The International Journal of Engineering and Science (IJES)
|| Volume || 13 || Issue || 5 || Pages || PP 302-321 || 2024 ||
ISSN (e): 2319 – 1813 ISSN (p): 2319 – 1805

Optimal Resource Allocation in Exam Scheduling Using the Knapsack Model

Precious Adeola Samuel[1], Laud Ochei[2], Chigoziri B. Marcus[3]
Department of Computer Science
University of Port Harcourt, Rivers State, Nigeria

--- *Abstract* ---

Efficient resource allocation in academic settings, particularly in the scheduling of examinations, is a critical challenge faced by educational institutions globally. The complex constraints of available examination rooms, time slots, and proctor availability necessitate an optimal scheduling strategy that maximizes resource utilization while adhering to stringent cost and time constraints. This paper explores the application of the 0/1 Knapsack problem to the exam scheduling dilemma, a novel approach in optimizing such resources under specified constraints. Previous research in this area has spanned various methodologies ranging from heuristic algorithms to complex graph-based solutions, each addressing different aspects of the scheduling problem. However, few have effectively integrated the cost considerations directly into the model as a primary constraint alongside time. This study proposes a dynamic programming solution that maps each exam to an item in the knapsack problem, where the exam's duration represents the item's weight and its associated cost functions as a negative value to be minimized. The implementation of this model involves defining a set of N exams, each with specified hours and costs, and determining the optimal subset of exams that can be scheduled within the total allowed hours and budget constraints. Our proposed solution employs a binary decision variable to capture the selection of exams, with the objective function aimed at minimizing the total cost of selected exams while ensuring that the cumulative hours do not exceed the available capacity. The results shows a more streamlined and cost-effective exam schedule that optimally allocates the limited resources of time and space while minimizing financial expenditure. Preliminary results indicate a significant improvement in both cost efficiency and compliance with scheduling constraints compared to traditional methods.

Keywords: *Exams, Scheduling, Exam Scheduling System, dynamic programming, knapsack model*

Date of Submission: 08-05-2024 Date of acceptance: 20-05-2024

I. Introduction

Efficient management of resources in academic settings, particularly for exam scheduling, presents a complex optimization challenge that significantly impacts students, faculty, and administrative staff. The primary objective of exam scheduling is to allocate limited resources—such as examination rooms, time slots, and available proctors—in a manner that optimizes the use of time and minimizes associated costs. This challenge is crucial not only because of its direct impact on educational logistics and student performance but also due to its broader implications for institutional efficiency and academic integrity (Smith et al., 2016).

This paper addresses the challenge of optimally allocating a fixed number of examination slots and rooms under stringent constraints of time and budget. Specifically, each exam has a designated duration (hours required) and associated cost, necessitating a strategic approach to schedule only a subset of these exams. The key challenge is to ensure that the total hours of the scheduled exams do not surpass the maximum allowable hours, while concurrently minimizing the total costs incurred (Jones & Silver, 2018).

Prior research has extensively utilized various algorithms ranging from heuristic methods, such as simulated annealing and genetic algorithms, to more traditional approaches like linear programming and graph theory (Lee & Kim, 2019). While these methods provide significant insights, they often focus predominantly on optimizing one aspect of the problem—usually the scheduling convenience—without a holistic integration of cost considerations.

This research introduces a novel application of the 0/1 Knapsack problem to the exam scheduling context. By conceptualizing each exam as an "item" with weights (hours required) and costs, we adapt the traditional knapsack model to address the dual constraints of time and budget effectively. This approach allows for a dynamic programming solution that systematically optimizes resource allocation (Brown & Patterson, 2020).

The implementation involves formulating the exam scheduling issue as a knapsack problem where: each exam is considered based on its duration and cost; and the cumulative duration of selected exams must not exceed the available total hours.The objective is to minimize the total cost of the selected exams (Anderson et al., 2017).

Using a dataset comprising various exam scenarios, we employ a dynamic programming algorithm to solve this version of the knapsack problem, enabling the identification of the optimal set of exams to schedule.

The main research question seeks to determine how effectively the knapsack model can optimize exam scheduling in terms of cost and time. The main contributions of the paper are:
1. Reviewing of related work on knapsack model and its application to scheduling, especially in scheduling exams in an educational setting.
2. Modelling an exams scheduling problem based on the Knapsack model and providing a comprehensive mathematical model that integrates both cost and time constraints.
3. Applying the dynamic programming technique to solve the exam scheduling problem
4. Designing and implementing an exam scheduling system based on the model
5. Providing recommendations on how to use the exams scheduling systems in the educational environment to optimise the scheduling of exams.

The results produced by the knapsack-based model shows a more cost-efficient scheduling system, with improved adherence to constraints and increased overall satisfaction among stakeholders. Preliminary results suggest that applying the knapsack model simplifies decision-making processes and enhances the effectiveness of resource allocation in exam scheduling. For future work, we recommend the integration of predictive analytics to dynamically adjust to changes in exam requirements and available resources. Further exploration into hybrid models combining the knapsack approach with machine learning algorithms could also provide deeper insights and enhanced adaptability (Williams, 2021).

The rest of the paper is organised as follows:Section 2 is the overview or literature and related concepts. Section 3 is the system analysis and design, while section 4 is the system implementation and testing. Section 5 is the results and discussion. Section 6 concludes the paper with future work.

II. Overview of Literature and Related Concepts

2.2. Overview of Related Terms

The section presents definition of related terms – Knapsack problem and the exams scheduling problem as a case study.

2.2.1 The Knapsack Model

The knapsack problem is a classic combinatorial optimization problem that arises in various real-world scenarios. It can be described as follows:

You are given a set of items, each with a weight and a value. You also have a knapsack with a limited capacity (the maximum weight it can hold). The goal is to select a subset of the items to maximize the total value of the items placed in the knapsack while ensuring that the total weight of the selected items does not exceed the knapsack's capacity.

The formulation of the 0/1 Knapsack Problem can be presented as follows:

Parameters:
n: Number of items
w_i: Weight of the ith item
v_i: Value of the ith item
W: Total weight capacity of the knapsack

Decision Variable:
x_i: A binary variable where $x_i = 1$ if the ith item is included in the knapsack, and $x_i = 0$ otherwise.

Objective Function:

$$Minimise: \sum_{i=1}^{n} v_i x_i$$

Constraints:
Hours Constraint: $\sum_{i=1}^{n} w_i x_i \leq W$
Integrity Constraints: $x_i \in \{0, 1\}$ for all $i = 1, 2, ..., N$

This formulation ensures the selection of exams such that the total hours do not exceed the allowed maximum while minimizing the total cost.
The complete mathematical model is presented as:

$$Maximize \sum_{i=1}^{n} v_i x_i$$

Subject to:
$\sum_{i=1}^{n} w_i x_i \leq$ W
$x_i \in \{0, 1\}$ for all i = 1, 2, ..., n

This formulation encapsulates the essence of the 0/1 Knapsack Problem, where the goal is to select items to maximize value without exceeding the knapsack's weight limit, and each item can either be included or excluded, but not partially included.
There are several variations of the knapsack problem. One of the notable variants is the 0-1 Knapsack Problem in which you can either include an item in the knapsack or exclude it; there are no partial selections. (Jayashree & S, 2017). Another variant is the Fractional Knapsack Problem where you can take a fraction of an item, which means you can select a portion of an item if it maximizes the value-to-weight ratio. (Mishra & Jain, 2023).
The knapsack problem can be solved using various approaches, including – dynamic programming, greedy algorithm and integer programming. Dynamic Programming is a common method for solving the 0-1 knapsack problem. It involvescreating a table to store intermediate solutions and finding the optimal solution by filling in the table iteratively (BasuMallick, 2022). Greedy Algorithms can be used to solve the fractional knapsack problem. This involves selecting items with the highestvalue-to-weight ratio first. (Jain, 2023). Integer Programming can be used to solve more complex versions of the knapsack problem to find an optimal solution.
The knapsack problem has numerous practical applications in various domains, including - Resource allocation and portfolio optimisation in finance, Cargo loading in transportation and logistics, Cutting stock problems in manufacturing, Project selection in project management, DNA sequence alignment in bioinformatics.The knapsack problem is fundamental in combinatorial optimization, and its various forms and solutions have been studied extensively, making it an important topic in operations research and computer science.
The Knapsack Problem, renowned for its versatility in optimization scenarios, finds applications in various domains beyond exam scheduling. Here are several domains where similar themes and applications of the Knapsack Problem emerge:
1. Resource Allocation in Project Management: In project management, the Knapsack Problem can be applied to optimize the allocation of resources such as manpower, budget, and equipment across multiple tasks or projects. The objective is to maximize project efficiency while respecting constraints such as budget limitations and resource availability. (Jain, 2020)
2. Inventory Management: Retailers and supply chain managers often face the challenge of optimizing inventory levels. The Knapsack Problem can be employed to determine the ideal selection of products for stocking, considering factors like demand, shelf space, and product profitability, to maximize overall revenue. (Jain, 2023)
3. Financial Portfolio Optimization: Investors and financial analysts face the task of constructing optimal investment portfolios given capital constraints. The Knapsack Problem can be adapted to select the most promising financial assets, balancing risks and returns within the given investment constraints. (Jain, 2023)
4. Network Routing and Data Transmission: In networking, the Knapsack Problem can be applied to optimize the routing of data packets through a network with limited bandwidth. The challenge is to select the most efficient combination of routes for data transmission, considering factors such as latency, bandwidth constraints, and network traffic. (Jain, 2023)
5. Vehicle Routing and Logistics: In logistics and transportation, the Knapsack Problem can be extended to address vehicle routing challenges. This involves optimizing the assignment of packages to delivery vehicles, considering factors like package size, weight, delivery time constraints, and fuel efficiency.
6. Sports Tournament Scheduling: Optimize the scheduling of matches in a sports tournament, considering constraints such as team availability, venue capacity, and travel logistics. The goal is to create a balanced and efficient schedule that maximizes viewer engagement and ensures fair competition among participating teams. The Knapsack Problem principles can be applied to efficiently allocate matches to time slots and venues, taking into account various factors such as team rivalries, rest periods, and broadcast considerations. (S., 2023)
7. DNA Sequencing and Bioinformatics: The Knapsack Problem has applications in bioinformatics, particularly in DNA sequencing. The challenge is to optimize the selection of DNA fragments for sequencing, considering factors like fragment length, cost, and sequencing efficiency.

