# Enforcing Business Rules in University Timetabling Through Database Constraints and Heuristics

Marcus V. Reisdoefer Pereira
mvreisdoefer@inf.ufpr.br
Federal University of Paraná
Curitiba, Paraná, Brazil

Muriki Gusmão Yamanaka
mgyamanaka@inf.ufpr.br
Federal University of Paraná
Curitiba, Paraná, Brazil

Marcos A. Castilho
marcos@inf.ufpr.br
Federal University of Paraná
Curitiba, Paraná, Brazil

Simone Dominico
simone@inf.ufpr.br
Federal University of Paraná
Curitiba, Paraná, Brazil

## Abstract

Educational institutions increasingly rely on software systems to manage timetables, which raises the classroom assignment problem, assigning each lecture to a suitable room with adequate capacity while preventing resource conflicts such as overlapping allocations. In this paper, we propose a generic relational database model that enforces core business rules of university timetabling at the database level, supporting both the initial allocation phase and subsequent timetable changes throughout the term. Our approach leverages PostgreSQL range types and exclusion constraints (GiST/tsrange) to prevent overlapping reservations and inconsistent assignments, and models real-world requirements such as joint classes, room ownership, and ad-hoc event reservations. We also discuss how heuristic weighting can be coupled with the data model to guide an initial allocation procedure. The proposed model is illustrated and validated in the context of the Federal University of Paraná (UFPR).

## Keywords

Classroom Assignment, Database Modeling, Heuristic Algorithms, Timetabling

## 1 Introduction

In the most general sense, an educational timetabling problem consists in assigning meetings between parties while respecting a set of mandatory and non-mandatory constraints, such as time ranges, room availability, and room capacity. This problem is generally modeled as an optimization problem and has been proven to be NP-complete [4].

Recent approaches to solving the school timetabling problem focus primarily on the optimization model and the algorithm used to solve it [15], often relying on heuristics, meta-heuristics, and evolutionary techniques. Due to the many different ways this problem appears, like in high schools or universities, and the cultural differences between countries, a wide range of solutions exist, which makes some fit only for specific use cases.

While much of the research focuses on algorithmic aspects, little attention has been given to the role of data modeling in enforcing institutional business rules within school timetabling systems. As a consequence, many systems rely heavily on application-level checks, which are harder to maintain and insufficient to guarantee long-term consistency under frequent timetable changes.

In practice, timetables are subject to frequent manual and incremental changes throughout the academic term, such as room swaps, special events, or ad-hoc reservations. When business rules are enforced only at the application or algorithmic level, these changes may easily lead to inconsistent or conflicting allocations.

In this paper, we propose a generic relational database model for university timetabling that supports the initial classroom allocation and enforces core business rules throughout the semester. The model encodes constraints such as overlapping prevention, exclusive room ownership, and joint classes directly at the database level.

We separate the classroom assignment problem into two complementary layers, being an allocation procedure responsible for generating an initial feasible assignment, and a database model responsible for enforcing integrity constraints and business rules during timetable evolution.

The remainder of this paper is organized as follows. In section 2 we elaborate on the background and terminology used in this paper. section 3 contains a few works relevant to this paper. Then, in section 4 we discuss the business rules required for the modeling of the problem. In section 5 we describe how the database is modeled to implement some of the business rules. Finally, in section 6 we show the algorithm used for the initial room allocation.

## 2 Background

The School Timetabling problem we model here represents a university where classes have a start and end date, which generally represents a semester. Instead of relying on a fixed notion of academic semesters, we adopt a time-range based representation, which allows the model to accommodate annual courses and special cases with arbitrary start and end dates.

In this model, the university runs the allocation algorithm once every semester or so, where most of the classroom allocations are made. Then, the database model is responsible for ensuring business rules are not violated by users of the system when performing changes to the allocations. This separation allows the system to maintain consistency even when timetable changes are performed manually or incrementally after the initial allocation.

Classes are only allowed inside classrooms at reasonable distances, with the proper resources, and any other preferences required by the user. Also, since classrooms have different capacities, locations, resources, and may be owned by different entities, that

will determine how they will be available to be used in the allocation algorithm. For example, in this paper's case study, rooms owned by the sector may be used by any department in that sector by default, but rooms owned by a department are normally available only to that department, unless intentionally shared. These characteristics motivate the need for explicit constraints in the data model, rather than treating room selection as a purely algorithmic decision.

## 2.1 Terminology

This section describes the terminology we use in this paper, which is crucial to define the database model itself, since much of the regular terminology is used interchangeably in the real world:

- **Class**: Refers to a group of students enrolled in a particular subject, e.g. Calculus. Note that it's possible that multiple Calculus *classes* exist.
- **Time range**: A start timestamp and end timestamp. All time ranges in this paper refer to a full date (year, month, day), hour and minute.
- **Lesson**: It is a *time range* where one or more classes of the same subject may meet in a room to have some sort of *lecture*.
- **Allocation**: A *time range* where a room is considered occupied. A *lesson* that has an assigned room is considered allocated.
- **Event**: An event is a special name we give to a ad-hoc room allocation, e.g. for a congress. This differs from most allocations because those are generally just "repeated" lessons, spread across an entire semester. While events typically happen for a one-time meeting, for example.

