České vysoké učení technické v Praze
Fakulta informačních technologií

Kombinatorická optimalizace

První domácí úkol

Experimentální srovnání algoritmů

Autor: David Omrai (omraidav)

PRAHA, PROSINEC 2022

Contents

1 Úvod

V této práci se zaměřuji na zodpovězení otázky, která se lehce liší od té v zadání domácího úkolu. *Jaký algoritmus je nejlepší pro těžké instance při co nejmenším počtu iterací a nastavení počtu resetů na 1 a s počtem iterací, který bude zohledňovat časovou složitost i kvalitu výsledků.*

V této otázce si vymezuji nastavení algoritmů tak, aby každý běh pracoval s jedním náhodným řešením, které se pak dále snaží zlepšit. Každý výsledek tak bude reprezentovat jedno nastavení a n iterací, které jej řešily.

Tuto práci mám rozdělenou na tři hlavní části. V první popíši použitá data, použité algoritmy (jak jsem je získal). Ve druhé se zaměřím na vybrání nejlepšího nastavení počtu iterací s ohledem na kvalitu řešení i časovou náročnost a vykonám úvodní experiment, kterým si zvolím nejvíce vypovídající metriky (na omezené sadě dat). Ve třetí a hlavní části pak popíši hlavní experiment, popíši a zobrazím metriky měření a na základě získaných informací dojdu k závěru práce, jaký z algoritmů obstál v naměřených datech experimentu.

2 Popis dat a algoritmů

V této sekci budu popisovat jednotlivé části experimentů. Použitá data (instance), algoritmy a metriky.

2.1 Data

V tomto domácím úkolu se řeší problém 3-SAT. Jedná se speciální variantu problému SAT (plnitelnost booleovské formule). V této úloze jde o to zjistit, zda existuje interpretace, který by vyhovovala dané booleovské formuli. Problém 3-SAT drží určitý formát těchto formulí a to tak, že v každé klauzuli, jsou právě tři literály. Formule je v konjunktivní normální formě. Níže uvádím příklad formule této úlohy.

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

K zajištěný dat, které v této úloze využiji jsem si vybral tři sady těžších instancí z knihovny SATLIB. Ty jsou dostupné na školní webové stránce courses [1]. Jedná se o sady instancí uf20, uf50 a uf75, které mají ve svém názvu počet proměnných. Sada uf20 je rozdělena do instancí s 81 a 91 klauzulemi (pro každou z nich je zde tisíc instancí). Stejně tak má sada uf50 instance s 200 a 218 klauzulemi (rovněž je pro každou z nich zde 1000 instancí). Sada uf75 má pouze 100 instancí s 325 klauzulemi. Tento fakt využiji v prvotním experimentu.

2.2 Algoritmy

Cílem celého tohoto úkolu je srovnat dvojici algoritmů mezi sebou na těžkých 3-SAT problémech a rozhodnout, který z nich je lepším. Jedná se o algoritmus GSAT a ProbSAT.

2.2.1 GSAT

Jedná se o lokální prohledávací algoritmus k řešení problému SAT. Algoritmus začíná přiřazováním náhodných hodnot (pravda nepravda) všem proměnným formule. Pokud jsou splněny všechny klauzule, tak algoritmus končí a vrací nalezené ohodnocení. Pokud nejsou splněny, tak se algoritmus vrací zpátky na začátek, a znova ohodnotí proměnné. Při změně hodnot proměnných se snaží minimalizovat počet nesplněných klauzulí.

Tento algoritmus jsem neimplementoval, neboť nám byl dán společně s úlohou. Jedná se o verzi gsat2 dostupnou na školních stránkách courses[2].

Pro účely této úlohy budu nastavovat parametr pravděpodobnosti **p** na hodnotu **0.4**. Zároveň bylo potřeba nastavit parametr **-r time**, který zaručil, že se bude pro každý běh náhodný generátor generovat vskutku náhodná čísla.

2.2.2 ProbSAT

Tento algoritmus byl zadán ve formě pseudokódu společně s úlohou. Algoritmus má v sobě dva cykly, jeden který nastavuje náhodné ohodnocení proměnných formule (tries) a druhý, který se snaží toto ohodnocení zlepšit (flips). Oproti GSAT se při nesplnění ohodnocení všech klauzulí změní ohodnocení proměnným s pravděpodobností danou funkcí f. Tato funkce bere v potaz počet klauzulí, které se po změně hodnoty stanou splnitelnými a počet klauzulí, které se změnou hodnoty stanou nesplnitelnými. Celkově pak vzorec vypadá následovně.

$$make := pocet_nove\_splnitelnych\_klauzuli_po\_zmene\_hodnoty\_promenne\_x$$

$$break := pocet\_nove\_nesplnitelnych\_klauzuli\_po\_zmene\_hodnoty\_promenne\_x$$

$$make^{cm}/(eps + break)^{cb}$$

V rovnici si lze všimnout proměnných eps, cm a cb. Eps značí malou hodnotu, která zabraňuje dělení nulou (v programu je nastavena na hodnotu 0.0001). Cm a cb jsou fixními parametry algoritmu a jsou nastaveny na **cm=0** a **cb=2.3**.