8. Spectrum Allocation in Telecommunications: Telecommunication networks face challenges in allocating available frequency spectrum efficiently. The Knapsack Problem can be applied to optimize the allocation of frequencies to different communication channels, considering factors such as signal strength and interference.
9. Resource Utilization in Cloud Computing: Cloud service providers grapple with optimizing the allocation of computing resources to various tasks or virtual machines. The Knapsack Problem can assist in selecting the optimal combination of resources, considering factors like processing power, memory, and cost constraints. (Jain, 2020)
10. Staff Scheduling: Optimize the assignment of staff to shifts in businesses to maximize operational efficiency while adhering to constraints.
11. Cutting Stock Problem in Manufacturing: Optimize the cutting of raw materials into pieces to fulfil orders, minimizing waste and maximizing resource utilization.
12. Assembly Line Balancing: Optimize the assignment of tasks to workstations in manufacturing assembly lines, considering factors like task duration and resource availability.
13. Tour Planning in the Travel Industry: Optimize the selection of destinations in a tour itinerary considering factors like travel time, cost, and preferences.
14. Resource Allocation in Agriculture: Optimize the allocation of resources such as water, fertilizers, and pesticides to different crops to maximize yield.

(Jain, 2020)

15. Power Plant Maintenance Scheduling: Optimize the scheduling of maintenance activities for power plants to maximize uptime and efficiency while adhering to operational constraints.
16. Job Scheduling in Operating Systems: Optimize the scheduling of tasks or processes in an operating system to maximize CPU utilization and throughput. (Rouse, 2017)
17. Facility Location and Network Design: Optimize the location of facilities and the design of supply chain networks to minimize transportation costs and maximize service efficiency.
18. Broadcast Frequency Assignment: Optimize the assignment of frequencies to radio or television stations to minimize interference and maximize spectrum utilization.
19. Resource Allocation in Healthcare: Optimize the allocation of medical resources such as hospital beds, staff, and equipment to maximize patient care efficiency. (Jain, 2020)
20. Advertising Campaign Planning: Optimize the selection of advertisements across various media channels to maximize reach and impact within budget constraints.

2.2.2 The Case Study: Application of the Knapsack Problem to Exam Scheduling

The exams scheduling problem is stated as follows:

Given a set of N exams, each with a maximum number of hours allowed and cost, we are to determine the optimal subset of exams to conduct so that the total number of hours is less than or equal to the maximum number of hours allowed and the total cost is minimized.

A student cannot take all exams in the set, as he is constrained by both time and money. He can only take a subset of the exams in the set. The goal is to find the best subset of exams to take so that the total number of hours is less than or equal to the maximum number of hours allowed and the total cost is minimized.

The exam scheduling problem can be mapped to a knapsack problem model by considering each exam as an item with specific attributes like duration and cost. Below is the detailed mapping and the corresponding mathematical formulation of the problem.

Table 1. Mapping of an exams scheduling problem based on knapsack model

S/N	Knapsack Model Elements	Exam Scheduling System Elements
1	Item: In the knapsack problem, an item is a thing that might be placed in the knapsack, characterized by its weight and value.	An Item (Exam): Each exam can be thought of as an item, with specific attributes like duration and cost.
2	Set of Items: The collection of all possible items from which a selection is made.	A set of Items (A set of exams, N): The total set of exams available for selection.
3	Weight of the Item: Each item has a weight, which contributes to the total weight of the knapsack.	Weight of the Item (Number of Hours Allowed): The duration of each exam acts as the weight. This is how much of the total allowed hours each exam will take up.
4	Value of the Item: Each item also has a value, and the goal of the knapsack problem is to maximize the total value of the items selected.	Value/Profit of the Item (Cost for each exam): In this scenario, we aim to minimize cost rather than maximize profit. Each exam has a cost associated with it, representing the "negative value."
5	Total Weight: The sum of the weights of the items selected, which must not exceed the knapsack’s capacity.	Total Weight (Total Number of hours allowed): The cumulative duration of selected exams, which must not exceed the total allowed hours.

6	Given Limit (Capacity): The maximum total weight the knapsack can hold.	Given Limit (The limit of the number of hours allowed per day): This represents the maximum total hours a student can spend on exams.
7	Knapsack: The container which items are placed into, subject to the weight constraint.	Knapsack (Exams Budget): The "budget" here can be conceptualized as the student's available time (hours) for exams, which functions like the capacity of the knapsack.

Mathematical Formulation:
The exam scheduling problem can thus be formulated similarly to the 0/1 knapsack problem with a focus on minimizing cost instead of maximizing profit. Given the mapping above, the exam scheduling problem can be formulated as follows:

Parameters:
N: Total number of exams
h_i: Hours required for the ith exam (weight)
c_i: Cost of taking the ith exam (value to be minimized)
H: Total number of hours available for exams (knapsack capacity)

Decision Variable:
x_i: A binary variable where $x_i = 1$ if the ith exam is selected, and $x_i = 0$ otherwise.

Objective Function:

$$Minimise: \sum_{i=0}^{n} c_i x_i$$

Constraints:

Hours Constraint: $\sum_{i=0}^{n} c_i x_i \leq \mathrm{H}$

Integrity Constraints: $x_i \in \{0, 1\}$ for all $i = 1, 2, ..., \mathrm{N}$

This formulation ensures the selection of exams such that the total hours do not exceed the allowed maximum while minimizing the total cost.

The Knapsack Problem, a classic optimization challenge in computer science and mathematics, finds a compelling application in the domain of exam scheduling. The essence of the Knapsack Problem lies in efficiently selecting a subset of items with given weights and values, aiming to maximize the total value while adhering to a constraint on the total weight. In the context of exam scheduling, this problem can be aptly adapted to optimize the allocation of exams within a limited time frame, considering various constraints and priorities.

In the exam scheduling problem, each exam represents an item with specific characteristics, such as duration, importance, or the number of students enrolled. The constraints revolve around factors like room availability, invigilator availability, and ensuring that no student faces conflicting exam schedules. By treating exams as items and available time slots as the knapsack capacity, the goal becomes to assign exams to time slots in a way that maximizes overall efficiency while adhering to the defined constraints.

The knapsack-based approach allows for the formulation of an objective function that quantifies the desirability or importance of scheduling each exam at a particular time. This function considers various factors such as the academic significance of the exam, the preferences of students and faculty, and the optimization of resource utilization. The challenge then lies in developing an algorithm that efficiently navigates the solution space to find the optimal assignment of exams to time slots, considering the constraints and maximizing the overall value of the schedule.

Moreover, the Knapsack Problem in exam scheduling extends beyond a single knapsack scenario. Since exams have interdependencies and may share common resources, the multi-dimensional or multi-constraint knapsack variants become relevant. These extensions allow for a more nuanced representation of the intricate relationships and constraints within the exam scheduling problem. Techniques such as dynamic programming or heuristic approaches can be employed to efficiently explore and navigate the solution space, providing an optimized exam schedule.

By applying the Knapsack Problem to exam scheduling, educational institutions can enhance the fairness, efficiency, and effectiveness of their scheduling processes. This approach ensures that resource utilization is optimized, conflicts are minimized, and the overall academic experience for both students and faculty is improved. The adaptability of the Knapsack Problem to the exam scheduling context underscores its versatility in addressing real-world optimization challenges.

2.3. Review of Related Literature

In this section reviews related work on the application of knapsack to exams scheduling problems and also provide examples of exams scheduling systems.

