# MI-PAA
Úkol 2 – zpráva

Tomáš Přeučil

3. listopadu 2018

## 1 Zadání úlohy

Zadáním úlohy bylo naprogramování pokročilých technik pro řešení problému batohu. Konkrétně branch and bound, dynamické programování podle ceny/kapacity a FPTAS algoritmus. Dále pak testování těchto metod.

## 2 Použité prostředky

### 2.1 Programovací jazyky a software

Úloha byla řešena v jazyce Python ve verzi 3.6 pod operačním systémem OS X 10.11.6. Program byl spouštěn z Bashe a tudíž pro spuštění nebylo využito žádné IDE. Pro měření času byla využita knihovna time a funkce *time.process_time()*. Jedná se o novinku od verze 3.3, která pracuje velmi podobně jako doporučovaná knihovna *timeit*.

### 2.2 Konfigurace testovacího stroje

Testování bylo provedeno na MacBooku Pro 13", Early 2011, modelové číslo MC700LL/A. Stroj obsahuje CPU Intel Core i5 2415M (2,3 GHz) a 16 GB RAM. Jediný další rozdíl oproti výchozí konfiguraci je vyměněný disk (za SSD), což však v tomto případě nehraje roli.

## 3 Rozbor možných variant řešení, rámcový postup jejich implementace a jejich popis

Metodu branch and bound lze naprogramovat dvěma různými způsoby. První je postupné lineární procházení vstupu seřazeného dle poměru cena/váha a stavění stromu s dle přítomnosti věcí v batohu. Tato metoda se snaží procházet tou cestou ve stromě, která vede nejvyššímu potenciálnímu zisku. Toho je dosaženo pomocí prioritní fronty uzlů.

Druhou, na naprogramování značně jednodušší, metodou je rekurzivní zanořování a zaříznutí větve rekurze v případě překročení kapacity batohu, nebo v případě že aktuální řešení nemůže přesáhnout nejlepší již nalezené řešení.

K tomu stačí na začátku znát sumu všech hodnot (která je v rekurzivním stromě předávána níže, vždy snížena o hodnotu aktuální věci) a mít jednu globální proměnnou s aktuálním maximem.

Co se dynamického programování týče, je možné volit mezi dekompozicí podle váhy a podle ceny. Obě varianty snižují výpočetní náročnost na úkor náročnosti paměťové a předpočítávají výsledky podproblémů tak, aby žádný z nich nebyl řešen vícekrát.

Pro dekompozici podle ceny stačí vytvořit rekurzivní pole o rozměru kapacita x počet věcí. V případě že vypočítáme řešení pro danou kapacitu a danou věc, vložíme ho do pole. Na začátku funkce pak stačí kontrolovat, jestli daný podproblém již nebyl řešen a dle toho se buď zanořit hlouběji, nebo vrátit již spočtené řešení.

U dekompozice podle ceny je nutné vytvořit pole o rozměru počet věcí x všechny možné ceny. To může být obrovský problém při použití desetinných čísel. Zde je tabulku nutné procházet iterativně a následně za posledního sloupce vybrat řešení, což se provede tak, že hledáme nejvyšší index v posledním sloupci, kde kapacita nepřekračuje kapacitu batohu.

Algoritmus FPTAS je rozšířením dynamického programování. Umožňuje škálovat osu udávající cenu výměnou za sníženou přesnost. Při použití celých čísel se může jednat například o vydělení všech cen nějakou konstantou, kterou je výsledek následně přenásoben.

# 4 Rámcový popis postupu řešení

Idea byla testovat vše z jednoho programu. Pro každou metodu byl tedy vytvořen „wrapper“, který je volán z hlavního programu.

Byly vytvořeny funkce pro branch and bound, dynamické programování dle ceny i kapacity a funkce pro dekompozici dle ceny byla rozšířena o možnost použití algoritmu FPTAS. Všechny funkce byly implementovány přesně dle postupu uvedeného v předchozí sekci.

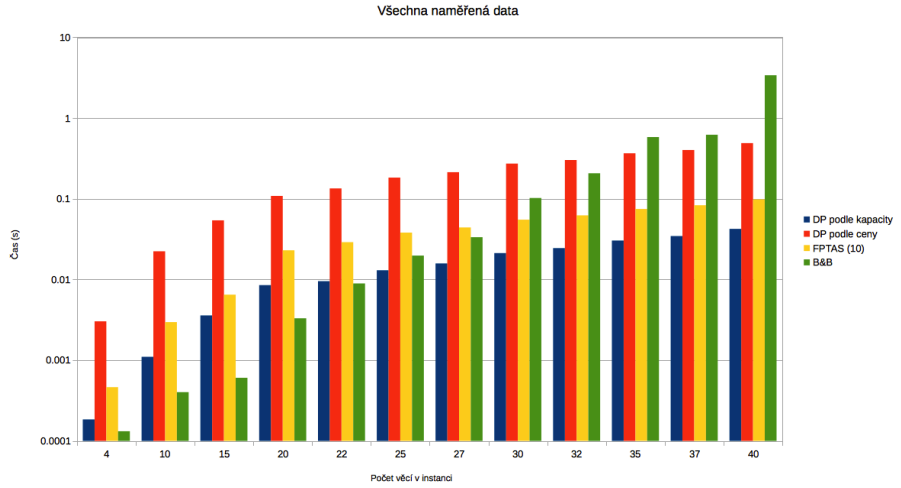
Z důvodu rozumného měření času byly pro počet věcí menší než 20 volány všechny funkce stokrát a výsledný čas byl stem vydělen.

