Problem Chosen	2023 ShuWei Cup Summary Sheet	Team Control Number
D		2023091419459

Title:

Mathematical Modeling and Optimization of the Laundry Process: A Comprehensive Application of Genetic Algorithms, Linear Programming, and Backtracking Algorithms

Abstract:

In the first problem, we employed a genetic algorithm to find the most effective laundry strategy. This process included real-number encoding of each laundry plan, random generation of an initial population, and evaluation of each plan's effectiveness through a fitness function. We considered optimizing the number of washes and the amount of water used per wash, and delved into the impact of stain solubility rate, initial stain quantity, and available water on the washing effect. This method allowed us to precisely adjust washing parameters and analyze them comparatively to achieve the best cleaning efficiency and resource utilization.

In the second problem, our goal was to design a time-efficient laundry plan. Assuming equal washing time for each cycle and unlimited water resources, we thoroughly analyzed the impact of variations in stain solubility rate and initial stain quantity on the optimal washing strategy. Our model ensures that the final residual stain quantity is less than one-thousandth of the initial stain amount, thus proposing a time-saving and efficient washing method.

In the third problem, we utilized a linear programming model to optimize the selection and quantity of detergent. We defined decision variables and an objective function, aiming to minimize washing costs while considering the cleaning efficiency of the detergent. Through detailed data analysis of different stains on various garments, we were able to precisely specify the most suitable detergent and required quantity, achieving the best balance between cost-effectiveness and cleaning performance.

In the fourth problem, considering the mixed washing restrictions of different fabric materials, our goal was to provide a cost-effective and efficient cleaning solution. We analyzed the mixed washing relationships between different garments and used a backtracking algorithm to devise a comprehensive washing strategy that considered both cost and effective cleaning while protecting the garments.

Keywords :
Genetic Algorithm, Linear Programming, Data Analysis, Resource Management, Mixed Washing Limitations, Backtracking Algorithm, Cost-Benefit Analysis, Washing Efficiency

Content

1. Introduction

1.1 Background

Laundry cleaning is a daily activity for people. The stain-removing function of laundry detergent comes from certain surfactant chemicals. They enhance the permeability of water and utilize an intermolecular electrostatic repulsion mechanism to remove dirt particles. The mechanical action of a washing machine or hand friction leads to the displacement of dirt particles surrounded by surfactant molecules, with dirt particles adhering to the lipophilic part of the surfactant molecule. This causes dirt particles still suspended on the surface during the rinsing stage to be removed. In daily life, from small family operations to hotels and professional institutions, laundry cleaning is necessary. Finding ways to keep laundries clean and tidy at lower costs is a problem we need to study. Therefore, it is necessary to establish mathematical models to solve laundry problems and find the most optimal washing schemes.

1.2 Work

My team and I conducted in-depth research and applied multiple algorithms to solve complex problems in the laundry cleaning process. Firstly, we used a genetic algorithm to find the optimal cleaning strategy. This included defining a fitness function, initializing potential washing schemes, and optimizing these schemes through an iterative process, thus finding optimal solutions in terms of the number of washes and water usage. Next, we employed linear programming to maximize the efficiency of detergent use. By conducting data analysis and modeling for garments, stains, and detergents, we established a set of standards to ensure that when dealing with different types of garments and stains, the most suitable detergent is chosen, thereby minimizing overall costs. Finally, we applied a backtracking algorithm to solve the problem of mixed washing of garments. Through effective grouping of garments, we not only improved cleaning efficiency but also reduced potential damage to the garments.

2. Problem analysis

2.1 Analysis of question one

To find the optimal solution for cleaning stains, namely removing stains from clothes as much as possible under limited water resources, we chose the genetic algorithm. We first defined a fitness function to evaluate the effectiveness of the washing plans. Then, we initialized a series of potential washing plans as the population. Through their iterative optimization, we found an optimal solution, specifically the best washing plan regarding the number of washes and the amount of water used per wash. By changing the values of a_k , initial dirt value, and available water, we analyzed and compared each optimal solution, discussing the impact of each parameter on the optimal washing plan.

2.2 Analysis of question two

For the second question, with unlimited total water availability and equal washing times for each cycle, the final requirement is that the residual dirt amount does not exceed one-thousandth of the initial dirt amount. Other conditions are similar to the first problem, so the solubility is progressively decreasing. Considering that the solubility decreases proportionally, it will quickly diminish to a very small value, causing the dirt amount to decrease slowly. It's possible that the dirt amount may not converge to below one-thousandth of the initial amount. Therefore, we adjusted the initial solubility and the rate of solubility decay to explore and solve for the optimal washing plan.

2.3 Analysis of question three

The third question involves a variety of stains on multiple types of garments and the selection of various detergents. We needed to determine which detergent is most efficient for washing different stains on different garments, in order to minimize the total cost, including the cost of detergents and water resources. We first created three-dimensional charts for data analysis of the stains on the garments and the properties of various detergents. Then, we used a linear programming model to select and allocate detergents to achieve the best cleaning effect with limited resources. We calculated the required amount (in grams) of different detergents for different stains on various garments, aiming to minimize costs while ensuring cleaning quality.

2.4 Analysis of question four

The objective of the fourth question is actually quite similar to the third, focusing on cost optimization and reducing dirt to one-thousandth, but with added constraints. Not all garments can be mixed for washing, so we needed to group the garments for more efficient cleaning. We established adjacency matrices to solve for the grouping of garments, aiming to minimize the number of groups. Then, following the approach of the third question, we calculated the choice and cost of detergents for each group.

3. Symbol and Assumptions

3.1 Symbol Description

Table 1: Symbol Description

Serial number	symbol	significance	unit
1	**D**	**The amount of dirt on the original garment**	**g**
2	**W**	**Total water available**	**kg**
3	a_k	**The solubility of the stain in water at the k wash**	**g/kg**
4	x_k	**The amount of water used in the k wash**	**t**
5	**n**	**Number of washes**	**times**

3.2 Fundamental assumptions

1. The solubility of the detergent is not affected by external environmental factors such as water temperature. Every gram of detergent requires 1kg of water for dissolution.
2. The duration of each washing cycle is the same and is not influenced by the type or amount of detergent used.
3. The initial amount of dirt refers to the dirt amount before the first wash.
4. There are no quantity limitations for garments that can be mixed in the wash.
5. Detergents can be used in combination without any incompatibility affecting their performance, and the solubility of each detergent is independent of others.

4. Model

4.1 Genetic model

The genetic algorithm[1] is a method that seeks the optimal solution by simulating the natural evolutionary process. This algorithm, through mathematical means and computer simulation, transforms the problem-solving process into processes similar to the crossover and mutation of chromosome genes in biological evolution. In solving complex combinatorial optimization problems, it usually achieves better optimization results more quickly compared to some conventional optimization algorithms.

The genetic algorithm is a general algorithm for solving search problems and has the following characteristics: (1) It starts the search from a collection of strings, covering a large area, which is beneficial for global optimization; (2) It reduces the risk of being trapped in local optima and the algorithm itself is easy to parallelize;(3) It employs dynamic adaptive techniques to automatically adjust algorithm control parameters and coding accuracy during the evolutionary process.
The basic operation process of the genetic algorithm is as follows:

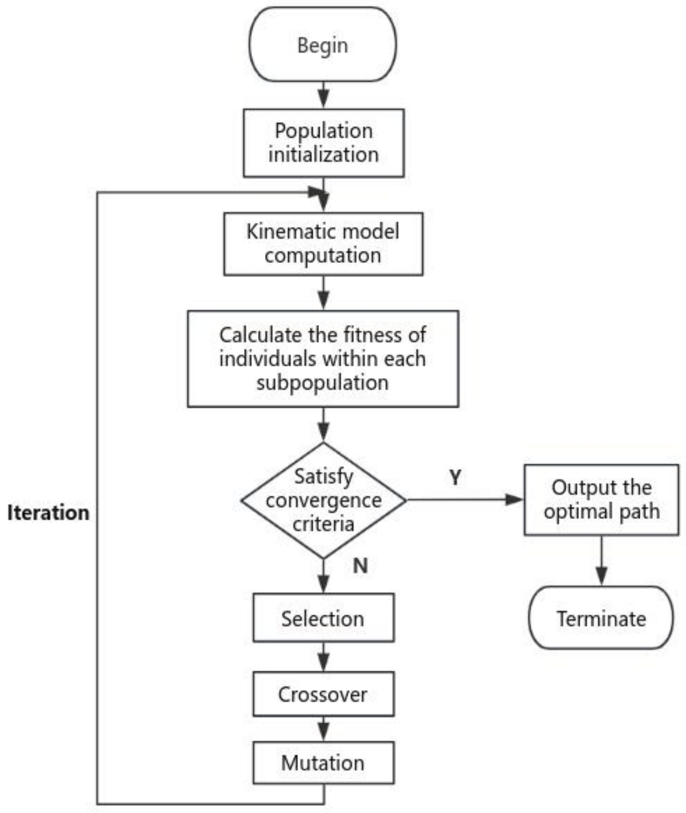

Begin
Population initialization
Kinematic model computation
Calculate the fitness of individuals within each subpopulation
Satisfy convergence criteria
Y
Output the optimal path
Terminate
N
Selection
Crossover
Mutation
Iteration

Figure 1: Genetic algorithm flow

5.Test the Models

5.1 Model construction and solution of problem 1

5.1.1 Model construction

1.Encoding

·**Real-number encoding:**in this problem, is more direct and effective than binary encoding. Each individual (washing scheme) can be represented as a real-number array of length n, where n is the maximum number of washes. Each element in the array,x_k,represents the amount of water used for the k-th wash.

2.Initial Population

·**Generate randomly:**Generate a certain number of washing schemes randomly as the initial population. It is essential to ensure that the total water volume for each scheme does not exceed the available water capacity, W.

3.Fitness Function

·**Fitness Calculation:** Calculate fitness based on the remaining dirt level or dirt removal rate. For example, if fitness is based on the remaining dirt level, then a lower remaining dirt level corresponds to a higher fitness.

·**Formula:** Assuming the initial dirt level is D, the remaining dirt level after the kth wash can be represented as:

$$D \times \prod_{i=1}^{k} (1 - a_i \times \frac{x_i}{W})$$

a_i is the stain solubility for the ith wash

x_i is the amount of water used for the ith wash

4.Selection

·**Roulette wheel selection:** Select individuals based on their fitness. Individuals with higher fitness have a greater probability of being chosen.

5.Crossover

·**Single-point crossover:** Randomly select a point, and exchange the encodings of two individuals from that point onward.

·**Pay attention to water volume constraints:** After crossover, adjustments may be needed to ensure that the total water volume for each scheme does not exceed W.

