Václav Kobera (koberva1) - BI:KOP - Homework 2

Introduction (úkol)

Dáno: vektor proměnných (x_1...xn), $x_i \in \{0,1\}$, dále Booleova formule těchto proměnných v konjunktivní normální formě o *m* klauzulích (součtových termech), dále pak pro každou klauzuli *c* váha *w*(*c*).

Sestrojit: ohodnocení Y proměnných X takové, že součet vah splněných klauzulí je maximální.

Problém řešte některou z pokročilých heuristik:

- simulované ochlazování
- genetický algoritmus

Postup

pro tuto úlohu byla zvolena heuristika založená na simulovaném ochlazování.

- White Box
- Iniciální nastavení
- Úprava heuristické hodnoty
- Optimalizace koeficientu chlazení a růstu exponentu
- Výsledné nastavemí
- Black Box

White Box

Krok 1 (Iniciální nastavení)

pseudokód simulovaného ochlazování

```
var  s  =  randomState()
var bestState = s
while(!isFrozen(T))
        while (!equilibrium ()) {
            s_new  ← neighbour(s) //Pick a random neighbour
            If (s_new.isBetterThan(s) or P(s_new, s, T) ≥ random(0, 1)): // P is pro
                s  ← s_new
            If (s.isBetterThan(bestState)):
                    bestState = s
    }
    T  ← coolDown(T)
return bestState
```

prvotní nastavení

- **počáteční stav** - náhodný (náhodně ohodnoceny jednotlivé proměnné)
- **počáteční teplota** - součet normalizovaných vah – **všechny váhy byly namapovány do intervalu (0, 100)**
- **isFrozen()** - pokud je teplota nižší než 1
- **quilibrium()** - je konstanta 30 (30 inner cycles)
- **A.isBetterThan(B)** a **P()** pracují s heuristickou hodnotou zvolenou na základě následujícího vzorečku:

```
heuristicValue = sumOfActiveWeights  * (sumOfSuccessClausules / AllClausules)

A.isBetterThan(B) vrátí true pokud A má vyžší heuristickou hodnotu než B
```

pro **P()** byla zvolena funkce exp(- δ/T)) kde δ je rozdíl heuristických hodnot (viz https://courses.fit.cvut.cz/NI-KOP/lectures/index.html přednáška 8)

- **coolDown()** - aktuální teplota vynásobena koeficientem 0.95
- **neighbour(s)** - prohodí hodnotu jedné (náhodné) proměnné v aktulním stavu

Testování kroku 1

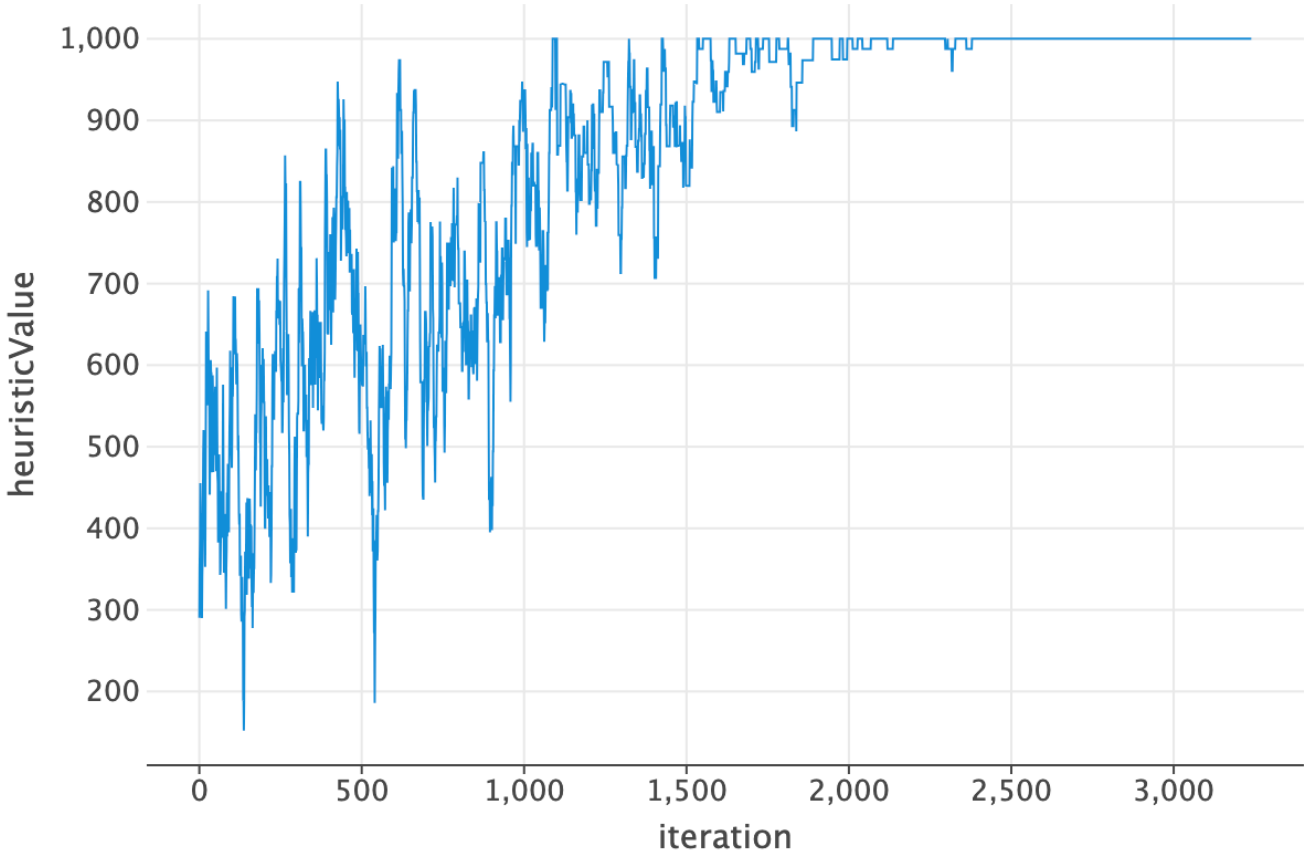
Testování proběhlo na instanci wuf20-71-M/20-71-01.mwcnf