Následný výstup byl zpracován pomocí awk a LibreOffice.

# 5 Naměřené výsledky

Nejprve byly naměřeny doby výpočtu pro všechny algoritmy. Ty znázorňuje graf 1 s logaritmickým měřítkem. Pro přehlednost jsou data porovnávána i v dalších grafech.

Z detailních porovnání bylo jako první provedeno srovnání různých dekompozic v dynamickém programování s poměrně překvapivými výsledky. Rekurzivní algoritmus dekompozice podle hmotnosti se ukázal jako výrazně rychlejší než iterace při dekompozici dle ceny. Proto bylo provedeno experimentální měření počtu provedených operací a ukázalo se, že rekurze musí provést výrazně méně průchodů. Například pro instanci 9054 o 10 prvcích jsou hodnoty 554 a 1512 operací. Pro instanci 9035 o 4 prvcích je rozdíl ještě markantnější s 25 a 213 operacemi. Porovnání času je na grafu 2.




Obrázek 1: Časy všech výpočtů

Dalším srovnáním je čas výpočtu za pomocí FPTAS algoritmu na základě parametru. Nepřekvapivě se se zvyšující hodnotou časy snižují – alg. FPTAS výrazně snižuje velikost pole k vyplnění. Tato měření proběhla na instanci o velikosti 40 věcí. Data jsou na grafu 3.

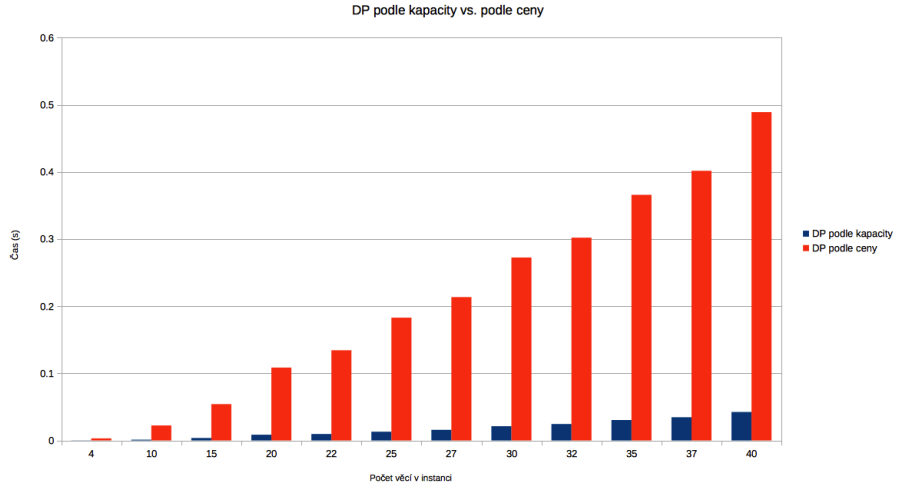
Dále byla srovnána doba výpočtu pro dekompozici dle ceny a FPTAS s hodnotou parametru 10. Měření lze vidět na grafu 4. Opět dochází k nepřekvapivému zrychlení, které je cca pětinásobné.

Poslední sledovanou veličinou byla relativní maximální a průměrná chyba. Zde nastalo největší překvapení a zklamání. Maximální chyba je i při vydělení ceny dvěma velmi, **velmi**, vysoká. To je bohužel dáno obsahem vstupních dat. U vyšších hodnot parametru dokonce maximální chyba dosáhla tak vysoké hodnoty, že algoritmus vůbec nenašel řešení. Je ale nutné dodat, že toto jsou **mezní případy**. Výskyty takto vysoké chyby jsou ve výstupu velmi ojedinělé. Průměrná chyba toto ukazuje, ačkoli i ta je u vysokých hodnot parametrů daleko od zanedbatelnosti. Měřená data se nachází v příloze a jejich reprezentaci lze najít na grafu 5.

Jako poslední by se hodilo říci, že FPTAS je vhodný primárně pro desetinné vstupy (kde při vysoké přesnosti je spotřeba paměti extrémní), nebo pro velmi velká čísla (opět reprezentována jako float).

