Řešení problému vážené splnitelnosti booleovské formule pokročilou iterativní metodou

16. ledna 2021, Radek Šmíd

https://github.com/Smidra/KOP-ukol-5

1 Specifikace úlohy

Cílem úlohy bylo implementovat jeden z pokročilých algoritmů pro řešení váženého problému 3SAT. Na výběr bylo z:

- Simulované ochlazování
- Genetické algoritmy
- Tabu search

Po úspěšném nasazení algoritmu bylo za úkol jeho vlastnosti ověřit experimentálním vyhodnocením. Zpráva by měla dokládat záznam o celém postupu řešení problému a nsazení. Proces experimentace (white box i black box fáze testování) jsem se pokusil zachytit v tomto dokumentu. Výsledné naměřené hodnoty pak prezentovat a pokusit se o interpretaci[1].

2 Stručný popis zvoleného algoritmu

Zvolil jsem si algoritmus simulovaného ochlazování, stejně jako v předchozí úloze. Je to pravděpodobnostní metoda řešení problému založená na analogii se simulací ochlazování tuhnoucího kovu. Metoda prohledává stavový prostor a má tendenci zůstávat v lokálních minimech. Algoritmus nezaručuje nalezení globálního minima, ale správým nastavením jeho parametrů (teplota, chladící koeficient) můžeme docílit velmi přesných výsledků.

3 Rozbor řešení

Řešení staví na pevných základech objektového návrhu z předchozích cvičení. Pro tuto úlohu jsem ale nemohl předchozí kód pouze rozšířit o novou metodu. Musel jsem značné části celého kódu přepsat a vytvořit tak nové třídy, které korektně implementují problém váženého 3SAT (dále už jen 3SAT). Instance 3SAT problému jsou drženy v poli jako objetky třídy CNFInstance. Třídu KnapsackState z předchozího cvičení jsem přepsal jako třídu CNFState. Ta reprezentuje jeden stav (ohodnocení proměnných) nějaké instance 3SAT.

Má metody:

- is_solution() – Vrací bool podle toho jestli jsou všechny maxtermy pravda pro toto ohodnocení. Pokud nějaký maxterm není splněný, poznamená si proměnné, které se v něm vyskytují jako „podezřelé“.
- refresh() – Zavolá is_solution() aby vystavěl aktuální množinu podezřelých proměnných. Pak spočítá celkovou váhu pro toto konkrétní ohodnocení.
- flip(n) – Proměnnou na pozici n invertuje (0 na 1, nebo 1 na 0)
- randomize() – Ohodnotí proměnné náhodně a zavolá metodu refresh().
- random_start() – Pokusí se oprakovaným voláním metody randomize() nalézt platné řešení. Pokud ho po n krát 1000 krocích nenalezne, spokojí se s jakýmkoliv náhodným stavem.
- is_better(state_challenger) – Porovná, jestli tento stav lepší než challenger stav. (Více 3.4.)
- random_neighbour() – Vrátí náhodný sousední stav. (Více 3.5.)

[1] https://moodle-vyuka.cvut.cz/mod/assign/view.php?id=89703

3.1 Stavový prostor

Jako stavový prostor 3SAT problému jsem zvolil všecha možná ohodnocení proměnných. Algoritmus tedy připouští i takové stavy, které mají proměnné ohodnocené tak, že celková CNF není splněna. Zvolil jsem tento prostor proto, že je spojitý a jistě obsahuje optimální řešení. Taková volba značně ztíží vyladění algoritmu, protože jistě budou existovat takové stavy, které mají lepší váhu, ale nejsou řešení. Pro získání co nejlepších vlastností podle přednášky je prospěšné začínat z náhodného stavu. Proto je třeba vygenerovat na začátek náhodné řešení. Jenže to také nemusíme najít.

3.2 random_start()

Tato funkce se použije při hledání startovacího stavu. V cyklu randomizuje stav a ptá se jestli je to validní řešení. Tento cyklus opakuje až do tisícinásobku počtu proměnných. Pokud se jí ani pak nepodaří najít validní řešení svojí práci vzdá a začne od jakéhokoliv náhodného řešení.

3.3 Operace

Zvolené operace jsou změna jednoho bitu. Tedy buď změna ohodnocení proměnné z 1 na 0, nebo naopak. Tato operace je implementována metodou flip(). Po změně je vždy volána metoda refresh(), která atributy objektu aktualizuje.

3.4 is_better(state_challenger)

Metoda, která rozhodne, jestli je lepší tento stav, nebo vyzivatel. Po dlouhé úvaze jsem se rozhodl tuto metodu ponechat implementovanou obdobně jako v úloze 4. Existují čtyři možnosti, které mohou nastat.

- Tento (je řešení) x Vyzivatel (je řešení) – Rozhodne lepší váha.
- Tento (není řešení) x Vyzivatel (není řešení) – Rozhodne lepší váha.
- Tento (není řešení) x Vyzivatel (je řešení) – Tento je lepší.
- Tento (je řešení) x Vyzivatel (není řešení) – Vyzivatel je lepší.

3.5 random_neighbour()

Se použije ve funkci **try()**, kde vybírám náhodného souseda. Tedy takový stav, pro který stačí použít jednou metodu flip. Nalezený soused nemusí být validní řešení. Abych pomohl heuristice od „bezcílného“ bloudění, tak se nejdříve podívám, zda je stav řešení. Pokud není, pak cesta k řešení vede pouze skrz obracení proměnných přítomných v nesplněných maxtermech. Pokud stav je řešení, vyberu náhodnou proměnnou (s plným vědomím, že pokud to nebude řešení, tak vždy zvítězí stávající stav).