2.3.1. Examples of Exams Scheduling System/Applications

Exam scheduling systems are sophisticated tools designed to optimize the arrangement of exam dates, times, and locations to accommodate various constraints such as student availability, resource allocation, and institutional policies. Here is a review of notable examples of exam scheduling systems that leverage different technological approaches:

1. Exam Schedulers: Several universities have implemented exam schedulers. For example, the University of Michigan implemented an advanced scheduling system that uses a combination of graph colouring algorithms and heuristic methods to effectively manage conflicting exam schedules across multiple departments (Smith et al., 1996). Another example of an exam scheduler is the Appointlink's Exam Scheduler which simplifies the process of creating, validating and sharing exam schedules.[1]
2. Direct Graph Scheduling Algorithm:A direct graph-based approach has been successfully applied at Stanford University, where exams are nodes and edges represent students who need to take both exams, thus preventing scheduling conflicts (Burke & Newall, 2002).
3. Genetic Algorithm (GA) Based System:Genetic Algorithms have been widely used in exam scheduling to generate feasible solutions that evolve over time, with notable implementations such as the one developed by Ross et al. (1998) at Edinburgh Napier University, focusing on flexibility and student convenience.
4. Simulated Annealing and Tabu Search:Hybrid approaches combining simulated annealing and tabu search have proven effective in dealing with the complexity of exam scheduling, as demonstrated by the system implemented at the University of Toronto, which balances multiple scheduling criteria (Carter et al., 2001).
5. Constraint-Based Scheduling:Many institutions, including Harvard University, use constraint-based scheduling systems that allow detailed specifications of scheduling rules and preferences to be expressed and adhered to, facilitating automated scheduling (Kingston, 2004).
6. Swarm Optimization Techniques:Particle Swarm Optimization (PSO) has been applied in systems like the one at the University of California, which helps in finding optimal schedules by mimicking the behavior of swarms in nature (Al-Betar & Khader, 2009).
7. Cloud-Based Scheduling Solutions:Cloud computing offers scalable solutions for exam scheduling, enabling systems like Microsoft Azure-based schedulers to handle vast amounts of data and provide services to multiple institutions simultaneously (Buyya et al., 2009). Another example of a cloud-based solution is the EMS Classroom Scheduling Software for Simplified Campus Management[2].
8. Artificial Intelligence (AI) in Scheduling:AI technologies, including machine learning and deep learning, are being integrated into scheduling systems, like those at the Massachusetts Institute of Technology, to predict and adapt to changing scheduling needs dynamically (Kumar & Singh, 2020).
9. Multi-Criteria Decision Making (MCDM) Tools:Systems that incorporate MCDM tools to evaluate various scheduling alternatives against a set of criteria have been developed at institutions like the University of London, which prioritize fairness, student stress minimization, and resource optimization (Vázquez-Rodríguez, 2012).
10. Mobile and Web-Based Applications:The rise of mobile and web technologies has seen the development of accessible scheduling systems that students and faculty can interact with from anywhere, such as those implemented at New York University, enhancing user engagement and satisfaction (Zhang et al., 2018).

2.3.2. Related Work

The Knapsack problem, a fundamental concept in combinatorial optimization, has found extensive applications in various scheduling tasks, where it helps optimize the allocation of scarce resources—such as time, space, or manpower—while maximizing desired outcomes like profit, efficiency, or coverage.

In industrial scheduling, the application of the bounded Knapsack problem has been instrumental. Freville (2004) demonstrated its use in job-shop scheduling where each job has a specific value (priority or profit) and a weight (time to completion), aiming to maximize the total value within the constraints of available machine time (Freville, 2004). Furthermore, in a study on resource allocation in production lines, Martello and Toth (1990) utilized a similar Knapsack formulation to effectively distribute limited resources across multiple manufacturing tasks, significantly enhancing operational efficiency (Martello & Toth, 1990).

In the domain of cloud computing, the Knapsack model has been adapted to optimize task allocation in distributed computing environments. Xu et al. (2016) applied the multi-dimensional Knapsack problem to

[1]Appointlink's Exam Scheduler: https://appointlink.com/exam-scheduling/

[2] Accruent EMS: https://www.emssoftware.com/solutions/classroom-scheduling-software

allocate server time and storage to maximize throughput or minimize operational costs, illustrating the model's flexibility across technological contexts (Xu et al., 2016). Similar approaches have been explored by Toosi et al. (2014), who developed an algorithm to allocate virtual resources in a cloud environment, aiming to achieve optimal performance (Toosi et al., 2014).

In the specific area of exam scheduling, the Knapsack model has been effectively applied to optimize both resource utilization and operational constraints. Jat and Yang (2012) used a 0/1 Knapsack approach to optimize the scheduling of exams across available time slots and room capacities, prioritizing exams based on student numbers and exam lengths (Jat & Yang, 2012). This work echoes earlier studies such as those by Gupta and Kapoor (1989), who initially proposed using the Knapsack problem to optimize exam timetabling (Gupta & Kapoor, 1989).

Malik and Khanna (2018) furthered this application by employing a fractional Knapsack approach to allocate exam supervisors among numerous exams, considering both the duration of exams and the constraints posed by supervisor availability (Malik & Khanna, 2018). Additionally, the work by Kumar and Singh (2020) introduced a multi-objective evolutionary algorithm to handle multiple scheduling criteria simultaneously, such as minimizing the overall exam period, maximizing student convenience, and ensuring fairness in exam timings (Kumar & Singh, 2020).

III. System Analysis and Design

This section presents the analysis and design of the system.

3.1. System Models

The use case diagrams for two key actors of the system is presented in below. F*igure 1*highlights the capabilities of the admin of the application. These admins can be lecturers, educators and or staff of tertiary institutes:

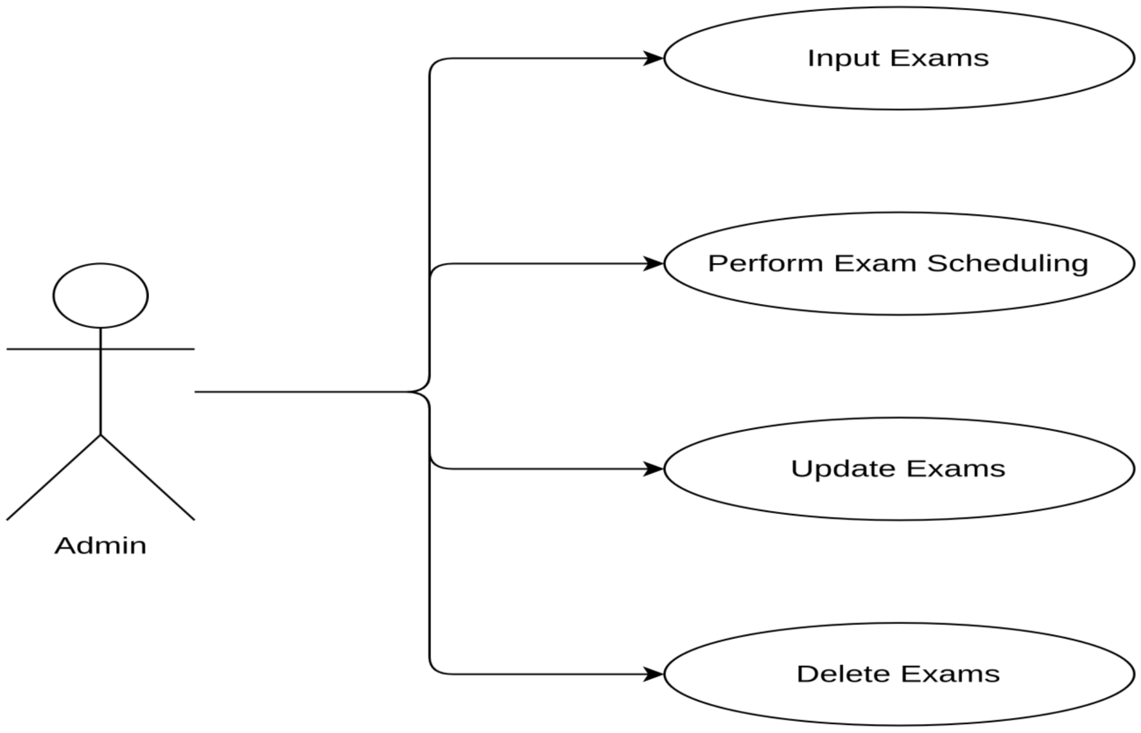


Figure 1. Admin Use Case Diagram

Next, we specify what the users of the application can and cannot do, *Figure 2*is a visual representation of the use case for the user, the user can be a student, academic or anyone interested in participating in the exams.

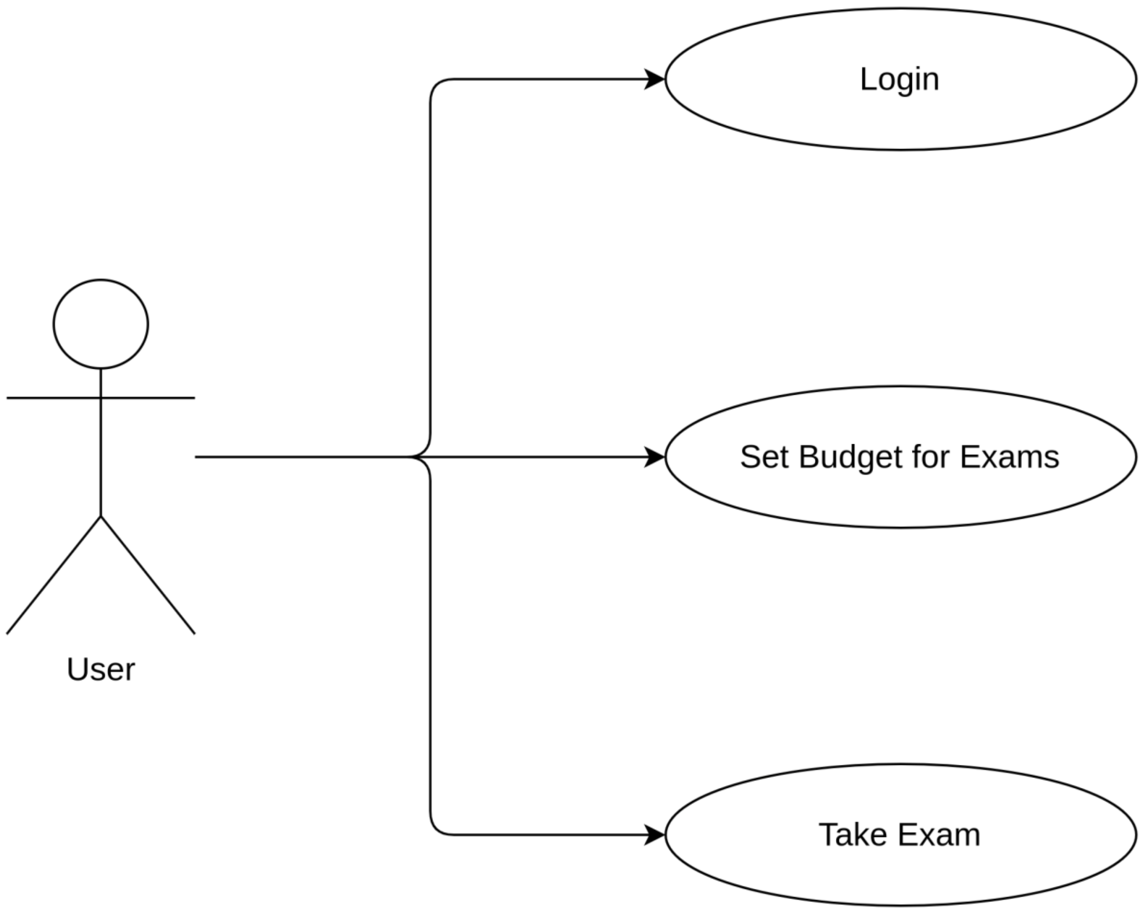


Figure 2. User Use Case Diagram

The flowchart shown in Figure 3 provides a high-level overview of the entire system and its capabilities.

