Exploring Satisfiability – Bringing Theoretical Informatics to Schools

Annika Vielsack
vielsack@kit.edu
Karlsruhe Institute of Technology
Karlsruhe, Germany

Beatrice Deutschmann
beatrice.deutschmann@gmx.de
Independent
Karlsruhe, Germany

Abstract

The field of computer science comprises a broad range of different areas and topics. While some areas and topics are very present in computer science education, others rarely find their way into classrooms. One of these less common topics in education is satisfiability, despite its importance and interesting real world applications. These real world connections might act as a gateway to theoretical informatics, thus bringing more theoretical and mathematical aspects of computer science to the classroom.

In order to teach satisfiability, we have created a browser-based serious game and a collection of complementary unplugged materials. In a playful manner, students explore the DPLL-algorithm to solve simple satisfiability problems using the WebApp. Afterwards they consolidate their newly acquired knowledge with the unplugged material. Both components encourage students to explore and interact with the provided teaching material hands-on.

A study involving 24 tenth-grade students and seven experts from (computer) science and education indicates that this approach is very promising.

CCS Concepts

• **Theory of computation** → **Logic**; • **Applied computing** → **Education**; **Interactive learning environments**.

Keywords

Satisfiability, DPLL, Computer Science Education, Theoretical Informatics

ACM Reference Format:

Annika Vielsack and Beatrice Deutschmann. 2026. Exploring Satisfiability – Bringing Theoretical Informatics to Schools. In *Proceedings of the 31st ACM Conference on Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2026), July 10–15, 2026, Madrid, Spain.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3803400.3809351

1 Introduction

The aim of this work is to demonstrate that advanced concepts from theoretical informatics can be introduced to secondary school students in an engaging and motivating manner. Although such topics are often considered too abstract or demanding for this age group, we show that – when supported by an appropriate pedagogical design – they can become both accessible and meaningful. To exemplify this, we present a teaching unit that introduces students to propositional satisfiability and the DPLL algorithm through a combination of digital exploration and unplugged activities.

Propositional satisfiability (SAT) plays a central role in computer science. Given a Boolean formula, the SAT problem asks whether there exists an assignment of truth values that makes the formula evaluate to true [12]. SAT was the first problem proven to be NP-complete in the early 1970s [8], marking a milestone in computational complexity theory and shaping the study of efficient computation. Despite its theoretical depth, SAT can be represented through intuitive examples and lends itself to visual and interactive learning approaches, making it a promising topic for school-level exploration.

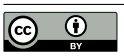
As SAT problems are NP-complete, no efficient algorithm is known that solves all instances. Nevertheless, several procedures work effectively in practice for many classes of formulae. One of the earliest approaches is the Davis–Putnam (DP) algorithm from 1960 [10]. It was later refined in 1962 as the Davis–Logemann–Loveland (DLL) algorithm [9]. The modern Davis–Putnam–Logemann–Loveland (DPLL) algorithm combines elements of both and forms the basis of many contemporary SAT solvers, which systematically search for a satisfying assignment and can therefore determine whether a propositional formula is satisfiable [4].

A central design principle of this teaching unit is inspired by the *Computer Science Unplugged* [16] approach, which has shown that complex computational ideas can be taught effectively through hands-on, game-like activities without requiring computers. Unplugged methods reduce abstraction, promote active reasoning, and make algorithmic processes tangible for learners. Building on this idea, our unit pairs a playful WebApp – which allows students to explore satisfiability and experiment with the DPLL algorithm visually – with a subsequent unplugged phase in which students perform the algorithm manually using physical materials. This combination encourages a deeper understanding: the digital environment lowers the entry barrier through guided exploration, while the unplugged activities require deliberate reasoning and help students internalise the structure of the algorithm.

In this way, we illustrate how an advanced topic from theoretical informatics can be meaningfully adapted for school contexts, and how algorithmic thinking and logical reasoning can be fostered through a carefully balanced interplay of digital and unplugged learning experiences.

