Exploration Guided Conflict Directed Clause Learning (CDCL) SAT Solving

Md Solimul Chowdhury

November 20, 2019

Abstract

Heuristic search is an important subarea of Artificial Intelligence. An inherent issue of any heuristic-guided search method is the inaccuracy of its heuristic estimation, which may result in poor search guidance. Looking ahead by random exploration to evaluate the future search states is a successful approach in game-playing, deterministic planning, and automated theorem proving.

Boolean Satisfiability (SAT) is a canonical NP-complete problem. Despite its hardness in theory, modern Conflict Directed Clause Learning (CDCL) SAT solvers can solve very large real-world SAT instances with surprising efficiency. One key component of a CDCL SAT solver is its clause learning process, which learns a new clause from a conflict, whenever one occurs. Clause learning is highly beneficial for the search, as each learned clause prunes the search space. Hence, the efficiency of CDCL SAT solvers crucially depends on their ability to generate conflicts at a fast rate.

Another key component of a CDCL SAT solver is its *branching heuristic*, which selects a variable to branch on during the search. After selecting a branching variable, a boolean value is assigned to it by using a *polarity selection heuristic*. Standard CDCL heuristics for branching and polarity selection are designed based on a look-back principle that uses information of past search states to generate conflicts at a fast rate. However, these heuristics are often inaccurate, as not every decision leads to a conflict.

In this thesis work, I propose to study exploration methods based on random walks in the context of CDCL SAT solving. The purpose of performing exploration is to find valuable heuristic information about a search state based on sampling of *future* search states, as opposed to standard CDCL heuristics, which evaluate a search state based on the information from *past* search states. I have made some progress on this proposed thesis work, which includes empirical identification of weaknesses of CDCL branching and polarity heuristics and the development and preliminary evaluation of *expSAT*, an algorithmic framework for exploration based CDCL SAT solving that extends the popular CDCL branching heuristic VSIDS with exploration. In the remaining time of this thesis work, I propose to design exploration based CDCL branching and polarity heuristics.

Contents

List of Tables

List of Figures

1 Introduction and Preliminaries

1.1 Introduction

The field of heuristic search traditionally rested on two pillars: efficient search algorithms and high-quality heuristic functions. In recent years, the toolbox of heuristic search has been expanded in two major ways: by adding sophisticated exploration methods to the search, which provides more robustness compared to methods that rely on the quality of the heuristic alone; and by learning better heuristics through machine learning methods.

Exploration can potentially make a search more robust by mitigating "early mistakes" caused by inaccurate heuristics [50]. Examples of exploration methods are Monte Carlo Tree Search (MCTS) [11] and random walk techniques, which have been successfully applied to both classical [33, 51] and motion planning [23]. MCTS-based search methods were shown to be effective in game-playing [31], automated theorem proving [16] and puzzle domains [42]. Perhaps the best-known example, which combines many of the recent advances in both MCTS and machine-learned heuristics is the super-human strength Go-playing program AlphaGo [46].

Given a formula $\mathcal{F}$ of boolean variables, the problem of Boolean Satisfiability (SAT) is to either determine variable assignments that satisfy $\mathcal{F}$, or report UNSAT in case no such assignment exists [10]. In general, SAT solving is known to be NP-complete [13]. Complete SAT solvers based on the Davis Putnam Logemann Loveland (DPLL) framework [14] employed heuristics-guided backtracking tree search. Solvers such as GRASP [45] and Chaff [32] substantially enhanced the DPLL framework by adding conflict analysis and clause learning in Conflict Directed Clause Learning (CDCL) SAT solvers. Despite the hardness of SAT problems, modern CDCL SAT solvers can solve very large real-world problem instances from important domains, such as hardware design verification [19], software debugging [12], planning [41], and encryption [30, 47], with surprising efficiency.

The key decision-making step in a CDCL SAT solver is its *branching decision* on a variable assignment, which is performed by using a combination of *branching heuristic* and *polarity heuristic*. A branching heuristic selects a variable from the current set of unassigned variables and a polarity heuristic assigns a boolean value to the variable selected by the branching heuristic. Both variable and value selection have a dramatic effect on search efficiency.

The conflict-driven Variable State Independent Dynamic Sum (VSIDS) heuristic [32] and its variants have been the leading branching heuristics for over 15 years. Their dominance has recently been challenged by heuristics named Learning Rate Based (LRB) and Conflict History Based (CHB), which are based on modeling variable selection as an online multi-armed bandit problem [25, 26]. All the state of the art CDCL branching heuristics prioritize selection of variables which have appeared in the recent conflicts. The phase saving heuristic [38] is the state of the art CDCL polarity selection heuristic. Given a variable v to branch on, the phase saving heuristic assigns the same boolean value to v which was assigned to v last time it was propagated. Standard CDCL branching and polarity selection heuristics are based on a look-back principle of heuristic estimation from previous search states.

By definition, search heuristics are imperfect estimations of an action's quality. In this thesis, I am studying the extent of inaccuracy in the look-back based state of the art CDCL branching and phase saving heuristics to derive important insights. My goal is to exploit these insights to develop exploration enhanced CDCL branching and polarity heuristics, which I expect to be more efficient than current state of the art heuristics. My progress on this proposed topic includes: (i) empirical identification of inaccuracy of heuristic estimation of VSIDS, which leads the search to pathological phases of Conflict Depression (CD), (ii) the development of an algorithmic framework named *expSAT* that performs random explorations to rectify CDCL SAT search during CD phases, and (iii) empirical evaluation of *expSAT* which shows the performance improvements of two state of the art CDCL SAT solvers extended with the *expSAT* approach. In the remaining time of this thesis work, I will continue to design random exploration based CDCL branching and polarity selection heuristics.

1.2 Preliminaries

1.2.1 CDCL SAT Solving

SAT Formula and SAT Solving Task A SAT formula is a conjunction of *clauses*. A clause is a disjunction of *literals*. A literal is either a variable or its negation. For example, $\mathcal{S}$ below is a SAT formula, with five variables and four clauses.

$$\mathcal{S} = (x_1 \vee \neg x_5) \wedge (\neg x_2 \vee \neg x_1) \wedge (x_3 \vee x_4) \wedge (\neg x_3 \vee \neg x_1 \vee x_5)$$

Given a SAT formula $\mathcal{F}$, the SAT solving task is to either determine a variable assignment that satisfies $\mathcal{F}$ (i.e, a proof that $\mathcal{F}$ is SAT) or reports the unsatisfiability of $\mathcal{F}$ in case no such assignment exists (i.e, a proof that $\mathcal{F}$ is UNSAT). For example, given the SAT formula $\mathcal{S}$, a SAT solver finds the solution $x_1 = false,\ x_2 = true,\ x_3 = true,\ x_4 = true,\ x_5 = false$, which satisfies $\mathcal{S}$.

Emergence of CDCL SAT solvers SAT solvers based on the DPLL framework [14] were the first generation of complete SAT solvers that employ heuristics guided state space search to solve a given SAT formula. In the late nineties, much more powerful Conflict Driven Clause Learning (CDCL) SAT solvers such as GRASP [45], Chaff [32] had emerged. SAT solving in the CDCL framework is fundamentally inspired by the DPLL framework, but differs substantially in the way it performs search.

The Inner Workings of CDCL SAT solvers A CDCL SAT solver works by extending an initially empty *partial assignment* $assign(\mathcal{F})$, a set of literals representing how the corresponding variables are assigned. In each *branching decision*, the solver extends the current partial assignment by selecting a variable v from the current set of unassigned variables $uVars(\mathcal{F})$, and assigns a boolean value to v. v is called the *branching variable*. A branching decision is associated with a *decision level* ≥ 1, which denotes the depth of the search tree where the branching decision has taken place. After the assignment of v, *unit propagation* (UP) simplifies $\mathcal{F}$ by deducing a new set of implied variable assignments, which are also added to the current partial assignment. After UP, the search moves down to the next decision level to make another branching decision.

