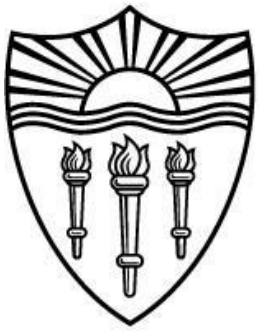

A Comprehensive Project Report Submitted in Partial Fulfillment of the Academic Requirements for the Course in Optimization Techniques for Data Science

University of Southern California

Time-Table Scheduling using Simple Temporal Constraints

By

Jason Sutanto

Riten Bhagra

Project Supervisor: Dr. Satish Kumar Thittamaranahalli

Applied Data Science
University of Southern California

ABSTRACT

This project presents a novel approach to timetable scheduling by integrating difference constraints with graph theory, utilizing the Bellman-Ford algorithm. We transform scheduling into a graph-based problem, where the Bellman-Ford algorithm is employed to calculate the earliest and latest start times for activities, enhancing both the efficiency and flexibility of scheduling. A key feature of our methodology is the inclusion of an interactive component, allowing users to modify schedules to suit changing requirements. This approach focuses on practically implementing these concepts in a user-friendly scheduling tool tailored for real-world applications. Additionally, our approach incorporates shortest-path computations to identify independent events, streamlining schedule adjustments by isolating events unaffected by others. This aspect further enhances the model's adaptability and responsiveness to dynamic scheduling environments. Our findings demonstrate simplifying and adapting complex scheduling tasks, combining theoretical graph algorithms with practical needs.

This the link to our github project: https://github.com/JSutanto19/DSCI-599-Frontend

CONTENTS

1. INTRODUCTION AND RELATED WORK

Temporal constraint problems are prevalent in diverse computer science areas, including scheduling, program verification, and parallel computation. These problems have gained significant attention in AI-related fields such as common-sense reasoning [7,8], natural language understanding [9,10], and planning [11]. Various formalisms for representing and reasoning about temporal knowledge have been developed, including Allen's interval algebra [12], Vilain and Kautz's point algebra [13], linear inequalities, and time map representations, each supported by specific reasoning algorithms. While extensive research has been conducted on general constraints, its application to temporal constraints has been less explored. Our project presents a comprehensive approach to temporal reasoning using a constraint-network framework, addressing the gap in applying general constraint research to temporal problems. This enables a multifaceted analysis of algorithmic schemes, focusing on complexity and applicability, and introduces a minimal network representation for encoding all temporal relations between pairs of events, including their absolute time differences.

A pivotal element of our study is the management of metric temporal information. Conventional methodologies, such as Allen's interval algebra and Vilain and Kautz's point algebra, have constraints in effectively addressing this aspect. To tackle these challenges, we have adopted a methodology that accentuates the temporal distance between time points, thereby creating a framework of linear inequalities. In our approach, we utilize difference constraints as a means to model these linear inequalities, enhancing the precision and applicability of our temporal analysis.

The foundational elements of our study are propositions, each typically linked to a time interval. For instance, statements like "I was driving a car" or "the book was on the table" each corresponds to a duration in which they are true. The temporal data can be relative (such as "A happened before B") or more quantifiable (like "A started at least 3 hours before the end of B"). To articulate vaguer information, the system might also employ disjunctive phrases, like "arrive before or after lunch," with a subset of these phrases being a focus of our discussion. In addition, the system allows for references to specific times (e.g., 4:00 p.m.) and the duration of events (like "A lasted for a minimum of two hours"). With such temporal data, our system aims to deduce answers to questions about the feasibility of a proposition at a given time, possible timings for a proposition, and potential temporal relations between two propositions.

Various models have been proposed for representing temporal information. If propositions represent events, and each event P_i is linked to a time span $I_i=[A_i, B_i]$, then the timing of events can be depicted through constraints on these intervals or their start and end points. Allen introduced the concept of temporal knowledge as constraints on 13 possible relationships between interval pairs. However, computing all possible relationships between intervals is complex, leading Vilain and Kautz to propose constraints on the intervals' start and end points, resulting in a more efficient polynomial-time algorithm, albeit with limited problem-solving scope. Ladkin and Maddux later introduced an algebraic method for similar

issues. Our system is particularly designed to handle metric information. Since Allen's interval algebra and Vilain and Kautz's point algebra are less suited for this, our approach differs. We focus on time points, which could be the start or end of an event or a specific time like 10:00 p.m. Malik and Binford, and Valdés-Pérez, suggested constraining the temporal distance between these points. For example, if X_i and X_j are time points, a constraint might be $X_j - X_i \leq c$, forming a set of linear inequalities.

In the context of temporal reasoning and constraint satisfaction, our system can be illustrated through a practical example involving daily commutes. Consider the scenarios of two individuals, Robin and Raj, with varying transportation options and associated travel times. Robin has the option to commute to work by car, taking approximately 20-50 minutes, or by bus, which takes over 30 minutes. Raj, on the other hand, can travel by car within 10-20 minutes or opt for a carpool, which takes 30-40 minutes. The complexity of their schedules provides a rich context for applying our temporal reasoning system.

Let's define P1 as the proposition 'Robin was going to work' and P2 as 'Raj was going to work.' These propositions, P_1 and P_2, are tied to time intervals $[X_1, X_2]$ and $[X_3, X_4]$ respectively, where X_1 denotes the time Robin left home and X_4 marks Raj's arrival at work. To model the temporal aspects of their commutes, we introduce a series of constraints based on their transportation choices and commute durations.

