LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

CHAIR OF THEORETICAL COMPUTER SCIENCE AND THEOREM PROVING

Strategies and optimization approaches for the exchange of clauses in SAT Solvers

Varol Cagdas Tok

Bachelor's Thesis
in Computer Science plus Computerlinguistik

Supervisor: Prof. Dr. Jasmin Blanchette

Advisor: Lydia Kondylidou

Submission Date: July 3, 2024

Disclaimer

I confirm that this thesis type is my own work and I have documented all sources and material used.

Munich, July 3, 2024 Varol Cagdas Tok

Acknowledgments

I would like to thank my advisor, Lydia Kondylidou, not only for her support and knowledge but also for her insightful ideas, consistent motivation, and genuine interest in my work, which played a crucial role in developing my ideas further. Her kindness and dedication have been a source of inspiration, making this journey much more rewarding. Additionally, I am grateful to Prof. Dr. Jasmin Blanchette for accepting me to write this thesis, providing me with the opportunity to pursue my academic goals under his supervision.

Abstract

In the domain of parallel SAT solving, the optimization of clause exchange and management emerges as a crucial factor in enhancing solver efficiency and addressing the challenges of increasingly complex problems. This thesis investigates the implementation of dynamic mechanisms for managing clause exchanges in parallel SAT solvers, with a focus on the Glucose SAT Solver's parallel version, Glucose Syrup. It addresses the challenges posed by extensive clause databases in complex SAT problems. The study introduces two primary methods: a clause size-based filtering strategy to limit the transmission of longer clauses, combined with a dynamic threshold management system that adaptively adjusts size thresholds for clause acceptance in line with the solver's evolving needs. Additionally, the work explores dynamic adjustments to the variable decay factor based on performance metrics, aimed at enhancing solver efficiency. Given the high frequency of clause exchanges, adding extra layers of complex criteria could detrimentally impact the overall performance of the solver. These methods seek to balance efficiency and flexibility in clause management, with the aim to contribute to ongoing efforts in SAT solver optimization.

Keywords: Parallel SAT Solving, Clause Management, Glucose Syrup Solver, Dynamic Threshold Management, Clause Size-based Filtering, Variable Decay Adjustment, Conflict-Driven Clause Learning (CDCL), Boolean Satisfiability, Computational Efficiency, Solver Optimization.

Kurzfassung

Im Bereich des parallelen SAT-Solvings stellt die Optimierung des Austauschs und des Managements von Klauseln einen entscheidenden Faktor dar, um die Effizienz des Solvers zu steigern und die Herausforderungen zunehmend komplexer Probleme zu bewältigen. Diese Arbeit untersucht die Implementierung dynamischer Mechanismen zur Verwaltung von Klauselaustauschen in parallelen SAT-Solvern, mit einem besonderen Fokus auf die parallele Version des Glucose SAT-Solvers, Glucose Syrup. Sie adressiert die Herausforderungen, die durch umfangreiche Klauseldatenbanken in komplexen SAT-Problemen entstehen. Die Studie stellt zwei primäre Methoden vor: eine auf der Klauselgröße basierende Filterstrategie, um die Übertragung längerer Klauseln zu begrenzen, kombiniert mit einem dynamischen Schwellenwertmanagement-System, das die Größenschwellenwerte für die Klauselakzeptanz adaptiv an die sich entwickelnden Bedürfnisse des Solvers anpasst. Zusätzlich erforscht die Arbeit dynamische Anpassungen des Variable Decays basierend auf Leistungsmetriken, um die Effizienz des Solvers zu verbessern. Wegen der hohen Frequenz des Klauselaustauschs könnte das Hinzufügen zusätzlicher Schichten komplexer Kriterien die Gesamtleistung des Solvers negativ beeinflussen. Diese Methoden zielen darauf ab, ein Gleichgewicht zwischen Effizienz und Flexibilität im Management von Klauseln zu finden und somit zu den fortlaufenden Optimierungsbestrebungen bei SAT-Solvern beizutragen.

Keywords: Parallel SAT Solving, Clause Management, Glucose Syrup Solver, Dynamic Threshold Management, Clause Size-based Filtering, Variable Decay Adjustment, Conflict-Driven Clause Learning (CDCL), Boolean Satisfiability, Computational Efficiency, Solver Optimization.

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Overview . . . 1

2 Preliminaries **2**
- 2.1 SAT Solvers . . . 2
  - 2.1.1 Boolean Expressions . . . 2
- 2.2 Normal Forms . . . 2
  - 2.2.1 Conjuctive Normal Form . . . 2

3 CDCL SAT Solvers **3**
  - 3.0.1 Variable Assignment in CDCL Algorithm . . . 5
  - 3.0.2 Unit Propagation . . . 5
  - 3.0.3 Conflict Detection and Analysis . . . 5
  - 3.0.4 Clause Learning . . . 5
  - 3.0.5 Non-Chronological Backtracking . . . 5
  - 3.0.6 Iteration and Solution . . . 6
- 3.1 Parallel SAT Solvers . . . 8
  - 3.1.1 Portfolio-Based Solvers . . . 9
  - 3.1.2 Divide-and-Conquer Solvers . . . 10
  - 3.1.3 Challenges in Parallel SAT Solving . . . 11
  - 3.1.4 Advancements and Applications . . . 13
- 3.2 Glucose . . . 13

4 Strategies for Clause Management in Parallel SAT Solvers **14**
- 4.1 Clause size based approach . . . 14
  - 4.1.1 Dynamic Threshold Management . . . 15
- 4.2 Dynamic var decay Adjustment . . . 17
  - 4.2.1 Conflict Analysis and Adjustment Preparation . . . 17

5 Performance and Measurements **19**
- 5.1 Tests and test environment . . . 19
- 5.2 Analysis . . . 19

6 Code and Data **21**

7 Analysis and Conclusion **21**

8 Future Work **22**

Bibliography **25**

1 Introduction

1.1 Motivation

In the domain of computational problem-solving, SAT solvers, short for Boolean Satisfiability solvers, stand out as critical tools in determining the satisfiability of Boolean formulas. Their importance is highlighted by their extensive applications across various fields, including artificial intelligence, software verification, and electronic design automation (1). These solvers are renowned for their ability to efficiently navigate complex logical problems, a skill that is increasingly vital in the context of modern technological challenges. As we progress further into the digital era, the role of SAT solvers in resolving intricate logical tasks continues to grow, underlining their significance in the advancement of computational methodologies and technologies.

The recent progress in the development of SAT solvers is marked by a number of significant improvements. Central to these are the advancements in heuristic algorithms, the integration of advanced machine learning techniques, and the establishment of flexible, scalable data structures (2). Together, these developments have substantially improved the performance of SAT solvers. There has been a clear increase in their processing speed, and they are now more capable of efficiently managing larger and more complex problems (3).

The introduction of parallel SAT solving represents a major advancement in this field. By running several instances of SAT solvers simultaneously, this method greatly enhances the problem-solving ability, surpassing the capabilities of conventional SAT solving. However, this move towards parallel processing presents its own set of challenges, especially in terms of efficient information sharing – specifically, the sharing of clauses – among the solver instances. Effective management of this clause sharing is vital, as random or disorganized distribution can create processing delays, thereby negating the advantages of parallelism (4).

This Bachelor thesis explores the optimization of clause exchange strategies in parallel SAT solvers, aiming to enhance their efficiency and effectiveness. By refining the selection and sharing of clauses, the research focuses on preventing information overload while ensuring the dissemination of useful information. The optimization techniques applied include a clause size-based approach, dynamic threshold management, and dynamic variable decay adjustment. These strategies are designed to adjust the solver's behavior dynamically, favoring the exchange of beneficial clauses and improving decision-making processes.

The application of these optimizations to a modified version of the Glucose SAT solver yielded positive results, particularly in solving problems that previously led to timeouts. This improvement underscores the potential of tailored clause exchange strategies to significantly boost the problem-solving capabilities of parallel SAT solvers, supporting their use in more complex and larger-scale problems.

1.2 Overview

This thesis provides a detailed study on improving clause exchange in parallel SAT solvers. Chapter 2 establishes a foundational understanding of key concepts such as Boolean expressions and Normal Forms. Chapter 3 continues to build the foundation by discussing Conflict-Driven Clause Learning (CDCL), parallel SAT solving, and details on Glucose and its parallel version, Glucose Syrup. Chapter 4 is dedicated to developing methods that enhance Glucose Syrup's performance, focusing

particularly on clause exchange strategies. Chapters 5 through 7 encompass the empirical research phase, covering performance metrics examination, and analysis of experimental results to determine the effectiveness of the strategies. Chapter 8 offers a perspective on future research directions and possibilities for further advancements in the field, based on the conclusions of this study. Overall, this thesis presents a structured examination into improving clause exchange in parallel SAT solvers, concluding with insights for ongoing and future research in this evolving field.

