543 Final Project
Meal Planning

Josh Pullen, Oren Bell, Colton Lewis, Tri Pham

***Abstract*—This project studies the problem of planning meals to optimize for cost, preparation time, and variety in what you eat. We formulate the meal planning problem as a Lagrangian optimization problem with non-linear constraints and use the branch and bound approach to derive its corresponding solution. We then proceed to explore different diet configurations, subject to a variety of nutritional constraints. We find expected solutions for simple test cases, and we explore optimal solutions to real nutritional constraints like those recommended by the FDA and those typically used in bodybuilding.**

I. Introduction

There are few decisions people face as frequently as the decision of what to eat, and each of the several meals consumed each day represent a culmination of time and money invested. While these commodities are valuable for anyone, they tend to be in especially short supply for college students.

A recent study shows how a lack of confidence in food preparation and financial worry prevent college students from having healthy diets Knol et al. [2019]. This study reports that students who could not grocery shop due to their financial instability also had lower health and grades. In general, there are also various other factors that can affect college students' diets. A survey from CNBC finds that 36% of college students are considered "food insecure," mostly due to financial constraints Hess [2018]. However, that does not include the students who, due to limited time or the aforementioned budgetary constraints, get enough sustenance but poor nutritional variety. In essence, the selection of meals by college students with limited time and budget, but an unavoidable biological need for nutrition, represents an optimization problem.

To address this problem, we have designed a solution leveraging branch and bound optimization to select an optimal combination of meals to keep the user well nourished over the course of a week while minimizing monetary cost and preparation time.

Nutrients	Lower constraint	Upper constraint
Sugar (g)	0	252
Fiber (g)	105	N/A
Sodium (mg)	10500	16100
Vitamin A (IU)	21000	70000
Vitamin C (mg)	630	14000
Vitamin D (IU)	4200	28000
Vitamin E (IU)	150	10500
Vitamin B12 (mcg)	16	N/A
Calcium (mg)	7000	17500
Iron (mg)	56	315
Potassium (mg)	23800	N/A

TABLE I: Weekly nutritional constraints for an adult male

The algorithm selects these meals from a set of meals that we, as students, commonly eat. Here, the term "meal" is used very loosely to mean a discrete quantity of food that is preparable as a unit. It may be a single dietary pill, a ready-to-go snack, and proper entrée, or a mealprepped batch that's meant to be consumed over multiple days.

Each individual meal in the set is defined by a subset of features. First and foremost, each meal is defined by its nutritional information, with values provided for the following: Calories, carbohydrates, protein, fat (saturated and unsaturated), sugar, fiber, sodium, vitamins (A, B12, C, D, E), calcium, iron, and potassium. Additionally, the meals are defined by their estimated cost per meal and estimated time to prepare. Finally, we included a feature to capture the defining qualities of meal, which is used to evaluate sameness among entries in the data set.

For simplicity, we have chosen our nutritional constraints to be based on an adult male of 19 years or older. Nonetheless, these constraints can be updated easily to reflect constraints for other genders

and age ranges. As we later define our nutritional constraints to be on a weekly basis, we assume that the nutrients considered meet their daily values. For example, if a college male student consumes at least 2000 calories daily, then their weekly calories constraint is set to be at least 14000. Above is a brief table denoting some additional weekly nutritional constraints for our problem, with their Recommended Dietary Allowances (RDAs) and Tolerable Upper Intake Levels (ULs) recommended by NIH USDA [2020] FDA [2022]. Although these might not necessarily be strict constraints, they help establish general lower and upper bounds that are solvable with our chosen branch and bound approach.

