Addressing the Diet Problem with Constraint Programming Enhanced with Machine Learning

Sara Jazmín Maradiago Calderón, Juan José Dorado Muñoz(✉), Juan Francisco Díaz Frías, and Robinson Andrey Duque Agudelo

AVISPA Research Group, Escuela de Ingenieria de Sistemas y Computacion, Universidad del Valle, Cali, Colombia
{sara.maradiago,juan.jose.dorado,juanfco.diaz, robinson.duque}@correounivalle.edu.co
http://avispa.univalle.edu.co

Abstract. In Colombia there is a problem related to eating habits that has its origin, mainly, in two causes: the lack of budget that allows access to a wider variety of food, and the lack of awareness among the population about their nutritional needs. To tackle this issue, a solution has been proposed using a Constraint Programming (CP) approach enhanced with Machine Learning (ML) for a version of the Diet Problem (DP).

A CP model was developed to find a shopping list that meets a family's nutritional needs while minimizing costs; and a synthetic dataset was created to test the model, which was run multiple times to collect results. Since DP is an NP-complete problem and computational time to find optimal solutions varies from one solver to another, a ML classifier was used to choose a solver that best performs in small cap time limits based on instance features (i.e., selection from an Algorithm Portfolio). After carrying out an extensive evaluation of the CP model, including our approach that implements a Classifier for algorithm selection, the model correctly selects the best solver over 68.07% of the time, for a sample of 1378 instances.

By analyzing the performance of different solvers on a set of instances, it can be predicted which solver is likely to achieve the best results on new instances. This approach could be extended to tuning solver parameters, which would further improve their efficiency and effectiveness. (The dataset used for the creation of this paper is available on: https://github.com/Git-Fanfo/dataset_CCC)

Keywords: Constraint Programming · Machine Learning · Classifier · Algorithm Selection · Diet Problem

1 Introduction

The problem of optimal food distribution dates back to ancient times, from the moment the first societies were formed. The methods to ensure equitable distribution were limited to dividing portions for different foods based on arbitrary parameters such as size, age, sex, occupation, and social caste. This arithmetic-based method persisted for most of human history, aiming to feed armies, institutions, households, and others while minimizing costs.

M. Tabares et al. (Eds.): CCC 2023, CCIS 1924, pp. 57–70, 2024.
https://doi.org/10.1007/978-3-031-47372-2_6

It was not until the 20th century, as modern computer science took its first great steps, that the Diet Problem emerged. It is an optimization problem model created by George Stigler in 1947 [11], "motivated by the desire of the United States military to ensure nutritional requirements at the lowest cost" [6]. Stigler formulated the problem in terms of Linear Programming, seeking to minimize the function corresponding to the total cost of the food to be purchased while satisfying a set of constraints related to a person's nutritional requirements. The DP is considered a challenging problem, classified as NP-Complete, and has been approached by multiple researchers since the early 1960s in their attempt to computerize it.

This study employs Constraint Programming to address the Diet Problem. Various methods exist for propagating and evaluating constraint problems, and the search for the optimal solution depends on the algorithm (solver) utilized. However, it is known that the chosen solver's performance relies on the specific parameters of the problem, including the problem domain, constraints, problem size, and objective function. To mitigate this issue, the utilization of Machine Learning is proposed to identify patterns that can establish a correlation between the initial parameters and the most suitable solver, leading to the discovery of better solutions.

In the following sections, we start by presenting a detailed explanation of the fundamental subjects for this study, Constraint Programming (Sect. 2.1), Algorithm Portfolio (Sect. 2.2) and Machine Learning (Sect. 2.3). Next, we delve into the details of the Diet Problem Model Formulation (Sect. 3) as a Constraint Programming model; this model defines the parameters, variables, and constraints necessary to optimize a shopping list for meeting nutritional requirements within a given budget with the objective to minimize nutritional deficiencies or excesses and minimize cost using a weighted sum approach. We then present the experimental setup (Sect. 4) and results, showcasing the performance and effectiveness of our proposed method.

Finally, we discuss the conclusions (Sect. 5) drawn from our findings and outline potential directions for future work. Since our Machine Learning approach showcased precise predictions, validating its effectiveness and potential for improvement, we were able to significantly reduce the computational time for solving the computationally intensive NP-Complete Diet Problem. This opens up possibilities for optimizing food distribution in previously infeasible real-world scenarios.