UP may lead to a *conflict* due to a *falsified* or *conflicting* clause, which cannot be satisfied under the current partial assignment. *Conflict analysis* determines the root cause of a conflict and generates a *learned clause (lc)* that prevents this conflict from reappearing in the future, thereby pruning the search. A *backjumping level (bl)* is computed from *lc*. If *bl* is 0 then the formula is UNSAT[1], otherwise, the search backtracks to *bl* and continues from there by making an assignment to a variable that appears in *lc*. At any given state, if all the clauses in $\mathcal{F}$ are satisfied by the assignments in $assign(\mathcal{F})$, then the $\mathcal{F}$ is SAT with respect to $assign(\mathcal{F})$.

A CDCL SAT solver can learn clauses at a very fast rate. Keeping all the learned clauses in a clause database can quickly exhaust memory, and can lead to poor speed. A CDCL SAT solver routinely manages the clause database by deleting learned clauses that it considers unimportant.

A CDCL SAT solver performs many *restarts* with an empty partial assignment. All other aspects of the current state of the search, such as the learned clauses, the heuristic scores and various parameter values are preserved at a restart. After a restart, the search starts building a new partial assignment.

Please see [10] for a detailed overview of CDCL SAT Solving.

The Importance of Fast Conflict Generation Whenever a conflict occurs during the search, a CDCL SAT solver learns a clause from that conflict. The learned clauses help to prune the search space, which has a huge impact on solving efficiency. In fact, in [39], Pipatsrisawat et. al. showed that for UNSAT formulas, the shortest refutation proofs found by the CDCL SAT solving process with clause learning are only polynomially longer than shortest refutation proofs produced by *general resolution*, the strongest known proof system.

In [27], Liang et. al. showed that more efficient branching heuristics have the following empirical properties: on average, (a) they generate more conflicts per branching decision and (b) learned clauses from those conflicts are of higher quality. Therefore generating conflicts at a fast rate and learning high quality clauses from those conflicts are very important aspects of efficient CDCL SAT solving.

CDCL Branching Heuristics The standard CDCL SAT branching heuristics are designed following the look-back principle. They prioritize the selection of variables which have been involved in recent conflicts. The intuition is that such variables generate more conflicts, if assigned again. Here we briefly discuss VSIDS and LRB as representative CDCL branching heuristics.

VSIDS: *Variable State Independent Decaying Sum* (VSIDS), introduced by Moskewicz et. al. in [32], is a popular family of dynamic branching heuristics. We focus on exponential VSIDS as a representative member. VSIDS maintains an *activity score* for each variable in a given SAT formula. During conflict analysis, VSIDS increases the activity score of each variable that is involved in conflict resolution by a *variable bumping factor* g^z, where $g > 1$ is a constant and z is the count of the number of conflicts in the search so far. VSIDS puts a strong focus on variables that participated in the most recent conflicts.

LRB: In the *Learning Rate Based* (LRB) branching heuristic [26], a variable v is regarded to become alive when it is assigned by a branching decision or propagation, and dead when

[1]The decision made at the decision level *bl* is the root cause of the current conflict. If *bl* is 0, then it indicates that the current conflict is not caused by any decision and the formula is inherently unsatisfiable.

it is unassigned by backtracking. Assume that when v gets assigned (resp. unassigned), z (resp. $z' > z$) is the count of number of conflicts in the search so far. z (resp. z') marks the birth (resp. death) of v. This heuristic tracks the participation count $P(v)$ of v in generation of learned clauses within the conflict interval $I(v) = z' - z$. When v is unassigned, LRB computes the *reward* $R(v) = \frac{P(v)}{I(v)}$, which is the rate of its participation in learned clause generation. An activity score for v is computed from $R(v)$. In search, the variable with maximum activity score is selected for branching.

CDCL Polarity Heuristics

The Phase Saving Heuristic: Assume that v^+ and v^- are the two literals where variable v is assigned *true* and *false*, respectively. Also assume that at decision step i, a variable u is assigned, which creates a propagation with v^x, where $x \in \{+, -\}$ and that this propagation is followed by a conflict. After conflict analysis, during backtracking, the phase saving heuristic saves this last assigned polarity x of v in $polarity[v]$. At decision $j > i$, assume that the CDCL branching heuristic selects v. The phase saving heuristic selects the literal v^x to branch on [38].

In state of the art CDCL SAT solvers, such as Glucose [1] and Maple [2], $polarity[v]$ is initialized to $-$ for each variable v of a given SAT formula. This initialization of phases to negative polarity is an artifact of the encoding of most SAT benchmarks, which generally produce formulas with more positive than negative literals [20].

Learning Rate, Learned Clause Quality and Glue Clauses

Global Learning Rate The Global Learning Rate (GLR) [27] is defined as $\frac{n_c}{n_d}$, where n_c is the number of conflicts generated in n_d decisions. GLR measures the average number of conflicts that a solver generates per decision.

The Literal Block Distance (LBD) Score The LBD score of a learned clause c [7] is the number of distinct decision levels in c. If $\text{LBD}(c) = n$, then c contains n propagation blocks, where each block has been propagated within the same branching decision. Intuitively, variables in a block are closely related, and learned clauses with lower LBD score tend to have higher quality.

Glue Clauses *Glue clauses* [7] have LBD score of 2. They are the most important types of learned clauses. A glue clause connects a literal from the current decision level with a block of literals assigned in a previous decision level. Glue clauses have more potential to reduce the search space more quickly than learned clauses with higher LBD scores.

State of the Art CDCL Solver Systems At present, two families of solvers dominate the SAT competitions.

- **The Glucose Family:** Glucose[1] and its numerous extensions use VSIDS and variants of VSIDS as their branching heuristic. Solvers from this family also use rapid restart strategies and maintain aggressive clause database cleaning schedules.

- **The Maple Family:** MapleCOMSPS [2] and its numerous variants use a hybridization of various branching heuristics, such as VSIDS, LRB and Dist [49]. Solvers from this family use less aggressive clause database reduction and restart policies than the Glucose family.

1.2.2 Exploration Methods

Monte Carlo Random Walk Nakhost et. al. [33] proposed *Monte Carlo Random Walk* (MRW) in the context of deterministic planning. At a given state s of the search, MRW performs a fixed number of random walks in the local neighborhood of s. A walk consists of a fixed number of steps. The goal of exploration in MRW is to find a state s^*, which is a neighboring state of s with best heuristic score among the explored neighbouring states. Afterwards, search selects the sequence of actions that leads to s^* and repeats the process.

2 Research Proposal and Methodology

In this section, I present my PhD research proposal and methodology in detail. I organize my research plan into six topics. Each is described in a subsection, where the statements of one or more proposals are followed by a description of the proposed methodology for each proposal.

2.1 Topic 1: Empirical Properties of CDCL Branching and Polarity Heuristics

Both branching and polarity selection heuristics of CDCL SAT solvers are designed based on a look-back principle, which uses the conflict history to compute heuristics scores. In contrast, exploration entails probing of future search states, a look-ahead principle.

Standard CDCL SAT heuristics are optimized to generate conflicts at a fast rate. In the presence of strong look-back based heuristics, a crucial question is if we can use look-ahead based exploration methods to design more efficient CDCL heuristics. As the first research topic, I propose to study the empirical properties of modern CDCL branching and polarity selection heuristics to derive novel insights, which are expected to help the development of exploration based CDCL SAT heuristics.

2.1.1 Study the Conflict Generation Process of CDCL Branching Heuristics

Proposal 2.1.1 I propose to study the time distribution of conflicts generated with CDCL branching heuristics, as a function of decision steps. This study is expected to shed light on the weaknesses of the look-back branching heuristics, which look-ahead based exploration methods may exploit.

Methodology 2.1.1 For this project, I plan to study a small number of Glucose and Maple based systems that use the state of the art branching heuristics VSIDS and LRB.

I plan to modify these solvers to collect the statistics on the search process, which will potentially answer the following questions pertaining to the conflict generation pattern of the CDCL branching heuristics:

- What fraction of the decision steps produces at least one conflict?
- What fraction of the decision steps produces multiple conflicts?
- Are there phases in the CDCL SAT solving process which do not produce any conflicts?
- Are there phases in the search process, where lower quality clauses are learned as compared to the global average?

2.1.2 Study the Inaccuracy of Polarity Value Estimation

Proposal 2.1.2 The phase saving heuristic is a longstanding state of the art heuristic for value assignments to branching variables. I propose to study this heuristic to reveal insights, with the goal of developing a better exploration based polarity selection heuristic.

