1 MI-PAA, Úloha 2: Řešení problému batohu dynamickým programováním, metodou větví a hranic a aproximativním algoritmem

Marián Hlaváč, 11 Nov 2017 (hlavam30)
marian.hlavac@fit.cvut.cz
https://github.com/mmajko/knapsack-problem

1.1 Zadání úlohy

- Naprogramujte řešení problému batohu:
- metodou větví a hranic (B&B) tak, aby omezujícím faktorem byla hodnota optimalizačního kritéria. Tj. použijte ořezávání shora (překročení kapacity batohu) i zdola (stávající řešení nemůže být lepší než nejlepší dosud nalezené),
- metodou dynamického programování (dekompozice podle kapacity nebo podle cen),
- FPTAS algoritmem, tj. s použitím modifikovaného dynamického programování s dekompozicí podle ceny (při použití dekompozice podle kapacity není algoritmus FPTAS).

1.2 Možné varianty řešení

Problém batohu je možné řešit hrubou silou, heuristicky, dynamickým programováním, algoritmem "meet-in-the-middle" a dalšími způsoby.

V minulé úloze jsme se zabývali řešením hrubou silou a urychlení výpočtu pomocí heuristik. V této úloze bylo využito dvou způsobů urychlení výpočtu. Jedním ze způsobů je lepší prořezávání stavového prostoru. Druhým způsobem je využití dynamického programování a pomocí aproximačního schématu plně polynomiálního času (FPTAS - fully polynomial-time approx. scheme).

1.3 Popis postupu řešení

Algoritmus a celý program poskytující výsledky je napsán v jazyce Rust. Program načte instance z datových souborů určených pro tuto úlohu a vypočte řešení všemi implementovanými metodami. Zapíše délku provádění výpočtu a všechna data poskytne v CSV formátu. Druhým nástrojem je Jupyter Notebook, který poskytnutá data vizualizuje.

Oproti minulé úlohy byl algoritmus hrubou silou přepsán z implementace pracující s bitovými maskami do rekurzivní varianty. Umožnilo to tak jednodušší implementaci Branch & Bound prořezávání.

Prořezávání stavového prostoru obohacené o Branch & Bound zajistí, že je prostor prořezán shora i zdola. Větve rekurze, které by přesáhly kapacitu batohu, nebo by neposkytly lepší výsledek, než dosavadní dosažený jsou ořezány.

Dynamické programování umožňuje zrychlení výpočtu na úkor pamětové náročnosti. Byla zvolena varianta dekompozice podle ceny, díky čemu pak lze snadno implementovat FPTAS.

Konečně, FPTAS představuje způsob, jak ovlivňovat "kvalitu" a rychlost výpočtu. Vstupní proměnnou je možné definovat maximální relativní chybu. Čím větší je povolená tato relativní chyba, tím by měl být výpočet teoreticky rychlejší.

1.3.1 Kostra algoritmu

Celý algoritmus je k nahlédnutí ve zdrojových souborech programu na GitHubu. Pro rychlou představu je níže uveden úryvek z funkce mající na starost FPTAS výpočet.

```
...
    // Find the largest price in knapsack
    let max_price = knap.items.iter().fold(0, |acc, &x| if x.price > acc { x.price } else { ac
    let ratio = (1.0 - accuracy) * max_price as f32 / knap.items.len() as f32;

    // Modify prices in knapsack
    let mut mod_knap = knap.clone();
    mod_knap.items = knap.items.iter().map(|item| { KnapItem {
        id: item.id, weight: item.weight, price: f32::floor(item.price as f32 / ratio) as u16
    } }).collect();

    // Solve with dynamic solver
    let mut solution = solver_dynamic::solve(mod_knap);
...
```

Branch & Bound drží v paměti nejlepší dosažený výsledek a před každým sestoupením do větve rekurze zkontroluje cenu zbývajících předmětů, které nebyly do batohu přidány. Pokud je tato cena sečtená s cenou batohu nižší, než nejlepší dosažený výsledek, je jasné, že prohledáváním této větve se k lepšímu výsledku již nemůžeme dostat a můžeme rekurzi na tomto místě ukončit.

