Vrije Universiteit Amsterdam

Universiteit van Amsterdam

Master Thesis

Workflow Planning under Constraints

Author: Radu-Marian Doroș (2749875)

1st supervisor:	Adam Belloum
daily supervisor:	Tim Müller
2nd reader:	Thomas van Binsbergen

A thesis submitted in fulfillment of the requirements for the joint UvA-VU Master of Science degree in Computer Science

February 6, 2024

"I am the master of my fate, I am the captain of my soul"
from Invictus*, by William Ernest Henley*

Abstract

Context. Workflow management systems, crucial in sectors like healthcare and manufacturing, require efficient task coordination while adhering to complex policies. Approaching these challenges through the lens of the Workflow Satisfiability Problem (WSP) emerges as a primary method, attributed to its efficiency and intuitive nature.

Goal. The primary objective of this work is to develop a planning framework for workflow workloads that can handle a broad spectrum of constraints, including both user-independent (UI) and general constraints, while sacrificing as little efficiency as possible.

Method. A prototype workflow planner, written in Rust, was developed to test the feasibility of the approach. The system employs a combination of pattern-based backtracking algorithms and DPLL-style solvers to address different classes of constraints. Benchmarking tools were utilized to validate the implementation and compare its performance with existing solutions.

Results. The system demonstrated effective scalability and flexibility in handling UI constraints. Although its performance in scenarios combining UI with non-UI constraints didn't match that of more specialized solutions, it displayed adequate capability for handling smaller-scale problems.

Conclusions. The study confirms the viability of a generalized approach to workflow satisfiability under a diverse range of constraints, at least for workloads with a low number of steps. Future exploration will focus on studying optimization methods, examining the system's effectiveness in various environments, and applying the methodology to more complex constraint hierarchies.

Contents

1

Introduction

Nowadays workflow management systems play an increasingly critical role across various industries, from healthcare to manufacturing and beyond. A workflow, in the simplest terms, refers to a series of interlinked processes or tasks that are part of a larger operation or project. These workflows can often be represented as Directed Acyclic Graphs (DAGs), where each task is a node in the graph, and the flow from one task to another is depicted by directed edges, emphasizing the sequential and dependent nature of tasks. Workflows are essential for coordinating and executing complex tasks efficiently across various sectors, ensuring that each component of the process seamlessly integrates with the next. They serve as the backbone of many operations, coordinating tasks and resources to ensure efficiency and compliance with legal or organizational policies. However, the very complexity that makes these systems indispensable also makes them difficult to manage. In particular, the often contradictory and conflicting constraints that govern these workflows can make planning an intricate and computationally challenging problem.

1.1 A Motivating Example

To concretize the challenges and complexities inherent in workflow management systems, especially those dealing with sensitive data and multiple stakeholders, let us consider a real-world scenario in the healthcare sector. Consider a consortium of hospitals collaborating on a medical research project aimed at analyzing patient data for performing research in healthcare. In this multi-hospital environment, each institution stores its data locally and is understandably reluctant to freely share this sensitive information due to concerns around privacy and data security.

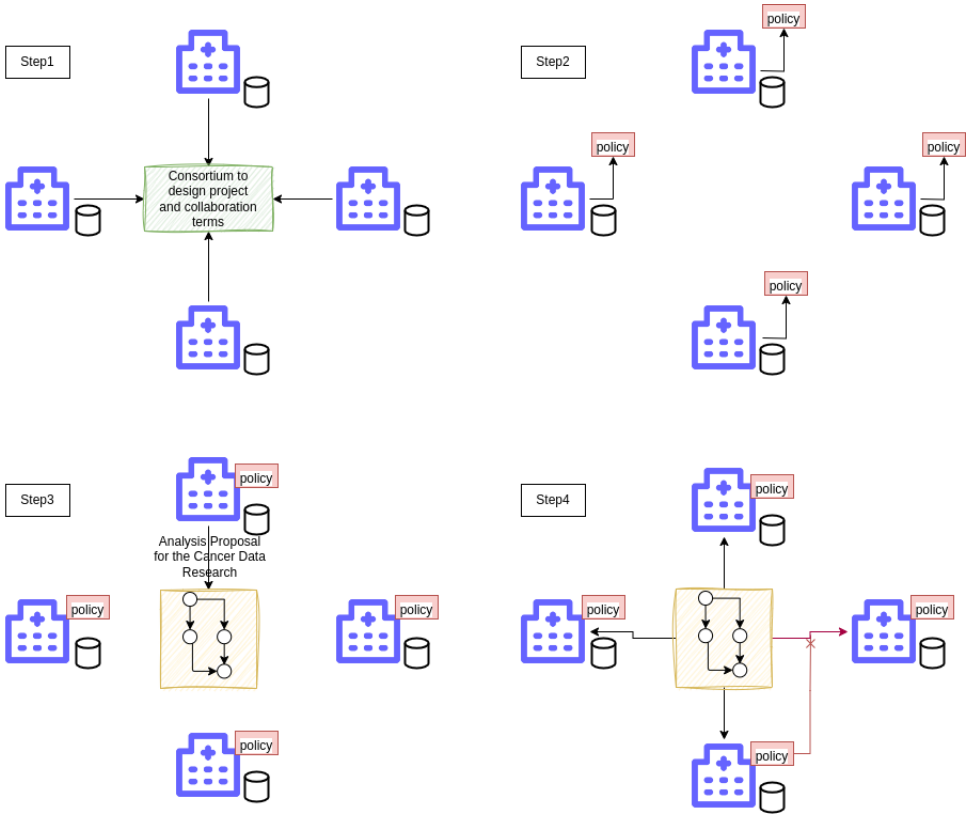

Figure 1.1: The depicted process illustrates a collaborative process among Hospital Partners. The steps are as follows: 1. Hospital Partners come together to discuss and finalize research terms during a convention. 2. A consensus is reached to adopt a system for customized policy checking. 3. One of the partners initiates a request for data sharing and analysis (this is represented as a workflow request). 4. The central planner assigns the workflow to those partners who comply with each hospital's policy rules. In this specific case, the policy rules of one hospital lead to the disapproval of a workflow allocation request from another hospital.

To facilitate the research while adhering to these concerns, each hospital could define its policies for data usage and sharing. These policies serve a dual purpose:

- They act as safeguards, ensuring that any system or research project accessing the data abides by the specific constraints and permission rules set by each hospital.
- They also open the door for broader collaboration. By clearly outlining what is permissible, these policies could allow other systems or research projects to utilize the data—something that would be impossible without such clearly defined rules.

This example underscores the importance of efficient and flexible policy enforcement in workflow systems. Effective management of individual constraints from multiple stakeholders is essential, as is the need for speed and efficiency in the planning process. Figure 1.1

provides a visual representation of this scenario, illustrating how the collaborative workflow among hospital partners might typically be structured. These considerations naturally lead us to the general discussion on the methods and mechanisms of policy enforcement in workflow systems in the next section.

1.2 Policy Enforcement in Workflow Systems

Before delving deeper into policy enforcement, it is important to define how workflows are conceptualized in this work. Workflows are represented as graph structures, with tasks depicted as nodes connected by their dependencies or required order of execution. The process of planning in workflow systems entails assigning these tasks to suitable agents or resources, ensuring that all constraints and dependencies are met.

As highlighted by the preceding example, policy enforcement is a critical feature that enhances the security, compliance, and operational integrity of workflow systems. It acts as the regulatory framework that ensures tasks and activities align with some established rules. The mechanisms for enforcing policies can generally be classified into two primary approaches:

1. Access Control (28): This involves pre-execution checks to determine whether a specific action or access should be allowed according to pre-defined rules and constraints.

2. Continuous Monitoring (Usage Control) (26): In this approach, the system continuously monitors actions and access patterns during execution, intervening only when a violation is detected.

This thesis focuses on the first approach, Access Control. It is worth mentioning that the first approach serves as a foundational building block for the second. In the context of continuous monitoring, access control serves as the initial gatekeeping step, dictating the terms under which subsequent actions unfold.

In practical terms, the importance of Access Control becomes particularly clear when considering dynamic workflows that are susceptible to changes over time during executions. We can imagine a scenario in which a workflow is initially planned to be executed by a specific actor, only to find out during the execution that the actor's actions violate the system's policies. In such cases, the task would need to be reallocated to another actor who is compliant with the rules. The complexity increases when the permissibility of task allocation is contingent on preceding tasks, which may necessitate a wholesale reallocation

of multiple tasks within the workflow. In such contexts, the mechanisms of Access Control are crucial. By ensuring at a planning phase that task allocations comply with policies, we mitigate the risk of extensive reallocations and the computational costs associated with them.

While Access Control serves as a strong initial gatekeeper for policy enforcement, it is important to note that there might be scenarios where a policy that was evaluated as permissible during the planning stage could become impermissible by the time of execution. This discrepancy may arise due to internal state changes in the policy evaluation mechanisms between the planning and execution phases. Although such occurrences are arguably rare given the relatively short time frame between planning and execution, they nonetheless could happen.

1.3 Problem Statement & Objectives

The problem this thesis aims to address is how to achieve efficient planning in workflow systems under general constraints, creating an Access Control framework. Specifically, the constraints under which planning occurs can incorporate Access Control rules, thereby transforming the planning process into an inherent form of policy enforcement. The research will concentrate on the following objectives:

- **Tailoring Planning Approaches to Complex Constraints:** This aspect aims to explore and develop planning methods that are both efficient and capable of accommodating a wide array of constraints, including but not limited to GDPR purpose/permissions mechanics described in Basin et al. (2) and role-based access control (RBAC) INCITS (18).

- **Benchmarking and Analysis:** The goal here is to establish appropriate evaluation metrics and methodologies for assessing the effectiveness of the proposed planning methods. This involves running benchmark experiments against existing workflow planning algorithms to provide a comparative analysis.

- **Applicability Framework:** This component focuses on creating a conceptual framework that guides organizations in adapting their specific operational rules and requirements into the proposed problem formulation, thereby ensuring the general applicability of our approach.

1.4 Approach

Our approach primarily aims to identify a feasible plan that meets all constraints, known as a 'satisfiable' plan. This term refers to a solution where every task in the workflow is assigned in a manner that aligns with all the specified operational and policy requirements. To achieve this goal, we categorize constraints based on their type and then apply one of two main strategies:

1. For constraints of a specific type that lend themselves to optimization, we employ pattern-based planning techniques. As explained by Karapetyan et al. (20), these techniques involve constraints that yield the same results for any allocations of a partition of the workflow nodes. This approach leverages the uniformity in constraint behavior to facilitate more efficient workflow planning.

2. For constraints that diverge from this predictable pattern, we resort to the naive plan enumeration approach. In cases where constraints are more complex or irregular, lacking a discernible pattern, this method involves systematically exploring all possible task assignments within the workflow. Within this category, we also explore optimization techniques and heuristics to improve efficiency.

1.5 Summary of the Following Chapters

In Chapter 2, we explore the Background notions, providing the necessary theoretical groundwork for the thesis. Chapter 3 focuses on reviewing existing literature and studies relevant to our research field, emphasizing the differences in approach and methodology from our selected strategies. Chapter 4 outlines our approach to addressing the problem statement. Chapter 5 presents the experimental results, demonstrating the effectiveness of our framework. Finally, Chapter 6 concludes the thesis and discusses future avenues for research.

2

Background

As mentioned in the introductory Chapter 1, the scope of the thesis is to find a general and practical solution to the problem of planning under constraints. This chapter provides a background that lays the foundation of the solution we introduce in the subsequent chapters. We review the formulation of the workflow satisfiability problem, introduced in the seminal work of Wang and Li (31) and then further analyzed and explored by Karapetyan et al. (20) and Karapetyan and Gutin (19). The later articles provide us with critical building blocks for our designed system. These works offer critical insights and foundational elements for our system's design, detailed in Chapter 4, where we integrate pattern-based backtracking techniques, discussed in the following sections, with naive methods for handling more general cases.

