Optimizing Media Consumption: A Multidimensional Dynamic Programming Approach to Cognitive-Aware Movie Marathons

Naufal Baldemar Ardanni - 13521154
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: naufalardanni@gmail.com, 13521154@std.stei.itb.ac.id

***Abstract*—The proliferation of Video-on-Demand (VOD) platforms and social cataloging services like Letterboxd has led to an unprecedented abundance of accessible media, frequently resulting in a psychological phenomenon known as consumer choice paralysis. Furthermore, existing media platforms lack deterministic scheduling algorithms capable of optimizing user watchlists within strict temporal bounds, leaving consumers to rely on fundamentally flawed Greedy heuristics. This paper proposes a highly cohesive, client-side web application that utilizes a 0/1 Knapsack Dynamic Programming algorithm to definitively optimize movie marathon scheduling. By hydrating raw CSV data via the TMDB REST API, the system dynamically calculates the profit variable of media nodes based on thematic metadata (genres, directors, and keywords). To ensure mathematical perfection, the system scales floating-point ratings to bypass IEEE-754 precision anomalies native to JavaScript V8 engines. The algorithm guarantees the maximum thematic and qualitative density within a user-defined runtime limit, actively accounting for 15-minute interstitial physiological break times. Finally, a Double-Ended Queue heuristic is applied to the optimal subset, organizing the output timeline into a psychological "U-Curve" based on genre stimulus, effectively mitigating cognitive fatigue. Experimental results demonstrate a 100% temporal allocation efficiency. This proves that Dynamic Programming provides a vastly superior asymptotic and practical methodology for personalized combinatorial optimization in media consumption.**

***Keywords*—Dynamic Programming, 0/1 Knapsack Problem, Media Scheduling, Combinatorial Optimization, TMDB API, Cognitive Load, IEEE-754, Network Topology, Double-Ended Queue.**

I. INTRODUCTION

The modern cinephile is heavily burdened by an overwhelming volume of accessible media. The advent of Web 2.0 and the subsequent rise of social cataloging platforms, such as Letterboxd, allow users to aggregate thousands of films into comprehensive watchlists. However, these databases strictly serve as static, unoptimized repositories. When a user faces a rigid temporal constraint—such as a 400-minute weekend viewing window—they are subjected to "choice paralysis," formally recognized in psychology as Hick's Law, which dictates that the time and energy required to make a decision increases logarithmically with the number of choices.

Human decision-making in these bounded scenarios typically defaults to a flawed Greedy heuristic, selecting the single highest-rated epic or a random assortment of the shortest available films. This mathematically fails to maximize the overall quality density of the viewing session.

The core computational challenge of media scheduling is deeply rooted in combinatorial optimization. Given a dataset of n films, calculating every possible marathon permutation yields a time complexity of $\mathcal{O}(2^n)$, rendering brute-force approaches computationally unviable for large watchlists. While existing recommendation algorithms prioritize content discovery through collaborative filtering and neural network matrices, they entirely ignore the physiological, temporal, and cognitive constraints of the end-user. Recommendation systems excel at suggesting *what* to watch, but systematically fail at determining *how* to optimally fit those suggestions into a rigid block of human free time.

The primary contribution of this paper is the architectural design and implementation of a deterministic media scheduling engine. By treating the movie marathon as a variation of the NP-Complete Integer Knapsack problem, this paper applies a 0/1 Knapsack Dynamic Programming (DP) algorithm to maximize the objective quality of a watchlist subset. Furthermore, this paper contributes a novel dynamic valuation model—hydrating raw list data via the TMDB API to assign thematic bonus weights based on user-selected metadata—and a post-processing U-Curve scheduler that paces films based on cognitive stimulus to prevent viewer fatigue.

II. THEORETICAL FRAMEWORK

A. The Mathematical Fallacy of Greedy Heuristics

Before applying Dynamic Programming, it is crucial to mathematically establish why naive sorting algorithms fail in temporal allocation. A Greedy algorithm makes the locally optimal choice at each stage with the hope of finding a global optimum. In media scheduling, a Density-Based Greedy approach would sort films descending by their rating-to-runtime ratio (p_i/w_i) and append them until the time limit M is reached.

