ŽILINSKÁ UNIVERZITA V ŽILINE
FAKULTA RIADENIA A INFORMATIKY

Písomná práca na dizertačnú skúšku

METASTRATÉGIE PRI RIEŠENÍ PROBLÉMU JOB-SHOP

na získanie vedecko-akademickej hodnosti doktor filozofie v odbore

37-01-9 Dopravná a spojová technológia

Ing. Peter Mičunek

Školiteľ: Doc. RNDr. Stanislav Palúch CSc.
Miesto a dátum: Bratislava, november 2002

OBSAH

KAPITOLA 1

ÚVOD DO PROBLEMATIKY

1.1 Úvod

V praktickom živote sa často stretávame s úlohami vykonať určité činnosti, na určitom mieste, v danom čase. Problémy tohto typu sa snaží riešiť teória rozvrhov, alebo tiež teória rozvrhovania. Vo všeobecnosti sa rozvrhovanie zaoberá problémami, v ktorých sa snažíme optimálne rozmiestniť dané činnosti na dostupné zariadenia, pri dodržaní určitých obmedzujúcich podmienok.

Rozvrhom je napríklad rozdelenie výrobných úloh v podniku na jednotlivé stroje, tvorba školského rozvrhu, rozvrhovanie úloh v jadre operačného systému, určenie ktoré z lietadiel žiadajúcich o pristátie v ktorom čase nechať pristať na ktorú pristávaciu dráhu a ktorú dráhu letiskovej budovy mu prideliť, ktorý spoj realizovať ktorým autobusom, atď. Dôležitou podtriedou rozvrhovacích problémov sú *dopravné rozvrhy*, ktorými sa zaoberajú napríklad diela Černého [14], Černého a Kluvánka [46] a Peška [62].

Rozvrhovacie úlohy chceme vyriešiť tak, aby boli z daného hľadiska optimálne. Na to je nevyhnutné, aby sme mali k dispozícii funkciu, ktorá číselne ohodnotí kvalitu každého možného riešenia. Túto funkciu budeme nazývať *kritérium optimality*. Na základe tejto funkcie môžeme matematickými prostriedkami hľadať optimálne riešenie, t.j. riešenie, ktoré túto funkciu minimalizuje (resp. maximalizuje).

Rozvrhovacie úlohy sa podľa svojich vlastností rozdeľujú do jednotlivých tried. Najznámejšie triedy úloh sú *Job-Shop*, *Flow-Shop* a *Open-Shop*. Tento dokument skúma *Job-Shop* triedu rozvrhovacích úloh, ktorá je považovaná za zvlášť obtiažny kombinatoricko-optimalizačný problém. Exaktné metódy riešenia týchto úloh sú natoľko časovo náročné, že sú prakticky nepoužiteľné. Táto situácia viedla k vzniku a rozvoju heuristických prístupov, ktoré v súčasnej dobe poskytujú veľké spektrum rôznych algoritmov, z ktorých sa v tejto oblasti osvedčili hlavne metaheuristiky *tabu search*, *simulated annealing* a *genetické algoritmy*.

Metaheuristiky sú také heuristické prístupy, ktoré za istých okolností umožňujú dostať sa z lokálneho minima, v ktorom väčšina metód uviazne. Pracujú na diskrétnej množine všetkých riešení danej úlohy a v jednotlivých krokoch prechádzajú od jedného prípustného riešenia k druhému.

V práci sa chcem zamerať na modifikáciu, alebo návrh metaheuristík tak, aby čo najefektívnejšie hľadali optimálne, alebo dostatočne kvalitné prípustné rozvrhy.

1.2 Formulácia problému

Medzi už spomínané základné pojmy teórie rozvrhovania patria operácia, úloha a stroj. Tieto pojmy si teraz bližšie popíšeme a definujeme.

- *Operácia* O_i je základný technologický úkon, ktorý sa spracováva na stroji. Predpokladá sa, že operácia už nie je ďalej deliteľná. Je definovaná svojou dĺžkou (časom potrebným na jej spracovanie) a typom.
- *Úloha* J_i je konečná postupnosť operácií $\{O_{i1}, O_{i2}, \ldots, O_{im_i}\}$[1], ktoré treba vykonať v rámci jednej zákazky. Premenná m_i vyjadruje počet operácií úlohy J_i. Niekedy sa úloha označuje ako *dávka*.
- *Stroj* M_i je zariadenie, ktoré spracováva jednotlivé operácie. Poznáme stroje *univerzálne*, ktoré môžu spracovať ľubovoľnú operáciu, ľubovoľnej úlohy a líšia sa len dobou spracovania operácií a stroje *špecializované*, ktoré sú určené na spracovanie len niektorých operácií. Stroj sa niekedy označuje aj ako *procesor*.

Hneď na začiatku je potrebné upozorniť na rozdiel medzi pojmami poradie spracovania (sequencing) a rozvrh (schedule). Jedno poradie spracovania, ktoré predstavuje usporiadanie operácií na strojoch môže predstavovať nekonečné množstvo rozvrhov. Rozvrh získame určením času začiatku a konca vykonávania všetkých operácií na všetkých strojoch v danom poradí spracovania.

Z hľadiska poradia spracovania operácií rozoznávame v teórii rozvrhovania dve skupiny úloh. Do prvej skupiny patria úlohy, pri ktorých záleží na poradí spracovania jednotlivých operácií a do druhej skupiny, kde nezáleží na poradí spracovania. Z tohto vyplýva, že v úlohách kde záleží na poradí spracovania operácií pri začatí spracovania ľubovoľnej úlohy, musí byť skončené spracovanie všetkých jej predchodcov.

Reláciu, ktorá určuje, či určitá operácia bude spracovaná skôr ako druhá, nazývame *precedencia* a označujeme ju $\prec$. Táto relácia je tranzitívna, t.j.

$$\text{ak} \quad x \prec y \quad \text{a} \quad y \prec z \quad \text{potom} \quad x \prec z.$$

Skutočnosť, že operácia O_i predchádza O_j zapíšeme $O_i \prec O_j$. Ak neexistuje žiadna operácia O_k taká, že $O_i \prec O_k \prec O_j$ tak hovoríme, že operácia O_i bezprostredne predchádza operácii O_j a označíme to $O_i \prec\prec O_j$.

V teórii rozvrhovania platia dve všeobecné podmienky pre operácie, úlohy a stroje:

1. Každý stroj v danom časovom okamžiku môže vykonávať najviac jednu operáciu.
2. V danom časovom okamihu sa nemôžu na strojoch spracovávať dve a viac operácií jednej úlohy.

Problémy v teórii rozvrhovania rozdeľujeme na *statické* a *dynamické*. Statické problémy sa vyznačujú tým, že do prázdneho systému vstupujú všetky úlohy naraz. Práve tieto problémy budú skúmané v tejto práci. V dynamických systémoch úlohy vstupujú do systému priebežne. Okamžiky vstupu jednotlivých úloh do systému sa dajú popísať iba štatisticky, pomocou pravdepodobnostného rozdelenia intervalu medzi jednotlivými vstupmi. Doba spracovania operácie býva tiež náhodná veličina.

[1] Operácie budeme indexovať podľa účelu dvoma spôsobmi. Indexovanie O_i vyjadruje číslo operácie v rámci všetkých operácií. Indexovanie O_{ij} vyjadruje j-tu operáciu i-tej úlohy.

1.3 Vstupné dáta rozvrhovacieho problému

Pre každú úlohu J_i môžu byť špecifikované tieto vstupné dáta:

- operácie O_i, spracovávané na strojoch;
- čas spracovania p_{ij} je doba potrebná na spracovanie j-operácie i-tej úlohy;
- r_i najskôr možný začiatok spracovania úlohy J_i;
- d_i požadovaný čas ukončenia úlohy J_i;
- a_i maximálna prípustná doba pobytu úlohy v systéme $a_i = d_i - r_i$;
- w_i váha úlohy vyjadruje relatívnu dôležitosť úlohy J_i;
- f_i nákladová funkcia je reálna funkcia, ktorá meria náklady $f_i(t)$ vynaložené na to, aby úloha J_i bola ukončená v čase t.

1.4 Výstupné dáta rozvrhovacieho problému

Výstupné dáta rozvrhovacieho problému, sú dáta, ktoré môžeme vypočítať pre každú úlohu J_i daného rozvrhu. Väčšinou sú to tieto dáta:

- C_i čas skončenia úlohy J_i;
- F_i doba pobytu úlohy J_i v systéme;
- W_i celková doba čakania úlohy J_i v systéme;
- L_i časová odchylka od plánovaného času ukončenia úlohy J_i, platí že $L_i = C_i - d_i$;
- $T_i = \max\{0, L_i\}$ oneskorenie úlohy J_i;
- $E_i = \max\{0, -L_i\}$ predstih úlohy J_i;
- $U_i = 0$, ak $C_i \leq d_i$, inak $U_i = 1$ penalizačná jednotka úlohy J_i.

Všetky výstupné dáta rozvrhovacieho problému sú vlastne funkciami základnej veličiny C_i.

1.5 Kritéria optimality rozvrhovacieho problému

Optimálnosť rozvrhovej úlohy môže byť posudzovaná podľa rôznych kritérií. Podľa toho aké kladieme na rozvrh požiadavky, môžeme nám ísť o čo najskorší koniec rozvrhu, alebo o minimalizáciu nákladov spojených s oneskorením úloh a pod. Kritériá vychádzajú z výstupných dát rozvrhovacieho problému. Všetky ďalej uvedené kritériá sa budú minimalizovať.

$$C_{\max} = \max_{\forall i}\{C_i\} \qquad L_{\max} = \max_{\forall i}\{L_i\} \qquad T_{\max} = \max_{\forall i}\{T_i\} \qquad F_{\max} = \max_{\forall i}\{F_i\}$$

Ďalšiu skupinu kritérií tvoria súčty časov ukončenia jednotlivých úloh, časov diferencií, časov oneskorení, časov pobytu v systéme a podobne:

$$\sum C_i = \sum_{i=1}^{n} C_i \qquad \sum L_i = \sum_{i=1}^{n} L_i \qquad \sum T_i = \sum_{i=1}^{n} T_i \qquad \sum U_i = \sum_{i=1}^{n} U_i$$

V prípade nerovnakého významu jednotlivých úloh môžeme použiť ako kritériá vážené sumy predchádzajúcich veličín.

$$\sum w_i C_i = \sum_{i=1}^{n} w_i C_i \qquad \sum w_i L_i = \sum_{i=1}^{n} w_i L_i$$

$$\sum w_i T_i = \sum_{i=1}^{n} w_i T_i \qquad \sum w_i U_i = \sum_{i=1}^{n} U_i$$

V niektorých úlohách sa ako potrebné kritériá objavujú priemerné hodnoty časov ukončenia úloh, ich časovej diferencie, omeškania a pobytu v systéme.

$$\overline{C} = \frac{1}{n}\sum_{i=1}^{n} C_i \qquad \overline{L} = \frac{1}{n}\sum_{i=1}^{n} L_i \qquad \overline{T} = \frac{1}{n}\sum_{i=1}^{n} T_i \qquad \overline{U} = \frac{1}{n}\sum_{i=1}^{n} U_i$$

Všetky funkcie používané ako kritériá optimality boli v konečnom dôsledku funkciami časových okamžikov ukončenia $C_1, C_2, \ldots, C_n$. Teória rozvrhovania pracuje aj zo zložitejšími kritériami tvaru $\mathbf{M} = f(C_1, C_2, \ldots, C_n)$. Medzi nimi zaujímajú zvláštne postavenie *regulárne kritériá*. Kritérium $\mathbf{Z} = f(C_1, \ldots, C_n)$ nazveme regulárnym, keď je minimalizačné, a keď je neklesajúcou funkciou každej svojej premennej $C_1, \ldots, C_n$. Kritéria $C_{\max}$, $L_{\max}$, $T_{\max}$, $\overline{C}$, $\overline{W}$, $\overline{L}$, $\overline{T}$ patria do triedy regulárnych kritérií.

1.6 Matematický model Job-Shopov

Job-Shop rozvrhovací problém v nasledujúcom texte tiež označovaný ako JSSP (Job-Shop Scheduling Problem) je klasický problém operačnej analýzy s veľkým počtom aplikácií, ale s malým počtom praktických metód riešenia. Môžeme ho považovať za problém alokácie operácií s obmedzeným počtom zdrojov, obmedzujúcimi podmienkami a optimalizačným kritériom. Job-Shop môžeme formulovať ako problém spracovania konečnej množiny úloh $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ na konečnej množine strojov $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$. Budeme predpokladať, že každá úloha $J_i \in \mathcal{J}$ má rovnaký počet operácií $J_i = \{O_{i1}, \ldots, O_{im}\}$.

Každá úloha J_i musí byť spracovaná všetkými strojmi a keďže počet operácií v každej úlohe je rovný počtu strojov, každá operácia v úlohe je spracovaná práve na jednom stroji z množiny $\mathcal{M}$. Operácie každej úlohy J_i musia byť rozvrhnuté v zadanom, predefinovanom poradí a teda platí

$$O_{i1} \prec O_{i2} \prec \ldots \prec O_{im}$$

Táto požiadavka sa nazýva *podmienka precedencie*. Platí že, každá úloha $J_i = \{O_{i1}, \ldots, O_{im}\}$ je usporiadanou podmnožinou množiny operácií $\mathcal{O}$, kde:

$$\mathcal{O} = J_1 \cup \ldots \cup J_n \;\text{ a zároveň }\; \mathcal{O} = M_1 \cup \ldots \cup M_m.$$

Ďalej počet všetkých operácií $\mathcal{O} = \{O_1, \ldots, O_N\}$ je $N = \sum_{i=1}^{n} m$. V ľubovoľnom časovom okamihu môže každý stroj vykonávať iba jednu úlohu a každá úloha môže byť spracovávaná

iba na jednom stroji. Operácia $O_i \in \mathcal{O}$ sa vykoná bez prerušenia v predefinovanom poradí na určenom stroji $M_j \in \mathcal{M}$ za čas p_i. Ďalej nech π_k je usporiadanie operácií na stroji $\mathcal{M}_k$ a nech Π_k je množina všetkých možných usporiadaní operácií na stroji $\mathcal{M}_k$. Potom poradie spracovania na všetkých strojoch je definované ako $\pi = \{\pi_1, \pi_2, \ldots, \pi_m\}$, kde $\pi \in \Pi = \Pi_1 \times \Pi_2 \times \ldots \Pi_m$.