Note that even though all time ranges have a date, most allocations will only happen within the same day, since most classes only happen for a few hours a day at most.

It is important to emphasize the difference between classes and lessons. Lessons represent meetings in a classroom for classes, while classes represent a group of people. Multiple classes may frequent the same lesson, but then have separate lessons later. For instance, both classes of *Calculus* may have the same lesson and therefore use the same room at 13:30, but then separate and each have its own lesson and room at 15:30.

## 2.2 Key challenges

There are some important challenges when it comes to solving the classroom assignment problem, particularly concerning the wide variety of edge cases that arise from real-world business rules that need to be enforced. The main challenges taken in consideration in this study include:

- **Dynamic Room Demand**: The demand for rooms varies across departments and time slots. While some school systems may have predictable and stable room needs, others may experience fluctuating values. Balancing room availability with fluctuating demands requires a flexible allocation approach.
- **Overlapping Time Constraints**: Lessons often overlap with one another, specially in a university scenario, creating potential conflicts in room assignments. Managing these conflicts involves careful scheduling to avoid room double-booking while satisfying the time preferences of both instructors and students.
- **Classroom Suitability**: Certain subjects require specialized rooms with specific equipment, such as laboratories, seminar rooms, or auditoriums. Allocating these rooms is more difficult because not all classrooms can be used for every type of class. Ensuring that each class is assigned a suitable room based on its needs adds another layer of complexity.
- **Fairness and Equity**: Fairness in room allocation is crucial, especially for classes that occur frequently, such as core subjects. The allocation algorithm must strive to avoid bias in room assignments, ensuring that no particular department or subject is unfairly advantaged or disadvantaged in terms of room quality or schedule.
- **Scalability**: As the number of rooms, classes, and time slots increases, the computational complexity of finding optimal allocations rises. The system must be designed to handle larger universities with many departments and courses, ensuring scalability while maintaining efficiency and correctness. This has to be valid both for the algorithm and the database model meant to manage this system.

Addressing these challenges requires not only efficient and effective allocation algorithms, but also a data model capable of enforcing institutional constraints throughout the timetable.

# 3 Related Works

The integration of database systems with constraint satisfaction problems and algorithms has not been widely discussed. Ataullah et al. [3] highlight the importance of integrating business-level policies into database systems. While their work focuses on generic policy enforcement, our approach applies these principles specifically to university timetabling, mapping concrete institutional rules to relational constraints.

In the field of timetabling, Vrielink et al. [16] explore the development of timetabling applications, with a focus on bridging the gap between theoretical models and real-world educational requirements. Their work serves as an inspiration for our study, where we leverage similar timetabling principles to optimize our database model and heuristic approach.

Kingston [11] provides a comprehensive overview of timetabling problems, including high school, university examination, and course timetabling. The study examines various subproblems, such as student sectioning and room assignment, and emphasizes how these challenges can be modeled and solved in real-world scenarios. This research is particularly relevant to our work, as it references the diversity of educational environments and how business rules must be enforced for each different school structure.

The Hungarian algorithm, a foundational tool in combinatorial optimization, has been applied in numerous domains, including robotics and timetabling. Shopov [14] examines the use of the Hungarian algorithm in robotic formations, comparing two modifications of the algorithm and investigating its time complexity for various numbers of agents. This study resonates with our work, as we also utilize a modified version of the Hungarian algorithm to match resources efficiently within a database framework.

Building on the Hungarian algorithm, Abeywickrama [1] proposes an optimization technique that calculates costs incrementally rather than all at once, improving efficiency for real-world tasks. This incremental approach is relevant to our methodology, where we similarly adapt the algorithm to handle large-scale, dynamic problems by reducing computational overhead by solving a bigger problem instance incrementally.

Elloumi et al. [8] present a theorem and reduction procedure inspired by dominance criteria, aiming to reduce the problem size while maintaining feasibility. Their work demonstrates how to handle large-scale assignment problems more efficiently through the application of heuristic methods. Our approach shares similarities with theirs, as we employ a reduction technique to manage the complexity of our problem while ensuring that feasibility constraints are met.

Lastly, Prado et al. [13] discuss heuristic solutions applied to their university, a case study that parallels our own. By drawing on their experience, we adapt heuristic techniques to solve complex classroom assignment problems specific to our university's needs, further strengthening the link between theory and practice.

In summary, existing approaches predominantly address university timetabling as a standalone optimization problem. Our work shifts part of the responsibility for correctness to the database layer, ensuring that institutional business rules remain enforced even after the optimization process has completed and the timetable evolves over time.

