Řešení problému maximální vážené splnitelnosti (MWSAT)

Abstrakt

Tento 'report' se zabývá řešením problému maximální vážené splnitelnosti booleovské formule (MWSAT). Jakožto aplikovanou heuristiku jsem zvolil simulované ochlazování.

Obsah

1 Úvod

1.1 Definice Problému & Popis Úkolu

Zadání vyžaduje návrh 'přijatelného' řešení problému MWSAT, v rámci kterého se snažíme najít platné ohodnocení vážených proměnných řešících sadu booleovských klauzulí, přičemž chceme sumu vah *ne-negovaných* proměnných (tj. `x = True`) optimalizovat. Definice problému dává na výběr mezi extrémy, zvolil jsem tedy cílit na maximalizaci.

Na řešení máme aplikovat některou pokročilou heuristiku – v případě této práce simulované chlazení (*simulated annealing)*. Proces je rozdělen na 2 fáze:

- Návrh a nastavení (*White box* fáze)
- Měření a vyhodnocení (*Black box* fáze)

Ve White box fázi se snažíme navrhnout řešení a poté se nějakým logickým iterativním procesem dopracovat k (pokud možno) ideální počáteční konfiguraci parametrů. Black box fáze je měření a vyhodnocení výsledků z celého vzorku použitých dat.

1.2 Použitá data

Finální měření bylo provedeno na následujících setech (z Courses):

- `wuf20-91-[MNQR]`
- `wuf50-218-[MNQR]`
- `wuf75-325-[MNQR]`

Jména sad mají formát '`wuf[VARS]-[CLAUSES]-[DF]`', kde `VARS` značí počet proměnných, `CLAUSES` počet klauzulí v problému, a `DF` vyjadřuje *složitost* dané sady. Problémy v `M` a `N` by měly být snadnější, zatímco `Q` a `R` by měly být těžké až *klamné* v kontextu optimalizace výsledné váhy.

Zmíněné sady by měly dostatečně pokrýt jak velikostní, tak složitostní škálu vyžadovanou pro řádné zhodnocení řešení.

1.3 Použité metriky

Pro interpretaci a vizualizaci dat jsou využity převážně tyto metriky:

- **Průměrná váha současného řešení v dané iteraci algoritmu** – Bylo využito v rámci white box fáze (2.3). Hlavní účel této metriky byl získat příznak kvality algoritmu při jeho iterativním návrhu. Nebyl použit v black box fázi hlavně pro to, že vedlejší jev průměrování více běhů je 'zamaskování' konců všech běhů kromě toho, který trval nejdéle, protože osa grafu končí právě jeho poslední iterací. Proto je vhodnější jako taková 'fitness' funkce pro sledování průměrného běhu algoritmu, což bylo žádoucí při návrhu.

- **Průměrný počet splněných klauzulí v dané iteraci algoritmu** – Popis analogický s předchozí metrikou.

- **Úspěšnost nalezení řešení** – Běh byl považován za úspěšný, pokud našel validní řešení (validita nezahrnuje podmínku týkající se minimální váhy apod.). Spíše 'sanity check' metrika, která v podstatě podmíněně musí být rovna nebo velmi blízko 100%.

- **Průměrně nalezené % z optima pro daný soubor** – Jakou průměrně váhu měla řešení oproti optimu na daném souboru/setu. Hlavní metrika pro určení kvality našeho řešení.

- **Četnost nalezení přesně optim** – Jaké procento běhů skončilo nalezením právě optimálního řešení na daném souboru/setu.

2 Whitebox fáze: Návrhy a ladění algoritmu

2.1 Krok 0 – Hrubý test na naivním algoritmu

Na začátku projektu jsem se rozhodl pro implementaci naivního algoritmu, který ignoroval snahu maximalizovat výslednou váhu, pouze hledal řešení pseudo-náhodným přepínáním proměnných. Hlavní motivace za tímto krokem je, že jsem chtěl zkusit najít nějaký *baseline* algoritmus, od kterého se odrazit a se kterým porovnávat pozdější iterace (pokud tento naivní algoritmus hledal konzistentně lepší váhy atd., něco je špatně s nejnovější iterací aktuálního algoritmu...). Také sloužil k lepšímu seznámení se s heuristikou simulovaného chlazení v praxi.

Tento krok nakonec nebyl pro další postup nijak stěžejní, neboť takovýto algoritmus absolutně nemohl škálovat, takže nemělo cenu se pokoušet vytvářet nějaký baseline pro porovnávání – pro větší sady byl časově neúnosný. Z tohoto důvodu jsem od detailnější práce s ním upustil, ale pro úplnost ho chci ve white box fázi zmínit.

2.2 Krok 1 – První iterace reálného algoritmu

Při prvotním návrhu algoritmu jsem si vyčlenil následující cíle:

1. **Nějakým způsobem zahrnout váhu jednotlivých proměnných do procesu přepínání v dané iteraci**
2. **Nějakým způsobem zahrnout význam jednotlivých proměnných v kontextu klauzulí do procesu přepínání v dané iteraci**
3. **Implementovat mechanismus, který skrze běh algoritmu občasně přijme i zhoršující přepnutí stavů** – proti uváznutí v lokálním extrému a také snaha vybalancovat *deterministické sklony* bodu 1. a 2.

Na základě těchto cílů jsem vytvořil první iteraci algoritmu. Lze rozdělit na 2 části – **Pre-computing** a **Solve** fáze.

