Heuristika simulovaného ochlazování pro řešení MaxWeightedSAT

Luboš Zápotočný

České vysoké učení technické v Praze - fakulta informačních technologií
zapotlub@fit.cvut.cz

I. Úvod

Problém splnitelnosti booleovské formule (označováno z angličtiny SATISFIABILITY, zkráceně SAT) označuje problém nalezení splňujícího (vyhovujícího) ohodnocení logické formule v konjunktivní normálním formě tak, aby byly všechny její klauzule splněné.

SAT byl první problém o kterém se dokázalo, že je NP-úplný [1]. Tedy pro tento problém neexistuje (za předpokladu P $\neq$ NP) efektivní algoritmus, který by tento problém řešil v polynomiálním čase. Jelikož se jedná o NP-těžký (NP-úplné problémy jsou podmnožinou NP-těžkých) problém, lze na tento problém převést instance všech problému ze tříd P a NP.

MaxWeightedSAT označuje optimalizační verzi hledání SAT ohodnocení zároveň s kritériem pro maximalizaci součtu vah proměnných ohodnocených 1 (True). Problém je tedy rozšířen o atributy $w(x_i)$ pro všechny proměnné x_i reprezentující váhové ohodnocení jednotlivých proměnných.

Tato práce se zaměřuje na řešení výše zmíněného NP-těžkého optimzačního problému pomocí heuristiky simulovaného ochlazování.

Algoritmus náhodně prochází stavovým prostorem ohodnocení proměnných formule tak, aby maximalizoval součet vah pozitivně ohodnocených proměnných a zároveň aby toto ohodnocení splňovanou zadanou formuli.

Stavový prostor je tedy vektor (pole) boolevských ohodnocení (True/False) jednotlivých proměnných. Operátorem přechodu do nového stavu je logická změna jednoho náhodného bitu v tomto vektoru.

Algoritmus přechází do zlepšujících stavů a s určitou pravděpodobností přechází také do zhrošujících stavů. Tímto postupem se heuristika snaží zamezit uvíznutí v lokálních extrémech.

Heuristika začíná s vysokou teplotou, která ovlivňuje mimo jiné také pravděpodobnost přijetí nezlepšujících stavů. To vede k velkému prozkoumání stavového prostoru v prvních krocích algoirtmu. Každý krok algorimu sníží tuto teplotu o násobek chladícího faktoru (lineárně). Tímto chlazením se také snižuje pravděpodobnost přijetí zhrošujícího stavu a heuristika tímto konverguje k nalezení optimálního řešení.

Podrobný popis datových sad lze najít na stránkách kurzu NI-KOP ČVUT FIT [2].

II. Implementace

Heuristika a experimentální vyhodnocení byly naprogramovány v jazyce Python. Při hledání ideálních parametrů heuristiky či efektivity algoritmu není uvažována efektivita jazyka jako takového. Všechny metriky jsou univerzálně přenositelné mezi různými hardwarovými platformami.

Hlavní část heuristiky je zobrazena na obrázku 1 kde lze nahlédnou, jak algoritmus pracuje.

Pravděpodobnost přijetí zhoršujícího řešení je závislá na rozdílu hodnot účelové funkce mezi aktuálním řešením a potenciálním novým řešením. Zároveň je závislá na aktuální teplotě. Čím větší teplota, tím větší pravděpodobnost,

```
temperature = self.initial_temperature

while temperature > self.final_temperature:
    for _ in range(self.num_iterations_per_temperature):
        new_solution = self.perturb_solution(current_solution)

        new_objective = self.objective_function(new_solution)
        delta = new_objective - current_objective

        if delta > 0 or self.probability(abs(delta), temperature) > random.uniform(0, 1):
            current_solution = new_solution
            current_objective = new_objective

            if current_objective > best_objective:
                best_solution = current_solution
                best_objective = current_objective

    temperature = self.cool_temperature(temperature)
```

