UNIVERSITÉ DU QUÉBEC À MONTRÉAL

# SOLVEURS MAXSAT ANYTIME : CONCEPTION D'UN BANC D'ESSAI ET DÉVELOPPEMENT D'UN SOLVEUR DE RECHERCHE LOCALE

MÉMOIRE

PRÉSENTÉ

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUE

PAR

AMADOU SALL

JUILLET 2026

**REMERCIEMENTS**

Je rends d'abord grâce à DIEU, qui m'a accordé la santé, la force et la persévérance nécessaires pour mener ce travail à terme.

J'adresse mes plus sincères remerciements à ma famille, en particulier à mon père, à ma mère, à ma femme, à mes sœurs, à mes frères et à ma grand-mère, ainsi qu'à toute ma famille élargie, notamment ma belle-mère, mes cousins, mes oncles, mes tantes et tous mes proches, pour leur soutien, leurs prières et leurs encouragements.

Je remercie également mes amis de tous horizons, avec une mention spéciale à mes amis proches, qui ont contribué de près ou de loin à l'aboutissement de ce mémoire.

Enfin, j'exprime ma profonde gratitude à mon directeur de recherche, Florent Avellaneda, pour son encadrement, ses conseils et son accompagnement, ainsi qu'à l'ensemble des professeurs qui m'ont formé durant cette maîtrise.

**TABLE DES MATIÈRES**

## TABLE DES FIGURES

**LISTE DES TABLEAUX**

**ACRONYMES**

**MSE** Évaluation MaxSAT.

**SAT** problème de satisfiabilité booléenne.

**MaxSAT** problème de satisfiabilité maximale.

**WPMaxSAT** problème de satisfiabilité maximale partielle pondérée.

**CNF** forme normale conjonctive.

**WCNF** forme normale conjonctive pondérée.

**CDCL** apprentissage de clauses guidé par les conflits.

**P** temps polynomial.

**NP** temps polynomial non déterministe.

**NPO** optimisation en NP.

**SLS** recherche locale stochastique.

**AUC** aire sous la courbe.

**DTW** déformation temporelle dynamique.

**SDT** score de domination temporelle.

**SDT-log** score de domination temporelle logarithmique.

**UML** langage de modélisation unifié.

**CLI** interface en ligne de commande.

**API** interface de programmation d'application.

**CSV** valeurs séparées par des virgules.

**JSON** notation objet de JavaScript.

## NOTATION

### MaxSAT et évaluation

$I$ ensemble (ou famille) d'instances considérées dans une campagne expérimentale.

$i$ instance particulière, avec $i \in I$.

$\mathrm{SOL}(i)$ ensemble des solutions candidates associées à l'instance $i$.

$s$ solveur considéré dans une comparaison expérimentale.

$\sigma$ affectation complète courante dans le solveur expérimental.

$\sigma^x$ affectation obtenue après retournement de la variable $x$.

$f(i, s)$ fonction objectif d'un problème d'optimisation, définie pour $s \in \mathrm{SOL}(i)$.

$T$ budget temporel total ou *timeout*.

$t$ instant de mesure, avec $t \in [0, T]$.

$\varphi$ formule booléenne, généralement en CNF ou WCNF.

$w_j$ poids objectif de la clause souple $C_j$ en MaxSAT pondéré.

$\mathrm{cost}(\sigma)$ coût de l'affectation $\sigma$, égal à la somme des poids des clauses souples violées.

$\mathrm{Cost}_{s,i}(t)$ meilleur coût connu du solveur $s$ sur l'instance $i$ à l'instant $t$.

$\mathrm{Best}_i(t)$ meilleur coût observé à l'instant $t$ sur l'instance $i$ parmi les solveurs comparés.

$\mathrm{Score}_{s,i}(t)$ score relatif du solveur $s$ sur l'instance $i$ à l'instant $t$.

$I_s(t)$ ensemble des instances prises en compte dans l'agrégation à l'instant $t$.

$\mathrm{Score}_s(t)$ score moyen agrégé du solveur $s$ à l'instant $t$ sur l'ensemble des instances couvertes.

$\mathrm{AUC}$ aire sous une courbe de performance.

$\mathrm{SDT}_s$ score de domination temporelle du solveur $s$ sur un intervalle temporel donné.

$\mathrm{SDT}_s^{\log}$ version logarithmique du score de domination temporelle, donnant davantage de poids aux débuts de trajectoire.

$x$ variable booléenne d'une instance SAT ou MaxSAT.

$s(x)$ score local associé à la variable $x$, utilisé pour guider le choix du retournement de variable le plus favorable.

$\Delta(x, \sigma)$ variation de coût induite par le retournement de la variable $x$ dans l'affectation $\sigma$.

**RÉSUMÉ**

Ce mémoire étudie les solveurs MaxSAT *anytime*, c'est-à-dire des algorithmes capables de produire rapidement une solution valide puis d'en améliorer progressivement la qualité sous une contrainte de temps. Le travail poursuit deux objectifs complémentaires : proposer un cadre d'évaluation mieux adapté à la dynamique temporelle de ces solveurs et disposer d'une base expérimentale modulaire pour analyser finement plusieurs mécanismes de recherche locale.

La première contribution est *MaxSAT Runner*, un banc d'essai qui capture les améliorations de coût au fil de l'exécution, reconstruit des trajectoires *anytime*, agrège plusieurs répétitions et produit des visualisations ainsi que des indicateurs comparatifs adaptés à l'analyse temporelle.

La seconde contribution est *EvalMaxSAT Anytime*, une base expérimentale de solveur pour *Weighted Partial MaxSAT*, conçue pour étudier de manière contrôlée l'effet du redémarrage, de la pondération dynamique et du débiaisage dans la sortie des minima locaux.

Les expériences montrent que *MaxSAT Runner* permet de distinguer des comportements qui restent invisibles dans une lecture fondée uniquement sur le coût final, tandis qu'*EvalMaxSAT Anytime* met clairement en évidence le rôle structurant de la pondération dynamique et l'apport plus nuancé du débiaisage selon l'horizon temporel et le type d'instances considéré.

**Mots-clés :** MaxSAT anytime, évaluation expérimentale, recherche locale stochastique, trajectoires de convergence, pondération dynamique.

# INTRODUCTION

L'optimisation combinatoire est au cœur de nombreux problèmes industriels et scientifiques, allant de la planification logistique à la vérification formelle de systèmes. Parmi les formalismes permettant de modéliser ces problèmes, le problème de la satisfiabilité booléenne maximale (*Maximum Satisfiability*, MaxSAT) s'est imposé comme un cadre de référence, en raison de son expressivité et des progrès continus des solveurs modernes (Biere *et al.*, 2009).

Dans des contextes opérationnels, obtenir une solution optimale demeure toutefois souvent hors de portée dans les délais disponibles, en raison de la complexité intrinsèquement NP-difficile de ces problèmes (Garey et Johnson, 1979). C'est dans cette perspective que s'inscrivent les algorithmes dits *anytime* (Zilberstein, 1996). Contrairement aux approches exactes qui ne livrent une solution qu'à la fin de l'exécution, un algorithme *anytime* peut produire à tout instant une solution valide dont la qualité s'améliore progressivement avec le temps de calcul alloué. Cette propriété est déterminante pour les systèmes interactifs ou soumis à des contraintes temporelles, où une réponse rapide, même sous-optimale, est préférable à l'absence de réponse.

Cependant, l'évaluation des solveurs *anytime* reste un défi méthodologique. Les métriques classiques, telles que le temps jusqu'à l'optimum ou le nombre d'instances résolues, ne capturent pas la dynamique temporelle de la convergence. Comme le soulignent Dolan et Moré (Dolan et Moré, 2002), la comparaison d'algorithmes d'optimisation requiert des outils d'analyse plus fins, capables de caractériser la performance sur l'ensemble de l'horizon d'exécution, et pas seulement à un temps final fixé.

Malgré l'importance croissante des approches *anytime*, notamment avec l'essor de solveurs fondés sur la recherche locale stochastique tels que NuWLS (Chu *et al.*, 2023), il existe encore peu d'outils standardisés permettant d'évaluer et de comparer de manière robuste les trajectoires d'amélioration des solveurs. Les cadres d'évaluation actuels, notamment ceux associés à la MaxSAT Evaluation (Berg *et al.*, 2024b), privilégient généralement des critères centrés sur la qualité finale à un temps limite (*timeout*).

Cette pratique tend à masquer des différences importantes entre solveurs durant les premières phases d'exécution, alors que ces phases sont cruciales dans de nombreux scénarios d'usage.

Ce mémoire vise à combler cette lacune en proposant une méthodologie et un outillage dédiés à l'analyse approfondie des solveurs MaxSAT *anytime*. En parallèle, il s'appuie sur ces outils pour concevoir une base expérimentale de solveur permettant d'étudier, de manière contrôlée, plusieurs mécanismes internes de recherche locale et leur effet sur la dynamique de résolution.

La démarche de ce mémoire est structurée autour de deux questions de recherche explicites :

1. **QR1.** Dans quelle mesure un banc d'essai fondé sur les trajectoires temporelles permet-il de comparer plus finement des solveurs MaxSAT *anytime* qu'une lecture limitée au coût final au temps limite ?
2. **QR2.** Quels mécanismes de sortie de minima locaux — redémarrage, pondération dynamique et débiaisage — améliorent le plus favorablement la dynamique *anytime* d'un solveur expérimental de recherche locale ?

Les contributions principales de ce travail sont les suivantes :

1. **MaxSAT Runner (banc d'essai *anytime*)** : nous proposons un outil open-source de banc d'essai conçu spécifiquement pour l'évaluation des solveurs MaxSAT *anytime*. L'outil permet l'exécution en *streaming*, l'enregistrement des trajectoires de coût au cours du temps, l'agrégation des résultats par instance et par solveur, ainsi que l'exploitation de métriques adaptées à l'analyse *anytime*, telles que le score relatif normalisé, les temps d'atteinte du meilleur coût et les métriques de domination temporelle. Il intègre également des techniques d'analyse exploratoire, dont un regroupement des trajectoires fondé sur des distances entre séries temporelles telles que *Dynamic Time Warping* (DTW) (Salvador et Chan, 2007).

   L'outil est mis à l'épreuve sur les *tracks anytime weighted* et *anytime unweighted* de la MaxSAT Evaluation 2024, ce qui permet d'illustrer l'apport d'une lecture temporelle plus fine des comportements de solveurs. Le terme *track*, utilisé dans le cadre de cette compétition, désigne une catégorie regroupant un type d'instances, un protocole d'évaluation et un ensemble de règles communes.
2. **EvalMaxSAT Anytime (base expérimentale de solveur)** : nous proposons une base expérimentale de solveur MaxSAT *anytime*, fondée sur des mécanismes de recherche locale pour *Weighted Partial MaxSAT*. La contribution ne réside pas principalement dans la proposition immédiate d'un solveur compétitif face aux meilleurs systèmes de l'état de l'art, mais dans la mise en place d'un cadre modulaire permettant d'étudier, de manière contrôlée, plusieurs mécanismes internes de recherche.

L'étude met plus particulièrement l'accent sur les mécanismes de sortie de minima locaux, notamment le redémarrage, la pondération dynamique et le débiaisage, évalués sur un banc d'essai local contrôlé à l'aide des protocoles et métriques fournis par MaxSAT Runner.

Ce mémoire est structuré comme suit :

- Le Chapitre 1 présente le cadre théorique nécessaire, incluant des rappels de théorie de la complexité, ainsi que les bases des problèmes SAT/MaxSAT et des approches de résolution.
- Le Chapitre 2 dresse un état de l'art des techniques de résolution MaxSAT, depuis les approches exactes jusqu'aux méthodes stochastiques modernes, et discute les méthodes d'évaluation et de comparaison existantes.
- Le Chapitre 3 décrit MaxSAT Runner : son architecture, les formats de données utilisés, la chaîne de collecte des événements, les métriques *anytime* retenues, ainsi que les méthodes d'analyse et de visualisation mises en place.
- Le Chapitre 4 présente les résultats expérimentaux obtenus avec MaxSAT Runner sur les tracks *anytime weighted* et *unweighted* de la MaxSAT Evaluation 2024, ainsi qu'une mise en perspective méthodologique des analyses produites.
- Le Chapitre 5 présente EvalMaxSAT Anytime, le solveur expérimental développé dans ce mémoire, ses structures de données, les mécanismes étudiés, les variantes retenues et leur évaluation expérimentale sur un banc d'essai local contrôlé.
- Enfin, la conclusion générale synthétise les contributions du mémoire, discute les limites du travail réalisé et propose plusieurs perspectives de recherche.

Afin de faciliter la vérification et la réutilisation des contributions logicielles présentées dans ce mémoire, les artefacts utilisés sont documentés dans la bibliographie et considérés comme des ressources logicielles à part entière. Le banc d'essai *MaxSAT Runner* fait l'objet d'une notice bibliographique dédiée (Sall, 2026b). L'implémentation *EvalMaxSAT Anytime* est également documentée comme un artefact logiciel distinct (Sall, 2026a).

# CHAPITRE I
# CADRE THÉORIQUE

Ce chapitre présente les notions théoriques nécessaires à la compréhension du problème étudié dans ce mémoire. L'objectif n'est pas encore de décrire les contributions proposées, mais de poser un cadre conceptuel commun permettant d'aborder, dans les chapitres suivants, les méthodes de résolution et les comparaisons expérimentales. Nous introduisons d'abord les bases de la complexité algorithmique et des problèmes d'optimisation, puis les fondements de SAT et de MaxSAT, avant de terminer par les notions générales liées aux algorithmes *anytime* et à leur évaluation.

## 1.1 Complexité algorithmique et problèmes de décision

L'étude de la complexité algorithmique vise à caractériser les ressources nécessaires à la résolution d'un problème, en particulier le temps de calcul et l'espace mémoire, en fonction de la taille de l'entrée (Sipser, 2012; Arora et Barak, 2009). Dans le cas des problèmes booléens, la taille d'une instance peut être décrite, selon le niveau de détail souhaité, par le nombre de variables, le nombre de clauses, ou encore la taille totale de l'encodage.

### 1.1.1 Mesures de coût et notations asymptotiques

Pour comparer des algorithmes de manière abstraite, on s'intéresse généralement à leur comportement asymptotique lorsque la taille de l'entrée augmente. Les notations $O(\cdot)$, $\Omega(\cdot)$ et $\Theta(\cdot)$ permettent de décrire respectivement une borne supérieure, une borne inférieure et un ordre de grandeur asymptotique (Sipser, 2012).

Ces notations ne rendent pas compte de tous les aspects pratiques d'une implémentation, mais elles jouent un rôle essentiel pour distinguer les familles de problèmes qui admettent, ou non, des algorithmes efficaces à grande échelle. Elles constituent ainsi le langage de base de la théorie de la complexité.

### 1.1.2 Problèmes de décision et classes $\mathbf{P}$ et $\mathbf{NP}$

Un *problème de décision* est un problème dont la réponse attendue est binaire, typiquement « oui » ou « non ». La classe $\mathbf{P}$ regroupe les problèmes qui peuvent être décidés en temps polynomial par une machine déterministe. La classe $\mathbf{NP}$ regroupe les problèmes pour lesquels, lorsqu'une instance est positive, il existe un certificat de taille polynomiale pouvant être vérifié en temps polynomial (Sipser, 2012; Arora et Barak, 2009).

Il est important de souligner que $\mathbf{NP}$ ne désigne pas la classe des problèmes « nécessairement difficiles », mais la classe des problèmes pour lesquels une solution candidate peut être vérifiée efficacement. Cette distinction est centrale pour comprendre pourquoi certains problèmes de recherche ou d'optimisation sont étroitement liés à des problèmes de décision.

### 1.1.3 Réductions polynomiales et NP-complétude

La notion de réduction polynomiale formalise l'idée qu'un problème $\Pi_1$ n'est pas plus difficile qu'un problème $\Pi_2$ si toute instance de $\Pi_1$ peut être transformée en une instance de $\Pi_2$ en temps polynomial, de manière à préserver la réponse (Karp, 1972; Garey et Johnson, 1979). Les réductions sont donc l'outil fondamental qui permet de comparer la difficulté relative de problèmes différents.

Un problème est dit *NP-complet* s'il appartient à $\mathbf{NP}$ et si tout problème de $\mathbf{NP}$ peut s'y réduire en temps polynomial. Le théorème de Cook-Levin établit que le problème SAT est NP-complet (Cook, 1971). Les travaux de Karp ont ensuite établi la NP-complétude de nombreux problèmes combinatoires classiques (Karp, 1972; Garey et Johnson, 1979). Cette classification explique pourquoi, en pratique, on développe des solveurs spécialisés, des heuristiques ou des méthodes d'approximation pour de tels problèmes (Papadimitriou, 1994).

## 1.2 Problèmes d'optimisation et classe NPO

Dans de nombreuses applications, la question n'est pas seulement de savoir si une solution existe, mais de déterminer la *meilleure* solution selon un critère donné. On passe alors d'un problème de décision à un problème d'optimisation.

### 1.2.1 Définition générale

Un problème d'optimisation est généralement défini par :

- un ensemble d'instances $I$ ;
- pour chaque instance $i \in I$, un ensemble de solutions admissibles $s \in SOL(i)$ ;
- une fonction objectif $f(i, s)$, calculable efficacement ;
- un critère d'optimisation, consistant à minimiser ou à maximiser $f$.

La classe **NPO** (*NP Optimization*) regroupe les problèmes d'optimisation pour lesquels :

- la taille d'une solution candidate est bornée polynomialement par la taille de l'instance ;
- la faisabilité d'une solution peut être vérifiée en temps polynomial ;
- la valeur de la fonction objectif peut être calculée en temps polynomial.

Cette classe fournit un cadre théorique naturel pour raisonner sur des problèmes comme MaxSAT (Krentel, 1988; Papadimitriou et Yannakakis, 1991).

### 1.2.2 Lien entre optimisation et décision

Un problème d'optimisation est souvent associé à une version décisionnelle. Par exemple, au lieu de demander quelle est la meilleure valeur possible d'une fonction objectif, on peut demander s'il existe une solution dont la valeur est au moins égale à un certain seuil, ou au plus égale à une borne donnée. Cette relation entre optimisation et décision permet de relier les problèmes de la classe NPO aux classes classiques de la théorie de la complexité (Papadimitriou et Yannakakis, 1991).

Dans le cas de MaxSAT, cette idée prend une forme très naturelle : la version décisionnelle consiste à demander s'il existe une affectation satisfaisant au moins un certain nombre de clauses, ou, dans une formulation de minimisation, si l'on peut atteindre un coût inférieur ou égal à une borne donnée.

### 1.2.3 Approximation et compromis qualité–temps

Lorsque l'optimum exact est difficile à obtenir, on s'intéresse à des solutions approchées ou à des algorithmes capables de produire rapidement des solutions de bonne qualité. Les notions d'approximation, de rapport de performance et de réductions préservant l'approximation sont utiles pour formaliser ces compromis (Papadimitriou et Yannakakis, 1991; Vazirani, 2001; Ausiello *et al.*, 1999).

Même lorsque l'on ne cherche pas à établir une garantie théorique d'approximation, cette perspective reste importante : elle rappelle que le temps de calcul et la qualité de solution sont souvent en tension, et que l'évaluation d'un solveur d'optimisation doit tenir compte de ce compromis.

## 1.3 SAT : formalisme booléen et résolution

Cette section introduit le formalisme du problème SAT ainsi que les principaux concepts nécessaires à sa résolution.

### 1.3.1 Variables, littéraux, clauses et forme CNF

Le problème SAT (*Boolean Satisfiability Problem*) consiste à déterminer s'il existe une affectation des variables booléennes qui rende vraie une formule donnée. Dans la pratique, les formules sont très souvent représentées en *forme normale conjonctive* (CNF, *Conjunctive Normal Form*) (Sipser, 2012; Biere *et al.*, 2009).

Une formule CNF est une conjonction de clauses, et chaque clause est une disjonction de littéraux. Un littéral est soit une variable booléenne $x$, soit sa négation $\neg x$. Une affectation satisfait une clause si au moins un de ses littéraux est vrai, et satisfait la formule entière si toutes les clauses sont satisfaites.

Cette représentation est particulièrement importante, car elle constitue le format d'entrée standard de nombreux solveurs SAT et MaxSAT. Elle permet également de raisonner de manière uniforme sur les contraintes booléennes.

### 1.3.2 Des méthodes classiques à CDCL

Historiquement, les premières méthodes générales pour SAT remontent à la procédure de Davis et Putnam, puis à l'algorithme DPLL (Davis–Putnam–Logemann–Loveland), qui combine branchement, simplification et retour arrière (Davis et Putnam, 1960; Davis *et al.*, 1962).

Les solveurs SAT modernes reposent majoritairement sur le paradigme CDCL (*Conflict-Driven Clause Learning*), qui étend DPLL par l'apprentissage de clauses à partir des conflits rencontrés pendant la recherche (Marques-Silva et Sakallah, 1999; Moskewicz *et al.*, 2001). Ce mécanisme d'apprentissage améliore considérablement l'efficacité pratique en évitant de répéter certaines erreurs de recherche. L'évolution des heu-

ristiques de branchement, des politiques de redémarrage et des structures de données a ensuite permis l'émergence de solveurs extrêmement performants sur de nombreuses familles d'instances (Biere *et al.*, 2009).

Il n'est pas nécessaire, dans ce chapitre, d'entrer dans tous les détails de ces mécanismes. Il suffit de retenir que SAT constitue aujourd'hui une brique algorithmique fondamentale, souvent utilisée comme sous-routine dans des problèmes d'optimisation plus complexes, notamment MaxSAT.

## 1.4 MaxSAT et ses variantes

Cette section introduit MaxSAT et ses principales variantes. Elle présente successivement les formulations non pondérée et pondérée, la distinction entre clauses dures et clauses souples, ainsi que le format WCNF utilisé pour représenter les instances. Elle se termine par un aperçu des grandes familles de méthodes de résolution.

### 1.4.1 De SAT à MaxSAT

MaxSAT peut être vu comme une généralisation d'optimisation de SAT (Biere *et al.*, 2009; Li et Manyà, 2009). Alors que SAT demande si toutes les clauses d'une formule peuvent être satisfaites simultanément, MaxSAT cherche une affectation qui satisfasse le plus grand nombre possible de clauses, ou, de manière équivalente, qui minimise le nombre de clauses violées.

Cette reformulation est naturelle dans de nombreuses applications réelles, où certaines contraintes peuvent être considérées comme souhaitables sans être absolument obligatoires. MaxSAT permet ainsi de modéliser des compromis entre faisabilité stricte et qualité de solution.

### 1.4.2 MaxSAT non pondéré

Dans sa forme non pondérée, MaxSAT associe la même importance à toutes les clauses. L'objectif consiste alors à maximiser le nombre de clauses satisfaites, ou, de manière équivalente, à minimiser le nombre de clauses falsifiées (Li et Manyà, 2009).

Si $\varphi = \{C_1, \ldots, C_m\}$ est un ensemble de clauses et $s$ une affectation, le coût non pondéré de $s$ est défini par

$$\text{cost}(s) = |\{C_j \in \varphi \mid s \not\models C_j\}| \,.$$

Il correspond au nombre de clauses de $\varphi$ qui sont fausses sous l'affectation $s$. Une affectation optimale est une affectation qui minimise ce coût.

### 1.4.3 MaxSAT pondéré

Dans la variante pondérée, chaque clause $C_j$ est associée à un poids strictement positif $w_j$. Toutes les violations n'ont donc plus la même importance : violer une clause fortement pondérée est plus coûteux que violer une clause faiblement pondérée (Li et Manyà, 2009; Ansótegui *et al.*, 2009).

La fonction objectif s'écrit alors :

$$\text{cost}(s) = \sum_{j : C_j \text{ est violée par } s} w_j.$$

Le cas non pondéré apparaît comme un cas particulier du cas pondéré, dans lequel tous les poids valent 1.

### 1.4.4 Clauses dures, clauses souples et Partial MaxSAT

Dans de nombreuses modélisations, toutes les contraintes n'ont pas le même statut. Certaines doivent être satisfaites absolument : on parle alors de *clauses dures*. D'autres peuvent être violées au prix d'une pénalité : ce sont les *clauses souples* (Li et Manyà, 2009).

Le *Partial MaxSAT* distingue explicitement ces deux catégories. Une solution admissible doit d'abord satisfaire toutes les clauses dures, puis optimiser la satisfaction des clauses souples. Lorsque les clauses souples sont en plus pondérées, on obtient la variante la plus générale, souvent appelée *Weighted Partial MaxSAT* (WPMaxSAT) (Ansótegui *et al.*, 2009).

Cette taxonomie est importante :

- **MaxSAT** : toutes les clauses sont souples et non pondérées ;
- **Weighted MaxSAT** : toutes les clauses sont souples mais pondérées ;
- **Partial MaxSAT** : présence de clauses dures et de clauses souples non pondérées ;
- **Weighted Partial MaxSAT** : présence de clauses dures et de clauses souples pondérées.

WPMaxSAT englobe donc les variantes précédentes et constitue le cadre le plus général.

### 1.4.5 Représentation en WCNF

En pratique, les instances MaxSAT sont souvent représentées dans un format dérivé de la CNF, appelé *WCNF* (*Weighted CNF*) (Ansótegui *et al.*, 2009). L'idée est d'associer un poids à chaque clause, tout en réservant une convention particulière pour distinguer les clauses dures des clauses souples. Historiquement, cela se fait souvent à l'aide d'un poids spécial, strictement supérieur à la somme des poids des clauses souples, afin de garantir qu'une violation d'une clause dure soit toujours inacceptable au regard de l'objectif.

Cette représentation est particulièrement utile car elle permet de traiter, dans un même cadre syntaxique, les variantes pondérées et partielles de MaxSAT. Elle joue également un rôle central dans les formats d'entrée des solveurs et dans les campagnes de benchmark.

### 1.4.6 Grandes familles d'approches pour MaxSAT

Sans entrer encore dans l'état de l'art détaillé, il est utile d'indiquer qu'il existe plusieurs grandes familles d'approches pour résoudre MaxSAT (Li et Manyà, 2009; Biere *et al.*, 2009). Certaines méthodes sont exactes et visent l'optimum certifié ; d'autres sont incomplètes et cherchent rapidement de bonnes solutions sans garantie d'optimalité.

Parmi les approches exactes, une famille importante repose sur des appels successifs à un solveur SAT dans des schémas dits *core-guided*. Lorsqu'une formule est déclarée insatisfaisable, le solveur SAT peut extraire un noyau insatisfaisable, c'est-à-dire un sous-ensemble de clauses qui ne peuvent pas être satisfaites simultanément. Les clauses souples impliquées dans ce noyau sont alors progressivement relâchées, généralement à l'aide de variables supplémentaires et de contraintes de cardinalité, de façon à identifier le coût minimal nécessaire pour rendre l'instance satisfaisable (Fu et Malik, 2006).

Dans cette famille, *Open-WBO* constitue une plateforme logicielle modulaire qui regroupe plusieurs algorithmes exacts pour MaxSAT dans une architecture commune. Son intérêt ne réside donc pas seulement dans ses performances, mais aussi dans le fait qu'il facilite l'implantation, la comparaison et l'expérimentation de différentes stratégies de résolution (Martins *et al.*, 2014).

*EvalMaxSat* illustre une génération plus récente de solveurs exacts fondés sur SAT. Il utilise de manière répétée un solveur SAT pour vérifier la faisabilité de différentes bornes sur le coût et pour resserrer progressivement l'intervalle dans lequel se situe l'optimum. Ce type d'approche permet non seulement de trouver une solution de bonne qualité, mais également d'établir qu'aucune solution de coût inférieur n'existe (Avellaneda, 2021).

À côté de ces méthodes exactes, il existe aussi des approches incomplètes, souvent fondées sur la recherche locale stochastique. Ces approches sont particulièrement pertinentes lorsque l'on cherche de bonnes solutions en temps limité, ce qui conduit naturellement à la notion d'algorithme *anytime*.

## 1.5 Algorithmes *anytime*

Cette section introduit les algorithmes *anytime* et les éléments nécessaires à leur évaluation.

### 1.5.1 Définition générale

Un algorithme *anytime* est un algorithme qui peut être interrompu à tout moment et qui est capable de retourner une solution valide, généralement de meilleure qualité lorsque le temps disponible augmente (Zilberstein, 1996). Ce type d'algorithme est particulièrement intéressant lorsque le temps de calcul est contraint, ou lorsque l'on souhaite obtenir rapidement une solution exploitable puis l'améliorer progressivement.

Dans le cadre de l'optimisation, cette propriété est particulièrement précieuse : même si l'optimum n'est pas atteint, l'utilisateur dispose d'une solution intermédiaire dont la qualité peut être suivie dans le temps.

### 1.5.2 Trajectoires de qualité

Pour un algorithme d'optimisation *anytime*, la performance ne se résume pas à une valeur finale au timeout. Elle peut être décrite par une trajectoire de qualité, c'est-à-dire une fonction $c(t)$ qui associe à chaque instant $t$ la meilleure valeur objective trouvée jusque-là (Zilberstein, 1996; Hoos et Stützle, 2004).Le temps $t$ correspond au temps réel écoulé depuis le début de l'exécution, mesuré en secondes. Cette mesure dépend de la machine et de l'environnement d'exécution, ce qui impose de comparer les solveurs dans des conditions expérimentales homogènes.

Dans le cas de MaxSAT, on s'intéresse souvent à un coût à minimiser. La trajectoire $c(t)$ est alors non croissante : plus le solveur progresse, plus le coût peut diminuer, ou au moins rester inchangé. En pratique, ces trajectoires sont souvent des fonctions en escalier, car les améliorations n'apparaissent qu'à des instants discrets correspondant à la découverte d'une meilleure solution.

Cette représentation est importante car elle permet de distinguer plusieurs comportements :

- des solveurs très réactifs, qui améliorent vite leurs solutions au début ;
- des solveurs plus lents au démarrage, mais capables de mieux converger à long horizon ;
- des solveurs réguliers, dont les améliorations sont progressives et stables.

### 1.5.3 Déterminisme, stochasticité et répétitions

Tous les solveurs ne se comportent pas de manière déterministe. Les méthodes de recherche locale stochastique, en particulier, peuvent produire des trajectoires différentes d'une exécution à l'autre selon la graine aléatoire utilisée (Hoos et Stützle, 2004). Dans ce cas, une seule exécution ne suffit pas à caractériser un solveur.

Il est alors nécessaire, dans une étude expérimentale, de considérer plusieurs répétitions indépendantes et d'agréger les résultats de manière appropriée. Cette précaution est essentielle pour distinguer les performances dues à une tendance réelle de celles qui pourraient être liées à une fluctuation aléatoire.

## 1.6 Résumé du chapitre

Ce chapitre introduit les notions théoriques qui servent de base à la suite du mémoire en rappelant les concepts essentiels de la complexité algorithmique, de la NP-complétude et des problèmes d'optimisation, puis en présentant le formalisme de SAT et son prolongement naturel vers MaxSAT.

Les principales variantes de MaxSAT — non pondéré, pondéré, partiel et pondéré partiel — sont distinguées, de même que leur représentation en WCNF.

Enfin, nous introduisons la notion d'algorithme *anytime*. Ces éléments fournissent, au chapitre suivant, le point d'appui nécessaire pour examiner plus précisément les approches de résolution et l'état de l'art associé.

# CHAPITRE II
# ÉTAT DE L'ART

Ce chapitre présente l'état de l'art relatif à la résolution de MaxSAT, avec un accent particulier sur les solveurs *anytime* et sur les cadres expérimentaux utilisés pour les comparer. Le Chapitre 1 introduit les notions générales de complexité, les définitions de SAT et de MaxSAT, ainsi que les principes de base des algorithmes *anytime*. Ici, l'objectif est d'examiner les grandes familles de solveurs, les principales tendances méthodologiques et les compétitions récentes qui structurent le domaine.

## 2.1 Des solveurs SAT aux solveurs MaxSAT

### 2.1.1 SAT comme brique de base

Une grande partie des solveurs MaxSAT modernes repose sur l'utilisation répétée d'un solveur SAT comme sous-routine. Cette dépendance s'explique par la maturité atteinte par les solveurs SAT, en particulier ceux fondés sur le paradigme CDCL (*Conflict-Driven Clause Learning*) (Biere *et al.*, 2009). Après les approches historiques de Davis–Putnam et DPLL (Davis et Putnam, 1960; Davis *et al.*, 1962), les techniques d'apprentissage de clauses, de propagation efficace, de redémarrage et de gestion des clauses apprises ont permis l'émergence de solveurs extrêmement performants (Marques-Silva et Sakallah, 1999; Moskewicz *et al.*, 2001).

Des solveurs comme zChaff (Moskewicz *et al.*, 2001), MiniSat (Eén et Sörensson, 2004), Glucose (Audemard et Simon, 2009) ou CryptoMiniSat (Soos *et al.*, 2009) ont joué un rôle important dans cette évolution. Au-delà de leur usage direct pour SAT, ils ont surtout fourni les mécanismes de raisonnement et les bases logicielles qui sont ensuite réutilisés dans de nombreux solveurs MaxSAT.

### 2.1.2 Pourquoi MaxSAT réutilise SAT

Dans les solveurs MaxSAT exacts, le solveur SAT sert souvent à tester la faisabilité d'une borne sur le coût, à extraire des cœurs insatisfaisables ou à propager des contraintes additionnelles (Biere *et al.*, 2009). Dans les solveurs anytime, les méthodes SAT-based sont également utilisées pour fournir des solutions faisables ou pour affiner des bornes pendant l'optimisation. Cette relation étroite entre SAT et MaxSAT explique pourquoi l'évolution des solveurs MaxSAT est historiquement liée à celle des solveurs SAT.

## 2.2 Grandes familles de solveurs MaxSAT

### 2.2.1 Solveurs exacts

Les solveurs exacts cherchent à produire une solution optimale accompagnée d'une preuve qu'aucune solution de meilleure qualité n'existe. Ils s'appuient généralement sur des approches *SAT-based*, qui transforment le problème MaxSAT en une suite de problèmes SAT de décision. Une première famille importante regroupe les approches par *bornes*, où l'on cherche à déterminer si une solution de coût au plus $K$ existe, puis où l'on ajuste progressivement cette borne (Biere *et al.*, 2009).

Une deuxième famille majeure est celle des approches *core-guided*. Lorsqu'une formule est insatisfaisable, le solveur SAT peut extraire un *noyau insatisfaisable* (*unsat core*), c'est-à-dire un sous-ensemble de clauses déjà contradictoire à lui seul. Autrement dit, même si l'on ignore le reste de la formule, les clauses de ce sous-ensemble ne peuvent pas être satisfaites simultanément. Dans les approches *core-guided*, ces noyaux servent à identifier les clauses souples impliquées dans les conflits, puis à reformuler progressivement l'instance en relaxant certaines de ces clauses et en ajoutant des contraintes supplémentaires (Fu et Malik, 2006). Ce paradigme a profondément structuré la résolution exacte moderne de MaxSAT.

Un petit exemple permet d'en illustrer le principe. Considérons une instance de MaxSAT partiel composée des clauses dures

$$\mathcal{H} = \{(\neg a \vee \neg b), (\neg a \vee \neg c), (\neg b \vee \neg c)\}$$

et des clauses souples de poids $1$

$$\mathcal{S} = \{(a, 1), (b, 1), (c, 1)\}.$$

Les clauses dures imposent qu'au plus une des variables $a$, $b$ et $c$ soit vraie, tandis que les clauses souples expriment au contraire une préférence pour rendre vraies les trois variables. L'ensemble complet est donc insatisfaisable.

Un solveur SAT peut par exemple retourner le noyau

$$U_1 = \{(a, 1), (b, 1), (\neg a \vee \neg b)\}.$$

Ce noyau montre qu'il est impossible de satisfaire simultanément les deux clauses souples $(a, 1)$ et $(b, 1)$ tout en respectant les clauses dures. Une approche *core-guided* va alors relaxer les clauses souples du

noyau, par exemple en remplaçant $(a)$ et $(b)$ par

$$(a \vee r_a) \qquad \text{et} \qquad (b \vee r_b),$$

où $r_a$ et $r_b$ sont des variables de relaxation, puis en ajoutant une contrainte de cardinalité qui encode le fait qu'au moins une des clauses du noyau devra être abandonnée. Si, au cours des itérations suivantes, le solveur extrait également les noyaux

$$U_2 = \{(a, 1), (c, 1), (\neg a \vee \neg c)\} \qquad \text{et} \qquad U_3 = \{(b, 1), (c, 1), (\neg b \vee \neg c)\},$$

il devient clair qu'une seule des trois clauses souples peut être satisfaite. Le coût optimal est donc égal à $2$, puisqu'il faut nécessairement renoncer à deux clauses souples.

Cet exemple met en évidence l'idée centrale des méthodes *core-guided* : chaque noyau extrait prouve qu'une partie du coût optimal est inévitable, puis guide une reformulation de l'instance qui rend les itérations suivantes plus informatives. Plus largement, de nombreux solveurs exacts contemporains s'inscrivent dans le cadre des approches SAT-based, où le problème MaxSAT est reformulé en une séquence de problèmes SAT enrichis par des contraintes supplémentaires (Ansótegui *et al.*, 2013). Parmi les solveurs représentatifs de cette famille, on peut citer Open-WBO (Martins *et al.*, 2014), RC2 (Ignatiev *et al.*, 2019) ainsi qu'EvalMaxSat (Avellaneda, 2021).