2 Constraint Programming and Machine Learning

2.1 Constraint Programming

Constraint Programming (CP) is a problem-solving methodology that centers on the definition and resolution of problems through the use of constraints. It involves the specification of variables, their possible values, and the necessary conditions that must be met. A key component in the CP approach is the utilization of a solver algorithm, which is a specialized software tool or component responsible for finding solutions to constraint satisfaction problems.

A Constraint Optimization Problem (COP) is a type of problem that involves finding the best possible solution while satisfying a set of constraints. It requires optimizing an objective function by adjusting variables within specified bounds, ensuring that all

constraints are satisfied. The goal is either to minimize or maximize the objective function, depending on the problem's requirements.

A "solver" in Constraint Programming applies various algorithms to systematically explore the search space defined by the constraints assigning values to the variables, propagating constraints, and backtracking when necessary. The solver's role is to efficiently navigate through the solution space, considering different combinations and configurations of variable assignments, until a valid solution is found or proven to be impossible [8].

2.2 Algorithm Portfolio

The term "Algorithm Portfolio" was first introduced by Huberman, Lukose, and Hogg in 1997, where they describe a strategy to execute several algorithms in parallel [4]. An algorithm portfolio refers to a collection of different algorithms that are selected and combined strategically to solve a particular problem or class of problems. Instead of relying on a single algorithm, an algorithm portfolio aims to leverage the strengths and weaknesses of multiple algorithms to improve the overall performance and robustness. Each algorithm in the portfolio may excel in specific situations, and by selecting the most suitable algorithm for a given problem instance, better results can be achieved. For this project, the algorithm portfolio consists of a set of Minizinc Solvers that will be evaluated according to their performance.

2.3 Machine Learning

Machine Learning (ML) refers to a collection of techniques designed to automatically identify patterns in data and utilize these patterns to make predictions or informed decisions in uncertain situations [7]. It is commonly divided into two main types: supervised learning and unsupervised learning. In supervised learning, which is the focus of this project, the training dataset includes a set of input features and their corresponding labels, enabling the model to learn the mapping between them and facilitating classification and prediction tasks.

In the context of Machine Learning, a classification problem assigns input data instances to predefined categories based on their features. The objective is to train a classification model that can accurately classify new, unseen instances into the correct categories. The training data for a classification problem consists of labeled examples, where each instance is associated with a known class label. In real-world scenarios, there is often limited knowledge about the relevant features, therefore, many candidate features are typically introduced, resulting in the presence of irrelevant and redundant features that can lead to overfitting in the training model [12]. A relevant feature is one that directly contributes to the target concept, while an irrelevant feature does not have a direct association with it, but still affects the learning process. A redundant feature does not provide any new information about the target concept [1]. Hence, it is crucial to select only the most informative features that provide relevant information for the specific problem at hand. The classification model establishes relationships between the selected features and the labeled examples to define decision boundaries that differentiate between classes, enabling accurate classification of unseen data.

Random Forest is an ensemble learning method that utilizes randomized decision trees. It creates a diverse set of classifiers by introducing randomness during the construction of each tree. The ensemble prediction is obtained by averaging the predictions of individual classifiers. The parameter "n_estimators" represents the number of decision trees included in the ensemble. It determines the size and complexity of the random forest model [9].

k-fold Cross Validation is a technique that addresses the problem of overfitting by splitting the data into subsets. The model is trained on a portion of the data, and its performance is evaluated on the remaining subset. This process is repeated multiple times, using different subsets for training and testing, to obtain a more robust estimation of the model's performance. The parameter "k" represents the number of subsets (folds) into which the data is divided for cross-validation, but each subset is used for both training and testing the model. The model is trained on k-1 subsets (i.e., using k-1 folds) and evaluated on the remaining 1 subset (i.e., using the last fold). This process is repeated k times, each time using a different subset as the evaluation set, until all subsets have been used as the evaluation set exactly once [10].