Ak celkový čas spracovania úlohy J_i v rozvrhu je C_i, tak potom čas najdlhšie spracovávanej úlohy v systéme sa nazýva makespan $C_{\max}$. Našou úlohou je určiť poradie spracovania π, ktoré bude minimalizovať optimalizačné kritérium $C_{\max}$ tak, aby priradenie operácií k strojom bolo prípustné t.j. aby boli splnené precedenčné aj kapacitné podmienky.

$$C^{\star}_{\max} = \min(C_{\max})$$

Každú operáciu O_{ij} môžeme tiež charakterizovať usporiadanou trojicou (i, j, k), ktorá vyjadruje skutočnosť, že j-ta operácia i-tej úlohy J_i vyžaduje k-ty stroj. Ďalej bude pre Job-Shop platiť, že všetky úlohy sa budú môcť začať vykonávať v čase nula, teda $r_i = 0$ a žiadna z úloh nebude mať určený požadovaný čas ukončenia spracovania. Pre všetky úlohy bude platiť, že ich váha bude rovnaká, teda $w_i = 1$.

1.7 Klasifikácia rozvrhovacích problémov

Tak ako všetky ostatné rozvrhovacie problémy, tak sa aj JSSP klasifikuje podľa stroja, operácie, precedencie a ostatných základných charakteristík. Na základe týchto charakteristík navrhol Conway a kol. [18] klasifikáciu, ktorá sa skladá zo štyroch políčok $A|B|C|D$. Táto schéma však nepopisovala dostatočne dobre problémy, ktoré obsahovali podmienky navyše. Preto Graham a kol. [38] navrhli trojpoličkovú schému $\alpha|\beta|\gamma$, ktorú neskôr Vaessens [77] jemne upravil. V nasledujúcom texte si tieto klasifikácie popíšeme.

1.7.1 Conway-Maxwell-Millerova klasifikácia

Conway zaviedol pre klasifikáciu rozvrhovacích problémov štvormiestnu schému $A|B|C|D$, kde:

- A charakterizuje proces vstupu požiadaviek do systému. Pre statické systémy toto číslo vyjadruje počet úloh, ktoré vstupujú naraz do systému. Keď je v políčku A uvedené n znamená to, že v systéme je ľubovoľný konečný počet úloh.
- B vyjadruje počet strojov v systéme. Keď je v políčku B uvedené m znamená to, že v systéme môže byť ľubovoľný konečný počet strojov.
- C charakterizuje, v akom prostredí sa stroj nachádza. $C \in \{\star, F, P, J, O, D\}$, kde:
  - $\star$ je systém s jedným strojom
  - F je *Flow-Shop* systém, kde každá úloha $\{J_i\}_{i=1}^{n}$ obsahuje m operácií a každá operácia O_{ij} má byť vykonaná na stroji μ_{ij} za p_{ij} časových jednotiek, pričom všetky úlohy prechádzajú strojmi v rovnakom určenom poradí.
  - P predstavuje permutačný Flow-shop,

J je *Job-Shop* systém, kde každá úloha $\{J_i\}_{i=1}^n$ obsahuje m operácií. Operácie prechádzajú strojmi v rôznom poradí. Každá operácia O_{ij} má byť vykonaná na pridelenom stroji μ_{ij} za p_{ij} časových jednotiek, pričom $\mu_{i(j-1)} \neq \mu_{ij}$ pre $j = 2, \ldots, m_i$. Teda Flow-Shop je v podstate špeciálnym prípadom Job-Shop problému.

O je značka pre skupinu úloh, označovaných ako *Open-Shop*. V tejto triede úloh sa každá úloha $\{J_i\}_{i=1}^n$ skladá z m operácií a operácia O_{ij} sa má vykonať na stroji μ_{ij} za p_{ij} časových jednotiek, pričom na poradí operácií nezáleží.

D je *Dag-Shop*, alebo *Mixed-Shop* čo je zovšeobecnenie Job-Shop problému v tom zmysle, že poradie, ktorým úlohy prechádzajú strojmi, môže byť určené rôzne. Môže to byť napr. rôzne poradie ako Job-Shop, alebo rovnaké poradie ako Flow-Shop.

D charakterizuje kriteriálnu funkciu.

1.7.2 MacCarthy-Liuova modifikácia

MacCarthy a Liu [53] rozšírili políčko C predchádzajúcej klasifikácie o popis prostredia v ktorom stroj pracuje a o vlastnosti úlohy. $C \subset \{k\text{-}par, str, prec, prmt, unit, eq, depend, setup\}$, kde

- k-par: systém obsahuje k paralelných strojov
- str: reťazec úloh
- $prmt$: je dovolené prerušenie spracovania operácie
- eq: pre všetky úlohy je určený rovnaký čas spracovania
- $setup$: sekvenčne závislý počiatočný čas
- r_j: úlohy s rôznymi začiatočnými časmi spracovania
- $prec$: precedenčné podmienky
- $unit$: jednotkový čas spracovania úloh
- $depend$: v systéme je závislosť úloh

1.7.3 Graham-Lawler-Lenstra-Rinnooy Kanova klasifikácia

Táto klasifikácia je určená len pre klasifikáciu statických problémov, pre ktoré platí, že jednotlivé úlohy vchádzajú do prázdneho systému po dávkach $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$. V tejto klasifikácie sa rozvrhovacie problémy popisujú trojmiestnou schémou $\alpha|\beta|\gamma$[2].

$\boxed{\alpha}$ je trojmiestna položka $\alpha = \alpha_1\alpha_{1a}\alpha_2$, ktorá charakterizuje množinu strojov $\mathcal{M}$. Parameter α_1 popisuje vlastnosti množiny strojov, α_{1a} je dodatočná charakteristika prostredia a parameter α_2 popisuje počet strojov. Parameter α_1 môže nadobúdať tieto hodnoty $\alpha_1 = \{\star, P, Q, R, O, F, J\}$ kde:

[2]Dnes sa táto klasifikácia používa najčastejšie a postupne sa do nej pridávajú ďalšie a ďalšie charakteristiky.

$\star$ v systéme je jeden stroj

P v systéme je m identických paralelných strojov.

Q v systéme je m uniformných pralelných strojov. Uniformita znamená, že všetky stroje majú rovnaké vlastnosti okrem rýchlosti. Potom sa doba spracovania operácie vypočíta ako $p_{ij} = p_i.q_j$. Kde číslo q_j je tzv. faktor rýchlosti stroja M_j.

R v systéme je m ľubovoľných paralelných strojov.

Uvedené hodnoty parametra α_1 sa používajú pre úlohy, ktoré obsahujú iba jednu operáciu, ktorá môže byť vykonaná na ľubovoľnom stroji. Nasledujúce hodnoty parametra popisujú situácie, keď každá z úloh $\{J_i\}_{i=1}^n$ obsahuje viac operácií.

O označuje *Open-Shop* systémy.

F označuje *Flow-Shop* systémy.

J označuje *Job-Shop* systémy.

G označuje všeobecný *General-Shop* systém (relácia precedencie medzi ľubovoľnými operáciami).

C označuje *Cycle-Shop* systém (podobá sa na F, ale stroj môže byť použitý úlohami viac než raz).

C označuje *Mixed-Shop* systém, kombináciu O a J.

Parameter α_{1a} môže nadobúdať tieto hodnoty $\alpha_{1a} = \{\varnothing, MPM, MPT\}$ kde:

$\varnothing$ nie sú žiadne dodatočné špecifikácie prostredia.

MPM viacúčelové stroje (operácia sa môže vykonať na danej podmnožine strojov).

MPT viacstrojové operácie (operácia používa viac než jeden procesor naraz).

$\boxed{\beta}$ je druhá položka v schéme. $\beta = \beta_1\beta_2\beta_3\ldots$ popisuje charakteristiky úlohy, ktoré sú definované nasledovne:

β_1 charakterizuje možnosť prerušenia operácie. Keď sa $\beta_1 = pmtn$ je dovolené prerušenie. To znamená, že ľubovoľná operácia môže byť prerušená a neskôr obnovená. Keď sa $\beta_1 = \star$ znamená to zákaz prerušenia operácie.

β_2 vyjadruje obmedzenú existenciu zdrojov (resources). $\beta_2 = res$ predpokladá prítomnosť S obmedzených zdrojov $R_1, R_2, \ldots, R_s$ s takými vlastnosťami, že každá úloha J_i vyžaduje počas svojho vykonávania použitie r_{hi} jednotiek zo zdroja R_h. Predpokladá sa, že v danom čase sa zo žiadneho zdroja nemôže využiť viac ako 100%. $\beta_2 = res1$ predpokladá prítomnosť len jedného obmedzeného zdroja. $\beta_2 = \star$ nepredpokladá existenciu obmedzených zdrojov.

β_3 vyjadruje precedenčnú reláciu na množine úloh $\mathcal{J}$. $\beta_3 = prec$ vyjadruje existenciu relácie precedencie na množine úloh $\mathcal{J}$. K úlohe je špecifikovaný aj príslušný digraf bezprostrednej precedencie $\mathbb{G}_{\prec\prec}$. Ak $\mathbb{G}_{\prec\prec}$ obsahuje orientovanú cestu z J_i do J_k, znamená to, že $J_i \prec J_k$, a vtedy sa požaduje, aby úloha J_i bola úplne skončená skôr ako začne vykonávať úloha J_k. $\beta_3 = tree$ $\mathbb{G}_{\prec\prec}$ je koreňový strom, v ktorom má každý vrchol maximálne jedného predka, alebo nasledovníka. Pre $\beta_3 = \star$ nie je definovaná precedencia.

β_4 vyjadruje vstupné časy $r_1, r_2, \ldots, r_n$ úloh $J_1, J_2, \ldots, J_n$ do systému. $\beta = r_i$ sú špecifikované vstupné časy, ktoré môžu byť pre každú úlohu odlišné. Pri $\beta_4 = \star$ predpokladáme, že $r_1 = r_2 = \ldots = r_n = 0$

β_5 charakterizuje množinu úloh $\mathcal{J}$ z hľadiska počtu operácií jednotlivých úloh. $\beta_5 = g_i \leq g$ znamená, že je špecifikovaná horná hranica pre počty operácií g_i. β_5 vyjadruje, že nie je špecifikovaná horná hranica počtu operácií, úloh vchádzajúcich do systému.

β_6 charakterizuje doby trvania operácií úloh vchádzajúcich do systému. $\beta_6 = p_{ij} = 1$ znamená, že každá operácia má jednotkovú dobu spracovania. $p_{ij} \leq p$ vyjadruje dolnú a hornú hranicu pre p_{ij}. $\beta_6 = \star$ vyjadruje, že operácia nie je časovo obmedzená.

$\boxed{\gamma}$ charakterizuje kriteriálnu funkciu.

Príklady:

$1||C_{max}$ systém s jedným strojom, minimalizujeme čas výstupu poslednej úlohy zo systému.

$J_m|prec|\sum F_i$ Job-Shop s m strojmi. Medzi úlohami existuje precedencia a minimalizuje sa celková doba pobytu úloh v systéme.

$F2||C_{\max}$ Flow-Shop s dvoma strojmi s minimalizáciou času výstupu poslednej úlohy zo systému. Každá úloha pozostáva z dvoch operácií.

1.8 Reprezentačné modely

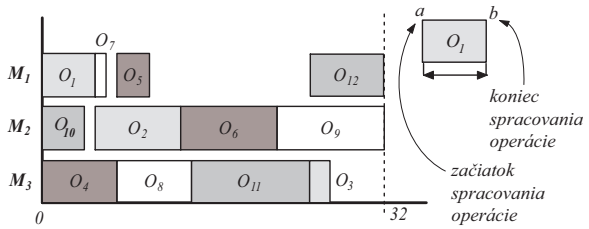
Často je výhodné pre lepšiu názornosť, alebo z iných dôvodov rozvrh znázorniť graficky. Najpoužívanejšie sú dva spôsoby grafickej reprezentácie Job-Shopov, *Ganttov diagram* [30] a *disjunktívny graf* [65].

1.8.1 Ganttov diagram

Najjednoduchšou a jednou z najpoužívanejších metód na zobrazenie rozvrhu je Ganttov diagram [30]. Tento diagram popisuje vzťahy medzi úlohami a strojmi v čase. Na x-ovej osi sa znázorňuje čas a na y-ovej osi sú znázornené jednotlivé stroje. Každá operácia je reprezentovaná štvorčekom, ktorého dĺžka predstavuje čas spracovania operácie. Štvorčeky patriace rovnakým úlohám sú nakreslené rovnakým spôsobom a štvorčeky patriace rôznym úlohám sa od seba odlišujú. Na príklade si ukážeme viac. Tabuľka 1.1 obsahuje typické dáta pre rozvrhovací problém a obrázok 1.1 predstavuje Ganttov diagram vytvorený s tychto dát.

	J_1			J_2			J_3			J_4		
O_i	O_1	O_2	O_3	O_4	O_5	O_6	O_7	O_8	O_9	O_{10}	O_{11}	O_{12}
M_i	1	2	3	3	1	2	1	3	2	2	3	1
p_i	5	8	2	7	3	9	1	7	10	4	11	7

Tabuľka 1.1: Zadanie JSSP s rozmerom 4×3



Obr. 1.1: Príklad Ganttovho diagramu

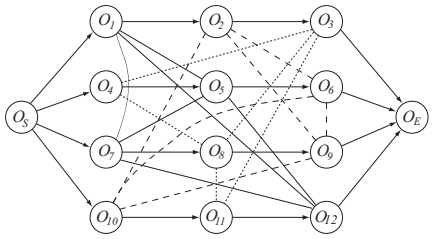
1.8.2 Grafový model

I napriek tomu, že Ganttov diagram je veľmi populárny a často používaný na zobrazovanie rozvrhov, používa sa aj iná metóda zobrazovania tzv. *disjunktívny* graf.

