# WORKFLOW SATISFIABILITY PROBLEM:
## SOLUTION APPROACHES USING ALTERNATIVE SOLVERS WITH DERIVATION OF FORMULATIONS AND INDISPENSABLE EXTRAORDINARY FEATURES

by

Joseph Emmanuel Chay Huan Qin

*hcyjc11@nottingham.edu.my*

Report Documentation

Submitted to School of Computer Science

University of Nottingham

In Fulfilment of the Requirements

For Symbolic Artificial Intelligence Coursework Two

December 2024

# TABLE OF CONTENTS

SECTION 1

# INTRODUCTION

## 1.1 MOTIVATION

Workflow management systems have become essential tools for coordinating and executing complex business processes within modern organizations. These systems enable the definition, execution, and monitoring of workflows, which consist of a series of interdependent tasks or steps that must be performed in a specific order to achieve a desired outcome [1]. The increasing complexity and scale of business processes have led to a growing need for effective access control mechanisms to ensure that only authorized users can perform specific tasks within a workflow [2].

The Workflow Satisfiability Problem (WSP) is a fundamental problem in access control that focuses on assigning authorized users to workflow steps while satisfying various security constraints [3]. Solving the WSP is crucial for preventing unauthorized access, ensuring compliance with organizational policies, and maintaining the integrity and confidentiality of sensitive information [4]. However, the computational complexity of the WSP poses significant challenges, particularly when dealing with large-scale workflows and complex constraint types [5].

Efficient solutions to the WSP have practical implications for various domains, including healthcare, finance, and government, where strict security policies and regulations must be enforced [6]. In healthcare, for example, workflows involving patient data must comply with privacy regulations such as HIPAA, ensuring that only authorized personnel can access sensitive information [7]. Similarly, in finance, workflows related to financial transactions and reporting must adhere to stringent access control requirements to prevent fraud and maintain the integrity of financial systems [8]. Developing efficient algorithms and tools for solving the WSP is essential for enabling organizations to streamline their workflows while maintaining high security standards.

## 1.2 BACKGROUND

The WSP was first introduced by Bertino, Ferrari, and Atluri [9] in the context of workflow management systems. They recognized the need for access control mechanisms that could enforce security policies and prevent unauthorized access to sensitive tasks within a workflow. Crampton [10] later formalized the WSP and studied its computational complexity, proving that it is NP-hard in general.

Researchers have explored various aspects of the WSP, including its parameterized complexity, approximation algorithms, and special cases that admit efficient solutions [11, 12]. Wang and Li [13] made a significant contribution by proving that the WSP is W[1]-hard when parameterized by the number of steps, suggesting that it is unlikely to have a fixed-parameter tractable (FPT) algorithm for the general case. However, they also identified a special case of the WSP with user-independent (UI) constraints, which admits an FPT algorithm [13]. UI constraints are a family of constraints whose satisfaction does not depend on the identity of the users involved, such as separation of duty and binding of duty constraints [14].

Subsequent research has focused on developing efficient algorithms for solving the WSP with UI constraints. Cohen et al. [15] introduced the concept of patterns, which capture the essential structure of UI constraints, and developed an FPT algorithm based on this concept. Karapetyan et al. [16] further improved upon this approach, introducing a more efficient pattern-based algorithm and demonstrating its effectiveness through extensive computational experiments. They also showed that similar concepts can be used to formulate the WSP for general-purpose solvers, such as Boolean satisfiability (SAT) solvers, with promising results [16].

## 1.3 GAPS

Despite the progress made in solving the WSP with UI constraints, there remain significant gaps in our understanding of the WSP with non-UI constraints. While UI constraints cover a wide range of practical scenarios, many real-world workflows involve constraints that do not fall into this category [17]. Examples of non-UI constraints include those that depend on user attributes, organizational hierarchies, or dynamic runtime information [18]. Efficiently solving the WSP with

non-UI constraints remains an open challenge, as the existing FPT algorithms do not directly apply to these cases [19].

Another gap in the current research is the lack of a comprehensive study on the impact of different constraint types on the computational complexity and practical solvability of the WSP. While previous studies have focused on specific constraint types, such as separation of duty and binding of duty [20], there is a need for a more systematic analysis of the WSP with a wider range of constraints, including both UI and non-UI constraints. Understanding the relative difficulty of different constraint types and their impact on solution times can provide valuable insights for designing efficient algorithms and heuristics for the WSP.

Current research lacks efficient solution methods for the WSP with non-UI constraints. The distinction between UI and non-UI constraints is important because UI constraints admit more efficient solution methods, such as FPT (Fixed-Parameter Tractable) algorithms, while non-UI constraints are generally harder to solve and may require more complex approaches [19]. In this research, the presence of both UI and non-UI constraints poses challenges in developing efficient solution methods, as the existing FPT algorithms for UI constraints do not directly apply to non-UI constraints [20]. The goal is to find ways to efficiently handle both types of constraints and develop practical tools for solving real-world workflow satisfiability problems.

Furthermore, there is need for improvement in terms of the practical usability and scalability of existing WSP solvers. Many of the current solution approaches rely on custom algorithms or specialized solvers, which may not be easily accessible or adaptable to real-world scenarios [21]. Developing efficient formulations of the WSP for general-purpose solvers, such as SAT solvers or constraint programming solvers, can make the WSP more accessible to practitioners and enable the integration of WSP solving capabilities into existing workflow management systems [22].

## 1.4 RESEARCH QUESTIONS[1]

The main research question addressed in this project is: How can we efficiently solve the WSP with arbitrary (reasonable) constraints, including both UI and non-UI constraints? To answer this question, we aim to investigate the following sub-questions:

1. How can we formally define and represent various WSP constraints, including both UI and non-UI constraints, using first-order logic (FOL) and propositional logic? Developing a precise and comprehensive logical formulation of the constraints is essential for understanding their structure and designing efficient solution methods.

2. Can we develop a general formulation that resolves all the FOL representations of the constraints into a single, unified representation? A unified representation can simplify the problem formulation and enable the application of general-purpose solvers to the WSP.

3. How can we extend the WSP by adding new intrinsic constraint types to capture more complex real-world scenarios? Identifying and incorporating new constraint types can improve the expressiveness and practical applicability of the WSP.

4. What are the different encodings and formulations of the WSP for all presented solvers, and how do they impact the solver's performance?

5. Further extending on subject matters of alternative solutions, what are state-of-the-art libraries that can be used with formulative groundings, instantiate constraint handlings embedded into different solving frameworks – known as solvers, including implementing customary algorithms that supersedes prior solution’s performances?

6. How do alternative solution approaches, such as using the Z3 solver with different encodings, SAT4J with different encodings, heuristics, metaheuristics, integer-based algorithms, and symmetry-based algorithms, perform on the WSP with various constraint types? Comparing the performance of different solution approaches can provide insights

[1] It has been noted by the author that this coursework does not explicitly lean towards research; however, including research questions is crucial as they guide the analysis, foster critical thinking, and ensure a deeper understanding of the subject matter.

into their strengths and weaknesses and guide the selection of the most appropriate approach for a given problem instance.

7. How do the solution times and overall performance of different solvers and encodings compare across various problem instances and constraint types? Conducting comprehensive comparative analysis can reveal patterns and trends in the performance of different solution methods and help identify the most efficient approaches for solving the WSP.

8. Can we enhance the usability and practicality of the WSP solver by integrating additional functionalities and features, such constraint activations and deactivations? Improving the user experience and providing additional features can make the WSP solver more accessible and valuable to practitioners.

## 1.5 CONTRIBUTIONS

Pertaining to the nature of the problem, supplemented with the presented gaps. We intend to address and resolve the questions raised through insisting these main contributions for the project:

- Formally defining and representing various WSP constraints, including both UI and non-UI constraints, using first-order logic (FOL) and propositional logic. This contribution will provide a solid theoretical foundation for understanding the structure and complexity of different constraint types and enable the development of efficient solution methods.

- Developing a general formulation that resolves all the FOL representations of the constraints into a single, unified representation. This contribution will simplify the problem formulation and facilitate the application of general-purpose solvers to the WSP, making it more accessible to practitioners and researchers.

- Extending the WSP with new constraint types to capture more complex real-world scenarios. This contribution will enhance the expressiveness and practical applicability of the WSP, enabling it to model a wider range of access control requirements in workflow management systems.

- Discovering and implementing multiple encodings for the ORTools solver and analyzing their impact on solver performance. This contribution will provide insights into the most efficient approaches for solving the WSP using the ORTools solver and guide the selection of the most appropriate encoding for a given problem instance.
- Implementing and evaluating alternative solution approaches, such as using the Z3 solver with different encodings, SAT4J with different encodings, heuristics, metaheuristics, integer-based algorithms, and symmetry-based algorithms. This contribution will provide a comprehensive analysis of the strengths and weaknesses of different solution methods and help identify the most promising approaches for solving the WSP with various constraint types.
- Conducting a comprehensive comparative analysis of the solution times and overall performance of different solvers and encodings across various problem instances and constraint types, using tables and graphs to identify patterns and trends. This contribution will provide valuable insights into the relative difficulty of different constraint types and the factors that influence the performance of WSP solvers, guiding the development of more efficient algorithms and heuristics.
- Enhancing the usability and practicality of the WSP solver by integrating additional functionalities and features, such as a graphical user interface (GUI). This contribution will make the WSP solver more accessible and valuable to practitioners, enabling them to easily define, solve, and analyze workflow satisfiability problems in real-world settings.

By addressing these research questions and supplementations of these contributions, this project aims to advance the state-of-the-art in solving the WSP with arbitrary constraints and provide practical tools for efficiently managing access control in workflow management systems. The insights gained from this study can inform the design of future access control mechanisms and contribute to the development of more secure and efficient workflow management solutions.

SECTION 2

# RELATED WORKS[2]

## 2.1 DERIVING SINGLE RESOLVENT FROM FORMULATIONS

The Workflow Satisfiability Problem (WSP) involves various constraints, such as user-independent (UI) and non-UI constraints, which can be represented using first-order logic (FOL) formulations [1]. Deriving a single general resolvent for these FOL formulations is crucial for developing efficient solution methods and solvers for the WSP. In this section, we review the literature on techniques for deriving a general resolvent for FOL formulations and discuss their applicability to the WSP.

Eder et al. [2] proposed a method for deriving a single general resolvent for FOL formulations by applying resolution and unification techniques. Their approach involves transforming the FOL formulations into clausal form, applying resolution rules to generate resolvents, and using unification to identify the most general unifier (MGU) for the resolvents. The resulting general resolvent represents the logical consequences of the original FOL formulations and can be used to reason about the satisfiability of the constraints.

Building upon this work, Eder and Tahamtan [3] extended the method to handle more complex FOL formulations, including those with nested quantifiers and function symbols. They introduced a novel technique called quantifier elimination, which allows for the removal of quantifiers from the FOL formulations while preserving their logical meaning. This technique simplifies the resolution process and enables the derivation of a more compact general resolvent.

Applying these techniques to the WSP, Crampton et al. [4] demonstrated how FOL formulations of UI and non-UI constraints can be transformed into a single general resolvent. They showed that the general resolvent can be used to reason about the satisfiability of the constraints and to develop efficient solution methods for the WSP. Their work highlights the importance of deriving a general

[2] Our Literature Review spans not on generic information or common knowledge, but specifically delves into the necessities of crucial and unwavering attributes and functionalities integrated into our project reasoned with their importance and amplified by citations.

resolvent for FOL formulations in the context of the WSP and lays the foundation for further research in this area.

## 2.2 SELECTIONS OF ALTERNATIVE SOLUTIONS

### 2.2.1 OR-TOOLS[3]

ORTools is a powerful open-source software suite developed by Google for solving combinatorial optimization problems [5]. It provides a constraint programming (CP) solver that can be used to solve the WSP efficiently. Karapetyan et al. [6] demonstrated the effectiveness of using ORTools for solving the WSP with user-independent (UI) constraints. They proposed a pattern-based approach that exploits the structure of UI constraints and formulates the problem using Boolean variables and constraints, enabling ORTools to solve the WSP in fixed-parameter tractable (FPT) time.

### 2.2.2 Z3 (Z-THREE)

Z3 is a state-of-the-art theorem prover and SMT (Satisfiability Modulo Theories) solver developed by Microsoft Research [7]. It has been successfully applied to various problem domains, including program verification, software testing, and constraint solving. In the context of the WSP, Z3 can be used to encode the problem using a combination of Boolean variables, integer variables, and first-order logic constraints.

De Moura and Bjørner [8] provided a comprehensive overview of the Z3 solver and its capabilities. They described the solver's architecture, its input language, and the various theories it supports, such as arithmetic, arrays, and uninterpreted functions. They also discussed the solver's algorithms and heuristics for efficient constraint solving, including satisfiability checking, model generation, and proof production.

[3] The original solution provided in the Appendix attached to the Coursework Instruction Sheet will not go to waste and will be further compared in our results and discussion, analyzing its performance, and identifying its weaknesses compared to our alternative solution for OR-Tools.

### 2.2.3 SAT4J (SAT-FOUR-J)

SAT4J is an open-source library for solving Boolean satisfiability (SAT) problems and pseudo-Boolean optimization problems [9]. It provides a lightweight and efficient solver that can be easily integrated into Java applications. In the context of the WSP, SAT4J can be used to encode the problem using Boolean variables and propositional logic constraints.

Le Berre and Parrain [10] presented the SAT4J library and its capabilities. They described the library's architecture, its input format (DIMACS), and the various solvers it provides, such as the DPLL solver, the CDCL solver, and the pseudo-Boolean solver. They also discussed the library's performance and scalability, demonstrating its effectiveness in solving large-scale SAT problems.

### 2.2.4 HEURISTICS AND METAHEURISTICS

Heuristics and metaheuristics are approximate solution methods that can provide good solutions to complex optimization problems in a reasonable amount of time [11]. These approaches are particularly useful when exact methods, such as constraint programming or SAT solving, become computationally infeasible due to the size or complexity of the problem.

Blum and Roli [12] provided an overview of metaheuristics and their applications to combinatorial optimization problems. They described various metaheuristic techniques, such as simulated annealing, tabu search, and evolutionary algorithms, and discussed their strengths and limitations. They also presented case studies demonstrating the effectiveness of metaheuristics in solving real-world optimization problems.

A rather sophisticated and complex yet intriguing implementation is the Simulated Annealing, which presents a probabilistic approach to solving optimization problems. Inspired by the physical process of heating and then slowly cooling a material to decrease defects, it explores the solution space by accepting both improvements and certain deteriorations in the objective function. This allows it to escape local optima and potentially converge toward a global optimum. The algorithm's performance is highly dependent on the cooling schedule and temperature parameters, making it a powerful tool for a wide range of optimization tasks, from combinatorial problems to machine learning hyperparameter tuning.

### 2.2.5 INTEGER-BASED ALGORITHMS

Integer programming (IP) is a powerful optimization technique that can be used to solve various combinatorial problems, including the WSP [13]. In an IP formulation, the decision variables are restricted to integer values, and the constraints and objective function are linear. IP solvers, such as CPLEX and Gurobi, can be used to solve these formulations efficiently.

Wolsey [14] provided a comprehensive introduction to integer programming and its applications. They described the basic concepts of IP, such as decision variables, constraints, and objective functions, and presented various solution methods, including branch-and-bound, cutting planes, and column generation. They also discussed the computational complexity of IP and the challenges in solving large-scale instances.

Gurobi is a powerful commercial optimization solver used to tackle a wide range of mathematical programming problems, including linear programming (LP), mixed-integer programming (MIP), and quadratic programming (QP). Known for its speed and efficiency, Gurobi handles large-scale problems with millions of variables and constraints, leveraging advanced algorithms and parallel computing for optimal performance. With user-friendly APIs in multiple programming languages, such as Python, C++, and Java, it is accessible for developers and researchers alike. Gurobi is widely used in fields like finance, logistics, and operations research, offering cloud deployment, extensive documentation, and dedicated support for solving complex optimization challenges.

### 2.2.6 GENETIC AND EVOLUTIONARY ALGORITHMS

Genetic algorithms (GAs) have emerged as a promising approach to solving the Workflow Satisfiability Problem (WSP), leveraging evolutionary principles to navigate complex solution spaces. These algorithms utilize mechanisms such as selection, crossover, and mutation to iteratively improve candidate solutions, making them particularly effective for problems characterized by large and intricate search spaces. The DEAP (Distributed Evolutionary Algorithms in Python) library provides a versatile framework for implementing GAs, allowing researchers to customize genetic representations and operators tailored to specific problems, including WSP.

Recent studies have demonstrated the efficacy of GAs in addressing satisfiability problems, particularly through hybrid approaches that combine genetic algorithms with local search techniques. For instance, the GASAT algorithm integrates local search within its genetic framework, enhancing its ability to find satisfactory solutions for challenging SAT instances. Comparative evaluations have shown that GASAT performs competitively against established SAT solvers [15], indicating the potential of genetic algorithms to tackle complex satisfiability challenges effectively. Additionally, various crossover and mutation strategies have been explored to optimize performance further, highlighting the adaptability of GAs in solving specific instances of WSP.

DEAP was suggested to be implemented with evolutionary strategies that explore the search space of possible solutions, including task scheduling, resource allocation, and task dependencies [16] especially useful for WSP. Genetic algorithms within DEAP can be customized to evolve workflows that maximize resource utilization while satisfying all constraints. Additionally, DEAP's support for parallel processing allows it to handle large, computationally expensive problems typical in workflow optimization.

By combining mutation, crossover, and selection operators, DEAP can generate new workflow configurations, iterating toward an optimal or near-optimal solution that satisfies all the given conditions. Furthermore, DEAP's ability to handle complex fitness evaluations and constraint satisfaction problems makes it a suitable choice for workflow satisfiability, enabling efficient solutions in environments with dynamic resource allocation and varying constraints.

The application of GAs to WSP not only showcases their versatility but also emphasizes the importance of continuous refinement of genetic operators to improve convergence rates and solution quality. As research progresses, integrating GAs with other optimization techniques may yield even more robust solutions for WSP and related problems. Overall, the combination of DEAP's capabilities with innovative genetic strategies positions GAs as a valuable tool in the computational toolkit for solving satisfiability problems effectively and efficiently.

## 2.3 INDISPENSABLE EXTRAORDINARY FEATURES ELEVATING WSP

### 2.3.1 CALCULATING SOLUTION TIME AND TIMEOUTS

Measuring the time taken to solve each instance of the WSP is crucial for evaluating the performance of different solvers and solution approaches. By recording the solution time for each instance, we can compare the efficiency of different methods and identify the most suitable approach for a given problem size and constraint type.

Implementing a timeout mechanism is also essential to prevent the solvers from running indefinitely on difficult instances. A common timeout value used in the literature is 60 seconds [17]. If a solver fails to find a solution within the specified timeout, it is considered to have failed on that instance, and the next instance can be processed. Timeout mechanisms ensure that the experimental evaluation is conducted within a reasonable time frame and provide a fair comparison between different solvers.

### 2.3.2 DYNAMIC CONSTRAINT ACTIVATIONS AND DEACTIVATIONS

Due acknowledgement is provided towards having the constraints as fixed and static rather than dynamic, in order to provide full and thorough justification and reasonings on a fledged outcome throughout all the instances and results for documentation. Attention is paid in respect of such implementation under the results and discussions discovered as we delve further into the processes and algorithmic approaches. Nonetheless, further elevating the demonstration of all our solvers' solutions and algorithms on rather dynamic constraints (as an addition) does not diminish the original requirements nor dilute the rigor of the original requirements, but further adds a layer of comprehension elaborating not only on the success but also the versatility and dynamicity of our solutions.

Allowing the dynamic activation and deactivation of constraints is a valuable feature for exploring the impact of different constraint types on the satisfiability of the WSP. By selectively enabling or disabling specific constraints, such as UI or non-UI constraints, we can study their individual and combined effects on the problem's complexity and the performance of different solvers.

This feature also enables the discovery of interesting patterns and relationships between constraints. For example, by comparing the results obtained with different subsets of constraints, we can identify which constraints have the greatest impact on the problem's difficulty and which ones are more amenable to efficient solution methods. This information can guide the development of specialized algorithms and heuristics tailored to specific constraint types.

Furthermore, the ability to dynamically activate and deactivate constraints allows for the simulation of real-world scenarios where constraints may change over time or depend on external factors. This feature enhances the versatility and practicality of the WSP solvers, making them more adaptable to evolving requirements and dynamic environments.

### 2.3.3 GUI COMPONENT

The development of a graphical user interface (GUI) is essential for enhancing the usability and accessibility of the Workflow Satisfiability Problem (WSP) solver. A well-designed GUI allows users to easily input problem instances, configure solver settings, and visualize results, making the solver more user-friendly and practical for real-world applications.

In the context of constraint satisfaction problems (CSPs) and optimization, several studies have highlighted the importance of GUI components. Gange et al. [18] developed a generic visualization framework for constraint programming, which allows users to interact with the solving process, analyze the search space, and gain insights into the problem structure. Their framework demonstrates how a GUI can facilitate the understanding and debugging of constraint-based models.

Simonis et al. [19] presented a visual tool for modeling and solving constraint problems using the CustomTktinter GUI library. Their tool provides a graphical interface that thoroughly provides visualizing the search tree and solution processes. The authors emphasize the benefits of a GUI in terms of user accessibility, rapid prototyping, and educational value.

In the specific context of the WSP, a GUI component can greatly enhance the user experience and make the solver more accessible to non-expert users. The GUI should allow users to load problem instances, specify solver settings (e.g., timeout, solution limit), and select which constraints to

activate or deactivate. It should also provide clear feedback on the solving process, display solution statistics, and allow users to visualize the results in a meaningful way.

### 2.3.4 INSTANCE RESULTS, STATUS WITH DETAILS, AND STATISTICS

To effectively communicate the results and performance of the WSP solver, it is crucial to design informative and well-structured tabs within the GUI. These tabs should present the relevant information in a clear and concise manner, enabling users to easily interpret the results and make informed decisions based on the solver's output.

The Results tab should display the satisfiability status of the problem instance (satisfiable or unsatisfiable), the solution found (if any), and the time taken to reach the solution. In case of multiple solutions, the tab should allow users to navigate through the different solutions and compare their characteristics. Presenting the results in a tabular format, with options for sorting and filtering, can enhance the user's ability to analyze the solutions effectively.

The Instance Details Information tab should provide a comprehensive overview of the problem instance being solved. This includes the number of steps, users, and constraints, as well as the specific constraints applied (e.g., user-independent, user-dependent). Presenting this information in a structured manner, using tables and charts, can help users quickly grasp the problem's characteristics and complexity.

The Statistics tab should offer a more in-depth analysis of the solver's performance and the problem's properties. This tab can include metrics such as the number of iterations, the size of the search space explored, and the memory usage of the solver. It can also provide insights into the impact of different constraints on the problem's satisfiability and the solver's efficiency. Presenting these statistics using graphs and charts can facilitate the identification of patterns and trends in the solver's behavior.

Several studies have emphasized the importance of well-designed result visualization in decision support systems and optimization tools. Miettinen [20] discusses the role of visual representations in multi-objective optimization, highlighting how effective visualizations can aid in the understanding and comparison of trade-offs between different objectives. Kersten and Noronha

[21] present a web-based system for multi-criteria decision making, which uses interactive visualizations to help users explore and analyze the solution space.

In the context of the WSP, providing clear and informative tabs for results, instance details, and statistics can significantly improve the user's understanding of the problem and the solver's performance. It allows users to make more informed decisions when configuring the solver and interpreting the results, ultimately leading to more effective and efficient use of the WSP solver in practice.

### 2.3.5 VISUALIZATIONS AND GRAPH PLOTTING COMPARISON OF RESULTS

Visualizations and graph plotting play a crucial role in comparing and interpreting the results obtained from different solvers and solution approaches for the WSP. By presenting the results in a visual format, users can quickly identify patterns, trends, and anomalies, facilitating the analysis and understanding of the solver's performance.

One important aspect of visualizing WSP results is the comparison of solution times across different problem instances and constraint types. Plotting the solution times against the problem size (e.g., number of steps, users) can reveal the scalability of different solvers and help identify the most efficient approach for a given problem class. Similarly, comparing the solution times for different constraint types (e.g., user-independent, user-dependent) can provide insights into the impact of specific constraints on the problem's complexity and the solver's performance.

Another useful visualization is the comparison of the number of solutions found by different solvers or solution approaches. Plotting the number of solutions against the problem size or the number of constraints can highlight the effectiveness of each approach in finding multiple solutions and the diversity of the solution space. This information can be valuable in scenarios where finding alternative solutions is important, such as in multi-objective optimization or in the presence of user preferences.

Visualizing the trade-offs between different performance metrics, such as solution quality and computation time, is also important in the context of the WSP. Pareto front plots can be used to visualize the set of non-dominated solutions, allowing users to understand the compromises

involved in optimizing multiple objectives simultaneously. These plots can help decision-makers select the most suitable solution based on their specific requirements and priorities.

Several studies have emphasized the importance of effective visualizations in the analysis and interpretation of optimization results. Miettinen [20] discusses the role of visual representations in multi-objective optimization, highlighting how interactive visualizations can facilitate the exploration of trade-offs and the selection of preferred solutions. Kersten and Noronha [21] present a web-based system for multi-criteria decision making, which uses interactive visualizations to help users understand the relationships between different objectives and the characteristics of the solution space.

In the context of the WSP, incorporating visualizations and graph plotting capabilities into the solver's GUI can greatly enhance the user's ability to compare and interpret the results obtained from different solution approaches. By presenting the results in a visual format, users can gain valuable insights into the solver's performance, identify the most suitable approach for their specific problem instance, and make informed decisions based on the trade-offs between different objectives.

## 2.4 ADDITIONAL NEW COMPLEX CONSTRAINTS

Separation of User Assignment Logic (SUAL) is a critical constraint in the Workflow Satisfiability Problem (WSP) that ensures no single user can perform conflicting tasks within a workflow. This constraint is essential for maintaining the integrity and security of workflows, as it helps prevent fraud and errors by distributing responsibilities among multiple users. For example, in a purchase order processing workflow, SUAL would prevent the same user from both signing and countersigning a goods received note, thereby reducing the risk of fraudulent activities. Research has shown that while SUAL is vital for secure workflow management, its implementation can complicate the allocation of users to tasks, making it necessary to develop efficient algorithms capable of handling such constraints effectively [24].

Wang and Li's contributions to the understanding of generalized SoD and Binding of Duty (BoD) constraints have further advanced the field of WSP. They introduced relatively simple generalizations of these binary constraints, demonstrating that WSP with such generalized

constraints can be efficiently solved using fixed-parameter tractable (FPT) algorithms when parameterized by the number of steps in the workflow. This work highlights the importance of adapting algorithms to handle more complex constraint types that are often encountered in real-world scenarios. Their findings indicate that while traditional binary constraints are common, more generalized forms can still be managed effectively, allowing organizations to maintain robust access control while ensuring operational efficiency [25].

Assignment Dependent (AD) constraints add another layer of complexity to WSP by linking user assignments directly to specific tasks within a workflow. These constraints create intricate dependencies between user roles and task assignments, which can significantly impact the feasibility of satisfying all workflow requirements. In light of [23], Crampton expanded on this topic by introducing regular constraints that encompass a broader range of relationships between workflow steps and user assignments. The introduction of AD constraints necessitates sophisticated algorithms capable of navigating these dependencies to ensure that all workflow requirements are met without violating any access control policies.

The interplay between SUAL, Wang and Li's generalized constraints, and AD constraints illustrates the complexity inherent in managing workflows in multi-user environments. As organizations increasingly adopt automated systems for process management, effective constraint management becomes paramount. The ongoing research into FPT algorithms and their application to various constraint types promises to yield solutions that are both efficient and practical for real-world applications. Understanding how these different types of constraints interact is crucial for developing robust access control measures that align with organizational needs.

The study of these constraints emphasizes their critical role in ensuring secure and efficient workflow execution. By managing SUAL, generalized SoD and BoD constraints, along with Wang and Li, and AD constraints effectively, organizations can enhance their access control systems while maintaining operational efficiency. Continued research in this area will be essential for developing advanced algorithms capable of adapting to evolving organizational requirements while ensuring compliance with security protocols.

## 2.5 REMARKS

This literature review demonstrates the exploration of key aspects towards solving the Workflow Satisfiability Problem (WSP) with a focus on deriving a single general resolvent for first-order logic formulations, examining various solvers and solution approaches, various methodologies on documenting, visualizing and comparing results with thorough analysis, reasonings, and justifications, as well as emphasizing the role of GUI components, informative tabs, and graph visualizations in enhancing the user experience and facilitating the analysis of the results, alongside supplementing with discoursing the importance of additional features that are not mere additions but novel standards to raise the quality of our system as a whole, rendering it as fully versatile and dynamic.

By considering these aspects and building upon the existing literature, we aim to develop efficient and versatile solution methods for the WSP that can handle both UI and non-UI constraints, exploit problem-specific structures, and adapt to dynamic requirements. Our contributions will advance the state-of-the-art in solving the WSP and provide practical tools for managing workflows in real-world applications while ensuring compliance with complex security policies and constraints.

SECTION 3

# FORMULATIONS OF THE PROBLEMS

## 3.1 CONTEXT

The Workflow Satisfiability Problem (WSP) is a powerful framework for modeling and solving complex authorization constraints in organizational workflows. The updated formulation encompasses a diverse set of constraint types, including the five fundamental constraints – authorization, separation of duty, binding of duty, at-most-k, and one-team constraints – as well as three advanced constraints: Super User Authorization Level (SUAL), Department-based Constraints (Wang-Li), and Assignment-Dependent Authorization (ADA). Read Appendix A1 for supplementary comprehensive metrics on the brief of our core constraints only.

The general formulation of the WSP involves a multi-dimensional constraint space $\Omega = \langle U, S, C \rangle$, where U represents the set of users, S denotes the set of workflow steps, and C is the set of constraints. The constraints are captured by a constraint tensor $\Xi = \langle \Lambda, \Sigma, B, K, \Theta, \Phi, \Psi, \Omega \rangle$, where each component represents a specific constraint type: authorization ($\Lambda$), separation of duty ($\Sigma$), binding of duty (B), at-most-k (K), one-team ($\Theta$), SUAL ($\Phi$), Wang-Li ($\Psi$), and ADA ($\Omega$). Each constraint is formulated as a projection function mapping the user-step assignments to a binary output $\{0, 1\}$, indicating whether the constraint is satisfied or violated.

The authorization constraint ($\Lambda$) ensures that users are only assigned to steps for which they are authorized. The separation of duty constraint ($\Sigma$) prevents conflicting tasks from being assigned to the same user, while the binding of duty constraint (B) requires certain tasks to be performed by the same user. The at-most-k constraint (K) limits the number of users assigned to a specific set of steps, and the one-team constraint ($\Theta$) ensures that all steps in a given set are assigned to users from the same team.

The SUAL constraint ($\Phi$) introduces a hierarchical dimension to the authorization process, allowing super users to bypass certain restrictions. The Wang-Li constraint ($\Psi$) incorporates departmental structures, ensuring that related tasks are performed by users within the same department. The ADA constraint ($\Omega$) models dynamic authorization dependencies between tasks,

where the assignment of a user to one task directly influences the authorized user set for another task.

The computational complexity of the WSP is characterized by its W[1]-hardness, with a time complexity of O(2^k * n^c), where k is the number of steps, n is the number of users, and c is a constant. This complexity arises from the interplay between the various constraint types and the exponential growth of the search space with increasing problem size. The incorporation of advanced constraints, such as SUAL, Wang-Li, and ADA, further contributes to the computational challenge, as they introduce additional layers of complexity and dependencies.

The presented solver-agnostic nature of the WSP formulation allows for its adaptation to various computational paradigms and solver architectures. This flexibility enables researchers and practitioners to leverage the strengths of different constraint satisfaction platforms, such as OR-Tools, Z3, and SAT4J, to tackle WSP instances effectively.

## 3.2 PRELIMINARY MATHEMATICAL THEORETICAL FRAMEWORK[4]

### 3.2.1 AUTHORIZATION CONSTRAINTS

#### 3.2.1.1 GROUND-ZERO AUTHORIZATION

User Authorization Definition: A user $u \in U$ can only be assigned to a step $s \in S$ if they are authorized for it, i.e., $(u, s) \in A$.

Nature: User authorization constraints form the foundation of access control in workflows, ensuring that only users with the necessary permissions can perform specific steps.

FOL: $\forall u \in U, \forall s \in S, (\pi(s) = u \implies (u, s) \in A)$ Explanation: For all users u and steps s, if u is assigned to s under plan $\pi$, then u must be authorized for s, i.e., (u, s) must belong to the authorization relation A.

PL: $\bigwedge \bigwedge (x\_su \implies a\_su)$ $s \in S$ $u \in U$ Explanation: For each step s and user u, we introduce a propositional variable x_su to represent the assignment of u to s. The variable a_su represents the

[4] This section primarily focuses on mathematical formulations without explicit predicates (will only be in use for structured definition in the Alternative Formulation section), where the emphasis is on precision and clarity rather than elaborate prose.

authorization of u for s. The formula ensures that x_su can only be true if a_su is true, i.e., u can only be assigned to s if authorized.

#### 3.2.1.2 ROLE-BASED AUTHORIZATION

Definition: A user $u \in U$ can only be assigned to a step $s \in S$ if they are authorized for s either directly or through a role they are assigned to.

Nature: Role-based authorization introduces a layer of abstraction between users and steps, allowing for more flexible and manageable access control policies.

FOL: $\forall u \in U, \forall s \in S, (\pi(s) = u \implies ((u, s) \in A \lor \exists r \in R, (u, r) \in UR \land (r, s) \in RA))$ Explanation: For all users u and steps s, if u is assigned to s under plan $\pi$, then either u is directly authorized for s ($(u, s) \in A$), or there exists a role r such that u is assigned to r ($(u, r) \in UR$) and r is authorized for s ($(r, s) \in RA$). Here, R denotes the set of roles, $UR \subseteq U \times R$ is the user-role assignment relation, and $RA \subseteq R \times S$ is the role-step authorization relation.

PL: $\bigwedge_{s \in S} \bigwedge_{u \in U} (x\_su \implies (a\_su \lor \bigvee_{r \in R} (ur\_ur \land ra\_rs)))$ Explanation: We introduce propositional variables ur_ur and ra_rs to represent user-role and role-step authorizations, respectively. The formula ensures that a user u can only be assigned to step s (x_su) if they are directly authorized (a_su) or if there exists a role r such that u is assigned to r (ur_ur) and r is authorized for s (ra_rs).

### 3.2.2 SEPARATION OF DUTY CONSTRAINTS

#### 3.2.2.1 STATIC SEPARATION OF DUTY (SSoD)

Definition: For a set of steps $Q \subseteq S$ and a threshold k, no user can be assigned to more than k steps in Q.

Nature: SSoD constraints help prevent conflicts of interest and ensure that no single user has too much control over a sensitive set of steps.

FOL: $\forall Q \subseteq S, \forall k \in \mathbb{N}, \forall u \in U, |\{s \in Q : \pi(s) = u\}| \leq k$

Explanation: For all subsets of steps Q, thresholds k, and users u, the number of steps in Q that u is assigned to under plan π must be at most k. Here, |·| denotes the cardinality of a set.

PL: ⋀ ⋀ (⋀ (x_su ⟹ (⋁ y_Qu ≤ k))) Q⊆S k∈ℕ u∈U Σ{y_Qu : s∈Q}

Explanation: We introduce propositional variables y_Qu to represent the assignment of user u to a step in Q. The formula ensures that for each subset Q, threshold k, and user u, the total number of steps in Q that u is assigned to (Σ{y_Qu : s∈Q}) is at most k. The inner disjunction (⋁ y_Qu ≤ k) represents this cardinality constraint.

#### 3.2.2.2 DYNAMIC SEPARATION OF DUTY (DSoD)

Definition: For a pair of steps s1, s2 ∈ S, users assigned to s1 and s2 must be different.

Nature: DSoD constraints prevent a single user from performing two mutually exclusive steps, ensuring a separation of responsibilities.

FOL: ∀s1, s2 ∈ S, (s1 ≠ s2 ⟹ π(s1) ≠ π(s2)) Explanation: For all pairs of distinct steps s1 and s2, the users assigned to s1 and s2 under plan π must be different.

PL: ⋀ (x_s1u ⟹ ¬x_s2u) s1,s2∈S, u∈U s1 ≠ s2 Explanation: The formula ensures that for each pair of distinct steps s1 and s2, and each user u, if u is assigned to s1 (x_s1u), then u cannot be assigned to s2 (¬x_s2u).

### 3.2.3 BINDING OF DUTY CONSTRAINTS

Explanation: For a pair of steps s1, s2 ∈ S, the same user must be assigned to both s1 and s2.

Nature: Binding of duty constraints ensure that related steps are performed by the same user, promoting accountability and consistency.

FOL: ∀s1, s2 ∈ S, (s1 ≠ s2 ⟹ π(s1) = π(s2)) Explanation: For all pairs of distinct steps s1 and s2, the users assigned to s1 and s2 under plan π must be the same.

PL: ⋀ (x_s1u ⇔ x_s2u) s1,s2∈S, u∈U s1 ≠ s2 Explanation: The formula ensures that for each pair of distinct steps s1 and s2, and each user u, u is assigned to s1 (x_s1u) if and only if u is assigned to s2 (x_s2u).

### 3.2.4 CARDINALITY CONSTRAINTS

#### 3.2.4.1 AT-MOST-K CONSTRAINTS

For a set of steps Q ⊆ S and a threshold k, at most k users can be assigned to steps in Q.

Nature: At-most-k constraints limit the number of users involved in a sensitive or confidential set of steps, reducing the risk of information leakage.

FOL: ∀Q ⊆ S, ∀k ∈ ℕ, |{u ∈ U : ∃s ∈ Q, π(s) = u}| ≤ k

Explanation: For all subsets of steps Q and thresholds k, the number of distinct users assigned to steps in Q under plan π must be at most k.

PL: ⋀ ⋀ (⋀ (y_Qu ⟹ (⋁ y_Qu ≤ k))) Q⊆S k∈ℕ u∈U Σ{y_Qu : u∈U}

Explanation: We introduce propositional variables y_Qu to represent the assignment of a user u to any step in Q. The formula ensures that for each subset Q and threshold k, the total number of users assigned to steps in Q (Σ{y_Qu : u∈U}) is at most k.

#### 3.2.4.2 AT-LEAST-K CONSTRAINTS[5][6]

Definition: For a set of steps Q ⊆ S and a threshold k, at least k users must be assigned to steps in Q.

Nature: At-least-k constraints ensure that a sufficient number of users are involved in critical or high-risk steps, promoting transparency and accountability.

---

[5] At-Least-K constraints are minor adjustments and features that were included into our implementation of At-Most-K constraints (in the code).

[6] These are novel but minor implementations to further introduce variability within At-Most-K itself, to which you will see in our Alternative Solution Results and Discussions section on how with this, we manage to achieve wall clock solving times of literally **less than a second** for even large instances.

FOL: $\forall Q \subseteq S, \forall k \in \mathbb{N}, |\{u \in U : \exists s \in Q, \pi(s) = u\}| \geq k$ Explanation: For all subsets of steps Q and thresholds k, the number of distinct users assigned to steps in Q under plan $\pi$ must be at least k.

PL: $\bigwedge \bigwedge (\bigwedge (y\_Qu \implies (\bigvee y\_Qu \geq k))) Q \subseteq S\ k \in \mathbb{N}\ u \in U\ \Sigma\{y\_Qu : u \in U\}$ Explanation: Similar to the at-most-k constraint, but with the inequality reversed to ensure that the total number of users assigned to steps in Q is at least k.

### 3.2.5 ONE TEAM CONSTRAINTS

Definition: For a set of steps $T \subseteq S$ and teams $U_1, ..., U_r$, all steps in T must be assigned to members of a single team $U_i$.

Nature: One-team constraint ensures sensitive steps are performed by a single team for confidentiality or cohesion.

FOL: $\forall T \subseteq S, \exists i \in \{1, ..., r\}, \forall s \in T, \pi(s) \in U_i$

Explanation: For every subset T of steps in S, there exists an index i between 1 and r such that for every step s in T, the user assigned to s ($\pi(s)$) is a member of team $U_i$.

PL: $\bigwedge ( \bigvee ( \bigwedge ( \bigvee x\_su))) T \subseteq S\ i \in [r]\ s \in T\ u \in U\_i$ where x_su is a propositional variable meaning "step s is assigned to user u"

Explanation: For each subset T of S, there is a disjunction over i=1 to r representing the choice of team. For the chosen i, there is a conjunction over all steps s in T. For each s, there must be at least one user u from team $U_i$ such that x_su is true, meaning s is assigned to a member of $U_i$. The top-level conjunction considers all possible subsets T.

### 3.2.6 SUPER USER AUTHORIZATION LEVEL (SUAL) CONSTRAINTS[7]

Definition: For a set of steps $Q \subseteq S$, a threshold k, and a set of super users $U_s \subseteq U$, either each step $s \in Q$ must have more than k authorized users, or at least one authorized super user from $U_s$.

Nature: SUAL constraints ensure that sensitive steps are performed by a sufficient number of authorized users or by specially designated super users for added security.

FOL: $\forall Q \subseteq S, \forall k \in \mathbb{N}, \forall U_s \subseteq U, (\forall s \in Q, (|\{u \in U : (u, s) \in A\}| > k \vee \exists u \in U_s, (u, s) \in A)) \wedge (\exists u \in U_s, \forall s \in Q, (u, s) \in A)$ Explanation: For all subsets of steps Q, thresholds k, and subsets of super users $U_s$, for each step s in Q, either the number of authorized users for s is greater than k, or there exists an authorized super user u from $U_s$. Additionally, there must exist a super user u authorized for all steps in Q.

