# Report on Assessment Timetabling Using Z3 Solver with Automated Conflict Detection and Resolution Suggester

## A) Introduction

This report presents the implementation and solution of an assessment timetabling problem using the Z3 SMT (Satisfiability Modulo Theories) solver in Python. Z3, developed by Microsoft Research, is a high-performance theorem prover that supports various logical theories, making it suitable for solving complex constraint satisfaction problems. Python's integration with Z3 through its API provides a flexible platform for modeling and solving such problems.

The assessment timetabling problem involves scheduling exams for students while satisfying multiple constraints related to room capacities, time slots, student exam schedules, and invigilator assignments. In addition to the standard constraints, two additional constraints are incorporated:

1. **Invigilator Assignment Constraint**: Ensure each exam has exactly one assigned invigilator.
2. **Invigilator Uniqueness Constraint**: Ensure each invigilator is assigned to only one unique exam.

An **Automated Conflict Detection and Resolution Suggester** feature has been implemented as an extraordinary enhancement to the solver's usability. This feature identifies conflicting constraints when the solver finds the problem unsatisfiable and suggests possible solutions to these conflicts for the user's consideration. Additionally, a visual indicator (a rotating spinner using the Halo library) has been added to inform the user that the solver is processing, ensuring the software appears responsive.

Additionally, a **Halo Loading Spinner** has been integrated to provide real-time feedback during the computation process. This spinner visually informs the user whether the code is still running, thereby preventing confusion about whether the computer has frozen or the solver is stuck. This improvement significantly enhances the user experience, especially for complex test instances where the solving process may take longer.

The purpose of this report is to detail the problem formulation, discuss alternative approaches, explain the implementation with the new features, evaluate the solver's performance using provided test instances, and provide insights based on the experimental results.

## B) Formulation of the Problems

The assessment timetabling problem is formulated as a Constraint Satisfaction Problem (CSP). The goal is to assign exams to rooms and time slots, assign invigilators to exams, and schedule students for exams while satisfying all constraints.

**Variables and Functions**

- **Exams (E)**: Set of exams to be scheduled.
- **Students (S)**: Set of students taking exams.
- **Rooms (R)**: Set of available rooms.
- **Time Slots (T)**: Set of available time slots.
- **Invigilators (I)**: Set of available invigilators.

**Functions:**

- **ExamRoom(e)**: Assigns a room to exam *e*.
- **ExamTime(e)**: Assigns a time slot to exam *e*.
- **ExamStudent(e, s)**: Indicates if student *s* is taking exam *e*.
- **ExamInvigilator(e)**: Assigns an invigilator to exam *e*.

**Constraints**

1. **Room and Time Slot Assignment Constraint**:
    - Each exam must be scheduled in exactly one room and one time slot.
    - $\forall e \in E, \exists! \, r \in R, \exists! \, t \in T$ such that ExamRoom($e$) = $r$ and ExamTime($e$) = $t$.
2. **Room Occupancy Constraint**:
    - No two exams are scheduled in the same room at the same time.
    - $\forall e_1, e_2 \in E$, if $e_1 \neq e_2$ and ExamRoom($e_1$) = ExamRoom($e_2$) and ExamTime($e_1$) = ExamTime($e_2$), then False.
3. **Room Capacity Constraint**:
    - The number of students taking an exam does not exceed the room capacity.
    - $\forall e \in E$, student_count($e$) ≤ room_capacity(ExamRoom($e$)).
4. **Student Scheduling Constraint**:

- A student cannot have exams in consecutive time slots.
- $\forall s \in S$, $\forall e_1, e_2 \in E$, if ExamStudent($e_1$, $s$) and ExamStudent($e_2$, $s$) and $e_1 \neq e_2$, then |ExamTime($e_1$) - ExamTime($e_2$)| > 1.

5. **Invigilator Assignment Constraint**:
   - Each exam has exactly one assigned invigilator.
   - $\forall e \in E$, $\exists! \, i \in I$ such that ExamInvigilator($e$) = $i$.
6. **Invigilator Uniqueness Constraint**:
   - Each invigilator is assigned to only one unique exam.
   - $\forall e_1, e_2 \in E$, if $e_1 \neq e_2$ and ExamInvigilator($e_1$) = ExamInvigilator($e_2$), then False.

## C) Alternative Formulations and Solutions

The assessment timetabling problem is inherently a Constraint Satisfaction Problem (CSP), but there are alternative approaches to model and solve it, offering potential advantages in scalability, efficiency, and structured constraint management. Two of the most notable methods are Graph Coloring and Integer Linear Programming (ILP).

---

1. Graph Coloring Approach

Graph coloring is a natural way to model scheduling problems where conflicts between entities need to be avoided.

Representation

- Nodes: Each node represents an exam.
- Edges: An edge connects two nodes if the corresponding exams share common constraints. For example:
  - Exams have overlapping students.
  - Exams cannot be scheduled simultaneously due to room constraints.
- Colors: Colors represent time slots. Assigning a "color" to a node means scheduling the corresponding exam in a specific time slot.

Objective

The objective is to assign a color (time slot) to each node (exam) such that:

1. No two connected nodes (conflicting exams) share the same color (time slot).
2. Room capacities and invigilator assignments are incorporated as additional constraints.