Disjunktívny graf je orientovaný graf $\mathbb{G} = (V, C \cup D)$. Kde V predstavuje množinu vrcholov korešpondujúcu s operáciami. Táto množina obsahuje dva prídavné vrcholy: zdrojový a cieľový, ktoré reprezentujú začiatok a koniec rozvrhu. Zdrojový vrchol sa označuje O_S a predchádza všetky operácie a cieľový vrchol sa označuje O_E a je následníkom všetkých ostatných operácií. Oba pridané vrcholy (operácie) majú nulový čas spracovania p_i. C je množina konjunktívnych hrán, ktoré vyjadrujú počiatočnú podmienku precedencie medzi každými dvoma operáciami v danej úlohe. To znamená, že medzi dvoma operáciami bude táto hrana $[(i,j,k),(i',j',k')] \in C$ práve vtedy, keď $i' = i+1$ a $j = j'$. D predstavuje neorientované disjunktívne hrany, ktoré spájajú neutriedené operácie, ktoré vyžaduje rovnaký stroj na spracovanie. To znamená, že medzi dvoma operáciami je *neorientovaná* hrana $((i,j,k),(i',j',k')) \in D$ práve vtedy keď, $k = k'$. Každá hrana je ohodnotená kladným číslom, ktoré predstavuje čas spracovania tej operácie v ktorej hrana začína.

Disjunktívny graf pre problém popísaný v tabuľke 1.1 predstavuje graf na obrázku 1.2. Konjunktívne hrany sú reprezentované plnými orientovanými čiarami disjunktívne hrany prerušovanými neorientovanými. Rôzne typy prerušovaných čiar predstavujú rôzne stroje, na ktorých sa operácie spracovávajú.

Nájdenie optimálneho rozvrhu pre systém $J_m||\ f$ potom spočíva v určení orientácie na



Obr. 1.2: Disjunktívny graf reprezentujúci zadanie s tabuľky 1.1

neorientovaných hranách migrafu $\mathbb{G}$ takým spôsobom, aby vzniknutý graf $\mathbb{G}$ bol acyklický a zároveň minimalizoval kriteriálnu funkciu f.

1.9 Model celočíselného lineárneho programovania

Nech je $\mathcal{O}_0$ množina indexov všetkých operácií bez predchodcov, $\mathcal{O}_T$ množina indexov všetkých operácií bez následníkov, x_i čas ukončenia operácie O_i a p_i doba jej spracovania.

Skutočnosť, že rozvrhovanie začína v čase 0, vyjadrujú podmienky

$$x_i - p_i \geq 0 \qquad \forall i \in \mathcal{O}_0$$

Označme $I_{\mathcal{J}}$ množinu všetkých usporiadaných dvojíc indexov (i,j) takých, že O_i, O_j sú operácie tej istej úlohy a $O_i \prec\prec O_j$. Ďalej označíme $I_{\mathcal{M}}$ množinu všetkých usporiadaných dvojíc indexov (i,j) takých, že $i < j$ a operácie O_i, O_j vyžadujú na spracovanie ten istý stroj.

Bezprostrednú precedenciu operácií v rámci úloh vyjadruje sústava obmedzení:

$$x_j - p_j \geq x_i \qquad \forall (i,j) \in I_{\mathcal{J}}$$

Tieto ohraničenia zodpovedajú orientovaným hranám v grafe $\mathbb{G}$. Teraz treba ešte obmedzenia a premenné, ktoré určia orientáciu pre neorientované hrany v $\mathbb{G}$. Pre každú dvojicu $(i,j) \in I_{\mathcal{M}}$ označme y_{ij} binárnu premennú, ktorá je rovná 1 práve vtedy, keď operácia O_i predchádza O_j na spoločnom stroji M_k. Pre dve operácie O_i, O_j musí platiť jedna z nerovností

$$\left.\begin{array}{ll} x_j - x_i \geq p_j & \text{ak } y_{ij} = 1 \\ x_i - x_j \geq p_i & \text{ak } y_{ij} = 0 \end{array}\right\} \qquad \forall (i,j) \in I_{\mathcal{M}}$$

Potom pre veľmi veľké číslo H musia platiť obe nerovnosti

$$\left.\begin{array}{l} x_j - x_i + H(1 - y_{ij}) \geq p_j \\ x_i - x_j + H y_{ij} \geq p_i \end{array}\right\} \qquad \forall (i,j) \in I_{\mathcal{M}}$$

Pretože operácie každej úlohy sú lineárne usporiadané, existuje v $\mathcal{O}_T$ pre každú úlohu práve jedna terminálna operácia. Preto množina $\{x_i |\ i \in \mathcal{O}_T\}$ je množinou časových okamžikov ukončení všetkých úloh z $\mathcal{J}$.

Nech C predstavuje čas ukončenia poslednej úlohy z dávky $\mathcal{J}$. Potom pre C platí

$$C \geq x_i \qquad \forall i \in \mathcal{O}_T$$

a pre jeho minimalizáciu stačí zvoliť kriteriálnu funkciu C. Model pre minimalizáciu kritéria $C_{\max}$ potom vyzerá následovne:

$$\begin{array}{lll} \min & C & \\ \text{st:} & C \geq x_i & \forall i \in \mathcal{O}_T \qquad (1) \\ & x_i - p_i \geq 0 & \forall i \in \mathcal{O}_0 \qquad (2) \\ & x_j - p_j \geq x_i & \forall (i,j) \in I_{\mathcal{J}} \qquad (3) \end{array}$$

$$\left.\begin{array}{l} x_j - x_i + H.(1 - y_{ij}) \geq p_j \\ x_i - x_j + H.y_{ij} \geq p_i \\ y_{ij} \in \{0,1\} \end{array}\right\} \qquad \forall (i,j) \in I_{\mathcal{M}} \qquad (4)$$

Pre minimalizáciu kritéria $\sum C_i$ máme tento model:

$$\begin{array}{lll} \min & C & \\ \text{st:} & x_i - p_i \geq 0 & \forall i \in \mathcal{O}_0 \qquad (2) \\ & x_j - p_j \geq x_i & \forall (i,j) \in I_{\mathcal{J}} \qquad (3) \end{array}$$

$$\left.\begin{array}{l} x_j - x_i + H(1 - y_{ij}) \geq p_j \\ x_i - x_j + H y_{ij} \geq p_i \\ y_{ij} \in \{0,1\} \end{array}\right\} \qquad \forall (i,j) \in I_{\mathcal{M}} \qquad (4)$$

Greenberg navrhol nasledujúci spôsob riešenia. V prvom kroku sa vypustia podmienky (4) a dostaneme úlohu lineárneho programovania. Ak je výsledné riešenie prípustné, teda také, že žiadne dve úlohy nevyžadujú ten istý stroj, v tom istom čase, dostali sme optimálne riešenie. Ak nastal problém pre operácie o_i a o_j, riešime dva podproblémy, jeden s dodatočnou podmienkou

$$x_j - x_i \geq p_j$$

a druhý s dodatočnou podmienkou

$$x_i - x_j \geq p_i$$

Riešením každého z podproblémov dostaneme dolnú hranicu kriteriálnej funkcie. Ak je niektorá z dolných hraníc väčšia než doterajšie rekordné minimum, v tejto vetve už nebudeme pokračovať. Ak je niektoré riešenie z podproblémov prípustné, porovnáme hodnotu jeho kriteriálnej funkcie s doterajším rekordom. Ak došlo k zlepšeniu, aktualizujem rekord. Takýmto spôsobom skúmam všetky perspektívne vetvy stromu riešení.

1.10 Typy rozvrhov

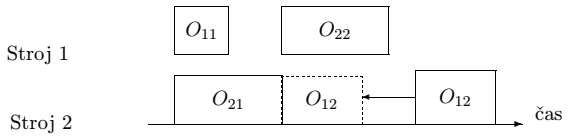
Podľa toho ako sú jednotlivé operácie rozvrhnuté poznáme rozvrhy aktívne, semi-aktívne a bez prestojov.

Lokálny ľavý posun v rozvrhu je také časové posunutie začiatku spracovania niektorej operácie dopredu (na skoršiu dobu), pri ktorom sa neporuší prípustnosť rozvrhu a zachová sa poradie spracovania operácií na každom stroji (obrázok 1.3).

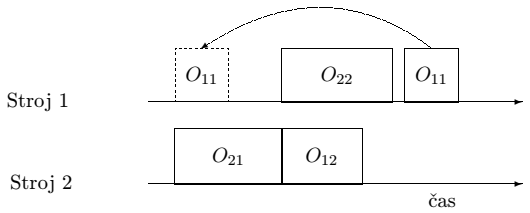
Globálny ľavý posun je taký presun niektorej operácie dopredu, pri ktorom sa zachová prípustnosť rozvrhu bez zmeny časovej polohy ostatných operácií (obrázok 1.4).

Semiaktívny rozvrh je rozvrh, v ktorom neexistuje lokálny ľavý posun.

Aktívny rozvrh je rozvrh, v ktorom neexistuje globálny ľavý posun.



Obr. 1.3: Lokálny ľavý posun



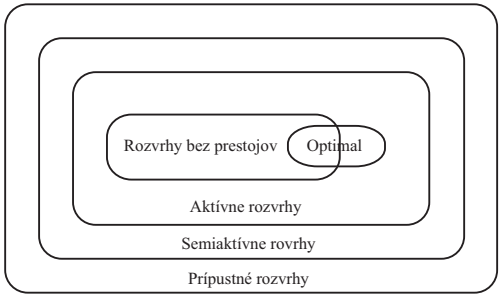
Obr. 1.4: Globálny ľavý posun

Zbytočný (umelý) prestoj je časový interval (t_1, t_2), v ktorom sa na niektorom stroji M_j nevykonáva žiadna operácia napriek tomu, že je v systéme operácia, ktorá by sa mohla začať vykonávať v čase t_1.

Rozvrh bez prestojov je taký, v ktorom neexistuje zbytočný prestoj. Platí, že ak existuje lokálny ľavý posun, tak existuje zbytočný prestoj. Treba si tiež uvedomiť, že rozvrh bez zbytočných prestojov nemusí zaručovať rozvrh s optimom.

Na obrázku 1.5 je pomocou množinového diagramu znázornený vzťah medzi jednotlivými typmi rozvrhov.

Hovoríme, že $\mathcal{S}$ je dominantná množina rozvrhov pre kritérium f, ak v $\mathcal{S}$ existuje aspoň jeden optimálny rozvrh z hľadiska kritéria f. Platí, že množina aktívnych (semiaktívnych) rozvrhov je dominantná pre $Jm||\ f$ a f je regulárne kritérium.



Obr. 1.5: Typy rozvrhov

KAPITOLA 2

SÚČASNÝ STAV RIEŠENEJ PROBLEMATIKY

2.1 Úvodné poznámky

V súčasnosti sa na riešenie JSSP využíva celá škála kombinatoricko-optimalizačných metód, heuristík a metaheuristík. Všetky ich môžeme rozdeliť do dvoch skupín na *exaktné* metódy a na *aproximačné* (približné). Aproximačné metódy nám nezaručia, že výsledok ktorý pomocou nich získame bude optimálny. Získame ho však rýchlo a to, že je blízko optima (môže byť aj optimálny) nám vyhovuje. Naproti tomu exaktné metódy sa pri takýchto výpočtovo náročných problémoch ako je JSSP, blížia k optimálnemu riešeniu pomaly, ale isto. Problém je ale v tom, že im to trvá tak dlho, že sú prakticky nepoužiteľné.

V súčasnosti existujú exaktné metódy pre rozmermi malé rozvrhovacie problémy. V praxi sa však vyskytujú práve tie opačné, veľké. Medzi exaktné metódy, ktoré sa použili pri riešení JSSP patria metódy vetiev a hraníc, metódy matematického programovania (lineárne, celočíselné a zmiešané programovanie), dekompozičné techniky atď.

Naproti tomu sa aproximačné metódy používajú vo veľmi širokej miere. Môžeme ich rozdeliť na *konštrukčné* metódy, ktoré výsledok skonštruujú podľa nejakého postupu a na *iteratívne* metódy, ktoré systematicky prehľadávajú svoje okolie. Medzi aproximačné metódy patria *tabu search*, *simulované chladenie*, *genetické algoritmy*, *bottleneck heuristiky* a iné. Prehľad metód použitých na riešenie Job-Shop rozvrhovacieho problému je na obrázku 2.1 na konci tejto kapitoly.

2.2 Dispečérske prioritné pravidlá (pdrs)

Technika dispečérskych prioritných pravidiel[1] bola jedna z prvých techník, ktorá sa použila na riešenie JSSP problému. Jej výhody spočívajú v ľahkej implementácii a podstatnej redukcii výpočtov. Princíp metódy spočíva v tom, že v každom kroku pridelíme nerozvrhnutým operáciam prioritu a operácia s najväčšiou prioritou sa pridá do rozvrhu.

Prvé práce o pdrs napísali Jackson [40], Smith [67], Rowe a Jackson [64], Giffler a Thompson [34] a Gere [32]. Jedna z najznámejších prác porovnávajúca rozvrhovacie heuristiky prezentuje až 113 pdrs heuristik. Príklad najpopulárnejších pdrs popisuje tabuľka 2.1.