Algoritmus jsem si pro účely tohoto úkolu napsal vlastní a nevěnoval tolik pozornosti jeho náročnosti při výpočtu větších instancí, ale o tom se zmíním až u hlavního experimentu. Algoritmus jsem psal v programovacím jazyce Python, neboť to bylo nejjednodušší a nejrychlejší. Psát jej znovu, tak bych zapřemýšlel o C++, kde by se dal celkový běh násobně zrychlit. Nicméně se tento program spouští z terminálu a je třeba mít nainstalovaný Python3. Pro spuštění stačí zadat příkaz v následujícím tvaru $./probsat.py - i < inputfile > - o < outputfile > - f < maxflips > - f < maxturns >$. Ze zmíněných přepínačů a vstupů je třeba zadat pouze **-i** společně s cestou k instanci.

3 Prvotní experiment

V prvotním experimentu se zaměřuji na část položené otázky. Tou je otázka, jak nastavit algoritmům parametr počtu opakování. Z hraní si s jednou instancí jsem si bral trojici hodnot, které budu testovat. Beru v potaz jak rychlost výpočtu tak počet splněných klauzulí.

3.1 Data

Pro prvotní experiment využívám stejné sady problému 3-SAT. Jelikož se zatím nejedná o finální experiment, budu jej spouštět nad omezeným počtem instancí každé sady. Jelikož je v sadě uf75-325 pouze stovka instancí, budu z každé sady brát tento počet. Sady uf20 a uf50 mají ale dva typy instancí, s různým počtem klauzulí. Tento fakt jsem bral v potaz a z každé z nich vzal rovnoměrné. Pro prvotní experiment jsem tedy vzal ze sady *uf20-81 prvních 50*, z *uf20-91 prvních 50*, z *uf50-200 prvních 50*, z *uf50-218 prvních 50* a z *uf75-325 prvních 100*.

3.2 Běh experimentu

Pro každou sadu jsem si tedy připravil 100 instancí s různorodým počtem parametru a klauzulí. Tím chci dosáhnout dostatečné objektivity, ale s ohledem na časovou náročnost vyhodnocení těchto instancí jednotlivými algoritmy. Celkem tedy bude provedeno 100 běhů na 100 instancích 3 datových sad pro 3 nastavení parametru max flips.

Každý algoritmus tedy poběží 3 * 30,000 krát. Vygeneruje pro každé nastavení 3 soubory po 30,000 řádcích výsledků běhů algoritmu. Tyto výsledky mají formát |**počet kroků, max. kroků, počet splněných klauzulí, celkem klauzulí**|.

Pro získání dat pro jednotlivé algoritmy a sady jsem připravil skript **pilot_run.sh**. Pro jednotlivá nastavení max flips paralelně spustí oba algoritmy. Výsledek se uloží do textových souborů
pdata_res_<date>_<max_flips>_<algorithm_name>.log.

3.3 Max flips

V předchozí podsekci jsem naznačil, že budu využívat trojice nastavení parametru max flips, tedy iterací jednotlivých algoritmů. Tyto hodnoty jsem vybral **400, 800 a 1600**. Neboli dvojnásobky předchozích hodnot. Lze očekávat, že s vyšší hodnotou bude i počet splněných klauzulí vyšší, ale co budu studovat je o kolik to je. Zdali se vyplatí tento parametr nastavovat vysoko, nebo stačí například zlatý střed.

Data jsem zpracoval a naměřil v python notebooku data_proc.ipynb.

max-flips	ProbSAT	GSAT
400	20848	19233
800	23258	21566
1600	25252	23847

Z dat je patrné, že *ProbSAT dosahuje lepších výsledků před všechny nastavení*. To nyní však neměřím. *S vyšším počtem max-flips dosahují oba algoritmy znatelně lepších výsledků*. Proto ve hlavním experimentu využiji nastavení *max-flips=1600*. Již při těchto hodnotách se ProbSAT projevil jako časově-ji náročnější, proto jsem se nepokoušel o vyšší hodnoty.

3.4 Metriky

Po nalezení vhodného ohodnocení parametru max-flips je zapotřebí si vybrat vhodné metriky, který využiji k dosažení závěru. Samostatný počet úspěšných ku neúspěšných sice určitou metrikou je, ale nebere v potaz počet kroků, průměrnou střední hodnotu, odchylku, dominanci a další. Proto se v této části prvotního experimentu zaměřím na výběr právě těchto metrik (společně s nastíněním toho, co dělají).

Pro metriky jsem využil výsledky algoritmů nad sadou uf75-325. Jedná se o nejtěžší sadu plus nastavením parametru max-flips na 1600 se jedná o výsledky, které získám i v hlavním experimentu. Nyní však krátce k použitým metrikám.

steps metrika počítá průměrný počet iterací algoritmu přes všechny instance. Jednotlivé opakování na instancích jsou zprůměrována.

fined steps metrika počítá průměrný počet iterací algoritmu přes všechny běhy. Jediným rozdílem oproti metrice steps je, že neúspěšnému běhu je zvýšen jeho počet iterací 10násobně. Tímto přístupem se snáze v grafu projeví kvalita algoritmů.

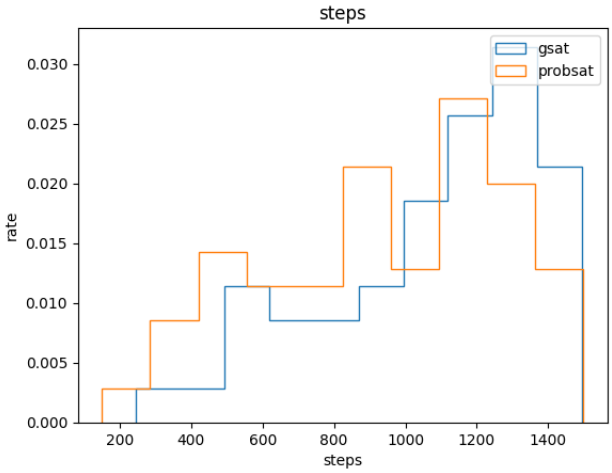
succ je jednoduchou metrikou, která počítá pro každý běh na jedné instanci počet, kdy algoritmus nalezl úspěšné řešení.