V pre-computing fází vypočítáme ohodnocení `score` pro každou proměnnou zadaného MWSAT problému. Jedná se o číselné vyjádření důležitosti proměnné v kontextu problému. Pro danou proměnnou `x` s váhou `w` platí

```
score[x]=(alpha * w + beta * (diff))
```

kde `alpha` a `beta` jsou modifikátory ovlivňující příspěvek `w` a `diff`, a `diff` je rozdíl mezi počtem klauzulí, které `x` úspěšně řeší, pokud je `True` vs `False`. V následující fázi pak `score` ovlivňuje průběh hledání řešení.

(*Pozn.:* `score` může být i záporné, pokud `diff`< 0. Výstup z pre-computing fáze je klesající seznam proměnných dle `score`, takže to ničemu nevadí)

Hlavní část algoritmu pak náhodně vybere nesplněnou klauzuli a v ní přepne nějakou proměnnou, přičemž tato volba je ovlivněna pořadím proměnných z první fáze algoritmu. Zároveň je toto přepnutí s pravděpodobností `p` nahrazeno méně ideálním přepnutím za účelem uniknutí z potenciálního lokálního extrému. Tato pravděpodobnost při běhu algoritmu postupně klesá.

Ze zběžného měření s touto implementací vyplynulo, že tento přístup už vede k vcelku dobrým výsledkům (jak procentem nalezených řešení, tak jejich ohodnocením vůči optimu), ale pouze na lehčích sadách (M a N) a do určité velikosti (cca 50 proměnných). Pro další iteraci algoritmu je tedy třeba zvětšit 'oblast' stavového prostoru, kterou algoritmus pokryje (pokud možno bez přímého zvětšení iterací, neboť to by vedlo na velmi špatné škálování). Dále

by také bylo vhodné implementovat opatření proti stagnaci, neboť jsem při výpisech průběhů algoritmu zaznamenal, že algoritmus někdy mnoho iterací zůstává na stejném počtu splněných klauzulí bez posunu směrem k nalezení řešení.

2.3 Krok 2 – Finální iterace reálného algoritmu se simulovaným chlazením

Jakožto odpověď na nedostatky předchozí iterace byl algoritmus obohacen o 2 mechanismy:

1. **Periodická perturbace** – nahrazuje pravděpodobnost odmítnutí `p`. Každých n iterací přepne 1 až 3 náhodně zvolené proměnné $\Rightarrow$ snaha snížit potenciální přílišnou závislost na `score`, která by vedla k až moc deterministickému chování. Tím pádem i pomáhá vyskočit z lokálních extrémů.

2. **Opatření proti trvající stagnaci** – v průběhu algoritmu zaznamenáváme doposud nejlepší nalezený stav řešení a pokud během n iterací nenajdeme lepší $\Rightarrow$ zvýšíme šanci přijmout zhoršující stavy (nyní skrze vnitřní proměnnou, později skrze teplotu, o tom víc níže) ve snaze vyskočit ze stagnace.

Z malého vzorku výsledků této implementace jsem usoudil, že je dostatečně dobrá na to, abych na ní aplikoval heuristiku simulovaného chlazení. Po menších úpravách byl algoritmus obohacen o mechanismus chlazení (který například nahradil vnitřní proměnnou odmítnutí stavu použitou u stagnace).

Pro stručnost není uveden přímo zdrojový kód implementace (dostupný v příloze), pseudokód demonstrující logický průběh algoritmu (anglicky pro jasnější a stručnější popis):

- P – daný MWSAT problém (instance třídy `WeightedSATParser`)
- T_0 – počáteční teplota (`T0`)
- α – tempo ochlazování (`alpha`)
- M – horní strop počtu iterací (`max_iterations`)
- T_{min} – minimální teplota (`T_min`)
- S_{max} – maximální stagnace (`max_stagnation`)

Algorithm 1 SATSolverScored2

INPUT – Problem P, init temp T_0, cooling rate α, max iterations M, min temp $T_{\min}$, max stag $S_{\max}$

