*VYSOKÁ ŠKOLA EKONOMICKÁ V PRAZE*

*FAKULTA INFORMATIKY A STATISTIKY*




Název bakalářské práce:

# Aplikace genetického algoritmu na rozvrhování výroby strojírenské firmy

| | |
|---|---|
| Autor: | **Jiří Stariat** |
| Katedra: | **Katedra ekonometrie** |
| Obor: | **Matematické metody v ekonomii** |
| Vedoucí práce: | **Ing. Veronika Skočdopolová, Ph.D.** |

**Prohlášení**:

Prohlašuji, že jsem bakalářskou práci na téma „Aplikace genetického algoritmu na rozvrhování výroby strojírenské firmy“ zpracoval samostatně. Veškerou použitou literaturu a další podkladové materiály uvádím v seznamu použité literatury.

V Praze dne 25. května 2016 .....

Jiří Stariat

**Poděkování**:

Rád bych na tomto místě poděkoval Ing. Veronice Skočdopolové, Ph.D. za vedení mé bakalářské práce. Dále bych také rád poděkoval Ing. Janu Zouharovi, Ph.D. za podnětné návrhy, které tuto práci obohatily. Rovněž děkuji firmě NSCZ, s.r.o. za poskytnutí dat, na základě kterých je tato práce vypracována.

## Abstrakt

*Název práce*: Aplikace genetického algoritmu na rozvrhování výroby strojírenské firmy
*Autor*: Jiří Stariat
*Katedra*: Katedra ekonometrie
*Vedoucí práce*: Ing. Veronika Skočdopolová, Ph.D.

Tato práce se zabývá rozvrhovací úlohou, jejími speciálními typy a metodami jejího řešení. Rozvrhovací úloha je jedním ze základních problémů operačního výzkumu, řadí se mezi kombinatorické úlohy. Cílem rozvrhovací úlohy je přiřadit určité činnosti a zdroje jednotlivým časovým okamžikům. Rozvrhovací úloha patří do třídy NP-úplných úloh. Má tedy tak vysokou výpočetní náročnost, že v současné době není znám algoritmus, který by exaktně vyřešil její libovolnou instanci v polynomiálním čase. Proto se k jejímu řešení, hlavně rozsáhlejších úloh, využívá heuristických a metaheuristických metod. V této práci je podrobně popsána zejména metaheuristická metoda genetického algoritmu, jejíž aplikace na rozvrhování výroby konkrétní strojírenské firmy, je předmětem této práce.



**Klíčová slova**: Genetický algoritmus, rozvrhování, metaheuristiky, operační výzkum, aplikace v praxi



## Abstract

*Title*: Application of genetic algorithm for production scheduling of engineering company
*Author*: Jiří Stariat
*Department*: Department of Econometrics
*Supervisor*: Ing. Veronika Skočdopolová, Ph.D.

This thesis is engaged in scheduling problem, his special types and methods of solving. Scheduling problem is a common operations research problem, which ranks among combinatorial problems. The aim of the scheduling problem is to assign certain activities and resources to individual time moments. Scheduling problem is NP-complete problem. Its computational complexity is thus so high, that there is currently no known algorithm that precisely solve its any instance in polynomial time. Is therefore used for its solution heuristics and metaheuristcs. In this thesis is described in detail metaheuristics of genetic algorithm. Application of genetic algorithm for production scheduling of specific engineering company is the main objective of this thesis.



**Keywords**: Genetic algorithm, scheduling, metaheuristic, operations research, application in practice

# Obsah

**ÚVOD** ..... **1**

**1 ROZVRHOVACÍ ÚLOHA** ..... **2**

1.1 FORMULACE OBECNÉ ROZVRHOVACÍ ÚLOHY ..... 2
1.2 SPECIÁLNÍ TYPY ROZVRHOVACÍCH ÚLOH ..... 4

**2 METODY ŘEŠENÍ** ..... **8**

2.1 OBTÍŽNOST ..... 8
2.2 EXAKTNÍ ALGORITMY ..... 9
2.3 HEURISTICKÉ ALGORITMY ..... 11
2.4 METAHEURISTICKÉ ALGORITMY ..... 12

**3 GENETICKÝ ALGORITMUS** ..... **14**

3.1 TEORIE SCHÉMAT ..... 15
3.2 NO-FREE-LUNCH (NFL) ..... 16
3.3 KOMPONENTY GENETICKÉHO ALGORITMU ..... 16

**4 PROGRAMOVACÍ PROSTŘEDÍ MATLAB** ..... **23**

**5 PRAKTICKÁ ČÁST** ..... **25**

5.1 PŘEDSTAVENÍ SPOLEČNOSTI ..... 25
5.2 ZADÁNÍ PROBLÉMU ..... 25
5.3 ŘEŠENÍ PROBLÉMU ..... 27

**ZÁVĚR** ..... **32**

**LITERATURA** ..... **33**

## Seznam tabulek

Tabulka 2-2-1: Vztah mezi dobou trvání a velkostí vstupu *n* ..... 8
Tabulka 3-1: Turnajový výběr ..... 19
Tabulka 5-1: Stroje ..... 26
Tabulka 5-2: Výrobky-ilustrační příklad ..... 27
Tabulka 6-1: Výrobky 1. část ..... 35
Tabulka 6-2: Výrobky 2. část ..... 36
Tabulka 6-3: Výrobky 3. část ..... 37
Tabulka 6-4: Výrobky 4. část ..... 38
Tabulka 6-5: Výrobky 5. část ..... 39

# Seznam obrázků

Obrázek 1-1: Příklad job-shop 3×3 ..... 5
Obrázek 3-1: Binární reprezentace, pořadí činností ..... 18
Obrázek 3-2: Jednobodové křížení pro $k = 4$ ..... 20
Obrázek 3-3: Dvoubodové křížení pro $k_1 = 3$, $k_2 = 8$ ..... 20
Obrázek 3-4: Nelineární křížení ..... 21
Obrázek 5-1: Původní Ganttův diagram ..... 29

# Úvod

Téma mé bakalářské práce jsem se snažil zvolit tak, aby mělo nějaký praktický přínos. Shodou okolností bylo potřeba v jedné z firem spjatých s naší rodinou zjednodušit a zautomatizovat rozvrhování výroby. Rozhodl jsem se tedy, že se pokusím ve své bakalářské práci rozvrhnout výrobu této strojírenské firmy.

Cílem mé bakalářské práce je tedy rozvrhnout výrobu konkrétní strojírenské firmy NSCZ. Abych byl schopen tento cíl splnit, budu nejprve muset splnit dílčí cíle. Mezi ně patří nastudovat problematiku rozvrhovací úlohy a zvolit který její speciální typ je aplikovatelný na mou praktickou úlohu. V dalším kroku budu muset zvolit vhodnou metodu pro řešení tohoto speciálního typu rozvrhovací úlohy a teprve poté se tuto metodu pokusím aplikovat na rozvrhování výroby společnosti NSCZ.

První kapitola této práce bude tedy věnována rozvrhovací úloze a jejím speciálním typům. V druhé kapitole popíši metody řešení tohoto kombinatorického problému. Rozdělím tyto metody do tří skupin: na exaktní, heuristické a metaheuristické. Takto rozdělené skupiny charakterizuji a uvedu některé jejich zástupce. Třetí kapitolu pak celou věnuji metaheuristické metodě genetického algoritmu, která bude následně v praktické části použita i na řešení rozvrhování výroby společnosti NSCZ. Ve čtvrté kapitole uvedu stručný popis programového prostředí MATLAB, ve kterém bude řešena výpočetní část rozvrhování výroby NSCZ. Praktická část této práce bude uvedena v kapitole páté. V této kapitole představím společnost NSCZ, uvedu požadavky na rozvrhování výroby této společnosti a popíši její řešení.

Věřím, že tato práce splní svůj účel a výrazně zjednoduší proces plánování výroby ve firmě NSCZ s.r.o. a doufám, že tato bakalářská práce může být přínosem i pro další studenty.

# 1 Rozvrhovací úloha[1]

Teorie rozvrhování se věnuje úlohám časového rozmístění aktivit na konkrétní zařízení. Příkladem rozvrhovací úlohy je například rozvrh výroby výrobků.

## 1.1 Formulace obecné rozvrhovací úlohy

Obecně je rozvrhovací úloha s omezenými zdroji definována následovně:

Mějme

- množinu aktivit, které musí být vykonány,
- množinu zdrojů, s jejichž pomocí aktivity vykonáme,
- množinu omezení, která musí být dodržena,
- optimalizační kritérium.

Cílem rozvrhovací úlohy je přiřadit jednotlivým aktivitám zdroje a aktivity přiřadit časovým intervalům, tedy vytvořit rozvrh. Takový rozvrh, který bude splňovat veškeré omezující podmínky, rozvrhne veškeré aktivity a zároveň bude hodnota jeho optimalizačního kritéria alespoň tak dobrá, jako hodnota optimalizačního kritéria, kteréhokoli jiného přípustného rozvrhu.

V obecné formě má úloha následující vlastnosti:

- každá aktivita může být vykonána jedním nebo více způsoby, podle toho jaké zdroje využívá,
- každá aktivita může mít libovolný počet precedenčních aktivit, ale může začít, až když jsou hotovy všechny aktivity jí předcházející,
- každá aktivita může být v jejím průběhu přerušena,
- každá aktivita může využívat jeden, nebo více různých zdrojů,
- množství zdroje, které aktivita spotřebovává, se může v jejím průběhu měnit,
- zdroje mohou být obnovitelné (např. stroje) nebo neobnovitelné (např. materiály),
- dostupnost zdrojů se může lišit v čase,
- zdroje mohou mít časová omezení.

Aby bylo možné zachytit nejistotu běžnou v reálných problémech, obecnou formulaci rozšíříme o následující vlastnosti:

- dostupnost zdrojů se může v čase měnit,
- požadavky na zdroje se můžou v čase měnit,
- cíle se mohou měnit (optimalizační kritérium).

V další části bude rozvrhovací úloha podrobněji rozebrána z pohledu aktivit, zdrojů, omezení a optimalizačního kritéria.

[1] Není-li uvedeno jinak, kapitola vychází ze zdroje [1].

## Aktivity

Každá aktivita má měřitelné odhady výkonnostních kritérií, jako je doba trvání, náklady s ní spojené, nebo množství zdrojů, které spotřebuje. Aktivita může vyžadovat jeden nebo více zdrojů, jejichž spotřeba se může v průběhu trvání aktivity měnit. Aktivita může mít více způsobů provedení, podle toho jaké a kolik zdrojů využívá. Například pokud je daná aktivita prováděna na jednom stroji nebo pokud bude prováděna souběžně na více strojích, bude mít rozdílné doby trvání, požadavky na zdroje a náklady s ní spojené. Aktivita může být tvořena jednou či více částmi. Aktivitu můžeme mezi jednotlivými částmi přerušit a dokončit později. Jednotlivé části můžeme během jejich průběhu přerušit pouze někdy (např. pokud je na fréze nastaven a spuštěn program na výrobu komponenty, může být zastaven až po jeho dokončení, nikoli v průběhu frézování). Spotřeba zdroje aktivitou může být fixní po celou dobu trvání projektu (např. obsluha stroje), fixní na začátku projektu (např. nájem za stroje), proměnná (např. platba nějaká částky za každou hodinu kdy je aktivita vykonávána), nebo kombinace těchto možností.

## Zdroje

Zdroje dělíme na obnovitelné a neobnovitelné. Obnovitelné zdroje sice mohou být v každém časovém úseku dostupné pouze v omezeném množství, ale v dalším časovém úseku se jejich dostupnost znovu obnoví. Typickým příkladem obnovitelných zdrojů je lidská práce. Jakmile jsou však vyčerpány neobnovitelné zdroje, v dalším časovém úseku už je nemáme k dispozici, což může mít za následek až nepřípustnost rozvrhu. Příkladem neobnovitelných zdrojů je kapitál. Zdroje se mohou lišit také ve schopnostech, v ceně a v dalších měřitelných charakteristikách. Například jeden pracovní tým, může být efektivnější než druhý, dostupnost zdrojů se může v čase lišit a zdroje také nemusí být k dispozici v důsledku nepředvídatelných poruch či havárií.

## Omezující podmínky a optimalizační kritérium

Omezující podmínky a optimalizační kritérium se definují během formulace úlohy. Omezující podmínky určují přípustnost rozvrhu, optimalizační kritérium určuje jeho optimalitu. Omezením týkajícím se rozvrhování aktivit jsou především precedenční podmínky, tedy požadavek na dokončení jiných aktivit nebo jejich částí před zahájením dané aktivity. Dále mohou být některé aktivity omezeny časově, například demolice budov v obydlených částech nemohou být prováděny ve večerních hodinách. Omezení mohou být také zdrojově orientovaná. Zdroje bývají omezeny buďto časově, máme je k dispozici pouze v určitý čas, nebo množstevně, máme jich omezené množství. Omezení se mohou týkat i hodnoty optimalizačního kritéria, které může být omezeno horní mezí. Přípustný nebo také konzistentní rozvrh je takový rozvrh, který splňuje veškeré omezující podmínky. Zatímco optimální rozvrh je přípustný a zároveň má nejlepší hodnotu optimalizačního kritéria. Jako optimalizační

kritérium bývá nejčastěji volena minimalizace celkové doby projektu. Nicméně za optimalizační kritérium může být zvolena i minimalizace nákladů, maximální čistá současná hodnota budoucí cashflow a řada dalších. V reálných aplikacích většinou používáme více optimalizačních kritérií, která jsou téměř vždy konfliktní. Například pokud se minimalizuje doba trvání, s největší pravděpodobností se zvýší náklady. Čím více optimalizačních kritérií bude při hodnocení rozvrhu použito, tím více konfliktů vznikne, proto je nutné stanovit vztah mezi jednotlivými kritérii, na jehož základě bude možné rozhodnout, které kritérium má v danou chvíli vyšší důležitost.

