

Primerjava reševanja ILP v Matlab in Gurobi

David Rubin (david.rubin@student.um.si)

9. januar 2021

1 Uvod

V tem poročilu si bomo ogledali kako se lahko pohitri reševanje ILP problemov z uvedbo distribuiranega procesiranja. Kot standarden čas reševanja bomo uporabili MATLAB oziroma njegovo funkcijo *intlinprog* iz *optimization toolbox-a*. Kot predlagano pohitritev, smo si v tem poročilu ogledali orodje Gurobi ¹. Orodje prav tako omogoča določevanje števila uporabljenih niti, tako da lahko orodje tudi nekoliko bolj pošteno primerjamo z *intlinprog*, ko nastavimo enojno nit. Za testne probleme smo si izbrali zbirko MIPLIB2017 [1]. Celotna zbirka je precej obsežna, zato smo se odločili za *benchmark* verzijo, ki vključuje le del vseh problemov. Zbirko in pripravo testnih podatkov smo predstavili v poglavju 3. Predvidevamo, da je bralec seznanjen s programskim okoljem MATLAB, zato smo podali le kratek uvod v Gurobi.

2 Gurobi

Gurobi je komercialna programska oprema, ki omogoča optimiziranje problemov iz:

- lineranega programiranja (angl. *linear programming* - LP),
- mešanega celoštevilskega linearnega programiranja (angl. *mixed-integer linear programming* - MILP),
- kvadratičnega programiranja (angl. *quadratic programming* - QP),
- mešanega celoštevilskega kvadratičnega programiranja (angl. *mixed-integer quadratic programming* - MIQP),
- kvadratično omejenega programiranja (angl. *quadratically constrained programming* - QCP),
- mešano celoštevilskega kvadratično omejenega programiranja (angl. *Mixed-integer quadratically constrained programming* - MIQCP).

¹Orodje Gurobi je dostopno na <https://www.gurobi.com/>. Za potrebe te naloge smo uporabili akademsko licenco

Pokriva torej veliko območje problemov, za zadnje štiri pa je vredno omeniti, da podpira tako konveksne kot tudi konkavne probleme. Za reševanje nabora problemov imajo implementirane metode Simplex in paralelno oviro s križanjem, sočasnostjo, in sejanjem (angl. *parallel barrier with crossover, concurrent, and sifting*) za bolj zahtevne pa uporabijo tudi deterministično paralelno ne-tradicionalno iskanje, heuristike, rezanja ravnin in pa razbijanje simetrij [3].

Algoritmi so zasnovani deterministično, torej bi več zagonov skozi isti model dalo enake rezultate. Zasnovani so tudi tako, da po privzetem uporabijo vsa jedra, ki so v nekem sistemu na voljo. Trdijo tudi, da njihovi QCP in MIQCP algoritmi nudijo dvakrat boljše učinkovitost kot njihov glavni tekmeči [2]. Slednje trditve ne moremo sami potrditi, ne moremo pa je niti zavreči, tako da prepuščamo bralcu, da sam oceni resničnost. Podjetje so ustanovili dr. Robert Bixby, ki je pred tem tudi ustanovil CPLEX in pa Dr. Zonghao Gu ter Dr. Edward Rothberg. Slednja sta pred ustanovitvijo sodelovala pri CPLEX kot glavna inženirja [4]. Kot zanimivost, ime *GuRoBi* je nastalo iz njihovih priimkov. Med njihove partnerje spadajo podjetja kot je recimo Toyota, Microsoft, Google, AirFrance in pa tudi SAP.

Gurobi podpira množico programskih jezikov in oblik programiranja. Nudijo objektno orientirane vmesnike (JAVA, C++, .NET in Python), matrično orientiranje (C, MATLAB in R), povezavo do standardnih modelnih jezikov (AIMMS, AMPL, MPL) in druge [3]. V kolikor bi orodje želeli uporabljati le v jeziku Python se lahko preskoči namestitev celotne knjižnice in uporabi `pip` ali **Anaconda** paket. Nudijo pa tudi možnost porazdeljenega reševanja problemov na gruči (*Gurobi Remote Services*), ki pa zahteva ločeno licenco.

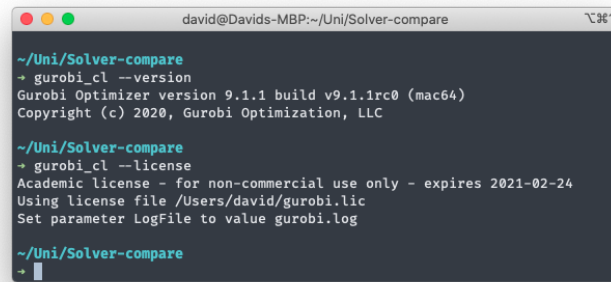
2.1 Priprava okolja

Za uporabnike je na voljo vodič za hitro namestitev za vse tri večje operacijske sisteme². Posebno pozornost je potrebno posvetiti sekciji za pridobitev akademske licence. Zahtevana je uporaba epoštnega naslova univerze in imeti dostop do spleta v času aktivacije. Po prijavi nam je na voljo dostop do prenosa njihove knjižnice in CLI orodja³. Znotraj dokumenta navedejo, da se za akademski dostop poleg licenčnega ključa preverja tudi domensko ime iz lokacije dostopa. Pri naši aktivaciji se ni pojavila težava (ukaz `grbgetkey <koda>`), kljub temu, da smo med aktivacijo dostopali do njihovega strežnika zunaj omrežja univerze. V kolikor Gurobi med aktivacijo sporoči kakšno napako (primer *hostname ... not recognized as belonging to an academic domain*), pa je možna uporaba VPN, ki je na voljo vsem študentom (in zaposlenim) Univerze v Mariboru.

V kolikor je orodje nameščeno in je bila aktivacija uspešna, bi sedaj morali imeti delujoč *Gurobi optimizer* kot je prikazano na sliki 1. Ob težavah se je najbolje obrniti na njih uraden vodič glede na vaš operacijski sistem [5]. CLI orodje podpira 14 formatov za podajanje optimizacijskega modela in drugih pomožnih podatkov [6]. V nadaljevanju bomo uporabljali dva: LP in MPS format. LP format je namenjen za boljšo berljivost iz strani človeka, medtem ko je MPS format najstarejši in najpogostejše uporabljen za shranjevanje matematičnih programskih modelov. MPS format je veliko strožji kar se tiče zapisa (fiksni začetni stolpci za vrednosti

²Vodič je dostopen na <https://www.gurobi.com/documentation/quickstart.html>

³Programska oprema Gurobi se pridobi na naslovu <https://www.gurobi.com/downloads/gurobi-software/>



```
david@Davids-MBP:~/Uni/Solver-compare
~/Uni/Solver-compare
+ gurobi_cl --version
Gurobi Optimizer version 9.1.1 build v9.1.1rc0 (mac64)
Copyright (c) 2020, Gurobi Optimization, LLC

~/Uni/Solver-compare
+ gurobi_cl --license
Academic license - for non-commercial use only - expires 2021-02-24
Using license file /Users/david/gurobi.lic
Set parameter LogFile to value gurobi.log

~/Uni/Solver-compare
+ 
```

Slika 1: Primer izpisa ukazov za podatke o verziji in licenci po uspešni namestitvi *Gurobi optimizerja* na macOS.

ipd.), zato ga bomo le uporabili pri testnih primerih.

