jméno: Mykyta Boiko
login: boikomyk

Domácí úkol č.1

Problém batohu (Knapsack Problem)

---

1. Specifikace problému:

Vyřešit *rozhodovací* verzi problému knapsack 0/1.

Krátký popis problému batohu:
Cílem je umístění podmnožiny předmětů (každý předmět má svoji cenu a hmotnost) do batohu, který je omezené nosnosti tak, aby cena nákladu byla maximální.

Rozhodovací problém je otázka typu buď ANO nebo NE na nějaké množině vstupů.

V kontextu dané úlohy problém batohu je reprezentován:
- batohem omezené nosnosti (**M** - nosnost)
- sadou předmětů, popsaných dvěma parametry: cena a hmotnost (dvojice - {$\boldsymbol{c}_1$, $\mathbf{v}_1$})
- minimální vyhovující cena řešení (**C** - minimální cena)

Musíme odpovědět na otázku, zdali existuje alespoň jedna kombinace předmětů umístěna do batohu, aby nebyl překocen limit nosnosti a zároveň byla splněna podmínka na minimální cenu (součet cen těchto předmětů alespoň ***C***).

Technické aspekty:
Vstupní data jsou reprezentovány dvěma sadami (NR a ZR) instancí rozhodovací verze optimalizačního 0/1 batohů pro n=4 10 15 20 22 25 27 30 32 35 37 40, kde n je počet předmětů.

Souhrn a doplňující informace:
Za úkol jsem měl naprogramovat program, který umí řešit rozhodovací verzi problému batohu 0/1:
- hrubou silou (*angl*. brute force)
- metodou větví a hranic (*angl*. branch and bounds)

(vice *1. Exaktní metody řešení rozhodovacího problému .. [link]*)

2. Rozbor možných variant řešení a jejích postupů:

K dispozici existuje několik různých algoritmu/metod řešících daný problém, založených na různých programovacích přístupech (na přístupů dynamického programování, přístupů větvení a hranic atd.)

Před tím jak se pustíme popisovat jednotlivé algoritmy je nutné říct, že lepší kombinace předmětů je ta, která (předpokládáme, že podmínka na nosnost batohu je splněna):

- buď má větší celkovou cenu
- nebo cena je stejná jako v předchozím nalezeném nejlepším řešení, ale celková hmotnost věci je menší

To jsou vhodné podmínky, třeba pokud máme za úkol najít konstruktivní verzi problému, avšak pro řešení rozhodovacího problému stačí najít první vyhovující kombinaci věci (jde vlastně o odpověď jestli existuje alespoň jedna vhodná kombinace).

První domácí úkol vyžadoval implementace dvou algoritmů:

BRUTE FORCE (ŘEŠENÍ HRUBOU SILOU)

Jde vlastně o to, že procházíme/iterujeme přes všechny kombinace předmětů v batohu. V každém průchodu se vypočítá celková hmotnost a cena přidaných předmětů. Jakmile narážíme na vyhovující kombinaci, totiž celková hmotnost věci v batohu nebude přesahovat nosnost batohu a zároveň bude splněna podmínka na minimální cenu řešení, tím zastavujeme algoritmus. Dostáváme odpověď *ANO* na rozhodovací verzi problému.

V nejhorším případě máme projít přes všechny kombinace a na konce zjistit, že kombinace věcí vyhovující podmínkám neexistuje. Tím dostáváme odpověď *NE*.

BRANCH AND BOUNDS (ŘEŠENÍ METODOU VĚTVÍ A HRANIC)

Toto řešení vychází z původního Brute Force řešení. Tentokrát stavíme strom všech možných kombinací řešení a procházíme jej do hloubky s tím, že ořezáváme podstromy, které nevedou ke správnému řešení. Na to dáváme podmínku, která nám v každém kroce říká, zda maximální teoretická cena batohu, kterou při daně konfiguraci můžeme dosáhnout, je lepší, než cena již nalezeného lepšího řešení.

Pro každou z podtříd řešení se vypočítá horní mez(*angl*. upper bound) maximální hodnoty cenové funkce dosažené řešeními patřícími do podtřídy. Na základě těchto horních mezí se provádí další fáze větvení, vypočítávají se nové horní meze atd. Dokud se nakonec nedosáhne řešení, které má hodnotu cenové funkce větší než horní hranice všech podtříd a také vyšší než hodnoty cenové funkce pro všechna dříve získaná řešení.

Maximální teoretickou cenu dostáváme tak, že naplňujeme batoh na maximum s tím, že postupně bereme předměty seřazené podle poměru cena/váha. V případě, pokud nějaký předmět do batohu nepřidáváme, tak připočteme jeho poměrnou část ceny.

Každý uzel stromu má dva syny a zpravidla levý syn je definován operaci přidávání dalšího předmětů ze seřazené posloupnosti. Pravý syn je definován kombinaci, ve které další předmět ze seřazené posloupnosti nepřidáváme.
Taky neprocházíme větve stromů, ve kterých překročíme nosnost batohu.
Tim se trochu zlepšuje exponenciální složitost algoritmu.