2.1 Workflow Satisfiability Problem (WSP)

In addressing the challenge of planning under constraints, the concept of the Workflow Satisfiability Problem (WSP) emerges as a central theme in research (9). Recognized as a specific class of satisfiability problem, WSP concerns itself with the allocation of workflow tasks to users in a manner that adheres to predefined constraints. This section delves into the foundational definitions and theoretical constructs of WSP as established by the seminal works of Wang and Li (31) and further elaborated by Karapetyan et al. (20). These definitions form the cornerstone of our understanding of WSP and provide the necessary terminological and conceptual framework for the subsequent discourse

Definitions The Workflow Satisfiability Problem (WSP) can be formally defined as follows: Given a set of workflow steps S, a set of users U, and a set of constraints

$C = \{c_1, c_2, ..., c_l\}$, where each c_i represents an evaluation function applied over plans $\pi : S \rightharpoonup U$ (i.e. partial functions from S to U), constraints $\mathcal{C} : U^S \to \{0, 1, \omega\}$ are defined as:

$$\mathcal{C}(\pi) = \begin{cases} 0 & \text{if } \pi \text{ violates the constraint,} \\ 1 & \text{if } \pi \text{ satisfies the constraint,} \\ \omega & \text{if the scope of } \pi \text{ is incomplete for full evaluation.} \end{cases}$$

A plan π satisfies all constraints if and only if every constraint c_i evaluates to true (equivalently, $c_1(\pi) = ... = c_l(\pi) = 1$). It is assumed that each constraint c_i can be evaluated in polynomial time with respect to the sizes of S and U, ensuring computational feasibility. Constraints may not require π to be fully defined over all elements of S and can evaluate partially defined plans. While authorization sets $A : S \to 2^U$ are introduced in the model, representing constraints that act as simplified allowlists for individual steps, they are not essential for all cases of the WSP. However, in many instances where constraints C are more limited, these authorization sets become particularly useful. Therefore, an instance of a WSP problem is effectively defined as a quadruple (S, U, A, C), where the role of A varies depending on the nature and restrictiveness of the constraints in C.

It is important to understand why constraints should be able to handle partial plans. Mostly constraints are assumed to concentrate on a portion of a plan (a scope). Once that scope is defined (users planned to be assigned to the scope's steps), the constraint will not necessarily require other parts of the plan and can be evaluated already. This comes in handy during the planning phase when plans are built incrementally and scoped constraints can reject partial plans.

Another aspect noteworthy to highlight is that the Workflow Satisfiability Problem (WSP) resembles a generalized version of the Boolean satisfiability problem (SAT) when its application is confined to scenarios in which workflow nodes are assignable to only two potential users. In this context, SAT involves assigning Boolean values to a set of variables in such a way that they satisfy a predetermined set of clauses. For a comprehensive overview of the current algorithmic approaches to this problem, the work in Vizel et al. (29) serves as an excellent reference. In Chapter 3, we delve into a variety of methods, insights, and techniques derived from SAT solvers, exploring their relevance and applicability to the Workflow Satisfiability Problem.

Fixed Parameter Tractability (FPT) FPT algorithms are characterized by a computational complexity that is polynomial with respect to the size of the input data N and exponential solely in terms of a fixed parameter k. This is generally expressed as $f(k) \cdot O(N^c)$, where $f(k)$ is a function of the parameter k, and c is a constant. In the case of WSP, the complexity of planning algorithms depends on two variables: the number of steps k and the number of users N. When the number of steps k is bounded and relatively small, FPT approaches become highly practical and effective, even as the number of users N increases. This scenario is common in many real-life applications, making FPT methodologies particularly suited for WSP. Consequently, it is important to determine the scenarios in which FPT is possible within the context of WSP.

As subsequent discussions will show, the WSP can support FPT algorithms under specific constraints, whereas the introduction of more general constraints negates this possibility.

2.1.1 Authorization Sets

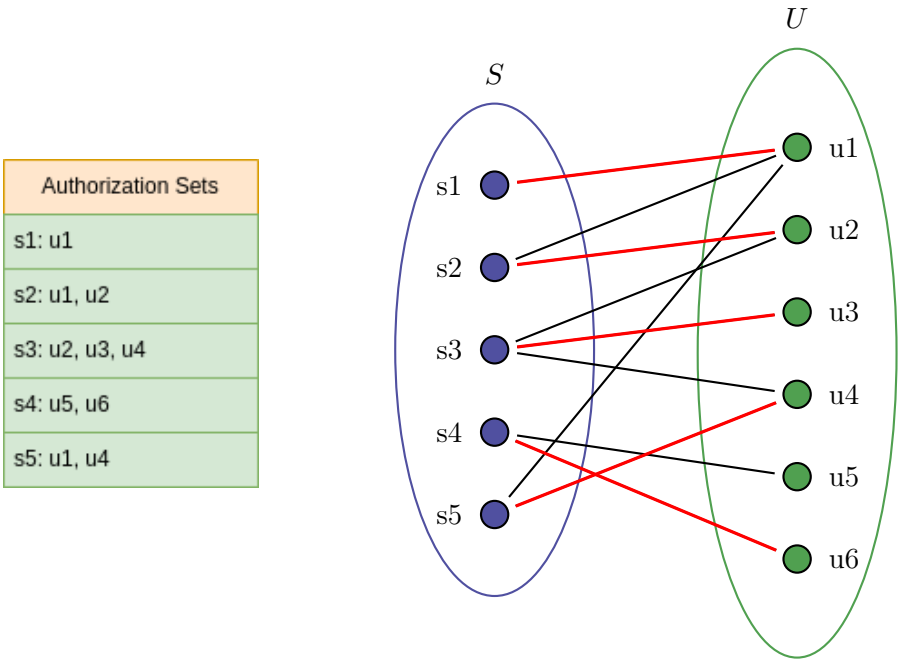

Figure 2.1: Authorization Sets: planning as Max Bipartite Graph Matching. The edges in the graph represent the authorization table's permissions, where each edge connects a user to a permissible step. The red edges specifically denote the chosen allocations in the bipartite matching, indicating the users assigned to each step in the final plan. Note that each user can be allocated to only one step.

Before delving into the pattern-based method, it is useful to consider a foundational

building block: the concept of authorization sets for each workflow step. In this context, the task of assigning users to specific workflow steps can be formulated as a maximum bipartite matching problem, if users can be allocated to a workflow step at most once. Specifically, a bipartite graph is generated with workflow steps and users as the two disjoint sets of vertices. Edges between these sets are determined by membership in the respective authorization sets. A valid plan in this scenario corresponds to a bipartite matching that covers all workflow steps, as illustrated in Figure 2.1.

The maximum bipartite graph problem is known to be solvable in $O(nm)$ with the Kuhn-Munkres (21) algorithm where n is the number of vertices in one set and m the number of edges in the bipartite graph. Hopcroft-Karp-Karzanov (17) is a more efficient bipartite graph matching algorithm that runs in $O(\sqrt{n}m)$.

2.1.2 Pattern-Based Algorithms

User Independent Constraints Another concept introduced in Gutin (14) is the concept of user-independent constraints. These constraints have the following properties: Any labeling of the steps with users $\pi : S \to U$ that satisfies the user-independent constraint has the property that we can replace all instances of any user with another user. That is, given a remapping $\tau : U \to U$, we have the composed function $\tau \circ \pi : S \to U$ also satisfying the UI constraints.

In this context, the integration of both the authorization sets A and the constraints C that are user-independent proves to be highly advantageous. User-independent constraints cannot target users for steps specifically, a role that is effectively fulfilled by authorization sets that dictate the banning or allowance of users for individual steps. The need for having them both also provides a solid rationale for the subsequent chapters of this work, which focus on the design of a more general constraint framework.

Patterns Another important related concept is the Pattern. Patterns are partitions of the set of steps S into disjoint subsets $S_1, \ldots, S_p$ such that $\bigcup_{i=1}^{p} S_i = P$ and $S_i \cap S_j = \emptyset$ for all $i \neq j$. These subsets are referred to as blocks. For convenience, each block is assigned to a user, while different blocks are assigned to different users. Importantly, if multiple blocks were assigned the same user, an equivalent pattern could be formed by merging those blocks.

Given a user assigning plan function $\pi : S_i \to U$, in the context of User-Independent (UI) constraints, if π satisfies these constraints, then the plan will also be valid under any remapping $\tau : U \to U$. In other words, $\tau \circ \pi : P \to U$ will also satisfy the UI

constraints. This property allows us to evaluate these constraints using arbitrary user assignments. Consequently, the planning problem under UI constraints simplifies finding a valid pattern, making it essentially a partition enumeration problem. The number of partitions of k elements is known to be defined as Bell's number B_k. In Karapetyan et al. (20) B_k is approximated as $O(2^{k \log_2 k}) = O(k^k)$.

Having examined the definitions of user-independent constraints, we will now turn our attention to the specific constraints outlined in Karapetyan et al. (20) as examples:

- Binding of Duty (BoD): This constraint is satisfied when, within a two-step scope s, t, the user assigned to step s is the same as the user assigned to step t, denoted as $\pi(s) = \pi(t)$. BoD could be useful in scenarios where continuity or consistency of task handling is important. For instance, in legal or financial document processing, the same individual might be required to handle multiple steps to ensure consistency and accountability.

- Separation of Duty (SoD): This constraint is satisfied when, within a two-step scope s, t, the assigned user of step s is different than the assigned user of t, i.e $\pi(s) \neq \pi(t)$. SoD constraints can be useful in preventing conflicts of interest and ensuring checks and balances in financial systems.

- At-least-k and at-most-k: The at-least-k constraint is satisfied if for a scope T, the number of unique users assigned to the scope $|\pi(T)| \geq k$. The at-most-k constraint is defined symmetrically. These constraints are useful in situations requiring diversity or limiting concentration in task allocation. For example, in project management, ensuring that a task is seen by a minimum number of different eyes (At-least-k) or preventing workload overload on individuals (At-most-k).

It should be noted that the Binding of Duty constraint is given mostly only as an example since using it trivially simplifies the problem. With this constraint, we can simply transform the problem to a simpler one by merging the connected components generated by the BoD steps together and transforming the remaining constraints to refer to the new merged step.

Wang and Li (31) notes that in the case the constraints are only formed of the Separation of Duty constraint, the satisfiability problem is equivalent to a Graph k-colorability problem (22). In the graph k-colorability problem, a graph is said to be k-colorable if we can assign at most k colors to vertices to the vertices of the graph such that all vertices have distinct colors from their adjacent vertices. Furthermore, they show that these types of constraints admit FPT type complexity of $O(k^{k+1}n)$.

2.1.3 Combining Pattern Enumeration and Bipartite Matching

The two methods discussed Section 2.1.1 and Section 2.1.2 are effectively integrated into a two-step process in Karapetyan et al. (20).

This process involves:

1. Enumerating the partitions that fulfill the user-independent constraints.
2. Mapping the patterns to specific users according to the authorization sets using maximum bipartite matching.

The combined approach is concisely represented in pseudo-code as seen in Listing 1.

