2. Řešení problému batohu dynamickým programováním, metodou větví a hranic a aproximativním algoritmem

Ladislav Martínek

1 Zadání úlohy

Naprogramujte řešení problému 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,
3. FPTAS algoritmem, tj. s použitím modifikovaného dynamického programování s dekompozicí podle ceny.

2 Rozbor řešení

Problém batohu a rozbor řešení pomocí hrubé síly a jednoduché heuristiky jsem podrobněji popsal v úloze č.1. Rozbor problému v této zprávě tedy nebudu rozvádět a zaměřím se především na nově metody, které byly do implementovány do testovacího programu.

Metodu hrubé síly z předchozího úkolu jsem malinko opravil, aby opravdu testovala všechny možnosti, tedy jsem opravil přerušení po přesažení kapacity batohu, i když zlepšení je nepatrné, nyní metoda hrubé síly opravdu vyzkouší všechny možnosti.

První novou metodou je metoda větví a hranic (B&B), která vylepšuje původní algoritmus o ořezávání větví, které již nemohou dosáhnout lepšího řešení, než které bylo dosaženo doposud. Pokud si uchovávám nejlepší dosažené řešení, tak při každém kroku rekurze můžu k aktuální ceně přičíst součet cen všech zbývajících předmětů a pokud cena nebude větší než současné maximum, tak mohu tuto větev ukončit. Dále lze odřezávat větve, které přesáhly maximální kapacitu batohu.

Dalším implementovaným algoritmem bylo dynamické programování. Dynamické programování je vždy spojeno buď s dekompozicí podle ceny nebo podle kapacity. Hodnoty z běhu jsou tedy ukládány do tabulky velikosti $n * M$, kde n je počet předmětů a M je závislé na dané dekompozici. Pokud je dekompozice podle ceny, je M maximální cena a hodnoty v tabulce jsou dosažené kapacity. Pokud podle kapacity, tak M je kapacita batohu a hodnoty v tabulce jsou dosažené ceny. Algoritmus se proto také nazývá pseudopolynomiální, je polynoiální ve velikosti instance, ale obsahuje parametr, který s její velikostí nesouvisí. Souvisí buď s kapacitou batohu nebo maximální cenou.

Metoda FPTAS je plně polynomiální aproximační schéma. Algoritmus je implementován na základě algoritmu dynamického programování s dekompozicí podle ceny. Algoritmus umožnuje nastavení parametru ϵ. Tímto parametrem lze měnit přesnost řešení. Pomocí ϵ lze nastavit maximální relativní chybu. Podle vzorce

$$\text{nová cena předmětu} = \text{původní cena předmětu} / \frac{\epsilon * \max(\text{cena předmětu})}{n}$$

je přepočtena cena předmětů a tím je zmenšena (při hodně malé volbě ϵ a malé maximální ceně a vetšímu n se může i zvětšit) tabulka dynamického programování a snížena paměťová i časová náročnost.

3 Popis kostry algoritmu

Základní kostra algoritmu zůstala stejná, pořád umožnuje obě použití, jak *solve*, tak *stats*. Přibyly boolovské přepínače pro každý algoritmus, tedy zdali má být testován a pro FPTAS přepínač očekává seznam epsilonů, které má otestovat. Měření času je stále stejné a počet opakování je prováděno v závislosti na rychlosti výpočtu dané instance.

Algoritmus provede měření pro všechny uvedené algoritmy zkontroluje zdali jsou optima stejná u algoritmů, která mají optima dosáhnout a spočítá relativní chyby u aproximačních algoritmů ($\frac{c(OPT)-c(APX)}{c(OPT)}$). Data jsou ukládána do do csv souboru pro každou instanci.

Všechna měření jsou vždy prováděna i s předzpracováním. Jediné co algoritmus dostává jsou parametry (např. max kapacita nebo ϵ) a dvě pole. Jedno s cenami a druhé s hmotnostmi.

3.1 Metoda větví a hranic (B&B)

Metodu větví a hranic jsem implementoval do rekurze a při každém přidání předmětu je kontrolováno zdali řešení nepřekročilo maximální váhu a nebo zdali může překonat prozatimní nejlepší řešení. Součet zbývajících cen provádím při každé kontrole. Tento krok by mohl být zrychlen předpočítáním jednotlivých součtů do pole vždy pro x přeskočených hodnot.

3.2 Dynamické programování

Zde jsem implementoval obě dekompozice. V dekompozici podle kapacity vytvořím pole pomocí numpy o velikosti $(n+1) * (\text{kapacita batohu} + 1)$ a inicializované nulami. V každém kroku do dané buňku ukládám aktuální dosaženou cenu, pokud již v buňce cena je a je vyšší, oříznu tuto větev.

V dekompozici podle ceny vytvořím pole o velikosti $(n+1) * (\text{součet cen} + 1)$ a inicializované maximální hodnotou. V každém kroku do dané buňku ukládám aktuální dosaženou kapacitu, pokud již v buňce hodnota je a je nižší, oříznu tuto větev.

Rekonstrukci řešení provedu zpětně od optimálního, tak že se koukám do vedlejší buňky zdali je hodnota stejná, pokud ano předmět na dané pozici nebyl přídán a pokud ne posunu se v tabulce a patřičný počet kroků a předmět přidám.