Obrázek 1: *Hlavní smyčka algoritmu*

```
@staticmethod
def probability(delta: int, temperature: int):
    exponent = -delta / temperature
    if exponent > 1:
        return 1
    return math.exp(exponent)
```

Obrázek 2: *Výpočet pravděpodobnosti přijetí zhoršujícího řešení*

že bude přijato zhoršující řešení. Konkrétní implementaci lze vidět na obrázku 2. Výsledek této funkce je následně v hlavním cyklu porovnán s náhodnou hodnotou v internalu od 0 do 1 a pokud tato hodnota převyšuje vygenerovanou náhodnou hodnotu přijímáme zhrošující řešení.

Zlepšující řešení přijímáme vždy.

Průchod stavovým prostorem zajišťuje funkce *perturb_solution* (obrázek 3) která flipuje náhodné bity v aktuálním ohodnocení. Tímto se můžeme dostat k méně optimálnímu řešení než které aktuálně máme, ale zajišťujeme si tím větší průchod možných konfigurací a mitigujeme uváznutí v lokálních extrémech.

Algoritmus simulovaného ochlazování postupně odhlazuje aktuální teplotu. Na konci každé iterace je v hlavním cyklu volána metoda zobrazená na obrázku 4. Chladící faktor je

```
def perturb_solution(self, assignment: tuple[bool]) -> tuple[bool]:
    assignment = list(assignment)
    for _ in range(self.perturbation_flips):
        variable = random.randint(0, len(assignment) - 1)
        assignment[variable] = not assignment[variable]
    return tuple(assignment)
```

Obrázek 3: *Perturbace ohodnocení proměnných*

```
def cool_temperature(self, temperature: int):
    return self.cooling_factor * temperature
```

Obrázek 4: *Metoda pro chlazení teploty*

parametr heuristiky v rozsahu od 0 do 1, který je v následujících sekcích práce experimentálně nastaven na ideální hodnotu.

Při implementaci a testování heuristiky byly použity tyto testovací sady

- wuf20-71
- wuf20-71R
- wuf20-91
- wuf20-91R
- wuf50-200
- wuf50-219
- wuf50-218R
- wuf75-325
- wuf100-430

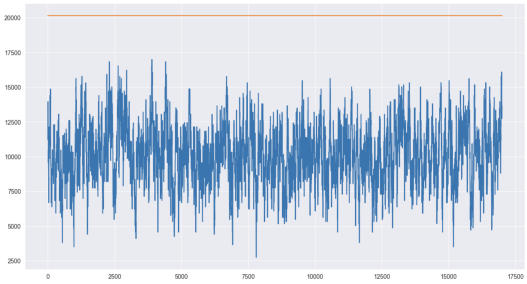
Parametry heuristiky jsou následující

- *initial_temperature* - počáteční teplota
- *final_temperature* - konečná teplota
- *num_iterations_per_temperature* - počet iterací vnitřního cyklu ochlazování
- *perturbation_flips* - počet náhodně vybraných bitů pro flipnutí při prohledávání stavového prostoru
- *cooling_factor* - desetinné číslo (0, 1) reprezentující lineární funkci pro chlazení
- *penalty* - celočíselná (záporná) hodnota penalizující nesplněnou klauzi (*None* pro aktivaci adaptivní penalizace)

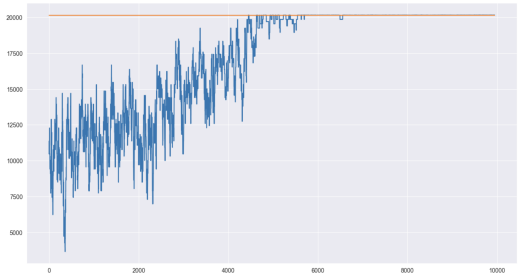
III. White box testování

Jeden z prvních testů celého program odhalil závažnou chybu v programu. Chyba byla odhalena až po detailním výpisu grafu aktuálně nejlepších řešení heruistiky. Teorie simulovaného ochlazování říka, že se snižující teplotou by se mělo řešení ustálit a zlepšovat. To na grafu zobrazeném na obrázku 5 ale vůbec není patrné.

