Nasazení pokročilé heuristiky

NI-KOP // Ondřej Wrzecionko

13.12.2022

Zadání

Vyřešte problém maximální vážené splnitelnosti booleovské formule (Max Weighted SAT) pokročilou iterativní metodou. Problém řešte některou z pokročilých heuristik (**simulované ochlazování**, genetický algoritmus). Heuristika musí zvládat instance o velikosti minimálně 20 až 50 proměnných v celém stanoveném rozsahu. Základní kód musí být vypracován samostatně, v libovolném programovacím jazyce. Po nasazení heuristiky ověřte její vlastnosti experimentálním vyhodnocením a přesvědčivě doložte, jaký rozsah instancí heuristika zpracovává.

Zpráva musí dokládat racionální přístup k řešení problému, od nastavení **(white box fáze)** po závěrečné vyhodnocení **(black box fáze)**. V popisu nastavení uvádějte i slepé uličky, neúspěšné pokusy. V popisu black box fáze uveďte, proč pokládáte výsledky za průkazné.

Abstrakt

V 2. domácím úkolu jsem se zabýval tvorbou programu implementujícího algoritmus **simulovaného ochlazování** pro problém maximální vážené splnitelnosti booleovské formule. Nejprve jsem provedl nastavení heuristiky ve třech iteracích white-box fáze, následně jsem nastavení experimentálně ověřil v black-box fázi. Výsledná heuristika dosáhla úspěšnosti 60-85 % pro rovnoměrné rozložení vah a 35-80 % pro zavádějící rozložení dat.

1 Úvod

Mým cílem je napsat **algoritmus**, který s pomocí metody simulovaného ochlazování vyřeší obtížné instance max weighted SAT problému v rozsahu alespoň 20-50 proměnných, vhodně navrhnout jeho parametry a následně ho experimentálně vyzkoušet.

Obtížnost instance problému max weighted SAT vychází z obtížnosti původní instance, jelikož v podstatě řešíme dva problémy – výsledné ohodnocení musí být řešením "přidruženého" SAT problému, a zároveň musí mít **optimální** váhu.

Stejně jako u SAT instancí, i zde platí, že za obtížné instance budeme považovat ty, kde je poměr počtu klauzulí ku počtu proměnným přibližně 4.3. Navíc nám zde ale bude obtížnost určovat ještě rozložení vah – pokud bude takové, že proměnné, které ve výsledku **nesmí** být ohodnoceny 1 budou mít **největší** váhu, bude to značně ztěžovat průchod stavovým prostorem.

2 Materiál

2.1 Instance

Pro testování programu jsem využil instance ze sad wuf20-91, wuf50-218 a wuf75-325 dostupných na stránkách předmětu NI-KOP.

Každá ze sad obsahuje **čtyři** varianty: u variant M/N byly váhy jednotlivých proměnných generovány **rovnoměrně**, nalezení řešení by tedy mělo být při správně zvolených parametrech bez problémů. Varianty Q/R mají úmyslně generované **větší** váhy u proměnných, které **nemůžou** být ohodnoceny 1, a vytvářejí tak tedy **zavádějící** úlohu.

2.2 Vstup a výstup

Program přijímá vstup ve formátu **MWCNF**. Řádky začínající písmenem **c** obsahují komentáře, **p** definici problému (mwcnf počet proměnných, počet klauzulí), **w** váhy jednotlivých proměnných. Následují definice jednotlivých klauzulí.

Výstupem programu na **standardním** vstupu je řešení ve formátu **MWCNF**, tedy nejprve jméno instance, pak dosažená váha řešení a řešení samotné. Na **chybový** výstup se vypíše počet iterací, číslo iterace, na které nastalo poslední zlepšení, počet splněných klauzulí, splněných klauzulí celkem, počet splněných klauzulí v původním řešení a dosažená váha.

```
wuf20-08.mwcnf 442 -1 -2 3 4 -5 6 -7 -8 9 -10 11 -12 13 -14 -15
-16 -17 -18 -19 -20 0
20540 10537 91 91 86 442
```