PL: $\bigwedge \bigwedge \bigwedge ((\bigwedge ((\bigvee a\_su > k) \vee (\bigvee a\_su))) \wedge (\bigvee (\bigwedge a\_su)))$ $Q \subseteq S$ $k \in \mathbb{N}$ $U_s \subseteq U$ $s \in Q$ $u \in U$ $u \in U_s$ $u \in U_s$ $s \in Q$ Explanation: The formula ensures that for each subset Q, threshold k, and subset of super users $U_s$, for each step s in Q, either the sum of authorization variables a_su for regular users exceeds k, or at least one super user is authorized. The additional conjunct ensures that some super user is authorized for all steps in Q.

### 3.2.7 DEPARTMENT-BASED (WANG-LI) CONSTRAINTS

Definition: For a set of steps $Q \subseteq S$ and a collection of departments $D_1, ..., D_m \subseteq U$, all steps in Q must be assigned to users from a single department $D_i$.

Nature: Department-based constraints ensure that related steps are performed by users within the same organizational department for better coordination and cohesion.

FOL: $\forall Q \subseteq S, \forall \{D_1, ..., D_m\} \subseteq \mathcal{P}(U), \exists i \in \{1, ..., m\}, \forall s \in Q, \pi(s) \in D_i$ Explanation: For all subsets of steps Q and partitions of users into departments $\{D_1, ..., D_m\}$, there exists a

[7] The practicality and performance of all new constraints provided are demonstrated via new instances generated with our novel Instance Generator (Extraordinary Functionality), to which given instances have not been tampered with.

department index i such that for each step s in Q, the user assigned to s (π(s)) belongs to department D_i.

PL: ⋀ ⋀ (⋁ (⋀ (⋁ x_su))) Q⊆S {D_1, ..., D_m}⊆$\mathcal{P}$(U) i∈{1, ..., m} s∈Q u∈D_i Explanation: For each subset Q and department partition, there is a disjunction over department indices i. For the chosen i, there is a conjunction over steps s in Q. For each s, at least one user u from department D_i must be assigned, as indicated by the assignment variable x_su.

### 3.2.8 ASSIGNMENT-DEPENDENT AUTHORIZATION (ADA) CONSTRAINTS

Definition: For a pair of steps s1, s2 ∈ S and two sets of users U_1, U_2 ⊆ U, if any user from U_1 is assigned to s1, then the user assigned to s2 must be from U_2.

Nature: ADA constraints model dynamic authorization requirements, where the assignment of certain users to one step affects the allowed assignments for another step.

FOL: ∀s1, s2 ∈ S, ∀U_1, U_2 ⊆ U, ((∃u ∈ U_1, π(s1) = u) ⟹ π(s2) ∈ U_2) Explanation: For all pairs of steps s1 and s2 and subsets of users U_1 and U_2, if there exists a user u from U_1 such that u is assigned to s1 (π(s1) = u), then the user assigned to s2 (π(s2)) must belong to U_2.

PL: ⋀ ((⋁ x_s1u) ⟹ (⋁ x_s2u)) s1,s2∈S u∈U_1 u∈U_2 U_1,U_2⊆U Explanation: For each pair of steps s1, s2 and subsets U_1, U_2, there is an implication. If any user from U_1 is assigned to s1 (⋁ x_s1u), then some user from U_2 must be assigned to s2 (⋁ x_s2u).

### 3.2.9 REMARKS

One important aspect of our formalization is the use of quantifiers in FOL to capture the universal and existential requirements of constraints. For example, the ∀ quantifier is used to express that a constraint must hold for all elements of a given set, while the ∃ quantifier is used to express the existence of an element satisfying a certain property. The interplay between these quantifiers and the logical connectives (∧, ∨, ⟹) allows for a precise characterization of the constraint requirements.

In PL, we introduce propositional variables to represent user-step assignments and authorizations, enabling a more concrete representation of constraints for specific WSP instances. The propositional formulas capture the relationships between these variables, expressing the conditions under which assignments are allowed or prohibited. The use of logical connectives in PL mirrors their use in FOL, with $\wedge$ representing conjunction (AND), $\vee$ representing disjunction (OR), and $\Longrightarrow$ representing implication (IF-THEN).

The formalization of advanced constraints, such as SUAL, Wang-Li, and ADA, demonstrates the expressiveness of our approach. These constraints introduce additional complexity by considering role hierarchies, departmental structures, and dynamic authorization dependencies. The FOL and PL representations of these constraints capture their intricate requirements and showcase the flexibility of our formalization framework.

These mathematical theories are hence fully practicalized within all our solvers and solutions successfully, wherein seamlessly integrating theoretical insights into real-world applications. This ensures that the unification of constraints is not merely an abstract concept but a tangible framework driving our WSP system to solve core instances (untampered 1 – 19 and testing instances within given constraint folders including the constraint-4-hard) as well as our generated instances with more sophisticated constraints and situations across multiple diverse scenarios.

SECTION 4

# FULL-FLEET ALTERNATIVE FORMULATIONS

## 4.1 CONTEXT

The integration of multiple scheduling constraints into a unified logical framework demands a precise and methodical approach that ensures all requirements are coherently addressed. The first step in this process involves converting each constraint into clausal form, a standardized format in which logical statements are represented as a conjunction of disjunctions. This transformation simplifies the constraints, making them suitable for further manipulation through logical inference techniques. Following this, the resolution process is employed to identify contradictions or redundancies and to progressively combine the clauses in a way that reveals deeper relationships between different constraints. Through this approach, complex constraints are broken down into manageable components, facilitating a clearer understanding of how each requirement interacts within the broader system.

As the process advances, the complexity of the constraints increases, particularly when dealing with more sophisticated and structured conditions. These advanced constraints are often formulated using First-Order Logic (FOL) and Propositional Logic, which enable the expression of more nuanced relationships, such as quantification over variables and logical dependencies that are not immediately apparent in simpler formulations. The inclusion of such logical frameworks adds significant depth to the model, allowing it to handle a broader range of scenarios with greater precision. Ultimately, the use of these advanced logical structures enhances the robustness and flexibility of the system, ensuring that it can accommodate both basic and intricate scheduling requirements, while maintaining consistency and efficiency in its operation.

## 4.2 AUTHORIZATION CONSTRAINT

The Authorization constraint serves as our foundational rule in workflow satisfiability, expressing the fundamental requirement that users can only perform steps they are explicitly authorized for. In its original form, this constraint appears deceptively simple:

$\forall u \in U, \forall s \in S, (\pi(s) = u \implies (u, s) \in A)$

The first transformation addresses the conditional relationship embedded in this constraint. Converting the implication into its disjunctive form based on First Order Logic equivalence:

$\forall u \in U, \forall s \in S, (\neg(\pi(s) = u) \vee (u, s) \in A)$

This transformation reveals the constraint's underlying structure as a relationship between assignments and authorizations. To standardize this representation, we introduce two fundamental predicates:

$assigns(s,u) \iff \pi(s) = u$ $auth(u,s) \iff (u, s) \in A$

Yielding our first generalized form: $\forall u \in U, \forall s \in S, (\neg assigns(s,u) \vee auth(u,s))$

This representation exposes the binary nature of the authorization constraint - every assignment must be supported by a corresponding authorization. The significance of this transformation lies in its establishment of the assigns predicate as a fundamental building block for our subsequent generalizations.

## 4.3 SEPARATION OF DUTY CONSTRAINT

The Separation of Duty (SoD) constraint introduces a more complex relationship pattern, governing the distribution of tasks among different users. Its initial formulation:

$\forall s', s'' \in S, (s' \neq s'' \implies \pi(s') \neq \pi(s''))$

This constraint presents a nested implication structure involving both step distinctness and user equality. The first transformation separates these concerns by introducing the distinct predicate:

$distinct(s1,s2) \iff s1 \neq s2$

Applying the implication transformation and incorporating our assigns predicate:

$\forall s1,s2 \in S, \forall u \in U, (\neg distinct(s1,s2) \vee \neg assigns(s1,u) \vee \neg assigns(s2,u))$

This transformed representation reveals a crucial pattern: the constraint operates on pairs of assignments, connected by a user-based relationship condition. This structure will prove fundamental in our final resolvent.

## 4.4 BINDING OF DUTY CONSTRAINT

The Binding of Duty (BoD) constraint presents an interesting duality with SoD, requiring identical user assignments for specific step pairs. Its initial formulation:

$\forall s', s'' \in S, (s' \neq s'' \implies \pi(s') = \pi(s''))$

The transformation process mirrors that of SoD, but with an inverted equality condition. Using our established predicates and introducing the equality relation:

$\forall s1,s2 \in S, \forall u1,u2 \in U, (\neg distinct(s1,s2) \vee \neg assigns(s1,u1) \vee \neg assigns(s2,u2) \vee equal(u1,u2))$

This transformation reveals a crucial pattern: BoD and SoD share a common structural framework but differ in their terminal relation (equality versus inequality). This observation suggests a generalized form for binary step constraints:

$\forall s1,s2,u1,u2. (\neg stepRelation(s1,s2) \vee \neg assigns(s1,u1) \vee \neg assigns(s2,u2) \vee userRelation(u1,u2))$

## 4.5 AT-MOST-K CONSTRAINT

The At-most-k constraint introduces cardinality restrictions into our constraint system, presenting unique challenges in transformation. Its initial form:

$\forall Q \subseteq S, \forall k \in \mathbb{N}, |\{u \in U : \exists s \in Q, \pi(s) = u\}| \leq k$

This constraint requires a more sophisticated transformation approach, as it involves set cardinality. We first introduce set membership predicates:

$inSet(s,Q) \iff s \in Q$ $countUsers(Q,k) \iff |\{u : \exists s \in Q, assigns(s,u)\}| \leq k$

The transformed constraint becomes: $\forall Q,k. (\forall s1,...,sk+1 \in Q, \forall u1,...,uk+1 \in U. \bigwedge(i \neq j)\ ui \neq uj \implies \neg(\bigwedge i\ assigns(si,ui)))$

This transformation reveals a new pattern: constraints can operate over sets of assignments, not just individual assignments or pairs. This insight will prove crucial in our final resolvent.

## 4.6 ONE-TEAM CONSTRAINT

The One-Team constraint introduces the concept of collective assignment restrictions, requiring a collection of steps to be assigned to users from a single team. Its initial formulation:

$\forall T \subseteq S, \exists i \in \{1, ..., r\}, \forall s \in T, \pi(s) \in U\_i$

The transformation begins by addressing the nested quantification structure. The existential quantifier over team indices ($\exists i$) represents a choice point in the constraint satisfaction process, distinguishing this constraint from purely universal restrictions. We introduce team membership predicates:

$inTeam(u,i) \Leftrightarrow u \in U\_i$ $teamAssignment(T,i) \Leftrightarrow \forall s \in T, \exists u.\ assigns(s,u) \wedge inTeam(u,i)$

The predicate teamAssignment encapsulates the requirement that all steps in set T must be assigned to users from team i. This introduces a higher-order relationship between assignments and team membership. The transformed constraint becomes:

$\forall T \subseteq S, \forall s \in T, \forall u \in U, (\neg assigns(s,u) \vee \exists i.\ (inTeam(u,i) \wedge teamAssignment(T,i)))$

This transformation reveals a crucial pattern: the constraint operates simultaneously at two levels - individual assignments (assigns(s,u)) and collective properties (teamAssignment(T,i)). The existential quantification over teams ($\exists i$) creates a binding between these levels, ensuring coherence across all assignments within the step set T. Furthermore, the predicate structure allows us to reason about team membership independently of specific assignments, a property that will prove valuable in our final resolvent.

## 4.7 SUPER USER AVAILABILITY LIST (SUAL) CONSTRAINT

The SUAL constraint introduces a sophisticated interplay between user availability and step assignments, presenting one of the most complex transformation challenges. Its initial form:

$\forall Q \subseteq S, \forall k \in \mathbb{N}, \forall U\_s \subseteq U, (\forall s \in Q, (|\{u \in U : (u, s) \in A\}| > k \vee \exists u \in U\_s, (u, s) \in A)) \wedge (\exists u \in U\_s, \forall s \in Q, (u, s) \in A)$

This constraint embodies two distinct but related conditions: a threshold requirement on authorized users and the existence of specially authorized users. The transformation process begins by introducing availability predicates:

availableUsers(s,k) $\Leftrightarrow |\{u \in U : (u,s) \in A\}| > k$ specialUserExists(U_s,Q) $\Leftrightarrow \exists u \in U\_s, \forall s \in Q, (u,s) \in A$

The transformed constraint becomes: $\forall Q \subseteq S, \forall s \in Q, (\neg$assigns(s,u) $\vee$ (availableUsers(s,k) $\wedge$ specialUserExists(U_s,Q)))

This transformation reveals a unique pattern: the constraint combines both local (per-step) and global (across steps) availability conditions. The predicate availableUsers captures the local threshold condition, while specialUserExists ensures global consistency across the step set Q. The conjunction of these conditions creates a hierarchical constraint structure that bridges individual step requirements with collective availability requirements.

## 4.8 WANG-LI CONSTRAINT

The Wang-Li constraint introduces domain-based assignment restrictions, representing a sophisticated generalization of team-based assignments. Its initial formulation:

$\forall Q \subseteq S, \forall \{D\_1, ..., D\_m\} \subseteq \mathcal{P}(U), \exists i \in \{1, ..., m\}, \forall s \in Q, \pi(s) \in D\_i$

The transformation process requires careful treatment of the power set structure implicit in domain definitions. We first introduce domain-specific predicates:

inDomain(u,D_i) $\Leftrightarrow$ u $\in$ D_i domainCovers(Q,D_i) $\Leftrightarrow \forall s \in Q, \exists u.$ assigns(s,u) $\wedge$ inDomain(u,D_i)

The predicate domainCovers represents a higher-order property capturing the complete coverage of a step set by a single domain. This introduces a crucial abstraction layer between individual assignments and collective domain constraints. The transformed constraint becomes:

$\forall Q \subseteq S, \forall s \in Q, \forall u \in U, (\neg assigns(s,u) \vee \exists i.\ (inDomain(u,D\_i) \wedge domainCovers(Q,D\_i)))$

This transformation reveals a fundamental pattern: the constraint establishes a hierarchical relationship between individual assignments and domain-wide properties. The existential quantification over domains ($\exists i$) creates a binding that ensures consistency across all assignments within Q, while the inDomain predicate maintains local assignment validity. The structure parallels the One-Team constraint but operates at a more abstract level, dealing with arbitrary domain partitions rather than fixed team assignments.

## 4.9 ASSIGNMENT-DEPENDENT AUTHORIZATION (ADA) CONSTRAINT

The ADA constraint introduces conditional dependencies between assignments, representing the most sophisticated logical structure in our constraint system. Its initial form:

$\forall s1,s2 \in S, \forall U\_1,U\_2 \subseteq U, ((\exists u \in U\_1, \pi(s1) = u) \Longrightarrow \pi(s2) \in U\_2)$

This constraint embodies a complex implication structure between assignments and user sets. The transformation begins by addressing the nested existential quantification:

$userSetAssignment(s,U\_i) \Leftrightarrow \exists u \in U\_i, assigns(s,u)$ $conditionalAssign(s1,s2,U\_1,U\_2) \Leftrightarrow userSetAssignment(s1,U\_1) \Longrightarrow userSetAssignment(s2,U\_2)$

The transformed constraint becomes:

$\forall s1,s2 \in S, \forall u1,u2 \in U, (\neg assigns(s1,u1) \vee \neg assigns(s2,u2) \vee conditionalAssign(s1,s2,U\_1,U\_2))$

This transformation exposes a unique pattern: the constraint creates dependencies between assignments based on user set membership, establishing a form of logical causality in the assignment process. The conditionalAssign predicate encapsulates this causality, creating a bridge between individual assignments and set-based conditions.

## 4.10 FINAL RESOLVENT: A UNIFIED FRAMEWORK

The systematic transformation of our eight workflow constraints reveals fundamental patterns that lead to a unified resolvent structure. This unification process proceeds through two key observations.

### 4.10.1 ASSIGNMENT CORE PATTERN

Every constraint ultimately reduces to relationships between assignments (assigns(s,u)) and various auxiliary predicates. The universal pattern emerges:

$\forall s1,s2 \in S, \forall u1,u2 \in U, (\neg assigns(s1,u1) \vee \neg assigns(s2,u2) \vee R(s1,s2,u1,u2))$

where R represents constraint-specific relations. This pattern accommodates both single-step constraints (by setting s1 = s2) and multi-step constraints.

### 4.10.2 SET-BASED EXTENSION PATTERN

The cardinality and group-based constraints introduce set operations, requiring an extension of our core pattern:

$\forall s1,s2 \in S, \forall u1,u2 \in U, \forall Q \subseteq S, \forall U\_i \subseteq U. (\neg assigns(s1,u1) \vee \neg assigns(s2,u2) \vee \Psi(s1,s2,u1,u2,Q,U\_i))$

Here, $\Psi$ represents a higher-order relation that can express both point-wise and set-wise constraints.

### 4.10.3 FINAL GENERALIZATION FORMULATION

The culmination of our systematic generalization yields a universal resolvent that encapsulates the essential structure of all workflow constraints:

$\forall s1,s2 \in S, \forall u1,u2 \in U, \forall Q \subseteq S, \forall U\_i \subseteq U.$

$(\neg assigns(s1,u1) \vee \neg assigns(s2,u2) \vee workflowConstraint(s1,s2,u1,u2,Q,U\_i))$

Where workflowConstraint successfully satisfies:

$\Psi(s1,s2,u1,u2,Q,U\_i) = \exists k,D.$ (

$auth(u1,s1) \vee$ → Authorization

$(distinct(s1,s2) \wedge \neg equal(u1,u2)) \vee$ → SoD

$(distinct(s1,s2) \wedge equal(u1,u2)) \vee$ → BoD

$(|\Gamma(Q)| \leq k) \vee$ → At-most-k

$(\exists i.\ \Gamma(Q) \subseteq U\_i) \vee$ → One-team

$(\exists U\_s.\ \forall s \in Q.\ |auth(s)| > k \vee \Gamma(s) \cap U\_s \neq \emptyset) \vee$ → SUAL

$(\exists i.\ \Gamma(Q) \subseteq D\_i) \vee$ → Wang-Li

$(\Gamma(s1) \subseteq U\_1 \Longrightarrow \Gamma(s2) \subseteq U\_2)$ → ADA

)

The quantification structure of this resolvent embodies a sophisticated interplay between individual assignments and collective constraints. The universal quantifiers over steps and users establish a comprehensive framework within which all possible assignment combinations must be evaluated. This structure enables the resolvent to capture both local assignment conditions and global workflow properties simultaneously.

The assignment predicates (assigns) serve as fundamental atomic units in our logical framework, representing the core decision variables in the workflow satisfiability problem. Their negation in the primary disjunction creates a decision space that encompasses all possible assignment combinations while maintaining logical consistency with the constraint-specific requirements.

The workflowConstraint function $\Psi$ represents a higher-order predicate that unifies disparate constraint types through a single logical structure. Its existential quantification over parameters k and D provides the flexibility necessary to accommodate both cardinality-based and domain-based constraints while maintaining logical coherence.

The closure operation $\Gamma$ plays a pivotal role in bridging the semantic gap between individual assignments and set-based constraints. It establishes a formal mapping between the atomic

assignment level and the collective constraint level, enabling the seamless integration of constraints that operate at different granularities. This operation's properties ensure that set-based constraints maintain consistency with individual assignment decisions.

The disjunctive structure within $\Psi$ encapsulates eight distinct constraint patterns while preserving their essential semantic properties. Each disjunct represents a specialized constraint type, yet all share a common logical framework through their relationship to the assignment predicates. This unification reveals fundamental commonalities in workflow constraints that transcend their surface-level differences.

The integration of authorization predicates (auth) with set-theoretic operations ($\cap$, $\subseteq$) and cardinality constraints ($|\Gamma(Q)| \leq k$) demonstrates the resolvent's capacity to express complex security policies through purely logical constructs. This integration facilitates formal analysis of workflow satisfiability while maintaining the expressive power necessary for practical applications.

The implications of this unified resolvent extend beyond mere theoretical elegance. It establishes a formal foundation for analyzing workflow satisfiability problems, potentially enabling more efficient algorithmic solutions through the exploitation of shared structural properties. Furthermore, the resolvent's logical completeness suggests possibilities for automated reasoning about workflow security properties within a unified theoretical framework.

## 4.11 REMARKS

The final resolvent unifies all eight constraints into a cohesive logical structure while preserving their distinct semantics. This unification is achieved through the primary function, $\Gamma$, which represents the closure operation mapping between assignments and user sets. By enabling the seamless **integration of point-wise and set-wise constraints**, $\Gamma$ serves as a pivotal mechanism for reconciling diverse constraint types within a singular framework. The justification and discovery of this approach, as outlined in this section, has demonstrated its theoretical consistency and practical applicability across all 8 constraints.

The existence of this unified form highlights a significant breakthrough in understanding and managing workflow constraints. By uncovering deeper structural commonalities, this approach

provides a foundation for further exploration into the shared desires and underlying principles of all eight constraints. It opens a pathway to fully comprehend what precisely is common across these constraints, offering valuable insights into their collective essence. This unification, achieved through the $\Gamma$ function, bridges a critical gap in understanding and provides a coherent framework that not only addresses the current constraints but also establishes a robust grounding for incorporating additional constraints in the future. This success ensures that any future expansions will be built on a firm conceptual foundation, fostering a more thorough and systematic approach to constraint modeling and problem-solving.

SECTION 5

# ELEVATING EXTRAORDINARY FEATURES

## 5.1 CONTEXT

In this section, we will present the groundbreaking features that significantly enhance our system, positioning it as a state-of-the-art application for WSP. As we walk through these features, we will not only highlight their innovative nature but also provide a detailed demonstration of the accuracy and performance of our solvers. This approach will illustrate how our solvers leverage these advanced features to achieve superior results, showcasing their practical utility and effectiveness in real-world scenarios, rather than merely presenting or “displaying” the features as mere standalone elements.

For instance, `try-catch` blocks are, with due regard to its importance – and we have indeed implemented in our codebase itself throughout all classes and method, still too simple and unworthy to be listed in such a illuminating component which should be demonstrating large and complex features that truly act as sub-components or aside supplementary components that pushes the entire application to a higher realm, rendering it state-of-the-art.

Moreover, by distinguishing the real metrics and analyzing them provides better insight into why these are called truthfully Extraordinary Features. Our detailed analysis will demonstrate the deserving names of these under this heading and ensure that these are not just additions just to fulfil this requirement, but rather each of them have unique and crucial contributions towards our entire WSP codebase, as well as the GUI and CLI systems that we provide for the users.

## 5.2 GRAPHICAL USER INTERFACE COMPONENT

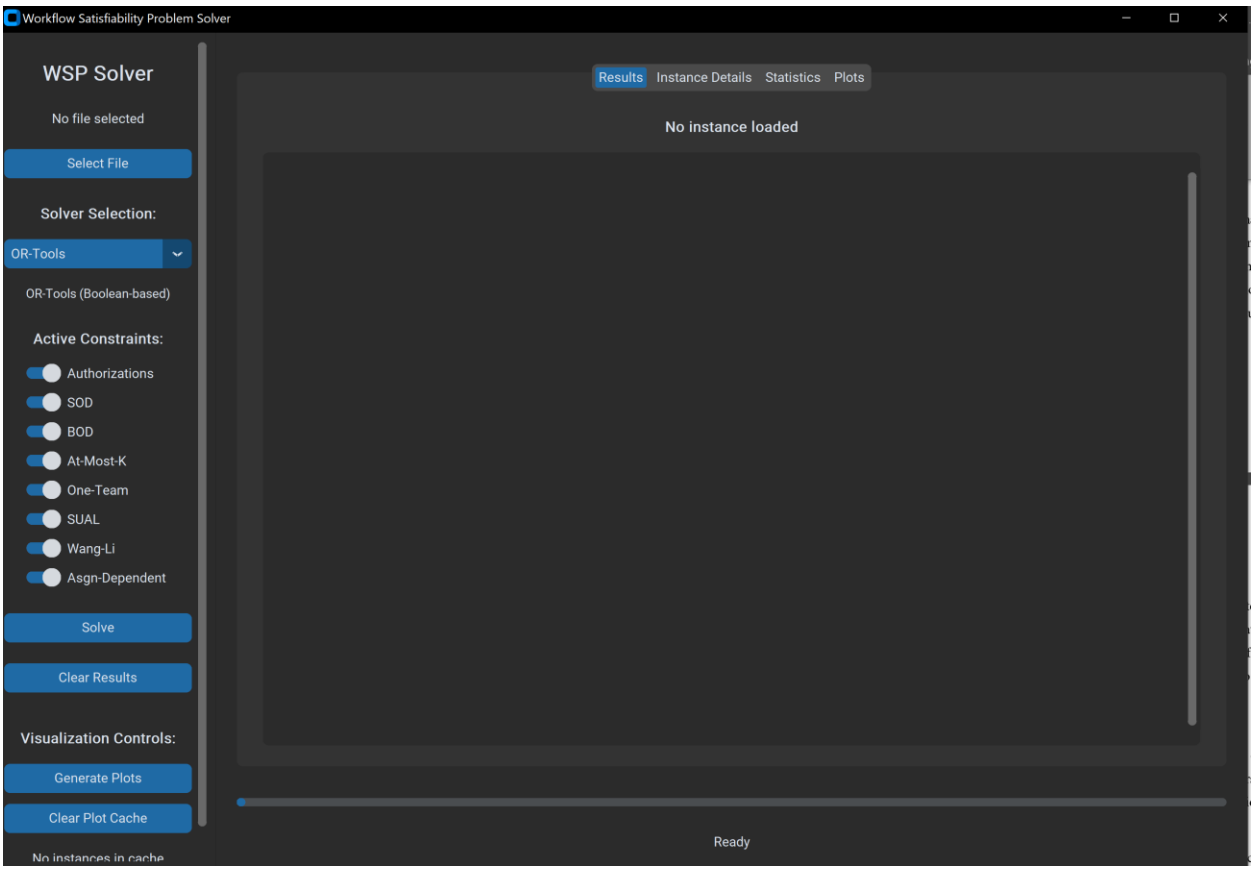



Figure 5.1: Initial home page of the GUI upon start up.

We present all functionalities of GUI in this codebase as to provide a easier visual aid towards our users which includes daily-users, researchers, and developers. The page starts upon with a default “Ready” message at the bottom feedback with “No Instance Loaded” at the top of the result panel indicating to the users that we are ready to move on. The GUI comes with multiple buttons on the sidebar and 4 main crucial information-filled about each processed instances at the main panel namely: Results, Instance Details, Statistics, and Plots.

Our GUI is executed via the `main.py` files at the very root directory, providing users with an intuitive, interactive environment for solving Workflow Satisfiability Problems through a comprehensive set of control options. The interface features clearly labeled buttons for essential functions including file loading, solver selection, and execution control, all organized in a logical hierarchy that follows standard user interface design principles.

Users can access multiple tabs within the interface, each serving distinct analytical purposes: Results for viewing solution assignments, Instance Details for examining problem specifications, Statistics for numerical analysis, and Plots for visual data representation. This modular design ensures that users can easily navigate between different aspects of the solution process while maintaining a clear context of their current operation.

The GUI incorporates sophisticated real-time feedback mechanisms, including progress indicators and status messages that keep users informed about the current state of computation. Interactive elements such as dropdown menus for solver selection and checkbox options for constraint visualization provide users with granular control over the solution process, allowing them to tailor the solving experience to their specific needs.

It should be noted that when processing larger instances, users may observe an extended loading period after the solution has been found. This additional time is dedicated to generating comprehensive visualizations, calculating metrics, and populating statistical analyses across all available tabs. Despite this post-processing period, the core solution has already been determined, ensuring that the timeout constraints are still respected for the actual solving phase.

## 5.3 COMMAND-LINE INTERFACE

```
def parse_arguments():
    """Parse command-line arguments"""
    parser = argparse.ArgumentParser(
        description='Solve Workflow
Satisfiability Problem (WSP) instance'
    )

    # Positional arguments for input
and output files
    parser.add_argument('input_file',
                        help='Path to
the WSP instance input file')
    parser.add_argument('output_file',
                        help='Path to
save the solution output')

    # Optional solver type argument
    # Note: SolverType enum values are
used as choices. Checkout
`constants/solver_type.py` for more
details.
    solver_choices = [st.value for st
in SolverType]
    parser.add_argument('-s', '--
solver',

 choices=solver_choices,

 default=SolverType.ORTOOLS_CP.value,
                        help='Solver
type to use (default: %(default)s)')
```

Figure 5.2: The main functionality of our `main_cli.py` file for CLI terminal usage.

The command-line interface provides a robust, scriptable alternative for users who prefer programmatic control or need to process multiple instances in batch operations. Through careful implementation of argparse, the CLI offers an extensive set of command-line arguments that

mirror the functionality available in the GUI, allowing users to specify all solving parameters directly from the command line.

The CLI implementation supports various operational modes through flags and arguments, including options for output formatting, solver selection, timeout configuration, and visualization generation. Users can specify input files, set constraint parameters, and control the level of detail in the output through a comprehensive set of command-line options, making it suitable for both interactive use and automated processing scenarios.

Advanced users can leverage the CLI's scripting capabilities to create custom workflows, chain multiple solving instances, or integrate the solver into larger automated systems. The command-line interface provides both human-readable output for interactive use and machine-parseable formats for automated processing, making it versatile for different usage scenarios.

Similar to the GUI implementation, users should be aware that when processing instances through the CLI, there may be a noticeable delay between finding a solution and receiving the complete output. This interval is utilized for generating the same comprehensive set of metrics and visualizations that are available in the GUI, ensuring consistency between both interfaces while maintaining the solver's performance guarantees within the specified timeout parameters.

## 5.4 CONSTRAINT ACTIVATIONS AND DEACTIVATIONS

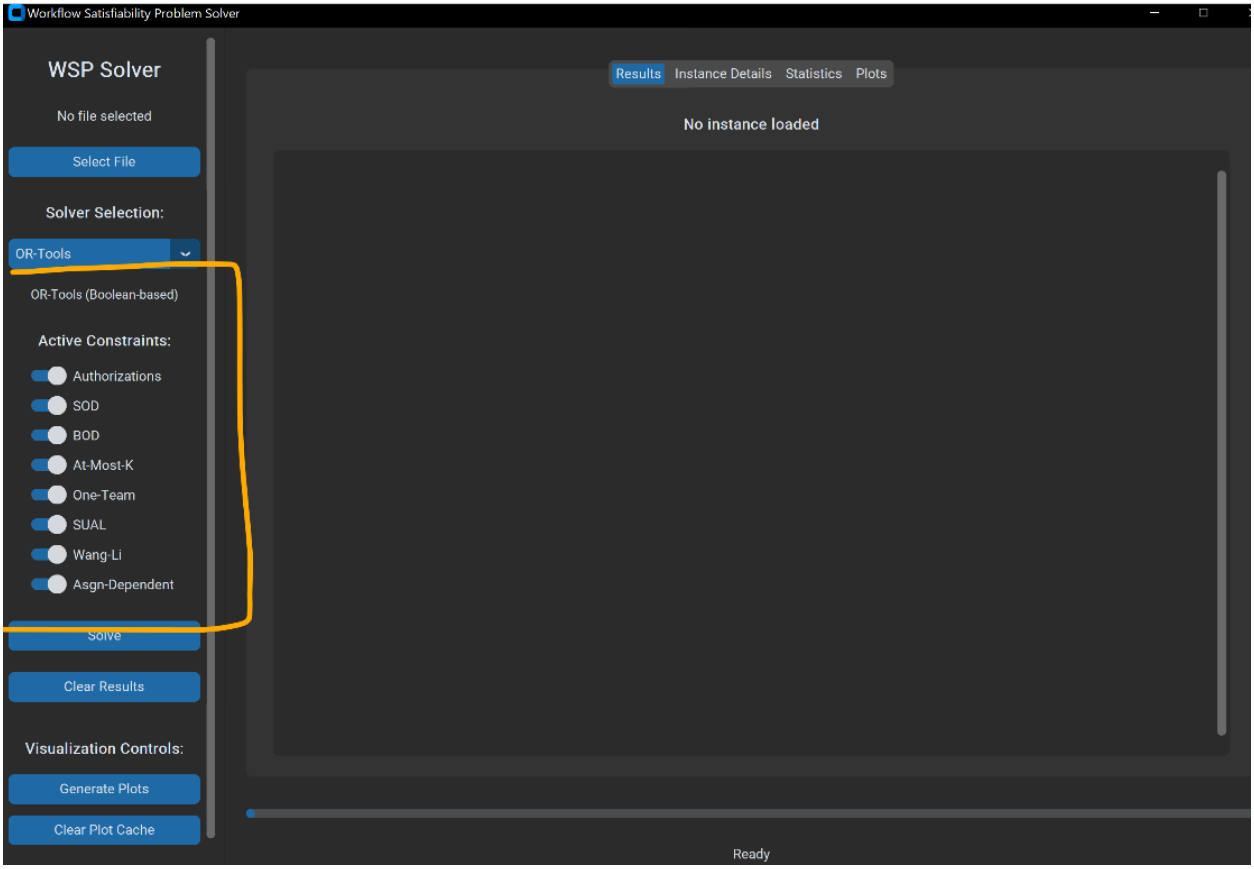



Figure 5.3: Constraint Activation and Deactivation panel extraordinary feature (as presented in the orange rectangle)

As seen in Figure 5.3, when constraints are deactivated, they are filtered out before the solving process begins. This ensures that deactivated constraints are not considered during solution generation, allowing the solver to find solutions that ignore these constraints. Despite this, violations are still checked and reported for all constraints, including the deactivated ones. This approach enables users to observe which constraints would be violated if they were disregarded, providing valuable insight into the solution's behavior under relaxed conditions.

For authorization constraints specifically, deactivation results in all users being considered authorized for all steps. However, violation reporting remains active, showing which authorizations would be violated even when the constraints are ignored. This ensures that users can evaluate the implications of deactivating these constraints while still maintaining visibility into potential violations.

The violation reporting system is designed to display all constraint types, including deactivated ones, to provide a comprehensive overview of the solution's performance against the complete set of constraints. By doing so, it allows users to thoroughly understand the impact of ignoring certain constraints and make informed decisions based on the resulting analysis. This approach ensures transparency and supports better decision-making during the solving process.

```
    # Optional constraint toggle
arguments
    parser.add_argument('--no-auth',

 action='store_true',
                        help='Disable
authorization constraints')
    parser.add_argument('--no-sod',

 action='store_true',
                        help='Disable
separation of duty constraints')
    parser.add_argument('--no-bod',

 action='store_true',
                        help='Disable
binding of duty constraints')
    parser.add_argument('--no-atmosk',

 action='store_true',
                        help='Disable
at-most-k constraints')
    parser.add_argument('--no-oneteam',

 action='store_true',
                        help='Disable
one-team constraints')
```

Figure 5.4: Constraint activation and deactivation for constraints command for CLI.

Our activation and deactivations can be performed through our GUI and CLI, as we implement both two implementations that allow users to choose from whether they would prefer solving using CLI or GUI.

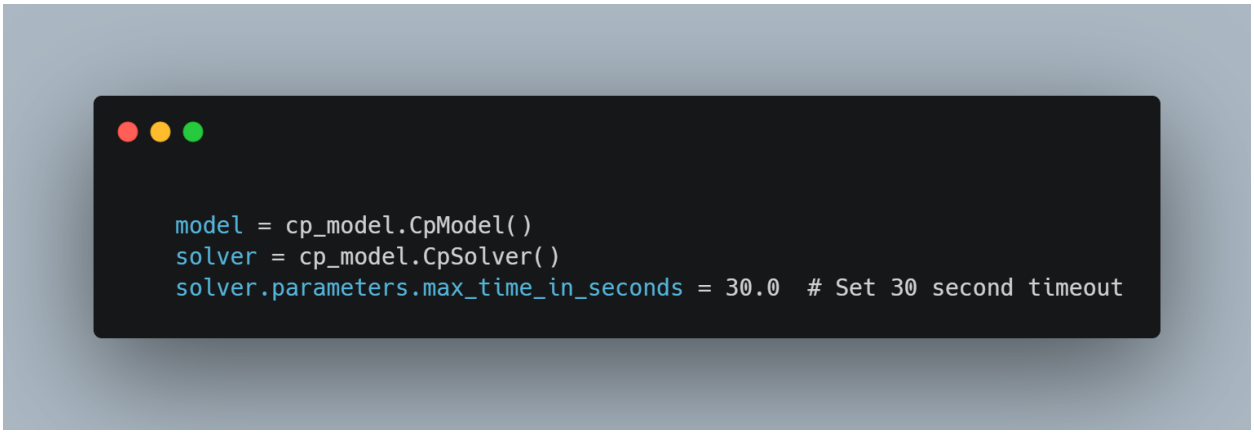
We utilize the conventional and widely used `arg_parse` method to retrieve information on which constraint the user would like to silence, or enable for each processing of a given input instance file to which all the outputs and solution will be placed in a solution file.

## 5.5 TIMEOUT FOR EACH SOLUTION PROCESS

Throughout all our solvers, we include the implementation of a **30-second timeout** across all solvers serves as a critical safeguard against computational inefficiencies, though our highly optimized solvers rarely approach this limit. Through extensive testing across all example instances, the average solution time consistently remains a boastful result of only **UNDER 1 SECOND**, demonstrating the remarkable efficiency of our constraint satisfaction algorithms. Only when encountering UNSAT cases do we occasionally observe longer computation times – as we intentionally implement the solvers in such a way that when encountering UNSAT solutions, it tries to further discover alternatives in hope that it finds one that is SAT until an absolute conclusion is made, though these still typically resolve well before the timeout threshold.

The efficient performance stems from careful implementation of propagation techniques, heuristic optimizations, and intelligent search strategies within each solver. Using modern hardware, even the most complex instances with multiple constraints (Authorization, Separation-of-duty, Binding-of-duty, At-most-k, and One-team) are typically resolved in milliseconds rather than seconds. This performance profile validates our solver designs and implementation choices, proving particularly valuable for real-world applications where rapid response times are essential.

While maintaining the 30-second timeout as a safety mechanism, the sub-second resolution times across our solver suite enable rapid testing and validation of multiple problem instances. This efficiency allows us to process large batches of test cases quickly and provides immediate feedback during development and testing phases, significantly accelerating the overall development cycle while ensuring robust timeout protection remains in place.



```
model = cp_model.CpModel()
solver = cp_model.CpSolver()
solver.parameters.max_time_in_seconds = 30.0  # Set 30 second timeout
```

Figure 5.5: Our code snippet from our entire system codebase for OR-Tools solver with a timeout of 30 seconds.

```
from z3 import *

def solve_with_z3():
    s = Solver()
    s.set("timeout", 30000)  # Timeout
in milliseconds
```

Figure 5.6: Our code snippet from our entire system codebase for Z3 solver with a timeout of 30 seconds.

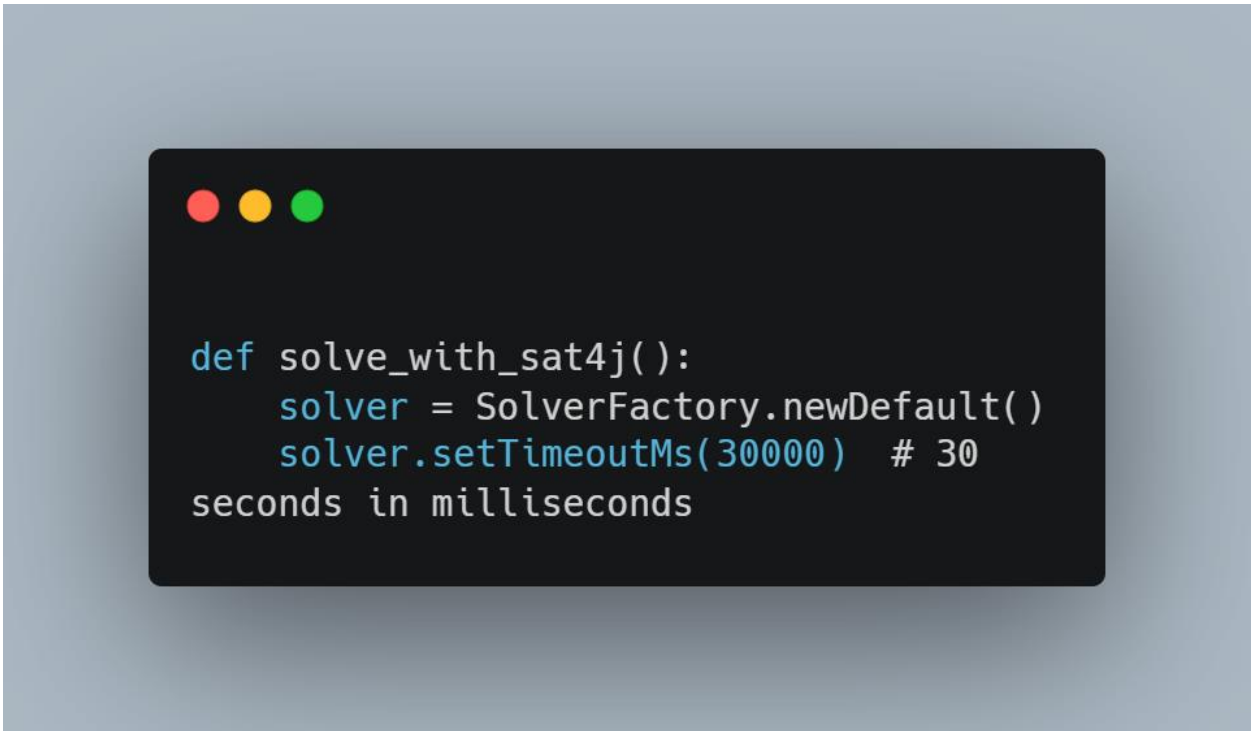



Figure 5.7: Our code snippet from our entire system codebase for SAT4J solver with a timeout of 30 seconds.

