simulated Annealing with Constraint Programming. *BT Technology*, Vol. 13, No. 1, 1995, pp. 121-127.

[Cras 93] Cras, J. Y.: A Review of Industrial Constraint solving Tools. A report in *AI Perspective Series*, Oxford, United Kingdom, 1993.

[CsoPar 96] Csontó, J. and Paralič, J.: A Look at CLP - Theory and Application. In *Proc. of the 13th European Meeting on Cybernetics and Systems Research EMCSR'94*, Vienna, 1996, pp. 1125-1129.

[CsoPar 97] Csontó, J. and Paralič, J.: A Look at CLP - Theory and Application. *Applied Artificial Intelligence*, Vol. 11, No. 1, 1997, pp. 59-70.

[Dechter 92] Dechter, R.: Constraint Networks, in *Encyclopedia of Artificial Intelligence*, 2nd edition, Wiley and Sons, 1992, pp 276-285.

[DiazCod 93] Diaz, D. and Codognet, P.: A Minimal Extension of the WAM for clp(FD). In *Proc. of the 10th International Conference on Logic Programming*, ICLP'93, Budapest, Hungary, MIT Press 1993.

[Dincbas a kol. 88a] Dincbas, M., Van Hentenryck, P., Simonis, H., Aggoun, A., Graf, T. and Berthier, F.: The Constraint Logic Programming Language CHIP. In *Proc. of the International Conference on 5th Generation Computer Systems*, ICOT, 1988.

[Dincbas a kol. 88b] M. Dincbas, H. Simonis, P. Van Hentenryck: Solving a Cutting-Stock Problem in Constraint Logic Programming, in *Proc. of 5th International Conference on Logic Programming*, MIT Press, 1988, pp. 43-58.

[Dincbas a kol. 90] Dincbas, M., Simonis, H. and Van Hentenryck, P.: Solving Large Combinatorial Problems in Logic Programming. *Journal of Logic Programming*, Vol. 8, No. 1/2 , 1990, pp. 75-93.

[Fox 87] Fox, M.: *Constraint-Directed Search: A Case Study of Job-Shop Scheduling*, Morgan Kaufman, 1987.

[French 82] French, S.: *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop*. Wiley & Sons, England, 1982.

[Fruehwirth a kol. 93] Fruehwirth, T., Herold, A., Kuechenhoff, V., Le Provost, T., Lim, P., Monfroy, E. and Wallace M.: Constraint Logic Programing. An Informal Introduction. Technical Report ECRC-93-5, ECRC, 1993.

[Fruehwirth 94b] Fruehwirth, T.: Constraint Handling Rules, Chapter in *Constraint Programming: Basics and Trends* (Podelski, A. ed.), Springer LNCS 910, May 1994, pp. 90-107.

[Heinze a kol. 92] N. Heinze, S. Michaylov, P. Stuckey: CLP($\Re$) and Some Electrical Engineering Problems, in *Journal of Automated Reasoning*, No. 9, 1992, pp. 231-260.

[Hentenryck 89] Van Hentenryck, P.: *Constraint Satisfaction in Logic Programming*. Logic Programming Series, The MIT Press, Cambridge, MA, 1989.

[HenCar 88] Van Hentenryck, P. and Carillon, J.P.: Generality versus Specificity: an Experience with AI and OR techniques, in *Proc. of the AAAI'88*, pp. 660-664.

[Hentenryck a kol. 93] Van Hentenryck, P., Saraswat, V. and Deville, Y.: Design, Implementations, and Evaluation of the Constraint Language cc(FD). Technical Report CS-93-02, CS Department, Brown University, 1993.

[JafLas 87] Jaffar, J. and Lassez J. L.: Constraint Logic Programming. In *Proc. of the 14th Symp. POPL'87*, Munich, January 1987, pp. 111-119.

[JafMich 87] Jaffar, J. and Michaylov, S.: Methodology and Implementation of a CLP System. In *Proc. of the 4th International Conference on Logic Programming*, MIT Press, Melbourne, 1987, pp. 196-217.

[JafMah 94] Jaffar, J. and Maher, M.: Constraint Logic Programming: A Survey, *Journal of Logic Programming*, Vol. 19, No. 20, 1994, pp. 503-581.

[Jánošíková 94] Jánošíková, Ľ.: Optimalizačné úlohy na dopravných sieťach a metóda tabu search. Kandidátska dizertačná práca, Vysoká škola dopravy a spojov v Žiline, 1994.

[JouSol 93] Jourdan, J. and Sola, T.: The versatility of handling disjunctions as Constraints. In. *PLILP*, 1993, pp. 60-74.