### Rozvrhování jako dynamická úloha

U většiny rozvrhů a plánů je už od prvního okamžiku jasné, že se budou v průběhu projektu měnit. Rozvrh zůstává stejný pouze v okamžiku, kdy je projekt dokončen. Je to způsobeno tím, že není možné vytvořit odhad, který by přesně kopíroval realitu. V reálném projektu totiž musíme čelit například zpoždění dodávek, zpoždění prací, poruchám na strojích, nemocem zaměstnanců a dalším náhodným jevům. Proto je důležité, aby se optimalizátor dokázal dobře přizpůsobit změnám a přitom bral v potaz již hotovou práci.

## 1.2 Speciální typy rozvrhovacích úloh

Vzhledem k tomu, že definice obecné rozvrhovací úlohy je velice obsáhlá, existují další konkrétněji definované úlohy, které nám umožní za cenu zjednodušení modelované reality snížit velikost prohledávaného prostoru. Těmito úlohami jsou úlohy typu job-shop, open-shop, flow-shop a rozvrhování projektů.

### Rozvrhování projektů

V problémech rozvrhování projektů se jeden projekt skládá ze souboru aktivit, které jsou určeny precedenčními vztahy, odhadovanými délkami trvání jednotlivých aktivit, popřípadě mohou obsahovat další omezení, jako například omezení nákladů. Za optimální rozvrh nejčastěji považujeme rozvrh s minimální celkovou délkou.

*Rozvrhování projektů s omezenými zdroji* je speciální úlohou rozvrhování projektů, při které má každá aktivita určité požadavky na spotřebu zdrojů, jejichž množství je omezené.

Dalším případem je *multimodální rozvrhování projektů s omezenými zdroji*, v tomto případě mohou být aktivity vykonány více než jedním způsobem a každý způsob má rozdílné požadavky na zdroje a jinou dobu trvání. Musí se tedy rozhodnout, jak bude aktivita provedena, takže se v procesu objevuje i plánování.

*Multiprojektové rozvrhování* má za úkol vytvořit rozvrh pro více současně probíhajících projektů [2].

## Job-shop

Cílem je nalezení optimálního rozvrhu pro *n* úloh na *m* strojích. Kde každá úloha prochází stroji v předem daném pořadí, čas strávený na každém stroji může být i nulový. Jako optimální se automaticky bere rozvrh s minimální celkovou délkou. Stroje nemusí být v provozu po celou dobu, tudíž se do celkové délky započítává kromě času stráveného na strojích i délka prostojů. Jednotlivé úlohy mohou být zpracovávány na různých strojích, mít rozdílné náklady, nebo mít různou časovou náročnost.

Formálně můžeme úlohu zapsat takto [3, str. 134]. Mějme množinu *n* úloh $\{J_i\}_{1 \le i \le n}$, které jsou zpracovávány na m strojích $\{M_j\}_{1 \le j \le m}$. Mějme matici **X** typu $n \times m$, kde *i*-tý řádek určuje pořadí, v jakém *j*-tá úloha prochází jednotlivými stroji. Každá úloha je přiřazena každému stroji právě jednou a každý stroj může zpracovávat pouze jednu úlohu najednou. Zpracování úlohy $J_i$ na stroji $M_j$, označujme jako operaci $O_{ij}$. Dále mějme matici **P** typu $n \times m$, kde prvek $p_{ij}$ je nepřetržitý časový interval, po který je zpracovávána operace $O_{ij}$, přičemž $p_{ij} \geq 0$. Rozvrhem potom rozumíme množinu dokončených časů pro každou operaci $\{C_{ij}\}_{1 \le i \le n,\ 1 \le j \le m}$, která vyhovuje omezujícím podmínkám. Účelová funkce bude označována $C_{max}$, a může být popsána jako čas potřebný k uskutečnění všech úloh, neboli celkový čas rozvrhu. Cílem této úlohy je minimalizace účelové funkce $C_{max}$. Příklad, jak může být zadána úloha typu job-shop, je na obrázku 1-1. Reprezentace řešení tohoto problému pomocí Ganttova diagramu, je na obrázku 1-2.

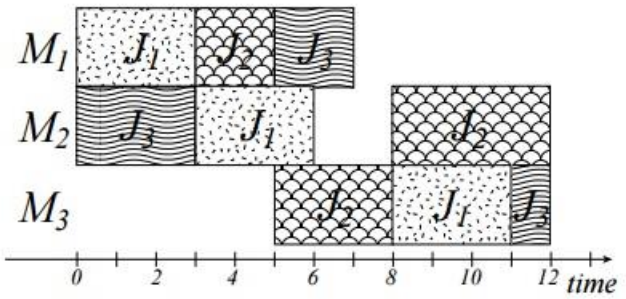
Obrázek 1-1: Příklad job-shop 3×3

Příklad úlohy job-shop 3×3

| činnosti | pořadí zdpracování činností na strojích j (doby trvání) | | |
|---|---|---|---|
| 1 | 1 (3) | 2 (3) | 3 (3) |
| 2 | 1 (2) | 3 (3) | 2 (4) |
| 3 | 2 (3) | 1 (2) | 3 (1) |

Zdroj: [3], vlastní zpracování

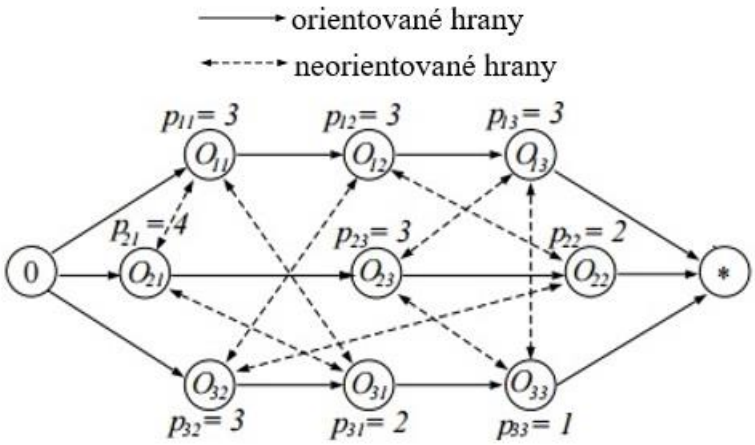
Obrázek 1-2: Reprezentace řešení problému 3×3 pomocí Ganttova diagramu




Zdroj: [3]

Úloha typu job-shop, může být formálně popsána také pomocí grafu G = (V,C ∪ D), kde V je množina uzlů, odpovídající činnostem, které jsou vykonávané na strojích, rozšířená o uzel S, reprezentující začátek a uzel T, reprezentující konec rozvrhu, C je množina orientovaných hran, zastupujících precedenční podmínky, D je množina neorientovaných hran zastupujících dvojice činností, které jsou vykonávány na stejném stroji. Grafová reprezentace úlohy typu job-shop, je na obrázku 1-3., kde každý uzel $O_{ij}$, má ohodnocení $p_{ij}$, které vyjadřuje dobu vykonávání činnosti *i* na stoji *j*. Řešením úlohy je nalezení takového pořadí operací na každém stroji, při kterém bude celkový rozvrh minimální. Přípustného řešení lze docílit nahrazením orientovaných hran neorientovanými tak, aby v grafu nevznikl cyklus [3, str. 135].

Obrázek 1-3: Grafová reprezentace úlohy job-shop 3×3




Zdroj: [3], vlastní zpracování

## Open-shop

Rozvrhovací problém open-shop je definován podobně jako job-shop, pouze s tím rozdílem, že u open-shopu nezáleží na pořadí, v jakém úlohy stroji procházejí. Pořadí úloh na strojích je tedy také předmětem optimalizace.

Formálně je tedy možné problém open-shop definovat takto: mějme množinu strojů $M_j$, kde $j = 1, 2, \ldots, m$, množinu úloh $J_i$, kde $i = 1, 2, \ldots, n$ a matici **P** typu $n \times m$, jejíž prvky $p_{ij}$, vyjadřují dobu po kterou je úloha $J_i$ zpracovávána strojem $M_j$, přitom $p_{ij} \geq 0$ a účelovou funkci $C_{max}$, lze interpretovat jako „celkovou délku rozvrhu“. Každá úloha je přiřazena každému stroji právě jednou, vykonávání úlohy nemůže být přerušeno, úloha může být v každém časovém intervalu vykonávána pouze na jednom stroji a každý stroj může během jednoho časového intervalu vykonávat pouze jednu úlohu. Řešením problému open-shop je pak takové přiřazení úloh a strojů jednotlivým časovým intervalům, aby celková délka rozvrhu byla minimální [4].

## Flow-shop

Flow-shop je speciálním případem problému job-shop, kde všechny úlohy musejí být zpracovány na strojích ve stejném pořadí, a každá úloha musí být zpracována na každém stroji.

Formální definice problému flow-shop. Mějme množinu strojů $M_j$, kde $j = 1, 2, \ldots, m$, množinu úloh $J_i$, kde $i = 1, 2, \ldots, n$ a vektor **x** délky $m$, který je tvořen posloupností strojů, které mají být použity pro realizaci úlohy $J_i$, přičemž každý stroj je úloze přiřazen vždy právě jednou. Mějme také matici **P** typu $n \times m$, jejíž prvek $p_{ij}$, vyjadřuje čas po který je úloha $J_i$ vykonávána na stroji $M_j$, přitom $p_{ij} > 0$ a účelovou funkci $C_{max}$ lze interpretovat jako „celkovou délku rozvrhu“. Každá úloha je přiřazena každému stroji právě jednou, vykonávání úlohy nemůže být přerušeno, úloha může být v každém časovém intervalu vykonávána pouze na jednom stroji a každý stroj může během jednoho časového intervalu vykonávat pouze jednu úlohu. Řešením problému flow-shop je pak takové přiřazení úloh a strojů jednotlivým časovým intervalům, aby celková délka rozvrhu byla minimální [4].

# 2 Metody řešení

Tato kapitola je věnována metodám řešení rozvrhovací úlohy. V úvodu se zmiňuji o obtížnosti a výpočetní složitosti rozvrhovací úlohy. Dále pak uvádím zástupce exaktních, heuristických i metaheuristických metod.

## 2.1 Obtížnost

Rozvrhovací úloha je obtížná na řešení. Jedním z důvodů je i to, že je potřeba velkého množství dat, aby bylo možné sestavit model a definovat všechny vztahy mezi aktivitami a zdroji. Tou hlavní věcí, která činí rozvrhovací úlohu obtížnou na řešení je ale její kombinatorický charakter a z toho plynoucí důsledky pro její výpočetní složitost, a tedy i řešitelnost.

### Výpočetní složitost

Pro porovnání rychlosti a efektivity algoritmů se využívá asymptotická složitost. Asymptotická složitost zjišťuje, jakým způsobem se bude měnit chování algoritmu v závislosti na změně vstupních dat. Zapisuje se pomocí Landauovy notace, nebo také Omikron notace. Tento zápis vyjadřuje horní mez pro délku běhu algoritmu v závislosti na velikosti vstupu, více viz [5]. Za efektivní jsou považovány takové algoritmy, které „běží v polynomiálním čase“, tedy je možné jejich výpočetní složitost vyjádřit jako O(p($n$)), kde p($n$) je polynomiální funkce vstupu. Tyto algoritmy patří do třídy P-těžkých algoritmů, kam patří všechny algoritmy, které jsou v polynomiálním čase řešitelné na deterministickém Turingově stroji. Turingův stroj je teoretický počítač, který popsal Matematik Alan Turing [6]. Algoritmy z vyšších tříd složitosti nejsou považované za efektivní, kvůli vysoké časové náročnosti, která je činí prakticky nepoužitelnými, což je dobře vidět z tabulky 2-1, která popisuje vztah mezi dobou výpočtu a velikostí vstupu $n$, při rychlosti $10^6$ operací za sekundu.

Tabulka 2-2-1: Vztah mezi dobou trvání a velikostí vstupu $n$

| | **velikost $n$** | | | | | |
|---|---|---|---|---|---|---|
| **složitost** | 10 | 20 | 30 | 40 | 50 | 60 |
| $n$ | .00001s | .00002 s | .00003 s | .00004 s | .00005 s | .00006 s |
| $n^2$ | .0001 s | .0004 s | .0009 s | .0016 s | .0025 s | .0036 s |
| $n^3$ | .001 s | .008 s | .027 s | .064 s | .125 s | .216 s |
| $n^5$ | .1 s | 3.2 s | 24.3 s | 1.7 min | 5.2 min | 13 min |
| $2^n$ | .001 s | 1 s | 17.9 min | 12.7 dní | 35.7 let | 366 století |
| $3^n$ | .059 s | 58 min | 6.5 let | 3855 století | $2\times10^8$ století | $1.3\times10^{13}$ století |

Zdroj [4], vlastní tvorba

Kombinatorické úlohy, do kterých se řadí i rozvrhovací úloha, jsou ze třídy NP-úplných úloh. To jsou takové nedeterministicky polynomiální problémy, na které jsou polynomiálně redukované všechny ostatní problémy z NP. Nedeterministicky polynomiální problémy, jsou takové problémy, které lze v polynomiálně omezeném čase řešit na nedeterministickém Turingově stroji. Tyto úlohy jsou charakteristické tím, že v současnosti není znám algoritmus, který by exaktně vyřešil libovolnou instanci v polynomiálním čase, lze u nich pouze pro dodaný výsledek ověřit jeho správnost v polynomiálním čase. Dokonce pokud by platilo, že třídy asymptotické složitosti P ≠ NP, tokový algoritmus by ani existovat nemohl, více viz [5]. Tuto vlastnost mají také všechny úlohy, na který jde libovolný NP-úplný problém redukovat, tedy i job-shop, open-shop, flow-shop a další.

