



COMP390

2023/24

Puzzluoku

Student Name: Paramveer Khambay

Student ID: 201608155

Supervisor Name: Varloot Estelle

DEPARTMENT OF  
COMPUTER SCIENCE

University of Liverpool  
Liverpool L69 3BX

"Where the obstacles are so very heavy, the name of the Lord shall rescue you in an instant." - Guru Arjan Dev Ji

Dedicated to my Father, Mother, and Sister.

## Acknowledgements

I would like to thank my family for all their support. I would like to also thank my friends from home Ramandeep and Manraj for supporting me through everything. I would also like to thank my friends Ali, Baqir, Ismael, Ayman and Supra for their valued time and friendship. I would lastly like to thank Hasan and Fida, I am forever in debt to them for their emotional and academic support throughout my 3 years, it would have not been possible without them.



COMP390

2023/24

A Sudoku Solver

DEPARTMENT OF COMPUTER SCIENCE

University of Liverpool Liverpool  
L69 3BX

## Abstract

This dissertation presents the development of "Puzzluoku," an innovative application designed to solve Sudoku puzzles using a SAT solver. The project addresses the need for a specialised solver that can efficiently manage the complexity and variety of Sudoku puzzles without compromising on speed or accuracy. Leveraging the power of C#, the application combines sophisticated computational algorithms with a user-friendly interface, catering to both novices and seasoned Sudoku enthusiasts. The core of the project explores the computational theory behind Sudoku and its NP-completeness, employing advanced algorithmic strategies for effective problem-solving within a practical application context.

"Puzzluoku" exemplifies the successful integration of theoretical computer science with practical software development, setting a new benchmark for puzzle-solving applications. The dissertation details the systematic approach taken in designing, implementing, and testing the application, highlighting the challenges, and learning outcomes encountered along the way. Through this project, a deeper understanding of SAT solvers and their application to Sudoku puzzles is achieved, demonstrating the potential of such technologies to enhance user engagement and problem-solving efficiency in digital platforms.

## Statement of Ethical Compliance

Data Category: A

Participant Category: 2

I confirm that I have read the ethical guidelines and will follow them during this project. Further details can be found in the relevant sections of this proposal.

Table of Contents

## Table of Contents

Acknowledgements.....	3
Abstract.....	5
Statement of Ethical Compliance.....	5
Introduction & Background .....	7
Design.....	11
Implementation .....	17
Testing.....	24
Project ethics .....	27
Conclusion and Future Work .....	27
BSC Criteria .....	28
References .....	29

# Introduction & Background

In the evolving field of digital solutions for logic-based entertainment, the intersection of user-friendly software and sophisticated computational algorithms presents a significant opportunity for innovation. My dissertation introduces "Puzzluoku," an application designed to transform the experience of solving Sudoku puzzles through a SAT solver. This software is not merely a utility for automating puzzle solving but a comprehensive platform aimed at enhancing user interaction with one of the most popular puzzles globally. The initiative behind Puzzluoku is rooted in a dual objective: to simplify the resolution of puzzles through advanced algorithmic support and to enrich the user experience by offering an intuitive and interactive interface.

Sudoku puzzles challenge solvers to fill a grid with digits under strict constraints, a task that can be daunting and time-consuming. Traditional methods of solving these puzzles, either manually or via generic computational solvers, do not cater specifically to the nuanced variations of Sudoku that enthusiasts often encounter. The primary motivation for developing Puzzluoku is to address these gaps by providing a tailored, efficient, and engaging solution that appeals to a broad spectrum of Sudoku enthusiasts, from novices to seasoned experts.

The code for the "Puzzluoku" Sudoku solver project is written in C#. This choice was made for several strategic reasons. First, C# offers a robust and versatile environment with strong support for object-oriented programming principles, which are essential for structuring complex software solutions like a SAT solver. The language's rich set of libraries, particularly for handling data structures and performing logical operations, significantly streamlines the development process. Additionally, C# is well-integrated with the .NET framework, providing powerful tools for developing user interfaces, which enhanced the usability and accessibility of the "Puzzluoku" solver. Moreover, C# offers excellent debugging and testing tools within Visual Studio, which greatly facilitated the tracking of logical errors and performance optimisation. The choice of C# was thus driven by the need for a reliable, efficient, and developer-friendly programming environment that could support the intricate and performance-sensitive nature of the project.

## **Problem Statement and Project Motivation**

The challenge at the heart of Puzzluoku is twofold. Firstly, there is a need for a solver that can handle the complexity and variety of Sudoku puzzles without compromising on speed or accuracy. Current solutions often fall short in optimising for more complex grids or fail to provide a user-friendly interface that can engage users effectively. Secondly, the interaction model for puzzle-solving applications often lacks personalisation and adaptability, features that could significantly enhance user satisfaction and engagement.

The development of Puzzluoku is driven by the aspiration to fill these gaps. By creating a solver that not only performs with high efficiency but also integrates seamlessly into a pleasant user interface, this project aims to set a new standard for how puzzle-solving applications should operate. Furthermore, the application seeks to foster a sense of community and continuous engagement through features that allow users to track their progress, save puzzles, and interact with a global community of like-minded enthusiasts.

## **Background**

Sudoku, a logic-based number placement puzzle, has captured the interest of both enthusiasts and researchers due to its intriguing mathematical structure and computational complexity. Central to this dissertation is the exploration of Sudoku from the perspective of computational complexity, particularly its relationship with the Boolean satisfiability problem (SAT), NP-completeness, and the various algorithms designed to solve or analyse it.

### **Sudoku and Computational Complexity**

Sudoku puzzles involve filling a 9x9 grid with digits such that each column, each row, and each of the nine 3x3 sub grids contain all the digits from 1 to 9. This simple set of rules belies the often complex and challenging nature of the puzzles, especially as the number of pre-filled "hint" digits decreases.

The computational analysis of Sudoku touches on the concept of NP-completeness, a category of decision problems to which Sudoku belongs. NP-complete problems are significant in theoretical computer science because they are solvable in polynomial time by a nondeterministic Turing machine and any NP problem can be reduced to any of them. The general problem of solving Sudoku is NP-complete, which implies that there is no known algorithm that can solve all Sudoku puzzles quickly (i.e., in polynomial time), and it is generally believed that no such algorithm exists.

### **The NP-Completeness of Sudoku**

Sudoku, a popular puzzle involving a grid of numbers, is more than just a leisurely activity; it presents intriguing challenges from the perspective of computational theory. This section delves into the computational complexity of Sudoku, focusing on its classification as an NP-complete problem and the implications thereof.

The concept of NP-completeness, a cornerstone of computational complexity theory, provides a framework for. The NP-completeness of Sudoku can be demonstrated through a reduction from the 3-SAT problem, which is a well-known NP-complete problem. This involves constructing a Sudoku puzzle that is solvable if and only if a given instance of 3-SAT is satisfiable. The process of reduction from 3-SAT to Sudoku underscores the computational depth of Sudoku, revealing that solving the general case of Sudoku is as hard as any of the classic NP-complete problems.

The implications of this are significant for both theoretical computing and practical applications. It suggests that, except for certain special cases, no algorithm can efficiently solve every possible Sudoku puzzle. This has led researchers to explore various computational strategies, including heuristic methods and approximation algorithms, which seek to solve Sudoku puzzles in reasonable timeframes even though they may not guarantee a solution in every case.

### **Algorithmic Approaches to Solving Sudoku**

Given its NP-completeness, various algorithmic strategies have been applied to solve Sudoku puzzles. These include backtracking, brute-force algorithms, and more sophisticated techniques like constraint propagation and logical deduction. However, as with other NP-complete problems, these methods can struggle with particularly complex puzzles, exhibiting exponential time complexity in the worst case.

For instance, one common approach is to use constraint programming, which treats Sudoku as a set of constraints that must be satisfied. This method is quite powerful for most puzzles but can degrade in performance as puzzles increase in complexity or as constraints become more restrictive. Similarly, SAT solvers have been adapted to handle Sudoku by encoding the puzzle as a Boolean satisfiability problem. This transformation leverages advanced features of SAT solvers, such as clause learning and heuristics for variable selection, to tackle difficult Sudoku puzzles.

### **Sudoku as a SAT Problem**

The transformation of Sudoku puzzles into SAT problems provides a bridge between discrete mathematics and algorithmic logic. This approach models the puzzle as a satisfiability problem where each cell contains a logical variable that can be true (if the cell contains a certain number, say 1) or false (if it does not). This encoding leads to a set of clauses where the solution to the SAT problem corresponds to a solution to the Sudoku puzzle.

One of the most effective encodings uses the polynomial 3-SAT reduction, where the Sudoku grid is translated into a Boolean formula composed of clauses with three literals. This method allows the application of efficient SAT solvers to tackle even the most challenging Sudoku puzzles. Research has shown that certain Sudoku puzzles, which are particularly difficult for humans, also present significant challenges to SAT solvers, marking them as hard instances within the realm of computational problem-solving.

This background section establishes the complex interplay between Sudoku as a recreational puzzle and its representation within the realm of computational theory. The NP-completeness of Sudoku places it among the most intriguing of puzzles from a computational perspective, offering a fertile ground for applying and testing a variety of algorithmic approaches. The conversion of Sudoku to a SAT problem not only provides a robust framework for solving it using computational tools but also enhances our understanding of NP-complete problems and the capabilities of SAT solvers. This theoretical foundation sets the stage for the exploration of specific algorithms and their implementation in solving Sudoku puzzles, as detailed in subsequent sections of this dissertation.

#### **Paper 1: "Solving NP-Complete Problem Using ACO Algorithm" [1]**

This paper delves into the application of Ant Colony Optimisation (ACO) algorithms to NP-complete problems, with a particular focus on the Sudoku puzzle. The discussion begins by outlining the foundational principles of ACO, originally developed for the Travelling Salesman Problem (TSP), and its adaptation to tackle other complex problems, including Sudoku. The paper emphasises the inherent challenges posed by NP-complete problems—categories of problems for which no known polynomial-time algorithm can guarantee a solution for all cases. Sudoku, a puzzle typically played on a 9x9 grid, falls into this category, primarily because solving it involves determining the validity of filling the grid according to a set of rules, a task that grows exponentially harder with larger grid sizes.

The adaptation of ACO to Sudoku involves simulating the behaviour of ants searching for food, where paths between the food source and the colony are analogous to potential solutions of the puzzle. In this metaphor, pheromone levels on paths guide the ants, encouraging them to explore promising solutions more frequently. The paper describes how each 'ant' in the algorithm represents a potential solution to the Sudoku puzzle, with pheromone trails indicating the quality of these solutions based on previously explored paths. This method leverages the collective memory of previous attempts to intensify the search around the most promising areas of the solution space.

The theoretical underpinning provided by the paper centres on how ACO algorithms manage the complexity of NP-complete problems like Sudoku by incrementally constructing solutions and utilising a probabilistic method to decide between multiple paths. This approach allows the algorithm to escape local optima—a common problem in traditional optimisation algorithms—by continuing to explore less pheromone-rich paths. The paper further explores the modification of the standard ACO algorithm to enhance its performance specifically for Sudoku, highlighting the dynamic adjustment of pheromone levels to prevent premature convergence on suboptimal solutions.

Moreover, the paper positions the discussion within the context of ongoing research into metaheuristic algorithms, suggesting that understanding and improving these algorithms' performance on NP-complete problems can lead to more robust solutions across a range of computational issues. The adaptability of ACO to different problem structures, as demonstrated by its application to both TSP and Sudoku, underscores the potential of evolutionary algorithms to evolve and improve over time, much like the biological systems they emulate.

#### **Paper 2: "Techniques for Solving Sudoku Puzzles" [2]**

The paper discusses the application of backtracking—a depth-first search algorithm that tries to build a solution incrementally, abandoning a path as soon as it determines it cannot possibly lead to a successful complete solution. While effective for smaller or less complex Sudoku puzzles, backtracking suffers from scalability issues, often leading to exponential time complexity with larger or more complex grids.

Simulated annealing, inspired by the physical process of heating and then slowly cooling a material, is another technique explored in this paper. It is used to find an approximate global optimum of a given function by trying to avoid getting trapped in smaller local optima. This method is particularly effective in escaping local minima, making it useful for more challenging Sudoku puzzles that may defy simpler algorithms.

The third technique discussed is the method of alternating projections, used to find a feasible point in the intersection of a set of convex sets, which can be adapted to find solutions by iteratively projecting onto sets of constraints defined by the Sudoku rules. This method's

mathematical underpinnings provide a robust framework for handling the constraints typical of NP-complete problems, allowing it to contribute to a solution space exploration that respects the puzzle's rules.

In examining these methods, the paper emphasises that while some strategies may succeed in smaller or simpler instances, they may not scale efficiently due to the inherent complexity of NP-complete problems. It highlights the need for a variety of strategies to address different aspects of such problems effectively.

The theoretical importance of this discussion lies in its contribution to understanding how different algorithms manage the intrinsic complexity of NP-complete problems. Each algorithm offers a unique approach to navigating the solution space of these computationally intensive problems, providing insights into both their limitations and their potential. This exploration helps bridge the gap between theoretical computational complexity and practical problem-solving strategies in the field of computer science.

