České vysoké učení technické v Praze
Fakulta informačních technologií

Kombinatorická optimalizace

Druhý domácí úkol

Nasazení pokročilé heuristiky

Autor: David Omrai (omraidav)

PRAHA, LEDEN 2024

Contents

1 Úvod

V této práci se zaměřuji na implementaci pokročilé (meta)heuristiky pro řešení problému maximální vážené splnitelnosti booleovské formule (MWSAT), bez interaktivních zásahů napříč rozdílnými velikostmi instancí.

Jako pokročilou heuristiku jsem zvolil simulované ochlazování, které jsem postupem experimentů, debugovaní a zkoumání komplexnosti výpočtů postupně po-upravoval, abych dosáhl co nejlepších výsledků.

Tato práce je rozdělena na tři logické celky. V první popíši jaké data jsem si pro tuto práci zvolil, jaké algoritmy využívám (jejich implementaci) a parametry stroje, na kterém experimenty provádím. Druhá část je věnována white-box fázi nasazení heuristiky, kde postupně ladím hyper-parametry heuristiky, měním strategie a logické bloky implementace heuristiky, včetně dat, ze kterých vyvozuji závěry. A nakonec ve třetí a poslední části se věnuji black-box fázi, kde testuji odladěnou heuristiku v závěrečném vyhodnocení, zkoumám její výkon na řadě různě obtížných instancí, které nebyli využity v předchozí fázi.

2 Popis systému, dat a algoritmů

V této sekci popíši, jakého problému se táto práce věnuje, v jakém formátu jsou její instance, jakou jsem zvolil pokročilou heuristiku a její části, které se nemění dále v práci.

2.1 Informace o systému

Veškeré experimenty jsem provádél na svém osobním počítači, který má následující parametry.

CPU AMD Ryzen 7 3700U [8 jader]

RAM 32 GB [29.2 GiB] + 24 GB swap

OS openSUSE na SSD

2.2 Data

V této práci se řeší problém maximální vážené splnitelnosti booleovské formule, nebo taktéž MWSAT, s formulemi v konjunktivní normální formě s možnou různou délkou literálů v klauzulích. Nicméně použité instance v této úloze si formule drží pevný formát těchto formulí a to tak, že v každé klauzuli jsou právě tři literály, což tedy tvoří MW3SAT, ale program lze aplikovat i na instance s různou délkou klauzulí. Příklad zjednodušené formule využité v této práci.

$$(A \vee B \vee \neg C) \wedge (C \vee \neg D \vee B) \wedge (\neg A \vee \neg B \vee D)$$

Data pro tuto práci jsem čerpal ze školních webových stránek courses [2], která jsou na bázi SATLIB instancí, které byly doplněny o váhy. Tyto instance jsou rozděleny do čtyř kategorií M, N, Q a R. Instance M a N by měly mít váhy, které podporují nalezení řešení. Heuristika, která řeší určitou instanci v sadě M, by měla řešit stejnou instanci sady N. Instance Q a R vytvářejí mírně zavádějící úlohu, u kterých se počítá, že se kvalitní heuristiky přizpůsobí.

Využité instance pro experimenty a testování jsou tedy wuf20-91, wuf50-218 a wuf100-430, které mají ve svém názvu počet klauzulí a instancí. Voleny byly tak, abych obsáhl jak jednoduché na ladění, těžší a nejtěžší na ověření kvalit heuristiky.

2.3 Algoritmy

V této práci využívám pouze jednu heuristiku, a tou je simulované ochlazování pro řešení instancí problému MWSAT.

2.3.1 Simulované ochlazování

Simulované ochlazování vychází z techniky, při které se opakovaným zahříváním a ochlazováním materiálu (žíhání) zlepšuje jeho kvalita. Na počátku simulovaného ochlazování se nastaví proměnná představující teplotu na vysokou hodnotu. Během procesu prohledávání dochází k pomyslnému ochlazovaní teploty, snižováním její hodnoty. Čím vyšší teplota je, tím je umožněna častější volba zhoršujících řešení oproti aktuálnímu. Toto řeší problém s lokálními optimy, ze kterých metaheuristika ze začátku snáze uniká. Se snižováním teploty se snižuje i šance přijmutí zhoršujícího řešení, a tak dochází k postupnému umožnění prohledávání cílového prostoru, které je potenciálně blízké globálnímu optimu. Proces postupného ochlazování je to, co tuto metaheuristiku činí velice efektivní při hledání řešení blízké optimálnímu a při řešení velkých problémů obsahující četný počet lokálních optim. [1]

2.3.2 Implementace

Podobně jako v předešlé úloze jsem si pro vývoj algoritmu/heuristiky zvolil programovací jazyk Python pro jeho jednoduché nasazení, a možnost jej spouštět jako skript z příkazového řádku.

Program je složen ze dvou tříd, jednou pro samostatnou heuristiku SimAnn, a druhou jako strukturu EvalInfo využívám pro uschování informací o získaném ohodnocení. Simulované ochlazování má několik hyper-parametrů, které se dále v práci pokouším experimentálně nastavovat, pro dosažení nejlepšího výkonu. Mezi tyto parametry se řadí počáteční teplota, finální teplota, maximální počet iterací a míra ochlazování.

Jak jsem již předem zmínil, program lze spustit i jako skript a to pomocí tohoto příkazu.

```
./samwsat.py -i <instance_path> -t <init_temp> -n <max_iter> -c <cooling_fact>
```

Program má dva výstupy, na standardní výstup výsledek, ve kterém jsou zahrnuty i informace o běhu, druhý pak je na standardní chybový, kde jsou informace o přijatých řešení s jejich váhami během běhu heuristiky.

Standardní výstup má následující formát.

```
<step> <temp> <sat> <sat_num> <unsat_num> <weight>
```

step číslo kroku heuristiky

temp aktuální teplota

sat 1 pokud je formule splněna, jinak 0

sat_num počet splněných klauzulí

unsat_num počet nesplněných klauzulí

weight váha řešení

Informace o běhu pak následující.

```
<sat> <sat_num> <unsat_num> <weight> <steps> <duration>
```

sat 1 pokud je formule splněna, jinak 0

sat_num počet splněných klauzulí

unsat_num počet nesplněných klauzulí

weight váha řešení

steps celkový počet kroků heuristiky

duration celkový čas výpočtu

3 White box fáze

V této kapitole se budu zabývat postupným laděním implementace, testováním a vylepšováním implementovaného algoritmu, pro docílení co nejlepších výsledků.

