Computational Intelligence for Optimization
Professors: Leonardo Vanneschi & Berfin Sakallioglu
May 2023

WEDDING SEATING CHART PROBLEM

Group 19:
Catarina Duarte 20191211
Inês Magessi 20220590
Tiago Santos 20220629

Index

Link for git repository: https://github.com/inesmcm26/cifo_project

1. Definition of the Problem

The longevity of a marriage is not determined by the success of the wedding day, but it sure helps! In the middle of nerves and preparations, an important part of a smooth ceremony is the correct seating arrangement of all the guests at the mealtime. With that in mind, we took on the challenge of creating an optimal seating arrangement of 64 people across 8 tables aiming to maximize the relations among the guests. To achieve this, we used Genetic Algorithms, a computational intelligence tool for optimization, and a pairwise relationship matrix of all wedding attendees (see Table 1 and Fig. 1). Through this approach, we were able to generate an optimal seating plan that betters the ceremony, seeing that every table fosters positive interactions among the seated guests.

Our algorithm is designed to work for any number of guests and tables, as long as there is a corresponding relationships matrix dataset available. This flexibility allows us to adapt the seating arrangement optimization to various wedding sizes and seating configurations, ensuring a personalized and enjoyable experience for all attendees. It is important to notice that the presented problem is a Combinatorial Optimization Problem, known for being NP-Complete, which can be optimized using GGA, Grouping Genetic Algorithm, an extension to the standard GA. What distinguishes GGA is the group-based scheme used to encode solutions in the search space. This also means that the genetic operators have to work at the group level. With this type of operators, procedures are performed in a more controlled way, determining which groups and elements can vary according to the constraints and objectives of the problem.

2. Individual Representation

Initially, we attempted to employ an array of 64 genes to represent our individuals, where each gene denoted a guest as an integer. Our intention was to use a standard genetic algorithm that utilized this array, dividing it into equal-sized segments to represent tables. However, we soon realized that this approach would lead to redundant solutions since it considered the order of guests within tables. To avoid such redundancies, it would be necessary to incorporate more complex methods and genetic operators for the Individual and the initialization of the population would be considerably challenging.

To address this, we treated each table as a distinct subset of the problem. Instead of using the previously mentioned representation, we opted for a different approach that works under the paradigm of GGA. In this new representation, individuals are defined by a list of 8 genes, where each gene corresponds to a table. All the genes within this representation have the same length and contain the IDs of the 8 guests seated at that particular table.

For the internal representation of each table we opted for python sets, given they have several properties that make them well-suited for this purpose. Firstly, sets are mutable, enabling dynamic modification of the group of guests. Furthermore, sets are unordered collections of unique elements, aligning perfectly with our focus on grouping guests on a table rather than defining their specific seat, and ensuring that no guests are repeated within a table. Finally, sets offer improved performance compared to lists since adding and removing elements in sets typically have a time complexity of O(1) due to the utilization of hash tables, while the same operations in lists would require O(n) time.

We chose to use lists to store the sequence of genes, since the mutability of lists allows for easily adding or removing elements, which proved to be valuable during crossover operations. Additionally, the indexing property of lists simplifies the process of exchanging guests between tables, as we can easily keep track of the source table of a removed guest.

Overall, this approach provided multiple advantages. Our new implementation became more organized, offering a clear and intuitive structure that facilitated our understanding of the problem. Moreover, this representation allowed us to design genetic operators that operate at the table level, simplifying the manipulation and evolution of the seating arrangement. Not only did it streamline our approach, but it also reduced the search space, resulting in no isomorphic solutions.

3. Fitness Function

To understand our fitness function it is helpful to conceptualize each table as a bidirectional graph. In this graph, the guests are represented as nodes, while the weighted edges between them are reflected by the values present in the pairwise relationship matrix.