3.3 FPTAS

FPTAS je implementováno stejně jako dynamické programování s dekompozicí podle ceny. Jediná změna proběhla v předzpracování cen a tabulky pro dynamické programování. Ceny jsou přepočítány podle vzorce z sekce 2 a podle nových cen vytvořena nová tabulka.

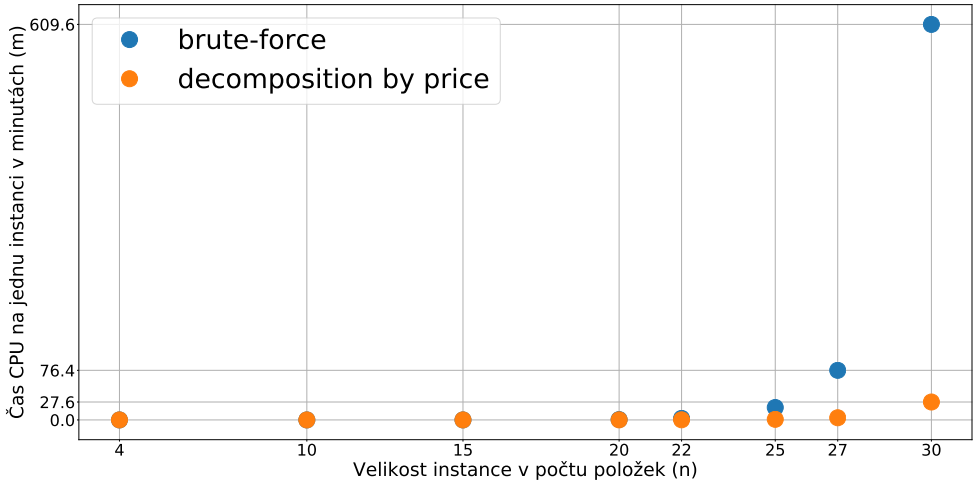
Při rekonstrukci řešení postupuji stejně jako u dynamického programování, jen pokud předmět do řešení dávám tak přičítám původní hodnotu na dané pozici.

4 Experimenty

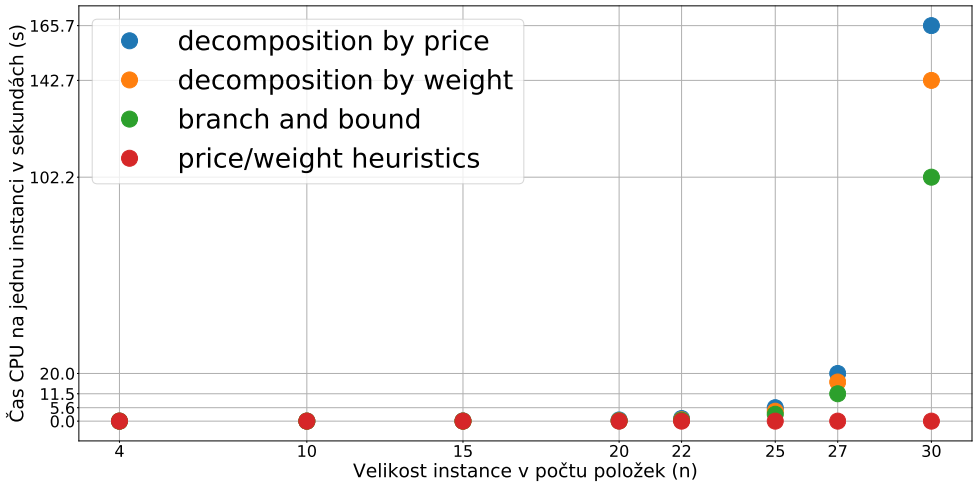
Experimenty jsem prováděl v režimu jednoho vlákna na starším datovém serveru v podobě starého notebooku, který v době výpočtu nebyl používán. Výsledky tedy nejsou ovlivněny jinými běžícími programy. Procesor na testovacím stroji: *Intel Pentium T3400 (2 cores). Taktován na 2.16 GHz s 1 MB cache*. Měření času CPU probíhalo v knihovně *timeit* s několika násobným průchodem pro menší instance.

4.1 Srovnání exaktních metod pro řešení problému batohu

Na grafu 1 jsem zanesl kromě brute-force řešení také řešení dekompozicí podle ceny. Je patrné, že složitost řešení s roustoucím n pro brute-force metodu roste velice rychle,tedy řešení pomocí dynamického programování umožní velice uspořit čas.



Obrázek 1: Brute-force ve srovnání s dynamickým programováním s dekompozicí podle ceny. Časová náročnost. Na grafu jsou průměrné hodnoty.



Obrázek 2: Časová náročnost obou způsobů dynamického programování, metody branch and bound a jednoduché heuristiky z předchozího úkolu. Na grafu jsou průměrné hodnoty.

