1. Řešení problému batohu metodou hrubé síly a jednoduchou heuristikou

Zadání úlohy:

- Naprogramujte řešení problému batohu hrubou silou (tj. exaktně). Na zkušebních datech pozorujte závislost výpočetního času na n.
- Naprogramujte řešení problému batohu heuristikou podle poměru cena/váha.
- Pozorujte:
  - závislost výpočetního času na n
  - průměrnou relativní chybu (tj. zhoršení proti exaktní metodě)
  - maximální relativní chybu

Za úkol jsme měli vytvořit program, který řeší tzv. problém batohu. Cílem tohoto optimalizačního problému je najít takovou podmnožinu objektů, která splňuje omezující kritérium. V tomto případě, maximální nosnost batohu a zároveň maximalizovat zisk v podobě celkové hodnoty předmětů vložených do batohu. Vstupními proměnnými jsou objekty s váhou a cenou a batoh určený svou nosností.

Rozbor řešení:

Nejjednodušším přístupem je rekurzivní řešení problému, kdy vyzkoušíme všechny možné podmnožiny přidávaných objektů. Protože počet všechno možností je 2^k, kde k je počet objektů, čas potřebný k výpočtu roste exponenciálně v závislosti na k. Výhodou tohoto řešení je rychlost pro menší instance problémů, protože není nutné žádné předzpracování vstupů. Toto řešení dává vždy exaktní a správné výsledky jednoduše proto, že vyzkoušíme všechny možnosti.

Druhou metodou je seřadit si objekty podle poměru cena / váha a postupně je přidávat do batohu od nejvýhodnějšího objektu. Poměr nám slouží jako odhad, kterým směrem se vydat ve stavovém prostoru úlohy. Protože chceme počítat pouze odhad řešení je jedno, že výsledky mohou být horší než u metody hrubou silou. Výhoda tohoto přístupu je rychlost, kdy nám stačí k * log(k), operací nutných k seřazení a poté jeden průchod polem objektů a postupné přidávání do batohu. Tento přístup ale nedává exaktní výsledky, protože je závislý na vstupních parametrech. Stačil by například objekt s výhodným poměrem, který ale zabere místo pro více v průměru výhodnějších objektů a výsledkem je suboptimální hodnota.

Cílem této úlohy bylo experimentálně vyhodnotit časovou náročnost optimálního (bruteforce) řešení a heuristického řešení. Dalším výstupem je chybovost heuristického řešení, kdy vyhodnocujeme pro každý běh programu maximální a průměrnou relativní chybu heuristického a optimálního řešení.

Pro každou instanci je spuštěno optimální i heuristické řešení, spočítána relativní chyba. Tento proces se opakuje pro celý balík instancí a výsledkem je průměrný i celkový čas obou řešení a průměrná a maximální relativní chyba. Relativní chyba udává zhoršení heuristického řešení oproti optimálnímu.

Experimentální vyhodnocení:

V předmětu Efektivní implementace algoritmů jsme si vyzkoušeli implementaci několika algoritmů, pro práci s desetinnými čísly. Použil jsem proto vyzkoušené přepínače kompilátorů, které jsem upravil pro tuto úlohu.

Použité přepínače kompilátorů GCC:

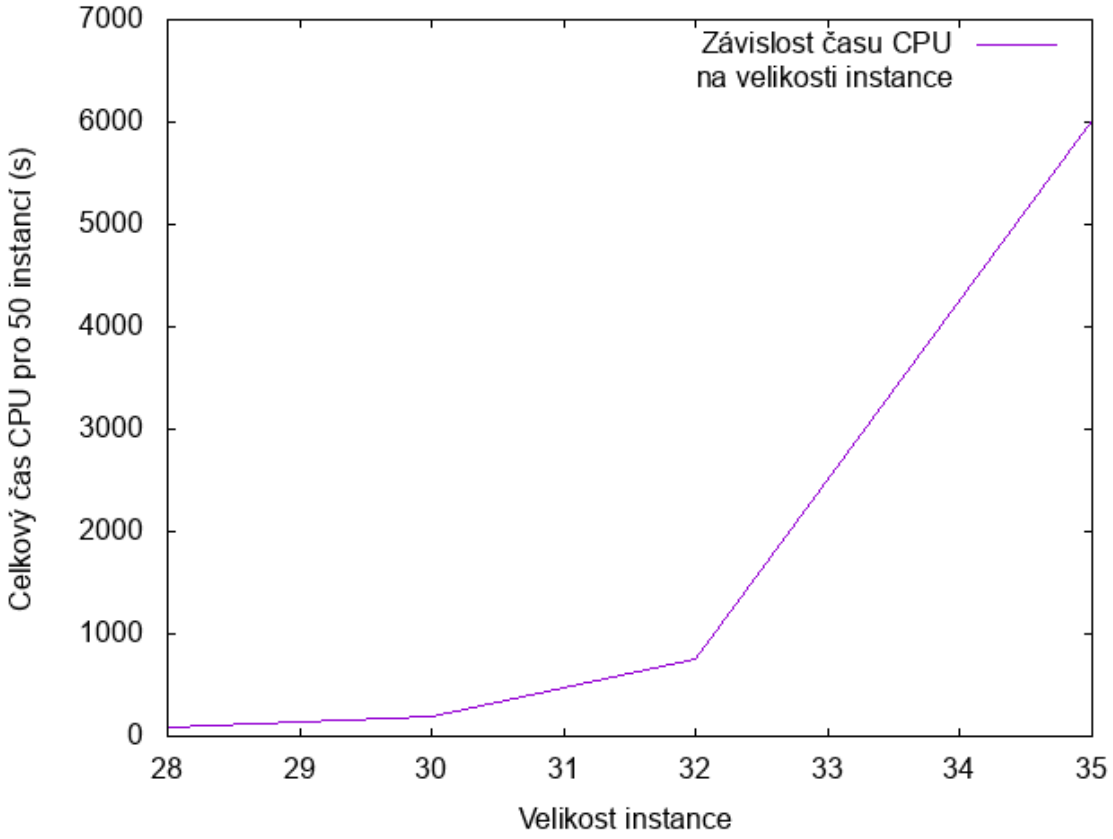
-Ofast -flto -Wno-sign-compare -std=c++11 -fopenmp -fstrict-aliasing -mfpmath=sse -mavx -mpc32 -funsafe-math-optimizations -ffast-math

Testovací soustava:

Intel(R) Core(TM) i5-3570K CPU @ 3.40GHz + 4 cores + Hyper Threading

Úlohy byly spouštěny v režimu jednoho vlákna a na PC neběžela žádná náročná úloha. K měření času jsem použil knihovnu OpenMP, se kterou mám dobré zkušenosti.

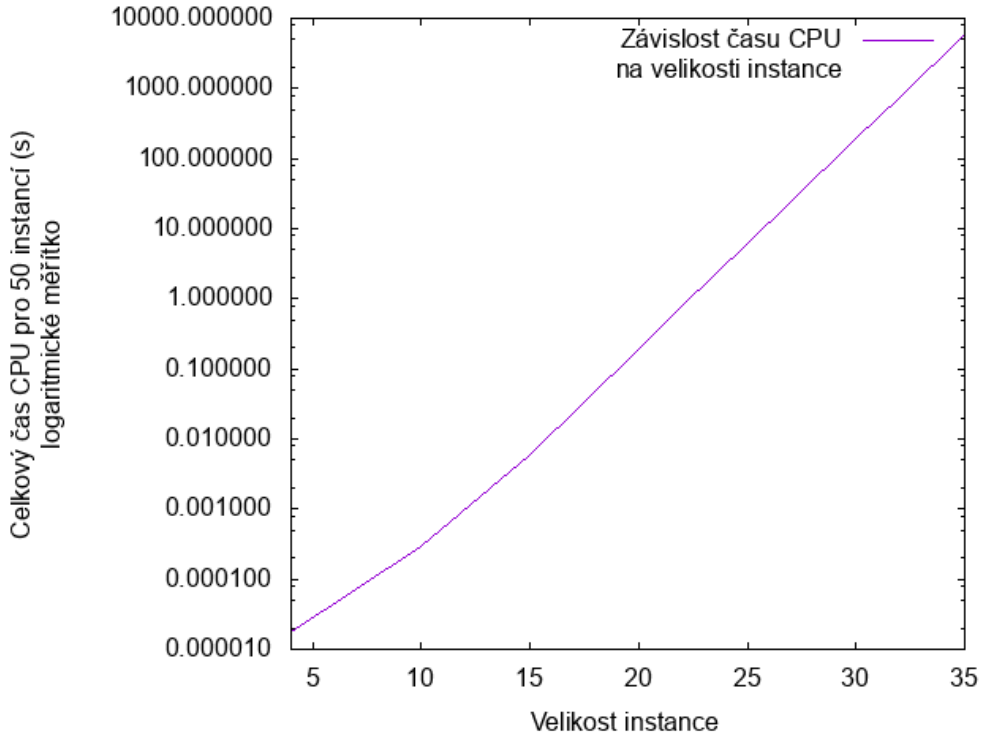
Naivní bruteforce řešení:



1. Graf závislosti celkového času (s) pro 50 instancí na její velikosti

Z důvodu nedostateného výkonu jsem vynechal instance o velikostech 37 a 40, protože bruforce řešení by trvalo několik hodin.

