– DSL –
Milestone 6 : *Testing SAT solver variants*

Kerian Thuillier

Janvier 2020

Résumé

Rapport pour le *Milestone 6* du module DSL de M1 SIF du binôme Kerian Thuillier et Alex Coudray. Il contient une analyse des résultats (temps de calcul, qualité de la réponse) obtenus pour plusieurs solveurs SAT avec différents paramètres et version.

Table des matières

Introduction

Compte rendu du *Milestone 6* du module DSL, il s'agit d'une analyse des résultats obtenus sur un benchmark de formules SAT sous forme CNF. Le benchmark a été exécuté 10 fois pour chacun des solveurs, solveurs appelés avec différentes versions et différents paramètres.

Dans ce rapport, nous répondons aux questions suivantes :

— Quel est le meilleur solveur ?
— Y a-t-il des différences entre les solveurs ? Sont-elles dues à des bogues ?
— Y a-t-il des formules SAT plus dures que les autres ?
— Peut-on considérer uniquement un sous-ensemble du benchmark pour réaliser notre étude ?

1 Condition d'expérimentation

1.1 Solveurs

Les solveurs utilisés sont ceux demandés pour ce cours, c'est-à-dire SAT4J, MINISAT et CRYPTOMINISAT. Ces solveurs ont été utilisés sous différentes versions :

SAT4J Ce solveur a été exécuté de trois façons différentes : directement avec la librairie JAVA, grâce à un appel du fichier JAR associé et par un appel à MAVEN. De plus, plusieurs versions de SAT4J ont été utilisées, notamment les versions 2.0.0, 2.0.4, 2.2.3 et 2.3.1. Notons toutefois que seule la version 2.3.1 a utilisé l'appel via la librairie JAVA. En effet, nous n'avons pas réussi à modifier dynamiquement la librairie associée au projet. La consigne étant d'avoir un code automatisé, nous n'avons donc pas modifié manuellement le lien à cette librairie.

MINISAT Nous avons utilisé la version 2.2.0 de ce solveur. Seule cette version a été testée, car nous n'avons pas réussi à installer deux versions de MINISAT simultanément sur notre machine. Nous avons cependant appelé MINISAT avec différentes valeurs du paramètre *rnd-freq* : 0.01, 0.2, 0.4, 0.6, 0.8, 0.99. Bien que la documentation nous explique qu'une valeur proche de 0 pour *rnd-freq* est plus efficace, nous avons tenu à tester un panel plus large de valeurs.

CRYPTOMINISAT Pour ce solveur, nous avons employé la version 5.7.8. CRYPTOMINISAT. N'ayant pas de paramètres particuliers et n'arrivant toujours pas à avoir plusieurs versions différentes de ce solveur sur notre machine, nous avons donc uniquement testé cette version.
Notons toutefois que nous avons tenté d'installer différentes versions de CRYPTOMINISAT. Nous avons eu une erreur de compilation chaque fois que nous essayions de suivre les instructions de compilation du GIT [1]. D'autres groupes ont essayé de nous débloquer, sans comprendre la nature de cette erreur. Nous avons donc dû abandonner l'idée de tester plusieurs versions de CRYPTOMINISAT.

1.2 Benchmark

Afin de tester nos différents solveurs, nous avons utilisé le benchmark fourni avec le projet et disponible à l'adresse suivante `https://github.com/diverse-project/samplingfm/tree/master/Benchmarks`.

1. `https://github.com/msoos/cryptominisat`

Seuls les éléments présents à la racine ont été exécutés, soit 70 formules SAT. Les formules des sous-dossiers n'ont pas été utilisées par manque de temps, le temps de calcul nécessaire pour exécuter le benchmark 10 fois étant considérable.

Chacun de ces 70 éléments a été exécuté 10 fois pour chaque solveur. Seules deux instances n'ont pas été exécutées jusqu'au bout : *diagStencilClean.sk_41_36.cnf* et *listReverse.sk_11_43.cnf*. Leurs exécutions n'ont pas terminé après plus de 10 heures de calcul. Ces instances n'ont été exécutées que sur un sous-ensemble des solveurs, il est donc impossible de les utiliser pour la comparaison.