3.6 Normalizace ceny

Aby nehrály roli jednotky ve kterých je váha uvedena (koruny/haléře), tak jsem rozdíl cen implementoval speciální funkcí. Ta vezme váhu prvního stavu ($c1$) a vypočítá konstantu a pomocí které lze naškálovat $c1$ na 100. Váhu druhého stavu pak vynásobí stejnou konstantou. Od 100 pak odečte $c2$ a získá normalizovaný rozdíl vah stavů, který se nebude s jednotkami měnit. Tím získáme stabilnější algoritmus, protože abolutní velikosti vah nebudou mít vliv na teplotu při vyhodnocování funkce try(). Úplně stejně jako s cenami v úloze 4.

3.7 Ukončení

Funkce frozen vrátí True po 300 kolech beze změny. Funkce equilibrium vrátí True po dvojnásobku počtu věcí v instanci.

3.8 Bonus

Třída CNFState obsahuje pole libovolného množství objektů třídy Maxterm. Všechny metody její metody volají metody této třídy, když s maxtermy operují. Třída maxterm obsahuje pole pravdivostních hodnot:

- 0 – Tuto proměnnou maxterm neobsahuje
- -1 – Tuto proměnnou maxterm obsahuje v negaci
- 1 – Tuto proměnnou maxterm obsahuje a není v negaci

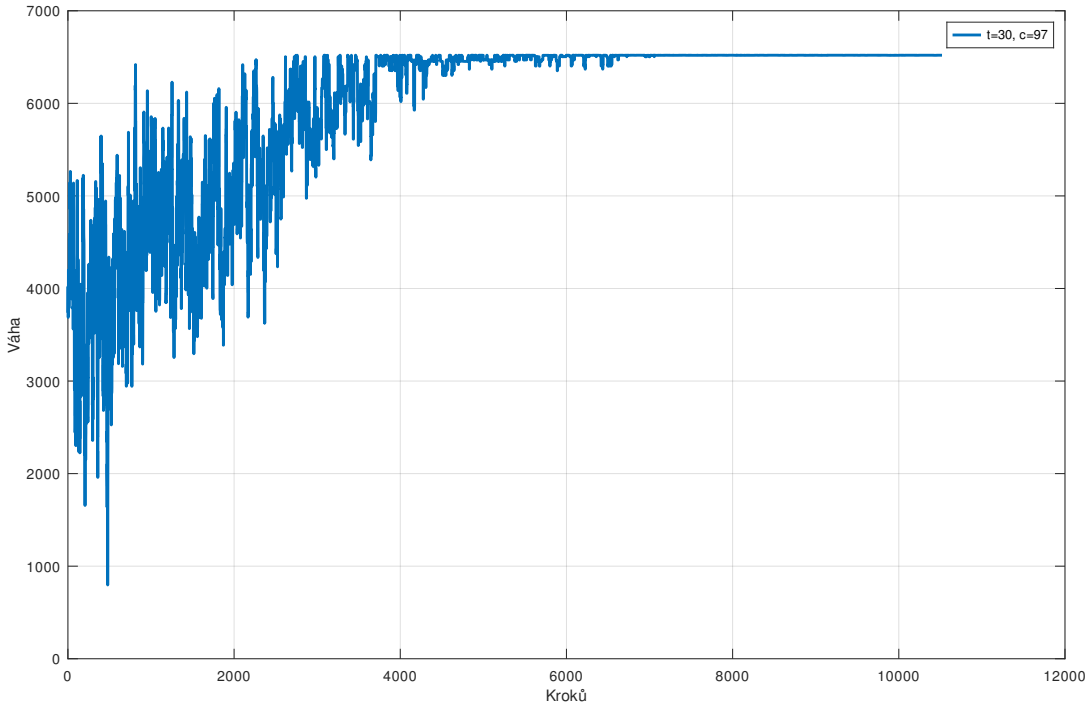
Při dotazu na splnitelnost iteruje metoda přes toto pole. Tedy maxtermy mohou obsahovat libovolné množství proměnných. Více proměnných funguje včetně načítání ze souboru, které přestane stavět maxterm až při přečtení nuly dle specifikace.

4 Experimentace

Proces experimentace jsem si téměř celý automatizoval. Program generuje do souboru **chart_X_YZ.dat** vývoj váhy, přímo zatímco ji počítá. GNU Octave tento soubor čte a na požádání z něj rovnou vytvoří graf vývoje váhy v krocích.

Používám dodané instance **wuf20-78-M1** o velikosti dvaceti proměnných a sedmdesáti osmi max-termy. Prvotní nastavení parametrů by mohlo být náhodné, ale rozhodl jsem se pro výsledné hodnoty čtvrtého úkolu. Tedy $t = 30$ a $c = 0.97$. V první fázi se pokusím vyladit algoritmus na instanci problému s $id = 01$.

Sleduji vývoj ceny v grafu níže. Algoritmus došel k řešení 6403, což je správné řešení první instance. Graf vypadá tak, jak by měl – konverguje v polovině až dvou třetinách.