À côté des approches *core-guided*, une autre famille importante est celle des approches par *implicit hitting set*. L'idée n'est plus de reformuler immédiatement l'instance après chaque noyau extrait, mais d'accumuler les noyaux découverts puis de calculer un plus petit ensemble de clauses souples qui intersecte chacun d'eux. Un tel ensemble est appelé *hitting set* : il correspond à un choix minimal de clauses souples à sacrifier pour briser tous les conflits observés. Dans l'exemple précédent, les projections des trois noyaux sur les clauses souples sont

$$\{a, b\}, \qquad \{a, c\}, \qquad \{b, c\}.$$

Chercher un *minimum hitting set* revient alors à trouver le plus petit ensemble qui rencontre chacun de ces trois ensembles. Toute solution a ici une taille $2$, par exemple $\{a, b\}$, $\{a, c\}$ ou $\{b, c\}$. On retrouve ainsi le même coût optimal que dans l'analyse *core-guided*, mais obtenu à travers une autre logique algorithmique.

Cette distinction est importante, car elle montre que plusieurs familles de solveurs exacts exploitent une même information de base, à savoir les noyaux insatisfaisables, mais de manière différente. Les méthodes

*core-guided* utilisent directement les noyaux pour reformuler progressivement l'instance, tandis que les approches *implicit hitting set* délèguent le choix des clauses à sacrifier à un sous-problème combinatoire distinct (Saikko *et al.*, 2016). D'autres approches exactes combinent encore des techniques issues de la programmation linéaire en nombres entiers, de la pseudo-booléenne ou de schémas hybrides SAT/IP. Par exemple, LMHS illustre une approche hybride SAT-IP fondée sur cette logique (Saikko *et al.*, 2016; Davies et Bacchus, 2013).

Plus généralement, les solveurs exacts modernes se distinguent moins par une opposition stricte entre familles que par la manière dont ils combinent prétraitement, raisonnement SAT, contraintes de cardinalité et techniques d'optimisation auxiliaires.

### 2.2.2 Solveurs incomplets et anytime

Les solveurs incomplets, souvent appelés solveurs *anytime* dans le contexte MaxSAT, ont un objectif différent : ils cherchent à produire rapidement une solution réalisable de bonne qualité, puis à l'améliorer au cours du temps, sans garantir l'optimalité (Zilberstein, 1996). Ils sont particulièrement adaptés aux contextes où le budget de calcul est limité ou variable, et où l'on préfère disposer rapidement d'une bonne solution plutôt que d'attendre une preuve d'optimalité.

Dans ce cadre, la qualité d'un solveur ne peut pas être évaluée uniquement à la fin de l'exécution. Il faut également considérer sa capacité à améliorer rapidement la solution, à maintenir une progression régulière et à converger vers de faibles coûts à horizon plus long. C'est pourquoi l'évaluation des solveurs anytime repose sur des protocoles différents de ceux des solveurs exacts.

## 2.3 Recherche locale stochastique pour MaxSAT

### 2.3.1 Origines et principes

La recherche locale stochastique (*Stochastic Local Search*, SLS) constitue une famille majeure de méthodes anytime pour MaxSAT (Hoos et Stützle, 2004). Ces approches prolongent des idées développées initialement pour SAT, comme GSAT et WalkSAT, où l'on part d'une affectation complète puis où l'on modifie localement certaines variables pour améliorer progressivement la solution (Selman *et al.*, 1992; Selman *et al.*, 1994).

Appliquées à MaxSAT, ces méthodes ne cherchent pas directement à établir l'optimalité. Elles explorent l'espace des affectations afin de réduire le coût courant, souvent à l'aide de mécanismes de pondération adaptative des clauses, de perturbations contrôlées ou de critères de sélection guidant les retournements de variables.

Dans la suite du document,nous désignons un tel retournement par le terme *flip*, conformément à l'usage courant dans la littérature sur les algorithmes de recherche locale pour SAT et MaxSAT (Fukunaga, 1997; Luo *et al.*, 2014; Cai *et al.*, 2013).

Leur principal avantage est leur capacité à produire rapidement de bonnes solutions, en particulier sur de grandes instances où les méthodes exactes peuvent être trop coûteuses.

### 2.3.2 Méthodes représentatives

Parmi les solveurs de recherche locale marquants, *SATLike* a constitué une référence importante pour la piste *incomplete* (Cai et Luo, 2018). Son idée centrale est de conserver un cadre de recherche locale relativement proche de celui utilisé pour SAT, tout en l'adaptant à la structure de MaxSAT par un schéma de pondération soigneusement conçu. Autrement dit, plutôt que d'introduire une machinerie spécialisée complexe distinguant fortement clauses dures et clauses souples à tous les niveaux de l'algorithme, *SATLike* cherche à exploiter cette structure principalement à travers la gestion des poids. Cette simplicité relative constitue l'un de ses intérêts : elle montre qu'un guidage efficace peut déjà être obtenu à partir d'un mécanisme de pondération bien calibré.

*NuWLS* prolonge cette ligne de travail en s'intéressant plus finement au rôle des poids dans la dynamique de recherche (Chu *et al.*, 2023). Les auteurs observent d'une part que les poids initiaux attribués aux clauses souples influencent fortement la trajectoire suivie par le solveur, et proposent pour cette raison une méthode d'initialisation dédiée. D'autre part, *NuWLS* introduit une règle de mise à jour qui distingue explicitement le traitement des clauses dures et celui des clauses souples. L'idée générale est que ces deux types de contraintes ne jouent pas le même rôle dans la recherche : les clauses dures structurent la faisabilité, tandis que les clauses souples orientent l'optimisation. En séparant leurs mécanismes de pondération, *NuWLS* obtient un guidage plus précis de la recherche locale.

*BandMaxSAT* introduit ensuite une idée différente, inspirée de l'apprentissage séquentiel (Zheng *et al.*,

2022). Le solveur associe un modèle de type *multi-armed bandit* aux clauses souples de l'instance : chaque "bras" correspond à une clause souple, et le mécanisme apprend progressivement quelles clauses il est le plus utile de viser lorsqu'il faut orienter la recherche ou sortir d'un optimum local. Le problème n'est donc plus seulement de choisir quelle variable inverser à partir d'un score local immédiat, mais aussi de déterminer quelle direction de recherche semble la plus prometteuse au regard de l'expérience accumulée. Cette approche ajoute une couche d'adaptation stratégique au-dessus de la recherche locale classique.

*FPS* (*Farsighted Probabilistic Sampling*) part du constat que beaucoup de solveurs MaxSAT en recherche locale ne considèrent qu'un seul *flip* de variable par itération (Zheng *et al.*, 2023). Or ce mécanisme peut conduire à des optima locaux de qualité limitée et réduire la diversité des trajectoires de recherche. L'idée de *FPS* est donc d'élargir localement l'horizon de décision : au lieu d'évaluer uniquement le meilleur *flip* simple, la méthode échantillonne aussi des paires de variables et estime le bénéfice potentiel de deux *flip* successifs. Le solveur choisit alors entre le meilleur mouvement simple observé et le meilleur mouvement à deux étapes parmi ceux qui ont été échantillonnés. Cette stratégie permet d'introduire une forme d'anticipation locale, qui améliore la capacité d'exploration sans changer entièrement le cadre général de la recherche locale.

Enfin, *SPB-MaxSAT* introduit une perspective encore différente, en réutilisant dans un contexte de recherche locale une idée issue des solveurs complets (Zheng *et al.*, 2024b). Plus précisément, le solveur s'appuie sur une contrainte pseudo-booléenne de conflit souple (*Soft conflict Pseudo-Boolean*, ou SPB), habituellement utilisée pour forcer la recherche d'une meilleure solution dans des méthodes exactes, et l'intègre dans le système de pondération lui-même. Au lieu de considérer cette contrainte comme une restriction externe de l'espace de recherche, *SPB-MaxSAT* la transforme en signal de guidage pour la pondération des clauses. Les auteurs y ajoutent en outre une stratégie adaptative de mise à jour des poids, qui rompt avec les ajustements constants plus traditionnels. Le résultat est une forme de guidage plus structuré, où la pondération reflète non seulement l'état local courant, mais aussi une information plus globale sur les conflits liés à l'amélioration de la solution.

Pris ensemble, ces travaux montrent que la recherche locale pour MaxSAT s'est progressivement déplacée d'une logique de voisinage relativement simple vers des mécanismes de guidage de plus en plus élaborés. Les progrès récents ne tiennent pas seulement à de meilleurs scores locaux, mais à une meilleure maîtrise de plusieurs dimensions complémentaires : l'initialisation des poids, la différenciation entre clauses dures

et clauses souples, le choix de directions de recherche prometteuses, l'anticipation de séquences de mouvements, et l'exploitation de contraintes plus structurées pour guider l'optimisation. La recherche locale MaxSAT apparaît ainsi aujourd'hui comme une famille algorithmique riche, dont les variantes se distinguent par la manière dont elles organisent et exploitent ces différentes sources d'information.

## 2.4 Solveurs anytime modernes et hybridation

### 2.4.1 Large Neighborhood Search et approches voisines

Au-delà de la recherche locale classique, plusieurs travaux ont cherché à améliorer les solveurs anytime MaxSAT en utilisant des voisinages plus structurés. Large Neighborhood Search (LNS) constitue un exemple important de cette évolution. L'idée générale est de fixer une partie de la solution courante et de réoptimiser un sous-ensemble de variables dans un voisinage plus large que celui d'un simple *flip* locale. Cette approche a été appliquée à MaxSAT dans un cadre anytime par Hickey et Bacchus, qui montrent qu'un solveur LNS peut améliorer efficacement des solutions sous-optimales produites par d'autres solveurs anytime (Hickey et Bacchus, 2022).

LNS illustre un point important de l'état de l'art : les solveurs anytime modernes ne se limitent plus à une opposition entre SAT-based d'un côté et SLS de l'autre. Ils explorent aussi des stratégies intermédiaires, capables de réutiliser une solution courante tout en effectuant une réoptimisation partielle plus globale.

### 2.4.2 L'hybridation SAT-based / SLS

Une tendance particulièrement forte dans les travaux récents est l'hybridation entre méthodes SAT-based et méthodes de recherche locale. Les raisons de cette hybridation sont bien identifiées : les approches SAT-based sont généralement efficaces pour trouver des solutions faisables et pour exploiter un raisonnement logique riche, tandis que les méthodes SLS sont souvent très efficaces pour améliorer rapidement une solution déjà disponible (Berg *et al.*, 2024b). Les deux familles apparaissent donc largement complémentaires.

Cette complémentarité est explicitement discutée dans des travaux récents. Par exemple, Lübke et Berg soulignent que les solveurs anytime MaxSAT récents combinent de plus en plus des techniques SAT-based et SLS, afin d'exploiter simultanément la qualité de raisonnement des premières et la capacité d'amélioration rapide des secondes (Lübke et Berg, 2025). Cette évolution confirme que l'état de l'art ne peut plus être décrit comme une simple juxtaposition de familles séparées : la frontière entre SAT-guided et SLS tend à

devenir plus poreuse, au profit de solveurs hybrides plus performants.

### 2.4.3 Solveurs anytime récents représentatifs

La MaxSAT Evaluation 2024 fournit un bon aperçu des solveurs anytime récents les plus compétitifs. Parmi les solveurs marquants se trouvant dans le tableau 2.1 figurent *SPB-MaxSAT-c-Band*, *SPB-MaxSAT-c-FPS*, *NuWLS-c-IBR* et *TT-Open-WBO-inc-glucose*, qui illustrent bien la diversité des approches actuelles (Berg *et al.*, 2024b; Jin *et al.*, 2024; Jiang et Chen, 2024; Nadel, 2024).

Table 2.1 – Exemples de solveurs MaxSAT anytime récents et idée principale.

| **Solveur** | **Année** | **Famille** | **Idée principale** |
|---|---|---|---|
| *NuWLS* | 2023 | SLS | Pondération adaptative de clauses pour améliorer la recherche locale (Chu *et al.*, 2023). |
| *BandMaxSAT* | 2022 | SLS | Guidage de la recherche locale par modèle multi-bras (Zheng *et al.*, 2022). |
| *FPS* | 2023 | SLS | Échantillonnage probabiliste à horizon plus large pour guider la recherche (Zheng *et al.*, 2023). |
| *SPB-MaxSAT* | 2024 | SLS | Utilisation d'une contrainte pseudo-booléenne de conflit souple à poids dynamiques (Zheng *et al.*, 2024b). |
| *SPB-MaxSAT-c-Band* | 2024 | Hybride | Combinaison de *SPB-MaxSAT-c* avec *BandMaxSAT* (Jin *et al.*, 2024). |
| *SPB-MaxSAT-c-FPS* | 2024 | Hybride | Combinaison de *SPB-MaxSAT-c* avec *FPS* (Jin *et al.*, 2024). |
| *NuWLS-c-IBR* | 2024 | Hybride | Version améliorée de *NuWLS-c* soumise aux tracks anytime MSE 2024 (Jiang et Chen, 2024). |
| *TT-Open-WBO-Inc* | 2024 | SAT-based / hybride | Solveur anytime fondé sur des techniques SAT-based et des améliorations incrémentales (Nadel, 2024). |

## 2.5 La MaxSAT Evaluation comme cadre de référence

### 2.5.1 Rôle de la compétition

La MaxSAT Evaluation (MSE) constitue aujourd'hui le principal cadre de comparaison standardisé pour les solveurs MaxSAT. Elle fournit à la fois des benchmarks, des règles communes, des descriptions de solveurs et des résultats détaillés, ce qui en fait une référence importante pour situer l'état de l'art (Berg *et al.*, 2024b).

L'édition 2024 distingue trois grands volets : un *track exact*, un *track anytime* et une vitrine *incremental*.

Dans la terminologie de la compétition, un *track* désigne une catégorie officielle d'évaluation regroupant un type de problème, un ensemble d'instances, un protocole expérimental et des règles de comparaison communes. Une vitrine désigne quant à elle un volet destiné à présenter et à mettre en valeur des approches particulières dans un cadre expérimental distinct des principales catégories de compétition.

Le *track anytime* est lui-même divisé en deux catégories, *weighted* et *unweighted*, correspondant respectivement aux instances pondérées et non pondérées. Les solveurs y sont évalués à deux horizons temporels fixes, soit 60 secondes et 300 secondes (Berg *et al.*, 2024b).

### 2.5.2 Résultats marquants de la MSE 2024

Dans le track *anytime unweighted*, la MSE 2024 classe en tête *SPB-MaxSAT-c-Band*, suivi de *NuWLS-c-IBR* et de *TT-Open-WBO-inc-glucose*, aussi bien à 60 secondes qu'à 300 secondes (Berg *et al.*, 2024b). Dans le track *anytime weighted*, *SPB-MaxSAT-c-FPS* occupe la première place à 60 secondes comme à 300 secondes, devant *TT-Open-WBO-inc-glucose* puis *NuWLS-c-IBR* (Berg *et al.*, 2024b).

Table 2.2 – Podiums officiels de la MSE 2024 pour les tracks anytime.

| **Track** | **Horizon** | **Rang** | **Solveur** |
|---|---|---|---|
| Unweighted | 60 s | 1 | *SPB-MaxSAT-c-Band* |
| | | 2 | *NuWLS-c-IBR* |
| | | 3 | *TT-Open-WBO-inc-glucose* |
| Unweighted | 300 s | 1 | *SPB-MaxSAT-c-Band* |
| | | 2 | *NuWLS-c-IBR* |
| | | 3 | *TT-Open-WBO-inc-glucose* |
| Weighted | 60 s | 1 | *SPB-MaxSAT-c-FPS* |
| | | 2 | *TT-Open-WBO-inc-glucose* |
| | | 3 | *NuWLS-c-IBR* |
| Weighted | 300 s | 1 | *SPB-MaxSAT-c-FPS* |
| | | 2 | *TT-Open-WBO-inc-glucose* |
| | | 3 | *NuWLS-c-IBR* |

Ces résultats sont importants pour deux raisons. D'une part, ils mettent en évidence le rôle central joué par les solveurs hybrides dans l'état de l'art récent. D'autre part, ils montrent que la hiérarchie des solveurs

dépend du track considéré, ce qui justifie une analyse séparée des instances *weighted* et *unweighted*.

## 2.6 Évaluation des solveurs anytime

### 2.6.1 Pourquoi l'évaluation est plus délicate

L'évaluation des solveurs anytime est plus délicate que celle des solveurs exacts, car la qualité d'une méthode ne dépend pas uniquement du résultat final au timeout. Elle dépend également de la vitesse à laquelle le solveur améliore sa solution, de la régularité de cette amélioration et de sa robustesse sur différentes instances ou différentes exécutions (Zilberstein, 1996; Hoos et Stützle, 2004).

Dans cette perspective, un classement à un instant unique peut être insuffisant. Deux solveurs peuvent obtenir des scores proches à 300 secondes tout en ayant eu des comportements très différents pendant l'exécution. L'un peut être excellent très tôt puis stagner, tandis que l'autre peut progresser lentement mais dépasser ses concurrents à long horizon.

### 2.6.2 Mesures et outils d'analyse

La littérature propose plusieurs outils pour analyser les performances de solveurs sur des ensembles d'instances. Les *profils de performance* de Dolan et Moré permettent par exemple de comparer des algorithmes à partir de performances relatives normalisées (Dolan et Moré, 2002). Lorsque les solveurs sont stochastiques, il est également important de considérer plusieurs exécutions indépendantes et d'analyser la dispersion des résultats (Hoos et Stützle, 2004).

Au-delà des classements finaux, l'analyse de trajectoires devient particulièrement pertinente pour les solveurs anytime. Elle permet d'étudier la qualité de solution en fonction du temps, d'identifier des comportements de convergence différents et de mieux comprendre les compromis entre réactivité initiale et performance finale. Cette idée apparaît de manière implicite dans les compétitions récentes, où les classements anytime sont déjà évalués à plusieurs horizons temporels (Berg *et al.*, 2024b).

### 2.6.3 Vers une lecture comportementale des solveurs

L'un des enseignements majeurs de l'état de l'art est que les solveurs anytime doivent être étudiés comme des systèmes dynamiques de recherche, et non uniquement comme des algorithmes produisant une valeur

finale. Cette perspective est particulièrement naturelle pour les méthodes SLS et hybrides, dont le comportement peut varier fortement selon la structure de l'instance, la graine aléatoire et le budget de temps (Hoos et Stützle, 2004).

Ainsi, l'évaluation expérimentale des solveurs anytime repose moins sur une unique métrique universelle que sur un ensemble d'indicateurs complémentaires capables de décrire différents aspects de la performance : qualité finale, progression temporelle, robustesse et comportement moyen sur plusieurs instances.

## 2.7 Synthèse du chapitre

L'état de l'art montre que la résolution de MaxSAT s'organise aujourd'hui autour de deux grandes dynamiques complémentaires. La première est celle des solveurs exacts, largement structurés par des approches SAT-based et core-guided. La seconde est celle des solveurs anytime, où la recherche locale stochastique, les voisinages élargis et surtout les approches hybrides jouent un rôle de plus en plus important.

Les compétitions récentes, en particulier la MaxSAT Evaluation 2024, confirment cette évolution. Elles montrent à la fois la maturité des solveurs exacts et la montée en puissance de solveurs anytime hybrides capables de combiner réactivité, amélioration progressive et qualité finale. Cet état de l'art fournit ainsi le contexte nécessaire pour comprendre les choix méthodologiques et expérimentaux présentés dans la suite du mémoire.

# CHAPITRE III

# MÉTHODOLOGIE : CONCEPTION DE MAXSAT RUNNER

Ce chapitre présente la méthodologie ayant guidé la conception de *MaxSAT Runner*, un banc d'essai destiné à l'évaluation temporelle des solveurs MaxSAT *anytime*. Il expose d'abord les limites d'une comparaison fondée uniquement sur des résultats observés à des horizons fixes. Il décrit ensuite la chaîne de traitement mise en place, depuis l'exécution contrôlée des solveurs et la collecte des améliorations jusqu'à la reconstruction des trajectoires, leur agrégation et le calcul des métriques comparatives. Enfin, il présente les principaux modes d'utilisation de l'outil.

## 3.1 Introduction

Dans la littérature, pour comparer l'efficacité des solveurs MaxSAT *anytime*, il est courant d'utiliser un score fondé sur la qualité de la solution trouvée relativement à la meilleure solution connue après un temps imparti. Cette logique est notamment utilisée dans la compétition de référence *MaxSAT Evaluation* (Berg *et al.*, 2024a), où les solveurs sont comparés à des horizons fixes, par exemple après 60 secondes ou 300 secondes. Une telle approche présente l'avantage d'être simple, standardisée et adaptée à l'établissement d'un classement global.

— Pour une instance $i$, le score d'un solveur $s$ est défini par

$$\mathrm{score}(s, i) = \frac{\mathrm{cost}_{\mathrm{best}}(i) + 1}{\mathrm{cost}_s(i) + 1}.$$

— Le score vaut $0$ si aucune solution n'est trouvée.

Cependant, cette granularité reste insuffisante lorsqu'il s'agit non plus seulement de classer des solveurs, mais d'analyser finement leur comportement au cours du temps. En effet, le développement d'un solveur MaxSAT *anytime* repose généralement sur l'ajout incrémental de nouvelles techniques et architectures qui améliorent progressivement la performance globale. Ainsi, une nouvelle stratégie peut permettre d'obtenir plus rapidement de bonnes solutions sans nécessairement modifier le coût final observé au moment du *timeout*. Dans ce cas, une comparaison fondée uniquement sur le score final ne permet pas de mettre en évidence l'apport réel de la modification.

Deux difficultés apparaissent alors. La première est un **effet de masquage dû au temps limite**. Si une version modifiée d'un solveur atteint une très bonne solution bien avant la version de base, mais que cette dernière finit par la rattraper avant le *timeout*, les deux versions recevront un score final identique, alors même que leur comportement temporel diffère nettement. La seconde est une **perte d'information liée à l'agrégation finale**. Une modification peut améliorer la performance pour certains types d'instances, réduire la variabilité entre exécutions ou accélérer la convergence initiale, sans que ces effets apparaissent clairement dans une moyenne calculée uniquement à la fin de l'exécution.

C'est dans cette perspective qu'a été développé **MaxSAT Runner**, un banc d'essai conçu pour exécuter, observer, agréger et analyser des trajectoires de résolution.

## 3.2 Objectifs de l'outil

La conception de MaxSAT Runner repose sur quatre objectifs principaux.

### 3.2.1 Prendre en compte la dynamique de convergence au cours du temps

Le premier objectif est de ne pas limiter l'évaluation des solveurs aux seuls résultats observés à la fin d'un *timeout* arbitraire. Dans le cas de solveurs *anytime*, la performance dépend aussi de la vitesse à laquelle une bonne solution est atteinte, de la régularité de la progression et du moment où surviennent les améliorations. L'outil doit donc permettre de prendre en compte cette dynamique de convergence tout au long de l'exécution.

### 3.2.2 Garantir la reproductibilité des comparaisons

Le deuxième objectif est d'assurer des comparaisons aussi robustes et reproductibles que possible. Une campagne expérimentale doit pouvoir être relancée dans des conditions comparables, et ses résultats doivent rester interprétables a posteriori. Cette exigence concerne à la fois la stabilité du protocole, la conservation des informations utiles sur les exécutions et la possibilité de vérifier ou de rejouer une comparaison.

### 3.2.3 Assurer la simplicité d'utilisation

Le troisième objectif est de conserver un outil simple à utiliser. Un banc d'essai expérimental n'est réellement utile que s'il peut être configuré et exécuté sans surcharge technique excessive. Il doit donc être suffisamment simple pour permettre de lancer rapidement des comparaisons entre solveurs, sur différents ensembles d'instances, sans devoir modifier en profondeur son fonctionnement à chaque nouvelle campagne.

### 3.2.4 Produire des résultats interprétables

Le quatrième objectif est de produire des résultats faciles à lire, à visualiser et à interpréter. L'outil ne doit pas seulement collecter des données ; il doit aussi fournir des sorties permettant de comprendre clairement les différences de comportement entre solveurs. Cette interprétabilité est indispensable pour soutenir l'analyse expérimentale et pour intégrer les résultats dans une démarche scientifique plus large.

## 3.3 Méthodes mises en œuvre pour répondre aux objectifs

Pour répondre aux objectifs précédents les choix méthodologiques suivants sont retenus. L'idée générale est de partir de l'exécution brute des solveurs, d'en extraire des événements temporels pertinents, puis de transformer ces observations en trajectoires et en indicateurs exploitables pour la comparaison.

### 3.3.1 Principe général de la chaîne de traitement

L'approche retenue dans MaxSAT Runner repose sur une chaîne de traitement en plusieurs étapes. Un solveur est d'abord exécuté sur une instance dans un cadre contrôlé. Les sorties produites pendant l'exécution sont observées afin de détecter les améliorations successives de coût ainsi que les événements de fin d'exécution. Ces événements sont ensuite horodatés, puis utilisés pour reconstruire une trajectoire temporelle décrivant l'évolution du meilleur coût connu. Enfin, ces trajectoires sont agrégées et transformées en métriques et en visualisations.

La figure 3.1 présente une vue d'ensemble de cette chaîne de traitement en la reliant directement aux principaux composants fonctionnels de l'outil. Elle met ainsi en correspondance les étapes méthodologiques et l'architecture générale de MaxSAT Runner : le moteur d'exécution prend en charge l'exécution contrôlée des

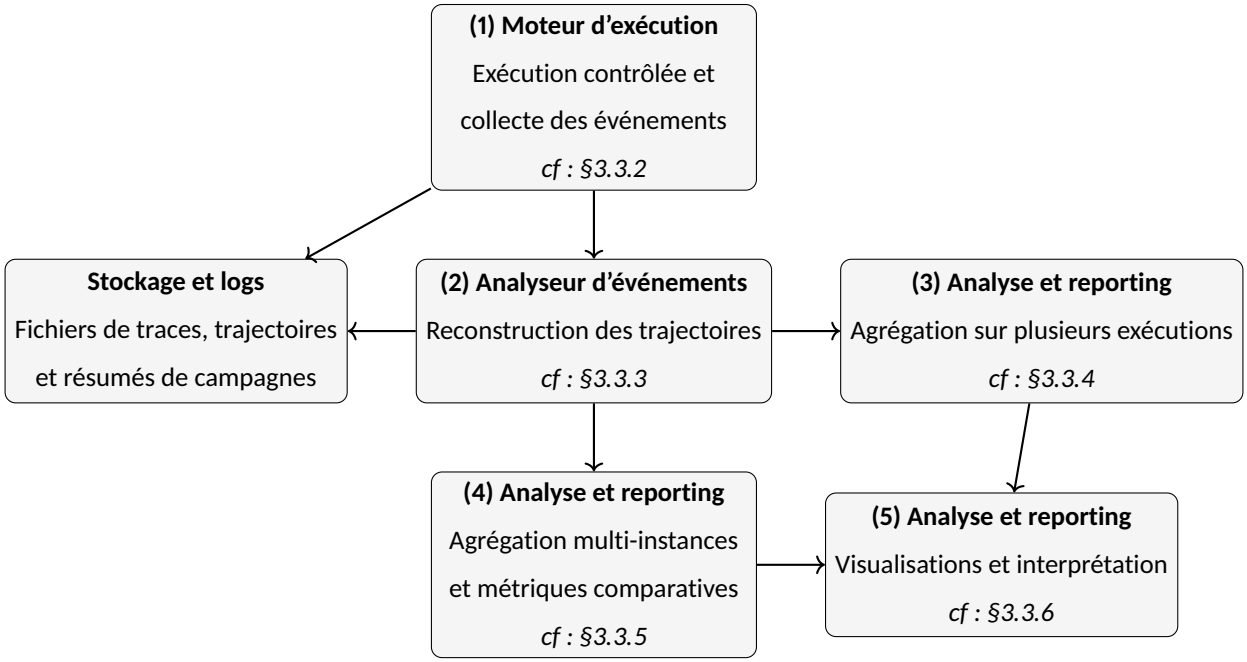



Figure 3.1 – Vue d'ensemble de la chaîne de traitement et de l'architecture générale de MaxSAT Runner. Les numéros renvoient aux sous-sections qui détaillent chaque étape.

solveurs et la collecte des événements ; l'analyseur d'événements permet la reconstruction des trajectoires à partir des sorties observées ; le stockage et les logs assurent la conservation des données produites au fil des exécutions ; enfin, le module d'analyse et de reporting regroupe les opérations d'agrégation, de calcul de métriques et de visualisation.

### 3.3.2 Exécution contrôlée et collecte des événements

Chaque solveur est exécuté comme un processus indépendant, avec un temps limite fixé à l'avance. Pendant cette exécution, les sorties textuelles sont lues au fil de l'eau afin de détecter les informations pertinentes produites par le solveur. Dans le cadre de MaxSAT, les solveurs suivent une convention de sortie standard : les lignes commençant par *c* correspondent à des commentaires, tandis que les lignes de la forme *o <coût>* signalent qu'une nouvelle meilleure solution a été trouvée, la valeur indiquée correspondant au coût de la meilleure solution connue à cet instant.

À chaque occurrence d'une ligne *o*, *MaxSAT Runner* enregistre un événement associé à un horodatage relatif mesuré depuis le début du *run*. Pour un solveur $s$, une instance $i$ et une exécution $r$, l'ensemble des données

observées est noté

$$D_{s,i,r} = \{(t_1, o_1), (t_2, o_2), \ldots, (t_k, o_k)\},$$

où chaque couple $(t_j, o_j)$ représente une amélioration observée au temps $t_j$, avec $o_j$ le coût annoncé par le solveur. Les lignes de commentaire ne participent pas à cette représentation et seules les lignes *o* alimentent l'ensemble $D_{s,i,r}$.

Le modèle retenu est donc centré sur les événements. Pour chaque exécution, *MaxSAT Runner* produit deux fichiers au format CSV. Le premier est un fichier d'événements contenant une ligne pour chaque amélioration de coût observée au cours de l'exécution.

Dans ce fichier, le champ `solver_tag` correspond à l'identifiant de la configuration du solveur, tandis que `solver_alias` désigne le nom court utilisé pour l'affichage et l'agrégation des résultats. Le champ `solver_cmd` contient la commande utilisée pour lancer le solveur, et `instance` identifie l'instance MaxSAT traitée. Le champ `run_id` distingue les différentes exécutions réalisées pour un même couple solveur–instance. Enfin, `event_idx` indique l'ordre d'apparition de l'événement, `elapsed_sec` le temps écoulé depuis le début de l'exécution, en secondes, et `cost` la meilleure valeur de coût annoncée par le solveur à cet instant.

Le second fichier est un fichier de métadonnées qui résume l'issue de chaque exécution. Il reprend les champs `solver_tag`, `solver_alias`, `solver_cmd`, `instance` et `run_id`, afin de relier les métadonnées aux événements correspondants. Le champ `optimum_found` indique si le solveur a signalé qu'une solution optimale avait été atteinte, tandis que `exit_code` contient le code de sortie du processus et permet de distinguer une terminaison normale d'une erreur d'exécution.

Cette organisation permet de préserver, dès la phase de collecte, la succession effective des améliorations observées au cours de l'exécution, tout en conservant séparément les informations de contexte et de terminaison nécessaires à l'analyse.

### 3.3.3 Reconstruction des trajectoires

À partir des fichiers d'événements et de métadonnées produits pour chaque *run*, *MaxSAT Runner* reconstruit deux fichiers globaux qui servent de base aux analyses ultérieures : *trajectories.csv* et *summary.csv*. Pour un solveur $s$, une instance $i$ et une exécution $r$, on note $c_{s,i,r}(t)$ la trajectoire temporelle reconstruite, c'est-

à-dire l'évolution du meilleur coût connu au temps $t$.

Le fichier *trajectories.csv* est obtenu par concaténation de l'ensemble des événements observés sur tous les *runs*. Chaque ligne correspond à une amélioration effectivement détectée pendant une exécution et conserve les informations d'identification du solveur et de l'instance, l'identifiant du *run*, l'indice de l'événement, le temps écoulé depuis le début de l'exécution et le coût observé. Une colonne supplémentaire, *basename*, est ajoutée afin de faciliter l'identification de l'instance dans les traitements ultérieurs.

Le fichier *summary.csv* fournit quant à lui une vue synthétique avec une ligne par exécution. Il est construit en combinant, pour chaque couple solveur–instance–*run*, le dernier événement observé dans les trajectoires avec les métadonnées de fin d'exécution. Il contient notamment les champs *solver_tag*, *solver_alias*, *solver_cmd*, *instance*, *run_id*, *final_cost*, *time_to_best_sec*, *optimum_found* et *exit_code*.

Ainsi, la reconstruction ne se limite pas à conserver des logs élémentaires : elle produit une représentation tabulaire homogène des trajectoires et de leurs résumés, directement exploitable pour les étapes suivantes d'agrégation, de calcul de métriques et de visualisation.

### 3.3.4 Agrégation des trajectoires sur plusieurs exécutions

Les solveurs MaxSAT étant non déterministes, deux exécutions d'un même solveur sur une même instance peuvent produire des trajectoires différentes. Afin d'obtenir une information plus robuste et moins sensible aux fluctuations propres à un *run* particulier, *MaxSAT Runner* permet donc d'agréger plusieurs exécutions d'un même solveur sur une même formule.

Cette agrégation est effectuée à partir des trajectoires reconstruites dans *trajectories.csv*. Pour un solveur $s$ et une instance $i$, on dispose ainsi d'un ensemble de trajectoires correspondant aux différents *runs* réalisés. L'objectif n'est alors plus d'observer une exécution isolée, mais de résumer le comportement typique du solveur sur cette instance, tout en réduisant l'effet du bruit expérimental.

En notant $R_{s,i}$ l'ensemble des exécutions disponibles pour le solveur $s$ sur l'instance $i$, et $c_{s,i,r}(t)$ la trajectoire temporelle associée au *run* $r \in R_{s,i}$, une trajectoire agrégée peut être définie par

$$\bar{c}_{s,i}(t) = \frac{1}{|R_{s,i}|} \sum_{r \in R_{s,i}} c_{s,i,r}(t).$$

Ainsi, lorsque plusieurs exécutions sont disponibles pour un même couple solveur–instance, cette agrégation permet de construire une trajectoire représentative à partir des différentes trajectoires observées. Lorsqu'au contraire une seule exécution est disponible, elle se réduit naturellement à la trajectoire de ce *run* unique. Cette étape fournit ainsi, pour chaque solveur et chaque instance, une représentation plus stable qui servira ensuite de base à l'agrégation entre plusieurs instances.

### 3.3.5 Agrégation multi-instances et métriques comparatives

Une fois les trajectoires agrégées au niveau des exécutions pour chaque couple solveur–instance, il reste à comparer les solveurs sur un ensemble d'instances. Cette étape soulève une difficulté importante : les coûts bruts ne sont pas directement comparables d'une instance à l'autre, car leurs ordres de grandeur peuvent varier fortement. Une moyenne directe des coûts serait donc peu interprétable et risquerait de donner un poids excessif à certaines instances.

Pour résoudre ce problème, *MaxSAT Runner* adopte une normalisation locale, définie séparément pour chaque instance et pour chaque instant d'observation. À un temps donné $t$, pour une instance $i$, on note $Best_i(t)$ le meilleur coût fini observé parmi les solveurs comparés. Le score relatif local d'un solveur $s$ est alors défini à partir de son coût courant $Cost_{s,i}(t)$.

Plus précisément, l'absence de solution pour un solveur à l'instant $t$ est interprétée comme un coût infini. Ainsi, si aucun solveur n'a encore produit de solution à cet instant, tous obtiennent le score $1$. Dès qu'au moins un solveur dispose d'une solution, un solveur dont le coût n'est pas encore défini reçoit le score $0$. Lorsque le meilleur coût observé vaut $0$, le score vaut $1$ pour les solveurs ayant également atteint ce coût nul, et $0$ pour les autres. Dans le cas général, le score est donné par :

$$Score_{s,i}(t) = \begin{cases} 1 & \text{si aucun solveur n'a encore trouvé de solution à l'instant } t, \\ 0 & \text{si } Cost_{s,i}(t) \text{ n'est pas défini}, \\ 1 & \text{si } Best_i(t) = 0 \text{ et } Cost_{s,i}(t) = 0, \\ 0 & \text{si } Best_i(t) = 0 \text{ et } Cost_{s,i}(t) > 0, \\ \dfrac{Best_i(t)}{Cost_{s,i}(t)} & \text{sinon.} \end{cases}$$

Cette normalisation attribue toujours la valeur $1$ au meilleur solveur observé à l'instant considéré, tandis

que les autres solveurs reçoivent un score proportionnel à leur éloignement relatif de ce meilleur coût. Elle rend ainsi les performances comparables entre instances, malgré des échelles de coûts potentiellement très différentes.

Il devient alors possible d'agréger les résultats au niveau global en calculant, pour chaque solveur, la moyenne de ses scores relatifs sur l'ensemble des instances couvertes à l'instant $t$ :

$$\overline{Score}_s(t) = \frac{1}{|I_s(t)|} \sum_{i \in I_s(t)} Score_{s,i}(t),$$

où

$$I_s(t) = \{\, i \in I \mid Cost_{s,i}(t) \text{ est défini} \,\}$$

désigne l'ensemble des instances effectivement couvertes par le solveur $s$ à l'instant $t$.

#### 3.3.5.1 Définitions formelles des indicateurs terminaux

Pour les tableaux de synthèse à l'horizon final $T$, nous utilisons les définitions suivantes. L'ensemble des instances couvertes par le solveur $s$ à l'horizon final est

$$I_s^{\text{fin}} = I_s(T) = \{\, i \in I \mid Cost_{s,i}(T) \text{ est défini} \,\}.$$

Le nombre reporté dans la colonne *Instances couvertes* est donc le cardinal $|I_s^{\text{fin}}|$.

