Popis problému:

Naprogramujte řešení batohu:

1) 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),
2) metodou dynamického programování (dekompozice podle kapacity nebo podle cen),
3) FPTAS algoritmem, tj. s použitím modifikovaného dynamického programování s dekompozicí podle ceny (při použití dekompozice podle kapacity není algoritmus FPTAS).

Rozbor možných řešení a algoritmus

Principiálně je řešení zadané popisem problém, záleží tedy spíše na implementačních detailech. Pro implementaci jsem zvolil jazyk Python (což nejspíš není nejvhodnější jazyk, ale je mi aktuálně nejbližší).

1) První část úkolu poskytuje největší možnost kreativity ve volbě generování kombinací, které jsou potřeba pro projití všech validních řešení problému batohu. Tuto část úlohy řeším rekurzí, kdy při každém volání kontroluji překročení kapacity batohu (ořezávání shora) a jestli nejdu do větve s horším, než dosud nalezeným řešením (ořezávání zdola). Algoritmus tedy rekurzivně generuje ořezanou množinu možných řešení a ty validní přidává do pole výsledků. V dalším kroku už se následně jenom vybere nejdražší řešení a řešení je na světě.
2) Řešení dekompozicí podle ceny je následující: Předpokládáme, že w_1, w_2,…, w_n, W jsou pozitivní čísla. Definujeme m[i,w] jako maximální hodnotu která může být dosažena s váhou menší nebo rovnou než w za použití věcí až do i (prvních i věcí). Následně rekurzivně definujeme m[i,w] jako:
    a. m[0,w] = 0
    b. m[i,w] = m[i-1,w] , když $w_i > w$
    c. m[i,w] = max(m[i-1,w], m[i-1,w-w_i]+w_i), když $w_i < w$

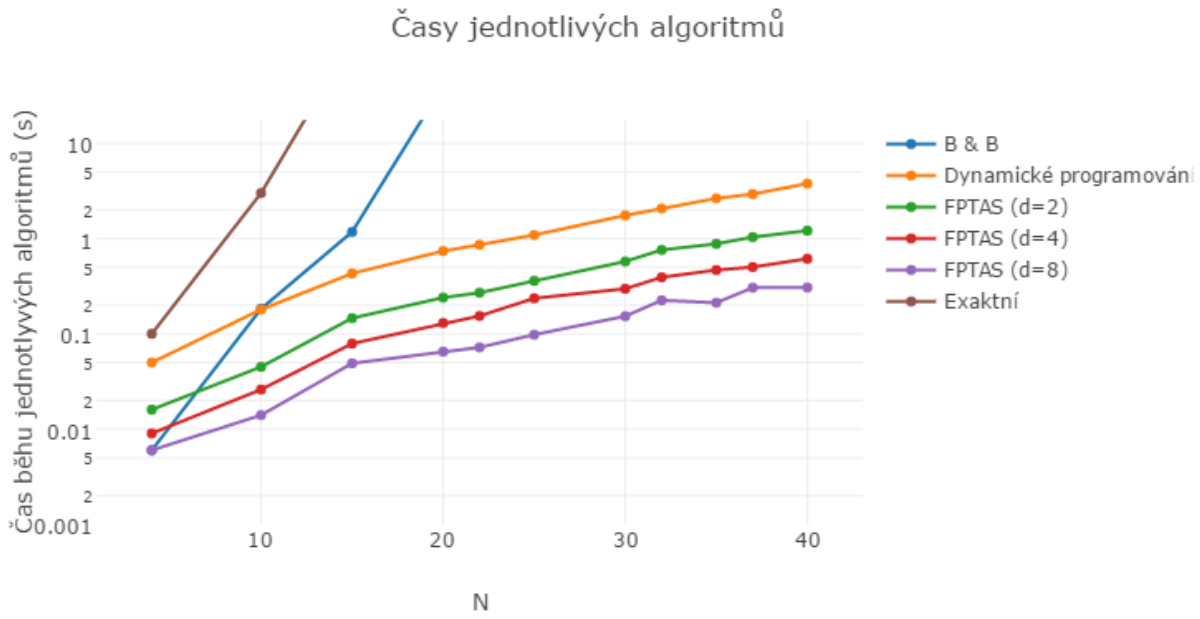
    Řešení pak nalezneme na pozici m[n,W]. Aby bylo řešení efektivní, ukládáme si mezivýsledky do tabulky.
3) Algoritmus FPTAS funguje na základě zanedbání b nejméně významných bitů. Algoritmus je následující:
    - C_M = max {c1 , c2 , …, cn }.
    - K = eps * C_m/n
    - Pro i = 1…n, nechť c_i ’ = floor(c_i/K)
    - Řešení je výstup instance n, M, c_1 ’, c_2 ’ , …, c_n ’ , w_1 , w_2 , …, w_n získaný pomocí dynamického programování s dekompozicí podle ceny

