Zpráva k 2. domácímu úkolu z předmětu MI-PAA

Jan Sokol
sokolja2@fit.cvut.cz

11. listopadu 2018

Abstrakt

Úkolem bylo nalézt řešení 0/1 problému batohu hrubou silou (tj. nalezení skutečného optima). Dále bylo třeba zkušebních datech pozorovat závislost výpočetního času na n (kde n je počet věcí v batohu). Druhou částí ukolu naprogramování řešení problému batohu dalšími, pokročilými metodami.

- První byla metoda větví a hranic (B&B). A to tak, aby omezujícím faktorem byla hodnota optimalizačního kritéria. Tj. při ořezávání shora omezení bylo překročení kapacity batohu. Omezení zdola bylo řešeno podmínkou, že stávající řešení nemůže být lepší než nejlepší dosud nalezené. Tato metoda je lepší (rychlejší) prořezávání prostorem, než je hrubá síla,
- metodou dynamického programování,
- FPTAS algoritmem, (tj. s použitím modifikovaného dynamického programování s dekompozicí podle ceny).

Na těchto datech bylo třeba pozorovat závislost výpočetního času na n (a to také s metodami z minulé úlohy - hrubou silou a jednoduchou heuristikou).

1 Výběr jazyka, popis implementovaných metod

Pro svou implementaci problému batohu jsem si vybral jazyk Python. Ačkoli to je jazyk interpretovaný a nečekal jsem závratné rychlosti výpočtů, mojím výběrem byl pro to, že jsem jazyk znal a pro jakýkolik koncept je pro mne nejrychlejší.

V případě hledání řešení hroubou silou jsem těžil z materiálů v přednáškách, tak i na internetu.

Metoda branch and bound zajišťuje, že prostor je prořezáván jak zdola, tak shora. Ty větve v rekurzi, které by neposkytly lepší výsledek (či by přesáhny kapacitu batohu), nejsou dále procházeny. V paměti držen nejlepší výsledek (globální hodnota). Před každým sestoupením do spodní větve se zkontroluje cena zbývajících (ještě nepřidaných) předmětů. Pokud součet cen zbývajících itemů a držené ceny batohu je menší, než nejlepší výsledek, k lepší hodnotě už se není možné dostat a průchod ukončujeme.

Pomocí metody dynamického programování přesouváme náročnost na CPU na paměťovou náročnost. Vybral jsem dekompozici dle ceny - abych funkce dále mohl využít i pro metodu FPTAS. V paměti držíme tabulku (decomposition table), kam ukládáme mezivýpočty. Sloupce jsou ceny, řádky jsou předměty. Těmito mezivýpočty jsou aktuální váhy v batohu. Výsledek je poté možné vidět ve spodním řádku - ta hodnota, co je nejvíce napravo.

Při výpočtu FPTAS můžeme ovlivnit kvalitu výsledku tím, že nastavíme proměnnou accurancy. Tou je možné definovat maximální relativní chybu, se kterou algoritmus bude pracovat. Zde jde o zanedbání určitého počtu bitů z ceny. Ceny předmětů jsou zpoměrovány, a poté je výpočet stejný, jako u dynamického programování.

2 Testovací Hardware

Všechny testy byly prováděny na cloudové linuxové instanci v AWS, běžící na Red Hat Enterprise Linux 7. Velikost instance byla: 2 Core CPU / 8 GB RAM, v názvosloví AWS **m4.large**.