Algorithm 1 UI Planning

Require: $G,\ C,\ A$
Ensure: $Plan(G)$ **satisfies** $C,\ A$
1: $generator \leftarrow Partition\_Generator(G)$
2: **while** $part \leftarrow generator.next()$ **do**
3: **if** $part$ **satisfies** C **and** $match \leftarrow find\_bipartite\_matching(part, A)$ **then**
4: **return** $match$
5: **end if**
6: **end while**
7: **return** $\emptyset$

In this process, we explore through B_k partitions in the worst case, each requiring verification via determining a maximal bipartite match. It is important to note that this analysis does not take into account the complexity associated with evaluating constraints, nor does it consider any state-managing overhead complexity of the *partition.next*() function. Assuming the use of an $O(n \cdot k)$ bipartite matching algorithm, the complexity becomes $O(B_k \cdot n \cdot k)$, which is Fixed-Parameter Tractable (FPT) because B_k, while exponential, does not depend on n.

2.1.4 Optimizations for Pattern Enumeration

This subsection summarizes essential optimizations highlighted in the study by Karapetyan et al. (20), expanding upon the basic methods of pattern enumeration and bipartite matching as discussed in Listing 1. These enhancements are particularly significant for their ability to increase pruning, thereby enabling the algorithm to bypass the need to evaluate all potential partitions in many cases.

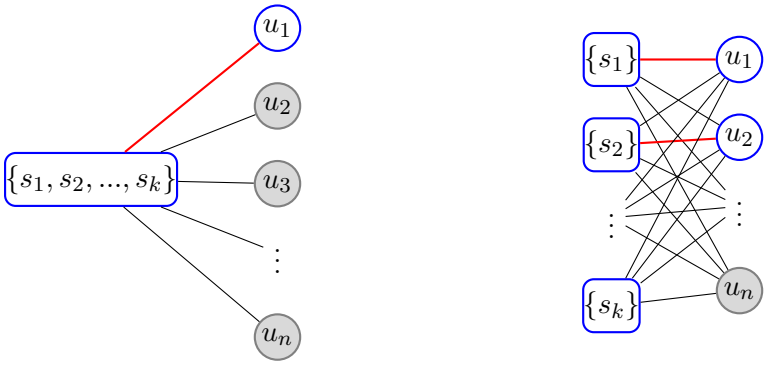

Figure 2.2: This diagram presents the two extreme cases in the partitioning of a set containing steps $s_1..s_k$: one where each element forms a singleton set and the other where all elements are grouped into a single partition. While these are the extremes, it is imperative to acknowledge the entire spectrum of partition possibilities that exist between these two cases. The number of average parts in all partitions of a set of k elements is described in Odlyzko and Richmond (25) as $O(\frac{k}{logk})$. For our purposes, we will simply use k as an approximation of it, since it simplifies calculations while also not significantly altering the results complexity-wise.

2.1.4.1 Incremental eligible partition generation

The first property leveraged by Karapetyan et al. for optimization is the monotonic property of the pattern function P: if a pattern $P(S')$ is valid (satisfies all constraints) for some $S' \subseteq S$, then all sub-patterns $P(S'')$ are also valid for every $S'' \subseteq S'$. In other words, when a constraint evaluates positively for a scope, larger scopes containing it will not change their positive evaluation. This property allows for an incremental evaluation of constraints, enabling us to efficiently enumerate and assess subset partitions.

This is very convenient also because generating full partitions requires an incremental approach already, where we find sub-partitions before completing an integral one. In Listing 1 we can think of the $Partition\_Generator :: next()$ method to now return partial partitions.

This approach also offers flexibility in the order in which the partial partition generator adds nodes to the configuration. Specifically, we can prioritize nodes that frequently appear in the constraints, thereby increasing the likelihood of early pruning of invalid partitions.

2.1.4.2 Incremental bipartite matching

Similar to the previous observation from Section 2.1.4.1, that evaluations remain consistent in the presence of extending subsets, the matching problem has similar properties as described in Karapetyan et al. (20). Suppose we have an existing partial partition $P = \{b_1, b_2, ..., b_k\}$ that is eligible. We aim to add new nodes to the matching without

having to rerun the entire matching algorithm; instead, our goal is to update the existing solution. When adding the new node, we distinguish between two options, as also illustrated in Figure 2.5:

- We extend some block b_i to include s. This affects the bipartite graph the following way: we must remove all edges that are connecting b_i to all users that are not authorized for s. Here again, we distinguish between 2 cases:
  1. If the old matching did not contain an edge that is removed when adding s, we can keep the old matching.
  2. Otherwise, we remove the edge from the matching and we need to find a new M-augmenting path starting in the b_i. In this context, an M-augmenting path is a path that starts and ends with unmatched vertices and alternates between edges that are not in the matching. Given that we already have a full matching except for the removed node, the goal is to find an M-augmenting path that will include this unmatched node, thereby restoring the full matching.
- s becomes a new singleton block. In this case, we can keep the previous matching and we need to find a new M-augmenting path starting in the s step.

The operation of finding an augmented path starting from a block node requires performing a DFS of complexity O(V + E). Besides this, also graph update operations are required to be done. We need to allow both the deletion and addition of nodes and edges.

2.1.4.3 Prioritizing nodes that are inside predicates

An improvement can be made that increases the pruning power. Since pruning occurs in cases when evaluations of the attributes refute some subset, we can reorder the planning algorithm to focus on the nodes that are more present in the predicates, therefore heuristically increasing the chances that pruning will occur. This can be applied when expanding the current subset of nodes (picking the next step node to be added).

In their approach to node selection, Karapetyan et al. (20) utilize the transparency of constraints. They assign weights to each candidate node, with these weights being indicative of the likelihood of a node triggering a pruning action within the algorithm. This weighting system is rooted in the probability that a given constraint, upon being fulfilled by the node, will lead to pruning. Nevertheless, it should be noted that this method of prioritization may not be feasible for constraints that are less transparent or more complex, where such direct assignment of probabilities is not straightforward.

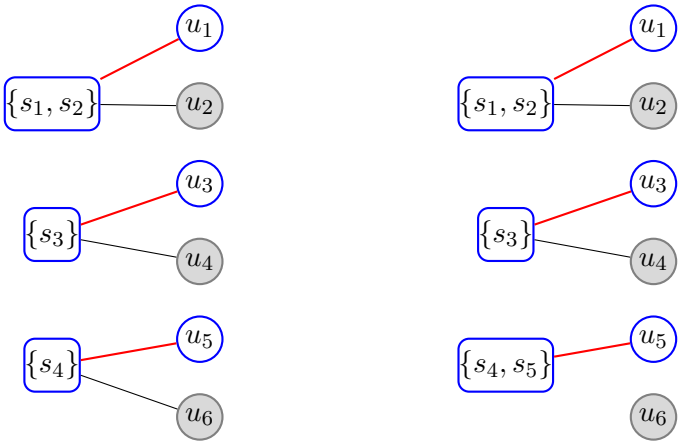

Figure 2.3: For all cases: Every edge represents a potential assignment, and red edges indicate actual assignment. Case 1: when adding the new node s_5 in the block of s_4, remove edges to users not authorized for s_5. In this case, the previous assignment for the block is still valid.

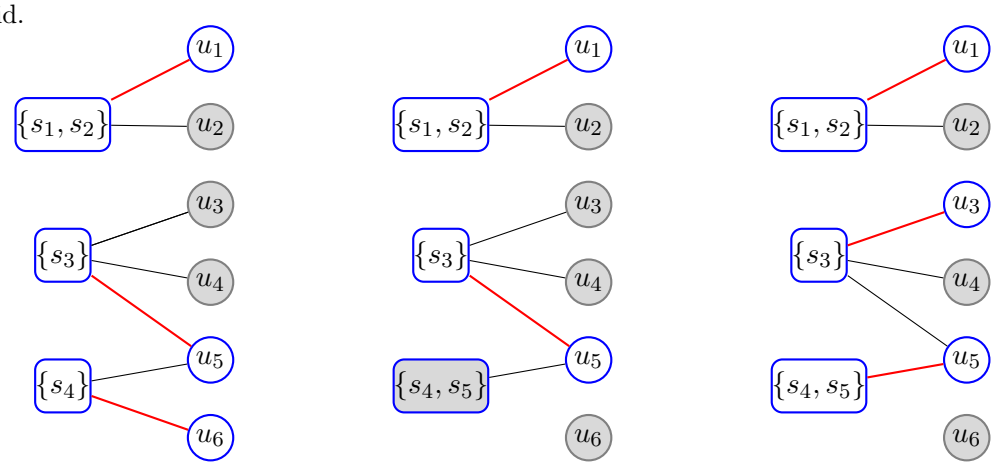

Figure 2.4: Case 2: when adding the new node s_5 in block of s_4 and remove edges to users not authorized for s_5, u_6. In this case, the previous assignment for the block becomes invalid and we need to find an M-augmenting path starting from s_4, s_5. Paths to u_3 and u_4 exist and we repair s_3 with u_3 to obtain the new full matching.

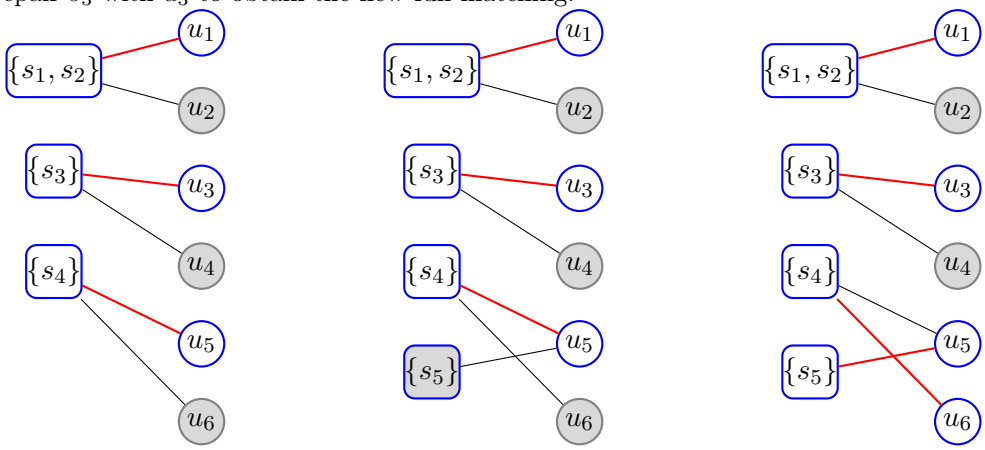

Figure 2.5: Case 3: when adding the new node s_5 and creating a singleton block. We now need to pair a new user and we need to find an M-augumenting path. We find one to u_6 and we repair s_4 with u_6, allowing s_5 to be paired with u_5.

2.1.5 Handling more general constraints

To address the broad scope of UI constraints, Karapetyan and Gutin (19) have demonstrated a method for converting general constraints into a format suitable for UI problem-solving. This involves reformulating these constraints into distinct branches of valid authorization sets and then verifying the feasibility of these sets by assessing if they allow for appropriate user-pattern matches. While this approach proves that general constraints can indeed be adapted to this UI framework, the inherent complexity of some constraints could result in extensive branching, resulting in significant computational overhead.

Karapetyan and Gutin (19) highlight a range of constraints that are not inherently compliant with User-Independent (UI) criteria. These include:

- Super-User At-Least Constraint (SUAL): This specifies that within a certain scope T in S, given a parameter h and a group of super users X, if the total number of users allocated to T falls below h, then these users must all belong to the super user set X. SUAL can be particularly useful in high-security or sensitive environments where certain tasks require a minimum number of qualified individuals (super users) to handle them. If the number of individuals available falls below a certain threshold, this constraint ensures that only those with specific, higher-level clearance or qualifications (the super users) are assigned to these tasks.

- Wang-Li (WL) (31): This constraint mandates that within a specific scope T in S, involving d distinct sets of departmental users $U_1, U_2, ..., U_d$, all assigned users in T should come from the same departmental set U_i. This constraint is particularly relevant in scenarios where conflict of interest policies must be enforced. It ensures that tasks within a particular scope are handled by users from a single, distinct department or group, thereby preventing the crossover of information or influence between different departments.