2.2 Reševanje ILP problemov z Gurobi optimizer

Za boljši vploged v orodje, si bomo v nadaljevanju ogledali enostaven primer optimizacijskega problema in kako se le ta reši s pomočjo Gurobi optimizerja. Predstavljen primer je prilagojena verzija primera v vodiču za hiter zagon [5].

2.2.1 Opis problema

V času pisanja poročila je blizu konec koledarskega leta. *United States Mint* (ustanova, ki proizvaja ameriške kovance), ima še nekaj neporabljene zaloge materialov za proizvodnjo kovancev. Predpostavimo, da si želijo porabiti vso preteklo zalogo, preden se lotijo nabave materialov za serijo kovancev novo leto. Ustanova proizvaja šest različnih kovancev, sestava katerih pa je podana v tabeli 1. Ustanova sedaj želi proizvesti takšne tipe kovancev, da bo njihova skupna vrednost največja. Kakšna naj bo količina vsakega proizvedenega kovanca, če imamo podano zalogo vsakega materiala?

Tabela 1: Vsebnost materialov v ameriških kovancih glede na US Mint specifikacijo [7]

	Cent	Nickel	Dime	Quarter	Half	Dollar
Baker (Cu)	0.0625g	3.7500g	2.0791g	5.1977g	10.3954g	7.1685g
Nikelj (Ni)		1.2500g	0.1889g	0.4723g	0.9446g	0.1620g
Cink (Zn)	2.4375g					0.4860g
Mangan (Mn)						0.2835g
Skupaj	2.5g	5.0g	2.268g	5.670g	11.340g	8.1g

2.2.2 Priprava vhodnih datotek

Za rešitev problema potrebujemo izvesti 3 korake:

1. Določitev odločitvenih spremenljivk

2. Določitev odločitvene funkcije (v tem primeru linearne)

3. Določitev omejitev

Odločitvene spremenljivke so dokaj jasne: zanima nas količina vsakega kovanca, ki ga naj proizvedemo. Ker smo programerji in za lepšo nazornost jim bomo podali opisna imena, za matematike pa še vektorske spremenljivke. Spremenljivke torej poimenujemo *Cents*, *Nickels*, *Dimes*, *Quarters*, *Halves*, *Dollars* (oziroma $x_1, x_2 \dots x_6$). Vpeljimo pa še imena spremenljivk za porabljen količino materialov *Cu*, *Ni*, *Zn*, *Mn* (y_1, y_2, y_3 in y_4). Te spremenljivke so določene in predstavljajo material, ki je na zalogi. Za zalogo bomo predpostavili 1000g bakra in po 50g vsakega ostalega materiala. Formalen matematični zapis modela (ki ga maksimiziramo) se glasi:

$$f(x) = 0.01x_1 + 0.05x_2 + 0.1x_3 + 0.25x_4 + 0.5x_5 + 1x_6 \quad (1)$$

pri čemer so x_n količine proizvedenih kovancev, koeficienti pred njimi pa predstavljajo vrednost posameznega kovanca. Omejitve so sledeče:

$$\begin{aligned} 0.0625x_1 + 3.7500x_2 + 2.0791x_3 + 5.1977x_4 + 10.3954x_5 + 7.1685x_6 &\leq 1000 \\ 1.2500x_2 + 0.1889x_3 + 0.4723x_4 + 0.9446x_5 + 0.1620x_6 &\leq 50 \\ 2.4375x_1 + 0.4860x_6 &\leq 50 \\ 0.2835x_6 &\leq 50 \\ x_1, x_2, x_3, x_4, x_5, x_6 &\in \mathbb{N} \end{aligned} \quad (2)$$

Skratka količina porabljenega materiala ne sme presežati zaloge, prav tako pa ne moremo proizvesti manj kot celoto kovanca (celoštevilska omejitev). Isto omejitev lahko zapišemo na programerski način (prej omenjene opisne spremenljivke) in LP format zapisa (glej kodo 1). Slednje nam predstavlja enega izmed možnih vhodov v Gurobi optimizer.

Koda 1: Primer LP formata za problem proizvodnje kovancev

```
Maximize
    .01 Pennies + .05 Nickels + .1 Dimes + .25 Quarters + .5 Halves + 1 Dollars
Subject To
    Copper: .0625 Pennies + 3.75 Nickels + 2.0791 Dimes + 5.1977 Quarters +
        ↪ 10.3954 Halves + 7.1685 Dollars - Cu = 0
    Nickel: 1.25 Nickels + .1889 Dimes + .4723 Quarters + .9446 Halves + .162
        ↪ Dollars - Ni = 0
    Zinc: 2.4375 Pennies + .486 Dollars - Zn = 0
    Manganese: .2835 Dollars - Mn = 0
Bounds
    Cu <= 1000
    Ni <= 50
    Zn <= 50
    Mn <= 50
Integers
    Pennies Nickels Dimes Quarters Halves Dollars
End
```

LP format ima nekaj posebnosti. V podanem primeru imamo štiri sekcije (*Maximize*, *Subject to*, *Bounds*, *Integers*), ki si morajo vedno slediti v tem vrstnem redu. Sekcij je lahko več in tudi

znotraj sekcij se lahko pojavi več scenarijev. Vse to je opisano v navodilih za uporabo⁴. Druga posebnost je zapis vrednosti (spremenljivk, konstant, operatorjev ...). Te morajo biti ločene s presledki oziroma novimi vrsticami. Tako je `+ 0.1x` napačen zapis. Pravilno je `+ 0.1 x`. Tretje splošno pravilo je, da se spremenljivke vedno pišejo na levi in konstante na desni (primer tega je v sekciji *Subject to*). Na koncu pa naj še omenimo, da je po privzetem spodnja meja enaka 0, razen če je eksplicitno zapisana. Zapis `Cu <= 1000` je v resnici `0 <= Cu <= 1000`. To pomeni, da Gurobi po privzetem vsako spremenljivko omeji na nenegativna števila.

