Exaktní a heuristická řešení konstruktivního problému batohu

8. listopadu 2020, Radek Šmíd

https://github.com/Smidra/KOP-ukol-2

1 Specifikace úlohy

Cílem bylo naprogramovat konstruktivní řešení problému batohu šesti způsoby.

1. Hrubá síla
2. Metoda větví a hranic
3. Metoda dynamického programování
4. Greedy heuristika
5. Redux heuristika
6. FPTAS pseudopolynomiální aproximační schéma

Naši implmentaci pak experimentálně ověřit a vyhodnotit při výpočtu na stažených datech. Složitost výpočtu instance problému měříme v čase výpočtu.[1]

2 Rozbor řešení

Při programování vycházím z objektově orientovaného návrhu z vypracovaného předchozího úkolu. Ten byl rozšířen o měření času výpočtu v procesu pomocí python funkce *process_time()*. Tato funkce se nachází v pythonu od verze 3.3.[2]

Řešení pomocí hrubé síly a metody větví a hranic nebylo pozměneno od předchozího úkolu. Ostatní čtyři způsoby jsem naimplementoval opět jako metody třídy „KnapsackInstance“.

Dále jsem program rozšířil o automatické počítání průměrných a maximálních hodnot času a chyby. Stejně tak jako o další skript zlepšující automatizaci.

Poměr ceny a váhy pro greedy a redux heuristiku je atribut třídy věci, který se vypočítá přímo v konstruktoru. Greedy pak pouze seřadí pole věcí v instanci podle jednoho z jejích atributů. Redux heuristika využívá kódu identického s implementací greedy, akorát je rozšířena o porovnání s nejlepší věcí před uložením řešení.

Metoda dynamického programování provádí dekompozici podle ceny. Její kód je pak využit i v implementaci FPTAS.

3 Popis kostry algoritmu

Implementace algoritmu FPTAS se skládá z pěti kroků. Vstupem je požadované epsilon (e).

3.1 Příprava hodnot pro FPTAS

Nejdříve se jedná o nalezení největší váhy předmětu v instanci společně s vyřazením věcí těžších než maximální povolená hmotnost. Pokud jsou vyřazeny všechny věci, řešením jsou samé nuly a FPTAS končí. Výpočet K může být v extrémních případech zkomplikován tím, že je zadané e příliš malé. V takový moment je epsilon postupně zvyšováno, dokud není K nenulové. S vypočeným K dojde poté k redukci včech cen. To může sice vnést do výpočů drobné chyby, ale není to častý případ.

3.2 Alokace polí

Je vytvořeno pole pro dekompozici a pomocné pole pro konstrukci řešení.

[1] https://moodle-vyuka.cvut.cz/mod/assign/view.php?id=89697
[2] https://docs.python.org/3/library/time.html

3.3 Vyplnění polí

Pole dekompozice je celé proiterováno. Na počátku je pole plné nekonečen, kromě hodnoty $[0][0] = 0$. Pro každé políčko obsahující něco jiného než nekonečno se pokusíme zapsat do dvou buněk v dalším sloupci. Nejprve do řádky vyšší o cenu věci stávajícího sloupce (věc byla přidána) a pak do buňky ve stejné řádce (věc nebyl přidána). Do jednotlivých polí zapisujeme celkovou váhu všech přijatých předmětů, ale pouze tehdy pokud je menší než případná přítomná hodnota.

3.4 Nalezení řešení

Řešení se nachází v posledním sloupci tabulky. Jde o nejvýše postavenou buňku (nejlepší možná cena) která obsahuje přijatelnou váhu. Řešení musí být ještě zpět přenásobeno hodnotou K, aby odpovídalo původním cenám.

3.5 Cleanup

Pole jsou smazána, aby opakované běhy nezahlcovaly paměť OS.

4 Naměřené hodnoty a interpretace výsledků

K dispozici byly tři testovací sady: náhodná (NK), zlá cena (ZKC), zlá váha (ZKW). Všech šest algoritmů bylo stpuštěno na všech třech sadách. FPTAS dokonce třikrát se třemi hodnotami epsilon (0.1, 0.3, 0.5). Všechna data z experimentů jsou k dispozici ve složkách **calculated_xy** v repozitáři projektu.