#### **Paper 3: "Polynomial 3-SAT Reduction of Sudoku Puzzles"** [3]

This paper provides an in-depth examination of the NP-completeness of Sudoku by demonstrating how Sudoku puzzles can be reduced to the Polynomial 3-Satisfiability Problem (3-SAT), which is a classic NP-complete problem. The reduction process explained in the paper not only underscores Sudoku's computational complexity but also frames it within the broader context of theoretical computer science, providing a bridge between practical puzzle solving and computational complexity theory.

The paper begins by outlining the rules of Sudoku and its interpretation as a logic-based constraint satisfaction problem. It then proceeds to describe how each Sudoku puzzle can be expressed as a boolean formula, where each cell in the puzzle is associated with a boolean variable whose value corresponds to the presence of a specific number. The constraints of Sudoku are translated into clauses in a boolean formula, making the puzzle a specific instance of 3-SAT.

This reduction is significant as it allows the use of tools and techniques developed for 3-SAT to solve Sudoku puzzles. Moreover, it highlights the inherent difficulty of solving Sudoku, as 3-SAT is well-known in computational theory for its NP-completeness, meaning that there is no known algorithm that can solve all instances of the problem in polynomial time. This theoretical exploration deepens the understanding of Sudoku's complexity and situates it firmly within the class of problems that are central to NP-completeness discussions.

Furthermore, the paper discusses various implications of this reduction, including how it helps in understanding the limits of algorithmic approaches to Sudoku. By establishing that Sudoku can be reduced to 3-SAT, the paper not only reaffirms the puzzle's position as an NP-complete problem but also suggests that techniques effective for 3-SAT might be adapted for Sudoku. This includes heuristic methods, exact algorithms, and approximation techniques that have been developed to tackle 3-SAT problems.

The discussion extends to the practical implications of this theoretical understanding, suggesting that insights gained from the study of 3-SAT could lead to more efficient or more robust algorithms for solving particularly difficult instances of Sudoku. This cross-pollination of methods between different areas of computational theory exemplifies the dynamic nature of research in NP-completeness and the continuous search for better solutions to these challenging problems.

In conclusion, the paper not only clarifies the computational standing of Sudoku within NP-completeness but also enriches the dialogue between practical problem-solving and theoretical computer science. It encourages a re-evaluation of the strategies used to tackle Sudoku and similar puzzles, advocating for a more nuanced approach that leverages advanced theoretical insights to enhance practical applications.

#### **Paper 4: "NP-Completeness and Heuristic Solutions for Sudoku"** [4]

This paper delves into the NP-completeness of Sudoku and explores various heuristic methods that can be used to tackle this and other NP-complete problems. It provides a comprehensive analysis of the limitations and potentials of heuristic solutions when applied to problems that cannot be solved efficiently by traditional algorithmic approaches.

The paper begins with a detailed explanation of why Sudoku is considered NP-complete, referencing its complexity in terms of the exponential number of possible arrangements that satisfy the puzzle's constraints. It then explores the application of heuristic methods such as genetic algorithms, particle swarm optimisation, and others, illustrating how these strategies can provide effective solutions for Sudoku by exploring the solution space in a non-deterministic manner.

The core of the discussion focuses on the trade-offs between the completeness, accuracy, and efficiency of heuristic solutions. While these methods do not guarantee finding the optimal solution, they are particularly valuable for finding good enough solutions within a reasonable time frame, which is often acceptable for practical purposes like puzzle-solving.

The paper also highlights the adaptability of heuristic methods to the specific characteristics of Sudoku, discussing parameter tuning and the customisation of heuristics to better exploit the structure of the puzzle. This adaptability is crucial for enhancing the performance of heuristic algorithms in specific contexts, demonstrating a key strength of heuristic approaches in dealing with NP-complete problems.

In conclusion, the paper makes a significant contribution to the understanding of NP-completeness by showcasing how heuristic solutions can be effectively tailored to address the complexities of such problems. It advocates for a balanced approach to problem-solving that recognises the limitations of traditional algorithms and leverages the strengths of heuristic methods to achieve practical results. This approach fosters a more inclusive understanding of computational complexity and encourages ongoing innovation in the development of solutions for NP-complete problems.

#### **Paper 5: "Sudoku as a SAT Problem"** [5]

This paper introduces two SAT encodings for Sudoku: minimal and extended. The minimal encoding sufficiently characterises the constraints of Sudoku puzzles using a concise number of clauses. In contrast, the extended encoding includes additional redundant clauses that improve solver performance on harder puzzles. This paper provides insights into how different SAT solvers respond to these encodings and illustrates that while simple puzzles might be solvable with minimal encoding, the more complex ones benefit significantly from the extended encoding. The paper serves as a practical guide on the benefits of redundant encoding in enhancing the efficiency of SAT solvers when applied to puzzles of varying difficulties.

#### **Paper 6: "SUDOKUSAT—A Tool for Analysing Difficult Sudoku Puzzles"** [6]

This research focuses on using SAT solving as a method to classify the difficulty of Sudoku puzzles. It introduces SUDOKUSAT, a tool that utilises different SAT resolution techniques to assess and categorise Sudoku puzzles by difficulty. The paper particularly examines the effectiveness of unit resolution combined with failed literal propagation. Their findings suggest that while basic techniques can solve simpler puzzles, more complex resolution techniques are required to handle the most challenging puzzles, known as 'AI Escargot' and others of similar difficulty. This paper emphasises the practical application of SAT solving techniques in distinguishing puzzle difficulties, contributing to the development of more challenging Sudoku puzzles.

#### **Paper 7: "SAT Encodings for Sudoku Problems"** [7]

This paper explores different SAT encodings tailored for Sudoku, each designed to optimise the efficiency of the SAT solvers. It contrasts direct encoding, where each possible number placement is encoded as a boolean variable, with more complex encoding schemes that include additional constraints to reduce the solution space. The paper argues that while direct encoding is simple and intuitive, it often leads to larger and more complex SAT problems. Instead, they propose optimised encoding schemes that strategically add constraints to the SAT formulation to pre-emptively eliminate large swathes of invalid solutions, thus streamlining the solving process. This approach significantly reduces the computational burden on SAT solvers, making it a valuable strategy for tackling the most challenging Sudoku grids.

#### **Paper 8: "Advanced SAT Techniques Applied to Sudoku"** [8]

This paper investigates advanced SAT solving techniques and their application to solving Sudoku puzzles. It introduces techniques like clause learning, conflict-driven backtracking, and heuristic-based variable selection, which are not typically used in straightforward SAT applications but can be highly effective in complex Sudoku scenarios. The paper demonstrates through experimental results how these advanced techniques can dramatically improve the efficiency of solving even the most difficult Sudoku puzzles. By implementing these advanced strategies, the paper showcases the potential of SAT solvers to not only solve puzzles but also assist in analysing the logical structure of Sudoku to understand why certain puzzles are inherently more difficult than others.

### **Aims and Requirements Analysis**

The design and functionality of Puzzluoku are guided by a clear set of aims and technical requirements, derived from a thorough analysis of existing solutions and potential user needs. The following sections detail these requirements, which serve as the blueprint for the application's development.

#### Aims:

1. Develop a specialised SAT solver that efficiently solves Sudoku puzzles by translating the puzzles into a SAT problem, leveraging logical deductions rather than brute force methods.
2. Design an interactive user interface that simplifies the process of puzzle input, offers intuitive navigation, and displays solutions in a clear and understandable manner.
3. Ensure adaptability and scalability of the application to handle various puzzle complexities and large user volumes without performance degradation.

#### Functional Requirements:

1. Puzzle Input and Validation: Users must be able to input puzzles in multiple formats and sizes, with the system capable of validating the integrity of the inputs.
2. Dynamic SAT Solver: Implement a backend SAT solver that processes inputs and returns accurate solutions promptly.
3. Solution Display: Post-solution, the application should graphically display the completed puzzle in a format that is easy to understand.
4. User Account Management: Provide secure user authentication and account management capabilities, including the ability to save and retrieve past puzzles.
5. Error Handling and Notifications: Detect and manage incorrect puzzle inputs, notifying users of errors and providing tips for correction.

#### Non-Functional Requirements:

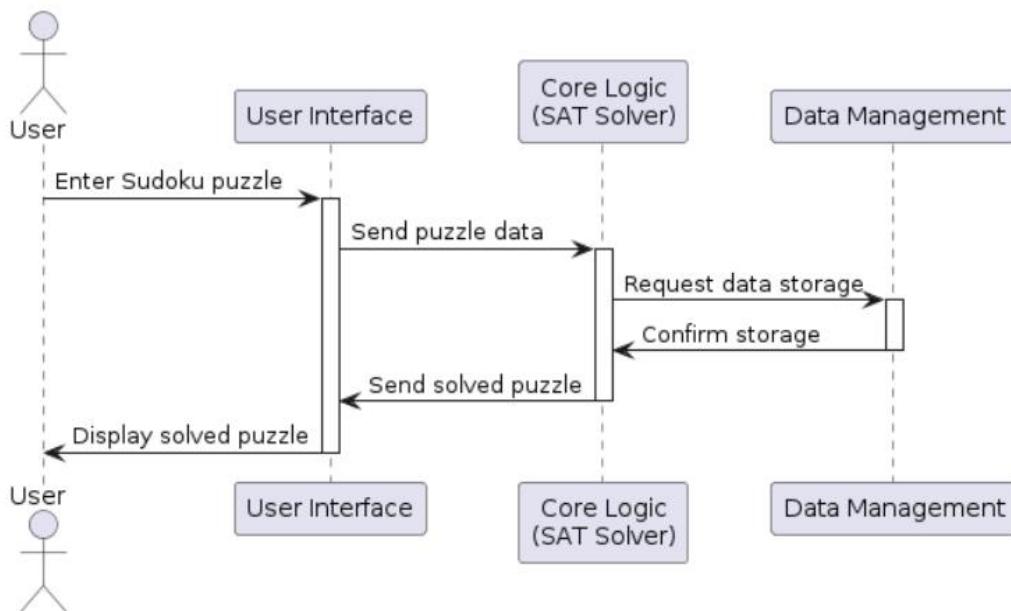
1. Usability: The UI/UX design should cater to all user levels, with a focus on simplicity and ease of use to ensure a low learning curve.
2. Performance: The application must perform efficiently under varying loads, ensuring quick response times even for complex puzzles.

3. Security: Implement robust security measures to protect user data and privacy, including encrypted data storage and secure communication channels.
4. Accessibility: Design the interface to be accessible on various devices and screen sizes, ensuring a responsive and adaptive user experience.

In summary, Puzzluoku is positioned to be a pioneering solution in the digital puzzle-solving space, blending advanced algorithmic processing with a user-centric design ethos. The subsequent sections of this dissertation will delve into the technical implementation of these requirements, demonstrating how each element contributes to the overarching goal of enhancing the Sudoku solving experience. Through this meticulous approach to design and development, Puzzluoku aims to meet the needs of its users and redefine the standards of interactive puzzle-solving platforms.

## Design

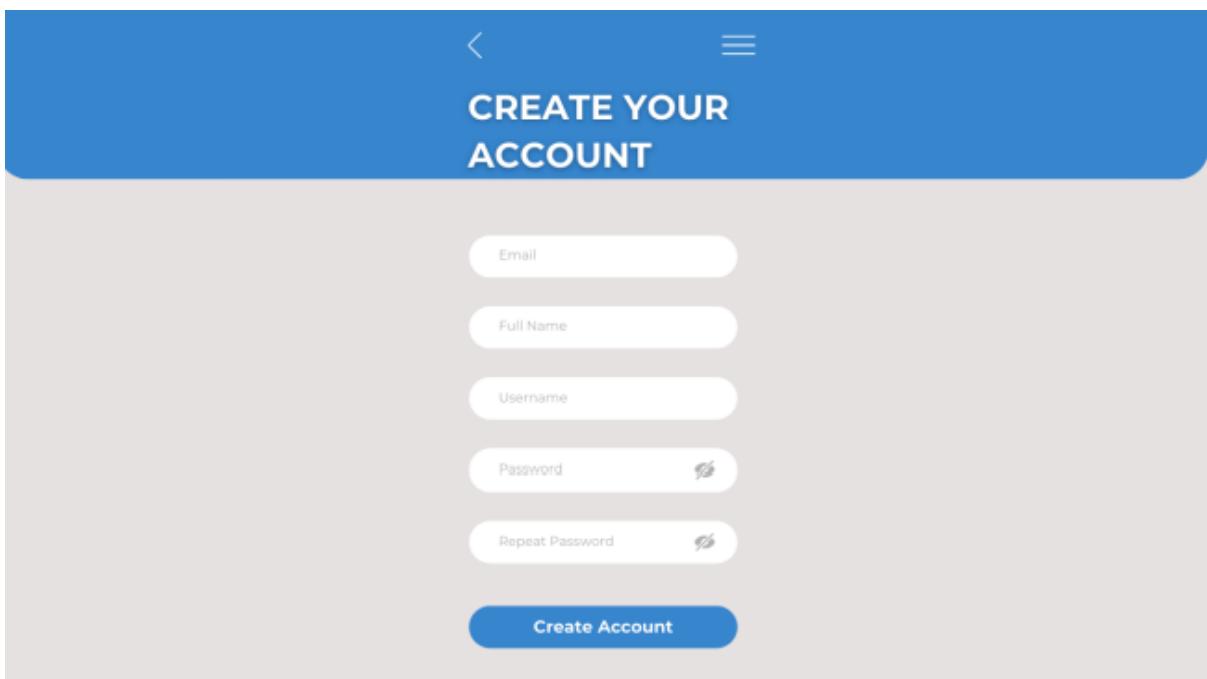
The planned architecture for the "Puzzluoku" Sudoku solver project is designed to encompass several key components that work together to provide a robust and efficient solution for solving Sudoku puzzles using a SAT solver. The system is structured into three primary layers: the User Interface (UI), the Core Logic (which includes the SAT solver), and Data Management. Each component is essential for ensuring the functionality and user experience of the system are optimal. Below is a UML diagram showcasing the overall architecture of the system.