For Robin, the time to reach work by car is between 20 to 50 minutes, and by bus, it's more than 30 minutes. Thus, the temporal distance between X_1 (Robin's departure) and X_2 (Robin's arrival at work) can be expressed as a constraint: $20 \leq X_2 - X_1 \leq 50$ for the car, or $X_2 - X_1 > 30$ for the bus. Similarly, for Raj, the constraints for commuting by car are $10 \leq X_4 - X_3 \leq 20$, and by carpool, $30 \leq X_4 - X_3 \leq 40$. Suppose we have additional data points, such as specific times when Robin and Raj left their homes or arrived at their workplaces. These details would impose further constraints on the X values. For instance, if Robin left home at 7:30 a.m, we could establish a constraint like $7:30 \leq X_1 \leq 7:40$, assuming a 10-minute window for departure.The core of our system lies in managing these constraints to answer queries about the consistency of the given information, the feasibility of specific transportation choices, and the potential departure and arrival times. The disjunctive nature of the sentences, indicating multiple possible scenarios, adds complexity to the problem. Our system, therefore, not only interprets these disjunctive statements but also processes them within the broader framework of temporal constraints.

In our framework, we incorporate the Bellman-Ford algorithm, renowned for its efficiency in finding shortest paths in weighted graphs, to handle difference constraints in scheduling. This algorithm’s ability to determine the earliest and latest start times for each activity within a timetable is central to our approach. Moreover, the inclusion of an interactive module in our scheduling tool empowers users to dynamically adjust and customize schedules, making our system adaptable and user-friendly.

Our project also delves into the practical implications of these theoretical concepts in real world applications, particularly focusing on the challenges of timetable scheduling. By integrating theoretical graph algorithms with practical scheduling needs, we aim to offer a robust and flexible solution to complex scheduling problems.
The subsequent sections of this paper will detail the methodology, implementation, and evaluation of our approach. We will demonstrate how our system effectively simplifies the scheduling process, provides flexibility in managing temporal constraints, and how the

interactive component enhances the overall user experience. By bridging the gap between theoretical constructs and practical application, our research contributes significantly to the field of temporal reasoning and scheduling.

2. TEMPORAL CONSTRAINT SATISFACTION PROBLEM

The foundational concepts for defining a temporal constraint satisfaction problem (TCSP) are largely based on the general CSP framework. In a TCSP, we deal with a collection of variables, $X_1, \ldots, X_n$, each representing a time point and having a continuous domain. Constraints in a TCSP are defined by a series of intervals:

$$\{I_1, \ldots, I_N\} = \{[a_1, b_1], \ldots, [a_n, b_n]\}. \tag{2.1}$$

A unary constraint, T_i, limits the domain of variable X_i to a specific set of intervals, essentially forming a disjunction:

$$(a_1 \leq X_i \leq b_1) \vee \ldots \vee (a_n \leq X_i \leq b_n). \tag{2.2}$$

In the case of a binary constraint, T_{ij}, the permissible values for the difference $X_j - X_i$ are determined, represented as:

$$(a_1 \leq X_j - X_i \leq b_1) \vee \ldots \vee (a_n \leq X_j - X_i \leq b_n). \tag{2.3}$$

These constraints are typically presented in a canonical form, with each interval being mutually exclusive.

A binary TCSP network comprises a set of variables $X_1, \ldots, X_n$ and their associated unary and binary constraints. This network is depicted through a directed constraint graph, where nodes symbolize variables and each edge $i \rightarrow j$ indicates constraints T_{ij}, labeled by their respective interval set. A special point, X_0, denoting the "starting time" is included for reference. All times are relative to X_0, allowing us to treat constraints T_i as binary constraints T_{0i} with the same interval representation.

Consider the following scenario involving a series of events in Robin's day. Let X_0 represent the start of the day. The events and their associated temporal constraints are as follows:

- **Bedtime** (X_0): Robin goes to bed at 12 AM, marking the start of his day.
- **Sleep Duration** (X_1): Robin sleeps for 4 to 5 hours after going to bed. This is represented as a binary constraint T_{01} with the interval [4 hours, 5 hours].
- **Breakfast Time** (X_2): After waking up, Robin spends 1 to 2 hours for breakfast. This introduces a binary constraint T_{12} with the interval [1 hour, 2 hours].

- **Commute to Work** (X_3): It takes Robin 2 to 3 hours to travel to work after breakfast. This introduces a binary constraint T_{23} with the interval [2 hours, 3 hours].
- **Work Duration** (X_4): Robin works for 6 to 7 hours. This introduces a binary constraint T_{34} with the interval [6 hours, 7 hours].
- **End of Day** (X_5): Robin winds up his day by 4 PM. This introduces a a binary constraint T_{04} with the interval [16 hours].

The directed constraint graph for this TCSP can be conceptualized with nodes representing the variables X_0, X_1, X_2, X_3, and X_4, and edges indicating the temporal constraints between these events.

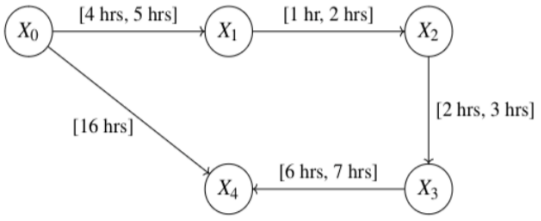

Figure 2.1: Directed constraint graph for the revised TCSP scenario

A solution in a TCSP is a tuple $X = (x_1, ..., x_n)$ where the assignment $\{X_1 = x_1, ..., X_n = x_n\}$ satisfies all constraints. A value v is deemed feasible for variable X_i if there exists a solution where $X_i = v$. The collection of all feasible values for a variable forms its minimal domain.

