Maximizing Profit in Charizard Pokémon Card PSA Grading: A Comparative Analysis of 0/1 Knapsack DP and Greedy Heuristics.

Optimalisasi Keuntungan pada Grading PSA Kartu Pokémon Charizard melalui Analisis Perbandingan Algoritma Dynamic Programming 0/1 Knapsack dan Heuristik Greedy

Dzakwan Muhammad Khairan Putra Purnama - 13524145
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: dzakwan.mkpp@gmail.com, 13524145@std.stei.itb.ac.id

Abstract* — The trading card market, particularly for Pokémon, has witnessed exponential growth, transforming professional card grading into a strategic investment vehicle. Professional Sports Authenticator (PSA) grading significantly enhances a card's market value; however, this value appreciation is contingent upon upfront grading costs. This study formulates the selection of Charizard cards for PSA grading as a classic 0/1 Knapsack Problem, aiming to maximize total net profit under restricted budgetary constraints. We implement and compare two algorithmic approaches: Dynamic Programming (DP), which guarantees a globally optimal solution, and a Greedy Algorithm based on the profit-to-cost ratio (density). Experiments conducted on a dataset of 75 unique Charizard cards across three budget scenarios ($100, $250, and $500) reveal that the Greedy Algorithm achieves the identical maximum profit as the DP approach in all cases, while demonstrating superior computational efficiency. These findings indicate that for this specific dataset, the profit-to-cost distribution inherently aligns with the greedy choice property, rendering the Greedy approach a highly efficient and effectively optimal alternative for grading submission optimization.

***Keywords* — 0/1 Knapsack problem, dynamic programming, greedy algorithm, optimization, Pokémon trading cards, PSA grading**

I. Introduction

The Pokémon Trading Card Game (TCG) has transcended its origins as a tabletop game to emerge as a significant alternative investment asset class. Within this market, Charizard cards consistently command the highest premiums, driven by extreme collector demand and perceived scarcity [6]. A pivotal determinant of card valuation is the card's physical condition, officially authenticated and graded by third-party institutions such as Professional Sports Authenticator (PSA) [7]. Achieving a high grade (e.g., PSA 9 or 10) can exponentially inflate a card's market value relative to its "raw" or ungraded state, significantly impacting the potential return on investment (ROI) [7].

However, the submission process for grading entails substantial financial overhead. PSA imposes non-trivial fees for each submission, scaled according to the declared value and service tier [7]. For investors managing extensive collections with limited capital, the decision-making process of selecting which specific cards to submit to maximize the aggregate profit presents a complex combinatorial optimization challenge [5]. This resource allocation problem can be mathematically modeled as a **0/1 Knapsack Problem**, where the "knapsack capacity" denotes the investor's total capital budget for grading fees, the "weight" of each item represents the grading cost, and the "value" corresponds to the estimated net profit [5].

From a theoretical standpoint, the 0/1 Knapsack Problem is a classic example of an ***NP-Complete*** problem [3]. According to computational complexity theory, this classification implies that no known algorithm can solve every instance of the problem in polynomial time [3], [4]. To address this challenge, researchers typically employ two distinct algorithmic strategies:

- *Dynamic Programming (DP):* An approach that ensures mathematical optimality by exploring the state space using memoization breaking the problem into stages and states to circumvent redundant computations [2].
- *Greedy Algorithm:* A strategy that makes locally optimal choices at each stage (sorting items based on their profit-to-cost density) with the hope of reaching a global optimum efficiently [1].

The primary objective of this study is to evaluate the trade-offs between optimality, accuracy, and computational efficiency across three budget constraints ($100, $250, and $500). By comparing the systematic DP approach [2] with the locally optimal Greedy strategy [1], this research provides a data-driven methodology for investors to optimize their grading portfolios effectively [6], [7].

The main contributions of this paper are summarized as follows:

1. Formulating the novel application of the 0/1 Knapsack Problem to optimize Pokémon TCG PSA grading submissions.
2. Implementing and empirically comparing the exact Dynamic Programming approach against a heuristic Greedy density-based algorithm.
3. Demonstrating that for this specific economic dataset, the Greedy algorithm elegantly achieves the global optimum with exponential time-efficiency (0.00005s execution time), providing a highly scalable model for real-world investors.