Si $\mathcal{S}$ désigne l'ensemble des solveurs comparés, le nombre de *victoires* mesure, pour un solveur $s$, le nombre d'instances sur lesquelles il atteint le meilleur coût final observé parmi les solveurs effectivement comparables sur cette instance. En notant

$$\mathcal{S}_i(T) = \{\, s' \in \mathcal{S} \mid Cost_{s',i}(T) \text{ est défini} \,\},$$

on définit

$$Wins_s = \left| \left\{ i \in I_s^{\text{fin}} \mid Cost_{s,i}(T) = \min_{s' \in \mathcal{S}_i(T)} Cost_{s',i}(T) \right\} \right|.$$

En cas d'égalité parfaite, chaque solveur ex æquo reçoit une victoire sur l'instance considérée.

Pour chaque couple $(s, i)$ avec $i \in I_s^{\text{fin}}$, le **temps vers le meilleur coût** est défini par

$$t^\star_{s,i} = \min\{\, t \in [0, T] \mid Cost_{s,i}(t) = Cost_{s,i}(T) \,\}.$$

Autrement dit, $t^{\star}_{s,i}$ est le premier instant où le solveur atteint la meilleure valeur qu'il produira avant la fin de l'exécution. Le temps moyen vers le meilleur coût reporté dans les tableaux est alors

$$\overline{t^{\star}}_s = \frac{1}{|I_s^{\mathrm{fin}}|} \sum_{i \in I_s^{\mathrm{fin}}} t^{\star}_{s,i}.$$

Ces définitions permettent de distinguer clairement trois lectures complémentaires : la couverture finale, la domination finale par instance et la réactivité avec laquelle la meilleure valeur finale est atteinte.

Afin de résumer l'information temporelle par une seule grandeur globale, nous considérons d'abord l'aire sous la courbe du score moyen, notée $\mathrm{AUC}_s$, définie sur un intervalle $[t_{\min}, t_{\max}]$ par

$$\mathrm{AUC}_s = \int_{t_{\min}}^{t_{\max}} \overline{\mathrm{Score}}_s(t)\, dt.$$

Cette quantité mesure la performance cumulée du solveur $s$ sur l'ensemble de l'intervalle considéré. Toutefois, sa valeur dépend directement de la durée de cet intervalle. Afin d'obtenir une mesure comparable entre des intervalles de durées différentes, nous introduisons le **score de domination temporelle**, défini par

$$\mathrm{SDT}_s = \frac{1}{t_{\max} - t_{\min}} \int_{t_{\min}}^{t_{\max}} \overline{\mathrm{Score}}_s(t)\, dt = \frac{\mathrm{AUC}_s}{t_{\max} - t_{\min}}.$$

Le $\mathrm{SDT}_s$ correspond ainsi à l'aire sous la courbe du score moyen, normalisée par la durée totale de l'intervalle observé. Puisque $\overline{\mathrm{Score}}_s(t)$ prend des valeurs comprises entre $0$ et $1$, le $\mathrm{SDT}_s$ appartient également à l'intervalle $[0, 1]$. Une valeur élevée indique que le solveur demeure, en moyenne, proche des meilleures performances observées pendant une grande partie de l'exécution.

Dans le cas où l'analyse temporelle est représentée sur une échelle logarithmique, il est également utile de considérer une seconde variante, plus sensible aux performances précoces. Nous définissons alors le **score de domination temporelle logarithmique** par

$$SDT_s^{\log} = \frac{1}{\ln(t_{\max}) - \ln(t_{\min})} \int_{t_{\min}}^{t_{\max}} \overline{Score}_s(t)\, d\ln(t), \qquad t_{\min} > 0.$$

Cette seconde métrique accorde un poids relatif plus important au début de l'exécution, ce qui est particulièrement pertinent dans un contexte *anytime*, où les premières améliorations sont souvent décisives.

### 3.3.6 Visualisations et interprétation

Au-delà du calcul des scores et des agrégations, *MaxSAT Runner* vise à produire des résultats directement interprétables. Dans cette perspective, les données issues de *trajectories.csv* et de *summary.csv* sont restituées sous forme de représentations graphiques destinées à mettre en évidence les différences de comportement entre solveurs au cours du temps. L'objectif n'est pas seulement d'illustrer les résultats, mais de rendre visibles des phénomènes qui resteraient difficiles à percevoir dans de simples tableaux numériques, comme la rapidité de convergence initiale, la présence de paliers ou encore les changements de hiérarchie entre solveurs selon l'horizon temporel considéré. Les fichiers *trajectories.csv* et *summary.csv* constituent bien les bases tabulaires exploitées pour ces sorties analytiques.

Les visualisations produites par l'outil comprennent notamment des courbes de score moyen au cours du temps, des trajectoires de coût pour des instances particulières, ainsi que des vues synthétiques permettant de comparer plusieurs solveurs sur une même campagne expérimentale. Les courbes de score moyen donnent une lecture globale de la compétitivité relative des solveurs à différents instants, tandis que les trajectoires par instance permettent d'examiner plus finement la dynamique propre de certaines exécutions. Cette complémentarité entre vue agrégée et vue locale est essentielle pour interpréter correctement les résultats expérimentaux.

Une difficulté importante tient cependant à la distribution temporelle des améliorations. Dans le cas des solveurs *anytime*, de nombreuses améliorations surviennent très tôt dans l'exécution, tandis que d'autres apparaissent plus tard de manière plus espacée. Pour cette raison, l'utilisation d'une échelle logarithmique sur l'axe du temps permet souvent de mieux lire les graphiques : elle met davantage en valeur les premières phases de la recherche, où les différences entre solveurs sont souvent les plus marquées, tout en conservant une vue d'ensemble sur l'horizon complet d'exécution. Ce choix de représentation contribue directement à l'objectif d'interprétabilité poursuivi par *MaxSAT Runner*.

Ainsi, les visualisations ne constituent pas un simple complément de présentation. Elles font partie intégrante de la démarche méthodologique, en fournissant un support de lecture adapté à l'analyse de comportements *anytime*, là où une comparaison limitée à des valeurs finales ou à quelques horizons fixes risquerait de masquer une partie importante de l'information.

## 3.4 Mode d'utilisation

MaxSAT Runner a été conçu pour être utilisé selon deux modalités complémentaires : en ligne de commande, pour les campagnes expérimentales scriptées, et via une interface web, pour une utilisation plus interactive. Dans les deux cas, la logique générale reste la même : définir une campagne, exécuter les solveurs sur un ensemble d'instances, puis exploiter les résultats produits.

La première étape consiste à configurer une campagne expérimentale. L'utilisateur renseigne le répertoire des instances, le motif de sélection des fichiers, le répertoire de sortie, l'identifiant de la campagne ainsi que le temps limite d'exécution. Il définit ensuite la liste des solveurs à comparer, chaque solveur étant associé à un alias et à une commande contenant le chemin de l'instance à traiter. En ligne de commande, une campagne peut par exemple être lancée comme suit 3.1 :

Listing 3.1 – Exemple de lancement d'une campagne MaxSAT Runner en ligne de commande.

```
maxsat-runner run \
  --solver "solverA=/chemin/solverA {inst}" \
  --solver "solverB=/chemin/solverB --old-format {inst}" \
  --instances data/instances/demo \
  --pattern .wcnf \
  --out data/runs \
  --tag demo_cli \
  --timeout-sec 30
```

Dans cette commande, chaque option *–solver* doit obligatoirement contenir le motif *{inst}*, qui est remplacé automatiquement par le chemin absolu de l'instance traitée.

Une fois cette configuration validée, l'outil lance les solveurs de manière contrôlée et enregistre, pour chaque exécution, les événements observés ainsi que les métadonnées associées. La Figure 3.2 illustre cette première phase de paramétrage et de lancement d'une campagne à travers l'interface web.

Listing 3.2 – Exemple de génération de statistiques en ligne de commande.

```
maxsat-runner stats \
  --runs data/runs \
  --out data/reports \
  --by solver_alias
```

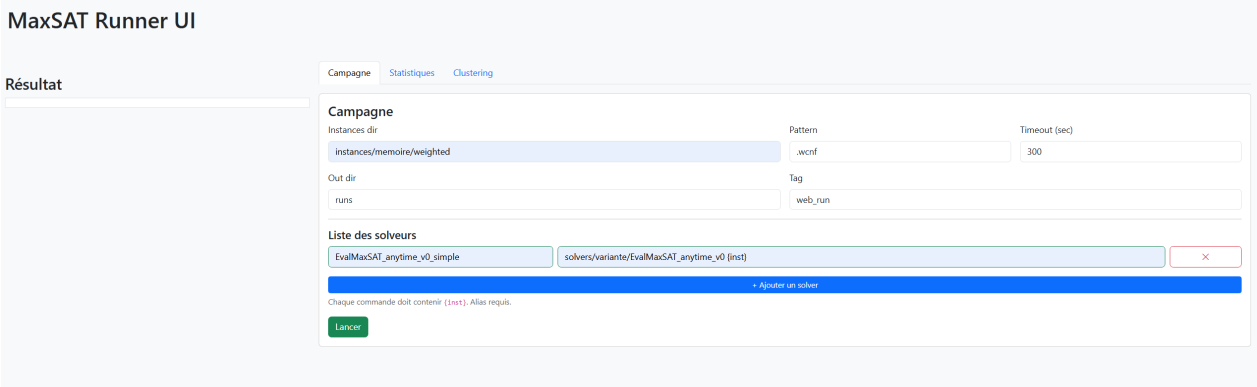



Figure 3.2 – Interface de configuration d'une campagne dans MaxSAT Runner.

La seconde étape correspond à l'exploitation statistique des exécutions produites. À partir des journaux générés lors des campagnes, MaxSAT Runner reconstruit les fichiers globaux de trajectoires et de résumé, puis produit différents résultats d'analyse.

L'utilisateur peut notamment choisir une clé d'agrégation, restreindre l'étude à une fenêtre temporelle donnée, fixer un instant d'observation particulier ou encore se concentrer sur une instance précise. L'outil génère alors des tableaux récapitulatifs et plusieurs visualisations, comme les trajectoires de coût, les scores relatifs au cours du temps ou les synthèses par solveur. Cette phase d'analyse est illustrée à la Figure 3.3.

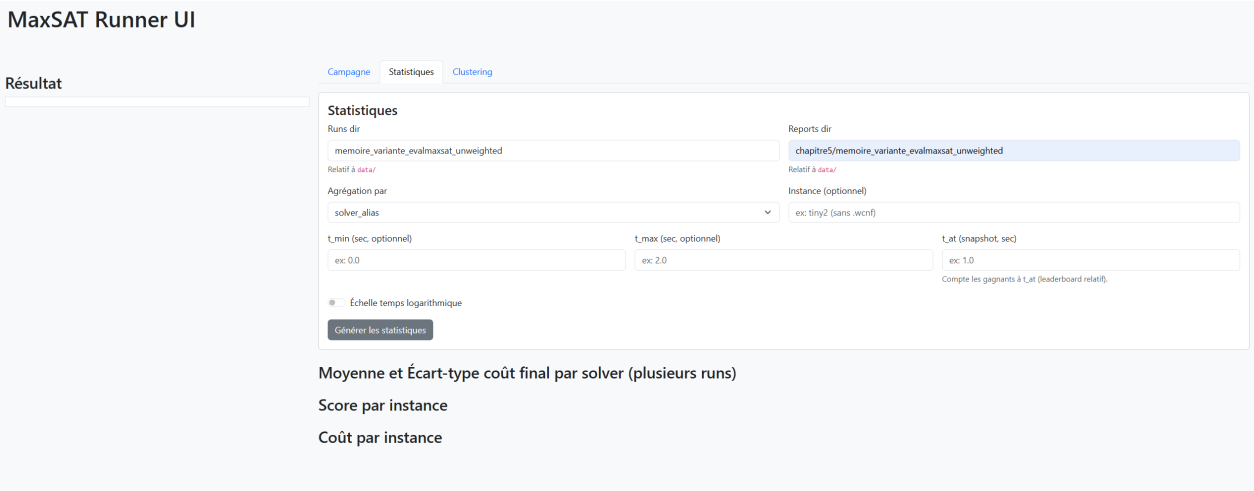



Figure 3.3 – Interface de génération des statistiques et des visualisations dans MaxSAT Runner.

L'interface web reprend ainsi l'organisation générale de l'outil en deux volets principaux : une page de campagne pour lancer les exécutions et une page de statistiques pour produire les rapports et les visualisations.

Un mode d'emploi plus opérationnel de *MaxSAT Runner*, comprenant les exemples d'installation, de lancement de campagnes, de génération de statistiques et de clustering, est donné à l'annexe A.1.

## 3.5 Résumé du chapitre

Ce chapitre présente la méthodologie retenue pour la conception de **MaxSAT Runner**, un outil destiné à l'évaluation de solveurs MaxSAT *anytime*. Nous montrons d'abord que les approches classiques fondées sur un score observé à horizon fixe, bien qu'utiles pour établir un classement global, restent limitées lorsqu'il s'agit d'analyser finement la dynamique de convergence des solveurs au cours du temps.

À partir de ce constat, nous définissons ensuite les principaux objectifs assignés à l'outil : prendre en compte l'évolution temporelle des performances, garantir la reproductibilité des comparaisons, conserver une utilisation simple et produire des résultats directement interprétables. Ces objectifs servent de cadre à l'ensemble des choix méthodologiques présentés dans la suite du chapitre.

Nous décrivons ensuite la chaîne de traitement mise en œuvre dans MaxSAT Runner, depuis l'exécution contrôlée des solveurs et la collecte des événements jusqu'à la reconstruction des trajectoires, leur agré-

gation et le calcul des différentes métriques d'analyse. Enfin, nous présentons à un niveau synthétique ses principaux modes d'utilisation, en ligne de commande comme via l'interface web.

Ainsi, MaxSAT Runner ne se limite pas à un simple lanceur de solveurs : il constitue un cadre méthodologique complet pour observer, comparer et analyser le comportement temporel de solveurs MaxSAT *anytime*. Le chapitre suivant s'appuie sur cet outil pour présenter les campagnes expérimentales réalisées et les résultats obtenus.

# CHAPITRE IV
# MAXSAT RUNNER : RÉSULTATS EXPÉRIMENTAUX

## 4.1 Introduction

Ce chapitre présente les résultats expérimentaux obtenus à l'aide de *MaxSAT Runner* sur les instances des tracks *anytime* de la MaxSAT Evaluation 2024. Son objectif n'est pas seulement de comparer plusieurs solveurs MaxSAT à partir d'indicateurs finaux usuels, mais également de mettre à l'épreuve le cadre méthodologique introduit au chapitre précédent, fondé sur l'analyse des trajectoires *anytime*. Il s'agit plus précisément d'examiner dans quelle mesure l'observation continue de l'évolution des solutions permet d'enrichir l'interprétation du comportement des solveurs, au-delà d'une lecture limitée à quelques horizons fixes ou à un classement final.

Dans cette perspective, l'organisation du chapitre accorde une place centrale aux **visualisations agrégées des trajectoires**, qui constituent l'expression la plus directe de l'apport méthodologique de *MaxSAT Runner*. Ces représentations permettent d'étudier la dynamique temporelle des solveurs, d'identifier leurs profils de progression, de repérer d'éventuels changements de hiérarchie au cours du temps et de mieux distinguer les comportements associés aux horizons courts, intermédiaires et longs.

L'analyse est conduite séparément sur les instances *unweighted* et sur les instances *weighted*. Ce choix est cohérent avec la structuration même de la MaxSAT Evaluation 2024, qui distingue explicitement ces deux catégories. Il répond aussi à une exigence de rigueur analytique, en évitant une agrégation trop hétérogène susceptible de mêler des comportements de résolution de nature différente et d'affaiblir la portée interprétative des résultats.

Les expériences rapportées dans ce chapitre ont été relancées sur le superordinateur *Rorqual*, opéré par Calcul Québec et mis à disposition de la communauté de recherche par l'Alliance de recherche numérique du Canada (Digital Research Alliance of Canada, 2024; Calcul Québec, 2025b; Calcul Québec, 2025a). Cette campagne porte sur l'ensemble des instances du track *anytime unweighted* (216 instances) et du track *anytime weighted* (229 instances) de la MaxSAT Evaluation 2024 (Berg *et al.*, 2024a; MaxSAT Evaluation 2024, 2024). Ce choix permet de replacer l'évaluation dans un cadre expérimental plus large, plus robuste et plus reproductible.

## 4.2 Protocole expérimental

Les expériences ont été conduites sur les instances des tracks *anytime* de la MaxSAT Evaluation 2024 (Berg *et al.*, 2024b; MaxSAT Evaluation 2024, 2024). La campagne couvre l'ensemble des 216 instances du track *unweighted* et l'ensemble des 229 instances du track *weighted*. Dans certaines analyses agrégées à horizon temporel fixé, le nombre d'instances effectivement prises en compte peut toutefois être légèrement inférieur lorsqu'aucun solveur ne produit de résultat exploitable sur une instance dans la fenêtre temporelle considérée. L'étude repose ainsi sur un corpus large, représentatif des deux grandes catégories d'instances utilisées dans les compétitions récentes.

Les solveurs comparés correspondent aux principaux systèmes évalués dans le cadre de la MaxSAT Evaluation 2024 (Berg *et al.*, 2024b; Berg *et al.*, 2024a). Nous considérons *NuWLS-c-IBR*, un solveur de recherche locale avancée (Jiang et Chen, 2024), *SPB-MaxSAT-c* et ses variantes *Band* et *FPS*, qui combinent des mécanismes de recherche locale et des variantes récentes de pondération heuristique (Jin *et al.*, 2024), ainsi que *TT-Open-WBO-Inc*, un solveur anytime basé sur SAT incrémental (Nadel, 2024). Dans notre campagne, *TT-Open-WBO-Inc* est exécuté avec deux solveurs SAT sous-jacents distincts, *IntelSAT* et *Glucose*, afin d'évaluer l'effet du backend SAT à architecture MaxSAT globale inchangée.

La campagne expérimentale a été exécutée sur le cluster *Rorqual* de Calcul Québec. Les exécutions ont été orchestrées par le gestionnaire de tâches *Slurm*, de manière à paralléliser systématiquement les benchmarks.

Chaque tâche correspond à un couple solveur–instance. Pour chaque couple, trois exécutions indépendantes ont été réalisées afin de tenir compte de la variabilité éventuelle des solveurs. Chaque exécution a été lancée avec un temps limite de 300 secondes. Le choix de cet horizon temporel s'appuie directement sur le protocole de la MaxSAT Evaluation, avec lequel nous cherchons à établir une comparaison cohérente. Dans la configuration retenue, jusqu'à 50 tâches ont été exécutées en parallèle, avec une allocation de 8 cœurs et 32 Go de mémoire par tâche.

Les instances, les exécutables des solveurs et le code source de *MaxSAT Runner* ont été conservés dans l'espace projet, utilisé pour le stockage durable des ressources expérimentales. Les données intermédiaires générées pendant les exécutions ont quant à elles été produites dans l'espace *scratch*, mieux adapté au stockage temporaire de fichiers volumineux et aux opérations intensives de lecture et d'écriture. Chaque

exécution produit un journal d'événements *anytime* ainsi qu'un ensemble de métadonnées de fin d'exécution. Cette organisation permet de dissocier clairement la phase de calcul massif de la phase d'agrégation.

Pour chaque couple solveur–instance, les trois exécutions sont ensuite agrégées en une trajectoire représentative selon la procédure décrite à la section 3.3.4. Les résultats sont finalement consolidés à l'aide de la commande d'agrégation de *MaxSAT Runner*, qui reconstruit les trajectoires complètes, calcule les statistiques de synthèse et génère les visualisations nécessaires à l'analyse.

L'outil produit également un graphique de distribution des scores au cours du temps. Pour chaque solveur et à chaque instant, ce graphique présente le score relatif moyen ainsi que les scores minimal et maximal observés sur les instances exploitables. La courbe moyenne est ainsi accompagnée d'une bande min–max représentant l'étendue des performances. Cette bande permet de visualiser la dispersion globale des scores. La variance et les intervalles de confiance ne sont pas affichés dans les figures principales afin de préserver leur lisibilité et d'éviter une surcharge visuelle.

Dans ce chapitre, l'analyse repose sur trois niveaux de lecture complémentaires. Le premier est une lecture visuelle et agrégée des trajectoires au cours du temps. Le deuxième est une lecture synthétique au moyen d'indicateurs calculés à plusieurs horizons temporels. Le troisième est une lecture qualitative à l'échelle de quelques instances individuelles. Enfin, la comparaison avec la MSE 2024 est conduite dans une logique de mise en perspective : le but n'est pas de reproduire mécaniquement le classement officiel, mais d'examiner ce que l'analyse des trajectoires apporte à l'interprétation des performances.

## 4.3 Résultats agrégés sur les instances *unweighted*

Cette section présente les résultats obtenus sur l'ensemble des instances *unweighted*. L'objectif est d'évaluer les solveurs non seulement à partir de leur position finale, mais aussi à partir de la forme de leurs trajectoires au fil du temps. Dans cette perspective, nous commençons par les visualisations agrégées produites par *MaxSAT Runner*, avant de les mettre en perspective avec les résultats officiels de la MSE 2024.

### 4.3.1 Visualisations agrégées des trajectoires

Nous examinons d'abord les trajectoires agrégées sur l'ensemble des instances *unweighted*. Afin de rendre simultanément visibles les toutes premières phases de la résolution et l'évolution à moyen et long terme,

nous retenons ici une représentation unique du score relatif moyen sur l'horizon complet de 300 secondes, avec une abscisse en échelle logarithmique. À titre complémentaire, nous présentons également l'évolution du coût moyen au fil du temps. Cette seconde visualisation apporte un éclairage supplémentaire sur la dynamique brute d'amélioration des solveurs.

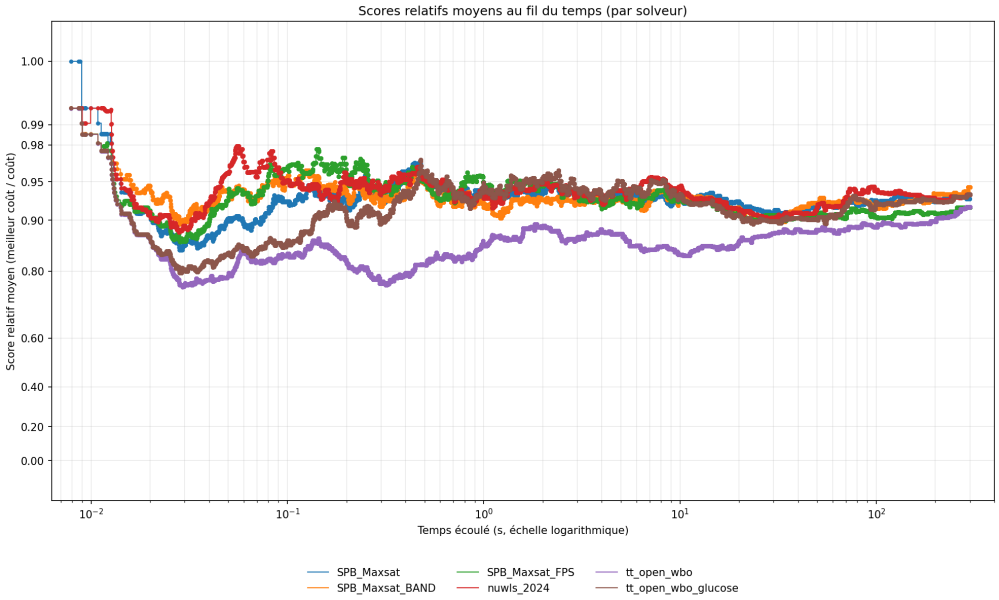



Figure 4.1 – Évolution du score relatif moyen sur les instances *unweighted* sur l'horizon complet de 300 secondes, avec une abscisse logarithmique. Cette vue d'ensemble permet d'observer la dynamique générale des solveurs sur toute la durée d'exécution.

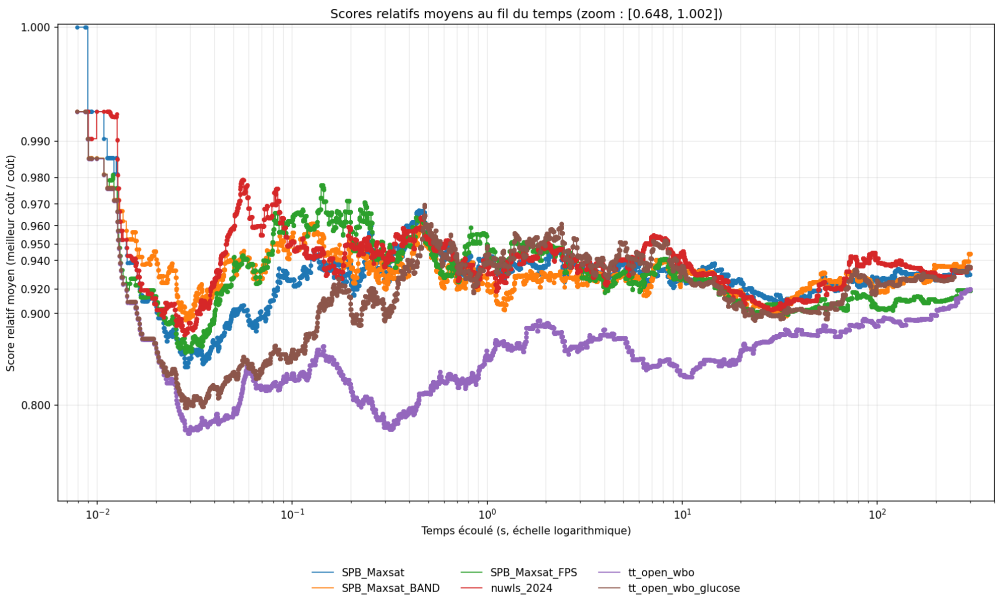



Figure 4.2 – Vue complémentaire de l'évolution du score relatif moyen sur les instances *unweighted*, sur le même horizon de 300 secondes et avec la même abscisse logarithmique, mais avec un zoom sur la zone utile de l'axe des ordonnées afin de mieux faire apparaître les écarts fins entre solveurs.

La Figure 4.1 met d'abord en évidence le caractère instable des toutes premières fractions de seconde. La Figure 4.2 complète cette lecture en resserrant l'axe des ordonnées, ce qui permet de mieux visualiser les différences de performance lorsque les courbes deviennent très proches les unes des autres. À cette échelle, les trajectoires fluctuent fortement, ce qui reflète à la fois la rapidité des premières améliorations et le fait que tous les solveurs ne couvrent pas encore exactement les mêmes instances au même instant. Cette phase initiale reste néanmoins informative, car elle montre que des écarts de réactivité apparaissent très tôt.

Lorsque l'on s'éloigne des tout premiers instants, la hiérarchie moyenne devient plus lisible. *NuWLS-2024* reste de façon générale dans le groupe de tête sur l'ensemble de la trajectoire utile. *SPB-MaxSAT-BAND* et *SPB-MaxSAT* demeurent eux aussi très compétitifs, avec des profils proches mais non identiques. *TT-Open-WBO-Inc* dans sa variante *Glucose* reste relativement proche de ce groupe de tête, tandis que la variante *IntelSAT* apparaît plus souvent légèrement en retrait. *SPB-MaxSAT-FPS* est globalement compétitif, mais moins régulier que les meilleurs solveurs dans cette catégorie.

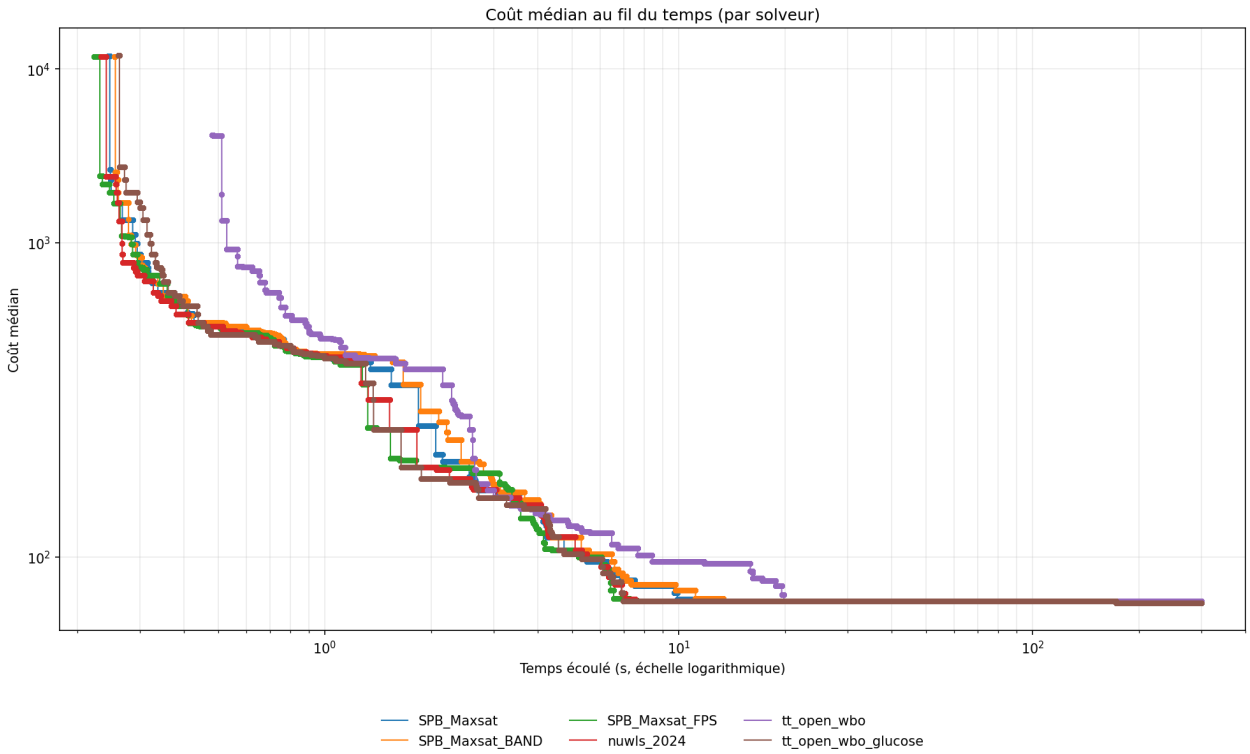



Figure 4.3 – Évolution du coût médian sur les instances *unweighted* sur l'horizon complet de 300 secondes, avec des axes logarithmiques afin de représenter les différents ordres de grandeur.

La Figure 4.3 présente l'évolution du coût médian pour chaque solveur. Les coûts diminuent rapidement au début de l'exécution, puis convergent progressivement vers des valeurs très proches entre 0 et 100. La variante de *TT-Open-WBO-Inc* utilisant *IntelSAT* se distingue principalement par une diminution plus tardive durant les premières secondes.

Les instances sans solution sont temporairement associées à un coût infini, La médiane limite ainsi l'influence des valeurs extrêmes, mais les coûts bruts restent non normalisés.

Dans l'ensemble, ces visualisations confirment que les différences entre solveurs *unweighted* existent, mais restent modestes lorsque l'on considère les scores relatifs normalisés. Les figures de score permettent de comparer les performances sur une base commune, tandis que le coût médian fournit une lecture complémentaire de l'évolution des solutions.

### 4.3.2 Mise en perspective avec la MSE 2024

Les résultats précédents peuvent être comparés, de manière qualitative, aux résultats officiels de la MaxSAT Evaluation 2024 sur le track *unweighted*. Dans la présentation officielle, seuls les meilleurs variants de chaque solveur sont retenus dans les graphiques de classement, et les scores moyens à 60 et 300 secondes placent *SPB-MaxSAT-c-Band* légèrement devant *NuWLS-c-IBR*, puis *TT-Open-WBO-inc-glucose*. Les écarts restent faibles, tant à 60 secondes qu'à 300 secondes (Berg *et al.*, 2024a).

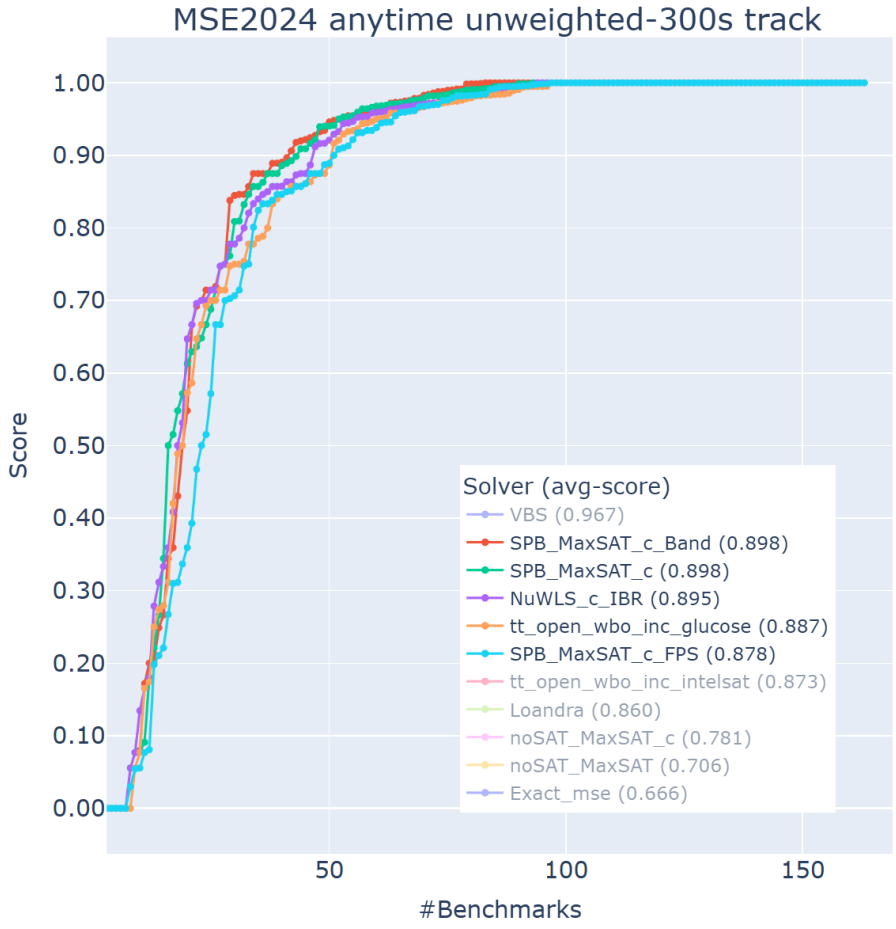



Figure 4.4 – Courbe officielle de la MSE 2024 pour le track *anytime unweighted* à 300 secondes. Source : MaxSAT Evaluation 2024 (Berg *et al.*, 2024a).

La Figure 4.4 propose une synthèse des performances des solveurs à l'horizon de 300 secondes. Si cette représentation permet de comparer globalement les solveurs sur l'ensemble des instances, elle présente néanmoins certaines limites du point de vue de l'interprétation. La forte superposition des courbes réduit

sensiblement la lisibilité de la figure : les solveurs de tête apparaissent fortement resserrés, ce qui rend difficile l'identification visuelle d'une hiérarchie claire entre eux. Cette proximité reflète certes la compétitivité réelle des solveurs, mais elle limite la capacité de la figure à faire apparaître des différences fines de comportement.

En outre, cette représentation repose sur une métrique agrégée à horizon fixe. Elle ne donne aucune information sur la dynamique temporelle ayant conduit au score final. Or deux solveurs ayant des scores finaux similaires peuvent suivre des trajectoires très différentes : l'un peut trouver rapidement une bonne solution puis stagner, tandis qu'un autre progresse plus lentement mais reste plus compétitif sur une plus grande partie de l'exécution. Ces différences sont masquées dans la représentation officielle.

À l'inverse, les visualisations produites par *MaxSAT Runner* rendent visibles la réactivité initiale, les phases de progression intermédiaire et les tendances à long horizon. Elles n'ont pas vocation à remplacer la métrique officielle, mais à la compléter par une lecture plus riche du comportement effectif des solveurs.

### 4.3.3 Lecture complémentaire par indicateurs synthétiques

Les visualisations agrégées gagnent à être complétées par des indicateurs synthétiques. Le Tableau 4.1 présente les principales métriques temporelles calculées sur les instances *unweighted*.

L'AUC correspond à l'aire sous la courbe du score relatif moyen et mesure ainsi la performance cumulée du solveur sur l'ensemble de l'intervalle observé. Le SDT correspond à cette aire normalisée par la durée de l'intervalle ; il prend des valeurs comprises entre $0$ et $1$ et représente la compétitivité moyenne du solveur au cours du temps.

Les métriques AUC-log et SDT-log reposent sur une intégration selon une échelle temporelle logarithmique, ce qui accorde davantage d'importance aux performances obtenues pendant les premières phases de l'exécution.

Table 4.1 – Métriques de domination temporelle sur les instances *unweighted*.

| Solveur | AUC | SDT | AUC-log | SDT-log |
|---|---|---|---|---|
| NuWLS-2024 | **280.11669** | **0.93375** | **9.86799** | **0.94707** |
| SPB-MaxSAT-BAND | 279.09248 | 0.93035 | 9.51419 | 0.94452 |
| SPB-MaxSAT | 278.93508 | 0.92981 | 9.95990 | 0.94429 |
| TT-Open-WBO-glucose | 277.63886 | 0.92553 | 8.82968 | 0.92613 |
| TT-Open-WBO | 276.13955 | 0.92054 | 8.63111 | 0.91411 |
| SPB-MaxSAT-FPS | 273.97181 | 0.91327 | 9.48680 | 0.93317 |

Ces résultats montrent que, sur l'ensemble de l'horizon, *NuWLS-2024* domine les principales métriques temporelles en échelle linéaire et logarithmique, à l'exception de l'AUC-log où *SPB-MaxSAT* conserve un léger avantage. Cette lecture conduit à interpréter *NuWLS-2024* comme le solveur le plus *constant* sur l'ensemble de la trajectoire. Il ne domine pas nécessairement à tout instant isolé, mais il reste durablement proche des meilleures performances observées.

