

A Constraint-Based Optimization Approach to Prioritizing Agile Issue Backlogs

by

Tianna-Lee Salmon

BSc in Computer Science, UWI, Jamaica, 2024

A Thesis Submitted in Partial Fulfilment of
the Requirements for the Degree of

Master of Computer Science

In the Graduate Academic Unit of Computer Science

Supervisor: Francis Palma, Ph.D., Faculty of Computer Science

Examining Board: Hung Cao, Ph.D., Faculty of Computer Science, Chair
Hong Chen, Ph.D., Department of Electrical and
Computer Engineering
Wei Song, Ph.D., Faculty of Computer Science

This thesis is accepted by the
Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

December, 2025

© Tianna-Lee Salmon, 2025

Abstract

Prioritizing software artifacts such as bugs is challenging due to multiple, often conflicting criteria. Traditional approaches either rely heavily on stakeholder input or attempt full automation without adequately considering these factors. This thesis investigates the use of Google’s CP-SAT constraint solver combined with simulated user elicitation to generate issue orderings. Using historical Jira data, we evaluate how constraint configurations including priority class, issue type, dependencies, and creation date approximate a gold-standard resolution order, and we assess the impact of varying both the quantity and accuracy of user input. We conduct project-level analyses, apply constraints at the assignee level, carry over issues between sprints, and compare CP-SAT with Multi-Objective Particle Swarm Optimization and Ant Colony Optimization. Our results show increased elicitation generally improves alignment, the system is robust to errors, local prioritization enhances performance, CP-SAT usually outperforms nature-inspired methods, and issue rollover improves performance, though the method can be further refined.

Acknowledgements

I acknowledge my supervisor, Dr. Francis Palma, for providing direction and outlining the ideas that shaped this work. His guidance informed the approach taken throughout this project.

This research was supported by the New Brunswick Innovation Foundation (NBIF) through NBIF TRF award #TRF2023-003.

I also acknowledge the financial and institutional support from the Faculty of Computer Science, University of New Brunswick.

Table of Contents

Abstract	ii
Acknowledgments	iii
Table of Contents	iv
List of Tables	ix
List of Figures	xiii
List of Abbreviations	xiv
List of Appendices	xv
1 Introduction	1
2 Background	6
2.1 Analytic Hierarchy Process	6
2.2 Nature-Driven Optimization	7
2.2.1 Ant Colony Optimization (ACO)	8
2.2.2 Multi-Objective Particle Swarm Optimization (MOPSO)	8
2.2.3 Genetic Algorithms	9
2.3 Constraint-Driven Optimization	10
2.3.1 Local vs Global Constraints	11

2.3.2	The CP-SAT Solver	12
2.4	Evaluation Metrics	12
2.4.1	Disagreement	13
2.4.2	Average Distance	13
2.5	Gold Standard	14
3	Related Work	15
3.1	Search Strategy	15
3.2	Search Terms	16
3.3	Search String	17
3.4	Traditional Prioritization Methods	17
3.5	Nature-Inspired Optimization	18
3.6	Constraint-Driven Optimization	19
3.7	Learning-to-Rank Techniques	20
3.8	Discussions	21
3.8.1	AI Techniques Used	21
3.8.2	Artifacts Prioritized	21
3.8.3	Challenges and Limitations	22
3.8.4	Evaluation Metrics	22
3.8.5	Datasets Used	22
4	Research Methodology	24
4.1	Data Extraction	24
4.1.1	TAWOS Dataset	24
4.1.2	Project and Sprint Selection	25
4.2	Data Preprocessing	27
4.2.1	Issue Rollover Between Sprints	30
4.2.2	Constraint Inputs	30

4.2.3	Configurations	32
4.3	Solving Constraints	32
4.3.1	Constraint Initialization	33
4.3.2	Parameter Initialization	34
4.3.3	Initial Solution Generation	34
4.3.4	Iterative Elicitation and Refinement	35
4.4	Final Ordering	37
5	Results	40
5.1	Role of Interaction (RQ1)	42
5.1.1	Project-Level Analysis	43
5.1.2	Sprint-Level Analysis	46
5.2	Role of Constraints (RQ2)	46
5.2.1	Project-Level Analysis	47
5.2.2	Sprint-Level Analysis	50
5.3	Robustness (RQ3)	51
5.3.1	Project-Level Analysis	51
5.3.2	Sprint-Level Analysis	53
5.4	Role of Local Constraints (RQ4)	54
5.4.1	Project-Level Analysis	56
5.4.2	Sprint-Level Analysis	56
5.5	Unresolved Issue Rollover (RQ5)	57
5.5.1	Project-Level Analysis	60
5.5.2	Sprint-Level Analysis	60
5.6	Comparison to State-Of-the-Art Methods	
	(RQ6)	61
5.6.1	Project-Level Analysis	61

6	Discussion	65
6.1	Role of Interaction	65
6.2	Role of Constraints	66
6.3	Robustness	67
6.4	Impact of Assignee-Level Prioritization	67
6.5	Handling of Unresolved Issues	68
6.6	Comparison with State-of-the-Art Methods	69
6.7	Implications	69
6.8	Limitations	71
6.9	Algorithm Complexity	71
6.10	Threats to Validity	73
	6.10.1 External Validity	73
	6.10.2 Internal Validity	73
	6.10.3 Construct Validity	75
	6.10.4 Reliability Validity	75
7	Conclusion and Future Work	77
7.1	Conclusion	77
7.2	Future Work	78
	Bibliography	87
A	RQ1: Role of Interaction Statistical Analysis	88
B	RQ2: Role of Constraints Statistical Analysis	92
C	RQ3: Robustness Statistical Analysis	96
D	RQ4: Impact of Assignee-Level Prioritization Statistical Analysis	100
E	RQ5: Handling of Unresolved Issues Statistical Analysis	104

F RQ6: Comparison with State-of-the-Art Methods Statistical Analysis	106
--	-----

Vita

List of Tables

3.1	Major terms and their synonyms or alternative spellings.	16
4.1	Median and IQR values for disagreement and average distance across configurations in all projects.	26
4.2	Issue counts for the five largest sprints (S1–S5) in each project (P4, P12, P13, P25) before issue rollover	27
4.3	Issue counts for the five largest sprints (S1–S5) in each project (P4, P12, P13, P25) after issue rollover	27
4.4	Mapping of priority labels to priority classes	28
4.5	Mapping of issue types to numeric precedence values	29
4.6	Sample processed data for P2-S4	30
4.7	Simulation configurations showing varied and fixed values across research questions	33
4.8	Constructed constraints from sample data in Table 4.6, with associated weights and retractability for P2-S4	34
4.9	Sample parameters for the optimization process for P2-S4	34
4.10	Sample candidate solutions for P2-S4 and the corresponding cost breakdown	35
4.11	Sample elicitation log with disagreement pairs, elicited pairs, and counts for P2-S4. <i>The highlighted row indicates that we have reached the maximum elicitation.</i>	36

4.12	Sample final population of solutions for P2-S4 with cost breakdown, disagreement count, and average distance from gold standard ordering	36
5.1	Comparison of CP-SAT, ACO, and MOPSO median disagreement and average distances for Project 4. <i>Minimum values per sprint/project are bolded.</i>	40
5.2	Comparison of CP-SAT, ACO, and MOPSO median disagreement and average distances for Project 12. <i>Minimum values per sprint/project are bolded.</i>	41
5.3	Comparison of CP-SAT, ACO, and MOPSO median disagreement and average distances for Project 13. <i>Minimum values per sprint/project are bolded.</i>	42
5.4	Comparison of CP-SAT, ACO, and MOPSO median disagreement and average distances for Project 25. <i>Minimum values per sprint/project are bolded.</i>	43
5.5	Statistical tests for Project 4 showing significance of varying elicitations. <i>Bold p-values indicate statistical significance ($p < 0.05$).</i>	45
5.6	Statistical tests for Project 12 showing significance of varying constraints. <i>Bold p-values indicate statistical significance ($p < 0.05$).</i>	47
5.7	Statistical tests for Project 13 showing significance of varying error rates. <i>Bold p-values indicate statistical significance ($p < 0.05$).</i>	51
5.8	Statistical tests for Project 12 showing significance of global constraints (c1) versus their local counterparts (cl). <i>Bold p-values indicate statistical significance ($p < 0.05$).</i>	54
5.9	Project 4 comparisons for carry-over vs. non-carry-over. <i>Bold p-values indicate statistical significance ($p < 0.05$).</i>	57
5.10	Statistical tests for Project 25 comparing CP-SAT against MOPSO and ACO. <i>Bold p-values indicate statistical significance ($p < 0.05$).</i>	63

A.1	Statistical tests for Project 12 showing significance of varying elicita- tions. <i>Bold p-values indicate statistical significance ($p < 0.05$).</i>	89
A.2	Statistical tests for Project 13 showing significance of varying elicita- tions. <i>Bold p-values indicate statistical significance ($p < 0.05$).</i>	90
A.3	Statistical tests for Project 25 showing significance of varying elicita- tions. <i>Bold p-values indicate statistical significance ($p < 0.05$).</i>	91
B.1	Statistical tests for Project 4 showing significance of varying constraints. <i>Bold p-values indicate statistical significance ($p < 0.05$).</i>	93
B.2	Statistical tests for Project 13 showing significance of varying con- straints. <i>Bold p-values indicate statistical significance ($p < 0.05$).</i>	94
B.3	Statistical tests for Project 25 showing significance of varying con- straints. <i>Bold p-values indicate statistical significance ($p < 0.05$).</i>	95
C.1	Statistical tests for Project 4 showing significance of varying error rates. <i>Bold p-values indicate statistical significance ($p < 0.05$).</i>	97
C.2	Statistical tests for Project 12 showing significance of varying error rates. <i>Bold p-values indicate statistical significance ($p < 0.05$).</i>	98
C.3	Statistical tests for Project 25 showing significance of varying error rates. <i>Bold p-values indicate statistical significance ($p < 0.05$).</i>	99
D.1	Statistical tests for Project 4 showing significance of global constraints (c1) versus their local counterparts (cl). <i>Bold p-values indicate statisti- cal significance ($p < 0.05$).</i>	101
D.2	Statistical tests for Project 13 showing significance of global constraints (c1) versus their local counterparts (cl). <i>Bold p-values indicate statisti- cal significance ($p < 0.05$).</i>	102

D.3	Statistical tests for Project 25 showing significance of global constraints (c1) versus their local counterparts (cl). <i>Bold p-values indicate statistical significance ($p < 0.05$).</i>	103
E.1	Project 12 comparisons for carry-over vs. non-carry-over. <i>Bold p-values indicate statistical significance ($p < 0.05$).</i>	104
E.2	Project 13 comparisons for carry-over vs. non-carry-over. <i>Bold p-values indicate statistical significance ($p < 0.05$).</i>	105
E.3	Project 25 comparisons for carry-over vs. non-carry-over. <i>Bold p-values indicate statistical significance ($p < 0.05$).</i>	105
F.1	Statistical tests for Project 4 comparing CP-SAT against MOPSO and ACO. <i>Bold p-values indicate statistical significance ($p < 0.05$).</i>	107
F.2	Statistical tests for Project 12 comparing CP-SAT against MOPSO and ACO. <i>Bold p-values indicate statistical significance ($p < 0.05$).</i>	108
F.3	Statistical tests for Project 13 comparing CP-SAT against MOPSO and ACO. <i>Bold p-values indicate statistical significance ($p < 0.05$).</i>	109

List of Figures

4.1	Overview of proposed methodology	25
5.1	Median Disagreement (a) and Median Average Distance (b) across varying elicitations for P4	44
5.2	Median Disagreement (a) and Median Average Distance (b) across varying constraint configurations for P12	48
5.3	Median Disagreement (a) and Median Average Distance (b) showing varying error rates for P13	52
5.4	Median Disagreement (a) and Median Average Distance (b) comparing local and global constraint configurations for P12	55
5.5	Median Disagreements for Carry-Over (a) and Non-Carry-Over (b) showing impact of issue rollover at varying elicitations for P4	58
5.6	Median Average Distances for Carry-Over (a) and Non-Carry-Over (b) showing impact of issue rollover at varying elicitations for P4	59
5.7	Median Disagreement (a) and Median Average Distance (b) across different optimization methods for P25	62

List of Abbreviations

CP-SAT – Constraint Programming with Boolean Satisfiability

SMT – Satisfiability Modulo Theories

PO – Product Owner

ACO – Ant Colony Optimization

MOPSO – Multi-Objective Particle Swarm Optimization

GA – Genetic Algorithm

List of Appendices

RQ1: Role of Interaction Statistical Analysis	88
RQ2: Role of Interaction Statistical Analysis	92
RQ3: Robustness Statistical Analysis	96
RQ4: Impact of Assignee-Level Prioritization Statistical Analysis	100
RQ5: Handling of Unresolved Issues Statistical Analysis	104
RQ6: Comparison with State-of-the-Art Methods Statistical Analysis	106

Chapter 1

Introduction

In software development teams, efficiently managing resources is essential to maximize business value. However, determining which tasks should take priority is inherently subjective, as different stakeholders often value different aspects of the work. For example, a CEO may prioritize initiatives with the highest potential business impact while a lead developer may focus on technical dependencies or maintainability. As a result, the final ordering of tasks can vary depending on who is making the decision.

To ensure software success, it is necessary to make holistic decisions that balance these competing factors, including customer value, business impact, time constraints, financial limitations, and team capacity. In agile environments, such as Scrum, the Product Owner is responsible for ordering the product backlog to maximize the product's overall value. Ineffective prioritization can be costly, leading to financial losses, delayed releases, or diminished product quality. Regardless of the type of software artifact being prioritized (e.g., requirements, bugs, or tasks), manually maintaining an ordering is time-consuming and inefficient. Automating this process can significantly improve efficiency and consistency while giving teams greater control over the factors used for prioritization.

Traditional prioritization methods fall into two main categories: manual and automated. Manual approaches, such as pairwise comparisons using the Analytic Hierarchy Process (AHP) [44, 41] and its variants, such as Incomplete AHP (IAHP) [37], allow stakeholders to prioritize at a granular level. However, while intended to improve efficiency, these methods do not scale well due to the inherently high level of user elicitation required, as noted by prior work [55, 37, 39].

Automated techniques attempt to overcome these scalability issues but introduce their own challenges. For instance, nature-inspired techniques like Interactive Genetic Algorithms (IGA) [52, 51, 22], Ant Colony Optimization (ACO) [1, 18, 21], and Multi-Objective Particle Swarm Optimization (MOPSO) [34, 1, 15] are metaheuristic, meaning they may not always guarantee optimal solutions or perfect alignment with subjective stakeholder priorities. Similarly, Machine learning (ML) methods, such as Learning-to-Rank [48, 23], may fail to adhere to hard constraints, such as maintaining strict technical dependencies between issues, since their optimization process focuses on statistical patterns in the training data rather than explicitly enforcing domain-specific rules. Additionally, the explainability of these models can be an issue, since their decision processes often emerge from complex learned interactions that are difficult to interpret [13].

A promising but less explored alternative is the use of constraint solvers combined with AHP-style user elicitation [37, 55]. This approach incorporates the factors stakeholders commonly consider when prioritizing issues, such as priority class and issue type, while still allowing user input when the solver’s recommendation differs. By drawing on historical data, the solver learns which combinations of feature weightings have produced effective orderings in the past and uses this information to guide prioritization for future iterations.

This thesis aims to:

- Evaluate the impact of user elicitation on alignment with historical issue ordering, which serves as the gold standard;
- Examine the influence of constraints on prioritization;
- Evaluate the robustness of the system to errors in user input;
- Compare system performance when constraints are applied locally at the assignee level versus globally, reflecting how prioritization typically occurs in real-world scenarios;
- Compare system performance when unresolved issues are carried into subsequent sprints versus when they are not, simulating how work naturally rolls over in real-world sprint planning; and,
- Compare the CP-SAT approach with state-of-the-art optimization techniques, specifically Ant Colony Optimization (ACO) and Multi-Objective Particle Swarm Optimization (MOPSO).

In order to address these objectives, we investigate the following research questions:

RQ1 (Role of Interaction): *Does the interactive method improve issue prioritization compared to the non-interactive method in agile software development?*

Findings: Increasing the number of user elicitations generally improved alignment with the gold-standard ordering, highlighting the value of stakeholder input in issue prioritization.

RQ2 (Role of Constraints): *How do different constraint configurations affect the alignment of the final ordering with the gold standard?*

We investigate RQ2 through two sub-questions:

- **RQ2.1 (Varying Constraints):** *To what extent does adding or removing certain constraints influence the alignment of the final ordering with the gold standard?*
- **RQ2.2 (Varying Weights):** *How does adjusting constraint weights affect the alignment of the final ordering with the gold standard?*

Findings: Both the selection and weighting of constraints influenced alignment with the gold standard, showing that considerations of the appropriate combination of factors are crucial for achieving realistic prioritization.

RQ3 (Robustness): *How robust is the proposed method with respect to errors committed by the analyst during the interactive session?*

Findings: Our proposed method generally remained robust even when 20% of user inputs were incorrect, almost consistently outperforming the baseline approach that orders issues solely by creation date.

RQ4 (Impact of Assignee-Level Prioritization): *How does applying constraints at the assignee level influence prioritization results compared to applying them across the entire backlog?*

Findings: Applying constraints at the assignee level generally improved alignment, suggesting that local prioritization more accurately reflects real-world prioritization efforts.

RQ5 (Handling of Unresolved Issues): *How effectively does the proposed approach manage unresolved issues that are carried forward across sprints in simulated real-world backlog conditions?*

Findings: Simulating the carry-over of unresolved issues generally improved alignment with the gold standard and produced more consistent prioritization outcomes, though

further research is needed to confirm these results with a more robust carry-over method.

RQ6 (Comparison with State-of-the-Art Methods): *How does the proposed approach perform relative to state-of-the-art prioritization methods?*

Findings: CP-SAT exhibited overall better alignment with the gold standard compared to metaheuristic optimization methods: ACO and MOPSO, likely due to its ability to simultaneously incorporate both stakeholder input and constraints.

Overall, our findings indicate that interactive elicitation enhances prioritization performance up to a moderate level, after which the benefits plateau. Constraint selection and weighting significantly affect alignment with the gold standard, and the system remains robust, although performance may sometimes decline at higher error rates ($\approx 20\%$). Applying constraints at the assignee level generally improves performance, and carrying unresolved issues across sprints improves outcomes. Finally, the CP-SAT approach proves effective, generally outperforming state-of-the-art metaheuristic methods: MOPSO and ACO under the evaluated conditions.

This thesis is organized as follows: Section 2 presents background information relevant to this work, followed by a review of related work in Section 3. Section 4 describes the methodology and the proposed solution. Section 5 presents results, which are then discussed in Section 6. Finally, Section 7 concludes the thesis and highlights future work.

Chapter 2

Background

This section presents the foundational concepts and methods relevant to our approach. We review the Analytic Hierarchy Process, nature-driven optimization techniques, and constraint-driven techniques. Finally, we describe the evaluation metrics used to quantify prioritization quality, providing the formal definitions needed to interpret the results.

2.1 Analytic Hierarchy Process

The Analytic Hierarchy Process (AHP) is a structured decision-making methodology that decomposes complex problems into a hierarchy of criteria and alternatives [42]. At each level of the hierarchy, decision-makers perform pairwise comparisons between elements, indicating the relative importance or preference of one element over another. These comparisons are then used to compute numerical weights that quantify the relative priority of each element. By aggregating these weights across the hierarchy, AHP produces a ranked ordering of alternatives that reflects the combined influence of multiple criteria, providing a systematic way to translate qualitative judgments

into quantitative scores.

To derive the weights, AHP constructs a pairwise comparison matrix for each set of elements at the same hierarchical level. Each entry in the matrix represents the relative preference of one element over another, typically using a numerical scale (e.g., 1–9). The principal eigenvector of this matrix is then computed, and its components are normalized to sum to one, producing the relative weights for the elements. AHP also includes a consistency check to measure the coherence of the pairwise judgments. If inconsistencies exceed a predefined threshold, decision-makers may be prompted to revise their comparisons. Finally, the weights are propagated through the hierarchy using a weighted sum, combining the relative importance of criteria with the performance of alternatives to generate an overall ranking.

While AHP effectively captures subjective user input, its reliance on manual elicitation can limit scalability and adaptability in dynamic or large-scale environments [44, 39]. Specifically, for N issues, AHP requires $N \times (N - 1)/2$ pairwise comparisons, which grow quadratically and become increasingly burdensome as the number of issues increases.

2.2 Nature-Driven Optimization

Nature-driven optimization techniques are inspired by natural processes and biological behaviors. They belong to the broader class of metaheuristics, which are problem-solving strategies designed to find a sufficiently good solution to an optimization problem, particularly when exact methods are too slow or impractical.

2.2.1 Ant Colony Optimization (ACO)

ACO is inspired by the foraging behavior of ants, which deposit pheromones to communicate paths to food sources [21, 1]. In optimization, artificial “ants” construct candidate solutions probabilistically, guided by pheromone trails that are updated based on the quality of the solutions. Over multiple iterations, the algorithm converges toward high-quality solutions by reinforcing successful paths while allowing for the exploration of alternative solutions.