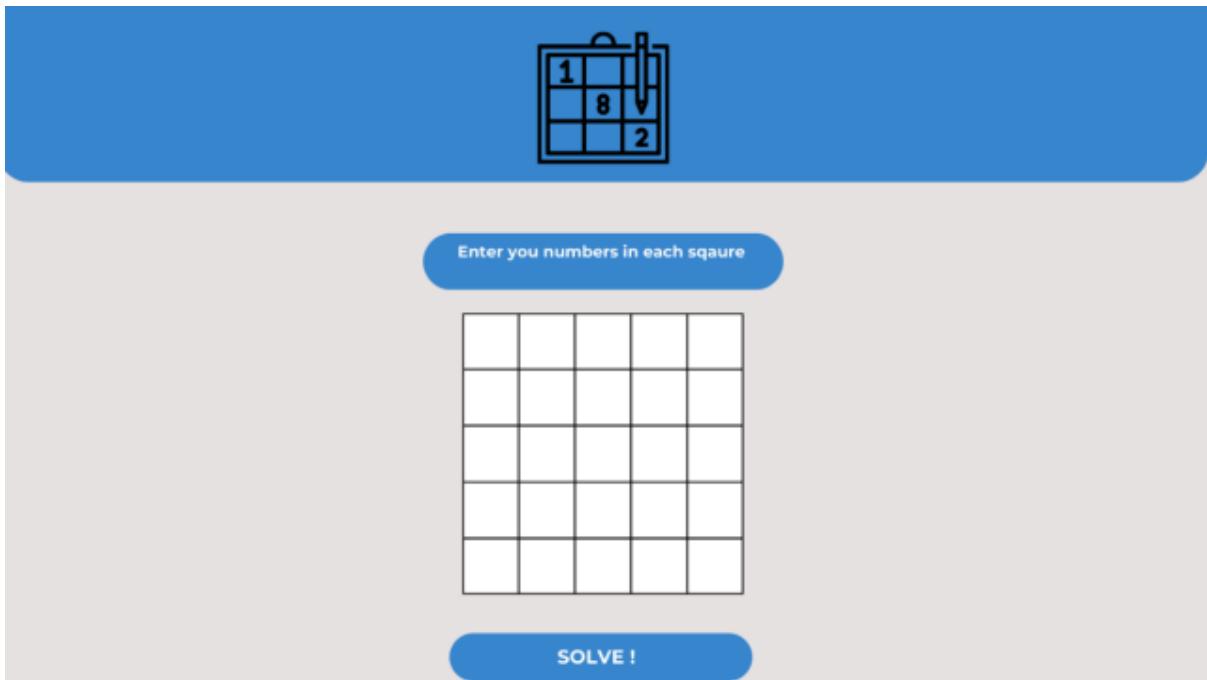
**User Interface (UI):** The UI serves as the front-end of the system. It is designed to be intuitive and user-friendly, allowing users to effortlessly input Sudoku puzzles, initiate the solving process, and view solutions. The initial plan includes developing the UI as part of a web application to ensure accessibility and ease of use across different platforms and devices. This web-based approach aims to make the "Puzzluoku" solver accessible to a broader audience, providing a seamless experience whether accessed via desktop or mobile devices. To implement this, I will be leveraging modern web development frameworks and technologies such as React or Angular for the frontend, therefore enhancing the interactivity and responsiveness of the UI.



The above figure shows a clean and minimal design for the login page of my app allowing for the user to create an account or allow them to reset their password.



This is a snippet of what the create an account page would look like ensuring that the individual provides all the necessary credentials.



Above is a mock up of what a sudoku grid would look like ( please note that the grid in implementation would be a 9x9 grid ).

**Core Logic (SAT Solver):** At the heart of the "Puzzluoku" system is the SAT solver, responsible for the logical processing and solving of the Sudoku puzzles. The decision to implement the solver in C++ is driven by the language's performance efficiencies and control over system resources, which are critical for the performance-intensive processes involved in solving SAT problems. C++ offers precise control over both memory management and system operations, making it an ideal choice for implementing complex algorithms that are both efficient and scalable. C++ provides granular control over system resources, which is crucial for optimising the performance of compute-intensive applications. The ability to manage memory allocation and deallocation explicitly allows the solver to handle large datasets and complex algorithms efficiently. This is particularly important for a SAT solver, which may need to handle significant amounts of data and solve complex logical constructs under tight performance constraints.

In addition to this, C++ offers close-to-hardware programming capabilities, which can significantly enhance the performance of the system. This access allows for optimisations that are not possible in higher-level languages, such as tweaking the way memory is accessed, optimising cache usage, and reducing overheads that can slow down a system. For the SAT solver, this means improved execution speeds and enhanced handling of the logic processing required to decode and solve Sudoku puzzles.

C++'s compatibility with powerful performance profiling tools like Valgrind, GProf, or Visual Studio's built-in profiler enables detailed analysis of the solver's performance. These tools can help identify bottlenecks in the code, such as inefficient algorithms or memory leaks, which could significantly degrade performance. By analysing how each component of the solver performs under different conditions, optimisations can be made to enhance efficiency.

Another reason as to why I intend to use C++, as not only can the performance of the solver be assessed in detail, but the insights gained from such assessments can be directly applied to make immediate and impactful optimisations. For instance, understanding memory usage patterns and CPU cycles can lead to specific changes in how data structures are implemented or how algorithms are executed. This direct feedback loop is crucial for iterative development where each optimisation cycle directly contributes to a more refined and efficient solver. [9] [10]

**Data Management:** This layer is concerned with the storage, retrieval, and management of puzzle data and user sessions. It ensures that user inputs, puzzle states, and solution histories are handled accurately and securely. For effective data management, the system will integrate with robust database systems to facilitate operations like storing ongoing game states, retrieving past puzzles, and managing user profiles. For data handling I aim to use firebase. As a platform developed by Google, Firebase offers a wide range of services designed to address common application needs such as database management, user authentication, analytics, and more. [11]

One of the standout features of Firebase is its Realtime Database. This cloud-hosted database allows data to be stored and synchronised in real-time among users. It's incredibly suitable for interactive applications because it ensures that the user experience is seamless and responsive, as data updates are instantaneously reflected across all connected clients.

Additionally, Firebase provides robust Authentication features, which support various methods including passwords, phone numbers, popular federated identity providers like Google, Facebook, and Twitter, and more. This simplifies the process of building a secure login system, while also ensuring that user data is protected. Firebase Authentication is integrated with other Firebase services and offers comprehensive security features to safeguard against threats and ensure that user credentials are managed securely.

Overall, Firebase is a powerful tool that can significantly accelerate development time by providing backend services that are easy to implement and manage. Its scalability, real-time data capabilities, and strong security measures make it an ideal choice for developing sophisticated, interactive, and secure applications. [12]

**Integration and Workflow:** The architecture is designed to ensure seamless integration between these components. Users will interact with the web-based UI to input Sudoku puzzles, which are then forwarded to the core logic where the SAT solver processes them. The results are managed and stored by the data management component before being displayed back to the user through the UI. This workflow promises a cohesive operation from input through to solution, providing a smooth and responsive user experience.

To facilitate the integration of the web-based UI, typically developed in a high-level language like JavaScript or TypeScript, with the C++ core logic, we employ a well-defined API endpoint. This API acts as a bridge, allowing the two different programming environments to communicate effectively. The API endpoint will handle requests from the UI, pass them to the SAT solver for processing, and then return the solved puzzles back to the UI. This setup not only streamlines the interaction between the front-end and back-end technologies but also encapsulates the complexity of the SAT solving process, making it transparent to the user. [13]

By creating this API endpoint, we also open up possibilities for future scalability, such as integrating additional services or enhancing the system's capabilities without significant disruptions to the existing user interface. The use of two different programming languages, each suited to their tasks—C++ for intensive logic computations and a web-friendly language for the UI—ensures that each component of the system can be optimised for performance and maintainability. This strategic division of responsibilities underpins a robust, efficient, and scalable architecture.

In the development of the "Puzzluoku" Sudoku solver, employing the appropriate data structures is crucial for achieving both functionality and efficiency. Among the choices made, 2D arrays stand out as a fundamental structure for representing Sudoku grids within the system. This choice is driven by both the nature of Sudoku puzzles and the requirements of the SAT solver algorithm.

A Sudoku puzzle is a 9x9 grid, which can be perfectly represented as a two-dimensional array in programming. Some of the reasons as to why I have chosen to use 2D arrays in my project.

- **Direct Representation:** The structure of a 2D array closely mirrors the physical layout of a Sudoku grid, where rows and columns are clearly defined. This direct correspondence makes it intuitive to implement and manipulate the grid in code, as each array index directly translates to a row and column in the puzzle.
- **Ease of Access:** Accessing or modifying a cell in the grid is straightforward and efficient with a 2D array. You can access any cell directly using its row and column indices, which is highly efficient (constant time complexity,  $O(1)$ ).
- **Efficiency in Iterations:** Iterating through a 2D array to apply rules or check conditions is efficient and simple. This is particularly beneficial for functions that need to check every row, column, or the 3x3 sub grids common in Sudoku. The two-level structure of the array allows for natural loops over rows and columns, facilitating quick checks and updates.
- **Contiguous Allocation:** In languages like C++, 2D arrays are typically stored in row-major order, meaning all elements of a row are stored in contiguous memory locations. This increases the locality of reference and can improve cache utilisation during operations that scan rows, a common requirement in Sudoku solving algorithms.

Many Sudoku solving techniques and algorithms can be implemented more straightforwardly with 2D arrays. Whether checking for duplicates within rows and columns, implementing backtracking solutions, or setting up constraints for the SAT solver, the 2D array structure aligns well with these operations. For the SAT solver integration, translating a 2D array structure into SAT expressions is systematic. Each cell's possible values can be encoded into Boolean variables efficiently, leveraging the grid's structure for generating constraints and expressions. [14] [15] [16] [17]

The core logic of the "Puzzluoku" Sudoku solver will be underpinned by a SAT solver. This solver will leverage several sophisticated algorithmic strategies to efficiently tackle the problem of solving Sudoku puzzles, which are translated into SAT (Boolean satisfiability) problems. Here are the key algorithms planned for implementation:

- 1) Backtracking
  - a) Functionality: Backtracking is a depth-first search approach for traversing the search space of possible solutions until a solution is found or all possibilities are exhausted. It will be used in the SAT solver to systematically explore possible assignments for the variables representing Sudoku cells.
- 2) Unit Propagation
  - a) Functionality: Unit propagation is a rule-based simplification technique used to reduce the problem size and make further deductions based on current variable assignments. If a variable assignment results in a clause where all literals are false except one, that one must be true.
- 3) Clause Learning
  - a) Functionality: Clause learning is a technique used to prevent the solver from revisiting the same conflicting state. When a contradiction is discovered, the SAT solver analyses the decisions that led to the contradiction and learns a new clause that prevents the same sequence of decisions.

To enhance the performance of the SAT solver in solving Sudoku puzzles, several optimisation techniques will be considered and potentially implemented: [18] [19] [20]

- 1) Heuristic Variable Selection
  - a) Description: This involves choosing which variable to assign a value to next in a way that is likely to lead to a solution more quickly. Common heuristics include the "Minimum Remaining Values" (MRV) which selects the variable with the fewest possible values left.
- 2) Two-Watch Literal Scheme
  - a) Description: A technique used in modern SAT solvers where two literals in each clause are watched, and the clause is only reevaluated for unit propagation if one of these literals is changed.
- 3) Conflict-Driven Clause Learning Optimisation
  - a) Description: Improving the efficiency of clause learning by selectively learning clauses that are likely to be useful in the future, based on the conflict analysis.

