Na tomto místě bude oficiální zadání vaší práce

- Toto zadání je podepsané děkanem a vedoucím katedry,
- musíte si ho vyzvednout na studiijním oddělení Katedry počítačů na Karlově náměstí,
- v jedné odevzdané práci bude originál tohoto zadání (originál zůstává po obhajobě na katedře),
- ve druhé bude na stejném místě neověřená kopie tohoto dokumentu (tato se vám vrátí po obhajobě).

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačové grafiky a interakce

Diplomová práce

Rozvrhování s částečně obnovitelnými zdroji

Bc. Michal Vlček

Vedoucí práce: prof. Dr. Ing. Zdeněk Hanzálek

Studijní program: Otevřená informatika, Magisterský

Obor: Softwarové inženýrství

11. května 2015

Poděkování

Děkuji především své rodině, že mi byla oporou a pro jejich podporu při studiu na vysoké škole. Dále bych rád poděkoval vedoucímu práce panu prof. Dr. Ing. Zdeňkovi Hanzálkovi za věnovaný čas a cenné rady.

Prohlášení

Prohlašuji, že jsem práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.
Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 11. května 2015 .....

Abstract

This master thesis aims to algorithms for solving scheduling problem subject to partially renewable resources. This problem is generalization of the well known $\mathcal{NP}$-hard problem Resource Constrained Project Scheduling Problem (RCPSP). There are proposed three algorithms. All of them has been designed and implemented. It is ILP method, List Scheduling heuristic and meta-heuristic based on genetic algorithm. These three are compared with each other on experimental data instances. The biggest part of the work is dedicated to the genetic algorithm.

Abstrakt

Tato diplomová práce se zabývá algoritmy pro řešení problému rozvrhování s částečně obnovitelnými zdroji. Tento problém je zobecněním $\mathcal{NP}$-těžkého plánovacího problému Resource Constrained Project Scheduling Problem (RCPSP). V práci jsou navrženy a implementovány celkem tři algoritmy - ILP metoda, List Scheduling heuristika a metaheuristika založená na genetickém algoritmu. Algoritmy jsou mezi sebou porovnány v experimentech s testovacími daty. Hlavní zaměřením je genetický algoritmus.

Obsah

Seznam obrázků

Seznam tabulek

Seznam zdrojových kódu

Seznam algoritmů

Kapitola 1

Úvod

1.1 Motivační příklad

Představte si následující problém: Je firma s výrobní halou, jež má velké množství zakázek, které je nutné předem - řekněme v horizontu měsíce - naplánovat. Zakázky jsou různorodého charakteru, a tak pro jejich realizaci jsou vyžadovány specifické stroje (soustruhy, frézy, lisy, atp.) - ty mohou zpracovávat vždy maximálně jednu úlohu v čase. Při zpracování zakázky je kromě volných strojů také potřeba určitého množství materiálu (kupříkladu to mohou být šrouby či matice). Těchto materiálů je omezené množství (skladové zásoby) a o jejich zásobování se starají externí dodávky v předem naplánovaných časových intervalech. Je jasné, že k provedení a vyrobení dané zakázky musí být na skladě dostatečné množství materiálů. Je možné, že některé ze zakázek na sebe navazují, čili může nastat situace, kdy jedna z nich (nebo i více) musejí být vykonány před započetím jiné. U zakázek je také předem udán jejich termín dokončení a pokud se nedodrží, naskakují penále, které chceme minimalizovat nebo se jich zcela vyvarovat.

Zadání v bodech:

- množina úloh (řádově stovky až jednotky tisíc)
- různé časy zpracování úloh
- data dokončení úloh
- prioritní vztahy úloh
- přerušení (preempce) úloh není povoleno
- různé typy zdrojů - stroje (obnovitelné) vs. materiály (částečně obnovitelné)
- doplňování materiálů (různé časy a množství)
- paralelní identické procesory
- kriteriální funkce - *total tardiness* (vysvětleno dále v 2.4.1)
- nefunkční požadavek - implementace v C# jazyce

1.2 Cíle

Cílem této práce je nejprve prostudovat zadaný problém v dostupné literatuře. Ze získaných informací vytvořit matematickou formulaci ILP[1] modelu. Dalším bodem je navržení a následná implementace algoritmů spolu s jejich srovnáním při experimentech. Počítá se, že vstupní data by měla být ve formě XLS souboru, čili implementace by měla umět importovat instanční data z tohoto formátu. To samé se týká výstupu algoritmu, výsledek spolu ze zadáním by měl být zpětně exportován ve stejném formátu.

Zbývající text je organizován následovně: V kapitole 2 je detailně zanalyzován a popsán zadaný problém spolu se základní terminologií, bez jejíž znalosti by porozumění textu bylo přinejmenším obtížné. Dále pak v kapitole 3 je zformulován matematický model pro ILP metodu. Metaheuristika založená na genetickém algoritmu je navržena v kapitole 4. V kapitole 5 jsou některé detaily implementace, náhled na celkovou artchitekturu a komponenty aplikace a jejich vzájemný komunikační tok. Předposlední kapitola 6 je vyhrazena pro experimenty s implementovanými algoritmy a na závěr (7) je zhodnocení projektu.

[1] ILP - celočíselné lineární programování

Kapitola 2

Analýza

Tato kapitola shrnuje poznatky získané ze související literatury. V motivačním příkladu popsaném v předchozí kapitole je skryt rozvrhovací optimalizační problém. Konkrétněji se jedná o rozvrhování s omezenými zdroji, které bude v následující sekci představeno. Pro získání detailnějšího pohledu do oblasti kolem RCPSP budou zmíněny související pojmy spolu s metodami řešení.

2.1 RCPSP

Problém rozvrhování s omezenými zdroji (z angl. *Resource-Constrained Project Scheduling Problem*, dále jen RCPSP) je autory *S. Hartmannem* a *D. Briskornem* v [18] jednoduše popsán jako projekt složený z aktivit, které musí být rozvrženy s ohledem na jejich vzájemné vztahy a požadavky na zdroje. To vše za účelem minimalizace celkové délky rozvrhu. Tzv. *precedence* relace definuje vztah mezi některými aktivitami; indikuje, že jedna z aktivit nesmí ve výsledném rozvrhu figurovat před skončením všech jejích předchůdců. Každá aktivita vyžaduje určité množství zdrojů k tomu, aby mohla být provedena. Jakmile aktivita začne být jednou vykonávána, musí být také bez přerušení dokončena (tzn. *preempce* aktivit není povolena). V tomto případě mají zdroje svou iniciální kapacitu a každou časovou periodu se obnovují do této původní kapacity. Proto se jim říká obnovitelné zdroje. Výsledný rozvrh je přiřazení časů spuštění všem aktivitám spolu s přiřazením na konkrétní zdroje. Cílem je najít takový rozvrh, který je časově nejkratší.

RCPSP je mocný a v literatuře velmi známý termín, ale sám o sobě nedokáže pokrýt všechny situace z praxe, proto se v literatuře objevují jeho různé modifikace a zobecnění. Z hlediska složitosti patří do množiny $\mathcal{NP}$-těžkých úloh [6].

2.2 Zdroje

Při srovnání s motivačním příkladem z úvodu je zřejmá určitá podobnost s RCPSP. Stroje si lze představit jako obnovitelné zdroje a aktivity jako jednotlivé zakázky. Jediné, co zatím nelze k ničemu přiřadit, jsou materiály.

Nejprve ale ke zdrojům obecně. V RCPSP se objevují 2 hlavní skupiny zdrojů - *obnovitelné* (renewable) a *neobnovitelné* (nonrenewable) (popř. jejich kombinace) [15]. *Obnovitelné*

znamená, že určitý počet jednotek zdroje je dostupný pro každou časovou jednotku. Mezi zástupce klasických obnovitelných zdrojů patří například lidé nebo stroje. U *neobnovitelných* zdrojů je kapacita vyměřena na celý projekt - nejsou v čase obnovovány. Jakmile je jejich určitá kapacita použita, není zpětně obnovena. Typickým příkladem neobnovitelných zdrojů jsou finance nebo materiál.

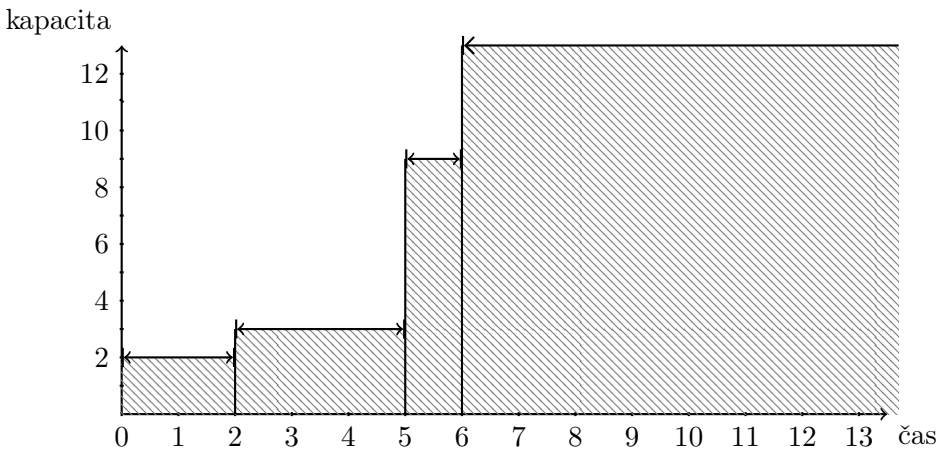
Jejich kombinací jsou tzv. *doubly-constrained* zdroje. Ty mají omezení jak na bázi jednotek času, tak na celý rozvrh. Může tak být modelováno například využití člověka v projektu. Omezení na den je v podobě osmihodinové pracovní směny a omezení na celou délku rozvrhu např. na maximum 100 hodin.

2.2.1 Částečně (ne)obnovitelné zdroje

Částečně (ne)obnovitelné zdroje (v angl. *partially (non)renewable resources*) jsou neobnovitelné zdroje, jejichž dostupnost (kapacita) je určena jen v předem definovaných podmnožinách času (často intervalech [10]).

Matematický popis Označme množinu těchto částečně (ne)obnovitelných zdrojů jako R^π. Pro každý zdroj $k \in R^\pi$ existuje množina Π_k obsahující podmnožiny časových period $P_{kj} \in \Pi_k$. Každá taková podmnožina P_{kj} je asociována s kapacitou a^π_{kj} označující dostupnost zdroje k pro tu danou podmnožinu časů. [15].

Lépe tomu bude porozuměno z následujícího příkladu: Mějme zdroj, který je v časovém intervalu $\langle 0, 2\rangle$ (tedy v periodách $0, 1, 2$) dostupný s kapacitou 2, v $\langle 2, 5\rangle$ s kapacitou 3, v $\langle 5, 6\rangle$ s kapacitou 9 a v intervalu $\langle 6, \infty)$ (až do konce rozvrhu) s kapacitou 13. Kapacita tohoto zdroje v čase je graficky znázorněna na obrázku 2.1.



Obrázek 2.1: Dostupnost/kapacita částečně (ne)obnovitelného zdroje v čase

Částečně (ne)obnovitelné zdroje jsou zobecněný typ zdrojů. Dají se jím namodelovat jak obnovitelné, tak neobnovitelné zdroje (dokonce také *doubly-constrained* zdroje) - jsou to speciální případy tohoto typu [7, 10].

Částečně (ne)obnovitelný zdroj se specifikovanou dostupností v intervalech pro každou časovou jednotku, je v podstatě *obnovitelný* zdroj. Vedle toho zdroj tohoto typu, který je dostupný na jediném intervalu, ale v celé šíři délky rozvrhu, je *neobnovitelný* zdroj.

V [28] zmiňují *financial resources*, které mají podobné chování jako zmíněné „materiály“. Popisují je jako typické neobnovitelné zdroje, které se dynamicky v čase navyšují a jsou konzumovány aktivitami, které o ně soutěží. Dynamické navýšení znamená, že jsou zdroje v každé periodě doplňovány o určité množství. Navrhují polynomiální algoritmus pro 2 speciální připady tohoto problému. Jeden z nich předepisuje konstantní dodávky zdrojů v každém čase, tím se dá předvídat kapacita zdrojů. Druhým případem jsou jednotné požadavky všech aktivit na tento zdroj. Podobně stavěný problém (s tím rozdílem, že aktivity mohou zdroje jak konzumovat, tak také produkovat) je popisován v [21]. V [10] definují tzv. *storage resources*. Jejich dostupnost je závislá na jejich předchozím využití. Jejich aktuání volná kapacita závisí na požadavcích již rozvržených aktivit.

V článku [9] uvažují problém, kdy vykonání aktivity produkuje množství produktu do skladových zásob. Ze skladu jsou produkty odebírány zakázkami, které mohou poptávat více typů produktů. Cílem je minimalizovat výsledný stav zásob při uspokojení všech poptávek. Výsledný stav skladu může být také omezen horní mezí (v reálných podmínkách sklady nemají neomezenené kapacity).

Zajímavým článkem je [14], kde se zabývají speciálními *kumulativními* zdroji spolu s těmi klasickými. Optimalizují délku rozvrhu a kromě specifikace problému také navrhují matematický model spolu s heuristickým řešením pomocí genetického algoritmu. Z tohoto článku bylo čerpáno při tvorbě algoritmu.

V literatuře se u zdrojů, které mají podobné chování jako materiály, vyskytují názvy:

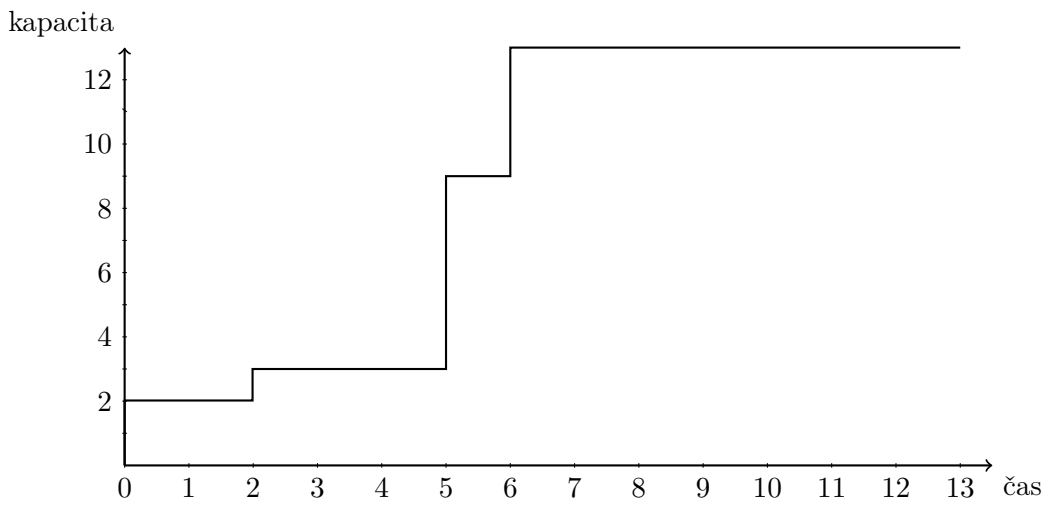
- *financial resources* [28]
- *raw materials* [17]
- *inventory resources* [25]
- *storage resources* [10]
- *cumulative resources* [13]
- *reservoirs* [22]

2.2.2 Kumulativní zdroje

Kumulativní zdroje jsou neobnovitelné zdroje, které mohou být v čase doplňovány, a tím se chtějí příblížit problematice materiálů a součástek. Mají předem daný zásobovací plán, který je zadán ve formě dodávek zdrojů v časových periodách. Tím vznikají časové profily zdrojů [10], které popisují kapacitu zdroje v závislosti na časovém intervalu. Oproti klasickým neobnovitelným zdrojům (které mají konstantní kapacitu v čase) mají kumulativní zdroje proměnnou kapacitu v závislosti na čase. Z předchozích vět lze usoudit, že tento typ zdroje odpovídá popisu „materiálu“.