3. Popis kostry algoritmu:

V přiloženém souboru/archívu můžete najít zdrojový kód, který je dobře okomentovaný. Tady bude uvedeny jen krátký a stručný popis kostr každého z algoritmů.

BRUTE FORCE:

Na začátku vypočítáme počet všech možných kombinací. Jelikož máme 0/1 problém batohu, jinak řečeno pokud bychom řešili konstruktivní problém, tak odpověď by měla vypadat ve tvaru posloupnosti čísel 0/1 délky n. (např. pro n=8 : 01010011). Čteme to tak, že do batohu přidáváme předměty 2,4,7 a 8. Pokud v binárním zápisu na nějaké pozice je 1 => přidáváme předmět do batohu, jinak nepřidáváme.
Tím můžeme jednoduše spočítat celkový počet všech kombinaci:

Pro každý předmět: **1** - přidat, **0** - nepřidávat. Celkem máme 2 varianty. Pro n předmětů celkem máme 2^n.
Iterujeme od COMBINATIONS:=0...2^N. V každé iteraci podle pozice všech 1 v binární reprezentaci COMBINATIONS přidáváme předměty do batohu. Pokud je kombinace validní, t.j. nosnost batohu není překročena, tak jdeme dál a kontrolujeme podmínku na minimální cenu. V případě, že kombince taký bude vyhovovat podmínce na cenu, tak tím algoritmus zastavujeme, jelikož jsme dostali řešení ANO rozhodovacího problému.

Jak již bylo uvedeno výše, v nejhorším případě máme projít přes všechny kombinace předmětů v batohu a to pokud jenom poslední kombinace je validní nebo hledaná kombinace neexistuje a řešením rozhodovacího problémů bude NE.

BRANCH AND BOUNDS:

Na začátku seřadíme všechny předměty dle takového kritéria jak poměr *cena/hmotnost*. Vkládat předměty do batohu budeme začínat z předmětu, který má toto kritérium nejlepší a v každém dalším kroku buď se přidává nebo se nepřidává předmět s další nejlepší hodnotou tohoto kritéria.

Vytváříme prázdnou frontu (*FIFO - first in, first out*). Vytvoříme *fake* uzel rozhodovacího stromu a zařadíme ho do fronty (cena a hmotnost uzlu je 0). Zatímco fronta není prázdná budeme procházet stavový prostor. Pro pokračování v prozkoumání větve je nutné aby její horní mez (upper bound) byla větší než dosud dosažené maximum.

K odhadu horní meze se využívá jenom součet cen předmětů, které se do batohu skutečně vejdou. Co se týče třeba předmětů, které se do batohu nevejdou, tak připočítáme jejich poměrnou část ceny. Je třeba poznamenat, že odhad ceny větví je nutné přepočítat jen pro větví, kde předmět se nepřidává. Odhad ceny větví se vypočítá v lineárním čase. Každopádně, tím se celý strom řešení dobře prořeže.

Stejně jako u Brute Force algoritmu, jelikož řešíme rozhodovací verze problémů, tak nepotřebujeme hledat nejlepší možné řešení, stačí jen první vyhovující stanoveným podmínkám. To znamená, že v každém kroku (před tím, jak začít zpracovávat další uzel ze fronty) kontrolujeme jestli je validní, už bylo nalezeno lepší řešení a zároveň podmínka na minimální cenu je splněna. V kladném případě, zastavujeme celý algoritmus a vracíme odpověď ANO.

V nejhorším případě zase budeme muset projít celý strom. Proto složitost algoritmu je exponenciální. Doba běhu je velmi závislá na tom jak dobře se povede stavový prostor prořezat.

4. Konfigurace počítače:

MacBook Pro mid 2014

Procesor: 2,5GHz čtyřjádrový Intel Core i7
Paměť: 16 GB 1600MHz paměti DDR3L na desce
Grafická karta: Intel Iris Pro 1536 MB

