České vysoké učení technické v Praze
Fakulta elektrotechnická

Bakalářská práce

Optimální řízení nákladních výtahů

Vypracoval: Miloslav Stibor
Vedoucí práce: Ing. Michal Kutil

České vysoké učení technické v Praze, fakulta elektrotechnická

Katedra řídicí techniky Školní rok: 2005/2006

Z a d á n í b a k a l á ř s k é p r á c e

Student: Miloslav S t i b o r

Obor: Kybernetika a měření

Název tématu: Optimální řízení nákladních výtahů

Z á s a d y p r o v y p r a c o v á n í :

1. Seznamte se se základy teorie rozvrhování a se základními heuristikami pro rozvrhování na paralelních procesorech.
2. Popište úlohu optimalizace nákladních výtahů terminologií rozvrhování. Nalezněte algoritmus hledající řešení tohoto problému. Algoritmus implementujte v Matlabu s použitím funkcí Scheduling toolboxu.
3. Vytvořte uživatelské rozhraní umožňující použití algoritmu v praktickém provozu s ukázkovými případovými studiemi.

Seznam odborné literatury: Dodá vedoucí práce

Vedoucí bakalářské práce: Ing. Michal Kutil

Datum zadání bakalářské práce: září 2005

Termín odevzdání bakalářské práce: 26. 5. 2006

L.S.

Prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

Prof. Ing. Vladimír Kučera, DrSc.
děkan

V Praze, dne 14. 11. 2005

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) 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 ……. …….
podpis

Poděkování

Na tomto místě bych rád poděkoval vedoucímu mé práce Ing. Michalu Kutilovi, bez jehož pomoci, rad a připomínek by tato práce nemohla vzniknout. Můj vděk patří také všem, jenž mě v mé práci vytrvale podporovali.

Téma

Optimální řízení nákladních výtahů

Abstrakt

Tato práce popisuje optimalizaci řízení obecného počtu nákladních výtahů v průmyslovém provozu. Cílem je nalézt a implementovat vhodné heuristické algoritmy pro minimalizaci průměrné doby odbavení nákladu s ohledem na jeho prioritu.

K tomuto účelu je v prostředí *MATLAB* implementován základní kombinatorický algoritmus *List Scheduling* a na něm založené heuristiky EST a ECT takovým způsobem, aby rozšířily stávající *TORSCHE: Scheduling Toolbox*, vyvíjený na katedře řídicí techniky.

Vhodnost použití těchto algoritmů v praktickém provozu je zkoumána statistickým porovnáním jejich výsledků a časové náročnosti na odpovídajících příkladech. Praktické nasazení je pak demonstrováno pomocí grafického uživatelského rozhraní, rovněž implementovaného v prostředí *MATLAB*.

Theme

Cargo lift optimal control

Abstract

This work describes an optimization of the control of an inexplicit number of cargo lifts in the industry plant. The goal is to find and to implement a suitable heuristic algorithms intended for minimizing of an average time of clear-goods with reference to its priority.

For this purpose, the basic combinatorial algorithm *List Scheduling* and heuristics EST and ECT based on it are implemented in *MATLAB*. Algorithms upgrade existing *TORSCHE: Scheduling Toolbox*, which has been developed at the Department of Control Engineering.

The efficiency of using of those algorithms in practice is inspected by statistic comparison of their results and time complexity on suitable examples. Assignment is demonstrated by graphical user interface which is also implemented in *MATLAB*.

Obsah

Seznam obrázků

Seznam tabulek

Seznam *MATLAB* kódů

1. Úvod

V dnešní uspěchané době tvoří rčení: „*Čas jsou peníze.*“ jedno ze základních pravidel prosperujícího obchodu. Expediční náklady se stávají nezanedbatelnou součástí celkových nákladů na výrobu konečného produktu. Efektivní využití existujících prostředků, tvořících součást distribučního řetězce v jakémkoliv oboru, je velmi zajímavý problém, jehož optimalizací lze dosáhnout snížení režijních nákladů a urychlení distribuce.

Cílem mé práce bude navrhnout a realizovat řešení optimalizace provozu nákladních výtahů za pomoci existujících heuristických algoritmů. Kritériem optimalizace bude minimalizace průměrné doby odbavení nákladu s ohledem na dobu jeho exspirace. Dále provedu vzájemné porovnání těchto algoritmů, diskusi výsledků a realizaci grafického uživatelského rozhraní, umožňujícího demonstraci jejich využití v praktickém provozu.

V první části mé práce je nutné nejprve stručně vysvětlit základní pojmy z teorie rozvrhování, potřebné k pochopení problematiky. Dále je nezbytné vyjádřit zadanou optimalizační úlohu v přesné a jednoznačné terminologii rozvrhování.

Na základě této teoretické analýzy přistoupím k nalezení, popisu a implementaci vhodných heuristických algoritmů v prostředí *MATLAB* takovým způsobem, aby doplňovaly stávající *TORSCHE: Scheduling Toolbox*, vyvíjený na katedře řídicí techniky. Vzájemné statistické porovnání algoritmů bude provedeno na příkladech simulujících skutečný provoz se zaměřením na jejich časovou náročnost a vzdálenost výsledků od optimálních řešení, v závislosti na proměnné hustotě úloh ve výsledném rozvrhu.

Vzhledem k charakteristice problému, který spadá do třídy *NP-těžkých* úloh, nebude implementován algoritmus hledající optimální řešení, neboť jeho časová náročnost by byla vzhledem k nalezeným řešením v praktickém provozu nevyužitelná. Pro účely srovnání s výsledky suboptimálních algoritmů budou optimální řešení hledána pomocí nástroje *OPL Studio*.

Posledním bodem této práce bude návrh a realizace grafického uživatelského rozhraní umožňujícího přehledné zadávání, editaci a zobrazení jednotlivých úloh včetně jejich řešení. Toto rozhraní bude rovněž implementováno v prostředí *MATLAB*.

2. Teorie rozvrhování

Tato kapitola je stručným úvodem do teorie rozvrhování, který obsahuje základní informace potřebné pro pochopení dané problematiky. Podrobnější informace širšího rozsahu lze nalézt například v [Brucker] nebo [Pinedo].

2.1. Rozvrhování (*scheduling*)

Rozvrhováním v obecné rovině rozumíme přiřazení zdrojů úlohám v čase. Široké využití lze nalézt v informatice, stavebnictví, výrobní sféře, administrativě, ale i v energetice atd.

O statickém rozvrhování (*off line scheduling*) hovoříme tehdy, jsou-li všechny informace o množině úloh známy již před aplikací samotného algoritmu. Algoritmy této kategorie pracují ve dvou fázích. V první fázi jsou úlohy seřazeny do určitého pořadí a ve druhé pak postupně rozvrženy.

Naproti tomu o dynamickém rozvrhování (*on line scheduling*) hovoříme v případě, že algoritmus dostává úlohy průběžně a ve většině případů je nucen rozhodovat o umístění úlohy bez jakékoli znalosti budoucího vývoje situace. Tato chybějící znalost mu zabraňuje garantovat nalezení optimálního řešení. [Leung]

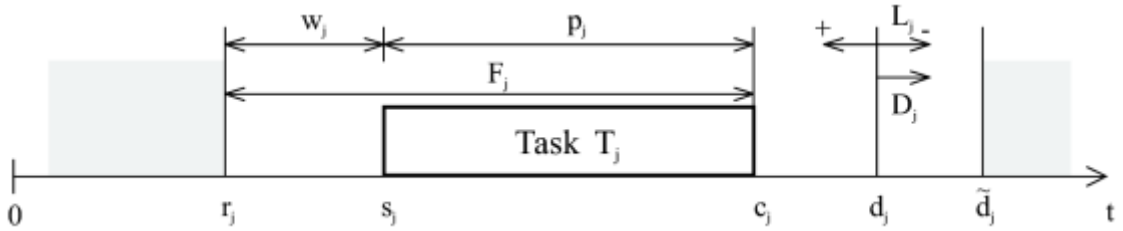
2.2. Základní vstupní data rozvrhování

- úlohy
- zdroje
  - procesory
  - ostatní
- daná omezení
- penalizační funkce

[RDU]

2.3. Úloha (*task*)

Úloha je základním prvkem teorie rozvrhování. Reprezentuje proces nebo diskrétní událost o určité délce trvání, jenž chceme zařadit do rozvrhu a může mít následující parametry [Błażewicz], [Scheduling]:



obr. 2.1: parametry úlohy

- $\mathbf{p_j}$ doba vykonávání (*processing time*)
- $\mathbf{c_j}$ okamžik dokončení (*completion time*)
- $\mathbf{s_j}$ okamžik započetí (*starting time*)
- $\mathbf{r_j}$ okamžik disponibility (*release time*)
- $\mathbf{w_j}$ prodleva (*waiting time*)[1]
- $\mathbf{F_j}$ reakční doba $F_j = c_j - r_j$ (*flow time/response time*)
- $\mathbf{\tilde{d}_j}$ poslední přípustný okamžik dokončení (*deadline*)
- $\mathbf{d_j}$ okamžik požadovaného dokončení (*duedate*)
- $\mathbf{L_j}$ zpoždění v záporném i kladném smyslu $L_j = c_j – dj$ (*latness*)
- $\mathbf{D_j}$ zpoždění pouze v záporném smyslu $D_j = \max\{c_j – d_j,0\}$ (*tardiness*)

2.4. Standardní notace

Vzhledem k různorodosti oborů, ve kterých lze teorii rozvrhování s úspěchem použít, je více než vhodné, vyjadřovat řešené problémy nějakým jednotným a univerzálním jazykem. Za tímto účelem definovali Graham a Błażewicz takzvanou standardní notaci $\alpha \mid \beta \mid \gamma$. Ta přehledným způsobem popisuje počet a typ procesorů, jednotlivá omezení problému a samotné kritérium optimalizace.

α - počet a typ procesorů

- **P** identické (*identic*) paralelní procesory s totožnou, konstantní rychlostí
- **Q** uniformní (*uniform*) paralelní procesory s rozdílnou, ale konstantní rychlostí
- **R** nesouvztažné (*unrelated*) paralelní procesory s proměnnou rychlostí, závislou na typu zpracovávané úlohy
- **O** dedikované procesory pro úlohy typu „Open shop“
- **F** dedikované procesory pro úlohy typu „Flow shop“
- **J** dedikované procesory pro úlohy typu „Job shop“

Poslední tři typy procesorů nebudeme blíže definovat, pro naši práci je nebudeme potřebovat. Jen zmíníme, že dedikovaný procesor je takový, který je určen jen pro jeden konkrétní typ úloh. V těchto případech se pak většinou pracuje s několika takovými procesory, z nichž každý je určen pro různý typ úloh slučujících se do tzv. zakázek *(Jobs)*.

Informace o počtu procesorů se ve standardní notaci uvádí jako číslo bezprostředně za označením typu procesorů. Tedy například „Q2“ znamená, že se jedná o dva uniformní paralelní procesory.

[1] V této práci, pokud nebude uvedeno jinak, budeme symbolem $\mathbf{w_j}$ označovat prioritu úlohy (*weight*).

β – omezení problému

pmtn	preempce, úloha může být přerušena a dokončena později
res	přídavné zdroje
prec	obecné precedenční závislosti (*precedence constraints*), tedy případ problému, kdy možnost započetí nějaké úlohy závisí na dokončení jiné
tree	precedenční závislosti stromové struktury
chain	precedenční závislosti řetězové struktury
$\mathbf{r_j}$	okamžik disponibility úlohy (*release time*)
$\mathbf{d_j}$	okamžik požadovaného dokončení úlohy (*duedate*)
$\mathbf{\tilde{d}_j}$	nejzazší přípustný okamžik dokončení úlohy (*deadline*)
no-wait	prodleva (*waiting time*) musí být u všech úloh rovna nule

γ – kritérium optimalizace

c_{max} - celková délka rozvrhu, $\sum c_j$ - suma okamžiků dokončení všech úloh, $\sum w_j c_j$ – suma vážených okamžiků dokončení úloh, L_{max} - minimalizace zpoždění, atd.

2.5. Výpočetní složitost algoritmů

Dříve než přistoupíme k nalezení algoritmů, které řeší náš problém rozvrhování na paralelní procesory, je vhodné zmínit se krátce o výpočetní složitosti algoritmů.

Řekněme, že úloha *U* má horní odhad složitosti *f(n)*, jestliže existuje algoritmus, který řeší *U* a má časovou složitost *O(f(n))*. Jinými slovy, horní odhad složitosti úlohy nám dává jakýkoli algoritmus, který danou úlohu řeší a jehož časovou složitost umíme odhadnout shora. [Mařík]

Dle časové složitosti lze algoritmy dělit na algoritmy s polynomiální a nepolynomiální výpočetní složitostí. Algoritmus s polynomiální časovou složitostí je takový algoritmus, jehož složitost lze popsat funkcí *O(p(k))*, kde *p* je polynom a *k* je velikost instance dané úlohy.

Algoritmy, jejichž časovou složitost nelze předešlou funkcí vyjádřit, jsou algoritmy s nepolynomiální složitostí. To znamená, že jejich časovou složitost nelze shora omezit polynomiální funkcí, jelikož roste například logaritmicky či exponenciálně. Při řešení úloh pomocí takových algoritmů pak dochází k nežádoucí „výpočetní explozi“, která znemožňuje použití algoritmu na větší instance úloh, pokud požadujeme výsledek v nějakém, pro nás přijatelném čase.

2.6. Třídy úloh

V předchozí podkapitole jsme hovořili o časové složitosti algoritmů, tedy zcela konkrétních postupů, řešících reálné úlohy. O složitosti však lze hovořit i v případě samotných úloh. Ty lze na základě podobných rysů rozdělit do tříd složitosti i přesto, že se navzájem liší ve způsobu jakým jsou zadány a ve formě v jaké je očekáván výsledek. Připomeňme tedy jen krátce možné třídy složitosti úloh [Mařík]:

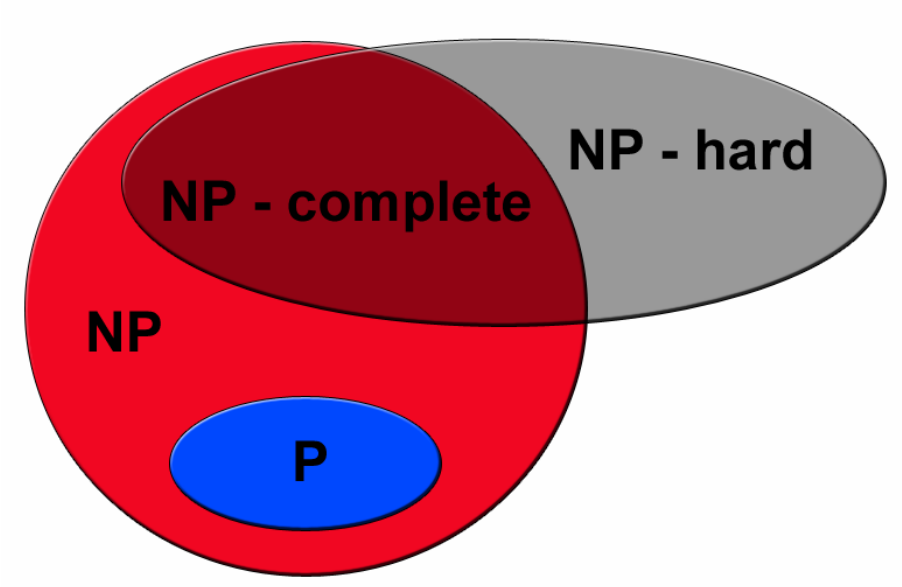
P (*polynomial*)
Třída *P* je třída všech rozhodovacích úloh $\mathcal{U}$, pro něž existuje polynomiální algoritmus, který řeší $\mathcal{U}$, tj. algoritmus, který má složitost O(p(n)) pro nějaký polynom p(n).

NP (*nondeterministic polynomial*)
Třída *NP* je třída rozhodovacích úloh, pro něž existuje nedeterministický algoritmus pracující v polynomiálním čase.

NP – úplné (*NP-complete*)
Rozhodovací úloha $\mathcal{U}$ je *NP-úplná*, jestliže je *NP*-úlohou a každá *NP*-úloha se na $\mathcal{U}$ polynomiálně redukuje. Tyto úlohy jsou nejtěžší mezi všemi *NP*-úlohami.

NP - těžké (*NP-hard*)
Rozhodovací úloha $\mathcal{U}$ je *NP-těžká*, jestliže se na ni polynomiálně redukuje každá *NP*-úloha.

Množina všech *P* úloh je podmnožinou *NP*-úloh, není ale dokázáno a nepředpokládá se, že by tyto množiny byly totožné. Podrobnější informace lze nalézt např. v [Mařík]. Předpokládané rozdělení tříd dobře ilustruje následující obrázek:



obr. 2.2: rozdělení tříd úloh

3. Formulace zadání v terminologii rozvrhování

Z předcházejícího stručného úvodu do teorie rozvrhování je na první pohled zřejmé, že optimální řízení nákladních výtahů je velice široký pojem, který je třeba pro další práci blíže specifikovat. V této kapitole tedy náš problém konkretizujeme a vyjádříme ve standardní $\alpha \mid \beta \mid \gamma$ notaci. Jednoznačně tak definujeme typ procesorů, daná omezení a požadované kritérium optimalizace.

Představme si zadavatele jako firmu nebo jiný subjekt, jenž disponuje omezeným počtem výtahů (procesorů) a požaduje optimální využití těchto prostředků tak, aby byla minimalizována průměrná doba odbavení nákladu. Tomuto požadavku formálně odpovídá[2] kritérium optimalizace $\sum c_j$. Výtahy mají shodné rozměry[3] a totožnou konstantní rychlost, nezávislou na druhu převáženého nákladu (typ procesorů P). Předřazenou část distribučního řetězce tvoří nákladní vozidla, přijíždějící k výtahům ve stanovený čas (omezení r_j) podle předem známého rozpisu (statické rozvrhování). Pokud je již vozidlo přistaveno k některému z výtahů, musí být celý jeho náklad přepraven právě tímto výtahem (rozvrhování bez preempcí)[4]. Dalším omezením je fakt, že zadavatel požaduje, aby jednotlivým nákladům bylo možné přiřadit různou prioritu v závislosti na době exspirace zboží obsaženého v nákladu. Tím se optimalizační kritérium mění na minimalizaci vážené průměrné doby odbavení, čemuž odpovídá $\sum w_j c_j$.

Takto formulované zadání lze ve standardní notaci označit jako $\mathbf{P \mid r_j \mid \sum w_j c_j}$, tedy jako rozvrhování na identické paralelní procesory s definovanými okamžiky disponibility úloh, řešící minimalizaci váženého součtu dob dokončení úloh. Vzhledem k nestejným okamžikům disponibility úloh se jedná o *NP – těžkou* úlohu a to dokonce i pro jeden procesor a úlohami se shodnými prioritami.