3 Diet Problem Model

Parameters:

- **n:** total amount of products available to buy ($n \in \mathbb{N}$).
- **budget:** budget available to make the purchase ($budget \in \mathbb{N}$).
- $\mathbf{groceries}$: array containing information about each product available for purchase. Each row contains information about a product, where the columns respectively represent: the amount of protein per unit, the amount of carbohydrates per unit, the amount of fat per unit, the amount available in inventory and the price per unit. Then $groceries_{p,d} \in \mathbb{N}$, where $p \in \{1, ..., n\}$ represents the index of each *product*, and $d \in \{1, ..., 5\}$ represents the *columns* mentioned above for each product.
- $\mathbf{requirements}$: array containing information on the nutritional requirements of the person or group of people. Each row represents one type of macronutrient: protein, carbohydrates, and fat. The first column represents the minimum quantity required per day and the second represents the maximum quantity required per day. Then $requirements_{m,l} \in \mathbb{N}$, where $m \in \{1, ..., 3\}$ represents each *macronutrient* and $l \in \{1, ..., 2\}$ represents the *columns* mentioned above for each macronutrient.
- $\mathbf{offset}$: array containing the values of the maximum deviation allowed for each macronutrient, will be used in the objective function. Then $offset_{m,l} \in \mathbb{N}$, where $m \in \{1, ..., 3\}$ represents the *macronutrient* and $l \in \{1, 2\}$ represents the lower and upper *offset* respectively.
- **variety:** maximum limit of units of the same product that can be purchased ($variety \in \mathbb{N}$).

Variables:

- $\mathbf{grocerylist}$: represents the shopping list, stores the number of units to be suggested for each product p ($grocerylist_p \in \mathbb{N}$).

- **acumprice**: represents the accumulated price of the shopping list. It will be used in the objective function ($acumprice \in \mathbb{N}$).
- **protein, carbo and fat:** they represent the amount of total protein, carbohydrate, and fat on the grocery list ($protein, carbo, fat \in \mathbb{N}$).
- **lackPro, lackCar and lackFat:** they represent the missing quantity of proteins, carbohydrates and fats necessary in the diet, for the requirements given according to the shopping list ($lackPro, lackCar, lackFat \in \mathbb{N}$).
- **excessPro, excessCar and excessFat:** they represent the excess quantity of proteins, carbohydrates and fats in the diet, for the requirements given according to the shopping list ($excessPro, excessCar, excessFat \in \mathbb{N}$).

Constraints:

- **Variety:** ensures that the number of units of each product on the shopping list does not exceed the value of *variety* (Eq. 1).

$$grocerylist_p \leqslant \text{variety}, \quad 1 \leq p \leq n \tag{1}$$

- **Protein, Carbohydrates and Fats:** constraints that calculate the amount of total protein (Eq. 2), carbohydrate (Eq. 3), and fat (Eq. 4) on the grocery list, respectively.

$$protein = \sum_{p=1}^{n} grocerylist_p * groceries_{p,1} \tag{2}$$

$$carbo = \sum_{p=1}^{n} grocerylist_p * groceries_{p,2} \tag{3}$$

$$fat = \sum_{p=1}^{n} grocerylist_p * groceries_{p,3} \tag{4}$$

- **Range:** these constraints establish the missing or excess amounts of each macronutrient in the diet, comparing the amounts calculated above with the dietary requirements defined in the *requirements* parameter. There are three constraints that measure deficiency (Eq. 5, Eq. 7, Eq. 9) and three that measure the excess (Eq. 6, Eq. 8, Eq. 10).

  - **Protein Limits:**

$$lackPro = \begin{cases} requirements_{1,1} - protein & \text{si } requirements_{1,1} > protein \\ 0 & \text{if not} \end{cases} \tag{5}$$

$$excessPro = \begin{cases} protein - requirements_{1,2} & \text{si } protein > requirements_{1,2} \\ 0 & \text{if not} \end{cases} \tag{6}$$

- **Carbohydrates Limits:**

$$lackCar = \begin{cases} requirements_{2,1} - carbo \text{ si } requirements_{2,1} > carbo \\ 0 \qquad\qquad\qquad\qquad \text{if not} \end{cases} \tag{7}$$

$$excessCar = \begin{cases} carbo - requirements_{2,2} \text{ si } carbo > requirements_{2,2} \\ 0 \qquad\qquad\qquad\qquad \text{if not} \end{cases} \tag{8}$$