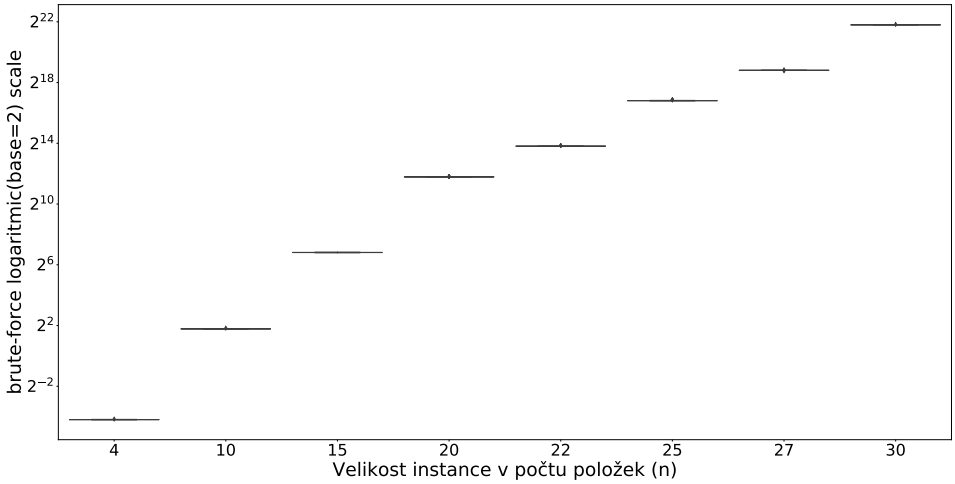
Na dalším grafu 2 jsem do grafu zanesl obě dekompozice, metodu větví a hranic a také heuristiku cena/hmotnost pro srovnání. U těchto metod mě velice zajímalo jaká je variance řešení. Pří porovnání průměrů a mediánu, byl v datech velký rozdíl. Z tohoto důvodu jsme se rozhodl vytvořit krabicové grafy (box ploty). Grafy zobrazují data podle kvartilů. Střední část je ohraničená 1. a 3. kvartilem. Jednotlivé body jsou odlehlé hodnoty. Celkově lze z box plotů vyčíst variabilitu dat.

Všechny boxploty jsou vytvořeny s logaritmickým časem pro lepší vizualizaci, který však nemá vliv na variabilitu dat.

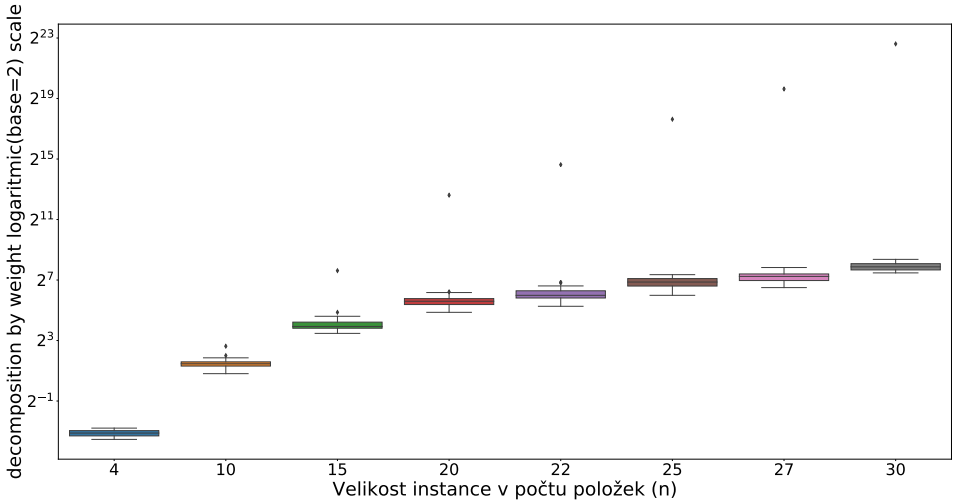
Na grafu 3 je vidět box plot vytvořený pro časy brute-force řešení. Je vidět, že výsledné hodnoty jsou blízko u sebe, tedy očekávaná je i nízka variance při výpočtu na různých instancí dat.

Další dva grafy 4 a 5 jsou výsledky z měření dynamického programování. Jak pro dekompozici podle ceny, tak i podle kapacity. Na tomto grafu jsou zajímavé body, které jsou vidět nad jednotlivými boxy. Tyto odlehlé hodnoty jsou "těžké"instance problému, pro které trval výpočet dlouho oproti obvyklé době. Je vidět, že výpočet na těchno instancích trvá velmi dlouho. Dalšími měřeními by bylo zajímavé zjistit, jaké specifikace má problém, který je pro dekompozici složitý.

Na grafu 6 je vidět boxplot pro metodu větví a hranic. I zde se nacházejí těžké instance problému. Dále je z grafu patná větší variance časů řešení, kde pro nějaké problémy je metoda velice rychlá,



Obrázek 3: Krabicový graf popisující variabilitu měření metody hrubé síly vždy pro 50 instancí problému.



Obrázek 4: Krabicový graf popisující variabilitu měření metody dekompozice podle kapacity vždy pro 50 instancí problému.

ale existuje mnoho instancí, ve kterých nedojde k požadovanému prořezání. Opět by bylo zajímavé najít tyto optížné instance a nebo je umět například generovat.