2 Preliminaries

2.1 SAT Solvers

This section introduces SAT Solvers and explains their properties. Especially, it provides some background knowledge on boolean expressions and highlights the most important aspects of SAT solvers and their development over the years. Various examples are also presented to further explain the usage and importance of SAT solvers.

The SAT, or Boolean Satisfiability Problem, challenges us to find a truth assignment for variables that will satisfy a given Boolean formula, representing a cornerstone in computational theory as a benchmark NP-complete problem (5). The exploration of SAT is not just an academic exercise; it has profound implications for computational complexity and practical computation. SAT solvers emerge as critical tools in this context, optimizing logical operations and thus impacting various fields, including circuit design and algorithmic research. Their development and refinement embody the practical innovation and theoretical exploration at the heart of computational logic.

2.1.1 Boolean Expressions

At the core of SAT problems lie Boolean expressions, constructed from variables and logical operators (AND, OR, NOT) that can assume true or false values. The operations of these expressions adhere to well-defined rules. For example, the AND operation yields true only when both operands are true, encapsulating the essence of conjunction in logical reasoning. Similarly, the OR operation signifies disjunction, requiring at least one operand to be true, while the NOT operation represents negation by inverting the truth value of its operand.

The manipulation and evaluation of Boolean expressions by SAT solvers are fundamental steps in determining the satisfiability of complex problems, showcasing the direct application of basic computational logic in advanced problem-solving scenarios.

2.2 Normal Forms

2.2.1 Conjuctive Normal Form

Conjunctive Normal Form (CNF) is an approach to Boolean logic that expresses formulas as conjunctions of clauses or terms with an AND or OR. Each clause, connected by a conjunction (AND), must be either a literal or contain a disjunction (OR) operator. Literals are seen in CNF as conjunctions of literal clauses and conjunctions that happen to have a single clause. It is possible to convert statements into CNF that are written in another form, such as disjunctive normal form. This structured format—a conjunction of clauses, with each clause being a disjunction of

literals—simplifies and unifies the expressions, making them more amenable to the algorithms employed by SAT solvers.

The transition to Conjunctive Normal Form (CNF) marks a critical juncture in SAT solving, connecting the theoretical underpinnings of Boolean expressions with the practical mechanisms of SAT solvers. Adopting CNF does more than just streamline the problem format; it leverages the foundational principles of Boolean logic to enhance the solvers' efficiency. By transforming complex Boolean formulas into a consistent and structured form, SAT solvers can apply standardized techniques more effectively, demonstrating the seamless integration of basic logical constructs into sophisticated computational tools.

$$\text{CNF} = (\text{Clause}_1) \wedge (\text{Clause}_2) \wedge \ldots$$

Where:

- Each Clause is a disjunction of literals: $(\text{Literal}_{i1} \vee \text{Literal}_{i2} \vee \ldots)$.
- A Literal is either a variable or its negation.

An example:

$$(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3 \vee \neg x_4) \wedge \ldots$$

Here:

- x_1, x_2, x_3 are variables.
- $\neg x_2, \neg x_4$ are negated variables (negations).
- $(x_1 \vee \neg x_2)$ and $(x_2 \vee x_3 \vee \neg x_4)$ are clauses.

3 CDCL SAT Solvers

The concept of CDCL (Conflict-Driven Clause Learning) SAT solvers is primarily inspired by Davis–Putnam–Logemann–Loveland (DPLL) solvers (1). As a result, a reasonable knowledge of the organization of DPLL is assumed. In order to offer a detailed account of CDCL SAT solvers, several concepts have to be introduced, which serve to formalize the operations implemented by any DPLL SAT solver. DPLL corresponds to backtrack search, where each step selects a variable and a propositional value for branching purposes. With each branching step, two values can be assigned to a variable, either 0 or 1. Branching corresponds to assigning the chosen value to the chosen variable. Afterward, the logical consequences of each branching step are evaluated. Each time an unsatisfied clause (i.e., a conflict) is identified, backtracking is executed. Backtracking corresponds to undoing branching steps until an unflipped branch is reached. Backtracking will undo this branching step when both values have been assigned to the selected variable at a branching step. If, for the first branching step, both values have been considered, and backtracking undoes this first branching step, then the CNF formula can be declared unsatisfiable. This kind of backtracking is called chronological backtracking.
DPLL is characterized by its methodical approach:

1. **Variable Assignment:** Select an unassigned variable and assign it a truth value (true or false). This decision is heuristic-driven, aiming to simplify the formula as much as possible with each choice.

2. **Unit Propagation:** If a clause becomes a unit clause (i.e., all but one of its literals are false), the remaining literal must be made true. This automatic assignment helps reduce the formula's complexity, moving the solver closer to a solution.

3. **Conflict Detection:** If a conflict is encountered (a clause becomes unsatisfied), the algorithm backtracks to the last decision point to try a different assignment. Conflict detection is crucial for avoiding futile paths and ensuring the algorithm does not pursue unsolvable configurations.

4. **Backtracking:** Upon conflict, DPLL reverts to earlier assignments to explore alternate paths in the search space. This step is iterative, allowing the solver to systematically explore different combinations of variable assignments (6).

Algorithm 1 DPLL Algorithm Pseudocode

```
Result: Either SAT or UNSAT
while not all variables assigned do
    if unit propagation returns conflict then
        return UNSAT
    end if
    x ← choose splitting var
    val ← choose initial assignment for x
    create new decision level with x = val
    while unit propagation returns conflict do
        if decision level == 0 then
            return UNSAT
        end if
        backtrack and set splitting variable for previous decision level to be the negation of the original choice
    end while
end while
return SAT
```

While DPLL provided a robust framework for SAT solving, its performance on complex instances was hindered by linear backtracking and the lack of a method to utilize the insights gained from conflicts, limiting its effectiveness in navigating extensive search spaces.

CDCL SAT solvers, built upon DPLL's foundation by incorporating key innovations such as non-chronological backtracking and clause learning. By intelligently analyzing conflicts to learn new clauses, CDCL solvers avoid repeating the same search paths that led to conflicts, substantially narrowing the search space. The introduction of clause learning, alongside the strategic choice of backtracking points, enhances the solver's efficiency, enabling it to address more complex SAT problems. This evolution from DPLL to CDCL has significantly transformed SAT solving practices, broadening their applicability and effectiveness in fields like model checking, bioinformatics, and cryptography. The CDCL algorithm follows a structured approach involving several key steps:

3.0.1 Variable Assignment in CDCL Algorithm

In the CDCL algorithm, the first step involves assigning values to variables, a foundational aspect borrowed from the DPLL algorithm. This assignment is critical as it sets the stage for the subsequent logical deductions and operations that the algorithm performs. The choice of which variable to assign and what value to assign it is guided by various heuristics designed to optimize the solver's efficiency.

3.0.2 Unit Propagation

Following each variable assignment, the CDCL algorithm undertakes unit propagation. This process entails the evaluation of clauses within the problem to identify any that become unit clauses—those with only one remaining unassigned literal. The value of this unassigned literal is then determined in such a manner as to satisfy the clause. Unit propagation is a powerful tool in the CDCL, serving to streamline the problem by systematically reducing the search space and eliminating inconsistent paths early in the solving process.

3.0.3 Conflict Detection and Analysis

A pivotal moment in the CDCL process is the detection of conflicts, which are instances where the current assignments lead to an unsatisfied clause. The occurrence of a conflict triggers a detailed analysis to pinpoint the series of decisions and unit propagations that culminated in the conflict. This step is not merely about identifying that a conflict has occurred but involves a deep dive into the causal chain, laying the groundwork for learning from the conflict and avoiding similar pitfalls in future iterations.

3.0.4 Clause Learning

Conflict analysis leads to the discovery of new clauses. Each learned clause captures a specific set of variable assignments that should be avoided to prevent the same conflict from happening again. When these clauses are added to the solver's database, CDCL solvers broaden their understanding of the problem landscape. This insight helps the solver avoid going down paths that lead to conflicts (6).

