Artificial Intelligence - Timetable

Theodor-Ioan Rolea, 333CA

Faculty of Automatic Control and Computer Science, University Politehnica Bucharest

Abstract. This paper focuses on providing an efficient means of organizing a timetable using two popular Artificial Intelligence algorithms, namely A* and Hill-Climbing. Through a robust and fail-fast heuristic, the algorithms can produce viable results in a reasonable time horizon.

Keywords: AI, Artificial Intelligence, Timetable, A*, Hill-Climbing

1 Understanding the Problem

Implement A* and Hill-Climbing algorithms to automatically generate a class schedule with no or minimal conflicts, based on typical university structures.

1.1 Elements of the Problem

- **Time slots:** Each slot is 120 minutes (e.g., 8-10), with 6 slots in a workday (from 8 AM to 8 PM).
- **Weekdays:** Classes are held from Monday to Friday.
- **Rooms:** Limited number of rooms with specific capacities and subjects.
- **Subjects:** Each subject has a defined number of students requiring room assignment.

1.2 Room Assignment Example

- Subject X with 90 students and 2 available rooms:
  - Room A with 50 seats
  - Room B with 20 seats
- Valid assignments for Subject X:
  - Two time slots in Room A ($2 \times 50 = 100$ ¿= 90)
  - One slot in Room A and two in Room B ($1 \times 50 + 2 \times 20 = 90$ ¿= 90)

1.3 Hard Constraints

- Only one subject can be taught by one professor in a specific room during a time slot.
- A professor can only teach one subject in one room per slot.
- A professor can have a maximum of 7 teaching slots per week.
- A room can accommodate a number of students less than or equal to its maximum capacity.
- All students in a subject must be assigned to classes for that subject.
- All professors can only teach the subjects they specialize in.
- Rooms can only be used for the subjects they are designated for.

1.4 Soft Constraints

- **Professor Preferences:**
  - Preference for specific days or avoidance of certain days.
  - Example:
    - Monday → prefers to teach on Monday.
    - Not Tuesday → prefers not to teach on Tuesday.
  - Preference for or against specific time slots.
  - Example:
    - 8-12 → prefers any slot from 8-10 or 10-12.
    - Not 14-20 → avoids slots from 14-16, 16-18, or 18-20.

2 State Representation

- **timetable:** The current timetable, showing the allocation of subjects, professors, and rooms for each time slot.
- **timetable_specs:** Initial inputs for the task.
- **subjects_remaining:** A list of subjects yet to be scheduled in the timetable.
- **prof_hrs:** A dictionary showing the number of hours each professor is scheduled to work.
- **profs_in_interval:** Information on which professors are scheduled to teach in each time interval.
- **cost:** The cost of the current state.
- **prof_cost:** The cost associated with professor flexibility. If a professor can teach three subjects and is assigned over a professor that can teach only one of the subjects, the cost will rise due to inflexibility in the timetable.
- **classroom_cost:** The cost related to classroom flexibility, same as the prof_cost.

3 A*

3.1 Heuristic

The A* algorithm relies on a heuristic function to estimate the cost from the current state to the goal state. The following describes the heuristic used in our implementation:

- **Base Heuristic:** The initial heuristic value is derived from the total count of subjects remaining to be scheduled. This base value provides a simple estimate of how much work remains to complete the timetable.
- **Weighted Costs for Professor and Classroom Constraints:**
  - The heuristic includes additional weights for the costs associated with professor and classroom constraints. A high ‘prof_cost‘ or ‘classroom_cost‘ contributes more to the heuristic value.
  - These weights help the algorithm consider the complexity of managing professor schedules and classroom assignments. In our implementation, each of these costs is weighted by a factor of 4, emphasizing their importance in the scheduling process.

– **Constraint Violations:**
  - The heuristic heavily penalizes constraint violations, applying a higher weight to this factor. This reflects the critical impact that constraint violations have on the feasibility of a timetable.
  - In our heuristic, constraint violations are weighted by a factor of 10, indicating that a valid schedule with minimal violations is the top priority.

3.2 Differences from the laboratory function

The A* algorithm is nested inside its own class, where the heuristic is also calculated, and the main difference from the laboratory function is the addition of a closed_set. This improved the algorithm speed and efficiency by not returning to states that have already been visited.

By integrating these factors, the heuristic guides the A* algorithm toward states that have fewer remaining subjects, lower professor and classroom costs, and minimal constraint violations. This results in an efficient search for an optimal timetable that meets both hard and soft constraints.

