Řešení problému batohu metodou hrubé síly a jednoduchou heuristikou

1 SPECIFIKACE ÚLOHY

Cílem tohoto úkolu bylo naprogramovat řešení problému batohu a to hrubou silou a pomocí jednoduchých heuristik. Zpráva také obsahuje srovnání těchto algoritmů z hlediska časového výkonu a korektnosti výsledků. Více informací viz odpovídající stránka na Eduxu.

2 ROZBOR MOŽNÝCH VARIANT ŘEŠENÍ

V **heuristické části** bylo cílem volit nejdříve položky s nejlepším poměrem cena/váha, případně hladově podle nejvyšší ceny či nejnižší hmotnosti. Nejjednodušeji tohoto lze dosáhnout tím, že si prvky nejdříve seřadíme dle daného kritéria a pak už pouze vybíráme po jednom. Další řešení by bylo postupné hledání maxima a odebírání již využitých prvků.

V **exaktní části** potřebujeme vyzkoušet všechny kombinace možných předmětů. To lze udělat pomocí generátoru kombinací a následným otestováním dané kombinace. Po otestování všech kombinací a ukládání nejlepšího výsledku musíme dosáhnout optimálního řešení.

3 POPIS POSTUPU ŘEŠENÍ A ALGORITMU

Program nejdříve načte data ze vstupních souborů a namapuje data z textové podoby na interní objekty, jako jsou např. **KnapsackInstance** či **KnapsackItem**.

3.1 HRUBÁ SÍLA

0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Klíčem řešení metodou hrubé síly je generátor kombinací, který např. pro instanci problému o velikosti třech věcí vygeneruje kombinace viz tabulka vlevo.

O vyřešení konkrétního problému se stará třída **BruteKnapsackSolver**, které je předána instance problému. Ta za využití výše popsaného generátoru provede součet nad násobkem ceny s hmotností, a to postupně pro všechny možné kombinace. Pro každou kombinaci pak zkontroluje, jestli daná kombinace nepřesahuje hmotnost batohu a zda již nemáme lepší řešení. V opačném případě si kombinaci uloží, jako dosavadní nejlepší řešení. Po vyzkoušení všech kombinací máme nejlepší řešení.

3.2 HEURISTIKY

Řešení heuristikou je jednodušší na implementaci, protože nepotřebujeme generátor kombinací.

O vyřešení problému za pomocí heuristiky se stará třída **HeuristicKnapsackSolver**, které je předána heuristika ve formě komparátoru. (**Comparator** je v jazyce Java rozhranní, které se používá k řazení kolekcí, polí a dalších struktur, obsahujících porovnatelné prvky.) V programu jsem implementoval tyto heuristiky tři:

1. Řazení dle poměru cen/váha (vzestupně)
2. Řazení dle ceny (vzestupně)
3. Řazení dle váhy (sestupně)

HeuristicKnapsackSolver seřadí prvky dané instance pomocí zvolené heuristiky a následně pouze postupně přidává prvky z instance do batohu, přičemž před každým přidáním zkontroluje, jeslti se daná věc do batohu ještě vejde.

Pokud se již daná věc do batohu nevejde, mohl by zde program skončit. Pro větší přiblížení se optimu však program věc pouze přeskočí a pokusí se do batohu vložit i zbytek věcí, protože by se zde mohla nacházet věc lehčí, která se do batohu ještě vejde.

4 NAMĚŘENÉ VÝSLEDKY

4.1 HW / SW KONFIGURACE TESTOVACÍHO SYSTÉMU

- CPU Intel® Core™2 Duo; 2.26 GHz, 2.27 GHz
- 4 GB RAM
- OS Windows 8 64-bit
- Java 7

4.2 ZPŮSOB MĚŘENÍ

Čas byl měřen pomocí třídy **ThreadMXBean** a byl průměrován přes všechny instance dané velikosti. Pro úlohy příliš malé velikosti bylo měření spuštěno v cyklu vícekrát a následně vyděleno počtem běhů. Např. pro nejnižší velikost 4 jsem musel měření každé instance akumulovat na milionu měření, protože běh jednoho výpočtu trvá pouze přibližně jednu mikrosekundu.