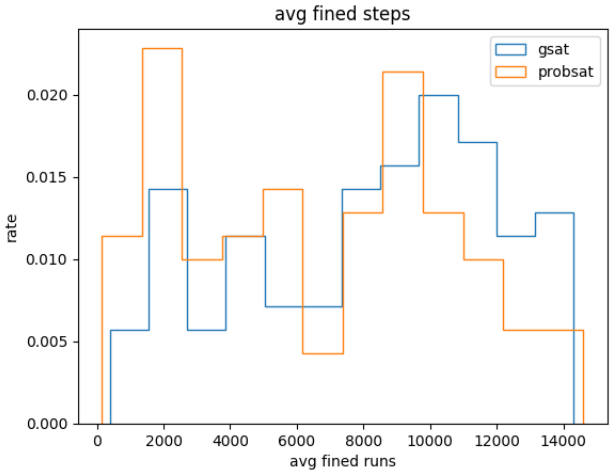
μ je parametrem lognormálního rozložení. Jedná se o odhad střední hodnoty pomocí průměru logaritmů běhů.

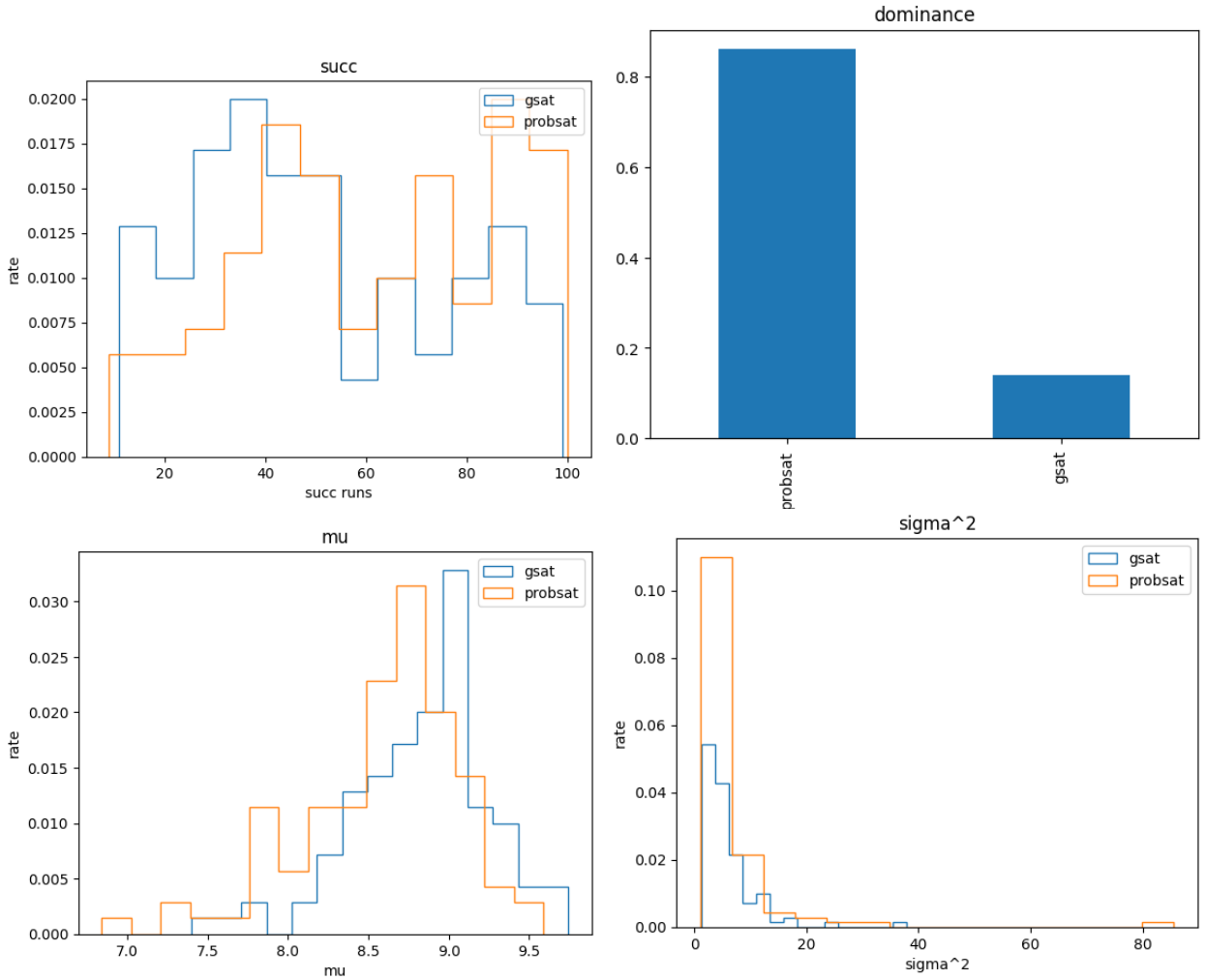
σ^2 je dalším parametrem lognormálního rozložení. Jedná se o odhad hodnoty rozptylu.

dominance je metrikou, kterou jsem si sám vytvořil. Využívám pro ni korigovanou CDF a pro každé opakování nad jednotlivými instancemi jednotlivých algoritmů počítám kolikrát jedna druhé dominovala. Tyto informace po každém běhu algoritmů nad jednotlivými instancemi spočítám a po poslední porovnám kolikrát jaká té druhé dominovala.

