Project C

Project report

Organization of cultural days (shows + lunch + concerts)

BELKHITER Yannis, TEXIER Lucas, IMT Mines Alès, 2IA

Introduction

Organizing cultural events is a complex task that requires careful optimization of available resources. In this context, a company organizing cultural days proposed an interesting problem:

How to allocate all families registered for a given week while respecting a set of constraints and minimizing a non-linear cost function?

This issue is particularly delicate, because it is necessary to take into account the preferences of each family while ensuring that the number of participants on a day does not exceed the limits set by the organizing company, under penalty of penalties and additional costs.

In this project report, we will present a solution to resolve this problem of organizing cultural days. We will begin by describing the constraints and objectives of this project, as well as the different methods that were considered. We will then present in detail the implementation we implemented and the results we obtained.
Finally, we will conclude by discussing the prospects for applying this method and possible avenues for improvement.

I/ Definition of the project

A/ Definition of the problem

The project consists of developing a program in C/C++ or Python to solve two problems:

ÿ **Question 1:** The first problem involves reading five csv files containing different choice lists, each containing six columns: the number of people per family and five weekday preferences (from 0 to 6). The program must propose optimal or pseudo-optimal solutions depending on the objective function.

ÿ **Question 2:** The second problem involves managing the addition of three additional families who wish to participate in an already planned event. The organizing company has finalized the schedule and no longer wishes to modify it. The three additional families have a number of people between 3 and 7. The company decides to assign these families to the day when there are the fewest people. Additional families want to get as much compensation as possible for being assigned to a day different from their first choice.

The program should solve both problems using an exhaustive exploration method to explore the solution space and find the optimal solution for each input instance.
The program must also provide a complete analysis, including its algorithmic complexity and the results obtained for each input instance presented in the form of a summary table.

We will choose to program these algorithms in C, which is more efficient and faster in compiling the program than the Python language.

B/ Definition of constraints

The project involves assigning families to events over a **seven-day period.** Each family has given **five preferences** for the days of the week and the **number of members** in each family is also known.

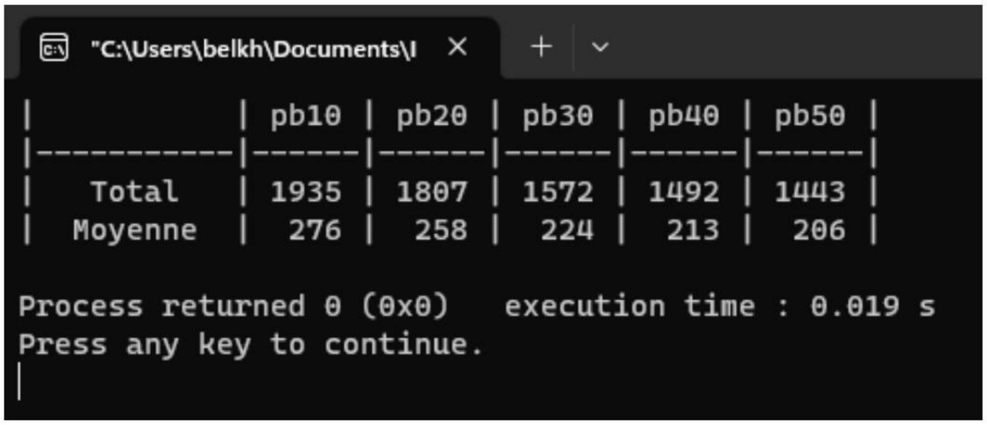
ÿ The first constraint to respect is that all families must be assigned to a event, using their preferences if possible.

ÿ The second constraint is that the **total number of participants** for **each day** must **not exceed 250** and must be at least **125.** This means that the number of families assigned to an event on a given day must be carefully planned to meet this constraint.

$$\forall\ \text{jour}, 0 \leq \text{jour} \leq 6 : 125 \leq \mathbf{pop(jour)} \leq 250,$$

The goal is to find an optimal assignment of families to events using the exhaustive exploration method to minimize the number of families assigned to a day different from their first choice and thus reduce the expenses of the organizing company.

However, after observing the number of families per day, we can see that the first two problems do not satisfy this second constraint (average number of people per day greater than 250, etc.). We will therefore satisfy the second constraint for problems 3, 4 and 5.



