Optimizing the March Madness Bracket: A Stochastic Approach to Maximizing Expected Payout

Ethan Nigrin

Abstract

This paper presents a computational framework for generating optimal brackets for NCAA "March Madness" betting pools. Unlike traditional approaches that maximize expected accuracy, this method maximizes expected payout by accounting for the game-theoretic dynamics of pool size and opponent behavior. By simulating the tournament outcome and the specific composition of opponent brackets via Monte Carlo methods, I identify a submission that balances predictive accuracy with strategic differentiation.

1 Problem Formulation

1.1 Context: The NCAA Tournament Structure

Each spring, 64 college basketball teams compete for the national championship in a single-elimination tournament known as "March Madness." The tournament includes six rounds of play, resulting in a total of 63 games.

1.2 The Betting Pool Mechanism

Before the tournament begins, individuals commonly join prediction pools. The mechanics of these pools are as follows:

- **Submission:** Each participant submits a "bracket," a complete prediction of the winner for every game in the tournament.
- **Blind Entry:** Participants cannot view opposing brackets until the deadline passes and all selections are locked.
- **Payout:** Participants submit money into a prize fund. The participants whose brackets score the most points, based on the pool's specific scoring rules, receive a share of the prize fund based on the pool's payout rules.

1.3 The Objective

The computational task is to generate a single bracket that maximizes **Expected Payout** ($E[P]$) against N unknown opponents. It is crucial to distinguish this objective from maximizing the expected raw score ($E[S]$), as the two objectives are not necessarily congruent.

2 Theoretical Motivation

This problem presents a unique challenge because the optimal strategy is not immediately intuitive. While a novice approach focuses solely on basketball analysis, the true complexity lies in the intersection of statistical inference, game theory, and crowd psychology.

2.1 The Limits of "Chalk" (The Expected Score Fallacy)

The most common strategy employed by participants is to select the teams with the highest probability of winning. This is often referred to as picking "chalk."

While this method theoretically maximizes a bracket's **Expected Score** ($E[S]$), it often fails to maximize **Expected Payout** ($E[P]$). This discrepancy exists due to the "crowded field" phenomenon:

- If a team is a heavy favorite, a large percentage of the pool is likely to pick them.
- If that favorite wins, Participants who choose that team must then rely on differentiating picks in earlier rounds.
- Consequently, the most likely outcomes can become the least profitable due to high ownership.

2.2 The Risk-Reward Trade-off

To maximize payout, a participant must consider differentiating their bracket from the field. This requires a strategy that balances **Accuracy**—picking winners—with **Uniqueness**—picking winners that opponents missed.

This introduces a complex optimization problem:

1. **Calibration:** Can I accurately assess the "true" win probability of a specific team?
2. **Opponent Modeling:** Can I accurately predict the distribution of opponent picks?
3. **Differentiation:** How much risk should be taken? A "contrarian" pick offers high leverage. If it hits, the participant surges past the field, but it comes with a lower probability of occurrence.

2.3 The Impact of Pool Size (N)

The optimal balance between accuracy and variability is sensitive to the number of opponents (N) in the pool.

- **Small Pools ($N < 20$):** Due to the small sample size, there is significant variance in opponent pick distribution. Thus, a participant might win simply by picking strong teams, as it is possible no one else picked them.
- **Large Pools ($N > 1000$):** As N increases, the Law of Large Numbers dictates that the distribution of opponent picks will converge toward the public consensus. Therefore, as N increases, a participant cannot rely on variance in the opponent pick distribution when making picks.

The problem of March Madness optimization is not merely about predicting basketball games; it is a multi-variable computational problem requiring the maximization of payout against an unknown distribution of N opponents, where the optimal strategy shifts based on crowd psychology and pool size.

3 High-Level Approach

To address the complexities outlined above, I decomposed the solution into two distinct phases: **Simulation** and **Optimization**.