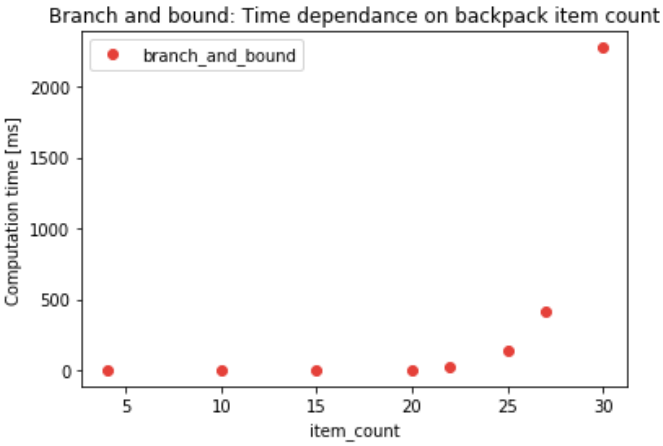
3 Měření výpočetního času

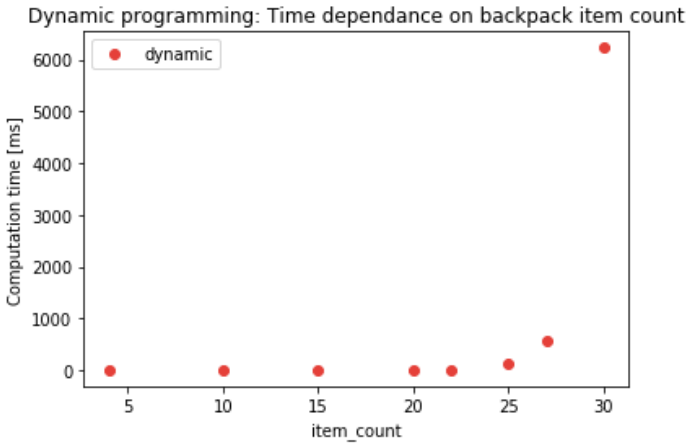
Výpočet běhu funkce je řešen tak, že je spočten strojový čas před během funkce, a také po něm. Tyto časy jsou od sebe odečteny a je vrácen čas v ms.

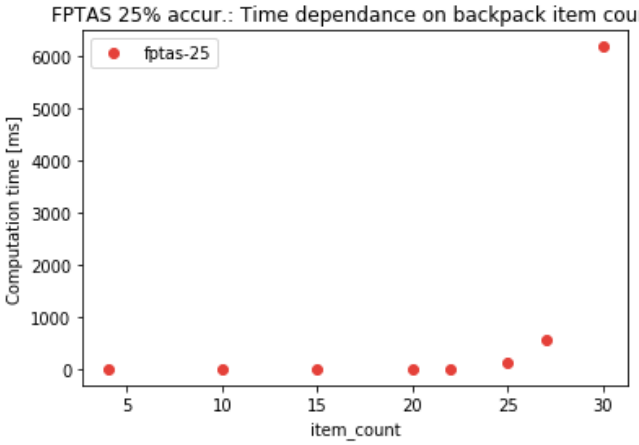
```
def timing(f):
    def wrap(*args):
        time1 = time.time()
        ret = f(*args)
        time2 = time.time()
        measured_time.append(
          {'type': f.__name__,
           'time': (time2-time1)*1000.0})
        return ret
    return wrap
```

3.1 Výpočetní časy jednotlivých metod

Pro zajímavost přidávám grafy časů výpočtů v závislosti na velikosti batohu. Tyto grafy jsou vytvořeny pro hrubou sílu, branch and bound, dynamické programování a FPTAS s nastavenou přesností $\varepsilon = 0.25 (\varepsilon \in (0; 1))$.