6.Mutation

·**Random mutation:** Randomly select certain genes of an individual (in this case, the water usage for a particular wash) and make small random adjustments.

·**Consider constraints:** Ensure that the total water volume after mutation does not exceed W.

7.Iteration

·**Repeat Execution:** Repeatedly perform selection, crossover, and mutation processes until a predetermined number of iterations are reached or fitness reaches a certain level. This is applicable in practical scenarios.

In practical applications, it's important to consider the following points:

·**Parameter tuning:** The performance of genetic algorithms heavily relies on parameter settings such as population size, crossover rate, mutation rate, etc.

·**Water volume constraints:** Throughout the entire process, always ensure that the total water volume for individual schemes remains within the specified limits.

·**Result validation:** The final optimal solution obtained should be validated to ensure its effectiveness in practical applications.

5.1.2 Problem solving

Under the conditions where the available water volume is set to 100g, the initial dirt level is 10g, a_0=0.8, a_k=0.5 × a_{k-1} , we can obtain the following results through a genetic model:

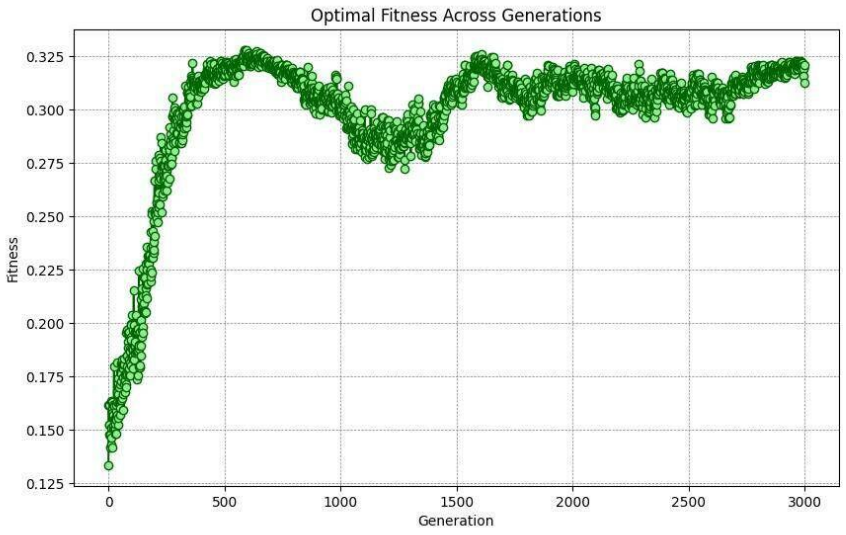


Figure 2: The Best Fitness Change Graph

This two-dimensional chart displays the variation in the best fitness for each generation. The two axes are as follows:

X-axis: Generation, representing the progress of the genetic algorithm.

Y-axis: Fitness, showing the fitness values of the best individual in each generation.

The lines in the graph illustrate how fitness changes over time. It is typically observed that fitness increases rapidly in the early stages of the algorithm and then gradually stabilizes. [4]The fluctuations in fitness may indicate a balance between exploration (introducing new genetic variations through mutation and crossover) and exploitation (selecting and retaining good solutions). Eventually, fitness reaches a relatively stable high value, which may suggest that the algorithm is approaching either a global optimum or a local optimum.

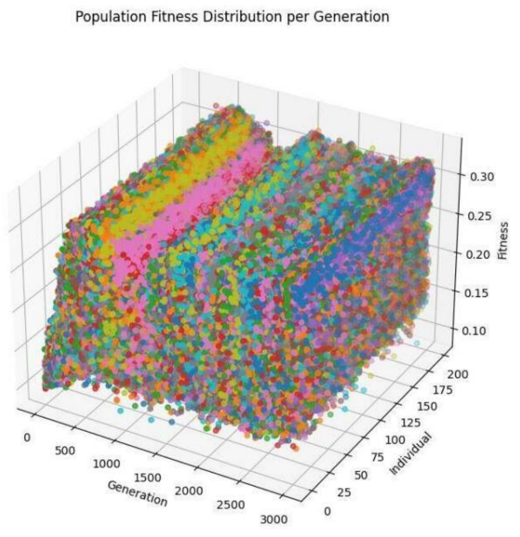


Figure 3: Population Fitness Distribution Graph

This three-dimensional scatter plot displays the fitness distribution of the population for each generation during the execution of the genetic algorithm. The three axes represent:

X-axis: Generation, showing the progress of the genetic algorithm.

Y-axis: Individual, representing each individual in the population.

Z-axis: Fitness, displaying the fitness values of each individual.

The colored points in the graph represent the fitness of different individuals, and the diversity in colors may indicate the diversity within the population. It can be observed that the fitness distribution is initially dispersed, but as the genetic algorithm progresses, fitness starts to cluster at higher values, indicating that the algorithm is finding better solutions. The overall fitness of the population increases with the increase in generation, which is a typical characteristic of the genetic algorithm optimization process.

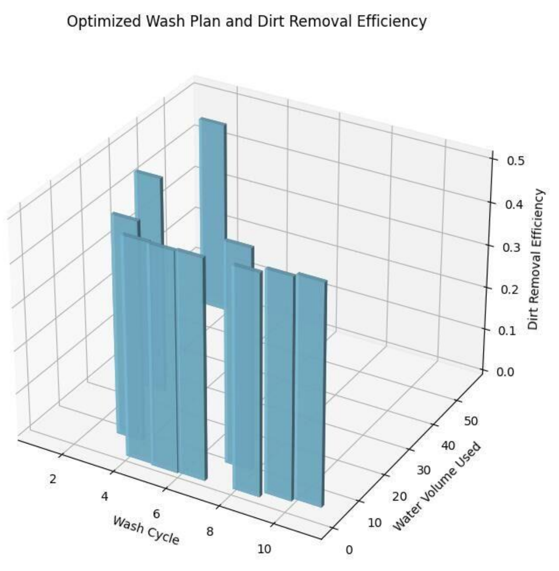


Figure 4: Three-dimensional diagram of optimal washing scheme
X-axis represents the number of washes.
Y-axis represents the amount of water used in each wash.
Z-axis represents the corresponding dirt removal rate.

Best Solution Water Distribution:

```
 [5.24893102e+01 1.94990016e+01
9.01415992e+00 8.01896872e-02
7.46526165e-02 9.49922394e-02
9.27290834e+00 8.58072088e-02
4.44616240e+00 4.94281574e+00]
```

Based on the data from our optimal washing scheme, we can see that the water usage for each wash varies. The first wash requires approximately 52kg of water, the second wash around 9kg, and so on. This indicates that the initial washes use the most water, and as the number of washes increases, the required water gradually decreases. Based on the relationship between the number of washes and the required water, we can conclude the optimal washing scheme:

Begin with using more water in the early washes, and then gradually reduce the water usage. Around the seventh wash, the reduction in dirt becomes minimal. Therefore, washing seven times is the optimal number of washes.

Based on the analysis and comparison of the results obtained by running the genetic algorithm, we can summarize the impact of various parameters on the objective:

(1) Solubility(a_k):As the solubility a_k decreases, the optimal number of washes increases. The initial dirt level has no effect on the number of washes. When the

available water volume is less than a critical value, a larger available water volume leads to a larger optimal number of washes. However, once the available water volume surpasses the critical value, the optimal number of washes remains constant (with the available water volume having no further impact).

(2) Available water volume: The following four graphs show available water volumes of 10, 100, 500, and 1000kg, with a dirt level of 10g. It is evident that with more water, the optimal number of wash cycles increases. When the water volume reaches a critical value (which appears to be between 100kg and 500kg in this case), the impact on the optimal number of cycles diminishes, and the number stabilizes. For example, the optimal wash cycle counts for 500kg and 1000kg are similar and higher than the counts for the lower water volumes.

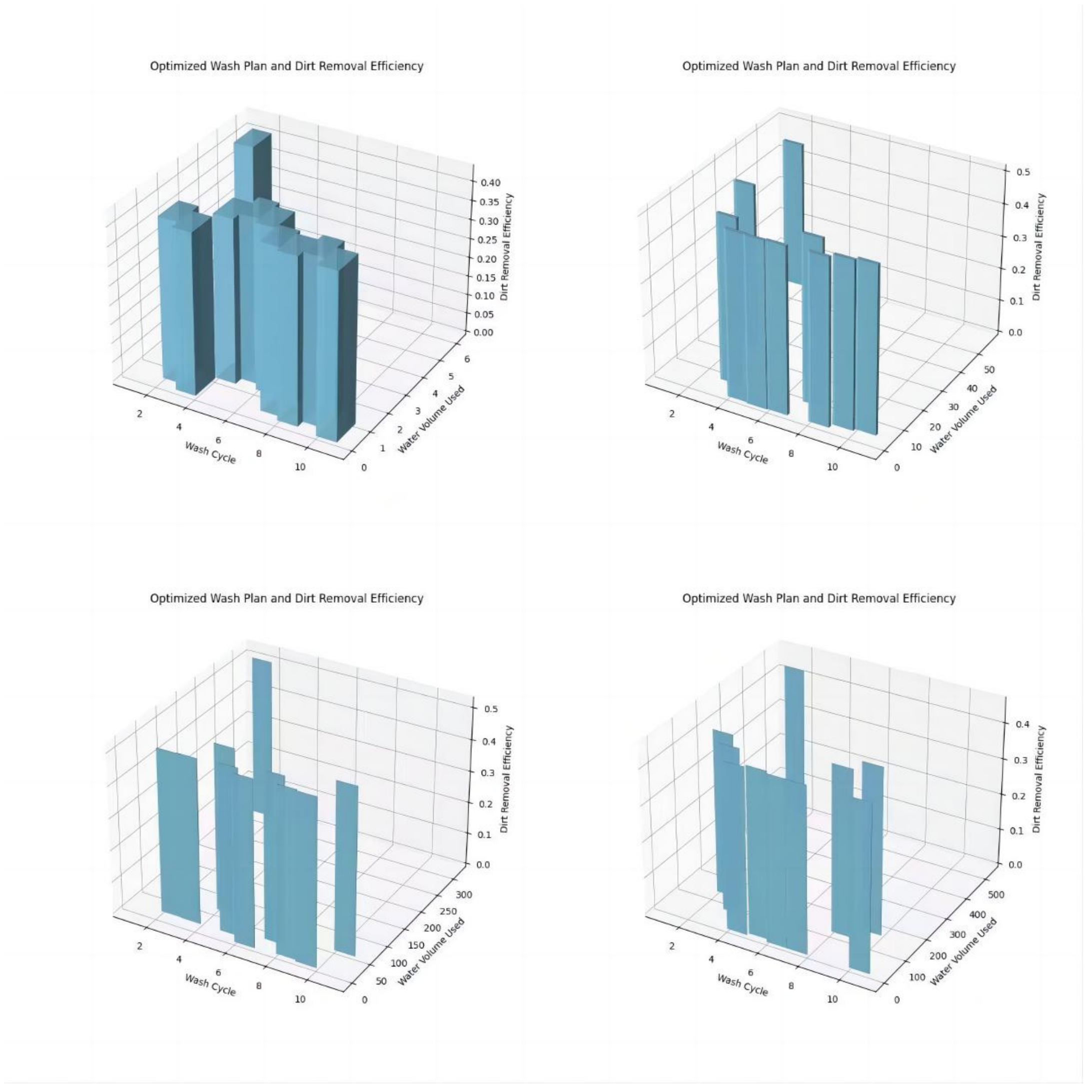


Figure 5: Comparison of water availability

(3) Initial dirt level: In the following four graphs, the initial dirt levels are 1g, 5g, 10g, and 50g, with an available water volume of 1000kg. It is evident that a higher initial dirt level makes washing more challenging, potentially requiring more wash cycles or a larger available water volume to achieve optimal results.