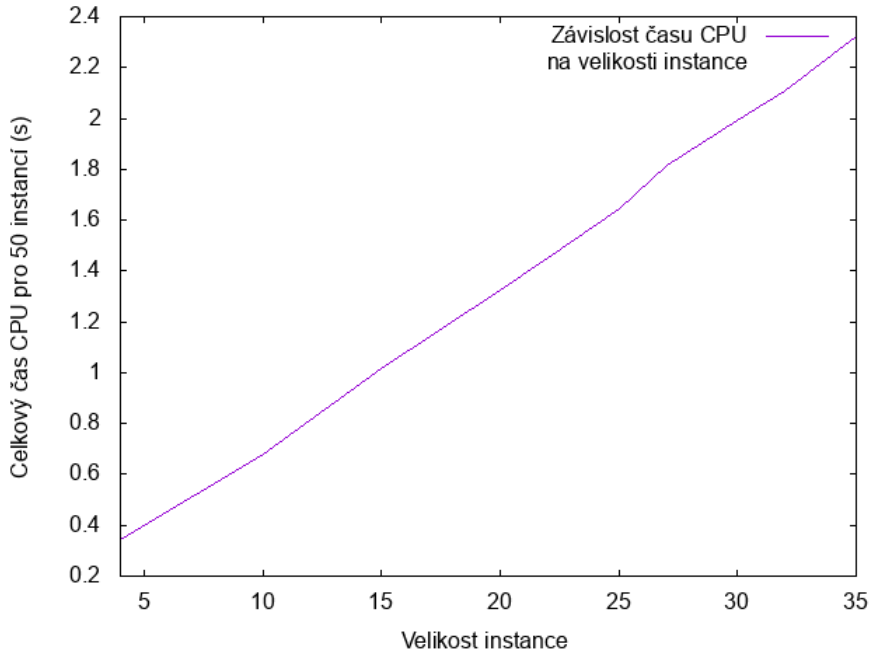
Graf [1] naměřených časů pro naivní řešení kopírují průběh exponeniály, tím potvrzují odhadovanou exponenciální závislost časové složitosti na velikosti vstupní instance.



2. Graf závislosti celkového času (s) pro 50 instancí na její velikosti s logaritmickým měřítkem na ose y

Z grafu [2] je vidět, že závislost je téměř lineární a protože je na ose y aplikováno logaritmické měřítko je závislost v datech exponenciální. Tento výsledek podporuje náš odhad.

Heuristické řešení:



3. Graf závislosti celkového času (s) pro 50 instancí na její velikosti pro 10^6 spuštění.

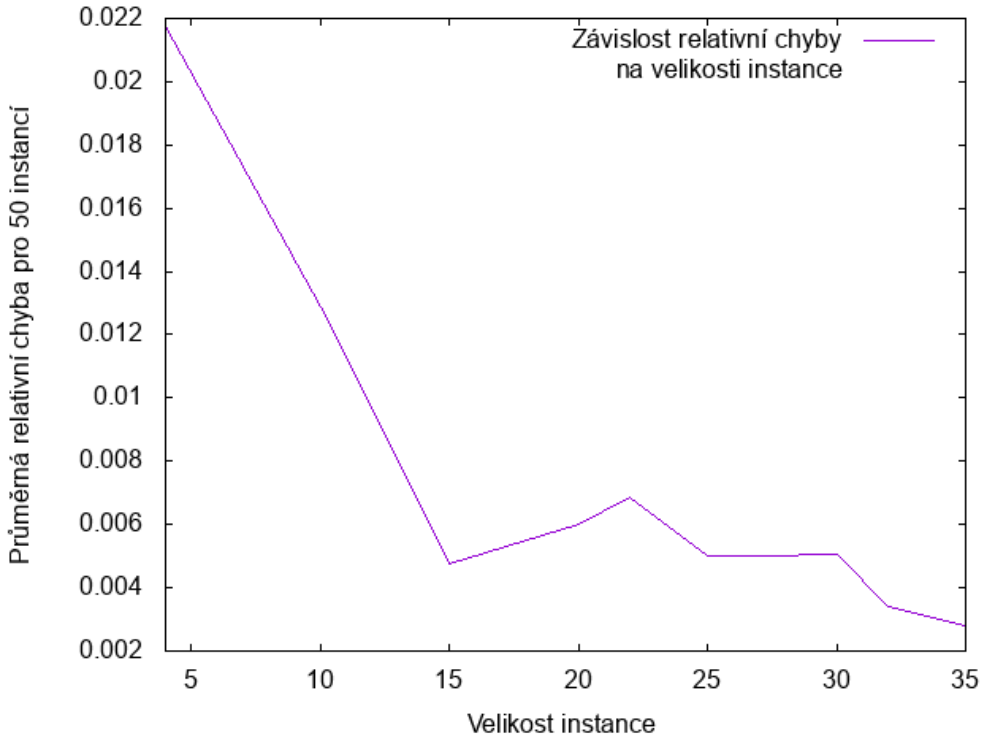
Heuristické řešení poskytuje odhad řešení v téměř lineáním čase v závislosti na velikosti instance. Přesnější závislost je n * log(n), protože vstupní data je potřeba seřadit, zbytek algoritmu je prostý průchod polem v lineárním čase.