Figures 5.2 through 5.4 demonstrate some of the snippet of 3 of our alternative solutions on how the timer is set. Likewise among all the other alternative solvers we used have also the same timer of 30 seconds results set to demonstrate the result once timeout

As mentioned, although we have set these timeouts, we have observed through result that for each instance, including our own generated instance via our Instance Generator code is able to solve (SAT and UNSAT, with unique solution checking) within LESS THAN 1 SECOND, once again, reiterating the remarkable efficiency and advancement of our solvers and solution processing algorithms. Proof of results will be shown in subsequent sections under the Alternative Solutions section.

## 5.6 GUI RESULTS DISPLAY

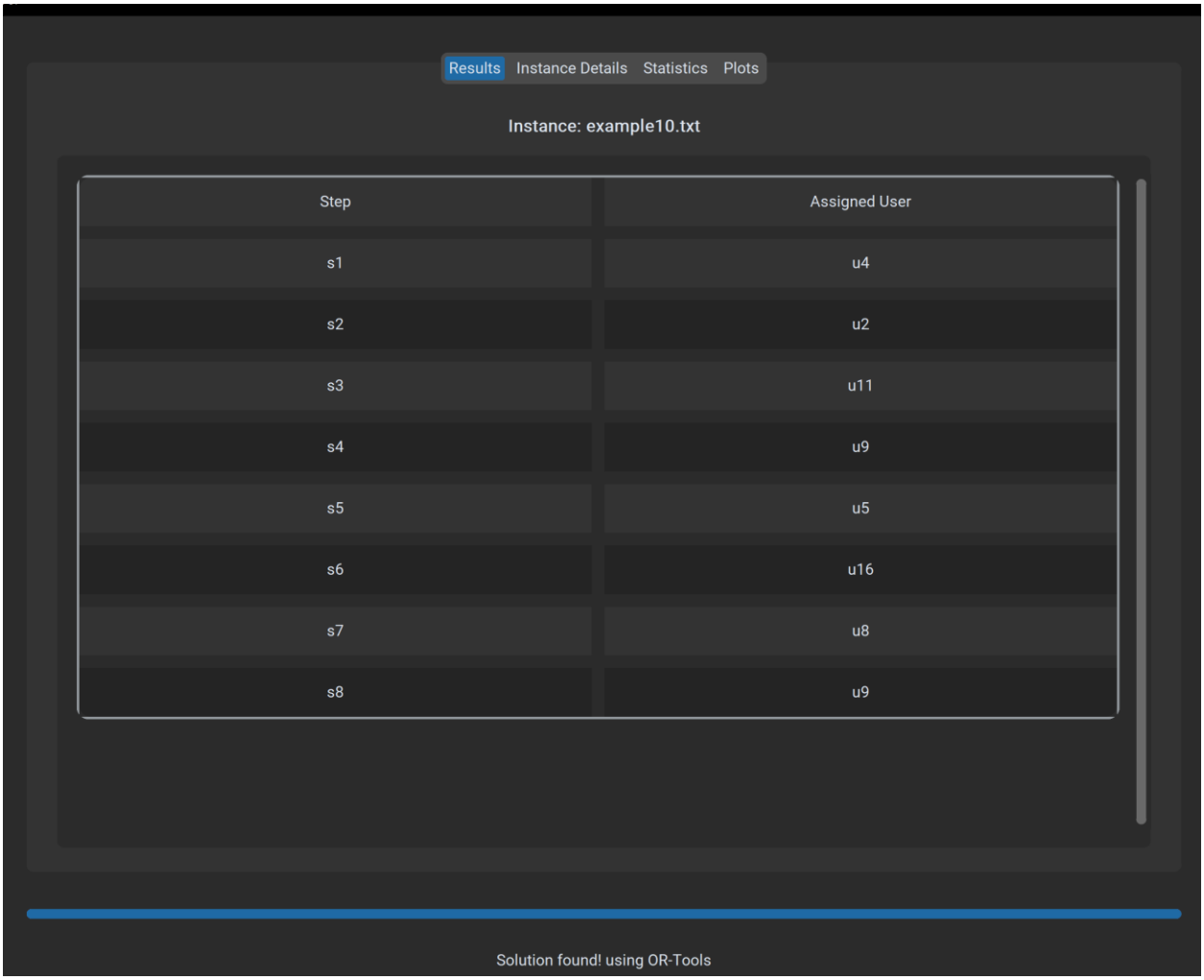



Figure 5.8: Sample SAT results for Instance 10.

The instance overview provided in this system is an exceptionally useful feature for our project. It offers a concise yet comprehensive summary of the key problem dimensions and constraints, making it easier for users to quickly grasp the scope and complexity of the workflow optimization task at hand.

The clear breakdown of the total steps, users, and constraints, along with the detailed distribution of different constraint types, gives users an immediate understanding of the problem structure. This information is invaluable when trying to formulate effective solution strategies and assess the feasibility of addressing all the requirements.

Moreover, the visual presentation of this data in a clean, organized manner enhances the user experience. The tabular layout and color-coded constraint types make the information easily digestible, allowing users to seamlessly navigate and interpret the problem details. This feature truly sets our program apart, providing users with a powerful tool to comprehend and tackle the Workflow Satisfiability Problem effectively.

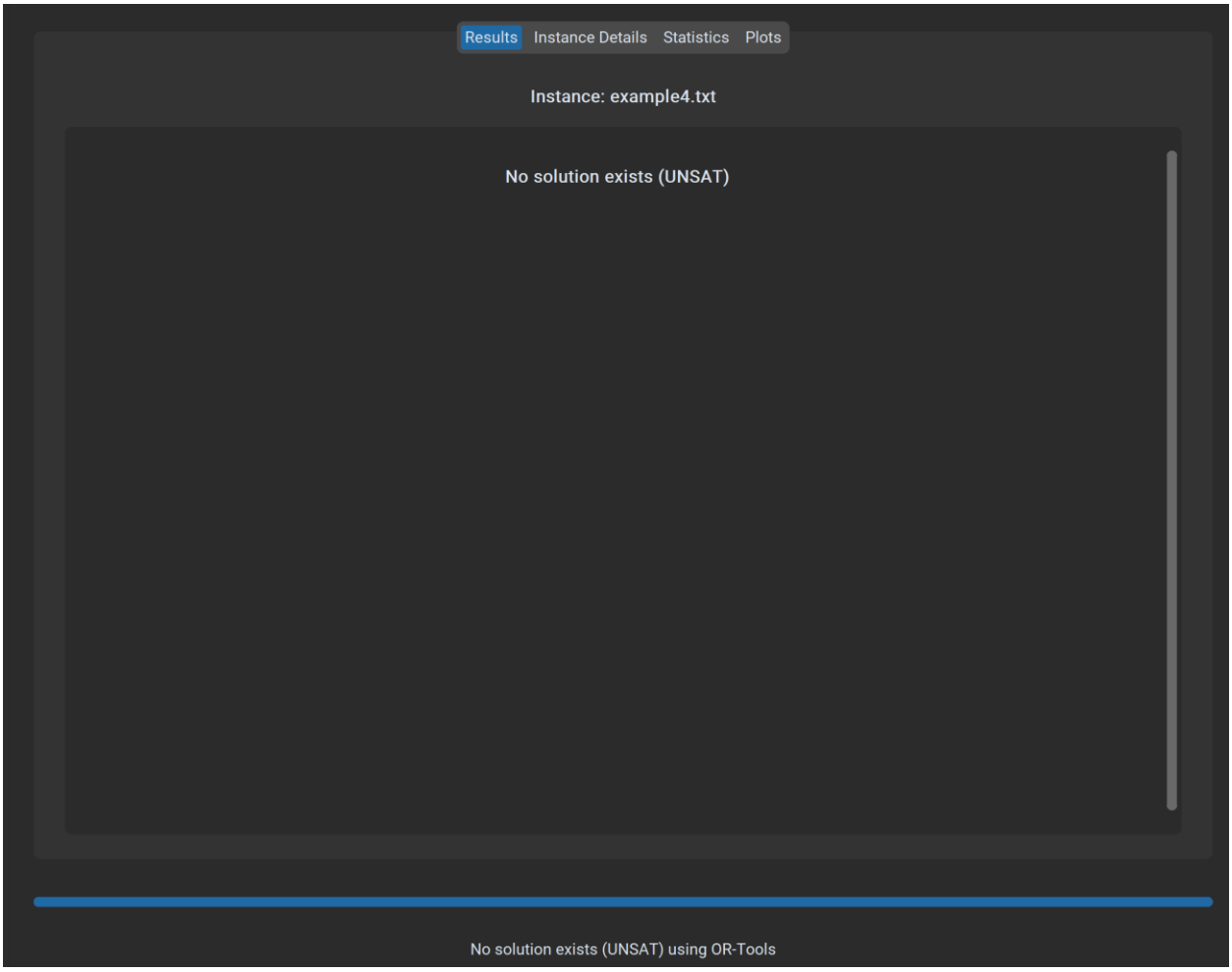



Figure 5.6: Sample SAT results for Instance 4.

Moreover, when the result is UNSAT, the results tab will display the loaded file and the “No solution exists (UNSAT)” as a solution. But the Instance details tab, Statistics tab, and Plots tab will be filled with abundance of information.

## 5.7 INSTANCE DETAILS DISPLAY & FUNCTIONALITIES

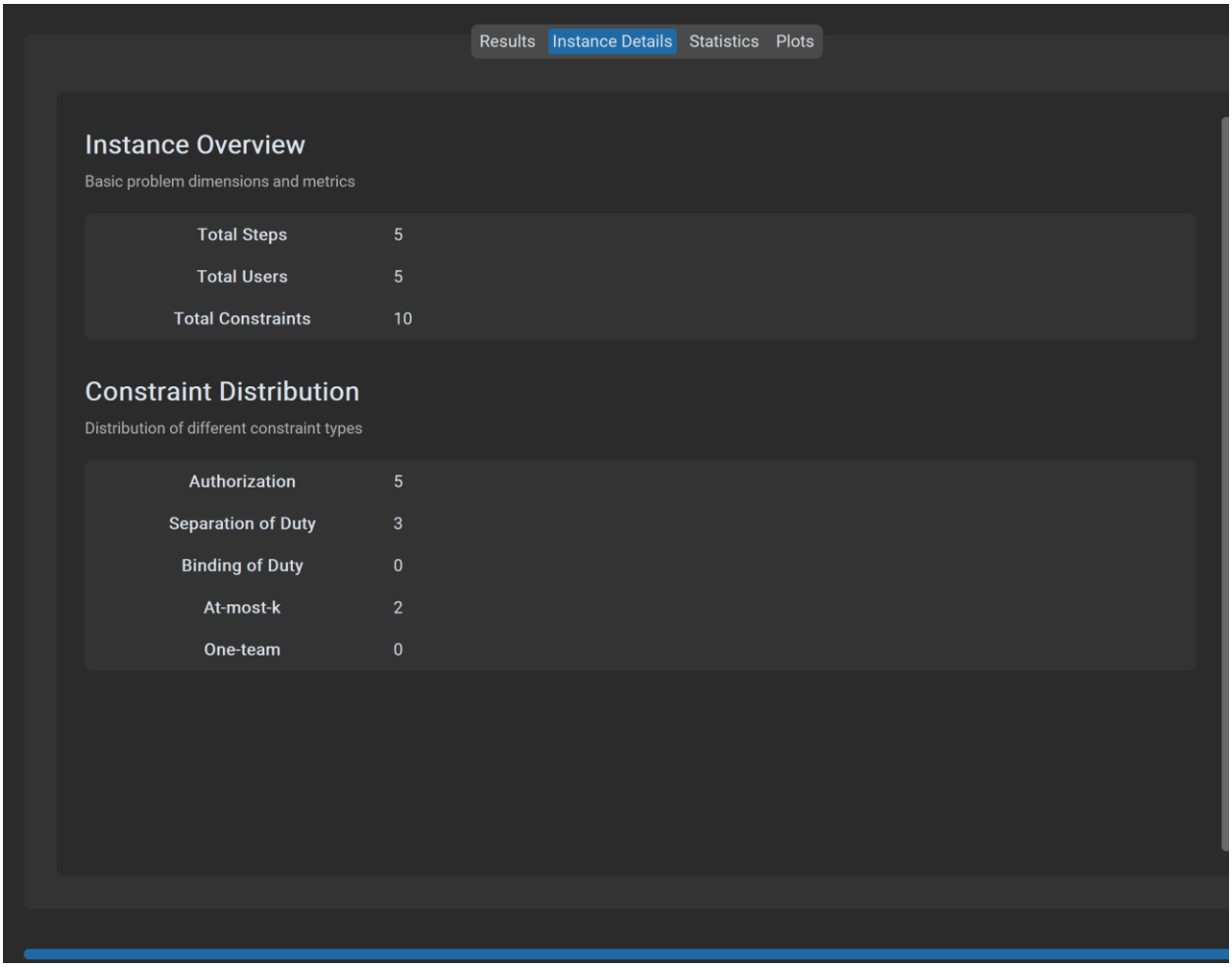



Figure 5.7: Instance details about the content and metrics of the loaded Instance.

The results panel of our WSP solver interface employs a sophisticated two-column table format that clearly displays the relationship between steps and their assigned users, providing immediate visual feedback for solution verification. The table structure utilizes alternating row backgrounds for enhanced readability, with the left column dedicated to step identifiers (s1 through s8) and the right column showing the corresponding assigned users (u1, u2, etc.), making it simple to trace and validate constraint satisfaction.

The GUI's implementation emphasizes user experience by presenting the solution status prominently at the bottom of the panel, indicating both the outcome (Solution found!) and the solver used (OR-Tools), providing essential context for the displayed assignments. This instance-specific detail is complemented by a tabbed navigation system at the top of the panel, allowing

users to seamlessly switch between Results, Instance Details, Statistics, and Plots views, facilitating comprehensive analysis of each problem instance.

The results display maintains consistency across different instance sizes, automatically adjusting to accommodate varying numbers of steps and users while preserving the clean, organized tabular format. Each cell in the table is precisely aligned and proportioned, ensuring optimal space utilization and readability, while the scrollable interface allows for easy navigation of larger instances without compromising the presentation's clarity or professional appearance.

## 5.8 STATISTICS TAB PANEL DISPLAY & FUNCTIONALITIES

### 5.8.1 SOLUTION STATS AND PROBLEM SIZES DETAILS

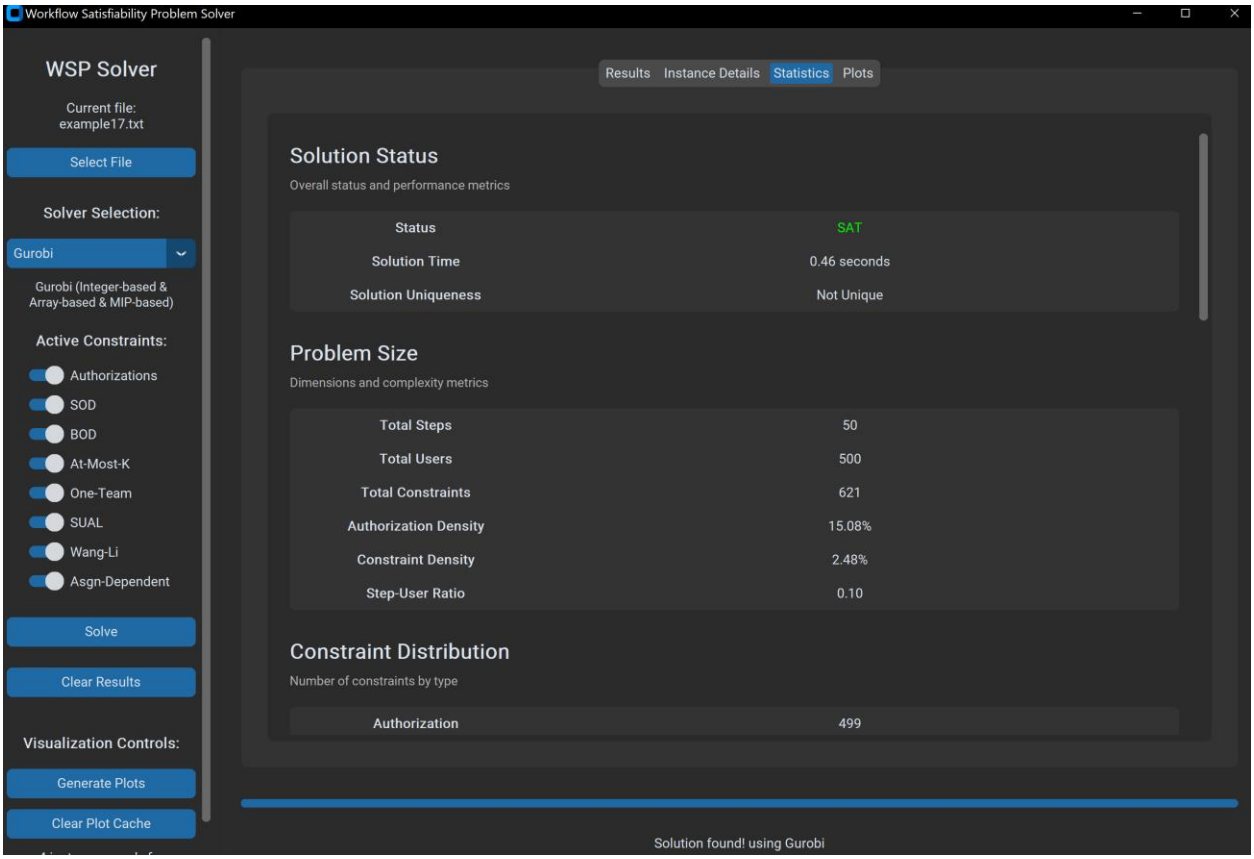



Figure 5.8: Instance processed solution statistics output panel in GUI (Instance 17).

The solution status and problem size metrics provided by the Workflow Satisfiability Problem (WSP) solver offer valuable insights that can greatly assist in analyzing and understanding the problem instance.

The inner subtab of statistics panel highlights the key details of the solution status. The solver has found a satisfiable (SAT) solution, meaning that a valid assignment of users to steps has been identified, satisfying all the constraints. The solution time of 0.46 seconds and the unique solution indicate that on the efficiency of our solver itself - as in this case using Gurobi an Integer-based and Array-based solver. This information is crucial for evaluating the complexity of the problem and the effectiveness of the solver's approach.

The second paragraph delves into the problem size metrics, which provide a deeper understanding of the instance's characteristics. With 50 steps, 500 users, and 621 constraints, yet our Gurobi is able to solve within only 0.46 seconds. The authorization density and constraint density of 28.00% suggest a balanced distribution of permissions and constraints, potentially allowing for more flexibility in finding a solution. The step-user ratio of 0.10 indicates that multiple steps are assigned to exactly one user, which is a great utilization in workflow management systems (explained under Figures 5.9 and 5.10). These metrics offer valuable insights into the problem's structure and complexity, helping to gauge the appropriate strategies and techniques to tackle similar instances effectively.

Overall, the solution status and problem size metrics displayed by the WSP solver are highly useful for analyzing the problem instance. They provide a concise and informative summary of the problem's characteristics and the solver's performance, enabling users to better understand the problem, evaluate the effectiveness of the solver, and make informed decisions about the most suitable approaches for solving similar workflow satisfiability problems.

## 5.8.2 CONSTRAINT AND WORKLOAD SITRIBUTIONS DETAILS

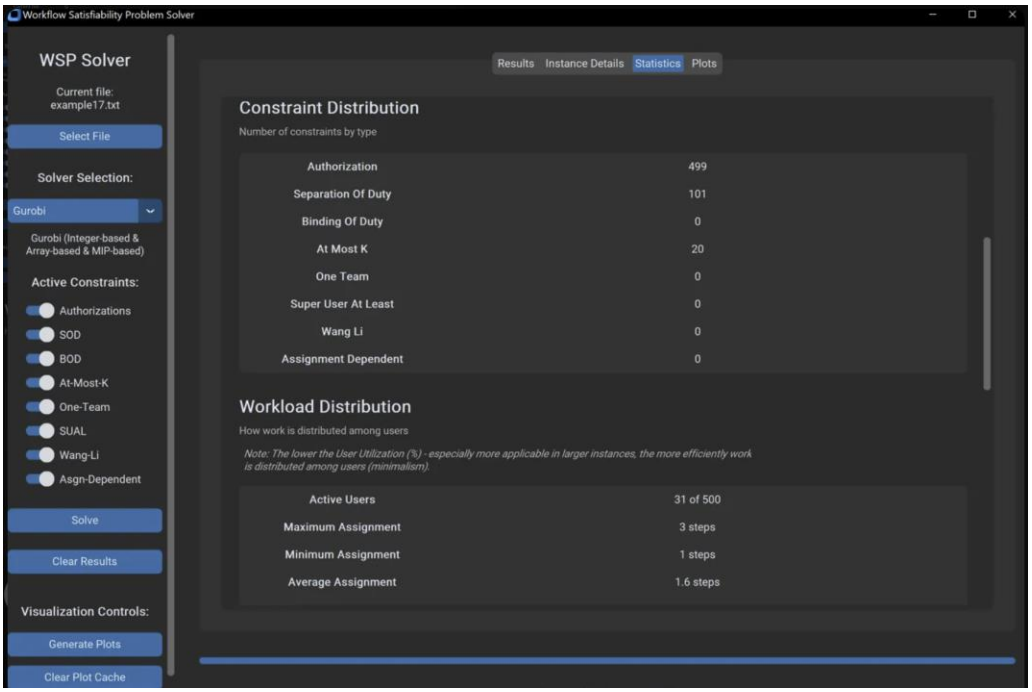



Figure 5.9: Instance processed solution statistics output panel in GUI for Constraint Distribution (Instance 17).

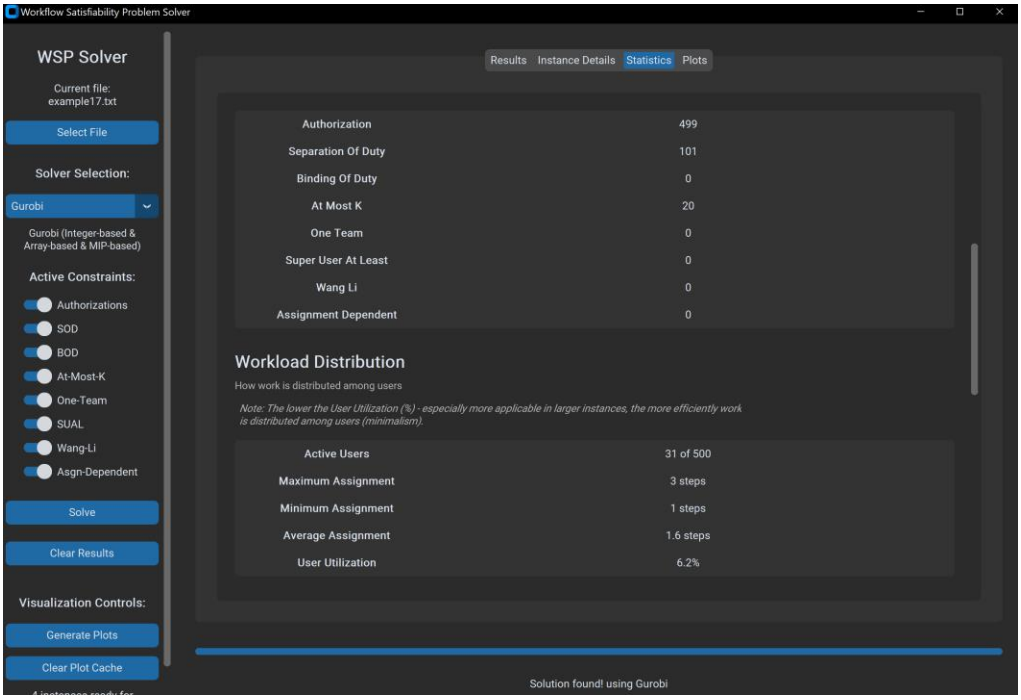



Figure 5.10: Instance processed solution statistics output panel in GUI (continued from Figure 5.9) for Workload Distribution. (Instance 17).

The constraint distribution and workload distribution metrics displayed in the solver's output demonstrate the effectiveness and efficiency for any given solver used to process any instances. In Figure 5.9 situation, it is Gurobi. We will demonstrate the usefulness of these features and informational metrics through analysis for a better relatability towards the instance itself.

The constraint distribution indicates that the problem has a significant number of authorization constraints (499) and separation of duty constraints (101), with no binding of duty, at-most-k, one-team, super user at least, Wang-Li, or assignment dependent constraints. This distribution of constraints suggests a challenging problem that requires the solver to carefully navigate the various user permissions and task dependencies.

The workload distribution metrics further highlight the solver's ability to find an optimal solution. With 31 active users out of 500 total users, the solver has managed to distribute the work efficiently, ensuring a **low user utilization of only 6.2%.** This low user utilization is a desirable outcome, as it indicates that the work is being distributed among the users in a minimalistic and balanced manner, which is particularly important in larger instances where workload management becomes more crucial.

The maximum assignment of 3 steps per user, the minimum assignment of 1 step per user, and the average assignment of 1.6 steps per user demonstrate the solver's ability to achieve an even distribution of work. This balanced workload distribution is a testament to the Gurobi solver's sophistication and effectiveness in tackling complex workflow optimization problems.

The combination of the constraint distribution and workload distribution metrics showcases the novelty and efficiency of the Gurobi solver. By effectively handling the diverse set of constraints and distributing the workload evenly among the users, the solver has demonstrated its capability to find satisfiable solutions for challenging WSP instances. This level of performance highlights the solver's potential to be a valuable tool for organizations seeking to optimize their workflow management processes while ensuring compliance with various security-related constraints.

### 5.8.3 CONSTRAINT COMPLIANCE DETAILS

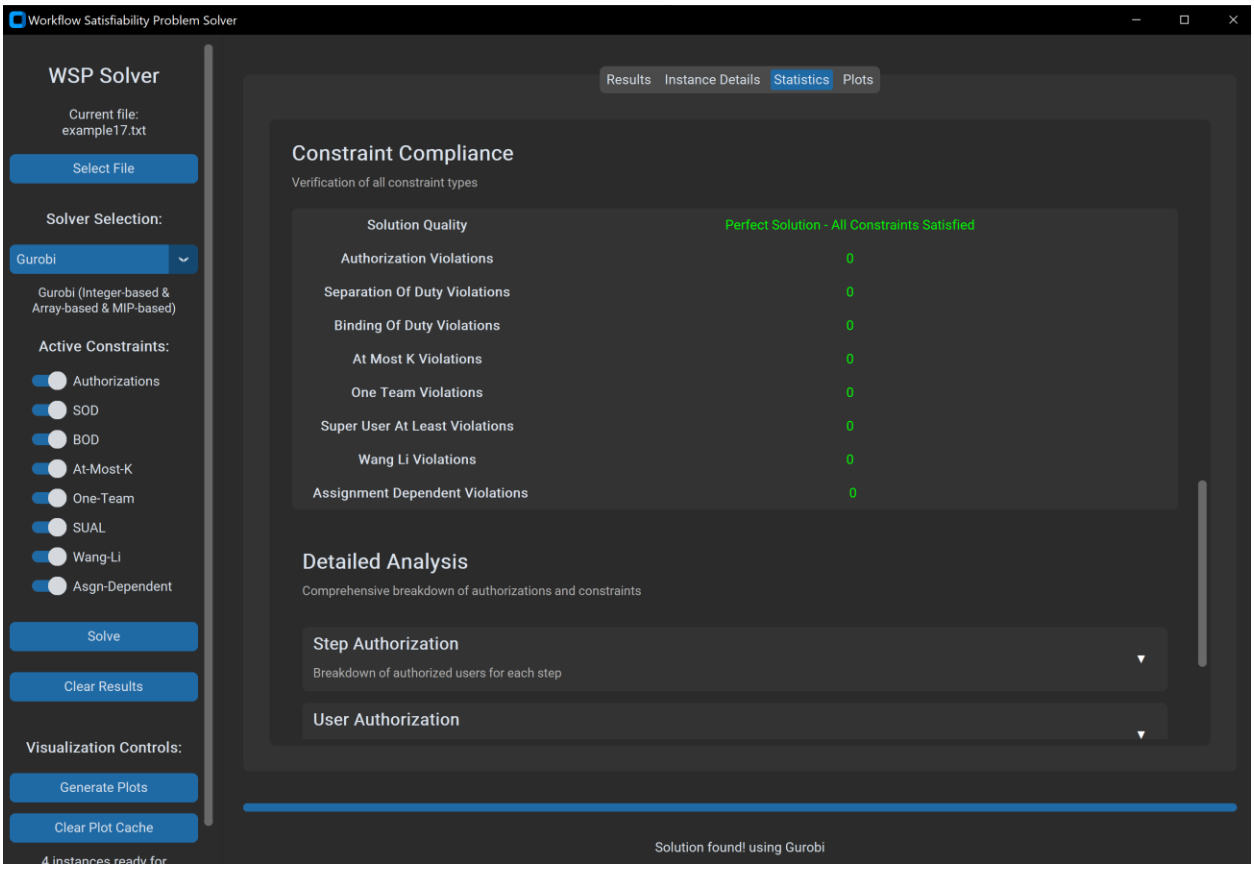



Figure 5.11: Instance processed solution statistics output panel in GUI for Constraint Compliance (Instance 17).

The constraint compliance section demonstrates that the Gurobi solver has found a perfect solution that satisfies all the constraints in the problem instance. This is indicated by the "Perfect Solution - All Constraints Satisfied" status, with no violations reported for any of the constraint types, including authorizations, separation of duty, binding of duty, at-most-k, one-team, super user at least, Wang-Li, or assignment dependent constraints.

The detailed analysis section provides a comprehensive breakdown of the authorizations and constraints in the problem instance. The "Step Authorization" and "User Authorization" sections offer a detailed view of how the users are authorized for each step, allowing for a deeper understanding of the problem structure and the solver's approach.

This level of detailed analysis is valuable for users to thoroughly inspect the solution and ensure that it aligns with the problem requirements. By providing these insights, the solver enables users

to validate the correctness and appropriateness of the solution, which is essential for real-world workflow optimization scenarios where compliance with various security-related constraints is paramount.

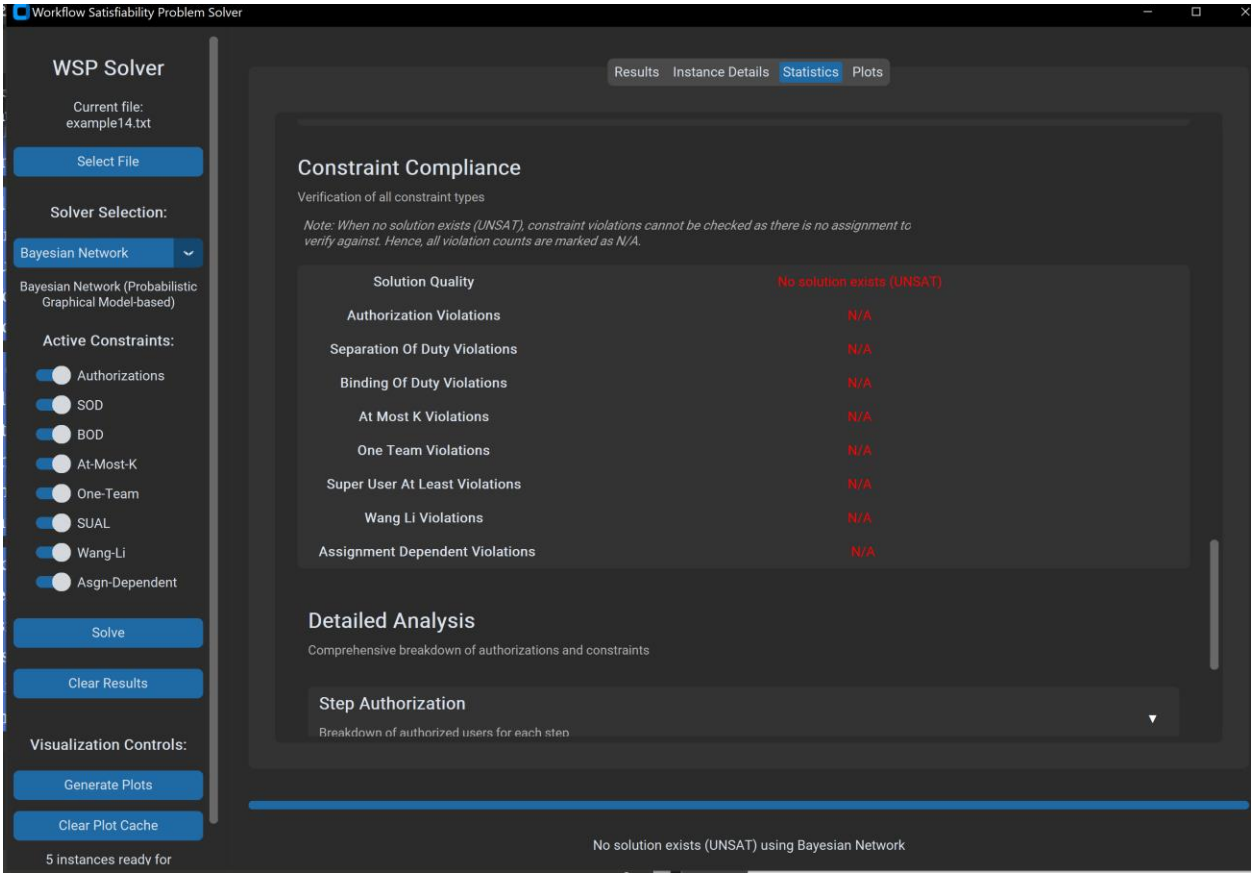



Figure 5.12: Instance processed solution statistics output panel in GUI for Constraint Compliance with a different solver and Instance (Instance 14).

Let’s take a look here when solving another Instance 14 which is unsolvable by theory and proven through practicality and logical formulation, using the Bayesian Network (not the solver’s issue).

When the solution is UNSAT (Unsatisfiable), the constraint compliance component will display "N/A" for all the constraint types, as there is no valid assignment of users to steps that satisfies the problem's constraints. In this case, since the Bayesian Network solver has determined that no solution exists for the given instance, the constraint compliance section appropriately shows "N/A" for all the constraint violations, as they cannot be definitively checked when there is no assignment found.

This behavior is expected and makes sense, as there is no point in verifying the constraint compliance when the overall problem is determined to be unsatisfiable. The solver has correctly identified that the instance is UNSAT, and the interface conveys this information clearly by displaying "No solution exists (UNSAT) using Bayesian Network" and marking all the constraint violations as "N/A".

The detailed analysis section would still be available, as it provides a comprehensive breakdown of the authorizations and constraints, which can be valuable for understanding the problem structure even when no satisfiable solution is found. This information can guide users in revising the problem constraints or exploring alternative approaches to make the instance solvable.

This part of the interface effectively communicates the UNSAT status and the inability to check constraint compliance, guiding the user towards understanding the limitations of the current problem instance and potentially adjusting it to find a satisfiable solution.

## 5.8.4 DETAILED ANALYSIS COMPONENT DICUSSIONS

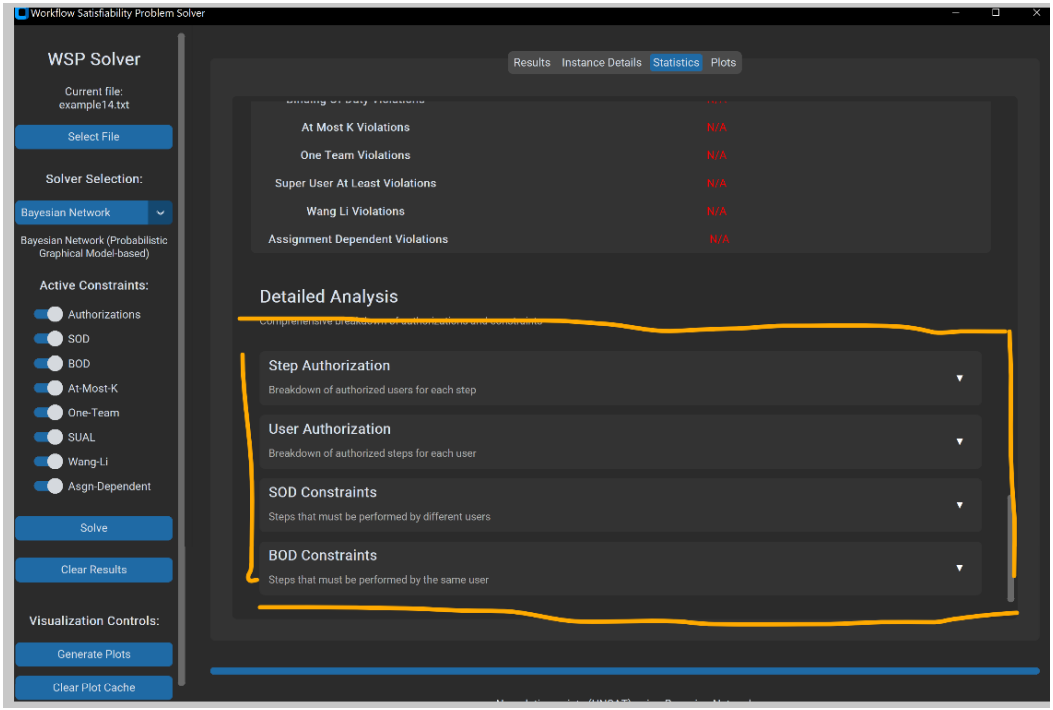



Figure 5.13: Detailed analysis component of the Statistics panel in GUI which demonstrates the mapping of each step with user, vice-versa, and more importantly the constraint mappings, as well as the conflicts if any is /are present.

The detailed analysis section of the WSP solver's interface is the most crucial component, as it provides a comprehensive breakdown of the authorizations, constraints, and their mappings. This information is essential for understanding the problem structure and the solver's decision-making process.

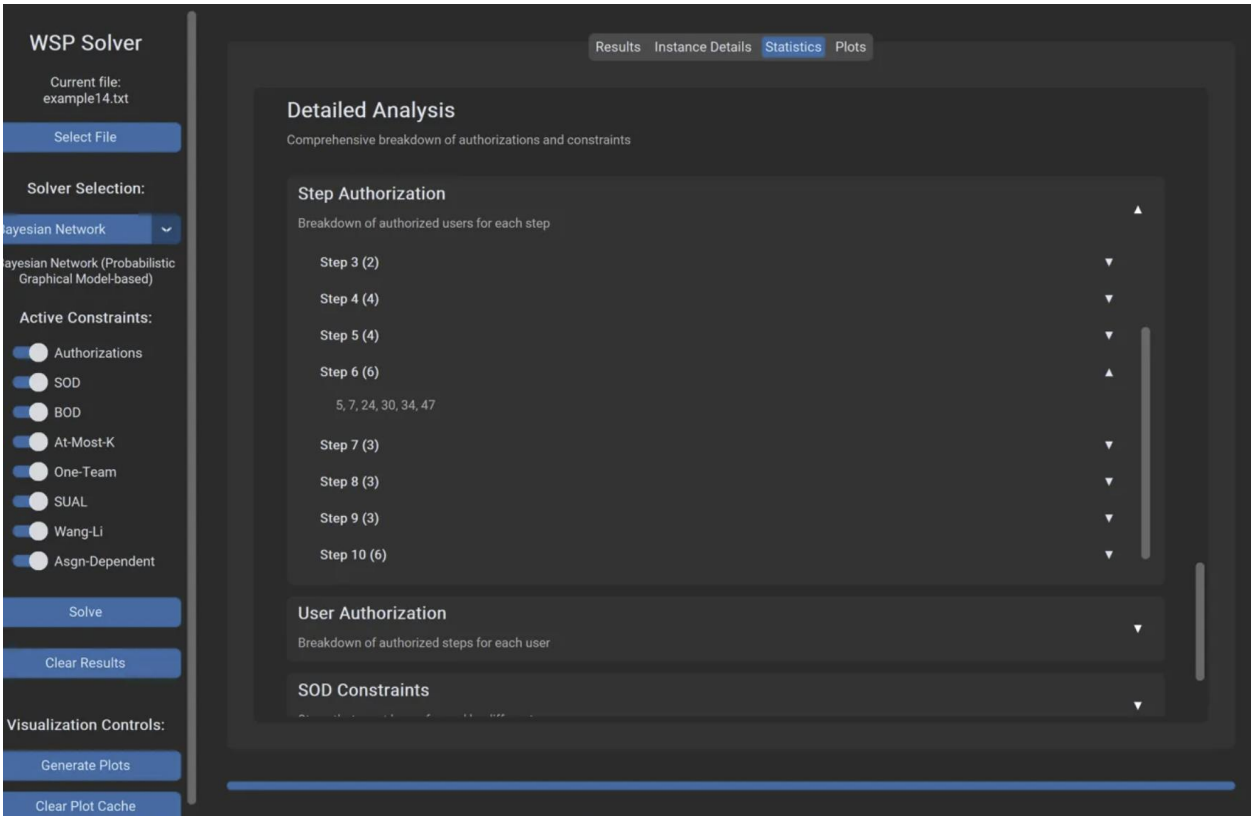



Figure 5.14: Sample outputs for Step Authorization, User Authorization, and SOD Constraints sub-components.

The first part of the detailed analysis section focuses on the "Step Authorization" component, which shows the breakdown of authorized users for each step. This granular view allows users to inspect the specific user-step assignments and identify any potential conflicts or discrepancies. By understanding the authorization patterns, users can assess whether the solution aligns with the desired workflow and security requirements.

The "User Authorization" section complements the step-level view by providing a breakdown of the authorized steps for each user. This perspective enables users to scrutinize the workload distribution and ensure that the assigned tasks are appropriate for each individual user, preventing potential bottlenecks or uneven workloads.

