Leveraging SAT-SMT for TA Scheduling Optimization: Enhancing Efficiency and Effectiveness

KISHAN THAKURANI, University of Twente, The Netherlands

The scheduling of teaching assistants (TAs) is a complex task, similar to the Nurse Rostering Problem. The TA scheduling problem involves finding the best way to assign TAs to different time slots. A feasible schedule must adhere to all the strict constraints, while an optimal schedule maximizes the flexible constraints. This paper discusses the methodology and implementation of an algorithm that produces an optimized schedule using Z3 and Google OR-Tools. It also evaluates the strengths and shortcomings of these libraries. Additionally, the paper explores research on preference modeling and potential methods for gathering preferences, which are informed by discussions and polls among TAs.

Additional Key Words and Phrases: Scheduling, Optimization Techniques, SAT-SMT Solvers, Constraint Programming, Preference Modeling

1 INTRODUCTION

Institutions rely on employee scheduling for adequate staffing and efficient resource utilization. The same applies within the context of our university, with teaching assistants (TAs) for various modules. This process involves assigning TAs to sessions based on availability, expertise, and preferences. Traditionally, people have performed TA scheduling manually, a time-consuming method prone to suboptimal results, taking up to 50 hours per module. During module 2 (Software Systems), we realized that it would be possible to schedule TAs algorithmically and did so using Python. While this algorithm produced feasible results, it did not produce optimal ones.

In recent years, constraint programming has emerged as a powerful tool for solving complex scheduling problems. This method involves defining a set of constraints and using a solver to find solutions that satisfy these constraints. Constraint programming is well-suited for scheduling issues that affect numerous hard constraints (e.g., availability, maximum working hours) and soft constraints (e.g., preferences, balanced workloads) [6]. By automating the scheduling process, institutions can ensure a fairer distribution of tasks and improve overall satisfaction among TAs while significantly reducing the time required for module planning.

The TA scheduling problem shares many similarities with the Nurse Rostering Problem (NRP), which has been extensively studied in operations research. The NRP involves assigning nurses to shifts while satisfying various hard and soft constraints. Hard constraints include requirements such as each shift needing a minimum number of nurses, while soft constraints involve preferences and other factors that improve the overall schedule quality [10]. Researchers have employed various methodologies to tackle the NRP, including constraint programming approaches, which have shown promising results in balancing constraints and optimizing schedules [6].

TScIT 41, July 5, 2024, Enschede, The Netherlands

Tools like the Z3 solver [11], developed by Microsoft Research, and Google OR-Tools [5] have been instrumental in solving scheduling problems, such as NRP or the job-shop scheduling problem. This research aims to provide a comprehensive solution to the TA scheduling problem by leveraging the strengths of different constraint-solving techniques and incorporating preference modeling.

A significant challenge in solving the TA scheduling problem arises from its inherent computational complexity. The problem is NP-hard because it is an optimization problem with an exponential search space of On^m, where n is the number of TAs and m is the number of sessions; this means that even with a relatively small number of TAs and sessions, the number of possible assignments is vast. For instance, with just 3 TAs and three sessions, the search space already involves $3^3 = 27$ possible arrangements, as illustrated in Figure 1. As the number of TAs and sessions increases, the complexity grows exponentially, making it challenging to find optimal solutions within a reasonable timeframe.

Contributions. In this paper, we will address the following questions: How can we leverage constraint-solving techniques to optimize scheduling the TAs? How can preferences of TAs and teachers be accurately modeled and integrated into a constraint-based scheduling system?

2 PRELIMINARIES

2.1 Constraint Programming

Constraint programming is a paradigm for solving combinatorial problems that involve finding values for problem variables within given constraints. These constraints define the conditions that solutions must satisfy. The strength of constraint programming lies in its ability to handle complex constraints and provide feasible solutions efficiently [12]. In TA scheduling, constraints can be divided into hard constraints, which must be strictly satisfied, and soft constraints, which are desirable but not mandatory. The goal is to find solutions that meet all hard constraints and optimize the satisfaction of soft constraints.

2.2 SAT-SMT Solvers

SAT (Satisfiability) and SMT (Satisfiability Modulo Theories) solvers are tools used in constraint programming to solve logical formulas. SAT solvers determine if an assignment of variables exists that makes a given Boolean formula ‘true.‘ SMT solvers extend SAT solvers by incorporating additional theories such as arithmetic, bit-vectors, arrays, and uninterpreted functions. The Z3 solver, developed by Microsoft Research, is one of the most efficient SMT solvers available. It has been widely used in software verification and analysis applications due to its robust performance and versatility [2].