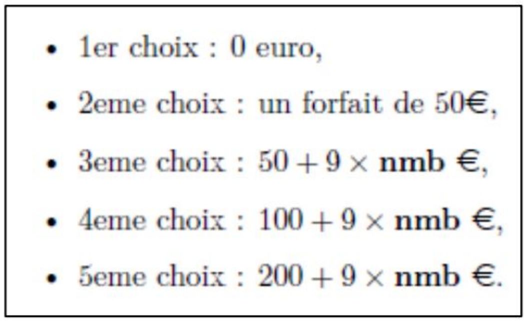
```
"C:\Users\belkh\Documents\I

|           | pb10 | pb20 | pb30 | pb40 | pb50 |
|-----------|------|------|------|------|------|
|   Total   | 1935 | 1807 | 1572 | 1492 | 1443 |
|  Moyenne  |  276 |  258 |  224 |  213 |  206 |

Process returned 0 (0x0)   execution time : 0.019 s
Press any key to continue.
```

Figure 1: Calculation of the total number of people to be affected by problem

C/ Definition of objectives

The main objective of the organizing company is to **minimize the costs** associated with assigning families to days different from their first choice, by using the exhaustive exploration method to optimize the assignment of families to events over a period of time. of seven days. Assignment costs are defined according to the preferred choice of each family:

Cost(family) =



- 1er choix : 0 euro,
- 2eme choix : un forfait de 50€,
- 3eme choix : $50 + 9 \times$ **nmb** €,
- 4eme choix : $100 + 9 \times$ **nmb** €,
- 5eme choix : $200 + 9 \times$ **nmb** €.

Figure 2: Calculation of the cost of each family

In addition, the company must respect the constraint on the number of participants for each day, which must not exceed 250 and must be at least 125 (constraint no. 2). Therefore, the distribution of the number of participants over the week must be as homogeneous as possible to avoid any significant population difference between consecutive days.

A penalty cost, calculated by experts, is therefore associated with the population difference between each consecutive day:

- Si jour < 6, Penalité (jour) $=$

$$((\mathrm{pop(jour)} - 125)/400) * \mathrm{pop(jour)}^{(|\mathrm{pop(jour)} - \mathrm{pop(jour+1)}|/50)}$$

- Et pour le dernier jour, si jour $= 6$, Penalité(jour) $=$

$$((\mathrm{pop(jour)} - 125)/400) * \mathrm{pop(jour)}^{(|\mathrm{pop(jour)} - \mathrm{pop(jour)}|/50)}$$

La ***fonction objectif*** (fonction économique) est donc :

$$\sum_{f=1}^{f=\mathrm{nbFamile}} \mathrm{Cout}(f) + \sum_{j=0}^{j=6} \mathrm{Penalite}(j)$$

Figure 3: Calculation of penalties for each consecutive day

The objective function of society is therefore to **minimize the total cost,** which includes the **costs of assigning families** as well as the **penalty costs linked to the difference in population between consecutive days.** This economic function will be used to guide the exhaustive exploration of all possible assignments in order to find the optimal assignment that meets the imposed constraints while minimizing total costs.

La ***fonction objectif*** (fonction économique) est donc :

$$\sum_{f=1}^{f=\mathrm{nbFamile}} \mathrm{Cout}(f) + \sum_{j=0}^{j=6} \mathrm{Penalite}(j)$$

Figure 4: Calculation of the total cost to minimize

II/ Description of the method

Given the form of the objective function, the algorithm must assign an equivalent number of people per day, that is to say, assign the same number of people on each day to reduce the value of the differences between consecutive days (and therefore to minimize the calculation of penalties); while maximizing the allocation of families' first choices, therefore by allocating the preferred day to families.

A/ Naive algorithm:

The first algorithm presented is a fairly naive algorithm, which will assign each family its number one choice. The latter is quite basic and does not allow for minimization since the choice of each family is quite random. Certainly the cost linked to the assignment of each family will be zero, but the cost of the penalties will be variable and very high (exponential, etc.). This algorithm is therefore not effective, it must be improved.