The fitness of a table can be evaluated by summing the weighted edges within the graph. This entails calculating the total value of all connections between guests, taking into account the respective weights assigned to each relationship. As a result, the resulting fitness captures the collective strength of the interpersonal bonds within a specific table. The fitness of a table can be calculated using the following formula:

$$Fitness\ of\ the\ Table\ =\ F_t(t)\ =\ \sum_{i,j \in t} w_{i,j}$$

Where:

- *i, j* denotes a pair of guests in table t
- $w_{i,j}$ represents the value of the relationship matrix between guests *i* and *j*

The solution's fitness is calculated by summing the fitness of all the tables belonging to the solution. This can be represented using the following formula:

$$Fitness\ of\ the\ Individual\ =\ F(i)\ =\ \sum_{k=1}^{t} F_t(k)$$

Where:

- *t d*enotes the total number of tables of the solution

4. Selection

Selection plays a crucial role in a genetic algorithm as it determines which individuals will be chosen for reproduction. In order to choose the best selection algorithm, we examined the influence of three methods on the performance of our GA. To ensure an unbiased assessment, we selected eight random combinations of genetic operators to run with each selection method. Each algorithm was executed for 100 generations, with 30 independent runs, and the final results were averaged across the runs. The median performance of the eight algorithms for each selection method is visualized in Fig. 2.

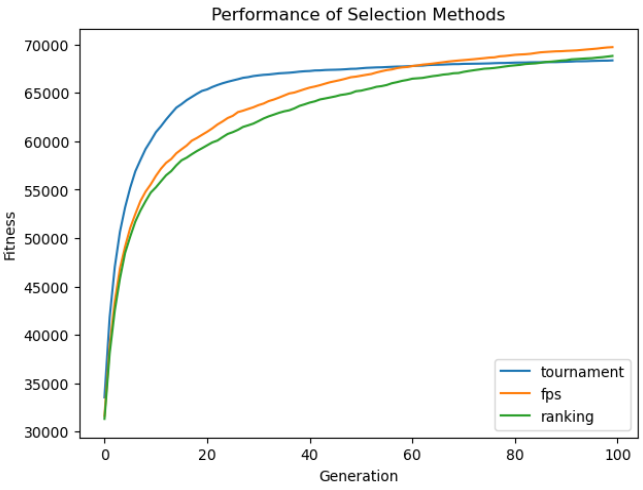

Fig. 2 - Comparison of Selection Methods

This approach allows us to evaluate each selection method's impact on the algorithm's performance across multiple operator configurations. By avoiding a restricted analysis based on a single fixed configuration of crossover and mutation operators, we can obtain more reliable insights into the effects of different selection methods.

The plot reveals that all selection methods exhibit similar average best fitness values in the last generation. However, it is evident that tournament selection converges much faster than the other methods, despite achieving a slightly lower final fitness value. Additionally, through experimentation, we verified that tournament selection significantly outperforms the other methods in terms of computational efficiency.

Based on the obtained experimental results, the constraints on available time for running the algorithms, and the theoretical support for the effectiveness of tournament selection in the literature, we have decided to adopt tournament selection as the default method going forward.

5. Genetic Operators

Considering our decision to use GGAs to optimize our problem and a group-based solutions representation, genetic operators need to act at the group level. It is also important to remember that we did not take into account the seating order within the tables but only the set of guests assigned to it.

One key reason for using this approach is to avoid generating redundant solutions that would unnecessarily increase the search space. Therefore, it is also crucial that the solutions generated by the genetic operators are not only valid, but also non-redundant. An invalid solution occurs when the same guest is seated at multiple tables, while redundant solutions involve merely rearrangements of guests within the same table or complete swaps of tables. All genetic operators implemented on this project comply with these requirements. For visual representations of those, please refer to the Appendix.

5.1. Crossover

5.1.1. Group-based Crossover

The *Group-based Crossover* (GBX) is applied to two parents and creates one offspring. This child is initialized with a random number of full table arrangements selected from the first parent, ranging from a third to a half of the tables. After removing the guests seated in the inherited tables of the first parent from the tables of the second parent, the remaining are rearranged in a greedy way by iteratively merging tables that result in the highest fitness and passing them down to the offspring (see Fig. 3).