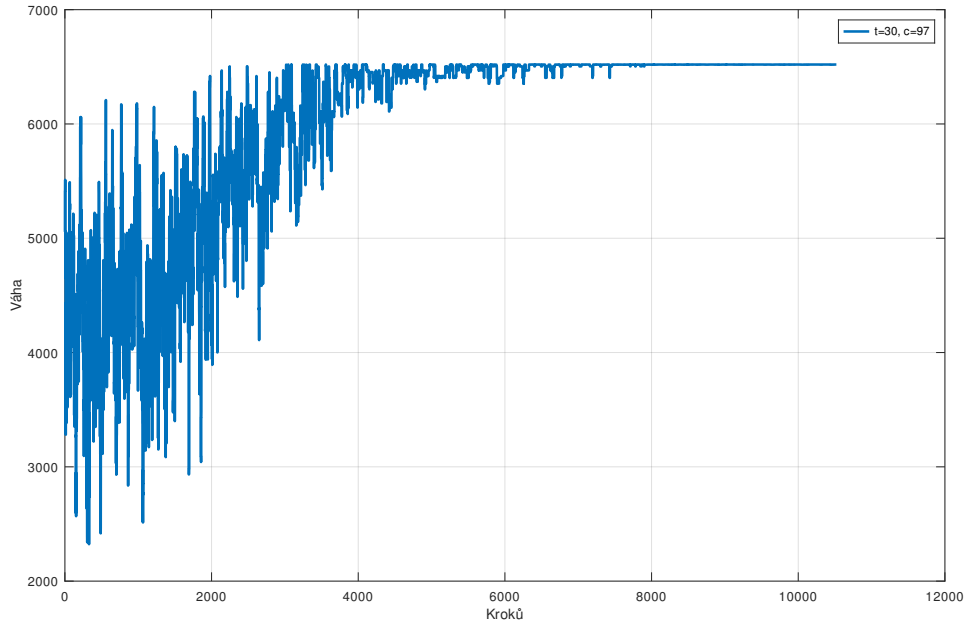


Obrázek 4.1 Vývoj ceny stavu pro první experiment.

Všechno sice vypadá ideálně hned na první pokus, ale po bližším prozkoumání vah samotných zjišťuji dva problémy. První je, že se váhy ustálí na hodnotě 6520, nikoliv na 6403. Druhý problém je, že to, co běh ukončilo, je pevně daná dolní mez teploty. Tedy ne daný počet kol beze změny. V přibližně poslední třetině se stavy mění pouze mezi 6520 a 6519.

Nejprve se pokusím opravit druhý problém. Prohlédnu metodu is_better(), ale ta vrátí „True“ pouze pokud je stav ostře lepší. Domněnka -1 erroru při porovnání tedy platit nebude. Půjde tedy pravděpodobně o rozhodnutí heuristiky zkusit přijmout i horší stav. To by mělo jít změnit úpravou parametrů.

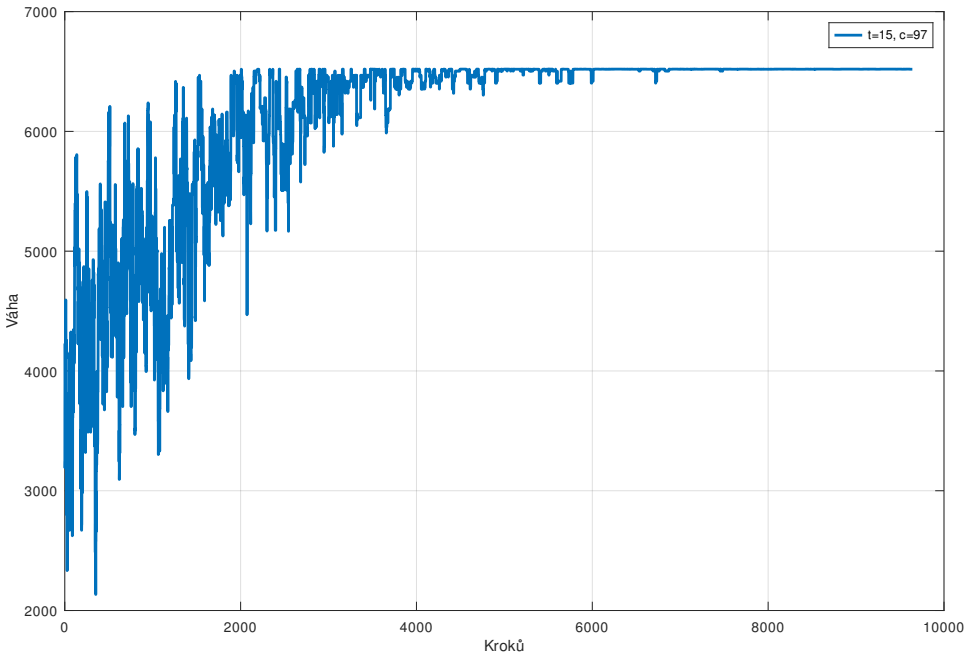
První problém je záludnější. Intuitivně bych pomohl heuristice držet se u řešení tak, že bych ho více odměnil. Jenže ve stávající implementaci řešení vždy vítězí nad neřešením. Nelze mu tedy více přidat. Je, ale možné, že algoritmus mate chytře implementovaný výběr souseda. Možná způsobil přesný opak zamýšleného a místo rovnoměrného procházení prostorem pomale dojde k řešení a rychle ho zničí. Změním tedy implementaci. Nyní nebere proměnné v nesplněných maxtermech v úvahu.



Obrázek 4.2 Vývoj ceny stavu pro druhý experiment.