3. SIMPLE TEMPORAL PROBLEM

A Simple Temporal Problem (STP) is a type of Temporal Constraint Satisfaction Problem (TCSP) where every constraint is defined by a singular interval. In this framework, an edge $i \rightarrow j$ in the network is annotated with an interval $[a_{ij}, b_{ij}]$. This interval signifies the constraint:

$$a_{ij} \leq X_j - X_i \leq b_{ij}. \tag{3.1}$$

This constraint can also be interpreted through a dual set of inequalities:

$$X_j - X_i \leq b_{ij}, \tag{3.2}$$

$$X_i - X_j \leq -a_{ij}. \tag{3.3}$$

Solving an STP is equivalent to finding solutions for a series of linear inequalities associated with the variables X_i. The general problem of resolving linear inequalities is well-acknowledged in operations research. Solutions can be found using methods like the

simplex algorithm, though it can be computationally intensive, or Khachiyan's algorithm, which is complex in practical applications. However, the specific linear inequalities in an STP can be addressed more simply in a graph format, allowing for the application of shortest path algorithms like Bellman Ford. In formal terms, an STP is associated with a directed, edge-weighted graph $G_d = (V, E_d)$, known as a distance graph (distinct from the constraint graph). This graph maintains the same set of nodes as G, and each edge $i \rightarrow j$ is assigned a weight a_{ij}, representing the linear inequality $X_j - X_i \leq a_{ij}$.

`

The graph for an STN, S =(T,C), is a graph, G =(T,E), where:

Time-points in S $\Longleftrightarrow$ nodes in G

Constraints in C $\Longleftrightarrow$ edges in E:

$X_j - X_i \leq a_{ij} \Longleftrightarrow$ where a_{ij} is a edge weight from X_i to X_j.

Moreover, for our example discussed in figure 2.1, the STP constraints are given by:

$$4 \leq x1 - x0 \leq 5$$

$$1 \leq x2 - x1 \leq 2$$

$$2 \leq x3 - x2 \leq 3$$

$$6 \leq x4 - x3 \leq 7$$

$$0 \leq x4 - x0 \leq 16$$

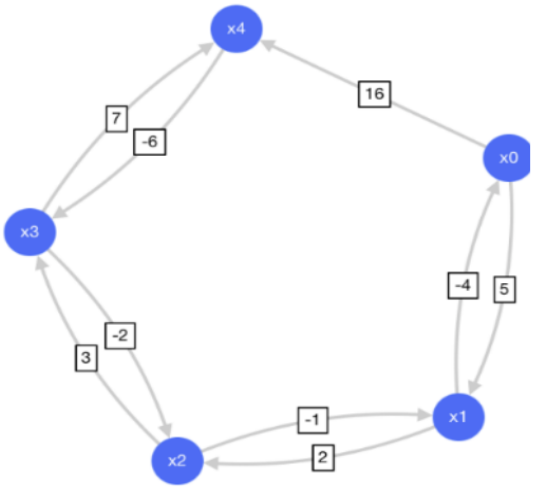

Figure 3.1: Graphical representation of the STP constraints discussed above.

4. Bellman-Ford algorithm:

Introduction to the Bellman-Ford Algorithm

The Bellman-Ford algorithm is a fundamental technique in computer science for finding the shortest path from a single source vertex to all other vertices in a graph. This algorithm is versatile, as it can be applied to both weighted and unweighted graphs, making it a crucial tool in network analysis, routing, and map navigation.

Versatility and Limitations

Unlike Dijkstra's algorithm, which is another popular shortest path algorithm, Bellman-Ford is capable of handling graphs that contain edges with negative weights. This unique feature allows it to be used in a broader range of applications, including systems where costs or distances might decrease over certain paths. However, its versatility has a limitation: the algorithm is unable to compute the shortest path in graphs that contain negative cycles. In such graphs, the presence of cycles with negative overall weight leads to paths that can indefinitely decrease in cost, making it impossible to find a minimum distance that is well-defined.

Principle of Relaxation in Bellman-Ford Algorithm

The core principle behind the Bellman-Ford algorithm is 'relaxation', which is a process of iteratively updating the shortest distance estimates of vertices. Initially, the distance to each vertex from the source is assumed to be infinite. The algorithm then repeatedly relaxes the edges of the graph, updating the distance estimates to reflect shorter paths if they are found. This process is crucial for accurately determining the shortest paths in the graph.

Detecting Negative Cycles with Bellman-Ford

An important aspect of the Bellman-Ford algorithm is its ability to detect negative cycles in a graph. The edges in the graph are relaxed for N-1 iterations (where N is the number of vertices) to calculate the shortest paths. If the shortest distance to a vertex can be reduced on the Nth relaxation, it indicates the presence of a negative cycle. The rationale is that a decrease in the shortest distance estimate after N-1 relaxations implies that the path has looped back on itself, revisiting vertices, which is a characteristic of a negative weight cycle. This feature is particularly important in financial systems, network routing, and other applications where negative weight cycles could disrupt the calculation of shortest paths.