Vhodným rozšířením tohoto zadání je pak možnost rozvrhovat úlohy i na uniformní resp. nesouvztažné procesory, což by odpovídalo situaci s výtahy rozdílných rychlostí či rozměrů resp. výtahy s rychlostmi závislými na právě přepravovaném nákladu například vlivem různých hmotností nákladů. Ve standardní notaci by tedy šlo o problém $\mathbf{Q \mid r_j \mid \sum w_j c_j}$ respektive $\mathbf{R \mid r_j \mid \sum w_j c_j}$.

Dále je možné omezit rozvrhování úloh precedenčními závislostmi. Takové omezení by našlo uplatnění u složitějšího modelu přepravy například přes mezisklad, kde náklad vložený do meziskladu jako první by mohl být dále přepravován až po odbavení všech nákladů vložených do meziskladu později. Tato situace by spolu s předchozím rozšířením odpovídala problému $\mathbf{R \mid r_j, prec \mid \sum w_j c_j}$.

---

[2] Počet nákladů, které je třeba přepravit (úloh) je v daném okamžiku konstantní. Minimalizací součtu okamžiků odbavení tak zároveň minimalizujeme průměrnou dobu odbavení.

[3] Uvažujeme zde případ, kdy by rozdílné rozměry výtahů mohly mít za následek odlišné rychlosti přepravy nákladu. K takové situaci by mohlo dojít pokud by uvažovaný náklad mohl být jedním výtahem přepraven naráz a jiným na několik fází, díky jeho nedostatečným rozměrům vzhledem k objemu nákladu.

[4] Přejezdy vozů k jiným výtahům by byly značně neekonomické.

4. List Scheduling a heuristiky

V této kapitole popíšeme základní algoritmus *List Scheduling* včetně některých, na něm založených heuristických algoritmů.

4.1. List Scheduling

Jako jednoduchý kombinatorický algoritmus je *List Scheduling* (*LS*) dobře znám již polovinu století. V tomto algoritmu jsou úlohy vybírány z předem daného seznamu (*list*). Jakmile je procesor volný, zařadí se na něj první dostupná úloha ze seznamu. Dostupnost úlohy v tomto případě znamená, že úloha je uvolněná (disponibilní) a pokud je vázána precedenčními závislostmi, jsou všichni její předchůdci dokončeni. [Leung]

Vzhledem k jednoduchosti algoritmu a faktu, že řešení problému lze nalézt pouhým seřazením úloh v seznamu do vhodného pořadí, je tento algoritmus nejznámějším postupem rozvrhování. Díky tomu, že ke své funkci nepotřebuje žádné informace o již rozvržených úlohách, ani o zatím nerozvržených úlohách, je velice mocným nástrojem v dynamickém rozvrhování (*on line scheduling*), zvláště pak v neprediktivním (*nonclairvoyance*) dynamickém rozvrhování. [Leung]

Dodejme, že pro více procesorů je algoritmus totožný, jen v každé iteraci je uvažován vždy procesor s aktuálně nejnižším časem a ostatní procesory jsou relaxovány. Je vhodný pro řešení problému $\mathbf{P||c_{max}}$, $\mathbf{P|prec|c_{max}}$ nebo jednoprocesorových $\mathbf{1||c_{max}}$ a $\mathbf{1|prec|c_{max}}$. V této práci použijeme výše zmíněný algoritmus jako základ, který rozšíříme o heuristiky řešící naše zadaní.

Při řešení problému $\mathbf{P||c_{max}}$ má algoritmus *LS* lineární asymptotickou časovou složitost *O(n)* a chová se jako aproximační algoritmus s maximální vzdáleností řešení od optima:

$$R_{LS} = 2 - \frac{1}{m}, \qquad \text{[Błażewicz]}$$

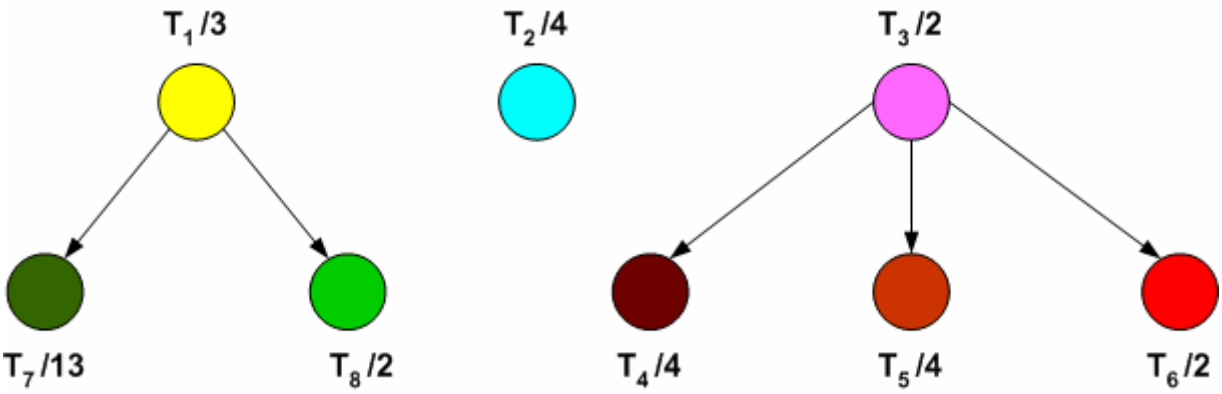
kde *m* je počet procesorů a R_{LS} je poměr nalezeného řešení vůči optimu. Je patrné, že při řešení tohoto problému s jedním procesorem nalezne algoritmus optimální řešení vždy. S rostoucím počtem procesorů pak stoupá maximální možná vzdálenost od optimálního řešení, není však nikdy větší, než dvojnásobek optima.

4.1.1. Paradoxy algoritmu List Scheduling

Jednoduchost algoritmu *List Scheduling* může mít za následek jeho neočekávané chování. V situacích, ve kterých bychom předpokládali snížení celkové délky rozvrhu, se délka rozvrhu může paradoxně zvýšit. S takovým chováním algoritmu se můžeme setkat v následujících situacích:

- počet procesorů se zvýší
- doba vykonávání všech úloh se sníží
- některé precedenční závislosti jsou relaxovány
- změní se pořadí úloh v seznamu

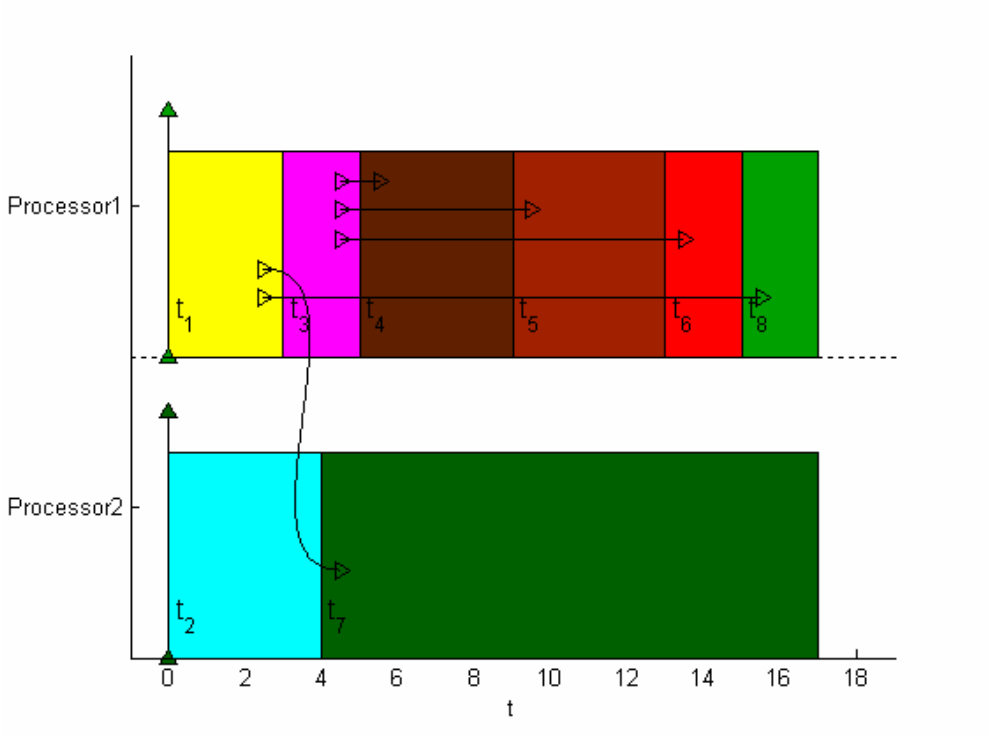
Demonstrujme toto paradoxní chování algoritmu *List Scheduling* na následujícím teoretickém příkladu uvedeném v [Błażewicz]. Uvažujme seznam úloh {T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8} s dobami vykonávání a precedenčními závislostmi znázorněnými následujícím grafem.



obr. 4.1: paradoxy LS, zadaná množina úloh

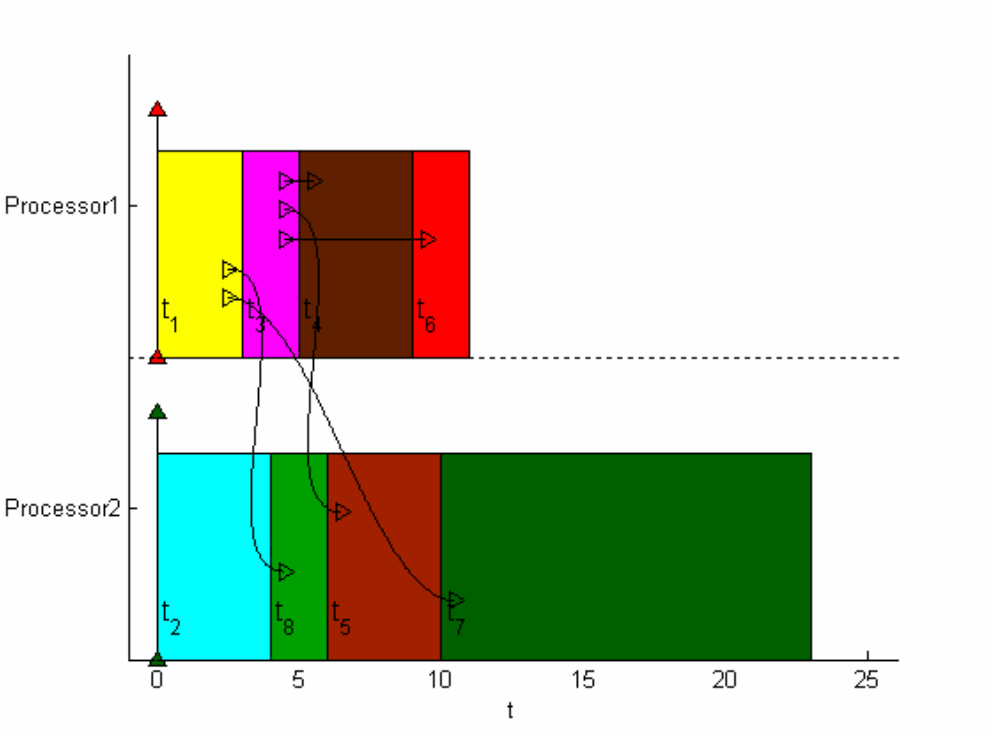
Tyto úlohy budeme rozvrhovat na dva paralelní identické procesory a požadavkem bude minimalizace kritéria c_{max}. Řešíme tedy problém vyjádřený ve standardní notaci jako **P2|prec|c_{max}**. Výsledné rozvrhy budou prezentovány formou *Ganttových diagramů* [Błażewicz].

Příklad je zadán vhodným způsobem tak, že algoritmus nalezne optimální řešení problému s celkovou délkou rozvrhu $c_{max} = 17$.



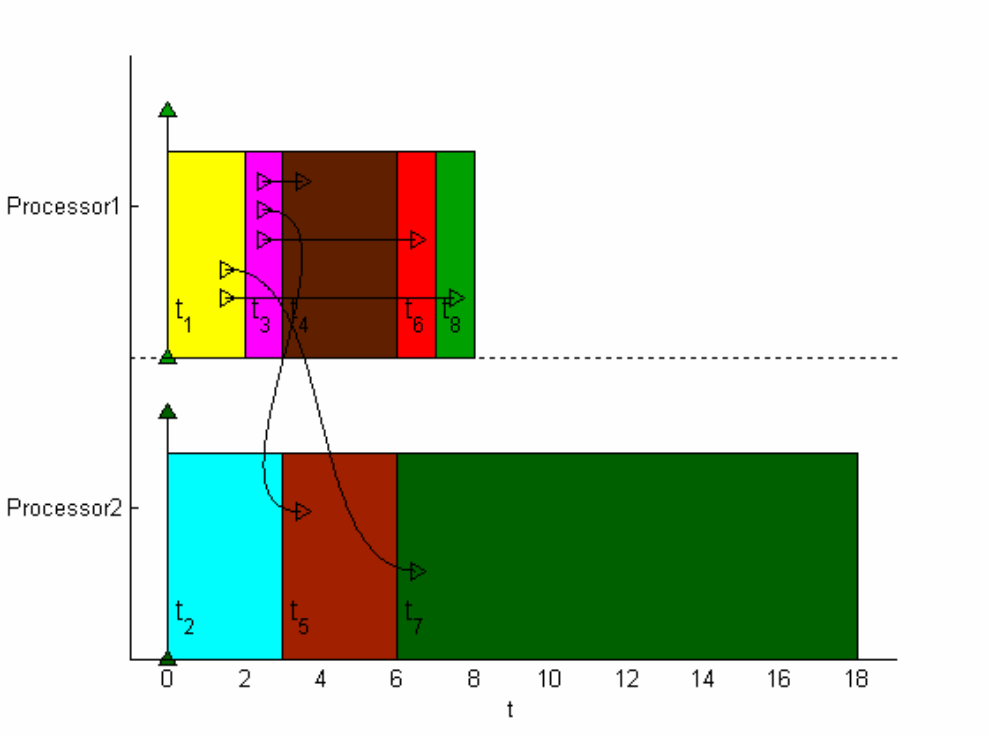
obr. 4.2: paradoxy LS, optimální řešení

Pouhou záměnou pořadí dvou úloh v seznamu na { T_1, T_2, T_3, T_4, T_5, T_6, $\mathbf{T_8}$, $\mathbf{T_7}$} se celková doba rozvrhu zvýší na $c_{max} = 23$.



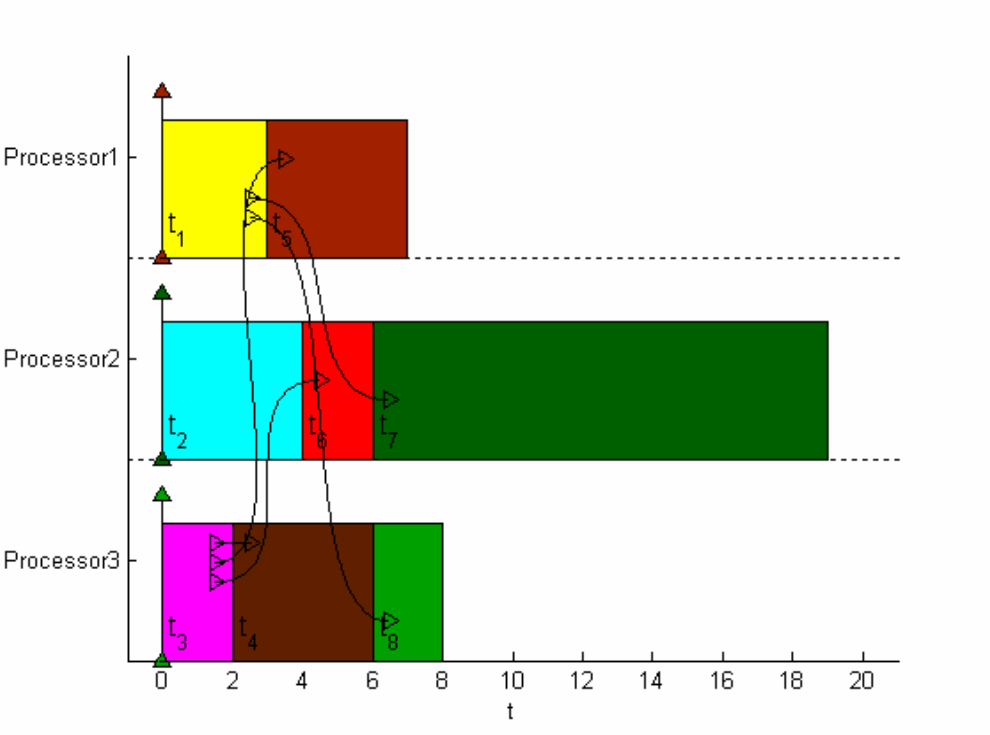
obr. 4.3: paradoxy LS, změna pořadí úloh

Snížení doby vykonávání všech úloh $p_j' = p_j - 1$ vede k prodloužení rozvrhu na $c_{max} = 18$.



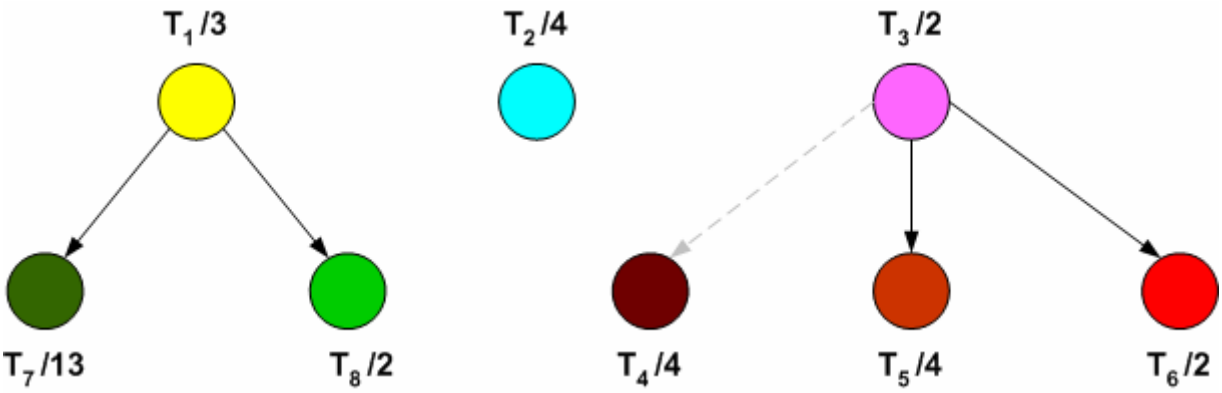
obr. 4.4: paradoxy LS, snížení doby vykonávání úloh

Rovněž přidáním jednoho procesoru se celková doba rozvrhu zvýší na $c_{max} = 19$.

