

NI-KOP

Semestrální práce

5. ledna 2021

Řešení problému vážené splnitelnosti
booleovské formule metodou simulovaného
ochlazování

Contents

1	Specifikace úlohy.....	4
2	Návrh heuristiky	4
2.1	Main loop.....	4
2.2	Funkce <i>getStartState</i>	5
2.2.1	Náhodný stav.....	5
2.3	Funkce <i>frozen</i>	5
2.3.1	Konstantní	5
2.3.2	Závislé na poměru odmítnutých stavů.....	5
2.4	Funkce <i>equilibrium</i>	5
2.4.1	Škálování v závislosti na velikosti instance	5
2.5	Funkce <i>cost</i>	5
2.5.1	Soft penalizace	6
2.5.2	Hard penalizace.....	6
2.6	Funkce <i>try</i>	6
2.6.1	Náhodné procházení (random).....	7
2.6.2	Řízené procházení (improve)	7
2.7	Funkce <i>cool</i>	7
2.8	Adaptační mechanismy.....	7
3	Měření výsledků a konfigurace heuristiky.....	8
3.1	Iterace 1 - Počáteční konfigurace heuristiky.....	9
3.1.1	Funkce try	9
3.1.2	Měření na všech instancích a zhodnocení měření	11
3.1.3	Měření na všech instancích podruhé.....	11
3.2	Iterace 2 – minimalizace chybovosti pro těžší sady.....	11
3.2.1	Měření na všech instancích.....	12
3.3	Iterace 3 – minimalizace chybovosti pro větší instance.....	12
3.3.1	Měření na větších instancích, pro která jsou dostupná referenční optima	12
3.3.2	Měření na více sadách.....	15
3.3.3	Otestování adaptačního mechanismu restartování teploty.....	15

3.3.4	Zhodnocení iterace	16
3.4	Iterace 4 – měření doby běhu	17
3.4.1	Instance o velikosti $n = 20$	17
3.4.2	Větší instance.....	20
3.5	Otestování iterativní síly	20
3.6	Finální nastavení heuristiky	22
4	Závěr	23

1 Specifikace úlohy

Je dána Booleova formule F o n proměnných, $N = (x_1, x_2, \dots, x_n)$ v konjunktivní normální formě (CNF). Dále jsou dány celočíselné kladné váhy těchto n proměnných $W = (w_1, w_2, \dots, w_n)$. Nalezněte přiřazení $Y = (y_1, y_2, \dots, y_n)$ proměnných x_1, x_2, \dots, x_n , takové, že $F(Y) = 1$ a součet S vah proměnných nastavených do 1 je maximální. Omezte se na vážený 3-SAT problém.

2 Návrh heuristiky

Pro implementaci byl zvolen jazyk C#, výstupem je Windows Form aplikace, která umožňuje v grafickém prostředí měnit parametry heuristiky a vstupní instance. Aplikace také vykresluje graf informující o průběhu heuristiky.

Čas běhu byl měřen pomocí systémové třídy *Stopwatch*. Při měření času je každá instance spouštěna pětkrát, výsledným časem je průměr těchto pěti běhů. Jelikož některé strategie využívají generátor náhodných čísel, tak je těchto 5 instancí vždy spouštěno se stejným seedem.

Při implementaci jsem především dbal na flexibilitu v rámci parametrizace algoritmu. Parametrizované jsou nejen hodnoty některých konstant, ale i všechny funkce, které řídí výpočet. Konkrétní implementace metod pro řízení výpočtu (`cool()`, `equilibrium()`, ...) je řešiči poskytnuta až za běhu programu, lze tedy řešiči poskytovat různé implementace a různě je kombinovat (použil jsem návrhový vzor [Strategy](#)).

2.1 Main loop

Hlavní loop byl implementován velmi podobně, jako bylo předvěno na přednášce. Kostra hlavního loopu řešiče vypadá následovně:

```
var state = getStartState(...);
var best = state;
var temperature = startingTemperature;
while (!frozen (...)) {
    while (equilibrium(...)) {
        state = try(state);
        if (state.score > best.score && state.isSatisfiable)
            best = state;
    }
    temperature = cool(...);
}
```

2.2 Funkce *getStartState*

Tato funkce se stará o vytvoření počátečního stavu, který je heuristice poskytnut. Byla implementována jedna varianta.

2.2.1 Náhodný stav

Tato varianta vygeneruje náhodný stav (každý literál je náhodně ohodnocen jako *true/false*)

2.3 Funkce *frozen*

Určuje, kdy se má heuristika zastavit, neboli dojde k zamrznutí. Implementovány byly dvě varianty.

2.3.1 Konstantní

Tato varianta oznamí zamrznutí, pokud teplota klesne pod zadanou konstantu.

2.3.2 Závislé na poměru odmítnutých stavů

Tato varianta oznamí zamrznutí, pokud teplota klesne pod zadanou konstantu, nebo bylo během předchozího equilibria odmítnuto $k\%$ stavů, kde k je zadaná konstanta. Má tedy zamezit situacím, kdy heuristika již při nižších teplotách necestuje po stavovém prostoru.

2.4 Funkce *equilibrium*

Tato funkce řídí, kolik pokusů o nalezení nového stavu bude provedeno v rámci jedné teploty.

2.4.1 Škálování v závislosti na velikosti instance

Aby byla heuristika flexibilní při nasazení na instance o různých velikostech, je vhodné nastavený počet kroků equilibria škálovat v závislosti na velikosti instance.

Toto škálování je možné dělat mnoha způsoby, nejpřímočařejší je nastavený počet kroků přímo vynásobit velikostí instance. Délku equilibria lze ještě následně korigovat pomocí parametru, kterým se tato hodnota vynásobí.

2.5 Funkce *cost*

Aby bylo možné stavy jednoduše porovnávat, tak se každému stavu vypočte hodnota *cost*. Tato hodnota se v základu skládá z optimalizační hodnoty, ale je zapotřebí do této hodnoty zakomponovat i informaci o tom, zda je daný stav řešením problému. **Zvolil jsem přístup, kdy optimalizační hodnotu penalizuji za každou nesplněnou klauzuli.** Tento přístup tedy umožňuje heuristiku hrubě směrovat k většímu počtu splněných klauzulí a zároveň zohledňovat optimalizační hodnotu. Tuto penalizaci je dále možné rozdělit na dva způsoby

2.5.1 Soft penalizace

Hodnota $cost$ je vypočtena následovně:

$$cost = optVal - unsatisfied * p$$

Kde $optVal$ je optimalizační hodnota konfigurace, $unsatisfied$ je počet nesplněných klauzulí a p je nastavený parametr pro penalizaci.

Tento způsob penalizace připouští případ, že některé nesplnitelné konfigurace mohou mít větší hodnotu $cost$, než splnitelné konfigurace s menší optimalizační hodnotou. Nevýhoda tohoto přístupu je v tom, že při takto nastavené hodnotě není zaručená konvergence k nějakému řešení. Na druhou stranu je však zajištěn širší průchod stavového prostoru, protože je heuristika ochotnější přijímat konfigurace, které nejsou řešením

2.5.2 Hard penalizace

Pokud je formule splnitelná, pak je hodnota $cost$ rovna optimalizační hodnotě konfigurace. Pokud není splnitelná, pak je hodnota $cost$ vypočtena následovně:

$$cost = optVal - maxOptVal - unsatisfied * p$$

Kde $optVal$ je optimalizační hodnota konfigurace, $maxOptVal$ je maximální optimalizační hodnota instance (tedy případ, kdy jsou všechny literály nastaveny na 1), $unsatisfied$ je počet nesplněných klauzulí a p je nastavený parametr pro penalizaci.

Tento způsob penalizace zaručí, že každá splnitelná konfigurace bude mít větší cost, než libovolná nesplnitelná konfigurace. Bude tím zajištěna konvergence k nějakému řešení, je však omezován průchod stavovým prostorem, protože jsou nesplnitelné konfigurace ve značné nevýhodě oproti splnitelným.

2.6 Funkce *try*

Tato funkce určuje, jakým způsobem bude procházen stavový prostor – na momentální stav aplikuje námi navržený operátor. Takto získaný stav je následně porovnán s předchozím stavem pomocí hodnoty $cost$. Pokud má nový stav hodnotu $cost$ větší, než předchozí stav, pak je vždy přijat.

Pokud má nový stav hodnotu $cost$ menší, pak je tento stav přijat s pravděpodobností

$$e^{\frac{\delta}{T}}$$

Kde T je momentální teplota a $\delta = new.cost() - prev.cost()$.

2.6.1 Náhodné procházení (random)

Velmi jednoduchý způsob, jakým lze procházet stavový prostor je negace ohodnocení náhodného literálu. Bylo by možné takto negovat i více náhodných literálu naráz, ale tyto způsoby jsou velmi podobné ve smyslu, že je výběr nového stavu zcela náhodný.