The probability that ant k moves from component i to component j is given by

$$P_{ij}^k = \frac{(\tau_{ij})^\alpha (\eta_{ij})^\beta}{\sum_{l \in \mathcal{N}_i} (\tau_{il})^\alpha (\eta_{il})^\beta},$$

where τ_{ij} is the pheromone intensity, η_{ij} is the heuristic desirability, and α and β control their relative influence.

After each iteration, pheromone levels are updated using

$$\tau_{ij} \leftarrow (1 - \rho)\tau_{ij} + \sum_k \Delta\tau_{ij}^k,$$

where ρ is the evaporation rate and $\Delta\tau_{ij}^k$ represents the pheromone deposited by ant k based on the quality of its solution [21].

2.2.2 Multi-Objective Particle Swarm Optimization (MOPSO)

MOPSO extends the Particle Swarm Optimization (PSO) framework [22] to multi-objective problems [34, 15]. Each particle represents a candidate solution and moves through the solution space influenced by its own best-known position as well as the

best-known positions of other particles. The algorithm maintains a Pareto front of non-dominated solutions, allowing simultaneous optimization of multiple objectives.

Particle velocities and positions are updated using

$$v_i(t+1) = \omega v_i(t) + c_1 r_1 (p_i - x_i(t)) + c_2 r_2 (g_i - x_i(t)),$$

$$x_i(t+1) = x_i(t) + v_i(t+1),$$

where p_i is the particle's personal best position, g_i is a leader chosen from the archive, ω is the inertia weight, and c_1, c_2 are acceleration coefficients.

The Pareto archive maintains non-dominated solutions, and selecting leaders from this archive encourages both convergence and diversity in the set of trade offs [15].

2.2.3 Genetic Algorithms

Genetic algorithms are inspired by the process of natural selection. Candidate solutions, represented as “chromosomes”, undergo operations analogous to biological evolution, including selection, crossover, and mutation. Over successive generations, populations evolve toward increasingly optimal solutions. Genetic algorithms are particularly effective for exploring large, complex search spaces and handling multiple, conflicting objectives [6]. A variant, Interactive Genetic Algorithms (IGAs), relaxes the requirement for fully specifying candidate solutions or constraints, allowing the algorithm to operate effectively in situations with incomplete or partially defined problem information. IGAs are useful when the search space is vast or not fully known, providing flexibility while still drawing upon the evolutionary principles of standard genetic algorithms [52, 51].

2.3 Constraint-Driven Optimization

Constraint-driven optimization leverages constraints to guide the search for optimal solutions within a defined problem space [37, 55]. Constraints specify the conditions relevant to the problem and shape how the solver evaluates candidate solutions. They can be categorized as follows:

- **Non-retractable (Hard) constraints:** These constraints must be strictly satisfied in all candidate solutions. Any violation $v_i(\mathbf{x}) > 0$ renders a solution *infeasible*, and the solver discards such solutions. Only assignments where all hard constraints are satisfied ($v_i(\mathbf{x}) = 0$ for all i) are considered *feasible*. Among feasible solutions, the solver seeks an *optimal* one by minimizing the total violation cost from any remaining soft constraints.
- **Retractable (Soft) constraints:** These constraints can be relaxed at a cost, allowing the solver to balance competing requirements by incurring weighted violation penalties w_i . The total violation is captured by the cost function defined in Equation 2.1.

The cost function integrates these considerations, enabling the solver to identify an ordering that reduces disagreement and positional deviation (discussed in Section 2.4), provided that the constraints align with elicited stakeholder preferences, revealing whether the modeled rules accurately reflect real-world prioritization behavior.

In practice, satisfying all constraints is not always possible, as most optimization problems involve complex or conflicting scenarios. The solver minimizes an objective (or cost) function that penalizes constraint violations according to their assigned weights. A solution with zero cost does not necessarily represent the most desirable prioritization; it only indicates that all constraints are formally satisfied within the

solver’s model. This distinction highlights the difference between formal constraint satisfaction and actual stakeholder preferences.

Formally, let \mathbf{x} denote a candidate prioritization (an ordering of tasks), and let $\mathcal{C} = \{c_1, c_2, \dots, c_n\}$ represent a set of constraints. Each constraint c_i has an associated weight $w_i \geq 0$ indicating its relative importance. The solver minimizes a cost function $f(\mathbf{x})$, defined as the weighted sum of constraint violations (Equation 2.1) [37]:

$$f(\mathbf{x}) = \sum_{i=1}^n w_i \cdot v_i(\mathbf{x}), \quad (2.1)$$

where $v_i(\mathbf{x}) \geq 0$ measures the violation of constraint c_i under solution \mathbf{x} . A violation of zero indicates complete satisfaction of the constraints and does not contribute to the cost function.

2.3.1 Local vs Global Constraints

Constraints can be applied at varying levels of granularity within the prioritization process. In this thesis, we investigate the effects of applying constraints in two ways:

1. **Globally:** Global constraints are applied across the entire sprint, without consideration of the individual assignees. In this approach, the system evaluates all issues collectively and constraints are, therefore, enforced whenever the relevant conditions hold, regardless of who is assigned to each issue. For example, a constraint based on issue creation date would consistently prioritize earlier issues over later ones across the whole sprint, independent of the assignees responsible for them.
2. **Locally:** Local constraints are applied at the level of individual assignees, ensuring that issue prioritization respects each assignee’s own prioritization

decisions. For instance, given a set of issues assigned to a single developer during a sprint, the developer may determine the order in which to complete only that subset of issues. Consequently, rather than universally prioritizing issue I1 over I2 based on a constraint such as creation date, this constraint is enforced only when both issues are assigned to the same individual. When issues are allocated to different assignees, the constraint may not accurately determine their relative order, as each assignee would progress through their tasks independently. Therefore, this method of applying the constraints, further analyzed in *RQ4*, evaluates the impact of real-world issue assignment practices on prioritization outcomes.

2.3.2 The CP-SAT Solver

The CP-SAT solver¹ by Google is a constraint programming tool that integrates Boolean satisfiability (SAT) techniques with optimization capabilities. CP-SAT efficiently searches for feasible solutions that satisfy hard constraints while minimizing violations of soft constraints. It supports flexible modeling of both local and global constraints, allows for weighted violations, and can optimize multiple objectives simultaneously. These features make it well-suited for evaluating the impact of the various research questions explored in this thesis.

2.4 Evaluation Metrics

We evaluated the performance of the issue prioritization algorithm across multiple projects and sprints. Each simulation (see Table 4.7 for configurations) was run 15 times, and for every run, the median disagreement and average distance metrics were

¹https://developers.google.com/optimization/cp/cp_solver

computed from a population of 10 solutions.

In the context of this study, the following two key metrics were used to evaluate the accuracy of a candidate ordering:

2.4.1 Disagreement

Disagreement quantifies the difference between the solver’s ordering of issues, Ord_1 , and the gold standard ordering, Ord_2 , over the same set R . It counts the number of pairs of issues that are ordered oppositely in the two sets of solutions [20, 37]. Formally, disagreement is defined as the size of the set:

$$\text{Disagreement}(Ord_1, Ord_2) = \{(R_i, R_j) \in Ord_1 \mid (R_j, R_i) \in Ord_2\}$$

where (R_i, R_j) implies that issue R_i precedes R_j in ordering Ord_1 , while the reverse order holds in Ord_2 .

A lower disagreement value indicates closer alignment to the gold standard in terms of pairwise ordering.

2.4.2 Average Distance

Average distance measures the average positional difference between each issue in a candidate ordering \mathbf{x} and its corresponding position in a gold standard ordering \mathbf{g} [20, 37]. It is defined as:

$$\text{AverageDistance}(\mathbf{x}, \mathbf{g}) = \frac{1}{n} \sum_{i=1}^n |\mathbf{x}_i - \mathbf{g}_i|$$

where:

- n is the total number of issues,
- \mathbf{x}_i is the position of the i^{th} issue in the candidate ordering,
- \mathbf{g}_i is the position of the same issue in the gold standard.

A lower average distance indicates closer alignment to the gold standard in terms of absolute issue positions.

2.5 Gold Standard

In this thesis, the *gold standard* represents a reference ordering of issues that reflects the prioritization observed in historical project data. It serves as the benchmark against which candidate solutions produced by the prioritization algorithms are evaluated.

For each project and sprint, the gold standard is constructed based on the actual resolution order of issues, capturing the sequence in which tasks were completed in the past. While true optimality is infeasible in this prioritization context due to subjectivity and the need to balance multiple factors such as urgency, complexity, and cost, the historical resolution order implicitly reflects considerations that may not be present in the data, making it a reasonable benchmark for comparison.

Formally, let $\mathbf{g} = [g_1, g_2, \dots, g_n]$ denote the gold standard ordering of n issues in a given sprint. Candidate solutions \mathbf{x} generated by the solver are then compared to \mathbf{g} using the disagreement and average distance metrics defined in Section 2.4. These metrics quantify the degree to which a proposed ordering matches the historical resolution order, allowing for a systematic evaluation of prioritization accuracy.

Chapter 3

Related Work

This section reviews existing studies relevant to AI-based prioritization of software artifacts, highlighting the main approaches, findings, and limitations reported in the literature as well as other related work that provides a good background for this thesis.

3.1 Search Strategy

We conducted an in-depth literature review prior to this research and identified a clear gap in the use of constraint solvers for the prioritization problem. Our search strategy involved querying the following databases: ACM Digital Library, IEEE Xplore, SpringerLink, and Google Scholar.

The following objectives guided our literature review on the use of artificial intelligence for artifact prioritization in software engineering:

- Identify the AI techniques used for prioritizing artifacts;
- Identify the types of artifacts (e.g., requirements, issues, test cases) to which

Table 3.1: Major terms and their synonyms or alternative spellings.

Major Term	Synonyms/Alternate Spellings
AI	Artificial Intelligence, Machine Learning, Deep Learning, Neural Networks, Reinforcement Learning, Natural Language Processing
Software Engineering	Software Development, Software Projects, Software Systems, Agile Development
Prioritization	Ranking, Classification, Ordering, Optimization
Evaluation	Performance, Assessment
Artifact	Feature, Requirement, Bug, Test Case, Issue, Defect, Fault

these techniques have been applied;

- Identify the key challenges and limitations reported in existing AI-based prioritization studies;
- Identify the evaluation metrics used to assess the effectiveness of AI-driven prioritization methods; and
- Identify the datasets commonly used in research on AI-based prioritization.

3.2 Search Terms

The following steps were taken to build the search terms, outlined in Table 3.1:

1. Identify major terms from the research topic.
2. Include synonyms and alternative spellings.
3. Use Boolean operators (AND, OR) to refine search queries.

3.3 Search String

The following is the search string generated from Table 3.1:

```
("AI" OR "Artificial Intelligence" OR "Machine Learning" OR "Deep Learning"  
OR "Neural Networks" OR "Reinforcement Learning" OR "Natural Language  
Processing")  
AND ("Software Engineering" OR "Software Development" OR "Software  
Projects" OR "Software Systems" OR "Agile Development")  
AND ("Prioritization" OR "Ranking" OR "Classification" OR "Ordering" OR  
"Optimization")  
AND ("Artifact" OR "Feature" OR "Issue" OR "Bug" OR "Defect" OR "Fault"  
OR "Requirement" OR "Test Case")
```

3.4 Traditional Prioritization Methods

Manual prioritization methods rely heavily on stakeholder judgment to rank software artifacts, such as requirements, tasks, or issues. These approaches are among the earliest in requirements engineering and project management, offering transparency and interpretability. Common methods include the Analytic Hierarchy Process (AHP) [42, 44], where stakeholders perform pairwise comparisons between requirements to derive priority weights, and Cumulative Voting [11] (also known as the “100-point method”), in which decision-makers distribute points across requirements according to their perceived importance. MoSCoW prioritization [2] (Must Have, Should Have, Could Have, Will Not Have) is another widely used technique that relies on categorical priority labels rather than numeric weighting.

While intuitive and engaging for stakeholders, these methods are time-consuming, subjective, and scale poorly as the number of requirements grows [39, 5]. In dynamic or large-scale environments, manually maintaining consistent rankings becomes infeasible.

Additionally, reliance on expert input introduces variability across projects and teams, motivating the adoption of automated, data-driven prioritization approaches.

3.5 Nature-Inspired Optimization

Nature-inspired optimization techniques, including Genetic Algorithms (GA), Particle Swarm Optimization (PSO), Whale Optimization Algorithm (WOA), and Ant Colony Optimization (ACO), have been applied to software artifact prioritization, generally outperforming manual methods such as AHP [4, 52, 51].

For instance, Nazir et al. [34] applied a Multi-Objective Particle Swarm Optimization (MOPSO) method to organize test cases, maximizing early fault detection while minimizing execution time and cost. MOPSO outperformed Multi-Objective Genetic Algorithms (MOGA), largely because GAs struggle with complex tasks due to the combinatorial explosion caused by multiple mutating components. Comparative studies further show that while ACO can surpass MOPSO, both approaches exhibit strong global optimization capabilities [1].

Genetic algorithms are also commonly used to prioritize software artifacts, such as test cases, by guiding the selection of the next element while respecting overall prioritization constraints [6]. Tonella et al. [52, 51] introduced an Interactive Genetic Algorithm (IGA) for requirements prioritization, comparing it to the Incomplete Analytic Hierarchy Process (IAHP) [52]. Their results show IGA outperforms IAHP by integrating predefined priorities and stakeholder preferences to align outcomes with user expectations. Notably, this approach is scalable and effective even in non-interactive settings. WOA has also been applied to requirement prioritization, where candidate solutions are iteratively updated using whale foraging strategies to optimize ordering under stakeholder constraints, and it was shown to outperform

AHP [4].

3.6 Constraint-Driven Optimization

Constraint-driven optimization is another approach employed for prioritizing software artifacts. Previous work mainly leveraged Satisfiability Modulo Theory (SMT) solvers, which assess the satisfiability of logical formulas [7], for software verification tasks [30, 9, 35]. However, a focused line of research has applied SMT to requirements prioritization [37, 55], with the most recent work in this area using Z3, a widely used SMT solver developed by Microsoft Research [20]. This prior work forms the foundation for the approach explored in this thesis.

While effective, SMT solvers are primarily designed for satisfiability checking, so their adaptation for optimization can introduce scalability challenges. Early work encoded prioritization as a MAX-SAT problem using a Yices-based solver¹, aiming to maximize the total weight of satisfiable clauses [37]. This approach yielded better results than IAHP and IGA for small datasets (up to 49 requirements). Later, reformulating the problem as a CHECK-SAT instance and solving it with Z3 improved solution quality metrics, namely disagreement and average distance, highlighting a trade-off between optimality and scalability [10, 55].

In contrast, hybrid solvers like Google’s CP-SAT², which combines Constraint Programming with SAT-solving techniques, address scalability more directly and have been shown to produce optimal solutions efficiently [32, 28]. As a result, this thesis employs CP-SAT, whose efficiency enables the large-scale experimentation needed to investigate the six research questions.

Although nature-inspired and constraint-driven optimization methods differ in how

¹<https://yices.csl.sri.com/>

²https://developers.google.com/optimization/cp/cp_solver

they search the solution space, both ultimately aim to produce high-quality orderings of software artifacts by balancing multiple, often conflicting, objectives. This suggests a shared framework of multi-objective prioritization under constraints. Building on prior work, this thesis examines how a modern constraint-based approach can improve scalability and solution quality in prioritization, and evaluates its effectiveness by comparing it with two established nature-inspired techniques, ACO and MOPSO.

3.7 Learning-to-Rank Techniques

Learning-to-Rank (LTR) represents a class of machine learning techniques originally developed for search engines and recommendation systems [33, 54], and has more recently been applied to software engineering prioritization tasks [48, 39, 36, 23]. Rather than relying on explicit stakeholder scoring, LTR models learn to infer the relative importance of items (such as requirements, defects, or tasks) from historical data. The input typically consists of feature representations of items (e.g., issue metadata such as priority class, dependencies, or past resolution times), along with a reference ordering or set of pairwise preferences derived from previous prioritization decisions.

LTR approaches can be broadly categorized into pointwise, pairwise, and listwise frameworks. Pointwise methods predict an absolute priority score for each item; pairwise methods (such as RankSVM or RankNet) learn which of two items should come first; and listwise methods optimize directly for the quality of the overall ranked list. These models produce an explicit ordering or partial ordering of items, which can then guide decision-making or serve as input to higher-level optimization processes [23].

Compared to manual methods, LTR provides a scalable and adaptive mechanism for

prioritization that can generalize from historical data and continuously improve as more feedback is collected. However, it also introduces challenges related to model interpretability, accuracy, and data availability, especially in domains where labeled historical rankings are sparse or inconsistent [39].

3.8 Discussions

Based on the systematic literature review conducted, the following trends and gaps were identified in AI-based test artifact prioritization:

3.8.1 AI Techniques Used

Traditional machine learning models, particularly Random Forest (RF) [3] and Support Vector Machines (SVM) [25], are widely used for prioritization tasks due to their ability to learn from historical execution data. Deep learning approaches, including CNNs [53] and DNNs [19, 47], are increasingly being explored, indicating growing interest in capturing more complex patterns. Reinforcement Learning (RL) [12, 17] and metaheuristic approaches such as the Firefly Algorithm (FA) [27], Ant Colony Optimization [21] and the Whale Optimization Algorithm [4], support adaptive prioritization strategies. Clustering methods (Hierarchical Clustering [29]) are also employed, though less frequently.

3.8.2 Artifacts Prioritized

Test cases are the most common target for AI-based prioritization [8, 38, 46], likely because ground truth is more readily available than for other artifacts, such as requirements [24, 26] or bug reports [3, 47]. This highlights a potential area for future

research.

3.8.3 Challenges and Limitations

Key limitations include sensitivity to dataset characteristics, hyperparameter tuning, and variability in feature importance. RF [3] models, for example, may over-rely on specific features, thereby reducing adaptability. Deep learning models, such as DNNs [47, 19], can suffer from catastrophic forgetting or performance variability, particularly in high-volatility environments or when working with limited historical data. RL [12, 17] approaches heavily depend on reward function design, and metaheuristic methods such as FA [27] may not always yield optimal solutions. Computational cost and interpretability also pose challenges, particularly for complex models [13].

3.8.4 Evaluation Metrics

For test case prioritization, most studies use APFD [34, 48] or its variants. Bug prioritization is often treated as a classification problem, predicting bug severity [47], and evaluated using standard model-based metrics such as accuracy and precision. Requirements prioritization, which has been studied less, typically relies on disagreement or average distance metrics [55, 37], along with model-based metrics [24].

3.8.5 Datasets Used

Studies on both test case and bug prioritization predominantly use real-world datasets, such as Defects4J [31], Bugzilla [3], and IOF/ROL [45], sometimes augmented with synthetic data [45]. In contrast, requirements prioritization has been less studied and does not have a widely adopted dataset.

Overall, while AI-based prioritization shows promise, the specifics of the methods used often depend on the type of prioritization task and the artifacts involved. This thesis addresses a key gap in existing approaches: producing explicit, constraint-aware prioritization orderings that are artifact-independent, building primarily on prior work done on requirements prioritization [55, 37].

Chapter 4

Research Methodology

This section presents the overall methodology employed in our study. We first describe the dataset and preprocessing steps, followed by the experimental setup and evaluation metrics. Our approach is structured to ensure reproducibility and rigorous comparison of the proposed methods.

Figure [4.1](#) provides an overview of the steps taken in this methodology.

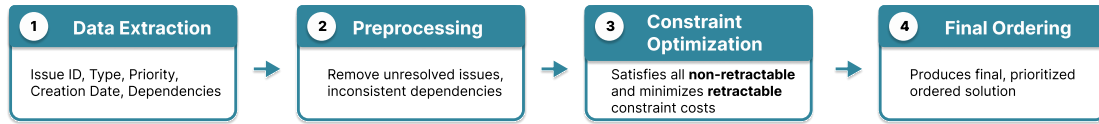
4.1 Data Extraction

This section presents our dataset selection and preparation, along with the proposed algorithm and a motivating example.

4.1.1 TAWOS Dataset

The TAWOS dataset [\[49\]](#) has primarily been applied to story point and agile effort estimation [\[40, 50\]](#). However, given its detailed issue-level data from Jira projects, such as associations with specific sprints and projects, and inter-issue relationships,

Figure 4.1: Overview of proposed methodology



including dependencies, it is well-suited for prioritization tasks. The metadata from this dataset enabled us to identify the factors that influenced the final issue ordering in past sprints, which we implemented as constraints in our approach. This makes it ideal for our research, as it contains information that is likely to influence stakeholder preferences, providing a rich set of real-world factors to model and test our prioritization approach effectively.

For clarity, we denote these constraints using a shorthand format: c , p , t , and d correspond to *creation date*, *priority class*, *type class*, and *dependency*, respectively. In this notation, numbers following each letter indicate the relative weight of that constraint in the prioritization.

4.1.2 Project and Sprint Selection

Given a set of issues from an Agile sprint, the objective was to generate an ordering that closely approximates the actual resolution order, referred to as the gold standard. This gold standard was derived from the historical sequence in which developers actually resolved issues. Each issue is associated with metadata, including priority class, creation date, issue type, and declared dependencies, which are used to formulate constraints for the optimization process. Based on the best-performing constraint configuration across all projects, we then evaluated the extent to which pairwise elicitation could further refine the ordering.