3.1 První fáze

Prvotní implementace simulovaného ochlazování pro řešení problému maximálního ohodnocení vážené booleovské formule měla řadu problémů, které znemožňovaly její běh v adekvátním čase.

U simulovaného ochlazování jsem jako další validní stav bral pouze ten, který splňoval danou formuli, abych se mohl zaměřit pouze na váhy. Avšak běh tohoto řešení byl i pro malé instance velice dlouhý. Pseudokód mého minulého řešení vypadal takto.

```
...
newSol = getNextNeighbour(currSol)
while (evalFormula(formula, newSol) == False) {
    newSol = repair(newSol)
}
...
```

Toto byl už od počátku nesprávný přístup k upravení simulovaného ochlazování pro tuto úlohu. A motivoval mě změnit logiku tak, abych připouštěl i nesplnění formule budu získávat nové řešení a jak je budu přijímat/zamítat.

Nejdříve mě napadlo penalizovat formule, které nejsou splněny pomocí speciální metriky, avšak ze snah takovou vymyslet nevzešlo nic, s čím bych byl spokojený a co by pevně rozlišovalo ty splněné a ty nesplněné, navíc se započítání váhy. Proto jsem přišel s následující logikou.

3.1.1 Závěr

Pro vyřešení zmíněného problému jsem zavedl následující akceptující mechanismus pro nové řešení.

```
def is_better(new_state, old_state):
    # Do both solve the formula?
    if (new_state.sat == True and old_state.sat == True):
      return new_state.weight > old_state.weight
    # Does the new solve the formula and the old don't?
    if (new_state.sat == True and old_state.sat == False):
      return True
    # Do both not solve the formula?
    if (new_state.sat == False and old_state.sat == False):
      return new_state.sat_num > old_state.sat_num
    return False
```

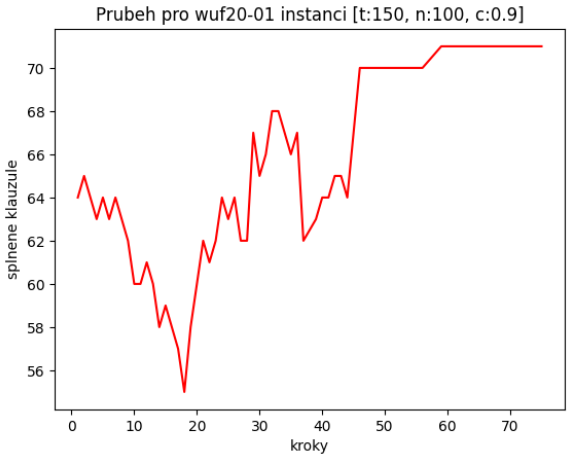
Tento mechanismus přijme nové řešení, pokud je splněna alespoň jedna z jeho podmínek. Otázkou nyní zůstává, jak implementovat přijmutí zhoršujícího řešení. U toho jsem se inspiroval z přednášek a má tedy následující tvar.

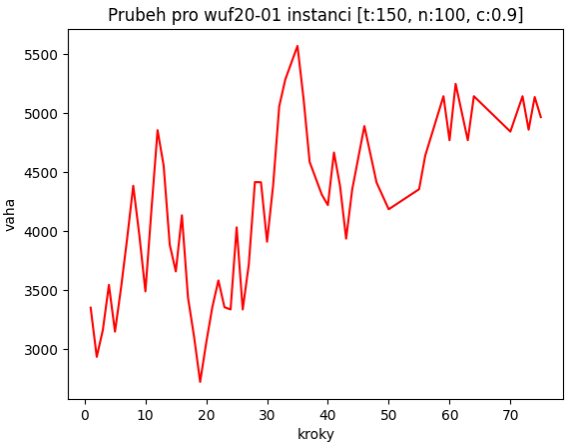
```
def accept_worse(curr_state, old_state, t):
  if t == 0:
    return False
  sat_diff = old_state.sat_num - curr_state.sat_num
  return random.uniform(0, 1) < math.exp(- (sat_diff) / t)
```

Ve zmíněném kódu lze pozorovat, ze se nejdříve ověří, že teplota není nulová. Následně se vypočítá rozdíl splněných klauzulí řešení a tato informace se využije v pravděpodobnosti přijetí.

3.1.2 Experiment

Pro demonstraci navržené implementace jsem SA spustil nad instancí wuf20-01, s teplotou 150 a ochlazovacím faktorem 0.9. I tato instance byla pro původní implementaci velice časově náročná, nyní se již podařilo splnit veškeré klauzule. Výsledná nejlepší váha byla 5248, oproti optimální 6829.





Z průběhu splněných klauzulí lze pozorovat podobnost s vhodného běhu metaheuristiky z přednášky, což se ne tak úplně dá říci o průběhu vah. Toto je nejspíše způsobenu, ze akceptující podmínka pro řešení, které splňuje formuli není závislé na teplotě. Bylo by tedy vhodné prozkoumat, jak lépe motivovat SA k nalezení lepšího řešení, co jednak splňuje formuli a zároveň maximalizuje váhu, o to se pokusím v dalších fázích.

