Universitat Politècnica de Catalunya

Facultat d'Informàtica de Barcelona

Informatics Engineering

Bachelor's Degree Thesis

Understanding Literal Block Distance in SAT solvers

Author:
Adrià Lozano Navarro

Director:
Robert Nieuwenhuis

Mention:
Computer Science

Codirector:
Albert Oliveras Llunell

Thursday 25th June, 2020

I would like to thank Robert Nieuwenhuis and Albert Oliveras, for helping me during the project, sharing his experience and giving me the chance to work on this project.

This thesis is linked to an article published on LPAR-23 [1] and some points of it are used alongside this document.

Abstract

We shed new light on the Literal Block Distance (LBD) and glue-based heuristics used in current SAT solvers. For this, we first introduce the concept of stickiness: given a run of a CDCL SAT solver, for each pair of literals we define, by a real value between 0 and 1, how sticky they are, basically, how frequently they are set at the same decision level.

By means of a careful and detailed experimental setup and analysis, we confirm the following quite surprising fact: given a SAT instance, when running different CDCL SAT solvers on it, no matter their settings or random seeds, the stickiness relation between literals is always very similar, in a precisely defined sense.

We then analyze how quickly stickiness stabilizes in a run, and show that it is stable even under different encodings of cardinality constraints. We then describe how and why these solid new insights lead to heuristics refinements for SAT (and extensions, such as SMT) and improved information sharing in parallel solvers.

Contents

List of Figures

List of Tables

1 Introduction

This is a Bachelor's Degree Thesis of the Bachelor Degree in Informatics Engineering provided by *Universitat Politècnica de Catalunya (UPC)*. The thesis will be developed as a project of the computer science department (CS).

1.1 Context

1.1.1 Boolean satisfiability problem (SAT)

The Boolean satisfiability problem (SAT) is known to be a NP-complete problem [5].

Despite the hardness of the problem, nowadays modern SAT-solvers based on *Conflict-Driven Clause Learning* (CDCL) solve large real world problems in a reasonable time. This is due to the mix of several key components which have a huge impact on the performance of the solvers, some of these are: preprocessing and inprocessing, branching heuristics, restart policies, intelligent conflict analysis and *clause learning* [6].

Clause learning helps the solver search by reducing the search space, even though, since clause learning is made at a really high rate in CDCL solvers, a management of the learned clauses is needed and has become another key component of modern solvers, which reduce the number of clauses learned by keeping only the most relevant ones.

Before the apparition of *Glucose* [7] the relevance of a clause was valued by its activity. *Glucose* proposed a new measure named Literal Block Distance (LBD) which is defined as the different decision levels of the literals in a clause.

1.2 Problem formulation

After the changes that the apparition of LBD measurement brought to modern solvers and in order to understand this measure, we want to take a look inside decision levels of different SAT solvers, which define the LBD measure of the clauses.

The idea then is to compare different SAT solvers runs with different search strategies, on the same problem, in order to check the similarity between them.

So the question that defines our problem is:

Given two runs of CDCL SAT solvers, how similar are their decision levels?

1.3 Stakeholders

In this section we will mention the different stakeholders associated to the project. We define stakeholder as a person or organization in charge of developing the project or involved with it.

1.3.1 Student

In this project the student will be in the role of a experimental researcher, he will be the one in charge of the experiments design and implementation, the execution of these and the extraction of results. He will be also involved on the results analysis and explanation.

1.3.2 Director and codirector

Besides their role as director and codirector of the project, the director and codirector will be in the role of head researchers. They will be in charge of supervising the project, analyzing the results and the decision making, but they may be involved in any of the others parts of the project if needed.

1.3.3 LOGPROG group

The Logic and Programming research group is the group where the research work will be located. This group is part of the Computer Science department from *Universitat Politècnica de Catalunya*.

2 Related work and justification

Since the apparition of *Glucose* LBD has became the standard criteria to evaluate learnt clauses on CDCL SAT solvers, there has been several studies related to it due the high impact it had on the performance of modern solvers.

Some studies try to relate LBD to other components of the solver different to clause learning, is the case of the study made by Md Solimul Chowdhury, Martin Müller, and Jia-Huai You try to relate the Glue Clause idea in order to characterize glue variables[8].

There are another few studies which try to understand a concrete part of the LBD clause learning in order to improve the efficiency of the time used to clause learning, for example Tomohiro Sonobe looks inside the LBD updating process in order to predict whether LBD update can occur for a certain clause [9]

On this context is where we question if literal block distance, related directly to decision levels, is inherent to the problem, the run or the solver. And depending of the result if it has some interesting applications as, for example, the development of portfolio-based SAT solvers.

3 Scope

3.1 Objectives

The requirements needed to achieve this project are the next ones:

- Create a measure to characterize the decision levels on a SAT solver run.
- Create a notion of similarity between runs which adds up with the things that are compared.
- Be capable of analyzing the results in order to determine if our notion of similarity and measure of the runs are pleasant and if that is the case, be capable of analyzing the results

3.2 Requirements

Once the problem and the requirements have been cleared, the next step is to make a breakdown of the objectives and the procedures that have to be made.
So, we can organize the objectives into the next ones:

1. Modify modern competitive SAT solvers and custom and simple SAT solvers to extract the decision levels of different runs.
2. Study and validate the different measures of the runs. This includes developing tools to process the decision levels of a run and turning it to the data needed by the measure.
3. Study and validate the different notions of similarity. This includes developing tools to make the comparisons.
4. Get different and heterogeneous SAT problems, in order to get enough results that validates the study.
5. Create the proper tools to perform large scale experiments to minimize the time needed to get the results.
6. Analyze the results and design and perform any other study according to these.

3.3 Risks

On this section we will explain the different risks that may have an effect on the study, and possible solutions to these

3.3.1 Problems related to the extraction of the decision levels

If the modifications done to the solvers modify the behavior of these or makes the execution slower get the data of the tougher problems could be difficult.

Solution: Change the modifications to made them constant in time.

3.3.2 Problems related to measure of the runs

If the measures proposed are not well suited or are computationally expensive, getting the information needed would be tough or, in the worst case, not profitable.

Solution: The solution here is simple, make sure with a little well known sample that the measure created is good before advancing in to further experiments.

3.3.3 Problems related to the notion of similarity

Similarly to the previous risk, if the notion of similarity proposed is not well suited or is computationally expensive, making the comparisons would be tough or, in the worst case, not profitable.

Solution: Make sure with a little well known sample that the notion created is good before advancing in to further experiments.

3.3.4 Implementation problems

As in any software projects, during the realization of the study we may find errors on the implementation of the methods used.

Solution: We need to be careful in order to avoid the maximum number of errors and with the rest, we need to detect them and fix them the faster we can.

3.3.5 Hardware related problems

For the project we will need one or more computers, where we will do the development of the methods and tools needed and the execution of the experiments, here the risk would be the crash of the computer with the experiments data.

Solution: To minimize the risk of losing data, we will make several backups of the information in different disks.

4 Methodology

4.1 Used methodology

To select the best methodology the components of the development team, the objectives and the requirements of the project need to be considered. In this case we have a development team that consists in a experimental researcher and two head researchers and both the objectives and the requirements are strongly related to the exit of the measures of the runs and the concept of similarity created.

Under this circumstances, the methodology that suits better the project is the prototype based methodology. The idea is to build a prototype that includes a first measure for the SAT solver runs and design a method to compare those measures, then execute some simple experiments on them to see how do they work, after those experiments we will proceed to analyze the results and decides if the measure and the concept of similarity used are satisfactory, if they are not, we will modify and test them, if they are satisfactory we will proceed to a bigger number of larger experiments.