Methodology 2.1.2 For this project, I plan to modify a small number of CDCL SAT solvers that use the phase saving heuristic for polarity selection, and collect statistics to answer the following questions:

- Given a SAT formula, to what extent do the *final* polarity values of the variables (i.e., the polarity values assigned to those variables at the end of the search) of that formula, agree with the polarity values, assigned by the heuristic?
- To what extent does the phase saving heuristic help or fail to generate conflicts?
- To what degree does the polarity value assignment of a given variable change in the course of a search?

2.1.3 Better understanding of CDCL Branching Heuristics

Proposal 2.1.3 The longstanding dominance of the branching heuristic VSIDS has recently been challenged by the newer LRB heuristic. While both of these heuristic methods perform online heuristic estimation, their methodology is different.

VSIDS computes the heuristic score of a variable based on its participation in the generation of recent learned clauses. LRB computes its score based on the *rate* of participation in learned clauses during the window of conflicts, within which it had remained assigned. Thus LRB evaluates a variable with respect to a window of conflicts, while for VSIDS, there is no such window. LRB is more temporally focused than VSIDS.

Another important difference between VSIDS and LRB comes from the way that these two heuristics increase the activity scores of variables. To increase the activity score of a variable VSIDS maintains a dynamic variable bumping factor, which is shared by variables. In contrast, LRB uses a reward value which is unique for each variable.

The empirical understanding of the similarities and differences between LRB and VSIDS is crucial to apply exploration based heuristic in conjunction with these two heuristics.

Methodology 2.1.3 I propose to take a set of instances of small size from a fixed set of SAT benchmarks and perform a study on (a) the decision frequency pattern, (b) the propagation pattern, (c) the conflict generation pattern, and (d) the pattern of qualities of the learned clauses for both VSIDS and LRB. I plan to modify the solvers that employ these heuristics and collect relevant data to study all of these four patterns.

2.2 Topic 2: Exploration Methods and Exploration Statistics

2.2.1 Exploration Methods

Proposal 2.2.1 The purpose of performing exploration in the CDCL SAT search space is to uncover valuable heuristic information by sampling future search states. How to perform exploration in CDCL SAT is an important research question, which I propose to pursue.

Methodology 2.2.1 In [33], the authors have developed the Monte Carlo Random Walk (MRW) technique to perform exploration in deterministic planners, where each action selection is guided by statistics collected from exploration of the future search states. In that work, during an exploration episode, a fixed number of random walks are performed. A random walk is composed of a fixed number of *random steps*, where each random step selects a random action. I plan to tailor MRW technique to perform exploration in the CDCL SAT search space, under the following considerations:

- In CDCL SAT search, a random branching decision is followed by unit propagation. These two steps in combination provide the only way to perform a *random step*. I plan to design random steps by using these two operations, under the following two considerations:

  - Performing exploration before every decision via a sequence of random walks incurs high overhead. The overhead must be balanced with the performance by triggering exploration *only* intermittently.
  - A relevant question is: what are the appropriate states of the search for performing exploration? At a given state of the search, if the look-back based heuristic employed in the search is not performing well (i.e., not generating conflicts), then it may need guidance from exploration. The study of empirical properties of CDCL branching and polarity heuristics are expected to help to answer this question.

- At a given search state, performing exploration will change that search state (i.e., by extending the partial assignment). On the completion of each exploration, the search state at which that exploration had started must be restored.

2.2.2 Collecting Exploration Statistics

Proposal 2.2.2 For CDCL SAT solving, generating conflicts at a faster rate is important, as each conflict generates a learned clause that prunes the search space. Learning high-quality clauses that have high pruning power is crucial.

I propose to collect conflict information associated with future search states, which are reached via exploration. My conjecture is that conflict information of future search states is important to make better branching decisions.

Methodology 2.2.2 I intend to collect the following conflict information of a future state reached via exploration: (a) an indication of whether a conflict has been generated in the reached state, and (b) For a state with conflict, the LBD score of the learned clause, which reflects the learned clause quality which is derived from that conflict.

2.3 Topic 3: Exploration based Branching and Polarity Heuristics

In this step, the statistics collected during exploration are used to make more efficient branching decisions. I propose to design branching and polarity selection heuristics from exploration statistics.

2.3.1 Branching Heuristics from Exploration Statistics

Proposal 2.3.1 There are several ways to use the exploration statistics to improve branching heuristics. The first research issue is (a) How to compute exploration scores from exploration statistics, and the second one is (b) How to use the exploration scores as part of an effective branching heuristic.

Methodology 2.3.1 (a) Given a random walk that produces a conflict, an unknown combination of random steps contribute to the generation of this conflict. The problem of computing a score for each random step is the problem of credit assignment which is studied in reinforcement learning. I intend to study and adopt methods from the reinforcement learning literature (such as [48]) to solve the problem. (b) The standard look-back based heuristics, such as VSIDS/LRB are very strong. It is very likely that a heuristic which combines standard CDCL branching heuristics scores with exploration scores, will work better than scores computed based on exploration alone. A similar approach was used in deterministic planning to combine standard planning heuristics with exploration [51].

2.3.2 Polarity Selection Heuristics from Exploration Statistics

Proposal 2.3.2 I propose to study variable polarity selection based on exploration. Here, the purpose of exploration is to identify polarity values for branching variables, which are better than the standard polarity selection heuristics.

Methodology 2.3.2 A variable v can be assigned one of two polarity values: *false* or *true*, which creates negative or positive literal assignment of v, respectively. I intend to design an exploration method which evaluates both polarities of v based on their conflict generation potential and the quality of the learned clauses derived from the conflicts discovered during this exploration.

I plan to use the same exploration method for branching variable selection as mentioned in Methodology 2.2.1 and 2.2.2, with minor changes due to the difference between a variable selection and polarity selection.

2.4 Topic 4: Solver Selection, Implementation, Experiments, and Evaluation

2.4.1 Solver Selection and Implementation

Proposal 2.4.1 I propose to implement exploration based CDCL SAT algorithms on top of modern state of the art CDCL SAT solver systems.

Methodology 2.4.1 The first step for the implementation is to select suitable baseline CDCL SAT solvers.

With the rapid development of new techniques, the landscape of CDCL SAT solving has become large and contains solvers with varying complexity. For example, a total of 106 solvers participated in the SAT competition-2018. Some solvers are relatively more complex than others, which can potentially range from those with elementary ingredients of CDCL SAT solving such as, Minisat-v2.2.0, to highly complex solvers, which combine many incremental changes, such as MapleLCMDistChronoBT.

To select a subset of solvers from this large and complex pool to extend them with our exploration approach, I intend to do the following: Start with a state of the art solver, which is moderately complex and successful, such as Glucose and move towards more complex solvers, which use multiple CDCL branching heuristics, such as the Maple systems.

2.4.2 Experiments and Evaluation

Proposal 2.4.2 I will perform large scale experiments with my exploration based CDCL SAT solvers to evaluate them against their baselines.

Methodology 2.4.2 For *experimentation*, I intend to use recent SAT competitions as the primary source of benchmark instances. These instances come from a diverse range of real-world application domains and are thus well suited for evaluation of CDCL SAT solvers. I plan to run my experiments on Linux workstations, provided by my research group. In the SAT competition, a solver can run a given instance for up to a maximum of 5000 seconds. I will use the same timeout limit for my experiments.

For *evaluation*, I plan to perform an apple-to-apple comparison between each baseline solver and its exploration based extension. I plan to use three metrics for evaluation:

- Number of solved instances that a given solver solves from a fixed test set.
- Average solving time, the total time taken to solve all solved instances, divided by the count of solved instances.

- In SAT competitions, the PAR-2 score[2] is used for the evaluation of participating solvers. It is defined as the sum of all runtimes for solved instances + 2*timeout for unsolved instances. Lower scores are better. The PAR-2 scheme combines two metrics into a single number. Better solvers have smaller PAR-2 score.

2.5 Topic 5: Identify Effective Domains For Exploration

Proposal 2.5 Exploration of a search space can be viewed as a type of look-ahead approach that probes future search states in order to take better decisions at the current state. Deterministic look-ahead has been shown to be effective for extremely hard benchmark domains [21, 29].