Key Advantages

- Visualization: The graph structure provides a clear, visual representation of conflicts, making it easier to understand the relationships between exams.
- Algorithm Availability: Several efficient graph coloring algorithms (e.g., Greedy Coloring, Backtracking, Welsh-Powell) can be adapted to the problem.
- Scalability: This approach scales well for problems with many exams but relatively few conflicts, as sparse graphs are computationally easier to handle.

Enhancements for Realism

1. Weighted Graphs:
    - Assign weights to edges based on the number of shared students between exams, prioritizing heavily conflicted exams.
2. Multi-coloring:
    - Use different types of colors to represent both time slots and room assignments.

Challenges

- Complex Constraints: Incorporating room capacities and invigilator assignments into a pure graph model can be complex.
- Large Graphs: Highly interconnected graphs with dense edges may require significant computational effort.

Algorithm Implementation

A potential algorithmic pipeline for graph coloring in this context could involve:

1. Graph Construction: Build a graph where nodes and edges represent exams and their conflicts.
2. Color Assignment:

- Use a Greedy Coloring algorithm to assign colors based on the degree of conflicts.

3. Constraint Handling: Post-process to handle additional constraints, such as room capacities, by splitting or merging nodes as needed.

2. Integer Linear Programming (ILP)

Integer Linear Programming provides a more structured and mathematical approach to the problem, especially for handling numerical constraints like room capacities and invigilator availability.

**Modeling**

ILP formulates the problem as an optimization problem with integer variables, a linear objective function, and a set of linear constraints.

**Variables**

1. **Exam-Time Slot Assignment**:

   $X_{et} \in \{0,1\}$ Binary variable indicating whether exam e is scheduled in time slot t.

2. **Exam-Room Assignment**:

   $y_{er} \in \{0,1\}$
   Binary variable indicating whether exam e is scheduled in room r.

3. **Invigilator Assignment**:

   $Z_{ei} \in \{0,1\}$ Binary variable indicating whether invigilator i is assigned to exam e.

---

**Objective Function**

Minimize the total cost of scheduling conflicts, such as:

- Overlapping exams for students.
- Over-utilization of room capacities.
- Overloading invigilators.

---

**Constraints**

1. **Time Slot Uniqueness**:
   Each exam is assigned to exactly one time slot:

$$\sum_{t} x_{et} = 1, \forall e$$

   **Room Capacity**:
   The number of students assigned to a room does not exceed its capacity:

$$\sum_{e} y_{er} \cdot student\_count(e) \leq capacity(r), \forall r$$

   **Invigilator Assignment**:

   - Each exam must have exactly one invigilator:

$$\sum_{e} y_{er} \cdot student\_count(e) \leq capacity(r), \forall r$$

   - Each invigilator can oversee only one exam at a time:

$$\sum_{t} z_{ei} \cdot x_{et} \leq 1, \forall i, t$$

2. **Student Overlap**:
   Students cannot have overlapping exams:

$$X_{et1} + X_{et2} \leq 1, \forall_{e1,e2} \text{ with shared students and } t1 = t2$$

Advantages

- Exact Solutions: ILP guarantees an optimal solution (within computational limits).
- Flexibility: Constraints can be added or adjusted easily.
- Structured Handling: The mathematical formulation naturally handles numerical constraints like capacities and overlaps.

Challenges

- Computational Complexity: ILP solvers can become computationally expensive for large instances.
- Solver Dependency: Requires advanced solvers like Gurobi or CPLEX, which may have licensing costs.

Example Solver

Using Gurobi, the workflow would be:

1. Define the variables ($x_{et}, y_{er}, z_{ei}$).
2. Add constraints as linear equations or inequalities.
3. Specify the objective function (e.g., minimize scheduling conflicts).
4. Run the solver to obtain the optimal solution.

---

Comparative Analysis

| Aspect | Graph Coloring Approach | Integer Linear Programming (ILP) |
|---|---|---|
| Ease of Implementation | Simpler to implement for basic conflicts. | Requires a mathematical formulation. |
| Scalability | Scales well for sparse graphs. | May struggle with very large problems. |
| Handling Complex Constraints | More challenging to incorporate room capacities or invigilators. | Handles numerical constraints naturally. |
| Optimality | Heuristic-based, not guaranteed to find optimal solutions. | Guaranteed optimal solutions (given sufficient resources). |
| Flexibility | Limited to scheduling conflicts. | Highly flexible for diverse constraints. |
| Solver Availability | Requires a graph library (e.g., NetworkX). | Requires advanced solvers like Gurobi. |

---

Potential Hybrid Approach

A hybrid approach combining Graph Coloring and ILP could offer the best of both methods:

1. Preprocessing with Graph Coloring:
    - Use graph coloring to assign initial time slots and reduce the problem complexity.
2. Optimization with ILP:
    - Use ILP to fine-tune the schedule, incorporating additional constraints like room capacities and invigilator assignments.

This approach leverages the speed of graph algorithms for initial conflict resolution and the precision of ILP for detailed optimization.

**Conclusion for Alternative formulation and solution**

Both Graph Coloring and ILP offer viable alternative formulations for the assessment timetabling problem. Graph Coloring excels in visualization and simplicity, making it suitable for quick conflict resolution. ILP, on the other hand, provides a rigorous and flexible framework for handling complex constraints. Depending on the problem size and the importance of exact solutions, either approach—or a hybrid method—can be chosen to address the timetabling challenge effectively.

## D) Implementation

The solver is implemented in Python using the Z3 API. Key enhancements and features include the **Automated Conflict Detection and Resolution Suggester**, the option for users to choose between single or multiple solutions, and a visual processing indicator using a spinner.

