

**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta strojní
Ústav automatizace a informatiky**

**ROZVRHOVÁNÍ VÝROBY POMOCÍ METOD
LOKÁLNÍHO HLEDÁNÍ**

**Diplomant : Lubomír Čapatý
Vedoucí : RNDr. Jiří Dvořák, CSc.**

Brno, květen 1998

ANOTACE :

Má práce se zabývá otázkou řešení složitých kombinatorických problémů se zvláštním zřetelem na problémy rozvrhování výroby, tak zvaný **job shop scheduling**. V první části své práce se zabývám historií a principy kombinatoriky jako takové, v druhé části pojmem job shop rozvrhování a samotnými vyhledávacími algoritmy, jako jsou metoda lokálního minima, simulované žhání a tabu search. Ve třetí části se zabývám popisem svého programu a jeho zajímavých částí spolu se zevrubným popisem vývojového prostředí DELPHI, ve kterém jsem celý program napsal. Ve čtvrté části podrobně popisuji zkušební příklady, které jsem použil pro vyzkoušení vlastností mého programu.

ANNOTATION :

Meine Arbeit befasst sich mit Fragen der Lösung von komplizierten kombinatorischen Problemen mit außergewöhnlicher Rücksicht auf Probleme von Einteilung der Produktion, den sogenannten **job shop scheduling**. Im ersten Teil meine Arbeit befasse ich mich mit der Geschichte und den Prinzip des Kombinatoris als solche, im zweiten Teil befasse ich mich mit dem Begriff des job shop einteilens und den allein ausgesuchten Algorithmus als Methoden des lokalen Minimums, simulieren Glühens und Tabu search. Im dritten Teil befasse ich mich mit der Beschreibung meines Programms und seiner interessanten Teile zusammen mit vereinfachter Beschreibung von entwickelter Umwelt DELPHI, in der ich ganze Programme geschrieben habe. Im vierten Teil beschreibe ich ausführlich Test-Beispiele, die ich für die Erprobung der Eigenschaften meines Programms benutzt habe.

ANNOTATION

I solve a problem of combinatorics in branch of production scheduling in my thesis, so - called **job shop scheduling**. In the first part of my thesis I have sketched the history and principles of combinatorics and in the second part I have employed myself in notion of job shop scheduling and searching algorithms, for example the method of local search, simulated annealing and taboo search. In the third part of my thesis I have described the program which has been created by myself in the developmental environment DELPHI. The examples described in the fourth part helped me to verify the correct function and characteristics of the program.

PODĚKOVÁNÍ:

Rád bych poděkoval svému vedoucímu diplomové práce RNDr. Jiřímu Dvořákovi za jeho odborné rady při práci na diplomové práci. Dále bych chtěl poděkovat svému kolegovi Pavlovi Houškovi za rady při tvoření zdrojového kódu a panu ing. Vladimíru Jurutkovi za rady při jeho odladování. Zároveň bych chtěl poděkovat svým spolubydlícím z kolejí, bez jejichž rad bych tuto práci vytvořil daleko dříve.

Lubomír Čapatý

Obsah

1. Úvod	7
1.1 Kombinatorické problémy	7
1.2 Problém NP - úplnosti	7
1.3 Způsoby řešení	7
1.4 Job shop scheduling	7
2. Popis jednotlivých metod	9
2.1 Parametry hledání	9
2.1.1 Sousedství	9
2.1.2 Ukončující podmínky	9
2.1.3 Model úlohy	9
2.2 Metoda lokálního hledání	9
2.3 Simulované žhání	10
2.4 Tabu search	12
2.4.1 Princip metody Tabu Search	12
2.4.2 Krátkodobá paměť	13
2.4.3 Dlouhodobá paměť	13
2.4.4 Aspirační kritéria	13
2.5 Algoritmus Nowicki & Smutnicki	14
2.5.1 Definice problému pro tento algoritmus	14
2.5.2 Sousedství	15
2.5.3 Tabu seznam	17
2.5.4 Strategie prohledávání sousedství	17
2.5.5 Algoritmus Tabu Search	19
3. Kódování pro job shop rozvrhování	20
3.1 Uvedení do problému	20
3.2 Reprezentace založená na operacích (operation - based representation)	21
3.3 Reprezentace založená na pracích (job - based representation)	22
3.4 Reprezentace založená na preferenčním seznamu (preference list - based representation)	23
3.5 Reprezentace založená na vztahu mezi pracemi (job pair relation - based representation)	24
3.6 Reprezentace založená na prioritních pravidlech (priority rule - based representation)	25
3.7 Reprezentace disjunktivním grafem (disjunctive graph-based representation)	26
3.8 Reprezentace časy dokončení (completion time-based representation)	27
3.9 Reprezentace založené na strojích (machine-based representation)	28
3.10 Reprezentace náhodným klíčem (random key representation)	28
3.11 Vyhodnocení	28
3.11.1 Složitost dekodéru	29
3.11.2 Vlastnosti kódovaného prostoru a mapování	29
3.11.3 Paměťové požadavky	29
3.12 Závěr	29
4. Programová část	30
4.1 Popis systému DELPHI	30
4.1.1 Základ systému	30
4.1.2 Prodejní varianty systému Delphi	30

4.1.3	Objektově orientované programování	30
4.1.4	Vizuální komponenty	31
4.1.5	Editace zdrojového kódu	31
4.1.6	Formuláře	31
4.1.7	Object inspector	31
4.1.8	Správa a překlad projektů	32
4.1.9	Database desktop	32
4.1.10	Image editor	32
4.1.11	Pomocné nástroje	32
4.2	Popis programu Plánování výroby	33
4.2.1	Úvodní slovo	33
4.2.2	Konfigurace	33
4.2.3	Obsah pracovního adresáře	33
4.2.4	Ovládání programu	34
4.2.5	Spuštění programu a první obrazovka	34
4.2.6	Popis formuláře STROJE	35
4.2.7	Popis formuláře VÝROBKY	35
4.2.8	Popis formuláře OPERACE	36
4.2.9	Popis formuláře VÝPOČET	36
4.2.10	Popis formuláře PRŮBĚH VÝPOČTU	37
4.2.11	Popis formuláře ZÁLOHA	38
4.2.12	Popis formuláře OBNOVENÍ ZÁLOHY	39
4.3	Popis jednotlivých knihoven a procedur	39
4.3.1	Knihovna HLAVNI	39
4.3.2	Knihovna STROJE	40
4.3.3	Knihovna VYROBKY	40
4.3.4	Knihovna OPERACE	40
4.3.5	Knihovna OBNOVIT	41
4.3.6	Knihovna UVOD	41
4.3.7	Knihovna VKLADANI	41
4.3.8	Knihovna ZALOHA	41
4.3.9	Knihovna ZPRAVA	42
4.3.10	Knihovna VYPOCET	42
4.3.11	Knihovna PRUBEH	42
5.	Ověřování chodu programu pomocí předřešených případů	44
5.1	Úvod	44
5.2	J. Adams, E. Balas and D. Zawack (1988),	44
5.3	J. Adams, E. Balas and D. Zawack (1988)	45
5.4	Fisher and Thompson 1968	45
5.5	Fisher and Thompson 1963	46
5.6	Lawrence 1968	46
5.7	Lawrence 1984	47
5.8	Lawrence 1968	47
5.9	Lawrence 1968	48
5.10	D. Applegate, W. Cook - B. Gamble 1991	49
5.11	D. Applegate, W. Cook - B. Shepherd 1991	49

5.12 D. Applegate, W. Cook - G.Steiner 1991	50
5.13 D. Applegate, W. Cook - B.Cook 1991	51
5.14 Storer, Wu, and Vaccari hard 1992	51
5.15 Storer, Wu, and Vaccari hard 1992	52
6. Závěr	54

1. Úvod

1.1 Kombinatorické problémy

Kombinatorické problémy můžeme posuzovat jako určitý typ vícerozměrné mozaiky. Tyto problémy vytváří velké množství možných řešení, ale standardní matematické metody nejsou použitelné. V každém z těchto problémů se schovává možnost tzv. kombinatorické exploze. Nejznámějším příkladem kombinatorických problémů je známý problém obchodního cestujícího. Jedná se o příklad, ve kterém obchodní cestující musí navštívit určitý počet měst a skončit ve svém bydlišti a zároveň ujet co nejkratší vzdálenost. Z matematického hlediska jednoduchý problém vyzkoušení všech variant v praxi kombinatoricky exploduje, např. pro n navštívených míst je počet kombinací $(n-1)!/2$. Pro n o velikosti 20 při rychlosti jednoho milionu kombinací za sekundu je doba výpočtu několik tisíc let. V průběhu doby se různí autoři snažili této explozi vyhnout, ať dynamickým programováním nebo lokálním prohledáváním.

1.2 Problém NP - úplnosti

Základem této teorie článek S. Cooka v roce 1971. Je dokázáno, že určité přesně definované matematické problémy jsou nerozhodnutelné a tudíž principiálně nemůže existovat algoritmus schopný řešit všechny případy těchto problémů. Cook analyzoval třídy jednotlivých problémů, které se dnes označují jako P a NP. Třída P obsahuje všechny problémy, které lze řešit v reálném čase a třída NP obsahuje všechny problémy, pro které lze navržené řešení prověřit v reálném čase. Například u problému obchodního cestujícího sice nezjistíme, zda námi vytvořené pořadí je nejoptimálnější, ale snadno zjistíme, kolik kilometrů cestující ujede. Dosazováním různých variant do výpočtu můžeme získávat další řešení a hledat optimum.

1.3 Způsoby řešení

Řešení těchto problémů se rozděluje do dvou způsobů, a to jednak exaktní matematické metody, jednak heuristické (náhodné) metody. Je důležité poznamenat, že exaktní metody se při řešení neosvědčily a nepřinášejí vždy optimální výsledky. Naopak u heuristických metod můžeme při vhodném formulaci problému velmi dobrých a vždy optimálních řešení.

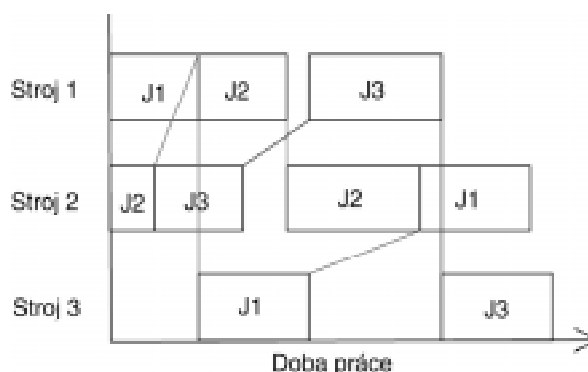
Speciální kapitolou heuristických algoritmů jsou evoluční algoritmy. Tyto algoritmy se inspiroují v procesech v přírodě. Mezi nevýhody těchto algoritmů patří pravděpodobnostní charakter výsledků. Z toho vyplývá po každém spuštění jiný výsledek. Mezi další problémy patří správná reprezentace zadání a následné ohodnocení účelové funkce.

Mezi základní a nejznámější evoluční algoritmy v současné době patří:

- Genetický algoritmus
- Simulované žíhání
- Tabu Search

1.4 Job shop scheduling

Problém rozvrhování (scheduling) je určen množinou strojů, množinou prací (jobů) členěných do sledů jednotlivých operací a stanovením vztahů mezi stroji a operacemi. Řešení tohoto problému spočívá v nalezení nejlepších pořadí prací na zadaných strojích (jedná se o sekvenční problém). V jednodušším případě jsou



Obrázek 1.1

stroje uspořádány sériově a všechny práce přes ně procházejí ve stejném pořadí (tzv. flow shop scheduling). Ve složitějším případě se vyskytuje i paralelní uspořádání strojů (tzv. job shop scheduling). Při řešení mohou být sledovány různé cíle, jako např. minimalizace celkové doby zpracování, minimalizace ztrát spojených s nesplněním prací v požadovaných termínech, minimalizace prostojů, minimalizace rozpracované výroby apod.

Problém rozvrhování patří obecně mezi NP-úplné problémy, což znamená, že přesné optimalizační metody jsou prakticky použitelné pouze pro úlohy omezeného rozsahu. Rozsáhlejší problémy je nutno řešit heuristickými metodami, které sice nezaručují nalezení optimálního řešení, ale jsou schopny v přiměřené době poskytnout řešení, jež je „uspokojivé“. Heuristické metody mohou být postaveny jednak na problémově-specifických principech, vázaných vždy pouze na určitý typ problému, jednak na principech obecnějších. V této práci se budu věnovat specifickým principům rozvrhování výroby.

Deterministické rozvrhování strojů (deterministic machine scheduling) může být matematicky definováno následovně: Máme skupinu m strojů a n prací. Práce n se skládá z množiny operací, které musí být provedeny ve stanovené posloupnosti. Každá operace o_{ij} musí být provedena na určitém stroji j a má dobu provedení t_{ij} , která je deterministicky známá. Naším úkolem je minimalizovat účelovou funkci (C_{max}), což v našem případě znamená poskládat operace, tak aby výsledný čas na stroji, na kterém je nejdelší ze všech, byl co nejmenší. Ukázka takového rozvrhu je na obrázku 1.1.

Rozvrh určuje časové intervaly, ve kterých jsou jednotlivé operace prováděny. Tyto jsou proveditelné pouze tehdy, jestliže vyhovují následujícím omezením. První podmínka je požadavek jedné operace v jeden okamžik na jednom stroji. Druhá podmínka je zachování správné posloupnosti operací na jednotlivých pracích.

2. Popis jednotlivých metod

2.1 Parametry hledání

2.1.1 Sousedství

Při hledání optimálního výsledku je potřeba vytvářet sousedství. Jedná se vlastně o seznam dalších stavů, z nichž jeden bude vybrán pro další postup. Jak je vidno, na způsobu výběru sousedů záleží celý výsledek. Matematicky můžeme definovat sousedství takto: Necht' M je prostor přípustných řešení. Na množině M se definuje určitá relace souslednosti, která pro každé přípustné řešení x umožňuje stanovit jeho jisté okolí $S(x)$ jako množinu přípustných řešení sousedících s x . Např. v případě jednoduchého sekvenčního problému je problém řešení tvořen permutacemi indexů $1, 2, \dots, m$ jednotlivých prací a jeho velikost je tvořena $m!$.

Jednou z možností, jak vytvářet množinu sousedů aktuálního řešení je prohození operací na dvou vybraných pozicích. Všichni sousedé aktuálního řešení jsou pak jednoznačně určeni dvojicemi i a j takovými, že $i < j$. Máme-li např. permutaci $(5, 8, 1, 3, 4, 2, 7, 6)$, pak při volbě pozic $i = 3, j = 6$ dostaneme permutaci $(5, 8, 2, 3, 4, 1, 7, 6)$. Jednoduchým matematickým výpočtem dojdeme k velikosti množiny sousedů jako $m(m-1)/2$. Toto sousedství jsem zahrnul do svého programu. Další sousedství vytvoříme přepsáním vybrané operace i jinou nižší operací j a následným posunem všech operací mezi nimi doleva. Máme-li např. permutaci $(5, 8, 1, 3, 4, 2, 7, 6)$, pak při volbě pozic $i = 3, j = 6$ dostaneme permutaci $(5, 8, 3, 4, 2, 1, 7, 6)$. Toto sousedství vytváří obdobný počet sousedů. V mém programu zahrnuji modifikaci tohoto sousedství, kdy za hodnotu i dosazuji vždy 1. Vytváření sousedství může být libovolné, proto je množství způsobů vytváření sousedů je velmi velké.

2.1.2 Ukončující podmínky

Protože možnost, že by došlo k propočítání všech variant, je omezena celkovou délkou výpočtu, je nasnadě, že výpočet musí být ukončován jinak. Nejjednoduší ukončování je stiskem ukončovací klávesy, dalším způsobem je omezení počtu kroků, doby výpočtu a hledaného času. Všechny tyto způsoby jsem do svého programu zahrnul.

2.1.3 Model úlohy

Modifikace sousedství probíhá u každé metody jiným způsobem, zejména podle druhu zvolené reprezentace a vybrané metody. Model úlohy si můžeme představovat jako určitý druh seznamu, který naplňujeme jednotlivými operacemi. Tyto operace jsou uspořádány specifickým způsobem. Naši úlohu modelujeme v paměti počítače pomocí objektu TList (v doslovném překladu seznam).

2.2 Metoda lokálního hledání

Tato metoda patří mezi nejstarší metody prohledávání stavového prostoru. Algoritmus lokálního hledání lze popsat následujícím způsobem. Nejprve se vybere počáteční řešení a pak se v každé další iteraci z množiny sousedů aktuálního řešení vybírá nějaké další řešení, kterým se nahrazuje řešení dosavadní. Výběr dalšího řešení z množiny $S(x)$ se děje obvykle pomocí metody nejstrmějšího nebo náhodného spádu. V prvním případě se prohledává celá množina sousedů aktuálního řešení a vybírá se řešení s nejnižší hodnotou účelové funkce.

Ve druhém případě se v náhodném pořadí zkoumají sousedé aktuálního řešení a vybírá se prvé řešení s nižší hodnotou účelové funkce. Tyto kroky probíhají do ukončení výpočtu.

Z popisu metody lokálního hledání je zřejmé, že získané řešení obvykle není globálně optimální, ale spíše jen lokálně optimální (a to ještě vzhledem k definování sousednosti). Existují různé pokusy o překonání tohoto nedostatku, jako např. zvětšení množiny sousedů nebo opakované startování celé procedury z různých náhodně volených počátečních řešení. Větší naději na úspěch než uvedené příklady dává myšlenka, že aby postup neuvázl v lokálním optimu, je nutno netrvat nadále na strategii postupně se zlepšujících řešení a určitým úsporným a řízeným způsobem připustit i kroky, po nichž dojde ke zhoršení hodnoty účelové funkce. Na této myšlence jsou založeny některé moderní heuristické metody, jako např. simulované žíhání a tabu search. Jiný přístup představují genetické algoritmy, které pracují s populací řešení určitého rozsahu a mění ji v analogii s vývojem genetických struktur.

2.3 Simulované žíhání

Teorii simulovaného žíhání podrobně popsal Reeves [2]. Koncepce metody simulovaného žíhání je založena na analogii mezi simulací fyzikálního procesu žíhání a procesem řešení optimalizačního problému. Jestliže je pevný materiál zahřát na teplotu vyšší než jeho bod tání a následovně ochlazen zpět do pevného stavu, pak strukturální vlastnosti zchlazeného materiálu závisí na rychlosti chlazení. Při simulaci je na zkoumaný materiál pohlíženo jako na systém částic. Postupně jsou generovány stavy systému a počítají se odpovídající změny energie δE . Při poklesu energie se přejde do nového stavu, zatímco při růstu energie je nový stav akceptován s pravděpodobností P ,

$$P(\delta E) = \exp\left(\frac{-\delta E}{k_B t}\right) \quad (1)$$

kde k_B je fyzikální konstanta známá jako Boltzmanova konstanta. Inspirací jsou zde zákony termodynamiky, podle nichž je při teplotě t pravděpodobnost růstu energie o δE dána vztahem (1). Proces probíhá tak dlouho, dokud není dosaženo stabilního “zmrazeného” stavu.

Analogie mezi fyzikálním systémem a optimalizačním modelem je založena na následujících ekvivalentech:

termodynamická simulace	kombinatorická optimalizace
stav systému	přípustné řešení
energie	hodnota účelové funkce
změna stavu	přechod k sousednímu řešení
teplota	řídící parametr
zmrazený stav	heuristické řešení

Navržený přístup může být považován za variantu výše zmíněného lokálního hledání, v níž se množina sousedů aktuálního řešení náhodným způsobem prohledává, přičemž řešení způsobující vzestup hodnoty účelové funkce jsou akceptována v závislosti na hodnotě výrazu $\exp(\delta / t)$, kde δ je přírůstek účelové funkce a t řídící parametr.

Kostra algoritmu simulovaného žíhání pro řešení problému sestává z následujících kroků:

Vyber počáteční řešení x_0 ;

Vyber počáteční hodnotu t ; { t označuje “teplotu”}

Vyber počáteční hodnotu n ; { n označuje délku vnitřního cyklu}

$x^* := x_0$;

```
repeat
  for i := 1 to n do begin
    Generuj  $x_1$  z  $S(x_0)$ ; {náhodné generování přípustného řešení
                        z množiny sousedů aktuálního řešení}
     $\delta := f(x_1) - f(x_0)$ ;
    if  $\delta < 0$  then {pohyb do lepšího řešení je akceptován} begin
       $x_0 := x_1$ ;
      if  $f(x_0) < f(x^*)$  then  $x^* := x_0$ ;
    end
    else
      if  $\exp(\delta/t) > \text{random}[0,1)$  then  $x_0 := x_1$ ;
      {pohyb do horšího řešení je akceptován s pravděpodobností
       danou výrazem  $\exp(-d/t)$ ; test je implementován pomocí
       náhodného čísla generovaného z rovnoměrného rozdělení
       na intervalu  $[0,1]$ }
    end ;
     $t := a(t)$ ; { $a(t)$  je funkce redukce teploty}
     $n := g(t)$ ; { $g(t)$  je funkce změny délky cyklu}
  until Podmínka_ukončení; {řešení  $x^*$  je aproximací optimálního
                           řešení}
```

Při konkretizaci uvedeného postupu na určitý typ problému je nutno učinit řadu rozhodnutí, která lze rozdělit na generická a problémově specifikovaná. Rozhodnutí obou typů musejí být dělána opatrně, neboť mají vliv na rychlost algoritmu a na kvalitu získaného řešení. V literatuře lze nalézt řadu doporučení, založených na teoretických i praktických výsledcích. Téměř všechny úspěšné aplikace zmiňované v literatuře ukazují, že tato rozhodnutí byla učiněna až po mnoha experimentech.