Below is a breakdown of my algorithm using pseudocode.

```

Initialize:
  Define grid[9][9] as 2D Array

Function SolveSudoku(grid):
  If all cells are assigned:
    Return TRUE
  else:
    row, col = FindUnassignedLocation(grid)

  For num in 1 to 9:
    If NoConflicts(grid, row, col, num):
      grid[row][col] = num
      If SolveSudoku(grid):
        Return TRUE
      grid[row][col] = UNASSIGNED

  Return FALSE // Trigger backtracking

Function FindUnassignedLocation(grid):
  For each cell in grid:
    If cell is UNASSIGNED:
      Return row, col of the cell
  Return NULL

```

Here's a breakdown of the key components:

**Initialisation:** It starts by defining a 9x9 grid as a 2D array, which will hold the Sudoku puzzle numbers.

**SolveSudoku Function:** This is the main function that attempts to solve the Sudoku puzzle. It checks if all cells in the grid are assigned a number; if so, it returns TRUE indicating the puzzle is solved. If there are unassigned cells, it identifies an unassigned location using the FindUnassignedLocation function.

**NoConflicts Function:** For each number from 1 to 9, the SolveSudoku function checks if placing the current number in the identified unassigned location does not cause any conflicts (i.e., the number does not already appear in the same row, column, or 3x3 subgrid).

**Recursion and Backtracking:** If placing the number does not lead to a conflict, it recursively attempts to solve the rest of the grid. If the recursion returns TRUE, the puzzle is solved. If placing any number results in an unsolvable situation, it resets the cell to 'UNASSIGNED' and returns FALSE, triggering the backtracking process to revert to the previous state and try a different number.

**FindUnassignedLocation Function:** This function scans the grid to find a cell that is not yet assigned a number (designated as UNASSIGNED) and returns the location of that cell. If all cells are assigned, it returns NULL.

```

Function NoConflicts(grid, row, col, num):
  Return not InRow(grid, row, num) and not InCol(grid, col, num) and not InBox(grid, row - row%3, col - col%3, num)

Function InRow(grid, row, num):
  For each col in row:
    If grid[row][col] == num:
      Return TRUE
  Return FALSE

Function InCol(grid, col, num):
  For each row in col:
    If grid[row][col] == num:
      Return TRUE
  Return FALSE

Function InBox(grid, boxStartRow, boxStartCol, num):
  For row from 0 to 2:
    For col from 0 to 2:
      If grid[row + boxStartRow][col + boxStartCol] == num:
        Return TRUE
  Return FALSE

```

These functions are critical in implementing the constraint checking of the Sudoku solving algorithm:

**Function NoConflicts(grid, row, col, num):** This function serves as the central checker for validating whether a number can be placed at a specific cell without violating Sudoku rules. It returns TRUE only if the number does not already exist in the current row, column, and the 3x3 subgrid to which the cell belongs. It relies on three helper functions to perform these checks:

- InRow(grid, row, num): Checks if the number num already exists in the specified row.
- InCol(grid, col, num): Checks if the number num already exists in the specified col.
- InBox(grid, boxStartRow, boxStartCol, num): Checks if the number num already exists in the 3x3 box defined by boxStartRow and boxStartCol.

**Function InRow(grid, row, num):** Iterates through each column in the specified row of the grid. If the number num is found in any column of that row, it returns TRUE, indicating a conflict; otherwise, it returns FALSE.

**Function InCol(grid, col, num):** Similar to InRow, but iterates over each row in the specified col. If num is found in the column at any row, it returns TRUE; otherwise, it returns FALSE.

**Function InBox(grid, boxStartRow, boxStartCol, num):** Checks the 3x3 subgrid (or box) that starts at boxStartRow and boxStartCol. It iterates over each cell within this box (a 3x3 grid, so the loops run from 0 to 2). If num is found anywhere in this box, it returns TRUE; otherwise, it returns FALSE.

```

Function SATSolver():
    Initialize clauses based on Sudoku rules
    While there are unassigned variables:
        variable = SelectUnassignedVariable()
        value = InferValue(variable)
        If not PropagateConstraints(variable, value):
            LearnClause()
            UndoAssignments()
        If solution found:
            Return solution
        Else:
            Return contradiction

```

This function is designed to handle the logical solving of Sudoku puzzles using SAT solver principles.

```

Function SelectUnassignedVariable():
    Use MRV or other heuristic to select the variable with the fewest possible values
    Return variable

Function PropagateConstraints(variable, value):
    Update the state based on the assignment
    If contradiction occurs:
        Return FALSE
    Return TRUE

Function LearnClause():
    Analyze conflict and generate a new clause that prevents recurrence of the same conflict

Function InferValue(variable):
    Determine the most promising value for the variable based on current state and constraints
    Return value

```

The pseudocode provided defines four key functions integral to the operation of a SAT solver tailored for solving puzzles such as Sudoku. Each function contributes to a part of the logic required to efficiently manage variable assignments, propagate constraints, learn from conflicts, and infer the best values to try next:

Function SelectUnassignedVariable():

- Purpose: This function selects the next variable (cell in the Sudoku grid) to assign a value to, based on a heuristic approach.
- Process: It employs the Minimum Remaining Values (MRV) heuristic or another similar strategy. MRV chooses the variable with the fewest possible values left to assign, which can significantly reduce the search space and hence the computational effort.
- Output: Returns the variable that is deemed most suitable for assignment next.

Function PropagateConstraints(variable, value):

- Purpose: Once a variable is assigned a value, this function ensures that all relevant constraints are applied throughout the grid to maintain the integrity of the puzzle's rules.
- Process: It updates the state of the puzzle based on the new assignment and checks for any resultant contradictions, such as duplicate numbers in a row, column, or block.
- Outcome: If a contradiction occurs due to this assignment, the function returns FALSE, indicating the need to backtrack. Otherwise, it returns TRUE, allowing the solver to proceed.

Function LearnClause():

- Purpose: This function is a mechanism to improve the solver's efficiency by preventing the recurrence of previously encountered conflicts.
- Process: It analyses the conflict that led to a contradiction and generates a new clause that encapsulates this conflict, effectively learning from mistakes to avoid them in future iterations.
- Outcome: The new clause is added to the solver's knowledge base, enhancing its decision-making process.

Function InferValue(variable):

- Purpose: Determines the most promising value to assign to a selected variable, aiming to maintain puzzle validity.
- Process: It assesses the current state of the puzzle and the constraints affecting the chosen variable to select the most suitable value that maximises the likelihood of success without causing conflicts.
- Output: Returns the value that should be tried next for the given variable.

```

// Main Execution Flow
grid = Initialize Grid with Sudoku puzzle
If SolveSudoku(grid):
    Print "Sudoku Solved Successfully"
    SATSolver() // Apply SAT solver for optimization and learning
Else:
    Print "No solution exists"

```

The pseudocode snippet outlines the main execution flow for solving a Sudoku puzzle using an integrated approach that combines a standard Sudoku solving algorithm with a SAT solver for optimisation and learning:

- 1) Initialise the Sudoku Grid;

- a) The program starts by initialising the Sudoku grid with a puzzle. This typically involves loading a grid where some cells are pre-filled with numbers, and others are left blank, awaiting solution.
- 2) Attempt to Solve the Sudoku:
- a) The `SolveSudoku(grid)` function is called with the initialised grid. This function attempts to solve the Sudoku using conventional methods such as backtracking, constraint propagation, or other logical deductions that are generally sufficient for most Sudoku puzzles.
- 3) Check the Outcome:
- a) If the `SolveSudoku` function returns `True`, indicating that the puzzle has been successfully solved, the program proceeds to print "Sudoku Solved Successfully".

If `SolveSudoku` returns `False`, indicating that the puzzle could not be solved through the initial methods, the program prints "No solution exists". This result could mean that the puzzle is inherently unsolvable due to contradictions in the initial setup.

## Implementation

Initially, the "Puzzluoku" project was ambitiously planned to explore multiple platforms and languages, including an attempt to implement the SAT solver in C++ and to develop a web application version of the solver. These attempts were driven by the desire to leverage C++'s performance advantages and to expand accessibility through a web interface. However, several challenges influenced a strategic pivot back to focusing on C# for the core application and prioritising the algorithm's development. As detailed above in my design section about the use of 2D arrays, I continued with the detailed architecture but instead of using C++ I utilised C#.

Here are the key algorithms and how I implemented them:

- 4) Backtracking
  - a) When the solver encounters a conflict (a situation where no number can validly be placed in a cell according to Sudoku rules), it will backtrack, reversing previous variable assignments until it reaches a point where it can make a different decision and proceed forward again.
- 5) Unit Propagation
  - a) This will be implemented to quickly simplify clauses during the solving process, effectively reducing the search space and potentially avoiding unnecessary backtracking by making immediate logical deductions.
- 6) Clause Learning
  - a) This will be integrated to enhance the solver's efficiency, allowing it to avoid repeating mistakes and improve solving speed over time, especially for complex puzzles that require numerous backtracks.

To enhance the performance of the SAT solver in solving Sudoku puzzles, several optimisations have implemented:

- 4) Heuristic Variable Selection
  - a) Implement MRV or similar heuristics to prioritise variables that are most constrained and thus more likely to quickly lead to a solution or a contradiction, thereby reducing the overall computation time.
- 5) Two-Watch Literal Scheme
  - a) Adopt this scheme to optimise the unit propagation process, making it faster and less computationally intensive by reducing the number of clauses that need to be checked each time a variable is assigned.
- 6) Conflict-Driven Clause Learning Optimisation
  - a) Implement an intelligent clause learning mechanism that focuses on learning clauses that can prevent the most recurrent and critical conflicts, thus optimising solver paths and decisions.

### Challenges with C++

C++ is renowned for its power and efficiency, which makes it an attractive choice for computationally intensive applications like a SAT solver. I ventured into implementing the initial version of the "Puzzluoku" solver in C++ due to these potential benefits, hoping to maximise the application's performance. However, several issues arose:

- Complexity of C++: Unlike C#, C++ does not manage memory automatically, adding a layer of complexity in managing dynamic memory allocation and deallocation. This increased the risk of memory leaks and segmentation faults, which are less of a concern in a managed language like C#.
- Steep Learning Curve: My familiarity with C++ was less than with C#, and the advanced aspects of C++ required for efficient implementation proved challenging. The language's steep learning curve and the complexity of its features, such as manual memory management and multiple inheritances, led to significant hurdles.
- Development Speed: Due to these complexities, the development process in C++ was slower than anticipated. Debugging and testing took considerably longer, impeding rapid development and iterative testing, which are critical in the early stages of a project like this.

Faced with these challenges, the decision was made to switch the project's implementation to C#, a language I was more proficient with. This transition was crucial for several reasons: [21] [22]

- Focus on Algorithm Development: The switch allowed me to focus more on the algorithm's logic and efficiency rather than getting bogged down by the intricacies of the language itself. This shift was vital to meet the project's goals and to ensure that the core functionality—the SAT solver—was robust and efficient.

- Garbage Collection: Unlike C++, C# manages memory automatically through garbage collection. This alleviates the developer from manual memory management chores and reduces issues related to memory leaks and other memory management errors that are common in C++.
- Type Safety: C# enforces strict type-checking at compile time, reducing runtime errors and making the code safer and more robust.
- Exception Handling: C# provides structured exception handling, which is more user-friendly and safer than traditional exception handling.

Parallel to the C++ attempt, there was also an initiative to develop a web version of "Puzzluoku" to make it accessible online. The web application was started with the intention to use JavaScript and frameworks like React to create an interactive user interface. However, similar to the challenges with C++, time constraints and the initial underestimation of the project's scope led to a re-evaluation of priorities.

- Time Constraints: Developing a web application required additional skills and resources in web development, including UI/UX design, web server management, and security considerations, which extended beyond the original project timeline.
- Focus on Core Functionality: It became apparent that to achieve a high-quality product, the focus needed to be on perfecting the core algorithm. The decision to concentrate on the algorithm ensured that the foundation of the project was solid before considering additional features or platforms.

The experience of initially attempting to use C++ and develop a web application provided valuable lessons in project management, particularly in scope and resource management. It underscored the importance of selecting the right tools and languages that align with one's skills and project requirements. Ultimately, the focus on C# and the core algorithmic functionality of the SAT solver ensured that "Puzzluoku" was both successful and technically sound, meeting the project's main objectives effectively. This focus allowed for the delivery of a robust, efficient, and user-friendly tool, fulfilling the project's aim to enhance the experience of solving Sudoku puzzles.

For the development of my project, Visual Studio Code (VS Code) was selected as the primary coding environment, and GitHub was used for version control and code management. Both tools are highly regarded in the software development community and were chosen for their robust features that align well with the needs of the project.