Z výše uvedeného je jasně vidět, že pro praktické úlohy, které mají často počty aktivit a zdrojů v řádu stovek, či tisíců, je exaktní postup prohledávání prostoru nepoužitelný, zejména pokud vezmeme v úvahu, že praktické úlohy mají dynamický charakter a potřebu změny rozvrhu v závislosti na nepředvídatelných událostech i několikrát denně. Vzniká tedy potřeba algoritmu, který najde v krátkém čase „dostatečně dobré“ řešení. Tuto potřebu naplňují heuristické algoritmy [2].

### Jak pracují heuristické algoritmy

Na řešení rozvrhovací úlohy se používají dva druhy heuristických algoritmů, konstruktivní a prohledávací. Konstruktivní algoritmy tvoří rozvrh tak, že postupně přidávají jednotlivé úlohy. Prohledávací algoritmy vygenerují náhodné řešení, které potom prohledáváním stanoveného prostoru upravují a vylepšují.

Prohledávání prostoru je obtížné nejen kvůli jeho rozsáhlosti a mnohorozměrnosti, ale zejména kvůli jeho narušené souvislosti, způsobené omezujícími podmínkami. Vzniká tak poměrně řídký, nesouvislý prostor s velkým množstvím lokálních extrémů. Je důležité, aby prohledávací algoritmy neměly sklon k předčasné konvergenci v lokálních optimech. Musí být navrženy tak, aby vhodně kombinovaly prohledávání v celé šíři prostoru a lokální vylepšování [2].

## 2.2 Exaktní algoritmy

Exaktní algoritmy jsou algoritmy, které dospějí k optimálnímu řešení vždy, pokud existuje alespoň jedno přípustné řešení. Jak je ale uvedeno v kapitole výše, nemusí k němu dospět v polynomiálním čase. Pokud přípustné řešení neexistuje tak exaktní metody dojdou ke zjištění, že žádné řešení neexistuje. Při použití heuristických metod takovou jistotu nemáme.

Nejzákladnějším přístupem exaktních metod je prohledávání celého prostoru a vybrání nejlepšího řešení. Spousta exaktních metod je ale navržena tak, aby neprohledávala neperspektivní oblasti prostoru řešení a vyhnula se tak tomuto neefektivnímu přístupu.

## Metoda CPM

Příkladem exaktní metody je metoda kritické cesty, neboli CPM (Critical Path Method). Metoda CPM nalezne časové rezervy a činnosti kritické pro celkovou dobu projektu. Kritické činnosti jsou takové činnosti, jejichž celková časová rezerva RC je rovna 0.

Mějme následující informace o projektu:

- Seznam činností, které mají být rozvrženy,
- dobu trvání každé činnosti,
- precedenční vztahy mezi činnostmi
- časové lhůty pro dokončení činností.

Na základě těchto informací, metoda CPM zavádí následující charakteristiky činností:

- Nejdříve možné začátky činností,
- nejdříve možné konce činností,
- nejpozději přípustný konec činnosti,
- nejpozději přípustný začátek činnosti.

Z těchto charakteristik lze spočítat časové rezervy jednotlivých činností jako rozdíl nejdříve možného začátku, nejpozději přípustného konce a doby trvání činnosti. Činnosti s nulovou časovou rezervou tvoří takzvanou *kritickou cestu*. Činnosti ležící na kritické cestě jsou určující pro celkovou délku projektu. Pokud se nějaká z těchto činností zpozdí, zpozdí se i projekt, jelikož celková doba projektu se rovná celkové době kritické cesty [7].

Metoda CPM pracuje s fixním bodovým odhadem dob trvání jednotlivých činností a neomezenými zdroji. Pro odstranění těchto nedostatků byly vyvinuty například metoda PERT (Program Evaluation and Review Technique), či metoda CCPM (Critical Chain Project Management). Metoda PERT bere na rozdíl od metody CPM doby trvání, již jako náhodné veličiny. Metoda CCPM je zobecněním metody CPM, která bere v potaz omezenost zdrojů. Více o metodách PERT a CPM viz [7]. O metodě CCPM je možné si přečíst více v práci [8].

## Metoda větví a mezí

Metoda větví a mezí, nebo také *branch-and-bound* systematicky prohledává stanovený prostor tak, že ho postupně dělí na menší části a neperspektivní části. Z hlediska optimality už ho dále neprohledává. Myšlenkou metody je strom, který začíná u kořene, kde obsahuje množinu všech řešení, strom se rozvětví, tím se množina řešení rozdělí na podmnožiny a ty neperspektivní větve se prořežou, čímž se zmenší prohledávaný prostor. Algoritmus v prvním kroku provede větvení (*branching*), tedy rozdělí prohledávaný prostor do jednotlivých větví, které při sjednocení dají původní stanovený prostor. Tím vytvoří graf stromu, jehož uzly reprezentují jednotlivá řešení. V dalším kroku algoritmus spočítá hodnoty dolních mezí všech větví (*bounding*). V posledním kroku, kterému se říká prořezávání (*pruning*), algoritmus porovnává hodnoty dolních mezí jednotlivých větví a ty, které vykazují nejnižší hodnoty dolních mezí zachová, zbytek „prořeže“, tedy vyřadí z dalšího prohledávání. Takto algoritmus pokračuje, dokud nenajde řešení, jehož hodnota dolní meze není vyšší než hodnota kterékoli jiné dolní meze. O tom jak použít metodu větví a mezí na rozvrhovací úlohu pojednává

například práce „*A branch and bound algorithm for the job-shop scheduling problem"* od Petera Bruckera, Bernda Jurische a Bernda Sieverse [9].

## 2.3 Heuristické algoritmy

Tyto druhy algoritmů využívají k hledání techniky, které jsou založeny na předchozích zkušenostech s úlohami stejného typu. Těmto technikám se říká heuristiky a algoritmy jsou schopny s jejich pomocí nalézt „dobré" řešení. Neposkytují však žádnou záruku optimality řešení a to že heuristika nenajde přípustné řešení, nutně neznamená, že takové řešení neexistuje. I přes tyto problémy jsou heuristiky hojně využívány, protože jsou schopny často najít suboptimální řešení v relativně krátkém čase.

Heuristická pravidla jsou u rozvrhovacích úloh hojně používána zejména na rozvrhovací úlohy s omezenými zdroji. Těchto heuristik existuje celá řada, 113 je jich uvedeno například v práci [10]. Porovnání řešení některých heuristických pravidel s optimálními řešeními 83 rozvrhovacích úloh je k vidění v práci [11]. Níže jsou uvedeny tři heuristiky používané pro řešení rozvrhovacích úloh, které dosahují dobrých výsledků [11].

### Greatest Resource Demand (GRD)[2]

Tato metoda pracuje tak, že do rozvrhu přednostně zařadí činnost s nejvyšší hodnotou $p_j$, kde $p_j$ je velikost spotřeby zdrojů, $p_j = d_j \sum_{i=1}^{m} r_{ij}$, kde $d_j$ představuje délku činností, $r_{ij}$ je spotřeba zdroje $i$ činností $j$ za jednotku času a $m$ představuje počet zdrojů.

### Resource Scheduling Method (RSM)

Pravidlo RSM přednostně zařazuje do rozvrhu činnosti $i$ s minimální hodnotou $d_{ij}$, kde $d_{ij}$ představuje nárůst celkové délky projektu, pokud aktivita $j$ následuje po aktivitě $i$, $d_{ij} = Max[(EFT_i - LST_j)]$, kde $EFT_i$ představuje nejdříve možný konec činnosti $i$ *a* $LST_j$ představuje nejpozdější přípustný začátek činnosti $j$. Takto jsou porovnány všechny dvojice činností.

### Minimum Job Slack (MINSLK)

MINSKL pracuje tak, že pokud vznikne konfliktní požadavek na zdroje, zařadí do rozvrhu přednostně aktivitu s minimální časovou rezervou (*slack*), kterou spočítá metodou kritické cesty. Časová rezerva aktivit je v každém kroku přepočítána a u zpožděných aktivit je snížena o jejich zpoždění.

[2] Veškeré heuristické metody v této kapitole, jsou převzaty ze zdroje [11].

### Minimum Late Finish Time (LFT)

Tato metoda pracuje tak, že při konfliktním požadavku na zdroje, zařadí do rozvrhu první činnost, která má na kritické cestě dřívější, nejpozději přípustný konec činnosti.

### Minimum Least Total Float (LTF)

Tato heuristika pracuje stejně jako MINSLK, s tím rozdílem, že se nijak nebere v úvahu případné zpoždění nejdříve možných začátků na kritické cestě.

O dalších heuristických algoritmech, jako například *Greatest Resource Utilization, Shortest Imminent Operation, Random, Reschedule* je možné si přečíst více v pracích [10] a [11].

## 2.4 Metaheuristické algoritmy

Typickým znakem metaheuristických algoritmů prohledávání stanoveného prostoru je, že na rozdíl od deterministických algoritmů pro prohledávání stanoveného prostoru, jako je například gradientní algoritmus [12], se metaheuristické algoritmy při prohledávání stanoveného prostoru nepřesouvají vždy do sousedního stavu s nejlepší hodnotou účelové funkce. Pravděpodobnost, že bude navštíven každý stav je u metaheuristických algoritmů tedy rovná 0. Přestože metaheuristických algoritmů existuje celá řada, viz [13], níže bude uveden pouze seznam vybraných algoritmů, plus stručný popis jak pracuje algoritmus simulovaného žíhání. Celá další kapitola bude pak věnována podrobnému popisu genetického algoritmu.

### Seznam vybraných metaheuristik[3]

- Tabu Search
- Iterated Local Search
- Hybrid Optimization Algorithms
- Scatter Search
- Particle Swarm Optimization
- Trees and Genetic Programming
- Island Models
- Master-Slave Fitness Assessment
- Niching: Diversity Maintenance Methods
- Non-Dominated Sorting
- Genetic Algorithm
- Simulated Annealing

[3] Popis těchto metod je uveden v práci [13].

## Simulované žíhání (SA)

Simulované žíhání (*Simulated annealing*) je pravděpodobnostní optimalizační metoda prohledávání stanoveného prostoru, která je založena na simulaci žíhání oceli. Tato metoda se snaží zabránit uvíznutí v lokálním minimu tím, že provádí i změny k horšímu, velké hlavně v počátku. Velikost změny je ovlivněna teplotou, čím je teplota vyšší, tím větší změny se provádějí. Tento algoritmus pracuje pouze s jedním kandidátem na řešení. Na rozdíl od obyčejného gradientního algoritmu, který přijímá pouze řešení lepší než stávající, SA přijímá s určitou pravděpodobností i řešení horší než stávající. Pravděpodobnost přijetí i horšího řešení se odvíjí od teploty. Existuje zde přímá závislost mezi teplotou a přijetím horšího řešení. Teplota je v průběhu algoritmu postupně snižována podle rychlosti konvergence. Když algoritmus konverguje rychle, teplota se snižuje také rychle a tím je algoritmu bráněno v pokračování do stavu s horší hodnotou. Pokud algoritmus konverguje pomalu, teplota je snižována také pomalu, aby se zabránilo uvíznutí v lokálním minimu. Více o metodě simulovaného žíhání je uvedeno například v práci [14].

# 3 Genetický algoritmus

Genetický algoritmus (GA) je metaheuristická metoda prohledávání stanoveného prostoru, která na rozdíl od například simulovaného žíhání neoperuje pouze s jediným aktuálním potenciálním řešením, ale s celou množinou těchto řešení, takzvanou *populací*. Za použití GA je možné řešit úlohy, jejichž potenciální řešení je permutací určitých prvků a lze mu podle určitých kritérií přiřadit numerické ohodnocení. Těmito úlohami jsou například úloha obchodního cestujícího, úloha tvorby optimální strategie, či rozvrhovací úloha [2].

Termín genetický algoritmus byl poprvé použit Johnem Hollandem v jeho knize „*Adaptation in Natural and Artificial Systems"* z roku 1975. Hollandova práce byla pro rozvoj GA velice důležitá, ale podobné myšlenky se objevily, už v 60. letech v Německu v práci „*Evolutionsstrategie*" (Ingo Rechenberg a Hans-Paul Schwefel) a v USA v práci „*Evolutionary Programing*" (Lawrence J. Fogel). Společným rysem těchto myšlenek bylo použití mutace a selekce na populaci virtuálních jedinců, což vedlo k její adaptaci. Tato myšlenka je inspirována neodarwinovskou teorií evoluce. Přesto se ale této myšlence podařilo nějakých slibných výsledků dosáhnout až 80. a 90. letech 20. století, zejména díky nárůstu výpočetního výkonu, který je z důvodu značné výpočetní náročnosti GA nezbytný [15].