Pseudocode:

```
foreach X_i do
    d(X_i) := ∞
d(S) := 0;
for i := 1 to n − 1 do
    foreach edge (X_i, δ, X_j) do
        d(X_j) := min{d(X_j), d(X_i) + δ};
foreach edge (X_i, δ, X_j) do
    if d(X_j) > d(X_i) + δ then return NO;
return {D(S,X_1),..., D(S,X_n)};
```

The Bellman-Ford algorithm, as described by the provided pseudocode, is a classic algorithm in computer science for finding the shortest path from a single source vertex to all other vertices in a weighted graph. It is particularly noteworthy for its capability to handle negative edge weights.

The algorithm begins with an initialization process where it assigns a tentative distance value to every vertex in the graph: infinity for all nodes except the source node, which is assigned a distance of zero. This setup represents the assumption that the source node is at zero distance from itself, and all other nodes are initially unreachable.

Following initialization, the algorithm enters a phase of relaxation, which is iteratively performed N-1 times, where N is the total number of vertices in the graph. During each iteration, the algorithm attempts to update the distance to each vertex by considering all edges. If the current distance to a vertex X_j is greater than the distance to an adjacent vertex X_i plus the edge weight δ between them, the distance to X_j is updated to this sum. This step is repeated, as updating one vertex's distance might enable a shorter path to a neighboring vertex in subsequent iterations.

After completing the relaxation steps, the algorithm checks for negative weight cycles. This is done by examining each edge to see if the distance to the destination vertex of the edge can still be reduced. If a reduction is possible, it implies that a negative weight cycle exists, since the shortest paths should have already been finalized after N-1 relaxations. The presence of such a cycle would mean that an infinitely decreasing path length is possible, which is not solvable for a shortest path algorithm. Hence, the algorithm returns 'NO' if a negative cycle is detected.

If no negative cycles are found, the algorithm proceeds to return the shortest distances from the source vertex to each other vertex in the graph. This final step concludes the Bellman-Ford algorithm, providing a comprehensive solution for single-source shortest path problems that can accommodate the intricacies of negative edge weights and detect unsolvable scenarios involving negative cycles.

4.1. Earliest and Latest Times Using Bellman-Ford

In the realm of scheduling and managing temporal constraints, a critical aspect is the association of each event, denoted as X_i, with a specific time window. This assignment necessitates effective strategies to manage these constraints. Prominently, two principal solutions are employed to address this, augmented by the computational efficiency of the Bellman-Ford algorithm for determining shortest distances.

The first solution, known as the Latest Execution-Time Solution, prioritizes the latest possible commencement times for each event. This approach is methodically calculated using the formula $X_i = D(Z, X_i)$, where D represents the delay function relative to a reference point Z. The Bellman-Ford algorithm plays a pivotal role here, computing the shortest distances and thereby determining the most delayed yet feasible starting times within the constraints of the time windows.

Conversely, the Earliest Execution-Time Solution adopts a different strategy by determining the earliest possible start time for each event. This is quantified by the formula $X_i = -D(X_i, Z)$, indicating the calculation of the earliest commencement time relative to the reference point Z. Here again, the Bellman-Ford algorithm is instrumental in efficiently computing the shortest paths to ascertain the earliest possible initiation times for the events, within their designated time windows.

By employing these two approaches, coupled with the Bellman-Ford algorithm for shortest path calculations, the scheduling process becomes more flexible and efficient. Each event is scheduled either as late as possible, adhering to the upper bounds of the time windows, or as early as possible, ensuring a prompt commencement. Thisdual strategy, backed by the computational prowess of the Bellman-Ford algorithm, allows for a more dynamic and adaptable scheduling framework, accommodating various temporal constraints and ensuring the optimal use of time for each scheduled event.

However, it's crucial to note that the Bellman-Ford algorithm assumes the absence of negative cycles in the graph representing the scheduling constraints. If negative cycles are present, it indicates that the total length of a cycle is negative, leading to an infinite reduction in the path length by repeatedly traversing the cycle. In such scenarios, no feasible solution can be determined using this algorithm, as the existence of negative cycles implies the inability to establish consistent temporal constraints for scheduling the events.

5. Rescheduling: Adapting to Changes in Scheduling Constraints

In dynamic environments, schedules often require adjustments to accommodate changes in constraints or priorities. This section delves into the process of recalculating schedules when such adjustments are necessary, particularly focusing on a specific task chosen as a new starting

point. When a task is rescheduled, it effectively becomes the new reference point (denoted as x0) in the scheduling framework.

The process begins by recalculating the constraints in relation to this new reference task. This involves reassessing and adjusting the duration and sequence of subsequent tasks to ensure they align with the newly defined start time. This adjustment is crucial as it maintains the integrity of the schedule, ensuring that subsequent tasks do not start before the completion of the preceding task, especially the newly rescheduled one.

Once the constraints are recalculated, the Bellman-Ford algorithm is reapplied. In this context, the algorithm serves two primary purposes: First, to determine the latest possible start times for all tasks, considering the new reference point. This is achieved by recalculating the shortest distances from the reference task to all other tasks, ensuring that each task begins as late as possible within the newly defined constraints. Second, the algorithm is used to calculate the earliest start times by computing the shortest paths in a graph with reversed edges, thereby indicating the earliest each task can begin without causing a delay in the overall schedule.

This rescheduling approach ensures flexibility and responsiveness in the scheduling process. By recalculating the constraints and applying the Bellman-Ford algorithm in light of a new starting point, the schedule is dynamically updated to reflect the current priorities and requirements. This adaptability is crucial for effective time management, especially in projects where conditions and resource availability may change frequently.