3.1 Simulate the World

Because of the statistical complexity of the problem, I used a Monte Carlo simulation to model the problem stochastically. I generated a massive dataset of "Pool Instances." Each instance represents one possible realization of the future, containing:

1. **A Tournament Outcome:** A future tournament outcome generated via Monte Carlo simulation using predictive basketball analytics.
2. **Opponent Brackets:** A set of N competing brackets generated to mimic the pick tendencies of the general public.

3.2 Optimize the Bracket

With a robust simulation of the world, I employed an optimizer to explore the decision space of possible brackets to identify a single submission that yields the highest mean payout across all simulated pool instances.

4 Methodology: Tournament Outcome Prediction

To solve the optimization problem of maximizing payout, I must first be able to generate realistic realizations of the tournament. My approach relies on Monte Carlo simulation where I predict the outcome of the tournament by simulating each game in the first round, advancing the winners to the second round, and repeating this process for each round until a champion is crowned.

This simulation requires a robust probabilistic model for single-game outcomes, specifically a method to calculate the probability that Team i defeats Team j.

4.1 Model Assumption

I postulate that the outcome of any single college basketball game is a stochastic process. Specifically, I model the points scored by Team i minus the points scored by Team j as a random variable drawn from a Normal Distribution:

$$Points_i - Points_j = D_{ij} \sim \mathcal{N}(\mu_{ij}, \sigma_{ij}^2) \tag{1}$$

Where:

- D_{ij} is the final point differential for the specific matchup between Team i and Team j.
- μ_{ij} is the expected point differential for the specific matchup.
- σ_{ij} represents the standard deviation of the point differential for the specific matchup.

4.2 Inference (Prediction Phase)

To operationalize this model, I utilize the **Adjusted Efficiency Margin (**$AdjEM$**)** and **Adjusted Tempo (**$AdjT$**)** metrics provided by Ken Pomeroy (KenPom).

Metric Definitions:

- $AdjEM_i$ **(Adjusted Efficiency):** The expected point differential between Team i and a theoretical "average" team per 100 possessions.
- $AdjT_i$ **(Adjusted Tempo):** The expected number of possessions by Team i in a 40-minute game against a theoretical "average" team.

I derived that μ_{ij} is defined as follows:

$$\mu_{ij} = \left(\frac{AdjEM_i - AdjEM_j}{100}\right) \times \left(\frac{2}{\frac{1}{AdjT_i} + \frac{1}{AdjT_j} - \frac{1}{AdjT_{avg}}}\right) \tag{2}$$

Where $AdjT_{avg}$ is the expected number of possessions by a theoretical average team in a 40-minute game against another theoretical average team. The first term $\left(\frac{AdjEM_i - AdjEM_j}{100}\right)$ represents the expected point differential per possession, and the second term $\left(\frac{2}{\frac{1}{AdjT_i} + \frac{1}{AdjT_j} - \frac{1}{AdjT_{avg}}}\right)$ represents the expected number of possessions in the game.

Similarly, I derived that σ_{ij}^2 is defined as follows:

$$\sigma_{ij}^2 = \sigma_{avg}^2 \times \left[\frac{1}{2AdjT_{avg}}\left(\frac{2}{\frac{1}{AdjT_i} + \frac{1}{AdjT_j} - \frac{1}{AdjT_{avg}}}\right)\right] \tag{3}$$

Where σ_{avg} is the inherent standard deviation of point differentials in average-paced college basketball games, empirically observed to be approximately 11 points.

Win Probability Calculation: With the distribution parameters defined, calculating the win probability becomes a straightforward inference task. Under my model assumption, the condition that Team i defeats Team j is equivalent to the condition that the realized point margin D_{ij} is greater than zero.

$$P(\text{Team } i \text{ Wins}) = P(D_{ij} > 0)$$

Since D_{ij} is normally distributed with mean μ_{ij} and standard deviation σ_{ij}, this probability corresponds to the area under the Gaussian probability density function (PDF) from 0 to positive infinity. This is computed analytically using the error function (erf):

$$P(\text{Win}) = \frac{1}{2}\left[1 + \text{erf}\left(\frac{\mu_{ij}}{\sigma_{ij}\sqrt{2}}\right)\right] \tag{4}$$

5 Methodology: Opponent Modeling

To maximize Expected Payout ($E[P]$), it is insufficient to merely predict game outcomes; I must also predict the specific composition of the opposing brackets in the pool. Since the exact brackets of the N opponents are unknown until the deadline, I rely on public aggregate data to simulate a representative distribution of the field.