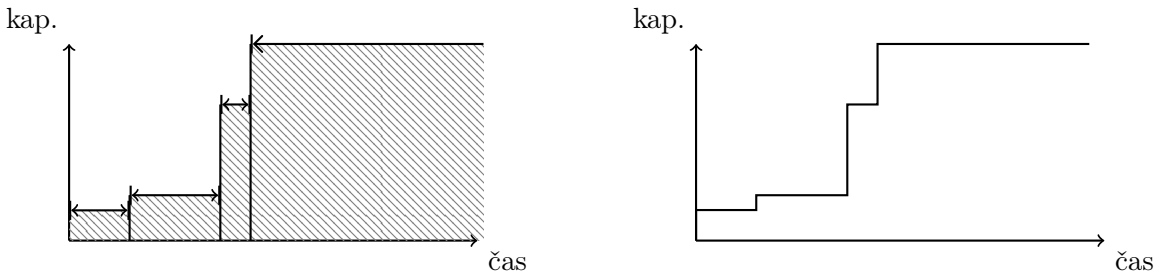
Pozor! Neplést tento typ zdroje s kumulativním zdrojem v kontextu s obnovitelnými zdroji. To jsou zdroje, které mají kapacitu vyšší než 1 a může na nich být zpracováváno více aktivit zároveň. Jejich opakem jsou zdroje disjunktní [10].

Na obrázku 2.2 je ukázán profil kumulativního zdroje, který v čase 0 má kapacitu 2, v čase 2 je doplněn o kapacitu 1, v čase 5 doplněn o 6 jednotek a v čase 6 má kapacitu 13 (doplnění o 4 jednotky). Kdybychom chtěli srovnat klasický obnovitelný zdroj s neměnnou kapacitou, jeho profil by byla vodorovná přímka s y souřadnicí v hodnotě jeho kapacity.



Obrázek 2.2: Profil kumulativního zdroje

Kumulativní zdroje se dají namodelovat jako částečně (ne)obnovitelný zdroj. Na obrázku 2.3 je vidět, že ukázkové příklady z obou typů zdrojů dávají stejný výsledek.



Obrázek 2.3: Srovnání dostupností zdrojů v čase - část. obnov. vs. kumulativní

Pro rozvrhování s tímto typem zdrojů se bude dále v dokumentu používat označení *RCPSP-Cu* (Resource-constrained project scheduling problem subject to cumulative resources), který byl použit v [14].

2.3 RCPSP-Cu

RCPSP-Cu může být formulováno následovně: Je dán projekt, který je složen z:

- n **aktivit**
- R **obnovitelných zdrojů**, u nichž je známa jejich (konstantní) kapacita R_k (kde $k = 1, \ldots, \mathrm{R}$) pro každý časový úsek
- CR **kumulativních zdrojů**, u kterých je určena neklesající kapacita v čase t jako CR_{lt} (kde $l = 1, \ldots, \mathrm{CR}$)

U každé aktivity je specifikována její **doba trvání** (*process time*) p_j ($j = 1, \ldots, n$) a tzv. **požadovaný termín dokončení** (*due date*) d_j (doba, do kdy by měla být aktivita hotova). Každá z aktivit má požadavky na oba typy zdrojů, přičemž požadavek na obnovitelné zdroje r_{jk} je v každé periodě provádění aktivity a požadavek na neobnovitelné zdroje c_{jl} vyžaduje zdroje jen při startu aktivity. U některých aktivit je definován vztah (relace následnosti), kdy aktivita i musí skončit dříve než druhá j může být započata ($i \rightarrow j$). V literatuře je omezení nazýváno *precedence*. Přerušení (preempce) aktivit není povoleno.

Cílem je nalézt splnitelné rozvržení aktivit spolu s minimální hodnotou kriteriální funkce. Rozvrh je ve formě vektoru $\mathrm{S} \in \mathbb{Z}^n$ kde $S_j (j \in \{1, \ldots, n\}; S_j \geq 0)$ obsahují start časy aktivit. Přičemž rozvrh je splnitelný, pokud jsou splněny následující podmínky:

1. *precedence* omezení aktivit
2. omezení aktivit na zdroje - není překročena kapacita zdroje

2.4 Kriteriální funkce

Je rovnice, jejíž výsledek chceme optimalizovat (minimalizovat či maximalizovat). Je to přepis cíle do matematického výrazu. Cílem může být například co nejkratší délka výsledného rozvrhu (v angl. nazývaná *makespan*), to je jedna z nejrozšířenějších kriteriálních funkcí (proto se objevovala ve většině prozkoumaných článků). V našem případě nám ale jde o minimalizaci finančních penalizací vzniklých při nedodržení termínů dodání.

2.4.1 Tardiness

Tardiness je jednou z možných kriteriálních funkcí. Definuje zpoždění aktivity, tzn. přesah jejího průběhu po jejím *due date* (zadaný termín dokončení). Pokud ke zpoždění nedojde, její hodnota je rovna 0. Závisí na času dokončení aktivit F_j, kde zpoždění aktivity j se rovná $T_j T_j := max\{0, F_j - d_j\}$. V zadaném problému se jako optimalizační kritérium bere **total tardiness** - výsledná hodnota je součet všech zpoždění aktivit:

$$\sum_{j=1}^{n} T_j$$

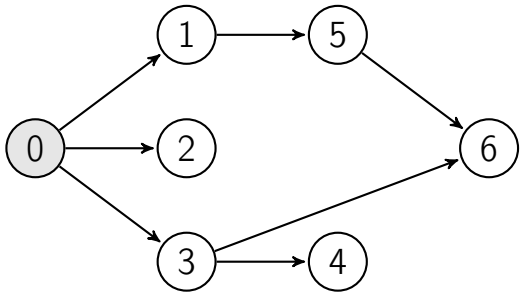
Nyní, při již známé specifikaci, můžeme zadat kompletní ukázkový přiklad.

2.5 Příklad

Je zadána instance projektu s $n = 6$ aktivitami, R = 2 obnovitelnými zdroji, CR = 1 neobnovitelným zdrojem. Kapacita obnovitelných zdrojů je pro první zdroj $R_1 = 2$ a pro druhý $R_2 = 3$. Jediný kumulativní zdroj v zadání je doplňován následovně: v čase 0 má kapacitu 2, v čase 2 je doplněn o 1, v čase 5 o 6, a v čase 6 o 4. Jeho profil je na obrázku 2.2. Relace následnosti je $1 \to 5 \to 6, 3 \to 4, 3 \to 6$ a parametry aktivit:

j	p_j	d_j	r_{j1}	r_{j2}	c_{j1}
1	4	5	0	2	2
2	2	5	2	0	1
3	3	6	1	0	4
4	3	8	0	2	3
5	2	5	1	0	1
6	5	12	0	1	2

Graficky lze relaci následnosti a celý projekt reprezentovat grafem $G = (V, E)$, kde množina vrcholů $V := 1, 2, \ldots, n$ obsahuje aktivity a množina hran $E = \{(i, j) | i, j \in V; i \to j\}$ reprezentuje precedence omezení. Takto reprezentovaný projekt s aktivitami ve vrcholech se označuje AON (*activity on node*) reprezentace [10]. Na obrázku 2.4 je zakreslen graf instance ze zadaného příkladu, vrchol 0 je pomocný (nepatří do aktivit ze zadání).

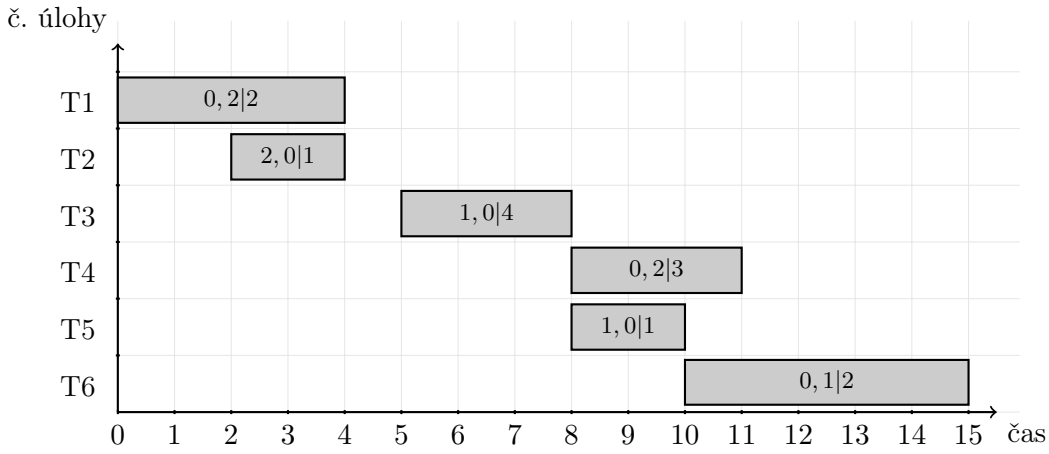


Obrázek 2.4: *Activity on node* reprezentace

Jeden z možných rozvrhů (ne nejlepší!) je zakreslen do grafu na obrázku 2.5. Výsledný rozvrh má řazení aktivit $1, 2, 3, 4, 5, 6$, které splňuje všechny omezení. Aktivity jsou popsány svými požadavky na zdroje - $r_1, r_2 | c_1$. V grafu je v intervalu (4,5) viditelná pauza, žádná aktivita nemůže být zahájena, protože je nedostatek kumulovaného zdroje - čeká se na jeho dodávku, která přichází v čase 5. Další zajímavostí je začátek aktivity 6 až v čase 10, ta totiž musí začít díky vzájemným relacím až po skončení aktivity 5.

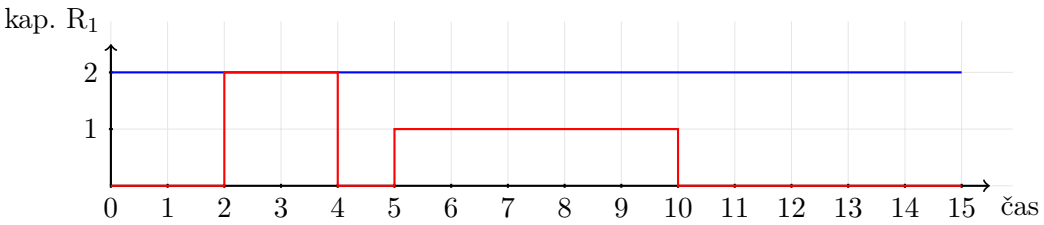
Hodnota kriteriální funkce, kvůli které se rozvrhuje, je rovna 13. Přispívá do ní:

- aktivita 3 (končí v čase 8, ale $d_3 = 6$, proto $T_3 = \max\{0, 8 - 6\} = 2$),
- aktivita 4 ($T_4 = 3$),
- aktivita 5 ($T_5 = 5$),
- aktivita 6 ($T_6 = 3$).

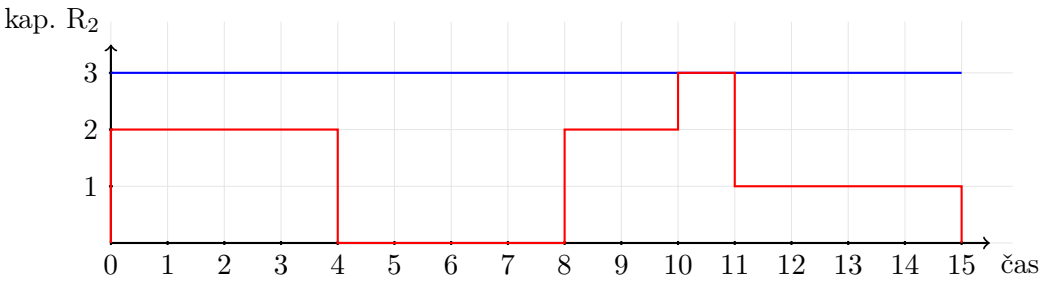


Obrázek 2.5: Graf pro pořadí aktivit $1, 2, 3, 4, 5, 6$ - total tardiness $= 13$

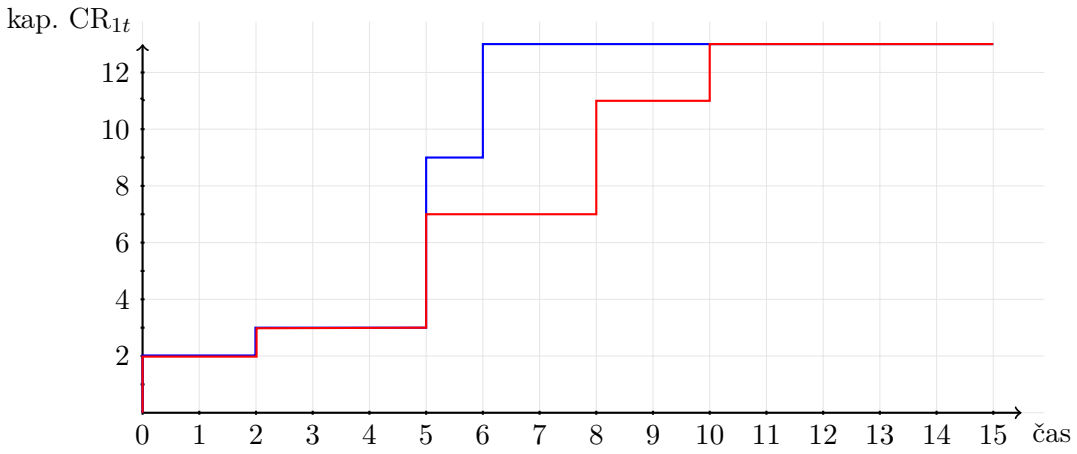
Na obrázcích 2.6, 2.7, 2.8 jsou profily kapacit jednotlivých zdrojů (modrá křivka) a jejich aktuální vytížení od rozvržených aktivit (červená křivka). Aby bylo dodrženo omezení na kapacitu zdrojů, nesmí být požadavky aktivit vyšší než kapacita zdroje (nesmí překročit profilovou křivku), a to v jakémkoli čase. U obnovitelných zdrojů (obr. 2.6 a 2.7) jsou zdroje naplno využívány jen chvílemi. Kumulativní zdroj 2.8, je kvůli dodávkam až v průběhu rozvrhu využíván plně téměř neustále - až do času 5 jsou plně využité kapacity (křivky se překrývají) - to lze i vidět na obrázku 2.8.



Obrázek 2.6: Křivka vytížení obnovitelného zdroje r_1

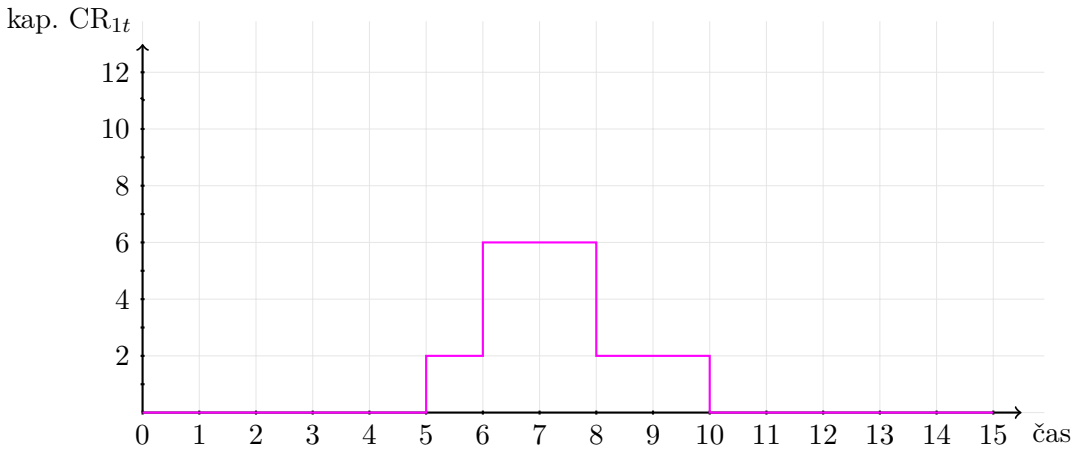


Obrázek 2.7: Křivka vytížení obnovitelného zdroje r_2



Obrázek 2.8: Křivka vytížení kumulativního zdroje c_1

Na obrázku 2.9 je ještě ukázána volná/nevyužitá kapacita kumulativního zdroje v čase (např. aktuální stav zásob materiálu na skladě). Graf je získán odečtením křivek z předchozího grafu 2.8. Modrá křivka kapacita zdroje a červená je jeho aktuální kapacitní vytížení, jejich odečtením získáme volnou kapacitu (fialová barva).