A natural question is whether there are any benchmark domains with special structure, which can especially benefit from exploration? If there are such benchmarks, then what properties of these benchmarks do exploration-based CDCL solvers exploit?

Methodology 2.5 I intend to study the SAT literature to find benchmarks which are considered to be difficult for modern CDCL SAT solvers. One example of such a benchmark is from an encryption domain, namely the preimage attack on SHA-1 cryptographic benchmarks [36]. Another example of a hard SAT benchmark is finding a Hamiltonian cycle in a complete graph [28].

For the selected hard benchmarks, I plan to use instance generators that are available, or write new ones otherwise.

2.6 Topic 6: Analysis of Experimental Results to Reveal Insights

Proposal 2.6 I propose to perform analysis of the experimental results for the exploration based methods that I develop. Analysis of the experimental results is crucial to reveal novel insights, which may drive further developments in this direction.

2.6.1 Performance of Random Walks

I propose to analyze the experimental data to answer the following questions: (I) For a given exploration algorithm and a problem set, how many conflicts per random step does exploration uncover? (II) What is the average quality of learned clauses that are derived from these conflicts?

2.6.2 Explanation of Performance Gain/Decline

Given an exploration method and a problem set, it is important to understand the reasons why that exploration method is effective or ineffective on that problem set. Such analysis will not only reveal new insights, but also drive further improvement. I am proposing to pursue this question for every exploration method I will be developing.

[2]https://baldur.iti.kit.edu/sat-competition-2017/index.php

2.6.3 Understanding behavior of exploration based method on SAT and UNSAT instances

The empirical study of modern CDCL SAT solvers on SAT competition benchmarks reveals that they behave differently on SAT and UNSAT instances. For example, for SAT instances, it is typical for a CDCL SAT solver to create a lot of unit propagations *suddenly*, which is not usually the case with UNSAT instances [8]. In [37], it is shown that the quality of learned clauses has different effect on solving SAT and UNSAT instances. While clauses with moderately high LBD score (moderate quality) help the efficient solving of UNSAT instances, learning only critically low LBD (high quality) clauses helps in solving SAT instances efficiently.

I propose: (I) to study the differences in behavior of exploration based CDCL SAT solvers in solving SAT and UNSAT instances, and (II) based on this understanding, design exploration strategies that improve CDCL SAT solving separately for SAT and UNSAT instances.

Methodology 2.6 For the research issues from 2.6.1 to 2.6.3, I plan to collect and use various metrics data, such as GLR, average LBD, number of decisions taken by the search, number of generated glue clauses etc. for each instance. I plan to write python/matlab scripts to further analyze the collected data to understand the behavior of my developed methods.

3 Related Work

Randomized exploration in SAT is used in local search methods such as GSAT [44] and WalkSAT [43]. The *Satz* algorithm [24] heuristically selects a variable x, then performs two separate unit propagations with x and $(\neg x)$ respectively, in order to evaluate the potential of x. Modern CDCL SAT solvers include exploration components such as a small percentage of random variable selection [15]. Exploration can make a search process more robust by allowing an escape from *early mistakes* caused by inaccurate heuristics [50]. Examples of recently popular exploration methods in search are Monte Carlo Tree Search (MCTS) [11], and the random walk techniques used in both classical [33, 51] and motion planning [23]. The automated theorem prover Monte-Cop [16] uses MCTS techniques to guide the proof search. These techniques motivated our work on random exploration in CDCL SAT.

In contrast to the DPLL and CDCL SAT framework, which employ depth-first search, in [40], a SAT solver named `UCTSAT` is proposed that employs Monte-Carlo style UCT (Upper Confidence bounds applied to Trees) search. Given a SAT instance, `UCTSAT` repeatedly invokes UCT search at the root and incrementally builds a SAT search tree based on the value estimation of the search states. The value estimation of a state is derived by using the outcomes of random samplings of states that were performed in the previous iterations.

In [6], Audemard et. al. have proposed a hybrid solver `SATHYS`. `SATHYS` employs a CDCL SAT solver and a local search SAT solver. The search alternates between these two solvers, where the solvers exchange relevant information. In this hybrid solver, the local search solver helps the CDCL solver by identifying the most promising literal assignment to branch on and the CDCL search process guides the local search process to flee from local minima.

The recent Conflict History Based (CHB) [25] and Learning Rate Based (LRB) [26] heuristics model variable selection as a Multi-Armed Bandit (MAB) problem, which is solved using the Exponential Recency Weighted Average (ERWA) algorithm. Both of these heuristics compute rewards from the conflict history of unassigned variables, in order to rank them.

In [27], Liang et. al. propose a look-ahead based branching heuristic that greedily maximizes the GLR score. The method in [5] uses preprocessing to identify a set of highly relevant learned clauses, which are likely to have LBD score 2. These clauses are added to the original formula before search.

4 Finished Work

I have made progress on some of the proposed research topics, which I describe in the following subsections.

4.1 Empirical Properties of CDCL Branching Heuristics

This section describes my progress on Topic 1 - *Empirical Properties of CDCL Branching Heuristics and Polarity Heuristics.*

4.1.1 Conflict Depression in VSIDS Branching Heuristic

So far, I have studied the VSIDS branching heuristic to find answers of the questions from Proposal 2.1.1. I have formulated the novel notion of Conflict Depression (CD) phase, a pathological phase of CDCL SAT search, which is defined below. I have found experimental evidence that with VSIDS, CD phases occur frequently.

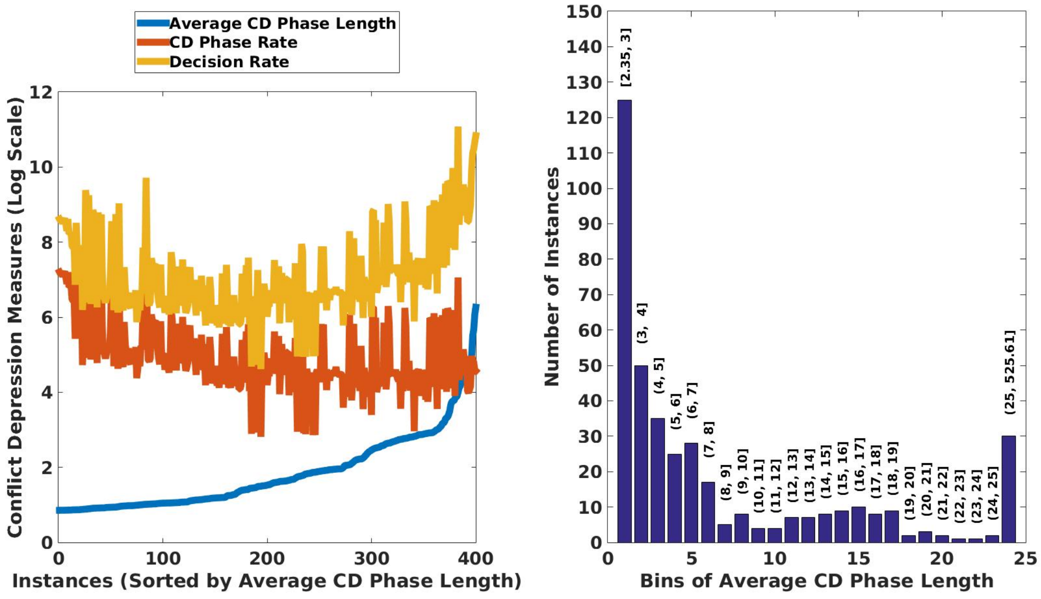
Notions and Definitions Consider a run of a CDCL SAT solver S which makes a total of d decisions. In each decision, a variable is selected according to a branching heuristic. Each decision i ($0 < i \leq d$) leads to some number $c_i \geq 0$ of conflicts. We can represent the conflict history of the search by the sequence of c_i. We define a *conflict depression (CD) phase* as a sequence of one or more consecutive decisions with no conflict. We further define the *length* of a CD phase as the number of decisions in it. For example, consider the conflict history of decisions (1,**0**,**0**,**0**,**0**,4,2,1,**0**,1,**0**,**0**), where a number represents the number of conflicts at that decision. This history contains 3 CD phases: one starting at decision 2 with length 4, one starting at decision 9 with length 1, and one at the end with length 2.