Listing 1: Ukázka výstupu programu pro instanci `wuf20-08.mwcnf` ze sady `wuf20-91-Q`

2.3 Algoritmus

Při řešení problému jsem vycházel z metody **simulovaného ochlazování** *(simulated annealing)*.

```
state = randomState()
best = null
while (!frozenSA()) {
    repeat(N) {
        state = trySA(state)
        if (state.better(best))
            best = state
    }
    coolSA()
}
```

Listing 2: Simulované ochlazování v pseudokódu

Funkce `randomState` vygeneruje počáteční stav pro simulované ochlazování. V mém algoritmu je tento stav **vygenerován** jako **náhodný** vektor 0 (false) a 1 (true).

Funkce `frozenSA` určuje, kdy se algoritmus simulovaného ochlazování zastaví. Ve svém algoritmu jsem implementoval dva způsoby zastavení: dosažením **cílové teploty** T_{fin} nebo maximálním počtem iterací beze změny (`stopIterations`).

Parametr N neboli **délka ekvilibria** určuje, kolikrát se bude opakovat vnitřní smyčka programu *(kolikrát zkusíme najít řešení problému při stejné teplotě T)*.

Funkce `coolSA` sníží teplotu. Tuto funkci jsem implementoval jako pouhé násobení chladícím koeficientem `alpha`.

```
val toFlip = 1 + Random.nextInt(instance.varCount)
val newState = flip(state, toFlip)
if (newState.cost > state.cost)
    return newState

val acceptThoughWorse = Random.nextDouble() < exp((newState.cost
- state.cost) / temperature)
return if (acceptThoughWorse) newState else state
```

Listing 3: Kód funkce `trySA`

Funkce `trySA` vybere náhodného souseda z **okolí** aktuálního stavu. Jelikož jsem se rozhodl implementovat stavový prostor s **operátorem** negace jedné proměnné, tato funkce provede **negaci** náhodně vybrané proměnné.

Pokud je **cena** řešení po negaci lepší, řešení se automaticky **přijme**. Pokud je cena **horší**, řešení se přijme s pravděpodobností $e^{\frac{cost(new)-cost(old)}{T}}$.

Jako **cenovou funkci** jsem implementoval součin **váhy** daného řešení a poměru počet splněných klauzulí ku počtu klauzulí celkem.

```
weight(values) * (satisfied(values) / instance.clauseCount)
```

Listing 4: Výpočet ceny daného řešení

Při **testování** algoritmu jsem si ale všiml, že se v polovině případů algoritmus **zasekne** v lokálním maximu, které má velmi **podobnou** cenu jako optimální řešení.

```
var solution = Solver(instance, configuration).solveInstanceSA()
repeat(configuration.maxTries) {
    if (solution.satisfiedClauses == solution.totalClauses)
        return solution // solution found, end
    solution = Solver(instance, configuration).solveInstanceSA()
}
```

Listing 5: Wrapper algoritmu simulovaného ochlazování

Celý algoritmus simulovaného ochlazování jsem tedy obalil do wrapperu, který algoritmus spustí nejvýše `maxTries` krát a zastaví se, jakmile najde řešení.

3 White-box fáze

3.1 Parametry

Algoritmus, který používám k řešení problému, obsahuje následující **parametry**: počáteční teplota, délka ekvilibria, koeficient chlazení, koncová teplota, počet iterací bez zlepšení, po kterých se program zastaví a maximální počet spuštění algoritmu (maxTries).

Počáteční a koncová **teplota** určují, s jakou pravděpodobností se v jednotlivých krocích budou **přijímat** řešení s **horší** cenou. Hodnoty teploty by měly být voleny v **závislosti** na možném rozsahu ceny, což v aktuálním případě odpovídá **rozsahu** 0 až $\sum w_i$. V komentáři k datovým sadám se můžeme dozvědět, že váhy byly generovány v rozsahu 100 až 1500, **průměrná** váha tedy bude $800 \cdot n_v$, kde n_v je počet proměnných.

Délka ekvilibria určuje, jak dlouho se bude algoritmus pokoušet hledat řešení při jedné teplotě. Pro tento parametr můžeme volit typickou hodnotu 100.