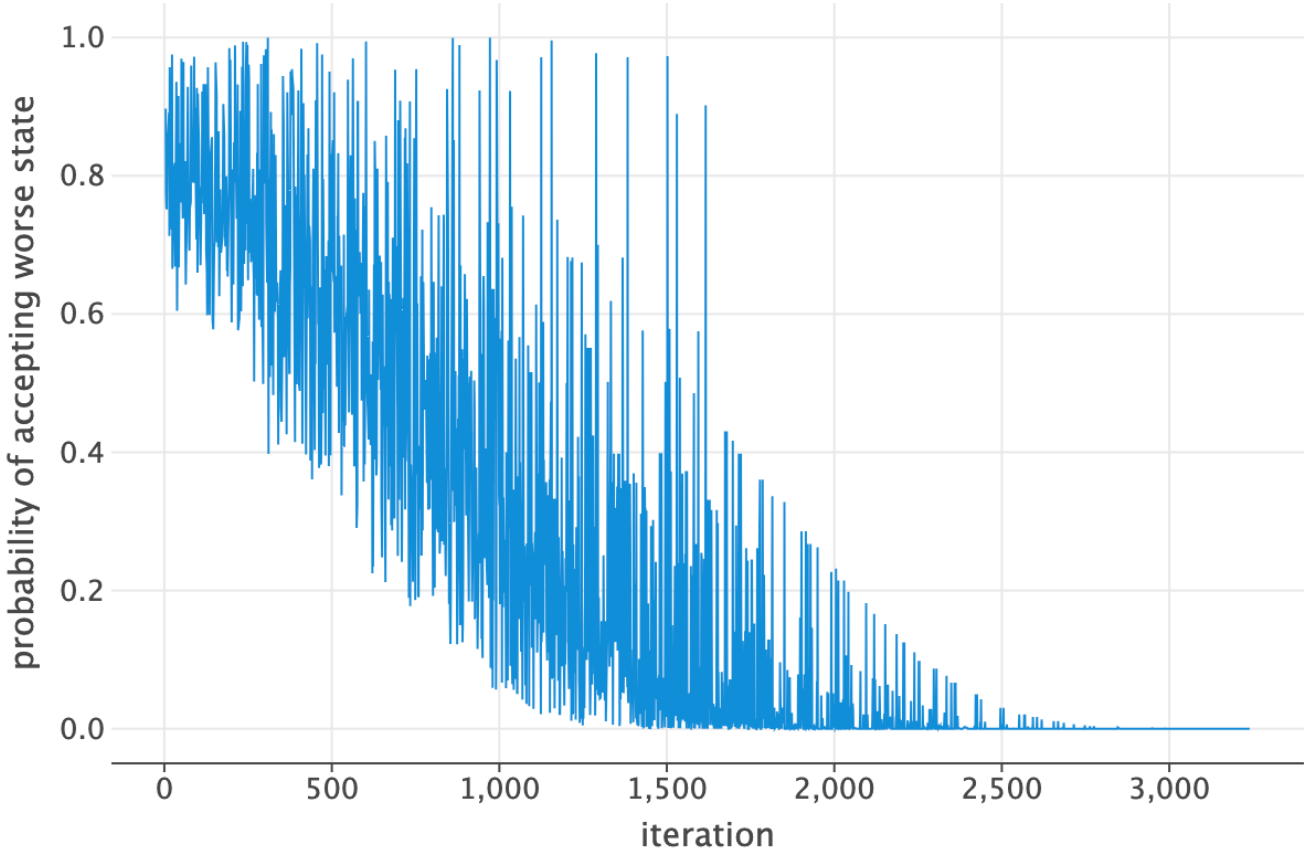
Toto naivní nastavení ovšam na první pohled nebylo moc kvalitní, jelikož preferovalo hodnocení kde byly všechny proměnné nastaveny na true a násobící zlomek nepřevážil jednotlivé hodnoty vah.

avrg. runs	satisfied	satisfiedWithOptimum	avrg. heuristic value	avrg. satisfied clausules	selected variables
1590	0/1000	0/1000	1015	64.0	20.0

- heuristika nepenalizovala nesplněné klauzule dostatečně

Průběh heurické hodnoty a pravděpodobnosit přijmout zhoršující stav vypadají poměrně dobře


heuristicValue
1,000
900
800
700
600
500
400
300
200
0
500
1,000
1,500
2,000
2,500
3,000
iteration


probability of accepting worse state
1.0
0.8
0.6
0.4
0.2
0.0
0
500
1,000
1,500
2,000
2,500
3,000
iteration

Krok 2 (Úprava heuristické hodnoty)

1. (sumOfSuccessClausules / AllClausules) je vždy v rozmezí 0 do 1 závostlost je ovšem lineární na tomto intervalu podle počtu splněných klauzulí. Toto lze změnit umocněním tooto zlomku na exponent > 1

```
heuristicValue = sumOfActiveWeights * (sumOfSuccessClausules / AllClausules) ^ E
```

sumOfSuccessClausules / AllClausules je vždy hodnota v rozsahu <0, 1> proto je možné volit E i pro hodnoty s desetinou čárkou (např 2.23)