II. Related Theory and Practice

Our problem can be mapped to the form of a traditional non-linear optimization problem with inequality constraints.

$$\text{minimize} \quad f(X) \tag{1}$$
$$\text{subject to:} \quad g_j(X) \leq 0 \tag{2}$$

Where X is a vector input representing the number of each meal consumed and $g_j(X)$ comprise some set of constraints. Our constraints are all linear. A constraint limiting our weekly diet to less than 700g of sugar, for instance, would be mathematically expressed in equation 3, where S is a vector listing the sugar content of each meal.

$$S^T X - 700 \leq 0 \tag{3}$$

Our objective function has three constituent parts to optimize: cost, time, and variety. The former two are linear, whereas variety is not. The final objective function is given as

$$f(X) = C(X) + \beta T(X) - \gamma H(X) \tag{4}$$
$$f(X) = C^T X + \beta T^T X - \gamma \sum_j b_j^T X \ln b_j^T X \tag{5}$$

$C(X)$ is the cost we try to minimize and is a simple vector multiplication between the cost of each meal, C^T, and the quantity of each consumed, X. $T(X)$ is the total time spent prepping the meal plan and is likewise a linear operation with T^T being the time to prepare each meal.

β and γ are coefficients meant to adjust the preference of each metric. Cost has an implicit coefficient, α, which is assumed to be 1. This implicit $\alpha = 1$ means we can think of the objective function as being expressed in dollar units. β can then be viewed as dollars per minute and is a measure of how the user values their time. γ is less concrete, unit-wise.

$H(X)$ is mean to be an entropy calculation. The sparse matrix b_{ij} is a collection of coefficients assigning an occurrence rate of each ingredient tag to a meal. i indexes the meal, and j indexes the tag.

The vector operation $0 \leq b_j X \leq 1$ returns the percentage of calories in the mealplan attributable to that ingredient. Individual entries in the matrix b are calculated recursively as

$$b_{ij} = \frac{Cal_i}{mCal^T X} \tag{6}$$

Where m is the number of ingredients in meal j, Cal_j is the number of calories per meal j, and $Cal^T X$ is the total calories in mealplan X.

This entropy calculation is the only non-linear component of this problem, and Shannon Entropy is known to be convex Cover [1999]. The calculation of b_{ij} is merely a normalization step that does not detract from the convexity. Since this variety metric is subtracted from strictly linear elements, we know our objective function is concave. And thus any local minima, if one exists, will also be a unique global minima that can be found through simple gradient descent. We will discuss later in II-B why this minima doesn't exist.

Given the nature of the problem, we constrain our solution to be an integer solution. You can't have 76% of a sandwich as a diet recommendation. As with many discrete optimization problems, we leverage the branch and bound algorithm to artificially restrict our search space.

A. Branch and Bound

Branch and bound is a discrete optimization technique for finding optimal solutions without needing to perform an exhaustive search over the full set of all possible solutions S. It covers the solution

space as a tree, dividing it up into disjoint subsets by adding additional constraints or partial solutions. The bounding principle allows us to quickly prune branches without needing to full enumerate all solutions they contain. For Y_1, Y_2 disjoint subsets of S, if we know that some feasible solution from Y_1 has some value k_1, and the optimal relaxed solution from Y_2 has value k_2, with $k_2 \geq k_1$, we can immediately disregard the set Y_2, since no solutions in Y_2 can score better than the solution scoring k_1 from set Y_1. Being able to prune branches is key to finding solutions quickly.

The branching procedure is also straightforward. Each node in the tree represents some subset Y_k of S via adding additional constraints. We solve the relaxed version of the problem (see II-B) over Y_k to get a lower bound for the best possible solution in Y_k. If the value found is greater than the best integer solution found so far, we may prune the branch immediately by the above bounding principle. We may also prune the branch if there are no feasible solutions found. If the solution found happens to be integral, and has a score lower than the best integral solution found so far, we update the best solution. Otherwise, we split the space Y_k into disjoint subsets via adding additional constraints on some variables, and proceed to another node in the tree. This process continues until all nodes have been enumerated or pruned, and is guaranteed to result in finding the optimal solution.