II. Literature Review

A. The 0/1 Knapsack Optimization Problem

The Knapsack problem is a fundamental and extensively studied problem in the field of combinatorial optimization. It serves as a generalized model for various resource allocation scenarios where a decision-maker must select a subset of items from a given set to maximize total value without exceeding a strict capacity limit [5]. In the specific variation known as the 0/1 Knapsack problem, fractional items are not permitted; an item must either be selected entirely or rejected completely. Due to this discrete binary constraint, the problem is classified under the *NP-Complete* complexity class [3], meaning no polynomial-time algorithm is known to exist to solve all instances optimally [4].

Mathematically, the problem can be modeled as an objective function that seeks to maximize the total value. Let N represent the total number of available items. Each item i (where $1 \leq i \leq N$) is associated with a specific weight (w_i) and a specific value or profit (v_i). Let W be the maximum weight capacity of the knapsack. The decision variable x_i is a binary variable that equals 1 if the item i is included in the knapsack, and 0 otherwise. The objective function to maximize the total profit is defined by Eq. 1, subject to the capacity constraint defined by Eq. 2 [5].

$$Maximize \sum_{i=1}^{N} (x) v_i x_i$$

(Equation 1)

$$\sum_{i=1}^{N} w_i x_i \leq W, x_i \in \{0, 1\}$$

(Equation 2)

B. Dynamic Programming Algorithm

Dynamic Programming (DP) is a systematic algorithmic paradigm designed to solve complex optimization problems by breaking them down into simpler, overlapping subproblems [2]. Rather than recomputing the solutions to these subproblems, DP stores their results in a memory structure, significantly reducing computational redundancy. For the 0/1 Knapsack problem, DP provides a method to guarantee the discovery of the absolute global optimum [2].

The process begins by defining a state function $f_k(x)$, which represents the maximum possible value that can be attained using a subset of the first k items with a maximum available capacity of x. The optimal solution is iteratively computed using the state transition formula shown in Eq. 3 [2].

$$f_k(x) = max\{v_k + f_{k-1}(x - c_k), f_{k-1}(x)\}$$

(Equation 3)

In this equation, v_k represents the value (profit) of the k-th item, and c_k represents its cost (weight). If the cost of the current item c_k is less than or equal to the current capacity x, the algorithm evaluates two scenarios: including the item (adding its value v_k to the optimal value of the remaining capacity x - c_k) or excluding it (retaining the optimal value $f_{k-1}(x)$). The algorithm selects the maximum of these two choices. If $c_k > x$, the item cannot be included, and $f_k(x)$ simply inherits the value of $f_{k-1}(x)$. This bottom-up tabulation approach ensures exact optimality but operates with a pseudo-polynomial time and space complexity of $O(N \times W)$ [5], which can become highly memory-intensive when the maximum capacity W is extraordinarily large.

C. Greedy Algorithm

The Greedy algorithm is an alternative heuristic approach to combinatorial optimization that makes the locally optimal choice at each discrete stage with the intent of reaching a global optimum [1]. Unlike DP, which evaluates all possible states, the Greedy approach commits to decisions sequentially without reconsidering them. In the context of the Knapsack problem, the standard Greedy algorithm operates by evaluating the efficiency of each item, represented by its profit-to-weight density [1].

Step-by-step procedures for the Knapsack Greedy Algorithm are as follows:

1. Calculate the profit-to-weight ratio ($r_i = v_i / w_i$) for each item i in the dataset.
2. Sort all items in descending order based on their computed r_i values.
3. Initialize the total profit and total accumulated weight to zero.
4. Iterate through the sorted list, sequentially adding items to the knapsack if the addition of the item's weight does not cause the accumulated weight to exceed W.
5. Terminate the process when no more items can fit within the remaining capacity.

While this ratio-based Greedy algorithm guarantees a mathematically optimal solution for the Fractional Knapsack problem, it generally does not guarantee a global optimum for the 0/1 Knapsack constraint [1]. This limitation arises because high-ratio items might leave small, unusable fractions of

remaining capacity that could have been better utilized by a combination of items with slightly lower ratios. However, the primary advantage of the Greedy approach is its computational efficiency. The performance bottleneck is the initial sorting mechanism, resulting in a significantly faster time complexity of *O(N log N)* [1].