This best-performing configuration, $c1p1$, was selected based on the trends observed in Table 4.1. No configuration clearly dominated across all four metrics, so $c1p1$ was

Table 4.1: Median and IQR values for disagreement and average distance across configurations in all projects.

Configuration	Overall Median Disagreement	IQR of Disagreement	Overall Median Avg Distance	IQR of Avg Distance
b	500.75	432.75	14.15	7.33
c1	494.75	425.25	14.15	7.48
c1p1	451.00	410.88	13.25	7.27
c1p1t1	488.25	387.88	14.18	6.63
c1p1t2	458.25	434.38	13.89	7.69
c1p2t1	449.75	461.75	13.70	7.02
c1t1	475.50	349.25	13.51	6.16
c2p1t1	502.50	439.63	14.19	7.42

chosen as a balanced compromise, minimizing disagreement and average distance while maintaining consistency. **c1t1**, which was the most stable across both metrics and performed well overall, would have also been a reasonable choice. Nevertheless, **c1p1** was selected for its favourable balance across all evaluated criteria. It is important to note that this constraint was chosen based on performance before issue rollover, and the **c1p1** configuration was consistently used throughout this research.

Our methodology was applied to the five largest sprints (denoted as S1–S5) across four Jira-based software projects (denoted as P4, P12, P13, and P25). Each of the four projects focuses on a different domain, providing a diverse yet relevant set of projects for analysis. Apache Mesos (P4) is a cluster management system, Hyperledger Indy Node (P25) is a platform for decentralized digital identity on blockchains, Appcelerator Studio (P13) is an IDE for cross-platform mobile app development, and the Titanium SDK (P12) is a software development kit that provides the underlying tools for building those mobile apps. These projects offer a varied foundation for exploring how software development teams across different technological domains may approach issue prioritization.

Table 4.2: Issue counts for the five largest sprints (S1–S5) in each project (P4, P12, P13, P25) before issue rollover

Project ID	Project Name	S1	S2	S3	S4	S5
P4	Apache Mesos	80	67	57	54	51
P12	The Titanium SDK	92	59	51	48	44
P13	Appcelerator Studio	48	49	36	34	34
P25	Hyperledger Indy Node	42	35	29	30	29

Table 4.3: Issue counts for the five largest sprints (S1–S5) in each project (P4, P12, P13, P25) after issue rollover

Project ID	Project Name	S1	S2	S3	S4	S5
P4	Apache Mesos	72	67	60	57	54
P12	The Titanium SDK	82	62	52	48	45
P13	Appcelerator Studio	43	48	37	35	34
P25	Hyperledger Indy Node	37	36	30	30	29

4.2 Data Preprocessing

Prior to analysis, a series of pre-processing steps were applied to ensure that only meaningful issue records were passed into the solver:

- Resolution Status and Validity:** Only issues with non-null resolution and resolution date fields, and with resolution values in *Complete*, *Fixed*, *Done*, *Implemented*, *Resolved*, *Deployed*, *Completed* were retained. Issues with statuses such as *Invalid* or *Won't Fix* were excluded, as these entries typically indicate that an issue was dismissed rather than actively addressed, and thus do not reflect tasks that were actually resolved. Including them could introduce noise, i.e., triaged vs. completed work.
- Rapid Resolution Cases:** Issues with a priority class of 1–3 that were resolved in under one hour were excluded to reduce bias from issue tickets that may have been retroactively created. In this context, genuine task execution refers to work that was actively performed and tracked within the system. Issues

Table 4.4: Mapping of priority labels to priority classes

Priority Label	Priority Class
Blocker, Blocker - P1, Highest	1
Critical, Critical - P2, High	2
Major, Major - P3, Medium	3
Minor, Minor - P4, Low	4
Trivial, Trivial - P5, Lowest	5

resolved within brief periods may instead reflect administrative updates or post-hoc logging, which can distort the solver’s interpretation of prioritization.

- **Missing Priority Values:** Issues with missing priority values were excluded. This decision aligns with our objective to reduce bias introduced by assumptions about missing data. Future work could explore principled inference techniques, such as training lightweight classifiers using labeled historical data.
- **Priority Class:** The dataset provides issue priority labels in text form. These were mapped to a numerical scale to represent precedence levels. This mapping groups semantically similar priority labels into a unified numerical value, where lower values indicate higher priority, as shown in Table 4.4.
- **Type Class:** The dataset provides issue types in textual format. To establish a numerical precedence order, each issue type was mapped onto a scale from 1 to 17 based on the actual average resolution time, with 1 indicating the fastest resolution and 17 the slowest. This mapping is illustrated in Table 4.5.
- **Dependencies:** To extract task dependencies from linked issues, we focused on JIRA link types that clearly imply task precedence or blocking relationships. Only dependencies associated with issues meeting the valid resolution criteria were included to maintain data consistency. Furthermore, only those with semantics indicating an ordering were considered valid dependencies. These were extracted by identifying specific keywords within the names of JIRA

Table 4.5: Mapping of issue types to numeric precedence values

Issue Type	Type Class	Issue Type	Type Class
Release	1	Task	10
Sub-task	2	Suggestion	11
Story	3	Support Request	12
Technical task	4	New Feature	13
Investigation	5	Technical Debt	14
Test Task	6	Epic	15
Enhancement Request	7	Wish	16
Improvement	8	Documentation	17
Bug	9		

issue links. Link names were included if they contained any of the following substrings: *Gantt*, *Depend*, *Block*, *Required*, *Complete*, *Child*, *Parent*, or *Follow*.

These link names were then mapped into two categories depending on their description:

- **Dependent:** *dependent, depends on, has to be done after, is blocked by, requires, blocked, is fixed by, depends upon, dependent, start is earliest end of, FS-depends on, SS-depends on, FF-depends on, is dependent of, follows*
- **Dependency:** *is depended on by, has to be done before, blocks, fixes, is depended upon by, is required by, earliest end is start of, required by, is FS-depended by, is FF-depended by, is SS-depended by, followed by, depended on by*

Link types such as *Reference*, *Documented*, or *Tested* were excluded due to their semantic ambiguity and potentially inconsistent use across teams. Future work may investigate the role that soft dependencies, such as these, play in the prioritization process. To simulate a realistic development environment, dependencies that contradicted the observed issue resolution order were excluded. This assumes that project team members would have an accurate semantic understanding of dependencies in their context, unlike the generalized nature

of a cross-project dataset such as TAWOS.

Table 4.6 shows an example of the data after preprocessing, using P2-S4 as a reference. The *dependencies* column lists the issues that must be completed *before* the associated issue ID.

Table 4.6: Sample processed data for P2-S4

Project	Sprint	Issue ID	Priority	Type Class	Creation Date	Dependencies	Resolution Date
P2	S4	I-100	2	1	2023-02-15	None	2023-03-05
P2	S4	I-101	1	3	2023-03-01	I-100, I-104	2023-03-15
P2	S4	I-102	2	2	2023-03-01	None	2023-03-10
P2	S4	I-104	2	3	2023-01-25	None	2023-03-01

4.2.1 Issue Rollover Between Sprints

To better simulate real-world scenarios, a percentage of issues from each sprint was rolled over into the subsequent sprint, reflecting cases where certain issues remain unresolved (e.g., due to time constraints, technical challenges, or other blockers). For this study, the last 10% of issues (i.e., those resolved last within a sprint) were carried over into the following sprint.

Table 4.2 provides an overview of the number of issues per sprint in each project before rollover, while Table 4.3 shows the corresponding numbers after rollover.

4.2.2 Constraint Inputs

The model was provided with several input constraints derived from the dataset. The constraints are categorized into *retractable* and *non-retractable* based on whether their values can be adjusted during the optimization.

Retractable Constraints The solver can violate these constraints to optimize the objective function:

- **Priority Class (p):** Lower numeric values are given precedence over higher numeric values.
- **Type Class (t):** Lower numeric values are given precedence over higher numeric values.
- **Creation Date (c):** Older issues are given precedence over newer issues. The date component of the creation timestamp was used for this purpose.

Each retractable constraint is assigned a weight representing the cost incurred if the constraint is violated during the optimization process. For example, a configuration labeled `p1c1` indicates that both *Priority* and *Creation Date* constraints are weighted at 1. These weights determine how strongly the solver penalizes violations of each constraint when generating solutions.

Non-Retractable Constraints These constraints must be strictly enforced and cannot be changed during optimization:

- **Position Uniqueness:** Each issue is assigned a unique position in the ordering sequence. This ensures that no two issues share the same position, thereby enforcing a strict, conflict-free prioritization order.
- **Dependencies (Linked Issues):** Each issue must have all dependencies resolved prior to its own resolution.
- **Solution Uniqueness:[†]** This ensures that the current solution differs from a reference solution in at least one issue position, preventing the solver from returning identical orderings repeatedly.

- **Pairwise Elicitations:**[†] This simulates user input during solver optimization to refine the ordering preferences of the elicited pair.

4.2.3 Configurations

To systematically evaluate the impact of various factors on issue prioritization, we designed a set of simulation configurations that vary key parameters. These configurations form the basis for analyzing the algorithm’s performance across multiple sprints and projects. The specific configurations used for each simulation are listed in Table 4.7.

The baseline configuration, referred to as **c1** or simply **b**, is defined as the use of creation date constraints with no elicitations and no errors. This configuration serves as the primary basis for evaluating the robustness of the results in *RQ3*. Non-retractable constraints, such as dependency constraints, are enforced across all configurations, including the baseline, as they represent fixed structural requirements that must be met.

4.3 Solving Constraints

Algorithm 1 outlines the procedure for prioritizing issues within a sprint and project, incorporating constraints and iterative user elicitation to refine the ordering. The algorithm takes as input a set of issues I for a given sprint and project, along with associated constraints derived from issue metadata and dependencies.

To illustrate, consider the simplified input data for Sprint 4 of Project 2 (P2-S4)

^{4†} Constraint added during solver optimization.

Table 4.7: Simulation configurations showing varied and fixed values across research questions

Sim#	RQ#	Varying Values	Fixed Values
1	RQ1	Elicitations (0, 25, 50, 100)	constraint=c1p1, error=0
2	RQ2.1	Constraints (c1, c1p1, c1t1, c1p1t1)	eli=50, error=0
3	RQ2.2	Constraint Weights (c1p1t1, c2p1t1, c1p2t1, c1p1t2)	eli=50, error=0
4	RQ3	User error rate (b, 0, 10, 20)	eli=50, constraint=c1p1
5	RQ4	Local Constraints (clp1, clt1, clp1t1)	eli=50, error=0
6	RQ5	Unresolved Issues (0, 25, 50, 100)	constraint=c1p1, error=0
7	RQ6	State-of-the-Art Comparisons (CP-SAT, ACO, MOPSO)	constraint=c1p1, error=0

shown in Table 4.6. The gold standard ordering, based on resolution dates, was:

$$I-104 \prec I-100 \prec I-102 \prec I-101.$$

4.3.1 Constraint Initialization

(Lines 1-7): The algorithm extracts the necessary parameters from the input data: *priority class*, *type class*, *creation date*, and *dependencies*, and then constructs the constraints. Non-retractable constraints are enforced to maintain strict orderings, i.e., unique issue positions and dependency partial orders (Lines 4-5). Additionally, the initial set of retractable constraints, derived from creation dates, priority class, and type class, is added (Line 7). $\langle I-1, I-2 \rangle$ indicates that $I-1$ must precede $I-2$. Table 4.8 provides an example using the `c1p1t1` constraints that would be extracted for P2-S4.

Table 4.8: Constructed constraints from sample data in Table 4.6, with associated weights and retractability for P2-S4

Constraint Types	Constraint Pairs	Weights	Non-Retractable
Unique Positions	All issues must occupy distinct positions	–	✓
Dependencies	$\langle I-100, I-101 \rangle$; $\langle I-104, I-101 \rangle$	–	✓
Priority Class	$\langle I-101, I-100 \rangle$; $\langle I-101, I-102 \rangle$; $\langle I-101, I-104 \rangle$	1	
Type Class	$\langle I-100, I-102 \rangle$; $\langle I-100, I-101 \rangle$; $\langle I-100, I-104 \rangle$; $\langle I-102, I-101 \rangle$; $\langle I-102, I-104 \rangle$	1	
Creation Date	$\langle I-104, I-100 \rangle$; $\langle I-104, I-101 \rangle$; $\langle I-104, I-102 \rangle$; $\langle I-100, I-101 \rangle$; $\langle I-100, I-102 \rangle$	1	

Table 4.9: Sample parameters for the optimization process for P2-S4

Parameter	Value	Description
MAX_ELI	2	Maximum number of elicitations
POP_SIZE	5	Size of the population maintained during optimization
ERROR_RATE	0	The user error rate threshold

4.3.2 Parameter Initialization

(Lines 8-10): The parameters specific to the problem are initialized. Table 4.9 shows the parameter values used in this phase.

4.3.3 Initial Solution Generation

(Lines 16-23): Using the CP-SAT solver, an initial population of candidate orderings, S is generated. These solutions satisfy all non-retractable constraints and attempt to satisfy the retractable constraints. When elicitation is involved, the initial population size is limited to 2. This smaller initial set ensures that candidate solutions are available to generate informative pairs for elicitation. Table 4.10 provides an example of data that can be included in the original set of solutions before elicitation.

Table 4.10: Sample candidate solutions for P2-S4 and the corresponding cost breakdown

Final Ordering	Priority Cost	Type Cost	Creation DateCost	Overall Cost
I-100, I-104, I-102, I-101	3	1	1	5
I-104, I-100, I-101, I-102	2	3	0	5

4.3.4 Iterative Elicitation and Refinement

(Lines 24-34): While multiple candidate orderings remain and the maximum number of elicited pairs is not exceeded, the algorithm simulates user input by eliciting pairwise preferences that distinguish between candidate solutions. Each elicitation adds new constraints, pruning the solution space and updating the population P_n . Based on the candidate solutions in Table 4.10, the disagreement pairs between the two solutions are: $\langle I-100, I-104 \rangle$ and $\langle I-102, I-101 \rangle$. Since the error rate is 0, we do not simulate user error; therefore, these pairs will be added as non-retractable constraints, where the elicited pair is based on the order in which they appear in the gold standard. This is shown in Table 4.11. Once maximum elicitation is reached, Line 33 allows for re-searching the solution space with the new elicitation constraints. Since these are hard constraints, the solver cannot break them. They must always be satisfied. The new set of candidate solutions is shown in Table 4.12. The last two solutions were added to the original candidate solutions, yielding an overall population of 4 candidate solutions at the end of the process.

If the population size were less than 4, then Lines 28-29 of the algorithm would update P_n to include the two original solutions and only one option from the newly generated solutions. Since the cost of the solutions increased after elicitation, there is a risk that these solutions may not be included in the search space, depending on the population size and the cost of existing solutions. This cost increase aligns with findings that `c1p1t1` is not the best-performing constraint configuration, even though

Table 4.11: Sample elicitation log with disagreement pairs, elicited pairs, and counts for P2-S4. *The highlighted row indicates that we have reached the maximum elicitation.*

Disagreement Pair	Elicited Pair	Elicitation Count
$\langle I-100, I-104 \rangle$	$\langle I-104, I-100 \rangle$	1
$\langle I-102, I-101 \rangle$	$\langle I-102, I-101 \rangle$	2

it performs reasonably well overall [43]. In fact, the final candidate solution showed increases in both metrics, as eliciting only two pairs did not sufficiently constrain the search space to align with the gold standard. This highlights the importance of the chosen constraint configurations, the maximum elicited pairs, and the population size, as they all affect the final outcome, similarly to how multiple objectives are considered in MOPSO [34]. Once convergence is reached or the maximum elicitation budget is spent, the algorithm computes fitness metrics comparing candidate orderings to the gold standard ordering, using disagreement counts and average distance measures. At this point, Line 35 calculates the fitness (disagreement and average distance) and returns the final solutions as shown in Table 4.12.

Table 4.12: Sample final population of solutions for P2-S4 with cost breakdown, disagreement count, and average distance from gold standard ordering

Final Ordering	Priority Cost	Type Cost	Creation DateCost	Overall Cost	DIS	AD
I-100, I-104, I-102, I-101	3	1	1	5	1	0.5
I-104, I-100, I-101, I-102	2	3	0	5	1	0.5
I-104, I-102, I-100, I-101	3	3	1	7	1	0.5
I-102, I-104, I-100, I-101	3	2	2	7	2	1

Note that this algorithm is repeated 15 times for each configuration, as outlined in Table 4.7, to obtain the final set of median disagreements and average distances.

4.4 Final Ordering

The final output is the population that best satisfies the given constraints. However, it is important to note that satisfying all constraints does not necessarily result in minimal disagreement. In other words, achieving a cost of 0 does not guarantee a disagreement score of 0. We evaluated the generated orderings against the gold standard using disagreement and average distance. Table 4.12 provides an example of the final output population along with its evaluation metrics.

For each research question, we used boxplots to visualize the distribution of median disagreements and median average distances across all projects. Boxplots were generated for each simulation configuration 4.7 and its corresponding research question, showing how these metrics varied across runs at the project level. This analysis provided a clear, project-level view of how these metrics varied across all simulated conditions for each research question.

Following the guidelines for choosing appropriate statistical tests outlined by Chicco et al. [14], we selected nonparametric methods whenever the assumption of normality was violated and used post-hoc corrections to control for multiple comparisons.

To ensure statistically sound comparisons across the different simulation configurations, we first assessed the distribution of metric values within each group using the Shapiro–Wilk test. Since the assumption of normality was violated for all data sets, we proceeded with nonparametric statistical tests for all research questions.

For research questions involving three or more configurations (RQ1, RQ2, RQ3, RQ4, and RQ6), we applied the Kruskal–Wallis H test to assess the overall statistical significance of performance metric differences. Following a significant Kruskal–Wallis result, we then performed pairwise Mann–Whitney U tests as a post-hoc analysis to identify specific differences between configurations. The resulting p -values were

adjusted using the Holm correction to control the Family-Wise Error Rate (FWER). Conversely, for research questions requiring only pairwise comparisons (RQ5), we directly applied the Mann–Whitney U test to compare the two specific configurations. These p -values were also subjected to the Holm correction to maintain the Family-Wise Error Rate.

Algorithm 1 Issue prioritization algorithm for a single sprint

```
1: IN:
2:    $I$ : set of issues for a given sprint and project
3: NON-RETRACTABLE CONSTRAINTS:
4:    $\forall i \in I : 0 \leq \text{pos}(i) < |I|, \forall (i \neq j) : \text{pos}(i) \neq \text{pos}(j)$ 
5:    $\{\text{ord}_1, \dots, \text{ord}_k\} \subseteq I \times I$  {partial orders from D}
6: RETRACTABLE CONSTRAINTS:
7:    $\{\text{ord}_1, \dots, \text{ord}_k\} \subseteq I \times I$  {partial orders from C, P, T}
8: MAX_ELI  $\leftarrow$  max number of elicited pairs
9: POP_SIZE  $\leftarrow$  max number of retained solutions
10: ERROR_RATE  $\leftarrow$  simulated user elicitation error rate
11: GOLD_STD  $\leftarrow$  real-world resolution order
12: BASELINE  $\leftarrow$  naive ordering by issue creation timestamp
13: OUT:
14:    $P_n =$  population of  $m$  ordered issue sets, where each  $i_k^{(j)} \in I$ 
15: PROCEDURE:
16: if  $\text{sprint} \neq \text{first\_sprint}$  then
17:    $\text{rollover} \leftarrow$  RolloverIssues()
18: end if
19:  $\text{params} \leftarrow$  ExtractParams( $I$ )
20:  $\text{constraints} \leftarrow$  BuildConstraintModel( $\text{params}$ )
21:  $\text{eli\_pairs} \leftarrow \emptyset$  {elicited  $(i, j)$  where  $i \prec j$  in GOLD_STD}
22:  $\text{err\_pairs} \leftarrow \emptyset$  {reverse errors}
23:  $S \leftarrow$  CPSATSolve( $\text{constraints}$ )
24: while  $|S| > 1$  and  $|\text{eli\_pairs}| < \text{MAX\_ELI}$  do
25:    $(\text{eli\_new}, \text{err\_new}) \leftarrow$  Elicit( $S, \text{ERROR\_RATE}$ )
26:    $\text{eli\_pairs} \leftarrow \text{eli\_pairs} \cup \text{eli\_new}$ 
27:    $\text{err\_pairs} \leftarrow \text{err\_pairs} \cup \text{err\_new}$ 
28:   if  $|P_n| \geq \text{POP\_SIZE}$  and  $\min(\text{Cost}(S)) \leq \max(\text{Cost}(P_n))$  then
29:      $P_n \leftarrow S \cup P_n$ 
30:   else if  $|P_n| < \text{POP\_SIZE}$  then
31:      $P_n \leftarrow S \cup P_n$ 
32:   end if
33:    $S \leftarrow$  CPSATSolve( $\text{constraints} \cup \text{eli\_pairs}$ )
34: end while
35:  $\text{fitness} \leftarrow$  ComputeFitness( $P_n, \text{GOLD\_STD}$ )
36: return  $P_n$ 
```

Chapter 5

Results

This section presents the results obtained by running the simulation configurations outlined in Table 4.7 and addresses the six research questions.