Naměřené hodnoty časů

Měření probíhalo na počítači s konfigurací s procesorem i5-2410M, Windows 10 a prostředím Python 2.7.

N	4	10	15	20	22	25	30	32	35	37	40
Naivní (s)	0.1	3	120	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
B&B(s)	0.006	0.183	1.175	39.52	166.4	n/a	n/a	n/a	n/a	n/a	n/a
Dyn. Programování (s)	0.05	0.18	0.43	0.74	0.86	1.09	1.75	2.07	2.64	2.93	3.78
FPTAS (sf=2)(s)	0.016	0.045	0.146	0.24	0.27	0.36	0.575	0.76	0.88	1.036	1.21
FPTAS (sf=4)(s)	0.009	0.026	0.079	0.13	0.154	0.236	0.297	0.392	0.467	0.502	0.6139
FPTAS (sf=8)(s)	0.006	0.014	0.049	0.065	0.072	0.098	0.153	0.225	0.212	0.305	0.306

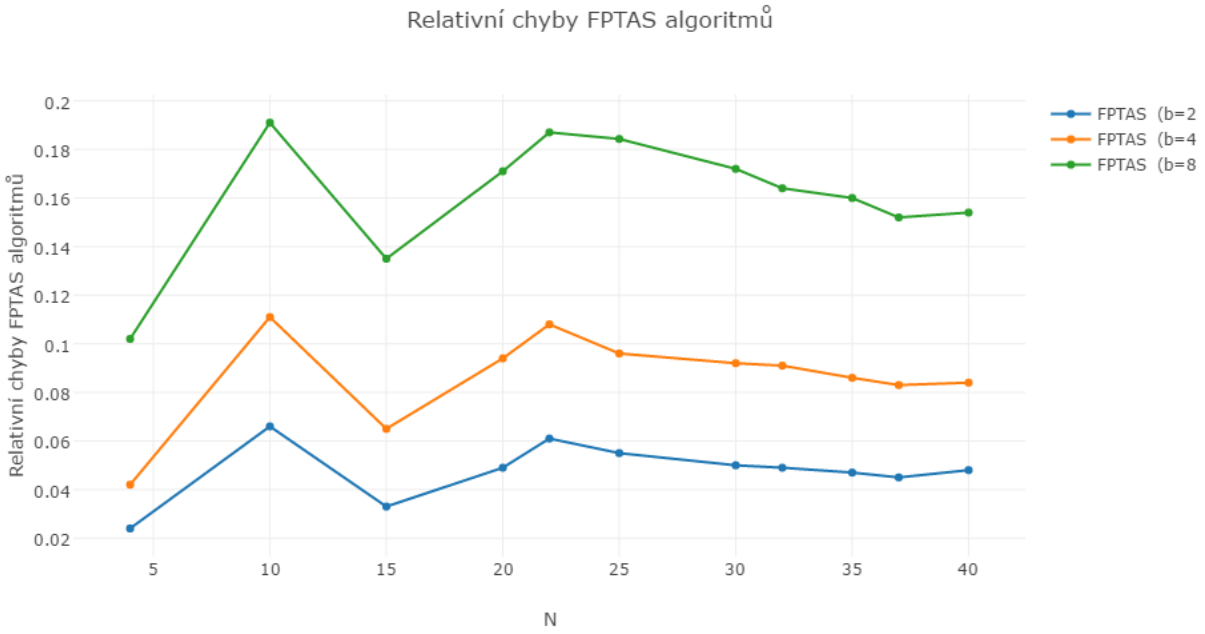
Jak ukazuje tabulka I graf, prořezávání v B&B výrazně zrychluje naivní metodu a v rozumném čase je možné dopočítat i hodnoty pro vyšší N. Vysoká asymptotická složitost ale stále neumožňuje efektivně počítat vyšší N.