3.2 Druhá fáze

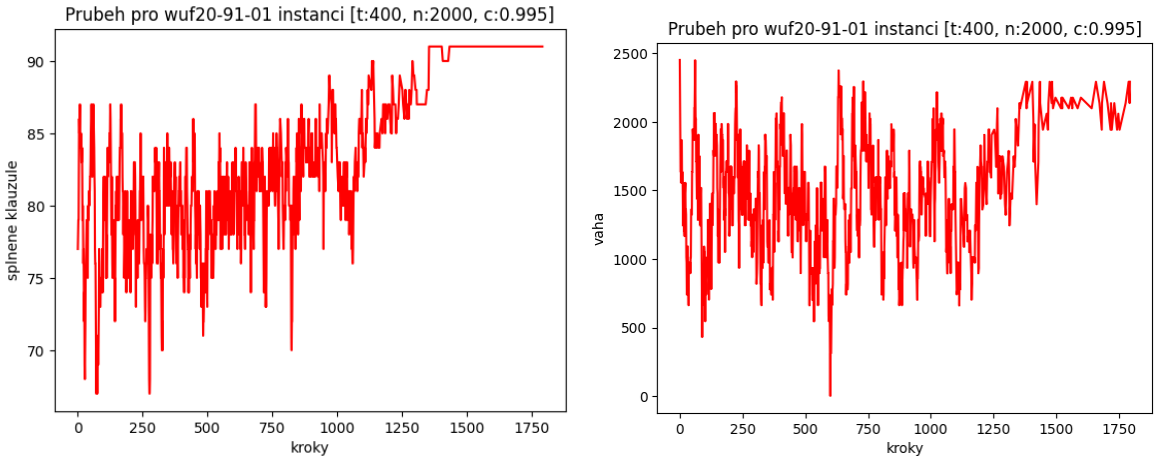
Z minulé fáze mi nedalo přemýšlet o implementaci akceptace zhoršujícího řešení. Nyní je závislé pouze na tom, kolik klauzulí řeší. Zajímalo by mě tedy, jestli zahrnutím této skutečnosti i váhy získám lepší výsledky.

3.2.1 První varianta

První varianta je ta původní a to tedy v následujícím formátu.

```
eps = sat_num_new - sat_num_old
return random.uniform(0, 1) < math.exp((- eps / t))
```

Pro otestování spustím tuto variantu na instanci wuf20-01 z wuf20-91-M.



Nejlepším nalezeným řešením bylo následující.

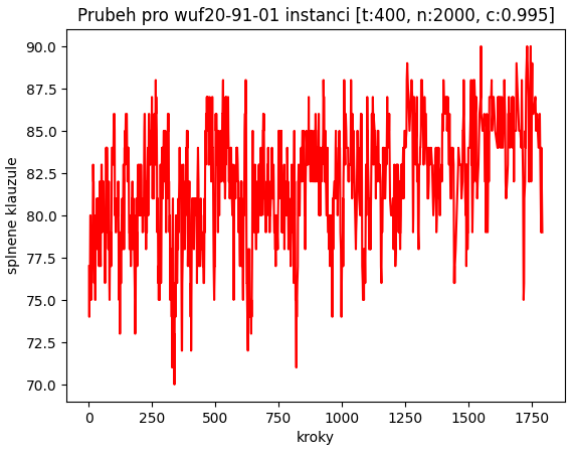
```
1 91 0 2293 1793 0.18665814399719238
```

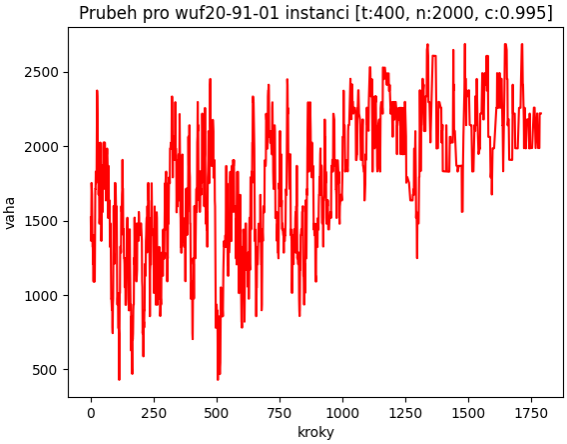
Informace v řádku výše jsou stejné, jak jsem je popisoval v předešlé kapitole, a to <splněno> <splněných klauzulí> <nesplněných klauzulí> <váha> <nalezeno v kroku> <čas výpočtu>.

3.2.2 Druhá varianta

Ve druhé variantě zkusím namísto počtu splněných klauzulí využít pouze váhu ohodnocené formule. U této změny nepředpokládám kladné výsledky, avšak rád bych viděl, jaký bude průběh heuristiky.

```
sum_weights = weight_new + weight_old
eps = (weight_new/sum_weights)*100 - (weight_old/sum_weights)*100
return random.uniform(0, 1) < math.exp((- eps / t))
```





Nejlepší nalezené řešení bylo následující.

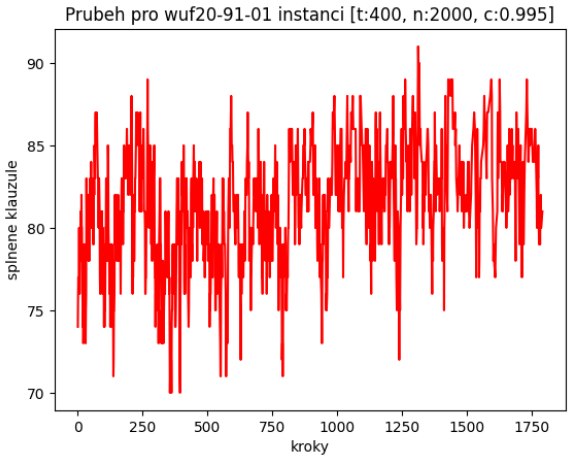
```
0 90 1 1985 1793 0.1857898235321045
```

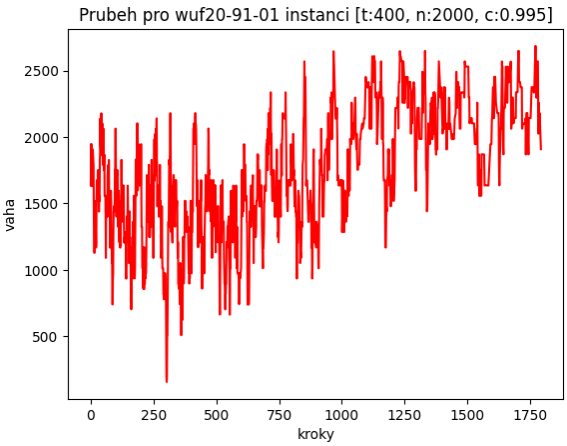
3.2.3 Třetí varianta

V poslední variantě se pokusím vzít v úvahu jak splněné klauzule tak i váhy. Toho dosáhnu tak, že si spočítám pro každou splněnou klauzuli váhy kladně ohodnocených proměnných co se v ní nacházejí.

```
def get_sat_clause_weight(state):
    state_weight = 0

    for clause in formula:
      if self.evaluate_clause(clause, state) == True:
        for c_x in clause:
          if c_x > 0 and state[abs(c_x)-1] == 1:
            state_weight += self.weights[abs(c_x)-1]

    return state_weight

new_w = self.get_sat_clause_weight(curr_state.eval)
old_w = self.get_sat_clause_weight(old_state.eval)
weights_sum = new_w + old_w
eps = (old_w/weights_sum)*100 - (new_w/weights_sum)*100
return random.uniform(0, 1) < math.exp(- (eps) / t)
```

V kódu jsem každou klauzuli vyhodnotil, zdali je splněna, pokud ano, tak jsem sečetl váhy proměnných, které se v ní nacházejí v kladném ohodnocení.