5. Naměřené výsledky:

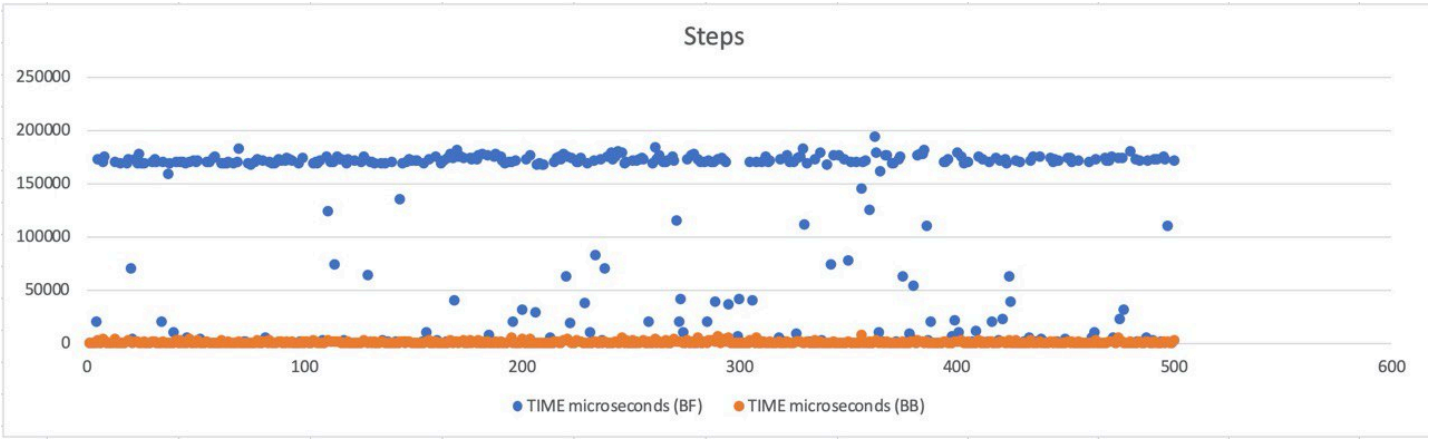
Měření výsledků se provádělo na sadě dát NR15.

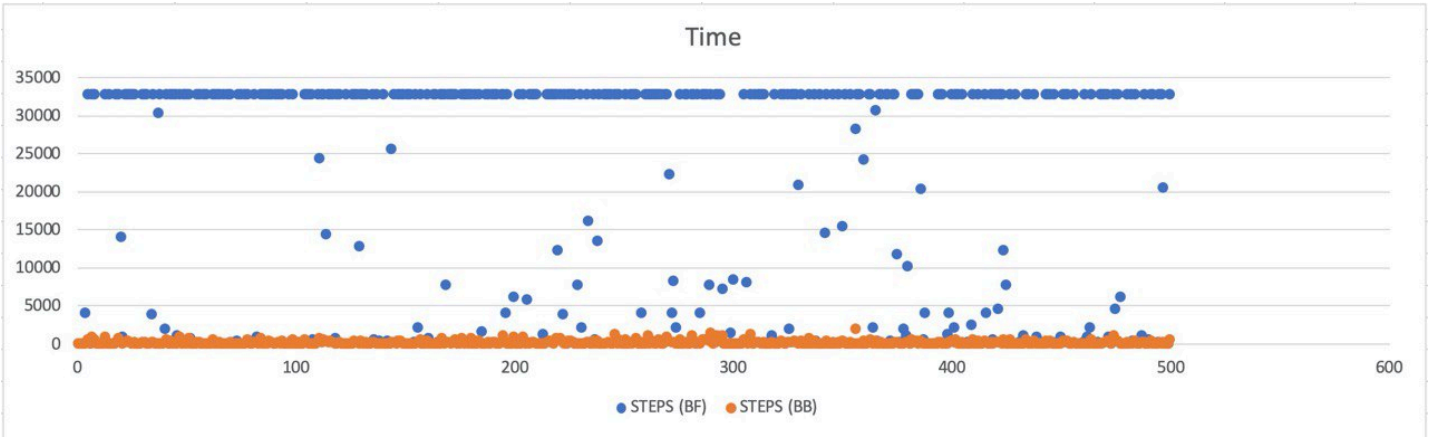
Brute Force					
Time			Steps		
Max	Min	Average	Max	Min	Average
193907,39	5,925	89648,801	32768	1	17083,69

Branch and Bounds					
Time			Steps		
Max	Min	Average	Max	Min	Average
7586,045	12,838	720,59003	1909	1	167,00601

Z dát uvedených v tabulce je vidět, že **Branch and Bounds** je rychlejší a za menší počet kroku dokáže najít optimální řešení. A je to díky dobrému ořezáváni stavového prostoru.
Jediná zajímavost v této tabulce je, že minimální čas běhu algoritmu **Brute Force** je menší, než u **Branch and Bounds**. Zřejmě to platí pro případ, kde počet kroku se rovná 1, t.j. první nalezena kombinace už vyhovovala stanoveným podmínkám.

V podstatě je to způsobeno tím, že algoritmus **Branch and Bounds** potřebuje na začátku seřadit předměty podle poměru cena/váha a pak se ještě vypočítají odhady cen větví za lineární čas.
Algoritmus **Branch and Bounds** v každém uzlu tráví více času, než **Brute Force** s kombinaci, ale stále má lepší výsledky.