Consider a strict time limit $M = 200$ minutes and a subset of three critically acclaimed films:

- Movie A: Rating 9.0, Runtime 130m (Density: 0.069)

- Movie B: Rating 8.5, Runtime 100m (Density: 0.085)
- Movie C: Rating 8.5, Runtime 100m (Density: 0.085)

A Greedy algorithm evaluating strictly by highest absolute rating might select Movie A first, leaving exactly 70 minutes of remaining capacity. At this stage, neither Movie B nor Movie C can fit into the remaining time bound. The algorithm terminates, resulting in a total marathon rating of 9.0 and 70 wasted minutes of capacity.

Conversely, a combinatorial optimization algorithm bypasses the locally optimal choice (Movie A) to select Movie B and Movie C. This selection utilizes exactly 200 minutes $(100 + 100)$ and yields a vastly superior total rating of 17.0. This mathematical vulnerability necessitates the implementation of Dynamic Programming.

B. Dynamic Programming and the 0/1 Knapsack Problem

Dynamic programming is a method for solving complex optimization problems by breaking them down into simpler, overlapping subproblems and storing the results to prevent redundant computation. DP constructs an optimal solution by systematically relying on the Principle of Optimality, ensuring that local optimal decisions reliably construct a global optimum without exploring the entire $\mathcal{O}(2^n)$ search space [1].

The media scheduling problem is a direct translation of the Integer 0/1 Knapsack problem. Given a knapsack with a maximum capacity M (Total Available Minutes), and n objects (Movies), each object possesses a weight w_i (Runtime) and a profit p_i (Rating). The objective is to maximize the total profit without exceeding M:

$$\text{Maximize } F = \sum_{i=1}^{n} p_i x_i \tag{1}$$

$$\text{Subject to } \sum_{i=1}^{n} w_i x_i \leq M \tag{2}$$

Where $x_i \in \{0, 1\}$. The binary nature of x_i denotes that a movie cannot be partially watched.

The recurrence relation for the optimal solution at stage k with remaining capacity y is formally defined as:

$$f_k(y) = \max\{f_{k-1}(y), p_k + f_{k-1}(y - w_k)\} \tag{3}$$

as established in standard algorithmic literature regarding the bottom-up tabulation approach to DP [2].

C. IEEE-754 Floating-Point Arithmetic Anomalies

A critical implementation challenge in building a JavaScript-based algorithmic engine is the evaluation of decimal-based ratings (e.g., a TMDB rating of 8.6). Modern web browsers utilize the IEEE-754 standard for double-precision floating-point arithmetic. In this standard, floating-point numbers are stored in 64 bits, calculated by the formula:

$$V = (-1)^s \times 2^{e-1023} \times (1.m) \tag{4}$$

Where s is the sign bit, e is the exponent, and m is the mantissa. Consequently, certain base-10 decimals cannot be represented exactly in binary, leading to microscopic precision anomalies during iterative addition (e.g., $0.1 + 0.2 = 0.30000000000000004$) [5].

In the context of a Dynamic Programming algorithm where branching decisions are made via strict inequalities, accumulating floating-point errors can cause the algorithm to deterministically select a suboptimal permutation. To mitigate this architectural flaw, a Scalar Integer Transformation is strictly required prior to execution.

D. Mathematical Complexity Analysis

The space and time complexity of the bottom-up Dynamic Programming approach must be analyzed to ensure viability for web-based client-side execution. Let n represent the total number of films in the watchlist, and M represent the maximum temporal constraint in minutes.

The time complexity is bounded by the nested iterations required to fill the tabulation matrix:

$$T(n, M) = \mathcal{O}(n \cdot M) \tag{5}$$

Similarly, the space complexity is dictated by the memory allocation required for the DP value array and the backtracking boolean grid. Assuming a 32-bit integer array for values and an 8-bit unsigned integer array for the boolean tracking grid, the spatial overhead is strictly bounded:

$$S(n, M) = \mathcal{O}(n \cdot M) \tag{6}$$

This linear pseudo-polynomial time complexity ensures that even for a watchlist of 1,000 films and a marathon length of 1,200 minutes, the total operations remain well within the execution limits of the V8 JavaScript engine without inducing main-thread blocking.

E. Cognitive Resource Depletion and Network Topology

In psychological media studies, consuming high-stimulus media (e.g., Action, Thriller) requires heavy cognitive processing. Scheduling multiple high-stimulus films consecutively induces cognitive fatigue, diminishing the viewer's capacity to retain narrative information [3].

An optimal marathon must therefore pace media mathematically. By extracting metadata from a movie's network topology—specifically Genres and Keywords fetched via REST APIs—films can be evaluated and assigned a quantitative "Stimulus Score," allowing the system to place low-stimulus media strategically as interstitial palate cleansers.