[Kirkpatrick a kol. 83] Kirkpatrick, S., Gelatt, C.D. and Vecchi, M.P.: Optimization by Simulated Annealing. *Science*, No. 220, 1983, pp. 671-680.

[Mach 96] Mach, M.: Using Genetic Algorithms for Problems with Soft Constraints. In *Proc. of the 3rd Workshop on Artificial Intelligence Techniques AIT'96*, Brno, 1996, pp. 173-174.

[Malindžák 96] Malindžák, D.: Situačné riadenie zavážania narážacích pecí VSŽ a.s., *AT&P Journal*, č. 4, 1996, str. 25-32.

[Mayoh 94] Mayoh, B., Tyungu, E. and Uustalu, T.: Constraint Satisfaction and Constraint Programming: A Brief Lead-In, in *Constraint Programming*, Mayoh, B., Tyungu, E., Penjam, J. (eds.), Springer Verlag, 1994, pp. 1-24.

[MeiBri 95] Meier, M. and Brisset, P.: Open Architecture for CLP, Technical report, ECRC-95-10, European Computer-Indusry Research Centre, 1995.

[MutTho 63] Muth, J. and Thomson, G.: *Industrial Scheduling*. Prentice Hall, 1963.

[Lever a kol. 95] Lever, J., Wallace, M. and Richards, B.: Constraint Logic Programming for Scheduling and Planning. In *BT Technology Today*, Vol.13, No.1, 1995, pp. 73-80.

[Lim 94] Lim, P.: Implementation of the *ECL^{i}PS^e* Rational Constraint Solver, Technical Report ECRC-94-23, European Computer-Indusry Research Centre, 1994.

[Lloyd 84] Lloyd, J.W.: *Foundations of Logic Programming*, Springer Verlag, 1984.

[MudPre 95] Mudambi, S. and Prestwich, S.: Improved Branch and Bound in Constraint Logic Programming, paper submitted to CP'95, Munich 1995.

[Paralič 95] Paralič, J.: Constraint Logic Programming - An Overview. Rigorózna práca, ECRC Mníchov - Technická univerzita v Košiciach, Jún 1995.

[ParCso 96] Paralič, J. and Csontó, J.: The CLP Approach to Solve a Scheduling Application. In *Proc. of the 7th International Symposium on INFORMATION SYSTEMS'96*, Varaždin, September 1996, pp. 179-190.

[ParSab 95] Paralič, J. and Sabol, T.: Analysis of the Constraint Satisfaction Algorithms with respect to Configuration Design. Technical Report TR-Encode-TUKE-2-95, Technical University of Košice, 1995.

[Paralič a kol. 96] Paralič,J., Sabol, T., Mach, M. and Hatala, M.: Configuration Design as Constraint Satisfaction. In *Proc. of the 10th International Computer Science Conference microCAD'96*, Section H: Information Technology in Mechanical Engineering, Miskolc, February 1996, pp. 64-69.

[ProWal 92] Le Provost, T. and Wallace, M.: Generalized Constraint Propagation Over the CLP scheme, Technical Report ECRC-92-1, European Computer-Indusry Research Centre, 1992.

[Puget 94] Puget, J.F.: A C++ Implementation of CLP, Technical Report 94-01, ILOG S.A., Gentilly, France, 1994.

[Schmotzer 97] Schmotzer, M.: Analýza a návrh nových prostriedkov pre riešenie úloh časového rozvrhovania v prostredí logického programovania ohraničení. Diplomová práca, Katedra kybernetiky a umelej inteligencie, Technická univerzita Košice, 1997.

[Svieženy 96] Svieženy, P.: Moduly systému rozvrhovania brám. Diplomová práca, Katedra kybernetiky a umelej inteligencie, Technická univerzita Košice, 1996.

[Tsang 93] Tsang, E.: *Foundations of Constraint Satisfaction*. Academic Press, 1993.

[Tsang 95] Tsang, E.: Scheduling Techniques - A Comparative Study. In *BT Technology Today*, Vol.13, No.1, 1995, pp. 16-28.

[Wallace 95] Wallace, M.: Introducing The Practical Applicatios of Constraints. In *Proc. of the 7th Conference on Practical Application of Constraint Technology*, Paris, April 1995.

[WueMue 95] Wuertz, J. and Mueller, T.: Constructive Disjunction Revisited. In *Workshop Logische Programmierung*, 1995.

PROGRAMOVÁ PRÍLOHA

P.1 Popis implementácie ohraničenia pre hranovú B-konzistenciu

Toto ohraničenie má päť argumentov, pričom Aa, Ba sú doménové premenné označujúce počiatočné časy dvoch operácií zdieľajúcich ten istý zdroj. Ad Bd sú celé čísla označujúce trvania týchto operácií. Flag je tiež doménová premenná s dvoma možnými hodnotami: 1 ak operácia A bude vo výslednom rozvrhu pred operáciou B a 2 ak tomu bude naopak.