Dynamické programování přenáší výpočetní náročnosti na paměťovou náročnost. Držíme v paměti tabulku, do které ukládáme provedené výpočty. Konkrétně v této implementaci jde o dekompozici podle ceny. Do tabulky tak ukládáme výsledné váhy, sloupce představují ceny a řádky jednotlivé předměty. Výsledek pak nalezneme na posledním řádku v nejpravějším nenulovém sloupci.

FPTAS pak pomocí formule upravující ceny předmětů umožní vložit do výpočtu další proměnnou, kterou lze výpočet ovlivnit. Efektivně jde o zanedbávání určitého počtu bitů hodnot ceny. Výpočet je až na pozměněné ceny totožný s výpočtem pomocí dynamického programování s kompozicí podle ceny a je tak použita stejný kód implementace.

1.4 Surová naměřená (raw) data

Níže uvedená tabulka je náhled na kompletní surová výstupní data z programu. Data lze získat jednoduchým způsobem (např. pro kontrolu) -- spuštěním skriptu `generate.sh`, který vytvoří soubor `results.csv` obsahující tato data.

1.4.1 Sloupce

Názvy sloupců se vyskytují i dále v textu, zde je jejich stručný popis:

- knap_id - identifikátor instance
- item_count - počet předmětů (konfigurace instance)
- method - metoda výpočtu
- Recursive - rekurzivní
- BranchAndBound - prořezávání Branch & Bound
- Dynamic - dynamickým programováním, dekompozice podle ceny
- FPTAS25 - FPTAS s 25% přesností (max. rel. chybou 0.75)
- FPTAS50 - FPTAS s 50% přesností (max. rel. chybou 0.50)
- FPTAS75 - FPTAS s 75% přesností (max. rel. chybou 0.25)
- price - vypočtená celková cena batohu

- elapsed_ms - doba výpočtu v milisekundách
- optimal_price - optimální cena batohu

Out[1]:

	knap_id	method	item_count	price	elapsed_ms	optimal_price
0	9000	Recursive	4	473	0.010678	473
1	9000	BranchAndBound	4	473	0.003109	473
2	9000	Dynamic	4	473	0.016369	473
3	9000	FPTAS25	4	418	0.009753	473
4	9000	FPTAS50	4	446	0.000994	473
5	9000	FPTAS75	4	459	0.000921	473
6	9001	Recursive	4	326	0.002636	326
…	…	…	…	…	…	…	…
1758	9293	Recursive	25	3210	5022.000000	3210
1759	9293	BranchAndBound	25	3210	85.000000	3210
1760	9293	Dynamic	25	3210	256.000000	3210
1761	9293	FPTAS25	25	3128	219.000000	3210
1762	9293	FPTAS50	25	3156	217.000000	3210
1763	9293	FPTAS75	25	3184	209.000000	3210
1764	9294	Recursive	25	2746	4958.000000	2746

[1765 rows x 6 columns]

1.5 Výsledky měření

Níže jsou uvedeny výsledky derivované z dat. Rychlosti řešení jsou seskupeny podle počtu předmětů a zprůměrovány. Výsledné průměrné časy jsou ke každé metodě řešení uvedeny jak tabulkou, tak grafem.

Měřítko grafů je lineární na vertikální ose.

