# Rapport de projet - DSL

Nicolas Bailluet, Samuel Bouaziz

Janvier 2020

## 1 Introduction

### 1.1 Corpus de test

Nous avons réalisé notre *benchmark* en utilisant les fichiers de test disponible sur le dépôt suivant : `https://github.com/diverse-project/samplingfm/tree/master/Benchmarks`. En raison du temps d'exécution important des tests sur certains fichiers, nous avons effectué nos mesures sur les 47 premiers fichiers à la racine du dépôt. De plus, un *timeout* de une heure a été imposé pour chaque instance.

### 1.2 Environnement d'exécution

Les tests ont été effectués dans l'environnement suivant :
- CPU : Intel(R) Core(TM) i5-4200U CPU @ 1.60GH
- RAM : 8Go DDR3
- Swap : 16Go SSD M.2
- IDE : Eclipse

Un coeur entier du processeur était dédié au solveur en cours d'exécution.

### 1.3 Solveurs

Le *benchmark* a été réalisé en utilisant les solveurs suivants :
- Sat4J : jar standalone et bibliothèque java, version 2.3.1 ;
- Minisat : jar standalone, paramètre `rnd-freq = 0.2, 0.5, 1.0`, version 2.2.0 ;
- Cryptominisat : jar standalone, version 5.6.8.

## 2 Benchmarks

### 2.1 Sat4J jar vs bibliothèque Java

Les temps d'éxecution de sat4j-java et sat4j-jar sont très proches. On pouvait s'attendre à un coût en plus pour la version jar, le temps de lancer le jar et c'est ce qui est observé.

| résultat | sat4j-java | sat4j-jar |
|---|---|---|
| sat | 44 | 44 |
| unsat | 2 | 2 |
| timeout | 1 | 1 |

FIGURE 1 – Comparaison des résultats obtenus avec sat4j-java et sat4j-jar.

| temps d'éxecution (s) | sat4j-java | sat4j-jar |
|---|---|---|
| moyen | 110,1 | 110,8 |
| minimum | 0,001 | 0,152 |
| maximum | 3600 | 3601 |

FIGURE 2 – Comparaison du temps d'éxecution de sat4j (en secondes) en fonction de la version java ou jar.

On observe aussi qu'un test (`listReverse.sk_11_43.cnf`) a *timeout* pour les deux versions du solveur, c'est donc le temps qui correspond au temps d'exécution le plus élevé (une heure).

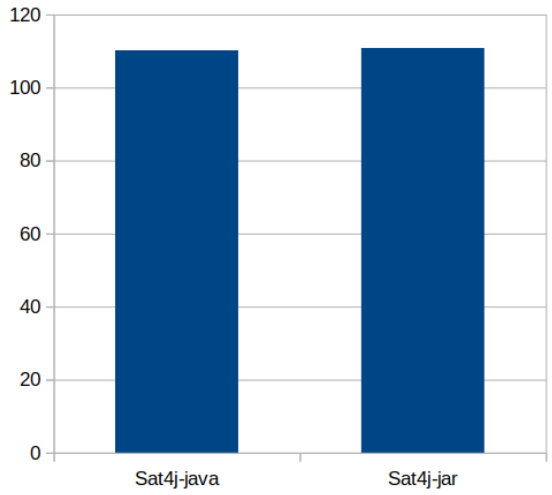


FIGURE 3 – Diagramme de comparaison du temps d'exécution moyen de sat4j-java (en secondes) et sat4j-jar.

## 2.2 Minisat `rnd-freq 0.2` vs `0.5` vs `1.0`

| résultat | minisat rnd-freq 0.2 | minisat rnd-freq 0.5 | minisat rnd-freq 1.0 |
|---|---|---|---|
| sat | 45 | 45 | 44 |
| unsat | 2 | 2 | 2 |
| timeout | 0 | 0 | 1 |

FIGURE 4 – Comparaison de minisat en fonction de l'option rnd-freq.

| temps d'éxecution (s) | minisat rnd-freq 0.2 | minisat rnd-freq 0.5 | minisat rnd-freq 1.0 |
|---|---|---|---|
| moyen | 9,542 | 51,60 | 113,9 |
| minimum | 0,004 | 0,004 | 0,003 |
| maximum | 314,6 | 2251 | 3605 |

FIGURE 5 – Comparaison du temps d'éxecution de minisat (en secondes) en fonction de l'option rnd-freq.

D'après les temps d'éxecution obtenus, minisat est beaucoup plus efficace avec l'option `rnd-freq 0.2`.

En plus d'avoir l'exécution la plus longue en moyenne, minisat avec l'option `rnd-freq 1.0` est la seule version qui a *timeout* sur un des tests (`listReverse.sk_11_43.cnf`), ce qui n'est pas surprenant sachant que l'option `rnd-freq` définit la proportion de choix faits au hasard dans la recherche de solutions.