GA si z genetiky vypůjčil nejen mechanismus, ale také terminologii. Populací je myšlena množina potenciálních řešení. Všichni jedinci jsou kódováni řetězcem symbolů, tyto řetězce se nazývají chromozomy. Tímto genetickým kódem (*genotypem*) je reprezentována původní, proměnná, jejíž hodnotu (*fenotyp*) optimalizujeme. Formálně tedy vektor potenciálních řešení x z prostoru řešení $\mathcal{X}$ reprezentuje řetězec *s* o délce *l*, který je tvořen symboly za pomoci zobrazení

$$\mathrm{r}: \mathcal{A}^l \rightarrow \mathcal{X}.$$

Tradiční, nikoli však jediná možná reprezentace, je reprezentace binární, která využívá abecedu $\mathcal{A} = \{0, 1\}$. Algoritmem tedy není prohledáván prostor fenotypů $\mathcal{X}$, ale prostor genotypové reprezentace $\mathcal{A}^l$.

Je nasimulována evoluce populace. Při této evoluci se v každé iteraci (*generaci*) kvalitní jedinci, kteří reprezentují dobrá řešení, rozmnožují a nekvalitní vymírají. K rozpoznání kvality jedince (*fitness*) slouží účelová funkce, která na základě fenotypu určuje jeho kvalitu:

$$f: \mathcal{X} \rightarrow \mathbb{R}.$$

Jedinci se reprodukují takzvaným *křížením*, které zajišťuje výměnu informací mezi kvalitními jedinci. Pro lepší představu, kříží-li se chromozomy $s_1 = (a_1, a_2, a_3, a_4, a_5)$ a $s_2 = (b_1, b_2, b_3, b_4, b_5)$, jejich potomci budou mít chromozomy buďto $s'_1 = (b_1, b_2, a_3, a_4, a_5)$, nebo $s'_1 = (a_1, a_2, a_3, b_4, b_5)$. O to, aby se zabránilo uvíznutí v globálním optimu, se stará další genetický operátor *mutace*, tím, že vnáší do populace diversitu. *Mutace* funguje tak, že náhodně upraví náhodně vybraného jedince, například u jedince $s'_1 = (1, 0, 0, 1, 1, 0)$ změní druhý gen v chromozomu, takže $s'_1 = (1, 1, 0, 1, 1, 0)$ [2].

Jak pracuje genetický algoritmus [2]:

1. V čase $t = 0$, vytvoří počáteční populaci potencionálních řešení P*(t),* o velikosti *n*, kde $P(t) = \{s_1^t, \dots, s_n^t\}$.
2. Každého jedince $s_n^t$ dekóduje a ohodnotí účelovou funkcí.
3. Vybere jedince pro křížení s pravděpodobností proporcionální k jejich ohodnocení z kroku 2. a křížením vytvoří nové jedince.
4. Algoritmus vybere jedince pro mutaci a provede ji.
5. V čase $t = t + 1$ vytvoří novou populaci.
6. Pokud jsou splněny podmínky ukončení, algoritmus končí, jinak se vrací ke kroku 2.

Tento popis slouží pouze pro představu, jak genetický algoritmus pracuje, pro praktickou implementaci algoritmu je třeba specifikace dalších věcí, které budou projednány níže.

## 3.1 Teorie schémat

Pomocí teorie schémat John Holland vysvětluje, proč je výhodnější prohledávat prostor binární reprezentace potencionálních řešení $\mathcal{A}^l$, než prostor potencionálních řešení $\mathcal{X}$ a proč GA konverguje k suboptimálnímu řešení [15].

Tato myšlenka stojí na třech hlavních pilířích. Pro její porozumění je stěžejní se nejprve seznámit s tím, co je to schéma. Schéma je podmnožinou prostoru $\mathcal{A}^l$, ve které chromozomy sdílejí svou určitou část. Schéma reprezentuje řetězec symbolů z binární abecedy a zástupný znak „*“ o který je abeceda rozšířena $\mathcal{A} = \{0, 1\} \cup *$. Například schéma (1, *, *, 1) reprezentuje všechny tyto řetězce (1, 0, 0, 1), (1, 1, 1, 1), (1, 1, 0, 1), (1, 0, 1, 1) [15].

První z Hollandových myšlenek je myšlenka *implicitního paralelismu*, tedy představa, že informace o mnoha schématech mohou být získávány souběžně. Za určitých podmínek, které závisí na velikosti populace a vlastnostech schématu, Holland odhaduje, že populace o velikosti M, obsahuje informace o schématech velikosti $O(M^3)$. Nicméně se ukázalo, že ve skutečnosti schémata nemohou být zpracovávána souběžně, protože nezávislé odhady jejich kvality *fitness* nelze získat obecně[15].

Druhou Hollandovou myšlenkou je, že lze odvodit dolní mez počtu výskytů, konkrétního schématu v příští generaci GA. Holland ukázal, že pokud je $N(S, t)$, počet instancí schématu *S* v generaci *t*, pak bude očekávaný počet instancí *S* v příští generaci *t*+1 omezen zdola podle vztahu

$$E[N(S, t + 1)] \geq \frac{F(S,t)}{F(t)} N(S, t)[1 - \varepsilon(S, t),$$

ve kterém $F(S,t)$ představuje průměrnou kvalita *fitness* schématu *S*, $\bar{F}(t)$ představuje průměrnou kvalitu populace, $\varepsilon(S, t)$ je vyjádření pravděpodobnosti zničení schématu *S* pomocí genetických operátorů křížení a mutace, která roste s délkou schématu. Tím, že ocení stochastickou a dynamickou povahu tohoto stavu, došel Holland k poněkud extravagantnímu závěru, že počet krátkých nadprůměrných schémat v populaci exponenciálně roste. Nicméně

tento vzorec platí pouze pro jednu následující generaci, protože v dalších generacích už F(*S*,*t*), $\bar{F}(t)$, $\varepsilon(S,t)$ nejsou nezávislé od zbytku populace [15].
Třetí Hollandovou myšlenkou je, že kombinace malých kousků genotypů (dobrých schémat) do větších řetězců, by byl dobrý způsob pro hledání optimálních řešení [15]. Tuto myšlenku ale vyvrací Goldberg, ve své práci [16].

## 3.2 No-Free-Lunch (NFL)

Tato věta, jejíž autory jsou D. Wolpert a W. Macready říká, že žádná optimalizační metoda není lepší než náhodné prohledávání, pokud posuzujeme její průměrnou úspěšnost na všech typech optimalizačních úloh [17]. Z toho vyplývá, že pro úspěšnost algoritmu je důležité, aby byl přizpůsoben konkrétní úloze. U GA je možné uplatnit informace, které o úloze víme, u výběru reprezentace, nebo při tvorbě genetických operátorů křížení a mutace. Například implementace GA pro rozvrhovací úlohu bude jiná, než implementace GA pro úlohu tvorby optimální strategie [2].

## 3.3 Komponenty Genetického algoritmu

### Reprezentace

Reprezentace určuje velikost a vlastnosti prostoru, který bude GA prohledávat. Určuje také, jak bude probíhat konstrukce genetických operátorů křížení a mutace. Nejčastěji se používá binární reprezentace, ale je možné použít také reálnou reprezentaci.

**Binární reprezentace**

Jak již bylo uvedeno, binární reprezentace je nejčastěji používaná reprezentace. Na této reprezentaci Holland postavil teorii schémat, viz kapitola 3.1. Binární reprezentace je založena na abecedě $\mathcal{A} = \{0, 1\}$. Algoritmem tedy není prohledáván prostor fenotypů $\mathcal{X}$, ale prostor genotypové reprezentace $\mathcal{A}^l$, viz kapitola 3. Ve prospěch binární reprezentace hovoří také teorie implicitního paralelismu, viz kapitola 3.1 a to, že za pomocí binární abecedy lze kódovat i největší počet schémat pomocí jednoho bitu informací [18].

Binární reprezentace je vhodná zejména v případech, kdy je ohodnocení jedinců účelovou funkcí výpočetně náročné. Je vhodná z důvodu, že používá abecedu s nízkou kardinalitou, tedy proto, aby byli různé geny v populaci dostatečně zastoupeny, stačí menší populace, než v případě, že by byla použita abeceda s vysokou kardinalitou. V případě použití abecedy s vysokou kardinalitou a výpočetně náročným ohodnocením jedinců účelovou funkcí, by mohlo dojít k tomu, že by bylo znemožněno prohledávání celého prostoru řešení. Naopak při použití binární reprezentace s malou populací, tento problém nevzniká [2].

Naopak při prohledávání spojitého prostoru vysoké dimenze, u kterého je navíc potřebná vysoká přesnost, je binární reprezentace nevhodná. V těchto případech jsou nutné velice dlouhé chromozomy, které se ale projeví velkou výpočetní i paměťovou náročností [2].

Negativní jevy spojené s binární reprezentací jsou pak takzvaný *Hamming cliff* a redundance. Hamming cliff - je situace, ve které se binární kódy dvou po sobě jdoucích hodnot liší na každé pozici (*„například řetězec „*01111*“ reprezentující hodnotu* 15*, „*10000*“ reprezentující hodnotu* 16“ [2]). Tato situace totiž může negativně ovlivnit schopnost GA lokálních vylepšení řešení a jeho následné konvergenci k optimu. Redundance nastávají v případech, kdy je kódovaný parametr z diskrétní množiny s kardinalitou, jež není druhou mocninou přirozeného čísla. Některé z možných chromozomů pak nereprezentují potenciální řešení, jsou tedy redundantní („*například reprezentujeme-li celé číslo z množiny* {0, 1, ..., 10} *čtyřbitovým řetězcem, kód „*1111*“ je redundantní“* [2]). Jak se vypořádat s redundantním chromozomem pokud se objeví v populaci, popisuje práce [15].

**Reálná reprezentace**

Na rozdíl od binární reprezentace je reálná reprezentace založena na abecedách vyšší kardinality. V reálné reprezentaci jsou jedinci zastupováni vektorem reálných nebo přirozených čísel. Tato čísla jsou seřazena podle seznamu, u rozvrhovací úlohy například podle seznamu pořadí činností.

Jak již bylo zmíněno výše, reálná reprezentace je vhodná zejména u prohledávání spojitého prostoru vysoké dimenze a při numerických optimalizacích s vysokou požadovanou přesností. U reálné reprezentace není potřeba dekódovat chromozomy dříve, než se ohodnotí, takže dojde k úspoře výpočetního času. Výhodou této reprezentace je navíc také *gradualita*, což znamená, že malá změna genotypu působí malou změnu fenotypu. Toto u binární reprezentace nemusí vždy platit, viz Hamming cliff [2].

Genetické algoritmy s reálnou reprezentací dosahují lepších výsledků, jak v rychlosti výpočtu, tak v jeho přesnosti [19]. Navíc je možné jejich výkonost dále zlepšovat zapojením genetických operátorů, které využívají znalostí o řešené úloze. Konstrukce těchto operátorů je v prostředí reálné reprezentace výrazně snazší, než v prostředí binární reprezentace [2].

Reálné reprezentace je využíváno i v algoritmu, pomocí kterého je řešena praktická část této práce. V tomto algoritmu je reálně reprezentováno jak pořadí činností, tak rozvrh. Na příklad této reprezentace se tedy můžete podívat na přiloženém CD v Příloha 3 input_NSCZ_21×14. Příklad binární reprezentace, který udává pořadí činností převzatý z práce [2], je na obrázku 3-1.

Obrázek 3-1: Binární reprezentace, pořadí činností

$$\begin{array}{l} \\ \text{činnost 1} \lhd \text{činnost 2} \\ \text{činnost 1} \lhd \text{činnost 3} \\ \text{činnost 2} \lhd \text{činnost 3} \end{array} \begin{array}{c} \begin{array}{cccc} S1 & S2 & S3 & S4 \end{array} \\ \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

Zdroj: práce [2]

## Inicializace

Inicializací se rozumí vytvoření počáteční populace. Počáteční populace se standardně vytváří náhodným generováním chromozomů jedinců. V některých případech může být ale náhodná inicializace neefektivní, například u GA s malou populací se může stát, že tento způsob nepokryje celý prohledávaný prostor. V těchto případech se inicializace vytváří systematicky. Inicializaci je možné provádět také z dobrých řešení, která byla předem získána pomocí heuristik. GA s takto vytvořenou inicializací najde mnohdy lepší řešení rychleji, ale také s větší pravděpodobností předčasné konvergence [15].

## Velikost populace

Je důležité zvolit správnou velikost populace. Tato volba bude mít totiž zásadní vliv na GA. Pokud bude zvolená populace příliš malá, s velkou pravděpodobností nezvládne pokrýt celý prostor, což může způsobit předčasnou konvergenci. V opačném případě, pokud zvolíme populaci příliš velkou, GA bude mít velkou výpočetní náročnost a to zejména proto, že bude muset ohodnocovat spoustu jedinců. Výpočet pak bude trvat neadekvátně dlouho. Někdy je uváděno, že optimální velikost populace se zvětšuje s délkou chromozomu exponenciálně, v takovém případě by ale byl GA nepoužitelný pro praktické příklady. Za pomocí experimentů bylo ale ukázáno, že takto velká populace není zapotřebí [15]. Pro praktické využití je mnohem důležitější otázka minimální velikosti populace, při které GA ještě vykazuje dobré výsledky. Otázka minimální populace je řešena například v práci [20].

## Terminace

Terminace, neboli ukončení Genetického algoritmu. Jak již bylo zmíněno dříve metaheuristické metody, na rozdíl od deterministických nekončí konvergencí do lokálního optima, musí tedy být ukončeny, jinak by běžely stále do nekonečna. K ukončení algoritmu se používají 3 druhy kritérií. Nejčastěji se algoritmus omezí počtem iterací nebo maximální dobou běhu. Lze se ale také řídit stavem populace a ukončit algoritmus při poklesu diverzity populace pod určitou mez, kterou je třeba předem specifikovat, nebo pokud v posledních x generacích nedojde ke zlepšení dosavadního nejlepšího řešení.