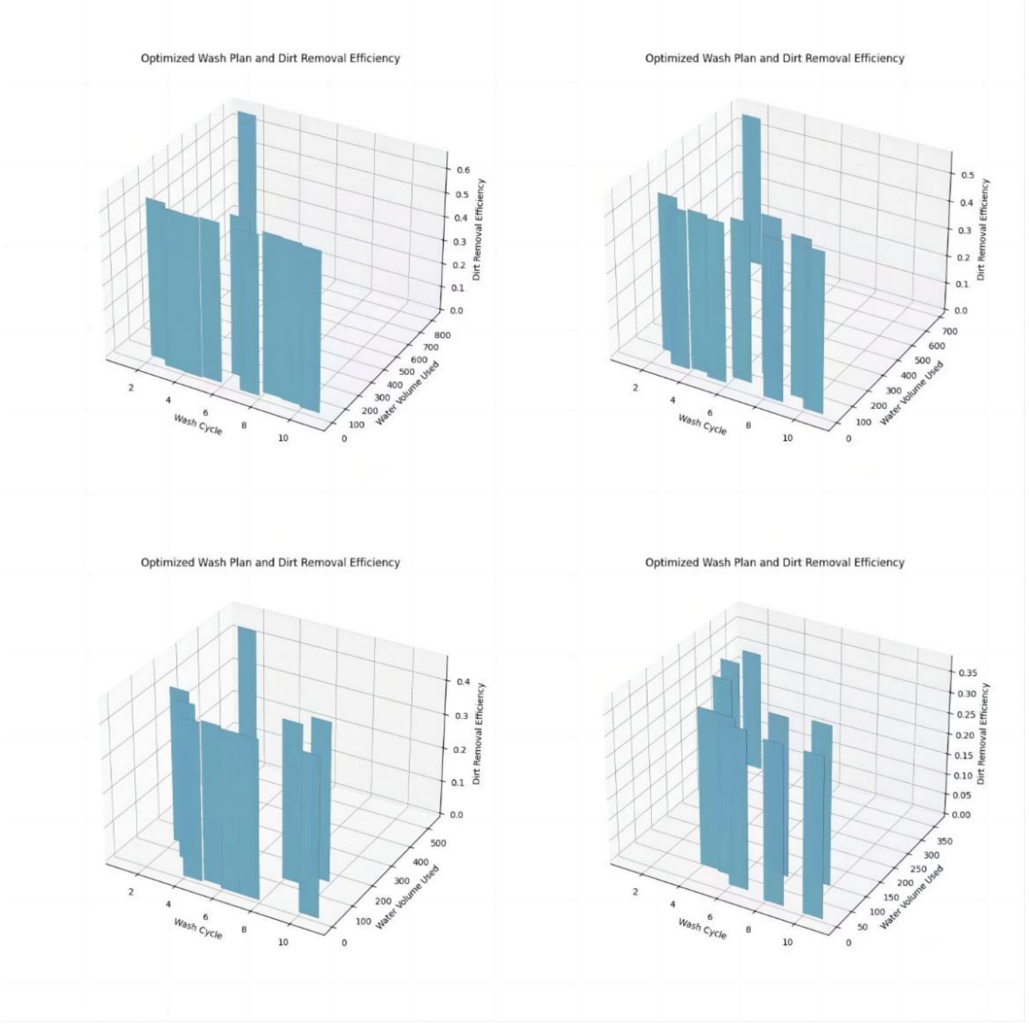


Figure 6: Comparison of initial dirt quantity

5.2 Model Construction and Solution for Problem Two

5.2.1 Model Construction and Solution

In Problem Two, there is no restriction on the available water volume, and each cleaning cycle has the same duration. The requirement is that the final residue of dirt should not exceed one-thousandth of the initial dirt amount, and the goal is to provide the most time-efficient cleaning solution. Other conditions are similar to those in Problem One, where the solubility of dirt gradually decreases. The model construction part is similar, but it discusses two main scenarios.

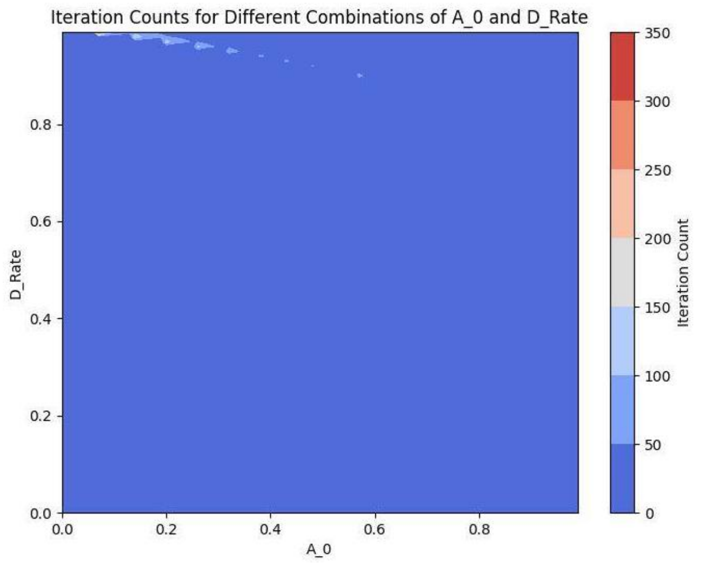
Scenario One: If the solubility formula remains the same as in Problem One

That is, $a_0=0.80$,$a_k=0.5a_{k-1}$,k=2,3,4.......In the program, we assumed $a_0=0.8$,$a_k=0.5\times a_{k-1}$,After calculating the residue of dirt through multiple cleaning cycles, it was found that the final dirt amount converges to around 8%, which does not meet the requirement of one-thousandth. Therefore, we need to adjust the initial solubility a_0 and the ratio of solubility reduction D_{Rate},and find the optimal combination that minimizes the number of cleaning cycles while achieving a residue of dirt less than one-thousandth.

Scenario Two: Considering a Change in Initial Solubility and Solubility Reduction Ratio

We conducted a programming search for different values of a_0 and D_{Rate} to find a combination that minimizes the number of cleaning cycles and performed the relevant discussion and analysis.

(1) First, by restricting the values of a_0 and D_{Rate} to the range [0,1], we conducted a traversal search and obtained the two-dimensional and three-dimensional graphs as well as the output results as follows:



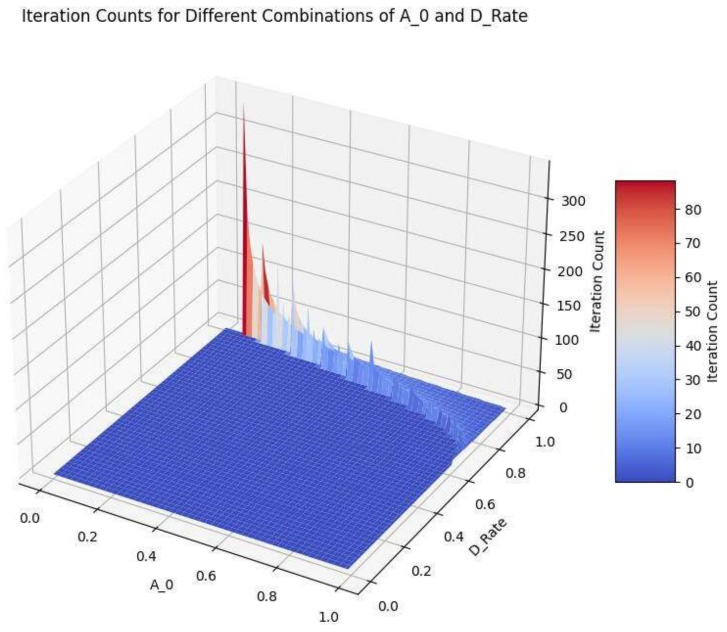


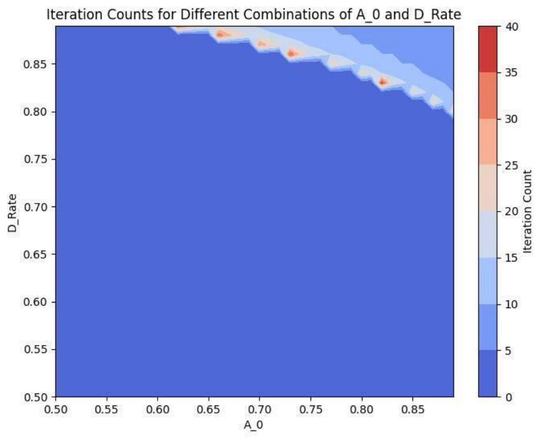
Figure 8:A_0 and D_Rate iterations (ranges [0,1])

Output is as follows:The minimum iteration count is 2, achieved with A_0 = 0.98 and D_Rate = 0.97.

The X-axis represents the initial solubility of dirt a_0 the Y-axis represents the reduction ratio of cleaning effectiveness D_{Rate} ,and the Z-axis represents the number of iterations.

Based on the results from the three-dimensional graph, it is evident that as long as D_{Rate} is sufficiently large and $a_0 > 0.06$,it is possible to reduce the remaining dirt to less than one-thousandth of the initial dirt amount after multiple cleaning cycles. However, it's essential to note that the minimum decay coefficient for solubility should also be at least 0.62; otherwise, it will be impossible to achieve a reduction to one-thousandth of the dirt amount.The maximum number of cleaning cycles is approximately 80, while the minimum number of cleaning cycles is 2. In the scenario where a_0 is set to 0.98,D_{Rate} is set to 0.97 ,only two cleaning cycles are required to reduce the dirt to one-thousandth of the initial amount. This represents a highly efficient cleaning solution.