6. Implementation of frontend and backend:

Backend and Algorithm Implementation
This section of the report outlines the algorithm used in the project to manage and optimize task scheduling. The algorithm, written in Python, utilizes the NetworkX library for graph theory and the Bellman-Ford algorithm for shortest path computation. It addresses the problem of scheduling a series of tasks within a given time frame, considering variable task durations.

Algorithm Overview:

The algorithm comprises several key functions, each designed to handle different aspects of the scheduling problem:

User Input Handling: The get_user_input function collects the number of tasks and their respective durations from the user. It supports both fixed and ranged durations.

Time Format Conversion: The convert_to_24_hour_format function converts time from a 12-hour format to a 24-hour format.

Constraint Formatting: The format_constraints function formats the constraints for output, showing the permissible duration range for each task.

Graph Construction: The build_graph function creates a directed graph using NetworkX, representing tasks and their constraints as nodes and edges.

Bellman-Ford Implementation: The bellman_ford function implements the Bellman-Ford algorithm to find the shortest path in the graph, which corresponds to the optimal task schedule.

Graph Visualization: The print_graph function outputs the constructed graph's nodes and edges.

Time Conversion: The time_conversion function adjusts the calculated hours back to a 12-hour format for user-friendly output.

Constraint Adjustment: The adjust_constraints function modifies the constraints if the user opts to change the start time of any task.

Main Process
The main function orchestrates the entire process:

Input Collection: It starts by collecting user inputs for tasks and their durations, as well as the desired start and end times for the day.

Constraint Addition: It adds a global constraint that encompasses the total available hours in the day.

Graph Building and Analysis: The function then constructs a directed graph based on these inputs and constraints, and applies the Bellman-Ford algorithm to determine the earliest and latest start times for each task.

Output Display: The function displays formatted constraints, the graph structure, and the calculated earliest and latest start times for each task.
Schedule Adjustment: Users have the option to modify the start time of tasks. The algorithm recalculates and updates the schedule based on these changes.

Flask API: We utilized a Flask API to facilitate data transfer between the backend and our frontend components. This allowed us to integrate Python code seamlessly with our frontend. To achieve this, we established two POST endpoints: '/visualize'. These endpoints are crucial for updating the state of the frontend upon execution, ensuring a dynamic and responsive user experience.

Frontend Implementation Approach
Choosing Next.js and Tailwind CSS:
For the frontend, Next.js was our choice for building components that underscore the main functionality of the application. With its server-side rendering and static site generation capabilities, Next.js significantly enhances performance and SEO, making it an ideal framework for a contemporary, responsive web application. We paired this with Tailwind CSS for styling, leveraging its utility-first approach for quick, customizable designs. This combination ensures our application is not only functionally efficient but also visually engaging and consistent.

Incorporating D3.js for Graph Visualization:
To effectively present bidirectional edge graphs, we integrated D3.js. This tool adeptly visualizes the directions, weights of edges, and nodes produced by our algorithm, offering a clear and interactive graphical representation.

Component Implementation and Data Flow

Component-Based Architecture:

Our project adopts a component-based architecture that is both robust and modular, with 'page.tsx' serving as the root of our component tree. This particular architecture was chosen for its flexibility and scalability, enabling our development team to manage and update individual components effectively without impacting the entire system. The 'page.tsx' root component acts as a central hub, orchestrating the flow of data and managing the state across the application. This structured hierarchy allows for the seamless integration of various child components, such as forms for data input, dynamic graph representations for visual feedback, shortest path visualizations for algorithmic clarity, and schedule components for temporal organization. Each component is designed to be self-contained, with clearly defined interfaces for communication. This not only simplifies the debugging process but also enhances the collaborative workflow, as developers can work on different components simultaneously without conflict.

Integration with Backend and Data Processing:

The integration of the frontend with the backend is a critical aspect of our application, ensuring that user inputs are effectively processed to deliver meaningful outputs. The data flow within our application is meticulously designed to be both efficient and intuitive.

Here's an in-depth look at how it functions:

User Input and Data Transmission: At the frontend, users are presented with a streamlined form that captures essential data points. Once the form is submitted, the data is packaged into a structured format and sent to the backend via a POST request to the Flask Python API, targeting the '/visualize' endpoint. This interaction serves as the initial trigger for the data processing pipeline.

Algorithmic Processing: Upon receiving the data, our backend unleashes the power of the constraint satisfaction problem (CSP) scheduling algorithm. This algorithm is fine-tuned to handle complex datasets and generate optimal solutions. It meticulously processes the input data to construct a graph of nodes and edges, calculating the shortest paths and transit times utilizing advanced heuristics and optimization techniques.

Visualization and Scheduling: The results of the CSP algorithm, now in the form of a comprehensive graph, are relayed to the visualizer component on the frontend. This component is adept at translating raw data into an interactive visual format, offering users a vivid representation of the paths and schedules. Simultaneously, the '/schedule' endpoint is engaged to perform additional computations, refining the raw output into a coherent and detailed schedule.

Frontend Display and User Interaction: Finally, the frontend receives the processed schedule, where it is artfully rendered into a series of cards. This presentation is not just visually appealing but also designed to be user-friendly, allowing for easy interpretation of the algorithm's results. Each card provides a snapshot of the schedule, with timings, locations, and other pertinent details, facilitating an intuitive user experience. The card-based display is chosen for its modern aesthetic and its ability to present complex information in a digestible format, ensuring that users can effortlessly grasp the intricacies of their custom-generated schedules.