Nejlepší nalezené řešení je následující.

```
1 91 0 1984 1793 0.36182308197021484
```

3.2.4 Závěr

Jelikož jde v této úloze jak o splnitelnost formule i o maximalizaci váhy splněných formulí, využil jsem různých přístupů jak tento faktor do přijímání zhoršujících řešení vztáhnout. Avšak druhá varianta prokázala, že zaměření se čistě na váhy nevede k splnitelnosti formule, tak dobře jak první varianta, u které nastává ekvilibrium, při němž se zůstává na nejvyšší splnitelnosti klauzulí, i lze pozorovat snížení výběru horších vah. V třetí variantě, jsem si chtěl ověřit, zdali zkombinování předešlých variant lze dospět k lepšímu balancování mezi splnitelností formule i maximalizaci vah. Nicméně se výsledek velice podobá druhé variantě se dvojnásobnou časovou náročností.

Proto budu dále v úloze využívat první variantu.

3.3 Třetí fáze

V této fázi se chci zaměřit na problém splnitelnosti i na dosažení optimální váhy. Z předešlých fází jsem zjistil, že ačkoliv najdu řešení, které splňuje klauzuli, není optimálním. Domnívám se, že by mohl být problém ve způsobu kterým generuji nové sousedy, a to že vždy měním pouze jeden bit. Rád bych zkusil různé přístupy.

Toto testování budu provádět na instancí, u kterých jsou dostupné optimální řešení, chci zjistit jak si vede splňování formule i hledání optimální váhy (maximalizace).

Využívám tedy instance ze sady wuf20-91-M.

Parametry SA zatím nastavuji adhoc, které se projevily být úspěšnými, abych mohl testovat různé způsoby implementace generování nových sousedů. V dalších fázích se zaměřím na faktorový návrh, u kterého se budu snažit o správné nastavení těchto hyper-parametrů SA, pro balancování explorace a exploatace.

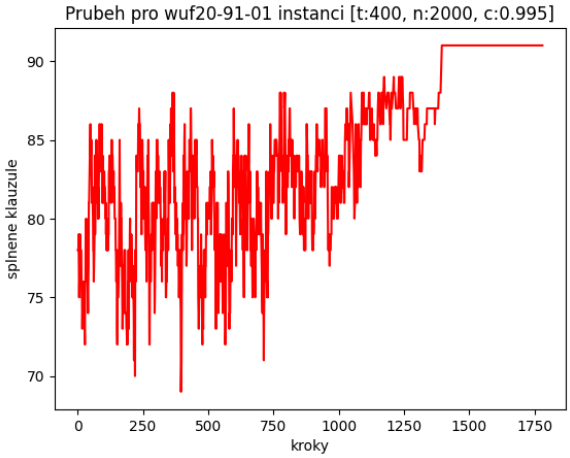
3.3.1 První varianta

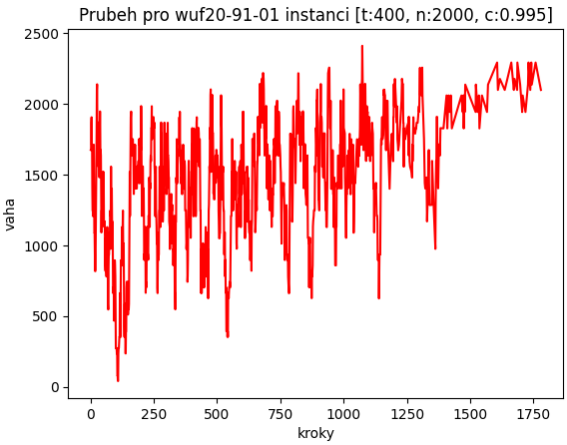
V první variantě využívám změnu jednoho bitu v ohodnocení proměnných.

Pro implementaci této logiky používám následující kód.

```
def get_random_neighbour(self, state):
    rand_index = random.randint(0, self.var_num - 1)
    state[rand_index] = (state[rand_index] + 1) % 2
    return state
```

Nejdříve se podívám na průběh jednoho z experimentů.





Nyní mě zajímá, kolik z 30 běhů SA s touto variantou generování sousedů skončilo úspěšným ohodnocením formule a jaká byla průměrná chyba vah.

splnitelnost	22/30
průměrná chyba	105.41
průměrný čas výpočtu	0.12 s
optimálních řešení	14/30

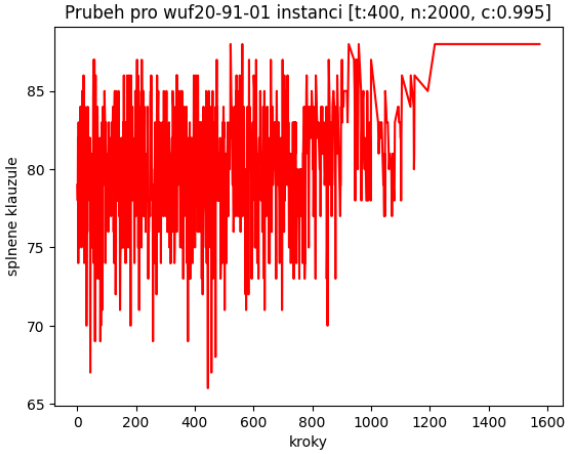
3.3.2 Druhá varianta

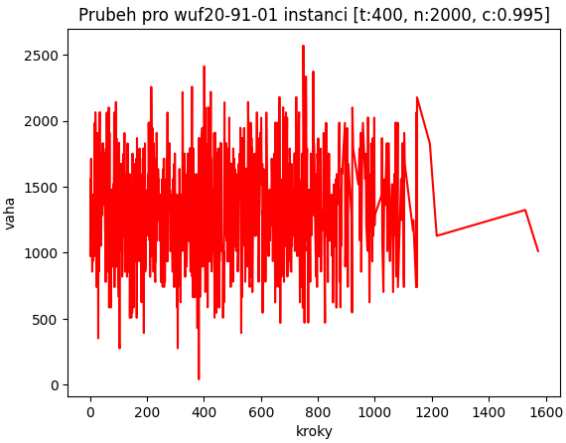
Ve druhé variantě se může změnit každé ohodnocení proměnných.