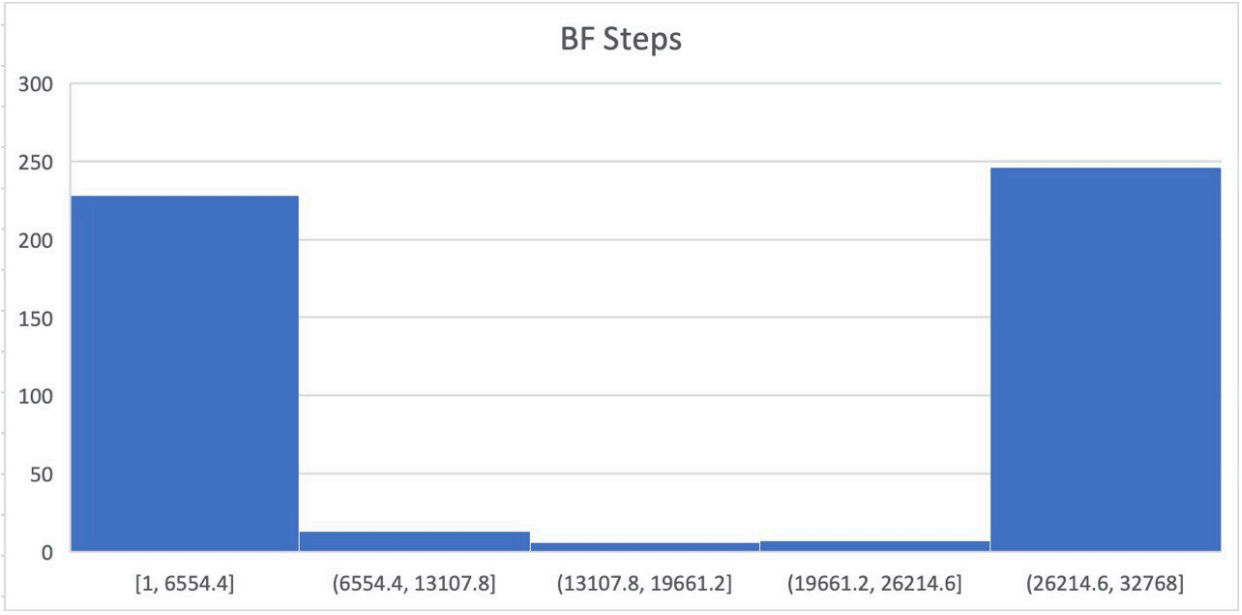




Z dalších grafů pozorujeme porovnání počtu probraných konfigurace a času, za který konfigurace vyhovující podmínkám(řešící problem) byla nalezena.

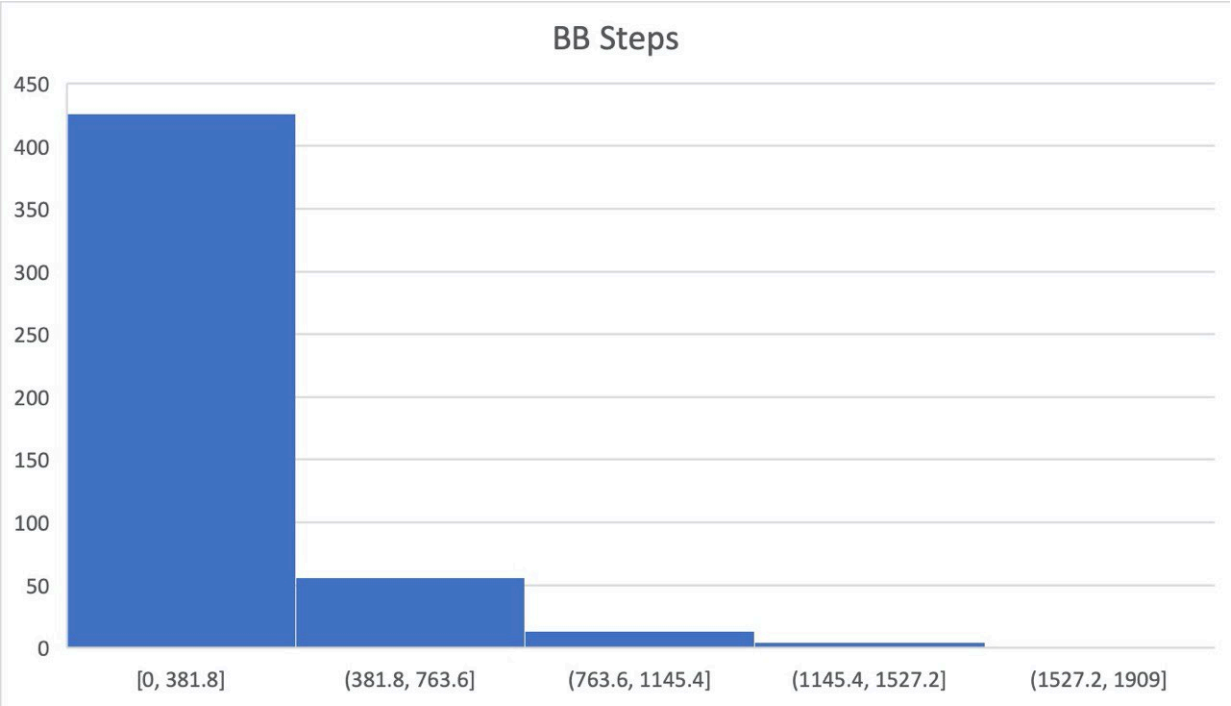
Stejně je vidět, že **Branch and Bounds** za menší počet probraných konfigurací(stejně tak i za menší čas) může najít řešení. Bohužel, na sadě dát NR15 není jasně patrna tendence **Branch and Bounds,** v nejhorším případě, degenerovat až na **Brute Force**. Pravděpodobně byla zvolena vhodná heuristika, která zaručuje expandování větve nejpravděpodobněji vedoucí k optimálnímu řešení. Na větších sadech dát, třeba NR40 už je vidět tendenci **Branch and Bounds** k degenerování k **Brute Force** (exponenciální složitost).