1.5.1 Průměrné rychlosti řešení pro všechny metody

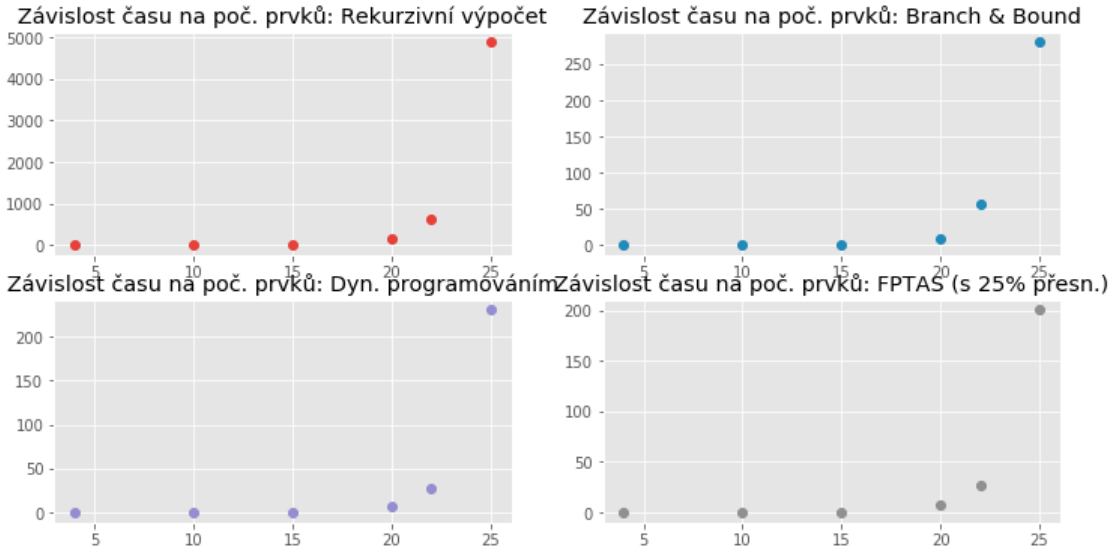
V tabulce níže lze porovnat, jaké byly průměrné rychlosti výpočtu pro různé počty předmětů v batohu. Uvedené časy jsou v milisekundách.

Out[2]:

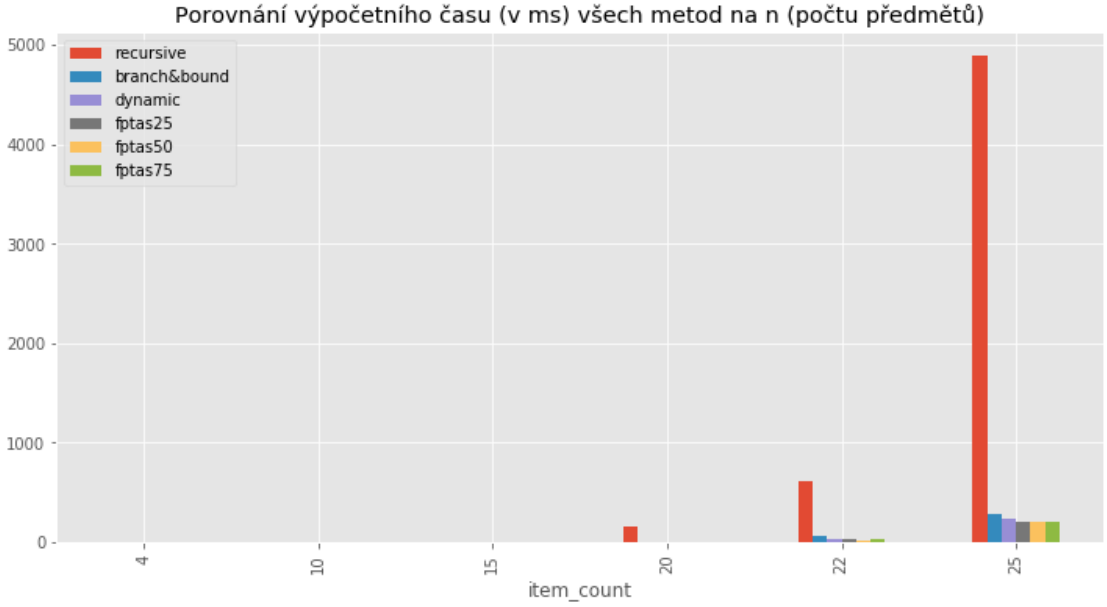
item_count	recursive	branch&bound	dynamic	fptas25	fptas50	fptas75
4	0.002749	0.002628	0.001708	0.001022	0.000851	0.001005
10	0.156713	0.068365	0.012568	0.008723	0.007277	
15	4.560000	0.237215	0.240835	0.234945	0.226642	
20	161.040000	7.892184	7.160000	7.020000	6.760000	
22	614.860000	55.899166	26.740000	27.580000	24.120000	
25	4886.600000	280.318182	230.772727	200.750000	211.113636	

```
10          0.007300
15          0.230458
20          6.600000
22         26.840000
25        212.795455
```