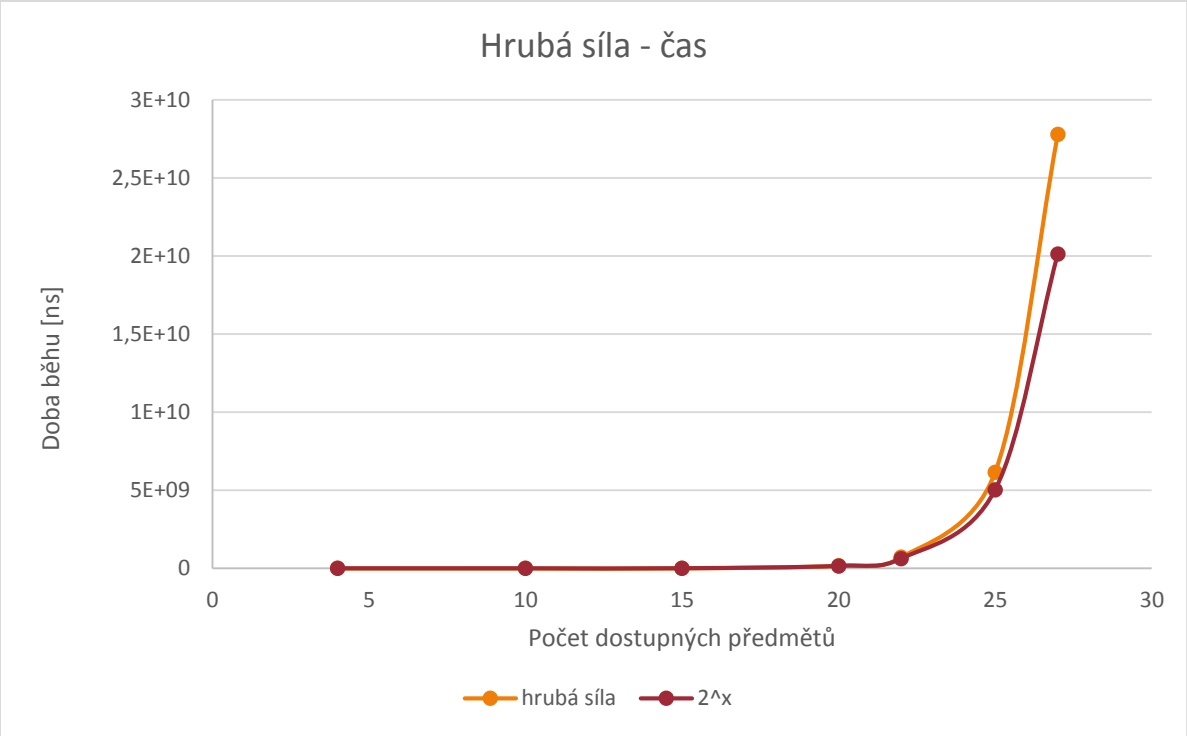
Chyby cen heuristik oproti hrubé síly jsem počítal dle pokynů na Eduxu. Tedy pro výpočet relevantní chyby byl použit vzorec $\varepsilon = (C(OPT) - C(APX)) / C(OPT)$ a byla počítána průběžně, ne na finálním součtu cen.

4.3 Výsledky

4.3.1 Čas běhu hrubé síly

Počet předmětů	Hrubá síla [ns]
4	1025
10	101899
15	4368028
20	168574680
22	720100616
25	6159543484
27	27779098070

Je vidět, že složitost algoritmu odpovídá počtu kombinací, které musíme otestovat, a tedy složitost roste exponenciálně s velikostí problému. Po ilustraci jsem do grafu přidal křivku 150*2^x.