3.0.5 Non-Chronological Backtracking

Distinct from traditional backtracking, where the solver would retreat stepwise to the most recent decision, CDCL employs non-chronological backtracking. This strategy involves jumping back not just to the previous decision but to a more strategically advantageous point in the decision tree. This leap is informed by the insights gained from the learned clause, aiming to revisit a juncture where applying the new knowledge can avert the previously encountered conflict. Such targeted backtracking significantly enhances the efficiency of the search process, enabling a more focused and expedient navigation through the solution space.

3.0.6 Iteration and Solution

The CDCL algorithm is inherently iterative, cycling through the steps of decision-making, unit propagation, conflict detection and analysis, clause learning, and non-chronological backtracking. This cycle repeats, gradually converging on a solution or establishing the problem's unsatisfiability. Each iteration refines the solver's understanding of the problem, progressively closing in on a viable solution by eliminating inconsistencies and leveraging the cumulative insights gained through clause learning and strategic backtracking.

Example 2.2.1.

1. $(\neg x_1 \vee x_2)$
2. $(x_1 \vee x_3)$
3. $(\neg x_3 \vee x_4)$
4. $(\neg x_2 \vee \neg x_4)$
5. $(x_4 \vee x_5)$
6. $(\neg x_2 \vee \neg x_5)$

Variable Assignment:

- Initially, assign $x_1 = \text{True}$ based on a heuristic decision.

Unit Propagation:

- From $x_1 = \text{True}$, unit propagation leads to $x_2 = \text{True}$ because of clause 1 $(\neg x_1 \vee x_2)$.
- With $x_2 = \text{True}$:
  - Clause 6 $(\neg x_2 \vee \neg x_5)$ triggers unit propagation, leading to $x_5 = \text{False}$ to satisfy the clause.
  - Because of $x_2 = \text{True}$, clause 4 $(\neg x_2 \vee \neg x_4)$ necessitates $x_4 = \text{False}$ to satisfy the clause.
  - With $x_4 = \text{False}$, clause 5 $(x_4 \vee x_5)$ forces $x_5 = \text{True}$ for satisfaction, leading to a direct conflict.

Conflict Detection and Analysis:

- The assignment of $x_5 = \text{True}$ contradicts the earlier propagation that set $x_5 = \text{False}$, resulting in a conflict.

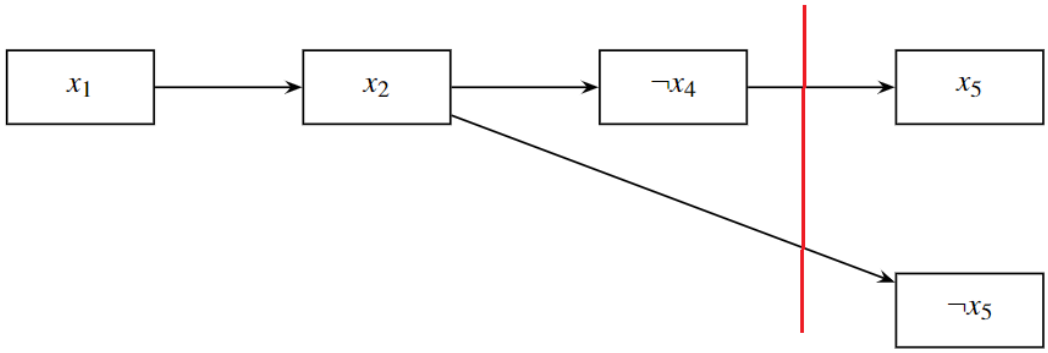

Figure 1: The implication graph with conflict cut.

Clause Learning:

- The learned clause from analyzing the conflict is $(\neg x_2 \vee x_4)$. This clause reflects the contradiction between x_2 and x_4's assignments leading to the conflict.

Non-Chronological Backtracking:

- Given the learned clause, non-chronological backtracking revisits the initial decision of $x_1 = \text{True}$.
- The solver changes x_1's value to False, aiming to resolve the underlying conflict efficiently.

Resolution: After non-chronological backtracking and setting x_1 to false, the variable assignments are reevaluated as follows:

- Setting $x_1 = \text{False}$ implies $x_3 = \text{True}$ due to clause 2 $(x_1 \vee x_3)$.
- With $x_3 = \text{True}$, clause 3 $(\neg x_3 \vee x_4)$ implies x_4 should be set to true.
- Also, $x_4 = \text{True}$ means x_2 must be set to false to satisfy clause 4 $(\neg x_2 \vee \neg x_4)$.
- Since $x_4 = \text{True}$, clause 5 $(x_4 \vee x_5)$ is satisfied regardless of x_5's value.
- With $\neg x_2 = \text{True}$, clause 6 $(\neg x_2 \vee \neg x_5)$ is satisfied regardless of x_5's value.

This reevaluation leads to a satisfying assignment for all clauses, thus resolving the conflict and satisfying the formula.

In summary, CDCL SAT solvers represent a significant leap in the field of SAT solving, combining the foundational principles of the DPLL algorithm with advanced techniques like clause

Algorithm 2 CDCL Algorithm Pseudocode

```
1: function CDCL(φ, ν)
2:     if (UnitPropagation(φ, ν) == CONFLICT) then
3:         return UNSAT
4:     end if
5:     d ← 0
6:     while (¬AllVariablesAssigned(φ, ν)) do
7:         (x, ν) ← PickBranchingVariable(φ, ν)                ▷ Decide stage
8:         d ← d + 1                                           ▷ Increment decision level
9:         ν ← ν ∪ {(x, ν)}
10:        if (UnitPropagation(φ, ν) == CONFLICT) then         ▷ Deduce stage
11:            β = ConflictAnalysis(φ, ν)                      ▷ Diagnose stage
12:            if (β < 0) then
13:                return UNSAT
14:            else
15:                Backjump(φ, ν, β)
16:                d = β                                       ▷ Decrement decision level due to backjumping
17:            end if
18:        end if
19:    end while
20:    return SAT
21: end function
```

learning and non-chronological backtracking. This amalgamation of strategies enables CDCL solvers to efficiently tackle complex SAT problems, making them a critical tool in computational problem-solving. The ongoing enhancements and adaptations in CDCL continue to open new avenues in the realm of algorithmic problem-solving.

3.1 Parallel SAT Solvers

Parallel SAT solving has emerged as a critical development in computational problem-solving, specifically aimed at addressing the complexities of Boolean Satisfiability (SAT) problems. This approach harnesses the power of concurrent processing across multiple processors to efficiently solve increasingly intricate SAT problems.

Parallel SAT solving was developed in response to the computational challenges posed by complex SAT problems, which often surpass the capabilities of traditional single-threaded solvers. The advancement of parallel SAT solving has paralleled technological developments in computing hardware, especially the advent of multi-core processors and the growth of distributed computing networks. These advancements have facilitated the effective implementation of parallel processing in SAT solving (7).

To effectively illustrate the efficiency of parallel SAT solvers, consider the scenario of solving a complex cryptographic puzzle, such as cracking a difficult encryption algorithm. Traditional SAT solvers, operating on a single processor, might take an impractically long time to navigate through the immense search space of possible solutions. However, a parallel SAT solver, dividing the task

among multiple processors, can explore various parts of the search space simultaneously. This collaborative effort significantly reduces the overall solving time, making it possible to crack the encryption in a fraction of the time it would take a single-threaded solver. Parallel SAT solvers can be broadly classified into two main categories:

3.1.1 Portfolio-Based Solvers

Portfolio-based solvers involve running multiple instances of SAT solvers in parallel. These instances can either be configured similarly or with different heuristics and settings. The diversity in solver configurations can be beneficial for tackling a wide range of problem types. These instances work independently on the same problem, and the process concludes when any one of the instances finds a solution (4). This approach is known for its effectiveness in various computational scenarios.