Penalizace křivkou mi přišla výhodnější pro tuto heuristickou hodnotu než skokový přechod pomocí penalizace neúspěšných běhů násoběním konstantou. Tato metoda může připomínat například gama korekci. https://www.cambridgeincolour.com/tutorials/gamma-correction.htm

pro exponent testueme hodnoty 2 2.5 3 3.5 4 4.5 5
na instancích wuf20-71-M/20-71-01 až wuf20-71-M/20-71-05

počet opakování: 500
formát dat: (satisfied) — (satisfiedWithOptimum)

File	2	2.5	3	3.5	4	4.5	5
20-71-01	0 — 0	500 — 500	500 — 500	500 — 490	500 — 474	500 — 450	500 — 444
20-71-02	500 — 500	500 — 500	500 — 500	500 — 498	500 — 495	500 — 495	500 — 476
20-71-03	0 — 0	500 — 500	500 — 500	500 — 490	500 — 465	500 — 436	500 — 426
20-71-04	0 — 0	495 — 493	496 – 494	493 – 491	500 — 491	499 — 483	500 — 473
20-71-05	500 — 500	500 — 500	500 — 500	500 — 500	500 — 500	500 — 500	500 — 497

- z naměřených dat lze vidět že optimální exponent bude někde mezi 2.5 a 3

měření hodnot exponentu 2.3 až 3.0 po 0.05 krocích s 200 opakováním na instancích wuf20-71-M/20-71-01 až wuf20-71-M/20-71-050 úspěšné běhy a optimální běhy byly napříč těmito instancemi sčítány

(max 10_000)

Exponent	satisfied	optimums
2.3	8877	8751
2.35	9077	8950
2.4	9061	8928
2.45	9072	8941
2.5	9064	8927
2.55	9069	8907
2.6	9085	8916
2.65	9265	9090
2.7	9272	9081
2.75	9283	9095
2.8	9504	9276
2.85	9522	9299
2.9	9515	9301
2.95	9521	9292
3.0	9515	9260

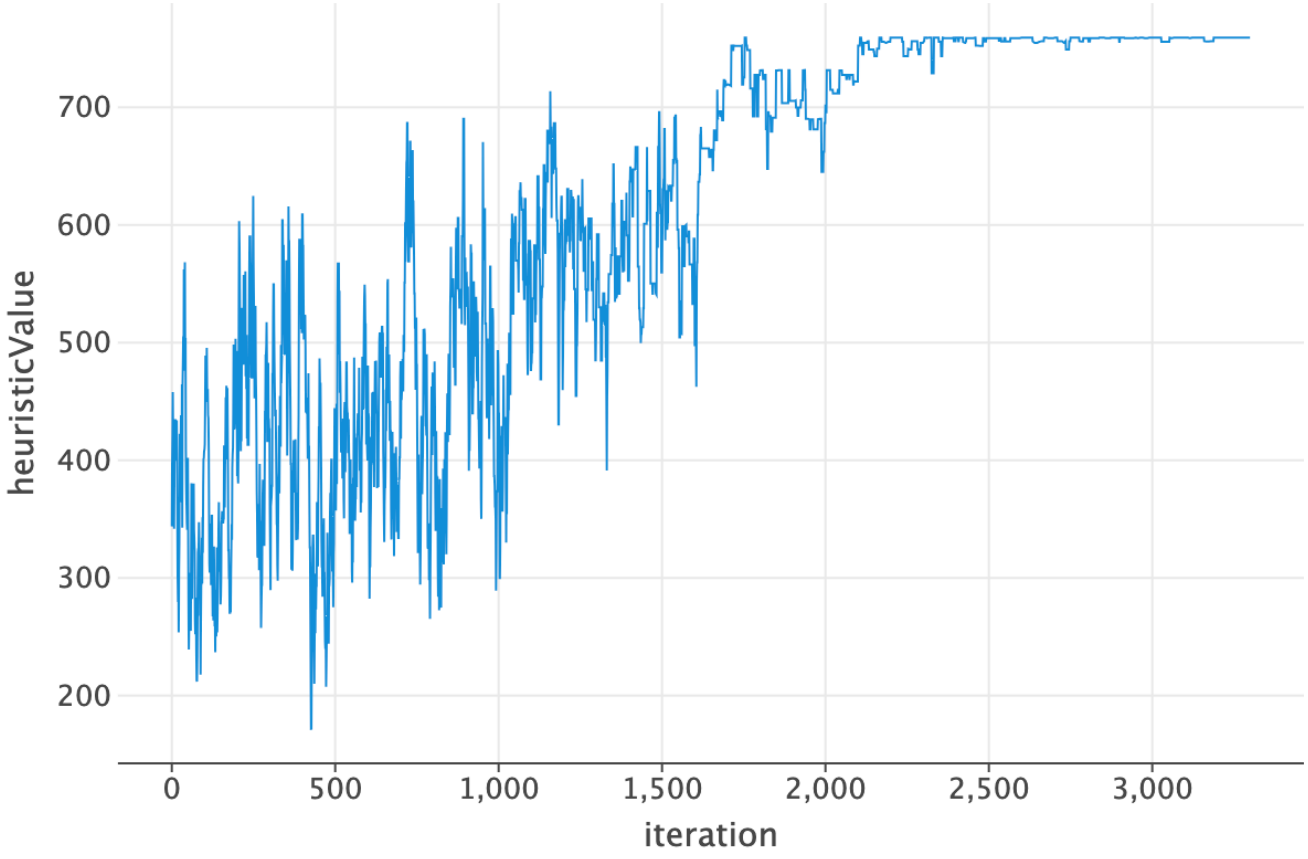
- zde je vidět že optimum exponentu je přibližně okolo 2.9
- tento exponent udává poměr kvality zda je formule splněna a ohodnocení váhami.