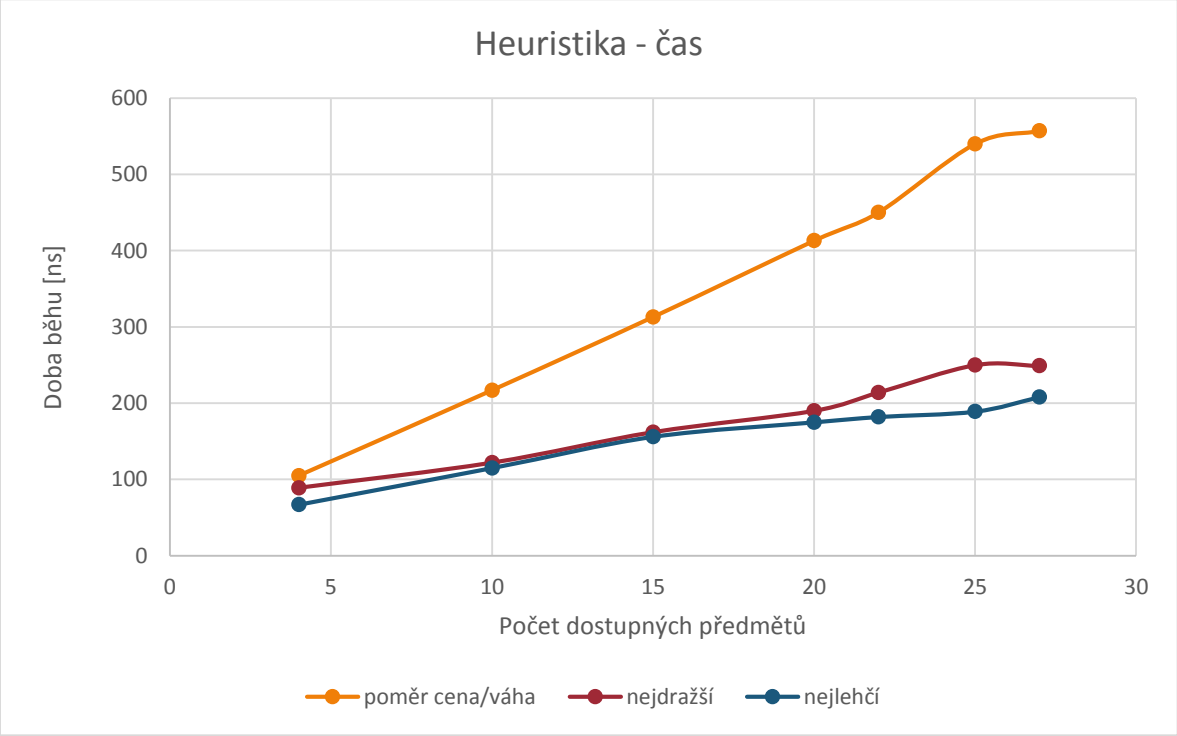


4.3.2 Čas běhu heuristik

Počet předmětů	Poměr cena/váha [ns]	Nejdražší [ns]	Nejlehčí [ns]
4	105	89	67
10	217	122	115
15	313	162	156
20	413	190	175
22	450	214	182
25	540	250	189
27	557	249	208

Následující graf příliš neodpovídá asymptotické složitosti heuristik, neboť nejnáročnější část heuristického algoritmu je řazení zadaných předmětů (vše ostatní je maximálně n), kterých je v mém případě maximálně 27. Složitost použitého řadícího algoritmu Quick Sort (interní v JVM) je průměrně n*log(n), ale na takto malých vstupech nelze očekávat, že se nám v grafu projeví logaritmická křivka.

Co na grafu vypozorovat lze, je skutečnost, že výpočet poměrů (dělení) přeci jen počítači zabralo o několik nanosekund více, než jednoduché porovnávání. Také lze srovnáním s předchozím grafem (a dat z tabulek) vypozorovat, že časy heuristik jsou mnohonásobně nižší.