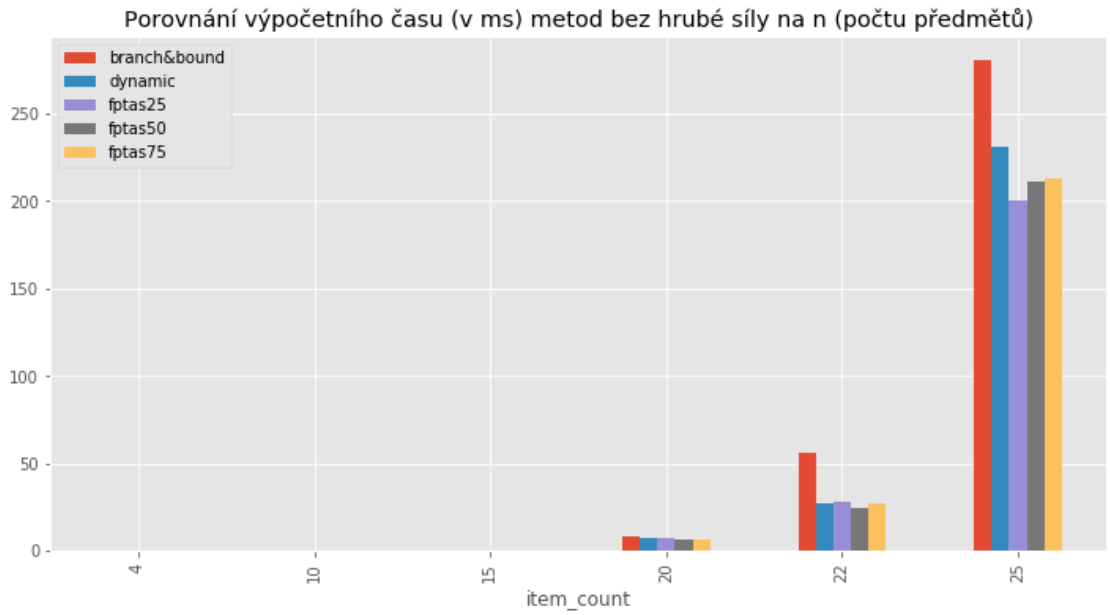
Out[3]: `[<matplotlib.lines.Line2D at 0x109acec88>]`



Out[4]: `<matplotlib.axes._subplots.AxesSubplot at 0x1097c3fd0>`



Out[5]: <matplotlib.axes._subplots.AxesSubplot at 0x10d397a20>



Výsledky náročnosti výpočtu hrubou silou nejsou překvapivé, díky znalostem z minulé úlohy.

Pokud prozkoumáme pouze metody, které jsou oproti minulé úloze "nové", lze vypozorovat a potvrdit tak, že rychlost vzrůstu náročnosti výpočtu při vzrůstajícím n je stejná jako rychlost vzrůstu při hrubé síle, ovšem lze vypozorovat znatelnou redukci výpočetního času. Výpočet je až 25 x rychlejší, než výpočet hrubou silou.

Ukazuje se, že metoda Branch & Bound může být často rychlejší, než řešení dynamickým programováním. Tento fakt nejvíce ovlivňuje charakteristika vstupních dat (optimální řešení se najde velmi brzo).

Z grafů lze vyčíst, že se mou implementací nepodařilo potvrdit, že by výpočetní náročnost klesala s větší povolenou relativní chybou.

1.5.2 Porovnání relativní chyby u FPTAS

Na grafu níže lze vidět jaké byly relativní chyby pro jednotlivé přesnosti (max. relativní chyby) algoritmu FPTAS.

