International Journal of Computer Trends and Technology
ISSN: 2231–2803 / https://doi.org/10.14445/22312803/IJCTT-V72I10P107

Volume 72 Issue 10, 36-42, October 2024

Original Article

Optimization of Class Scheduling Problem: A Multi-Constraint Approach for Effective Resource Allocation and Space Utilization

Paulami Bandyopadhyay

Sr. Data Engineer, Milliman Inc, Chicago, IL, USA.

Corresponding Author : paulami88@gmail.com

Received: 22 August 2024 Revised: 26 September 2024 Accepted: 11 October 2024 Published: 23 October 2024

Abstract *- With the rapid growth of student enrollment and the expansion of academic offerings in universities and colleges worldwide, the task of scheduling classes within existing timetables and facilities has become increasingly complex. Today, class scheduling requires consideration of multiple factors, including room availability, capacity, instructors' preferences, and more. This problem is considered to be NP-complete and has received some research during the past few years. Several formulations and algorithms have been proposed to solve scheduling problems, most of which are based on local search techniques. In this paper, 2 different types of algorithms have been compared to solve the class scheduling problem: the random restart Hill-Climbing algorithm and the A-Star algorithm.*

Keywords *- A Star, Class Scheduling, Hill-Climbing, NP-complete, Searching Algorithms.*

1. Introduction

One of the biggest obstacles in the field of educational and operational research is the class scheduling problem. Universities have to deal with an ever-expanding course catalogue, which makes it necessary to assign a wide variety of classes to classrooms that effectively have different capacities. The goal of this optimization problem is to create a course schedule that maximizes the effective and efficient use of currently available facilities while also adhering to a comprehensive set of university constraints. A number of interrelated factors contribute to the Class Scheduling Problem's complexity, making it a complex challenge. First, there is significant variety in the number of students enrolled in each course. Comparably, there is a great deal of variance in classroom capacities. The existence of course-specific classroom constraints further compounds these factors. Furthermore, professors frequently share their preferences for the day of the week, time slot, and even break schedule for the courses they are assigned to teach. These elements, along with the requirement to adhere to numerous regulations specific to the university, make the process of assigning courses to classrooms an extremely complex undertaking. It is not enough to just make sure a classroom can hold all of the students enrolled in the course. A method like this would result in less-than-ideal space use, which could impair learning and increase student discontent. Imagine the following scenario: there are two courses, each with six and nineteen students. Moreover, suppose there are two classrooms: one with twenty seats and another with fifty. Although it is technically possible to arrange either course in either space, it would be more strategic to place the larger course in the larger classroom. This example best illustrates the complexity of the class scheduling problem. In order to guarantee efficient operation and the best use of resources within universities, researchers are still investigating advanced approaches to deal with this complex issue.

2. Problem Statement

Receiving a set of courses, each with a specific number of students, a set of rooms with a specific capacity, and a list of instructors with their list of preferences, the goal is to assign each course to a room and a time slot, according to a list of constraints. There are 2 types of constraints that need to be satisfied:

2.1. Hard Constraints

Below is the list of constraints that must be satisfied for the solution to be valid. If any of these constraints are not satisfied, the solution is invalid.

- Within a designated timeslot and in a given room, only one subject may be taught by a single instructor.
- During any given time slot, an instructor may teach only one subject, and this instruction must occur within a singular room.
- An instructor may conduct classes in a maximum of 7 time slots per week.

- Within a specified time slot, a room may accommodate a number of students equal to or less than its predetermined maximum capacity.
- All students enrolled in a particular subject must have designated class hours allocated for that subject.
- Instructors are restricted to teaching only the subjects in which they are specialized.
- All rooms are designated for classes pertaining only to the subjects for which they have been assigned.

2.2. Soft Constraints

Below is the list of constraints that are not mandatory, but they are taken into account when evaluating the quality of the solution.

- An instructor may express preferences regarding specific days of the week for teaching or may wish to avoid teaching on certain days.
- An instructor may have preferences regarding specific time slots during the day for teaching or may wish to avoid teaching during certain time slots.
- An instructor may prefer not to have a break exceeding a certain number of hours between consecutive classes.