2.6.2 Řízené procházení (improve)

Bylo by vhodné procházení stavového prostoru nějakým způsobem korigovat, aby bylo preferováno procházení ve směru, který je nadějný k nalezení nějakého řešení. Tento způsob procházení nejprve vygeneruje náhodné číslo v rozmezí intervalu $(0,1)$. Tato hodnota určuje výběr ze 4 možností:

1. Prohození ohodnocení jednoho náhodného literálu
2. Náhodný nový stav
3. Prohození ohodnocení jednoho náhodného literálu, který figuruje v některé nesplněné klauzuli. Pravděpodobnost pro výběr literálu roste s počtem nesplněných klauzulí, ve kterých figuruje.
4. Prohození ohodnocení jednoho náhodného literálu, který je nastaven na hodnotu *false*.

Každá možnost má konfigurovatelnou pravděpodobnost, během které nastane, přičemž dílčí pravděpodobnosti všech možností musí být v součtu 1. Konfigurací těchto dílčích pravděpodobností lze tedy docílit různého stylu procházení stavového prostoru.

První dvě možnosti jsou přítomny, aby bylo zabráněno uvíznutí v lokálním optimu. Je pravděpodobně vhodné je nastavit na relativně nízké hodnoty, aby byla preferována snaha o zlepšení konfigurace.

Třetí možnost slouží k nasměrování na splnitelný stav a čtvrtá možnost slouží ke směrování k optimu. Není samozřejmě nijak garantováno, že obecně tento výběr nových stavů bude opravdu směřovat k nějakému lokálnímu splnitelnému optimu, ale pomáhá alespoň korigovat směr, kterým je ve většině případů vhodné se vydat.

2.7 Funkce *cool*

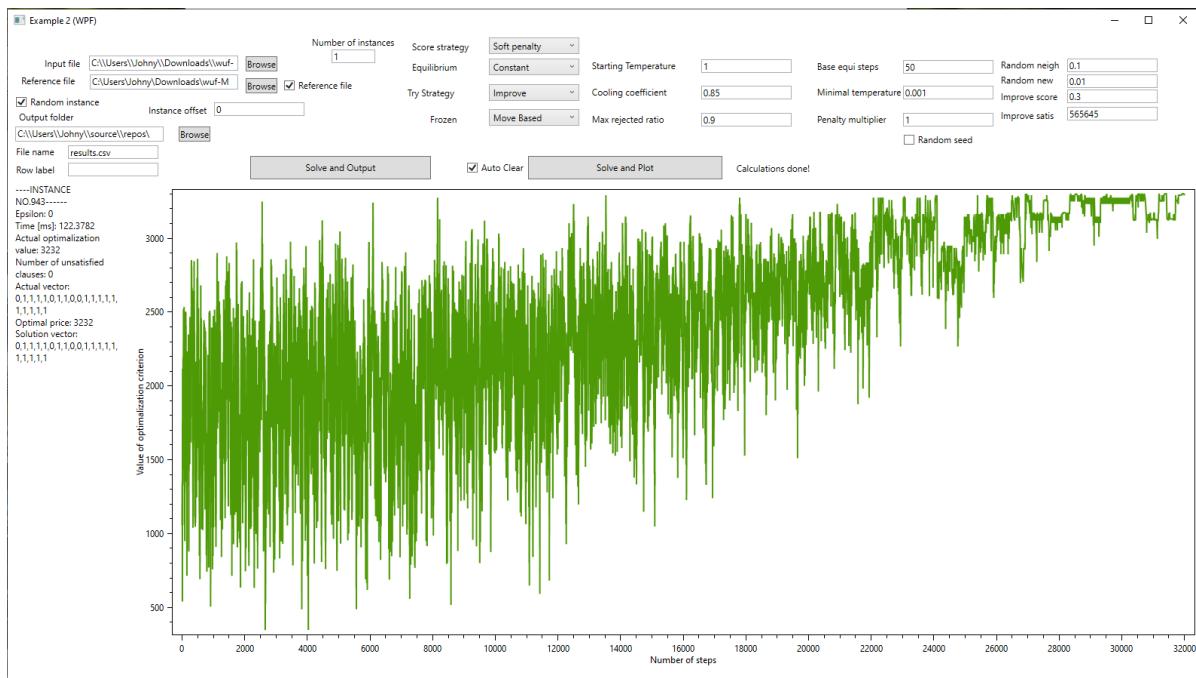
Tato funkce určuje, o kolik bude po běhu *equilibria* snížena teplota. Implementoval jsem pouze variantu, která teplotu vynásobí koeficientem q , $0 < q < 1$.

2.8 Adaptační mechanismy

Pokud se stane, že heuristika zamrzne bez nalezení jediného splnitelného stavu, pak je možné teplotu restartovat a nechat heuristiku běžet dále. Toto lze opakovat vícekrát, s větším možným počtem opakování však hrozí velmi dlouhý běh aplikace v případě, kdy je řešení instance velmi málo (nebo dokonce žádné).

3 Měření výsledků a konfigurace heuristiky

V této kapitole popisuji proces konfigurace heuristiky. Konfiguraci heuristiky jsem realizoval iterativně, přičemž jsem se vždy zaměřil na aspekt heuristiky, který bych chtěl zlepšit (schopnost pracovat na jiné sadě instancí, relativní chybovost, rychlos...). V rámci každé iterace nejprve provádím měření změn na malém množství instancí, přičemž sleduji především graf „cestování“ heuristiky po stavovém prostoru a kvalitu řešení. Pokud pozorují zlepšení výsledků, pak takto nastavené heuristice poskytnu větší množství instancí a sleduji kvalitu řešení a dobu běhu. „White-box“ fázi jsem vždy prováděl pomocí své aplikace (viz obrázek 1), která mi přímo vykreslovala graf průběhu heuristiky pro jednu instanci/průměrná data o průběhu několika instancí.



Obrázek 1 ukázka (osklivého ale funkčního) UI implementovaného řešice

Při nastavování heuristiky se primárně zaměřuji na nízkou chybovost a především nalezení nějakého řešení. Až po dosažení uspokojivé kvality řešení se snažím heuristiku časově optimalizovat tak, aby byla zachována podobná kvalita řešení.

V prvních třech iteracích jsem se nejprve zaměřil především na kvalitu řešení. Až poté jsem se zaměřil na minimalizaci doby běhu, přičemž jsem se snažil udržet kvalitu řešení na podobné úrovni.

Výsledky byly měřeny v OS Windows 10 64bit s procesorem AMD Ryzen 5 2600 six-core (3.4 GHz), 16GB RAM. Při meření jsem na stroji sledoval YouTube, ale intenzivně jsem jej nijak nezatěžoval.

Pro testování jsem použil poskytnuté sady instancí **wuf**. Příklad značení: wuf20-78M je sada, kde každá instance má velikost $n = 20$ a počet klauzulí je roven 78. Každá sada je dále označena písmenem, kde

- Sady M a N obsahují instance, jejichž váhy by měly podporovat nalezení řešení. Obsahují také velké množství řešení. Sada N obsahuje řádově větší váhy literálů, než instance M .
- Sady Q a R obsahují instance, jejichž váhy mohou vytvářet mírně zavádějící úlohy. Sada R obsahuje řádově větší váhy literálů, než instance Q .
- Sada A obsahuje instance, které vytváří zavádějící úlohy a obsahují malé množství řešení. Poměr počtu klauzulí ku počtu literálů je pro tyto instance cca 4.5 což, jak [bylo ukázáno](#), je poměr, při kterém je řešení 3SAT „nejobtížnější“.

3.1 Iterace 1 - Počáteční konfigurace heuristiky

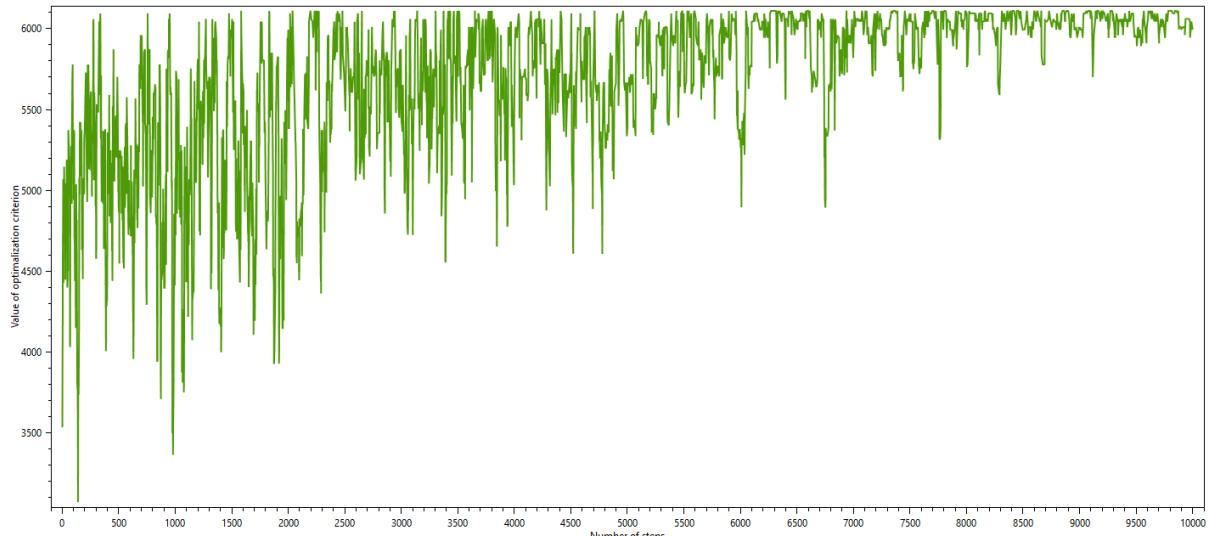
Cílem této iterace bylo dosáhnout nízké relativní chyby na sadě instancí **wuf20M** a **wuf20N**. Počáteční základní parametry heuristiky (teplota, cooling coefficient, ...) jsem nastavil podle zkušeností z předchozí úlohy a pomocí grafického prostředí mé aplikace, kde jsem si mohl s parametry „pohrát“.