V následující části se zaměří na využití představených metrik nad výsledky běhů algoritmů nad sadou uf75-325 s nastavením max-flips na 1600.







Z jednotlivé grafy prezentují různé informace o jednotlivých algoritmech. Metrika steps ukazuje, že GSAT průměrně vyžaduje méně kroků k nalezení řešení. Namísto toho ProbSAT spíše vyžaduje více, neboť jeho nejčastější hodnota počtu běhů se pohybuje okolo 1,300. Nicméně po nahlédnutí do grafu metriky avg steps se tento závěr obrací a jde vidět, že namísto toho spíše ProbSAT dosahuje lepších výsledků. Problém na který tento graf naráží je že neúspěšné běhy mohou jednak klamat, neboť jejich počet je roven maximálnímu počtu běhů a mohou být běhy, kde se maximální v maximálním počtu běhů, nebo u něj nalezne řešení.

Grafy metrik succ a dominance ukazují, že algoritmus ProbSAT častěji problém vyřeší v jednotlivých opakování nad instancemi. Dominance ukazuje značný rozdíl a tedy i dominance ProbSATu nad GSATem.

V poslední části porovnávám metriky odhadů parametrů lognormálního rozdělení. Lze z nich vypozorovat, že algoritmus GSAT má značně menší deviace od střední hodnoty skrz všechny instance. U rozptylů algoritmu ProbSAT je viditelný outliner okolo hodnoty 80.

4 Hlavní experiment

V tomto experimentu získávám a evaluuji výsledky z běhů jednotlivých algoritmů na větším množství instancí a s více opakováním.

4.1 Data

Pro tento experiment jsem si vybral vždy ty těžší varianty instancí z jednotlivých sad problémů. Budu využívat datové sady uf20-91, uf50-218 a uf75-325. Všechny tyto sady využiji celé, což dělá 1,000 instancí z uf20-91, 1,000 instancí z uf50-218 a 100 instancí z uf75-325. Celkem tedy 2,100 těžkých instancí.

4.2 Běh experimentu

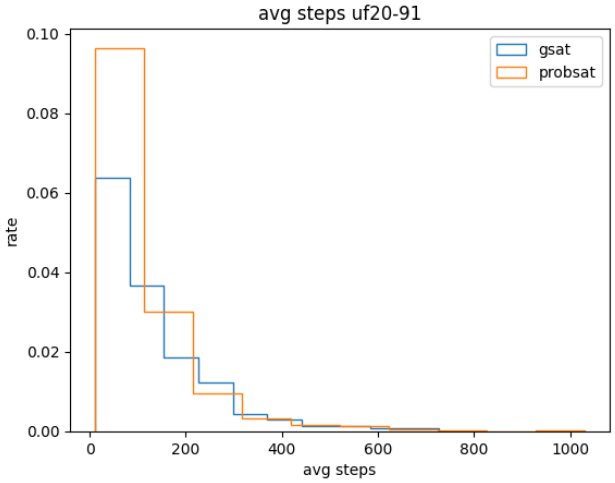
Oba algoritmy byly spuštěny nad jednotlivými instancemi **500krát**. Toto nastavení jsem zvolil abych celkem získal přes milión výsledků a zároveň algoritmy neběžely příliš dlouho.

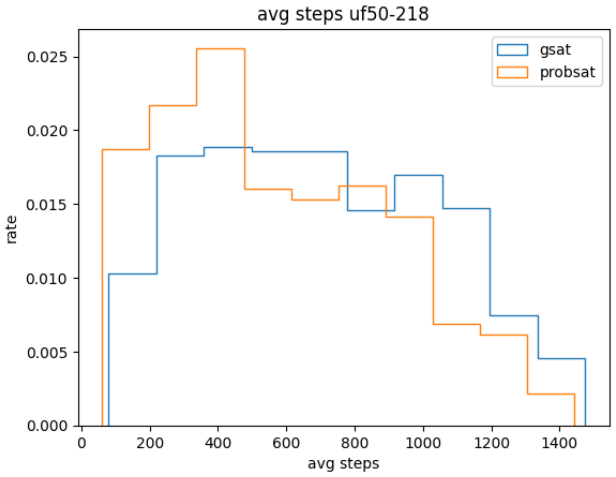
Jelikož jsem algoritmus ProbSAT psal sám, a ne příliš bral v potaz jeho komplexitu, tak se to v této části značně projevilo. Pro rychlejší běh experimentu jsem každý algoritmus spustil paralelně a to ještě na čtvrtině instancí každé sady. Celkem tedy běželo 18 procesů počítajíce problémy. Algoritmus GSAT byl značně rychlejší a jeho poslední část doběhla okolo hodinky. Namísto toho u ProbSATu doběhl poslední běh po dni. Zpětně bych se snažil o to zdokonalit program, ale s již naměřenými daty jsem nechtěl vše zahazovat a začínat od začátku. Avšak v tomto úkolu neměřím časovou náročnost, ale krokovou.