Daný problém spočíval ve výpočtu a použití hodnoty delta (na obrázku 1). Parametr delta se vypočítává odečtením hodnoty účelové



Obrázek 5: *Simulované ochlazování bez konvergence*



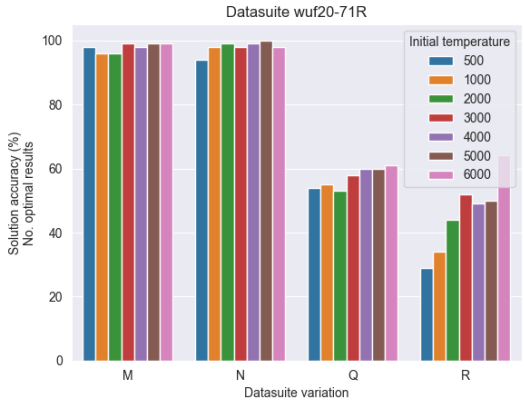
Obrázek 6: *Simulované ochlazování s konvergencí*

funkce perturbovaného ohodnocení a hodnoty účelové funkce aktuálního ohodnocení. Následně je tato hodnota delta porovnána, zdali je větší než 0, což znamená zlepšení, v tom případě pertubované řešení nahrazuje aktuální ohodnocení a cyklus pokračuje dále.

Problém nastával ale v tom, že na parametru delta je také závislý výpočet pravděpodobnosti přijetí horšího řešení. V tomto případě je nutné počítat s hodnotou delta v absolutní hodnotě. Protože pro nízké teploty je tato delta velmi malá a zapříčinilo to přijetí mnoha zhoršujících řešení. Obrázek 6 zobrazuje graf vývoje aktuálně nejlepšího řešení problému. Na tomto grafu je již vidět trend ochlazování a konvergence k optimálnímu řešení.

Následně byly testovány jednotlivé parametry heuristiky a závislost jejich nastavení na úspěšnosti nalezení optimálního řešení. Na následujících grafech je zobrazana úspěšnost nalezení optimálního řešení. Heuristika nalézá také neoptimální řešení, která jsou blízko optimálnímu, tato chyba zde však porovnána nebyla.

Základním testem bylo nastavení počáteční



Obrázek 7: *Experiment nastavení počáteční teploty 20-71R*

teploty. Obrázky 7 a 8 zobrazují úspěšnosti nalezení optimálního řešení na instancích wuf20-71R a wuf50-218R s různými hodnotami počáteční teploty. Testované hodnoty byly z množiny

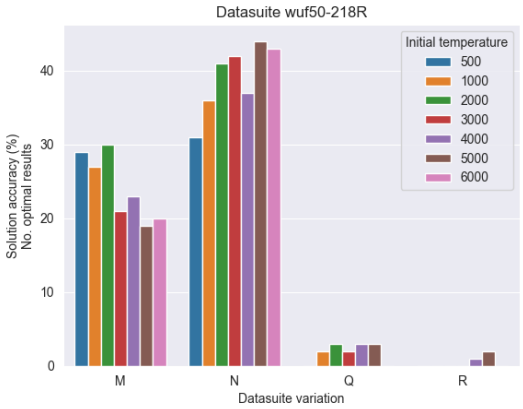
- 500
- 1000
- 2000
- 3000
- 4000
- 5000
- 6000

V těchto testech se hodnota 5000 jakožto počáteční teplota ukázala být nejvhodnější s přihlédnutím na úspěšnost v složitejších instancích a délce výpočetního času.