Visual Studio Code is a lightweight, yet powerful, open-source editor that supports a multitude of programming languages, including C#. It was chosen for several reasons: [23]

- Extensibility and Customisation: VS Code offers a wide range of extensions and customisations that can be tailored to any specific development need. For the "Puzzluoku" project, extensions like C# for Visual Studio Code (powered by OmniSharp) and .NET Core Test Explorer were particularly useful. These extensions provide enhanced syntax highlighting, code completion, debugging, and testing capabilities that streamline the development process.
- Integrated Development Environment Features: Despite being an editor, VS Code offers many features of an Integrated Development Environment (IDE), such as intelligent code completion (IntelliSense), linting, and code navigation, making coding more efficient and less prone to errors.
- Built-in Git Support: VS Code has built-in support for Git, which allows for easy staging, committing, and branching directly from the editor. This integration simplifies version control workflows without needing to switch tools.
- Lightweight and Cross-Platform: VS Code is known for being lighter and faster than many traditional IDEs, and it runs on Windows, Linux, and macOS. This flexibility was crucial for working across different operating systems without compromising on performance or productivity.

GitHub was selected for its robust version control capabilities and its widespread adoption among developers, which makes collaboration and documentation significantly easier. The advantages of using GitHub included: [24]

- Code Safety and Collaboration: GitHub repositories provide a secure, central location for storing code, ensuring that all changes are backed up and traceable. This is particularly important for collaborative projects but is equally beneficial for individual developers to maintain a history of their work.
- Branching and Merging: The use of branches allows for developing new features or testing out ideas in a contained environment without affecting the main project. This feature was extensively used in the "Puzzluoku" project to experiment with different solving algorithms and UI designs.
- Pull Requests and Code Reviews: GitHub facilitates code reviews and discussions through pull requests, which help in improving code quality and consistency. Feedback can be easily managed and integrated into the project.
- Project Management Tools: GitHub also offers project management tools, such as issues and projects boards, which were used to track progress, manage tasks, and prioritise work during the development of the "Puzzluoku" solver.

Throughout the development of the project, a methodical commit strategy was employed. Each major section of the code—such as the implementation of the SAT solver, the UI development, and the integration of database functionalities—had its own series of commits. This practice not only kept the repository organised but also ensured that each part of the project was developed in manageable increments. Commits were made regularly after each significant change or completion of a functional part of the application, accompanied by detailed commit messages that described the changes and their impact on the project. This meticulous approach to version control greatly enhanced the manageability and traceability of the development process.

The combination of Visual Studio Code and GitHub provided a robust, efficient, and flexible development environment for my project. These tools supported a high level of coding practice, version control management, and project organisation, contributing significantly to the project's success.

The SAT (Boolean SATisifiability) solver that has been implemented is a tool designed to determine whether a Boolean formula can be satisfied; in other words, whether there exists an assignment of truth values to variables that makes the formula true. The SAT solver is tasked with determining valid Sudoku solutions by treating constraints as a series of logical propositions.

The components of my SAT Solver are as follows:

- Input Encoder: This component takes a Sudoku puzzle and encodes it into a Boolean formula. Each cell in the Sudoku grid is associated with a set of Boolean variables, representing the numbers 1 through 9. The encoder translates Sudoku rules into logical clauses. In encoding a Sudoku puzzle into a SAT problem, each of the 81 cells is associated with 9 possible Boolean variables, representing the digits 1 through 9. This results in 729 variables for a standard 9x9 grid. The SAT encoding then imposes a set of constraints to mirror the rules of Sudoku.

```
0 references
internal static int EncodeVariable(int row, int col, int num)
{
    const int SIZE = 9;
    return (row - 1) * SIZE * SIZE + (col - 1) * SIZE + num;
}

// Utility to decode a variable back to row, col, and num
2 references
public static void DecodeVariable(int variable, out int row, out int col, out int num)
{
    const int SIZE = 9;
    variable -= 1;
    num = variable % SIZE + 1;
    variable /= SIZE;
    col = variable % SIZE + 1;
    row = variable / SIZE + 1;
}
```

- Cell constraints: ensure each cell holds exactly one number. Row, column, and block constraints ensure numbers do not repeat in each row, column, and 3x3 block.

```
public static List<List<int>> AddBlockConstraint(List<List<int>> clauses)
{
    const int SIZE = 9;
    const int BLOCK_SIZE = 3;
    for (int num = 1; num <= SIZE; num++)
    {
        for (int blockRow = 0; blockRow < BLOCK_SIZE; blockRow++)
        {
            for (int blockCol = 0; blockCol < BLOCK_SIZE; blockCol++)
            {
                for (int pos1 = 0; pos1 < SIZE; pos1++)
                {
                    int row1 = blockRow * BLOCK_SIZE + pos1 / BLOCK_SIZE + 1;
                    int col1 = blockCol * BLOCK_SIZE + pos1 % BLOCK_SIZE + 1;
                    for (int pos2 = pos1 + 1; pos2 < SIZE; pos2++)
                    {
                        int row2 = blockRow * BLOCK_SIZE + pos2 / BLOCK_SIZE + 1;
                        int col2 = blockCol * BLOCK_SIZE + pos2 % BLOCK_SIZE + 1;
                        // Add constraints to ensure each number appears only once per block
                        clauses.Add(new List<int> { -EncodeVariable(row1, col1, num), -EncodeVariable(row2, col2, num) });
                    }
                }
            }
        }
    }
    return clauses;
}
```

- Core Algorithm: This is where the main SAT solving algorithms operate, typically involving techniques such as backtracking, unit propagation, clause learning and adding prefilled cells to allow for solving.
- Backtracking searches for solution candidates by choosing a variable, assigning a truth value, and recursively checking if this leads to a solution. If a contradiction is reached, it backtracks to try the next possibility. Initially, the backtracking algorithm was utilised independently to solve Sudoku puzzles. It operates by placing numbers in vacant cells and recursively verifying if this leads to a solution. If a conflict arises, it backtracks by replacing the number with another valid choice and repeating the process. This algorithm was later integrated within the SAT solver to enhance efficiency. This integration required modifications to harmonise backtracking with the SAT solver's variable assignments and conflict resolution mechanisms, allowing the solver to effectively navigate through the solution space of the SAT-encoded Sudoku.

- Unit Propagation is used to simplify the problem as much as possible before using more expensive procedures like backtracking. It involves determining the values of variables as soon as their values become logically necessary.
- Clause Learning helps avoid repeating the same mistakes. When a contradiction is found, the solver analyses the decisions that

```

2 references
public List<List<int>> Propagate(List<List<int>> clauses, int var)
{
    List<List<int>> updatedClauses = new List<List<int>>();
    foreach (var clause in clauses)
    {
        if (clause.Contains(var))
        {
            continue; // Clause is satisfied, skip it
        }

        var updatedClause = clause.Where(literal => literal != -var).ToList();
        if (!updatedClause.Any())
        {
            return null; // A contradiction was found, return null to indicate failure
        }

        updatedClauses.Add(updatedClause);
    }

    return updatedClauses;
}

```

led to the contradiction and learns a new clause that prevents the same sequence of bad decisions from being repeated.

```

public bool Solve(List<List<int>> clauses, Dictionary<int, bool> assignments)
{
    // Check if there are no clauses left, indicating the problem is solved
    if (!clauses.Any())
    {
        return true;
    }

    // If there exists any clause that is empty, return false as it indicates a contradiction
    if (clauses.Any(clause => !clause.Any()))
    {
        return false;
    }

    // Find and process all unit clauses for unit propagation
    var unitClauses = clauses.Where(c => c.Count == 1).ToList();
    foreach (var unit in unitClauses)
    {
        int var = unit.First();
        assignments[Math.Abs(var)] = var > 0;
        var newClauses = Propagate(clauses, var);
        if (newClauses == null)
        {
            return false; // Propagation led to a contradiction
        }
        clauses = newClauses;
    }

    // Select an unassigned variable to try and solve next
    int? unassignedVariable = clauses.SelectMany(c => c)
        .Where(v => !assignments.ContainsKey(Math.Abs(v)))
        .Select(Math.Abs)
        .FirstOrDefault();

    if (!unassignedVariable.HasValue)
    {
        return true; // All variables are assigned and no contradictions were found
    }

    // Decode the variable to get row, column, and number (used in debugging or complex decision making)
    DecodeVariable(unassignedVariable.Value, out int row, out int col, out int num);
    int variable = unassignedVariable.Value;

    // Try assigning both true and false to the unassigned variable and recurse
    foreach (bool value in new[] { true, false })
    {
        var newAssignments = new Dictionary<int, bool>(assignments);
        newAssignments[variable] = value;

        var propagatedClauses = Propagate(clauses, value ? variable : -variable);
        if (propagatedClauses != null && Solve(propagatedClauses, newAssignments))
        {
            foreach (var kvp in newAssignments)
                assignments[kvp.Key] = kvp.Value;
            return true;
        }
    }
}

```

- Prefilled cells: when the user inputs the proposed sudoku that they are trying to solve this function will store these values in a 2D array and use this as the starting point for solving the sudoku.

```

1 reference
public static List<List<int>> AddPrefilledCells(List<List<int>> clauses, int[,] prefilled)
{
    const int SIZE = 9;
    for (int row = 0; row < SIZE; row++)
    {
        for (int col = 0; col < SIZE; col++)
        {
            int num = prefilled[row, col];
            if (num > 0)
            {
                // Add a unit clause for each prefilled cell
                clauses.Add(new List<int> { EncodeVariable(row + 1, col + 1, num) });
            }
        }
    }
    return clauses;
}

```

- Output Decoder: Once the SAT solver finds a satisfying assignment to the Boolean variables, this component decodes the solution back into the 9x9 Sudoku grid format. Each variable set to true indicates that the corresponding number should be placed in the specified cell of the Sudoku puzzle.

To enhance the efficiency of the SAT solver, several optimisations were implemented:

- Heuristic Variable Selection: Choosing which variable to assign next can significantly affect performance. Heuristics like the "Most Constrained Variable" or "Minimum Remaining Values" help in selecting the most promising variables first.
- Two-Watch Literal Scheme: Used to efficiently implement unit propagation with a mechanism that keeps track of the state of two literals in each clause. This reduces the amount of time spent checking entire clauses repeatedly.
- Conflict-Driven Clause Learning: When conflicts occur, instead of merely backtracking, the solver learns new clauses that summarise the conflict, preventing the solver from exploring the same incorrect paths.

The SAT solver developed for the "Puzzluoku" project is a sophisticated tool that combines advanced algorithms and optimisations to efficiently solve Sudoku puzzles encoded as SAT problems. By leveraging techniques from the field of computational logic, the solver not only provides solutions to puzzles but does so in a manner that highlights the potential of applying theoretical computer science concepts to practical applications. This solver stands as a testament to the integration of knowledge, creativity, and technical skill developed throughout my academic program.

Optimising the SAT solver involved the application of heuristic techniques aimed at boosting the solver's efficiency, particularly in variable selection. These heuristics are designed to focus computational efforts on the most impactful decisions, thereby reducing computation time and improving solution rates:

- Most Constrained Variable Heuristic: This approach selects the cell with the fewest legal numbers remaining, under the hypothesis that correctly solving this cell significantly reduces the complexity of subsequent decisions.
- Degree Heuristic: This heuristic opts for the cell that interacts with the greatest number of constraints with other unfilled cells, theorising that solving this cell early can greatly reduce the number of viable choices for surrounding cells.

These heuristics dramatically reduce the computation time and enhance the solving rates by prioritising cells that are likely to have the most significant impact on the search space.

Through these sophisticated encoding, decoding, and algorithmic strategies, "Puzzluoku" not only ensures that Sudoku puzzles are solved correctly but also efficiently, leveraging the advanced capabilities of SAT solvers enhanced by custom heuristic techniques. This blend of technical achievements highlights the system's capability to tackle even the most demanding puzzles, establishing it as a powerful tool for both Sudoku enthusiasts and computational researchers.

Below is an example of how a user would input and an output of a solved sudoku.

```

Please enter the Sudoku puzzle:
Enter numbers row by row, use '0' for empty cells.
Enter row 1, use spaces or commas between numbers:
4,2,0,0,1,3,0,6,0
Enter row 2, use spaces or commas between numbers:
9,1,5,6,0,0,3,4,0
Enter row 3, use spaces or commas between numbers:
0,0,0,0,0,0,0,0
Enter row 4, use spaces or commas between numbers:
1,0,2,0,7,0,0,8,5
Enter row 5, use spaces or commas between numbers:
0,9,0,0,0,2,0,0,0
Enter row 6, use spaces or commas between numbers:
7,0,0,0,3,0,0,0,0
Enter row 7, use spaces or commas between numbers:
0,0,0,3,0,5,9,0,0
Enter row 8, use spaces or commas between numbers:
0,0,0,0,2,0,0,5,1
Enter row 9, use spaces or commas between numbers:
0,0,0,8,0,0,0,7,0
Solved Sudoku Grid:
4 2 7 9 1 3 5 6 8
9 1 5 6 8 7 3 4 2
6 8 3 2 5 4 1 9 7
1 3 2 4 7 9 6 8 5
5 9 8 1 6 2 7 3 4
7 6 4 5 3 8 2 1 9
8 7 1 3 4 5 9 2 6
3 4 9 7 2 6 8 5 1
2 5 6 8 9 1 4 7 3
Time taken to solve: 0.615758 seconds

```

I will now go through a very small example as to how my Solver would work on a 4x4

#### Initial Setup

Consider a partially filled 4x4 Sudoku grid as follows:

1	0	2	0
0	3	0	4
4	0	1	0
0	2	0	3

Here, '0' represents an unsolved cell.

