Nasledovný text je vrybraný z: **J. Paralič (1997) Riešenie úloh rozvrhovania logickým programovaním ohraničení. Dizertačná práca, Katedra kybernetiky a umelej inteligencie, FEI, Technická univerzita v Košiciach, máj 1997, 114 s.**

ROZVRHOVANIE NA VIACERÝCH PROCESOROCH

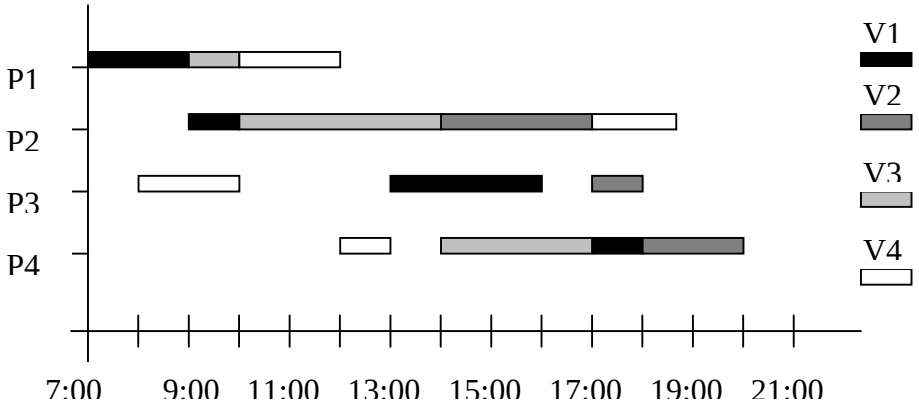
1.1 Príklad

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.

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.

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 1 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*).

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.

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.

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

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 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.

SÚČASNÝ STAV PROBLEMATIKY

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 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: $a_i = d_i - r_i$

[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.

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 ($\overline{F}$).
   - 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.

[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.

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

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

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, metóda vetvenia a medzí, prehľadávanie tabu) 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 3 rozdelené do skupín a na základe zvolených kritérií porovnané.

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.

Metóda vetvenia a medzí

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

Metóda vetvenia a medzí 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. 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 metódy vetvenia a medzí 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.

Metódu vetvenia a medzí 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 ohraničenia). 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 ⟨7..20⟩ (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 ⟨7..17⟩. 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 ⟨9..15⟩.

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 metódy vetvenia a medzí 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.

Prehľadávanie Tabu

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.

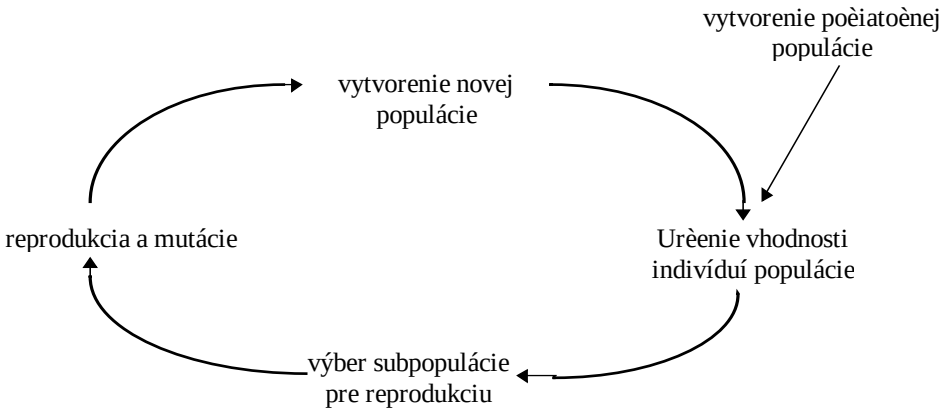
V praxi sa zaužívali dva spôsoby vytvárania novej populácie. Jeden z nich (generatívny) nahrádza všetky indivíduá starej aktuálnej generácie novými indivíduami. Pri druhom sa nová generácia tvorí z novovzniknutých indivíduí a z indivíduí starej generácie výberom indivíduí s najvyššou vhodnosťou.

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 2 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[5].

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 krížencov 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.

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

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

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 metódu vetvenia a medzí. 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.

2.4 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 metódy vetvenia a medzí zastrešená vhodným predikátom s príslušným počtom argumentov.

Metóda vetvenia a medzí 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 algoritmu vetvenia a medzí 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 metóda vetvenia a medzí.

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 metóda vetvenia a medzí.

Vygenerovať počiatočné riešenie (t.j. stanoviť ***hornú hranicu riešenia***) je možné pomerne jednoducho 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)

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).

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. **Metóda vetvenia a medzí** 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. V princípe existujú dve stratégie (sú implementované napr. V CLP jazykoch *CHIP* a *ELCiPSe*) [MudPre 95]:

MINMAX[6] 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[7] 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)[8], 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.

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

POROVNANIE METÓD NA RIEŠENIE ÚLOH ROZVRHOVANIA

3.1 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, metóda vetvenia a medzí, 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.

3.2 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), metóda vetvenia a medzí, 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.

Metódu vetvenia a medzí 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, vetvenia a medzí 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, metódu vetvenia a medzí, 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*
Metóda vetvenia a medzí	• *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*	• *vyžaduje funkciu susednosti, ktorá je rozhodujúca pre efektívnosť*
Simulované žíhanie	• *hill climbing môže uviaznuť v lokálnych optimách*	• *funkcia susednosti rozhoduje o efektívnosti* • *plán ochladzovania je kritický*
tabu search	• *simulované žíhanie a tabu search sa z nich snažia vyviaznuť*	• *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.

3.3 Zaradenie CLP do systému metód

Keď sa pozrieme na CLP z pohľadu metód popísaných v prehľadovej časti dizertačnej práce (časť 1.2) a ich členenia v časti 3.1, 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), metóda vetvenia a medzí (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.