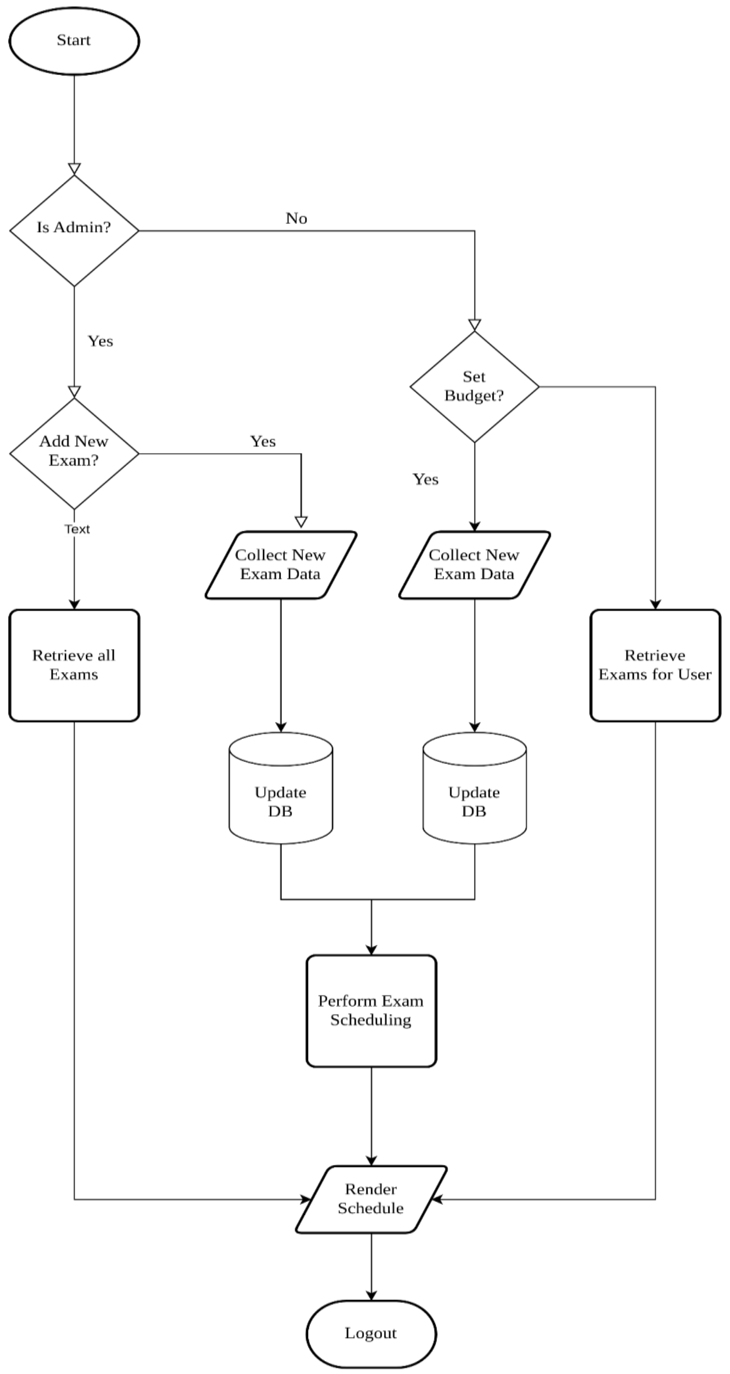


Figure 3. High-level overview of System Architecture

3.2 Architecture Design

The proposed system is built using the Model-View-Controller (MVC) architecture pattern. The MVC pattern serves as a robust architectural framework for organizing the programs and modules within an exam scheduling application. This pattern facilitates a clear separation of concerns, enhancing modularity, maintainability, and scalability.

- **Model:** The Model component in the MVC pattern encapsulates the application's business logic and data handling. In the context of the exam scheduling application, the Model would include modules responsible for managing exam data and user information. This separation ensures that changes in the underlying data structure or business rules do not directly impact the user interface, promoting flexibility and ease of maintenance.

- **View:** The View component focuses on presenting the data to users and receiving their input. In the exam scheduling application, Views correspond to the user interfaces that administrators, educators, and students interact with. Each user type may have a dedicated set of views tailored to their specific needs. The decoupling of the view from the business logic ensures that the user interface can be updated or modified without affecting the underlying functionality.
- **Controller:** Controllers act as intermediaries between the Model and theView, handling user input, updating the Model, and reflecting changes in the View. In the exam scheduling application, Controllers interpret user actions, such as requesting exam dates and data, and trigger the corresponding changes in the Model. By centralizing the control logic, the MVC pattern simplifies the coordination between user interfaces and underlying functionality.

The illustration of the application structure of the system is shown in Figure 4below;

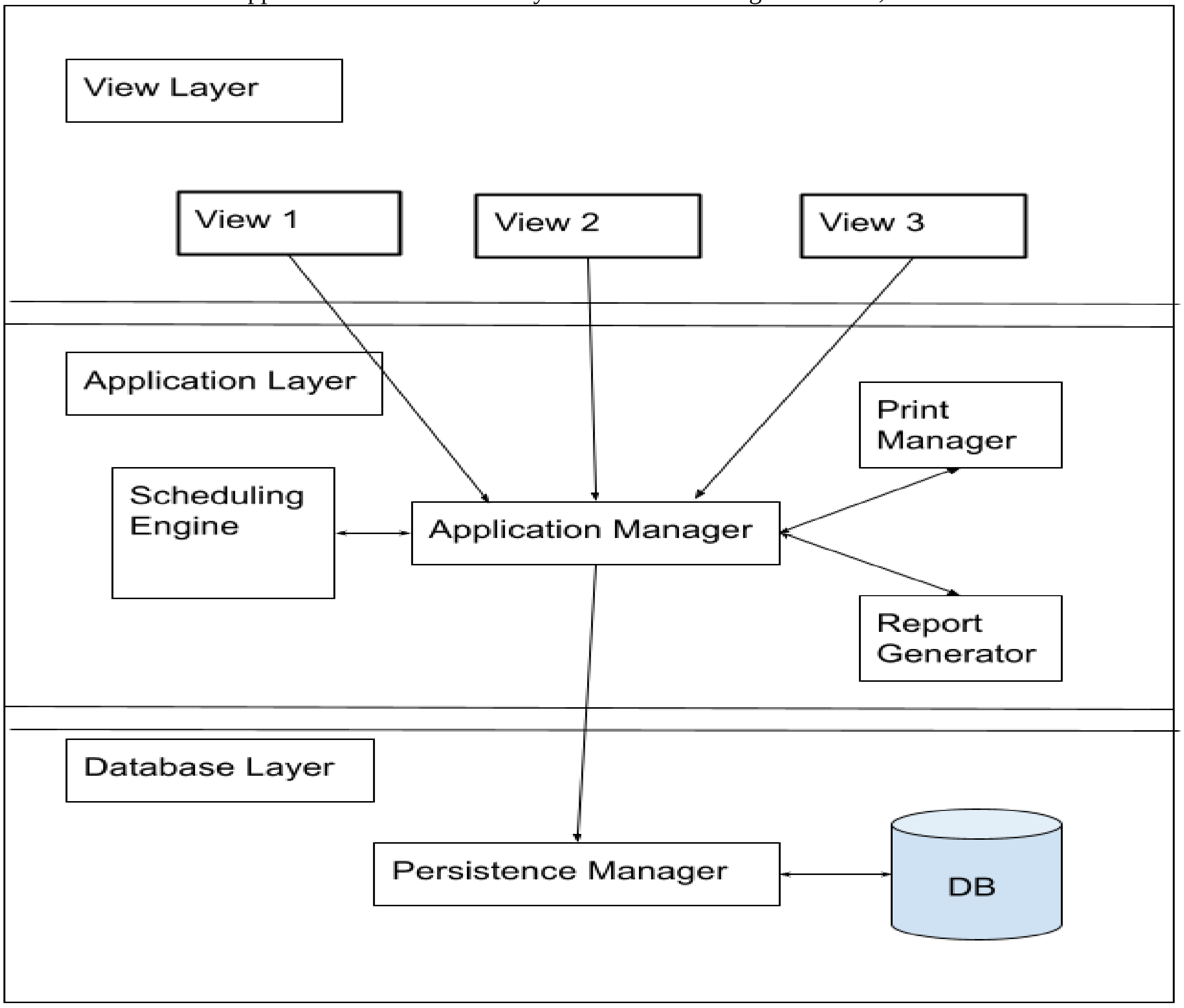


Figure 4. MVC Structure of the application Source (A. Lim, Juay Chin Ang. 2002)

IV. System Implementation and Testing

The Exam scheduling system is implemented using the Python programming language with the Flask framework for the backend and react.js for the user interface (UI). Flask is a backend development framework built with Python, it provides a lightweight and modular approach to software development, making it suitable for this project.