4.1 Srovnání výpočetních časů

Pro každou sadu dat jsem vytvořil jeden graf. V každém grafu se nachází čtrnáct čar rozdělených barvou podle typu algoritmu. Křížky značí maximální naměřený čas, kolečka průměrný ze všech měřených běhů. FPTAS je uvedeno pouze s nejvyšší (0.1) a nejnižší (0.5) přesností, aby nebyly grafy přehlceny informacemi. Grafy jsou vykresleny na logaritmické ose, pro zvýšení přehlednosti.

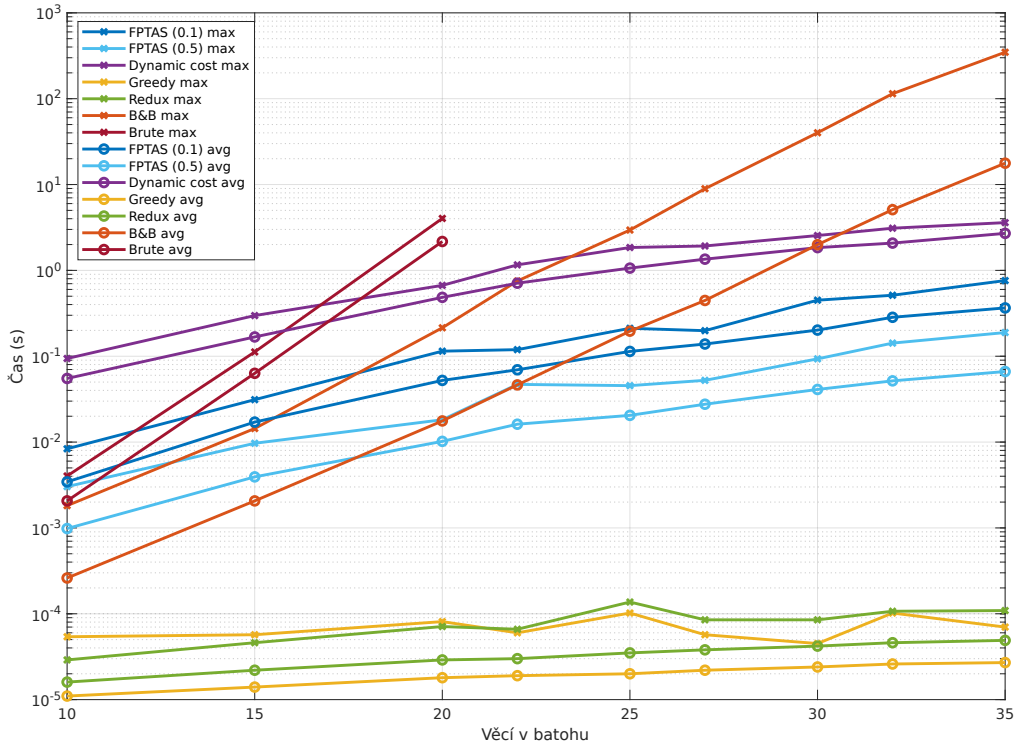

Figure 4.1 Časy výpočtu pro instance NK

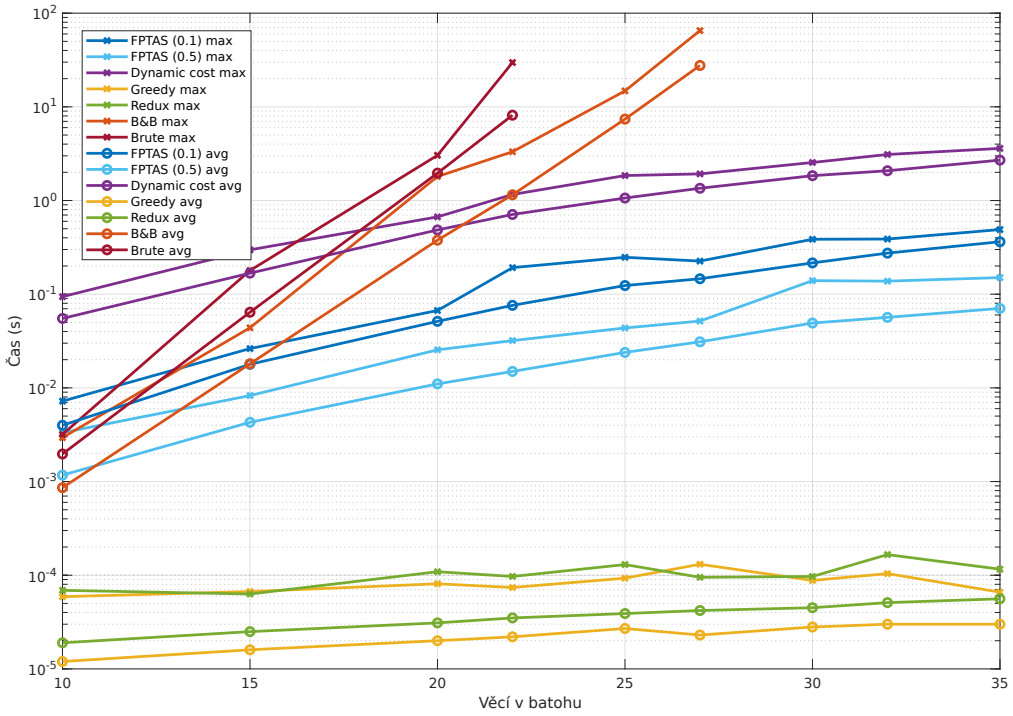

Figure 4.2 Časy výpočtu pro instance ZKC