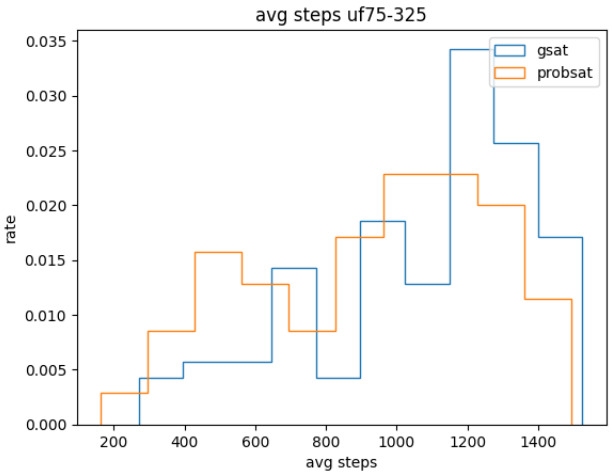
Pro každou instanci se tedy provedlo **500 běhů**, celkem bylo **2,100 instancí**, což dělá celkem **1.05 miliónů běhů** (pro každý algoritmus).

4.3 Metriky

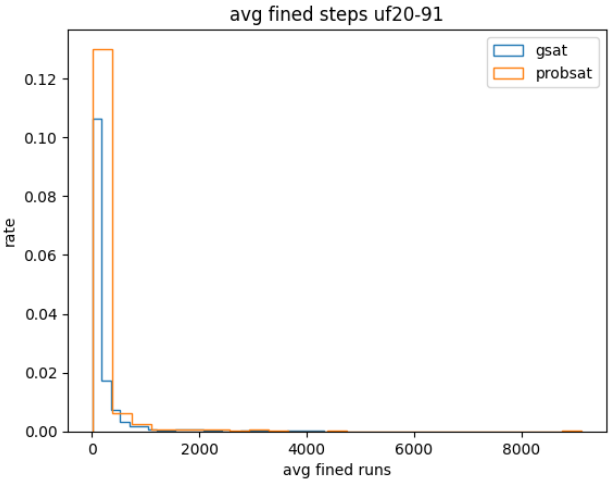
4.3.1 avg steps

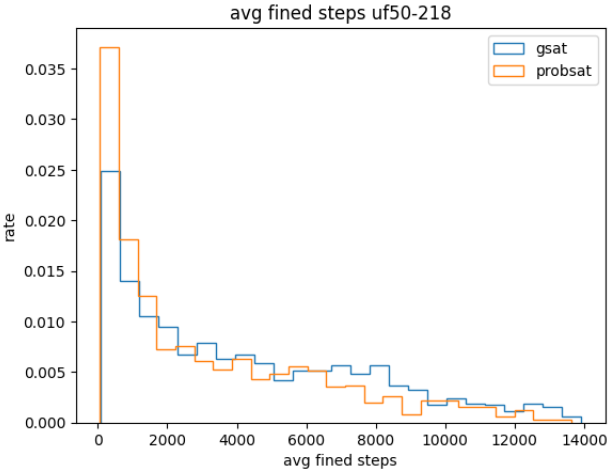


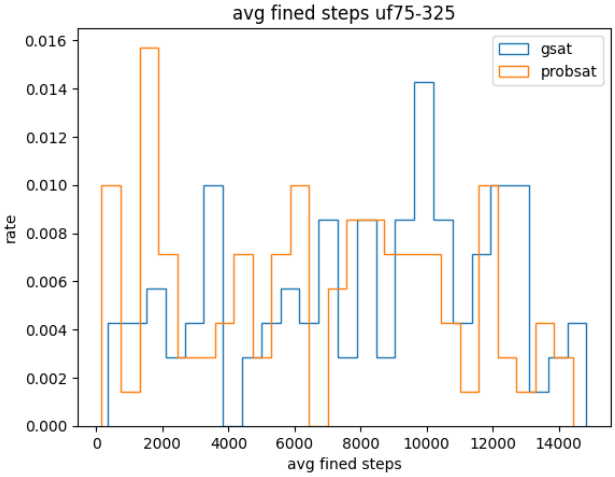




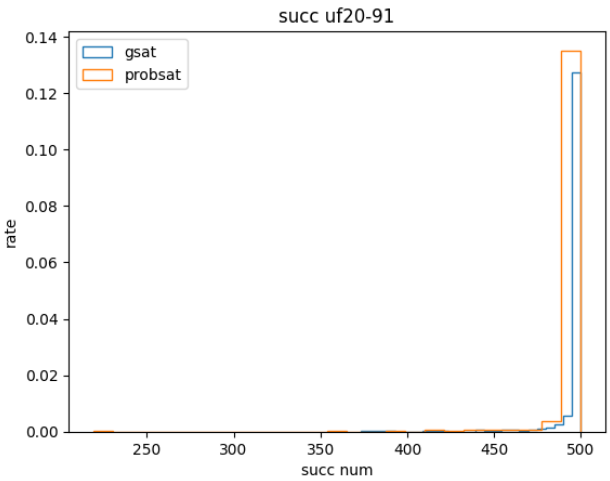
4.3.2 avg fined steps

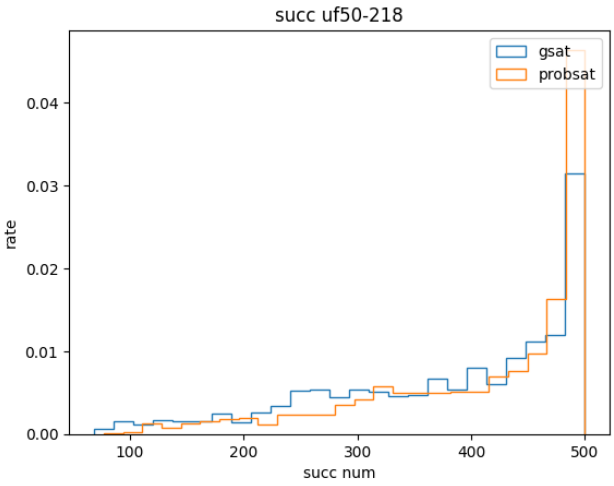


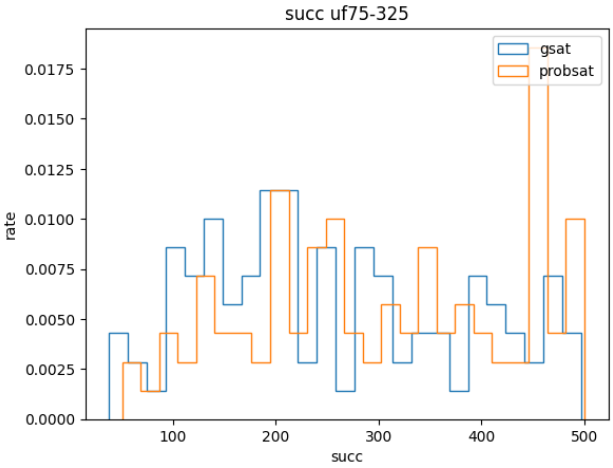




