A Heuristic Strategy for the Berghain Challenge

Roko Čubrić
roko.cubric555@gmail.com
Faculty of Electrical Engineering and Computing
University of Zagreb

September 6, 2025

Abstract

The Berghain Challenge is a problem where you need to fill a club to a specific capacity while also meeting quotas for different kinds of people. You also want to reject as few people as possible. This paper describes a simple heuristic method to solve this. We give each person a score based on how much they are needed and compare it to a threshold that changes over time. This method is fast and works well.

1 Problem Definition

The goal is to manage entry to a club. We have the following fixed values:

- N: The total number of people the club can hold.
- $\mathcal{A}$: The set of all possible attributes a person can have.
- M_i: The minimum number of people required for each attribute $i \in \mathcal{A}$.
- p_i: The probability that a person arriving has attribute i.

Our job is to fill the club to capacity N and make sure we have at least M_i people for every attribute, while turning away the minimum number of people.

2 The Heuristic Model

Our model decides whether to accept or reject a person, P_k, by calculating a score, $S_t(P_k)$, and comparing it to a dynamic threshold, τ_t. The decision is made according to the following rule:

$$\text{Decision}_t(P_k) = \begin{cases} \text{Accept} & \text{if } S_t(P_k) > \tau_t \\ \text{Reject} & \text{otherwise} \end{cases} \tag{1}$$

The following subsections detail how the score and threshold are calculated.

2.1 Calculating the Score

The score for a candidate is based on how much their attributes are needed at that moment. This is a sum of values for each attribute they have.

First, we define the **Need** for an attribute i as how many more people we need with that attribute:

$$\eta_t(i) = \max(0, M_i - c_t(i)) \tag{2}$$

where $c_t(i)$ is our current count of admitted people with attribute i.

Next, we calculate the **Urgency**. This value increases as the club gets fuller, making it more important to get the attributes we still need.

$$U_t(i) = \frac{\eta_t(i)}{N - a_t} \tag{3}$$

Here, a_t is the number of people already admitted.

The **Scarcity** of an attribute makes rare attributes more valuable.

$$\sigma(i) = \frac{1}{p_i} \tag{4}$$

Finally, we add a **Finisher's Bonus** to give a big boost to people who can help us complete a quota that is almost full.

$$B_t(i) = \begin{cases} \beta \cdot \sigma(i) & \text{if } 0 < \eta_t(i) \leq \eta_{\text{bonus}} \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

Here, β and η_{bonus} are parameters we can tune.

The final score for a person P_k with a set of attributes A_k is the sum of these parts for all of their needed attributes:

$$S_t(P_k) = \sum_{i \in A_k, \eta_t(i) > 0} [U_t(i) \cdot \sigma(i) + B_t(i)] \tag{6}$$

2.2 The Dynamic Threshold

We are picky at the start and less picky at the end. We do this by having a threshold that starts high and gets lower as the club fills up. We define the progress $\rho_t = a_t/N$. The threshold τ_t is:

$$\tau_t = \tau_0 \cdot (1 - \rho_t^\gamma) \tag{7}$$

τ_0 (base threshold) and γ (decay shape) are also tunable parameters.

3 Special Rules for Edge Cases

Sometimes the scoring logic isn't enough, so we have two special rules.

- **Desperation Mode:** If the number of people we need for an attribute is equal to or greater than the number of empty spots left, we must accept anyone who has that attribute.

- **Mop-Up Mode:** Once all attribute quotas are met, our only goal is to fill the club. At this point, we accept everyone.

4 Finding the Best Parameters

Our model has four parameters we need to set: τ_0, γ, β, and η_{bonus}. To find the best values, we used a computer simulation.

First, we wrote a program to generate a random queue of people, where each person's attributes were based on the given probabilities p_i.

Then, we used an iterative Monte Carlo method to find good parameter values.

1. We started by picking many random combinations of the four parameters. For the decimal-valued parameters we used a normal distribution, and for the integer parameter we used randint.
2. For each combination, we ran our simulation 10 times and averaged the score. Using the average score helps make sure the result isn't just due to random luck on one run.
3. We took the best 50% of the parameter combinations and discarded the rest.
4. In the next round, we focused our search around the parameter values that worked well. We did this by taking the mean and variance of the top half and using a new normal distribution with a smaller variance to pick new random values.

We repeated this process several times. Each time, we narrowed the search space, allowing us to zoom in on the parameter values that gave the best performance, as shown in Figure 1.

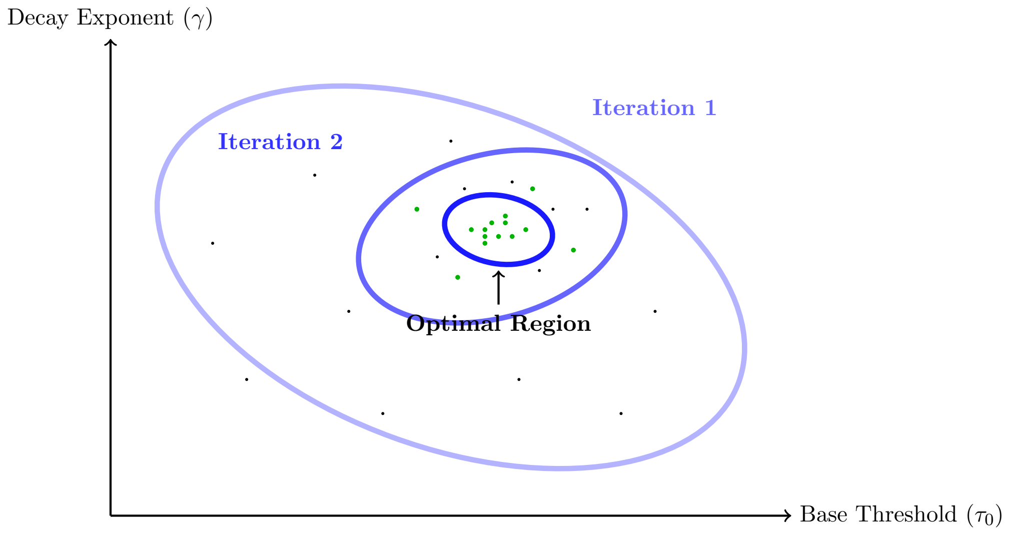

Figure 1: Visualization of the iterative Monte Carlo optimization for two hyperparameters. The search space (blue ellipses) shrinks and centers on the region of best-performing samples (green dots) from the previous iteration.