Problém Batohu

Specifikace úlohy

Je dáno:

- celé číslo n (počet věcí)
- celé číslo M (kapacita batohu)
- konečná množina $V = \{v_1, v_2, \dots, v_n\}$ (hmotnosti věcí)
- konečná množina $C = \{c_1, c_2, \dots, c_n\}$ (ceny věcí)

Sestavte množinu $X=\{x_1, x_2, \dots, x_n\}$, kde každé x_i je 0 nebo 1, tak, aby platilo: $v_1x_1 + v_2x_2 + \dots + v_nx_n <= M$ (aby batoh nebyl přetížen) a výraz $c_1x_1 + c_2x_2 + \dots + c_nx_n$ nabýval maximální hodnoty pro všechny takové množiny (cena věcí v batohu byla maximální).

Varianty řešení

Variant řešení je několik. Řešení hrubou silou, heuristikou nebo dynamickým programováním. Také je možné jednotlivé metody optimalizovat (hrubou sílu pomocí metody větví a hranic a dynamické programování pomocí FPTAS algoritmu). Řešení hrubou silou zkouší všechny možnosti a vybere z nich tu nejlepší. Řešní metodou větví a hranic postupuje stejně, ale nepočítá ty možnosti, u kterých je dopředu známo, že na nejlepší řešení nedosáhnou. Řešení pomocí heuristiky je více. První přidává věci do batohu pouze na základě jejich váhy, druhé naopak pouze podle jejich ceny. Nejlepší možností je přidávat věci podle poměru cena/váha. Řešení dynamickým programováním přistupuje k úloze úplně jinak a původní exponenciální složitost převádí na lineární složitost, ale vzhledem k součtu cen věcí v batohu. Proto se tento algoritmus nazývá pseudopolynomiální. Tuto metodu lze dělat pomocí dekompozice podle ceny nebo podle hmotnosti. Já jsem zvolil dekompozici podle ceny, abych mohl využít algoritmu FPTAS. Tento algoritmus zanedbává b nejnižších bitů cen a tím zmenšuje velikost výsledné tabulky.

Postup řešení

Řešení hrubou silou spočívá v tom, že se vygenerují všechny povolené kombinace, u každé se spočte cena a kombinace s nejlepší cenou se vezme jako výsledek úlohy.

Popis řešení

hrubá síla

Pro řešení hrubou silou používám binární vektory. Nejprve si vypočítám číslo $n = 2^{pocet\_veci}$ a potom si pro každé číslo od 0 do n vypočtu jeho binární reprezentaci (převedu ho do binárního tvaru). Pokud dané binární číslo má na pozici i číslo 1, tak ho přidám do výsledku, pokud má 0, tak ne. Pokud cena přesáhne kapacitu batohu, tak končím výpočet pro dané číslo n. Nakonec spočtu cenu věcí v batohu a zapamatuji si nejlepší z zatím zjištěných cen.

heuristika

Pro řešení pomocí heuristiky jsem zvolil poměr cena/váha. Pro každou věc jsem vypočetl poměr cena/váha. poté jsem věci seřadil a postupně přidával do batohu, dokud byl součet hmotností menší než kapacita batohu M.

metoda větví a hranic

V metodě větví a hranic jsem musel přepsat celý algoritmus hrubé síly, protože původně jsem procházel všechny možnosti pomocí generování binární reprezentace čísel. V této metodě jsem se chtěl vyhnout rekurzi a proto jsem použil zásobník na který si ukládám nezpracované větve stromu řešení. Základem zlepšení algoritmu je zahození těch větví, které nemají šanci na úspěch. Zahazuji větve které mají hmotnost větší než je kapacita batohu a také ty, u kterých je jejich maximální cena menší, než je nejlepší cena, kterou jsem již vypočetl. V každém kroku si vyzvednu jednu variantu ze zásobníku, pokud je jeho cena lepší než aktuálně nejlepší cena, prohlásím ho za zatím nejlepší řešení, vygeneruji jeho potomky, zjistím jestli mají šanci stát se lepším řešením a pokud ano, tak je přidám na zásobník. V opačném případě je zahodím a s nimi i všechna řešení od nich odvozené. Původně jsem cenu řešení a maximální možnou cenu dané instance zjišťoval v každém kroku sečtením cen všech jejích prvků. Této lineární metody jsem se ale zbavil a nyní si při vytvoření nového prvku aktualizuji jeho ceny v konstantním čase. Pokud do daného řešení prvek přidávám, jeho maximální cena se nezmění, ale vzroste jeho cena o cenu přidávaného prvku. Naopak pokud prvek nepřidávám, jeho cena se nezmění ale maximální cena klesne, opět o cenu prvku, který nepřidávám.

dynamické programování

V této metodě vyplňuji tabulku (o rozměrech: počet věcí * suma cen) čísel. Tabulka má na nultém řádku `0` a v nultém sloupci nekonečno (`Integer.MAX_VALUE`). V ostatních buňkách je vždy minimum z čísla vlevo od této buňky a čísla na pozici `w[i - c[j-1]][j-1] + v[j-1]` (pokud tato pozice existuje v tabulce, jinak je zde opět nekonečno). Po vyplnění této tabulky procházím její poslední sloupec a hledá se řešení (první hodnota, jejíž hmotnost je menší rovna než kapacita batohu). Poté se již posunuji po sloupcích na začátek a pokud si dvě sousední hodnoty neodpovídají, přidám prvek do řešení a skočím na řádek o velikost jeho ceny níž.

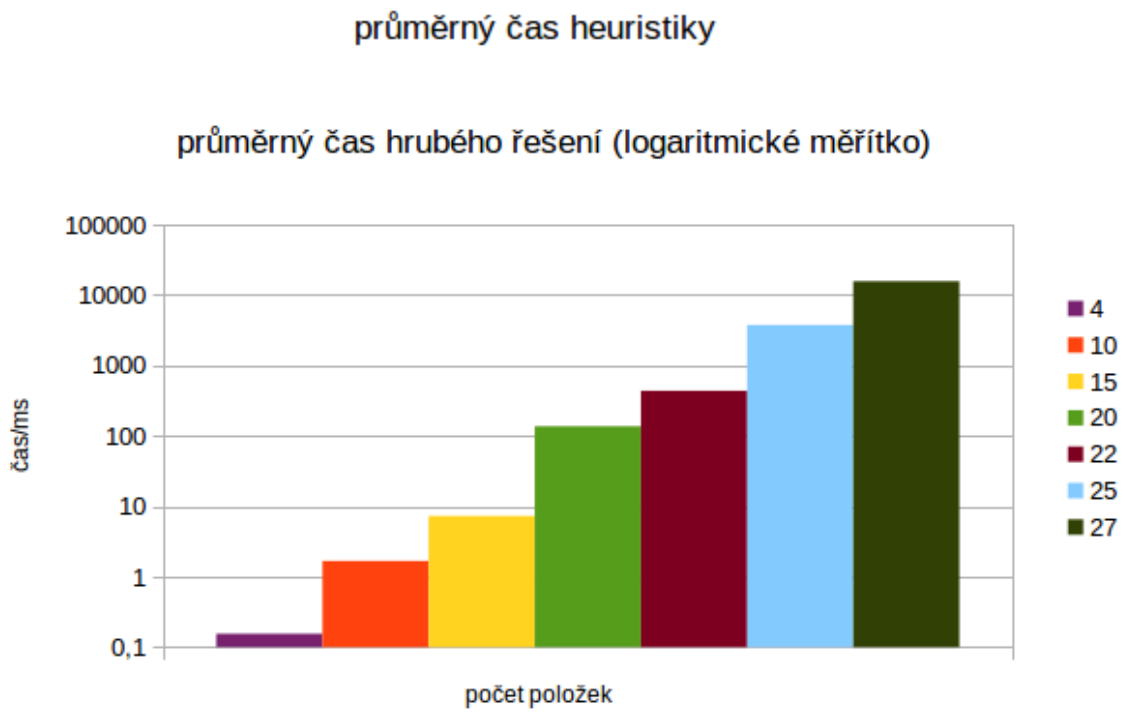
FPTAS algoritmem

Pro FPTAS algoritmus jsem použil stejný algoritmus dynamického programování, pouze jsem jeho cenu vydělil nějakým násobkem `2`. Tudíž se zmenšila tabulka řešení, ale řešení již není přesné. Výsledky lze pozorovat z přiložených grafů.