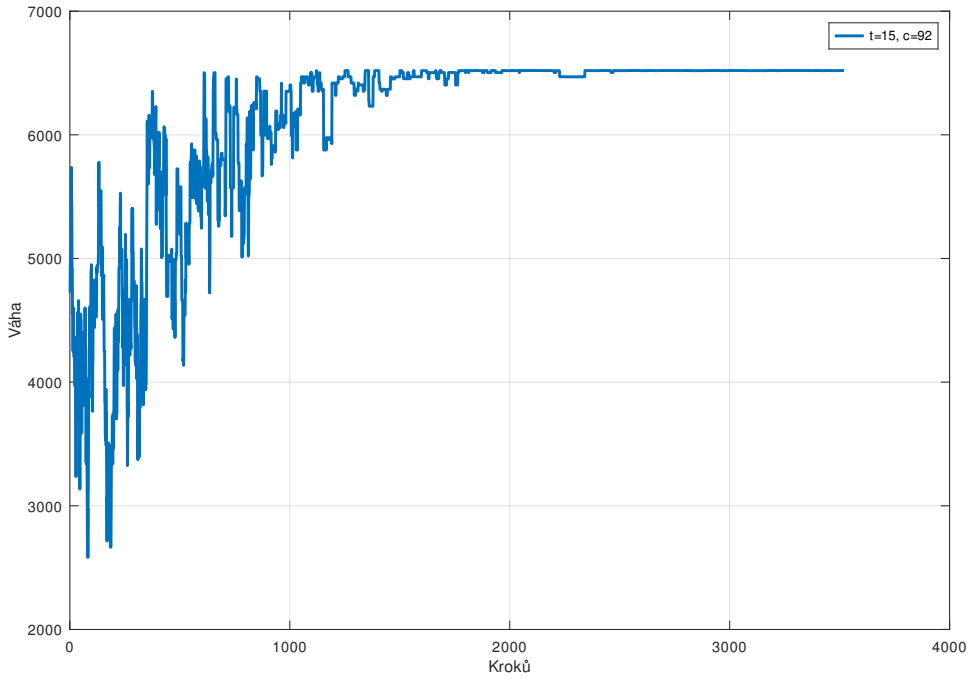
Běh programu opět odhalil správné řešení problému. Bohužel opět vykazuje stejné chyby jako předchozí. Tedy to nevypadá, že by něco razantně měnilo. Z grafu je navíc vidět, jak heuristice mna počátku běhu delší dobu trvá než začně k řešení konvergovat. To by mohlo odpovídat změně v kódu – cesta k řešení je těžší. Jenže změna naváděla na jakékoliv řešení, nehledě na váhu. Mohl bych tento problém ověřovat empiricky, ale v této fázi to nevypadá, že by změna měla nějaký pozitivní efekt vedoucí k vyřešení stanovených problémů. I přesto změnu zatím v kódu nechám, protože je tak průchod symetričtější.

Další možností, proč se program ukončí ve špatném stavu je příliš veliká diverzifikace. Pokusím se jí snížit snížením počáteční teploty ze třiceti na patnáct.



Obrázek 4.3 Vývoj ceny stavu pro třetí experiment.

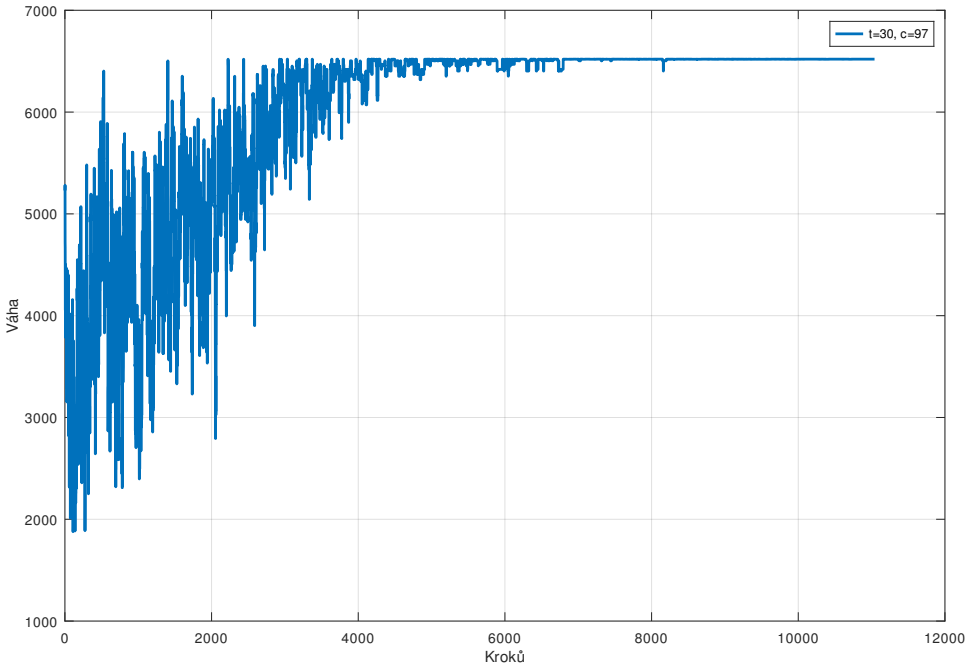
Viditelně se zvětšila intezifikace, zmenšil se počet kroků, ale oba problémy stále přetrvávají. Nezbývá, než se pokusit zvýšit intenzifikaci, tentokrát snížením chladícího koeficientu (chlazení bude rychlejší). Změním chladící koeficient z původních 0.97 na 0.92.



Obrázek 4.4 Vývoj ceny stavu pro čtvrtý experiment.

Sice se mi podařilo téměř trojnásobně zmenšit počet kroků při hledání a stále najít správné řešení, ale oba problémy přetrvávají. Otázkou stále zůstává, jak heuristiku přesvědčit, aby raději zůstala v optimálním řešení.