Shrneme, že **Branch and Bounds** je velmi závislá na vybrané heuristice a na konkrétních instancích problému.

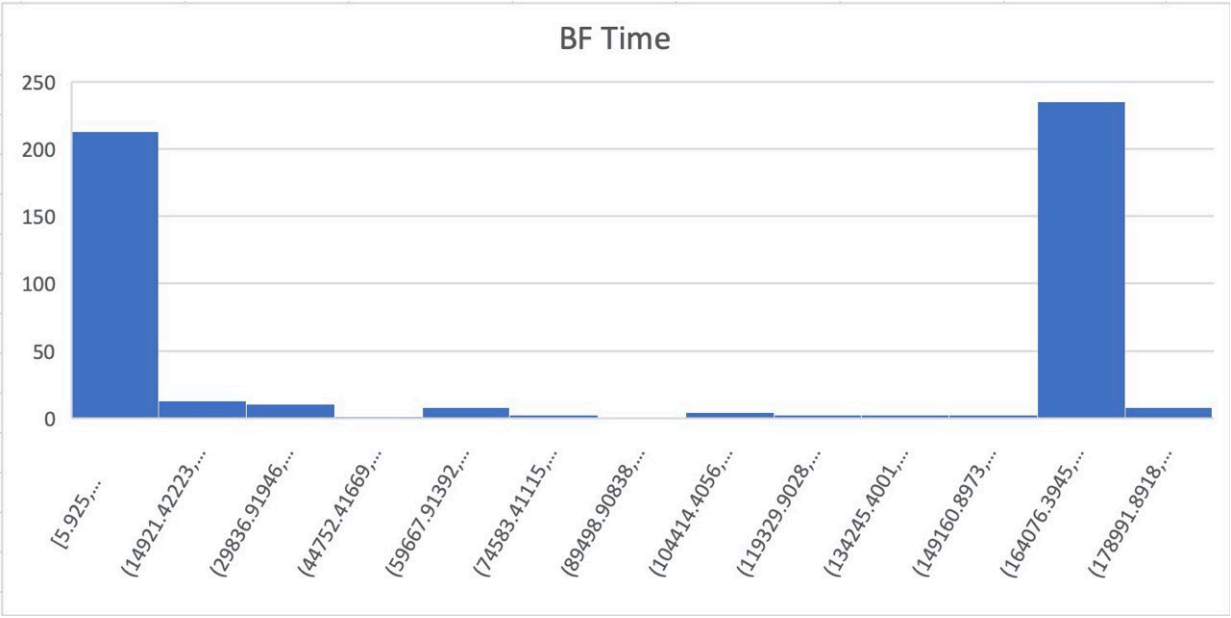


pro sadu NR15 je n=15, z toho plyne 2^15=32768 max počet kroků/všech kombinací

Na histograme pozorujeme četnost počtu probraných konfigurace/kombinace. Nás by mělo zajímat dva největších sloupců. Vidíme, že nejčastěji **Brute Force** dokáže vyřešit problém za buď skoro maximální počet kroku 26214-32768 nebo za 1/5 od maximálního počtu kroků 1-6554 kroků.