1.3 Conditions

Les résultats ont été obtenus en exécutant notre implémentation sur une machine sous *Ubuntu 18.04.3 LTS - 64bits* avec *Gnome 3.28.2*. Le système dispose de 8 Go de mémoire et d'un processeur *Intel® Core™ i5-8300H CPU @ 2.30GHz × 8*. Le benchmark a été exécuté d'un seul bloc.

2 Analyse des résultats

Les données obtenues sont fournies avec ce projet sous la forme de deux fichiers CSV :

benchmark-10x.csv qui contient les temps moyens et les écarts-types des 10 exécutions de chaque solveur pour chaque instance.

benchmark_data.csv qui contient des informations sur les différentes instances : nombre de variables, nombre de clauses et pourcentage des clauses n'ayant qu'un seul littéral.

Le script R utilisé pour traiter les données CSV est également fourni avec ce rapport.

2.1 Meilleur solveur

Il y a plusieurs façons de définir le meilleur solveur : le meilleur en temps de calcul ou le meilleur en qualité de la solution.

Intéressons-nous tout d'abord à l'étude de la qualité des solutions. Grâce à un script R fourni avec le projet, nous avons pu constater que les solveurs retournent toujours des solutions en accord les uns avec les autres.

Ne pouvant nous comparer les solveurs grâce à la qualité des solutions fournies, nous devons nous intéresser au temps de calcul nécessaire à chaque solveur pour résoudre les instances. Pour cela, nous avons assigné un *rang* à chaque solveur pour chaque instance. Nous avons commencé par déterminer l'intervalle de confiance à 95% du temps de calcul moyen. Rappelons la formule de calcul d'intervalle à 95% (on considère que les temps de calcul suivent une distribution normale) :

$$\left[\text{mean} - 1.95 \times \frac{\text{standard deviation}}{\sqrt{10}}\right]$$

Les solveurs sont ensuite classés du plus lent au plus rapide pour chacune des instances grâce aux bornes inférieure et supérieure de cet intervalle. Le rang d'un solveur pour une instance donnée est la moyenne de la position obtenue avec la borne inférieure et avec la borne supérieure. Un exemple est disponible tableau 1.

Nous pouvons ensuite calculer les rangs moyens pour chaque solveur. Nous obtenons les résultats présentés dans le tableau 2. Nous pouvons constater que les rangs moyens de MINISAT, quelle que soit la valeur de *rnd-freq*, restent assez proches les uns des autres. Le meilleur

Solveur	Version	Borne Inférieure (ms)	Borne Supérieure (ms)	Rang
SAT4J_JAVA	2.3.1	3.43	30.57	8
SAT4J_JAR	2.0.0	70.6	85.4	10.5
SAT4J_JAR	2.0.4	67.83	80.17	9.5
SAT4J_JAR	2.2.3	72.3	79.7	10
SAT4J_JAR	2.3.1	73.28	96.72	12
SAT4J_COMP	2.0.0	1441.85	1512.15	16
SAT4J_COMP	2.0.4	1431.76	1508.24	15
SAT4J_COMP	2.2.3	1266.78	1327.22	13
SAT4J_COMP	2.3.1	1268.47	1337.53	14
Cryptominisat	5.7.8	2.22	15.78	6.5
Minisat rnd-frq=0.01	2.2.0	-0.87	18.87	4
Minisat rnd-frq=0.2	2.2.0	0.77	3.23	1.5
Minisat rnd-frq=0.4	2.2.0	0.77	3.23	1.5
Minisat rnd-frq=0.6	2.2.0	0.77	3.23	1.5
Minisat rnd-frq=0.8	2.2.0	0.77	3.23	1.5
Minisat rnd-frq=0.99	2.2.0	0.77	3.23	1.5

TABLE 1 – Calcul du rang pour chacun des solveurs pour l'instance *10.sk_1_46.cnf*.

temps de calcul est obtenu pour *rnd-freq* fixé à 0.01. Nous pouvons même constater que plus la valeur de *rnd-freq* est élevée, plus le solveur est lent. Ce qui confirme ce que nous avions vu dans la documentation de MINISAT.

Le solveur ayant le meilleur temps de calcul moyen est SAT4J_JAVA. Ce résultat n'est pas forcément étonnant : ce solveur est le seul utilisé comme librairie JAVA. Les autres solveurs sont tous appelés via des commandes systèmes réalisées depuis notre programme, ce qui nécessite la création de terminaux. En prenant ce fait et le fait que les rangs moyens entre SAT4J_JAVA et MINISAT ne diffèrent que de 1.5 points en compte, il est possible de conclure que MINISAT est le meilleur des solveurs. Plus précisément, il s'agit de MINISAT version 2.0.0 avec *rnd-freq* valant 0.01.