2 Related Work

Research on teaching advanced topics from theoretical informatics in schools is still limited, and satisfiability in particular is rarely addressed explicitly in secondary education. While logical reasoning



ITiCSE 2026, Madrid, Spain

ACM ISBN 979-8-4007-2634-7/2026/07
https://doi.org/10.1145/3803400.3809351

and constraint-based puzzles occasionally appear in mathematics or computer science classrooms, they are typically not framed as instances of SAT, nor are students introduced to formal solving procedures such as the DPLL algorithm. Existing school-level materials tend to focus on graph-theoretical puzzles, pattern discovery tasks, or simplified logic problems, which aim to foster deductive reasoning but do not require learners to engage with formal algorithms.

Digital tools for working with SAT problems have been developed primarily for undergraduate education. GraphBench [6] provides an environment for exploring SAT instances, allowing users to modify variable assignments and observe their effects. However, it does not implement the DPLL algorithm and is designed for learners who are already familiar with propositional logic and satisfiability. LearnSAT [5] offers a text-based environment in which various DPLL-based solvers can be executed step by step. It includes the option to display decision trees, but presupposes a substantial amount of conceptual background, including familiarity with clauses, unit propagation, and backtracking. As such, these tools support practice at university level but are not directly suitable for use in secondary school settings.

Several authors have proposed general principles for designing interactive learning environments and serious games for computer science education [1, 3, 7, 15]. Gamification elements such as levels, rewards, and visual feedback have been shown to increase motivation and support sustained engagement when learning complex concepts. Research in human–computer interaction further highlights the importance of reducing cognitive load, providing clear visual cues, and offering immediate feedback – features that are particularly relevant for algorithmic topics, where the sequence of actions and the interdependence of steps must be understood precisely.

Unplugged activities have also been used to introduce complex algorithmic ideas through physical manipulation, simplified representations, or tangible metaphors [16]. These approaches can reduce abstraction, promote deliberate reasoning, and encourage students to articulate their thought processes.

The approach presented in this paper builds on these strands of research but differs from existing work in its explicit focus on satisfiability and on adapting the DPLL algorithm for secondary education. By integrating a playful WebApp with structured unplugged materials, the unit provides both exploratory access and guided consolidation, bridging a gap between theoretical informatics and school-appropriate pedagogy. To our knowledge, no prior work has presented a comparable combination of digital and analogue methods for teaching satisfiability and the DPLL algorithm at this level.

3 Background

The DPLL algorithm works on propositional formulas in conjunctive normal form (CNF), so called CNFSAT problems. Every propositional logic formula can be transformed into CNF using boolean algebra and even many constraint-satisfaction-problems (CSP) can be written as CNFSAT problems [13]. Although the transformation into CNF can itself be a meaningful learning topic, we will focus on CNFSAT problems.

For a propositional formula B over boolean variables x_i, we use the following terminology [4]:

- The formula is *satisfiable* if and only if an interpretation exists that evaluates B to *true*. This interpretation is called a *model*.
- A *literal* is a (negated) variable $l = x$ or $l = \neg x$.
- A *clause* is a disjunction of literals, e.g. $x_1 \vee x_2 \vee \neg x_3$.
- A *pure literal* is a literal l whose complement l^c is not used in any clause of B.
- A *unit clause* is a clause that consists of only one literal, e.g. $\neg x$.
- An *empty clause* does not contain any literals and is denoted as $\square$.
- If the formula is empty, we write $B = \varnothing$.
- A formula in CNF can be written as a set of clauses and a clause can be written as a set of literals, e.g. $B = (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$ can be written as $\{\{x_1, x_2\}, \{\neg x_1, \neg x_2\}\}$.

The DPLL algorithm uses a set of rules:

- *Satisfiable:* If all clauses are removed ($B = \varnothing$), the problem is satisfiable.
- *Unsatisfiable:* If chronological backtracking is completed and one clause is empty ($\square$), the problem is unsatisfiable.
- *Pure-literal rule:* If B contains a pure literal l, delete all clauses containing l.
- *Unit-literal rule:* If B contains a unit clause $\{l\}$, delete all clauses containing l and remove l^c from all remaining clauses ($red_l(B)$).
- *Chronological Backtracking:* Set one variable x_i to *true* and continue with the algorithm ($B_{x_i,t}$). If the formula is not satisfiable with $x_i = true$, set x_i to *false* and proceed accordingly ($B_{x_i,f}$).