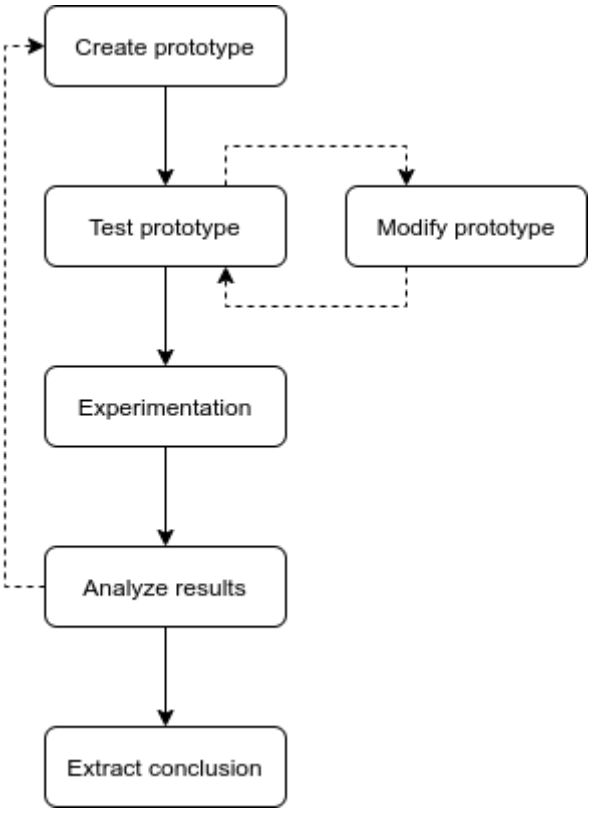

Figure 1: Prototype based methodology scheme. Own compilation.

4.2 Project monitoring

In order to the correct monitoring of the project we will be doing regular meetings, where, apart from discussing the content of the study, the correct flow of the project will be evaluated.

5 Time planning

5.1 Program

The realization of the project will occur between the 10th of February, were the classes start to the 30th of June, where the presentation of this thesis is supposed to be, in any case the dates are approximated due to the ending and the steps to be done in the project are closely related to the result of the experiments.

5.2 Project planning

5.2.1 Previous learning

The first step in all types of research is to acquire a basic knowledge field, which must be enough to do the research, this part is crucial in the developing of the other parts, specially on the analysis part, where that knowledge will have a major impact on the quality of the conclusions extracted of the experimentation results.

5.2.2 Planification

The next step is the planification of the research and the experimentation (the initial objective in the context of this thesis). This part can be divided into this items:

- **Contextualization:** The thesis is introduced both theoretically (definition of the problem to be solved) and practically (what is supposed to be done in order to solve the problem).
- **Related work and justification:** In this part a research of what has been done in the field of study and in relation to the specified problem is done, there must be a justification for the realization of the study, relating this justification to the previous studies done in the field.
- **Scope:** The requirements, objectives, and risks involved of the project are defined, and the expected steps in order to achieve the objectives and avoid the risks.
- **Project planning:** The steps to be done are structured and timed. Alternative plans are also specified.
- **Budget and sustainability:** A study of the costs and the impact in the environment of the research is done.

5.2.3 Design of the study

When the initial study is done it is followed by the design of the research itself, not theoretically but practically.

In this case, what we are going to do is defining the approximation we will be making in each of the research steps, this is rather what measure are we going to use to summarize the run of a SAT-solver or how are we going to store and compare this information.

Once this aspects have been decided we will proceed to find out what the pipeline of the tasks needed is the optimal for our research.

5.2.4 Task description

In the case of this thesis, and due to the type of study that is going to be made, we are in a position where there are several tasks that are defined and other ones that are not. More specifically, the tasks related to the preparation of the experiments and the SAT solvers modifications are fixed while the ones related to the creation of the different measures to develop the study are variable due to the importance of the results of the analysis of each experiment.

Despite that, we can define some generic tasks, which we can arrange depending of the dependencies between them:

1. Previous learning
2. Planification
3. Design of the study
4. Modifications of the SAT-solvers
5. Selection of SAT problems
6. Generation of a measure to study decision levels
7. Generation of a notion of similarity
8. Design of the experimentation
9. Experimentation
10. Analysis and conclusions of the obtained results
11. Documentation of all the study

The steps between 6 and 9 are probably going to be repeated more than one time until we find a measure and a notion strong enough to get conclusions about the experiments. For this reason the estimated time related to this tasks has been estimated upwardly.

5.2.5 Final objective

On this part all the documentation related to the thesis is being checked and finished. This will include all the data and results related to the study, followed by the conclusions and justifications that are needed.

5.3 Estimated times

The time estimation of each task is the following:

Task	**Expected time (hours)**
Previous learning	20
Planification	80
Design of the study	20
Modifications of the SAT-solvers	20
Selection of the SAT problems	10
Generation of a measure to study decision levels	60
Generation of a notion of similarity	60
Design of the experimentation	60
Experimentation	80
Analysis and conclusions of the obtained results	20
Documentation of all the study	40
Total	**470**

Table 1: Table with the expected time in hours of each task. Own compilation.

5.4 Task dependencies

The task dependencies are the following ones:

Task	**Dependencies**
1. Previous learning	-
2. Planification	1
3. Design of the study	2
4. Modifications of the SAT-solvers	2
5. Selection of the SAT problems	1
6. Generation of a measure to study decision levels	4
7. Generation of a notion of similarity	6
8. Design of the experimentation	5, 7
9. Experimentation	8
10. Analysis and conclusions of the obtained results	9
11. Documentation of all the study	10

Table 2: Table with the dependencies between tasks. Own compilation.

5.5 Gantt diagram

This is the Gantt diagram for the study, even though there are some tasks that could be done in parallel (the ones with no dependencies between them), due to the limited resources the major part are done sequentially, the only exception is the first part of the documentation, which is done through all the project.

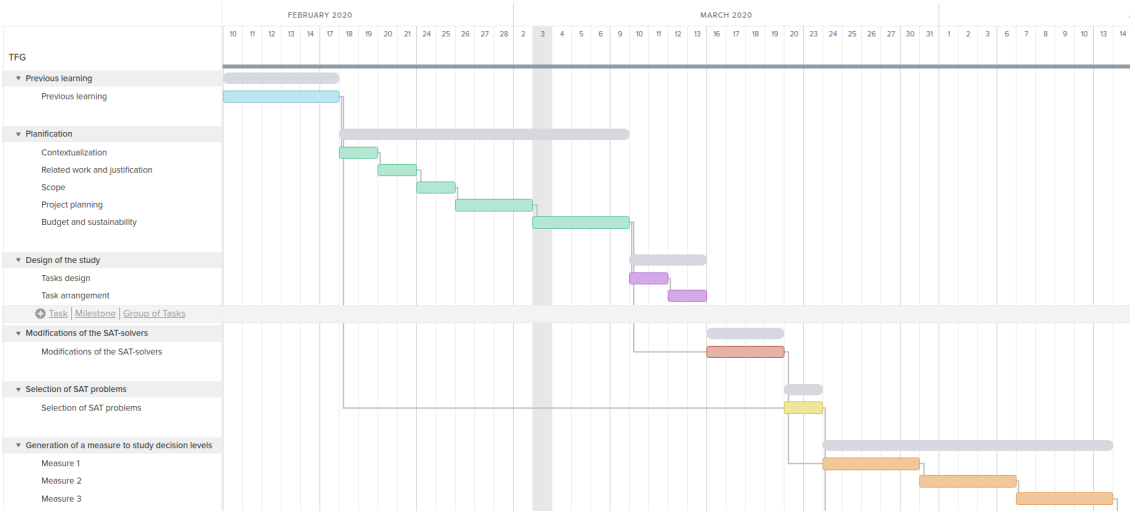

Figure 2: Gantt diagram(part 1) done with TeamGantt [3]. Own compilation

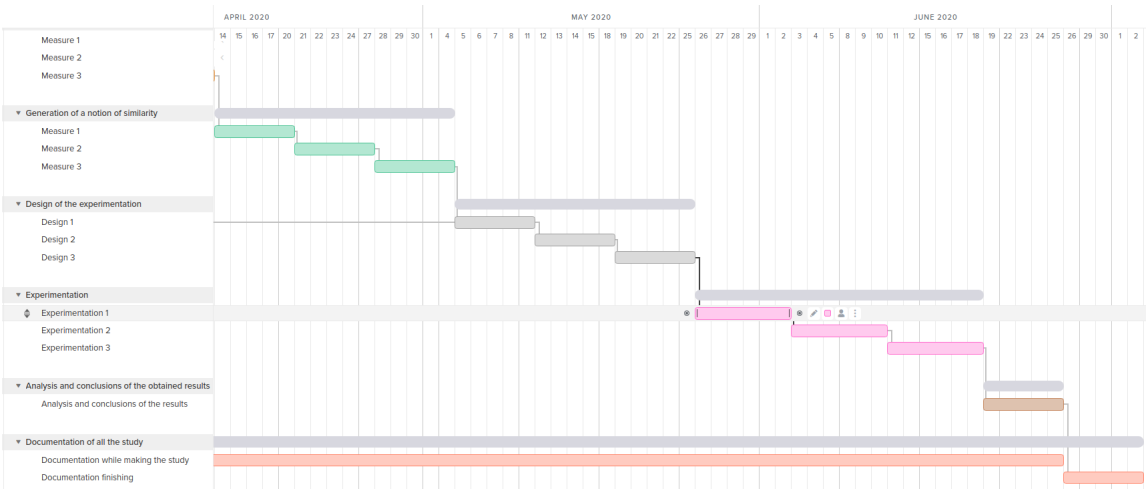

Figure 3: Gantt diagram(part 2) done with TeamGantt [3]. Own compilation

5.6 Required resources

5.6.1 Hardware

For the realization of the study the needed hardware is the next one:

- **Computer:** Where the programming, design and control of experiments and documentation will be done. In this case it will be done in one of the students own.
- **Servers:** Where the experiments will be running. In this case we will be using the ones from the ones from the logical programming research group (LOGPROG) of the CS department.

5.6.2 Software

For the realization of the study the needed software is the next one:

- **Python [10]:** Programming language where the control of the experiments will be done.
- **C++ [11]:** Programming language where the main methods of the experiments and the SAT solvers tracked are made.
- **Sublime Text 3 [12]:** Text editor that will be used for the coding part.
- **Emacs [13]:** Text editor that will be used for the coding part.
- **GitHub [14]:** Cloud storing side where the documentation and the programs done will be held
- **TeamGantt [3]:** Web tool to store and manage the Gantt diagram of the project
- **Racó de la FIB [15]:** Platform to manage the project.
- **Brave [16]:** Web browser used for visualizing pdf files and to search documentation.

5.6.3 Human resources

In terms of human resources the presence of some researchers will be needed (in this case this role will be done by the student, the director and the codirector of the project), there will be also needed the presence of a director and a codirector, the director will be Robert Lukas Mario Nieuwenhuis while the codirector will be Albert Oliveras Llunell.

5.6.4 Others

Other resources needed will be electricity and the space of work of the people involved in the project.

5.7 Risk management: alternative plans and obstacles

Different obstacles may occur during the development of the project, this obstacles will need to be treated as soon as possible to avoid them from interfering with the right fulfilment of the project.

On this section we will treat the risks exposed on 3.3, in order to propose alternative tasks, see the time impact that these may have to the duration of the project and the additional resources that may be needed.

5.7.1 Extraction of the decision levels

The alternative in this part would be the usage of another SAT solver in order to extract the data needed for the experiment, the duration of the project would be increased by 5 to 10 hours for each new SAT solver needed.

5.7.2 Measure of the runs and notion of similarity

Alternatives plans for this parts would be changing the way to study the decision levels extracted, this would have a huge impact on the duration of the project, and this impact will be strongly related to the new method of study, the impact would be at least 100 hours.

5.7.3 Hardware

There are two possible obstacles related to hardware: longer times than expected used in computing and any type of data loss.

If the first of them happen the solution is extending the period of this time, this will extend the duration of the project depending on how many extra time is needed, the affectation of the duration would be between 5 and 40 hours.

If the second obstacle is the one we find we will need to redo some experiments this will have an impact of 80 hours in the worst case scenario.

5.8 Final temporal planification

As it was expected, the time has suffered some changes, being the most relevant one a compression of the dates due to the deadline of LPAR23, the other changes are little variations in the last of the tasks.

The next table shows the final time extension of all tasks:

Task	**Expected time (hours)**
Previous learning	20
Planification	80
Design of the study	30
Modifications of the SAT-solvers	20
Selection of the SAT problems	10
Generation of a measure to study decision levels	50
Generation of a notion of similarity	70
Design of the experimentation	60
Experimentation	100
Analysis and conclusions of the obtained results	20
Documentation of all the study	20
Total	**480**

Table 3: Table with the expected time in hours of each task. Own compilation.

6 Budget

In the previous section we mention the resources used for the development of the study. Now what will be done is break down those resources and look the cost of them.

We will also talk in this section about the possible upsets that may occur and how we are supposed to manage this deviations.

6.1 Human budget

The human resources are three researchers: the director, the codirector and the student itself:

Role	Hours	Prize/Hour	Total
Cathedratic researcher	190	25 €/h	4750 €
Aggregate professor researcher	190	20 €/h	3800 €
Ordinary researcher	470	9 €/h	4230 €

Table 4: Table of the human resources budget

6.2 Hardware budget

The hardware resources costs include the computer used to manage the experimentation and the servers used to run the experiments, the first one budget is the cost of the computer itself and for the budget related to the servers we will calculate the cost using the prize per hour of Google Cloud Computing [17]:

Product	Prize	Hours	Units	Total
Computer	799 €	-	1	100 €
Servers	1.15 €/hour	80	4	368 €

Table 5: Table of the hardware budget

6.3 Software budget

The software resources used are described in the previous section, all software used is Open Source or has a free version available, which is the one we will use, so the cost related to software is 0.

6.4 Other resources budget

Finally here are the cost of the main indirect resources, which are the electricity and the workspace cost, for the electricity, the total of kWh used is only based on the lighting needed to use the computer because the power of the servers and the lighting

of the workspace is included in the cost of these, the workspace prize is based on the prize of MOB - Barcelona [18] coworking:

Resource	Prize	Units	Total
Electricity	0,12€/kWh	300kWh	15 €
Workspace	190 €/month	15 (3 spaces x 5 months)	2850 €

Table 6: Table of the indirect resources budget

6.5 Total budget

Once all the costs have been broke down here are the final predicted costs of the project:

Concept	Cost
Human budget	12.780 €
Hardware budget	468 €
Software budget	0 €
Others budget	2865 €
Total	16.113 €

Table 7: Table of the total budget

6.6 Budget deviation management

The main problem that may occur during the study is a possible extension of the study due to the risks commented before, the longer is the time needed to obtain results and make the experiments, the more will be the budget needed.

In order to minimize this deviations what will be done is reorganize the tasks established in the Gantt diagram, the impact of any deviation will be calculated using the total time of the project, which will be extracted from the Gantt diagram after the reorganization of the tasks.

7 Sustainability report

7.1 Environmental dimension

Have you estimated the environmental impact of undertaking the project?

Although a concrete estimation of the environmental impact has not been made, we need to consider that the resources used in the realization of it are slight, and all of them, are also used for others purposes.

Have you considered how to minimize the impact, for example by reusing resources?

As it has been said in the previous question the resources used for the project that could be reused for other tasks are reused. These are: the computer, the servers, and the workspaces.

In what ways will your solution environmentally improve existing solutions?

Being this a research project all the possible improves related to the project are difficult to estimate, but if the research results are used to improve modern SAT solvers, which are used to many types of scheduling problems, such as reorganizing transport routes in order to make these more efficient the impact could be a big one.

7.2 Economic dimension

In what ways will your solution economically improve existing solutions?

As it is said in the previous section the economical improves related to the project are difficult to calculate, but as said previously, if the research results are used to improve modern SAT solvers to be used for larger scheduling problems the economical impact could be big.

7.3 Social dimension

What do you think undertaking the project has contributed to you personally?

On a personal level the project will contribute knowledge about logical programming, SAT solvers running and SAT solvers performance. The project will also improve my general programming skills and my ability to extract conclusions from results.

In what ways will your solution socially improve (quality of life) existing? Is there a real need for the project?

As said before section the social improves related to the project are also difficult to

calculate, but as said previously, if the research results are used to improve modern SAT solvers, allowing them to be used for larger scheduling problems could improve the quality of life of several people in different ways.