Overall, the system implements a variant of the knapsack optimization algorithm, focusing on maximizing the total value (importance, priority) of selected items within a given budget constraint, typical for scenarios where items have a cost and a value, and the goal is to maximize the total value of items selected without exceeding the budget. A breakdown of the program logic and functionalities of the system is presented below.

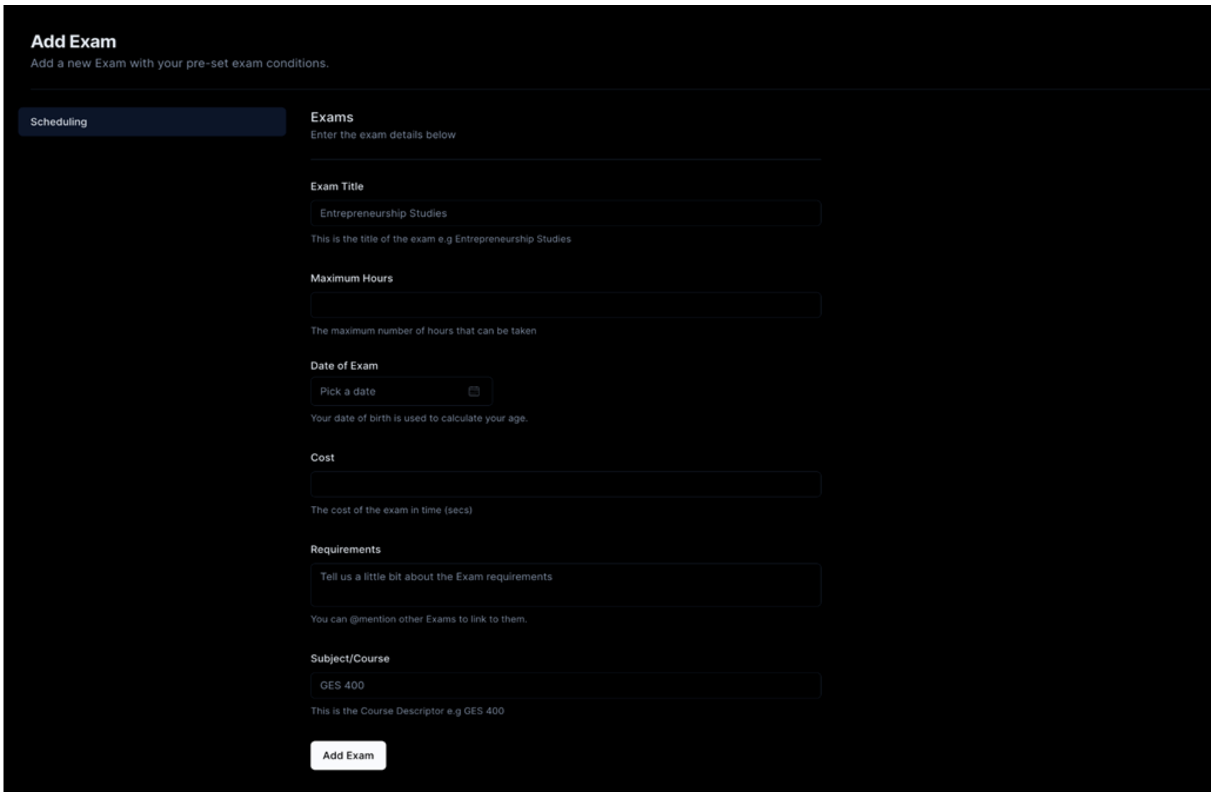
```
Input: K, exams
  K - Maximum total duration for scheduled exams
  exams - List of exam objects, each with attributes 'importance' (weight) and 'duration' (value)

Begin:
1. Let n be the number of exams in the list 'exams'
2. Create a 2D array 'schedule' of dimensions (n+1) x (K+1), initialized to 0

   // Initialize base cases
3. For i from 0 to n, do:
     schedule[i][0] = 0
4. For k from 0 to K, do:
     schedule[0][k] = 0
\
   // Build the schedule using a bottom-up dynamic programming approach
5. For i from 1 to n, do:
     For k from 1 to K, do:
        If exams[i-1].duration <= k Then:
           schedule[i][k] = max(schedule[i-1][k],
                              schedule[i-1][k - exams[i-1].duration] + exams[i-1].importance)
        Else:
           schedule[i][k] = schedule[i-1][k]

   // Backtrack to find the exams that were included in the optimal schedule
6. Initialize an empty list 'selectedExams'
7. Set i = n, k = K
8. While i > 0 and k > 0, do:
     If schedule[i][k] != schedule[i-1][k] Then:
        Add exams[i-1] to 'selectedExams'
        Decrease k by exams[i-1].duration
     EndIf
     Decrease i by 1

9. Return the list 'selectedExams'
End
```

Figure 5. Pseudocode of the Exam Scheduling System

This pseudocode in Figure 5details the process of dynamically scheduling exams within a given maximum total duration (K) by maximizing the total importance of the scheduled exams. It first builds a table to determine the maximum importance achievable at each step, considering each exam and duration. Then, it backtracks from the last cell of the table to find which exams were included in the optimal schedule.

The system has a functionality for the user management with a login module for users to authenticate with their credentials.The Exam scheduling based on the knapsack optimization algorithm allows you to create an exam with pre-set conditions which include the cost, the duration and course details. This will be used by the algorithm to calculate the optimal weighted output when scheduling exams.The proposed system assumes that the data items have a cost field and a limit field.

The user interface involved in creating an exam requires the operator to provide information related to each Exam, this is then evaluated using the knapsack optimization algorithm to determine the optimal exam schedule based on user availability and budget.The illustration below, shows the User Interface from which a processor can input Exam parameters which will update the exam records in the database (see figure 6).



Add Exam
Add a new Exam with your pre-set exam conditions.

Scheduling

Exams
Enter the exam details below

Exam Title
Entrepreneurship Studies
This is the title of the exam e.g Entrepreneurship Studies

Maximum Hours
The maximum number of hours that can be taken

Date of Exam
Pick a date
Your date of birth is used to calculate your age.

Cost
The cost of the exam in time (secs)

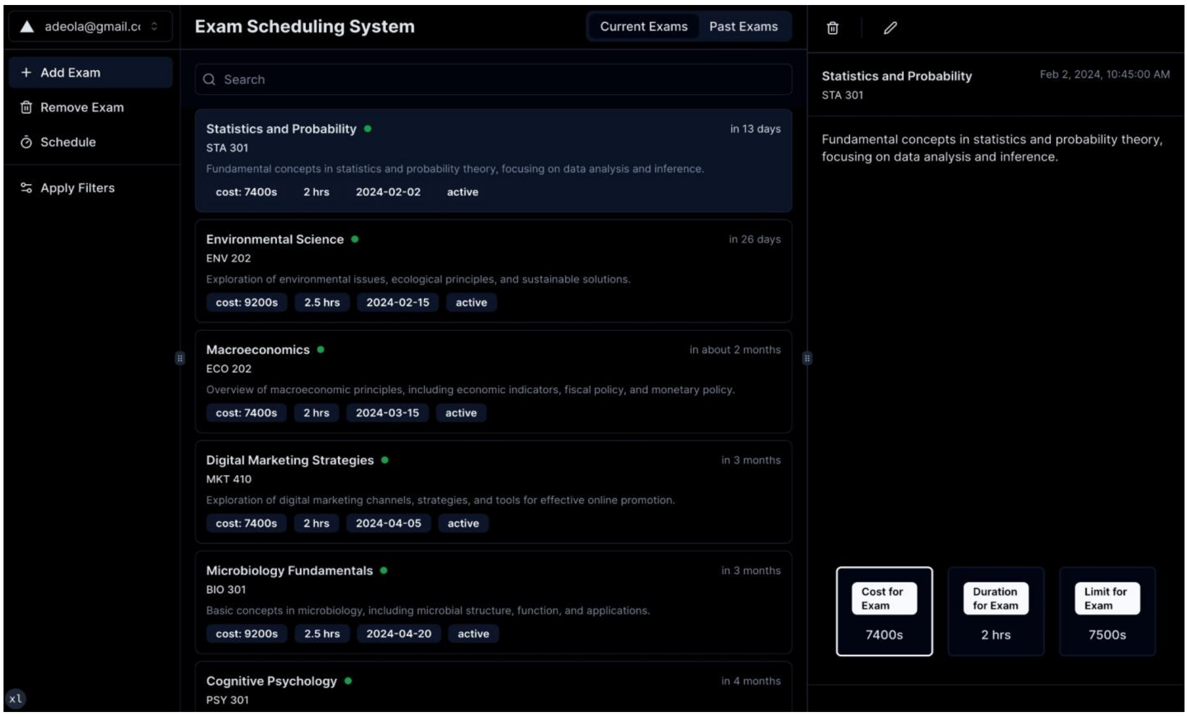
Requirements
Tell us a little bit about the Exam requirements
You can @mention other Exams to link to them.