In its original form, the DPLL algorithm is highly nondeterministic: the order in which unit clauses are processed and the choice of variables during backtracking are not fixed. Since these choices may significantly affect runtime, we introduce deterministic selection rules: variables are named alphabetically, and both unit clauses and branching variables are processed in alphabetical order. To further simplify the algorithm for educational purposes, we omit the pure-literal rule. Although this rule may reduce runtime, it does not affect satisfiability, as chronological backtracking eventually explores all necessary assignments. This leaves us with the pedagogically reduced DPLL algorithm in Algorithm 1, that is a deterministic recursive backtracking algorithm that returns if a given formula is satisfiable or unsatisfiable. Using the decisions from chronological backtracking and setting the literals removed by the unit-literal rule to *true*, the algorithm additionally provides a model for the given formula.

4 From Plugged to Unplugged – The Concept

The main aim of the presented lesson is to introduce secondary school students to a topic of theoretical informatics, namely SAT problems. Since the broader field of satisfiability is too extensive to be covered within a limited time frame, the lesson focuses on solving CNFSAT problems using the DPLL algorithm. As the goal is to execute the algorithm rather than to implement it, the unit follows an unplugged approach.

Function DPLL(B)
  if $B = \emptyset$ **then** # Satisfiable
    return true;
  end
  if $\square \in B$ **then** # Unsatisfiable
    return false;
  end
  if $\exists$ *unit clause* $\{l\} \in B$ **then** # Unit-literal rule
    return DPLL($red_l(B)$);
  else # Chronological Backtracking
    Choose x_i in alphabetical order;
    if $DPLL(B_{x_i,t})$ **then**
      return true;
    else
      return DPLL($B_{x_i,f}$);
    end
  end

Algorithm 1: The pedagogically reduced DPLL algorithm

Part 1: Introducing the DPLL algorithm	
SAT problems	Introduction to SAT problems
WebApp	Playing the serious game
Discussion	Summary and reflection
Part 2: Solving SAT problems	
Repetition	SAT problems and the DPLL algorithm
Puzzles	Working with the unplugged material
Discussion	Summary and reflection

Table 1: Exemplary lesson plan

Because the DPLL algorithm involves several interdependent rules and logical steps, students should be guided while still being encouraged to explore the underlying concepts independently. To balance exploration and structured support, we developed a serious game that introduces satisfiability and the DPLL algorithm through an interactive and playful approach.

After playing the game, students are expected to understand how the DPLL algorithm works. However, since the tool automates parts of the process, students then apply the algorithm without digital support. To enable this transfer, they are provided with unplugged materials that build on their prior experience in the game.

We designed the unit for a duration of 90 to 180 minutes, depending on the students' prior experience. They require no specific prior knowledge, though familiarity with algorithmic thinking and deterministic reasoning is beneficial. Due to the conceptual complexity of the algorithm, the lesson targets students from grade eight onwards.

The first half of the unit focuses on introducing the DPLL algorithm and the second half gives the students the opportunity to derive SAT problems from real life problems and to solve them. An overview of the lesson plan is shown in Table 1.

The overall learning objectives of the unit are:

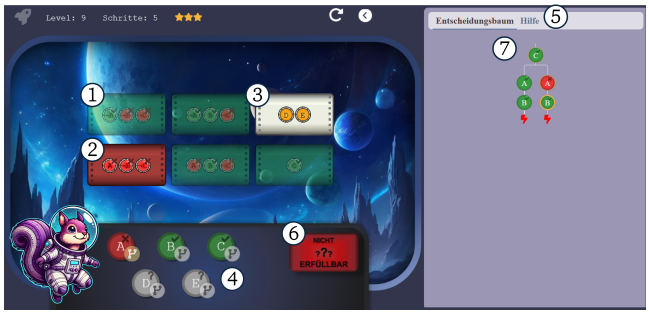


Figure 1: Screenshot of the WebApp

(LO1) Students can identify variables, literals and clauses in simple use cases.
(LO2) Students can determine whether a given problem is satisfiable by applying the DPLL algorithm.
(LO3) Students can document the execution of the DPLL algorithm using a decision tree.