8 Important concepts

8.1 Boolean satisfiability problem (SAT)

In propositional logic, a formula is and expression containing the operators AND and OR, literals and parentheses, where a literal is either a propositional variable or its negation, then we can define satisfiability as:

Definition 1 *A formula ϕ is satisfiable if there exists an assignment of values to its variables that makes ϕ true.*

The Boolean satisfiability problem (SAT) is then defined as:

Definition 2 *Given a formula ϕ determine if its satisfiable*

8.1.1 Conjunctive Normal Form (CNF)

Nowadays SAT solvers determine the satisfiability of formulas in conjunctive normal form (CNF), this means that the formula is a conjunction of clauses, where a clause is a disjunction of literals:

$$(A \vee \neg B \vee \neg C) \wedge (\neg D \vee E \vee F) \wedge (\neg A \vee E \vee B) \wedge ... \wedge (\neg C \vee D \vee \neg F)$$

Every propositional formula can be converted into an equivalent formula that is in CNF. This transformation is based on rules about logical equivalences: double negation elimination, De Morgan's laws, and the distributive law.

In some cases this conversion to CNF can lead to an exponential explosion of the formula, to avoid this there are several types of transformation that preserve satisfiability rather than equivalence in the formula.

8.2 Davis–Putnam–Logemann–Loveland (DPLL) [2]

In logic and computer science, the Davis–Putnam–Logemann–Loveland (DPLL) algorithm is a complete, backtracking-based search algorithm for solving the CNF-SAT problem.

The basic backtracking algorithm runs by choosing a literal, assigning a truth value to it, simplifying the formula and then recursively checking if the simplified formula is satisfiable; if this is the case, the original formula is satisfiable; otherwise, the same recursive check is done assuming the opposite truth value.

The DPLL algorithm enhances over the backtracking algorithm by the eager use of the following rules at each step:

Unit propagation

If a clause is a unit clause, i.e. it contains only a single unassigned literal, this clause

can only be satisfied by assigning the necessary value to make this literal true. Thus, no choice is necessary. In practice, this often leads to deterministic cascades of units, thus avoiding a large part of the naive search space.

Pure literal elimination

If a propositional variable occurs with only one polarity in the formula, it is called pure. Pure literals can always be assigned in a way that makes all clauses containing them true. Thus, these clauses do not constrain the search anymore and can be deleted.

Then, unsatisfiability of a given partial assignment is detected if one clause becomes empty, that occurs if all its variables have been assigned in a way that makes the corresponding literals false, while satisfiability of the formula is detected either when all variables are assigned without generating the empty clause or all clauses are satisfied.

The DPLL algorithm can be summarized in the following pseudo code, where ϕ is the CNF formula:

Algorithm 1 DPLL(ϕ)

if ϕ is a consistent set of literals **then**
  return true
end if
if ϕ contains an empty clause **then**
  return false
end if
for every unit clause l in ϕ **do**
  $\phi \leftarrow$ unit-propagate(l, ϕ)
end for
for every literal l that occurs pure in ϕ **do**
  $\phi \leftarrow$ pure-literal-elimination(l, ϕ)
end for
l$\leftarrow$ choose-literal(ϕ)
return DPLL($\phi \wedge$ l) or DPLL($\phi \wedge$ not(l))

Figure 4: DPLL pseudo code. [2]

Decision level

In association to the unit propagation and literal decision processes in the DPLL algorithm the concept of decision level arises.

When using unit propagation, variables choosen that lead through the actual

branching process are called decision variables and those assigned values as a result of unit propagation are called implied variables. Then, a decision level is formed by all assignations, of decision and implied variables, derived from deciding a variable and do the unit propagation process.

8.3 Conflict-Driven Clause-Learning algorithm

Conflict-Driven Clause-Learning (CDCL) is an algorithm for solving the Boolean satisfiability problem (SAT), it is also an evolution of the Davis-Putnam-Logemann-Loveland (DPLL) algorithm, being the main difference between them the analysis after each conflict that concludes in back jumping instead of back tracking and clause learning.

Back jumping

In DPLL when a conflict arises the procedure is to do backtracking to the previous decision level and then do a change in the assignation of the decision variable, here is an example of how backtracking is used in the DPLL algorithm:

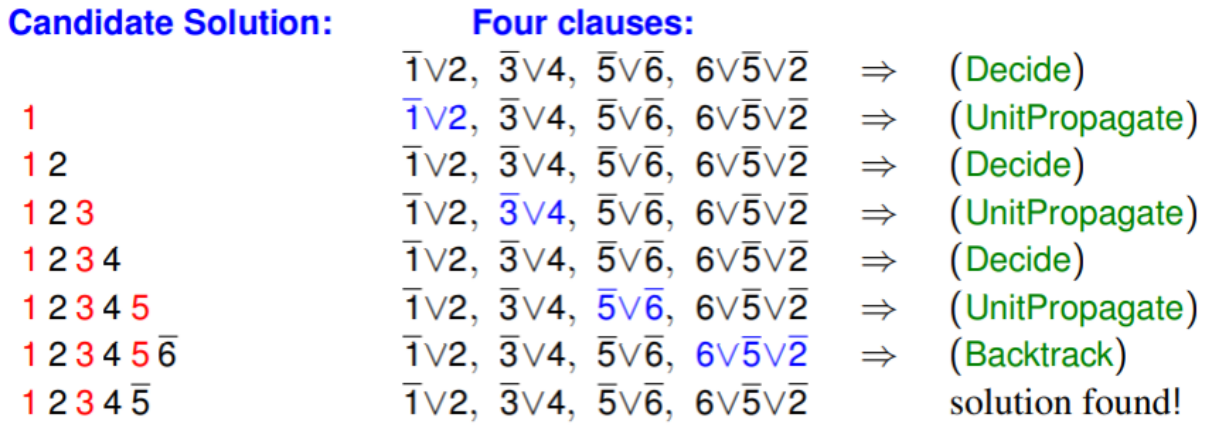

Figure 5: Backtracking example. [4]

As we can see, decision level 3, 4 is irrelevant for the conflict with $6 \wedge \neg 5 \wedge \neg 2$, this means that we can propagate $\neg 5$ earlier, jumping decision level 3, 4:

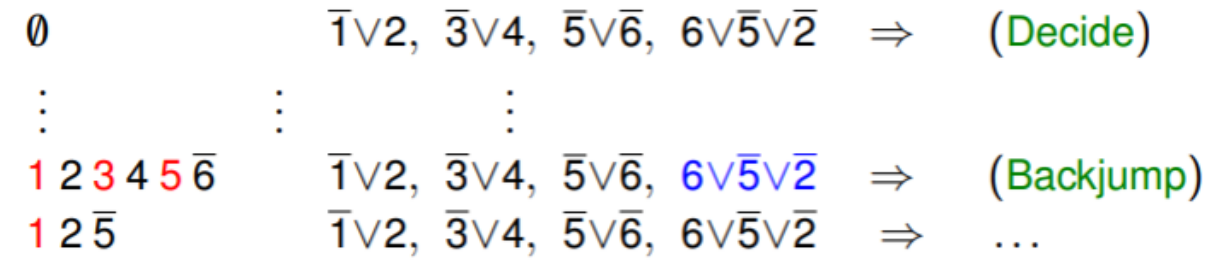

Figure 6: Back jumping example. [4]

This procedure of further backtrack to the decision level that is involved in the conflict instead of the previous one is called back jumping.

Clause learning

In the conflict analysis process, each time a back jump is done a back jump clause related to it is created (in the previous example the clause would be $\neg 2 \wedge \neg$), this clause is the logical consequence of the clause set, and reveals the previously mentioned propagation at an earlier level. Then, the back jump clause (also called lemma) is added to the set of clauses in the formula to do this unit propagation in the future and avoid the same conflict from happening again.

Then the entire process of conflict analysis is [4]:

1. Find a back jump clause $C \wedge I$
    - That is a logical consequence of the clause set
    - That reveals a unit propagation of I at an earlier decision level d (where C is false)
2. Return to decision level d and do the propagation

In parallel to the clause learning process the need of a management of these new lemmas is created. If all new lemmas are kept, there will be not only a problem with disk space but also the unit propagation will became too slow because of the excess of clauses. We need to delete lemmas during the execution, and to do that a system to rate lemmas is a key point of modern CDCL based SAT solvers.