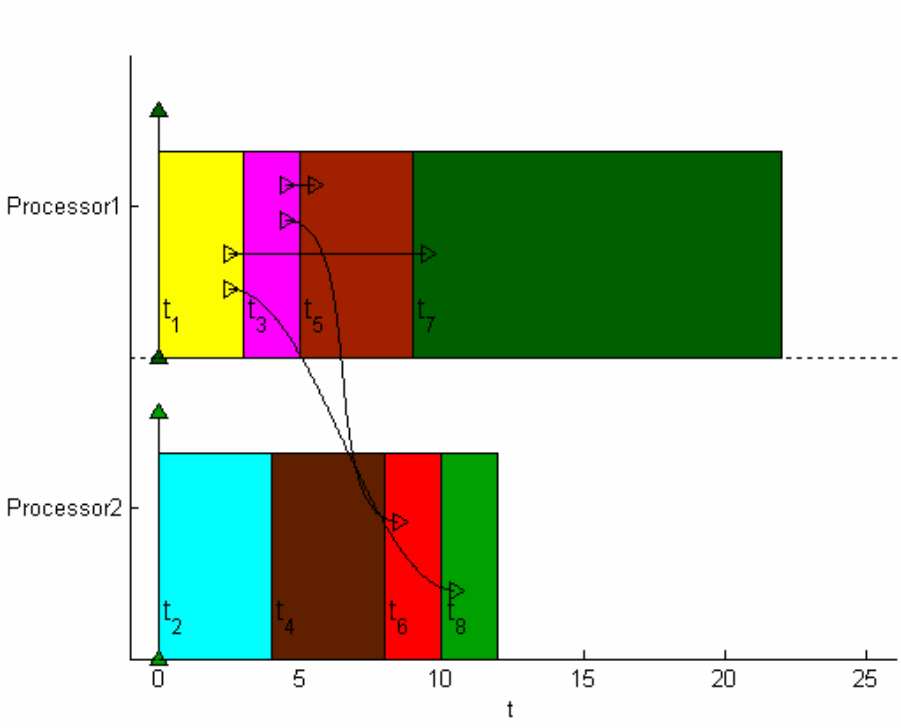


obr. 4.5: paradoxy LS, zvýšení počtu procesorů

Relaxací jedné z precedenčních závislostí se délka rozvrhu zvýší na $C_{max} = 22$.



obr. 4.6: paradoxy LS, relaxace precedenční závislosti, graf



obr. 4.7: paradoxy LS, relaxace precedenční závislosti, rozvrh

Tyto anomálie jsou ve *Scheduling Toolbox*u implementovány jako funkce *listsch_paradox_demo.m*. Blíže je popisuje [Graham], který také nalezl horní hranice změn délky rozvrhu v závislosti na změnách jednoho či více parametrů.

4.1.2. Algoritmus List Scheduling ve Scheduling Toolboxu

Algoritmus *List Scheduling* jsem implementoval jako součást *Scheduling Toolbox*u funkcí:

```
S = listsch(T,p,m [,h] [,v]);
```

Vstupem je soubor úloh (*taskset*) *T*, problém *p*, který má algoritmus řešit a počet procesorů *m*, na které mají být úlohy rozvrženy. Dále je možné zadat volitelný vstupní parametr *h* určující použitý heuristický algoritmus a *verbose mod v* určující úroveň komentářů vypisovaných algoritmem. Výstupem je pak rozvrh *S*.

Volitelné parametry lze funkci předávat i pomocí konfigurační proměnné *Scheduling Toolbox*u vytvořené funkcí *schoptionsset*:

```
option = schoptionsset('heuristic',h,'verbose',v);
S = listsch(T,p,m [,option]);
```

Soubor úloh *T* je možné definovat více způsoby pomocí funkce *taskset*. K nejjednodušším patří následující postup pomocí vektoru časů vykonávání úloh *ProcessTime*:

```
T = taskset(ProcessTime,prec);
```

Pokud mezi úlohami existují precedenční závislosti, jsou uvedeny v matici *prec* o *j* řádcích a *i* sloupcích, kde $j = i = n$ je počet úloh v seznamu. Pokud úloha *i* je následníkem úlohy *j*, pak $prec(j,i) = 1$, jinak $prec(j,i) = 0$. V případě, že precedenční závislosti nejsou definovány, je možné tuto matici vynechat. Další možné způsoby definice souboru úloh si ukážeme na následujících praktických příkladech.

Problém *p* definujeme pomocí funkce *problem*, jejíž vstupním parametrem je řetězec ve formě standardní notace:

```
p = problem('P|prec|Cmax');
```

Poslední verze funkce *listsch* je schopna řešit problém $\mathbf{R|r_j,prec|\sum w_j c_j}$ a všechny z něho odvozené, jednodušší problémy viz. [Brucker&Knust].

Heuristický algoritmus *h* lze zvolit z několika již implementovaných heuristik jako LTP, SPT, EST a ECT:

```
S = listsch(T,p,m,'EST');
```

Dále je možné implementovat vlastní heuristický algoritmus viz. kapitola 4.2.4 - Uživatelsky definované heuristiky.

Verbose mod v rozlišuje tři úrovně výpisů komentářů funkce *listsch*, 0 – žádný výpis, 1 – stručný výpis a 2 – detailní výpis:

```
S = listsch(T,p,m,2);
```

Veškeré informace o implementované funkci *listsch* jsem uvedl včetně ukázkových příkladů použití v dokumentaci, která je vytvářena souběžně s vývojem *Scheduling Toolbox*u viz. [Scheduling].

Příklad 4.1: Demonstrace algoritmu List Scheduling

Uvažujme soubor pěti úloh s následujícími parametry:

	jméno	doba vykonávání
1	$task_1$	4
2	$task_2$	5
3	$task_3$	3
4	$task_4$	1
5	$task_5$	2

tabulka 4.1: demonstrace LS, zadání

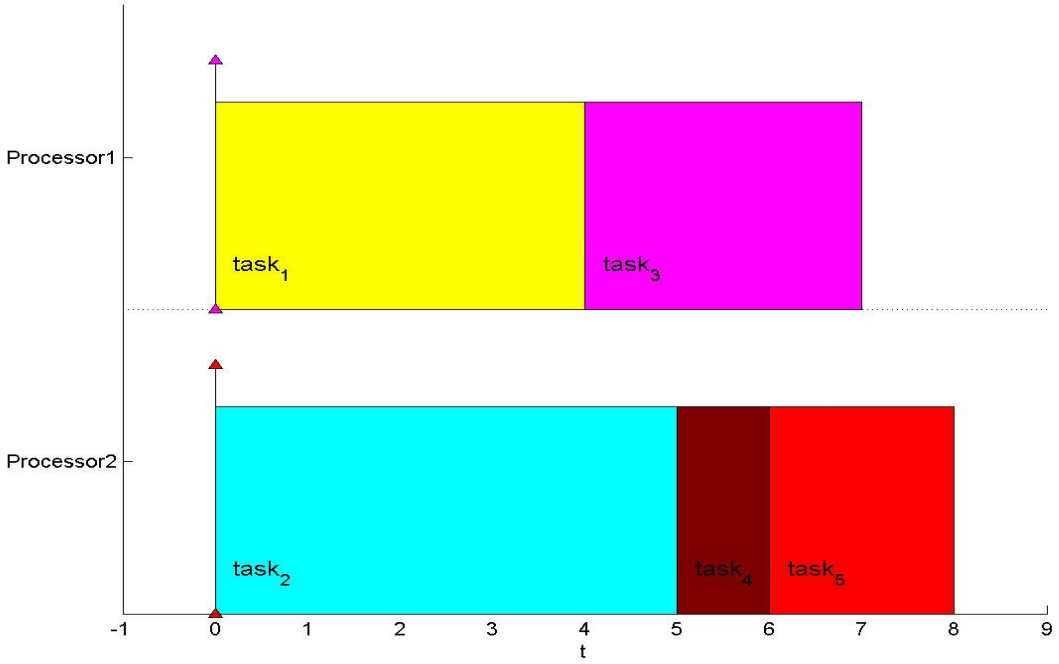
Požadujeme, aby tyto úlohy byly rozvrženy na dva identické paralelní procesory. Optimalizačním kritériem bude celková délka rozvrhu c_{max}. Jedná se tedy o problém, jenž lze ve standardní notaci vyjádřit jako $\mathbf{P2||c_{max}}$.

Demonstrujme použití algoritmu *List Scheduling* v prostředí *MATLAB*. Nadefinujeme si tedy vstupní parametry a vypočteme výsledný rozvrh. Zároveň zde předvedeme další ze způsobů jak definovat množinu úloh pomocí funkcí *task* a *taskset*.

```
% matlab kód bp_00.m
% demonstrace algoritmu List Scheduling

t1 = task('task1',4);     % definice úlohy 1
t2 = task('task2',5);     % definice úlohy 2
t3 = task('task3',3);     % definice úlohy 3
t4 = task('task4',1);     % definice úlohy 4
t5 = task('task5',2);     % definice úlohy 5

T = taskset([t1 t2 t3 t4 t5]);   % definice souboru úloh
p = problem('P|prec|Cmax');  % definice problému

S = listsch(T,p,2);              % algoritmus List Scheduling pro dva procesory
plot(S);                         % vykreslení rozvrhu
```

Matlab kód 4.1: demonstrace algoritmu List Scheduling

Na obrázku obr. 4.8 je uveden výsledný rozvrh algoritmu *List Scheduling* ve formě *Ganttova* diagramu. Z něho je patrné, že úlohy byly rozvrženy v pořadí, v jakém jsou definovány v seznamu. Tedy $\mathbf{task_1}$, $\mathbf{task_2}$, $\mathbf{task_3}$, $\mathbf{task_4}$ a $\mathbf{task_5}$.



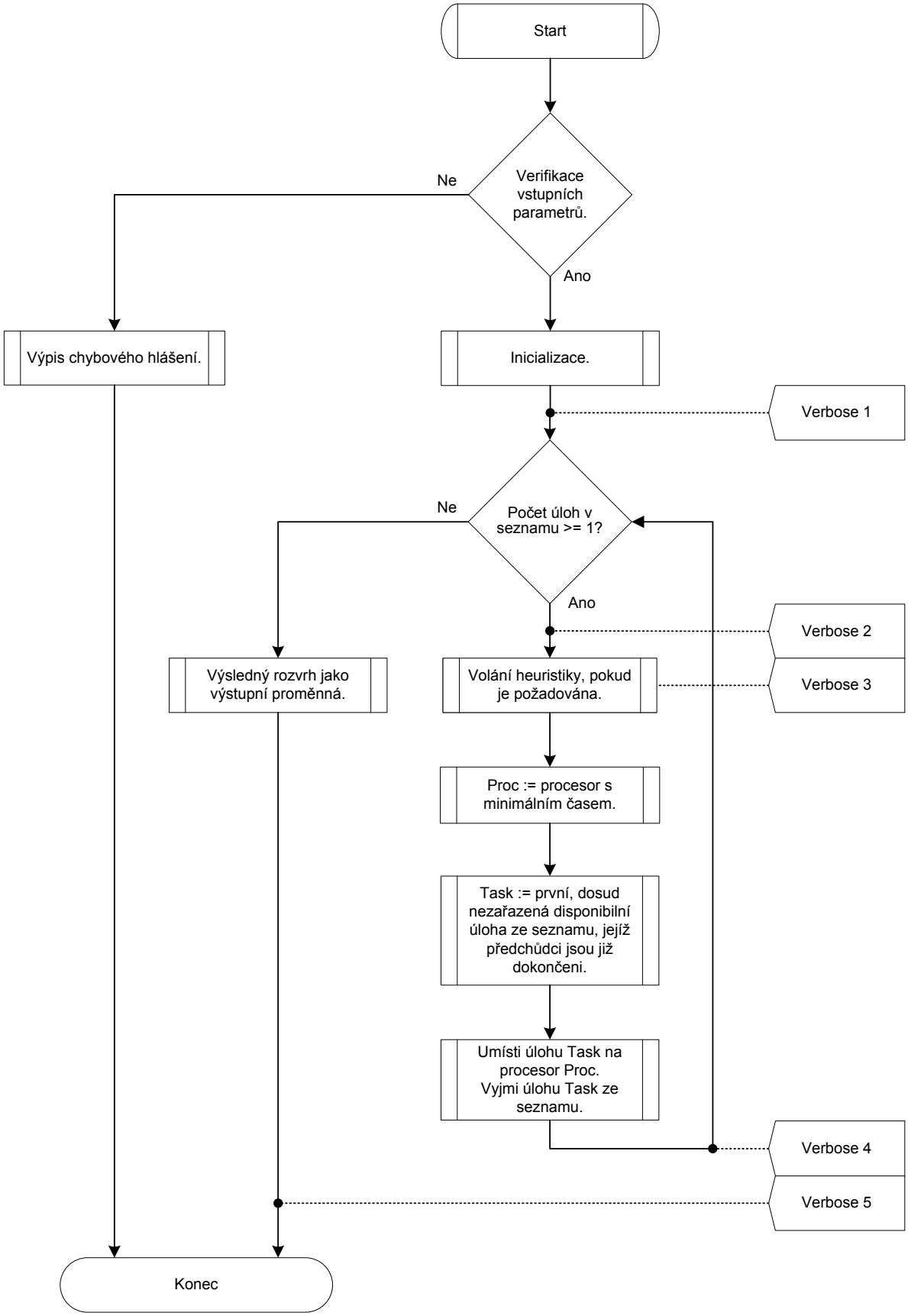
obr. 4.8: demonstrace LS, rozvrh

4.1.3. Implementace algoritmu List Scheduling

Na obrázku obr. 4.9 je znázorněn vývojový diagram algoritmu *List Scheduling* implementovaného funkcí *listsch*. Po kontrole vstupních parametrů, zejména pak ověření, zda je algoritmus schopen řešit požadovaný problém, následuje inicializace vnitřních proměnných a hlavní smyčka algoritmu.

V té je nejprve volána požadovaná heuristika, následuje zvolení procesoru s minimálním časem a výběr první dosud nezařazené disponibilní úlohy ze seznamu, jejíž předchůdci jsou již dokončeni. Tato úloha je umístěna na vybraný procesor a odstraněna ze seznamu úloh. Hlavní smyčka a tím i celý algoritmus terminuje v okamžiku, kde se v seznamu již nenachází žádná úloha. Výstupem funkce je rozvrh a informace o výpočetní době algoritmu a hodnotě minimalizovaného kritéria.

Pokud je aktivován *verbose mod*, vypisuje funkce v průběhu celého programu informace o právě provedených akcích ve třech možných stupních do okna *MATLAB workspace*, což je vhodné zejména pro demonstraci funkce algoritmu či výuku. Pro zachování přehlednosti kódu je *verbose mod* realizován samostatným skriptem *listsch_verbose.m*, který však sdílí všechny proměnné s funkcí *listsch*.



obr. 4.9: vývojový diagram algoritmu *List Scheduling*

4.2. Heuristiky

Heuristický neboli suboptimální algoritmus je takový postup, který směřuje k nalezení řešení, negarantuje však nalezení optimálního řešení pro každou instanci optimalizačního problému.

Nutným předpokladem pro použití těchto algoritmů v praxi je samozřejmě omezení jejich složitostní funkce shora polynomiální funkcí nízkého řádu. Další podmínka pak vyplývá z ohodnocení vzdálenosti mezi řešením, jež nalezla heuristika a optimálním řešením. Toto ohodnocení může vycházet ze středního nebo nejhoršího případu vstupní instance.

Nicméně, pro některé kombinatorické algoritmy může být dokázáno, že nalezení heuristického algoritmu s blíže specifikovanou přesností není možné a jedná se o stejně těžkou otázku, jako nalezení algoritmu s polynomiální složitostí pro *NP – úplnou* úlohu. [Błażewicz]

Dodejme jen, že pokud lze pro heuristický algoritmus nalézt horní hranici vzdáleností jeho řešení od optimálních řešení, nazývá se aproximačním algoritmem.

4.2.1. EST

Pokud před aplikací algoritmu *List Scheduling* seřadíme úlohy v seznamu do neklesající posloupnosti podle času disponibility r_j (*release time*), je výsledkem algoritmus známý jako *EST* [5](*Earliest Starting Time first*), řešící problém $\mathbf{P|r_j|\sum c_j}$.

Pro tuto heuristiku není známa žádná funkce ohraničující její přesnost. [Błażewicz] Nemůžeme tedy v obecné rovině říci nic o vzdálenosti jejích řešení od řešení optimálních. Asymptotická časová složitost této heuristiky vychází z časové složitosti algoritmu pro třídění n prvků. *Každý algoritmus, který ke třídění používá operaci porovnávání, má časovou složitost O(nlogn)*. [Mařík] Je tedy zřejmé, že heuristika EST má rovněž asymptotickou časovou složitost *O(nlogn)*.

Pro naše účely je však potřeba tuto heuristiku upravit tak, aby na výsledný rozvrh měla vliv i priorita úloh. Úlohy tedy seřadíme do neklesající posloupnosti podle poměru

$$r_j' = \frac{r_j}{w_j}, \qquad \textbf{(4.1)}$$

kde r_j je okamžik disponibility, r_j‘ vážený okamžik disponibility a w_j je priorita úlohy j. Heuristika poté řeší náš problém $\mathbf{P|r_j|\sum w_j c_j}$.

Heuristický algoritmus EST jsem do *Scheduling Toolbox*u implementoval funkcí *sort_est.m* volanou na základě volitelného parametru funkce *listsch*:

```
S = listsch(T,p,m,‘EST‘);
```

Ukažme si použití algoritmu EST na praktickém příkladu, ze kterého je patrný vliv priority úlohy na výsledný rozvrh.

[5] V některých publikacích je tento algoritmus také označován zkratkou ETF (*Earliest Task First*) viz [Boeres].