## Selekce

Pomocí selekce jsou vybírány vhodní jedinci pro následné křížení. Selekce se provádí pomocí různých metod.

První metoda, která zde bude uvedena, je takzvaný *ruletový výběr*. Tato metoda je nejčastěji používanou metodou selekce obecně. Metoda funguje tak, že sečte ohodnocení *fitness* u každého jedince z populace a podle velikosti jednotlivých ohodnocení přiřadí jedincům odpovídající velikost „*ruletového kola*“, pak je vykonán náhodný „*hod*“ a jedinec, který je vybrán, postoupí do následující generace. Pravděpodobnost volby *i*-tého jedince je následující,

$$p(i) = \frac{f(i)}{\sum_{j=1}^{N} f(j)},$$

kde $i \in \{1, \ldots, \mathrm{N}\}$, N představuje počet jedinců v populaci $f(i)$ je velikost ohodnocení *i*-tého jedince v populaci a $\sum_{j=1}^{N} f(j)$ je suma ohodnocení všech jedinců v populaci[21].

Další metoda selekce je *turnajový výběr*. Tato metoda je nejpoužívanější metodou selekce u aplikací GA. Turnajový výběr je oblíbený hlavně z důvodu jednoduchosti implementace a zároveň zachování kvality selekčního tlaku. Metoda funguje tak, že vždy vybere jedince do „turnaje“, tyto jedinci pak „soupeří“, dále postupuje pouze jedinec s největší hodnotou *fitness*. Tímto je dodržen základní předpoklad, že „přežívají silnější jedinci“, tedy jedinci s vyšší kvalitou, vzhledem k tomu že jedinci do turnaje jsou vybíráni náhodně, je zaručena i diverzita populace. Jedinci mohou být do turnaje vybrány několikrát (pokud mají jiného soupeře) [21]. Vše je dobře patrné z obrázku 3-2, který jsem převzal z práce [21].

Tabulka 3-1: Turnajový výběr

| Jedinec číslo | Ohodnocení jedince | Jedinci vybraní do souboje | Výherce souboje | Poznámky |
|---|---|---|---|---|
| 1 | 5 | (1, 3) | 3 | |
| 2 | 11 | (1, 4) | 4 | |
| 3 | 14 | (6, 9) | 6 | |
| 4 | 4 | (5, 9) | 5 | |
| 5 | 2 | (1, 2) | 2 | |
| 6 | 16 | (2, 3) | 3 | |
| 7 | 8 | (7, 8) | 7 | |
| 8 | 8 | (8, 7) | 8 | náhodný výběr |
| 9 | 1 | (6, 5) | 6 | |

Zdroj: práce [21]

Poslední metoda, která bude v mé práci uvedena, je metoda *pořadové selekce*. Tato metoda seřadí jedince podle hodnoty fitness od nejlepšího po nejhoršího. Pak probíhá náhodný

výběr jedinců s tím, že jedinci s lepší hodnotou účelové funkce jsou vybíráni s větší pravděpodobností. Další metody selekce a jejich podrobnější popis je možné najít v práci [21].

## Křížení

Jeden z hlavních genetických operátorů je právě křížení, s jehož pomocí získáváme nové jedince, kteří nebyli vytvořeni při inicializaci, či během předchozích generací křížení. Křížení skládá nové kandidáty na řešení z částí stávajících řešení a tím zvětšuje prohledávaný prostor. Tímto genetickým operátorem je také snižováno riziko uváznutí v lokálním extrému. Do křížení se dostávají jedinci na základě selekce, viz kapitola 3.3.5.

Budou zde uvedeny dva základní typy binárního křížení a jeden typ, který se využívá na úlohy, kde je chromozom reprezentován permutací prvků, jako je tomu i v algoritmu použitém v praktické části této práce.

Prvním typem binárního křížení je *jednobodové křížení*. Tento genetický operátor binárního křížení je nejrozšířenější. Jednobodové křížení pracuje tak, že náhodnou generací vybere číslo mezi 0 až K-1, kde K vyjadřuje délku chromozomu, proto je důležité, aby vygenerované číslo *k*, bylo maximálně hodnoty K-1, jinak by probíhalo křížení od poslední pozice a nic by se nezměnilo [21]. Příklad pro $k = 4$, můžete vidět nížena obrázku 3-3, tento příklad je převzat z práce [21].

Obrázek 3-2: Jednobodové křížení pro $k = 4$

| Původní chromozómy (rodičovské) | | Nové chromozómy (potomci) |
|---|---|---|
| **(1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0)** | → | **(1, 1, 0, 0,** 1, 0, 1, 1, 0, 1, 0, 1) |
| (0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1) | → | (0, 1, 0, 0, **1, 1, 1, 1, 1, 0, 0)** |

Zdroj: práce [21]

Druhým typem binárního křížení, který zde bude uveden, je *dvoubodové křížení*. Tento typ křížení pracuje stejně jako jednobodové křížení s rozdílem, že vygenerujeme dvě čísla z rozmezí 0 až K-1 a křížení pak vypadá jako na obrázku 3-4.

Obrázek 3-3: Dvoubodové křížení pro $k_1 = 3$, $k_2 = 8$

| Původní chromozómy (rodičovské) | | Nové chromozómy (potomci) |
|---|---|---|
| **(1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0)** | → | **(1, 1, 0,** 0, 1, 0, 1, 1, **1, 1, 0, 0)** |
| (0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1) | → | (0, 1, 0, **0, 1, 1, 1, 1,** 0, 1, 0, 1) |

Zdroj: práce [21]

Nyní zde bude uvedeno *nelineární křížení*. Tento typ křížení je použit i v algoritmu z praktické části této úlohy a je vhodný pouze pro úlohy, jejichž jedinci jsou tvořeny permutací

prvků bez opakování. Nelineární křížení je navrženo tak, aby nalézalo souvislé části, které se liší pouze v pořadí prvků a jejich prohozením vytvářelo nové jedince.

Obrázek 3-4: Nelineární křížení




Zdroj: práce [2]

Další metody křížení, jak pro binární kódování, tak pro permutační kódování jedinců, jsou uvedeny v práci [21].

## Mutace

Genetický operátor mutace má vnášet do populace diverzitu a bránit tak uvíznutí v lokálním optimu. Mutace náhodně vybere jedince a provede změnu jednoho náhodně vybraného genu. Pravděpodobnosti pro tento náhodný výběr mohou být buď konstantní, jako je tomu například v algoritmu používaném v praktické části této práce, kde je pravděpodobnost mutace pevně nastavena na hodnotu 0.001 a nebo může být proměnná, podle současného stavu populace. Například, pokud se již několik generací nepodařilo vylepšit hodnotu nejlepšího řešení, může být zvýšena pravděpodobnost mutace, což může pomoci k vymanění se z lokálního optima a nalezení lepšího řešení. Lze také zavést pravděpodobnost mutace jako klesající funkci času, takto funguje například i simulované žíhání, viz kapitola 2.3.1

V binární reprezentaci funguje mutace jednoduše, když je vybrán s určitou pravděpodobností náhodný gen a ten je změněn z „0“ na „1“, nebo naopak. Pokud je ale jedinec vyjádřen jako permutace prvků bez opakování, tak by nahrazení genu jiným náhodným genem, například u rozvrhovací úlohy, kde by chromozom reprezentoval pořadí činností, vytvářel pouze nepřípustná řešení. Toto se dá řešit použitím operátoru mutace, který náhodně vybere dva geny chromozomu, ty mezi sebou prohodí, vytvořený jedinec může zastupovat přípustné řešení, ale je důležité si dát pozor, aby neporušoval precedenční podmínky nebo zdrojová omezení. Některé operátory mutace, které jsou vhodné pro úlohy s reálnou reprezentací, je možné najít v práci [21].

## Nová populace

V Hollandově návrhu GA, se po vykonání genetických operátorů, vytvoří nová populace, která kompletně nahradí stávající populaci. Tím se ale přijde o všechna dobrá řešení, která byla doposud nalezena. Aby se tomuto zabránilo, používá se takzvaná strategie elitismu (*elitism*). Tato strategie pracuje tak, že nejlepšího jedince převede nezměněného do další generace a zbytek populace nahradí. Další strategie používané k zachování nejlepších jedinců jsou například *population overlap* a *steady-state*. První ze jmenovaných je zobecnění strategie

elitismu, které přetahuje do další generace více nejlepších jedinců. Strategie steady-state nahrazuje pouze nejhoršího jedince z generace.

### Diverzita populace

Aby GA generoval co nejlepší výsledky, je důležité udržet diverzitu populace, tím se zabrání předčasné konvergenci do nevhodného lokálního optima. Diverzita populace může být ovlivňována velikostí populace, nastavení vyšší pravděpodobnosti mutace, či použitím strategií, které zajistí udržení diverzity populace. Toho může být dosaženo například tím, že bude zakázáno připuštění duplicitních chromozomů do nové populace, nebo zavedením cíleného výběrů jedinců pro operátor křížení. Kritériem pro výběr může být například podobnost, nebo odlišnost chromozomů.

### Implementace omezujících podmínek

Opravný algoritmus je jedním z přístupů k implementaci omezujících podmínek. Opravný algoritmus pracuje tak, že najde řešení, která jsou kvůli omezujícím podmínkám nepřístupná a „opraví je“ tak, aby byla přístupná a zároveň co nejpodobnější původním nepřípustným řešením. Aby opravný algoritmus fungoval dobře, musí být navržen pro konkrétní typ úlohy. Opravný algoritmus je využit i v algoritmu používaném v praktické části této práce.

Dalším možným přístupem je vytvořit takovou reprezentaci, ve které by jedinci nemohli reprezentovat nepřípustná řešení a ve které by byly genetické operátory navrženy tak, že by vytvářely pouze přípustná řešení. Tento přístup je zase možné použít pouze pro konkrétní typ úlohy.

Je možné také použít přístup, který se chová jako by omezující podmínky neexistovaly. Tento přístup kromě běžného ohodnocení jedinců účelovou funkcí, penalizuje nepřípustná řešení pomocí penalizační funkce. Penalizovaní jedinci potom s vysokou pravděpodobností neprojdou selekcí. U tohoto přístupu je také důležité rozhodnout, jak by měla penalizační funkce vypadat a zda vytvořit penalizační funkci tak, aby penalizovala všechny nepřípustná řešení stejně, nebo podle toho, jak moc porušili omezující podmínky. Tento přístup je nevhodný na úlohy, ve kterých je malý poměr počtu přípustných řešení vzhledem k počtu nepřípustných [2]. Tímto přístupem se zabývá například práce [19].

# 4 Programovací prostředí MATLAB[4]

MATLAB je zkrácení výrazu matrix laboratory, tedy česky maticová laboratoř, což odráží skutečnost, že matice jsou klíčovou datovou strukturou, pro výpočty v MATLABu. MATLAB je výkonné interaktivní programovací prostředí, určené k vědeckým a inženýrským výpočtům. Výhodou MATLABu je bezpochyby to, že na rozdíl od jiných programovacích prostředí, je zadávání problémů a řešení v MATLABu stejné jako v matematice, bez klasického programování. Základním datovým prvkem tohoto programovacího prostředí je matice, které se nezadává rozměr. Díky tomu dokáže MATLAB řešit mnoho numerických problémů výrazně rychleji, než při použití běžných programovacích jazyků, jako jsou například Basic, nebo C. Další z velkých výhod MATLABu je bezpochyby jeho snadná rozšiřitelnost, díky které můžete doplňovat systém o vlastní funkce (M-soubory), nebo i celé aplikace. Navíc existuje spousta konkrétně zaměřených rozšíření MATLABu, těmto rozšíření se říká toolbox. Toolboxy jsou skupiny M-souborů se zaměřením na konkrétní třídy problémů.

V současné době existují rozšíření pro tyto třídy problémů:

- zpracování signálů,
- zpracování obrazů,
- teorie řízení,
- identifikace systému,
- optimalizace,
- neuronové sítě,
- spliny,
- statistika,
- symbolická matematika.

**Příkazové okno**

K zadávání příkazů slouží takzvané příkazové okno. Pokud se v tomto okně zobrazí (>>), je možné zadat příkaz.

Menu příkazového okna

- Skupina *File* obsahuje příkazy pro vytváření, otevírání a spouštění M-souborů, dále také příkazy pro ukládání práce, tisk, a ukončení MATLABu.
- Skupina *Edit* obsahuje příkazy pro kopírování zvolené oblasti z příkazového okna do schránky MATLABu a příkazy pro zobrazení obsahu schránky do příkazového okna.
- Složka *Options* obsahuje příkazy na nastavení barev, fontů a volby editoru M-souborů.
- Složka *Windows* zobrazuje seznam všech oken MATLABu, skrz tuto složku se mezi nimi můžete přepínat.
- Složka *Help* - zde jsou zahrnuty nápovědy pro MATLAB.

[4] Není-li uvedeno jinak, kapitola vychází ze zdroje [21].

Při praktickém využití většinou nezadáváme úlohy přímo do příkazového okna, ale tvoříme takzvané *skripty*. Do těchto skriptů se dá přehledně zapsat kód algoritmu, který může být opatřen poznámkami nebo pro přehlednost třeba rozdělen do více skriptů. Skripty navíc kód ukládají, takže nebude ztracen při spuštění jiného algoritmu.