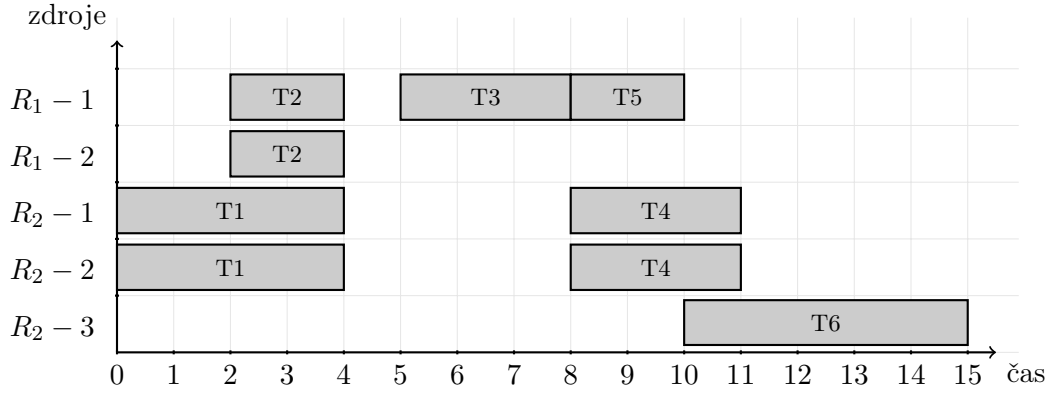


Obrázek 2.9: Křivka volné/nevyužité kapacity kumulativního zdroje c_1

Pokud vytvoříme rozvrh pro danou instanci, chtěli bychom vědět, na jakých konkrétních jednotkách zdrojů jsou aktivity rozvrženy. Když toto tvrzení převedeme do praxe, tak máme například 3 fyzické jednotky soustruhu (kapacita = 3) a chceme vědět, na které z nich bude aktivita zpracována. Pro každý zdroj k označíme jeho jednotky pořadovým číslem od 0 do $R_k - 1$. Aktivity mohou vyžadovat více typů zdrojů, a to v množství větším než jedna jednotka. Proto údaj, na jakých jednotkách aktivita běží, budeme udávat v rozsahu pořadových čísel těchto jednotek. Například rozsah $\langle 0, 1\rangle$ značí, že aktivita používala zdroje s číslem 0 a 1. V tabulce 2.1 je ukázáno jedno z možných přirazení aktivit na jednotky zdroje pro ukázkový příklad.

č. aktivity	č. zdroje	rozsah jednotek zdroje
1	1	$\langle 0, 1 \rangle$
2	0	$\langle 0, 1 \rangle$
3	0	$\langle 0, 0 \rangle$
4	1	$\langle 0, 1 \rangle$
5	0	$\langle 0, 0 \rangle$
6	1	$\langle 2, 2 \rangle$

Tabulka 2.1: Přiřazení aktivit na jednotky zdrojů



Obrázek 2.10: Ganttův graf využití zdrojů

2.6 Metody řešení

Při řešení rozvhovacích problémů (a kombinatorických problémů obecně) se zpravidla používají 2 hlavní metody - **exaktní** vs. **heuristické** přístupy.

Exaktní (nebo také úplné simplexní) metody jsou postupy, které procházejí celý prostor (doménu) řešení a tudíž dokáží naleznout optimální řešení. Jejich hlavním problémem je časová náročnost, proto na větší instance jejich použitelnost klesá. Zástupci úplných metod jsou např. Branch&Bound nebo ILP.

Na druhou stranu heuristika je postup, při kterém si například zvolníme nějaké omezení za cílem získat výsledek v rychlejším čase. Tento přístup neprochází celý prostor řešení, a proto nám nezaručuje nejlepší řešení. Metoda je založena na důkladné znalosti řešeného problému, kdy pomocí jednoduché procedury jsme schopni získat rychlé a dostatečně kvalitní řešení. Tyto přístupy se v praxi často využívají při řešení velkých instancí náročných problémů, na které nejsou dostatečné HW prostředky a spokojíme se s neoptimálním řešením.

2.6.1 ILP

Celočíselné lineární programování je metoda pro výpočet matematických optimalizačních problémů, jejíž cílem je optimalizovat (minimalizovat nebo maximalizovat) lineární kriteriální funkci, která neporušuje omezení definovaná v podobě lineárních rovnic či nerovnic [8]. Formálněji zapsáno má obecná minimalizace zápis

$$\begin{aligned} \min \quad & \mathbf{c}^T \cdot \mathbf{x} \\ & \mathbf{A} \cdot \mathbf{x} \leq \mathbf{b} \end{aligned}$$

kde matice $\mathbf{A} \in \mathbb{R}^{m \times n}$ s vektory $\mathbf{b} \in \mathbb{R}^m$ a $\mathbf{c} \in \mathbb{R}^n$ definují problém. Cílem je najít takový vektor $\mathbf{x} \in \mathbb{Z}^n$, jehož hodnoty minimalizují kriteriální funkci $\mathbf{c}^T \cdot \mathbf{x}$ a všechna omezení $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}$ jsou splněna. Každý řádek matice $\mathbf{A}$ s hodnotou vektoru $\mathbf{b}$ odpovídá lineární nerovnici.

2.6.2 Schémata pro generování rozvrhu

Schémata pro generování rozvrhu v angličtině Schedule Generation Schemes (SGS) jsou jádrem většiny rozvrhovacích heuristik [19]. Je to procedura, která krok po kroku vytvoří rozvrh postupným vkládáním aktivit. Rozlišuje se sériové nebo paralelní generovací schéma. Sériové má n (počet aktivit) iterací a v každé vybírá jednu aktivitu, kterou je možné v dané chvíli vložit do rozvrhu (má všechny své předchůdce již v rozvrhu). Vloží ji na nejbližší možný čas s přihlédnutím na omezení zdrojů a předchůdců. Při výběru vhodné aktivity se může stát, že takových aktivit je více. Pak přichází na řadu prioritní pravidla, podle kterých se aktivita vybírá. Paralelní verze neiteruje po aktivitách, ale po časech, do kterých se rozvrhují aktivity.

2.6.3 Prioritní pravidla

Pravidel pro výběr aktivit je celá řada. V [17] používají řazení aktivit sestupně dle termínů dokončení d_j - EDD (*Earliest Due Date*). V [12] je popsáno řazení upřednostňující aktivity s delší dobou trvání, které vyžadují menší množství zdrojů. V [13] je popsáno pravidlo, které řadí podle součtu časů nejzazšího možného startu a konce aktivity (*latest start time + latest finish time* - LSTLFT). Článek [23] shrnuje několik dalších pravidel, přičemž nejlepších výsledků pro minimalizaci délky rozvrhu zde dosahuje právě LSTLFT. Pro naše účely (minimalizace *total tardiness*) se zdá být vhodné řazení dle termínů dokončení. Použitá pravidla budou popsána v kapitole 4.

2.6.4 List Scheduling

List scheduling [26] je jedna ze známých heuristik při řešení RCPSP za účelem minimalizace délky rozvrhu, která ve svém jádru využívá právě SGS. Její důležitou součástí je list (seznam) aktivit, který předáváme jako vstupní data. Tudíž generační schéma nevybírá aktivitu samo, ale pořadí k rozvržení již má předem definované ze zadaného aktivity listu (AL).

Sériové SGS generuje *aktivní rozvrhy* [12]. To jsou takové rozvrhy, kde žádná z aktivit nemůže být rozvržena dříve, aniž by se musela posunout (zpozdit) jiná. Z toho vyplývá, že pokud předáme algoritmu správně seřazené aktivity, pak dostaneme optimální řešení [19].

Při vhodném výběru pravidla pro řazení aktivit a upravení schématu pro kumulativní zdroje se tato heuristika dá aplikovat také na *RCPSP-Cu*.

2.6.5 Metaheuristiky

Heuristiky jsou závislé na problému, vztahují se k němu a jsou určeny právě k jeho řešení.

Na straně druhé stojí metaheuristiky. Ty jsou nezávislé na řešeném problému, jsou to pouze strategie k prohledávání jeho prostoru řešení. Ve svém vnitřku mohou používat konkrétní heuristiku k obstarání nějakého dočasného řešení. S ním nadále pracují a pomocí různých úprav (např. prohození aktivit) procházejí prostor řešení k získání lepšího výsledku. I když bylo řečeno, že jsou problémově nezávislé, mohou být parametrizovány, a tím i lépe adaptovány na konkrétní problém. Zástupci metaheuristik jsou genetické algoritmy (GA) či simulované žíhání (SA). Více o metodách obecně a jejich souhrn je v [19].

V této práci bude implementována jedna úplná metoda pomocí ILP, jedno heuristické řešení v podobě List schedulingu a jedna metaheuristika založená na Genetickém algoritmu.

Kapitola 3

Model

Pro přehlednost je v následující tabulce[1] shrnuta notace parametrů, které se budou v modelu používat. Velká část omezení v modelu je převzata z [14].

n	- počet aktivit
R	- počet obnovitelných zdrojů
CR	- počet kumulativních zdrojů
R_k	- kapacita zdroje k
CR_{kt}	- kapacita kumulativního zdroje k v čase t
P_j	- množina předchůdců pro aktivitu j
p_j	- doba trvání aktivity j
d_j	- termín dokončení aktivity j
r_{jk}	- požadavek na zdroj k aktivity j
c_{jl}	- požadavek na kumulativní zdroj l aktivity j
T	- horní mez (odhad) času
S_j	- čas startu aktivity j v rozvrhu
F_j	- čas dokončení aktivity j v rozvrhu

Tabulka 3.1: Zástupné symboly parametrů

V modelu je několik omezujících nerovnic vyjadřujících všechna omezení a jedna rozhodovací proměnná x_{jt}, která indikuje, pokud je aktivita j započata v čase t. Protože je proměnná indexovaná časem (jedná se o tzv. *time-indexed* model [21]), musí se stanovit časový horizont T, který shora omezuje rozvrh - všechny aktivity se musí rozvrhnout do času T. Čas je rozdělen do period jednotné délky a do těchto period se rozvrhuje [5].

$$x_{jt} = \begin{cases} 1, & \text{pokud aktivita } j \text{ začne v čase } t. \\ 0, & \text{jinak.} \end{cases}$$

[1] notace je dostupná i jako samostatný list v příloze 7

3.1 Horní mez času - T

Horní mez chceme stanovit co nejnižší, protože tím snížíme počet proměnných v modelu, a tím zvýšíme jeho výkonnost - zvýší se rychlost výpočtu. Horní odhad T se musí nějakým způsobem stanovit už ze zadání. Je to nejnižší hodnota délky rozvrhu, kterou dokážeme určit ještě před započetím rozvrhování. Nyní již k samotnému stanovení odhadu:

Rozvrh musí určitě obsahovat všechny rozvrhované aktivity, a proto je do odhadu zakomponován součet jejich dob trvání p_j. To představuje vložení všech aktivit za sebe; tím je zaručeno splnění omezení na kapacity obnovitelných zdrojů. Tato hodnota odhadu ale počítá s tím, že materiály pro vykonání všech aktivit jsou již dostupné. Toto ale není zaručeno, protože materiály mohou být dodávány až v průběhu rozvrhu. Celá kapacita materiálů bude dostupná až po jejich poslední dodávce. K vypočitané hodnotě se tedy dále připočte čas poslední dodávky materiálů. Proměnná $times_l$ značí časy dodávek materiálu l.

$$T = \sum_{j=1}^{n} p_j + \max\{times_l\}, \qquad l = 1..\mathrm{CR}$$

Časový odhad k průvodnímu příkladu je po dosazení spočten jako:

$$T = (2 + 4 + 3 + 3 + 2 + 5) + \max\{0, 2, 5, 6\} = 19 + 6 = 25$$

3.2 Matematický model

$$\min \quad \sum_{j=1}^{n} \sum_{t=0}^{T} \max\{0, x_{jt}(t+p_j-d_j)\} \tag{3.1}$$

$$\sum_{t=0}^{T} x_{jt} = 1, \qquad j = 1..n \tag{3.2}$$

$$\sum_{t=0}^{T} x_{it}(t+p_i) \leq \sum_{t=0}^{T} x_{jt} \cdot t, \qquad j = 1..n, i \in P_j \tag{3.3}$$

$$\sum_{j=1}^{n} \sum_{\tau=t-p_j}^{t} r_{jk} x_{j\tau} \leq \mathrm{R}_k, \qquad k = 1..\mathrm{R}, t = 0..T \tag{3.4}$$

$$\sum_{j=1}^{n} \sum_{\tau=0}^{t} c_{jl} x_{j\tau} \leq \mathrm{CR}_{lt}, \qquad l = 1..\mathrm{CR}, t = 0..T \tag{3.5}$$

$$x_{jt} \in \{0, 1\}, \qquad j = 1..n, t = 0..T \tag{3.6}$$

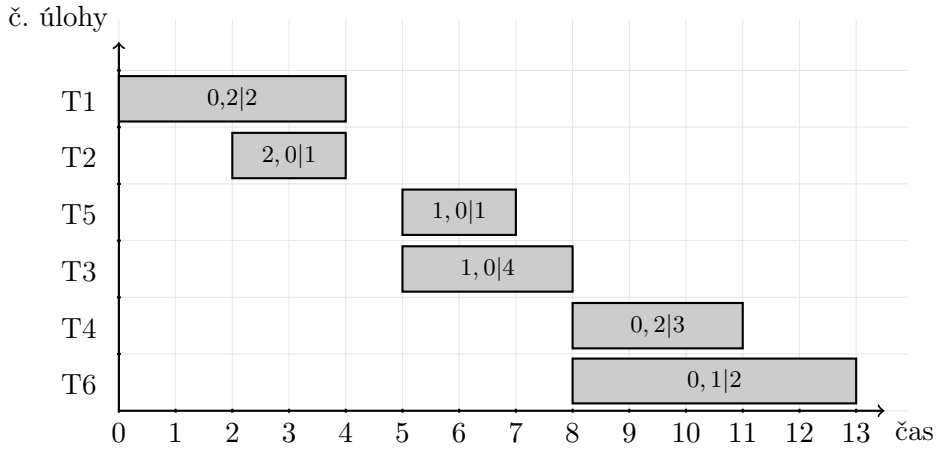
V modelu výše je kriteriální funkce (3.1) *total tardiness*. Rozhodovací proměnná určuje čas započetí aktivity, ze kterého se dopočte čas dokončení a následné zpoždění. První omezení (3.2) zajišťuje, že každá aktivita bude zahrnuta do rozvrhu, tzn. že v nějakém čase bude započata. Zároveň také říká, že v rozvrhu bude přesně a jen jednou. Každá aktivita j musí mít jen jeden čas zahájení t, tzn. $x_{jt} = 1$.

Další omezení (3.3) zajišťuje splnění relace následnosti mezi aktivitami. Pro všechny předchůdce $i \in P_j$ aktivity j musí platit, že čas, kdy skončí $x_{it}(t + p_i)$, musí být menší nebo roven času $x_{jt} \cdot t$, kdy začíná aktivita j. Podobné omezení by šlo namodelovat s následovníky.