- **Fats Limits:**

$$lackFat = \begin{cases} requirements_{3,1} - fat \text{ si } requirements_{3,1} > fat \\ 0 \qquad\qquad\qquad\qquad \text{if not} \end{cases} \tag{9}$$

$$excessFat = \begin{cases} fat - requirements_{3,2} \text{ si } fat > requirements_{3,2} \\ 0 \qquad\qquad\qquad\qquad \text{if not} \end{cases} \tag{10}$$

– **Offset:** these constraints ensure that the deficiency and excess values are within the minimum and maximum offset range allowed, with three restrictions for deficiency (Eq. 11, Eq. 13, Eq. 15) and three for excess (Eq. 12, Eq. 14, Eq. 14).

- **Protein Offset:**

$$lackPro \leq offset_{1,1} \tag{11}$$

$$excessPro \leq offset_{1,2} \tag{12}$$

- **Carbohydrates Offset:**

$$lackCar \leq offset_{2,1} \tag{13}$$

$$excessCar \leq offset_{3,2} \tag{14}$$

- **Fats Offset:**

$$lackFat \leq offset_{3,1} \tag{15}$$

$$excessFat \leq offset_{3,2} \tag{16}$$

– **Inventory:** ensures that the number of units of each product on the shopping list does not exceed the value of available units in inventory, which is defined in column 4 of the parameter $groceries_{p,d}$ (Eq. 17).

$$grocerylist_p \leqslant groceries_{p,4}, 1 \leq p \leq n \tag{17}$$

Objective: due to the nature of the problem, it was necessary to implement a multi-target feature. In this case, the weighted sum method was proposed, which has three objectives that must be **minimized**:

- **Lacks:** corresponds to minimum compliance with nutritional requirements. It is the relationship between the deficiencies of each macronutrient and the minimum allowable gap.
- **Excesses:** corresponds to maximum compliance with nutritional requirements. It is the relationship between the excesses of each macronutrient and the maximum allowable gap.
- **Budget:** corresponds to minimizing the accumulated cost of the chosen products. The relationship between the total price of food and the budget.

For the balancing of the weights in the weighted sum, the weights of all the objectives are equalized in an equitable relationship, in this case, the results obtained in each objective were transformed to a percentage scale so that they can be compared with each other.

For the lacks objective the energy consumption of each macronutrient deficiency is calculated, establishing a relationship with the minimum offset percentage allowed for each one, then the average is calculated by summing and dividing by 3, rounding off down since $mnt \in \mathbb{N}$ (Eq. 18).

$$lacks = \left\lfloor \frac{\left\lfloor \frac{lackPro*100}{offset_{1,1}} \right\rfloor + \left\lfloor \frac{lackCar*100}{offset_{2,1}} \right\rfloor + \left\lfloor \frac{lackFat*100}{offset_{3,1}} \right\rfloor}{3} \right\rfloor \tag{18}$$

For the **excesses objective**, a similar procedure is applied. The energy consumption of each macronutrient excess is calculated, establishing a relationship with the percentage of maximum offset allowed for each one, then the average is calculated by adding and dividing by 3, rounding down since $mnt \in \mathbb{N}$ (Eq. 19).

$$excesses = \left\lfloor \frac{\left\lfloor \frac{excessPro*100}{offset_{1,2}} \right\rfloor + \left\lfloor \frac{excessCar*100}{offset_{2,2}} \right\rfloor + \left\lfloor \frac{excessFat*100}{offset_{3,2}} \right\rfloor}{3} \right\rfloor \tag{19}$$

For the budget objective there is the variable $acumprice$, which represents the accumulated price for the purchase of all the products (Eq. 20). As with the previous objective. The percentage value is calculated with the total budget, rounding down since $mnt \in \mathbb{N}$ (Eq. 21).

$$acumprice = \sum_{p=1}^{n} grocerylist_p * groceries_{p,5} \tag{20}$$

$$bud = \lfloor \frac{acumprice}{budget} \times 100 \rfloor \tag{21}$$

The weight variables W_1, W_2, and W_3 are added as adjustable values to balance the previously established relationships (Eq. 22).

$$\text{mnt} = lacks \times W_1 + excesses \times W_2 + bud \times W_3 \tag{22}$$

Finally, the objective of the problem is defined as minimizing mnt (Eq. 23).