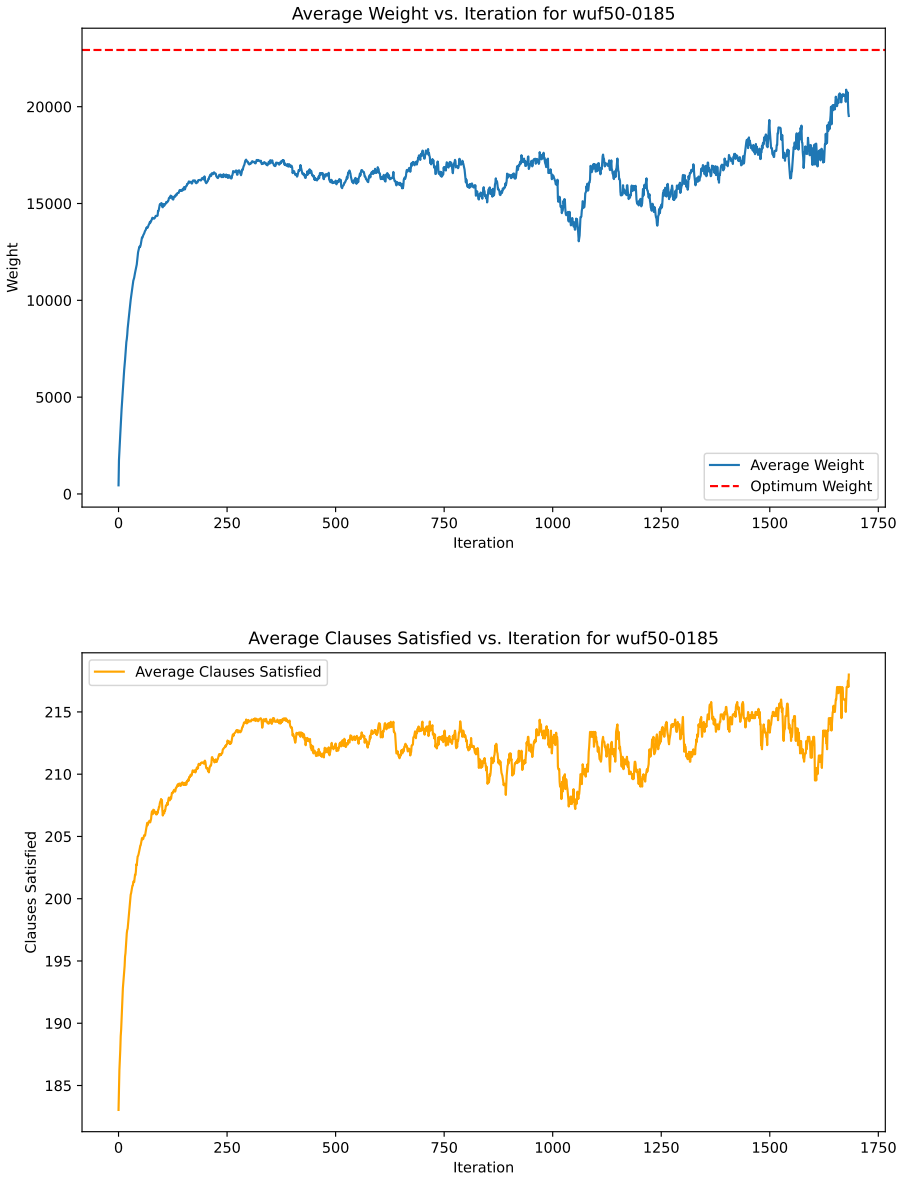
OUTPUT – Best found solution $\mathbf{x}_{\text{ret}}$, weight W_{ret}

Precompute *score*.
$T \leftarrow T_0$, $W_{\text{ret}} \leftarrow 0$, $stag \leftarrow 0$.
for $i = 1$ to M **do**
    Identify unsatisfied clauses U.
    if $U = \emptyset$ **then return** $\mathbf{x}_{\text{ret}}, W_{\text{ret}}$
    end if
    flip variable in a random $c \in U$. ▷ influenced by *score*
    Calculate Δ (change in solution quality).
    Accept new solution with probability p_{acc} based on T.
    if Improved solution **then**
        $\mathbf{x}_{\text{ret}}, W_{\text{ret}}, stag \leftarrow 0$. ▷ update
    else
        $stag \leftarrow stag + 1$.
    end if
    if $stag \geq S_{\max}$ **then**
        $T \leftarrow 0.5 * T_0$, $stag \leftarrow 0$. ▷ anti-stag reheat
    end if
    $T \leftarrow \max(T \cdot \alpha, T_{\min})$ ▷ cooling schedule.
    if $i \bmod 100 = 0$ **then**
        Flip 1–3 random variables. ▷ perturbation
    end if
end for
return $\mathbf{x}_{\text{ret}}, W_{\text{ret}}$

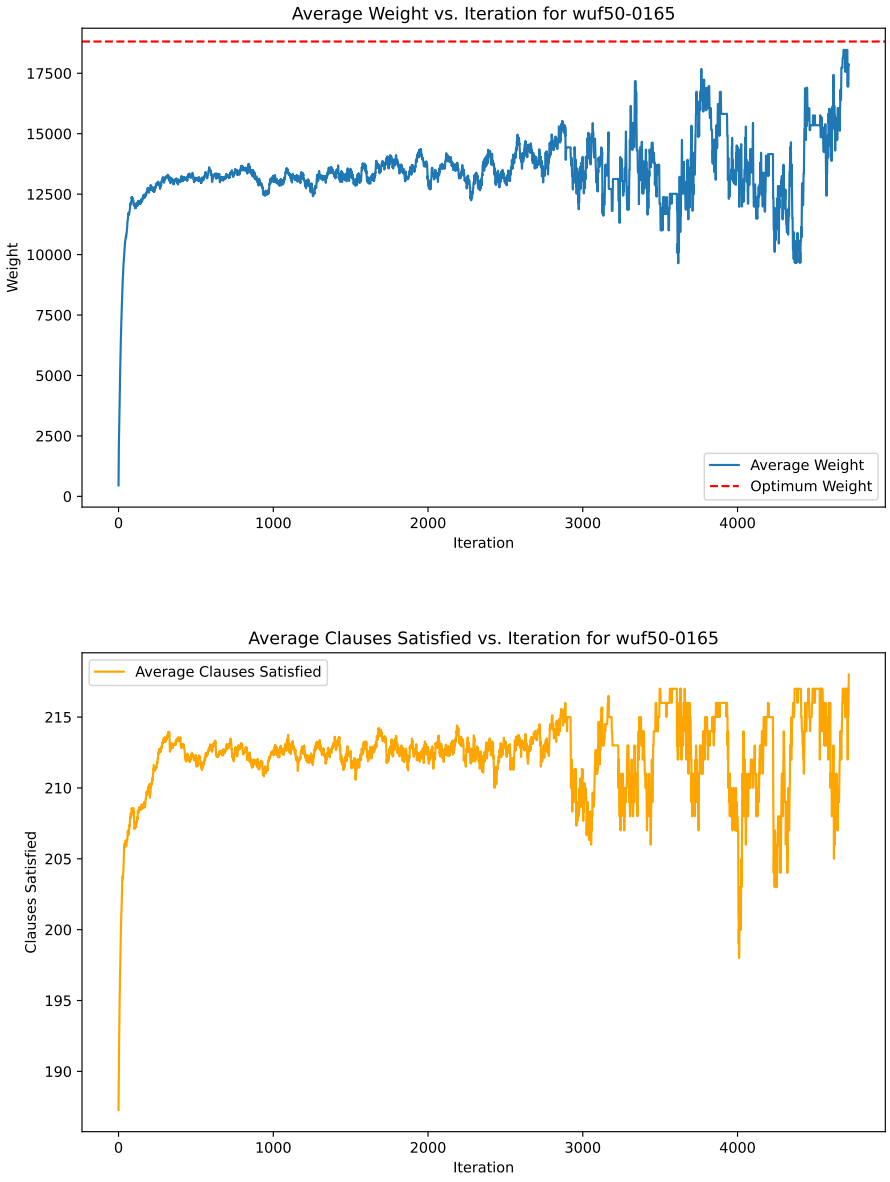
V tomto případě uvedu i výstupy z měření, neboť to je finální iterace algoritmu před laděním parametrů a black box fází. Tuto implementaci jsem pro zběžné zhodnocení spustil na souborech různých obtížností ze středně velké sady `wuf50-218`. Měření na daném souboru proběhlo stokrát, přičemž jako hlavní výstup byl průměrný počet splněných klauzulí v daném kroku a průměrná váha proměnných řešení v daném kroku. Vstupní parametry jsem zvolil následovně:

- `T0 = 17.5`
- `T_min = 0.01`
- `max_stagnation = 30`
- `alpha=0.98`
- `max_iterations = 100000`

Jako ilustraci výstupu přikládám grafy (2.3) pro dva soubory složitosti M a N, které ukazují vývoj váhy (resp. splněných klauzulí) daného stavu řešení v dané iteraci.



Obrázek 1: Výstup z `wuf50-0185` (M)



Obrázek 2: Výstup z `wuf50-0165` (N)

2.4 Nastavení parametrů chlazení

Dalším, a v podstatě posledním krokem whitebox fáze bylo nalezení vhodné konfigurace vstupních parametrů, hlavně tedy `T0`, `alpha` a `max_stagnation`,

neboť z hraní si s nimi bylo zřejmé, že mají nejvýznamnější vliv na výkon algoritmu. Pro nalezení parametrů jsem zvolil **porovnávání výsledků pro jednotlivé konfigurace na stejném vzorku dat**. Postup hledání byl:

1. **Základní nastavení mezí** – Určení první konfigurace, ze které budeme vycházet. Zvolil jsem v podstatě podle pocitu ze zběžných měření (vidět v 1).

2. **Generace konfigurací s pomocí interpolace** – Ze zadaných mezí (ve formátu `.json` souboru) jsem si vygeneroval vzájemné kombinace hodnot parametrů při interpolaci přes meze s pěti vzorky (tj. každý parametr má jednu z 5 hodnot v intervalu jeho mezí, které tento interval rovnoměrně pokrývají). Celkem tedy vznikne 125 konfigurací (minimální teplota a počet iterací je konstantní).

3. **Měření na malém vzorku dat různých obtížností** – Každá konfigurace byla poté desetkrát spuštěna na 12 souborech ze sady `wuf50-218` (tři z každé obtížnosti) a výsledky byly ukládaný do `.csv` souborů.

4. **Analýza výsledků a nalezení nejlepší konfigurace** – Z těchto výsledků byla poté určena nejlepší konfigurace pro daný soubor a také celkově nejlepší konfigurace. Roli při určování hrálo **dosažené % optimálního řešení** u daného problému a úspěšnost nalezení řešení. Za celkově nejlepší konfiguraci považuji nastavení dosahující nejlepší úspěšnosti a % nalezení optima napříč obtížnostmi. Finální `.csv` soubor z první iterace hledání je vidět v 3.

5. **Druhá iterace s nastavením mezí kolem nejlepší konfigurace** – Celý výše zmíněný postup byl opakován s novým základním nastavením (v 2) vycházejícím z nejlepší konfigurace. Výsledky lze vidět v 4.

Tabulka 1: Základní nastavení (Původní)

Parametr	Levá mez	Pravá mez
T0	10.0	40.0
alpha	0.90	0.999
max_iterations	100000	100000
T_min	0.001	0.001
max_stagnation	15	200

Tabulka 2: Základní nastavení (Iterace 1)

Parametr	Levá mez	Pravá mez
T0	20.0	30.0
alpha	0.98	0.9999
max_iterations	100000	100000
T_min	0.001	0.001
max_stagnation	45	75

Tabulka 3: Výsledky nastavení (Původní)

File	Config ID	Avg. % of Optimal	Success Rate (%)
wuf50-01-M	17	96.2382	100.0
wuf50-01-N	56	96.0150	100.0
wuf50-01-Q	53	71.3126	100.0
wuf50-01-R	83	64.1371	100.0
wuf50-02-M	8	98.7860	100.0
wuf50-02-N	64	98.4847	100.0
wuf50-02-Q	21	62.8931	100.0
wuf50-02-R	96	71.1557	100.0
wuf50-03-M	118	93.2482	100.0
wuf50-03-N	125	93.0676	100.0
wuf50-03-Q	50	74.3770	100.0
wuf50-03-R	26	74.1815	100.0
BEST	72	76.8324	100.0