(2) In the previous solution, the values of a_0 and D_{Rate} were relatively large. Therefore, we have narrowed down the range of a_0 and D_{Rate} to [0.5, 0.9] to generate new solutions. The two-dimensional and three-dimensional graphs resulting from the traversal search, along with the output results, are as follows:



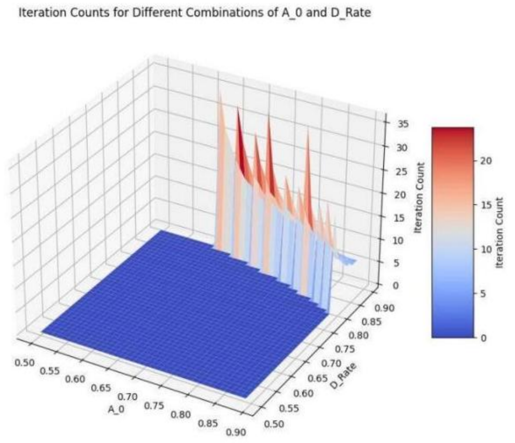


Figure 8: A_0 and D_Rate iterations (ranges [0.5,0.9])

Output is as follows:The minimum iteration count is 6, achieved with A_0 = 0.86 and D_Rate = 0.89.

According to the results from the three-dimensional graph, it can be observed that only when $a_0 > 0.6$ and $D_{Rate} > 0.8$, it is possible to reduce the cleaning dirt to one-thousandth of the initial dirt amount. The maximum number of cleaning cycles required is 34, while the minimum is 6. In the scenario where a_0 is set to 0.86, D_{Rate} is set to 0.89, only 6 cleaning cycles are needed to reduce the dirt to one-thousandth of the initial amount.

5.2.2 Discussion and Analysis of the Impact of a_k and Initial Dirt Amount on the Optimal Solution

After analyzing the results, we can summarize the impact of a_k and the initial dirt amount on the optimal solution as follows:

Solubility (a_k): The influence of initial solubility a_0 and the decay coefficient D_{Rate} of a_k on the optimal solution has been discussed in section 5.2.1 and will not be repeated here.

Initial Dirt Amount: It is evident that dirt is removed proportionally, so the initial dirt amount does not have an impact on the solution. It only affects the amount of water used in a single cleaning cycle. However, in this problem, the available water volume is unlimited, so it can be considered as having no impact on the solution.

5.3 Model Construction and Solution for Problem Three

5.3.1Data Analysis

First, we visualize and display the types and quantities of dirt on the clothing, as shown in Table 1:

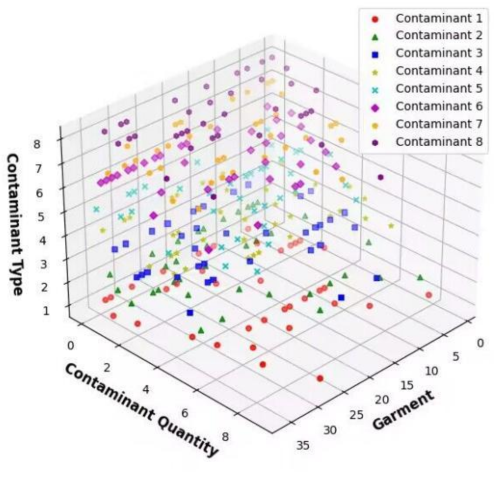


Figure 9: 3D diagram of the type and amount of dirt on clothing

Here are specific observations based on the results:

Some types of dirt (e.g., red circles, Type 1) appear on most clothing items, suggesting that Type 1 dirt is common.

Some types of dirt have a significantly higher quantity on specific clothing items, indicating that these clothing items may be used in specific environments or are more prone to contamination by certain types of dirt.

Clothing items with label numbers close to 36 appear to have a sparser distribution of dirt quantity and type, possibly because these items have less contact with sources of contamination or are washed more frequently.

Some types of dirt, such as purple diamonds (Type 6) and light blue stars (Type 4), are relatively less common. This suggests that these types of dirt are either less common in general or are easier to clean.

In summary, the chart displays the distribution and quantity of different types of dirt on various clothing items, providing data support for further cleaning strategies. For example, if a particular type of dirt is abundant, it may be necessary to develop specialized cleaning methods to effectively remove it. Additionally, it can help analyze the effectiveness of detergents, especially for stubborn types of dirt.

We also visualized the solubility of different types of dirt by detergents in Table 2:

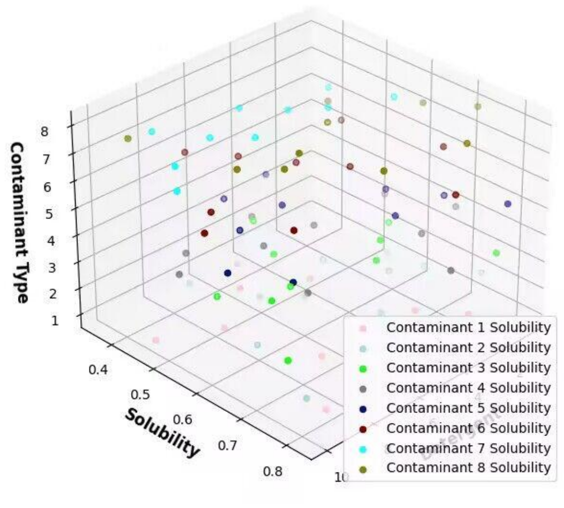


Figure 10: Three-dimensional diagram of the solubility of detergent to different dirt

Here are specific observations based on the results:

Differences in Detergent Efficacy: Different detergents show significant variations in cleaning efficacy for the same type of dirt. Some detergents exhibit higher solubility for certain types of dirt, while they perform less effectively for other types.

Hard-to-Clean Dirt: The solubility of dirt types 4 and 5 is generally low, indicating that these types of dirt may be challenging to remove.

Consistency in Certain Dirt Types: Dirt types 2 and 3 show relatively consistent solubility across different detergents, suggesting that most detergents have similar cleaning efficacy for these types of dirt.

Specialized Detergents: Dirt types 1 and 8 exhibit higher solubility values, indicating the presence of specific detergent formulations that are highly effective in handling these types of dirt.

Optimizing Detergent Selection: Based on solubility data, it is possible to choose the most effective detergent for specific types of dirt. This information can also guide detergent development to enhance the ability to clean stubborn dirt.

In summary, this chart provides valuable information for detergent selection and optimization of the washing process. It helps in choosing efficient detergents to effectively remove dirt from clothing items.

5.3.2 Model Construction and Solution

We use a linear programming approach[2] for the solution, and the model construction steps are as follows:

1. Define decision variables: x_{ij} , representing the amount of the ith detergent used to clean the jth type of dirt.

2. Formulate the objective function: The formula for minimizing the total cost is as follows:

$$\text{Min}\mathit{imize}\sum_{i=1}^{10}\sum_{j=1}^{8} x_{ij} \times y_i$$

y_i represents the unit price of the ith detergent

3.Design constraint functions to ensure that each type of dirt is cleaned sufficiently with an appropriate amount of detergent. For j types of dirt and k pieces of clothing, the formula is as follows:

$$\sum_{i=1}^{10} x_{ij} \times z_{ij} \geq p_{kj}$$

z_{ij} represents the solubility of detergent ifor dirt j,

p_{kj} represents the quantity of dirt j on clothing item k.

4.Consider other constraints: Ensure that all x_{ij} values must be non-negative.

5.Build the model: Define the problem, decision variables, objective function, and constraints using a linear programming library.

6.Data processing: Input the provided data into the linear programming solver to find the optimal solution.

7.Result analysis: Analyze the solution results to confirm that all constraints are satisfied, and the goal of minimizing costs has been achieved.

The results are as follows:

```
At line 2 NAME              MODEL
At line 3 ROWS
At line 293 COLUMNS
At line 6054 RHS
At line 6343 BOUNDS
At line 6344 ENDATA
Problem MODEL has 288 rows, 2880 columns and 2880 elements
Coin0008I MODEL read with 0 errors
Option for timeMode changed from cpu to elapsed
Presolve 29 (-259) rows, 58 (-2822) columns and 58 (-2822) elements
Perturbing problem by 0.001% of 0.09 - largest nonzero change 9.9661821e-07
( 0.0011073536%) - largest zero change 0
0   Obj 85.62306 Primal inf 191.11108 (29)
29   Obj 102.8232
Optimal - objective value 102.82306
After Postsolve, objective 102.82306, infeasibilities - dual 0 (0), primal 0 (0)
Optimal objective 102.8230596 - 29 iterations time 0.002, Presolve 0.00
Option for printingOptions changed from normal to all
Total time (CPU seconds):           0.03     (Wallclock seconds):           0.03

x_(2,_1,_1) = 10.38961
x_(2,_1,_10) = 6.4935065
x_(2,_1,_11) = 6.4935065
x_(2,_1,_12) = 7.7922078
x_(2,_1,_14) = 6.4935065
x_(2,_1,_15) = 3.8961039
......
x_(6,_8,_35) = 7.7922078
x_(6,_8,_36) = 5.1948052
x_(6,_8,_4) = 3.8961039
x_(6,_8,_5) = 2.5974026
x_(6,_8,_7) = 1.2987013
x_(6,_8,_9) = 5.1948052
```

Find the optimal solution (additional optimal solutions are available in the appendix).

From the results, it is evident that the linear programming model has successfully found the optimal solution.[5] The output consists of two main parts: log content and the optimal solution. Here is our analysis of these results:

(1) Log Content:

• Model Initialization: The model was successfully initialized with no errors.

Preprocessing: Some rows and columns of the model were simplified during preprocessing.

• Solving Process: The model went through a certain number of iterations to find the optimal solution.

• Final Objective Value: The objective function value of the optimal solution is 102.82306 yuan.
• Solution Time: The entire solving process took a very short amount of time, indicating the efficiency of the linear programming solver.
(2) Optimal Solution:
The values of decision variables in the output represent the quantities of each detergent used for each type of dirt and each clothing item in the optimal solution. For example, "x_(6,_8,_9) = 5.1948052" indicates that 5.1948052 grams of the 6th detergent are used to clean dirt type 8 on the 9th clothing item.

The advantages of this optimal solution include:
• Cost Minimization: The obtained optimal solution ensures cost minimization while cleaning the clothing.
• Detergent Allocation: The results provide a strategy for detergent usage, which can guide actual washing operations to ensure cost-effectiveness.
• Operational Guidance: These values of decision variables can serve as specific guidelines for the amount of detergent to use during the washing process.