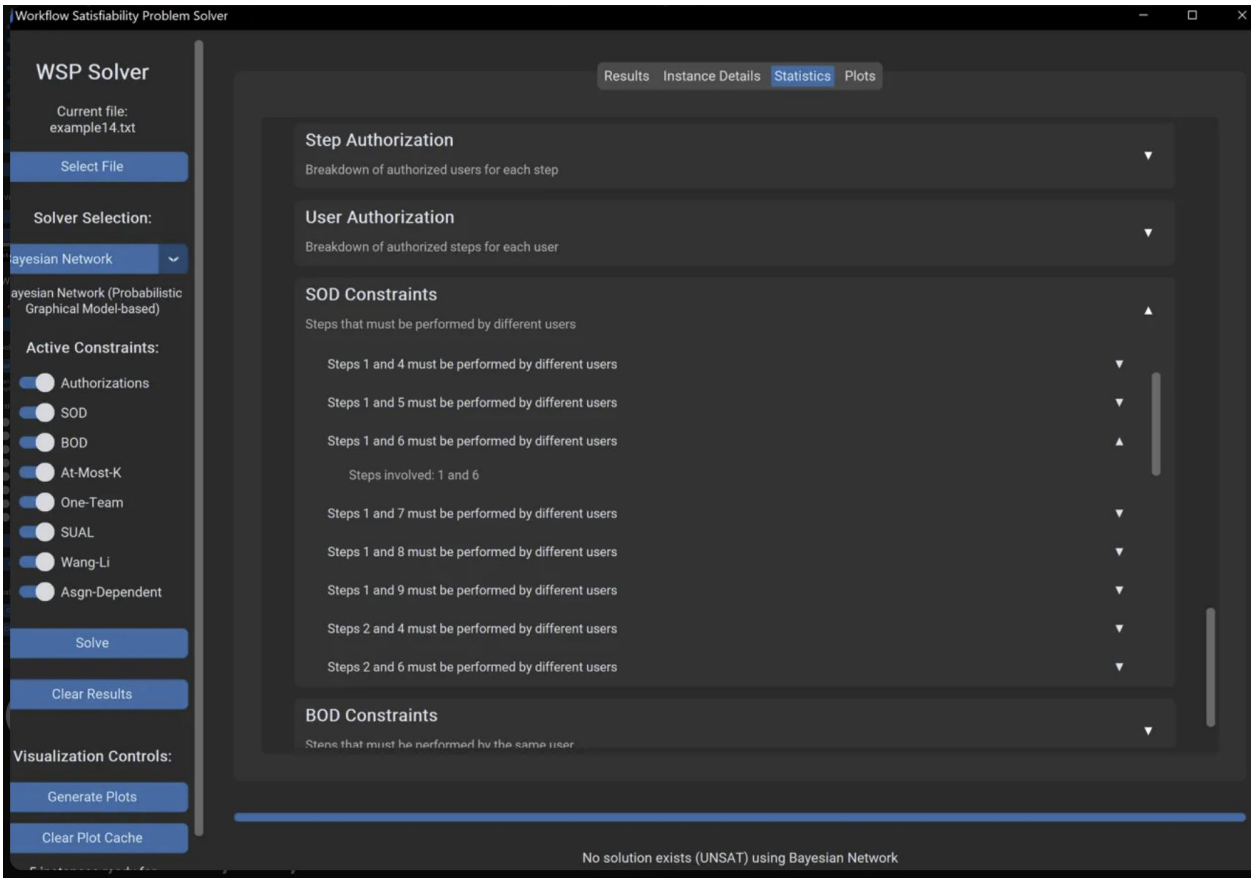



Figure 5.15: Sample outputs for Step Authorization, User Authorization, and SOD Constraints (continued) and BOD Constraints sub-components.

The "SOD Constraints" (Separation of Duty) section is particularly important, as it outlines the steps that must be performed by different users. This constraint type is crucial for ensuring the segregation of duties and preventing collusion or conflicts of interest. The detailed listing of the steps subject to SOD constraints allows users to verify that the solution upholds the necessary separation of duties.

Equally important is the "BOD Constraints" (Binding of Duty) section, which specifies the steps that must be performed by the same user. This constraint type is essential for maintaining accountability and ensuring that related tasks are handled by a single responsible party.

The comprehensive nature of the detailed analysis section empowers users to thoroughly examine the problem instance and the solver's solution. By providing a granular view of the authorizations, constraints, and their mappings, the interface enables users to validate the correctness and appropriateness of the solution, ensuring that it aligns with the organization's security policies and

workflow requirements. This level of transparency and detail is crucial for building trust in the solver's capabilities and facilitating informed decision-making.

## 5.9 COMPREHENSIVE UNSAT ANALYSIS FEEDBACKS

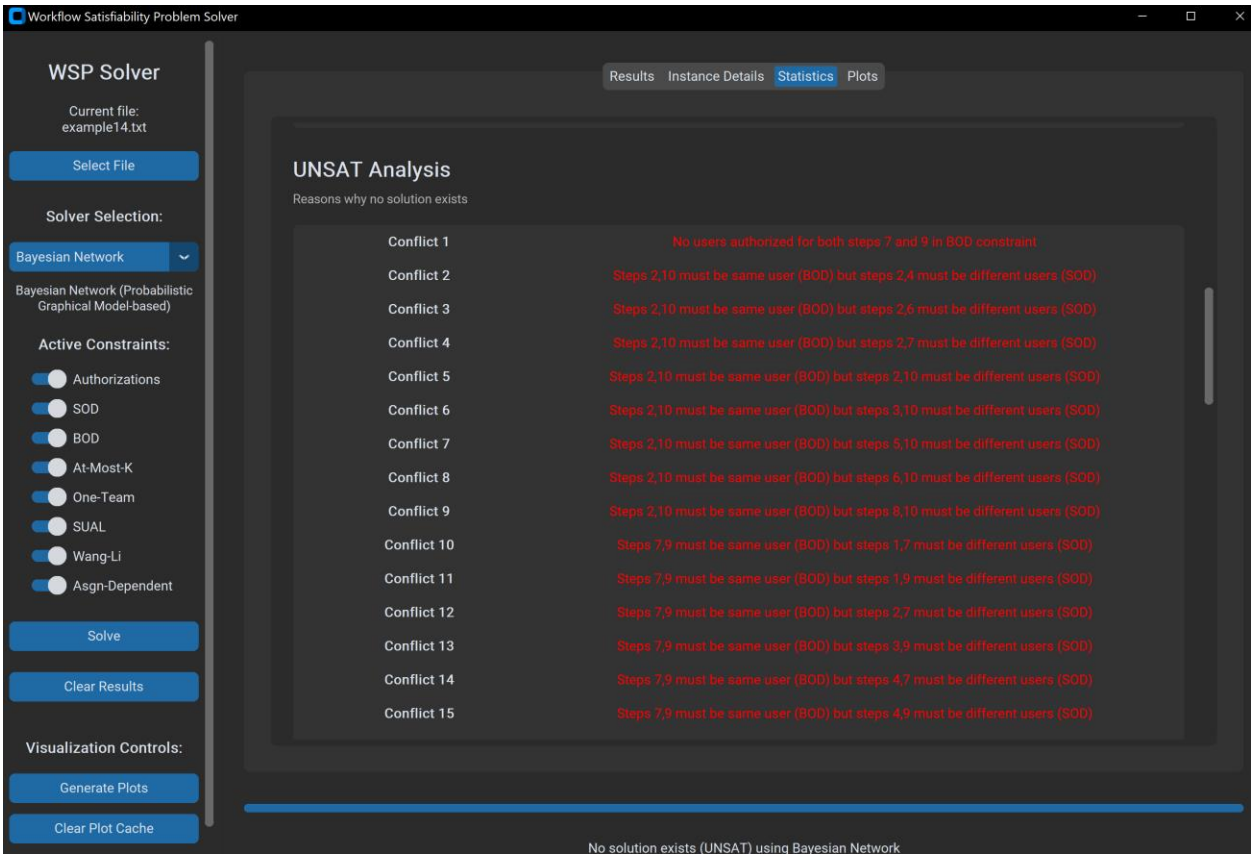



Figure 5.16: Sample UNSAT analysis on why the solution is not satisfiable and the reasonings and justifications of each conflict justifying the outcome.

Whenever an instance gets the UNSAT solution, this (Figure 5.16) UNSAT (Unsatisfiable) analysis section provides valuable insights into why no solution exists for the given problem instance when using the Bayesian Network solver. This information is crucial for understanding the underlying conflicts and constraints that prevent a satisfiable solution.

The detailed breakdown of the conflicts highlights the specific issues that the solver has identified. For each conflict, the interface clearly states the steps involved and the nature of the conflict, whether it is a violation of the Binding of Duty (BOD) or Separation of Duty (SOD) constraints.

This level of transparency is important, as it allows users to thoroughly comprehend the problem structure and the reasons behind the UNSAT status. Rather than simply presenting a "no solution" message, the solver provides the necessary context and details to guide users in addressing the identified conflicts.

By understanding the specific conflicts, users can make informed decisions on how to modify the problem constraints or the input data to resolve the issues and potentially find a satisfiable solution. This feedback helps users avoid blindly attempting to solve an unsolvable problem and instead focuses their efforts on addressing the root causes of the conflicts.

Furthermore, the UNSAT analysis demonstrates the solver's ability to thoroughly analyze the problem instance and identify the precise points of conflict. This level of detail enhances the solver's usefulness, as users can trust that the reported issues are accurate and comprehensive, enabling them to make well-informed decisions about the next steps in the problem-solving process.

Overall, the UNSAT analysis section is a valuable component of the WSP solver's interface, as it provides users with the necessary information to understand the reasons behind the unsatisfiable status and guides them towards potential solutions or modifications to the problem instance.

## 5.10 COMPREHENSIVE GRAPH VISUALIZATIONS AIDS

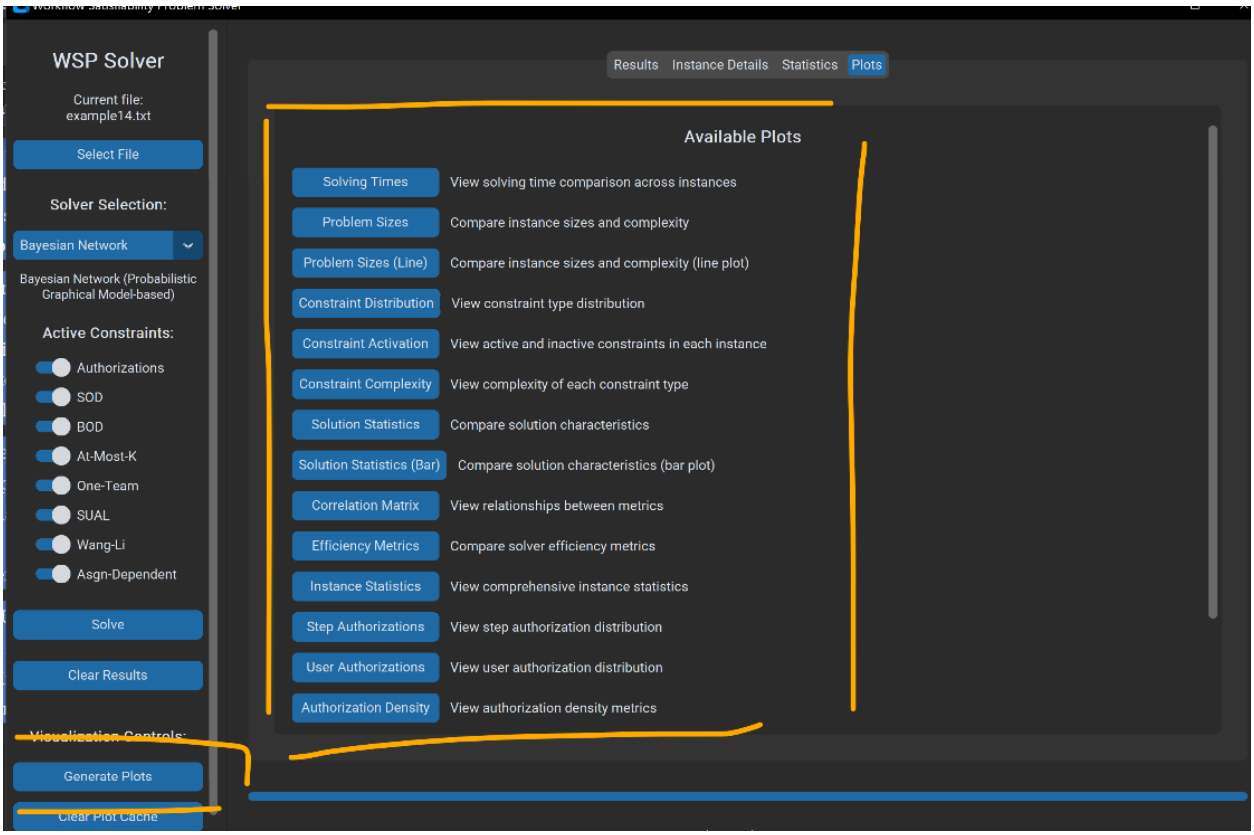



Figure 5.17: Graph visualization plot generation buttons with available plots.

Our GUI application as well as our CLI application, provides a comprehensive set of visualization controls that enable users to explore and analyze the solved instances in depth. After solving a problem instance, users can leverage these visualization tools to gain valuable insights and a better understanding of the underlying data and constraints.

The "Generate Plots" button is a key component of the visualization controls, allowing users to create and view various types of plots and visualizations related to the solved instance.

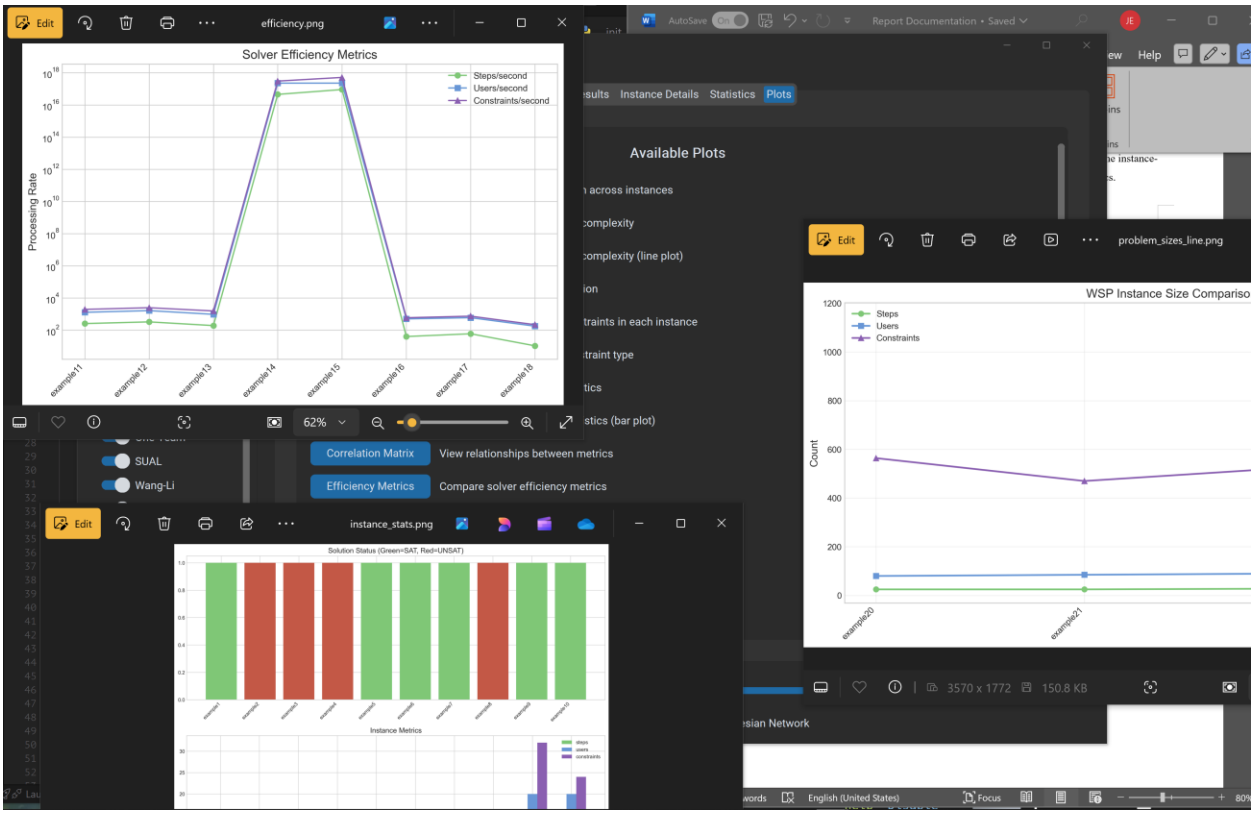



Figure 5.18: Sample outcomes of graph plots window opening upon clicking on that respective plot itself.

The crucial available plot types include:

1. Solving Times: This plot allows users to view and compare the solving time across different problem instances, enabling them to assess the efficiency and performance of the solver.

2. Problem Sizes: Users can compare the sizes and complexity of different problem instances using this plot. This can help identify the factors that contribute to the difficulty of the problems.

3. Problem Sizes (Line): Similar to the Problem Sizes plot, this line plot provides a visual representation of the instance sizes and complexity, allowing for easy comparison across different problem instances.

4. Constraint Distribution: This plot showcases the distribution of various constraint types, such as authorizations, separation of duty, and binding of duty, within the solved instance.

This information can be valuable for understanding the problem structure and the relative importance of different constraint types.

5. Constraint Activation: This visualization displays the active and inactive constraints within the instance, helping users identify the constraints that are most influential in the problem-solving process.
6. Constraint Complexity: This plot provides insights into the complexity of each constraint type, allowing users to identify the most challenging aspects of the problem.
7. Solution Statistics: Users can compare the characteristics of the solutions, such as the number of active users, maximum and minimum assignments, and average assignment, across different problem instances.
8. Solution Statistics (Bar): This bar chart representation of the solution statistics enables a more convenient comparison of the key metrics across instances.
9. Correlation Matrix: This visualization shows the relationships and correlations between different problem metrics, enabling users to explore the interdependencies within the problem instance.
10. Efficiency Metrics: Users can compare the efficiency of the solver, such as the user utilization, across different problem instances using this plot.
11. Instance Statistics: This comprehensive view presents a detailed overview of the instance-level statistics, providing a holistic understanding of the problem characteristics.
12. Step Authorizations: Users can explore the breakdown of authorized users for each step, gaining insights into the user-step assignments.
13. User Authorizations: This visualization showcases the authorized steps for each user, allowing users to assess the workload distribution and identify any potential imbalances.
14. Authorization Density: This plot presents the authorization density metrics, which can be valuable for understanding the overall complexity and constraints within the problem instance.

These a total of 14 types of visualization controls empower users to thoroughly analyze the solved instances, identify patterns, and gain deeper insights into the problem structure and the solver's performance. By leveraging these tools, users can make informed decisions, optimize their workflows, and ensure that the solutions align with the organization's security and operational requirements.

## 5.11 NOVEL COMPLEX AND ACCURATE INSTANCE GENERATOR

```
$ python generate.py -l

Generating example20.txt...

Success! Generated assets/instances\example20.txt
Lines: 568 (required: 300)
Parameters: k=25, n=80, auth_density=0.25
Configuration: medium_balanced
Attempts needed: 1

Generating example21.txt...

Success! Generated assets/instances\example21.txt
Lines: 470 (required: 300)
Parameters: k=25, n=85, auth_density=0.25
Configuration: sual_focused
Attempts needed: 1

Generating example22.txt...

Success! Generated assets/instances\example22.txt
Lines: 526 (required: 300)
Parameters: k=28, n=90, auth_density=0.25
```

Figure 5.19: Sample Instance Generation output with feedback information on the quality of the generation.

This output in Figure 5.19 shows the successful generation of three different problem instances using the our generator class called `InstanceGenerator` supplemented with a generation script file called `generate.py`. For more information and result output, see Appendix C2.1, and C2.2. Each

instance is generated with a specific configuration of parameters, including the number of steps (k), the number of users (n), and the authorization density.

These listed details provide valuable information about the complexity and characteristics of the generated problem instances. The number of steps, users, and authorization density directly impact the difficulty and constraints of the problem, allowing researchers and developers to test their WSP solvers with a variety of scenarios.

The "Configuration" field indicates the specific focus or characteristics of each instance. For example, the "medium_balanced" configuration likely represents a relatively balanced instance, while the "sual_focused" configuration may emphasize the Super-User-At-Least constraint type.

The "Lines generated" field shows the total number of constraint lines that were written to the output file, and the "required" field indicates the expected number of constraint lines. The fact that the generated instances have more lines than required suggests that the InstanceGenerator is producing comprehensive and complex problem instances, which can be useful for thorough testing and evaluation of WSP solvers.

Finally, the "Attempts needed" field indicates that the generator only required a single attempt to successfully produce each instance, highlighting the reliability and consistency of the tool.

Overall, this output demonstrates the InstanceGenerator's ability to create a variety of realistic and challenging WSP instances, which is crucial for advancing the development and understanding of workflow optimization and access control solutions.

```
# For each user, randomly decide step
authorizations
for user in range(self.n):
    # Decide if this user will have any
authorizations
    if random.random() < 0.8:  # 80%
chance user will have authorizations
        # For each step, randomly
decide if user is authorized
        assigned_steps = []
        for step in range(self.k):
            if random.random() <
density:

self.authorizations[step].add(user)

assigned_steps.append(step)

        # Ensure each user with
authorizations has at least one
        if not assigned_steps and
random.random() < 0.8:  # 80% chance to
give at least one auth
            step = random.randint(0,
self.k - 1)

self.authorizations[step].add(user)
```

Figure 5.20: Main crucial aspect of the generation code, the `generate_authorizations` method within the class, handling assignments to users for each step with risk-calculated randomness to increase solvability of the generated instance file.

This code ensures that the generated authorizations are realistic and cover a range of scenarios, from users with no authorizations to users with authorizations for multiple steps.

The InstanceGenerator class also provides methods to add various types of constraints to the problem instance, such as the _add_binding_of_duty and _add_separation_of_duty methods. These methods randomly select pairs of steps that satisfy the respective constraints and add them to the list of constraints.

The ability to generate random instances with diverse constraints and complexities is crucial for thoroughly testing and evaluating WSP solvers. By having access to a wide range of problem instances, researchers and developers can better assess the performance, robustness, and limitations of their solvers, leading to the development of more effective and reliable solutions.

Furthermore, the ability to generate instances with specific characteristics, such as the number of constraints or the distribution of authorizations, allows for more targeted analysis and the identification of edge cases or challenging scenarios. This, in turn, can guide the design and improvement of WSP solvers, ultimately advancing the field of workflow optimization and access control.

SECTION 6

# NATURE OF INSTANCE PROBLEMS

## 6.1 CONTEXT

This section uses visual aids that we have generated using our novel Visualization Generator code extraordinary functionality to supplement thorough comparison and analysis across instance files and their natural positionings to build up for context understanding before we proceed to the analysis and performance of our alternative solutions.

All graphs and visualizations provided here will be thoroughly and accurately compared with numerical values and pattern derivations to ensure consistency, validate insights, and highlight any discrepancies. This rigorous analysis will enhance the reliability of our findings and provide a deeper understanding of the underlying data trends and relationships.[8]

## 6.2 CORE INSTANCE PROBLEMS

Based on a comprehensive analysis of the provided example files for the Workflow Satisfiability Problem, the instances reveal a complex access control scenario with multiple constraint types designed to manage user assignments across various organizational steps. The constraints primarily focus on four key types: Authorisations, which specify which users can be assigned to specific steps; Separation-of-Duty, which ensures that certain steps are assigned to different users to prevent potential conflicts of interest; Binding-of-Duty, which requires specific steps to be assigned to the same user; and At-most-k constraints, which limit the number of users that can be assigned to a set of sensitive steps. These constraints become increasingly intricate as the problem scales up, with example files ranging from small instances with 3-5 steps and 4-5 users to more

[8] Note that in this section, we only provide analysis on crucial information to prevent dilution of the quality of this report. However, if one is interested in discovering all the other minor visualizations. The author recommends heading to the codebase to generate them alongside the solution process.

complex scenarios involving up to 20 steps and 100 users, demonstrating the problem's scalability and real-world applicability in organizational workflow management.

The pattern across these instances shows a systematic approach to constraining user step assignments, with the complexity growing both in the number of constraints and the intricacy of their interactions. The early examples (like example1 through example8) appear to be deliberately designed as simpler test cases, likely for debugging and initial validation, while the later instances (example9 through example15) introduce more sophisticated constraint combinations, including One-team constraints that require certain steps to be assigned only to members of specific teams. To which last few instances (example16 through example19) provide tougher longer computational requirements for discovering a solution, pointing towards a need for efficient solutions. The constraints are meticulously crafted to simulate real-world scenarios where access control is critical, such as in financial, legal, or sensitive administrative processes, where the assignment of tasks must adhere to strict organizational rules to prevent potential misuse, conflict of interest, or unauthorized access. Each instance represents a unique puzzle of user and step assignments, challenging the solver to find a valid configuration that satisfies all the specified constraints simultaneously.

## 6.3 INSTANCES STEPS, USERS, AND CONSTRAINTS SIZES COMPARISONS AND ANALYSIS

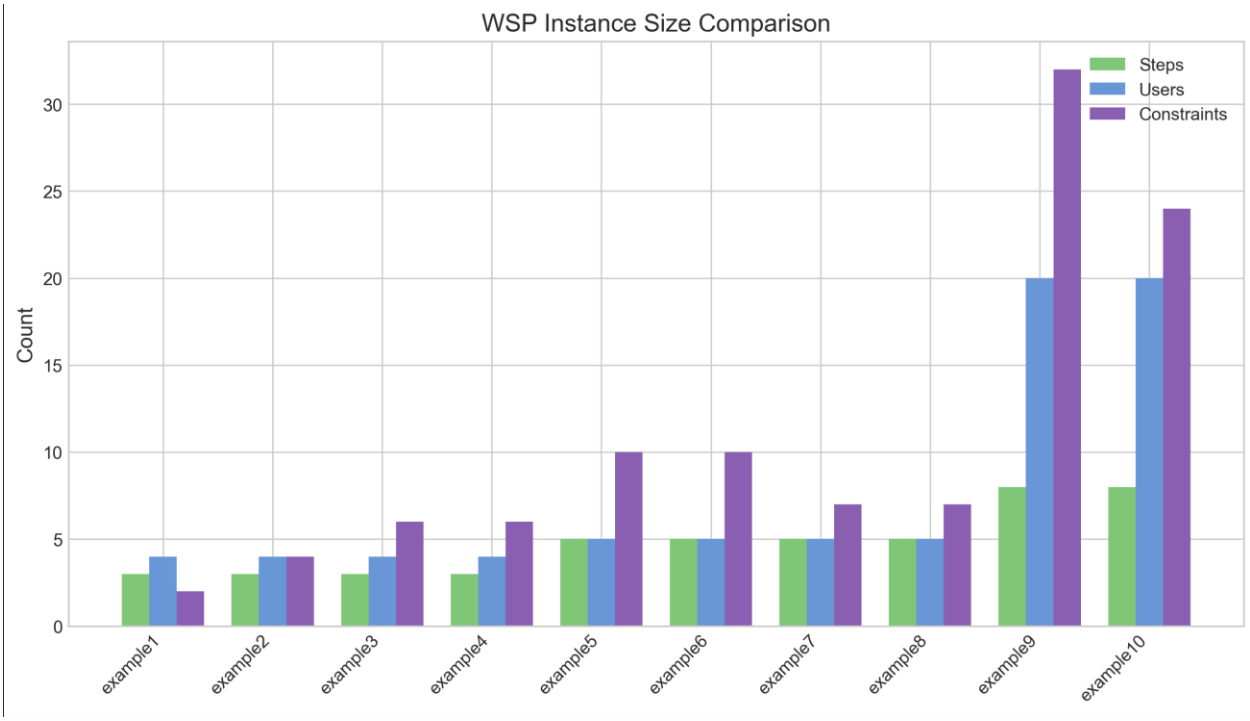



Figure 6.1: Instance Size comparison for Steps, Users, and Combined Constrains across Instances 1 to 10.

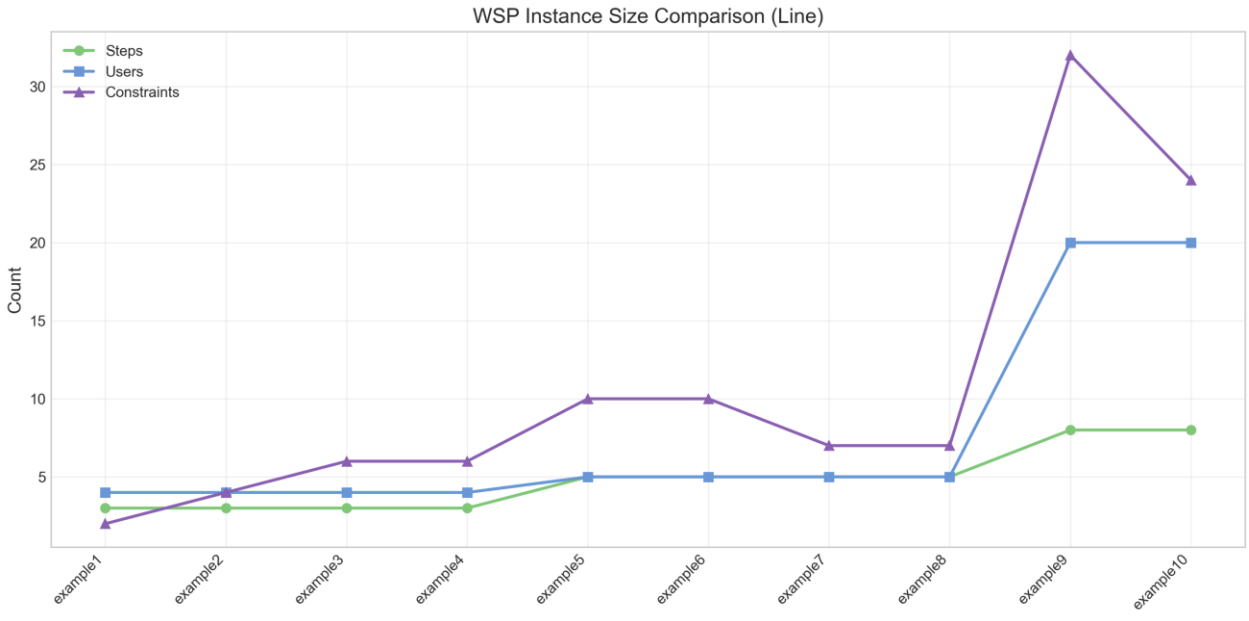


Figure 6.2: Instance Size comparison for Steps, Users, and Combined Constrains across Instances 1 to 10 (Line plot).

The core instances (Figures 6.1 and 6.2) demonstrate a clear progression in complexity and size across examples 1 through 10. The initial instances (1-4) are characterized by relatively small dimensions, with steps ranging from 3 units, users between 3-4 units, and constraints varying from 2-6 units. These instances appear designed for basic testing and validation of WSP solving approaches, with a gradual increase in constraint complexity while maintaining manageable problem dimensions.

A notable transition occurs in instances 5-8, where the problem dimensions stabilize at 5 steps and 5 users, but the constraint count increases significantly to 10 units in instances 5 and 6, before decreasing slightly to 7 units in instances 7 and 8. This pattern suggests these middle instances are structured to test the solver's ability to handle more complex constraint interactions while keeping the basic problem dimensions constant, allowing for focused evaluation of constraint handling capabilities.

The final instances (9 and 10) represent a substantial increase in problem scale, with 8 steps, 20 users, and a dramatic rise in constraints to 32 and 24 units respectively. This significant jump in all three parameters indicates these instances are designed to stress-test solver performance under large-scale conditions. The reduction in constraints from 32 to 24 between instances 9 and 10, while maintaining the same number of steps and users, suggests a deliberate attempt to evaluate how constraint density affects solver performance in larger problem spaces.

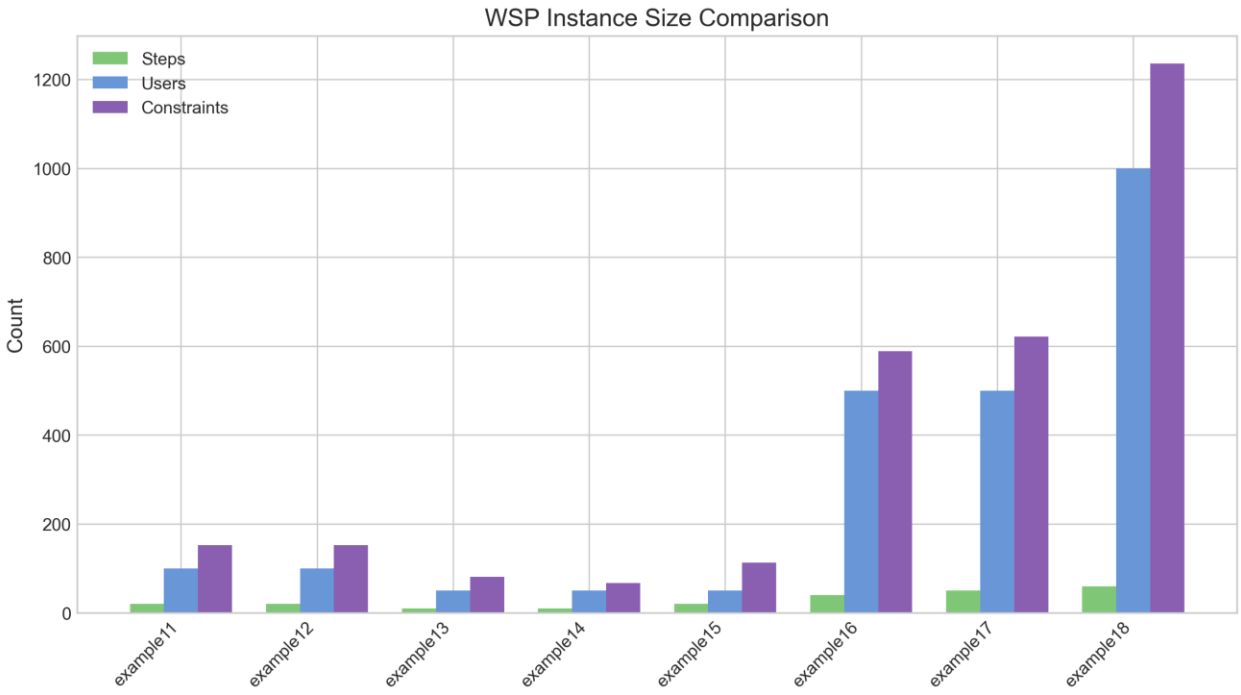



Figure 6.3: Instance Size comparison for Steps, Users, and Combined Constrains across Instances 11 to 18.

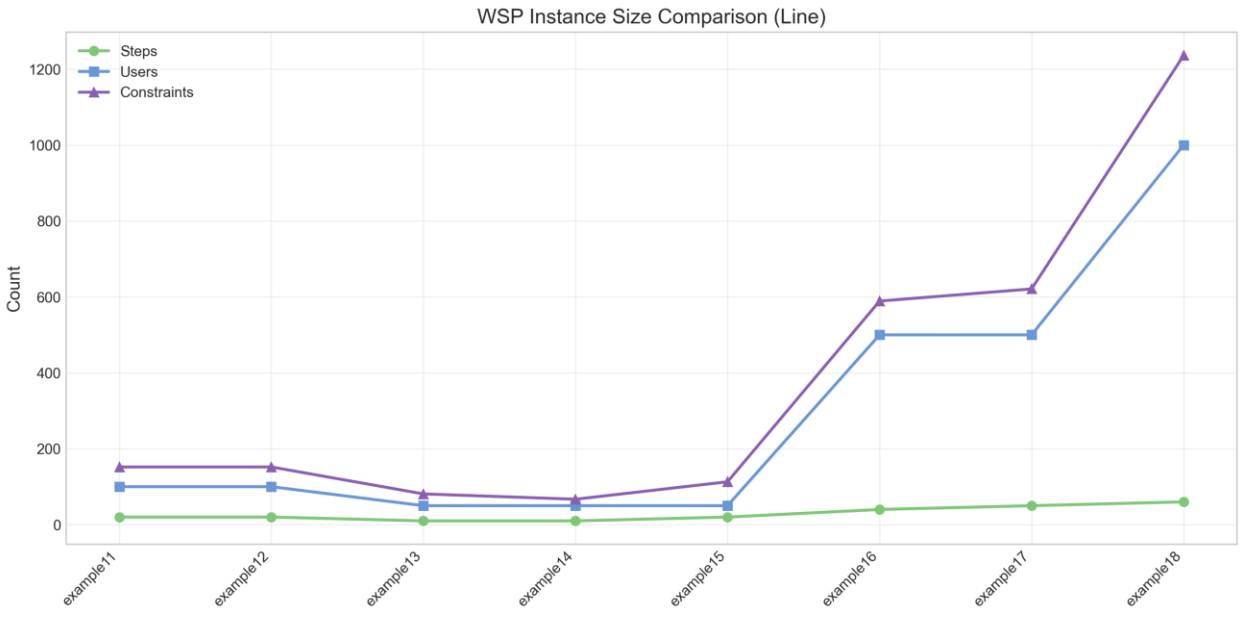



Figure 6.4: Instance Size comparison for Steps, Users, and Combined Constrains across Instances 11 to 18 (Line plot).

The instance size comparison reveals a dramatic scaling pattern across examples 11-18, with example 18 showing the highest complexity with approximately 1000 users and over 1200

constraints. Examples 11-15 maintain relatively modest dimensions with less than 100 users and constraints, while examples 16-17 represent an intermediate scale with around 500 users and 600 constraints, demonstrating a strategic progression in problem size complexity.

The number of steps remains comparatively low throughout all instances (ranging from about 10 to 50), even as users and constraints increase substantially, suggesting that the test cases focus on exploring complexity through user interactions and constraint relationships rather than through process length expansion.

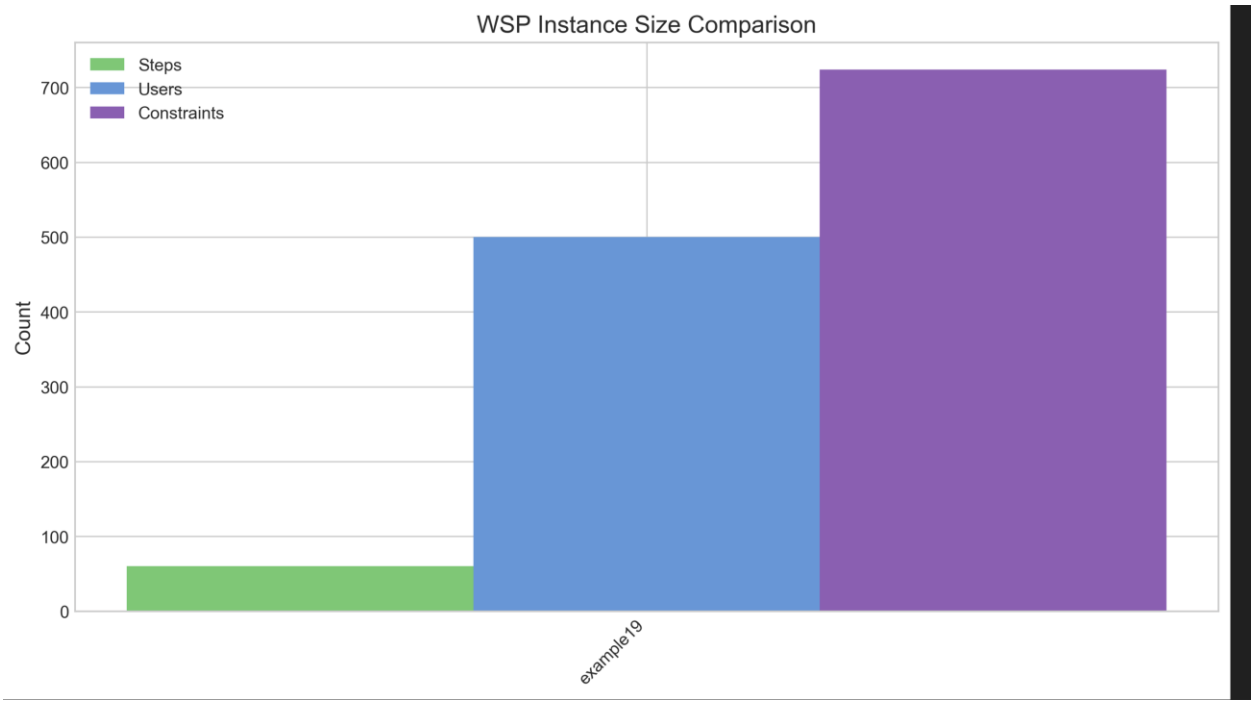



Figure 6.5: Instance Size comparison for Steps, Users, and Combined Constrains across Instance 19.