Despite its strengths, Z3 struggles with large-scale scheduling problems due to the exponential growth of the solution space with the number of variables and constraints. This limitation necessitates

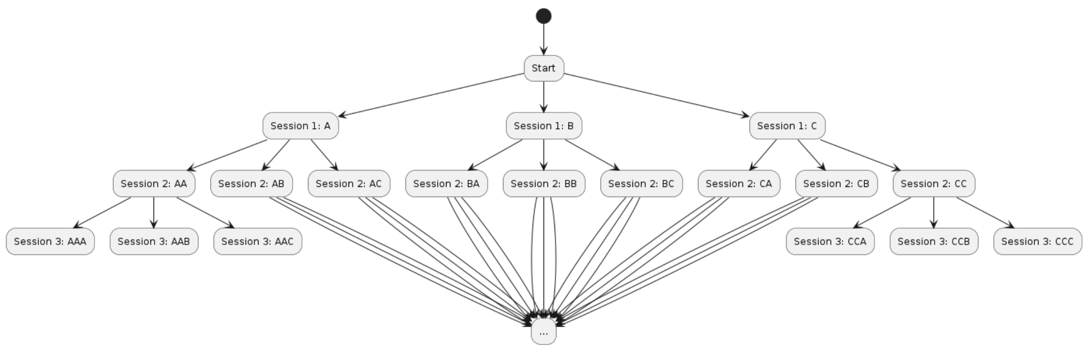

Fig. 1. Search space of TA assignment

exploring alternative approaches, such as Google OR-Tools, which can provide feasible solutions within a specified time frame using constraint programming and optimization techniques [7].

2.3 Scheduling Algorithms

Scheduling algorithms are designed to allocate resources to tasks over time. In TA scheduling, the resources are TAs, and the tasks are teaching sessions. Practical scheduling algorithms must account for the availability, preferences, and expertise of TAs while ensuring that all sessions are adequately staffed. Various algorithms have been developed to tackle scheduling problems, ranging from simple heuristic methods to more complex optimization techniques [9].

Heuristic methods (IE greedy algorithms) provide quick solutions but may not yield optimal results. Optimization techniques, including constraint programming and integer linear programming, aim to find the best solution by exploring the entire solution space. However, these techniques can be computationally intensive, especially for large problems. Hybrid approaches combining different methods can balance solution quality and computational efficiency [8].

3 METHODOLOGY

3.1 Overview

This chapter outlines the methodology we used to address the TA scheduling problem. It begins with a detailed description of the problem formulation. The chapter also discusses the data preparation, the design of the experiments, and the evaluation metrics used to assess the proposed solution's performance.

3.2 Problem Formulation

We define the TA scheduling problem as follows:

- **Input Variables**
  - **TA names**: A list of strings representing the names of the TAs (in our example 60 of them)
    `ta: List[str]`
  - **Time Slots**: A list of strings representing different time slots in which sessions occur, for example, "1 General Tool Installation", representing the first session.
    `ts: List[str]`
  - **TA preferences**: In Excel, each TA marks a session as 0, -1, or 1, so this is a dictionary that translates a TA for a specific session into their preference.
    `preference[ta, ts]: Dict[(str, str), int]`
  - **Date and time slots per session**: Dictionaries that convert a timeslot into their corresponding start and end time (represented by python `datetime`)
    `start_time[ta, ts]: Dict[(str, str), time]`
    `end_time[ta, ts]: Dict[str, str), time]`
  - **Number of needed TAs per session**: Maps each session string to the amount of TAs needed for that session
    `needed_tas[ts]: Dict[str, int]`
  - **Number of rooms per session**: Maps each session string to the number of rooms used for that session
    `needed_rooms[ts]: Dict[str, int]`
- **SMT Variables**:
  - Whether or not a TA is assigned to a session, either True or False, one per session per TA, so in our case with 60 TAs and 110 sessions, we would make $60 \cdot 110 = 6600$ of these.:
    `Boolean(f'assigned_{ta}_{ts}')`
  - Difference between the number of hours each TA is assigned. This variable represents, for each TA (ta1), the total number of hours they work subtracted from the number of hours every other TA works. For 60 TAs, this is 60^2 or 3600 variables:
    `Int(0, 200, f'diff_{ta1}_{ta2}')`
  - Room that each TA is assigned to for each session. For each TA assigned to a session, this variable is set between 1 and 3 to see if they go to rooms 1, 2, or 3. There are also $60 \cdot 110 = 6600$ of these variables. :
    `Int(0, 3, f'room_{ta}_{ts}')`
  - Homeroom of each TA. Unlike the previous variable, which is per session, there is one homeroom per TA, and we try to ensure each TA spends as much time in their homeroom as possible:
    `Int(0, num_rooms - 1, f'homeroom_{ta}')`
- **Constraints**:
  - **Hard Constraints**:
    * Each session must have the needed number of TAs scheduled
    * TAs must not be scheduled for overlapping sessions.

    - No TA is scheduled for a session they are unavailable for.
    - TAs must be balanced across rooms
  - **Soft Constraints**:
    - Maximize TA preferences for each session.
    - Balance the TA workload
    - Maximize the amount of time a TA spends in their homeroom.
- **Objective**: To find a schedule that satisfies all hard constraints and optimizes the satisfaction of soft constraints.