V nasledujúcom programe vysvetľujúce komentáre začínajú vždy znakom %.

```
disjunctionB(Aa, Ad, Ba, Bd, Flag):-
    (integer(Flag) ->    % ak je už nastavený príznak usporiadania
     (
      ( Flag == 1 ->      % ak je príznak 1 (operácia A musí byť pred B)
          Aa + Ad #<= Ba; % musí platiť toto precedenčné ohraničenie
        ( Flag == 2 ->  % ak je príznak 2 (operácia B musí byť pred A)
            Ba + Bd #<= Aa; % musí platiť toto precedenčné ohraničenie
            fail              % inak nemožno splniť disjunkciu
            )
        )
      )
      ; % ak nie je ešte nastavený príznak, treba otestovať hranice domén
     ( domain_copy(Aa, Aa1), % vytvor kópie domén pre účely testovania
        domain_copy(Ba,Ba1), % aby sa pritom neporušili pôvodné
        subcall(Aa1 + Ad #<= Ba1, _) % test 1) - otestuj, ale nepropaguj -
     ->                                % môže zmeniť iba domény Aa1,Ba1
    ( domain_copy(Aa, Aa2),
        domain_copy(Ba,Ba2),
        subcall(Ba2 + Bd #<= Aa2, _) % test 2) - otestuj, ale nepropaguj -
      ->                               % môže zmeniť iba domény Aa2,Ba2
        mindomain(Aa1,Aa1m),% ulož spodné hranice domén Aa1,Aa2 do
        mindomain(Aa2,Aa2m), % pomocných premenných Aa1m,Aa2m
        min(Aa1m,Aa2m,NAam), % zisti minimum a ulož ho do NAam
        maxdomain(Aa1,Aa1M), maxdomain(Aa2,Aa2M), % to isté pre
        max(Aa1M,Aa2M,NAaM), % horné hranice domén Aa1, Aa2
        mindomain(Aa, Aam), maxdomain(Aa, AaM), % zisti hranice
                                    %  pôvodnej premennej Aa
```

```
    (  (NAam \== Aam ; NAaM \==AaM) % ak sa zmenila aspoň jedna hranica
      ->
       New_Aa :: NAam..NAaM, % vytvor novú doménu
       Aa = New_Aa                % uprav doménu pôvodnej premennej
    ;
      true
    ),              % rovnaký postup pre úpravu domény premennej Ba
   mindomain(Ba1,Ba1m), mindomain(Ba2,Ba2m),
   min(Ba1m,Ba2m,NBam),
   maxdomain(Ba1,Ba1M), maxdomain(Ba2,Ba2M),
   max(Ba1M,Ba2M,NBaM),
   mindomain(Ba, Bam), maxdomain(Ba, BaM),
    (  (NBam \== Bam ; NBaM \==BaM)
      ->
       New_Ba :: NBam..NBaM,
       Ba = New_Ba
    ;
      true
    ), % nakoniec treba ešte uspať tento cieľ a nastaviť jeho aktiváciu pri
       % naviazaní príznaku Flag a pri zmene dolnej alebo hornej hranice
       % domény niektorej z premenných Aa , Ba
   suspend(disjunctionB(Aa,Ad,Ba,Bd,Flag),4,Flag -> inst),
   suspend(disjunctionB(Aa,Ad,Ba,Bd,Flag),4, [Aa,Ba] -> min),
   suspend(disjunctionB(Aa,Ad,Ba,Bd,Flag),4, [Aa,Ba] -> max),
  ; % ak neuspel test 2) musí sa vnútiť usporiadanie A pred B
   Aa + Ad #<= Ba,
   Flag = 1   % nastav zodpovedajúco príznak Flag
 )
; % ak neuspel test 1) musí sa vnútiť usporiadanie B pred A
 Ba + Bd #<= Aa,
 Flag = 2    % nastav zodpovedajúco príznak Flag
)).
```