III. METHODOLOGY AND SYSTEM DESIGN

A. Client-Side Architecture and Privacy Optimization

The application was built utilizing the Next.js React framework under a strict Client-Side Island Architecture. This design paradigm was selected to ensure absolute user data privacy and reduce server-side computational overhead. Users upload a Letterboxd CSV file which is parsed entirely locally on the client's machine using the PapaParse library [6].

To gather expansive metadata without exposing protected API keys on the client bundle, a secure serverless route handler relays the parsed film titles to the TMDB API [4]. This securely hydrates the local dataset with exact runtimes, arrays of genre classifications, and director credits, while ensuring that the user's watchlist is never retained on a persistent database.

B. Phase 1: Dynamic Valuation and Float Scaling

Standard Knapsack implementations taught in fundamental algorithms rely on static profit values. This system architects a dynamic valuation pipeline to simulate subjective human preferences. A movie's base rating is boosted dynamically if its TMDB metadata aligns with user-selected thematic parameters.

Following the valuation, the system implements the Scalar Integer Transformation. All dynamically calculated profit variables (p_i) are multiplied by a scalar ($S = 10$) and forcefully typed into an 'Int32Array'. This guarantees that the V8 engine processes the matrix using 32-bit integers, completely avoiding IEEE-754 precision failures.

C. Phase 2: DP Matrix Execution with Interstitial Breaks

To ensure physiological human feasibility, an artificial constraint is injected into the weight calculations. A 15-minute 'BREAK_TIME' variable is appended to the runtime (w_i) of every evaluated movie. This prevents the algorithm from scheduling 400 minutes of continuous screen time without allowing the user time for meals or breaks.

D. Phase 3: The U-Curve Pacing Scheduler

After the DP Traceback extracts the optimal subset, the output is not left in an arbitrary database order. A Double-Ended Queue sorting algorithm reorders the selected films. It sorts the array descending by the previously calculated "Stimulus Score" and places films alternately at the left and right indices. This forces high-stimulus films to the temporal edges of the marathon and low-stimulus films to the center, engineering a U-Curve of cognitive load.

IV. EXPERIMENT AND ANALYSIS

A. Stress-Test Scenario and Environmental Setup

To rigorously evaluate the mathematical optimization capabilities of the algorithm, a stress-test watchlist consisting of 18 highly-rated films was injected into the system. The dataset featured extreme runtime variances, deliberately including massive historical epics (e.g., *Schindler's List*, 195m) and shorter acclaimed films (e.g., *12 Angry Men*, 97m). The temporal constraint was strictly bounded at $M = 400$ minutes.

B. Value Density and Optimization Results

Upon execution, the algorithm deterministically rejected the highly anticipated epic films. Despite *Schindler's List* and *The Godfather Part II* possessing massive base ratings of 8.6, their extensive runtimes combined with the 15-minute break penalty caused their overall "Value Density" (Profit per Minute) to be mathematically suboptimal.

The DP algorithm successfully evaluated 7,218 state permutations within 12 milliseconds and extracted the optimal