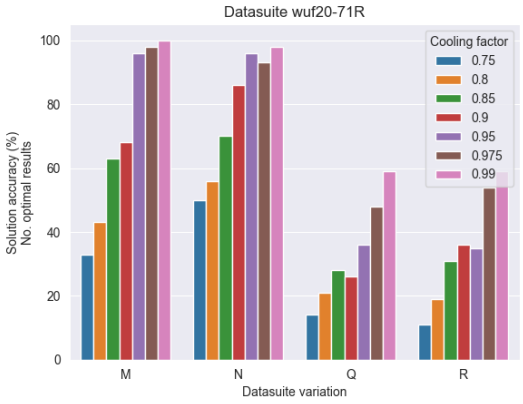
Dále byl testován parametr chladícího faktoru. Tento parametr ovlivňuje rychlost konvergence metody a zároveň společně s nastavením počáteční a koncové teploty určuje počet kroků hlavního cyklu.

Nastavení tohoto parametru bylo testováno na hodnotách

- .75
- .8
- .85
- .9
- .95
- .975
- .99



Obrázek 8: *Experiment nastavení počáteční teploty 50-218R*



Obrázek 9: *Experiment nastavení chladícího faktoru 20-71R*

Grafy na obrázcích 9 a 10 zobrazují úspěšnost nalezení optimálního řešení v závislosti na nastavení hodnoty chladícího faktoru.

Experiment navrhuje jako nejlepší hodnotu .99, výsledná hodnota ale byla zvolena .975 z důvodu rychlejší konvergence - rychlejšího běhu algoritmu.

Algoritmus obsahuje účelovou funkci, která nabývá hodnoty vah aktuálního ohodnocení, ale zároveň penalizuje nesplněné klauzule. Tímto postupem je zajištěna maximalizace vah řešení společně se snahou nalézt ohodnocení, které splňuje zadanou formuli. Zároveň je zde ale volnost, která umožuje heuristice některé klauzule nesplnit, pokud je splněná většina ostatních.

Obrázek 10: *Experiment nastavení chladícího faktoru 50-218R*

```python
def objective_function(self, assignment: tuple[bool]):
    unsatisfied_clauses = self.mwcnf.unsatisfied_clauses(assignment)

    penalization = (1 - (unsatisfied_clauses / self.mwcnf.clauses_count)) * (unsatisfied_clauses * self.penalty)
    penalization += self.mwcnf.variables_count / 8

    return sum([
        self.mwcnf.weights[index] for index, variable in enumerate(assignment) if variable
    ]) + penalization
```

Obrázek 11: *Účelová funkce heuristiky*

Po několika iterací a manuálním testování úspěšnosti a efektivity algoritmu byla zvolena tato výpočetní formule (obrázek 11), která úměrně penalizuje nesplněné klauzuje skalárním násobkem (penalizace). Tento skalární násobek je také parametr heuristiky a pokud je vyplněn 0, heuristika si automaticky vhodnou penalizaci dopočte z vektoru vah jednotlivých proměnných (obrázek 12).

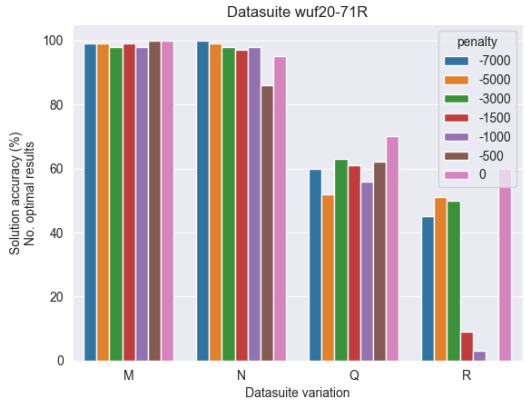
Výsledný experiment pro nastavení správné hodnoty penalizace obsahoval tyto parametry

- -7000
- -5000
- -3000
- -1500
- -1000
- -500
- 0 (automatický výpočet dle vah)