Dalším typem souboru jsou takzvané funkční soubory. Tyto soubory musí obsahovat na začátku v prvním řádku slovo *function* a název, kterým má být tento soubor pojmenován. Do funkčních souborů je pak možné uložit jakýkoli algoritmus, který lze následně používat jako klasickou MATLABovskou funkci. Algoritmus musí být zadán obecně, aby bylo možné jeho využití pro různé příklady.

Jedinou funkcí MATLABu, která zde bude uvedena je funkce *help*. Tato funkce zajišťuje nápovědu ke všem MATLABovským funkcím, stačí zadat help do příkazového řádku spolu s názvem funkce, ke které je požadována nápověda a funkce help vrátí požadované informace. Více o programovacím prostředí MATLAB je například v práci [21].

# 5 Praktická část

Tato kapitola je věnována aplikaci genetického algoritmu na konkrétní případ rozvrhování výroby strojírenské společnosti NSCZ s.r.o. Součástí této kapitoly je představení společnosti a konkrétních požadavků, které je třeba při rozvrhování výroby dodržet. Dále zde jsou představeny jednotlivé činnosti a stroje, které jsou k výrobě používány. Následně je zde popsána samotná aplikace genetického algoritmu na tuto konkrétní úlohu rozvrhování výroby.

## 5.1 Představení společnosti

Tato bakalářská práce je vypracována na datech společnosti NSCZ s.r.o. Tato společnost patří společnosti Nakagawa Sangyo Co.,Ltd., což je japonská strojírenská společnost zaměřená na automobilový průmysl. Tato společnost má kromě České republiky další závody v Japonsku, Číně, Anglii, Indonésii, Thajsku a USA. Zkratka NSCZ znamená Nakagawa Sangyo Czech Republic a stejně jako její mateřská společnost vyrábí komponenty pro automobilový průmysl. NSCZ se specializuje na výrobu výfukových komponentů spalovacích motorů. Z těchto komponentů pak ve společnosti Futaba Czech, s.r.o. kompletují výfukové systémy pro automobily Toyota, Suzuki, Peugeot, Citroen a Opel. Společnost NSCZ byla založena 23. února 2004 a v současné době zaměstnává 54 lidí. Společnost jsem si vybral z důvodu dostupnosti dat a ochoty systémové spolupráce zaštítěné funkcí statutárního orgánu, kterým je můj otec. Navíc se zde nabízí možnost práci dál rozvíjet a skutečně aplikovat na rozvrhování činností společnosti NSCZ.

## 5.2 Zadání problému

Cílem této bakalářské práce je rozvrhnout výrobu společnosti NSZC na jednotlivých strojích tak, aby byly splněny preferenční podmínky a výroba se stihla v požadovaném čase. Společnost dostává objednávky třikrát týdně. Na výrobu a kompletaci dané objednávky jsou vždy přibližně dva dny. Ve výrobě se pracuje v nepřetržitém provozu na tři směny. Celý výrobní proces probíhá na 14 různých typech strojů, na kterých se vyrábí 21 různých typů výrobků. Společnost drží skladem výrobky v různých fázích jejich výrobního procesu, na přibližně 14 dní dopředu. Pro zjednodušení nejsou v této práci brány v potaz skladové zásoby ani obsluha strojů. Předpokládá se tedy, že každý stroj má v každém časovém okamžiku svou obsluhu a nejsou žádné skladové zásoby výrobků, ani jejich komponentů.

## Stroje

V tabulce 5-1 jsou uvedeny všechny druhy strojů používané společností NSCZ na výrobu komponentů pro výfukové potrubí. Je zde také popsáno množství strojů každého druhu, které má společnost k dispozici. V posledním sloupci jsou pak vypsané aktivity, které jednotlivé stroje vykonávají při výrobě. Názvy strojů a aktivit jsou převzaty z datových podkladů, které mi poskytla společnost NSCZ.

Tabulka 5-1: Stroje

| Název stroje | Počet kusů k dispozici | Aktivity vykonávané na stroji |
|---|---|---|
| rychloběžný perforační lis | 1 | cutting-punching |
| klikový lis 100t | 2 | punching, U / O bend, blankyng It., resize, ohnutí, zúžení, roztažení, |
| svářečka TIG | 4 | TIG weld, resize |
| lis 160t | 1 | cutting, cutting-punching, |
| lis 250 t | 1 | U / O bend, cutting, cutting-punching, obstřih |
| hydraulický lis 25t | 2 | resize |
| klikový lis 100t s otvorem ve stole | 1 | necking |
| svářečka MIG | 2 | MIG weld |
| ruční práce | | kompletace |
| hydraulický lis 100t | 2 | press, resize |
| klikový lis 25t | 1 | cutting, tab-cut |
| 1-bodová svářečka | 2 | spot weld |
| 8-bodová el. Svářečka | 1 | spot weld jazyk |
| navíječka | 1 | bulky |

Zdroj: Data NSCZ, vlastní zpracování

## Výrobky

Tabulky s popisy všech 21 výrobků jsou kvůli svému rozsahu uvedeny v příloze tabulkách 6-1 až 6-5. V tabulce 5-2 níže je popsán pouze jeden výrobek. Tato tabulka slouží jako ilustrační příklad. V prvním sloupci je vždy uveden kód výrobku, v druhém sloupci jsou aktivity, které musí být k jeho výrobě vykonány. Vedle každé aktivity je vždy uveden stroj, na kterém je aktivita vykonávána a čas, jaký vykonání aktivity zabere v sekundách. V posledním sloupci je pak uvedeno pořadí, v jakém musí být jednotlivé aktivity vykonávány. Pokud jsou ve sloupci Stroje nějaké stroje dohromady ohraničeny tučně, znamená to, že po sobě jdoucí operace jsou vykonávány na stejném stroji a budou při výpočtu považovány za jednu operaci. Toto zjednodušení je prováděno, aby bylo možné zadat úlohu job-shop pomocí matice $\mathbf{X}$ typu $n \times m$, kde $i$-tý řádek udává, v jakém pořadí prochází úloha i jednotlivými stroji a každý stroj je zde vždy právě jednou. Toto zjednodušení nijak nenaruší přípustnost této úlohy a vzhledem

k tomu, že je potřeba řešení, které bude kratší než zadaná horní mez, ne nutně optimální, tak může být provedeno, aniž by zásadně ovlivnilo výsledek.

Tabulka 5-2: Výrobky-ilustrační příklad

| Kód výrobku | Operace | Stroje | Čas operace v sekundách | Pořadí operací |
|---|---|---|---|---|
| 0Q010-1500 | cutting-punching | rychloběžný perforační lis YAMADA | 8 | 1 |
| | punching | klikový lis 100t | 17 | 2 |
| | U / O bend | klikový lis 100t | | |
| | TIG weld,resize | svářečka TIG | 20 | 3 |
| | cutting | lis 160t | 3 | 4 |
| | U / O bend | lis 250t | 7 | 5 |
| | TIG weld,resize | svářečka TIG | 16 | 6 |
| | resize | hydraulický lis 25t | 7 | 7 |
| | kompletace | ruční práce | 9 | 8 |
| | necking | klikový lis 100t s otvorem ve stole | 19 | 9 |
| | MIG weld | svářečka MIG | 27 | 10 |

Zdroj: Data NSCZ, vlastní zpracování

## 5.3 Řešení problému

Vzhledem k tomu, že rozvrhovací úloha není náplní mého bakalářského studia a nikdy předtím jsem se s tímto tématem nesetkal, musel jsem první nastudovat, co to vlastně rozvrhovací úloha je. Poté, co jsem se s problematikou rozvrhování seznámil a byl jsem schopen rozpoznat, že úloha NSCZ je úloha typu job-shop, jsem začal hledat metodu vhodnou pro řešení tohoto problému.

### Nevydařené pokusy

Mým hlavním kritériem, při výběru metody byl čas. Vzhledem k tomu, že se ve společnosti NSCZ rozvrhuje výroba vždy na dva dny, musel jsem najít takovou metodu, která úlohu vyřeší v řádu maximálně desítek minut. To se ukázalo při velikosti úlohy NSCZ jako problém. První jsem se pokoušel úlohu řešit za pomoci *TORSCHE scheduling toolbox* [22]. *TORSCHE* je rozšíření programového prostředí MATLAB pro rozvrhovací úlohy. K řešení jsem zde chtěl použít algoritmus *cpshopscheduler*, o kterém jsem se dočetl v bakalářské práci Ing. Jiřího Ciglera, Ph.D. „*Programování s omezujícími podmínkami v Scheduling Toolboxu*“ [23]. Bohužel jsem po konzultaci s panem Ing. Ciglerem zjistil, že tento algoritmus byl podporován pouze v beta verzi tohoto toolboxu, která nebyla nikdy vydána. Po prvním neúspěchu jsem se vrátil k literatuře a hledal metodu řešení úlohy typu job-shop, která by ji

řešila rychle a pracovala dobře i pro velké úlohy. Nakonec jsem se rozhodl pro metodu genetického algoritmu, právě proto, že pracuje rychle i pro velké úlohy. Podařilo se mi získat zdrojový kód genetického algoritmu napsaný v programovém prostředí MATLAB. Autorem tohoto kódu byl Maxim Vedenev. Tento algoritmus ale bohužel nebyl schopen řešit úlohu NSCZ, kvůli její velikosti. Algoritmus pracoval dobře pouze pro úlohy o maximální velikosti 8×8. Kód algoritmu přikládám v příloze na CD s názvem Příloha 1 GA_Vedenev.

## Výsledné řešení

Po neúspěchu s algoritmem pana Vedeneva, se mi podařilo získat jiný zdrojový kód Genetického algoritmu. Tento algoritmus již pracuje dobře a dostatečně rychle i pro velké úlohy. Algoritmus jsem převzal z bakalářské práce *„Genetický algoritmus jako metoda řešení rozvrhovací úlohy“* od Martina Hanzala [2]. Tento algoritmus využívá reálnou reprezentaci pořadí činností, reálnou reprezentaci rozvrhu a metody turnajové selekce.

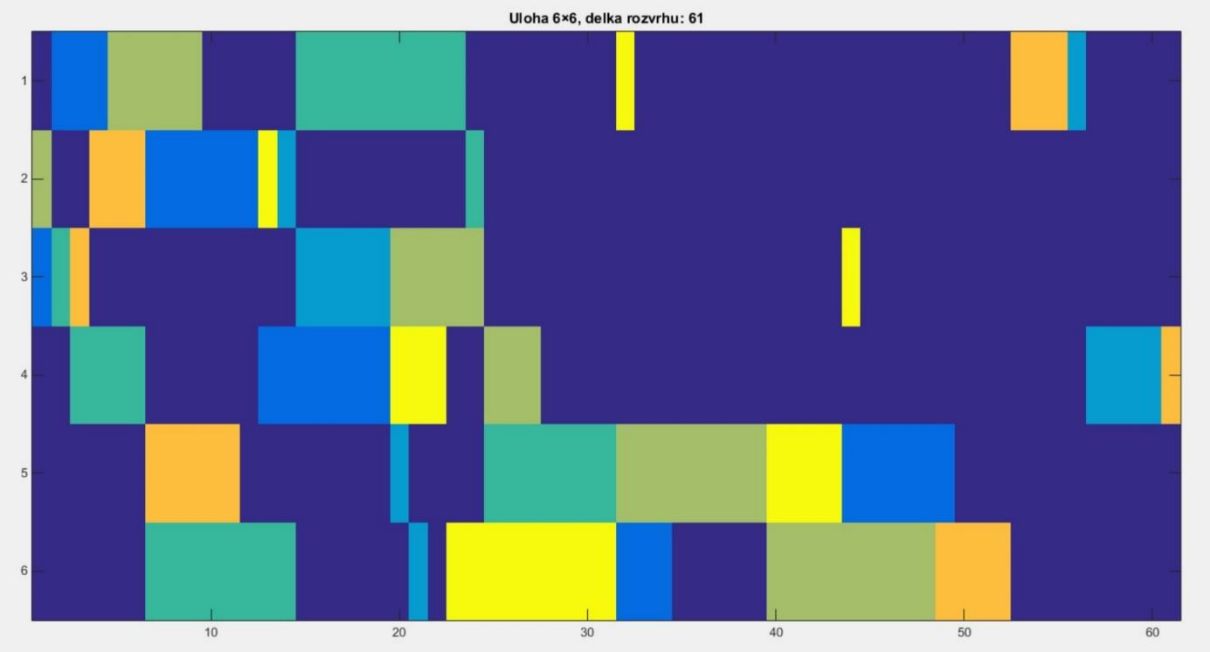
Úloha NSCZ je definována jako úloha typu job-shop, která je obvykle zadána pomocí matice **X** typu $n \times m$, ve které $i$-tý řádek udává pořadí, v jakém úloha i prochází jednotlivými stroji a matice **P** typu $n \times m$, kde prvek $p_{ij}$ vyjadřuje čas, po který je úloha i vykonávána na stroji j. Pokud stroj nezpracovává danou úlohu, $p_{ij} = 0$. Z důvodu správné funkce algoritmu jsou nulové hodnoty v matici **P** nahrazeny jedničkami. Doby trvání jednotlivých aktivit jsou zadány v sekundách. Pan Hanzal matice **X** a **P** ve svém algoritmu spojil do jediné matice **data**, pomocí které se do algoritmu zadají potřebné informace. Matice **dat** je typu $n \times m$, její řádky představují jednotlivé činnosti, liché sloupce představují pořadí na strojích (číslované od 0) a sudé sloupce obsahují časy strávené činnostmi na strojích.