**1. Reading Input Data**

- **Parsing Input Files**: The solver reads from input files provided, which contain the number of students, exams, slots, rooms, invigilators, room capacities, and exam-student mappings.

**2. Variable and Function Declarations**

- **Variables**: Declare variables for exams, rooms, time slots, students, and invigilators.

- **Functions**: Define functions (ExamRoom, ExamTime, ExamInvigilator) representing assignments.

**3. Defining Ranges**

- **Constraints on Variables**: Add constraints to specify valid ranges for variables (e.g., exam indices between 0 and number_of_exams - 1).

**4. Adding Constraints**

- **Implementing Constraints**: Implement constraints as per the problem formulation using Z3's syntax.
- **Use of Quantifiers**: Use ForAll, Exists, and logical connectors (And, Or, Implies) to express constraints.

**5. Automated Conflict Detection and Resolution Suggester**

The Automated Conflict Detection and Resolution Suggester feature is an extraordinary enhancement to the solver's robustness, enabling it to handle unsatisfiable instances intelligently. This feature provides meaningful feedback to the user by identifying conflicts and suggesting feasible adjustments. Unlike a static suggestion system, the software dynamically validates the suggested resolutions, ensuring their viability before presenting them to the user.

---

A. Conflict Detection

1. Unsatisfiable Core:
    - When the solver determines that the problem is unsatisfiable (unsat), it retrieves the unsatisfiable core using s.unsat_core().
    - The unsatisfiable core provides a minimal set of constraints that cannot be satisfied together, pinpointing the root cause of the conflict.
2. Identifying Conflicts:
    - The software analyzes the unsatisfiable core to determine which specific constraints are in conflict, such as:
        - Invigilator assignment constraints.
        - Room capacity constraints.
        - Scheduling conflicts due to insufficient time slots.

---

B. Conflict Resolution Suggestions

The software employs dynamic Adjustment Strategies to resolve conflicts based on the constraints identified in the unsatisfiable core. These suggestions are tailored to the nature of the conflict and validated for feasibility before they are presented.

1. Increase Number of Invigilators:
    - If conflicts involve invigilator constraints, the software suggests increasing the number of invigilators to match the number of exams.
    - Validation: The software checks if the increase resolves the conflict. If not, it iteratively adjusts the number of invigilators until a satisfiable solution is found.
2. Increase Room Capacities:
    - If conflicts involve room capacity constraints, the software suggests increasing room capacities to accommodate the largest exam.
    - Validation: The solver verifies whether the adjusted capacities resolve the conflict. If the conflict persists, the software further adjusts the room capacities iteratively.
3. Increase Number of Time Slots:
    - If conflicts involve scheduling constraints, the software suggests increasing the number of time slots to create more opportunities for non-overlapping schedules.
    - Validation: The solver ensures that the additional time slots resolve the conflict. If not, the software continues to increment the number of time slots until a feasible solution is identified.

---

C. Validation of Suggested Solutions

A key enhancement of this feature is the iterative validation of suggested solutions:

- Dynamic Adjustment:
    - The software not only suggests a resolution but also validates its feasibility by re-running the solver after applying the suggested changes.
    - If the adjusted problem remains unsatisfiable, the software iteratively modifies the suggestion until the problem becomes satisfiable.

- Output:
    - Once a satisfiable solution is found, the software prints:
        1. The resolved timetable, including exam schedules, room assignments, time slots, and invigilator allocations.
        2. The final validated suggestion, detailing the adjustments made (e.g., increased number of invigilators, additional time slots, or expanded room capacities).

---

D. User Consideration

1. No Automatic Adjustments:
    - The solver does not alter the original test instance directly. Instead, it provides validated suggestions for the user to consider.
2. Unsatisfiable Results:
    - If the problem is inherently unsolvable due to hard constraints that cannot be relaxed (e.g., zero-capacity rooms), the software informs the user with an explanation of the conflicting constraints.
3. Iterative Approach:
    - The iterative process ensures that the final suggestion and timetable provided are both valid and practical, reducing the burden on users to manually test and adjust the problem setup.

---

Enhanced User Experience

This approach significantly improves the usability and reliability of the solver:

- Confidence in Suggestions: Users receive actionable, validated suggestions that are guaranteed to resolve the conflict.
- Dynamic Adjustments: The iterative validation process ensures that users do not encounter repeated unsatisfiable results after applying the suggested changes.
- Comprehensive Output: By providing both the adjusted problem setup and the final resolved timetable, users can directly implement the solution without further intervention.

This feature exemplifies the solver's capacity to adapt to complex real-world constraints while maintaining a high level of user-friendliness and practical utility.

**6. User Choice for Solutions**

- **Single or Multiple Solutions**: The user is given a choice to select whether they want a single solution or multiple solutions (up to 10 different solutions).
- **Implementation**: If multiple solutions are selected, the solver iteratively finds up to 10 unique solutions that satisfy all constraints.

**7. Visual Processing Indicator**

- **Spinner Implementation**: A rotating spinner using the Halo library has been added to indicate that the solver is processing.
- **User Assurance**: This ensures the user that the software has not crashed and is working on finding a solution.

**8. Solving the Problem**

- **Invoke Solver**: Use s.check() to determine satisfiability.
- **Extracting Model**: If satisfiable, extract the model using s.model().