Consider a SAT problem P and n solver instances $S_1, S_2, \ldots, S_n$, where each instance S_i is configured differently. Let A_i denote a possible solution (assignment) found by solver S_i. The portfolio-based approach aims to find:

$$\text{Find } A_i \text{ such that } P(A_i) = \text{true, for any } i \in \{1, 2, \ldots, n\} \tag{1}$$

The process concludes successfully if:

$$\exists i \in \{1, 2, \ldots, n\} : P(A_i) = \text{true} \tag{2}$$

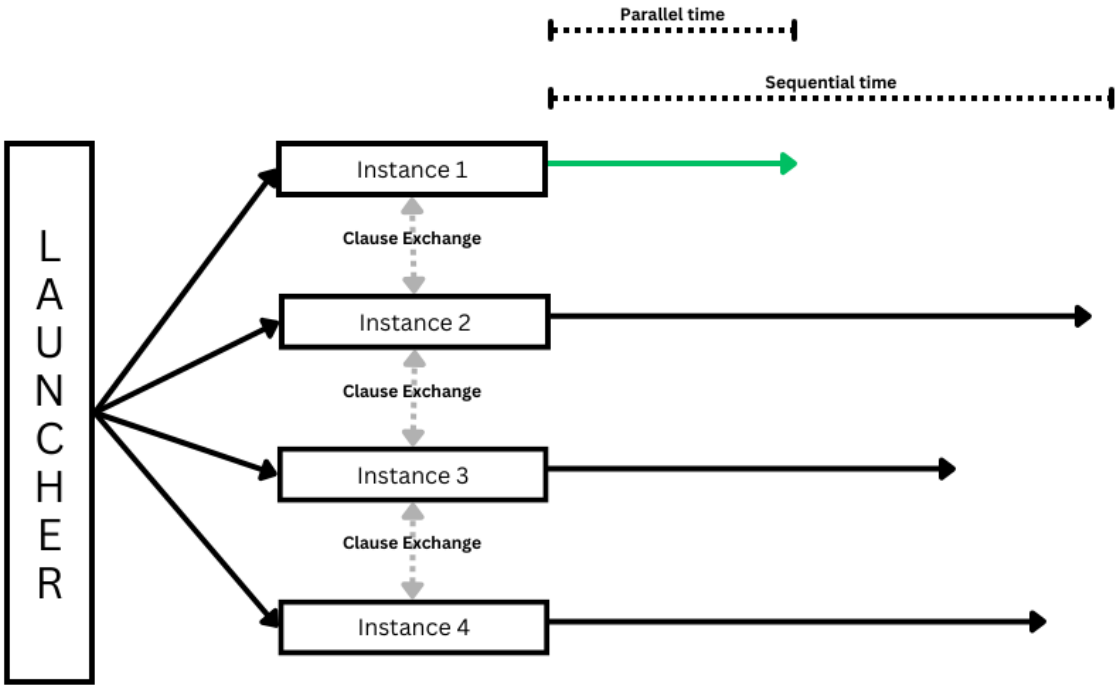

Figure 2: Portfolio-based SAT Solving (4).

3.1.2 Divide-and-Conquer Solvers

Divide-and-conquer solvers strategically partition a SAT problem into smaller, manageable sub-problems and distribute them across different processors (8). Each processor, then, dedicates its computational resources to solving its assigned sub-problem. The key to success in this approach lies in the careful construction of these sub-problems. This allows for a conclusive determination of the original problem's satisfiability based on the satisfiability of its sub-problems.

Let a SAT problem P be expressed in terms of variables $X = \{x_1, x_2, \ldots, x_n\}$. The divide-and-conquer method decomposes P into m sub-problems $P_1, P_2, \ldots, P_m$, with each sub-problem P_j containing a subset of the variables from X. The primary objective is to identify a set of variable assignments that satisfy all sub-problems concurrently, which, by extension, would satisfy the original problem:

$$P(A) = \bigwedge_{j=1}^{m} P_j(A_j) = \text{true} \tag{3}$$

This logical conjunction implies that P is satisfiable if there exists a combination of assignments $\{A_1, A_2, \ldots, A_m\}$ such that every sub-problem P_j is satisfied by its corresponding assignment A_j. However, this assertion is contingent upon the correct construction of the sub-problems: they must be independent and collectively exhaustive, without overlooking any interdependencies between the variables across different sub-problems. Achieving this often requires sophisticated partitioning strategies to ensure the integrity of the decomposition.

In the context of parallel SAT solving, the division of the problem into sub-problems and the subsequent sharing of learned clauses between solvers are very important. If any sub-problems are not truly independent, or if their collective satisfiability does not encompass the original problem's requirements, solvers must communicate effectively. This communication typically involves the sharing of learned clauses, which can prevent individual solvers from pursuing nonviable paths and promote convergence on a solution that satisfies the entire set of sub-problems (4).

Thus, while the equation outlines the ideal scenario, the practical application of this approach must account for the nuanced dynamics of solver interdependencies and the potential need for iterative communication to align solver efforts towards a unified resolution of the original SAT problem. During the solving process, solvers engage in learning new clauses based on their computations. These learned clauses are crucial as they can prevent other solvers from exploring paths that lead to conflicts or redundancies, thus streamlining the collective effort towards finding a solution. Effective communication protocols are established to facilitate the sharing of these clauses, ensuring that all solvers are synchronized in their search and that their collaborative efforts are constructive.

It is vital to implement dynamic load balancing and clause sharing mechanisms thoughtfully to avoid the pitfalls of unbalanced workloads and inefficient clause exchanges (9). When solvers become idle, they may employ strategies such as work stealing to remain productive, asking for new jobs from busier solvers. A well-designed divide-and-conquer parallel SAT solver will dynamically adapt to the evolving problem space, balancing workloads and integrating learned clauses to efficiently converge on a solution (4).

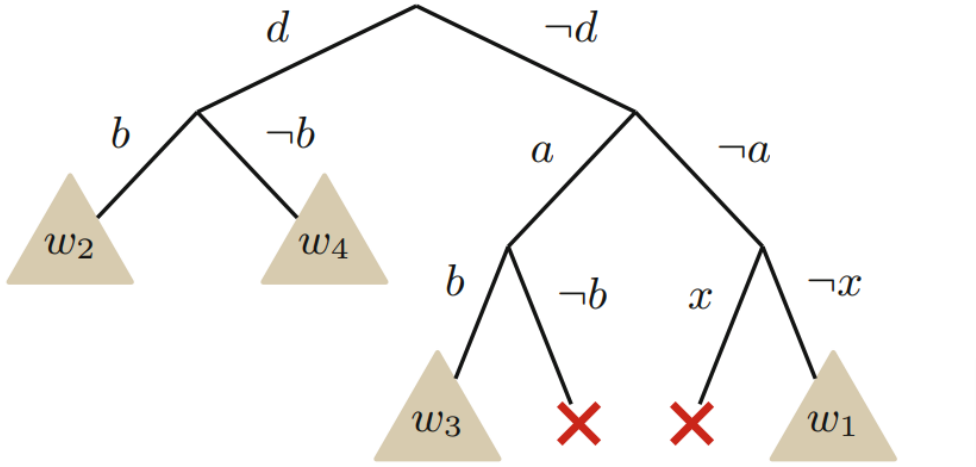

Figure 3: Divide-And-Conquer SAT Solving (7).

3.1.3 Challenges in Parallel SAT Solving

Despite its advantages, parallel SAT solving faces several challenges:

Communication Overhead: In parallel SAT solving, especially within divide-and-conquer strategies, processors often need to exchange information, such as learned clauses, partial assignments, or search progress. This inter-processor communication introduces overhead, quantified by the time T_{comm} spent on communication, which does not contribute directly to problem-solving. The efficiency of a parallel SAT solver can be significantly hindered by excessive communication overhead, especially if the communication infrastructure is not optimized or if the algorithm requires frequent exchanges of large amounts of data. Effective solutions need to minimize communication or ensure that its cost is outweighed by the parallelization benefits. The communication overhead can be represented as:

$$T_{comm} = \sum_{i=1}^{N} t_i \tag{4}$$

where N is the number of communication events and t_i is the time taken for each communication event.

Load Balancing: Achieving optimal load balancing in parallel SAT solving is a critical challenge. The goal is to ensure that all processors are equally busy, avoiding scenarios where some processors are idle due to lack of work while others are overloaded (4). This uneven distribution can lead to inefficiencies and increased solving times, negating some of the benefits of parallelization. Effective load balancing requires dynamic workload distribution strategies that can adapt to the evolving computational needs of the solving process, taking into account the varying complexities of different parts of the search space. Load balancing can be mathematically assessed using the variance of processor workloads:

$$V_{load} = \frac{1}{P}\sum_{i=1}^{P}(w_i - \overline{w})^2 \tag{5}$$

where P is the number of processors, w_i is the workload on processor i, and $\overline{w}$ is the average workload across all processors. The variance (V_{load}) measures the spread of workloads among the processors. A smaller V_{load} value indicates a more even distribution of workloads, which is the desired outcome for load balancing. This is because a lower variance signifies that each processor's workload is closer to the average workload ($\overline{w}$), implying that work is more evenly distributed. In contrast, a higher V_{load} suggests significant discrepancies in workload distribution, with some processors potentially being overburdened while others remain underutilized. Thus, minimizing V_{load} is essential for enhancing the overall efficiency of parallel SAT solvers, as it ensures that computational resources are utilized optimally, leading to potentially faster and more efficient problem-solving. In Figure 4, worker w3 proves its subspace to be unsat, and asks for a new one. Worker w2 is chosen to divide and share its subspace. In Fig. 4, m is chosen as division variable and two new guiding paths are created, one for w2 and one for w3. Worker w3 now works on a new subspace and its new guiding path is (d b ¬m), while the guiding path of w2 is (d b m) (7).