[1] Názvy dispečérske prioritné pravidlo, dispečérske pravidlo, rozvrhovacie pravidlo, alebo sekvenčné pravidlo sa často používajú ako synonymá.

pdrs	pravidlo priority
SPT	najkratší čas spracovania operácie
SPT/TWKR	najmenší pomer SPT a celkového času do skončenia úlohy
LRM	najdlhší čas do skončenia úlohy okrem danej operácie
MWKR	najväčší čas do skončenia úlohy
MOPR	najväčší počet ešte nespracovaných operácií v úlohe
SPT/TWK	najmenší pomer času spracovania operácie k času spracovania úlohy
FCFS (FIFO)	ktorá prvá príde, prvá bude obslúžená
LSO	najdlhšia nasledujúca operácia
Fhalf	prvá polovica preferovaná
NINQ	ďalšia operácia na stroji bude tá, ktorá ma musí najmenej čakať na spracovanie

Tabuľka 2.1: Niektoré pravidlá priority pre pdrs

Dispečérske prioritné pravidlá môžeme rozdeliť do troch tried. *Prvá trieda* obsahuje jednoduché prioritné pravidlá, ktoré sú založené na informáciach o úlohách. Sú to napríklad pravidlá SPT, MWKR, NINQ s tabuľky 2.1.

Druhá trieda je kombináciou dispečérskych pravidiel s prvej triedy. Tieto pdrs môžu závisieť od situácie, ktorá nastane počas riešenia. Typickým príkladom môže byť pravidlo použitia SPT, pokiaľ dĺžka fronty nie je väčšia ako konštanta c, ak je zmení sa pravidlo na FIFO.

Tretia trieda obsahuje pravidlá označované ako *prioritné pravidlá s váhami*. Základnou myšlienkou je použiť viac než jednu informáciu o úlohe pri zostavovaní rozvrhu. Jednotlivým informáciám sú priradené váhy, ktoré odrážajú ich dôležitosť. Pravidlo priority potom môže vyzerať napríklad takto $priorita = w_1.\text{MWKR} + w_2.\text{MOPR}$.

Význam prioritných dispečérskych pravidiel spočíva hlavne v ich využití v niektorých metaheuristikách na nájdenie počiatočného riešenia, ktoré sa potom ďalej zlepšuje. Prioritné dispečérske pravidlá už dnes nedávajú dostatočne kvalitné výsledky, na to aby sa mohli použiť samostatne.

2.3 Bottleneck heuristiky (SBP)

Heuristika založená na bottlenecku *Shifting bottleneck procedúra* (SBP) bola prvá, ktorá dokázala vyriešiť problém FT10[2]. SBP je charakteristická nasledujúcimi činnosťami: Identifikáciou podproblému, označením Bottlenecku (úzkeho hrdla), riešením podproblému, reoptimalizáciou rozvrhu.

V princípe je to metóda využívajúca tri samostatné algoritmy na vyriešenie problému $P_\infty|\ prec\ |C_{\max}$, problému $1|\ r_i\ |L_{\max}$ a problému $1|\ ri, preemp\ |L_{\max}$. Táto heuristika postupuje nasledovne:

Nech $\mathcal{M}$ je množina všetkých strojov. V každej iterácii heuristiky predstavuje množina $\mathcal{M}_0$ množinu strojov, pre ktoré už je určená postupnosť operácii (t.j. bola určená orientácia disjunktívnych hrán pre tieto stroje). Výsledkom iterácie je určenie kritického stroja $M \in \mathcal{M} - \mathcal{M}_0$, ktorý sa priradí k množine $\mathcal{M}_0$ a určenie poradia spracovania operácií na tomto stroji. Na to, aby sa tento stroj určil, sa najprv odstránia z migrafu všetky neorientované hrany. V grafe zostanú len hrany zodpovedajúce bezprostrednej precedencii operácií

[2]FT10 Fisher-Thompsonov benchmarkový príklad s desiatimi úlohami a desiatimi strojmi. Pozri ďalej.

na úlohách a hrany reprezentujúce bezprostrednú precedenciu operácií spracovávaných na strojoch z množiny $\mathcal{M}_0$. Výsledkom je graf, ktorý zobrazuje situáciu, kedy je možné paralelne spracovať všetky operácie vykonávané na strojoch z množiny $\mathcal{M} - \mathcal{M}_0$. To predstavuje systém $P_\infty|\ prec\ |C_{\max}$, ktorý sa dá riešiť metódou časového plánovania. Jej výsledkom je časové okno $\langle r_i, d_i)$ pre každú operáciu O_i ,v ktorom sa má vykonať, aby bola dodržaná dĺžka rozvrhu $C_{\max}(\mathcal{M}_0)$. Pre operácie prislúchajúce strojom z $\mathcal{M} - \mathcal{M}_0$ sa postupne pre každý stroj vyrieši problém $1|\ r_i\ |L_{\max}$ a stroj, pre ktorý bolo $L_{\max}$ maximálne, sa označí za kritický. Postupnosť spracovania operácii tohto stroja je rovnaká, aká bola v riešení problému $1|\ r_i\ |L_{\max}$ pre stroj M. Množina $\mathcal{M}_0$ sa upraví : $\mathcal{M}_0 = \mathcal{M}_0 \cup \{M\}$. Iterácie pokračujú, kým sa neurčia postupnosti spracovania operácií na všetkých strojoch.

2.4 Metódy lokálneho prehľadávania a metaheuristiky

Metóda lokálneho prehľadávania nazývaná aj metóda prehľadávania okolia je založená na myšlienke, že aktuálne riešenie je možné zlepšiť nejakou malou zmenou. S každou ďalšou zmenou riešenia sa snažíme nájsť stále lepšie a lepšie riešenie. Každý kombinatoricko-optimalizačný problém, ktorý riešime, sa skladá z určitého počtu prvkov, ktoré ho popisujú, alebo definujú. Sú to: účelová funkcia (optimalizačné kritérium) $f : \Pi \to \mathbb{R}$, jednoduché pravidlo generujúce prechody z jedného riešenia do druhého a priestor riešení Π, ktorý je určený obmedzujúcimi podmienkami. Niekedy sa táto množina prvkov nazýva aj *konfigurácia problému*. Našou úlohou je nájsť optimálne riešenie π^*, kde $f(\pi^*) \leq f(\pi)$ pre každé $\pi \in \Pi$. Pozrieme sa bližšie na jednotlivé prvky konfigurácie.

Základom každej metaheuristiky je jednoduchá zmena aktuálneho riešenia π na iné susedné riešenie π'. Táto zmena sa bude nazývať *prechod* a označíme ju v. Pravidlo, ktoré definuje množinu všetkých prechodov $V(\pi)$ daného poradia spracovania π, nazveme *mechanizmus prechodov* a zobrazenie $\mathcal{N} : \Pi \to 2^\Pi$, ktoré vytvára množinu $\mathcal{N}(\pi)$ všetkých poradí spracovania, ktoré dostaneme z riešenia π, aplikovaním všetkých prechodov z $V(\pi)$ nazveme *okolie*. Nech $Q(\pi, v) \in \Pi$ je poradie spracovania, ktoré získam použitím prechodu v na riešenie π. Potom je okolie definované ako

$$\mathcal{N}(\pi) = \{Q(\pi, v) : v \in V(\pi)\}$$

Každé riešenie $\pi \in \mathcal{N}(\pi)$ sa nazýva susedné riešenie, alebo skrátene sused. Cieľom tejto metódy je progresívne pôsobiť na aktuálne riešenie prostredníctvom susedov tak, aby sme dosiahli zlepšujúce riešenie. Jednoduchú metódu lokálneho prehľadávania popisuje algoritmus 1.

V metódach lokálneho prehľadávania sa výber riešenia z okolia určuje pomocou *výberových kritérií*. Medzi bežné výberové kritéria patria:

- Nájdenie prvého lepšieho suseda (first improvement). Toto kritérium je použité napr. v threshold algoritmoch.
- Nájdenie najlepšieho suseda z okolia (best improvement). Je použité v shifting bottleneck, vo vkladacích algoritmoch, v tabu search a pod.

Vo všeobecnosti riešenie JSSP problému môžeme považovať za množinu rozhodnutí o tom, ktorá operácia sa rozvrhne v nasledujúcom kroku. Dorndorf a Pesch [25] navrhli, že by sa mala skonštruovať nejaká schéma, ktorá naviguje tieto lokálne rozhodnutia v oblasti prehľadávania tak, aby sa dosiahlo globálne kvalitné riešenie za rozumný čas. Špecifické heuristiky riešiace

Algoritmus 1 Schéma lokálneho prehľadávania okolia

1.Krok Inicializácia
- Formuluj počiatočné prípustné riešenie.
- Porovnaj riešenie s najlepším riešením. Ak riešenie vyhovuje pravidlu zastavenia **stop**. Ak nie a nastalo zlepšenie najlepšieho riešenia aktualizuj ho.

2.Krok Výber
- Urči ďalšie riešenie z okolia tak, aby vyhovovalo výberovému pravidlu.
- Ak výberové pravidlo nie je splnené, alebo ak sa aktivovalo pravidlo zastavenia **stop**.

3.Krok Kontrola
- Označ nový výsledok za aktuálny a choď na druhý stupeň kroku jedna.

daný problém, sú v tomto prípade krátkozraké, pokiaľ nie sú podporené nejakou metaheuristikou. Práve táto situácia dala priestor pre vznik metaheuristik, ktoré kombinujú špecifické vlastnosti problému lokálneho prehľadávania s vlasnosťami meta-riešiteľa, čo nám umožňuje dosiahnuť kvalitné riešenie. Metaheuristiky JSSP problému sú založené na okolí, ktoré navrhli Grabowski, Matsuo, Van Laarhoven, a Nowicki a Smutnicki.

2.4.1 Problem space metódy

Problem space sú heuristiky, ktoré generujú veľa rôznych počiatočných riešení, použitím rýchlych, špecifických, konštrukčných techník, ktoré sú potom iteračne zlepšované lokálnym prehľadávaním. Príkladom týchto heuristík sú *problem space* a *heuristic space* prehľadávanie, a *greedy random adaptive search* procedúra.

Prehľadávanie problem space a heuristic space. Nový pohľad na okolie navrhol Storer a kol. [71]. Usúdili, že radšej ako tradičné okolie budú prehľadávať okolie v priestore problému (problem space), alebo v priestore heuristiky (heuristic space). Obe tieto metódy v každej iterácii lokálneho prehľadávania využívajú nejakú rýchlu heuristiku h. Táto heuristika je prostriedkom na priame včlenenie špecifických znalostí problému do prehľadávania a preto je skonštruovaná tak, aby dávala dobré výsledky pre konkrétny problém.

Problem space vytvoríme zmenou pôvodného zadania rozvrhovacieho problému. Túto zmenu uskutočníme už spomínanou heuristikou h. Napríklad v JSSP môžeme použiť heuristiku SPT a budeme ňou meniť čas spracovania operácií. Okolie problem space môžeme definovať ako množinu všetkých zmenených rozvrhovacích problémov, kde pre každú zmenu času spracovania operácie platí, že je menšia ako predom určená konštanta ε. Nech P_0 je vektor pôvodných časov spracovania operácií a nech D je vektor veľkosti zmien časov spracovania, ktorý má rovnakú dimenziu ako vektor P_0. Okolie $\mathcal{N}_p(\pi)$ pôvodného problému je definované ako

$$\mathcal{N}_p(\pi) = \{\forall P_0 + D : \|D\| < \varepsilon\}$$

kde $\|\|$ predstavuje vzdialenosť. Okolie $\mathcal{N}_s(\pi)$, ktoré vznikne aplikáciou heuristiky h vyzerá nasledovne

$$\mathcal{N}_s(\pi) = \{\forall h(P) : P \in \mathcal{N}_p(\pi)\}$$

Podstatou metódy je, že heuristika pracuje (prideľuje operáciam prioritu) so zmenenými dátami, ale pre výsledný rozvrh a určenie jeho makespanu sú použité originálne dáta.

V *heuristics space* sa prehľadávanie spolieha na schopnosť definovať parametrickú verziu h. Pre Job-Shop môže parametrická funkcia vyzerať napríklad takto:

$$Priorita = \beta_1 PT + \beta_2 WR + \beta_3 OR$$

kde PT (job's processing time), WR (work remaining) a OR (operations remaining) sú prioritné dispečérske pravidlá. Nech

$$\beta_t \in \{Pravidlo_1, Pravidlo\ _2, \ldots, Pravidlo_R\} \text{ pre } t = 1, 2, \ldots, W$$

Vektor $(\beta_1, \ldots, \beta_W)$ je parametrizáciou dispečérskej heuristiky. Okolie v tomto priestore potom môže byť jednoducho definované, ako zmena jedného pravidla vo vektore pravidiel v danom čase.

Na prehľadávanie nového okolia sa potom použije nejaký algoritmus prehľadávania okolia napr. iterative improvement algoritmus. Storer a kol. [72] skombinoval heuristiku h popísanú vyššie s niekoľkými iteračnými algoritmami na prehľadávanie okolia. Výsledkom bolo, že jeho problem space genetický algoritmus sa zaradil medzi najlepšie metódy.

Greedy Random Adaptive Search procedúra (GRASP). Greedy random adaptive search procedúra sa skladá z dvoch fáz z konštrukčnej a z iteratívnej fázy. Konštrukčná fáza je založená na postupnom vkladaní operácií do rozvrhu, až kým sa nerozvrhnú všetky operácie. V iteračnej fáze nájdeme najlepšie riešenie z okolia rozvrhu, ktorý sme získali v konštrukčnej fáze.

V konštrukčnej fáze sa všetky operácie usporiadajú do zoznamu kandidátov, podľa toho aké ohodnotenie im pridelí tzv. *greedy funkcia*. Niekoľko najlepších kandidátov sa umiestni do reštriktívneho zoznamu kandidátov (RCL). Podstata stochastickej povahy algoritmu je v tom, že v danom kroku konštrukčnej fázy sa výsledná operácia z RCL vyberie náhodne.

V iteratívnej fáze sa použije procedúra lokálneho prehľadávania, ktorá následne nahradí súčasné riešenie lepším riešením z okolia. Táto fáza sa skončí, keď už nemôže nájsť lepšieho suseda. Potom sa algoritmus vráti do konštrukčnej fázy a vygeneruje sa nové počiatočné riešenie. Algoritmus sa skončí ak dosiahneme postačujúce riešenie, alebo sa vykoná zadaný počet iterácií.