3.3 Implementation of constraints

- **TA Number:** The goal with this constraint is to ensure that the number of TAs assigned to a session is the amount needed. There is, however, the risk that there are not enough available TAs; because of this, two ways to represent this is with either a soft constraint, minimizing the difference between the amount of TAs assigned and needed, or defining a variable (required) and using hard constraints to ensure that the assigned number of TAs is equal to the required amount: $\forall_{ts} \quad \text{required\_tas}[ts] = \min(\text{available}, \text{needed\_tas}[ts-1])$ $\forall_{ts} \quad \text{assigned}[ts] = \text{required}[ts]$
- **Overlap:** To ensure no overlapping sessions, we keep track of each session's start and end times. As such, we add the hard constraint that if there is an overlap for each pair of sessions, then a TA can only be assigned to one of the two. $\forall_{ts1,ts2}(\text{start}[ts1] < \text{end}[ts2]) \wedge (\text{start}[ts2] < \text{end}[ts1]) \Rightarrow \forall_{ta}, \sum[\text{assigned}[(ta, ts1)], \text{assigned}[(ta, ts2)]] \leq 1$
- **Availability:** If a TA is not available, it should not be possible to assign them to a session: $\text{preference\_score}[(ta, ts)] == -1 \Rightarrow \text{assigned}[(ta, ts)] = 0$
- **Room Balancing:** To ensure TAs are balanced across rooms, we ensure the number of TAs in the room with the lowest amount of TAs is at most one less than the room with the most TAs; this number is also equal to $\left\lfloor \frac{\text{needed\_tas[ts]}}{\text{needed\_rooms[ts]}} \right\rfloor$ which means the constraint can be written as:

$$\forall_{ts}\text{needed\_rooms[ts]} > 0 \Rightarrow$$
$$\forall_{r \in rooms\ x \in \{1..\text{needed\_rooms[ts]}\}}$$
$$\sum \text{count}(r == x) \geq \left\lfloor \frac{\text{needed\_tas[ts]}}{\text{needed\_rooms[ts]}} \right\rfloor$$

- **Maximizing TA Preferences:** Our algorithm offers two implementations. The first method involves removing TA preferences from the logical solver's equation. Instead, it marks TAs as unavailable if more than enough TAs prefer a session. The second method counts the number of sessions given to a TA they prefer and then maximizes the sum of these sessions.

$$\forall_{ta}\ \texttt{preference\_score}[ta] =$$
$$\sum_{\texttt{slot}}^{\texttt{ts}} \texttt{assigned[slot]} \cdot \texttt{preference}[ta, \texttt{slot}]$$
$$\texttt{Maximize} \sum_{\texttt{a}}^{\texttt{ta}} \texttt{preference\_score[a]}$$

- **Balance TA workload:** From both a management and a fairness perspective, it makes sense to want TAs to have as similar workload as possible. To do this, we want to ensure that the sum of the differences between the TA hours is as low as possible.

$$\forall_{ta1,ta2}\ \texttt{diff}[(ta1, ta2)] =$$
$$\Big| \sum_{session}^{ts} \texttt{assigned}(ta1, session) \cdot \texttt{duration}(session)$$
$$- \sum_{session}^{ts} \texttt{assigned}(ta2, session) \cdot \texttt{duration}(session) \Big|$$
$$\text{Minimize} \sum_{ta1,ta2}^{ta} \texttt{diff}[(ta1, ta2)]$$

- **Maximize homeroom time**: We had to keep track of how long each TA is in their homeroom to maximize that amount.

$$\forall_{ta}\texttt{time\_in\_homeroom}[ta] =$$
$$\sum^{ta,ts} \texttt{duration}[ts] * \texttt{is\_assigned\_to\_homeroom}[ta, ts]$$
$$\texttt{Maximize} \sum^{ta} \texttt{time\_in\_homeroom}[ta]$$

4 PREFERENCE MODELLING

One difficult situation for TAs with many available sessions is the inability to say how much they want a session accurately. This was another critical point of interest, as it allowed us to explore alternative methods of receiving our input on the preference score.

4.1 Binary Preference Scoring

In the binary preference model, TAs label each session as "willing" or "unwilling." This simplified approach streamlines preference collection, especially when only basic information about availability is required. For instance, a TA would assign a "1" (willing) or "0" (unwilling) to each session.

While this model is advantageous for management due to its simplicity and the ease of analyzing availability, it has significant drawbacks for the TAs. The binary model limits TAs from expressing varying degrees of choice for different sessions. This limitation can lead to lower overall satisfaction among TAs, as they cannot indicate which sessions they prefer more strongly. As a result, TAs might feel that their individual preferences are not adequately considered, potentially leading to dissatisfaction and decreased morale.

4.2 Fixed-Preference Scoring

In the fixed-preference scoring model, TA preferences are represented as discrete values. Each TA assigns scores to sessions based on their preferences. The goal is to maximize the sum of scores while satisfying all constraints. This model is one that TAs are used to, as it is how it has been done in many modules, including the one this algorithm is based on (Module 2: Software Systems). The way this model works is that for each session, a TA chooses to leave the slot as "1", "0", "-1," or blank, which roughly equates to "preferred," "don't mind," "unpreferred," and "unavailable." In this way, assigning a TA to a session they have left blank is impossible.