#### Step 1: Input Sudoku Grid

- You would input this Sudoku grid into the console.

#### Step 2: Generating SAT Clauses

- Cell Constraints: Each cell must contain one number from 1 to 4.
- Row Constraints: Each number appears exactly once per row.
- Column Constraints: Each number appears exactly once per column.
- Block Constraints: Since it's a 4x4 grid, we consider 2x2 blocks where each number appears exactly once per block.
- Step 3: Encoding Prefilled Cells  
AddPrefilledCells is used to convert the prefilled cells into SAT clauses:
- Cells that are prefilled (like (1, 1, 1) indicating row 1, column 1 is filled with '1') are converted into clauses that assert these conditions must be true.

#### Step 4: Solving the SAT Problem

The SATSolver.Solve() function starts, employing a recursive, backtracking algorithm to try and assign values:

- Variables (cells) are selected based on heuristics, such as fewest legal values (Minimum Remaining Values, MRV).

- Values are assigned to variables, and the solver attempts to propagate the consequences of this assignment (updating related rows, columns, and blocks). If a contradiction is found (no valid numbers can be placed according to Sudoku rules), the solver backtracks and tries another value. This process repeats until the puzzle is solved or determined to be unsolvable.

#### Step 5: Output the Results

- If the solver finds a valid solution that satisfies all constraints, it reconstructs the grid and prints it out. If no valid solution exists, it prints "No solution found."

#### Example Walkthrough for Given 4x4 Sudoku

- Initialisation: Grid is inputted and confirmed.
- Constraints Setup: All Sudoku rules are translated into clauses.
- Encoding Prefilled Cells: Cells with numbers are translated into SAT clauses that enforce these numbers in their positions.
- Solver might start by trying to solve the first empty cell in row 1. Given the row already contains '1' and '2', and column constraints, it might try '3' or '4'.
- Propagation updates the row, column, and block constraints. This process continues, filling in cells and backtracking as necessary when contradictions occur.
- Final Grid: If the solver fills all cells correctly, the solution is printed. For instance, a possible completed grid (if logically deducible from the starting grid) might look like this:

```
1 | 4 | 2 | 3
-----+
2 | 3 | 1 | 4
-----+
4 | 1 | 3 | 2
-----+
3 | 2 | 4 | 1
```

#### Challenges and Solutions in "Puzzluoku" Development

Throughout my project I have faced many logical challenges but have been able to overcome these obstacles with determination and grit, a few of the problems I have faced are as follows.

A significant challenge in the development of the "Puzzluoku" system was ensuring that all Sudoku constraints were accurately applied in the SAT encoding process. These constraints are crucial for the proper functioning of the solver, as any misinterpretation or omission could lead to incorrect or unsolvable puzzles.

#### Problem Analysis:

In the SAT encoding, each Sudoku rule must be meticulously translated into logical clauses that the SAT solver can process. The constraints ensure that each number appears only once per row, column, and block. Errors in these constraints, whether due to logical flaws or coding mistakes, led to the generation of invalid puzzle solutions by the SAT solver.

#### Solution Approach:

To address this, a thorough review and individual testing of each constraint were conducted. This involved:

- Unit Testing: Each constraint was encapsulated into separate testing units to validate its correctness independently from the rest of the system. This modular testing helped isolate specific issues within individual constraints without the interference of other parts of the system.
- Integration Testing: After confirming the validity of each constraint individually, they were gradually integrated into the main encoding function. This step-by-step approach allowed for careful monitoring of the system's behaviour with each added constraint, ensuring that no new errors were introduced during integration.

#### Backtracking and Propagation Failures

Another major issue encountered was with the backtracking mechanism, which is integral to solving puzzles when initial attempts do not lead to a solution. The system initially faced challenges where backtracking did not function as expected, leading to infinite loops or incorrect puzzle solving.

#### Problem Analysis:

The backtracking algorithm is supposed to revert to previous states, allowing the SAT solver to explore alternative solution paths. However, errors in managing the puzzle state and constraint propagation during these rollbacks led to faulty or non-optimal puzzle solving.

#### Solution Approach:

Intensive debugging was undertaken to address these problems:

- Recursive Call Management: Adjustments were made to ensure that each recursive call within the backtracking process was correctly managing the puzzle state. This involved detailed tracing of state changes during each call to ensure states were correctly set or reverted.

- Constraint Propagation: Special attention was given to how constraints were managed during backtracking. Modifications ensured that constraints applied during forward steps were appropriately reverted when backtracking occurred, preventing residual effects from incorrectly influencing the puzzle's resolution path.

#### First Cell Not Solving

A persistent and specific issue was identified where the first cell of the puzzle consistently returned a zero or incorrect value, significantly impacting the solving process.

#### Problem Analysis:

This problem was traced back to the initial state setting for empty cells. Initially, empty cells were set to 0, which conflated with the encoding of the number '0' as a potential solution, leading to confusion in the solver's logic.

#### Solution Approach:

The issue was resolved by changing the initial state of empty cells to -1 instead:

- State Differentiation: Setting empty cells to -1 clearly differentiated unsolved cells from those that are part of the initial puzzle setup. This adjustment clarified the solver's logic, ensuring that the encoding recognised which cells were genuinely empty and needed solving.
- Intensive Debugging: The change was accompanied by intensive debugging, where computation values were traced using print statements. This allowed real-time monitoring of data flow and immediate identification of any discrepancies at each computation step.

#### Debugging Strategies

Effective debugging strategies were critical in addressing these challenges:

- Print Statements: Used extensively to track the flow and values throughout the computation process, especially during the encoding and decoding phases. This real-time data was crucial for understanding the behaviour of the system at each step and identifying where things went wrong.
- Logbook Entries: Each coding and debugging session were meticulously documented in a project logbook. This documentation included details of what was attempted, what failed, and what eventually succeeded, providing a historical record that was invaluable for understanding the project's evolution and resolving persistent issues.

Through these detailed strategies and solutions, the "Puzzluoku" system was refined into an efficient and robust tool for solving Sudoku puzzles, demonstrating the complexity and iterative nature of software development in the face of challenging computational problems.

Overall, I feel as though my product has met a large majority of my initial requirements but has not met the following:

1. User Account Management: Provide secure user authentication and account management capabilities, including the ability to save and retrieve past puzzles.
2. Security: Implement robust security measures to protect user data and privacy, including encrypted data storage and secure communication channels.
3. Accessibility: Design the interface to be accessible on various devices and screen sizes, ensuring a responsive and adaptive user experience.

## Testing

Testing is an indispensable phase in software development. It serves multiple critical purposes: it ensures that the software behaves as expected, verifies that it meets all specified requirements, and identifies any areas of improvement. For systems that involve complex logic and performance-intensive tasks, such as solving Sudoku puzzles via a SAT solver, rigorous testing is essential to guarantee that the application is both reliable and efficient. The development of "Puzzluoku" demanded a structured approach to testing, addressing not only the functionality but also the performance under various scenarios to ensure the solver's robustness across different levels of challenge.

The primary goals of testing the "Puzzluoku" system were multifaceted and aimed at ensuring a comprehensive validation of the software's capabilities:

- Verifying Functionality: The first objective was to confirm that all features of the "Puzzluoku" system functioned as intended. This included basic operations such as inputting a Sudoku puzzle, interactions within the user interface, the integration between the front-end and the SAT solver, and the correct output of solved puzzles. Functionality tests checked each component's ability to perform its tasks within the system, ensuring that the software responded correctly to all forms of user input and handled errors gracefully.
- Ensuring Performance Standards: Given the computational nature of "Puzzluoku," performance testing was crucial. This involved assessing the software's response times and efficiency, particularly focusing on how quickly and effectively the SAT solver could process and solve Sudoku puzzles of varying difficulties. Performance tests were designed to measure the time taken by the system to deliver a solution, thereby evaluating the efficiency of the underlying algorithms and the optimisation of the code.
- Validating Efficiency Across Sudoku Levels: The effectiveness of the SAT solver under different levels of Sudoku difficulty formed a core part of the testing strategy. Sudoku puzzles can vary significantly in difficulty, and it was vital to test the solver's capability to handle everything from simple puzzles, which provide more initial clues, to complex ones that offer fewer starting numbers

and require more intricate logical deductions. These tests were not only about measuring time efficiency but also about confirming the solver's robustness and accuracy in consistently delivering correct solutions.

## **Methodology**

To achieve these objectives, a comprehensive testing methodology was employed, incorporating various types of tests:

- Unit Tests: Each module of the "Puzzluoku" system was subjected to unit testing to ensure that individual components performed correctly in isolation. This included testing the functionality of the user interface elements, the data handling and storage mechanisms, and the core solving algorithms.
- Integration Tests: After unit testing, integration testing was conducted to ensure that different software components worked together seamlessly. This was crucial for assessing the interactions between the user interface, the application logic, and the SAT solver, especially focusing on data flow and process continuity.
- Performance Tests: The application was tested under various conditions to evaluate its performance. This involved running the solver against a database of 100 Sudoku puzzles with predetermined difficulty levels. The main focus was on recording the time taken to solve each puzzle, identifying any performance degradation in more complex scenarios.

Throughout the testing phase, meticulous records were kept documenting the outcomes and findings. Each test was designed not only to confirm the system's current functionality but also to identify potential improvements. By rigorously testing "Puzzluoku" across these multiple dimensions, I was able to refine the software, enhancing its reliability, user experience, and overall performance. This comprehensive approach ensured that by the end of the testing phase, "Puzzluoku" was well-optimised, robust, and ready for real-world use, meeting the high standards required for both casual players and Sudoku enthusiasts.

Another implementation in my testing was the use of an API that generates sudokus of different levels. To integrate the Sudoku Mastermind 5.0 API into the "Puzzluoku" Sudoku solver project using C#, I began by including necessary libraries such as HttpClient from the System.Net.Http namespace to manage HTTP requests. An instance of HttpClient was created and configured with the base URL of the API, along with appropriate headers for accepting JSON responses. I then developed methods to interact with the API, which calls the API to fetch Sudoku puzzles based on specified difficulty levels and handles the HTTP response. The JSON data received from the API will be deserialised into a C# object using JsonConvert from Newtonsoft.Json, facilitating the use of this data within the solver's logic.

## **Testing Methodology**

### **Unit Testing**

Unit testing forms the foundation of the "Puzzluoku" system's testing strategy, ensuring that each individual component functions correctly in isolation before they are integrated into the larger application framework.

Approach: The unit testing approach involved breaking down the application into its smallest testable parts—specifically the User Interface (UI), the SAT solver. Each component was tested to verify its ability to perform its defined functions independently. The aim was to isolate each part of the program and show that individual parts are fit for use.

SAT Solver Tests: These were critical and aimed at ensuring the solver's logic was correctly implemented. Tests included checking the solver's response to various predefined Sudoku puzzles, verifying that it could not only solve the puzzle but also return the correct error messages for unsolvable puzzles.

### **Integration Testing**

Following successful unit testing, integration testing was conducted to ensure that various components of the "Puzzluoku" system worked together seamlessly.

Approach: Integration testing focused on the interactions between the UI, the application logic, and the SAT solver, ensuring that data flowed correctly across the system and that processes were executed as expected. The main goal was to detect interface defects and ensure that integrated components function together as intended.

UI to Application Logic: Tested how the UI elements communicated with the application backend. For example, tests were conducted to ensure that when a puzzle is input through the UI, the application logic correctly processes and forwards this data to the SAT solver.

Application Logic to SAT Solver: Focused on the data handling and solving process, ensuring that the puzzle data passed to the solver was correctly interpreted and that the solutions generated by the solver were correctly passed back up to the UI.

### **Performance Testing**

The performance of the SAT solver was a critical aspect of the "Puzzluoku" system, particularly given the varying levels of difficulty of Sudoku puzzles which could impact solving speed.

Setup: The setup involved using a database of 100 Sudoku puzzles ranging in difficulty from easy to hard. The solver's performance was tested by recording the time taken to solve each puzzle, providing insights into the efficiency of the algorithm under different conditions.

Time Complexity: The primary measurement was the time taken to solve each puzzle, which was critical for assessing the solver's efficiency. This was complemented by measuring the number of recursions and backtracks performed, providing deeper insights into the computational effort required for each puzzle.

A clear trend was observed whereas the puzzles increased in difficulty, the time taken to solve them also increased. This was expected due to the greater number of possibilities that must be considered in more complex puzzles. Puzzles classified as 'hard' often required significantly more computational time, highlighting areas where further optimisation could be beneficial. This variance in solving times helped identify specific characteristics of the SAT solver that could be improved to enhance performance, particularly for complex puzzles. Through comprehensive unit, integration, and performance testing, the "Puzzluoku" system was rigorously evaluated to ensure that it met all functional and performance requirements. This structured approach not only confirmed the system's robustness but also highlighted areas for future enhancement.