Table 5.1: Comparison of CP-SAT, ACO, and MOPSO median disagreement and average distances for Project 4. *Minimum values per sprint/project are bolded.*

Metric/ Sprint	CP-SAT: Elicitations				CP-SAT: Error Rate			ACO	MOPSO
	0	25	50	100	0%	10%	20%	Median	Median
Disagreement									
s1	1197.5	1156.0	1124.5	1117.0	1124.5	1125.5	1148.5	–	–
s2	1182.0	1159.5	1073.5	1052.5	1073.5	1072.0	1094.5	–	–
s3	1150.5	1133.5	1087.0	1096.0	1087.0	1170.5	1094.0	–	–
s4	1029.0	942.0	896.5	900.0	896.5	821.5	901.5	–	–
s5	431.0	408.0	398.5	377.0	398.5	433.5	415.0	–	–
Project Median	1150.5	1133.5	1073.5	1052.5	1073.5	1072.0	1094.0	903.0	913.0
Average Distance									
s1	22.14	21.79	21.19	21.10	21.19	21.21	21.53	–	–
s2	23.81	23.39	21.84	21.37	21.84	21.85	22.00	–	–
s3	27.30	27.17	26.67	26.65	26.67	27.47	26.73	–	–
s4	24.61	23.25	22.86	22.82	22.86	19.68	22.56	–	–
s5	10.94	10.54	10.26	10.07	10.26	10.69	10.57	–	–
Project Median	23.81	23.25	21.84	21.37	21.84	21.21	22.00	20.42	20.63

Table 5.2: Comparison of CP-SAT, ACO, and MOPSO median disagreement and average distances for Project 12. *Minimum values per sprint/project are bolded.*

Metric/ Sprint	CP-SAT: Elicitations				CP-SAT: Error Rate			ACO	MOPSO
	0	25	50	100	0%	10%	20%	Median	Median
Disagreement									
s1	1199.5	1173	1157	1159.5	1157	1159	1206.5	–	–
s2	468.5	420.5	384	367	384	417	480.5	–	–
s3	632	612.5	613.5	612	613.5	629.5	661	–	–
s4	503	469	469	469.5	469	484.5	545	–	–
s5	429.5	412	413.5	414	413.5	419.5	443	–	–
Project Median	503	469	469	469.5	469	484.5	545	645	648
Average Distance									
s1	20.66	20.16	20.07	20.04	20.07	19.95	20.59	–	–
s2	10.71	9.58	9	8.55	9	9.69	11.55	–	–
s3	17.13	16.85	16.81	16.90	16.81	17.12	18.04	–	–
s4	13.92	12.98	13	13.04	13	13.38	15.04	–	–
s5	14.31	13.87	13.89	13.91	13.89	13.93	14.49	–	–
Project Median	14.31	13.87	13.89	13.91	13.89	13.93	15.04	17.29	17.30

Tables 5.1, 5.2, 5.3, and 5.4 present a high-level comparison of CP-SAT, ACO, and MOPSO for Projects 4, 12, 13, and 25, respectively, across all sprints, highlighting both disagreement and average distance metrics. ACO and MOPSO were selected as comparison methods because they are commonly used in studies evaluating state-of-the-art prioritization approaches [1, 18], providing a natural baseline for assessing constraint-based optimization. Although exact experimental conditions cannot be perfectly matched across these methods, we ensured the comparison was as fair as possible by using the same number of average runs and population size for ACO and MOPSO as for CP-SAT.

Overall, CP-SAT generally outperformed both ACO and MOPSO, with the notable exception of Project 4 (Table 5.1). Alignment with the gold standard typically improved as more user input was elicited; however, diminishing returns were occasionally

Table 5.3: Comparison of CP-SAT, ACO, and MOPSO median disagreement and average distances for Project 13. *Minimum values per sprint/project are bolded.*

Metric/ Sprint	CP-SAT: Elicitations				CP-SAT: Error Rate			ACO	MOPSO
	0	25	50	100	0%	10%	20%	Median	Median
Disagreement									
s1	311	289	285	287	285	290.5	313.5	–	–
s2	559	550.5	550	550	550	562	550	–	–
s3	170.5	137.5	136	136	136	161	171.5	–	–
s4	274	234	248.5	246	248.5	241.5	281.5	–	–
s5	154.5	139.5	141.5	142.5	141.5	145.5	146.5	–	–
Project Median	274	234	248.5	246	248.5	241.5	281.5	368	327
Average Distance									
s1	10.05	9.67	9.63	9.60	9.63	9.63	10.37	–	–
s2	15.33	15.33	15.29	15.33	15.29	15.42	15.35	–	–
s3	6.19	5.35	5.30	5.30	5.30	5.89	6.54	–	–
s4	10.74	9.60	10.29	10.20	10.29	9.71	11.31	–	–
s5	6.44	5.94	6.03	6.03	6.03	6.15	6.06	–	–
Project Median	10.05	9.60	9.63	9.60	9.63	9.63	10.37	13.18	12.49

observed. For example, in Sprint 3 of Project 25 (Table 5.4), disagreement increased slightly from 50 to 100 elicitations while average distance remained unchanged. Additionally, increasing the simulated error rate generally led to worse performance, highlighting the importance of accurate user input in the prioritization process.

5.1 Role of Interaction (RQ1)

RQ1 investigated the impact of varying user elicitation on the alignment of prioritization order with the gold standard under constraint `c1p1` and 0% error rate.

Table 5.4: Comparison of CP-SAT, ACO, and MOPSO median disagreement and average distances for Project 25. *Minimum values per sprint/project are bolded.*

Metric/ Sprint	CP-SAT: Elicitations				CP-SAT: Error Rate			ACO	MOPSO
	0	25	50	100	0%	10%	20%	Median	Median
Disagreement									
s1	321	295.5	290.5	290.5	290.5	300.5	307	–	–
s2	349	340	340.5	343.5	340.5	350	389.5	–	–
s3	199	194	193.5	194	193.5	205.5	204.5	–	–
s4	225.5	206.5	207.5	209	207.5	227.5	245	–	–
s5	226.5	198	205	199.5	205	219.5	246	–	–
Project Median	226.5	206.5	207.5	209	207.5	227.5	246	234	236
Average Distance									
s1	11.68	10.92	10.65	10.70	10.65	11.03	11.35	–	–
s2	12.58	12.56	12.56	12.61	12.56	12.69	14.31	–	–
s3	9.60	9.40	9.37	9.37	9.37	9.70	10	–	–
s4	10.13	9.73	9.70	9.70	9.70	10.23	10.70	–	–
s5	10.41	9.38	9.72	9.52	9.72	10.34	11.17	–	–
Project Median	10.41	9.73	9.72	9.70	9.72	10.20	11.17	10.76	10.53

5.1.1 Project-Level Analysis

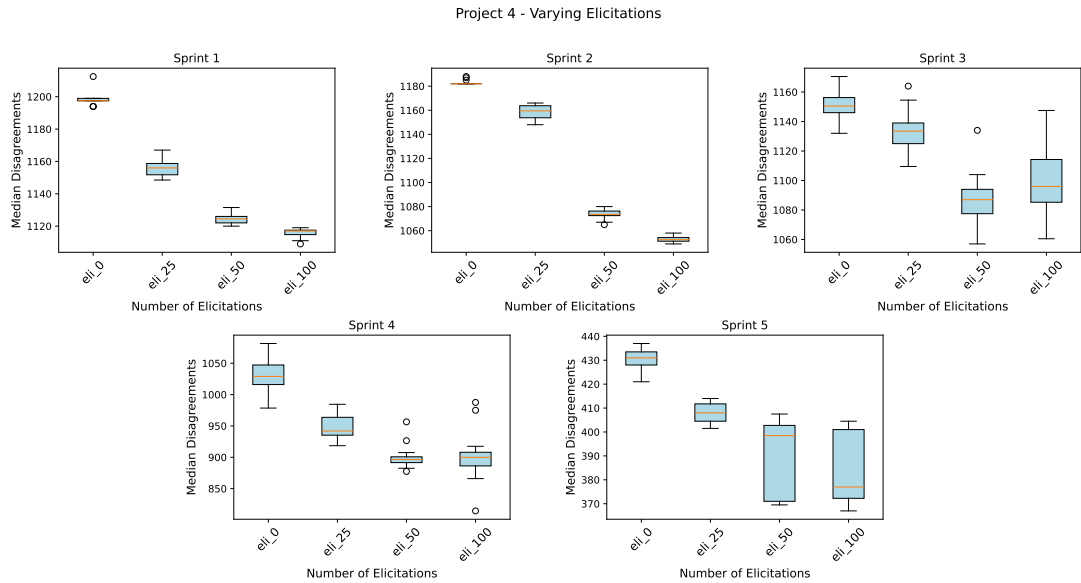
Performance generally improved in terms of both disagreement and average distance as additional user input was incorporated, as illustrated in Figure 5.1. This trend is particularly evident in the median values of the boxplots.

As shown in Table 5.1, there is a consistent decrease in both median disagreement and average distance with increasing elicitation levels, a pattern further supported by the statistics reported in Table 5.5.

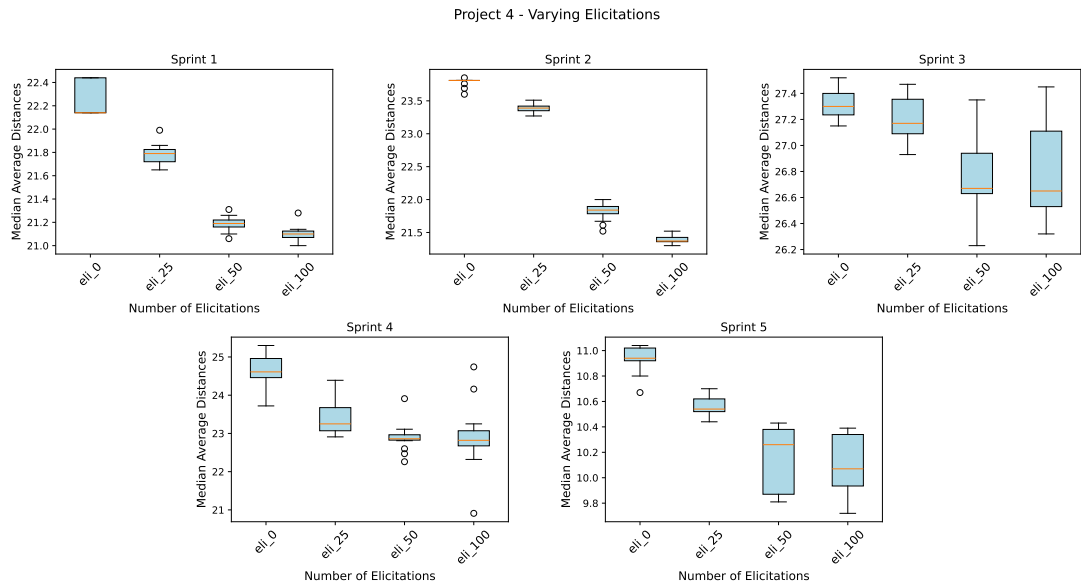
However, the difference in performance between 50 and 100 elicitations was the only pair that was not statistically significant for Project 4, suggesting that gains may begin to plateau after approximately 50 elicitations in Project 4. Notably, for disagreement, the comparisons between 0 vs. 50 and 0 vs. 100 elicited pairs yielded essentially identical, extremely small p -values, indicating that 50 elicited pairs are

Figure 5.1: Median Disagreement (a) and Median Average Distance (b) across varying elicitations for P4

(a) Median Disagreement



(b) Median Average Distance



likely sufficient to achieve highly significant improvements. Furthermore, for average distance, the 0 vs. 50 comparison exhibited even stronger statistical significance

Table 5.5: Statistical tests for Project 4 showing significance of varying elicitations. *Bold p-values indicate statistical significance ($p < 0.05$).*

Metric	Comparison	Test	p-value
Disagreement	–	Kruskal-Wallis	<0.0001
	0 vs 25	Mann-Whitney U	0.0023
	0 vs 50	Mann-Whitney U	<0.0001
	0 vs 100	Mann-Whitney U	<0.0001
	25 vs 50	Mann-Whitney U	0.0001
	25 vs 100	Mann-Whitney U	0.0001
	50 vs 100	Mann-Whitney U	0.4963
Average Distance	–	Kruskal-Wallis	<0.0001
	0 vs 25	Mann-Whitney U	0.0262
	0 vs 50	Mann-Whitney U	0.0011
	0 vs 100	Mann-Whitney U	0.0014
	25 vs 50	Mann-Whitney U	0.0072
	25 vs 100	Mann-Whitney U	0.0025
	50 vs 100	Mann-Whitney U	0.3893

than 0 vs. 100. This suggests that a moderate elicitation level of 50 pairs is likely sufficient to capture the majority of the benefit, with additional input beyond this point providing diminishing returns.

For Project 12 (Table A.1), Project 13 (Table A.2), and Project 25 (Table A.3), statistical significance was observed only when each interactive method was compared to the non-interactive method for disagreement. This indicates that the differences between the interactive methods themselves did not result in significant improvements in outcomes. A similar trend was observed for average distance in Projects 13 and 25, while Project 12 showed no statistical significance for any comparison of elicitation levels in terms of average distance. Further analysis is needed to determine whether this lack of significance is due to the global constraint `c1p1` being misaligned or whether other factors are contributing to this outcome.

5.1.2 Sprint-Level Analysis

At the sprint level, the benefits of elicitation are generally evident, as illustrated in Figure 5.1. However, increasing elicitation from 50 to 100 pairs does not always result in improvement and in some cases even shows a slight worsening, which explains the lack of statistical significance noted in Table 5.5.

For example, Sprint 3 shows a minor increase in disagreement (0.83%) when moving from 50 to 100 elicitations, while average distance improves only marginally by 0.07%, as reported in Table 5.1. Additionally, at the sprint level, smaller sprints, such as Sprint 3, exhibit greater variability at higher elicitation levels, suggesting that the solver sometimes struggles to reconcile conflicting constraints with stakeholder preferences.

This suggests that the specific constraint configuration `c1p1` does not fully align with Project 4's historical resolution order, highlighting the role constraints play in achieving an accurate ordering. Despite this, median disagreements generally improve, indicating that the solver can overcome misaligned constraints when sufficient elicitation is provided, though the likelihood of higher-disagreement solutions increases.

Overall, the results confirm that increasing elicitation generally improves performance and that eliciting around 50 pairs at the sprint level is likely sufficient to capture most of the benefit, with additional input providing only marginal gains.

5.2 Role of Constraints (RQ2)

RQ2 investigated the impact of varying constraint configurations on the alignment of prioritization order with the gold standard. This research question was further

divided into two sub-questions:

- **RQ2.1 (Varying Constraints):** To what extent does adding or removing certain constraints influence the alignment of the final ordering with the gold standard?
- **RQ2.2 (Varying Weights):** How does adjusting constraint weights affect the alignment of the final ordering with the gold standard?

Although these sub-questions focus on different aspects of constraints, the results for both were analyzed together and are presented jointly in Figure 5.2 and Table 5.6. This approach provides a holistic view of how both the presence and weighting of constraints interact to influence alignment with the gold standard.

Table 5.6: Statistical tests for Project 12 showing significance of varying constraints. *Bold p-values indicate statistical significance ($p < 0.05$).*

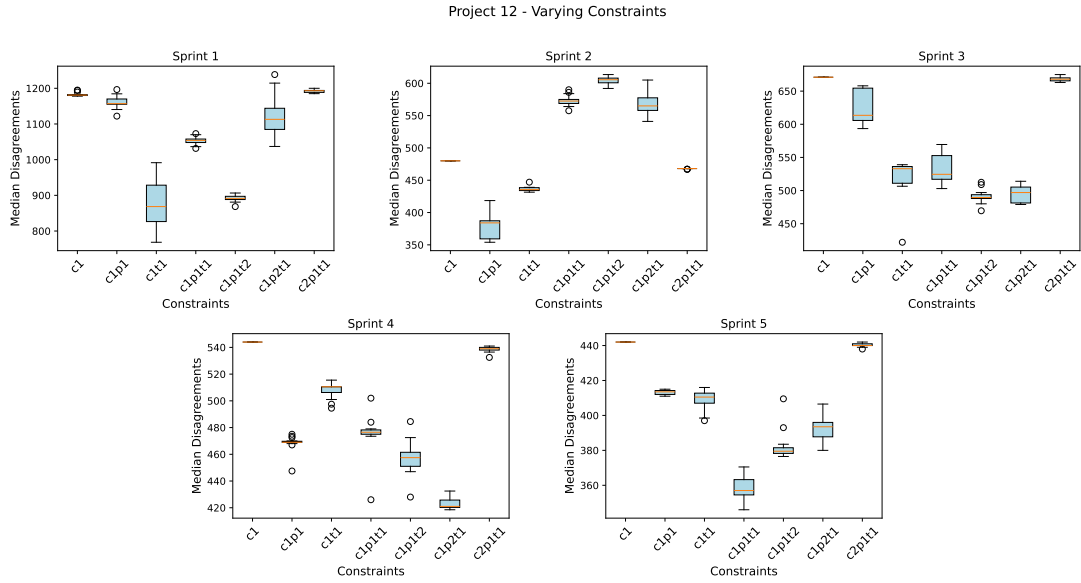
Metric	Comparison	Test	p-value
Disagreement	–	Kruskal-Wallis	0.0005
	c1 vs c1p1	Mann-Whitney U	0.0044
	c1 vs c1t1	Mann-Whitney U	0.0023
	c1 vs c1p1t1	Mann-Whitney U	1.0000
	c1 vs c1p1t2	Mann-Whitney U	0.4154
	c1 vs c1p2t1	Mann-Whitney U	0.0376
	c1 vs c2p1t1	Mann-Whitney U	1.0000
Average Distance	–	Kruskal-Wallis	< 0.0001
	c1 vs c1p1	Mann-Whitney U	0.0495
	c1 vs c1t1	Mann-Whitney U	< 0.0001
	c1 vs c1p1t1	Mann-Whitney U	0.0002
	c1 vs c1p1t2	Mann-Whitney U	< 0.0001
	c1 vs c1p2t1	Mann-Whitney U	0.0005
	c1 vs c2p1t1	Mann-Whitney U	1.0000

5.2.1 Project-Level Analysis

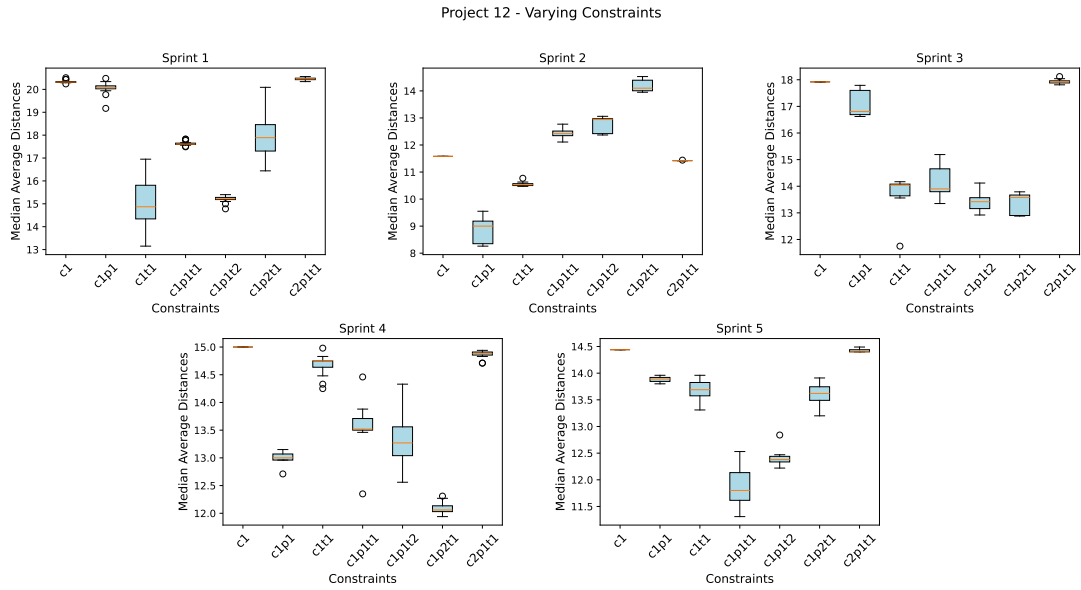
Overall, no single constraint configuration consistently outperformed the others across all sprints for Project 12, as shown in Figure 5.2. While most constraint configurations

Figure 5.2: Median Disagreement (a) and Median Average Distance (b) across varying constraint configurations for P12

(a) Median Disagreement



(b) Median Average Distance



outperformed the baseline, **c2p1t1**, which weights the creation date more heavily than the other two constraints, performed roughly at the same level as the baseline,

showing no significant improvement over ordering by creation date. This is further supported by Table 5.6, where `c2p1t1` did not achieve statistical significance when compared to `c1`. Notably, it was the only configuration for Project 12 that failed to achieve significance in terms of average distance. This pattern was also observed for Project 4 (Table B.1), Project 13 (Table B.2), and Project 25 (Table B.3) where `c2p1t1` did not achieve statistical significance compared to `c1` for either metric.

Regarding disagreement, most configurations involving all three constraints did not achieve statistical significance compared to `c1`, except for `c1p2t1`, which was the best-performing configuration for Sprint 4 across both metrics. Although this configuration achieved significance relative to `c1`, it had the highest p -value among the significant configurations, suggesting that, for Project 12, pairwise constraint combinations may better approximate real-world prioritization than using all three constraints simultaneously.