Figures 1 and 2 show an example of the branch and bound process on a toy problem. Here, the relaxed version of the problem drops the integer constraints - rather than restrict x_i to be either 0 or 1, we allow it to be in the range $[0, 1]$ as a real number. At each node, we add a constraint to the first variable that was non-integral in the optimal solution. As shown in Figure 1, in order to quickly find a valid solution, we use depth-first search, descending down the tree in order to find a solution under the original constraints. Once we find the first solution, we may use its value as a bound for other branches. In Figure 2, the scores for optimal relaxed values on the unvisited nodes are computed. Since they both have a value greater than the integral solution found from depth-first search, the two branches are pruned, and the search completes since there are no nodes left.

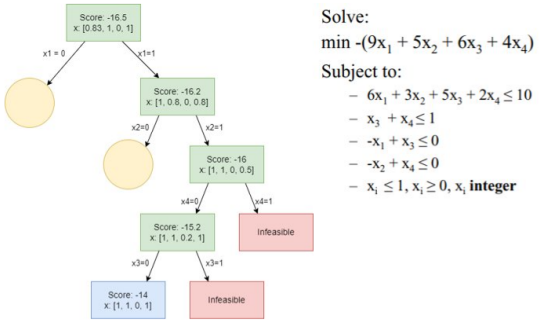

Fig. 1: Branch and Bound Initial Search

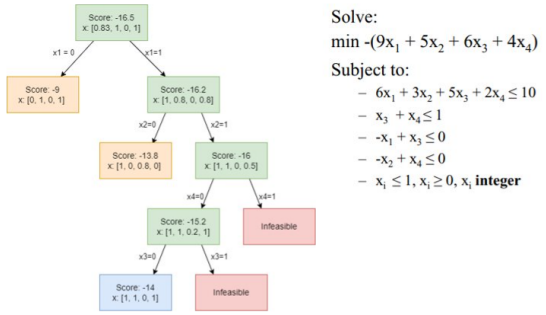

Fig. 2: Pruning Branches

B. *Relaxed Optimization*

As stated earlier, our problem is convex, but does not have a global minima. This is in part due to the linear cost and time tending towards negative infinity, see a simple 2-meal example in Figure 3. But even the variety metric alone doesn't have a minima. Seen in Figure 4, the extremum of the variety function, in blue, is not a point but a line, in black. As a artifact of the normalization we performed in formula 6, the important thing is not

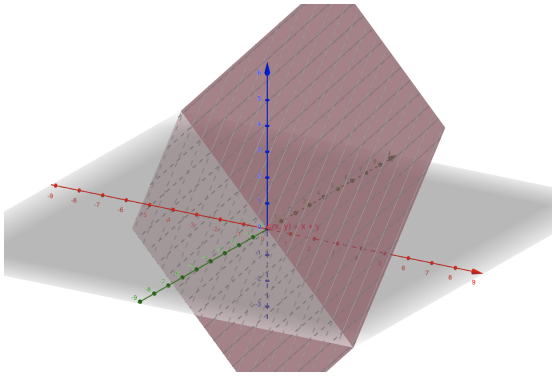

Fig. 3: Variety and Objective Function

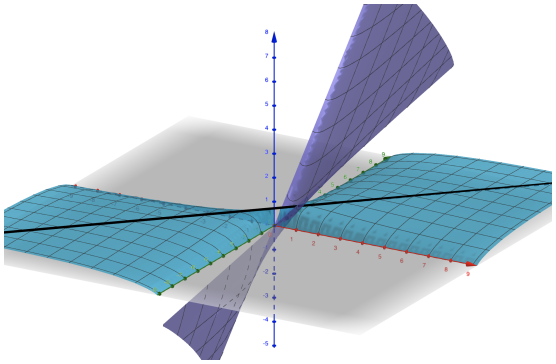

Fig. 4: Linear Function

the absolute quantities of X, but their proportional values to each other.

Traditionally, Shannon entropy is only defined for positive integers. But, as another artifact of the normalization step, our variety function is defined for all positive integers OR all negative integers, but no combination of the two. Combined with the linear cost and time functions, this creates a choke point of our final function, in purple at $< 0, 0, ... >$. Eating negative meals is a nonsense suggestion, so for practical purposes, we consider the origin to be our global minima, and enforce this with bounding constraints.