4.3 Budget-Based Preference Modelling

The budget-based preference model allows TAs to allocate a budget of preference points across sessions. Our experiences as TAs and discussions with fellow TAs inspired this approach. We polled the TA community to understand the importance of different soft constraints, revealing the need for a more flexible preference system. The poll was conducted by giving a list of soft constraints and asking TAs to rate each one on a scale of importance, with one being least important and five being most important. The poll results in 2 showed that TAs had widely varying opinions on the importance of each constraint, with all averages in the range 2.8 to 3.8 (with the notable outlier of lowering standard deviation having the most 5s), most questions have nearly as many 'low preference' as 'high preference,' indicating that there is no way to definitively say which constraint should be prioritized over another since choosing that is equally helpful to one TA as it is damaging to another. The budget-based model enables a better balance between TA satisfaction and scheduling constraints by giving TAs a budget (for example, with 110 sessions, we could theoretically give 110 'points' to each TA). For each session, TAs can allocate any positive integer as their preference as long as their total preferences do not exceed their budget. This method simplifies the algorithm by providing a clear metric to maximize (the number of points given) while allowing for greater flexibility in expressing preferences.

5 EXPERIMENTAL SETUP

5.1 Data Collection

The experiments' data is collected from the TA scheduling spreadsheet of Module 2: Software Systems. The dataset includes TA availability, preferences, and sessions. Synthetic data was randomly generated for the budget-based system to simulate various preferences.

5.2 Tools and Software

- **Z3 Solver**: Used for encoding and solving the constraint satisfaction problem.
- **Google OR-Tools**: Used for optimization and constraint-solving, providing a practical implementation framework.
- **Python**: Scripting and interfacing with solvers and tools.
- **Microsoft Excel**: Used for data analysis

5.3 Implementation Details

- **Encoding the Problem**: The information is taken from the Excel file where the TA preferences were given and is analyzed with pandas data frames to extract the necessary information to encode the variables and constraints properly.
- **Solving Strategy**: The constraints are written in Z3 and SAT-SMT, and algorithms are written incrementally, adding additional constraints between each version.
- **Evaluation Metrics**: Performance is evaluated based on the quality of schedules, computational time, and TA satisfaction. Metrics include the number of constraint violations, the total preference score, and the workload balance among TAs.

6 RESULTS

6.1 Iteration 1: Initial comparison statistics

This section presents the experiment results. Table 1 highlights the first experiment's results, which helped us understand the direction of our research. The number of TAs assigned to a session and the duration of a session are both static numbers. This means that the average should always be 85.57. This result showed that Z3 produced an unfeasible result after running for 6 hours. After strengthening the constraints to provide a more rigid search space for Z3, the program ran for over 48 hours without terminating, showing us that a library that allowed for a maximum time to be set (such as OR-Tools, which we limited to 5 minutes) would provide better results. The other highlighted cells in this table show the rest of the critical information, such as the best-performing method for each measure taken. As seen, OR-Tools have outperformed manual scheduling in nearly every category except for minimums. Another way to visualize the disparity between the assigned TA hours would be with graph 3, which shows that the results generated by Z3 are the least balanced since the number of hours has many peaks and valleys that the other two don't. Furthermore, while the OR-Tools and manual methods are relatively close to the mean, it's easy to tell that the OR-Tools line is more consistent than manual scheduling. Another interesting note about Z3 is that it is biased toward the variables it encounters first. This can be seen in the graph, as the general trend of the Z3 line is sloping downwards, showing that Z3 prefers to give sessions to TAs who are alphabetically first.

6.2 Iteration 2: Preference score

The algorithm ensured that TAs only got their preferred sessions if another TA with a higher preference score could not get that session. This was done in a purely pythonic way, simply by checking:

```python
# Check if there are more TAs with the highest
# preference than needed
if len([ta for ta in TAs if
       preference_score[(ta, slot)] == 3])
       >= needed_tas[i - 1]:
    for ta in TAs:
        if preference_score[(ta, slot)] != 3:
            availability[(ta, slot)] = False
# Check if there are more TAs with at least a medium
# preference score than needed
elif len([ta for ta in TAs if
        preference_score[(ta, slot)] >= 2])
        >= needed_tas[i - 1]:
    print(f"Session {slot} is using TAs with medium
    preference")
    for ta in TAs:
        if preference_score[(ta, slot)] < 2:
            availability[(ta, slot)] = False
else:
    print(f"Session {slot} is using TAs with low
            preference")
```

However, to push the limits of the optimizer, we decided to attempt to formulate this as another optimization problem within the