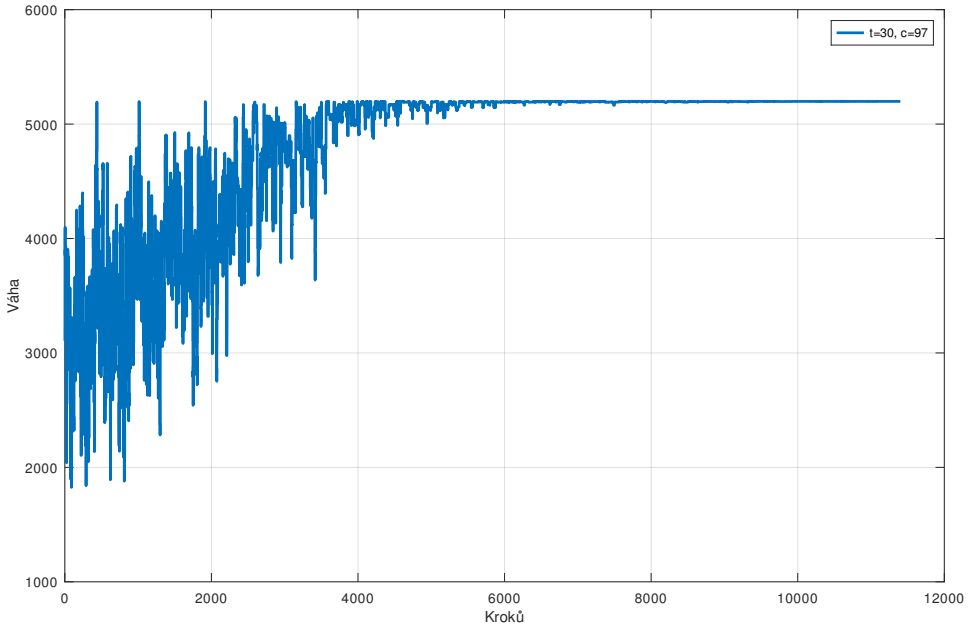
Další možnost jak opravit, že se výpočet ukončí na příliš nízké teplotě a ne na stabilní hodnotě je zkrátka snížit definici „příliš nízké teploty“. Pustím tedy běh s původními hodnotami teplota 30 a chladící koeficient 0.97. Nyní ale desetkrát zmenším escape temperature na 0.001.



Obrázek 4.5 Vývoj ceny stavu pro pátý experiment.

To úspěšně vyřešilo problém předčasného ukončení. Teplota se nyní úspěšně stabilizuje na hodnotě 6520 po tři sta kroků, než se heuristika ukončí. Problémem stále zůstává, že to není optimální řešení.

Zatím vždy průchodem algoritmus řešení nalezl. Pokud se ale vývoj váhy neustálí na správné hodnotě, řešení nalezl „cestou“. Vyzkouším, jestli ochlazování stejně stabilně nalezne řešení i pro druhou instanci. Pustím tedy algoritmus bez dalších úprav na druhou instanci.

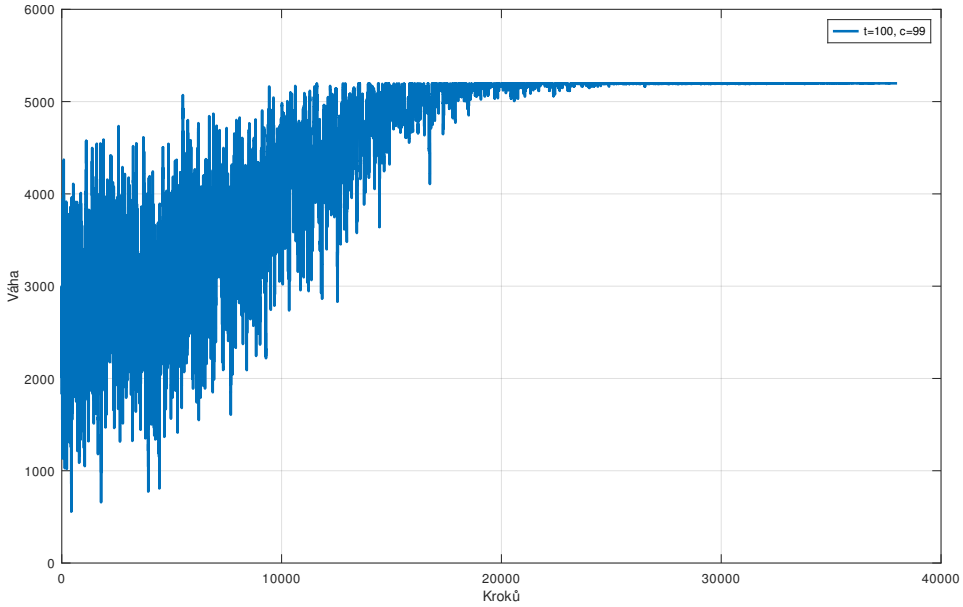


Obrázek 4.6 Vývoj ceny stavu pro šestý experiment (druhá instance).

Algoritmus se ustálil na hodnotě 5198 po tři sta kol a ukončil se. Jako nejlepší řešení předložil váhu 4118. To není nejlepší řešení. Jinak vývoj váhy a čas vypadá obdobně předchozím grafům. Zkusím špatný výsledek zreplikovat opakovaným během algoritmu.

Běh	Výsledek	Správně
1	4118	NE
2	4747	ANO
3	3195	NE
4	3999	NE
5	3718	NE
6	4359	NE

Čas i vývoj zůstávají nezměnené, ale ukazuje se, že algoritmus velmi chybuje. Stabilně správný výsledek byl tedy pravděpodobně důsledekm specifické konfigurace první instance. Doplním si tedy program o výpis vždy při porovnání stavů. Po bližším prozkoumání ale neodhalím žádnou chybu v implementaci metody is_better(). Vrátím se tedy k úpravě hodnot. Pokud heuristika nenechází správné řešení, musím jí více diverzifikovat. Tedy výrazným zvýšením počáteční teploty ze 30 na 100 a chladícího koeficientu z 0.97 na 0.99.