5.1 Data Source: Public Pick Frequencies

I utilize publicly available data from CBS Sports, a major platform hosting millions of tournament brackets. For every team i and every round k, CBS publishes the percentage of users who selected Team i to **win** that specific round.

I define $C_{i,k}$ as the proportion of all public brackets that selected Team i to win in Round k.

5.2 Simulation Procedure (Forward Approach)

One intuitive method for generating opponent brackets is the Forward approach, which mimics the chronological order of the tournament.

In this method, for each opponent bracket, I simulate picks starting with the First Round, then the Second round, and so on until the Championship game is decided. For any matchup in Round k between Team i and Team j, the conditional probability that an opponent picks Team i is derived by normalizing their aggregate popularity for that round:

$$P(\text{Pick } i \mid i, j \text{ in Round } k) = \frac{C_{i,k}}{C_{i,k} + C_{j,k}} \tag{5}$$

5.3 Simulation Procedure (Backward Approach)

The alternative method, which I employ in the final model, is the Backward approach. I generate an opponent bracket by filling the bracket in reverse order, starting with the National Champion and working backward to the First Round.

Step 1: Sampling the Champion
I begin by selecting the tournament winner. I treat the distribution of all 64 teams as a categorical distribution, where the probability of selecting Team i is equal to their specific crowd frequency for winning the Championship ($C_{i,6}$).

$$P(\text{Champion} = i) = \frac{C_{i,6}}{\sum_{n=1}^{64} C_{n,6}} \tag{6}$$

Once Team i is selected as the Champion, they are automatically placed as the winner of all six of their scheduled matchups (from the First Round through the Finals).

Step 2: Recursive Backfilling (Finding Opponents)
With the winner of a specific Round k slot determined (let us call this $Team_{win}$), I must determine who they defeated in that round ($Team_{loss}$).

- The opponent must originate from the specific sub-region of the bracket opposite $Team_{win}$ for that round. Let S_{branch} be the set of valid teams in that opposing sub-region.
- I sample the losing team (let us call this $Team_{loss}$) from S_{branch} proportional to their likelihood of **reaching** that round (which is equivalent to winning the previous Round $k-1$).

$$P(\text{Opponent} = j \mid j \in S_{branch}) = \frac{C_{j,k-1}}{\sum_{m \in S_{branch}} C_{m,k-1}} \tag{7}$$

I repeat this process recursively for every empty slot in the bracket until every pick is decided.

5.4 Rationale: The Trade-off for Precision

Ideal modeling of opponent behavior would utilize a dataset of individual completed brackets to capture correlations between picks (e.g., "Homer Bias," where a participant systematically over-selects teams from a specific conference). However, public platforms only release aggregate marginal statistics ($C_{i,k}$), not the covariance structure of user selections.

Ideally, a simulation from public pick frequencies would perfectly match public pick frequencies for every team in every round. However, due to the mathematical inconsistencies inherent in using aggregate data to model complex, correlated individual choices, the Forward and Backward simulation methods are not guaranteed to perfectly preserve the data.

More specifically, the Forward approach accumulates error as it progresses, distorting the distribution of later rounds (Final Four and Champion). Conversely, the Backward approach fixes the Champion distribution as an input parameter, shifting the approximation error to the earlier rounds.