Below are images of our frontend:

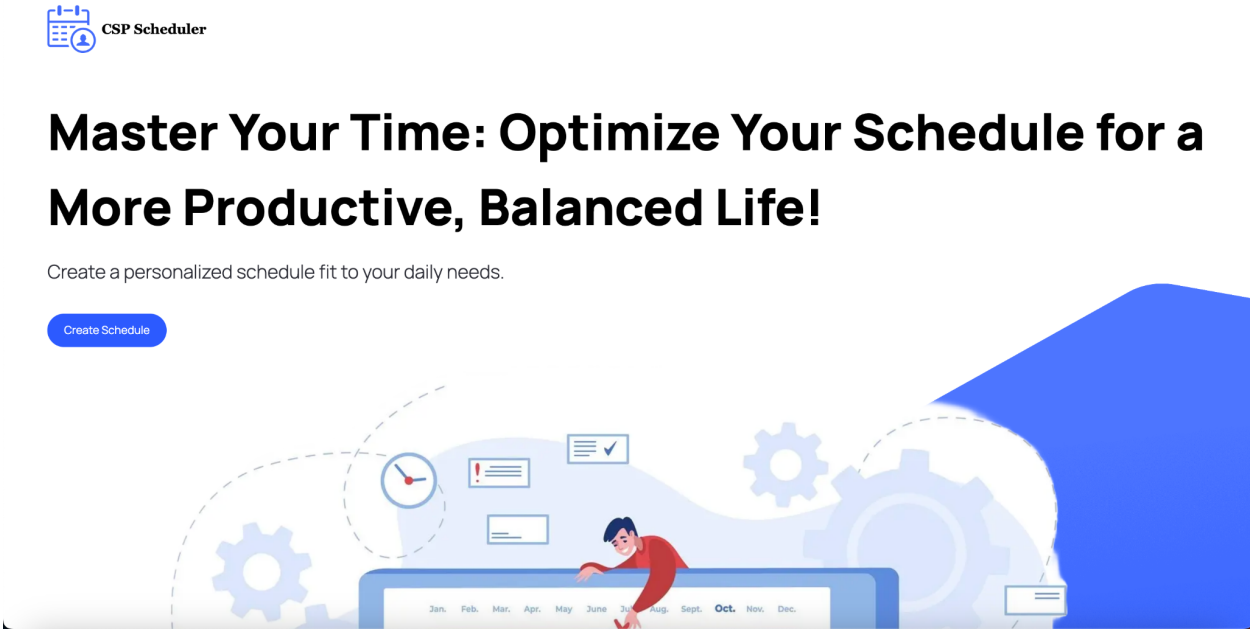

Figure 5.1 our landing page

Figure 5.2 Our Form Component

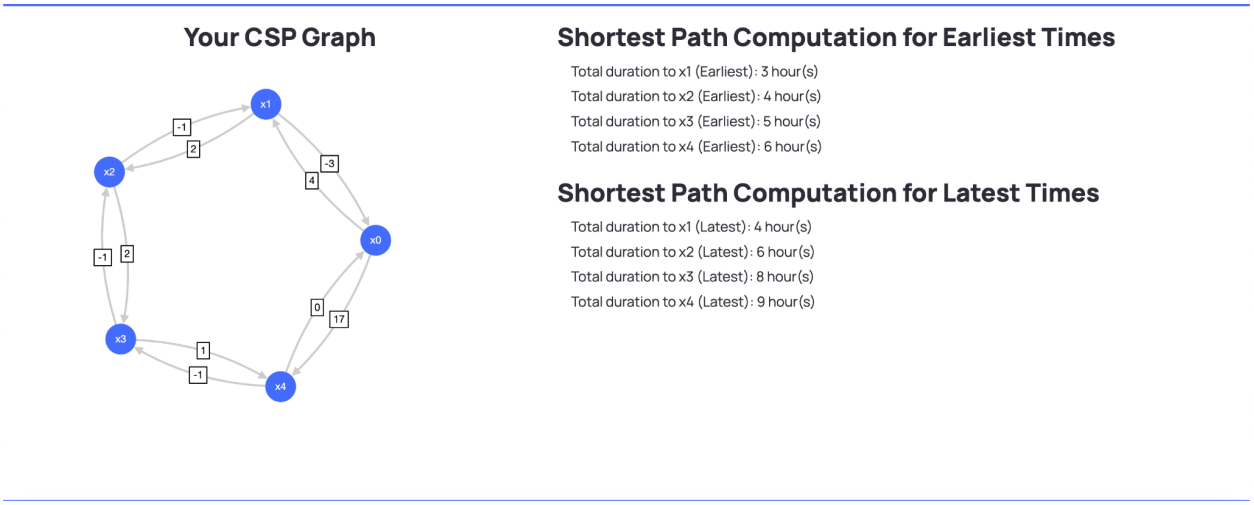

Figure 5.3 Graph and Shortest Path Component

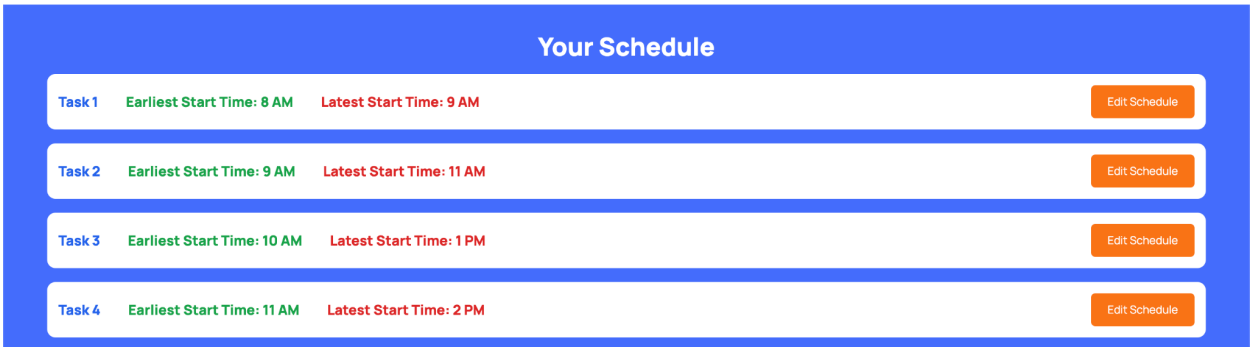

Figure 5.4 Scheduler component

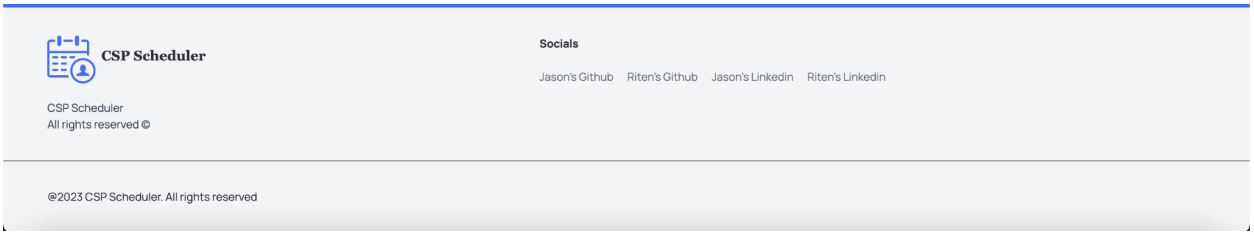

Figure 5.5 Our Footer

This comprehensive integration between the frontend and backend, coupled with a robust data processing pipeline, forms the backbone of our application, ensuring that users receive accurate, timely, and actionable insights from their inputs.

7. Limitations

1. Sensitivity to Dynamic Changes: While the system allows for interactive modifications, it may have limitations in real-time responsiveness to dynamic changes. The re-computation time

necessary for the Bellman-Ford algorithm to adapt to changes in scheduling can introduce delays, potentially rendering it less effective in environments where schedules are highly fluid and require immediate updates.

2. Handling of Negative Cycles: Our approach does not accommodate the generation of schedules in the presence of negative cycles within the graph. If such cycles exist due to the input constraints, the Bellman-Ford algorithm will fail to produce a valid schedule. This limitation requires pre-processing to ensure that input data does not lead to negative cycles, which may not always be feasible in complex scheduling scenarios.

3. Complexity in Interpretation for End-Users: Although the tool is designed to be user-friendly, the underlying complexity of graph-based scheduling could be challenging for end-users without technical backgrounds. The interpretation of schedules, particularly understanding the implications of changes in constraints and the resultant modifications in the timetable, may not be intuitive for all users. This could necessitate additional training or the development of more simplified interfaces to bridge the gap between the algorithmic complexity and user experience.

4. Dependency on Accurate Input Data: The scheduling tool's effectiveness is heavily contingent on the accuracy of the input data provided by users. Since the system relies on user-provided temporal constraints to construct schedules, any inaccuracies or misrepresentations in this data can lead to schedules that are not only suboptimal but also potentially impractical. This limitation underscores the necessity for rigorous data validation and error-checking mechanisms to ensure the reliability of schedule outputs.