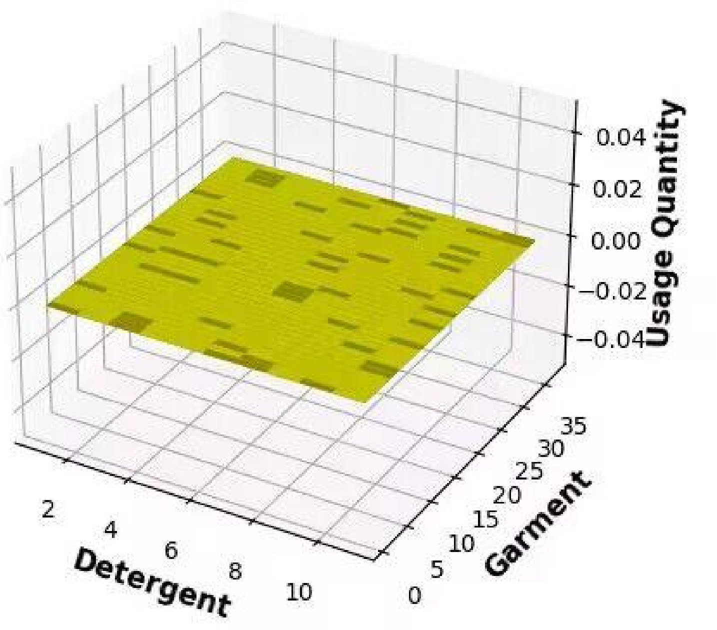


Figure 11: Three-dimensional diagram of the optimal solution
Conclusion:
By constructing a linear programming model, an optimized washing plan with both cost-effectiveness and cleaning efficiency has been obtained. This plan specifies which detergent and quantity should be used for each type of dirt on each clothing item. Such analysis and results are of practical significance for washing plans.

5.4 Model Construction and Solution for Problem 4

Data Processing:

In Table 2, a total of eight types of clothing are provided. Based on the differences in material, color, and other aspects of clothing as shown in Table 4, we can summarize the mixed washing relationships for the eight types of clothing as shown in the table below:

Table 2: Clothing Mixing Restrictions

Materials	1	2	3	4	5	6	7	8
1	√	×	√	√	√	√	×	×
2	×	√	×	×	×	×	×	√
3	√	×	√	×	×	√	√	√
4	√	×	×	√	×	√	√	√
5	√	×	×	×	√	×	√	√
6	√	×	√	√	×	√	√	√
7	×	×	√	√	√	√	√	√
8	×	√	√	√	√	√	√	√

Materials represented by 1-8 are as follows: Material 1 - Cotton, Material 2 - Silk, Material 3 - Cotton Linen Fabric, Material 4 - Polyester, Material 5 - Pure Wool, Material 6 - Cotton Polyester, Material 7 - Wool Polyester, Material 8 - Synthetic Silk.

The symbol √ indicates that mixing washing is allowed, and × indicates that mixing washing is not allowed.

Summarize the content of the table above into a matrix, where 0 represents that clothing cannot be mixed, and 1 represents that clothing can be mixed. The row and column numbers represent clothing of different materials. The matrix content is as follows:

$$A = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

We need to classify clothing based on the matrix above so that the clothing in each group can be mixed for washing. To obtain a more efficient cleaning plan, we should minimize the total number of clothing groups, and each group should contain as many clothes as possible. This problem can be seen as a maximum clique problem in graph theory. The matrix can be viewed as an adjacency matrix, where the rows and columns of the matrix represent vertices in the graph (here, different materials of clothing), and the elements in the matrix indicate whether these vertices (clothing) are connected (i.e., can be grouped together).

To find the minimum number of combinations, we can attempt to find a maximum clique partition of the graph. A clique is a subgraph in which all vertices are mutually connected. Maximum clique partitioning refers to dividing the vertices of the graph into as few cliques as possible, with each vertex belonging to a clique, and all vertices within each clique are mutually connected. This problem is NP-complete, and there is no efficient algorithm that can solve it in polynomial time for large graphs. However, since the graph here is small-scale (only 8 vertices), we can attempt a direct calculation.

We used a backtracking algorithm to find all possible combinations and then select the optimal combination. Specifically, we first created a recursive function that tries to place the current clothing item into an existing group or create a new group at each step, until all items are placed. In the end, we found the minimum number of groups to be 4, and one possible grouping is as follows:

Table 3: Grouping of Clothing for Mixed Washing

Groups	Types of Clothing Included
Combination1	1,3,6
Combination2	2,8
Combination3	4,7
Combination4	5

Symbol Definitions:

Let x_{ij} represent the quantity of the j-th type of contamination on the i-th clothing item, where i = 1,...132 and j = 1,...8.

Sum up all clothing items for each type of contamination to obtain the total amount d_j of different types of contaminants, where j = 1,2,...8.

Let w_k be the amount of the k-th detergent added during washing, where k = 1,2,...10.

Let z_{kj} represent the cleaning effect of the k-th detergent on the j-th type of contamination.

Let p_k represent the amount of the k-th detergent as w_k, where k = 1,2,...,10.

Let z_{kj} represent the cleaning effect of the k-th detergent on the j-th type of contamination.

Let p_k be the unit price of the k-th detergent, and w_k can represent the available water quantity.

Let D_j represent the total remaining contamination for each type of contamination.

Let P be the final washing cost.

Model Construction:

Based on the conditions mentioned above, we can establish the following multi-objective programming model[3]:

(1) Decision Variables: w_k - the quantity of the k-th detergent added during washing

(2) Objective Function:

$$\min D_j = d_j - \sum_{k=1}^{10} w_k \cdot z_{kj}$$

$$\min P = \sum_{k=1}^{10} \left(w_k p_k + 3.8 \cdot w_k \right)$$

(3) Constraints:

$$\mathrm{w}_k \geq 0, k = 1,2K10$$

For the solution to this problem, we have used the ε-constraint method. The ε-constraint method is a multi-objective optimization algorithm used to solve optimization problems with multiple decision variables and multiple objective functions. Its basic idea is to transform a multi-objective problem into a series of single-objective subproblems by introducing the parameter ε to control the weight relationships between objective functions. In the ε-constraint method, the optimization direction of each objective function, whether it is minimization or maximization, needs to be determined first. Then, each objective function is transformed into a constraint, and the constraints are converted into constraints on the objective functions by introducing the ε parameter. By continuously adjusting the ε value, a series of solution sets can be obtained when solving the subproblems, representing the optimal solutions under different optimization directions of the objective functions. Finally, through the analysis and decision-making on these solution sets, the global optimal solution or a set of effective non-dominated solution sets can be obtained.

According to the requirements of the problem, the main objective of finding a cleaning plan is to save costs. Therefore, the final washing cost P is chosen as the primary objective of interest. The total remaining pollutant amount D_j is included in the constraints, and ε is set to one-thousandth. The final optimization model is as follows:

$$\min P = \sum_{k=1}^{10} \left(w_k p_k + 3.8 \cdot w_k \right)$$

$$s.t = \begin{cases} D_j = d_j - \sum_{k=1}^{10} w_k \cdot z_{kj} \leq 0.001d, j = 1,2,K8 \\ w_k \geq 0, k = 1,2K10 \end{cases}$$

Solving combinations 1 to 4 separately yields the following table of results:

Table 4: Optimal washing scheme table

	Combination1	Combination2	Combination3	Combination4	Total
Detergent1	72.10	61.68	310.70	123.51	567.99
Detergent2	0	0	0	0	0
Detergent3	188.42	157.93	0	0	346.35
Detergent4	0	0	0	0	0
Detergent5	0	0	0	0	0
Detergent6	0	0	0	0	0
Detergent7	0	0	0	0	0
Detergent8	0	0	0	0	0
Detergent9	0	0	0	0	0
Detergent10	0	0	0	0	0
Minimum Cost	1028.7	867.3	853.3	500.23	3249.53

The final washing plan involves dividing all the clothes into four groups for washing:

Group 1:Washing cotton,cotton linen,and cotton polyester clothes with a min-cost of 1028.7 CNY, using 72.10g of detergent 1 and 1,188.42g of detergent 3.

Group 2: Washing silk and artificial silk clothes with a minimum cost of 867.30 CNY, using 61.68g of detergent 1 and 1,157.93g of detergent 3.

Group 3: Washing polyester and wool-polyester clothes with a minimum cost of 853.30 CNY, using 310.7g of detergent 1.

Group 4: Washing pure wool clothes with a minimum cost of 500.23 CNY, using 123.51g of detergent 1.

The total cost is 3249.53 CNY, with a total detergent usage of 567.99g of detergent 1 and 1,346.35g of detergent 3.

6.Strengths and Weakness

Problem One: Using Genetic Algorithms to Solve the Optimal Stain Cleaning Solution

Advantages:

Global Search Capability: Genetic algorithms are suitable for finding globally optimal washing solutions, making them effective for handling complex laundry problems that involve multiple stains and detergent combinations.

Strong Adaptability: They can dynamically adjust search strategies to accommodate diverse laundry requirements.

Disadvantages:

Encoding and Constraint Issues: Genetic algorithms may encounter difficulties in accurately representing all laundry constraints in the problem.

Computation Time: For complex washing scenarios, genetic algorithms may require longer computation times, impacting quick decision-making.

The advantages and disadvantages of Problem Two are similar to Problem One, as they also involve global search for washing solutions and constraint representation.

Problem Three: Using Linear Programming Models to Optimize Detergent Allocation

Advantages:

Cost Efficiency: Linear programming models effectively minimize costs, a critical consideration in the laundry business.

Clear Operational Guidelines: The model provides specific detergent usage strategies, aiding in guiding actual laundry operations.

Disadvantages:

Parameter Sensitivity: In practical laundry settings, the model's effectiveness may be limited by the accuracy of parameter settings, such as detergent concentrations and stain levels on clothing.

Linear Constraints: Some non-linear factors in the washing process may not be fully accounted for.

Problem Four: Using Backtracking Algorithms to Handle Clothing Mixing Constraints

Advantages:

Comprehensiveness: For small-scale clothing mixing problems, backtracking algorithms can ensure consideration of all possible combinations, resulting in the optimal mixing solution.

Disadvantages:

Scale Limitations: When dealing with a large number of clothing items for mixing, the efficiency of backtracking algorithms may significantly decrease, making them unsuitable for large-scale commercial laundry scenarios.

Practicality: In real-world operations, simplified methods may be needed for quick decision-making, as backtracking algorithms can be overly complex and time-consuming.

7.Conclusion

In this paper, we successfully apply a variety of algorithms to optimize the laundry process. Using genetic algorithms, we found the most effective cleaning strategy, taking into account the number of washes and water consumption. Linear programming helps us minimize detergent costs while ensuring cleaning efficiency. The backtracking algorithm effectively solves the problem of laundry mixing and improves the overall washing efficiency. Our research not only demonstrates the powerful capabilities of these algorithms in theory, but also provides practical solutions for practical laundry scenarios. Our model emphasizes the importance of cost-effectiveness and resource optimization to provide value to the washing business. In the future, we plan to further explore the application of these algorithms in larger and more complex scenarios, as well as consider more constraints and variables in real-world operations.