3.1.1 Funkce try

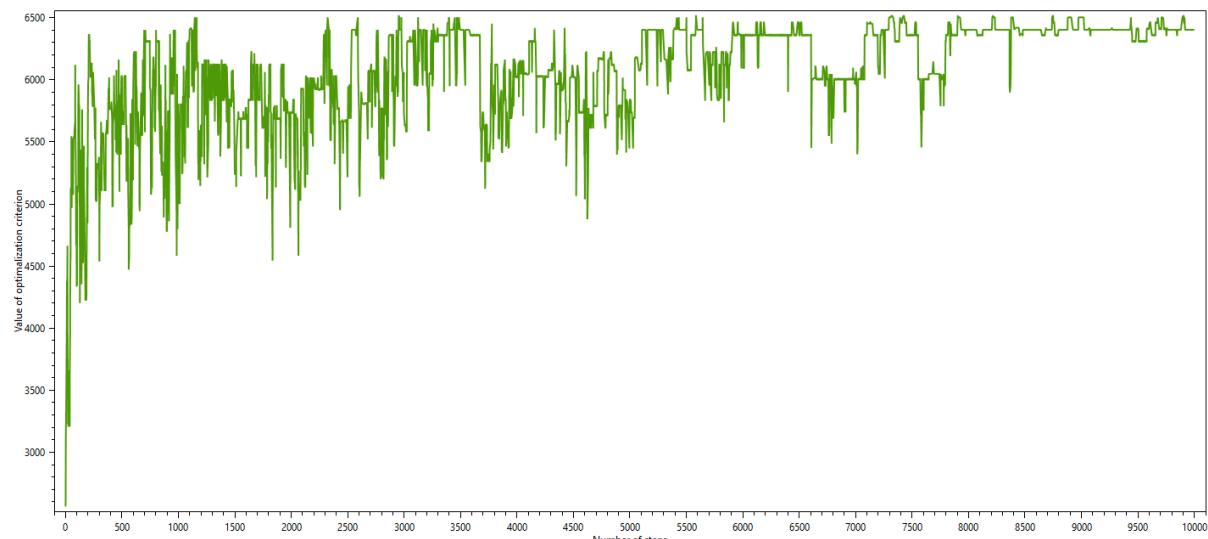
Operátor, který umožňuje pouze přechod do náhodného sousedního stavu (prohození jednoho ohodnocení) se ukázal být dle očekávání nespolehlivý, protože i v rámci malé sady instancí dosahoval velmi rozdílných výsledků (které zároveň nebyly příliš dobré) při opakovaném běhu heuristiky.

Byl tedy nasazen operátor „*improve*“. Nastavení pravděpodobností pro jednotlivé varianty (popsané v 2.6.2) nepřinášelo výrazné změny ve výsledcích, na podrobnější nastavení se tedy případně zaměřím u těžších instancí. **Jedinou výraznou změnou bylo nastavení pravděpodobnosti pro navštívení náhodného souseda na 0 (tím se prakticky zakázala).** V tom případě heuristika občas uvázla v lokálním optimu, tato varianta je tedy velmi důležitá pro správné fungování operátoru.

Při porovnání grafů reprezentujících prohledávání stavového prostoru je patrné, že operátor *improve* (graf 2) při vysoké teplotě také prohledává stavový prostor poměrně „doširoka“, ale držel se mnohem blíže optimální hodnotě, než operátor *random* (graf 1). Zároveň však není patrné, že by operátor *improve* někdy při vysoké teplotě uvízl v lokálním optimu, z tohoto pohledu to tedy vypadá, že operátor pracuje dobře.



Graf 1 – průběh prohledávání stavového prostoru pro operátor *Random*



Graf 2 - průběh prohledávání stavového prostoru pro operátor *Improve*

Na následující tabulce lze pozorovat obrovské zlepšení, kterého bylo dosaženo při nasazení operátoru „improve“.

<i>Wuf20-78M 600 instancí</i>	MIN epsilon	AVG epsilon	MAX epsilon	Počet instancí bez nalezeného řešení
Random	0	0.142	1	67
Improve	0	0.000007	0.0026	0

Tento výsledek lze pokládat za úspěch – pro všechny instance bylo nalezeno řešení, které zároveň bylo optimální, či velmi blízko optima. Ukázalo se zároveň, že operátor *improve* poskytuje mnohem lepší výsledky, než operátor *random*.

3.1.2 Měření na všech instancích a zhodnocení měření

Se získanými parametry bylo tedy provedeno měření na všech instancích velikosti $n = 20$ ze sad M a N. Instancí bylo dohromady 2000.

	MIN epsilon	AVG epsilon	MAX epsilon	Počet instancí bez nalezeného řešení
<i>Wuf20-78M</i>	0	0.000075	0.037	0
<i>Wuf20-78N</i>	0	0.34	1	330

Ale ouha, heuristika pro sadu N dosahuje mnohem horších výsledků, než pro sadu M, přestože by měly být obdobně těžké. **Ukázalo se, že jsem opomenul důležitou adaptaci parametru – škálování teploty na základě velikosti optimalizačních kritérií instance.** Sada N měla oproti sadě M velmi vysoké hodnoty pro váhy ohodnocení, což zapříčinilo, že při výpočtu pravděpodobnosti, zda bude nový stav přijmut jich byla naprostá většina odmítnuta i při „vysoké teplotě“. Teplota, která byla u instance N považována za vysokou, byla u instance M nízká.

Poupravil jsem tedy kód, aby byla teplota škálována na základě součtu vah všech literálů. Stejným způsobem je škálována i minimální teplota.

3.1.3 Měření na všech instancích podruhé

Po této úpravě jsem znovu provedl měření na všech instancích

	MIN epsilon	AVG epsilon	MAX epsilon	Počet instancí bez nalezeného řešení
<i>Wuf20-78M</i>	0	0.000004	0.004	0
<i>Wuf20-78N</i>	0	0	0	0

Měření dopadlo co se týče relativní chybovosti velmi dobře – téměř pro všechny 2000 instancí bylo nalezeno optimum (parametry pro teploty jsem pravděpodobně nastavil až zbytečně vysoko, nicméně bylo stále provedeno průměrně jen 65 000 kroků uvnitř heuristiky).

3.2 Iterace 2 – minimalizace chybovosti pro těžší sady

Cílem této iterace bylo dosáhnout nízké relativní chyby i pro těžší sady instancí **wufQ20**, **wufR20** a **wufA20**. Tyto instance dle poskytnutého popisu mají méně řešení a mohou vytvářet zavádějící úlohu.

Při pilotním měření nebyly odhaleny zásadní nedostatky, heuristika stále vždy našla řešení a dosáhla relativně dobré relativní chyby.

3.2.1 Měření na všech instancích

Oproti první iteraci jsem mírně navýšil koeficient chlazení, aby chlazení probíhalo pomaleji. Ostatní parametry jsem ponechal stejné.

	MIN epsilon	AVG epsilon	MAX epsilon	Počet instancí bez nalezeného řešení
<i>Wuf20-78Q (991 instancí)</i>	0	0.009	0.09	0
<i>Wuf20-78R (991 instancí)</i>	0	0.01	0.15	0
<i>Wuf20-88A (215 instancí)</i>	0	0	0	0
<i>Wuf20-91A (386 instancí)</i>	0	0	0	0

K mému velkému překvapení bylo pro „nejtěžší“ instance vždy nalezeno optimální řešení. Oproti tomu instance ze sady Q a R občas nedosahují optimálního řešení, výsledky jsou však stále poměrně solidní, cíl iterace tedy považuji za splněný.

3.3 Iterace 3 – minimalizace chybovosti pro větší instance

Cílem této iterace bylo otestovat heuristiku pro větší instance a případně ji nakonfigurovat tak, aby řešení pro velké instance dosahovalo dobré kvality.