## 4 Business Rules

Real-world timetabling systems require explicit representation and enforcement of domain-specific business rules. Managing integrity rules can be challenging as the number of restrictions and their complexity increase. As an example, the work from [3] proposes a general purpose policy modeling and constraint management framework that can integrate numerous aspects of business level requirements within database systems. Here, we focus on a set of representative rules for the School Timetabling Problem and discuss how they can be translated into database constraints and heuristic decisions.

### 4.1 Joint class

A key modeling requirement in classroom assignment systems is to clearly define what constitutes an allocatable entity. At first glance, it is normal to think that a class is simply assigned to a classroom with one professor exclusively. However, in universities, sometimes two or more professors may agree to combine their classes, either because both classes are small, and it might be more space-efficient to allocate a single room, or because they have decided to take turns teaching. Either way, from a resource utilization perspective, allocating a classroom that remains unused throughout the semester is undesirable. Hence, here we make the distinction that a class is always held by one professor and a lesson can be the join of one or more classes. That way, what actually allocates a room is a lesson.

The Figure 1 illustrates this example. There are two classes of the discipline *Algorithms*, one held by the professor Marcus and one by professor Luiz. Since both classes follows the same topics and there exists a classroom that can occupy all of the students, the professors decide that it is the better for the two classes to be combined in a lesson and allocated to the classroom *PA-01*.

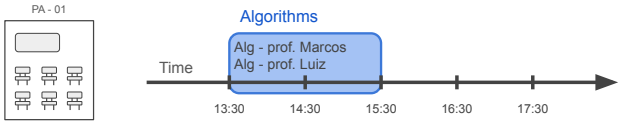


**Figure 1: Two classes combined in one lesson that is allocated to a classroom**

### 4.2 Lessons overlap

Although it's possible to have multiple classes being allocated to a single classroom, multiple lessons cannot overlap if they are using the same room. In the real world, this could lead to a resource conflict between the professors and students. Therefore, multiple lessons assigned to the same classroom must not overlap in time.

The Figure 2 represents this problem. The *Algorithms* lesson is allocated to use the classroom *PA-01* between 13:30 and 15:30, but at 14:30 the professor from the lesson *Calculus* appears with its students. From this point on, the professors could disagree, with the *Algorithms* professor saying that they were allocated first and the *Calculus* professor arguing that they were properly allocated in the system.

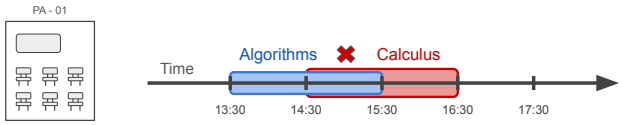


**Figure 2: The overlapping of two lessons in the same classroom.**

Therefore, it is imperative that the system makes it impossible for resource conflicts like this to arise. In this paper this is the role of the database model.

### 4.3 Multiple Lesson Allocation

Another common problem to solve is that one lesson cannot be allocated to multiple classrooms at the same time. Since professors and students cannot physically attend multiple locations simultaneously, one could argue that a professor could teach online, but at that point, the need to allocate a classroom no longer exists. For this rule it is established that, for all lessons at the university, it is not acceptable for a lesson to be allocated more than once with overlapping time range.

This situation is illustrated by Figure 3. The *Algorithms* lesson is being held in the *PA-01* classroom, but at some time the lesson is also allocated to the *PA-02* classroom. Since both cannot be used at the same time by the same people, one of them ends up not being used, resulting in a waste of resources.

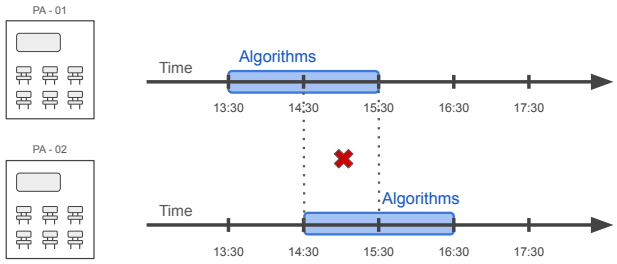


**Figure 3: Example of one lesson being assigned to more than one classroom during the same time interval**

## 4.4 Polymorphic ownership

Polymorphic ownership is a common problem for inheritance representation in relational databases. This occurs whenever the relationship between a table and a set of other tables depends on a condition. In the context of this work it is necessary to determine who is responsible for managing each classroom. Also, each classroom must be associated with exactly one administrative owner.

The Figure 4 show this situation. When someone needs to create a new allocation for a classroom, the system needs to know who is the owner of the classroom. This is because each ownership has its own set of rules to be followed when it comes to using their classrooms. For example, in the UFPR case study, it is forbidden for secretaries from departments to allocate anything in another department's rooms. However, sector classrooms may be used by any department in the sector.