Sloupeček expected představuje očekávanou maximální relativní chybu. Maximum a minimum jsou krajní případy a mean je průměrná relativní chyba.

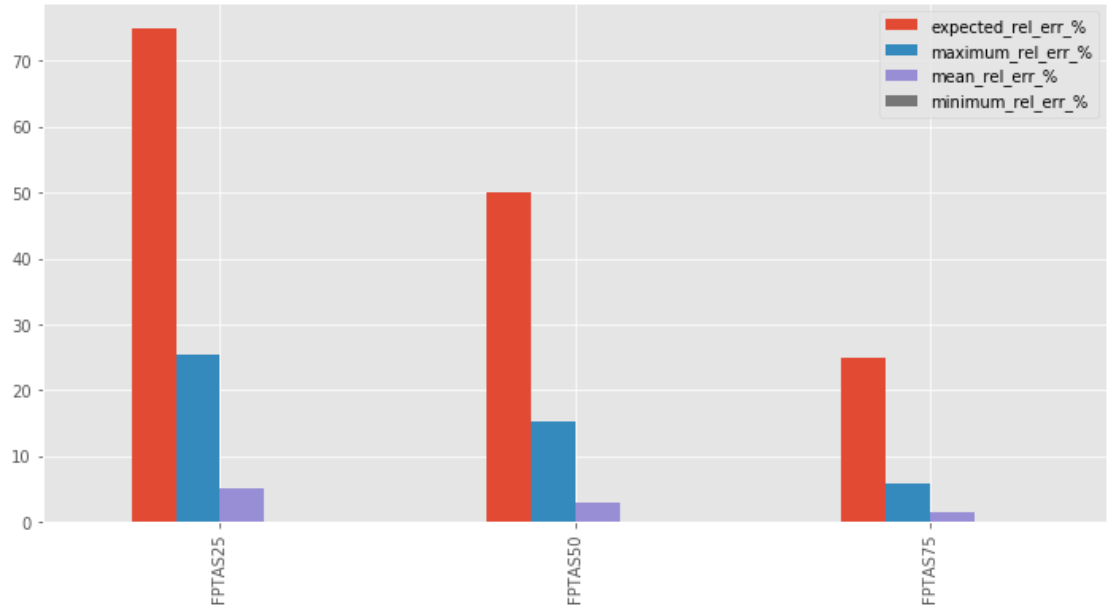
Out[6]:

	expected_rel_err_%	maximum_rel_err_%	mean_rel_err_%	\
FPTAS25	75	25.446429	4.999808	
FPTAS50	50	15.360502	2.874709	

```
FPTAS75                  25          5.956113        1.430135

         minimum_rel_err_%
FPTAS25                0.0
FPTAS50                0.0
FPTAS75                0.0
```

```
Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x10d552908>
```



Ukázalo se, že se průměrná chyba pohybovala daleko za hranicí očekávané relativní chyby, dokonce i maximální chyby se pohybovaly daleko od očekávané hranice.

Minimální chyba byla vždy nulová. Některé instance se podařilo spočítat k optimální hodnotě.

1.6 Závěr

Urychlení výpočtu pomocí prořezávání a dynamickým programováním bylo úspěšné. Zrychlení výpočtu je daleko lepší, než zrychlení, kterého se podařilo dosáhnout v minulé úloze.

U FPTAS se ovšem už takového úspěchu pravděpodobně nedosáhlo. Zrychlení oproti ostatním metodám není znatelné, naopak se v některých případech ukázala časová náročnost jako vyšší.

Některé výpočty byly vypočteny skutečně s chybami, tyto chyby se průměrně pohybovaly kolem 2,6% (relativní chyba). Dalo by se však očekávat, že chyby budou daleko větší a více úzce kopírovat stanovenou hranici maximální relativní chyby. Zároveň se očekávalo další výrazné zrychlení.

Vzhledem k tomu, že jsme byli schopní nalézt optimum v podobných časech, se FPTAS v tomto případě nevyplácí.

Zdrojové soubory úlohy lze najít na GitHubu. Link je uveden v hlavičce zprávy.