**9. Outputting Results**

- **Display Results**:
    - Print whether the problem is satisfiable.
    - If satisfiable, display the exam schedules (exam, room, slot, invigilator).
    - If unsatisfiable, display the conflicting constraints and the suggested resolutions.
- **Output to File**:
    - The final results are also written to a file named output.txt for easy access and further review.
    - This file contains:
        - A summary of whether the problem was satisfiable or not.
        - The full timetable (if satisfiable), including room assignments, time slots, and invigilator assignments.
        - Suggested adjustments and resulting resolutions (if applicable).
- **Multiple Solutions**: If multiple solutions are selected, up to 10 different solutions are displayed.

- **Suggestions**: If unsatisfiable, display the conflicting constraints and the suggested resolutions.

**Known Bugs and Limitations**

- **Symbolic Outputs**: In some cases, the solver may output symbolic expressions instead of concrete values after conflict resolution.
- **Scalability**: The implementation may not scale well for very large instances due to the use of quantifiers.
- **Performance**: The solver takes approximately 7 minutes to check for satisfiability, produce solutions, recommend suggestions for unsatisfiable instances, check if suggestions can result in a satisfiable solution, and produce the solution if the suggested change is viable.

**Known Bugs and Limitations**

- **Symbolic Outputs**: In some cases, the solver may output symbolic expressions instead of concrete values after conflict resolution.
- **Scalability**: The implementation may not scale well for very large instances due to the use of quantifiers.
- **Performance**: Performance may degrade with increased problem complexity.

## E) Evaluation

The solver was tested on several sample instances provided, with varying sizes and complexities. These instances include both satisfiable and unsatisfiable cases. Users were given the option to select single or multiple solutions. When multiple solutions were selected, the software provided up to 10 different solutions instead of just one.

**Sample Instances and Results**

**Satisfiable Instances**

1. **sat1.txt**
    - **Parameters**: 3 students, 1 exam, 1 slot, 1 room (capacity 3), 4 invigilator.
    - **Result**: Satisfiable.

- **Output**:

```
========================================
Processing sat1.txt:
========================================
Time Taken to solve this instance: 5 ms
sat
Solution 1:
    Exam: 0  Room: 0  Slot: 0  Invigilator: 0
```

---

2. **sat2.txt**
    - **Parameters**: 3 students, 2 exams, 3 slots, 1 room (capacity 2), 4 invigilators.
    - **Result**: Satisfiable.
    - **Output**:

```
========================================
Processing sat2.txt:
========================================
Time Taken to solve this instance: 2 ms
sat
Solution 1:
    Exam: 0  Room: 0  Slot: 0  Invigilator: 0
    Exam: 1  Room: 0  Slot: 2  Invigilator: 1
```

---

3. **sat3.txt**
    - **Parameters**: 3 students, 2 exams, 2 slots, 1 room (capacity 2), 4 invigilators.
    - **Result**: Satisfiable.
    - **Output**:

```
========================================
Processing sat3.txt:
========================================
Time Taken to solve this instance: 1 ms
sat
Solution 1:
    Exam: 0  Room: 0  Slot: 0  Invigilator: 0
    Exam: 1  Room: 0  Slot: 1  Invigilator: 1
```

---

4. **sat4.txt**
    - **Parameters**: 3 students, 2 exams, 1 slot, 2 rooms (capacities 1 and 2), 4 invigilators.
    - **Result**: Satisfiable.
    - **Output**:

```
========================================
Processing sat4.txt:
========================================
Time Taken to solve this instance: 2 ms
sat
Solution 1:
    Exam: 0  Room: 1  Slot: 0  Invigilator: 0
    Exam: 1  Room: 0  Slot: 0  Invigilator: 1
```

---

5. **sat5.txt**
    - **Parameters**: 4 students, 2 exams, 2 slots, 1 room (capacity 5), 4 invigilators.
    - **Result**: Satisfiable.
    - **Output**:

```
========================================
Processing sat5.txt:
========================================
Time Taken to solve this instance: 1 ms
sat
Solution 1:
    Exam: 0  Room: 0  Slot: 0  Invigilator: 0
    Exam: 1  Room: 0  Slot: 1  Invigilator: 1
```

6. **sat6.txt**
    - **Parameters**: 9 students, 3 exams, 6 slots, 5 rooms with varying capacities, 4 invigilators.
    - **Result**: Satisfiable.
    - **Output**:

```
========================================
Processing sat6.txt:
========================================
Time Taken to solve this instance: 4 ms
sat
Solution 1:
    Exam: 0  Room: 2  Slot: 4  Invigilator: 2
    Exam: 1  Room: 0  Slot: 2  Invigilator: 1
    Exam: 2  Room: 1  Slot: 0  Invigilator: 0
```

---

7. **sat8.txt**
    - **Parameters**: 10 students, 3 exams, 5 slots, 4 rooms with varying capacities, 4 invigilators.
    - **Result**: Satisfiable.
    - **Output**:

```
========================================
Processing sat8.txt:
========================================
Time Taken to solve this instance: 9 ms
sat
Solution 1:
    Exam: 0  Room: 1  Slot: 4  Invigilator: 0
    Exam: 1  Room: 0  Slot: 2  Invigilator: 1
    Exam: 2  Room: 2  Slot: 0  Invigilator: 2
```

**Unsatisfiable Instances with Suggested Conflict Resolution**

1. **sat7.txt**
    - **Parameters:** 14 students, 5 exams, 5 slots, 4 rooms with varying capacities, 4 invigilators.

