

Zpráva k 2. domácímu úkolu z předmětu MI-PAA

Jan Sokol
sokolja2@fit.cvut.cz

9. 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 úkolu 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, mojí výběrem byl pro to, že jsem jazyk znal a pro jakýkoliv koncept je pro mne nejrychlejší.

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

Metoda branch and bound zajišťuje, že prostor je prořezáván jak shora, tak zdola. Ty větve v rekurzi, které by neposkytly lepší výsledek (či by přesáhly 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 accuracy. Tou je možné definovat

maximální relativní chybu, se kterou algoritmus bude pracovat. Zde jde o zanedbání určitého počtu bitů z ceny. Ceny předmětů jsou zpoměřovány, 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**.

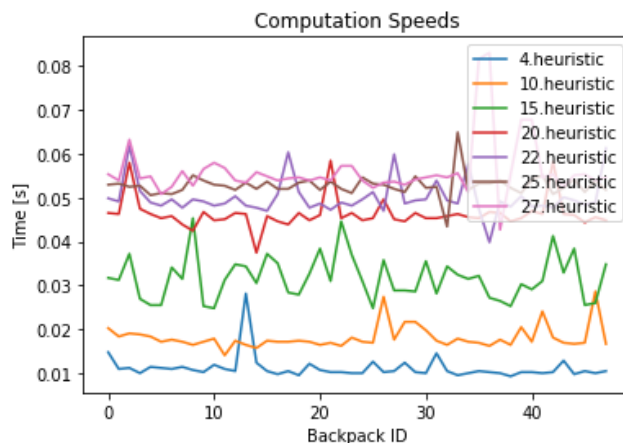
3 Měření výpočetního času

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
```

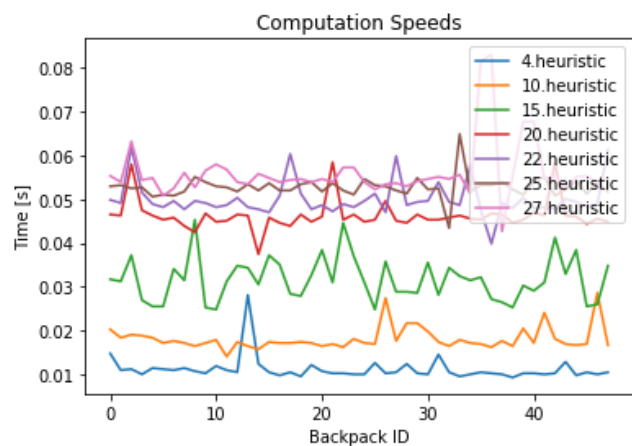
3.1 Srovnání výpočetních časů

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



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

...



4 Shrnutí a výsledky

Pomocí metody hrubé síly jsem dosáhl pouze velikosti 27 - při velikosti bahohu již vypočtení testovacích dat trvalo více než 24 hodin. Díky tomu v grafech větší data pro měření rychlosti nejsou přiložena. Relativní chyba používá data z referenčního řešení, a díky tomu, že řešení heuristikou je mnohem rychlejší, než hrubou silou, proto data obsahuje pro všechny zadaná data.

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