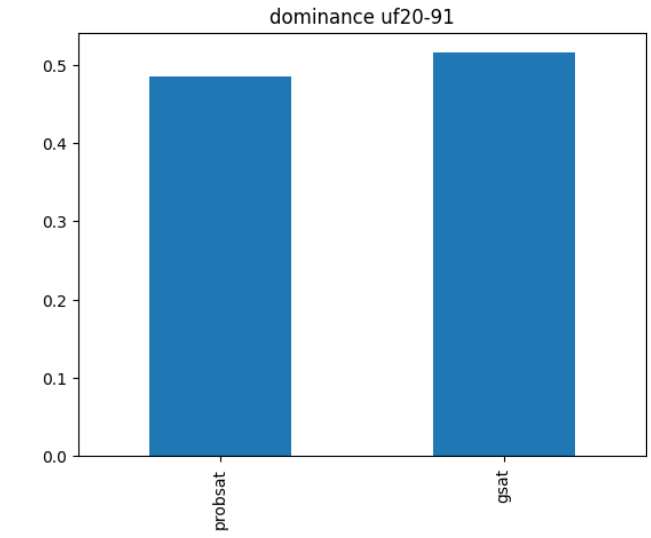
4.3.3 succ

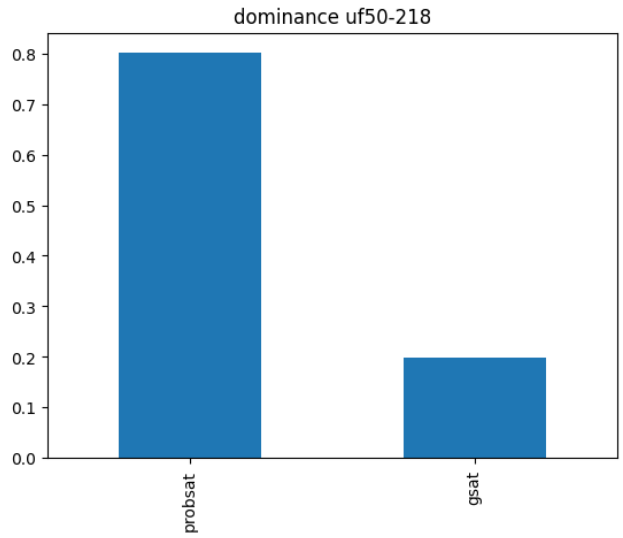

succ uf20-91
gsat
probsat
rate
succ num
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
250
300
350
400
450
500

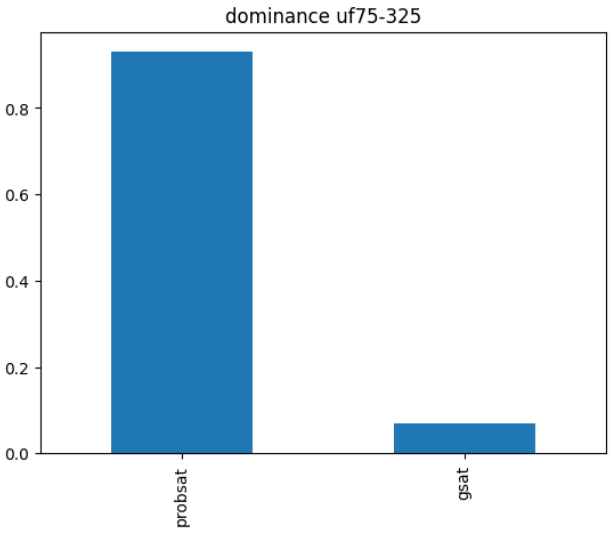

succ uf50-218
gsat
probsat
rate
succ num
0.04
0.03
0.02
0.01
0.00
100
200
300
400
500


succ uf75-325
gsat
probsat
rate
succ
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
100
200
300
400
500