- **Result:** Unsatisfiable.
- **Conflicting Constraints Detected:**
  - Each exam has exactly one assigned invigilator.
  - Each invigilator is assigned to only one unique exam.
- **Suggested Resolution :**
  - Increased number of invigilators to 5 (matching the number of exams).
- **Suggestion's Result:** Conflicts resolved.
- **Output:**

```
========================================
Processing sat7.txt:
========================================
Time Taken to solve this instance: 27 ms
unsat
Conflicting constraints detected:
  - Each invigilator can be assigned to only one exam.

Suggested adjustments to resolve conflicts:
  - Increased number of invigilators to 5.
Suggested changes resolved conflict successfully.

Solution 1:
  Exam: 0  Room: 0  Slot: 4  Invigilator: 2
  Exam: 1  Room: 3  Slot: 0  Invigilator: 0
  Exam: 2  Room: 0  Slot: 2  Invigilator: 4
  Exam: 3  Room: 2  Slot: 0  Invigilator: 1
  Exam: 4  Room: 1  Slot: 1  Invigilator: 3
```

---

2. **sat9.txt**
   - **Parameters:** 24 students, 5 exams, 10 slots, 4 rooms with varying capacities, 4 invigilators.
   - **Result:** Unsatisfiable**.**
   - **Conflicting Constraints Detected:**
     - Each exam has exactly one assigned invigilator.
     - Each invigilator is assigned to only one unique exam.
   - **Suggested Resolution Attempt:**

- Increased number of invigilators to 5.

- **Suggestion's Result:** Conflicts resolved.
- **Output:**

```
========================================
Processing sat9.txt:
========================================
Time Taken to solve this instance: 97 ms
unsat
Conflicting constraints detected:
  - Each invigilator can be assigned to only one exam.

Suggested adjustments to resolve conflicts:
  - Increased number of invigilators to 5.
Suggested changes resolved conflict successfully.

Solution 1:
  Exam: 0  Room: 0  Slot: 8  Invigilator: 4
  Exam: 1  Room: 2  Slot: 0  Invigilator: 1
  Exam: 2  Room: 1  Slot: 2  Invigilator: 0
  Exam: 3  Room: 1  Slot: 6  Invigilator: 2
  Exam: 4  Room: 2  Slot: 4  Invigilator: 3
```

---

3. **sat10.txt**
   - **Parameters:** 28 students, 6 exams, 10 slots, 5 rooms with varying capacities, 4 invigilators.
   - **Result:** Unsatisfiable.
   - **Conflicting Constraints Detected:**
     - Each exam has exactly one assigned invigilator.
     - Each invigilator is assigned to only one unique exam.
   - **Suggested Resolution Attempt:**
     - Increased number of invigilators to 6.
   - **Suggestion's Result:** Conflicts resolved.
   - **Output:**

```
========================================
Processing sat10.txt:
========================================
Time Taken to solve this instance: 108 ms
unsat
Conflicting constraints detected:
  - Each invigilator can be assigned to only one exam.

Suggested adjustments to resolve conflicts:
  - Increased number of invigilators to 6.
Suggested changes resolved conflict successfully.

Solution 1:
   Exam: 0  Room: 1  Slot: 7  Invigilator: 5
   Exam: 1  Room: 1  Slot: 5  Invigilator: 0
   Exam: 2  Room: 0  Slot: 9  Invigilator: 1
   Exam: 3  Room: 0  Slot: 2  Invigilator: 2
   Exam: 4  Room: 1  Slot: 0  Invigilator: 3
   Exam: 5  Room: 2  Slot: 3  Invigilator: 4
```

**Additional Unsatisfiable Instances with Conflict Resolution**

**1. unsat1.txt**

- **Parameters**: 3 students, 1 exam, 4 slots, 2 rooms (capacities 1 and 2), 4 invigilator.
- **Result:** Unsatisfiable.
- **Conflicting Constraints Detected:**
  - Room capacity constraints for Exam 0 in both rooms.
  - Each exam must be timetabled in exactly one room and slot, and no two exams share a room and slot.
- **Suggested Resolution Attempt:**
  - Increased room capacities to accommodate all students.
- **Suggestion's Result:** Conflicts resolved.
- **Output:**

```
========================================
Processing unsat1.txt:
========================================
Time Taken to solve this instance: 2 ms
unsat
Conflicting constraints detected:
  - Exam times or rooms are out of range.
  - Room capacity exceeded.

Suggested adjustments to resolve conflicts:
  - Increased capacity of room 0 by 5.
  - Increased capacity of room 1 by 5.
  - Increased number of time slots by 1.
Suggested changes resolved conflict successfully.

Solution 1:
    Exam: 0  Room: 0  Slot: 0  Invigilator: 0
```

---

**2. unsat2.txt**

- **Parameters:** 3 students, 2 exams, 2 slots, 1 room (capacity 1), 4 invigilators.
- **Result:** Unsatisfiable.
- **Conflicting Constraints Detected:**
    - Room capacity constraint for Exam 0 in room 0.
    - Each exam must be timetabled in exactly one room and slot.
- **Suggested Resolution Attempt:**
    - Increased room capacity to accommodate the largest exam.
- **Suggestion's Result:** Conflicts resolved.
- **Output:**

```
========================================
Processing unsat2.txt:
========================================
Time Taken to solve this instance: 4 ms
unsat
Conflicting constraints detected:
  - Exam times or rooms are out of range.
  - Room capacity exceeded.

Suggested adjustments to resolve conflicts:
  - Increased capacity of room 0 by 5.
  - Increased number of time slots by 1.
Suggested changes resolved conflict successfully.

Solution 1:
  Exam: 0  Room: 0  Slot: 0  Invigilator: 0
  Exam: 1  Room: 0  Slot: 2  Invigilator: 1
```