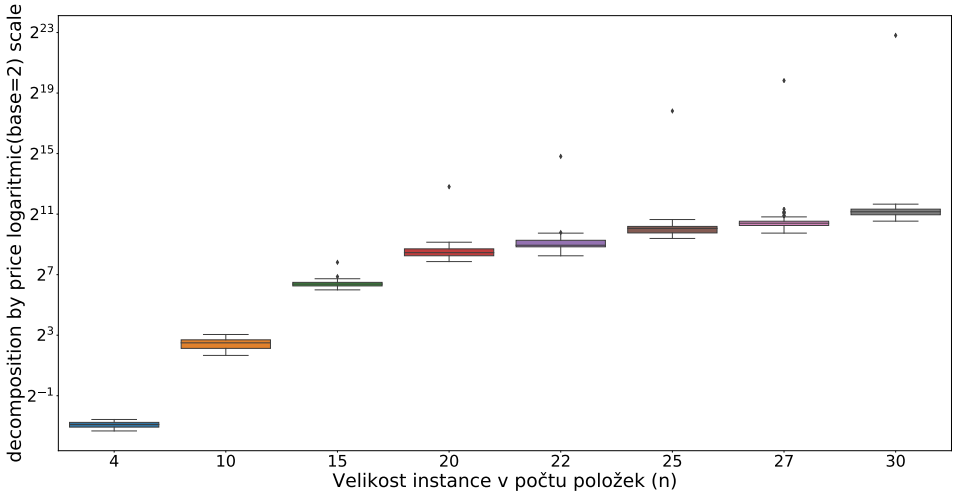
I přes horší časy při experimentech má dynamického programování výhodu, že má mnohem menší varianci výpočetních časů.

4.2 FPTAS meření času

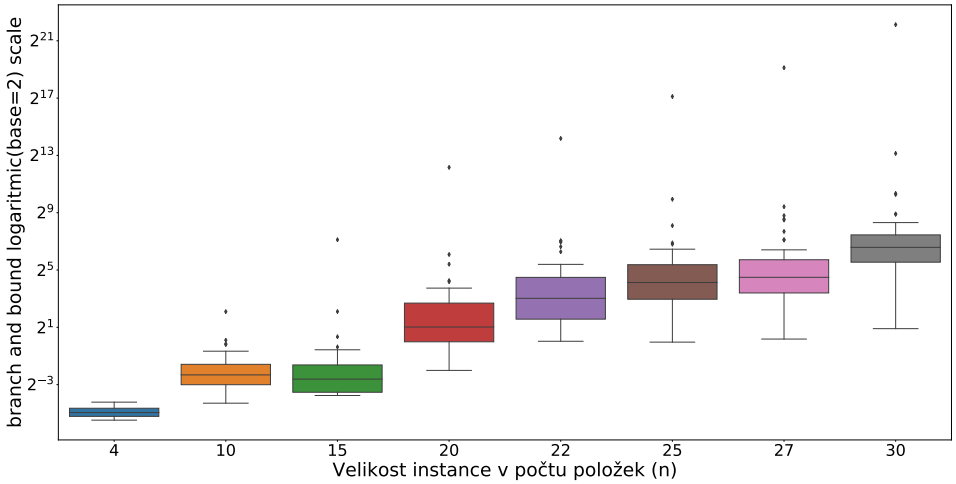
FPTAS jsem otestoval s různými nastaveními pro maximální povolenou chybu. Zde jsem časové grafy uvedl jak pro průměr, tak pro medián, po předchozím pozorování na krabicových grafech. Je vidět, že výsledný čas se přesně odvíjí od nastavené chyby. Velkou výhodou algoritmu je tedy možost nastavení maximální chyby.

Na grafech 7 a 8 jsou vidět výsledky měření pro různá ϵ a do grafu byly pro srovnání přidány metody dekompozice podle ceny a metoda větví a hranic. Je vidět, že změnou parametru lze jednoduše měnit výpočetní náročnost.

Na těchto měření jsou také zajímavé boxploty. Například pro $\epsilon = 0.5$ na grafu 10 se zde stále vyskytují dané "těžké"problémy. Ale při volbě $\epsilon = 0.95$ na grafu 9 se již pro $n = 27$ daný problém nevyskytuje a tento skok bude patrný i na grafu závislosti ϵ a času. Tedy aproximační schéma



Obrázek 5: Krabicový graf popisující variabilitu měření metody dekompozice podle ceny vždy pro 50 instancí problému.



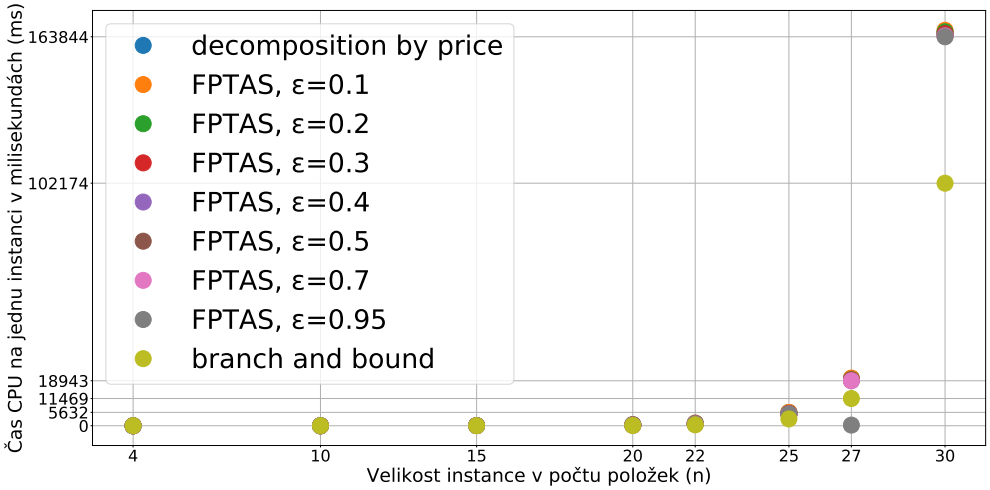
Obrázek 6: Krabicový graf popisující variabilitu měření metody větví a hranic vždy pro 50 instancí problému.