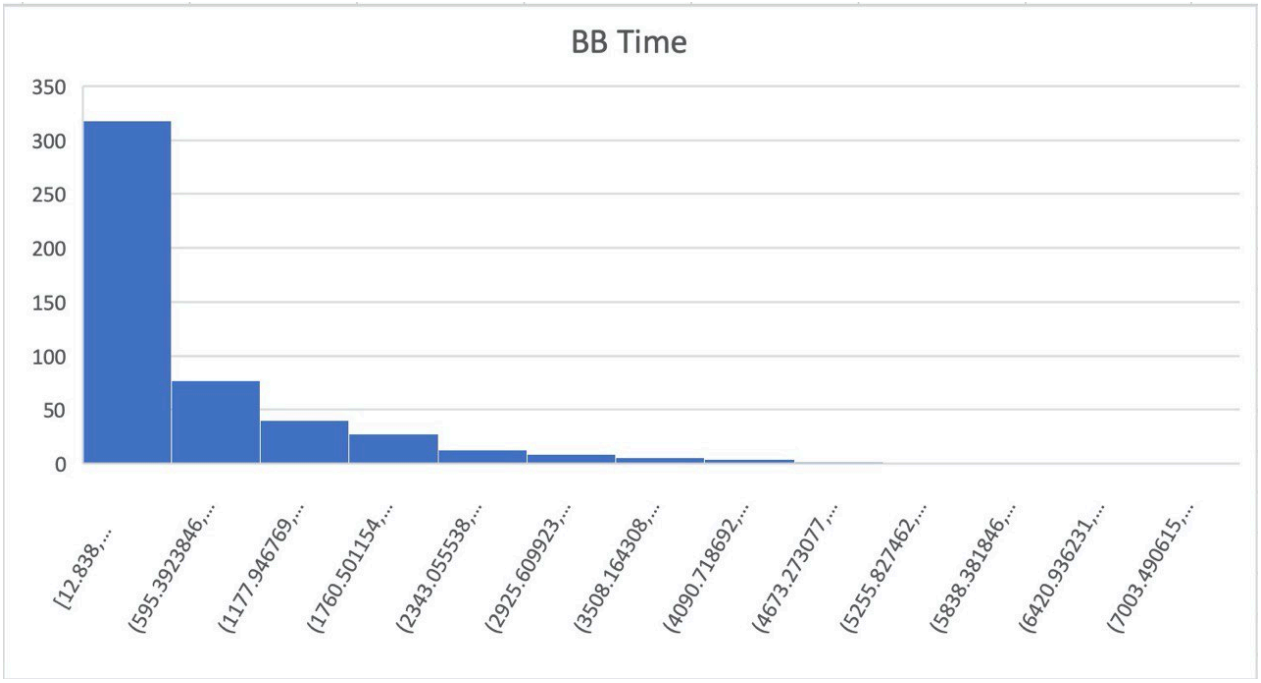


Na rozdíl od **Brute Force**, algoritmus **Branch and Bounds** nám ukazuje mnohem lepší výsledky. Nejčastěji **Branch and Bounds** daří najít správnou konfiguraci za 0-381 kroků.
Dá se říct, že **Branch and Bounds** docela dobře řeší sadu NR15 a nikdy počet probraných kombinací nespadá do max hodnoty v 32768. Stejně pokud bychom řešili konstruktivní problém, histogram četnosti kroků se by moc lišil.



Docela logicky je to, že histogram **Brute Force** pro čas je skoro stejný jako počet probraných kroku. Stejně pozorujeme dva největší sloupce. Buď **Brute Force** dokáže vyřešit

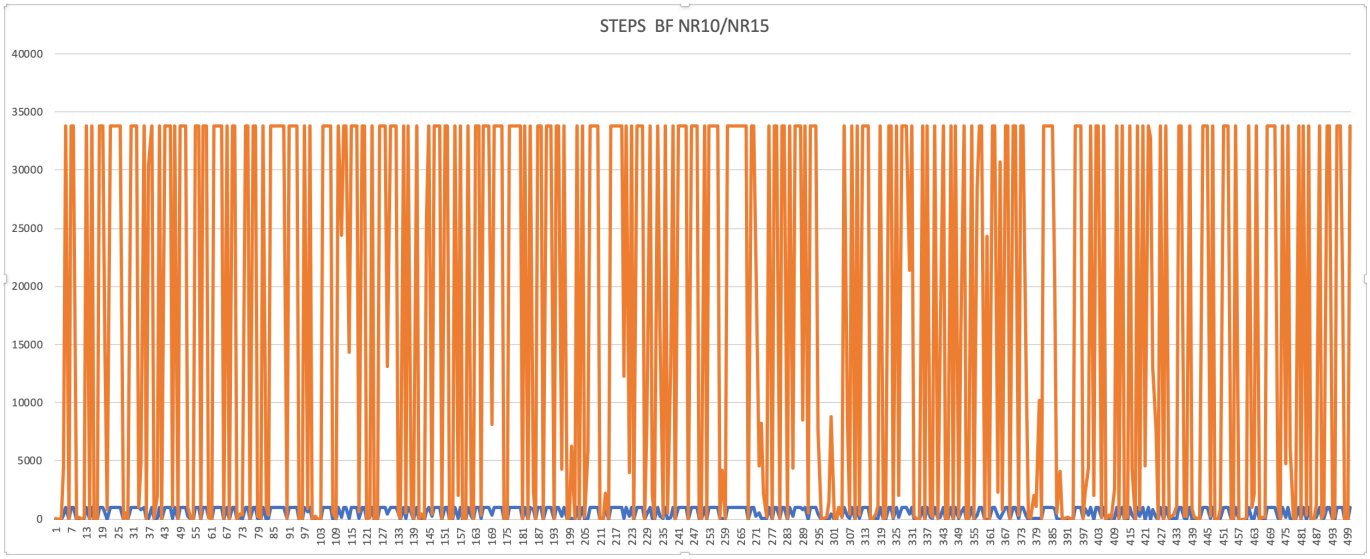
problém už za *k* prvních probraných konfiguraci s časem 5.925-14920 milisekund, nebo probere skoro všechny kombinace a to za čas 164076-178990 milisekund.