$$\textbf{solve minimize } mnt \tag{23}$$

4 Experiments

For the elaboration of this paper, a computer equipped with an AMD Ryzen™ 7 5800X processor was utilized, accompanied by 16 GB of RAM memory. The experimentation took place within an isolated environment employing Python 3.9.13 and Minizinc 2.7.4 which is a language for specifying constrained optimization and decision problems over integers and real numbers. A MiniZinc model does not dictate how to solve the problem, the MiniZinc compiler can translate it into different forms suitable for a wide range of solvers, such as Constraint Programming (CP), Mixed Integer Linear Programming (MIP) or Boolean Satisfiability (SAT) solvers [2]. Each test was conducted using instances generated by a synthetic instance generator (Sect. 4.1). The evaluation of the model, as described in Sect. 3, was performed using a search annotation provided by Minizinc (variable selection: **largest**, value choice: **indomain_max**) and with some solvers installed on Minizinc (Sect. 4.2).

The objective of this study is to observe the presence of patterns in the synthetic instances received by the Minizinc model, thereby justifying the employment of Machine Learning techniques to select the algorithm that offers the best solution in the shortest time [3].

4.1 Synthetic Instance Generation

Because the base Minizinc model requires data files to work, a synthetic instance generator written in Python was implemented. It provides a synthetic database that can be used to test the model. These files consist of random data that is consistent with nutritional inputs, requirements and budget constraints generated using real-world data from the Instituto Colombiano de Bienestar Familiar (ICBF) [5].

4.2 Algorithm Selection

In order to create the Algorithm Portfolio, a comparison was conducted among five solvers available for Minizinc. An initial set of tests were performed on multiple instances, wherein the value of *n* (number of products) and related constraints were varied, while maintaining a timeout of 5 s, in order to observe the behavior on five different solvers (i.e., reach the optimal solution within the cap time).

Table 1. Number of solved instances with a timeout of 5 s.

n	HiGHS	COIN-BC	OR-TOOLS	Gecode	Chuffed
5	100	100	87	6	18
20	100	98	17	1	0
50	100	98	11	0	2
100	100	92	10	0	0
500	99	94	3	0	0
1000	79	79	6	0	0
2000	27	85	0	0	0
Total	605	646	134	7	20

Table 1 contains the results of 700 experiments (100 per number of products). The values represent the number of instances an algorithm could solve in the cap time. OR-TOOLS, Gecode and Chuffed decreased their solving rate significantly as n increased, while HiGHS and COIN-BC won on the 86.42% and the 92.28% of the cases, respectively. As a result, the last two were selected as potential contenders for achieving the best solution within the optimal time limit.

alg_portfolio = [HiGHS, COIN-BC]

4.3 Creating the Training Dataset from Synthetic Instances

The chosen algorithms are evaluated within a maximum time limit of 5 s, using 1400 synthetic instances. which range from $10 \leq n \leq 1700$. The algorithm that successfully solves each instance with the best *mnt* solution and in the shortest time is labeled as the winner. Subsequently, a list called *labels* is generated, associating each instance with its respective winning algorithm. Once the labeling process finished, the list had an irregular distribution between both algorithms, where HiGHS won 49.21% of the time and COIN-BC, the 50.78%. Therefore, *undersampling* was applied to COIN-BC, resulting in a balanced dataset with an exact 50/50 distribution (i.e., 689 instances for each).

The *features* matrix was constructed with n rows and 45 columns, where each column represents statistical values of the numeric instances, translating each instance into its corresponding set of features (Appendix. B).

4.4 Exploration of Feature Selection in Machine Learning

For the feature selection, three main tools were used: the correlation matrix (Appendix A), a scatter plot analysis (Fig. 2), and our previous knowledge about the problem. These were employed to compare the interrelationships among all the available features and identify those that may need to be removed to enhance the performance of the model. In a correlation matrix, a value close to 1 means that the compared features are highly directly correlated. That's why features with values greater than 0.7 were

removed. The features left were compared into a scatter plot to find for similar patterns between them, discarding the ones with high correlation values.