Adapting our objective to the Lagrangian form, we seek to minimize

$$\mathcal{L}(X, \lambda) = f(X) + \sum_i \lambda_i g_i(X) \tag{7}$$

Where λ_i is a Lagrangian multiplier for constraint g_j. It is constant at our optimal valid solution x^*, and has certain guaranteed properties if x^* exists. Among these, dual feasability (see formula 8) and complimentary slackness (see formula 9)Kuhn and Tucker [1951].

$$\lambda_i \geq 0 \tag{8}$$

$$\forall i | \lambda_i g_i(x^*) = 0 \tag{9}$$

We can treat λ_i as another input to $\mathcal{L}(X, \lambda)$, and the minima of that function will meet all constraints specified in $\mathbf{g}(X)$

As with any function, extremums can be found by setting the function derivative to zero. The derivative of our linear components are easy enough to find

$$C(X) = C^T X \implies \nabla C(X) = C \tag{10}$$

$$T(X) = T^T X \implies \nabla T(X) = T \tag{11}$$

$$g_j(X) = G_j^T X \implies \nabla g_j(X) = G_j \tag{12}$$

The variety function is a little more difficult. We skip the proof and provide it's derivative below.

$$\nabla H(X) = \langle \sum_j b_{ij} + b_{ij} \ln b_{ij} x_i \rangle |_i \tag{13}$$

Thus it follows

$$\nabla \mathcal{L}(X, \lambda) = \begin{cases} C + T + \nabla H(X) + \sum_j \lambda_j \nabla g_j(X) \\ \boldsymbol{g}(X) \end{cases} \tag{14}$$

Unfortunately, solving this system of equations reduces into a form of exponential sums, seen below, which are not solvable by arithmetic means, so we resort to gradient descent instead to find a numerical approximation.

$$g_j(\lambda) = a_j e^{A^T \lambda} + b_j e^{B^T \lambda} + \cdots + z = 0 \tag{15}$$

As mentioned before, our objective function $f(X)$ is concave, and all our constraints are linear, so gradient descent is guaranteed to find a unique optimum.

Given our derivatives, we perform an iterative gradient descent, incrementing values of X and λ until convergence. At each iteration, we check what constraints are inactive (ie satisfied), and their respective λ is set to zero, per the complimentary slackness condition of Kuhn and Tucker [1951]. We have refrained from employing the popular penalty and multiplier methods here as they result in lengthy and sometimes ill-fit convergence. Instead, the λ use ordinary gradient descent along with our ordinary inputs.

Our gradient length is set to 1 initially and will scale down whenever the angle between subsequent gradients is detected to be larger than $\pi/2$. That is, when it begins "circling the drain". The scaling factor is given by

$$s = 0.9 + \frac{\tanh \sqrt{V}}{10} \tag{16}$$

Where V is the sum violation of all constraints. This was chosen to improve fitting near the edges of constraints. Since λs are "turned off" once crossing into a constraint boundary, the value of x^k will very quickly shoot back out, and bounce back and forth across the constraint boundary. This $\tanh$ aspect allows a large violation to inhibit the attenuation of our gradient vector, so it shrinks faster when inside the boundary. When all constraints are met, $s = 0.9$. When large violations are found, $s \lessapprox 1$. This procedure continues until the gradient length reaches a sufficiently small level. Pseudocode is presented in Algorithm 2 of Section III.

III. Technical Details

Branch and Bound

Our branch and bound implementation defines nodes as a set of bounds on each input variable, creating constraints on how many times a meal can be or must be selected. To process a given node, we solve the relaxed (non-integer) version of the problem with the node's variable constraints in addition to the fixed nutritional constraints. We then follow the bounding principle discussed in Section II-A. If there is no feasible solution with the constraints and variable bounds, we prune the branch immediately. If the score of the relaxed optimal solution is worse than the best integral score found so far, we may also prune the branch. If the solution happens to be very close to integral, we treat it as integral and record its score - if the score of the integral version is better than the best solution found so far, we update the best solution.