Ve společnosti NSCZ se vyrábí 21 finálních výrobků, z toho se jich 7 skládá ze tří komponentů, 1 ze dvou a zbytek je tvořen pouze jedním komponentem. V prvním pokusu jsem problém definoval jako problém rozvrhnutí 36 úloh na 14 strojích. Za úlohy $i = 1, 2, \ldots, n$, jsem bral jednotlivé komponenty výrobků. Ukázalo se, že takto definovaný problém nepřináší dobré výsledky. Úloha je díky své velikosti a díky příliš velkému počtu nul v matici **P** velice nepřehledná. Výpočet je neúnosně dlouhý a navíc bych musel řešit problém jak zachovat preference mezi jednotlivými úlohami. Vstupní matice **data** o rozměru 36×14 jsou k vidění na přiloženém CD, pod názvem Příloha 2 input_NSCZ_36×14. Nakonec jsem se rozhodl úlohu raději předefinovat. Definoval jsem ji jako problém rozvrhnutí 21 úloh na 14 strojů, její finální podoba je také k vidění na přiloženém CD, pod názvem Příloha 3 input_NSCZ_21×14. Za úlohy $i = 1, 2, \ldots, n$, beru pouze finální výrobky. Řešení problému se značně zrychlilo, zpřehlednilo a zbavil jsem se také problému preferencí mezi úlohami. Dále tedy pracuji s touto formulací úlohy NSCZ.

Abych zlepšil praktickou využitelnost algoritmu ve společnosti NSCZ, provedl jsem v něm dvě změny. První změnu jsem udělal ve vykreslení řešení do Ganttova diagramu. Pan Hanzal využívá k vykreslení grafu funkci *imagesc*. Tato funkce vykreslí Ganttův diagram a jednotlivé úlohy obarví různými barvami. Osa X zde znázorňuje čas a osa Y znázorňuje jednotlivé stroje. Tento graf je sice dobrý pro vytvoření představy, jak bude výroba rozvržena,

nicméně z něj nevyčtete, jak jdou konkrétní úlohy za sebou na kterém stroji. Graf ilustračního příkladu 6×6 můžete vidět na obrázku 5-1.

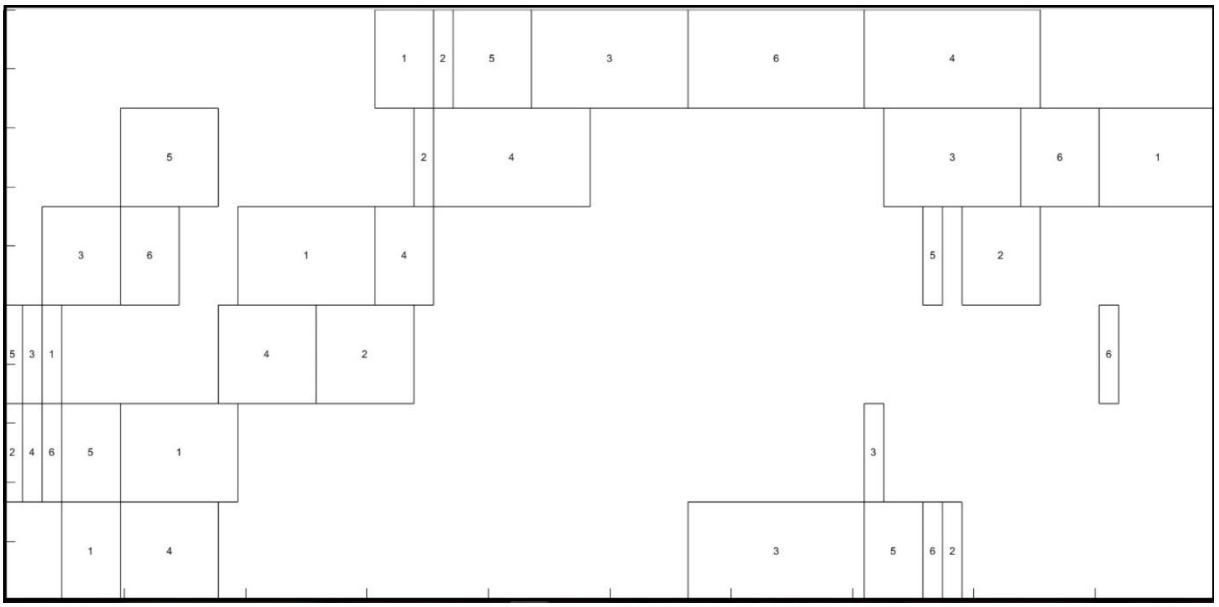
Obrázek 5-1: Původní Ganttův diagram




Zdroj: Vlastní zpracování

Jelikož mým cílem je použitelnost tohoto algoritmu na rozvrhování výroby NSCZ, musel jsem graf upravit tak, aby z něho bylo možné skutečně rozvrhnout výrobu. Proto jsem s pomocí pana Ing. Jana Zouhara, Ph.D. napsal kód, který rozdělí graf na obdélníky, do jejichž středu vypíše číslo úlohy. K vycentrování grafu jsem musel posunout osy na samotný kraj zobrazované plochy, takže algoritmus vykresluje pouze prostor mezi osami a přišel jsem o stupnice na osách. Na rozvrhnutí výroby mi ale stačí vědět celkový čas rozvrhu a jaké výrobky mám v jakém pořadí vyrábět na jakém stroji. Toto mi algoritmus plně poskytuje, čas rozvrhu vypíše a zbytek vyčtu z Ganttova diagramu, který jsem upravil tak, aby na něm bylo jasně patrné pořadí činností na jednotlivých strojích. Tento graf si můžete k ilustračnímu příkladu 6×6 prohlédnout na obrázku 5-2.

Obrázek 5-2: Upravený Gantův diagram



Zdroj: Vlastní zpracování

**Zdrojový kód pro vykreslení upraveného Ganttova diagramu:**

```
finish  = 0;
start   = 0;
task    = 0;
machine = 0;

konce = round(Dn(1,2:end-1));
konce = reshape(konce,m,n)'; %konce cinnosti na strojich
P_rnd = round(P);
rozvrh = zeros(m,Dn(1,end));
for i = 1:m  % i je stroj
    for j = 1:n  % j je cinnost
        cinnost = best_seq(i,j);
        konec = konce(cinnost,i);
        delka = P_rnd(cinnost,i);
        if delka > 0
            rozvrh(i,konec - delka + 1:konec) = cinnost; %ROZVRH
        end
           task(end+1) = cinnost;
           finish(end+1) = konec;
           start(end+1) = konec - delka;
           machine(end+1) = i;
    end
end

  finish = finish(2:end);
  start = start(2:end);
  machine = machine(2:end);
  task = task(2:end);
```

```
figure % vykreslý prázdný graf
  axis([0 bestSF 0 m])% vytvoří osu y od 0 do m (počet strojů) a osu x od 0
do celkové délky rozvrhu
   mainh = axes('position',[0 0 1 1])% posune osy až na kraj obrazovky, aby
byl graf správně vycentrovaný
for obdelnik = 1:length(finish)
     dim = [(start(obdelnik)/bestSF) ((machine(obdelnik)-1)/m)
((finish(obdelnik) - start(obdelnik))/bestSF) (1/m)];% určí dimenzy grafu
     str = num2str(task(obdelnik));% vypíše číslo úlohy do obdelníku
     annotation('textbox', dim, 'String', str, 'HorizontalAlignment',
'center', 'VerticalAlignment', 'middle');
end
```

Aby algoritmus fungoval správně a mohl ho obsluhovat i člověk bez znalostí programovacího prostředí MATLAB, musel jsem provést ještě několik změn v zadávání dat. Jak jsem již zmínil, data se do algoritmu zadávají pomocí matice **data**. To ale znamená, že musíte definovat matici **data** znovu, vždy pokud změníte požadované množství výrobků, které chcete vyrobit. Aby ve společnosti NSCZ nemusely neustále přepisovat matici **data**, vytvořil jsem ji vynásobením prvků na stejných pozicích matic **B** a **M.** Kde matice **B** je totožná s původní maticí **data,** tedy obsahuje preferenční vztahy mezi jednotlivými aktivitami a časy strávené činnostmi na strojích při výrobě jednoho kusu každého výrobku. Matice **M** je matice, která udává pro jaké množství jednotlivých komponentů každého výrobku se má výroba rozvrhnout. Vynásobením prvků na stejných pozicích těchto matic tedy získáme matici data s preferenčními vztahy v lichých sloupcích a s dobami trvání jednotlivých činností pro požadované množství výrobků v sudých sloupcích. Matice **M** je tvořena proměnnými, aby bylo možné často měnit požadované množství s minimální pracností. Při každém spuštění algoritmu se funkce *input* (funkce v MATLABU, která se zeptá uživatele na to, jakým číslem má nahradit konkrétní proměnou) zeptá na hodnoty těchto proměnných a jediné co je potřeba udělat, je zadat počty výrobků, pro které chceme výrobu rozvrhnout. Není tedy problém i pro člověka bez znalostí programovacího prostředí MATLAB rozvrhnout výrobu pro jakékoliv množství výrobků.

Pro správnost výstupu jsem ještě musel vydělit sudé sloupce matice data, množstvím strojů jednotlivých druhů, které se na výrobě podílejí. Algoritmus v jeho konečné podobě přikládám na CD s názvem Příloha 4 GA_NSCZ.

# Závěr

Cílem mé bakalářské práce bylo naplánovat výrobu společnosti NSCZ. Abych tento cíl mohl splnit, musel jsem splnit také dílčí cíle. Nejprve jsem se musel seznámit s problematikou rozvrhovací úlohy a metodami jejího řešení, neboť jsem se s nimi během svého studia zatím nesetkal. Mým dalším dílčím cílem bylo vybrat vhodnou metodu řešení pro můj konkrétní příklad rozvrhování výroby a aplikovat ji na něj. Všechny tyto cíle se mi podařilo splnit.

V první části práce jsem podrobně popsal problematiku rozvrhovací úlohy a její speciální typy. Ve druhé části jsem pak rozdělil metody řešení rozvrhovací úlohy na tři skupiny, kterými jsou exaktní metody, heuristické metody a metaheuristické metody. Každou z těchto skupin jsem stručně popsal a uvedl příklady algoritmů, které do ní spadají. Celou další kapitolu jsem pak věnoval popisu metaheuristické metody genetického algoritmu. Genetický algoritmus jsem podrobně popsal, jelikož jsem si ho vybral jako metodu vhodnou k řešení příkladu rozvrhování výroby NSCZ. V následující kapitole jsem stručně popsal programovací prostředí MATLAB, ve kterém je prováděna výpočetní část rozvrhování výroby NSCZ. Poslední kapitolu jsem věnoval samotné aplikaci genetického algoritmu na konkrétní příklad rozvrhování výroby společnosti NSCZ.

Tím, že se mi podařilo aplikovat genetický algoritmus na tuto konkrétní úlohu rozvrhování výroby, jsem splnil i hlavní cíl této práce. Upravil jsem použitý algoritmus tak, aby byl vhodný pro úlohu NSCZ. Zpracoval jsem data, která mi poskytla společnost NSCZ, tak aby šla použít jako vstupní data do tohoto algoritmu. Následně jsem definoval úlohu NSCZ tak, aby neporušovala omezující podmínky, dosahovala dostatečné hodnoty účelové funkce a byla vypočitatelná v relativně krátkém čase. Jedna iterace výpočtu trvá zhruba 10 minut a většinou dosahuje dostatečné hodnoty účelové funkce. Tím jsem výrazně zjednodušil rozvrhování výroby ve společnosti NSCZ. Doposud byl tento proces totiž schopen naplánovat pouze jediný člověk, který měl dostatečné zkušenosti a schopnosti. I přesto rozvrhoval výrobu zhruba dvě hodiny. Touto prací jsem dosáhl ušetření času tohoto jedinečného člověka a co více, dosáhl jsem jeho částečné nahraditelnosti. Pokud tedy bude tento člověk nyní časově indisponován, je schopen pomocí algoritmu, který jsem v této práci upravil a aplikoval na úlohu NSCZ, rozvrhnout výrobu kdokoli.

Protože se mi podařilo splnit všechny cíle a vidím, že moje práce je v praxi skutečně využitelná, chtěl bych v budoucnu na tuto práci navázat diplomovou prací. Rád bych rozvrhování společnosti NSCZ naplánoval tak, aby počítalo i se všemi skladovými zásobami, tedy bralo v potaz jak zásoby výrobků v různém stádiu dokončení, tak zásoby materiálů, které jsou k dispozici. V ideálním případě bych chtěl, aby dokázalo rozvrhnout i obsluhu na jednotlivé stroje. Tím bych vytvořil automatický systém plánování výroby, kterému by stačilo zadat pouze požadované množství výrobků.

# Literatura

[1] WALL, M. B. (1996). A genetic algorithm for resource-constrained scheduling (Doctoral dissertation, Massachusetts Institute of Technology).

[2] HANZAL, M. (2014). Genetický algoritmus jako metoda řešení rozvrhovací úlohy (Bakalářská práce, Vysoká škola ekonomická v Praze).

[3] YAMADA, T., & NAKANO, R. (1997). Job shop scheduling. In Genetic algorithms in engineering systems (pp. 134-160). Cambridge University Press.

[4] GAREY, M. R., & JOHNSON, D. S. (1979). Computers and intractability. San Francisco: freeman.