D. Pokémon TCG Economics and PSA Grading

The secondary market for the Pokémon Trading Card Game (TCG) has rapidly transitioned from a niche hobby into a highly capitalized alternative asset class [6]. Specific character cards, most notably Charizard, consistently dominate market demand and command immense financial premiums. A fundamental component of modern TCG economics is third-party authentication and grading, primarily dominated by Professional Sports Authenticator (PSA) [7].

PSA evaluates the physical condition of a raw card based on centering, corners, edges, and surface quality, ultimately encapsulating the card and assigning it a grade on a precise 1-to-10 scale. A "PSA 10" (Gem Mint) designation can amplify a card's market value exponentially compared to its ungraded state. However, the grading process incurs a non-negligible cost. PSA utilizes a tiered fee structure where the grading fee scales with the card's declared market value and the desired processing speed [7].

For an investor possessing a portfolio of raw Charizard cards, optimizing grading submissions is a strict resource-constrained decision that fundamentally mirrors the 0/1 Knapsack problem. The parameter mapping for this economic scenario is outlined in Table I. By applying computational algorithms to this framework, investors can establish an objective methodology to maximize return on investment (ROI) under strict capital limitations.

TABLE I. KNAPSACK PARAMETER MAPPING FOR PSA GRADING.

Knapsack Parameter	Math Variable	Grading Context Equivalent
Capacity	W	Total available capital for grading fees (Budget)
Weight	w_i	PSA grading fee for a specific card
Value	v_i	Projected net profit (Graded Value - Raw Value - Fee)

III. METHODOLOGY

The proposed combinatorial optimization framework involves a sequence of data extraction and algorithmic evaluation steps designed to maximize the financial return of grading Pokémon trading cards using Dynamic Programming and Greedy algorithms, as shown in Figure 1. The process begins with reading the empirical dataset (**`dataset_charizard_75_final.csv`**) and defining the desired maximum capital budget constraints (W). The raw tabular data is first parsed into independent arrays, separating the grading costs and projected net profits to simplify the data structure and reduce computational overhead. Once the parameter arrays are established, the economic scenario is strictly modeled mathematically as a 0/1 Knapsack problem. This step transforms the financial metrics into discrete weights and values, preparing the dataset for rigorous algorithmic processing. Next, both the Dynamic Programming and Greedy algorithms are applied to the formulated data. The DP approach systematically evaluates all state transitions to guarantee mathematical optimality, while the Greedy heuristic sorts the cards by profit-to-cost density to execute a highly computationally efficient selection. Finally, the execution results are recorded across multiple budget scenarios, capturing the maximum profit, total budget utilized, explicitly selected card subsets, and precise execution times to facilitate a comprehensive empirical comparison.

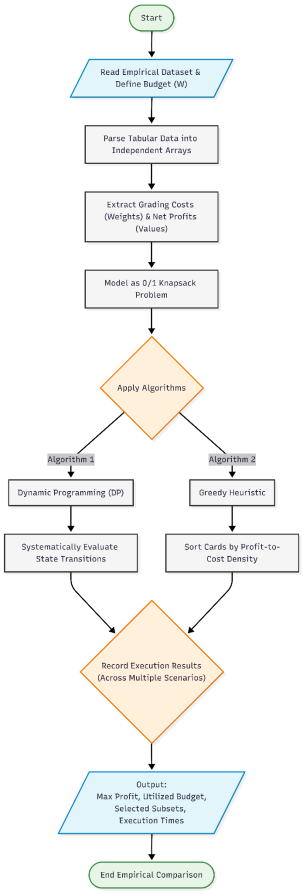

Fig. 1. Flowchart of the algorithm comparison procedure.

A. Experimental Environment Setup

The primary computational machine used to conduct the algorithmic experiments in this research is equipped with an 11th Gen Intel Core i7 processor and 32 GB of RAM, operating on Windows 11 Pro 64-bit. This hardware provides sufficient memory to handle extensive matrix tabulations without memory paging overhead.