B/ Naive reattribution algorithm:

The second algorithm presented is a more advanced algorithm than the first, which is executed in two stages. The first step is to assign each family their number one choice (like the first algorithm).
Then, the algorithm will select the busiest day (max_loc, where the most people are affected), and the least busy day (min_loc, where the fewest people are affected).

It will affect, in order of reading of the families, the first family that it finds by browsing the linked list including the day max_loc in choice 1 and the day min_loc in choice 2. If no family is found, it affects the first family found in its choice 3, then in its choice 4 if again none is found, then in its choice 5. We apply this algorithm 100 times to the data structure, and we save the configuration each time if it gives a minimal cost. At the output, we therefore have the most optimal configuration for this algorithm.

In short, this algorithm is much more efficient than the first because it makes it possible to “smooth” the number of people allocated per day, so as to greatly reduce the penalty cost of the configuration. Nevertheless, the cost linked to the allocation of families becomes non-zero, but remains much lower than the cost linked to the penalty. We thus find much more efficient results with this resolution algorithm.

C/ Reallocation algorithm by family size:

The third algorithm is structured in the same way as the second, but it differs in the choice of families that are reallocated. While the “naive reassignment algorithm” chooses families in order reading the linked list, this algorithm will select the family using the difference between the quantity max_loc and min_loc. In fact, the algorithm calculates this difference, and chooses a family of size ÿdelta/2ÿ with delta = max_loc - min_loc. Thus, the family that we are going to select will ideally have its choice 2 equal to min_loc, and will make it possible to equalize the number of people who are allocated to max_loc and on min_loc. After several rounds of the loop, “smooth” the number of people allocated per day more finely than the second algorithm and will theoretically give better results than the previous algorithm.

D/ Reattribution algorithm by difference:

The last algorithm is structured in the same way as the second, but it differs in the choice of days where families are chosen. While the “family size reallocation algorithm” chooses the days where there are the most/fewest people allocated, this algorithm selects the max/min days according to the largest successive difference. In fact, the algorithm calculates this difference and chooses in day max_loc and day min_loc the days corresponding to the greatest absolute difference of successive days.
Thus, the family that we will select will ideally have its choice 2 equal to min_loc, and will make it possible to equalize the number of people who are allocated to max_loc and min_loc. The choice of families inside max_loc and min_loc is identical to the previous algorithm. After several loops, “smooth” the number of people allocated per day more finely than the third algorithm and will theoretically give better results than the previous algorithm.

III/ Implementations

We also implemented this method in C for several reasons. First of all, C is a very fast and efficient programming language, which is especially important for solving problems involving large amounts of data. Additionally, C is a low-level programming language, which allows very fine control over memory and performance, which is important for maximizing algorithm efficiency. Finally, C is a very commonly used programming language for data processing algorithms, which makes it a logical choice for solving this type of problem.

A/ Reading and structuring of data:

We have csv files available with the size of the families for each line, and the 5 choices of the family in order of preference. To read this file, we have programmed the “read_csv” function which reads CSV files. The data is stored in a linked list of family structures. Each family structure contains information about the number of family members, the five weekday choices, and the day assigned to them. The function uses “strtok” to extract each field from the CSV row and stores them in the family structure. Finally, the function returns a pointer to the head of the linked list.

B/ Calculation of costs:

The “cost(head)” function calculates the total cost of the solution by iterating through the linked list of families and adding the cost associated with each family based on its assigned day. The cost is calculated using the family's choices and the number of family members. The “penalty(head)” function calculates the penalties associated with the distribution of families over the different days of the week. To do this, it goes through the linked list of families and calculates the sum of penalties for each day. Penalties are calculated using the numbers of family members assigned to each day and taking into account the population difference between consecutive days. The function returns the total sum of penalties for the given solution.

These two functions calculate the total scores for a given solution of the optimization problem by resetting the pointer to the head of the linked list.

C/ Naive algorithm:

The “assign_days()” function aims to minimize the total cost of assigning families to their preferred days, by traversing the linked list and assigning for each family its “assigned_day” to its choice 1.

D/ Naive reattribution algorithm:

This program uses an iterative approach to improve the solution of an optimization problem. The program loops 100 times to try to improve the current solution. For each iteration, it goes through the families one by one and tries to find **in reading order** a more suitable visiting day by swapping the current day with the least preferred day among all available days. It then uses the "cost" and "penalty" functions to calculate the cost and penalty of the new solution and compares it to the best solution found so far. If the new solution is better, the program updates the sum_global, penalties_global, and global_score variables to reflect the new optimal solution.