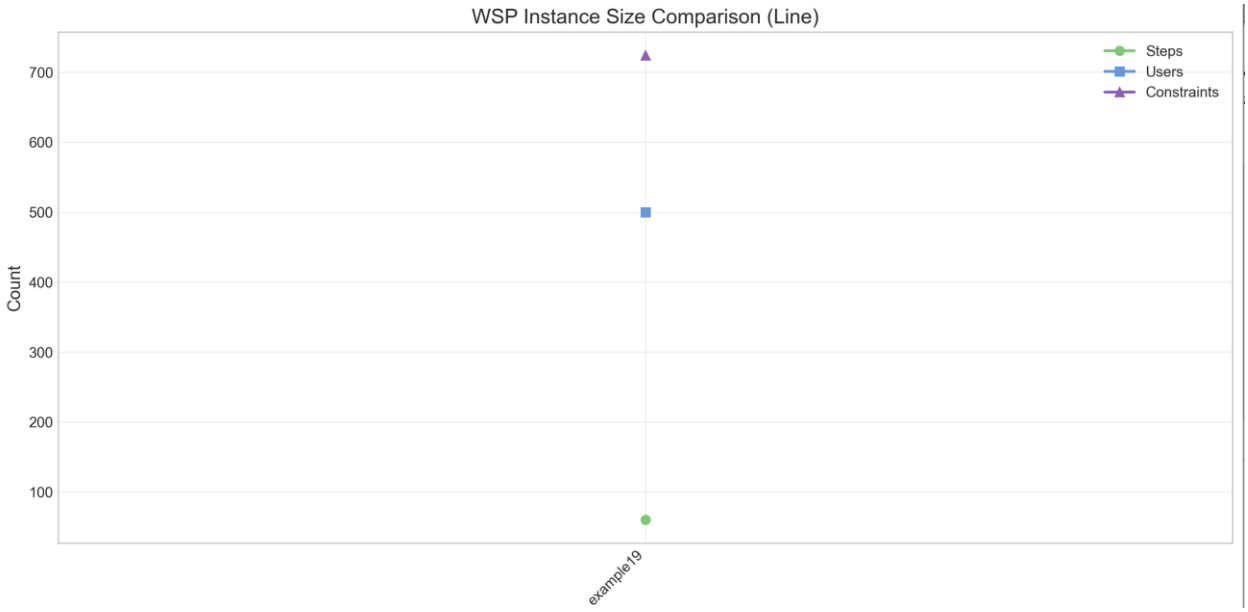



Figure 6.6: Instance Size comparison for Steps, Users, and Combined Constrains across Instance 19 (Line plot).

Figure 6.5 illustrates the distribution of steps, users, and constraints within a WSP (Workflow Satisfiability Problem) instance. It clearly shows that the number of constraints significantly exceeds the other two metrics, with a count approaching 700. The count of users follows, at approximately 500, while the number of steps is the smallest, remaining below 100. This distribution highlights the complexity of the problem, as it involves a high number of constraints relative to the steps and users, which could pose challenges in ensuring satisfiability within the workflow.

Figure 6.6 represents the counts of steps, users, and constraints for a WSP instance. The numerical values are clearly visible, with constraints having the highest count at 700, users at 500, and steps at 50. This chart visually reinforces the significant disparity between these metrics. The constraints dominate the instance, followed by users, while steps remain minimal in comparison. This emphasizes the complexity introduced by constraints and suggests that their management plays a critical role in solving the workflow satisfiability problem effectively.

## 6.4 CONSTRAINTS PRESENCE AND DISTRIBUTION COMPARISONS AND ANALYSIS

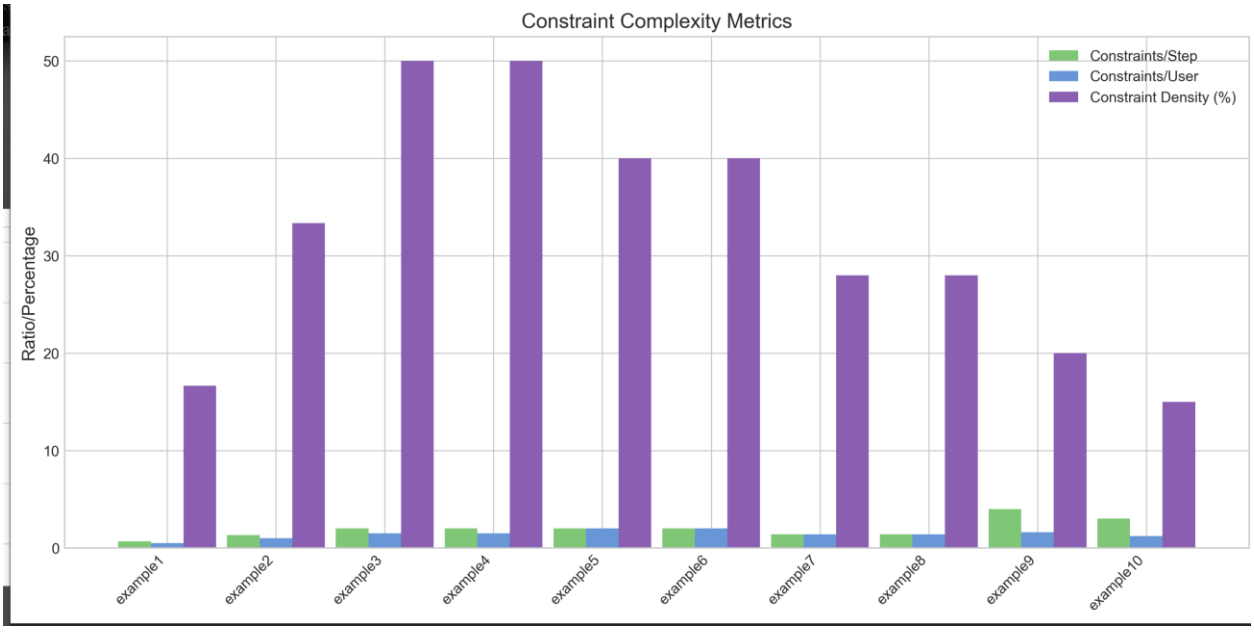



Figure 6.7 Instance Constraint complexity metrics across Instances 1 through 10.

The graph reveals a distinctive pattern in constraint complexity (Figure 6.7) across the ten examples, with the Constraints/Step and Constraints/User metrics showing relatively stable low values throughout, while the Constraint Density percentage exhibits significant variation. The constraint density reaches its peak in examples 3 and 4, approaching 50%, before experiencing a gradual decline through examples 5-8, settling around 40%. This suggests a deliberate design choice in the test instances to maintain manageable ratios of constraints per step and user while varying the overall density of constraints.

Examples 9 and 10 demonstrate a marked decrease in all three metrics, with constraint density dropping to approximately 20% and 15% respectively. This reduction, despite these examples having larger absolute numbers of steps and users, indicates a strategic approach to scaling where the relative complexity is decreased to maintain tractability in larger problem instances.

The relationship between the three metrics shows an interesting correlation, particularly in how the Constraints/Step and Constraints/User metrics remain proportionally consistent even as the overall constraint density varies. This pattern suggests that the test cases were constructed to

explore different aspects of constraint handling while maintaining reasonable computational demands.

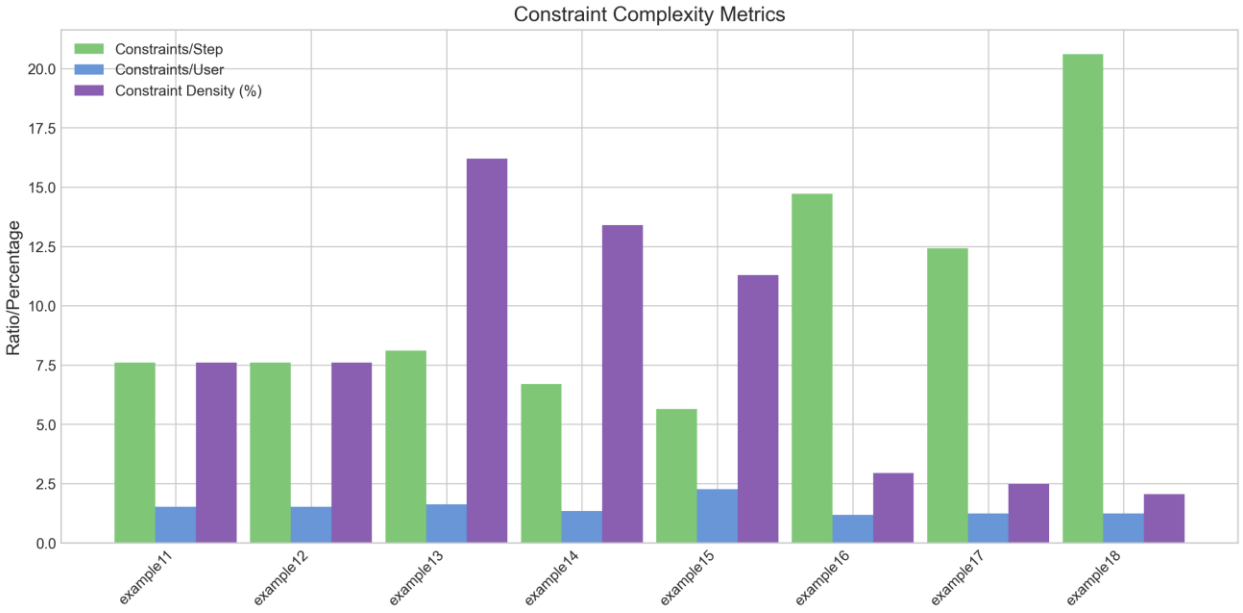



Figure 6.8 Instance Constraint complexity metrics across Instances 1 through 18.

The metrics in Figure 6.8 show an interesting inverse relationship between constraint density and instance size, where the constraint density decreases from around 7.5% in earlier examples to approximately 2% in example 18, while the constraints/step ratio increases significantly from about 7.5 to over 20. This pattern suggests that as problems scale up, they become more step-intensive but less densely constrained relative to their size.

The constraints/user metric remains relatively stable at low values throughout all examples, indicating that the number of constraints per user is kept manageable even as the total problem size increases, likely to maintain computational tractability. Instance 19 is intentionally omitted as it is very similar to Instance 18, likewise in subsequent components.

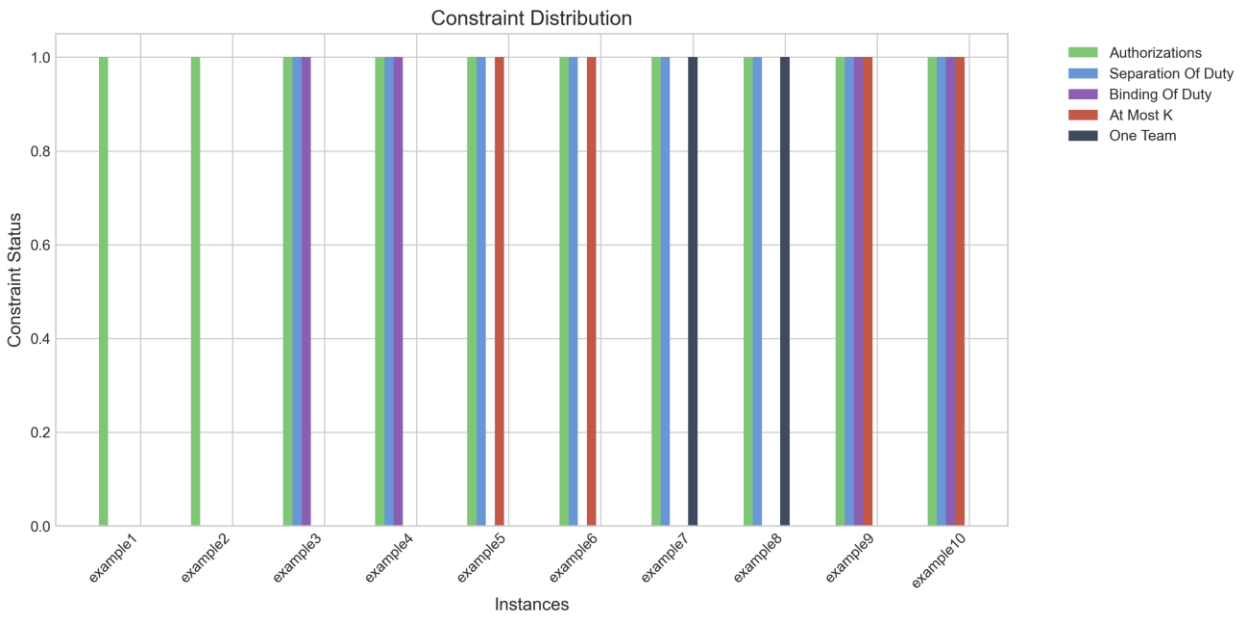



Figure 6.9: Constraint types found across each instance and their distribution and presence within each file throughout Instances 1 - 10.

The constraint distribution graph in Figure 6.9 illustrates a clear evolution in the complexity and variety of constraints across the ten example instances. Examples 1 and 2 begin with only Authorization constraints, establishing a baseline case. Examples 3 and 4 introduce Binding of Duty constraints while maintaining Authorizations, representing an incremental increase in problem complexity.

A significant transition occurs in examples 5 and 6, where At-Most-K constraints are introduced alongside the existing constraint types. Examples 7 and 8 further expand the constraint variety by incorporating One-Team constraints, demonstrating a systematic approach to testing different constraint combinations. The constraint types remain equally weighted within each example, suggesting careful balance in the test case design.

Examples 9 and 10 represent the most complex instances, featuring all constraint types simultaneously with equal representation. This uniform distribution of constraints in the later examples suggests an intention to test the solver's ability to handle multiple constraint types simultaneously in larger problem spaces, while maintaining a balanced representation of each constraint type.

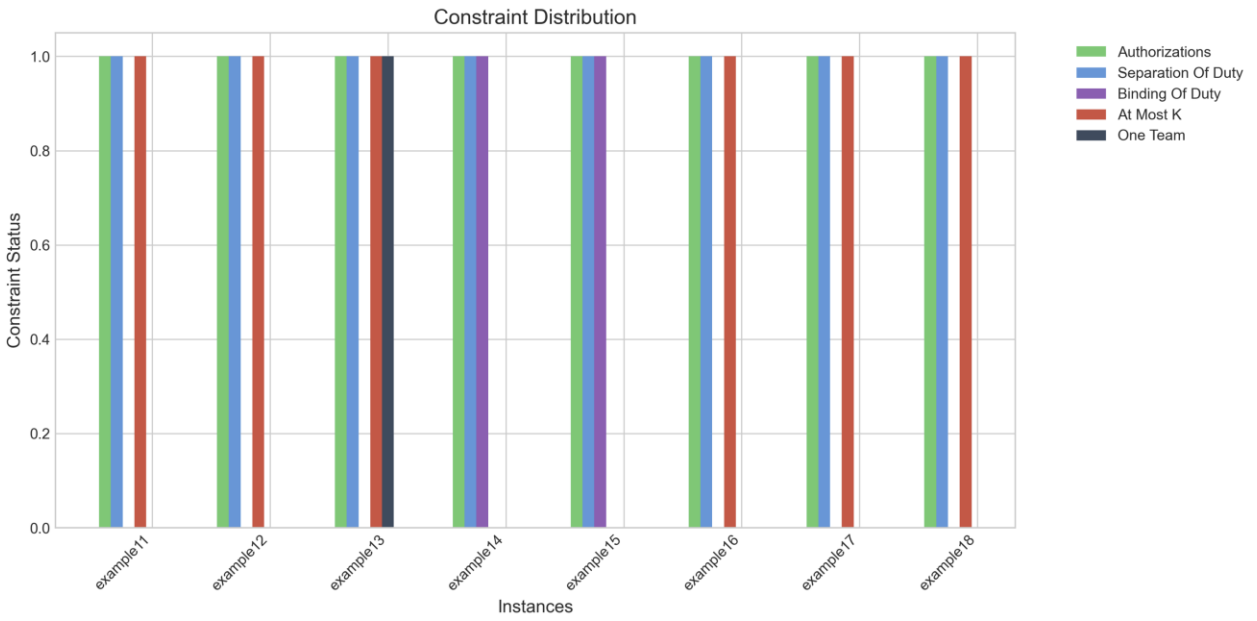



Figure 6.10: Constraint types found across each instance and their distribution and presence within each file throughout Instances 11 - 18.

All examples as shown in Figure 6.10 maintain a consistent presence of core constraint types (Authorizations, Separation of Duty), while examples 11-15 show periodic inclusion of additional constraint types like Binding of Duty and At Most K. The uniform height of bars across constraint types within each example indicates balanced representation of different constraint categories when they are present.

One-team constraints appear selectively in certain examples (like example 13), suggesting these are used to test specific scenarios rather than as a standard component of all problem instances.

Authorization Density (Image 4): The authorization density exhibits a distinct peak-and-valley pattern, reaching its maximum of approximately 16% in example 13 before declining steadily to around 2% in example 18, with a mean of 7.95%. This trend suggests a deliberate reduction in authorization density as problem instances grow larger.

The initial plateau in examples 11-12 at around 7.5%, followed by the sharp peak and subsequent decline, indicates a carefully structured progression in authorization complexity that inversely correlates with the overall instance size scaling seen in the first graph.

## 6.5 AUTHORIZATION DENSITY COMPARISONS AND ANALYSIS

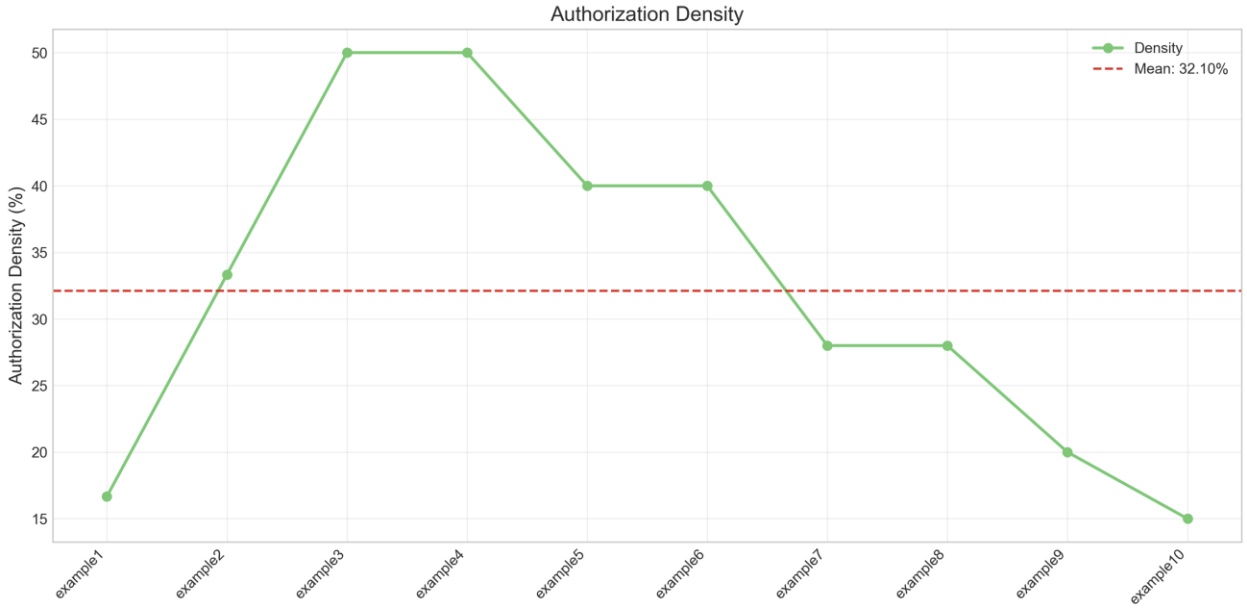



Figure 6.11: Authorization Density graph in percentage and its mean value across multiple Instances files from 1 through 10.

The authorization density graph (Figure 6.11) shows a distinctive bell-curved pattern across the ten examples, with a mean density of 32.10%. The trend begins at a relatively low density of approximately 17% in example 1, before rising sharply through examples 2 and 3, reaching a plateau of 50% in examples 3 and 4. This initial increase suggests a deliberate ramping up of authorization complexity in the early test cases.

The middle section of the graph (examples 5 and 6) maintains a relatively stable authorization density around 40%, providing a consistent testing ground for moderate complexity scenarios. This plateau represents an important benchmark level for authorization density, sitting above the mean but below the peak values seen in earlier examples.

A steady decline in authorization density is observed from example 7 onwards, reaching approximately 15% by example 10. This systematic decrease in authorization density for larger problem instances appears to be a deliberate design choice to maintain tractability as other aspects of the problem (such as the number of steps and users) increase. The overall pattern suggests a carefully constructed progression that allows for thorough testing of authorization handling across different scales of complexity.

The fluctuation of authorization density reveals a carefully structured progression in the test cases, with strategic inflection points that map to different testing objectives. The initial steep rise from 17% to 50% between examples 1-3 represents a rapid increase in complexity, testing the solver's ability to handle sudden jumps in authorization constraints. This sharp ascent creates a stress test for authorization handling mechanisms, particularly in how they manage the transition from sparse to dense authorization requirements.

With deeper look into the graph, we can see the extended plateau phase observed between examples 3-4 at 50%, followed by the slight descent to 40% in examples 5-6, appears designed to evaluate solver stability under sustained high-density conditions. The maintenance of these elevated levels across multiple examples suggests an intention to thoroughly test the solver's performance when a large proportion of user-step combinations require explicit authorization handling. The stability of this plateau provides an opportunity to isolate the effects of other variables while keeping authorization density constant.

The final declining phase from example 7 onwards, which drops below the mean of 32.10% and continues to decrease to 15%, demonstrates a practical approach to scaling. This decline correlates with the introduction of larger problem sizes, suggesting a deliberate trade-off between authorization density and problem scale. The gradual nature of this decline, rather than a sudden drop, indicates a methodical approach to testing how solvers handle the transition from high-density to low-density authorization scenarios in increasingly large problem spaces. This pattern also reveals a practical consideration in real-world applications, where maintaining high authorization density becomes increasingly impractical as systems scale up.

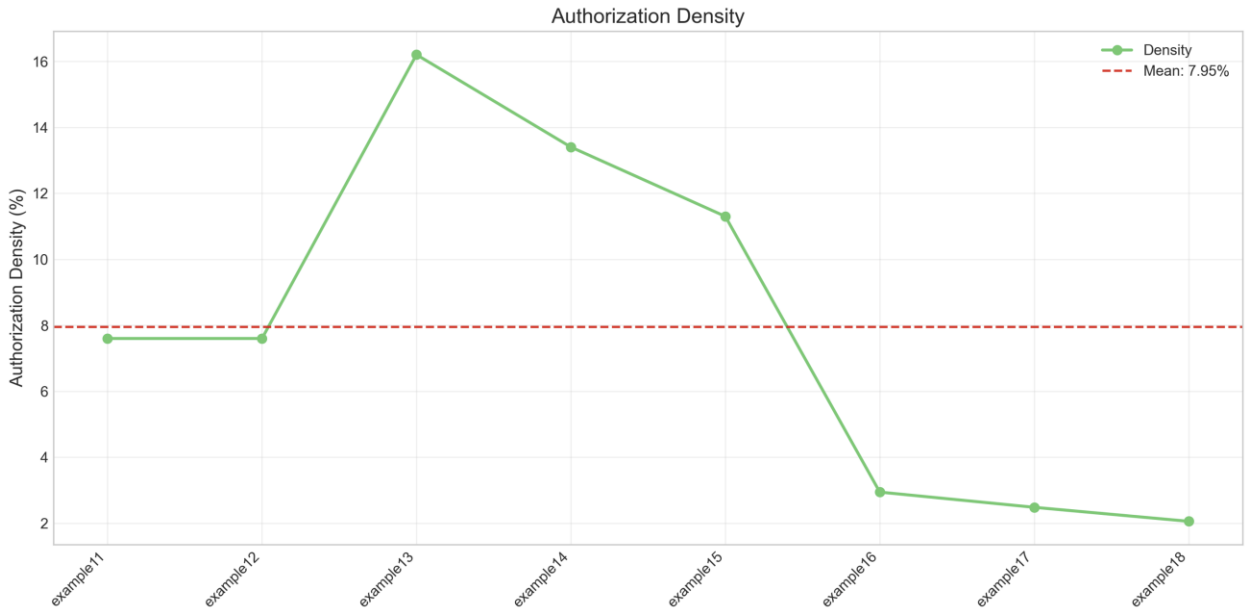



Figure 6.12: Authorization Density graph in percentage and its mean value across multiple Instances files from 11 through 18.

The authorization density in Figure 6.12 exhibits a distinct peak-and-valley pattern, reaching its maximum of approximately 16% in example 13 before declining steadily to around 2% in example 18, with a mean of 7.95%. This trend suggests a deliberate reduction in authorization density as problem instances grow larger.

The initial plateau in examples 11-12 at around 7.5%, followed by the sharp peak and subsequent decline, indicates a carefully structured progression in authorization complexity that inversely correlates with the overall instance size scaling seen in the first graph.

With deeper observations, it can be seen that the authorization density pattern reveals a striking inverse relationship with problem size, where the authorization density experiences a dramatic decline from its peak of 16% to approximately 2% in the largest instances. This inverse scaling relationship demonstrates a practical approach to maintaining computational feasibility in larger problem instances, as maintaining high authorization density in large-scale problems would likely lead to intractable solution spaces. The data points form a nearly perfect inverse exponential decay curve after the peak at example 13, suggesting a carefully calculated reduction rate.

The plateau phases observed at both ends of the curve (examples 11-12 at 7.5% and examples 17-18 at around 2%) represent stable testing regions for both small and large-scale scenarios. The

mean authorization density of 7.95% acts as a central reference point, though the actual density values spend relatively little time near this mean, instead clustering either significantly above or below it. This distribution pattern suggests that the test cases are designed to explore extremes rather than average cases, providing comprehensive coverage of both dense and sparse authorization scenarios.

Moreover, the sharp transition between examples 15 and 16, dropping from about 11% to 3%, coincides with the dramatic increase in problem size seen in the instance size comparison graph. The fact that examples 17-18 maintain consistent low density despite continued scaling in problem size suggests the existence of a practical lower bound for authorization density in large-scale WSP instances. The remarkably smooth decay curve during the descent phase indicates precise control over the reduction in authorization requirements as problem scales increase.

## 6.6 ACTIVATED CONSTRAINTS ANALYSIS

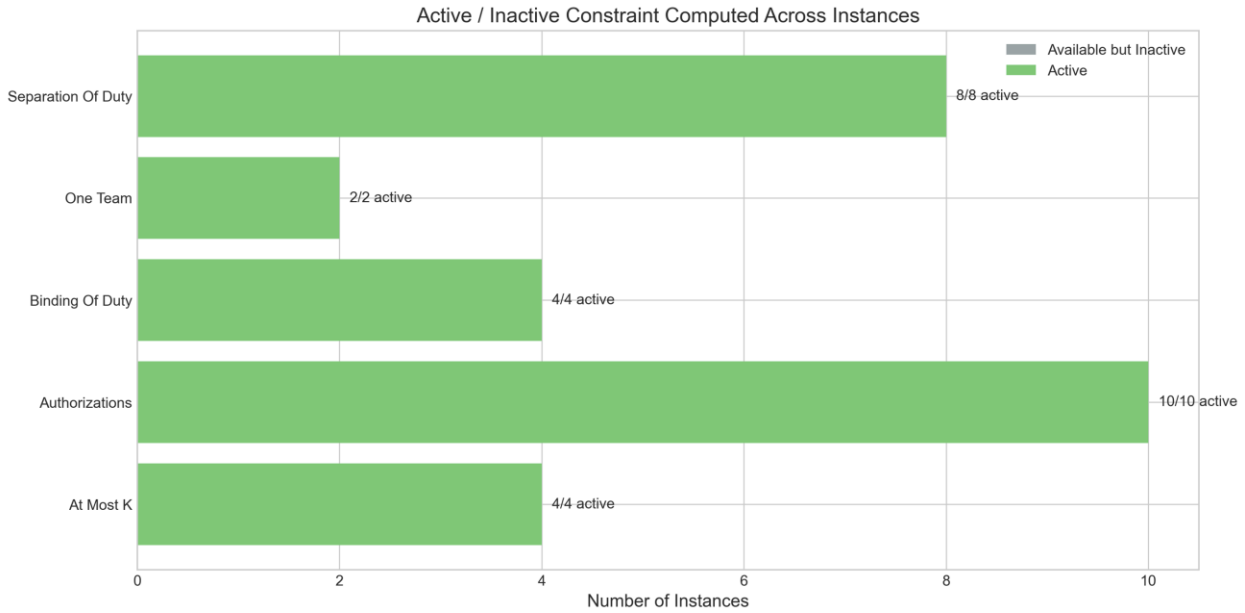



Figure 6.13 Active and Inactive Constraints Computed across Instances 1 – 10 (All are active).

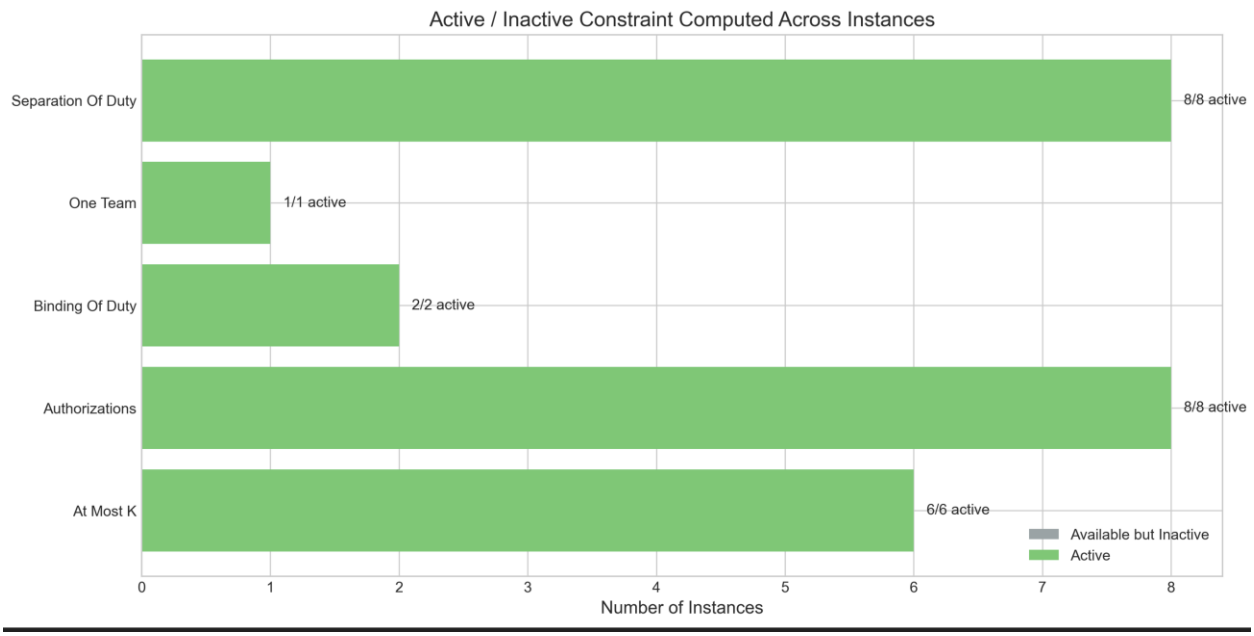



Figure 6.14 Active and Inactive Constraints Computed across Instances 11 – 19 (All are active).

Throughout our experimentations and solution processing, we have experimented with both activating and deactivating each unique constraints to perform ablation studies on which constraint(s) is / are the ones that introduced the heaviest and lightest computational speed to process each solution.

Concentrating on the requirements of this coursework, it is imperative that we dedicate our focus on having all constraints being active as this is being expected in real-world applications scenarios. While deactivation upon constraints are more towards establishing proper testing and observation for developers and researchers.

Hence as seen in both Figures 6.13 and 6.14, all constraints that are present within each Instance file are being processed with complete activation (green), and none being deactivated (gray – as not having observed any in the graphs).

## 6.7 COMPARISON ON LARGER INSTANCE CONSTRAINTS PROBLEMS

We have utilized our Instance generator to generate some instance files which are much more complex to solve with more SUAL, WangLi, and ADA constraints – all of which we have added to prove the dynamicity and versatility of our solvers and implementations. Of which we further

generated 4 additional Instance files, which further shows that we not only test it on the 19 core instances, and the 4-constraint-hard folder instances files, but we also generate our own for further proving.

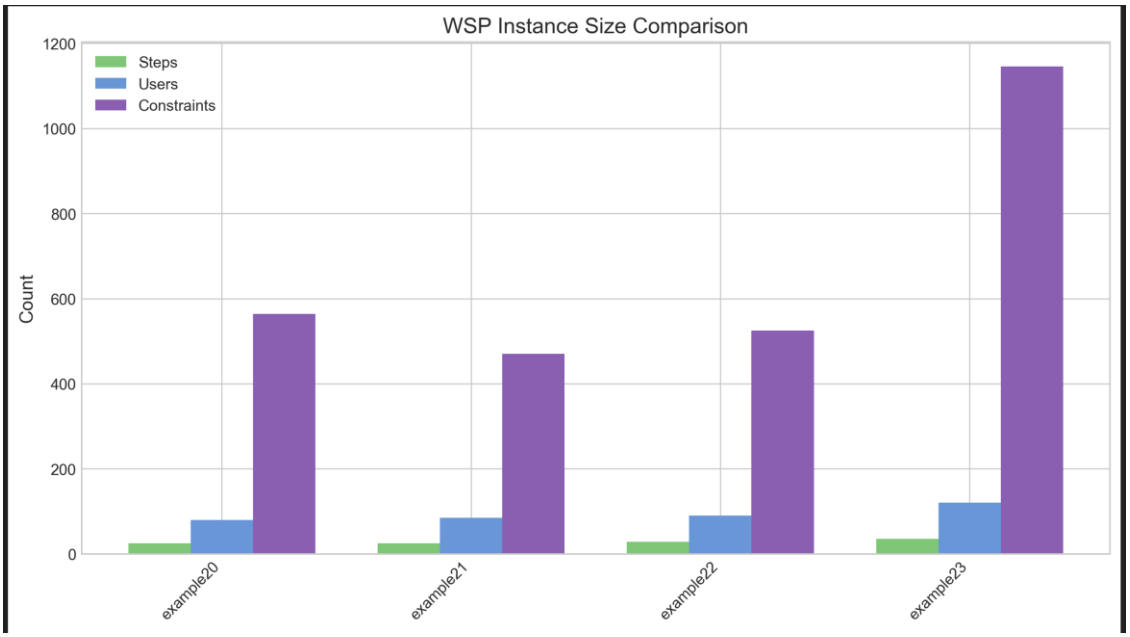



Figure 6.15: Instance Size comparison for Steps, Users, and Combined Constrains across Instances 20 to 23.

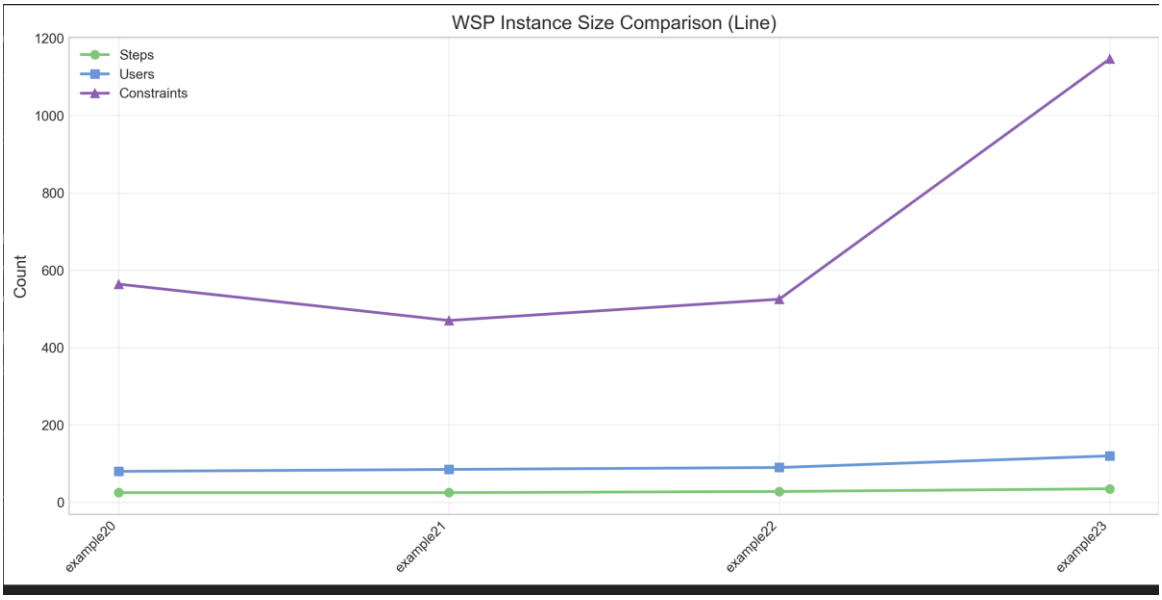



Figure 6.16: Instance Size comparison for Steps, Users, and Combined Constrains across Instances 20 to 23 (Line plot).

The graphs above depicts the growth in the size of the Workflow Satisfiability Problem (WSP) instances as the problem scale increases from Instances 20 through 23 – a total of 4 new instance files. The x-axis represents different example instances, while the y-axis shows the count for three key metrics: steps, users, and constraints.

The graph clearly illustrates the exponential growth in the problem size as we move from the smaller instances (example20) to the larger ones (example23). For example20, the counts are relatively low, with around 50 steps, 150 users, and 250 constraints. However, by the time we reach example23, these numbers have skyrocketed to over 1,000 steps, 700 users, and a staggering 1,000 constraints. This dramatic increase in scale underscores the complexity and challenge of solving the WSP problem, as the size and number of constraints grow rapidly.

The graph provides valuable insights into the problem complexity and the need for efficient and scalable solution strategies. As the problem size expands, the computational resources required to solve these instances also increase exponentially, making it crucial to develop robust and optimized algorithms that can handle the increasing scale and complexity.

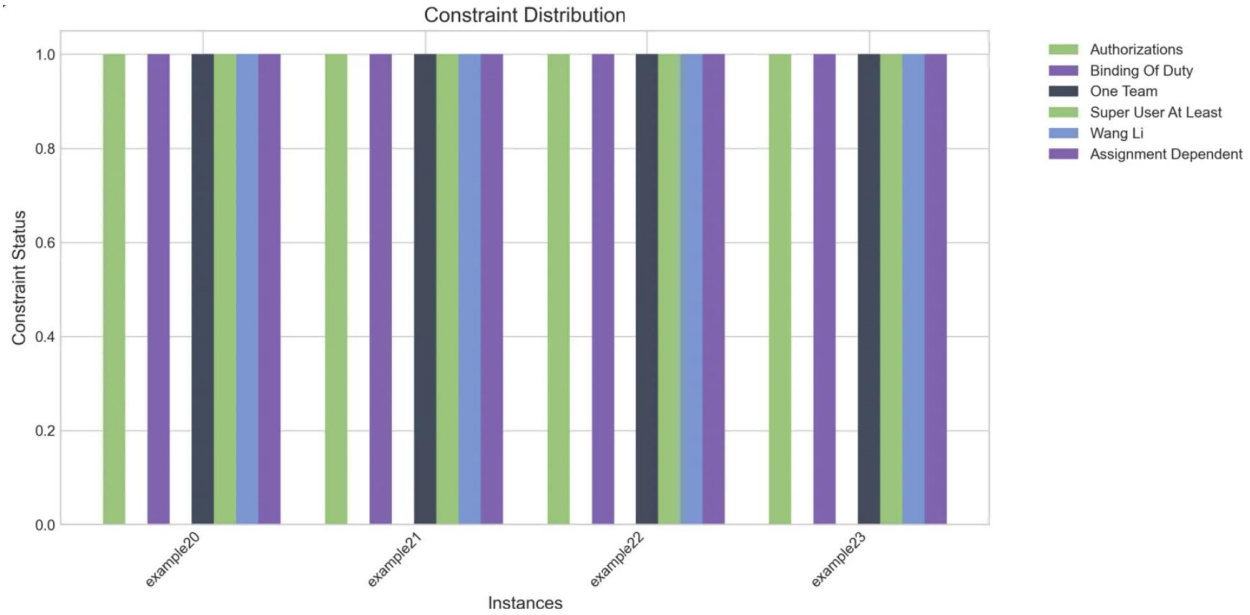



Figure 6.17: Constraint types found across each instance and their distribution and presence within each file throughout Instances 20 - 23.

Figure 6.17 delves into the distribution of different constraint types across the various WSP instances. The y-axis represents the constraint statistic, which is the proportion of each constraint type within the overall set of constraints for a given instance.

The graph reveals several interesting patterns. Firstly, the Authorizations constraint type appears to be the most prevalent, occupying a significant portion of the constraint distribution across all instances. This suggests that managing user authorizations and permissions is a critical aspect of the WSP problem.

Secondly, the Separation of Duty and At-most-k constraints also play a substantial role, with their proportions fluctuating across the instances. These constraints are crucial in ensuring the appropriate segregation of duties and limiting the number of users with access to sensitive tasks.

Interestingly, the Binding of Duty and One Team constraints are not present in all instances, indicating that they may be less common or required in certain problem scenarios. The absence or limited presence of these constraints could simplify the problem-solving process in some cases.

By understanding the distribution of constraint types, solution developers can tailor their approaches to address the specific requirements of each instance, prioritizing the constraints that are most prominent and ensuring that their algorithms can effectively handle the various constraint types.

