Diet Optimization for Dysbiotic Microbiome

Amirhesam Abedsoltan, Deevanshu Goyal, Mahsa Nafisi

January 2023

1 Problem statement

Diet has been shown to affect the composition of the human gut microbiome by changing the number and activity of different microbial communities by providing or depleting the necessary nutrients. This important role suggests that a modified and personalized diet could be utilized to help re-establish a healthy microbiome from a diseased state. Given the heterogeneity of these communities, and their differences across individuals and time, this approach presents two main challenges:

1→ defining the healthy vs. disease state microbiome profile and,
2→ finding an optimal diet that can most efficiently restore the healthy gut microbiome.

In this project, we aim to address the second challenge.

2 Understanding the Data

We are given a list of 400 Amplicon Sequence Variants (ASVs) with their respective relative abundances (RA) in multiple pathological microbiome samples. These samples are further mapped to three categories - "Normal", "Deviant", and "Other" (we ignore the third category). Lastly, a Nutrient Impact Matrix (NIM) tabulates the predicted propensity of each ASV to benefit from each nutrient. Based on the given data, we define the following:

1. We define $A := |ASV|$ as total number of pathological microbes, and $NL := |Nutrients|$ as total number of nutrients.
2. **Test Microbiome Sample(s) (TMS)**: A vector of length A. Index $a \in \{0, \dots, A-1\}$ represents an ASV with Relative Abundance (RA) as the indexed value. Corresponds to the *Deviant* sample.
3. **Reference Sample Collection (RSC)**: An $A \times S$ matrix where $a \in \{0, \dots, A-1\}$ represents an ASV, and $s \in \{0, \dots, S-1\}$ represents a reference sample, and value RA_{as} represents the relative abundance (RA) of each ASV in each sample. Corresponds to the collection of "Normal" samples. Furthermore, we refer to each row of this matrix by RA_a.
4. **Nutrient Impact(NI) Matrix (NIM)**: An $A \times N$ matrix with values $NI_{an} \in (0, 1]$ where each row $a \in \{0, \dots, A-1\}$ represents an ASV and each column $n \in \{0, \dots, NL-1\}$ represents a nutrient. The value NI_{an} can be interpreted as the likelihood of observing a growth in the population of ASV_a given the use of nutrient n.

For a fixed ASV, referred to as ASV_a, we look at corresponding row in RSC, refered to as RA_a, and define the following values,

$$maxRA_a = mean(RA_a) + std^1(RA_a), \tag{1}$$
$$minRA_a = mean(RA_a) - std(RA_a) \tag{2}$$

Using the parameters defined above, we classify the corresponding ASV_a in the TMS (denoted as TMS_a) into one of the following three classes,

$$N = \{TMS_a : minRA_a < TMS_a < maxRA_a, \quad \forall a \in [0, A-1]\} \quad \text{(Normal)}$$
$$U = \{TMS_a : TMS_a < minRA_a, \quad \forall a \in [0, A-1]\} \quad \text{(Under-represented)}$$
$$O = \{TMS_a : TMS_a > maxRA_a, \quad \forall a \in [0, A-1]\} \quad \text{(Over-represented)}$$

Objective: For each nutrient n and each microbe ASV_a, the matrix NIM provides us with a probability in the range $(0, 1]$, indicating the likelihood that ASV_a will utilize the nutrient n if it is included in the diet. If our aim is to boost the population of ASV in set U, we should select nutrients with high utilization probability for this group. Conversely, if we aim to prevent an increase in the population of ASV in set O, we should choose nutrients with low utilization probability for this group. To implement this, we define the reward function $R(M)$ for any set M of nutrients, with size m, i.e $|M| = m$. The purpose of the reward function is to penalize set M whenever a member of set O utilizes at least one nutrient in M with a probability of at least ϵ_O. On the other hand, set M will be rewarded whenever a member of set U utilizes at least one nutrient in M with a probability of at least ϵ_U. Therefore, mathematically, R can be defined as follows, where $\mathbb{1}$ is a function such that $\mathbb{1}(True) = 1$ and $\mathbb{1}(False) = 0$ (note that the probabilities of ϵ_O and ϵ_U can be considered hyperparameters that can be adjusted later as necessary).