I utilize the Backward method because standard scoring systems are heavily weighted toward later rounds (e.g., the Championship is often worth 32x more than a Round 1 game). For the purpose of maximizing Expected Payout, accurately modeling the distribution of the Championship and Final Four picks in opponent brackets is far more critical than achieving granular precision in the First Round. Furthermore, when simulating opponent brackets with the Backward approach, the distribution of first round picks passed a hypothesis test against public pick frequencies at the 0.01 significance level, indicating that the error propagation in the Backward approach was not significant. While advanced techniques such as Latent Variable Models can theoretically reduce errors, the Backward heuristic provides a computationally efficient and accurate approximation.

6 Methodology: Pool Simulation & Computational Optimization

To accurately estimate Expected Payout ($E[P]$), I must evaluate potential brackets against a statistically significant number of pool simulations. I define a single “Pool Instance” as a tuple containing one specific Tournament Outcome and a set of N Opponent Brackets. To ensure convergence, I targeted a sample size of $M = 10,000$ pool instances.

6.1 The Computational Bottleneck

A direct simulation of this magnitude presents a significant computational challenge. A naive implementation using nested loops would require iterating through 10,000 tournament simulations, generating N unique opponent brackets for each, and individually scoring every bracket. In Python, processing this sequentially for a pool size of 100 proved computationally prohibitive, with estimated runtime extending into days.

6.2 Efficient Score Pre-computation

Instead of generating unique opponents for every single tournament outcome (which is redundant), I generated a fixed “Universe” of M representative opponent brackets and M representative tournament outcomes. I then computed how *every* opponent bracket performs against *every* tournament outcome.

By using matrix operations for scoring, I reduced the compute time from days to a few minutes.

6.3 Sampling and Filtering

From this precomputed $(10,000 \times 10,000)$ matrix, I construct the final training data for the optimizer:

1. **Sampling:** For each tournament outcome j, I randomly sample N opponent bracket scores.
2. **Top-K Filtering:** Since payout structures typically reward only the top percentiles (e.g., 1st, 2nd, 3rd place), the exact scores of poor-performing brackets are irrelevant. To minimize memory overhead, I discard all opponent scores except the top K values for each pool instance.

Result: The final output is a compact dataset of 10,000 simulated pools, where each pool contains a tournament outcome and the “Target Score Thresholds” required to win money in that specific realization of the tournament.

7 Methodology: Bracket Optimization

With a robust simulation of tournament outcomes (Section 3) and a pre-computed dataset of opponent scores for each tournament outcome (Section 5), I can formally define the objective. I seek the single bracket that maximizes the expected payout across M simulated pool instances:

$$\max_{\text{Bracket}} E[P] = \frac{1}{M} \sum_{m=1}^{M} \text{Payout}(\text{Bracket}, \text{Pool}_m) \tag{8}$$

Given the search space of 2^{63} possible brackets, exhaustive search is computationally intractable. To solve this, I designed a custom heuristic optimizer that exploits the hierarchical scoring structure of the tournament.

7.1 Initialization (Warm Start)

Standard local search algorithms are sensitive to their starting conditions. Instead of initializing with a random bracket or a “Chalk” (favorite-heavy) bracket, I initialize the search using the highest-scoring bracket found within the generated opponent pool. This provides a “warm start,” placing the optimizer in a high-potential region of the solution space immediately.

7.2 The “Backward-Greedy” Search Strategy

I employ an optimization approach that works in **reverse chronological order**, starting with the Championship game and working backward to the First Round.

Rationale: In standard scoring systems, the point value of a game typically doubles in each subsequent round (e.g., Round 1 = 1 point, Championship = 32 points). Consequently, the choices made in later rounds have a significantly higher impact on the final payout than choices in earlier rounds. By optimizing the high-leverage decisions first, I effectively constrain the search space for the lower-value decisions.

7.3 Iterative Optimization Procedure

The algorithm refines the bracket through multiple passes. A single optimization pass proceeds as follows:

1. **Step 1: Championship Optimization**
   The process begins at the Championship Game (Round 6). I identify a subset of teams capable of reaching this stage. For each candidate team, I generate a temporary bracket where that team is set to win the Championship *and* all preceding games required to reach the final. I evaluate the Expected Payout for each temporary bracket against the simulated pools. The candidate that yields the highest payout becomes the fixed winner in the current bracket.