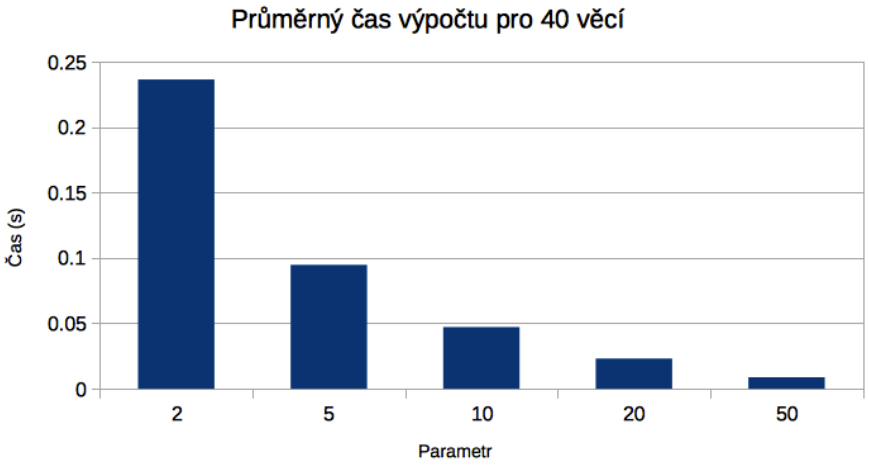
# 6 Závěr

Algortmus Branch and Bound se choval zcela dle očekávání. Sice je i nadále exponenciální (protože se pořád jedná o bruteforce), ale ve srovnání s hrubou sílou je výrazně rychlejší. Dynamické programování nejdříve překvapilo rozdílem mezi rychlostí dekompozice dle ceny a kapacity, ale nakonec se ukázalo, že se jedná pouze o implementační záležitost. FPTAS algoritmus zklamal výsledky maximální chyby v momentě, kdy jsou na vstupu celá čísla. V momentě, kdy by na vstupu byla čísla desetinná a docházelo by k zanedbávání desetinných míst,

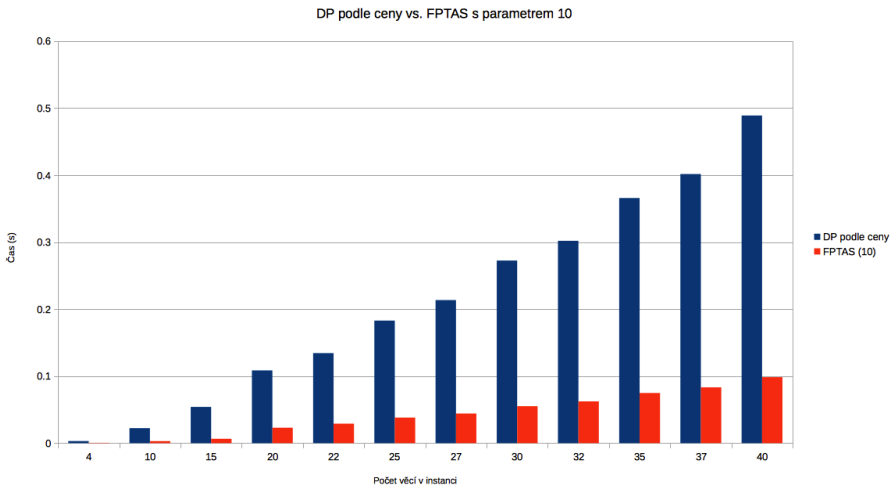



Obrázek 2: Srovnání dynamického programován podle kapacity a podle ceny

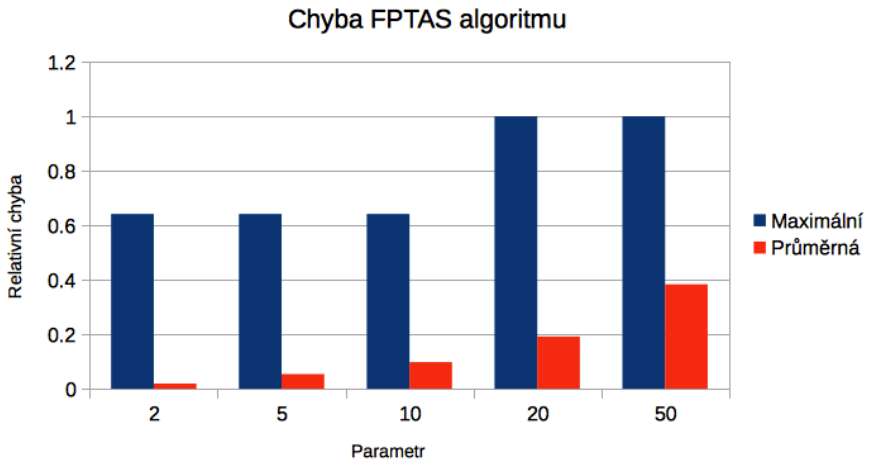
výsledek by jistě byl lepší. Průměrná chyba byla pro nízké hodnoty parametru v přijatelných mezích, ale opět – mezní případy zde opravdu vadí.




Obrázek 3: Časy výpočtu FPTAS alg. s ohledem na hodnotu parametru




Obrázek 4: Porovnání časů výpočtu pro DP a FPTAS




Obrázek 5: Chyba FPTAS algoritmu