Tabulka 4: Výsledy nastavení (Iterace 1)

File	Config ID	Avg. % of Optimal	Success Rate (%)
wuf50-01-M	95	96.3132	100.0
wuf50-01-N	13	96.0902	100.0
wuf50-01-Q	14	59.0167	100.0
wuf50-01-R	101	67.0768	100.0
wuf50-02-M	104	98.4224	100.0
wuf50-02-N	24	98.7272	100.0
wuf50-02-Q	15	61.8039	100.0
wuf50-02-R	69	67.8277	100.0
wuf50-03-M	64	94.5895	100.0
wuf50-03-N	105	93.7711	100.0
wuf50-03-Q	51	72.2488	100.0
wuf50-03-R	70	75.2453	100.0
BEST	113	76.0610	100.0

Po dvou iteracích ladění nastavení jsem se rozhodl přejít k měření black box fáze s konfigurací v tabulce 5. Tato konfigurace by měla dosahovat nejlepších výsledků napříč sadami a zároveň měla 100% úspěšnost nalezení řešení na vybraném vzorku dat:

Tabulka 5: Nastavení pro Black box

Parametr	Hodnota
`T0`	30.0
`alpha`	0.98995
`max_iterations`	100000
`T_min`	0.001
`max_stagnation`	60

3 Blackbox fáze: Měření, vizualizace a vyhodnocení

3.1 Měření & Formát výstupu

V rámci měření byl algoritmus spuštěn se všemi soubory ze sad zmíněných v 1.2, přičemž každá instance byla měřena stokrát a následně zprůměrována.

Výstupem z každého jednotlivého měření byl `.csv` soubor obsahující informace o **nejlepším nalezeném řešení**, **optimálním řešení pro daný soubor**, a také **dosaženém % z daného optima**. Další zpracování pak zahrnovalo zprůměrování všech běhů pro daný soubor a následné uložení do hromadného `.csv` souboru pro danou sadu a obtížnost. Finálním a ústředním výstupem z měření byla tedy sada celkem dvanácti (4 obtížnosti × 3 sady) souborů:

- `BATCH-20-91-[MNQR].csv`
- `BATCH-50-218-[MNQR].csv`
- `BATCH-75-325-[MNQR].csv`

přičemž tyto soubory jsou formátu:

```
Formula | avg % of optimum found | optimums found | blunders
```

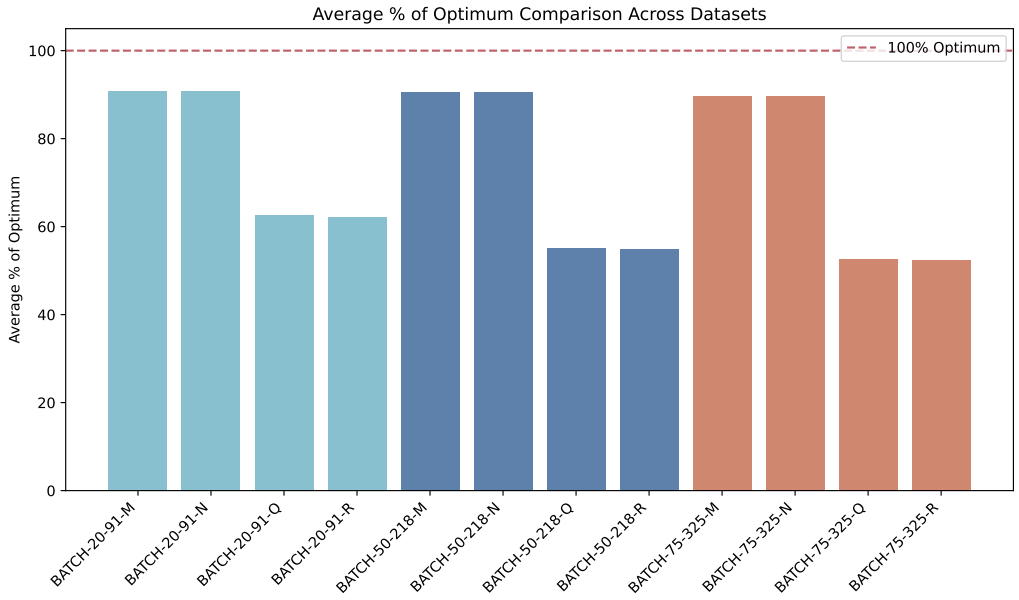
kde význam všech sloupců, až možná na **blunders**, je snad jasný. Sloupec **blunders** ukazuje absolutní (tj. neprůměrovaný, jenom nasčítaný ze všech opakování běhu souboru) počet neúspěšných běhů pro daný soubor.

TLDR: Každý z finálních souborů pro danou sadu a obtížnost obsahuje záznam pro každý soubor dané složitosti a sady, který reflektuje průměrný výkon ze všech běhů na tomto souboru. Na konci každého finálního souboru pak byl ještě přidán řádek shrnující výkon algoritmu na sadě a obtížnosti.

3.2 Výsledky – Nalezené % z optima

První metrika pro vyhodnocení výkonu algoritmu a jeho konfigurace je to, jak se výsledky v průměru blížily optimální hodnotě. Tato hodnota je pro každou sadu vyobrazena ve sloupcovém grafu 3 doprovázeném tabulkou dat 6. Z grafu můžeme jasně vidět, že hlavním faktorem ovlivňujícím výstup algoritmu je kategorie obtížnosti, zatímco velikost je zanedbatelná. To znamená, že náš algoritmus škáluje dobře v kontextu velikosti datasetů pro tento úkol.