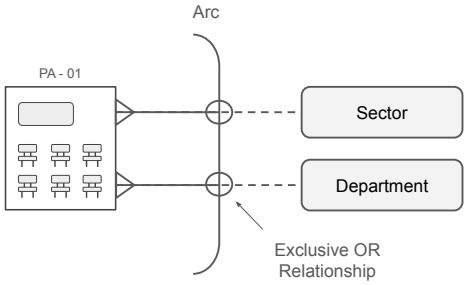


**Figure 4: A classroom can have either a sector or a department as its owner.**

This model could be expanded to better fit different university structures, instead of our model, which represents sectors that contain multiple departments.

## 4.5 Ad-hoc classroom reservations

Again, for universities, classrooms are frequently used to host academic events. In these cases, the system must support temporary and non-recurring classroom reservation. Our model covers that case by making each event the same as a reservation, so instead of assigning a lesson to the classroom at that specific time range, like a semester, we may choose to assign a single-time event to it. Thus, this rule states that the system must be able to reserve one or more classrooms for an event one or more times.

Figure 5 illustrates this rule. Every week, on Tuesdays and Thursdays, the *Algorithms* lesson is held in room *PA-01*, but on November 10, there will be a special lecture for an event and classes will be canceled or moved to another classroom.


**Figure 5: Example of a event in the middle of the allocations**

That also has a consequence that we can keep a history of room allocations, so changing the room for a certain class only changes the allocations for the current date and those after, which might be useful for certain types of preference optimizations for future executions of the allocation algorithm.

## 4.6 Arbitrary room allocation constraints

When dealing with simpler school timetabling, like an elementary school, it is common that a lesson can be held in any classroom in the building that can hold the number of students. But, in our context of universities, a classroom may be geographically far away from other classrooms or may not have the proper equipment to hold the teaching. This would create difficulties or even make it impossible for students and professors to have a lesson. As a result, a lesson can only be allocated to a predefined set of classrooms.

As it is shown in Figure 6, the *Algorithms* lesson can be allocated in the classrooms *PA-01* and *PA-02*, but cannot be in *PA-03*. The reason is that computer science students in a programming lesson might need computers, and allocating a lesson in an unsuitable classroom (like a chemistry lab) would likely make the class unfeasible. Other than that, it's a way to model hard constraints for classroom assignment, disallowing assignment of classrooms that are too far away, for example.

These constraints *must* be informed by the users. Because in order for the model to be consistent with real-world business rules, lessons must not be allocated in invalid rooms. So, by default, lessons cannot be allocated anywhere, unless specified otherwise.

# 5 Database Model

Based on the business rules defined in the previous section this section describes how these constraints can be represented using a relational database model.

## 5.1 Lessons with Joint Classes

Classes and Lessons represent a *many-to-many* relationship. A lesson may be attended by multiple classes, and each class may participate in multiple lessons, characterizing a many-to-many relationship. The representation of this relation is achieved in a relational model by creating two tables *lessons* and *classes*, and one junction table to ensure the many-to-many relationship is present.

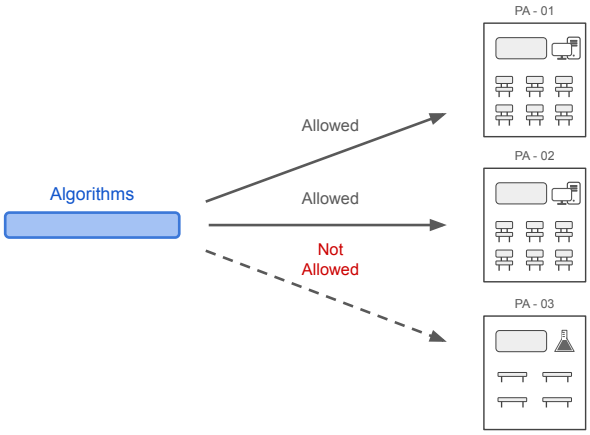


**Figure 6: Example of which classrooms are allowed to allocate a lesson.**

Once a lesson is allocated to a classroom, it is possible to query the classes by joining the tables.

## 5.2 Lessons Overlap Constraint

To represent the allocation of a lesson in a classroom the *Reservation* relation, shown in Table 1, is used. This table creates a relationship with the other tables *lessons* and *classrooms*. Each row has a *start_time* timestamp and a *end_time* timestamp to represent the time range of the allocation. To prevent two lessons from overlapping in the same classroom, we use exclusion constraints with GiST (Generalized Search Tree) index and tsrange (timestamp range) functions in *PostgreSQL*. PostgreSQL was chosen due to its native support for range types and exclusion constraints, which allow temporal conflicts to be enforced declaratively at the database level. In the constraint, we specify to block whenever two tuples with the same *Classroom* (like *DI-01*) have an overlap in time with *start_time* and *end_time* as shown in the *lesson_overlap* constraint in Listing 1.

```
-- Prevents Two Lessons Overlap
ALTER TABLE reservation ADD CONSTRAINT lessons_overlap
  EXCLUDE USING gist (
    Classroom WITH =,
    tsrange(start_time, end_time) WITH &&
  );

-- Prevents Lesson Multiple Allocation
ALTER TABLE reservation ADD CONSTRAINT multiple_lesson
  EXCLUDE USING gist (
    Lesson WITH =,
    tsrange(start_time, end_time) WITH &&
  );
```

**Listing 1: Lessons Overlap and Multiple Allocation Constraints Definition**

Although time intervals could be represented as a single range attribute, we explicitly store start and end timestamps to simplify constraint definition and comparison semantics.