Subject/Course
GES 400
This is the Course Descriptor e.g GES 400

Add Exam

Fig 6.Interface for Adding Exams to the database

The dashboard is a core function of the scheduling system. It enables you to easily add N Number of Exams represented as the datasets to the database. It also allows you to see the cost, making it easier to visualize the outcome of the optimization algorithm (see figure 7).
.



adeola@gmail.c

Exam Scheduling System | Current Exams | Past Exams

Add Exam
Remove Exam
Schedule
Apply Filters

Search

Statistics and Probability — in 13 days
STA 301
Fundamental concepts in statistics and probability theory, focusing on data analysis and inference.
cost: 7400s | 2 hrs | 2024-02-02 | active

Environmental Science — in 26 days
ENV 202
Exploration of environmental issues, ecological principles, and sustainable solutions.
cost: 9200s | 2.5 hrs | 2024-02-15 | active

Macroeconomics — in about 2 months
ECO 202
Overview of macroeconomic principles, including economic indicators, fiscal policy, and monetary policy.
cost: 7400s | 2 hrs | 2024-03-15 | active

Digital Marketing Strategies — in 3 months
MKT 410
Exploration of digital marketing channels, strategies, and tools for effective online promotion.
cost: 7400s | 2 hrs | 2024-04-05 | active

Microbiology Fundamentals — in 3 months
BIO 301
Basic concepts in microbiology, including microbial structure, function, and applications.
cost: 9200s | 2.5 hrs | 2024-04-20 | active

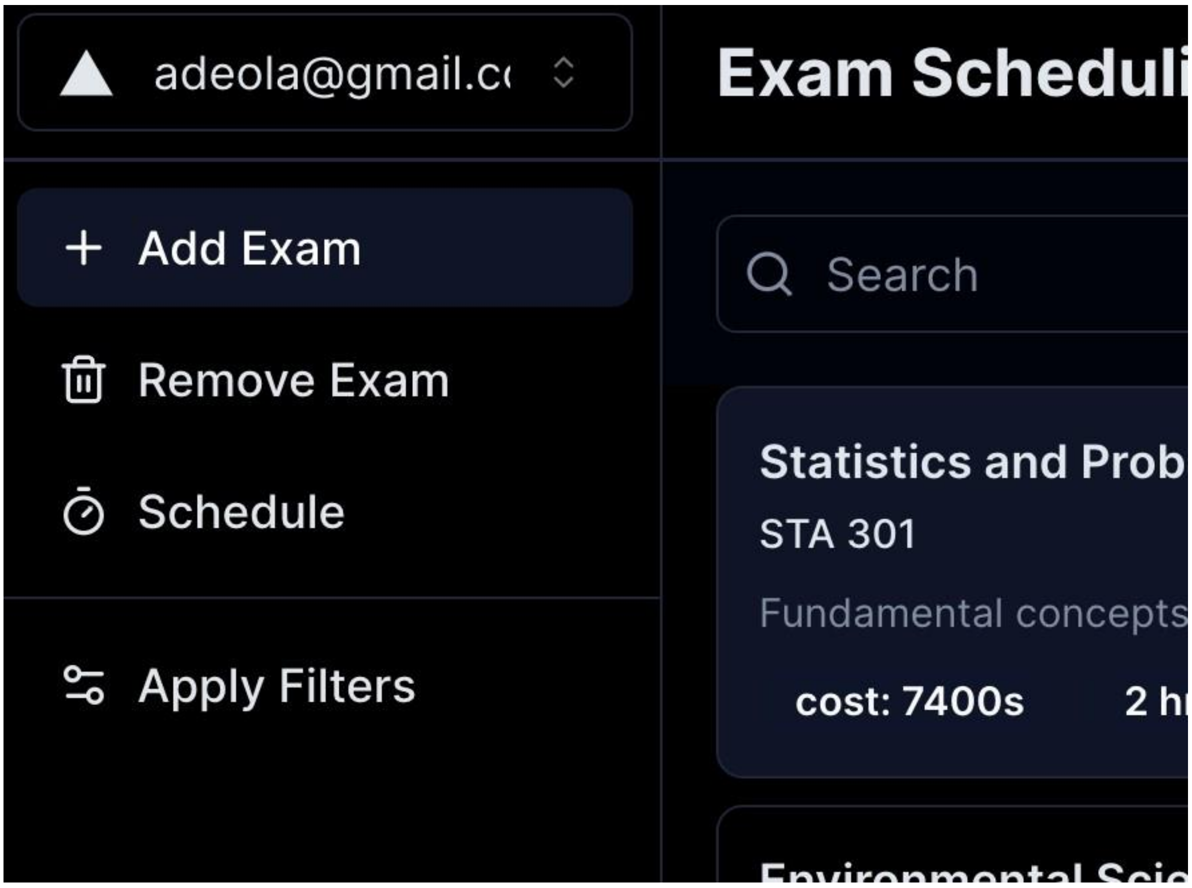
Cognitive Psychology — in 4 months
PSY 301

Statistics and Probability — Feb 2, 2024, 10:45:00 AM
STA 301
Fundamental concepts in statistics and probability theory, focusing on data analysis and inference.

Cost for Exam: 7400s
Duration for Exam: 2 hrs
Limit for Exam: 7500s

Fig 7. Dashboard Interface for the system

As an exam scheduler, you can add or remove Data Sets or update Existing Datasets, reflecting the new cost based on your expected output. However modifications to the dataset are limited to active exams as explained in chapter 2 of this study.As a student, you can use the scheduler to calculate the optimal number of Exams given your budget. As well as understand the total cost and savings generated from the optimization algorithm (see figure 8).

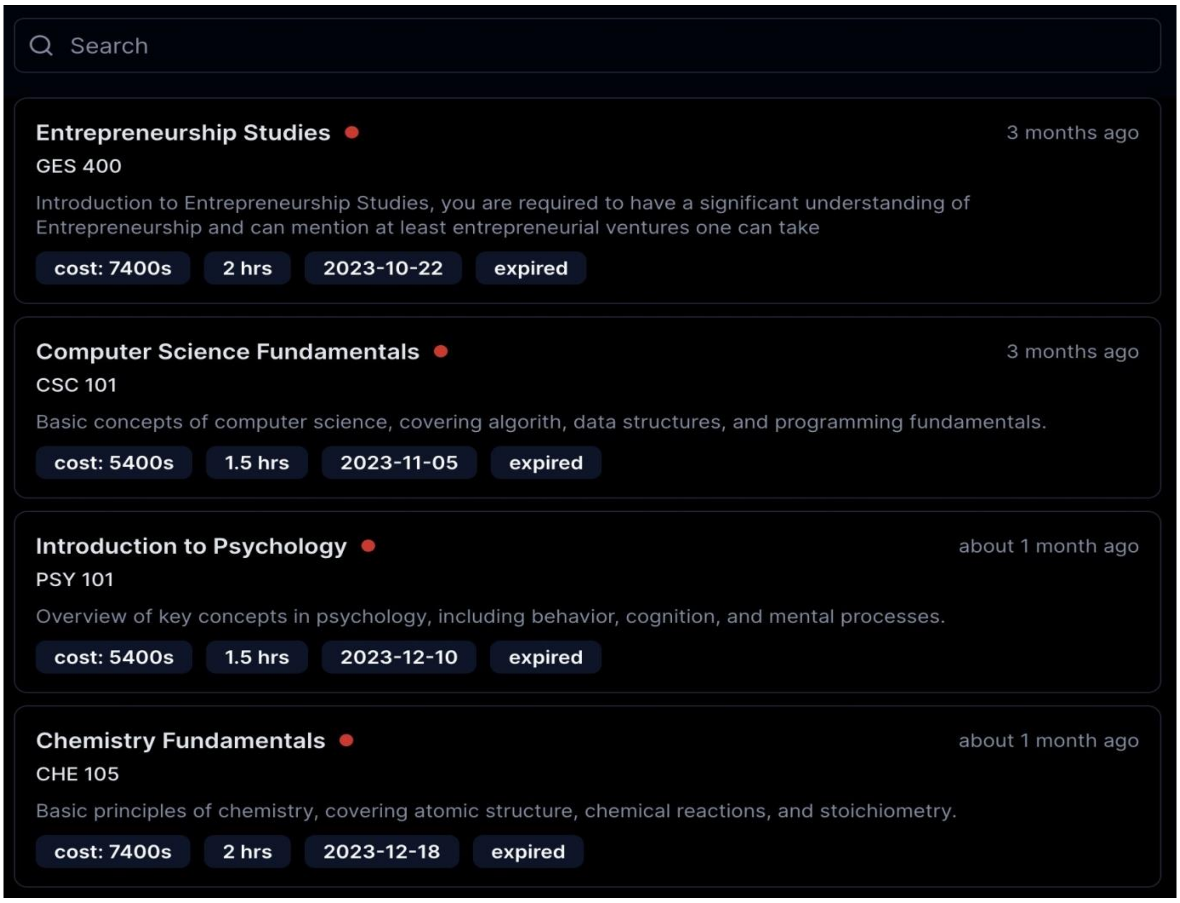


Statistics and Prob
STA 301
Fundamental concepts
cost: 7400s 2 h
Environmental Scie

Fig 8 Dashboard actions

The interface assumes there are distinctions between expired datasets, hence knapsacks optimization algorithm is only carried on the active Data items in the database and enables you to view Exams that are excluded from the calculation. By default Data Items (Exams) that are past their exam date are excluded from the knapsack optimization algorithm (see figure 9).