Generická rozhodnutí zahrnují volbu parametrů samotného žihacího algoritmu. Jde zejména o počáteční teplotu, počáteční délku cyklu, plán chlazení (cooling schedule) určený způsobem změn teploty a délky cyklu. Jestliže má být konečné řešení nezávislé na volbě výchozího řešení, musí být počáteční teplota tak vysoká, aby dovolila téměř volnou výměnu sousedních řešení. Pro redukci teploty je často používána geometrická redukční funkce $a(t) = at$, kde $0 < a < 1$ (zkušenosti ukazují, že nejlepší jsou hodnoty mezi 0,8 a 0,99). Délka cyklu pro určitou teplotu je obvykle ve vztahu k velikosti množiny sousedů a může se při každé nové teplotě zvyšovat buď geometricky, tj. $g(n) = bn$, ($b > 1$), nebo aritmeticky, tj. $g(n) = n + c$, ($c > 0$).

Problémově specifická rozhodnutí se týkají prostoru řešení, struktury množin sousedů a účelové funkce. Prostor řešení by měl být udržován co nejmenší. Při stanovení struktury množin sousedních řešení je důležitá podmínka dosažitelnosti, požadující, aby každé řešení bylo dosažitelné z libovolného jiného řešení postupnou aplikací vztahu sousednosti. Náhodné generování sousedního přípustného řešení může být obtížné, když množina sousedů je rozsáhlá a složitá, nebo když prostor řešení je omezen přísnými podmínkami přípustnosti. Množinu sousedů je tedy třeba definovat tak, aby měla jednoduchou strukturu a únosný rozsah. Problém přísných omezení může být překonán uvolněním podmínek přípustnosti a rozšířením účelové funkce o člen penalizující porušení původních podmínek. Účelová funkce i struktura množiny sousedů by měly být konstruovány tak, aby výpočet změny hodnoty účelové funkce proběhl rychle a efektivně. To je možné např. v případě, kdy není nutné pro nové řešení počítat celou účelovou funkci.

2.4 Tabu search

2.4.1 Princip metody Tabu Search

Tabu search (dále jen TS) je metaheuristické hledání navržené k nalezení téměř optimálního řešení kombinatoricko-optimalizačního problému. Tato metoda byla navržena původně Gloverem a dalšími v roce 1975 a další upřesnění a vývoj této metody byl proveden opět Gloverem v letech 1986 -90. Dnes se stručně popisuje následovně. Na začátku je základní pojem, který nazýváme pohyb. Tento pojem je definován jako funkce, která transformuje řešení na jiné řešení. Pro všechna řešení je definována množina pohybů směřující k jiným řešením. Tato množina vytváří podmnožinu řešení, kterou nazýváme sousedství, podrobněji se sousedstvím zabýváme na začátku této kapitoly. TS začíná z počátečního řešení a v každém kroku je v sousedství daného řešení hledáno další řešení s cílem najít souseda, většinou toho nejlepšího. Pohyb, který nás vede k nejlepšímu sousedovi je vykonán a poté nově získané řešení je stanoveno jako základní pro další krok. Abychom zabránili cyklování, a vedli vyhledávání k tzv. „správnému“ řešení v oblasti možných řešení, je historie vyhledávání uchována v paměti a použita při dalším vyhledávání. Jsou zde přinejmenším dvě třídy paměti, a to krátkodobá paměť pro nedávné pohyby a dlouhodobá paměť pro starší pohyby a jejich atributy. Mezi mnoho paměťových struktur používaných v TS patří třída krátkodobých pamětí, která je nazývána tabu seznam a hraje základní roli. Tento seznam neumožňuje návrat na řešení, navštívená předchozích *maxt* krocích, kde *maxt* je zadané číslo. V praxi tento seznam uchovává zakázané pohyby a jejich atributy nebo atributy zakázaných řešení spíše než zakázaná řešení. Když je pohyb z řešení do jeho souseda vykonán, uložíme opačnou cestu na konec seznamu *H* a odstraníme první prvek z tohoto seznamu. Samozřejmě se může stát, že zajímavým krokem vedoucím k lepšímu řešení je zakázaný krok. Abychom tento krok mohli vykonat, musí být definována tzv. aspirační funkce. Pravidla pro ukončování této metody jsou podrobně definována na začátku této kapitoly. Další rozvinutější a komplikovanější implementace TS metody jsou doporučovány pro náročné optimalizační problémy a jsou diskutovány podrobně v dokumentech pana Glovera (1990) a Glover a Laguny (1993).

Mělo by být poznamenáno, že jakkoli je implementace metody TS problémově orientovaná, potřebuje partikulární definice strukturovaných elementů, jako jsou pohyby sousedství, struktura paměti, aspirační funkce strategie vyhledávání sousedství, STOP pravidla, počáteční řešení, hodnoty mnoha ladících parametrů jako *maxt*, *maxitr* a stupeň aspirace *a*. Závěrečné numerické vlastnosti algoritmu (provedení, rychlost konvergence, doba běhu) tak leží, jak na strukturovaných elementech, tak na ladících parametrech.

V algoritmu TS je sousedství aktuálního řešení $S(x)$ nahrazeno modifikovaným sousedstvím $S_H(x)$. Z hlediska krátkodobých strategií do něj nepatří ty prvky $S(x)$, které jsou dosažitelné pouze pohyby náležejícími do zakázané (tabu) množiny pohybů. S ohledem na dlouhodobé strategie může $S_H(x)$ obsahovat naopak prvky, které se v $S(x)$ nevyskytují (zejména se jedná o výborná lokální optima, získaná v předchozím procesu řešení). Tabu search užívá paměť také k modifikaci vyhodnocování přístupných řešení. To znamená, že účelová funkce $f(x)$ je nahrazena funkcí $f_H(x)$, která je vytvořena z $f(x)$ doplněním penalizačních nebo motivačních členů (příkladem může být penalizace řešení přímo úměrná frekvenci jeho dosavadního výskytu). Modifikace účelové funkce také může často souviset s uvolněním omezujících podmínek problému a penalizací těch řešení, která tyto podmínky nesplňují.

Kostra tabu search může být popsána následovně:

Vyber počáteční řešení $x_0 \in M$;

$x^* := x_0$;

$H := \{H \text{ představuje historii dosavadního postupu}\}$;

repeat

```
 $x_1 := \text{prvek } S_H(x_0) \text{ s nejmenší hodnotou } f_H(x);$   
if  $f(x_1) < f(x^*)$  then  $x^* := x_1;$   
 $x_0 := x_1;$   
 $H := \text{aktualizace } H \text{ \{druh aktualizace závisí na velikosti maxt\}};$   
until Podmínka_ukončení; \{obvykle dosažení počtu iterací maxitr\}  
\{Bod  $x^*$  je aproximací optimálního řešení\}
```

S každým pohybem z aktuálního řešení může být spojena řada atributů. Atributem může být jakýkoli aspekt daného problému, který se v důsledku pohybu mění. V případě sekvenčního problému lze jako atributy pohybu použít dvojice indexů prací, jejichž výměnou v permutaci se pohyb uskutečnil, indexy jednotlivých prací, vztahy mezi indexy prací a jejich pozicemi, vztahy mezi bezprostředními předchůdci a následníky apod.

Atributy pohybu jsou v tabu search využívány k tvorbě tzv. tabu omezení. Ve snaze zabránit návratům k předchozím řešením se jako tabu klasifikují atributy “inverzní” k atributům každého provedeného pohybu.

2.4.2 Krátkodobá paměť

Protože paměť počítače má jen omezenou velikost, tabu klasifikace se pro každý takto označený atribut udržuje pouze po t následujících iterací. Tuto paměť nazýváme krátkodobou pamětí. Klíčovou otázkou u krátkodobé paměti je určení “dobré” doby trvání zákazu. Pravidla pro určení této doby mohou být tříděna podle možnosti měnit počet iterací $maxt$, a to na statická nebo dynamická. Statická pravidla vybírají hodnotu $maxt$, která zůstává fixována po celou dobu hledání, kdežto dynamická pravidla dovolují hodnotu t měnit. Praktické zkušenosti ukazují, že dynamická pravidla jsou obvykle robustnější než statická. Často je také vhodné připustit, aby různé typy atributů definujících tabu omezení měly různé hodnoty $maxt$.

2.4.3 Dlouhodobá paměť

Dlouhodobá paměť obsahuje informace o frekvenci výskytu atributů v dosavadním procesu hledání a rozšiřuje tak základ pro výběr přípustných pohybů. Tyto informace se obvykle používají tak, že se s jejich pomocí určují motivační nebo penalizační hodnoty, kterými se modifikují hodnoty pohybů. Důležitou roli hraje dlouhodobá paměť zejména v procesech intenzifikace a diverzifikace. Intenzifikační strategie se při hledání řešení zaměřují na podporu “dobrých” atributů. Diverzifikační strategie namísto toho generují řešení zahrnující atributy významně odlišné od těch, které se vyskytují v procesu dosavadního hledání.

2.4.4 Aspirační kritéria

Při implementaci tabu omezení musí být vzata v úvahu jedna důležitá výjimka, a sice že tabu omezení není neporušitelné za všech okolností. Kdyby např. zakázaný pohyb vedl k řešení lepšímu, než byla všechna dosud dosažená, může být jeho tabu klasifikace ignorována. Podmínka, která dovoluje ignorování tabu omezení, se nazývá aspirační kritérium. Vhodné užití takových kritérií může významně přispět ke zdokonalení procesu hledání.

2.5 Algoritmus Nowicki & Smutnicki

Závěrem bych zde rád upozornil na asi nejmodernější implementaci heuristické metody taboo search na job shop rozvrhování. Podrobnější popis je uveden v dokumentech [5]. Objevitelé metody pochází z Polska a pracují na Technical University of Wrocław. Metoda používá speciální kódování disjunktivním grafem, které bude popsáno v další kapitole. Tato metoda dosahuje nejlepších výsledků a je velkým příslibem do budoucna.

2.5.1 Definice problému pro tento algoritmus

Máme množinu jobů $J = 1, \dots, n$, množinu strojů $M = 1, \dots, m$ a množinu operací $O = 1, \dots, o$. Množina O se skládá z jednotlivých podmnožin operací pro každý job. Job j se skládá z sekvence operací o_j indexovaných postupně jako $(l_{j-1} + 1, \dots, l_{j-1} + o_j)$, které musí být prováděny v tomto pořadí a kde $l_j = \sum_{i=1}^j o_i$ zatím co celkový počet operací na prvním jobu j je $1, \dots, n$, ($l_0 = 0$), a $\sum_{i=1}^n o_j = o$. Operace i musí být prováděna na stroji $\mu_i \in M$ během nepřerušného výrobního času $\tau_i > 0, i \in O$. Předpokládáme, že kterékoli po sobě jdoucí operace na stejném jobu budou prováděny na rozdílných strojích. Předpoklad je učiněn pouze pro jednoduchost popisu a není nutný. Dvě po sobě jdoucí operace na stejném jobu prováděné na stejném stroji můžou být rozděleny fixními operacemi prováděnými na fixních strojích během nekonečného procesního času.

Každý stroj může ve stejném okamžiku vykonávat pouze jednu operaci. Odpovídající rozvrh je definován počátečním časem $S_i \geq 0, i \in O$. Problém spočívá v nalezení proveditelné minimalizované účelové funkce $\max_{i \in O} (S_i + \tau_i)$. Je vhodné pro tento problém, aby byl reprezentován pomocí disjunktivního grafu. Nejprve je možné poznat, že množina operací O může být přirozeně rozdělena do podmnožin $M_k = \{i \in O : \mu_i = k\}$, kde každá z nich odpovídá množině operací, které jsou být prováděny na stroji K , a $m_k = |M_k|, k \in M$. Procesní pořadí operací na stroji k je definováno jako permutace $\pi_k = (\pi_k(1), \dots, \pi_k(m_k))$ na $M_k, k \in M$; $\pi_k(i)$ udává element M_k , který je v pozici i v π_k . Necht' Π_k je množina všech permutací M_k . Výrobní rozvrh operací na stroji je definován m -tuple $\pi = (\pi_1, \dots, \pi_m)$, kde π náleží do $\Pi_k = \Pi_{k1} \times \Pi_{k2} \times \dots \times \Pi_{km}$. Pro výrobní rozvrh π_k můžeme vytvořit disjunktivní graf $G(\pi_k) = (O, R \cup E(\pi_k))$ s množinou uzlů O a množinou hran $R \cup E(\pi_k)$ kde :

$$R = \bigcup_{j=1}^n \bigcup_{i=1}^{o_j-1} \{(l_{j-1} + i, l_{j-1} + i + 1)\}$$

$$U = \bigcup_{k=1}^m \bigcup_{i=1}^{m_k-1} \{(\pi_k(i), \pi_k(i+1))\}$$

Hrany z množiny R reprezentují výrobní rozvrh operací na jobech, a kde hrany z množiny $E(\pi)$ reprezentují výrobní rozvrh operací na strojích. Každý uzel $i \in O$ v disjunktivním grafu má váhu τ_i a každá hrana má váhu 0. Všimněte si, že každý uzel $G(\pi)$ má nanejvýš dva přímé sousedy následníky a nanejvýš dva přímé

sousedy předchůdce. Výrobní postup π je proveditelný pouze tehdy, když graf $G(\pi)$ neobsahuje žádný cyklus. Je dobře známo, že jakýkoli proveditelný výrobní rozvrh π vytváří proveditelný konečný rozvrh S_i , $i \in O$. Navíc startovací čas S_i se rovná délce nejdelší cesty přicházející do vrcholu i (ale bez τ_i) v $G(\pi)$, $i \in O$; když uzel nemá předchůdce, potom je jeho délka nula. Účelová funkce $C_{\max}(\pi)$ pro výrobní rozvrh π se rovná délce nejdelší cesty (kritické cesty) v $G(\pi)$. Nyní můžeme znovu formulovat job shop problém jako hledání proveditelného výrobního rozvrhu π ležícího v Π který minimalizuje $C_{\max}(\pi)$.

Daná kritická cesta v $G(\pi)$ z $u = (u_1, \dots, u_w)$, kde $u_i \in O$, $1 \leq i \leq w$ a w je počet uzlů v této cestě. Kritická cesta u závisí na π , ale pro jednoduchost při zápisu ji nebudeme uvádět explicitně. Kritická cesta je rozdělena do podsekvencí B_1, \dots, B_r nazývaných bloky v π na u (podrobněji v literatuře Grabowski et al. 1986), kde :

$$(1) B_j = (u_{a_j}, u_{a_{j+1}}, \dots, u_{b_j}), j = 1, \dots, r$$

$$a_1 = 1 \leq a_1 \leq b_1 < b_1 + 1 = a_2 \leq b_2 < b_2 + 1 = a_3 \dots a_r \leq b_r = w$$

$$(2) B_j = \text{obsahuje operace prováděné na nějakém stroji } \mu(B_j) = \mu_{u_{a_j}}, j = 1, \dots, r$$

$$(3) \text{ dva po sobě jdoucí bloky obsahují operace prováděné na rozdílných strojích jako například } \mu(B_j) \neq \mu(B_{j+1}), j = 1, \dots, r-1.$$

Jednoduše můžeme říct, že bloky jsou maximální podskupiny u , které obsahují operace prováděné na stejném stroji. Pro ilustraci uvádíme následující příklad.

Příklad : Následující problém řeší tři joby, dvanáct operací a dva stroje, $n = 3$, $m = 2$, $o = 12$. Job 1 spočívá ze sekvence sedmi operací (1, 2, 3, 4, 5, 6, 7), job 2 spočívá ze sekvence tří operací (8, 9, 10), a job 3 spočívá ze sekvence dvou operací (11, 12). Operace (1, 3, 5, 7, 8, 10, 11) musí být prováděny na stroji 1 a mají výrobní čas $t = 1$ zatímco operace (2, 4, 6, 9, 12) musí být prováděny na stroji 1 a mají výrobní čas $t = 2$. Uvažujeme jistý proveditelný postup $\pi = (\pi_1, \pi_2)$, kde $\pi_1 = (1, 8, 3, 11, 5, 10, 7)$ a $\pi_2 = (2, 4, 9, 12, 6)$. $G(\pi)$ obsahuje jednoduchou kritickou cestu $u = (1, 8, 3, 4, 9, 12, 6, 7)$, $w = 8$. Tato cesta je rozdělena do $r = 3$ bloků $B_1 = (1, 8, 3)$, $B_2 = (4, 9, 12, 6)$, $B_3 = (7)$ a $\mu(B_1) = \mu(B_3) = 1$, $\mu(B_2) = 2$, $a_1 = 1$, $b_1 = 3$, $a_2 = 4$, $b_2 = 7$, $a_3 = 8$, $b_3 = 8$.

2.5.2 Sousedství

Existují mnohé definice pohybů navržené většinou na vzájemné výměně sousedních a nesousedních párů operací na stroji. Van Laarhoven et. al (1992) a Taillard (1989) definovali pohyb v pro variantu pro výrobní rozvrh π dvojicí operací (x, y) , které jsou 1) po sobě jdoucí na některém ze strojů a 2) po sobě jdoucí na některé kritické cestě na $G(\pi)$. Poté definují sousedství jako variantu získanou z π aplikacemi všech takovýchto pohybů. Velikost takového sousedství závisí na počtu kritických cest v π , a na počtu operací každé kritické cesty, může se jednat relativně o velké číslo. Je zřejmé, že čas procesoru kterékoli implementace závisí v principu na výpočtové složitosti vyhledávání jednoho sousedství (a tím pádem na velikosti sousedství). Taillard (1989) redukoval množství kalkulací pro rozsáhlé sousedství aplikováním spodní hranice na místo kalkulací pracovního rozsahu implicitně pro výběr nejlepšího souseda. Na místo tohoto přiblížení doporučují redukovat velikost sousedství (založeno na definici sousedství Matsuo 1988). Hlavní myšlenka této redukce spočívá v odstranění některých pohybů z množiny provedené Van Laarhovenem (1992), pro které je známo předem (bez výpočtu hodnoty účelové funkce), že nemohou okamžitě zlepšit $C_{\max}(\pi)$. Budeme uvažovat množinu pohybů nazývanou jako vzájemná výměna blízko hranice bloků jediné kritické

cestě, přesněji budeme uvažovat pouze jedinou libovolně zvolenou kritickou cestu u v $G(\pi)$ a bloky B_1, \dots, B_r definované pro u .

Vyměníme první dvě (a poslední dvě) operace v každém bloku B_2, \dots, B_{r-1} a to v každém bloku, který se skládá nejméně ze dvou operací. V prvním bloku B_1 vyměníme poslední dvě operace a analogicky v posledním bloku vyměníme pouze první dvě. Potom formálně definujeme množinu pohybů z π jako $V(\pi) = \bigcup_{j=1}^r V_j(\pi)$, kde :

$$V_1(\pi) = \begin{cases} \{(u_{b_1-1}, u_{b_1})\} & \text{jestliže } a_1 < b_1 \quad a \quad r > 1 \\ \emptyset & \text{jinak} \end{cases}$$

$$V_j(\pi) = \begin{cases} \{(u_{a_j}, u_{a_{j+1}}), (u_{b_{j-1}}, u_{b_j})\} & \text{jestliže } a_j < b_j \\ \emptyset & \text{jinak} \end{cases}$$

$$V_r(\pi) = \begin{cases} \{(u_{a_r}, u_{a_{r+1}})\} & \text{jestliže } a_r < b_r \quad a \quad r > 1 \\ \emptyset & \text{jinak} \end{cases}$$

Množina pohybů není prázdná pouze tehdy, když počet bloků je větší jak jedna ($r > 1$) a když zde existuje přinejmenším jeden blok s počtem elementů větším než jedna. Necht' $Q(\pi, v) \in \Pi$ udává výrobní rozvrh získaný aplikováním pohybu v na výrobní rozvrh π . Sousedství $H(\pi)$ z π je definováno jako všechny výrobní rozvrhy získané aplikováním pohybů $V(\pi)$,

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

Pro příklad, ukázaný předchozí kapitole, předvedeme jednoduchý pohyb $(8, 3)$ v bloku B_1 dva pohyby $(4, 9)$, $(12, 6)$ v bloku B_2 , a žádné pohyby v bloku B_3 , tzn. $V_1(\pi) = \{(8, 3)\}$, $V_2(\pi) = \{(4, 9), (12, 6)\}$, $V_3(\pi) = \emptyset$ a $V(\pi) = \{(8, 3), (4, 9), (12, 6)\}$. V tomto případě sousedství tvoří tři nové výrobní rozvrhy získané aplikováním odpovídajících pohybů na výrobní rozvrh π .

$$H(\pi) = \{Q(\pi, (8,3)), Q(\pi, (4,9)), Q(\pi, (12,6))\}$$

kde

$$Q(\pi, (8, 3)) = ((1, 3, 8, 11, 5, 10, 7), (2, 4, 9, 12, 6)),$$

$$Q(\pi, (4, 9)) = ((1, 8, 3, 11, 5, 10, 7), (2, 9, 4, 12, 6)),$$

a

$$Q(\pi, (12, 6)) = ((1, 8, 3, 11, 5, 10, 7), (2, 4, 9, 6, 12)).$$

Analyzujeme vlastnosti sousedství vytvořených výměnou sousedních operací v proveditelném výrobním rozvrhu π . Udává pomocí $H''(\pi)$ množinu výrobních rozvrhů, kde každý z nich byl získán z π výměnou

párů po sobě jdoucích operací na stroji. $H''(\pi)$ obsahuje přesně $\sum_{k=1}^m (m_k - 1)$ výrobních rozvrhů (některé z nich mohou být neproveditelné). Necht' $H'(\pi)$ je sousedství použité Laarhovenem (viz. začátek této kapitoly).