Pro implementaci této logiky jsem využil následující kód.

```
def get_random_neighbour_1(self, state):
    for i in range(len(state)):
      state[i] = (state[i] + random.randint(0, 1)) % 2
    return state
```

Nejdříve si zobrazím průběh jednoho experimentu.





Nyní mě zajímá, kolik z 30 běhů SA s touto variantou generování sousedů skončilo úspěšným ohodnocením formule a jaká byla průměrná chyba vah.

splnitelnost	1/30
průměrná chyba	156.0
průměrný čas výpočtu	0.09 s
optimálních řešení	0/30

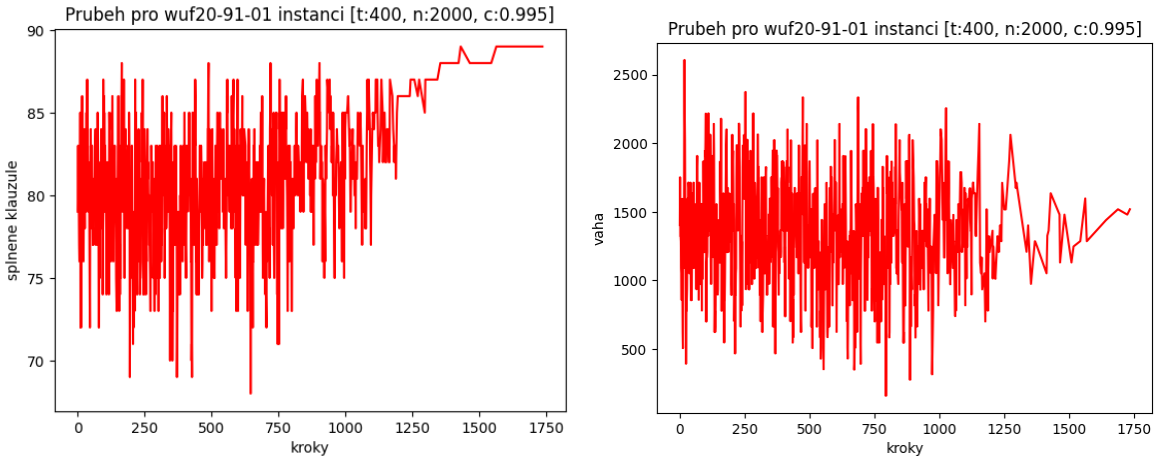
3.3.3 Třetí varianta

U třetí varianty zkusím namísto změny všech ohodnocení proměnných změnit pouze 1/4.

Pro implementaci této logiky jsem využil následující kód.

```
def get_random_neighbour_2(self, state):
    for _ in range(int(len(state)/4)):
      rand_index = random.randint(0, self.var_num - 1)
      state[rand_index] = (state[rand_index] + 1) % 2
    return state
```

Nejdříve si zobrazím průběh jednoho experimentu.



Nyní mě zajímá, kolik z 30 běhů SA s touto variantou generování sousedů skončilo úspěšným ohodnocením formule a jaká byla průměrná chyba vah.

splnitelnost	6/30
průměrná chyba	148.67
průměrný čas výpočtu	0.10 s
optimálních řešení	2/30

3.3.4 Čtvrtá varianta

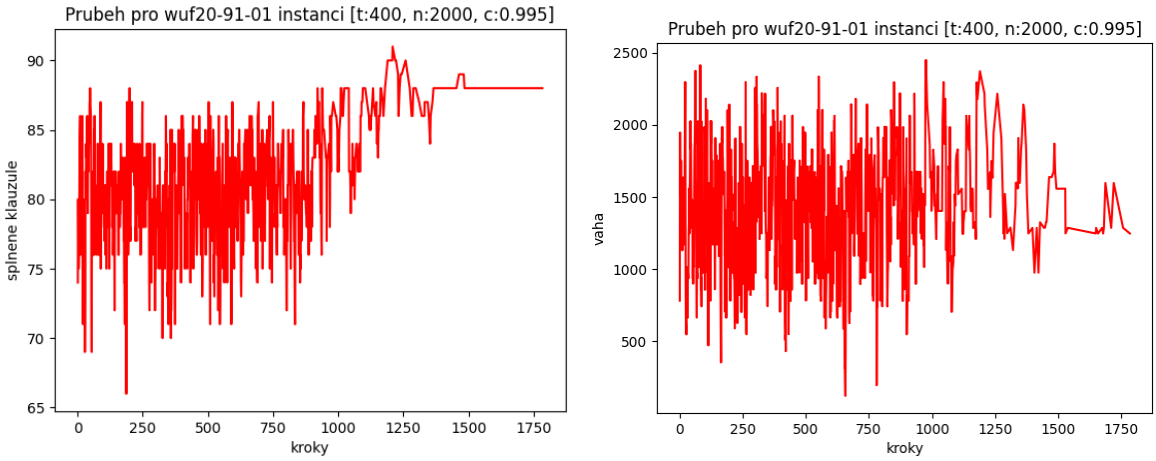
Jelikož předešlé dvě varianty sice neuspěly natolik, jako ta první, stále bych rád využil určitým způsobem obě, a nakombinoval je tak, aby při vyšší teplotě se spíš upřednostňovala změna více ohodnocení proměnných, a se snižující se postupně upřednostňovala změna jednoho bitu.

Nejprve jsem zkombinoval tedy první a druhou variantu.

Pro implementaci této logiky jsem využil následující kód.

```
def get_random_neighbour_3(self, state, t):
    one_bit_prob = math.exp(-t)

    if random.randint(0, 1) < one_bit_prob:
      return self.get_random_neighbour(state)
    else:
      return self.get_random_neighbour_1(state)
```

Nejdříve si zobrazím průběh jednoho experimentu.