Na druhou stranu není tak přizpůsobivý v případě 'klamných' problémů, jak bychom si přáli (více v diskuzi 3.4).



Obrázek 3: Průměrně dosažené % optimálního řešení v každé sadě

Tabulka 6: Tabulka hodnot

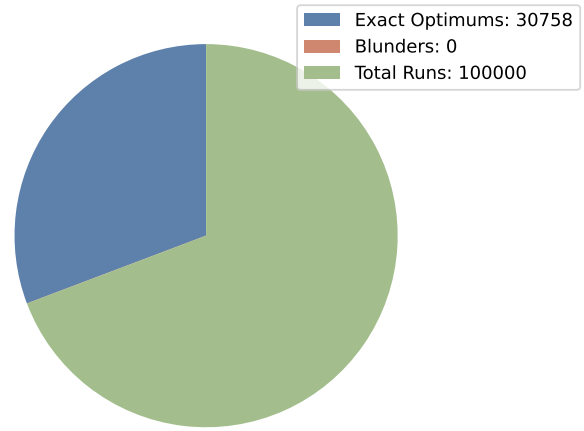
Sada	Hodnota
20-91-M	90.8 %
20-91-N	90.8 %
20-91-Q	62.5 %
20-91-R	62.1 %
50-218-M	90.7 %
50-218-N	90.7 %
50-218-Q	55.0 %
50-218-R	54.8 %
75-325-M	89.6 %
75-325-N	89.7 %
75-325-Q	52.6 %
75-325-R	52.5 %

3.3 Výsledky – Počet nalezených optim & Neúspěšných běhů

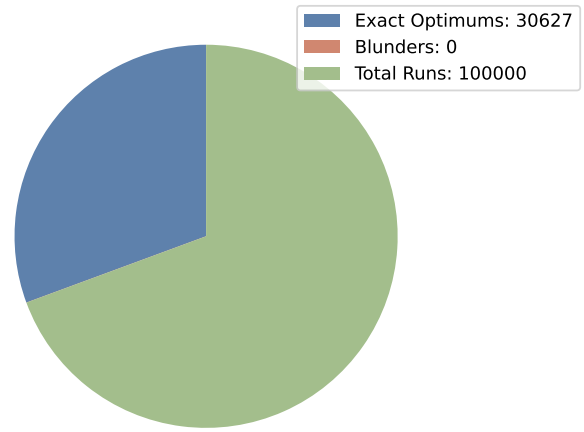
Druhá metrika nabízí přehled o tom, kolik běhů skončilo nalezením přesně optimálního řešení a kolik běhů bylo neplatných. Z následujících grafů 4 můžeme vyvodit, že zde je hlavní proměnnou velikost problému, což dává smysl. Větší problém má větší stavový prostor s (zpravidla) více řešeními, takže je větší šance, že vrátíme nějaké jiné řešení, než právě to optimální. Oproti tomu vliv složitosti na počet nalezených optim je nejspíš zanedbatelný. Z výsledků to dokonce spíš vypadá, že větší složitost (zpravidla tedy i méně řešení) vede na malinko větší množství nalezených optim, než u lehkých sad stejné velikosti.

Pokud se zaměříme na neúspěšné běhy, obtížnost i velikost se zdají obě být faktory. Počet neúspěšných řešení považuji za dobrý, neboť je jich velmi málo. Nejhorší sada problémů – `75-325-R` – má neúspěšnost pouze 1.8 %.