P.2 Popis programu pre riešenie úloh typu job-shop

V tejto časti sú uvedené najdôležitejšie časti programu používaného pre testovanie úloh typu job-shop so vstupnými údajmi v tvare popísanom v časti 6.1.1. Vysvetľujúce komentáre v programe začínajú vždy znakom %.

```
:- nodbgcomp. % nekompiluj informácie pre debugger (rýchlejšie)
:- lib(fd).        % knižnica konečných domén

jobshop(Limit) :-      % Limit je zadaná horná hranica nákladov, t.j. čas
    % ukončenia všetkých operácií, ktorý sa snačíme minimalizovať
E::0..Limit,           % Počiatočná doména premennej - dĺžka rozvrhu
job_shop_data(NbJobs,NbMachines,T0List), % vstupné údaje
map(T0List,extend_task(Limit),TaskList),% rozšíri štruktúru pre
     % každú operáciu na  t(Cvyr, Cop, Cstr, Trv, Start, Koniec).
    % kde Start je doménová  premenná pre počiatočný čas operácie.
    % a End je doménová premenná pre čas ukončenia operácie.
    % Obe premenné sa nainicializujú na 0..Limit a naviac sa položí
    % ohraničenie tvaru Start + Trv #<= Koniec.
job_task_matrix(NbJobs,TaskList,JobMatrix0),  % zoradí operácie
    % v zozname TaskList podľa jednotlivých výrobkov a uloží
    % ich do samostatných zoznamov v matici JobMatrix0
map(JobMatrix0,sort,JobMatrix), % usporiada ju podľa čísel operácií
mach_task_matrix(NbMachines,TaskList,MachMatrix),% analogicky sa
     % vytvorí matica strojov.
     % Nasleduje výpočet dolnej hranice rozvrhu a jej výpis
map(JobMatrix,tasks_time,JobTimes),
map(MachMatrix,tasks_time,MachTimes),
append(JobTimes,MachTimes,LowerTimes),
maximum(LowerTimes,LowerBound),
write('lower bound: '),write(LowerBound),nl,flush(user),
    % nasleduje úprava časov, ktoré zostávajú do ukončenia
    % jednotlivých operácií, vzhľadom na ich predpísané poradie
map0(JobMatrix,till_times),
    % ďalšie ohraničenia
map0(JobMatrix,prec_constr(E)), % orezanie domény podľa
    % vypočítanej dolnej hranice na E::LowerBound..Limit
map0(MachMatrix,disj_constr), % tu sa ukladá ohraničenie
    % disjunctive/3 popísané v časti 4.3 (pri príslušných testoch)
    % nasleduje heuristika
map(MachMatrix,make_disjunction,DisjuncsList), % vytvor množinu
    % disjunkcií medzi dvojicami operácií d(t1,t2) pre jednotlivé
    % stroje a následne vyhlaď tento zoznam na jednoúrovňový
append(DisjuncsList,Disjuncts0),
```

```
bench_sort(Disjuncts0,Disjuncts), % usporiadaj disjunkcie
    % zostupne podľa celkovej dĺžky dvojice operácií D0 + D1
disjunction_2(Disjuncts),
     % tu sa alternatívne pre testovacie účely vkladalo ohraničenie pre
     % hranovú konzistenciu (či už úplnú, alebo hranovú B-konzistenciu)
min_max(% hľadaj optimálne riešenie pomocou branch-and-bound[20]
(           % Tu je možné použiť alternatívne aj noevé stratégie hľadania
          % optimálneho riešenia popísané v časti 7.2.
disjunctions_as_choice(Disjuncts), % disjunkcie ako alternatívy
labeling(TaskList),          % výberu. Prehľadávanie cez labeling
domain_minimum(E,E)
),
E,LowerBound,Limit,0), % minimalizuj čas rozvrhu E, pričom dolná
    % hranica je LowerBound a horná Limit, hľadaj optimálne
    % riešenie (povolená odchýlka od optimálneho riešenia 0%)
nl, write( 'minimum:  '), write(E), nl, nl, % minimálna dĺžka rozvrhu
print_gant(NbMachines,TaskListl).      % nájdený optimálny rozvrhu
```

Nasledujú niektoré dôležité predikáty použité v hlavnom programe. Najmä tie pre riadenie prehľadávania vnútri min_max/5, ako aj jednotlivé alternatívy použitia disjunkcií

```
labeling([]).
labeling([t(_,_,_,_,X,_)|Y]) :-
        indomain(X),
        labeling(Y).

disjunctions_as_choice([]).
disjunctions_as_choice([Disj|Disjs]) :-
        disjunction_as_choice(Disj),
        disjunctions_as_choice(Disjs).

disjunction_as_choice(d(t(_,_,_,D0,S0,_),t(_,_,_,_D1,S1,_))) :-
        S1 #>= S0 + D0.
disjunction_as_choice(d(t(_,_,_,_D0,S0,_),t(_,_,_,D1,S1,_))) :-
        S0 #>= S1 + D1.
    % pre zisťovanie počtu navracaní možno pridať ďalšiu klauzulu
disjunction_as_choice(_Choice) :-
        incval(backtrack_local), % inkrementuje externé počítadlo návratov
        fail.
```