dokázalo problém dostatečně zjednodušit pro rychlé řešení avšak na úkor negarantování nějaké přiměřené hranice chyby. Toto je nejvíce patrné pro $\epsilon = 0.99$ na grafu 11. V těchto případech nám však algoritmus již negarantuje přijatelnou relativní chybu.

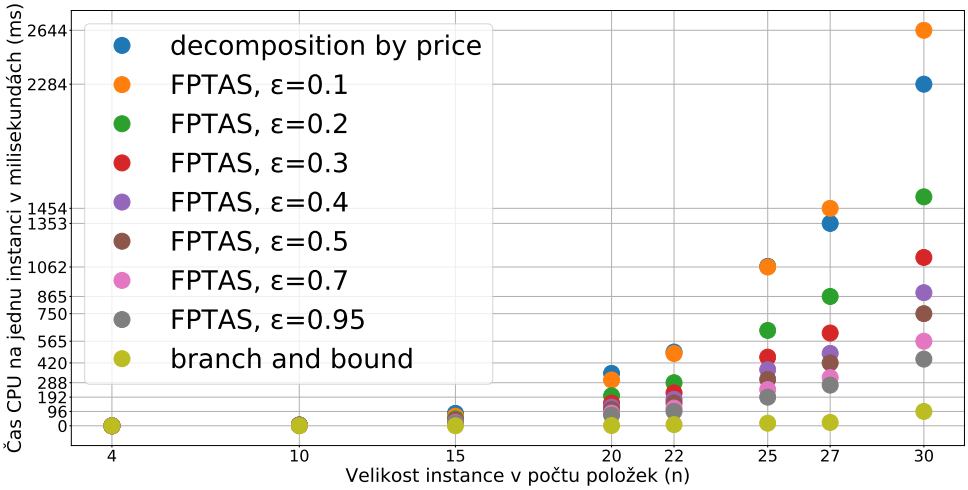
Na dalším grafu jsem zobrazil závislost časové složistosti na velikosti povolené chyby. Zde jsem uvedl graf 12 této závisloti na velikosti instance 25. Na dalším grafu 13 je stejná závislost pro velikost instance 27, kde došlo k aproximování těžkých problémů pro vysoké hodnoty ϵ.

4.3 FPTAS meření relativní chyby

Na grafu 14 je vidět závislost maximální chyby pro různě zvolenou maximální chybu v algoritmu FPTAS. Meřená chyba nikdy nepřekonala stanovenou maximální chybu. Na dalším grafu 15 je nejvyšší hodnota z grafu 14 dána do srovnání s heuristikou cena/hmotnost. Zde je vidět, že heuristika i pro zvolenou velkou maximální chybu FPTAS, vykazuje vetší chybu než algoritmus FPTAS pro instance v testovací sadě. Na posledním grafu 16 je vidět závislost maximální chyby na ϵ pro vybraná n. Je vidět, že skutečná chyba je mnohem menší než předpokládaná maximální relativní chyba.



Obrázek 7: Časová náročnost pro různé volby ϵ v FPTAS ve srovnání s dekompozicí a metodou branch and bound. Hodnoty jsou uvedeny pro průměr.



Obrázek 8: Časová náročnost pro různé volby ϵ v FPTAS ve srovnání s dekompozicí a metodou branch and bound. Hodnoty jsou uvedeny pro medián.