This study was developed using the Python programming language version 3.12. The implementation was carried out within a local virtual environment to manage software dependencies effectively. The core library utilized for data manipulation and extraction was **`pandas`** version 3.0.3, while the built-in **`time`** module was utilized for high-resolution execution timing. This isolated setup ensures strictly

deterministic dependency management and provides precise computational benchmarking for evaluating algorithmic performance.

B. Dataset Preparation

The empirical dataset utilized in this study is a structured CSV file denoted as **`dataset_charizard_75_final.csv`**. The dataset encapsulates the financial metrics of $N = 75$ distinct Charizard Pokémon cards. The data encompasses three critical variables necessary for the 0/1 Knapsack problem formulation:

- `Nama_Kartu`: The specific identifier and set nomenclature of the Charizard card.
- `Biaya_Grading_USD`: The upfront capital cost required to submit the card to PSA for authentication and grading.
- `Profit_Bersih_USD`: The projected net profit after successfully grading and liquidating the card at its estimated PSA 10 market value.

Nama_Kartu	Harga_Raw_USD	Biaya_Grading_USD	Estimasi_Harga_PSA10_USD	Profit_Bersih_USD
Mega Charizard X ex #125 (Phantasmal Flames)	856	50	4280	3374
Charizard ex #199 (Scarlet & Violet 151)	403	25	2015	1587
Charizard #4 (Base Set)	391	25	1955	1539
Mega Charizard X EX #23 (Promo)	46	15	230	169
Mega Charizard Y ex #294 (Ascended Heroes)	600	50	3000	2350
Charizard #4 (Celebrations)	196	25	980	759
... (69 rows omitted) ...	...	...	...	...

Fig. 2. Dataset representation in a tabular format (sample of 5 records).

C. Data Extraction and Mapping

The preprocessing phase involves loading the CSV file into a memory-efficient Pandas DataFrame and extracting the necessary columns into independent sequential arrays. In the context of the 0/1 Knapsack optimization problem, `Biaya_Grading_USD` is mapped as the "weights" array (w_i), while `Profit_Bersih_USD` is mapped as the "values" or profit array (v_i). The extraction function returns these arrays along with the total scalar number of items (N), ensuring that the data structure is perfectly formatted for direct algorithmic processing.

D. Dynamic Programming Application

The Dynamic Programming (DP) algorithm is designed to evaluate all possible submission combinations without redundant recalculations, guaranteeing the discovery of the absolute maximum profit. The process begins by initializing a two-dimensional matrix K of dimensions $(N + 1) \times (W + 1)$ filled with zeros, where W denotes the designated maximum capital budget.

Step-by-step procedures for the DP execution are as follows:

1. Initialize the 2D array K[i][w] with zeros for the item index i from 0 to N and the capacity w from 0 to W.
2. Iterate through each card i from 1 to N.
3. For each card, iterate through all possible budget sub-capacities w from 1 to W.
4. If the grading cost of the current card (w_{i-1}) is less than or equal to w, calculate the maximum between including the card (v_{i-1}+K[i-1][w-w_{i-1}]) and excluding the card ($K[i-1][w]$). Store this optimal scalar in $K[i][w]$.
5. If the grading cost exceeds w, the item cannot be included; inherit the optimal value from the previous stage: $K[i][w] = K[i-1][w]$.
6. Upon populating the matrix, perform a backtracing loop starting from K[N][W] to identify the exact indices of the selected cards. If $K[i][w] \neq K[i-1][w]$, it mathematically proves that card i-1 was selected, and the remaining budget w is consequently reduced by w_{i-1}.

```python
def knapsack_dp(W, weights, values, n):
    # Inisialisasi matriks K dengan nol
    K = [[0 for _ in range(W + 1)] for _ in range(n + 1)]

    # Membangun matriks (Bottom-Up)
    for i in range(n + 1):
        for w in range(W + 1):
            if i == 0 or w == 0:
                K[i][w] = 0
            elif weights[i-1] <= w:
                K[i][w] = max(values[i-1] + K[i-1][w-weights[i-1]], K[i-1][w])
            else:
                K[i][w] = K[i-1][w]

    # Proses Backtracing untuk menemukan item terpilih
    res = K[n][W]
    w = W
    item_terpilih = []
    for i in range(n, 0, -1):
        if res <= 0:
            break
        if res == K[i-1][w]:
            continue
        else:
            item_terpilih.append(i-1)
            res -= values[i-1]
            w -= weights[i-1]
    return K[n][W], item_terpilih
```

Fig. 3. Python implementation of the Dynamic Programming matrix traversal and backtracing.