A teraz časť venovaná ohraničeniam. Jednak precedenčným, ale aj ďalším časovým ohraničeniam. Nakoniec sú popísané jednotlivé alternatívy použitia disjunktných ohraničení v tomto programe.

```
prec_constr([_Task],_E). % generovanie precedenčných ohraničení
```

[20] Popis zabudovaného predikátu min_max/5 možno nájsť v časti 2.4 tejto práce.

```
prec_constr([t(_,TN0,_,D0,S0,_)|TR],E) :-
        TR = [t(_,TN1,_,D1,S1,_)|_R],
        TN0 < TN1,!,
        D1 + S1 #<= E,
        D0 + S0 #<= S1,
        prec_constr(TR,E).
prec_constr([T1,T2|_],_E) :-
        write('tasks '),write(T1),write(' and '),write(T2),
        write(' are not sorted! '),nl,fail.

bound_constr(t(_,_,_,D,S,_),E) :-
        S + D #<= E.

extend_task(t(A,B,C,D),t(A,B,C,D,E,_F),Limit) :-
        E::0..Limit.
```

Disjunkcie sa ešte v programe využívajú pre heuristické usporiadanie a teda následný výber podľa súčtu dĺžok dvojice operácií zdieľajúcich ten istý zdroj. Najprv sa vyberajú najdlhšie dvojice.

```
make_disjunction([],[]).
make_disjunction([Task|TLR],DJR) :-
        make_disjunction(TLR,DJR0),
        make_disjunction_i(TLR,Task,DJR0,DJR).

make_disjunction_i([],_Task,DJR,DJR).
make_disjunction_i([Task1|DR],Task0,DJR0,DJR) :-
        (
           Task0 @< Task1 ->
           Disj = d(Task0,Task1) ;
           Disj = d(Task1,Task0)
        ),
        make_disjunction_i(DR,Task0,[Disj|DJR0],DJR).
```

Pre posilnenie účinnosti disjunktných ohraničení, mimo základného použitia disjunkcií ako nezávislých alternatív v rámci min_max/5 (disjunctions_as_choice), možno ďalšie implementované spôsoby (popísané podrobne v kapitole 5) reprezentovať nasledovne. V prípade hranovej konzistencie ide o volanie disjunction_2(Disjuncts) v hlavnom programe. Tento predikát kladie pre každú dvojicu disjunkcií dodatočné ohraničenie a je definovaný nasledovne:

```
disjunction_2([]).
disjunction_2([Disj|DS]) :-
     disjunction_2_i(Disj),   % v prípade úplnej hranovej konzistencie
 % disjunction_2B_I(Disj), % v prípade hranovej B-konzistencie
     disjunction_2(DS).
```

1. Predikát disjunction/5 (popis viď. časť 5.2.1) reprezentujúci úplnú hranovú konzistenciu je potom použitý nasledovne:

```
disjunction_2_i(d(t(_,_,_,D0,S0,_),t(_,_,_,D1,S1,_))) :-
    F :: 1..2,
    disjunction(S0,D0,S1,D1,F).
```

2. Predikát disjunction_choose/5 (popis viď. časť 5.2.2) reprezentujúci hranovú B-konzistenciu je zadávaný rovnakým spôsobom:

```
disjunction_2B_i(d(t(_,_,_,D0,S0,_),t(_,_,_,D1,S1,_))) :-
    F :: 1..2,
    disjunction_choose(S0,D0,S1,D1,F).
```

3. Predikát disjunctive/3 realizujúci algoritmus vychádzajúci z práce Carlier a Pinson (popis viď. časť 5.3) je volaný nasledovne:

```
disj_constr(TL) :-
        gen_cumul(TL,StartList,DurationList,_ResourceList),
        disjunctive(StartList,DurationList,_Flags).

gen_cumul([],[],[],[]).
gen_cumul([t(_,_,_,D,S,_)|TR],[S|SL],[D|DL],[1|RL]) :- !,
        gen_cumul(TR,SL,DL,RL).
```

Z pomocných predikátov spomeniem len map/3 a map0/2 používané na mapovanie nejakej funkcie na zoznam štruktúr (teda vykonanie nejakého cieľa s každým z prvkov zoznamu).

```
map([],_,[]).
map([H|T],Goal0,[MH|MT]) :-
        Goal0 =.. [F|Args],
        Goal =.. [F,H,MH|Args],
        Goal,
        map(T,Goal0,MT).

map0([],_).
map0([H|T],Goal0) :-
        Goal0 =.. [F|Args],
        Goal =.. [F,H|Args],
        Goal,
        map0(T,Goal0).
```