Definice : $H(\pi)$ je podmnožinou $H'(\pi)$ a $H'(\pi)$ je podmnožinou $H''(\pi)$. Předpokládejme, že $G(\pi)$ má pouze jedinou kritickou cestu u , pak $H'(\pi)$ obsahuje $\sum_{k=1}^r (|B_k| - 1)$ sousedů, a H obsahuje

$$\min\{1, |B_1| - 1\} + \sum_{k=2}^{r-1} \min\{2, |B_k| - 1\} + \min\{1, |B_r| - 1\}$$

sousedů, kde $|B_k| = b_k - a_k + 1$ je hodnotou k -tého bloku. Jestliže $G(\pi)$ má více kritických cest, pak velikost $H'(\pi)$ narůstá, zatímco velikost $H(\pi)$ bude zůstávat stále stejná. Lze říct, že velikost $H(\pi)$ je podstatně menší než $H'(\pi)$ v případě, že máme více dlouhých bloků nebo více kritických cest. Navíc, jestliže máme pouze jediný blok v u ($r = 1$) pak $H'(\pi)$ má přinejmenším $|B_1| - 1 = w - 1$ sousedů (jestliže $w \geq 2$), zatímco $H(\pi)$ je nulové.

Jak bylo ukázáno, pro jakékoliv proveditelné $\pi^0 \in \Pi$ s konečnou sekvencí $\pi^0, \pi^1, \dots, \pi^k$ existuje takové π^k , které je optimálním výrobním rozvrhem a π^{i+1} ležící v $H'(\pi)$, $i = 0, \dots, k-1$ (související vlastnosti). Navíc $H'(\pi)$ obsahuje pouze proveditelné výrobní rozvrhy a pro každé $\alpha \in H'(\pi) \setminus H(\pi)$, je α neproveditelné nebo $C_{\max}(\alpha) \geq C_{\max}(\pi)$. Jinými slovy, takové výrobní rozvrhy, které nevedou k $H'(\pi)$ jsou „méně zajímavé“ z pohledu zlepšování výsledku. Poněvadž $H(\pi)$ je podmnožinou $H'(\pi)$, pak i ona obsahuje pouze proveditelné výrobní rozvrhy. Nyní si ukážeme, že výrobní rozvrhy, které nevedou k $H(\pi)$ (ale můžou patřit mezi $H'(\pi)$), patří také mezi „neslibná“ řešení pro další hledání.

TEORÉM : Pro každý výrobní rozvrh platí $\alpha \in H'(\pi) \setminus H(\pi)$, $C_{\max}(\alpha) \geq C_{\max}(\pi)$

Nabízené sousedství $H(\pi)$ nám umožňuje formulovat jisté dostatečné podmínky pro optimalizaci výrobního rozvrhu.

VLASTNOST : Jestliže $V(\pi)$ je prázdná množina, pak je π optimální výrobní rozvrh.

Všimněte si, že $V(\pi) = \emptyset$, jestliže všechny operace z u jsou prováděny na stejném stroji nebo všechny operace z u náleží k stejnému jobu. Experimentální pravidla z prvního odstavce ukazují, že použití této vlastnosti zastaví vyhledávání v 20 případech ze sta.

Závěrem je jasné, že tyto dvě vlastnosti neplatí pro $H(\pi)$.

2.5.3 Tabu seznam

Nechť $T = (T_1, \dots, T_{\max})$ je tabu seznam fixní délky \max , kde $T_j \in O \times O$ je zakázaný pohyb, a platí, že $1 \leq j \leq \max$. Tabu seznam je inicializován s nulovým elementem $T_j = (0, 0)$, $1 \leq j \leq \max$. Pohyb $v = (x, y)$ je přidán do tabu seznamu T (zavádíme operátor $T \oplus v$) následujícím standartním způsobem : posuneme tabu seznam T doleva a vložíme v na pozici \max (nastavujeme $T_j = T_{j+1}$, $j = 1, \dots, \max - 1$ a $T_{\max} = v$). Taková to forma T je použita pouze pro jednoduchost záznamu. V praxi je T uveden jako cirkulární seznam, který aplikuje operátor \oplus , posunuje pouze ukazatel na poslední záznam a může být vykonáván v $O(1)$ čase (tzn. přepisuje se jen jeden záznam).

Pro každý pohyb $v = (x, y)$ existuje pohyb $\bar{v} = (y, x)$, který nazýváme inverzní pohyb. Jestliže byl vykonán pouze pohyb v , potom pohyb \bar{v} bude zakázán a přidán do T .

2.5.4 Strategie prohledávání sousedství

Nechť je zadáno π , $V(\pi)$, $H(\pi)$, a C^* , počáteční výrobní rozvrh, množina pohybů, sousedství, a nejlepší známá hodnota účelové funkce. Předpokládejme, že množina pohybů $V(\pi)$ není prázdná. Klasifikujeme tyto pohyby do tří kategorií : nezakázané (U), zakázané, ale užitečné (FP) a zakázané a neúžitečné (FN). Pohyby z množiny $V(\pi) \setminus T$ jsou U -pohyby, kdežto ty z množiny $V(\pi) \cap T$ jsou zakázané. Zakázané pohyby

jsou užitečné, jestliže vedou do prostoru s hodnotou účelové funkce kraší než C^* . To jest FP -pohyby jsou definovány jako množina

$$A = \{v \in V(\pi) \cap T : C_{MAX}(Q(\pi, v)) < C^*\}$$

Zbývající pohyby jako z množiny $V(\pi) \cap T \setminus A$ jsou FN -pohyby. Je přirozené, že aby byl pohyb vybrán, měl by pocházet z množiny U a FP -pohybů. Pohyb vytvářející minimální hodnotu účelové funkce, je vykonán jen jednou (dělati bychom stále to stejné). V případě, že nemáme U -pohyby ani FP -pohyby, problém výběru je pojednáván zjednodušeně (výběr FN -pohybů je náhodný nebo se provede vymazání celého T). Toto bylo diskutováno detailně v originálním TS dokumentu Gloverem 1989. Všimněte si, že sousedství použité v tomto algoritmu je podstatně menší než sousedství definované ostatními autory. Proto situace, ve kterých jsou dostupné pouze FN -pohyby se objevují mnohem častěji než v jiných algoritmech. V tomto kontextu je strategie výběru FN -pohybů důležitá. Navíc poslední studie potvrdily, že pro jakékoli obtížné problémy může TS implementace obsahovat příslušné pravidlo pro vytváření takového výběru.

V naší TS implementaci nabízíme výběr nejstaršího pohybu (podobně jako Glover 1989) a navíc provádíme speciální modifikaci TS seznamu založeného na replikaci nejmladšího pohybu. Přesněji, jestliže $V(\pi)$ obsahuje jediný pohyb, pak nemáme problémy s výběrem - tento pohyb je automaticky vybrán. V jiných případech je problém výběru je řešen modifikací tabu seznamu : opakuj $T = T \oplus T_{max}$ dokud $V(\pi) \setminus T \neq \emptyset$ (toto může být prováděno v $O(maxt)$ čase). Poté množina $V(\pi) \setminus T$ obsahuje přesný a jediný pohyb, který může být vybrán jako další. Můžeme tedy říci, že tato modifikace T je forma dynamického řízení délky tabu seznamu. Pro ilustraci zvažte množinu pohybů $V(\pi) = \{(4, 5)(1, 2)\}$ obsahující pouze FN -pohyby a taboo seznam :

$$T = \{(2, 3)(1, 2)(6, 5)(4, 5)(2, 7)\}$$

délky $maxt = 5$. Výše uvedené pravidlo vybere pohyb $v' = (1, 2)$ a modifikovaný tabu list bude vypadat takto :

$$T = \{(6, 5)(4, 5)(2, 7)(2, 7)(2, 7)\}$$

Nakonec bude inverzní pohyb $v' = (2, 1)$ přidán do T , který vytvoří finální podobu taboo seznamu.

$$T' = T \oplus \bar{v}' = \{(4, 5)(2, 7)(2, 7)(2, 7)(2, 1)\}$$

S uvažováním předchozího jsme v podstatě formulovali následující proceduru vyhledávání sousedství (NSP). NSP začíná z výrobního rozvrhu π , neprázdným seznamem $V(\pi)$, současným tabu seznamem T a nejlepší hodnotou účelové funkce C^* . Procedura vrací pohyb v' , nový výrobní rozvrh π' a modifikovaný tabu seznam T' .

Krok 1 : Najděte množinu FP -pohybů

$$A = \{v \in V(\pi) \cap T : C_{MAX}(Q(\pi, v)) < C^*\}$$

Jestliže $(V(\pi) \setminus T) \cup A \neq \emptyset$ pak vyber $v' \in (V(\pi) \setminus T) \cup A$ tak, že :

$$C_{MAX}(Q(\pi, v')) = \min\{C_{MAX}(Q(\pi, v)) : v \in (V(\pi) \setminus T) \cup A\}$$

a pokračuj na **krok 3**.

Krok 2 : Jestliže $|V(\pi)| = 1$ pak vyber $v' \in V(\pi)$ jinak opakuj $T = T \oplus T_{MAX}$ dokud $V(\pi) \setminus T \neq \emptyset$.

Pak vyber $v' \in V(\pi) \setminus T$.

Krok 3 : Nastav $\pi' = Q(\pi, v')$ a $T = T \oplus \bar{v}'$.

NSP bude použita jako součást algoritmu ukázaného v následující kapitole.

2.5.5 Algoritmus Tabu Search

To je velmi jednoduchý náčrt použití NSP na klasickém TS algoritmu. Necht' začínáme s existujícím primárním výrobním rozvrhem a s primárním prázdným tabu seznamem. Ke každé iteraci nalezneme množinu pohybů $V(\pi)$. Následující použitím NSP vybereme pohyb $v' \in V(\pi)$, který vymezí sousedství $\pi' = Q(\pi, v')$ a posléze vytvoříme nový tabu seznam T' . Výrobní rozvrh π' a tabu seznam T' jsou množiny uspořádané pro další iteraci. Nejlepší nalezená hodnota účelové funkce C^* a příslušný nejlepší výrobní rozvrh π^* jsou současně přepsány. Algoritmus se zastaví v jeden ze dvou případů : pokud se najde optimální výrobní rozvrh nebo když počet iterací bez zlepšení výrobního rozvrhu přesáhne *maxiter*. Shora uvedené úvahy nás vedou k následujícímu algoritmu Tabu Search (TSA), který začíná s výrobním rozvrhem π^* (nalezeným jakýmkoliv heuristickým algoritmem), $C^* = C_{MAX}(\pi^*)$, $T = \emptyset$, $\pi = \pi^*$, $iter = 0$, a vracející nejlepší výrobní rozvrh π^* a $C^* = C_{MAX}(\pi^*)$.

Krok 1 : Nastav $iter = iter + 1$. Najdi množinu pohybů $V(\pi)$. Jestliže $V(\pi) = \emptyset$, pak zastav výpočet; π^* je optimální.

Krok 2 : Najdi pohyb $v' \in V(\pi)$, sousedství $\pi' = Q(\pi, v')$ a modifikuj tabu T' aplikací NSP. Nastav $\pi = \pi'$, $T = T'$.

Krok 3 : Jestliže $C_{MAX}(\pi) < C^*$, pak nastav $\pi^* = \pi$, $C^* = C_{MAX}(\pi)$, $iter = 0$ a jdi na krok 1.

Krok 4 : Jestliže $iter = maxiter$ zastav výpočet; jinak jdi na **krok 1**.

3. Kódování pro job shop rozvrhování

3.1 Uvedení do problému

Při popisu jednotlivých kódování jsem vycházel z dokumentů [4]. Pod pojmem kódování v oblasti práce s job shop rozvrhování rozumíme vytvoření určitého způsobu zápisu posloupností operací na strojích do určitého druhu seznamu, který nám umožní provádění výpočtu na počítačích a nalezení co nejlepšího řešení v co nejkratším čase. V neposlední řadě je třeba posuzovat navržené kódování také z hlediska paměťových nároků a složitosti algoritmů, protože i když je dnes výpočetní technika na vysoké úrovni, heuristické výpočty neustálým opakováním stejných operací patří k jednomu z nejvíce náročných algoritmů na velikost paměti. Na zvoleném způsobu kódování záleží, které varianty se budou při heuristickém prohledávání zkoumat. Například pomocí kódování založeného na operacích vytváříme pouze takové kódy, které z hlediska uspořádání jednotlivých operací za sebou na jednotlivých jobech, vytváří vždy správné rozvrhy. Otázkou ovšem zůstává, jestli tím, že vynecháme řešení, která jsou nesmyslná, zároveň neztratíme cestu k řešení, která by byla pro nás výhodná. Tyto a mnoho dalších úvah musí každý programátor vytvářející program pro job shop rozvrhování vzít v potaz.

Při převádění našeho zadání do seznamu musíme mít na zřeteli tři zásadní otázky kódování. Jako první musíme řešit otázku, zda námi vytvořený seznam je proveditelný. Proveditelností rozumíme otázku, zda výsledné řešení dekódované ze seznamu leží v oblasti daného problému. Další otázkou je legálnost výsledku, a to z hlediska zejména pořadí jednotlivých operací na jobech atd. Poslední otázkou je jedinečnost námi vytvořeného seznamu, který musí odpovídat jedinečnému rozvrhu.

V průběhu posledních několika let bylo navrženo následujících 9 reprezentací pro job shop scheduling:

- reprezentace založená na operacích (operation - based representation)
- reprezentace založená na pracích (job - based representation)
- reprezentace založená na preferenčním seznamu (preference list - based representation)
- reprezentace založená na vzájemném vztahu mezi pracemi (job pair relation - based representation)
- reprezentace založená na prioritních pravidlech (priority rule - based representation)
- reprezentace založená na disjunktivním grafu (disjunctive graph - based representation)
- reprezentace založená na časech dokončení (completion time - based representation)
- reprezentace založená na strojích (machine - based representation)
- reprezentace náhodných klíčů (random keys representation)

Tyto reprezentace mohou být rozčleněny do následujících dvou základních kódovacích postupů:

- přímý přístup
- nepřímý přístup

V přímém přístupu je rozvrh zakódován do seznamu jednotlivých operací a heuristické metody v tomto seznamu vytváří přesně definovaná sousedství, pomocí kterých probíhá vlastní hledání. Do této skupiny patří reprezentace založená na operacích, reprezentace založená na pracích, reprezentace založená na vzájemném vztahu mezi pracemi, reprezentace založená na časech dokončení a reprezentace náhodných klíčů.

V nepřímém přístupu se místo samotných operací rozvrhují takzvaná dispečerská pravidla, pomocí kterých posléze uspořádáme operace. Cílem hledání je tedy nalezení co nejlepší sekvence dispečerských pravidel. Rozvrh je nakonec konstruován z této nejlepší sekvence dispečerských pravidel postupným dosazováním operací. Do této skupiny patří reprezentace založená na preferenčním seznamu, reprezentace založená na prioritních pravidlech, reprezentace založená na disjunktivním grafu a reprezentace založená na strojích.

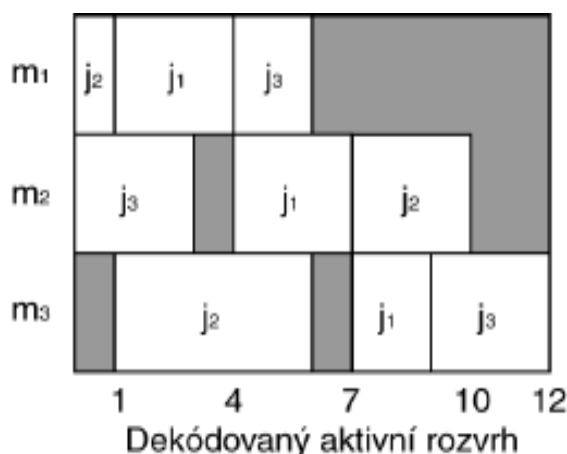
Operace			
	1	2	3
Job	Čas operace :		
j1	3	3	3
j2	1	5	3
j3	3	2	3
	Pořadí strojů :		
j1	m1	m2	m3
j2	m1	m3	m2
j3	m2	m1	m3

Tabulka 3.1

K popisu jednotlivých reprezentací je použita tato tabulka 3.1, která obsahuje 3 joby a 3 stroje.

3.2 Reprezentace založená na operacích (operation - based representation)

Tato reprezentace kóduje rozvrh jako sled prvků, kde každý prvek představuje jednu operaci. Jednou z přirozených cest, jak pojmenovat každou operaci, je použití přirozených čísel (podobně jako u při permutační reprezentaci TSP). Bohužel, kvůli existenci přednostních omezení, ne všechny možné permutace těchto čísel reprezentují proveditelné rozvrhy. Proto byla navržena alternativa : pojmenovat všechny operace obsažené v dané práci stejným symbolem a potom interpretovat toto dle ostatních výskytů v daném seznamu operací. U problému skládajícího se z n - prací a m - strojů seznam obsahuje $n \times m$ prvků. Každá práce je obsažena v seznamu přesně m krát a každé zopakování neoznačuje konkrétní operaci práce, ale jakýsi ukazatel, který říká, že zde po vytvoření rozvrhu bude příslušná operace této práce. Pokud na stroji budou operace dvě, snadno zajistíme, aby se nepředběhly. Stačí při vytváření rozvrhu u daného stroje při vkládání operace na místo prvního ukazatele dané práce vložit operaci, která se má provádět jako první na tomto stroji, pak operaci,



Obrázek 3.1

která se má provádět jako druhá atd. Již na první pohled je vidět, že jakákoliv permutace seznamu a následné vytvoření rozvrhu musí vždy reprezentovat proveditelné řešení.

Rozvrh je dekódován ze seznamu následujícím postupem:

- Nejprve přelož seznam ukazatelů na seznam uspořádaných operací.
- Potom generuj rozvrh metodou jednorůchodové heuristiky, která vypadá takto : První operace na seznamu je rozvrhována nejdříve, potom druhá operace atd.

Každá operace, která je zapisována, je umístěna na nejlépe dostupném čase příslušného stroje. Proces je

opakován, dokud všechny operace nejsou rozvrženy. Je-li rozvrh generován touto procedurou, je garantováno, že se jedná o *aktivní* rozvrh schopný výpočtu. Řekneme, že rozvrh je *aktivní*, pokud neexistuje operace, která může začít dříve bez zpoždění ostatních operací. Dalším stavem je tzv. *non-delay* rozvrh (bez prodlení), ve kterém žádný stroj není zastaven, pokud by mohl vykonávat jiné operace.

Uvažujme problém, skládající se ze 3 prací a 3 strojů, jak je popsán v tabulce.3.1 Předpokládejme, že seznam ukazatelů vypadá následovně [2 1 1 1 2 2 3 3 3]. Protože se každá práce skládá ze tří operací, vyskytují se přesně tři časy také v seznamu. Každý ukazatel unikátně označuje konkrétní operaci a může být určen vzhledem k pořadí výskytu v sekvenci. Necht' o_{jim} znamená i -tou operaci j -té práce na m -tém stroji. Potom může být seznam přeložen do unikátního seznamu uspořádaných operací [$o_{211} o_{111} o_{122} o_{133} o_{223} o_{232} o_{312} o_{321} o_{333}$]. Operace o_{211} má nejvyšší prioritu a je rozvrhována jako první, potom o_{111} , atd. Výsledný aktivní rozvrh je ukázán na obrázku 3.1.

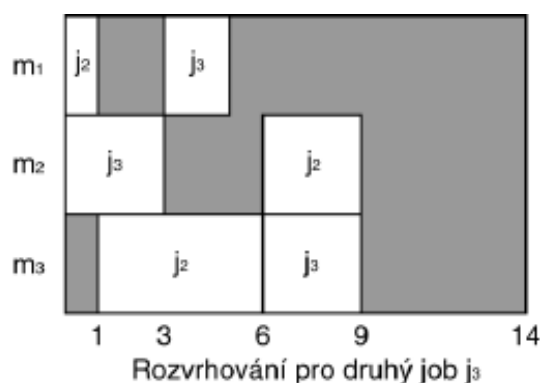
3.3 Reprezentace založená na pracích (job - based representation)

Tato reprezentace se skládá ze seznamu n -prací a rozvrh je sestavován s ohledem na sled prací. Pro danou sekvenci prací jsou všechny operace první práce v seznamu rozvrhovány jako první, jako další jsou pak uvažovány operace druhé práce v seznamu atd., dokud nejsou všechny operace práce rozvrženy. Tento proces je opakován s každou z prací v seznamu vytvořeném při hledání vhodné sekvence. Jakákoliv permutace prací odpovídá proveditelnému rozvrhu.