Suppose S takes a total of d decisions, encounters u CD phases and makes r restarts. We define the *Decision Rate* (DR) as d/r and the *CD phase Rate* (CDR) as u/r. We also define the *Fraction of Decisions with Conflicts* (FDC) as a measure related to, but different from the Global Learning Rate (GLR): FDC is the fraction of decisions which produce at least one conflict. This measure counts decisions with conflicts, not conflicts, which is a different measure since some decisions cause multiple conflicts.

Experiments I performed an experiment with glucoseLCM, a state of the art CDCL SAT solver from the Glucose family for 400 instances from SAT Competition-2018. We call these instances the *Test Set*. I instrumented glucoseLCM to collect statistics on DR, CDR, Average CD phase length, GLR, and FDC.

The left plot of Figure 1 shows the Decision Rates (DR), CD phase Rate (CDR) and average CD phase length (in log scale) for the instances of Test Set instances are sorted by average CD phase length. We observe that the average CD phase length is short for most instances, but still consists of multiple decisions. Furthermore, irrespective of their average CD phase length, for all-most all of the instances CD phases occur at a high rate w.r.t. the decision rates.

Figure 1: Conflict Depression plots for Test Set with glucoseLCM



The histogram on the right side of Figure 1 shows the distribution of the average length of CD phases. This average ranges from 2.35 to 525.61. 125 instances have very short length (at most 3, leftmost bin). The distribution is heavy-tailed, with over 30 instances with average length greater than 25 (rightmost bin).

Overall, the data indicates that for glucoseLCM on the Test Set, conflict depressions occur frequently and often last over multiple decisions (high average CD phase length), which is a serious problem for search efficiency.

In Table 1, column C shows that the average GLR values for all three types of problems are close to 0.5, so the number of conflicts is about half the number of decisions. In contrast, the FDC values in column D are much lower, averaging 0.2506 over all instances. Therefore, about 75% of all decisions do not produce any conflict.

To summarize, my analysis shows that the typical search behavior alternates between high-conflict bursts and longer depression phases. I conjecture that the beginning of a CD phase corresponds to a new region in the search space where VSIDS scores are not a good predictor of a variable's future performance. In such a phase, VSIDS fails to generate any

Table 1: Statistics for Test Set on GLR and FDC with glucoseLCM

A: Type	**B: #Instances**	**C: GLR**	**D: FDC**
Satisfiable	95	0.4915	0.2562
Unsatisfiable	97	0.4718	0.2543
Unsolved	208	0.5060	0.2543
All	400	0.4943	0.2506

conflicts. No conflict means no learned clauses, and the solver only performs truth value propagations.

4.1.2 Measuring the accuracy of the Phase-Saving polarity heuristic

This experimental study shows that the phase-saving heuristic has some degree of inaccuracy in polarity estimation, which answers the first question mentioned in Methodology 2.1.2. First, I present some notions and definitions, which are used in this experiment. The notions developed for this study are at their preliminary stage and may require further refinements.

Definitions Consider a run of a CDCL SAT solver on a SAT formula $\mathcal{F}$. At the end state E of the search, let $assign(\mathcal{F})$ be the set of variables which have been assigned a value. $assign(\mathcal{F})$ represents the endpoint of a branch of the whole search tree for $\mathcal{F}$, where a SAT (resp. UNSAT) formula $\mathcal{F}$ is already satisfied (resp. proved to be unsatisfiable).

The interpretation of the variable assignments for the variables in $assign(\mathcal{F})$ are different for SAT and UNSAT instances. Let $unassigned(\mathcal{F}) = vars(\mathcal{F}) \setminus assign(\mathcal{F})$.

- For a SAT instance $\mathcal{F}$, in the branch for $assign(\mathcal{F})$, the variables in $unassigned(\mathcal{F})$ do not need to be assigned to prove the satisfiability of $\mathcal{F}$.
- For an UNSAT instance $\mathcal{F}$, the assignments in $assign(\mathcal{F})$ are sufficient to generate a conflict that proves the unsatisfiability of $\mathcal{F}$ in this branch.

Polarity Error of a Variable: We are interested in quantifying the polarity error for the variables in $assign(\mathcal{F})$ as with these assignments, $\mathcal{F}$ becomes satisfiable (if $\mathcal{F}$ is SAT) or a conflict is generated that proves the unsatisfiability of $\mathcal{F}$ (if $\mathcal{F}$ is UNSAT). For each variable $v \in assigns(\mathcal{F})$, let $p(v)$ and $n(v)$ be the number of times that v is assigned positively and negatively during the course of the search, respectively.

With given threshold $0.5 < \theta \leq 1$, we say that v has a *positive polarity error*, if at E, $v = false$ and

$$\frac{p(v)}{p(v) + n(v)} > \theta$$

Similarly, we say that v has a *negative polarity error*, if at E, $v = true$ and

$$\frac{n(v)}{p(v) + n(v)} > \theta$$

.

Intuitively, a variable v has positive polarity error (resp. negative polarity error), if at the end state E of the search, v is assigned to *false* (resp. *true*), but was assigned to positive polarity (resp. negative polarity) more often than negative polarity (resp. positive polarity) in the course of the search.

Polarity Error of a Formula: At E, let $0 \leq pe \leq |assign(\mathcal{F})|$ and $0 \leq ne \leq |assign(\mathcal{F})|$ be the total number of variables with positive and negative polarity errors, respectively. We define *positive polarity error* for $\mathcal{F}$, $pe(\mathcal{F})$ as $\frac{pe}{|assign(\mathcal{F})|}$. Similarly, we define *negative polarity error* for $\mathcal{F}$, $ne(\mathcal{F})$ as $\frac{ne}{|assign(\mathcal{F})|}$. The *combined polarity error* for $\mathcal{F}$ at E, $pne(\mathcal{F})$ is defined as $\frac{pe+ne}{|assign(\mathcal{F})|}$.

Table 2: Statistics for 400 instances from SAT competition-2018 on Average $pe(\mathcal{F})$, $ne(\mathcal{F})$ and $pne(\mathcal{F})$ with glucoseLCM

A: Type	**B: #Instances**	**Average C:**$pe(\mathcal{F})$	**D: Average** $ne(\mathcal{F})$	**E: Average** $pne(\mathcal{F})$
Satisfiable	95	0.01987	0.1217	0.1416
Unsatisfiable	97	0.02526	0.1872	0.2124
Combined	192	0.0226	0.1553	0.1779

Experiments Table 2 shows the average values for $pe(\mathcal{F})$, $ne(\mathcal{F})$ and $pne(\mathcal{F})$ with glucoseLCM for 192 instances from the Test Set, separately for SAT and UNSAT instances. We ran this experiment with a timeout of 5000 seconds. For this experiment, we set $\theta = 0.75$.

On average, over the 192 instances, the phase saving heuristic in glucoseLCM has positive polarity error of 0.0226 (Column C) and negative polarity of error of 0.1553 (Column D). The combined average polarity error is 0.1779 (Column E).

This is an interesting observation, as the phase saving heuristic has many more negative polarity errors than positive polarity errors. My conjecture on this imbalance in error for these two polarities is as follows: glucoseLCM initializes the polarity for each variable of a given formula to - and during the course of the search, it preserves the initial negative polarity values for most of the variables. Near the end of the search, just before finding the final assignment, for a good number of variables (almost 16% Column D, Table 2), negative polarity switches to positive polarity.

To summarize, the phase saving heuristic has a moderate degree of inaccuracy in assigning polarity values to branching variables, and there is room for improvement.

4.2 Exploration Based CDCL Solver and Branching Heuristics

Here, I present my progress on the proposals presented in Sections 2.2 and 2.3.

4.2.1 The *expSAT* algorithm

During a CD phase, VSIDS is ineffective. Is it possible to correct the course of the search by identifying promising variables that are currently under-ranked by VSIDS? I have formulated a solver framework named *expSAT*, which performs random exploration that probe the future search space. The goal is to discover branching variables that are likely to lead to *good* conflicts from which important clauses can be learned.

