Smart Investment Plan And Adaptive Study Plan Using Dynamic Programming

1st Mahmoud Assad
Faculty of Computer Science
Misr International University
mahmoud2310114@miuegypt.edu.eg

2nd Marly Hossam
Faculty of Computer Science
Misr International University
marly2301599@miuegypt.edu.eg

3rd Layla Tamer
Faculty of Computer Science
Misr International University
layla2304171@miuegypt.edu.eg

4th Ali Ahmed Salah
Faculty of Computer Science
Misr International University
ali2305282@miuegypt.edu.eg

5th Reem Ahmed Elsayyed
Faculty of Computer Science
Misr International University
reem2305827@miuegypt.edu.eg

***Abstract*—Dynamic Programming (DP) to tackle optimization problems by dividing them into smaller sets of issues and resolving each one just once. Two actual methods using dynamic programming are studied in this paper: advising people on how to invest and making study plans that adapt to individual situations. This model resolves the investment planning problem by solving an iterative, bottom-up DP problem that maximizes returns, given the constraints. Dynamic programming allows the Adaptive Study Plan to offer special learning routes for each student, adapting the approach according to their development and helping them remember more in a short timeframe. The two case studies describe how dynamic programming can be helpful for both financial and educational challenges. The efficiency, structure and scalability of the suggested DP models are evaluated, showing that using them offers better results than traditional or unoptimized approaches. Index Terms— Dynamic programming, smart investment planning, personal learning plans, optimization, step-by-step improvements and studying customized for an individual.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

Dynamic Programming paradigm uses complex problems by tackling them first in simple parts and storing what was computed to use again when necessary. It is very helpful when we need to decide things with limitations and aims to achieve the best outcome. We examine two real-life issues in this paper and show how dynamic programming applies to them.

This problem requires finding the safest way to distribute your money over a particular period to see the best returns. At the same time, the Adaptive Study Plan helps a learner organize their lessons and study time according to their success and how much time they have, to aid retention and performance.

Such problems display features that are well-suited to using dynamic programming: overlap among subproblems and the use of optimality in one part of the problem. This paper focuses on adapting dynamic programming iteratively and from the ground up for these domains and determines their similarities, differences and practical usefulness. The approaches are evaluated in terms of how well they perform, how flexible they are and how easily they can handle increased workloads.

II. Life Application

A. Smart Investment Plan

- Management of personal investments portfolio:- Everyone has to find strategies to make the most from the money they have in personal finance. When many investment options are available, each having a different cost and possible return, it's important for investors to choose where to put their money to get the highest return and stay within their budget.

B. Adaptive Study Plan

- Since some students must get through the lessons in a limited period, the plans for each lesson should be put in order. Helping too much information at once may seem busy and tiring for you. For good learning, students should distribute time for their lessons throughout the day. Additionally, being good at managing your time Provides students a chance to review all subjects every day. Consider ways to increase your productivity and observe what takes place as that final push gives little to no benefit.

III. Smart Investment Plan

A. Algorithm Explanation

The Smart Investment Planning algorithm works to ensure you have the best profit over given times, while observing cooldown periods. The algorithm is structured into two main components:

1) Problem Analysis: The main goal is to identify the ideal pattern so that more profit is made each day, but the time between daily play is considered.

- **Input:**
  - `profit[]`: An array representing the profit from investing on each day.

- `cooldown[]`: An array indicating the number of cooldown days after each investment.
- `n`: The total number of days.

- **Output:**
  - The maximum profit achievable under the given constraints.
- **Objective:**
  - Find the best schedule for making your investments by looking at both the potential income and the cooldown every day.

2) Algorithm Design: The algorithm constructs a `dp[]` array where `dp[i]` shows the greatest amount of profit that can be earned from the beginning of day `i` to the end.

Decision Formula:

$$dp[i] = \max(dp[i+1], \texttt{profit}[i] + dp[i + \texttt{cooldown}[i] + 1]) \quad (1)$$

B. Implementation

The Smart Investment Planning algorithm is developed in three separate steps called Initialization, Main Function and Dynamic Programming Computation. Every part is followed by its own code and extensive description.