Toto nastavení dobře fungovalo pro všechny instance 21-71-M/N tedy lehčí z pohledu sat tak bez zavádějcích vah, ale pro data 21-91 a pro 21-71-Q/R to většinou neskončilo ve stevu který měl všechny klauzule soplněné.

Exerimentama s exponentem nad jednotivými instancemi bylo zjištěno že každá instance má svůj “sweet spot” pro hodnotu exponentu. Tato hodnota ovšem je pro každou instanci různá.

- 20.71-Q-05 (E = 2.9)

E	avrg. runs	satisfied	satisfiedWithOptimum	avrg. heuristic value	avrg. satisfied clausules	selected variables
3	3300	0/1000	0/1000	751.37	64.0	17.0
5	3300	0/1000	0/1000	719.58	66.4	15.52
10	3300	332/1000	314/1000	673.11	67.52	15.01
15	3300	895/1000	646/1000	642.71	68.83	14.69
20	3300	987/1000	590/1000	623.91	68.89	14.48
25	3300	999/1000	506/1000	610.65	69.22	14.32

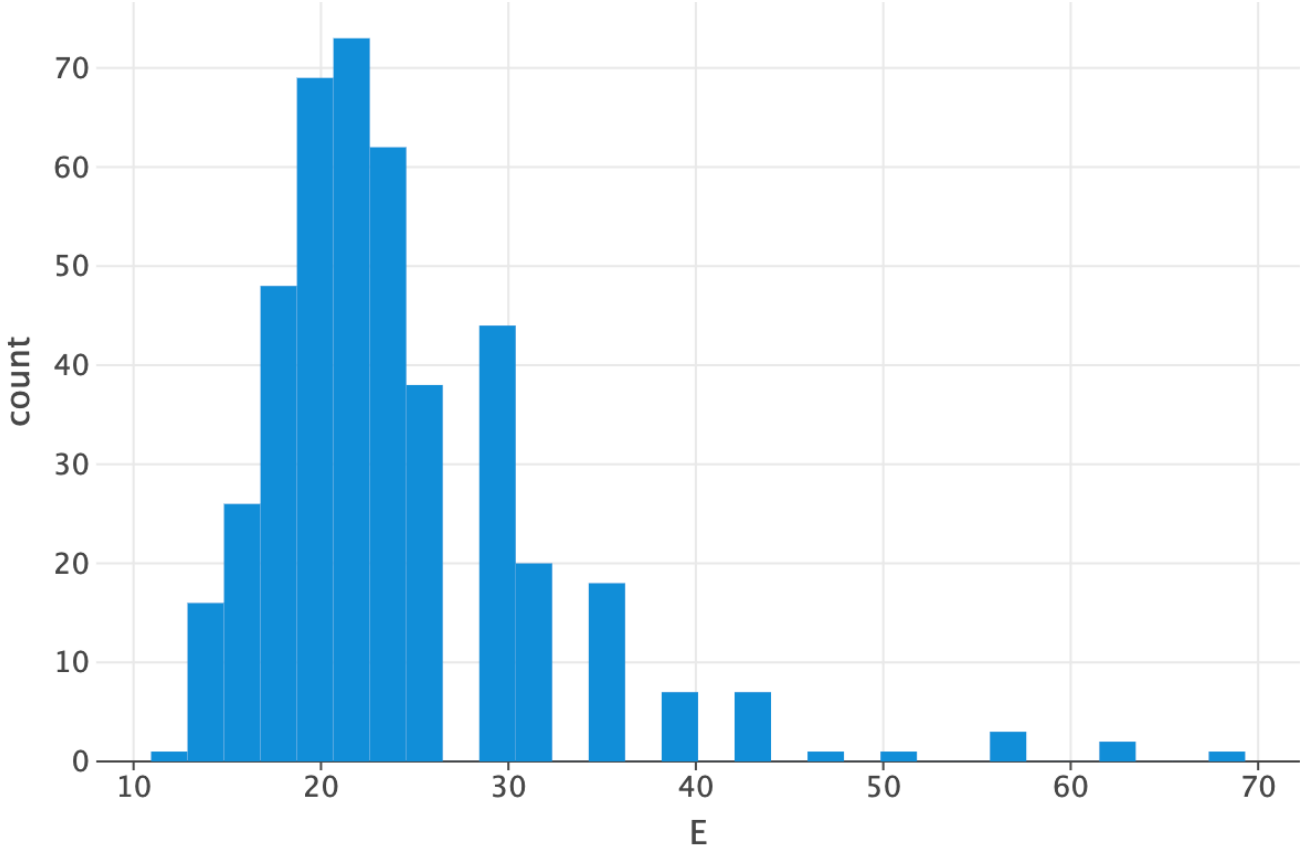


P1E: Přesná hodnota exponentu se nasavuje přímo při běhu algoritmu vždy když se zjistí že soused má všechny klauzuje splněné a zároveň má nižší hodnotu heuristické hodntoy tak se hodnota E vynásobí koeficientem 1.1 (při změně exponentu se teplota vrátí na původní jelikož všechny heuristické hodnoty v prvním běhu budou jiné)