Otherwise, we select one of the non-integral variables in the solution and create two bounds for it. If the variable x_i takes some value k in the optimal solution, we create a set of two bounds - $x_i \leq \lfloor k \rfloor$, and $x_i \geq \lceil k \rceil$. This creates two new nodes as children of the one being processed, which are then pushed onto the stack of all nodes.

The order that these two child nodes go on the stack is important for performance issues. To guarantee a $O(\log |X|)$ height to our tree, we would ordinarily split the solution space for x_i in half. That is, split at 64, 32, 16, 8, etc. By splitting at k instead, we get better average case performance, by skipping straight to the $x_i = 1$ value. But we risk an $O(n)$ runtime. Eg if $k = 2, 3, 4, \ldots$. To avoid this, we always put the child node with the smallest solution space on the top of the stack, to ensure that some integer solution is found quickly. Given the concavity of our problem, we expect an integer solution to be very close to the relaxed solution. This integer solution can then be used to prune its uncle nodes.

We iterate over the nodes of the tree in depth-first order to quickly find a feasible integral solution to prune other branches with. Using a score-based heap instead of the stack lead to much slower runtime, since it had to explore many more nodes before it could find an integral solution to begin pruning with. Pseudocode for this process is shown in Algorithm 1.

Algorithm 1 Branch and Bound

```
best ← ∞
sol ← []
S ← empty stack
push default bounds onto S
while S is not empty do
    b ← pop from S
    x ← relaxed solution with bounds b
    if no feasible x or score(x) ≥ best then
        (do nothing, branch pruned)
    else
        if x is integral and score(x) < best then
            best ← score(x)
            sol ← x
        else
            find x_i = k | k ∉ ℤ
            create b_l with bound x_i ≤ ⌊k⌋
            create b_r with bound x_i ≥ ⌈k⌉
            push b_l, b_r onto S
        end if
    end if
end while
```

Relaxed Optimization

Our relaxed optimization is done by formulating the problem in a Lagrangian form and performing gradient descent until an optimum is reached. The

length of the gradient vector is shrunk when it rotates by more than $90\,\mathrm{deg}$. The factoring of this shrinkage is $0.9 \leq s < 1$, and is larger when violations are occurring, as a way to motivate convergence within bounds instead of bouncing repeatedly across a boundary.

Values of λ are set to zero when their constraint is satisfied, and otherwise incremented by $g_j(x^k)$, where g_j is their respective constraint, and x^k is the current estimate of x^*. Pseudo code is given in Algorithm 2. There are detail omitted here that were meant to treat idiosyncrasies of our problem. For instance, the calculations of g_j are done as vector multiplication. We also check the value of V at each iteration, and if it's 0, we set a flag to indicate that a solution does in fact exist, which modifies our stopping conditions slightly. This addresses the issue of accidentally terminating at an invalid solution while ping-ponging across a constraint boundary. Lastly, many values are capped at arbitrarily high numbers to prevent them from going to infinity, which causes issues with using numpy.

Algorithm 2 Gradient Descent of Lagrangian Problem

$X \leftarrow < 1, 1, \cdots >$
$\lambda \leftarrow < 0, 0, \cdots >$
$S \leftarrow 1$
$d \leftarrow \nabla \mathcal{L}(X, \lambda)$
$grad \leftarrow \hat{d} * S$
while $S > 0.0001$ **do**
  $X \leftarrow X - grad$
  $V \leftarrow 0$
  for g_j in **g do**
    if $g_j(X) > 0$ **then**
      $\lambda_j \leftarrow \lambda_j + g_j(X)$
      $V \leftarrow V + g_j(X)$
    else
      $\lambda_j \leftarrow 0$
    end if
  end for
  $d' \leftarrow \nabla \mathcal{L}(X, \lambda)$
  if $\arccos \hat{d'} \cdot \hat{d} > \pi/2$ **then**
    $S \leftarrow S * (0.9 + \tanh(\sqrt{V})/10)$
  end if
  $d \leftarrow d'$
end while

IV. Experimental Results

By adjusting the constraints we assign and the values of β and γ, we can tweak the diet recommendation for individual user preference. Men generally need more fiber McManus [2019], whereas women need more iron HSPH [2020]. We conducted a collection of experiments tweaking these variables and present our algorithm's recommendation below.