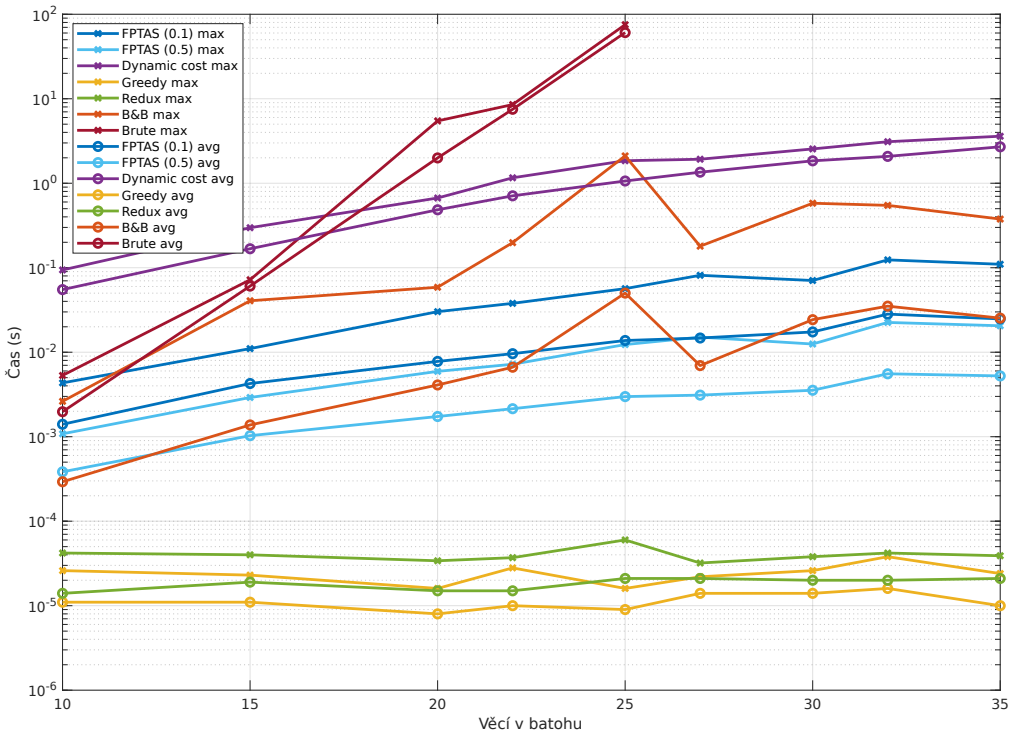

Figure 4.3 Časy výpočtu pro instance ZKW

4.2 Porovnání relativních chyb

Obdobně jako u srovnání času jsou grafy rozděleny podle typu dat. U chybovosti jsem vynechal algoritmy které nechybují (dynamic, brute, b&b). V grafu chybovosti pro ZKC splývá chybovost greedy a redux. I když zde logaritmická osa není příliš intuitivní je to jedinný způsob jak zároveň zachytit vysokou chybovost redux a jemné rozdíly mezi variantami FPTAS. Varianty FPTAS jsou zde uvedeny všechny naměřené, tedy 0.1, 0.3, a 0.5.