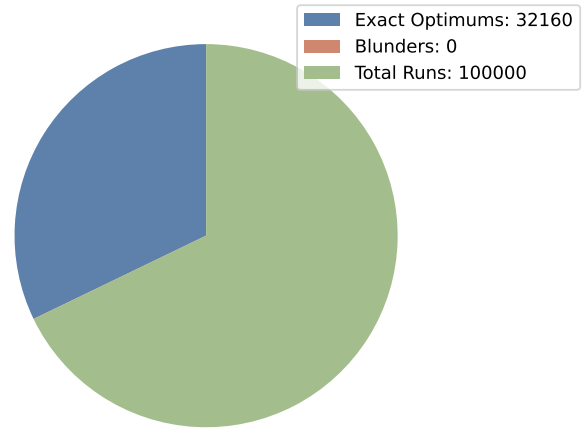
Run Distribution - BATCH-20-91-M.csv



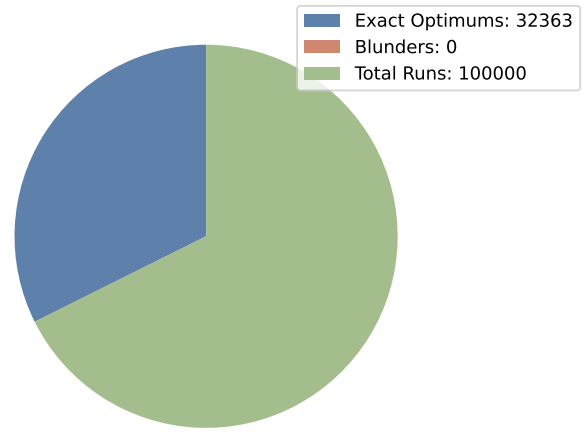
Run Distribution - BATCH-20-91-N.csv



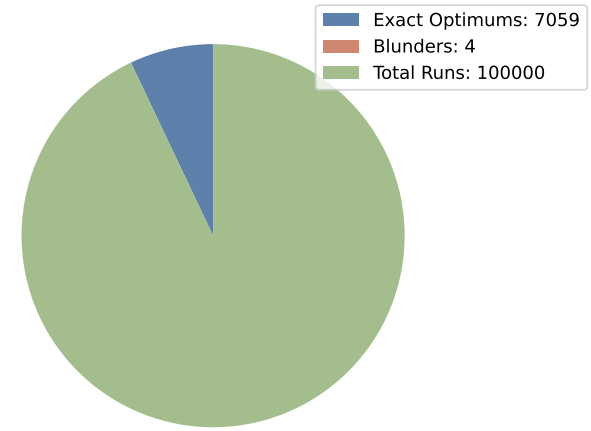
Run Distribution - BATCH-20-91-Q.csv



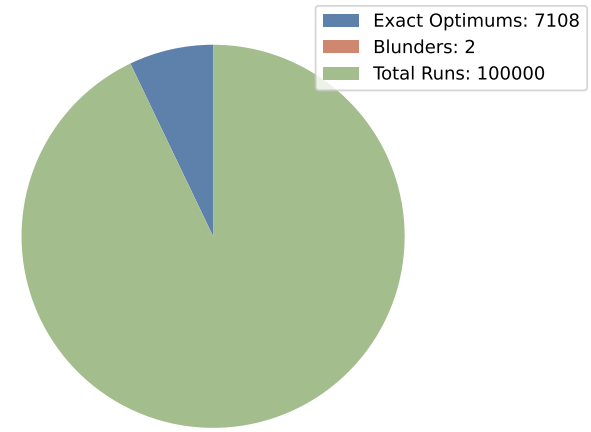
Run Distribution - BATCH-20-91-R.csv



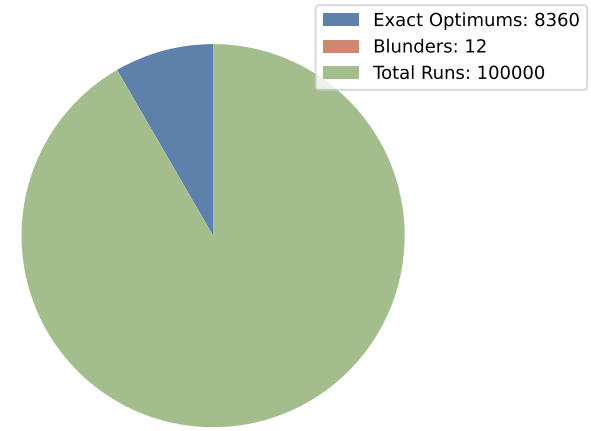
Run Distribution - BATCH-50-218-M.csv



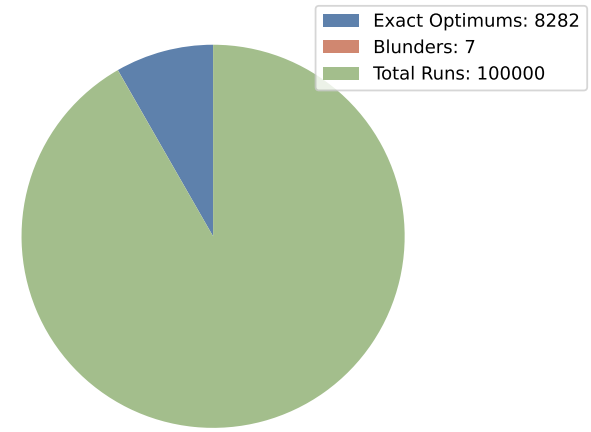
Run Distribution - BATCH-50-218-N.csv

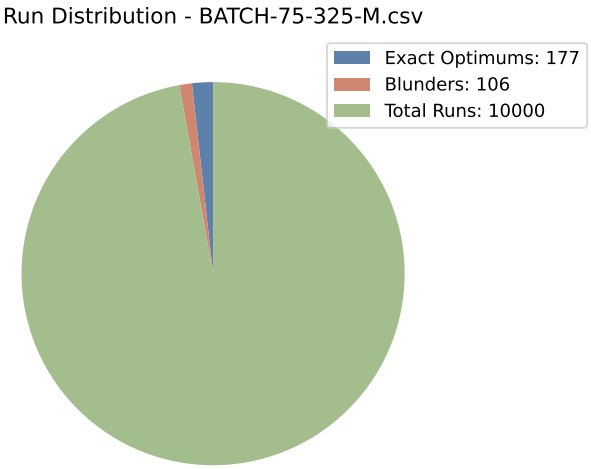


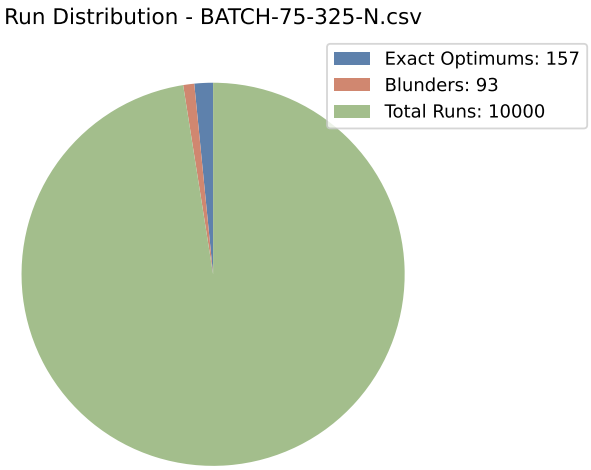
Run Distribution - BATCH-50-218-Q.csv

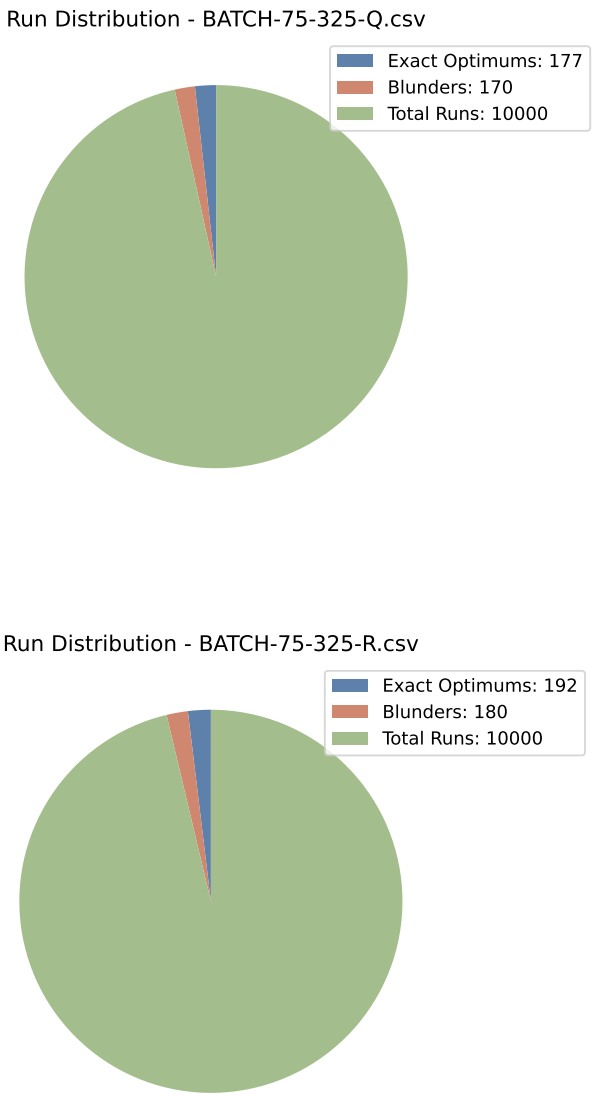


Run Distribution - BATCH-50-218-R.csv




Run Distribution - BATCH-75-325-M.csv
Exact Optimums: 177
Blunders: 106
Total Runs: 10000


Run Distribution - BATCH-75-325-N.csv
Exact Optimums: 157
Blunders: 93
Total Runs: 10000



Obrázek 4: Grafy nalezených optim & neúspěšných běhů

3.4 Diskuze

Z výsledků black box fáze považuji řešení za dostatečně efektivní, nicméně ne bez výhrad. Je zřejmé, že pro problémy Q a R není tento přístup tak přizpůsobivý, jak by mohlo být požadováno.

Algoritmus vykazuje dobrou škálovatelnost a vysokou úspěšnost při hledání řešení, jak je vidět z výsledků. Nicméně maximalizace vah u složitějších problémů upadá – kolem 57 % optima, zatímco u jednodušších se pohybuje kolem 90 %.

Teorie založená na výsledcích měření a zpětném přemýšlení nad postupem návrhu algoritmu je následující – (aspoň částečnou) příčinou je přílišná korekce, ke které došlo během vývoje algoritmu. Jak je popsáno ve white box sekci textu (2.2), v průběhu návrhu algoritmu jsem čelil problémům se škálováním a nalezením řešení pro složité instance. To mě vedlo k implementaci mechanismů, které preferovaly nalezení řešení i za cenu