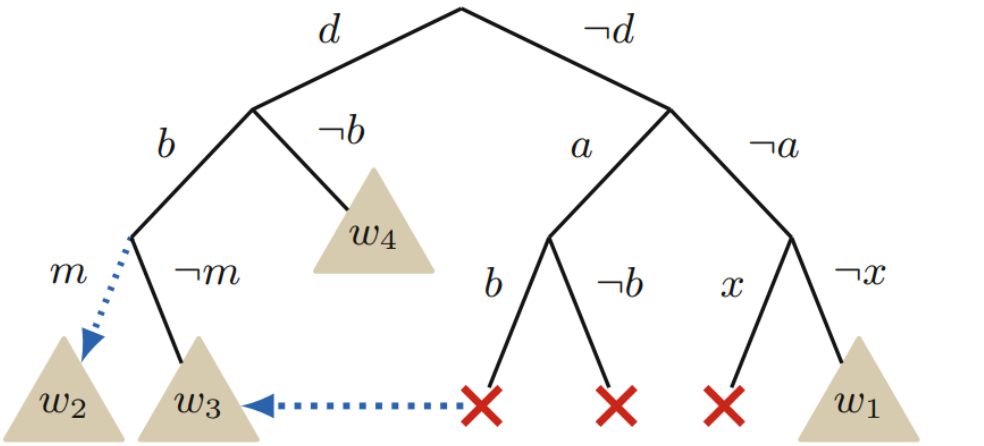

Figure 4: Load balancing through work stealing (7).

Scalability: Scalability refers to the ability of a parallel SAT solver to effectively utilize an increasing number of processors. As the number of processors grows, managing them and the resources they require becomes more complex (3). Scalability challenges include maintaining efficient inter-processor communication, avoiding bottlenecks, and ensuring that the overhead of managing parallelism does not negate the performance gains. To achieve scalability, parallel SAT solvers must implement strategies that ensure additional processors contribute to a proportional increase in solving speed, without leading to diminishing returns. This often involves sophisticated resource management and algorithmic strategies designed to maximize parallel efficiency. Scalability can be quantified by the speedup ratio S:

$$S = \frac{T_1}{T_P} \tag{6}$$

where T_1 is the execution time using a single processor, and T_P is the execution time using P processors. Ideal scalability is achieved when $S = P$, indicating that the execution time is inversely proportional to the number of processors.

3.1.4 Advancements and Applications

Recent advancements in parallel SAT solving include the development of advanced load-balancing algorithms, the integration of machine learning for optimizing solver configurations, and improved communication protocols for distributed computing. These developments have broadened the applicability of parallel SAT solvers in areas such as cryptographic analysis, extensive software verification, and complex scheduling in large-scale projects.
In summary, parallel SAT solving represents a significant evolution in tackling modern SAT problems, addressing challenges such as communication overhead and load balancing through ongoing research and technological enhancements. These improvements continuously refine the efficiency and scope of parallel SAT solvers, solidifying their role as vital tools in a range of complex computational domains.

3.2 Glucose

Glucose SAT Solver is a prominent tool in the realm of computational logic and satisfiability solving. Initially introduced in 2009, its core functionality revolves around an advanced scoring scheme for clause learning mechanisms in modern SAT solvers. This approach significantly enhances the solver's efficiency in handling SAT problems. Glucose's design is based on the concept of "glue clauses," which are specific kinds of clauses that the solver identifies and retains during the search process. These clauses are crucial for the solver's efficiency, as they represent the backbone of its learning mechanism. Glucose's ability to detect and preserve these clauses is a key factor contributing to its superior performance compared to other solvers.

One of the distinctive features of Glucose is its foundation on Minisat, a well-known SAT solver. This relationship implies that Glucose inherits many of Minisat's efficient strategies while also building upon them with its unique techniques. Users of Glucose are encouraged to acknowledge Minisat in their citations, highlighting the collaborative and iterative nature of advancements in SAT solving technologies. Over the years, Glucose has seen various iterations and improvements. The solver has consistently demonstrated its capabilities in numerous SAT competitions, securing top positions across different categories, such as parallel processing and incremental solving. These achievements not only illustrate the solver's robustness but also underscore the ongoing research and development efforts behind its success.

Glucose's development and maintenance have been a collaborative effort, primarily led by Gilles Audemard and Laurent Simon. Their continuous work and contributions to the field of SAT solving have made Glucose a go-to tool for researchers and practitioners dealing with complex computational logic problems.

In summary, Glucose SAT Solver stands out as a highly effective and adaptable tool in the field of SAT solving. Its innovative approach to clause learning, coupled with the ongoing enhancements

and a strong foundation in established SAT solving principles, makes it a valuable asset for tackling a wide range of logical and computational challenges.

Glucose Syrup represents an advanced version of the Glucose SAT Solver, specifically designed for parallel SAT solving. Introduced as part of the evolution of the Glucose solver, Glucose Syrup is tailored to harness the power of parallel processing, enabling it to handle more complex and computationally demanding SAT problems more efficiently.

The development of Glucose Syrup was driven by the increasing need for more powerful SAT solvers capable of exploiting modern multi-core processors. By adapting and refining the original Glucose solver for parallel execution.

A key feature of Glucose Syrup is its innovative clause sharing mechanism. This mechanism is crucial for parallel SAT solving, as it allows multiple processing cores to effectively share and manage clauses, significantly improving the solver's overall efficiency.

Glucose Syrup's parallel capabilities do not compromise its foundational principles and strategies inherited from the original Glucose solver. It continues to focus on efficient clause learning and management, leveraging the concept of glue clauses, which remains a central aspect of its problem-solving approach.

The development of Glucose Syrup has emphasized flexibility and adaptability, offering various compilation options that allow users to tailor the solver to their specific needs. With features such as max memory size and count of threads, Glucose Syrup provides a range of configurations to suit different problem types and computing environments.

In essence, Glucose Syrup is a testament to the continuous evolution and innovation in the field of SAT solving. By extending the capabilities of the original Glucose solver to the realm of parallel processing, Glucose Syrup represents a significant step forward in tackling the increasingly complex and diverse challenges encountered in computational logic and SAT Solving (10).

4 Strategies for Clause Management in Parallel SAT Solvers

This chapter highlights the critical challenges and innovative strategies for clause management in parallel SAT solvers. In the journey to optimize SAT solvers, particularly in the realm of parallel computing, I encountered a pivotal challenge: managing the overwhelming flow of clauses in complex problems, especially where clauses are large and the volume of shared information between processing units is excessive. This observation led me to ponder whether a more aggressive elimination of clauses could enhance performance, especially considering the vast size of the clause database. It became apparent that in such environments, where the exchange of clauses is both frequent and voluminous, aggressively removing the clauses that do not pass a clause size filter could improve efficiency.

4.1 Clause size based approach

In parallel computing, the management of clauses, especially in the context of the SAT solver Glucose, is essential for balancing complexity, efficiency, and performance. A straightforward strategy for clause management based on clause size simplifies the process, addressing the operational needs of frequent clause exchanges between threads. This approach is designed to enhance system performance while minimizing computational overhead.

For Glucose, efficient clause exchanges are crucial for distributing the workload and achieving parallel processing. Simplification is favored due to the computational overhead associated with complex management methods, which can slow down the solver's operations. A complex approach, while offering refined control, may inadvertently reduce performance by imposing a significant computational burden.

The strategy I developed for use with Glucose's parallel solver is to discard clauses that exceed a predetermined size threshold. This assertive approach sacrifices the retention of potentially useful clauses for the sake of maintaining operational simplicity and efficiency. The goal is to ensure high performance across the solver's operations by minimizing unnecessary computational load.