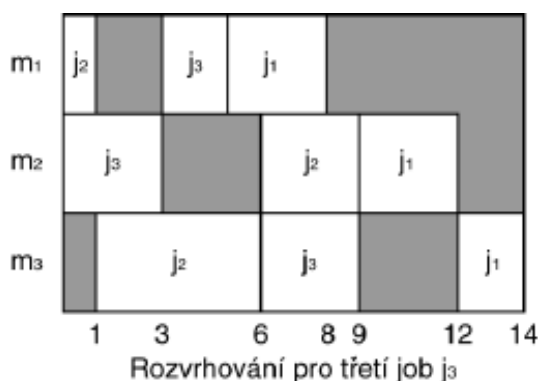
Uvažujme seznam zadaný takto [2 3 1]. První zpracovávaná práce je práce j_2 . Přednostní omezení pro j_2 je [$m_1 m_3 m_2$] a odpovídající procesní čas pro každý stroj je [1 5 3]. Za prvé, operace práce j_2 jsou rozvrhovány, jak je ukázáno na obrázku 3.2. Potom je zpracovávána práce j_3 . Její operace upřednostňují stroje v pořadí [$m_2 m_1 m_3$] a jejich odpovídající procesní časy pro každý stroj jsou [3 2 3]. Každá z těchto operací je rozvrhována v nejlepší dostupném procesním čase, jak je ukázáno na obrázku 3.3. Nakonec jsou rozvrhovány operace práce j_1 , jak je ukázáno na obrázku 3.4.



Obrázek 3.2



Obrázek 3.3



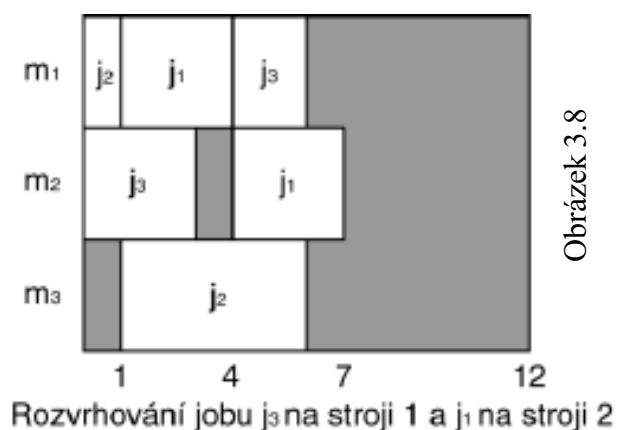
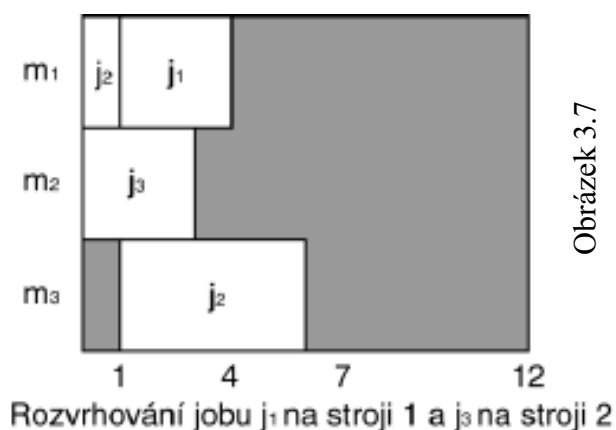
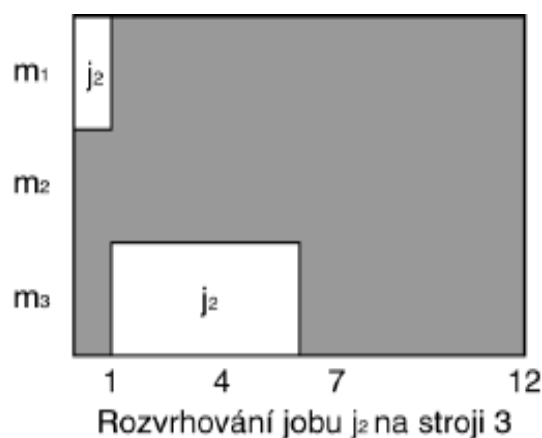
Obrázek 3.4

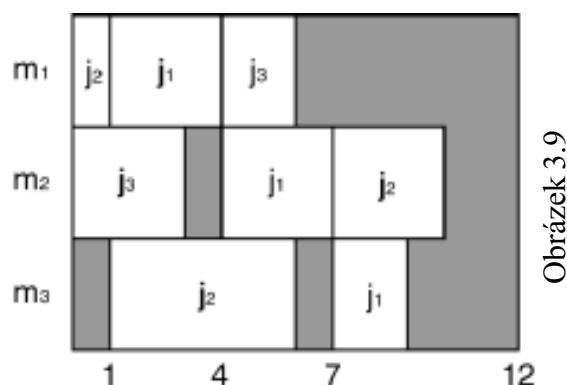
3.4 Reprezentace založená na preferenčním seznamu (preference list - based representation)

Tato reprezentace byla původně navržena Davisem pro jeden určitý druh rozvrhovacího problému. Falkenauer a Bouffouix ji užili pro řešení problému job-shop rozvrhování, který má termíny spuštění a ukončení operací. Croce ji aplikoval na klasický problém job-shop rozvrhování.

Pro problém n -prací a m -strojů job-shop rozvrhování je seznam formován z m podseznamů, každého pro jeden stroj. Každý podseznam je řetězcem symbolů o délce n a každý symbol identifikuje operaci, která má být provedena na relevantním stroji. Podseznamy nepopisují sekvenci operací na stroji, ale jsou to takzvané preferenční seznamy. Každý stroj má svůj vlastní preferenční seznam. Aktuální rozvrh je vydedukován ze seznamu pomocí simulace, která analyzuje stav čekacích front před strojem, a pokud je to nutné, užívá preferenčního seznamu k určení rozvrhu. Pokud ho užije, je vybrána ta operace, která se vyskytuje na prvním místě v preferenčním seznamu.

Nyní si názorně ukážeme, jak odvodit aktuální rozvrh z konkrétního seznamu. Uvažujme jednoduchý příklad, zadaný dle příkladu na tabulce 3.1. Předpokládaný seznam je například ve tvaru $[(2\ 3\ 1)(1\ 3\ 2)(2\ 1\ 3)]$. První část $(2\ 3\ 1)$ je preferenční seznam pro stroj m_1 , část $(1\ 3\ 2)$ je preferenční seznam pro stroj m_2 a část $(2\ 1\ 3)$ pro stroj m_3 . Z tohoto preferenčního seznamu můžeme odvodit, že první preferenční operace jsou práce j_2 na stroji m_1 , j_1 na stroji m_2 a j_2 na stroji m_3 . Vzhledem k přednostním omezením operace je pouze j_2 na m_1 schopna rozvrhování, takže je rozvrhována na m_1 prvně, jak je ukázáno na obrázku 3.5. Další rozvrhovanou operací je j_2 na m_3 , jak je ukázáno na obrázku 3.6. Nynější stávající preferenční operace jsou j_3 na m_1 , j_1 na m_2 a m_3 . Protože nejsou všechny z nich rozvrženy ve stávajícím čase, hledáme druhé preferenční operace v každém seznamu, jsou to j_1 na m_1 , j_3 na m_2 a m_3 . Rozvrhovatelne operace jsou j_1 na m_1 a j_3 na m_2 .



Rozvrhování jobu j_2 na stroji 2 a j_1 na stroji 3

Hotový rozvrh

Rozvrhneme je, jak je ukázáno na obrázku 3.7. Další rozvrhovatelne operace jsou j_3 na m_1 a j_1 na m_2 , které následovně rozvrhneme, jak je ukázáno na obrázku 3.8. Po tomto aktu jsou další rozvrhovatelne operace j_2 na m_2 a j_1 na m_3 , jak je ukázáno na obrázku 3.9. Poslední rozvrhované operace jsou j_3 na m_3 , jak je ukázáno na obrázku 3.10. Nyní jsme dokončili dekódování rozvrhu ze seznamu a dostali jsme proveditelný rozvrh s minimalizační funkcí rovnou 12. S použitím této dekódovací procedury dostaneme ze všech možných seznamů vždy proveditelný rozvrh.

Croce argumentoval, že dedukční procedura generuje pouze rozvrhy bez prodlení, takže si nemůžeme být jisti, že bylo dekódováno optimální řešení. Vytvořil poměrně propracovanou proceduru ohodnocování následujícího kroku tak, aby pomohl dedukční proceduře získat aktuální rozvrh. Bohužel jeho názor není pravdivý. Když generujeme rozvrh bez prodlení, musíme nejprve identifikovat kritický stroj, který může začít jako první, a potom musíme vybrat takovou operaci, kterou můžeme nejdříve začít zpracovávat na kritickém stroji. Protože v procesu dedukce je vybírána další operace vzhledem k preferenčnímu seznamu, rozhodně nezískáme rozvrh bez zpoždění, ale aktivní rozvrh. Např. rozvrh na šestém obrázku je aktivní rozvrh, ale nikoli rozvrh bez zpoždění, protože stroj m_3 zůstává v nečinnosti od času 6 do času 7, kdyby mohl začít zpracovávat třetí operaci práce j_3 na stroji 3.

Tato reprezentace byla použita například Kobayasim, Gifflerem nebo Thompsonem.

3.5 Reprezentace založená na vztahu mezi pracemi (job pair relation - based representation)

Nakano a Yamada při navrhování této reprezentace užili binární matici k zakódování rozvrhu a to tak, že daná matice je určena vzhledem k přednostním vztahům dvojic prací na příslušném stroji. Uvažujme následující problém 3 prací a 3 strojů. Přednostní omezení u operací pro práce a jeden proveditelný rozvrh pro problém je zadán v tabulce 3.2.

Binární symboly jsou definovány tak, aby identifikovaly vzájemné vztahy mezi dvojicemi prací. Příklad definice je zde :

Příklad pro 3 práce a 3 stroje							
Pořadí operací				Pořadí prací			
Job	Pořadí strojů			Stroj	Pořadí prací		
j1	m1	m2	m3	m1	j2	j1	j3
j2	m1	m3	m2	m2	j3	j1	j2
j3	m2	m1	m3	m3	j2	j1	j3

Tabulka 3.2

$$x_{ijm} = \begin{cases} 1, & \text{jestliže operace } i \text{ má být rozvržena před operací } j \\ 0, & \text{jinak.} \end{cases}$$

Nyní si předvedeme jednoduchý příklad na přednostní vztahy dvou prací (j_1, j_2) na strojích (m_1, m_2, m_3) . Vzhledem k zadanému seznamu snadno dostaneme kombinaci $(x_{121} x_{122} x_{123}) = (0 \ 1 \ 0)$. Pro dvojici prací (j_1, j_3) máme $(x_{131} x_{132} x_{133}) = (1 \ 0 \ 1)$, a pro dvojici prací (j_2, j_3) jsou přednostní vztahy na strojích (m_1, m_3, m_2) $(x_{231} x_{233} x_{232}) = (1 \ 1 \ 0)$. Jak je vidno, sekvence symbolů x_{ijm} pro dvojici prací by měla zůstat shodná se sekvencí operací první práce i . Například pro dvojici prací (j_2, j_3) je sekvence operací práce j_2 (1 3 2), takže podobné proměnné jsou provedeny jako seznam $(x_{231} x_{233} x_{232})$ a ne seznam $(x_{231} x_{232} x_{233})$. Jestliže shrneme tyto výsledky, dostaneme tuto binární matici reprezentace pro zadaný proveditelný rozvrh:

$$\begin{array}{l} (j_1, j_2) \text{ na } (m_1, m_2, m_3) : \\ (j_1, j_3) \text{ na } (m_1, m_2, m_3) : \\ (j_2, j_3) \text{ na } (m_1, m_3, m_2) : \end{array} \begin{vmatrix} x_{121} & x_{122} & x_{123} \\ x_{131} & x_{132} & x_{133} \\ x_{231} & x_{233} & x_{232} \end{vmatrix} = \begin{vmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{vmatrix}$$

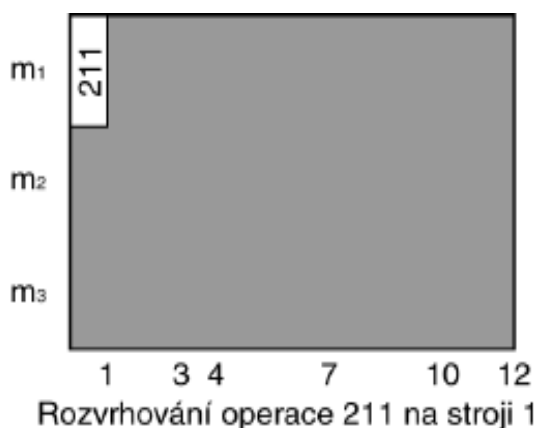
Tato reprezentace je snad nejsložitější ze všech reprezentací pro job-shop rozvrhování a má také největší mredundanci. Vedle této složitosti jsou vyprodukované seznamy této reprezentace většinou ilegální, ať už jsou produkovány počáteční procedurou nebo heuristickými metodami.

3.6 Reprezentace založená na prioritních pravidlech (priority rule - based representation)

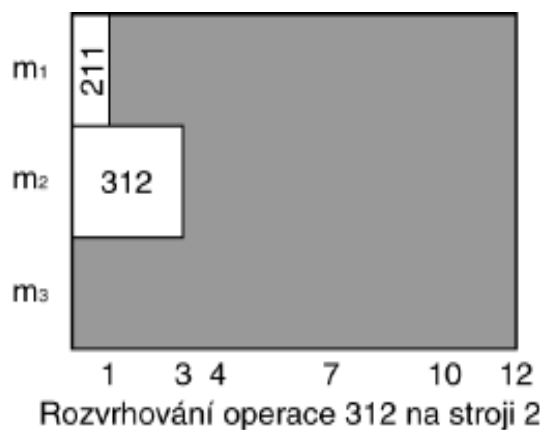
Tato reprezentace, navržená Dorndorfem a Peschem[10], je založená na prioritních pravidlech, kde je seznam zakódován jako sekvence dispečerských pravidel pro vlastní operace a skutečný rozvrh je sestaven metodou prioritního skládání operací, založenou na sledu dispečerských pravidel.

Prioritní dispečerská pravidla jsou pravděpodobně nejvíce užívána programátory a heuristiky k řešení rozvrhovacích problémů z důvodu jejich jednoduché použitelnosti a menší časové náročnosti. Gifferův a Thompsonův algoritmus může být považován za běžný základ všech heuristik založených na prioritních pravidlech. Jedním z problémů je rozpoznat efektivní prioritní pravidlo. Rozsáhlá shrnutí a diskuse o prioritních pravidlech vytvořili Panwalker, Iskender, Haupt a Blackstone.

Pro problém 3 prací a 3 strojů je seznam řetězcem o $n \times m$ vstupech $(p_1, p_2, \dots, p_{nm})$. Vstup p_i reprezentuje jedno pravidlo z množiny předdefinovaných pravidel. Vstup na i -té pozici říká, že konflikt v i -té operaci Gifferova a Thompsonova algoritmu by se mohl vyřešit použitím prioritního pravidla p_i . Lépe řečeno, operace z množiny konfliktů má být vybrána pomocí pravidla p_i . Aby nevznikla návaznost, používá se náhodný



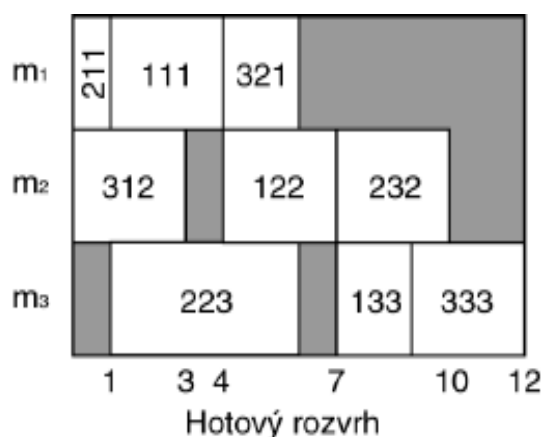
Obrázek 3.11



Obrázek 3.12



Obrázek 3.12



Obrázek 3.13

výběr. Pro další ukázkou definujeme tyto proměnné :

PS_t = částečný rozvrh obsahující t rozvržených operací,

S_t = množina rozvrhovatelných operací v iteraci t , korespondující s daným PS_t ,

δ_i = nejbližší čas, ve kterém může operace $i \in S_t$ začít,

ϕ_i = nejbližší čas, ve kterém může být operace $i \in S_t$ dokončena,

C_t = množina konfliktních operací v iteraci t .

A nyní podrobně vysvětlím algoritmus, pomocí kterého vytvoříme samostatný rozvrh. Na obrázcích 3.11 až 3.13 je podrobně tento postup rozkreslen.

Krok 1 : Nechť $t = 1$, PS_t začíná jako prázdný částečný rozvrh a S_t obsahuje všechny operace bez předchůdců.

Krok 2 : Určíme $\phi_t^* = \min_{i \in S_t} \{\phi_i\}$ stroj m^* na kterém může ϕ_t^* být provedena. Jestliže existuje více strojů než tento jeden, vybereme jeden náhodným číslem.

Krok 3 : Vytvoříme konfliktní množinu c_t , která obsahuje všechny operace $i \in S_t$ s $\delta_i < \phi_t^*$, které vyžadují stroj m^* . Vybereme podle prioritního pravidla jednu operaci z c_t , přidáme tuto operaci do PS_t na místo, kde to je možné, tím vytvoříme nový částečný rozvrh PS_{t+1} . Jestliže existuje více operací se shodným prioritním pravidlem p , potom volíme náhodně.

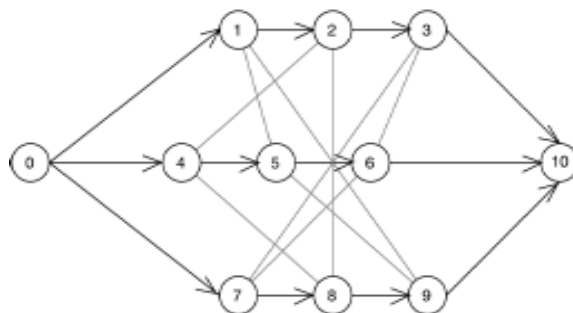
Krok 4 : Obnovíme PS_{t+1} přesunutím vybrané operace z S_t a přidáním přímého nástupce operace S_t . Zvýšíme t o jednu.

Krok 5 : Zpět na krok 2, dokud není vytvořen celý rozvrh.

3.7 Reprezentace disjunktivním grafem (disjunctive graph-based representation)

Na tuto reprezentaci, vytvořenou Tamaki a Nashikawa, můžeme pohlížet, jako na odrůdu reprezentace založené na vztahu mezi pracemi. Problém job-shop rozvrhování pak může být vyjádřen disjunktivním grafem. Disjunktivní graf je reprezentován následně : $G(N, A, E)$, kde :

- N obsahuje uzly reprezentující všechny operace.
- A obsahuje hrany spojující po sobě jdoucí operace stejného jobu.
- E obsahuje disjunktivní hrany spojující operace, které mají být provedeny stejným strojem.



Obrázek 3.14

Disjunktivní graf z 3 jobů a 3 stroji

Omezení disjunktivního grafu je provedeno hranami E , které jako disjunktivní hrany jsou obousměrné. Sestavením rozvrhu stanovíme orientaci všech disjunktivních hran tak, abychom určili sled operací na jednotlivých strojích. Jakmile je sled pro stroj určen, disjunktivní hrany jsou nahrazeny konjunktivními hranami. Obrázek 3.14 ilustruje příklad pro disjunktivní graf pro 3 joby a 3 stroje. Seznam se sestává z binárních řetězců, které odpovídají seznamu hodnot diskretních hran v E , jak ukazuje následující příklad, kde každé e_{ij} leží na disjunktivních hranách mezi uzly i a j a je definováno :

$$e_{ij} = \begin{cases} 1 & \text{stanoví orientaci od uzlu } j \text{ k } i \\ 0 & \text{stanoví orientaci od uzlu } i \text{ k } j \end{cases}$$

Seznam pořadí disjunktivních hran :

e_{15}	e_{19}	e_{59}	e_{24}	e_{28}	e_{48}	e_{36}	e_{37}	e_{67}
----------	----------	----------	----------	----------	----------	----------	----------	----------

Výsledný seznam :

[0	0	1	1	0	0	0	1	1]
----	---	---	---	---	---	---	---	----

Cílem job-shop rozvrhování je najít pořadí operací na každém stroji. To znamená stanovit orientaci disjunktivních hran tak, aby výsledný graf byl acyklický. Tím totiž garantujeme, že mezi operacemi nebudou konflikty přednosti. Je jasné, že libovolný seznam může vytvořit cyklický graf, který by byl neproveditelný. Proto tento seznam nebude použit k reprezentaci rozvrhu, ale pouze jako preference rozhodnutí. K odvození rozvrhu se používá *critical path-based procedure* (postup kritické cesty). V průběhu procesu dedukce, kdy na stroji se vyskytne konflikt dvou uzlů (operací), je příslušná část seznamu využita k určení pořadí zpracování těchto dvou operací. Jinými slovy, stanoví orientaci disjunktivní hrany mezi tyto dva uzly.

3.8 Reprezentace času dokončení (completion time-based representation)

Tuto reprezentaci opět navrhli Yanada a Nakano. Výsledek je seznamem pořadí časů dokončení operací. Pro stejný příklad, jako je v tabulce 1, může být chromozon zastoupen tímto vyádřením:

$$[c_{111} \ c_{122} \ c_{133} \ c_{211} \ c_{223} \ c_{232} \ c_{312} \ c_{321} \ c_{333}]$$

kde c_{ijr} znamená čas dokončení operace i jobu j na stroji r . Je jednoduché vyvodit, že takováto reprezentace není vhodná například pro většinu genetických operátorů, protože získáme neplatný rozvrh. Proto Yamada a Nakano navrhli pro genetické hledání speciální operátor křížení.