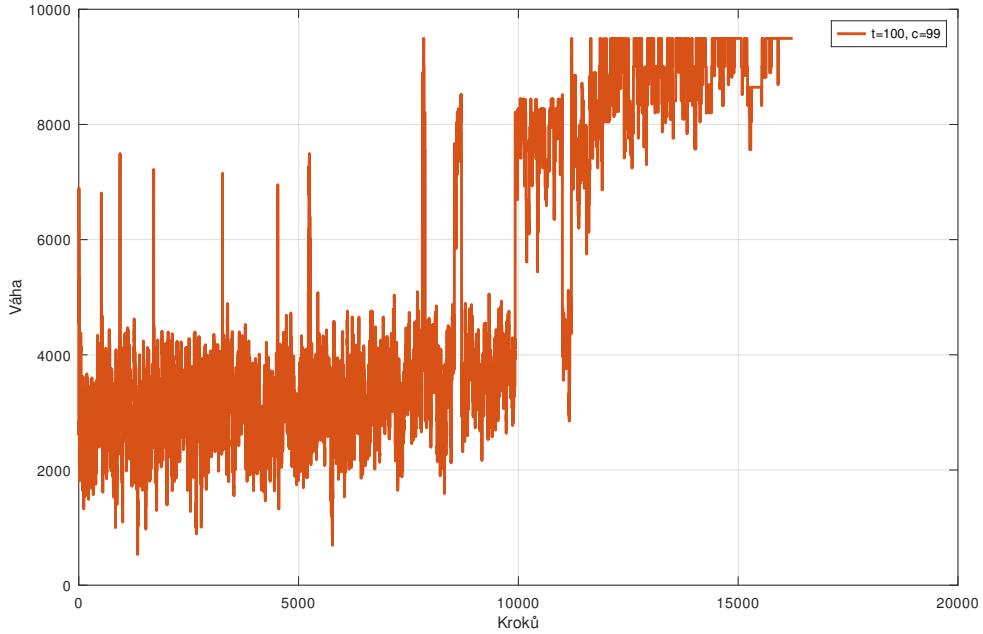
Obrázek 4.7 Vývoj ceny stavu pro sedmý experiment (druhá instance).

To vyprodukovalo správné řešení 4747, i když ve dvakrát až třikrát horším čase. Čas je stále snesitelně dlouhý a tak se pokusím zreplikovat dosažení správného výsledku opakovaným během algoritmu. Tím nahlédnu, jestli šlo o pouhou náhodu, nebo je podle očekávání algoritmus opravdu „silnější“.

Běh	Výsledek	Správně
1	4747	ANO
2	4747	ANO
3	4747	ANO
4	4747	ANO
5	4359	NE
6	4747	ANO

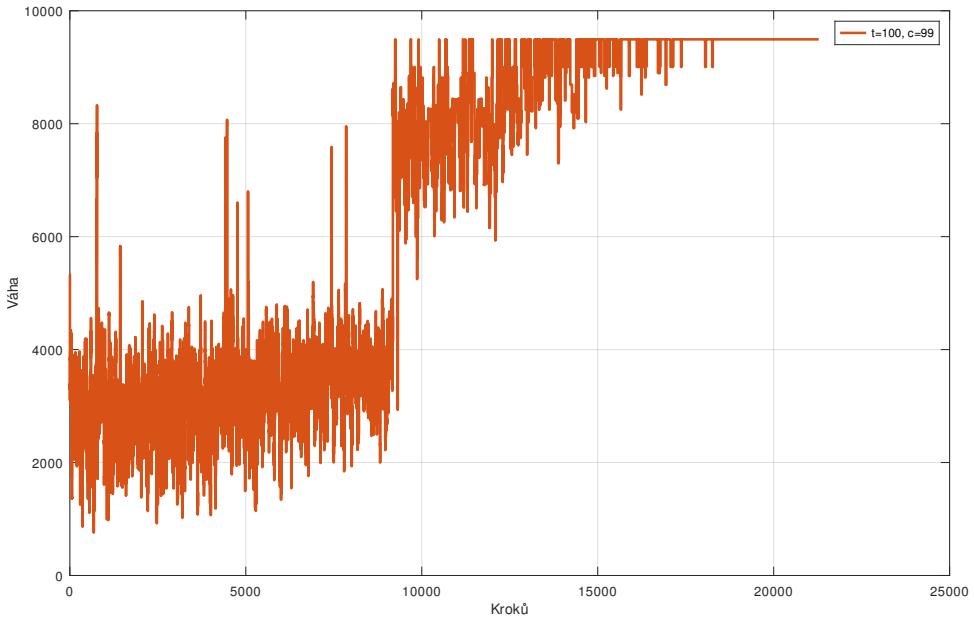
Došlo k výraznému zlepšení. Heuristika už ve většině případů našla správné řešení. Stále se na něm, ale neustálila. Stav ke kterému heuristika i přes všechny dosavadní snahy dokonverguje je přijetí všech proměnných čímž dosáhne sice nejvyšší ceny, ale nesplní požadavek splnitelnosti. Odměna za nalezení platného stavu už je maximální možná. Odměnu ale implementuje metoda is_better(). Přeprogramuji

zmíněnou část tak, aby váha splněného stavu byla násobena konstantou REWARD přímo při přepočítání pomocí metody refresh(). Metoday is_better() pak bude pouze srovnávat váhy. Jako násobek REWARD zvolím náhodně vybranou hodnotu 2.