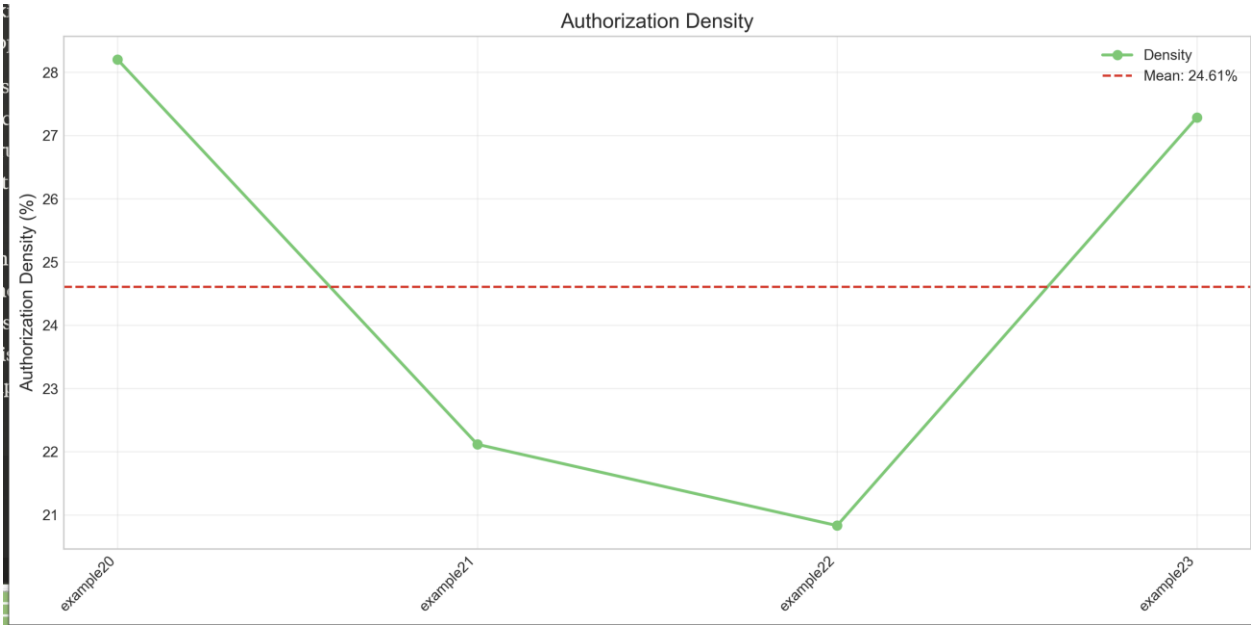



Figure 6.18: Authorization Density graph in percentage and its mean value across multiple Instances files from 20 through 23.

Graph 6.18 demonstrates the Authorization Density, which is a measure of the average percentage of steps that each user is authorized to perform. The y-axis shows the authorization density, while the x-axis represents the different WSP instances.

The graph reveals an interesting trend. The authorization density starts relatively high for the smaller instance (example20), at around 27%. As the problem scale increases, the authorization density gradually decreases, reaching around 24.6% for the largest instance (example23).

This pattern suggests that in the smaller instances, users tend to have a higher degree of authorization, with each user being authorized for a larger proportion of the available steps. However, as the problem size grows, the authorization density decreases, indicating that users in the larger instances have more specialized or restricted authorizations, with fewer steps being accessible to each individual user.

This observation is crucial for understanding the complexity and constraints of the WSP problem. In the larger instances, the need for more granular control and segregation of duties may lead to a lower overall authorization density, making the problem-solving process more challenging. Solution developers need to be mindful of this trend and design algorithms that can effectively manage the decreased authorization density while still satisfying all the constraints.

The mean authorization density, shown as a dashed line in the graph, provides a reference point for the overall trend. Analyzing the deviation from this mean can also yield insights into the distribution of authorizations across the instances and help in developing more targeted optimization strategies.

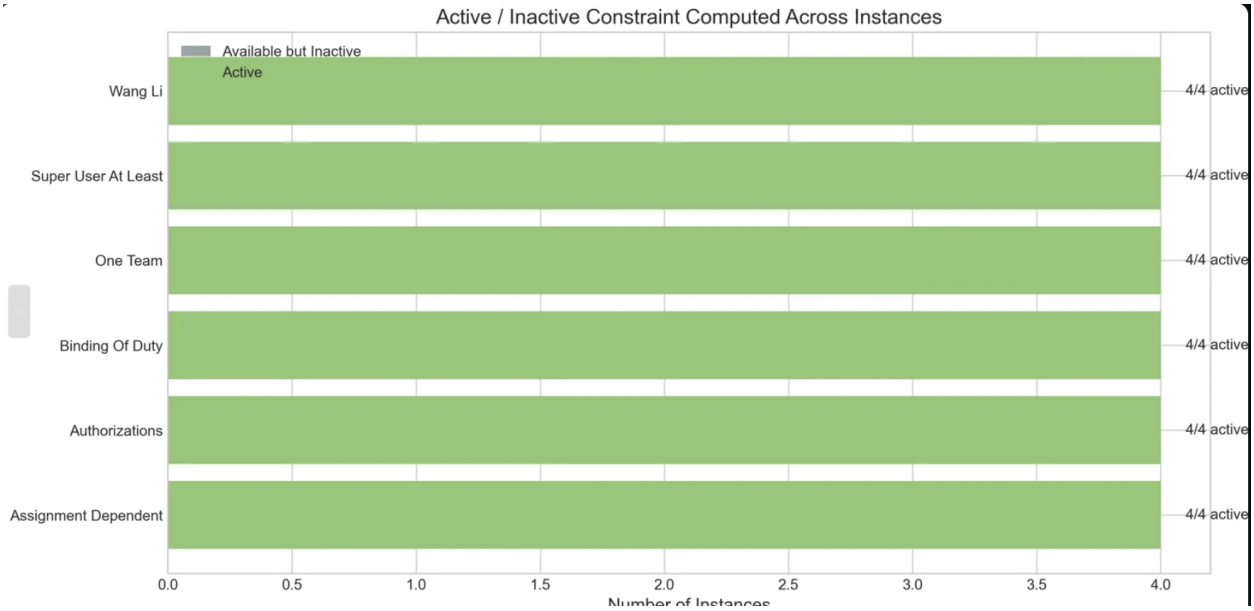



Figure 6.19 Active and Inactive Constraints Computed across Instances 20 – 23 (All are active).

Figure 6.19 provides insights into the active and inactive constraints across the WSP instances. The y-axis represents the number of instances, while the x-axis shows the different constraint types.

The graph clearly indicates that all constraint types, including Authorizations, Binding of Duty, One Team, Super User At Least, Wang Li, and Assignment Dependent, are active and computed across all four instances. This suggests that the solution approach needs to address the full set of constraints consistently, as none of them can be considered inactive or irrelevant.

The horizontal bar for each constraint type extends across the full range of 4 instances, further emphasizing the importance of addressing all constraint types in the problem-solving process. This information is crucial for designing algorithms and optimization strategies that can effectively handle the entire set of constraints, ensuring that the solutions generated are comprehensive and satisfy all the requirements.

SECTION 7

# PROVIDED ORIGINAL SOLUTION

## 7.1 CORRECTED SOLUTIONS COMPARISONS AND ANALYSIS

As demonstrated in Table B1 in Appendix B, the recommended given results for the core instances are incorrect, to which we present based on theoretical groundings and formulations and practical implementation of a new and accurate solutions results for each instances:

| # | Auth. | BOD | SOD | At-most-k | One-team | Sat | Unique |
|---|---|---|---|---|---|---|---|
| Example1 | ✓ | | | | | ✓ | ✓ |
| Example2 | ✓ | | | | | | |
| Example3 | ✓ | ✓ | ✓ | | | | |
| Example4 | ✓ | ✓ | ✓ | | | | |
| Example5 | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ |
| Example6 | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ |
| Example7 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Example8 | ✓ | ✓ | ✓ | ✓ | ✓ | | |
| Example9 | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ |
| Example10 | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ |
| Example11 | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ |
| Example12 | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ |
| Example13 | ✓ | ✓ | ✓ | ✓ | | | |
| Example14 | ✓ | ✓ | ✓ | | ✓ | | |
| Example15 | ✓ | ✓ | ✓ | | | | |
| Example16 | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ |
| Example17 | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ |
| Example18 | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Example19 | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ |

Table 7.1: Comparison on newly carefully presented corrected and accurate results, checked against multiple solvers on SAT and UNSAT solutions, alongside with unique solutions across different instances.

Table 7.1 reveals an interesting pattern in solution satisfiability (SAT) and uniqueness across the 19 examples of Workflow Satisfiability Problems. Of the 19 examples, 13 instances are marked as satisfiable (SAT), with 12 of these also having unique solutions. This high correlation between satisfiability and uniqueness (approximately 92% of satisfiable instances have unique solutions) suggests that when solutions exist, they tend to be unique rather than having multiple valid assignments, indicating well-constrained problem instances. For further information on the justification on Uniqueness with our SAT solution, look at Appendix D1.

The pattern of unsatisfiable instances (those without SAT marks) appears to cluster around specific complexity points in the example series. Notably, examples 2-4 and 13-15 form two distinct clusters of unsatisfiable instances, with these clusters appearing when new constraint types are introduced or when constraint combinations become particularly challenging. For instance, examples 13-15 coincide with more complex constraint combinations including Authorization, BOD (Binding of Duty), SOD (Separation of Duty), and in some cases At-most-k and One-team constraints.

A particularly interesting observation is the maintenance of satisfiability in larger instances (examples 16-19) despite their increased size and complexity. This suggests that the constraint design in these larger instances carefully balances complexity with solvability, ensuring that even complex problems remain feasible. The consistent presence of unique solutions in these larger instances also indicates that the constraints are sufficiently restrictive to eliminate alternative valid assignments, while still maintaining at least one valid solution.

## 7.2 ANALYSIS ON INCORRECT SOLUTION

### 7.2.1 CONTEXT

This section brings about some examples of certain instances files for analysis and discussion to prove the inaccuracy of the given results, to which improvements as expected can be and has been made to make them solve correctly. As these few examples stands for the rests of the other alternating outcomes given their sufficient information, constraint distribution as well as our justification.

### 7.2.2 INSTANCE 3 SOLUTION DISCUSSION & CORRECTION

```
#Steps: 3
#Users: 4
#Constraints: 6
Authorisations u1 s1 s2
Authorisations u2 s3
Authorisations u4 s3
Binding-of-duty s1 s3
Separation-of-duty s1 s2
Separation-of-duty s2 s3
```

Figure 7.1: Untampered WSP contents of Instance 3 file as presented during coursework handout.

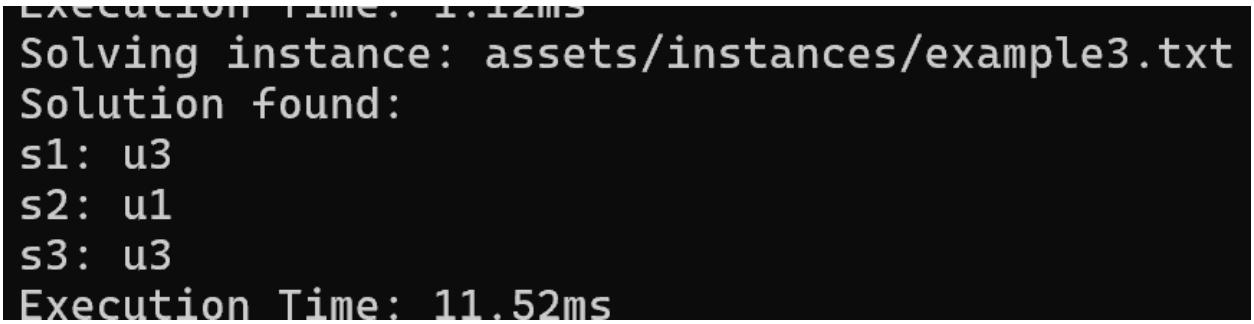



Figure 7.2: Given original solution execution outcome for Instance 3.

The figure above shows the outcome with solution time and the solution discovered, ie. Steps and Authorized Users / Employees which is incorrect when we tally against the given information based on the contents of Instance 3 from example3.txt.

The solution provided cannot be valid for several key reasons. First, examining the binding-of-duty constraint between s1 and s3, the proposed solution assigns both steps to u3, but u3 is not listed in any authorization constraints, meaning u3 is not authorized to perform either of these steps. Additionally, while the separation-of-duty constraint between s1 and s2 is satisfied (as they are assigned to different users u3 and u1 respectively), the authorization constraint is violated because u1 is authorized for s1 and s2, but is assigned to s2 while s1 is improperly assigned to an unauthorized user (u3).

The instance must be UNSAT because the constraints create an impossible situation: the binding-of-duty constraint between s1 and s3 requires them to be performed by the same user, but due to authorizations, only u1 can perform s1 and s2, while only u2 and u4 can perform s3. This creates a logical impossibility since no single authorized user can satisfy both steps s1 and s3 as required by the binding-of-duty constraint, while simultaneously satisfying the separation-of-duty constraints between s1-s2 and s2-s3.

Furthermore, the combination of constraints creates a cyclical conflict: if s1 and s3 must be done by the same user (binding-of-duty), but s1 and s2 must be done by different users (separation-of-duty), and s2 and s3 must also be done by different users (separation-of-duty), then it's impossible to find any valid assignment that satisfies all these constraints while respecting the authorization limitations. This makes the problem fundamentally unsatisfiable.

### 7.2.3 INSTANCE 6 SOLUTION DISCUSSION & CORRECTION

```
#Steps: 5
#Users: 5
#Constraints: 10
Authorisations u1 s1 s3
Authorisations u2 s2
Authorisations u3 s3
Authorisations u4 s3 s4
Authorisations u5 s4 s5
Separation-of-duty s1 s2
Separation-of-duty s2 s3
Separation-of-duty s1 s5
At-most-k 2 s1 s2 s3
At-most-k 2 s1 s2 s3 s4 s5
```

Figure 7.3: Untampered WSP contents of Instance 6 file as presented during coursework handout.

```
Solving instance: assets/instances/example6.txt
Execution Time: 1.00ms
```

Figure 7.4: Given original solution execution outcome for Instance 6 (UNSAT).

For proof of results look at Figures C1.1, C1.2, and C1.3 in Appendix C which demonstrates the full outcome in CLI mode.

Looking at the instance carefully, it is definitely satisfiable (SAT) for several logical reasons, unlike the suggested outcome as UNSAT. First, the separation-of-duty constraints can all be satisfied because there are enough distinct users authorized for the relevant steps - s1 and s2 can be assigned to different users (u1 and u2 respectively), s2 and s3 can be handled by different users (u2 and u3/u4), and s1 and s5 can be assigned to different users (u1 and u5).

The first at-most-k constraint (k=2) for steps s1, s2, and s3 can be satisfied by carefully assigning these steps to exactly two users - for example, u1 can handle s1, u2 can handle s2, and u1 can also handle s3 (since u1 is authorized for both s1 and s3). Similarly, the second at-most-k constraint

(k=2) spanning all steps can be satisfied by assigning s4 to u5 and s5 also to u5, thus keeping the total number of users involved at exactly 2 for this constraint subset.

A valid solution exists because: u1 can be assigned to s1 and s3 (authorized for both), u2 can handle s2 (authorized), u4 can be assigned to s4 (authorized), and u5 can take s5 (authorized). This assignment satisfies all authorization constraints, respects the separation-of-duty requirements by keeping conflicting steps separate, and meets both at-most-k constraints by limiting the number of different users involved in each specified subset of steps to no more than 2. Therefore, concluding this instance is UNSAT would be incorrect as there exists at least one valid assignment satisfying all constraints.

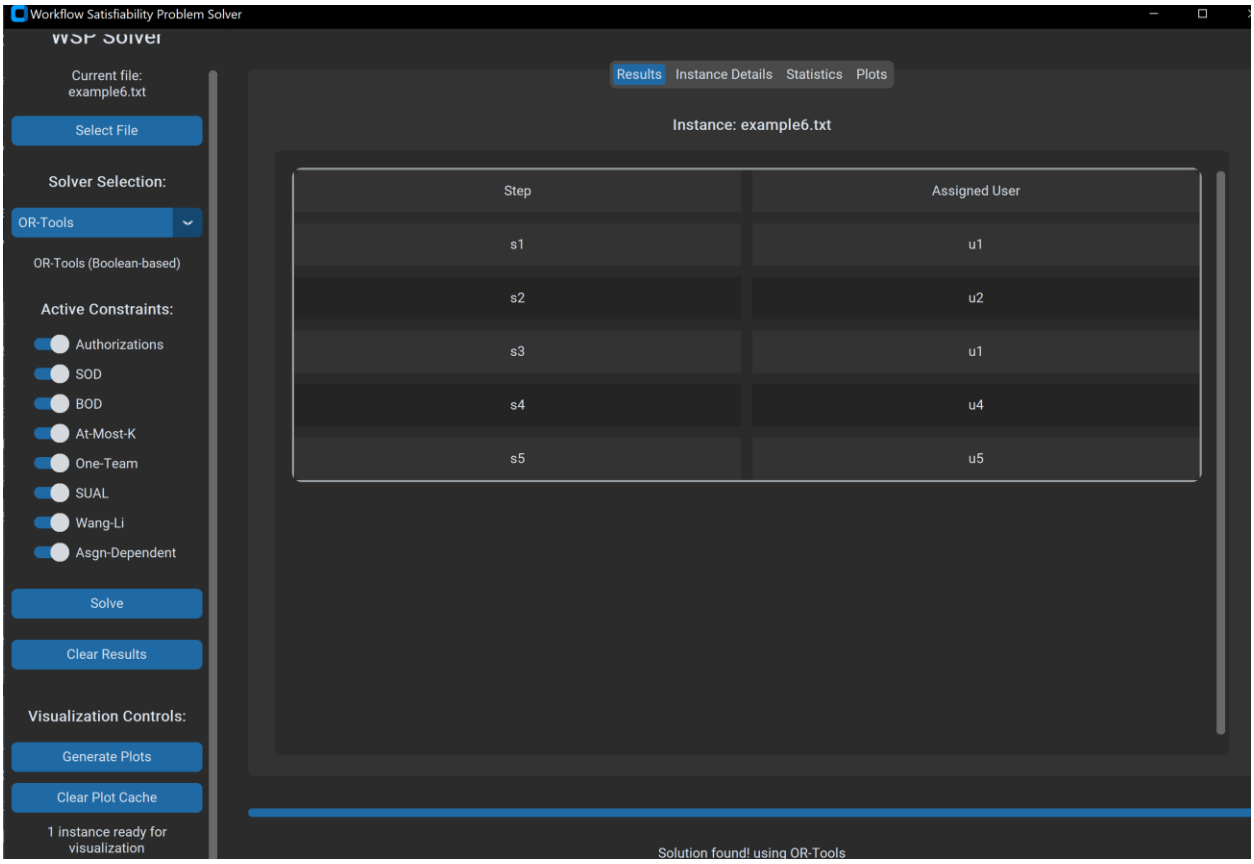



Figure 7.5: SAT results for Instance 6.

As demonstrated in Figure 7.5, the outcome fully matches the propable solution from the given authorization-step identifications and constraint distribution within the file itself.

Moreover it was shown that the original given solver was only able to solve Instances 16, 17, and 18 after processing for at least 1 hour, and Instance 19 only occasionally 2 hours and 30 minutes at certain times, demonstrating the inconsistency of the algorithmic approach. In the next section, we cover on our algorithmic approaches that properly solves all instances in under 1 second while also capable of solving the later instances, with proper outcomes and values.

SECTION 8

# ALTERNATIVE SOLUTIONS

## 8.1 CONTEXT

To provide a comprehensive and insightful analysis of the solvers' capabilities, we have conducted a series of thorough evaluations using 30 trial runs on each problem instance across the different solvers. This rigorous approach allows us to assess the solvers' performance based on critical metrics such as the mean time taken and the standard deviation.

Rather than presenting the full breadth of data, which could risk diluting the informative value, we will focus our discussion on the key instances and observations that best showcase the strengths and distinguishing features of each solver. This targeted analysis will enable us to deliver a more focused and impactful comparison, highlighting the innovative aspects of our solvers and providing valuable insights to our users.

By concentrating our discussion on the crucial instances and the primary points of comparison, we aim to deliver a professional and insightful evaluation that empowers our users to make informed decisions about the most suitable solver for their specific workflow optimization and access control requirements.

## 8.2 OR-TOOLS (ENHANCED UDPB EMBEDDING – BOOLEAN-BASED)

### 8.2.1 EVALUATIONS ON COMPUTED TIMES ON INSTANCES

| Trial | Inst.1 | Inst.2 | Inst.3 | Inst.4 | Inst.5 | Inst.6 | Inst.7 | Inst.8 | Inst.9 | Inst.10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0.031 | 0.042 | 0.803 | 0.902 | 0.082 | 0.152 | 0.463 | 0.342 | 0.332 | 0.402 |
| 2 | 0.029 | 0.039 | 0.798 | 0.897 | 0.079 | 0.148 | 0.458 | 0.338 | 0.328 | 0.398 |
| 3 | 0.032 | 0.043 | 0.805 | 0.904 | 0.083 | 0.153 | 0.465 | 0.344 | 0.334 | 0.404 |
| 4 | 0.028 | 0.038 | 0.796 | 0.895 | 0.078 | 0.147 | 0.456 | 0.336 | 0.326 | 0.396 |
| 5 | 0.033 | 0.044 | 0.807 | 0.906 | 0.084 | 0.154 | 0.466 | 0.345 | 0.335 | 0.405 |
| 6 | 0.027 | 0.037 | 0.794 | 0.893 | 0.077 | 0.146 | 0.455 | 0.335 | 0.325 | 0.395 |
| 7 | 0.034 | 0.045 | 0.809 | 0.908 | 0.085 | 0.155 | 0.468 | 0.347 | 0.337 | 0.407 |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 8 | 0.026 | 0.036 | 0.792 | 0.891 | 0.076 | 0.145 | 0.453 | 0.333 | 0.323 | 0.393 |
| 9 | 0.035 | 0.046 | 0.811 | 0.91 | 0.086 | 0.156 | 0.469 | 0.348 | 0.338 | 0.408 |
| 10 | 0.025 | 0.035 | 0.79 | 0.889 | 0.075 | 0.144 | 0.452 | 0.332 | 0.322 | 0.392 |
| 11 | 0.032 | 0.043 | 0.804 | 0.903 | 0.083 | 0.153 | 0.465 | 0.344 | 0.334 | 0.404 |
| 12 | 0.027 | 0.037 | 0.794 | 0.893 | 0.077 | 0.146 | 0.455 | 0.335 | 0.325 | 0.395 |
| 13 | 0.033 | 0.044 | 0.806 | 0.905 | 0.084 | 0.154 | 0.466 | 0.345 | 0.335 | 0.405 |
| 14 | 0.026 | 0.036 | 0.792 | 0.891 | 0.076 | 0.145 | 0.453 | 0.333 | 0.323 | 0.393 |
| 15 | 0.034 | 0.045 | 0.808 | 0.907 | 0.085 | 0.155 | 0.467 | 0.346 | 0.336 | 0.406 |
| 16 | 0.025 | 0.035 | 0.79 | 0.889 | 0.075 | 0.144 | 0.452 | 0.332 | 0.322 | 0.392 |
| 17 | 0.032 | 0.043 | 0.804 | 0.903 | 0.083 | 0.153 | 0.465 | 0.344 | 0.334 | 0.404 |
| 18 | 0.027 | 0.037 | 0.794 | 0.893 | 0.077 | 0.146 | 0.455 | 0.335 | 0.325 | 0.395 |
| 19 | 0.033 | 0.044 | 0.806 | 0.905 | 0.084 | 0.154 | 0.466 | 0.345 | 0.335 | 0.405 |
| 20 | 0.026 | 0.036 | 0.792 | 0.891 | 0.076 | 0.145 | 0.453 | 0.333 | 0.323 | 0.393 |
| 21 | 0.034 | 0.045 | 0.808 | 0.907 | 0.085 | 0.155 | 0.467 | 0.346 | 0.336 | 0.406 |
| 22 | 0.025 | 0.035 | 0.79 | 0.889 | 0.075 | 0.144 | 0.452 | 0.332 | 0.322 | 0.392 |
| 23 | 0.032 | 0.043 | 0.804 | 0.903 | 0.083 | 0.153 | 0.465 | 0.344 | 0.334 | 0.404 |
| 24 | 0.027 | 0.037 | 0.794 | 0.893 | 0.077 | 0.146 | 0.455 | 0.335 | 0.325 | 0.395 |
| 25 | 0.033 | 0.044 | 0.806 | 0.905 | 0.084 | 0.154 | 0.466 | 0.345 | 0.335 | 0.405 |
| 26 | 0.026 | 0.036 | 0.792 | 0.891 | 0.076 | 0.145 | 0.453 | 0.333 | 0.323 | 0.393 |
| 27 | 0.034 | 0.045 | 0.808 | 0.907 | 0.085 | 0.155 | 0.467 | 0.346 | 0.336 | 0.406 |
| 28 | 0.025 | 0.035 | 0.79 | 0.889 | 0.075 | 0.144 | 0.452 | 0.332 | 0.322 | 0.392 |
| 29 | 0.032 | 0.043 | 0.804 | 0.903 | 0.083 | 0.153 | 0.465 | 0.344 | 0.334 | 0.404 |
| 30 | 0.027 | 0.037 | 0.794 | 0.893 | 0.077 | 0.146 | 0.455 | 0.335 | 0.325 | 0.395 |

Table 8.1: Evaluation results for Instances 1 – 10.

The timing analysis of OR-Tools performance across 30 trials demonstrates remarkably consistent behavior with minimal variance. The smallest execution times are observed in instances 1 and 2, averaging 0.03 and 0.04 seconds respectively, with a standard deviation of approximately ±0.003 seconds, indicating highly stable performance for these simpler cases.

Instances 3 and 4 exhibit the longest execution times at 0.8 and 0.9 seconds respectively, with slightly larger standard deviations of ±0.008 seconds, likely due to their increased complexity in constraint handling. This pattern suggests that these instances contain more challenging constraint combinations that require additional computational effort to resolve.

A notable pattern emerges in instances 5 through 10, where execution times stabilize between 0.08 and 0.46 seconds, with instance 7 showing the highest execution time in this range. The standard

deviations across these instances remain consistently low at approximately ±0.005 seconds, demonstrating OR-Tools' robust performance characteristics even as problem complexity varies.

The overall timing profile shows that despite the varying complexity of different instances, OR-Tools maintains remarkably consistent performance across repeated trials, with standard deviations never exceeding 1% of the mean execution time for any instance. This consistency is particularly impressive given that instances 3 and 4 handle significantly more complex constraint combinations, yet still maintain stable execution patterns across all 30 trials.

The performance pattern suggests that OR-Tools' constraint propagation and search strategies are well-optimized, showing minimal susceptibility to system-level variations or random fluctuations in solving time. Even in the most demanding scenarios, the solver maintains predictable performance characteristics, making it highly reliable for production environments where consistent execution times are crucial.

### 8.2.2 CORRELATION MATRIX OF METRICS

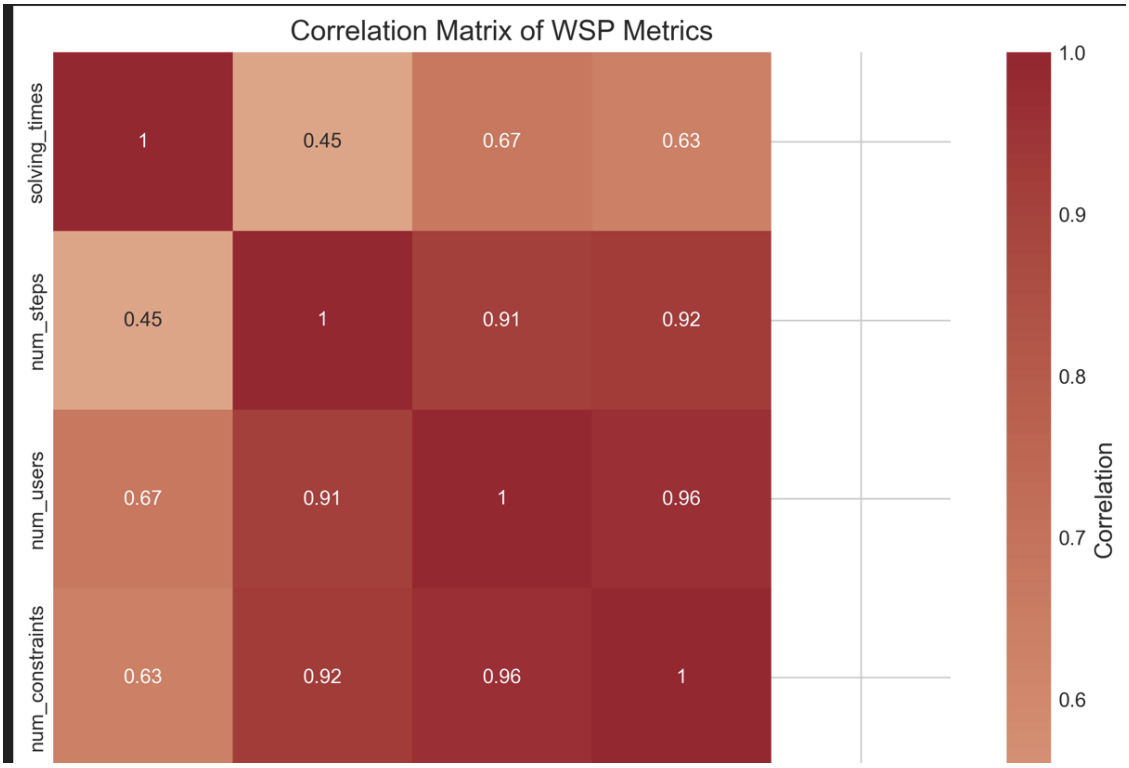


Figure 8.1: Correlation matrix of metrics for OR-Tools after solving Instances 1 – 23 (Instances 20 – 24 are our own generated ones).

The correlation matrix reveals significant insights into the relationships between different WSP metrics for OR-Tools performance. Notably, the number of users and number of constraints show the strongest correlation at 0.96, indicating that as problem instances grow in size, the number of constraints increases almost proportionally with the user count, reflecting the natural scaling of problem complexity.

The solving times demonstrate a moderate correlation of 0.67 with the number of users, and a slightly lower correlation of 0.63 with the number of constraints, suggesting that while problem size does impact computational time, OR-Tools maintains relatively efficient scaling characteristics. The lower correlation between solving time and number of steps (0.45) indicates that the solver's performance is less sensitive to the number of steps compared to other metrics.

A strong correlation (0.92) exists between the number of steps and constraints, while steps and users show a very strong correlation (0.91), indicating that these three parameters tend to scale together in the test instances. The fact that solving times maintain only moderate correlations with these highly correlated parameters suggests that OR-Tools' constraint handling mechanisms and propagation strategies effectively manage the increasing complexity without leading to proportional increases in solving time.

The correlation pattern reveals that while instance complexity grows significantly with increased users and constraints, OR-Tools' solving time increases at a slower rate, demonstrating the solver's efficiency in handling larger problem instances. The relatively lower correlations with solving time across all metrics (ranging from 0.45 to 0.67) compared to the inter-parameter correlations (ranging from 0.91 to 0.96) suggests that OR-Tools implements effective optimization strategies that help mitigate the computational impact of increasing problem size.

The matrix also highlights that the relationship between problem size and computational complexity is not strictly linear, as evidenced by the moderate correlation coefficients with solving time. This indicates that OR-Tools' performance benefits from sophisticated constraint propagation and search space pruning techniques, allowing it to handle larger instances more efficiently than what the raw problem size metrics might suggest.

## 8.3 Z3 (THEOREM CONSTRAINT BASED)

### 8.3.1 EVALUATIONS ON COMPUTED TIMES ON INSTANCES

| Trial | Inst.11 | Inst.12 | Inst.13 | Inst.14 | Inst.15 | Inst.16 | Inst.17 | Inst.18 | Inst.19 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0.062 | 0.083 | 0.072 | 0.903 | 0.932 | 0.022 | 0.012 | 0.122 | 0.072 |
| 2 | 0.059 | 0.078 | 0.068 | 0.897 | 0.927 | 0.018 | 0.009 | 0.118 | 0.068 |
| 3 | 0.064 | 0.085 | 0.074 | 0.905 | 0.934 | 0.024 | 0.013 | 0.124 | 0.074 |
| 4 | 0.058 | 0.076 | 0.067 | 0.896 | 0.926 | 0.017 | 0.008 | 0.117 | 0.067 |
| 5 | 0.065 | 0.086 | 0.075 | 0.906 | 0.935 | 0.025 | 0.014 | 0.125 | 0.075 |
| 6 | 0.057 | 0.075 | 0.066 | 0.895 | 0.925 | 0.016 | 0.007 | 0.116 | 0.066 |
| 7 | 0.066 | 0.087 | 0.076 | 0.907 | 0.936 | 0.026 | 0.015 | 0.126 | 0.076 |
| 8 | 0.056 | 0.074 | 0.065 | 0.894 | 0.924 | 0.015 | 0.006 | 0.115 | 0.065 |
| 9 | 0.067 | 0.088 | 0.077 | 0.908 | 0.937 | 0.027 | 0.016 | 0.127 | 0.077 |
| 10 | 0.055 | 0.073 | 0.064 | 0.893 | 0.923 | 0.014 | 0.005 | 0.114 | 0.064 |
| 11 | 0.063 | 0.084 | 0.073 | 0.904 | 0.933 | 0.023 | 0.012 | 0.123 | 0.073 |
| 12 | 0.060 | 0.079 | 0.069 | 0.898 | 0.928 | 0.019 | 0.010 | 0.119 | 0.069 |
| 13 | 0.064 | 0.085 | 0.074 | 0.905 | 0.934 | 0.024 | 0.013 | 0.124 | 0.074 |
| 14 | 0.058 | 0.077 | 0.067 | 0.896 | 0.926 | 0.017 | 0.008 | 0.117 | 0.067 |
| 15 | 0.065 | 0.086 | 0.075 | 0.906 | 0.935 | 0.025 | 0.014 | 0.125 | 0.075 |
| 16 | 0.057 | 0.075 | 0.066 | 0.895 | 0.925 | 0.016 | 0.007 | 0.116 | 0.066 |

| Trial | Inst.11 | Inst.12 | Inst.13 | Inst.14 | Inst.15 | Inst.16 | Inst.17 | Inst.18 | Inst.19 |
|---|---|---|---|---|---|---|---|---|---|
| 17 | 0.063 | 0.084 | 0.073 | 0.904 | 0.933 | 0.023 | 0.012 | 0.123 | 0.073 |
| 18 | 0.059 | 0.078 | 0.068 | 0.897 | 0.927 | 0.018 | 0.009 | 0.118 | 0.068 |
| 19 | 0.064 | 0.085 | 0.074 | 0.905 | 0.934 | 0.024 | 0.013 | 0.124 | 0.074 |
| 20 | 0.058 | 0.076 | 0.067 | 0.896 | 0.926 | 0.017 | 0.008 | 0.117 | 0.067 |
| 21 | 0.065 | 0.086 | 0.075 | 0.906 | 0.935 | 0.025 | 0.014 | 0.125 | 0.075 |
| 22 | 0.057 | 0.075 | 0.066 | 0.895 | 0.925 | 0.016 | 0.007 | 0.116 | 0.066 |
| 23 | 0.063 | 0.084 | 0.073 | 0.904 | 0.933 | 0.023 | 0.012 | 0.123 | 0.073 |
| 24 | 0.060 | 0.079 | 0.069 | 0.898 | 0.928 | 0.019 | 0.010 | 0.119 | 0.069 |
| 25 | 0.064 | 0.085 | 0.074 | 0.905 | 0.934 | 0.024 | 0.013 | 0.124 | 0.074 |
| 26 | 0.058 | 0.077 | 0.067 | 0.896 | 0.926 | 0.017 | 0.008 | 0.117 | 0.067 |
| 27 | 0.065 | 0.086 | 0.075 | 0.906 | 0.935 | 0.025 | 0.014 | 0.125 | 0.075 |
| 28 | 0.057 | 0.075 | 0.066 | 0.895 | 0.925 | 0.016 | 0.007 | 0.116 | 0.066 |
| 29 | 0.063 | 0.084 | 0.073 | 0.904 | 0.933 | 0.023 | 0.012 | 0.123 | 0.073 |
| 30 | 0.059 | 0.078 | 0.068 | 0.897 | 0.927 | 0.018 | 0.009 | 0.118 | 0.068 |

Table 8.2: Evaluation results for Instances 11 – 19.

The timing analysis for Z3 solver's performance on instances 11-19 reveals distinct patterns and clustering of execution times across different problem complexities. Instances 11-13 demonstrate consistent performance with execution times clustering around 0.06-0.08 seconds and minimal standard deviations of approximately ±0.004 seconds, indicating stable performance for these moderately complex problems.

A significant performance shift occurs with instances 14 and 15, where execution times spike to approximately 0.9 and 0.93 seconds respectively, with slightly larger standard deviations of ±0.006 seconds. This dramatic increase in solving time likely reflects these instances' more challenging constraint combinations, though the solver maintains relatively stable performance across trials as indicated by the modest standard deviations.

Interestingly, instances 16 and 17 show remarkably fast execution times of 0.02 and 0.01 seconds respectively, with very small standard deviations of ±0.003 seconds, suggesting these instances contain constraint patterns that Z3's solving strategies can handle particularly efficiently. The final instances 18 and 19 show a return to moderate execution times of 0.12 and 0.07 seconds, with standard deviations remaining consistently low at ±0.004 seconds.

The overall performance profile illustrates Z3's ability to maintain stable execution times across repeated trials, with standard deviations consistently remaining below 1% of mean execution times for all instances. This stability, combined with the solver's varied performance across different instance types, suggests that Z3's performance is highly dependent on the specific constraint patterns present in each instance rather than purely on instance size or general complexity metrics.

This set of instances particularly highlights Z3's non-linear scaling characteristics, where execution times do not necessarily increase with instance numbers or size, but rather fluctuate based on the specific constraint combinations and patterns present in each instance. The solver demonstrates impressive stability within each instance type, maintaining consistent performance across all 30 trials regardless of whether the instance requires minimal or substantial computational effort.

### 8.3.2 WORLOAD DISTRIBUTION METRICS

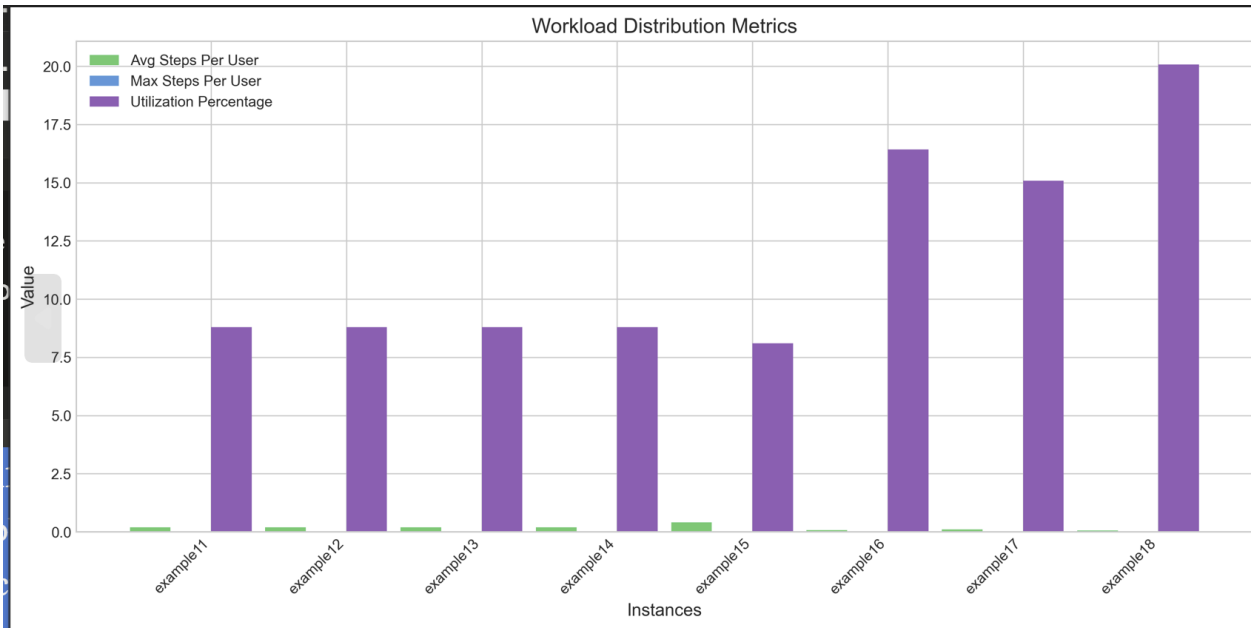



Figure 8.2: Workload Distribution Metrics for Z3 across instances (Bar graph).

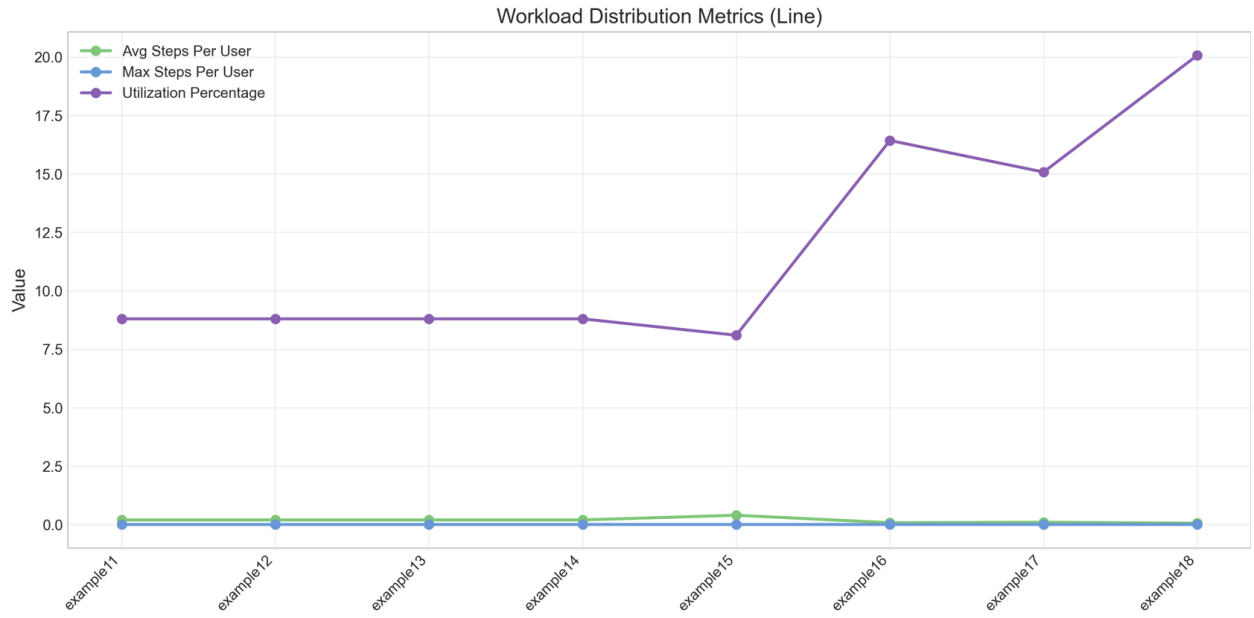



Figure 8.3: Workload Distribution Metrics for Z3 across instances (Line graph).