2.2.3 Zagon in rešitve

Najenostavnejši način uporabe Gurobi, kadar imamo podan model v datoteki, je preko ukazne vrstice. Z ukazom `gurobi_cl <model.lp>` lahko poženemo orodje in se problem prične reševati. Kot rešitev imamo na standardnem izhodu podano samo vrednost cenične funkcije, če želimo videti še vrednosti posameznih spremenljivk, je potrebno dodati parameter `ResultFile=<izhod.sol>`. Celoten ukaz je torej:

```
gurobi_cl ResultFile=code/coins.sol code/coins.lp
```

Izpis ukaza je viden na sliki 2, izhodna datoteka z rešitvami pa v kodi 2.

```

~/Uni/Solver-compare
+ gurobi_cl ResultFile=code/coins.sol code/coins.lp
Academic license - for non-commercial use only - expires 2021-02-24
Using license file /Users/david/gurobi.lic
Set parameter LogFile to value gurobi.log

Gurobi Optimizer version 9.1.1 build v9.1.1rc0 (mac64)
Copyright (c) 2020, Gurobi Optimization, LLC

Read LP format model from file code/coins.lp
Reading time = 0.00 seconds
: 4 rows, 10 columns, 18 nonzeros
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads
Optimize a model with 4 rows, 10 columns and 18 nonzeros
Model fingerprint: 0x90cc2da6
Variable types: 4 continuous, 6 integer (0 binary)
Coefficient statistics:
  Matrix range [6e-02, 1e+01]
  Objective range [1e-02, 1e+00]
  Bounds range [5e+01, 1e+03]
  RHS range [0e+00, 0e+00]
Found heuristic solution: objective -0.0000000
Presolve removed 1 rows and 6 columns
Presolve time: 0.00s
Presolved: 3 rows, 4 columns, 9 nonzeros
Variable types: 0 continuous, 4 integer (0 binary)

Root relaxation: objective 1.149294e+02, 2 iterations, 0.00 seconds

  Nodes      |      Current Node      | Objective Bounds      | Work
  Expl Unexpl | Obj Depth IntInf | Incumbent  BestBd  Gap   It/Node Time
-----
  0   0   114.92942   0   1   -0.00000   114.92942   -     -     0s
H   0   0               |   114.8500000   114.92942   0.07%   -     0s
H   0   0               |   114.9000000   114.92942   0.03%   -     0s
  0   0   114.92941   0   1   114.90000   114.92941   0.03%   -     0s
  0   1   114.92941   0   1   114.90000   114.92941   0.03%   -     0s

Explored 94 nodes (94 simplex iterations) in 0.01 seconds
Thread count was 8 (of 8 available processors)

Solution count 3: 114.9 114.85 -0

Optimal solution found (tolerance 1.00e-04)
Best objective 1.149000000000e+02, best bound 1.149000000000e+02, gap 0.0000%

Wrote result file 'code/coins.sol'

```

Slika 2: Primer izpisa na standardni izhod pri reševanju problema s kovanci.

⁴Navodila za LP format v Gurobi https://www.gurobi.com/documentation/9.1/refman/lp_format.html#format:LP.

Koda 2: Generirana datoteka z rešitvami za problem kovancev

```
# Objective value = 114.9
Pennies 0
Nickels 0
Dimes 4
Quarters 50
Halves 0
Dollars 102
Cu 999.3884
Ni 40.8946
Zn 4.9571999999999996e+01
Mn 2.8916999999999998e+01
```

Na standardnem izhodu ukaza lahko vidimo nekaj podatkov o problemu, vidimo da se je problem reševal v 8 nitih in da je potreboval 94 Simplex iteracij za doseženo rešitev \$114.9. V izhodni datoteki z rešitvami pa se nahajajo še količine posameznih kovancev: 4 *Dimes*, 50 *Quarters* in 102 *Dollars*.

Obstaja tudi *Gurobi Interactive Shell*, ki temelji na *Python Shell*. Potek ukazov je zelo podoben programiranju v Python, omogoča pa hitrejšo spreminjanje modela in zaganjanje v primerjavi z ročnim spreminjanjem vhodne datoteke in vnovičnega poganjanja ukazov. Bolj podroben opis je na voljo v uradni dokumentaciji ali v hitrem vodiču [5].

2.3 Uporaba s Python

Gurobi ima na voljo tudi vmensik za Python. Slednjega lahko namestimo na tri načine: preko **pip**, ročno ali pa uproabimo **Anaconda**. Ne glede na način, ki ga izberemo bo še vedno potrebna pridobitev in namestitev veljavne licence, kot smo opisali v enem izmed prejšnjih poglavjih. Za potrebe te naloge smo se odločili za namestitev v Python virtualno okolje in uporabo **pip**. Paket se imenuje **gurobipy** in v času pisanja naloge (še) ni bil dostopen v javnem PyPi strežniku. Zato se za namestitev uporabi njihov privaten strežnik in ukaz:

```
python3 -m pip install -i https://pypi.gurobi.com gurobipy
```

Ročna namestitev se izvede s pomočjo **setup.py**, ki se nahaja znotraj mape kamor smo namestili Gurobi med pridobivanjem licence. Pri obeh omenjenih načinih namestitve je potrebno paziti na kompatibilno verzijo Python (podprte so 2.7 in 3.6 - 3.9 in le 64-bitne različice). Zadnja možnost pa je uporaba **Anaconde**⁵

```
conda config --add channels https://conda.anaconda.org/gurobi
```

```
conda install gurobi
```

Namestitev se lahko preveri tudi s pomočjo **Jupyter notebook**, ki bi moral biti priložen temu poročilu (nahaja se v **code/Gurobi-demo.ipynb**). V kolikor si pa bralec želi igrati s tem Jupyter zvezkom, pa je na voljo tudi **requirements.txt**. V splošnem za zvezek potrebujete **numpy**, **scipy** in **gurobipy** (iz privatnega strežnika), vse drugega so odvisnosti teh paketov. Primer ukazov za macOS/Linux, ki pripravijo okolje in odprejo zvezek za urejanje:

```
python3 -m venv genv
```

```
source genv/bin/activate
```

```
python3 -m pip install -r requirements.txt
```

⁵Anaconda je distribucija Pythona dosegljiva na <https://www.anaconda.com/>

jupyter notebook code/Gurobi-demo.ipynb

Bralec, ki si ne želi urejati zvezek, si lahko tudi ogleda statičen zvezek na Github repozitoriju kjer se nahaja to poročilo⁶. Hiter povzetek uporabe Gurobi v Python skupaj z LP datotekami je viden v kodi 3.