Let $\mathcal{F}$ be a CNF SAT formula. In addition to $\mathcal{F}$, *expSAT* also accepts four *exploration parameters* nW, lW, p_{exp} and ω, where $1 \leq nW, lW \leq uVars(\mathcal{F})$, $0 < p_{exp}, \omega \leq 1$. These parameters control the exploration aspects of *expSAT*. The details of these parameters are given below.

Given a CDCL SAT solver, *expSAT* modifies it as follows:
(I) Before each branching decision, if a substantially large CD phase[3] is detected then with probability p_{exp}, *expSAT* performs an *exploration episode*, consisting of a fixed number nW of random walks. Each walk consists of a limited number of *random steps*. Each such step consists of (a) the uniform random selection of a currently unassigned *step variable* and assigning a boolean value to it using a standard CDCL *polarity* heuristic, and (b) Unit Propagation (UP). A walk terminates either when a conflict occurs during UP, or after a fixed number lW of random steps have been taken. Figure 2 illustrates an exploration episode amid a CD phase.
(II) Each variable v that participates in any of the nW walks receives an exploration score, which is the average of the *walk-scores* of v. The walk-score $ws(v)$ for v is: $\frac{\omega^d}{lbd(c)}$, if the walk where v participates, ends with a conflict and 0, otherwise. d is the distance between the step for v and the step at which the conflict has occurred, $0 < \omega \leq 1$ is the decay factor and $lbd(c)$ is the LBD score of the learned clause c, which is derived from the conflict that terminates the walk. We assign credit to all the step variables in a walk that ends with a conflict and give higher credit to variables closer to the conflict.
(III) The novel branching heuristic `expVSIDS` adds VSIDS score and *expScore* of the unassigned variables. A variable v^* with maximum combined score is selected for branching.
(IV) All other components remain the same as in the underlying CDCL SAT solver.

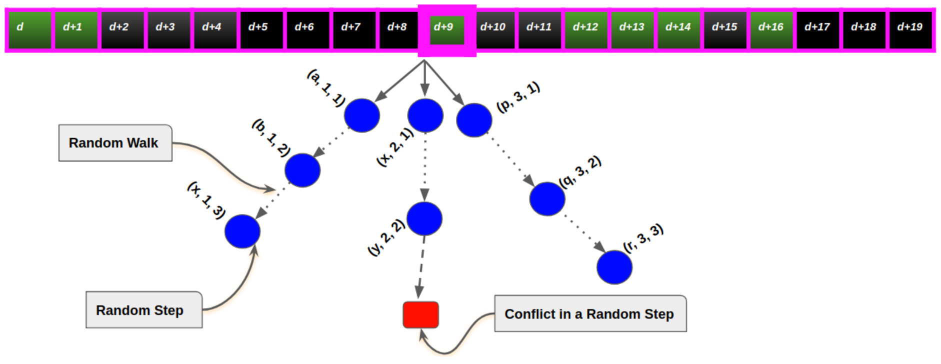
Example: Using the three random walks of Figure 2, we show how to compute ws and *expScore* of variables. Only the second walk produces a conflict. Let c be the derived clause from this conflict, with $lbd(c) = m$.

The walk and exploration scores for all variables participating in the first and third random walk are 0. The variables x and y which participate in the second walk receive non-zero walk and exploration scores: $ws(y) = \frac{\omega^0}{m} = \frac{1}{m}$ and $ws(x) = \frac{\omega^1}{m}$. Since y only appears in this walk, but x appears in two walks, the exploration scores of y and x are $\frac{1}{m}$, and $(\frac{\omega}{m})/2$, respectively.

The Parameter Adaptation Algorithm: A parameter setting that is effective for one instance may not be effective for another. We use an adaptive algorithm *paramAdapt* to dynamically control when to trigger exploration episodes, and how much exploration to perform in an exploration episode. The three exploration parameters p_{exp}, nW, and lW are adapted between CDCL restarts based on the search behavior.

[3]Given a run of a solver on a given SAT instance, let R be the measure $\frac{\#decisions\_without\_conflicts}{\#decisions\_with\_conflicts}$, where *#decisions_with_conflicts* (resp. *#decisions_without_conflicts*) are the number of decisions with conflicts (resp. without conflicts) encountered by the search so far. $R + 1$ is the average number of decisions taken until one generated a conflict. In *expSAT*, we call a CD phase *substantial*, if its length is greater than R.

Figure 2: The 20 adjacent cells denote 20 consecutive decisions starting from the d^{th} decision, with $d > 0$, where a green cell denotes a decision with conflicts and a black cell denotes a decision without conflicts. Say that amid a CD phase, just before taking the $(d + 9)^{th}$ decision, *expSAT* performs an exploration episode via 3 random walks each limited to 3 steps. The second walk ends after 2 steps, due to a conflict. A triplet (v, i, j) represents that the variable v is randomly chosen at the j^{th} step of the i^{th} walk.



4.2.2 Baseline Selection, Implementation and Experiments

This subsection presents my progress on the proposal described in Section 2.4, *Solver Selection, Implementation, Experiments and Evaluation.*

Solver Selection and Implementation I have selected glucoseLCM and Maple_CM [4] as my baseline CDCL SAT solvers. My implementation of the *expSAT* approach in glucoseLCM and Maple_CM has resulted in two new CDCL SAT solvers, expGLCM and expM_CM.

Maple_CM uses two heuristics: *LRB* and *VSIDS*. Based on the activation of these heuristics, a run in Maple_CM is divided into two stages: In stage 1, which lasts for the first 2500 seconds of a run, it uses LRB. In Stage 2, which follows afterward, it uses VSIDS. In expM_CM, we apply the *expSAT* approach only at stage 2.

Evaluation of the *expSAT* approach I compare the performance of the *expSAT* systems against their baselines on the 400 instances from Test Set.

- **glucoseLCM VS expGLCM**
  - expGLCM solves 9 more instances (total solved 201) than glucoseLCM (total solved 192), with a strong performance for SAT instances (solves 10 more) and slightly weaker performance for UNSAT ones (solves 1 fewer).
  - The average solving time for expGLCM is lower (894.06 seconds for 201 instances) than the average solving for glucoseLCM (1018.86 seconds for 192 instances).
  - Overall, expGLCM achieves a lower (better) PAR-2 score of 2169778.32, compared to glucoseLCM with 2275621.74.

[4]Both solvers are available at: http://sat2018.forsyte.tuwien.ac.at/solvers/main_and_glucose_hack/

- **Maple_CM VS expM_CM**
  - Maple_CM and expM_CM solves 228 and 230 instances, respectively. Among thse solved instances, 210 instances are solved in stage 1, where runs in both solvers are identical (expM_CM does not perform exploration there). In stage 2, expM_CM solves 2 additional instances (both SAT).
  - The average solving time for expM_CM (767.32 seconds over 230 solved instances) is slightly higher than the average solve-time of Maple_CM (744.49 seconds over 228 instances).
  - expM_CM achieves a slightly lower (better) PAR-2 score of 1876483.89 compared to Maple_CM, which has a PAR-2 score of 1889744.09. Overall, expM_CM performs slightly better than Maple_CM, which is reflected in its lower PAR-2 score.

Overall, this experiments pointed out the strength of the *expSAT* approach in solving satisfiable instances from the latest SAT competition.

4.3 Identification of Benchmark Domain

In this Section, I report my progress on the proposal described in Section 2.5 *Identification of Benchmark Domain*.

4.3.1 The SHA-1 preimage attack benchmark

My search for specific benchmarks where the *expSAT* approach is particularly effective, has identified *SHA-1 preimage attack*, a benchmark from an encryption domain [36].

The SHA-1 cryptographic function is known to be difficult to invert [22]. A source of difficulty in inversion comes from a feature of this function called *preimage resistance: Given an encryption function f and a hash H, it is computationally difficult to find an input message M, such that $f(M) = H$.* Preimage attack attempts to find such a message M.

This benchmark is known to be challenging for current CDCL SAT solvers.

Instance Generation: In [36], Nossum has developed an instance generator for this benchmark. The difficulty of instances is controlled by three parameters, *rounds*, *hash-bits* and *message-bits*. For generating hard instances, the author of [36] has recommended the following value ranges of these parameters: *rounds* between 23-30, *hash-bits* between 66-97, and *message-bits* 0. I have generated 40 hard SHA-1 preimage attack instances by using this generator with these value ranges of these parameters.