Given a set of clauses $C = \{c_1, c_2, \ldots, c_n\}$ in a SAT problem, the clause size-based filtering can be mathematically defined as follows:

$$C_{filtered} = \{c \in C \mid |c| \leq T_{size}\} \tag{7}$$

where $|c|$ denotes the number of literals in clause c, and T_{size} is the size threshold. Clauses with a size greater than T_{size} are discarded:

$$C_{discarded} = \{c \in C \mid |c| > T_{size}\} \tag{8}$$

Given the nature of complex SAT problems, which typically feature extensive clause databases and predominantly long clauses, an analytical perspective on the strategy of aggressive clause size-based filtering reveals its strategic advantage. For SAT problems characterized by a significantly large clause database C, we can mathematically evaluate the impact of this filtering strategy through the retention probability P_{retain} of clauses:

$$P_{retain} = 1 - \frac{|C_{discarded}|}{|C|} \tag{9}$$

Here, $|C_{discarded}|$ signifies the count of clauses eliminated for surpassing the designated size threshold, while $|C|$ represents the entirety of the clause population before the application of size-based filtering. This equation elucidates a pivotal observation: within the expansive realm of C, excising a fraction of clauses $|C_{discarded}|$ scarcely detracts from the solver's capacity to deduce a solution. This minimal impact is particularly manifest in scenarios dominated by lengthy clauses, illustrating that aggressive filtering predominantly expunges clauses with marginal or no relevance to efficient problem resolution.

Therefore, when confronted with the dual challenges of a large clause database and the prevalence of extended clauses, adopting an assertive stance on clause size-based filtering does not compromise, but rather, potentially enhances the solver's performance. This method ensures that computational efforts are concentrated on evaluating clauses that are inherently more manageable and likely to be crucial for unraveling the SAT problem. Thus, aggressive clause filtering emerges not as a hindrance but as a facilitator of efficiency, meticulously pruning the clause database to retain only those elements most conducive to solving complex SAT problems efficiently.

4.1.1 Dynamic Threshold Management

A key aspect of this strategy is the dynamic adjustment of clause acceptance thresholds, allowing the system to respond adaptively to the evolving needs of the solver. The system monitors the

distribution of clause sizes and adjusts thresholds incrementally, ensuring that the solver remains responsive without causing instability. This adaptive approach is critical for managing the flow and size of clauses effectively, significantly impacting the solver's overall throughput. Here's a step-by-step breakdown of how this strategy is applied:
Management of the clause buffer ensures that only appropriately sized clauses are added, and overflow is prevented by removing older clauses as necessary. This process aligns with the dynamic threshold management, facilitating the addition of clauses in accordance with the solver's operational strategy. Initial Setup Two separate thresholds are set for clause sizes, maxvalueeven and maxvalueodd, to differentiate handling based on the thread ID (even or odd). This dual-threshold system allows for flexible management tailored to varying computational loads. A maximum threshold value, maxthreshold, is defined to cap the adaptive increase of clause size acceptance thresholds, ensuring the system remains within operational bounds. Clause Addition Process (pushClause Function) Determine Current Length Threshold: For each incoming clause, the function first determines the current acceptance threshold based on the thread ID (even or odd). This differentiation ensures that clauses are managed in a way that reflects their originating thread's computational context. Size Check and Buffer Management; If the size of the incoming clause is within the current threshold, the system proceeds to check if the clause buffer has enough space to accommodate the new clause without exceeding the maximum buffer size (maxsize). If the buffer is full, older clauses are removed to make room for the new one. This ensures that the buffer remains within its capacity limits, maintaining system stability and responsiveness. Handling Clauses Above the Threshold: Clauses that exceed the current size threshold are not added. Instead, their occurrence increments an aboveThresholdCount. When this count reaches the maxthreshold, it triggers an increase in the current length threshold for the respective thread ID group (even or odd). This adaptive mechanism allows the system to accommodate larger clauses over time, reflecting changes in the computational landscape or problem complexity.

Threshold Adjustment (increaseThreshold Function) If the count of clauses exceeding the current threshold hits the maxthreshold, the acceptance threshold for the respective thread ID group is increased by one. This is a controlled adaptation, ensuring that the solver can gradually adjust to handling larger clauses without abrupt changes that could destabilize operations. Rate Limiting: The thresholdIncreaseRate monitors the frequency of threshold adjustments. If adjustments occur too rapidly, indicating a significant shift in clause characteristics, the maxthreshold itself is increased. This rate-limiting step prevents the system from becoming too permissive or strained by excessively large clauses, maintaining a balance between adaptability and performance.

The simplified yet adaptive threshold management strategy within Glucose's parallel SAT solver demonstrates an effective approach to clause management in a complex, multi-threaded environment. By balancing simplicity with adaptability, it ensures efficient processing across a wide range of clause sizes, maintaining system stability and high performance. This methodology not only meets the operational requirements of SAT solving but also offers insights into designing efficient computational systems in parallel computing.

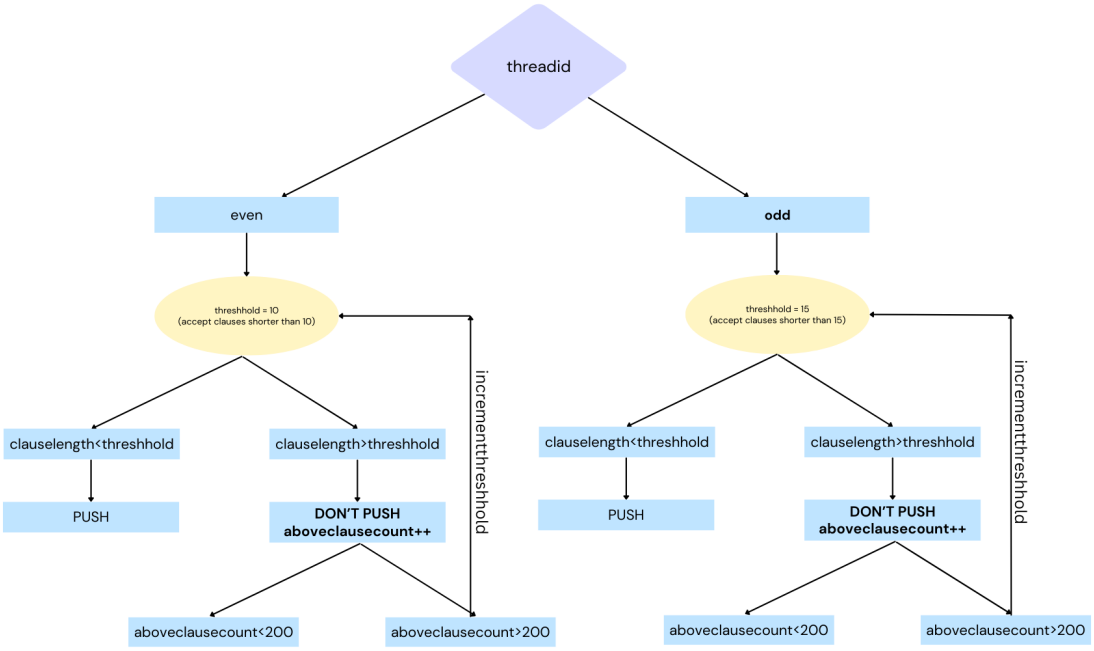

Figure 5: Dynamic Threshold Strategy.

4.2 Dynamic var decay Adjustment

Starting with an overview of Glucose's initial strategy for incrementally increasing the variable decay factor (vardecay), we note that Glucose periodically boosts vardecay by 0.01 every 5000 conflicts, provided it remains below a predefined maxvardecay. This method aims to gradually adjust the solver's focus towards variables involved in recent conflicts, thereby enhancing its problem-solving capabilities over time. Before delving into performance-based adjustments, the function performs essential preliminary checks:
Null Pointer Verification: It first checks if the MultiSolvers object (ms) is a nullptr. An error message is issued if true, preventing further execution to avoid crashes or undefined behavior.
Solver Availability: Next, it confirms that the solvers vector within ms is not empty. An error message is emitted if no solvers are available, ensuring that there are solvers to adjust.

4.2.1 Conflict Analysis and Adjustment Preparation

Average Conflicts Calculation: The total number of conflicts across all solvers is calculated to determine the average conflict count. This average serves as a baseline for assessing each solver's performance, setting the stage for tailored adjustments. Dynamic Performance-Based Parameter Adjustment For each solver within MultiSolvers, the process unfolds as follows:

Conflict Difference: The difference between the solver's conflicts and the average conflicts ($conflictChange$) is calculated. Adjustment Factor Calculation: An $adjustmentFactor$ is derived by dividing $conflictChange$ by the average conflicts, guiding the subsequent parameter adjustments.