Graf [3], na kterém jsou tyto hodnoty vyneseny, vypadá jako přímka. Pro účely měření času CPU byly hodnoty nejprve seřazeny a pak bylo spuštěno heuristické řešení a to 10^6 krát. Z tohoto důvodu je čas CPU nutný k seřazení vstupního pole zanedbatelný.

Relativní chyba:

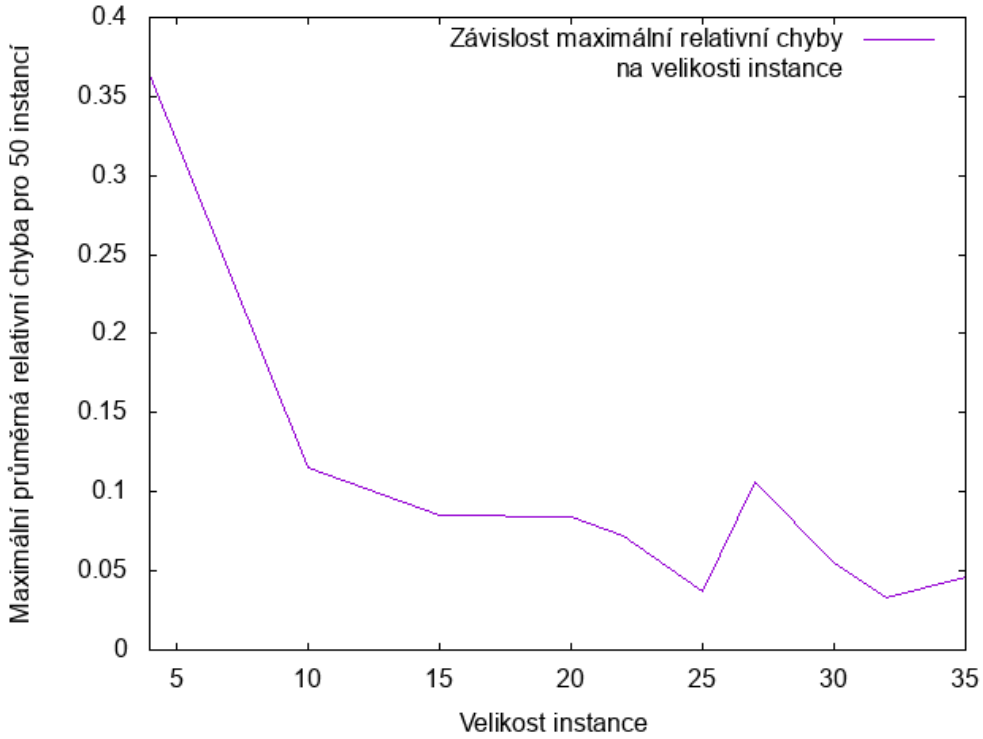
Měření relativní chyby jsem prováděl tak, jak je doporučeno v materiálech na eduxu pomocí vzorece a následného zprůměrování pro zadané instance:

ε = (C(OPT) - C(APX)) / C(OPT)
C(OPT) je cena optimálního řešení
C(APX) je cena vrácená naší heuristickou metodou.



4. Graf závislosti průměrné relativní chyby pro 50 instancí na její velikosti.

Graf [4] zobrazuje průměrnou relativní chybu, která porovnává cenu nejlepšího řešení pro 50 instancí problému. Z grafu je čitelné, že chyba klesá se zvějšující se velikostí instance.



5. Graf závislosti maximální relativní chyby pro 50 instancí na její velikosti.

Graf [5] zobrazuje maximální relativní chybu, která se vyskytla mezi výsledky pro 50 instancí problému. Z grafu je čitelné, že i maximální chyba klesá se zvějšující se velikostí instance.

Závěr:

Během měření jsme zjistili, že rozdíl exaktní metody a heuristické aproximace se snižuje s rostoucí velikostí problému. To je výhodné, protože heuristická metoda je časově poměrně nenáročná, ale optimální řešení metodou bruteforce má exponenciální složitost.

Je nutné říct, že se jedná pouze o statistická data. Ale můžeme s jistou mírou pravděpodobnosti říct, že pro podobné instance našeho problému bude chybovost a časové složitost výpočtu stejná, nebo srovnatelná.

Celkově mě výsledky překvapily, čekal jsem větší rozdíl mezi optimálním a heuristickým řešením.