2.2 Différences entre solveurs

Dans cette section, nous nous intéressons aux différences entre les solveurs. Comme nous l'avons vu dans la section précédente, les solveurs sont tous en accord les uns avec les autres sur la satisfiabilité des formules. Il ne semble donc pas avoir de bogue fonctionnel des solveurs sur ces instances.

Nous pouvons néanmoins nous intéresser aux différences de performance, en termes de temps de calcul, des solveurs. Pour cela, nous avons associé à chaque solveur pour chaque instance le taux de différence entre le temps de calcul du solveur résolvant l'instance le plus rapidement, et le temps de calcul du solveur étudié. Par exemple, si CRYPTOMINISAT est le plus performant sur une instance i donnée avec un temps de calcul de 50ms, alors MINISAT RND-FREQ=0.01 qui a un temps de calcul de 75ms aura 50% de différence avec CRYPTOMINISAT.

Nous allons nous intéresser à deux comparaisons différentes des performances :

— comparaison de l'ensemble des solveurs et de l'ensemble des solveurs sans SAT4J_JAVA pour ne considérer que des solveurs appelés de la même façon par notre implémentation. Cela permet de ne pas avoir le biais induit par l'instanciation et l'utilisation d'un terminal de commande.

— comparaison des différentes versions de SAT4J pour l'appel avec le fichier JAR et par

Solveur	Version	Rang moyen	Rang final
SAT4J_JAVA	2.3.1	2.13	1
SAT4J_JAR	2.0.0	9.93	9
SAT4J_JAR	2.0.4	10.12	10
SAT4J_JAR	2.2.3	10.32	11
SAT4J_JAR	2.3.1	11.06	12
SAT4J_COMP	2.0.0	15.32	15
SAT4J_COMP	2.0.4	15.43	16
SAT4J_COMP	2.2.3	13.23	13
SAT4J_COMP	2.3.1	13.34	14
Cryptominisat	5.7.8	4.51	7
Minisat rnd-frq=0.01	2.2.0	3.66	2
Minisat rnd-frq=0.2	2.2.0	3.75	3
Minisat rnd-frq=0.4	2.2.0	3.8	4
Minisat rnd-frq=0.6	2.2.0	3.83	5
Minisat rnd-frq=0.8	2.2.0	4.1	6
Minisat rnd-frq=0.99	2.2.0	4.98	8

TABLE 2 – Rangs moyens et finaux obtenus pour chacun des solveurs. Les rangs finaux sont obtenus en ordonnançant les rangs moyens.

MAVEN. Cela nous permet de voir si la performance de SAT4J s'est améliorée entre les versions ou bien si un bogue de performance est apparu.

2.2.1 Comparaison de l'ensemble des solveurs

Commençons par nous intéresser aux différences de performance en considérant l'ensemble des solveurs. Le tableau 3 résume les résultats obtenus : les différences moyennes et médianes. Nous étudions la médiane car elle nous permet de nuancer l'analyse de la moyenne qui est fortement impactée par les valeurs extrêmes. Comme précédemment, nous constatons que SAT4J_JAVA obtient les meilleurs résultats et ce, sur les deux métriques. Nous remarquons également que les solveurs SAT4J_JAR et SAT4J_COMP (MAVEN) ont des problèmes de performances, ils mettent respectivement plus de 2 000% et plus de 15 000% de temps que le meilleur solveur pour chaque instance. Ces résultats peuvent être normaux pour les solveurs appelés par MAVEN, la durée d'exécution prenant en compte le temps nécessaire pour exécuter l'environnement MAVEN. Cependant, les résultats obtenus avec SAT4J_JAR sont plus étranges. Il y a une forte différence de performance entre ces solveurs et SAT4J_JAVA qui est appelé comme librairie. Ces différences de performances peuvent être dues à l'environnement JAVA utilisé.