1. Rapid Increase in Conflicts: adjustmentFactor > 0.15: Significantly higher conflicts indicate a solver exploring challenging search space areas. In response, *var_decay* and *max_var_decay* are decreased to temper the solver's pace in transitioning away from current variable focuses. Concurrently, *firstReduceDB* is increased to delay clause database reduction, facilitating a deeper analysis of existing clauses.

2. Moderate Increase in Conflicts: adjustmentFactor > 0.05: This scenario triggers less intense adjustments than a rapid increase, aiming to recalibrate the solver's exploration strategy without drastically altering its course.

3. Decrease in Conflicts: adjustmentFactor < -0.05: Fewer conflicts suggest efficient exploration, meriting an increase in *var_decay* and *max_var_decay* to foster aggressive exploration. *firstReduceDB* is decreased to accelerate clause reduction, sharpening the solver's focus on promising solution paths.

4. Stable Conflict Rate: adjustmentFactor between -0.05 and 0.05: In this case, the solver's *var_decay* and *max_var_decay* are slightly decreased, and the *firstReduceDB* is adjusted based on the solver's conflicts and the maximum limit for *firstReduceDB*.

The *firstReduceDB* is reset to its initial value when it reaches the maximum limit or drops to zero. If the *adjustmentFactor* is less than -0.05 and the solver's conflicts are less than the average conflicts, the *firstReduceDB* is decreased. If the *firstReduceDB* drops to zero or less, it is reset to its initial value.

This dynamic adjustment mechanism empowers each solver within MultiSolvers to refine its strategy based on comparative performance, aiming to enhance solving process efficiency. By adjusting solver behavior in alignment with current performance trends, this method elevates the overall problem-solving efficacy.

Integrating real-time performance data with strategic parameter adjustments, this approach equips SAT solvers to more adeptly navigate the complexities of SAT problems. It underscores the paramount importance of adaptability and precision in computational problem-solving, leveraging nuanced performance insights to optimize solver strategies effectively.

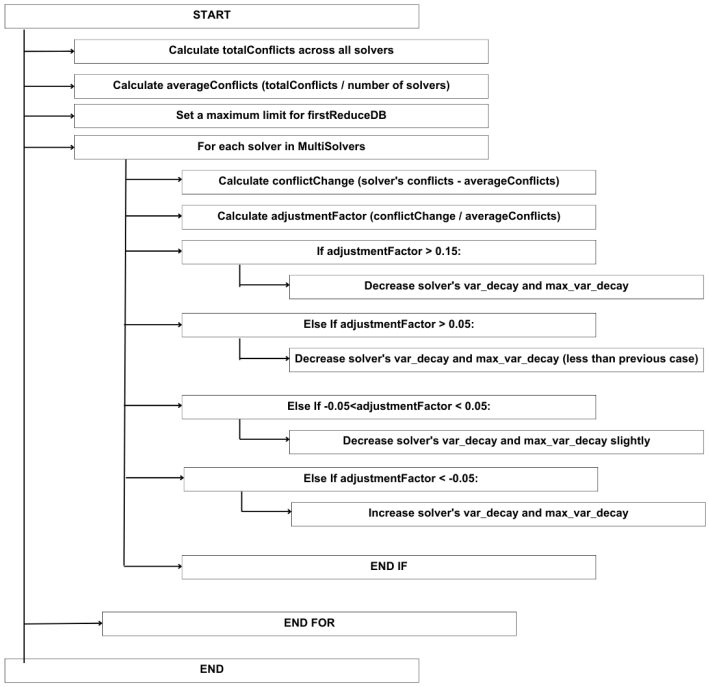

Figure 6: Dynamic var decay adjustment

5 Performance and Measurements

5.1 Tests and test environment

The experiments were conducted on a personal workstation configured with Windows 10, supplemented by Ubuntu through the Windows Subsystem for Linux (WSL). The hardware specifications include:

- CPU: Intel(R) Core(TM) i7-7700HQ, 4 cores with Hyper-Threading, enabling 8 logical processors.
- Memory: 32GB RAM.
- Storage: 256GB TOSHIBA NVMe SSD and an additional 500GB SanDisk SSD.
- Networking: Intel(R) Dual Band Wireless-AC 8265.
- Graphics: NVIDIA GeForce GTX 1050Ti.

5.2 Analysis

This section delves into the performance outcomes derived from the evaluation of approximately 230 benchmarks collected from the years 2010 to 2017. These benchmarks, ranging from straightforward to highly intricate problems, serve as the foundation for a comparative analysis between a modified version of Glucose syrup and the standard Glucose syrup across a diverse set of scenarios.

The core of the evaluation centered on CPU runtime. This allowed for a detailed examination of how each version responds under different computational challenges. Upon reviewing the results, a differentiated performance landscape emerged. For less complicated problems, both the modified version of Glucose syrup and the standard Glucose syrup displayed similar levels of efficiency. However, as the complexity of the benchmarks increased, distinct differences in performance became apparent. The modified version of Glucose syrup managed to process more complex benchmarks more efficiently than its counterpart. This improvement suggests that the modifications introduced to Glucose syrup effectively enhance its ability to tackle demanding computational tasks. Such enhancements could stem from the strategies that were introduced in this thesis.

This analysis was conducted with the aim of objectively comparing the performance differences between the two versions under specific conditions. The findings underscore the modified version's enhanced performance, particularly in handling more complex scenarios.