No Constraints

Under this experiment, we assign $\beta = 0.15$, $\gamma = 0.25$, but impose no constraints beyond our input set being positive integers: $x \in X \mid x \geq 0$. Predictably, our result is the global extrema at the origin, i.e. eat nothing. Technically, starving is the best way to save money on food and time on cooking.

Calorie Constraint

Under this experiment, we assign $\beta = 0.15$, $\gamma = 0.25$, and add a constraint that weekly calories must exceed 14000 (2000cal/day). We get the recommendation illustrated in 5, which is dominated by cost-effective and ready to eat foods such as bagels, cereal, and even raw butter. If we set $\gamma = 0$, removing the variety constraint, we observe a recommendation of only butter. It is the most time and cost effective item in our dataset. This illustrates a need to incorporate a variety metric to prevent the algorithm from hyperfocusing on a few select ideal foods. This effect is less common once we add other nutritional constraints, which have a way of enforcing their own variety.

This proposed meal plan has a cost of \$18.14 and required 108 minutes of preparation.

Macronutrient Constraints

Instead of focusing solely on calories, we may also want to control their source. Calories can be obtained from 3 macronutrients: carbohydrates, fats, and protein. A typical American diet will apportion calories from these sources in a 50/30/20 ratio. We constrain our problem to attempt to match this ratio, $\pm 5\%$. Results are seen in Figure 6.

This proposed meal plan has a cost of \$40.1 and requires 178 minutes of preparation.