Naměřené výsledky

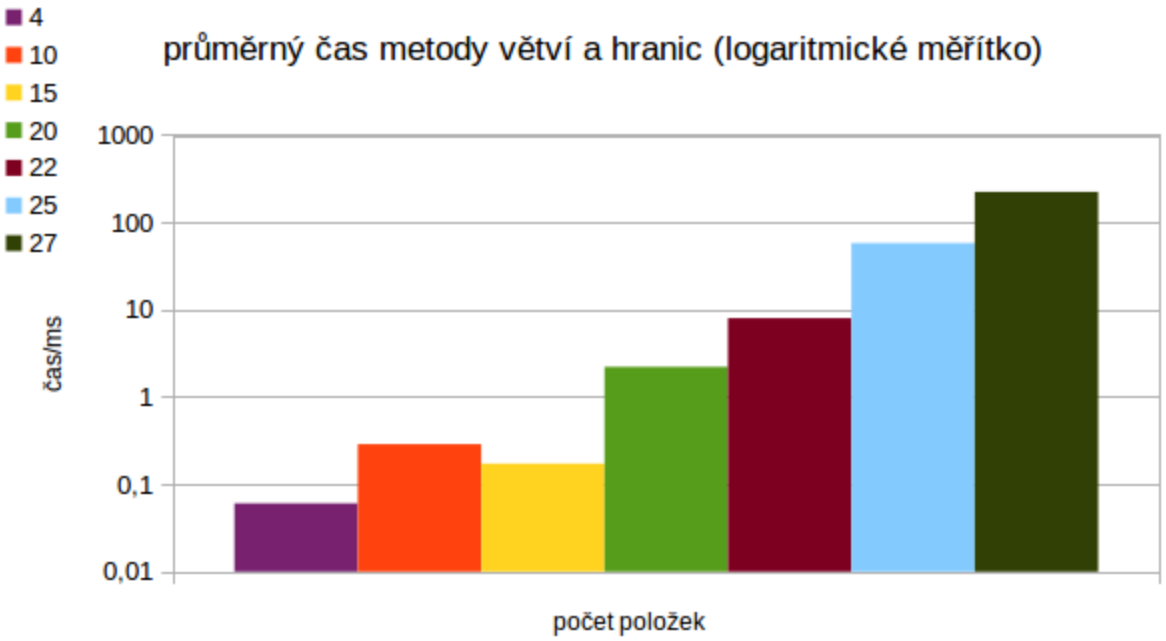
počet hodnot	čas heuristického výpočtu / µs	čas hrubé síly / ms	čas metody větví a hranic / ms	čas dynamického prog. / ms
4	0,4342	0,0690	0,0603	0,1505
10	0,4278	0,6494	0,2870	0,1740
15	0,6548	3,8400	0,1720	0,2585
20	0,8968	122,2189	2,2090	0,3818
22	0,8718	514,6400	8,0002	0,4695
25	1,0102	4 316,2806	57,7655	0,5531
27	1,1060	17 723,0777	222,5748	0,6099

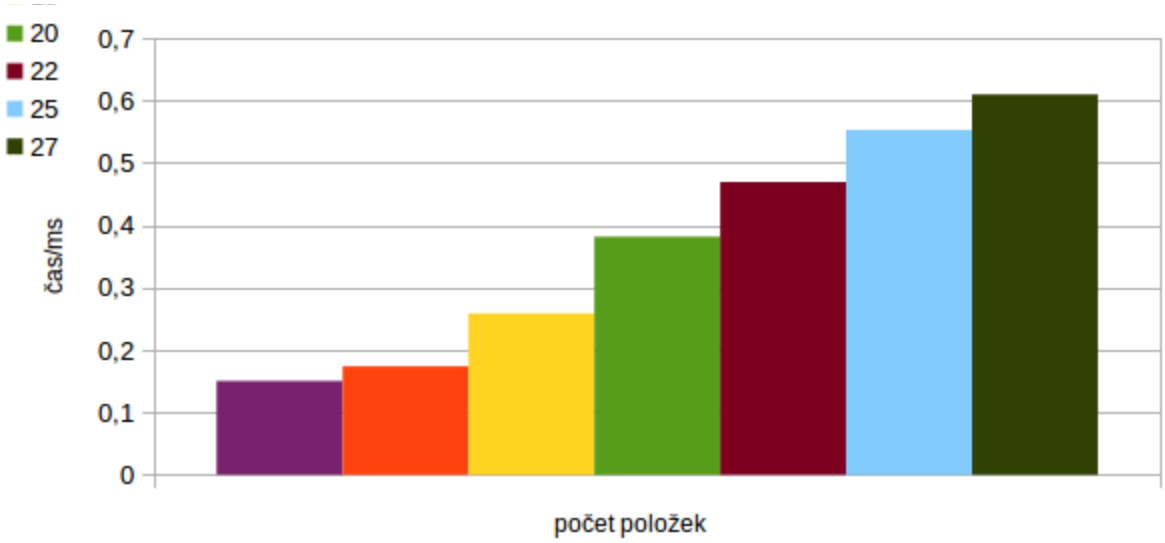
Jak je vidět na následujícím grafu, čas při počítání úlohy pomocí heuristiky roste přibližně lineárně. To je podle očekávání.



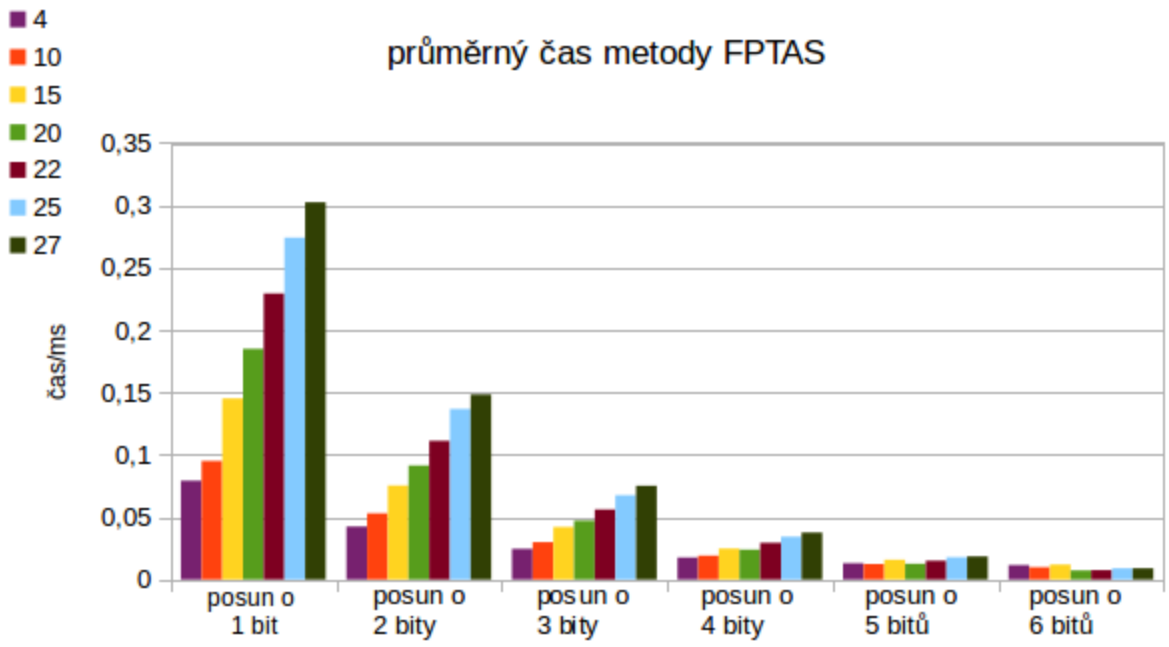
Na druhém grafu je vidět, že čas hrubého řešení roste exponencielně s velikostí řešené úlohy. Pro lepší zobrazení jsem v grafu zvolil logaritmické měřítko, protože při použití klasického měřítka byly vidět pouze poslední 3 sloupce.