8.4 Literals Block Distance

The latest improvement in modern SAT solvers has been the introduction on 2009 of the literals block distance (LBD) concept brought by Gilles Audemard and Laurent Simon [7]:

Definition 3 *Literals Blocks Distance (LBD) Given a clause C, and a partition of its literals into n subsets according to the current assignment, s.t. literals are partitioned with respect to their decision level. The LBD of C is exactly n.*

LBD has become the best way to rate lemmas towards selecting the ones that have to be kept in substitution of the previous activity-based measurement, in order to determine the value of a learned clause, the lower LBD the better the lemma.

9 Study of the stability of decision levels [1]

In order to have a better understanding of LBD we want to know if the importance of this measure is attached to a run or to the problem itself. Due to the correlation between the literals block distance of a clause and decision levels, it is easy to see that, if decision levels between runs are similar, the lemmas that may appear and also its LBD will be similar or at least have a relation, then, what we want to see is the similarity between decision levels of different runs.

To do the analysis we first need a method to extract decision levels of SAT solver runs and then do comparisons between them, this means we are in need of a way to characterize those decision levels and then a comparison method between those characterizations.

9.1 Characterization of the decision levels on a SAT solver run

9.1.1 What we want to characterize

Once it has been decided that our analysis will be made based on the image of all decision levels we need to find a way to process those to summarize this information into something accountable and space affordable (the file with all the decision levels of a run may be to large, specially in hard problems), for the purpose of that we have decided that our measure must relate pairs of variables, and summarize if those variables have been decided together more or less frequently.

This frequency is clearly affected by the lemmas added in the conflict analysis and deleted in the cleanups, so we can conclude that if the frequencies shown in different runs is similar, it would mean that the decision levels between them are similar.

This idea of characterization is also interesting because it can be seen as an undirected weighted graph of the run, where the nodes are the variables and the edges value is the notion of frequency that needs to be created, allowing the comparisons between this graphs to be done with previously created graph comparison methods.

9.1.2 First idea of Stickiness

What is left then is to concrete how this frequency is going to be determined, we have to take into account what information can be extracted from the decision levels file, this information ultimately is resumed to how many times a variable has been assigned and how many times a pair of variables have been assigned together, with this information the next step is to create a notion of Stickiness that must tell whether two variables use to be assigned together or not.

Taking that to account this is the first definition of Stickiness:

Definition 4 *Within a given run R of a CDCL SAT solver, for each variable x, we*

define $n_R(x)$ *as the total number of times* x *gets assigned in* R*, and for a pair of variables* x *and* y*, we define* $n_R(x, y)$ *to be the total number of times both variables get assigned* at the same decision level*, then the stickiness of* x *and* y *in* R *is:*

$$stick_R(x, y) = \frac{2\ n_R(x, y)}{n_R(x) + n_R(y)}$$

This definition is convenient because of the output of the formula is always a real number between 0 and 1, returning 0 if two variables are never assigned together and 1 if they are always assigned together.

9.1.3 Stickiness

Despite the well performance of the previous definition of stickiness at a experimental level, we wanted to change it a bit, in order to had a more statistical-based formula, so the previous definition was changed to:

Definition 5 *Within a given run* R *of a CDCL SAT solver, for each variable* x*, we define* $n_R(x)$ *as the total number of times* x *gets assigned in* R*, and for a pair of variables* x *and* y*, we define* $n_R(x, y)$ *to be the total number of times both variables get assigned* at the same decision level*, then the stickiness of* x *and* y *in* R *is the (conditional) probability, at a given moment during the run, that* x *and* y *are assigned together at the same decision level, provided that both are assigned:*

$$stick_R(x, y) = \frac{n_R(x, y)/T}{n_R(x)/T + n_R(y)/T - n_R(x, y)/T} = \frac{n_R(x, y)}{n_R(x) + n_R(y) - n_R(x, y)}$$

(where T is the total number of decision levels, or decisions, in R).

This new definition maintains all the previously said pros, the output being a real number between 0 and 1 and also the good results in a small experimental level while also providing a better suited definition.

9.2 Comparisons within the graphs

Then, given a run R on a given CNF-SAT problem, over variables $\mathcal{X}$, its *stickiness function* $stick_R \colon \mathcal{X} \times \mathcal{X} \to [0 \dots 1]$ maps pairs of variables (or literals) to their stickiness: $(x, y) \mapsto stick_R(x, y)$. Now assume we have two different runs R and R' (of two possibly completely different CDCL solvers). The question arises: how can we quantify, again by a number between 0 and 1, the *similarity* $Sim(R, R')$ between the stickiness functions $stick_R$ and $stick_{R'}$?

As it was said earlier, the function $stick_R$ can be seen as a weighted graph with vertices $\mathcal{X}$ and where $weight(x, y) = stick_R(x, y)$. Several notions for weighted graph similarity exist that do not make much sense for stickiness due to most pairs of $stick_R(x, y)$ are zero or close to zero.

Then what we need is a new notion, this notion should give more importance to the comparisons where one or both of the edges is close to 1. Then, the first notion of similarity became this one:

$$Sim_1(R, R') = \frac{\sum_{\{x,y\}\subseteq\mathcal{X}} 1 - |stick_R(x,y) - stick_{R'}(x,y)|}{\sum_{\{x,y\}\subseteq\mathcal{X}} 1}$$

This notion seems good at a first glance, but after trying it in small experiments we find out that due to the majority of $stick_R$ pairs being close to 0, the absolute difference between them, is going to be something small compared to 1, so in fact with this comparison method, each time that the bigger edge in the comparison is small, it does not matter if the other pair of the comparison does not even exist.

In fact, this similarity concept give us similarities close to 0.8 when R and R' are runs of the same SAT solver, on two CNFs that are identical except for a random permutation of variable names, when in fact this comparison should give results close to 0.

To overcome this, we then defined:

$$Sim(R, R') = \frac{\sum_{\{x,y\}\subseteq\mathcal{X}} min(\ stick_R(x,y),\ stick_{R'}(x,y)\ )}{\sum_{\{x,y\}\subseteq\mathcal{X}} max(\ stick_R(x,y),\ stick_{R'}(x,y)\ )}$$

which still gives results in $[0 \ldots 1]$ and closer to 1 if many pairs (x, y) are similarly sticky in R and in R', and will gave us similarities close to 0 in the previous case where R and R' are runs on the same SAT solver, with the same problem except for the permutation.

It is noticeable that $Sim(R, R')$ is quite demanding, for example, since most literal pairs have low stickiness, if a large majority have stickiness, say, 0.3 in one run and 0.1 in the other run, they decisively contribute pushing $Sim(R, R')$ to 0.33. This remarks the value of high similarities results, something persuaded when creating the notion too.

10 Experimentation

10.1 Solvers, tools and problems

10.1.1 Solvers

In an early stage a solver made by our own was used to run little experiments, with simple problems, in order to check the correct work of the stickiness and similarity notions, this solver was provided by the LOGPROG group, and it was modified by adding LBD to the lemma evaluation process, and also to extract all decision levels within a run to a external file.

After the first results in those small experiments the need of using several solvers and versions of them to validate the results and expand the experimentation appeared, the solvers used for this were Cadical [19] and Glucose [20], that were selected from the 2019 SAT Race.

This two new solvers were also modified to extract decision levels from a their run, this was harder than with the first solver because of different types of preprocessing or inprocessing, such as variable elimination or variable renaming, that are used in this solvers.

Also, to complement the experimentation, we will be using two more versions of the same Cadical in addition to the default one, one tuned to solve faster the satisfiable problems and another one tuned to solve faster unsatisfiable problems. This will be done to see the similarity within the same solver with different internal parameters and configurations.

10.1.2 Tools

Along with the previously mentioned solvers several tools were required to run the experiments, being the more relevant of all the tools used the following ones:

Counter

A tool to process the decision levels file and extract $n_R(x)$ for all literals and $n_R(x, y)$ for all pairs of literals assigned in the same decision level was obviously needed. Concretely, the program used to do this process creates a new counter file (which can be seen as the graph of the run) every 100.000 conflicts.

It must be said that processing decision levels files efficiently is non-trivial: counting $n_R(x)$ is fast, but $n_R(x, y)$ requires quadratic work in the size of each decision level with lots of random accesses to this quadratic number of counters (in fact $4n^2$ for literals if there are n variables; that is why, as we will see later, problems with a very large number of variables were avoided).