Algoritmus 2 Princíp GRASP procedúry

1.Krok Inicializácia
- Konštrukčná fáza: vygeneruj riešenie.

2.Krok Výber
- Iteratívna fáza: Nájdi najlepšie riešenie z okolia riešenia vygenerovaného v konštrukčnej fáze.

3.Krok Kontrola
- Porovnaj riešenie s najlepším riešením. Ak riešenie vyhovuje pravidlu zastavenia **stop**. Ak nie a nastalo zlepšenie najlepšieho riešenia aktualizuj ho.
- Choď na krok jedna.

2.4.2 Threshold algoritmy

Veľmi populárnou skupinou lokálnych prehľadávacích metód sú *threshold algoritmy*, ktoré akceptujú prechod, ak rozdiel medzi aktuálnym riešením a riešením suseda je menší ako daný threshold (prah) $\mathcal{L}$, t.j. $f(\pi') - f(\pi) < \mathcal{L}$.

Iterative improved (IM). Jednoduchým príkladom threshold algoritmu je IM, ktorý príjme prvé zlepšujúce riešenie zo svojho okolia v dôsledku nastavenia thresholdu na nulu. Z počiatočného náhodne generovaného rozvrhu tieto metódy priamo hľadajú lokálne minimum dovtedy, pokiaľ je riešenie aspoň tak dobré, alebo lepšie než všetky riešenia v okolí. Iterative improved je najjednoduchšia trieda iteračných techník lokálneho prehľadávania, ktorá je základom pre komplikovanejšie metódy. Jednoduchou variáciou IM je *steepest descent* (najstrmšie klesanie). Táto metóda ohodnotí všetky prechody v jeho okolí a vyberie ten najlepší.

Aarts a kol. [1] [2] aplikoval multi-start iteratívny zlepšujúci algoritmus (MSIM), ktorý sa skončí keď vyprší časový limit. Multi-start IM začína z náhodne generovanej sekvencie. Potom tento algoritmus iteratívne zlepšuje aktuálne riešenie s lepším riešením z okolia. Ak sa nedá nájsť lepší sused t.j. dosiahne sa lokálne optimum, prehľadávanie začne znova z nejakého náhodne vybraného bodu. Toto sa opakuje pokiaľ nenastane jedna z podmienok zastavenia.

Treshold accepting (TA). V TA algoritme Dueck [27] sa nepoužívajú negatívne prahy. $\mathcal{L}$ je na začiatku nastavený na veľkú hodnotu, čo umožní ľahko prijať aj nezlepšujúce prechody. Postupne sa $\mathcal{L}$ znižuje až na nulu tak, že sa vyberajú iba zlepšujúce prechody. Algoritmy Great Deluge a Record-to-record Travel (RRT) Dueck [26] predstavujú dva varianty TA. Tieto metódy tiež akceptujú slabšie riešenie a to tak dlho pokiaľ sú nad určitým prahom.

Large step optimalizácia. Metóda optimalizácie veľkým krokom, ktorú navrhli Martin, Otto a Felten [54] [55], je dvojfázová optimalizačná metóda pozostávajúca z *veľkého kroku* a potom z *malého kroku*. Malé kroky sú najčastejšie vykonávané metaheuristikou, zatiaľ čo veľké kroky obsahujú aplikáciu špecifických techník problému, ktoré dovoľujú prekročiť lokálne minimum.

Algoritmus 3 Optimalizačná metóda large step

1.Krok Inicializácia
- Formuluj počiatočné prípustné riešenie x

2.Krok Vykonaj tento krok NUMITER krát
- Veľký krok: Urob taký prechod z x do x', ktorý silne naruší pôvodný rozvrh x
- Malý krok: Aplikuj metódu lokálneho prehľadávania na rozvrh x' a získaj rozvrh x''
- Vykonaj test prijateľnosti riešenia, ak x'' je prijateľné tak $x := x''$

3.Krok Vráť najlepšie nájdené riešenie x_{best}

Táto technika je relatívne nová a v JSSP má ešte málo aplikácií. Lourenço a Zwijnenburg [52] použili v malom kroku tabu search a vo veľkom kroku two random machine implementations (2rand-car) a zistili, že tento prístup dáva lepšie výsledky ako samostatný tabu search.

Simulated annealing (SA). Simulated annealing (simulované chladenie) je algoritmus využívajúci kladné a náhodne generované prahy. Nezávisle od seba ho navrhli Kirkpatrick a kol. [45] a Černý [15]. Algoritmus vychádza z fyzikálnej predstavy žíhania tuhého telesa. Najprv sa teleso zahreje na vysokú teplotu a potom sa postupne ochladzuje. V tomto procese je vysoká pravdepodobnosť zániku defektov kryštálovej mriežky telesa.

Pre naše potreby je teleso reprezentované aktuálnym riešením π a energia telesa (teplo) hodnotou $f(\pi)$. Metódou simulovaného chladenia teda minimalizujeme optimalizačné kritérium (teplo). Podstata algoritmu spočíva v tom, že nové riešenie π', ktoré vzniklo náhodne s π nahradí pôvodné riešenie π s pravdepodobnosťou Metropolisovho vzorca [57].

$$P(\pi \to \pi') = \min\left\{1, e^{\frac{-f(\pi)-f(\pi')}{c_i}}\right\}$$

Algoritmus 4 Simulated annealing

1.Krok Inicializácia
- Nastav počiatočnú teplotu $c_0 > 0$.

2.Krok Výber
- Náhodne vyber prechod z okolia.
- Ak je lepší než aktuálny prechod príjmi ho a choď na Krok 3.
- Ak si neprijal prechod skús to pomocou Metropolisovho vzorca. Ak ani toto nepomohlo choď na začiatok tohto kroku.
- Ak sa vykoná L cyklov zredukuj teplotu.
- Skonči keď je dosiahnuté pravidlo zastavenia.

3.Krok Kontrola
- Označ nový výsledok za aktuálny a choď na druhý kroku.

2.4.3 Genetické algoritmy (GA)

Ďalším typom optimalizačných techník, ktoré vychádzajú z procesov v prírode sú *genetické algoritmy* (Holland [39]). Genetické algoritmy sa snažia využiť predstavu a hnacích silách evolúcie živej hmoty. Základnou myšlienkou tohto prístupu je, že jedinec z lepšími schopnosťami (fitness) než bežná populácia má väčšiu šancu prežiť a stať sa rodičom. Potomkovia, ktorí vzniknú v tomto procese, majú ešte lepšie schopnosti prežiť ako ich rodičia. Na to aby sme mohli GA použiť musíme každé riešenie daného problému zakódovať do genetického reťazca, ktorý sa v súlade s genetickou terminológiou nazýva *chromozóm*. Jednotlivé položky tohto reťazca sa nazývajú *gény* a zvyčajne sa reprezentujú dvoma hodnotami 0, alebo 1. Počiatočná populácia vznikne náhodným vygenerovaním množiny chromozómov. Následne GA generáciu za generáciou zlepšujú tieto chromozómy až dovtedy pokiaľ nebude splnené pravidlo zastavenia. Typický GA proces obsahuje tri hlavné genetické operátory: výber, mutáciu a kríženie.

1. *Výber* - tento operátor náhodne vyberá jedincov z populácie na reprodukciu. Pravdepodobnosť, že bude jedinec na túto činnosť vybratý závisí od jeho hodnoty *fitness*.

2. *Kríženie* - dva vybrané chromozómy, ktoré môžeme považovať za partnerov sa navzájom skombinujú tak, aby vznikli nové chromozómy. Prvým krokom kríženia je náhodné vygenerovanie miest, v ktorých sa chromozómy rozdelia. V druhom kroku sa rozdelené časti chromozómov navzájom skombinujú a spoja. Vzniknú tak dva nové chromozómy.

3. *Mutácia* - Táto operácia náhodne zmení bit génu z 1 na 0, alebo z 0 na 1. Cieľom tejto činnosti je zabrániť uviaznutiu genetickému procesu v lokálnom minime. Pravdepodobnosť mutácie nejakého génu je obyčajne nízka, aby neprekážala genetickým operáciam.

Operátory v klasických GA pracujú s chromozómami, ktoré sú binárne zakódované, čo nevyhovuje rozvrhovacím problémom. Preto sa pre rozvrhovacie problémy vyvinuli efektívnejšie

kódovacie schémy. Reprezentačné schémy používané v Job-Shopoch môžeme voľne roztriediť do deviatich kategórií (Cheng a kol. [16]), ktoré sú založené na:

1. Operáciach
2. Úlohách
3. Vzťahoch medzi úlohami
4. Celkovom čase
5. Náhodných kľúčoch
6. Preferenčnom zozname
7. Prioritných pravidlách
8. Disjunktívnom grafe
9. Strojoch

Jednotlivé položky v schéme tohto delenia môžeme rozdeliť podľa spôsobu kódovania na priame a nepriame. V *priamych* metódach sa π zakóduje do chromozómu a na zlepšenie rozvrhu sa použijú genetické operátory, ktoré spôsobia vývoj týchto chromozómov. Položky 1 až 5 zo schémy delenia sú príkladom priamych metód kódovania. V *nepriamych* metódach sa zakóduje do chromozómu sekvencia rozhodovacích preferencií. Napríklad dispečérske pravidlo pre priradenie úlohy a genetický operátor sa aplikujú k zlepšeniu usporiadania rôznych preferencií. π je potom generovaný zo sekvencie preferencií. Príkladom týchto metód sú kategórie 6 až 9.

Prvú GA metódu na JSSP použil Davis [22] ako nepriamu metódu. Jeho metóda bola založená na preferovanom usporiadaní operácií na každom stroji. Tento prístup ďalej rozvíjali Falkenauer a Bouffouix [28], ktorí zakódovali operácie spracovávané na strojoch ako preferenčný zoznam obsahujúci reťazce symbolov.

Jedna z najmladších reprezentácií založená na preferenčnom zozname bola navrhnutá Kobayashim a kol. [47], kde chromozóm je reťazec symbolov dĺžky n a každý symbol identifikuje operáciu spracovávanú na stroji. Tamaki a Nishikawa [75] navrhli nepriamy spôsob založený na disjunktívnom grafe. V tomto prípade chromozóm obsahuje binárny reťazec korešpondujúci s usporiadaným preferenčným zoznamom disjunktívnych hrán.

Jednu z prvých priamych metód navrhli Nakano a Yamada [58]. Metóda spočíva v binárnom zakódovaní na základe precedenčných relácií medzi operáciami na stroji. Táto stratégia sa nazýva *forcing* a modifikuje chromozóm vtedy, ak môže operácia vykonať ľavý posun. Yamada a Nakano tento prístup ešte vylepšili definovaním operátora kríženia GA/GT, ktorý je založený na Giffler-Thompsonovom algoritme.

Norman a Bean [59] navrhli priamu metódu založenú na náhodných kľúčoch. Každý gén (náhodný kľúč) pozostáva z dvoch častí: z celej časti čísla z množiny $\{1, 2, \ldots, m\}$ a desatinnej časti čísla z intervalu $(0, 1)$. Celá časť génu predstavuje stroj a desatinná časť poradie spracovania na stroji.

Bierwirth [12] vytvoril všeobecnú permutačnú schému GA, s cieľom zlepšiť existujúce metódy. Jeho chromozóm reprezentuje permutáciu úloh. V ďalších prácach analyzovali tri operátory kríženia, ktoré uchovávajú relatívne a absolutné permutačné poradie operácií.

I napriek tomu, že existuje množstvo prístupov na riešenie JSSP založených na GA, dosiahnuté výsledky nie sú dobré. Z jednotlivých prác vyplýva, že GA nie sú dobré hlavne v koncovkách, lebo operátory kríženia postupne strácajú svoju efektívnosť. Na riešenie týchto problémov sa do GA vložila metóda známa ako *Genetic Local Search* (GLS). V GLS schéme dieťa, ktoré vznikne z GA operátora je použité ako počiatočné riešenie pre prehľadávanie okolia, ktoré iteratívne mení potomka pokiaľ sa dostaneme k lokálnemu optimu. Lokálne minimum sa potom použije v ďalšej generácií spolu s rekombinačným operátorom.

Výhody použitia GLS v genetických algoritmoch zdôraznil vo svojich prácach Della Croce [20], ktorý začlenil do svojich genetických algoritmov rôzne štruktúry okolia a zlepšil tak niektoré dovtedajšie výsledky.

Algoritmus 5 Genetic local search

1.Krok ***Inicializácia.***
- Formuluj populáciu počiatočných prípustných riešení.
- Porovnaj riešenia s najlepším riešením. Ak nejaké riešenie vyhovuje pravidlu zastavenia **stop**. Ak nie a nastalo zlepšenie najlepšieho riešenia aktualizuj ho.

2.Krok ***Výber***
- Urči fitness-i populácie a vyber vhodných rodičov.
- Prvá úroveň: Použi na vybratých rodičoch rekombinačný operátor.
- Druhá úroveň: Aplikuj metódu generovania okolia na deti pokiaľ sa nedosiahne lokálne optimum.

3.Krok ***Kontrola***
- Priraď lokálne optimálne dieťa do populácie zmazaním slabších členov. potom choď na druhý stupeň kroku jedna.

2.4.4 Tabu search (TS)

Iteratívnu optimalizačnú techniku *tabu search* (zakázané prehľadávanie) navrhol a prvý krát použil Glover v roku 1985 a neskôr [35] [36] [37] ju postupne vylepšoval. Metóda štartuje z prípustného počiatočného riešenia, ktoré sme získali ľubovoľnou konštrukčnou metódou, napr. prioritným dispečérskym pravidlom. Identifikuje množinu prechodov a nájde v nej najlepší prechod (prehľadá sa celé okolie). Vykonaním tohto prechodu získame riešenie, ktoré je východzím bodom pre ďalší krok metódy. Presnejšie povedané, identifikujeme novú množinu prechodov, ktorá prislúcha novému riešeniu. Z novej množiny prechodov zase vyberiem najlepší prechod a takto sa to opakuje. Treba povedať, že najlepším prechodom sa myslí taký prechod, ktorého vykonaním dosiahneme minimálnu hodnotu optimalizačného kritéria.