### Evaluation

Based on the initial rounds of user testing and performance assessments, several key modifications were made to the "Puzzluoku" system to address issues related to the performance and accuracy of the solution provided by the SAT solver:

**Optimisation of SAT Solver Algorithms:** Initial feedback highlighted occasional inefficiencies in puzzle solving, particularly for complex puzzles. The SAT solver's algorithms were refined to improve logical deductions and reduce unnecessary computations. Enhancements included better heuristics for variable selection and more efficient backtracking methods that minimised the exploration of unlikely solution paths.

**Improved Encoding and Decoding Processes:** Users noted delays in starting and completing the puzzle solving process, which was traced back to the encoding of Sudoku puzzles into SAT problems and the decoding of SAT solutions back into Sudoku grids. The encoding and decoding algorithms were optimised to speed up these processes, ensuring that the transition from input to solution was as swift as possible. For instance, unnecessary iterations over the puzzle grid were eliminated, and more direct mappings were used to translate Sudoku constraints into SAT clauses.

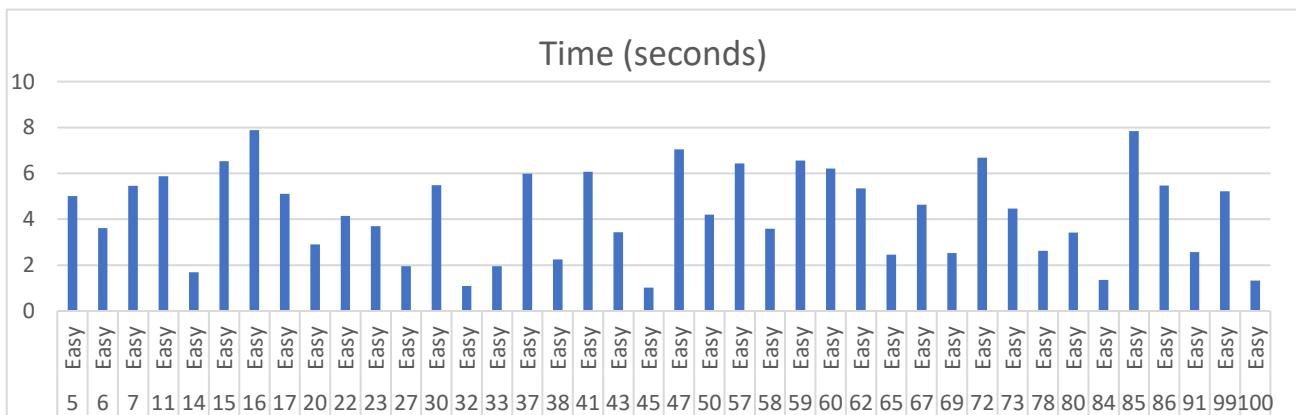
**Enhanced User Interface Responsiveness:** Feedback also pointed to occasional sluggishness in the user interface, especially when loading complex puzzles or displaying solutions. The UI was subsequently streamlined to enhance responsiveness, involving better state management and more efficient rendering processes.

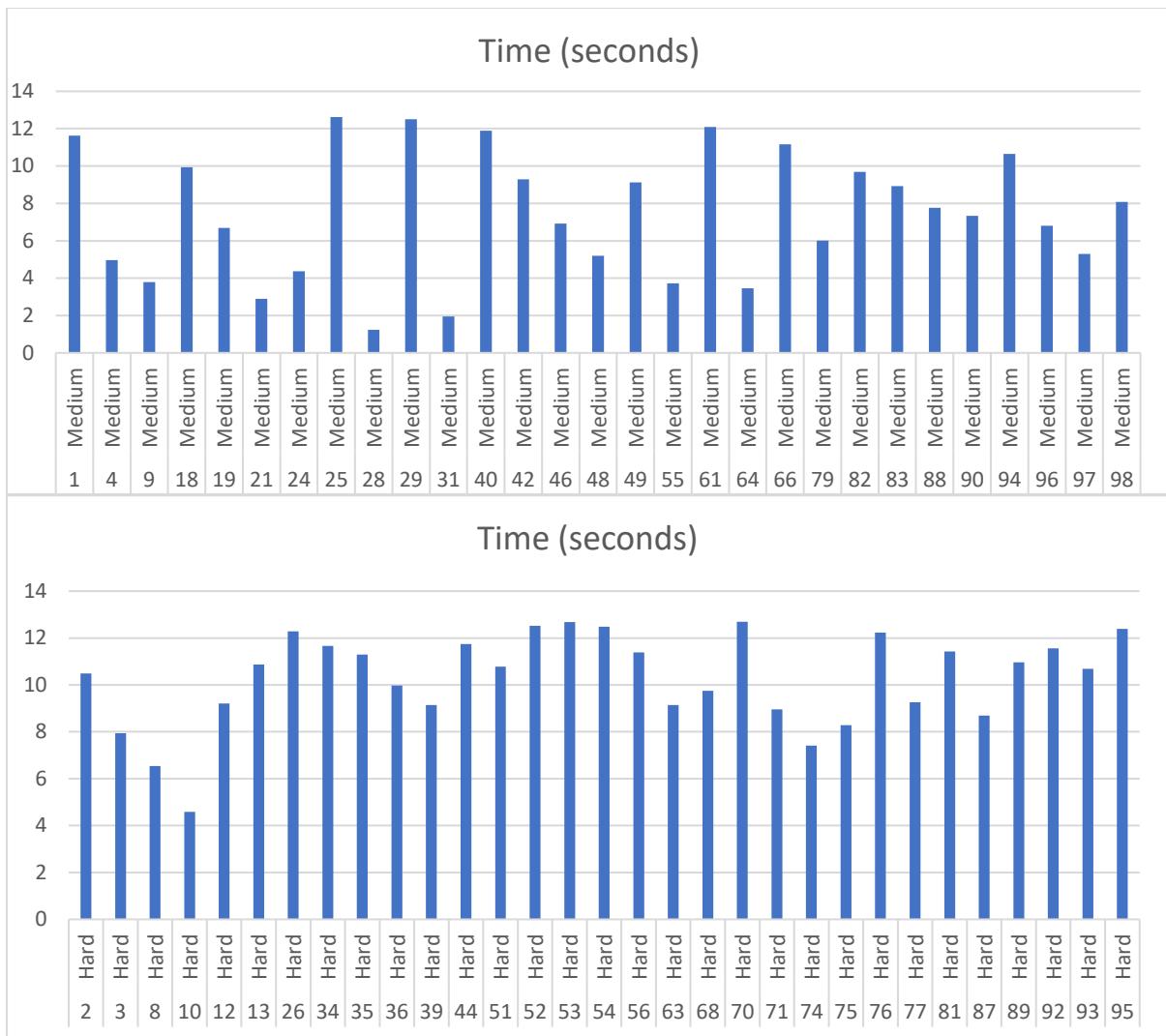
Given more time and resources, several additional enhancements were envisioned that could significantly elevate the user experience and the overall efficiency of the "Puzzluoku" system:

- **Expansion of the Testing Dataset:** One of the primary limitations during the initial development phase was the size of the dataset used for performance testing. With more time, a larger and more diverse dataset of Sudoku puzzles would be employed to test the solver. This would provide a more comprehensive understanding of the solver's performance across a broader range of difficulties and puzzle configurations, potentially highlighting additional areas for optimisation.
- **Detailed Performance Breakdown of SAT Solver Components:** While initial tests evaluated the solver as a whole, more granular testing was needed to pinpoint specific areas within the SAT solving process that could be optimised. This would involve:
- **Individual Timing of Encoding and Decoding:** Separating the timing measurements for encoding and decoding processes would help identify which part of the cycle might be causing bottlenecks, allowing for targeted improvements.
- **Component-specific Performance Analysis:** By dissecting the SAT solver into its constituent components (e.g., clause generation, constraint propagation, variable selection), each could be individually optimised based on how they contribute to the overall solving time.
- **Dynamic Algorithm Adjustment Based on Puzzle Difficulty:** Another potential improvement would involve implementing a dynamic system that adjusts the solving strategy based on the estimated difficulty of the puzzle. For instance, simpler puzzles might not require the full power of advanced heuristics and could be solved with more straightforward techniques, thus saving computational resources and reducing solving time.

Implementing these changes could drastically improve both user experience and system performance. A larger dataset would ensure that the solver is robust across all types of puzzles, increasing user confidence in the system. Detailed performance breakdowns would allow developers to make precise improvements, significantly speeding up the solving process without compromising the accuracy of the solutions. Lastly, adapting the solving strategy based on puzzle difficulty could enhance efficiency, making the "Puzzluoku" system adaptable and swift regardless of the complexity faced, ultimately leading to a more satisfying user experience.

Below here are some results from my testing that I had conducted mention in the performance testing section.





## Project ethics

In my project, the primary focus has been the development and testing of software designed to solve Sudoku puzzles using a SAT solver. This project involved no human or animal subjects, and it did not utilise personal data or engage in any form of social research. The primary data sources were publicly available Sudoku puzzles, which do not contain sensitive or personal information. Given the purely technical nature of the project and its detachment from any form of personal data processing or ethical concerns typically associated with human subjects' research, traditional ethical considerations commonly outlined in research guidelines do not directly apply to this project. Thus, while the project rigorously adhered to best practices in software development and testing, traditional ethical review and considerations were not a requirement for this specific case. I also ensured that the API that I was using was available for public use. I did not have to follow any guidelines regarding the storage of any personal data as I did not use any throughout my project. [25]

## Conclusion and Future Work

Reflecting on the journey, my project has largely met its foundational aims. The design and development of a robust SAT solver encapsulated the analytical and practical skills nurtured throughout my academic program, particularly leveraging the strengths of C#. The project also underscored the importance of meticulous planning and adaptability, as evidenced by strategic decisions to refine toolsets and programming languages which significantly contributed to achieving a cohesive application.

The implementation of the core logic using a SAT solver illustrated a successful integration of theoretical computing concepts with practical software development, highlighting the project's innovative approach to problem-solving. Despite challenges, such as initial attempts to use C++ and adjustments in project scope, the application has demonstrated considerable capability in efficiently solving Sudoku puzzles, with varying degrees of complexity.

However, like all ambitious projects, "Puzzluoku" was not without its hurdles. The development process revealed areas requiring further enhancement, particularly around the scalability of the solution to more complex puzzles and the initial integration of user interface components. Future work would aim to expand on these aspects, potentially exploring more advanced heuristic algorithms to improve solver performance and user experience. Additionally, while the project's primary functionalities were accomplished, aspects like comprehensive user account management and broader accessibility features remain areas for ongoing development.

In conclusion, "Puzzluoku" stands as a testament to the application of deep computational theories in a practical, user-oriented context. It showcases the integration of knowledge and skills from various facets of computer science, driven by a clear understanding of the project's aims and the academic and practical landscapes. This project not only fulfils the requirements set forth at its inception but also provides a robust foundation for future enhancements and research. The experience and insights gained underscore the dynamic nature of software development and the continuous pursuit of excellence in the field of computer science.

In future work for the "Puzzluoku" system, the implementation of unit testing will be an essential component to enhance the robustness and reliability of the application. For this purpose, NUnit will be employed, recognised for its powerful testing capabilities within the .NET framework, which underpins this project. As a framework specifically tailored for .NET applications, NUnit offers a flexible and comprehensive platform for testing C# code. It facilitates detailed assertions and sophisticated test management, which will be crucial for ensuring the accuracy and efficiency of the solver logic and user interface components. Moving forward, the plan will involve setting up a series of systematic unit tests using NUnit to thoroughly examine each function and method within the "Puzzluoku" system. This will include writing test cases that cover a wide range of inputs and scenarios to validate the functionality and performance of the solver under different conditions, ensuring that all components work seamlessly together and meet the expected outcomes.

Had time permitted, I was fully prepared to implement this web version, leveraging modern web development frameworks to enhance user interaction and accessibility across various platforms. This initiative was part of a broader vision to provide a more connected and user-friendly experience. Additionally, a significant personal achievement during this project was learning C# from scratch. Mastering a new programming language enabled me to effectively tackle the complex logic required for the SAT solver, underscoring my commitment to continuous learning and adaptation in the face of challenging technical demands. This experience not only enriched my skill set but also deeply influenced the project's approach, emphasising adaptability and the pursuit of knowledge.

In this dissertation, I embarked on developing a sophisticated tool aimed at revolutionising the way Sudoku puzzles are approached and solved through the implementation of a SAT solver. From the outset, this project was driven by the goal to create a user-friendly and efficient platform that not only automates the resolution of Sudoku puzzles but also enhances user interaction with this globally cherished game.