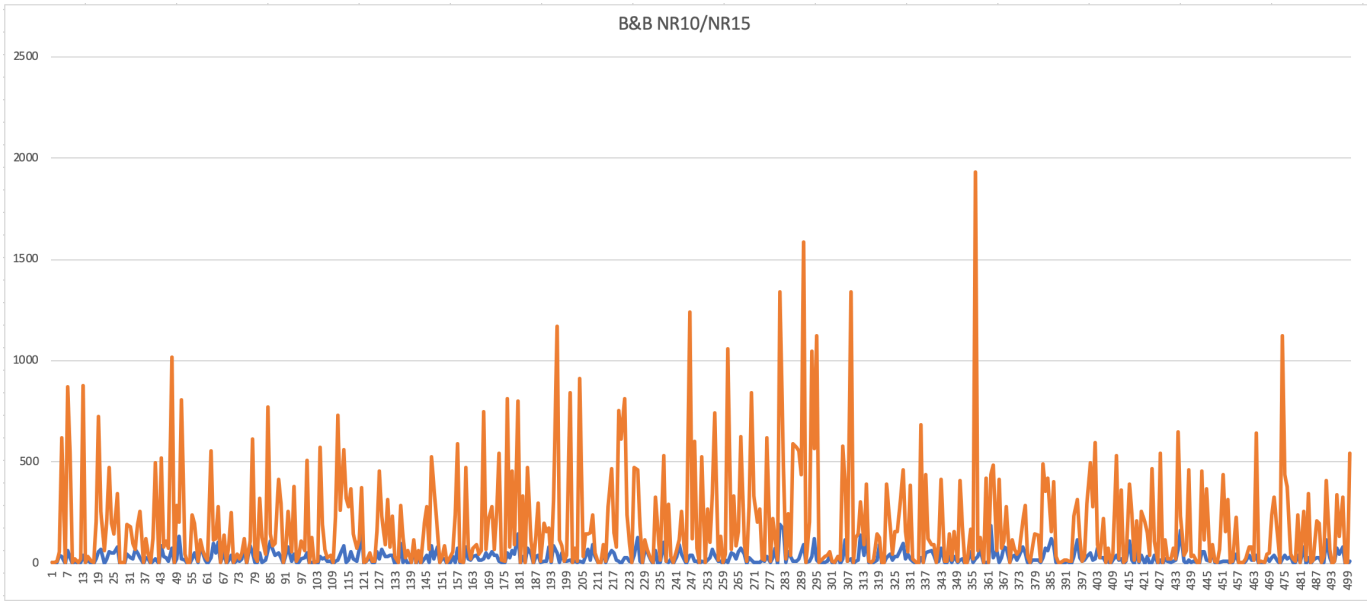


To samé můžeme říct i o **Branch and Bounds** čase. Na sadě NR15 **Branch and Bounds** většinou daří najít řešení za minimální čas a t.j. 12-595 milisekund a 595-1177 milisekund. Což je mnohem rychlejší než u **Brute Force** v nejhorších případech. Třeba přibližně o 300-krať rychleji.

6. Závislost výpočetní složitosti na velikosti instanci:

Pro experimentální vyhodnocení závislosti výpočetní složitosti na velikosti instanci budeme muset provést ještě navíc měření na jiné sadě dát. Vezmeme si jako druhou sadu NR10.