E/ Algorithm for reattribution by family size:

This program uses an iterative approach to improve the solution of an optimization problem. The program loops 100 times to try to improve the current solution. For each iteration, it goes through families one by one and tries to find **in order of family size** a more suitable visiting day by swapping the current day with the least preferred day among all available days. It then uses the "cost" and "penalty" functions to calculate the cost and penalty of the new solution and compares it to the best solution found so far. If the new solution is better, the program updates the sum_global, penalties_global, and global_score variables to reflect the new optimal solution.

F/ Difference reattribution algorithm:

This program uses an iterative approach to improve the solution of an optimization problem. The program loops 100 times to try to improve the current solution. For each iteration, it goes through families one by one and tries to find a more suitable visiting day by swapping the current day with the least preferred day among all available days. It takes the families by selecting **the greatest successive difference between the families** to minimize the “penalty” function. It then uses the "cost" and "penalty" functions to calculate the cost and penalty of the new solution and compares it to the best solution found so far. If the new solution is better, the program updates the sum_global, penalties_global, and global_score variables to reflect the new optimal solution.

IV/ Results

A/ Naive algorithm:

Here are the results obtained for the naive algorithm:

Problems	pb10.csv	pb20.csv	pb30.csv	pb40.csv	pb50.csv
Score	7578	646	202	1152	858
Cost	0	0	0	0	0
Penalties	7578	646	202	1152	858
Execution time (s)	0.021	0.032	0.016	0.012	0.019

B/ Naive reattribution algorithm:

Here are the results obtained for the naive reattribution algorithm:

Problems	pb10.csv	pb20.csv	pb30.csv	pb40.csv	pb50.csv
Score	650	278	166	527	213
Cost	550	200	50	250	150
Penalties	100	78	116	277	63
Execution time (s)	0.041	0.032	0.038	0.052	0.044

C/ Reallocation algorithm by family size:

Here are the results obtained for the reallocation algorithm by family size:

Problems	pb10.csv	pb20.csv	pb30.csv	pb40.csv	pb50.csv
Score	595	290	166	540	188
Cost	200	100	50	195	100
Penalties	395	190	116	345	88
Execution time (s)	0.049	0.062	0.069	0.044	0.064

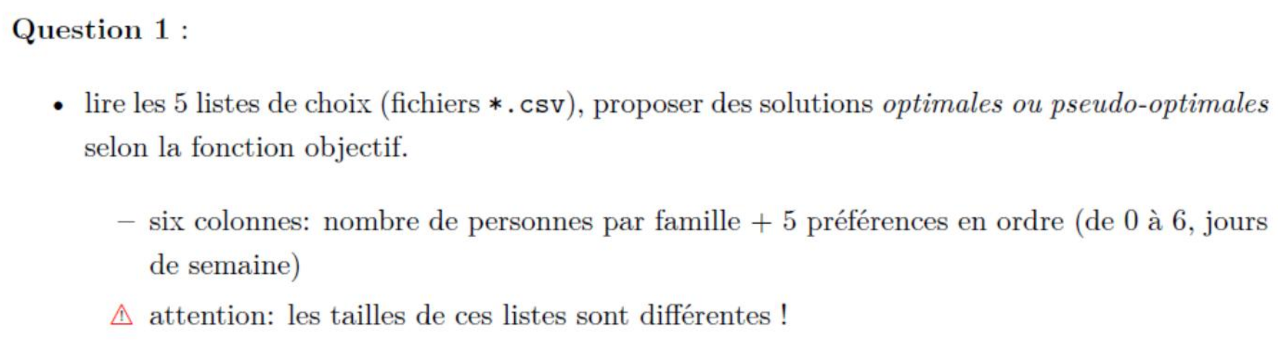
D/ Reattribution algorithm by difference:

Here are the results obtained for the reallocation algorithm by difference between the number of people assigned per successive day:

Problems	pb10.csv	pb20.csv	pb30.csv	pb40.csv	pb50.csv
Score	641	180	108	514	274
Cost	263	50	50	50	50
Penalties	378	130	58	464	224

Execution time (s)	0.047	0.047	0.048	0.047	0.047

V/ Question I



Question 1 :

- lire les 5 listes de choix (fichiers `*.csv`), proposer des solutions *optimales ou pseudo-optimales* selon la fonction objectif.
  - six colonnes: nombre de personnes par famille + 5 préférences en ordre (de 0 à 6, jours de semaine)
  - ⚠ attention: les tailles de ces listes sont différentes !