3. Methods of Approach

3.1. Random Restart Hill Climbing

This is a local search algorithm commonly applied in optimization problems. It begins with an initial solution and iteratively explores neighboring solutions, selecting the one that improves the objective function the most. This process continues until a local optimum is reached, where no better solution can be found in the immediate neighborhood.

When the algorithm reaches a local optimum, it restarts the search from a random initial solution. The algorithm terminates when a specified number of iterations have been completed or when a solution that satisfies all constraints is found.

3.2. A Star

This is an informed search algorithm widely used for pathfinding and graph traversal tasks. It intelligently combines both actual path costs and heuristic estimates to guide the search towards the goal efficiently. A* maintains a priority queue of nodes to be explored and selects the most promising node based on a combination of the cost incurred so far and the estimated cost to reach the goal. This allows A* to efficiently find the optimal path while intelligently pruning the search space, making it highly effective for solving a wide range of optimization problems.

4. Algorithm Design

4.1. State Representation

The state representation for the Class Scheduling Problem consists of a schedule that assigns each course to a room and a time slot. The state is represented as a class that contains the following fields:

- *file_name:* The name of the file from which the data was read.
- *yaml_dict:* A dictionary containing the data read from the file.
- *size:* A tuplet (days, time_slots) representing the size of the schedule.
- *schedule:* A dictionary that has the following structure: {(day: str) : {(time_slot: (int, int)) : {(classroom : str) : ((instructor : str), (subject : str))}}}.
- *students_per_subject:* A dictionary that contains the number of students for each subject that needs to be scheduled.
- *count_teacher_slots:* A dictionary that contains the number of scheduled slots for each instructor.
- *trade_off:* A number that represents the trade-off between the number of constraints satisfied and the chosen classroom at each step (used in A Star).

4.1.1. Initial State

The initial state is generated as an empty schedule, which is initialized with the parameters derived from the input file, including the number of days, time slots, and classrooms. Additionally, the number of students per subject is obtained from the input file, while the count of scheduled slots for each instructor is set to zero. This initial state serves as the starting point for the search algorithms, enabling them to iteratively allocate courses to rooms and time slots until a valid schedule is achieved. An alternative method for generating the initial state involves randomly assigning course instructors to rooms and time slots in a manner that adheres to the hard constraints. A comparative analysis of these two approaches will be conducted in the Initial State Selection section.

4.1.2. Generating Neighbors

The neighbors of a state are generated by considering all possible combinations of assigning a course to a room, a time slot and an instructor while ensuring adherence to the hard constraints. In the initial phase, all potential neighbors that adhere to both the hard and soft constraints are generated. Subsequently, in the event that no neighbors satisfying all soft constraints are found, a secondary phase ensues where only neighbors satisfying the hard constraints are generated. This strategy reduces the total number of neighbors generated, allowing the algorithm to prioritize those that satisfy all constraints. Consequently, the algorithm minimizes time wastage by avoiding the generation of neighbors that would not be utilized, resulting in a reduced number of states generated.

4.1.3. Initial State Selection

As previously mentioned, the initial state can be generated in two methodologies: either as an empty schedule or through the random assignment of courses, instructors, and rooms to time slots. The former method exhibits a greater degree of determinism, initializing the schedule with vacant slots, whereas the latter introduces stochasticity into the initial

state generation process. Upon experimentation with both approaches, it became evident that the random initialization method could often yield solutions that fail to adhere to all specified soft constraints. This underscores the importance of carefully considering all potential subsequent states that may arise from the current state when designing the random initialization method. For instance, in instances where an instructor´s preferences are violated within a particular time slot, the algorithm should endeavor to substitute the instructor with another whose preferences remain unviolated for that time slot. Moreover, the algorithm should endeavor to substitute the time slot with another or substitute the room with another that has the same capacity in total.