5. Performance Issues for Large-Scale Applications: The tool demonstrates efficiency and robustness for moderate-sized problems. However, as the scale of the scheduling problem increases, the tool might encounter significant performance challenges. The Bellman-Ford algorithm, while powerful, involves computational intensities that grow with the size of the graph. Handling very large and complex scheduling scenarios, such as those with numerous events and constraints, could result in longer processing times, thus affecting the overall responsiveness and usability of the tool in large-scale applications. This performance bottleneck is an important consideration when deploying the tool for enterprise-level scheduling tasks.

7.1 Conclusion:

This report has detailed an innovative approach to the challenge of timetable scheduling by leveraging the principles of graph theory combined with difference constraints, utilizing the Bellman-Ford algorithm as the cornerstone of our methodology. Our graph-based framework marks a significant advancement in the field, enhancing the efficiency and flexibility with which start times for various activities are determined. This shift to a graph problem paradigm represents a notable departure from traditional methods, streamlining complex scheduling operations into more manageable and efficient processes.

A pivotal feature of our system is its interactive and user-friendly design, which embodies the seamless integration of complex theoretical models with tangible, real-world applications. The interactive component of our tool is specifically tailored to empower users, granting them the

ability to actively engage with and modify schedules as per their evolving requirements. This alignment of algorithmic sophistication with intuitive user experience is indicative of our commitment to bridging the gap between advanced computational strategies and end-user accessibility.

The adaptability and responsiveness of our model are further augmented by the implementation of shortest path analyses. This approach is instrumental in identifying independent events within the schedule, thereby facilitating quick and streamlined adjustments. Such a capability is invaluable in dynamic scheduling environments where conditions are subject to frequent change, necessitating a model that can swiftly respond without the need for extensive recalculations.