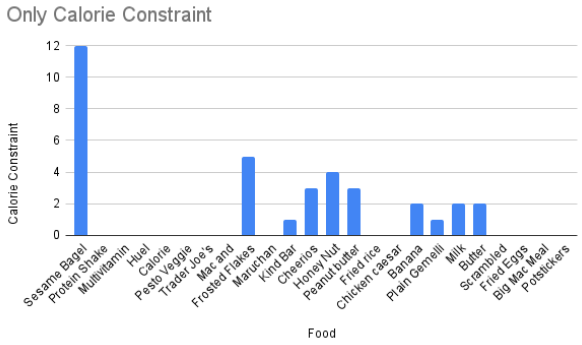

Fig. 5: Only Calorie Constraint

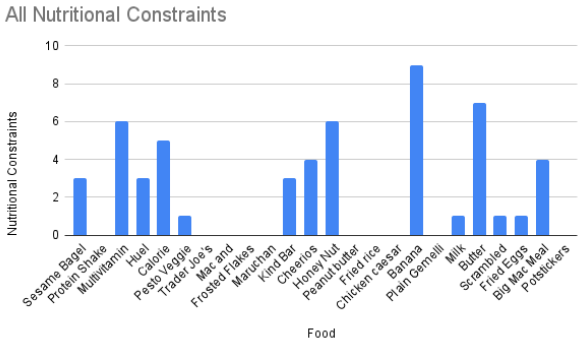

Fig. 7: FDA Recommended Constraints

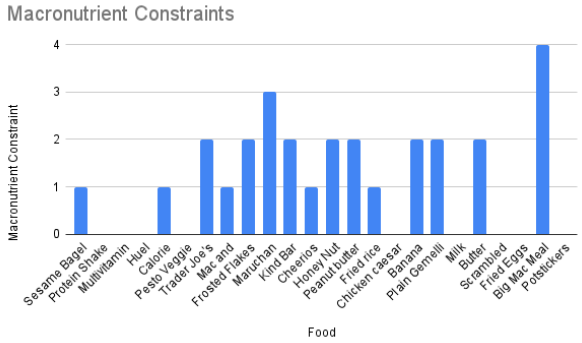

Fig. 6: With 50/30/20 Macronutrient Ratios

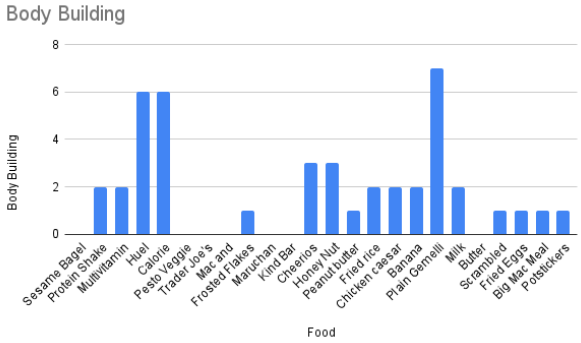

Fig. 8: Bodybuilding Diet

All Nutrition Constraints

Imposing nutritional constraints typical for an adult male, according to U.S. Departments of Agriculture (USDA), Health and Human Services (HHS) and Food and Drug Administration (FDA) recommendations USDA [2020] FDA [2022], we arrive at the mealplan suggested in 7. We will now refer to this as our ”base diet”.

This proposed meal plan has a cost of $62.11 and requires 115 minutes of preparation. The pricetag is now in the realm typical for a collegiate grocery budget, but the 2 hours of cooking is a noticeable improvement over our lifestyle.

Bodybuilding

Adjusting the constraints now, we attempt to construct a diet for an individual requiring 3000cal/day and a 60/10/30 balance of carbs, fat, and protein. Coefficients are reset to $\beta = 0.15$ and $\gamma = 0.25$. The result is in 8. Note the prevalence of protein-heavy meal replacement shakes like Huel and Calories Supplements. The rest of the diet is fairly heavy in carbohydrates with cereal, pasta, and fried rice.

This proposed meal plan has a cost of $76.59 and requires 320 minutes of preparation, the majority of which comes from boiling pasta and rice.

Keto

Using the same constraints as the base diet, we adjust the macronutrients to a 10/70/20 balance of carbs/fat/protein. This is to create a keto diet, a diet low in carbohydrates meant to drive the body into a state of ketosis, which reduces appetite and helps weight loss while retaining muscle definition.

No solution was reached in this case, as the data set only includes meals commonly eaten by college students, which do not lend themselves to the strict constraints of this keto diet.

Only Time Matters

Here, we keep the nutritional constraints recommended by the FDA, but assign a coefficient of 0

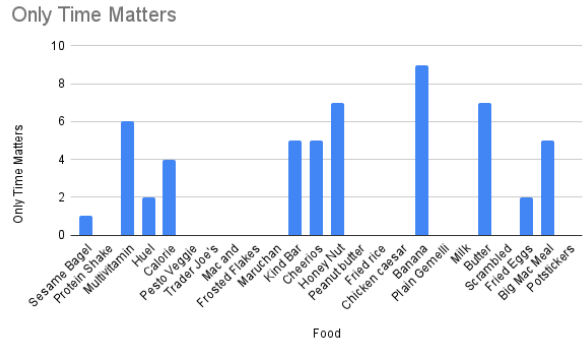

Fig. 9: Only Minimize Time

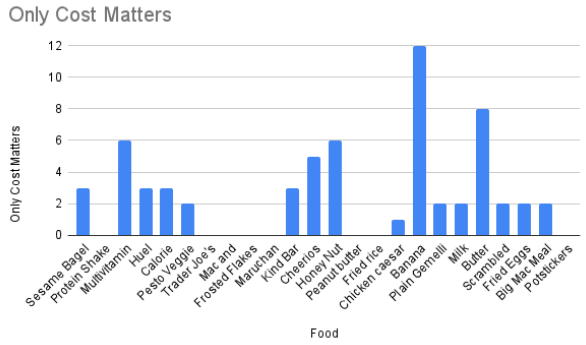

Fig. 10: Only Minimize Cost

to both cost and variety. We seek to only minimize time. The result can be seen in 9. Note that every meal selected takes under 5 minutes to prepare. The total time to prep this plan is 69 minutes, a 40% decline over our base diet. We also saved a bit of money, down to $59.64.