- Assignment-Dependent Authorisation (ADA): It states that in scope with two steps, s_1 and s_2, and two sets of users, U_1 and U_2, if step s_1 is assigned to a user from U_1, then step s_2 must be assigned to a user from U_2. ADA is applicable in workflows where the assignment of one task influences the suitability or eligibility of users for subsequent tasks. One example where ADA is useful might be in situations where the skill level or performance history of a user assigned to one task influences the assignment of subsequent tasks. This can ensure that more experienced or senior

users oversee or follow up on tasks handled by less experienced or historically less reliable users.

While the concept holds merit in its exploration of constraint nature and demonstrates a tangible method for converting constraints within branching UI frameworks, the idea suffers from the fact that it is a "manual" approach. The process of translating a general constraint into an alternate formulation necessitates the creation of a unique authorization set for each potential evaluation branch of the constraint predicate. It is imperative to establish methods for the automatic transformation of general constraints into UI constraint branches, as this is a critical condition for the practicality of the method.

2.2 Applications Discussion

Next, we briefly look at two cases where the WSP formulations can be applied, even when only limiting them to the authorization sets structure: handling of GDPR purposes & permissions and role-based access control (RBAC).

2.2.1 GDPR purposes & permissions

The work by Basin et al. (2) on GDPR compliance underscores the significance of aligning data collection with specific, consented purposes. They advocate for a methodology where each business process is distinctly tied to its purpose, ensuring that data handling adheres strictly to the GDPR's requirement for purpose-specific consent. This approach is compatible with the presented pattern-based formulations. In this framework, we apply this principle by identifying users whose permissions match the intended purposes of each workflow step. This is achieved by constructing authorization sets for each step, where the set contains users with permissions that align with that step's purpose as illustrated in Figure 2.6.

2.2.2 RBAC

Likewise, this approach can also be applied to translate formulations within Role-Based Access Control (RBAC) (18). While Wang and Li (31) focus on the R2BAC model, Gutin (14) asserts that standard RBAC configurations are compatible with their proposed formulation.

RBAC (18) is a widely used framework for managing user permissions within a system. In RBAC, access rights are not assigned to individual users directly; instead, they are

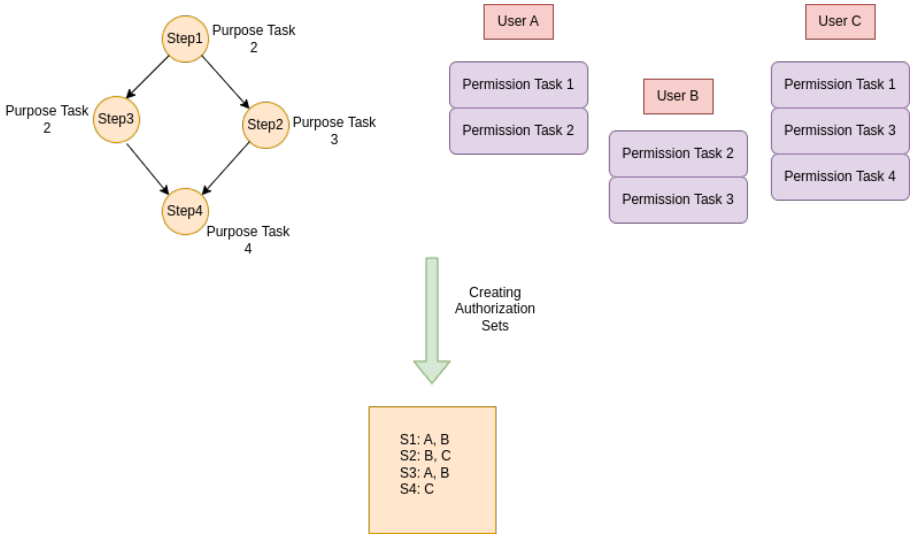

Figure 2.6: Translating GDPR purpose/permission labelling, as proposed by Basin et al. (2), into the authorization set formulations. Compiling these sets is as simple as identifying for each step's purposes the set of users/sites that have gathered permissions for that task.

associated with roles, and users are then assigned to these roles. Each role encompasses a specific set of permissions that pertain to the user's authority and responsibility within the organization.

In RBAC, since permissions are role-based rather than user-specific, it aligns seamlessly with the concept of user-independent constraints in workflow satisfiability problems. This is because UI constraints are designed to be applicable irrespective of the specific user identities, focusing instead on the roles or functions that the users fulfill. Similarly, authorization sets in RBAC can be easily conceptualized: each role can be seen as an authorization set with its own unique set of allowable actions within the system. This compatibility allows for an effective application of RBAC principles in workflows governed by UI constraints.

2.3 Chapter Conclusions

This chapter has laid the necessary groundwork for introducing our planning system in Chapter 4. We summarized the literature on User-Independent (UI) concepts and looked at a Fixed-Parameter Tractable (FPT) algorithm that handles UI constraints, providing us with insights into the planning problem under constraints. In the following chapters, we will leverage these foundational concepts to design our system.

3

Related Work

Before delving into our approach and implementation, it is imperative to review related works and motivate the rationale behind our diverging path. In Section 3.1 we look at an attempt at creating a hierarchy of constraint expressivity due to dos Santos and Ranise (9). Then we check briefly the ways optimization is done in non-constrained workflow planning. And last, we briefly check some relevant literature on solver.

3.1 Generalization to constraint hierarchy

The comprehensive survey in dos Santos and Ranise (9) presents various constraints related to the Workflow Satisfiability Problem (WSP) and its extensions. We observe distinct classes of constraints in their conceptual diagram 3.1:

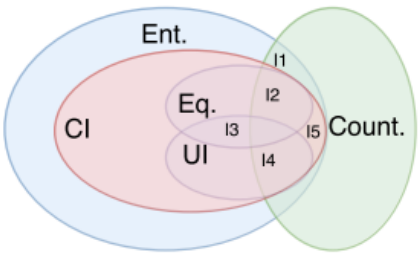

Figure 3.1: Constraint Hierarchy, shown in dos Santos and Ranise (9)

- **Class-Independent Constraints (CI)** introduced by Crampton et al. (5): These constraints establish equivalence classes of users, allowing for the interchangeable assignment of users within the same class. An example is the formation of organiza-

tional departments, where any member of a department is a suitable candidate for assignment.

- **Counting Constraints (Count)**: These constraints evaluate the frequency of a user's assignments to tasks within specific scopes.
- **Entailment Constraints (Ent)**: A generalization over the class-independent constraints, these entailment constraints facilitate more complex binary relations between users beyond the equivalence relations typical in class-independent scenarios.
- **User-Independent Constraints (UI)** are the constraint types we already mentioned in Chapter 2.
- **Equivalence Constraints (Eq):** Constraints based on equivalence relations between the assignees of the workflow nodes.

The analysis of constraint classes, particularly those related to user-independent constraints, presents a noteworthy area of study. However, the practicality and boundaries of applying more general constraints, such as Entailment Constraints, within an FPT framework remain to be thoroughly explored.

3.2 Workflow Planning in General (no constraints)

The realm of workflow planning, in the usual case, when the formulation is devoid of constraints, has a substantial body of research. Notably, Grandl et al. (13) offer an approach focusing on the identification and scheduling of "troublesome" tasks—those expected to have extended durations. Their strategy revolves around the strategic scheduling of related tasks (e.g., parents, siblings, children) in conjunction with these troublesome tasks. However, an obvious and critical limitation of this approach is its lack of consideration for constraints, a fundamental requirement in our context.

Furthermore, it's important to highlight that their methodology is centered around optimization - seeking the most efficient plan, rather than simply identifying the first plan.

3.3 Solvers

Solvers can be conceptualized as computational entities designed to resolve specified problems by transforming them into mathematical formulations, such as systems of equations. We can borrow some ideas from them, like optimizations, and port them to our problem.

We next look at some options, that are possible to be applied to perform workflow planning under constraints.

We discuss briefly some concepts and ideas from the fields of general-purpose solvers as well as some WSP extended work.

3.3.1 SAT with DPLL and its Heuristics

Boolean satisfiability (SAT) problem refers to the problem of assigning values to a set of boolean variables that are subjected to some constraints. Constraints are limited to propositional calculus. However, some of the developed techniques can be used as enhancements and future work to the problems we look at in Chapter 4.

The optimizations used in Boolean satisfiability, as pioneered in Davis et al. (7), that target the order in which steps are explored by SAT solvers can offer valuable insights for WSP solvers:

- **Unit Propagation**: This technique involves monitoring nodes within a predicate to identify when the final node is explored, then having solvers focus on the last node of a constraint for enhanced pruning chances (the solver will now have access to and will enumerate the complete scope of the predicate).
- **Pure Literal Elimination**: The identification of literals appearing both in their original and negated forms allows for arbitrary value assignments.
- **Backjumping**: An efficiency-improving technique where the solver, upon encountering a failure at a certain variable, can skip retrying previous values and jump back to an earlier variable for reassessment.

3.3.2 Satisfiability-Modulo-Theory (SMT)

Expanding on the concepts of Boolean satisfiability, as discussed in De Moura and Bjørner (8), the Solvers-Modulo-Theory (SMT) approach integrates theories and solvers for more comprehensive solutions. Unlike SAT, which only deals with propositional logic, SMT can handle formulas with rich theories like arithmetic, bit-vectors, arrays, and uninterpreted functions, making it more suitable for a wide range of practical applications. Despite this, finding such a universally applicable theory that caters to our case remains a work in progress. Our objective is to maintain maximal flexibility for users (constraint checker implementers), allowing them to utilize any language of their choice.

While Karapetyan and Gutin (19) showcases the application of OR Tools' (12) Pseudo-boolean formulations, our pursuit is to achieve an even higher degree of generality.

3.4 Summary

Firstly, we see that the mentioned approaches suffer from a lack of generality and customization ease. However, this is not the case with the WSP approach. In this context, we have the flexibility to examine and tailor our implementation, as well as adapt it to broader generalizations, as will be demonstrated in Chapter 4. It is important to note, however, that our exploration is constrained by time limitations. This necessitates a focused and efficient approach to our research, thereby precluding an in-depth investigation into all the potentially interesting aspects of related work.

4

Design

4.1 Introduction

This chapter focuses on the design and implementation of a workflow planner capable of handling general constraints in complex environments, as exemplified by the multi-hospital scenario described in Chapter 1. The planner's architecture and functionalities are tailored to address the challenges of sensitive data handling and to facilitate effective collaboration amongst stakeholders sharing data for research purposes.

To account for the need for the generality of the constraints, these constraints are treated as functions mapping plans, which are themselves functions $\pi : S \to U$, to a Boolean value. Predicates then, have the signature: $p : U^S \to \{0, 1, \omega\}$. Within this formulation, 0 denotes a situation where the plan conflicts with the given constraint. 1 is indicative of a plan that aligns with or satisfies the constraint. The symbol ω is employed to denote instances where the determination of the plan's compliance is uncertain, potentially due to incomplete information in scenarios where the plan has not yet assigned all workflow nodes, which are within the constraint's scope, to a user.

4.2 System Architecture

4.2.1 High-Level Design

In response to the scenario of a consortium of hospitals, the system is designed for flexibility and security. Each hospital (site) can define its constraints through HTTP(s) request handlers, mirroring the need for each hospital to set its data usage policies while collaborating securely. The high-level design of the system is illustrated in Figure 4.1.