3.9 Reprezentace založené na strojích (machine-based representation)

Tuto reprezentaci navrhli Dorndorf a Pesch [10]. Seznam je zakódován jako sled strojů a rozvrh je sestrojen heuristickým posouváním úzkého místa po sledu.

Heuristickým posouváním úzkého místa, navržená Adamsem, je pravděpodobně nejsilnějším postupem mezi dodnes známými heuristikami pro job-shop rozvrhování. Dává do sledu stroj po stroji a bere každý čas, kdy je stroj identifikován jako úzké místo mezi stroji, které ještě nebyly přidány do sledu. Pokaždé, kdy je nový stroj přidán do sledu, všechny předešlé stanovené sledy jsou místně přeoptimalizovány. Procedury identifikace úzkého místa a místní přeoptimalizace jsou založeny na opakovaně řešeném problému rozvrhování jednoho stroje, což je zjednodušením úvodního problému. Hlavním přispěním jejich metody je způsob užití tohoto zjednodušení pro rozhodnutí o pořadí, ve kterém měl být daný stroj vložen do sledu strojů. Heuristika posouvání úzkého místa vychází z klasické myšlenky přidělování priority k úzkým místům strojů. Rozdílné měření kvality úzkého místa nám dá různé sledy úzkých míst strojů. Adams také navrhl výpočetní vizi této heuristiky, v níž uvažuje různé sledy strojů. Místo hledání ve výpočtovém stromu Dorndorf a Pesch navrhli genetickou strategii k určení nejlepšího sledu strojů pro heuristické posouvání úzkého místa. Rozdíl mezi heuristickým posouváním úzkého místa a heuristickým algoritmem je, že úzké místo již není rozhodovacím kritériem pro volbu dalšího stroje, který je kontrolován daným seznamem.

3.10 Reprezentace náhodným klíčem (random key representation)

Tuto reprezentaci poprvé uvedl Bean. Touto technikou mohou heuristické algoritmy vytvářet proveditelný rozvrh bez vytvoření dodatečných režii (přebytečných řídicích prvků) pro širokou různorodost problému sledů a optimalizací. Norman a Bean dále zevšeobecnil metodu job-shop rozvrhování. Reprezentace random key zakódovává řešení pomocí *náhodného čísla*.

Tyto hodnoty jsou užívány jako třídící klíče k rozkódování řešení. Pro n -jobů a m -strojů se každý podseznam (náhodný klíč) skládá ze dvou částí. První částí je celé číslo v množině $\{1, 2, 3, \dots, m\}$ a druhou částí zlomek získaný náhodně z intervalu $(0, 1)$. Celá část jakéhokoliv náhodného čísla se interpretuje jako přidělení stroje pro job. Členění zlomkových částí poskytuje sled prací na každém stroji. Uvažujeme stejný příklad jako v tabulce 1. Předpokládejme, že seznam vypadá takto :

[1,34 1,09 1,88 2,66 2,91 2,01 3,23 3,21 3,44]

Členění klíčů pro stroj 1 podle stoupajícího pořadí má za výsledek sled jobů 2->1->3, pro stroj 2 sled jobů 3->1->2 a pro stroj 3 sled jobů 2->1->3. Necht' o_{jm} ukazuje na job j na stroji m . Výsledek proto může být přeložen na unikátní seznam seřazených operací :

[o21 o11 o31 o32 o12 o22 o23 o13 o33]

Je jednoduché rozpoznat, že ve výše uvedeném sledu prací může dojít k chybě přednosti operací. Proto Norman a Bean doplnili toto kódování pseudokódem pro zacházení s omezením přednosti operací.

3.11 Vyhodnocení

V předchozí části jsme si shrnuli různé kódovací techniky. V této části je názorně zhodnotíme, a to podle následujících kritérií :

- složitost dekodéru
- vlastnosti kódovaného prostoru a mapování

- požadavky na paměť

3.11.1 Složitost dekodéru

Každý způsob kódování má jiný způsob práce s operacemi. Do toho se odvíjí složitost dekodéru, kterou rozumíme stupeň přenechání práce při rozvrhování na ostatní prvky výpočtu, například vyhledávací algoritmy atd. Stupeň složitosti dekodéru může být klasifikován v některé z následujících úrovní :

Level 0 : žádný dekodér. Sem patří *repräsentace založená na časech dokončení*. V tomto případě jsou všechny obtíže svaleny na vyhledací algoritmy.

Level 1 : jednoduché mapování vztahů. Sem patří *repräsentace založená na operacích, repräsentace založená na vzájemném vztahu mezi pracemi a repräsentace založená na pracích*.

Level 2 : jednoduchá heuristika. Sem patří *repräsentace založená na prioritních pravidlech*.

Level 3 : složitá heuristika. Sem patří *repräsentace založená na preferenčním seznamu, kódování užitě Kobayashim, repräsentace založená na disjunktivním grafu a repräsentace založená na strojích*.

3.11.2 Vlastnosti kódovaného prostoru a mapování

Kódovací prostor může být klasifikován do dvou tříd, jedna obsahuje pouze prostor proveditelných řešení a druhá prostor všech řešení. Mezi repräsentace, které patří do první třídy, náleží *repräsentace založená na operacích, repräsentace založená na prioritních pravidlech, repräsentace založená na pracích a repräsentace disjunktivním grafem*. Ostatní repräsentace patří do druhé třídy.

3.11.3 Paměťové požadavky

Pro n -prací a m -strojů při nadefinované délce seznamu jako $m \times n$ podseznamů, dostaneme kódování pro job-shop, které může být klasifikováno do tří tříd :

- 1) kódovaná délka je kratší než standartní délka. Do této třídy patří *repräsentace disjunktivním grafem, repräsentace založená na strojích a repräsentace založená na pracích*. Všimněte si, že kódovací prostor pro *repräsentaci na založené na strojích* pouze koresponduje k části prostoru řešení, ale neobsahuje celkový prostor řešení. Proto zde není garance, že najdeme optimum kódovacími technikami.
- 2) Kódovaná délka je delší než standartní délka. Pouze *repräsentace založená na vzájemném vztahu mezi pracemi*. Repräsentace je vysoce přebytná.
- 3) Kódovací délka je shodná se standartní. Sem patří zbývající repräsentace.

3.12 Závěr

Po posouzení všech vlastností daných kódovacími technikami, zejména složitosti programování, rychlosti dosažení výsledků, možnosti minout cestu ke kvalitativně lepšímu řešení jsem pro svůj program na vyhledávání pomocí heuristických metod zvolil repräsentaci založenou na preferenčním seznamu.

Závěrem bych rád zdůraznil, že abychom poskytli důvěryhodné posouzení těchto kódovacích technik, je důležité provést další porovnávací studie těchto technik při standartních experimentálních podmínkách užitím benchmarkových testů.

4. Programová část

4.1 Popis systému DELPHI

4.1.1 Základ systému

Systém Delphi je plně adaptabilní 32 bitový vývojový systém, umožňující vytvářet spustitelné programy pod operační systém WINDOWS 95 a WINDOWS NT. Jedná o dvoucestný nástroj, vizuální prostředí okamžitě reaguje na vlastní změnu odpovídající změnou zdrojového kódu, jehož velkou část Delphi dokáže generovat zcela automaticky. Přesto má programátor celý zdrojový kód pod svou kontrolou a automaticky generované části může libovolně měnit. Tento nový způsob programování nazývaný odborníky RAD, v překladu *Rapid application development*, umožňuje programátorům zkvalitnit a urychlit práci.

Vizuální komponenty vkládané do formuláře jsou sledovány a měněny pomocí *Object inspectoru*. Jejich vlastní nastavení, jako je pozice v okně, popisky, barva atd jsou současně uloženy v souboru s příponou *.dfm. Nejpoužívanější komponenty jsou umístěny přímo v liště, kde je jejich použití snadnější. Firma Borland, která je autorem systému, nabízí i dodávku nestandardních vizuálních komponent, které se jednoduchým způsobem přidávají do systému.

4.1.2 Prodejní varianty systému Delphi

Na trhu je dnes již třetí verze systému Delphi, která navazuje na předešlé verze, jak co se týče kompatibility vytvořených programů, tak i plně 32 bitového přístupu. Systém Delphi se zároveň distribuuje v několika, zpravidla ve třech modifikacích, které se liší samozřejmě cenou a úrovní podpory databázových nástrojů, prostředí, editor a kompilátor zůstávají naprosto stejné.

Základní varianta (*Standard*, dříve *Desktop*) obsahuje všechny funkce potřebné pro psaní běžných programů a s omezenou podporou databází. Střední varianta (*Professional*, dříve *Developer*) obsahuje lepší podporu pro správu rozsáhlejších databází, více dokumentace a větší množství pomocných nástrojů. Nejvyšší varianta (*Client/Server suite*) je určena profesionálním programátorům a je určena vývoj nejrozsáhlejších aplikací typu client-server. Obsahuje specifické nástroje týkající se jazyka SQL, podporu pracovních skupin a Internetu.

4.1.3 Objektově orientované programování

Hlavní výhodou systému Delphi je plně objektové prostředí. Další výhodou je vysoká zapouzdřenost standardních WIN API funkcí v předem připravených objektech. Tím řešením se ve většině případů vyloučí přímé používání těchto funkcí v programu

Mezi další výhody systému patří mimo jiné rychlý 32-bitový compiler a debugger, plná podpora a integrace Windows95, včetně nových ovládacích prvků, možnost opakovaného použití celých formulářů a knihoven ve formě *.dll souborů, možnost tvorby tzv. balíčků, kdy se do výsledného programu sestaví pouze určené komponenty, vysoce výkonný editor pro psaní kódu, vynikající podpora databází, podpora pro multithreading, nové 32bitové datové typy proměnných, sada pomocných nástrojů pro ladění programů, práci s databázemi a editaci souborů zdrojů.

4.1.4 Vizuální komponenty

Práce s vizuálními komponentami je jednoduchá a vzdáleně připomíná stavebnici Lego. Pokud nějakou komponentu, například tlačítko, chceme použít ve svém formuláři, stačí jen klepnout na příslušnou ikonu a umístit komponentu do formuláře. Velikost, barvu, popisky lze kdykoliv změnit. K rychlejší orientaci slouží bublinková nápověda.

4.1.5 Editace zdrojového kódu

Po návrhu designu formuláře je obvykle potřebné napsat programový kód ošetřující různé události, které mohou nastat při běhu programu. Tento kód píšeme v okně editoru, ve kterém je možné mít otevřeno několik souborů současně. Tento editor je v systému Delphi ve verzi 3.0 velmi výkonným a uživatelsky příjemným nástrojem. Např. při napsání jména objektu a tečky, je možné stisknout klávesy *Ctrl+space* a editor vyvolá malé okno, ve kterém jsou uvedeny všechny metody a vlastnosti přístupné uvedenému objektu. Díky tomu odpadá neustálé hledávání v helpu a hledání definic a vlastních syntaxí jednotlivých objektů. Tato funkce editoru zahrnuje i objekty vytvořené v průběhu programování, musí být ovšem splněna podmínka správné definice třídy objektů a správné registrace instance objektu. Další vlastností editoru, která je přístupná pouze u poslední verze programu, je možnost zobrazovat hodnoty vlastních proměnných za běhu programu pouhým najetím kurzoru myši nad jméno této proměnné v zdrojovém kódu. Výsledek zobrazí ve formě bublinové nápovědy.

4.1.6 Formuláře

Vytvořením nového projektu se vytvoří v Delphi prázdný formulář, který je připraven pro další práci. Může se začít také s již existujícím formulářem (při použití připravených šablon, popř. po vytvoření nových). Projekt (budoucí spustitelný soubor) může obsahovat libovolný počet formulářů a dialogových oken. Do formuláře se dají jednoduše vkládat komponenty zvolené v paletě komponent a následně můžeme měnit jejich velikost, pozici, název a veškeré další vlastnosti příslušející vybrané komponentě v *Object inspectoru*. Prostředí umožňuje měnit vlastnosti nejen jedné komponenty, ale i více komponent zároveň tak, že pomocí myši a současně stisknuté klávesy *shift* označíme celou skupinu, u které chceme měnit požadované vlastnosti. Pro nastavení vzájemné polohy několika komponent slouží *Alignment palette*.

4.1.7 Object inspector

Při práci s formulářem se používá k nastavení vlastností komponenty nebo samotného formuláře okno Object Inspector. Object inspector neobsahuje seznam všech vlastností vybrané komponenty, ale jenom ty vlastnosti, které je možné editovat v průběhu návrhu aplikace a vytváření vzhledu formuláře. Zbývající vlastnosti jsou tzv. *run-time*. To znamená, že jsou přístupné pouze za běhu aplikace a o jejich změnu se musí provést programátor vložím příslušných příkazů do zdrojového kódu. Roletka (*Object selector*) v horní části okna udává jméno aktuální komponenty a její datový typ a je možné pomocí ní vybrat jakoukoliv komponentu, kterou formulář obsahuje. Pravý sloupec Object inspectoru umožňuje správnou úpravu datového typu nebo vlastnosti. V závislosti na typu vlastnosti můžeme zadat řetězec, číslo nebo zvolit hodnotu z rozbalovacího seznamu, nebo vyvolat specifický editor (indikovaný tlačítkem se třemi tečkami).

Okno Object inspectoru obsahuje dvě položky. Je to *Properties*, kde je možno nastavovat statické vlastnosti komponenty a *Events*, kde definujeme chování komponenty v závislosti na provedené akci (stisk klávesy, klik myši a pod.). Pokud chceme komponentě definovat novou metodu, pak na záložce *Events*

vybereme typ požadované akce a dvakrát na ni poklepeme myší. Systém sám vytvoří deklaraci procedury a nadepíše ve zdrojovém kódu patřičnou hlavičku procedury i s náležitým názvem a parametry. Dále ještě nadepíše začátek a konec procedury a otevře editorové okno se zdrojovým kódem a kurzor nastaví na první příkazový řádek nově vytvořené procedury.

4.1.8 Správa a překlad projektů

Každý projekt se zobrazuje v okně *Project Manager* (menu View), kde vidíme seznam všech formulářů a jednotek tvořících daný projekt. Zde můžeme libovolně přidávat další formuláře, odstraňovat nebo ukládat projekty jako šablony pro pozdější využití.

Projekt je možné překládat několika způsoby. Prvním způsobem je stisknutí klávesy F9 nebo volbou *Run*. Tímto postupem je sice projekt přeložen do výsledného .exe souboru a následně spuštěn, ale v tomto případě se překládají pouze jednotky, které se od posledního překladu změnily. Pokud však zvolíme příkaz *Build All*, pak se přeloží všechny zdrojové jednotky bez ohledu na to, zda se změnily nebo ne.

Všechny informace o projektu potřebné k sestavení výsledné aplikace jsou uloženy v souboru *.dpr. Tento projektový soubor obsahuje seznam všech zdrojových souborů, které projekt tvoří a je zakončen seznamem se spáženými formuláři. Při samotném překladu se nejprve z každého zdrojového souboru vytvoří přeložená jednotka *.dcu. Potom se přeložené jednotky, které tvoří projekt, společně s kódem knihoven spojí do výsledného spustitelného *.exe souboru. Pro potřeby ladění je zde umístěn i standartní debugger, umožňující spouštět program po krocích, zastavovat na určených místech a sledovat hodnoty proměnných

4.1.9 Database desktop

Database desktop je program pro správu, vytváření, editaci a kontrolu databází. Mezi jeho největší výhody zřejmě patří podpora většiny databázových formátů, jako například dBase, Paradox v několika verzích, FoxPro, Acces, ASCII a další. Zvládá také přístup k databázím SQL. Vytváření a editace dat v databázích je stejně snadná jako v tabulkovém procesoru. Zde je ovšem třeba upozornit na nepříjemnou vlastnost všech databázových programů. Každý program si databázi otevře jen pro sebe, takže snadno nastane situace, kdy při spuštění programu se pokusíte změnit databázi v Database Desktopu a program vám to nedovolí.

4.1.10 Image editor

Dalším pomocným nástrojem je *Image editor*. Je to jednoduchý bitmapový editor s nesčetným použitím. Dokáže bez problémů zpracovávat soubory ikon, tlačítek, kurzorů a mimo jiné i tzv. soubory zdrojů *.res, které používají projekty v Delphi. Díky tomuto speciálnímu editoru je vytvoření systémové ikony, nebo interního kurzoru pro vlastní aplikaci velmi jednoduché.

4.1.11 Pomocné nástroje

V tomto systému je přístupno mnoho dalších přidavných užitečných nástrojů, které zjednodušují práci při vývoji nových aplikací. Mimo jiné mezi ně patří i *Alignment pallete*, *Menu designer* a mnoho dalších expertů a průvodců pro návrh předpřipravených dialogových oken a panelů. Je zde na místě se zmínit také o *WinSight*, který umožňuje sledovat tok zpráv ve Windows.

4.2 Popis programu Plánování výroby

4.2.1 Úvodní slovo

Tento program, jehož oficiální název zní *Plánování výroby verze 2.1*, byl vytvořen za účelem rozvrhování výroby pomocí nejmodernějších heuristických algoritmů. Jelikož se jedná o program, ve kterém se dané metody mají i vyzkoušet, zabudoval jsem do těla programu i funkci zobrazování průběhu, která sleduje nalezené hodnoty a výsledek automaticky vykresluje na obrazovku a funkci zapisování průběhu, kde zapisuji průběh celého výpočtu po 500 interacích.

4.2.2 Konfigurace

Protože se jedná o program pracující pouze pod systémem WINDOWS 95, platí pro spuštění programu podobná hardwarová omezení jako pro samotný operační systém, pokud ovšem je požadován vyšší výkon, hlavní faktorem ovlivňujícím rychlost výpočtu je rychlost procesoru a kvalita základové desky. Ostatní faktory jako objem dosažené paměti, přenosová rychlost a přístupová doba hardisku jsou relevantní pouze při začátku výpočtu, kdy se celé vypracované zadání přesune z souborů DBF do seznamové struktury v paměti počítače. Pro zadání maximálně 10 jobů na 10 strojů dostatečně vyhovuje procesor Pentium 100 MHz a paměť 16 MB RAM. Pokud ovšem chceme dosáhnout lepších výsledků, je třeba volit procesor minimálně 166 MHz a počet zpracovaných interací se minimálně zdvojnásobí. Co s týče vybavení systému potřebnými knihovnami, stačí mít BORLAND DATABASE ENGINE, který se ostatně dodává spolu s instalací mého programu.

4.2.3 Obsah pracovního adresáře

Po instalaci programu do zadaného pracovního adresáře, bude tento obsahovat několik důležitých souborů. při práci s programem je možné některé z těchto souborů upravovat. Proto zde popíši, k čemu jednotlivé soubory slouží.

adresar.dbf - Databázový soubor obsahující jména a pořadová čísla uložených zadání. Pokud program při práci se zálohami zjistí nějaké nepatřičnosti, například chybějící soubory v záloze, smazaný adresář zálohy atd., soubor automaticky opraví, tzn. obvykle smaže větu v databázovém souboru a upozorní na nastalou chybu.

mashiny.dbf - Databázový soubor obsahující jména a pořadová čísla strojů. Při spuštění programu se automaticky vymaže a nastaví jako EXCLUSIVE, což znamená, že program půjde spustit jenom jednou. Pokud by při spuštění programu soubor nebyl přítomen v pracovním adresáři, bude automaticky zkopírován z adresáře ZALOHA.

vyrobky.dbf - Databázový soubor obsahující jména a pořadová čísla prací. Při spuštění programu se automaticky vymaže a nastaví jako EXCLUSIVE, což znamená, že program půjde spustit jenom jednou. Pokud by při spuštění programu soubor nebyl přítomen v pracovním adresáři, bude automaticky zkopírován z adresáře ZALOHA.

operace.dbf - Databázový soubor obsahující jména, časy a pořadová čísla operací, příslušná pořadová čísla strojů a prací. Při spuštění programu se automaticky vymaže a nastaví jako EXCLUSIVE, což znamená, že program půjde spustit jenom jednou. Pokud by při spuštění programu soubor nebyl přítomen v pracovním adresáři, bude automaticky zkopírován z adresáře ZALOHA.

plany.exe - Samotný program

Při standardní instalaci se rovněž vytvoří v pracovním adresáři další podadresáře. Zde je jejich význam :

ZALOHA - Adresář obsahující prázdné záložní soubory *adresar.dbf*, *masiny.dbf*, *vyrobky.dbf* a *operace.dbf*. Pokud je smazán, program obvykle vyžaduje přeinstalaci.

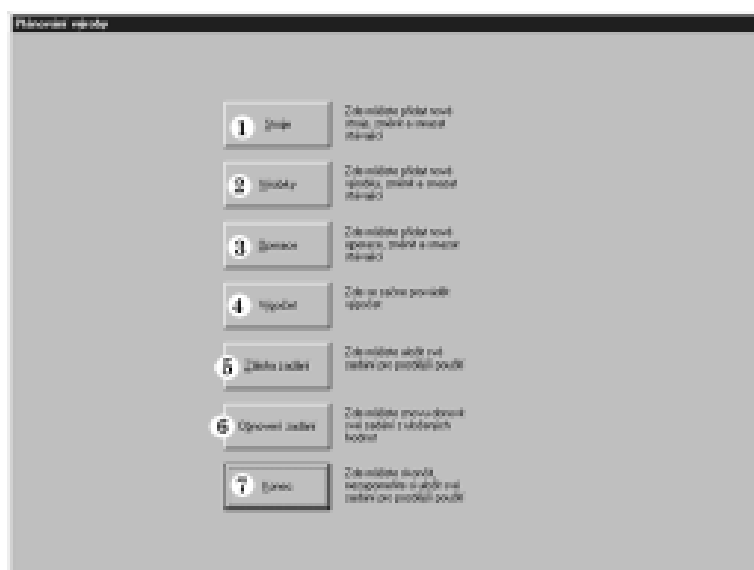
ZADANI_X - Adresář obsahující jednotlivé zálohy souborů *mashiny.dbf*, *vyrobky.dbf* a *operace.dbf*, kde se za písmeno X doplňují jednotlivá pořadová čísla zadání podle souboru *adresar.dbf*. Při zjištění jakýchkoli nesrovnalostí bývá adresář obvykle smazán spolu s příslušnou větou v souboru *adresar.dbf*.