Koeficient ochlazování určuje, jak rychle bude probíhat ochlazování. Při příliš prudkém ochlazování může dojít k příliš rychlé konvergenci a **uváznutí** v lokálním maximu, zatímco při příliš pomalém ochlazování může dojít k divergenci, kdy program nedojde v daném počtu kroků k řešení. Hodnoty tohoto parametru se typicky pohybují mezi 0.8 *(včetně)* a 1 *(už ne)*.

3.2 Faktorový návrh na instanci malé velikosti

Nejprve jsem provedl **faktorový návrh** na jedné instanci, a to konkrétně `wuf20-91-M/wuf20-103.mwcnf`. Volil jsem pevně parametry `stopIterations` 10 000, délku ekvilibria 100, `maxTries` 10 a faktorový návrh jsem prováděl na **počáteční teplotě** a **koeficientu ochlazování**.

	α=0.8	α=0.9	α=0.95
T=1000	499 (12 024)	500 (14 303)	499 (19 023)
T=4000	500 (12 652)	500 (15 628)	499 (21 710)
T=8000	500 (12 949)	500 (16 274)	500 (23 021)

Table 1: Faktorový návrh na instanci `wuf20-103.mwcnf` *(počet úspěšných běhů / 500 spuštění, průměrný počet iterací)*

Z faktorového návrhu na této instanci vyplývá, že je v podstatě jedno, které parametry volím, program i tak **téměř vždy** nalezne optimální řešení. Vzhledem k tomu, že se jedná o instanci s 20 proměnnými, tedy takovou, kterou bych zvládl řešit i hrubou silou, nejedná se o nic zvláštního.

3.3 Faktorový návrh na instanci větší velikosti

Jako druhý krok jsem udělal faktorový návrh na instanci wuf50-218-N/wuf50-0108.mwcnf, se stejně volenými parametry jako v případě první instance.

	α=0.8	α=0.9	α=0.95
T=1000	1 (12 314)	12 (14 706)	32 (18 881)
T=4000	4 (12 750)	27 (15 879)	48 (21 519)
T=8000	6 (13 135)	17 (16 696)	55 (23 508)

Table 2: Faktorový návrh na instanci `wuf50-0108.mwcnf` *(počet úspěšných běhů / 500 spuštění, průměrný počet iterací)*

Z faktorového návrhu můžeme vidět, že heuristika s takto nastavenými parametry **nefunguje** správně na větších instancích. I s nastavením **koeficientu ochlazování** na ještě větší hodnotu (0.99) je stále úspěšných jen přibližně 172 z 500 instancí.

3.4 Vylepšení algoritmu

Nejprve jsem se pokoušel zvýšit délku ekvilibria, případně zvětšit počet iterací, po kterých se výpočet zastaví. V ani jednom případě se mi ale nepodařilo dospět k lepší než 50 % úspěšnosti na instancích velikosti 50 s **rovnoměrným** rozložením. U instancí se zavádějícím rozložením vah algoritmus nedospěl ke správnému výsledku **ani jednou**.

Při analýze **nesprávných** výsledků, ke kterým algoritmus dospěl jsem zjistil, že je problém v tom, že algoritmus **neumožňuje** nastavit rozdílnou důležitost váhám a počtu splněných klauzulí při výpočtu **ceny**.

Změnil jsem tedy výpočet cenové funkce, a to na $\sum w_i \cdot W + satisfied$, kde W je konstanta `weightPenalty`, která **znevýhodňuje** váhy. Tuto konstantu je nutno volit dost malou na to, aby řešení **konvergovalo**, ale také dost **velkou** na to, aby váha řešení **nebyla** zanedbávána.

3.5 Faktorový návrh na instanci větší velikosti (2)

Faktorový návrh jsem **zopakoval** na instanci větší velikosti. Na základě **předchozího** faktorového návrhu jsem nechal `stopIterations` na 10 000, délku ekvilibria na 100, `maxTries` na 10.

