Zpráva k 3. domácímu úkolu z předmětu MI-PAA

Jan Sokol
sokolja2@fit.cvut.cz

2. prosince 2018

Abstrakt

Zadáním třetí úlohy bylo rozloženo do třech bodů. Tím prvním bylo prozkoumání citlivosti metod řešení problému batohu na parametry instancí generovaných generátorem náhodných instancí (přiloženým v zadání úlohy). V případě podezření na další závislosti nám bylo doporučeno, abychom modifikovali zdrojový tvar generátoru. Na základě zjištění z minulého bodu jsme měli provést experimentální vyhodnocení kvality řešení a výpočetní náročnosti. Zkoumat jsme měli obzvlášť metody:

- hrubá síla (pokud z implementace není evidentní úplná necitlivost na vlastnosti instancí),
- metoda větví a hranic,
- dynamické programování (dekompozice podle ceny a/nebo hmotnosti),
- heuristika - poměr cena/váha.

Na těchto metodách jsme měli zkoumat zejména závislost výpočetního času (případně počtu testovaných stavů) a rel. chyby (v případě heuristiky) na:

- maximální váze věcí,
- maximální ceně věcí,
- poměru kapacity batohu k sumární váze,
- granularitě.

1 Výběr jazyka

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.

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
```

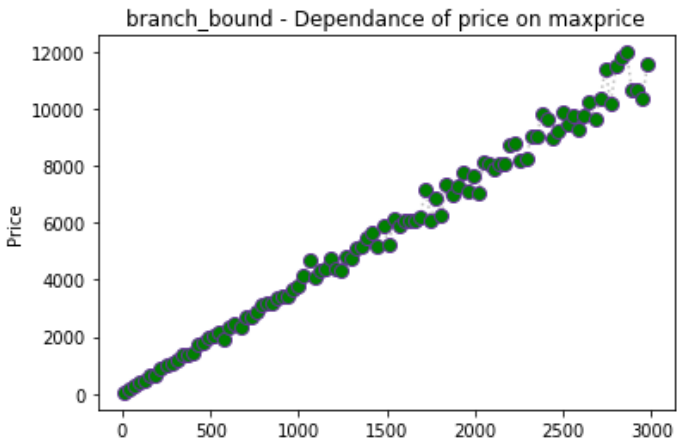
3.1 Postup experimentu a vybrané parametry

Jako vstup mého experimentu jsem zvolil závislost zkoumaného parametru z abstraktu (maximální váha věcí, maximální cena věcí, poměr kapacity batohu k sumární váze a granularita) na dvou mnou vybraných parametrech - času běhu řešení instance daným algoritmem dosažení spočtené ceny. Pro zjednodušení celého experimentu jsem postup vždy udělal stejný, jen se změněnými parametry.

Oba typy experimenty měly stejně definovány jejich výstupy. Jak u jednoho, tak i u druhého jsme schopni popsat kvalitu algoritmů, jen lehce jiným způsobem.

Závislost vypočtené průměrné ceny na definovaném parametru ze zadání může být občas nepřesné a zavádějící. Například v experimentu, kde měníme maximální cenu věcí, s parametry:

```
'items': 10, 'instances': 10,
'capacityratio': 0.5, 'maxweight': 100,
 'maxprice': 10, 'exp': 4, 'balance': 0
```



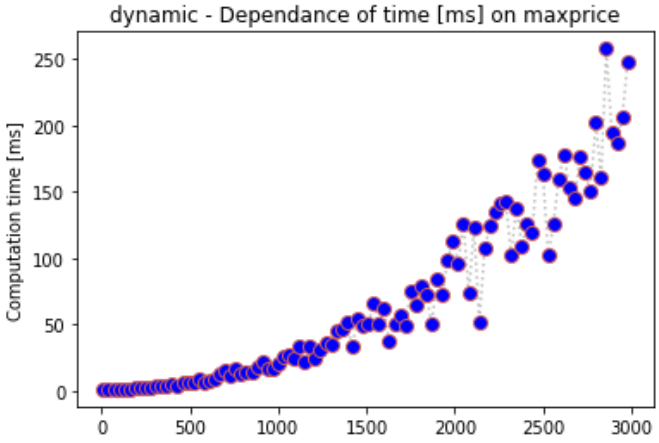
Když je experiment nastaven tak, že je dostupných jenom

10 předmětů (s jiejich maximální hodnotou 1), tak se nemůže stát, že by celková hodnota batohu přesáhla 10. A to vidíme přesně na grafu výše.

4 Zkoumání maximální hodnoty věcí

Při nastavení maximální ceny věcí na různé hodnoty, můžeme vidět že dynamickému algoritmu roste výpočetní čas se zvětšením maximální hodnoty.