4.2.4 Ovládání programu

Program se ovládá převážně myší s podporou klávesových zkratk. Každé tlačítko má jedno písmeno přiřazené jako klávesovou zkratku a zároveň toto písmeno podtržené v nápisu na tlačítku. Ostatní ovládací prvky, jako jsou přepínače atd., jsou ovládané šipkami.

4.2.5 Spuštění programu a první obrazovka

Program se spouští ikonou **Plánování výroby** v položce Programy nabídky Start nebo souborem **Plany.exe**. Po spuštění naběhne úvodní obrazovka s krátkým představením autora, potom naběhne základní



Obrázek 4.1

nabídka jako na obrázku 4.1. Pokud stiknete tlačítko označené číslem **1**, spustí se formulář umožňující zadávání nových strojů, změnu jejich nastavení atd. Tlačítkem označeným číslem **2** spustíte formulář umožňující zadávání nových prací, změnu jejich nastavení atd. Tlačítkem označeným číslem **3** spustíte formulář umožňující zadávání nových operací, změnu jejich nastavení atd. Tlačítkem označeným číslem **4** spustíte formulář nastavující parametry výpočtu. Tlačítkem označeným číslem **5** spustíte formulář umožňující provést zálohu vytvořeného zadání, popřípadě smazání ostatních záloh. Tlačítkem označeným číslem **6** spustíte formulář umožňující provést obnovení zálohy, popřípadě smazání ostatních záloh. Tlačítkem označeným číslem **7** ukončíte program.

4.2.6 Popis formuláře STROJE

Formulář spustíte tlačítkem STROJE v hlavní nabídce. Dostanete se do okna jako na obrázku 4.2. Pod číslem 1 najdete pořadové číslo stroje, pod číslem 2 najdete jméno příslušného stroje a pod číslem 3 jeho seřizovací čas. Tlačítkem označeným číslem 4 přidáte stroj do seznamu, tlačítkem označeným číslem 5

Seznam strojů :

1	2	3
ID	Název	Seřiz. čas
1	Stroj 1	0
2	Stroj 2	0
3	Stroj 3	0
4	Stroj 4	0
5	Stroj 5	0

Buttons: 4 Nový stroj, 5 Změnit jméno stroje, 6 Změnit seřizovací čas stroje, 7 Smazat stroj, 8 Zpět

Obrázek 4.2

změníte jméno na aktivním stroji, tlačítkem označeným číslem 6 změňte seřizovací čas na aktivním stroji, tlačítkem označeným číslem 7 smažete aktivní stroj a tlačítkem označeným číslem 8 se vrátíte do hlavní nabídky.

4.2.7 Popis formuláře VÝROBKY

Formulář spustíte tlačítkem VÝROBKY v hlavní nabídce. Dostanete se do okna jako na obrázku 4.3.

Seznam strojů :

1	2	3
ID	Název	Seřiz. čas
1	Stroj 1	0
2	Stroj 2	0
3	Stroj 3	0
4	Stroj 4	0
5	Stroj 5	0

Buttons: 4 Nový stroj, 5 Změnit jméno stroje, 6 Změnit seřizovací čas stroje, 7 Smazat stroj, 8 Zpět

Obrázek 4.3

Pod číslem 1 najdete pořadové číslo výrobku, pod číslem 2 najdete jméno příslušného výrobku a pod číslem 3 počet kusů. Tlačítkem označeným číslem 4 přidáte stroj do seznamu, tlačítkem označeným číslem 5 změňte jméno na aktivním výrobku, tlačítkem označeným číslem 6 změňte počet kusů na aktivním výrobku, tlačítkem označeným číslem 7 smažete aktivní výrobek a tlačítkem označeným číslem 8 se vrátíte do hlavní nabídky.

4.2.8 Popis formuláře OPERACE

Formulář spustíte tlačítkem OPERACE v hlavní nabídce. Dostanete se do okna jako na obrázku 4.4. Pod číslem **1** najdete pořadové číslo stroje, pod číslem **2** najdete jméno příslušného stroje a pod číslem **3**

The screenshot shows a software window titled "Operace". It contains three main sections:

- Seznam strojů:** A table with columns 1 (ID), 2 (Název), and 3 (Stav). It lists 5 machines (Stroj 1 to Stroj 5) with status 0.
- Seznam výrobků:** A table with columns 4 (ID), 5 (Název), and 6 (Počet). It lists 10 jobs (Job 1 to Job 10) with a count of 1.
- Seznam operací:** A table with columns 7 (ID), 8 (ID), 9 (ID), 10 (Čas), 11 (Název), and 12 (Pořadí). It lists 5 operations (Oper 1 to Oper 5) with various times and sequence numbers.

At the bottom right, there are several buttons: 13 (Nový), 14 (Změnit jméno), 15 (Změnit čas), 16 (Smazat), and 17 (Zpět).

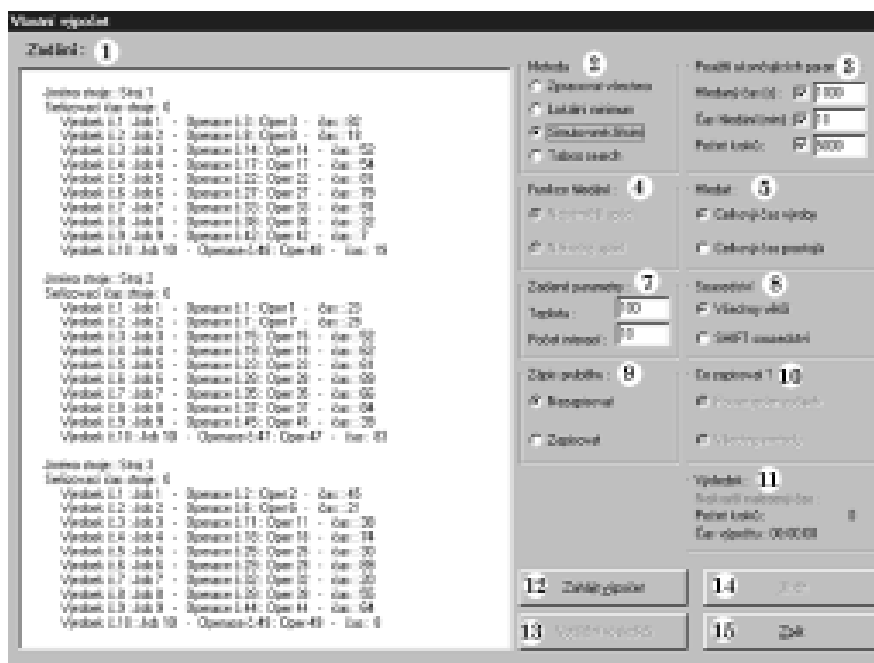
Obrázek 4.4

seřizovací čas. Pod číslem **4** najdete pořadové číslo výrobku, pod číslem **5** najdete jméno příslušného výrobku a pod číslem **6** počet kusů. Pod číslem **7** najdete pořadové číslo operace, pod číslem **8** najdete pořadové číslo stroje, na kterém se operace provádí, pod číslem **9** najdete pořadové číslo výrobku, na kterém se operace provádí, pod číslem **10** najdete čas operace, pod číslem **11** najdete jméno operace a pod číslem **12** najdete pořadí operace na výrobku. Tlačítkem označeným číslem **13** přidáte operaci do seznamu, a to tak, že jako stroj, na kterém se bude operace provádět, bude nastaven aktivní stroj, jako výrobek, na kterém se bude operace provádět, bude nastaven aktivní výrobek a jako pořadí na výrobku bude automaticky nastaveno poslední pořadí na výrobku. Tlačítkem označeným číslem **14** změníte jméno na aktivní operaci, tlačítkem označeným číslem **15** změníte výrobní čas na aktivní operaci, tlačítkem označeným číslem **16** smažete aktivní operaci a tlačítkem označeným číslem **17** se vrátíte do hlavní nabídky.

4.2.9 Popis formuláře VÝPOČET

Formulář spustíte tlačítkem VÝPOČET v hlavní nabídce. Dostanete se do okna jako na obrázku 4.5. Pod číslem **1** najdete při spuštění vypsání zadání, po výpočtu se zde objeví nejlepší řešení se všemi potřebnými hodnotami jako je použitá metoda, počet kroků atd. Tuto položku lze před samotným vytištěním nebo uložením do souboru upravovat jako standardní textový soubor. Pod číslem **2** najdete přepínač umožňující nastavení použité metody. Pod položkou ZPRACOVAT VŠECHNO rozumíme hledání, při kterém se projdou všechny kombinace. Bohužel, jak jsem naznačil již v úvodu, tato metoda se dá použít pouze na menší zadání, ovšem nalezne stoprocentně nejlepší řešení. Pod položkou LOKÁLNÍ MINIMUM rozumíme hledání lokálního minima, které je podrobně popsáno v 2. kapitole. U tohoto hledání rozlišujeme, zda se jedná o hledání pomocí náhodného nebo nejstrmějšího spádu, což je rozlišeno přepínačem označeným číslem **4** a také popsáno podrobně v kapitole 2. Pod číslem **3** najdeme přepínač určující tak zvané ukončující podmínky. Hodnota

v položce HLEDANÝ ČAS určuje velikost času v sekundách, po jehož nalezení se výpočet zastaví, hodnota v položce ČAS HLEDÁNÍ určuje dobu, po jejímž uplynutí se výpočet zastaví, a hodnota POČET KROKŮ určuje počet iterací, po kterých se výpočet zastaví. Ukončující podmínky se využijí pouze tehdy, pokud jsou označeny v malém okénku u názvu položky. Pokud bude označeno více podmínek, použije se vždy ta, která se naplní jako první. Jestliže nebude zaškrtnuta ani jedna, výpočet může trvat teoreticky nekonečně dlouho. Pod číslem 5 najdeme přepínač, který určuje, zda budeme hledat nejmenší čas výroby nebo nejmenší celkové prostoje. Pod číslem 8 najdeme přepínač určující způsob vytváření sousedství pro jednotlivé metody. Pod číslem



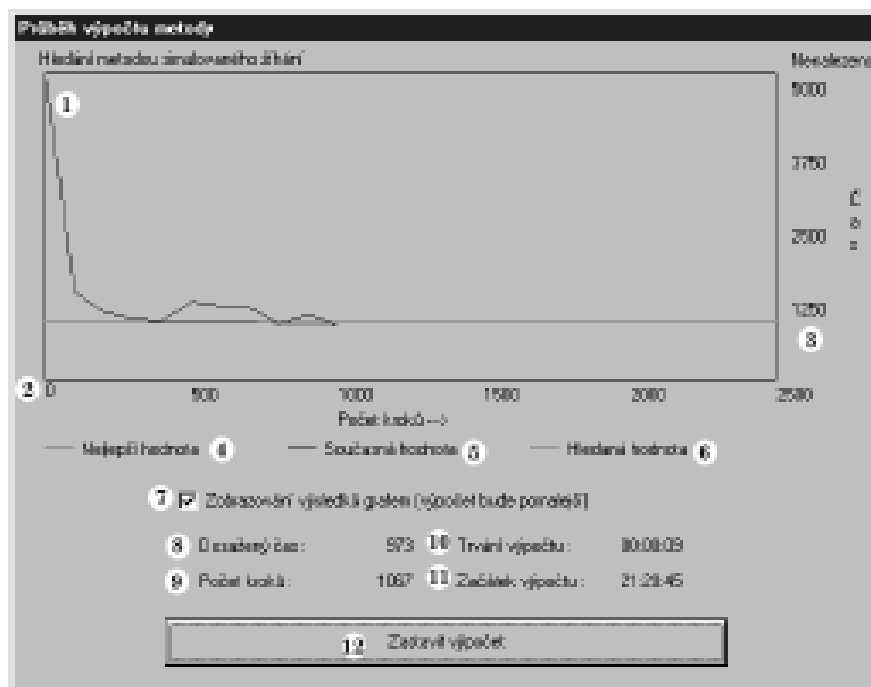
Obrázek 4.5

9 najdeme přepínač umožňující zapisovat průběžné výsledky do souboru *PRUBEH.TXT*. Zapisuje se po 500 krocích a při novém výpočtu se celý soubor smaže. Pod číslem 10 najdeme speciální přepínač umožňující postupně vyzkoušet všechny metody na 5000 interakcí a výsledek zapisovat opět do souboru *PRUBEH.TXT*. Způsob zápisu do tohoto souboru je popsán v kapitole řešených příkladů. Po proběhnutí tohoto výpočtu je výsledek zobrazený na obrazovce obvykle nejlepším řešením poslední metody, tzn. Tabu Search se sousedstvím SHIFT. Pod číslem 11 najdeme výsledné hodnoty, jako je dosažený čas, počet kroků a dobu výpočtu. Tlačítkem označeným číslem 12 spustíme samotný výpočet, tlačítkem označeným číslem 13, které je přístupné pouze po výpočtu, vytiskneme výsledné řešení, tlačítkem označeným číslem 14, které je přístupné pouze po výpočtu, uložíme řešení do textového souboru a tlačítkem označeným číslem 15 se vrátíme zpět do hlavní nabídky.

4.2.10 Popis formuláře PRŮBĚH VÝPOČTU

Formulář spustíte tlačítkem PRŮBĚH VÝPOČTU v formuláři VÝPOČET. Dostanete se do okna jako na obrázku 4.6. Pod číslem 1 najdeme graf ukazující průběh výpočtu, kde červená čára znamená nejlepší výsledek, jak je popsáno pod číslem 4, a modrá současnou nalezenou hodnotu řešení, jak je popsáno pod číslem 5. Vodorovná hnědá čára ukazuje hodnotu hledaného výsledku, jak je popsáno pod číslem 6. Pod číslem 2 najdeme osu x , která ukazuje provedený počet kroků, pod číslem 3 osy y , která ukazuje velikost výrobního času řešení. Přepínačem označeným číslem 7 zapínáme grafické zobrazování výsledků. Pod číslem 8 se průběžně zobrazuje dosažený čas, pod číslem 9 se zobrazuje počet provedených iterací, pod číslem 10 se nachází celková doba výpočtu a pod číslem 11 celkový čas. Tlačítkem označeným číslem 12

jednak spouštíme výpočet a jednak předčasně ukončujeme. Pro lepší orientaci se při výpočtu automaticky umisťuje jméno použité metody na záhlaví formuláře.



Obrázek 4.6

4.2.11 Popis formuláře ZÁLOHA

Formulář spustíte tlačítkem ZÁLOHA v hlavní nabídce. Dostanete se do okna jako na obrázku 4.7. Pod číslem 2 se nachází jednotlivé zálohy ve formě databázových vět. Pokud stisknete klávesu ENTER na jednotlivé větě, jak je naznačeno pod číslem 1, program nabídne místo vytvoření nové zálohy přepsání stávající. Do položky 3 se zapisuje jméno nové zálohy. Tlačítkem označeným číslem 4, které je přístupné, pouze po zadání jména nové zálohy, uložíme naše zadání, tlačítkem označeným číslem 5, které je přístupné, pouze pokud je alespoň jedna záloha, smažeme aktivní zadání a tlačítkem označeným číslem 6 se vrátíme zpět.

1

2

3

4

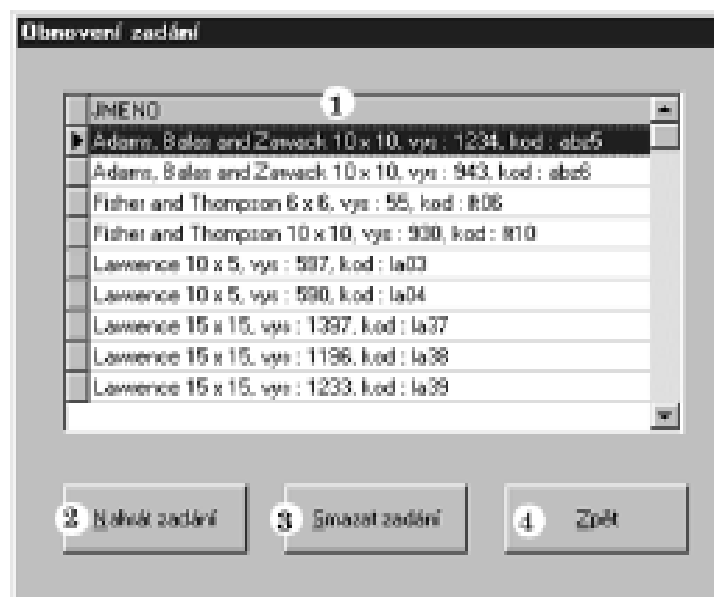
5

6

Obrázek 4.7

4.2.12 Popis formuláře OBNOVENÍ ZÁLOHY

Formulář spustíte tlačítkem OBNOVENÍ ZÁLOHY v hlavní nabídce. Dostanete se do okna jako na obrázku 4.8. Pod číslem 1 se nachází jednotlivé zálohy ve formě databázových vět. Tlačítkem označeným číslem 2 obnovíme naše zadání, tlačítkem označeným číslem 3 smažeme aktivní zadání a tlačítkem označeným číslem 4 se vrátíme zpět.



Obrázek 4.8

4.3 Popis jednotlivých knihoven a procedur

4.3.1 Knihovna HLAVNI

Jedná se o základní knihovnu, jejíž objekty se spouští jako první. Tato knihovna obsahuje pouze formulář TForm1, který je hlavní nabídkou mého programu.

Formulář TForm1 obsahuje tyto globální proměnné :

nastaveni_filtru_operaci : boolean; - tato proměnná naznačuje, zda se má Tabulka TABLE3 při změně aktivní věty v tabulce TABLE2 třídit,
Cesta : string; - uloží celou cestu do pracovního adresáře pro pozdější využití;

tyto zajímavé vlastnosti :

Table1 : TTable; - tabulka obsahující databázový soubor *mashiny.dbf*,
Table2 : TTable; - tabulka obsahující databázový soubor *vyrobky.dbf*,
Table3 : TTable; - tabulka obsahující databázový soubor *operace.dbf*,
Table5 : TTable; - tabulka obsahující databázový soubor *adresar.dbf*,

a tyto zajímavé metody :

procedure DataSource2DataChange(Sender : TObject; Field : TField); - spustí se vždy, když dojde ke změně v tabulce TABLE2, a filtruje tabulku TABLE3 podle aktuálního výrobku,

procedure FormCreate(Sender: TObject); - spustí se při vytvoření fomuláře, vymaže databázové soubory *mashiny.dbf*, *vyrobky.dbf* a *operace.dbf* a nastaví je jako EXKLUSIVE,

function Zprava(obsah : string ;stav : integer) : boolean; - spustí formulář TZpravaForm, pak zobrazí okno, kde je vypsáný *obsah*, a čeká na stisknutí tlačítka *Budiž*, pokud je toto stisknuto, vrátí TRUE, pokud ne, vrátí FALSE,

function Vlozit(obsah : string ;var promena_f : string) : boolean;
- spustí formulář TVlozitForm, pak zobrazí okno, kde je vypsáný *obsah* a čeká na zadání *promene_f* a na stisknutí tlačítka *Budiž*. Pokud je toto stisknuto, vrátí TRUE, pokud ne, vrátí FALSE.

4.3.2 Knihovna STROJE

Tato knihovna obsahuje pouze formulář TStrojeForm zajišťující práci se stroji.

Formulář TStrojeForm obsahuje tyto metody :

procedure NovyClick(Sender: TObject); - přidá nový stroj,
procedure Zmenit_jmeClick(Sender: TObject); - změni jméno stroje,
procedure Zmenit_serClick(Sender: TObject); - změni seřizovací čas stroje,
procedure SmazatClick(Sender: TObject); - smaže stroj,
procedure KonecClick(Sender: TObject); - ukončí formulář.

4.3.3 Knihovna VYROBKY

Tato knihovna obsahuje pouze formulář TVyrobkyForm zajišťující práci s výrobky.

Formulář TVyrobkyForm obsahuje tyto metody :

procedure NovyClick(Sender: TObject); - přidá nový výrobek,
procedure Zmenit_jmeClick(Sender: TObject); - změni jméno výrobku,
procedure Zmenit_kusClick(Sender: TObject); - změni počet kusů výrobku,
procedure SmazatClick(Sender: TObject); - smaže stroj,
procedure KonecClick(Sender: TObject); - ukončí formulář.

4.3.4 Knihovna OPERACE

Tato knihovna obsahuje pouze formulář TOperaceForm zajišťující práci s operacemi.

Formulář TOperaceForm obsahuje tyto metody :

procedure NovyClick(Sender: TObject); - přidá novou operaci,
procedure Zmenit_jmeClick(Sender: TObject); - změni jméno operace,
procedure Zmenit_casClick(Sender: TObject); - změni výrobní čas operace,
procedure SmazatClick(Sender: TObject); - smaže operaci,
procedure KonecClick(Sender: TObject); - ukončí formulář,
procedure DBGrid2CellClick(Column: TColumn); - spustí se po každém kliknutí na tabulku výrobků a přefiltruje TABLE3.