Příklad 4.2: Demonstrace algoritmu EST

Uvažujme soubor pěti úloh s parametry danými následující tabulkou.

	jméno	doba vykonávání	okamžik disponibility	priorita 1	priorita 2
1	$task_1$	4	1	1	1
2	$task_2$	4	2	1	1
3	$task_3$	2	3	1	3
4	$task_4$	4	0	1	1
5	$task_5$	1	2	1	1

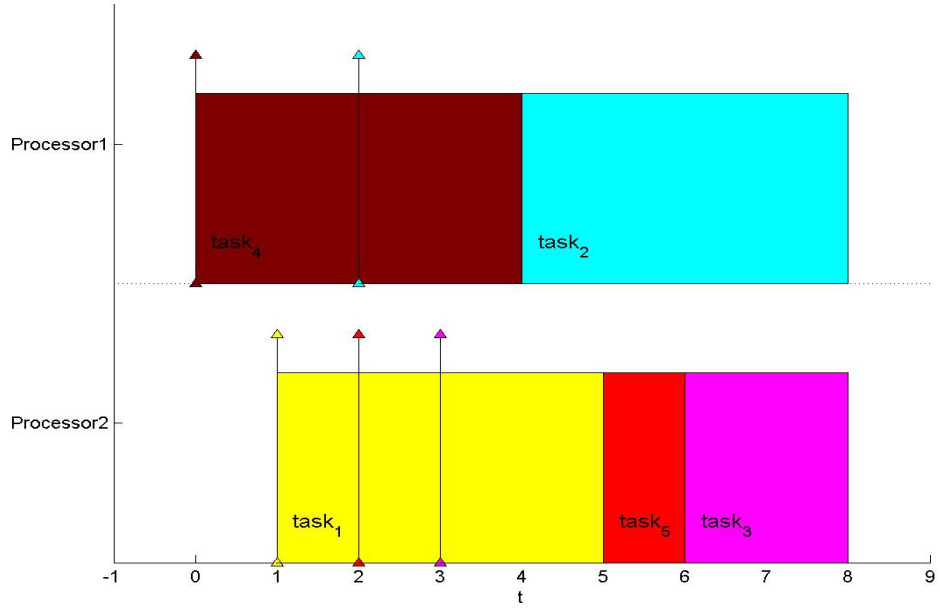
tabulka 4.2: demonstrace EST, zadání

Požadujeme rozvržení těchto úloh na dva identické paralelní procesory s omezením okamžiku disponibility r_j a priority úloh w_j. Kritériem optimalizace bude $\sum w_j c_j$. Tento problém lze ve standardní notaci označit jako $\mathbf{P2|r_j|\sum w_j c_j}$.

```
% matlab kód bp_01.m
% demonstrace algoritmu EST

t1 = task('task1',4,1);          % definice úlohy 1
t2 = task('task2',4,2);          % definice úlohy 2
t3 = task('task3',2,3);          % definice úlohy 3
t4 = task('task4',4,0);          % definice úlohy 4
t5 = task('task5',1,2);          % definice úlohy 5
T = taskset([t1 t2 t3 t4 t5]);   % definice souboru úloh
p = problem('P|rj|sumCj');       % definice problému

S = listsch(T,p,2,'EST');        % algoritmus EST pro dva procesory
figure(1);                       % nové okno
plot(S);                         % vykreslení rozvrhu

T.Weight = [1 1 3 1 1];          % změna priority úlohy task3 na hodnotu 3
p = problem('P|rj|sumwCj');      % definice problému
S = listsch(T,p,2,'EST');        % algoritmus EST pro dva procesory
figure(2);                       % nové okno
plot(S);                         % vykreslení rozvrhu
```

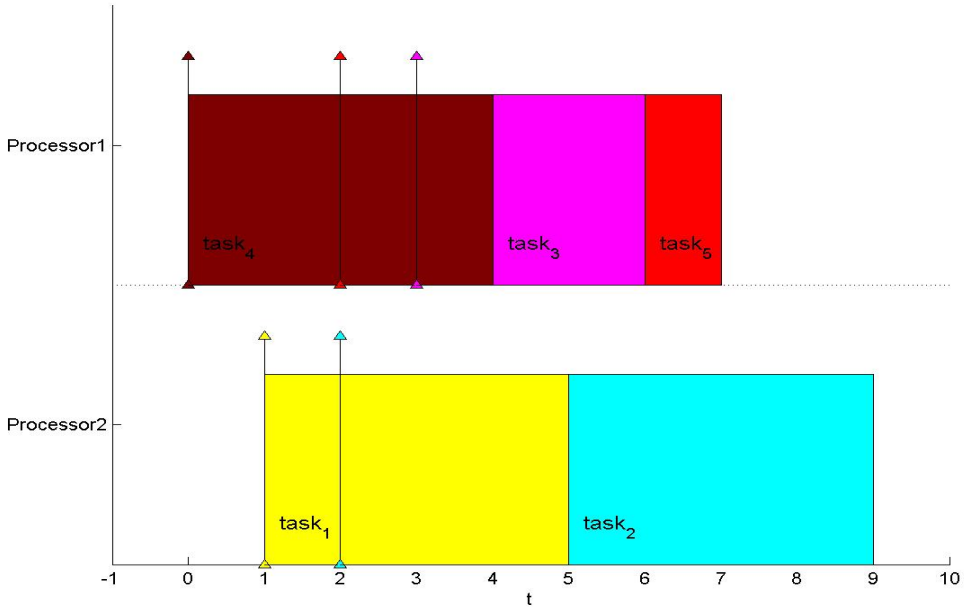
Matlab kód 4.2: demonstrace algoritmu EST

Výsledný rozvrh algoritmu EST pro úlohy stejných priorit:



obr. 4.10: demonstrace EST, úlohy shodných priorit

Výsledný rozvrh algoritmu EST po zvýšení priority úlohy $task_3$ $w_3 = 3$:



obr. 4.11: demonstrace EST, úlohy rozdílných priorit

Z obrázku obr. 4.10 je patrné, že úlohy jsou do rozvrhu vkládány v pořadí podle neklesajícího času disponibility, tedy **task$_4$**, **task$_1$**, **task$_2$**, **task$_5$** a na závěr **task$_3$**. Pokud se v seznamu vyskytuje více úloh s totožným časem disponibility, tak jako v případě úloh **task$_2$** a **task$_5$**, jsou do rozvrhu zařazeny v pořadí v jakém se nacházejí v seznamu.

Na obrázku obr. 4.11 je znázorněn rozvrh heuristiky EST po změně priority jedné z úloh. Zvýšení priority úlohy **task$_3$** na w_3 = 3 odpovídá dle vztahu (4.1) snížení okamžiku disponibility této úlohy z r_3 = 3 na r_3 ‘= 1. Ve skutečnosti má tato úloha stále původní okamžik disponibility, ale pro takto upravenou heuristiku EST se jeví jako srovnatelná s úlohou task$_1$ a proto je do rozvrhu zařazena bezprostředně po této úloze.

4.2.2. ECT

Přerovnáním úloh v seznamu do neklesající posloupnosti podle času dokončení c_j v každé iteraci algoritmu *List Scheduling* dostáváme algoritmus ECT (*Earliest Completion Time first*), řešící problém **P|rj|∑cj**.

Pro tuto heuristiku není rovněž známa žádná funkce ohraničující její přesnost. [Błażewicz] Za horní hranici časové složitosti algoritmu ECT lze považovat $O(n^2 logn)$ viz. *A. příloha*.

Úlohy jsou zde vybírány ze seznamu v pořadí od nejmenšího času dokončení k největšímu. Situace je zde však mírně odlišná než u algoritmu EST, neboť je nutné toto seřazení provést opakovaně, při každé iteraci algoritmu *List Scheduling*. Reálný čas dokončení úlohy není závislý jen na času disponibility a délce vykonávání úlohy, ale i na aktuálním času procesoru a to vztahem

$$c_j = \max\left(r_j, t_{proc}\right) + p_j, \qquad \textbf{(4.2)}$$

kde c_j je čas dokončení, r_j okamžik disponibility, p_j doba vykonávání úlohy j a t_{proc} je čas procesoru, který je právě uvažován. Podobně jako u heuristiky EST modifikujeme algoritmus ECT tak, aby priorita úlohy ovlivňovala výsledný rozvrh seřazením úloh do neklesající posloupnosti dle váženého času dokončení viz. vztah (4.3). Po tomto rozšíření umožňuje heuristika řešit problém **P|r$_j$|∑w$_j$c$_j$**.

$$c_j' = \frac{c_j}{w_j}, \qquad \textbf{(4.3)}$$

Algoritmus ECT jsem do *Scheduling Toolbox*u implementoval obdobně jako EST, tedy funkcí *sort_ect.m* volanou na základě volitelného parametru funkce *listsch*:

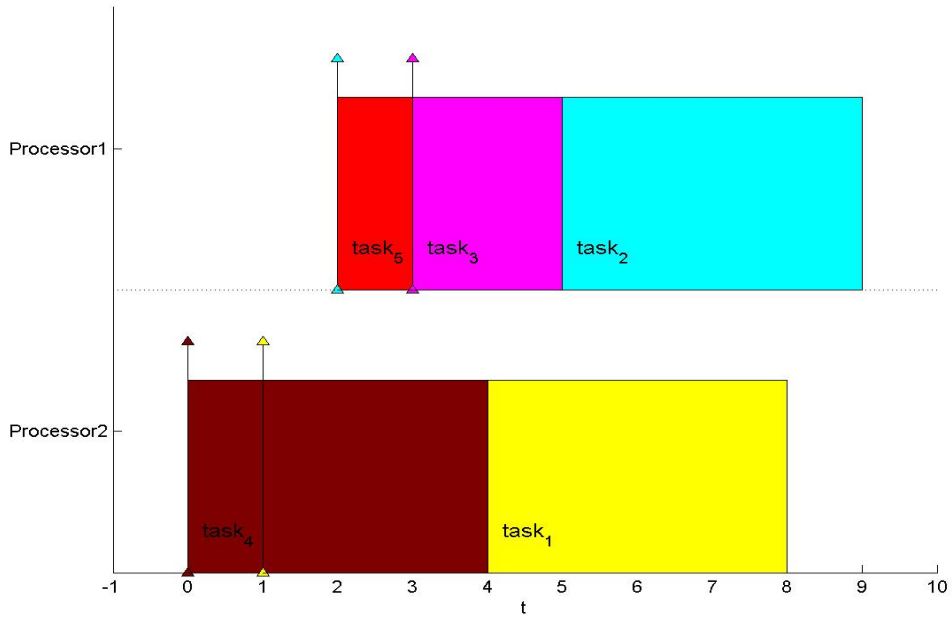
```
S = listsch(T,p,m,‘ECT‘);
```

Použití algoritmu ECT demonstrujme na stejném příkladu jako algoritmus EST.

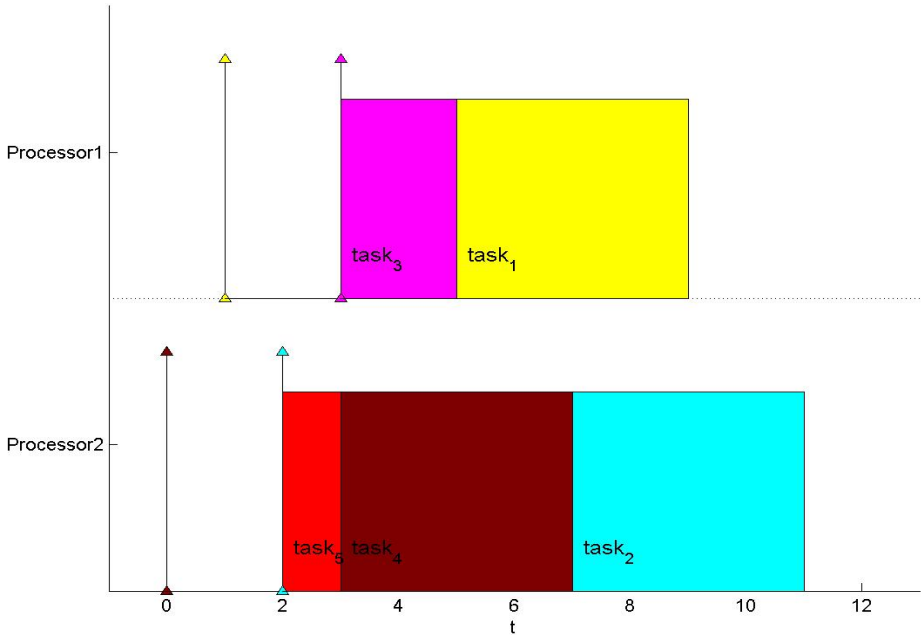
Příklad 4.3: Demonstrace algoritmu ECT
Zadání viz. příklad 4.2: Demonstrace algoritmu EST.

```
% matlab kód bp_02.m
% demonstrace algoritmu ECT

t1 = task('task1',4,1);          % definice úlohy 1
t2 = task('task2',4,2);          % definice úlohy 2
t3 = task('task3',2,3);          % definice úlohy 3
t4 = task('task4',4,0);          % definice úlohy 4
t5 = task('task5',1,2);          % definice úlohy 5
T = taskset([t1 t2 t3 t4 t5]);   % definice souboru úloh
p = problem('P|rj|sumCj');       % definice problému

S = listsch(T,p,2,'ECT');         % algoritmus ECT pro dva procesory
figure(1);                       % nové okno
plot(S);                         % vykreslení rozvrhu

T.Weight = [1 1 3 1 1];          % změna priority úlohy task3 na hodnotu 3
p = problem('P|rj|sumwCj');      % definice problému
S = listsch(T,p,2,'ECT');        % algoritmus ECT pro dva procesory
figure(2);                       % nové okno
plot(S);                         % vykreslení rozvrhu
```

Matlab kód 4.3: demonstrace algoritmu ECT



obr. 4.12: demonstrace ECT, úlohy shodných priorit



obr. 4.13: demonstrace ECT, úlohy rozdílných priorit

Z obr. 4.12 je patrné, že úlohy byly na procesory rozvrženy v pořadí **task$_5$**, **task$_4$**, **task$_3$**, **task$_1$**, **task$_2$**. Změna priority úlohy **task$_3$** z $w_3 = 1$ na $w_3 = 3$ však sníží dobu dokončení podle vztahu (4.3) na $c_3' = 1{,}66$, což je v počátečním stavu nejnižší hodnota. Tím se úloha **task$_3$** stává první v pořadí. Pořadí zbylých úloh zůstalo zachováno.

4.2.3. Další heuristiky

Podobných heuristických algoritmů popsaných v předchozí kapitole existuje celá řada. Zmiňme v této kapitole alespoň krátce některé z nich.

4.2.3.1. LPT

Přerovnáním úloh v seznamu do nerostoucí posloupnosti podle doby jejich vykonávání p_j před aplikací algoritmu *List Scheduling*, dostáváme algoritmus známý jako LPT (*Longest Processing Time first*), který obecně řeší problém $\mathbf{P||c_{max}}$. Tato heuristika je ve *Scheduling Toolbox*u rovněž implementována.

Jedná se o aproximační algoritmus s časovou složitostí *O(nlogn)*, jehož přesnost pro nejhorší případ vstupní instance problému $\mathbf{P||c_{max}}$ dána vztahem

$$R_{LPT} = \frac{4}{3} - \frac{1}{3m}, \qquad \text{[Błażewicz]}$$

kde *m* je počet procesorů, na které jsou úlohy rozvrhovány a R_{LPT} je poměr nalezeného řešení oproti optimálnímu řešení.

Další vztah, vyjadřující horní hranici nepřesnosti algoritmu LPT vychází nejen z počtu procesorů *m*, ale i z počtu úloh *k* na procesoru, jehož poslední úloha uzavírá rozvrh:

$$R_{LPT} = 1 + \frac{1}{k} - \frac{1}{km}. \qquad \text{[Błażewicz]}$$

4.2.3.2. SPT

Pokud před aplikací algoritmu *List Scheduling* přerovnáme úlohy v seznamu do neklesající posloupnosti podle jejich doby vykonávání p_j, dostáváme algoritmus SPT (*Shortest Processing Time first*), řešící problém $\mathbf{P||\sum c_j}$.

Asymptotická časová složitost algoritmu SPT je *O(nlogn)*. Tato heuristika je ve *Scheduling Toolboxu* také implementována.

4.2.3.3. HLF

Algoritmus HLF (*Highest Level First*) je určen pro řešení problému $\mathbf{P|in\text{-}tree,\ p_j = 1|c_{max}}$ a je též znám jako Huův algoritmus (*Hu's algorithm*). Po vhodné úpravě stromové struktury precedenčních závislostí ho lze použít i pro řešení problému $\mathbf{P|out\text{-}tree,\ p_j = 1|c_{max}}$. Nespornou výhodou je možnost tento algoritmus implementovat s lineární časovou náročností *O(n)*.

Jeho základem je vytvoření dočasného druhého seznamu úloh, ve kterém se nacházejí pouze ty úlohy z původního seznamu, které nemají předchůdce nebo jsou všichni jejich předchůdci v aktuálním čase již dokončeni.

Tyto úlohy poté seřadíme do nerostoucí posloupnosti podle úrovně, na které se nacházejí ve stromové hierarchii. Přičemž pokud jde o „sbíhavou" (*in-tree*) hierarchii, má kořen stromu nejnižší úroveň.

Z takto seřazeného seznamu vybereme první úlohu, umístíme ji na procesor, vyjmeme z původního seznamu a dočasný seznam označíme za neplatný. Celý cyklus se opakuje a terminuje v okamžiku, kdy jsou již všechny úlohy rozvrženy. Krátce tento algoritmus zmiňuje [Leung] a více se jím zabývá např. [Błażewicz].

4.2.4. Uživatelsky definované heuristiky

Požadavkem na implementaci heuristických algoritmů založených na algoritmu *List Scheduling* byla snadná rozšiřitelnost o další heuristiky s možností implementovat vlastní, uživatelské heuristiky. Z důvodu těsné vazby mezi heuristikami a samotným algoritmem *List Scheduling*, jsou heuristiky volány na základě volitelného parametru tohoto algoritmu.

Z vývojového diagramu na obr. 4.9 je patrné, že heuristický algoritmus je opakovaně volán v každé iteraci algoritmu *List Scheduling*. Toto řešení je vhodné například pro heuristiku ECT, kdy se doba dokončení úloh (*completion time*) může mezi iteracemi měnit a je tedy nutné úlohy znovu seřadit. Naproti tomu heuristiku EST je nutné volat pouze v první iteraci, disponibilita úloh se v dalších iteracích nemění. Proto, jak si dále ukážeme, předává algoritmus *List Scheduling* heuristice číslo iterace ve vstupním parametru a až v těle heuristiky se rozhodne, zda je nutné úlohy přerovnat či nikoliv.

List Scheduling s vlastní heuristikou lze spustit obdobným způsobem, jako s již implementovanou heuristikou, prostým uvedením jména funkce, v níž je heuristický algoritmus implementován jako jeden z parametrů funkce *listsch*.

```
S = listsch(T,p,m,‘MaHeuristika‘);
```

Heuristiku je též možné spustit samostatně nad souborem úloh.

```
[taskset, order] = MaHeuristika(taskset [, iteration, procesor]);
```

Vstupem funkce je soubor úloh *taskset*. Volitelně je pak možné předávat heuristice aktuální iteraci algoritmu *List Scheduling iteration* a číslo procesoru *processor*, který je právě uvažován. Výstupem je funkcí upravený soubor úloh *taskset* a vektor *order* vyjadřující nové pořadí úloh v souboru vůči původnímu pořadí.

Zde je nutné podotknout, že parametry *iteration* a *procesor* jsou volitelné pro samostatné spuštění heuristiky nad souborem úloh, nicméně funkce *listsch* volá každou heuristiku vždy i s těmito parametry.