Additionally, the unit fosters algorithmic thinking, particularly deterministic reasoning and backtracking strategies.

5 Grasping the Concept – The WebApp

After a short introduction to SAT problems, highlighting the real-world relevance, the students are introduced to the WebApp. It is designed to introduce the students to the DPLL algorithm step by step while maintaining a playful character.

The game includes a small story element: A space squirrel welcomes the students aboard and asks for help. The game is designed to resemble the control centre of a spaceship as seen in Figure 1. Through a large viewing window the students can see parts of a spaceship with labelled panels. The control panel has buttons with the same labels as the panels and an additional big rectangular button. On the right side of the screen, the students are provided with additional information about the algorithm that is not integrated into the story.

The user interface, shown in Figure 1, not only sets the scene but also provides a graphical representation of abstract concepts: Clauses are displayed as parts of a spaceship (1,2,3), literals are displayed as panels (1,2,3) and variables are portrayed as buttons of the control panel (4). When a variable is clicked it changes its state from *undefined* (? / grey) to *true* (✓ / green), then to *false* (× / red) and back to *undefined* (4). This alters the state of the corresponding literals. A literal whose corresponding variable is not yet assigned is displayed in yellow. Literals that evaluate to *true* are marked in green (✓), while literals that evaluate to *false* are marked in red (×).

The DPLL algorithm from Algorithm 1 deletes unit clauses and literals. Instead of removing these elements from the interface, satisfied clauses are rendered transparent and otherwise eliminated literals are visually marked as *false*. This way, the students still see the representation of the original formula, allowing them to follow the progression of the algorithm without altering its output. Moreover, by keeping the clauses and literals, the students can not only decide whether the problem is satisfiable or not, but can easily find a model using the shown values of the variables.

Level	Technical Terms	Concepts
1	Variable, Literal, Clause	Satisfiable Problems
2		Negated Literals, Unsatisfiable Problems
3	Unit Clause	Unit Propagation
4		Multiple Unit Clauses
5	Branching	Choice of variable for backtracking
6		Chronological Backtracking

Table 2: Overview of technical terms and concepts introduced in the first six levels.

To execute the algorithm in Algorithm 1 the students have to perform the four steps of the algorithm. The tool assists them while doing so.

- If all clauses are satisfied (1), the result button (6) turns green and the formula is *satisfiable* with the current variable assignment set via the control panel (4) as model.
- If there is a conflicting clause (2), i.e. all its literals evaluate to *false*, the formula could be *unsatisfiable* and the result button (6) turns red. The students have to decide if they completed chronological backtracking, i.e. the formula is not satisfiable, or if there is another case left that they have to try out.
- To apply the *unit-literal rule*, the students set the corresponding variable (4) accordingly.
- To keep track of the *chronological backtracking*, the students are provided with a decision tree (7), that automatically tracks all decisions they make while executing the algorithm. When students perform a branching decision, they click the tree symbol on the variable (4). This adds a branch in the decision tree and sets the value of said variable to *true*. If they have to come back to that decision, they can change the value of the variable to *false* and thus switch to the other branch in the decision tree.

After assigning a value to a variable, all literals and clauses are evaluated and marked according to their new stage.

As the students proceed, they unlock new levels. The first six levels function as a tutorial. The students get to know the technical terms and the components of the algorithm as described in Table 2. During the tutorial, the students can click the buttons as they like and can restart the level as often as they wish, but they have to stick to the DPLL algorithm in order to proceed to the next level.

The information from the tutorial is available throughout the whole game and can be accessed in the panel on the right side (5), where the user can switch between the decision tree, the explanation for the tutorial levels and the combined information from the tutorial about the DPLL algorithm.