Afin de ne pas réduire le biais de notre analyse, nous avons décidé d'effectuer la même étude en ignorant les temps de calcul de SAT4J_JAVA. Ainsi, nous ne conservons que les solveurs appelés par des commandes systèmes, l'ensemble des données est impacté par le coût en performance de l'instanciation du terminal de commande. Les résultats sont disponibles tableau 4. Avec ces nouvelles données, nous constatons que CRYPTOMINISAT est considéré comme plus performant si nous nous fions à la médiane, alors qu'il s'agit de MINISAT si nous nous fions à la moyenne. Il n'y a donc pas de différence de performance significatives entre ces deux solveurs.

Solveur	Version	Différence Médiane	Différence Moyenne
SAT4J_JAVA	2.3.1	0	38.12
SAT4J_JAR	2.0.0	2267.5	8048.91
SAT4J_JAR	2.0.4	2198.75	17006.79
SAT4J_JAR	2.2.3	2336.25	4812.2
SAT4J_JAR	2.3.1	2425	4872.27
SAT4J_COMP	2.0.0	25368.75	37246.19
SAT4J_COMP	2.0.4	26325	41084.40
SAT4J_COMP	2.2.3	17187.5	29038.98
SAT4J_COMP	2.3.1	16968.75	28959.82
minisat rnd-freq=0.01	2.2.0	178.57	1874.29
minisat rnd-freq=0.2	2.2.0	155	644.95
minisat rnd-freq=0.4	2.2.0	155	407.57
minisat rnd-freq=0.6	2.2.0	172.86	616.56
minisat rnd-freq=0.8	2.2.0	179.22	671.14
minisat rnd-freq=0.99	2.2.0	187.86	1671.22
cryptominisat	5.7.8	110.8	3514.67

TABLE 3 – Différences de performances moyennes et médianes pour chaque solveur. Ces résultats sont obtenus en calculant les différences de performance entre les solveurs pour chaque instance.

2.2.2 Comparaison des versions de SAT4J

Il est également intéressant de comparer les performances des différentes versions de SAT4J (par fichier JAR et par MAVEN). Les données sont présentes dans les tableaux 5 et 6. Nous pouvons ainsi constater que les performances ne se sont pas améliorées entre les versions 2.0.0 et 2.3.1. En effet, les résultats obtenus sur SAT4J_JAR montrent de meilleures performances sur les dernières versions, tandis que ceux obtenus sur SAT4J_COMP montrent l'inverse. L'ensemble des résultats restant très proche les uns des autres, nous concluons qu'aucun bogue de performance ne s'est glissé dans les dernières versions et que la performance de SAT4J ne s'est pas améliorée.

2.3 Formules plus dures

Dans cette section, nous allons chercher à définir ce qu'est une formule plus dure que les autres, une formule dont la vérification de satisfiabilité nécessite un grand temps de calcul. Pour déterminer ces critères, nous nous sommes intéressés à trois éléments : le nombre de variables de la formule, le nombre de clauses et le taux de clauses ne possédant qu'un seul littéral.

Les figures 1 et 2 illustrent l'évolution du temps de calcul en fonction du nombre de variables dans les formules. Sur le première, nous constatons qu'il y a du bruit dans les données, il est difficile d'évaluer nettement l'impact du nombre de variables sur les performances des solveurs. Le temps de calcul nécessaire pour déterminer la satisfiabilité des formules semblent croître avec le nombre de variables.
Ce résultat est vérifié par la seconde figure (2). Sur cette figure, nous ne considérons que le temps minimal requis pour l'un de nos solveurs pour traiter les formules. Nous remarquons clairement l'augmentation du temps de calcul en fonction du nombre de variables. Plus une formule aura de variables, plus le solveur prendra du temps pour résoudre le problème de satisfiabilité.

Solveur	Version	Différence Médiane	Différence Moyenne
SAT4J_JAR	2.0.0	1253.57	1585.27
SAT4J_JAR	2.0.4	1306.25	1993.48
SAT4J_JAR	2.2.3	1311.11	1586.92
SAT4J_JAR	2.3.1	1395.14	1612.85
SAT4J_COMP	2.0.0	16958.33	23768.64
SAT4J_COMP	2.0.4	17930	23997.54
SAT4J_COMP	2.2.3	12118.18	18865.84
SAT4J_COMP	2.3.1	12122.73	18823.57
minisat rnd-freq=0.01	2.2.0	28.57	98.27
minisat rnd-freq=0.2	2.2.0	28.57	67.78
minisat rnd-freq=0.4	2.2.0	29.29	57.98
minisat rnd-freq=0.6	2.2.0	29.29	77.61
minisat rnd-freq=0.8	2.2.0	43.65	82.05
minisat rnd-freq=0.99	2.2.0	47.22	193.37
cryptominisat	5.7.8	0	220.62

TABLE 4 – Différences de performances moyennes et médianes pour chaque solveur en ignorant SAT4J_JAVA. Ces résultats sont obtenus en calculant les différences de performance entre les solveurs pour chaque instance.