Rather than adding more work to the previous program counting $n_R(x)$ and

$n_R(x, y)$ for literals and variables, a new program made to extract the graph of variables from the one of literals was also made, being the original counter program in charge of only counting literals.

Comparator

Another obviously needed tool is one capable of running the comparisons between counter files/graphs, the program designed for this uses the fact that the counter files are ordered to do the comparison in $O(n)$.

Experiments executor

The previous mentioned programs, including the solvers, work sequentially, in order to speed up the total time execution of all experiments a program capable of executing several experiments in parallel in every machine used was needed.

The program created to fulfill that purpose is a simple program with it most important feature being the management of disk memory due to the problem that is created by the size of decision levels files, being each of this files capable of reaching 70 GB in an hour.

10.1.3 Problems

The problems selected for the experimentation are separated in two groups, the first group is formed by little demo leagues confection problems provided by Barcelogic and the second one is formed by 2019 SAT race problems.

The first group was used to check the correct work of the notions and tools created and the second group was used to run the formal experiments to validate the hypothesis extracted from the first ones.

To select the problems of the second group randomly we downloaded all available problems from the SAT race 2019 page, and then selected 10 out of the 170 old problems by picking the multiples of 17 from the ordered list, from this ones three were discarded because of having too many variables ($> 100,000$) being selected the next one in order.

Then, the final list of problems from SAT race 2019 selected was:

- 9-50-sc2017
- ctl-4291-567-2-unsat-sc2013
- frb59-26-1.used-as.sat04-891-sc2011
- le450-15b.col.15-sc2018
- par32-1-c.shuffled-as.sat03-1531-sc2002

- rpoc-xits-12-UNKNOWN-sc2009
- tseitingrid7x160-shuffled-sc2016
- 001-80-12-sc2014
- smulo032-sc2012
- AProVE07-26-sc2007

10.2 Experiments

For all experiments we used 1-hour solver timeouts. Since Cadical does heavy inprocessing, eliminating variables, when comparing stickiness of two runs we only consider their intersection of variables (and of course renaming them back to their original numbers).

10.2.1 First insight in stickiness

To get a first insight of stickiness we wanted to see how it is distributed (on average on the four solvers) for each problem, here we can see the distribution of all stickiness pairs:

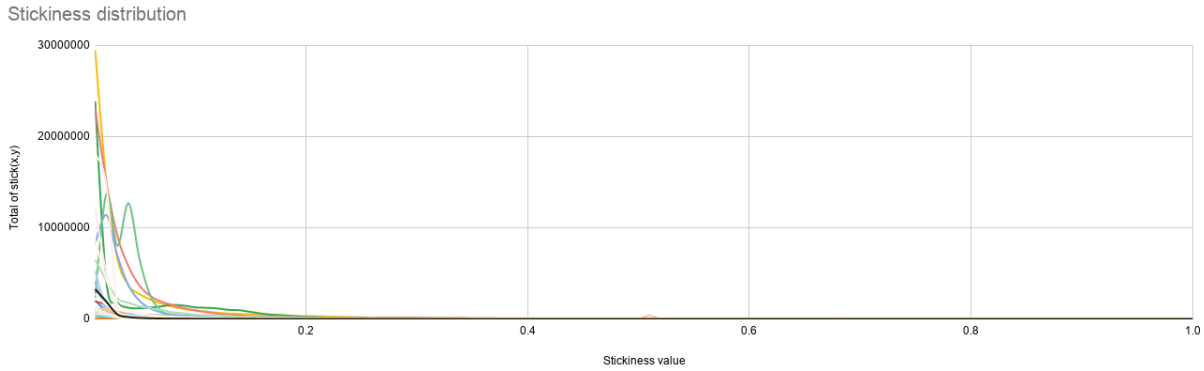

Figure 7: Distribution of stickiness values. [1]

As we can see, most pairs have very low stickiness, close to 0 in most cases.

10.2.2 Unrelated runs similarity

The first experiment is one that has been mentioned earlier, to each problem of the list a parallel problem was created, this new problem was exactly equal to the first one except for a permutation in the name of the variables. Then for each solver we executed both problems in the same random seed and then do a comparison between them. It must be noticed that executing in the same random seed means that the decision levels files and the counter files are also identical except for the permutation of variable names.

The objective of this experiment is to check that our similarity notion is consequent and does not give high similarities on different problems no matter how similar they are.

Here we can see the results for the 40 pairs of runs:

	cad1	cad2	cad3	glu
9-50-sc2017	0.04	0.09	0.02	0.01
ctl	0.03	0.07	0.10	0.04
frb59	0.01	0.07	0.07	0.08
le450-15b	0.05	0.02	0.05	0.07
par32-1-c	0.09	0.01	0.07	0.04
rpoc-xits	0.04	0.08	0.03	0.10
tseiting	0.07	0.07	0.08	0.09
001-80	0.03	0.04	0.06	0.10
smulo	0.06	0.03	0.09	0.01
AProVE	0.05	0.03	0.10	0.01

Table 8: Comparison of unrelated runs. [1]

The results show $sticky_R$ and $sticky_{R'}$ have, as expected, very low similarity, never higher than 0.1, this confirms that our similarity notion is consequent because unrelated runs indeed have low similarity.

10.2.3 Similarity within a certain problem with the same solver

The second experiment will consist in the comparison of the same problem in the same solver with two different random seeds.

To show if decision levels are stable on the same problem, the first approach is to compare two runs only differentiated by its random seed, this different random seed in the solvers used imply small changes in the run, such as certain random decisions in certain moments.

Here are the similarities for the new 40 pairs of runs:

	cad1	cad2	cad3	glu
9-50-sc2017	0.90	0.93	0.93	0.90
ctl	0.78	0.71	0.70	0.70
frb59	0.87	0.82	0.80	0.87
le450-15b	0.87	0.88	0.94	0.89
par32-1-c	0.92	0.86	0.85	0.90
rpoc-xits	0.77	0.75	0.76	0.79
tseiting	0.60	0.61	0.64	0.69
001-80	0.62	0.59	0.59	0.68
smulo	0.60	0.60	0.68	0.68
AProVE	0.68	0.66	0.73	0.69

Table 9: Comparison of runs of the same solver with same problem. [1]

We can see considerably big similarities, then, we can conclude that decision levels involving runs on the same SAT solver and problem are certainly related. This result is quite surprising taking to account that the behaviour of SAT solvers is indeed a chaotic one.

10.2.4 Similarity within a certain problem between different solvers and configurations

After we have seen that stickiness in a certain solver and problem is related between runs, the next step is to check if this only happens between runs of the same solver and configuration or is something associated to the problem.

To check that the next experiment will consist on the comparison of runs of the same problem in different solvers, this makes $\binom{4}{2} = 6$ comparisons for each problem (each run has similarity 1 with itself) for a total of 60 comparisons:

	cad1-cad2	cad1-cad3	cad1-glu	cad2-cad3	cad2-glu	cad3-glu
9-50-sc2017	0.76	0.79	0.65	0.68	0.67	0.58
ctl	0.75	0.71	0.55	0.68	0.54	0.56
frb59	0.82	0.86	0.50	0.79	0.51	0.50
le450-15b	0.75	0.76	0.53	0.64	0.52	0.55
par32-1-c	0.77	0.60	0.55	0.64	0.66	0.53
rpoc-xits	0.68	0.70	0.53	0.63	0.54	0.53
tseiting	0.63	0.50	0.48	0.52	0.51	0.56
001-80	0.63	0.50	0.51	0.57	0.55	0.53
smulo	0.65	0.50	0.48	0.53	0.51	0.56
AProVE	0.68	0.5	0.52	0.54	0.51	0.61

Table 10: Comparison of different solvers on the same problem. [1]

As we can see the similarities are still high despite the big differences between the solvers and the configurations, this result is even more surprising than before and clearly indicates that decision levels are somewhat stable in a problem.

10.2.5 Evolution of stickiness in a certain run

In this experiment for each of the 40 runs (10 instances, 4 solvers), we list the similarities between final stickiness of the run and stickiness after different numbers of conflicts.