## 5.3 Multiple Lesson Allocation Constraint

The same exclusion constraint mechanism can be applied to prevent a single lesson from being allocated to multiple classrooms with overlapping time ranges. In this case, instead of using the *Classroom* attribute in the exclusion constraint, the *Lesson* attribute must be used, as shown in Listing 1. This ensures that the database blocks any insertion or update in which the same *Lesson* (like *Algorithms*) intersects with *start_time* and *end_time*.

The Listing 2 exemplifies two attempts of the insertion of inconsistent data. In the first statement, a new lesson called *Databases* is allocated in the classroom *DI-01*, conflicting with the already allocated lesson *Algorithms*. In the second statement, this time the *Algorithms* lesson is allocated in another room *PA-02* overlapping in time with the already allocated lesson in *DI-01*. In both cases, the error return shows that the database successfully protected the data against inconsistencies.

```
-- Fail by two lessons overlapping the same room
Timetabling=# INSERT INTO reservation VALUES
  ('Databases', NULL, 'DI-01',
   '2026-04-16 14:30:00', '2026-04-16 16:30:00');
ERROR:  conflicting key value violates exclusion
     constraint "lessons_overlap"

-- Fail by lesson multiple allocation
Timetabling=# INSERT INTO reservation VALUES
  ('Algorithms', NULL, 'PA-02',
   '2026-04-16 14:30:00', '2026-04-16 16:30:00');
ERROR:  conflicting key value violates exclusion
     constraint "multiple_lesson"
```

**Listing 2: Lessons Constraint Protection**

## 5.4 Mutually Exclusive Relationships (ARC)

To represent polymorphic ownership, we use mutually exclusive relationships (ARC) in the relational model [9]. The relational model does not allow a direct representation of multiple references for a single foreign key. If one wants a relation to reference $N$ other relations, then $N$ attributes must be created. In this case, to ensure that the database only allows each tuple to reference at least one of the relations, it is necessary to construct mutual exclusion constraints also called ARC relations. A check constraint ensures that exactly one of the foreign keys (Sector or Department) is non-null for each classroom.

To solve the problem of polymorphic ownership in the *Classroom* relation using ARC, first two attributes, *Sector* and *Department*, are created, each having a foreign key to its corresponding table as shown in Table 2. Later, the check constraint Listing 3 is created over the *Sector* and *Department* attributes to test if the sum of the number of non NULL attributes is equal to one in each tuple. If true, then only one owner is defined and the model is still consistent. Otherwise, either all attributes are NULL (sum equals to zero) or there exists at least two non NULL attributes (sum is greater than one) as shown in Table 3 and the model is not consistent anymore. These constraints are enforced independently of application logic, ensuring consistency under concurrent access and incremental updates.

If the application attempts to insert a classroom with both sectors and departments defined or undefined, the database will throw an error, keeping its data consistent as shown in Listing 4. This

| Reservation | | | | |
|---|---|---|---|---|
| Lesson | Event | Classroom | Start_Time | End_Time |
| Algorithms | NULL | DI-01 | 2026-04-16 13:30:00 | 2026-04-16 15:30:00 |
| Calculus | NULL | PA-01 | 2026-04-16 14:30:00 | 2026-04-16 16:30:00 |
| NULL | Computer on the Beach XVII | DI-01 | 2026-04-16 15:30:00 | 2026-04-16 17:30:00 |

**Table 1: Allocations of lessons *Algorithms*, *Calculus* and event "*Computer on the Beach*" in *Reservation* table**

does mean that whoever is making queries has to adapt it to the expected owner type of the classroom, which we found preferable to the alternative of having separate tables for each classroom owner type, that could generate more complex queries nonetheless, although better fitting standard software engineering data modeling diagrams.

| Classroom | | |
|---|---|---|
| Name | Sector | Department |
| PA-01 | Sciences | NULL |
| DI-01 | NULL | Informatics |

**Table 2: Consistent polymorphic ownership of a classroom.**

| Classroom | | |
|---|---|---|
| Name | Sector | Department |
| PA-02 | NULL | NULL |
| DI-02 | Sciences | Informatics |

**Table 3: Inconsistent polymorphic ownership of a classroom.**

```
-- Room sector + department owner polymorphism
ALTER TABLE classroom ADD CONSTRAINT poly_owner
  CHECK ((
        (sector     IS NOT NULL)::integer +
        (department IS NOT NULL)::integer
      ) = 1
  );
```