The constraint `c1t1` was the most statistically significant when compared to `c1` for Project 12, achieving p -values of 0.0023 for disagreement and ≈ 0.0000 for average distance. Importantly, the globally chosen best configuration, `c1p1`, also achieved statistical significance compared to `c1` in both metrics, though its p -values were not the lowest. This was similarly observed for Project 25 (Table B.3), where `c1p1` was the only configuration to achieve statistical significance compared to `c1` in terms of disagreement. However, `c1p1` did not achieve significance relative to `c1` for Project 4 (Table B.1) or Project 13 (Table B.2). Project 13 did show significance for `c1p1t2`, which generally outperformed `c1`, although there were some sprints where it underperformed. Project 4 did not exhibit any single constraint configuration with consistent statistical significance across all sprints.

These findings highlight the importance of selecting constraint configurations based on historical project data rather than assuming a global configuration will perform

optimally. Despite being suboptimal for Project 12, `c1p1` still produced disagreement and average distance values that outperformed both ACO and MOPSO, as shown in Table 5.2. However, the variation in results suggests that exploring additional constraint pairings may be necessary to identify the ideal configuration for each project.

5.2.2 Sprint-Level Analysis

Constraint performance varied across sprints; however, the trends observed for both disagreement and average distance generally aligned. For instance, in Sprint 1, `c1t1` had the lowest median disagreement, with a similar pattern observed for average distance. This indicates that constraint configurations similarly influence the solution space to align with the gold standard, both in ordering (disagreement) and in positioning (average distance). While no single constraint emerged as a definitive best, `c1t1` performed consistently well, whereas `c2p1t1` consistently underperformed, as supported by the statistical analysis in Table 5.6.

The absence of a clear winner may stem from limitations in our experimental setup, which may not fully capture real-world prioritization patterns, or from the need to explore a broader set of constraints, as most projects did not exhibit a universally optimal configuration. Nevertheless, as shown in Table 5.2, even under suboptimal constraint choices and experimental conditions, CP-SAT consistently outperformed the metaheuristic methods ACO and MOPSO for Project 12, showcasing the effectiveness of constraint-based optimization.

5.3 Robustness (RQ3)

This research question investigated the effect of varying user error rates on disagreement and average distance when eliciting up to 50 pairs using the c1p1 constraints.

Table 5.7: Statistical tests for Project 13 showing significance of varying error rates. *Bold p-values indicate statistical significance ($p < 0.05$).*

Metric	Comparison	Test	p-value
Disagreement	–	Kruskal-Wallis	0.0135
	b vs err_0	Mann-Whitney U	0.0882
	b vs err_10	Mann-Whitney U	0.4718
	b vs err_20	Mann-Whitney U	0.6730
	err_0 vs err_10	Mann-Whitney U	0.1153
	err_0 vs err_20	Mann-Whitney U	0.0372
	err_10 vs err_20	Mann-Whitney U	0.2954
Average Distance	–	Kruskal-Wallis	0.0103
	b vs err_0	Mann-Whitney U	0.0124
	b vs err_10	Mann-Whitney U	0.2658
	b vs err_20	Mann-Whitney U	0.2408
	err_0 vs err_10	Mann-Whitney U	0.2460
	err_0 vs err_20	Mann-Whitney U	0.2518
	err_10 vs err_20	Mann-Whitney U	0.2795

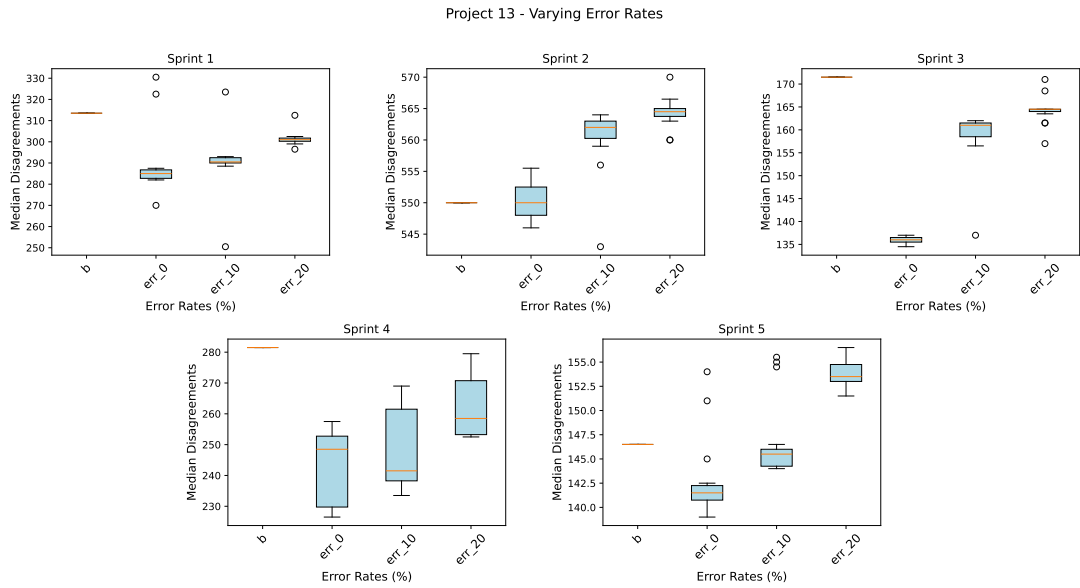
5.3.1 Project-Level Analysis

As shown in Figure 5.3, both median disagreement and median average distance generally worsen as the error rate increases for Project 13, which aligns with expectations. This trend is also evident in Table 5.3, where higher error rates consistently correspond to degraded performance.

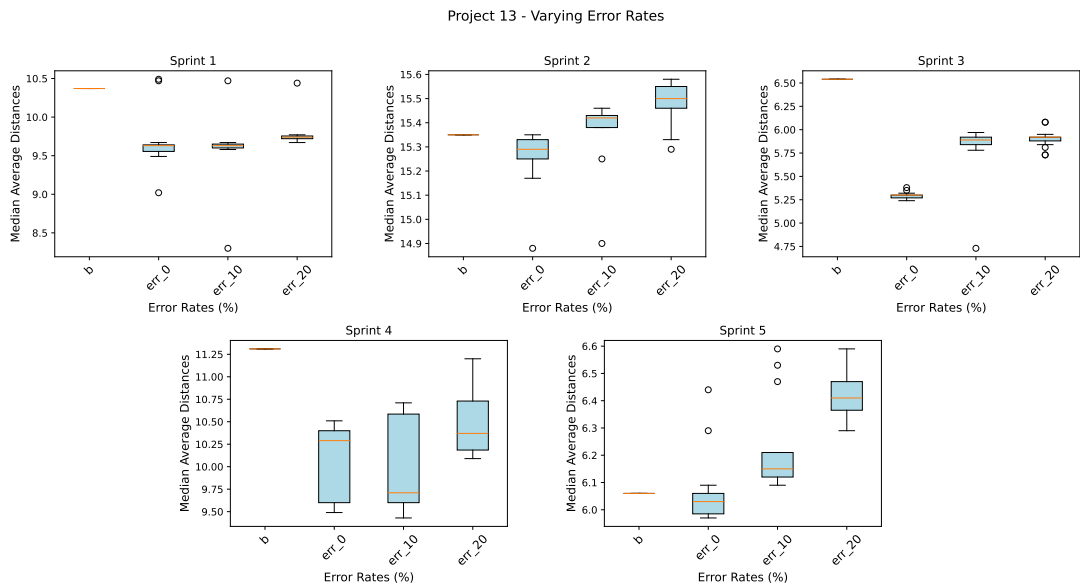
Statistical tests in Table 5.7 show that, for disagreement, the increase from 0% to 20% error resulted in a statistically significant worsening ($p = 0.0372$), indicating that higher error rates significantly misalign the ordering relative to the gold standard for Project 13. Other pairwise comparisons for disagreement did not achieve significance, suggesting that moderate error rates (up to 10%) have a limited impact on solution

Figure 5.3: Median Disagreement (a) and Median Average Distance (b) showing varying error rates for P13

(a) Median Disagreement



(b) Median Average Distance



quality.

For average distance, only the comparison between the baseline (creation date with

0 elicitations and 0% error) and 0% error with 50 elicitations reached statistical significance ($p = 0.0124$), highlighting the improvement in average distance when our method is applied compared to the naive baseline of ordering by creation date. This trend was also observed for Project 12 (Table C.2) and Project 25 (Table C.3), but notably not for Project 4 (Table C.1), likely indicating that the chosen constraints did not sufficiently align with Project 4 to improve performance relative to the baseline. While not statistically significant, the low p-value of 0.0882 for disagreement between the baseline and 0% error with 50 elicitations for Project 13 further supports this improvement.

Overall, these results highlight the robustness of the CP-SAT approach. The system generally maintains strong ordering and positional alignment even in the presence of moderate input errors and consistently outperforms the baseline configuration. Notably, at a 20% error rate, CP-SAT still outperforms both ACO and MOPSO, as shown in Table 5.3, further emphasizing its resilience. This trend is also observed for Project 12, as seen in Table 5.2. However, Project 4 (Table 5.1) did not outperform ACO or MOPSO even at 0% error, while Project 25 (Table 5.4) outperformed ACO and MOPSO only up to a 10% error rate. These differences could be attributed to the constraints employed for each project or to the number of issues being prioritized.

5.3.2 Sprint-Level Analysis

At the sprint level, the trends become more apparent, with both disagreement and average distance generally worsening as more errors are introduced. However, the magnitude of these degradations is relatively small. For instance, the difference between the median disagreement at 10% and 20% error rate for Sprint 5 is only 0.68%. The statistical analysis in Table 5.7 further emphasizes the lack of significance in performance differences across error rates. This indicates that the system can

generally tolerate imperfect environments while still producing results that outperform ACO and MOPSO in both disagreement and average distance, as shown in Table 5.3.

Based on Figure 5.3, error rates of up to 20% generally continue to outperform the baseline, as seen in Sprints 1, 3, and 4 for both metrics. However, there are exceptions. In Sprint 2, for example, the 0% error condition achieves performance comparable to the baseline, whereas the 10% error condition performs worse than the baseline. Similarly, in Sprint 5, a 20% error rate results in performance below the baseline. These observations highlight that, while the system is generally robust, misalignment between constraint configurations and project- or sprint-level prioritization can occasionally lead to degraded performance relative to naive creation-date ordering.

5.4 Role of Local Constraints (RQ4)

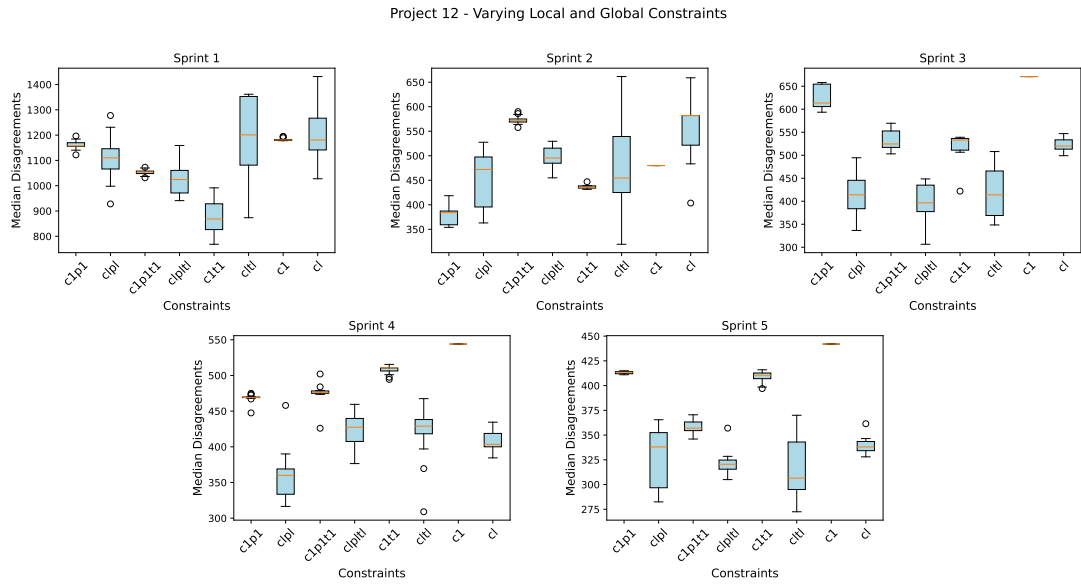
RQ4 investigated the effect of local constraints compared to global constraints on disagreement and average distance when eliciting up to 50 pairs with 0% error.

Table 5.8: Statistical tests for Project 12 showing significance of global constraints (c1) versus their local counterparts (cl). *Bold p-values indicate statistical significance ($p < 0.05$).*

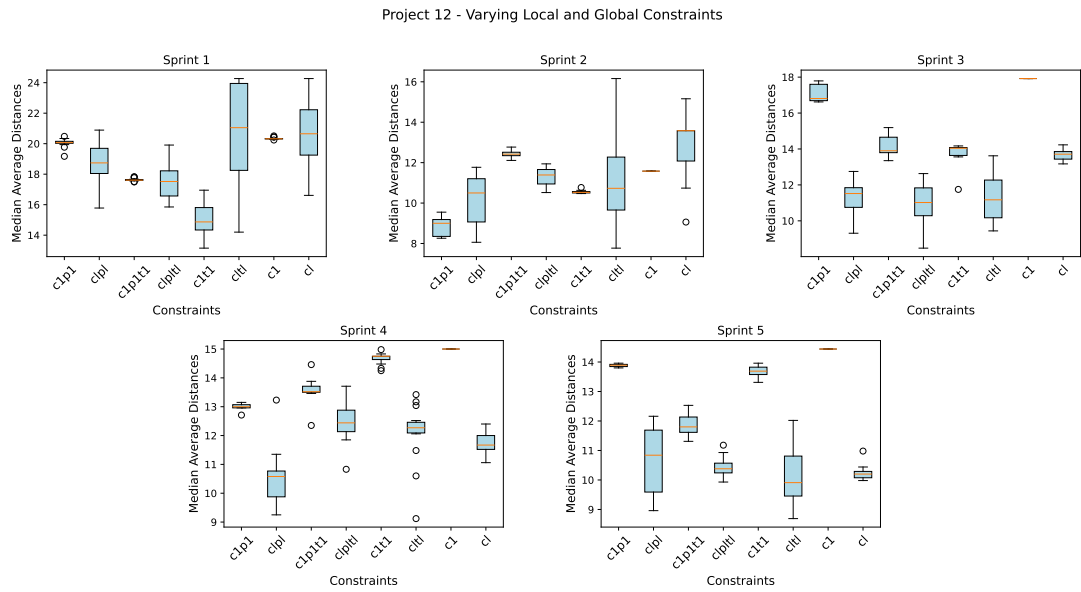
Metric	Comparison	Test	p-value
Disagreement	–	Kruskal-Wallis	< 0.0001
	c1p1 vs clp1	Mann-Whitney U	0.0050
	c1p1t1 vs clp1t1	Mann-Whitney U	0.0050
	c1t1 vs clt1	Mann-Whitney U	0.0836
	c1 vs cl	Mann-Whitney U	0.0462
Average Distance	–	Kruskal-Wallis	< 0.0001
	c1p1 vs clp1	Mann-Whitney U	0.0012
	c1p1t1 vs clp1t1	Mann-Whitney U	< 0.0001
	c1t1 vs clt1	Mann-Whitney U	0.0012
	c1 vs cl	Mann-Whitney U	0.0001

Figure 5.4: Median Disagreement (a) and Median Average Distance (b) comparing local and global constraint configurations for P12

(a) Median Disagreement



(b) Median Average Distance



5.4.1 Project-Level Analysis

Figure 5.4 illustrates the effect of applying constraints locally versus globally in Project 12. Local constraints generally outperformed global ones, a trend reinforced by Table 5.8, where most pairwise comparisons were statistically significant. Configurations `clp1t1` and `c1p1t1` showed the largest differences for both disagreement and average distance, indicating that using all three constraints amplifies the impact of local versus global application in the case of Project 12.

This trend largely holds across the other projects. In Projects 4 and 25 (Table D.1 and Table D.3), all comparisons were statistically significant, with local constraints generally performing better than global ones. Project 13 also generally showed better performance with local constraints, although none of the differences reached statistical significance. This could be attributed to the smaller number of issues in Project 13, as seen in Table 4.3.

However, since its size is similar to Project 25, which achieved statistical significance across all comparisons, the lack of significance in Project 13 may be due to other factors, such as the constraints not aligning in a way that makes the differences between global and local application pronounced. Nevertheless, the fact that the system still performed better when constraints were applied locally, even without statistical significance, highlights the strong impact of applying constraints at the assignee level.

5.4.2 Sprint-Level Analysis

At the sprint level, Figure 5.4 reinforces the project-level findings. In Sprints 3, 4, and 5, local constraints consistently outperformed their global counterparts. For instance, `clp1` achieved lower disagreement and average distance than `c1p1`, and

c1t1 performed better than c1t1.

There are exceptions in Sprints 1 and 2, where local constraints occasionally underperformed. For example, c1t1 exhibits higher median disagreement and average distance than c1t1 in Sprint 1. Additionally, the interquartile range (IQR) of c1t1 is larger than that of other configurations in most sprints, suggesting potential instability in its performance.

These patterns suggest that in larger sprints, the difference between local and global constraint performance may be smaller. In some cases, local constraints may even slightly reduce alignment, potentially because a greater number of issues per assignee dilutes the impact of assignee-specific constraints, making it harder for local adjustments to consistently improve overall prioritization.

5.5 Unresolved Issue Rollover (RQ5)

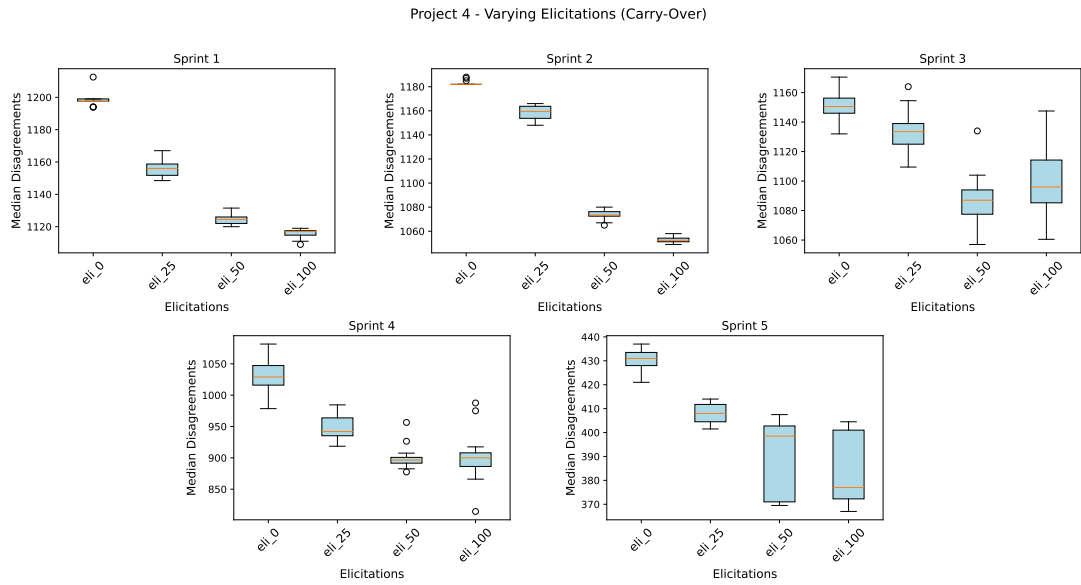
RQ5 investigated how effectively the system managed unresolved issues being carried over to subsequent sprints.