E. Greedy Algorithm Application

As a comparative baseline, a heuristic Greedy Algorithm is implemented. This algorithm circumvents the construction of an exhaustive matrix and instead makes locally optimal decisions based strictly on the cost-efficiency of each card.

Step-by-step procedures for the Greedy execution are as follows:

1. Create a dynamic array of dictionaries, where each object represents a card encapsulating its original index, cost (w_i), profit (v_i), and a calculated density ratio (r_i).
2. Compute the efficiency ratio: $r_i = v_i / w_i$. If $w_i = 0$, the ratio is explicitly set to 0 to prevent zero-division anomalies.
3. Sort the entire array of cards in descending order based strictly on their computed r_i value.
4. Initialize the total_profit and total_weight accumulators to zero.

5. Iterate through the sorted array sequentially. If adding the current card's cost to total_weight does not exceed the maximum budget W, append the card to the selection array and update the respective accumulators.

```python
def knapsack_greedy(W, weights, values, n):
    items = []

    # 1. Menghitung rasio profit terhadap biaya (density ratio)
    for i in range(n):
        rasio = values[i] / weights[i] if weights[i] > 0 else 0
        items.append({'index': i, 'rasio': rasio, 'weight': weights[i], 'value':
values[i]})

    # 2. Mengurutkan kartu berdasarkan rasio tertinggi (Descending)
    items.sort(key=lambda x: x['rasio'], reverse=True)

    total_profit = 0
    total_weight = 0
    item_terpilih = []

    # 3. Iterasi seleksi kartu (Local Optimum Choice)
    for item in items:
        # Jika kartu masih muat di dalam sisa modal, masukkan ke subset
        if total_weight + item['weight'] <= W:
            item_terpilih.append(item['index'])
            total_profit += item['value']
            total_weight += item['weight']
    return total_profit, item_terpilih
```

Fig. 4. Python implementation of the the Greedy sorting and selection procedure.

F. Execution and Scenario Testing

To comprehensively evaluate the performance disparity and accuracy trade-offs between the two algorithmic approaches, the primary execution pipeline (jalankan_komparasi) tests the empirical dataset across a predefined spectrum of budget constraint scenarios: $W \in \{100, 250, 500\}$.

For each constrained scenario, both the DP and Greedy algorithms are invoked sequentially. To measure the precise computational efficiency, Python's `time.perf_counter()` function is utilized immediately before and after the execution of each algorithm. This method invokes a highly accurate hardware clock with the highest available resolution, measuring the execution duration in fractional seconds. Ultimately, the experimental pipeline outputs the maximum profit achieved, the total grading budget utilized, the algorithmic execution time, and an explicit trace of the selected cards, thereby facilitating a rigorous empirical comparison between mathematical optimality and heuristic efficiency.

IV. Result and Discussion

The optimization evaluation was conducted using a dataset of 75 Charizard Pokémon cards with an overarching goal to maximize the net profit of PSA grading submissions. The experiment aimed to assess how two distinct algorithmic approaches Dynamic Programming (DP) and the Greedy heuristic perform under varying capital constraints. The algorithms were tested across three discrete budget scenarios ($100, $250, and $500), and their performance was measured based on the absolute maximum profit achieved, the total budget utilized, and the computational execution time. The comprehensive results of this experiment are presented in Table II.

TABLE II. Experiment Results - Optimization Summary

Budget Limit (W)	Algorithm	Max Profit Achieved	Utilized Budget	Execution Time (s)
$100	Dynamic Programming	$20,433	$95	0.001932
$100	Greedy Algorithm	$20,433	$95	0.000047
$250	Dynamic Programming	$46,436	$240	0.006388
$250	Greedy Algorithm	$46,436	$240	0.000072
$500	Dynamic Programming	$63,781	$495	0.013196
$500	Greedy Algorithm	$63,781	$495	0.000048

The most significant and striking finding from the empirical data is that the Greedy algorithm achieved the exact same absolute maximum profit as the Dynamic Programming approach across all three budget constraints. In the highly restrictive $100 scenario, both algorithms selected the Charizard [1st Edition] #4 and Charizard VMAX #SV107, yielding a total profit of $20,433 with $5 of leftover capital.