---

**3. unsat3.txt**

- **Parameters:** 15 students, 5 exams, 7 slots, 3 rooms with varying capacities, 4 invigilators.
- **Result:** Unsatisfiable.
- **Conflicting Constraints Detected:**
  - Room capacity constraints for Exam 0 in all rooms.
  - Each exam must be timetabled in exactly one room and slot.
- **Suggested Resolution Attempt:**
  - Increased room capacities to accommodate the largest exam.
- **Suggestion's Result: Conflicts resolved.**
- **Output:**

```
========================================
Processing unsat3.txt:
========================================
Time Taken to solve this instance: 85 ms
unsat
Conflicting constraints detected:
  - Exam times or rooms are out of range.
  - Each invigilator can be assigned to only one exam.
  - Room capacity exceeded.

Suggested adjustments to resolve conflicts:
  - Increased capacity of room 0 by 10.
  - Increased capacity of room 1 by 10.
  - Increased capacity of room 2 by 10.
  - Increased number of time slots by 2.
  - Increased number of invigilators to 5.
Suggested changes resolved conflict successfully.

Solution 1:
   Exam: 0  Room: 2  Slot: 8  Invigilator: 4
   Exam: 1  Room: 1  Slot: 6  Invigilator: 0
   Exam: 2  Room: 1  Slot: 0  Invigilator: 1
   Exam: 3  Room: 0  Slot: 4  Invigilator: 3
   Exam: 4  Room: 2  Slot: 2  Invigilator: 2
```

---

**4. unsat4.txt**

- **Parameters:** 16 students, 4 exams, 5 slots, 5 rooms (one with zero capacity), 4 invigilators.
- **Result:** Unsatisfiable**.**
- **Conflicting Constraints Detected:**
    - Room capacity constraints due to zero-capacity room and insufficient capacities in other rooms.
    - Each exam must be timetabled in exactly one room and slot.
- **Suggested Resolution Attempt:**
    - Increased room capacities to accommodate the exams.
- **Suggestion's Result:** Conflicts resolved.
- **Output:**

```
========================================
Processing unsat4.txt:
========================================
Time Taken to solve this instance: 41 ms
unsat
Conflicting constraints detected:
  - Exam times or rooms are out of range.
  - Each invigilator can be assigned to only one exam.
  - Room capacity exceeded.

Suggested adjustments to resolve conflicts:
  - Increased capacity of room 0 by 10.
  - Increased capacity of room 1 by 10.
  - Increased capacity of room 2 by 10.
  - Increased capacity of room 3 by 10.
  - Increased capacity of room 4 by 10.
  - Increased number of time slots by 2.
  - Increased number of invigilators to 4.
Suggested changes resolved conflict successfully.

Solution 1:
   Exam: 0  Room: 2  Slot: 6  Invigilator: 1
   Exam: 1  Room: 0  Slot: 4  Invigilator: 0
   Exam: 2  Room: 2  Slot: 2  Invigilator: 2
   Exam: 3  Room: 1  Slot: 0  Invigilator: 3
```

---

**5. unsat5.txt**

- **Parameters:** 10 students, 4 exams, 4 slots, 5 rooms with various capacities, 4 invigilators.
- **Result:** Unsatisfiable.
- **Conflicting Constraints Detected:**
  - No student has overlapping exams.
  - Each exam must be timetabled in exactly one room and slot.
- **Suggested Resolution Attempt:**
  - Increased number of time slots to 5.
- **Suggestion's Result:** Conflicts resolved.
- **Output:**

```
========================================
Processing unsat5.txt:
========================================
Time Taken to solve this instance: 24 ms
unsat
Conflicting constraints detected:
  - Exam times or rooms are out of range.
  - Each invigilator can be assigned to only one exam.
  - Students cannot have overlapping exams.

Suggested adjustments to resolve conflicts:
  - Increased number of time slots by 1.
  - Increased number of invigilators to 4.
Suggested changes resolved conflict successfully.

Solution 1:
   Exam: 0  Room: 2  Slot: 0  Invigilator: 3
   Exam: 1  Room: 1  Slot: 2  Invigilator: 0
   Exam: 2  Room: 4  Slot: 4  Invigilator: 1
   Exam: 3  Room: 0  Slot: 0  Invigilator: 2
```

---

**6. unsat6.txt**

- **Parameters:** 8 students, 3 exams, 3 slots, 3 rooms with capacities favouring two larger rooms, 4 invigilators.
- **Result:** Unsatisfiable.
- **Conflicting Constraints Detected:**
    - No student has overlapping exams.
    - Each exam must be timetabled in exactly one room and slot.
- **Suggested Resolution Attempt:**
    - Increased number of time slots to 4.
- **Suggestion's Result:** Conflicts resolved.
- **Output:**

```
========================================
Processing unsat6.txt:
========================================
Time Taken to solve this instance: 20 ms
unsat
Conflicting constraints detected:
  - Students cannot have overlapping exams.
  - Exam times or rooms are out of range.

Suggested adjustments to resolve conflicts:
  - Increased number of time slots by 2.
Suggested changes resolved conflict successfully.

Solution 1:
  Exam: 0  Room: 1  Slot: 4  Invigilator: 2
  Exam: 1  Room: 1  Slot: 2  Invigilator: 1
  Exam: 2  Room: 0  Slot: 0  Invigilator: 0
```