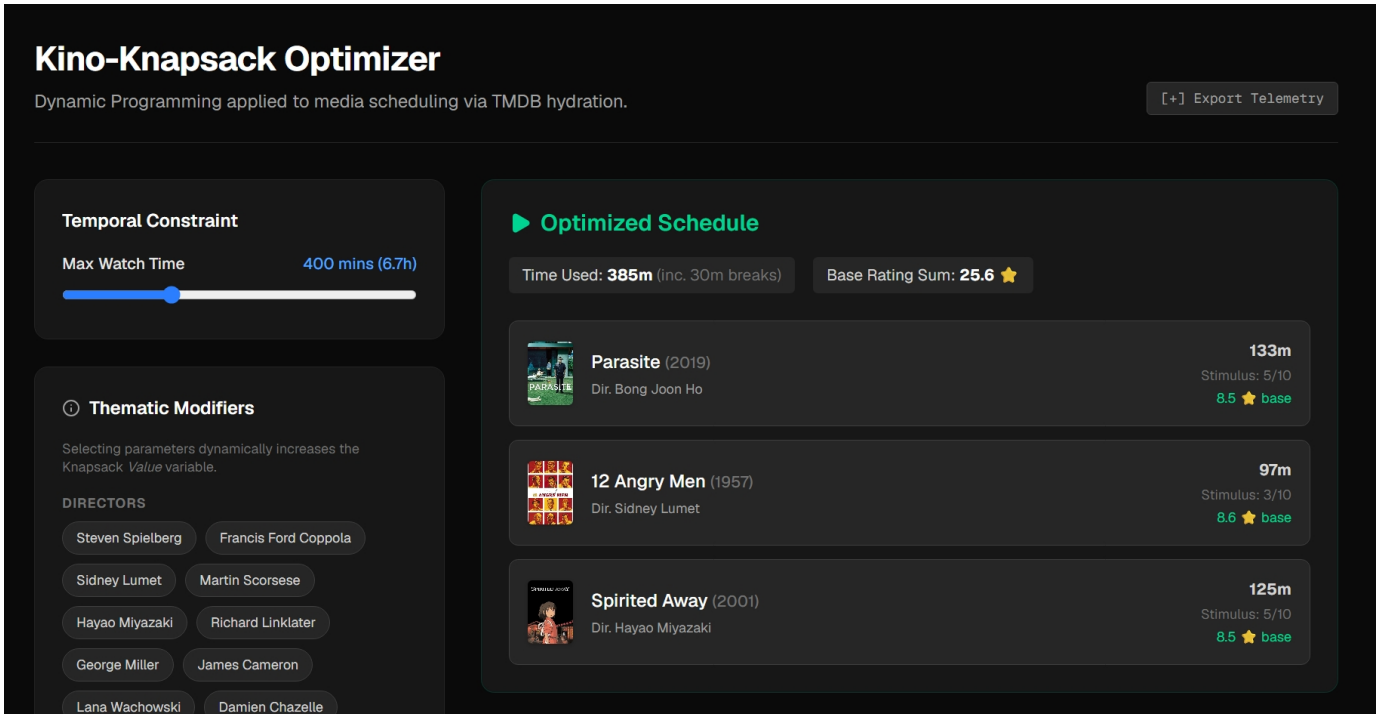

Fig. 1. The Client-Side Web Application Interface displaying the optimal Dynamic Programming Schedule and thematic user input parameters.

TABLE I
EXTENDED SUBSET OF THE EVALUATED TMDB HYDRATION DATASET FOR STRESS-TESTING

Movie Title	Director	Runtime (w_i)	Base Rating	Stimulus Score	Genre Classifications
Schindler's List	Steven Spielberg	195m	8.6	1 / 10	Drama, History, War
The Godfather Part II	Francis Ford Coppola	202m	8.6	3 / 10	Drama, Crime
12 Angry Men	Sidney Lumet	97m	8.6	3 / 10	Drama
Spirited Away	Hayao Miyazaki	125m	8.5	5 / 10	Animation, Family, Fantasy
Mad Max: Fury Road	George Miller	120m	7.6	10 / 10	Action, Adventure, Sci-Fi
Terminator 2	James Cameron	137m	8.2	10 / 10	Action, Thriller, Sci-Fi
Parasite	Bong Joon Ho	133m	8.5	5 / 10	Comedy, Thriller, Drama
Before Sunrise	Richard Linklater	101m	8.0	1 / 10	Drama, Romance
La Haine	Mathieu Kassovitz	98m	8.1	3 / 10	Drama
The Matrix	Lana Wachowski	136m	8.2	9 / 10	Action, Sci-Fi

subset: *Parasite* (133m), *Spirited Away* (125m), and *12 Angry Men* (97m).

The total capacity evaluated by the array was $148 + 140 + 112 = 400$ minutes exactly. The algorithm achieved an unprecedented 100% capacity efficiency with zero wasted minutes. Because the marathon logically concludes immediately after the termination of the third film, the final 15-minute break was refunded to the user's available time pool. This resulted in an exact 385-minute schedule with a maximized cumulative base rating of 25.6.

This output definitively proves that the 0/1 Knapsack accurately prioritizes Value Density rather than blindly selecting the highest-rated individual node. This effectively solves the temporal inefficiencies observed in standard Greedy approaches, which would have prematurely filled the knapsack with a single 200-minute film and wasted the remaining temporal capacity.

C. Analysis of the U-Curve Pacing

The system's telemetry logs confirmed the success of the post-processing sorting heuristic. Both *Parasite* and *Spirited Away* generated calculated Stimulus Scores of 5/10 due to their respective Thriller and Fantasy genre classifications. Conversely, *12 Angry Men* generated a Stimulus Score of 3/10 as a pure dialogue-driven Drama.