Fig. 5. Representative premium Charizard cards selected by the algorithms: *(a.)* Charizard [1st Edition] #4 and *(b.)* Charizard VMAX #SV107.

This perfect convergence continues even at the highest tested capacity of $500, where both algorithms identically selected a combination of 9 premium cards, accumulating a massive profit of $63,781 and effectively utilizing $495 out of the $500 budget.

This total convergence is mathematically rare for a 0/1 Knapsack problem constraint, where the Greedy approach typically falls into a sub-optimal local maximum. The phenomenon observed in this study indicates that the Charizard dataset exhibits a specific profit-to-cost density distribution where the most efficient cards perfectly align to fill the knapsack without leaving significant, unusable fractional capacity. Consequently, the local optimums chosen by the Greedy heuristic naturally cascaded into the absolute global optimum mathematically guaranteed by the DP approach.

While the financial outputs were identical, the computational efficiency between the two algorithms highlights a massive fundamental trade-off. Figure 5 visually illustrates the disparity in execution time. The Greedy algorithm, characterized by an $O(N \log N)$ complexity due to its sorting mechanism, maintained a nearly instantaneous execution time of approximately 0.00005 seconds regardless of the budget scale. Conversely, the DP approach, bounded by an $O(N \times W)$ complexity, demonstrated a linear increase in execution time relative to the budget expansion. At a $500 budget, the DP algorithm required 0.013196 seconds making it approximately 275 times slower than the Greedy counterpart.

In conclusion, the results highlight a practical realization in combinatorial economic optimization: while Dynamic Programming provides mathematical peace of mind by guaranteeing a global optimum, it suffers from severe computational scaling limitations. However, for the specific domain of Pokémon TCG grading optimization, the Greedy algorithm demonstrates exceptional robustness. It yielded a 0% profit loss compared to DP while offering exponentially superior computational speed. For investors scaling this model to thousands of cards or managing specialized portfolios, the Greedy algorithm is highly recommended as the most efficient and practically optimal solution.

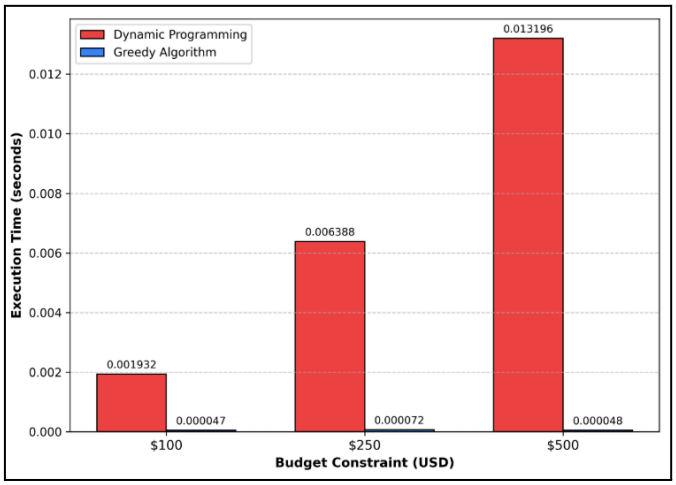

Fig. 6. Execution Time Comparison: Dynamic Programming vs Greedy Algorithm.

V. Conclusion and Suggestions

This study substantiates the efficacy of applying combinatorial optimization algorithms to maximize financial returns in Pokémon Trading Card Game (TCG) grading submissions. The empirical results illuminate a critical trade-off between strict mathematical optimality and computational efficiency. Across all evaluated budget constraints ($100, $250, and $500), the heuristic Greedy algorithm remarkably converged to the exact absolute maximum profit achieved by the Dynamic Programming (DP) approach. Despite this financial parity, a profound divergence in computational speed was observed. While the DP approach guarantees a global optimum, its execution time scales linearly with budget expansion due to its pseudo-polynomial complexity. Conversely, the Greedy algorithm maintains near-instantaneous execution irrespective of the capacity constraint. At the maximum $500 budget tier, the Greedy heuristic performed approximately 270 times faster than the DP counterpart. Consequently, the density-based Greedy algorithm is highly recommended for this specific economic domain, offering investors and collectors a highly scalable, rapid portfolio optimization tool without any degradation in financial yield.