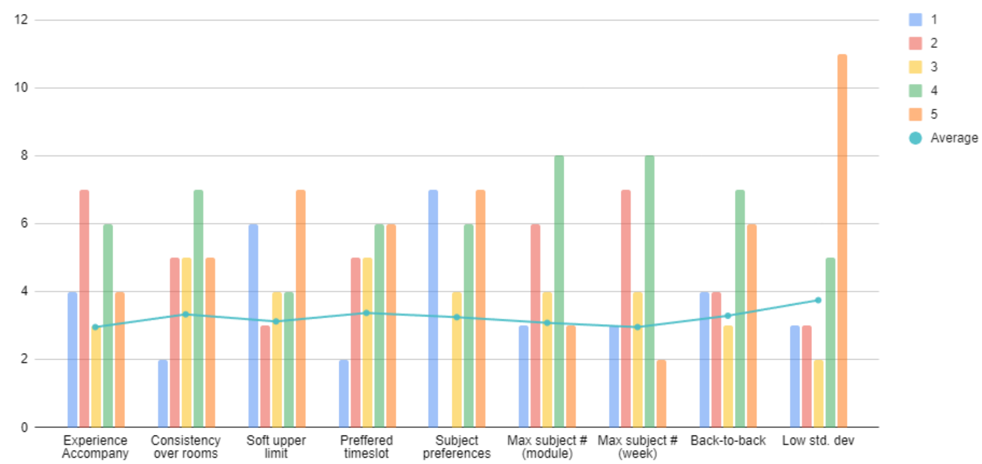

Fig. 2. TA Responses, with column charts dictating how many TAs responded to each option and a line chart indicating the average

	Average	Deviation	Median	Min	Max	Range	Time Taken
Manual	85.57	17.91	84	32	141	109	5 hours
Z3	82.17	58.43	8	4	220	216	6 hours
OR-Tools	85.57	12.09	87	26	118	92	5 minutes

Table 1. Comparison of Different Methods with Highlighting. Red highlights indicate infeasibility, and green highlights indicate the best value across the three methods. Average (mean hours assigned), Deviation (standard deviation, lower is better), Median (middle value, should be close to average), Minimum (fewest hours assigned, higher is better), Maximum (most hours assigned, lower is better), Range (spread of hours, lower is better), IQR (interquartile range, lower is better). Highlighted values indicate the best performance.

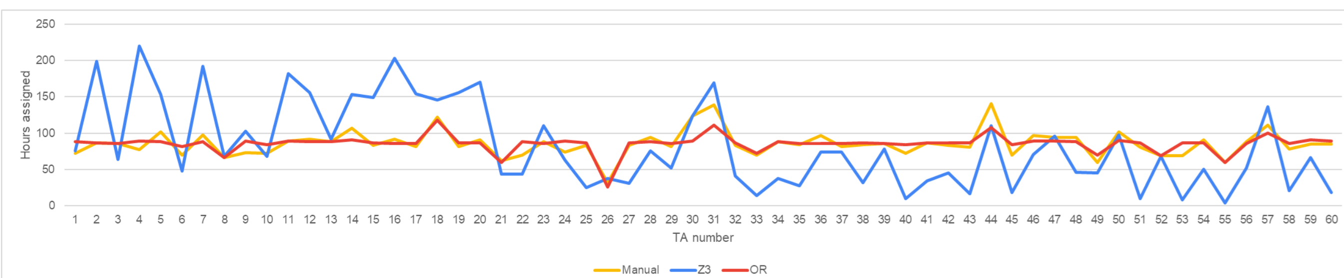

Fig. 3. Graph: TA number vs Assigned hours

same solver. To do this, we wanted to maximize TAs' "preference score," which is the sum of their preferences for each assigned session. So, if two sessions were assigned, where one had a preference of 1, and the other had a preference of -1, they would score 0.

We quickly realized that this system had a flaw. Many people were getting sessions they didn't want (preference score -1). This is because the algorithm now has two goals: maximize the score and achieve balance (minimize the sum of differences). To this end, if the algorithm encounters two sets of TAs, one with 0 and 0 and the other with -1 and 1, it would pick whichever contributes better to the balancing. As such, We also tried weighing it a bit differently; We left one as "give one point" and 0 as "give 0 points" but changed -1 to -2 and -10000 to see the effects it would have on the sessions given and the deviation. The results are in Table 2.

As expected, when -1s were given a score of -2, far fewer -1s were assigned, going from 100 to 9, and when the weight was essentially set to infinity, it went all the way down to 0 sessions. But the fascinating information here is the standard deviation, which was significantly lower than the previous method. This was a paradigm shift for us; until then, we had assumed that since the algorithm's maximum time remained constant, adding more variables would lead to a less optimized result. We had not considered the trade-off in flexibility; having more variables allows sessions previously removed by hard constraints to be viewed as possible sessions to achieve the algorithm goals. These new variables allowed us to create a schedule with a standard deviation of 4 while giving as few TAs as possible sessions they would not want.

	Deviation	Min	Max	#1s assigned	#0s assigned	#-1s assigned
Equal weight	5.38	68	111	1027	234	100
Weighted (1, 0, -2)	4.00	70	101	1256	96	9
Weighted (1, 0, $-\infty$)	7.34	70	118	1263	98	0

Table 2. TA preference assignment with different weighting methods

6.3 Iteration 3: Room Allocation

Another constraint that has yet to be discussed is the room allocation of TAs. Reflecting on how to implement this, we realized it could be disconnected from the rest of the algorithm. As such, we decided to implement this within the current algorithm and separately as its algorithm. Since this does not give the algorithm more flexibility like in Scenario 2, the results in Table 3 make sense for the reason mentioned earlier: the algorithm is given more variables to assign in the same amount of time, increasing complexity and thereby decreasing the solution quality it's able to produce.

However, with an external algorithm, we could use the best algorithm from the previous section and attempt to use it as the input for the room allocation algorithm. By doing this, we were guaranteed to receive the result of standard deviation four from Table 2.