4 Random Restart Hill-Climbing

4.1 Hill-Climbing Algorithm

The Hill-Climbing algorithm is a local search technique that starts from a given state and iterates through neighboring states to find a better solution. It aims to move to a neighboring state with a lower cost, indicating progress toward an optimal solution. The following describes the key components of our Hill-Climbing implementation:

– **Initial State and Neighbor Generation:**
  - The Hill-Climbing algorithm begins with an initial state that represents an incomplete timetable. This state contains information about subjects remaining, professor hours, classroom assignments, and associated costs.
  - Neighboring states are generated by assigning subjects to available slots in the timetable, respecting constraints such as room capacity, professor availability, and class schedules.
  - The algorithm iterates through these neighbors to find the one with the lowest cost, considering factors like professor and classroom costs, as well as constraint violations.
– **Evaluation and Cost Function:**
  - The cost function used for Hill-Climbing evaluates the feasibility of a timetable by considering the number of constraint violations and the overall cost of assignments.
  - The cost reflects the number of subjects remaining to be scheduled, additional costs for managing professor and classroom assignments, and penalties for constraint violations.

– **Iteration and Local Optima:**
  - Hill-Climbing moves to the best neighboring state with the lowest cost. If no improvement is found among the neighbors, the algorithm stops, indicating a local optimum.
  - To prevent getting stuck in a local optimum, Hill-Climbing was combined with a random restart strategy.

4.2 Random Restart Strategy

The random restart approach enhances Hill-Climbing by allowing the algorithm to restart from a new random initial state when it reaches a local optimum. The following describes how random restarts are implemented in our algorithm:

– **Maximum Restarts and Iterations:**
  - The random restart strategy runs Hill-Climbing for a specified number of iterations or until a valid timetable is found.
  - If Hill-Climbing gets stuck in a local optimum, the algorithm restarts from a new random state, with a maximum limit on the number of restarts.
– **Generating New Initial States:**
  - Upon restarting, a new initial state is created based on the existing timetable structure. This new state is derived from a random assignment of subjects, professors, and classrooms, ensuring a fresh start for the algorithm.
  - This process helps explore different areas of the search space, reducing the risk of getting stuck in suboptimal configurations.
– **Tracking Results:**
  - The algorithm tracks the number of restarts, iterations, and states generated to monitor the performance and efficiency of the search.
  - If a valid timetable is found, the algorithm stops and returns the successful result.

4.3 Benefits and Differences from A*

The random restart Hill-Climbing algorithm provides a simple yet effective approach to solving the timetable scheduling problem. While A* explores a broader search space with an informed heuristic, Hill-Climbing focuses on local optimization, relying on random restarts to overcome local optima.

The main benefit of the random restart strategy is its ability to avoid stagnation in local optima, making it suitable for scenarios where the search space is complex, and conventional Hill-Climbing would struggle to find optimal solutions. The restart mechanism adds flexibility to the search process, allowing exploration of new areas in the search space.

By incorporating these features, the random restart Hill-Climbing algorithm provides a robust method for finding feasible timetables, especially in scenarios with multiple constraints and complex scheduling requirements.

5 Why sometimes constrains are breached

Especially for Random Restart Hill-Climbing, constraints are breached more often than A*. It has to do with the way the cost was constructed, sometimes not converging towards an answer and getting stuck in a loop. There had to be a compromise between speed and accuracy, so some professors are going to be stuck teaching in an interval they do not want to. :)

6 Algorithms comparison

Table 1. Comparison of A* and Random Restart Hill Climbing

Test Case	Algorithm	Execution Speed	States Created	Constraints Breached
dummy	A*	0.1s	392	0
	RRHC	0.2s	1142	0
orar_mic_exact	A*	10s	19673	0
	RRHC	1m	36232	0
orar_mediu_relaxat	A*	3m	107732	0
	RRHC	5m	310218	7
orar_mare_relaxat	A*	8m	314817	0
	RRHC	13m	805542	20
orar_constrans_incalcat	A*	2m	52081	4
	RRHC	3m 30s	235562	24