-The first algorithm presented "Naive algorithm" assigns each family its number one choice, which does not allow the score to be minimized since significant penalties are generated.

-The second algorithm "Naive Reassignment Algorithm" consists of assigning the number one choice to each family, then assigning families to the most full day and the least full day based on their choices. This algorithm makes it possible to smooth out the number of people allocated per day and considerably reduce the penalties compared to the first.

-The third algorithm "Reallocation Algorithm by Family Size" follows the same method as the second, but it chooses families based on the difference between the maximum and minimum amount of people allocated per day. The algorithm makes it possible to equalize the number of people allocated on the most and least busy days. It obtains much lower costs than the second algorithm, however the penalties increase the score.

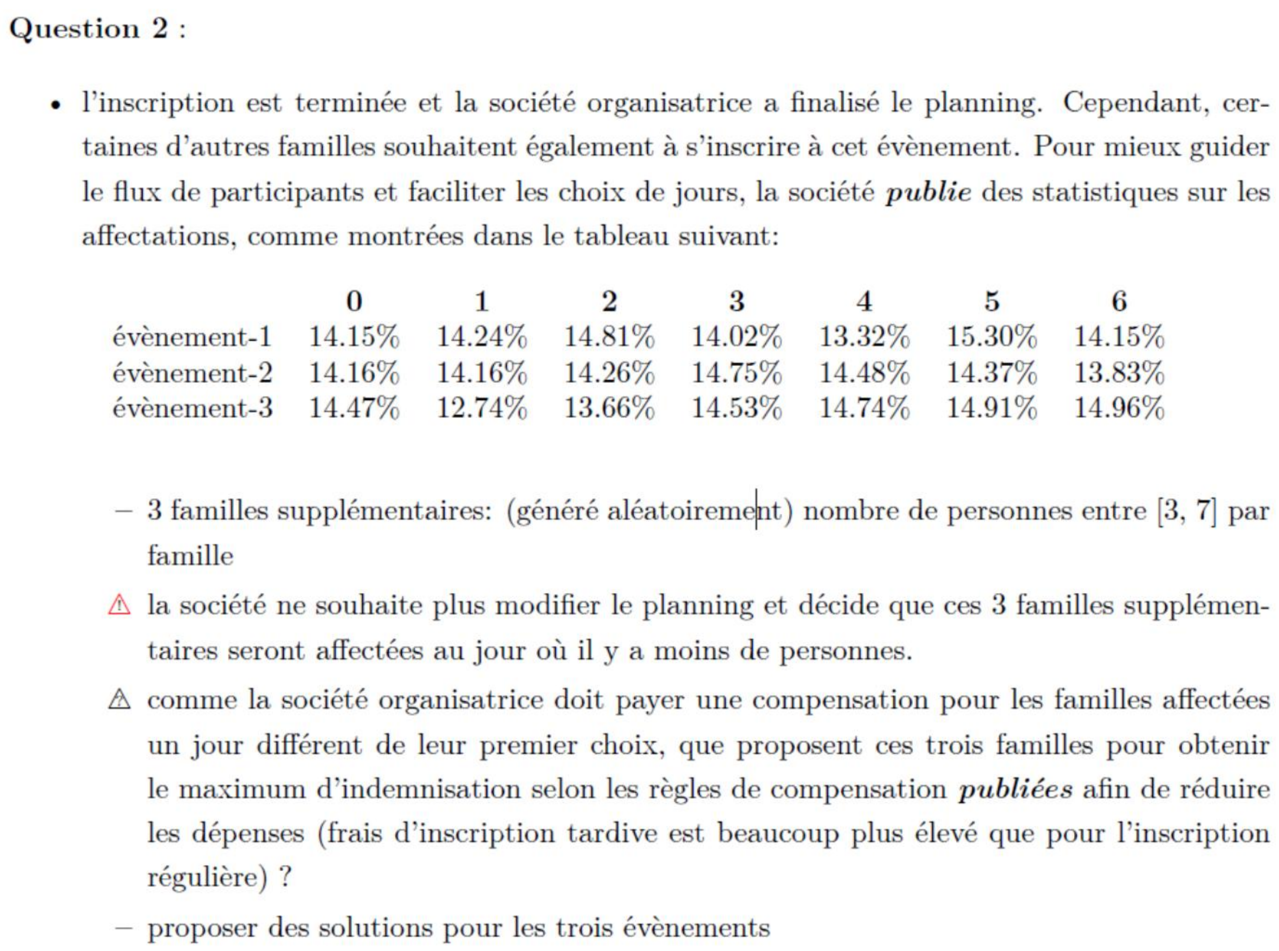
-The fourth algorithm "Difference Reallocation Algorithm" also follows the same method as the second, but it chooses the most and least busy days based on the largest successive difference between days. The families are then chosen in the same way as in the third algorithm.