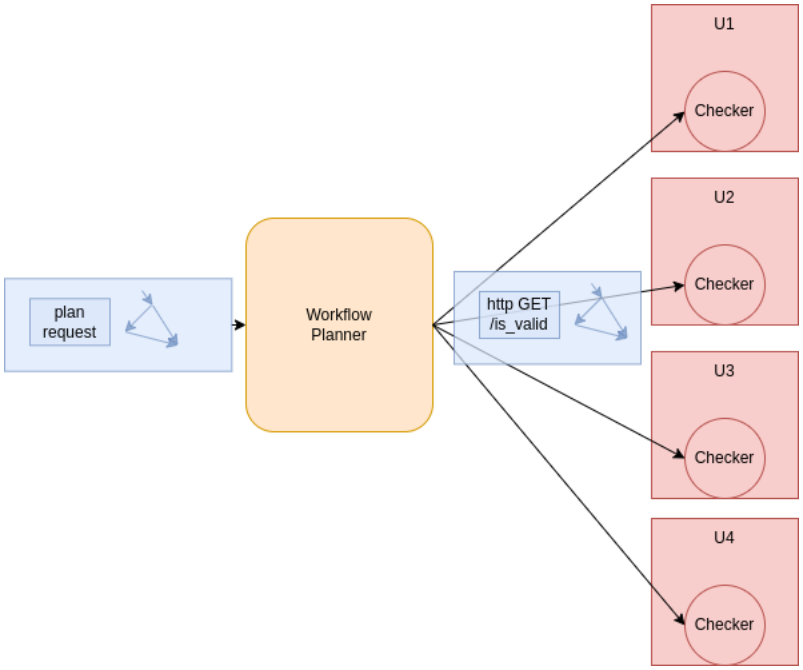

Figure 4.1: High Level Design

An important aspect of the system's architecture is its technological agnosticism. By making a simple HTTP server interface pivotal to the interaction mechanism, the system allows each participating hospital to choose its preferred programming language for defining constraints. This approach effectively circumvents the limitations often imposed by conventional constraint-solving methodologies, which typically mandate adherence to specific technologies or design languages. Consequently, hospitals can leverage their existing technical expertise and resources without being constrained by the need to adapt to a new or unfamiliar technology framework. Moreover, the incorporation of black-box predicates, functioning as opaque HTTP request handlers, enables hospitals to implement complex decision-making processes internally, without exposing sensitive logic or data structures.

Dynamic Policy Adaptation Based on Logged Historical Data A natural extension of the system is a dynamic policy adaptation mechanism, which utilizes a history of requests and plan executions. These actions are logged in an operations log, observable by all participating sites. This log serves as a foundation for sites to adapt and update their policy functions. For instance, if a site identifies a breach or non-compliance with an agreed protocol by another participant, it can modify its policies accordingly. Subsequent workflow requests from the violating party could be subjected to stricter scrutiny or even

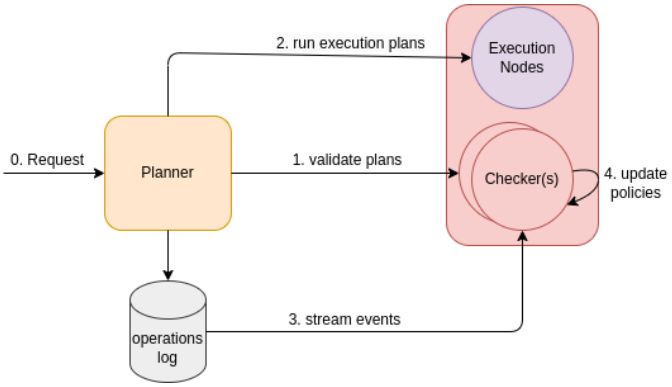

Figure 4.2: Illustration of the operations log, enabling checkers to observe events and adapt their policy constraints accordingly.

rejection, particularly in cases involving unsupervised collaborations.

While the formulation and design of these adaptation mechanisms and rules can present a complex challenge and fall outside of the scope of this work, the availability of an operations log offers the opportunity for checkers to monitor and respond to activities within the system. This monitoring capability enables checkers to adapt their policies as needed, aligning with our design's emphasis on flexibility. Figure 4.2 provides a visual illustration of the operations log.

Checker-Eventlog Synchronization The challenge of synchronizing constraint checkers with the event log parallels the issue of managing stale followers in database replication contexts. A variety of synchronization methods, discussed in sources like Wang et al. (30), offer several potential solutions. The selection of a synchronization strategy depends largely on the acceptable delay in the update times of the checkers. In our discussed scenario, considering that a slight delay in checkers updating their state may not be critically detrimental, we can opt for a synchronization approach that provides 'good enough' timeliness.

4.2.1.1 Implementing Scoping Mechanisms

Checkers in our system have the flexibility to formulate their predicate constraints regardless of complexity. Nonetheless, as highlighted in the study by Wang and Li (31), the complexity of the general problem (with general constraints), not accounting for the time complexity of constraint evaluation, grows to $O(n^k)$. Our method involves segregating constraints into different classes and employing distinct algorithms for solving their respective subproblems, which, in our context, combines the pattern backtracking algorithm

discussed in the background Section 2.1.2 with a DPLL-style solver (7) for more general constraints.

To effectively categorize the constraints, we may consider two primary approaches:

- **Automated Detection of Constraint Classes:** Utilizing language analysis techniques to automatically identify the constraint class defined by each checker. One potential approach lies in creating reliable tests that can accurately identify constraint classes. For example, a test suite could be created to verify if the user independence property is maintained upon constraint creation. This suite can be designed to ascertain whether the constraint evaluations remain consistent across various permutations of user assignments, in line with the defining property of user independence.

- **User-Defined Constraint Labeling:** Here, users themselves label their constraints according to the type, based on the understanding that they will strive for efficient constraint planning as collaborative contributors.

Based on the literature we summarized in Chapter 2, we identify three distinguishable classes of constraints: authorization sets, user-independent, and non-user-independent constraints.

Scoping Operations: In line with the conceptual frameworks presented in Section 2.1.2, the implementation of scoping mechanisms is a key consideration for our system. The currently described high-level design, however, does not enable checkers to directly communicate their operational scopes to the planner. Recognizing the potential for optimization through scoping, our design should handle a way of determining the scopes. For this, we extend our planner, as illustrated in Figure 4.3, to initially broadcast a plan request to all checkers. Upon receiving this request, each checker determines and communicates back to the planner the specific scope within which it will operate.

Authorization Set Propagation Mechanism: Since we want to make use of the techniques explored in Chapter 2, we need to find a proper way of informing the planner of authorization sets. This can be done while broadcasting the client's workflow planning request, where checkers inform the planner of their interested scope. Additionally, we make the checkers also send their authorization sets. On the planner, the full authorization set is constructed by intersecting the sets of allowed users for each step by each checker.

Practical Scenario for Scope Determination: Consider a scenario where a checker is tasked with overseeing protocols involving sensitive data operations. Upon identifying a request relevant to its domain—such as steps involved in handling privacy-sensitive

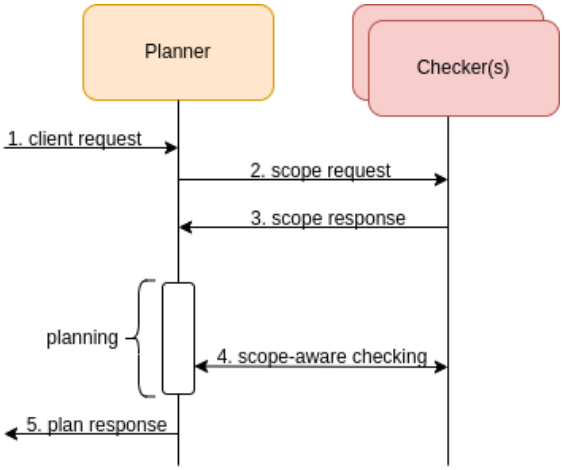

Figure 4.3: Planning Flow

information—the checker informs the planner of its intention to assess the planned task assignments within a defined nearby scope. This scope might encompass the initial analysis phase and any related subsequent steps, to ensure the desired protocol is implemented.

Altogether, the high-level steps that are executed by the planner are seen in Figure 4.3. Step by step we do the following:

1. A client of the workflow system issues a workflow request.
2. The planner feeds the request to all registered checkers.
3. Each of the checkers announces the scopes they are interested in, where the planner needs to verify their constraints acceptance. The checker also messages their authorization sets, for specifying simple allowlists constraints.
4. The planner aggregates the scopes based on the responses received by the checkers and tags them accordingly to prepare them for each type of planning.
5. Then, the planner starts the main planning routines, finding the allocation configuration that satisfies all checkers. Listing 2 describes the way we combine the UI planning from the background chapter with the planning of the general constraints.
   (a) We first find a configuration for the general part of the workflow request.
   (b) We then correlate this sub-plan with sub-plans of the UI part.
6. The planner finds the satisfying configuration and feeds it to a plan execution orchestrator, or is unable to find the correct configuration and rejects it as a response to the client.

Algorithm 2 Hybrid Planning Algorithm

Require: $G,\ A,\ C_{ud},\ Scope_{ud},\ C_{ui}$
Ensure: $Plan(G)$ **satisfies** $C_{ui},\ A,\ C_{ud}$
1: $generator \leftarrow BT_{Generator}(G, C_{ud} \cup C_{ui}, Scope_{ud})$
2: **while** $subplan_{ud} \leftarrow generator.next()$ **do**
3: $A' \leftarrow A \setminus subplan_{ud}$
4: $subplan_{ui} \leftarrow solve_{ui}(G, C_{ui}, A')$
5: **if** $subplan_{ui} \neq \emptyset$ **then**
6: **return** $subplan_{ui} \cup subplan_{ud}$
7: **end if**
8: **end while**
9: **return** $\emptyset$

4.3 Analysis

4.3.1 Computing the Complexity

To evaluate the complexity of our algorithm, we primarily focus on its worst-case scenario, not considering the constraint evaluation in our calculation. While this approach provides a boundary on the algorithm's performance, a more practical analysis would benefit from considering the average case complexity.

The central challenge in average case analysis is establishing a standard for what constitutes an 'average case'. The inherent input-dependent nature of the algorithm's complexity necessitates a thorough examination of typical input data. This examination predominantly takes one of two forms:

- **Combinatorial Analysis of Random Inputs:** This method entails the study of random graphs (workflows) and constraint expressions, through combinatorial techniques. A survey of available techniques is done by Canon et al. (3) for generating workflow scheduling graphs. However, work in generating random constraints is lacking. We will use the random graphs generated by Karapetyan and Gutin (19).

- **Empirical Analysis Using Representative Data:** This approach focuses on employing real-world representative datasets to evaluate the algorithm's performance. In the realm of big data, benchmarks such as TPCx-HS (24) and TPCx-BB (4) provide datasets that are real-world inspired. An empirical analysis for WSP could make use of these benchmarks.

An added dimension of complexity in our analysis arises from the implementation of pruning methods in the algorithm. These methods, which aim to enhance performance by narrowing the search space, can significantly impact the efficiency of the algorithm. However, their effectiveness is very dependent on the specific nature of the input, introducing a further element of variability.

A phenomenon describing the behavior of SAT-solving algorithms, including our workflow satisfiability planning, is the Phase Transition, described in Gent and Walsh (11) for general satisfiability problems and illustrated in Figure 4.4 for WSP difficulty in relation to the growing number of constraints.