The next table shows how stickiness evolves along a run of a solver:

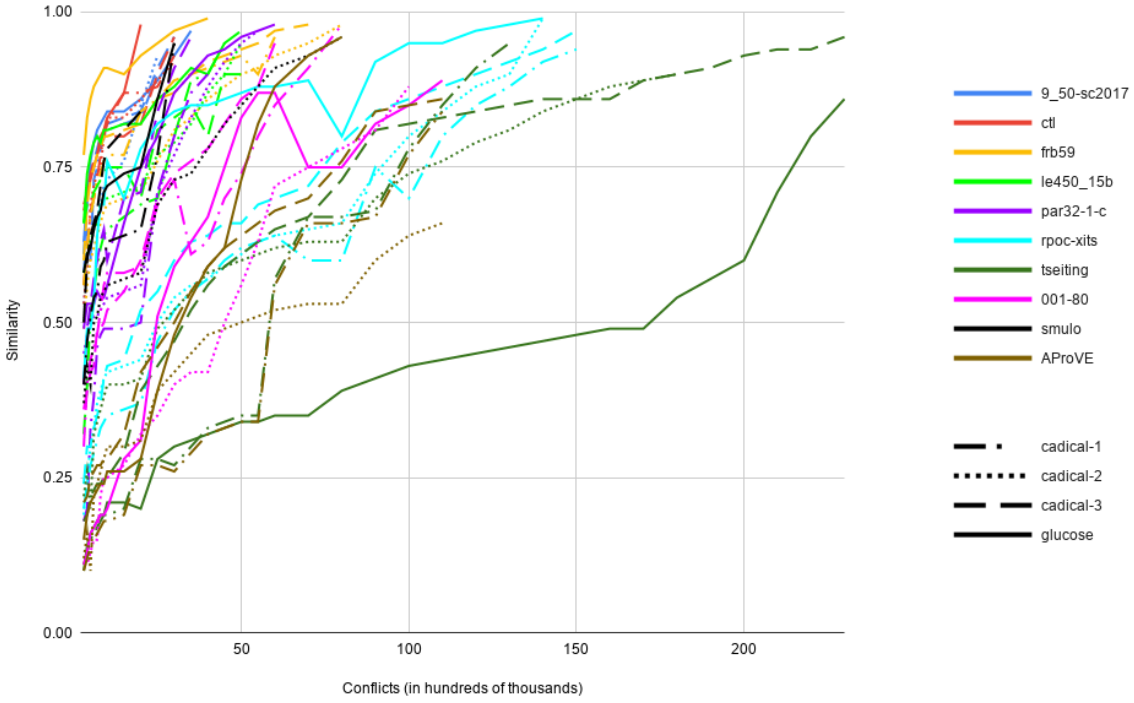

Figure 8: Stickiness evolution along a SAT solver run. [1]

As we can see, most runs quickly achieve similarities above 0.5, but not all. This gives some insight into the question of how stable the "traditional" LBD values of clauses are in a given run.

10.2.6 Similarity within different encodings of the same problem

The fifth experiment objective was to check if stickiness remains stable under different cardinality constraints encodings. Practical problems frequently involve arithmetical constraints, in particular, cardinality constraints, of the form $l_1 + \ldots + l_n \geq k$ (or $\leq$ or $=$). Solver performance depends significantly on how these constraints are encoded into CNF form.

To do the experiment, a simplified version of a real-world problem of one of Barcelogic's professional sport scheduling customers, a major European football league was used. The problem involves scheduling a double round-robin league, a combinatorial object with many exactly-one-constraints (each team one match per round, each match on exactly one round, etc.), as well as many other cardinality constraints (on this round, at most four Sunday matches, at least 5 Saturday matches, at most 4 top-50 matches per round, etc.).

We ran the four solvers on the problem encoded: A) using direct encodings without auxiliary variables (30,685 variables and more than 4 million clauses), B) using standard sophisticated cardinality encodings ([21])(47,681 variables and some 600 thousand clauses). In version B), the first 30,365 variables are the ones of version

A), the other ones are auxiliar variables created by the encodings.

For each solver, the similarity on these common literals of the stickiness relations of version A) and version B) is:

cad1	cad2	cad3	glu
0.65	0.74	0.68	0.73

Table 11: Comparison of the same solver on the same problem with two different encodings. [1]

As we can see, stickiness remains highly stable under different encodings.

11 How can this new insights help improving solvers? [1]

LBD-based cleanups have become standard in state-of-the-art SAT solving. It seems that our notion of stickiness, along with our analysis of its stability along runs, and its similarity between different runs with different solvers, provides interesting new insights in this context:

Dynamic LBD:

Some state-of-the-art solvers forever keep the LBD score of a clause c according to the stack when c was generated, a snapshot of that moment; others update it from time to time according to a later stack, which again may be an imprecise snapshot. Our results seem to indicate that it could make sense to collect stickiness information during the run and consider clause deletion criteria based on a *dynamic LBD* notion: the LBD of c would be the smallest k such that the literals of c can be split into k disjoint subsets, where pairs of literals inside each subset are highly sticky, say, ≥ 0.95.

Stickiness in portfolio solvers:

Most current parallel SAT solvers are based on the portfolio approach: running several different CDCL solvers in parallel, with different randoms seeds and/or settings, in the hope that one of them gets lucky. One relatively open question is: which information to share between nodes? Experts seem to agree on sharing units and/or binary clauses, but not on sharing glue or other low-LBD clauses. Our results seem to explain why the latter might be useful, especially when a notion such as the mentioned dynamic LBD is used.

12 Relationship with syntactic properties of CNFs [1]

In contrast with our semantic and dynamic analysis of the relationship between variables, several studies based on syntactic and static analysis of CNFs exist. They are mostly based on the *Variable Incidence Graph (VIG)* of a CNF F. In this weighted graph, every variable in F corresponds to a node and, for every clause with k variables, we create $\binom{k}{2}$ edges connecting all pairs of variables in the clause. The weight of an edge takes into account how many clauses contain both variables and which is their length. More formally:

$$w(x,y) = \sum_{\substack{C \in F \\ x,y \in C}} \frac{1}{\binom{|C|}{2}}$$

It has been shown [22] that VIGs constructed from industrial SAT instances are highly *modular* in the following sense: one can partition the vertices into *communities* in such a way that there is a very big amount of edges connecting vertices within the same community, but a very low number of edges between vertices of different communities.

These results are purely syntactic, i.e., they do not take into account the behavior of SAT solvers when solving the formula. Indeed, using our "picky" notion of similarity, we found zero similarity between the VIGs and the graphs of our stickiness relation, that express semantic information of a concrete run. Even by using an oracle that adjusts all non-zero weights of the VIG to the weight in our graphs, this lack of similarity could not be fixed. However, one could still argue, though, that very different graphs might still have highly similar community structures.

However, despite the purely syntactic nature of VIGs, researchers have tried to prove that modularity is indeed related to the runtime behavior of SAT solver, via the notion of LBD. One example of this line of research is the paper [23]. It uses a simplified version of the aforementioned VIG, where now weights are simply 0 or 1 depending on whether there is a clause containing both variables. This seems to ignore important information and can lead to unexpected situations. For example, adding to the CNF a single clause which is the disjunction of all variables would render this variant of VIG a complete (and also completely meaningless) graph. More realistically, a different encoding of, say, the (omnipresent) at-most-one constraint $l_1 + \ldots + l_n \leq 1$ also leads to a completely different VIG: under the quadratic encoding the literals $l_1 \ldots l_n$ form a clique in the VIG, whereas for the ladder encoding *none* of them is connected and, in fact, l_1 and l_n are at distance n in the VIG.

A conclusion of [23], states that in conflict clauses c there is a high similarity between a) the number of VIG communities they involve $\#com(c)$ and b) their LBD, $LBD(c)$. A detailed analysis of [23] reveals, however, that what is shown is something else. Namely, that for many SAT instances the set S of data points $|LBD(c) - \#com(c)|$ from the first 100,000 lemmas c generated in a given run has a low standard deviation: usually less than 0.1, implying an very large majority of

the data points are at distance less than 0.1 from the average (this seems even more surprising since the data points are integer and the average needs not be integer). Moreover this seems beside our point: if all data points were exactly, say, 1000, this standard deviation would even be 0, even though the $LBD(c)$ and $\#com(c)$ would be very different for each lemma c. To know more about how similar $LBD(c)$ and $\#com(c)$ are, we would be more interested in the average, median or the quartiles of S.