The algorithm had two parts. First, it ensured a balance between rooms. This was done by creating a constraint such that the number of TAs per room must be greater than or equal to the number of TAs needed (integer) divided by the number of rooms. This ensured that no room had more than one TA more than any other.

We could then create an algorithm that maximizes the time a TA spends in their homeroom by saying for each session that needs a room allocated if a TA is assigned to that room and then maximizing the value of 'total_time_in_homeroom.' In this case, the best way to compare results is by calculating entropy; which refers to the imbalance in the allocation of sessions that TAs spend in their respective homerooms. Where entropy is calculated as:

$$H_{TA} = -\sum_{i=1}^{n} p_i \log(p_i)$$

where p_i is the proportion of time spent in room i, calculated as:

$$H_{TA} = -\left(\sum_{i=1}^{3}\left(\frac{K_i}{\sum_{j=1}^{3} K_j}\right)\log\left(\frac{K_i}{\sum_{j=1}^{3} K_j}\right)\right)$$

where K_i represents the time spent in room i. The overall entropy is then calculated as the average of these values across all TAs:

$$\forall_{TA}\, H = \frac{1}{N}\sum_{ta=1}^{N} H_{TA}$$

Where N is the total number of TAs. Using this calculation, we are able to find the results presented in Table 3. As can be seen, initially calculated as 0.4 with three rooms, the entropy dropped all the way to 0.1 following optimization efforts to maximize the total time TAs spend in their homerooms.

6.4 Scenario 2: Budget Based Preferences

For this experiment, we started by creating fictitious data, randomly generated, where for each TA for each session, the TA had a 50% chance they were available for the session. They would randomly generate a number to set as their budget if available. This led to values with extreme outliers, resulting in an overall lower standard deviation. However, this model cannot be judged by the same metrics as the rest since it was not built similarly. While other algorithms mainly focused on minimizing the difference between hours, this one focused on maximizing a new parameter, happiness values. Whether or not TAs would enjoy this system more is yet to be seen, but it allows for a different approach, giving a more human-centric algorithm rather than a mathematically balanced one. The results of this experiment will enable us to explore new approaches to finding what an optimized schedule truly means. It was interesting to see how combining the objectives would change the deviation, and it was able to come up with a significantly better deviation compared to the loss in happiness, as seen in Table 4.

6.5 OR-Tools time constraint

The OR-Tools experiments had a time constraint of 5 minutes to ensure termination in a reasonable amount of time. This decision was made after tests showed that running the solver for a day did not produce significantly better results. The results demonstrate that OR-Tools provide equally good outcomes within 5 minutes, as shown in Table 5. The results indicate that extending the time for OR-Tools beyond 5 minutes yields diminishing returns, confirming that a 5-minute constraint is practical and effective.

7 DISCUSSION

The results from our experiments offer significant insights into the efficacy of different scheduling methodologies and preference modeling approaches. This section discusses the implications of our findings, compares our approach with existing methods, and outlines potential areas for future work.

7.1 Comparison with Manual Scheduling

The comparison between our algorithmic approaches and manual scheduling reveals several advantages of the automated methods. Compared to the manual approach, the OR-Tools-based method produced schedules with higher preference scores and better balance among TAs. This suggests that algorithmic scheduling reduces the effort required and enhances the overall satisfaction of TAs by better aligning with their preferences, thereby aligning with the goal of this research.

	Average	Deviation	Range	IQR	Time	Avg entropy
Rooms integrated	85.57	20.87	113	25.5	5m	0.4
Rooms external (No homeroom)	85.57	4	31	3	>1s	0.4
Rooms external (Homeroom)	85.57	4	31	3	5m	0.1

Table 3. Comparison of Room Types

		Average	Deviation	Median	Min	Max	Range	IQR	Time Taken
No hour constraints, maximize budget	Happiness Values:	87.333 (80%)	13.343	85.5	61	110	49	19	1 second
	Hour Values:	85.567	34.481	84	14	147	133	58.75	
Hour constraints, max budget, equal weight	Happiness Values:	82.35 (74.86%)	19.646	76	36	110	74	32	5 minutes
	Hour Values:	85.567	21.989	94	18	108	90	14.25	

Table 4. Comparison of Happiness and Hour Values under Different Constraints

Time Interval	Average	Std. Deviation	Median	Min	Max	Range	IQR
5 minutes	85.5667	12.092	87	26	118	92	2.25
10 minutes	85.5667	12.058	87	26	118	92	1
30 minutes	85.5667	12.057	87	26	118	92	1
60 minutes	85.5667	12.057	87	26	118	92	1

Table 5. Results of OR-Tools with different time intervals

However, it is notable that manual scheduling occasionally resulted in better minimum workload assignments, showing constraints must be placed carefully to ensure that the metric for workload balancing is adequately defined. Throughout the experiments, we had the best results from keeping this metric as minimizing standard deviation.