**Listing 3: Polymorphic Constraint Definition**

```
-- Classroom without owner
Timetabling=# INSERT INTO classroom VALUES
  ('PA-02', NULL, NULL);
ERROR: new row for relation "classroom" violates check
     constraint "poly_owner"
DETAIL: Failing row contains (PA-02, null, null)

-- Classroom with more than one owner
Timetabling=# INSERT INTO classroom VALUES
  ('PA-02', 'Sciences', 'Informatics');
ERROR: new row for relation "classroom" violates check
     constraint "poly_owner"
DETAIL: Failing row contains (PA-02, Sciences,
     Informatics)
```

**Listing 4: Polymorphic Protection**

### 5.5 ARC Ad-hoc reservation constraint

Finally, to enforce the business rule defined in subsection 4.5 we apply the same ARC relationship in *Lesson* and *Event* from Table 1.

In the same way *Sectors* and *Departments* are foreign key attributes to relations of same name, it is also created a relation *Events* to hold information about the possible allocated events. To represent a mutually exclusive relationship, we define a check constraint that ensures that the sum of the non NULL values from both attributes is equal to 1 (Listing 5). The demonstration that the database protects itself against inconsistent insertions is already showed in Listing 4.

```
-- Either a lesson or an event polymorphism
ALTER TABLE reservation ADD CONSTRAINT poly_reservation
  CHECK ((
        (lesson IS NOT NULL)::integer +
        (event  IS NOT NULL)::integer
      ) = 1
  );
```

**Listing 5: Polymorphic Reservation Constraint Definition**

## 6 Allocation Algorithm

This section describes how the proposed data model can be coupled with an allocation procedure to generate an initial feasible classroom assignment.

### 6.1 Constraint Satisfaction Problem

The classroom assignment task is modeled as a constraint satisfaction problem (CSP) to explicitly describe the space of feasible allocations, where classrooms are assigned to lessons under a set of hard institutional constraints. These constraints include room availability, capacity limits, required resources, and temporal compatibility. In our context, variables correspond to lessons, domains correspond to the set of classrooms allowed for each lesson, and constraints are derived directly from the business rules described in section 4.

Each lesson is associated with concrete requirements, such as available time slots, room capacity, and mandatory resources (e.g., laboratories or multimedia equipment). These requirements partition classrooms into a set of *allowed rooms*, which yield feasible allocations, and *disallowed rooms*, which violate institutional constraints. These were described in subsection 4.6.

Similar formulations are common in school timetabling and can be mapped to Boolean satisfiability (SAT) models, where each potential room assignment for a lesson is classified as valid or invalid [5]. Classical CSP techniques, such as backtracking search and constraint propagation, are well understood, but quickly become impractical when applied to large university instances with multiple overlapping constraints.

The large number of constraints makes this problem complex and often NP-hard. The school timetabling problem itself was solved

using genetic algorithms [7] and evolutionary algorithms [17] in a similar manner. Approaches such as heuristics [6] are popular to achieve near-optimal results for the algorithm in a reasonable time too.

As a result, practical timetabling systems typically rely on heuristic or optimization-based methods to efficiently compute feasible solutions. It is also important to take into consideration that a near-optimal solution is most likely what's possible, since it's very hard to perfectly model the problem into an instance where the most optimal solution reflects the real world. Even more so when considering that including concepts such as "classroom preferences" can be involved.

## 6.2 The Hungarian Algorithm

The Hungarian Method [12], also known as the Munkres Algorithm, is a combinatorial optimization algorithm created to solve the *assignment problem*. In its most optimized form, this algorithm executes in $O(n^3)$ time.

The Hungarian Algorithm works by estimating a cost of assigning each individual agent to their respective tasks, with the goal to minimize this cost of each pair-wise assignment. This can be used for many real-world applications, but is specially useful for scheduling and timetabling tasks.

We employ the Hungarian Algorithm to solve a restricted assignment subproblem corresponding to a single time slice of the timetable. Rather than addressing the full timetabling problem at once, the algorithm is used to compute an initial feasible assignment of lessons to classrooms under a fixed temporal context. Then, it incrementally fills the timetable with feasible solutions until there is no longer any possible room allocation.

The key advantage of using the Hungarian Algorithm in this context is that it solves the assignment problem in *polynomial time* and works particularly well when there are *balanced sets of tasks and agents*. In our case, each room assignment corresponds to a task, and the "cost" can represent various factors such as proximity to other classes, resource requirements, or room utilization.

## 6.3 Combining Heuristics and Munkres

The algorithm we propose uses a weekly timetabling input, which better represents the actual university classroom allocation model, since classes are not reassigned to other rooms every week. Instead, normally, there is a weekly schedule that the classes follows, making the entire timetable for that class predictable. That way, it's possible to create smaller and simpler subproblems that can be solved using the Hungarian Method, and using it continuously to incrementally solve the original, more complex, problem.