4.3.5 Knihovna OBNOVIT

Tato knihovna obsahuje pouze formulář TObnovitForm obnovující uložené zadání.

Formulář TObnovitForm obsahuje tyto metody :

procedure NahratClick(Sender: TObject) ; - nahraje vybrané zadání,
procedure smazatClick(Sender: TObject) ; - smaže vybrané zadání,
procedure konecClick(Sender: TObject) ; - ukončí formulář.

4.3.6 Knihovna UVOD

Tato knihovna obsahuje pouze formulář TUvodForm seznamující uživatele s autorem.

Formulář TUvodForm obsahuje tyto metody :

procedure KonecClick(Sender: TObject) ; - ukončí formulář a spustí program.

4.3.7 Knihovna VKLADANI

Tato knihovna obsahuje pouze formulář TVkladForm, který je českou náhradou za příkaz **Input**.

Formulář TVkladForm obsahuje tyto globální proměnné :

promena :string; - proměnná, do které se uloží zadaná hodnota,
text : string; - text, který se vypíše jako výzva v okně formuláře,
kod_navrat : boolean; - návratová hodnota stisku tlačítek *Budiž* nebo *Zrušit*;

a tyto zajímavé metody :

procedure OKButtonClick(Sender: TObject); - kod_navrat bude TRUE,
procedure ZrusitClick(Sender: TObject); - kod_navrat bude FALSE.

4.3.8 Knihovna ZALOHA

Tato knihovna obsahuje pouze formulář TZalohaForm ukládající vytvořené zadání.

Formulář TZalohaForm obsahuje tyto globální proměnné

hodnota_zmacknuti : boolean; - je TRUE, pokud naposledy zmáčknutá klávesa je ENTER;

a tyto zajímavé metody :

procedure Edit1Change(Sender: TObject) ; - spustí se při zadání jména zálohy a zapne tlačítko *Uložit*,
procedure DBGrid1KeyUp(Sender: TObject; var Key: Word; Shift: TShiftState) ; - spustí se stisknutím klávesy, nabídne přepsání aktivní zálohy,
procedure DBGrid1KeyDown(Sender: TObject; var Key: Word; Shift: TShiftState) ; - spustí se puštěním klávesy, hlídá držení klávesy.

4.3.9 Knihovna ZPRAVA

Tato knihovna obsahuje pouze formulář TZpravaForm, který je českou náhradou za příkaz **MessageDlg**.

Formulář TZpravaForm obsahuje tyto globální proměnné :

text : string; - text, který se vypíše jako výzva v okně formuláře,
kod_navrat : boolean; - návratová hodnota stisku tlačítek *Budiž* nebo *Zrušit* ;

a tyto metody :

procedure OKButtonClick(Sender: TObject); - kod_navrat bude TRUE,
procedure ZrusitClick(Sender: TObject); - kod_navrat bude FALSE,
procedure FormActivate(Sender: TObject); - spustí se při načtení formuláře.

4.3.10 Knihovna VYPOCET

Tato knihovna obsahuje pouze formulář TVypocetForm, který obsahuje přípravu výpočtu.

Formulář TVypocetForm obsahuje tyto globální proměnné :

Vypis : TSeznam; - seznam, do kterého se ukládá výsledek před výpisem,
Kontrolni : TSeznam; - seznam operací, u kterých může dojít k kolizi přednosti,
Zaklad : TSeznam; - seznam, do kterého je uloženo celé zadání,
Vysledny_krok, vysledny_cas : integer; - výsledné hodnoty,
cas_zad, kroky, cas_trv : integer; - zadané ukončující podmínky,
typ, n, teplota : integer; - zadané parametry výpočtu,
cas_zacatek : string; - čas začátku výpočtu,
cas_vypoctu : string; - doba výpočtu;

a tyto zajímavé metody :

function Poc_vet_op(masina : integer) : integer; - vrací počet operací daného stroje,
procedure Naplneni_Zaklad; - převede zadání ze souborů *.DBF do seznamu ZAKLAD,
procedure Naplneni_Kontrolni; - naplní seznam KONTROLNI,
procedure Zobraz(var Seznam : TSeznam; typ : integer); - zobrazí daný seznam,
procedure Kopie (var Seznam1, Seznam2 : TSeznam); - zkopíruje jeden seznam do druhého.

4.3.11 Knihovna PRUBEH

Tato knihovna obsahuje pouze formulář TPrubehForm, který obsahuje samotný výpočet.

Formulář TPrubehForm obsahuje tyto globální proměnné :

Vysledek : TSeznam; - seznam, do kterého se ukládá nejlepší řešení,
Novy : TSeznam; - seznam, do kterého se ukládá nová kombinace řešení,
Stary : TSeznam; - seznam, ve kterém je uloženo řešení pro porovnávání,
Zaklad : TSeznam; - seznam, ve kterém je uloženo zadání,
Zakaz : TSeznam; - seznam, do kterého se ukládají použité varianty,
Kontrolni : TSeznam; - seznam, do kterého se ukládají operace, u kterých by mohlo dojít ke konfliktu přednosti,

cas_zad, kroky, cas_trv : integer; - zadané ukončující podmínky,
typ, n, teplota : integer; - zadané parametry výpočtu,
cas_vys : integer; - nejlepší dosažený čas,
cas_zacatek : string; - čas začátku výpočtu,
cas_vypoctu : string; - doba výpočtu,
Metoda, Met_lok, Soused, Zapis_prubehu, Co_zapisovat : integer; - proměnné
přebírající nastavené parametry z formuláře VYPOCET,
CheckBox1, CheckBox2, CheckBox3 : boolean; - proměnné přebírající nastavené parametry
z formuláře VYPOCET,
Beh : boolean; - proměnná zjišťující stav běhu programu,
zauzleni : boolean; - proměnná zjišťující, zda nedošlo k zauzlení výpočtu,
preruseni : boolean; - proměnná zjišťující přerušení výpočtu,
Graf : Tseznam; - seznam, do kterého se ukládají jednotlivé prvky grafu,
Pomer : Integer; - proměnná vyadřující poměr výšky grafu a velikosti času,
Bod_min_poc : TBod; - pomocná proměnná pro překreslování grafu,
dosavad_kroky : integer; - proměnná zjišťující počet dosavadních iterací,
Stav_prubehu : array [1..20] of integer; - pole hodnot pro zápis do souboru;

a tyto zajímavé metody :

function Kontrola_Casu : boolean; - kontroluje, zda výpočet neběží přesčas,
function Kon_ko : boolean; - kontroluje ukončující podmínky,
procedure Graf_ukaz(cas_nov, kroky : integer); - vykreslí v grafu další hodnotu,
procedure Prohod_Nahodne(var Seznam : TSeznam); - náhodně prohodí operace
na daném stroji,
procedure Prohod(var Seznam : TSeznam; Prvni, Druhy : integer); - prohodí
operaci s pořadovým číslem *První* s operací *Druhý*,
procedure Najdi_Okoli_Tabu(var Seznam : TSeznam; typ : integer); - uloží
do seznamu *Novy* nejlepšího souseda pro metodu *Tabu search*,
procedure Najdi_Okoli_Simul(var Seznam : TSeznam; typ : integer); -
uloží do seznamu *Novy* souseda pro metodu *Simulované žihání*,
procedure Kopie (var Seznam1, Seznam2 : TSeznam); - zkopíruje *Seznam2*
do *Seznam1*,
function Kontrola_Tabu(var Seznam : TList; typ : integer) : boolean; -
zkontroluje *Seznam*, zda vyhovuje *Tabu Search*,
function Kontrola_Simul(var Seznam : TList; typ : integer) : boolean;
- zkontroluje *Seznam*, zda vyhovuje *Simulovanému žihání*,
function Konec (Seznam : Tseznam) : boolean; - signalizuje, zda je již třeba skončit
s výpočtem času,
function Cas(var Seznam : TSeznam; typ : integer) : integer; - spočítá
výsledný čas řešení, a to buď jako prostoje, nebo jako celkový čas podle *typu*,
procedure Tabu; - spustí metodu *Tabu Search*,
procedure Simul; - spustí metodu *Simulované žihání*,
procedure Vsechny; - spustí metodu *Vyzkoušet všechny varianty*,
procedure Lokal; - spustí metodu *Lokálního minima*,
procedure Zapis_do_souboru; - zapíše do souboru *prubeh.txt* průběh výpočtu.

TABU V 6446 6008 5625 5292 4984 4678 4508 4376 4284 4133 4100
 TABU S 6446 5787 5659 5560 5276 5245 5245 5245 5245 5245 5245

5.3 J. Adams, E. Balas and D. Zawack (1988)

10 jobů na 10 strojů

job1 - 7 62 8 24 5 25 3 84 4 47 6 38 2 82 0 93 9 24 1 66
 job2 - 5 47 2 97 8 92 9 22 1 93 4 29 7 56 3 80 0 78 6 67
 job3 - 1 45 7 46 6 22 2 26 9 38 0 69 4 40 3 33 8 75 5 96
 job4 - 4 85 8 76 5 68 9 88 3 36 6 75 2 56 1 35 0 77 7 85
 job5 - 8 60 9 20 7 25 3 63 4 81 0 52 1 30 5 98 6 54 2 86
 job6 - 3 87 9 73 5 51 2 95 4 65 1 86 6 22 8 58 0 80 7 65
 job7 - 5 81 2 53 7 57 6 71 9 81 0 43 4 26 8 54 3 58 1 69
 job8 - 4 20 6 86 5 21 8 79 9 62 2 34 0 27 1 81 7 30 3 46
 job9 - 9 68 6 66 5 98 8 86 7 66 0 56 3 82 1 95 4 47 2 78
 job10 - 0 30 3 50 7 34 2 58 1 77 5 34 8 84 4 40 9 46 6 44

Zadaný výsledek : 943 s

kod : abz6

<u>Metoda</u>	<u>Sous</u>	<u>1</u>	<u>501</u>	<u>1001</u>	<u>1501</u>	<u>2001</u>	<u>2501</u>	<u>3001</u>	<u>3501</u>	<u>4001</u>	<u>4501</u>	<u>5001</u>
LOK_S	V	4526	4266	3922	3723	3577	3487	3421	3375	3311	3267	3182
LOK_S	S	4526	4211									Výpočet ukončen
LOK_N	V	4526	3761	3353	2721	2657	2437	2361	2227	2209	2183	2159
LOK_N	S	4526	4235	4235	4235	4235	4235	4235	4235	4235	4235	4235
SIMUL	V	4526	4067	3711	2970	2959	2746	2323	2157	2112	2050	2050
SIMUL	S	4526	4036	4036	3861	3861	3861	3490	3490	3490	3490	3490
TABU	V	4526	4266	4012	3859	3713	3577	3487	3421	3357	3267	3208
TABU	S	4526	4211	4211	4211	4186	4058	4032	4032	4032	4032	4032

5.4 Fisher and Thompson 1968

6 jobů na 6 strojů

job1 - 2 1 0 3 1 6 3 7 5 3 4 6
 job2 - 1 8 2 5 4 10 5 10 0 10 3 4
 job3 - 2 5 3 4 5 8 0 9 1 1 4 7
 job4 - 1 5 0 5 2 5 3 3 4 8 5 9
 job5 - 2 9 1 3 4 5 5 4 0 3 3 1
 job6 - 1 3 3 3 5 9 0 10 4 4 2 1

Zadaný výsledek : 55 s

kod : ft06

<u>Metoda</u>	<u>Sous</u>	<u>1</u>	<u>501</u>	<u>1001</u>	<u>1501</u>	<u>2001</u>	<u>2501</u>	<u>3001</u>	<u>3501</u>	<u>4001</u>	<u>4501</u>	<u>5001</u>
LOK_S	V	152	116	80	80							Výpočet ukončen
LOK_S	S	152	139									Výpočet ukončen

LOK_N	V	152	104	86	86	86	86	86	86	86	86	86
LOK_N	S	152	139	139	139	139	139	139	139	139	139	139
SIMUL	V	152	135	109	102	102	93	93	93	93	93	93
SIMUL	S	152	124	123	119	119	119	119	119	119	119	119
TABU	V	152	119	80	74	68	61	59	59	58	58	58
TABU	S	152	123	122	103	96	96	92	81	78	78	78

5.5 Fisher and Thompson 1963

10 jobů na 10 strojů

job1 - 0 29 1 78 2 9 3 36 4 49 5 11 6 62 7 56 8 44 9 21
 job2 - 0 43 2 90 4 75 9 11 3 69 1 28 6 46 5 46 7 72 8 30
 job3 - 1 91 0 85 3 39 2 74 8 90 5 10 7 12 6 89 9 45 4 33
 job4 - 1 81 2 95 0 71 4 99 6 9 8 52 7 85 3 98 9 22 5 43
 job5 - 2 14 0 6 1 22 5 61 3 26 4 69 8 21 7 49 9 72 6 53
 job6 - 2 84 1 2 5 52 3 95 8 48 9 72 0 47 6 65 4 6 7 25
 job7 - 1 46 0 37 3 61 2 13 6 32 5 21 9 32 8 89 7 30 4 55
 job8 - 2 31 0 86 1 46 5 74 4 32 6 88 8 19 9 48 7 36 3 79
 job9 - 0 76 1 69 3 76 5 51 2 85 9 11 6 40 7 89 4 26 8 74
 job10 - 1 85 0 13 2 61 6 7 8 64 9 76 5 47 3 52 4 90 7 45

Zadaný výsledek : 930 s

kod : ft10

<u>Metoda</u>	<u>Sous</u>	<u>1</u>	<u>501</u>	<u>1001</u>	<u>1501</u>	<u>2001</u>	<u>2501</u>	<u>3001</u>	<u>3501</u>	<u>4001</u>	<u>4501</u>	<u>5001</u>
LOK_S	V	3394	3021	2839	2713	2570	2430	2316	2187	2125	2052	1973
LOK_S	S	3394	3280									
LOK_N	V	3394	2987	2407	2168	1841	1818	1789	1789	1704	1690	1513
LOK_N	S	3394	3280	3280	3280	3280	3280	3280	3280	3280	3280	3280
SIMUL	V	3394	2547	2301	2235	2145	2144	2144	2144	2144	2120	2005
SIMUL	S	3394	3020	3020	3020	3020	3020	3020	3020	3020	3020	3020
TABU	V	3394	3146	2964	2838	2695	2525	2411	2316	2187	2052	2011
TABU	S	3394	3207	3207	3207	3207	3207	3207	3207	3207	2998	2923

5.6 Lawrence 1968

10 jobů na 5 strojů

job1 - 1 23 2 45 0 82 4 84 3 38
 job2 - 2 21 1 29 0 18 4 41 3 50
 job3 - 2 38 3 54 4 16 0 52 1 52
 job4 - 4 37 0 54 2 74 1 62 3 57
 job5 - 4 57 0 81 1 61 3 68 2 30
 job6 - 4 81 0 79 1 89 2 89 3 11
 job7 - 3 33 2 20 0 91 4 20 1 66
 job8 - 4 24 1 84 0 32 2 55 3 8

job9 - 4 56 0 7 3 54 2 64 1 39

job10 - 4 40 1 83 0 19 2 8 3 7

Zadaný výsledek : 597 s

kod : la03

<u>Metoda</u>	<u>Sous</u>	<u>1</u>	<u>501</u>	<u>1001</u>	<u>1501</u>	<u>2001</u>	<u>2501</u>	<u>3001</u>	<u>3501</u>	<u>4001</u>	<u>4501</u>	<u>5001</u>
LOK_S	V	1579	1358	1194	941	876	870	812	799	Výpočet ukončen		
LOK_S	S	1579	1579	Výpočet ukončen								
LOK_S	V	1579	1230	1070	971	955	955	905	891	891	891	848
LOK_S	S	1579	1579	1579	1579	1579	1579	1579	1579	1579	1579	1579
SIMUL	V	1579	1152	1051	1051	1051	1051	1044	993	967	967	967
SIMUL	S	1579	1480	1480	1442	1442	1442	1415	1380	1380	1380	1380
TABU	V	1579	1358	1236	983	897	870	814	799	769	769	769
TABU	S	1579	1370	1360	1360	1360	1360	1360	1360	1314	1314	1314

5.7 Lawrence 1984

10 jobů na 5 strojů

job1 - 0 12 2 94 3 92 4 91 1 7

job2 - 1 19 3 11 4 66 2 21 0 87

job3 - 1 14 0 75 3 13 4 16 2 20

job4 - 2 95 4 66 0 7 3 7 1 77

job5 - 1 45 3 6 4 89 0 15 2 34

job6 - 3 77 2 20 0 76 4 88 1 53

job7 - 2 74 1 88 0 52 3 27 4 9

job8 - 1 88 3 69 0 62 4 98 2 52

job9 - 2 61 4 9 0 62 1 52 3 90

job10 - 2 54 4 5 3 59 1 15 0 88

Zadaný výsledek : 590 s

kod : la04

<u>Metoda</u>	<u>Sous</u>	<u>1</u>	<u>501</u>	<u>1001</u>	<u>1501</u>	<u>2001</u>	<u>2501</u>	<u>3001</u>	<u>3501</u>	<u>4001</u>	<u>4501</u>	<u>5001</u>
LOK_S	V	2195	1773	1539	1272	1099	992	931	918	903	850	843
LOK_S	S	2195	1969	Výpočet ukončen								
LOK_N	V	2195	1973	1921	1844	1824	1792	1779	1776	1776	1764	1755
LOK_N	S	2195	2005	2005	2005	2005	2005	2005	2005	2005	2005	2005
SIMUL	V	2195	2557	1035	1035	969	969	969	941	941	941	936
SIMUL	S	2195	1916	1916	1686	1556	1434	1434	1434	1434	1434	1434
TABU	V	2195	1808	1539	1357	1176	992	941	921	914	850	843
TABU	S	2195	1969	1969	1955	1828	1824	1824	1824	1824	1824	1820

5.8 Lawrence 1968

15 jobů na 15 strojů

job1 - 5 19 6 64 11 73 9 13 2 84 14 88 3 85 10 41 12 53 13 80 1 66 7 46 8 59 4 25 0 62

job2 - 1 67 3 74 7 41 2 57 14 52 0 14 9 64 8 84 6 78 5 47 13 28 4 84 10 63 12 26 11 46
 job3 - 6 97 8 95 0 64 9 38 10 59 12 95 2 17 11 65 13 93 3 10 5 73 1 11 4 85 14 46 7 67
 job4 - 10 23 12 49 3 32 4 66 2 43 0 60 8 41 7 61 13 70 9 49 11 17 6 90 1 85 14 99 5 85
 job5 - 9 98 8 57 3 73 6 9 0 73 7 7 1 98 4 13 13 41 5 40 11 85 10 37 2 68 14 79 12 17
 job6 - 11 66 7 53 5 86 6 40 0 14 3 19 13 96 4 95 2 54 10 84 12 97 8 16 14 52 1 76 9 87
 job7 - 4 77 2 55 9 42 5 74 14 91 13 33 10 16 12 54 0 18 3 87 7 60 8 13 6 33 1 33 11 61
 job8 - 6 41 5 39 11 82 9 64 14 47 10 28 7 78 13 49 1 79 4 58 2 92 3 79 12 6 0 69 8 76
 job9 - 11 21 5 42 9 91 2 28 0 52 6 88 12 76 13 86 10 23 1 35 7 52 4 91 3 47 14 82 8 24
 job10 - 11 42 1 93 3 95 13 45 9 28 14 77 0 84 10 8 7 45 4 70 5 37 6 86 12 64 8 67 2 38
 job11 - 4 97 12 81 1 58 7 84 5 58 0 9 11 87 3 5 2 44 13 85 6 89 10 77 9 96 14 39 8 77
 job12 - 12 80 1 21 10 10 5 73 8 70 6 49 2 31 13 34 4 40 11 22 0 15 14 82 3 57 9 71 7 48
 job13 - 2 17 7 62 5 75 9 35 1 91 14 50 3 7 10 64 13 75 12 94 0 55 6 72 8 47 4 11 11 90
 job14 - 11 93 6 57 1 71 12 70 9 93 5 20 3 15 13 77 10 58 0 12 2 67 8 68 14 7 7 29 4 52
 job15 - 13 76 3 27 4 26 9 36 11 8 10 36 0 95 8 48 2 82 6 87 5 6 1 63 7 56 12 36 14 15

Zadaný výsledek : 1397 s

kod : la37

<u>Metoda</u>	<u>Sous</u>	<u>1</u>	<u>501</u>	<u>1001</u>	<u>1501</u>	<u>2001</u>	<u>2501</u>	<u>3001</u>	<u>3501</u>	<u>4001</u>	<u>4501</u>	<u>5001</u>
LOK_S	V	10072	9946	9882	9793	9667	9603	9538	9412	9191	9191	9065
LOK_S	S	10072	9977									
LOK_S	V	10072	9692	9418	8647	8531	8102	7721	7721	7442	7095	7049
LOK_S	S	10072	9977	9977	9977	9977	9977	9977	9977	9977	9977	9977
SIMUL	V	10072	8926	8836	8733	8378	8378	8258	7857	7295	7086	7073
SIMUL	S	10072	10072	9549	9549	9549	9549	9549	9549	9549	9549	9549
TABU	V	10072	10072	10072	10072	9793	9793	9793	9538	9538	9538	9191
TABU	S	10072	9977	9808	9549	9549	9549	9549	9549	9549	9549	9549