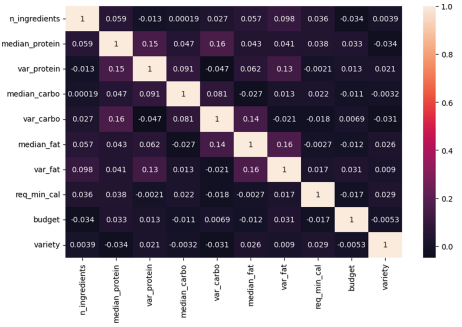


Fig. 1. Correlation matrix with selected features.

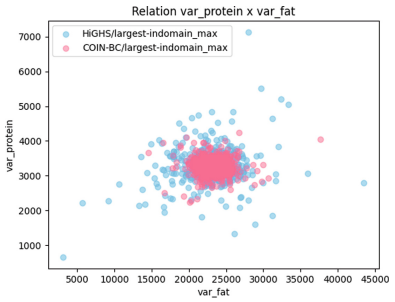


Fig. 2. Scatter plot of the relation between var_protein and var_fat.

In Fig. 2, it can be observed that COIN-BC shows a higher concentration in the values of protein and fat variations, while HiGHS exhibits a broader dispersion along these axes. In this particular case, we can infer that for an instance that falls outside the approximate range of $180,000 \leq var\_fat \leq 240,000$ and $25,000 \leq var\_protein \leq 39,000$, it is more likely to be solved using HiGHS. However, it is important to consider other features as well to obtain more precise predictions.

The presence or absence of remain features were discussed to make the necessary changes for the next iteration. By repeating this process, a lot of the initial features were removed progressively. Giving as a result the selected features to train the Machine Learning model: **n_ingredients, median_protein, var_protein, median_carbo, var_carbo, median_fat, var_fat, budget, variety**, as observed in the Correlation Matrix (Fig. 1).

4.5 Evaluating Model Precision

The training of the model was made using a Random Forest Classifier configured with a $n\_estimators$ value of 100 and a k-fold cross validation with a "k" value of 7, the selected features (Sect. 4.4) and labels (Sect. 4.3). The confusion matrix with the test group and a feature importance, were plot to analyze the results. The model was able to predict the algorithms that solve the problem more efficiently with an accuracy of 68.07%.

In the confusion matrix (Fig. 3), the vertical axis represents the actual values and the horizontal, the values predicted by the model. For HiGHS, 515 instances were predicted correctly, and 174 were wrongly given to COIN-BC; and for COIN-BC, 423 were predicted correctly while 266 were wrongly given to HiGHS.

While in the Feature Importance graph (Fig. 4), can be observed that the most influential features during the classification were "n_ingredients" and "req_min_cal", which can be analyzed in further feature selections to improve the model accuracy.

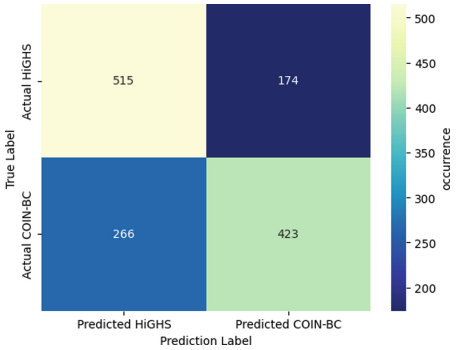


Fig. 3. Confusion matrix for the test group.

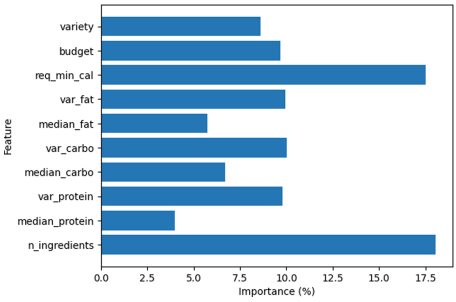


Fig. 4. Feature importance in prediction.

Finally, the results obtained are presented in a Classification Report (Table 2). This report provides an assessment of the model's performance by evaluating four key metrics. Precision, which represents the accuracy of positive predictions made by the model, was 0.66 for COIN-BC and 0.71 for HiGHS. Recall, which quantifies the model's ability to correctly identify all relevant positive instances, was 0.75 for COIN-BC and 0.61 for HiGHS. F1-score, a harmonic mean of precision and recall, was 0.70 for COIN-BC and 0.66 for HiGHS. Lastly, support, representing the number of instances of each class in the dataset, was 689 for both COIN-BC and HiGHS.