Ako prevenciu proti zacykleniu metódy v lokálnom optime a tiež na navádzanie metódy do perspektívnych oblastí sa uchováva história prechodov v pamäti. Rozoznávame dva druhy pamäte: krátkodobú pamäť pre mladšie prechody a dlhodobú pamäť pre staršie prechody. Krátkodobá pamäť sa nazýva *tabu list* a hrá hlavnú úlohu v tejto metóde.

Význam tabu listu spočíva v tom, že nedovoľuje vykonať prechod, ktorým by sme sa vrátili späť do nejakého riešenia, v ktorom sme už boli pred *maxt-* krokmi, kde *maxt-* je určená hodnota, ktorá udáva dĺžku tabu listu. Tabu list je teda prakticky množina zakázaných prechodov.

Môže sa však stať, že pre nás zaujímavý prechod bude tabu prechodom. Na to, aby sme mohli takýto prechod použiť si musíme definovať *aspiračnú funkciu*. Táto funkcia nám bude zhodnocovať zisk, ktorý nám vznikne použitím tohto tabu prechodu. Pravidlá pre zastavenie metódy môžu byť rôzne napríklad: nájdenie riešenia, ktoré je dostatočne blízke cieľovej hodnote, vykonanie n prechodov bez zlepšenia najlepšieho výsledku, alebo po vypršaní časového limitu pre chod programu.

2.5 Ant colony

Biologické štúdie mravčích kolón ukázali, že ich správanie je komplexné a organizované. Bolo dokázané, že jeden mravec nie je schopný priamo komunikovať s ostatnými mravcami o predošlých skúsenostiach. Štúdie popisujú ako mravce skúmajú svet pri hľadaní zdrojov jedla, a ako potom nájdu cestu späť k mláďatám. Počas tejto cesty identifikujú zdroj jedla pre ostatné mravce v kolóne. Robia to nepriamo pomocou feromónových stôp. Každý mravec po

Algoritmus 6 Jednoduchý tabu search algoritmus

1.Krok ***Inicializácia***

- Formuluj počiatočné prípustné riešenie.
- Začni s prázdnym tabu listom.
- Porovnaj získané riešenie s najlepším riešením. Ak riešenie vyhovuje pravidlu zastavenia **stop**. Ak nie a nastalo zlepšenie najlepšieho riešenia aktualizuj ho.

2.Krok ***Výber***

- Urči najlepšie riešenie z kandidátskej množiny, ktorá je podmnožinou okolia tak, aby bolo zohľadnené výberové pravidlo, tabu podmienky a aspiračná funkcia.
- Ak sa aktivovalo pravidlo zastavenia **stop**.

3.Krok ***Kontrola***

- Aktualizuj tabu list vložením vybraného prechodu. Označ nový výsledok za aktuálny a choď na tretí stupeň kroku jedna.

objavení jedla, zanecháva na ceste späť k mláďatám čiastočky feromónov, tak aby ostatné mravce tiež našli cestu k jedlu. Akumulované množstvo feromónov slúži ako distribuovaná pamäť, zdielaná všetkými ostatnými mravcami a označuje najpoužívanejšie cesty medzi mláďatami a jedlom. Mravce, ktoré sa stretnú s prekážkou medzi mláďatami a jedlom si najskôr vyberú náhodne smer obchádzania. Akonáhle je cesta označkovaná feromónovými časťami, bude závisieť od výberu ďalšieho mravca konfrontovaného s rovnakou prekážkou, či táto cesta zvýši svoj feromónový kredit. Čím viac mravcov prejde v nejakom čase po ceste, tým viac je na nej feromónov a tým je atraktívnejšia. Treba si ale uvedomiť, že s pribúdajúcim časom feromónové stopy vyprchajú a teda dlhšie cesty sú viac penalizované.

Dorigo [17] inšpirovaný týmto správaním navrhol algoritmus *ant system* na riešenie kombinatoricko-optimalizačných problémov, ktorý používa populáciu kooperujúcich agentov komunikujúcich prostredníctvom distribuovanej zdielanej pamäti.

Základnou časťou algoritmu je populácia m umelých mravcov, ktoré cyklicky generujú riešenia nejakého optimalizačného problému. Algoritmus pracuje na grafe, v ktorom mravce chodia po každej hrane z jedného vrchola do druhého a tak vytvárajú cestu reprezentujúcu riešenie. Každý vrchol môže mravec navštíviť iba raz a preto si udržuje zoznam navštívených vrcholov Z_k. Vo vrchole i sa mravec rozhodne pre ďalší vrchol j s pravdepodobnosťou:

$$P_{ij}(t) = \frac{[\tau_{ij}(t)]^{\alpha} \cdot \left[\frac{1}{d_{ij}}\right]^{\beta}}{\sum_{j \in T} [\tau_{ij}(t)]^{\alpha} \cdot \left[\frac{1}{d_{ij}}\right]^{\beta}},$$

kde τ_{ij} je množstvo feromónov na hrane medzi vrcholmi i a j, d_{ij} vzdialenosť medzi vrcholmi i a j, t čas a T množina vrcholov, ktoré nie sú v zozname Z_k. Parametre α a β nastavujú relatívny vplyv feromónov a vzdialenosti.

Keď každý mravec absolvuje kompletnú cestu, feromónové trasy sa aktualizujú podľa vzorca:

$$\tau_{ij}(t+1) = \rho\tau_{ij}(t) + \Delta\tau_{ij}(t, t+1),$$

kde ρ $(0 \leq \rho < 1)$ je koeficient vyjadrujúci mieru odparovania a $\Delta\tau_{ij}(t, t+1)$ je suma

pridaných feromónov na hrane (i, j).

$$\Delta\tau_{ij}(t, t+1) = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t, t+1)$$

kde $\Delta\tau_{ij}^{k}(t, t+1)$ je množstvo položených feromónov na jednotku dĺžky k-tim mravcom na ceste z vrchola i do vrchola j medzi časmi t a $t+1$. Existujú tri spôsoby ako vypočítať $\Delta\tau_{ij}^{k}(t, t+1)$:

1. *ant-quantity* vždy keď mravec cestuje z vrchola i do vrchola j zanechá na ceste konštantné množstvo feromónov Q_1

$$\Delta\tau_{ij}^{k}(t, t+1) = \begin{cases} Q_1/d_{ij} & \text{ak } k\text{-ty mravec ide z } i \text{ do } j \text{ v čase medzi } t \text{ a } t+1 \\ 0 & \text{inak} \end{cases}$$

2. *ant-density* na každej dĺžkovej jednotke cesty zanechá konkrétny mravec Q_2 feromónových jednotiek.

$$\Delta\tau_{ij}^{k}(t, t+1) = \begin{cases} Q_2 & \text{ak } k\text{-ty mravec ide z } i \text{ do } j \text{ v čase medzi } t \text{ a } t+1 \\ 0 & \text{inak} \end{cases}$$

3. *ant-cycle* $\Delta\tau_{ij}^{k}$ vypočíta po n iteráciach. V nasledujúcom vzorci premenná L_k predstavuje dĺžku trasy k-teho mravca.

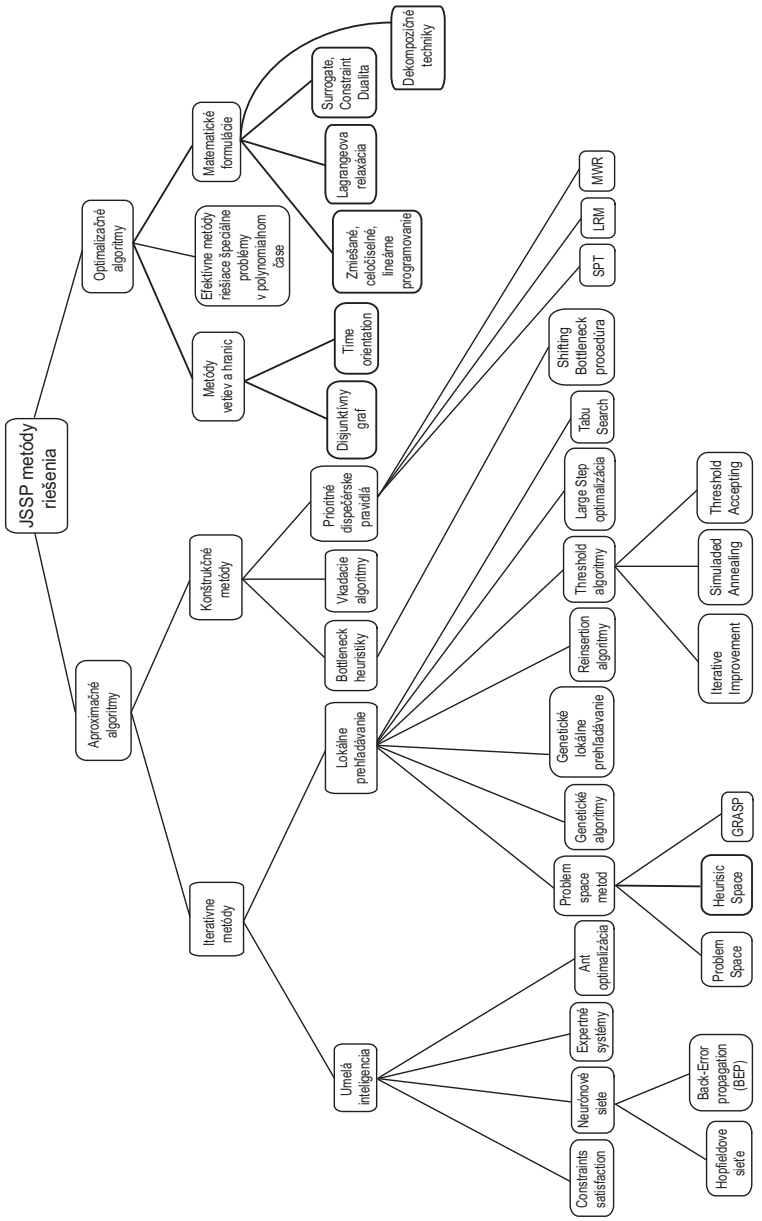
$$\Delta\tau_{ij}^{k}(t, t+n) = \begin{cases} Q_3/L_k & \text{ak } k\text{-ty mravec ide z } i \text{ do } j \text{ v čase medzi } t \text{ a } t+1 \\ 0 & \text{inak} \end{cases}$$

$$\tau_{ij}(t+n) = \rho\tau_{ij}(t) + \Delta\tau_{ij}(t, t+n),$$

$$\Delta\tau_{ij}(t, t+n) = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t, t+n)$$

Colorni, Dorigo, a Maniezzo boli prvý, ktorý použili *ant system* na JSSP problems z 15 strojmi a 15 úlohami. Ich riešenia sa odlišovali od optima o desať percent.

Zwaan and Marques navrhli odparovaciu konštantu a pohyblivé parametre. Ich riešenia sa v úlohe $J_{10}||C_{\max}$ odlišovali od optima o osem percent.



Obr. 2.1: Rôzne metódy na riešenie JSSP

KAPITOLA 3

NÁMETY A TEORETICKÉ VÝCHODISKÁ

3.1 Výpočtová zložitosť

Pri skúmaní Job-Shop rozvrhovacích problémov je veľmi dôležité uvedomiť si ich zložitosť. Ak sa náš problém skladá z n úloh, musí každý stroj spracovať n operácií, potom teda počet všetkých možných usporiadaní operácií na jednom stroji je $n!$. Ak počet permutácií pre každý stroj je $n!$, potom počet všetkých možných rozvrhov bude $(n!)^m$. Napríklad problém s 20 úlohami a 10 strojmi ma $7,2651.10^{183}$ možných riešení. Avšak veľa týchto rozvrhov nie je prípustných vzhľadom na podmienky precedencie a disjunkcie. Analýzou výpočtovej zložitosti JSSP za vo svojich prácach venovali Cook [19], Karp [44], Garey a Johnson [31]. Z týchto prác vyplýva, že Job-Shop patrí do triedy $\mathcal{NP}$-ťažkých problémov a vo všeobecnosti sa považuje za jeden z výpočtovo najzložitejších v matematike. Je treba poznamenať, že existujú aj algoritmy, ktoré riešia špecializované úlohy malých rozmerov v polynomiálnom čase, ale tie nám v praxi veľmi nepomôžu. Zaujímavý je tiež poznatok, že čím viac sa pomer úloh k strojom blíži k jednotke (t.j. počet strojov je blízky, alebo rovnaký počtu úloh), tým sa úloha ťažšie rieši. Z dôvodov takejto náročnosti sa pri riešení JSSP s výhodou používajú heuristické a metaheuristické metódy.

3.2 Benchmarkové problémy

Nájsť porovnateľné merítko rôznych metód a algoritmov, ktoré sa používajú na riešenie JSSP problému si vyžiadalo vznik *benchmarkových problémov*, ktoré sú bežným štandardom na ich testovanie a vzájomné porovnávanie. Momentálne existuje viacero sád benchmarkových problémov od rôznych autorov, ako Fisher a Thompson (FT) [29], Lawrence (LA) [49], Adams a kol. (ABZ) [3], Applegate a Cook (ORB) [5], Storer a kol. (SWV) [71], Yamada a Nakano (YN) [80], Taillard (TD) [74] a Demirkol a kol. (DMU) [24], ktoré sú voľne dostupné na internete[1]. Čas spracovania týchto problémov je určený náhodne v rámci daného intervalu generátormi, ktoré si autori navrhli sami.