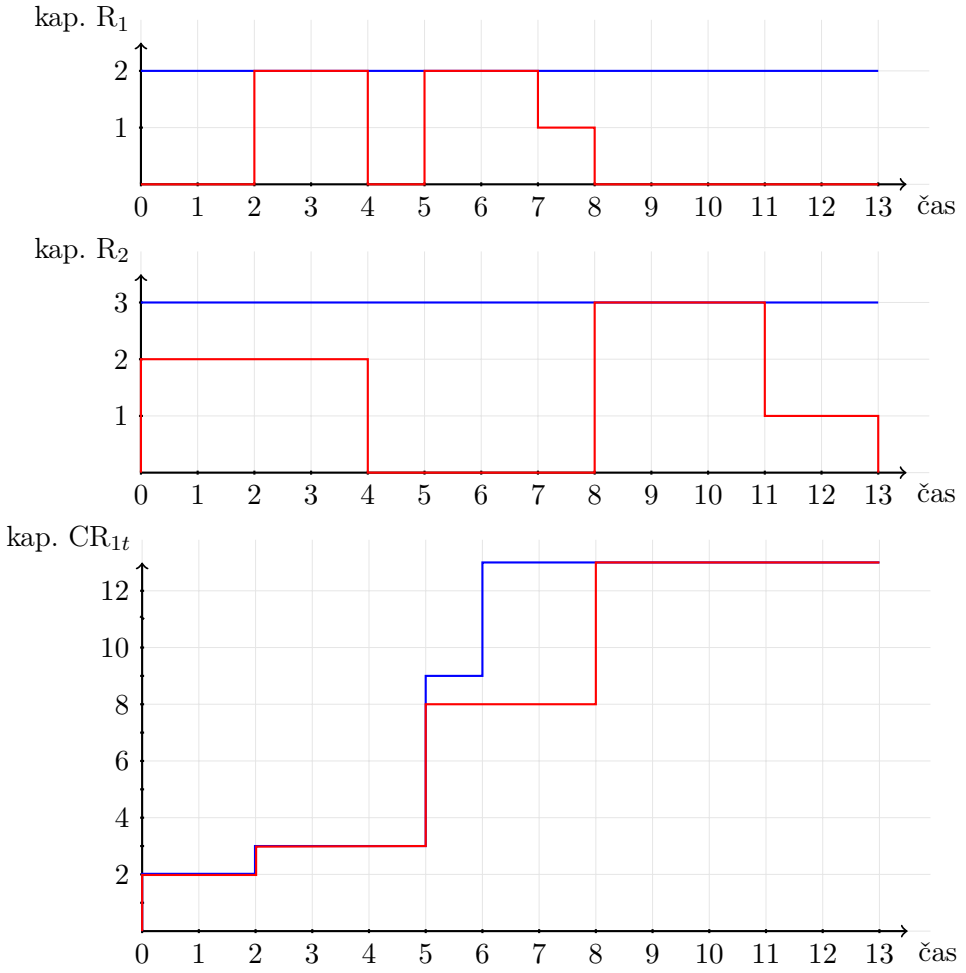
Skupina omezení (3.4) a (3.5) se týká zdrojů - dodržení jejich kapacit. U obnovitelných zdrojů se posčítají požadavky aktivit r_{jk} probíhající ve stejném čase, toto číslo nesmí překročit kapacitu zdroje R_k. To se kontroluje pro každý zdroj k v každém čase t. Pro čas t se spočítají aktuálně probíhající aktivity prozkoumáním intervalu $\langle t - p_j, t \rangle$, protože aktivita j může ovlivňovat čas t pouze zde. U kumulativních zdrojů se požadavky c_{jl} rozvržených aktivit vzhledem k času t načítají už od začátku rozvrhu a kontrolují proti kapacitě zdroje CR_{lt} v čase t (ta odpovídá hodnotě z časového profilu zdroje).

Poslední rovnice (3.6) obsahuje jen omezení oboru hodnot rozhodovací proměnné. Protože je to binární rozhodovací proměnná, její možné hodnoty jsou rovny 0 nebo 1.

ILP model zmíněný výše byl naprogramován v jazyku JAVA za využití knihoven IBM ILOG CPLEX [1] *solveru*. Solver je matematická knihovna, která obsahuje pokročilé mechanismy k řešení náročných matematických problémů. Na následující straně je v grafu 3.1 zanesen optimální rozvrh s hodnotou kriteriální funkce 8, který byl vrácen ILP solverem. Na dalších grafech 3.2 jsou grafy využití zdrojů pro tento rozvrh. Rozvrhy jsou pro příklad uvedený v předchozí kapitole.



Obrázek 3.1: Graf optimálního rozvrhu $(1, 2, 5, 3, 4, 6)$ - total tardiness $= 8$



Obrázek 3.2: Křivka vytížení zdrojů r_1, r_2 a c_1

Kapitola 4

Genetický algoritmus

V zadaném problému se při praktickém využití počítá s velkým počtem rozvrhovaných aktivit (stovky až tisíce), a proto exaktní metoda získání optimálního rozvrhu není příliš vhodná. K získání optimálního rozvrhu by musela prozkoumat v nejhorším případě všechny možné uspořádání aktivit, tzn. $n!$ možností. To se s reálnými HW prostředky nedá dost dobře stihnout. Proto se využívají metaheuristické přístupy, které mají postupy, jak pokrýt/projít co největší oblasti v celém prostoru řešení, a z nich vybírat ta nejlepší uspořádání. Jedním z přístupů je genetický algoritmus.

4.1 Provázanost s přírodou

Genetický algoritmus (GA) je evoluční přístup k řešení určitého problému, inspirovaný Darwinovou evoluční teorií. Tato prohledávací technika je založena na konceptu přírodního výběru a evoluce. Evoluce živočišného druhu je chápána jako postupný a přirozený vývoj života od prvního výskytu na Zemi až k mnoha různým formám života.

Genetický algoritmus byl poprvé představen Johnem Hollandem v 70. letech minulého století. Holland věřil, že kdyby se principy přírodní evoluce převedly do počítačového algoritmu, mohl by vzniknout nový směr, jak řešit náročné problémy [27].

Za účelem vyřešení úlohy se algoritmu nejprve předají náhodná iniciální řešení, která se vyhodnocují a postupně „šlechtí“ k lepším a lepším výsledkům. Iniciální řešení chceme co nejvíc náhodná, aby se co nejvíce rozprostírala po celém prostoru řešení.

Hlavní myšlenka GA spočívá v tom, že na jednotlivé prvky množiny přípustných řešení pohlížíme právě jako na nějakého živočicha, v jehož genech je nějakým způsobem zakódována informace, která popisuje jedno přípustné řešení. To, jak si tyto jedinci vedou, tj. jejich schopnost přežít a schopnost reprodukce, odpovídá tomu, o jak „dobrá“ řešení se jedná. Samotné hledání řešení pak spočívá ve výběru nějaké počáteční populace těchto živočichů a v následné simulaci jejího vývoje pod kontrolou evolučních mechanismů zahrnujících přirozený výběr, reprodukci, atd. Jak se tato populace z generace ke generaci vyvíjí, „špatná“ řešení mají tendenci vymírat, a naopak „dobrá“ řešení se mezi sebou kříží a mutují a produkují řešení ještě lepší [24].

4.2 Jak GA funguje

Prvním krokem je vytvoření nějaké počáteční populace přípustných řešení. Ta se zpravidla generuje náhodně, ale můžou se využít nějaká počáteční řešení získaná některou heuristikou. Před samotnou tvorbou populace se musí stanovit, v jaké formě budeme jedince reprezentovat, protože na ně budeme chtít aplikovat genetické operátory. Jako dostatečně obecné se jeví kódování ve formě řetězce či pole hodnot. Toto kódování má také svou analogii v genetice, kdy v podstatě řetězce odpovídají *chromozomům* a jednotlivé pozice v řetězci jednotlivým *genům* (a konkrétní hodnoty genů pak *alelám*). Každé řešení je jako jedinec v populaci reprezentován jedním řetězcem - chromozomem - proto se vedle označení jedinec používá také označení chromozom.

Po vzniku populace přichází její ohodnocení za účelem získání nové generace jedinců. To spočívá v ohodnocení každého jejího jedince (*fitness*) pomocí dodané ohodnocovací funkce tzv. *fitness function*. Při tvorbě nové generace přichazí nejprve výběr jedinců, kteří se budou podílet na jejím vzniku. Pravděpodobnost výběru daného jedince je úměrná jeho relativnímu ohodnocení. Postupů pro selekci jedinců je několik, jedním z nich je *turnaj*. Ten vezme například 4 náhodné jedince a z nich vybere toho s nejlepším ohodnocením. Samotný výběr ještě samozřejmě nezpůsobí vznik žádných novým jedincům. Od toho jsou tu genetické operátory křížení a mutace.

Pro křížení (*crossover*) je nejprve nutné vybrané jedince spárovat (proto se typicky volí sudá velikost populace). Na každý pár (rodiče) je pak aplikováno křížení s nějakou předem stanovenou pravděpodobností (obvykle velmi vysokou, např. 0,9). Křížením se vyměňují geny mezi dvěma jedinci. V nejjednodušším případě se může jednat např. o to, že se náhodně určí pozice v chromozomu a od této pozice se vymění geny. Vždy je však třeba volit takový mechanismus, aby křížením vznikl opět řetězec, který reprezentuje nějaké přípustné řešení [24]. Výsledkem křížení je nový potomek, který je kandidátem do následující generace. Velikost populace v GA je konstantní, křížení jedinců se opakuje do té doby, než ji nově vzniklí potomci zcela naplní.

Posledním krokem při tvorbě nové generace je aplikování mutace *mutation* na každého kandidáta. Mutace slouží k udržování genetické diverzity v populaci a tím brání, aby proces hledání sklouznul jen do oblasti nějakého lokálního optima. V binárním řetězci by se typicky jednalo o náhodné prohození 0 za 1. I zde, podobně jako u křížení, je třeba dbát na to, aby mutací narušený řetězec opět reprezentoval nějaké přípustné řešení. Mutace však významně narušuje podobu jedince, proto by měla být její aplikace omezena nějakou poměrně malou pravděpodobností (mezi 0,001 a 0,2) [24].

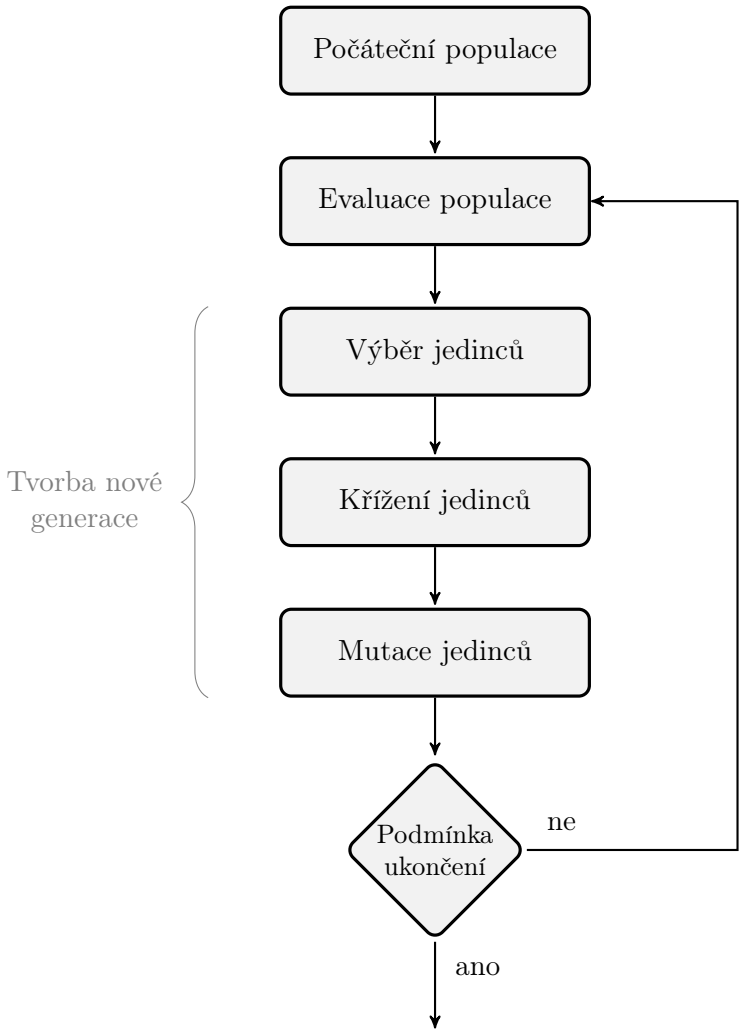
Za účelem rostoucí kvality generací se do nové generace může vkládat malé procento těch nejlepších jedinců z předchozí generace. Na tyto elitní jedince se neaplikují žádné genetické operátory - jedinec zůstane takový jaký je. Tomuto procesu se říká výběr elitních jedinců a procento těchto jedinců bývá velmi malé (max do 5 %), to z důvodu dostatečné diverzifikace. Typicky se tento výběr uskutečňuje na začátku, kdy se položí základ nové generace právě těmito jedinci, a ti jsou pak doplněni novými zkříženými a zmutovanými jedinci.

Proces získávání nových generací řešení se opakuje do té doby, dokud není splněna nějaká ukončovací podmínka (*termination condition*). Ta může být definována např. jako časový limit nebo omezení počtu generací. Při naplnění podmínky algoritmus končí a vrací nejlépe ohodnocené řešení.

Pravděpodobnostní hodnoty nejsou předepsané, jsou pouze orientační, a pro každý problém může být jejich nastavení mírně odlišné. Hodnoty se zkoušejí při experimentování a ze získaných výsledků se zvolí nejvhodnější hodnota.

Hlavní bloky jsou shrnuty v následujícím seznamu a v diagramu na obrázku 4.1 je pak pro přehled zakreslena kompletní struktura spolupráce těchto bloků.

- Tvorba počáteční populace
- Vyhodnocení populace
- Tvorba nové generace z předchozí
- Terminační kritérium



Obrázek 4.1: Proces Genetického Algoritmu

Následující sekce detailněji popisují reprezentaci jedince a jednotlivké bloky diagramu.

4.3 Reprezentace jedince

Volba správné reprezentace jedinců hraje jednu z důležitých rolí při implementaci algoritmu. U každého problému se může hodit trochu jiná reprezentace a má velký vliv na výslednou efektivitu algoritmu.

Cílem je zvolit co nejjednodušší reprezentaci, ze které snadno získáme ohodnocení jedince a genetické operátory s ní mohou pohodlně pracovat. Zde byla zvolena reprezentace seznamem aktivit - AL (*activity list*). V praxi je to pole hodnot složené z n genů, kde každý gen obsahuje číslo aktivity. Z toho vyplývá, že v chromozomu jedince se nesmí opakovat žádný gen (každá hodnota musí být unikátní) - jedná se o permutaci. Pořadí aktivit v poli určuje jejich prioritu, podle které se bude tvořit rozvrh. Reprezentaci jedince (tedy jeho pole) si lze představit jako na obrázku 4.2.

Tato reprezentace byla zvolena kvůli její nižší režii. Při dekódování jedince se nemusí nic provádět a rovnou máme obstarané pořadí aktivit, které potřebujeme později při ohodnocení jedince. Jediná údržba vzniká při tvorbě nových jedinců aplikováním genetických operátorů, ale o tom až dále.

1	2	3	4	5	6

Obrázek 4.2: Reprezentace jedince jako *Activity List* (AL)

4.4 Preprocessing

Před samotným během algoritmu, je dobré provést základní kontrolu, zda instance má vůbec nějaké řešení, tzv. *preprocessing* fázi.

Můžou nastat 3 případy, kdy instance nemá řešení. Prvním z nich je situace, kdy jedna z aktivit má požadavek na obnovitelný zdroj vyšší než je jeho kapacita. Druhým případem je nedostatek materiálů. Součet požadavků na kumulativní zdroje nesmí být větší než je počet celkem dodaných zdrojů. Poslední věcí, která může zapříčinit neřešitelnou instanci, jsou precedenční vztahy aktivit. Grafová reprezentace tohoto vztahu musí být acyklický graf; při vzniku cyklu by např. aktivita A měla předchůdce aktivitu B a naopak. Tím by nešlo v rozvrhu zajistit precedence omezení. Pokud žádný z těchto případů nenastane, tak pro instanci existuje nějaké řešení.

4.5 Počáteční populace

Do počáteční populace jsou generována náhodná přípustná řešení. Díky tomu lze na začátku získat dostatečně různorodé jedince. Avšak generování není zcela náhodné, protože je požadováno, aby vznikající AL byly *feasible* (aby splňovaly precedence omezení aktivit). Tím se vyloučí tvorba zbytečně vznikajících jedinců, ze kterých by stejně nevzniklo žádné splnitelné řešení. Splnitelné AL jsou generovány následovně [14]:

1. Prázdný seznam (AL).
2. Určení množiny způsobilých aktivit EA (tzv. *eligible activities*). Vkládáme takové aktivity, které již mají všechny své předchůdce obsažené v AL a sami nejsou v AL.
3. Z množiny EA se vybere náhodně jedna aktivita a vloží se na konec AL.
4. Pokud jsou všechny aktivity v AL, je konec. Pokud ne, pokračuje se krokem 2.