Na třetím grafu je zobrazen čas pomocí metody dynamického programování. Opět se jedná o téměř lineární růst.
Na následujícím grafu je čas pomocí metody větví a hranic. Pro tento graf jsem zvolil opět logaritmické měřítko. Je tedy vidět, že stejně jako u brute force metody, roste čas exponenciálně, jeho hodnoty jsou ale až 100 menší (u instance s 27 prvky).

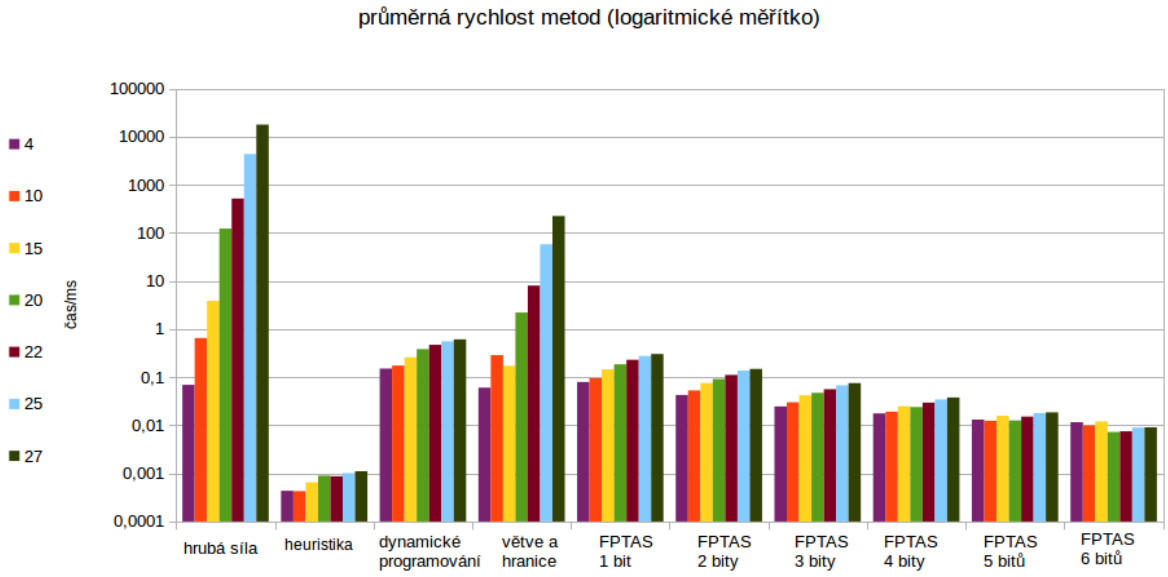


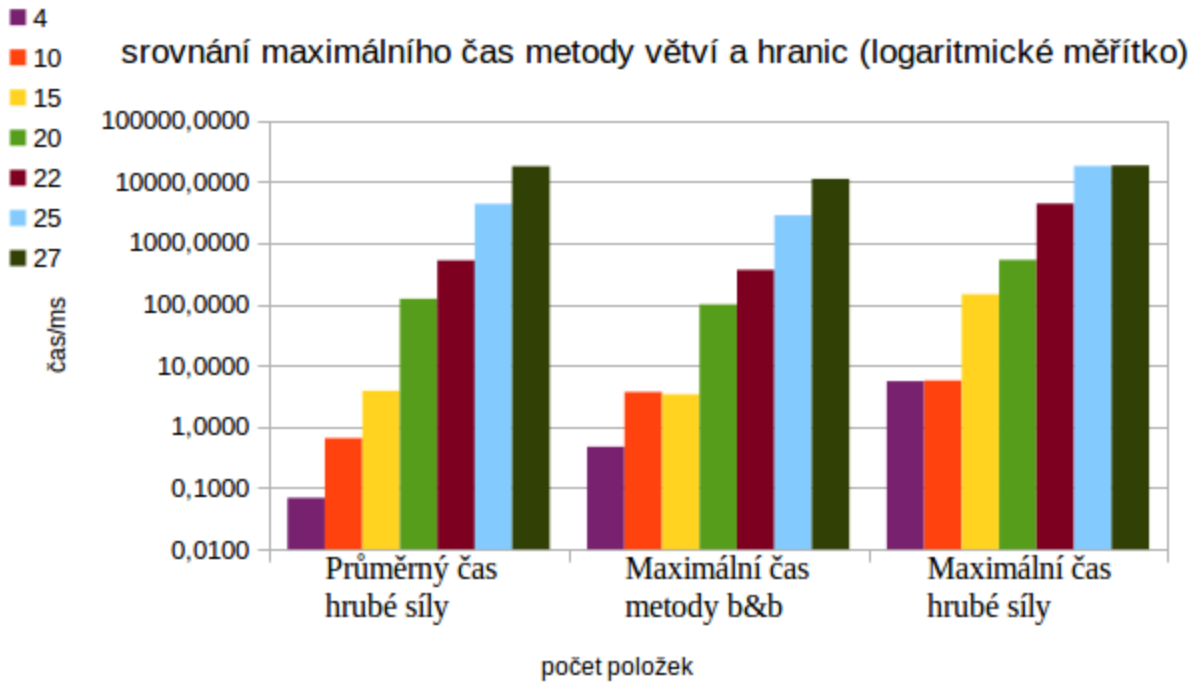


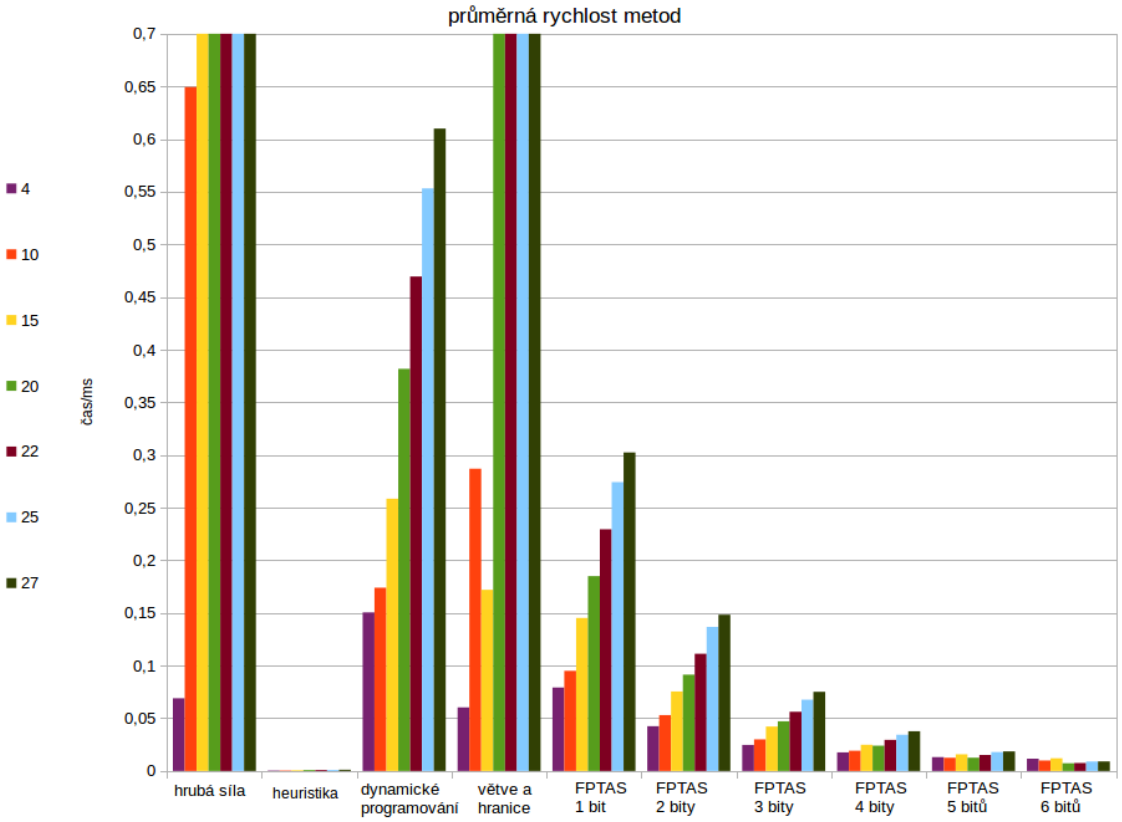
Na dalším grafu jsou časy metody FPTAS s posunem o 1 až 6 bitů. Je vidět, že zmenšením cen na polovinu, klesne výpočetní čas také přibližně na polovinu. Růst času s velikostí instance je přibližně lineární.