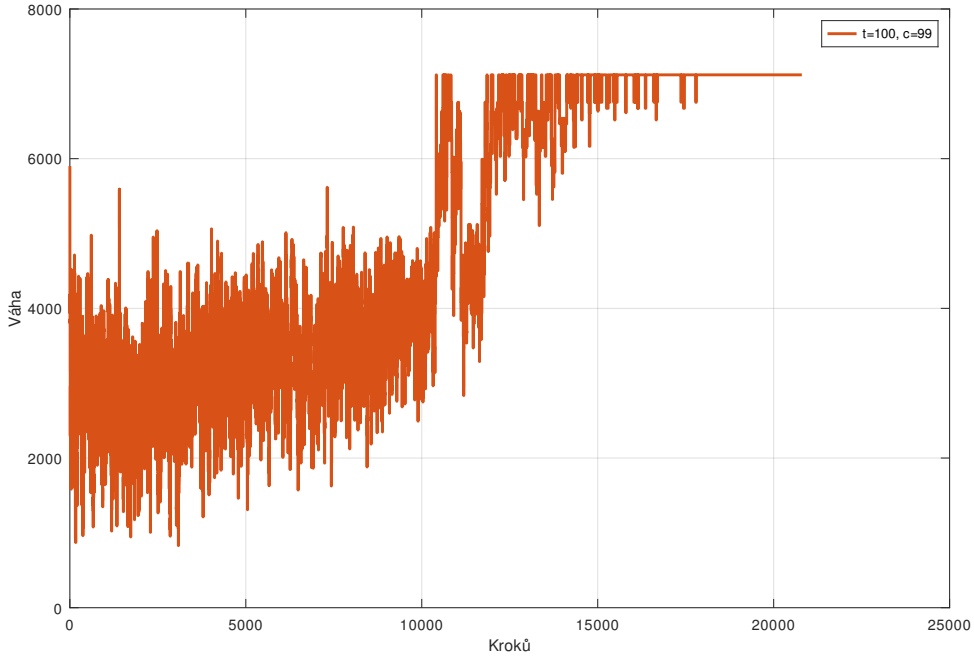
Obrázek 4.8 Vývoj ceny stavu pro osmý experiment (druhá instance).

Graf jsem se rozhodl kvůli zásadní změně v kódu obarvit jinou barvou. Výsledek ukazuje výrazně jiný postup, než všechny ostatní grafy. Zásadní posun je ale v tom, že se ochlazování ustálí na optimální hodnotě. Stavy které jsou násobeny koeficientem REWARD jsou zřetelné ve skocích. Z konce grafu je zřejmé, že heuristika neměla dostatek času se ustálit ve správném řešení. Počet kol beze změny tedy zvednu na desetinásobek (3000).



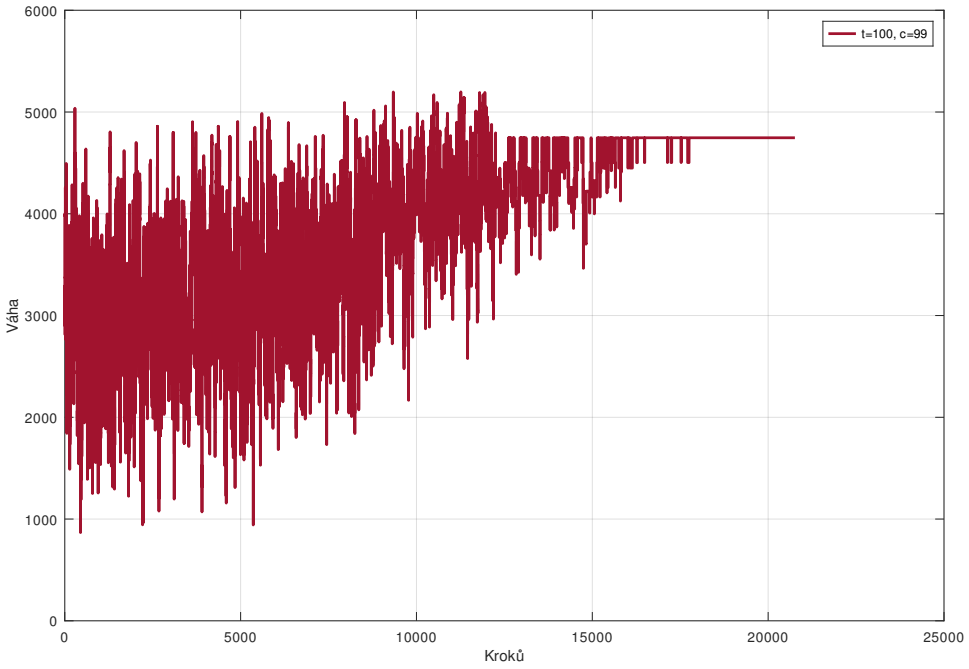
Obrázek 4.9 Vývoj ceny stavu pro devátý experiment (druhá instance).

Změna se projevila přesně podle očekávání. Graf se v posledních krocích ustálil na delší dobu a to na správné optimální hodnotě řešení. Tím jsem překonal druhý z problémů. Heuristika dovává REWARD-násobek řešení. Náhlý skok uprostřed vývoje, může způsobit uvíznutí v lokálním extrému. Zmenším tedy koeficient REWARD, abych alespoň částečně propojil části grafu před a po skoku. REWARD bude v dalším běhu 1.5.