určité ztráty efektivity v maximalizaci vah. Hlavním problémem se zdá být příliš agresivní přístup při výpočtu skóre jednotlivých proměnných, kde se algoritmus snaží kombinovat jak váhu proměnné, tak její příspěvek k řešení do jednoho hodnotového parametru. Toto jediné skóre ale nemůže dostatečně reflektovat oba aspekty současně, což vede k problémům.

U klamných problémů jsou váhy často rozloženy tak, že optimální řešení vyžaduje volbu proměnných s relativně nízkou váhou oproti zbytku. V aktuální implementaci algoritmus seřadí proměnné dle skóre a „jde odshora“, což často způsobuje, že se pohybuje vysoko nad optimálním řešením a následně se dostává do stagnace. V takových případech se aktivuje prevence stagnace umožňující přijetí zhoršujících stavů. Tento systém vede k celkem rychlému nalezení i pro těžké problémy, ale očividně až moc agresivně snižuje váhu finálního vráceného řešení. Tedy běh začne vysoko nad optimem a ve snaze najít řešení spadne příliš nízko pod něj.

Pro budoucí iterace algoritmu by bylo vhodné uvažovat o rozdělení skóre proměnných na dva samostatné parametry – jeden zaměřený na maximalizaci váhy a druhý na příspěvek k nalezení řešení. Poté by šlo postavit sofistikovanější logiku prohledávání stavového prostoru.

4 Závěr

Cíl tohoto projektu bylo provést implementaci a vyhodnocení heuristiky simulovaného ochlazování v kontextu problému maximální vážené splnitelnosti (MWSAT). Navržený algoritmus považuji za dostatečně dobrý, neboť prokázal schopnost efektivně řešit problémy v zadaných benchmarkových sadách, i přes pokles pro nejtěžší sady problémů. Věřím, že měření a zpracování dat

bylo provedeno adekvátně, a že případné nedostatky byly dostatečně okomentovány, přičemž pro ně bylo navrženo potenciální řešení a zlepšení pro budoucí iterace.