Počáteční teplotu jsem zvolil **4 000**, jelikož z předchozích návrhů vycházela nejlépe *(i tak ale na počáteční teplotě příliš nezáviselo)*. **Rozsah** koeficientu ochlazování jsem změnil na 0.9, 0.95 a 0.99 a jako druhý parametr jsem volil konstantu `weightPenalty` (W) 0.000001, 0.0005 a 0.0001.

	α=0.9	α=0.95	α=0.99
W=0.0005	414 (21 210)	343 (35 012)	241 (153 285)
W=0.0001	248 (21 717)	220 (34 475)	261 (135 434)
W=0.000001	138 (25 261)	126 (41 616)	201 (174 304)

Table 3: Faktorový návrh na instanci `wuf50-0108.mwcnf` *(počet optimálních běhů / 500 spuštění, průměrný počet iterací)*

Můžeme vidět, že tento algoritmus již dává mnohem **uspokojivější** výsledky. Při menší (0.0001) nebo příliš velké (0.000001) penalizaci vah je lepší volit **větší** koeficient chlazení, zajímavé je ale, že nejlepších výsledků dosahujeme pro penalizaci vah 0.0005 při **nejmenším** koeficientu chlazení. To může být tím, že při této penalizaci vah jsou vyšší hodnoty koeficientu chlazení příliš prudké, a dochází tedy k diverzifikaci.

Důležité je také podotknout, že zatímco při původní cenové funkci byl nízký počet optimálních běhů způsoben **nesprávnými výsledky**, v tomto případě byl způsoben běhy, které **našly** řešení, které akorát nemělo nejlepší hodnotu optimalizačního kritéria.

Pro kontrolu jsem se pokusil nejlepší konfiguraci, tedy $\alpha = 0.9$, W $= 0.0005$ ještě pozměnit $\rightarrow$ snížit teplotu, nebo snížit penalizaci vah. Po snížení koeficientu chlazení na **0.8** program doběhl úspěšně v **400** běžích, tedy **méně**. Po lehkém zvýšení penalizace vah na 0.001 doběhlo úspěšně **223** běhů, tedy **méně**.

Na základě faktorového návrhu pro novou cenovou funkci tedy vychází nejlépe $\alpha = 0.9, W = 0.0005, T = 4000$.

3.6 Ověření faktorového návrhu na sadě instancí

Parametry, které mi vyšly z faktorového návrhu na **jedné instanci** jsem ověřil na **celé sadě** instancí `wuf50-0108-N`. Zjistil jsem ale, že program instance z této sady často dosáhne optimálního řešení jen v několika desítkách procent případů. Program jsem tedy **potřetí** upravil, a to tak, že se jako **výsledné** řešení určí **nejlepší** z `maxRuns` (nový parametr) běhů *(na každý běh zůstává `maxTries` pokusů)*.

Tento upravený program již našel **optimální řešení** u **834** instancí z 1000, ve zbylých případech se řešení programu ve většině případů jen **lehce** lišilo od **optimálního**. Jedná se tedy o úspěšnost 83.4 %.

Faktorový návrh jsem se rozhodl tedy zpětně ověřit ještě na sadě `wuf20-91-M`. Program zde nalezl řešení pro 980 z 1000 instancí, jedná se tedy o **98** % úspěšnost.

```
var solution = Solver(instance, configuration).solveInstanceSA()
var best: Solution? = null
repeat(configuration.maxRuns) {
    repeatUntil(configuration.maxTries - 1) {
        if (solution.satisfiedClauses == solution.totalClauses) {
            if (solution.solutionWeight > (best?.solutionWeight
            ?: 0))
                best = solution
            return@repeatUntil
        }
        solution = Solver(instance,
        configuration).solveInstanceSA()
    }
}
return best ?: solution
```

Listing 6: Nová verze wrapperu algoritmu simulovaného ochlazování

3.7 Ověření faktorového návrhu na zavádějících instancích

Faktorový návrh jsem se rozhodl ověřit tedy ještě na zavádějících instancích, nejprve ze sady `wuf20-91-Q`. Na této sadě program nalezl řešení pro 941/1000 instancí, měl tedy **94** % úspěšnost.