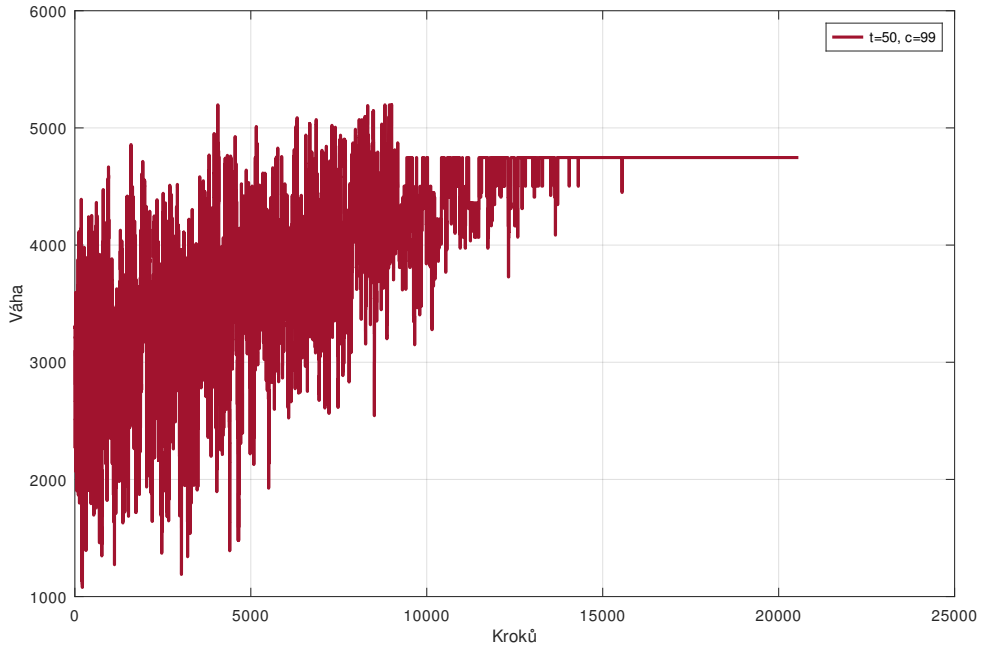
Obrázek 4.10 Vývoj ceny stavu pro desátý experiment (druhá instance).

Graf opět nalezl správné řešení, ale problém skoku přetrvává. Po zamyšlení, ale shledávám chybný graf, ne výpočet. Graf totiž pomocí jedné křivky vykresluje dvě rozdílné hodnoty. Váhu a REWARD-násobek váhy. To způsobuje skok. Násobenou výhu tedy využiji pouze k vnitřnímu fungování algoritmu. Do grafu zaznamenám váhu před násobením. Po přepsání části kódu vypisující váhu v kroku spustím ochlazování se stejnými parametry.



Obrázek 4.11 Vývoj ceny stavu pro jedenáctý experiment (druhá instance).

Barvu grafu jsem pozměnil abych naznačil, že se jedná o zásadní změnu v přístupu vykreslování. Ustálení je stále spíše na konci. Optimálně by mělo být ve středu až ve dvou třetinách. Toho se pokusím dostáhnout mírným zvýšením počtu tahů beze změny (na 5000) a snížením teploty (na 50).



Obrázek 4.12 Vývoj ceny stavu pro dvanáctý experiment (druhá instance).

S grafem jsem již poměrně spokojen. Než přejdu do fáze black-box testování pustím heuristiku vícekrát za sebou abych odchytil případně velmi chybující nastavení.

Běh	Výsledek	Správně
1	4747	ANO
2	4747	ANO
3	4747	ANO
4	4747	ANO
5	4747	ANO
6	4747	ANO

Heuristika se zatím vždy úspěšně ustálí na optimálním řešení za cca 1 minutu. Přejdu tedy do fáze black box testingu. Začnu s malým množstvím instancí ze souboru. Pustím heuristiku na prvních 10 instancí. Hodnoty srovnám s referenčním řešením.

50/0.99	MAX	AVG
Chyba	0%	0%
Čas	70.2s	58.2s

Heuristika vykazuje skvělé výsledky v přijatelném čase. Nechám ji pracovat přes noc. Za osm hodin je schopna stihnout kolem čtyř set instancí.

5 Zhodnocení experimentů

Experiment doběhl po šesti hodinách, tedy rychlost předčila očekávání. Celkové výsledky pro prvních čtyři sta instancí jsem shrnul do tabulky níže.

50/0.99	MAX	AVG
Chyba	44.1%	0.2%
Čas	81.5s	53.2s

Čas se poměrně stabilně drží kolem minuty. Vynikající je i průměrná chybovost, pouhých 0.2%. Znepokojující je pouze vysoká maximální chyba. Po bližším prozkoumání se potvrdí doměnka vycházející z nízké průměrné chyby – jedná se o ojedinělou hodnotu. Jedninečná hodnota byla pravděpodobně způsobena hraniční hodntou.

Veliká chyba se vyskytla u instance 339, kde podala výsledek 3995 místo dalého 7149. Po zkontrolování výsledku v řešení, ale zjišťuji, že je výsledek správně. Řešení 7149 patří instanci 340, která byla ze souboru později vypuštěna[1]. Python tedy ze souboru chybně přečetl správné řešení a vypočítal tak špatnou chybu instance. Po opravení chyby a důsledném překontrolování jsou výsledky následující.

50/0.99	MAX	AVG
Chyba	9.39%	0.09%
Čas	81.5s	53.2s

Takové výsledky ukazují, že heuristika vrací poměrně kvalitní výsledky pro tyto instance.

6 Závěr

V domácím úkolu jsem implementoval simulované ochlazování a pokusil se metodickým přístupem nalézt nejlepší parametry pro zadané NK–32 instance. Herustika velmi citlivě ragovala na změnu všech parametrů. Nejenom ale na dva hlavní (počáteční teplotu a ochlazovací koeficient). Je také zásadní mít korektně nastavené hodnoty ukončování pro metodu frozen. Zyšování počáteční teploty a chladícího koeficientu pomalu vylaďujeme intezifikaci a diverzifikaci heuristiky pro náš algoritmus. Vyzkoušel jsem si práci s pokročilou heuristikou a upravil svůj program na automatizaci úkonů spojených s vizualizací od pythonu až po matlab grafy.

[1] https://moodle-vyuka.cvut.cz/mod/assign/view.php?id=89703