Table 5.9: Project 4 comparisons for carry-over vs. non-carry-over. *Bold p-values indicate statistical significance ($p < 0.05$).*

Metric	Comparison	Test	p-value
Disagreement	eli_0	Mann-Whitney U	< 0.0001
	eli_25	Mann-Whitney U	< 0.0001
	eli_50	Mann-Whitney U	< 0.0001
	eli_100	Mann-Whitney U	< 0.0001
Average Distance	eli_0	Mann-Whitney U	< 0.0001
	eli_25	Mann-Whitney U	< 0.0001
	eli_50	Mann-Whitney U	< 0.0001
	eli_100	Mann-Whitney U	< 0.0001

Figure 5.5: Median Disagreements for Carry-Over (a) and Non-Carry-Over (b) showing impact of issue rollover at varying elicitations for P4

(a) Carry-Over



(b) Non-Carry-Over

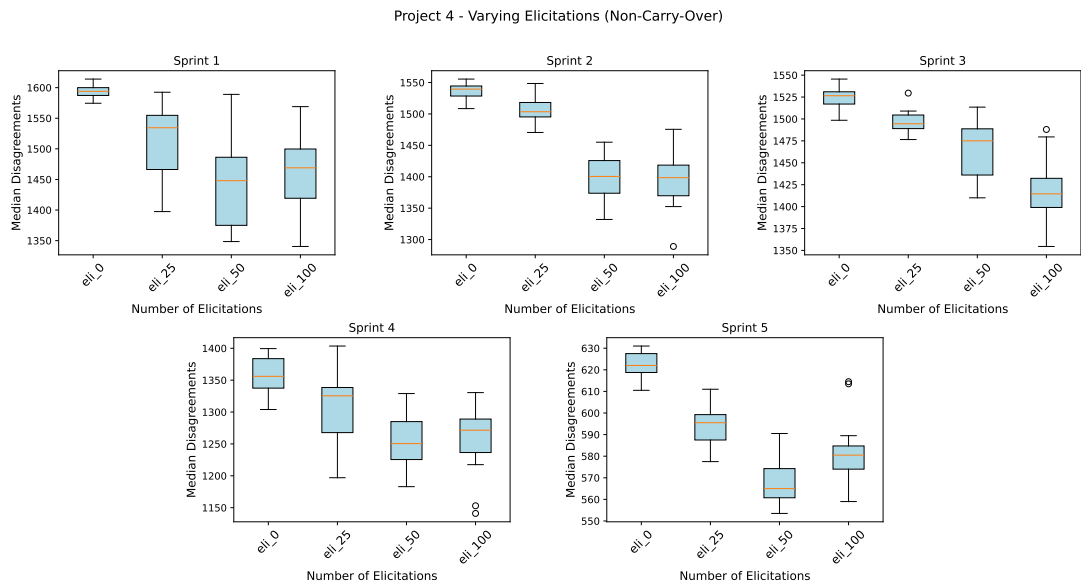
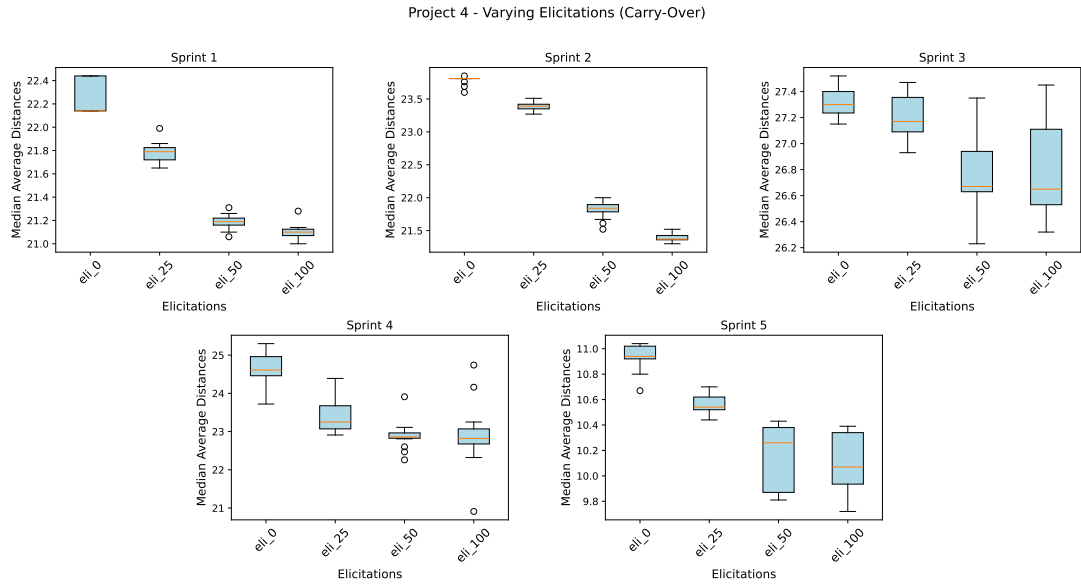
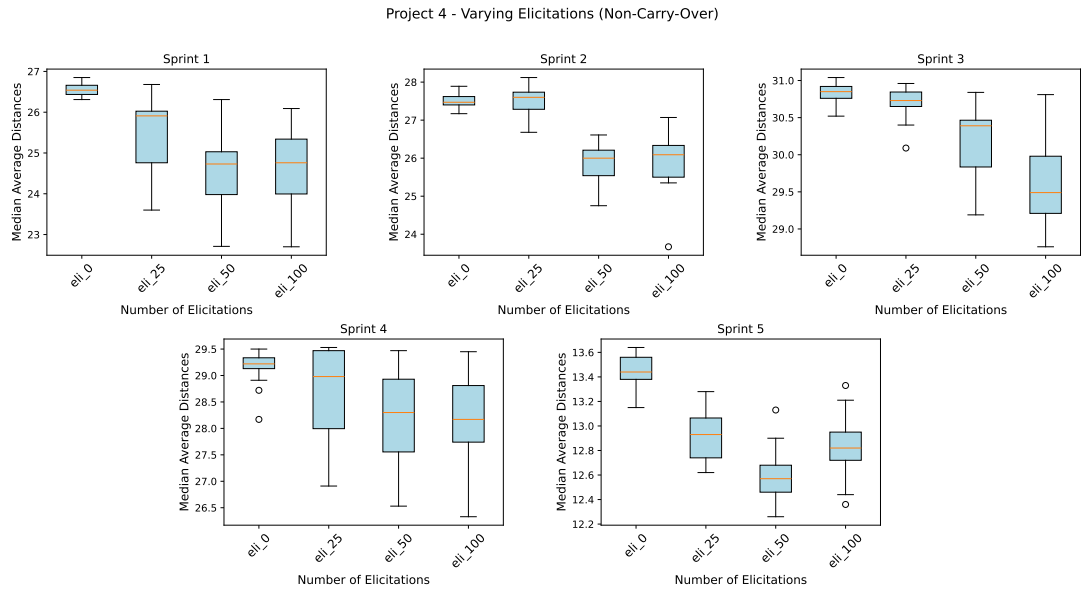


Figure 5.6: Median Average Distances for Carry-Over (a) and Non-Carry-Over (b) showing impact of issue rollover at varying elicitations for P4

(a) Carry-Over



(b) Non-Carry-Over



5.5.1 Project-Level Analysis

Figures 5.5 and 5.6 show that, for Project 4, performance generally improves under the simulated carry-over scenario compared to the non-carry-over scenario, for both disagreement and average distance metrics across sprints. This is generally reflected in both median values and inter-quartile ranges, suggesting a benefit from the carry-over simulation. Table 5.9 further supports this, with all elicitation levels showing statistical significance.

However, the simulation has limitations as only the largest five sprints were considered, which does not fully reflect the chronological order or the complex dependencies present in the actual project sprints. As a result, variation across projects is observed, highlighting that the effect of carry-over is not uniform.

For instance, Project 13 showed statistical significance for disagreement only at 0 and 100 elicitations, with no significant differences for average distance (Table E.2). Project 12 exhibited significance across all elicitation levels for disagreement only (Table E.1). In contrast, Project 25 showed statistically significant improvements for both metrics at all elicitation levels (Table E.3).

These results indicate that the impact of carrying over unresolved issues varies across projects and metrics, which may in part reflect limitations of the carry-over simulation method employed. Further research is needed to determine how these methodological factors influence the observed variability.

5.5.2 Sprint-Level Analysis

At the sprint level, the impact of the carry-over simulation generally mirrors the project-level trends for Project 4. Across most sprints, both disagreement and average distance metrics improve in the carry-over scenario compared to the non-carry-over

scenario, as shown in Figures 5.5 and 5.6. These improvements are visible not only in the median values but also in the inter-quartile ranges, indicating that the solver produces more consistent outcomes when unresolved issues are carried over to subsequent sprints.

Table 5.9 confirms that these improvements are statistically significant across all elicitation levels. The gains are most pronounced in earlier sprints (Sprints 1 and 2), while in later sprints, particularly Sprints 3 and 5, the improvements are smaller. This pattern highlights variation in the impact of the carry-over simulation across sprints, though the underlying factors require further research, potentially involving a more robust carry-over method.

5.6 Comparison to State-Of-the-Art Methods

(RQ6)

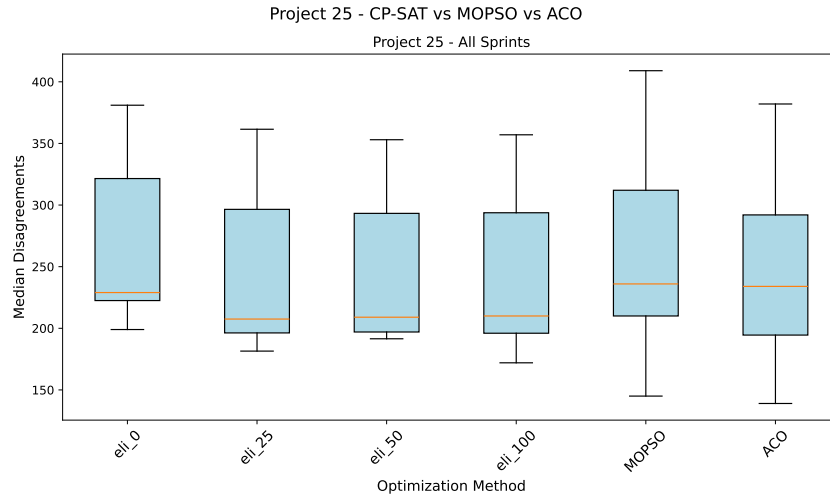
RQ6 investigated how the proposed method compares to state-of-the-art methods, namely Ant-Colony Optimization and Multi-Objective Particle Swarm Optimization (MOPSO).

5.6.1 Project-Level Analysis

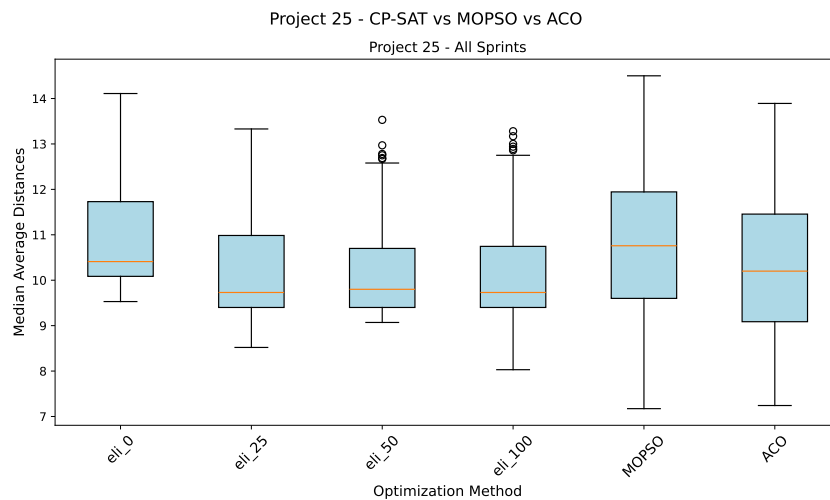
Based on Figure 5.7, for Project 25, CP-SAT consistently outperforms both MOPSO and ACO in terms of median disagreement and average distance, with average distance performance generally improving as more user input is elicited. Even at 0 elicitations, CP-SAT shows superior performance in both metrics, likely because, unlike the metaheuristic approaches, it incorporates both stakeholder input and constraints to guide the solver toward more optimal solutions. Table 5.4 highlights

Figure 5.7: Median Disagreement (a) and Median Average Distance (b) across different optimization methods for P25

(a) Median Disagreement



(b) Median Average Distance



this advantage, with CP-SAT achieving a 3.21% lower disagreement than ACO and 4.03% lower than MOPSO at 0 elicitations. For average distance, it outperforms ACO by 3.25% and MOPSO by 1.14%, demonstrating that even without elicited input, CP-SAT’s knowledge of historical data provides an edge for Project 25. Average distance improvements become more pronounced as additional user input is provided

Table 5.10: Statistical tests for Project 25 comparing CP-SAT against MOPSO and ACO. *Bold p-values indicate statistical significance ($p < 0.05$).*

Metric	Comparison	Test	p-value
Disagreement	–	Kruskal-Wallis	0.0312
	eli_0 vs MOPSO	Mann-Whitney U	1.0000
	eli_0 vs ACO	Mann-Whitney U	1.0000
	eli_25 vs MOPSO	Mann-Whitney U	1.0000
	eli_25 vs ACO	Mann-Whitney U	1.0000
	eli_50 vs MOPSO	Mann-Whitney U	1.0000
	eli_50 vs ACO	Mann-Whitney U	1.0000
	eli_100 vs MOPSO	Mann-Whitney U	1.0000
	eli_100 vs ACO	Mann-Whitney U	1.0000
Average Distance	–	Kruskal-Wallis	0.0203
	eli_0 vs MOPSO	Mann-Whitney U	1.0000
	eli_0 vs ACO	Mann-Whitney U	0.8013
	eli_25 vs MOPSO	Mann-Whitney U	1.0000
	eli_25 vs ACO	Mann-Whitney U	1.0000
	eli_50 vs MOPSO	Mann-Whitney U	1.0000
	eli_50 vs ACO	Mann-Whitney U	1.0000
	eli_100 vs MOPSO	Mann-Whitney U	1.0000
	eli_100 vs ACO	Mann-Whitney U	1.0000

but slightly decline in terms of disagreement after 25 elicitations. In addition, at a 20% error rate, CP-SAT’s performance degrades, resulting in higher disagreement and average distance than both ACO and MOPSO. However, despite generally better results, Table 5.10 shows that none of the differences are statistically significant for Project 25, with many p -values equal to 1. This suggests that observed benefits of CP-SAT are minimal which may be influenced by suboptimal constraint choices or by the specific environment setup, which might not fully align with Project 25’s historical characteristics.

In contrast, statistical significance was achieved at all elicitation levels for Project 12 and Project 13, as seen in Table F.2 and Table F.3. CP-SAT outperformed both ACO and MOPSO in disagreement and average distance, even at 0 elicitations and 20% error rates, when considering project medians (Table 5.2 and Table 5.3). For Project 4, Table F.1 shows statistical significance for average distance at lower

elicitation levels (0 and 25), where both MOPSO and ACO outperformed CP-SAT. However, no significant difference was observed for disagreement or at higher elicitation levels, indicating that neither MOPSO nor ACO consistently outperformed CP-SAT (Table 5.1).

These results suggest that CP-SAT has the potential to provide performance advantages, potentially in cases where the constraints more closely align with a project's historical resolution order. Further research is needed to understand why CP-SAT outperformed ACO and MOPSO in some projects (P12 and P13) while underperforming or stagnating in others (P4 and P25), and whether alternative configurations or constraint strategies could yield more consistent gains.

Chapter 6

Discussion

This section interprets the results presented in Section 5, highlighting key trends, practical implications, and insights across the research questions.

6.1 Role of Interaction

As shown in Figure 5.1, Project 4 generally exhibits improvements in both disagreement and average distance as additional elicitation is received. Table 5.5 further supports this trend, showing that all elicitation levels statistically significantly outperform the non-interactive baseline. However, gains may begin to plateau between 50 and 100 elicitations, as observed in Table 5.1. In some sprints, performance slightly decreases, highlighting a tradeoff between satisfying constraints and adhering closely to user input. Sprint-level analysis reveals that while most sprints show improvement beyond 50 elicitations, it can also lead to plateauing (as seen in Sprint 4) or worsening (as seen in Sprint 3). Other projects (Projects 12, 13, and 25) also show no improvement between elicitation levels except when compared to the non-interactive method.

Overall, these results suggest that moderate elicitation levels may provide an optimal balance between performance and stability, as previous work with SMT-based solvers such as Yices [37] and Z3 [55] has shown that they consistently outperform approaches relying solely on user input, such as AHP.

From a practical standpoint, these findings suggest that users should carefully consider the number of elicitation inputs provided. While increasing elicitation generally improves alignment with stakeholder preferences, levels beyond 50 may result in diminishing returns and introduce variability in some sprints, as observed in Sprints 1, 4, and 5. Setting elicitation too high could therefore reduce solution stability, while moderate levels appear to strike a balance between effectively incorporating user input and maintaining consistent performance.

6.2 Role of Constraints

Based on the results shown in Table 5.2 and Figure 5.2, our chosen constraint configurations exhibit varying performance across sprints, making it difficult to identify a single best overall configuration. However, `c1t1` performed consistently well across most sprints for Project 12, while `c2p1t1` generally performed poorly.

Similar variation in constraint performance was observed across other projects. This highlights the challenges Product Owners face when deciding which tasks to prioritize, demonstrating how much performance can vary depending on the chosen set of factors considered. These findings underscore the importance of further research exploring additional constraint configurations and establishing a more robust baseline for comparisons.

6.3 Robustness

The system generally declined in performance as the error rate increased, as shown in Figure 5.3. However, Table 5.7 shows that most performance differences were not statistically significant for Project 13, indicating the system’s robustness even as error rates increased. Table 5.3 further emphasizes this, as Project 13’s performance remains superior to ACO and MOPSO even at 20% error rates. This robustness can also be attributed to the effectiveness of the chosen constraints. As observed for Project 25 in Table 5.4 and Project 4 in Table 5.1, when constraints do not align properly with the gold standard, error rates have a greater impact, potentially degrading performance below ACO and MOPSO. This reinforces points raised in **RQ2** and **RQ4**, emphasizing that the constraint configuration is essential for providing a reliable baseline to improve upon with elicitation, as discussed in **RQ1**.

6.4 Impact of Assignee-Level Prioritization

The positive impact of local constraints is evident in our project-level analysis. Figure 5.4 shows that most sprints in Project 12 improve when employing the local variant of the constraint. Table 5.8 confirms that the performance differences between local and global constraints are statistically significant, highlighting the benefits of prioritizing at an assignee level rather than globally. This trend is also consistent across Projects 4, 13, and 25, although Project 13 did not achieve statistical significance for either metric.

This suggests that incorporating assignee-specific context enables the solver to reflect individual workloads and responsibilities better, resulting in more relevant and realistic prioritization orders.

Furthermore, the advantage of local prioritization appears more pronounced in smaller sprints, where the distribution of tasks across assignees can vary greatly. In contrast, in larger sprints, the difference between local and global approaches tends to diminish, as seen in Figure 5.8.

These findings indicate that project managers may gain more precise control over task sequencing and alignment with stakeholder preferences when assignee-level constraints are applied, particularly in projects with smaller, tightly scoped sprints or uneven task distributions.

6.5 Handling of Unresolved Issues

Figures 5.6 and 5.5 highlight the importance of historical context in guiding prioritization decisions. Across Project 4, carrying over unresolved issues generally led to improved median disagreement and average distance.

Table 5.9 indicates that these gains are statistically significant across all elicitation levels. However, these results are not fully indicative of real-world scenarios, as our issue carry-over method has limitations. For instance, we used the five largest sprints, which do not fully reflect real-world chronological ordering. We carried over 10% of the latest issues from the previous sprint, which may not have been prioritized in reality. Variability was also observed in other projects and even across sprints in Project 4, hinting at inconsistencies due to the carry-over method. Further research is needed to validate these findings in practical settings.

6.6 Comparison with State-of-the-Art Methods

For Project 25, CP-SAT outperformed ACO and MOPSO across most conditions, except the highest simulated error rate of 20%, where both ACO and MOPSO outperformed CP-SAT in both metrics.

In contrast, the best-performing CP-SAT configuration, using 25 elicited pairs for disagreement, achieved an improvement of 11.75% over ACO and 12.5% over MOPSO, demonstrating noticeably closer alignment with the historical resolution order. This indicates that CP-SAT is generally more effective at incorporating both constraints and elicited user preferences to produce orderings that reflect stakeholder priorities, except at higher error rates.

Similar performance trends are observed in Projects 12 and 13 (Table 5.2 and Table 5.3), reinforcing the robustness of CP-SAT across multiple project contexts. However, in Project 4, both ACO and MOPSO outperform CP-SAT (Table 5.1), which is likely attributable to the `c1p1` constraint configuration not being well-aligned with the project's gold-standard order. In this scenario, the metaheuristic methods were likely able to explore the solution space more freely and identify more optimal orderings than CP-SAT, constrained by suboptimal settings.

These results further highlight the findings for **RQ2** and **RQ4**, emphasizing the importance of selecting appropriate constraint configurations when using CP-SAT and illustrating that, while generally effective, its performance is sensitive to how well the constraints capture project-specific prioritization factors.

6.7 Implications

The study provides several practical insights for Product Owners:

1. Moderate elicitation levels (around 50 inputs) strike a balance between capturing stakeholder preferences and maintaining solution stability. Excessively high elicitation levels may introduce variability, especially if constraint configurations are misaligned with prioritization goals, so aiming for a moderate input size is recommended (Figure 5.1, Table 5.5).
2. Constraint choice and weighting meaningfully influence outcomes. Tailoring configurations to project-specific priorities, can significantly improve alignment and performance, rather than applying a uniform configuration across all projects (Figure 5.2, Table 5.6).
3. While the system is generally robust, higher error rates (around 20%) may degrade performance. Product Owners should be mindful of potential inaccuracies in inputs and consider mitigation strategies when errors are likely (Figure 5.3, Table 5.7)
4. Applying constraints at the assignee level is more effective than global constraints, especially in smaller sprints. Incorporating individual workload as context in the prioritization process can improve overall performance (Figure 5.4, Table 5.8).
5. Simulating issue rollover can generally enhance prioritization performance, though its effects across different projects require further investigation (Figures 5.6 and 5.5, Table 5.9).
6. The system generally outperforms metaheuristic optimization techniques (MOPSO and ACO), demonstrating that the factors and constraints selected meaningfully influence prioritization outcomes (Figure 5.7, Table 5.10).

6.8 Limitations

Several limitations relate to experimental design and scope:

No single constraint was best across all projects, but `c1p1` was chosen for this study as it was considered the strongest performer overall, as discussed in our methodology. Using this general constraint instead of project-specific configurations may have limited performance.