Další ověření proběhlo na sadě `wuf50-218-R`, která by měla být z testovaných sad nejtěžší. Na této sadě program již **nebyl** schopný najít řešení, jelikož musí mít generované řešení ještě větší penalizaci vah. Provedl jsem tedy faktorový návrh pro instanci `wuf50-05.mwcnf` ze sady `wuf50-218-R`:

	α=0.9	α=0.95	α=0.99
W=0.00001	17 (22 140)	31 (34 752)	48 (151 182)
W=0.000025	22 (23 055)	87 (35 210)	133 (133 240)
W=0.000001	10 (24 920)	13 (40 820)	21 (160 250)

Table 4: Faktorový návrh na instanci `wuf50-05.mwcnf` *(počet optimálních běhů / 500 spuštění, průměrný počet iterací)*

Nejlepších výsledků jsem dosáhl pro parametry $\alpha = 0.99$, W $= 0.000025$, **ověřil** jsem tedy tyto parametry na celé sadě `wuf50-218-R` s úspěšností na 641 z 1000 instancí – **64** %. Tyto parametry jsem následně ověřil i na **dalších** sadách (úspěšnost zůstala stejná, jako s parametry $\alpha = 0.9, W = 0.00005$).

4 Black-box fáze

4.1 Parametry

Na základě faktorového návrhu jsem **třikrát** provedl úpravu algoritmu s tím, že jsem postupně pro každou verzi provedl vždy nový faktorový návrh a ověřil **funkčnost** na každém typu dat.

Nyní je tedy čas algoritmus "zakrabičkovat" (nastavit mu **pevně** parametry) tak, aby klient vůbec nemusel vědět, že program nějaké parametry potřebuje a mohli jsme mu dodat out-of-the-box **funkční** program.

Z faktorového návrhu po ověření vyšly jako **nejlepší** následující parametry:

- počáteční teplota $T_0 = 4000$
- koeficient ochlazování $\alpha = 0.99$
- penalizace vah v cenové funkci $W = 0.000025$
- délka ekvilibria $N = 100$
- maximální počet pokusů `maxTries` $= 10$
- maximální počet běhů `maxRuns` $= 5$
- zastavení na základě počtu iterací bez zlepšení `stopIterations` $= 10000$

4.2 Experimenty

S danými parametry jsem provedl spuštění na následujících zkrácených sadách, dostupných z Courses: `wuf20-91R`, `wuf50-218R` a `wuf75-325R` (vždy na jejich variantách M, N, Q, R) s výsledky, které jsou dostupné níže.

Z daných běhů lze vidět, že program našel řešení u rovnoměrného rozložení vah vždy v alespoň **60 %** případů, což je vzhledem k obtížnosti daného programu **úspěch**.

Také lze vidět, že je program schopen pracovat na instancích menších (20 proměnných) i větších (50, 75 proměnných), a to s úspěšností mezi **35-85 %** podle velikosti a tom, jestli je daná instance zavádějící.

Pro datovou sadu `wuf100-430` experimenty byly provedeny, jelikož zde ale **neexistuje** soubor s referenčními optimálními vahami *(jedná se o problémy s **hodně** řešeními, není snadné najít všechna řešení a zjistit, která váha je optimální)*, nelze usuzovat, zda se jednalo o optimální řešení, nebo ne.

sada	úspěšných běhů	běhů celkem	úspěšnost
wuf20-91R-M	84	100	84 %
wuf20-91R-N	84	100	84 %
wuf20-91R-Q	80	100	80 %
wuf20-91R-R	82	100	82 %
wuf50-218R-M	68	100	68 %
wuf50-218R-N	71	100	71 %
wuf50-218R-Q	54	100	54 %
wuf50-218R-R	41	100	41 %
wuf75-325-M	60	100	60 %
wuf75-325-N	67	100	67 %
wuf75-325-Q	37	100	37 %
wuf75-325-R	35	100	35 %