$$R(M, NIM, O, U) \triangleq \sum_{i \in U} \mathbb{1}(1 - \prod_{l \in M} (1 - NI_{il}) \geq \epsilon_U) - \sum_{i \in O} \mathbb{1}(1 - \prod_{l \in M} (1 - NI_{il}) \geq \epsilon_O) \quad \text{(Auxiliary function)}$$

Input: The sets U, O, the matrix NIM of size $A \times NL$, and the allowed number of nutrients, m.
Output: A subset of at most m nutrients M, that maximises $R(M)$.

Simplifying assumptions: Biological systems are complex and intertwined, making it difficult to design a precise model to quantitatively explain their behavior. This necessitates major simplifying assumptions in formulation of models that attempt to describe them. Acknowledging this complexity, we describe the simplifying assumptions in our approach and their shortcomings in practical biological applications. The first step in designing a therapeutic approach is defining normal vs. pathological state. As discussed before, this benchmark is difficult to establish. The input RSC is a population-averaged number, masking non-pathological diversity in taxonomic profiles. As a result, individual outcomes of the suggested diet may not prove effective. We assume no inter-species dependencies, and our model does not account for the effect of competition once the fitness of a species improves upon receiving supplementation. This assumption does not hold in vivo where the activity, metabolic output, and population of a species, among other factors,

[1] Standard deviation

may have an effect on other communities. Furthermore, we assume each ASV's response to a diet is uniform and a result of combined effect of individual nutrients as defined by the cumulative NI score while in practice such an isolated system is not achievable in HGM.

3 Algorithmic approaches

The naive approach to maximize R is to try all possible pairs of available nutrients, up to a maximum of m, and choose the ones with the highest R. Although this approach yields the optimal solution, it becomes infeasible as n and m increase, with a complexity of $O(n^m)$ which is exponential in m.

Since finding the global optimum is infeasible, we need to use practical methods that can be implemented. First, we establish a baseline that allows us to compare meaningfully with other attempts. Our algorithmic approaches are based on randomized algorithms, Gibbs' sampling, and divide and conquer techniques.

The codebase used to implement the mentioned algorithms as well as the data used for the study have been uploaded on Github[2].

3.1 Naive randomized algorithm

An initial naive attempt to use randomized algorithms is to randomly select m nutrients, evaluate the score R of the first $k \leq m$ nutrients, and select the subset with the highest R score. This process is repeated at least 50,000 times, and the subset with the highest R score is kept. However, a major limitation of this approach is that each time a new set of m nutrients is sampled, all information from previous iterations is lost, and there is a risk of obtaining an even worse R score. Nevertheless, this method serves as a useful baseline for comparison.

3.2 Gibbs sampling

Once we have computed the baseline using the randomised algorithm approach, we will use the Gibbs sampling approach to make local improvements in the baseline set of nutrients that we have achieved. Below we discuss how our approach works, depending on the size of the baseline nutrient set.

As we have defined above, the baseline would be defined as a subset of at most m nutrients (M), that maximises R. If the number of selected nutrients is **less than** m, we will perform either of the following two actions: (i) sample a nutrient index uniformly from the total set of selected nutrients and replace it with another nutrient not present in the list; or (ii) randomly add a nutrient not selected earlier to the list. If the number of selected nutrients is **equal to** m, we will perform either of the following two actions: (i) sample a nutrient index uniformly from the total set of selected nutrients and replace it with another nutrient not present in the list; or (ii) randomly remove a nutrient from the list.