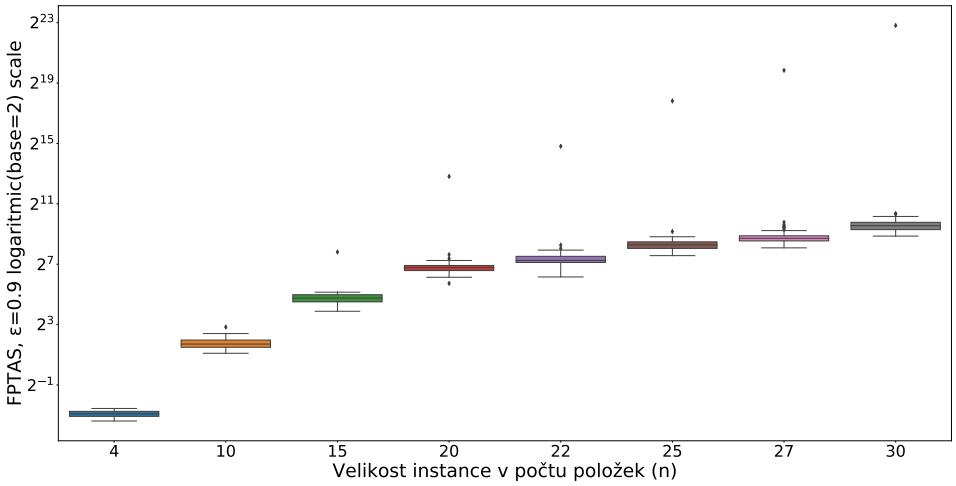
5 Závěr

Během měření se podařilo zjistit, metody B&B a dynamického programování dosahují velkého rozdílu ve složitosti oproti metodě hrubé síly. Dále, že dekompozice podle kapacity dosahuje nižších časů než dekompozice podle ceny. Metody dynamického programování mění složitost, která již není závislá exponenciálně na počtu předmětů, ale je ovlivněna hodnotami dané instance.

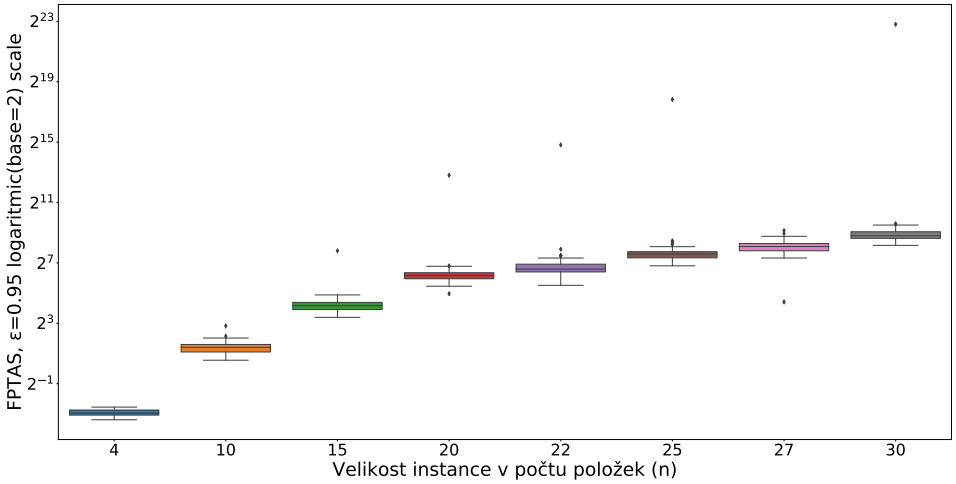
Avšak z exaktních metod byla nejrychlejší metoda větví a hranic (měla by mít stále exponenciální složitost), ale také tato metoda měla velký rozptyl meřených časů pro různé instance. Při přištích experimentech by bylo zajímavé metody dynamického programování vylepšit o metody B&B a zkusit naměřit výsledky.

Při měření jsem také narazil na fakt, že pro testované metody existují instance, pro které trvá výpočet velmi dlouho. Pro různé instance tedy může být optimální jiný algoritmus, který jí vyřeší rychleji nebo algoritmus, který ji vyřeší s minimální chybou.

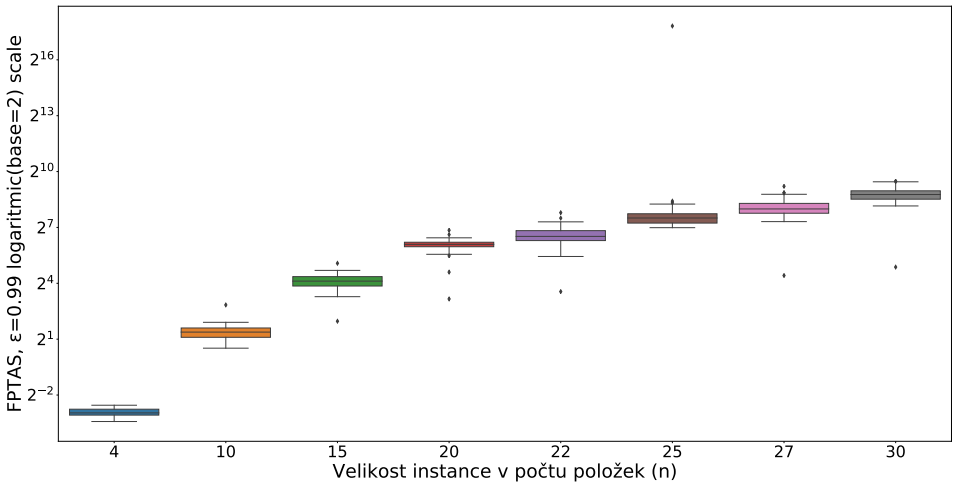
Plně polynomiální aproximační schéma FPTAS bylo zajímávé v tom, že jsme mohli měnit maximální povolenou relativní chybu a tím ovlivňovat časovou náročnost. Proto při volbě tohoto algoritmu je důležité zvážit požadavky na chybu a na rychlost a zvolit správné nastavení.