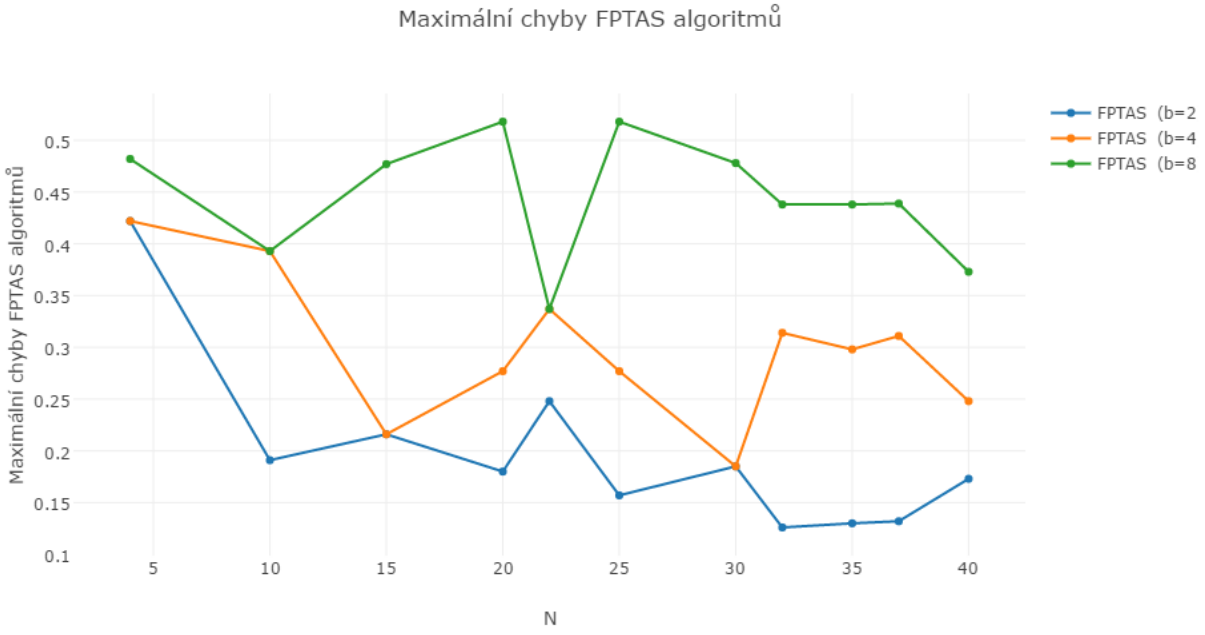


Metoda dynamického programování dale dramaticky snižuje asymptotickou složinost na O(n*W), díky čemuž je možné v daném prostřední dopočítat dataset n=40 pod čtyři vteřiny, přičemž se stále jedná o validní řešení (nic se neodhaduje, nezaokrouhluje).

Metoda PTPAS řešení dale urychluje, tentokrát ale už za cenu snížení přesnosti, tudíž se v řešení objevuje i nenulová relativní chyba. Její velikost můžeme shora omezit volbou b, tedy počtem nejméně významných bitů. Velikost relativní chyby bude maximálně eps=$(n*2^b)/C_m$, kde $C_M = \max \{c_1 , c_2 , \ldots, c_n \}$,

což, jak je vidět v grafu, je v tomto případě splněno. Z grafů je rovněž vidět, že s růstem b neroste jenom relativní chyba, ale i ta maximální relativní. Zároveň přímo úměrně klesá čas výpočtu.





Závěr

Cvičení v praxi ověřilo, že pro řešení problému batohu existují i optimalizované verze naivního algoritmu, které jsou ale stále docela pomalé a za dané situace není možné dopočítat větší datasety. Chytřejší řešení pomocí dynamického programování je znatelně efektivnější, největší dataset se podařilo

dopočítat pod čtyři vteřiny. Přičemž toto řešení stále garantuje správné řešení problému batohu. Posledním zkoušeným algoritmem je FPTAS, který řešení dále zrychluje, už ale nezachovává vlastnost garance správného řešení, nicméně je schopný garantovat maximální relativní chybu volbou b, tedy počtem nejméně významných bytů v cenách. Zároveň se zvětšující se relativní chybou se přímo úměrně snižuje čas výpočtu, od 1,21s pro d=2, až po 0,3s pro d=8.