[2]https://github.com/DeevanshuGoyal/CSE282A_Project

At each local iteration, we choose the action to be performed on the nutrient set by randomly choosing between the two possible actions while weighing each of them with a fixed probability: we choose to randomly remove/add a nutrient by a probability of 20%, and choose to randomly replace an existing nutrient in the set by a probability of 80%. Based on the outcome of this random sampling, we perform the corresponding action and obtain the transformed nutrient set.

Thus, in each iteration t of this approach, we have two nutrient lists at our disposal - the baseline set (let us refer to this as X_t) and the transformed set (let us refer to this as X_{t+1}). In each iteration t, we compute R for X_t and X_{t+1}. If $R(X_t) > R(X_{t+1})$, we retain the original baseline set and move onto the next iteration, i.e., $X_{t+1} = X_t$. Otherwise, we choose the transformed set and move to the next iteration, i.e., $X_{t+1} = X_{t+1}$. We make local improvements on a given set of nutrients across 1000 iterations, while choosing the baseline set of nutrients across 5000 iterations.

3.3 Randomized Divide and Conquer

Initially, we considered a naive greedy approach, which selects the top m nutrients with the highest R. However, this method is flawed as the top m nutrients may target the same underrepresented or over-represented nutrients.

To address this issue, we propose a modified divide and conquer approach. The nutrients are divided into m randomly selected sets, and one nutrient with the highest score as per the defined R function, is chosen from each set. This approach has a complexity of $O(n)$ and can be implemented efficiently. However, the resulting solution may not be optimal, as the way we divide the n nutrients into m subsets can significantly impact the final performance. For instance, the optimal solution could consist of several nutrients from one subset and none from others.

To overcome this limitation, we plan to use randomized algorithms. By repeating the algorithm t times, we can obtain a near-optimal solution with high probability. Although the complexity of this approach is $O(tn)$, the algorithms can run independently, allowing for efficient parallelisation with modern technologies. However, formalizing the conjecture that we can achieve a near-optimal solution with high probability requires theoretical thinking and may be challenging.

4 Results

When we are working on a task, it is important to have a baseline level of performance that we can use to compare our results against. This helps us determine if our approach is effective and whether our results are meaningful or not. For example, in a classification task, we can randomly assign labels to inputs to create a chance level of performance. This is useful as it gives us a point of comparison for the performance of our actual model.

However, in some cases, the performance of the chance level is not well-defined or may not be clear. This can happen when the task itself is ambiguous or when there is no clear standard for what constitutes random chance. In such cases, it is important to establish a baseline level of performance using a pure chance approach.

Once we have established a baseline level of performance, we can use it to evaluate the effectiveness of our approach. If our model performs significantly better than the baseline, then we can conclude that our approach is effective. On the other hand, if our model performs no better than the baseline, then we need to reevaluate our approach and consider other factors that may be affecting the performance of the model. Therefore, establishing a baseline level of performance is a crucial step in any task as it helps us measure the effectiveness of our approach and determine the significance of our results. In section 4.1, we establish this baseline and in section 4.2, we improve performance by locally fine-tuning nutrient choices. Finally, in section 4.3, we implement our divide-and-conquer approach with the hope of further improving the score.

In all algorithmic approaches, we utilized the hyperparameters ϵ_O and ϵ_U. These hyperparameters determine the probability threshold for any nutrient to affect an over-represented and under-represented ASV population in the TMS sample, respectively. A higher threshold results in a lower reward for the impact of nutrients on the respective category of ASVs. We experimented with threshold values ranging from 0.5 to 0.9 for each hyperparameter and ultimately selected $\epsilon_O = 0.9$ and $\epsilon_U = 0.5$. This effectively means that we measured the impact of the chosen subset of nutrients by placing greater emphasis on its impact on under-represented nutrients rather than over-represented ones. This decision was based on the final scores we obtained for different values of the hyperparameters.

4.1 Naive Randomized Algorithm

To establish the baseline for these experiments, we implemented the randomised algorithmic approach to select the best subset of m nutrients from the total number of available nutrients that maximised our defined reward function R. During the implementation, we tried to understand the nature of our results and optimise the final results by modifying the various variables such as m, which is the size of the nutrient subset chosen as well as the hyper-parameters.