Table 2. Classification report for the test group.

	Precision	Recall	f1-score	Support
$COIN-BC/largest-indomain\_max$	0.66	0.75	0.70	689
$HiGHS/largest-indomain\_max$	0.71	0.61	0.66	689

5 Conclusions and Future Work

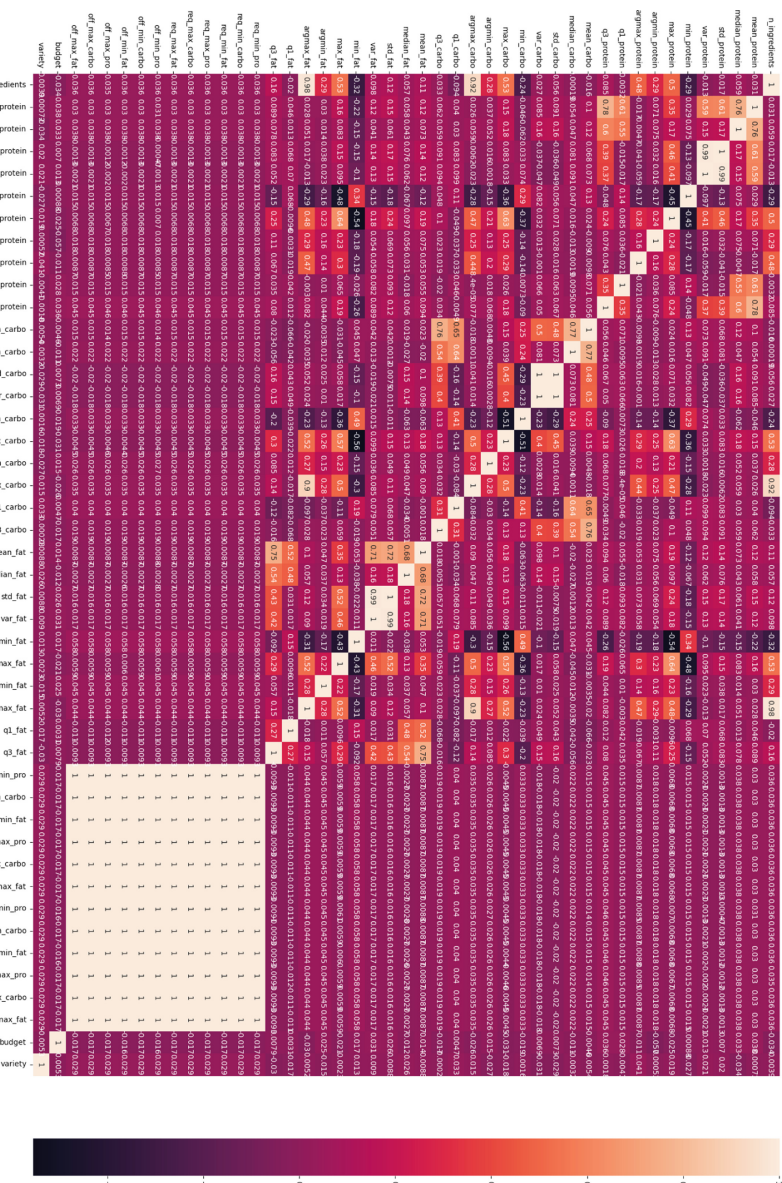
The existence of patterns that allow to predict the best solver to solve an instance has been evidenced and justifies the exploration of the Machine Learning approach to improve the precision even more. For this study, we applied our method using two solving algorithms, but allowing integration with additional solvers as they emerge, which holds significant importance for future experimentation and evaluation. The outcomes indicate potential for further improvement through tuning using feature selection algorithms for instance.

Given that the Diet Problem is an NP-Complete Problem, known for it's computationally intensive nature, this method offers a solution. Trough Machine Learning we were able to significantly reduce the computational time required to solve it. This approach may have the potential to optimize the NP-Complete problems solvable at a larger

scale. In the case of the Diet Problem, enabling the optimization of food distribution in real-world scenarios that were previously deemed infeasible.