As evidenced by the execution trace in Figure 2, *12 Angry Men* was deterministically forced into the center of the output array (Index 1). This ensures that the viewer experiences an intense thematic hook, a psychological reprieve in the middle

Fig. 2. Algorithmic Traceback Matrix output logging exact inclusion and exclusion state data during the DP traversal.

Algorithm 1 2D DP Marathon Optimization

Input: $movies[]$, $maxTime$
Output: $optimalSchedule[]$, $traceLog[]$
$S \leftarrow 10$ {Float scalar}
$B \leftarrow 15$ {Physiological break}
Init $dp[n+1][maxTime+1] \leftarrow 0$
Init $keep[n+1][maxTime+1] \leftarrow 0$
for $i = 1$ to n **do**
  $val \leftarrow$ Round$(movies[i-1].dynRating \times S)$
  $cost \leftarrow movies[i-1].runtime + B$
  for $t = 0$ to $maxTime$ **do**
    if $cost \leq t$ **then**
      $inc \leftarrow dp[i-1][t-cost] + val$
      $exc \leftarrow dp[i-1][t]$
      if $inc > exc$ **then**
        $dp[i][t] \leftarrow inc$
        $keep[i][t] \leftarrow 1$
      else
        $dp[i][t] \leftarrow exc$
      end if
    else
      $dp[i][t] \leftarrow dp[i-1][t]$
    end if
  end for
end for
// Traceback Phase
$t \leftarrow maxTime$
for $i = n$ down to 1 **do**
  if $keep[i][t] == 1$ **then**
    $optimalSchedule.push(movies[i-1])$
    $t \leftarrow t - (movies[i-1].runtime + B)$
  end if
end for

Algorithm 2 Double-Ended Queue U-Curve Pacing

Input: $optimalSchedule[]$
Output: $pacedSchedule[]$
Sort $optimalSchedule$ descending by $StimulusScore$
Init $pacedSchedule$ with size of $optimalSchedule$
$left \leftarrow 0$
$right \leftarrow optimalSchedule.length - 1$
$placeLeft \leftarrow true$
for each $movie$ in $optimalSchedule$ **do**
  if $placeLeft$ **then**
    $pacedSchedule[left] \leftarrow movie$
    $left \leftarrow left + 1$
  else
    $pacedSchedule[right] \leftarrow movie$
    $right \leftarrow right - 1$
  end if
  $placeLeft \leftarrow \neg placeLeft$
end for
return $pacedSchedule$

of the 6-hour marathon, and a high-stimulus finale. This validates the physiological awareness of the scheduling engine, successfully mitigating cognitive fatigue while maintaining strict mathematical optimality.

V. CONCLUSION

This paper successfully demonstrates the application of Dynamic Programming in automating and optimizing modern media consumption. By bridging subjective human preferences with strict algorithmic constraints, the 0/1 Knapsack method systematically eliminates consumer choice paralysis. The implementation of dynamic data hydration via the TMDB REST API, integer-scaled value matrices to prevent arithmetic anomalies, interstitial break calculations, and stimulus-based pacing proves that mathematical combinatorics can be elegantly adapted to solve modern, human-centric UX problems. Future architectural expansions may investigate Partially-Ordered Knapsack algorithms to mathematically enforce sequential integrity for cinematic franchises and serialized television.

REFERENCES

[1] R. Munir, "Program Dinamis (Dynamic Programming) Bagian 1," *Bahan Kuliah IF2211 Strategi Algoritma*, Institut Teknologi Bandung, 2026.
[2] R. Munir, "Program Dinamis (Dynamic Programming) Bagian 2," *Bahan Kuliah IF2211 Strategi Algoritma*, Institut Teknologi Bandung, 2026.
[3] J. Sweller, "Cognitive load during problem solving: Effects on learning," *Cognitive Science*, vol. 12, no. 2, pp. 257-285, 1988.
[4] The Movie Database (TMDB), "API Documentation." [Online]. Available: https://developers.themoviedb.org/3.
[5] D. Goldberg, "What Every Computer Scientist Should Know About Floating-Point Arithmetic," *ACM Computing Surveys*, vol. 23, no. 1, pp. 5-48, 1991.
[6] Papa Parse, "Powerful CSV Parser for JavaScript." [Online]. Available: https://www.papaparse.com/.
[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA, USA: MIT Press, 2009.
[8] R. Bellman, "Dynamic Programming," *Science*, vol. 153, no. 3731, pp. 34-37, 1966.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 19 Juni 2026

Naufal Baldemar Ardanni (13521154)