Na závěr uvádím srovnání časů všech metod. Nejprve v grafu s logaritmickým měřítkem a poté i s normálním, pro lepší představu.

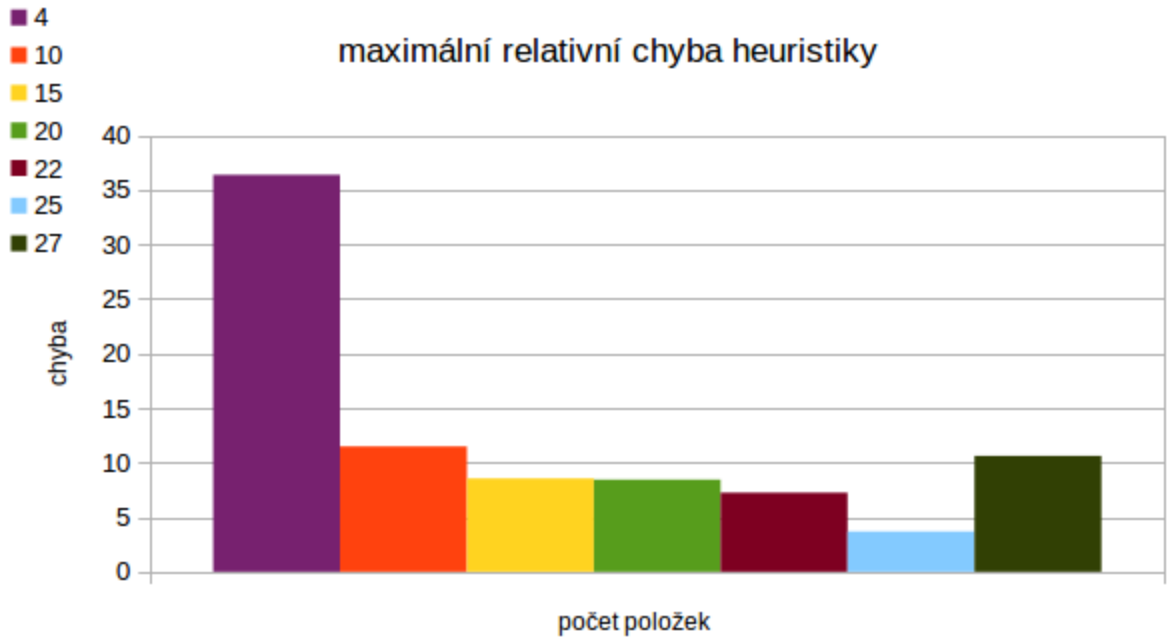






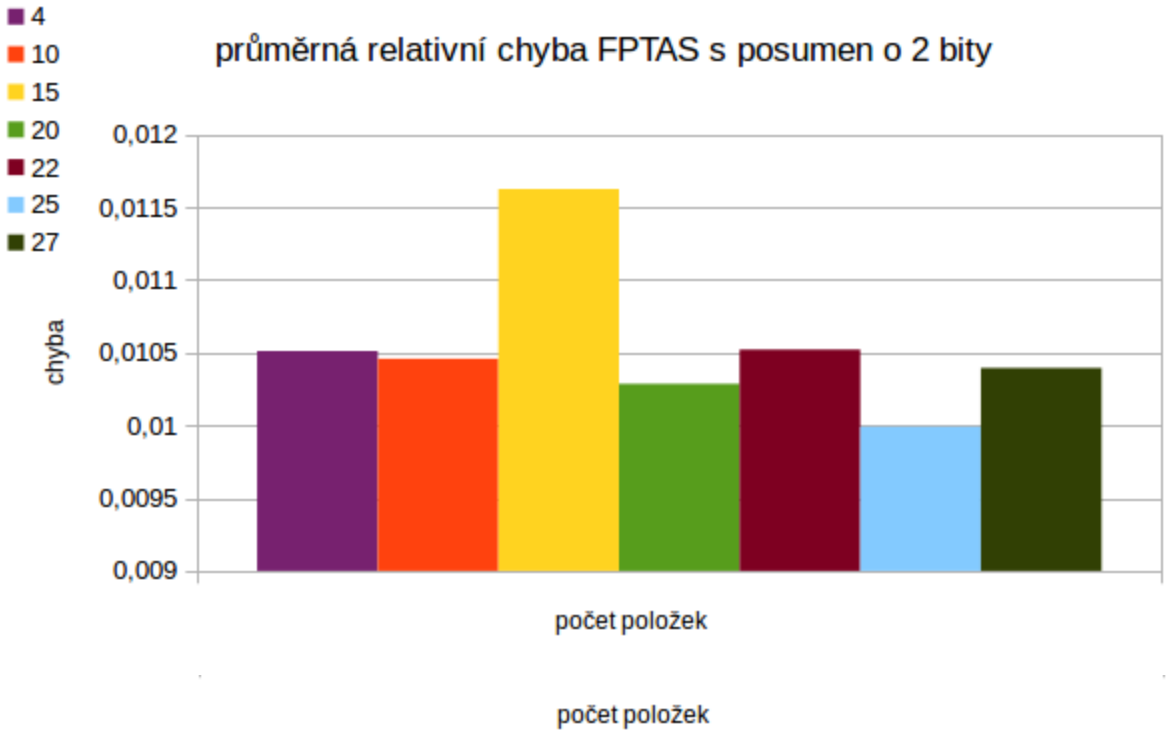
U metod které nejsou přesné (heuristika a FPTAS) uvádím grafy relativních chyb.

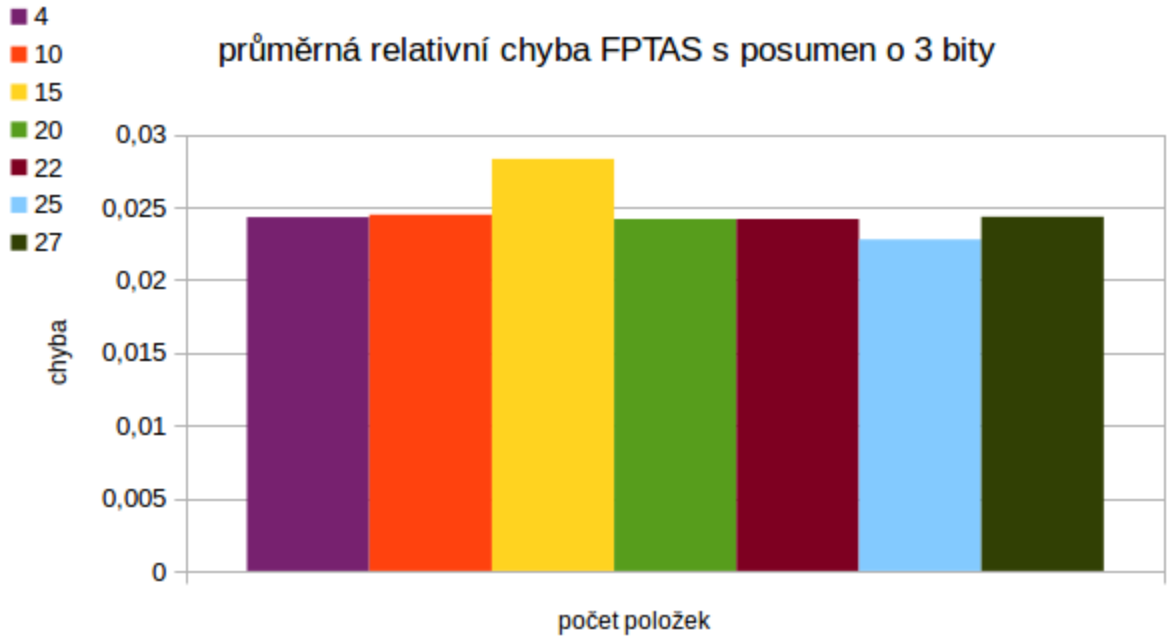
Na prvním grafu je vidět velikost průměrné relativní chyby heuristické metody oproti řešení hrubou silou (které bylo také kontrolováno vůči referenčnímu řešení). Graf přehledně ukazuje, že chyba se se zvyšujícím počtem položek zmenšuje.