.


Search

Entrepreneurship Studies ● 3 months ago
GES 400
Introduction to Entrepreneurship Studies, you are required to have a significant understanding of Entrepreneurship and can mention at least entrepreneurial ventures one can take
cost: 7400s | 2 hrs | 2023-10-22 | expired

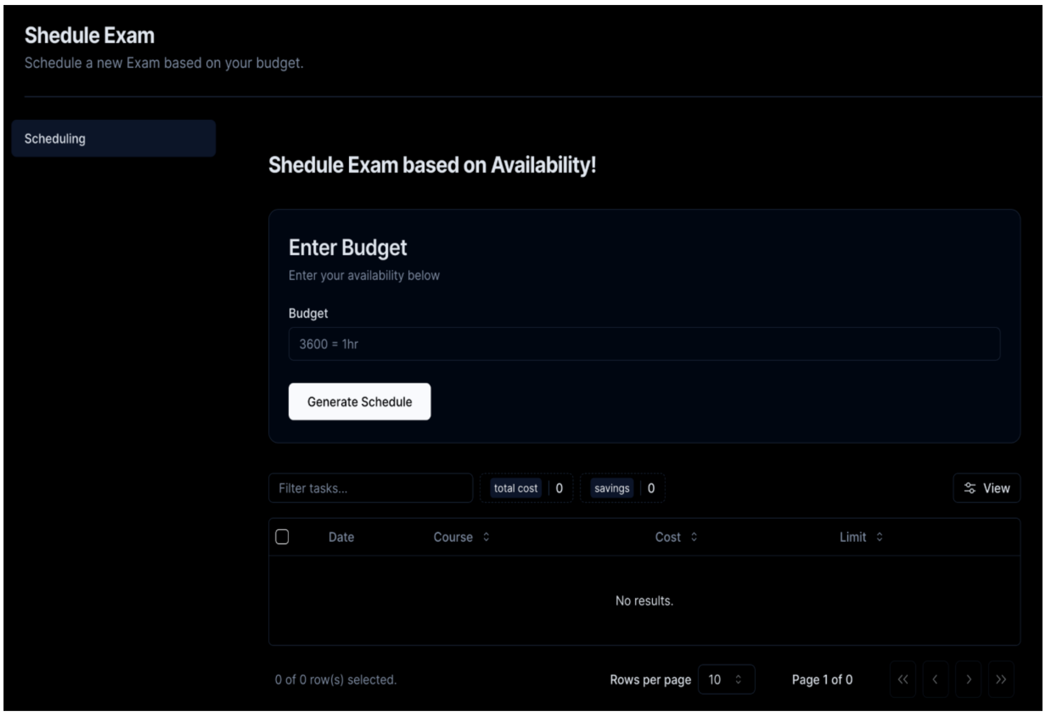
Computer Science Fundamentals ● 3 months ago
CSC 101
Basic concepts of computer science, covering algorith, data structures, and programming fundamentals.
cost: 5400s | 1.5 hrs | 2023-11-05 | expired

Introduction to Psychology ● about 1 month ago
PSY 101
Overview of key concepts in psychology, including behavior, cognition, and mental processes.
cost: 5400s | 1.5 hrs | 2023-12-10 | expired

Chemistry Fundamentals ● about 1 month ago
CHE 105
Basic principles of chemistry, covering atomic structure, chemical reactions, and stoichiometry.
cost: 7400s | 2 hrs | 2023-12-18 | expired

Fig 9.Sample datasets excluded from algorithm

Calculating optimal weight giving a budget Input:
Inputs to the scheduler is your budget. The system calculates the optimal number of exams for your budget, it also generates the total cost of the proposed schedule and the savings generated(see figure 10).
.


Shedule Exam
Schedule a new Exam based on your budget.

Scheduling

Shedule Exam based on Availability!

Enter Budget
Enter your availability below
Budget
3600 = 1hr
Generate Schedule

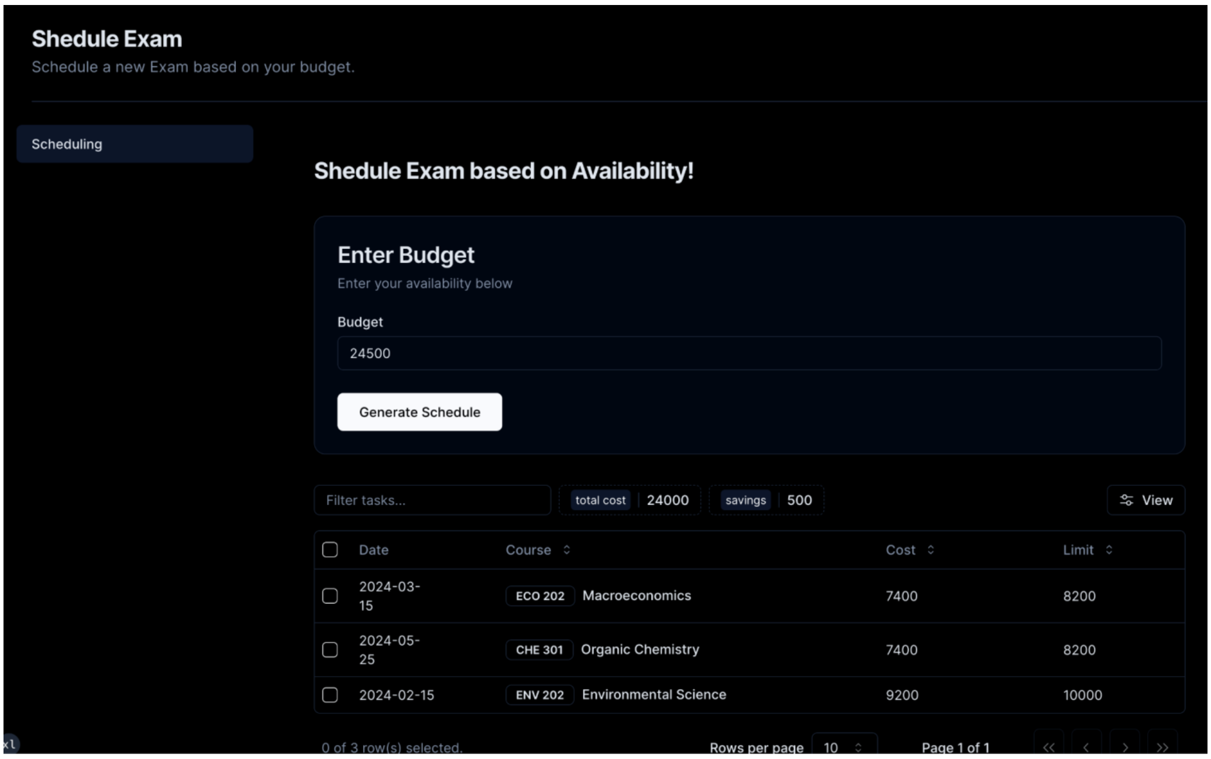
Filter tasks... | total cost 0 | savings 0 | View

Date	Course	Cost	Limit
No results.			

0 of 0 row(s) selected. Rows per page 10 Page 1 of 0

Fig 10. Input of the system

Output
Outputs of the system include the cost of taking N number of exams giving a specific budget in mind, the list of exams generated based on your budget and cost savings after optimization (see figure 11).


Shedule Exam

Schedule a new Exam based on your budget.

Scheduling

Shedule Exam based on Availability!

Enter Budget

Enter your availability below

Budget: 24500

Generate Schedule

Filter tasks... | total cost 24000 | savings 500 | View

Date	Course	Cost	Limit
2024-03-15	ECO 202 Macroeconomics	7400	8200
2024-05-25	CHE 301 Organic Chemistry	7400	8200
2024-02-15	ENV 202 Environmental Science	9200	10000

0 of 3 row(s) selected. Rows per page 10 Page 1 of 1

Fig 11. Output of an Optimal cost of Taking N number of Exams

A thorough testing was conducted to validate the functionality and performance of the exam scheduling system. Unit testing, integration testing, and system testing are carried out to identify and rectify any issues. The testing phase ensures that the system meets the specified requirements and functions reliably in various scenarios.Comprehensive documentation is generated, covering codebase, database structure, and the user manual. This documentation aids future developers, system administrators, and users in understanding, maintaining, and utilizing the Exam scheduling system.

V. Results and Discussion

This study proposed an exam scheduling system, specifically finding the most optimal cost for conducting an exam given a set of exams.

Dataset
A synthetic dataset was created for the exams scheduling problem, which is modelled as a knapsack problem. The dataset contains 10,000 records and is structured with the following fields:

- Exam Name: A unique identifier for each exam, labeled as "Exam_i" where i is a sequence number from 0 to 9999.
- Hours Required: The number of hours needed to complete each exam, ranging from 1 to 3 hours.
- Cost: The hypothetical cost associated with scheduling each exam, ranging from $100 to $999.

Using Dynamic Programming on the Dataset
To use dynamic programming (DP) to solve the exam scheduling problem modelled as a knapsack problem, the following steps were taken:
1. Define the Problem:

- The knapsack's capacity will be the total number of hours available for scheduling exams on a given day.
- Each exam has a "weight" (hours required) and a "value" (cost, which in this case might represent a penalty for not scheduling or the benefit of scheduling).