Koda 3: Primer uporabe LP formatov in gurobipy

```
import gurobipy as gp

m = gp.read('coins.lp') # preberi model
m.optimize() # optimiziraj

# izpisi rezultate
for v in m.getVars():
    print(f'{v.varName}={v.x}')
print(f'vrednost: {m.objVal}')
```

2.4 Uporaba z MATLAB

Enako kot pri Python paketu je treba najprej namestiti knjižnice in pridobiti veljavo licenco (opisano v poglavju 2.1). Za delo z MATLAB so pri Gurobi pripravili skripto, ki nam pomaga pri namestitvi vmesnika. Ta skripta (`gurobi_setup.m`) se nahaja v mapi `matlab` v direktoriju kamor smo namestili Gurobi. Privzeta lokacija (verzija 9.1.1) na macOS je `/Library/gurobi911/mac64/matlab` in na Windows `c:\gurobi911\win64\matlab`. Celoten ukaz namestitve na macOS (klican znotraj MATLAB-a) je potemtakem:

```
cd /Library/gurobi911/mac64/matlab
gurobi_setup
```

V kolikor se pojavi kakšna napaka je dobro najprej preveriti ali upoštevamo kompatibilnost, saj Gurobi podpira le 64-bitno različico MATLAB-a. Ob uspešni namestitvi bi se moral pojaviti napis podoben temu na sliki 3.



Slika 3: Izpis skripte po uspešni namestitvi Gurobi rutin v MATLAB na macOS.

Pokažimo osnovno uporabo na enakem problemu kot pri Python. V kolikor niste sledili

⁶Povezava do Jupyter zvezka <https://github.com/rubinda/gurobi-vs-matlab/blob/main/code/Gurobi-demo.ipynb>

Python zvezku, je problem sledeč:

$$\begin{aligned} &\text{maksimiziraj} && x + y + 2z \\ &\text{ob omejitvah} && x + 2y + 3z \leq 4 \\ &&& x + y \geq 1 \\ &&& x, y, z \text{ so binarne} \end{aligned} \tag{3}$$

V MATLAB si lahko pripravimo skript, ki s pomočjo Gurobi opiše problem, ga shrani kot LP format in požene optimizacijo (koda 4).

Koda 4: Reševanje osnovnega problema v MATLAB s pomočjo Gurobi

```
function gurobi_demo()
% Povzeto po Gurobi Quickstart vodicu
names = {'x'; 'y'; 'z'}; % Imena spremenljivk
model.A = sparse([1 2 3; -1 -1 0]); % Redka matrika A
model.obj = [1 1 2]; % Cenitvena funkcija
model.rhs = [4; -1]; % b (Right-hand side)
model.vtype = 'B'; % tip spremenljivk (B - binarne)
model.modelsense = 'max'; % maksimiziraj
model.varnames = names;

gurobi_write(model, 'gurobi_demo.lp'); % Zapisi problem v LP
params.outputflag = 0; % Prepreči Gurobi izpis med optimizacijo

result = gurobi(model, params); % Pozeni optimizacijo
% Izpisi resitve
disp(result);
for v=1:length(names)
    fprintf('%s %d\n', names{v}, result.x(v));
end
fprintf('Resitev: %e\n', result.objval);
end
```

Standardni izhod ob klicanju skripte je prikazan na sliki 4. Glaven ukaz je `gurobi(model, params)`, kjer podamo podatke o problemu in pa parametre za optimizacijo. Vse parametre in več o uporabi si bralec lahko prebere v uradni dokumentaciji ⁷. Za boljšo predstavbo pa je na voljo tudi več primerov uporabe, ki se nahajajo v mapi namestitve Gurobi (torej `<direktorij namestitve>/examples/matlab`).

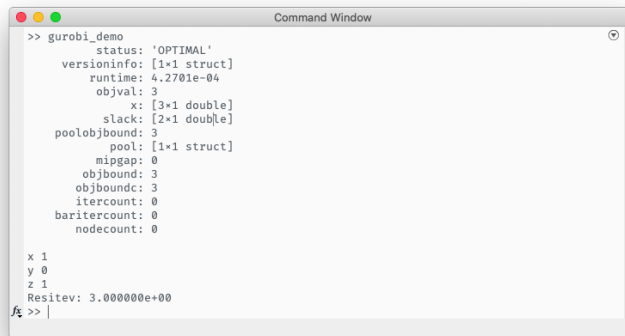
Poglejmo si zdaj še primer, kadar želimo uporabiti obsotejčo LP datoteko preko MATLAB vmesnika. Gurobi ponuja med drugim tudi funkcijo, ki to omogoča na zelo enostaven način (glej kodo 5 in sliko 5).

Koda 5: Primer reševanja LP problema s Gurobi MATLAB vmesnikom

```
function gurobi_demo_file()
% Prebere problem iz LP datoteke in ga resi
model = gurobi_read('coins.lp'); % Deluje tudi s *.mps
params.outputflag = 0; % Prepreči izpis med optimizacijo

result = gurobi(model, params); % Pozeni optimizacijo
% Izpisi rezultate
```

⁷Uradna Gurobi dokumentacija za MATLAB vmesnik je dostopna na https://www.gurobi.com/documentation/9.1/refman/matlab_api_overview.html#matlab:MATLAB



```
>> gurobi_demo
    status: 'OPTIMAL'
    versioninfo: [1x1 struct]
    runtime: 4.2701e-04
    objval: 3
    x: [3x1 double]
    slack: [2x1 double]
    poolobjbound: 3
    pool: [1x1 struct]
    mipgap: 0
    objbound: 3
    objboundc: 3
    itercount: 0
    baritercount: 0
    nodecount: 0

x 1
y 0
z 1
Resitev: 3.000000e+00
fx >> |
```

Slika 4: Izpis rešitve ob uporabi gurobi_demo skripte.

```
disp(result)
fprintf('Resitev: %e\n', result.objval);
for v=1:length(result.x)
    fprintf('%s = %d\n',model.varnames{v}, result.x(v));
end
end
```



```
>> gurobi_demo_file
    status: 'OPTIMAL'
    versioninfo: [1x1 struct]
    runtime: 0.0132
    objval: 114.9000
    x: [10x1 double]
    slack: [4x1 double]
    poolobjbound: 114.9000
    pool: [1x3 struct]
    mipgap: 0
    objbound: 114.9000
    objboundc: 114.9000
    itercount: 94
    baritercount: 0
    nodecount: 94

Resitev: 1.149000e+02
Pennies = 0
Nickels = 0
Dimes = 4
Quarters = 50
Halves = 0
Dollars = 102
Cu = 9.993884e+02
Ni = 4.089460e+01
Zn = 4.957200e+01
Mn = 2.891700e+01
fx >> |
```