For future research trajectories, it is recommended to explore advanced algorithmic variations, such as the Multiple Knapsack Problem (MKP), to simulate simultaneous submissions across competing grading authenticators (e.g., PSA, BGS, and CGC), each characterized by distinct fee structures and market premiums. Furthermore, incorporating bounded limitations such as a maximum allowable card count per submission tier would evolve the formulation into a Bounded Knapsack Problem (BKP), yielding a more realistic financial model. Additionally, deploying this algorithmic framework across broader and more volatile datasets, including modern Pokémon expansions or alternative TCG ecosystems like *Magic: The Gathering*, coupled with the integration of real-time market price APIs, would rigorously validate the generalizability and robustness of the Greedy heuristic under dynamic market conditions.

VI. Appendix

The complete source code, empirical dataset, and experimental scripts utilized in this study are open-source and can be accessed through the following GitHub repository link: https://github.com/dzakwanmkpp/MakalahSTIMA

Furthermore, a comprehensive video presentation demonstrating the algorithmic execution, code walk-through, and in-depth analysis of the experimental findings is available on YouTube at the following link: https://youtu.be/7YWlAnzAkkY?si=f4-sr5Xksyn8RER1

In addition, the slide deck utilized for the formal presentation of this research, summarizing the problem formulation, methodology, and empirical results, can be accessed through the following link: https://docs.google.com/presentation/d/1tLo5kHRr5z2iaI1jxWBKeqmxX3Nh0sUwuufqndnLvuQ/edit?usp=sharing

VII. Acknowledgment or Gratitude

The author expresses gratitude to God Almighty for His abundant blessings and grace throughout the process of writing this paper, enabling its timely completion. Appreciation is also extended to Mr. Dr. techn. Muhammad Zuhri Catur Candra, S.T, M.T., Mrs. Tricya Esterina Widagdo, S.T., M.Sc., and Mrs. Dr. Fazat Nur Azizah, S.T., M.Sc., lecturers of the IF2211 Algorithm Strategies course for class K03, for their guidance and valuable knowledge. They also provided comprehensive learning resources for this course, which greatly facilitated a deeper understanding of the material. Furthermore, heartfelt thanks are conveyed to the author's parents and younger sibling for their unwavering support and prayers, which have been a constant source of motivation and strength throughout this journey.

References

[1] **R. Munir**, "*Greedy Algorithm (Part 1-3),*" in Algorithm Strategy Lecture Notes, Department of Informatics, Bandung Institute of Technology, 2026.

[2] **R. Munir**, "*Dynamic Programming (Part 1-2),*" in Algorithm Strategy Lecture Notes, Department of Informatics, Bandung Institute of Technology, 2026.

[3] **R. Munir**, "*Theory of P, NP, and NP-Complete (Part 1-2)*," in Algorithm Strategy Lecture Notes, Department of Informatics, Bandung Institute of Technology, 2026.

[4] **S. A. Cook**, "*The complexity of theorem-proving procedures,*" in Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, 1971, pp. 151–158. (Fundamental reference for the NP-Complete class).

[5] **H. T. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein**, *Introduction to Algorithms, 3rd ed. MIT Press*, 2009. (Standard textbook for 0/1 Knapsack problem discussion).

[6] **PriceCharting**, "*Charizard Prices & List,*" 2026. [Online]. Available: https://www.pricecharting.com. [Accessed: June 18, 2026].

[7] **Professional Sports Authenticator (PSA)**, "*PSA Grading Standards,*" 2026. [Online]. Available: https://www.psacard.com/grading. [Accessed: June 18, 2026].

STATEMENT

I hereby declare that the paper I have written is entirely my own work. It is not an adaptation, translation, or copy of someone else's paper, and it does not contain any elements of plagiarism.

Bandung, 19th June 2026

Dzakwan Muhammad Khairan P. P. - 13524145