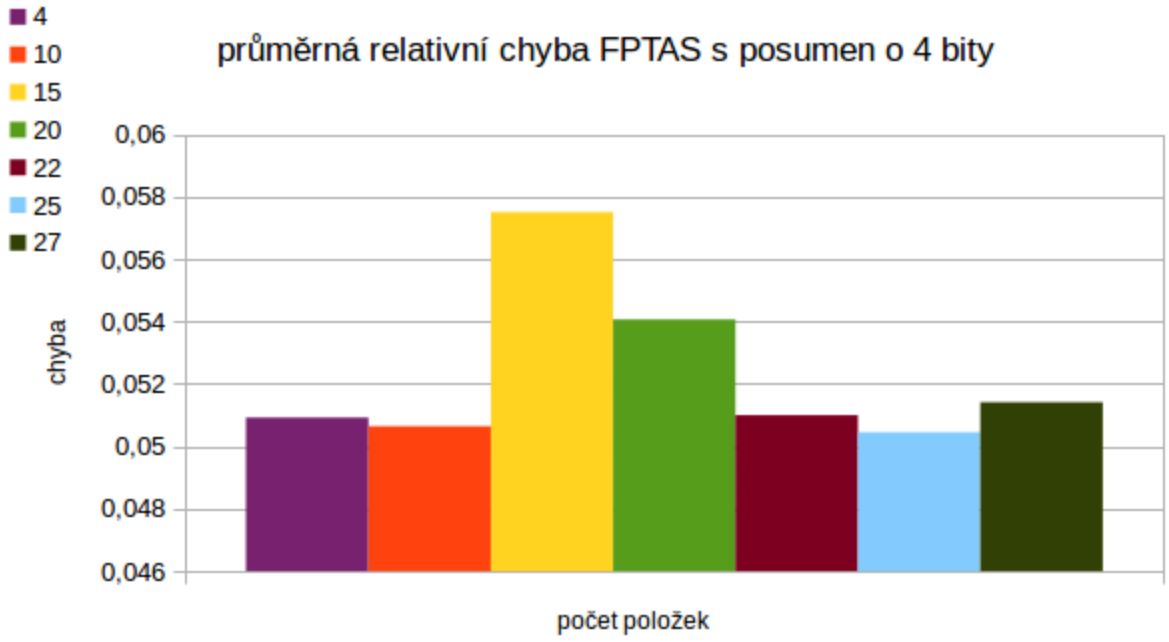


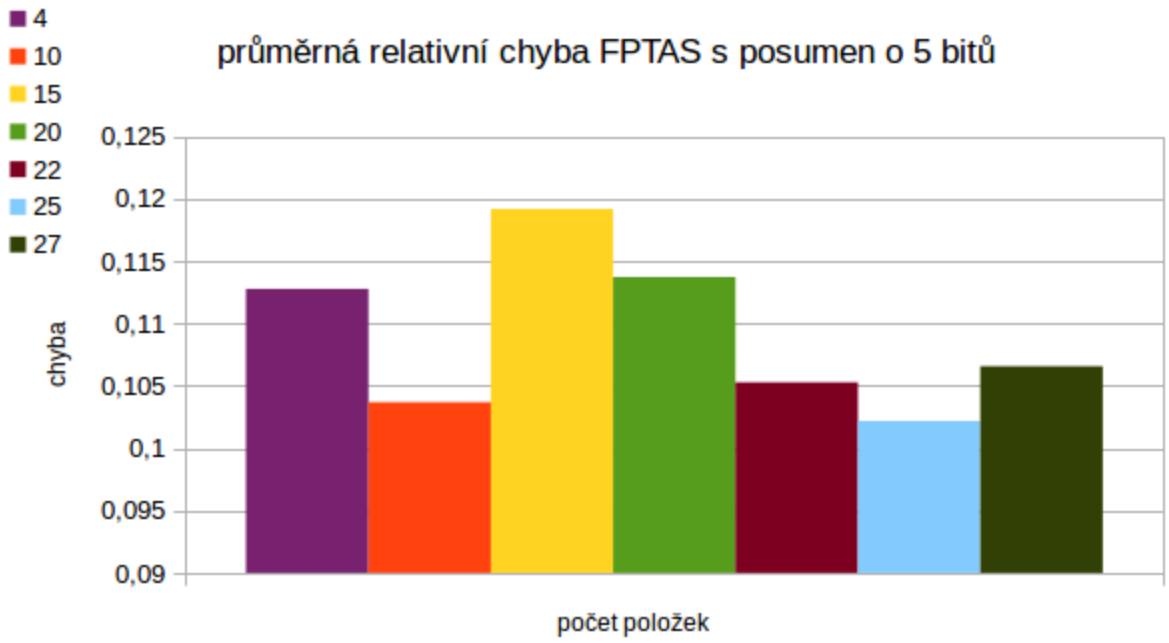
Na druhém grafu je vidět velikost maximální chyby pro jednotlivé sady instancí. Přestože chyba pořád klesá, poslední instance má chybu opět větší. Jedná se však pouze o jedno měření a průměrná chyba je dle očekávání (viz předchozí graf).

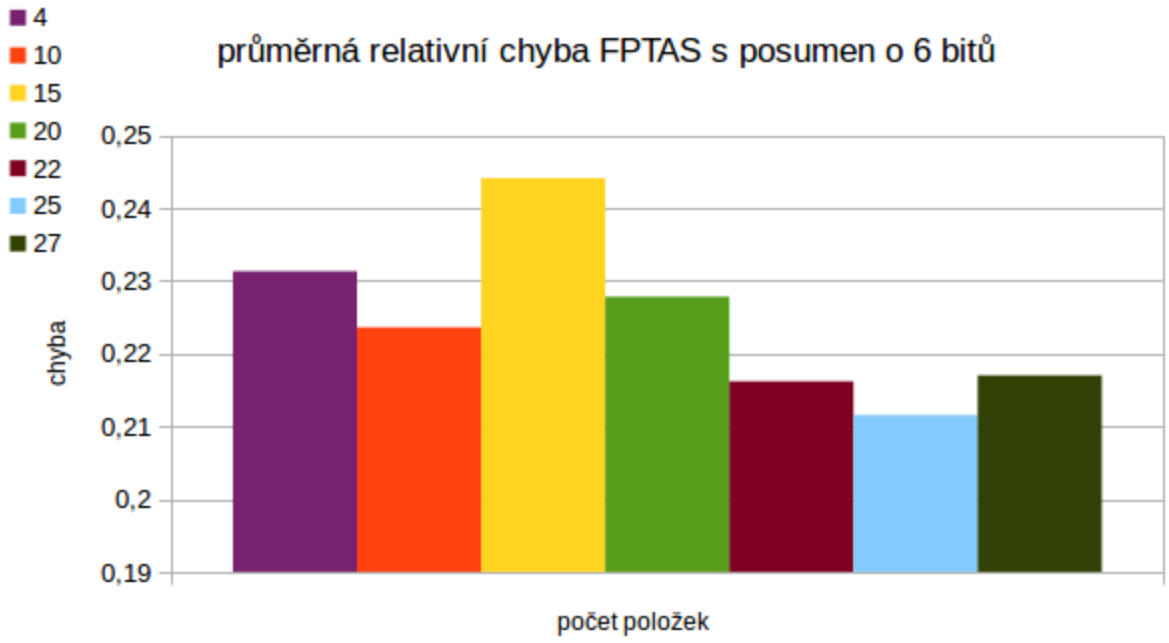
Následující graf zobrazuje velikost chyby pro metody FPTAS s posunem o 1 až 6 bitů.