La lecture terminale raconte toutefois une histoire légèrement différente. Le Tableau 4.2 présente plusieurs indicateurs calculés à l'horizon final. Les colonnes *Victoires*, *Instances couvertes* et *Temps moyen vers le meilleur coût* renvoient aux définitions formelles données à la section 3.3.5.1.

Table 4.2 – Indicateurs finaux sur les instances *unweighted*.

| Solveur | Victoires | Inst. couvertes | Coût final moyen | Temps moyen vers le meilleur coût (s) | Taux d'optimum (%) |
|---|---|---|---|---|---|
| SPB-MaxSAT-BAND | 135 | 212 | 262.185535 | 40.171027 | 19.81 |
| SPB-MaxSAT | 130 | 212 | 264.400943 | 40.746816 | 19.34 |
| NuWLS-2024 | 128 | 211 | 272.031447 | 34.740117 | 19.34 |
| SPB-MaxSAT-FPS | 128 | 211 | 282.195893 | 42.960350 | 17.69 |
| TT-Open-WBO-glucose | 127 | 211 | 264.331754 | 36.765155 | 19.67 |
| TT-Open-WBO | 120 | 209 | 283.966507 | 35.153611 | 13.72 |

Cette seconde lecture montre que *SPB-MaxSAT-BAND* domine davantage les indicateurs finaux. Il obtient le plus grand nombre de *victoires*, le meilleur coût final moyen, et une couverture légèrement supérieure à celle des solveurs les plus proches. Autrement dit, *NuWLS-2024* apparaît comme le solveur le plus régulier

sur la trajectoire complète, alors que *SPB-MaxSAT-BAND* ressort davantage comme le solveur le plus fort dans la lecture terminale. Cette dissociation est méthodologiquement importante : elle montre qu'une hiérarchie fondée uniquement sur le coût final ou sur le nombre de *wins* ne coïncide pas forcément avec une hiérarchie fondée sur la domination temporelle.

On peut aussi remarquer que *TT-Open-WBO-glucose* domine systématiquement *TT-Open-WBO* dans les principales lectures agrégées. Ce résultat suggère que, dans le cadre *unweighted* étudié ici, le choix du backend SAT influe réellement sur la qualité globale des trajectoires.

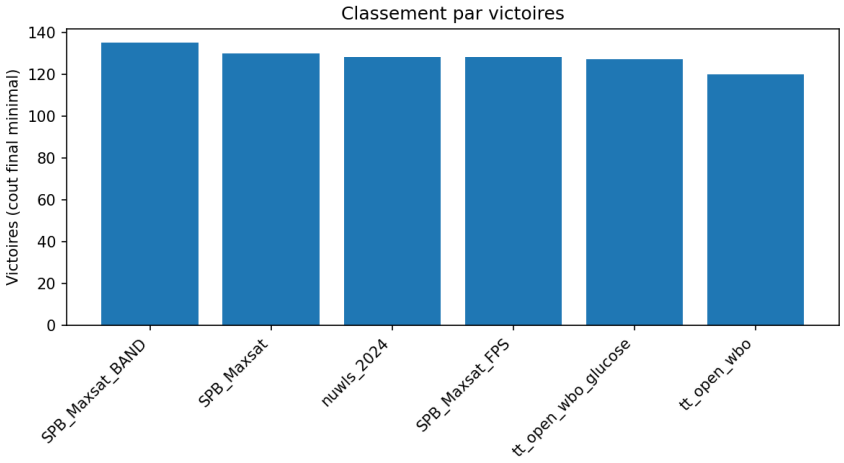



Figure 4.5 – Nombre de *victoires* par solveur sur les instances *unweighted*.

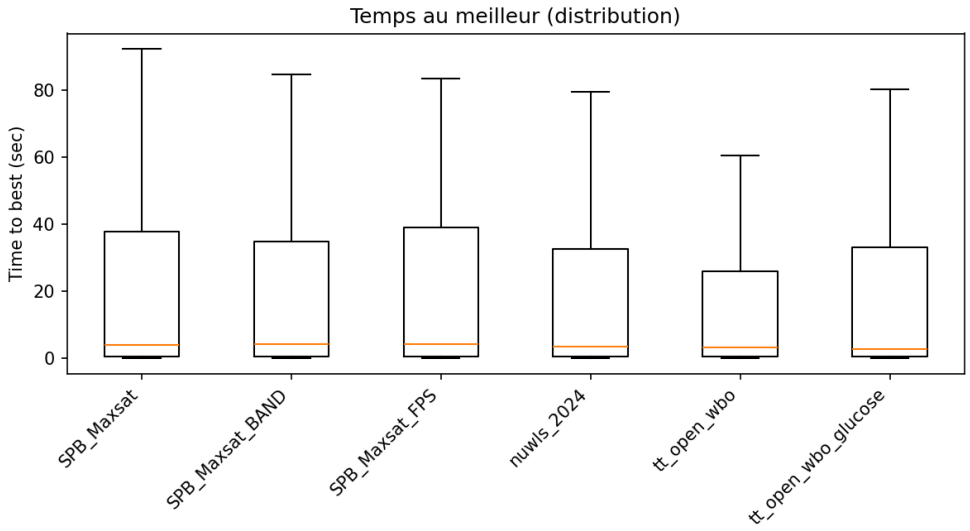


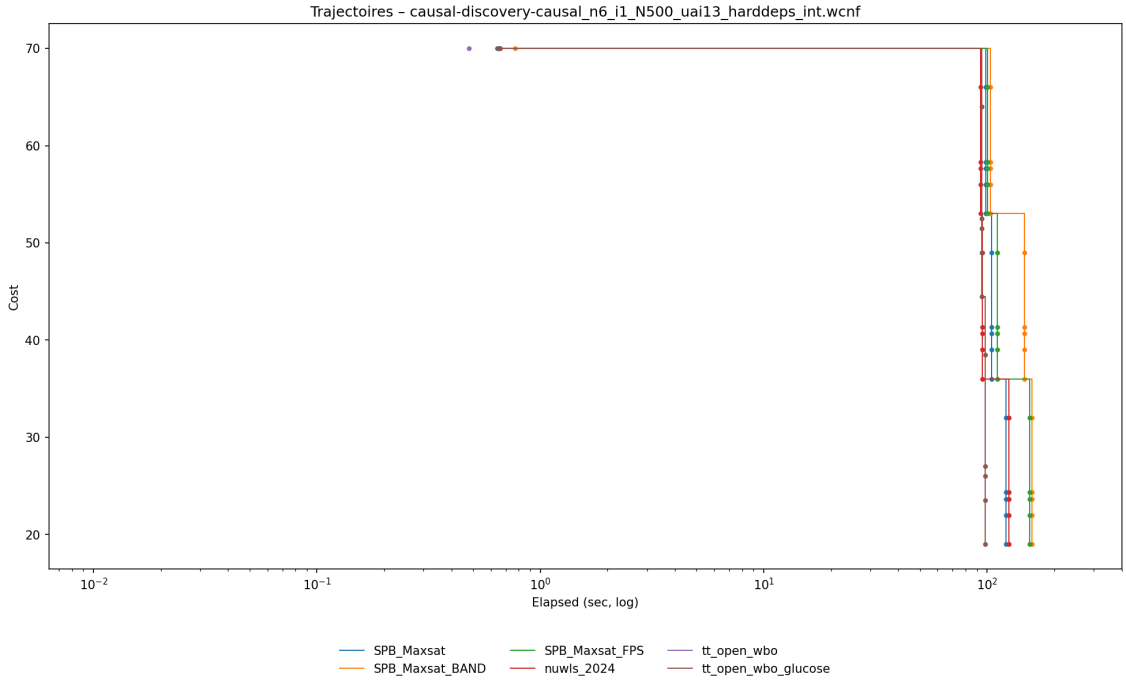

Figure 4.6 – Distribution du temps nécessaire pour atteindre le meilleur coût sur les instances *unweighted*.

Les Figures 4.5 et 4.6 confirment cette lecture complémentaire. La Figure 4.6 présente, pour chaque solveur, la distribution du temps nécessaire pour atteindre le meilleur coût obtenu sur chaque instance. Dans ce diagramme en boîte, le trait central représente la médiane, la boîte les premier et troisième quartiles, et les moustaches l'étendue des valeurs non atypiques. Les écarts restent globalement modestes, mais ils montrent que les solveurs les plus performants ne se distinguent pas seulement par la qualité finale de leurs solutions, mais aussi par la manière dont ils les atteignent.

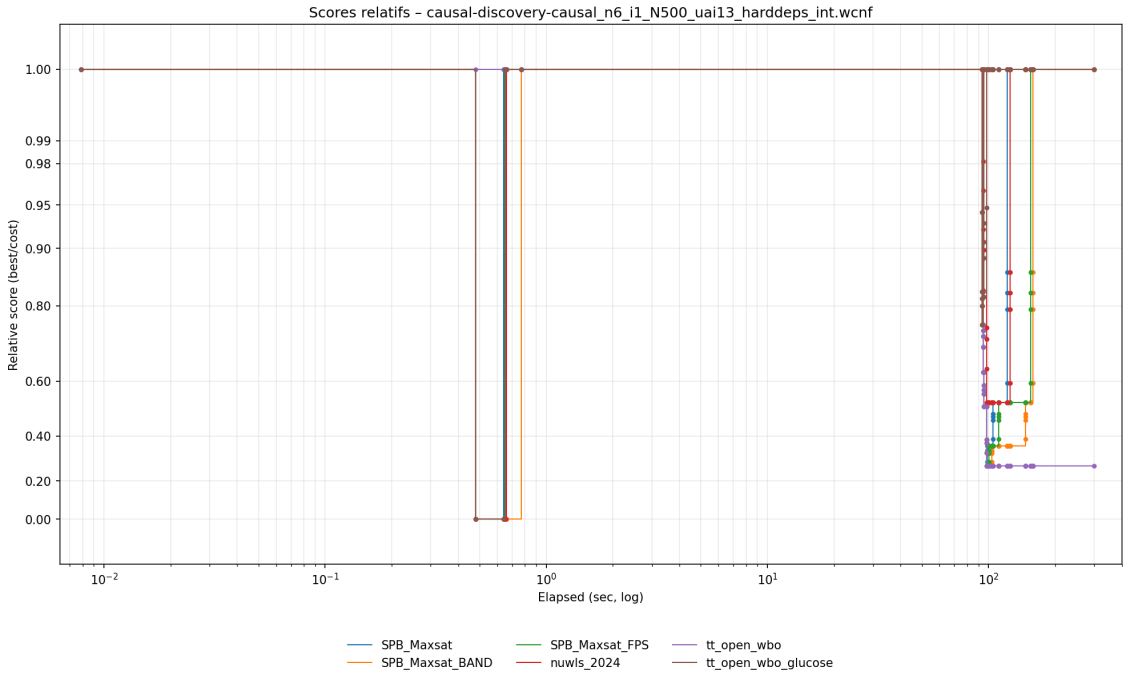
### 4.3.4 Illustration quantitative sur quelques instances

Les analyses agrégées donnent une vue globale robuste, mais elles peuvent lisser certaines différences locales. Afin de compléter cette lecture, nous présentons ici quelques instances *unweighted* représentatives. L'objectif n'est pas de substituer des exemples isolés aux tendances générales, mais d'illustrer concrètement certains comportements observés dans les agrégations, comme des changements de hiérarchie (l'ordre dans la performance ) , des stagnations prolongées ou des améliorations tardives.

#### 4.3.4.1 Instance *causal-discovery-causal_n6_i1_N500_uai13_harddeps_int.wcnf*.




(a) Trajectoires de coût.

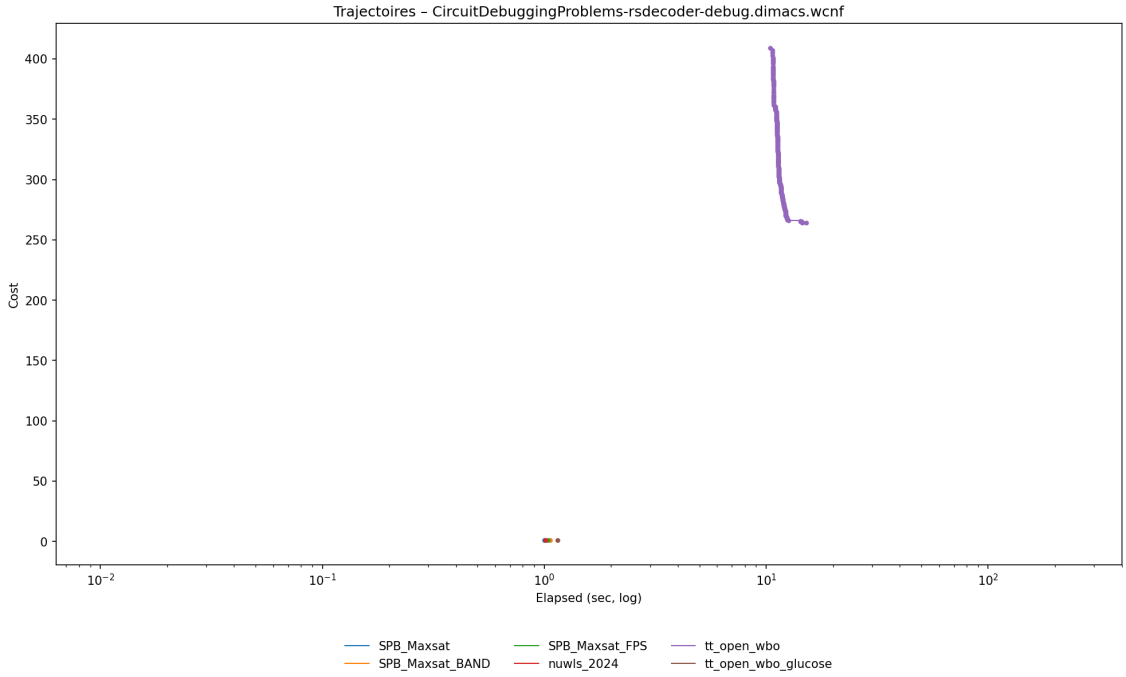



(b) Trajectoires de score relatif.

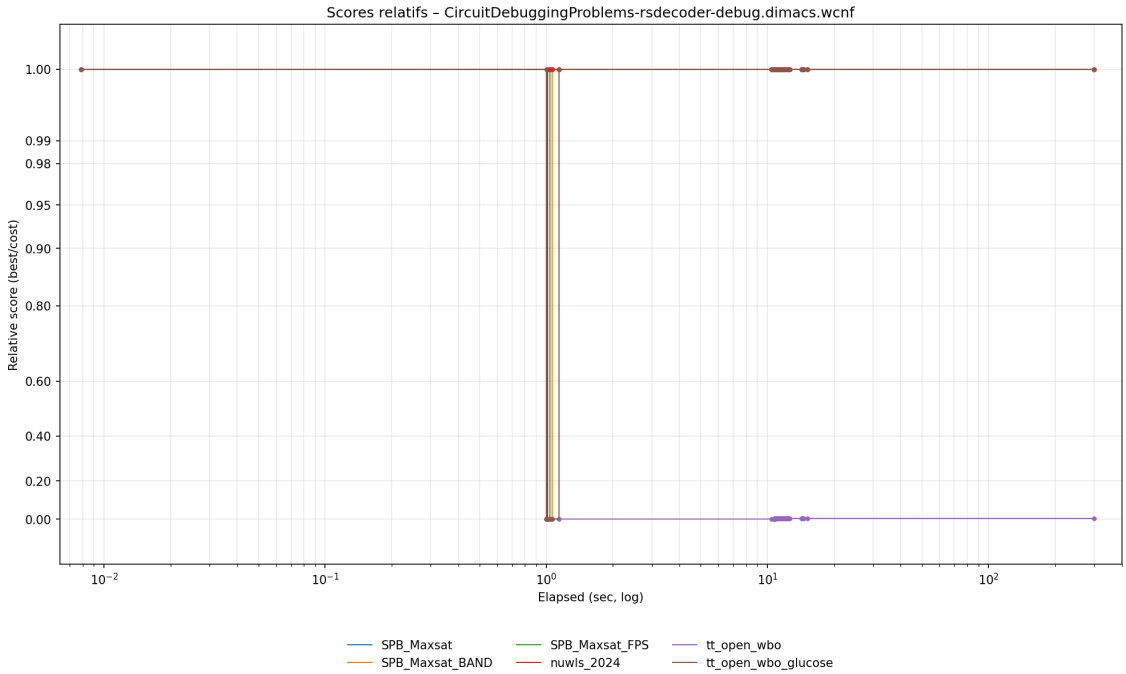
Figure 4.7 – Illustration quantitative sur l'instance `causal-discovery-causaln6i1N500uai13harddepsint.wcnf`.

Sur cette instance, les trajectoires de coût montrent un comportement relativement similaire pour l'ensemble des solveurs : la solution initiale reste stable pendant une longue période, suivie d'améliorations tardives concentrées dans une fenêtre temporelle réduite. Cette dynamique se traduit, dans l'espace des scores relatifs, par une transition abrupte entre des scores faibles et un score optimal. Aucun solveur ne domine clairement l'ensemble de l'exécution. Cette instance illustre donc un cas où la difficulté réside principalement dans l'accès à une amélioration tardive, plutôt que dans une progression continue.

#### 4.3.4.2 Instance *CircuitDebuggingProblems-rsdecoder-debug.dimacs.wcnf*.




(a) Trajectoires de coût.

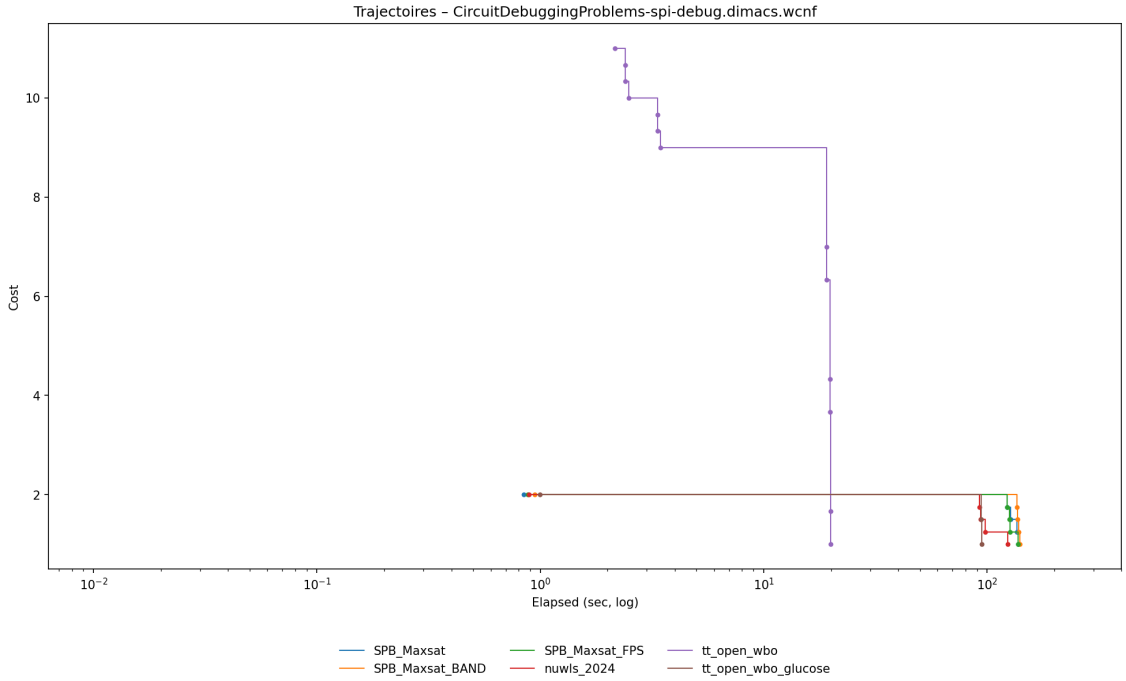



(b) Trajectoires de score relatif.

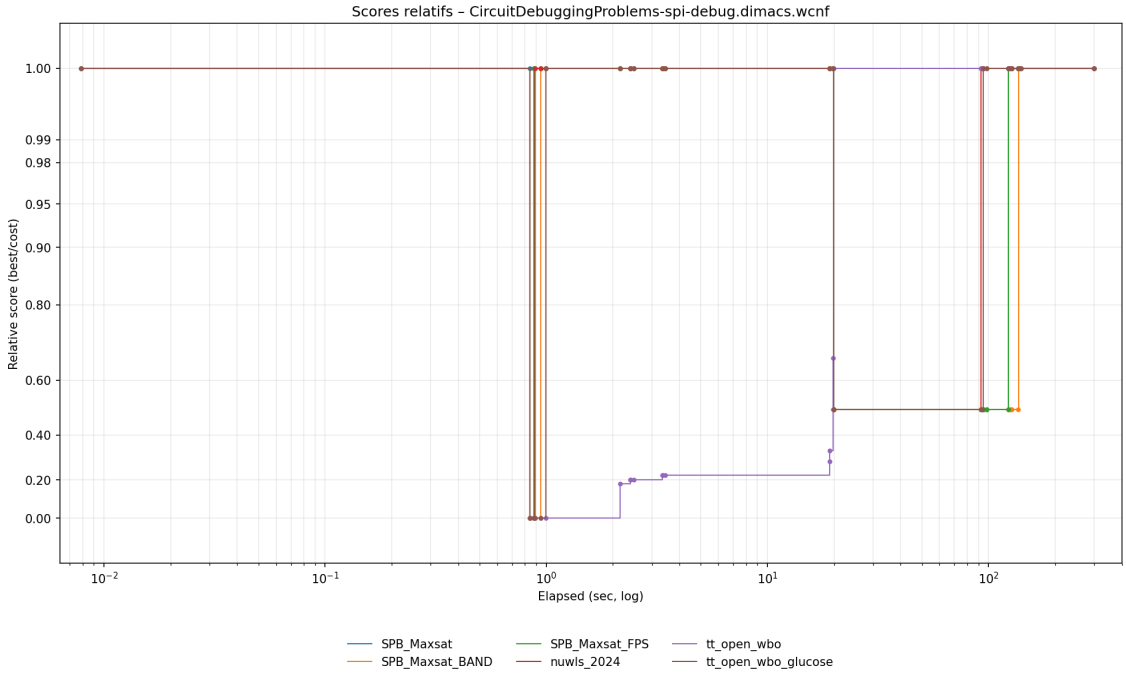
Figure 4.8 – Illustration quantitative sur l'instance `CircuitDebuggingProblems-rsdecoder-debug.dimacs.wcnf`.

Cette instance met en évidence un comportement très contrasté entre solveurs. Les solveurs de la famille SPB ainsi que *NuWLS-2024* atteignent rapidement une solution de coût minimal, ce qui se traduit par un score relatif égal à 1 dès les premières secondes. En revanche, la variante *tt_open_wbo* présente une trajectoire nettement différente : elle reste longtemps bloquée sur des solutions de coût élevé avant d'effectuer des améliorations progressives. Cette différence est particulièrement visible dans les scores relatifs, où l'écart entre solveurs reste maximal sur une large portion de l'axe temporel. Cette instance illustre clairement l'intérêt d'une analyse *anytime* : bien que certains solveurs puissent atteindre une solution optimale à terme, leur comportement intermédiaire peut être significativement moins performant.

#### 4.3.4.3 Instance *CircuitDebuggingProblems-spi-debug.dimacs.wcnf*.




(a) Trajectoires de coût.




(b) Trajectoires de score relatif.

Figure 4.9 – Illustration quantitative sur l'instance *CircuitDebuggingProblems-spi-debug.dimacs.wcnf*.

Sur cette instance, les différences de comportement sont encore plus marquées. Les solveurs SPB et *NuWLS-2024* atteignent rapidement des solutions proches de l'optimum, avec des trajectoires relativement stables. À l'inverse, la variante *tt_open_wbo* présente une progression tardive et irrégulière, caractérisée par des améliorations successives après un long plateau. Cette dynamique se traduit par des scores relatifs très faibles sur une grande partie de l'exécution, suivis d'une remontée progressive. Ce type de comportement met en évidence une faiblesse importante en contexte *anytime* : même si le solveur converge vers une bonne solution, il reste peu compétitif sur des horizons intermédiaires.

Dans l'ensemble, ces exemples confirment que les performances des solveurs ne peuvent être pleinement comprises à partir des seules moyennes. À l'échelle des instances, on observe à la fois des cas où les solveurs restent très proches, des cas où certains dominent très tôt, et des cas où un solveur reste longtemps bloqué avant de progresser. Cette diversité de comportements justifie précisément l'analyse des trajectoires *anytime*.

## 4.4 Résultats agrégés sur les instances *weighted*

Cette section reprend la même logique d'analyse sur les instances *weighted*. Comme nous allons le voir, la hiérarchie y est plus dépendante de l'horizon d'observation que dans le cas *unweighted*, ce qui renforce encore l'intérêt d'une lecture temporelle détaillée.

### 4.4.1 Visualisations agrégées des trajectoires

Nous présentons les trajectoires agrégées sur l'ensemble des instances *weighted*. Afin de capturer à la fois les comportements à très court horizon et les évolutions à moyen et long terme, nous retenons une représentation du score relatif moyen sur l'intervalle complet de 300 secondes, avec une abscisse en échelle logarithmique. À titre complémentaire, nous présentons également l'évolution du coût moyen observé au fil du temps. Cette seconde représentation fournit une lecture directe des valeurs brutes produites par les solveurs, tandis que le score relatif normalisé demeure la mesure principale pour leur comparaison entre plusieurs instances.

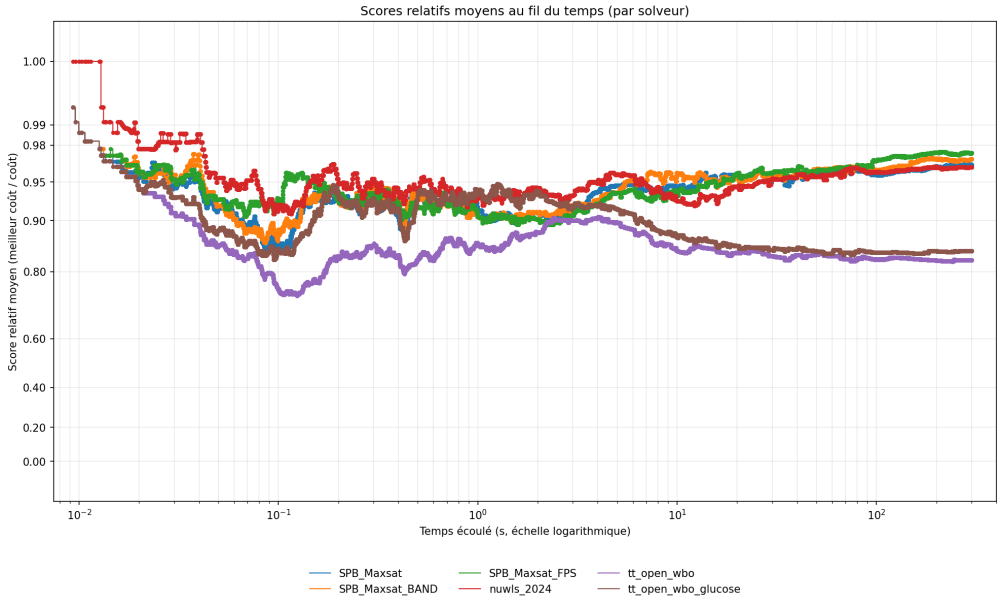



Figure 4.10 – Évolution du score relatif moyen sur les instances *weighted* sur l'horizon complet de 300 secondes, avec une abscisse logarithmique. Cette vue d'ensemble permet d'observer la dynamique globale des solveurs sur toute la durée d'exécution.

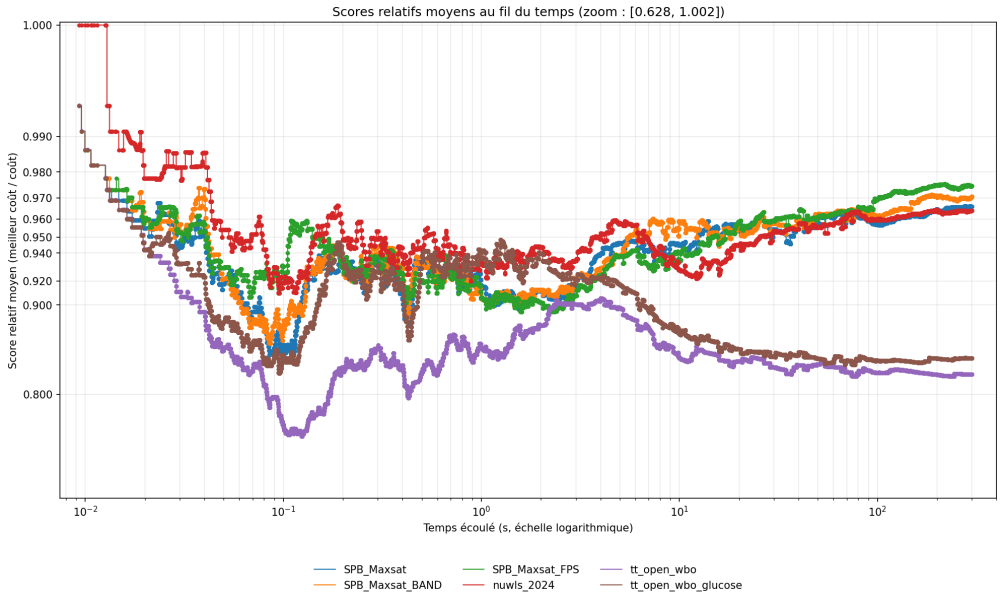



Figure 4.11 – Vue complémentaire de l'évolution du score relatif moyen sur les instances *weighted*, sur le même horizon de 300 secondes et avec la même abscisse logarithmique, mais avec un zoom sur la zone utile de l'axe des ordonnées afin de mieux faire apparaître les écarts fins entre solveurs.

La première observation importante est que les résultats *weighted* apparaissent moins homogènes que les résultats *unweighted*. La hiérarchie entre solveurs varie plus nettement selon l'horizon temporel, ce qui traduit des comportements plus contrastés entre les phases précoces et tardives de l'exécution.

Les courbes agrégées montrent en particulier que *NuWLS-2024* se révèle très compétitif pendant les premières phases de la recherche. Lorsque l'horizon temporel s'allonge, *SPB-MaxSAT-FPS* progresse plus favorablement et obtient finalement le score relatif moyen le plus élevé. *SPB-MaxSAT-BAND* et *SPB-MaxSAT* restent globalement proches du groupe de tête. En revanche, les deux variantes de *TT-Open-WBO-Inc* se détachent progressivement de ce groupe à partir des horizons intermédiaires. La variante utilisant *IntelSAT* demeure la plus en retrait, tandis que celle utilisant *Glucose* conserve des scores légèrement supérieurs.

La Figure 4.11 complète cette lecture en rendant plus visibles les écarts fins entre solveurs. Elle fait notamment apparaître la progression tardive de *SPB-MaxSAT-FPS*, ainsi que la dégradation relative des deux variantes de *TT-Open-WBO-Inc* lorsque la durée d'exécution augmente.

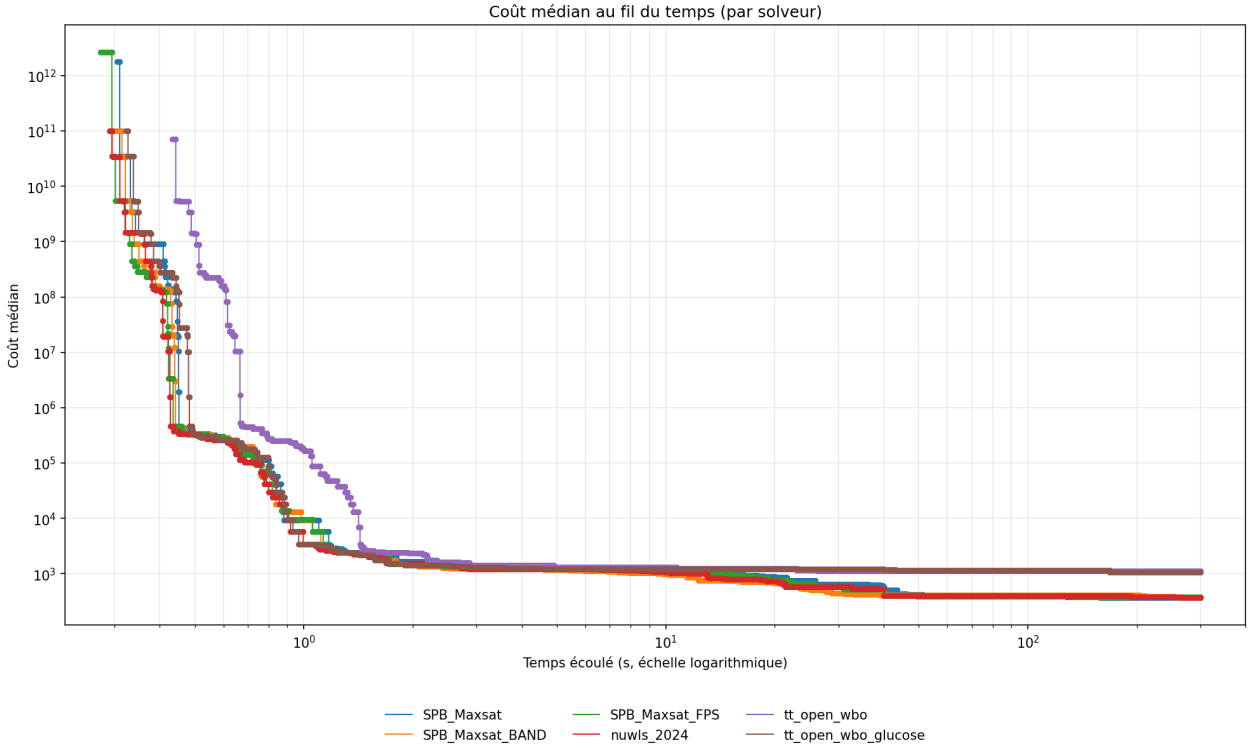



Figure 4.12 – Évolution du coût médian sur les instances *weighted* sur l'horizon complet de 300 secondes, avec des axes logarithmiques afin de représenter des coûts couvrant plusieurs ordres de grandeur.

La Figure 4.12 présente l'évolution du coût médian pour chaque solveur. Les instances sans solution sont temporairement associées à un coût infini, La médiane réduit ainsi l'influence des valeurs extrêmes, tandis que l'échelle logarithmique rend visibles des coûts variant de plusieurs ordres de grandeur.

on constate que les coûts diminuent fortement pendant les premières secondes, puis plus progressivement. Aux horizons longs, les variantes de *SPB-MaxSAT* et *NuWLS-2024* atteignent des coûts médians inférieurs à ceux des deux variantes de *TT-Open-WBO-Inc*. Ainsi cette figure fournit une lecture complémentaire des trajectoires, cependant les scores relatifs demeurent la mesure principale pour comparer les solveurs, puisque les coûts ne sont pas normalisés, ils sont brut et à échelles de grandeurs différentes.

### 4.4.2 Mise en perspective avec la MSE 2024

Les résultats précédents peuvent être comparés, de manière qualitative, aux résultats officiels de la MaxSAT Evaluation 2024 sur le track *weighted*. Dans la présentation officielle, *SPB-MaxSAT-c-FPS* arrive premier à 60 secondes comme à 300 secondes, devant *TT-Open-WBO-inc-glucose* puis *NuWLS-c-IBR* (Berg *et al.*, 2024a).