Once a level is completed, i.e. the students chose correctly if the problem is satisfiable or not, they can proceed to the next level or restart the current level. After the tutorial, students are no longer required to follow the algorithm strictly and may explore their own solution strategies. As the overall goal of the application is to introduce the students to the DPLL algorithm, they are rewarded, if they do stick to the algorithm. The number of actions is counted and displayed in the top bar. If the students follow the algorithm

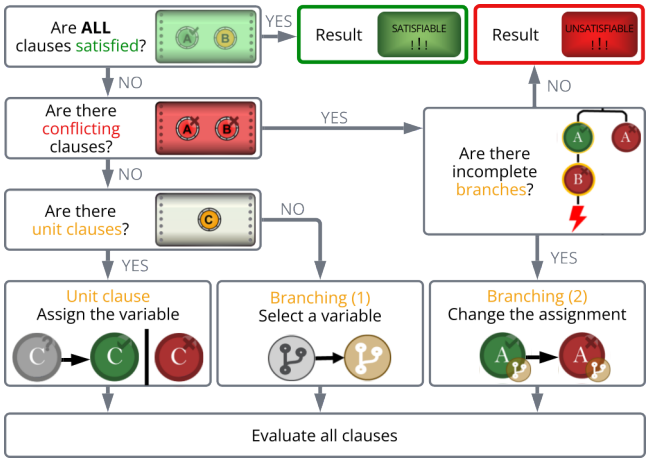


Figure 2: Flowchart of the DPLL algorithm as used within the WebApp.

accurately and perform no unintended actions, they earn three stars and a medal. If they perform more actions, they receive fewer stars. The medal is awarded if the decision tree is correct, i.e. the DPLL algorithm is executed correctly. In case the students do not remember the algorithm any more, they can use the interactive flowchart from Figure 2 where they can step through the algorithm or watch as the algorithm is executed automatically. This ensures that students always have access to a working example of the algorithm, while still being encouraged to recall the steps themselves, as the flowchart is displayed only after completing a level.

6 Consolidating the Concept – The Unplugged Part

After working with the WebApp, the students transition to an unplugged phase in which they apply the DPLL algorithm without automated guidance. While the WebApp supports learners by updating literals, highlighting satisfied clauses, and constructing the decision tree automatically, the unplugged material requires students to carry out these steps manually.

Many familiar constraint-satisfaction problems (CSPs), such as timetable construction [2, 11, 17] or Sudoku [14], can be expressed as CNFSAT problems. However, the resulting clause sets tend to be large and solving them with the DPLL algorithm would typically take significantly longer than solving the puzzle directly. In such cases, the educational advantage of applying the algorithm would not be apparent and the focus would shift from executing the DPLL algorithm to transforming CSPs into CNFSAT form. For this reason, we restrict ourselves to problems that can be translated into CNFSAT intuitively and without lengthy preprocessing

We designed four different types of puzzles as seen in Figure 3 each of which provides an intuitive scenario, such as pipe systems, logic gates, food constraints or obstacle courses. They are designed to be intuitively understandable while representing four different thematic domains, ensuring broad appeal and offering contexts that resonate with different student interests.

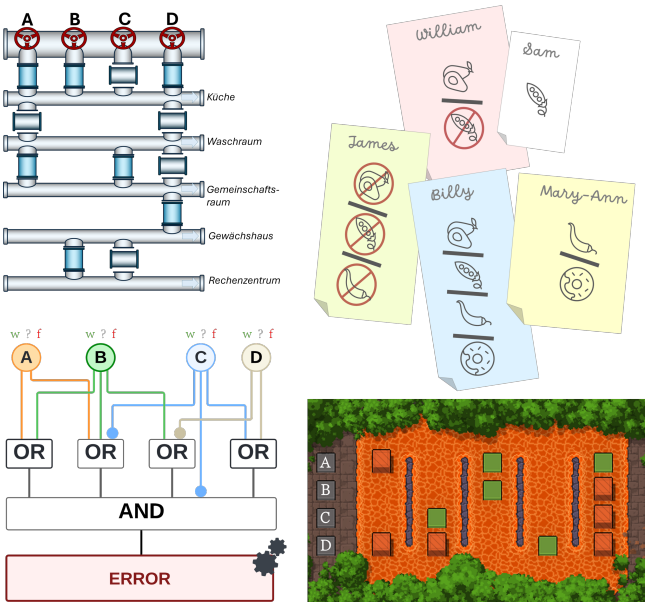


Figure 3: The four different types of puzzles used in the second part of the intervention.