This approach was found more difficult to implement. When trying to implement the first two substitutes, the algorithm was not able to find a solution that satisfies all the soft constraints, and it took quite a lot of time to find a partial solution due to a large number of constraints that should be checked while creating the neighbors. On the other hand, this approach will always find a partial solution that satisfies all the hard constraints. Thus, this one is recommended in cases when the soft constraints are not that important. The method that was used in the end was the empty schedule initialization. It is acknowledged that without the random initialization component, the Hill-Climbing algorithm may struggle to assign all students to a room, thereby violating a hard constraint. However, with the inclusion of the random restart, the algorithm will always find a solution that satisfies all the hard constraints, the soft ones being satisfied in most of the cases. This approach is recommended in cases when the soft constraints are more important than in the previous case, and it is easier to generate the neighbors.

4.2. Random Restart Hill Climbing

As previously mentioned, the Random Restart Hill Climbing algorithm is the most suitable for this approach. The algorithm is initialized with an empty schedule and generates neighbors that adhere to the hard constraints. The algorithm iteratively explores the neighborhood of the current state, randomly selecting one of the neighbors that satisfies the most constraints. This process continues until a local optimum is reached, at which point the algorithm restarts the search. The algorithm terminates when a solution that satisfies all constraints is found or when a specified number of iterations/restarts have been completed. A pseudocode of the algorithm is presented in Algorithm 1.

Algorithm 1: Random Restart Hill Climbing Algorithm

1: **procedure** *HILL_CLIMBING(max_restarts)* **return** *[is_final, total_iters, total_states, best_state]*
2: total_iters = 0
3: total_states = 0
4: best_state = ***None***
5: ***for*** *index in range(max_restarts)* ***do***
6: state = InitialState()
7: is_final, iters, states, state = STOCHASTIC_HILL_CLIMBING(state, total_iters)
8: total_iters+ = iters
9: total_states+ = states
10: **if** *is_final* **then**
11: **return** *[is_final, total_iters, total_states, state]*
*12***: if** *state does not have hard constraints* **then**
13: **if** *best_state == None or state has less soft constraints unsatisfied than best_state* **then**
14: best_state = state
15: **return** *[is_final, total_iters, total_states, best_state]*

Algorithm 1: Stochastic Hill Climbing Algorithm

1: **procedure** *STOCHASTIC_HILL_CLIMBING(state, max_iters)* **return** *[is_final, total_iters, total_states, best_state]*
2: total iters = 0
3: total states = 0
4: while total_iters < max_iters do
5: total iters+ = 1
6: **if** *state is final* **then return** *[****True****, total_iters, total_states, state]*
7: neighbors = state.generate_neighbors()
8: total_states+ = **len***(neighbors)*
9: **if** *neighbors ==* ***None*** **then return** *[****False****, total_iters, total_states, state]*
10: state = random.choice(from neighbors one of the neighbors with minimum number of constraints unsatisfied)
11: **return** *[****False****, total_iters, total_states, state]*

4.3. A Star

For the A Star algorithm, the state representation is the same as for the Random Restart Hill Climbing algorithm. The algorithm is initialized with an empty schedule and generates neighbors that adhere to the hard constraints. The algorithm iteratively explores the neighborhood of the current state. The frontier represents a heap that contains the states that need to be explored. The discovered is a dictionary that contains as keys the number of students that need to be scheduled for each subject and as values the cost of the state that brought about this configuration. The function used in the A Star algorithm is:

$$f(state) = g(state) + h(state) \quad (1)$$

where

$$h(state) = \text{total number of students that are not assigned} \quad (2)$$

$$g(state) = \text{number of constraints unsatisfied} * \text{weight} + \text{trade off} \quad (3)$$

$$\text{trade off} = \frac{\text{number of classrooms(subject)}}{\text{total number of classrooms}} \quad (4)$$

The heuristic function is admissible because the return value is always less than or equal to the actual cost of the state to reach a final one and is equal to 0 in the final states. On the other hand, the function h is not consistent, because a state from discovered can be added to the frontier with a smaller cost. The cost function is calculated as the number of constraints unsatisfied multiplied by a weight and the trade-off. In cases where the number of constraints unsatisfied is equal, multiplying it by a weight will prioritize the states that have the trade-off smaller. Moreover, when the number of constraints unsatisfied is different, the prioritization will be made based on the number of constraints unsatisfied, not on the trade-off. The trade-off is calculated as the number of classrooms that are assigned to a subject divided by the total number of classrooms. Adding a subject to a classroom that has fewer subjects assigned to it has been prioritized. For instance, if there was 2 subjects: A and B, 2 classrooms: 1 and 2, and classroom 1 is assigned to subject A and classroom 2 is assigned to both subjects while trying to assign a subject to classroom 2, subject B (trade off = 0.5) is chosen instead of subject A (trade off = 1). The pseudocode of the algorithm is presented in Algorithm 2.