One of the ways we keep the Munkres instance small is by using the information of hard room restrictions, modeled in subsection 4.6, to form the input data for the algorithm. This is also necessary because in a typical Munkres execution, any task can be assigned to any agent, which is not true for the classroom assignment problem.

This way the hard restrictions are respected by the algorithm by default, since they never are considered for allocation, and the soft restrictions are used to estimate the assignment cost when running Munkres. Other methods propose using "infinite costs" when dealing with hard restrictions. However, parts of the algorithm would have to be changed, and the presence of another assignment, even with infinite cost, could negatively affect the effectiveness of the proposed solver.

Algorithm 1 is used to solve a single room assignment instance, which is basically just an adaptation of Munkres meant to solve a weekly schedule of a small number of classes, and it is solved in polynomial time.

**Algorithm 1:** Hungarian Algorithm for a single timetable slice

```
Function solve_range(rooms, weekly_lessons, weights):
    M ← build_cost_matrix(weekly_lessons, rooms, weights)
    /* takes into consideration distances and capacity usage */
    attributed_indexes ← Munkres(M)
    assignment ← assign_rooms(weekly_lessons, rooms)
    return assignment
```

The calculation of costs can be updated to better fit any educational timetabling instance. In our study, prioritizing a higher room capacity usage with linear terms and using squared or even higher factors for distances resulted in a near-optimal solution. Specially when fine-tuning the weights for each term during test executions of the solver.

Still, it is necessary to use this algorithm multiple times, since it is only meant to solve inputs that represent a proper assignment problem, not the more general classroom assignment. Therefore, the algorithm we propose to solve the classroom assignment problem described in this work is described in pseudocode by Algorithm 2.

**Algorithm 2:** Incremental heuristic-based classroom assignment

```
Function room_attributor(rooms, weekly_lessons room_constraints):
    hours ←
      get_most_conflict_hours(rooms, weekly_lessons, constraints)
    assignments_by_hour ← {}
    for hour in hours do
        valid_weekly
          ← get_valid(rooms, weekly_lessons, constraints, hour)
        sorted_weekly ← sort_by_availability(rooms, valid_weekly)
        by_hour[hour]
          ← solve_range(rooms, sorted_weekly, weights)
    end
    assignments ← parse_assignments(by_hour)
    return assignments
```

This algorithm needs all rooms, weekly schedules, hard room restrictions and soft constraint weights as input to execute. The hard constraints are the same ones defined in subsection 4.6, and the solver uses this information to create each Hungarian Algorithm input, making sure no hard restriction is ever violated.

Each Munkres execution is prioritized according to heuristics. In our work, classes that have more options available to them are left for last. Similarly, classes with more students are assigned classrooms first, since it is true for our case study that larger classrooms are rarer. Moreover, certain time ranges during the day have more classes, so these are also solved first, and in our case, hour by hour.

This approach is computationally efficient and allows the algorithm to run in a matter of *seconds*, even with a large number of rooms and lessons. Our results with UFPR, with a problem size of

roughly 4800 classes, with 2 weekly lessons each on average, and approximately 350 rooms executed in approximately 4.4 seconds. In terms of effectiveness, the algorithm managed to allocate 98% of all weekly lessons while respecting all hard constraints.

Unlike other methods (e.g., backtracking or exhaustive search), the Hungarian Algorithm combined with prioritization heuristics for an incremental algorithm ensures that the solution is found in polynomial time, making it well-suited for real-time timetabling. Even more so to try different parameters to better optimize the result for the real world.

To have better results, it's as simple as creating heuristics to prioritize and give different weights for different classes/lessons, as a few other approaches had some success using them [2][10]. Particularly trying to prioritize classroom preferences, how many other classroom choices may be available for that specific lesson, how far the rooms are, among others. In the UFPR case study, weighting factors such as classroom proximity, the number of alternative rooms available for a lesson, and classroom preferences proved effective in producing feasible and practical allocations.

## 7 Conclusion

In this paper, we addressed the classroom assignment problem in university timetabling, focusing primarily on data modeling and integrity enforcement. By combining a relational database model with a heuristic-based allocation procedure, we presented a solution that supports both the initial allocation of classrooms and subsequent timetable updates throughout a semester.

Our approach emphasizes the role of a robust data model in enforcing real-world business rules, complementing the optimization process rather than relying solely on algorithmic checks. While many existing solutions focus primarily on allocation algorithms, this work highlights how database-level constraints contribute to system consistency, particularly when timetable changes occur during the semester, such as rescheduling or the addition of new events.

The proposed approach was illustrated using real data from the Federal University of Paraná, demonstrating its applicability in a real-world academic environment. Although the case study is specific to a single institution, the underlying modeling principles are general and can be adapted to other educational institutions with similar timetabling requirements.