[1]FT, LA, ABZ, ORB, SWV a YN problémy sú dostupné na ftp adrese ftp://mscmga.ms.ic.ac.uk/pub/jobshop1.txt, TD problém je dostupný na ftp adrese ftp://mscmga.ms.ic.ac.uk/ pub/jobshop2.txt a DMU je dostupné na ftp adrese ftp://gilbreth.ecn.purdue.edu/~uzsoy2/benchmark/problems.html

3.3 Ďalšie prvky JSSP

V predchádzajúcich častiach sme pri opise a definícii Job-Shop rozvrhovacieho problému a metaheuristik používaných pri jeho riešení, spomínali tieto dôležité prvky: úlohy, stroje, operácie, okolie a prechod. Prvé tri prvky boli definované z hľadiska rozvrhov, ale okolie a prechod iba z hľadiska pojmov lokálneho prehľadávania. Čo si treba predstaviť pod pojmami okolie a prechod konkrétne v rozvrhoch si teraz ukážeme. Ďalej treba zadefinovať ešte ďalšie dva dôležité pojmy z JSSP *kritickú cestu* a *blok kritickej cesty*.

3.3.1 Kritická cesta a jej bloky

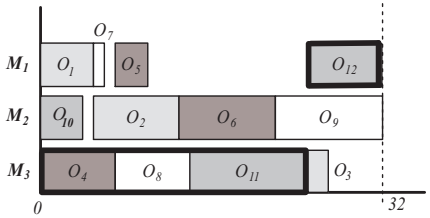
Najdlhšia cesta z O_S do O_E v grafe $\mathbb{G}$ reprezentujúca makespan sa nazýva *kritická cesta* a operácie z ktorých sa táto cesta skladá sa nazývajú *kritické operácie*. Kritickosť týchto operácií spočíva v tom, že ak sa kritická operácia oneskorí o nejaký čas o ten istý čas sa oneskorí aj makespan rozvrhu. Samozrejme za kritické nemôžeme považovať operácie O_S a O_E, ktoré sú len fiktívne. V ďalšom popise budeme uvažovať o kritickej ceste bez týchto operácií.

Nech $u = (u_1, u_2, \ldots, u_w)$ je kritická cesta a nech $u_i \in \mathcal{O}$, $1 \geq i \geq w$, kde w je dĺžka kritickej cesty, je kritická operácia. Potom dĺžka daného prípustného rozvrhu sa rovná súčtu všetkých časov spracovania kritických operácií.

Kritickú cestu môžeme rozložiť na reťazce $B_1, \ldots, B_r$, ktoré sa nazývajú bloky. Pre bloky platí:

- $B_j = (u_{a_j}, u_{a_j+1}, \ldots, u_{b_j})$, kde
  $j = 1, \ldots, r$
  $1 = a_1 \leq b_1 < b_1 + 1 = a_2 \leq b_2 < b_2 + 1 = a_3 \leq \ldots \leq a_r \leq b_r = w$
- Blok B_j obsahuje operácie, ktoré sú spracovávané na rovnakom stroji.
- Dva za sebou idúce bloky B_j a B_{j+1} obsahujú operácie spracovávané na rôznych strojoch

Jednoducho povedané blok je maximálny súvislý reťazec kritických operácií na kritickej ceste, ktorého operácie sa spracovávajú na rovnakom stroji.



Obr. 3.1: Ganttov diagram zo zvýraznenou kritickou cestou

Na obrázku 3.1 je kritickou cestou postupnosť operácií O_4, O_8, O_{11}, O_{12}. Prvý blok tvoria operácie O_4,O_8,O_{11} a druhý blok je tvorený operáciou O_{12}. V jednom rozvrhu môže existovať aj viacej kritických ciest.

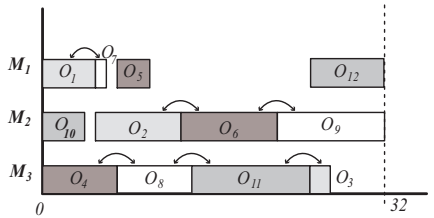
3.3.2 Prechod

Menej striktná definícia prechodu hovorí, že prechod je výmena poradia spracovávania dvoch susedných bezprostredne po sebe spracovávaných operácií na rovnakom stroji. Iná striktnejšia od Van Laarhovena [48] a Taillarda [73] definuje prechod v pre rozvrh π ako výmenu dvojice operácií (x, y), pre ktoré platí:

- x a y sa spracovávajú po sebe na rovnakom stroji
- x a y sú po sebe idúce operácie na kritickej ceste v grafe $\mathbb{G}$.

3.3.3 Okolia pre JSSP

Časová zložitosť algoritmov riešiacich JSSP závisí hlavne od veľkosti prehľadávaného okolia a od zložitosti výpočtu zhodnotenia prechodu. Vo všeobecnosti okolie určené menej striktnými prechodmi je pomerne veľké orázok 3.2. Preto sa vyvíjala snaha o jeho zredukovanie.



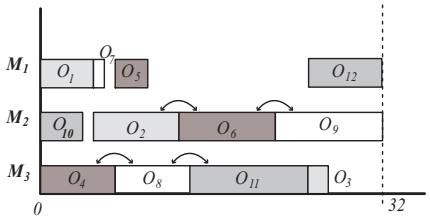
Obr. 3.2: Okolie definované menej striktnými prechodmi

Van Laarhoven [48] definoval okolie ako množinu rozvrhov získaných z rozvrhu π postupnou aplikáciou všetkých prechodov na kritických cestách v grafe $\mathbb{G}$ obrázok 3.3. Jeho okolie má tieto vlastnosti:

- Ak $\pi \in \Pi$ je prípustný rozvrh, tak výmenou dvoch susedných kritických operácií, ktoré sa spracovávajú na rovnakom stroji nevznikne nikdy neprípustné riešenie.
- Ak výmenou dvoch susedných operácií, ktoré nie sú kritické, získame z riešenia π nové riešenie π', tak kritická cesta v π' nemôže byť kratšia ako kritická cesta z π, pretože kritická cesta z π stále existuje v π'.
- Ak vyštartujeme z nejakého prípustného riešenia π, tak existuje postupnosť prechodov, ktorá nás privedie do optimálneho riešenia.

Prechody opakujúce sa vo viacerých kritických cestách sa v okolí použijú iba raz. Veľkosť tohto okolia závisí od počtu operácií na kritických cestách. Takto definované okolie je síce zredukované, ale stále môže byť relatívne veľké.

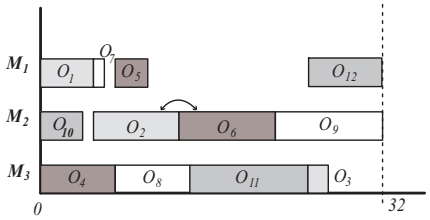
Ďalšiu redukciu okolia navrhol Matsuo [56]. Vo svojej práci dokázal, že ak sa vymenia dve kritické susedné operácie i a j, prechod nebude zlepšujúci, ak predchodca operácie i na stroji aj následník operácie j na stroji sú tiež kritické operácie. Teda význam majú iba výmeny



Obr. 3.3: Okolie navrhnuté Van Laarhovenom

prvých dvoch a posledných dvoch operácií v rámci každého bloku. Výmeny vo vnútri bloku nemajú význam.

Najväčšiu redukciu okolia až 90% dosiahli Nowicki a Smutnicki [60]. Vo svojej práci využívajú množinu prechodov tvoriacu okolie obrázok 3.4, ktorá obsahuje výmeny na hraniciach blokov, t.j. na začiatku a na konci bloku podobne ako Matsuo. Ich metóda má však dve odlišnosti. Prvou je, že vo svojej práci dokázali, že výmena prvej (resp. poslednej) dvojice operácií v prvom (resp. poslednom) bloku priamo nezlepšuje makespan ak tieto bloky obsahujú viac než dve operácie. Druhou odlišnosťou oproti predchádzajúcim okoliam je, že využívajú prechody vždy iba z jednej kritickej cesty.



Obr. 3.4: Okolie navrhnuté Nowickim a Smutnickim

Teda okolie navrhnuté Nowickim a Smutnickim tvorí množina prechodov vytvorená:

- V prvom bloku výmenou posledných dvoch operácií.
- V poslednom bloku výmenou prvých dvoch operácií.
- V bloku, ktorý nie je prvý ani posledný výmenou prvých dvoch operácií zo začiatku bloku a posledných dvoch operácií z konca bloku.

3.4 Námety pre okolie

Z predchádzajúceho popisu okolia Job-Shop rozvrhovacieho problému vyplýva, že správne navrhnutie okolia je jedna z najdôležitejších častí pri tvorbe každého algoritmu na riešenie tohto

problému. Momentálne sa ako najefektívnejšie javí okolie navrhnuté Nowickim a Smutnickim. I napriek takémuto kvalitnému okoliu a množstvu navrhnutých metód ešte stále existujú benchmarkové problémy, u ktorých sa nepodarilo nájsť optimálne riešenie. Preto by som sa v mojej práci pokúsil:

- Navrhnúť sofistikovanejšie okolie, ktoré by sa menilo v reálnom čase v závislosti od vývoja aktuálneho stavu riešenia. Konkrétne by sa menila veľkosť a štruktúra. Napríklad ak by v danom okolí nebol dostatočne dobrý prechod, okolie by sa samo zväčšilo (vytvorilo by sa iné väčšie okolie podľa inej definície).
- Paradoxne rozšíriť okolie a tým sa pokúsiť zrýchliť pokles k minimu. Nech mám riešenie π a k nemu prislúchajúcu množinu prechodov $V(\pi)$. Na rozvrh π aplikujem každý prechod z $V(\pi)$ a dostanem určité množstvo rozvrhov, z ktorých každý má svoje prechody atď. Nakoniec si vyberiem najlepší prechod. V tomto prípade bude dôležité preskúmať či je to efektívne a ak áno, do akej hĺbky sa oplatí ísť.
- Preskúmať možnosti Problem Space a Heuristic Space a pokúsiť sa navrhnúť podobné okolia, založené na nejakom porušení pôvodného zadania. Preskúmať do akej miery je vhodné problém narušiť.
- Pokúsiť sa zostrojiť viackriteriálne hodnotenie pri výbere prechodu t.j. vybrať prechod nie len podľa hodnoty makespanu, ale napríklad aj podľa štruktúry kritickej cesty a pod.

3.5 Námety pre kritickú cestu

Kritická cesta je jeden z hlavných komponentov podieľajúci sa na tvorbe okolia, pretože priamo ovplyvňuje rozsah množiny prechodov. Jej analýza môže prispieť k návrhu efektívnejšieho okolia. Preto by som chcel:

- Evidovať a analyzovať úseky kritickej cesty, ktoré sa často opakujú. Ak budem poznať dostatočne odlišné úseky viacerých kvalitných riešení, môžem ich analýzou zistiť, ktoré časti kritickej cesty sú vhodné na to, aby sa vyskytli v ďalších pokusoch nájsť optimum.
- V rámci kritickej cesty vyhodnocovať odchylky jednotlivých operácií od ich času najskoršie možného rozvrhnutia.

3.6 Komponenty a kombinácie heuristík

Každá heuristika, alebo metaheuristika sa skladá z určitých komponentov. Každý komponent má určité vlastnosti, ktoré sa dajú zmeniť a nastaviť pomocou parametrov. Práve nastavovanie týchto parametrov vo veľkej miere rozhoduje o tom, do akej miery bude heuristika efektívna a úspešná pri hľadaní riešenia. Hľadanie tých správnych parametrov je jedna z najťažších úloh a väčšinou si vyžaduje interaktivitu s človekom.

V svojej práci by som chcel pre jednotlivé metaheuristiky navrhnúť metódu automatického inteligentného výberu najlepších možných parametrov. Táto úloha si vyžiada zostavenie špeciálnych hodnotiacich kritérií, ktoré budú vychádzať zo zadania. Niektoré parametre by sa mohli meniť aj počas behu programu ako je to napríklad v metóde *Reactive Tabu-Search*.

Ďalším dôležitým prvkom metaheuristik je tzv. *počiatočné riešenie*. Je to riešenie, z ktorého metaheuristiky vychádzajú na začiatku svojej činnosti. Vo svojich prácach Applegate [5] a Lourenço [51], [50], upozornili na to, že metódy prehľadávania okolia závisia aj na počiatočnom riešení. Keď je počiatočné riešenie slabé, môže to byť aj príčinou slabého riešenia. Význam kvalitného počiatočného riešenia sa prejaví hlavne keď sa má metóda ukončiť po určitom počte iterácií.

KAPITOLA 4

CIEĽ, MATERIÁL A METÓDY PRÁCE

4.1 Cieľ práce

Hlavným cieľom práce je analyzovať heuristické a metaheuristické techniky, ktoré sa v súčasnosti používajú na riešenie Job-Shop rozvrhovacieho problému a navrhnúť ich zlepšenia, alebo vytvoriť nové metódy. Súčasťou práce bude analýza priestoru riešení, okolia a jeho metód generovania, analýza rôznych nastavovacích parametrov metaheuristík a analýza možnosti kombinácie rôznych metaheuristík (tzv. hybridné metódy).

Výsledkom práce by mali byť metódy a techniky, ktoré zabezpečia efektívne riešenie Job-Shop rozvrhovacieho problému, metódy a postupy analýzy priestoru, v ktorom sa problém rieši a pokus o zovšeobecnenie niektorých metód.

4.2 Materiál a metódy práce

V slovenskom, alebo českom jazyku je prác s tématikou teórie rozvrhovania veľmi málo. Medzi výnimky patria učebné texty Palúch [61], ktoré sú celé venované teórii rozvrhovania, kniha Sieťová analýza [76], v ktorej je v jednej z kapitol popis rozvrhovania a niektorých algoritmov a pár iných kníh, ktoré sa krátko zmieňujú o rozvrhovaní. Väčšinu materiálu potrebného na prácu som získal z internetu v elektronickej podobe.

4.3 Tézy dizertačnej práce

V mojej práci by som sa chcel zamerať hlavne na analýzu okolia, kritickej cesty, jednotlivých komponentov metaheuristík. Na riešenie pre dosiahnutie stanovených cieľov je potrebné:

- Podrobne analyzovať vybrané metaheuristiky, ako sú *tabu search*, *simulated annealing*, *large step*, *GRASP* atď., pokúsiť sa navrhnúť ich efektívne kombinácie a vytvoriť prehľad v študovanej problematike.
- Preskúmať, analyzovať a pokúsiť sa vytvoriť nové efektívnejšie metódy na generovanie počiatočného riešenia.
- Analyzovať už existujúce a skúsiť navrhnúť rôzne nové druhy okolí.