Algorithm 2: A Star Algorithm

1: **procedure** *ASTAR* **return** *[is_final, total_iters, total_states, best_state]*
2: frontier = []
3: discovered = {}
4: state = InitialState()
5: frontier.append((f(state), state))
6: discovered[state] = 0
7: total_iters = 0
8: total_states = 1
9: while frontier do
10: current state = frontier.pop(1)
11: total_iters+ = 1
12: if current_state == 0 **then return** *[True, total_iters, total_states, current_state]*
13: neighbors = current_state.generate_neighbors()
14: total_states+ = len(neighbors)
15: **for** *neighbor in neighbors* **do**
16: new cost = g(neighbor) + h(neighbor)
17: students_per_subject = neighbor.students_per_subject
18: if students_per_subject not in discovered or new_cost < discovered[students_per_subject] **then**
19: discovered[neighbor] = new_cost
20: frontier.append((new_cost, neighbor))
21: **return** *[**False**, total_iters, total_states, current_state]*

4.4. Complexities

The complexity of the Random Restart Hill Climbing algorithm is $O(n)$, where n is the total number of iterations. The complexity of the A Star algorithm is $O(bd)$, where b is the branching factor and d is the depth of the solution. The complexity of the generate neighbors function is $O(d * t * c * i * s)$, where d is the number of days, t is the number of time slots, c is the number of classrooms, i is the number of instructors, and s is the number of subjects. Because the number of days does not exceed 7 (the worst case) and the number of time slots does not exceed 12, the complexity becomes $O(c * i * s)$.

5. Evaluation

Both algorithms are evaluated based on the quality of the solutions they provide, the time required to find these solutions and the total number of states explored during the search. The quality of the solutions is evaluated based on the number of constraints satisfied. The time required to find the solutions is measured in milliseconds, and the total number of states explored is counted during the search process.

5.1. Tests Description

Small timetable exactly: Contains 3 courses, 2 rooms, and 13 instructors. The number of students for each course are 300, 330, and 330. The capacity of the rooms is 20 and 30. The instructors have few preferences. Relaxed average schedule: Contains 4 courses, 4 rooms, and 18 instructors. The number of students for each course is 660, 660, 665 and 685. The capacity of the rooms is 25, 25, 35 and 70. The instructors have few preferences. Relaxed high schedule: Contains 8 courses, 6 rooms, and 37 instructors. The number of students for each course is 470, 475, 475, 495, 500, 530, 535 and 550. The capacity of the rooms is 25, 30, 30, 35, 85 and 85. The instructors have few preferences. Time limit violated: Contains 4 courses, 2 rooms, and 17 instructors. The number of students for each course is 720, 750, 780 and 810. The capacity of the rooms is 15 and 90. The instructors have a lot of preferences. Orar bonus exact: Contains 5 courses, 5 rooms, and 23 instructors. The number of students for each course is 500, 510, 515, 520 and 545. The capacity of the rooms is 15, 15, 15, 15 and 50. The instructors have a lot of preferences, including break constraints.

5.2. Results

For the Hill-Climbing algorithm, the number of restarts was established at 100, while the maximum number of iterations for all tests was capped at 1000. In the case of the A* algorithm, a weight of 100 was applied. Each test involved the execution of both algorithms. The ensuing tables depict the outcomes across various categories: the count of unsatisfied soft constraints, the time taken to attain a solution, the total number of states explored during the search, and the iterations needed to reach a solution. Additionally, for the Hill Climbing algorithm, the number of restarts executed until a solution was obtained is delineated. The outcomes for the Hill Climbing algorithm are categorized based on the number of restarts. Moreover, the individual results for each test are presented in the following Tables 1 and 2.