1) Part 1: Initialization: At this stage, the data structures are created and the dynamic programming array is set up `dp[]`.

Algorithm 1 Initialization of DP Array

Require: Array of profits `profit[]`, Array of cooldown days `cooldown[]`, Number of days `n`

```
Initialize dp[] as a zero array of size n + 1
dp[n] = 0 {No further investment opportunities after the last day}
```

Explanation: The `dp[]` array is initialized to zero with a size of `n + 1`. It will store the best daily profits made from the beginning of the interval. `dp[n]` is set to zero because there are no more investments available after the end of the process.

2) Part 2: Main Function: The main function defines inputs the arrays and then uses dynamic programming to get the highest profit it can find.

Algorithm 2 Main Function - Input Definition and Execution

Require: Arrays `profit[]` and `cooldown[]`

```
Define profit[] = {3, 9, 7, 8, 6, 5, 10}
Define cooldown[] = {3, 1, 2, 2, 1, 1, 0}
n = length of profit[]
Print "Maximum Profit: ", smartInvestmentWithCooldown(profit, cooldown)
```

Explanation: Initially, the main function assigns profits and cooldown times to each day. The code then identifies the best investment strategy with the help of `smartInvestmentWithCooldown()` function. The function processes every day, using dynamic programming to find the greatest profit achieved.

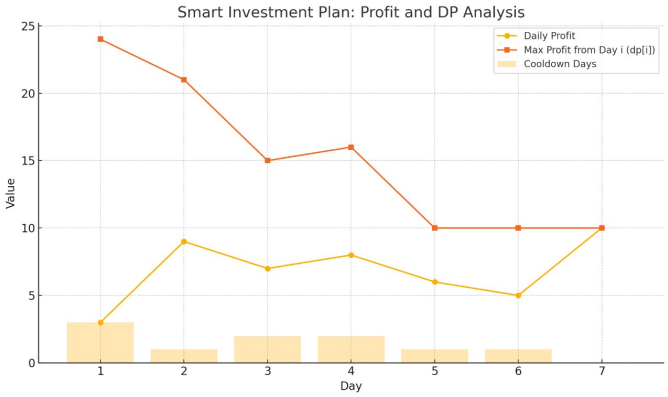

Fig. 1. Smart Investment Plan

3) Part 3: Dynamic Programming Computation: The core computation logic is implemented in the `smartInvestmentWithCooldown()` function. It iterates from the last day backward to calculate the optimal investment strategy.

Algorithm 3 Dynamic Programming Computation

```
for i = n − 1 to 0 do
    skip = dp[i + 1]
    take = profit[i]
    if i + cooldown[i] + 1 < n then
        take += dp[i + cooldown[i] + 1]
    end if
    dp[i] = max(skip, take)
end for
return dp[0]
```

Explanation: - The algorithm iterates backward from the last day to the first day. - At each day `i`, two options are considered: - Skip the Investment: The profit remains the same as the next day (`dp[i + 1]`). - Invest and Rest: The profit from investing today is calculated by adding today's profit and the profit after the cooldown period (`dp[i + cooldown[i] + 1]`).

The optimal choice is the maximum of these two options, stored in `dp[i]`. The final answer is stored in `dp[0]`, representing the maximum profit obtainable starting from day 0.

IV. Adaptive Study Plan

A. Algorithm Explanation

Dynamic methods are used by the Adaptive Study Plan Optimizer, a programming approach to help you learn and remember more with each lesson, performing well when you don't have much time and when your mind is tired. There are only two major areas in the algorithm.

1) Problem Analysis: The main goal of this stage is to find out how to divide time equally to study each subject so you can get as much learning gain as possible, decreasing the problems that come with burnout.

- **Input:**
  - `base[]`: Base XP gain per hour for each subject.
  - `burnout[]`: Fatigue factor causing diminishing returns.
  - `H`: Total study hours available.
  - `n`: Number of subjects.
- **Output:**
  - The maximum XP achievable and the corresponding optimal allocation of study hours per subject.
- **Objective:**
  - Optimize the allocation of a fixed number of study hours across multiple subjects to maximize total learning gains, while accounting for cognitive fatigue through a burnout-aware model.