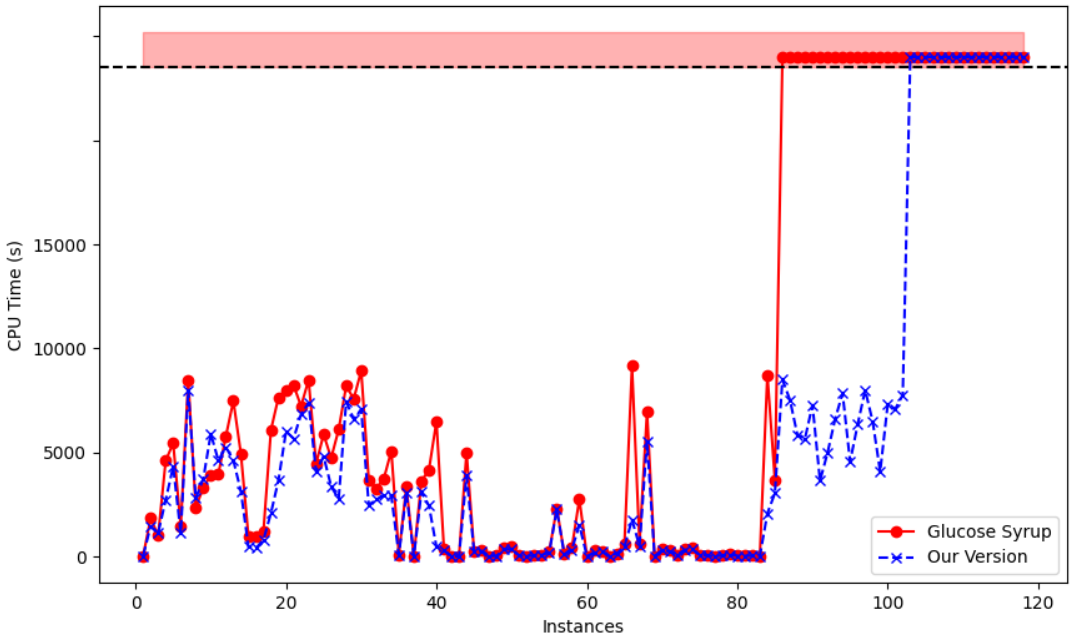

Figure 7: Benchmarks 2010-2014

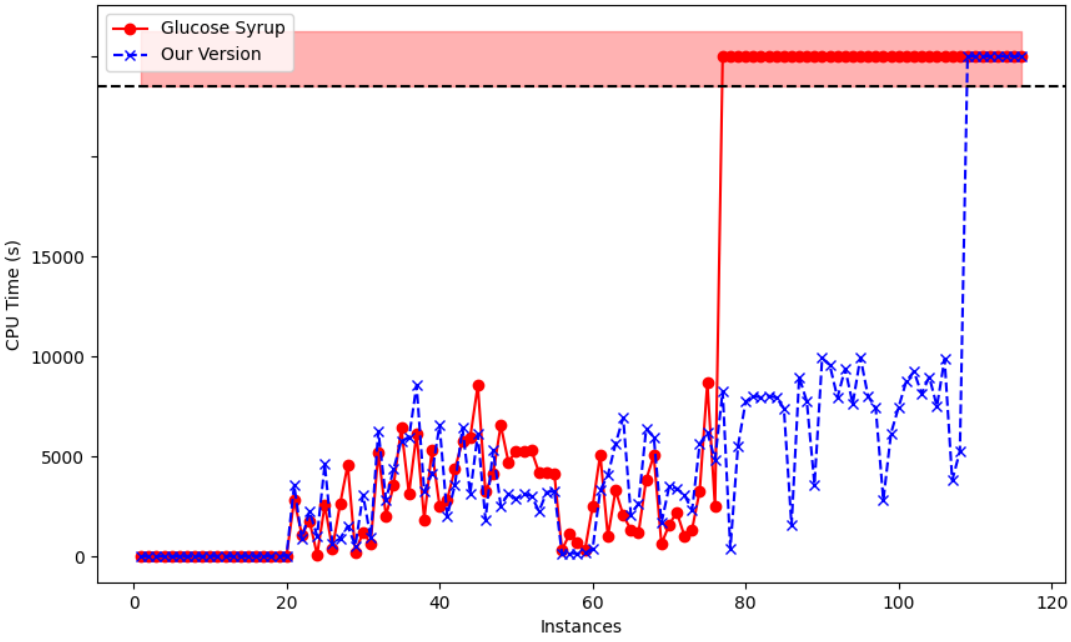

Figure 8: Benchmarks 2015-2017

6 Code and Data

The code is developed based on Glucose's GitHub repository(cmake edition), updated in the *parallel* directory of the code, and segmented into several classes such as `ClausesBuffer`, `ParallelSolver`, and `SolverConfiguration`. The code is available under this repository:glucose-optimized.

7 Analysis and Conclusion

In this thesis, an approach was explored to enhance the efficiency of SAT solvers, specifically focusing on Glucose Syrup, without introducing complex algorithmic changes. The emphasis was placed on developing methodologies that streamline clause management and parameter adjustment in a manner that improves solver performance while maintaining simplicity in implementation.

A strategy based on the size of clauses was implemented, aimed at reducing the computational load on Glucose Syrup by selectively retaining only those clauses within certain size limits. This method effectively minimized the overhead associated with processing a large volume of clauses, thereby optimizing the solver's ability to manage and process information more efficiently. By discarding clauses exceeding predetermined size thresholds, it was ensured that Glucose Syrup could concentrate on analyzing the most relevant and manageable pieces of data, thus enhancing its problem-solving capabilities.

Central to this approach was the dynamic adjustment of clause acceptance thresholds, a mechanism that allowed for the adaptation of the solver to varying computational demands. Through

continuous monitoring of clause size distributions, adjustments to thresholds were incrementally made, ensuring that Glucose Syrup remained responsive and stable under different problem-solving conditions. This adaptability played a crucial role in maintaining the solver's efficiency, as it provided a means to accommodate a broader range of clause sizes based on the evolving needs of the computational tasks at hand.

Furthermore, the adjustment of variable decay rates based on conflict analysis introduced an additional layer of optimization. By analyzing the performance and adjusting solver parameters in real-time, a more targeted approach to problem-solving was facilitated. This method allowed for the empirical tuning of key operational parameters, thereby enhancing the solver's focus on variables that are most relevant to the conflicts encountered. This strategic adjustment of parameters based on solver performance and conflict data highlighted the importance of adaptability in the optimization process, ensuring that Glucose Syrup could navigate through the complexities of SAT problems with improved efficiency.

The results from the implementation of these strategies were promising. Despite the elimination of a considerable number of clauses through the size-based selection process, an improvement in the performance of Glucose Syrup was observed. This outcome underscored the effectiveness of a selective and adaptable approach in enhancing solver efficiency

In conclusion, the research conducted in this thesis revealed that the application of selective clause management and dynamic parameter adjustment strategies could significantly improve the efficiency of SAT solvers like Glucose Syrup. By focusing on the simplification of clause management and the adaptability of solver parameters, it was demonstrated that enhancements in performance are attainable without the need for complex algorithmic modifications. These findings contribute to a broader understanding of how SAT solvers can be optimized and underscore the potential of methodological simplicity and flexibility in addressing computational challenges.

A key challenge encountered was the need to thoroughly understand the existing code and the techniques it employed. Modifying Glucose Syrup's already optimized system demanded a deep comprehension of its workings to ensure that any changes made would enhance, rather than detract from, its performance.

8 Future Work

The exploration of dynamic threshold management and dynamic variable decay adjustment in this thesis sets the stage for a series of intriguing future research avenues in the domain of parallel SAT solving. One particularly promising direction is the advancement of clause management strategies through the implementation of a priority queue mechanism. This approach aims to refine the process of clause exchange among processing units by prioritizing clauses based on a set of heuristics, such as clause utility, activity, and size, potentially enhancing the solver's efficiency by focusing on the most relevant clauses.

The development of such a priority queue mechanism necessitates a comprehensive investigation into appropriate heuristics for clause prioritization. These heuristics could include, but are not limited to, the frequency of clause usage in successful conflict resolutions, the historical significance of clauses in solving similar problems, and their compatibility with current dynamic threshold parameters. Implementing this system involves not only the design of a sophisticated scoring model to assess clause utility but also the integration of this model into the existing parallel

SAT solving framework, ensuring that clauses are dynamically ranked and exchanged based on their calculated priority.

A critical aspect of this future work will be the analysis of communication overhead associated with the priority queue system. Given the potential for increased data exchange among processing units due to dynamic clause prioritization, it is imperative to carefully evaluate whether the efficiency gains in solver performance justify the additional communication costs. This evaluation will require a meticulous experimental setup, comparing the performance of parallel SAT solvers employing the priority queue mechanism against those using traditional clause management strategies. Such comparative analysis will not only shed light on the practicality and effectiveness of the priority queue approach but also offer insights into optimizing the balance between communication overhead and computational efficiency.

Moreover, this line of research opens the door to further innovations in parallel SAT solving, including the exploration of machine learning models to predict clause utility more accurately, the development of adaptive algorithms for real-time adjustment of dynamic thresholds, and the investigation of cross-solver coordination strategies for more efficient clause exchange. The integration of these advanced techniques could lead to a new generation of parallel SAT solvers that are not only more adept at managing complex clause databases but also more efficient in navigating the vast solution spaces of challenging SAT problems.

In conclusion, the future work stemming from this thesis has the potential to contribute to the field of parallel SAT solving. By focusing on the refinement of dynamic threshold management and variable decay adjustment strategies, especially through implementing a priority queue mechanism for clause prioritization, is expected to provide insights into enhancing solver efficiency and effectiveness.

List of Figures

1 The implication graph with conflict cut. 7
2 Portfolio-based SAT Solving (4). 9
3 Divide-And-Conquer SAT Solving (7). 11
4 Load balancing through work stealing (7). 12
5 Dynamic Threshold Strategy. 17
6 Dynamic var decay adjustment 19
7 Benchmarks 2010-2014 20
8 Benchmarks 2015-2017 21

References

[1] Weiwei Gong and Xu Zhou. A survey of SAT solver. *AIP Conference Proceedings*, 1836(1):020059, 06 2017.

[2] Jia Hui Liang, Chanseok Oh, Minu Mathew, Ciza Thomas, Chunxiao Li, and Vijay Ganesh. Machine learning-based restart policy for cdcl sat solvers. pages 94–110, 2018.

[3] Youssef Hamadi, Said Jabbour, and Lakhdar Sais. 'manysat: a parallel sat solver'. *Journal on Satisfiability, Boolean Modeling and Computation*, 6:245–262, 2010.

[4] Marcos Barreto, Sergio Nesmachnow, and Andrei Tchernykh. Hybrid algorithms for 3-sat optimisation using mapreduce on clouds. *International Journal of Innovative Computing and Applications*, 2017.

[5] Ming Ouyang and Vasek Chvatal. Implementations of the dpll algorithm. 1999.

[6] Robin Coutelier et al. Chronological vs. non-chronological backtracking in satisfiability modulo theories. 2023.

[7] Tomáš Vojnar and Zhang Lijun, editors. *Tools and Algorithms for the Construction and Analysis of Systems*. 2019.

[8] Abhishek Nair, Saranyu Chattopadhyay, Haoze Wu, Alex Ozdemir, and Clark Barrett. Proof-stitch: Proof combination for divide and conquer sat solvers. 2022.

[9] Tomas Balyo, Peter Sanders, and Carsten Sinz. Hordesat: A massively parallel portfolio sat solver. 2015.

[10] Laurent Simon Gilles Audemard. On the glucose sat solver. 2018.