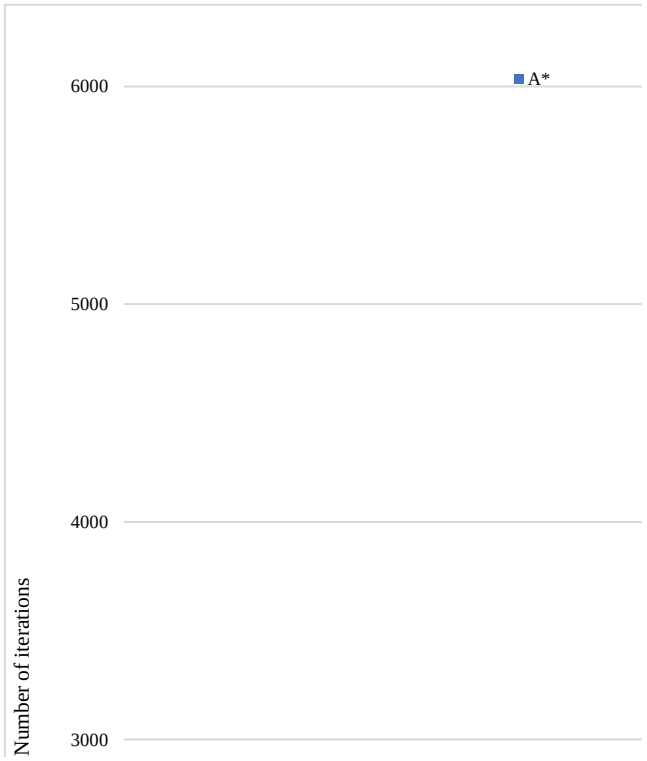

Fig. 1 Number of iterations required to find the solution

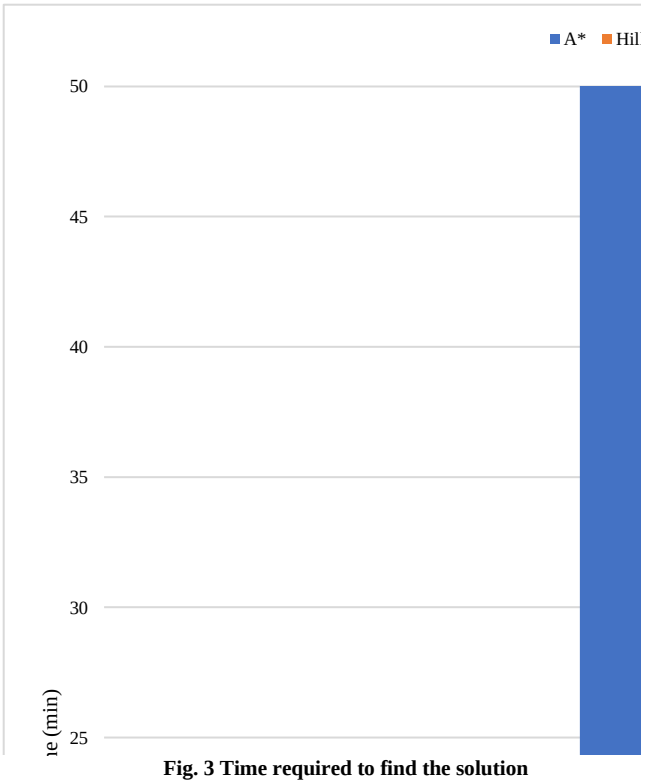

Fig. 3 Time required to find the solution

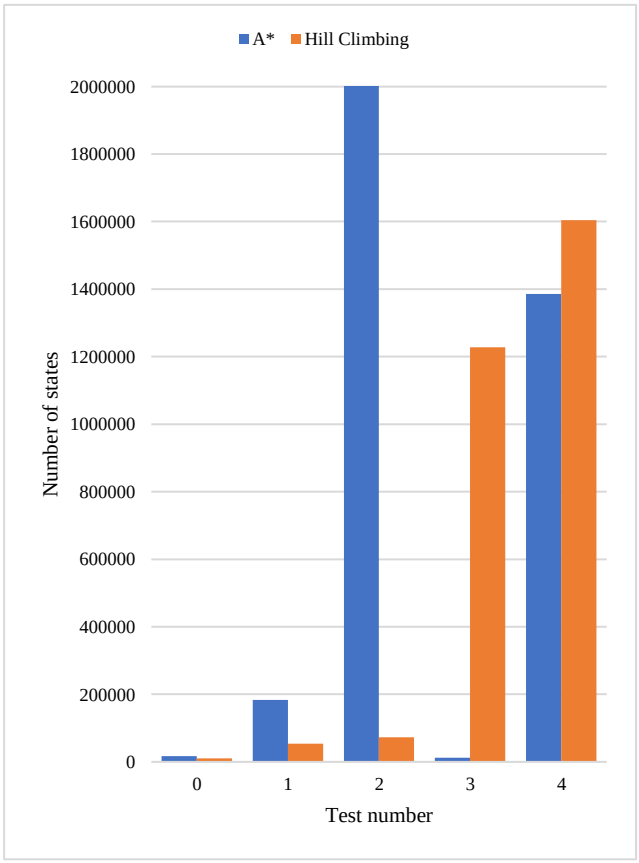

Fig. 2 Total number of states explored during the search

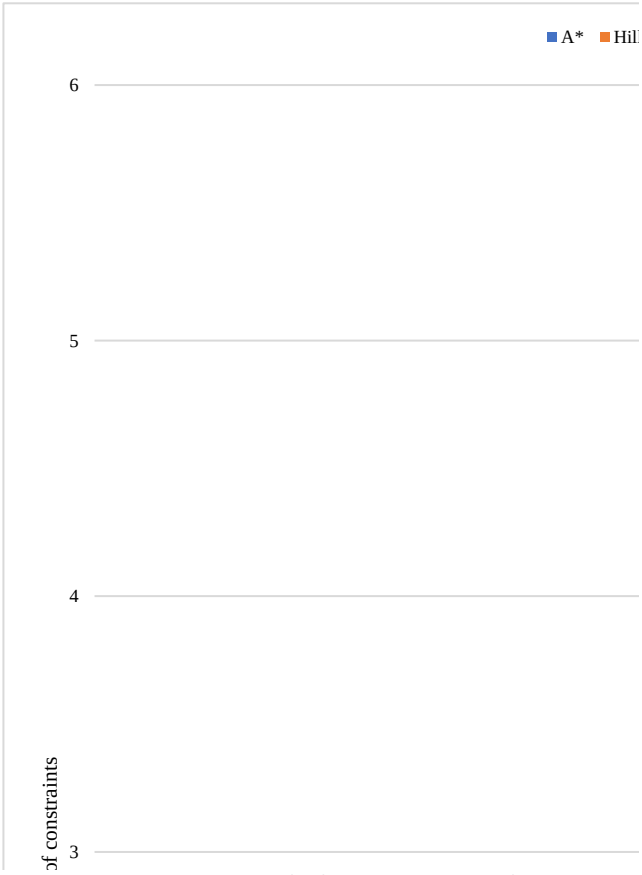

Fig. 4 Number of soft constraints unsatisfied

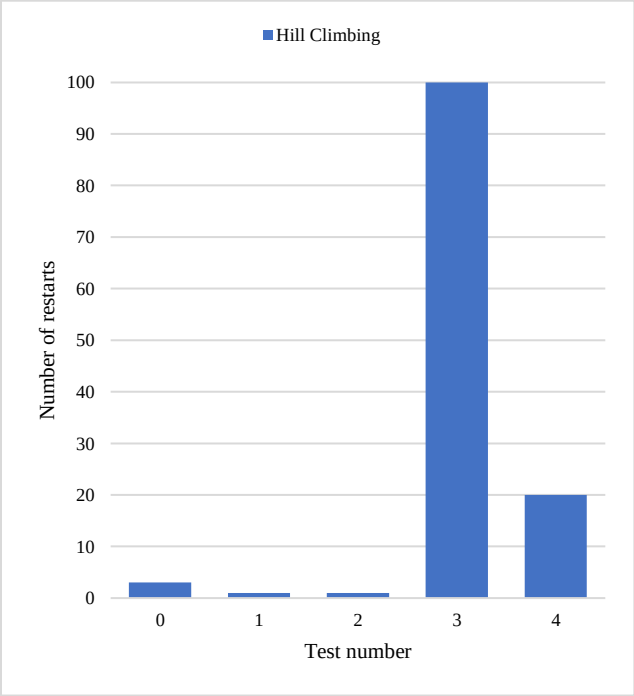

Fig. 5 Number of restarts until a solution was found for Hill Climbing

5.3. Observations

The Hill Climbing algorithm exhibits faster performance compared to the A* algorithm primarily due to its local search nature.

The Hill Climbing algorithm is more efficient in terms of the number of iterations required to reach a solution.

The A* algorithm, while slower, is more effective in terms of the number of unsatisfied constraints. The A* algorithm consistently yields solutions that satisfy all constraints, whereas the Hill Climbing algorithm occasionally encounters unsatisfied constraints.

Recognizing the prolonged duration required to attain a solution for orar_bonus_exact using the Hill-Climbing algorithm, the number of restarts was reduced to 20.