Solveur	Version	Différence Médiane	Différence Moyenne
SAT4J_COMP	2.0.0	42.12	66.33
SAT4J_COMP	2.0.4	44	74.23
SAT4J_COMP	2.2.3	0	3.98
SAT4J_COMP	2.3.1	0.19	4.28

TABLE 5 – Différences de performances moyennes et médianes pour SAT4J_COMP. Ces résultats sont obtenus en calculant les différences de performance entre les solveurs pour chaque instance.

Nous réalisons une comparaison similaire en étudiant le temps de calcul en fonction du nombre de clauses des formules. Cette étude est illustrée sur les figures 3 et 4. Tout comme précédemment, la première figure ne nous permet pas de conclure réellement sur l'impact du nombre de clauses sur les temps de calcul. Néanmoins, la seconde figure nous permet de constater que plus le nombre de clauses est élevé, plus le temps de calcul est important.

Finalement nous nous sommes intéressés à l'impact du taux de clauses à un littéral sur les temps de calcul. Ce critère semble pertinent car les clauses à un littéral sont triviales et permettent de fixer les valeurs des variables. Ainsi, l'instance *tableBasedAddition.sk_240_1024.cnf* possédant 961 clauses et variables et ayant un taux de clauses à un littéral de 100% est triviale à résoudre. Les résultats de cette étude sont disponibles figures 5 et 6. Nous constatons, grâce à ces graphiques, que plus ce taux est proche de 0%, plus les temps de calcul sont importants. Il y a moins de possibilités de fixer des variables. Ce taux de clauses à un littéral est donc important pour définir la difficulté d'une formule.

Pour conclure cette section, nous pouvons définir une formule dure comme une formule ayant un grand nombre de clauses et de variables mais avec le moins possible de clauses à un littéral. Ces trois facteurs ayant un impact sur les temps de calcul des solveurs, il faut donc maximiser le nombre de clauses et de variables en minimisant le taux de clauses à un littéral pour maximiser les temps de calcul (minimiser les performances des solveurs).

Solveur	Version	Différence Médiane	Différence Moyenne
SAT4J_JAR	2.0.0	0.68	29.39
SAT4J_JAR	2.0.4	1.93	28.64
SAT4J_JAR	2.2.3	5.2	40.2
SAT4J_JAR	2.3.1	6.95	42.35

TABLE 6 – Différences de performances moyennes et médianes pour SAT4J_JAR. Ces résultats sont obtenus en calculant les différences de performance entre les solveurs pour chaque instance.

2.4 Réduction de benchmark

Le dernier point traité par ce rapport est le problème de réduction de benchmark : sélectionner un sous-ensemble des éléments d'un benchmark permettant de valider des outils aussi efficacement que le benchmark complet. Pour répondre à cette question, nous allons nous baser sur les conclusions de la section précédente définissant la notion de formules difficiles.
Comme nous l'avons vu précédemment, plusieurs paramètres impactent la difficulté (*dureté*) d'une formule booléenne sous forme CNF : le nombre de clauses, le nombre de variables et le taux de clauses à un littéral. Ainsi, un benchmark nécessite d'avoir des formules de différentes complexités. Il est donc possible de sélectionner un sous-ensemble du benchmark contenant une couverture maximale du niveau de difficulté des formules.

Comme nous le voyons figures 6 et 5, il est déjà possible de retirer les formules ayant un taux de clauses à un littéral supérieur à 40%, passé ce seuil la difficulté des formules ne décroît plus. De plus, la difficulté des formules est considérablement plus impactée pour des taux inférieurs à 15%. Il faut donc principalement échantillonner le benchmark sur les éléments ayant des taux inférieurs à ce seuil.