Pseudokód sestavení AL je v algoritmu 4.1. Algoritmus má n iterací, v každé z nich se do AL vloží jedna aktivita, která v něm ještě není (první podmínka na řádku 4). Po doběhnutí vnější smyčky se v AL nachází n aktivit a algoritmus tímto vrací správně sestavený AL.

Pseudokód 4.1 Tvorba seznamu aktivit - AL

Input: J - množina aktivit
Input: P - pole s množinami předchůdců aktivit

```
 1: AL = {}, EA = {}                # EA obsahuje v každé iteraci vhodné aktivity do rozvrhu
 2: for 1...n do
 3:     for j ∈ J do
 4:         if j ∉ AL and P_j ⊆ AL then
 5:             přidej j do EA      # aktivity, které mají předchůdce v AL a sami v něm nejsou
 6:         end if
 7:     end for
 8:     (náhodně) vyber aktivitu i z EA
 9:     přidej i do AL
10:     vyprázdni EA
11: end for
12: return AL
```

V počáteční populaci jsou nyní náhodná řešení díky náhodnému výběru aktivity z EA. To sice zaručuje různorodost populace, ale nikoliv její kvalitu. Ta je získána pomocí prioritních pravidel, která určí aktivitu ke zvolení z EA. Některá pravidla byla popsána v sekci 2.6.3. V praxi se iniciální populace tvoří následovně: nejdřív se vytvoří jedinci pomocí prioritních pravidel a zbytek populace je doplněn náhodnými.

Nová pravidla V algoritmu je využito a navrženo několik pravidel. RCPSP-Cu pracuje s kumulativními zdroji, a tak se přímo vybízí je zakomponovat a využít v prioritních pravidlech. Bylo jich navrženo hned několik a všechny rozšiřují stávající EDD řazení. Pokud se v danou chvíli může rozvrhnout hned několik aktivit, protřídí se z nich ty, co mají nejnižší *due date*, a z této profiltrované množiny se dále aplikují nová pravidla.

U některých pravidel se používá pojem EST (*Earliest Start Time*). To značí nejdřívější možný čas, kdy může být aktivita vzhledem k relaci následnosti spuštěna a omezení na zdroje jsou potlačeny.

aktivita	p_j	d_j	***EST***	**požadavky**
1	4	5	0	2 + 2
2	2	5	0	2 + 1
3	3	6	0	1 + 4
4	3	8	3	2 + 3
5	2	5	4	1 + 1
6	5	12	6	1 + 2

Tabulka 4.1: Hodnoty používané v pravidlech

- **EDD** - Vybere se první aktivita s nejmenším *due date*.
- **EDD-Ma** - z EDD množiny aktivit se vybere ta s nejmenšími požadavky na kumulativní zdroje.
- **EDD-ESTReMa** - z EDD množiny aktivit se vybere ta s nejmenším EST spolu s nejmenším součtem požadavků na oba typy zdrojů. (První aktivita z ukázkového příkladu má součet požadavků obnovitelných zdrojů 2, kumulativních také 2 a EST, protože nemá žádného předchůdce, je rovno 0. To dává hodnotu 4).
- **EDD-ESTPTReMa** - U tohoto pravidla se do součtu přidá navíc *process time*.

Bylo zvolen větší počet pravidel (různé kombinace předchozích), protože pro každý typ instancí je každé pravidlo méně či více vhodné. Vybráním více pravidel je pak pokryto více druhů instancí. Pro instance, kde jsou zpožděné dodávky materiálů, se spíše hodí pravidla, která upřednostňují aktivity s menšími materiálovými nároky. Instance, kde je několik aktivit s velmi dlouhou dobou zpracování, byly lépe řešené pomocí pravidel, které vybraly dříve aktivity s nižší hodnotou p_j. Srovnání pravidel je ukázáno v kapitole experimentů. Ukázka několika řazení získaných postupem výše (mimo posledního) pro příklad z kapitoly 2:

- 1 2 3 4 5 6
- 3 4 1 5 6 2
- 1 2 5 3 4 6 - EDD pravidlo
- 2 1 5 3 4 6 - EDD-Ma pravidlo
- ~~3 4 6 2 1 5~~ - nevyhovující

4.6 Ohodnocení jedince

Ohodnocení spočívá v transformaci jedince na nějakou číselnou hodnotu, kterou dokážeme porovnat s ostatními a určit tak, jak dobrý jedinec je. V našem případě je kritériem již zmíněná *total tardiness* funkce, jejíž hodnotu použijeme.

Abychom tuto hodnotu získali z AL, je nutné k němu vygenerovat rozvrh, ze kterého se již *total tardiness* dokáže snadno získat. Tuto práci dělají generační schémata (SGS), konkretné jedno upravené pro RCPSP-Cu.

4.6.1 Seriové SGS pro RCPSP-Cu

Seriové rozvrhovací schéma ve své základní verzi představené v sekci 2.6.2 kontroluje precedence relace a kapacity klasických zdrojů. Pro RCPSP-Cu musí být mírně modifikováno, aby bralo v potaz i materiálové zdroje.

Aktivity jsou brány jedna po druhé z AL a vkládány do rozvrhu v nejdřívějším možném čase EFT (*earliest feasible time*). Ten musí vyhovovat precedence vztahům a kapacitním omezením obyčejných i kumulativních zdrojů. EFT je získáno tak, že se určí časy, kde každý splňuje jedno omezení, a vezme se jejich maximum. Nejprve čas, který splňuje precedence vztah EPT. To je čas, kdy jsou všichni předchůdci již dokončeni. Je to maximum z časů dokončení všech předchůdců, který se pro aktivitu j vypočte jako $F_j = S_j + p_j$ (start čas v rozvrhu plus čas zpracování).

Následuje zjištění času, kdy je dostatek zdrojů pro rozvržení aktivity - ERT (*earliest resource time*). S tím souvisí získání časové dostupnosti každého zdroje. U obnovitelných zdrojů je dostupnost konstantní, u kumulativních s časem rostoucí. Tím, že se nějaká aktivita vloží do rozvrhu v čase t, je nutné snížit hodnoty dostupnosti zdrojů. U požadavku aktivity na obnovitelný zdroj se hodnota sníží v každé periodě, kdy aktivita probíhá, tzn. na intervalu $\langle t, p_j \rangle$. U požadavku na kumulativní zdroje se hodnota jejich dostupnosti redukuje až do konce rozvrhu (do horní meze T), tzn. $\langle t, T \rangle$.

Takto průběžně aktualizované dostupnosti zaručují, že pokud chceme rozvrhnout aktivitu v čase t, tak se stačí podívat, jestli zbývající kapacita v časovém profilu pro čas t je větší nebo rovna jeho požadavku. Pokud není, postupně navyšujeme čas do té doby, než se nějaké zdroje uvolní.

Pseudokód 4.2 Tvorba rozvrhu z AL pomocí SGS

Input: AL - seznam pořadí aktivit

1: **for all** $j \in$ AL **do**
2: $EPT_j \leftarrow \max(S_i + p_i | i \in P_j)$ # P_j - předchůdci aktivity j
3: $ERT_j \leftarrow$ získej ERT_j # čas, kdy je dostatek zdrojů pro vykonání aktivity j
4: $EFT_j \leftarrow \max(EPT_j, ERT_j)$
5: $S_j \leftarrow EFT_j$
6: uprav dostupnosti zdrojů
7: **end for**

Takto se ohodnocuje každý jedinec populace, protože v závislosti na těchto hodnotách se následně generují další generace.

4.7 Tvorba nové generace

Po ohodnocení jedinců dochází k tvorbě nové populace, vybírají se elitní jedinci (*fitness* hodnota jedince je nejlepší) a ti se ponechávají do další generace beze změny. U zbylých dochází k párování a následnému křížení, mutaci nebo úplnému nahrazení.

Při aplikaci genetických operátorů musejí být operátory uzpůsobeny pro reprezentaci pomocí AL, která neumožňuje duplikaci aktivit. Stejně tak jako při tvorbě AL počátečních jedinců, i zde se musí zajistit, aby vzniklý AL měl korektní pořadí aktivit, které splňuje precedence vztahy.

4.7.1 Výběr elitních jedinců (Elite Selection)

U tohoto operátoru se specifika spjatá s AL reprezentací neprojevují. Tento operátor pouze vezme určitý počet nejlépe vyhodnocených jedinců a zkopíruje je do další generace tak jak jsou.

Množství jedinců, které se bere jako elitní, je určeno počtem procent z celkové velikosti populace a udává se předem jako parametr genetického algoritmu. Kupříkladu hodnota 5% s velikostí populace 100 znamená, že do další populace přejde 5 nejlepších jedinců bez sebemenší změny. To znamená, že následující generace nebude horší než předchozí. S procentuální hodnotou se to však nesmí přehnat, vyšší hodnota by mohla vést k předčasné konvergenci populace do lokálního minima [16].

4.7.2 Selekce rodičů (Mating)

Proces selekce se provádí pro získání 2 jedinců, kteří se spolu následně budou křížit. V algoritmu je použita strategie rulety (*roulette wheel*). Kde pravděpodobnost výběru jedince závisí na jeho *fitness* hodnotě - čím vyšší, tím i pravděpodobnost selekce stoupá. Párování je prováděno postupně a v každé iteraci se vždy náhodně vybírají zbylí jedinci v populaci. Proces se opakuje, dokud nejsou všichni jedinci z populace spárováni.

4.7.3 Křížení (Crossover)

Tento operátor je použit pro kombinaci existujících jedinců s jinými, a tím se zajišťuje genetická diverzita. Opět je stanovena nějaká pravděpodobnost křížení P_{cross}. To je hodnota mezi 0 a 1, která odpovídá pravděpodobnosti vzniku potomků z křížení jedinců. Hodnota rovna 1 znamená, že všechny vybrané dvojice budou zkříženy a vytvoří potomky do nové generace. Hodnota 0.85 znamená, že 85% dvojic bude zkříženo a zbytek párů projde do nové generace beze změny. Proces je znázorněn v pseudokódu 4.3.

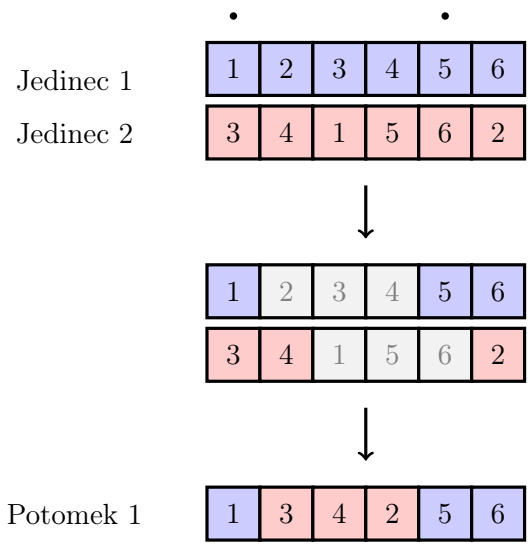
Při křížení jsou vzaty dva jedinci (otec a matka) a z nich vznikají 2 potomci (syn a dcera), používá se tzv. dvoubodové křížení (*two-point crossover*). Jsou vybrány 2 čísla k_1, k_2 ($k_1 < k_2$) menší než je počet aktivit. AL pro syna je vytvořen z prvků otcova AL - hodnoty na prvcích $\langle 0, k_1 \rangle$ a $\langle k_2, n \rangle$ - a zbytek hodnot $\langle k_1 + 1, k_2 - 1 \rangle$ je doplněn z matčina listu zachovávající jeho relativní řazení. Tím se zachovají precedence vztahy mezi aktivitami. Dcera je vytvořena analogicky. Krajní intervaly se vezmou z matky a vnitřní interval z otce.

Pseudokód 4.3 Genetické křížení

Input: G_1 - aktuální generace populace
Input: G_2 - nová generace

```
while G2 < G1 do                    # Dokud je velikost nové generace menší než aktuální
    rodiče ← selekce(G1)
    if rand() < P_cross then
        potomci ← křížení(rodiče)
        přidej potomky do G2
    else
        přidej rodiče do G2
    end if
end while
```

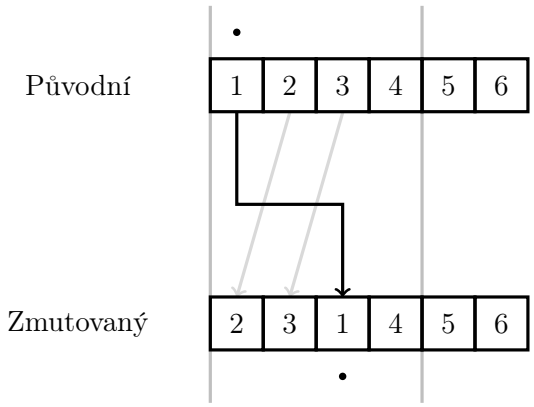
Na obrázku 4.3 jsou 2 náhodní jedinci (rodiče) $1, 2, 3, 4, 5, 6$ a $3, 4, 1, 5, 6, 2$. Indexy pro křížení jsou 0 a 4. Okrajové části potomka jsou jasné, vzniknou zkopírováním hodnot z jedince 1. Prostřední část, indexy $\langle 1, 3 \rangle$, se bere z jedince 2 tak, aby ve výsledném seznamu potomka byla každá aktivita zastoupena vždy jen jednou. Jedinec 2 se tedy postupně prochází a vkládá takové aktivity, které ještě nejsou obsaženy ve vzniknuvším potomkovi. Na obrázku jsou nejdříve zakresleny původní jedinci (otec a matka). V prostředním mezikroku jsou zvýrazněny geny, které se budou v potomkovi používat. Z modrého otce se vezmou krajní intervaly a z červené matky se vezme zbytek genů (při zachovaném pořadí!), které ještě nejsou obsaženy v krajních intervalech otce. Poslední krok je výsledný potomek.



Obrázek 4.3: Křížení jedinců

4.7.4 Mutace

Mutace je operátor, který je prováděn až na nové generaci populace, k jeho provedení stačí samotný jedinec. I zde musí být specifikován parametr v podobě pravděpodobnosti (P_{mut}). Postupně se prochází neelitní jedinci v nové generaci a se zadanou pravděpodobností P_{mut} jsou vybráni k mutaci. Každá aktivita (gen) z mutovaného jedince je vybrána s tou samou pravděpodobností P_{mut} k mutaci. Pokud je vybrána, pak dojde k změně jejího pořadí v AL. Nové pořadí musí být mezi posledním z předchůdců a nejbližším z následníků. To ponechá řazení v AL korektní. Náhodně se vybere místo mezi těmito dvěma popsanými body a tam se aktivita vloží. Zbytek aktivit se posouvá, aby vyplnil prázdné místo.



Obrázek 4.4: Mutace jedince

Obrázek 4.4 popisuje mutaci jednoho vybraného genu v jedinci. V tomto případě se jedná o gen na 1. místě s hodnotou 1 (označeno tečkou nahoře). Horní jedinec v obrázku je původní. Vstupuje do mutace a naznačené rozmezí vytyčuje, kam se mutovaný gen může vložit, aniž by došlo k porušení precedence relací. Vztahy mezi aktivitami jsou stejné jako z příkladu v druhé kapitole - $1 \rightarrow 5 \rightarrow 6, 3 \rightarrow 4, 3 \rightarrow 6$.

Aktivita s číslem 1 nemá žádného předchůdce, takže může být vložena od začátku seznamu. Naopak má dva následovníky - aktivitu č. 5 a 6. To značí, že může být vložena maximálně na 4. místo v seznamu. Z tohoto rozmezí se vybere náhodné číslo (např. 3) a na toto místo se vloží (gen, který zde byl původně, se posouvá vlevo a posouvá další geny).

V dolním jedinci už je výsledný zmutovaný jedinec, ve kterém je gen s hodnotou 1 na nové pozici.

4.8 Ukončující podmínka

Takto vznikla nová generace populace. Algoritmus neustále běží a postupně vyvíjí nové a nové generace. Poslední věcí je jeho ukončení. V algoritmu je zanesena ukončující podmínka, která se kontroluje po každé nově vytvořené generaci populace.

Jednou z podmínek je typicky časový limit běhu, nebo maximální počet vzniklých generací. V algoritmu je využívána kombinace těchto dvou, kdy buď nastane konec kvůli vyčerpání časového limitu, nebo se algoritmus dostane do svého maximálního počtu generací.