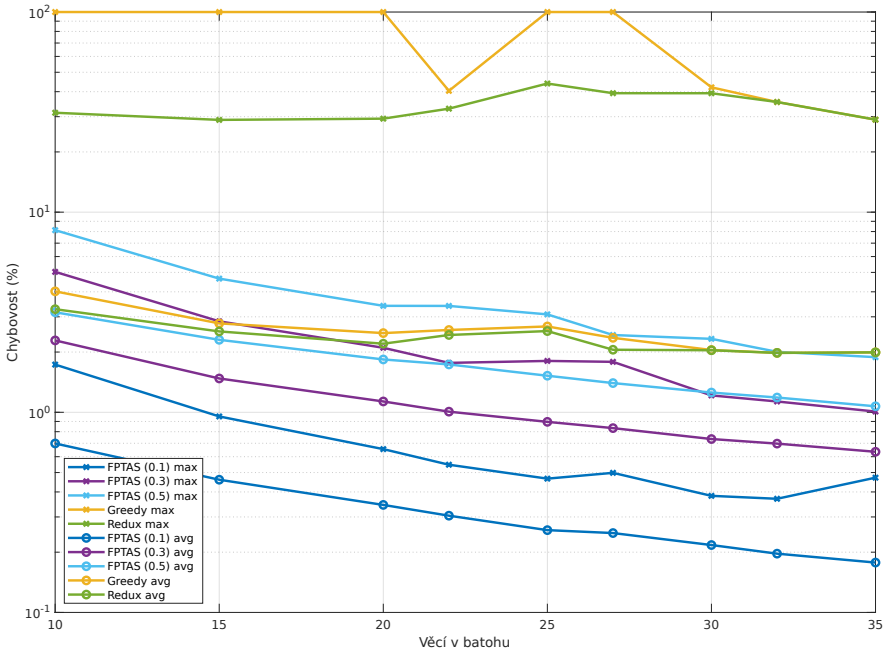

Figure 4.4 Chybovost výpočtu pro instance NK

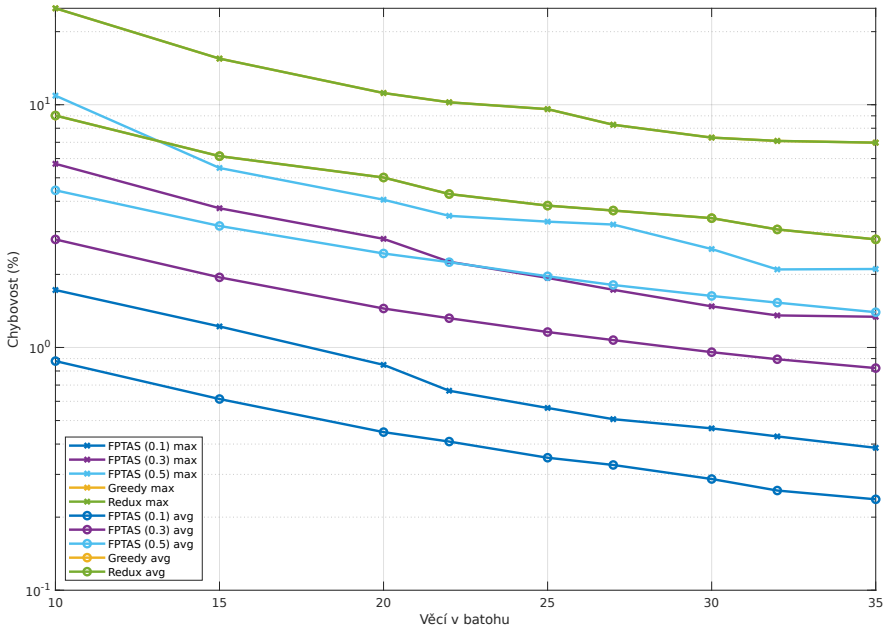

Figure 4.5 Chybovost výpočtu pro instance ZKC

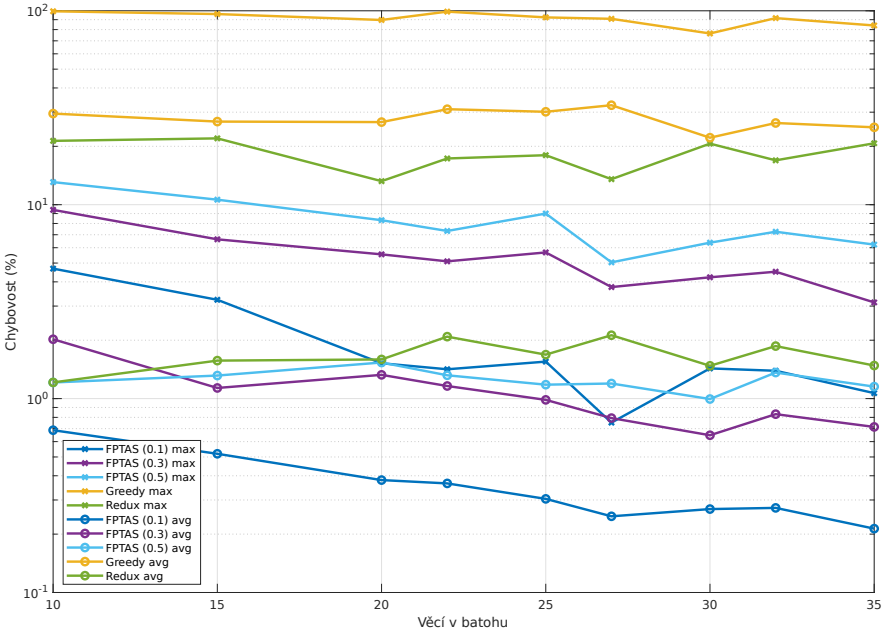

Figure 4.6 Chybovost výpočtu pro instance ZKW

5 Závěr

Zdaleka nejrychlejší výpočet nám zajistí heuristiky greedy a redux, bohužel jejich maximální chybovost je také zdaleka největší. I když v sadách NK a ZKC chybovost greedy klesá s množstvím věcí, nemusí tomu tomu tak být vždy, jak je vidět z chyb u sady ZKW. Redux je ve většině případů o trošku rychlejší než greedy, ale najdou se i výjimky (ZKC27). Co se týče rychlosti jsou si algoritmy velmi podobné. Naměřená chybovost byla identická pro sadu ZKC, protože se při výpočtu instancí ani jednou nestane, že by se zvolilo řešení vypočítané pomocí redux. Jinak je na tom redux vždy přinejhorším stejně jako greedy. Nejlepší výsledky vykazuje redux na sadě ZKW, kde má velmi dobrou průměrnou chybovost. I tam je ale jeho maximální chybovost druhá největší, hned po greedy.

Pro sady NK a ZKC jsou dle očekávání nejpomalejší hrubá síla a b&b. Ne ale pro všechny velikosti instancí. Kvůli vysoké režiji spojené s dekompozicí podle ceny (alokování pole) jsou rychlejší pro menší instance (cca 25). Speciálně ve srovnání s dynamic, které také poskytuje bezchybné výsledky.

Pro sadu ZKW to platí u hrubé síly, ale už ne pro b&b. Tam je dle naměřených výsledků metoda větví a hranic alespoň stejně tak dobrá jako dynamické programování.

FPTAS a dynamické programování vykazují obdobné chování. Čím větší zvolíme přesnost tím déle výpočet potrvá. I tak ale bude rychlejší než samotné dynamické programování. Chybovost FPTAS a dynamic pro instance NK a ZKC stabilně klesá. Čím více věcí v batohu, tím menší chyba. Také ale čím menší epsilon, tím menší chyba.

Pro všechna data se průměrná chyba se drží vždy pod hranicí stanovenou pomocí epsilon.