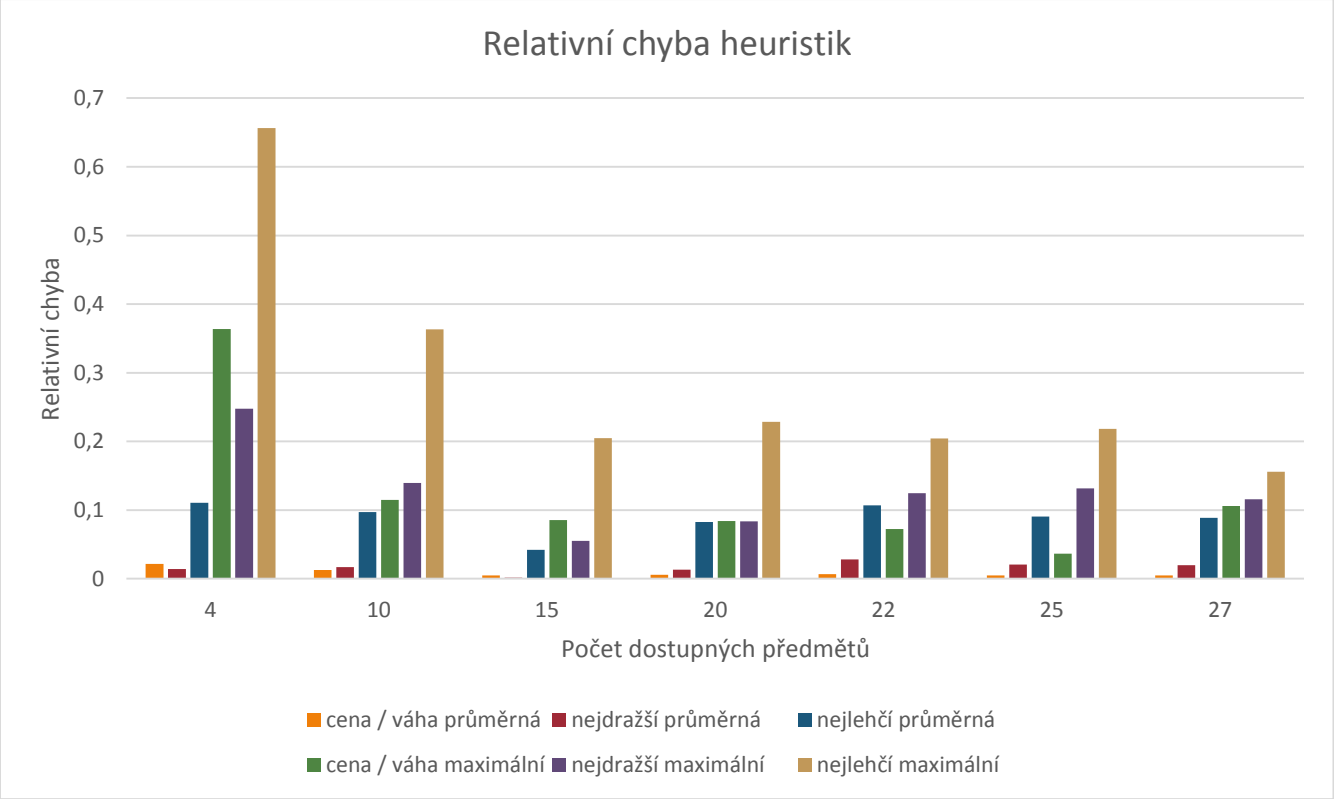


4.3.3 Průměrná a maximální relativní chyba

počet předmětů	cena / váha průměrná	nejdražší průměrná	nejlehčí průměrná	cena / váha maximální	nejdražší maximální	nejlehčí maximální
4	0,02174	0,01419	0,11072	0,36363	0,24764	0,65612
10	0,01286	0,01717	0,09716	0,1148	0,13976	0,36344
15	0,00475	0,00177	0,04193	0,08542	0,05531	0,20481
20	0,006	0,01316	0,08252	0,08433	0,08378	0,22858
22	0,00686	0,02826	0,10713	0,07228	0,12475	0,20435
25	0,00498	0,02073	0,09061	0,03678	0,13169	0,21845
27	0,00501	0,01971	0,08863	0,10601	0,11578	0,15595

Z grafu relativních chyb je možné vyčíst, že heuristika „**poměr cena/váha**“ je průměrně lepší, než heuristika „**nejdražší**“ a ta je lepší než heuristika „**nejlehčí**“.

Také je možné vidět, že relativní chyba klesá s velikostí instance (především ta maximální).



5 ZÁVĚR

Výsledky jsou interpretovány už v předchozí části „**Výsledky“**.

Závěrem uvádím, jak bychom se měli při řešení tohoto problému rozhodovat:

1. Pokud potřebujeme 100% **optimální** výsledky a máme dostatečně **malé instance**, měli bychom zvolit hrubou sílu.
2. Pokud máme **velké instance**, ale tolik nám nezáleží, pokud bude výsledek více či méně **neoptimální**, měli bychom zvolit heuristiku. Jak je vidět na základě měření, maximální i průměrná relativní chyba s velikostí instance klesá, takže bychom se přesto měli přiblížit optimu.
3. Pokud máme **malé instance** a **nezáleží** nám na optimálnosti, můžeme zvolit cokoliv.
4. Problémem zůstává, pokud máme **velké instance** a zároveň nám záleží na 100% **optimálnosti**. Tento případ je bohužel neřešitelný, protože se jedná o NP-těžký problém. V tomto případě musíme buď obětovat výpočetní čas a nechat algoritmus hrubé síly běžet celé hodiny, dny, roky, …, a nebo obětovat optimálnost a využít heuristiku.