We have earlier defined m as the size of the nutrient subset selected for dietary intervention to maximize the R function. We experimented with a range of values from {10, 20, ..., 60}. Ultimately, we settled on $m = 10$, as it resulted in the best possible reward (*-1 in this case*). We conducted 5,000 repetitions of the experiment for each value of m and kept the best performance.

4.2 Gibbs Sampling

With the setting described in the previous sub-section, for each iteration of the naive randomized algorithm, we locally enhanced its performance by utilizing 1000 Gibbs iterations as outlined in section 3.1. Through this approach, we observed that we successfully improved the final reward from -1 to 5. However, this drastic improvement in the final score comes at the cost of computational speed since this method is significantly time-consuming due to the extensive number of local iterations required.

Another point to note is the probability with which we are choosing the action to make local changes during each iteration of this approach. As mentioned before, we make a random addition/subtraction from the nutrient subset with a probability of 20%, whereas we make a random replacement in the subset with a probability of 80%. These probabilities were chosen after multiple

iterations of this approach but under future experimentation, these probabilities can be varied on other criterion (such as the size of the nutrient subset under question) to test the impact of the said criterion on the final improvement in the score.

4.3 Divide and Conquer

Finally, we implemented the algorithm outlined in section 3.3 across 5,000 iterations (similar to the other approaches). We observed that this algorithm produced results that were comparable to the Gibbs Sampling approach, but it achieved those results notably faster since it does not require local search, resulting in significant savings in computation speed.

Furthermore, an important observation was the fact that under this approach, the score progressively worsened for an increasing value of m (it went from 5 for $m = 10$, to -15 for $m = 25$). This makes sense since as m increases, we are bound to find nutrients that target the same set of ASVs, which effectively doesn't improve the score since we count the impact of such nutrients on any ASV as 1. This serves as a further proof for our choice of m as 10, that we initially established after analysing the results we obtained from section 4.1.

	Best score	Number of nutrients
Naive Randomized Algorithm	-1	10
Gibbs Sampling	5	10
Divide and Conquer	5	10

Table 1: Performance comparison of all algorithmic approaches.

5 Discussion

We believe that selecting the "correct" problem formulation, or more specifically, the reward function, can lead to even better solutions. Additionally, determining a biologically meaningful reward function is crucial for this problem. One possible next step is to use the same reward function but apply a weighted average to over-representative and under-representative summation in the reward function. This weight could be considered as a hyper-parameter that could be fine-tuned. More importantly, we could entirely change the reward function and meticulously select one that is biologically relevant.

Moreover during our experiments, we observed a significant variation in the relative abundances of ASVs in the "Normal" samples of the RSC. This resulted in a higher standard deviation in the ASV distribution and thus, defined a larger normal range for that particular ASV in a healthy person. Owing to this, we had to change the parameters ($maxRA_a$ and $minRA_a$) defined under section 2 to accommodate for the large variance in the reference sample data. This variability could be due to the subjective nature of defining a healthy population - a particular ASV could be present in a higher relative abundance in healthy person A as compared to another healthy person B, leading to the idea that normal samples should also be clustered on the basis on certain criteria.

Currently, we have adopted a naive approach of using the entire set of normal samples instead of clustering them. A possible next step could be to explore the clustering of the reference sample database (RSC) into various groups that better signifies the subjective definition of a healthy gut microbiome. Once we have achieved that, we can implement the algorithmic approaches described on each of the clusters separately, thus identifying a 'personalised' dietary intervention plan for each cluster.

6 Conclusion

When brute force algorithms are computationally infeasible and finding the optimal solution is NP-hard, randomized algorithms, in conjunction with greedy or local search approaches, can generate valuable local optima. In this report, we have demonstrated that a local search randomized algorithm and a greedy randomized algorithm can significantly enhance performance in a real data setting (summary of the results can be found in table 1).