3.3.1 Měření na větších instancích, pro která jsou dostupná referenční optima

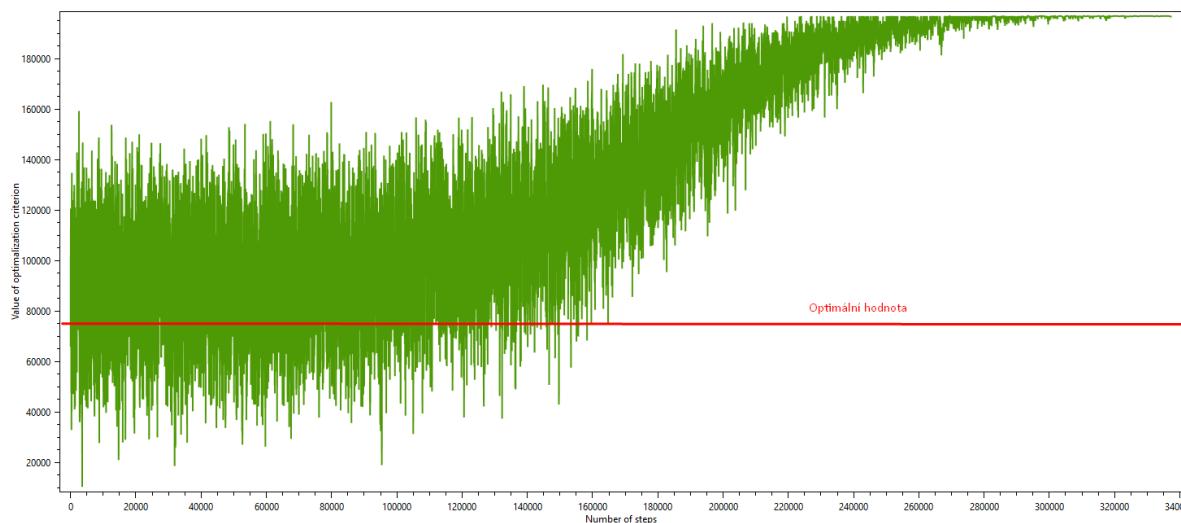
Poskytnutá sada instancí bohužel obsahovala velmi málo referenčních řešení pro instance o velikosti $n = 50$ a žádná referenční řešení pro $n = 75$. Nejprve jsem tedy otestoval několik málo instancí o velikosti 50, pro které bylo referenční řešení dostupné. Testoval jsem pouze instance ze sady N a R.

	MIN epsilon	AVG epsilon	MAX epsilon	Počet instancí bez nalezeného řešení
<i>Wuf50-201-N (34 instancí)</i>	0	0.002	0.02	0
<i>Wuf50-201-R (34 instancí)</i>	0.15	0.45	1	1

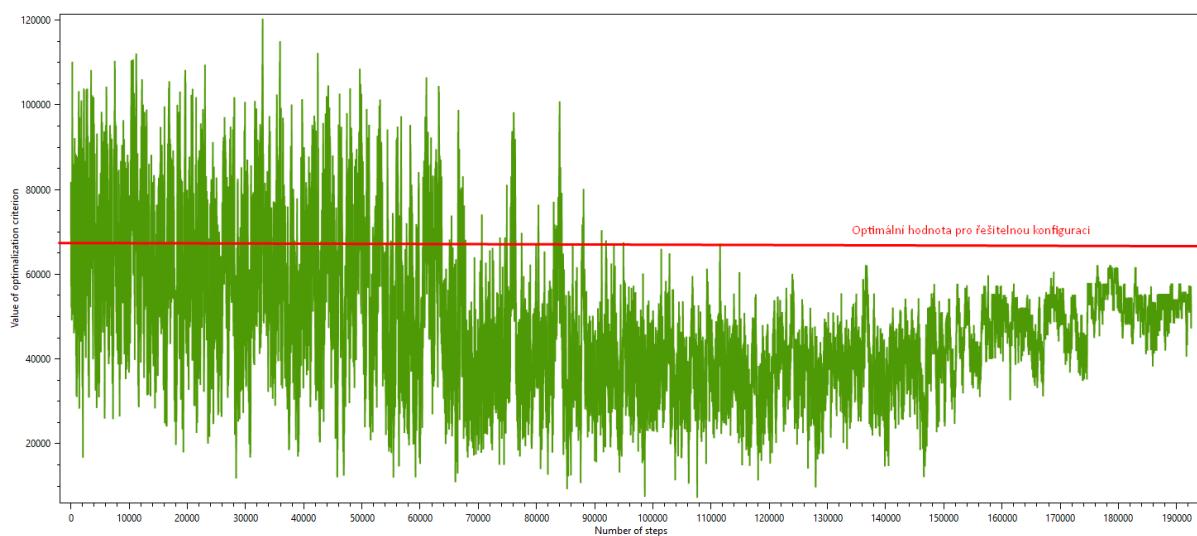
Heuristika si velmi dobře poradila s lehčími instancemi ze sady N. Sada R už představovala větší problém, pro jednu instanci dokonce nebylo nalezeno žádné řešení.

Při analýze jednotlivých instancí jsem zjistil, že **heuristika častokrát fázi intenzifikace strávila v konfiguracích s velmi vysokým ohodnocením**, tyto konfigurace však byly velmi daleko od optimálního řešení (optimální řešení mělo např. opt. hodnotu 70 000, zatímco opt. hodnoty intenzifikovaných konfigurací se pohybovaly kolem 180 000) – viz graf 3.

Příčinou tohoto chování bylo opět opomenutí škálování parametru na základě váhy ohodnocení, tentokrát v případě penalizace nesplnitelných konfigurací. Tyto instance obsahují konfigurace s velmi vysokými optimalizačními hodnotami, tato hodnota tedy „přebila“ penalizaci, která byla udělována za počet nesplněných klauzulí, nedocházelo tedy ke konvergenci k řešení. Tento parametr tedy nyní škáluje i podle součtu vah všech literálů. Na grafu 4 lze vidět, že po naštákování penalizace již fáze intenzifikace probíhá relativně blízko k optimálnímu řešení.



Graf 3 – nežádoucí chování heuristiky, kdy fáze intenzifikace je trávena ve stavech, které jsou velmi daleko od optimálního řešení

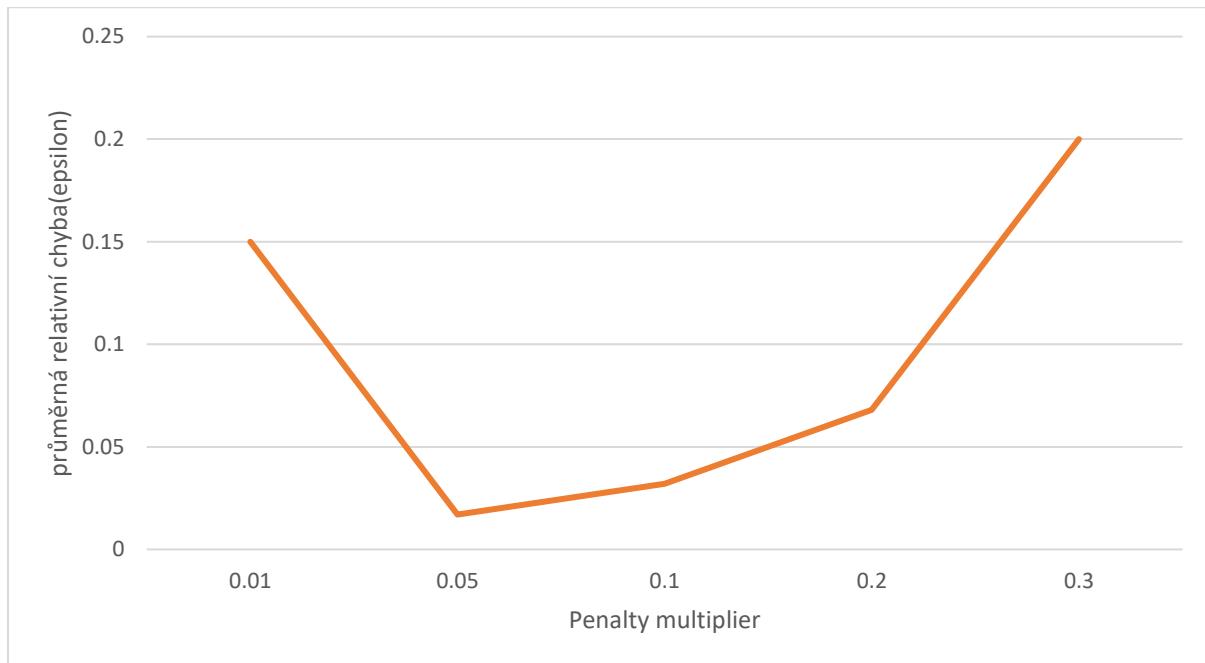


Graf 4 – chování heuristiky po naštákování penalizace, intenzifikace probíhá blízko optimálnímu řešení

Do jaké míry je konfigurace penalizována za nesplněné klauzule je stále řízeno parametrem, který nyní procentualně určuje, jak velkou částí součtu optimalizačních hodnot bude penalizace škálována. Tento parametr je zapotřebí správně nastavit – pokud bude nastaven příliš nízko, pak nebudou nesplněné klauzule dostatečně penalizovány a problém z dřívějška přetrvá. Pokud bude nastaven příliš vysoko, pak nebude heuristice umožněno i při vysokých teplotách vyváznout z lokálního optima.