P.3 Popis programov pre odhad hraníc optimálneho riešenia

Predikát pre odhad hornej hranice heuristikou EST+

Vstupom predikátu odhadni_hornu_hranicu/2 je zoznam štruktúr (TaskList) popisujúcich jednotlivé operácie, ktoré je potrebné rozvrhnúť v čase. Štruktúry majú tvar t(Cvyr, Cop, Cstr, Trv, Start, Koniec, Zvysok), kde:

Cvyr - číslo výrobku,
Cop - číslo operácie,
Cstr - číslo stroja, na ktorom má byť operácia vykonaná,
Trv - trvanie operácie,
Start - čas začiatku operácie (doménová premenná),
Koniec - čas ukončenia operácie (doménová premenná, Koniec=Start+Trv),
Zvysok - zvyšková práca danej operácie (súčet trvaní operácií, ktoré musia byť ešte vykonané za touto operáciou).

Druhým argumentom je premenná (E) s konečnou doménou, ktorá označuje koniec celého rozvrhu. Do domény tejto premennej sa premietajú všetky zmeny spôsobené priradzovaním konkrétnych hodnôt pre časy začiatku jednotlivých operácií v zozname TaskList.

Minimum domény tejto premennej po skončení výpočtu predstavuje samotnú hornú hranicu výsledného optimálneho rozvrhu. Hodnoty priradené počiatočným časom pre jednotlivé operácie zodpovedajú nájdenému suboptimálnemu riešeniu.

```
odhadni_hornu_hranicu(TaskList,E) :-
    odhadni_hornu_hranicu_(TaskList,E). % predikát končí neúspechom, aby
                % sa nenaviazali premenné pre štartovacie časy v TaskList
odhadni_hornu_hranicu(_,_).          % aby hlavný predikát skončil úspechom

odhadni_hornu_hranicu_(TaskList,E):-
    labeling1(TaskList), % tu dochádza ku generovaniu rozvrhu heuristikou
          % tým,že sa priradia počiatočné časy všetkým premenným
    mindomain(E,UpperBound), % samotné zistenie minimálnej dĺžky rozvrhu
    setval(horna_hranica,UpperBound),!,fail.
          % zistenú hodnotu hornej hranice uloží do globálenej premennej

labeling1([t(J1,T1,M1,D1,S1,E1,TE1)|Rest]) :-
    mindomain(S1,Min),  % najskorší možný čas začiatku operácie S1
    est(Rest, t(J1,T1,M1,D1,S1,E1,TE1), Min, t(_,_,_,_,S,_,_), RestTasks),
          % rozhodni, či táto operácia je vhodnejší kandidát na výber
          % v nasledujúcom kroku
    indomain(S),          % vyber konkrétnu hodnotu z domény S (najmenšiu)
    labeling1(RestTasks).% rekurzívne pre zvyšné operácie
labeling1([]).
```

```
est([], X, _, X, []).
est([t(J1,T1,M1,D1,S1,E1,TE1)|RestTasks],
     t(J,T,M,D,S,E,TE), Min, X, [t(J,T,M,D,S,E,TE)|RestNew] ) :-
    mindomain(S1,Min1),
    (Min1 < Min ; Min1 = Min, TE1 > TE), !,
        % jadro heurisiky: vyberá sa operácia s najskorším možným časom
        % začiatku, v prípade rovnosti potom tá s dlhším zvyškom prác
    est(RestTasks, t(J1,T1,M1,D1,S1,E1,TE1), Min1, X, RestNew).
est([t(J1,T1,M1,D1,S1,E1,TE1)|RestTasks], t(J,T,M,D,S,E,TE), Min, X,
    [t(J1,T1,M1,D1,S1,E1,TE1)|RestNew] ) :-
    est(RestTasks, t(J,T,M,D,S,E,TE), Min, X, RestNew).
```

Predikát pre odhad dolnej hranice

Predikát tasks_time/3 má tri argumenty. Prvým je zoznam štruktúr (TaskList) popisujúcich jednotlivé operácie, ktoré je potrebné rozvrhnúť v čase. Štruktúry majú rovnaký tvar ako v predchádzajúcom prípade pri odhade hornej hranice.