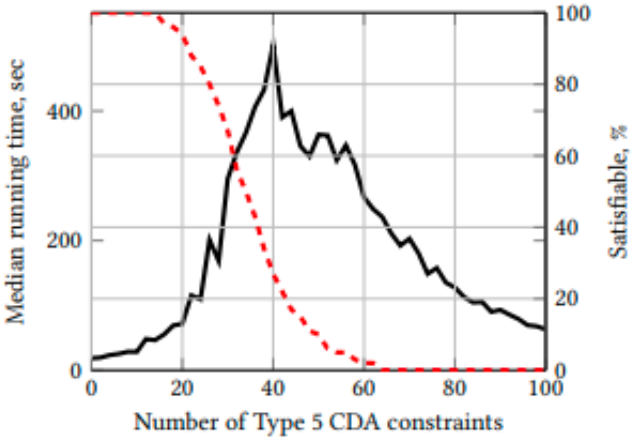

Figure 4.4: The Phase transition feature diagram. In Gutin and Karapetyan (15), a phase transition is observed about the number of constraints. The red line illustrates the probability of satisfiability, while the solid black line represents the corresponding running time. This phenomenon demonstrates that as the number of constraints increase, running times initially rise, reaching a peak before eventually decreasing. This trend correlates with the likelihood of satisfying the constraints. In scenarios with fewer constraints, the algorithm finds satisfiable configurations more readily, leading to early termination. Conversely, with an excess of constraints, the search space is pruned more aggressively, leading to earlier halts in exploration and reduced overall running times.

4.3.2 Worst Case Analysis

Given the difficulty of conducting an average case analysis within the time constraints of this thesis, we resort to a worst-case analysis instead. The scenarios we look at help us to understand the upper bounds of computational complexity and guides us in identifying the resilience of our algorithms under extreme conditions.

Assumptions for Worst-Case Analysis We base our analysis on the assumption that the constraints we deal with, will not allow early pruning in the decision tree. This means that each predicate will only reject a state at the very last node, thus requiring a full traversal of the search space. In the presence of pruning, estimations time complexities become more complicated.

In our analysis, we consider the overall structure of the workflow graph, which is a combination of UI and non-UI components. Therefore, we define the total number of steps or nodes in the graph as k, where k is the sum of the steps in the UI scope (k_{ui}) and the steps in the non-UI scope (k_{ud}) and together we have:

$$k = k_{ui} + k_{ud}$$

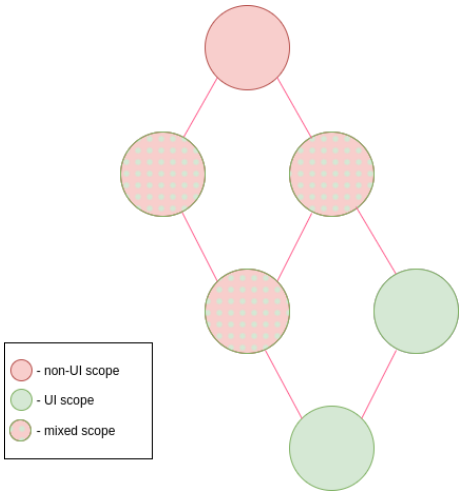

Figure 4.5: Illustration of scope types in the workflow graph. The total size of the workflow graph is denoted as k, with the combined size of the non-UI (user-dependent) and mixed scopes represented by k_{ud}, and the size of user-independent scopes represented by k_{ui}. It follows that $k = k_{ui} + k_{ud}$.

Case Analysis We identify three distinct cases based on the scope of the UI (User Interface) and non-UI constraints, as shown in Figure 4.5:

Case 1: Non-Intersecting Scopes When the scopes of UI and non-UI constraints do not intersect, each algorithm runs independently. The complexity for this scenario is $O(B_{k_{ui}} \cdot k_{ui} \cdot n + n^{k_{ud}})$, where $B_{k_{ui}}$ represents the complexity due to UI constraints, and $n^{k_{ud}}$ represents the complexity due to non-UI constraints.

Case 2: Non-UI Constraints Cover Entire Workflow In this case, the non-UI constraints encompass the entire workflow graph. Consequently, the complexity simplifies to $O(n^k)$. This scenario is less interesting from an algorithmic perspective as it does not leverage the benefits of our hybrid approach.

Case 3: Interleaved Constraints Covering the Entire Workflow The most complex and interesting scenario occurs when the non-UI and UI constraints are interleaved and together span the entire workflow. The complexity in this case is $O(n^{k_{ud}} \cdot 2^{k_{ui} \log k_{ui}} \cdot k_{ui} \cdot n)$. This complexity arises from the intertwining of the UI and non-UI constraints. Here, $O(k \cdot n)$ represents the complexity for performing maximal bipartite matching, as presented in Section 2.1.3.

Worst-Case Scenarios For the Non-Intersecting Scopes Case we can build a scenario where the constraints will reject only when the last node is labeled, therefore always making the planning algorithm explore the full search space. For the case where Non-UI Constraints cover the entire workflow, the worst-case scenario is straightforward: constraints cover the entire workflow graph and reject configurations at the last node. In contrast, constructing a worst-case scenario for the case that has Interleaved Constraints which cover the entire workflow requires more nuance. Here, we imagine a situation where non-UI constraints are met, but UI constraints fail at the last moment, such as in a failed bipartite matching scenario.

Practical Implications While these theoretical cases provide a framework for understanding the upper bounds of our algorithms, it is essential to recognize that real-world applications may not always align with these worst-case scenarios. Factors like algorithmic efficiency, data distribution, constraint early rejection chance, and real-world constraints often result in performance that deviates from these theoretical models.

Nevertheless, a particularly revealing insight is depicted in Figure 4.4, which illustrates the intrinsic pattern of varying difficulty – an easy-hard-easy transition – depending on the extent to which the constraints are applied. This more intuitive and empirical observation provides a practical and realistic understanding of the problem's nature, which extends beyond the limitations of theoretical worst-case analyses. The runtimes are primarily influenced by the peak points, which are dictated by the likelihood of satisfying the constraints.

4.3.3 Comparison with Naive Approach

We make a comparison between the two potential approaches for solving the general problem in the case of **Interleaved Constraints Covering the Entire Workflow**. The two solving algorithm candidates we compare are:

- **Full Naive Planning (Only General Constraints)**: $O(n^k)$
- **Hybrid Planning as described in Listing 2 (Combining UI and General Constraints)**: $O(n^{k_{ud}} \cdot 2^{k_{ui} \log k_{ui}} \cdot k_{ui} \cdot n)$.

An important consideration is to discern which method — Full Naive Planning or Hybrid Planning —yields a more efficient solution in relation to the specific values of the WSP parameters k (representing the number of steps in the workflow) and n (indicating the number of users involved). The decision to prefer naive planning over hybrid planning is markedly evident in contexts with a minimal user count, for instance, when $n = 2$. This is highlighted by inserting $n = 2$ into the complexity formulas for both approaches. Under these conditions, the naive method demonstrates greater efficiency, as illustrated by the relation $O(2^k) \leq O(2^{k_{ud}} \cdot k_{ui}^{k_{ui}} \cdot k_{ui} \cdot n)$.

Simplifying the inequality $n^{k_{ud}} k_{ui}^{k_{ui}} \cdot k_{ui} \cdot n < n^k$, using $k = k_{ui} + k_{ud}$ and implicitly $n^{k_{ui}} = n^{k-k_{ud}}$ leads to $k_{ui}^{k_{ui}} \cdot k_{ui} \cdot n < n^{k_{ui}}$. By substituting $x = \frac{k_{ui}}{n}$, the inequality becomes finally simplified to:

$$x^{k_{ui}-1} \cdot k_{ui}^2 \leq 1$$

We numerically solve this inequality and present a heat map in Figure 4.6, illustrating the conditions under which each planning approach is advantageous.

However, it is important to note that these results are estimations of actual performance. Due to the algorithm's susceptibility to unpredictable pruning, the worst-case analysis may diverge significantly from actual performance. Therefore, while these calculations provide a theoretical framework, real-world efficiency and effectiveness may vary.

4.4 The prototype

We created a prototype written in the Rust language, where we simplified the problem in the following way:

- The constraints are assumed to be directly defining the scopes. Scopes are embedded in the definitions of rules.

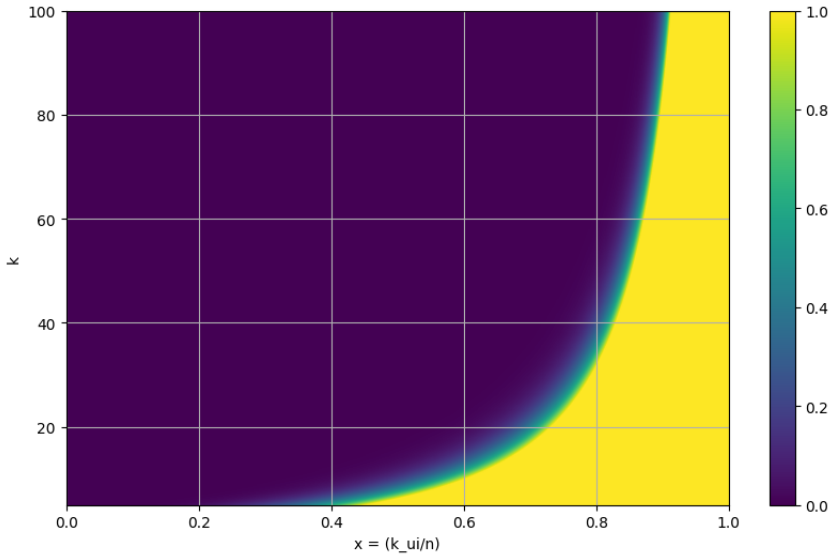

Figure 4.6: Values in the grid for the function $f(x, k) = min(x^{k-1}k^2, 1)$. The yellow area determines the subspace for which the naive algorithm should perform better.

- Constraint verification is conducted exclusively by checkers that are local to the planner, this approach simplifies the process by eliminating the need to establish separate HTTP(s) servers for each constraint or its respective checker.
- The prototype accommodates the constraint set as defined in Karapetyan and Gutin (19). Constraints are imported from files and maintained within the planner as generic function objects, with the signature $Fn(\&graph) \rightarrow bool$.

These assumptions don't limit us from easily extending our prototype in the final design described previously in Section 4.2.1. An advantage of having these limitations is the ability to utilize the tools from Karapetyan and Gutin (19) for generating benchmark sets, which include non-UI constraints. Instead of converting these constraints into branched authorization sets as detailed in their work, our prototype addresses them in their original form.

4.4.1 Implementation Features and Choices

4.4.1.1 Iterative Generators

For generating the plan configurations while exploring the search space, the algorithm may use a naive backtracking algorithm. In Listing 3, we give a possible iterative pseudo-code implementation example, similar to our implementation. This implementation allows for easy extension with our mentioned optimizations as follows:

Algorithm 4 Pseudocode of iterative backtracking that can be used for generating valid configurations

Require: *scope*, *max_user*, *constraints*

Ensure: Sequence of sub-plans satisfying constraints within *scope*

```
1: constructor BT_generator
2: this.state ← vector(scope.size(), -1)                    ▷ Initialize state vector
3: this.crt_index ← 0
4: this.constraints ← constraints
5: procedure NEXT()
6:     while true do
7:         if this.crt_index = −1 then
8:             return ∅
9:         end if
10:        this.state[crt_index] ← this.state[crt_index] + 1
11:        if this.state[crt_index] ≥ max_user then
12:            this.state[crt_index] ← −1
13:            this.crt_index ← this.crt_index − 1
14:            continue
15:        else
16:            if not this.constraints.evaluate(this.state) then
17:                continue                                  ▷ Configuration Invalid
18:            end if
19:        end if
20:        if this.crt_index = scope.size() then
21:            return this.state                          ▷ Valid Configuration Found
22:        end if
23:        this.crt_index ← this.crt_index + 1
24:    end while
25: end procedure
```

- Ordering can be solved by adding a new indirection layer (a vector that can represent the ordered indices of the initial steps).
- The previous indirection layer, is also capable of adding the filtering of users in the search space according to the authorization sets.
- Backjumping, discussed in Section 4.4.1.5, can also be implemented, particularly when considering the ordered indices of steps and their corresponding constraints. By tracking these indices, our system can efficiently backtrack to previous steps if all potential evaluations for a given node are exhausted, thereby optimizing constraint evaluation.