Parametr je samozřejmě závislý na několika dalších parametrech (například na počáteční teplotě), ale provedl jsem alespoň rychlé měření na sadě wuf50-201-R, abych přibližně zjistil ideální nastavení pro momentální parametry.

<i>Penalty multiplier</i>	MIN epsilon	AVG epsilon	MAX epsilon	Počet instancí bez nalezeného řešení
0.01	0	0.15	0.52	0
0.05	0	0.017	0.17	0
0.1	0	0.032	0.18	0
0.2	0	0.068	0.234	0
0.3	0	0.2	1	4



Graf 5 – vývoj průměrné relativní chyby v závislosti na hodnotě parametru *penalty multiplier*

Na základě měření se ukázalo, že ideální hodnotou pro dané instance a parametry co se kvality řešení týče je multiplier 0.05.

3.3.2 Měření na více sadách

Při testování instancí, pro která nebylo poskytnuto referenční řešení jsem zkoumal pouze počet splněných kluazulí. Epsilon bylo v tomto případě vypočítáno jako $\frac{počet_nesplněných}{počet_kluazulí}$. Testoval jsem pouze sady M, Q a A.

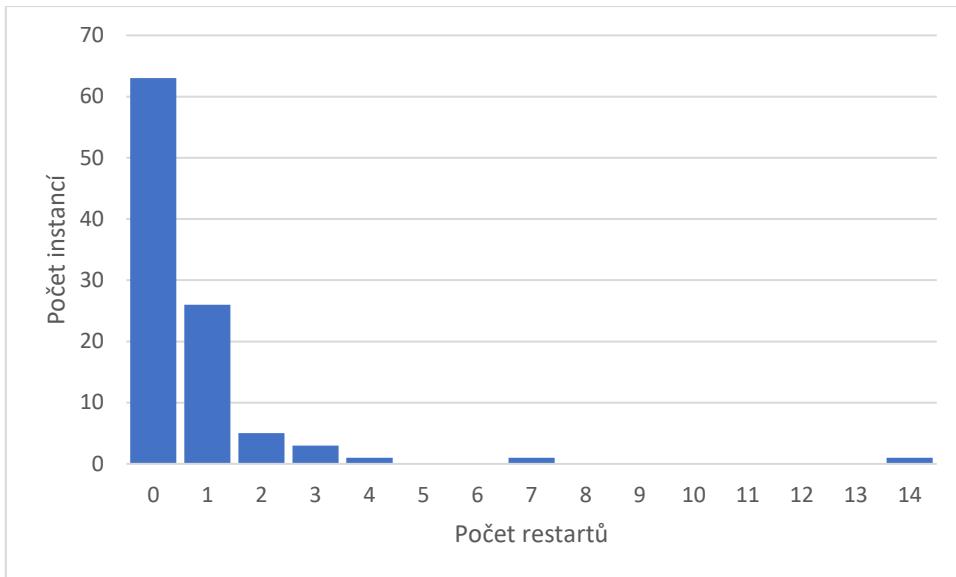
	Počet instancí bez nalezeného řešení
<i>Wuf50-201M</i> (200 instancí)	0
<i>Wuf75-310-M</i> (100 instancí)	0
<i>Wuf50-201-Q</i> (200 instancí)	0
<i>Wuf75-310-Q</i> (100 instancí)	0
<i>Wuf100-430-A</i> (100 instancí)	0

Heuristika nalezla řešení pro všechny testované instance i při poměrně rychlém chlazení (koeficient byl nastaven na 0.9). O celkové kvalitě řešení (jak blízko je optimu) bohužel nic nevíme.

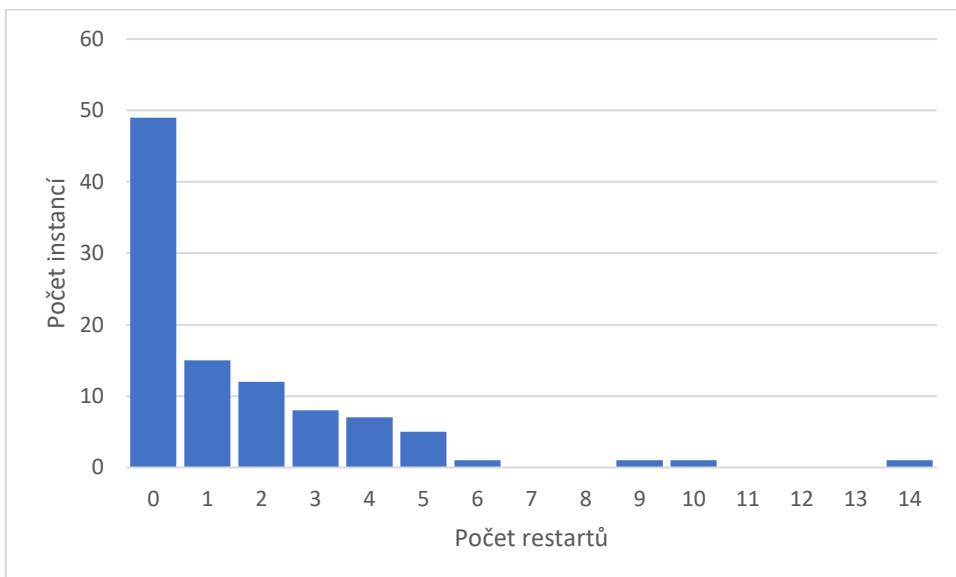
3.3.3 Otestování adaptačního mechanismu restartování teploty

Dále bych chtěl otestovat, zda je adaptační mechanismus ve formě restartování teploty přínosný pro běh heuristiky. Při dosavadní konfiguraci heuristika vracela řešení vždy, i pro největší instance z těžkých sad Q, R a A. Musel jsem tedy velmi výrazně snížit koeficienty chlazení a equilibria. Pro každé řešení jsem vracel informaci o tom, kolikrát byla restartována teplota (maximálně jsem dovolil 15 restartů).

Měření jsem prováděl na těch největších instancích ze sady A (instance mají 100 literálů). Koeficient chlazení jsem nastavil na 0.75, koeficient equilibria na 10 (počet kroků unvitř heuristiky se pohyboval kolem 10 000, takže je to už opravdu extrémní případ). Při měření na 100 instacích nebyl restart k nalezení řešení potřeba pro 63 instancí, pro 26 instaní byl zapotřebí pouze jednou, jak lze vidět z následujícího histogramu:



Pro jednu instanci nebylo nalezené žádné řešení ani po vyčerpání limitu pro počet restartů. Koeficient equilibria jsem následně snížil ještě více a provedl opětovné měření.



Potřebný počet restartů se už začal postupně „přelívat“ dále, nelze tedy říct, že by větší počet restartů byl zcela zbytečný. Jelikož jsem pro získání těchto hodnot musel heuristiku opravdu hodně přiškrtit, tak nepovažuji tento adaptační mechanismus za příliš přínosný a nebudu jej dále zkoumat.

3.3.4 Zhodnocení iterace

Bylo provedeno měření kvality řešení na větších instancích, přičemž byl odhalena další chyba v návrhu heuristiky. Po opravě této chyby heuristika vždy našla řešení, tuto iteraci tedy považuji za dokončenou.