5.9 Lawrence 1968

15 jobů na 15 strojů

job1 - 1 26 12 67 0 72 6 74 14 13 8 43 4 30 3 19 10 23 11 85 5 98 13 43 2 38 7 8 9 75
 job2 - 14 42 0 39 4 55 12 46 1 19 8 93 9 80 5 26 10 7 6 50 11 57 3 73 2 9 7 61 13 72
 job3 - 3 96 4 99 12 34 6 60 7 43 14 7 13 12 8 11 11 70 10 43 0 91 1 68 9 11 5 68 2 72
 job4 - 14 63 11 45 4 49 1 74 8 27 0 30 9 72 7 9 12 99 13 60 5 69 6 69 2 84 3 40 10 59
 job5 - 2 91 0 75 9 98 3 17 10 72 13 31 11 9 14 98 7 50 5 37 4 8 8 65 1 90 12 91 6 71
 job6 - 11 35 6 80 4 39 3 62 14 74 5 72 10 35 9 25 1 49 8 52 7 63 2 90 13 21 12 47 0 38
 job7 - 14 19 7 57 10 24 13 91 3 50 0 5 11 49 12 18 9 58 5 24 8 52 1 88 2 68 6 20 4 53
 job8 - 7 77 14 72 5 35 11 90 4 68 6 18 3 9 0 33 8 60 10 18 12 10 13 60 1 38 2 99 9 15
 job9 - 13 6 8 86 2 40 9 79 12 92 11 23 5 89 10 95 6 91 7 72 0 80 1 60 3 56 4 51 14 23
 job10 - 1 46 6 28 5 34 11 77 4 47 0 10 14 49 8 77 10 48 7 24 12 8 2 72 13 55 9 29 3 40
 job11 - 10 22 4 89 12 79 0 7 9 15 1 6 11 30 6 38 5 11 8 52 3 20 7 5 14 9 2 20 13 28
 job12 - 5 73 14 56 2 37 3 22 13 25 6 58 1 8 7 93 4 88 8 17 12 9 11 69 10 71 9 85 0 55
 job13 - 9 85 14 58 3 46 8 64 2 49 6 37 1 33 4 30 5 26 0 20 13 74 10 77 12 99 11 56 7 21
 job14 - 10 17 3 24 4 89 5 15 11 60 1 42 8 98 2 64 13 92 0 63 7 52 12 54 6 75 14 23 9 38
 job15 - 3 8 5 17 11 56 7 93 14 26 9 62 6 7 10 88 0 97 1 7 2 43 8 29 13 35 12 87 4 57

Zadaný výsledek : 1196 s

kod : la38

<u>Metoda</u>	<u>Sous</u>	<u>1</u>	<u>501</u>	<u>1001</u>	<u>1501</u>	<u>2001</u>	<u>2501</u>	<u>3001</u>	<u>3501</u>	<u>4001</u>	<u>4501</u>	<u>5001</u>
LOK_S	V	8873	8676	8332	8332	8135	7935	7935	7738	7738	7738	7556
LOK_S	S	8873	8667	8590	8463	8453	Výpočet ukončen					
LOK_N	V	8873	8684	7785	7785	7785	7541	6822	6822	6687	6145	5840
LOK_N	S	8873	8705	8705	8705	8667	8590	8511	8453	8453	8453	8453
SIMUL	V	8873	8500	8372	7813	6967	6518	6477	6434	6434	6433	6305
SIMUL	S	8873	8754	8350	8297	8297	8022	8022	8022	7750	7750	7750
TABU	V	8873	8873	8873	8873	8332	8332	8332	7935	7935	7935	7738
TABU	S	8873	8697	8650	8463	8453	8453	8453	8453	8453	8138	8138

5.10 D. Applegate, W. Cook - B. Gamble 1991

10 jobů na 10 strojů

job1 - 0 96 1 69 2 25 3 5 4 55 5 15 6 88 7 11 8 17 9 82
job2 - 0 11 1 48 2 67 3 38 4 18 7 24 6 62 5 92 9 96 8 81
job3 - 2 67 1 63 0 93 4 85 3 25 5 72 6 51 7 81 8 58 9 15
job4 - 2 30 1 35 0 27 4 82 3 44 7 92 6 25 5 49 9 28 8 77
job5 - 1 53 0 83 4 73 3 26 2 77 6 33 5 92 9 99 8 38 7 38
job6 - 1 20 0 44 4 81 3 88 2 66 6 70 5 91 9 37 8 55 7 96
job7 - 1 21 2 93 4 22 0 56 3 34 6 40 7 53 9 46 5 29 8 63
job8 - 1 32 2 63 4 36 0 26 3 17 5 85 7 15 8 55 9 16 6 82
job9 - 0 73 2 46 3 89 4 24 1 99 6 92 7 7 9 51 5 19 8 14
job10 - 0 52 2 20 3 70 4 98 1 23 5 15 7 81 8 71 9 24 6 81

Zadaný výsledek : 1005 s

kod : orb03

<u>Metoda</u>	<u>Sous</u>	<u>1</u>	<u>501</u>	<u>1001</u>	<u>1501</u>	<u>2001</u>	<u>2501</u>	<u>3001</u>	<u>3501</u>	<u>4001</u>	<u>4501</u>	<u>5001</u>
LOK_S	V	2185	2004	1940	1800	1765	1756	1710	1688	1680	1647	1628
LOK_S	S	2185	2185	Výpočet ukončen								
LOK_N	V	2185	2045	1940	1768	1694	1650	1569	1545	1545	1545	1545
LOK_N	S	2185	2185	2185	2185	2185	2185	2185	2185	2185	2185	2185
SIMUL	V	2185	2185	2185	2185	2093	2093	2008	2008	2008	2008	2008
SIMUL	S	2185	2076	2076	2076	2076	2076	2076	2076	2076	2076	2076
TABU	V	2185	2011	1956	1878	1800	1765	1744	1710	1688	1647	1631
TABU	S	2185	2086	2086	2086	2074	2074	2074	2074	2074	2074	2074

5.11 D. Applegate, W. Cook - B. Shepherd 1991

10 jobů na 10 strojů

job1 - 0 8 1 10 2 35 3 44 4 15 5 92 6 70 7 89 8 50 9 12
job2 - 0 63 8 39 3 80 5 22 2 88 1 39 9 85 6 27 7 74 4 69

job3 - 0 52 6 22 1 33 3 68 8 27 2 68 5 25 4 34 7 24 9 84
 job4 - 0 31 1 85 4 55 8 80 5 58 7 11 6 69 9 56 3 73 2 25
 job5 - 0 97 5 98 9 87 8 47 7 77 4 90 3 98 2 80 1 39 6 40
 job6 - 1 97 5 68 0 44 9 67 2 44 8 85 3 78 6 90 7 33 4 81
 job7 - 0 34 3 76 8 48 7 61 9 11 2 36 4 33 6 98 1 7 5 44
 job8 - 0 44 9 5 4 85 1 51 5 58 7 79 2 95 6 48 3 86 8 73
 job9 - 0 24 1 63 9 48 7 77 8 73 6 74 4 63 5 17 2 93 3 84
 job10 - 0 51 2 5 4 40 9 60 1 46 5 58 8 54 3 72 6 29 7 94

Zadaný výsledek : 1005 s

kod : orb04

<u>Metoda</u>	<u>Sous</u>	<u>1</u>	<u>501</u>	<u>1001</u>	<u>1501</u>	<u>2001</u>	<u>2501</u>	<u>3001</u>	<u>3501</u>	<u>4001</u>	<u>4501</u>	<u>5001</u>
LOK_S	V	4038	3723	3467	3113	2976	2770	2655	2575	2458	2394	2329
LOK_S	S	4038	3696									
LOK_N	V	4038	4038	3324	2556	2251	2246	2186	2109	1915	1899	1796
LOK_N	S	4038	3708	3708	3708	3708	3708	3708	3708	3708	3708	3708
SIMUL	V	4038	3470	3407	2729	2369	2297	2145	2145	2145	2095	2037
SIMUL	S	4038	3627	3627	3627	3627	3627	3627	3627	3627	3627	3627
TABU	V	4038	3723	3467	3242	3105	2899	2770	2655	2575	2394	2329
TABU	S	4038	3696	3696	3696	3696	3610	3610	3610	3610	3610	3610

5.12 D. Applegate, W. Cook - G.Steiner 1991

10 jobů na 10 strojů

job1 - 9 11 8 93 0 48 7 76 6 13 5 71 3 59 2 90 4 10 1 65
 job2 - 8 52 9 76 0 84 7 73 5 56 4 10 6 26 2 43 3 39 1 49
 job3 - 9 28 8 44 7 26 6 66 4 68 5 74 3 27 2 14 1 6 0 21
 job4 - 0 18 1 58 3 62 2 46 6 25 4 6 5 60 7 28 8 80 9 30
 job5 - 0 78 1 47 7 29 5 16 4 29 6 57 3 78 2 87 8 39 9 73
 job6 - 9 66 8 51 3 12 7 64 5 67 4 15 6 66 2 26 1 20 0 98
 job7 - 8 23 9 76 6 45 7 75 5 24 3 18 4 83 2 15 1 88 0 17
 job8 - 9 56 8 83 7 80 6 16 4 31 5 93 3 30 2 29 1 66 0 28
 job9 - 9 79 8 69 2 82 4 16 5 62 3 41 6 91 7 35 0 34 1 75
 job10 - 0 5 1 19 2 20 3 12 4 94 5 60 6 99 7 31 8 96 9 63

Zadaný výsledek : 887 s

kod : orb05

<u>Metoda</u>	<u>Sous</u>	<u>1</u>	<u>501</u>	<u>1001</u>	<u>1501</u>	<u>2001</u>	<u>2501</u>	<u>3001</u>	<u>3501</u>	<u>4001</u>	<u>4501</u>	<u>5001</u>
LOK_S	V	3193	2997	2827	2731	2639	2457	2379	2318	2261	2223	2194
LOK_S	S	3193	3132									
LOK_N	V	3193	2181	2119	2026	1971	1971	1736	1625	1613	1613	1613
LOK_N	S	3193	3131	3131	3131	3131	3131	3131	3131	3131	3131	3131
SIMUL	V	3193	2719	2238	2016	2016	2016	2016	1927	1927	1923	1923
SIMUL	S	3193	3128	3076	3076	3076	3076	3076	3076	3076	3076	3076
TABU	V	3193	3036	2897	2801	2709	2639	2457	2379	2318	2223	2194
TABU	S	3193	3132	3128	3128	3128	3128	2972	2926	2926	2918	2918

5.13 D. Applegate, W. Cook - B.Cook 1991

10 jobů na 10 strojů

job1 - 0 99 1 74 2 49 3 67 4 17 5 7 6 9 7 39 8 35 9 49
job2 - 0 49 3 67 4 82 2 92 1 62 5 84 8 45 6 30 7 42 9 71
job3 - 0 26 3 33 1 82 2 98 5 83 4 16 6 64 7 65 9 36 8 77
job4 - 0 41 1 62 4 73 3 94 6 51 5 46 2 55 9 31 7 64 8 46
job5 - 1 68 0 26 5 50 3 46 4 25 7 88 2 6 8 13 9 98 6 84
job6 - 0 24 6 80 2 91 3 55 1 48 8 99 4 72 9 91 7 84 5 12
job7 - 2 16 3 13 0 9 1 58 4 23 7 85 5 36 6 89 8 71 9 41
job8 - 2 54 0 41 3 38 4 53 1 11 5 74 9 88 6 46 7 41 8 65
job9 - 2 53 1 50 4 40 0 90 7 7 5 80 3 57 9 60 6 91 8 47
job10 - 2 45 0 59 8 81 3 99 6 71 1 19 4 75 7 77 9 94 5 95

Zadaný výsledek : 1010 s

kod : orb06

<u>Metoda</u>	<u>Sous</u>	<u>1</u>	<u>501</u>	<u>1001</u>	<u>1501</u>	<u>2001</u>	<u>2501</u>	<u>3001</u>	<u>3501</u>	<u>4001</u>	<u>4501</u>	<u>5001</u>
LOK_S	V	3644	3369	3043	2730	2627	2531	2431	2381	2309	2259	2249
LOK_S	S	3644	3503									
LOK_N	V	3644	3265	2466	2252	2086	2065	2014	2014	2014	2014	1953
LOK_N	S	3644	3620	3477	3477	3477	3477	3477	3477	3477	3477	3477
SIMUL	V	3644	2835	2579	2453	2285	2135	2135	2059	1936	1936	1936
SIMUL	S	3644	3644	3263	3263	3263	3263	3263	3263	3182	2938	2938
TABU	V	3644	3369	3139	2911	2730	2627	2527	2431	2375	2259	2249
TABU	S	3644	3503	3503	3503	3503	3503	3503	3503	3503	3503	3491

5.14 Storer, Wu, and Vaccari hard 1992

10 jobů na 10 strojů

job1 - 2 16 0 59 4 10 3 95 1 64 8 92 9 56 7 3 5 73 6 17
job2 - 1 5 4 64 3 30 2 14 0 96 9 11 8 73 7 35 6 93 5 12
job3 - 3 35 4 75 0 54 1 30 2 83 9 20 8 29 7 38 6 90 5 39
job4 - 4 29 3 21 0 52 2 93 1 20 5 5 7 11 8 53 9 56 6 98
job5 - 0 17 3 16 4 41 1 78 2 100 5 55 8 27 6 2 7 87 9 55
job6 - 3 97 1 32 4 84 2 71 0 38 9 64 7 16 5 5 6 41 8 41
job7 - 3 41 1 57 4 37 0 64 2 92 6 19 9 47 7 94 8 79 5 21
job8 - 0 23 3 67 1 39 4 98 2 63 8 83 5 45 6 89 9 81 7 44
job9 - 1 88 0 59 3 39 2 63 4 91 8 36 5 44 6 45 9 43 7 12
job10 - 2 29 1 17 0 6 3 74 4 51 9 14 6 2 5 56 7 49 8 14
job11 - 3 75 2 10 4 1 0 35 1 99 7 56 5 95 9 78 6 53 8 82
job12 - 0 75 2 96 1 21 3 90 4 55 6 23 7 40 9 76 8 55 5 45
job13 - 3 90 4 64 0 72 2 33 1 59 7 51 6 74 5 85 9 76 8 38
job14 - 3 57 1 84 2 87 4 2 0 68 8 4 5 77 6 37 7 37 9 94
job15 - 1 16 3 46 4 34 2 23 0 77 7 68 8 14 9 54 5 37 6 99

job16 - 4 24 1 73 2 92 0 43 3 42 5 81 7 99 8 88 9 80 6 5
 job17 - 1 56 2 51 0 3 4 87 3 25 5 62 7 11 8 88 6 68 9 29
 job18 - 2 85 3 3 4 21 0 49 1 79 8 38 5 37 9 72 7 18 6 18
 job19 - 0 2 3 55 1 31 2 29 4 98 5 92 6 43 8 99 7 67 9 41
 job20 - 4 69 3 64 0 61 1 13 2 31 5 6 8 84 9 94 7 32 6 54

Zadaný výsledek : 1493 s

kod : swv04

<u>Metoda</u>	<u>Sous</u>	<u>1</u>	<u>501</u>	<u>1001</u>	<u>1501</u>	<u>2001</u>	<u>2501</u>	<u>3001</u>	<u>3501</u>	<u>4001</u>	<u>4501</u>	<u>5001</u>
LOK_S	V	4081	3978	3978	3963	3963	3860	3860	3860	3860	3860	3809
LOK_S	S	4081	3978					Výpočet ukončen				
LOK_N	V	4081	3868	3567	3506	3375	3248	3231	3033	2978	2899	2764
LOK_N	S	4081	3978	3978	3978	3978	3978	3978	3978	3978	3978	3978
SIMUL	V	4081	4070	3705	3705	3705	3705	3603	3603	3603	3603	3603
SIMUL	S	4081	3929	3929	3929	3929	3929	3929	3929	3929	3929	3929
TABU	V	4081	4081	4081	4081	3963	3963	3963	3963	3860	3860	3860
TABU	S	4081	3978	3932	3932	3932	3932	3932	3932	3932	3930	3930

5.15 Storer, Wu, and Vaccari hard 1992

10 jobů na 10 strojů

job1 - 2 19 1 30 3 80 0 84 4 14 8 51 5 73 6 91 7 81 9 71
 job2 - 2 74 4 79 1 39 0 7 3 66 9 6 5 93 8 76 6 21 7 76
 job3 - 4 90 3 33 1 38 2 73 0 61 8 61 7 76 5 86 9 28 6 35
 job4 - 4 1 3 22 2 1 0 77 1 33 6 98 5 4 9 27 8 8 7 68
 job5 - 2 63 4 5 1 95 0 7 3 50 8 46 9 28 6 70 5 60 7 34
 job6 - 0 98 1 73 4 15 3 21 2 32 7 24 9 9 8 24 5 7 6 34
 job7 - 3 51 4 47 2 30 1 16 0 51 5 41 6 79 7 79 9 3 8 72
 job8 - 4 3 1 59 0 53 3 20 2 19 6 20 9 16 7 90 5 96 8 18
 job9 - 1 34 2 55 3 97 0 93 4 90 7 81 5 63 8 41 6 1 9 51
 job10 - 4 77 3 87 1 92 2 83 0 45 7 75 9 60 6 75 5 93 8 33
 job11 - 0 31 2 66 1 58 4 17 3 94 5 63 7 80 9 61 6 78 8 52
 job12 - 4 70 1 25 2 75 0 89 3 41 7 100 5 73 6 28 8 94 9 88
 job13 - 1 67 4 62 3 12 2 55 0 62 5 58 8 66 7 73 6 55 9 1
 job14 - 4 81 0 37 1 2 3 39 2 17 7 74 6 71 8 61 5 42 9 5
 job15 - 3 62 0 31 4 63 2 31 1 5 9 7 7 77 8 34 6 34 5 3
 job16 - 0 5 2 55 3 62 1 82 4 80 6 6 8 7 7 29 5 80 9 89
 job17 - 3 26 1 50 2 58 0 22 4 68 7 12 6 9 9 34 5 90 8 87
 job18 - 0 50 2 28 1 64 4 34 3 63 7 9 9 48 6 63 8 61 5 2
 job19 - 0 47 2 23 1 23 4 82 3 98 7 66 6 78 8 100 9 79 5 32
 job20 - 1 13 4 14 0 90 2 77 3 80 9 30 7 31 5 36 6 51 8 69

Zadaný výsledek : 1448 s

kod : swv05

[illegible]

6. Závěr

Po prostudování vypočtených hodnot dojdeme k závěru, že náš program se hodí pro návrhy menších sestav maximálně 10 jobů na 10 strojů. Větší sestavy by pravděpodobně, alespoň podle posloupnosti hodnot jednotlivých metod, byly relevantní při použití rychlejších počítačů a delší výpočetní lhůty, což v dnešní době rychle se obnovující výpočetní techniky není až takový problém. Při hodnocení jednotlivých metod snadno dojdeme k závěru, že naprosto nepoužitelná je metoda lokálního minima s nejstrmějším spádem se sousedstvím SHIFT. Při tomto řešení dojde k ukončení výpočtu zdaleka nejrychleji. Ostatní metody se liší podle jednotlivých zadání, obecně však lze doporučit Tabu search, Simulované žíhání pro složitější problémy a Lokální minimum s náhodným spádem pro jednodušší, vše se sousedstvím vytvořeným prohozením dvou operací. Jak je vidět problém job shop rozvrhování je velmi složitý a náročný na teoretické zpracování. Jeho úspěšné řešení ovšem sebou přináší vysoké úspory finančních zdrojů, a proto se domnívám, že v dnešní době bude výzkum problematiky job shop rozvrhování pokračovat neztěněnou silou.

SEZNAM POUŽITÉ LITERATURY :

- [1] Parker, G. C. : Deterministic Scheduling Theory. Chapman & Hall, London 1995.
- [2] Reeves, C.R. (ed.) : Modern Heuristic Techniques for Combinatorial problems Blackwell Scientific Publications 1993.
- [3] Glover, F. - Laguna, M. : Tabu Search. Kluwer Academic Publisher 1997.
- [4] Cheng, R. - Gen, M. - Tsujimura, Y. : A Tutorial Survey of Job - Shop Scheduling Problems Using Genetic Algorithms, Computers ind. Engng, 30, 1996, no. 4, pp. 984 - 997.
- [5] Nowicki, E. - Smutnicki, C. : A Fast Taboo Search Algorithm for the Job Shop Problem, Management Science, vol. 42, 1996, no.4, pp. 797 - 812.
- [6] Krishna, K. - Ganesan, K. - Janaki Ram, D. : Distributed Simulated Annealing Algorithms for Job Shop Scheduling, IEE Transactions on Systems, vol. 25, 1995, no.7, pp. 1102 - 1109
- [7] Mamalis, A. - Malagardis, I. : Determination of Due Dates in Job Shop Scheduling By Simulated Annealing, Computer Integrated Manufacturing Systems, vol. 9, 1996, č. 2, str. 65 -72.
- [8] Sun, D. - Batta, R.. : Effective Job Shop Scheduling Through Active Chain Manipulation, Computers Ops, vol. 22, 1995, no. 2, pp. 159 - 172.
- [9] Blazewicz, J. - Eckerker, H. - Pesch, E. : Scheduling Computer and Manufacturing Process, Springer - Verlag, Berlin, 1993.
- [10] Dorndorf, J. - Pesch, E. : Evolution Based Learning in Job Shop Scheduling Environment, Computers Ops, vol. 22, 1995, no. 1, pp. 25 - 40..