Only Cost Matters

In a related effort, we assign coefficients of 0 to both time and variety. Here we attempt to find the cheapest diet that meets FDA recommendations, by whatever means necessary. The result can be seen in 10. It does not save as much money as we had hoped, only down to $56.25. However, the meal prep time is up significantly to 207 minutes.

Since cost was already proportionally boosted well over time and variety, this metric had been fairly optimized even in the base diet, which is likely why we don't see much of an improvement. It is interesting that the time efficient diet has a comparable cost, highlighting the prevalence of convenience food in our dataset.

V. Conclusion

The need for quick, affordable, and substantial meals is a constant for college students. Moreover, it served as an optimization problem in need of solving. In general, we were satisfied with the flexibility of our solution and the ability to enforce variety to ensure our constrained solution sets represented realistic meal plans.

Unsurprisingly, the unconstrained solution was to go without any meals. Introducing constraints piece-wise continued to produce expected meal recommendations, with corresponding effects on cost and time. Similarly, adjusting the parameters for various diets altered the meal recommendations predictably, while also exposing some trends and limitations in our data set. Bodybuilding requires more calories and a change in the proportions of macronutrients, resulting in higher cost and preparation time. It is possible that, by updating our data set with meals better suited to these purposes, we could further reduce the prep time and save money. However, given the problem definition we chose (optimizing meals for an average college student), this represents an area for future growth that was beyond the bound of this initial investigation. The consequences of this limited scope was even more evident in our attempt to accommodate the keto diet, as the inability to produce an optimal solution can be traced back to an absence of meals in the data set that cater to the keto macronutrient balance. The takeaway from this is, once again, that accommodating more specific diets will require an updated data set that extends beyond the usual college student diet.

In conclusion, our solution can provide optimized meal recommendations for various criteria. Additionally, we have identified potential opportunities for improvement to accommodate more specific diets through updated data sets. Similarly, updating and expanding the data set could allow us to further improve the more general constrained solutions. Given the scope of our original problem definition, however, we are satisfied with the current performance.

VI. Contributions

Josh wrote the code for branch and bound, gave part of the project presentation, wrote the branch and bound sections of this paper in Related Theory and Technical Details, and contributed to the abstract.

Colton wrote the code for preprocessing and parsing the data set and wrote the introduction/conclusion sections of the paper.

Tri wrote the code for the nutritional constraints and updated parts of the abstract, introduction, and references.

Oren mapped the problem into Lagrangian form and did the differential calculus for us to perform the relaxed calculation, which he also implemented in code. He served as project manager, and wrote the bulk of sections 2 and 4, except the portion on branch-and-bound.

References

L. L. Knol, C. A. Robb, E. M. McKinley, and M. Wood. Very low food security status is related to lower cooking self-efficacy and less frequent food preparation behaviors among college students. *Journal of Nutrition Education and Behavior*, 51(3):357–363, 2019.

Abigail J. Hess. New study finds that 36% of college students don't have enough to eat, Apr 2018. URL https://www.cnbc.com/2018/04/06/new-study-finds-that-36-percent-of-college-students-dont-have-e html.

NIH. Dietary supplement fact sheets. URL https://ods.od.nih.gov/factsheets/list-all/.

USDA. Dietary guidelines for americans, 2020-2025, Dec 2020. URL https://www.dietaryguidelines.gov/sites/default/files/2020-12/Dietary_Guidelines_for_Americans_2020-2025.pdf.

FDA. Daily value on the new nutrition and supplement facts labels, Feb 2022. URL https://www.fda.gov/food/new-nutrition-facts-label/daily-value-new-nutrition-and-supplement-facts-labels.

Thomas M Cover. *Elements of information theory*. John Wiley & Sons, 1999.

HW Kuhn and AW Tucker. Nonlinear programming. In *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, pages 481–492. University of California Press, 1951.

Katherine D. McManus. Should i be eating more fiber?, Feb 2019. URL https://www.health.harvard.edu/blog/should-i-be-eating-more-fiber-2019022115927.

HSPH. Nutrition source - iron, Oct 2020. URL https://www.hsph.harvard.edu/nutritionsource/iron/.