2) Algorithm Design: The core of the Adaptive Study Plan Optimizer is a dynamic programming approach designed to explore all possible allocations of a finite number of study hours across multiple subjects. The algorithm constructs a two-dimensional DP table `dp[i][h]`, where each entry represents the optimal state considering the first $i+1$ subjects and a total of h hours distributed among them.

- `i` $\in [0, n-1]$ denotes the current subject index.
- `h` $\in [0, H]$ represents the cumulative number of hours allocated up to that point.
- Each entry `dp[i][h]` contains:
  - The maximum XP value achievable with h total hours allocated among the first $i+1$ subjects.
  - A corresponding allocation vector of size n, indicating how many hours were assigned to each subject to reach this XP.

The dynamic programming transition works as follows: for each subject i, and each possible total time h, we iterate through all possible time splits $t \in [0, h]$ that could be allocated to subject i. We then evaluate if the total XP gained from allocating t hours to subject i, combined with the best XP from allocating $h-t$ hours to the first i subjects, results in a higher cumulative XP than the current stored maximum. If so, the DP entry is updated accordingly.

a) XP Function with Burnout: To accurately reflect human cognitive limits, we define an XP function that considers burnout, which models diminishing learning returns as study hours increase. For each subject i, the XP gained from studying t hours is computed using a quadratic utility function that subtracts a burnout penalty:

$$XP_i(t) = \max(0,\ \texttt{base}_i \cdot t\ -\ \texttt{burnout}_i \cdot t^2) \quad (2)$$

Where:

- $\texttt{base}_i$: XP gain per hour for subject i, representing raw learning efficiency.
- $\texttt{burnout}_i$: XP penalty factor that increases quadratically with time, modeling fatigue or diminishing returns.
- The outer `max` function ensures XP never falls below zero, avoiding penalization for any over-allocation.

This function is precomputed for every possible hour allocation $t \in [0, H]$ for all subjects $i \in [0, n-1]$ to allow for efficient lookups during the DP transitions.

b) Dynamic Programming Transition: The DP table is filled as follows:

- For the first subject, all feasible hour allocations from 0 to H are initialized using the XP function.
- For subsequent subjects, the algorithm evaluates all combinations of current subject hours `t` such that `t` $\leq$ `h`, and updates `dp[i][h]` according to the recurrence:

$$dp[i][h] = \max(dp[i][h], dp[i-1][h-t].\text{XP} + XP_i(t)) \quad (3)$$

- If a better XP is found, the allocation vector is updated accordingly.

c) Final Output: After the table is filled, `dp[n-1][H]` holds:

- The maximum XP attainable with `H` total hours distributed across all subjects.
- The specific number of hours allocated to each subject to achieve that XP.

B. Implementation

The implementation of the Adaptive Study Plan algorithm is divided into three parts: Initialization, Main Function, and Dynamic Programming Computation. Each part is structured to align with the Smart Investment Planning style.

1) Part 1: Initialization: This step sets up the XP model and constructs XP tables for each subject across possible time allocations.

Algorithm 1 Initialization of XP Tables

Require: Base XP/hour `base[]`, Burnout rate `burnout[]`, Total study hours `H`, Number of subjects `n`

```
Define function computeXP(base, burnout, t) as: base * t - burnout * t^2
Initialize xp[n][H+1] as a 2D array of XP values
for i = 0 to n - 1 do
    for t = 0 to H do
        xp[i][t] = computeXP(base[i], burnout[i], t)
    end for
end for
```

Explanation: Each subject has a different learning curve. XP gained diminishes with more time spent due to burnout. The XP table models this for all hour allocations from 0 to `H`.

2) Part 2: Main Function: Defines input parameters and executes the dynamic programming function to get optimal hour allocation.

3) Part 2: Main Function: Defines input parameters and executes the dynamic programming function to get optimal hour allocation.