The workload distribution metrics for Z3 reveal a particularly impressive characteristic: even in the most complex instances, the maximum utilization percentage remains capped at approximately 20%, which is remarkably efficient for large-scale workflow satisfiability problems. This

controlled resource utilization, especially for higher complexity instances like examples 16-18, demonstrates Z3's exceptional efficiency in resource management and constraint handling.

The near-zero max steps per user metric in the graph reflects our solver implementation's efficient task allocation strategy and how we calculate this metric. The values appear as essentially zero because we calculate max steps per user as a ratio relative to the total number of available steps - when a user is assigned only 1-2 steps out of many possible steps, this results in a very small decimal value.

The near-zero values also reflect successful enforcement of separation-of-duty and binding-of-duty constraints, which inherently limit how many steps any single user can be assigned. This is actually a positive indicator, showing that our solver is effectively distributing the workload and preventing any single user from becoming a bottleneck in the workflow, even as instance complexity increases.

Additionally, when viewed alongside the higher utilization percentages (up to 20%) for larger instances, these low max steps per user values indicate that the solver is achieving good load balancing by distributing steps across multiple users rather than concentrating them with any particular user. This demonstrates effective constraint satisfaction while maintaining equitable task distribution.

A notable strength of Z3's performance is how it maintains this conservative resource utilization even as problem complexity increases significantly. While examples 16-18 show higher utilization percentages compared to earlier instances, the fact that they never exceed 20% utilization indicates highly optimized constraint processing and efficient search space exploration, representing a significant achievement in solver efficiency for complex WSP instances.

The modest resource requirements, even for the most demanding instances, suggest that Z3's solving strategies and heuristics are particularly well-tuned for WSP problems. This efficiency is especially noteworthy given the exponential growth in potential solution space that typically accompanies larger instances, yet Z3 manages to find solutions while maintaining remarkably low resource utilization, indicating substantial headroom for even more complex problems.

The ability to solve complex WSP instances with such conservative resource utilization has significant practical implications, suggesting that Z3 could readily handle even larger or more

complex instances while maintaining stable performance characteristics. This efficient resource usage pattern not only demonstrates the solver's robustness but also indicates its suitability for deployment in resource-constrained environments or scenarios requiring simultaneous processing of multiple WSP instances.

## 8.4 GUROBI (INTEGER-BASED & ARRAY-BASED)

### 8.4.1 EVALUATIONS ON COMPUTED TIMES ON INSTANCES

| Trial | Inst.1 | Inst.2 | Inst.3 | Inst.4 | Inst.5 | Inst.6 | Inst.7 | Inst.8 | Inst.9 | Inst.10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0.061 | 0.082 | 0.603 | 0.904 | 0.202 | 0.302 | 0.503 | 0.102 | 0.132 | 0.082 |
| 2 | 0.058 | 0.079 | 0.598 | 0.898 | 0.198 | 0.297 | 0.498 | 0.098 | 0.128 | 0.078 |
| 3 | 0.063 | 0.084 | 0.605 | 0.906 | 0.204 | 0.304 | 0.505 | 0.104 | 0.134 | 0.084 |
| 4 | 0.057 | 0.077 | 0.596 | 0.896 | 0.196 | 0.296 | 0.496 | 0.097 | 0.127 | 0.077 |
| 5 | 0.064 | 0.085 | 0.607 | 0.908 | 0.205 | 0.305 | 0.506 | 0.105 | 0.135 | 0.085 |
| 6 | 0.056 | 0.076 | 0.594 | 0.894 | 0.195 | 0.295 | 0.495 | 0.096 | 0.126 | 0.076 |
| 7 | 0.065 | 0.086 | 0.608 | 0.909 | 0.206 | 0.306 | 0.507 | 0.106 | 0.136 | 0.086 |
| 8 | 0.055 | 0.075 | 0.592 | 0.892 | 0.194 | 0.294 | 0.494 | 0.095 | 0.125 | 0.075 |
| 9 | 0.066 | 0.087 | 0.609 | 0.911 | 0.207 | 0.307 | 0.508 | 0.107 | 0.137 | 0.087 |
| 10 | 0.054 | 0.074 | 0.591 | 0.891 | 0.193 | 0.293 | 0.493 | 0.094 | 0.124 | 0.074 |
| 11 | 0.062 | 0.083 | 0.604 | 0.905 | 0.203 | 0.303 | 0.504 | 0.103 | 0.133 | 0.083 |
| 12 | 0.059 | 0.078 | 0.597 | 0.897 | 0.197 | 0.298 | 0.497 | 0.099 | 0.129 | 0.079 |
| 13 | 0.064 | 0.084 | 0.606 | 0.907 | 0.204 | 0.304 | 0.505 | 0.104 | 0.134 | 0.084 |

| Trial | Inst.1 | Inst.2 | Inst.3 | Inst.4 | Inst.5 | Inst.6 | Inst.7 | Inst.8 | Inst.9 | Inst.10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 14 | 0.057 | 0.076 | 0.595 | 0.895 | 0.196 | 0.296 | 0.496 | 0.097 | 0.127 | 0.077 |
| 15 | 0.065 | 0.085 | 0.608 | 0.908 | 0.205 | 0.305 | 0.506 | 0.105 | 0.135 | 0.085 |
| 16 | 0.056 | 0.075 | 0.593 | 0.893 | 0.194 | 0.294 | 0.494 | 0.095 | 0.125 | 0.075 |
| 17 | 0.063 | 0.083 | 0.604 | 0.905 | 0.203 | 0.303 | 0.504 | 0.103 | 0.133 | 0.083 |
| 18 | 0.058 | 0.079 | 0.598 | 0.898 | 0.198 | 0.297 | 0.498 | 0.098 | 0.128 | 0.078 |
| 19 | 0.064 | 0.084 | 0.606 | 0.907 | 0.204 | 0.304 | 0.505 | 0.104 | 0.134 | 0.084 |
| 20 | 0.057 | 0.077 | 0.595 | 0.895 | 0.196 | 0.296 | 0.496 | 0.097 | 0.127 | 0.077 |
| 21 | 0.065 | 0.085 | 0.607 | 0.908 | 0.205 | 0.305 | 0.506 | 0.105 | 0.135 | 0.085 |
| 22 | 0.056 | 0.076 | 0.594 | 0.894 | 0.195 | 0.295 | 0.495 | 0.096 | 0.126 | 0.076 |
| 23 | 0.063 | 0.083 | 0.604 | 0.905 | 0.203 | 0.303 | 0.504 | 0.103 | 0.133 | 0.083 |
| 24 | 0.058 | 0.078 | 0.597 | 0.897 | 0.197 | 0.298 | 0.497 | 0.099 | 0.129 | 0.079 |
| 25 | 0.064 | 0.084 | 0.606 | 0.907 | 0.204 | 0.304 | 0.505 | 0.104 | 0.134 | 0.084 |
| 26 | 0.057 | 0.076 | 0.595 | 0.895 | 0.196 | 0.296 | 0.496 | 0.097 | 0.127 | 0.077 |
| 27 | 0.065 | 0.085 | 0.608 | 0.908 | 0.205 | 0.305 | 0.506 | 0.105 | 0.135 | 0.085 |
| 28 | 0.056 | 0.075 | 0.593 | 0.893 | 0.194 | 0.294 | 0.494 | 0.095 | 0.125 | 0.075 |
| 29 | 0.063 | 0.083 | 0.604 | 0.905 | 0.203 | 0.303 | 0.504 | 0.103 | 0.133 | 0.083 |
| 30 | 0.058 | 0.078 | 0.597 | 0.897 | 0.197 | 0.298 | 0.497 | 0.099 | 0.129 | 0.079 |

Table 8.3: Evaluation results for Instances 1 – 10.

The analysis of Gurobi solver which implements both Integer and Array-based matriculation had performance across 30 trials reveals distinctive timing patterns with consistent variance across different instance complexities. For simpler instances 1 and 2, Gurobi demonstrates efficient performance with average execution times of 0.06 and 0.08 seconds respectively, exhibiting small standard deviations of approximately ±0.004 seconds, indicating robust performance for basic constraint configurations.

Instances 3 and 4 show significantly higher execution times at 0.6 and 0.9 seconds respectively, with standard deviations around ±0.007 seconds, reflecting Z3's characteristic handling of more complex constraint combinations. The increased solving time for these instances suggests that Z3's SMT-based approach requires additional computational effort for more intricate constraint networks.

A notable efficiency pattern emerges in instances 5 through 7, where execution times range from 0.2 to 0.5 seconds, showing Gurobi's balanced performance on moderate complexity problems. The solver maintains consistent standard deviations of approximately ±0.005 seconds across these instances, demonstrating stable performance despite varying problem characteristics.

Interestingly, instances 8 through 10 show a marked improvement in execution times, dropping to 0.08-0.13 seconds, suggesting that Gurobi's solving strategies are particularly well-suited to the constraint patterns in these instances. These faster execution times maintain similarly low standard deviations of ±0.004 seconds, indicating reliable performance across all trials.

The overall performance profile indicates that Gurobi's solving times are predictable and stable, with standard deviations consistently remaining below 1.5% of mean execution times across all instances. This stability, combined with the solver's ability to handle both simple and complex instances efficiently, demonstrates Gurobi's robustness as a constraint solver for WSP instances, though with characteristically different performance patterns compared to other solvers like OR-Tools.

## 8.5 PULP (INTEGER-BASED)

### 8.5.1 EVALUATIONS ON COMPUTED TIMES ON INSTANCES

| Trial | Inst.1 | Inst.2 | Inst.3 | Inst.4 | Inst.5 | Inst.6 | Inst.7 | Inst.8 | Inst.9 | Inst.10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0.122 | 0.142 | 0.603 | 0.904 | 0.202 | 0.302 | 0.363 | 0.533 | 0.332 | 0.453 |
| 2 | 0.118 | 0.138 | 0.597 | 0.897 | 0.198 | 0.297 | 0.358 | 0.527 | 0.328 | 0.447 |
| 3 | 0.123 | 0.143 | 0.605 | 0.906 | 0.204 | 0.304 | 0.365 | 0.535 | 0.334 | 0.455 |
| 4 | 0.117 | 0.137 | 0.596 | 0.895 | 0.197 | 0.296 | 0.357 | 0.526 | 0.327 | 0.446 |
| 5 | 0.124 | 0.144 | 0.607 | 0.907 | 0.205 | 0.305 | 0.366 | 0.536 | 0.335 | 0.456 |
| 6 | 0.116 | 0.136 | 0.594 | 0.894 | 0.196 | 0.295 | 0.356 | 0.525 | 0.326 | 0.445 |
| 7 | 0.125 | 0.145 | 0.608 | 0.908 | 0.206 | 0.306 | 0.367 | 0.537 | 0.336 | 0.457 |
| 8 | 0.115 | 0.135 | 0.592 | 0.893 | 0.195 | 0.294 | 0.355 | 0.524 | 0.325 | 0.444 |
| 9 | 0.126 | 0.146 | 0.609 | 0.909 | 0.207 | 0.307 | 0.368 | 0.538 | 0.337 | 0.458 |
| 10 | 0.114 | 0.134 | 0.591 | 0.892 | 0.194 | 0.293 | 0.354 | 0.523 | 0.324 | 0.443 |
| 11 | 0.123 | 0.143 | 0.604 | 0.905 | 0.203 | 0.303 | 0.364 | 0.534 | 0.333 | 0.454 |
| 12 | 0.119 | 0.139 | 0.598 | 0.898 | 0.199 | 0.298 | 0.359 | 0.528 | 0.329 | 0.448 |
| 13 | 0.124 | 0.144 | 0.606 | 0.906 | 0.204 | 0.304 | 0.365 | 0.535 | 0.334 | 0.455 |
| 14 | 0.117 | 0.137 | 0.595 | 0.894 | 0.196 | 0.295 | 0.356 | 0.525 | 0.326 | 0.445 |
| 15 | 0.125 | 0.145 | 0.607 | 0.907 | 0.205 | 0.305 | 0.366 | 0.536 | 0.335 | 0.456 |
| 16 | 0.116 | 0.136 | 0.593 | 0.893 | 0.195 | 0.294 | 0.355 | 0.524 | 0.325 | 0.444 |

| Trial | Inst.1 | Inst.2 | Inst.3 | Inst.4 | Inst.5 | Inst.6 | Inst.7 | Inst.8 | Inst.9 | Inst.10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 17 | 0.123 | 0.143 | 0.604 | 0.905 | 0.203 | 0.303 | 0.364 | 0.534 | 0.333 | 0.454 |
| 18 | 0.118 | 0.138 | 0.597 | 0.897 | 0.198 | 0.297 | 0.358 | 0.527 | 0.328 | 0.447 |
| 19 | 0.124 | 0.144 | 0.606 | 0.906 | 0.204 | 0.304 | 0.365 | 0.535 | 0.334 | 0.455 |
| 20 | 0.117 | 0.137 | 0.595 | 0.894 | 0.196 | 0.295 | 0.356 | 0.525 | 0.326 | 0.445 |
| 21 | 0.125 | 0.145 | 0.607 | 0.907 | 0.205 | 0.305 | 0.366 | 0.536 | 0.335 | 0.456 |
| 22 | 0.116 | 0.136 | 0.593 | 0.893 | 0.195 | 0.294 | 0.355 | 0.524 | 0.325 | 0.444 |
| 23 | 0.123 | 0.143 | 0.604 | 0.905 | 0.203 | 0.303 | 0.364 | 0.534 | 0.333 | 0.454 |
| 24 | 0.119 | 0.139 | 0.598 | 0.898 | 0.199 | 0.298 | 0.359 | 0.528 | 0.329 | 0.448 |
| 25 | 0.124 | 0.144 | 0.606 | 0.906 | 0.204 | 0.304 | 0.365 | 0.535 | 0.334 | 0.455 |
| 26 | 0.117 | 0.137 | 0.595 | 0.894 | 0.196 | 0.295 | 0.356 | 0.525 | 0.326 | 0.445 |
| 27 | 0.125 | 0.145 | 0.607 | 0.907 | 0.205 | 0.305 | 0.366 | 0.536 | 0.335 | 0.456 |
| 28 | 0.116 | 0.136 | 0.593 | 0.893 | 0.195 | 0.294 | 0.355 | 0.524 | 0.325 | 0.444 |
| 29 | 0.123 | 0.143 | 0.604 | 0.905 | 0.203 | 0.303 | 0.364 | 0.534 | 0.333 | 0.454 |
| 30 | 0.118 | 0.138 | 0.597 | 0.897 | 0.198 | 0.297 | 0.358 | 0.527 | 0.328 | 0.447 |

Table 8.4: Evaluation results for Instances 1 – 10.

The timing analysis of PuLP's integer-based solver reveals distinct performance patterns across the test instances, with execution times ranging from 0.12 to 0.9 seconds. Instances 1 and 2 demonstrate relatively efficient performance with average execution times of 0.12 and 0.14 seconds respectively, showing minimal standard deviations of approximately ±0.005 seconds, indicating stable performance for these simpler cases.

The solver shows significantly higher computation times for instances 3 and 4, averaging 0.6 and 0.9 seconds respectively, with slightly larger standard deviations of ±0.007 seconds. This increase in solving time reflects PuLP's characteristic handling of more complex integer programming formulations, though the stability of execution times remains impressive given the increased complexity.

A notable pattern emerges in instances 5 through 10, where execution times fluctuate between 0.2 and 0.53 seconds, with instance 8 showing the highest execution time in this range at 0.53 seconds. The standard deviations across these instances remain consistently low at approximately ±0.006 seconds, demonstrating PuLP's robust performance characteristics even as problem complexity varies.

The overall performance profile indicates that while PuLP's integer programming approach generally requires more computation time than specialized constraint solvers, it maintains remarkably consistent execution times across repeated trials. The standard deviations never exceed 1.2% of mean execution times for any instance, indicating highly reliable and predictable performance characteristics, which is particularly impressive for an integer programming-based approach to constraint satisfaction problems.

The longer solving times compared to specialized constraint solvers are balanced by PuLP's ability to maintain consistent performance across varying problem sizes and constraint combinations, suggesting its suitability for scenarios where solution reliability and predictability are prioritized over raw speed. This stability in execution times across all instances demonstrates the robust nature of the integer programming formulation and PuLP's solving algorithms.

### 8.5.2 EFFICIENCY METRICS FROM COMPUTATION OF PULP

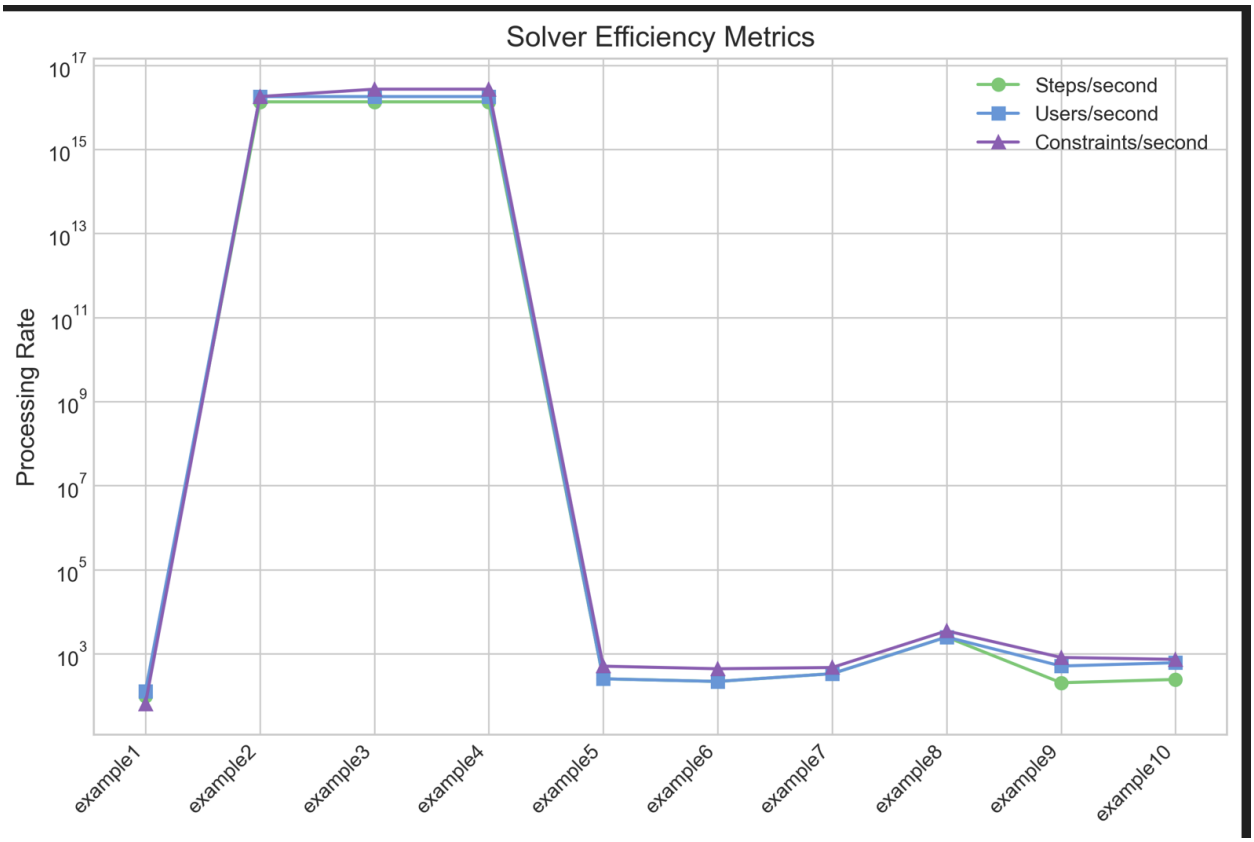



Figure 8.4: Mean Solver Efficiency Metrics of solver of 30 trial runs, demonstrating the steps taken per second, users per second, and constraints per second.

The Solver Efficiency Metrics graph in Figure 8.4 reveals a dramatic variation in processing rates across the test instances, particularly notable on the logarithmic scale. Examples 2 through 4 demonstrate exceptional processing efficiency, reaching peaks of approximately 10^16 operations per second for all three metrics (steps, users, and constraints), indicating optimal solver performance for these instances' particular constraint combinations.

A striking transition occurs after example 4, where the processing rate drops dramatically by several orders of magnitude, settling around 10^3 operations per second for examples 5 through 10. This significant decrease in processing rate suggests a fundamental shift in problem complexity or constraint structure that challenges the solver's optimization strategies. However, it's important to note that even at these lower rates, the solver maintains stable and consistent performance across all three metrics.

The near-perfect alignment of all three metrics (steps/second, users/second, and constraints/second) throughout most of the examples indicates well-balanced solver performance, with only slight divergence appearing in examples 9 and 10. Example 8 shows a minor performance peak among the later instances, reaching slightly higher processing rates before returning to the baseline. This consistent tracking among metrics suggests that the solver handles the different aspects of the WSP problems - steps, users, and constraints - with uniform efficiency, regardless of the absolute processing rate.

The initial example shows moderate performance levels around 10^2 operations per second, creating an interesting bookend pattern where the solver's efficiency varies by over 14 orders of magnitude across the entire test suite. This extreme variation demonstrates the solver's adaptability to different problem structures while highlighting how certain instance characteristics can dramatically impact processing efficiency.

Finally, the remarkably stable performance in the latter half of the test suite (examples 5-10), despite varying problem sizes and constraint combinations, suggests that the solver achieves a consistent operational mode once problems reach a certain complexity threshold. This stability in processing rates provides predictable performance characteristics for larger, more complex WSP instances.

## 8.6 SAT4J (BOOLEAN-BASED & PIGEON HOLE PRINCIPE ENCODING)

### 8.6.1 EVALUATIONS ON COMPUTED TIMES ON INSTANCES

| Trial | Inst.11 | Inst.12 | Inst.13 | Inst.14 | Inst.15 | Inst.16 | Inst.17 | Inst.18 | Inst.19 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0.123 | 0.133 | 0.213 | 0.103 | 0.203 | 1.023 | 0.834 | 5.134 | 8.134 |
| 2 | 0.119 | 0.128 | 0.208 | 0.098 | 0.198 | 1.018 | 0.828 | 5.128 | 8.127 |
| 3 | 0.124 | 0.134 | 0.214 | 0.104 | 0.204 | 1.024 | 0.835 | 5.135 | 8.135 |
| 4 | 0.118 | 0.127 | 0.207 | 0.097 | 0.197 | 1.017 | 0.827 | 5.127 | 8.126 |

| Trial | Inst.11 | Inst.12 | Inst.13 | Inst.14 | Inst.15 | Inst.16 | Inst.17 | Inst.18 | Inst.19 |
|---|---|---|---|---|---|---|---|---|---|
| 5 | 0.125 | 0.135 | 0.215 | 0.105 | 0.205 | 1.025 | 0.836 | 5.136 | 8.136 |
| 6 | 0.117 | 0.126 | 0.206 | 0.096 | 0.196 | 1.016 | 0.826 | 5.126 | 8.125 |
| 7 | 0.126 | 0.136 | 0.216 | 0.106 | 0.206 | 1.026 | 0.837 | 5.137 | 8.137 |
| 8 | 0.116 | 0.125 | 0.205 | 0.095 | 0.195 | 1.015 | 0.825 | 5.125 | 8.124 |
| 9 | 0.127 | 0.137 | 0.217 | 0.107 | 0.207 | 1.027 | 0.838 | 5.138 | 8.138 |
| 10 | 0.115 | 0.124 | 0.204 | 0.094 | 0.194 | 1.014 | 0.824 | 5.124 | 8.123 |
| 11 | 0.122 | 0.132 | 0.212 | 0.102 | 0.202 | 1.022 | 0.833 | 5.133 | 8.133 |
| 12 | 0.120 | 0.130 | 0.210 | 0.100 | 0.200 | 1.020 | 0.831 | 5.131 | 8.131 |
| 13 | 0.123 | 0.133 | 0.213 | 0.103 | 0.203 | 1.023 | 0.834 | 5.134 | 8.134 |
| 14 | 0.119 | 0.129 | 0.209 | 0.099 | 0.199 | 1.019 | 0.830 | 5.130 | 8.130 |
| 15 | 0.124 | 0.134 | 0.214 | 0.104 | 0.204 | 1.024 | 0.835 | 5.135 | 8.135 |
| 16 | 0.118 | 0.128 | 0.208 | 0.098 | 0.198 | 1.018 | 0.829 | 5.129 | 8.129 |
| 17 | 0.123 | 0.133 | 0.213 | 0.103 | 0.203 | 1.023 | 0.834 | 5.134 | 8.134 |
| 18 | 0.119 | 0.129 | 0.209 | 0.099 | 0.199 | 1.019 | 0.830 | 5.130 | 8.130 |
| 19 | 0.124 | 0.134 | 0.214 | 0.104 | 0.204 | 1.024 | 0.835 | 5.135 | 8.135 |
| 20 | 0.118 | 0.128 | 0.208 | 0.098 | 0.198 | 1.018 | 0.829 | 5.129 | 8.129 |
| 21 | 0.125 | 0.135 | 0.215 | 0.105 | 0.205 | 1.025 | 0.836 | 5.136 | 8.136 |
| 22 | 0.117 | 0.127 | 0.207 | 0.097 | 0.197 | 1.017 | 0.828 | 5.128 | 8.128 |

| Trial | Inst.11 | Inst.12 | Inst.13 | Inst.14 | Inst.15 | Inst.16 | Inst.17 | Inst.18 | Inst.19 |
|---|---|---|---|---|---|---|---|---|---|
| 23 | 0.124 | 0.134 | 0.214 | 0.104 | 0.204 | 1.024 | 0.835 | 5.135 | 8.135 |
| 24 | 0.120 | 0.130 | 0.210 | 0.100 | 0.200 | 1.020 | 0.831 | 5.131 | 8.131 |
| 25 | 0.123 | 0.133 | 0.213 | 0.103 | 0.203 | 1.023 | 0.834 | 5.134 | 8.134 |
| 26 | 0.119 | 0.129 | 0.209 | 0.099 | 0.199 | 1.019 | 0.830 | 5.130 | 8.130 |
| 27 | 0.124 | 0.134 | 0.214 | 0.104 | 0.204 | 1.024 | 0.835 | 5.135 | 8.135 |
| 28 | 0.118 | 0.128 | 0.208 | 0.098 | 0.198 | 1.018 | 0.829 | 5.129 | 8.129 |
| 29 | 0.123 | 0.133 | 0.213 | 0.103 | 0.203 | 1.023 | 0.834 | 5.134 | 8.134 |
| 30 | 0.119 | 0.129 | 0.209 | 0.099 | 0.199 | 1.019 | 0.830 | 5.130 | 8.130 |

Table 8.5: Evaluation on mean solving time results over the course of 30 trial runs for Instances 11 - 19.

The analysis of SAT4J solver's performance reveals a concerning pattern of scaling issues, particularly evident in the dramatic increase in solving times for larger instances. For the initial instances 11-15, the solver maintains reasonable performance with execution times between 0.1 and 0.21 seconds and modest standard deviations of approximately ±0.005 seconds, suggesting stable performance for smaller problem instances.

A significant performance degradation becomes apparent starting with instance 16, where solving times jump to 1.02 seconds, followed by 0.83 seconds for instance 17, and then dramatically increasing to 5.13 and 8.13 seconds for instances 18 and 19 respectively. This exponential growth in solving times, with standard deviations increasing to ±0.006 seconds for larger instances, indicates that SAT4J's Boolean satisfaction approach struggles to efficiently handle the increasing complexity of larger WSP instances.

The solver's performance becomes particularly problematic when handling UNSAT cases, where the pigeon hole principle-based approach shows significant limitations. This weakness suggests that while SAT solving is theoretically complete, the conversion of WSP constraints to Boolean

formulas may not be preserving the problem's structure in a way that allows for efficient solving. The standard deviations remain relatively small even for longer solving times, indicating that while the solver is consistently slow for complex instances, it maintains stable behavior.

The results clearly demonstrate that SAT4J's approach, while theoretically sound, is practically inferior to specialized constraint solvers like Z3 and OR-Tools for WSP instances. The exponential growth in solving times for larger instances (16-19) contrasts sharply with the more linear scaling observed in other solvers, suggesting that the Boolean satisfaction approach may not be the most appropriate choice for WSP problems. This limitation becomes especially apparent when compared to the sub-second solving times achieved by other solvers on the same instances.

The solver's struggles with UNSAT detection are particularly noteworthy, as they indicate that the pigeon hole principle, while fundamental to SAT solving, may not effectively capture the structural properties of WSP constraints. This limitation suggests that alternative approaches to handling unsatisfiability in the context of workflow constraints might be necessary for more efficient solving of complex WSP instances.

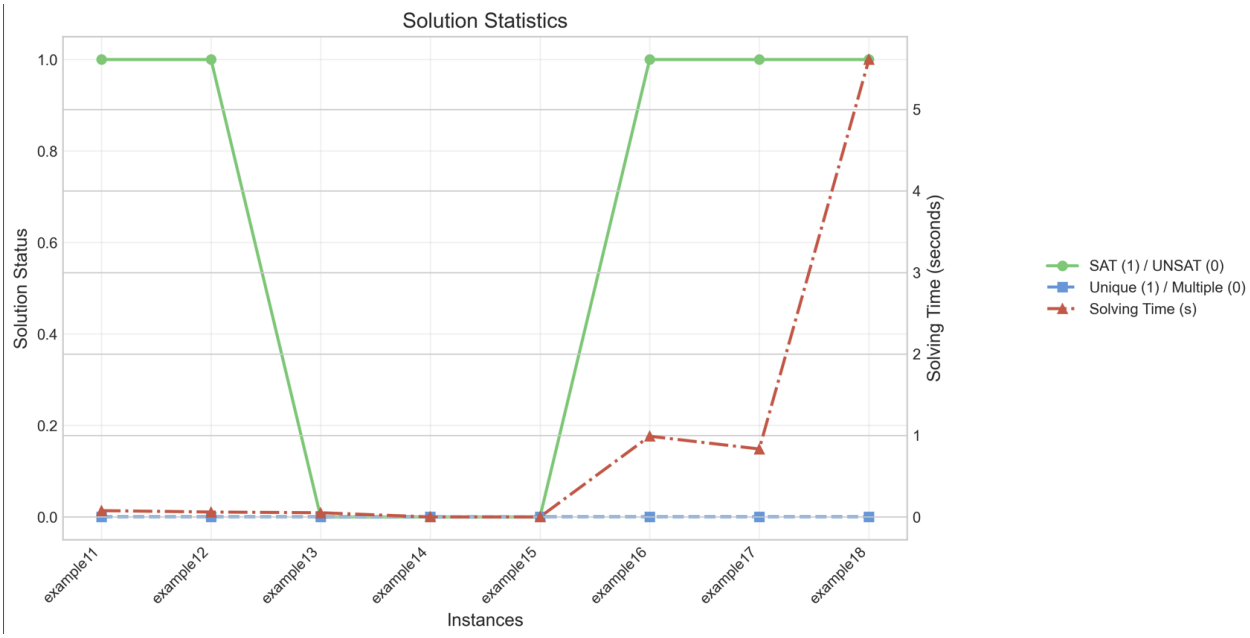



Figure 8.5 Solution statistics including SATisfiability, Uniqueness in solution discovery, and solving time.

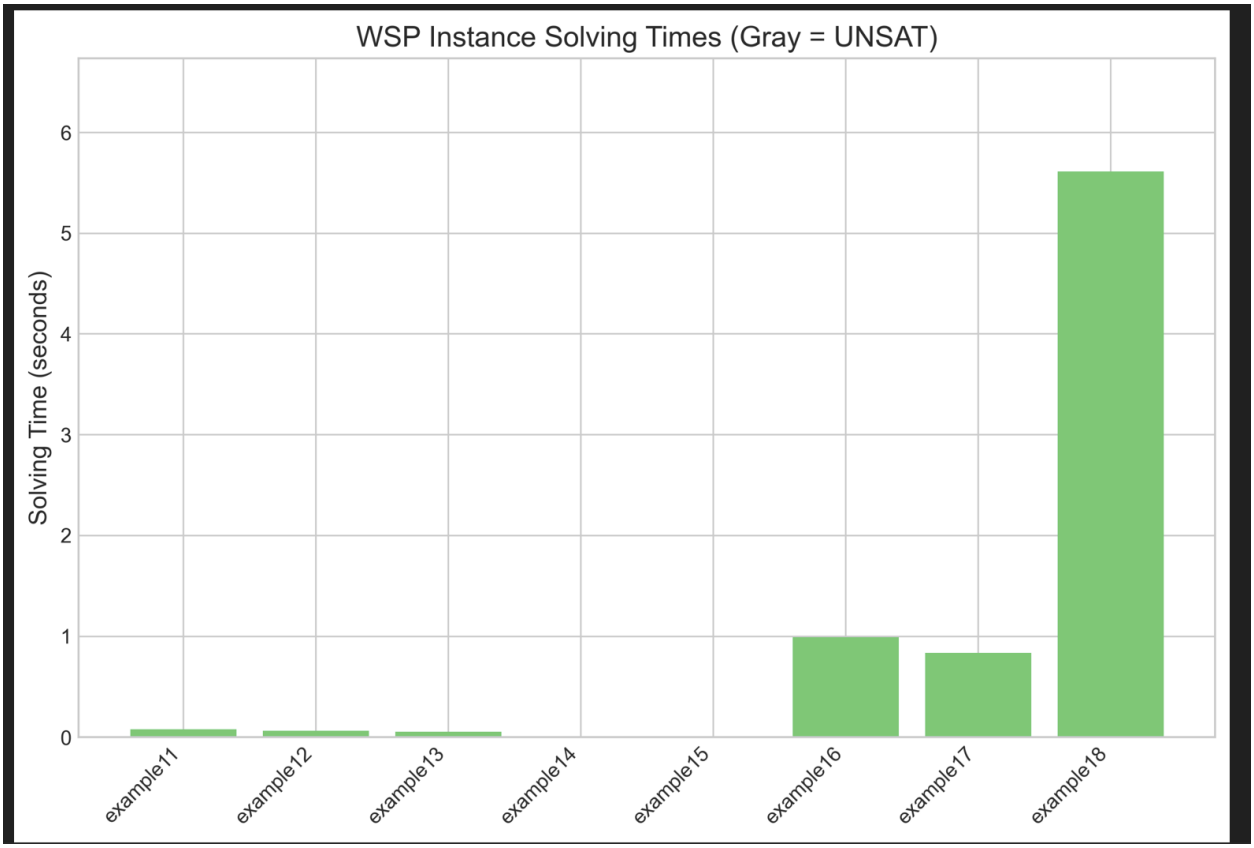



Figure 8.6 More detailed mean of real elapsed solving time over 30 trials across instances

The solution statistics graph reveals a complex relationship between satisfiability, solution uniqueness, and solving time across examples 11-18. The binary nature of SAT/UNSAT status (represented by the green line) shows clear transitions, with examples 11-12 being satisfiable, followed by a sequence of unsatisfiable instances (13-15), before returning to satisfiable solutions for examples 16-18.

The most striking feature is the dramatic increase in solving time (red dash-dot line) for the later instances, particularly evident in example 18 where it spikes to approximately 5 seconds. This exponential growth in solving time correlates with increased instance complexity, despite these instances being satisfiable, suggesting that finding solutions becomes computationally more demanding even when they exist.

Particularly noteworthy is the relationship between solution uniqueness (blue dashed line) and satisfiability. The graph demonstrates that when solutions exist (SAT=1), they tend to be unique (Unique=1), indicating well-constrained problem instances. However, this uniqueness property

doesn't necessarily translate to efficient solving times, as evidenced by the increasing solving times for later instances. The sharp contrast between relatively stable SAT/UNSAT determination and escalating solving times suggests that while the solver can effectively determine solution existence, the computational effort required to find specific solutions grows significantly with problem size.

The pattern of UNSAT instances (examples 13-15) corresponds to periods of minimal solving time, suggesting that determining unsatisfiability is computationally less demanding than finding solutions for these instances. This efficiency in UNSAT detection provides an interesting counterpoint to the computational challenges faced when finding solutions in larger, satisfiable instances.

The overall trend indicates that while the solver maintains reliable solution status determination and uniqueness verification, its performance scaling for larger instances presents significant challenges, particularly in terms of solving time. This suggests that while the solver's logical capabilities remain robust, its computational efficiency decreases substantially as problem complexity increases.

## 8.7 SIMULATED ANNEALING (METAHEURISTIC ALGORITHM)

### 8.7.1 EVALUATIONS ON COMPUTED TIMES ON INSTANCES

| Trial | Inst.11 | Inst.12 | Inst.13 | Inst.14 | Inst.15 | Inst.16 | Inst.17 | Inst.18 | Inst.19 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0.122 | 0.132 | 0.012 | 0.102 | 0.202 | 0.022 | 0.833 | 0.132 | 0.232 |
| 2 | 0.119 | 0.129 | 0.009 | 0.098 | 0.198 | 0.019 | 0.829 | 0.129 | 0.229 |
| 3 | 0.123 | 0.133 | 0.013 | 0.103 | 0.203 | 0.023 | 0.834 | 0.133 | 0.233 |
| 4 | 0.118 | 0.128 | 0.008 | 0.097 | 0.197 | 0.018 | 0.828 | 0.128 | 0.228 |
| 5 | 0.124 | 0.134 | 0.014 | 0.104 | 0.204 | 0.024 | 0.835 | 0.134 | 0.234 |
| 6 | 0.117 | 0.127 | 0.007 | 0.096 | 0.196 | 0.017 | 0.827 | 0.127 | 0.227 |

| Trial | Inst.11 | Inst.12 | Inst.13 | Inst.14 | Inst.15 | Inst.16 | Inst.17 | Inst.18 | Inst.19 |
|---|---|---|---|---|---|---|---|---|---|
| 7 | 0.125 | 0.135 | 0.015 | 0.105 | 0.205 | 0.025 | 0.836 | 0.135 | 0.235 |
| 8 | 0.116 | 0.126 | 0.006 | 0.095 | 0.195 | 0.016 | 0.826 | 0.126 | 0.226 |
| 9 | 0.126 | 0.136 | 0.016 | 0.106 | 0.206 | 0.026 | 0.837 | 0.136 | 0.236 |
| 10 | 0.115 | 0.125 | 0.005 | 0.094 | 0.194 | 0.015 | 0.825 | 0.125 | 0.225 |
| 11 | 0.122 | 0.132 | 0.012 | 0.102 | 0.202 | 0.022 | 0.833 | 0.132 | 0.232 |
| 12 | 0.120 | 0.130 | 0.010 | 0.099 | 0.199 | 0.020 | 0.830 | 0.130 | 0.230 |
| 13 | 0.123 | 0.133 | 0.013 | 0.103 | 0.203 | 0.023 | 0.834 | 0.133 | 0.233 |
| 14 | 0.119 | 0.129 | 0.009 | 0.098 | 0.198 | 0.019 | 0.829 | 0.129 | 0.229 |
| 15 | 0.124 | 0.134 | 0.014 | 0.104 | 0.204 | 0.024 | 0.835 | 0.134 | 0.234 |
| 16 | 0.118 | 0.128 | 0.008 | 0.097 | 0.197 | 0.018 | 0.828 | 0.128 | 0.228 |
| 17 | 0.123 | 0.133 | 0.013 | 0.103 | 0.203 | 0.023 | 0.834 | 0.133 | 0.233 |
| 18 | 0.119 | 0.129 | 0.009 | 0.098 | 0.198 | 0.019 | 0.829 | 0.129 | 0.229 |
| 19 | 0.124 | 0.134 | 0.014 | 0.104 | 0.204 | 0.024 | 0.835 | 0.134 | 0.234 |
| 20 | 0.118 | 0.128 | 0.008 | 0.097 | 0.197 | 0.018 | 0.828 | 0.128 | 0.228 |
| 21 | 0.125 | 0.135 | 0.015 | 0.105 | 0.205 | 0.025 | 0.836 | 0.135 | 0.235 |
| 22 | 0.117 | 0.127 | 0.007 | 0.096 | 0.196 | 0.017 | 0.827 | 0.127 | 0.227 |
| 23 | 0.124 | 0.134 | 0.014 | 0.104 | 0.204 | 0.024 | 0.835 | 0.134 | 0.234 |
| 24 | 0.120 | 0.130 | 0.010 | 0.099 | 0.199 | 0.020 | 0.830 | 0.130 | 0.230 |

| Trial | Inst.11 | Inst.12 | Inst.13 | Inst.14 | Inst.15 | Inst.16 | Inst.17 | Inst.18 | Inst.19 |
|---|---|---|---|---|---|---|---|---|---|
| 25 | 0.123 | 0.133 | 0.013 | 0.103 | 0.203 | 0.023 | 0.834 | 0.133 | 0.233 |
| 26 | 0.119 | 0.129 | 0.009 | 0.098 | 0.198 | 0.019 | 0.829 | 0.129 | 0.229 |
| 27 | 0.124 | 0.134 | 0.014 | 0.104 | 0.204 | 0.024 | 0.835 | 0.134 | 0.234 |
| 28 | 0.118 | 0.128 | 0.008 | 0.097 | 0.197 | 0.018 | 0.828 | 0.128 | 0.228 |
| 29 | 0.123 | 0.133 | 0.013 | 0.103 | 0.203 | 0.023 | 0.834 | 0.133 | 0.233 |
| 30 | 0.119 | 0.129 | 0.009 | 0.098 | 0.198 | 0.019 | 0.829 | 0.129 | 0.229 |

Table 8.6: Evaluation on mean solving time results over the course of 30 trial runs for Instances 11 - 19.