4.9 List scheduling pro RCPSP-Cu

Posledním implementovaným algoritmem je upravená heuristika *List scheduling* (LS) pro RCPSP-Cu. V podstatě se jedná o heuristiku, kterou se vyhodnocuje v předešlém genetickém algoritmu. LS je ve svém základu SGS (upravené pro RCPSP-Cu), kterému jsou předány seřazené aktivity podle nějakého prioritního pravidla. Rozdíl je v tom, že GA používá List scheduling opakovaně na každého jedince, zatímco LS jako takový běží jen jednou s daným pravidlem. To znamená, že GA nikdy nebude horší než LS.

4.10 Zařezávání vyhodnocování

Pro navýšení rychlosti genetického algoritmu bylo navrženo zrychlující vylepšení. Při vyhodnocování jedince pomocí SGS se průběžně v každé iteraci kontroluje *fitness* hodnota jedince (*total tardiness*). Dále je udržováno doposud nejlepší nalezené řešení ze všech generací. Pokud aktuální *fitness* hodnota v průběhu generování rozvrhu je vyšší než nejlepší nalezená, generování se může zaříznout a přejít k vyhodnocování dalšího jedince. Tento jedinec rozhodně nebude lepší než doposud nalezený nejlepší jedinec, proto si zaříznutí lze dovolit.

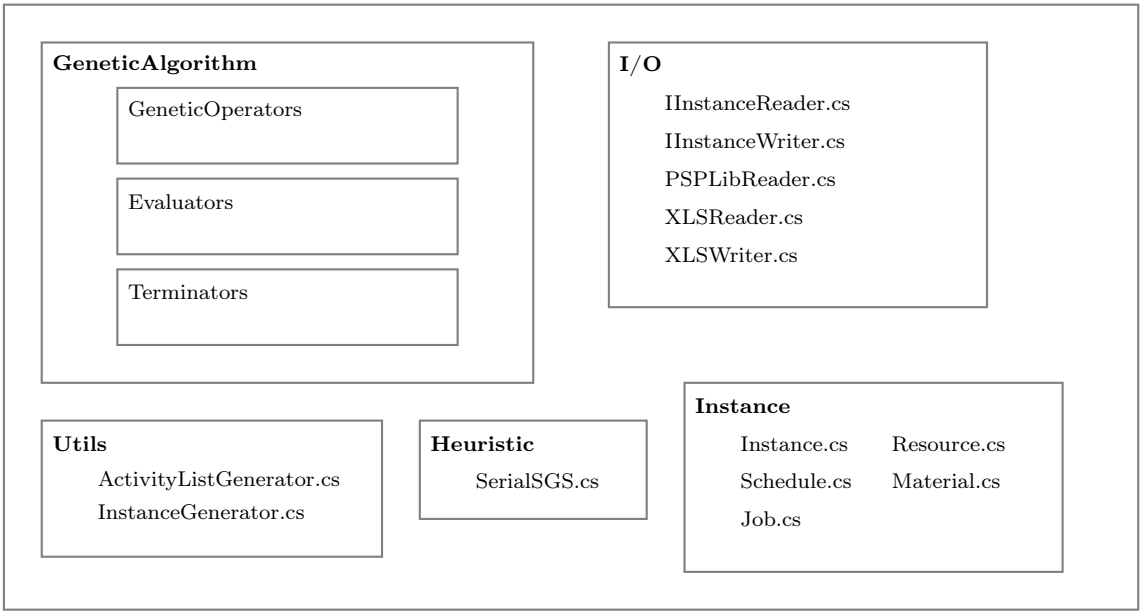
Kapitola 5

Implementace

Implementace GA je provedena v jazyce C# jako konzolová aplikace založená na .NET platformě. Jejím hlavním úkolem je načtení instance ze souboru, její vyřešení a následný zápis řešení zpět do souboru. Princip GA je obecně znám, a tak existují hotové knihovny, které již obsahují základní kostru algoritmu a mechanismy pro spolupráci jeho komponent. Na programátorovi je dosadit komponentám funkčnost. Jako kostra pro GA je použito existující řešení v podobě GAF [3] knihovny. Pro import a export z XLS formátu je použita knihovna EPPlus [2].

Aplikace je rozdělena na několik modulů, kde každý má svou specifickou funkčnost. Modulem je například genetický algoritmus. Ten obsahuje třídy, které spolupracují s GAF knihovnou. Jsou tam evaluátory jedinců, terminační kritéria a genetické operátory. Dalším modulem je I/O (input/output), který obsahuje třídy pro čtení a zápis instancí z a do různých formátů. Zde je možnost rozšíření o další formáty jednoduchou implementací poskytovaného rozhraní.

Kompletní struktura je na obrázku 5.1.



Obrázek 5.1: Struktura implementace

5.1 Kostra GA

V kódu 5.1 je vidět kostra GA. Nejprve je vytvořena populace o velikosti 50 jedinců. Postup pro vytvoření prvotní populace je skryt v metodě *generateInitPopulation*(), ale postup je stejný jako je popsán v sekci 4.5. Následuje vytvoření genetických operátorů (řádky 6 - 8). Elitní je použit přímo z knihovny, protože není nijak specifický. Ostatní dva (křížení a mutace) jsou již vlastní, a byly popsány dříve. Poté je vytvořen evaluátor jedinců. Tomu je předáno SGS spolu s instancí. Do tohoto objektu přichází každý z jedinců k ohodnocení. Jedinec ve formě AL určuje pořadí aktivit pro SGS, a ten vrací výsledek ohodnocení v podobě *total tardiness*. Dále je vytvořena terminační podmínka, která v tomto případě kontroluje čas běhu algoritmu a ukončí ho po uplynutí 50 vteřin.

Na řádku 14 se vytváří samotná instance genetického algoritmu, které se předá populace a evaluační metoda (ta je brána z objektu evaluátoru). Přidají se do něj genetické operátory (pozor, záleží na pořadí!) a algoritmus se spouští s předanou terminační podmínkou. GA má několik akcí, které lze odposlouchávat. Umí vyvolat akci při vytvoření nové generace a při ukončení běhu. Na ty se mohou navěsit posluchače ve formě funkcí (v tomto případě jsou navěšeny funkce s názvy *ga_OnGenerationComplete* a *ga_OnRunComplete*).

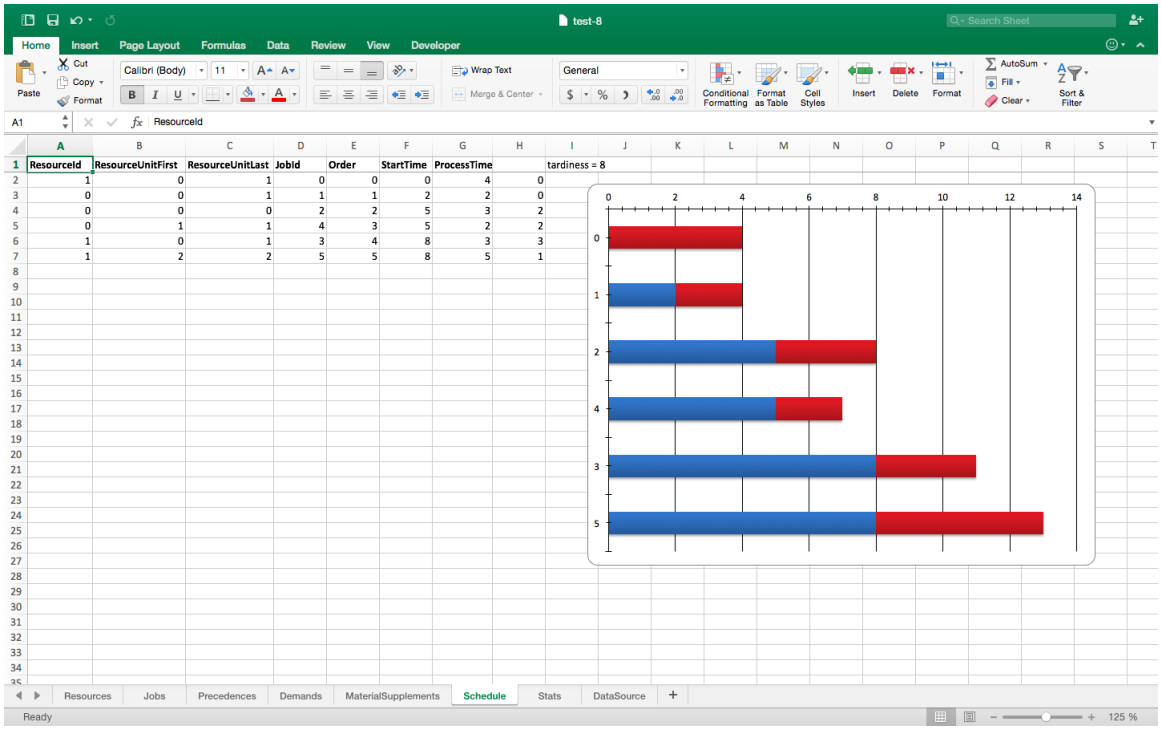
```
1  // populace
2  var population = new Population(50);
3  population.Solutions.AddRange(generateInitPopulation(50));
4
5  // geneticke operatory
6  var eliteOperator = new Elite(5);
7  var crossoverOperator = new RCPSPCrossover(0.7);
8  var mutationOperator = new RCPSPMutate(0.05, instance);
9  // evaluator
10 var evaluator = new ALFitnessEvaluator(new SGS(instance));
11 // terminacni podminka
12 var stopCondition = new ElapsedTimeCondition(50000);
13 // geneticky algoritmus
14 var ga = new GeneticAlgorithm(population, evaluator.calculateFitness);
15
16 ga.Operators.Add(elite);
17 ga.Operators.Add(crossover);
18 ga.Operators.Add(mutation);
19
20 // akce
21 ga.OnGenerationComplete += ga_OnGenerationComplete;
22 ga.OnRunComplete += ga_OnRunComplete;
23
24 // spusteni algoritmu
25 ga.Run(stopCondition.isSatisfied);
```

Kód 5.1: Kostra genetického algoritmu

5.2 XLS import/export

Struktura XLS souboru[1] (na obrázku 5.2 je vidět list s výsledným rozvrhem) obsahuje celkem 7 listů (z toho poslední 2 jsou pro výsledek) a je následovná:

- **Resources** - Obsahuje informace o zdrojích, jejich počet, typ a u obnovitelných zdrojů také kapacitu.
- **Jobs** - List obsahuje informace o všech aktivitách, jsou zde zádány jejich doby trvání a termíny dokončení.
- **Precedences** - Zde jsou obsaženy relace aktivit, co řádek to jedna relace mezi 2 aktivitami.
- **Demands** - Obsahuje informace o požadavcích aktivit na zdroje, každý řádek obsahuje číslo aktivity, zdroje a jeho požadované množství.
- **MaterialSupplements** - Obsahuje informace o zásobovacím plánu kumulativních zdrojů - časy a množství.
- **Schedule** - Obsahuje informace o výsledném rozvrhu, pořadí aktivit, startovací časy a využití jednotek zdrojů od aktivit. Obsahuje graf s rozvrženými aktivitami.
- **Stats** - Obsahuje statistiky využití každého zdroje vyobrazené v grafu.



Obrázek 5.2: XLS soubor: list s rozvrhem

[1] Celá šablona souboru je dostupná na přiloženém CD

Detailnější rozbor EPPlus knihovny pro práci s XLS s příklady pro zápis a načítání souborů spolu s tvorbou grafů je v příloze B.

Kapitola 6

Experimenty

V této kapitole je popsán postup provedených experimentů. Experimentální analýzou byly porovnány všechny naimplementované algoritmy - ILP metoda, Genetický algoritmus a upravená heuristika List scheduling.

Nejprve byla porovnána prioritní pravidla mezi sebou. Ty jsou používána v *List schedulingu* a v první populaci genetického algoritmu. Poté se vyhodnotí parametry GA jako např. velikost populace, pravděpodobnosti genetických operátorů a terminační kritéria. Následuje srovnání časové náročnosti na různě velkých instancích. Nakonec, když jsou u GA stanoveny nejvhodnější parametry a nastavení, je zpracováno srovnání výsledků s ILP metodou.

6.1 Testovací prostředí

Testovacím prostředím je běžný notebook s dvoujádrovým procesorem Intel(R) Core(TM) i5 @ 1.80GHz a 4GB RAM paměti. ILP model byl naprogramován v jazyce JAVA, ale využívá CPLEX solver, jehož zdrojové soubory byly k dispozici v binárním souboru ve verzi pro Mac OS X, takže spouštěn mohl být pouze pod tímto systémem. Zbylé dva algoritmy jsou implementovány v jazyce C# a spouštěny jsou na virtuálním stroji s Windows 8.1. Žádný algoritmus není tímto nijak (ne)zvýhodněn, protože implementace pod různými operačními systémy nejsou přímo srovnávány časově, ale jen hodnotami výsledků.

6.2 Testovací data

K experimentům je nutné obstarat sady nějakých vstupních dat, které budou algoritmy zpracovávat. Požadavkem ze zadání bylo vytvořit instance s přibližným počtem aktivit 30, 100, 300 a 1000. Část instancí byla použita ze známé knihovny PSPLIB [4, 20], která obsahuje vstupní data pro různé druhy RCPSP. Ta ovšem obsahuje pouze malé instance (max. 120 aktivit v instanci), proto její data byla použita jen pro malé instance. K zbývajícím datasetům byl vytvořen generátor instancí.

6.2.1 PSPLIB

Knihovna PSPLIb obsahuje instance pro více typů RCPSP, jedním z nich je také MRCPSP. Jedná se o *Multi-mode* RCPSP, které se od klasického RCPSP liší v módech aktivit. Každá aktivita má několik módů (verzí) a v jednom z nich musí být rozvržena. Verze se liší v době zpracování aktivit a v jejich požadavcích na zdroje [15]. Tyto instance obsahovaly 2 typy obnovitelných zdrojů s kapacitami, 2 typy neobnovitelných zdrojů s kapacitami, aktivity v 1 až 3 verzích spolu s jejich časy zpracování a vzájemnými vztahy.

Pro experimenty byly využity instance právě z množiny dat pro MRCPSP, které se nejvíce blíží studovanému problému. Převod instancí z MRCPSP na RCPSP-Cu byl následovný:

- Aktivity se vyskytují ve více módech, toto bylo potlačeno a byl vybrán vždy první mód aktivity. Tím se museli přepočítat celkové požadavky na zdroje (protože by mohlo dojít k nedostatku kapacit).
- Z neobnovitelných zdrojů se stanou kumulativní. To znamená, že požadavky aktivit na neobnovitelné zdroje jsou nyní považovány za požadavky na zdroje kumulativní. Jediné co chybí je zásobovací plán, ten bude dogenerován (o tom více v následující sekci).
- Hodnoty *due date* byly taktéž dogenerovány.

6.2.2 Generování dat

Pro větší instance byl naprogramován vlastní generátor se základními parametry generování. Na vstupu jsou předány požadavky kladené na výslednou instanci, a ta se následně vygeneruje.

Průběh generování Na vstupu jsou předány parametry například na počet aktivit, počet obnovitelných i kumulativních zdrojů, maximální *process* čas, či minimální a maximální počet předchůdců aktivity.

Podle těchto parametrů začne generování. Vygeneruje se zadaný počet obnovitelných zdrojů, kde kapacita je náhodné kladné číslo v určitém rozsahu (např. 1 - 20). Poté se mohou začít generovat aktivity.

Aktivity Každé z nich je přideleno postupně inkrementované ID. Doba trvání aktivity je náhodné číslo mezi 1 a maximálním limitem z parametru (např. 1 - 15). Poté jsou generováni předchůdci aktivity. Počet předchůdců je opět specifikován parametrem. Zde je generováno 0 až 2 předchůdců ke každé aktivitě, tzn. že instance nemusí být jedna celistvá síť aktivit, ale může být rozdělena do více grafů. Každý předchůdce je náhodně zvolená aktivita s ID nižším než aktivita samotná. Tím je zaručeno, že v grafu instance nemůže dojít k cyklům (díky topologickému uspořádání). Požadavky na obnovitelné zdroje jsou zvoleny náhodné do velikosti kapacity zdroje. Požadavky na kumulativní zdroje jsou náhodná čísla od 1 do 10. Po tvorbě všech aktivit jsou z několika z nich vytvořeny „dlouhé“ aktivity - doba trvání je ztrojnásobena. Tím je zajištěna různorodost aktivit.