Jako příklad implementace vlastní heuristiky uvedeme implementaci algoritmu EST funkcí pojmenovanou *MyEST*.

```
% MyEST.m

function [taskset, order] = MyEST(taskset, varargin)

if nargin>1
    if varargin{1}>1
        order = 1:length(taskset.tasks);
        return
    end
end

wreltime=taskset.releasetime./taskset.weight;
[taskset order]=sort(taskset,wreltime,'inc'); % sort taskset
```

Matlab kód 4.4: uživatelsky definovaná heuristika EST

Vstupním parametrem *vavrargin* jsou reprezentovány volitelné parametry *iteration* a *procesor*. Na základě parametru *iteration* je v těle heuristiky rozhodnuto o tom, zda bude soubor úloh přerovnán či nikoliv. Pokud je parametr *iteration* větší než jedna a heuristika je tedy volána poněkolikáté, nedochází k přerovnání úloh a funkce vrací vektor *order* reprezentující původní, nezměněné pořadí úloh v seznamu. Nyní můžeme tuto heuristiku použít v algoritmu *List Scheduling*,

```
S = listsch(T,p,m,‘MyEST‘);
```

nebo ji volat samostatně nad souborem úloh.

```
[taskset, order] = MyEST(taskset [, iteration, procesor]);
```

5. Experimenty

V následující kapitole porovnáme výsledky implementovaných heuristických algoritmů EST, ECT a výsledky vypočtené pomocí nástroje *OPL Studio*, hledajícího optimální řešení.

Jak již bylo zmíněno výše, není pro tyto heuristické algoritmy známa jakákoliv horní hranice vzdálenosti od optimálního řešení. Navíc je jejich úspěšnost silně závislá na konkrétních parametrech úloh v dané vstupní instanci. Z těchto důvodů není možné algoritmy porovnávat na jednotlivých příkladech, jejichž výsledek tak má minimální vypovídací hodnotu, ale je nutné přistoupit ke statistickému porovnání výsledků na dostatečném počtu náhodně vybraných příkladů.

5.1. Statistické porovnání algoritmů EST a ECT

Abychom se vyhnuli nadměrnému množství dat ve statistice, je vhodné se alespoň v krátkosti zamyslet nad závislostmi, které nás budou zajímat a nad významem parametrů, jež budeme považovat za proměnné nebo konstantní, jako počet procesorů a úloh v seznamu, okamžik disponibility, priorita a doba vykonávání každé úlohy. Zajímat nás bude především vzdálenost řešení jednotlivých algoritmů a jejich časová náročnost v závislosti na počtu úloh a proměnné hustotě rozvrhu.

Počet procesorů m volíme vzhledem k charakteru zadání[6] jako konstantní parametr ($m=3$), kdy příklady nebudou zcela triviální, ani příliš složité. Pro větší počet procesorů bychom dosáhli stejných výsledků, ale pro dosažení stejné hustoty výsledného rozvrhu bychom museli uvažovat větší počet úloh. To by vedlo ke zbytečně dlouhému výpočetnímu času.

Počet úloh n v seznamu zvolíme jako nezávisle proměnný parametr ovlivňující hustotu výsledného rozvrhu i časovou náročnost výpočtu v rozsahu, kdy je řešení příkladu pro tři procesory zcela triviální ($n=3$) až po složitý, avšak stále ještě relativně rychle řešitelný rozvrh ($n=30$).

Priorita w_j a doba vykonávání p_j každé úlohy bude vybrána jako vzorek z rovnoměrného rozdělení $w_j \in \langle 1,3 \rangle$ a $p_j \in \langle 1,10 \rangle$.

Dalším parametrem, jenž bude ovlivňovat hustotu výsledného rozvrhu bude vedle počtu úloh i rozsah, z něhož budeme vybírat okamžiky disponibility úloh. Provedeme tedy samostatná měření pro tři rozsahy rovnoměrného rozdělení $r_j \in \langle 0,10 \rangle$, $r_j \in \langle 0,25 \rangle$ a $r_j \in \langle 0,50 \rangle$.

Všechna měření budou provedena na množině pětiset takto náhodně vybraných příkladů a výsledné hodnoty budou vypočítány jako aritmetický průměr. Pro procentuální vyjádření vzdáleností výsledků heuristik EST a ECT bude jako základ považován výsledek algoritmu EST. Pro srovnání výsledků heuristik s optimem pak jako základ poslouží hodnota optimálního řešení.

[6] Předpokládá se konstantní počet výtahů v provozu.

5.2. Implementace statistiky v prostředí Matlab

Výpočet statistiky jsem implementoval v prostředí *MATLAB* skriptem *stat.m* a funkcemi *stat_row.m*, *stat_cell.m* a *randtaskset.m*.

Funkce *stat_cell.m* generuje pomocí *randtaskset.m* vždy 500 seznamů úloh náhodných parametrů vybraných z rovnoměrného rozdělení o daném rozsahu a vypočte průměrnou hodnotu nalezeného řešení pomocí heuristik EST a ECT. Navíc každý vygenerovaný seznam úloh uloží do datového souboru pro pozdější zpracování nástrojem OPL.

Funkce *stat_row.m* pak tuto funkci opakovaně volá a postupně zvyšuje počet úloh v seznamu. Do souboru uloží průměrný výsledek EST a ECT, průměrný čas řešení pomocí EST a ECT a informaci o tom, v kolika příkladech nalezla daná heuristika lepší řešení a v kolika příkladech jsou výsledky heuristik totožné. Skript *stat.m* opakovaně volá funkci *stat_row.m* pro každý ze tří rozsahů okamžiku disponibility úloh.

5.3. Nalezení optimálního řešení

Vzhledem k faktu, že mnou řešená optimalizační úloha spadá do třídy *NP - těžkých* úloh, je nalezení optimálního řešení velmi obtížný úkol, zejména pak pro instance o větším počtu úloh.

Možným řešením by mohlo být využití metody větví a mezí (*branch & bound*). Ta je však náročná na implementaci a vzhledem k tomu, že hledání optimálního řešení nemá být náplní mé práce, bylo od této metody upuštěno.

V mé práci využijeme k nalezení optimálního řešení nástroj *OPL Studio*, který pracuje na principu metody lineárního programování. Za tímto účelem jsem jazyce OPL implementoval model *evytahy.mod*, hledající optimální řešení problému $\mathbf{P \mid rj \mid \sum w_j c_j}$ a skript *evytahy.osc*, jenž pomocí tohoto modelu zpracuje všechna uložená data z předchozího generování statistiky v prostředí *MATLAB* a výsledky uloží do textového souboru.

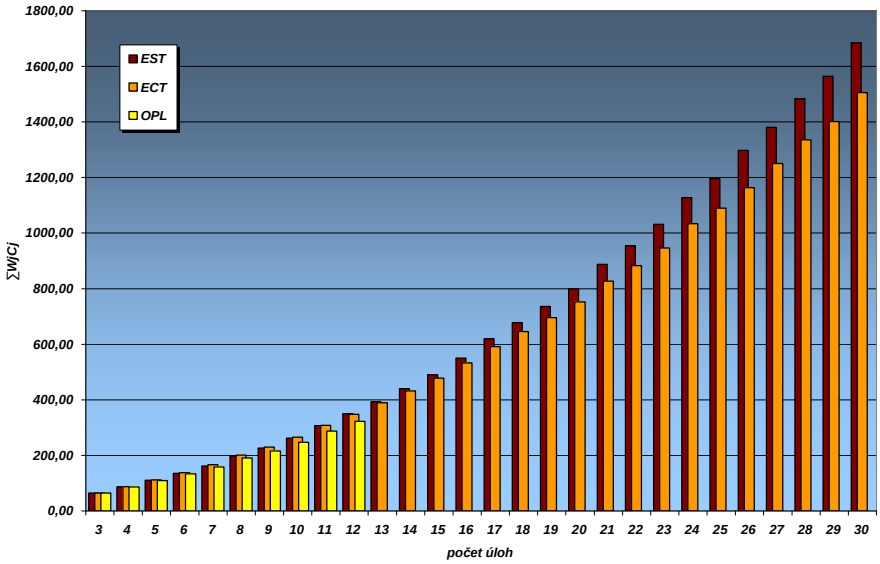
Maximální doba výpočtu jedné instance byla omezena na 20 sekund. Po uplynutí tohoto časového limitu je výpočet přerušen, výsledek je označen za neplatný a program pokračuje ve výpočtu následující instance.

5.4. Výsledky experimentů

V této podkapitole analyzujeme výsledky uvedených algoritmů prezentované formou grafů. Všechny naměřené hodnoty jsou v číselné formě uvedeny v tabulkách viz. *B. Příloha*.

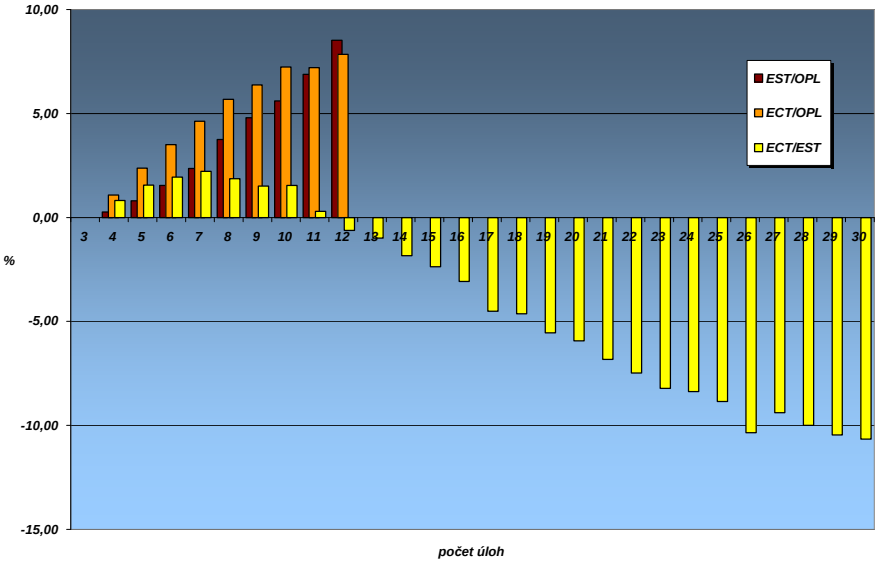
5.4.1. Porovnání algoritmů EST a ECT pro $r_j \in \langle 0,10 \rangle$

Tato situace simuluje provoz, ve kterém všechny nákladní vozidla přijíždějí k výtahům ve velice úzkém časovém rámci vzhledem k době trvání jedné úlohy. Výsledný rozvrh tedy vykazuje vysokou hustotu úloh. Následující graf znázorňuje průměrnou hodnotu kritéria $\sum w_j c_j$ v závislosti na počtu úloh v seznamu.



obr. 5.1: hodnota kritéria $\sum w_j c_j$ pro $r_j \in \langle 0,10 \rangle$

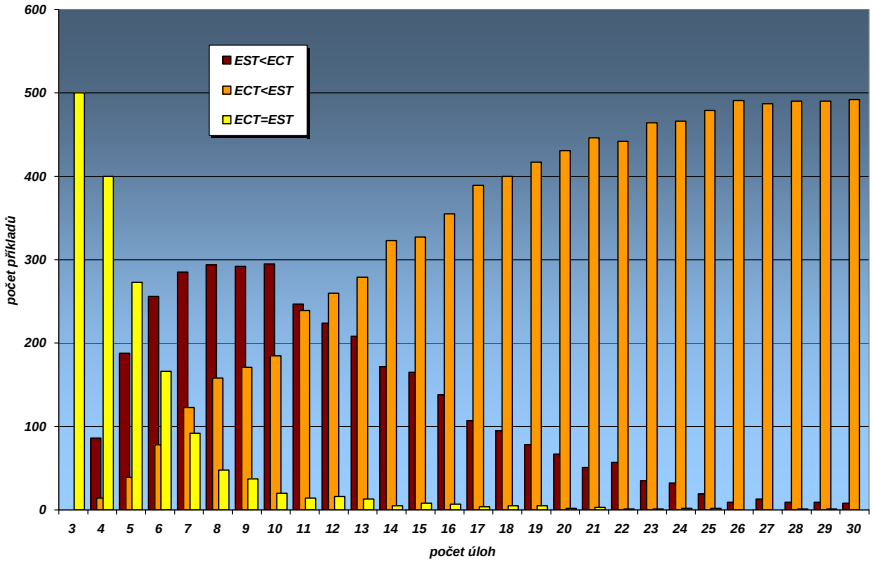
Pro názornost uvedeme tyto výsledky také jako procentuální rozdíly mezi heuristikami EST, ECT a optimálním řešením podle OPL.



obr. 5.2: vzdálenosti řešení algoritmů pro $r_j \in \langle 0,10 \rangle$

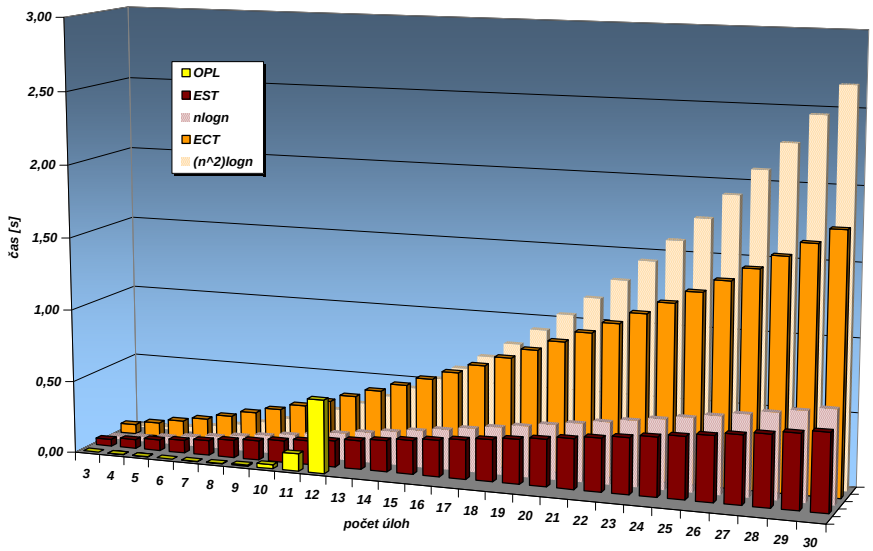
Z předchozích dvou grafů je zřejmé, že pro malý počet úloh (n = 3 až n = 11) dosahuje heuristika EST jen nepatrně lepších výsledků než ECT, maximálně o 2,21%. S rostoucím počtem úloh a tedy i hustotou rozvrhu se však heuristika ECT stává úspěšnější a to až o 10,65%. Tento zlom nastává při dvanácti úlohách v seznamu. Průměrná vzdálenost výsledků heuristiky ECT od výsledků EST je -3,87%. Průměrná vzdálenost výsledků heuristik od optimálního řešení je 3,45% pro EST a 4,60% pro ECT.

Na následujícím grafu je znázorněn počet příkladů, ve kterých heuristika EST resp. ECT nalezla lepší řešení než ECT resp. EST a počet příkladů, ve kterých obě heuristiky dospěly ke shodným výsledkům.



obr. 5.3: úspěšnost algoritmů pro $r_j \in \langle 0,10 \rangle$

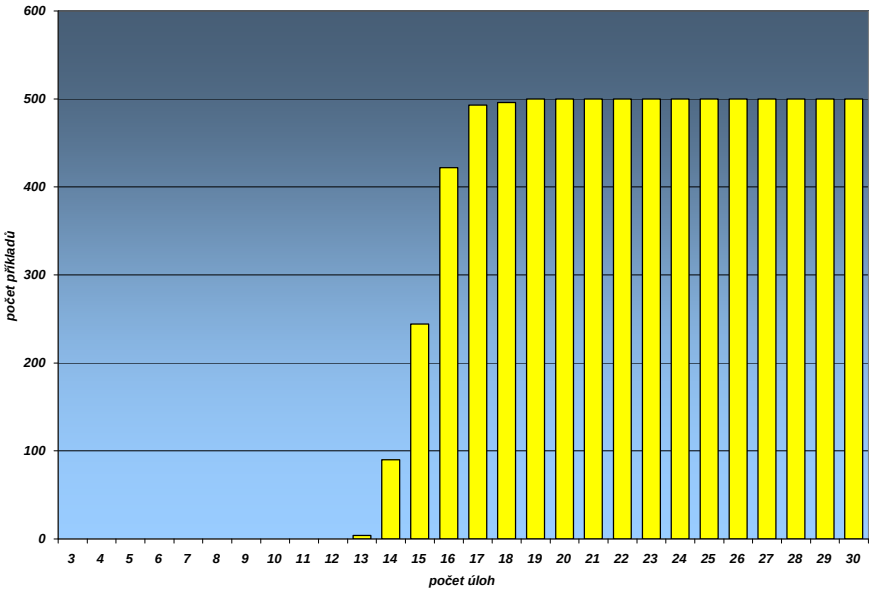
Pro zcela triviální příklad tří úloh v rozvrhu se výsledky obou heuristik nevyhnutelně shodují ve všech příkladech. S rostoucím počtem úloh v rozvrhu však počet příkladů se shodným výsledkem prudce klesá a pro více jak deset úloh jsou již obě heuristiky výrazně diferencované. Následující graf představuje závislost průměrné výpočetní doby algoritmů na počtu úloh v seznamu.



obr. 5.4: časová náročnost algoritmů pro $r_j \in \langle 0,10 \rangle$

V souladu s teoretickými vlastnostmi heuristiky EST roste její průměrná výpočetní doba pomaleji, než asymptotická časová složitost $n \log n$. Rovněž pro heuristiku ECT byla dodržena horní hranice asymptotické časové složitosti $n^2 \log n$.

Zajímavostí je zde průměrná výpočetní doba nalezení optimálního řešení pomocí nástroje OPL. Pro 12 úloh je její hodnota 0,5s, tedy hluboko pod stanoveným časovým limitem dvaceti sekund. Naproti tomu pro 13 úloh byl již tento časový limit pro některé z příkladů nedostačující. Počet příkladů, jenž nebylo možné vyřešit v časovém limitu v závislosti na počtu úloh demonstruje následující graf.