The students are presented with multiple problems, each requiring the students to work with a puzzle, a game board containing a set of clauses and a decision tree as shown in Figure 4. To solve a problem, students have to complete three steps:

(1) To derive the CNFSAT problem from the scenario, the students have to choose the game board with a set of clauses that belongs to the given problem.
(2) Using the game board, the students perform the DPLL algorithm and write down the decision tree.
(3) Students then decide whether the underlying SAT problem is satisfiable and compare their decision tree with a model solution to verify the correctness of their reasoning.

The game board is at the heart of the activity. Students interact with it by placing red and green transparent tokens on the variables and literals and by marking satisfied clauses with an erasable marker. Tokens can be rearranged and markings removed whenever chronological backtracking is required. This mirrors the previous plugged activity, while requiring students to think through each step deliberately.

The material is designed to shift the focus towards applying the algorithm and to provide a possibility for self-checking. To this end, students are provided with a wide selection of game boards to choose from. This not only saves time and prevents copying errors but also requires students to think about the derivation of clauses when identifying the correct set of clauses that corresponds to each puzzle, thereby supporting learning objective (LO1).

Furthermore, students receive more than one possible solution for the decision tree. This encourages them to compare, reflect and verify their reasoning rather than simply copying the solution. It also provides the possibility for self-checking before starting the next puzzle thus reducing the possibility of forming misconceptions.

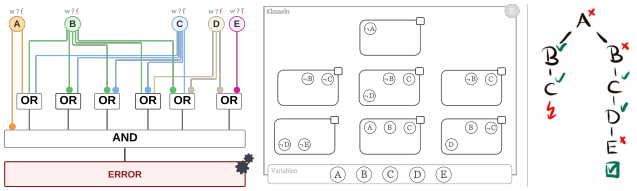


Figure 4: Puzzle, set of clauses and decision tree for one problem in the unplugged phase.

The documentation step is essential, as it reveals the reasoning process and allows both students and teachers to verify the correctness of the solution path and supports the achievement of learning objectives (LO2) and (LO3).

Overall, the unplugged component consolidates the procedural understanding gained during the plugged phase and strengthens the students' ability to execute the DPLL algorithm independently. It links abstract logical reasoning to tangible materials, promotes careful deduction, and enables learners to experience algorithmic backtracking as an active, strategic process rather than as an automated system feature.

7 Experience

To evaluate the feasibility, clarity, and motivational impact of the teaching unit, a pilot study was carried out with a secondary school class and a small group of experts. The aim of this study was not only to test the functionality of the WebApp in an authentic classroom setting but also to examine whether the combination of plugged and unplugged materials supports students in understanding and applying the DPLL algorithm.

The teaching unit was implemented in a grade 10 mathematics class at a girls' secondary school in Germany. A total of 24 students participated, working in pairs on tablet devices during the WebApp phase. The session lasted approximately 85 minutes, with around 50 minutes devoted to working with the WebApp and 15 minutes spent on the unplugged materials. The remaining time was used for instruction and administration. The students had only one year of mandatory computer science education in grade 7 and reported low familiarity with computer science.

A second group consisted of seven experts – including a teacher trainee, researchers in mathematics and computer science education, software developers and a medical student with extensive experience in digital games – who worked individually using laptops or tablets. Their feedback was intended to provide insights into the pedagogical design, usability, and conceptual clarity of the WebApp and the unplugged tasks.

The study was divided into two parts. The first part of the questionnaire was completed immediately after working with the WebApp, while the second part was completed after the unplugged activity. Due to the small number of participants, we report the results qualitatively.

Underlying Concepts. As previously mentioned, the DPLL algorithm has some complex underlying concepts. The participants understood most of them quite intuitively. The self-reported ability

to create a decision tree increased from before to after the unplugged activity. Because literals are not visually removed during unit propagation in the WebApp, some students struggled with the term *unit clause*. When using the WebApp, satisfied clauses were rendered transparent and therefore were not considered by the participants any more. When working with the unplugged material this automation was omitted. As a result, some students incorrectly considered clauses that had already been satisfied, even though these should no longer have played a role in subsequent steps.

User Interface of the WebApp. The user interface was generally perceived positively, although some issues were identified. Many participants focused primarily on the game elements and therefore overlooked the instructional hints provided in the help panel. Additionally, some suggestions for improvement were given, such as highlighting interactive elements more clearly.