Druhý argument je premenná (Time) na ktorú sa naviaže nájdená dolná hranica a nakoniec tretí argument predstavuje hornú hranicu optimálneho riešenia (Limit).

```
tasks_time(TaskList,Time,Limit) :-
    tasks_time(TaskList,0,Time1,Limit,MinS,Limit,MinE),
        % tu sa zistí súčet trvaní všetkých operácií v zozname (Time1),
        % minimálny čas začiatku (MinS) množiny operácií a minimálnu
        % zvyškovú prácu (MinE) spomedzi operácií v zozname
    Time is Time1 + MinS + MinE. % výsledný odhad dolnej hranice

tasks_time([t(_,_,_,D,Start,_,TE)|TL],S0,S,MS,MinS,ME,MinE) :-
    mindomain(Start,MinS1), % zisti minimálny možný čas začiatku
        % aktuálnej operácie a porovnaj ho s doteraz najmenším
    MinS1 < MS, TE < ME, !, % ak je jeho hodnota menšia a zároveň je
        % aj jeho zvyšková práca menšia než doteraz nájdená
    S1 is S0 + D, % pridaj trvanie tejto operácie k súčtu trvaní doteraz
        % preskúmaných operácií
    tasks_time(TL,S1,S,MinS1,MinS,TE,MinE).
        % uvažuj ďalej minimálny čas začiatku a zvyškovú prácu tejto
        % operácie

tasks_time([t(_,_,_,D,Start,_,TE)|TL],S0,S,MS,MinS,ME,MinE) :-
    mindomain(Start,MinS1),
    MinS1 < MS, !, % minimálny čas začiatku je menší ako doteraz
        % najmenší nájdený, avšak zvyšková práca nie
    S1 is S0 + D,
    tasks_time(TL,S1,S,MinS1,MinS,ME,MinE).
        % uprav iba najskorší čas začiatku a súčet trvaní operácií
```

```
tasks_time([t(_,_,_,D,Start,_,TE)|TL],S0,S,MS,MinS,ME,MinE) :-
        TE < ME, !, % zvyšková práca je menšia ako doteraz najmenšia
                % nájdená, avšak minimálny čas začiatku nie
        S1 is S0 + D,
        tasks_time(TL,S1,S,MS,MinS,TE,MinE).
                % uprav iba najmenšiu zvyškovú prácu a súčet trvaní operácií

tasks_time([t(_,_,_,D,Start,_,TE)|TL],S0,S,MS,MinS,ME,MinE) :-
        S1 is S0 + D,
        tasks_time(TL,S1,S,MS,MinS,ME,MinE).
                % uprav iba súčet trvaní operácií

tasks_time([],S,S,MinS,MinS,MinE,MinE).
                % na konci rekurzie skopíruj nájdené hodnoty súčtu trvaní
                % operácií v zozname (S),najskoršieho času začiatku (MinS)
                % a najmenšej zvyškovej práce (MinE) do výstupných argumentov
```

Tento predikát je volaný pre každý zdieľaný zdroj (resp. množinu operácií, ktoré ho zdieľajú) a u úloh typu job-shop aj pre každý výrobok (t.j. množinu operácií, z ktorých sa jeho výroba skladá). Napokon sa pre dolnú hranicu optimálneho riešenia vyberie najväčšia z vypočítaných hodnôt.