Experiments and Evaluation: I have evaluated the performance of expGLCM against its baseline glucoseLCM. We set a time limit of 36000 seconds per instance.

To put the hardness of these cryptographic instances into perspective, we performed experiments with the winner of the main track of SAT-2018, MapleLCMDistChronoBT. It solves only 1 instance out of these 40 instances.

expGLCM solves 3 satisfiable instances with an average time of 6680s, and glucoseLCM solves 2 satisfiable instances with an average time of 12277s.

In this experiment, expGLCM is clearly the winner.

5 Remaining Work

For the rest of my PhD program, I aim to work on the following research issues.
1. **Understanding LRB and extending it with the *expSAT* approach** So far, I have extended one state of the art CDCL branching heuristic VSIDS, which has resulted in `expVSIDS`. I plan to understand the differences between VSIDS and LRB, and develop `expLRB`, which will be an extension of LRB with *expSAT* ideas. I expect that all the components of my *expSAT* framework will stay the same, except the following:

- One important question is how to design the activity score increment method for LRB. VSIDS increases the activity score by using a bumping factor, which is same for all the variables at a given state of the search. In contrast, the activity score increment method of LRB does not use an explicit bumping factor. Instead, it uses reward values derived from past conflict history, which are unique for each variable. To successfully extend LRB with an exploration method, I will need to develop a different score bumping method for LRB.

I also plan to evaluate the *expSAT* approach with `expLRB` by implementing it on state of the art CDCL SAT solvers, such as Maple-based systems.
2. **Combining `expVSIDS` and `expLRB`** Most of the state of the art CDCL SAT solvers, such as Maple_CM, use a combination of VSIDS and LRB, which are activated in different phases of the search. I plan to extend those solvers with the *expSAT* approach, which will replace VSIDS and LRB with `expVSIDS` and `expLRB`, respectively.
3. **Polarity Selection Heuristic based on Exploration** I plan to design an exploration based polarity selection heuristic within the *expSAT* framework.

- Extend the random step component of the *expSAT* method to include the polarity of the selected random step variable.
- Extend the exploration score computation and score assignment method to include an exploration polarity score.
- Design polarity selection heuristics by using the exploration scores.

I also plan to evaluate the developed exploration based polarity heuristics by implementing them on the state of the art CDCL SAT solvers, such as Glucose, Maple_CM. I expect that will be easy to do across these solvers as they have similar code structure for polarity selection.
4. **Identify more exploration friendly benchmarks** I plan to identify more benchmarks, where the *expSAT* approach will show strong performance. My tentative plan is to include the following hard domains for this study: *encryption*, *planning* and *graph theory*. After identifying the hard benchmark domains, I plan to further limit my selection based on the notion of CD phase. I plan to select benchmarks, for which the average length of the CD phase is relatively high. This criterion is based on the following intuition:

- Given a problem instance and a CDCL SAT solver, frequent occurrence of CD phases with high average length indicates that the branching heuristics for that CDCL SAT solver are highly ineffective. For such benchmarks, exploration may help the solver to generate conflicts at a faster rate and improve its performance.

5. **Analysis of Empirical Results to Reveal Insights** For every method, I plan to perform extensive analysis of the experimental data to reveal insights to further improve those developed methods.

6 Expected Contributions, Contributions So Far and Timeline

6.1 Expected Contributions

The expected contributions of my PhD research are:

- Develop a series of full-fledged exploration based CDCL SAT solvers, with concrete understanding of the developed methods.
- My developed SAT solvers will push the state of the art of CDCL SAT solving by beating the current state of art solvers on recent SAT competition benchmarks.
- Identify benchmark domains where my developed method is particularly effective. My method should be adopted by the SAT community to solve benchmark instances from those domains.
  - If I can identify benchmarks of industrial strength, such as, encryption, planning and hardware design verification, then the relevant industries will also benefit.
- The central components for reasoning engines in other declarative problem solving paradigms, such as SAT Modulo Theory (SMT) [35] and Answer Set Solving (ASP) [17] are based on CDCL techniques. An improved CDCL SAT solving algorithm will impact these two paradigms as well.

6.2 Contributions So Far

Until now, my PhD research has made the following contributions:

6.2.1 Peer Reviewed Publications

(I) A student abstract in AAAI-2018, reports results for a preliminary version of the *expSAT* approach. This work was selected as a **finalist**[5] to participate in the 3 minutes paper presentation contest within the student track.

[5]Only the top 18% of the accepted abstracts were selected to be the finalists.

- Md. Solimul Chowdhury, Martin Müller, Jia-Huai You: Preliminary Results on Exploration-Driven Satisfiability Solving. AAAI 2018: 8069-8070. (**Student Abstract Finalist**)

(II) I have developed a new CDCL branching heuristic named *Glue Bumping (GB)*, which prioritizes the selection of variables that appear in glue clauses. The following paper based on the GB method has been accepted for CP 2019.

- Md Solimul Chowdhury, Martin Müller and Jia-Huai You. Exploiting Glue Clauses to Design Effective CDCL Branching Heuristics. CP 2019. (To Appear)

6.2.2 SAT Competition Results

I have participated in SAT competition-2018 and SAT Race-2019. The results are:

- **SAT competition-2018:** 41 solvers participated in the main track of SAT competition-2018. My solver expMC_VSIDS_LRB_Switch_2500 (expMC)[6] **secured the 6th position** in the SAT track [3].
- **SAT Race-2019:** 55 solvers participated in the SAT Race-2019. My solver expMaple_CM_GCBumpOnlyLRB, which integrates *expSAT*[7] and Glue Bumping methods became **first runner-up in UNSAT track** with respect to PAR-2 scoring and **second runner-up in SAT+UNSAT combined track** with respect to SCR scoring (based on total number of solved instances) [4].

6.2.3 Pushing the Boundary for Hierarchical Task Network Planning

Hierarchical Task Network (HTN) is a technique in AI planning that describes a given planning problem by the specification of its initial state and a task network as the objective to be achieved [34, 18]. In a recent work [9], Behnke et. al. have proposed a new SAT encoding for HTN planning. For experimental evaluation, the authors have used 6 SAT solvers from SAT Competition-2018, including expMC and 13 planners, which are the state of the art planners for the HTN domain. In their experiment, expMC was the best performing system among these 19 systems. Thus the *expSAT* approach has contributed to pushing the boundary for HTN planning.

6.3 Timeline

In Table 3, I show a tentative timeline for the activities outlined above over the rest of my PhD program. In case of overlapping timelines for two activities, the Duration column is left blank for the activity with smaller duration.

[6]This solver was developed based on a different version of the *expSAT* approach (does not use the notion of CD phase) than the version which is presented in this document.

[7]This solver implements the *expSAT* algorithm presented in this document.

Table 3: Timeline of Activities

Activity	**Start Date**	**End Date**	**Duration**
1. Understand LRB and extend it with the *expSAT* approach	August, 2019	October, 2019	3 Months
2. Write a paper for AAAI-2020	August, 2019	August, 2019	-
3. Combine `expVSIDS` and `expLRB`	November, 2019	January, 2020	3 Months
4. Write a paper for IJCAI/SAT/CP/SocS-2020	January, 2020	January, 2020	-
5. Develop a Polarity Selection Heuristic based on Exploration	February, 2020	September, 2020	8 Months
6. Prepare and Submit solvers for SAT Competition-2020	March, 2020	March, 2020	-
7. Write a paper for AAAI-2021	August, 2020	August, 2020	-
8. Identify and Evaluate Benchmark Domains	October, 2020	January, 2021	4 Months
9. Write a paper for IJCAI/SAT/CP/SocS-2021	January, 2021	January, 2021	-
10. Write Doctoral Dissertation	February, 2021	July, 2021	6 Months
11. Prepare and Submit solvers for SAT Competition-2021	March, 2021	March, 2021	-
12. Analyze the Empirical Results to Derive Insights	July, 2019	July, 2021	-
13. Write a journal paper for AIJ/JAIR	February, 2021	July, 2021	-
12. PhD Defense	August, 2021	August, 2021	-
Total			24 Months

References

[1] Glucose's home page, *http://sat2018.forsyte.tuwien.ac.at/solvers/main_and_glucose_hack/*, accessed date: 2019-04-25.