- Analyzovať kritickú cestu jej prvky a vzťahy medzi nimi.
- Preskúmať vplyv parametrov jednotlivých metaheuristík a navrhnúť možnosť ich automatické nastavovania pred a počas priebehu algoritmu.
- Navrhnuté algoritmy programovo efektívne implementovať na počítači.
- Preskúmať chovanie navrhnutých algoritmov a overiť ich na dostatočnom počte referenčných (benchmarkových) úloh, ktoré sú na ich testovanie určené a všeobecne používané.

LITERATÚRA

[1] E.H.L. Aarts, P.J.M. Van Laarhoven, and N.L.J. Ulder. Local search based algorithm for job-shop scheduling. Technical report, Department of Mathematics & Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands, 1991.

[2] E.H.L. Aarts, P.J.M. Van Laarhoven, and N.L.J. Ulder. Computational study of local search algorithms for job-shop scheduling. *ORSA Journal on Computing*, 6(2):118–125, 1994.

[3] J. Adams, E. Balas, and D. Zawack. The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34:391–401, 1983.

[4] S.B. Akers and J. Friedman. Non numerical approach to scheduling problems. *Operations Research*, 3:429–442, 1955.

[5] D. Applegate and W. Cook. A computational study of the job-shop scheduling problem. *ORSA Journal on Computing*, 3:149–156, 1991.

[6] E. Balas. An additive algorithm for solving linear programs with zero-one variables. *Operations Research*, 13:517–546, 1965.

[7] E. Balas. Diskrete programing by the filter method. *Operations Research*, 15:915–957, 1965.

[8] E. Balas. Machine sequencing via disjunctive graph: An implicit enumerative algorithm. *Operations Research*, 17:1–10, 1969.

[9] E. Balas. Machine sequencing: disjunctive graphs and degree-constrained subgraphs. *Naval Research Logistics Quqrterly*, 17:941–957, 1970.

[10] R. Battitti and G. Tecchiolli. The reactive tabu search. *ORSA Journal on Computing*, 6(2):126–140, 1994.

[11] J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.

[12] C. Bierwirth. A generalized permutation approach to job-shop scheduling with genetic algorithms. *OR Spektrum*, 17(2–3):87–92, 1995.

[13] S. Binato, W.J. Hery, D.M. Loewenstern, and M.G.C. Resende. A grasp for job shop scheduling. In C.C. Ribeiro and P. Hansen, editors, *Essays and surveys on metaheuristics*, pages 59–79. Kluwer Academic Publishers, 2001.

[14] J. Černý and P. Kluvánek. *Základy matematickej teórie dopravy.* Veda, Bratislava, 1991.

[15] V. Cerny. Thermodynamical approach to the travelling salesman problem: An effecient simulation alhorithm. *Journal of Optimization Theory and Applications*, 45:41–51, 1985.

[16] R. Cheng, M. Gen, and Y. Tsujimura. A tutorial survey of job-shop scheduling problems using genetic algorithms-i. representation. *Computers & Industrial Engineering*, 30(4):983–997, 1996.

[17] A. Colorni, M. Dorigo, and V. Maniezzo. Distributed optimization by ant colonies. In *Proceedings of ECAL91 - European Conference on Artificial Life,*. Elsevier Publishing, 1991.

[18] R.W. Conway, W.L. Maxwell, and L.W. Miller. *Theory of Scheduling*. Addison-Wesley, Reading Massachusetts, 1967.

[19] Stephen A. Cook. The complexity of theorem-proving procedures. In ACM, editor, *Conference record of third annual ACM Symposium on Theory of Computing: papers presented at the symposium, Shaker Heights, Ohio, May 3, 4, 5, 1971*, pages 151–158, New York, NY, USA, 1971. ACM Press.

[20] F. Della Croce, R. Tadei, and R. Rolando. Solving a real world project scheduling problem with a genetic approach. *Belgian Journal of Operations Research, Statistics and Computer Science*, 33(1–2):65–78, 1994.

[21] G.B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Res.*, 8:101–111, 1960.

[22] Lawrence Davis. Job scheduling with genetic algorithms. In John J. Grefenstette, editor, *Proceedings of the First International Conference on Genetic Algorithms and their Applications (ICGA'85)*, pages 136–140, Hillsdale, New Jersey, 1985. Lawrence Erlbaum Associates.

[23] W. Davis and A. Jones. A real-time production scheduler for stochastic manufacturing enviroment. *Internacional Journal of Computer Integrated Manufacturing*, 1(2):101–112, 1988.

[24] E. Demirkol, S. Mehta, and R. Uzsoy. Benchmarks for shop scheduling problems. *European Journal of Operational Research*, 109(1):137–141, 1998.

[25] U. Dorndorf and E. Pesch. Evolution based in a job-shop scheduling enviroment. *Computers and Operations Research*, 22(1):25–40, Jan 1995.

[26] Gunter Dueck. New optimization heuristics: The great deluge algorithm and the record-to-record travel. *Journal of Computational Physics*, 104:86–92, 1993.

[27] Gunter Dueck and Tobias Scheuer. Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing. *J. Comput. Phys. 90*, (1):161–175, 1990.

[28] E. Falkenauer and S. Bouffouix. A genetic algorithm for jop shop. In C. Harris, A. Copeland, and L. O'Conner, editors, *Proc. of the 1991 IEEE Int. Conf. on Robotics and Automation*, pages 824–829, Los Alamitos, CA, 1991. IEEE Computer Society Press.

[29] H. Fisher and G. L. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. In J. F. Muth and G. L. Thompson, editors, *Industrial Scheduling*, pages 225–251. Prentice-Hall, Englewood Cliffs, 1963.

[30] H.L. Gantt. Efficiency and democracy. *Transactions of the American Society of Mechanical Engineers*, 40:799–808, 1919.

[31] M. R. Garey, D. S. Johnson, and R. Sethi. The complexity of flowshop and jobshop scheduling. *Math. of Op. Res.*, 2(2):117–129, May 1976.

[32] W.S. Gere. Heuristics in job-shop scheduling. *Management Science*, 13:167–190, 1966.

[33] S. B. Gershwin. Hierarchical flow control: a framework for scheduling and planning discrete events in manufacturing systems. *Proceedings of IEEE Special Issue on Discrete Event Systems*, (77):195–209, 1989.

[34] B. Gifler and G.L. Thompson. Algorithms for solving production scheduling problems. *Operations Research*, 8(4):487–503, 1960.

[35] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13:533–549, 1986.

[36] Fred Glover. Tabu search– part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[37] Fred Glover. Tabu search– part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.

[38] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.G.H. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[39] John H. Holland. *Adaptation in Natural and Artifical Systems*. University of Michigan Press, 1975.

[40] J.R. Jackson. Scheduling a production line to minimise maximim tardiness. Technical report, Mangement Science Research Projects, University of California, Los Angeles, USA, 1955.

[41] J.R. Jackson. An extension of hohnson's result on job lot scheduling. *Naval Research Logistics Quarterly*, 3(3):201–203, 1956.

[42] J.R. Jackson. Simulation research on job-shop production. *Naval Research Logistics Quarterly*, 4:287–295, 1957.

[43] A. Jones and L.C. Rabelo. Survey of job shop scheduling techniques. NISTIR, National Institute of Standards and Technology, Gaithersburg, MD, 1998.

[44] R. M. Karp. *Reducibility Among Combinatorial Problems*, pages 85–103. Plenum Press, NY, 1972.

[45] S. Kirkpatrick, C. D. Gellatt, and M. P. Vecchi. Simulated annealing. *Science*, 220(671), 1983.

[46] P. Kluvánek. Optimalizácia riadenia dopravných systémov. Správa čiastkovej úlohy, iii-9-6/09, Žilina, 1983.

[47] Shigenobu Kobayashi, Isao Ono, and Masayuki Yamamura. An efficient genetic algorithm for job shop scheduling problems. In Larry J. Eshelman, editor, *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 506–511, San Francisco, July 15–19 1995. Morgan kaufmann Publishers.

[48] P.J.M. Van Laarhoven, E.H.L. Aarts, and J.K. Lenstra. Job shop scheduling by simulated annealing. *Oper. Res.*, 40(1):113–125, 1992.

[49] S.R. Lawrence. Suplement to resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques. *Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA 15213, USA, Oktober*, 1984.

[50] H.R.D Lourenço. *Computational Study of the Job-Shop and Flow-Shop Scheduling Problems, Ph.D. Thesis*. Tr-1060, School of Operations Research & Industrial Engineering, Cornell University, Ithaca, New York 14853-3801, July 1993.

[51] H.R.D. Lourenço. Job-shop. *Internacional Journal of Production Research*, 31(1):59–79, 1993.

[52] H.R.D. Lourenço and M. Zwijnenburg. Combining the large-step optimization with tabu-search: Application to the job-shop scheduling problem. In I.H. Osman and J.P. Kelly, editors, *Metaheuristics: Theory and Applications*, chapter 14, pages 219–236. Kluwer Academic Publishers, Boston, MA, USA, 1996.

[53] B. MacCarthy and J. Liu. Addresing the gap in scheduling research: A review of optimization and heuristic methods in production scheduling. *Internacional Journal of Production Research*, 31(1):59–79, 1993.

[54] Olivier Martin, Steve W. Otto, and Edward W. Felten. Large-step markov chains for the traveling salesman problem. *Complex Systems*, 5(3):299–326, June 1991.

[55] Olivier Martin, Steve W. Otto, and Edward W. Felten. Large-step Markov chains for the TSP incorporating local search heuristics. *Operations Research Letters*, 11(4):219–224, 1992.

[56] H. Matsuo, C.J. Suh, and R.S. Sullivan. A controlled search simulated annealing method for the general job-shop scheduling problem. Working Paper, Graduate School of Business, The University of Texas at Austin, Austin, USA, april 1988.

[57] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physic*, 21(6):219–224, June 1953.

[58] Ryohei Nakano and Takeshi Yamada. Conventional genetic algorithms for job shop problems. In Richard K. Belew and Lashon B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms (ICGA'91)*, pages 474–479, San Mateo, California, 1991. Morgan Kaufmann Publishers.

[59] B. Norman and J. Bean. Random keys genetic algorithm for the job-shop scheduling. *Engineering Design and Automation*, 3:145–156, 1997.

[60] E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job shop problem. *Management Science*, 42(6):797–813, June 1996.

[61] S. Palúch. *Teória rozvrhov.* Žilinská univerzita v Žiline, Fakulta riadenia a informatiky.

[62] Š. Peško. *Optimalizácia NP-ťažkých dopravných rozvrhov.* Habilitačná práca, Žilinská univerzita v Žiline, Fakulta Riadenia a informatiky, február 2002.

[63] M. Pinedo. *Scheduling: Theory Algorithms and Systems.* Prentice Hall, Englewood Cliffs, New Jersey, 1995.

[64] A.J. Rowe and J.R. Jackson. Research problems in production routing and scheduling. *Journal of Industrial Engineering*, 7:116–121, 1956.

[65] B. Roy and B. Sussmann. Les problemes d'ordonnancement avec contraintes disjonctive. *Note D.S. no. 9 bis, SEMA, Paris, France, Decembre*, 1964.

[66] I. Sabuncuoglu and M. Bayiz. Analysis of reactive scheduling problem in a job shop enviroment. *European Journal of Operational Research*, (126):567–586, 2000.

[67] W.E. Smith. Various optimizers for single stage production. *Naval Research Logistics Quarterly*, 3:59–66, 1956.

[68] Y. Sotskov, N.Y. Sotskova, and F. Werner. Stability of an optimal schedule in job shop. *Mgmt Sci*, 25(4):397–414, 1997.

[69] Y.N. Sotskov, T. Tautenhahn, and F. Werner. On the application of insertion techniques for job shop problems with setup times. *RAIRO Rech. Opér.*, 31(2):209–245, 1999.

[70] V. Srinivasan. A hybrid algorithm for the one machine sequencing problem to minimize total tardiness. *Naval Research Logistcs Quarterly*, 18:317–327, 1971.

[71] R.H. Storer, S.D. Wu, and R. Vaccari. New search spaces for sequencing problems with aplications to job-shop scheduling. *Management Science*, 38(10):1495–1509, 1992.

[72] R.H. Storer, S.D. Wu, and R. Vaccari. Problem and heuristic space search strategies for job-shop scheduling. *ORSA Journal on Computing*, 7(4):453–467, 1995.

[73] E. Taillard. Parallel taboo search techniques for job shop scheduling problem. Working Paper ORWP 98/11, Departement de Mathematiques, Ecole Polytechnique Federale De Lausanne, Lausanne, Switzerland, 1989.

[74] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operations Research*, 64:278–285, 1993.

[75] Hisashi Tamaki and Yoshikazu Nishikawa. A paralleled genetic algorithm based on a neighborhood model and its application to the jobshop scheduling. In Reinhard Männer and Bernard Manderick, editors, *Parallel problem solving from nature 2*, pages 573–582, Amsterdam, 1992. North-Holland.

[76] L. Unčovský and kol. *Modely sieťovej analýzy.* ALFA, 1991.

[77] R.J.M. Vaessens. *Generalized Job Shop Scheduling: Complexity and Local Search.* PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 1995.

[78] F. Werner and A. Winkler. Insertion techniques for heuristic solution of the job shop problem. Technical report, Technische Universität Otto von Guericke, Magdeburg Germany, 1992.

[79] T. Yamada and R. Nakano. Job-shop scheduling. In A.M.S. Zalzala and P.J. Fleming, editors, *Genetic algorithms in engineering systems*, 55, chapter 7, pages 134–160. The Institution of Electrical Engineers, 1997.

[80] Takeshi Yamada and Ryohei Nakano. A genetic algorithm applicable to large-scale job-shop problems. In Reinhard Männer and Bernard Manderick, editors, *Parallel problem solving from nature 2*, pages 281–290, Amsterdam, 1992. North-Holland.