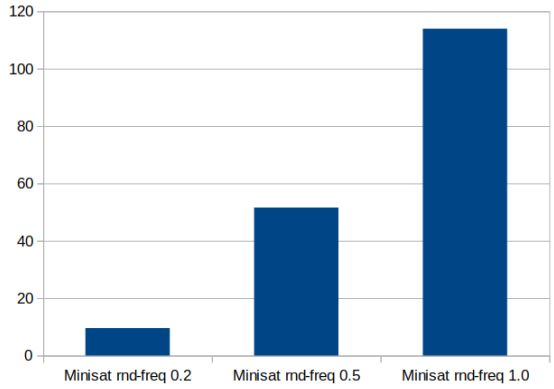


FIGURE 6 – Diagramme de comparaison du temps d'éxecution moyen de minisat (en secondes) en fonction de l'option `rnd-freq`.

## 2.3 Comparaison générale

On considère, pour la comparaison globale, les meilleurs résultats obtenus pour chaque solveur, c'est-à-dire `rnd-freq 0.2` pour minisat et `sat4j-java` pour Sat4J.

D'après la Fig. 7, on observe qu'en moyenne le solveur le plus rapide est minisat, suivi de peu par cryptominisat et enfin (loin derrière) Sat4J.

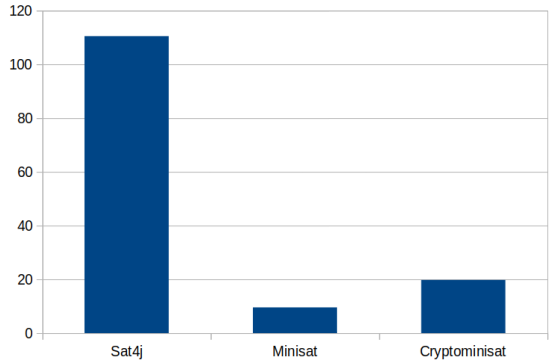


FIGURE 7 – Diagramme de comparaison du temps d'éxecution moyen (en secondes) en fonction du solveur.

Enfin, on observe sur les matrices de confusion (Fig. 10, Fig. 11 et Fig. 12) que hormis les *timeouts*, les résultats sont cohérents entre les différents solveurs testés, c'est-à-dire qu'il n'y a pas de désaccord sur la satisfiabilité d'une des formules testées.

| résultat | sat4j | minisat | cryptominisat |
|---|---|---|---|
| sat | 44 | 45 | 45 |
| unsat | 2 | 2 | 2 |
| timeout | 1 | 0 | 0 |

FIGURE 8 – Comparaison générale des résultats obtenus.

| temps d'éxecution (s) | sat4j | minisat | cryptominisat |
|---|---|---|---|
| moyen | 110,5 | 9,542 | 19,77 |
| minimum | 0,001 | 0,003 | 0,005 |
| maximum | 3601 | 314,6 | 616,4 |

FIGURE 9 – Comparaison générale des résultats obtenus (résultats en secondes).

| | sat4j SAT | sat4j UNSAT | sat4j TIMEOUT |
|---|---|---|---|
| minisat SAT | 44 | 0 | 1 |
| minisat UNSAT | 0 | 2 | 0 |
| minisat TIMEOUT | 0 | 0 | 0 |

FIGURE 10 – Matrice de confusion des résultats obtenus avec sat4j et minisat.

| | minisat SAT | minisat UNSAT | minisat TIMEOUT |
|---|---|---|---|
| cryptominisat SAT | 45 | 0 | 0 |
| cryptominisat UNSAT | 0 | 2 | 0 |
| cryptominisat TIMEOUT | 0 | 0 | 0 |

FIGURE 11 – Matrice de confusion des résultats obtenus avec cryptominisat et minisat.

| | sat4j SAT | sat4j UNSAT | sat4j TIMEOUT |
|---|---|---|---|
| cryptominisat SAT | 44 | 0 | 1 |
| cryptominisat UNSAT | 0 | 2 | 0 |
| cryptominisat TIMEOUT | 0 | 0 | 0 |

FIGURE 12 – Matrice de confusion des résultats obtenus avec sat4j et cryptominisat.

# 3 Conclusion

Les résultats montrent que les solveurs sont d'accord et permettent de comparer leurs performances, cependant le nombre de tests effectués est faible (47 fichiers).

On pourrait utiliser une machine plus puissante et adaptée à l'exécution d'un *benchmark* long et gourmand en ressources. On peut penser ainsi à louer une instance Amazon AWS ou encore utiliser un serveur dédié ou VPS.

Il serait également plus simple d'avoir un environnement d'exécution plus adapté au lieu de lancer les tests en tant que test jUnit dans l'IDE Eclipse.

Enfin il serait intéressant de tester d'autres potentielles options de minisat et cryptominisat, comme le nombre de *threads* utilisés qui pourrait grandement réduire le temps d'exécution sur des processeurs multicoeurs.