The A Star algorithm is more suitable for scenarios similar to orar_mic_exact, orar_mediu_relaxat, and orar_constrans_incalcat, where the number of neighbors generated is smaller. In contrast, the Hill Climbing algorithm is more appropriate for scenarios similar to orar_mare_relaxat and orar_bonus_exact, where the number of neighbors generated is larger.

Table 1. Results obtained for the Hill Climbing algorithm

No. set	No. iterations	No. states	Time mm.ss.ms	No. unsatisfied constraints	No. restarts
1	106	9989	0.4.317	0	3
2	71	53875	0.34.053	0	1
3	76	72621	0.77.753	0	1
4	5889	1228027	10.40.491	6	100
5	2221	1603927	22.47.289	5	20

Table 2. Results obtained for the A Star algorithm

No. set	No. iterations	No. states	Time mm.ss.ms	No. unsatisfied constraints
1	146	17073	0.8.537	0
2	158	183159	2.12.937	0
3	2883	2171276	52.49.856	0
4	91	12368	0.8.172	0
5	2542	1385784	27.15.878	0

6. Conclusion

In summary, based on the context of the implementation of the Class Scheduling Problem, the Hill Climbing algorithm outperforms the A* algorithm in terms of efficiency. A* achieves slower solution discovery but with no unsatisfied constraints. Conversely, Hill Climbing is faster but may occasionally encounter unsatisfied constraints. The