4.4.1.2 Optimizations

Our prototype incorporates various optimizations outlined in the background Chapter 2. However, some constraints necessitate insights into predicate internals, diverging from our aim of maintaining generality. For instance, the assignment of weights for node prioritization is challenging, given our goal of generality. Another notable limitation of node ordering is its impact on cache locality, presenting a trade-off between pruning efficiency and memory access performance. This is because node reordering introduces an additional layer of indirection which affects cache locality design.

4.4.1.3 Node Priority Optimization

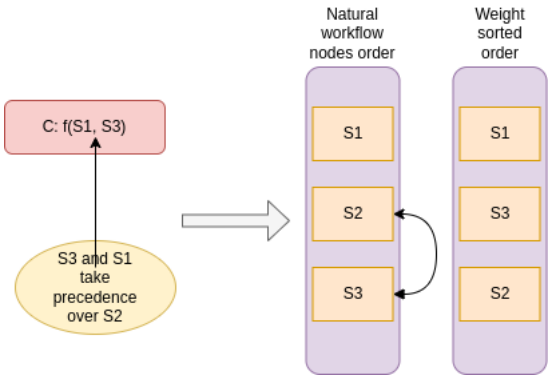

Figure 4.7: Node Prioritization Based on Encounter Frequency. This methodology prioritizes nodes according to the frequency of their references. Such prioritization facilitates the earlier processing of nodes with a higher frequency of references, thereby increasing the probability of pruning as illustrated in Figure 4.8.

We apply the Fail-First principle to enhance node ordering as described in Haralick and Elliott (16), inspired by SAT solvers. This approach involves prioritizing nodes based on their likelihood of leading to a failure, determined by the frequency of their appearances in the scopes of various constraints. We implement this by recording the scopes for each predicate and then arranging the nodes according to a weighted frequency of their occurrence in these constraint scopes. To illustrate the benefits of node ordering strategies, we provide an example with a very simplified constraint in Figure 4.8, where a good ordering can significantly enhance the efficiency of the pruning process. These node weights are adaptable and allows for tuning based on performance in validation tests.

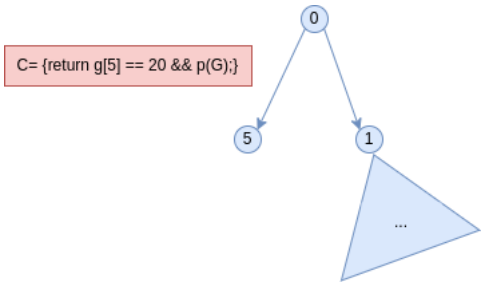

Figure 4.8: An example demonstrating the advantage of node ordering in constraint evaluation. Here, p represents an expression over the full graph. Early evaluation is facilitated when the node with id $= 5$ is labeled, allowing for the prompt rejection of unsuitable labels.

While hyperparameter optimization for specific worksets, such as those in Karapetyan and Gutin (19), is possible, this approach might not suit scenarios with opaque constraints. It risks tailoring the system to specific scenarios at the expense of broader applicability.

4.4.1.4 Combining techniques

Our system leverages the interaction between solvers, enhancing their efficacy through information exchange:

- For the general solver (non-UI), responsible for the configuration of general nodes:
  - The search is limited to labels present in the authorization set.
  - UI constraints are included to eliminate infeasible configurations early in the process.
- The bipartite matching routine as summarized in Section 2.1.1, part of the UI solver, dynamically adjusts the authorization set to reflect the user choices from the general solver's configuration, ensuring coherence between solvers and optimizing the search domain.

4.4.1.5 Backjumping

Backjumping, as detailed in Prosser (27), is generally implemented to identify conflicts in predicates. A potential approach encompasses:

1. Evaluating all constraints to identify the latest node leading to conflicts across all labelings for possible backjumping.
2. Upon encountering a situation where a node cannot be assigned due to constraints, the method allows for an immediate jump back to the penultimate node that was referenced in the constraints. This facilitates efficient backtracking by directly addressing the root cause of the conflict.

However, this method can incur high latency due to the need to run all predicate evaluations, making its practicality less general.

4.4.1.6 Implementation Summary:

Ultimately, our deliverable contribution is the development of a workflow satisfiability solver, which includes the following components:

- Two solver components to parse and manage constraints aligned with the benchmarks outlined in Karapetyan and Gutin (19). This encompasses the creation of a hybrid solver, shown in simplified form in Listing 2, which integrates:
  - A solver that is capable of handling general types of constraints.
  - A specialized solver for handling User Independent (UI) constraints, as shown in Chapter 2.
- Optimizations:
  - Node ordering based on their frequency as discussed in Section 4.4.1.3.
  - Combining techniques, utilizing results from a solver to filter out candidates when performing the subsequent User Independent solver as detailed in Section 4.4.1.4.

Several components were not fully developed due to time constraints and can constitute future work. These parts include:

- Comprehensive optimization of constraints and processing.

- Implementation of incremental bipartite matching.
- Integration of the backjumping technique.
- Detailed tuning of hyperparameters for node ordering.

4.4.1.7 Benchmark-Based Validation of the Implementation

The validation of our implementation's correctness was primarily conducted through a comparative analysis against established benchmarks and existing solvers. We utilized the benchmarking tools as outlined in Karapetyan and Gutin (19) for this purpose. By using these tools we could do a direct comparison between the performance and outcomes of our implementation and those produced by the solver referenced in Karapetyan and Gutin (19). This comparison was invaluable for promptly identifying any regressions or potential performance issues. Detailed evaluations and benchmark results is presented in Chapter 5.

5

Experimental Results

This chapter presents the results of the experimental evaluations conducted on our prototype. The experiments were designed following the methodologies outlined by Karapetyan and Gutin (19), which offered a comprehensive framework for adapting general constraints within the context of the Workflow Satisfiability Problem (WSP).

5.1 Workload Format and Constraint Descriptions

For our experiments, we employed a specific format for the workloads, including a combination of UI and non-UI constraint types. We remind the reader that the UI constraints are constraints that allow for replacing any allocated user with another one. The UI part consists of Separation of Duty (SoD) and At-most-3 (AM3) constraints as described in Section 2.1.2. The non-UI constraints are also constraints we already mentioned in a previous chapter, in Section 2.1.5:

1. Assignment Dependent (ADA) constraints.
2. Super-User At-Least constraints (SUAL).
3. Wang-Li (WL) constraints.

These constraints, along with tools for generating workloads provided by Karapetyan and Gutin (19), were crucial in our prototype's testing and measuring.

5.2 Evaluation Methodology

The methodologies employed in the study by Karapetyan and Gutin (19) and our approach differ slightly. Their assessments focus on two scaling behaviors:

- Workloads where $k = 18$, observing the growth of n, while keeping the value of k constant.
- Workloads where $10k = n$, increasing values of k, while keeping the $\frac{k}{n}$ ratio constant.

In contrast, our evaluation aims to verify the prototype's efficacy in smaller workload sizes. We primarily focus on:

- Runs with only user-independent constraints to validate our Pattern-based backtracking approach. These reuse the same workloads, limiting them to the UI constraints only.
- Runs combining both user-independent and non-user-independent constraints to assess the system's performance in more complex scenarios.

All instances, in addition to containing the defined constraints, also include authorization sets.

5.3 Benchmark Results

We run the benchmarks on the DAS-6 supercomputer (1), with SLURM jobs (32) distributed across 10 nodes. Each task runner runs a partition of the instances in parallel and gathers the average times for its partition of the instances. After the jobs have been run we aggregate the results and get the full averages of each configuration across the partitions. Each configuration pair (k, n) runs a total of 100 instances, which are then averaged.

5.3.1 User-Independent Constraint Runs

Figure 5.1 illustrates the results of running our implementation on UI-only workloads, as proposed by Karapetyan and Gutin (19). We observe notable scalability, with our system demonstrating efficient planning capabilities as the size of n increases. We attribute this mostly to our system running a Rust planner against their Python-based implementation since our optimizing efforts were simply modest. A potential other reason is the writing of the constraints in the more "pruneable" format, compared to the solver's translation to Pseudo-Boolean mathematical equations.

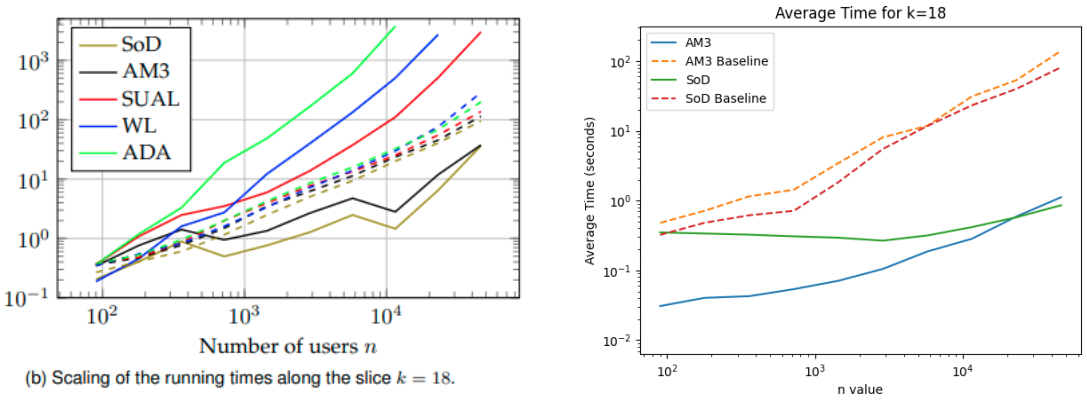

Figure 5.1: The performance of our implementation compared to Karapetyan and Gutin (19) for UI constraints with $k = 18$ and varying n.

5.3.2 Runs Incorporating Both User-Independent and General Constraints

In Figure 5.2, we examine the system's performance under combined UI and non-UI constraints. The results indicate that while our system may not match the efficiency of manual transformation methods, it effectively handles smaller configurations.

Interestingly, we achieved similar results with Wang-Li constraints as we obtained with UI constraints only. This is because compared to other non-UI configurations, Wang-Li had the smallest scope size, only 2, compared with 5 and 6 of SUAL and ADA.

A valuable direction for further investigation would involve enlarging the scope sizes of non-UI elements and evaluating the system's performance under these conditions. Additionally, conducting a study on Phase Transitions, as detailed in Figure 4.4, would be a significant enhancement. This would involve identifying the most challenging number of constraints in terms of solving time for various configurations.

5.4 Interpretation and Reproducibility

Our experiments demonstrate that the prototype, while not completely optimized, shows promising results, particularly in handling UI constraints efficiently. The performance differences observed are partly attributed to our use of Rust, in contrast to the Python-based solver employed in Karapetyan and Gutin (19). We have reproduced the results, as depicted in Figure 5.1, validating that the configuration of the Karapetyan and Gutin (19) solvers was correct and that we obtained similar results. The results of our experiments

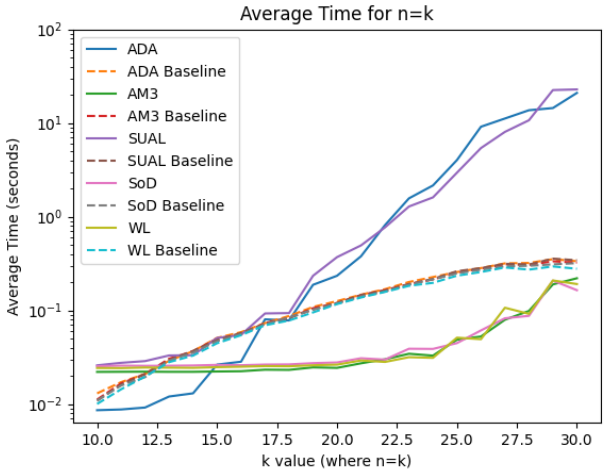

Figure 5.2: Results for combined UI and non-UI constraints, showing the system's performance across different workload sizes.

are reproducible, with the source code and work sets available via the author's GitHub repository.