Finally, the argued relationship between LBDs and communities, and the belief that lemmas with low LBDs are key in the performance of SAT solvers, have pushed researchers to look for preprocessing methods that infer clauses that involve few communities [24]. The hope is that these clauses would correspond to lemmas with low LBD learned during a run of a SAT solver and hence will speed up SAT solvers. In [24], modest gains are obtained for satisfiable problems.

All in all, since the stickiness relation (and, possibly, graphs derived from it) seems to capture interesting properties about solving SAT instances (more than purely syntactic graphs), we believe that these new concepts can lead to further insights and tools to be exploited in the future development of SAT solving technology, and carry over to extensions such as SMT [25, 26], Integer Linear Programming [27], Lazy Clause Generation [28], or Pseudo-Boolean solving [29], among others.

13 Conclusions

In this project we have shed new light on the Literal Block Distance heuristics, currently used in top modern SAT solvers. To do this we have created a new concept of stickiness to summarize all decision levels within a SAT solver run, and also a new notion of similarity to do comparisons between those run resumes.

In order to test and validate these new notions several experimental setup have also been designed, concretely a smaller setup with small problems to test these new notions and a larger setup based on SAT race 2019 problems to validate them. To execute this previous mentioned setups a bunch of tools have been created, this tools have not only been designed to execute the different experiments but also to be time and space affordable.

Also, in order to correct and extract conclusions from this experimental setups various careful analysis have been made during all the project, all in all we have concluded a quite surprising fact: given a SAT instance, when running different CDCL SAT solvers on it, no matter their settings or random seeds, the stickiness relation between literals is always very similar.

Finally, since stickiness relation seems to capture interesting properties about solving SAT instances, this project has also included and analysis of how our new notions are related to other previously defined properties of CNFs and how can this new insights help improving SAT solvers.

On a personal level, this project has helped me to complement and hugely increase the knowledge acquired in the degree and, specially, in the computer science mention. In particular the most deepen knowledge has obviously been in the logical programming field. This project has also served to introduce myself to the academical field, something that will for sure help me decide what I want to do in the future.

References

[1] R. Nieuwenhuis, A. Lozano, A. Oliveras, and E. Rodríguez-Carbonell, "Decision levels are stable: towards better sat heuristics," in *LPAR23. LPAR-23: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, ser. EPiC Series in Computing, E. Albert and L. Kovacs, Eds., vol. 73. EasyChair, 2020, pp. 1–11. [Online]. Available: https://easychair.org/publications/paper/Bz5Z

[2] Wikipedia, "Dpll algorithm," https://en.wikipedia.org/wiki/DPLL_algorithm, (Accessed on 06/22/2020).

[3] T. Gantt, "Teamgantt," https://www.teamgantt.com/, (Accessed on 03/2/2020).

[4] R. Nieuwenhuis, "Sat-based techniques for integer linear constraints." in *GCAI*, 2015, pp. 1–13.

[5] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, M. A. Harrison, R. B. Banerji, and J. D. Ullman, Eds. ACM, 1971, pp. 151–158. [Online]. Available: https://doi.org/10.1145/800157.805047

[6] A. Biere, M. Heule, and H. van Maaren, *Handbook of satisfiability*. IOS press, 2009, vol. 185.

[7] G. Audemard and L. Simon, "Predicting learnt clauses quality in modern SAT solvers," in *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, C. Boutilier, Ed., 2009, pp. 399–404. [Online]. Available: http://ijcai.org/Proceedings/09/Papers/074.pdf

[8] M. S. Chowdhury, M. Müller, and J.-H. You, "Characterization of glue variables in cdcl sat solving," *arXiv preprint arXiv:1904.11106*, 2019.

[9] T. Sonobe, "Looking inside literal blocks: Towards mining more promising learnt clauses in sat solving," in *2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 2016, pp. 972–979.

[10] P. S. Foundation, "Python official webpage," https://www.python.org/, (Accessed on 03/1/2020).

[11] I. JTC1, "C++ iso official website," https://isocpp.org/, (Accessed on 03/1/2020).

[12] S. H. P. Ltd, "Sublime text official website," https://www.sublimetext.com/, (Accessed on 03/1/2020).

[13] F. S. Foundation, "Gnu emacs official website," https://www.gnu.org/software/emacs/, (Accessed on 03/1/2020).

[14] I. GitHub, "Github official website," https://github.com/, (Accessed on 03/1/2020).

[15] FIB, "Racó official website," https://raco.fib.upc.edu/, (Accessed on 03/1/2020).

[16] Brave, "Brave official website," https://brave.com/, (Accessed on 03/1/2020).

[17] Google, "Google cloud official website," https://cloud.google.com/, (Accessed on 03/9/2020).

[18] MOB, "Mob - barcelona official website," https://mob-barcelona.com/, (Accessed on 03/9/2020).

[19] A. Biere, "CaDiCaL at the SAT Race 2019," in *Proc. of SAT Race 2019 – Solver and Benchmark Descriptions*, ser. Department of Computer Science Series of Publications B, M. Heule, M. Järvisalo, and M. Suda, Eds., vol. B-2019-1. University of Helsinki, 2019, pp. 8–9.

[20] G. Audemard and L. Simon, "On the glucose SAT solver," *Int. J. Artif. Intell. Tools*, vol. 27, no. 1, pp. 1 840 001:1–1 840 001:25, 2018. [Online]. Available: https://doi.org/10.1142/S0218213018400018

[21] I. Abío, R. Nieuwenhuis, A. Oliveras, and E. Rodríguez-Carbonell, "A parametric approach for smaller and better encodings of cardinality constraints," in *Principles and Practice of Constraint Programming - 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings*, ser. Lecture Notes in Computer Science, C. Schulte, Ed., vol. 8124. Springer, 2013, pp. 80–96. [Online]. Available: https://doi.org/10.1007/978-3-642-40627-0_9

[22] C. Ansótegui, M. L. Bonet, J. Giráldez-Cru, J. Levy, and L. Simon, "Community structure in industrial SAT instances," *J. Artif. Intell. Res.*, vol. 66, pp. 443–472, 2019. [Online]. Available: https://doi.org/10.1613/jair.1.11741

[23] Z. Newsham, V. Ganesh, S. Fischmeister, G. Audemard, and L. Simon, "Impact of community structure on SAT solver performance," in *Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, ser. Lecture Notes in Computer Science, C. Sinz and U. Egly, Eds., vol. 8561. Springer, 2014, pp. 252–268. [Online]. Available: https://doi.org/10.1007/978-3-319-09284-3_20

[24] C. Ansótegui, J. Giráldez-Cru, J. Levy, and L. Simon, "Using community structure to detect relevant learnt clauses," in *Theory and Applications of Satisfiability Testing - SAT 2015 - 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings*, ser. Lecture Notes in Computer Science, M. Heule and S. A. Weaver, Eds., vol. 9340. Springer, 2015, pp. 238–254. [Online]. Available: https://doi.org/10.1007/978-3-319-24318-4_18

[25] C. Barrett and C. Tinelli, "Satisfiability modulo theories," in *Handbook of Model Checking*. Springer, 2018, pp. 305–343.

[26] R. Nieuwenhuis, A. Oliveras, and C. Tinelli, "Solving SAT and SAT Modulo Theories: from an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T)," *Journal of the ACM*, vol. 53, no. 6, pp. 937–977, Nov. 2006.

[27] R. Nieuwenhuis, "The intsat method for integer linear programming," in *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, ser. Lecture Notes in Computer Science, B. O'Sullivan, Ed., vol. 8656. Springer, 2014, pp. 574–589. [Online]. Available: https://doi.org/10.1007/978-3-319-10428-7_42

[28] O. Ohrimenko, P. J. Stuckey, and M. Codish, "Propagation via lazy clause generation," *Constraints*, vol. 14, no. 3, pp. 357–391, 2009.

[29] N. Eén and N. Sörensson, "Translating pseudo-boolean constraints into sat," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 2, no. 1-4, pp. 1–26, 2006.