Comparison :

The second algorithm is more efficient than the first, because it smoothes the number of people allocated per day and considerably reduces the cost of penalties. The third and fourth algorithms make it possible to smooth the number of people allocated per day more finely than the second and should give better results in terms of costs. However, the penalties generated by the two algorithms do not represent a clear improvement.

All the calculation times of the algorithms seem to be of the same order of magnitude

VI/ Question II



Question 2 :

- l'inscription est terminée et la société organisatrice a finalisé le planning. Cependant, certaines d'autres familles souhaitent également à s'inscrire à cet évènement. Pour mieux guider le flux de participants et faciliter les choix de jours, la société ***publie*** des statistiques sur les affectations, comme montrées dans le tableau suivant:

	0	**1**	**2**	**3**	**4**	**5**	**6**
évènement-1	14.15%	14.24%	14.81%	14.02%	13.32%	15.30%	14.15%
évènement-2	14.16%	14.16%	14.26%	14.75%	14.48%	14.37%	13.83%
évènement-3	14.47%	12.74%	13.66%	14.53%	14.74%	14.91%	14.96%

  – 3 familles supplémentaires: (généré aléatoirement) nombre de personnes entre [3, 7] par famille

  ⚠ la société ne souhaite plus modifier le planning et décide que ces 3 familles supplémentaires seront affectées au jour où il y a moins de personnes.

  ⚠ comme la société organisatrice doit payer une compensation pour les familles affectées un jour différent de leur premier choix, que proposent ces trois familles pour obtenir le maximum d'indemnisation selon les règles de compensation ***publiées*** afin de réduire les dépenses (frais d'inscription tardive est beaucoup plus élevé que pour l'inscription régulière) ?

  – proposer des solutions pour les trois évènements

In this question, it is a question of randomly generating 3 families, and then assigning them one day after the round of allocation of the first families.

We will therefore use the rand() function to randomly select integers. In order to randomly assign family choices, we generate a permutation of the digits 0 to 6, then select the first five digits of that permutation.

The number of members is a random number drawn between 2 and 8.

The main thing now is to allocate to the families generated the days when there are the fewest people, while maximizing the compensation received by the family.

With a loop on families we will:

- Initialize an array with 3 values (indexes from 0 to 2 corresponding to the 3 families)
- Determine the minimum day (where the fewest people are allocated)
- Browse the family choices in descending order of preference (we start with the 5th choice of the family then we go back to the 1st choice).
- As soon as a family's choice corresponds to the minimum day, we note the family's choice number at the family index and we move on to the next family

NB: If a family does not include the minimum day in its choices, enter -1 in the table at the family index and move on to the next family.

Once the algorithm is finished, we note the highest choice (i.e. the maximum of the table of values). It therefore represents the smallest day preference among the 3 families. The maximum index represents the family that contains this smallest preference. We therefore attribute this choice to this family.

We start the loop again with the 2 remaining families then with the last family.

NB: If the two remaining families or the remaining family do not have the minimum day in their choices, then we repeat with the second minimum day.

Ultimately, this method makes it possible to respect the requirement relating to the allocation of families on the days when the fewest people are allocated, while maximizing the compensation of the families generated (highest preference = maximization of the cost and therefore of compensation).