7.2 Effectiveness of Z3 Solver

While powerful, the Z3 solver faces significant scalability issues with large-scale TA scheduling problems. The primary challenge is the exponential growth in the scheduling problem's search space, dramatically increasing complexity and computational requirements as the problem expands. In addition to this, the paper [1], which talks about vZ, which is a part of the Z3 solver, has a figure (15) that shows that even though the speed of the solver is better than other solvers in the 2014 MaxSAT competition, the amount of time taken still exponentially grows with the number of instances.

These factors limit Z3's effectiveness in handling large-scale TA scheduling problems, making it necessary to use solutions like OR-Tools, which provide a time-limiting function for such applications.

7.3 Advantages of OR-Tools

Implementing OR-Tools proved to be the most effective approach, providing feasible and optimal schedules within a reasonable timeframe. OR-Tools, developed by Google, is a powerful open-source software suite designed for solving combinatorial optimization problems. This makes sense, as according to both the OR site and the results of the MiniZinc competition (an annual constraint programming challenge) [13], OR-tools has won gold medals in all but one category every year since 2018. The only category in which OR-tools did not win gold was local search, as they weren't in that category.

OR-Tools excels in handling large-scale scheduling problems due to several key features. The CP-SAT solver efficiently handles constraint satisfaction problems by combining constraint programming with Boolean satisfiability techniques. This solver is highly effective for complex scheduling tasks due to its advanced features like lazy clause generation and conflict-driven learning. Lazy clause generation allows the solver to delay the creation of certain constraints until they are essential. This helps reduce the initial problem size and focus computational resources on the most promising parts of the search space. Conflict-driven learning, on the other hand, involves the solver learning from conflicts encountered during the search, avoiding similar disputes in the future. This significantly speeds up the solving process by pruning large portions of the search space that do not lead to feasible solutions [3].

Additionally, OR-Tools is designed to handle large datasets efficiently. The solver uses advanced techniques to find high-quality solutions within a limited time frame. Even when a problem instance is too large to be solved optimally within the given time, OR-Tools can produce a near-optimal solution quickly. This is achieved through sophisticated heuristics and optimization algorithms that focus on the most relevant parts of the problem first, ensuring that the most critical constraints are satisfied early in the process. This capability is handy for practical applications where solutions must be generated promptly, even if they aren't optimal [4].

Compared to Z3, which struggles with scalability due to the exponential growth of the search space, OR-Tools is better suited for large-scale and complex scheduling problems. While Z3's exponential solve time becomes inefficient with large datasets, OR-Tools' CP-SAT solver is designed to manage these challenges through techniques like lazy clause generation and conflict-driven learning.

This allows OR-Tools to maintain high performance and scalability, making it a better solution for large optimization problems.

These factors collectively highlight why OR-Tools is superior for large-scale TA scheduling problems. It offers a robust, flexible solution that outperforms Z3 in terms of scalability and efficiency.

7.4 Preference Modeling Techniques

Our study explored various preference modeling techniques, each with its unique advantages and limitations. While simple and easy to implement, the binary preference model lacks the granularity needed to capture varying degrees of preference, potentially leading to less satisfactory schedules for TAs. Fixed-preference scoring offers more nuanced input but can still be limiting in complex scenarios where TAs have high preferences for multiple sessions.

Inspired by our experiences and discussions with fellow TAs, the budget-based preference model allows for a more flexible and detailed representation of TA preferences. This model emerged as a potential solution due to the diverse opinions gathered from our poll on the importance of different soft constraints. Our simulations showed promising results, indicating that this model could significantly improve TA satisfaction and scheduling quality by giving each one a happiness score of over 80%, as we saw in Table 4.

While our initial research and simulations suggest the budget-based model's potential, further research and practical implementation are needed to evaluate its effectiveness fully. Future deployments should consider adopting this model to enhance scheduling quality and better accommodate the varying preferences of TAs.

7.5 Implications for Practice

The findings have several practical implications for institutions looking to improve their scheduling processes. Automated scheduling tools like OR-Tools can lead to more satisfactory schedules, reduce administrative burdens, and enhance TA morale. Adopting flexible preference modeling techniques can further improve the alignment between TA assignments and individual preferences. Furthermore, consider whether a constraint adds or limits flexibility when considering the choice of constraints. As previously mentioned, when we added the constraint of TA preferences, the standard deviation decreased by about half, but the room constraint nearly doubled. In cases like this, it's advisable to check if it is possible to use the solver for intermediate results; as we have demonstrated, solving two more minor problems can be significantly less work than solving one large one. Finally, keep in mind the size of the space, as more minor issues may be doable with various libraries, but for larger ones, some, such as Z3, are very inefficient.

7.6 Future Work

While this study comprehensively analyzes TA scheduling optimization, several areas warrant further investigation. Future work could explore hybrid models that integrate human decision-making with algorithmic optimization. Additionally, extending the research to other types of scheduling problems, such as student or staff scheduling or even non-academic scheduling, could provide broader insights into the applicability of these methods. Lastly, user feedback and satisfaction surveys could provide valuable data to refine further and validate the proposed methodologies.

8 CONCLUSION

This research presents a thorough approach to optimizing teaching assistant (TA) scheduling using constraint-solving techniques and advanced preference modeling. Implementing SAT-SMT solvers, particularly Google OR-Tools, has significantly improved efficiency and satisfaction over manual methods.