průměrná relativní chyba FPTAS s posumen o 3 bity
4
10
15
20
22
25
27
chyba
0,03
0,025
0,02
0,015
0,01
0,005
0
počet položek

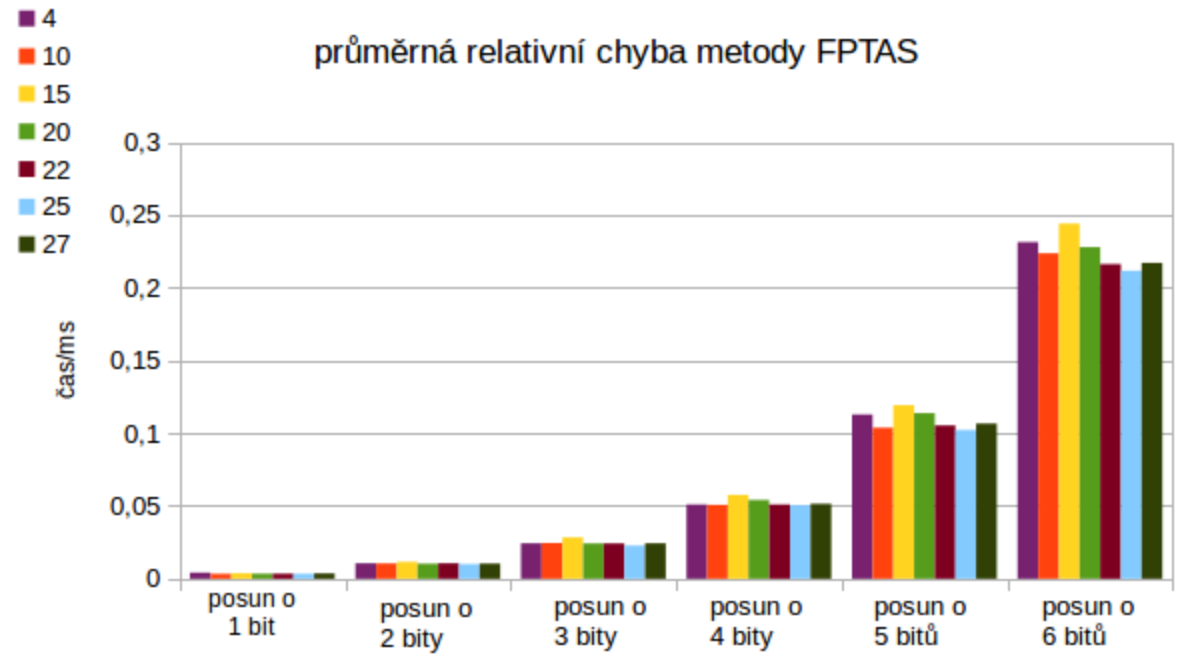

průměrná relativní chyba FPTAS s posumen o 4 bity
4
10
15
20
22
25
27
chyba
0,06
0,058
0,056
0,054
0,052
0,05
0,048
0,046
počet položek



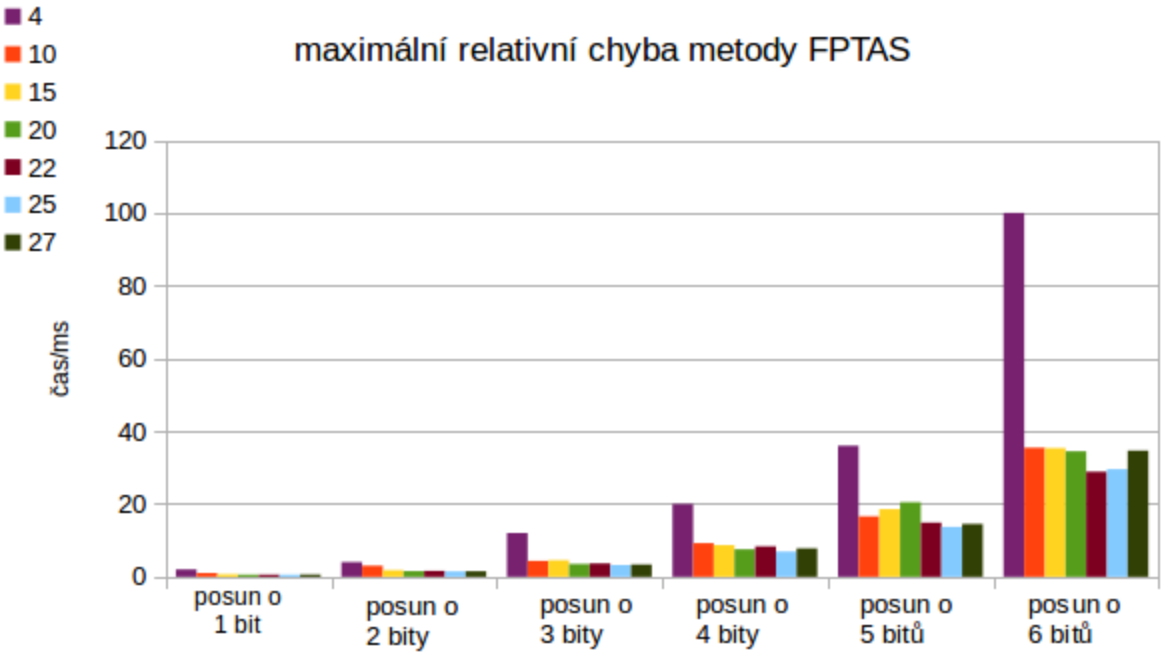


Jak se dalo očekávat, s rostoucím počtem bitů, o které se jednotlivé ceny posunou, se zvětšuje i průměrná chyba. Ta roste od nějakých 0,35% (při posunu o 1 bit) až po cca 23% (při posunu o 6 bitů). Velikost instancí nemá příliš velký vliv na průměrnou chybu.

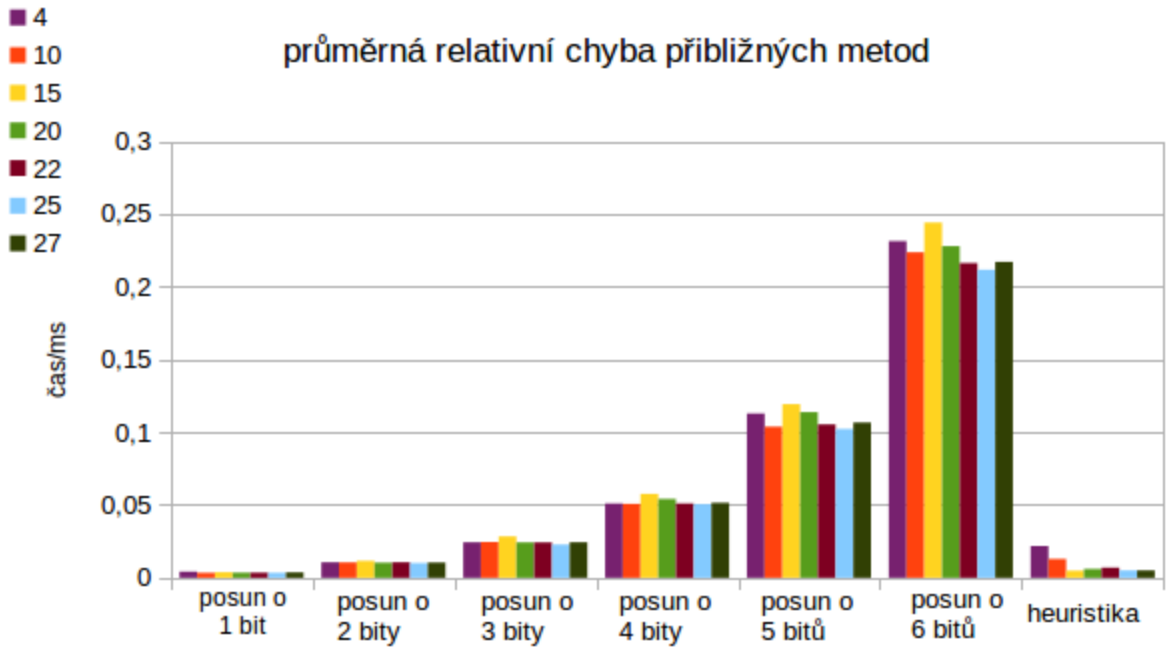
Dále uvádím graf souhrnný graf průměrných chyb. Zde je vidět rostoucí tendence chyb (přibližně exponenciální).