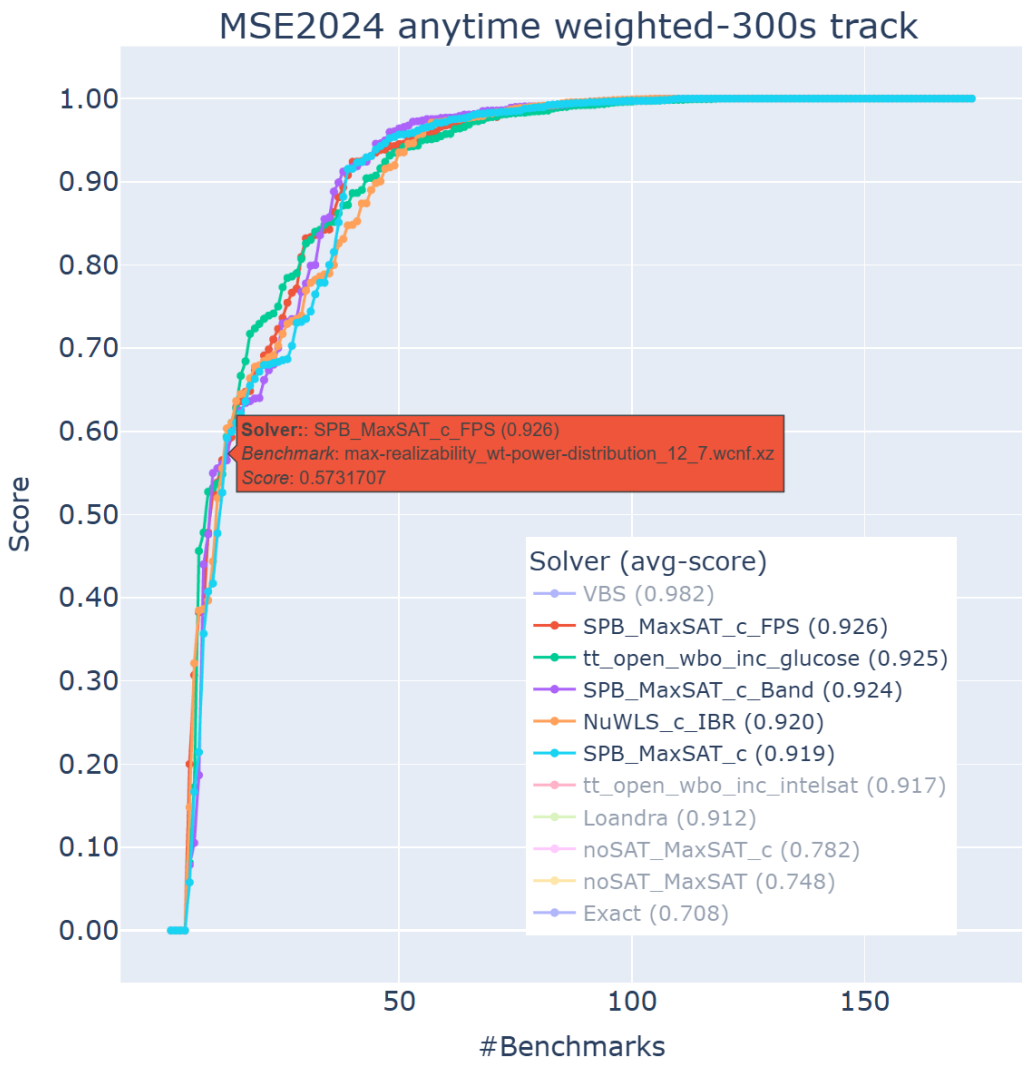



Figure 4.13 – Courbe officielle de la MSE 2024 pour le track *anytime weighted* à 300 secondes. Source : MaxSAT Evaluation 2024 (Berg *et al.*, 2024a).

La Figure 4.13 fournit une synthèse globale des résultats officiels, mais elle présente plusieurs limites lorsqu'il s'agit d'interpréter finement les comportements. Les courbes du groupe de tête sont fortement resserrées et largement superposées sur une grande partie de l'axe des abscisses. La lisibilité de la figure reste donc relativement faible pour les solveurs les plus compétitifs. La représentation remplit une fonction de synthèse, mais elle ne permet pas de distinguer clairement les écarts fins entre les solveurs.

Surtout, cette représentation repose sur une métrique agrégée à horizon fixe. Elle ne renseigne que sur l'état des performances à 300 secondes et ne montre pas comment ce résultat final a été atteint. Or, dans le

cas *weighted*, cette dimension temporelle est particulièrement importante : certains solveurs apparaissent plus forts à court horizon, tandis que d'autres prennent l'avantage plus tard. Ce type de bascule temporelle n'apparaît pas dans la courbe officielle.

À l'inverse, les représentations produites par *MaxSAT Runner* rendent visibles les trajectoires moyennes, les écarts entre horizons précoces et tardifs, ainsi que les changements de domination entre solveurs. Elles montrent donc non seulement quels solveurs sont compétitifs, mais aussi à quel moment et selon quelle dynamique cette compétitivité s'exprime.

### 4.4.3 Lecture complémentaire par indicateurs synthétiques

Les métriques de domination temporelle calculées sur les instances *weighted* sont résumées dans le Tableau 4.3.

Table 4.3 – Métriques de domination temporelle sur les instances *weighted*.

| Solveur | AUC | SDT | AUC-log | SDT-log |
|---|---|---|---|---|
| SPB-MaxSAT-FPS | **291.31890** | **0.97111** | 9.34164 | 0.93168 |
| SPB-MaxSAT-BAND | 290.13943 | 0.96717 | 9.48991 | 0.94369 |
| SPB-MaxSAT | 288.69942 | 0.96237 | 9.44713 | 0.94441 |
| NuWLS-2024 | 288.10489 | 0.96038 | **9.97404** | **0.96102** |
| TT-Open-WBO-glucose | 254.08435 | 0.84701 | 8.56236 | 0.89467 |
| TT-Open-WBO | 249.78169 | 0.83270 | 8.08161 | 0.88564 |

Ces résultats montrent que la hiérarchie dépend ici fortement de la métrique retenue. Si l'on considère les métriques intégrées sur tout l'horizon en échelle linéaire, *SPB-MaxSAT-FPS* arrive nettement en tête. Il domine à la fois l'AUC et le SDT, ce qui indique qu'il s'agit du solveur le plus fort lorsque l'on considère la trajectoire complète jusqu'à 300 secondes.

En revanche, si l'on privilégie davantage les débuts d'exécution à travers les métriques logarithmiques, *NuWLS-2024* devient le solveur dominant. Il obtient la meilleure AUC-log et le meilleur SDT-log. Cette lecture confirme l'idée que *NuWLS-2024* domine plus nettement les horizons précoces, alors que *SPB-MaxSAT-FPS* s'impose davantage sur les horizons longs.

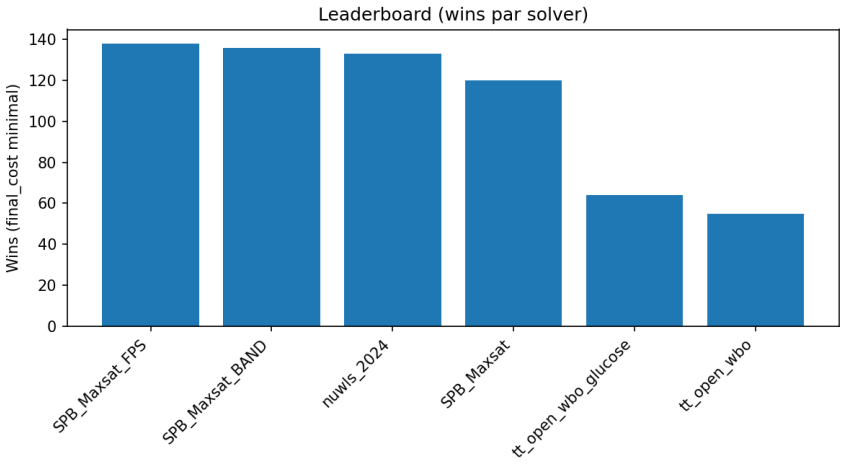



Figure 4.14 – Nombre de *victoires* par solveur sur les instances *weighted*.

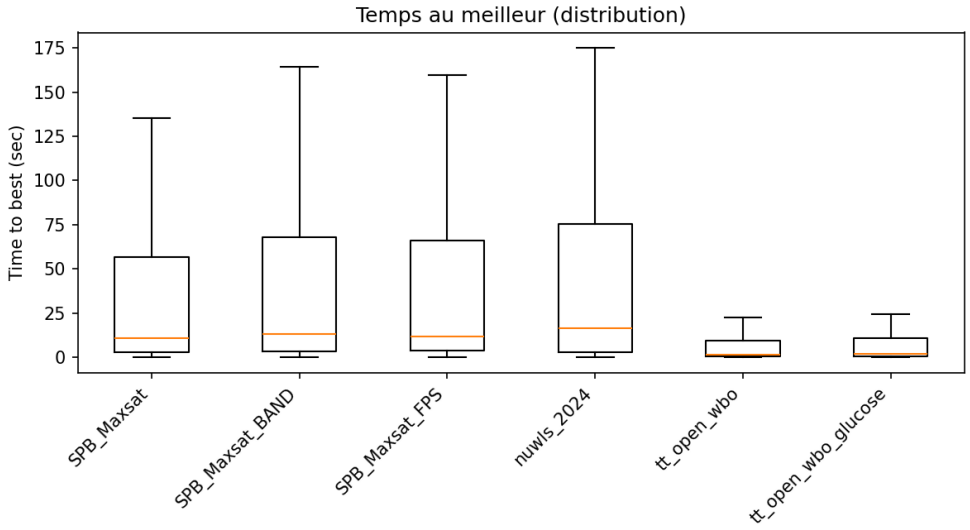



Figure 4.15 – Distribution du temps nécessaire pour atteindre le meilleur coût sur les instances *weighted*.

Le classement par *victoires* va dans le même sens que cette lecture longue durée. *SPB-MaxSAT-FPS* arrive en tête, devant *SPB-MaxSAT-BAND* puis *NuWLS-2024*. Inversement, les deux variantes de *TT-Open-WBO-Inc* restent nettement en retrait. Les distributions du temps au meilleur coût apportent un éclairage complémentaire : les variantes *TT-Open-WBO-Inc* atteignent souvent leur meilleure solution plus rapidement, mais cette rapidité initiale ne se traduit pas par une domination globale. Elles trouvent vite une solution acceptable, mais améliorent ensuite moins durablement leur coût que les solveurs du groupe de tête. À l'inverse, *NuWLS-2024* et les variantes SPB continuent à progresser plus longtemps, ce qui semble déterminant pour

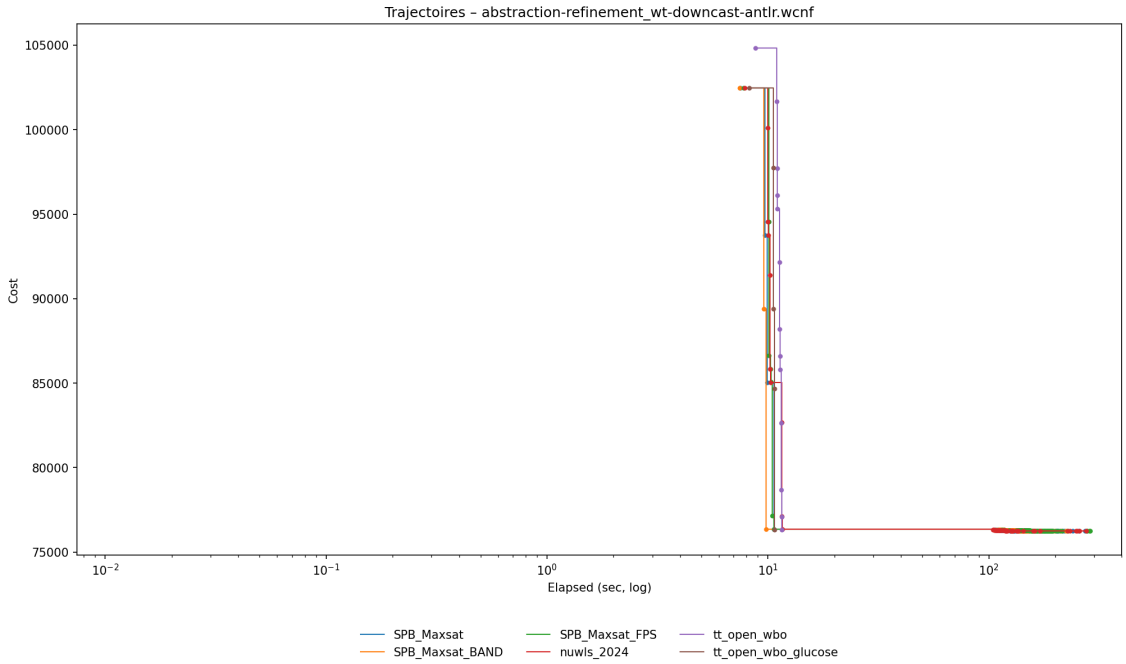
la qualité finale.

Dans l'ensemble, les résultats *weighted* peuvent être résumés de la manière suivante : *NuWLS-2024* domine les horizons précoces, tandis que *SPB-MaxSAT-FPS* s'impose plus clairement dès que l'on considère la trajectoire complète et les performances finales. Cette hiérarchie évolutive serait difficile à mettre en évidence à partir d'une seule mesure terminale.

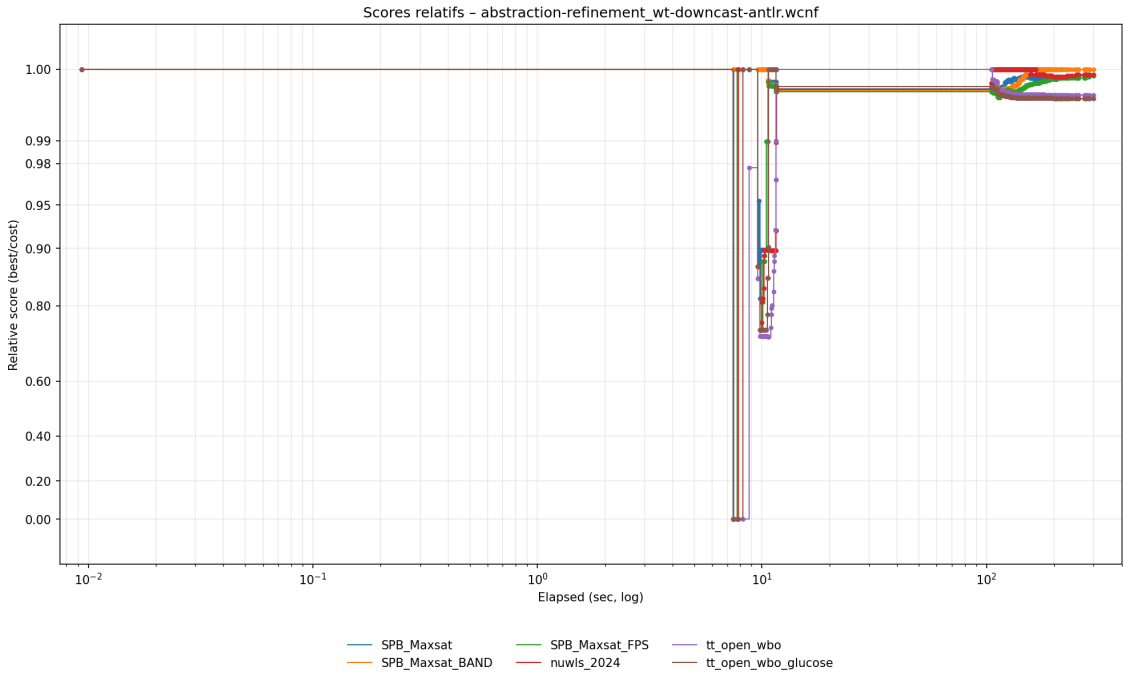
### 4.4.4 Illustration quantitative sur quelques instances

Nous complétons l'analyse par une étude à l'échelle de quelques instances issues du track *anytime weighted*. Contrairement au cas non pondéré, ces instances mettent davantage en évidence des phénomènes de hiérarchie temporelle entre solveurs, avec des inversions de dominance selon l'horizon considéré.

#### 4.4.4.1 Instance *abstraction-refinement_wt-downcast-antlr.wcnf*.




(a) Trajectoires de coût.

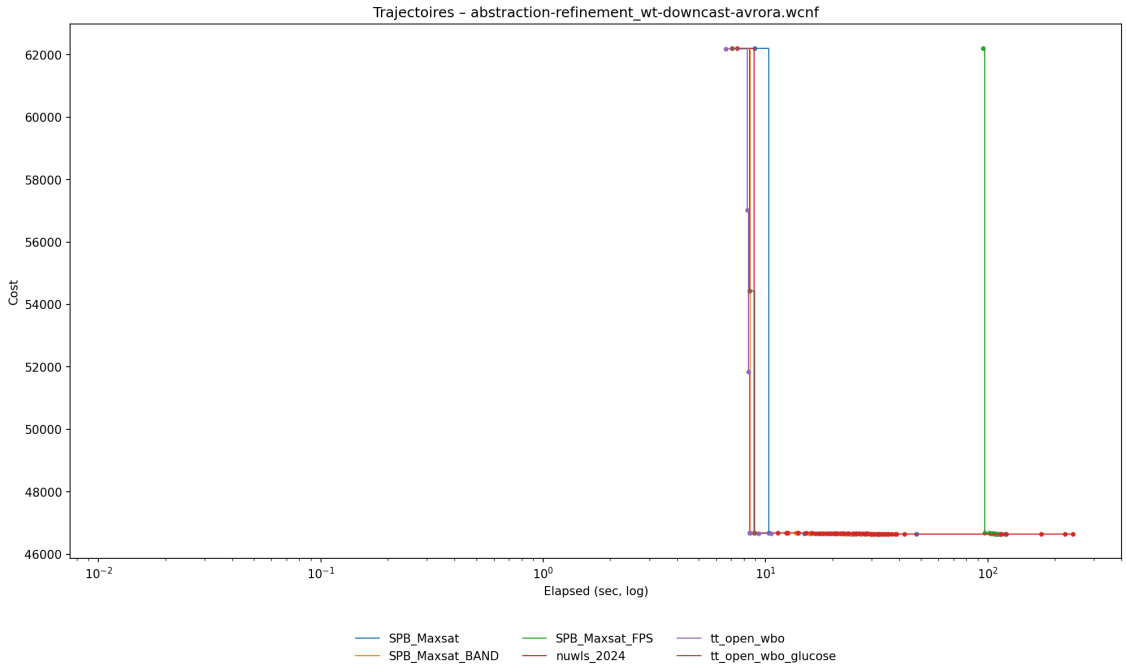



(b) Trajectoires de score relatif.

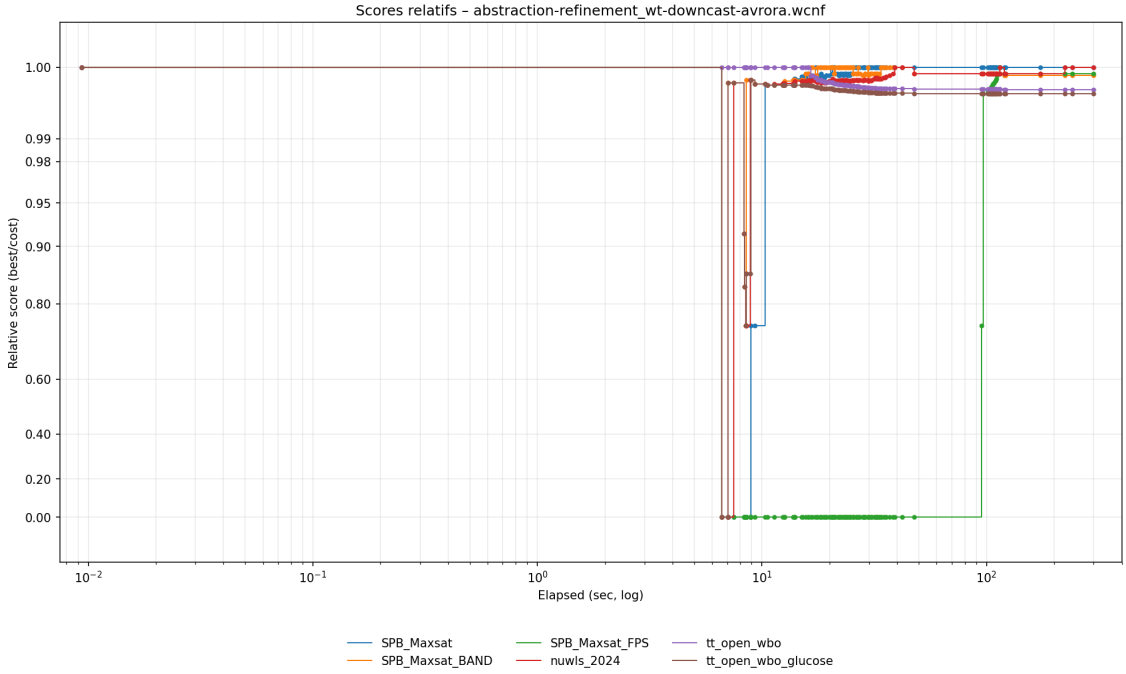
Figure 4.16 – Illustration quantitative sur l'instance *abstraction-refinement_wt-downcast-antlr.wcnf*.

Sur cette instance, l'ensemble des solveurs converge vers une solution de coût similaire, mais selon des dynamiques distinctes. Les trajectoires montrent une phase initiale caractérisée par des améliorations rapides, suivie d'un plateau relativement long. Du point de vue des scores relatifs, tous les solveurs atteignent rapidement des valeurs proches de 1, ce qui indique une convergence vers des solutions quasi optimales. Néanmoins, des écarts subtils apparaissent sur la phase intermédiaire : certains solveurs atteignent plus tôt le meilleur coût, tandis que d'autres présentent une progression plus graduelle. Cette instance illustre un cas où les différences entre solveurs sont faibles en valeur absolue, mais où l'analyse *anytime* permet de distinguer des comportements légèrement plus efficaces à court terme.

#### 4.4.4.2 Instance *abstraction-refinement_wt-downcast-avrora.wcnf*.




(a) Trajectoires de coût.

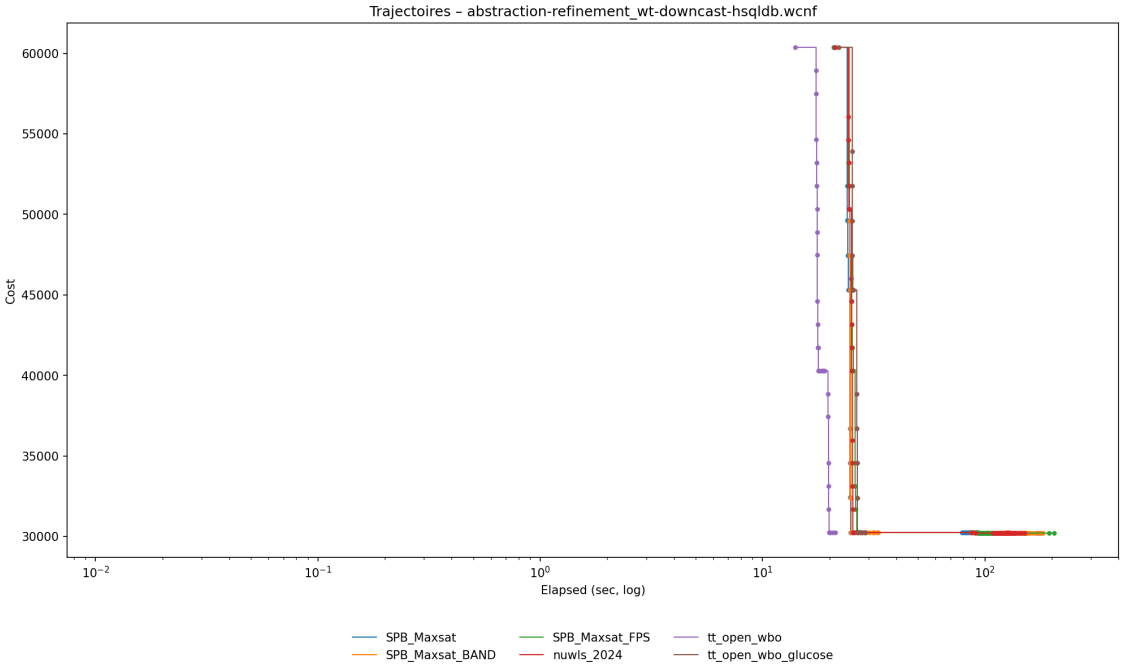



(b) Trajectoires de score relatif.

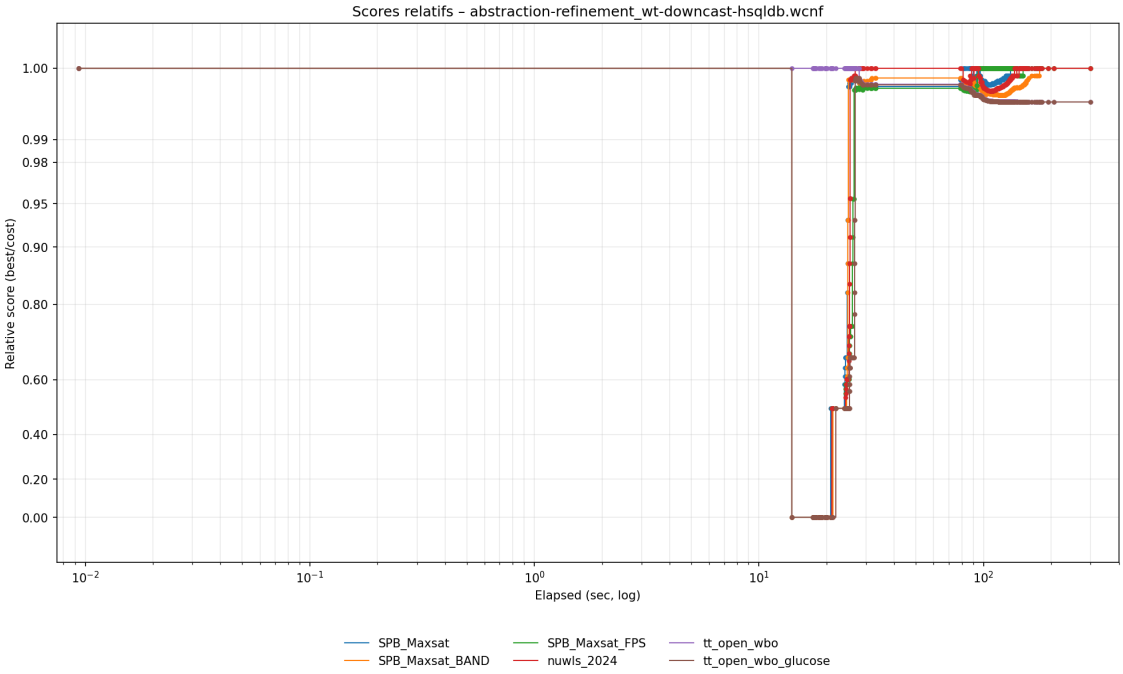
Figure 4.17 – Illustration quantitative sur l'instance *abstraction-refinement_wt-downcast-avrora.wcnf*.

Cette instance met en évidence un phénomène plus marqué de dépendance à l'horizon temporel. Certains solveurs atteignent très rapidement un score relatif maximal, ce qui traduit une excellente performance à court terme. Cependant, d'autres solveurs rattrapent progressivement cet avantage et atteignent à leur tour des scores proches de 1. La dynamique observée correspond donc à une inversion de dominance : un solveur initialement dominant perd son avantage à mesure que le temps augmente. Cette figure illustre de manière particulièrement claire qu'une hiérarchie observée à court horizon n'est pas nécessairement stable.

#### 4.4.4.3 Instance *abstraction-refinement_wt-downcast-hsqldb.wcnf*.




(a) Trajectoires de coût.




(b) Trajectoires de score relatif.

Figure 4.18 – Illustration quantitative sur l'instance *abstraction-refinement_wt-downcast-hsqldb.wcnf*.

Sur cette instance, les différences entre solveurs sont particulièrement visibles. Un solveur se distingue par une amélioration très précoce, atteignant rapidement des scores élevés, tandis que les autres progressent de manière plus tardive. Les courbes de scores relatifs mettent en évidence une transition abrupte entre une phase de stagnation et une phase de convergence rapide vers des valeurs élevées. Contrairement à l'instance précédente, certains solveurs ne parviennent toutefois pas à combler complètement l'écart, ce qui maintient une hiérarchie persistante. Cette instance illustre ainsi un cas où l'avantage initial d'un solveur se traduit plus directement par une domination globale.

Dans l'ensemble, l'analyse de ces instances pondérées met en évidence plusieurs phénomènes caractéristiques : la présence d'inversions de dominance, une convergence parfois rapide vers des solutions proches de l'optimum, mais aussi des écarts significatifs dans la vitesse d'accès à ces solutions. Ces observations confirment que, dans le cadre pondéré, l'évaluation des solveurs ne peut se limiter à une mesure finale et nécessite une analyse temporelle fine.

## 4.5 Discussion méthodologique

L'ensemble des résultats présentés dans ce chapitre met en évidence l'intérêt d'une approche centrée sur l'analyse des trajectoires *anytime*. Cette section propose une mise en perspective des choix méthodologiques adoptés, ainsi qu'une discussion de leurs apports et de leurs limites.

### 4.5.1 Apports de l'analyse des trajectoires

Contrairement aux approches classiques fondées sur des mesures finales à temps fixe, l'observation continue des trajectoires permet de capturer la dynamique complète du comportement des solveurs. Les résultats obtenus montrent notamment que la hiérarchie entre solveurs n'est pas stable dans le temps : un solveur performant à court horizon peut être dépassé à plus long terme, et inversement.

Cette variabilité temporelle est particulièrement visible sur les instances pondérées, où l'on observe fréquemment des inversions de dominance. Une évaluation limitée à un unique point temporel masque donc une partie importante de l'information. L'analyse des trajectoires permet ainsi de mieux comprendre non seulement quel solveur est performant, mais également *à quel moment* et *selon quelle dynamique*.

### 4.5.2 Intérêt de la normalisation par score relatif

L'introduction d'un score relatif normalisé permet de comparer les solveurs sur une base commune, indépendamment de l'échelle des coûts propres à chaque instance. Cette normalisation facilite l'agrégation des résultats et rend possible une analyse comparative cohérente.

Par ailleurs, l'évolution temporelle de ces scores met en évidence des comportements fins, tels que des phases de stagnation suivies d'améliorations rapides, ou encore des écarts persistants entre solveurs. Ces éléments sont difficilement observables à partir des seuls coûts absolus.

Cependant, cette métrique repose sur les valeurs extrêmes, c'est-à-dire le meilleur et le pire coût observés à chaque instant. Elle peut donc se montrer sensible à certains comportements atypiques ou à certaines couvertures partielles aux très petits horizons. Une interprétation prudente reste donc nécessaire, en particulier dans la toute première phase de l'exécution.

### 4.5.3 Comparaison avec la métrique de la MaxSAT Evaluation

La MaxSAT Evaluation propose une métrique agrégée basée sur des points d'évaluation discrets. Si cette approche permet d'établir un classement global, elle présente certaines limites du point de vue de l'analyse *anytime*. La représentation graphique officielle tend à produire des courbes fortement superposées, rendant la lecture difficile et limitant l'interprétation qualitative. Dans les tracks *unweighted* et *weighted* à 300 secondes, les différences entre solveurs de tête apparaissent peu marquées visuellement, alors même que nos analyses de trajectoires révèlent des écarts temporels significatifs.

À l'inverse, l'approche proposée dans ce mémoire permet de mieux distinguer les profils de progression des solveurs, en mettant en évidence les différences de réactivité initiale, de persistance dans l'amélioration et de domination à long horizon. Elle ne remplace pas la métrique officielle, mais la complète en apportant un niveau d'analyse plus fin.

### 4.5.4 Limites de l'approche proposée

Malgré ses apports, la méthodologie présente plusieurs limites. Tout d'abord, l'analyse des trajectoires génère un volume important de données, ce qui peut compliquer la visualisation et nécessiter des choix de

représentation adaptés. Ensuite, les résultats dépendent du protocole expérimental, en particulier du choix des instances, du temps limite et du nombre d'exécutions. Une généralisation des conclusions suppose donc le maintien d'un protocole homogène et reproductible.

Enfin, certaines métriques agrégées, comme l'aire sous la courbe ou les variantes logarithmiques, impliquent des choix de pondération temporelle qui influencent directement l'interprétation. Les métriques linéaires mettent davantage en valeur la domination globale sur l'ensemble de l'horizon, alors que les métriques logarithmiques privilégient les premiers instants de la recherche. Cette différence n'est pas un défaut en soi, mais elle doit être explicitée, car elle modifie la hiérarchie observée.

### 4.5.5 Positionnement de la contribution

Dans ce contexte, la contribution de ce travail ne réside pas dans la définition d'une nouvelle métrique unique, mais dans la proposition d'un cadre méthodologique permettant d'explorer le comportement *anytime* des solveurs de manière plus complète. L'outil *MaxSAT Runner* joue un rôle central dans ce cadre, en fournissant les mécanismes nécessaires à la collecte, à la reconstruction et à l'analyse des trajectoires. Il permet ainsi de passer d'une évaluation statique à une analyse dynamique, mieux adaptée à la nature des solveurs *anytime*.

Ce changement de perspective permet de mieux comprendre les compromis entre performance à court terme, régularité de la trajectoire et qualité finale des solutions. Il explique aussi pourquoi certaines hiérarchies observées dans la MSE 2024 peuvent être confirmées tout en étant réinterprétées à la lumière des trajectoires complètes.

## 4.6 Résumé du chapitre

Ce chapitre présente les résultats expérimentaux obtenus à l'aide de *MaxSAT Runner* sur les instances des tracks *anytime* de la MaxSAT Evaluation 2024, au terme d'une campagne exécutée sur le superordinateur *Rorqual*. L'analyse est conduite séparément sur les instances *unweighted* et *weighted*, en donnant la priorité aux visualisations agrégées des trajectoires, puis en les complétant par des indicateurs synthétiques et par quelques études de cas sur des instances représentatives.

Les résultats *unweighted* montrent une hiérarchie relativement serrée. *NuWLS-2024* apparaît comme le sol-

veur le plus constant au sens de la domination temporelle agrégée, tandis que *SPB-MaxSAT-BAND* domine davantage la lecture terminale. Les résultats *weighted* révèlent une hiérarchie plus dépendante de l'horizon : *NuWLS-2024* domine les débuts de trajectoire, alors que *SPB-MaxSAT-FPS* s'impose plus nettement lorsque l'on considère la trajectoire complète et les performances finales.

Cette organisation montre qu'une évaluation de solveurs *anytime* ne se réduit pas à un classement final, mais gagne à être lue comme une analyse de trajectoires, de hiérarchies temporelles et de profils de comportement. Le chapitre suivant s'appuie sur ce cadre pour étudier plus spécifiquement la deuxième contribution du mémoire, le solveur *EvalMaxSat Anytime*.

# CHAPITRE V

# EVALMAXSAT ANYTIME : STRUCTURES, MÉCANISMES ET ÉVALUATION

## 5.1 Introduction

Ce chapitre présente **EvalMaxSat Anytime**, le solveur MaxSAT *anytime* développé dans le cadre de ce mémoire. L'objectif poursuivi est de disposer d'une base de résolution modulaire, maintenable et algorithmiquement efficace, permettant d'étudier de manière contrôlée l'effet de différents mécanismes internes de recherche locale.

Le solveur étudié s'inscrit dans le cadre du *Weighted Partial MaxSAT*. Il repose sur une recherche locale opérant sur des affectations complètes, initialisées à partir d'une solution satisfaisant les clauses dures, obtenue avec l'aide d'un solveur SAT, puis progressivement améliorées par des *flips* de variables booléennes. Dans ce contexte, la performance du solveur dépend de sa capacité à maintenir, de façon incrémentale, les informations nécessaires à l'évaluation rapide des *flips*. Cette exigence conduit à introduire plusieurs structures dédiées au stockage de la formule, au suivi de l'état courant et au calcul des scores locaux.

Le chapitre est organisé en cinq parties. La première présente le noyau de recherche locale ainsi que les structures de données utilisées pour représenter la formule, l'affectation courante et les scores associés aux variables. La deuxième met en évidence les limites de ce noyau en présence de minima locaux, puis introduit les mécanismes proposés pour en sortir. La troisième définit les différentes variantes du solveur et le protocole expérimental utilisé pour les comparer de manière systématique. La quatrième présente les résultats obtenus et analyse l'effet des différents mécanismes sur la qualité des solutions et sur la dynamique *anytime*. Enfin, la dernière partie synthétise les principaux enseignements du chapitre, discute des limites et dégage la portée méthodologique.

## 5.2 Noyau de recherche locale et structures de données

Cette section présente le cœur du solveur, c'est-à-dire le mécanisme de recherche locale utilisé pour faire évoluer une affectation complète, ainsi que les structures de données qui permettent d'effectuer les mouvements de manière efficace. Cette organisation conditionne directement le coût des opérations principales et la capacité du solveur à servir de base expérimentale pour l'étude de variantes de recherche locale.

### 5.2.1 Principe général du noyau de recherche locale

On considère une instance de *Weighted Partial MaxSAT* définie par un ensemble de variables booléennes $X = \{x_1, \ldots, x_n\}$, un ensemble de clauses dures $\mathcal{C}_{hard}$, un ensemble de clauses souples $\mathcal{C}_{soft}$, et une fonction de poids

$$w : \mathcal{C}_{soft} \to \mathbb{R}_{>0}.$$

Une affectation complète est une fonction

$$\sigma : X \to \{0, 1\}.$$

L'objectif est de construire une affectation qui satisfasse toutes les clauses dures et minimise le poids total des clauses souples non satisfaites. Le coût d'une affectation $\sigma$ est donc défini par

$$\mathrm{cost}(\sigma) = \sum_{C \in \mathcal{C}_{soft},\, \sigma \not\models C} w(C),$$

sous la contrainte

$$\forall C \in \mathcal{C}_{hard}, \quad \sigma \models C.$$

La recherche locale manipule une affectation complète courante $\sigma$. À chaque étape, elle choisit une variable $x \in X$ et applique un *flip*, c'est-à-dire une inversion de la valeur de $x$. On note $\sigma^x$ l'affectation obtenue après ce *flip*. La variation de coût induite par ce mouvement peut alors être définie par

$$\Delta(x, \sigma) = \mathrm{cost}(\sigma) - \mathrm{cost}(\sigma^x).$$

Dans ce cadre, un *flip* est dit *favorable* s'il préserve la satisfaction des clauses dures et améliore strictement le coût, c'est-à-dire si

$$\forall C \in \mathcal{C}_{hard},\ \sigma^x \models C \qquad \text{et} \qquad \Delta(x, \sigma) > 0.$$

Autrement dit, il correspond à un mouvement qui réduit immédiatement le poids total des clauses souples non satisfaites sans quitter la région faisable.

Le noyau de recherche locale repose alors sur un schéma simple. La recherche commence à partir d'une affectation initiale satisfaisant les clauses dures, obtenue à l'aide d'un solveur SAT appliqué à la partie dure de la formule. À partir de cette affectation, le solveur applique successivement des *flips favorables* afin de

diminuer le coût de la solution courante, tout en mémorisant la meilleure affectation rencontrée depuis le début de l'exécution.