Hill Climbing algorithm is more suitable for scenarios where the soft constraints are less significant and the time required to find a solution is a critical factor. In contrast, the A* algorithm is more appropriate for scenarios where the soft constraints are more significant and the quality of the solution is paramount.

References

[1] Kian L. Pokorny, and Ryan E. Vincent, “Multiple Constraint Satisfaction Problems Using the A-Star (A*) Search Algorithm: Classroom Scheduling with Preferences,” *Journal of Computing Sciences in Colleges*, vol. 28, no. 5, pp. 152-159, 2013. [Google Scholar] [Publisher Link]

[2] Der-Fang Shiau, "A Hybrid Particle Swarm Optimization for a University Course Scheduling Problem with Flexible Preferences," *Expert Systems with Applications*, vol. 38, no. 1, pp. 235-248, 2011. [CrossRef] [Google Scholar] [Publisher Link]

[3] Russell, *Artificial Intelligence: A Modern Approach*, 2nd ed., Pearson Education, 2003. [Google Scholar] [Publisher Link]

[4] Abdoul Rjoub, "Courses Timetabling Based on Hill Climbing Algorithm," *International Journal of Electrical and Computer Engineering*, vol. 10, no. 6, pp. 6558-6573, 2020. [CrossRef] [Google Scholar] [Publisher Link]