P.4 Implementácia algoritmu LOGARITMICKÝ MINMAX

Predikát log_min_max(Ciel, Premenna, Minimum, Maximum, PO) nájde také riešenie cieľa Ciel, ktorý minimalizuje hodnotu doménovej premennej Premenna v rozsahu intervalu <Minimum, Maximum>. Riešenie mimo tohoto intervalu predikát nenájde. PO je povolená odchýlka od optima v percentách.

```
log_min_max(Ciel, Premenna, Minimum, Maximum, PO):-
aktualizuj_cas
   dvar_remove_smaller(Premenna,Minimum),
   dvar_remove_greater(Premenna,Maximum),
        % odstránenie hodnôt mimo intervalu <Minimum, Maximum>
        % z domény premennej pre čas ukončenia celého rozvrhu
   wake, % explicitne odštartuje propagáciu ohraničení
setval(hodnota(0),0), setval(hodnota(1),0),
   log_min_max_(Ciel, Premenna, Minimum, Maximum, PO).

log_min_max_( Ciel, Premenna, Minimum,  Maximum, PO) :-
    Odchylka is round((Maximum-Minimum)/Minimum*10000)/100,
         % Aktuálna odchýlka zaokrúhlená smerom nahor.
    (Odchylka =< PO, !, % ak bola dosiahnutá požadovaná presnosť riešenia
       log_min_max__(Ciel,Premenna,Minimum,Maximum)
         % výpis najlepšieho nájdeného riešenia
      ;  % v opačnom prípade (ešte nebola dosiahnutá požadovaná presnosť)
       nl, write(Minimum),write('..'),write(Maximum),
       write('  Odchylka od optima je najviac '),write(Odchylka), write(' %'),
       cas_vypoctu, % zistenie aktuálneho času výpočtu
       getval(hodnota(0),MH),getval(hodnota(1),PH_), % zistenie aktuálneho
          % počtu opakovaní hornej hranice (externá premenná hodnota(1)) a
          %hornej hranice v poslednom kroku (externá premenná hodnota(0))
       ( MH=Maximum,incval(hodnota(1)),PH=PH_ % ak sa horná hranica
          % nezmenila, inkrementuj počítadlo
       ;
         MH\=Maximum,setval(hodnota(0),Maximum), % ináč ulož novú hornú
         setval(hodnota(1),1),PH=0 % hranicu a počítadlo nastav na 1
       ),
```

```
        % prvá podmienka pre zmenu stratégie na MINMAX:
        % ak sa interval medzi hornou a dolnou hranicou zúži na 5 a menej
    ( Maximum-Minimum < 6,
      TestHranica_ = Maximum %nová horná hranica sa bude meniť len o 1
    ;   % druhá podmienka, ak sa horná hranica opakuje 3 a viackrát
      MH=Maximum,PH>1,
      TestHranica_ = Maximum %nová horná hranica sa bude meniť len o 1
    ;
      not((MH=Maximum,PH>1)), % ak sa horná hranica zmenila
      TestHranica_ is Maximum - fix((Maximum-Minimum)/2) % uprav novú
        %  hornú hranicu na stred intervalu
    ),
    ( TestHranica_ = Maximum, % tu sa realizuje deklarovaná zmena
      TestHranica is TestHranica_ - 1 % hornej hranice o 1 smerom dole
                                      % (stretégia MINAMAX)
    ;
      TestHranica_ \= Maximum, % ak nebola deklarovaná zmena stratégie
      TestHranica = TestHranica_ na MINMAX % použi delenie intervalu
                                      % na polovicu
    )
not((   % v prípade že sa nájde riešenie s novou hranicou nákladov,
        % potom sa sa táto stáva hornou hranicou intervalu nákladov
        % zároveň však neuspeje táto klauzula predikátu log_min_max_/5
        % takže výpočet bude pokračovat druhou klauzulou viď *
      dvar_remove_greater(Premenna,TestHranica),
      wake,
      Ciel, mindomain(Premenna,A), % trvanie aktuálne najlepšieho
         % nájdeného rozvrhu je A - túto hodnotu
      setval(log_min_max_value,A)  % ulož do globálnej premennej
      setval(ciel(0),Ciel), setval(ciel(1),A) % a nájdené riešenie tiež
     )),
        % v prípade že sa nenájde riešenie s novou hranicou nákladov,
        % potom sa nová hranica stáva dolnou hranicou intervalu
Hranica is TestHranica + 1,
dvar_remove_smaller(Premenna,Hranica),
wake,
( Maximum=Hranica, !, % ak sa uz interval zuzil na 0
  !,log_min_max__(Ciel,Premenna,Hranica,Maximum) % výpis riešenia
;
  !,log_min_max_(Ciel,Premenna,Hranica,Maximum,PO)
        % ináč algoritmus pokračuje rekurzívne ďalej pre nový interval
  )
),!.
```

```
log_min_max_( Ciel, Premenna, Minimum, _Maximum, PO) :-
        % *) tu pokračuje výpočet ak sa v predchádzajúcom kroku podarilo
        % nájsť riešenie s novou hranicou nákladov
   getval(log_min_max_value,Hranica),
        % zistí sa odpamatataná hodnota nákladov najlepšieho nájdeného
        % riešenia (novej hornej hranice)
   dvar_remove_greater(Premenna,Hranica),
   wake,
   log_min_max_(Ciel,Premenna,Minimum,Hranica,PO),!.

log_min_max__( Ciel, Premenna, Minimum,_Maximum) :-
        % tento predikát slúži pre výpis najlepšieho nájdeného riešenia
    getval(ciel(1),A), % vyber z globálnej premennej náklady aktuálne
        % najlepšieho riešenia
    ( A > 0, Maximum >= A, getval(ciel(0),Ciel), Min=A;
      not((A > 0, Maximum >= A)), Ciel,
      mindomain(Premenna,Min)
    ),
    Odchylka is round(10000*(Min-Minimum)/Minimum)/100,
   nl,nl,
   (Odchylka=:=0,
      write('Nasiel som OPTIMALNE riesenie s minimalnou cenou '),
      write(Min),writeln('.')
    ;
      Odchylka=\=0,
      write('Nasiel som riesenie s minimalnou cenou '),
      write(Min),writeln('.'),
      write('Jeho odchylka od optima je maximalne '),write(Odchylka),
      writeln(' %.')
   ), cas_vypoctu.

log_min_max__(_Ciel,_Premenna,_Hranica, Maximum) :-
   nl,write('Riesenie pre maximum '),write(Maximum),
   writeln(' nebolo najdene !'),!,fail.
```