Nous avons également constaté l'impact du nombre de variables sur la difficulté des formules. Il y a notamment une forte augmentation du temps de calcul au-dessus de 100 000 variables (voir figure 2). Il est donc nécessaire de conserver ces formules dans le benchmark final, celles-ci étant plus difficiles que les autres. L'évolution du temps de calcul étant croissante et linéaire, il est possible de sélectionner selon une loi uniforme, un sous-ensemble des instances de moins de 100 000 variables.

Le même constat est réalisé avec le nombre de clauses, voir figure 4. Le temps de calcul semble linéaire en fonction du nombre de clauses. Il est donc possible d'échantillonner uniformément les instances. Tout comme précédemment, il est préférable de conserver les instances ayant le plus de clauses afin de conserver les instances les plus complexes. Nous pouvons ici fixer un seuil à 250 000 clauses.

En effectuant ce traitement pour calculer un sous-ensemble du benchmark utilisé pour notre analyse, nous obtenons le même classement des solveurs que celui trouvé tableau 3. Pour ces tests, nous avons essayé différentes valeurs d'échantillonnage : conserver la moitié des instances, conserver un quart des instances. Dans les deux cas, le classement reste inchangé, seul l'ordre des solveurs MINISAT varie. Ainsi, nous passons d'un benchmark de 70 instances à un benchmark de, réciproquement, 22 instances (en moyenne) et 10 instances (en moyenne) sans altérer nos conclusions.

Conclusion

En conclusion, pour ce *Milestone 6* nous avons analysé les performances de différents solveurs avec différentes versions et différents paramètres d'appels, 16 cas testés. Nous avons

déterminé que le solveur testé le plus performant était SAT4J_JAVA mais que ce résultat pouvait être biaisé par la nature de l'appel du solveur : celui-ci est utilisé comme librairie tandis que les autres sont appelés par commande. Ainsi, en ne considérant que les solveurs appelés par commande, nous avons déterminé que MINISAT était le plus performant.

À travers notre étude, nous avons déterminé qu'il y avait des différences significatives de performance entre SAT4J et les autres solveurs (MINISAT et CRYPTOMINISAT). Néanmoins, nous avons conclu que cette différence était due aux environnements MAVEN et JAVA utilisés pour appeler les solveurs.

Nous avons également défini ce qu'est une formule booléenne sous forme normale conjonctive dure : une formule ayant un grand nombre de clauses et de variables possédant un faible taux de clauses simples (à un littéral). Basés sur ce constat, nous avons expliqué comment il était possible de réduire un benchmark sans réduire ces capacités de discrimination. Nous avons appliqué ces modifications au benchmark employé pour cette étude passant ainsi de 70 instances à environ une vingtaine sans que cela n'altère nos conclusions.

Pour poursuivre ce travail, il serait intéressant de continuer cette étude en ajoutant un critère de difficulté des formules : la fréquence moyenne d'apparition des variables dans les clauses. Une variable avec une faible fréquence d'apparition est plus facilement fixable par le solveur. Ainsi, si une formule a une fréquence moyenne d'apparition de variables faibles, elle devrait être plus simple à résoudre.

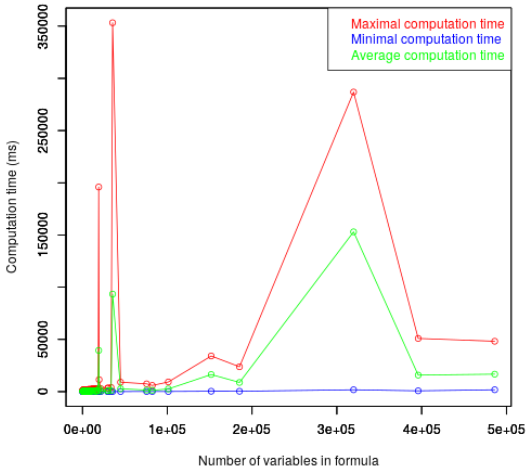

FIGURE 1 – Évolution du temps de calcul moyens des solveurs en fonction du nombre de variables des formules.