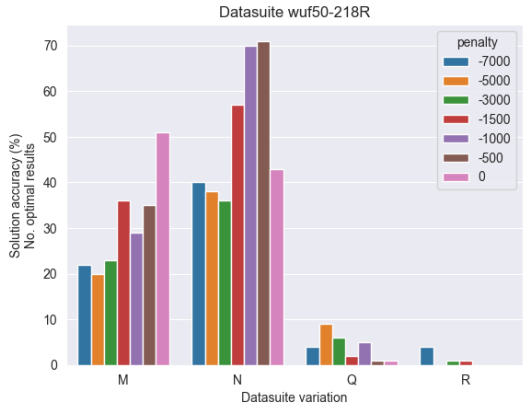
Graf na obrázcích 13 a 14 zobrazují procen-

```python
if not penalty:
    self.penalty = -int(sum(self.mwcnf.weights) / len(self.mwcnf.weights))
```

Obrázek 12: *Automatické vypočtení penalizačního skaláru*



Obrázek 13: *Experiment nastavení penalizačního skaláru 20-71R*



Obrázek 14: *Experiment nastavení penalizačního skaláru 50-218R*

tuální úspěšnost nalezení optimálního řešení v závislosti na penalizačním skaláru.

Tento experiment potvrzuje dobré nastavení penalizačního merchanizmu a automatického výpočtu penalizace pomocí vektoru vah pro menší instance, které mají neoptimálně nastavené váhy (varianty Q a R).

Automatický výpočet vah se ale neosvědčil při zpracovávní větších instancí a proto byla nakonec zvolena penalizační konstanta -5000, která měla obecně dobrou úspěšnost a zároveň zvládala neoptimální varianty úloh řešit s nadprůměrnou úspěšností.

IV. Black box testování

Rámec black box testování již měl zafixované parametry heuristiky, které v minlé sekci vyšli jako nejlepší. Jedná se o počáteční teplotu 5000, odhlazující faktor 0.975 a fixní penalizaci -5000.

V rámci této sekce byly zvoleny dvě hlavní skupiny instancí, na kterých bude úspěšnost heuristiky demonstrována

- wuf-20-71R
- wuf-50-218R

V rámci těchto skupin existují podskupiny M, N, Q a R, kde každá obsahuje 100 instancí. Jelikož heuristika využíva mnoho náhodných prvků, každá instance byla spuštěna 10. Celkově se tedy pro tyto dvě skupiny spustilo 8000 běhu algoritmu.

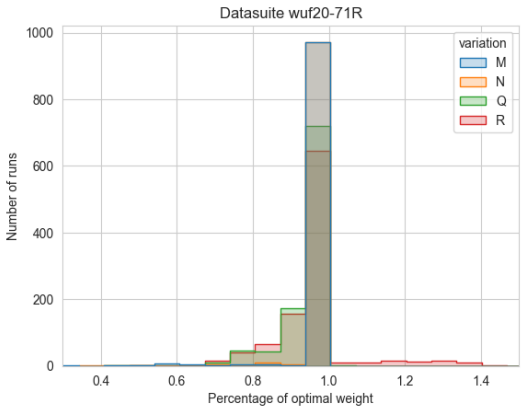
Histogram na obrázku 15 vyobrazuje procentuální odchylku váhy nalezeného ohodnocení od optimální váhy. V případě, že je odchylka menší než 1.0, heuristika nedokázala najít to nejlepší splňující ohodnocení. Pokud byla odchylka ostře větší než 1.0 znamená to, že heuristika nalezla ohodnocení s větším součtem vah. Jelikož máme data o optimálním řešení, říká nám to, že v těchto případech penalizace nesplněných klauzilí nebyla dostatečná a heuristika zvolila jako výhodnější stavy s větším součtem vah.

Pokud je odchylka přesně 1.0, tak heuristika našla optimální ohodnocení proměnných s největším váhovým součtem.