[5] CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., & Stein, C. (2001). Introduction to algorithms (Vol. 2, Section 3.1: Asymptotic notation, pp. 41–50.). Cambridge: MIT press.

[7] FIALA, P. (2012). Řízení projektů. Nakladatelství Oeconomica

[8] STRATTON, R. (2009). Critical chain project management theory and practice. Journal of Project Management and Systems Engineering, 149-173.

[9] BRUCKER, P., JURISCH, B., & SIEVERS, B. (1994). A branch and bound algorithm for the job-shop scheduling problem. Discrete applied mathematics, 49(1), 107-127.

[10] PANWALKAR, S. S., & ISKANDER, W. (1977). A survey of scheduling rules. Operations research, 25(1), 45-61.

[11] DAVIS, E. W., & PATTERSON, J. H. (1975). A comparison of heuristic and optimum solutions in resource-constrained project scheduling. Management science, 21(8), 944-955.

[13] LUKE, S. (2009). Essentials of metaheuristics (Vol. 3). Raleigh: Lulu.

[14] ČERNÝ, V. (1985). Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. Journal of optimization theory and applications, 45(1), 41-51.

[15] REEVES, C. R. (2010). Genetic algorithms. In Handbook of Metaheuristics (pp. 109- 139). US: Springer.

[16] GOLDBERG, D.E. (1989). Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, Reading, Massachusetts.

[17] WOLPERT, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation, 1(1), 67-82.

[18] HERRERA, F., LOZANO, M., & VERDEGAY, J. L. (1998). Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis. Artificial intelligence review, 12(4), 265-319.

[19] MICHALEWICZ, Z. (1992). Genetic algorithms + Data structures = Evolution program. Artificial Intelligence, Berlin: Springer

[20] REEVES, C. R. (1993). Using Genetic Algorithms with Small Populations. In Proceedings of the 5th International Conference on Genetic Algorithms (pp. 92- 99). Morgan Kaufmann Publishers Inc.

[21] STUDNIČKA, V. (2010). GENETICKÉ ALGORITMY – MULTI-CORE CPU IMPLEMENTACE (Diplomová práce, Vysoké učení technické v Brně).

[23] CIGLER, J. (2007). Programování s omezujícími podmínkami v Scheduling Toolboxu (Bakalářská práce, České vysoké učení technické v Praze)

# Internetové zdroje

[6] Turing machine. (2016, May 11). In Wikipedia, The Free Encyclopedia. Retrieved 9:59, April 11, 2016, from https://en.wikipedia.org/wiki/Turing_machine

[12] Gradient descent. (2016, April 30). In Wikipedia, The Free Encyclopedia. Retrieved 16:15, April 11, 2016, from https://en.wikipedia.org/wiki/Gradient_descent

[22] Retrived 16:00, April 12, 2016 from https://www.fel.cvut.cz/cz/vv/tymy/torsche.html

**Přílohy**

Tabulka 6-1: Výrobky 1. část

| Kód výrobku | Aktivity | Stroje | Čas operace v sekundách | Pořadí operací |
|---|---|---|---|---|
| 0Q010-1500 | cutting-punching | rychloběžný perforační lis YAMADA | 8 | 1 |
| | punching | klikový lis 100t | 17 | 2 |
| | U / O bend | klikový lis 100t | | |
| | TIG weld,resize | svářečka TIG | 20 | 3 |
| | cutting | lis 160t | 3 | 4 |
| | U / O bend | lis 250t | 7 | 5 |
| | TIG weld,resize | svářečka TIG | 16 | 6 |
| | resize | hydraulický lis 25t | 7 | 7 |
| | kompletace | ruční práce | 9 | 8 |
| | necking | klikový lis 100t s otvorem ve stole | 19 | 9 |
| | MIG weld | svářečka MIG | 27 | 10 |
| 0Q010-1510 | cutting-punching | rychloběžný perforační lis YAMADA | 8 | 1 |
| | blankyng It. | klikový lis 100t | 14 | 2 |
| | U / O bend | klikový lis 100t | | |
| | TIG weld | lis 250t | 3 | 3 |
| | cutting | svářečka TIG | 58 | 4 |
| | TIG weld 2 | svářečka TIG | | |
| | TIG weld 3 | svářečka TIG | | |
| | resize | hydraulický lis 25t | 7 | 5 |
| | kompletacea ce | ruční práce | 9 | 6 |
| | necking | klikový lis 100t s otvorem ve stole | 24 | 7 |
| | MIG weld | svářečka MIG | 27 | 8 |
| 0Y010-S107-1 | cutting | lis 160t | 2 | 1 |
| | U / O bend | klikový lis 100t | 9 | 2 |
| | TIG weld | svářečka TIG | 14 | 3 |
| | resize | hydraulický lis 25t | 7 | 4 |
| 0N080-S105 | cutting-punching | lis 250t | 5 | 1 |
| | press | hydraulický lis 100t | 12 | 2 |
| | U / O bend | klikový lis 100t | 10 | 3 |
| | TIG weld | svářečka TIG | 26 | 4 |
| | resize | hydraulický lis 25t | 12 | 5 |

Zdroj: Data NSCZ, vlastní zpracování

Tabulka 6-2: Výrobky 2. část

| Kód výrobku | Aktivity | Stroje | Čas operace v sekundách | Pořadí operací |
|---|---|---|---|---|
| **0Y040-S105** | cutting-punching | lis 250t | 9 | 1 |
| | U / O bend | klikový lis 100t | 12 | 2 |
| | TIG weld | svářečka TIG | 32 | 3 |
| | cutting | klikový lis 25t | 9 | 4 |
| | resize | hydraulický lis 100t | 10 | 5 |
| **0Y010-S105** | cutting-punching | lis 160t | 3 | 1 |
| | U / O bend | klikový lis 100t | 9 | 2 |
| | TIG weld | svářečka TIG | 14 | 3 |
| | resize | hydraulický lis 25t | 7 | 4 |
| **0Q020-1500** | cutting-punching | lis 250t | 8 | 1 |
| | blankyng It. | klikový lis 100t | 14 | 2 |
| | U / O bend | klikový lis 100t | | |
| | TIG weld | svářečka TIG | 16 | 3 |
| | cutting | lis 160t | 3 | 4 |
| | U / O bend | hydraulický lis 100t | 10 | 5 |
| | TIG weld | svářečka TIG | 17 | 6 |
| | resize | hydraulický lis 25t | 9 | 7 |
| | kompletace | ruční práce | 9 | 8 |
| | necking | klikový lis 100t s otvorem ve stole | 22 | 9 |
| | MIG weld | svářečka MIG | 28 | 10 |
| **OMO80-T302-10** | cutting-punching | lis 250t | 10 | 1 |
| | U / O bend | klikový lis 100t | 12 | 2 |
| | TIG weld | svářečka TIG | 30 | 3 |
| | tab-cut | klikový lis 25t | 8 | 4 |
| | spot weld | 1-bodová el. svářečka | 26 | 5 |
| **0Q020-R105** | cutting-punching | lis 160t | 3 | 1 |
| | U / O bend | klikový lis 100t | 9 | 2 |
| | TIG weld | svářečka TIG | 14 | 3 |
| | resize | hydraulický lis 25t | 8 | 4 |

Zdroj: Data NSCZ, vlastní zpracování

Tabulka 6-3: Výrobky 3. část

| Kód výrobku | Aktivity | Stroje | Čas operace v sekundách | Pořadí operací |
|---|---|---|---|---|
| **0Y050-S302-10** | cutting | lis 160t | 9 | 1 |
| | ohnutí | hydraulický lis 100t | 10 | 2 |
| | U / O bend | lis 250t | 10 | 3 |
| | TIG weld | svářečka TIG | 32 | 4 |
| | zúžení | klikový lis 100t | 11 | 5 |
| | spot weld jazyk | 8-bodová el. svářečka | 34 | 6 |
| | spot weld | 1-bodová el. svářečka | 26 | 7 |
| **69L00-S310050** | obstřih | lis 250t | 10 | 1 |
| | U / O bend | klikový lis 100t | 10 | 2 |
| | TIG weld | svářečka TIG | 21 | 3 |
| | resize | hydraulický lis 100t | 10 | 4 |
| **69L00-1500** | obstřih | lis 250t 1ks | 7 | 1 |
| | U / O bend | klikový lis 100t 2ks | 12 | 2 |
| | TIG weld | svářečka TIG 4ks | 23 | 3 |
| | U / O bend | lis 250t 1ks | 4 | 4 |
| | TIG weld | svářečka TIG 4ks | 17 | 5 |
| | resize | hydraulický lis 100t 2ks | 8 | 6 |
| | necking | klikový lis 100t s otvorem ve stole 1ks | 21 | 7 |
| | MIG weld | svářečka MIG 2 ks | 28 | 8 |
| **0Q030-1500** | cutting-punching | rychloběžný perforační lis YAMADA | 8 | 1 |
| | punching | klikový lis 100t | 17 | 2 |
| | U / O bend | klikový lis 100t | | |
| | TIG weld,resize | svářečka TIG | 20 | 3 |
| | U / O bend | lis 250t | 4 | 4 |
| | TIG weld,resize | svářečka TIG | 23 | 5 |
| | resize | hydraulický lis 25t | 8 | 6 |
| | kompletace | ruční práce | 9 | 7 |
| | necking | klikový lis 100t s otvorem ve stole | 19 | 8 |
| | MIG weld | svářečka MIG | 28 | 9 |

Zdroj: Data NSCZ, vlastní zpracování

Tabulka 6-4: Výrobky 4. část

| Kód výrobku | Aktivity | Stroje | Čas operace v sekundách | Pořadí operací |
|---|---|---|---|---|
| **0Q030-R106** | cutting-punching | rychloběžný perforační lis YAMADA | 8 | 1 |
| | blankyng It. | klikový lis 100t | 15 | 2 |
| | U / O bend | klikový lis 100t | | 3 |
| | TIG weld | svářečka TIG | 22 | 4 |
| **0Q060-R302-10** | cutting-punching | rychloběžný perforační lis YAMADA | 9 | 1 |
| | blankyng It. | klikový lis 100t | 15 | 2 |
| | U / O bend | klikový lis 100t | | |
| | TIG weld | svářečka TIG | 21 | 3 |
| | resize | hydraulický lis 100t | 9 | 4 |
| | spot weld | 1-bodová el. svářečka | 29 | 5 |
| **62M00-S310080-1500** | cutting-punching | rychloběžný perforační lis YAMADA | 9 | 1 |
| | punching | klikový lis 100t | 18 | 2 |
| | U / O bend | klikový lis 100t | | |
| | TIG weld,resize | svářečka TIG | 23 | 3 |
| | cutting | lis 160t | 3 | 4 |
| | U / O bend | hydraulický lis 100t | 10 | 5 |
| | TIG weld 2 | svářečka TIG | 17 | 6 |
| | resize | hydraulický lis 25t | 8 | 7 |
| | kompletace | ruční práce | 9 | 8 |
| | necking | klikový lis 100t s otvorem ve stole | 23 | 9 |
| | MIG weld | svářečka MIG | 28 | 10 |
| **62M00S310070** | cutting-punching | lis 250t | 9 | 1 |
| | U / O bend | klikový lis 100t | 12 | 2 |
| | TIG weld | svářečka TIG | 23 | 3 |
| | resize | hydraulický lis 25t | 10 | 4 |

Zdroj: Data NSCZ, vlastní zpracování

Tabulka 6-5: Výrobky 5. část

| Kód výrobku | Aktivity | Stroje | Čas operace v sekundách | Pořadí operací |
|---|---|---|---|---|
| **60R00-S310061** | cutting-punching | lis 250t 1ks | 10 | 1 |
| | U / O bend | klikový lis 100t 2ks | 10 | 2 |
| | TIG weld,resize | svářečka TIG 4ks | 21 | 3 |
| | resize | hydraulický lis 100t 2ks | 10 | 4 |
| **60R00-1501** | cutting-punching | rychloběžný perforační lis YAMADA 1ks | 9 | 1 |
| | blankyng It. | klikový lis 100t 2ks | 37 | 2 |
| | U / O bend | klikový lis 100t 2ks | | |
| | TIG weld,resize | svářečka TIG 4ks | 23 | 3 |
| | cutting | lis 160t 1 ks | 9 | 4 |
| | U / O bend | lis 250t 1ks | 10 | 5 |
| | TIG weld 2 | svářečka TIG 4ks | 20 | 6 |
| | resize | hydraulický lis 25t 2ks | 8 | 7 |
| | kompletaceace | ruční práce | 9 | 8 |
| | necking | klikový lis 100t s otvorem ve stole 1ks | 23 | 9 |
| | MIG weld | svářečka MIG 2 ks | 28 | 10 |
| **0N080-S106** | U / O bend | lis 250t 1ks | 10 | 1 |
| | TIG weld | svářečka TIG 4ks | 20 | 2 |
| | roztažení | klikový lis 100t 2ks | 10 | 3 |
| **60R002-10** | cutting-punching | rychloběžný perforační lis YAMADA 1ks | 8 | 1 |
| | U / O bend | klikový lis 100t 2ks | 12 | 2 |
| | TIG weld | svářečka TIG 4ks | 20 | 3 |
| | necking | klikový lis 100t s otvorem ve stole 1ks | 10 | 4 |
| | resize | hydraulický lis 100t 2ks | 10 | 5 |
| | spot weld | 1el. bod svářečka 2ks | 26 | 6 |
| | Bulky | navíječka 1ks | 51 | 7 |

Zdroj: Data NSCZ, vlastní zpracování