2. **Step 2: Recursive Descent (Undecided Games)**
   The algorithm then proceeds to the Final Four (Round 5). Note that by selecting a Champion in Step 1, one of the two Final Four games has already been effectively decided (the Champion must win their semi-final). Therefore, I only optimize the remaining **undecided** Final Four game. I repeat the evaluation process: modifying the bracket to test different winners for this slot, calculating the expected payout, and locking in the candidate that maximizes value.

3. **Step 3: Sequential Backfilling**
   This logic is applied recursively downwards through the Elite Eight, Sweet Sixteen, and so on. At each step, I focus only on the game slots that remain undecided by higher-level choices.

4. **Computational Efficiency (Pruning)**
   To maintain computational feasibility, I do not evaluate every possible team for every slot. Instead, for each game, I restrict the candidate list to a “Best Subset” of teams—specifically,

those with the highest KenPom Adjusted Efficiency Margin ($AdjEM$) eligible to reach that round.

7.4 Convergence

Once a full pass from the Championship to the First Round is complete, the entire process repeats. I continue these backward sweeps until a full iteration results in no changes to the bracket. This indicates that the solution has converged to a local maximum, typically stabilizing within a few minutes of runtime.

8 Results and Empirical Validation

To evaluate the efficacy of the proposed solution to maximizing expected payout in bracket pools, I analyzed the performance of the optimizer's generated bracket ("BracketBot") in both a controlled simulation environment and a live, high-stakes wagering pool.

8.1 Simulation Performance

I first evaluated the optimized bracket against the empirical distribution of opponent strategies generated in Section 4. Across 10,000 simulated pool instances with a pool size of 150, the optimizer successfully identified a candidate solution with an exceptionally high Expected Payout ($E[P]$).

Win Rate Comparison:

- **Naive Baseline:** A best performing opponent bracket from the public distribution achieved a 1st place finish in approximately **1.3%** of simulations.
- **BracketBot (Optimized):** The optimized bracket achieved a 1st place finish in **8.0%** of simulations.

This **6x improvement** demonstrates that the model successfully navigated the trade-off between accuracy and uniqueness, identifying a specific combination of picks that offered significant positive expected value (+EV) against the field.

8.2 Real-World Deployment (Case Study)

To further evaluate these theoretical results empirically, the optimized bracket was entered into a real-world pool consisting of $N = 150$ participants with a total prize purse of $2,000 for 1st place.

Performance Metrics: By the conclusion of the Regional Finals, BracketBot had achieved near-perfect accuracy in the high-leverage rounds, correctly predicting:

- **8 of 8** Elite Eight teams.
- **4 of 4** Final Four teams.

Validation of Differentiation Strategy: The results also highlighted the mechanics of the "Crowded Field" theory proposed in Section 2.1. In this specific pool, 36 other participants (24% of the field) selected the same National Champion as BracketBot.

- **The Problem:** Typically, sharing a champion with 24% of the pool reduces expected payout.
- **The Solution:** Because the optimizer had specifically selected a high-value path through the early rounds to maximize differentiation, BracketBot **led all 36 of these brackets** in total points entering the Final Four.

This suggested that the optimizer did not merely pick a winning team; it constructed a differentiated portfolio of picks that offered leverage against similar opponents.

8.3 Financial Outcome and Hedging

Entering the Final Four, BracketBot held the statistical lead in the pool, with all remaining picks favored by external betting markets. Recognizing the high equity value of this position, I executed a hedging strategy using commercial sportsbooks (DraftKings) to lock in a guaranteed profit regardless of the specific variance of the final games.

Ultimately, the bracket's selected champion (Duke) suffered an upset defeat, eliminating BracketBot from the 1st place prize in the pool. However, due to the strategic hedge placed against this outcome, the venture resulted in a net profit of **$530**, successfully converting the bracket's statistical edge into realized financial gain.