Ce fonctionnement général peut être résumé par l'algorithme 1.

**Algorithm 1** Noyau général de recherche locale

**Entrée :** une instance WPMaxSAT
**Sortie :** la meilleure affectation trouvée
Construire une affectation initiale $\sigma$ satisfaisant les clauses dures
Initialiser les structures représentant l'état courant
Mémoriser $\sigma^\star \leftarrow \sigma$
**while** le critère d'arrêt n'est pas atteint **do**
    Déterminer s'il existe une variable $x$ dont le retournement améliore l'affectation courante $\sigma$
    **if** un tel retournement existe **then**
        Appliquer le retournement choisi et mettre à jour $\sigma$
        Mettre à jour le coût courant et les structures auxiliaires
        **if** $\mathrm{cost}(\sigma) < \mathrm{cost}(\sigma^\star)$ **then**
            Mémoriser la nouvelle meilleure affectation : $\sigma^\star \leftarrow \sigma$
        **end if**
    **else**
        Signaler un blocage local
    **end if**
**end while**
**return** $\sigma^\star$

Ainsi, l'idée centrale est de ne pas réévaluer la formule dans son ensemble après chaque *flip*, mais de maintenir de manière incrémentale un ensemble d'informations suffisant pour guider la recherche. La section suivante s'intéresse précisément aux structures de données qui permettent de réaliser cette détection et ces mises à jour de manière efficace.

### 5.2.2 Représentation compacte de la formule

La formule est représentée de manière à permettre à la fois un accès direct aux clauses et un accès rapide aux clauses incidentes à un littéral donné. Deux besoins doivent en effet être satisfaits simultanément. D'une part, il faut pouvoir parcourir le contenu d'une clause pour évaluer son état ou mettre à jour les scores des variables qu'elle contient. D'autre part, lorsqu'on *flip* une variable, il faut identifier immédiatement les clauses affectées par ce mouvement, sans re-scanner toute la formule.

Pour répondre à cette exigence, les clauses sont stockées dans une structure compacte fondée sur un tableau contigu, accompagné d'indices de position permettant de retrouver rapidement les segments correspondant à chaque clause.

La figure 5.1 illustre le principe de stockage compact retenu pour les clauses. Cette organisation permet un accès direct à chaque clause tout en conservant une représentation contiguë bien adaptée à l'implantation en C++.

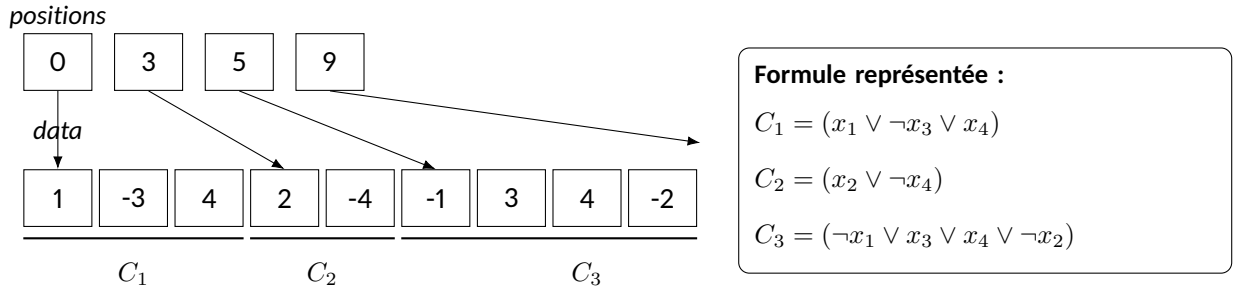



Figure 5.1 – Représentation compacte des clauses dans un tableau contigu, accompagnée d'un tableau de positions permettant d'identifier les segments correspondant à chaque clause.

Cette organisation limite les surcoûts de représentation et favorise une bonne localité mémoire. En parallèle, le solveur maintient des index inverses associant à chaque littéral signé la liste des clauses dans lesquelles il apparaît. Dans l'implémentation, ces index sont distingués selon que les clauses sont dures ou souples, ce qui permet de traiter séparément les deux types de contraintes lors des mises à jour.

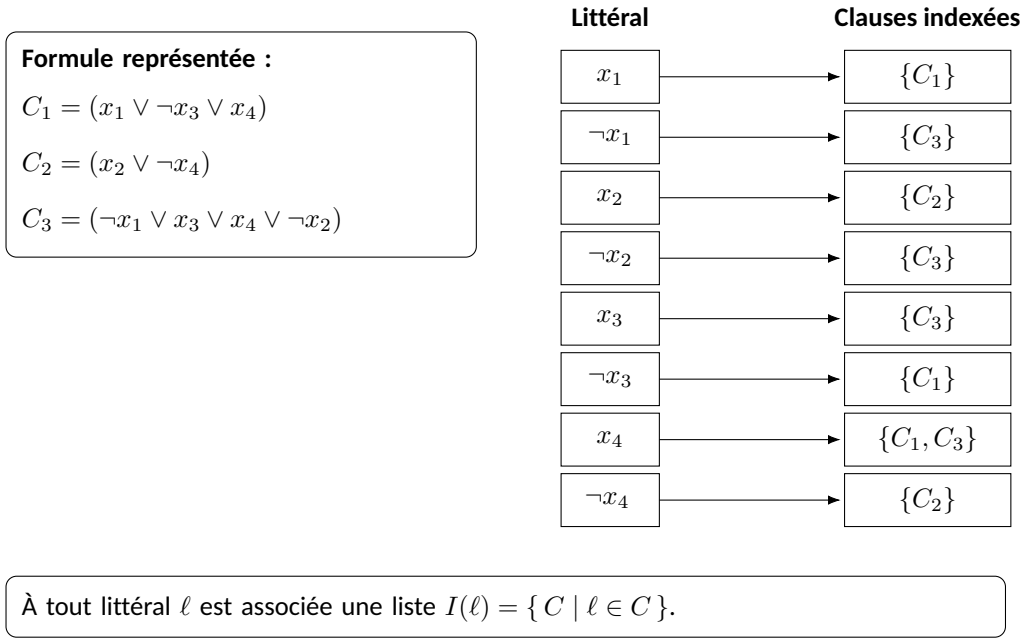



Figure 5.2 – Exemple d'index inverses par littéral signé pour la formule utilisée dans la figure précédente. À chaque littéral $\ell$ est associée la liste $I(\ell)$ des clauses dans lesquelles il apparaît.

Si l'on note $n$ le nombre de variables, $m$ le nombre de clauses et $L$ le nombre total d'occurrences de littéraux dans la formule, alors le coût mémoire de cette représentation est linéaire en la taille de l'instance, soit $\mathcal{O}(L + m + n)$. Dans le cas usuel où $n$ désigne le nombre de variables effectivement présentes dans la formule, on a $m \leq L$ et $n \leq L$, de sorte que cette borne se simplifie en $\mathcal{O}(L)$.

L'ajout d'une clause de taille $k$ dans cette structure est linéaire en $k$, puisque chaque littéral doit être inséré une fois dans la clause elle-même et une fois dans l'index inverse correspondant.

### 5.2.3 État courant de l'affectation

Au-delà du stockage statique de la formule, le solveur maintient un état courant décrivant l'affectation active ainsi qu'un ensemble d'informations dérivées mises à jour de façon incrémentale. Cette représentation dynamique comprend notamment l'affectation courante des variables, le coût de la solution, l'ensemble des clauses dures non satisfaites, l'ensemble des clauses souples non satisfaites, ainsi que, pour chaque clause, le nombre de littéraux qui la satisfont actuellement.

La figure 5.3 présente ces différentes composantes sur le même exemple que dans la section précédente, sous la forme d'un diagramme de type UML simplifié. Elle met en évidence les principales classes d'informations maintenues pendant la recherche locale.

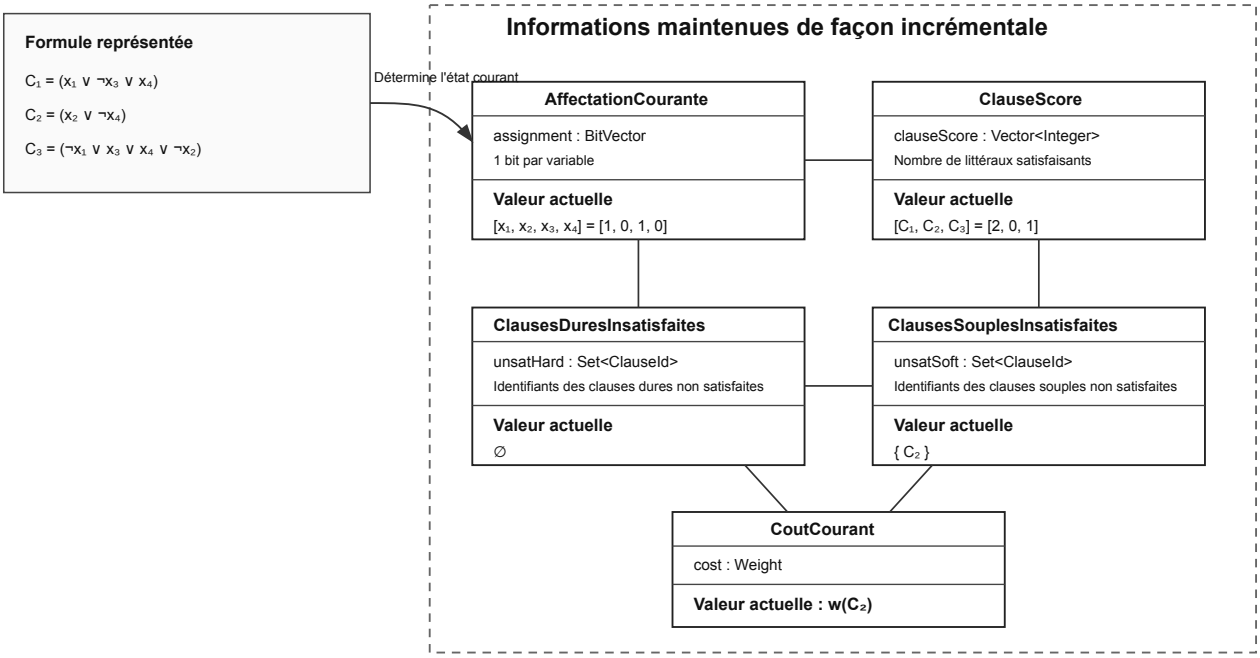



Figure 5.3 – Vue de type UML des informations maintenues de façon incrémentale pendant la recherche locale, sur le même exemple que dans la section précédente.

Cette dernière information joue un rôle central. En maintenant explicitement, pour chaque clause, le nombre de littéraux satisfaisants, il devient possible de mettre à jour l'état de la formule après un flip sans reparcourir toutes les clauses. Lorsqu'une variable change de valeur, seules les clauses dans lesquelles elle apparaît peuvent voir leur statut modifié. Le solveur n'a donc besoin de revisiter que ce voisinage local.

L'affectation courante est stockée sous la forme d'un *bitvector*, contenant un bit par variable.

L'espace mémoire supplémentaire requis pour cet état incrémental reste linéaire en le nombre de variables et de clauses. Il constitue un surcoût faible au regard du gain obtenu sur le temps de mise à jour.

### 5.2.4 Mise à jour incrémentale après un flip

Lorsqu'un flip est appliqué à une variable $x$, le solveur met à jour l'état courant en ne parcourant que les clauses contenant $x$ ou $\neg x$.

La figure 5.4 illustre, sur le même exemple que dans les sections précédentes, le caractère local de cette mise à jour. On considère ici un flip de $x_1$. Seules les clauses $C_1$ et $C_3$, qui contiennent respectivement $x_1$ et $\neg x_1$, sont susceptibles d'être affectées, tandis que $C_2$ reste inchangée.

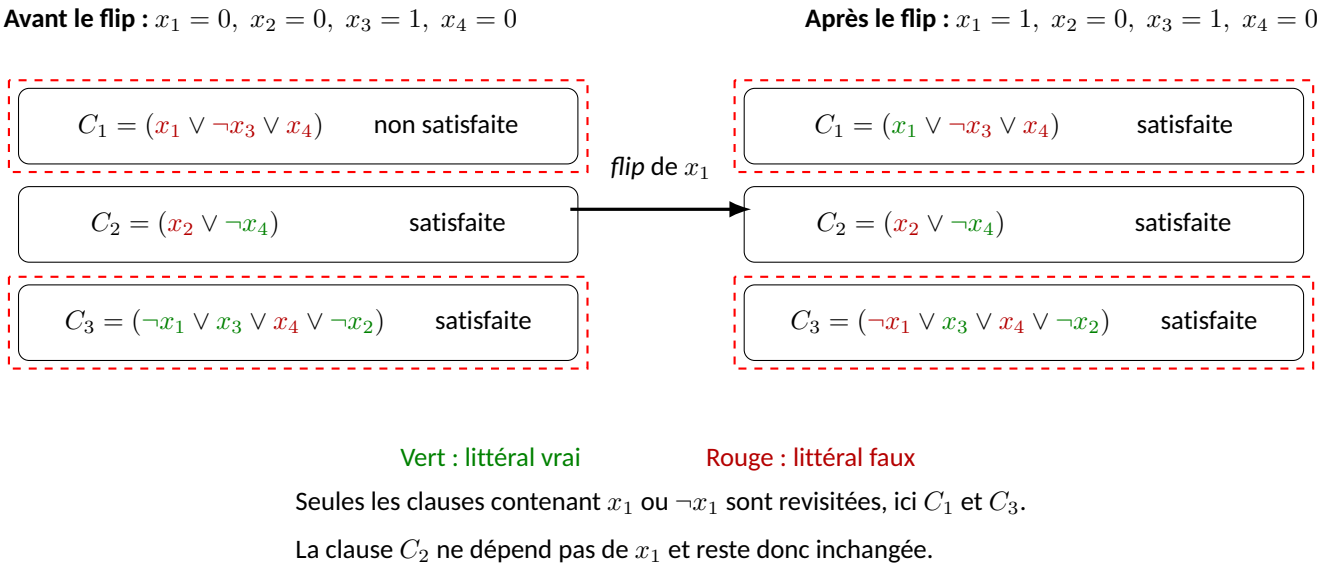



Figure 5.4 – Effet local d'un *flip* sur les clauses incidentes à une variable, en reprenant la formule utilisée dans les sections précédentes. Les littéraux vrais sont affichés en vert et les littéraux faux en rouge.

Pour chacune de ces clauses, le solveur met à jour le nombre de littéraux satisfaisants, ajuste éventuellement leur appartenance aux ensembles de clauses insatisfaites, puis corrige le coût courant si leur statut change. Cette propriété permet d'obtenir une première borne importante : la mise à jour de l'état logique de l'affectation est linéaire en le nombre total d'occurrences de la variable retournée. Si l'on note $\deg(x)$ ce nombre d'occurrences, alors

$$T_{\text{maj-état}}(x) = \mathcal{O}(\deg(x)).$$

Dans le cas où le degré des variables est supposé borné par une constante, cette borne se simplifie en

$$T_{\text{maj-état}}(x) = \mathcal{O}(1).$$

Autrement dit, sous cette hypothèse, le coût de mise à jour de l'état logique après un *flip* reste constant. Plus généralement, cette borne exprime le caractère local de la recherche : le coût d'un mouvement ne dépend pas de la taille totale de la formule, mais seulement du voisinage de la variable modifiée.

Cette mise à jour incrémentale constitue le socle du noyau de recherche locale. Elle permet d'envisager un grand nombre d'itérations à coût réduit, à condition de disposer également d'un mécanisme efficace pour identifier les flips favorables.

### 5.2.5 Scores variables et sélection des mouvements

Afin de guider la recherche, le solveur associe à chaque variable $x$ un score local $s(x)$ qui mesure l'effet immédiat du *flip* de $x$ sur l'état courant de la formule. Ce score est calculé à partir des informations déjà maintenues sur les clauses, en particulier du nombre de littéraux qui satisfont actuellement chacune d'elles.

Pour une clause $C$ incidente à $x$, on définit la contribution locale $\delta_C(x)$ par

$$\delta_C(x) = \begin{cases} +1 & \text{si } C \text{ est actuellement insatisfaite et si le } \textit{flip} \text{ de } x \text{ rend } C \text{ satisfaite,} \\ -1 & \text{si } C \text{ est actuellement satisfaite par un unique littéral et si le } \textit{flip} \text{ de } x \text{ détruit cette satisfaction,} \\ 0 & \text{sinon.} \end{cases}$$

Le score local de $x$ est alors donné par

$$s(x) = \sum_{C \in \mathrm{Inc}(x)} \delta_C(x),$$

où $\mathrm{Inc}(x)$ désigne l'ensemble des clauses contenant $x$ ou $\neg x$.

Ainsi, un score strictement positif signifie que le *flip* de $x$ améliore localement l'état courant ; un score nul ou négatif indique qu'il n'apporte pas de gain immédiat.

Ainsi, les scores sont stockés dans un vecteur indexé par variable, tandis qu'une structure auxiliaire maintient l'ensemble des littéraux de score strictement positif :

```
using Stack = MaLib::unordered_uint_set;
Stack lit_with_positive_score;
```

Cette structure satisfait l'invariant suivant : un littéral appartient à *lit_with_positive_score*, l'ensemble des *good lits*, si et seulement si son score est strictement positif. Lorsqu'un score devient nul ou négatif, il en est retiré ; lorsqu'il devient strictement positif, il y est inséré.

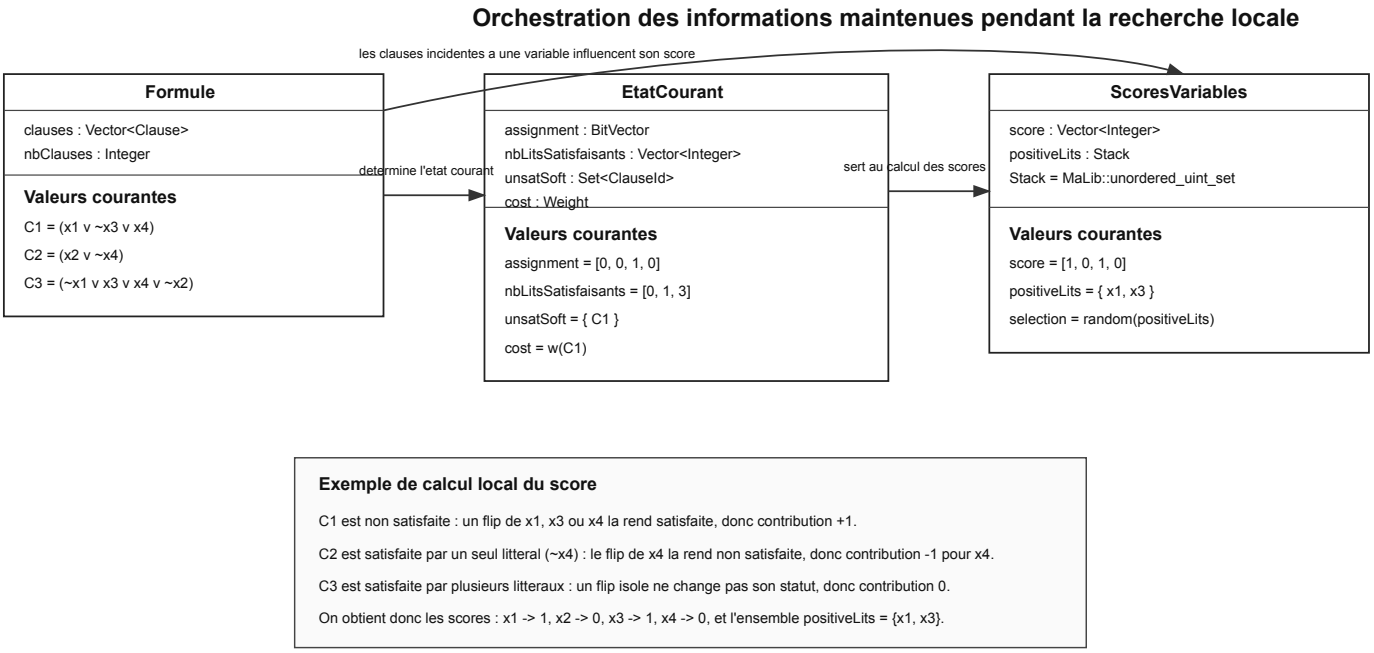



Figure 5.5 – Vue de type UML simplifiée de l'orchestration entre la formule, l'état courant et les scores des variables.

La figure 5.5 illustre cette organisation. Pour l'affectation

$$(x_1, x_2, x_3, x_4) = (0, 0, 1, 0),$$

la clause $C_1$ est insatisfaite et contribue donc positivement aux variables dont le *flip* la satisferait, ici $x_1$, $x_3$ et $x_4$. La clause $C_2$, satisfaite par un unique littéral, apporte une contribution négative à $x_4$. La clause $C_3$, satisfaite par plusieurs littéraux, n'apporte aucune contribution. On obtient ainsi

$$s(x_1) = 1, \qquad s(x_2) = 0, \qquad s(x_3) = 1, \qquad s(x_4) = 0,$$

et la structure *lit_with_positive_score* contient alors exactement les littéraux associés à $x_1$ et $x_3$.

L'accès au score d'une variable dans le vecteur indexé est constant. La maintenance de *lit_with_positive_score* repose sur des opérations d'insertion, de suppression et de test d'appartenance sans parcours global. Sous l'hypothèse usuelle où la mise à jour du score d'un littéral ne nécessite qu'un nombre borné d'opérations élémentaires sur ces structures, la lecture d'un score local et la maintenance de *lit_with_positive_score* s'effectuent donc en temps constant amorti.

Le surcoût mémoire associé à cet ensemble d'informations reste linéaire en le nombre de variables. Il demeure faible au regard de la taille globale de la formule, tout en jouant un rôle central dans l'efficacité

pratique du noyau de recherche locale.

### 5.2.6 Coût complet d'un mouvement

La mise à jour de l'état logique après un flip ne suffit pas, à elle seule, à maintenir le solveur. Lorsqu'une clause change de statut ou de nombre de littéraux satisfaisants, il faut également ajuster les scores des variables apparaissant dans cette clause. Cette seconde étape nécessite de reparcourir les littéraux des clauses affectées afin de corriger les contributions locales des variables concernées.

Par conséquent, le coût complet d'un mouvement ne se réduit pas à $\mathcal{O}(\deg(x))$. Si l'on note $C \ni x$ l'ensemble des clauses contenant la variable retournée, alors le coût dominant d'un flip s'écrit plus précisément

$$T_{\mathsf{flip}}(x) = \mathcal{O}\left(\sum_{C \ni x} |C|\right),$$

où $|C|$ désigne la taille de la clause $C$. Cette écriture rend compte du fait que le solveur ne parcourt pas toute la formule, mais seulement les clauses touchées par le mouvement, en rescannant leur contenu lorsque cela est nécessaire pour mettre à jour les scores locaux.

Dans le cas fréquent où la taille des clauses reste modérée, cette borne peut être interprétée comme une dépendance essentiellement locale au degré de la variable. Elle confirme que l'efficacité du solveur repose avant tout sur la capacité à limiter les mises à jour aux structures directement affectées par le flip.

### 5.2.7 Bilan de la section

Le noyau de recherche locale présenté ici repose sur une idée simple : maintenir suffisamment d'informations locales pour éviter toute réévaluation globale après chaque mouvement. La représentation compacte de la formule, l'indexation des clauses par littéraux, le suivi incrémental de l'état courant et le maintien explicite des scores par variable forment ensemble une architecture cohérente orientée vers l'efficacité et vers l'étude contrôlée de variantes de recherche locale.

Cette organisation permet un régime ordinaire de recherche dans lequel la sélection d'un mouvement favorable est peu coûteuse et la mise à jour de l'état reste locale. Elle fait toutefois apparaître une limite importante : lorsque plus aucun flip immédiatement bénéfique n'est disponible, le noyau se bloque dans un minimum local. La section suivante introduit précisément les mécanismes proposés pour dépasser cette

difficulté.

Le pseudocode détaillé du solveur figure à l'annexe A.2. Une description synthétique des principales structures de données est rassemblée à l'annexe A.3, et des remarques complémentaires d'implémentation sont données à l'annexe A.4.

## 5.3 Minima locaux et mécanismes de sortie

La section précédente a présenté le noyau de recherche locale et les structures de données qui permettent d'effectuer les mouvements de manière efficace. Tant qu'il existe des variables dont le *flip* présente un gain local strictement positif, ce noyau permet une progression rapide à coût maîtrisé. En pratique, cette situation ne se maintient pas indéfiniment. La recherche atteint fréquemment des configurations dans lesquelles aucun *flip* immédiatement favorable n'est disponible, alors même que la solution courante reste sous-optimale. Le noyau local se trouve alors bloqué dans un minimum local.

Ce phénomène est classique dans les approches de recherche locale pour SAT et MaxSAT. Plus généralement, la littérature sur les méthodes de type SLS souligne que la stagnation de la recherche, sous la forme de minima locaux ou de pièges locaux, constitue l'une des principales difficultés de ce type d'algorithmes. Pour cette raison, de nombreux travaux introduisent des mécanismes complémentaires destinés à redonner de la mobilité à la recherche lorsque l'évaluation locale ordinaire ne permet plus de progresser (Pham *et al.*, 2005; Pham *et al.*, 2012; Zheng *et al.*, 2024a).

### 5.3.1 Blocage du noyau local

Dans le cadre considéré ici, un minimum local correspond à une affectation complète pour laquelle aucune variable ne possède de score strictement positif. Autrement dit, l'ensemble des *good lits* est vide. Le noyau local ne dispose alors plus d'un mouvement immédiatement favorable, bien que certaines clauses souples puissent encore rester insatisfaites.

Cette situation provient du caractère local du critère de décision. Un mouvement peut être défavorable à court terme tout en étant nécessaire pour rejoindre ensuite une région plus prometteuse de l'espace de recherche. Un noyau purement glouton tend ainsi à se stabiliser sur des affectations dont aucun voisin immédiat n'apparaît meilleur, sans pour autant que la solution obtenue soit globalement satisfaisante.

La figure 5.6 illustre de manière schématique la situation dans laquelle une recherche locale peut se bloquer sur un minimum local sans avoir atteint la meilleure région du paysage de recherche.

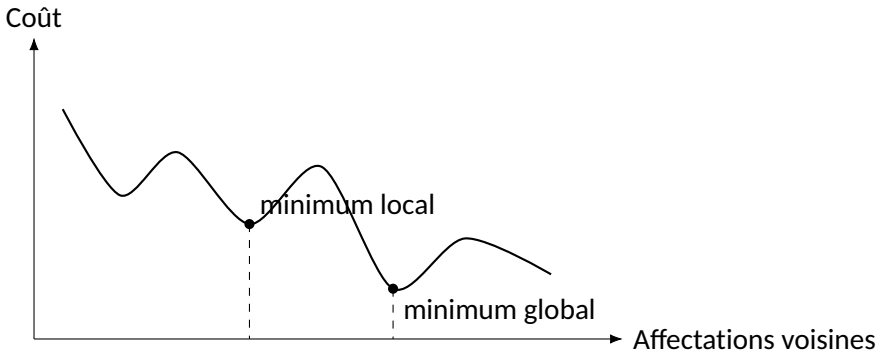



Figure 5.6 – Illustration schématique d'un paysage de recherche contenant un minimum local distinct du minimum global.

### 5.3.2 Arrêt et redémarrage comme premiers mécanismes de sortie

Une première réponse à ce blocage consiste à interrompre la recherche courante dès qu'aucun *flip* favorable n'est disponible. Une autre possibilité consiste au contraire à redémarrer depuis une nouvelle affectation faisable. Dans ce second cas, l'idée n'est pas de modifier le critère local lui-même, mais de replacer le noyau de recherche dans une autre région de l'espace des affectations.

Dans la base expérimentale étudiée ici, un redémarrage peut être construit à partir d'un solveur SAT appliqué aux clauses dures, éventuellement enrichi d'une information supplémentaire issue de la recherche en cours. Ce type de mécanisme constitue une première famille de sorties de minimum local, simple à mettre en place et facile à isoler expérimentalement.

### 5.3.3 Pondération dynamique des clauses insatisfaites

Une autre famille de mécanismes consiste à modifier temporairement l'information heuristique utilisée pour évaluer les mouvements. L'idée générale est d'augmenter le poids de certaines clauses insatisfaites afin de faire évoluer les scores locaux et de faire réapparaître des *flips* favorables. Dans ce cadre, les poids dynamiques utilisés par la recherche ne doivent pas être confondus avec les poids objectifs de l'instance : ils jouent uniquement un rôle de guidage heuristique.

Le recours à une pondération dynamique est classique dans la littérature sur la recherche locale pour SAT et MaxSAT. Le principe général consiste à augmenter le poids des clauses falsifiées lorsqu'un optimum local est atteint, afin de modifier temporairement les directions privilégiées par l'heuristique locale (Thornton *et al.*, 2004; Pham *et al.*, 2005). Des travaux plus récents sur MaxSAT local search rappellent également que le clause weighting reste une technique centrale de diversification (Zheng *et al.*, 2024a; Chu *et al.*, 2023).

### 5.3.4 Deux règles d'augmentation possibles

Une première possibilité consiste à augmenter les poids d'une quantité constante, par exemple de $1$. Ce type de mise à jour additive reste très présent dans les schémas classiques de clause weighting et constitue une référence naturelle pour l'étude expérimentale (Thornton *et al.*, 2004; Zheng *et al.*, 2024a).

Une deuxième possibilité consiste à calculer une augmentation minimale, suffisante pour faire apparaître un nouveau *good lit*. Si l'on note $s_{\max}$ le meilleur score parmi les variables d'une clause insatisfaite choisie, on peut définir

$$\delta = 1 - s_{\max}.$$

Cette règle constitue l'une des contributions d'implémentation proposées dans ce mémoire. Son intérêt est de produire la plus petite augmentation suffisante pour faire réapparaître un *good lit*, plutôt que d'introduire une perturbation uniforme du paysage heuristique. Elle permet ainsi d'évaluer si une mise à jour plus ciblée améliore le comportement de la recherche par rapport à une augmentation constante.

### 5.3.5 Débiaisage comme famille de mécanismes complémentaires

Quelle que soit la règle d'augmentation retenue, l'usage répété de poids dynamiques peut introduire un biais accumulé dans les scores locaux. À mesure que certaines clauses reçoivent des augmentations successives, leur influence peut refléter de plus en plus l'historique récent de la recherche, et de moins en moins la structure effective de l'instance au temps courant.

Dans une base expérimentale destinée à comparer plusieurs mécanismes, il est important d'identifier explicitement ce phénomène. Sans cela, il devient difficile de distinguer ce qui relève de la dynamique locale du noyau et ce qui provient simplement de l'accumulation historique de pondérations.

Pour limiter cet effet, plusieurs stratégies de débiaisage peuvent être envisagées. L'idée générale consiste à

éviter que les pondérations auxiliaires ne s'accumulent sans borne et à redonner progressivement du poids à l'information issue de l'état courant de la recherche. Selon les cas, cela peut passer par une décroissance, un lissage, une remise à l'échelle ou un oubli progressif des augmentations passées.

Dans le cadre de ce mémoire, l'objectif n'est pas seulement de rappeler l'intérêt général du débiaisage comme famille de mécanismes, mais de proposer et d'implanter une technique particulière de débiaisage intégrée à **EvalMaxSat Anytime**. C'est cette technique spécifique, et non la notion générale de débiaisage elle-même, qui constitue ici une contribution propre du travail.

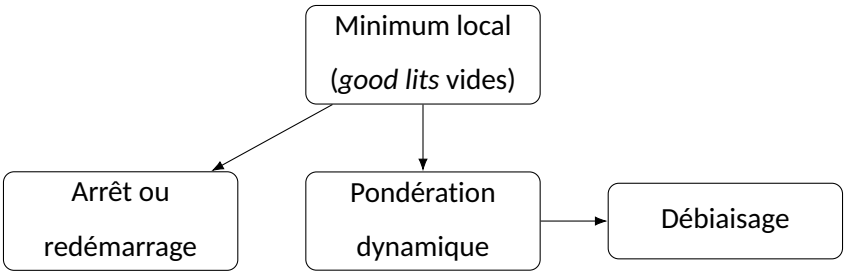



Figure 5.7 – Vue d'ensemble des principales familles de mécanismes discutées pour traiter les minima locaux.

Les variantes effectivement considérées dans les expériences sont introduites dans la section suivante.

### 5.3.6 Technique de débiaisage proposée : *WeightHistoryRing*

Le débiaisage est ici réalisé par une structure spécifique, `WeightHistoryRing`, qui mémorise temporairement les augmentations de poids heuristiques et les retire progressivement afin d'éviter une déformation durable du paysage heuristique.

Chaque augmentation est enregistrée sous la forme

$$(\textit{clauseId}, \delta, \textit{cycleId}),$$

où *clauseId* désigne la clause modifiée, $\delta$ la quantité ajoutée, et *cycleId* le cycle de recherche auquel appartient cette modification.

Le principe est celui d'une mémoire circulaire. Les ajustements récents sont conservés, car ils peuvent encore être utiles à court terme. Les plus anciens sont ensuite retirés par blocs, cycle par cycle. Lorsqu'un cycle

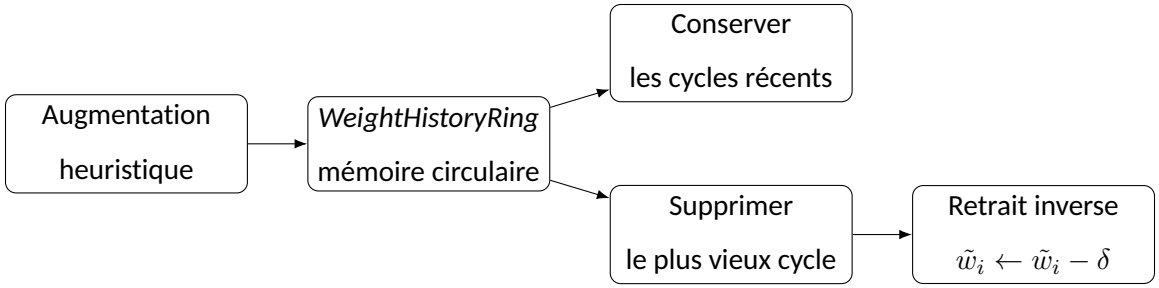



Figure 5.8 – Principe de *WeightHistoryRing* : les augmentations heuristiques sont conservées temporairement dans une mémoire circulaire, puis retirées progressivement lorsqu'elles deviennent trop anciennes.

est supprimé, l'opération inverse est appliquée aux clauses concernées :

$$\tilde{w}_i \leftarrow \tilde{w}_i - \delta.$$

Ce mécanisme est couplé à la dynamique du solveur. Lorsqu'une amélioration du meilleur coût est observée, les ajustements récents peuvent être conservés plus longtemps. En revanche, lorsque la recherche stagne, l'oubli peut être accéléré afin de rouvrir l'exploration.

Ainsi, le couple *incrémentation minimale + oubli progressif* constitue l'une des contributions d'implémentation de ce travail. L'incrémentation minimale fait réapparaître localement un *good lit*, tandis que *WeightHistoryRing* empêche que cette intervention ne devienne une déformation permanente du paysage heuristique.

**Algorithm 2** Principe simplifié de *WeightHistoryRing*

**function** Push($clauseId, \delta$)
    insérer $(clauseId, \delta, cycleId)$ dans le buffer circulaire
    **if** condition de fin de cycle **then**
        RemoveOldestCycle
        $cycleId \leftarrow cycleId + 1$
    **end if**
**end function**
**function** RemoveOldestCycle
    **for** chaque $(clauseId, \delta)$ du plus vieux cycle **do**
        RemoveWeightCallback($clauseId, \delta$)
    **end for**
**end function**

## 5.4 Variantes du solveur et protocole expérimental

La base de résolution présentée dans les sections précédentes a été conçue de manière modulaire afin de permettre l'activation ou la désactivation contrôlée de plusieurs mécanismes internes de recherche locale. Cette modularité rend possible une étude comparative dans laquelle les variantes partagent le même noyau de calcul, tout en différant par leur manière de réagir aux situations de stagnation. L'objectif de cette section est donc de définir précisément les variantes retenues pour l'étude, puis de présenter le protocole expérimental utilisé pour les comparer de manière systématique.

### 5.4.1 Logique générale des variantes étudiées

Les variantes considérées dans ce mémoire ne correspondent pas à des solveurs entièrement distincts, mais à des versions successives construites à partir d'une même base. Chaque nouvelle version ajoute ou modifie un mécanisme particulier, de façon à isoler autant que possible son effet sur la dynamique de recherche. Cette démarche permet de relier plus directement les différences observées en expérimentation aux choix algorithmiques introduits.

La progression retenue suit une logique simple. Une première version sert de point de départ minimal et se limite au noyau de recherche locale. Une deuxième introduit un mécanisme de redémarrage. Une troisième

ajoute une pondération dynamique des clauses insatisfaites, avec deux règles d'incrément possibles. Enfin, une quatrième complète ce dispositif par un mécanisme de débiaisage, lui aussi évalué sous deux configurations de pondération. L'ensemble forme ainsi une famille cohérente de variantes permettant d'étudier séparément l'effet du redémarrage, de la pondération et du débiaisage.

Au-delà de leur rôle dans la comparaison expérimentale, certaines de ces variantes permettent également d'évaluer deux contributions d'implémentation proposées dans ce travail. La première est une règle d'incrémentation minimale conçue pour faire réapparaître un *good lit* avec la plus faible perturbation possible du paysage heuristique. La seconde est une technique de débiaisage par mémoire circulaire, implantée dans **EvalMaxSat Anytime** sous la forme de la structure *WeightHistoryRing*, destinée à limiter l'accumulation des poids auxiliaires et à stabiliser la recherche lorsque l'horizon de résolution s'allonge. L'étude comparative de ces variantes vise donc non seulement à comparer plusieurs comportements de recherche, mais aussi à apprécier la portée pratique de ces deux contributions.