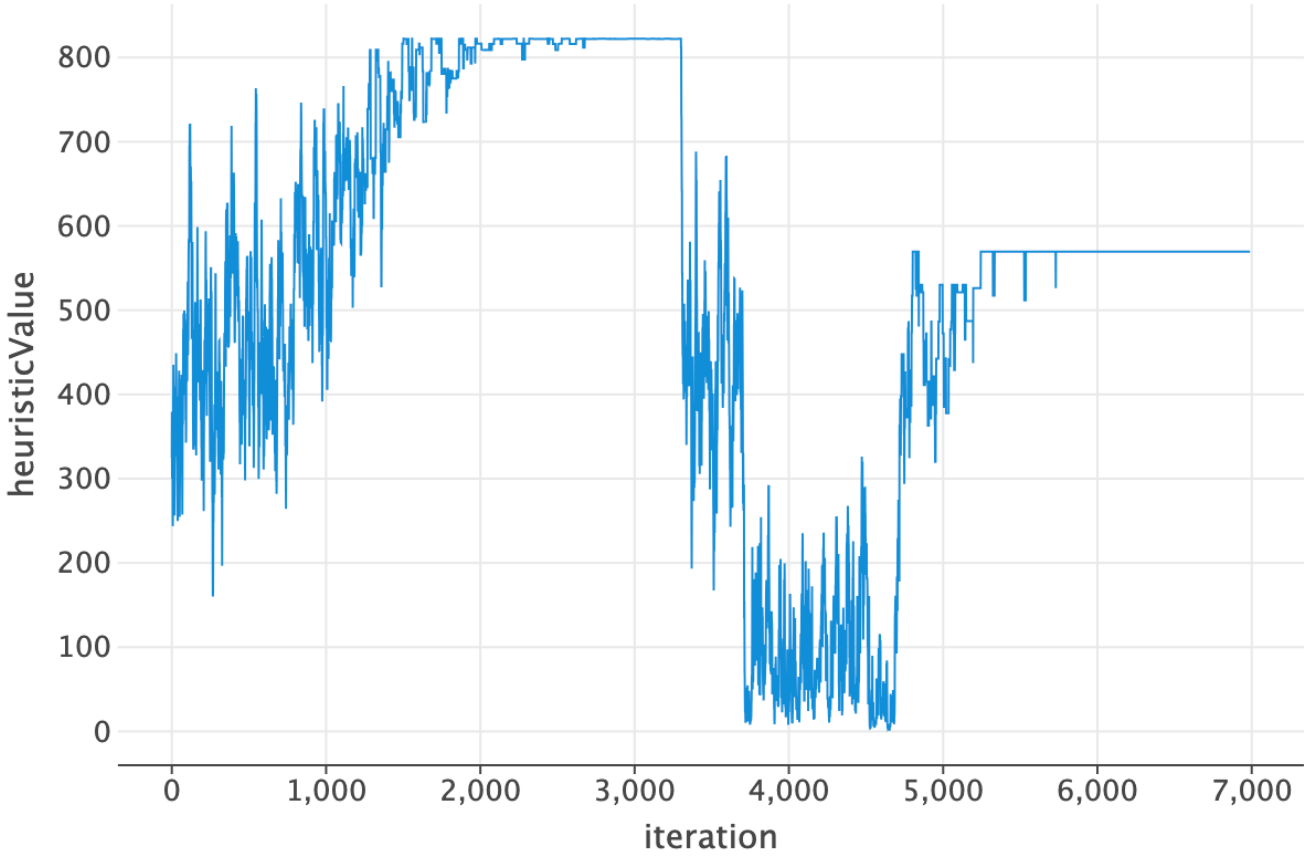
P2E: A pokdu algoritmus skončí a výsledný stav neobsahuje všechny clauzule beh se opakuje s expojnentem zvětšeným an dvojnásobek.

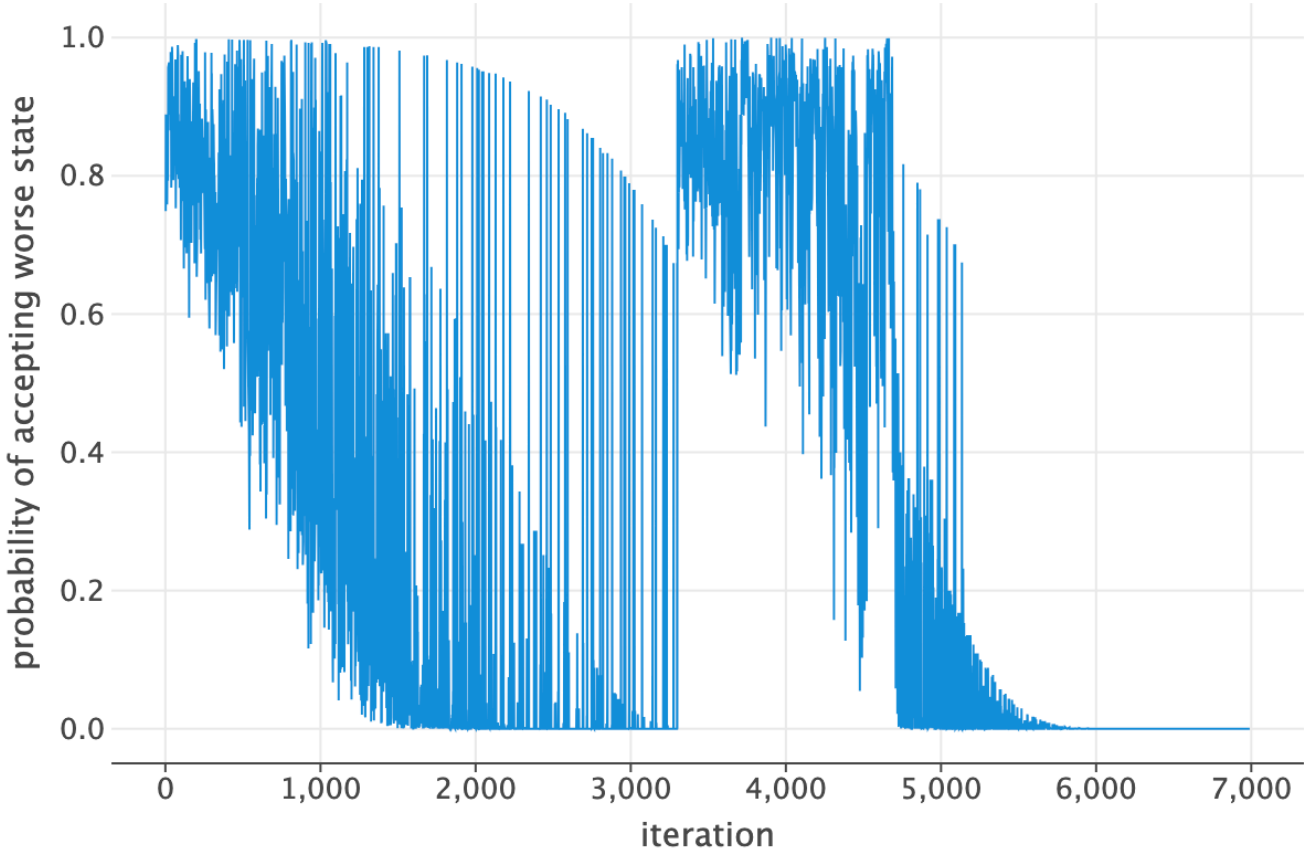
E	avrg. runs	satisfied	satisfiedWithOptimum	avrg. heuristic value	avrg. satisfied clausules	selected variables
Adaptive	13006	1000/1000	575/1000	751.37	64.0	17.0