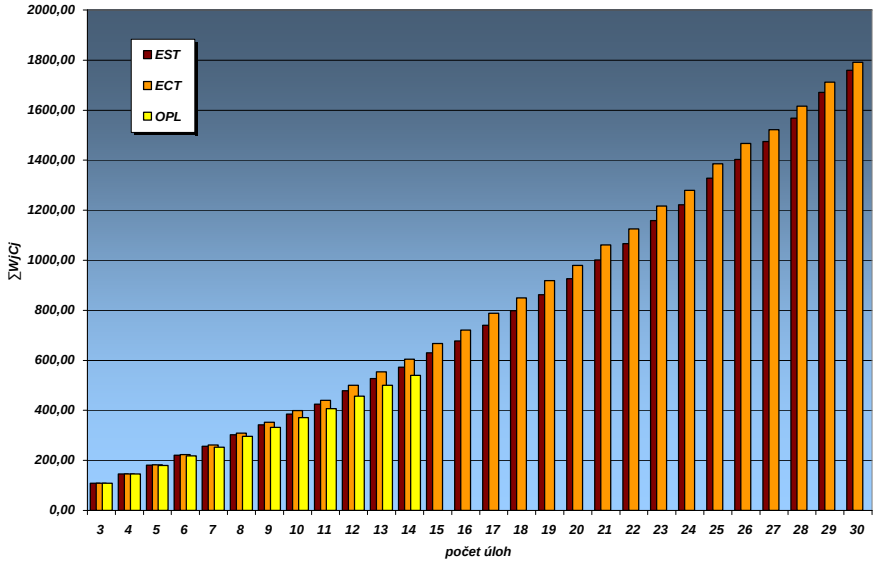


obr. 5.5: neúspěšnost nástroje OPL

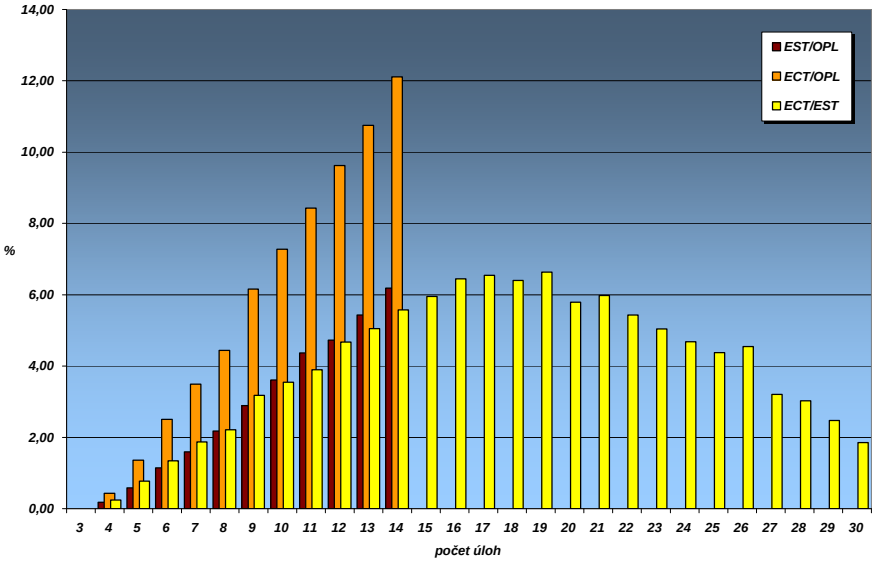
Časová složitost použitého algoritmu narůstá značnou rychlostí. Z praktických pokusů navíc vyplývá, že výpočetní doba tohoto algoritmu není závislá pouze na počtu úloh v seznamu, ale i na formální složitosti řešení dané instance a absolutní hodnotě parametrů úloh v instanci. Výpočetní doba tohoto algoritmu je tedy dána zejména počtem iterací nutných k nalezení optimálního řešení.

5.4.2. Porovnání algoritmů EST a ECT pro $r_j \in \langle 0,25 \rangle$

V tomto experimentu rozšíříme původní rozsah okamžiků disponibility úloh z $r_j \in \langle 0,10 \rangle$ na $r_j \in \langle 0,25 \rangle$. Simulujeme tedy provoz, ve kterém nákladní vozy mohou k výtahům přijíždět v širším časovém horizontu a snižujeme tím hustotu výsledného rozvrhu. Následující graf znázorňuje průměrnou hodnotu kritéria $\sum w_j c_j$ v závislosti na počtu úloh v seznamu.



obr. 5.6: hodnota kritéria $\sum w_j c_j$ pro $r_j \in \langle 0,25 \rangle$

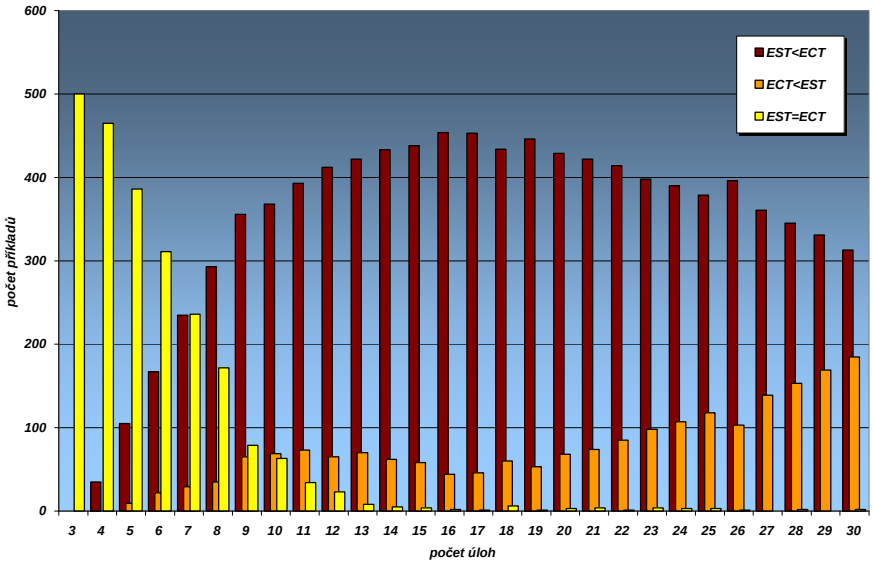


obr. 5.7: vzdálenosti řešení algoritmů pro $r_j \in \langle 0,25 \rangle$

Průměrná vzdálenost výsledků heuristiky ECT od výsledků EST je 3,96%. Průměrná vzdálenost výsledků heuristik od optimálního řešení pak 2,72% pro EST a 5,55% pro ECT.

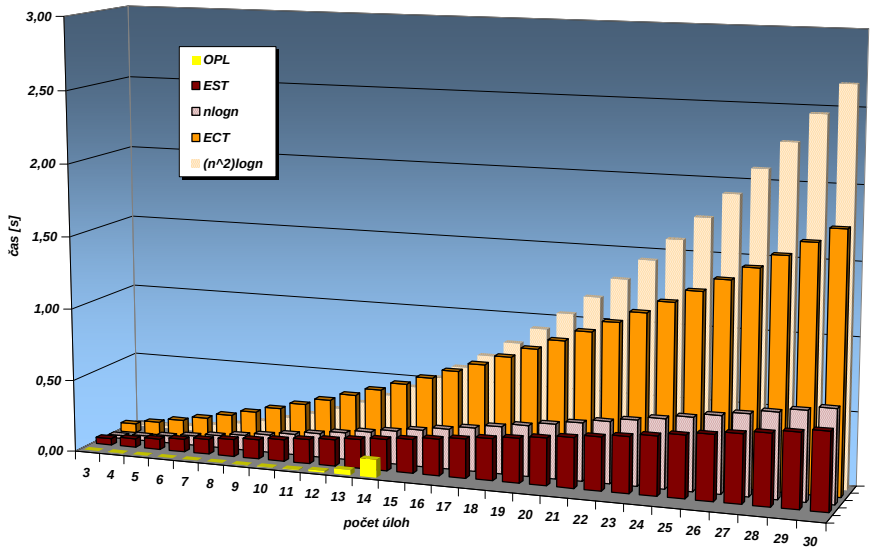
Rozšířením rozsahu, z něhož jsou vybírány okamžiky disponibility jednotlivých úloh jsme snížili hustotu výsledného rozvrhu. V takovém případě jsou výsledky heuristiky EST lepší, než výsledky ECT. S rostoucím počtem úloh však hustotu rozvrhu opět zvyšujeme a procentuální rozdíl mezi výsledky heuristik klesá. Je pravděpodobné, že pokud bychom dále zvyšovali počet úloh v seznamu, vykazovala by heuristika ECT lepší výsledky než EST podobně jako v předchozím experimentu.

Tuto domněnku potvrzuje i následující graf, který znázorňuje závislost úspěšnosti heuristik v pěti stech příkladech na počtu úloh.



obr. 5.8: úspěšnost algoritmů pro $r_j \in \langle 0,25 \rangle$

Je patrné, že pro větší počet úloh v seznamu má úspěšnost heuristiky ECT rostoucí a EST klesající tendenci.

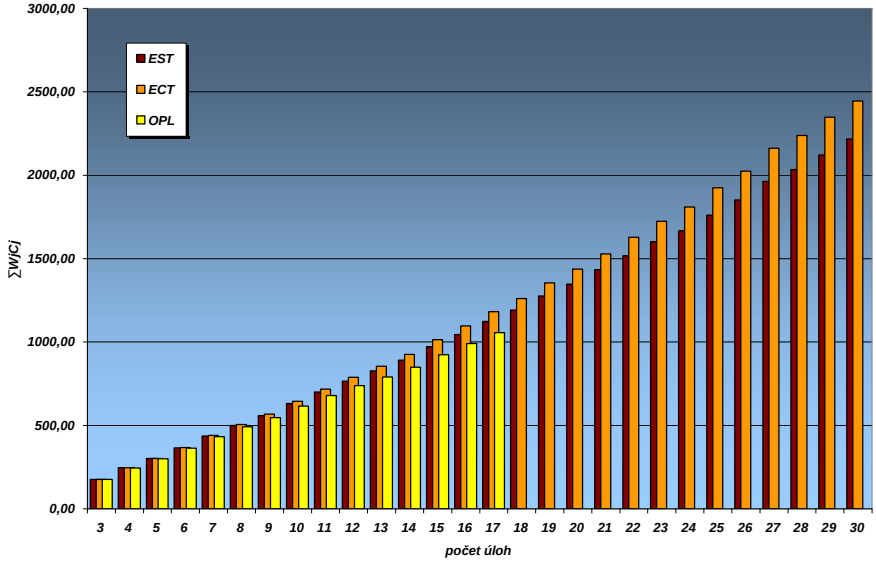


obr. 5.9: časová náročnost algoritmů pro $r_j \in \langle 0,25 \rangle$

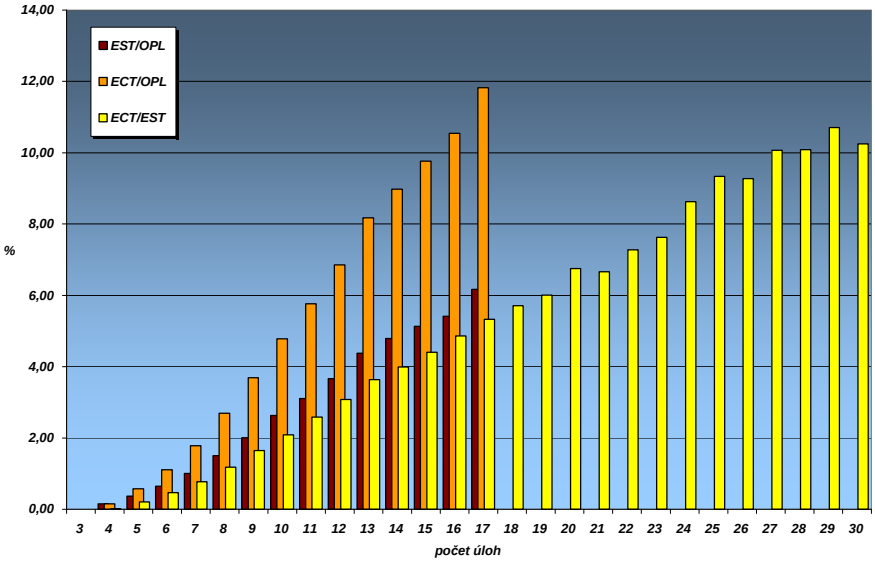
Stejně jako v předchozím experimentu jsou pro obě heuristiky splněny teoretické předpoklady asymptotické časové složitosti. Snížením hustoty rozvrhu jsme však snížili i počet iterací algoritmu OPL nutných k nalezení optimálního řešení. Díky tomuto faktu bylo OPL schopno vyřešit všech 500 příkladů až pro 14 úloh v seznamu a to dokonce s nižší průměrnou dobou výpočtu (120ms), než v předchozím experimentu pro 12 úloh v seznamu. Získali jsme tak platné hodnoty optimálního řešení až pro 14 úloh.

5.4.3. Porovnání algoritmů EST a ECT pro $r_j \in \langle 0{,}50 \rangle$

V posledním experimentu rozšíříme původní rozsah okamžiků disponibility úloh z $r_j \in \langle 0{,}10 \rangle$ na $r_j \in \langle 0{,}50 \rangle$. Simulujeme tedy provoz, ve kterém mohou nákladní vozy k výtahům přijíždět v širokém časovém horizontu vzhledem k době vykonávání úloh a nadále tím snižujeme hustotu výsledného rozvrhu. Následující graf znázorňuje průměrnou hodnotu kritéria $\sum w_j c_j$ v závislosti na počtu úloh v seznamu.



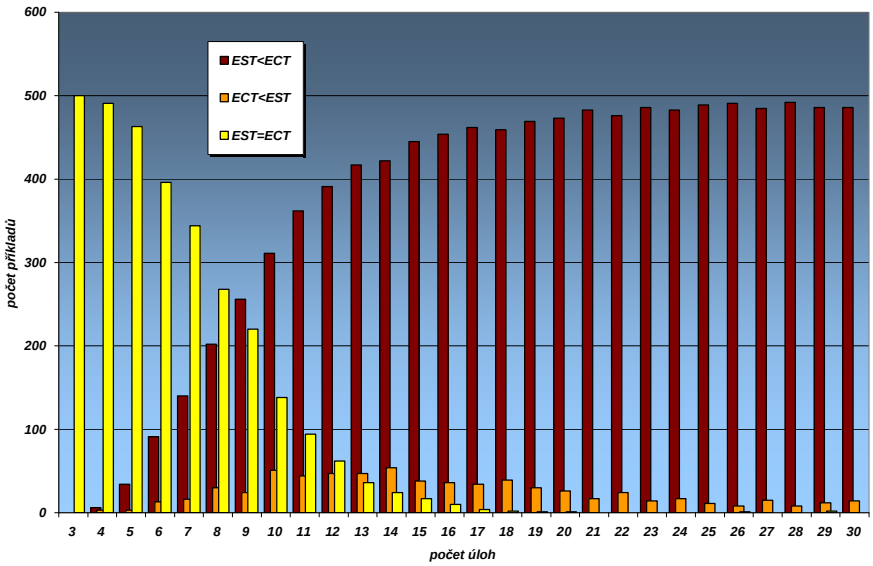
obr. 5.10: hodnota kritéria $\sum w_j c_j$ pro $r_j \in \langle 0{,}50 \rangle$



obr. 5.11: vzdálenosti řešení algoritmů pro $r_j \in \langle 0{,}50 \rangle$

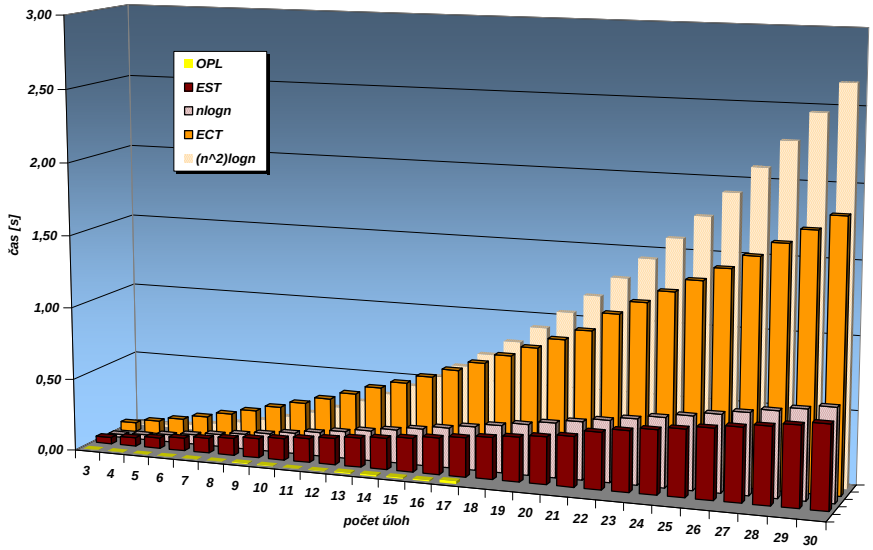
Průměrná vzdálenost výsledků heuristiky ECT od výsledků EST je 5,09%. Průměrná vzdálenost výsledků heuristik od optimálního řešení pak 2,73% pro EST a 5,11% pro ECT.

Tento experiment jen potvrzuje výsledky obou předchozích. Oproti předchozímu experimentu jsme snížili hustotu rozvrhu na polovinu a obdrželi jsme výsledky zhruba odpovídající předchozímu experimentu pro poloviční počet úloh.



obr. 5.12: úspěšnost algoritmů pro $r_j \in \langle 0,50 \rangle$

Z průběhů úspěšnosti heuristik vyplývá, že EST nalezla lepší řešení než ECT v drtivé většině příkladů.



obr. 5.13: časová náročnost algoritmů pro $r_j \in \langle 0,50 \rangle$

Výpočetní doby obou heuristik jsou totožné ve všech experimentech, již z principu těchto algoritmů závisejí pouze na počtu úloh. Dalším snížením hustoty rozvrhu jsme pomocí OPL obdrželi platné výsledky až pro 17 úloh v seznamu.

5.5. Vyhodnocení experimentů

Z provedených experimentů je zřejmé, že ve většině případů dosahuje heuristika EST lepších výsledků než ECT. Výjimkou je pouze první experiment s vysokou hustotou rozvrhu, kde ECT dosahuje pro větší počet úloh lepších výsledků. Průměrně se ovšem výsledky obou heuristik v jednotlivých experimentech neliší o více jak 5,09%. Další výhodou heuristiky EST je znatelně menší výpočetní doba, než výpočetní doba ECT. Maximální průměrná vzdálenost řešení EST od optimálního řešení je 3,45%, pro ECT pak 5,11%. Výsledky obou heuristik jsou tedy velmi blízké optimálním řešením.

Z výše uvedeného vyplývá, že heuristika ECT je vhodná pro řešení problémů velmi vysokou hustotou úloh ve výsledném rozvrhu, zejména pak vlivem úzkého intervalu disponibility úloh vůči průměrné době vykonávání. V případech vyšší hustoty úloh v rozvrhu, zapříčiněné jejich velkým počtem je třeba zvážit, zda se v dané situaci vyplatí nalezení lepšího řešení pomocí ECT za cenu delší výpočetní doby než s algoritmem EST. Algoritmus EST je vhodný pro řešení problémů s nižší hustotou úloh v rozvrhu nebo v případech kdy požadujeme nízkou výpočetní dobu.

Všechny výpočty byly provedeny na počítači, zapůjčeném katedrou řídicí techniky, s procesorem *Intel Pentium 4* – 3GHz s 2GB operační paměti RAM pod operačním systémem *Microsoft Windows Server 2003 Standard Edition – SP1*. První část výpočtu statistiky, prováděná v prostředí *MathWorks Matlab 7.0.1.*, měla celkovou výpočetní dobu 14 hodin 20 minut. Druhá část, prováděná v prostředí *OPL Studio 3.6* pak měla celkovou výpočetní dobu 4 dny.

6. Grafické uživatelské rozhraní

Abychom umožnili využití námi implementovaných heuristických algoritmů v praxi, je nutné rovněž implementovat dostatečně přehledné grafické uživatelské rozhraní (*GUI – Graphical User Interface*), které bude splňovat konkrétní požadavky na řízení nákladních výtahů.