### 5.4.2 Définition des variantes

#### 5.4.2.1 Version *v0* : noyau simple.

La variante *EvalMaxSat_anytime_v0_simple* correspond à la version de base du solveur. Elle repose uniquement sur le noyau de recherche locale présenté à la section précédente. Tant qu'au moins un *good lit* est disponible, la recherche continue. En revanche, si aucun mouvement localement favorable n'est disponible, l'exécution s'arrête. Cette version sert donc de point de référence minimal pour évaluer l'intérêt des mécanismes de sortie de minimum local.

#### 5.4.2.2 Version *v1* : redémarrage simple.

La variante *EvalMaxSat_anytime_v1_simple_restart* prolonge la version précédente en introduisant un mécanisme de redémarrage lorsqu'aucun *good lit* n'est disponible. Le redémarrage est construit à partir d'une nouvelle affectation faisable obtenue avec le solveur SAT sur les clauses dures, à laquelle est ajoutée une clause souple non satisfaite pendant le round courant. L'objectif est de relancer la recherche dans une autre région du paysage tout en conservant une initialisation guidée par la structure courante du problème.

#### 5.4.2.3 Version *v2* : pondération dynamique.

La famille *v2* prolonge la logique de redémarrage en introduisant une pondération dynamique des clauses insatisfaites. Dans ces variantes, le redémarrage n'est plus déclenché dès l'absence de *good lit*, mais lorsqu'aucune amélioration de la meilleure solution connue n'est observée pendant une durée donnée $t$. Cette modification vise à laisser davantage de temps à la recherche locale pour exploiter un paysage temporairement modifié par la pondération.

Deux règles d'incrément sont étudiées.

La variante *EvalMaxSat_anytime_v2_ponderation_inc1* utilise une augmentation constante égale à $1$. Ce choix fournit une version simple et classique de la pondération dynamique.

La variante *EvalMaxSat_anytime_v2_ponderation_inc_minweight* utilise au contraire une augmentation minimale calculée de façon à faire apparaître un nouveau *good lit*. Si $s_{\max}$ désigne le meilleur score dans la clause choisie, l'incrément est donné par

$$\delta = 1 - s_{\max}.$$

Cette variante permet d'évaluer si une mise à jour plus ciblée améliore le comportement de la recherche par rapport à une augmentation constante.

#### 5.4.2.4 Version *v3* : pondération et débiaisage.

La famille *v3* reprend les variantes *v2* en leur ajoutant la technique de débiaisage proposée dans ce travail, fondée sur la structure *WeightHistoryRing*. L'objectif est de limiter l'accumulation des poids heuristiques et d'éviter que l'historique des blocages influence trop durablement la sélection des mouvements.

La variante *EvalMaxSat_anytime_v3_debiais_inc1* combine ainsi le débiaisage avec une augmentation constante de $1$.

La variante *EvalMaxSat_anytime_v3_debiais_inc_minweight* combine quant à elle le débiaisage avec l'augmentation minimale

$$\delta = 1 - s_{\max}.$$

### 5.4.3 Synthèse des exécutables comparés

Les six exécutables retenus pour l'étude sont donc les suivants :

Table 5.1 – Synthèse des variantes étudiées

| Version | Mécanisme principal | Exécutable |
|---|---|---|
| v0 | noyau simple | *EvalMaxSat_anytime_v0_simple* |
| v1 | redémarrage simple | *EvalMaxSat_anytime_v1_simple_restart* |
| v2 | pondération, incrément constant | *EvalMaxSat_anytime_v2_ponderation_inc1* |
| v2 | pondération, incrément minimal | *EvalMaxSat_anytime_v2_ponderation_inc_minweight* |
| v3 | débiaisage + incrément constant | *EvalMaxSat_anytime_v3_debiais_inc1* |
| v3 | débiaisage + incrément minimal | *EvalMaxSat_anytime_v3_debiais_inc_minweight* |

Pris ensemble, ces exécutables permettent d'étudier successivement l'effet de trois dimensions : la présence ou non d'un mécanisme de redémarrage, l'introduction d'une pondération dynamique, puis l'ajout d'un débiaisage. Ils permettent également de comparer, pour les versions pondérées, deux stratégies d'augmentation des poids, l'une constante et l'autre minimale.

### 5.4.4 Jeu d'instances et cadre expérimental

Les variantes sont évaluées sur un sous-ensemble d'instances issu de la MaxSAT Evaluation 2024. Contrairement au chapitre 4, qui visait une évaluation large des solveurs sur les tracks complets de la compétition, le présent chapitre adopte volontairement un protocole local et contrôlé, orienté vers la comparaison fine de mécanismes internes d'un même solveur.

Le banc d'essai repose sur $40$ instances, réparties en deux groupes de même taille : $20$ instances pondérées et $20$ instances non pondérées. La sélection n'a pas été effectuée par tirage uniforme. Elle a été construite par échantillonnage raisonné à partir des benchmarks officiels, selon trois critères : *(i)* préserver la séparation entre les deux catégories d'instances ; *(ii)* conserver une diversité de familles d'instances, identifiables par leur provenance ou par les préfixes des noms de fichiers ; *(iii)* couvrir des difficultés contrastées, en évitant qu'une seule famille ou qu'une série d'instances très proches ne domine à elle seule les résultats.

L'objectif n'était donc pas de reproduire à petite échelle la distribution complète de la compétition, mais d'obtenir un sous-ensemble suffisamment varié pour comparer les mécanismes étudiés dans un cadre expérimental maîtrisé.

Pour chaque couple variante–instance, l'exécution est limitée à un temps de calcul maximal de $300$ secondes et répétée trois fois afin de tenir compte de la variabilité liée à l'initialisation et à la sélection aléatoire des mouvements. Les expériences sont réalisées sur une machine équipée d'un processeur Intel Core i7-14700F et de $32$ Go de mémoire vive.

Comme au chapitre précédent, les analyses ne reposent pas sur une exécution isolée. Pour chaque couple variante–instance, les trois exécutions sont agrégées en une trajectoire représentative selon la procédure décrite à la section 3.3.4. Les visualisations et les indicateurs synthétiques présentés dans ce chapitre sont calculés à partir de ces trajectoires agrégées.

Le pseudocode détaillé du solveur figure à l'annexe A.2. Une description synthétique des principales structures de données est rassemblée à l'annexe A.3, et des remarques complémentaires d'implémentation sont données à l'annexe A.4.

### 5.4.5 Mesures retenues

L'étude ne se limite pas à la qualité finale de la meilleure solution trouvée au *timeout*. Elle prend également en compte la dynamique *anytime* des variantes, c'est-à-dire leur manière de progresser au cours du temps. Cette perspective est cohérente avec l'objectif général de la base développée dans ce mémoire, qui n'est pas seulement de produire une valeur finale, mais aussi de permettre une analyse fine de l'évolution de la recherche.

Les mesures considérées incluent donc la qualité finale atteinte, le temps nécessaire pour atteindre la meilleure solution connue pendant une exécution, ainsi que les trajectoires de coût observées tout au long de l'exécution. Ces trajectoires sont exploitées dans la section suivante pour comparer les variantes non seulement à un horizon fixe, mais aussi selon leur comportement à différents instants de la recherche.

À partir de ces trajectoires agrégées, nous calculons également des métriques de domination temporelle, en particulier l'aire sous la courbe (AUC), le score de domination temporelle (SDT), ainsi que leurs variantes

logarithmiques AUC-log et SDT-log, afin de distinguer les comportements dominants selon l'horizon temporel considéré.

### 5.4.6 Rôle de cette étude comparative

L'intérêt principal de ce protocole est de permettre une comparaison progressive des mécanismes introduits dans le chapitre. La version *v0* fournit une base minimale. La version *v1* permet d'évaluer l'apport d'un redémarrage simple. Les variantes *v2* mesurent l'effet de la pondération dynamique selon deux règles d'augmentation. Enfin, les variantes *v3* permettent d'étudier si l'ajout d'un débiaisage améliore ou stabilise le comportement observé dans les versions pondérées sans débiaisage.

La section suivante présente les résultats obtenus sur ce banc d'essai et analysera l'effet de ces choix sur la qualité des solutions ainsi que sur la dynamique *anytime* du solveur.

## 5.5 Résultats expérimentaux et analyse

Cette section présente les résultats obtenus sur les variantes définies à la section précédente. L'objectif n'est pas seulement de comparer les versions du solveur à partir d'indicateurs finaux, mais surtout d'analyser la manière dont les différents mécanismes modifient la dynamique de recherche. Les résultats doivent donc être lus avant tout comme une étude de comportement.

Comme dans le chapitre précédent, l'analyse est conduite séparément sur les instances *weighted* et sur les instances *unweighted*. Dans chaque cas, nous commençons par une visualisation agrégée des trajectoires, qui fournit une lecture d'ensemble du comportement temporel des variantes. Cette lecture est ensuite complétée par des indicateurs synthétiques, afin de préciser les hiérarchies observées à différents horizons temporels.

Les observations rapportées dans cette section reposent sur une campagne expérimentale volontairement limitée à un sous-ensemble contrôlé d'instances. Elles visent avant tout à faire apparaître des tendances entre les variantes étudiées dans un cadre local cohérent, et non à établir une hiérarchie universelle entre mécanismes.

### 5.5.1 Résultats agrégés sur les instances *weighted*

#### 5.5.1.1 Visualisation agrégée des trajectoires

Nous examinons d'abord les trajectoires agrégées des différentes variantes sur l'ensemble des instances *weighted*. Afin de rendre simultanément visibles les comportements à très court horizon et les évolutions à moyen et long terme, nous retenons une représentation unique du score relatif moyen sur l'horizon complet de 300 secondes, avec une abscisse en échelle logarithmique.

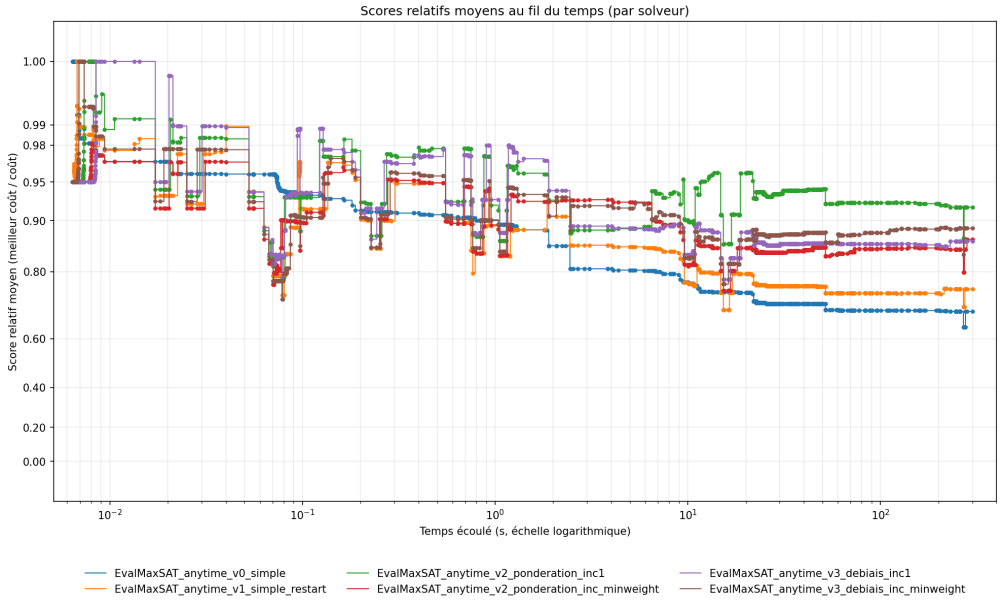



Figure 5.9 – Évolution du score relatif moyen des variantes d'*EvalMaxSat Anytime* sur les instances *weighted*, sur l'horizon complet de 300 secondes, avec une abscisse logarithmique. Cette vue d'ensemble met en évidence la dynamique globale des différentes variantes sur toute la durée d'exécution.

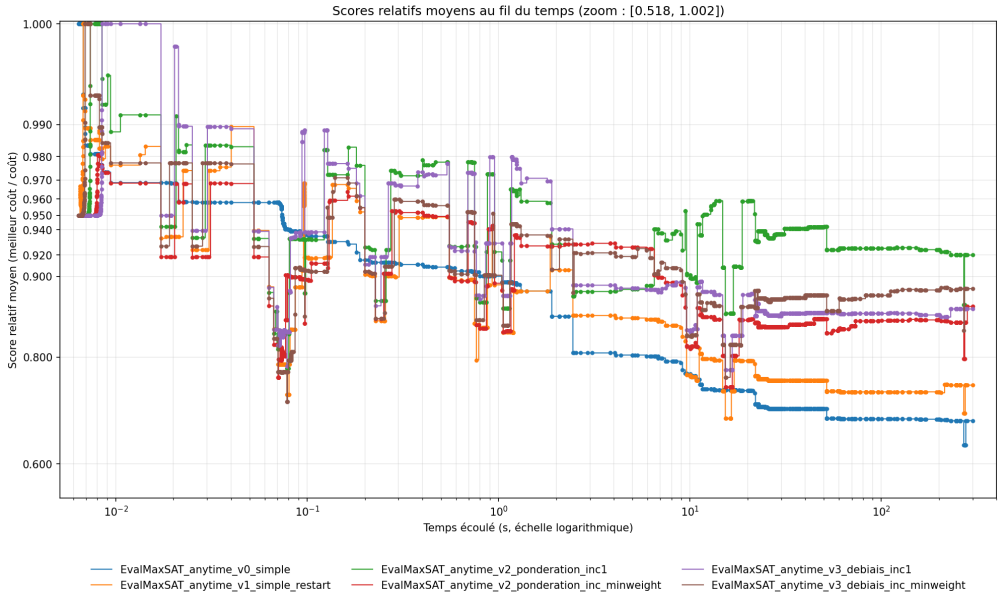



Figure 5.10 – Vue complémentaire de l'évolution du score relatif moyen des variantes d'*EvalMaxSat Anytime* sur les instances *weighted*, sur le même horizon de 300 secondes et avec la même abscisse logarithmique, mais avec un zoom sur la zone utile de l'axe des ordonnées afin de mieux faire apparaître les écarts fins entre variantes.

La Figure 5.9 met en évidence une compétition temporelle contrastée entre les variantes. Les premières fractions de seconde restent toutefois délicates à interpréter isolément, car les fluctuations y sont très rapides et les hiérarchies peuvent changer sur des intervalles extrêmement courts. La Figure 5.10 complète cette lecture en rendant plus visibles les écarts fins lorsque les courbes deviennent très proches. Malgré cela, l'interprétation visuelle seule ne suffit pas à caractériser précisément les rapports de domination entre variantes sur l'ensemble de l'horizon temporel. Pour cette raison, cette analyse est complétée par des indicateurs synthétiques de domination temporelle.

#### 5.5.1.2 Indicateurs synthétiques de domination temporelle

Le Tableau 5.2 synthétise cette lecture à l'aide de quatre métriques complémentaires. Les mesures AUC et SDT sont calculées sur l'échelle temporelle linéaire et résument la domination sur l'ensemble de l'horizon $[0, 300]$. Les mesures AUC-log et SDT-log reposent sur une échelle logarithmique du temps et donnent donc davantage de poids à la phase initiale de la recherche.

| Variante | AUC | SDT | AUC-log | SDT-log |
|---|---|---|---|---|
| v2_ponderation_inc1 | **277.42623** | **0.92477** | **10.09463** | **0.93846** |
| v3_debiais_inc_minweight | 264.83432 | 0.88280 | 9.82589 | 0.91348 |
| v3_debiais_inc1 | 258.40925 | 0.86138 | 9.85893 | 0.91655 |
| v2_ponderation_inc_minweight | 255.25572 | 0.85087 | 9.69063 | 0.90090 |
| v1_simple_restart | 226.14174 | 0.75382 | 9.28700 | 0.86338 |
| v0_simple | 210.19804 | 0.70068 | 9.09075 | 0.84514 |

Table 5.2 – Métriques de domination temporelle sur les instances *weighted*.

Le tableau confirme d'abord une hiérarchie beaucoup plus nette que ne le laisse parfois penser la seule inspection visuelle de la courbe. En effet, la variante *v2_ponderation_inc1* occupe la première place pour les quatre métriques. Elle domine donc non seulement sur l'horizon complet, mais aussi lorsque l'on accorde davantage d'importance aux débuts de trajectoire. Cette convergence entre AUC, SDT, AUC-log et SDT-log renforce l'idée que l'augmentation constante des poids heuristiques constitue ici le mécanisme le plus robuste.

La variante *v3_debiais_inc1* présente toutefois un profil intéressant. Si elle n'est que troisième en AUC et en

SDT sur l'échelle linéaire, elle remonte à la deuxième place pour les deux métriques logarithmiques. Cela suggère que l'ajout du débiaisage, combiné à un incrément constant, produit un comportement compétitif sur les phases précoces ou intermédiaires de la recherche, sans toutefois suffire à dépasser *v2_ponderation_inc1* lorsque l'on considère l'ensemble de l'horizon de 300 secondes.

Les variantes fondées sur l'incrément minimal restent en retrait. On observe néanmoins que *v3_debiais_inc_minweight* domine systématiquement *v2_ponderation_inc_minweight*. Le débiaisage améliore donc bien cette famille de variantes, mais sans inverser la hiérarchie principale : sur ce sous-ensemble *weighted*, ce n'est pas le débiaisage qui constitue le facteur décisif, mais bien le choix d'un incrément constant.

Dans l'ensemble, les résultats *weighted* conduisent donc à une conclusion claire. Le noyau simple et le redémarrage restent nettement en dessous des variantes pondérées. Parmi celles-ci, la pondération dynamique à incrément constant apparaît comme le mécanisme le plus efficace, tandis que le débiaisage joue ici un rôle secondaire, utile pour stabiliser certaines phases de la trajectoire, mais insuffisant pour renverser la hiérarchie globale.

### 5.5.2 Résultats agrégés sur les instances *unweighted*

#### 5.5.2.1 Visualisation agrégée des trajectoires

Nous examinons maintenant les trajectoires agrégées des variantes sur l'ensemble des instances *unweighted*. Comme précédemment, nous retenons une représentation unique du score relatif moyen sur l'horizon complet de 300 secondes, avec une abscisse en échelle logarithmique, afin de visualiser simultanément la phase initiale et les comportements à plus long terme.

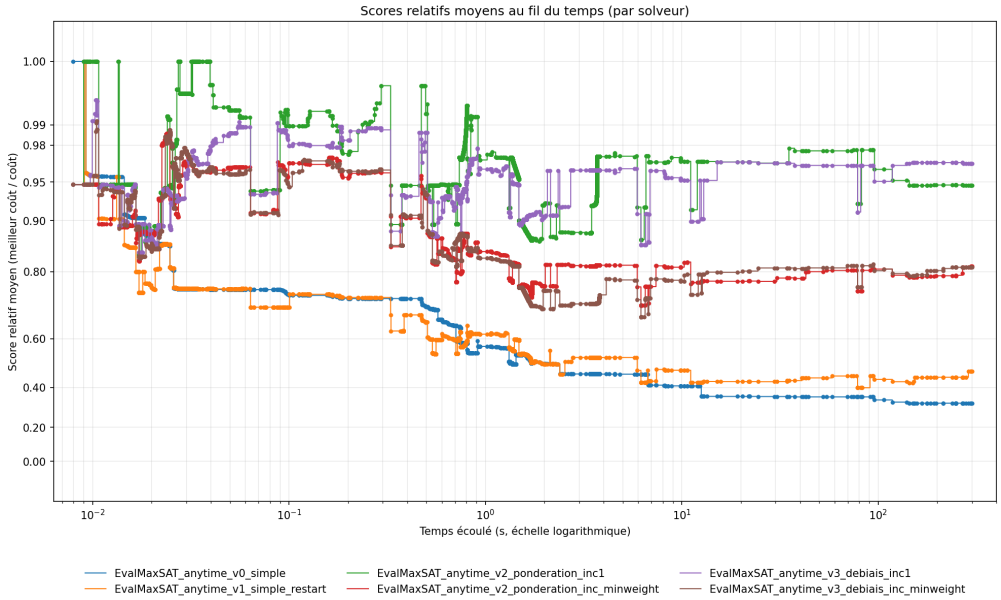



Figure 5.11 – Évolution du score relatif moyen des variantes d'*EvalMaxSat Anytime* sur les instances *unweighted*, sur l'horizon complet de 300 secondes, avec une abscisse logarithmique. Cette vue d'ensemble permet d'observer la dynamique globale des différentes variantes sur toute la durée d'exécution.

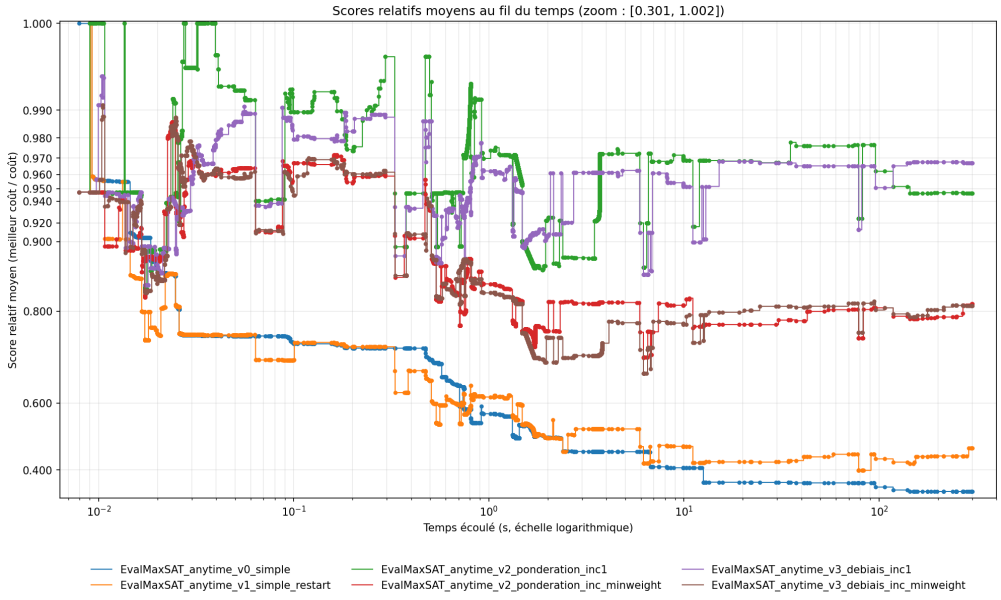



Figure 5.12 – Vue complémentaire de l'évolution du score relatif moyen des variantes d'*EvalMaxSat Anytime* sur les instances *unweighted*, sur le même horizon de 300 secondes et avec la même abscisse logarithmique, mais avec un zoom sur la zone utile de l'axe des ordonnées afin de mieux faire apparaître les écarts fins entre variantes.

La Figure 5.11 fait apparaître une hiérarchie plus structurée que dans le cas *weighted*. Deux variantes se détachent plus nettement de l'ensemble : *v2_ponderation_inc1* et *v3_debiais_inc1*. La première semble dominer la plus grande partie de la trajectoire utile, tandis que la seconde présente un comportement plus stable en fin d'horizon. La Figure 5.12 complète cette lecture en rendant plus visibles les écarts fins lorsque les courbes deviennent proches. Comme dans le cas *weighted*, cette impression visuelle gagne toutefois à être consolidée par des indicateurs synthétiques de domination temporelle.

#### 5.5.2.2 Indicateurs synthétiques de domination temporelle

Le Tableau 5.3 présente les mêmes métriques de domination temporelle que pour le sous-ensemble *weighted*. Leur intérêt est particulièrement fort ici, car elles permettent de distinguer plus clairement ce qui relève d'une domination précoce et ce qui relève d'une domination cumulée sur l'ensemble des 300 secondes.

| Variante | AUC | SDT | AUC-log | SDT-log |
|---|---|---|---|---|
| v3_debiais_inc1 | **289.20545** | **0.96404** | 10.04832 | 0.95321 |
| v2_ponderation_inc1 | 286.58085 | 0.95529 | **10.10069** | **0.95818** |
| v3_debiais_inc_minweight | 240.64405 | 0.80217 | 8.94920 | 0.84894 |
| v2_ponderation_inc_minweight | 238.11489 | 0.79374 | 9.00864 | 0.85458 |
| v1_simple_restart | 132.76368 | 0.44256 | 6.28332 | 0.59605 |
| v0_simple | 102.31365 | 0.34105 | 6.01509 | 0.57061 |

Table 5.3 – Métriques de domination temporelle sur les instances *unweighted*.

Contrairement au cas *weighted*, le tableau ne désigne pas ici un vainqueur unique pour toutes les métriques. En effet, *v3_debiais_inc1* obtient la meilleure AUC et le meilleur SDT sur l'échelle linéaire, tandis que *v2_ponderation_inc1* obtient la meilleure AUC-log et le meilleur SDT-log. Cette dissociation est particulièrement instructive pour l'interprétation des mécanismes.

Les métriques logarithmiques indiquent que *v2_ponderation_inc1* domine davantage lorsque l'on met l'accent sur les débuts de trajectoire. Autrement dit, l'incrément constant sans débiaisage semble offrir la progression la plus efficace dans les premières phases de la recherche. Cette lecture est cohérente avec la partie gauche de la courbe, où cette variante figure très souvent parmi les plus hautes.

À l'inverse, les métriques linéaires montrent que *v3_debiais_inc1* prend l'avantage lorsqu'on considère l'en-

semble de l'horizon jusqu'à 300 secondes. Le débiaisage ne semble donc pas agir ici comme un simple accélérateur immédiat. Son effet apparaît plutôt comme un mécanisme de régulation qui permet de mieux maintenir la qualité de la recherche lorsque le temps de résolution s'allonge. La partie terminale de la courbe confirme cette interprétation, puisque *v3_debiais_inc1* y devient la variante la plus compétitive.

Les variantes à incrément minimal restent, là encore, en retrait par rapport aux deux meilleures variantes. Elles se situent dans une zone intermédiaire, nettement au-dessus du noyau simple et du redémarrage, mais clairement en dessous des versions à incrément constant. Cela confirme que, sur ce banc d'essai, le choix de la règle d'incrément a un effet plus structurant que l'ajout du redémarrage seul.

Dans l'ensemble, les résultats *unweighted* mettent donc en évidence une articulation temporelle plus fine entre les mécanismes. La pondération dynamique à incrément constant domine sur les horizons précoces, tandis que le débiaisage permet d'obtenir la meilleure domination cumulée sur l'horizon complet. Ce point constitue l'un des résultats les plus intéressants de la campagne, car il montre que l'évaluation d'une variante dépend aussi du point de vue temporel adopté.

### 5.5.3 Comparaison transversale des mécanismes

Pris ensemble, les résultats ne désignent pas seulement une variante en tête ; ils permettent surtout de préciser le rôle des mécanismes introduits dans la base de solveur.

Le premier constat est que le noyau simple et le redémarrage restent nettement en retrait par rapport aux variantes pondérées, aussi bien sur les instances *weighted* que sur les instances *unweighted*. Le redémarrage apporte bien une amélioration par rapport au noyau initial, mais cette amélioration reste modeste au regard de l'effet produit par la pondération dynamique.

Le deuxième constat est que le choix de la règle d'incrément joue un rôle central. Dans les deux sous-ensembles, les variantes à incrément constant dominent les variantes à incrément minimal. Ce résultat est stable et apparaît aussi bien dans la lecture visuelle des trajectoires que dans les tableaux AUC/SDT. Il suggère que, dans cette campagne expérimentale, une perturbation plus simple mais plus franche du paysage heuristique est plus efficace qu'un ajustement strictement minimal.

Le troisième constat concerne le débiaisage. Son effet n'est pas uniforme selon le type d'instances considéré.

Sur les instances *weighted*, il n'inverse pas la hiérarchie globale : *v2_ponderation_inc1* reste la meilleure variante pour les quatre métriques de domination temporelle. Le débiaisage semble donc y jouer un rôle secondaire, en améliorant certaines variantes sans devenir le facteur principal de performance.

Sur les instances *unweighted*, en revanche, le débiaisage acquiert une portée plus nette. Il ne permet pas de dominer les toutes premières phases de recherche, comme le montrent les métriques logarithmiques, mais il améliore la stabilité à plus long horizon, au point de produire la meilleure domination cumulée sur 300 secondes. Le débiaisage apparaît ainsi moins comme un mécanisme d'accélération immédiate que comme un mécanisme de régulation dont l'intérêt augmente avec la durée d'exécution.

Cette comparaison transversale conduit donc à une lecture plus nuancée des mécanismes étudiés. La pondération dynamique constitue le levier principal de progression. Le choix d'un incrément constant est le facteur le plus robuste sur les deux sous-ensembles. Le débiaisage, quant à lui, agit de manière plus contextuelle : il reste secondaire sur les instances *weighted*, mais devient particulièrement pertinent sur les instances *unweighted* lorsque l'on s'intéresse au comportement à long horizon.

### 5.5.4 Lecture méthodologique des résultats

Ces résultats confirment l'intérêt de la base expérimentale modulaire introduite dans ce chapitre. Comme les variantes partagent le même noyau local, les écarts observés peuvent être reliés de manière relativement directe aux mécanismes ajoutés ou modifiés.

Trois enseignements principaux se dégagent. Premièrement, la sortie de minimum local constitue un point déterminant du comportement du solveur. Deuxièmement, la pondération dynamique apparaît comme le mécanisme le plus structurant parmi ceux étudiés. Troisièmement, le débiaisage n'apporte pas un gain uniforme dans tous les contextes, mais devient particulièrement pertinent lorsque l'effet de la pondération s'accumule sur des horizons temporels plus longs.

Dans l'ensemble, ces observations doivent être lues comme des tendances mises en évidence sur une campagne expérimentale limitée, mais construite de façon à rester suffisamment représentative pour comparer les mécanismes étudiés. Elles ne permettent pas d'établir une hiérarchie universelle entre toutes les variantes possibles, mais elles montrent clairement que les choix effectués pour traiter les minima locaux influencent fortement la dynamique *anytime* du solveur.

La discussion qui suit reprend ces résultats afin d'en dégager les principaux enseignements, les limites et la portée méthodologique.

## 5.6 Discussion et synthèse

Les résultats présentés dans la section précédente permettent de tirer plusieurs enseignements sur le comportement des mécanismes étudiés, tout en mettant en évidence certaines limites de la campagne expérimentale menée. L'intérêt principal de cette étude ne réside pas seulement dans l'identification d'une variante mieux classée que les autres sur le banc d'essai retenu, mais dans la compréhension plus fine du rôle joué par chaque mécanisme dans la dynamique de recherche. Dans cette perspective, la discussion porte ici sur trois points : le comportement du noyau simple face aux minima locaux, l'effet structurant de la pondération dynamique, et l'apport plus nuancé du débiaisage.

### 5.6.1 Enseignements principaux sur les mécanismes étudiés

Un premier enseignement concerne le noyau de recherche locale lui-même. Les résultats montrent qu'une stratégie simple, fondée uniquement sur les *good lits*, peut être très réactive dans les tout premiers instants de la recherche. Ce comportement est particulièrement visible sur les instances *weighted*, où la version *v0* obtient le meilleur score relatif dans la fenêtre temporelle la plus courte. Cette observation confirme qu'un noyau local léger, sans mécanisme additionnel, est capable d'exploiter rapidement les opportunités immédiates offertes par le paysage de recherche. En revanche, cette stratégie atteint rapidement ses limites dès lors qu'aucun *flip* favorable n'est disponible. Le noyau simple apparaît donc moins comme une solution suffisante à lui seul que comme une base de référence utile pour mesurer l'apport des mécanismes de sortie de minimum local.

Un deuxième enseignement, plus marqué, concerne la pondération dynamique. Sur la campagne expérimentale considérée, c'est ce mécanisme qui modifie le plus clairement le comportement du solveur. Dans les deux sous-ensembles étudiés, les variantes pondérées dépassent nettement les versions sans pondération dès que la recherche doit se prolonger au-delà des premières améliorations locales. Cela suggère que la pondération ne joue pas un rôle marginal ou simplement correctif, mais qu'elle transforme effectivement la manière dont le solveur explore l'espace des affectations. Elle permet de maintenir une progression là où le noyau simple et le redémarrage seul montrent plus rapidement leurs limites.

Le troisième enseignement concerne le choix de la règle d'incrément. Dans les expériences menées ici, l'augmentation constante des poids produit des comportements plus favorables que l'augmentation minimale. Cette observation est intéressante, car la règle minimale peut sembler plus naturelle du point de vue du contrôle local : elle introduit la plus petite perturbation permettant de faire apparaître un nouveau *good lit*. Pourtant, sur ce banc d'essai, cette stratégie ne conduit pas aux trajectoires les plus avantageuses. À l'inverse, une augmentation constante, plus simple et plus brutale, semble fournir une modification du paysage heuristique mieux adaptée à la poursuite de la recherche. Ce résultat ne signifie pas qu'une règle minimale est en soi moins pertinente dans tous les contextes, mais il indique qu'elle n'est pas la plus efficace parmi les choix étudiés ici.

Enfin, le débiaisage joue un rôle plus nuancé. Sur les instances *weighted*, il améliore certaines trajectoires et rend notamment *v3_debiais_inc1* plus compétitive sur les métriques en temps logarithmique, mais il ne renverse pas la hiérarchie globale : *v2_ponderation_inc1* reste la meilleure variante pour l'ensemble des indicateurs de domination temporelle. En revanche, sur les instances *unweighted*, l'ajout du débiaisage devient plus intéressant à long horizon, puisque *v3_debiais_inc1* obtient la meilleure domination cumulée sur l'intervalle complet de 300 secondes. Le débiaisage n'apparaît donc pas comme un mécanisme d'accélération immédiate, mais plutôt comme un mécanisme de régulation dont l'intérêt dépend du type d'instances et de l'horizon temporel considéré.

Ces observations permettent aussi de préciser la portée des deux contributions d'implémentation proposées dans ce mémoire. D'une part, la règle d'incrémentation minimale montre qu'il est possible de construire une pondération explicitement orientée vers la réapparition d'un *good lit*. Les résultats indiquent toutefois que, sur la campagne étudiée ici, cette stratégie ne conduit pas aux trajectoires les plus favorables, ce qui souligne l'écart possible entre élégance locale du mécanisme et efficacité globale. D'autre part, la technique de débiaisage `WeightHistoryRing` ne semble pas agir comme un accélérateur immédiat, mais plutôt comme un mécanisme de régulation dont l'intérêt apparaît lorsque la recherche se prolonge et que l'accumulation des poids heuristiques devient plus marquée.

### 5.6.2 Portée méthodologique de l'étude comparative

Au-delà des performances observées, cette étude met en évidence l'intérêt méthodologique d'une base de solveur modulaire. Comme les variantes partagent le même noyau de recherche locale et les mêmes structures de données principales, les écarts observés peuvent être attribués de manière relativement directe

aux mécanismes ajoutés ou modifiés. Cette propriété donne davantage de lisibilité à l'analyse expérimentale : au lieu de comparer des solveurs entièrement différents, on compare ici des versions construites par enrichissements successifs d'une même base.

Cette approche permet notamment de distinguer plusieurs niveaux d'intervention sur la recherche. Le noyau simple agit au niveau des mouvements immédiatement favorables. Le redémarrage agit au niveau de la relance globale de la recherche. La pondération agit au niveau des scores heuristiques en modifiant localement le paysage. Le débiaisage agit enfin au niveau de la mémoire laissée par ces modifications successives. La comparaison de ces mécanismes dans un cadre unifié permet donc non seulement d'observer des différences de performance, mais aussi de mieux comprendre à quel niveau chacun agit sur la dynamique du solveur.

De ce point de vue, la campagne expérimentale menée dans ce chapitre remplit bien son rôle. Elle ne vise pas à produire un solveur définitif, ni à rivaliser directement avec les meilleurs solveurs de l'état de l'art sur l'ensemble des benchmarks possibles. Elle permet plutôt d'examiner, dans un cadre contrôlé, les effets associés à plusieurs choix de conception. Cette finalité est cohérente avec l'objectif général de **EvalMaxSat Anytime**, conçu comme une base expérimentale autant que comme un solveur.

### 5.6.3 Limites de l'analyse

Les observations précédentes doivent toutefois être interprétées avec prudence. La première limite tient à la taille du banc d'essai. Même si le jeu d'instances a été construit de manière à rester représentatif, il demeure volontairement restreint. Les tendances observées sont donc informatives, mais elles ne suffisent pas à établir une hiérarchie générale entre mécanismes dans tous les contextes possibles.

La deuxième limite concerne la nature même des métriques utilisées. Les classements finaux donnent une indication utile sur la qualité atteinte au *timeout*, tandis que les scores relatifs par fenêtres temporelles rendent mieux compte de la dynamique *anytime*. Aucune de ces mesures, prise isolément, ne suffit à résumer entièrement le comportement d'une variante. C'est précisément pour cette raison que l'analyse a été articulée sur plusieurs indicateurs plutôt que sur un seul classement.

Une troisième limite tient au fait que seuls certains choix d'instanciation ont été étudiés. Pour la pondération, deux règles d'incrément ont été comparées, mais d'autres schémas sont possibles. De même, pour

le débiaisage, la campagne n'épuise pas l'ensemble des techniques envisageables. Les résultats obtenus permettent donc surtout d'éclairer les variantes effectivement testées, sans prétendre couvrir toutes les possibilités offertes par la littérature.