Future work includes extending the data model to support additional institutional constraints and conducting a more detailed evaluation of alternative allocation strategies and heuristic configurations. Further analysis of algorithm performance under larger and more dynamic scenarios is also a relevant direction for future research.

In conclusion, this work demonstrates that effective database modeling, combined with standard allocation techniques, provides a consistent and scalable foundation for classroom assignment in university timetabling, ensuring compliance with institutional business rules throughout the scheduling process.

## References


[1] Tenindra Abeywickrama, Victor Liang, and Kian-Lee Tan. 2021. Optimizing bipartite matching in real-world applications by incremental cost computation. *Proc. VLDB Endow.*, 14, 7, (Mar. 2021), 1150–1158. DOI: 10.14778/3450980.3450983.

[2] Leena N. Ahmed, Ender Özcan, and Ahmed Kheiri. 2015. Solving high school timetabling problems worldwide using selection hyper-heuristics. *Expert Systems with Applications*, 42, 13, 5463–5471. DOI: https://doi.org/10.1016/j.eswa.2015.02.059.

[3] Ahmed A Ataullah and Frank Wm Tompa. 2011. Business policy modeling and enforcement in databases. *Proceedings of the VLDB Endowment*, 4, 11, 921–931.

[4] Michael Carter and Craig Tovey. 1992. When is the classroom assignment problem hard? *Operations Research*, 40, (Feb. 1992). DOI: 10.1287/opre.40.1.S28.

[5] Emir Demirović and Nysret Musliu. 2017. Maxsat-based large neighborhood search for high school timetabling. *Computers & Operations Research*, 78, 172–180. DOI: https://doi.org/10.1016/j.cor.2016.08.004.

[6] Árton P. Dorneles, Olinto C.B. de Araújo, and Luciana S. Buriol. 2014. A fix-and-optimize heuristic for the high school timetabling problem. *Computers & Operations Research*, 52, 29–38. DOI: https://doi.org/10.1016/j.cor.2014.06.023.

[7] Dipankar Dutta, Jaya Sil, and Paramartha Dutta. 2020. A bi-phased multi-objective genetic algorithm based classifier. *Expert Systems with Applications*, 146, 113163. DOI: https://doi.org/10.1016/j.eswa.2019.113163.

[8] Abdelkarim Elloumi, Hichem Kamoun, Bassem Jarboui, and Abdelaziz Dammak. 2014. The classroom assignment problem: complexity, size reduction and heuristics. *Applied Soft Computing*, 14, 677–686. DOI: https://doi.org/10.1016/j.asoc.2013.09.003.

[9] Ramez Elmasri, Sham Navathe, et al. 2014. *Fundamentals of database systems*. Vol. 7. Pearson.

[10] George H.G. Fonseca, Haroldo G. Santos, and Eduardo G. Carrano. 2016. Integrating matheuristics and metaheuristics for timetabling. *Computers & Operations Research*, 74, 108–117. DOI: https://doi.org/10.1016/j.cor.2016.04.016.

[11] A. Sima Uyar, Ender Ozcan, and Neil Urquhart, (Eds.) 2013. *Educational timetabling*. *Automated Scheduling and Planning: From Theory to Practice*. Springer Berlin Heidelberg, Berlin, Heidelberg, 91–108. ISBN: 978-3-642-39304-4. DOI: 10.1007/978-3-642-39304-4_4.

[12] H. W. Kuhn. 1955. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2, 1-2, 83–97. DOI: https://doi.org/10.1002/nav.3800020109.

[13] Alan Souza Prado and SR de Souza. 2014. Problema de alocação de salas em cursos universitários: um estudo de caso. *Anais do XLVI Simpósio Brasileiro de Pesquisa Operacional*, 2054–2065.

[14] Ventseslav Kirilov Shopov and Vanya Dimitrova Markova. 2021. Application of hungarian algorithm for assignment problem. In *2021 International Conference on Information Technologies (InfoTech)*, 1–4. DOI: 10.1109/InfoTech52438.2021.9548600.

[15] Joo Siang Tan, Say Leng Goh, Graham Kendall, and Nasser R. Sabar. 2021. A survey of the state-of-the-art of optimisation methodologies in school timetabling problems. *Expert Systems with Applications*, 165, 113943. DOI: https://doi.org/10.1016/j.eswa.2020.113943.

[16] R. A. Oude Vrielink, E. A. Jansen, Erwin W. Hans, and Jos van Hillegersberg. 2019. Practices in timetabling in higher education institutions: a systematic review. *Ann. Oper. Res.*, 275, 1, 145–160. DOI: 10.1007/S10479-017-2688-8.

[17] Shuaipeng Yuan, Tieke Li, and Bailin Wang. 2020. A co-evolutionary genetic algorithm for the two-machine flow shop group scheduling problem with job-related blocking and transportation times. *Expert Systems with Applications*, 152, 113360. DOI: https://doi.org/10.1016/j.eswa.2020.113360.