To znamená, že musíme uživateli umožnit zadat libovolný počet nákladních vozidel (úloh) spolu s parametry jako název zakázky, okamžik příjezdu vozidla (okamžik disponibility úlohy), dobu přepravy nákladu vozidla výtahem (dobu vykonávání úlohy) a prioritu nákladu (prioritu úlohy) s ohledem na dobu exspirace zboží v zakázce.

Tím uživatel definuje vzniklou situaci (problém), kterou chce řešit. Zbývá definovat počet výtahů (procesorů) jimiž uživatel v dané chvíli disponuje a algoritmus, jímž chce realizovat výpočet rozvrhu s minimalizací průměrné doby odbavení nákladu (praktický význam minimalizace kritéria $\sum w_j c_j$). Výsledný rozvrh pak bude přehlednou formou prezentován uživateli.

6.1. Požadavky

Úkolem bylo vytvořit demonstrativní rozhraní pro *Scheduling Toolbox*, orientované na problematiku provozu nákladních výtahů a využívající mnou implementované algoritmy. Požadavkem bylo intuitivní ovládání, zpětná editace souboru úloh, možnost vracet se o několik kroků zpět v historii událostí a úzká spolupráce s pracovním prostředím *MATLAB workspace* v podobě importování a exportování souborů úloh. Dalším požadavkem pak byla rozšiřitelnost rozhraní vzhledem k dalším možnostem *Scheduling Toolboxu*, jako například možnost později vázat úlohy precedenčními závislostmi atd.

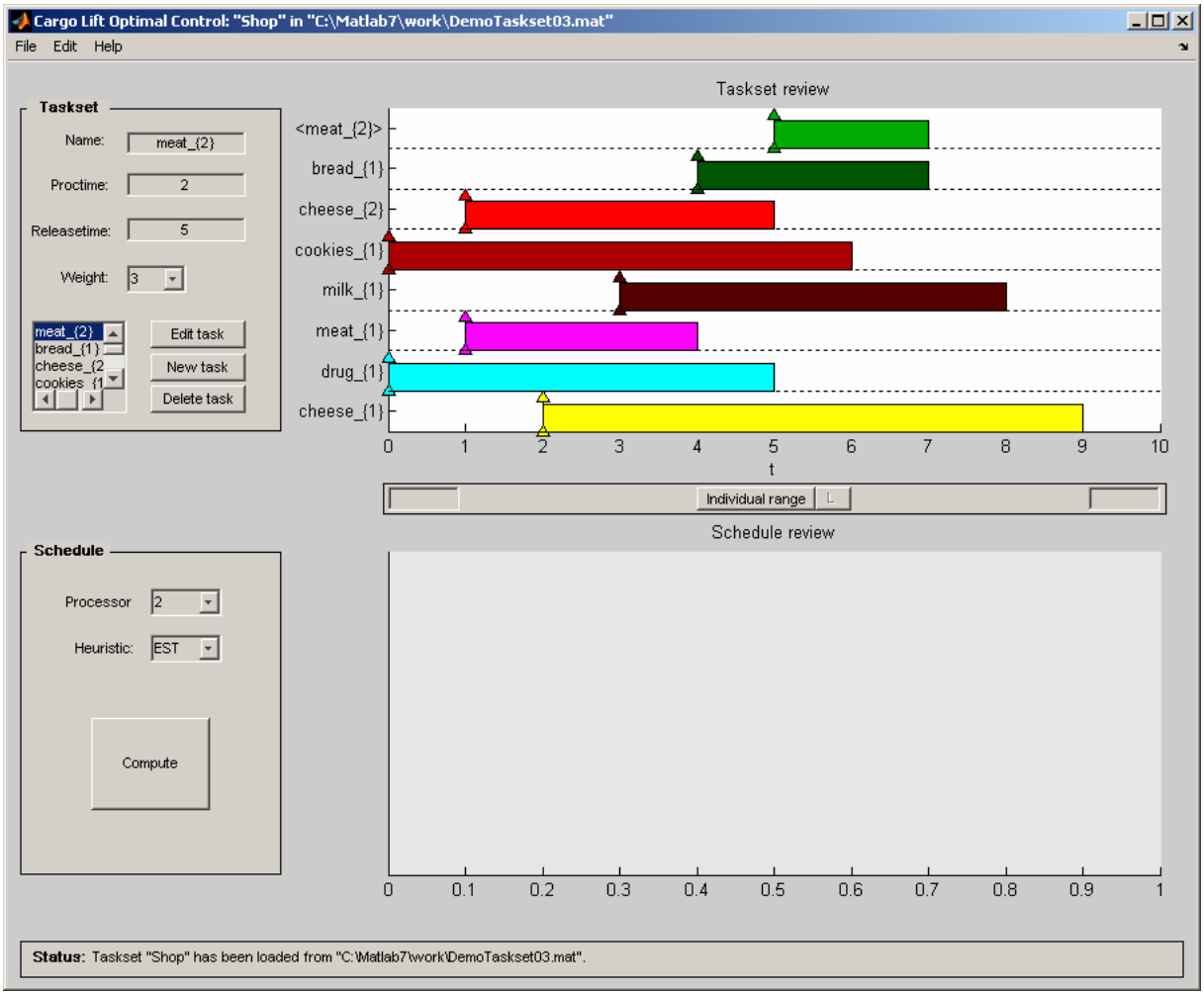
6.2. Implementace

Grafické uživatelské rozhraní jsem implementoval v prostředí *MATLAB* metodou *Switched Board Programming*, podrobně popsanou v [Zaplatílek]. Jedná se o metodu, kdy je celé uživatelské rozhraní implementováno jedinou funkcí *elevators.m*, která na základě událostí volá s odpovídajícími parametry sama sebe a reaguje tak na konkrétní akce uživatele.

Rychlost rozhraní byla zvýšena převodem *elevators.m* (m-file) na *elevators.p* (p-file) pomocí funkce *MATLABu pcode*. Tato funkce je schopna vytvořit ze standardního matlab kódu pseudokód, jehož doba vykonávání je zejména u grafických uživatelských rozhraní výrazně kratší.

6.3. Vlastnosti

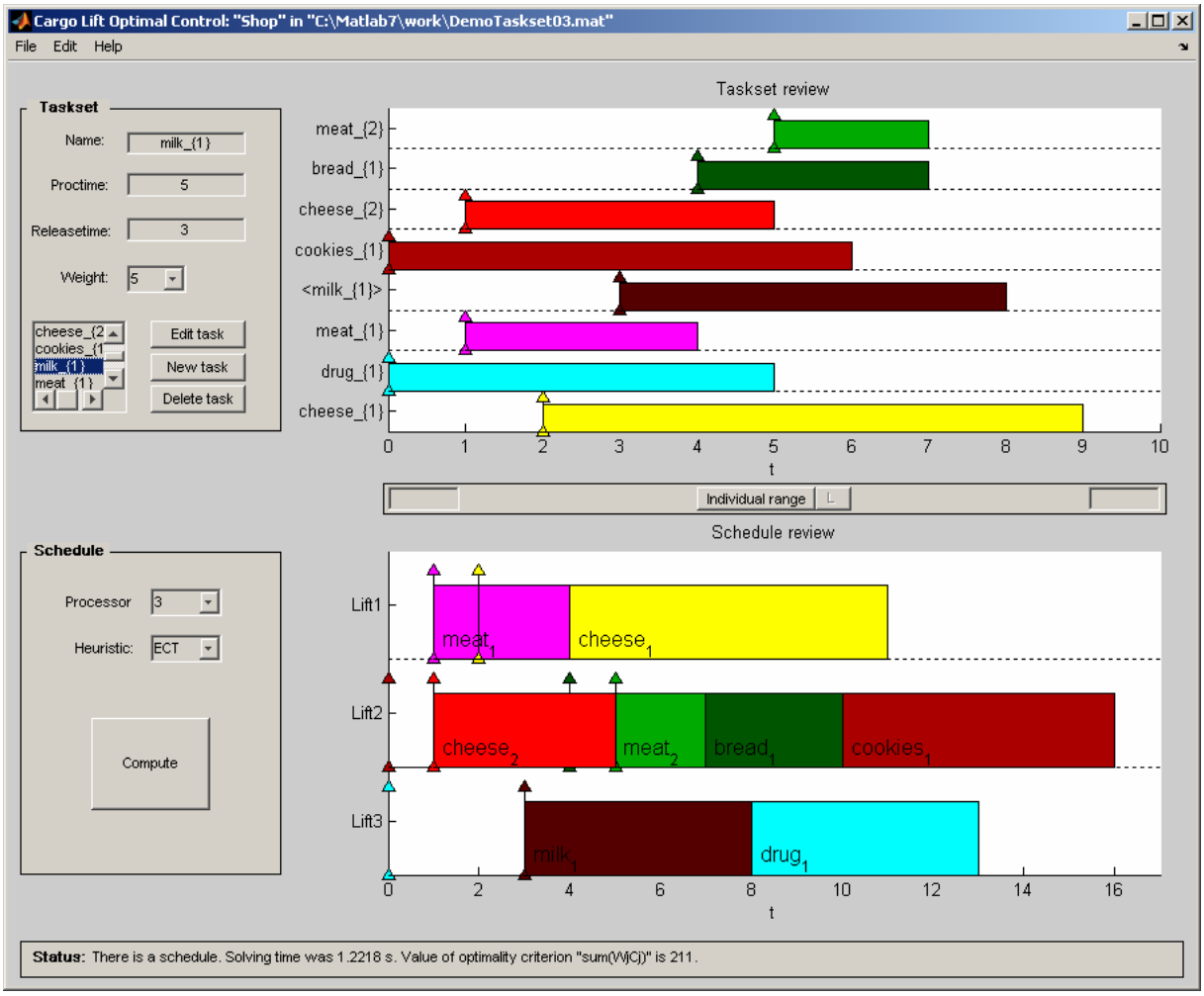
Na následujícím obrázku je znázorněno grafické uživatelské rozhraní s již nadefinovaným demonstrativním souborem úloh.



obr. 6.1: GUI, soubor úloh

Stiskem tlačítka *New task* je vytvořena nová úloha, jejíž parametry jsou nastaveny na předem definované hodnoty[7]. Tlačítko *Edit task* slouží k editaci parametrů právě vybrané úlohy. Výběr úlohy se provádí prostým poklepáním na název úlohy ve výpisu úloh. Jméno vybrané úlohy je poté na ose Y okna *Taskset review* ohraničeno znaky „<“ a „>“. Vybranou úlohu lze rovněž vymazat ze seznamu stiskem *Delete task*. Názvy úloh jsou programem automaticky indexovány[8] pro případ výskytu jednoho jména vícekrát. Výsledný soubor úloh je graficky znázorněn v okně *Taskset review*.

[7] Jméno úlohy = „task“, doba vykonávání úlohy (Proctime) = 0, okamžik disponibility úlohy (ReleaseTime) = 0, priorita úlohy (Weight) = 1.
[8] Požadavkem na tuto indexaci bylo dodržení formátu, který se používá ve standardu *LaTeX*, tedy ve formě „jméno_{index}“.



obr. 6.2: GUI, výsledný rozvrh

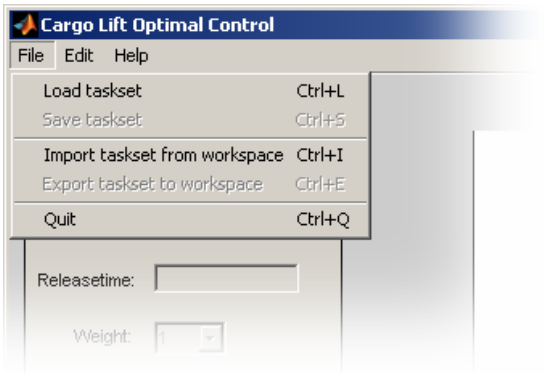
Po výběru počtu[9] procesorů, heuristického algoritmu a stisku tlačítka *Compute* je vypočten rozvrh zadaných úloh, který je zobrazen formou Ganttova diagramu v okně *Schedule review*. Situace je znázorněna na obr. 6.2.

Stavový řádek *Status* informuje v každé situaci uživatele o právě provedené akci nebo o důvodech, proč požadovanou akci nebylo možné provést. Po výpočtu rozvrhu také zobrazí celkovou dobu výpočtu a výslednou hodnotu kritéria $\sum w_j c_j$.

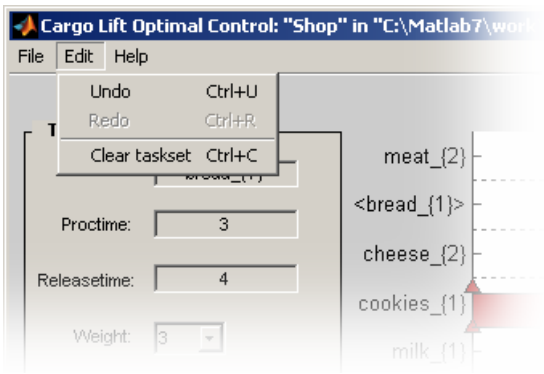
Pokud výsledný rozvrh v daném okamžiku neodpovídá zadanému souboru úloh, např. vlivem změny parametrů některé z úloh, změní se barva pozadí okna *Schedule review* z bílé na šedivou.

Obě okna, jak *Taskset review* tak *Schedule review* podporují funkci *zoom* pomocí pohybu myši a to jak v ose X tak Y. Dále je možné permanentně nastavit rozsah osy X u okna *Taskset review* stiskem *Individual range* a zadáním dolní a horní meze rozsahu. Stiskem tlačítka *L* je pak rozsah osy X okna *Schedule review* pevně svázán s rozsahem osy X okna *Taskset review*. Na následujících obrázcích jsou uvedena jednotlivá menu implementovaného rozhraní.

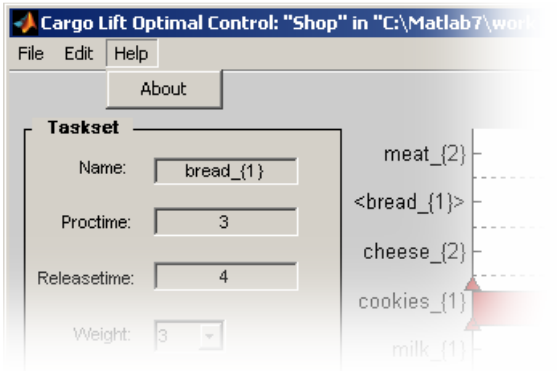
[9] Počet procesorů byl omezen na jeden až pět procesorů.



obr. 6.3: GUI, menu *File*



obr. 6.4: GUI, menu *Edit*



obr. 6.5: GUI, menu *Help*

V menu *File* lze importovat již vytvořený seznam úloh z prostředí *MATLAB workspace* pomocí příkazu *Import taskset from workspace* nebo načíst seznam úloh uložený na disku v datovém souboru typu **.mat* pomocí příkazu *Load taskset*. V jednom datovém souboru se může nacházet více proměnných typu *taskset*. Rozhraní umožňuje uživateli zvolit, která proměnná má být z datového souboru načtena a podporuje rovněž uložení více proměnných do jednoho souboru.

Pokud soubor úloh není v daném okamžiku prázdný, je možné ho obdobným způsobem rovněž ukládat na disk či exportovat do prostředí *MATLAB workspace* jako proměnnou libovolného jména.

Pomocí příkazů *Undo* a *Redo* v menu *Edit* se lze pohybovat zpět resp. vpřed v historii akcí, pokud je to zrovna možné[10]. Počet kroků historie, jenž si program pamatuje je pevně omezen na 20 akcí. V tomto menu je též možné vymazat celý soubor úloh příkazem *Clear taskset*. Menu *Help* obsahuje položku *About*, ve které jsou uvedeny základní informace o programu včetně aktuální verze a datumu posledních úprav.

[10] Po načtení nového *taskset*u z datového souboru je smazána historie akcí, příslušející *taskset*u předchozímu.

7. Závěr

Tato práce popisuje optimalizaci řízení nákladních výtahů v průmyslovém provozu pomocí vybraných suboptimálních algoritmů. Konkrétně se zabývá problémem přepravy nákladů s různými prioritami a okamžiky disponibility na obecném počtu výtahů ekvivalentních rozměrů a rychlostí s minimalizací průměrné doby odbavení nákladu.

Terminologií teorie rozvrhování jsme tedy problém popsali jako statické rozvrhování úloh s nestejnými okamžiky disponibility a prioritami na identické paralelní procesory. Tento problém lze ve standardní notaci označit jako $\mathbf{P \mid r_j \mid \sum w_j c_j}$.

Z existujících algoritmů byly vybrány dva heuristické algoritmy EST a ECT, vhodné pro řešení tohoto problému. Ty byly spolu se základním kombinatorickým algoritmem *List Scheduling*, nezbytným pro jejich funkci, implementovány v prostředí *MATLAB* takovým způsobem, aby vhodně rozšířily již existující *TORSCHE: Scheduling Toolbox*, vyvíjený na katedře řídicí techniky.

Dále bylo provedeno statistické porovnání výsledků a časové náročnosti uvedených algoritmů vzhledem k proměnné hustotě úloh ve výsledném rozvrhu na dostatečném počtu příkladů. Optimální řešení příkladů, sloužící k porovnání s výsledky heuristických algoritmů, byla hledána pomocí nástroje *OPL Studio*.

Z tohoto porovnání vyplynulo, že algoritmus ECT je vhodný pro řešení instancí s vysokou hustotou rozvrhu, ať již vlivem velkého počtu úloh nebo úzkého intervalu možných okamžiků disponibility úloh. Algoritmus EST je naopak vhodný pro řešení instancí s malou hustotou úloh a vyznačuje se výrazně menší časovou náročností než ECT. Obě heuristiky pak vykazovaly velmi malou vzdálenost od optimálních řešení v řádu jednotek procent.

Vezmeme-li v úvahu jednoduchost algoritmu EST, jeho snadnou implementaci, příznivou časovou náročnost a hlavně intuitivní princip, který by byl v praktickém provozu nejspíše realizován i bez teoretické studie, lze až na výjimečné situace s vysokou hustotou úloh tento algoritmus upřednostnit před algoritmem ECT.

Poslední bodem práce byla implementace grafického uživatelského rozhraní pro uvedené algoritmy v prostředí *MATLAB*, určené pro demonstraci schopnosti *Scheduling Toolbox*u řešit reálné problémy rozvrhování v praxi.

V závěrečné fázi práce byl algoritmus *List Scheduling* a obě heuristiky rozšířeny o možnost rozvrhování na uniformní i nesouvztažné paralelní procesory. Všechny zmíněné algoritmy jsou pak také schopny pracovat s omezením precedenčními závislostmi mezi úlohami, čehož v této práci nebylo využito.

8. Seznam použitých zdrojů

[Błażewicz]
BLAZEWICZ, J. aj. *Scheduling in Computer and Manufacturing Systéme*. Springer – Verlag. Berlin, 1996.