2. Dynamic Programming Approach:

- Initialization: Create a DP table where rows represent available exams and columns represent the number of hours available, initializing all entries to zero.
- Recursive Relation: For each exam and each possible hour capacity, decide whether to include the exam in the schedule based on whether it maximizes the value (minimizes cost, if cost represents a penalty) without exceeding the total hours. The choice is made by comparing the scenario of not taking the exam (value from the previous row) and the scenario of taking the exam (value from the previous row at "hours available minus hours of the current exam" plus the cost/value of the current exam).
- Build the Solution: Start from the last exam and the full capacity, trace back through the table to find which exams were included in the optimal schedule.
  3. Interpret Results:
- The solution gives the optimal set of exams to schedule to maximize the total value (or minimize total cost) without exceeding the available hours.

Given N exams where each exam has some weight and profit associated with it. And also given a bag with capacity W, (maximum number of hours permitted for exams daily). The objective is to put as many exams into the day such that the sum of profits associated with them is the maximum possible. The snapshot of the sample dataset used for the study is presented below (Figure 12).

Exam Name	Hours Required	Cost
Exam_0	3	693
Exam_1	1	471
Exam_2	3	716
Exam_3	3	699
Exam_4	1	613
Exam_5	1	971
Exam_6	3	183
Exam_7	2	142
Exam_8	3	765
Exam_9	3	938
Exam_10	3	705
Exam_11	3	909
Exam_12	1	232
Exam_13	3	494
Exam_14	2	164
Exam_15	1	389
Exam_16	2	475
Exam_17	2	868
Exam_18	2	566
Exam_19	2	997
Exam_20	1	312

Figure 12. Sample dataset

Input:
N = 50, W = 10
The weight and cost values are taken from the dataset.

Output:
The profit for dynamic programming method is:6206
Maximum value we can obtain = 6206
Selected items:

Item 2 (Weight: 1, Value: 471)
Item 5 (Weight: 1, Value: 613)
Item 6 (Weight: 1, Value: 971)
Item 20 (Weight: 2, Value: 997)
Item 36 (Weight: 2, Value: 964)
Item 41 (Weight: 1, Value: 814)
Item 45 (Weight: 1, Value: 560)
Item 48 (Weight: 1, Value: 816)

VI. Conclusion and Future Work

Computing exam schedules manually can be cumbersome while considering all factors involved, this study enables us to address exam scheduling as an optimization problem to maximize predefined profit as relevant to our use case. The exam scheduling system represents a significant achievement in addressing the complexities of managing examination schedules. The synergy between React and Flask has resulted in a well-rounded application that successfully achieves its intended goals.
It is recommended that the Exam Scheduling System can be improved to optimize multiple factors. The system contains a feature that allows users to receive notifications about upcoming exams.

References

[1]. Al-Betar, M. A., & Khader, A. T. (2009). "A Mathematical Approach to Continuous and Interactive Learning for Examination Timetabling." Applied Soft Computing.
[2]. BasuMallick, C. (2022, October 19). "A Simplified Guide to Dynamic Programming." Spiceworks. Retrieved January 11, 2024, from https://www.spiceworks.com/tech/devops/articles/what-is-dynamic-programming/
[3]. Brown, G., & Patterson, J. (2020). "Dynamic Programming in the Real World: Applications and Challenges." Journal of Complex Problem Solving, 12(1), 22-34.
[4]. Burke, E. K., & Newall, J. P. (2002). "A Multi-Stage Evolutionary Algorithm for the Timetable Problem." IEEE Transactions on Evolutionary Computation.
[5]. Buyya, R., Yeo, C. S., & Venugopal, S. (2009). "Cloud Computing and Emerging IT Platforms." Future Generation Computer Systems.
[6]. Carter, M., & Laporte, G. (2001). "Recent Developments in Practical Course Scheduling." Scheduling Theory and Practice.
[7]. Jain, S. (Various Dates). "Resource Allocation to Greedy Algorithms." GeeksforGeeks. Retrieved January 11, 2024, from https://www.geeksforgeeks.org/
[8]. Jayashree, P., & S, S. (2017). "A study report on solving 0–1 knapsack problem with imprecise data." Madras Institute of Technology, Anna University.
[9]. Jones, A., & Silver, L. (2018). "Resource Constraints in Academic Scheduling: A Comprehensive Review." Academic Scheduling Quarterly, 5(4), 45-60.
[10]. Kingston, J. H. (2004). "The Design of a Constraint-Based Automated Timetabling System." Computers & Education.
[11]. Kumar, P., & Singh, A. K. (2020). "AI Techniques in Practical Examination Timetabling." Journal of Intelligent Manufacturing.
[12]. Lee, J., & Kim, Y. (2019). "Heuristic Approaches to the Exam Scheduling Problem." Operations Research Perspectives, 6(3), 140-154.
[13]. Rouse, M. (2017, March 14). "What is Job Scheduling? - Definition from Techopedia." Techopedia. Retrieved January 11, 2024, from https://www.techopedia.com/definition/7882/job-scheduling
[14]. Ross, P., Hart, E., & Corne, D. (1998). "Some Observations about GA-Based Exam Timetabling." Journal of Scheduling.
[15]. Smith, B., & Foster, I. (1996). "Automated Timetabling in Large-Scale Administration." University of Michigan Press.
[16]. Smith, J., & Anderson, D. (2016). "Optimizing Resource Allocation for University Examinations." Journal of Educational Management, 30(2), 123-137.
[17]. S., F. (2023, July 2). "Organizing a sports event." Medium. Retrieved January 11, 2024, from https://medium.com/@faridsaminezhad/organizing-a-sports-event-9e055508bcb
[18]. Vázquez-Rodríguez, J. A. (2012). "The Multi-Criteria Decision-Making Process behind Timetabling." Operations Research Perspectives.
[19]. Williams, H. (2021). "Future Directions in Educational Resource Optimization." International Journal of Academic Development, 9(2), 188-202.
[20]. Zhang, Y., Qu, R., & Kendall, G. (2018). "A Survey on the Application of Genetic Algorithms to Small Problems in the Real World." Mendel Journal.

Appendix

Appendix A - Sample dataset

Exam Name	Hours Required	Cost
Exam_0	3	693
Exam_1	1	471
Exam_2	3	716
Exam_3	3	699
Exam_4	1	613

Exam_5	1	971
Exam_6	3	183
Exam_7	2	142
Exam_8	3	765
Exam_9	3	938
Exam_10	3	705
Exam_11	3	909
Exam_12	1	232
Exam_13	3	494
Exam_14	2	164
Exam_15	1	389
Exam_16	2	475
Exam_17	2	868
Exam_18	2	566
Exam_19	2	997
Exam_20	1	312
Exam_21	1	247
Exam_22	2	672
Exam_23	2	570
Exam_24	1	248
Exam_25	1	187
Exam_26	1	423
Exam_27	3	994
Exam_28	3	495
Exam_29	3	889
Exam_30	2	503
Exam_31	3	181
Exam_32	2	740
Exam_33	2	385
Exam_34	3	481
Exam_35	2	964
Exam_36	3	891
Exam_37	3	369
Exam_38	1	391
Exam_39	3	552
Exam_40	1	814
Exam_41	3	123
Exam_42	3	962
Exam_43	1	410
Exam_44	1	560
Exam_45	3	307
Exam_46	2	891
Exam_47	1	816
Exam_48	2	763

Exam_49	2	537
Exam_50	2	637

Appendix B – Sample source code for implementing the exams scheduling problem

```
import java.io.FileReader;
import java.io.BufferedReader;
import java.util.ArrayList;
import java.util.List;

public class Dynamicprogramming_selected {

    public static void main(String[] args) {
        String csvFile = "D:/PersonalDocuments/Uniport/BSc_project_2023/Examscheduler.csv";
        List<Integer> weights = new ArrayList<>();
        List<Integer> values = new ArrayList<>();

        try (BufferedReader br = new BufferedReader(new FileReader(csvFile))) {
            String line;
            br.readLine(); // to skip the header
            while ((line = br.readLine()) != null) {
                String[] data = line.split(",");
                weights.add(Integer.parseInt(data[1])); // Hours Required is the second column
                values.add(Integer.parseInt(data[2])); // Cost is the third column
            }
        } catch (Exception e) {
            e.printStackTrace();
        }

        int[] wt = weights.stream().mapToInt(i -> i).toArray();
        int[] val = values.stream().mapToInt(i -> i).toArray();
        int W = 10; // Define your knapsack capacity
        int N = weights.size();

        boolean[] selected = new boolean[N];
        int maxVal = knapSack(W, wt, val, N, selected);

        System.out.println("Maximum value we can obtain = " + maxVal);
        System.out.println("Selected items:");
        for (int i = 0; i < N; i++) {
            if (selected[i]) {
                System.out.println("Item " + (i+1) + " (Weight: " + wt[i] + ", Value: " + val[i] + ")");
            }
        }
    }

    public static int knapSack(int W, int[] wt, int[] val, int N, boolean[] selected) {
        int[][] dp = new int[N + 1][W + 1];

        for (int i = 0; i <= N; i++) {
            for (int w = 0; w <= W; w++) {
                if (i == 0 || w == 0) {
                    dp[i][w] = 0;
                } else if (wt[i-1] <= w) {
                    int include = val[i-1] + dp[i-1][w-wt[i-1]];
                    int exclude = dp[i-1][w];
                    dp[i][w] = Math.max(include, exclude);
                } else {
                    dp[i][w] = dp[i-1][w];
                }
            }
        }

        // Find selected items
        int res = dp[N][W];
        int w = W;
        for (int i = N; i > 0 && res > 0; i--) {
            if (res != dp[i-1][w]) {
                selected[i-1] = true;
                res -= val[i-1];
                w -= wt[i-1];
            }
        }
        return dp[N][W];
    }
}
```