Tutorial and Help. The tutorial and help were mostly perceived as valuable. Nevertheless, some hints for improvement were provided. For example, participants suggested that the algorithm should be visible in the help panel at all times and not only after completing a level and the feedback after finishing a level could be more concise.

The Concept. The overall concept of the intervention was evaluated positively. The experts in particular appreciated the combination of plugged and unplugged activities. Several students also reported that the WebApp had helped them understand the tasks and that their learning improved during the unplugged phase.

Overall, the pilot study indicates that the combined approach of digital exploration and unplugged consolidation is effective in supporting learners' understanding of satisfiability and backtracking algorithms. The WebApp provides an accessible gateway into a complex topic, while the unplugged materials ensure that students can apply the algorithm independently and reflect on their own reasoning processes. The results have informed further refinements of both the digital tool and the teaching materials and demonstrate the potential of the approach for broader classroom use.

8 Lessons Learned

The pilot study demonstrates that even conceptually demanding and comparatively unfamiliar topics from theoretical informatics can be taught successfully in secondary education when they are embedded in a carefully designed learning environment. Adapting such topics for the classroom requires thoughtful simplification, but it can open meaningful opportunities for developing algorithmic thinking, logical reasoning, and an appreciation for the foundations of computer science.

The combination of a plugged exploratory phase and an unplugged consolidation phase proved effective. The WebApp enabled students to experiment freely, receive immediate visual feedback, and develop an intuitive understanding of satisfiability and backtracking without being overwhelmed by syntactic details. At the same time, the subsequent unplugged activities required them to slow down, reason deliberately, and apply the DPLL algorithm independently of automated support.

Several observations from the implementation inform future refinements. First, students benefitted from the strongly visual and gamified design of the WebApp, which lowered the entry barrier to an otherwise abstract topic. However, some learners initially overlooked the explanatory texts and relied primarily on trial-and-error strategies, indicating that essential information should be highlighted more prominently. Second, the transition to the unplugged material revealed misunderstandings – particularly regarding unit clauses and unit propagation – that had been masked by the automated assistance in the digital environment. This underlines the importance of combining exploratory digital tools with structured analogue exercises to achieve robust conceptual understanding.

9 Conclusion

This work demonstrates that satisfiability and backtracking algorithms – topics traditionally associated with theoretical informatics – can be meaningfully introduced in secondary education when embedded in a carefully designed learning environment. By combining a playful WebApp with structured unplugged materials, the teaching unit enables students to explore the DPLL algorithm interactively and to consolidate their knowledge through deliberate, hands-on reasoning.

The WebApp lowers the entry barrier to an otherwise abstract topic by providing immediate visual feedback, intuitive representations of clauses and literals, and a gamified progression system that encourages sustained engagement. Students can experiment with satisfiability problems, observe the effects of unit propagation and backtracking, and build an initial understanding of the algorithm without being overwhelmed by syntactic detail. The subsequent unplugged phase complements this exploratory experience: without automated support, students must apply the DPLL algorithm manually, construct decision trees, and reason carefully about clause satisfaction. This phase revealed conceptual misunderstandings that had previously been masked by automation, highlighting its value for achieving robust understanding.

The pilot study, involving both secondary school students and experts from (computer) science and education, indicates that the combined approach is feasible, motivating, and pedagogically promising. Participants responded positively to the playful yet structured design, and several reported a clearer understanding of satisfiability and backtracking after completing the unplugged activities. Expert feedback further supports the didactic soundness of the approach, while also suggesting avenues for refinement, such as improving the visibility of instructional cues and enhancing the design of the help system.

Overall, this work shows that advanced concepts from theoretical informatics can be made accessible to younger learners when plugged exploration and unplugged consolidation are thoughtfully integrated. The presented unit provides a concrete example of how algorithmic thinking and logical reasoning can be fostered through a balance of guided interaction and hands-on practice. Future work may extend this approach to additional problem classes, explore adaptive scaffolding for diverse learners, or examine long-term effects on students' conceptual understanding and attitudes towards theoretical informatics.