Ultimately, our findings illustrate the impactful simplification that our model brings to complex scheduling tasks. By combining the theoretical rigor of graph algorithms with the practical demands of scheduling necessities, our approach sets a new benchmark for future scheduling solutions. It is our assertion that the principles and methodologies expounded in this report will serve as a catalyst for further innovation in scheduling technology, inspiring continued advancements that will carry the field forward into a new era of efficiency and user-centered design.

7.2 Future Work:

The limitations identified in the current iteration of our scheduling tool pave the way for several avenues of future work that can significantly enhance its functionality and user experience. The following sections delineate potential developments that can address the current shortcomings and propel the tool towards greater efficacy and wider applicability.

Enhanced Real-Time Responsiveness: Future iterations will aim to improve the system's responsiveness to dynamic changes. This could involve the integration of more sophisticated algorithms capable of incremental updates, thereby minimizing re-computation times. Research into algorithms that allow for partial re-computation in response to schedule changes, as opposed to full recalculations, will be a priority.

Robust Handling of Negative Cycles: To better accommodate scenarios where negative cycles may be present, future work will explore alternative algorithms or augmented versions of the Bellman-Ford algorithm that can handle such cycles. We will also investigate pre-processing techniques to detect and manage negative cycles before they impact the scheduling process.

User Experience Improvements: Recognizing the complexity that users face in interpreting graph-based schedules, we intend to develop more intuitive interfaces and visualization tools. These enhancements will help demystify the scheduling process, making it more accessible to users irrespective of their technical expertise. Educational resources and interactive tutorials could also be developed to aid users in understanding and utilizing the tool effectively.

8.1 Individual Contributions:

In this project, Jason exhibited a commendable level of expertise and dedication. His primary responsibilities encompassed the development of key frontend features, demonstrating a profound understanding of user interface design and client-side programming. Moreover, Mr. Sutanto was instrumental in creating figures 5.1 to 5.5, showcasing his adeptness in data visualization and presentation. Additionally, he was tasked with the development of a Flask-based API, a critical component for backend integration. This contribution was pivotal in ensuring efficient and seamless communication between the client interface and the server. Jason's diverse skill set and meticulous attention to detail were invaluable assets to the project, significantly enhancing both its functionality and user experience.

Riten was the mastermind behind our novel timetable scheduling method, ingeniously integrating graph theory and the Bellman-Ford algorithm with Python to manage scheduling challenges as a graph-based problem. His approach significantly enhances efficiency and flexibility by calculating the earliest and latest start times for activities. The system includes an interactive feature, enabling users to adapt schedules to changing needs, making it highly suitable for real-world scenarios. Additionally, Riten's model employs shortest-path computations to isolate independent events, streamlining adjustments in dynamic environments. It efficiently handles interconnected events with weighted edges, optimizing time-efficient sequences and dynamically responding to changing constraints. Overall, Riten's innovative system skillfully simplifies complex scheduling tasks by merging theoretical graph algorithms with practical problem-solving abilities.

REFERENCES

1. Tsamardinos, I., Muscettola, N., and Morris, P. (1998). Fast Transformation of Temporal Plans for Efficient Execution.In 15th National Conf. on Artificial Intelligence (AAAI-1998), pages 254–261. Xu, L. and Choueiry, B. Y. (2003).
2. *Tsamardinos, I., Muscettola, N., and Morris, P. (1998).Fast Transformation of Temporal Plans for Efficient Execution. In 15th National Conf. on Artificial Intelligence (AAAI-1998), pages 254–261. Xu, L. and Choueiry, B. Y. (2003).*
3. *Ramalingam, G., Song, J., Joskowicz, L., and Miller, R. E. (1999).Solving Systems of Difference Constraints Incrementally.Algorithmica, 23(3):261–275.*
4. *Ramalingam, G. and Reps, T. (1996). On the Computational Complexity of Dynamic Graph Problems. Theoretical Computer Science, 158:233–277.*
5. *Hunsberger, L. (2008). A practical temporal constraint management system for real-time applications. In European Conf. on Artificial Intelligence (ECAI-2008), pages 553–557.*
6. *A new efficient algorithm for solving the simple temporal problem.In 10th Int. Symp. on Temporal Representation and Reasoning and 4th Int. Conf. on Temporal Logic (TIME-ICTL-2003*
7. *Hanks and DN. McDermott, Default reasoning, nonmonotonic logics, and the frameproblem, in: Proceedings AAA1-86, Philadelphia, PA (1986) 328-333.*
8. *Y. Shoham, Reasoning about Change: Time and Causation from the Standpoint of Artificial Intelligence (MIT Press, Cambridge, MA, 1988).*
9. *J.F. Allen, Towards a general theory of action and time, Artif. lntell. 23 (2) (1984) 123-154.*
10. *K. Kahn and G.A. Gorry, Mechanizing temporal knowledge, Artif. Intell. 9 (1977) 87-108.*
11. *D.V. McDermott, A temporal logic for reasoning about processes and plans, Cogn. Sci. 6 (1982) 101-155.*
12. *J.F. Allen, Maintaining knowledge about temporal intervals, Commun. ACM 26 (11) (1983) 832-843.*
13. *M. Vilain and H. Kautz, Constraint propagation algorithms for temporal reasoning, in: Proceedings AAAI-86, Philadelphia, PA (1986) 377-382.*