```
'algorithm': 'dynamic', 'id_inst': 0, 'items': 10,
'instances': 10, 'capacityratio': 0.5,
'maxweight': 100, 'maxprice': 10,
'exp': 4, 'balance': 0
```



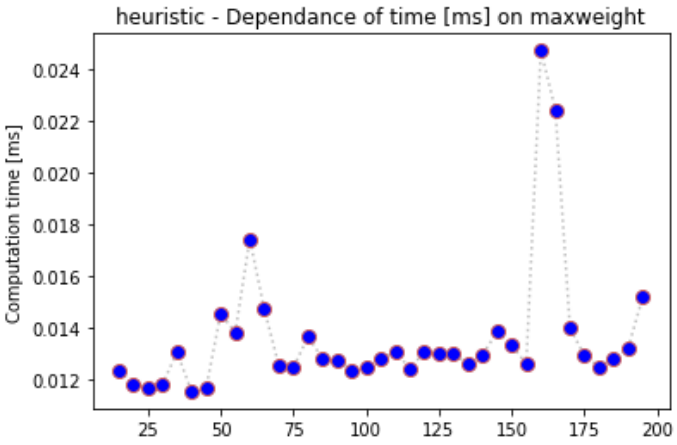
Časová závislost u dynamického programování v závislosti na maximální cenu věcí se zdá skoro exponenciální.

U všech ostatních metod je možné vidět, že maximální cena věcí výpočetní čas nijak neovlivňuje.

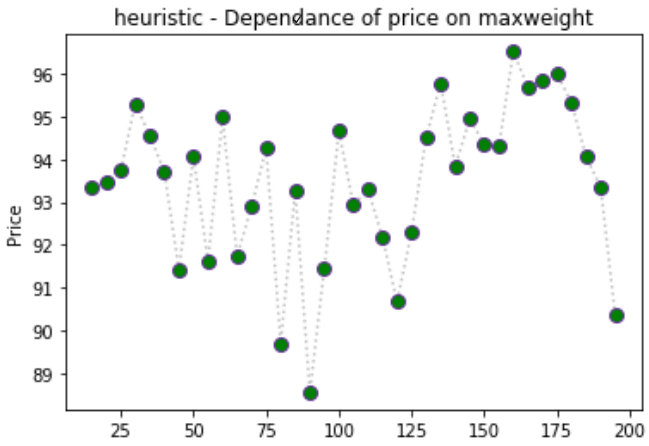
5 Zkoumání maximální váhy věcí

Algoritmus s heuristikou poměru cena/váha nám na časovém grafu ukazuje mírně stoupající trend – ale ten je nejspíše způsobem režií algoritmu (např. práce s pamětí, atp.). Parametry:

```
'algorithm': 'heuristic', 'id_inst': 0, 'items': 10,
'instances': 50, 'capacityratio': 0.5,
'maxweight': 15, 'maxprice': 25, 'exp': 4,
'balance': 0
```



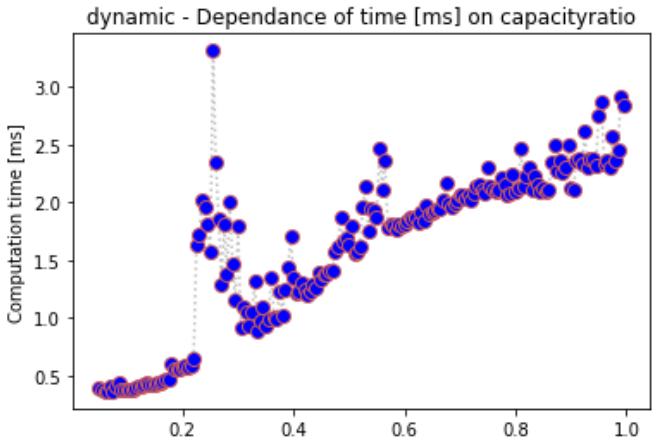
V cenových grafech jsem žádné závislosti či trendy nezaznamenal. Výsledky vypadají z většiny náhodně. Zde přikládám relevantní graf pro metodu heuristikou (se stejnými parametry jako výše), ostatní metody je možné vidět v přiloženém python notebooku.



6 Zkoumání poměru kapacity k sumární váze

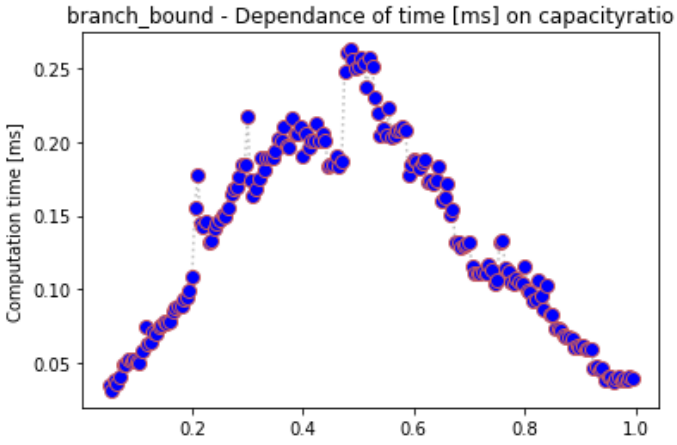
Zajímavou reakci ukázal algoritmus Dynamické programování. Z grafu níže můžeme vypozorovat, že dynamické programování bylo efektivnější při menší kapacitě vůči sumární váze.

```
'algorithm': 'dynamic', 'id_inst': 0, 'items': 10,
'instances': 5, 'capacityratio': 0.05,
'maxweight': 100, 'maxprice': 100,
'exp': 4, 'balance': 0
```



Zatímco jak můžeme vypozorovat u algoritmu Branch and Bound, ta je nejvíce efektivní jak pro malé kapacity v poměru k sumární váze, tak i ty velké. (Pro poměr 50% - 50% byl algoritmus nejméně efektivní).

```
'algorithm': 'branch_bound', 'id_inst': 0, 'items': 10,
'instances': 5, 'capacityratio': 0.05,
'maxweight': 100, 'maxprice': 100,
'exp': 4, 'balance': 0
```



U ostatních metod jsem žádnou závislost poměru kapacity k sumární váze nevypozoroval. Rád bych znovu odkázal na Jupyter notebook, kde je toto možné vidět.

7 Zkoumání granularity instancí

Při zkoumání reakce algoritmů na změnu granularity jsem měnil exponent (proměnná exp v konfiguraci) v krocích po 0.005, od 0 do 1. Dále jsem měnil balanci mezi malými věcmi (parametr -k), který byl nastaven na -1, 0 a 1.

Nicméně při žádných změnách jsem na žádné závislosti nenarazil. Pro zobrazení měřených výsledků doporučuji nahlédnout do Jupyter notebooku přiloženém v repu.

8 Shrnutí a výsledky

Výsledky z experimentů v této úloze nám převážně potvrzují vědomosti nabyté v předchozích úkolech.

Nejvíce mě překvapilo změnění poměru kapacity k sumární váze u metod branch and bound a dynamického programování. Z výsledků jsou mi jasné jednotlivé výhody a nevýhody algoritmů při změněných parametrech.

Dále bylo zajímavé pozorovat změnu maximalni hodnoty veci u dynamického programování, kdy výpočetní složitost algoritmu rostla velmi rychle při zvětšování proměnné.

Pro vytváření grafů bylo využito Python notebooku, který je přiložen v adresáři **report/report3/experiments.ipynb**, případně z něj vygenerovaný MD soubor v **report/report3/data/experiments.zip**. Grafy jsou vykresleny pomocí knihovny **mathplotlib**.

Data vypočtená z generátoru a běhu algoritmu je možné vidět v tomto repozitáři, v cestě **report/report3/**. Tento adresář také obsahuje funkce, pomocí kterých bylo experimentu dosaženo a pomocí kterých byly grafy vygenerovány.