---

**7. unsat7.txt**

- **Parameters:** 15 students, 5 exams, 6 slots, 3 rooms with varied capacities, 4 invigilators.
- **Result:** Unsatisfiable**.**
- **Conflicting Constraints Detected:**
  - Room capacity constraints for Exam 2 in all rooms.
  - Each exam must be timetabled in exactly one room and slot.
- **Suggested Resolution Attempt:**
  - Increased room capacities to accommodate the largest exam.
- **Suggestion's Result:** Conflicts resolved.
- **Output:**

```
========================================
Processing unsat7.txt:
========================================
Time Taken to solve this instance: 37 ms
unsat
Conflicting constraints detected:
  - Exam times or rooms are out of range.
  - Each invigilator can be assigned to only one exam.
  - Room capacity exceeded.

Suggested adjustments to resolve conflicts:
  - Increased capacity of room 0 by 5.
  - Increased capacity of room 1 by 5.
  - Increased capacity of room 2 by 5.
  - Increased number of time slots by 1.
  - Increased number of invigilators to 5.
Suggested changes resolved conflict successfully.

Solution 1:
  Exam: 0  Room: 2  Slot: 6  Invigilator: 3
  Exam: 1  Room: 1  Slot: 4  Invigilator: 4
  Exam: 2  Room: 1  Slot: 2  Invigilator: 1
  Exam: 3  Room: 2  Slot: 0  Invigilator: 2
  Exam: 4  Room: 0  Slot: 1  Invigilator: 0
```

---

**8. unsat8.txt**

- **Parameters:** 16 students, 4 exams, 3 slots, 3 rooms (one with zero capacity), 4 invigilators.
- **Result:** Unsatisfiable**.**
- **Conflicting Constraints Detected:**
  - No student has overlapping exams.
  - Each exam must be timetabled in exactly one room and slot.
- **Suggested Resolution Attempt:**
  - Increased number of time slots to 4.
- **Suggestion's Result:** Conflicts resolved.
- **Output:**

```
======================================
Processing unsat8.txt:
======================================
Time Taken to solve this instance: 69 ms
unsat
Conflicting constraints detected:
  - Exam times or rooms are out of range.
  - Each invigilator can be assigned to only one exam.
  - Students cannot have overlapping exams.

Suggested adjustments to resolve conflicts:
  - Increased number of time slots by 4.
  - Increased number of invigilators to 4.
Suggested changes resolved conflict successfully.

Solution 1:
   Exam: 0  Room: 1  Slot: 6  Invigilator: 3
   Exam: 1  Room: 1  Slot: 4  Invigilator: 0
   Exam: 2  Room: 1  Slot: 2  Invigilator: 2
   Exam: 3  Room: 0  Slot: 0  Invigilator: 1
```

---

**9. unsat9.txt**

- **Parameters**: 22 students, 9 exams, 8 slots, 5 rooms with various capacities, 4 invigilators.
- **Result:** Unsatisfiable**.**
- **Conflicting Constraints Detected:**
    - Students cannot have overlapping exams.
    - Exam times or rooms are out of range.
    - Each invigilator can be assigned to only one exam.
- **Suggested Resolution Attempt:**
    - Increased number of invigilators to 9.
    - Increased number of time slots by 9.
- **Suggestion's Result:** Conflicts resolved.

- **Output:**

```
========================================
Processing unsat9.txt:
========================================
Time Taken to solve this instance: 326745 ms
unsat
Conflicting constraints detected:
  - Students cannot have overlapping exams.
  - Each invigilator can be assigned to only one exam.
  - Exam times or rooms are out of range.

Suggested adjustments to resolve conflicts:
  - Increased number of time slots by 9.
  - Increased number of invigilators to 9.
Suggested changes resolved conflict successfully.

Solution 1:
   Exam: 0  Room: 2  Slot: 16  Invigilator: 2
   Exam: 1  Room: 2  Slot: 0  Invigilator: 8
   Exam: 2  Room: 1  Slot: 14  Invigilator: 5
   Exam: 3  Room: 1  Slot: 12  Invigilator: 0
   Exam: 4  Room: 1  Slot: 10  Invigilator: 4
   Exam: 5  Room: 1  Slot: 2  Invigilator: 1
   Exam: 6  Room: 1  Slot: 4  Invigilator: 7
   Exam: 7  Room: 1  Slot: 6  Invigilator: 6
   Exam: 8  Room: 2  Slot: 8  Invigilator: 3
```

---

**10. unsat10.txt**

- **Parameters**: 21 students, 8 exams, 10 slots, 3 rooms with varying capacities, 8 invigilators.
- **Result:** Unsatisfiable**.**
- **Conflicting Constraints Detected:**
  - No student has overlapping exams.
  - Each exam must be timetabled in exactly one room and slot.
- **Suggested Resolution Attempt:**
  - **Action:** Increased number of time slots to 11.