Další graf zobrazuje velikost maximální relativní chyby pro jednotlivé instance. Jak je vidět, posun čísel nejvíce ovlivní malé instance, kde se chyba hůře ztratí.



Na úplný konec ještě uvádím graf, který srovnává chybu heuristiky s chybami metod FPTAS.



Je vidět, že chyba heuristiky je přibližně stejná jako chyba FPTAS při posunu o 2 bity. Heuristika je ale v tomto případě mnohem rychlejší a proto je na tom z mého pohledu lépe.

Průměrná relativní chyba v %:

instance	FPTAS 1	FPTAS 2	FPTAS 3	FPTAS 4	FPTAS 5	FPTAS 6	heuristika
4	0,41	1,05	2,43	5,09	11,28	23,13	2,17
10	0,33	1,05	2,45	5,06	10,37	22,36	1,28
15	0,37	1,16	2,83	5,75	11,92	24,41	0,47
20	0,34	1,03	2,42	5,41	11,37	22,78	0.60
22	0,35	1,05	2,42	5,10	10,53	21,62	0,69
25	0,32	1,00	2,28	5,05	10,22	21,16	0,50
27	0,36	1,04	2,43	5,14	10,66	21,70	0,50

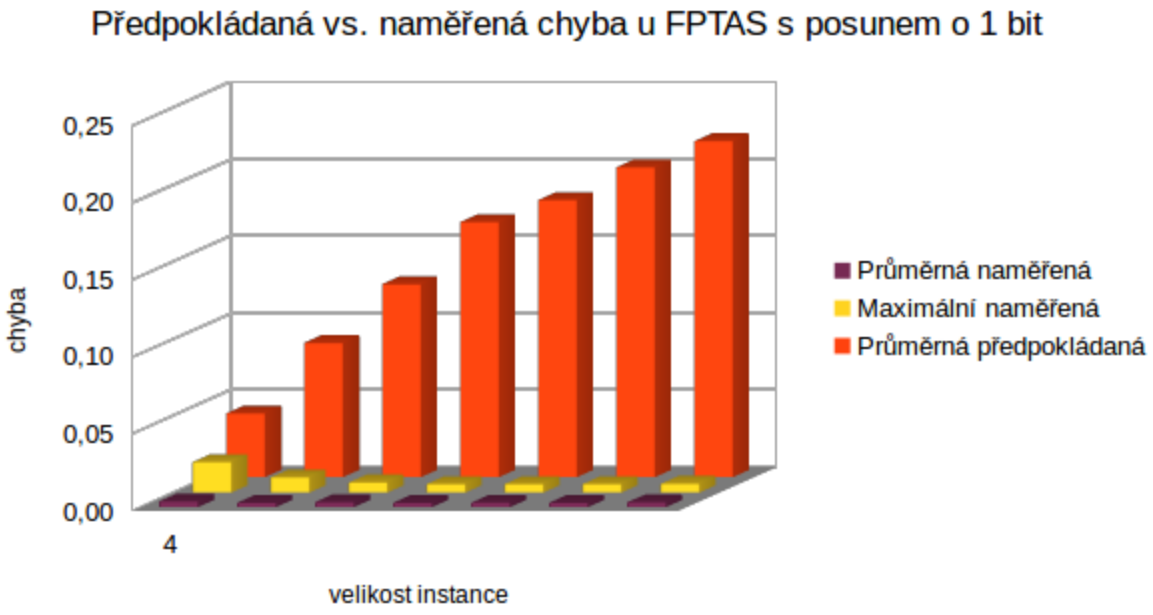
Předpokládaná průměrná chyba v %:

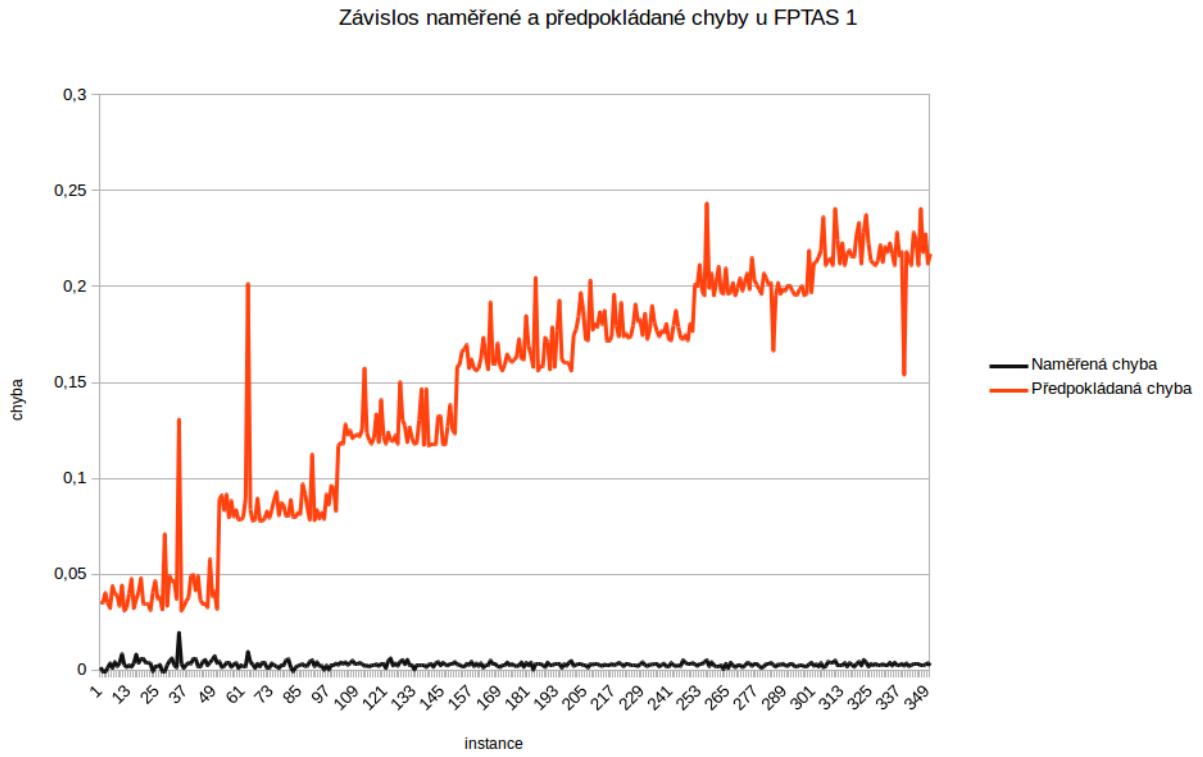
instance	FPTAS 1	FPTAS 2	FPTAS 3	FPTAS 4	FPTAS 5	FPTAS 6	heuristika
4	4,19	8,38	16,76	33,52	67,04	134,08	-
10	8,76	17,52	35,05	70,10	140,19	280,38	-