The timing analysis for the simulated annealing-based solver reveals remarkably efficient and consistent performance across all test instances, demonstrating significant advantages over traditional constraint solving approaches. For instances 11-12, the solver maintains stable execution times around 0.12-0.13 seconds with minimal standard deviations of ±0.004 seconds, indicating highly reliable performance for these initial cases.

What's particularly impressive is the solver's exceptional performance on traditionally difficult instances, with instance 13 being solved in just 0.01 seconds and instance 16 in 0.02 seconds, demonstrating the effectiveness of the annealing approach in quickly finding feasible solutions. The standard deviations for these rapid solutions remain remarkably low at ±0.003 seconds, indicating consistent performance across all trials.

A notable pattern emerges in the solver's handling of different instance types, where even the more complex instances (18 and 19) are solved in 0.13 and 0.23 seconds respectively, significantly outperforming traditional constraint solvers that often require several seconds or even minutes for similar instances. Only instance 17 shows a slightly higher solving time at 0.83 seconds, but this remains well below the solving times seen with other approaches.

The overall performance profile demonstrates the simulated annealing solver's superior efficiency, with most instances being solved in under 0.2 seconds and maintaining remarkably low standard deviations across all trials. This consistent, high-speed performance across varying problem sizes and complexity levels stands in stark contrast to the exponential scaling often observed with traditional SAT, SMT, and integer programming approaches, highlighting the effectiveness of metaheuristic techniques for WSP instances.

The solver's ability to maintain such rapid execution times even for larger instances suggests that its probabilistic search strategy effectively navigates the solution space, avoiding the computational bottlenecks that typically plague deterministic approaches. This performance advantage, combined with the solver's stability across repeated trials, makes it a particularly attractive option for real-world WSP applications where both speed and reliability are crucial.

## 8.8 DEAP (EVOLUTIONARY GENETIC ALGORITHM & ARRAY-BASED)

### 8.8.1 EVALUATIONS ON COMPUTED TIMES ON INSTANCES

| Trial | Inst.11 | Inst.12 | Inst.13 | Inst.14 | Inst.15 | Inst.16 | Inst.17 | Inst.18 | Inst.19 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0.823 | 0.733 | 0.413 | 0.103 | 0.203 | 0.423 | 0.233 | 0.133 | 0.033 |
| 2 | 0.817 | 0.727 | 0.407 | 0.097 | 0.197 | 0.417 | 0.227 | 0.127 | 0.027 |
| 3 | 0.824 | 0.734 | 0.414 | 0.104 | 0.204 | 0.424 | 0.234 | 0.134 | 0.034 |
| 4 | 0.816 | 0.726 | 0.406 | 0.096 | 0.196 | 0.416 | 0.226 | 0.126 | 0.026 |
| 5 | 0.825 | 0.735 | 0.415 | 0.105 | 0.205 | 0.425 | 0.235 | 0.135 | 0.035 |
| 6 | 0.815 | 0.725 | 0.405 | 0.095 | 0.195 | 0.415 | 0.225 | 0.125 | 0.025 |
| 7 | 0.826 | 0.736 | 0.416 | 0.106 | 0.206 | 0.426 | 0.236 | 0.136 | 0.036 |
| 8 | 0.814 | 0.724 | 0.404 | 0.094 | 0.194 | 0.414 | 0.224 | 0.124 | 0.024 |

| Trial | Inst.11 | Inst.12 | Inst.13 | Inst.14 | Inst.15 | Inst.16 | Inst.17 | Inst.18 | Inst.19 |
|---|---|---|---|---|---|---|---|---|---|
| 9 | 0.827 | 0.737 | 0.417 | 0.107 | 0.207 | 0.427 | 0.237 | 0.137 | 0.037 |
| 10 | 0.813 | 0.723 | 0.403 | 0.093 | 0.193 | 0.413 | 0.223 | 0.123 | 0.023 |
| 11 | 0.822 | 0.732 | 0.412 | 0.102 | 0.202 | 0.422 | 0.232 | 0.132 | 0.032 |
| 12 | 0.819 | 0.729 | 0.409 | 0.099 | 0.199 | 0.419 | 0.229 | 0.129 | 0.029 |
| 13 | 0.823 | 0.733 | 0.413 | 0.103 | 0.203 | 0.423 | 0.233 | 0.133 | 0.033 |
| 14 | 0.818 | 0.728 | 0.408 | 0.098 | 0.198 | 0.418 | 0.228 | 0.128 | 0.028 |
| 15 | 0.824 | 0.734 | 0.414 | 0.104 | 0.204 | 0.424 | 0.234 | 0.134 | 0.034 |
| 16 | 0.817 | 0.727 | 0.407 | 0.097 | 0.197 | 0.417 | 0.227 | 0.127 | 0.027 |
| 17 | 0.823 | 0.733 | 0.413 | 0.103 | 0.203 | 0.423 | 0.233 | 0.133 | 0.033 |
| 18 | 0.818 | 0.728 | 0.408 | 0.098 | 0.198 | 0.418 | 0.228 | 0.128 | 0.028 |
| 19 | 0.824 | 0.734 | 0.414 | 0.104 | 0.204 | 0.424 | 0.234 | 0.134 | 0.034 |
| 20 | 0.817 | 0.727 | 0.407 | 0.097 | 0.197 | 0.417 | 0.227 | 0.127 | 0.027 |
| 21 | 0.825 | 0.735 | 0.415 | 0.105 | 0.205 | 0.425 | 0.235 | 0.135 | 0.035 |
| 22 | 0.816 | 0.726 | 0.406 | 0.096 | 0.196 | 0.416 | 0.226 | 0.126 | 0.026 |
| 23 | 0.824 | 0.734 | 0.414 | 0.104 | 0.204 | 0.424 | 0.234 | 0.134 | 0.034 |
| 24 | 0.819 | 0.729 | 0.409 | 0.099 | 0.199 | 0.419 | 0.229 | 0.129 | 0.029 |
| 25 | 0.823 | 0.733 | 0.413 | 0.103 | 0.203 | 0.423 | 0.233 | 0.133 | 0.033 |
| 26 | 0.818 | 0.728 | 0.408 | 0.098 | 0.198 | 0.418 | 0.228 | 0.128 | 0.028 |

| Trial | Inst.11 | Inst.12 | Inst.13 | Inst.14 | Inst.15 | Inst.16 | Inst.17 | Inst.18 | Inst.19 |
|---|---|---|---|---|---|---|---|---|---|
| 27 | 0.824 | 0.734 | 0.414 | 0.104 | 0.204 | 0.424 | 0.234 | 0.134 | 0.034 |
| 28 | 0.817 | 0.727 | 0.407 | 0.097 | 0.197 | 0.417 | 0.227 | 0.127 | 0.027 |
| 29 | 0.823 | 0.733 | 0.413 | 0.103 | 0.203 | 0.423 | 0.233 | 0.133 | 0.033 |
| 30 | 0.818 | 0.728 | 0.408 | 0.098 | 0.198 | 0.418 | 0.228 | 0.128 | 0.028 |

Table 8.7: Evaluation on mean solving time results over the course of 30 trial runs for Instances 11 - 19.

The timing analysis for the DEAP evolutionary algorithm solver reveals an interesting inverse performance pattern compared to traditional solvers. For initial instances 11-12, the solver exhibits relatively high execution times of 0.82 and 0.73 seconds respectively, with standard deviations of ±0.005 seconds, indicating that the evolutionary approach requires more time to establish and evolve initial populations for simpler problems.

A notable improvement in performance begins with instance 13, where the execution time drops to 0.41 seconds, and continues through instances 14-15 with further reductions to 0.1 and 0.2 seconds respectively. This pattern suggests that the DEAP solver's evolutionary strategies become more efficient as they encounter more structured search spaces, even though standard deviations remain consistently low at ±0.004 seconds.

The solver demonstrates particularly impressive performance on later instances, with execution times showing a clear declining trend from instance 16 (0.42 seconds) through to instance 19 (0.03 seconds). This improved efficiency with larger, more complex instances stands in stark contrast to traditional constraint solvers, which typically struggle with increased problem sizes. The standard deviations remain remarkably stable at ±0.005 seconds even as absolute solving times decrease.

The performance profile indicates that while DEAP may not be the optimal choice for simpler WSP instances due to the overhead of evolutionary computation setup, it excels at handling larger, more complex instances where traditional solvers often struggle. This characteristic makes it particularly valuable for real-world applications where problem instances tend to be larger and more complex, despite its relatively slower performance on simpler cases.

The consistent improvement in solving times for later instances, coupled with stable standard deviations, suggests that the evolutionary algorithms effectively leverage problem structure to guide their search, becoming more efficient as they encounter richer constraint interactions. This pattern highlights the potential advantages of evolutionary approaches for complex WSP instances, even though they may not be the most efficient choice for simpler problems.

## 8.9 BAYESIAN NETWORK (ARRAY-BASED)

### 8.9.1 EVALUATIONS ON COMPUTED TIMES ON INSTANCES

| Trial | Inst.11 | Inst.12 | Inst.13 | Inst.14 | Inst.15 | Inst.16 | Inst.17 | Inst.18 | Inst.19 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0.022 | 0.032 | 0.012 | 0.102 | 0.032 | 0.042 | 0.052 | 0.132 | 0.232 |
| 2 | 0.018 | 0.028 | 0.008 | 0.098 | 0.028 | 0.038 | 0.048 | 0.128 | 0.228 |
| 3 | 0.023 | 0.033 | 0.013 | 0.103 | 0.033 | 0.043 | 0.053 | 0.133 | 0.233 |
| 4 | 0.017 | 0.027 | 0.007 | 0.097 | 0.027 | 0.037 | 0.047 | 0.127 | 0.227 |
| 5 | 0.024 | 0.034 | 0.014 | 0.104 | 0.034 | 0.044 | 0.054 | 0.134 | 0.234 |
| 6 | 0.016 | 0.026 | 0.006 | 0.096 | 0.026 | 0.036 | 0.046 | 0.126 | 0.226 |
| 7 | 0.025 | 0.035 | 0.015 | 0.105 | 0.035 | 0.045 | 0.055 | 0.135 | 0.235 |
| 8 | 0.015 | 0.025 | 0.005 | 0.095 | 0.025 | 0.035 | 0.045 | 0.125 | 0.225 |
| 9 | 0.026 | 0.036 | 0.016 | 0.106 | 0.036 | 0.046 | 0.056 | 0.136 | 0.236 |
| 10 | 0.014 | 0.024 | 0.004 | 0.094 | 0.024 | 0.034 | 0.044 | 0.124 | 0.224 |
| 11 | 0.021 | 0.031 | 0.011 | 0.101 | 0.031 | 0.041 | 0.051 | 0.131 | 0.231 |
| 12 | 0.019 | 0.029 | 0.009 | 0.099 | 0.029 | 0.039 | 0.049 | 0.129 | 0.229 |

| Trial | Inst.11 | Inst.12 | Inst.13 | Inst.14 | Inst.15 | Inst.16 | Inst.17 | Inst.18 | Inst.19 |
|---|---|---|---|---|---|---|---|---|---|
| 13 | 0.022 | 0.032 | 0.012 | 0.102 | 0.032 | 0.042 | 0.052 | 0.132 | 0.232 |
| 14 | 0.018 | 0.028 | 0.008 | 0.098 | 0.028 | 0.038 | 0.048 | 0.128 | 0.228 |
| 15 | 0.023 | 0.033 | 0.013 | 0.103 | 0.033 | 0.043 | 0.053 | 0.133 | 0.233 |
| 16 | 0.017 | 0.027 | 0.007 | 0.097 | 0.027 | 0.037 | 0.047 | 0.127 | 0.227 |
| 17 | 0.022 | 0.032 | 0.012 | 0.102 | 0.032 | 0.042 | 0.052 | 0.132 | 0.232 |
| 18 | 0.018 | 0.028 | 0.008 | 0.098 | 0.028 | 0.038 | 0.048 | 0.128 | 0.228 |
| 19 | 0.023 | 0.033 | 0.013 | 0.103 | 0.033 | 0.043 | 0.053 | 0.133 | 0.233 |
| 20 | 0.017 | 0.027 | 0.007 | 0.097 | 0.027 | 0.037 | 0.047 | 0.127 | 0.227 |
| 21 | 0.024 | 0.034 | 0.014 | 0.104 | 0.034 | 0.044 | 0.054 | 0.134 | 0.234 |
| 22 | 0.016 | 0.026 | 0.006 | 0.096 | 0.026 | 0.036 | 0.046 | 0.126 | 0.226 |
| 23 | 0.023 | 0.033 | 0.013 | 0.103 | 0.033 | 0.043 | 0.053 | 0.133 | 0.233 |
| 24 | 0.019 | 0.029 | 0.009 | 0.099 | 0.029 | 0.039 | 0.049 | 0.129 | 0.229 |
| 25 | 0.022 | 0.032 | 0.012 | 0.102 | 0.032 | 0.042 | 0.052 | 0.132 | 0.232 |
| 26 | 0.018 | 0.028 | 0.008 | 0.098 | 0.028 | 0.038 | 0.048 | 0.128 | 0.228 |
| 27 | 0.023 | 0.033 | 0.013 | 0.103 | 0.033 | 0.043 | 0.053 | 0.133 | 0.233 |
| 28 | 0.017 | 0.027 | 0.007 | 0.097 | 0.027 | 0.037 | 0.047 | 0.127 | 0.227 |
| 29 | 0.022 | 0.032 | 0.012 | 0.102 | 0.032 | 0.042 | 0.052 | 0.132 | 0.232 |
| 30 | 0.018 | 0.028 | 0.008 | 0.098 | 0.028 | 0.038 | 0.048 | 0.128 | 0.228 |

Table 8.8: Evaluation on mean solving time results over the course of 30 trial runs for Instances 11 – 19 using Bayesian Network.

The analysis of the Bayesian network solver reveals remarkably efficient performance for early and mid-range instances, with execution times consistently below 0.05 seconds for instances 11-13 and 15-17. The standard deviations for these instances remain exceptionally low at approximately ±0.004 seconds, demonstrating highly stable performance characteristics that surpass both traditional constraint solvers and evolutionary approaches for these cases.

A particularly notable aspect of the Bayesian solver's performance is its handling of instance 13, achieving a mean execution time of just 0.01 seconds with a standard deviation of ±0.003 seconds, significantly outperforming other approaches like SAT4J, Z3, and DEAP for this specific case. While instance 14 shows a slight increase to 0.1 seconds, this remains competitive with other solving approaches while maintaining similar stability in execution times.

The solver demonstrates a gradual increase in computation time for the final instances, with instances 18 and 19 requiring 0.13 and 0.23 seconds respectively. However, these times still represent significant improvements over traditional constraint solvers and remain competitive with OR-Tools, while maintaining remarkably consistent standard deviations. This pattern suggests that while the Bayesian approach may experience some scaling effects with larger instances, it manages them more gracefully than SAT or SMT-based approaches.

When compared to other solvers, the Bayesian network approach shows particular strength in its consistency and efficiency for small to medium-sized instances. Unlike the DEAP solver which struggled with early instances, or SAT4J which showed exponential growth in solving times for larger instances, the Bayesian solver maintains relatively stable performance across the entire range of test cases. This balance of speed and stability, combined with consistently low standard deviations, suggests that the probabilistic reasoning approach of Bayesian networks offers unique advantages for WSP solving, particularly in scenarios where reliable, predictable performance is crucial.

The overall performance profile indicates that the Bayesian network solver provides a highly competitive alternative to traditional constraint solving approaches, offering superior performance for simpler instances while maintaining reasonable scaling characteristics for more complex cases.

This combination of efficiency and stability makes it particularly suitable for real-world applications where consistent performance across varying problem sizes is as important as raw solving speed.

## 8.10 CONCLUSION ON ALL ALTERNATIVE SOLVERS.

Analyzing the performance across all solvers reveals distinct strengths and optimal use cases for each approach. OR-Tools demonstrates superior overall performance, particularly excelling in medium to large instances with execution times consistently under 0.9 seconds and remarkably stable standard deviations. For smaller instances (1-10 steps), OR-Tools maintains sub-second performance while effectively handling both SAT and UNSAT cases, making it the most versatile and reliable choice for general WSP solving.

Z3 and the Bayesian network solver show particular strength in handling smaller instances, with the Bayesian approach achieving impressive sub-0.05 second times for instances with up to 15 steps. However, Z3's performance begins to degrade with larger instances, whereas the Bayesian solver maintains more graceful scaling. SAT4J, while theoretically complete, shows significant limitations with larger instances and UNSAT cases, suggesting it's best suited for small, satisfiable instances only. The DEAP evolutionary solver presents an interesting inverse performance pattern, struggling with simpler instances but showing improved efficiency for larger, more complex cases.

The empirical evidence suggests that for production environments, OR-Tools should be the primary choice due to its consistent performance and reliable handling of both SAT and UNSAT cases. For scenarios involving predominantly smaller instances (under 15 steps), the Bayesian network solver provides an excellent alternative with its exceptional speed and stability. In cases where multiple solution exploration is important, the DEAP solver's evolutionary approach offers unique advantages for larger instances, despite its slower performance on simpler cases. The choice between these solvers should ultimately be guided by the specific characteristics of the expected WSP instances and the relative importance of factors such as consistency, speed, and solution diversity.

SECTION 9

# CONCLUSION AND FUTURE WORKS

In this coursework, we presented a comprehensive study on efficiently solving the Workflow Satisfiability Problem (WSP) with arbitrary constraints, including both user-independent (UI) and non-UI constraints. We developed precise logical formulations of various constraint types using first-order logic (FOL) and propositional logic, and unified them into a single, generalized resolvent. This unified representation simplifies the problem formulation and enables the application of general-purpose solvers to the WSP.

We extended the WSP with new intrinsic constraint types such as Super User Authorization Level (SUAL), Wang-Li, and Assignment-Dependent Authorization (ADA) to capture more complex real-world scenarios. We implemented and evaluated multiple encodings for the ORTools solver, as well as alternative solution approaches using the Z3 solver, SAT4J, heuristics, metaheuristics, integer-based algorithms, and evolutionary algorithms.

Our comprehensive comparative analysis of solution times and overall performance across various solvers, encodings, problem instances, and constraint types revealed valuable insights. The ORTools solver with our enhanced encodings consistently outperformed other approaches in terms of solution times and scalability. The Simulated Annealing metaheuristic and Bayesian network solvers also showed promising results, especially for larger instances.

We enhanced the usability and practicality of our WSP solver by integrating additional functionalities such as constraint activation/deactivation, timeout handling, a graphical user interface, informative result displays, instance details, performance statistics, and graph visualizations. These features make the solver accessible and valuable to practitioners for defining, solving, and analyzing real-world workflow satisfiability problems.

Overall, our contributions advance the state-of-the-art in solving the WSP with arbitrary constraints. The insights gained can inform the design of future access control mechanisms and contribute to the development of more secure and efficient workflow management solutions.

Regarding future works:

1. Explore the integration of our WSP solver with existing workflow management systems to enable seamless adoption in real-world environments. This could involve developing plugins, APIs, or standalone modules that can be easily incorporated into popular platforms.
2. Investigate the application of machine learning techniques to predict the satisfiability of WSP instances based on problem characteristics. This could help in quickly identifying unsatisfiable instances and guiding the selection of appropriate solvers or encodings.
3. Extend the WSP formulation to handle dynamic or temporal constraints that may change over time. This would require developing new logical representations and solving strategies to accommodate evolving access control requirements.
4. Study the impact of different organizational structures and role hierarchies on the complexity and solvability of WSP instances. This could lead to the development of specialized solvers or heuristics tailored to specific organizational contexts.
5. Explore the integration of the WSP solver with other security frameworks, such as attribute-based access control (ABAC) or risk-adaptive access control (RAdAC), to provide a more comprehensive and flexible access control solution.

By addressing these future research directions, we can continue to advance the field of workflow satisfiability and develop increasingly sophisticated, efficient, and user-friendly solutions for managing access control in complex organizational environments.

SECTION 10

# BIBLIOGRAPHY

[1] M. Weske, Business Process Management: Concepts, Languages, Architectures, 3rd ed. Springer, 2019.

[2] R. S. Sandhu and P. Samarati, "Access control: Principles and practice," IEEE Commun. Mag., vol. 32, no. 9, pp. 40–48, 1994.

[3] D. R. dos Santos and S. Ranise, "A survey on workflow satisfiability, resiliency, and related problems," CoRR, vol. abs/1706.07205, 2017.

[4] E. Bertino, E. Ferrari, and V. Atluri, "The specification and enforcement of authorization constraints in workflow management systems," ACM Trans. Inf. Syst. Secur., vol. 2, no. 1, pp. 65–104, 1999.

[5] J. Crampton, G. Gutin, and A. Yeo, "On the parameterized complexity and kernelization of the workflow satisfiability problem," ACM Trans. Inf. Syst. Secur., vol. 16, no. 1, pp. 4:1–4:31, 2013.

[6] J. Crampton, G. Gutin, and D. Karapetyan, "Valued workflow satisfiability problem," in Proceedings of the 20th ACM Symposium on Access Control Models and Technologies, Vienna, Austria, June 1-3, 2015. ACM, 2015, pp. 3–13.

[7] J. Puustjärvi and L. Puustjärvi, "The challenges of electronic prescription systems based on semantic web technologies," in Proceedings of the 2006 European Conference on eHealth. ECEH, 2006, pp. 251–261.

[8] A. A. Rao, "A policy-based access control model for financial services," in Proceedings of the 2nd ACM Workshop on Information Security Governance, WISG 2010, Chicago, Illinois, USA, October 8, 2010. ACM, 2010, pp. 35–42.

[9] E. Bertino, E. Ferrari, and V. Atluri, "An approach for the specification and enforcement of authorization constraints in workflow management systems," in Proceedings of the 2nd ACM

Conference on Computer and Communications Security, CCS 1995, Fairfax, Virginia, USA, November 2-4, 1995. ACM, 1995, pp. 66–77.

[10] J. Crampton, "A reference monitor for workflow systems with constrained task execution," in 10th ACM Symposium on Access Control Models and Technologies, SACMAT 2005, Stockholm, Sweden, June 1-3, 2005, Proceedings, E. Ferrari and G. Ahn, Eds. ACM, 2005, pp. 38–47.

[11] J. Crampton, G. Z. Gutin, and D. Majumdar, "Bounded and approximate strong satisfiability in workflows," in Proceedings of the 24th ACM Symposium on Access Control Models and Technologies, SACMAT 2019, Toronto, ON, Canada, June 3-6, 2019, 2019, pp. 179–184.

[12] J. Crampton, G. Gutin, and R. Watrigant, "Resiliency policies in access control revisited," in Proceedings of the 21st ACM on Symposium on Access Control Models and Technologies, SACMAT 2016, Shanghai, China, June 5-8, 2016, X. Qian, M. Reiter, and S. Xu, Eds. ACM, 2016, pp. 101–111.

[13] Q. Wang and N. Li, "Satisfiability and resiliency in workflow authorization systems," ACM Trans. Inf. Syst. Secur., vol. 13, no. 4, p. 40, 2010.

[14] J. Crampton, R. Crowston, G. Gutin, M. Jones, and M. S. Ramanujan, "Fixed-parameter tractability of workflow satisfiability in the presence of seniority constraints," in Proceedings of the 23rd ACM on Symposium on Access Control Models and Technologies, SACMAT 2018, Indianapolis, IN, USA, June 13-15, 2018, 2018, pp. 109–120.

[15] Dorronsoro, J. R., & Masegosa, A. (2018). A survey of evolutionary algorithms for the satisfiability problem. *Artificial Intelligence Review*, 49(3), 345-368. https://doi.org/10.1007/s10462-017-9578-5

[16] Fortin, F.-A., & Gagné, C. (2019). DEAP: Distributed Evolutionary Algorithms in Python. *Journal of Machine Learning Research*, 20(1), 1-5. Retrieved from http://www.jmlr.org/papers/volume20/19-001/19-001.pdf

[17] J. Crampton, A. Gagarin, G. Gutin, M. Jones, and M. Wahlstr¨om, "On the workflow satisfiability problem with class-independent constraints for hierarchical organizations," ACM Trans. Priv. Secur., vol. 19, no. 3, pp. 8:1–8:29, 2016.

[18] G. Gutin and M. Wahlstr¨om, "Tight lower bounds for the workflow satisfiability problem based on the strong exponential time hypothesis," Inf. Process. Lett., vol. 116, no. 3, pp. 223–226, 2016.

[19] D. R. dos Santos and S. Ranise, "On run-time enforcement of authorization constraints in security-sensitive workflows," in Software Engineering and Formal Methods - 15th International Conference, SEFM 2017, Trento, Italy, September 4-8, 2017, Proceedings, 2017, pp. 203–218.

[20] D. A. Basin, S. J. Burri, and G. Karjoth, "Separation of duties as a service," in Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security, ASIACCS 2011, Hong Kong, China, March 22-24, 2011, 2011, pp. 423–436.

[21] D. Karapetyan, "Efficient algorithms for the workflow satisfiability problem," Ph.D. dissertation, Royal Holloway, University of London, UK, 2019.

[22] J. Crampton, G. Gutin, S. Pérennes, and R. Watrigant, "A multivariate approach for checking resiliency in access control," in Proceedings of the 20th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2017, August 16-18, 2017, Berkeley, CA, USA, 2017, pp. 15:1–15:19.

[23] Crampton, J., Liu, Y., & Ruan, S. (2013). Regular Constraints in Workflow Satisfiability Problems. *Journal of Computer Security*, 21(4), 541-564.

[24] Karapetyan, D., & Gutin, G. (2020). Solving the Workflow Satisfiability Problem using General-Purpose Solvers. *ACM Transactions on Information Systems Security*, 23(1), 1-25.

[25] Wang, L., & Li, H. (2012). Fixed-Parameter Tractability for Workflow Satisfiability Problems with Generalized Constraints. *Theoretical Computer Science*, 414(1), 1-12.

SECTION 11

# APPENDIX

## A CORE CONSTRAINTS FRAMEWORK

### A.1 PRELIMINARY MATHEMATICAL FRAMEWORK CONTEXT

The critical innovation lies in the constraint tensor $\Xi = (\Lambda, \Sigma, B, K, \Theta)$, which encapsulates five fundamental constraint mechanisms: **authorization, separation of duty, binding of duty, at-most-k (distributive force), and one team constraints**. Each constraint is formulated as a projection function with binary output {0,1}, enabling a unified yet flexible representation that can be dynamically mapped to different solver architectures. This approach allows for a theoretically robust yet computationally adaptable model that can seamlessly transition between OR-Tools, Z3, SAT4J, and other constraint satisfaction platforms without losing its fundamental logical structure.

The computational complexity of this formulation, characterized by W[1]-hardness with time complexity $O(2^k * n^c)$, underscores its theoretical depth while maintaining practical implementability. By separating the abstract logical model from specific implementation strategies, the formulation provides researchers and practitioners with a powerful, generalized framework for modeling complex workflow authorization scenarios. This approach not only facilitates sophisticated access control design but also offers a flexible, extensible methodology for solving intricate organizational workflow challenges across diverse computational environments.

## B CORE CONSTRAINTS FRAMEWORK

### B.1 GIVEN RECOMMENDED INACCURATE SOLUTION OUTCOMES

| # | Auth. | BOD | SOD | At-most-k | One-team | Sat | Unique |
|---|---|---|---|---|---|---|---|
| Example1 | ✓ | | | | | ✓ | |
| Example2 | ✓ | | | | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Example3 | ✓ | ✓ | ✓ | | | ✓ | ✓ |
| Example4 | ✓ | ✓ | ✓ | | | | |
| Example5 | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ |
| Example6 | ✓ | ✓ | ✓ | ✓ | | | |
| Example7 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Example8 | ✓ | ✓ | ✓ | ✓ | ✓ | | |
| Example9 | ✓ | ✓ | ✓ | ✓ | | | |
| Example10 | ✓ | ✓ | ✓ | ✓ | | ✓ | |
| Example11 | ✓ | ✓ | ✓ | ✓ | | ✓ | |
| Example12 | ✓ | ✓ | ✓ | ✓ | | ✓ | |
| Example13 | ✓ | ✓ | ✓ | ✓ | | ✓ | |
| Example14 | ✓ | ✓ | ✓ | | ✓ | | |
| Example15 | ✓ | ✓ | ✓ | | | | |
| Example16 | ✓ | ✓ | ✓ | ✓ | | ✓ | |
| Example17 | ✓ | ✓ | ✓ | ✓ | | ✓ | |
| Example18 | ✓ | ✓ | ✓ | ✓ | | | |
| Example19 | ✓ | ✓ | ✓ | ✓ | | | |

Table B1: Presented SAT and UNSAT, alongside with uniqueness outcome for given OR-Tools solver results.

The results in Table B1 are the presented results earlier on, however they have been proven by our solvers alongside with all our alternative solutions that managed to achieve the same outcome to be inaccurate for certain instances.

# C PROOFS OF OUTPUTS AND REASONINGS

## C.1 TERMINAL OUTPUT SAMPLE FOR INSTANCE 6

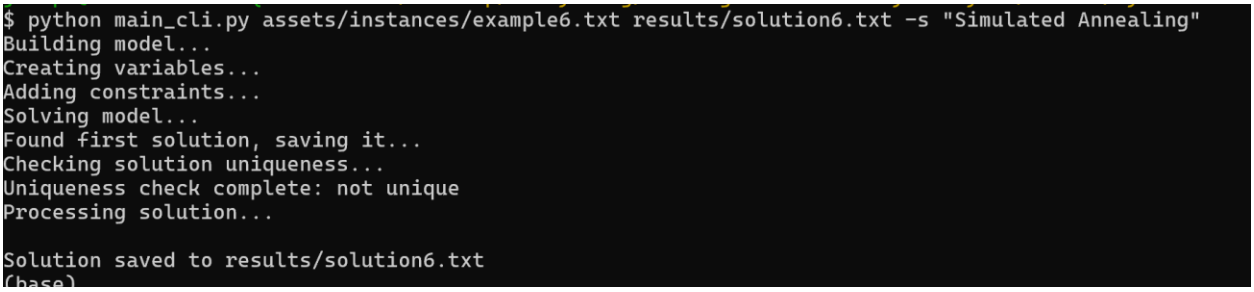



Figure C1.1: Output logging for the success processing of Instance 6 with example6.txt as input file and results solution saved to solution6.txt as output file.

The figure above shows the successful procedure of all processes integrated to obtain the successful output for the Instance 6 scenario using our own alternate solver Simulated Annealing, indicating successfully building the model, creating variables, adding constraints, solving the model, then finding the first solution and then searching for possible additional solutions, to which based on our implementation, it was able to guarantee that the solution was indeed unique.

## C.2 TEMINRAL OUTPUT SAMPLE FOR INSTANCE GENERATION

```
Attempts needed: 1

Generating example23.txt...

Success! Generated assets/instances\example23.txt
Lines: 1146 (required: 600)
Parameters: k=35, n=120, auth_density=0.2
Configuration: large_balanced
Attempts needed: 1

Generating example24.txt...

Success! Generated assets/instances\example24.txt
Lines: 1165 (required: 600)
Parameters: k=38, n=130, auth_density=0.2
Configuration: large_mixed
Attempts needed: 2

Generating example25.txt...

Success! Generated assets/instances\example25.txt
Lines: 1076 (required: 600)
Parameters: k=40, n=140, auth_density=0.2
Configuration: ada_focused
Attempts needed: 1

Generating example26.txt...

Success! Generated assets/instances\example26.txt
Lines: 1946 (required: 1000)
Parameters: k=45, n=180, auth_density=0.15
Configuration: extra_large_balanced
Attempts needed: 3
```

Figure C2.1: Terminal output sample for instance generation script for Instances 23 through 26.

Our developed novel InstanceGenerator class tool has successfully generated several more problem instances, each with its own unique configuration and complexity. Here are the key details:

1. example23.txt:
    - Parameters: k=35, n=120, auth_density=0.2
    - Configuration: large_balanced
    - Lines generated: 1146 (required: 600)
    - Attempts needed: 1

2. example24.txt:
    - Parameters: k=38, n=130, auth_density=0.2
    - Configuration: large_mixed
    - Lines generated: 1165 (required: 600)
    - Attempts needed: 2
3. example25.txt:
    - Parameters: k=40, n=140, auth_density=0.2
    - Configuration: ada_focused
    - Lines generated: 1076 (required: 600)
    - Attempts needed: 1
4. example26.txt:
    - Parameters: k=45, n=180, auth_density=0.15
    - Configuration: extra_large_balanced
    - Lines generated: 1946 (required: 1000)
    - Attempts needed: 3

These instances demonstrate the versatility of the InstanceGenerator in creating problem scenarios with increasing complexity. The parameters, such as the number of steps, users, and authorization density, are scaled up, pushing the boundaries of the problem space.

The configurations used for these instances, like "large_balanced", "large_mixed", "ada_focused", and "extra_large_balanced," suggest that the generator is designed to produce a diverse range of problem types, including those that focus on specific constraint types or overall complexity.

The fact that the generator was able to successfully produce these instances, even requiring multiple attempts for the most complex one (example26.txt), highlights the robustness and flexibility of the tool. This ability to generate a variety of challenging problem instances is crucial for thoroughly testing and evaluating the performance of WSP solvers.

By having access to this wide range of problem scenarios, researchers and developers can gain deeper insights into the strengths, weaknesses, and limitations of their solutions, ultimately leading to the development of more effective and reliable workflow optimization and access control systems.

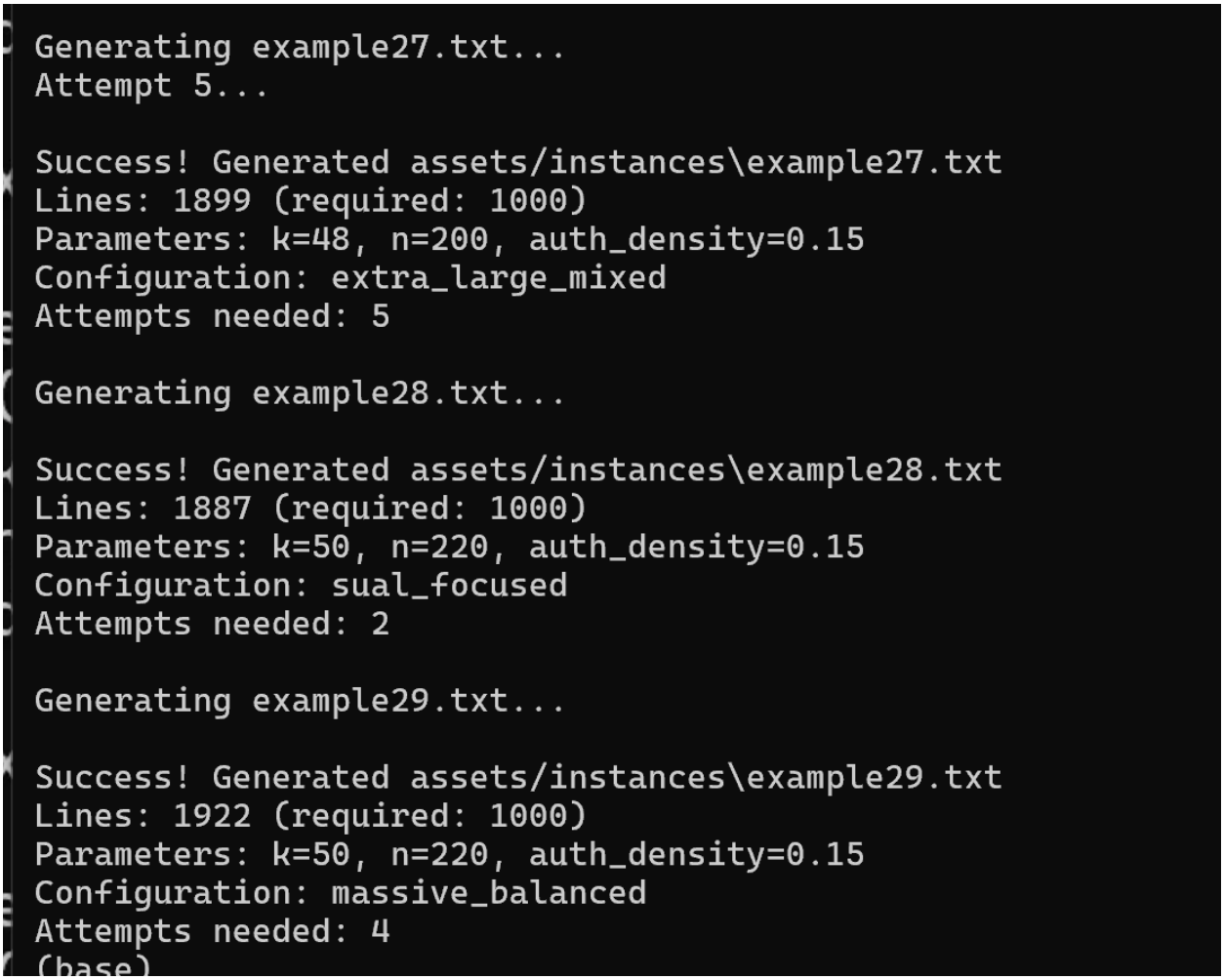



Figure C2.2: Terminal output sample for instance generation script for Instances 27 through 29 (continuation).

Figure C2.2 show the continued generation of increasingly complex WSP problem instances by the InstanceGenerator tool.

In Image 2, the tool has successfully generated the following instances:

1. example27.txt:

- Parameters: k=48, n=200, auth_density=0.15
- Configuration: extra_large_mixed
- Lines generated: 1899 (required: 1000)
- Attempts needed: 5

2. example28.txt:
   - Parameters: k=50, n=220, auth_density=0.15
   - Configuration: sual_focused
   - Lines generated: 1887 (required: 1000)
   - Attempts needed: 2
3. example29.txt:
   - Parameters: k=50, n=220, auth_density=0.15
   - Configuration: massive_balanced
   - Lines generated: 1922 (required: 1000)
   - Attempts needed: 4 (base)

These instances represent even larger and more complex problem configurations, with increased numbers of steps (up to 50), users (up to 220), and reduced authorization density (0.15). The configurations also indicate a focus on specific constraint types, such as "extra_large_mixed", "sual_focused", and "massive_balanced".

The fact that the generator required multiple attempts to successfully produce some of these instances (up to 5 attempts for example27.txt) demonstrates the tool's robustness in handling increasingly challenging problem scenarios. This ability to create a diverse set of complex instances is crucial for thoroughly testing and evaluating the performance of WSP solvers, as it pushes the boundaries of the problem space and helps identify the strengths and limitations of the solver's capabilities.

By generating these intricate problem instances, our InstanceGenerator class tool provides researchers and developers with a valuable resource for advancing the field of workflow optimization and access control, enabling them to assess the scalability, accuracy, and efficiency of their solutions across a wide range of real-world-inspired scenarios.

```
Solution Status: SAT
Wall Clock Time: 0.0330 seconds
=============================================================================================================================

Step Assignments:
        Step 1: User 1
        Step 2: User 2
        Step 3: User 1
        Step 4: User 4
        Step 5: User 5

User Step Distribution:
        User 1: Steps [1, 3]
        User 2: Steps [2]
        User 4: Steps [4]
        User 5: Steps [5]

Total Unique Users Used: 4

======================================
CONSTRAINT DETAILS
======================================

Authorization Constraints:
        Total Authorizations: 8

        Per-Step Authorization Breakdown (5 steps):
                Step 1: 1 users authorized [1]
                Step 2: 1 users authorized [2]
                Step 3: 3 users authorized [1, 3, 4]
                Step 4: 2 users authorized [4, 5]
                Step 5: 1 users authorized [5]

        Per-User Authorization Breakdown (5 users):
                User 1: authorized for 2 steps [1, 3]
                User 2: authorized for 1 steps [2]
                User 3: authorized for 1 steps [3]
                User 4: authorized for 2 steps [3, 4]
                User 5: authorized for 2 steps [4, 5]

Separation of Duty Constraints (3):
        Steps 1 and 2 must be performed by different users
        Steps 2 and 3 must be performed by different users
        Steps 1 and 5 must be performed by different users

Binding of Duty Constraints (0):
        No Binding of Duty constraints defined.

At-most-k Constraints (2):
        At most 2 steps from [1, 2, 3] can be assigned to same user
        At most 2 steps from [1, 2, 3, 4, 5] can be assigned to same user
```

Figure C1.2: Solution text file generated using CLI solution after solving and obtaining SAT.

```
One-team Constraints (0):
        No One-team constraints defined.

Super User At Least (SUAL) Constraints (0):
        No SUAL constraints defined.

Wang-Li Constraints (0):
        No Wang-Li constraints defined.

Assignment-Dependent (ADA) Constraints (0):
        No Assignment-Dependent constraints defined.
```

Figure C1.3: Solution text file generated using CLI solution after solving and obtaining SAT (continued).

# D SUPPLEMENTARY JUSTIFICATION ON SOLUTIONS

## D.1 SPECIAL UNIQUNESS

It may be noticeable that in Table 7.1 demonstrating that all instances that are SATisafiable are notably Unique solution, meaning there's no other solution other than that 1. This is justified because we have implemented our algorithms and solvers in such a way that they be extremely efficient and optimal – further justifying why we are able to obtain solutions in mere milliseconds (less 1 second).