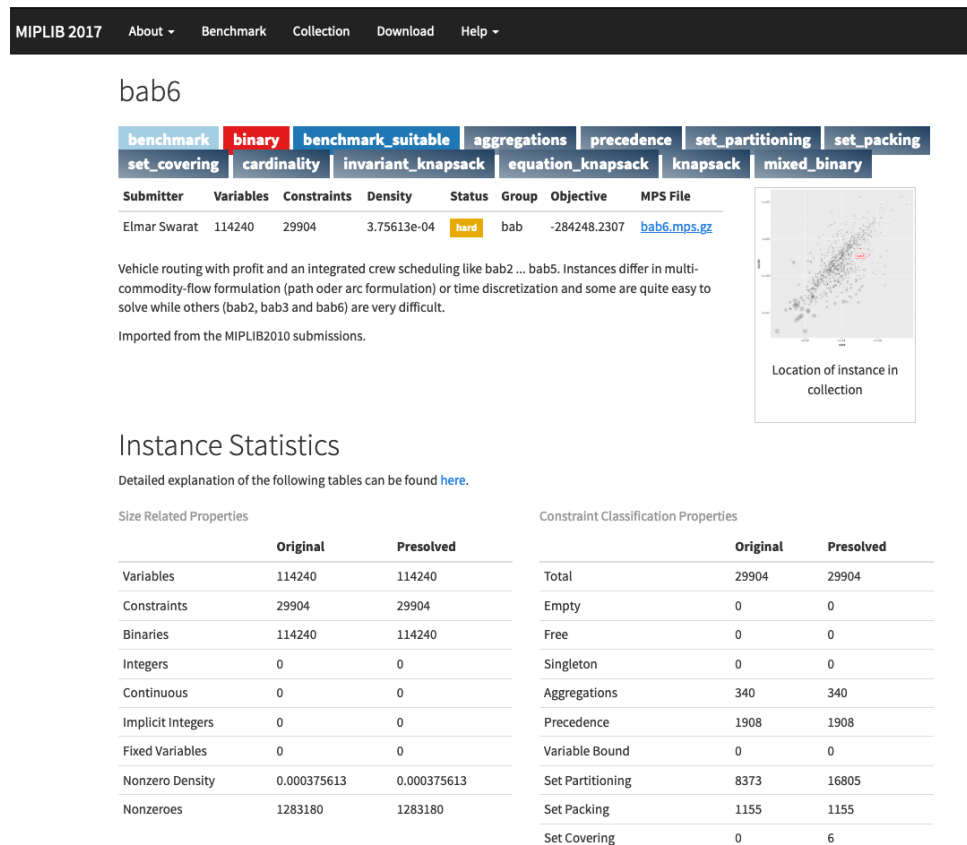
Slika 5: Standardni izhod pri zagou gurobi_demo_file.m skripte.

3 Testna zbirka

Testna zbirka MIPLIB2017 je nastala zaradi potrebe raziskovalcev po primerih MIP (angl. *mixed integer programs*) iz realnega sveta. V tem delu smo sicer želeli primerjati le celoštevilске probleme, vendar smo se zaradi enostavnejše primerjave odločili, da uporabimo kar nekaj primerov iz celotne testne zbirke, ki jo sestavljajo tudi mešani primeri (poleg celoštevilskih vsebujejo tudi zvezne spremenljivke). Drugi razlog za izbiro te zbirke je tudi dejstvo, da se je uveljavila kot standardni performančni test, v katerega so prispevali tudi vsi večji ponudniki optimizacijske

programske opreme (Gurobi, CPLEX, Fico, MOSEK ...) *Benchmark set* sestavlja 240 problemov, ki so zaradi praktičnih razlogov izbrani kot primeri z dobro numerično stabilnostjo in dobro rešljivostjo. *Collection set* pa vsebuje še veliko več problemov, med katerimi so tudi še nerešeni. [1].

Vsak izmed primerov je kategoriziran med težke ali lahke primere, podan pa ima tudi kratek opis o parametrih, omejitvah ipd. Na sliki 6 vidimo izsek o predstavitvi enega izmed primerov. Tašne predstavitve teh problemov so dostopne preko spletne strani http://mipilib.zib.de/tag_benchmark.html.



Slika 6: Predstavitev problema **bab6** na MIPLIB2017 spletni strani. Vir: http://mipilib.zib.de/instance_details_bab6.html

Zaradi omejitev strojne opreme (in posledično časovne) smo iz zbirke izbrali nekaj problemov, za katere smo preverili, da so rešljivi v relativno kratkem času. Težava s to testno zbirko je, da predvideva reševanje od precej enostavnih do relativno težkih problemov, ki se rešujejo s pomočjo distribuiranih sistemov. Primer takšnega problema, ki se sicer šteje med lažje, je 50v-10 (opis na http://mipilib.zib.de/instance_details_50v-10.html). Za slednjega navajajo, da so rešitev našli s pomočjo 2.000 jedrnega super računalnika, žal pa sami do takšne računske zmogljivosti nimamo dostopa.

4 Primerjava Gurobi in Matlab

Celotno testiranje smo izvedli v okolju MATLAB. Izbrali smo 10 problemov iz testne zbirke in vsakega trikrat poskusili rešiti s pomočjo MATLAB-ovega `intlinprog` in pa z Gurobi preko MATLAB vmesnika. Pri Gurobi smo preverili tudi hitrost reševanja ob uporabi 1, 2, 4 ali 8 jeder (omejitev računalnika). Skripta, ki nam to omogoča se imenuje `bench_gurobi.m` in bi morala biti priložena temu poročilu. Za njeno uporabo je potreben MATLAB (64-bitna različina z nameščenim *optimization toolbox*) in Gurobi MATLAB vmesnik z veljavno licenco (glej poglavje 2.1 glede pridobitve licence in poglavje 2.4 za uporabo Gurobi z MATLAB). Edina sprememba pri beleženju časa pri MATLAB je to, da reševanje ne ponovimo več v primeru dosežene časovne limite iskanja rešitve. Slednje smo sklenili vključiti zato, ker se pri MATLAB ne spremenijo parametri ob ponovnih zagonih in je verjetnost, da bo v vnovičnem poskusu problem rešil v omejenem času precej majhna. Prav tako pa smo na ta način skrajšali čas testiranja MATLAB za približno pet ur (razpolovili). Prav tako nismo poskušali optimizirati zagone glede na problem ampak smo uporabili privzete nastavitve za vsak problem. Lahko se zgodi, da ob primerni optimizaciji rezultati pridejo drugačni. Namen tega dela je pridobiti grobe ocene pohitritve, za katere ni nujno da odražajo stanje pri reševanju drugih problemov in ob drugačnih nastavitvah.