References

[1] Holland J H .Adaptation In Natural And Artificial Systems[J]. 1975.

[2] Liu Shan,Zhang Linling,Hao Lidong,.Continuous solution method for 0-1 linear programming[J]. Journal of Civil Aviation University of China,2013,31(3):45-49.

[3] Gao Weifeng,Liu Lingling,Wang Zhenkun,. A review of evolutionary multi-objective optimization algorithms based on decomposition[J]. Journal of software,2023,34(10):4743-4771.

[4] Li Wenye,Ming Mengjun,Zhang Tao,. Multimodal multi-objective optimization algorithm considering global and local Pareto frontiers[J]. Acta Automatica Sinica,2023,49(1):148-160.

[5] Min Xin. Application of linear programming to profit maximization and cost minimization problems[J]. Heilongjiang Science and technology Information,2013(21):125-126.

Appendix

#Question 1 code:

```
import numpy as np
import matplotlib.pyplot as plt


group_size = 200
generation_limit = 3000
pairing_chance = 0.8
mutation_likelihood = 0.01
total_water = 100
initial_dirt = 10
max_wash_cycles = 10



def evaluate_fitness(specimen):
    dirt_residual = initial_dirt
    for cycle_idx, water_vol in enumerate(specimen):
        solubility_coefficient = 0.8 * (0.5 ** cycle_idx)
        dirt_residual *= (1 - solubility_coefficient * (water_vol / total_water))
    return 1 / (dirt_residual + 1)



def create_population():
    individuals = []
    for _ in range(group_size):
        specimen = np.random.uniform(low=0, high=total_water,
size=max_wash_cycles)
        specimen /= specimen.sum() / total_water
        individuals.append(specimen)
    return individuals



def perform_selection(colony):
    fitness_scores = [evaluate_fitness(specimen) for specimen in colony]
    overall_fitness = sum(fitness_scores)
    selection_probabilities = [score / overall_fitness for score in fitness_scores]
    chosen_index = np.random.choice(range(group_size), p=selection_probabilities)
    return colony[chosen_index]


def execute_crossover(parent_a, parent_b):
    if np.random.rand() < pairing_chance:
        crossover_point = np.random.randint(1, len(parent_a) - 1)
        offspring_a = np.concatenate((parent_a[:crossover_point], parent_b[crossover_point:]))
        offspring_b = np.concatenate((parent_b[:crossover_point], parent_a[crossover_point:]))
        return offspring_a, offspring_b
    return parent_a, parent_b


def apply_mutation(specimen):
    for gene_idx in range(len(specimen)):
        if np.random.rand() < mutation_likelihood:
            specimen[gene_idx] = np.random.uniform(low=0, high=total_water)
    specimen /= specimen.sum() / total_water    # 保证水量总和符合限制
    return specimen


def run_genetic_algorithm():
    optimal_fitness_each_gen = []
    colony = create_population()
    all_fitnesses = []

    for gen in range(generation_limit):
        new_generation = []
        fitness_scores = [evaluate_fitness(specimen) for specimen in colony]
        all_fitnesses.append(fitness_scores)

        for i in range(0, group_size, 2):
            progenitor_a, progenitor_b = perform_selection(colony), perform_selection(colony)
            descendant_a, descendant_b = execute_crossover(progenitor_a, progenitor_b)
            new_generation.extend([apply_mutation(descendant_a), apply_mutation(descendant_b)])

        colony = new_generation
        fittest_individual = max(colony, key=evaluate_fitness)
        optimal_fitness_each_gen.append(evaluate_fitness(fittest_individual))

        if gen % 10 == 0 or gen == generation_limit - 1:
            print(f"Generation {gen}: Optimal Fitness =
{evaluate_fitness(fittest_individual)}")

    return max(colony, key=evaluate_fitness), optimal_fitness_each_gen,
all_fitnesses



best_solution, optimal_fitness_track, all_fitnesses = run_genetic_algorithm()
print("Best Solution Water Distribution:", best_solution)


plt.figure(figsize=(10, 6))
plt.plot(optimal_fitness_track, marker='o', linestyle='-', color='darkgreen',
markerfacecolor='lightgreen')
plt.title('Optimal Fitness Across Generations')
plt.xlabel('Generation')
plt.ylabel('Fitness')
plt.grid(True, linestyle='--', linewidth=0.5, color='grey')
plt.show()


fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')
for generation in range(generation_limit):
    fitnesses = all_fitnesses[generation]
    ax.scatter([generation] * group_size, list(range(group_size)), fitnesses)
ax.set_xlabel('Generation')
ax.set_ylabel('Individual')
ax.set_zlabel('Fitness')
plt.title('Population Fitness Distribution per Generation')
plt.show()


removal_efficiency = []
dirt_left = initial_dirt
for cycle_num, water_amt in enumerate(best_solution):
    removal_factor = 0.8 * (0.5 ** cycle_num)
    dirt_left *= (1 - removal_factor * (water_amt / total_water))
    removal_rate = 1 - dirt_left / initial_dirt
    removal_efficiency.append(removal_rate)


fig, ax3d = plt.subplots(figsize=(12, 8), subplot_kw={'projection': '3d'})
wash_cycle_nums = range(1, len(best_solution) + 1)
ax3d.bar3d(wash_cycle_nums, best_solution, np.zeros(len(best_solution)), 1, 1,
removal_efficiency, color='skyblue',
            alpha=0.8)

ax3d.set_xlabel('Wash Cycle')
ax3d.set_ylabel('Water Volume Used')
ax3d.set_zlabel('Dirt Removal Efficiency')
plt.title('Optimized Wash Plan and Dirt Removal Efficiency')
plt.show()
```

#Question 2 code:

```
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import Axes3D

max_iterations = 1000
initial_density = 1.0

# alpha_values = np.arange(0, 1, 0.01)
# mu_values = np.arange(0, 1, 0.01)
alpha_values = np.arange(0.5, 0.9, 0.01)
mu_values = np.arange(0.5, 0.9, 0.01)
iteration_counts = np.zeros((len(alpha_values), len(mu_values)))

smallest_iteration = np.inf
optimal_alpha = None
optimal_mu = None

for i, alpha in enumerate(alpha_values):
    for j, mu in enumerate(mu_values):
        density_iterations = np.zeros(max_iterations)
        density_iterations[0] = initial_density
        for iteration in range(1, max_iterations):
            alpha_iteration = alpha * (mu ** (iteration - 1))
            density_iterations[iteration] = (1 - alpha_iteration) * density_iterations[iteration - 1]
            if density_iterations[iteration] < 0.001:
                iteration_counts[i, j] = iteration
                if iteration < smallest_iteration:
                    smallest_iteration = iteration
                    optimal_alpha = alpha
                    optimal_mu = mu
                break

alpha_mesh, mu_mesh = np.meshgrid(alpha_values, mu_values)


colormap_choice = 'coolwarm'

# 2D plot
plt.figure(figsize=(8, 6))
plt.contourf(alpha_mesh, mu_mesh, iteration_counts.T, cmap=colormap_choice)
plt.colorbar(label='Iteration Count')
plt.xlabel('A_0')
plt.ylabel('D_Rate')
plt.title('Iteration Counts for Different Combinations of A_0 and D_Rate')
plt.show()

# 3D plot
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
surface_plot = ax.plot_surface(alpha_mesh, mu_mesh, iteration_counts.T, cmap=colormap_choice)

fig.colorbar(surface_plot, shrink=0.5, aspect=5, label='Iteration Count')
ax.set_xlabel('A_0')
ax.set_ylabel('D_Rate')
ax.set_zlabel('Iteration Count')
ax.set_title('Iteration Counts for Different Combinations of A_0 and D_Rate')
plt.show()


print(f"The minimum iteration count is {smallest_iteration}, achieved with A_0 = {optimal_alpha} and D_Rate = {optimal_mu}.")
```

#Question 3 code:

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd


data_garments = {
    'Garment': np.arange(1, 37),

    'Contaminant 1': [8, 3, 2, 2, 2, 0, 2, 1, 7, 5, 5, 6, 0, 5, 3, 0, 5, 5, 1, 2, 6, 5, 0, 1, 5,
1, 9, 6, 0, 7, 5, 4, 0,
                      0, 2, 1],
    'Contaminant 2': [5, 2, 8, 0, 0, 0, 7, 0, 2, 6, 0, 2, 6, 2, 0, 0, 6, 1, 7, 0, 5, 2, 4, 1, 0,
1, 2, 2, 3, 5, 2, 2, 0,
                      5, 1, 2],
    'Contaminant 3': [2, 4, 2, 5, 4, 0, 4, 4, 0, 4, 5, 8, 5, 3, 0, 0, 0, 2, 8, 1, 4, 0, 1, 2, 0,
3, 3, 4, 4, 0, 3, 0, 2,
                      2, 2, 5],
    'Contaminant 4': [4, 5, 1, 3, 7, 6, 5, 3, 5, 2, 0, 3, 4, 1, 0, 4, 5, 3, 2, 0, 0, 4, 5, 5, 5,
3, 2, 2, 0, 0, 3, 3, 4,
                      0, 4, 3],
    'Contaminant 5': [3, 0, 2, 2, 1, 5, 2, 6, 3, 2, 2, 4, 2, 2, 4, 0, 2, 4, 3, 4, 2, 5, 1, 3, 6,
0, 5, 0, 0, 5, 7, 1, 6,
                      1, 0, 4],
    'Contaminant 6': [1, 0, 2, 1, 3, 4, 2, 5, 1, 2, 4, 0, 2, 3, 5, 5, 4, 0, 0, 3, 3, 1, 0, 0, 3,
0, 6, 0, 0, 2, 0, 0, 0,
                      0, 3, 6],
    'Contaminant 7': [0, 0, 1, 4, 4, 0, 5, 2, 0, 4, 2, 2, 3, 4, 7, 6, 1, 4, 0, 0, 1, 3, 3, 4, 1,
4, 1, 4, 0, 2, 1, 2, 1,
                      5, 4, 1],
    'Contaminant 8': [0, 1, 0, 3, 2, 0, 1, 0, 4, 4, 4, 3, 1, 0, 6, 9, 1, 2, 4, 0, 1, 2, 3, 3, 2,
2, 2, 3, 0, 0, 1, 1, 0,
                      1, 6, 4]
}

df_garments = pd.DataFrame(data_garments)