The experiments considered only the five largest sprints per project and approximated issue carry-over based on resolution dates rather than full chronological sprint order.

Evaluation metrics (disagreement and average distance) are not normalized across sprint sizes, limiting cross-project comparisons.

The comparison with metaheuristic methods is constrained by using the same average run and population sizes; reducing input diversity may limit convergence and solution quality of these methods.

The baseline used for comparison, based on creation date, may not be the most robust choice. Alternative baselines could provide a stronger point of reference for evaluating performance.

Finally, the assumption that all issues coexist for constraint pairing and gold-standard comparisons may misrepresent real-world relationships, and the current system requires significant computational overhead, limiting scalability.

6.9 Algorithm Complexity

The issue prioritization problem considered in this thesis involves generating a total ordering of issues subject to a combination of hard (non-retractable) and soft

(retractable) constraints. Formally, given a set of n issues and a set of constraints \mathcal{C} , the solver must find a permutation of issues that minimizes the weighted sum of soft constraint violations while satisfying all hard constraints, as outlined in Algorithm 1.

From a computational perspective, this problem is challenging. If only hard constraints are present and they form a directed acyclic graph, a feasible ordering can be found in polynomial time using a standard topological sort. However, once soft constraints, pairwise elicited preferences (enforced as hard constraints), and potential conflicts are introduced, the problem becomes combinatorial: the number of possible orderings grows factorially with n ($O(n!)$), making exhaustive enumeration infeasible for realistic project sizes.

The optimization problem of finding an ordering that maximizes satisfaction of soft constraints is **NP-hard**, following the perspective outlined in [16], since it encompasses classical **NP-complete** problems, including weighted constraint satisfaction and topological ordering with conflicts. Even verifying whether a proposed ordering is optimal is computationally intractable when all constraints are taken into account. Consequently, optimal solutions for large instances cannot be guaranteed in polynomial time.

To address this, modern constraint solvers such as Google’s CP-SAT leverage techniques such as propagation, pruning, and heuristic search to explore the solution space more efficiently. These solvers can guarantee satisfaction of hard constraints for feasible instances while producing high-quality solutions with respect to soft constraints and elicited preferences, even though optimality cannot be ensured for large n . However, the solver’s practical performance depends on factors such as the number of issues, the density and structure of constraints, and the amount of user input.

6.10 Threats to Validity

This section discusses the threats to the validity of this thesis and outlines the efforts taken to mitigate them.

6.10.1 External Validity

External validity concerns the extent to which our findings generalize beyond the studied context.

The simulation environment does not fully reflect real-world project conditions. Only the five largest sprints from each project were analyzed, which limits the diversity of sprint sizes, team behaviors, and prioritization patterns represented in the dataset. This also affects the realism of the simulated issue rollover for **RQ5**, as the selected sprints do not correspond to the actual chronological ordering of sprints within each project. As a result, the carry-over mechanism may not accurately capture how unresolved issues persist across sprint boundaries in practice.

The study examined four projects from the TAWOS dataset [49], and these projects may not reflect characteristics found in other domains or in development methodologies beyond Scrum. Constraint preferences, team structures, and stakeholder decision-making vary across organizations, meaning the findings may not generalize broadly. Additional analysis on a larger and more diverse set of projects would be required to establish broader applicability.

6.10.2 Internal Validity

Internal validity refers to factors that may influence causal interpretations of the results. Several elements of the experimental setup could have influenced the observed

performance differences.

First, simulated user elicitations and fixed error rates introduce artificial noise that may not represent real human decision-making. Effort was made to ensure user input aligned with historical stakeholder behavior, but the manner in which the gold standard was defined presents limitations. Rather than assuming a single explicit ordering, a more realistic approach would involve multiple partial gold standards because issues do not always coexist within a sprint, so pairwise comparisons should account for temporal availability. In addition, not all issues are naturally compared against every other issue, meaning only some orderings should be enforced when evaluating alignment.

Although fixed error percentages (used in evaluating **RQ3**) allow for controlled experimentation, they do not capture the irregular, context-dependent nature of real user inconsistencies.

The constraint configurations were selected based on the overall best-performing constraint rather than optimized per project. This reduces project-level generalizability, since the global baseline configuration, `c1p1`, may not represent the ideal constraint set for each individual project.

Our analysis relies on nonparametric tests (Kruskal–Wallis H and Mann–Whitney U) due to the universal violation of the normality assumption (Shapiro–Wilk test). While robust to non-normal data, the nonparametric approach is less powerful than ANOVA, increasing the risk of a Type II error (false negative). Furthermore, the required post-hoc Mann–Whitney U tests employ the Holm correction to control the Family-Wise Error Rate (FWER). This necessary correction sets a more conservative significance threshold, which also contributes to the risk of Type II errors.

Additionally, performance may be affected by solver parameter choices, such as the time limit for CP-SAT, the population size and average runs. These choices constrain

the search space differently and may influence outcomes independent of the underlying prioritization logic.

6.10.3 Construct Validity

Construct validity concerns whether the measurements accurately capture the intended concepts.

The evaluation metrics used in this study: disagreement and average distance, are established measures in previous work [55, 37], but they do not capture all nuances of prioritization quality. The disagreement metric is not normalized by sprint size, which complicates comparisons across sprints or projects of differing scales. Larger sprints may naturally exhibit higher disagreement even when relative ranking quality is similar. This can skew the results of project-level analysis.

The gold-standard ordering is based on historical resolution order, assuming that the observed sequence reflects optimal or stakeholder-preferred prioritization. However, resolution order can be shaped by factors unrelated to prioritization, such as developer availability, unexpected changes in scope, or operational constraints. This limits the extent to which the gold standard reflects true stakeholder intent. While effort was taken to mitigate this issue, for example, by removing retroactively created issues and considering only issues that were actually resolved, some noise may still remain in the data.

6.10.4 Reliability Validity

Reliability refers to the consistency and reproducibility of the experimental results.

The simulations rely on stochastic processes when generating user elicitations, error rates, and metaheuristic search behavior. Although multiple runs were averaged to

reduce randomness, repeated experimentation could still lead to small variations in results. Metaheuristic methods such as ACO and MOPSO are particularly sensitive to initialization conditions, which may affect reproducibility unless seeds are fixed.

Additionally, the computational overhead of CP-SAT creates practical constraints on rerunning experiments at large scale. Since averaging all the configurations required long solver runtimes, it may not be feasible to repeat all experiments under slightly altered conditions, which poses a threat to long-term reproducibility and scalability.

Finally, implementation-specific factors, such as the way constraints were encoded, the solver time limit, and dataset pre-processing decisions, may influence outcomes. While care was taken to ensure correctness, these operational choices introduce potential sources of variability across repeated executions.

To increase reliability and reproducibility, all resources related to this thesis are available at <https://github.com/tia-salmon/CP-SAT-Optimization>.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

This study investigated the roles of user interaction, constraint selection, error tolerance, issue carry-over, and assignee-level prioritization in constraint-based artifact prioritization, and compared our approach with two metaheuristic optimization techniques: ACO and MOPSO.

Our findings indicate that moderate elicitation levels generally enhance alignment with stakeholder preferences while maintaining solution stability (**Role of Interaction, RQ1**). Careful selection and weighting of constraints significantly influence prioritization quality (**Role of Constraints, RQ2**). This is true for both the constraints chosen (**Varying Constraints, RQ2.1**) and the weights applied (**Varying Weights, RQ2.2**). The system was shown to be generally robust to errors during user elicitation (**Robustness, RQ3**). Assignee-level constraints provide additional benefits by capturing context-specific information relevant to individual assignees (**Impact of Assignee-Level Prioritization, RQ4**). Carrying over unresolved issues across sprints shows potential to improve prioritization, though further re-

search is needed to validate this due to limitations in the current carry-over method (**Handling of Unresolved Issues, RQ5**). Finally, CP-SAT was shown to generally outperform metaheuristic optimization techniques, MOPSO and ACO, under the simulated conditions depending on the project (**Comparison with State-of-the-Art Methods, RQ6**).

7.2 Future Work

Several directions can extend this study’s findings and address its current limitations.

One aspect that could be explored is incorporating chronological project histories to reflect real-world sprint carry-over practices better. This could be refined to track issues based on their actual creation and resolution dates, rather than approximating them using the most recently resolved issues from the previous sprint.

Similarly, constraint pairings and gold-standard comparisons could be made more precise by considering only issues that coexisted using partial orderings, thus allowing multiple gold standards to be evaluated.

Evaluation metrics could be further developed. Normalized measures and proportional elicitation levels would allow fairer comparisons across sprints of varying sizes, and additional metrics could enable deeper analysis of the effectiveness of our method beyond disagreement and average distance.

Another important direction is the incorporation of machine learning techniques such as Learning-to-Rank [48, 39] to complement constraint-based prioritization. While this thesis did not involve ML techniques, as our goal was to evaluate the constraint-based method in isolation, for practical applications, predictive models could help estimate stakeholder preferences and recommend constraint weightings dynamically based on historical data rather than relying on fixed and manually assigned constraints.

Hybrid approaches combining ML-based learning with constraint-based optimization could improve performance, scalability, robustness, and alignment with real-world project behaviors.

Finally, future studies should evaluate the approach in larger, more complex projects with varied artifact sizes, richer sprint histories, and diverse team structures to test generalizability and practical applicability in industrial settings.

Bibliography

- [1] Mugilan A, Tushar Totla, Yash Renwa, Charanya R, Stalin Subbiah, T. B. Dharmaraj, and S. Om Prakash, *Comparative analysis of ant colony optimization and particle swarm optimization for test case prioritization*, 2022 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT), Nov 2022, pp. 680–686.
- [2] Khadija Sania Ahmad, Nazia Ahmad, Hina Tahir, and Shaista Khan, *Fuzzy moscow: A fuzzy based moscow method for the prioritization of software requirements*, 2017 International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICICT), 2017, pp. 433–437.
- [3] Hafiza Anisa Ahmed, Narmeen Zakaria Bawany, and Jawwad Ahmed Shamsi, *Capbug-a framework for automatic bug categorization and prioritization using nlp and machine learning algorithms*, IEEE Access **9** (2021), 50496–50512.
- [4] Abdullah Alzaqebah, Raja Masadeh, and Amjad Hudaib, *Whale optimization algorithm for requirements prioritization*, 2018 9th International Conference on Information and Communication Systems (ICICS), April 2018, pp. 84–89.
- [5] Rahila Anwar and Muhammad Bilal Bashir, *A systematic literature review of ai-based software requirements prioritization techniques*, IEEE Access **11** (2023), 143815–143860.

- [6] Anu Bajaj and Om Prakash Sangwan, *A systematic literature review of test case prioritization using genetic algorithms*, IEEE Access **7** (2019), 126355–126375.
- [7] Clark Barrett and Cesare Tinelli, *Satisfiability modulo theories*, pp. 305–343, Springer International Publishing, Cham, 2018.
- [8] Antonia Bertolino, Antonio Guerriero, Breno Miranda, Roberto Pietrantuono, and Stefano Russo, *Learning-to-rank vs ranking-to-learn: strategies for regression testing in continuous integration*, Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (New York, NY, USA), ICSE '20, Association for Computing Machinery, 2020, p. 1–12.
- [9] Nikolaž Bjorner, *Smt solvers for testing, program analysis and verification at microsoft*, 2009 11th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Sep. 2009, pp. 15–15.
- [10] Endre Boros and Peter L. Hammer, *Pseudo-boolean optimization*, Discrete Applied Mathematics **123** (2002), no. 1, 155–225.
- [11] Panagiota Chatzipetrou, Lefteris Angelis, Per Rovegård, and Claes Wohlin, *Prioritization of issues and requirements by cumulative voting: A compositional data analysis framework*, 2010 36th EUROMICRO Conference on Software Engineering and Advanced Applications, 2010, pp. 361–370.
- [12] Fanliang Chen, Zheng Li, Ying Shang, and Yang Yang, *Focus on new test cases in continuous integration testing based on reinforcement learning*, 2022 IEEE 22nd International Conference on Software Quality, Reliability and Security (QRS), 2022, pp. 830–841.
- [13] Eddwin Cheteni and William Tichaona Vambe, *Explainability in machine learning & ai models for complex data structures on scorecards development in retail*

- banking*, 2024 4th International Multidisciplinary Information Technology and Engineering Conference (IMITEC), 2024, pp. 315–320.
- [14] Davide Chicco, Alexander Sichenze, and Giuseppe Jurman, *A simple guide to the use of student's t-test, mann-whitney u test, chi-squared test, and kruskal-wallis test in biostatistics*, *BioData Mining* **18** (2025), no. 1, 56.
- [15] C. A. Coello Coello and M. S. Lechuga, *Mopso: a proposal for multiple objective particle swarm optimization*, *Proceedings of the Evolutionary Computation on 2002. CEC '02. Proceedings of the 2002 Congress - Volume 02 (USA), CEC '02*, IEEE Computer Society, 2002, p. 1051–1056.
- [16] Stephen Cook, *The p versus np problem*, Department of Computer Science, University of Toronto, n.d., <https://www.cs.toronto.edu/handouts/cook-clay.pdf>.
- [17] Enrique A. Da Roza, Jackson A. Prado Lima, Rogério C. Silva, and Silvia Regina Vergilio, *Machine learning regression techniques for test case prioritization in continuous integration environment*, 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), 2022, pp. 196–206.
- [18] Ikram Daanoune, Abdennaceur Baghdad, and Abdelhakim Ballouk, *A comparative study between aco-based protocols and pso-based protocols in wsn*, 2019 7th Mediterranean Congress of Telecommunications (CMT), 2019, pp. 1–4.
- [19] Xueqi Dang, Yinghua Li, Mike Papadakis, Jacques Klein, Tegawendé F. Bissyandé, and Yves Le Traon, *Test input prioritization for machine learning classifiers*, *IEEE Transactions on Software Engineering* **50** (2024), no. 3, 413–442.
- [20] Leonardo De Moura and Nikolaj Bjørner, *Z3: an efficient smt solver*, *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools*

and Algorithms for the Construction and Analysis of Systems (Berlin, Heidelberg), TACAS'08/ETAPS'08, Springer-Verlag, 2008, p. 337–340.

- [21] M. Dorigo and G. Di Caro, *Ant colony optimization: a new meta-heuristic*, Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406), vol. 2, 1999, pp. 1470–1477 Vol. 2.
- [22] Ankita Gupta and Chetna Gupta, *Cdbr: A semi-automated collaborative execute-before-after dependency-based requirement prioritization approach*, Journal of King Saud University - Computer and Information Sciences **34** (2022), no. 2, 421–432.
- [23] Yechao Huang, Ting Shu, and Zuohua Ding, *A learn-to-rank method for model-based regression test case prioritization*, IEEE Access **9** (2021), 16365–16382.
- [24] Fadhl Hujainah, Rohani Binti Abu Bakar, Abdullah B. Nasser, Basheer Al-haimi, and Kamal Z. Zamli, *Srptackle: A semi-automated requirements prioritisation technique for scalable requirements of software system projects*, Information and Software Technology **131** (2021), 106501.
- [25] Shahid Iqbal, Rashid Naseem, Salman Jan, Sami Alshmrany, Muhammad Yasar, and Arshad Ali, *Determining bug prioritization using feature reduction and clustering with classification*, IEEE Access **8** (2020), 215661–215678.
- [26] Kamaljit Kaur and Parminder Kaur, *The application of ai techniques in requirements classification: a systematic mapping*, Artificial Intelligence Review **57** (2024), no. 3, 57.
- [27] Muhammad Khatibsyarbini, Mohd Adham Isa, Dayang N. A. Jawawi, Haza Nuzly Abdull Hamed, and Muhammad Dhiauddin Mohamed Suffian, *Test case prioritization using firefly algorithm for software testing*, IEEE Access **7** (2019), 132360–132373.

- [28] Bengt Lennartson, *Optimization of timed petri nets using cp-sat*, IFAC-PapersOnLine **58** (2024), no. 1, 90–95, 17th IFAC Workshop on discrete Event Systems WODES 2024.
- [29] Longbo Li, Yanhui Zhou, Yuan Yuan, and Shenghua Wu, *An extensive study on multi-priority algorithm in test case prioritization and reduction*, ASSE '21, Association for Computing Machinery, 2021, p. 48–57.
- [30] Tianhai Liu, Michael Nagel, and Mana Taghdiri, *Bounded program verification using an smt solver: A case study*, 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation, April 2012, pp. 101–110.
- [31] Qi Luo, Kevin Moran, Denys Poshyvanyk, and Massimiliano Di Penta, *Assessing test case prioritization on real faults and mutants*, 2018.
- [32] Eber Junio Borges Moreira and Sergio Antonio Andrade De Freitas, *A cp-sat approach for academic resource timetabling in higher education institutions: A case study at a major public university*, 2024 21st International Conference on Information Technology Based Higher Education and Training (ITHET), Nov 2024, pp. 1–8.
- [33] Naveen Kumar Navuri and Cvpr Prasad, *Integrating enhanced learning to rank into a hybrid deep learning system for optimized recommendations*, 2025 1st International Conference on Secure IoT, Assured and Trusted Computing (SATC), 2025, pp. 1–10.
- [34] Muhammad Nazir, Arif Mehmood, Waqar Aslam, Yongwan Park, Gyu Sang Choi, and Imran Ashraf, *A multi-goal particle swarm optimizer for test case prioritization*, IEEE Access **11** (2023), 90683–90697.

- [35] Jaideep Nijjar and Tefvik Bultan, *Unbounded data model verification using smt solvers*, 2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, Sep. 2012, pp. 210–219.
- [36] Safa Omri and Carsten Sinz, *Learning to rank for test case prioritization*, 2022 IEEE/ACM 15th International Workshop on Search-Based Software Testing (SBST), 2022, pp. 16–24.
- [37] Francis Palma, Angelo Susi, and Paolo Tonella, *Using an smt solver for interactive requirements prioritization*, Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (New York, NY, USA), ESEC/FSE '11, Association for Computing Machinery, 2011, p. 48–58.
- [38] SLRRongqi Pan, Mojtaba Bagherzadeh, Taher A. Ghaleb, and Lionel Briand, *Test case selection and prioritization using machine learning: a systematic literature review*, Empirical Software Engineering **27** (2021), no. 2, 29.
- [39] Anna Perini, Filippo Ricca, and Angelo Susi, *Tool-supported requirements prioritization: Comparing the ahp and cbrank methods*, Information and Software Technology **51** (2009), no. 6, 1021–1032.
- [40] Barkhah Permana, Ridi Ferdiana, and Azkario Pratama, *Large language model employment for story point estimation problems in agile development*, 2024 International Conference on Electrical Engineering and Computer Science (ICECOS), Sep. 2024, pp. 391–398.
- [41] Saif Ur Rehman Khan, Maha Younus, Javed Iqbal, and Muhammad Abdul Basit Ur Rahim, *A fuzzy ahp-based quantitative framework to prioritize the crowd-based requirements*, 2024 IEEE 24th International Conference on Software Quality, Reliability, and Security Companion (QRS-C), July 2024, pp. 680–691.

- [42] R.W. Saaty, *The analytic hierarchy process—what it is and how it is used*, Mathematical Modelling **9** (1987), no. 3, 161–176.
- [43] Tianna-Lee Salmon, Dawn MacIsaac, and Francis Palma, *Issue prioritization in agile development through the lens of constraint solving*, 2025 IEEE International Conference on Collaborative Advances in Software and COmputing (CASCON), Sep. 2025.
- [44] Mohammad Shameem, Rakesh Ranjan Kumar, Chiranjeev Kumar, Bibhas Chandra, and Arif Ali Khan, *Prioritizing challenges of agile process in distributed software development environment using analytic hierarchy process*, Journal of Software: Evolution and Process **30** (2018), no. 11, e1979.
- [45] Aizaz Sharif, Dusica Marijan, and Marius Liaaen, *Deeporder: Deep learning for test case prioritization in continuous integration testing*, 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2021, pp. 525–534.
- [46] Aurora SLRRamírez, Robert Feldt, and José Raúl Romero, *A taxonomy of information attributes for test case prioritisation: Applicability, machine learning*, ACM Trans. Softw. Eng. Methodol. **32** (2023), no. 1.
- [47] Hassan Tahir, Saif Ur Rehman Khan, and Syed Sohaib Ali, *Lcbpa: An enhanced deep neural network-oriented bug prioritization and assignment technique using content-based filtering*, IEEE Access **9** (2021), 92798–92814.
- [48] Peng Tang, Junfeng Wang, and Mingxing Liu, *Variational learning to rank for test case prioritization via prioritizing metric inspired differentiable loss*, Engineering Applications of Artificial Intelligence **141** (2025), 109776.
- [49] Vali Tawosi, Afnan Al-Subaih, Rebecca Moussa, and Federica Sarro, *A versatile dataset of agile open source software projects*, 2022.