Generování *due date* Protože hodnota *due date* přímo ovlivňuje výsledek celého rozvrhu, tak jeji generování bylo největším problémem. Buď byly hodnoty moc volné - *tardiness* rozvrhů bylo nulové, nebo naopak moc velké (u velkých instancí dosahovalo *tardiness* k stovkám tisíc). Nakonec byl navržen postup z algoritmu 6.1, který generoval nejpřijatelnější hodnoty. Každá aktivita má určitý offset - nejvyšší *duedate* ze svých předchůdců. Pokud předchůdce nemá, tak své ID. Poté se generuje náhodný koeficient, který je logaritmicky úměrný počtu aktivit. A z těchto dvou hodnot se ještě za pomoci doby trvání získá náhodný *duedate*, který je „tak akorát“.

Pseudokód 6.1 Generování *duedate*

Input: J - seřazená množina aktivit podle ID

1: **for** $j \in J$ **do**
2: offset = j
3: offset = $\max(d_i \mid i \in P_j)$
4: coef = $\mathrm{rand}(1, \log n)$
5: $d_j = \mathrm{offset} \cdot \mathrm{coef} + \mathrm{rand}(p_j,\ 2\mathrm{coef} \cdot p_j)$
6: **end for**

Zásobovací plány Posledním bodem jsou zásobovací plány kumulativních zdrojů. Počet dodávek je náhodné číslo, jehož horní mez jsou 2% z počtu aktivit + 2 (přičtení čísla je kvůli minimálnímu počtu dodávek) a dolní mez je polovina této hodnoty. Pro 30 aktivit je to $\langle 1, 2\rangle$ dodávek, pro 100 $\langle 2, 4\rangle$. Časy dodávek jsou rovnoměrně rozprostřeny do 1. třetiny odhadu času (suma dob trvání všech aktivit).Průměrný interval dodávek je třetina časového odhadu vydělená počtem dodávek $avg = (time\ /\ 3)\ /\ count$. Násobky tohoto intervalu jsou pak časy dodávek. Výpočet časů je ve výpisu 6.1. Instance, kde je časový odhad 300 a počet dodávek 4, bude mít zásobovací plán v časech 0, 25, 50, 75. Intervaly jsou pevné, v reálném světe tomu tak také bývá.

```
int maxMomentsCount = (numOfJobs / 50) + 2;
count = Random.Next(maxMomentsCount/2, maxMomentsCount);
averageTime = (time/3) / count;
int[] times = new int[count];
for (int j = 0; j < count; j++)
{
    times[j] = j * averageTime;
}
```

Kód 6.1: Generování zásobovacích časů

K určení množství doplňovaných materiálů v jednotlivých dodávkách je nutná znalost celkového součtu požadavků od všech aktivit. Celkový součet značí minimální množství materiálů, které je potřeba dodat, aby instance byla rozvrhnutelná. Je stanoveno průměrné množství (average) materiálu na jednu dodávku jako celkový součet požadavků dělený počtem dodávek, plus je zde náhodný přídavek/úbytek. Množství pro dodávku je tedy vypočteno jako

$$\mathrm{amount} = \mathrm{average} + \mathrm{rand}\Big(-\frac{\mathrm{average}}{2}, +\frac{\mathrm{average}}{2} \Big).$$

Může se stát, že bude u dodávek menší množství materiálu než průměrné (záporný přídavek) a celkové množství z dodávek by bylo menší než celkový součet z požadavků. V tom případě se do první dodávky přidá množství, které již pokryje celkové požadavky.

6.3 Experimenty

Před samotným *benchmarkem* bylo vygenerováno přesně 50 testovacích instancí z každé sady (J30, J100, J300, J1000)[1]. Instance z nejmenšího setu byly generovány z PSPLIB instancí a zbytek z generátoru. Uloženy byly v jednotném formátu XLS, ze kterého pak byly postupně načítány k testům.

6.3.1 Prioritní pravidla

Prvním srovnávacím testem je srovnání prioritních pravidel. Ty jsou aplikovány jedno po druhém do SGS a kritériem je co nejmenší *tardiness* vytvořeného rozvrhu. SGS s pravidly bylo spuštěno na každé instanci z každé sady. Jako výchozí bod je bráno známé prioritní pravidlo EDD, které bylo zvoleno, protože je vhodné pro kriteriální funkci *tardiness*.

Testováno bylo celkem 10 pravidel, přičemž do přehledu byly vybrány pouze 4, které byly použity i v GA. Pravidla byla v třech kategoriích a každá kategorie měla jiný základ. První měla základ pouze EDD pravidlo, druhá měla navíc ještě hodnotu *EST* a třetí počítala také s dobou vykonání aktivit. Každá kategorie měla 3 pravidla, kde se dále k základu různě počítalo s požadavky na zdroje:

- **Ma** - započtení součtu požadavků na materiály
- **Re** - započtení součtu požadavků na obnovitelné zdroje
- **ReMa** - započtení součtu požadavků na jakýkoliv typ zdroje

V tabulce 6.1 je ukázáno srovnání pro každou sadu, výsledky jsou v rámci celé sady zprůměrovány, čím menší číslo, tím lepší. Takto byly vyhodnoceny všechny sady instancí, následně byly tyto hodnoty sloučeny, aby daly celkový výsledek pro všechny sady. Tím byla vyhodnocena kvalita prioritních pravidel.

Pravidla, která nevycházejí z EDD nejsou tak kvalitní, proto se v přehledu ani neobjevují. Prvním pravidlem v tabulce je **EDD** a jak již bylo zmíněno, je to jakýsi referenční bod, se kterým se ostatní pravidla porovnávají. Druhé pravidlo **EDD-MA** má nejlepší výsledky ze své kategorie - proto bylo zařazeno. **EDD-ESTReMa** má nejvyrovnanější pořadí ze všech a umístilo se vždy na prvních čtyřech místech z deseti testovaných pravidel. Poslední pravidlo **EDD-ESTPTReMa** má nejlepší výsledky na instancích J100 a J300, proto je také zařazeno do nejlepších pravidel. Všechna z vybraných pravidel dosahují lepších výsledků než EDD.

[1] Názvy Jxxx označují počet aktivit v instanci (J = Job)

	EDD	EDD-Ma	EDD-ESTReMa	EDD-ESTPTReMa
J30	42,64	43,04	42,88	43,34
pořadí	2	5	4	6
J100	1363,88	1368,46	1318,24	1291,8
pořadí	9	10	4	1
J300	4833,42	4822,5	4778,04	4760,12
pořadí	9	6	3	1
J1000	46072,62	45725,42	45660,9	45787,7
pořadí	10	5	2	6
pořadí	4	3	1	2

Tabulka 6.1: Srovnání prioritních pravidel

6.3.2 Parametry GA

U implementovaného genetického algoritmu existují čtyři různé parametry. Pravděpodobnost křížení a mutace, procento elitních jedinců a velikost populace. U každého parametru byla zadána množina hodnot a postupným zkoušením všech kombinací byly získány nejlepší hodnoty parametrů.

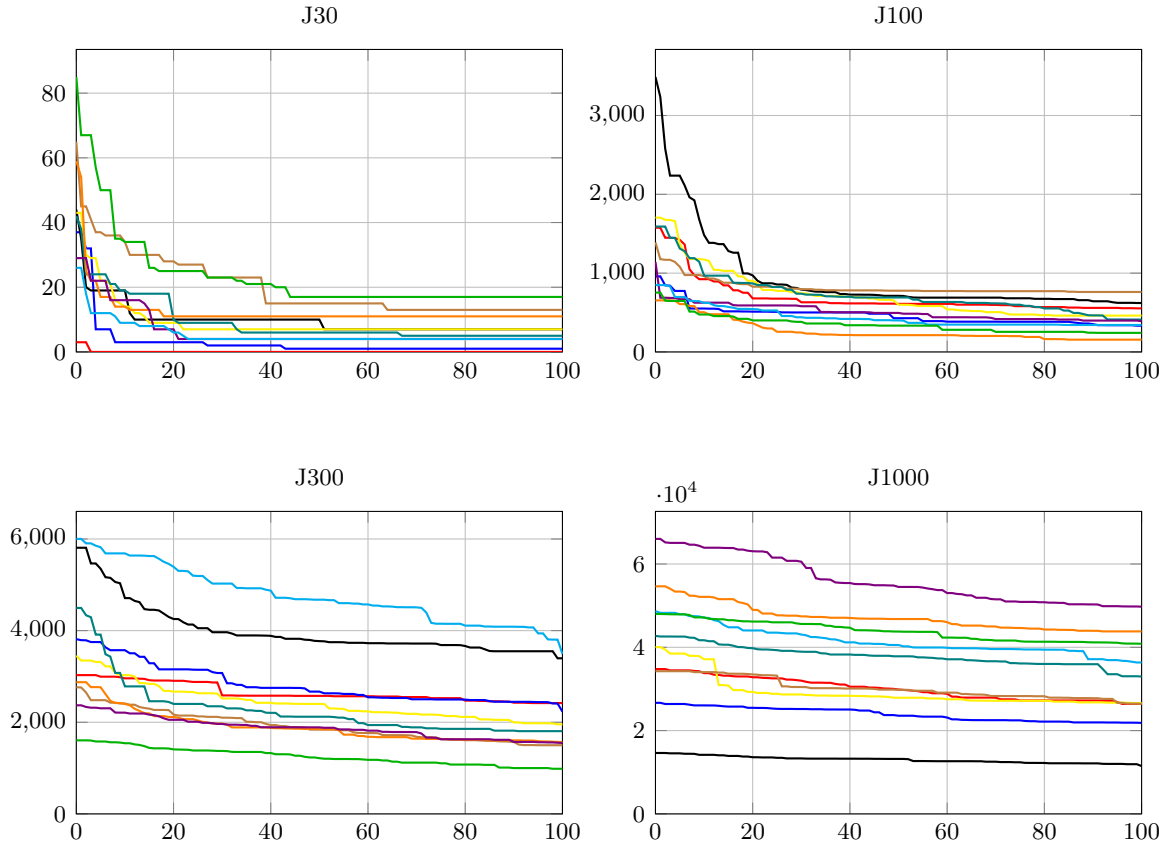
- Pcross - pravděpodobnost křížení $[0.7, 0.8, 0.85, 0.9]$
- P_{mut} - pravděpodobnost mutace $[0.01, 0.1, 0.2, 0.3]$
- P_{elite} - procento elitních jedinců $[2, 5, 10]$
- P_{pop} - velikost populace $[10, 20, 50]$

Parametry byly testovány na instancích ze sady J100 v 10 bězích. Cílem bylo získat co nejlepší *tardiness* hodnotu. Výsledky z 10 běhů se zprůměrovaly, a tím byl získán celkový výsledek. Nejlepší nastavení parametrů je v tabulce 6.2. Stanovené parametry jsou dále používány jako výchozí nastavení ve všech zbývajících experimentech.

Parametr	P_{pop}	P_{cross}	P_{mut}	P_{elite}
Hodnota	50	0.85	0.3	2%

Tabulka 6.2: Nastavení parametrů genetického algoritmu

Po zjištění výchozích hodnot pro parametry GA bylo také experimentováno s limitem počtu generací GA. Nejdříve byl maximální počet generací nastaven na 100 a algoritmus spuštěn s výchozími parametry. Na grafech v obrázku 6.1 jsou vidět výsledky pro 10 náhodných instancí z každé sady testovacích dat. Na ose X je vždy číslo generace a na svislé ose je *tardiness* nejlepšího jedince v odpovídající generaci populace. Jedinci se s rostoucí generací zlepšují a jejich *tardiness* se snižují. U menších instancí jsou zpočátku vidět výraznější skoky k lepšímu, ale postupem času se tato zlepšení čím dál více zmenšují. U větších instancí je zlepšení menší (graficky se zdá ještě menší, protože vertikální osa má větší rozsah hodnot než u menších instancí). Z těchto grafů lze vyčíst, že zlepšení se děje především v levé části grafu - prvních 50 generací. Proto byl zvolen limit genetického algoritmu na 40 generací. Okolo této mety již nedochází k výraznějšímu zlepšení rozvrhů, je to kompromis mezi časovou náročností algoritmu a kvalitou jeho výsledku.



Obrázek 6.1: Výsledky populace v čase (číslo generace)

6.3.3 Počáteční populace: náhodná vs. jedinci s prioritními pravidly

V následujícím experimentu je ukázáno, jaký dopad má obohacení 1. populace o nenáhodné jedince na výsledek algoritmu. Pokud jsou do počáteční populace přidáni jedinci

určení prioritním pravidlem, pak GA dosahuje mnohem lepších výsledků než při čistě náhodné populaci. V experimentu bylo vybráno 10 instancí ze sady J100 a 10 ze sady J300. Každá instance byla 10krát zpracována genetickým algoritmem, průměr hodnot je zanesen v tabulce 6.3. U sedmé instance (sady J100) je rozdíl obrovský. U této instance má v náhodně generované populaci nejlepší jedinec *fitness* hodnotu rovnu 2397 a po proběhnutí GA je ve 40. generaci vrácen výsledek 989. Při použití prioritních pravidel v první populaci je hodnota nejlepšího jedince rovna 654 a po čtyřiceti generacích je výsledek vylepšen až na hodnotu 197. U větších instancí (J300) jsou rozdíly ještě výraznější.

instance sady J100	i1	i2	i3	i4	i5	i6	i7	i8	i9	i10
1. gen. náhodná	4651	2298	3939	2488	3214	3149	2397	2678	1567	2418
1. gen. s pravidly	1576	959	3489	1704	1390	1591	654	1143	848	753
40. gen. náhodná	1996	1233	2039	1053	1593	1165	989	1203	835	1077
40. gen. s pravidly	593	330	930	717	804	658	197	329	515	353

instance sady J300	i1	i2	i3	i4	i5	i6	i7	i8	i9	i10
1. gen. náhodná	24986	21129	15006	20767	17665	14379	11358	13132	26840	14344
1. gen. s pravidly	3029	3814	5809	3452	2764	4495	2875	2370	6002	1605
40. gen. náhodná	15196	12811	10325	12241	10819	7119	6927	6980	15638	7944
40. gen. s pravidly	2650	2458	4382	2517	2027	2385	1899	1926	4719	1276

Tabulka 6.3: Vliv prioritních pravidel v počáteční populaci na výsledek

6.3.4 Srovnání algoritmů

Srovnány jsou 3 zde implementované algoritmy. Problém RCPSP-Cu s kriteriální funkcí *total tardiness* je natolik specifický, že jiná implementace k srovnání nebyla nalezena. U heuristiky *List Scheduling* je vybráno jako prioritní pravidlo EDD-ESTReMa, které v experimentech mělo nejlepší výsledky. ILP metoda měla nastaven časový limit na 15 minut.

Průměrné běhy algoritmů pro různé sady instancí jsou shrnuty v tabulce 6.4. ILP metoda byla schopna vrátit optimální výsledek v časovém limitu jen u nejmenší sady instancí, s tou si poradila do jedné vteřiny (průměrně cca 800ms). U instancí J300 a větších pátnáctiminutový limit nestačil.

	J30	J100	J300	J1000
GA	1.31s	3.56s	11.09s	38.08s
LS	0.001s	0.003s	0.04s	0.30s

Tabulka 6.4: Srovnání časů algoritmů

Vedle časového srovnání je také v tabulce 6.5 srovnání z hlediska kvality řešení. U menších instancí je výkonnost genetického algoritmu větší. Alespoň z hlediska procentuálního zlepšení řešení. U sady J100 je průměrné řešení LS rovno 1488, u GA je to 551. Z toho vyplývá,

že genetický algoritmus vrací průměrně o 63% lepší výsledek. U největší sady instancí je toto zlepšení menší, cca 14%. To je způsobeno velkým množstvím kombinací řazení úloh. U dvou menších sad lze srovnat GA i s ILP metodou. Hodnoty u ILP metody jsou výsledky algoritmu vrácené po 15 minutách výpočtu. U nejmenší sady J30 byly výsledky dosaženy v časovém limitu s velkou rezervou - čas výpočtu byl do jedné vteřiny. U sady J100 již nastaly problémy, u některých instancí časový limit stačil pro průchod kompletního prostoru řešení, ale u některých ne. Proto je v druhé tabulce (pro J100) přidán řádek (*gap*), který značí kolik procent prostoru řešení, ještě zbývá k prozkoumání. Hodnota *gap* 5 % značí, že nalezený výsledek je nejhůře 5 % od optima (5 % prostoru ještě nebylo prozkoumáno a je tedy možnost, že v této oblasti se nalezne lepší řešení).