data_detergents = {
    'Detergent': np.arange(1, 11),
    'Contaminant 1 Solubility': [0.54, 0.77, 0.7, 0.51, 0.39, 0.45, 0.69, 0.52, 0.8,
0.47],
    'Contaminant 2 Solubility': [0.75, 0.67, 0.64, 0.54, 0.72, 0.6, 0.55, 0.44, 0.65,
0.81],
    'Contaminant 3 Solubility': [0.78, 0.59, 0.62, 0.66, 0.43, 0.53, 0.62, 0.63, 0.56,
0.77],
    'Contaminant 4 Solubility': [0.69, 0.58, 0.71, 0.82, 0.57, 0.48, 0.56, 0.71, 0.49,
0.53],
    'Contaminant 5 Solubility': [0.8, 0.71, 0.63, 0.7, 0.46, 0.55, 0.47, 0.56, 0.73,
0.64],
    'Contaminant 6 Solubility': [0.66, 0.48, 0.78, 0.6, 0.53, 0.45, 0.38, 0.68, 0.55,
0.59],
    'Contaminant 7 Solubility': [0.55, 0.45, 0.5, 0.46, 0.4, 0.49, 0.44, 0.36, 0.47,
0.53],
    'Contaminant 8 Solubility': [0.73, 0.66, 0.8, 0.55, 0.6, 0.77, 0.64, 0.66, 0.61,
0.42]
}

df_detergents = pd.DataFrame(data_detergents)


fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')

colors = ['r', 'g', 'b', 'y', 'c', 'm', 'orange', 'purple']
markers = ['o', '^', 's', '*', 'x', 'D', 'p', 'h']

for i in range(1, 9):
    ax.scatter(df_garments['Garment'], df_garments[f'Contaminant {i}'], i,
color=colors[i-1], marker=markers[i-1], label=f'Contaminant {i}')

ax.set_xlabel('Garment', fontsize=12, fontweight='bold')
ax.set_ylabel('Contaminant Quantity', fontsize=12, fontweight='bold')
ax.set_zlabel('Contaminant Type', fontsize=12, fontweight='bold')
ax.view_init(30, 45)
ax.legend()

plt.show()




fig2 = plt.figure(figsize=(10, 7))
ax2 = fig2.add_subplot(111, projection='3d')
```

```
colors = ['pink', 'lightblue', 'lime', 'grey', 'navy', 'maroon', 'aqua', 'olive']

for i in range(1, 9):
    ax2.scatter(df_detergents['Detergent'],        df_detergents[f'Contaminant        {i}
Solubility'], i, color=colors[i-1], label=f'Contaminant {i} Solubility')

ax2.set_xlabel('Detergent', fontsize=12, fontweight='bold')
ax2.set_ylabel('Solubility', fontsize=12, fontweight='bold')
ax2.set_zlabel('Contaminant Type', fontsize=12, fontweight='bold')
ax2.view_init(30, 45)
ax2.legend()

plt.show()


import pandas as pd
import pulp



data_garments = {
    'Garment': list(range(1, 37)),
    'Contaminant 1': [8, 3, 2, 2, 2, 0, 2, 1, 7, 5, 5, 6, 0, 5, 3, 0, 5, 5, 1, 2, 6, 5, 0, 1, 5,
1, 9, 6, 0, 7, 5, 4, 0,
                      0, 2, 1],
    'Contaminant 2': [5, 2, 8, 0, 0, 0, 7, 0, 2, 6, 0, 2, 6, 2, 0, 0, 6, 1, 7, 0, 5, 2, 4, 1, 0,
1, 2, 2, 3, 5, 2, 2, 0,
                      5, 1, 2],
    'Contaminant 3': [2, 4, 2, 5, 4, 0, 4, 4, 0, 4, 5, 8, 5, 3, 0, 0, 0, 2, 8, 1, 4, 0, 1, 2, 0,
3, 3, 4, 4, 0, 3, 0, 2,
                      2, 2, 5],
    'Contaminant 4': [4, 5, 1, 3, 7, 6, 5, 3, 5, 2, 0, 3, 4, 1, 0, 4, 5, 3, 2, 0, 0, 4, 5, 5, 5,
3, 2, 2, 0, 0, 3, 3, 4,
                      0, 4, 3],
    'Contaminant 5': [3, 0, 2, 2, 1, 5, 2, 6, 3, 2, 2, 4, 2, 2, 4, 0, 2, 4, 3, 4, 2, 5, 1, 3, 6,
0, 5, 0, 0, 5, 7, 1, 6,
                      1, 0, 4],
    'Contaminant 6': [1, 0, 2, 1, 3, 4, 2, 5, 1, 2, 4, 0, 2, 3, 5, 5, 4, 0, 0, 3, 3, 1, 0, 0, 3,
0, 6, 0, 0, 2, 0, 0, 0,
                      0, 3, 6],
    'Contaminant 7': [0, 0, 1, 4, 4, 0, 5, 2, 0, 4, 2, 2, 3, 4, 7, 6, 1, 4, 0, 0, 1, 3, 3, 4, 1,
4, 1, 4, 0, 2, 1, 2, 1,
                      5, 4, 1],
    'Contaminant 8': [0, 1, 0, 3, 2, 0, 1, 0, 4, 4, 4, 3, 1, 0, 6, 9, 1, 2, 4, 0, 1, 2, 3, 3, 2,
2, 2, 3, 0, 0, 1, 1, 0,
                                    1, 6, 4]
}

df_garments = pd.DataFrame(data_garments)

data_detergents = {
      'Detergent': list(range(1, 11)),
      'Contaminant 1': [0.54, 0.77, 0.7, 0.51, 0.39, 0.45, 0.69, 0.52, 0.8, 0.47],
      'Contaminant 2': [0.75, 0.67, 0.64, 0.54, 0.72, 0.6, 0.55, 0.44, 0.65, 0.81],
      'Contaminant 3': [0.78, 0.59, 0.62, 0.66, 0.43, 0.53, 0.62, 0.63, 0.56, 0.77],
      'Contaminant 4': [0.69, 0.58, 0.71, 0.82, 0.57, 0.48, 0.56, 0.71, 0.49, 0.53],
      'Contaminant 5': [0.8, 0.71, 0.63, 0.7, 0.46, 0.55, 0.47, 0.56, 0.73, 0.64],
      'Contaminant 6': [0.66, 0.48, 0.78, 0.6, 0.53, 0.45, 0.38, 0.68, 0.55, 0.59],
      'Contaminant 7': [0.55, 0.45, 0.5, 0.46, 0.4, 0.49, 0.44, 0.36, 0.47, 0.53],
      'Contaminant 8': [0.73, 0.66, 0.8, 0.55, 0.6, 0.77, 0.64, 0.66, 0.61, 0.42],
      'Unit Price': [0.25, 0.09, 0.11, 0.19, 0.08, 0.1, 0.12, 0.11, 0.18, 0.22]
}


df_garments = pd.DataFrame(data_garments)
df_detergents = pd.DataFrame(data_detergents)


model = pulp.LpProblem("Minimize_Cost", pulp.LpMinimize)


x = pulp.LpVariable.dicts("x", ((i, j, k) for i in range(1, 11) for j in range(1, 9) for k in
range(1, 37)), lowBound=0)


water_cost = 3.8
model += pulp.lpSum(
      [x[(i, j, k)] * df_detergents.loc[i - 1, 'Unit Price'] for i in range(1, 11) for j in
range(1, 9) for k in
        range(1, 37)]) + water_cost


for k in range(1, 37):
      for j in range(1, 9):
            model += pulp.lpSum([x[(i, j, k)] * df_detergents.loc[i - 1, f'Contaminant
{j}'] for i in range(1, 11)]) >= \
                              df_garments.loc[k - 1, f'Contaminant {j}']
```

```
model.solve()


for v in model.variables():
    if v.varValue > 0:
        print(v.name, "=", v.varValue)


if model.status == pulp.LpStatusOptimal:
    print("Find the optimal solution")
else:
    print("No optimal solution was found")


detergent_usage = np.zeros((10, 36))


for v in model.variables():
    if v.varValue > 0:

        name_parts = v.name.split('_')
        if len(name_parts) == 5:
            i = int(name_parts[1])
            j = int(name_parts[2])
            k = int(name_parts[3])
            detergent_usage[i - 1, k - 1] += v.varValue


fig3 = plt.figure()
ax3 = fig3.add_subplot(111, projection='3d')


X = np.arange(1, 11)
Y = np.arange(1, 37)
X, Y = np.meshgrid(X, Y)
Z = detergent_usage.T


ax3.bar3d(X.ravel(), Y.ravel(), np.zeros_like(Z.ravel()), 1, 1, Z.ravel(), color='yellow',
shade=True)

ax3.set_xlabel('Detergent', fontsize=12, fontweight='bold')
ax3.set_ylabel('Garment', fontsize=12, fontweight='bold')
```

```
ax3.set_zlabel('Usage Quantity', fontsize=12, fontweight='bold')
ax3.grid(True)

plt.show()
```

#Question 3 OUTPUT:

```
At line 2 NAME           MODEL
At line 3 ROWS
At line 293 COLUMNS
At line 6054 RHS
At line 6343 BOUNDS
At line 6344 ENDATA
Problem MODEL has 288 rows, 2880 columns and 2880 elements
Coin0008I MODEL read with 0 errors
Option for timeMode changed from cpu to elapsed
Presolve 29 (-259) rows, 58 (-2822) columns and 58 (-2822) elements
Perturbing problem by 0.001% of 0.09 - largest nonzero change 9.9661821e-07
( 0.0011073536%) - largest zero change 0
0  Obj 85.62306 Primal inf 191.11108 (29)
29  Obj 102.8232
Optimal - objective value 102.82306
After Postsolve, objective 102.82306, infeasibilities - dual 0 (0), primal 0 (0)
Optimal objective 102.8230596 - 29 iterations time 0.002, Presolve 0.00
Option for printingOptions changed from normal to all
Total time (CPU seconds):       0.03   (Wallclock seconds):       0.03