- Histogram dosažených hodnot E pro instanci 21-71-Q-05



- Ukázka resetu heuristiky při nalezení konfliktu





testování nad různými instancemi

- 20-71-M (všechny instance každá 100 běhů)
  - Pravděpodobnost nalezení splněné formule: 100%

    - Pravděpodobnost nalezení optima: 97.14%
- 20-71-N (všechny instance každá 100 běhů)
    - Pravděpodobnost nalezení splněné formule: 100%
    - Pravděpodobnost nalezení optima: 97.5%
- 20-71-Q (všechny instance každá 100 běhů)
    - Pravděpodobnost nalezení splněné formule: 100%
    - Pravděpodobnost nalezení optima: 58.95%

pro lehčí instance M, N aktuální nastavení fungovalo celkem dobře ale pro instance Q, R hodnota **E** část byla příliš vysoká

ukázalo se že odstranění podmínky pro zvyšování exponentu při nalezení konfliktu P1E a úprava P2E na nižší hodnotu fungovalo lépe

P2E = 1.2, ochlazování = 0.95
- 20-71-Q
    - Pravděpodobnost nalezení splněné formule: 100%
    - Pravděpodobnost nalezení optima: 83.6%
- 20-91-M
    - Pravděpodobnost nalezení splněné formule: 100%
    - Pravděpodobnost nalezení optima: 99.11%
- 20-91-N
    - Pravděpodobnost nalezení splněné formule: 100%
    - Pravděpodobnost nalezení optima: 98.9%
    - avrg. Steps: 8035.96
- 20-91-Q
    - Pravděpodobnost nalezení splněné formule: 81%
    - Pravděpodobnost nalezení optima: 53.97%

Krok3 (Optimalizace koeficientu chlazení a růstu exponentu)

koeficient chlatzení budeme ozančovat C

Faktorový návrh byl testován na instanci 20-71-Q prvních 100 instancí každá 40 opakování (celkem 4000 pro jedno na nastavení)

- data jsou ve formátu: **úspěšnost (relativní_chyba) / počet_kroků**

P2E\C	0.90	0.92	0.95	0.97	0.98	0.99
1.1	74.80% (0.011) 20521.43	77.93% (0.008) 26059.52	83.23% (0.006) 42571.0	90.18% (0.003) 72421.26	93.68% (0.002) 110134.16	97.28% (>0.001) 223040.31
1.2	72.60% (0.013)	76.45% (0.010) 15440.91	83.58% (0.006) 25082.85	88.93% (0.004) 42398.12	92.68% (0.003) 63996.81	96.85% (>0.001) 129073.65
1.3	70.18% (0.015) 9386.57	74.48% (0.012) 11854.32	80.90% (0.007) 19248.33	88.08% (0.004) 32339.52	90.90% (0.003) 48902.14	96.15% (0.001) 98382.50
1.4	69.70% (0.016) 7880.79	72.55 (0.013) 9880.70	80.90% (0.008) 15987.06	87.23% (0.005) 26785.18	91.92% (0.003) 35982.83	96.20% (0.001) 81075.17
1.5	65.23% (0.018) 6993.70	70.95% (0.015) 8786.83	77.35% (0.010) 14195.96	85.73% (0.005) 23779.27	89.80% (0.004) 35982.83	95.10% (0.002) 72519.05
2.0	62.48% (0.024) 5192.70	66.07% (0.019) 6498.74	75.07% (0.012) 10471.89	82.13% (0.007) 17523.37	88.68% (0.004) 26366.63	94.30% (0.002) 52805.71

- Zvoleno bylo P2E = 1.4 a C = 98 - úspěšnos 91% s 0.3% reativní chybou a průměrným počtek kroků ~36000