- **Suggestion's Result:** Conflicts resolved.
- **Output:**

```
========================================
Processing unsat10.txt:
========================================
Time Taken to solve this instance: 3970 ms
unsat
Conflicting constraints detected:
  - Exam times or rooms are out of range.
  - Each invigilator can be assigned to only one exam.
  - Students cannot have overlapping exams.

Suggested adjustments to resolve conflicts:
  - Increased number of time slots by 3.
  - Increased number of invigilators to 8.
Suggested changes resolved conflict successfully.

Solution 1:
   Exam: 0  Room: 0  Slot: 0  Invigilator: 6
   Exam: 1  Room: 2  Slot: 10  Invigilator: 1
   Exam: 2  Room: 2  Slot: 2  Invigilator: 7
   Exam: 3  Room: 1  Slot: 8  Invigilator: 0
   Exam: 4  Room: 1  Slot: 0  Invigilator: 4
   Exam: 5  Room: 1  Slot: 12  Invigilator: 2
   Exam: 6  Room: 2  Slot: 6  Invigilator: 5
   Exam: 7  Room: 1  Slot: 4  Invigilator: 3
```

**Performance Analysis**

- **Conflict Detection Effectiveness:** The automated conflict detection feature effectively identifies conflicting constraints and provides valuable suggestions to the user.
- **User Involvement:** Suggestions allow users to consider adjustments to resources or constraints to achieve a feasible solution.
- **Solver Limitations:** The solver does not automatically resolve conflicts or adjust test instances, maintaining the integrity of the original problem setup.
- **Enhanced User Feedback:** The Halo loading spinner improves user experience by providing real-time feedback during the solving process. This addition is particularly beneficial for larger test instances, where solving time may increase.

## F) Discussions

### Impact of Automated Conflict Detection and Resolution Suggester

The implementation of the Automated Conflict Detection and Resolution Suggester feature has significantly enhanced the solver's usability and robustness by:

- Identifying Conflicts:
    - The feature uses the unsatisfiable core to pinpoint conflicting constraints when the solver determines the problem is unsatisfiable. This targeted identification allows users to understand the root cause of conflicts efficiently.
- Providing Validated Suggestions:
    - Unlike static suggestion systems, this feature dynamically proposes adjustments (e.g., increasing invigilators, time slots, or room capacities) and validates their feasibility by rerunning the solver. The iterative validation ensures that only solutions leading to a satisfiable outcome are presented to the user.
- Dynamic Resolution Process:
    - The suggester automatically tests its proposed adjustments, incrementally modifying them if necessary, until a satisfiable solution is achieved. This ensures users are provided with practical, actionable resolutions without additional manual effort.
- Supporting Decision-Making:
    - By outputting validated suggestions and the resolved timetable, the feature helps users understand the limitations of their current resources and constraints while enabling them to implement conflict-free solutions confidently.

This feature not only improves the user experience but also enhances the solver's reliability by ensuring all suggestions are guaranteed to resolve conflicts before they are presented.

### Challenges Identified

- **No Automatic Fixes:** The solver does not automatically fix unsatisfiable instances, which may require user intervention and judgment.
- **Resource Constraints:** Suggestions to increase resources may not always be practical or feasible in real-world scenarios.
- **Complexity of Adjustments:** Determining the optimal adjustments may be complex and require additional analysis.

**Potential Improvements**

- **Enhanced Feedback:** Providing more detailed explanations or alternative suggestions to resolve conflicts.
- **Constraint Prioritization:** Allowing users to specify which constraints can be relaxed or are most critical.
- **Incremental Solving:** Implementing strategies to find partial solutions or the best possible schedule within existing constraints.
- Extend the loading spinner functionality to include additional statuses, such as preprocessing input data or generating suggestions for unsatisfiable cases.

**Insights from Test Instances**

- **Invigilator Constraints:** Ensuring an adequate number of invigilators is crucial for satisfying invigilator assignment constraints.
- **Room Capacities:** Room capacity constraints are a common source of unsatisfiability; adjusting capacities or reallocating rooms may be necessary.
- **Time Slots:** Increasing the number of time slots can alleviate scheduling conflicts, especially when students have overlapping exams.
- **Real-Time Feedback:** The loading spinner ensures that users are aware of ongoing computations, reducing confusion or the perception of application lag.

## G) Conclusion

The implementation of the automated conflict detection and resolution suggester feature has improved the solver's ability to handle unsatisfiable instances by providing users with insightful suggestions. This enhancement demonstrates the practicality of using SMT solvers like Z3 for complex scheduling problems and provides a foundation for further advancements.

**Successes**

- **Improved Usability**: Users receive clear feedback on why a problem is unsatisfiable and how it might be resolved.
- **Conflict Identification**: The use of the unsatisfiable core allows for targeted suggestions.
- **Support for Decision-Making**: Suggestions help users make informed decisions about resource allocation and constraint adjustments.

**Limitations**

- **No Automatic Resolution**: The solver does not automatically fix unsatisfiable instances, relying on user action.
- **Practical Constraints**: Suggested adjustments may not always be feasible due to real-world limitations.
- **Complexity**: Users may need to perform additional analysis to determine the best course of action.

**References**


- de Moura, L. M., & Bjørner, N. (2008). **Z3: An Efficient SMT Solver**. *Tools and Algorithms for the Construction and Analysis of Systems*, 337–340. https://doi.org/10.1007/978-3-540-78800-3_24
- Microsoft Research. (n.d.). **Z3 Theorem Prover**. Retrieved from https://github.com/Z3Prover/z3
- Python Software Foundation. (n.d.). **Python Language Reference, version 3.8**. Retrieved from https://docs.python.org/3.8/
- Nikolaj, B. (n.d.). **Programming Z3**. Retrieved from https://theory.stanford.edu/~nikolaj/programmingz3.html