Nyní mě zajímá, kolik z 30 běhů SA s touto variantou generování sousedů skončilo úspěšným ohodnocením formule a jaká byla průměrná chyba vah.

splnitelnost	18/30
průměrná chyba	73.11
průměrný čas výpočtu	0.13 s
optimálních řešení	13/30

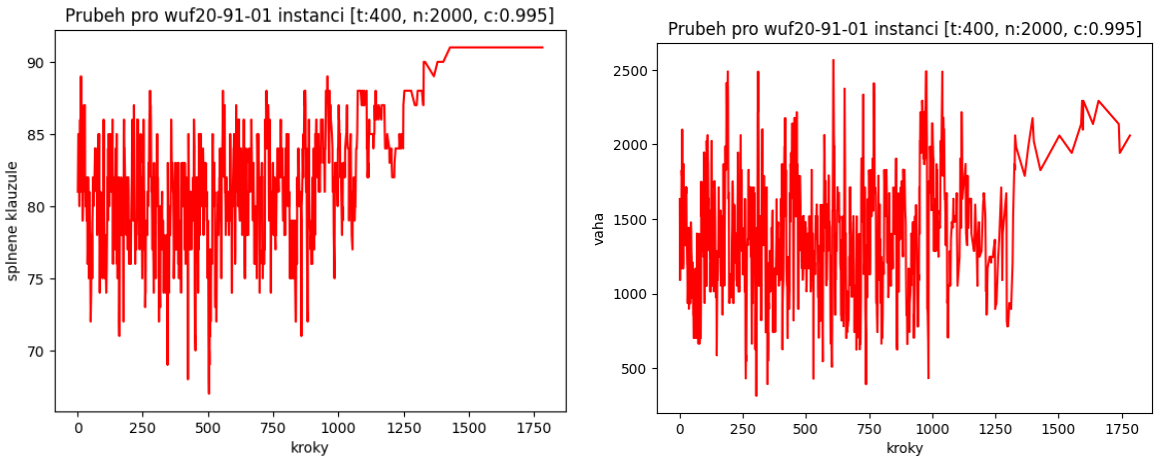
3.3.5 Pátá varianta

V poslední variantě zkouším nakombinovat první a třetí variantu, neboli změna jednoho bitu a změna 1/4 bitů.

Pro implementaci této logiky jsem využil následující kód.

```
def get_random_neighbour_3(self, state, t):
    one_bit_prob = math.exp(-t)

    if random.randint(0, 1) < one_bit_prob:
      return self.get_random_neighbour(state)
    else:
      return self.get_random_neighbour_2(state)
```

Nejdříve si zobrazím průběh jednoho experimentu.



Nyní si mě zajímá, kolik z 30 běhů SA s touto variantou generování sousedů skončilo úspěšným ohodnocením formule a jaká bych průměrná chyba vah.

splnitelnost	21/30
průměrná chyba	60.86
průměrný čas výpočtu	0.13 s
optimálních řešení	14/30

3.3.6 Závěr

Z naměřených experimentů je evidentní, že čistě náhodná změna více než jednoho bitu nevede k dobrým výsledkům. Nicméně právě možnost generovat ohodnocení s progresivním počtem změn, podle toho jak se teplota snižuje vede k lepším výsledkům, než samostatné měnění jednoho bitu.

Tento závěr lze pozorovat v porovnání páté varianty s první (čtvrtá varianta je ve stejném duchu, avšak není tak dobrá jako pátá). Oproti první pátá nalézá řešení, které v podobném počtu splňují celou formuli a zároveň snižuje průměrnou chybu ze 105.41 na 60.86.

Z pozorovaných výsledků experimentu tedy dále budu používat právě pátou variantu.

3.4 Čtvrtá fáze

V této fázi se chci zastavit u nastavování dvou vstupních hyper-parametrů tak, abych maximalizovat jednak úspěšnost splnění klauzule, tak i získání optimálního řešení.

Pro jednotlivá nastavení hyper-parametrů otestuji běh na několika instancích ze sady wuf75-325-M, kde jsou známi optimální řešení.

V první části faktorový návrh bude postaven na úspěšnosti splnění klauzulí a ve druhé pak počtu získaných optimálních řešení.

Hodnoty ve faktorovém návrh jsou voleny podle předchozích testovacích experimentů, u kterých nastavené hodnoty na [t: 400, cf: 0.995] vykazovaly hezké výsledky a průběh simulovaného ochlazování se podobal tomu, který jsme brali na přednášce. Proto jsem postavil faktorový návrh kolem těchto hodnot, menší než tyto hodnoty, a vyšší než tyto hodnoty.

Budu zkoumat několik faktorů kvality běhu, časovou náročnost, podíl nalezených optim, podíl splněných formulí, podíl splněných klauzulí, a v poslední řadě i průměrnou chybu vah.

Jelikož SA pracuje s randomizovaným počátečním ohodnocením i generováním sousedů, tak pro každou instanci spustím experiment s daným nastavením 10x, výsledky všech běhů s tímto nastavením napříč všemi 100 instancemi sady problémů wuf75-325N.

cooling factor		0.99	0.999	0.9999
init weight	300	0.000	0.104	0.425
	400	0.000	0.091	0.420
	500	0.000	0.094	0.433
	600	0.000	0.090	0.417

Table 1: Průměr splnitelnosti formulí

cooling factor		0.99	0.999	0.9999
init weight	300	0.974	0.994	0.998
	400	0.974	0.994	0.998
	500	0.974	0.994	0.998
	600	0.974	0.994	0.998

Table 2: Průměr splnitelnosti klauzulí

cooling factor		0.99	0.999	0.9999
init weight	300	0.000	0.006	0.107
	400	0.000	0.004	0.113
	500	0.000	0.003	0.125
	600	0.000	0.005	0.103

Table 3: Průměr nalezení optimální váhy

cooling factor		0.99	0.999	0.9999
init weight	300	-	1808.61	1083.02
	400	-	1729.65	1228.82
	500	-	1633.09	881.05
	600	-	1761.20	1108.99