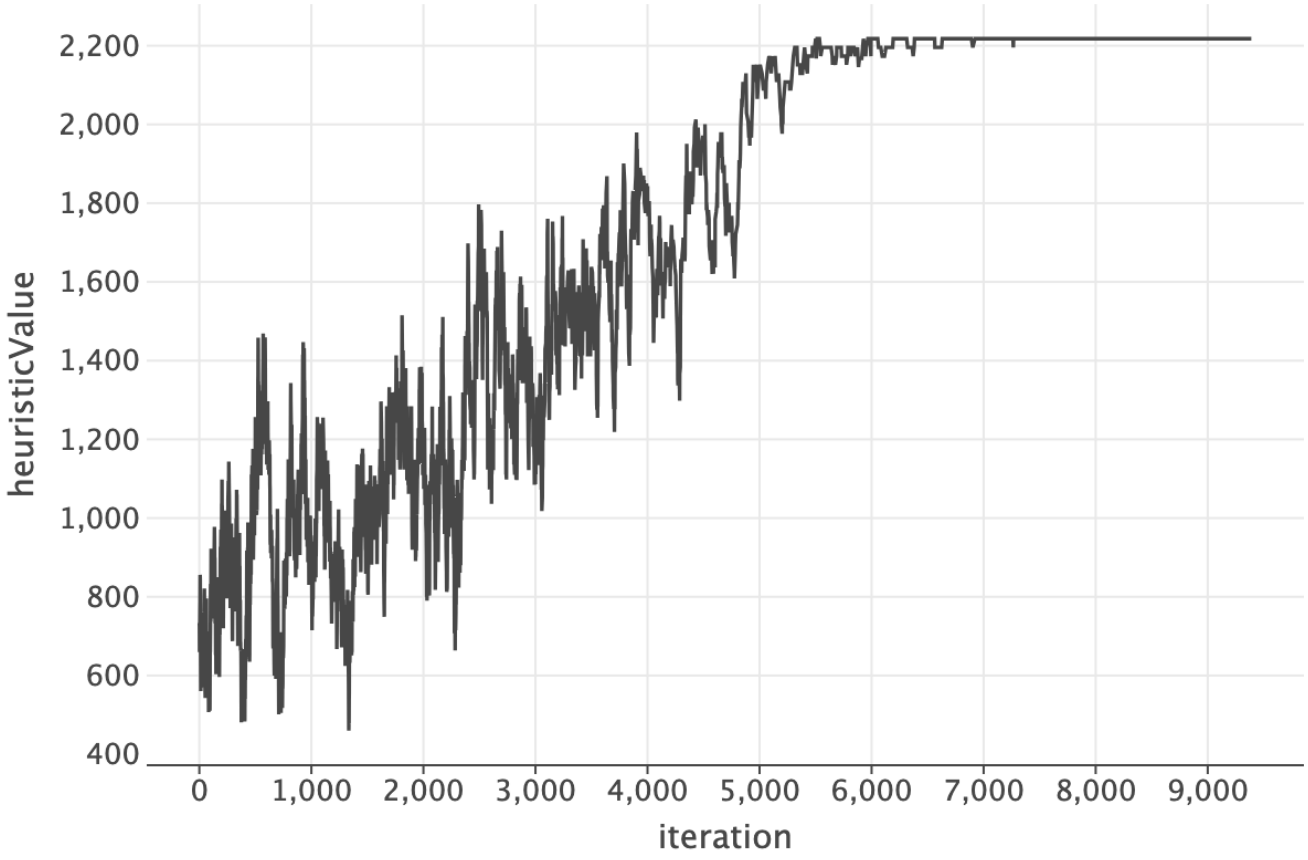
Finální úprava

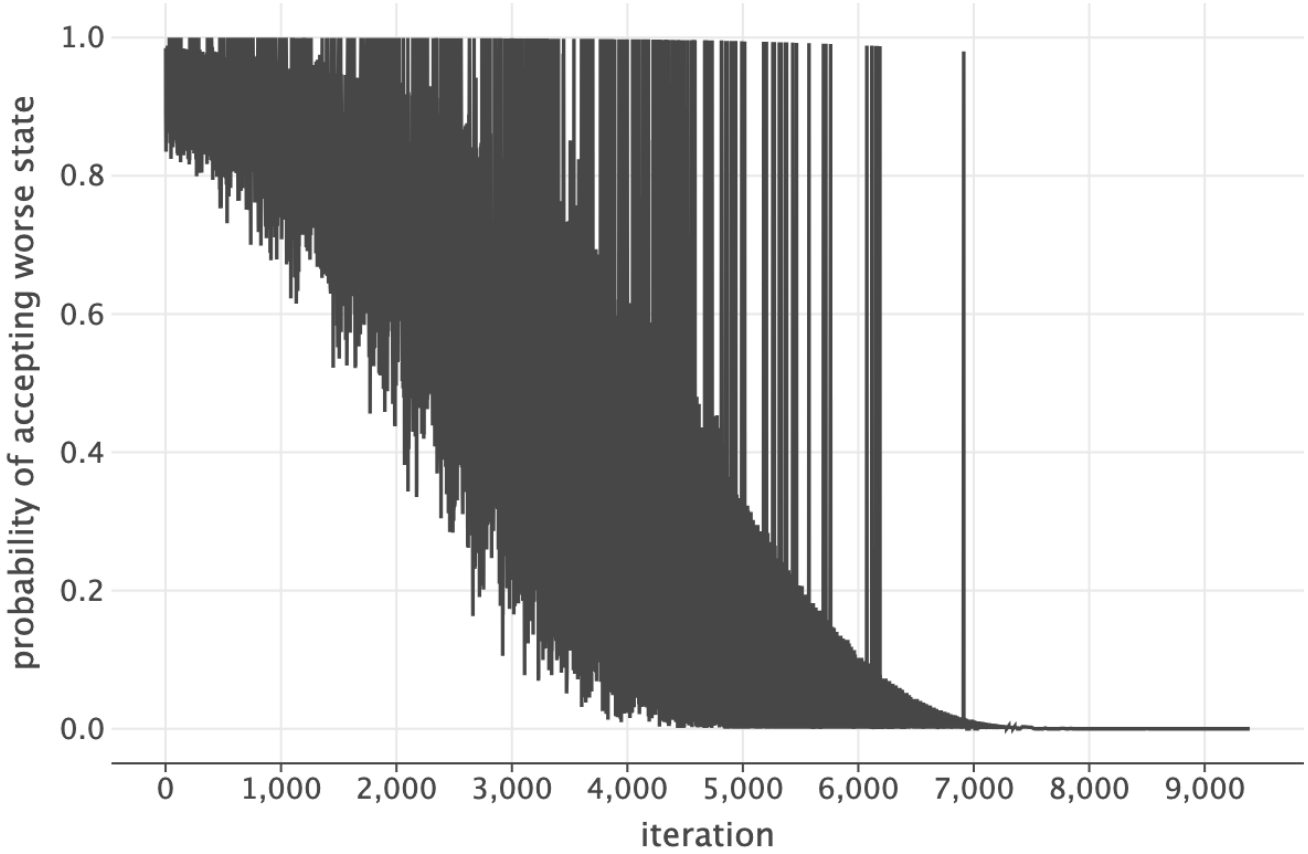
- **Počáteční stav:** náhodný (náhodně ohodnoceny jednotlivé proměnné)
- **Počáteční teplota:** součet normalizovaných vah **všechny váhy byly namapovány do intervalu (0, 100)**
- **isFrozen()** - pokud je teplota nižší než 1
- **quilibrium()** - je konstanta 30 (30 inner cycles)
- **A.isBetterThan(B)** a **P()** pracují s heuristickou hodnotou zvolenou na základě následujícího vzorečku: (optimalizováno v kroku 2 a 3[hodnota E])

