Project Report: Automatic Timetable Generator (ATTG)

1. Executive Summary

The **Automatic Timetable Generator (ATTG)** is a sophisticated software solution designed to automate the complex task of scheduling academic classes. By leveraging **Constraint Satisfaction Problem (CSP)** algorithms, the system generates conflict-free timetables that optimize resource utilization (rooms, instructors, and time slots) while adhering to strict academic rules. The project features a modern, responsive web interface for easy data management and delivers comprehensive Excel-based schedules.

2. Project Objectives

The primary objectives of the ATTG system are: * **Automation**: Eliminate manual scheduling efforts and reduce human error. * **Optimization**: Efficiently allocate limited resources such as classrooms and laboratories. * **Constraint Compliance**: Ensure all schedules adhere to hard constraints (e.g., no double-booking) and soft preferences (e.g., instructor day off). * **User Experience**: Provide an intuitive, drag-and-drop interface for administrative staff. * **Accessibility**: Generate easy-to-read Excel reports for students, faculty, and administration.

3. System Architecture

The project follows a client-server architecture with a clear separation of concerns between the user interface, the API layer, and the core scheduling engine.

3.1 Technology Stack

- **Frontend**:
  - **HTML5 & Tailwind CSS**: For a responsive, modern, and "futuristic" UI design.
  - **TypeScript**: Ensures type safety and robust client-side logic (compiled to JavaScript).
  - **Features**: Drag-and-drop file uploads, real-time progress tracking, and animated feedback.
- **Backend**:
  - **Python (Flask)**: Serves as the web server and API handler.
  - **Pandas**: Used for efficient CSV data parsing and manipulation.
  - **XlsxWriter**: Generates formatted Excel files.
- **Core Logic**:
  - **Custom CSP Solver**: Implemented in Python (`csp.py`) using Forward Checking and Backtracking algorithms.

3.2 Data Flow

1. **Input**: User uploads 5 CSV files (Courses, Instructors, Rooms, Timeslots, Sections) via the web UI.
2. **Processing**: The Flask server receives files and passes them to the CSP engine.
3. **Solving**: The engine builds a constraint graph and searches for a valid assignment.
4. **Generation**: Upon success, the system generates multiple Excel files (Main, Year-wise, Instructor-wise, Room-wise).
5. **Output**: Files are bundled into a ZIP archive and sent back to the client for download.

4. Core Algorithm: Constraint Satisfaction Problem (CSP)

The heart of the system is the CSP engine defined in `csp.py`. It models the scheduling problem mathematically.

4.1 Variables and Domains

- **Variables**: Each "Course-Section-Session" tuple (e.g., `CSC111::Group1::Lecture`).
- **Domains**: The set of all valid combinations of `(TimeSlot, Room, Instructor)`.

4.2 Constraints

The system enforces several layers of constraints:

- **Hard Constraints** (Must be satisfied):
  - **Resource Uniqueness**: An instructor cannot teach two classes at once. A room cannot host two classes at once.
  - **Section Availability**: A student section cannot attend two classes at once.
  - **Instructor Qualification**: Instructors must be qualified for the course.
  - **Role-Based Assignment**:
    - *Professors* → Lectures only.
    - *Assistant Professors* → Labs and Tutorials only.
  - **Room Type Matching**: Lectures in Lecture Halls, Labs in Laboratories.
- **Soft Constraints** (Preferences):
  - **Instructor Preferences**: Respecting "Not on [Day]" requests.
  - **Load Balancing**: Distributing classes evenly across the week (handled via domain shuffling).

4.3 Search Strategy

- **Minimum Remaining Values (MRV)**: The solver prioritizes assigning variables with the fewest legal moves left to fail fast and prune the search tree.
- **Forward Checking**: Whenever a variable is assigned, the system updates the domains of unassigned neighbors, removing values that would cause a conflict.
- **Backtracking**: If a dead end is reached, the algorithm backtracks to the previous decision point.

5. Key Features

5.1 Intelligent Course Handling

- **Multi-Session Support**: Handles complex course structures (Lecture + Lab + Tutorial).
- **Duration Management**: Automatically distinguishes between 90-minute Lectures/Labs and 45-minute Tutorials based on course configuration.
- **Shared Courses**: Manages Year 3 courses shared across multiple departments (AID, BIF, CSC, CNC).

5.2 Modern User Interface

- **Visual Feedback**: Color-coded upload zones and real-time status indicators.
- **Progress Tracking**: A step-by-step progress bar visualizes the generation process (Parsing → Solving → Generating).
- **Error Handling**: Clear notifications for invalid files or generation failures.

5.3 Comprehensive Reporting

The system generates a `timetables.zip` containing: 1. **Main Timetable**: The master schedule. 2. **Yearly Schedules**: Separate files for Year 1, 2, 3, and 4. 3. **Instructor Schedules**: Individual files for every faculty member. 4. **Room Schedules**: Individual files for every room. *All Excel files feature professional formatting and color-coding (Yellow for Lectures, Blue for Labs, Purple for Tutorials).*

6. Project Structure

```
attg/
  server.py              # Flask application entry point
  csp.py                 # Core scheduling logic (CSP solver)
  templates/
```

```
    index.html          # Frontend HTML template
static/
    js/                 # TypeScript and compiled JavaScript
    css/                # Stylesheets
    uploads/            # Temporary storage for uploaded CSVs
requirements.txt        # Python dependencies
package.json            # Node.js dependencies (for TypeScript)
README.md               # Documentation
```

7. Conclusion

The Automatic Timetable Generator successfully modernizes the academic scheduling process. By combining a robust mathematical backend with a user-friendly frontend, it ensures that complex academic constraints are met efficiently, saving time and resources for educational institutions.