4.1 Testno okolje

Testiranje se je izvajalo na prenosnem računalniku v operacijskem sistemu macOS. Procesor je 4-jedrni (8 niti) Intel Core i7 4700HQ (2.4GHz base in 3.4GHz boost), na voljo pa je imel 8GB delovnega pomnilnika (glej sliko 7). Verzija MATLAB je bila R2020b (9.9.0.1467703) 64-bitna različica iz 26. avgusta 2020, verzija Gurobi pa 9.1.1 build v9.1.1rc0 (mac64). Med posameznimi zagoni so tekli le najnujnejši procesi (torej ali MATLAB ali okno ukazne vrstice z Gurobi), prav tako pa je bil prenosnik priključen v napajanje in ni izvajal kakršnihkoli drugih opravil. Okvirni čas, v katerem se izvede performančno testiranje, je 10 ur za Gurobi in 5 ur za MATLAB.

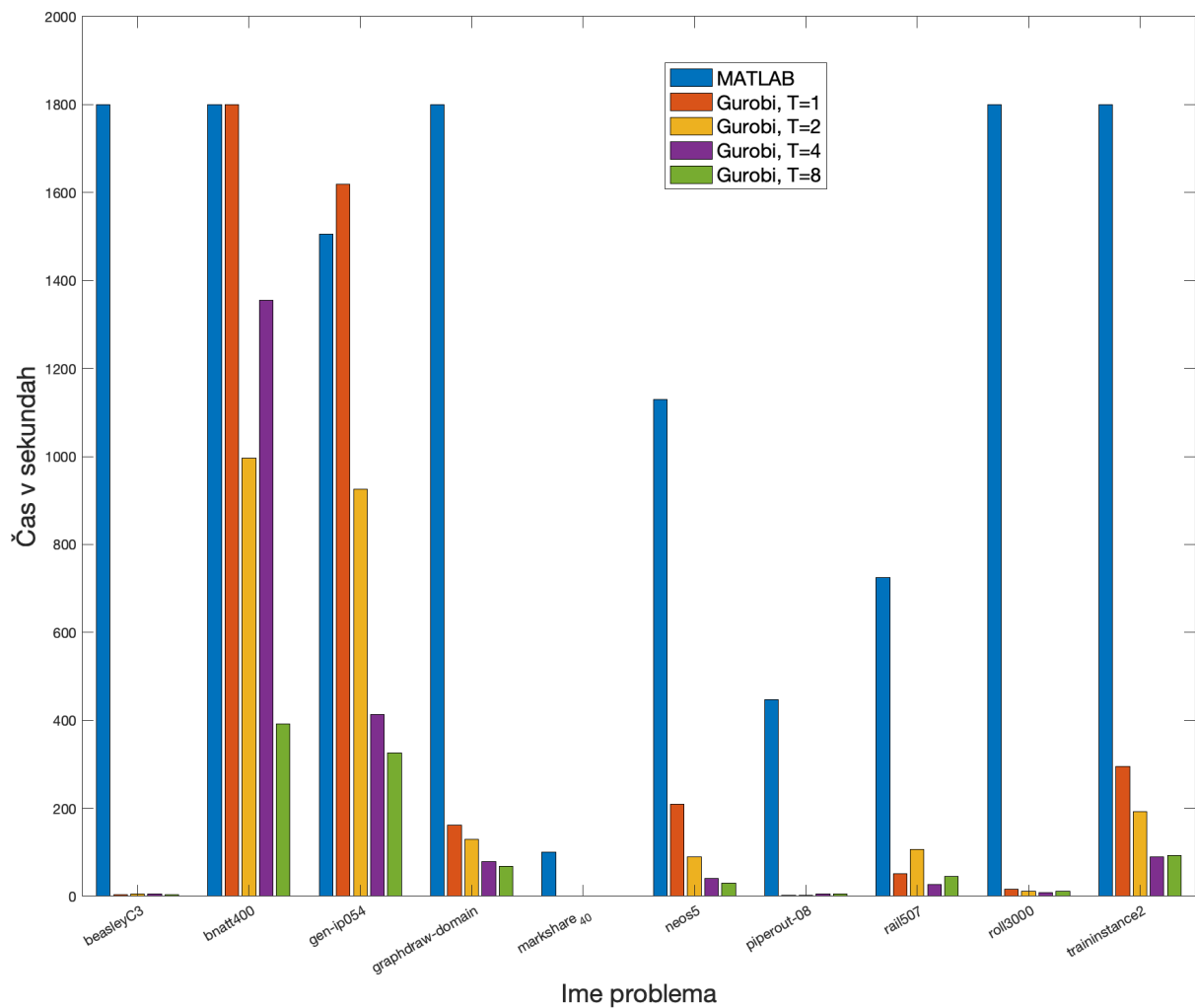
4.2 Performančni rezultati

Za čas testiranja smo vpeljali časovno omejitev pol ure na vsak zagon problema. Rezultati časov reševanja so vidni na sliki 8 in v tabeli 2. Stolpci ki dosegajo vrednost 1800s so bili ustavljeni zaradi prej omenjene limite. Nekateri problemi so bili rešeni precej hitro (v roku nekaj sekund), zato je nekoliko približan prikaz istih rezultatov še prikazan na sliki 9.