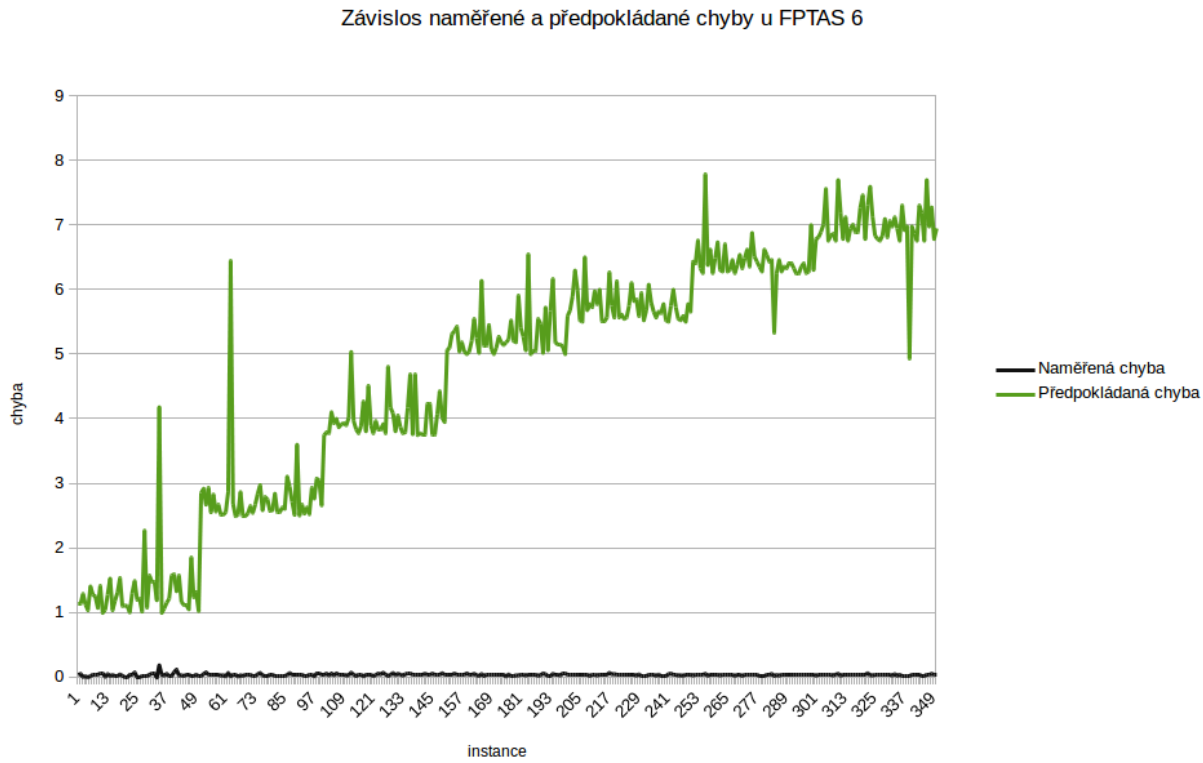
15	12,57	25,14	50,28	100,56	201,12	402,23	-
20	16,60	33,20	66,41	132,81	265,62	531,24	-
22	18,03	36,05	72,10	144,20	288,41	576,81	-
25	20,15	40,31	80,62	161,23	322,47	644,94	-
27	21,85	43,71	87,42	174,84	349,68	699,36	-

Maximální relativní chyba v %:

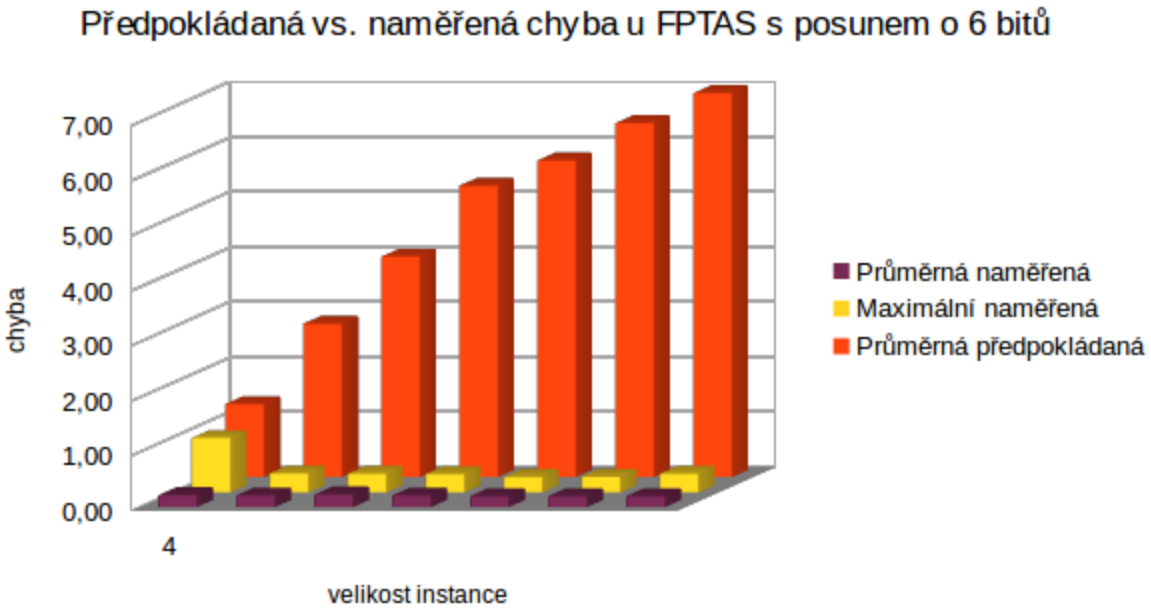
instance	FPTAS 1	FPTAS 2	FPTAS 3	FPTAS 4	FPTAS 5	FPTAS 6	heuristika
4	2,00	4,00	12,00	20,00	36,00	100,00	36,36
10	1,01	3,03	4,32	9,19	16,58	35,45	11,48
15	0,68	1,76	4,51	8,67	18,57	35,35	8,54
20	0,56	1,50	3,51	7,51	20,45	34,46	8,43
22	0,58	1,58	3,64	8,35	14,83	28,86	7,23
25	0,58	1,43	3,20	6,90	13,68	29,53	3,68
27	0,59	1,48	3,36	7,78	14,50	34,62	10,60







Na těchto dvou grafech je vidět závislost předpokládané chyby na naměřené pro všechny instance. Je vidět, že předpokládaná chyba je vždy větší než naměřená.



Závěr

Vypočítat řešení hrubou silo se mi podařilo pouze pro úlohy do 27 položek v batohu. Bylo by možné vypočítat i více, ale čas by byl již v řádu hodin a pro celou sadu 50 úloh by to bylo velice zdlouhavé. Naopak výpočet pomocí heuristiky téměř nic nestojí a jeho výsledek je okamžitě. Jeho relativní chyba je také velice malá a klesá od přibližně 2% u úloh se 4mi prvky až na pouhé 0,5% u úloh s více než 20ti prvky. Tato chyba je podle mého názoru ve většině případů akceptovatelná. Maximální chyba však činila cca 36%, což už je hodě, medián je ale 0. To znamená, že více než polovina případů byla vypočítána zcela přesně. Průměrná chyba algoritmu FPTAS a posunem o 1 bit je sice menší než heuristika (pouze cca 0,3%), medián je ale 0,34. To znamená, že více než polovina případů byla vypočítána s alespoň nějakou chybou. Při posunu o 5 bitů je maximální chyba 36% (stejně jako u heuristiky) a medián je 11%. Podle mě se nejvíce vyplatí metoda dynamického programování, jejíž rychlost je pro rozumné instance přijatelná a výsledek přesný. Urychlení v podobě FPTAS sice přináší časovou úsporu, ale dle mého názoru tato úspora nestojí za vzniklou chybu.

Konfigurace počítače

Procesor Intel CORE i7, frekvence 1,8 GHz, Operační paměť 8Gb. Linux Ubuntu 64 bitů. Naprogramováno v jazyce Java.