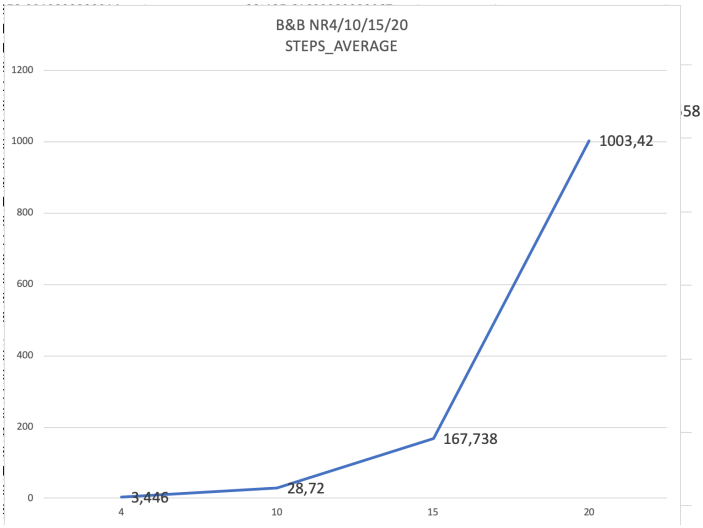


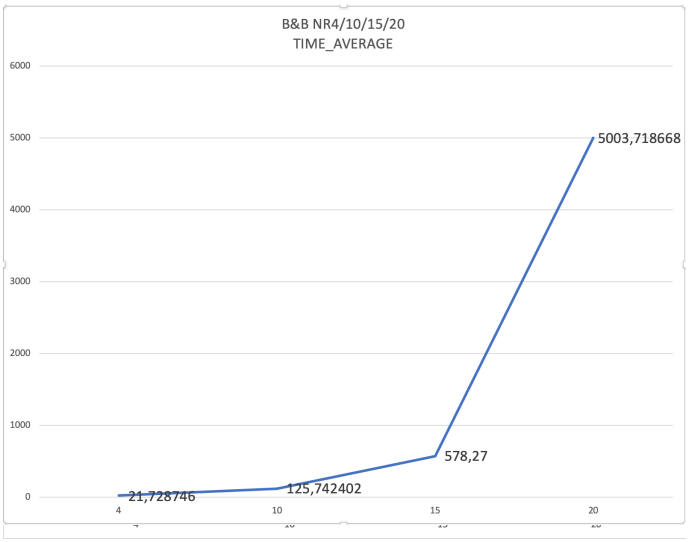


Oranžová čára označuje počet probraných konfigurace pro každou instanci sady NR15, kde maximálně počet kombinace je 2^15 = 32 768. Modrá čára označuje počet probraných konfigurace pro každou instanci sady NR10, kde maximální počet kombinace je 2^10 = 1024.

Dále zkusíme porovnat průměr počtu probraných konfiguraci/průměr času pro stejný algoritmus, ale na různých sadech dát (obsahující instancí různých délek: 4,10,15,20).

NR4/NR10/NR15/NR20				
	Brute Force		Branch and Bounds	
INSTANCE_LENGTH	STEPS_AVERAGE	TIME_AVERAGE	STEPS_AVERAGE	TIME_AVERAGE
4	9,71	22.41447	3,446	21.728746
10	550,086	2117.61843	28,72	125.742402
15	17083,69	412448.301	167,738	578.27
20	537539,558	3813375.696066	1003,42	5003.718668





Z uvedených grafů a pozorování, provedených v předchozích odstavcích, dá se říct:

- **Brute Force** algoritmus každopádně prohledává celý prostor, což je 2^n. Navíc v každém kroku (jednotlivé konfigurace) se navíc vypočítá suma vah a cen všech předmětů, což je **n**. Ve výsledků máme časovou složitost $O(n*2^n)$. To platí pro konstruktivní problém. Co se týče rozhodovacího problému, tak počet probraných konfiguraci je spíš závislý na distribuci předmětů stanovené instanci problému. Avšak, asymptotická složitost zůstává stejna. Což vede nás na exponenciální složitost.
- **Branch and Bounds** je skoro vždy rychlejší (díky ořezávání prostoru), ale stejně v nejhorším případě prohledává celý prostor, resp. celý rozhodovací strom. Neboť rozhodovací strom je binární (každý uzel má 2 potomky), což vede nás na $2^{n-1} - 1$ celkový počet uzlu. To je stejně exponenciální složitost.

Docela pěkně exponenciální závislost je viditelná v grafech průměru počtu probraných konfiguraci pro stejný algorithm na různých sadech. Jak u **Brute Force** tak i **Branch and Bounds** čára je ostře rostoucí.

7. Programovací detaily:

Při programování úlohy byl použít *Python3.9* programovací jazyk.
Všechny implementované metody jsou funkční a připravené pro testování.

SPOUŠTĚNÍ PROGRAMU:

usage: main.py [-h] [-cnt COUNT] [-in INPUT] [-ref REFERENCE] [-b] {Brute Force,Branch & Bounds}

Process input problem instances.

positional arguments:
{Brute Force,Branch & Bounds}

optional arguments:
-h, *--help* show this help message and exit
-cnt COUNT, *--count* COUNT
-in INPUT, *--input* INPUT
paste path to input files
-ref REFERENCE, *--reference* REFERENCE
paste path with references
-b, *--benchmark* turn on/off benchmark

PŘÍKLAD SPOUŠTĚNÍ:

python3.9 main.py bb -cnt=1 -in=data/task_1/NR/NR25_inst.dat -ref=data/task_1/NR/ —b

8. Závěr:

Celkem je vidět, že metoda větví a hranic (**Branch and Bounds**) v porovnání s hrubou silou poskytuje moc dobře výsledky. Nicméně je nutné si uvědomovat, že **Branch and Bounds** je velmi závislý hlavně na dvou věcích:

- volba heuristiky
- na hodnotách v zadání problému (instance problému na vstupu)

Většinou doba hledání řešení v **Branch and Bounds** je mnohem rychlejší než metodou hrubý síly, a je to díky ořezávání celkového prostoru řešení.

Občas, na velkých sadech dát se dá pozorovat nárůst na původní exponenciální složitost, stejně jak u **Brute Force**. Navíc **Branch and Bounds** je paměťově náročnější, než **Brute Force** a výpočet odhadů teoretické ceny pro nějaký uzel trvá nějakou dobu.

Avšak to záleží na implementaci algoritmu. Uvedené výš platí hlavně pro danou implementaci.