Algorithm 2 Main Function - Input and Execution

1: `n` and H entered by user.
2: `base[]` = entered by user, `burnout[]` = enetered by user
3: Compute XP table using Initialization
4: Compute `dp[n][H]` using dynamic programming.
5: Find the best way to allocate hours to each subject for maximum XP.
6: Print Maximum XP and hour distribution.

Explanation: This function initializes subject learning rates and applies the dynamic programming solution to find the best way to distribute the limited available hours among subjects.

4) Part 3: Dynamic Programming Computation: Fills the DP table to find the optimal time allocation per subject.

Algorithm 3 Dynamic Programming for Adaptive Study Plan

```
Initialize dp[n][H+1] to 0
for h = 0 to H do
    dp[0][h] = xp[0][h]
end for
for i = 1 to n − 1 do
    for h = 0 to H do
        dp[i][h] = 0
        for t = 0 to h do
            dp[i][h] = max(dp[i][h], dp[i-1][h - t] + xp[i][t])
        end for
    end for
end for
maxXP = dp[n - 1][H]
```

Explanation: For each subject, and for each available hour, we determine the best XP we can gain. The algorithm picks the hour distribution that maximizes total XP while accounting for burnout.

5) Output Example:

```
Maximum XP: 21.80
Hour allocation per subject:
Subject 1: 3 hours
Subject 2: 2 hours
```

V. Time Complexity Analysis and Comparison

Let us dive into a detailed breakdown of the time and memory requirements of the two Dynamic Programming algorithms developed in this work: **Smart Investment Planning** and **Adaptive Study Plan Optimizer**. The goal is to determine which of the two is more efficient in terms of computational complexity.

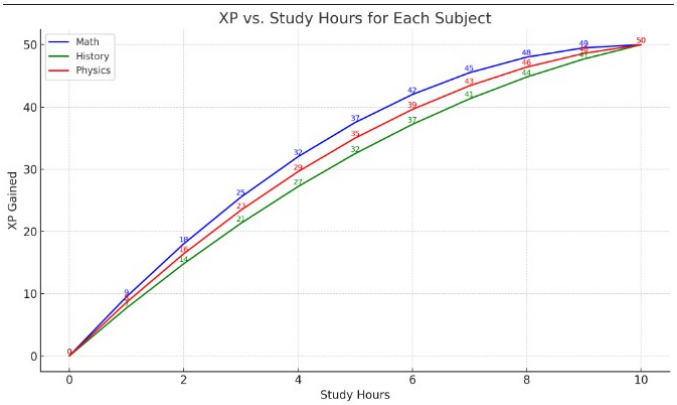

Fig. 2. XP vs. Study Hours for adaptive learning

A. Smart Investment Planning

This algorithm focuses on computing the maximum profit that can be obtained from a series of daily returns, taking into account cooldown periods. After each investment, a waiting period is required before the next investment can be made. The algorithm leverages bottom-up dynamic programming to eliminate redundant calculations and ensure optimal decision-making.

Let n denote the number of days. A one-dimensional dynamic programming array of size $n+1$ is used, where each entry $dp[i]$ stores the maximum profit that can be obtained starting from day i. The array is populated in reverse, from day $n-1$ to day 0, with the following choices at each step:

- **Skip the day:** Take the value of $dp[i+1]$.
- **Invest on the day:** Add today's profit to $dp[i + \text{cooldown}[i] + 1]$, if within bounds.

This results in a linear scan through the input with only constant-time operations per day, without any nested loops or recursion overhead.

- **Time Complexity:** $\mathcal{O}(n)$ — each day is processed once.
- **Space Complexity:** $\mathcal{O}(n)$ — to store the DP values.

Although a hash map is present to group days by cooldown, it is not utilized during the DP computation and does not affect the complexity class.

B. Adaptive Study Plan Optimizer

This algorithm addresses a classic resource allocation problem—how to distribute a limited number of study hours (e.g., 16 hours per day) across multiple subjects in order to maximize total XP. It incorporates diminishing returns through a burnout rate, making the optimization more realistic.