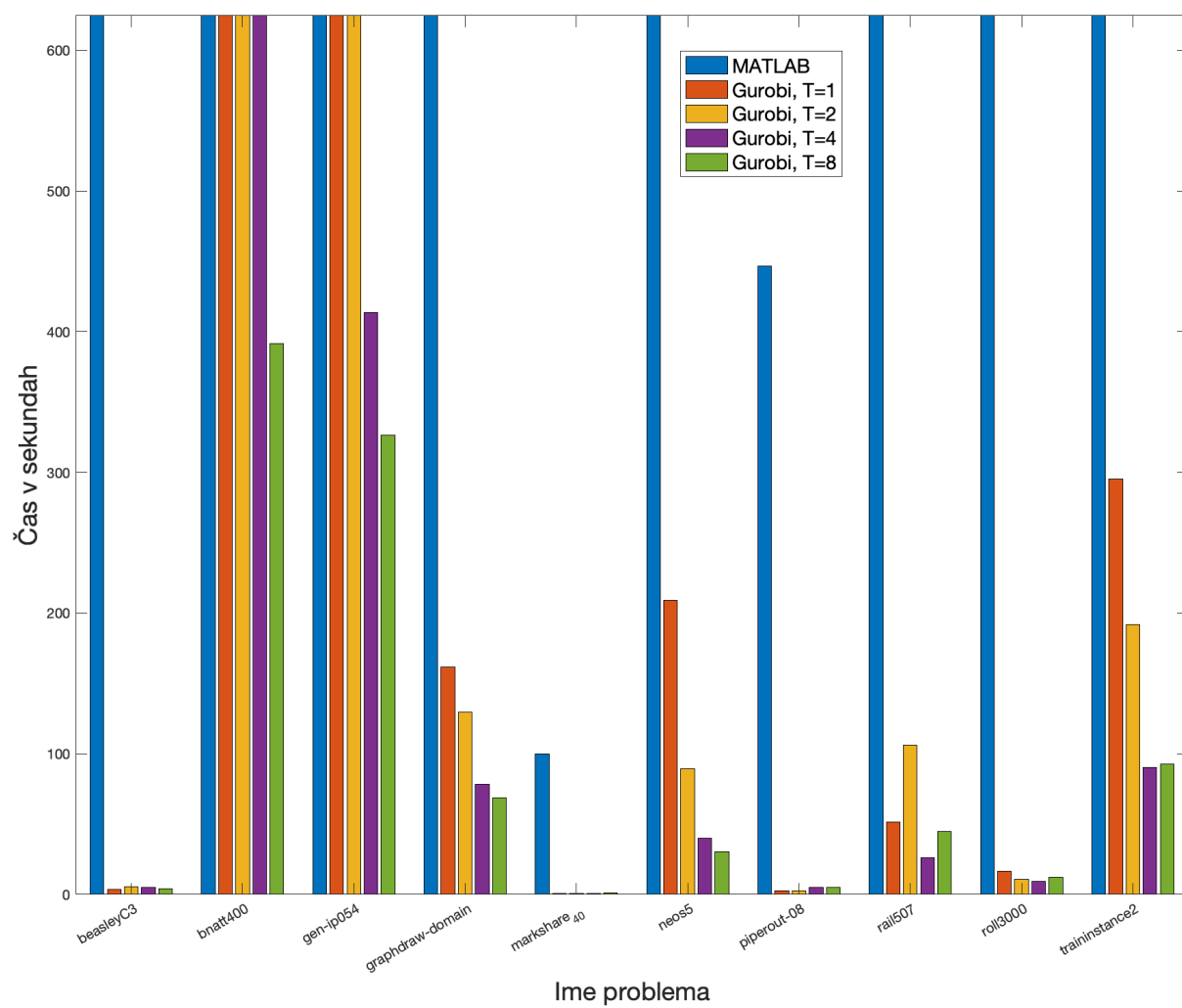
Iz rezultatov je najprej razvidno, da je Gurobi precej hitrejši od MATLAB. Če poračunamo povprečno pohitritev (s pomočjo enačbe 4), vidimo, da je v našem primeru Gurobi že ob uporabi ene niti kar 73.89% hitrejši od funkcije `intlinprog`. Pri uporabi 8 niti, pa je Gurobi v povprečju hitrejši za 93.57%. V tabeli 3 najdemo poračunane spremembe pohitritve glede na enačbo 4 še za preostale kombinacije. Tukaj je potrebno omeniti, da sta problema *rail507* in *beasleyC3* nekoliko popačila rezultate, saj je pri njiju prišlo do anomalije. Vidna je tudi na sliki ??, če primerjamo oranžen in rumen stolpec vidimo, da sta ta dva problema potrebovala več časa pri reševanju z dvema jedri kot pa z enim. V obeh primerih smo reševanje ponovili trikrat in vedno



Slika 7: Podatki računalnika, na katerem se je izvajalo performančno testiranje



Slika 8: Graf primerjave časov reševanja posameznih problemih z uporabo MATLAB `intlinprog` in Gurobi z različnim številom jeder (označeno s T= v legendi)



Slika 9: Približan primerjave časov reševanja problemov. Dodatno je treba opomniti, da pri problemu `markshare_4_0` Gurobi vsebuje čase hitreje od sekunde in so slabo vidni na grafu.

Naziv problema	\bar{t}_M	\bar{t}_{Grb1}	\bar{t}_{Grb2}	\bar{t}_{Grb4}	\bar{t}_{Grb8}
traininstance2	1800.06s*	295.51s	191.75s	90.24s	92.57s
roll3000	1800.04s*	16.73s	10.82s	9.18s	12.18
rail507	725.17s	51.34s	106.05s	26.21s	44.61s
piperout-08	446.44s	2.82s	2.84s	4.80s	5.05s
neos5	1129.62s	208.96s	89.49s	39.98s	30.41s
markshare_4_0	99.77s	0.86s	0.70s	0.64s	1.06s
graphdraw-domain	1800.03s*	161.48s	129.65s	78.40s	68.69s
gen-ip054	1505.69s	1618.25s	925.88s	413.84s	326.66s
bnatt400	1800.46s*	1800.16s*	996.54s	1355.31s	391.53s
beasleyC3	1800.10s*	3.66s	5.65s	5.25s	4.03s

Tabela 2: Povprečni časi reševanja problemov obravnavnih v tej nalogi, pri čemer je \bar{t}_M povprečni čas za MATLAB, \bar{t}_{GrbN} pa povprečni čas za Gurobi ob uporabi N niti. Časi označeni s * pomenijo, da je bilo reševanje ustavljeno zaradi nastavljene časovne limite

dobili enak rezultat. Težava morda leži v naravi problema, ali pa bi bila le potrebna optimizacija Gurobi za reševanje. V kolikor izpustimo ta dva primera, se povprečna pohitritev spremeni na -36.27% . Podobna zgodba se je dogajala tudi pri prehodu iz 4 na 8 jeder. Morda je to zaradi *hyper-threading*, saj je imel testni procesor samo 4 prava jedra in dodatne 4 navidezne. V našem primeru je potem bil čas reševanja na 8 nitih pri polovici primerov tudi daljši v primerjavi z 4 nitmi. Prišli smo do dveh sklepov: ali so navidezna jedra povzročila počasnejše delovanje ali pa je prišlo do prevelike režije za nekatere probleme in posledično daljše čase.