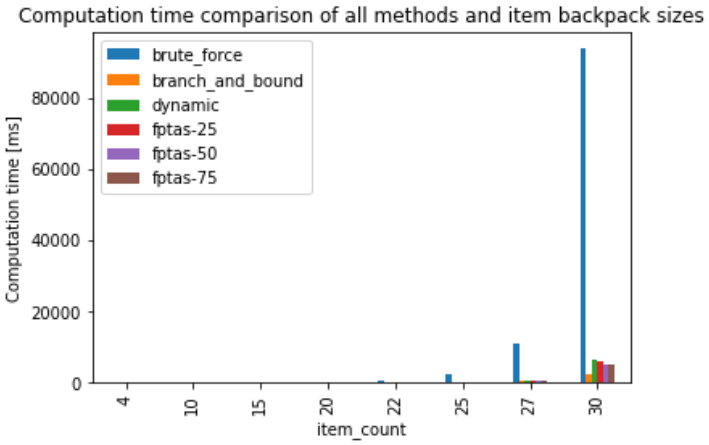






3.2 Srovnání výpočetních časů

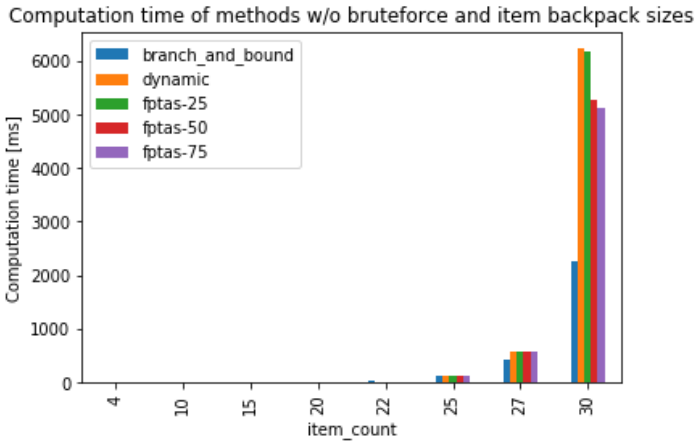
Zde přikládám srovnání výpočetních časů hrubé síly, B&B, dynamického programování a aproximativního algoritmu (FPTAS).



Výpočet hrubou silou je až 15x pomalejší, než ostatní metody. To ale není tak překvapivé, vzhledem k tomu, co jsme se dozvěděli v minulé úloze. Stále ale vidíme, že náročnost výpočtu u nových metod (Branch and bound, dynamické, FPTAS) znovu vzrůstá exponenciálně vzhledem k velikosti batohu. Ten čas je ale pořád mnohem menší, než u hrubé síly.

Metoda branch and bound ve většině případů může být rychlejší, než dynamické programování.

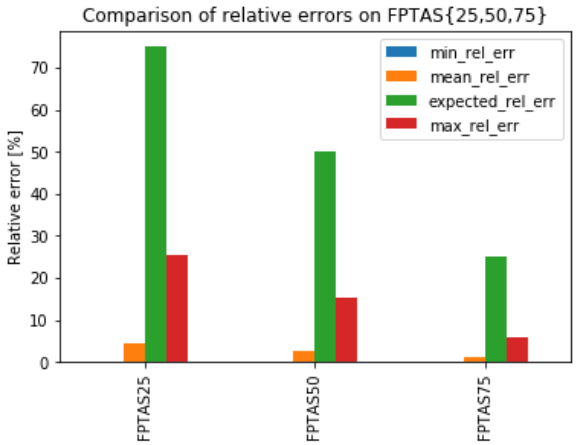
Teoreticky by se se snižující se přesností u algoritmu FPTAS měla snižovat i výpočetní složitost - což ale v mých výsledkách nevidím.



3.3 FPTAS - závislost chyby a výpočetního času algoritmu na přesnosti zobrazení

V grafu níže jsou vykresleny očekávané, (tato hranice byla vypočítaná předem) a průměrné, maximální a minimální relativní chyby (tyto hodnoty vnikly z dat). Minimální relativní chyba vždy byla nejmenší hodnota. Maximální relativní chyba je horní krajní hodnota. Očekávaná rel. chyba byla nastavena takto:

- 75 pro FPTAS25,
- 50 pro FPTAS50,
- 25 pro FPTAS75.



Jak je z grafů možné vidět, tak se průměrná chyba je hluboko pod tou očekávanou. I ty maximální se pohybovaly pod tou očekávanou.

4 Shrnutí a výsledky

Pomocí vykreslování grafů jsem dosáhl pouze velikosti 30 - při větší velikosti bahohu již vypočtení testovacích dat trvalo více než 24 hodin. Díky tomu v grafech větší data pro měření rychlostí nejsou přiložena.

Cílem této úlohy bylo urychlit výpočet řešení - což bylo, ať už u metody větvení, nebo dynamického programování, úspěšné.

Zrychlení u metody FPTAS už není tak velké - bylo rychlejší, než dynamické programování, ale pořád pomalejší, než Branch and Bound. Dle mého názoru za to může právě ta paměťová náročnost.

Pro vytváření grafů bylo využito Python notebooku, který je přiložen v adresáři **report/**. Grafy jsou vykresleny pomocí knihovny **mathplotlib**.