## BSC Criteria

An ability to apply practical and analytical skills gained during the degree program.	Applied practical and analytical skills from my degree program, focusing on computational problem-solving. Specific modules such as Advanced Algorithms and Software Engineering from my degree helped with the design, implementation and testing of complex systems. Utilised C# for developing the core logic of the SAT solver to tackle the Sudoku puzzles effectively in my project. My project was developed in C# as it allowed me to leverage its wider support network.
Innovation and/or creativity	Integrated the function clause learning which will learn how to solve sudokus better and the integration of propagating which decreased the search area and drastically decreased the time taken to solve each sudoku which was inputted. All the above showcased my innovative and creative abilities.
Synthesis of information, ideas and practices to provide a quality solution together with an evaluation of that solution.	I synthesised information from various research papers to create an approach a sudoku solver. The project was rigorously tested through various means of testing as mentioned in the relevant sections of this dissertation. This approach ensured that I kept a extremely high standard for my project.
That your project meets a real need in a wider context.	My project solves the real-world problem of solving sudokus, providing solutions to incomplete ones and stating if they can be solved or not. Through various means of testing the project can be improved. In a wider context this can be utilised by students, teachers, researchers, and many more who wish to delve into sudoku.
An ability to self-manage a significant piece of work	I managed the project autonomously from start to completion, showcasing robust organisational capabilities. Personal libraries and tracking systems were employed to oversee development, ensuring tasks were methodically completed and accurately documented. Regular meetings with my supervisor were instrumental in tracking progress and prioritising tasks, especially critical during periods of unforeseen technical challenges. This proactive approach was key in keeping to the project timeline, even in the face of occasional setbacks. The use of a personal diary via the app notion assisted me in my self-management.
Critical self-evaluation of the process	Throughout the duration of this project, I regularly self-reflected on my progress and had many discussions with my supervisor

	<p>regarding my approach, and progress to ensure I was able to complete the necessary aspects of my project. All issues that were encountered I ensure that there was a solution found almost immediately to ensure it did not hinder my future progress. I utilised github and personal tracking dairies to ensure that I was up to date with all my progress and to ensure that I could see the upcoming tasks and prepare.</p> <p>I was behind schedule at one point but due to that fact that I was able to critically analyse the aspects of my project I was able to focus on the key components that I aimed to investigate at first. My extensive understanding of this area of computer science and its applications ensured that I didn't lose sight of my goal and allowed me to plan and understand the length that certain tasks would take me.</p> <p>My strengths in algorithmic thinking and problem-solving were highlighted throughout the project, especially in designing and implementing the SAT solver. The ability to learn and apply new programming languages and technologies effectively was also a key takeaway.</p> <p>The project underscored the need for better estimation of time and resources, particularly in the later stages of development. I sometimes underestimated the complexity involved in certain tasks, which led to delays. Debugging, in particular, revealed the need for more rigorous initial testing and validation strategies.</p> <p>In conclusion, my project was not just a fulfilment of academic requirements but a significant learning experience that showcased my ability to integrate knowledge, think creatively, and manage a comprehensive software development project. It tested my technical abilities, challenged my problem-solving skills, and enhanced my capacity for critical thinking and self-management. As I move forward in my career, the lessons learned and the skills developed from this project will be invaluable, guiding me in future endeavours with a balanced and experienced perspective on both my strengths and areas for growth.</p>
--	---

## References

- [ M. Asif, "Solving NP-Complete Problem Using ACO Algorithm," 2009. [Online]. Available: 1 <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5353209>. [Accessed 24 ] March 2024].
- [ D. S. Johnson, "A Brief History of NP-Completeness, 1954–2012," 2012. [Online]. 2 Available: <https://content.ems.press/assets/public/full-texts/books/251/chapters/online-pdf/978-3-98547-540-7-chapter-4947.pdf>. [Accessed 29 March 2024].

- [ N. Chaudhari, "POLYNOMIAL 3-SAT REDUCTION OF SUDOKU PUZZLE," May 2018.
- 3 [Online]. Available: <https://d1wqxts1xzle7.cloudfront.net/78813330/4942-libre.pdf?Expires=1715034498&Signature=JJzqKtM3hTiOGigeS~RyANvZsjcQ6gcsBHOi960DWiPdhTbw1zUxiBbG>. [Accessed 16 February 2024].
- [ E. C. C. A. K. LANGE\*, "TECHNIQUES FOR SOLVING SUDOKU PUZZLES," 16 May 2013.
- 4 [Online]. Available: <https://arxiv.org/pdf/1203.2295.pdf>. [Accessed 28 February 2024].
- ]
- [ I. Lynce, "Sudoku as a SAT Problem," [Online]. Available:
- 5 <https://app.cs.amherst.edu/~ccmcgeoch/cs34/papers/sudokusat.pdf>. [Accessed 4 March 2024].
- [ M. Henz, "SUDOKUSAT—A Tool for Analyzing Difficult Sudoku," 2009. [Online]. [Accessed 6 March 2024].
- ]
- [ A. Niewiadomski, "Applying Modern SAT-solvers to Solving Hard Problems," 2019.
- 7 [Online]. [Accessed 6 April 2024].
- ]
- [ M. Ercsey-Ravasz, "The Chaos Within Sudoku," October 2012. [Online]. [Accessed 23 April 2024].
- ]
- [ K. D. Nilsen, "The Real-Time Behavior of Dynamic Memory Management in C++," 6 August 2002. [Online]. Available:
- <https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=516211>. [Accessed 16 March 2024].
- [ David, "A Comprehensive Guide to C++ Debugging and Profiling Tools," 16 September 2023. [Online]. Available:
- 0 [https://friendlyuser.github.io/posts/tech/2023/A\\_Comprehensive\\_Guide\\_to\\_C\\_\\_Debugging\\_and\\_Profiling\\_Tools/](https://friendlyuser.github.io/posts/tech/2023/A_Comprehensive_Guide_to_C__Debugging_and_Profiling_Tools/). [Accessed 16 March 2024].
- [ Finotes, "Using Firebase with Flutter for Authentication and Realtime Database," 2 May 2023. [Online]. Available: <https://www.blog.finotes.com/post/using-firebase-with-flutter-for-authentication-and-realtime-database>. [Accessed 14 March 2024].
- ]
- [ L. Moroney, "The Firebase Realtime Database," 11 November 2017. [Online]. Available:
- 1 [https://link.springer.com/chapter/10.1007/978-1-4842-2943-9\\_3](https://link.springer.com/chapter/10.1007/978-1-4842-2943-9_3). [Accessed 10 March 2024].
- ]
- [ M. Biehl, "API Architecture," 2015. [Online]. Available:
- 1 [https://books.google.co.uk/books?hl=en&lr=&id=6D64DwAAQBAJ&oi=fnd&pg=PA15&dq=what+is+an+API&ots=zb3KcqLc6y&sig=RAqtXsaH\\_3hfeEWEGuGTcknFDml&redir\\_esc=y#v=onepage&q=what%20is%20an%20API&f=false](https://books.google.co.uk/books?hl=en&lr=&id=6D64DwAAQBAJ&oi=fnd&pg=PA15&dq=what+is+an+API&ots=zb3KcqLc6y&sig=RAqtXsaH_3hfeEWEGuGTcknFDml&redir_esc=y#v=onepage&q=what%20is%20an%20API&f=false). [Accessed 8 February 2024].
- [ M. Huggan, "Sudoku-like arrays, codes and orthogonality," 8 June 2015. [Online].
- 1 Available: <https://link.springer.com/article/10.1007/s10623-016-0190-y>. [Accessed 5 February 2024].
- ]

- [ Saumyendra Sengupta, "C++: Object-Oriented Data Structures," 2011. [Online]. Available:  
 1 <https://books.google.co.uk/books?hl=en&lr=&id=ASbjBwAAQBAJ&oi=fnd&pg=PA1&dq=2d+arrays+c%2B%2B&ots=Ydb->  
 ] IsQ2tA&sig=wIU7M9nz65YKcr64Z5Uz1bLzl20&redir\_esc=y#v=onepage&q&f=false.  
 [Accessed 5 March 2024].
- [ A. V. Aho, "Data Structures and Algorithms," July 2001. [Online]. Available:  
 1 [https://d1wqtxts1xzle7.cloudfront.net/45322213/Data\\_Structures\\_and\\_Algorithms\\_-Alfred\\_V.\\_Aho\\_\\_John\\_E.\\_Hopcroft\\_and\\_Jeffrey\\_D.\\_Ullman-libre.pdf?1462305215=&response-content-disposition=inline%3B+filename%3DData\\_Structures\\_and\\_Algorithms\\_Chapter\\_1.pdf&Expires=1462305215](https://d1wqtxts1xzle7.cloudfront.net/45322213/Data_Structures_and_Algorithms_-Alfred_V._Aho__John_E._Hopcroft_and_Jeffrey_D._Ullman-libre.pdf?1462305215=&response-content-disposition=inline%3B+filename%3DData_Structures_and_Algorithms_Chapter_1.pdf&Expires=1462305215).  
 ] libre.pdf?1462305215=&response-content-disposition=inline%3B+filename%3DData\_Structures\_and\_Algorithms\_Chapter\_1.pdf&Expires=1462305215. [Accessed 1 March 2024].
- [ K. Mehlhorn, "Algorithms and Data Structures," 2008. [Online]. Available:  
 1 <https://link.springer.com/book/10.1007/978-3-540-77978-0>. [Accessed 2 March 2024].  
 7  
 ]
- [ M. W. Moskewicz, "Chaff: engineering an efficient SAT solver," June 2001. [Online]. Available:  
 1 [https://dl.acm.org/doi/abs/10.1145/378239.379017?casa\\_token=7y-1MPVW1GcAAAAA:t3gXe2u8\\_Dj8VUsbu9VVYXpdI2Wwo57Cun\\_lujDIdJt6bQNzsBbiCimn2d\\_vFCfJiUa7WyqAQkWx](https://dl.acm.org/doi/abs/10.1145/378239.379017?casa_token=7y-1MPVW1GcAAAAA:t3gXe2u8_Dj8VUsbu9VVYXpdI2Wwo57Cun_lujDIdJt6bQNzsBbiCimn2d_vFCfJiUa7WyqAQkWx). [Accessed 2 February 2024].
- [ Dave A. D. Tompkins, "Captain Jack: New Variable Selection Heuristics in Local Search for SAT," 2011. [Online]. Available: [https://link.springer.com/chapter/10.1007/978-3-642-21581-0\\_24](https://link.springer.com/chapter/10.1007/978-3-642-21581-0_24). [Accessed 20 March 2024].  
 ]
- [ J. Marques-Silva, "Chapter 4. Conflict-Driven Clause Learning SAT Solvers," [Online]. Available:  
 2 <https://ebooks.iospress.nl/volumearticle/56511>. [Accessed 2 March 2024].  
 0  
 ]
- [ M. Hertz, "Garbage collection without paging," June 2005. [Online]. Available:  
 2 [https://dl.acm.org/doi/abs/10.1145/1065010.1065028?casa\\_token=jNWoCr6UyuwAAAA1A:1Qj335afH6WIXXjqN1ggWGLok1swSHq2Y8B7uY\\_33DbBrAsEXzVLmtB3lIrnNej3OE7s9fn](https://dl.acm.org/doi/abs/10.1145/1065010.1065028?casa_token=jNWoCr6UyuwAAAA1A:1Qj335afH6WIXXjqN1ggWGLok1swSHq2Y8B7uY_33DbBrAsEXzVLmtB3lIrnNej3OE7s9fn)tcrAY. [Accessed 2 April 2024].
- [ N. Cacho, "How Does Exception Handling Behavior Evolve? An Exploratory Study in Java and C# Applications," 2014. [Online]. Available:  
 2 [https://ieeexplore.ieee.org/abstract/document/6976069?casa\\_token=lh7bpQ8FXRgAAA](https://ieeexplore.ieee.org/abstract/document/6976069?casa_token=lh7bpQ8FXRgAAA)AA:tKAUlowA9hiqirnKGko7LvjW9MfOUTVvpXBz4fYrDKqpxsXg5CPq1ZgRpQxXI8rUIVuPbsgAmA. [Accessed 15 April 2024].
- [ Visual Studio Code, "Hundreds of programming languages supported," 5 February 2024. [Online]. Available: <https://code.visualstudio.com/docs/languages/overview>. [Accessed 3 April 2024].  
 ]
- [ V. Cosentino, "A Systematic Mapping Study of Software Development With GitHub," 27 March 2017. [Online]. Available:  
 4 <https://ieeexplore.ieee.org/abstract/document/7887704>. [Accessed 12 April 2024].  
 ]

[ G. Dodig-Crnkovic, "On the Importance of Teaching," [Online]. Available:  
2 [https://d1wqxts1xzle7.cloudfront.net/30386605/teachingprofethics-5-libre.pdf?1390886291=&response-content-disposition=inline%3B+filename%3DOn\\_the\\_Importance\\_of\\_Teaching\\_Profession.pdf&Expires=1715820414&Signature=PiQibI5i06Lr8nqxi-wwhFUeAECKK4ujaFftwxcEbtq](https://d1wqxts1xzle7.cloudfront.net/30386605/teachingprofethics-5-libre.pdf?1390886291=&response-content-disposition=inline%3B+filename%3DOn_the_Importance_of_Teaching_Profession.pdf&Expires=1715820414&Signature=PiQibI5i06Lr8nqxi-wwhFUeAECKK4ujaFftwxcEbtq).  
] [Accessed 28 March 2024].