Enfin, le comportement observé dépend aussi du cadre expérimental retenu, notamment du *timeout*, du type d'instances considérées et du protocole de redémarrage. Dans ce chapitre, ces choix sont assumés comme faisant partie d'une étude ciblée sur les mécanismes internes du solveur. Ils doivent néanmoins être gardés à l'esprit dans l'interprétation des résultats.

### 5.6.4 Synthèse du chapitre

Dans l'ensemble, ce chapitre montre qu'un noyau de recherche locale simple peut constituer une base pertinente pour l'étude comparative de mécanismes plus avancés, à condition d'être soutenu par des structures de données adaptées et par une instrumentation suffisante pour observer la dynamique de recherche. L'analyse met en évidence que le traitement des minima locaux constitue un point central du comportement du solveur. Le redémarrage apporte une première réponse à cette difficulté, mais c'est surtout la pondération dynamique qui modifie en profondeur la trajectoire de recherche. Le débiaisage complète utilement ce dispositif dans certains contextes, en particulier lorsque la recherche se prolonge.

Plus largement, les résultats obtenus confirment l'intérêt d'une démarche expérimentale modulaire pour l'étude des solveurs *anytime*. En permettant d'isoler et de comparer plusieurs mécanismes au sein d'une même base de résolution, cette approche facilite une lecture plus fine des compromis entre simplicité, efficacité locale et stabilité à long terme.

La conclusion générale du mémoire prolonge cette discussion et reprend de manière synthétique les contributions, les limites et les perspectives du travail.

# CONCLUSION

Ce mémoire porte sur l'étude des solveurs MaxSAT *anytime*, c'est-à-dire des algorithmes capables de produire rapidement une solution valide, puis d'en améliorer progressivement la qualité sous une contrainte de temps. Ce cadre est particulièrement pertinent pour les problèmes d'optimisation combinatoire difficiles, où l'intérêt pratique d'un solveur ne dépend pas uniquement de sa capacité à atteindre l'optimum, mais aussi de la vitesse à laquelle il fournit de bonnes solutions intermédiaires. Dans cette perspective, l'objectif général du mémoire est double : mieux observer et comparer le comportement temporel de solveurs MaxSAT *anytime*, puis disposer d'une base algorithmique simple, modulaire et instrumentée pour étudier, de manière contrôlée, plusieurs mécanismes internes de recherche locale dans ce contexte.

Ces deux objectifs conduisent à deux contributions complémentaires. La première est *MaxSAT Runner*, un banc d'essai conçu pour capturer, reconstruire et analyser les trajectoires de résolution de solveurs MaxSAT *anytime*. La seconde est *EvalMaxSAT Anytime*, une base expérimentale de solveur destinée non pas à rivaliser immédiatement avec les meilleurs systèmes de l'état de l'art, mais à fournir un support clair, maintenable et instrumenté pour l'étude comparative de variantes heuristiques.

La première contribution, *MaxSAT Runner*, permet de déplacer l'évaluation d'une logique strictement classificatoire vers une logique plus analytique. L'outil permet de collecter en temps réel les améliorations de coût annoncées par les solveurs, de reconstruire leurs trajectoires sous forme de fonctions en escalier, puis de les agréger à différents niveaux. Il produit à la fois des sorties tabulaires et des visualisations exploitables dans une étude expérimentale, et calcule plusieurs familles d'indicateurs complémentaires, parmi lesquelles les scores relatifs à différents horizons, les métriques de domination temporelle, le temps moyen d'atteinte du meilleur coût, le coût final moyen, le nombre de *victoires* et le taux d'optimum.

Les résultats présentés au Chapitre 4 montrent que ce cadre d'analyse permettait de distinguer plusieurs profils de solveurs qui ne seraient pas nécessairement visibles dans un classement unique. Sur les instances *unweighted*, les écarts entre solveurs de tête restent relativement faibles, mais l'analyse des trajectoires montre que *NuWLS-2024* apparaît comme le solveur le plus constant au sens de la domination temporelle agrégée, tandis que *SPB-MaxSAT-BAND* domine davantage la lecture terminale. Sur les instances *weighted*, la hiérarchie dépend plus nettement de l'horizon considéré : *NuWLS-2024* domine les débuts de trajectoire, alors que *SPB-MaxSAT-FPS* s'impose plus clairement lorsque l'on considère la trajectoire complète et les

performances finales. Ces résultats confirment qu'un solveur *anytime* ne peut pas être évalué de manière satisfaisante à partir d'un unique indicateur final.

La deuxième contribution, *EvalMaxSAT Anytime*, consiste en la mise en place d'une base expérimentale de solveur MaxSAT *anytime* fondée sur des techniques de recherche locale stochastique pour *Weighted Partial MaxSAT*. La contribution ne réside pas dans la proposition immédiate d'un nouveau solveur compétitif face aux meilleurs solveurs récents, mais dans la conception d'un cadre modulaire, maintenable et suffisamment instrumenté pour isoler et comparer plusieurs mécanismes internes de recherche locale. Le solveur repose sur un noyau simple manipulant une assignation complète satisfaisant les clauses dures, puis améliorée par des *flips* guidés localement. À partir de cette base, plusieurs mécanismes de sortie de minima locaux sont étudiés, notamment le redémarrage, la pondération dynamique des clauses insatisfaites et une technique de débiaisage progressif fondée sur une mémoire circulaire, *WeightHistoryRing*.

Le Chapitre 5 montré que ces mécanismes influencent réellement la dynamique *anytime* du solveur. Le noyau simple apparaît capable de produire des améliorations rapides à très court terme, mais atteint rapidement ses limites lorsqu'aucun *flip* favorable n'est disponible. Le redémarrage constitue une première réponse utile à ce blocage, sans transformer la dynamique de recherche aussi fortement que la pondération dynamique. Celle-ci apparaît comme le mécanisme le plus structurant parmi ceux étudiés, en particulier au-delà des toutes premières itérations. Les résultats montrent également que, parmi les deux règles d'augmentation testées, l'augmentation constante produit des comportements plus favorables que l'augmentation minimale. Cette observation permet de mieux situer la portée de la règle d'incrémentation minimale proposée dans ce mémoire : elle constitue une stratégie de déblocage local conceptuellement ciblée, mais qui ne surpasse pas, sur la campagne étudiée, l'augmentation constante. Enfin, le débiaisage par *WeightHistoryRing* n'apparaît pas comme un mécanisme d'accélération immédiate, mais plutôt comme un mécanisme de régulation dont l'intérêt devient plus visible à plus long horizon.

L'un des enseignements centraux de ce mémoire est qu'un solveur *anytime* doit être évalué comme une trajectoire, et non comme un simple point final. Deux solveurs peuvent obtenir des résultats proches à un *timeout* donné tout en présentant des comportements très différents : l'un peut progresser très vite puis stagner, l'autre être plus lent au départ mais rester compétitif sur une plus grande partie de l'exécution. Une évaluation pertinente doit donc prendre en compte à la fois la qualité finale, la vitesse d'amélioration, la régularité de la trajectoire et la capacité à produire rapidement une bonne solution.

C'est précisément ce que permet *MaxSAT Runner*. Les expériences du Chapitre 4, menées sur l'ensemble des tracks *anytime unweighted* et *weighted* de la MaxSAT Evaluation 2024, montrent que cette approche ne se contente pas de confirmer certaines tendances observées dans les classements officiels : elle les réinterprète à la lumière des trajectoires complètes. En ce sens, l'apport principal du banc d'essai n'est pas seulement de classer les solveurs, mais de mieux comprendre *comment* ils sont performants.

Comme tout travail de recherche, ce mémoire présente plusieurs limites. La première concerne la portée des expériences sur *EvalMaxSAT Anytime*. Si le Chapitre 4 s'appuie sur une campagne large et reproductible sur les tracks complets de la MSE 2024, l'étude des variantes internes du solveur expérimental reste plus ciblée et repose sur un banc d'essai local contrôlé. Les tendances observées sont donc interprétables et utiles, mais elles ne suffisent pas à établir une hiérarchie universelle entre mécanismes heuristiques.

La deuxième limite tient à la simplicité volontaire de la base de solveur proposée. Cette simplicité est cohérente avec l'objectif de modularité et de lisibilité, mais elle implique aussi que certains mécanismes présents dans les meilleurs solveurs de l'état de l'art ne sont pas encore intégrés, ou ne le sont que sous une forme rudimentaire. Cela vaut notamment pour certaines stratégies de sélection avancées, des voisinages plus riches, des politiques de redémarrage plus élaborées ou encore des mécanismes de pondération plus sophistiqués.

La troisième limite concerne la méthodologie elle-même. L'analyse des trajectoires produit un volume important de données et impose des choix de représentation et d'agrégation. Les métriques linéaires et logarithmiques ne mettent pas en valeur les mêmes horizons, et leur interprétation doit donc être explicitée avec soin. Cette pluralité n'est pas un défaut, mais elle suppose une lecture méthodologique plus exigeante qu'un simple classement final.

Les résultats obtenus ouvrent plusieurs perspectives. Du côté de *MaxSAT Runner*, une première direction consiste à enrichir encore l'analyse des trajectoires, par exemple en approfondissant l'étude des distributions inter-exécutions, des profils de robustesse ou du lien entre comportement temporel et caractéristiques structurelles des instances. Une autre piste consiste à rapprocher davantage les analyses produites des protocoles officiels de compétition, en systématisant encore davantage l'exploitation de campagnes à grande échelle sur infrastructure de calcul intensif.

Du côté d'*EvalMaxSAT Anytime*, plusieurs prolongements apparaissent naturellement. Il serait d'abord utile

de poursuivre l'étude comparative des mécanismes de sortie de minima locaux, en revisitant les politiques de déclenchement, les règles d'augmentation des poids, les paramètres de mémoire ou encore les stratégies de redémarrage. Il serait également intéressant d'enrichir le voisinage exploré par le solveur, en introduisant par exemple des mouvements à deux *flips*, des perturbations plus structurées ou des mécanismes de diversification plus élaborés. Enfin, une piste particulièrement naturelle consiste à explorer davantage les stratégies hybrides, en s'inspirant du rapprochement observé dans l'état de l'art entre approches de recherche locale et approches SAT-based.

En définitive, ce mémoire a cherché à apporter une contribution à la fois méthodologique et algorithmique à l'étude des solveurs MaxSAT *anytime*. La première contribution, *MaxSAT Runner*, montre qu'il est possible de dépasser une lecture strictement finale des performances pour analyser plus finement les trajectoires de résolution. La seconde, *EvalMaxSAT Anytime*, montre qu'il est possible de disposer d'une base simple, modulaire et expérimentalement exploitable, au sein de laquelle des mécanismes précis peuvent être isolés, comparés et améliorés.

Si les résultats obtenus ne permettent pas encore de rivaliser avec les meilleurs solveurs de l'état de l'art, ils apportent néanmoins un cadre de travail cohérent, honnête et exploitable pour la poursuite de recherches futures. En ce sens, le mémoire remplit son objectif principal : poser des bases solides, à la fois pour mieux comprendre les solveurs *anytime* existants et pour développer, de manière progressive et contrôlée, de nouvelles approches de résolution MaxSAT.

# ANNEXE A

# DOCUMENTATION COMPLÉMENTAIRE

## A.1 README CLI de MaxSAT Runner

Cette annexe présente un mode d'emploi synthétique de `MaxSAT Runner` en ligne de commande. L'outil permet de lancer des campagnes expérimentales, d'exécuter des runs unitaires sur cluster, de générer des statistiques à partir des runs produits, puis d'effectuer un clustering d'instances à partir des trajectoires reconstruites. Les exemples ci-dessous reprennent les principales commandes disponibles dans la version actuelle du CLI.

### A.1.1 Installation

```
# Creer un environnement virtuel Python
python -m venv .venv

# Activer l'environnement (Linux / macOS)
source .venv/bin/activate

# Sous Windows PowerShell
# .\.venv\Scripts\Activate.ps1

# Mettre a jour pip
pip install -U pip

# Installer le projet
pip install -e ".[dev]"

# Dependances utiles pour le clustering
pip install scikit-learn
pip install networkx scipy fastdtw
```

Figure A.1 – Installation de MaxSAT Runner

### A.1.2 Vérification de l'installation

```
# Verifier les bibliotheques principales
python -c "import pandas, matplotlib; print('pandas', pandas.__version__, '|
    matplotlib', matplotlib.__version__)"

# Verifier que le binaire CLI est disponible
maxsat-runner --help
```

Figure A.2 – Vérification rapide de l'installation

### A.1.3 Organisation des données

```
data/
|- instances/                    # fichiers .wcnf d'entree
|- runs/                         # logs bruts + fichiers agreges
|  |- trajectories.csv           # trajectoires agregees
|  |- summary.csv                # resume agrege
|  '- logs/<alias>/<basename>/   # source de verite par exécution
|     |- <alias>_<basename>_<run_id>.csv
|     '- <alias>_<basename>_<run_id>_meta.csv
'- reports/                      # graphiques et CSV generes
```

Figure A.3 – Organisation usuelle des répertoires

### A.1.4 Lancer une campagne

La commande principale pour lancer une campagne est `maxsat-runner run`. Chaque option `-solver` doit contenir le motif `{inst}`, remplacé automatiquement par le chemin de l'instance. Le paramètre `-timeout-sec` permet d'imposer un temps limite ; en cas de dépassement, l'exécution est arrêtée avec un code de sortie 124.

```
maxsat-runner run \
  --solver "solverA=/chemin/solverA {inst}" \
  --solver "solverB=/chemin/solverB --old-format {inst}" \
  --instances data/instances/demo \
  --pattern .wcnf \
  --out data/runs \
  --timeout-sec 30
```

Figure A.4 – Exemple de campagne CLI

```
# Si un solveur depend d'un repertoire de travail particulier
maxsat-runner run \
  --solver "[cwd=/chemin/vers/bin] ./solver_exec {inst}" \
  --instances data/instances/demo \
  --pattern .wcnf \
  --out data/runs \
  --timeout-sec 30
```

Figure A.5 – Répertoire de travail spécifique par solveur

### A.1.5 Exécuter un seul run

La commande `run-one` exécute un unique couple solveur–instance. Elle est particulièrement adaptée à un usage sur cluster ou dans un script Slurm.

```
maxsat-runner run-one \
  --solver-alias solverA \
  --cmd "/chemin/solverA {inst}" \
  --instance data/instances/demo/exemple.wcnf \
  --out data/runs \
  --timeout-sec 30
```

Figure A.6 – Exécution d'un run unitaire

### A.1.6 Générer les statistiques

À partir des logs produits par les campagnes, la commande `stats` reconstruit ou recharge les fichiers globaux `trajectories.csv` et `summary.csv`, puis génère des rapports CSV et plusieurs figures. L'agrégation peut être faite par `solver_alias`, `solver_cmd` ou, lorsque cette information est disponible dans les données, `solver_tag`. Il est aussi possible de restreindre l'analyse à une instance, à une fenêtre temporelle donnée ou à un instant précis `t_at`. Des options permettent également d'activer ou non certaines sorties coûteuses, comme les figures par instance ou les statistiques finales complètes.

```
maxsat-runner stats \
  --runs data/runs \
  --out data/reports \
  --by solver_alias
```

Figure A.7 – Commande générale de statistiques

```
# Statistiques globales
maxsat-runner stats --runs data/runs --out data/reports

# Limiter l'analyse a la fenetre [0s, 10s]
maxsat-runner stats --runs data/runs --out data/reports --t-min 0 --t-max 10

# Snapshot a 5 secondes
maxsat-runner stats --runs data/runs --out data/reports --t-at 5.0

# Statistiques pour une instance precise
maxsat-runner stats --runs data/runs --out data/reports --instance tiny2

# Desactiver les figures par instance
maxsat-runner stats --runs data/runs --out data/reports --no-per-instance --no-per-
    instance-scores

# Desactiver les statistiques finales temporelles
maxsat-runner stats --runs data/runs --out data/reports --no-final-summary

# Activer les statistiques de replicas
maxsat-runner stats --runs data/runs --out data/reports --do-replicas-by-solver
```

Figure A.8 – Exemples de statistiques

```
reports/leaderboard.csv
reports/plot_leaderboard_wins.png
reports/plot_time_to_best_box.png

# sorties optionnelles selon les flags
reports/instances/plot_<instance>.png
reports/instances_scores/scores_<instance>.png
reports/average_scores_over_time.csv
reports/average_scores_over_time.png
reports/auc_scores.csv
reports/auc_scores_table.png
reports/replicas_by_solver.csv
reports/replicas_by_solver/replicas_<solver>.png
```

Figure A.9 – Principales sorties de la commande stats

### A.1.7 Clustering des instances

La commande `clusters` permet de regrouper les instances à partir de la similarité de leurs trajectoires. Les principales options concernent la métrique de comparaison, le nombre de clusters, la fenêtre temporelle, le nombre de points d'échantillonnage `T` et la stratégie d'échantillonnage temporel. Les métriques disponibles incluent notamment `spearman`, `pearson`, `cosine`, `l2`, `manhattan` et `dtw`.

```
maxsat-runner clusters \
  --runs data/runs \
  --out data/reports \
  --by solver_alias \
  --metric spearman \
  --k 3 \
  --t-min 0 --t-max 10 \
  --T 100 \
  --sampling log \
  --ratio 1.25
```

Figure A.10 – Exemple de clustering des instances

```
reports/distances.csv
reports/clusters_<k>.csv
reports/mst.png
```

Figure A.11 – Sorties principales de la commande clusters

### A.1.8 Lancer le serveur web

En complément de l'interface en ligne de commande, `MaxSAT Runner` peut être démarré en mode serveur pour exposer une API et une interface web.

```
maxsat-runner serve --host 127.0.0.1 --port 8000
```

Figure A.12 – Démarrage du serveur web

```
# Ouvrir ensuite dans un navigateur
http://127.0.0.1:8000
```

Figure A.13 – Accès à l'interface web

## A.2 Pseudocode détaillé d'EvalMaxSAT Anytime

Cette annexe présente un pseudocode synthétique de haut niveau pour *EvalMaxSAT Anytime*. Il ne remplace pas la description donnée au Chapitre 5, mais en résume les étapes essentielles sous une forme plus procédurale.

Le solveur maintient une affectation complète, satisfait d'abord les clauses dures lorsqu'elles sont réalisables, puis améliore progressivement la partie souple. Lorsque la recherche atteint un minimum local, il modifie temporairement certains poids heuristiques afin de faire réapparaître un *flip* favorable, puis oublie progressivement ces ajustements grâce à une mémoire circulaire.

**Algorithm 3** Schéma général d'EvalMaxSAT Anytime

```
1:  function EvalMaxSATAnytime(F, T)
2:      σ ← InitialHardSATSolution(F)
3:      initialiser les structures incrémentales
4:      σ* ← σ
5:      bestCost ← cost(σ)
6:      while temps écoulé < T do
7:          if au moins une clause dure est violée then
8:              choisir une variable de réparation x
9:              FlipAndUpdate(σ, x)
10:         else
11:             x ← GetGoodMove(σ)
12:             if x existe then
13:                 FlipAndUpdate(σ, x)
14:             else
15:                 choisir une clause souple violée C
16:                 x ← CreateGoodFlip(C)
17:                 FlipAndUpdate(σ, x)
18:             end if
19:         end if
20:         if cost(σ) < bestCost then
21:             σ* ← σ
22:             bestCost ← cost(σ)
23:             mémoriser le temps d'amélioration
24:             NotifyImprovement
25:         end if
26:         UpdateWeightHistoryRing
27:     end while
28:     return σ*
29: end function
```

Le cœur de la contribution proposée concerne la fonction CreateGoodFlip, appelée lorsque plus aucun mou-

vement strictement favorable n'est disponible.

**Algorithm 4** Création contrôlée d'un *flip* favorable

**function** CreateGoodFlip($C$)
    $m \leftarrow \max_{x \in C} s(x)$
    $\delta \leftarrow 1 - m$
    $\tilde{w}(C) \leftarrow \tilde{w}(C) + \delta$
    WeightHistoryRing.Push(id$(C), \delta$)
    **return** ArgMaxScoreInClause($C$)
**end function**

Cette règle garantit qu'au moins un littéral de la clause $C$ devient strictement favorable après l'ajustement. L'augmentation appliquée est minimale au sens où elle correspond exactement à la quantité nécessaire pour faire réapparaître un *flip* favorable.

Le mécanisme d'oubli progressif des poids heuristiques peut être résumé comme suit.

**Algorithm 5** Principe simplifié de `WeightHistoryRing`

**function** Push($clauseId, \delta$)
    enregistrer $(clauseId, \delta, cycleId)$ dans le buffer circulaire
    **if** condition de fin de cycle satisfaite **then**
        RemoveOldestCycle
        $cycleId \leftarrow cycleId + 1$
    **end if**
**end function**
**function** RemoveOldestCycle
    **for** chaque $(clauseId, \delta)$ du plus ancien cycle **do**
        RemoveWeightCallback($clauseId, \delta$)
    **end for**
**end function**

Ce mécanisme joue un rôle de régulation. Il permet au solveur de profiter temporairement des ajustements de poids utiles pour sortir d'un minimum local, sans laisser ces augmentations déformer durablement le

paysage heuristique.

## A.3 Structures de données principales

Cette section résume les principales structures de données utilisées dans l'implémentation d'EvalMaxSAT Anytime. L'objectif n'est pas de reproduire l'intégralité du code, mais de préciser les rôles algorithmiques des composants manipulés pendant la recherche.

### A.3.1 `AssignFormula`

La structure `AssignFormula` maintient l'état courant de l'affectation. Elle contient notamment :

- la valeur courante de chaque variable ;
- le compteur de satisfaction de chaque clause ;
- les ensembles de clauses dures et souples actuellement violées ;
- le coût objectif courant.

Pour chaque clause $C_i$, on maintient un compteur

$$\mathrm{satCount}(i) = \#\{\ell \in C_i \mid \ell \text{ est vrai sous } \sigma\}.$$

Ce compteur permet de savoir immédiatement si une clause est violée, satisfaite de façon critique (un seul littéral vrai), ou satisfaite par plusieurs littéraux.

Cette information est essentielle pour la mise à jour incrémentale des scores make/break après un *flip*.

### A.3.2 `FormulaManager`

`FormulaManager` constitue le composant central du solveur. Il gère :

- la formule WCNF ;
- les poids objectifs et les poids heuristiques ;
- les listes d'occurrences des littéraux ;
- la sélection des mouvements ;
- l'interaction avec `WeightHistoryRing`.

Il assure donc la cohérence entre la représentation de la formule, la logique de score et les ajustements dynamiques de poids.

### A.3.3 `MonScore`

`MonScore` maintient un score local par variable, ainsi qu'une structure de sélection rapide des variables ayant un score strictement positif. Le score d'une variable $x$ est calculé sous la forme :

$$s(x) = \sum_{i \in \mathrm{Make}(x)} \tilde{w}_i - \sum_{i \in \mathrm{Break}(x)} \tilde{w}_i,$$

où $\tilde{w}_i$ désigne le poids heuristique de la clause $i$.

Le composant permet donc :

- de mettre à jour efficacement les scores après un *flip* ;
- d'identifier rapidement les *flips* favorables ;
- d'éviter un balayage complet de toutes les variables à chaque itération.

### A.3.4 `WeightHistoryRing`

`WeightHistoryRing` est une mémoire circulaire destinée à stocker les augmentations temporaires de poids heuristiques. Chaque entrée correspond typiquement à un triplet de la forme :

$$(\texttt{clauseId}, \delta, \texttt{cycleId}).$$

Cette structure permet :

- d'enregistrer les modifications heuristiques récentes ;
- d'organiser ces modifications par cycles ;
- de supprimer progressivement les ajustements anciens.

Son rôle est donc de limiter le biais accumulé par les stratégies de pondération locale, en restaurant progressivement les poids heuristiques vers des valeurs plus proches de leur état initial.

### A.3.5 Listes d'occurrences

Pour chaque variable, le solveur maintient les listes des clauses où elle apparaît positivement et négativement. Ces listes sont cruciales, car lorsqu'une variable est retournée, seules les clauses qui contiennent cette variable peuvent voir leur état modifié.

Si l'on note $\deg(x)$ le nombre total d'occurrences de $x$ et $\neg x$ dans la formule, alors le coût pratique d'un

*flip* est essentiellement proportionnel à $\deg(x)$. Cette propriété explique pourquoi l'algorithme peut rester efficace sur des budgets de temps relativement courts.

## A.4 Remarques complémentaires sur l'implémentation

Les annexes présentées ici visent principalement à compléter le corps du mémoire en fournissant, sous une forme plus opérationnelle, un résumé du mode d'emploi de *MaxSAT Runner*, un pseudocode synthétique du solveur *EvalMaxSAT Anytime* et une présentation des principales structures de données utilisées dans l'implémentation.

Selon le niveau de détail souhaité, plusieurs prolongements pourraient être ajoutés dans une version ultérieure, par exemple des extraits de code illustrant les mises à jour incrémentales des compteurs de satisfaction, une documentation plus précise des paramètres influençant la gestion du `WeightHistoryRing`, ou encore des exemples détaillés de trajectoires de coût sur de petites instances. Dans le cadre du présent mémoire, les éléments retenus ont été limités aux composants les plus utiles pour la compréhension de la logique du solveur et de son instrumentation expérimentale.

**BIBLIOGRAPHIE**


Ansótegui, C., Bonet, M. L. et Levy, J. (2009). Solving (weighted) partial maxsat through satisfiability testing. Dans *Theory and Applications of Satisfiability Testing – SAT 2009*, volume 5584 de *Lecture Notes in Computer Science*, 427–440. Springer. http://dx.doi.org/10.1007/978-3-642-02777-2_39

Ansótegui, C., Bonet, M. L. et Levy, J. (2013). Sat-based maxsat algorithms. *Artificial Intelligence*, *196*, 77–105. http://dx.doi.org/10.1016/j.artint.2013.01.002

Arora, S. et Barak, B. (2009). *Computational Complexity : A Modern Approach*. Cambridge University Press.

Audemard, G. et Simon, L. (2009). Predicting learnt clauses quality in modern SAT solvers. Dans *Theory and Applications of Satisfiability Testing (SAT 2009)*, volume 5584 de *Lecture Notes in Computer Science*, 399–404. Springer.

Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A. et Protasi, M. (1999). *Complexity and Approximation : Combinatorial Optimization Problems and Their Approximability Properties*. Springer.

Avellaneda, F. (2021). A short description of the solver EvalMaxSAT. In F. Bacchus, J. Berg, M. Järvisalo, et R. Martins (dir.), *MaxSAT Evaluation 2021 : Solver and Benchmark Descriptions*, numéro B-2021-2 de Series of Publications B 10–11.

Berg, J., Jabs, C., Ihalainen, H. et Järvisalo, M. (2024a). Maxsat evaluation 2024. [Présentation]. Récupéré le 22 avril 2026 de https://maxsat-evaluations.github.io/2024/mse24-talk.pdf

Berg, J., Järvisalo, M., Martins, R., Niskanen, A. et Paxian, T. (2024b). *MaxSAT Evaluation 2024 : Solver and Benchmark Descriptions*. Rapport technique B-2024-2, University of Helsinki, Department of Computer Science

Biere, A., Heule, M., van Maaren, H. et Walsh, T. (dir.) (2009). *Handbook of Satisfiability*. IOS Press.

Cai, S. et Luo, C. (2018). SATLike : A deterministic algorithm for the incomplete MaxSAT problem. Dans *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI 2018)*, 1355–1361. http://dx.doi.org/10.24963/ijcai.2018/187

Cai, S., Luo, C., Fan, H. et Su, K. (2013). Improving walksat for random k-satisfiability problem with configuration checking and scoring functions. Dans *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*.

Calcul Québec (2025a). Rorqual – cluster and services description. Récupéré de https://metrix.rorqual.calculquebec.ca/

Calcul Québec (2025b). Rorqual : Un superordinateur attendu par la communauté de la recherche. Récupéré de https://www.calculquebec.ca/nouvelle/rorqual-un-superordinateur-attendu-par-la-communaute-de-la-recherche/

Chu, Y., Cai, S. et Luo, C. (2023). Nuwls : Improving local search for (weighted) partial maxsat by new weighting techniques. Dans *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, 3915–3923. http://dx.doi.org/10.1609/aaai.v37i4.25505. Récupéré de https://doi.org/10.1609/aaai.v37i4.25505

Cook, S. A. (1971). The complexity of theorem-proving procedures. Dans *Proceedings of the Third Annual ACM Symposium on Theory of Computing (STOC '71)*, 151–158. ACM. http://dx.doi.org/10.1145/800157.805047

Davies, J. et Bacchus, F. (2013). Exploiting the power of MIP solvers in MaxSAT. Dans *Theory and Applications of Satisfiability Testing (SAT 2013)*, volume 7962 de *Lecture Notes in Computer Science*, 166–181. Springer. http://dx.doi.org/10.1007/978-3-642-39071-5_13

Davis, M., Logemann, G. et Loveland, D. (1962). A machine program for theorem-proving. *Communications of the ACM*, *5*(7), 394–397. http://dx.doi.org/10.1145/368273.368557

Davis, M. et Putnam, H. (1960). A computing procedure for quantification theory. *Journal of the ACM*, *7*(3), 201–215. http://dx.doi.org/10.1145/321033.321034

Digital Research Alliance of Canada (2024). Acknowledging the alliance. Récupéré de https://www.alliancecan.ca/our-services/advanced-research-computing/acknowledging-alliance

Dolan, E. D. et Moré, J. J. (2002). Benchmarking optimization software with performance profiles. *Mathematical Programming*, *91*(2), 201–213. http://dx.doi.org/10.1007/s101070100263

Eén, N. et Sörensson, N. (2004). An extensible SAT-solver. Dans *Theory and Applications of Satisfiability Testing (SAT 2003)*, volume 2919 de *Lecture Notes in Computer Science*, 502–518. Springer. http://dx.doi.org/10.1007/978-3-540-24605-3_37

Fu, Z. et Malik, S. (2006). On solving the partial Max-SAT problem. Dans *Theory and Applications of Satisfiability Testing (SAT)*, 252–265. http://dx.doi.org/10.1007/11814948_25

Fukunaga, A. S. (1997). Variable-selection heuristics in local search for sat. Dans *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI)*.

Garey, M. R. et Johnson, D. S. (1979). *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W. H. Freeman.

Hickey, R. et Bacchus, F. (2022). Large neighbourhood search for anytime maxsat solving. Dans *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence (IJCAI 2022)*, 1818–1824. http://dx.doi.org/10.24963/ijcai.2022/253

Hoos, H. H. et Stützle, T. (2004). *Stochastic Local Search : Foundations and Applications*. Morgan Kaufmann.

Ignatiev, A., Morgado, A. et Marques-Silva, J. (2019). RC2 : an efficient MaxSAT solver. *Journal on Satisfiability, Boolean Modeling and Computation*, *11*(1), 53–64. http://dx.doi.org/10.3233/SAT190116

Jiang, M. et Chen, Y. (2024). Nuwls-c-ibr : Solver description. Dans *MaxSAT Evaluation 2024 : Solver and Benchmark Descriptions*, p. 22. University of Helsinki, Department of Computer Science. Récupéré de http://hdl.handle.net/10138/584878

Jin, M., He, K., Zheng, J., Xue, J. et Chen, Z. (2024). Combining BandMaxSAT and FPS with SPB-MaxSAT-c. In *MaxSAT Evaluation 2024 : Solver and Benchmark Descriptions* 23–24. University of Helsinki, Department of Computer Science

Karp, R. M. (1972). Reducibility among combinatorial problems. In *Complexity of Computer Computations* 85–103. Plenum.

Krentel, M. W. (1988). The complexity of optimization problems. *Journal of Computer and System Sciences*, *36*(3), 490–509. http://dx.doi.org/10.1016/0022-0000(88)90039-6

Li, C.-M. et Manyà, F. (2009). Maxsat, hard and soft constraints. In A. Biere, M. Heule, H. van Maaren, et T. Walsh (dir.), *Handbook of Satisfiability*, volume 185 de *Frontiers in Artificial Intelligence and Applications* 613–631. IOS Press

Lübke, O. et Berg, J. (2025). Sls-enhanced core-boosted linear search for anytime maximum satisfiability. Dans *31st International Conference on Principles and Practice of Constraint Programming (CP 2025)*, volume 340 de *Leibniz International Proceedings in Informatics (LIPIcs)*, 28 :1–28 :20., Dagstuhl, Germany. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. http://dx.doi.org/10.4230/LIPIcs.CP.2025.28. Récupéré de https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.CP.2025.28

Luo, C., Cai, S., Wu, W. W. et Su, K. (2014). Double configuration checking in stochastic local search for satisfiability. Dans *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*.

Marques-Silva, J. P. et Sakallah, K. A. (1999). GRASP : A search algorithm for propositional satisfiability. Dans *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 220–227. http://dx.doi.org/10.1109/ICCAD.1999.810752

Martins, R., Manquinho, V. M. et Lynce, I. (2014). Open-wbo : A modular MaxSAT solver. Dans *Theory and Applications of Satisfiability Testing (SAT)*, 438–445. http://dx.doi.org/10.1007/978-3-319-09284-3_33

MaxSAT Evaluation 2024 (2024). Benchmark sets. Récupéré de https://maxsat-evaluations.github.io/2024/benchmarks.html

Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L. et Malik, S. (2001). Chaff : Engineering an efficient sat solver. Dans *Proceedings of the 38th Design Automation Conference (DAC 2001)*, 530–535. ACM. http://dx.doi.org/10.1145/378239.379017

Nadel, A. (2024). TT-Open-WBO-Inc : An efficient anytime maxsat solver. *Journal on Satisfiability, Boolean Modeling and Computation*, *15*(1), 1–7. http://dx.doi.org/10.3233/SAT-231504. Récupéré le 22 avril 2026 de https://doi.org/10.3233/SAT-231504

Papadimitriou, C. H. (1994). *Computational Complexity*. Addison-Wesley.

Papadimitriou, C. H. et Yannakakis, M. (1991). Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, *43*(3), 425–440. http://dx.doi.org/10.1016/0022-0000(91)90023-X

Pham, D. N., Duong, T.-T. et Sattar, A. (2012). Trap avoidance in local search using pseudo-conflict learning. Dans *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*.

Pham, D. N., Lynce, I., Sattar, A. *et al.* (2005). Old resolution meets modern sls. Dans *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI)*.

Saikko, P., Berg, J. et Järvisalo, M. (2016). LMHS : A SAT-IP hybrid MaxSAT solver. Dans *Theory and Applications of Satisfiability Testing (SAT 2016)*, volume 9710 de *Lecture Notes in Computer Science*, 539–546. Springer. http://dx.doi.org/10.1007/978-3-319-40970-2_34

Sall, A. (2026a). EvalMaxSAT Anytime. [Dépôt logiciel GitHub]. Récupéré le 22 avril 2026 de https://github.com/FlorentAvellaneda/EvalMaxSAT_anytime

Sall, A. (2026b). MaxSAT Runner. [Dépôt logiciel GitHub]. Récupéré le 22 avril 2026 de https://github.com/Amzidou/Maxsat_logger

Salvador, S. et Chan, P. K. (2007). Toward accurate dynamic time warping in linear time and space. *Intelligent Data Analysis*, *11*(5), 561–580. http://dx.doi.org/10.3233/IDA-2007-11508

Selman, B., Kautz, H. A. et Cohen, B. (1994). Noise strategies for improving local search. Dans *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI 1994)*, 337–343. AAAI Press.

Selman, B., Levesque, H. J. et Mitchell, D. G. (1992). A new method for solving hard satisfiability problems. Dans *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI 1992)*, 440–446. AAAI Press.

Sipser, M. (2012). *Introduction to the Theory of Computation* (3 éd.). Cengage Learning.

Soos, M., Nohl, K. et Castelluccia, C. (2009). Extending SAT solvers to cryptographic problems. Dans *Theory and Applications of Satisfiability Testing (SAT 2009)*, volume 5584 de *Lecture Notes in Computer Science*, 244–257. Springer. http://dx.doi.org/10.1007/978-3-642-02777-2_24

Thornton, J., Pham, D. N., Bain, S. et Jr., V. F. (2004). Additive versus multiplicative clause weighting for sat. Dans *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI)*.

Vazirani, V. V. (2001). *Approximation Algorithms*. Springer. http://dx.doi.org/10.1007/978-3-662-04565-7

Zheng, J. *et al.* (2024a). A review of clause weighting techniques in maxsat local search. *arXiv preprint arXiv :2401.10589*.

Zheng, J., Chen, Z., Li, C.-M. et He, K. (2024b). Rethinking the soft conflict pseudo boolean constraint on maxsat local search solvers. Dans *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence (IJCAI-24)*, 1989–1997. http://dx.doi.org/10.24963/ijcai.2024/220. Récupéré de https://doi.org/10.24963/ijcai.2024/220

Zheng, J., He, K. et Zhou, J. (2023). Farsighted probabilistic sampling : A general strategy for boosting local search maxsat solvers. Dans *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, 4132–4139. http://dx.doi.org/10.1609/aaai.v37i4.25529. Récupéré de https://doi.org/10.1609/aaai.v37i4.25529

Zheng, J., He, K., Zhou, J., Jin, Y., Li, C.-M. et Manyà, F. (2022). Bandmaxsat : A local search maxsat solver with multi-armed bandit. Dans *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence (IJCAI-22)*, 1901–1907. http://dx.doi.org/10.24963/ijcai.2022/264. Récupéré de https://doi.org/10.24963/ijcai.2022/264

Zilberstein, S. (1996). Using anytime algorithms in intelligent systems. *AI Magazine*, *17*(3), 73–83.