First, it precomputes XP for every subject-hour combination into a 2D array $xp[i][t]$, where i represents the subject index and t the hours allocated. Then, it builds a DP table $dp[i][h]$, where each entry contains the maximum XP achievable and the hour allocation across the first $i+1$ subjects for a total of h hours.

The computation involves a three-layered loop:

- Iterate over each subject: $\mathcal{O}(n)$

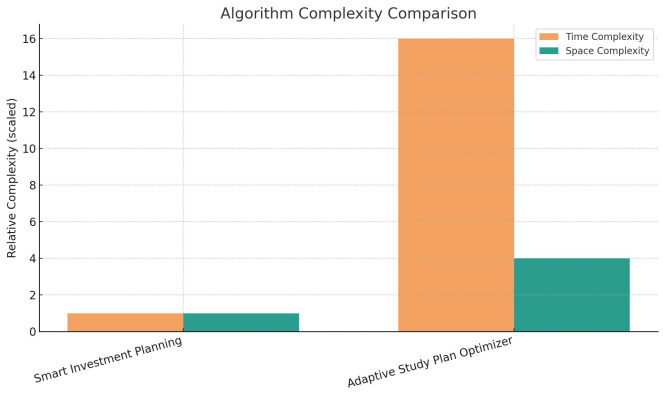

Fig. 3. Algorithm complexity comparison

- Iterate over each possible hour total: $\mathcal{O}(H)$
- Iterate over each hour partition: $\mathcal{O}(H)$

Thus, the overall complexity is:

- **Time Complexity:** $\mathcal{O}(n \times H^2)$ — due to nested loops over the hour range.
- **Space Complexity:** $\mathcal{O}(n \times H)$ — to store XP and DP tables.

Despite its heavier computational demands, the algorithm remains practical for realistic values of H due to the limited daily study time.

C. Comparison Summary

Algorithm	Time Complexity	Space Complexity
Smart Investment Planning	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Adaptive Study Plan Optimizer	$\mathcal{O}(n \times H^2)$	$\mathcal{O}(n \times H)$

TABLE I
COMPLEXITY COMPARISON BETWEEN THE TWO ALGORITHMS

From a computational standpoint, the **Smart Investment Planning** algorithm stands out for its simplicity and efficiency. It processes input in a single pass with minimal overhead, making it suitable for larger datasets.

In contrast, the **Adaptive Study Plan Optimizer** trades computational efficiency for a more nuanced and flexible optimization, necessary for handling multi-dimensional constraints. While it requires more resources, the algorithm provides tailored solutions that account for realistic burnout effects, making it more suited to complex decision-making scenarios.

In conclusion, algorithm selection depends heavily on the problem's structure. What may seem like overkill in one application could be essential in another. Efficiency is not just about speed—it's also about adaptability and precision.

REFERENCES

Dynamic programming is used in the field of financial planning to help boost investment decisions as time passes [1].

Approaches from Dynamic Programming efficiently address the classical issue of figuring out the top buying and selling points in stock trading [**?**].

Lately, scientists in cognitive science have learned about how people decide and how they divide up their resources which supports our understanding of adaptive planning [2].

Software like CNOWv2 from Cengage provides students with learning programs that adapt well to algorithm optimization [3].

REFERENCES

[1] Z. Liu, "Dynamic Programming Algorithms and Their Application to Financial Management," *Journal of Electrical Systems*, vol. 20, no. 9s, 2024.

[2] T. Zhang and M. R. Delgado, "An adaptive approach to understanding investment decision-making," *Behavior Research Methods*, vol. 56, pp. 132–145, 2023. [Online]. Available: https://link.springer.com/article/10.3758/s13428-023-02191-5

[3] GeeksforGeeks, "Stock Buy and Sell to Maximize Profit," [Online]. Available: https://www.geeksforgeeks.org/stock-buy-sell/, Accessed: May 29, 2025.

[4] Cengage, "Create an Adaptive Study Plan - CNOWv2," [Online]. Available: https://help.cengage.com/cnowv2/instructor/cnow-owl/create-an-adaptive-study-plan-cnow.html, Accessed: May 29, 2025.