- [50] Vali Tawosi, Rebecca Moussa, and Federica Sarro, *Agile effort estimation: Have we solved the problem yet? insights from a replication study*, IEEE Transactions on Software Engineering **49** (2023), no. 4, 2677–2697.
- [51] Paolo Tonella, Angelo Susi, and Francis Palma, *Using interactive ga for requirements prioritization*, 2nd International Symposium on Search Based Software Engineering, Sep. 2010, pp. 57–66.
- [52] Paolo Tonella, Angelo Susi, and Francis Palma, *Interactive requirements prioritization using a genetic algorithm*, Information and Software Technology **55** (2013), no. 1, 173–187, Special section: Best papers from the 2nd International Symposium on Search Based Software Engineering 2010.
- [53] Qasim Umer, Hui Liu, and Inam Illahi, *Cnn-based automatic prioritization of bug reports*, IEEE Transactions on Reliability **69** (2020), no. 4, 1341–1354.
- [54] Arif Usta, Ismail Sengor Altingovde, Rifat Ozcan, and Özgür Ulusoy, *Learning to rank for educational search engines*, IEEE Transactions on Learning Technologies **14** (2021), no. 2, 211–225.
- [55] Jonathan Winton and Francis Palma, *Improving software requirements prioritization through the lens of constraint solving*, 2023.

Appendix A

RQ1: Role of Interaction Statistical Analysis

This chapter provides a detailed statistical analysis of the impact of elicitations on the performance of our method (RQ1) for the remaining projects (P12, P13, and P25).

Table A.1: Statistical tests for Project 12 showing significance of varying elicitations. *Bold p-values indicate statistical significance ($p < 0.05$).*

Metric	Comparison	Test	p-value
Disagreement	–	Kruskal-Wallis	0.0014
	0 vs 25	Mann-Whitney U	0.0081
	0 vs 50	Mann-Whitney U	0.0079
	0 vs 100	Mann-Whitney U	0.0095
	25 vs 50	Mann-Whitney U	0.9250
	25 vs 100	Mann-Whitney U	0.6221
	50 vs 100	Mann-Whitney U	0.7365
Average Distance	–	Kruskal-Wallis	0.0495
	0 vs 25	Mann-Whitney U	0.0905
	0 vs 50	Mann-Whitney U	0.1277
	0 vs 100	Mann-Whitney U	0.1256
	25 vs 50	Mann-Whitney U	1.0000
	25 vs 100	Mann-Whitney U	0.8775
	50 vs 100	Mann-Whitney U	1.0000

Table A.2: Statistical tests for Project 13 showing significance of varying elicitations. *Bold p-values indicate statistical significance ($p < 0.05$).*

Metric	Comparison	Test	p-value
	–	Kruskal-Wallis	0.0075
	0 vs 25	Mann-Whitney U	0.0262
	0 vs 50	Mann-Whitney U	0.0227
Disagreement	0 vs 100	Mann-Whitney U	0.0247
	25 vs 50	Mann-Whitney U	1.0000
	25 vs 100	Mann-Whitney U	1.0000
	50 vs 100	Mann-Whitney U	0.8983
	–	Kruskal-Wallis	0.0071
	0 vs 25	Mann-Whitney U	0.0159
Average	0 vs 50	Mann-Whitney U	0.0305
Distance	0 vs 100	Mann-Whitney U	0.0245
	25 vs 50	Mann-Whitney U	1.0000
	25 vs 100	Mann-Whitney U	1.0000
	50 vs 100	Mann-Whitney U	0.9715

Table A.3: Statistical tests for Project 25 showing significance of varying elicitations. *Bold p-values indicate statistical significance ($p < 0.05$).*

Metric	Comparison	Test	p-value
Disagreement	–	Kruskal-Wallis	0.0016
	0 vs 25	Mann-Whitney U	0.0033
	0 vs 50	Mann-Whitney U	0.0102
	0 vs 100	Mann-Whitney U	0.0101
	25 vs 50	Mann-Whitney U	1.0000
	25 vs 100	Mann-Whitney U	0.9835
	50 vs 100	Mann-Whitney U	1.0000
Average Distance	–	Kruskal-Wallis	0.0014
	0 vs 25	Mann-Whitney U	0.0042
	0 vs 50	Mann-Whitney U	0.0077
	0 vs 100	Mann-Whitney U	0.0083
	25 vs 50	Mann-Whitney U	1.0000
	25 vs 100	Mann-Whitney U	0.9400
	50 vs 100	Mann-Whitney U	1.0000

Appendix B

RQ2: Role of Constraints

Statistical Analysis

This chapter provides a detailed statistical analysis of the impact of varying constraint configurations and constraint weights on the performance of our method (RQ2) for the remaining projects (P4, P13, and P25).

Table B.1: Statistical tests for Project 4 showing significance of varying constraints. *Bold p-values indicate statistical significance ($p < 0.05$).*

Metric	Comparison	Test	p-value
Disagreement	–	Kruskal-Wallis	0.0002
	c1 vs c1p1	Mann-Whitney U	1.0000
	c1 vs c1t1	Mann-Whitney U	0.0053
	c1 vs c1p1t1	Mann-Whitney U	0.8730
	c1 vs c1p1t2	Mann-Whitney U	0.3311
	c1 vs c1p2t1	Mann-Whitney U	1.0000
	c1 vs c2p1t1	Mann-Whitney U	0.8744
Average Distance	–	Kruskal-Wallis	<0.0001
	c1 vs c1p1	Mann-Whitney U	0.4448
	c1 vs c1t1	Mann-Whitney U	0.0763
	c1 vs c1p1t1	Mann-Whitney U	0.8010
	c1 vs c1p1t2	Mann-Whitney U	0.9921
	c1 vs c1p2t1	Mann-Whitney U	0.0490
	c1 vs c2p1t1	Mann-Whitney U	0.9439

Table B.2: Statistical tests for Project 13 showing significance of varying constraints. *Bold p-values indicate statistical significance ($p < 0.05$).*

Metric	Comparison	Test	p-value
Disagreement	–	Kruskal-Wallis	<0.0001
	c1 vs c1p1	Mann-Whitney U	1.0000
	c1 vs c1t1	Mann-Whitney U	1.0000
	c1 vs c1p1t1	Mann-Whitney U	1.0000
	c1 vs c1p1t2	Mann-Whitney U	0.0162
	c1 vs c1p2t1	Mann-Whitney U	1.0000
	c1 vs c2p1t1	Mann-Whitney U	0.5854
Average Distance	–	Kruskal-Wallis	<0.0001
	c1 vs c1p1	Mann-Whitney U	1.0000
	c1 vs c1t1	Mann-Whitney U	1.0000
	c1 vs c1p1t1	Mann-Whitney U	1.0000
	c1 vs c1p1t2	Mann-Whitney U	<0.0001
	c1 vs c1p2t1	Mann-Whitney U	1.0000
	c1 vs c2p1t1	Mann-Whitney U	1.0000

Table B.3: Statistical tests for Project 25 showing significance of varying constraints. *Bold p-values indicate statistical significance ($p < 0.05$).*

Metric	Comparison	Test	p-value
Disagreement	–	Kruskal-Wallis	0.0005
	c1 vs c1p1	Mann-Whitney U	0.0483
	c1 vs c1t1	Mann-Whitney U	0.3151
	c1 vs c1p1t1	Mann-Whitney U	1.0000
	c1 vs c1p1t2	Mann-Whitney U	0.0845
	c1 vs c1p2t1	Mann-Whitney U	0.0894
	c1 vs c2p1t1	Mann-Whitney U	1.0000
Average Distance	–	Kruskal-Wallis	<0.0001
	c1 vs c1p1	Mann-Whitney U	<0.0001
	c1 vs c1t1	Mann-Whitney U	0.0371
	c1 vs c1p1t1	Mann-Whitney U	0.0414
	c1 vs c1p1t2	Mann-Whitney U	0.0255
	c1 vs c1p2t1	Mann-Whitney U	0.0127
	c1 vs c2p1t1	Mann-Whitney U	1.0000

Appendix C

RQ3: Robustness Statistical Analysis

This chapter provides a detailed statistical analysis of the impact of errors during user elicitations on the robustness of our method (RQ3) for the remaining projects (P4, P12, and P25).

Table C.1: Statistical tests for Project 4 showing significance of varying error rates. *Bold p-values indicate statistical significance ($p < 0.05$).*

Metric	Comparison	Test	p-value
Disagreement	–	Kruskal-Wallis	0.0345
	b vs err_0	Mann-Whitney U	0.6644
	b vs err_10	Mann-Whitney U	1.0000
	b vs err_20	Mann-Whitney U	0.9430
	err_0 vs err_10	Mann-Whitney U	0.1527
	err_0 vs err_20	Mann-Whitney U	0.0229
	err_10 vs err_20	Mann-Whitney U	0.1087
	Average Distance	–	Kruskal-Wallis
b vs err_0		Mann-Whitney U	0.8458
b vs err_10		Mann-Whitney U	0.2034
b vs err_20		Mann-Whitney U	0.0453
err_0 vs err_10		Mann-Whitney U	0.6737
err_0 vs err_20		Mann-Whitney U	0.3879
err_10 vs err_20		Mann-Whitney U	0.3590

Table C.2: Statistical tests for Project 12 showing significance of varying error rates. *Bold p-values indicate statistical significance ($p < 0.05$).*

Metric	Comparison	Test	p-value
Disagreement	–	Kruskal-Wallis	0.0001
	b vs err_0	Mann-Whitney U	0.0008
	b vs err_10	Mann-Whitney U	0.0170
	b vs err_20	Mann-Whitney U	0.0532
	err_0 vs err_10	Mann-Whitney U	0.0818
	err_0 vs err_20	Mann-Whitney U	0.0434
	err_10 vs err_20	Mann-Whitney U	0.0477
Average Distance	–	Kruskal-Wallis	0.0015
	b vs err_0	Mann-Whitney U	0.0182
	b vs err_10	Mann-Whitney U	0.0259
	b vs err_20	Mann-Whitney U	0.0292
	err_0 vs err_10	Mann-Whitney U	0.3173
	err_0 vs err_20	Mann-Whitney U	0.0885
	err_10 vs err_20	Mann-Whitney U	0.1634

Table C.3: Statistical tests for Project 25 showing significance of varying error rates. *Bold p-values indicate statistical significance ($p < 0.05$).*

Metric	Comparison	Test	p-value
Disagreement	–	Kruskal-Wallis	< 0.0001
	b vs err_0	Mann-Whitney U	0.0032
	b vs err_10	Mann-Whitney U	0.0499
	b vs err_20	Mann-Whitney U	0.0930
	err_0 vs err_10	Mann-Whitney U	0.0397
	err_0 vs err_20	Mann-Whitney U	0.0001
	err_10 vs err_20	Mann-Whitney U	0.0894
Average Distance	–	Kruskal-Wallis	< 0.0001
	b vs err_0	Mann-Whitney U	< 0.0001
	b vs err_10	Mann-Whitney U	0.0008
	b vs err_20	Mann-Whitney U	0.0083
	err_0 vs err_10	Mann-Whitney U	0.0065
	err_0 vs err_20	Mann-Whitney U	0.0007
	err_10 vs err_20	Mann-Whitney U	0.3255

Appendix D

RQ4: Impact of Assignee-Level Prioritization Statistical Analysis

This chapter provides a detailed statistical analysis of the impact of assignee-level on the performance of our method, compared to global assignments (RQ4) for the remaining projects (P4, P13, and P25).

Table D.1: Statistical tests for Project 4 showing significance of global constraints (c1) versus their local counterparts (cl). *Bold p-values indicate statistical significance ($p < 0.05$).*

Metric	Comparison	Test	p-value
	–	Kruskal-Wallis	<0.0001
	c1p1 vs clpl	Mann-Whitney U	<0.0001
Disagreement	c1p1t1 vs clpltl	Mann-Whitney U	<0.0001
	c1t1 vs cltl	Mann-Whitney U	<0.0001
	c1 vs cl	Mann-Whitney U	<0.0001
	–	Kruskal-Wallis	<0.0001
Average	c1p1 vs clpl	Mann-Whitney U	<0.0001
Distance	c1p1t1 vs clpltl	Mann-Whitney U	<0.0001
	c1t1 vs cltl	Mann-Whitney U	<0.0001
	c1 vs cl	Mann-Whitney U	<0.0001

Table D.2: Statistical tests for Project 13 showing significance of global constraints (c1) versus their local counterparts (cl). *Bold p-values indicate statistical significance ($p < 0.05$).*

Metric	Comparison	Test	p-value
	–	Kruskal-Wallis	0.0209
	c1p1 vs clpl	Mann-Whitney U	1.0000
Disagreement	c1p1t1 vs clplt1	Mann-Whitney U	1.0000
	c1t1 vs clt1	Mann-Whitney U	1.0000
	c1 vs cl	Mann-Whitney U	0.6183
	–	Kruskal-Wallis	0.0026
Average	c1p1 vs clpl	Mann-Whitney U	0.7757
	c1p1t1 vs clplt1	Mann-Whitney U	1.0000
Distance	c1t1 vs clt1	Mann-Whitney U	1.0000
	c1 vs cl	Mann-Whitney U	1.0000

Table D.3: Statistical tests for Project 25 showing significance of global constraints (c1) versus their local counterparts (cl). *Bold p-values indicate statistical significance ($p < 0.05$).*

Metric	Comparison	Test	p-value
	–	Kruskal-Wallis	<0.0001
	c1p1 vs clpl	Mann-Whitney U	<0.0001
Disagreement	c1p1t1 vs clpltl	Mann-Whitney U	<0.0001
	c1t1 vs cltl	Mann-Whitney U	<0.0001
	c1 vs cl	Mann-Whitney U	<0.0001
	–	Kruskal-Wallis	<0.0001
Average	c1p1 vs clpl	Mann-Whitney U	<0.0001
Distance	c1p1t1 vs clpltl	Mann-Whitney U	<0.0001
	c1t1 vs cltl	Mann-Whitney U	<0.0001
	c1 vs cl	Mann-Whitney U	<0.0001

Appendix E

RQ5: Handling of Unresolved Issues Statistical Analysis

This chapter provides a detailed statistical analysis of the impact of carrying over unresolved issues from the previous sprint on the performance of our method, compared to not carrying them over (RQ4) for the remaining projects (P12, P13, and P25).

Table E.1: Project 12 comparisons for carry-over vs. non-carry-over. *Bold p-values indicate statistical significance ($p < 0.05$).*

Metric	Comparison	Test	p-value
Disagreement	eli_0	Mann-Whitney U	<0.0001
	eli_25	Mann-Whitney U	<0.0001
	eli_50	Mann-Whitney U	<0.0001
	eli_100	Mann-Whitney U	<0.0001
Average Distance	eli_0	Mann-Whitney U	0.1373
	eli_25	Mann-Whitney U	0.2168
	eli_50	Mann-Whitney U	0.1890
	eli_100	Mann-Whitney U	0.1784

Table E.2: Project 13 comparisons for carry-over vs. non-carry-over. *Bold p-values indicate statistical significance ($p < 0.05$).*

Metric	Comparison	Test	p-value
Disagreement	eli_0	Mann-Whitney U	0.0345
	eli_25	Mann-Whitney U	0.0661
	eli_50	Mann-Whitney U	0.0804
	eli_100	Mann-Whitney U	0.0481
Average	eli_0	Mann-Whitney U	0.2046
	eli_25	Mann-Whitney U	0.4076
Distance	eli_50	Mann-Whitney U	0.5385
	eli_100	Mann-Whitney U	0.4964

Table E.3: Project 25 comparisons for carry-over vs. non-carry-over. *Bold p-values indicate statistical significance ($p < 0.05$).*

Metric	Comparison	Test	p-value
Disagreement	eli_0	Mann-Whitney U	<0.0001
	eli_25	Mann-Whitney U	<0.0001
	eli_50	Mann-Whitney U	<0.0001
	eli_100	Mann-Whitney U	<0.0001
Average	eli_0	Mann-Whitney U	<0.0001
	eli_25	Mann-Whitney U	<0.0001
Distance	eli_50	Mann-Whitney U	<0.0001
	eli_100	Mann-Whitney U	<0.0001

Appendix F

RQ6: Comparison with State-of-the-Art Methods Statistical Analysis

This chapter provides a detailed statistical analysis comparing our method to Ant Colony Optimization and Multi-Objective Particle Swarm Optimization (RQ6) for the remaining projects (P4, P12, and P13).

Table F.1: Statistical tests for Project 4 comparing CP-SAT against MOPSO and ACO. *Bold p-values indicate statistical significance ($p < 0.05$).*

Metric	Comparison	Test	p-value
	–	Kruskal-Wallis	<0.0001
	eli_0 vs MOPSO	Mann-Whitney U	0.4650
	eli_0 vs ACO	Mann-Whitney U	0.9425
	eli_25 vs MOPSO	Mann-Whitney U	1.0000
Disagreement	eli_25 vs ACO	Mann-Whitney U	1.0000
	eli_50 vs MOPSO	Mann-Whitney U	0.9461
	eli_50 vs ACO	Mann-Whitney U	1.0000
	eli_100 vs MOPSO	Mann-Whitney U	1.0000
	eli_100 vs ACO	Mann-Whitney U	1.0000
	–	Kruskal-Wallis	<0.0001
	eli_0 vs MOPSO	Mann-Whitney U	0.0001
	eli_0 vs ACO	Mann-Whitney U	0.0028
Average	eli_25 vs MOPSO	Mann-Whitney U	0.0026
Distance	eli_25 vs ACO	Mann-Whitney U	0.0317
	eli_50 vs MOPSO	Mann-Whitney U	0.1260
	eli_50 vs ACO	Mann-Whitney U	0.2931
	eli_100 vs MOPSO	Mann-Whitney U	0.2297
	eli_100 vs ACO	Mann-Whitney U	0.2888

Table F.2: Statistical tests for Project 12 comparing CP-SAT against MOPSO and ACO. *Bold p-values indicate statistical significance ($p < 0.05$).*

Metric	Comparison	Test	p-value
	–	Kruskal-Wallis	<0.0001
	eli_0 vs MOPSO	Mann-Whitney U	0.0014
	eli_0 vs ACO	Mann-Whitney U	0.0026
	eli_25 vs MOPSO	Mann-Whitney U	<0.0001
Disagreement	eli_25 vs ACO	Mann-Whitney U	0.0001
	eli_50 vs MOPSO	Mann-Whitney U	<0.0001
	eli_50 vs ACO	Mann-Whitney U	0.0001
	eli_100 vs MOPSO	Mann-Whitney U	<0.0001
	eli_100 vs ACO	Mann-Whitney U	0.0001
	–	Kruskal-Wallis	<0.0001
	eli_0 vs MOPSO	Mann-Whitney U	<0.0001
	eli_0 vs ACO	Mann-Whitney U	<0.0001
Average	eli_25 vs MOPSO	Mann-Whitney U	<0.0001
Distance	eli_25 vs ACO	Mann-Whitney U	<0.0001
	eli_50 vs MOPSO	Mann-Whitney U	<0.0001
	eli_50 vs ACO	Mann-Whitney U	<0.0001
	eli_100 vs MOPSO	Mann-Whitney U	<0.0001
	eli_100 vs ACO	Mann-Whitney U	<0.0001

Table F.3: Statistical tests for Project 13 comparing CP-SAT against MOPSO and ACO. *Bold p-values indicate statistical significance ($p < 0.05$).*

Metric	Comparison	Test	p-value
	–	Kruskal-Wallis	<0.0001
	eli_0 vs MOPSO	Mann-Whitney U	<0.0001
	eli_0 vs ACO	Mann-Whitney U	<0.0001
	eli_25 vs MOPSO	Mann-Whitney U	<0.0001
Disagreement	eli_25 vs ACO	Mann-Whitney U	<0.0001
	eli_50 vs MOPSO	Mann-Whitney U	<0.0001
	eli_50 vs ACO	Mann-Whitney U	<0.0001
	eli_100 vs MOPSO	Mann-Whitney U	<0.0001
	eli_100 vs ACO	Mann-Whitney U	<0.0001
	–	Kruskal-Wallis	<0.0001
	eli_0 vs MOPSO	Mann-Whitney U	<0.0001
	eli_0 vs ACO	Mann-Whitney U	<0.0001
	eli_25 vs MOPSO	Mann-Whitney U	<0.0001
Average Distance	eli_25 vs ACO	Mann-Whitney U	<0.0001
	eli_50 vs MOPSO	Mann-Whitney U	<0.0001
	eli_50 vs ACO	Mann-Whitney U	<0.0001
	eli_100 vs MOPSO	Mann-Whitney U	<0.0001
	eli_100 vs ACO	Mann-Whitney U	<0.0001

Vita

Candidate's full name: Tianna-Lee Salmon

University Attended:

- Master's in Computer Science, University of New Brunswick, Fredericton, NB, Canada (2024-2026)
- Bachelor's in Computer Science, University of the West Indies, Kingston, Jamaica (2020-2024)

Publications:

- Journal:
 - **Salmon, T.** & Palma, F. *A Constraint-Based Optimization Approach to the Prioritization of Issues in Scrum*, **The Journal of Systems and Software** (Submitted in November 2025, Under Review).
- Conference:
 - **Salmon, T.**, MacIsaac, D., & Palma, F. (2025, October). *Issue Prioritization in Agile Development through the Lens of Constraint Solving*. In **35th International Conference on Collaborative Advances in Software and ComputINg (CASCON)**, November 2025, Toronto, ON, Canada.

- Workshop/Symposium:

- **Salmon, T.** & Palma, F. (2025). *Issue Prioritization in Agile Development through the Lens of Constraint Solving* [Poster], In **2025 Research Expo, Faculty Computer Science**, Fredericton, NB, Canada.

Conference Presentations:

- *35th* IEEE International Conference on Collaborative Advances in Software and COmputiNg (CASCON), November 2025, Toronto, ON, Canada.