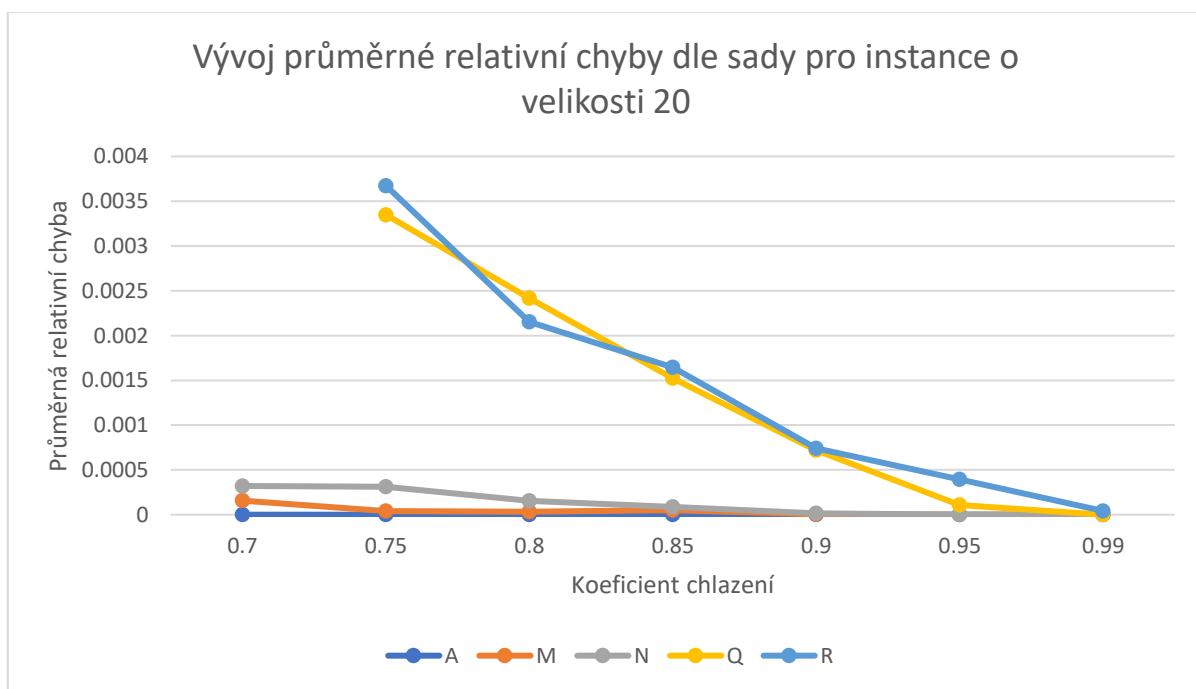
3.4 Iterace 4 – měření doby běhu

Cílem této iterace je otestovat dobu běhu heuristiky pro různé sady instancí a nastavení heristiky a uvést ji do poměru s relativní chybou. **Nastavení heuristiky je z větší části převzato z předchozích iterací.** V této iteraci měním pouze parametry pro rychlosť ochlazování (ochlazovací koeficient) a koeficient pro počet kroků uvnitř equilibria (koeficient kroků). Mohl bych měnit i počáteční teplotu, změna tohoto parametru by však ovlivnila další parametry (např. koeficient penalizace), proto to dělat raději nebudu. Pro připomenutí:

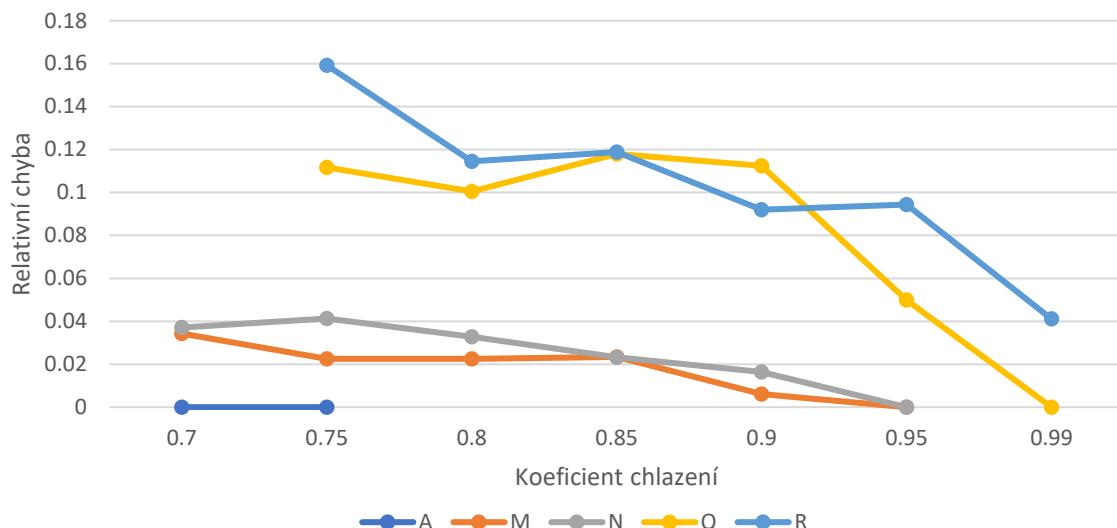
- Po běhu equilibria násobím teplotu ochlazovacím koeficientem, z čehož získám teplotu pro další equilibrium/zamrznu.
- Počet kroků uvnitř každého equilibria je vypočítán jako $koeficient_kroku * n$, kde n je velikost instance.

3.4.1 Instance o velikosti $n = 20$

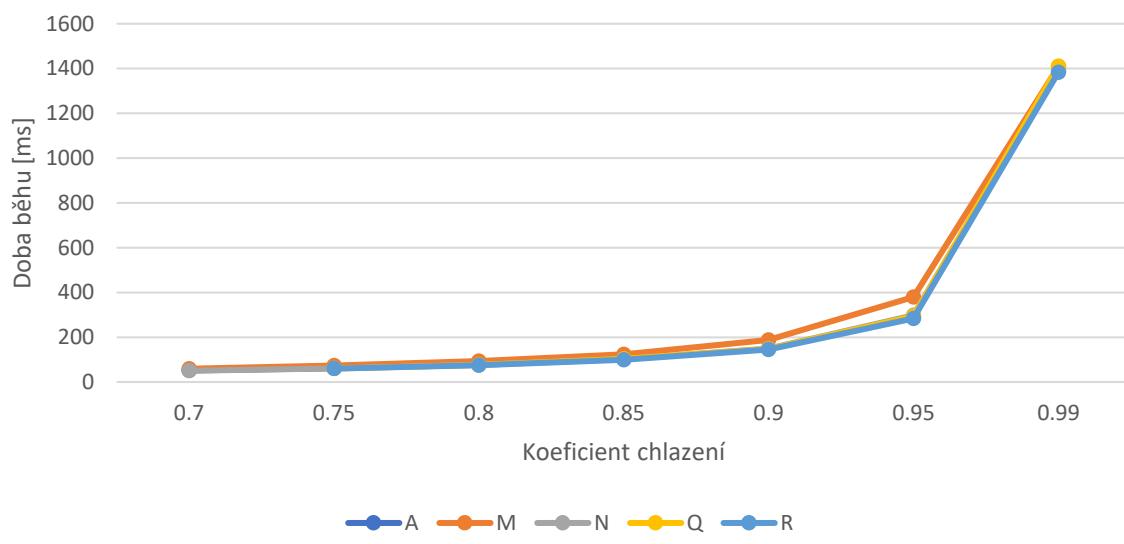
Měření jsem prováděl na všech poskytnutých instancích a sadách o velikosti $n = 20$, každá sada obsahuje 1000 instancí. Z časových důvodů jsem měnil pouze jeden parametr najednou (při tomto měření jsem měnil pouze koeficient chlazení). Dále pro stručnost uvádím pouze průměrné hodnoty (maxima ani minima se při měření příliš nevychylovala daleko od průměru)

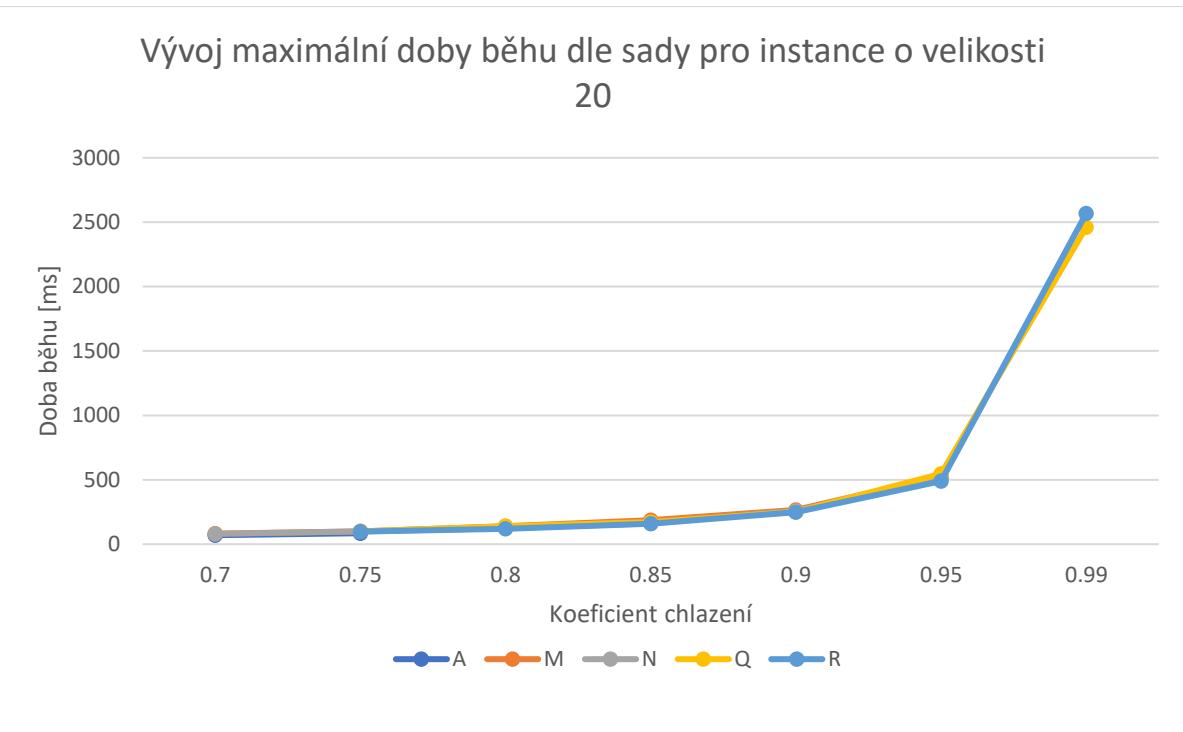


Vývoj maximální relativní chyby dle sady pro instance o velikosti 20



Vývoj průměrné doby běhu dle sady pro instance o velikosti 20





Měření ukázalo, že heuristika pro sady M a N vrací velmi kvalitní výsledky i při rychlém chlazení, kdy se průměrná doba běhu pro jednu instanci pohybuje okolo 50ms (což je cca 16000 celkových iterací skrze equilibrium loop pro představu). Sady Q a R představují o něco větší problém, kdy při rychlejším ochlazování je sice průměrná kvalita poměrně dobrá, maximální chyba však může dosáhnout i hodnoty 0.1. Průměrná doba běhu je pro všechny sady téměř identická.