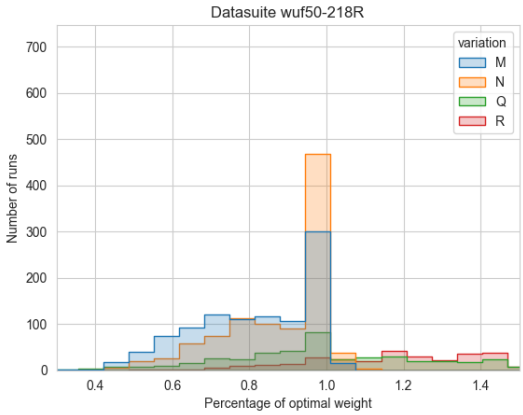
V histogramech na obrázcích 17 a 18 lze nahlédnout na procentuální počet splněných klauzulí. Na těchto grafech je vidět, že na instacích ze sady 20-71R dokázala heuristika téměř vždy nalézt splnitelné ohodnocení. Instance 50-218R mají o pár procent menší úspěšnost, hlavně varianta datové sady R.

V. Závěr

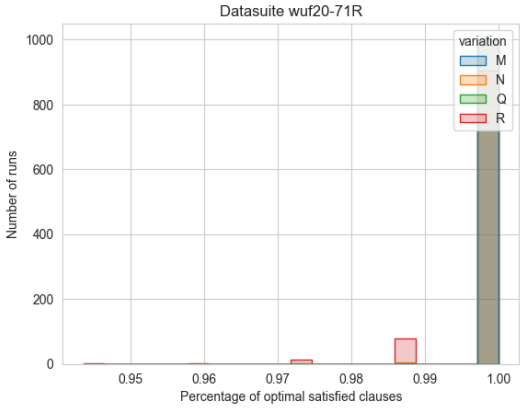
V této práci vznikla jednoduchá a lehce přepoužitelná knihovna pro souštění heuristiky simulovaného ochlazování na řešení NP-těžkého problém váženého SAT ohodnocení.



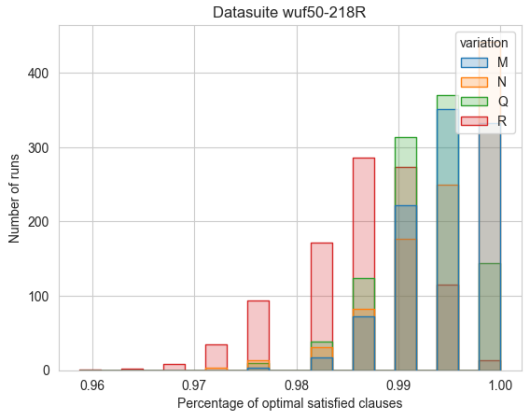
Obrázek 15: *Experiment nalezení chyby v maximální váze 20-71R*



Obrázek 16: *Experiment nalezení chyby v maximální váze 50-218R*



Obrázek 17: *Experiment nalezení chyby v maximální váze 20-71R*



Obrázek 18: *Experiment nalezení chyby v maximální váze 50-218R*

Heuristika splňuje zadání práce a relativně úspěšně řeší i složitější varianty úloh.

Implementací autor narazil na několik zajímavých poznatků a možných chyb, které při implementaci mohou nastat a opravil tyto chyby na základně experimentálního pozorování výsledků heuristiky.

Zároveň proběhlo několik experimentálních pokusů pro určení ideálních parametrů heuristky, které následně byly v black-box fázi pozorovány za přínosné.

Algoritmus do budoucích použití je vhodný spíše pro menší instance problémů, jelikož na instancích z sady 50-218R měl algoritmus úspěšnost nalezení optimálního řešení pod 50%.

Jelikož se jedná o randomizovaný algoritmus bylo provedeno několik běhů na stejných instancích a následně byly tyto výsledky započteny do finálních výsledků.

References

[1] Stoc '71: Proceedings of the third annual acm symposium on theory of computing. 1971.

[2] Jan Schmidt. Data a programy (nikop). `https://courses.fit.cvut.cz/NI-KOP/download/index.html`.