x_(2,_1,_1) = 10.38961
x_(2,_1,_10) = 6.4935065
x_(2,_1,_11) = 6.4935065
x_(2,_1,_12) = 7.7922078
x_(2,_1,_14) = 6.4935065
x_(2,_1,_15) = 3.8961039
x_(2,_1,_17) = 6.4935065
x_(2,_1,_18) = 6.4935065
x_(2,_1,_19) = 1.2987013
x_(2,_1,_2) = 3.8961039
x_(2,_1,_20) = 2.5974026
x_(2,_1,_21) = 7.7922078
x_(2,_1,_22) = 6.4935065
x_(2,_1,_24) = 1.2987013
x_(2,_1,_25) = 6.4935065
x_(2,_1,_26) = 1.2987013
x_(2,_1,_27) = 11.688312
x_(2,_1,_28) = 7.7922078
x_(2,_1,_3) = 2.5974026
x_(2,_1,_30) = 9.0909091
x_(2,_1,_31) = 6.4935065
x_(2,_1,_32) = 5.1948052
x_(2,_1,_35) = 2.5974026
x_(2,_1,_36) = 1.2987013
x_(2,_1,_4) = 2.5974026
x_(2,_1,_5) = 2.5974026
x_(2,_1,_7) = 2.5974026
x_(2,_1,_8) = 1.2987013
x_(2,_1,_9) = 9.0909091
x_(2,_3,_1) = 3.3898305
x_(2,_3,_10) = 6.779661
x_(2,_3,_11) = 8.4745763
x_(2,_3,_12) = 13.559322
x_(2,_3,_13) = 8.4745763
x_(2,_3,_14) = 5.0847458
x_(2,_3,_18) = 3.3898305
x_(2,_3,_19) = 13.559322
x_(2,_3,_2) = 6.779661
x_(2,_3,_20) = 1.6949153
x_(2,_3,_21) = 6.779661
x_(2,_3,_23) = 1.6949153
x_(2,_3,_24) = 3.3898305
x_(2,_3,_26) = 5.0847458
x_(2,_3,_27) = 5.0847458
x_(2,_3,_28) = 6.779661
x_(2,_3,_29) = 6.779661
x_(2,_3,_3) = 3.3898305
x_(2,_3,_31) = 5.0847458
x_(2,_3,_33) = 3.3898305
x_(2,_3,_34) = 3.3898305
x_(2,_3,_35) = 3.3898305
x_(2,_3,_36) = 8.4745763
x_(2,_3,_4) = 8.4745763
x_(2,_3,_5) = 6.779661
x_(2,_3,_7) = 6.779661
x_(2,_3,_8) = 6.779661
x_(2,_5,_1) = 4.2253521
x_(2,_5,_10) = 2.8169014
x_(2,_5,_11) = 2.8169014
x_(2,_5,_12) = 5.6338028
x_(2,_5,_13) = 2.8169014
x_(2,_5,_14) = 2.8169014
x_(2,_5,_15) = 5.6338028
x_(2,_5,_17) = 2.8169014
x_(2,_5,_18) = 5.6338028
x_(2,_5,_19) = 4.2253521
x_(2,_5,_20) = 5.6338028
x_(2,_5,_21) = 2.8169014
x_(2,_5,_22) = 7.0422535
x_(2,_5,_23) = 1.4084507
x_(2,_5,_24) = 4.2253521
x_(2,_5,_25) = 8.4507042
x_(2,_5,_27) = 7.0422535
x_(2,_5,_3) = 2.8169014
x_(2,_5,_30) = 7.0422535
x_(2,_5,_31) = 9.8591549
x_(2,_5,_32) = 1.4084507
x_(2,_5,_33) = 8.4507042
x_(2,_5,_34) = 1.4084507
x_(2,_5,_36) = 5.6338028
x_(2,_5,_4) = 2.8169014
x_(2,_5,_5) = 1.4084507
x_(2,_5,_6) = 7.0422535
x_(2,_5,_7) = 2.8169014
x_(2,_5,_8) = 8.4507042
x_(2,_5,_9) = 4.2253521
x_(2,_7,_10) = 8.8888889
x_(2,_7,_11) = 4.4444444
x_(2,_7,_12) = 4.4444444
x_(2,_7,_13) = 6.6666667
x_(2,_7,_14) = 8.8888889
x_(2,_7,_15) = 15.555556
x_(2,_7,_16) = 13.333333
x_(2,_7,_17) = 2.2222222
x_(2,_7,_18) = 8.8888889
x_(2,_7,_21) = 2.2222222
x_(2,_7,_22) = 6.6666667
x_(2,_7,_23) = 6.6666667
x_(2,_7,_24) = 8.8888889
x_(2,_7,_25) = 2.2222222
x_(2,_7,_26) = 8.8888889
x_(2,_7,_27) = 2.2222222
x_(2,_7,_28) = 8.8888889
x_(2,_7,_3) = 2.2222222
x_(2,_7,_30) = 4.4444444
x_(2,_7,_31) = 2.2222222
x_(2,_7,_32) = 4.4444444
x_(2,_7,_33) = 2.2222222
x_(2,_7,_34) = 11.111111
x_(2,_7,_35) = 8.8888889
x_(2,_7,_36) = 2.2222222
x_(2,_7,_4) = 8.8888889
x_(2,_7,_5) = 8.8888889
x_(2,_7,_7) = 11.111111
x_(2,_7,_8) = 4.4444444
x_(3,_6,_1) = 1.2820513
x_(3,_6,_10) = 2.5641026
x_(3,_6,_11) = 5.1282051
x_(3,_6,_13) = 2.5641026
x_(3,_6,_14) = 3.8461538
x_(3,_6,_15) = 6.4102564
x_(3,_6,_16) = 6.4102564
x_(3,_6,_17) = 5.1282051
x_(3,_6,_20) = 3.8461538
x_(3,_6,_21) = 3.8461538
x_(3,_6,_22) = 1.2820513
x_(3,_6,_25) = 3.8461538
x_(3,_6,_27) = 7.6923077
x_(3,_6,_3) = 2.5641026
x_(3,_6,_30) = 2.5641026
x_(3,_6,_35) = 3.8461538
x_(3,_6,_36) = 7.6923077
x_(3,_6,_4) = 1.2820513
x_(3,_6,_5) = 3.8461538
x_(3,_6,_6) = 5.1282051
x_(3,_6,_7) = 2.5641026
x_(3,_6,_8) = 6.4102564
x_(3,_6,_9) = 1.2820513
x_(5,_2,_1) = 6.9444444
x_(5,_2,_10) = 8.3333333
x_(5,_2,_12) = 2.7777778
x_(5,_2,_13) = 8.3333333
x_(5,_2,_14) = 2.7777778
x_(5,_2,_17) = 8.3333333
x_(5,_2,_18) = 1.3888889
x_(5,_2,_19) = 9.7222222
x_(5,_2,_2) = 2.7777778
x_(5,_2,_21) = 6.9444444
x_(5,_2,_22) = 2.7777778
x_(5,_2,_23) = 5.5555556
x_(5,_2,_24) = 1.3888889
x_(5,_2,_26) = 1.3888889
x_(5,_2,_27) = 2.7777778
x_(5,_2,_28) = 2.7777778
x_(5,_2,_29) = 4.1666667
x_(5,_2,_3) = 11.111111
x_(5,_2,_30) = 6.9444444
x_(5,_2,_31) = 2.7777778
x_(5,_2,_32) = 2.7777778
x_(5,_2,_34) = 6.9444444
x_(5,_2,_35) = 1.3888889
x_(5,_2,_36) = 2.7777778
x_(5,_2,_7) = 9.7222222
x_(5,_2,_9) = 2.7777778
x_(5,_4,_1) = 7.0175439
x_(5,_4,_10) = 3.5087719
x_(5,_4,_12) = 5.2631579
x_(5,_4,_13) = 7.0175439
x_(5,_4,_14) = 1.754386
x_(5,_4,_16) = 7.0175439
x_(5,_4,_17) = 8.7719298
x_(5,_4,_18) = 5.2631579
x_(5,_4,_19) = 3.5087719
x_(5,_4,_2) = 8.7719298
x_(5,_4,_22) = 7.0175439
x_(5,_4,_23) = 8.7719298
x_(5,_4,_24) = 8.7719298
x_(5,_4,_25) = 8.7719298
x_(5,_4,_26) = 5.2631579
x_(5,_4,_27) = 3.5087719
x_(5,_4,_28) = 3.5087719
x_(5,_4,_3) = 1.754386
x_(5,_4,_31) = 5.2631579
x_(5,_4,_32) = 5.2631579
x_(5,_4,_33) = 7.0175439
x_(5,_4,_35) = 7.0175439
x_(5,_4,_36) = 5.2631579
x_(5,_4,_4) = 5.2631579
x_(5,_4,_5) = 12.280702
x_(5,_4,_6) = 10.526316
x_(5,_4,_7) = 8.7719298
x_(5,_4,_8) = 5.2631579
x_(5,_4,_9) = 8.7719298
x_(6,_8,_10) = 5.1948052
x_(6,_8,_11) = 5.1948052
x_(6,_8,_12) = 3.8961039
x_(6,_8,_13) = 1.2987013
x_(6,_8,_15) = 7.7922078
x_(6,_8,_16) = 11.688312
x_(6,_8,_17) = 1.2987013
x_(6,_8,_18) = 2.5974026
x_(6,_8,_19) = 5.1948052
x_(6,_8,_2) = 1.2987013
x_(6,_8,_21) = 1.2987013
x_(6,_8,_22) = 2.5974026
x_(6,_8,_23) = 3.8961039
x_(6,_8,_24) = 3.8961039
x_(6,_8,_25) = 2.5974026
x_(6,_8,_26) = 2.5974026
x_(6,_8,_27) = 2.5974026
x_(6,_8,_28) = 3.8961039
x_(6,_8,_31) = 1.2987013
x_(6,_8,_32) = 1.2987013
x_(6,_8,_34) = 1.2987013
x_(6,_8,_35) = 7.7922078
x_(6,_8,_36) = 5.1948052
x_(6,_8,_4) = 3.8961039
x_(6,_8,_5) = 2.5974026
x_(6,_8,_7) = 1.2987013
x_(6,_8,_9) = 5.1948052
Find the optimal solution
```

#Question 4 code:

```
def can_be_together(item1, item2, matrix):
    return matrix[item1][item2] == 1 and matrix[item2][item1] == 1


def add_to_group(item, groups, matrix):
    for group in groups:
        if all(can_be_together(item, member, matrix) for member in group):
            group.append(item)
            return True
    return False
```

```
def find_min_groups(matrix):
    n = len(matrix)  # Number of items
    groups = []  # List to store the result groups

    for item in range(n):
        if not add_to_group(item, groups, matrix):
            # Item could not be added to existing groups, create a new group
            groups.append([item])

    return groups


# Given matrix
matrix = [
    [1, 0, 1, 1, 1, 1, 0, 0],
    [0, 1, 0, 0, 0, 0, 0, 1],
    [1, 0, 1, 0, 0, 1, 1, 1],
    [1, 0, 0, 1, 0, 1, 1, 1],
    [1, 0, 0, 0, 1, 0, 1, 1],
    [1, 0, 1, 1, 0, 1, 1, 1],
    [0, 0, 1, 1, 1, 1, 1, 1],
    [0, 1, 1, 1, 1, 1, 1, 1]
]

# Finding the minimum groups
groups = find_min_groups(matrix)

# Display the groups
for i, group in enumerate(groups, start=1):
    print(f"Group {i}: {', '.join(str(item + 1) for item in group)}")
```