The experiments highlight key findings. Z3, despite its power, struggles with scalability, taking over 48 hours without producing feasible results. In contrast, Google OR-Tools consistently generated high-quality schedules within a 5-minute timeframe. The OR-Tools' CP-SAT solver handled complex constraints effectively, resulting in schedules with an average deviation of just 12.09 hours compared to Z3's 58.43 hours.

Various preference modeling techniques were explored, with the budget-based preference model proving the most effective. This model allowed TAs to distribute preference points, significantly improving satisfaction scores. For example, weighting preferences reduced the number of unpreferred (-1) sessions from 100 to 9.

Room allocation, handled separately, also improved outcomes. The integrated approach nearly doubled the standard deviation, highlighting the benefits of solving complex problems in stages.

Practical implications are significant: automated OR-Tools scheduling reduces administrative burdens and improves TA satisfaction and workload balance. For instance, OR-Tools reduced the deviation in workload hours to 4.00 compared to manual scheduling's 17.91.

Future research should extend these methods to other scheduling contexts and incorporate user feedback for further refinement. Integrating machine learning could also enhance the personalization of scheduling solutions.

In conclusion, leveraging advanced constraint-solving and preference modeling techniques optimizes TA scheduling, enhancing efficiency, satisfaction, and workload balance, thus contributing to a more effective educational environment.

ACKNOWLEDGMENTS

Firstly, we thank Tom van Dijk for his guidance and support throughout the research process. Furthermore, we thank the 24 anonymous TAs who helped complete our survey.

DATA AVAILABILITY STATEMENT

The data supporting the findings of this study are available at Zenodo: https://zenodo.org/records/12508669. The data have been anonymized to protect the privacy of individuals.

TOOLS USED

During the preparation of this work, the author used Grammarly in order to enhance clarity and ensure grammatical accuracy. After using this tool, the author reviewed and edited the content as needed and takes full responsibility for the content of the work.

REFERENCES

[1] Nikolaj Bjørner, Anh-Dung Phan, and Lars Fleckenstein. 2015. νZ - An Optimizing SMT Solver. In *Tools and Algorithms for the Construction and Analysis*

of Systems, Christel Baier and Cesare Tinelli (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 194–199. https://doi.org/10.1007/978-3-662-46681-0_14
[2] Leonardo de Moura and Nikolaj Bjørner. 2008. Z3: An Efficient SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, C. R. Ramakrishnan and Jakob Rehof (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 337–340. https://doi.org/10.1007/978-3-540-78800-3_24
[3] Google. 2024. Google Optimization Tools (OR-Tools). https://github.com/google/or-tools. Accessed: 2024-06-21.
[4] Dr. Dominik Krupke. 2024. cpsat-primer. https://github.com/d-krupke/cpsat-primer. Accessed: 2024-06-21.
[5] Mahdi Mobini. 2024. Solving the nurse rostering problem using Google or-tools. https://medium.com/@mobini/solving-the-nurse-rostering-problem-using-google-or-tools-755689b877c0
[6] Chong Man Ngoo, Say Leng Goh, Nasser Sabar, San Sze, Salwani Abdullah, and Graham Kendall. 2022. A Survey of the Nurse Rostering Solution Methodologies: The State-of-the-Art and Emerging Trends. *IEEE Access* 10 (01 2022), 1–1. https://doi.org/10.1109/ACCESS.2022.3177280
[7] Mizuhito Ogawa and To Van Khanh. 2013. SAT and SMT: Their Algorithm Designs and Applications. In *2013 20th Asia-Pacific Software Engineering Conference (APSEC)*, Vol. 2. 83–84. https://doi.org/10.1109/APSEC.2013.118
[8] Miquel Palahí i Sitges et al. 2015. Reformulation of constraint models into SMT. (2015).
[9] Michael Pinedo and Khosrow Hadavi. 1992. Scheduling: theory, algorithms and systems development. In *Operations Research Proceedings 1991: Papers of the 20th Annual Meeting/Vorträge der 20. Jahrestagung*. Springer, 35–42. https://doi.org/10.1007/978-3-031-05921-6
[10] Ronghai Qu and Fang He. 2009. *A hybrid constraint programming approach for nurse rostering problems.* 211–224 pages. https://doi.org/10.1007/978-1-84882-215-3_16
[11] Sabino Francesco Roselli, Kristofer Bengtsson, and Knut Åkesson. 2018. SMT solvers for job-shop scheduling problems: Models comparison and performance evaluation. In *2018 IEEE 14th international conference on automation science and engineering (CASE)*. IEEE, 547–552. https://doi.org/10.1109/COASE.2018.8560344
[12] Francesca Rossi, Peter van Beek, and Toby Walsh (Eds.). 2006. *Handbook of Constraint Programming*. Foundations of Artificial Intelligence, Vol. 2. Elsevier. https://www.sciencedirect.com/science/bookseries/15746526/2
[13] G. Tack, P. Stuckey, et al. 2024. The MiniZinc Challenge. https://www.minizinc.org/challenge.html Accessed: 2024-05-10.