7 Data Visualization

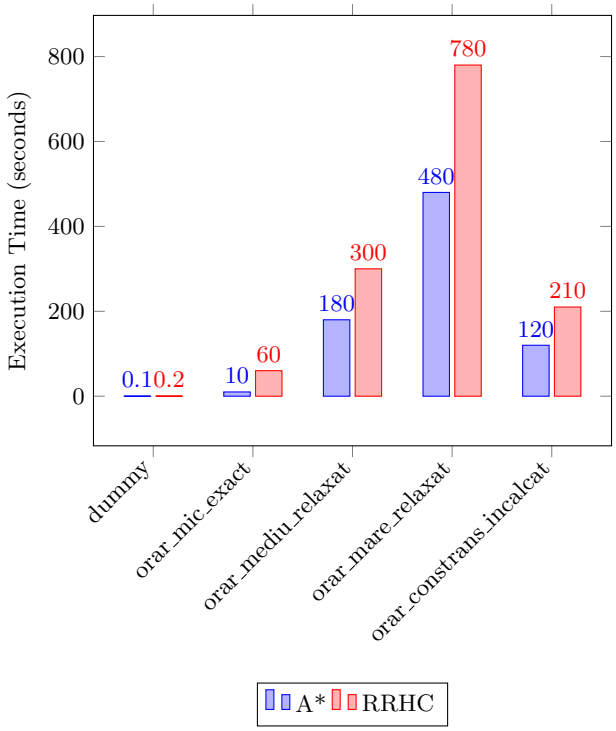

Fig. 1. Execution Time Comparison

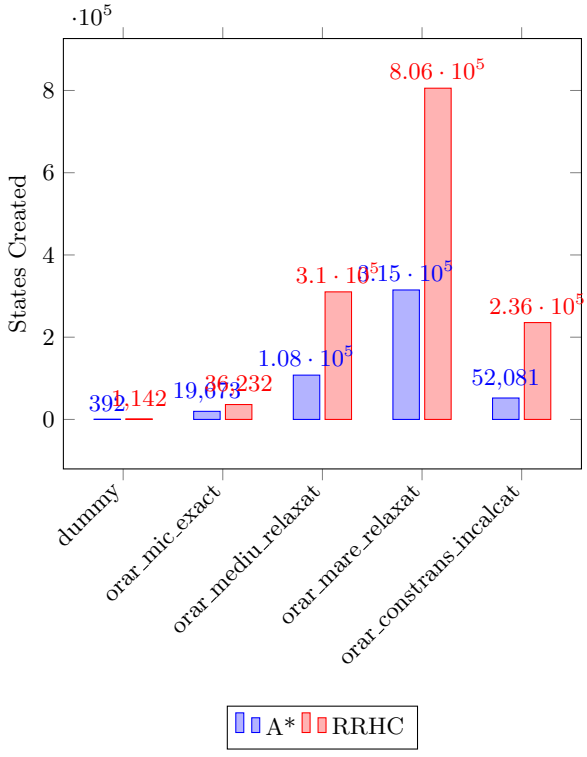

Fig. 2. States Created Comparison

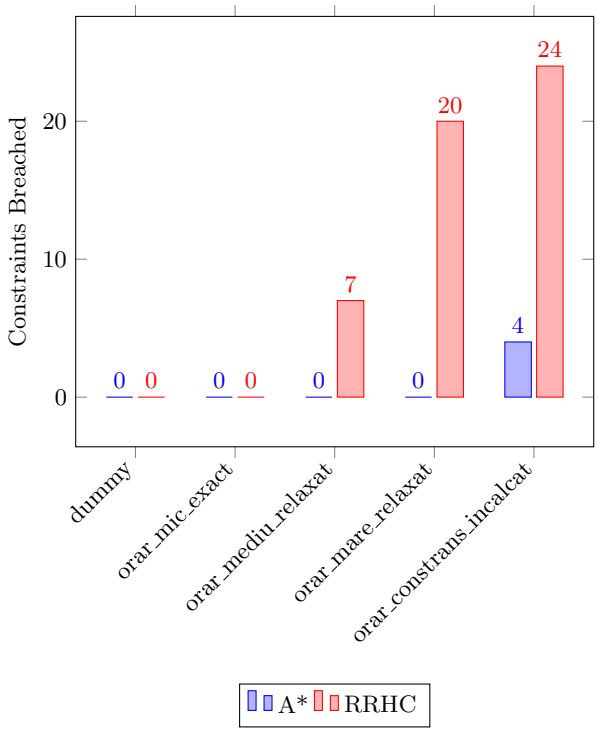

Fig. 3. Constraints Breached Comparison