The material is freely available under an open-source license at research.lehr-lern-labor.info/satisfiability.

References

[1] Ruedi Arnold and Werner Hartmann. 2007. Pragmatische Empfehlungen zur Entwicklung von interaktiven Lernumgebungen. In *Didaktik der Informatik in Theorie und Praxis – INFOS 2007 – 12. GI-Fachtagung Informatik und Schule*. Gesellschaft für Informatik e. V., Bonn, 171–182.

[2] Roberto Asín Achá and Robert Nieuwenhuis. 2014. Curriculum-based course timetabling with SAT and MaxSAT. *Annals of Operations Research* 218, 1 (July 2014), 71–91. doi:10.1007/s10479-012-1081-x

[3] Wolfgang Becker and Maren Metz (Eds.). 2024. *Serious Games und Gamification in der schulischen Bildung*. Springer Fachmedien Wiesbaden, Wiesbaden. doi:10.1007/978-3-658-44317-7

[4] Mordechai Ben-Ari. 2012. *Mathematical Logic for Computer Science* (3. ed ed.). Springer, London [u.a.].

[5] Mordechai Ben-Ari. 2013. LearnSAT: A SAT Solver for Education. In *Theory and Applications of Satisfiability Testing – SAT 2013*, David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Matti Järvisalo, and Allen Van Gelder (Eds.). Vol. 7962. Springer Berlin Heidelberg, Berlin, Heidelberg, 403–407. doi:10.1007/978-3-642-39071-5_30 Series Title: Lecture Notes in Computer Science.

[6] Markus A. Brändle. 2006. *GraphBench: Exploring the Limits of Complexity with Educational Software: Exploring the limits of complexity with educational software*. Ph. D. Dissertation. ETH Zurich. doi:10.3929/ETHZ-A-005128663 Artwork Size: 115 S. Medium: application/pdf Pages: 115 S..

[7] Andreas Butz, Antonio Krüger, and Sarah Theres Völkel. 2022. *Mensch-Maschine-Interaktion* (3. auflage ed.). De Gruyter Oldenbourg, Berlin Boston. doi:10.1515/9783110753325

[8] Stephen A. Cook. 1971. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing - STOC '71*. ACM Press, Shaker Heights, Ohio, United States, 151–158. doi:10.1145/800157.805047

[9] Martin Davis, George Logemann, and Donald Loveland. 1962. A machine program for theorem-proving. *Commun. ACM* 5, 7 (July 1962), 394–397. doi:10.1145/368273.368557

[10] Martin Davis and Hilary Putnam. 1960. A Computing Procedure for Quantification Theory. *J. ACM* 7, 3 (July 1960), 201–215. doi:10.1145/321033.321034

[11] Emir Demirović. 2017. SAT-Based Approaches for the General High School Timetabling Problem. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence Organization, Melbourne, Australia, 5175–5176. doi:10.24963/ijcai.2017/747

[12] Helmut Herold, Bruno Lurz, Jürgen Wohlrab, and Matthias Hopf. 2017. *Grundlagen der Informatik* (3., aktualisierte auflage ed.). Pearson, Hallbergmoos.

[13] David Kopec. 2021. *Algorithmen in Java: 32 Klassiker vom Rucksackproblem bis zu Neuronalen Netzen* (1. auflage ed.). Rheinwerk Verlag, Bonn.

[14] Inês Lynce and Joël Ouaknine. 2006. Sudoku as a SAT Problem. In *AI&M*. https://api.semanticscholar.org/CorpusID:10853506

[15] Bernhard Preim. 2010. Interaktive Systeme. 1: Grundlagen, Graphical User Interfaces, Informationsvisualisierung. Springer, Berlin, Heidelberg. doi:10.1007/978-3-642-05402-0 Num Pages: 628.

[16] Tim Bell, Ian H. Witten, and Mike Fellows. 2015. *CS Unplugged: An enrichment and extension programme for primary-aged students*. https://classic.csunplugged.org/documents/books/english/CSUnplugged_OS_2015_v3.1.pdf

[17] Roy Willemen. 2002. *School timetable construction: algorithms and complexity*. Number 2002,05 in IPA dissertation series.