Table 4: Průměrná chyba

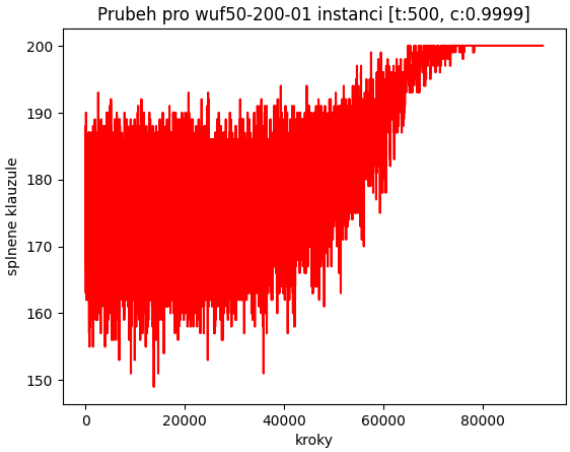
cooling factor		0.99	0.999	0.9999
init weight	300	0.14	1.50	14.86
	400	0.14	1.54	15.34
	500	0.15	1.58	15.67
	600	0.15	1.63	15.96

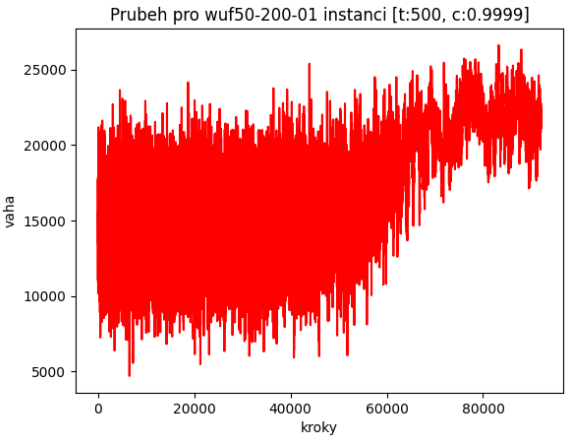
Table 5: Průměrná časová náročnost jednoho běhu experimentu

3.4.1 Závěr

Z pozorovaných dat by šlo usoudit, že nejlepším nastavením parametrů bude [t: 0.9999, iw: 500], avšak pro menší instance se úloha mění k nalezení řešení hrubou silou, počet prohledaných stavů se násobně zvyšuje, viz grafy běhů níže.

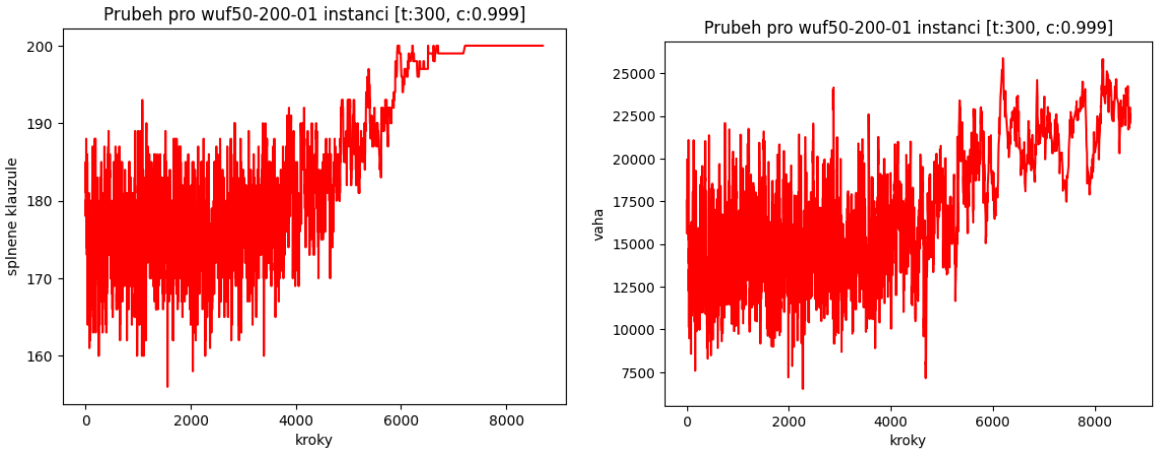
vysoká teplota a pomalé chlazení sice funguje u těžší instance, u jednodušších avšak představují neúměrné použití.





Z důvodu velké časové náročnosti (7s i na malé instanci) výpočtu, který toto nastavení vykazuje jsem se rozhodl, dle naměřených dat zvolit chladící faktor 0.999 s počáteční teplotou 300. Je to z důvodu, že dosahuje nejvyšší splnitelnosti formulí s tímto faktorem, nalezení optimální váhy taktéž, a oproti faktoru 0.9999 se splnitelnost klauzulí liší nepatrně, což se pokusím v další fázi vyřešit adaptačním mechanismem.

Následující grafy zobrazují změnu průběhu běhu s tímto nastavením.



Oproti předešlému nastavení trval výpočet zlomek času, okolo 1s, a dosáhl přitom podobných výsledků

3.5 Pátá fáze

V této fázi zkusím naimplementovat adaptační mechanismus, který u těžších instancí se pokusí nastavení SA tak, aby nalezl alespoň jedno ohodnoceni splňující formuli. Toto provedu pomocí resetu teploty, pokud SA nenašel žádné ohodnocení splňující formuli.

Nastavení maximálního možného počtu resetů zkusím experimentálně zjistit.

Podobně jako v předešlé fázi, pro každé nastavení tohoto parametru spustím 10 opakování na sto instancích sady wuf75-325N, u které nebyla z počtu splněných formulí má implementace dvakrát úspěšná. Pro každou počet resetu se tedy provedlo 1000 výpočtů.

Sleduji poměr splněných formulí ku všem, poměr splněných klauzulí ku všem, poměr nalezených optim a průměrnou časovou náročnost jednoho běhu heuristiky.

		sat formulas	sat clauses	optimums	weight diff	duration [s]
temp reset	0	0.104	0.994	0.006	0.007	1.50
	1	0.147	0.995	0.004	0.012	2.79
	2	0.203	0.996	0.006	0.013	3.99
	3	0.259	0.997	0.013	0.019	5.08
	4	0.312	0.997	0.018	0.021	6.18

Table 6: Nastavení resetů

3.6 Závěr

Z naměřených dat lze pozorovat lineární nárůst měřených hodnot, tedy se zvyšujícím se opakování, roste průměr splněných klauzulí, splněných formulí, roste průměrná chyba (více splněných formulí). Jelikož se zvyšujícím počtem resetů roste i časová náročnost výpočtu, a zlepšení je minimální, tak využiji dále hodnotu 2.