Table 5: Výsledky běhu programu na zkrácených datových sadách

```
for folder in blackbox/*
do
  for dataset in "$folder"/*[^.dat]
  do
    rm -f out.txt
    for file in "$dataset"/*
    do
      inst=${file#"$dataset/"}; inst=${inst%".mwcnf"};
      inst=${inst//"wuf"/"uf"}; printf "$inst " >> out.txt
      java -jar sawsat3.jar < "$file" 1>>out.txt 2>/dev/null
    done
    sort out.txt > "$dataset-prg.dat"
    df=$(diff -y --suppress-common-lines "$dataset-prg.dat"
    "$dataset-opt.dat" | sort | wc -l)
    echo "Different: $df / 100"
  done
done
```

Listing 7: Skript pro spouštění programu na jednotlivých datových sadách

5 Diskuze

5.1 Algoritmus

Nejprve jsem naimplementoval **čistý** algoritmus, ten měl ale velmi nízkou úspěšnost (v jednotkách procent) nezávisle na zvolených parametrech *(teplota, koeficient chlazení, délka ekvilibria, podmínka pro zastavení)*.

Po změně **cenové funkce** a přidání nového parametru **penalizace vah** se úspěšnost algoritmu zvýšila na nízké desítky procent (opět nezávisle na parametrech). Až po přidání **wrapperu**, který algoritmus spustí vícekrát a vrátí nejlepší výsledek, se **zlepšila** úspěšnost na výsledných 35 až 85 procent *(podle velikosti instance a rozložení vah)*.

Jelikož jsem implementoval algoritmus **simulovaného ochlazování** správně, tipuji, že problém je v návrhu stavového prostoru a funkci **trySA**, která dalšího souseda volí pouze flipnutím jedné proměnné. Zvláště u instancí s **velkým** počtem proměnným tak není pohyb stavovým prostorem dostatečně rychlý a rozsáhlý, což vede k takto nízké úspěšnosti.

5.2 Doba běhu a úspěšnost

Po nalezení nejvhodnějších parametrů ve faktorovém návrhu jsem provedl ještě experimentální vyhodnocení na zkrácených datových sadách, které ověřilo, že je úspěšnost opravdu **taková**.

Je ale nutné podotknout, že úspěšnost je měřena na základě počtu nalezení **optimálních** řešení – pokud měl problém 2 řešení s vahami 100 a 101, nalezení řešení s váhou 100 v 6 případech a s váhou 101 v 4 případech by bylo považováno za úspěšnost 40 %, byť je prakticky algoritmus **úspěšnější**, proto zde nižší úspěšnost úplně nevadí.

Algoritmus běží, zvláště pro instance větší velikosti delší dobu, než na kratších. Čas běhu se měří **obtížněji**, jelikož je závislý na přístroji, na kterém bude algoritmus spuštěn, počet instancí, který závisí na zvolené teplotě, koeficientu ochlazování a počtu opakování ale **vede** k miliardám iterací, proto na jedné instanci běží program nízké jednotky sekund.

Řešením by mohlo být nastavení **nižší** teploty a koeficientu chlazení pro **menší** instance (jak si lze povšimnout z prvních faktorových návrhů, na menších instancích stačil například i koeficient ochlazování $\alpha = 0.8$), případně **variabilní** nastavování parametrů na základě velikosti instance.

Figure 1: Grafy úspěšností pro jednotlivé sady (úspěšnost na počtu instancí) a celkový graf úspěšností

5.3 Randomizovanost

Poslední věc, kterou chci **diskutovat** je fakt, že algoritmus je **randomizovaný**, je tedy dostatečně reprezentativní vzorek 100 běhů? Na následujícím grafu je vidět pro jednotlivé sady **vývoj** úspěšnosti od první po poslední běh.

Jak lze vidět, grafy úspěšnosti pro jednotlivé sady se ustálí již po přibližně **40** instancích, jedná se tedy o dostatečně **reprezentativní** vzorek.

5.4 Závěr

V 2. domácím úkolu jsem provedl **nasazení** pokročilé heuristiky, od prvotního programu implementujícího algoritmus simulovaného ochlazování, přes **nastavení** (white-box fázi), až po experimentální **otestování** (black-box fázi). Vzniklá heuristika je schopna pracovat na instancích v **rozsahu** 20 až 75 proměnných s **úspěšností** od 35 do 85 procent.