6

Conclusions

For our thesis, we developed a hybrid solver capable of processing a wide array of constraints, with components specifically designed for general and User Independent (UI) constraints. However, certain aspects of our project remain underdeveloped, primarily due to time constraints. We take a look and assess the extent to which our objectives, outlined in Chapter 1, have been achieved.

6.1 Reviewing our Objectives' Completion

Returning to our study's primary goals, we assess the extent to which we have met them. Let's restate the summarized objectives of our thesis:

- Designing a system to allow for flexible constraints: Our design aimed to encompass the broadest range of constraints, treating them as abstract predicate functions over plans. Our prototype, though basic, met our expectations of generating satisfactory plans within these parameters. Furthermore, we've outlined a roadmap for enhancing the system into a more sophisticated version in Section 4.2, specifically designed for facilitating collaborative research in healthcare settings, running with distributed policy checkers and dynamic policy changes.

- Measuring and evaluating: We have utilized the instance generators and solvers provided by Karapetyan and Gutin (19) as a baseline for comparison and validation. Our performance evaluation, benchmarked against these established solvers, enabled us to assess the efficacy of our system within the field's standards. The primary objective was to develop a system that performs adequately for practical use, rather than exceeding existing benchmarks. To determine its practicality, we established a

performance criterion focused on response time for typical workloads. Our system consistently achieves evaluation times of under a few seconds for standard scenarios, aligning with our 'good enough' performance benchmark.

- In examining the system's applicability, we referenced the flexible framework of Karapetyan et al. (20) in Section 2.2. Our system's support for general constraints extends this flexibility, showing potential for very diverse application scenarios.

6.2 Future work

Given the time constraints, it was not feasible to fulfill every objective initially envisioned for this master's thesis. Additionally, during this work, we uncovered several potential areas of study. Below, we outline key suggestions for future research and relevant sources that warrant further investigation:

- Expand research to include the exploration of black-box solvers and surrogate models in constraint programming, which are discussed comprehensively in Michel and Van Hentenryck (23). This involves studying how these methods can enhance the efficiency and flexibility of constraint-solving in workflows, particularly in scenarios where direct analytical models are challenging to construct. Black-box solvers have the ability to handle problems with limited information about the underlying policy functions. If combined with surrogate models that approximate complex systems, they could significantly improve problem-solving strategies in diverse and dynamic workflow environments. Yet, integrating these methods into workflow planning under constraint-based environments remains largely unexplored and presents a significant opportunity for innovation.

- Investigate the potential of designing fixed-parameter tractable (FPT) algorithms in scenarios where the number of users (n) is constant, a situation often encountered in hospital research collaborations. This research would examine the viability and efficiency of FPT methods in these specialized contexts, assessing if they can effectively manage workflow satisfiability problems with a fixed user count.

- Enhance post-planning strategies by implementing continuous enforcement and resilience checks in workflows, following the guidelines in Crampton et al. (6). This includes adapting to policy changes and establishing fallback plans for potential execution failures.

- Conduct average case analyses to better understand typical workloads in real-world applications, or use mathematical tools like Random Graphs for theoretical insights.
- Develop parallelization techniques for handling large-scale planning tasks. Inspiration can be drawn from parallel Davis–Putnam–Logemann–Loveland (DPLL) solvers, as described in Feldman et al. (10), to improve workflow satisfiability algorithms.

Our study's primary focus was the adaptability of our framework to various constraints, a key aspect in the field of workflow satisfiability. Our way of tackling this was mostly from the perspective of allowing the constraints to be as general as possible. While we've made significant strides in this area, the practical application of our findings, particularly in specialized domains like hospital collaborations, requires deeper investigation. A targeted study exploring the specific needs and dynamics of hospitals engaged in collaborative research would be invaluable. Such a study, approached from a multidisciplinary angle, especially within policy enforcement in information systems, could provide critical insights into optimizing workflows in these complex environments.

References

[1] Henri Bal, Raoul Bhoedjang, Rutger Hofman, Ceriel Jacobs, Thilo Kielmann, Jason Maassen, Rob Van Nieuwpoort, John Romein, Luc Renambot, Tim Rühl, et al. The distributed asci supercomputer project. *ACM SIGOPS Operating Systems Review*, 34 (4):76–96, 2000. 39

[2] David Basin, Søren Debois, and Thomas Hildebrandt. On purpose and by necessity: compliance under the gdpr. In *Financial Cryptography and Data Security: 22nd International Conference, FC 2018, Nieuwpoort, Curaçao, February 26–March 2, 2018, Revised Selected Papers 22*, pages 20–37. Springer, 2018. 4, 16, 17

[3] Louis-Claude Canon, Mohamad El Sayah, and Pierre-Cyrille Héam. A comparison of random task graph generation methods for scheduling problems. In *Euro-Par 2019: Parallel Processing: 25th International Conference on Parallel and Distributed Computing, Göttingen, Germany, August 26–30, 2019, Proceedings 25*, pages 61–73. Springer, 2019. 27

[4] Paul Cao, Bhaskar Gowda, Seetha Lakshmi, Chinmayi Narasimhadevara, Patrick Nguyen, John Poelman, Meikel Poess, and Tilmann Rabl. From bigbench to tpcx-bb: Standardization of a big data benchmark. In *Performance Evaluation and Benchmarking. Traditional-Big Data-Internet of Things: 8th TPC Technology Conference, TPCTC 2016, New Delhi, India, September 5-9, 2016, Revised Selected Papers 8*, pages 24–44. Springer, 2017. 27

[5] Jason Crampton, Andrei Gagarin, Gregory Gutin, Mark Jones, and Magnus Wahlström. On the workflow satisfiability problem with class-independent constraints for hierarchical organizations. *ACM Transactions on Privacy and Security (TOPS)*, 19(3):1–29, 2016. 18

[6] Jason Crampton, Gregory Gutin, and Rémi Watrigant. Resiliency policies in access control revisited. In *Proceedings of the 21st ACM on Symposium on Access Control Models and Technologies*, pages 101–111, 2016. 43

[7] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962. 20, 25

[8] Leonardo De Moura and Nikolaj Bjørner. Satisfiability modulo theories: introduction and applications. *Communications of the ACM*, 54(9):69–77, 2011. 20

[9] Daniel Ricardo dos Santos and Silvio Ranise. A survey on workflow satisfiability, resiliency, and related problems. *arXiv preprint arXiv:1706.07205*, 2017. 6, 18

[10] Yulik Feldman, Nachum Dershowitz, and Ziyad Hanna. Parallel multithreaded satisfiability solver: Design and implementation. *Electronic Notes in Theoretical Computer Science*, 128(3):75–90, 2005. 44

[11] Ian P Gent and Toby Walsh. The sat phase transition. In *ECAI*, volume 94, pages 105–109. PITMAN, 1994. 28

[12] Google. Or-tools, 2024. URL `https://github.com/google/or-tools`. [Online; accessed 10-January-2024]. 21

[13] Robert Grandl, Srikanth Kandula, Sriram Rao, Aditya Akella, and Janardhan Kulkarni. {GRAPHENE}: Packing and {Dependency-Aware} scheduling for {Data-Parallel} clusters. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 81–97, 2016. 19

[14] Gregory Gutin. The workflow satisfiability problem with user-independent constraints. In *2019 First International Conference on Graph Computing (GC)*, pages 1–4. IEEE, 2019. 9, 16

[15] Gregory Gutin and Daniel Karapetyan. Constraint branching in workflow satisfiability problem. In *Proceedings of the 25th ACM Symposium on Access Control Models and Technologies*, pages 93–103, 2020. 28

[16] Robert M Haralick and Gordon L Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial intelligence*, 14(3):263–313, 1980. 35

[17] John E Hopcroft and Richard M Karp. An n^5/2 algorithm for maximum matchings in bipartite graphs. *SIAM Journal on computing*, 2(4):225–231, 1973. 9

[18] ANSI INCITS. Incits 359-2004. *Role based access control*, 2004. 4, 16

[19] Daniel Karapetyan and Gregory Gutin. Solving the workflow satisfiability problem using general purpose solvers. *IEEE Transactions on Dependable and Secure Computing*, 2022. 6, 15, 21, 27, 32, 35, 36, 37, 38, 39, 40, 42

[20] Daniel Karapetyan, Andrew J Parkes, Gregory Gutin, and Andrei Gagarin. Pattern-based approach to the workflow satisfiability problem with user-independent constraints. *Journal of Artificial Intelligence Research*, 66:85–122, 2019. 5, 6, 10, 11, 12, 13, 43

[21] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955. 9

[22] Kia Kai Li. Exploring k-colorability. *arXiv preprint cs/0702058*, 2007. 10

[23] Laurent Michel and Pascal Van Hentenryck. Activity-based search for black-box constraint programming solvers. In *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems: 9th International Conference, CPAIOR 2012, Nantes, France, May 28–June1, 2012. Proceedings 9*, pages 228–243. Springer, 2012. 43

[24] Raghunath Nambiar, Meikel Poess, Akon Dey, Paul Cao, Tariq Magdon-Ismail, Da Qi Ren, and Andrew Bond. Introducing tpcx-hs: the first industry standard for benchmarking big data systems. In *Performance Characterization and Benchmarking. Traditional to Big Data: 6th TPC Technology Conference, TPCTC 2014, Hangzhou, China, September 1–5, 2014. Revised Selected Papers 6*, pages 1–12. Springer, 2015. 27

[25] Andrew M. Odlyzko and L. Bruce Richmond. On the number of distinct block sizes in partitions of a set. *Journal of Combinatorial Theory, Series A*, 38(2):170–181, 1985. 12

[26] Jaehong Park and Ravi Sandhu. The uconabc usage control model. *ACM transactions on information and system security (TISSEC)*, 7(1):128–174, 2004. 3

[27] Patrick Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational intelligence*, 9(3):268–299, 1993. 36

[28] William Tolone, Gail-Joon Ahn, Tanusree Pai, and Seng-Phil Hong. Access control in collaborative systems. *ACM Computing Surveys (CSUR)*, 37(1):29–41, 2005. 3

[29] Yakir Vizel, Georg Weissenbacher, and Sharad Malik. Boolean satisfiability solvers and their applications in model checking. *Proceedings of the IEEE*, 103(11):2021–2035, 2015. 7

[30] Guozhang Wang, Joel Koshy, Sriram Subramanian, Kartik Paramasivam, Mammad Zadeh, Neha Narkhede, Jun Rao, Jay Kreps, and Joe Stein. Building a replicated logging system with apache kafka. *Proceedings of the VLDB Endowment*, 8(12):1654–1655, 2015. 24

[31] Qihua Wang and Ninghui Li. Satisfiability and resiliency in workflow authorization systems. *ACM Transactions on Information and System Security (TISSEC)*, 13(4): 1–35, 2010. 6, 10, 15, 16, 24

[32] Andy B Yoo, Morris A Jette, and Mark Grondona. Slurm: Simple linux utility for resource management. In *Workshop on job scheduling strategies for parallel processing*, pages 44–60. Springer, 2003. 39