[2] MapleSAT: Combining machine learning and deduction in SAT solvers, `https://sites.google.com/a/gsd.uwaterloo.ca/maplesat/`, accessed date: 2019-04-25.

[3] SAT Competition-2018 results, *http://sat2018.forsyte.tuwien.ac.at/index.php?cat=results*, accessed date: 2019-07-16.

[4] SAT Race-2019 results, `http://sat-race-2019.ciirc.cvut.cz/downloads/satrace19slides.pdf`, accessed date: 2019-07-16.

[5] Carlos Ansótegui, Jesús Giráldez-Cru, Jordi Levy, and Laurent Simon. Using community structure to detect relevant learnt clauses. In *Proceedings of SAT 2015*, pages 238–254, 2015.

[6] Gilles Audemard, Jean-Marie Lagniez, Bertrand Mazure, and Lakhdar Sais. Boosting local search thanks to CDCL. In *Proceedings of LPAR-17*, pages 474–488, 2017.

[7] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In *Proceedings of IJCAI 2009*, pages 399–404, 2009.

[8] Gilles Audemard and Laurent Simon. Refining restarts strategies for SAT and UNSAT. In *Proceedings of CP 2012*, pages 118–126, 2012.

[9] Gregor Behnke, Daniel Höller, and Susanne Biundo. Bringing order to chaos - a compact representation of partial order in sat-based htn planning. In *AAAI 2019*, 2019.

[10] A. Biere, A. Biere, M. Heule, H. van Maaren, and T. Walsh. *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications.* IOS Press, Amsterdam, The Netherlands, 2009.

[11] Cameron Browne, Edward Jack Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez Liebana, Spyridon Samothrakis, and Simon Colton. A survey of Monte Carlo Tree Search methods. *IEEE Trans. Comput. Intellig. and AI in Games*, 4(1):1–43, 2012.

[12] Cristian Cadar, Vijay Ganesh, Peter M. Pawlowski, David L. Dill, and Dawson R. Engler. EXE: automatically generating inputs of death. In *Proceedings of CCS 2006*, pages 322–335, 2006.

[13] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.

[14] Martin Davis, George Logemann, and Donald W. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.

[15] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *Proceedings of SAT 2003, Selected Revised Papers*, pages 502–518, 2003.

[16] Michael Färber, Cezary Kaliszyk, and Josef Urban. Monte Carlo tableau proof search. In *Proceedings of CADE 2017*, pages 563–579, 2017.

[17] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. *Answer Set Solving in Practice.* Morgan & Claypool Publishers, 2012.

[18] Ilce Georgievski and Marco Aiello. An overview of hierarchical task network planning. *CoRR*, abs/1403.7426, 2014.

[19] Aarti Gupta, Malay K. Ganai, and Chao Wang. SAT-based verification methods and applications in hardware verification. In *Proceedings of SFM 2006*, pages 108–143, 2006.

[20] Shai Haim and Marijn Heule. Towards ultra rapid restarts. *CoRR*, abs/1402.4413, 2014.

[21] Marijn Heule, Oliver Kullmann, and Victor W. Marek. Solving very hard problems: Cube-and-conquer, a hybrid SAT solving method. In *Proceedings of IJCAI 2017*, pages 4864–4868, 2017.

[22] Roman Jasek, Libor Sarga, and Robert Benda. Security review of the SHA-1 and MD 5 cryptographic hash algorithms. WSEAS Press, 2013.

[23] Steven M LaValle. *Planning algorithms.* Cambridge University Press, 2006.

[24] Chu Min Li and Anbulagan. Look-ahead versus look-back for satisfiability problems. In *Proceedings of CP 1997*, pages 341–355, 1997.

[25] Jia Hui Liang, Vijay Ganesh, Pascal Poupart, and Krzysztof Czarnecki. Exponential recency weighted average branching heuristic for SAT solvers. In *Proceedings of AAAI 2016*, pages 3434–3440, 2016.

[26] Jia Hui Liang, Vijay Ganesh, Pascal Poupart, and Krzysztof Czarnecki. Learning rate based branching heuristic for SAT solvers. In *Proceedings of SAT 2016*, pages 123–140, 2016.

[27] Jia Hui Liang, Hari Govind, Pascal Poupart, Krzysztof Czarnecki, and Vijay Ganesh. An empirical study of branching heuristics through the lens of global learning rate. In *Proceedings of SAT 2017*, pages 119–135, 2017.

[28] Fangzhen Lin and Yuting Zhao. ASSAT: computing answer sets of a logic program by SAT solvers. *Artif. Intell.*, 157(1-2):115–137, 2004.

[29] Guohua Liu and Jia-Huai You. Adaptive lookahead for answer set computation. In *Proceedings of ICTAI 2007*, pages 230–237, 2007.

[30] Fabio Massacci and Laura Marraro. Logical cryptanalysis as a SAT problem. *J. Autom. Reasoning*, 24(1/2):165–203, 2000.

[31] Jean Méhat and Tristan Cazenave. Combining UCT and nested Monte Carlo search for single-player general game playing. *IEEE Trans. Comput. Intellig. and AI in Games*, 2(4):271–277, 2010.

[32] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of DAC 2001*, pages 530–535, 2001.

[33] Hootan Nakhost and Martin Müller. Monte Carlo exploration for deterministic planning. In *Proceedings of IJCAI 2009*, pages 1766–1771, 2009.

[34] Dana Nau, Malik Ghallab, and Paolo Traverso. *Automated Planning: Theory & Practice.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[35] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). *J. ACM*, 53(6):937–977, 2006.

[36] Vegard Nossum. Instance generator for encoding preimage, second-preimage, and collision attacks on SHA-1. In *Proceedings of SAT Competition*, pages 119–120, 2013.

[37] Chanseok Oh. Between SAT and UNSAT: the fundamental difference in CDCL SAT. In *Proceedings of SAT 2015*, pages 307–323, 2015.

[38] Knot Pipatsrisawat and Adnan Darwiche. A lightweight component caching scheme for satisfiability solvers. In *Proceedings of SAT 2007*, pages 294–299, 2007.

[39] Knot Pipatsrisawat and Adnan Darwiche. On the power of clause-learning SAT solvers with restarts. In *Proceedings of CP 2009*, pages 654–668, 2009.

[40] Alessandro Previti, Raghuram Ramanujan, Marco Schaerf, and Bart Selman. Monte Carlo style UCT search for boolean satisfiability. In *Proceedings of AI*IA 2011*, pages 177–188, 2011.

[41] Jussi Rintanen. Engineering efficient planners with SAT. In *Proceedings of ECAI 2012*, pages 684–689, 2012.

[42] Christopher D. Rosin. Nested rollout policy adaptation for Monte Carlo Tree Search. In *Proceedings of IJCAI 2011*, pages 649–654, 2011.

[43] Bart Selman, Henry A. Kautz, and Bram Cohen. Local search strategies for satisfiability testing. In *Cliques, Coloring, and Satisfiability, Proceedings of a DIMACS Workshop 1993*, pages 521–532, 1993.

[44] Bart Selman, Hector J. Levesque, and David G. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of AAAI 1992.*, pages 440–446, 1992.

[45] João P. Marques Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. Computers*, 48(5):506–521, 1999.

[46] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[47] Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT solvers to cryptographic problems. In *Proceedings of SAT 2009*, pages 244–257, 2009.

[48] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning.* MIT Press, Cambridge, MA, USA, 2nd edition, 2018.

[49] Fan Xiao, Mao Luo, Chu-Min Li, Felip Manyá, and Zhipeng Lu. MapleLRB_LCM, Maple_LCM, Maple_LCM_Dist, MapleLRB_LCMoccRestart and Glucose3.0+width in sat competition 2017. In *Proceedings of SAT Competition 2017*, pages 22–23, 2017.

[50] Fan Xie, Martin Müller, Robert Holte, and Tatsuya Imai. Type-based exploration with multiple search queues for satisficing planning. In *Proceedings of AAAI 2014*, pages 2395–2402, 2014.

[51] Fan Xie, Hootan Nakhost, and Martin Müller. Planning via random walk-driven local search. In *Proceedings of ICAPS 2012*, 2012.