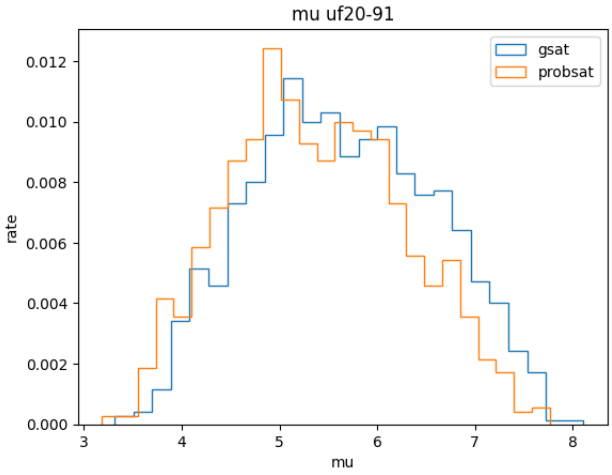
4.3.4 dominance

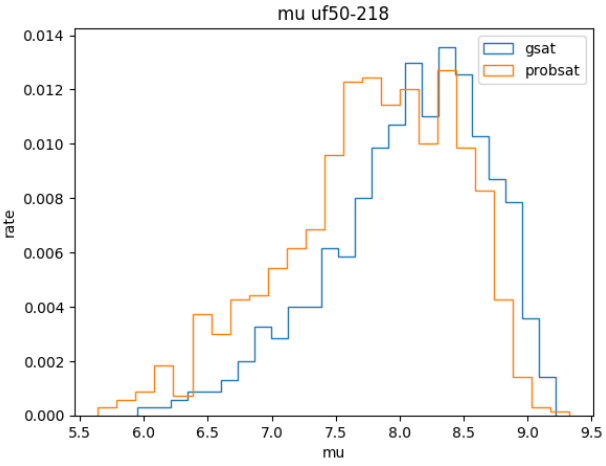


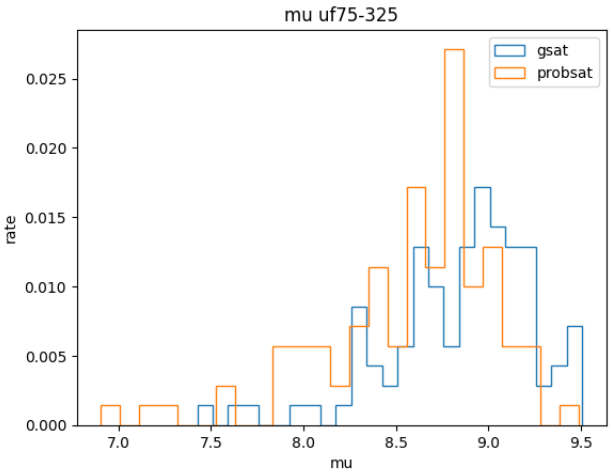




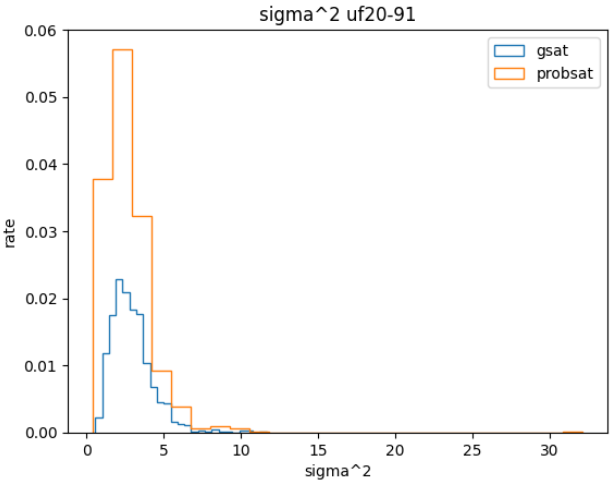
4.3.5 μ

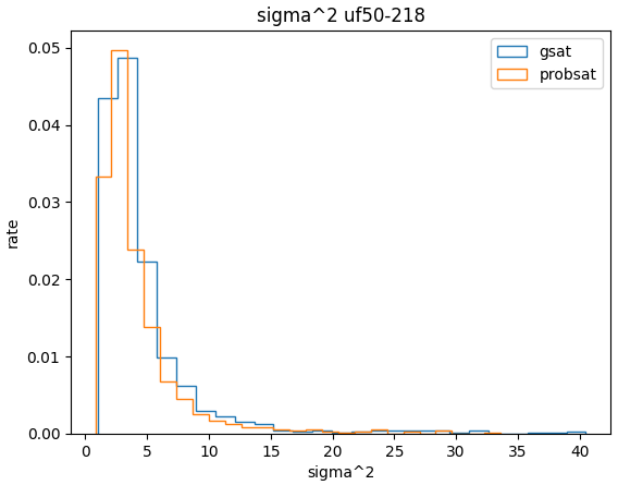

mu uf20-91
gsat
probsat
rate
mu
0.000
0.002
0.004
0.006
0.008
0.010
0.012
3
4
5
6
7
8

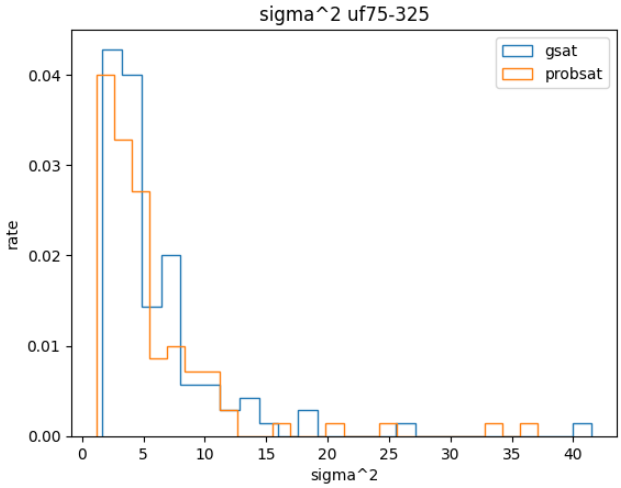

mu uf50-218
gsat
probsat
rate
mu
0.000
0.002
0.004
0.006
0.008
0.010
0.012
0.014
5.5
6.0
6.5
7.0
7.5
8.0
8.5
9.0
9.5


mu uf75-325
gsat
probsat
rate
mu
0.000
0.005
0.010
0.015
0.020
0.025
7.0
7.5
8.0
8.5
9.0
9.5