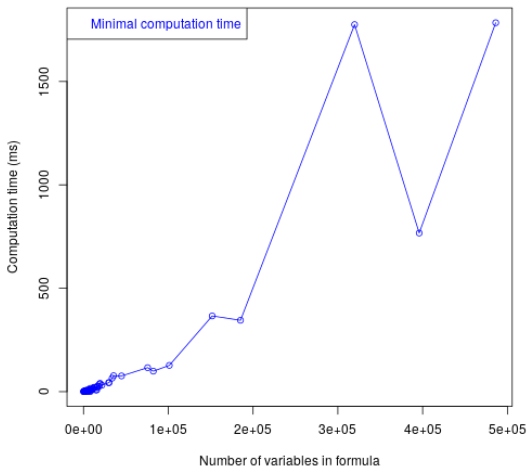

FIGURE 2 – Évolution du temps de calcul moyens du solveur le plus rapide en fonction du nombre de variables des formules.

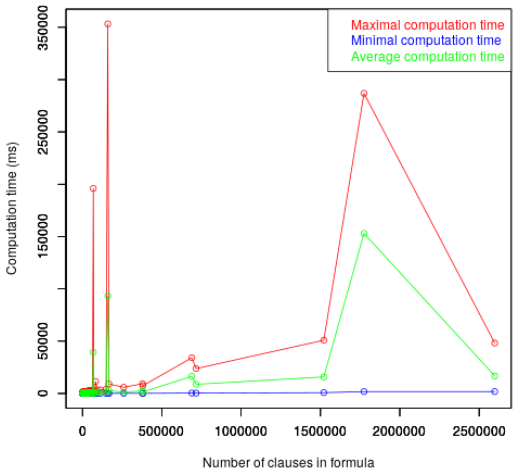

FIGURE 3 – Évolution du temps de calcul moyens des solveurs en fonction du nombre de clauses des formules.

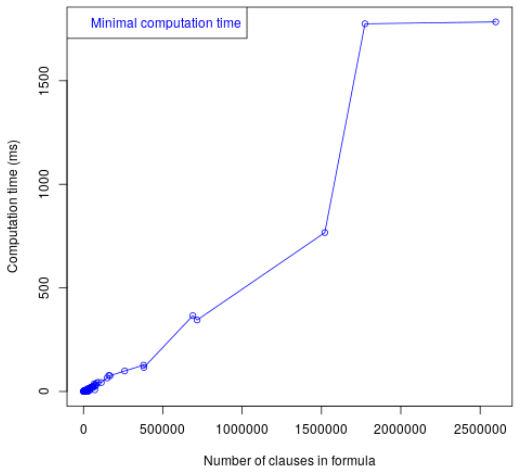

FIGURE 4 – Évolution du temps de calcul moyens du solveur le plus rapide en fonction du nombre de clauses des formules.

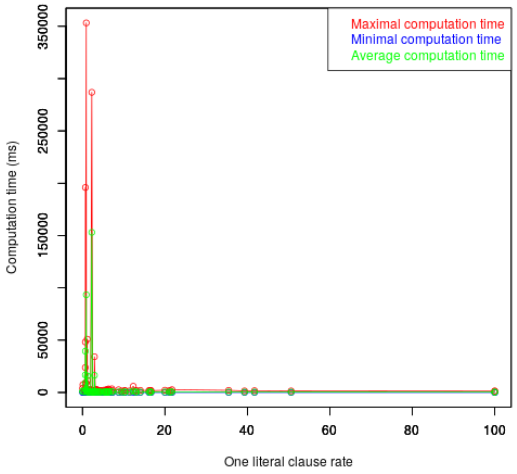

FIGURE 5 – Évolution du temps de calcul moyens des solveurs en fonction du taux de clauses à un littéral des formules.

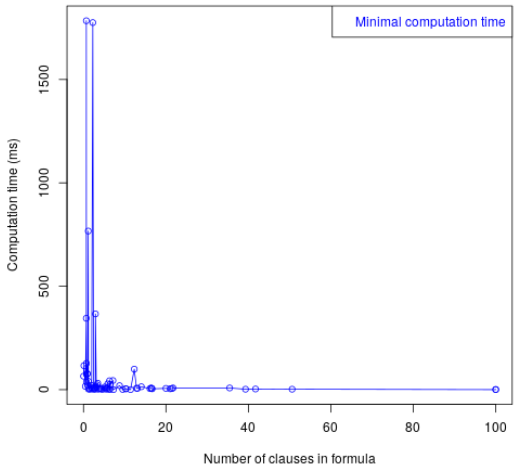

FIGURE 6 – Évolution du temps de calcul moyens du solveur le plus rapide en fonction du taux de clauses à un littéral des formules.