J30	i1	i2	i3	i4	i5	i6	i7	i8	i9	i10	prům.
ILP	0	1	7	7	11	4	11	3	4	11	6
GA	0	3	7	9	15	8	11	6	4	17	8
LS	20	37	51	70	65	46	61	90	27	85	55

J100	i1	i2	i3	i4	i5	i6	i7	i8	i9	i10	prům.
ILP	72	79	208	7	163	86	31	28	18	1	72
gap	75.00%	31.95%	21.60%	0.00%	0.00%	86	0.00%	0.00%	0.00%	0.00%	
GA	561	238	1007	728	760	615	243	328	476	549	551
LS	1606	968	3550	1704	1390	2096	654	1143	848	923	1488

J300	i1	i2	i3	i4	i5	i6	i7	i8	i9	i10	prům.
ILP	-	-	-	-	-	-	-	-	-	-	-
GA	2732	2989	3842	2452	2125	2407	1860	1982	4730	1405	2652
LS	3098	3814	5849	3452	2786	4708	3207	2370	6052	1605	3694

J1000	i1	i2	i3	i4	i5	i6	i7	i8	i9	i10	prům.
ILP	-	-	-	-	-	-	-	-	-	-	-
GA	30689	23566	12904	32967	32559	36848	48940	58262	40524	44693	36195
LS	35424	29205	14634	41053	35114	43767	54639	66142	49583	49484	41905

Tabulka 6.5: Srovnání řešení všech implementovaných algoritmů

6.4 Vyhodnocení

Efektivitou algoritmu je brán poměr kvality výsledku řešení a časové náročnosti. Z výsledků experimentů lze vyčíst, že ILP metoda se může zavrhnout - už při J100 instancích má problémy s řešením v limitu času.

U zbylých dvou algoritmů již volba lepšího přístupu není tak jednoznačná. Genetický algoritmus sice poskytuje kvalitnější řešení, ale zaostává v době běhu za List schedulingem. Čas běhu sice není nikterak dlouhý (u největších instancí cca 40 sekund), ale pořád je velký rozdíl to oproti desetinám sekund u druhé heuristiky. Pak záleží na konkrétní aplikaci kde by měl být algoritmus využíván, zda je únosná mez čekat 40 sekund na výpočet algoritmu a mít lepší řešení, nebo dostat horší řešení, ale ihned. Lze si představit obě možnosti využití.

Kapitola 7

Závěr

Všechny stanovené cíle uvedené v úvodu byly splněny. Zadaná problematika byla rešeršním zpracováním dopodrobna prostudována, a ze získaných znalostí byly naimplementovány celkem 3 algoritmy. Očekávalo se využití heuristického řešení vzhledem k velikostem instancí, to se také nakonec potvrdilo v experimentech. Pro velké instance (se kterými se spíše počítá při využití algoritmu) se jako vhodnější přístup jeví použití heuristik, protože již při instancích s počtem úloh 100 přestává ILP metoda fungovat - její časová náročnost je příliš vysoká.

Hlavním zaměřením práce byl vývoj genetického algoritmu. GA dodává dostatečné kvalitní výsledky v přijatelném čase. Základ algoritmu byl použit z literatury a byl upraven pro zadanou kriteriální funkci. S tím souvisel návrh nových prioritních pravidel a snaha o zrychlení algoritmu včasným zařezáváním vyhodnocování.

Další možné pokračování práce je v zakomponování reálných časových jednotek do problému. Nyní jsou v algoritmu časové údaje brány jako celočíselné hodnoty. Dále pak celou aplikaci převézt z konzolové na webovou. Vstupní data by pak nemusela být načítána jen z XLS souborů, ale mohli by být zadávány z webové aplikace. To samé platí o výstupu algoritmu, rozvrhy by mohli být zobrazovány na webu ve formě grafů a tabulek.

Další bádání by se dalo zaměrit na optimalizaci algoritmu při zjednodušení požadavků na zdroje. Nyní byla brána v úvahu nejobecnější možná varianta problému, kdy aktivita může mít požadavky klidně na všechny typy zdrojů a materiálů. V praxi se ale dle mého názoru spíše vyskytují zadání, kdy každá aktivita má požadavek jen na jeden typ materiálu a například jen na jednu jednotku klasického zdroje. Timto omezením by se dali zjednodušit (nebo i zcela zanedbat) některé procedury ve stávajícím řešení.

Literatura

[1] IBM CPLEX Optimizer, . Dostupné z: http://www.ibm.com/software/integration/optimization/cplex-optimizer/.

[2] EPPlus-Create advanced Excel spreadsheets on the server, . Dostupné z: http://epplus.codeplex.com.

[3] Genetic Algorithm Framework for .Net 4.0, . Dostupné z: http://aiframeworks.net/gaf.

[4] PSPLIB - Project Scheduling Library, . Dostupné z: http://www.om-db.wi.tum.de/psplib/main.html.

[5] ARTIGUES, C. – DEMASSEY, S. – NéRON, E. *Resource-constrained project scheduling: Models, algorithms, extensions and applications*. ISTE, 2008.

[6] BŁAŻEWICZ, J. – LENSTRA, J. K. – RINNOOY KAN, A. H. G. Scheduling subject to resource constraints: Classification and complexity. *Discrete Applied Mathematics*. s. 11–24.

[7] BÖTTCHER, J. et al. Project Scheduling Under Partially Renewable Resource Constraints. *Management Science*. 1999, 45, 4, s. 543–559.

[8] BOYD, S. – VANDENBERGHE, L. *Convex Optimization*. Cambridge University Press, 2004.

[9] BOYSEN, N. – BOCK, S. – FLIEDNER, M. Scheduling of inventory releasing jobs to satisfy time-varying demand: an analysis of complexity. *Jena research papers in business and economics*. , 15, s. 185–198.

[10] BRUCKER, P. – KNUST, S. *Complex Scheduling*. GOR-Publications, 2012.

[11] BUKATA, L. – ŠŮCHA, P. – HANZÁLEK, Z. Solving the Resource Constrained Project Scheduling Problem Using the Parallel Tabu Search Designed for the CUDA Platform. *Journal of Parallel and Distributed Computing*. 2015, 77, s. 58–68.

[12] CARRERA, S. – RAMDANE-CHERIF, W. – PORTMANN, M.-C. Scheduling problems for logistic platforms with fixed staircase component arrivals and various deliveries hypotheses. *Applied Operational Research*. , 2, s. 517–528.

[13] CHALESHTARI, A. S. – SHADROKH, S. A Branch and Bound Algorithm for Resource Constrained Project Scheduling Problem subject to Cumulative Resources. *World Academy of Science*. 2012, 6.

[14] CHALESHTARI, A. S. – SHADROKH, S. Resource Constrained Project Scheduling Problem subject to Cumulative Resources. *International Journal of Mining, Metallurgy & Mechanical Engineering*. 2013, 1, 1.

[15] DEMEULEMEESTER, E. L. – HERROELEN, W. S. *PROJECT SCHEDULING A Research handbook*. International Series in Operations Research & Management Science, 2002.

[16] GONÇALVES, J. F. – RESENDE, M. G. C. – MENDES, J. J. M. A Biased Random-Key Genetic Algorithm with Forward-Backward Improvement for the Resource Constrained Project Scheduling Problem. *Journal of Heuristics*. 2011, 17, 5, s. 467–486.

[17] GRIGORIEV, A. – HOLTHUIJSEN, M. – KLUNDERT, J. Basic Scheduling Problems with Raw Material Constraints. *Wiley InterScience*. , 52, s. 527–535.

[18] HARTMANN, S. – BRISKORN, D. A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*. 2010, 1, 207, s. 1–14.

[19] KOLISCH, R. – HARTMANN, S. Heuristic algorithms for solving the resource-constrained project scheduling problem - Classification and computational analysis. In *Handbook on recent advances in project scheduling*.

[20] KOLISCH, R. – SPRECHER, A. PSPLIB - A project scheduling problem library. *European Journal of Operational Research*. 1996, 96, 1, s. 205–216.

[21] KONÉ, O. et al. Comparison of mixed integer linear programming models for the resource-constrained project scheduling problem with consumption and production of resources. *Flexible Services and Manufacturing Journal*. 2013.

[22] LABORIE, P. Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results. *Artificial Intelligence*. , 143, s. 151–188.

[23] LOVA, A. – TORMOS, P. – BARBER, F. Multimode resource constrained project scheduling: scheduling schemes, priority rules and mode selection rules. *Inteligencia Artificial*. 2006, 10, 30, s. 69–86.

[24] LUNER, P. Jemný úvod do genetických algoritmů.
`http://cgg.mff.cuni.cz/~pepca/prg022/luner.html`, stav z 12. 4. 2015.

[25] NEUMANN, K. – SCHWINDT, C. Project scheduling with inventory constraints. *Mathematical Methods of Operations Research*. s. 513–533.

[26] SCHUTTEN, M. List scheduling revisited. *Operations Research Letters*. , 18, s. 167–170.

[27] TAVAKKOLI-MOGHADDAM, R. – MEHDIZADEH, E. A new ILP model for identical parallel-machine scheduling with family setup times minimizing the total weighted flow time by a Genetic algorithm. *International Journal of Engineering*. 2007, 20, 2, s. 183–194.

[28] XIE, J. Polynomial algorithms for single machine scheduling problems with financial constraints. *Operations Research Letters*. , 31, s. 39–42.

[29] ČAPEK, R. – ŠŮCHA, P. – HANZÁLEK, Z. Production Scheduling with Alternative Process Plans. *European Journal of Operational Research*. 2012, 217, 2, s. 300–311.

Notace

n počet aktivit

$\mathbf{R}$ počet obnovitelných zdrojů

$\mathbf{CR}$ počet kumulativních zdrojů

$\mathbf{R}_k$ kapacita zdroje k

$\mathbf{CR}_{kt}$ kapacita kumulativního zdroje k v čase t

P_j množina předchůdců pro aktivitu j

p_j doba trvání aktivity j

d_j termín dokončení aktivity j

r_{jk} požadavek na zdroj k aktivity j

c_{jl} požadavek na kumulativní zdroj l aktivity j

T horní mez (odhad) času

T_j *tardiness* aktivity j

S_j čas startu aktivity j v rozvrhu

F_j čas dokončení aktivity j v rozvrhu

Seznam použitých zkratek

C#	C Sharp
AL	Activity List
EDD	Earliest Due Date
EPPlus	Excel Package Plus
EST	Earliest Start Time
GA	Genetic Algorithm
GAF	Genetic Algorithm Framework
HW	Hardware
ILP	Integer Linear Programming
JAR	Java Archive
LS	List Scheduling
LSTLFT	Latest Start Time - Latest Finish Time
RCPSP	Resource Constrained Project Scheduling Problem
RCPSP-Cu	RCPSP subject to cumulative resources
SA	Simulated Annealing
SGS	Schedule Generation Scheme
XLS	Microsoft Excel Spreadsheet

Příloha A

Instalační příručka

Javovský kód je klasický JAVA projekt, který navíc obsahuje závislosti na externí knihovny. Jednou z nich je CPLEX solver a druhou je Apache POI pro práci s XLS soubory. K oběma knihovnám je nutné obstarat JAR soubory a ty pak přilinkovat jako externí závislosti projektu. Buď přes parametr `java -classpath` nebo přes IDE. Spouštěcí příkaz pak vypadá podobně tomu následujícímu:

```
java -classpath "/path/poi-3.11/*;/path/poi-3.11/ooxml-lib/*;/path/cplex.jar;
/path/ILOG.CP.jar" cz.cvut.fel.ilp.RCPSPCU
```

K rozběhnutí C# projektu je nutné mít nainstalovanou .NET platformu s Visual Studiem. Zdrojové soubory (projekt) z CD stačí načíst ve Visual Studiu a balíčkovací manažer Nuget se postará o dočtení dodatečných knihoven. V souboru `Benchmark.cs` lze vyčíst použití genetického algoritmu. Při spuštění hlavní metody benchmarku se defaultně začnou počítat nejmenší sady instancí (J30).

Příloha B

EPPlus - ExcelPackage Plus

Knihovna EPPlus je C# knihovna pro pokročilou práci s XLS(X) soubory. Umí číst data z existujícího souboru, psát do nového souboru nebo do předpřípravené šablony, generovat grafy a na buňky v souboru aplikovat excelovské funkce. To vše v lehce použitelném a krásně čitelném kódu.

Hlavní třídy s kterými knihovna pracuje jsou:

- `ExcelPackage` - Třída zaobalující celý XLS soubor.
- `ExcelWorksheet` - Třída reprezentující list XLS souboru. Poskytuje přístup k buňkám. Přístup ke kreslení grafů a tvarů je ve vlastnosti `Drawings`
- `ExcelRange` - Třída reprezentující rozsah buněk v listu. Přistupovat k rozsahu buňel lze přes pohodlný zápis známý z excelu. Pro přístup k jedné buňce je navíc možné použít i zápis ve formě indexů dvourozměrného pole. Příklad:
  - `["A1"]` nebo `[1,1]` (levá horná buňka)
  - `["A1:B1"]` (rozsah - první dva sloupce na prvním řádku)

```
ExcelPackage xls = new ExcelPackage(new FileInfo("example.xlsx"));
ExcelWorksheet worksheet = xls.Workbook.Worksheets["Resources"];
worksheet.Cells[1,1].Value = 1234; // atd.

// graf
var barChart = worksheet.Drawings.AddChart("scheduleChart", eChartType
    .BarStacked) as ExcelBarChart;

ExcelRange axis = ExcelRange.GetAddress(1, 1, 10, 1);
barChart.Series.Add(axis, ExcelRange.GetAddress(1, 2, 10, 2));
barChart.Series.Add(axis, ExcelRange.GetAddress(1, 3, 10, 3));

barChart.SetSize(600, 500);
barChart.Title.Text = "Schedule chart";
barChart.Legend.Remove();
```

Kód B.1: EPPlus ukázka

Nevýhody Jediné nevýhody, které byly zpozorovány, byly při práci s grafy. První není tolik omezující, knihovna podporuje většinu typů grafů kromě těch z kategorií *Bubble*-, *Radar*-, *Stock*- a *Surface*. Druhou nevýhodou je nemožnost vlastního stylování grafů, tzn. že generovaný graf má defaultní vzhled

Více informací lze najít na domovské stránce projektu http://epplus.codeplex.com.

Příloha C

Obsah přiloženého CD

Přiložené CD obsahuje následující soubory a adresáře:

```
/
├── article ..............................  LaTeX zdrojové soubory tohoto dokumentu
├── src ..................................................................  zdrojové kódy
│   ├── csharp-ga-impl ...............................  implementace GA v jazyce C#
│   │   └── datasets ........................................  testovací sady instancí
│   └── java-ilp-impl ............................  implementace ILP v jazyce JAVA
└── thesis.pdf ...................................  PDF soubor tohoto dokumentu
```

	EDD	**Res**	**Mat**	**ResMat**	**ESTRes**	**ESTMat**	**ESTResMat**	**ESTPTRes**	**ESTPTMat**	**ESTPTResMat**
J30	42,64	43,84	43,04	43,92	42,74	42,24	42,88	43,82	43,4	43,34
pořadí	2	9	5	10	3	1	4	8	7	6
J100	1363,88	1328,4	1368,46	1330,18	1316,78	1324,36	1318,24	1294,56	1324,78	1291,8
pořadí	9	7	10	8	3	5	4	2	6	1
J300	4833,42	4828,52	4822,5	4829,84	4788,28	4836,14	4778,04	4769,14	4799,62	4760,12
pořadí	9	7	6	8	4	10	3	2	5	1
J1000	46072,62	45914,14	45725,42	45906,12	45725,32	45677,6	45660,9	45845,1	45603,28	45787,7
pořadí	10	9	5	8	4	3	2	7	1	6
pořadí	8	9	7	10	2	4	1	6	4	3