Appendix

A Correlation Matrix (All Features)



B Candidate Features

- **n_ingredients:** Quantity of products.
- **mean_protein:** The mean protein value between between the products.
- **median_protein:** The median protein value between between the products.
- **std_protein:** The standard deviation protein value between between the products.
- **var_protein:** The variance protein value between between the products.
- **min_protein:** The minimum protein value between between the products.
- **max_protein:** The maximum protein value between between the products.
- **argmin_protein:** The position of the minimum protein value between between the products.
- **argmax_protein:** The position of the maximum protein value between between the products.
- **q1_protein:** The first quartile protein value between between the products.
- **q3_protein:** The third quartile protein value between between the products.
- **mean_carbo:** The mean carbohydrate value between between the products.
- **median_carbo:** The median carbohydrate value between between the products.
- **std_carbo:** The standard carbohydrate value between between the products.
- **var_carbo:** The variance carbohydrate value between between the products.
- **min_carbo:** The minimum carbohydrate value between between the products.
- **max_carbo:** The maximum carbohydrate value between between the products.
- **argmin_carbo:** The position of the minimum carbohydrate value between between the products.
- **argmax_carbo:** The position of the maximum carbohydrate value between between the products.
- **q1_carbo:** The first quartile carbohydrate value between between the products.
- **q3_carbo:** The third quartile carbohydrate value between between the products.
- **mean_fat:** The mean fat value between between the products.
- **median_fat:** The median fat value between between the products.
- **std_fat:** The standard fat value between between the products.
- **var_fat:** The variance fat value between between the products.
- **min_fat:** The minimum fat value between between the products.
- **max_fat:** The maximum fat value between between the products.
- **argmin_fat:** The position of the minimum fat value between between the products.
- **argmax_fat:** The position of the maximum fat value between between the products.
- **q1_fat:** The first quartile fat value between between the products.
- **q3_fat:** The third quartile carbohydrate value between between the products.
- **req_min_pro:** The minimum requirement of proteins.
- **req_min_carbo:** The minimum requirement of carbohydrates.
- **req_min_fat:** The minimum requirement of fats.
- **req_max_pro:** The maximum requirement of proteins.
- **req_max_carbo:** The maximum requirement of carbohydrates.
- **req_max_fat:** The maximum requirement of fats.
- **off_min_pro:** The minimum offset of proteins.
- **off_min_carbo:** The minimum offset of carbohydrates.
- **off_min_fat:** The minimum offset of fats.

- **off_max_pro:** The maximum requirement of proteins.
- **off_max_carbo:** The maximum requirement of carbohydrates.
- **off_max_fat:** The maximum requirement of fats.
- **budget:** The maximum budget.
- **variety:** The maximum variety.

References

1. Dash, M., Liu, H.: Feature selection for classification. Intell. Data Anal. **1**(1–4), 131–156 (1997)
2. Dekker, J.: Introduction to minizinc. https://www.minizinc.org/doc-2.7.2/en/intro.html
3. Dorado, J.J., Maradiago, S.J.: Menus. https://github.com/SJMC29/MENuS
4. Huberman, B.A., Lukose, R.M., Hogg, T.: An economics approach to hard computational problems. Science **275**(5296), 51–54 (1997). https://doi.org/10.1126/science.275.5296.51
5. ICBF: Resolución número 003803 de 2016 del ministerio de salud y protección social (2016)
6. Martos-Barrachina, F., Delgado-Antequera, L., Hernández, M., Caballero, R.: An extensive search algorithm to find feasible healthy menus for humans. Oper. Res. **22**, 1–37 (2022)
7. Murphy, K.P.: Machine Learning: A Probabilistic Perspective. MIT Press, Cambridge (2012)
8. Rossi, F., Van Beek, P., Walsh, T.: Handbook of Constraint Programming. Elsevier, Amsterdam (2006)
9. Scikit learn developers: 1.11. ensemble methods. Accessed May 2023. https://scikit-learn.org/stable/modules/ensemble.html#random-forests (2007-2023)
10. Scikit Learn developers: 3.1. cross-validation: evaluating estimator performance. Accessed May 2023. https://scikit-learn.org/stable/modules/cross_validation.html# (2007-2023)
11. Stigler, G.J.: The cost of subsistence. J. Farm Econ. **27**(2), 303–314 (1945)
12. Tang, J., Alelyani, S., Liu, H.: Feature selection for classification: a review. Data Classif. Algorithms Appl. 37 (2014)