[Boeres]
BOERES, C. – CHOCHIA, G. – THANISCH, P. *On the Scope of Applicability of the ETF Algorithm*. Scotland: The University of Edinburgh, August 1995.

[Brucker]
BRUCKER, P. *Scheduling Algorithms*. Fourth Edition, Berlin: Springer–Verlag, 2004. ISBN 3-540-20524-1.

[Bruno]
BRUNO, J. – COFFMAN, E.G. Jr. – SETHI, R. *Scheduling Independent Tasks To Reduce Mean Finishing Time*. Communications of the ACM. 17:382--387, 1974.

[Glasgow]
GLASGOW, J. – SCHACHNAI, H. *Minimizing the Flow Time for Parallelizable Task Systems*. Technical Report TR790. Technion IIT, November 1993.

[Graham]
GRAHAM, R.L. *Bounds for certain multiprocessing anomalies*. Bell System Tech. J. 45, 1966.

[Leung]
LEUNG, J. – ANDERSON, J. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis.*

[Mařík]
MAŘÍK, V. – ŠTĚPÁNKOVÁ, O. – LAŽANSKÝ, J. *Umělá inteligence (3).* 1. vydání. Praha: Academia, 2001. ISBN 80-200-0472-6.

[Matlab Documentation]
THE MATHWORKS. *Documentation: Matlab* [online]. [cit. 26.12.2005]. <http://www.mathworks.com/access/helpdesk/help/techdoc/matlab.shtml>.

[Matlab NewsGroup]
THE MATHWORKS. *Matlab Central: comp.soft-sys.matlab* [online]. <http://newsreader.mathworks.com/WebX?14@272.pp1zbarYgrP.0@/comp.soft-sys.matlab>.

[Pinedo]
PINEDO, M. *Scheduling Theory, Algorithms, and Systems*. Second Edition. New Jersey: Prentice Hall, 2002. ISBN 0-13-028138-7.

[RDU]
HANZÁLEK, Z. *Přednášky k předmětu 35RDU*. ČVUT - FEL. Praha, 2005.

[Scheduling]
ČVUT – FEL DCE. *Scheduling Toolbox* [online]. [cit. 5.1.2006]. ČVUT – FEL. Praha, 2005. < http://rtime.felk.cvut.cz/scheduling-toolbox/>.

[Zaplatílek]
ZAPLATÍLEK, K. – DOŇAR, B. *MATLAB – tvorba uživatelských aplikací*. 1. vydání. Praha: BEN, 2004. 216 s. ISBN 80-7300-133-0.

[Brucker&Knust]
BRUCKER, P. – KNUST, S. *Complexity results for scheduling problems* [online]. [cit. 15.12.2005].
<http://www.mathematik.uni-osnabrueck.de/research/OR/class/>.

A. Příloha – odvození asymptotické časové složitosti algoritmu ECT

Z principu algoritmu ECT vyplývá, že v iteraci *i* algoritmu *List Scheduling* se provádí operace třídění vždy (*n – i)* prvků. Kde *n* je aktuální počet úloh v seznamu. Odvození asymptotické časové složitosti algoritmu ECT tedy vychází z časové složitosti problému třídění *n* prvků. *Každý algoritmus, který ke třídění používá operaci porovnávání, má časovou složitost O(nlogn)*. [Mařík] Pokud by algoritmus ECT v každé iteraci třídil celou množinu úloh, měl by tedy časovou složitost:

$$O(ECT') = i * (n \log n).$$

Pro naše účely bude stačit tento, poměrně hrubý odhad časové složitosti algoritmu ECT'. Celkový počet iterací algoritmu ECT' v každé iteraci algoritmu *List Scheduling* tedy je roven počátečnímu počtu úloh v seznamu a jeho časovou složitost lze zapsat jako:

$$O(ECT') \leq n^2 \log n.$$

Pokud bychom trvali na přesném vyjádření asymptotické časové složitosti algoritmu ECT, mohli bychom ji vyjádřit součtem řady:

$$O(ECT) = n \log n + (n-1)\log(n-1) + (n-2)\log(n-2) + ... + (n-(n-1))\log(n-(n-1)).$$

Z výše uvedených vztahů je zřejmé, že platí:

$$O(ECT) \leq O(ECT').$$

B. Příloha – Výsledky statistického porovnání algoritmů

zkratka	význam
∑WjCj [-]	průměrná hodnota kritéria $\sum w_j c_j$ na pěti stech příkladech
čas [s]	průměrný výpočetní čas použitého algoritmu
EST<ECT	počet příkladů, ve kterých nalezl algoritmus EST lepší řešení než ECT
ECT<EST	počet příkladů, ve kterých nalezl algoritmus ECT lepší řešení než EST
EST=ECT	počet příkladů, ve kterých heuristiky nalezly stejné řešení
ECT/EST [%]	průměrný procentuelní rozdíl mezi výsledky ECT a EST; jako základ je brán průměrný výsledek EST
B	počet příkladů, při jejichž řešení byl překročen stanovený časový limit
EST/OPL [%]	průměrná procentuelní vzdálenost řešení EST od optimálního řešení
ECT/OPL [%]	průměrná procentuelní vzdálenost řešení ECT od optimálního řešení
nlogn	teoretická asymptotická časová složitost heuristiky EST
n^2logn	teoretická asymptotická časová složitost heuristiky ECT
-	výsledek nemohl být vypočítán

tabulka B.1: vysvětlivky pro statistické porovnání algoritmů

počet úloh	EST			ECT			EST=ECT	ECT/EST [%]	OPL			EST/OPL [%]	ECT/OPL [%]	nlogn	n^2logn
	∑WjCj [-]	čas [s]	EST<ECT	∑WjCj [-]	čas [s]	ECT<EST			∑WjCj [-]	čas [s]	B				
3	63,84	0,05	0	63,84	0,07	0	500	0,00	63,84	0,00	0	0,00	0,00	2,04E-02	8,59E-03
4	86,48	0,06	86	87,19	0,09	14	400	0,82	86,25	0,00	0	0,27	1,09	3,44E-02	1,93E-02
5	109,91	0,08	188	111,63	0,12	39	273	1,56	109,03	0,00	0	0,80	2,38	4,99E-02	3,49E-02
6	135,19	0,09	256	137,81	0,15	78	166	1,94	133,14	0,00	0	1,54	3,51	6,67E-02	5,60E-02
7	162,45	0,10	285	166,04	0,19	123	92	2,21	158,70	0,00	0	2,36	4,63	8,45E-02	8,28E-02
8	197,80	0,12	294	201,50	0,23	158	48	1,87	190,66	0,00	0	3,75	5,69	1,03E-01	1,16E-01
9	225,78	0,13	292	229,19	0,27	171	37	1,51	215,44	0,01	0	4,80	6,38	1,23E-01	1,55E-01
10	261,41	0,15	295	265,45	0,31	185	20	1,55	247,53	0,03	0	5,61	7,24	1,43E-01	2,00E-01
11	307,22	0,17	247	308,15	0,36	239	14	0,30	287,45	0,12	0	6,88	7,20	1,64E-01	2,52E-01
12	350,74	0,18	224	348,57	0,41	260	16	-0,62	323,19	0,50	0	8,52	7,85	1,85E-01	3,11E-01
13	393,02	0,20	208	389,11	0,46	279	13	-1,00	356,25	1,89	4	-	-	2,07E-01	3,77E-01
14	440,44	0,22	172	432,35	0,51	323	5	-1,84	315,62	4,31	90	-	-	2,29E-01	4,49E-01
15	489,45	0,23	165	477,86	0,57	327	8	-2,37	207,72	4,07	244	-	-	2,52E-01	5,29E-01
16	549,70	0,25	138	532,83	0,63	355	7	-3,07	65,46	1,40	422	-	-	2,75E-01	6,17E-01
17	619,21	0,27	107	591,30	0,69	389	4	-4,51	6,21	0,11	493	-	-	2,99E-01	7,11E-01
18	677,17	0,29	95	645,87	0,76	400	5	-4,62	3,41	0,09	496	-	-	3,23E-01	8,13E-01
19	736,59	0,30	78	695,70	0,83	417	5	-5,55	0	0	500	-	-	3,47E-01	9,23E-01
20	800,02	0,32	67	752,60	0,90	431	2	-5,93	0	0	500	-	-	3,72E-01	1,04E+00
21	887,75	0,34	51	827,25	0,97	446	3	-6,82	0	0	500	-	-	3,97E-01	1,17E+00
22	954,46	0,36	57	883,10	1,05	442	1	-7,48	0	0	500	-	-	4,22E-01	1,30E+00
23	1030,97	0,38	35	946,20	1,13	464	1	-8,22	0	0	500	-	-	4,47E-01	1,44E+00
24	1127,91	0,40	32	1033,45	1,21	466	2	-8,38	0	0	500	-	-	4,73E-01	1,59E+00
25	1195,34	0,42	19	1089,55	1,29	479	2	-8,85	0	0	500	-	-	4,99E-01	1,75E+00
26	1298,05	0,44	9	1163,79	1,38	491	0	-10,34	0	0	500	-	-	5,26E-01	1,91E+00
27	1380,65	0,46	13	1251,11	1,47	487	0	-9,38	0	0	500	-	-	5,52E-01	2,09E+00
28	1483,69	0,49	9	1335,38	1,57	490	1	-10,00	0	0	500	-	-	5,79E-01	2,27E+00
29	1564,61	0,51	9	1400,94	1,66	490	1	-10,46	0	0	500	-	-	6,06E-01	2,46E+00
30	1685,56	0,53	8	1505,99	1,76	492	0	-10,65	0	0	500	-	-	6,33E-01	2,66E+00
průměr	-	-	-	-	-	-	-	-3,87	-	-	-	3,45	4,60	-	-

tabulka B.2: statistické porovnání algoritmů pro $r_j \in \langle 0,10 \rangle$

počet úloh	EST			ECT			EST=ECT	ECT/EST [%]	OPL			EST/OPL [%]	ECT/OPL [%]	nlogn	n^2logn
	∑WjCj [-]	čas [s]	EST<ECT	∑WjCj [-]	čas [s]	ECT<EST			∑WjCj [-]	čas [s]	B				
3	108,40	0,04	0	108,40	0,06	0	500	0,00	108,40	0,00	0	0,00	0,00	2,04E-02	8,59E-03
4	145,68	0,06	35	146,03	0,09	0	465	0,24	145,41	0,00	0	0,18	0,43	3,44E-02	1,93E-02
5	180,90	0,07	105	182,30	0,12	9	386	0,77	179,85	0,00	0	0,58	1,36	4,99E-02	3,49E-02
6	220,63	0,09	167	223,60	0,15	22	311	1,35	218,13	0,00	0	1,15	2,51	6,67E-02	5,60E-02
7	256,29	0,10	235	261,07	0,19	29	236	1,87	252,27	0,00	0	1,59	3,49	8,45E-02	8,28E-02
8	302,43	0,12	293	309,13	0,23	35	172	2,21	295,99	0,00	0	2,18	4,44	1,03E-01	1,16E-01
9	341,48	0,13	356	352,32	0,27	65	79	3,17	331,88	0,00	0	2,89	6,16	1,23E-01	1,55E-01
10	385,21	0,15	368	398,85	0,31	69	63	3,54	371,79	0,00	0	3,61	7,28	1,43E-01	2,00E-01
11	424,13	0,17	393	440,65	0,36	73	34	3,90	406,38	0,01	0	4,37	8,43	1,64E-01	2,52E-01
12	477,69	0,18	412	500,02	0,41	65	23	4,68	456,13	0,01	0	4,72	9,62	1,85E-01	3,11E-01
13	526,60	0,20	422	553,19	0,46	70	8	5,05	499,49	0,04	0	5,43	10,75	2,07E-01	3,77E-01
14	573,23	0,22	433	605,21	0,51	62	5	5,58	539,84	0,12	0	6,18	12,11	2,29E-01	4,49E-01
15	630,19	0,23	438	667,71	0,57	58	4	5,95	587,98	0,49	3	-	-	2,52E-01	5,29E-01
16	676,72	0,25	454	720,34	0,63	44	2	6,45	616,59	1,29	12	-	-	2,75E-01	6,17E-01
17	739,83	0,27	453	788,27	0,69	46	1	6,55	608,85	2,32	54	-	-	2,99E-01	7,11E-01
18	798,29	0,29	434	849,38	0,76	60	6	6,40	469,36	2,46	173	-	-	3,23E-01	8,13E-01
19	861,69	0,30	446	918,83	0,83	53	1	6,63	333,04	2,44	276	-	-	3,47E-01	9,23E-01
20	925,94	0,32	429	979,53	0,90	68	3	5,79	173,25	1,59	391	-	-	3,72E-01	1,04E+00
21	1001,89	0,34	422	1061,84	0,97	74	4	5,98	94,14	0,72	444	-	-	3,97E-01	1,17E+00
22	1066,54	0,36	414	1124,51	1,05	85	1	5,44	27,58	0,27	483	-	-	4,22E-01	1,30E+00
23	1158,39	0,38	398	1216,75	1,13	98	4	5,04	9,35	0,11	495	-	-	4,47E-01	1,44E+00
24	1222,36	0,40	390	1279,62	1,21	107	3	4,68	2,23	0,00	499	-	-	4,73E-01	1,59E+00
25	1327,46	0,42	379	1385,57	1,29	118	3	4,38	0	0	500	-	-	4,99E-01	1,75E+00
26	1402,93	0,44	396	1466,73	1,38	103	1	4,55	0	0	500	-	-	5,26E-01	1,91E+00
27	1474,34	0,46	361	1521,61	1,47	139	0	3,21	0	0	500	-	-	5,52E-01	2,09E+00
28	1568,19	0,49	345	1615,69	1,57	153	2	3,03	0	0	500	-	-	5,79E-01	2,27E+00
29	1670,95	0,51	331	1712,25	1,66	169	0	2,47	0	0	500	-	-	6,06E-01	2,46E+00
30	1758,67	0,53	313	1791,21	1,76	185	2	1,85	0	0	500	-	-	6,33E-01	2,66E+00
průměr	-	-	-	-	-	-	-	3,96	-	-	-	2,74	5,55	-	-

tabulka B.3: statistické porovnání algoritmů pro $r_j \in \langle 0,25 \rangle$

počet úloh	EST			ECT			EST=ECT	ECT/EST [%]	OPL			EST/OPL [%]	ECT/OPL [%]	nlogn	n^2logn
	∑WjCj [-]	čas [s]	EST<ECT	∑WjCj [-]	čas [s]	ECT<EST			∑WjCj [-]	čas [s]	B				
3	175,97	0,04	0	175,97	0,06	0	500	0,00	175,97	0,00	0	0,00	0,00	2,04E-02	8,59E-03
4	246,00	0,06	6	246,02	0,09	3	491	0,01	245,64	0,00	0	0,15	0,16	3,44E-02	1,93E-02
5	301,97	0,07	34	302,59	0,12	3	463	0,20	300,86	0,00	0	0,37	0,57	4,99E-02	3,49E-02
6	365,68	0,09	91	367,37	0,15	13	396	0,46	363,35	0,00	0	0,64	1,11	6,67E-02	5,60E-02
7	436,29	0,10	140	439,64	0,19	16	344	0,77	431,96	0,00	0	1,00	1,78	8,45E-02	8,28E-02
8	499,46	0,12	202	505,34	0,23	30	268	1,18	492,06	0,00	0	1,50	2,70	1,03E-01	1,16E-01
9	558,30	0,13	256	567,50	0,27	24	220	1,65	547,33	0,01	0	2,00	3,68	1,23E-01	1,55E-01
10	631,85	0,15	311	645,06	0,31	51	138	2,09	615,62	0,01	0	2,64	4,78	1,43E-01	2,00E-01
11	699,86	0,17	362	717,95	0,36	44	94	2,58	678,79	0,00	0	3,11	5,77	1,64E-01	2,52E-01
12	764,84	0,18	391	788,41	0,41	47	62	3,08	737,81	0,00	0	3,66	6,86	1,85E-01	3,11E-01
13	825,69	0,20	417	855,70	0,47	47	36	3,63	791,07	0,01	0	4,38	8,17	2,07E-01	3,77E-01
14	890,99	0,22	422	926,59	0,51	54	24	4,00	850,27	0,01	0	4,79	8,98	2,29E-01	4,49E-01
15	972,02	0,23	445	1014,84	0,57	38	17	4,41	924,59	0,01	0	5,13	9,76	2,52E-01	5,29E-01
16	1044,55	0,25	454	1095,35	0,63	36	10	4,86	990,86	0,01	0	5,42	10,55	2,75E-01	6,17E-01
17	1122,14	0,27	462	1181,93	0,69	34	4	5,33	1056,97	0,02	0	6,17	11,82	2,99E-01	7,11E-01
18	1192,13	0,29	459	1260,18	0,76	39	2	5,71	1113,58	0,08	1	-	-	3,23E-01	8,13E-01
19	1277,02	0,30	469	1353,71	0,83	30	1	6,01	1182,10	0,10	2	-	-	3,47E-01	9,23E-01
20	1346,16	0,32	473	1437,02	0,90	26	1	6,75	1236,43	0,22	4	-	-	3,72E-01	1,04E+00
21	1433,59	0,34	483	1529,00	0,97	17	0	6,66	1291,24	0,38	11	-	-	3,97E-01	1,17E+00
22	1517,50	0,39	476	1627,87	1,10	24	0	7,27	1320,67	0,81	25	-	-	4,22E-01	1,30E+00
23	1601,52	0,41	486	1723,68	1,19	14	0	7,63	1376,57	1,20	30	-	-	4,47E-01	1,44E+00
24	1665,56	0,43	483	1809,24	1,27	17	0	8,63	1250,63	1,73	86	-	-	4,73E-01	1,59E+00
25	1760,08	0,46	489	1924,41	1,36	11	0	9,34	1087,71	1,79	154	-	-	4,99E-01	1,75E+00
26	1851,90	0,48	491	2023,64	1,45	8	1	9,27	948,55	1,87	211	-	-	5,26E-01	1,91E+00
27	1963,60	0,50	485	2161,27	1,54	15	0	10,07	693,93	1,62	298	-	-	5,52E-01	2,09E+00
28	2033,36	0,52	492	2238,44	1,63	8	0	10,09	481,86	1,59	363	-	-	5,79E-01	2,27E+00
29	2120,89	0,55	486	2348,02	1,73	12	2	10,71	295,09	1,09	419	-	-	6,06E-01	2,46E+00
30	2217,41	0,57	486	2444,64	1,83	14	0	10,25	111,99	0,43	471	-	-	6,33E-01	2,66E+00
průměr	-	-	-	-	-	-	-	5,09	-	-	-	2,73	5,11	-	-

tabulka B.4: statistické porovnání algoritmů pro $r_j \in \langle 0,50 \rangle$

C. Příloha – Přiložené CD

Součástí této práce je přiložený CD-ROM obsahující zdrojové kódy implementovaných funkcí, data vypočtená při statistickém porovnání algoritmů a tento dokument v elektronické podobě.