Velkým překvapením byly výsledky pro sadu A, kde i při velmi rychlém chlazení heuristika stále vždy našla optimální řešení. Pro tuto sadu jsem zafixoval koeficient chlazení na 0.8 a posouval koeficient určující počet kroků uvnitř equilibria.

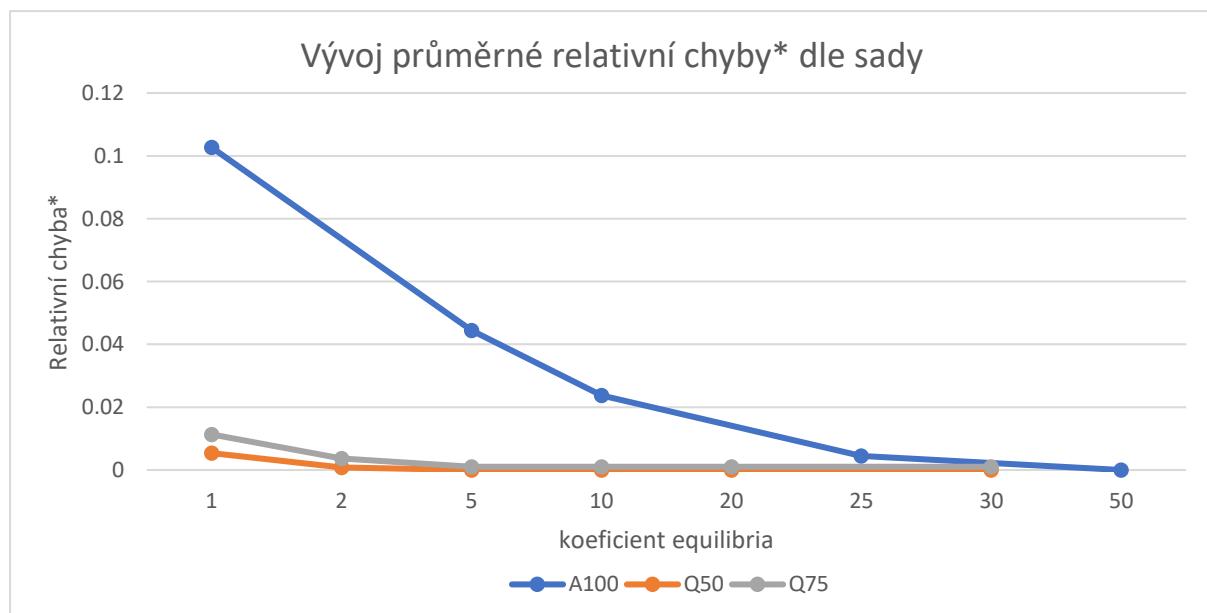
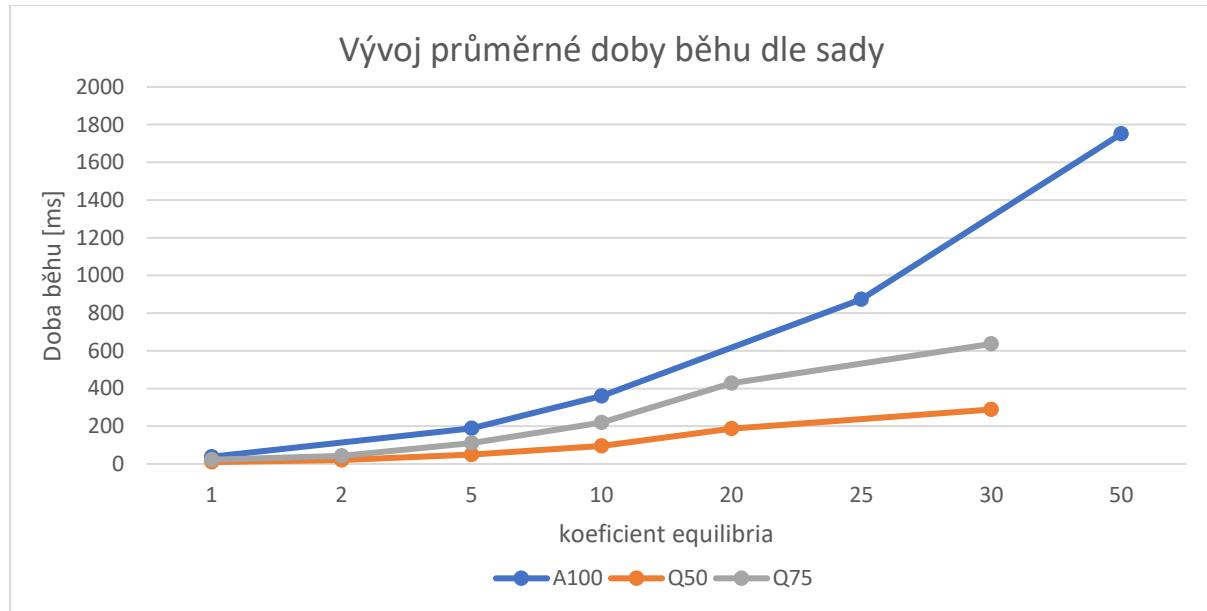
	Epsilon AVG	Epsilon MAX	Doba běhu AVG [ms]
8	0.000360012	0.04109589	11.97307953
10	0.000840243	0.142857149	14.9588707
12	0	0	17.89932465
15	0	0	22.23041349
20	0	0	31.00565349
30	0	0	46.67578093
40	0	0	61.89833953
50	0	0	77.1504214

Při hodnotě koeficientu 10 již algoritmus nenašel vždy optimum, někdy řešení bylo dokonce i poměrně daleko od optima. Dobu běhu však považuji za poměrně slušnou.

3.4.2 Větší instance

U větších instancí bohužel nejsou dostupná referenční řešení (nebo jich je velmi málo), budu proto pouze zkušet, zda bylo nalezeno splnitelné řešení. Zkoumat budu pouze „těžší“ sady instancí – Q a A . V tomto případě jsem zafixoval koeficient chlazení na 0.8 a hýbu s koeficientem equilibria.

*Relativní chyba byla počítána jako poměr $\frac{\text{pocet_nesplnenych_klauzuli}}{\text{pocet_klauzuli}}$