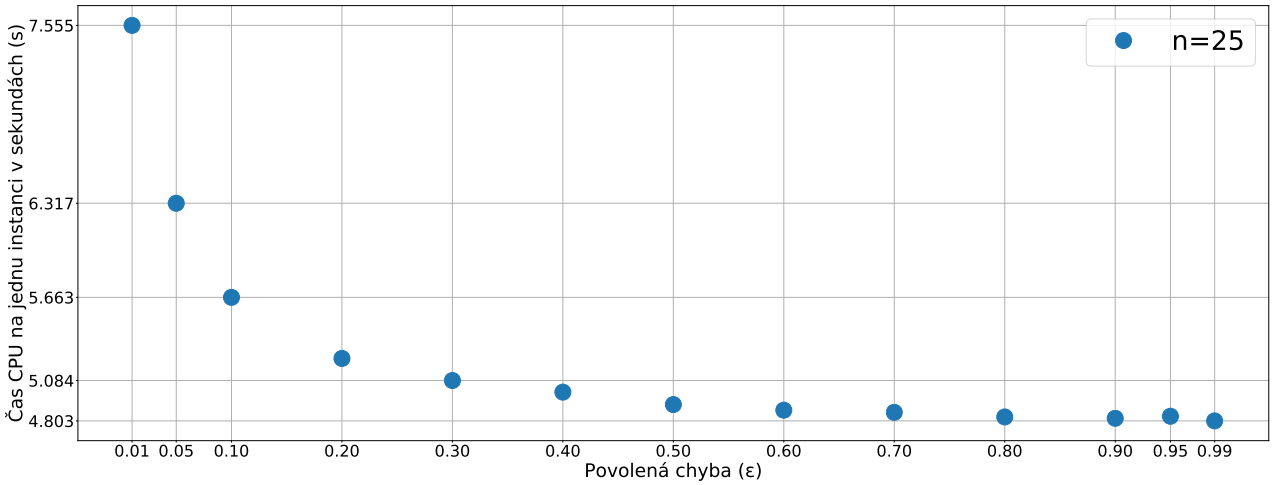
Obrázek 9: Krabicový graf popisující variabilitu měření metody FPTAS $\epsilon = 0.5$ vždy pro 50 instancí problému.



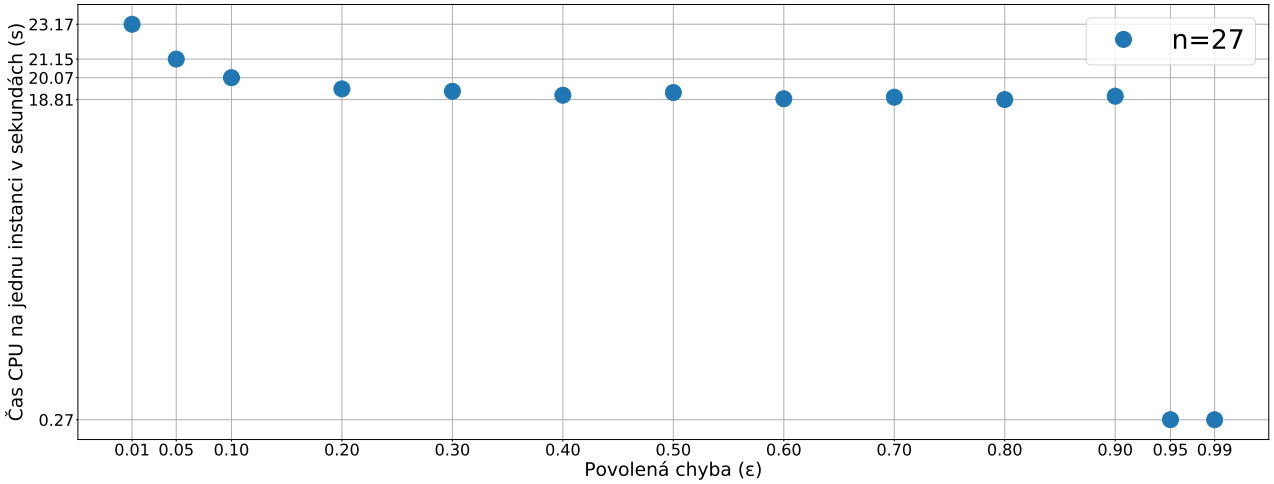
Obrázek 10: Krabicový graf popisující variabilitu měření metody FPTAS $\epsilon = 0.95$ vždy pro 50 instancí problému.



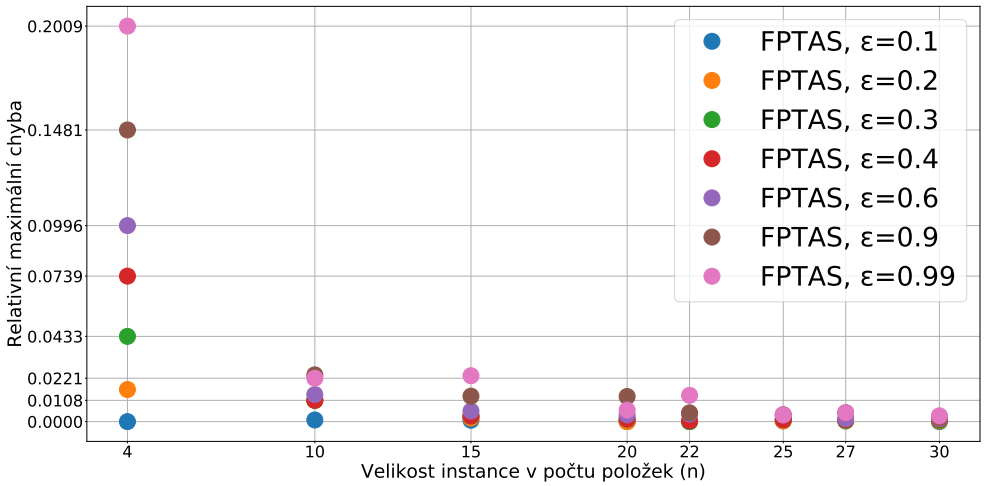
Obrázek 11: Krabicový graf popisující variabilitu měření metody FPTAS $\epsilon = 0.99$ vždy pro 50 instancí problému.



