

Timetable Scheduling Problem: CSP Analysis

Introduction

This report provides an analysis of the Constraint Satisfaction Problem (CSP) approach used to solve the 1CS timetable scheduling problem. The problem involves scheduling various courses across multiple groups, ensuring that all constraints are satisfied to create a valid and efficient timetable.

Variables

The variables in this CSP represent the individual sessions that need to be scheduled. Each variable is uniquely identified by a combination of:

- Course name (e.g., "Sécurité", "Artificial Intelligence")
- Session type (lecture, td, tp)
- Group ("all" for lectures, or specific group identifiers like "G1", "G2", etc.)
- Session index (to distinguish between multiple sessions of the same type)

Variables are represented in the format: "CourseName_SessionType_Group_Index". For example:

- "Sécurité_lecture_all_0": Represents a lecture for the Sécurité course for all groups
- "Artificial Intelligence_td_G1_25": Represents a TD session for the AI course for Group 1

Domains

The domain of each variable consists of all possible day-slot-room combinations where the session can be scheduled. Each value in a domain is a tuple of (day, time slot, room), where:

- Day: One of "Sunday", "Monday", "Tuesday", "Wednesday", "Thursday"
- Time slot: One of "08:30-10:00", "10:10-11:40", "11:45-13:15", "13:20-14:50", "15:00-16:30"
- Room: A specific room depending on the session type (Lecture Hall, TD Room, or TP Lab)

Domain Restrictions:

- Tuesday domains only include morning slots (first 3 time slots)
- Lecture sessions can only be scheduled in Lecture Halls
- TD sessions can only be scheduled in TD Rooms
- TP sessions can only be scheduled in TP Labs

Constraints

The timetable scheduling problem includes both hard and soft constraints. Hard constraints must be satisfied for a valid solution, while soft constraints are preferred but can be violated if necessary.

Hard Constraints:

- No room conflicts: Two sessions cannot be scheduled in the same room at the same time
- No teacher conflicts: A teacher cannot teach multiple sessions at the same time
- No group conflicts: A group cannot attend multiple sessions at the same time
- Lecture and group-specific session conflicts: When a lecture is scheduled for all groups, no group-specific sessions can be scheduled at the same time

- Maximum consecutive sessions: A group cannot have more than 3 consecutive sessions in a day
- Room type matching: Lectures must be in Lecture Halls, TD in TD Rooms, and TP in TP Labs
- Tuesday afternoon unavailability: No sessions can be scheduled on Tuesday afternoons

Soft Constraints:

- Teacher workdays: Teachers should not have to come to school more than 2 days per week

Backtracking Approach

The solution uses a backtracking search algorithm with several enhancements to efficiently find a valid timetable. The key components of this approach are:

1. Preprocessing with AC3 (Arc Consistency)

Before starting the backtracking search, the AC3 algorithm is used to enforce arc consistency in the constraint graph. This preprocessing step helps to reduce the domain sizes by eliminating values that are guaranteed to violate constraints, making the subsequent backtracking search more efficient.

2. Variable Selection Heuristic (MRV)

The Minimum Remaining Values (MRV) heuristic is used to choose which variable to assign next during the backtracking search. MRV selects the variable with the fewest legal values in its domain, which helps to identify potential failures earlier in the search process.

Implementation:

```
unassigned = [v for v in self.variables if v not in assignment]\nreturn\nmin(unassigned, key=lambda var: len([val for val in self.domains[var] if\nself.is_consistent(var, val, assignment)]))
```

3. Value Ordering Heuristic (LCV)

The Least Constraining Value (LCV) heuristic is used to decide the order in which values are tried for a selected variable. LCV orders values by the number of conflicts they cause with other unassigned variables, trying values that rule out the fewest options for neighboring variables first.

Implementation:

```
def count_conflicts(value):\n # Count how many values would be eliminated from\n other variables' domains if we assign this value\n conflict_count = 0\n for\n other_var in [v for v in self.variables if v not in assignment and v != var]:\n for other_val in self.domains[other_var]:\n # Create a temporary assignment to\n check consistency\n temp_assignment = assignment.copy()\n temp_assignment[var] =\n value\n if not self.is_consistent(other_var, other_val, temp_assignment):\n conflict_count += 1\n return conflict_count\n\n# Return values sorted by the\n number of conflicts they cause\nreturn sorted(self.domains[var],\nkey=count_conflicts)
```

4. Constraint Checking

The `is_consistent` function checks if assigning a specific value to a variable violates any constraints with the current partial assignment. It checks for conflicts in room usage, teacher availability, group availability, and the maximum consecutive sessions constraint.

The consecutive sessions constraint is handled with special care to ensure no group has more than 3 consecutive sessions. The algorithm keeps track of the session slots for each group on each day and prevents assignments that would create too many consecutive sessions.

5. Backtracking Search Algorithm

The main backtracking algorithm works as follows:

- If all variables have been assigned, return the complete assignment as the solution
- Select an unassigned variable using the MRV heuristic
- Try assigning values from the variable's domain, ordered by the LCV heuristic
- For each value, check if the assignment is consistent with all constraints
- If consistent, add the assignment and recursively continue with the remaining variables
- If a recursive call returns a solution, return that solution
- If no value leads to a solution, remove the current variable's assignment (backtrack) and return failure

Solution and Output

When a valid solution is found, it is converted into a timetable format that shows the schedule for each group. The solution includes:

- A complete assignment of each course session to a specific day, time slot, and room
- Separate timetables for each group (G1 through G6)
- Analysis of teacher workdays to check the soft constraint of maximum 2 workdays per teacher

Conclusion

The timetable scheduling problem is solved using a constraint satisfaction approach with backtracking search. The implementation uses several heuristics (MRV and LCV) and preprocessing (AC3) to improve efficiency. The complexity of this problem arises from the many constraints that must be satisfied simultaneously, including room, teacher, and group availability, as well as the maximum consecutive sessions constraint.

The backtracking search systematically explores the space of possible assignments until it finds a valid solution or determines that no solution exists. When a solution is found, it is presented as a timetable for each group, showing the course, session type, teacher, and room for each time slot.