Při testování se našla ještě chyba když E bylo neustále snižování u těžkých instancí s 50 a více proměnnými pak hodnota heuristiky se tratila v Float64 a neříkala prakticky nic než splněno a nesplněno a algoritmus se zacyklil, toto jsem nakonec upravil tak aby při E vštším než 10^5 se použije jiná heuristická hodnota:

```
heuristicValue =
          if(E < 10^5) sumOfActiveWeights * (sumOfSuccessClausules / AllClausules) ^
          else if(satisfied) sumOfActiveWeights + sumOfAllWeights
          else sumOfActiveWeights
```

- **coolDown()** - aktuální teplota vynásobena koeficientem 0.98 (zvoleno v kroku 3)
- **neighbour(s)** - prohodí hodnotu jedné (náhodné) proměnné v aktulním stavu
- Program se opakuje pokud skončí ve stavu který nemá splněny všechny clauzule hodnota E je rozšířena o 1.4 násobek (Krok 3)
- Finální průběh heuristiky nad 50-218-M-01


heuristicValue
2,200
2,000
1,800
1,600
1,400
1,200
1,000
800
600
400
0
1,000
2,000
3,000
4,000
5,000
6,000
7,000
8,000
9,000
iteration


probability of accepting worse state
1.0
0.8
0.6
0.4
0.2
0.0
0
1,000
2,000
3,000
4,000
5,000
6,000
7,000
8,000
9,000
iteration

Black Box Testing

V rámci blackbox fáze bylo hlavěn bráno důraz na výstupy z algoritmu (poměr splněných k nesplněným v procentech (Satisfied), správnost nalezeného optima v %, a relativní vzdálenost nalezeného optima od škutečného optima) a průměrný počtet kroků k dosažení výsledku

Počet kroků se mi zdál vhodnější než čas běhu jelikož je platformně a programově nezávislý

Všechny instance opakovány 100x pro MN a 10x každou pro QR instanci

instance	satisfied	satisfied optimum	relative error	avrg. Iterations
20-71M	100%	99.98%	8.78*10^(-8)	10205.61
20-71N	100%	99.90%	9.62*10^(-7)	10212.45
20-71Q	100%	91.36%	2.65 * 10^(-4)	40943.92
20-91M	99.99%	90.45%	1.20 * 10^(-4)	15731.88
20-91R	94.8%	68.4%	0.15	165976.74
20-91Q	93.8%	66.2%	0.16	166582.35
36-122M	100%	?	?	11454.72
36-122N	100%	?	?	11440.07
36-157M	99.95%	97.13%	9.04 * 10^(-4)	21880.797
36-157N	99.91%	97.03%	8.49 * 10^(-4)	21880.797
36-157Q	62.60%	12.9%	0.22	271214.46
50-218M	99.7%	96.0%	8.92*10^(-4)	24781.62
50-218N	99.6%	96.2%	7.46*10^(-4)	25485.24
75-325N	96.7%	85.9%	3.7*10^(-3)	43762.74
75-325N	96.39%	87.8%	2.6*10^(-3)	43004.10

Závěr

Cílem bylo vytvoři heuristický algoritmus který bude řešit problém MaxSat. Tento program byl vytvořen pomocí Simulovaného ochlazování a doladěn pomocí pozorování vývoje při volbě parametrů a faktorovým návrhem. Program poměrně efektivně řeší sady MN. Se sadami Q a R měl size naimplementovanou nějakou základní logioku ale na složitějších instancich nefungovala efektivně pro složitější instance kde se scházel fázový přechod s zavaádějícím řešením. Pro instance s 20 proměnnými to ještě fungovalo ale více proměnných (36) to už nezvládalo vůbec přesvědšivě.

Použité technologie

- Programovací jazyk algoritmu: Kotlin
- Grafy: Notebook (Jupyter Notebook s Kotlin programovacím Jazykem) a knihovny lets-plot a data frame
- IDEs: IntelliJ Idea - Kotlin a VSCode - HTML (Report)
- Template pro reporty: https://stackedit.io/