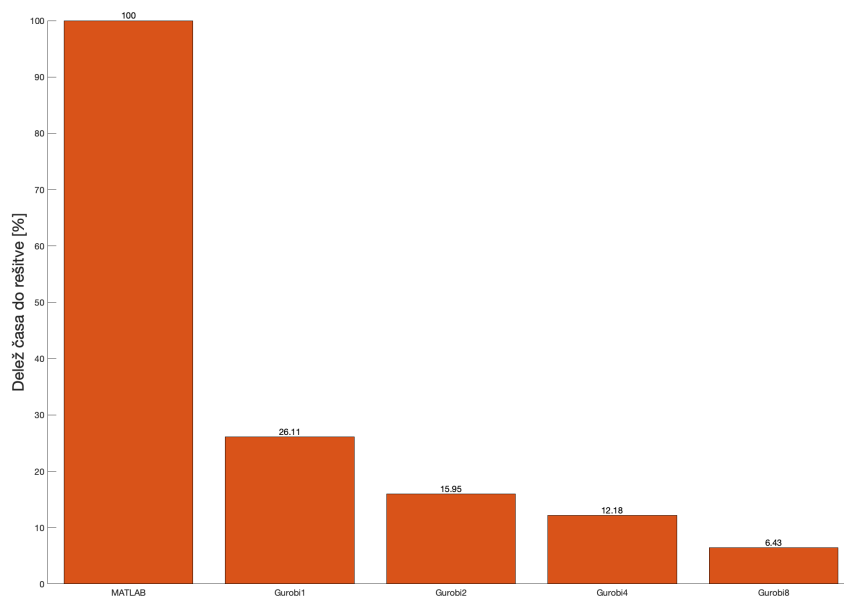
Na grafu 8 se pojavi še nekaj drugih anomalij. Naj najprej omenim, da so vse meritve časov povezane z Gurobi bile ponovljene trikrat. Ročno smo tudi preverili čase in imajo posamezne meritve standardni odklon premajhen, da bi recimo ena meritev drastično vplivala na povprečje. Zato tudi ne znamo popolnoma pojasniti časovno neskladje pri problemu *bnatt400* pri prehodu iz 2. niti v 4. Morda je to stvar optimizacije reševanja, saj se potem pri prehodu iz 4 niti v 8 čas skrajša na pričakovano raven. Pri relativno enostavnih problemih je moč opaziti tudi težavo s preveč režije: ob uporabi več jeder potrebujemo dalj časa da pridemo do iste rešitve. Na grafu je to lepo vidno pri problemu *piperout-08* ali pa *markshare_4_0*.

V splošnem gledano, je pohitritev pri spremembi (povečanju) števila jeder odvisna predvsem od problema. Izkaže se, da problemi z veliko vozlišči lahko pridobijo na hitrosti reševanja, kadar uporabimo več jeder. Smo pa ugotovili, da to pravilo ne drži vedno in se lahko pojavijo anomalije pri uporabi določenega števila jeder. Vsekakor pa lahko trdimo, da je orodje Gurobi zelo hitrejše od MATLABove funkcije `intlinprog`. Če pogledamo na graf 10 vidimo, da tudi pri uporabi ene same niti v povprečju Gurobi porabi skoraj četrtino časa za reševanje problemov v primerjavi z MATLAB.

$$pohitritev = \frac{t_{novi} - t_{stari}}{t_{stari}} \quad (4)$$

Pohitritev iz \rightarrow v	Vrednost pohitritve
$\Delta \bar{t}_M \rightarrow \bar{t}_{Grb1}$	-73.89%
$\Delta \bar{t}_M \rightarrow \bar{t}_{Grb2}$	-84.05%
$\Delta \bar{t}_M \rightarrow \bar{t}_{Grb4}$	-87.82%
$\Delta \bar{t}_M \rightarrow \bar{t}_{Grb8}$	-93.57%
$\Delta \bar{t}_{Grb1} \rightarrow \bar{t}_{Grb2}$	-9.22%
$\Delta \bar{t}_{Grb2} \rightarrow \bar{t}_{Grb4}$	-20.41%
$\Delta \bar{t}_{Grb4} \rightarrow \bar{t}_{Grb8}$	+2.55%
$\Delta \bar{t}_{Grb1} \rightarrow \bar{t}_{Grb4}$	-30.74%
$\Delta \bar{t}_{Grb1} \rightarrow \bar{t}_{Grb8}$	-29.75%
$\Delta \bar{t}_{Grb2} \rightarrow \bar{t}_{Grb8}$	-23.39%

Tabela 3: Povprečne pohitritve (glede na enačbo 4) pri spreminjanju števila niti v Gurobi in v primerjavi z MATLAB. \bar{t}_M predstavlja povprečni čas za MATLAB, \bar{t}_{GrbN} pa povprečni čas za Gurobi ob uporabi N niti



Slika 10: Primerjava povprečnih deležov časa (izraženi v odstotkih) porabljenih za reševanje naših testnih problemov.

5 Zaključek

V nalogi smo raziskali orodje Gurobi, njegovo uporabo in preverili njegovo performanso v primerjavi z MATLAB. Iskali smo tudi koeficient pohitritve pri uporabi več jeder za reševanje MIP problemov. Prišli so do ugotovitve, da koeficient pohitritve ni enak koeficientu povečanja jeder ampak je nekoliko manjši. Do večjega izraza je prihajalo pri spremembi iz 1 jedra v 2, kjer je v večini primerov pohitritev znašala nekje 35-40%. Smo pa tudi ugotovili, da so pohitritve tudi odvisne od samih problemov, ki jih rešujemo. Pri nekaterih se povečanje število jeder bolj izrazi v času reševanja kot pa pri ostalih.

Vsekakor pa bi priporočali uporabo orodja Gurobi, saj je za akademske namene brezplačen, omogoča pa znatno boljše performanse v primerjavi z MATLAB.

Literatura

- [1] GLEIXNER, A., HENDEL, G., GAMRATH, G., ACHTERBERG, T., BASTUBBE, M., BERTHOLD, T., CHRISTOPHEL, P. M., JARCK, K., KOCH, T., LINDEROTH, J., LÜBBECKE, M., MITTELMANN, H. D., OZYURT, D., RALPHS, T. K., SALVAGNIN, D., AND SHINANO, Y. MIPLIB 2017: Data-Driven Compilation of the 6th Mixed-Integer Programming Library. *Mathematical Programming Computation* (2020 [obiskano 26. decembra, 2020]). <http://miplib.zib.de/>.
- [2] GUROBI OPTIMIZATION LLC. *Gurobi Corporate Brochure*, 2020 [obiskano 26. decembra, 2020]. <https://www.gurobi.com/pdfs/Gurobi-Corporate-Brochure.pdf>.
- [3] GUROBI OPTIMIZATION LLC. *Gurobi Optimizer*, 2020 [obiskano 26. decembra, 2020]. <https://www.gurobi.com/products/gurobi-optimizer/>.
- [4] GUROBI OPTIMIZATION LLC. *Our Team - Gurobi*, 2020 [obiskano 26. decembra, 2020]. <https://www.gurobi.com/company/our-team/>.
- [5] GUROBI OPTIMIZATION LLC. *Quick Start Guides*, 2020 [obiskano 26. decembra, 2020]. <https://www.gurobi.com/documentation/quickstart.html>.
- [6] GUROBI OPTIMIZATION LLC. *Model File Formats*, 2020 [obiskano 27. decembra, 2020]. https://www.gurobi.com/documentation/9.1/refman/model_file_formats.html.
- [7] U.S. MINT. *Coin Specifications*, 2019 [obiskano 27. decembra, 2020]. <https://www.usmint.gov/learn/coin-and-medal-programs/coin-specifications>.