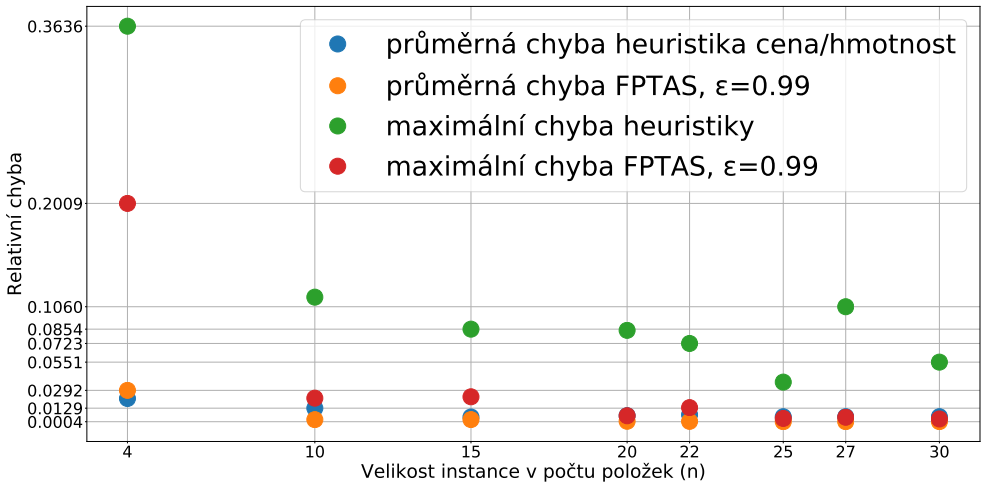
Obrázek 12: Graf ukazující závislost výpočetního času a zvolené maximální chyby ϵ pro velikost instance 25.



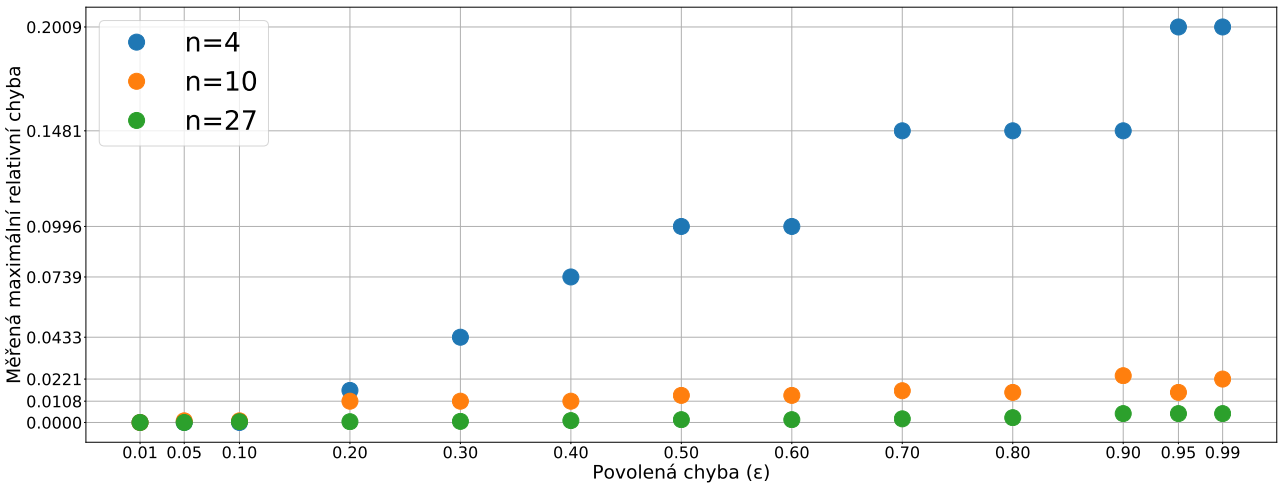
Obrázek 13: Graf ukazující závislost výpočetního času a zvolené maximální chyby ϵ pro velikost instance 25.



Obrázek 14: Graf ukazující závislost maximální relativní chyby na velikosti instance pro několik zvolených maximálních chyb ϵ.



Obrázek 15: Graf ukazující závislost maximální relativní chyby na velikosti instance pro vybranou maximální chybu ve srovnání s chybou heuristiky cena/hmotnost.



Obrázek 16: Graf ukazující závislost relativní chyby a zvolené maximální chyby ϵ pro vybrané velikosti instance.