4 Black box fáze

Po nastavování hyper-parameterů, hledání nejlepších logik pro implementaci SA bych rád v této fázi otestoval kvalitu implementovaného řešení. Nyní by mě zajímalo, jak moc si mé řešení vede na instancích, u kterých jsou k dispozici optimální váhy a jak stabilní toto nalézání optimálního řešení je.

4.1 Stabilita běhu

Z osobních experimentů jsem si všiml, že mé řešení u těžších instancí ne vždy nalezne ohodnocení, které by splňovalo formuli, a u lehčích, že ne vždy je optimálním, toto může být způsobeno náhodnou volbou počátečního řešení i náhodnou volbou nových ohodnocení. Proto bych nyní rád spustil 100x první instanci z každé sady, kterou jsem doposud nevyužil.

Sleduji trojici metrik, a to poměr úspěšných formulí ku všem, poměr splněných formulí ku všem a průměrnou chybu vah u splněných formulí.

		sat formulas	sat clauses	optimums	weight diff	duration [s]
instances	wuf20-71N-01	1.0	1.0	0.88	0.010	0.61
	wuf20-71Q-01	1.0	1.0	0.26	0.520	0.62
	wuf50-218N-01	0.37	0.99	0.25	0.002	2.87
	wuf50-218Q-01	0.37	0.99	0.20	0.040	3.04

Table 7: Stabilita běhu

4.2 Hloubkový experiment

Pro hloubkový experiment nejdříve otestuji jak si vede má implementace SA na instancích pro které je známo optimální řešení. Z každé sady instancí jsem vybral prvních 3 instance, nad kterými z důvodu náhodné generaci počátečního řešení i nových ohodnocení zopakuji výpočet 10x. Budu využívat dvě indikace kvality dosažených výsledků, prvním je poměr splněných klauzulí a druhým průměrná chyba nalezených vah.

Sleduji trojici metrik, poměr splněných klauzulí ku všem a průměrnou chybu vah u splněných formulí.

		sat clauses	weight error	duration [s]
instances	wuf20-71N	1.0	0.030	0.53
	wuf20-71M	1.0	0.040	0.48
	wuf20-71Q	1.0	0.099	0.47
	wuf20-71R	1.0	0.091	0.49
	wuf20-91N	1.0	0.007	0.66
	wuf20-91M	1.0	0.001	0.62
	wuf20-91Q	0.99	0.163	0.65
	wuf20-91R	0.99	0.172	0.76
	wu50-218N	0.99	0.016	2.66
	wuf50-218M	0.99	0.010	2.28
	wuf50-218Q	0.99	0.050	2.52
	wuf50-218R	0.99	0.066	2.61

Table 8: Hloubkový experiment

Nakonec bych rád své řešení ještě ověřil na těžších instancích sady wuf100-430, kde budu sledovat splnitelnost klauzulí a čas potřebný k výpočtu. Opět z každé sady vybírám trojici instancí, u kterých 10x opakuji výpočet.

Sleduju poměr splněných klauzulí ku všem.

		sat clauses	duration [s]
instances	wuf100-430N	0.99	5.36
	wuf100-430M	0.99	4.54
	wuf100-430Q	0.99	4.48
	wuf100-430R	0.99	4.51

Table 9: Experiment na těžkých instancích

4.2.1 Závěr

Z naměřených dat lze pozorovat, že si má implementace vede dobře na menších a nezavádějících instancích. To lze pozorovat z nalezeného ohodnocení splňující formuli ve všech bězích experimentu s podstatně malou chybou vah. U zavádějících a těžších instancí už se ne vždy nalezlo ohodnocení splňující formuli, ač splnitelnost klauzulí je stále velice vysoká. Má implementace se snaží balancovat mezi splnitelností a váhou, kde lze pozorovat, že není vždy vhodně vybalancováno. U posledního experimentu na nejtěžších instancích lze opět pozorovat vysoký procento splnění klauzulí, avšak úplná splnitelnost formulí ve většině bězích nenastala.

5 Závěr

Se samostatným spuštěním první implementace jsem měl řadu problémů, a to z hlediska velikostí výpočetní složitosti. Nepodařilo se mi získat vhodná řešení ani pro instance lehčích sad. Tento problém jsem se snažil v práci vyřešit představením nové logiky a prozkoumáváním optimalizační konstanty (počtu splněných klauzulí) s ohledem na vážené ohodnocení formulí, což jsem zkoumal ve druhé fázi.

Ve třetí fázi jsem se zaměřil na možnosti generování nových řešení a jak se při opakovaných pokusech mění nalezené ohodnocení pro splnění lehčích instancí. Zde se projevil balanc mezi změnou více bitů a jednoho, v závislosti na velikosti teploty, jako nejlepší. Avšak stále se nedařilo minimalizovat počet nesplněných formulí.

V čtvrté fázi jsem se snažil o faktorový návrh pro těžší instance, okolo hodnot, které vykazovaly kvalitní výsledky. Jako nejlepší z nich vzešel, avšak byl příliš nevhodný pro menší instance a časově náročný. S tím jsem byl nespokojen a zvolil jsem další vhodný přístup, který dosahoval akceptovatelných výsledků.

Nakonec ve páté fázi jsem se pokusil implementovat adaptační mechanismus pro případný reset a nalezení lepšího ohodnocení. S rostoucí hodnotou parametru vzrůstala časová náročnost a zlepšení bylo minimální.

V black box fázi jsem se pak snažil ověřit kvalitu nalezeného ohodnocení pomocí simulovaného žíhání na sadách, u kterých jsou známa optimální řešení. Sledoval jsem splnitelnost a další informace o běhu.

Ve výsledku bych své řešení oproti první nezdařené implementaci hodnotil jako uspokojivé, avšak s mnohem prostoru na zlepšení. Při případném zlepšení bych věnoval více času implementaci cenového skóre, které by lépe balancovalo váhu a splnitelnost formulí (což bylo v této práci více preferované). Má idea byla od začátku taková, že je nejdříve potřeba nalézt řešení splňující formuli, a až pak se zaměřit na samotnou váhu. To však přineslo horší splnitelnost formulí u těžších nebo zavádějících instancí, což jsem nepozoroval dříve.

References

[1] David Omrai. Evaluace frameworku seage. Bachelor's thesis.

[2] Jan Schmidt. Data a programy. https://courses.fit.cvut.cz/NI-KOP/download/index.html#mwsatinst. Accessed: 2024-01-06.