4.3.6 σ^2


sigma^2 uf20-91
gsat
probsat
rate
sigma^2
0.06
0.05
0.04
0.03
0.02
0.01
0.00
0
5
10
15
20
25
30


sigma^2 uf50-218
gsat
probsat
rate
sigma^2
0.05
0.04
0.03
0.02
0.01
0.00
0
5
10
15
20
25
30
35
40


sigma^2 uf75-325
gsat
probsat
rate
sigma^2
0.04
0.03
0.02
0.01
0.00
0
5
10
15
20
25
30
35
40

4.4 Evaluace metrik

Z dosažených výsledků lze předběžně vykoukat, že si algoritmus ProbSAT vedl lépe, i když na většině metrik měl dosti blízké výsledky GSATu. Dále i to, že metrika μ rozhodování, který z nich je lepší, nepomohla. Nyní se však zaměřím na jednotlivé metriky a porovnám výsledky algoritmů mezi sebou.

steps metriku lze nejlépe pozorovat na sadách uf50-91 a uf50-218. Zde je vidět, že algoritmus ProbSAT vyžaduje menší průměrný počet kroků k nalezení řešení. Na sadě uf75-325 je tento fakt o něco méně viditelný, ale stále pozorovatelný.

fined steps metrika není na sadě uf50-91 moc neprojevuje. Pro oba algoritmy platí, že mají spíše podobný průměrný počet kroků s tím, že se u nich objevují i outliners. Znatelnější rozdíly lze pozorovat na instancích sady uf50-218, kde ProbSAT opět značně vyhrává. GSAT se drží u menších průměrných krocích pod ním a převyšuje u vyšších hodnot. Na sadě uf75-325 lze pozorovat, že ProbSAT má spíše průměrné běhy menší.

succ metrika je na sadě uf50-91 spíše neprůkazná, oba algoritmy si drží podobný počet vyřešení problému. Větší rozdíly lze pozorovat na instancích sady uf50-218, kde ProbSAT ukazuje, že dokáže vyřešit problém spíše více krát nežli GSAT. Nakonec u sady uf75-325 se opět ukazuje ProbSAT jakožto lepší, kopíruje či převyšuje výsledky GSATu.

μ metrika se ukázala jako neprůkazná na všech sadách problému. Výsledky algoritmů jsou na ní dosti podobný.

σ^2 metrika ukazuje, že algoritmus ProbSAT dosahuje na sadě uf50-91 dosti lepších výsledků, majíce menší odchylku. U ostatních sad jsou výsledky neprůkazné.

dominance metrika ukazuje, že na sadě uf50-91 jsou výsledky GSATu a ProbSATu dosti podobné (GSAT lehce lepší). Na ostatních sadách však dominuje znatelně ProbSAT. Tato dominance je nejvíce viditelná na sadě uf75-325.

5 Závěr

V první části tohoto úkolu jsem se zaměřil na část položené otázky. Tou bylo jaký nastavení parametru max-flips je nejlepší pro tuto úlohu. Z časového i výkonnostního hlediska jsem usoudil, že právě hodnota 1,600.

Hlavní otázkou je pak, který z algoritmů je nejlepší pro těžké instance, při co nejmenším počtu iterací. Právě co nejmenší počet iterací je zadání součástí, kterou bylo třeba vzít v úvahu. Z naměřených dat a jejich reprezentaci několika metrikami lze usoudit, že algoritmus ProbSAT si vedl o poznání lépe. Na metrice steps a fined steps je třeba pozorovat, jak jsou rozloženy hodnoty kroků. Pro všechny běhy se ukázal právě ProbSAT jako lepší, má spíše nižší hodnoty než GSAT. Z menšího počtu kroků lze vyčíst, že algoritmus nalezl výsledek. Převažovaly by hodnoty blízké maximálnímu počtu kroků, tak by se nejednalo o kladnou indikaci kvality nalezení výsledku. Poměr toho, jak úspěšný každý z algoritmů byl se nachází v metrikách succ a dominance. U metriky succ lze pozorovat, ze opět ProbSAT spíše vyřešil v každém běhu instanci vícekrát, nežli algoritmus GSAT. Metrika dominance pak ukazuje, že ve skoro všech sadách byl ProbSAT dominantní GSATu.

Z pohledu výsledků zkoumaných metrik tedy usuzuji, že algoritmus ProbSAT i přes svou časovou náročnost je úspěšnější. ***Nejlepší algoritmus pro těžké instance při co nejmenším počtu iterací a nastavení počtu resetů na 1 a s počtem iterací, který zohledňuje časovou náročnost i kvalitu výsledků je ProbSAT***.

References

[1] Jan Schmidt. Data a programy. https://courses.fit.cvut.cz/NI-KOP/download/#_3-sat-instance-z-knihovny-satlib-splniteln%C3%A9. Accessed: 2022-12-06.

[2] Jan Schmidt. Programy. https://courses.fit.cvut.cz/NI-KOP/download/index.html#_programy. Accessed: 2022-12-06.