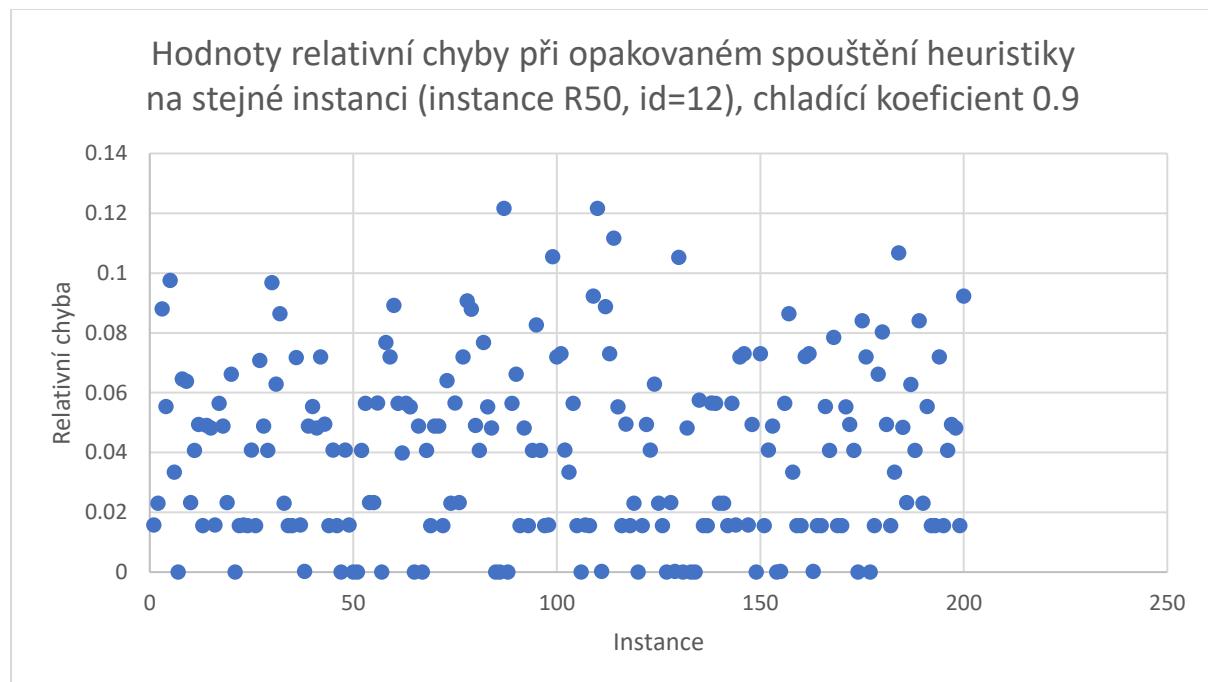
3.5 Otestování iterativní síly

Jelikož má heuristika hojně využívá randomizované techniky (především v operátoru), tak bych v této podkapitole chtěl změřit na několika vybraných instancích, zda

heuristika dosahuje podobné kvality při opakovaném spuštění. Vybral jsem 10 instancí z různých sad (z každé sady 2 instance o velikosti 20 a 50) a každou instanci takto spustil 200krát s různým startovním seedem. Heuristika byla nastavena tak, jak je popisováno v podkapitole 3.6, koeficient chlazení byl nastaven na 0.9, koeficient equilibra na 50.

Jediná instance, pro kterou heuristika nevracela vždy optimální řešení byla instance ze sady R o velikosti 50 (konkrétně se jedná o instanci s id 12). Relativní chyba se zde pohybovala cca v intervalu $<0, 0.12>$.

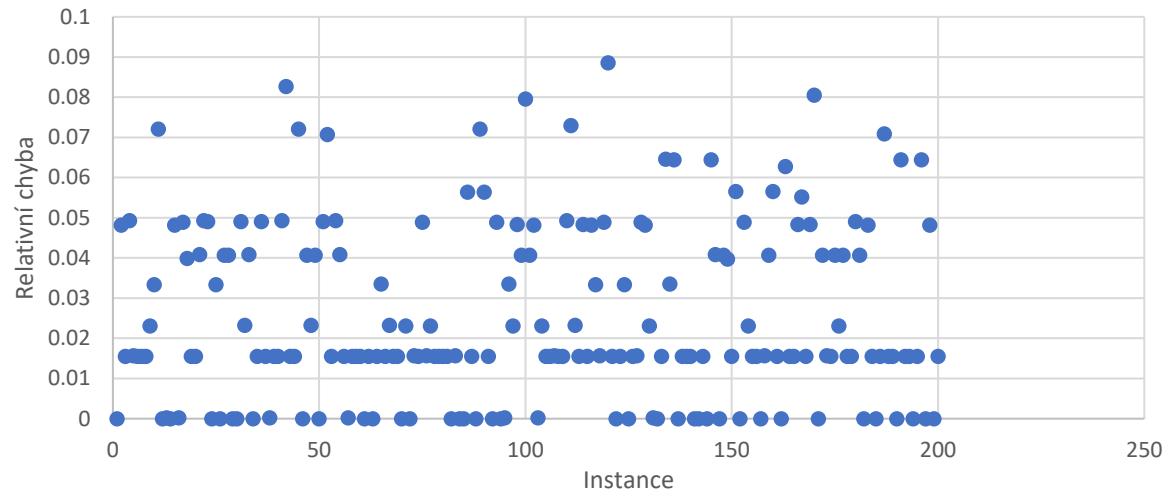
Average of Epsilon	Min of Epsilon	Max of Epsilon
0.040685691	0	0.121555582



Při zvýšení chladícího koeficientu na 0.95 (chlazení je poté pomalejší) se interval zmenšil, heuristika však stále často nepodávala přesné řešení.

Average of Epsilon	Min of Epsilon	Max of Epsilon
0.025360225	0	0.088518843

Hodnoty relativní chyby při opakovaném spouštění heuristiky na stejné instanci (instance R50, id=12), chladící koeficient 0.9



Jedná se však o velkou instanci z obtížné sady, chlazení by pravděpodobně muselo probíhat ještě pomaleji pro přesná řešení. Jelikož heuristika pro všechny ostatní instance vždy vrátila přesná řešení, tak si troufnou heuristiku označit za poměrně stabilní na základě testovaných případů.

3.6 Finální nastavení heuristiky

Na závěr této kapitoly bych chtěl zmínit nastavení heuristiky, ke kterému jsem v rámci měření dospěl. Některé parametry jsem z časových důvodů nenastavoval zcela formálně experimentálně (nastavoval jsem je ve white-box fázi na základě menších testů a intuice). Heuristika navíc pro tato nastavení pracuje poměrně uspokojivě, neměl jsem tedy příliš velkou motivaci parametry dále optimalizovat.

- **Operátor** pro procházení stavového prostoru byl zvolen výše popisovaný *improve*. Tento operátor má 4 parametry, které určují pravděpodobnost pro 4 varianty procházení, parametry byly od 1. iterace nastaveny následovně:
 - **Navštívení náhodného bitového souseda** (bit-flip) – 0.1
 - **Navštívení náhodného stavu** (libovolného) – 0.01
 - **Vylepšení skóre** (nastavení náhodného literálu na true) – 0.1
 - **Vylepšení splnitelnosti** (flip náhodného literálu, který figuruje v nesplněné klauzuli) – 0.79
- **Porovnávání stavů** bylo prováděno pomocí soft penalizace (nesplnitelný stav může mít lepší skóre, než splnitelný). Penalizace se odvíjí od počtu nesplněných klauzul a je navíc škálována součtem všech vah. Toto škálování je korigováno pomocí škálovacího parametru, který byl od iterace 3 nastaven na 0.05.

- **Zamrznutí** bylo prováděno na základě poměru odmítnutých stavů v rámci equilibria, nebo překročení minimální teploty. Poměr odmítnutých stavů byl nastaven na 0.9 (vůbec jsem jej experimentálně nevyhodnocoval, nastavil jsem jej podle „délky ocasu“ v grafu heuristiky při white box měření).
- **Počáteční a koncová teplota** je škálována součtem všech vah. Toto škálování je korigováno nastavitelnými koeficienty. Koeficient pro minimální teplotu byl vždy nastaven na 0.001 (tato hodnota není příliš podstatná, protože v naprosté většině případů zamrznutí nastane vysokým poměrem odmítnutých stavů). Koeficient pro maximální teplotu jsem od první iterace nastavil na 1 (opět nastaveno při white box měření).
- **Koeficient chlazení a koeficient počtu kroků equilibria** byly detailněji rozebrány ve čtvrté iteraci. **Ukázalo se, že heuristika je velmi spolehlivá pro koeficient chlazení = 0.95 a koeficient kroků equilibria = 50.**
- **Restartování teploty** se ukázalo nebýt příliš přínosné, protože jsem musel heuristiku velice „příškrtit“, aby byl restart nutný. Nicméně na základě měření jsem maximální počet restartů nastavil na 3, protože na histogramech byl nejčastější počet restartů do této hodnoty.

4 Závěr

V rámci této úlohy jsem si velmi podrobně vyzkoušel implementovat a nasadit heuristiku pro řešení problému 3SAT. Nejnáročnější částí celého tohoto procesu bylo jednoznačně nasazování heuristiky, kdy jsem objevil několik slabin a nedostatků mé heuristiky. Po několika iteracích se heuristiku podařilo nastavit tak, aby poskytovala řešení dobré kvality pro všechny poskytnuté sady. Bohužel mi nezbýl čas na to, abych mohl výkon heuristiky porovnat s jiným heuristickým řešičem/brute force metodou, výkon heuristiky však považuji za dobrý (je jednoznačně rádově lepší, než brute force, soudě dle počtu kroků uvnitř algoritmu).

Při nastavování heuristiky jsem mnoho času strávil ve white-box fázi, kdy jsem mohl detailně sledovat chování heuristiky v závislosti na nastavených parametrech. Díky tomu, že jsem investoval čas do tvorby UI aplikace, která automaticky vykreslí graf heuristiky (v případě běhu více instancí zobrazí statistiky o běhu), jsem si heuristiku mohl velmi dobře osahat a odladit. Bohužel se tento mnou zvolený způsob testování velmi těžko přenáší do kompaktních grafů a nezaručuje, že chování heuristiky je opravdu optimální. Kdybych úloze mohl obětovat více času, pak bych se zaměřil na detailnější experimentální vyhodnocení parametrů, které jsem u některých parametrů vůbec neuvedl.