GBX efficiently explores the search space by selectively merging tables based on their fitness. This approach helps in quickly identifying and incorporating beneficial seating arrangements into the offspring, leading to potentially faster convergence towards better solutions.

5.1.2. Eager Breeder Crossover

The *Eager Breeder crossover*, is a two-step process composed by a *Collection* phase followed by *Repair* (see Fig. 4). In this process, the tables in both parents are arranged in order of fitness, and the best tables are compared pairwise. The table with the highest fitness between the two best of the parents is selected and passed down to the offspring. This last step is repeated until the offspring has the maximum number of tables. If the same guest is seated at two different tables during this process, an invalid solution is generated. A *Repair* phase is implemented to address this issue.

During *Repair*, any guest that appears in multiple tables is removed from the table where he contributes the least to fitness. To fill the resulting empty seats, a greedy approach is employed, where the guests that were not seated are assigned to the table where they contribute the most to fitness. This strategy guarantees that each guest is seated only once, and it prioritizes tables with higher fitness.

This crossover aims to optimize fitness throughout its stages. In the Collection phase, the highest fitness tables are selected iteratively from the best tables of the parents. In the Repair phase, guests appearing in multiple tables are removed from the table where they have the smallest impact in fitness, while the vacant seats are filled using the opposite heuristic, prioritizing seatings that increase fitness. This approach gradually constructs a high-quality seating arrangement for the offspring.

5.1.3. Twin Maker

Unlike the other crossover methods, the *Twin Maker*, our custom crossover, creates two offspring (see Fig. 5). This is possible by keeping a random amount between a third and half of random guest positions of one parent and then filling the remaining seats with the unseated guests in the order they appear in the other parent. The procedure is repeated twice, alternating the roles of the first parent and the second parent each time.

This crossover selectively keeps a random portion of guest seats from one parent, while also preserving some kind of seating order between tables from the second parent, allowing for the conservation of potentially advantageous genetic material. This helps to maintain a balance between exploration and exploitation, as useful traits from both parents are inherited by the offspring.

5.2. Mutation

5.2.1. Swap Mutation

The *Swap* Mutation is a simple and widely used mutation operator. The big difference from the classic implementation to ours is that we do not allow for guest swaps within the same table (see Fig. 6). This is considered the least disruptive type of mutation as it simply exchanges the positions of two guests between two different tables. Despite being a small change, it effectively introduces diversity into the population and facilitates the exploration of various seating arrangements.

5.2.2. Merge and Split Mutation

The *Merge and Split* mutation, as the name suggests, involves two distinct steps. In the first step, two random tables are chosen, and all the guests seated at those tables are combined into a pool of guests. After that, this group of people is randomly divided, resulting in the creation of two new tables (see Fig. 7). This mutation is slightly more disruptive than the previous one, as it may result in multiple swaps of guests between the two tables.

5.2.3. The Hop Mutation

The Hop mutation is a straightforward concept. It involves moving a random guest from each table to the next table. By doing so, the majority of the genetic material remains unchanged, while the swapping of guests can potentially contribute to better outcomes by introducing diversity (see Fig. 8).

5.2.4. Dream Team Mutation

The *Dream Team* mutation algorithm selectively preserves guests with the strongest relationship at each table while shuffling the rest among tables (see Fig. 9). This algorithm achieves a balance between preserving strong relationships by prioritizing them at each table and introducing variation through a randomized shuffle. It is the most disruptive mutation operator among the ones employed.

6. Evolution

In this section, we explore the process of evolving the population towards better solutions. It all begins with population initialization, in which a set of individuals is created by randomly shuffling a sequence of integers representing the guests. This sequence is then divided into groups of the size of tables and the individual is added to the population, until it reaches the default count of 50 individuals. The chosen value for the population size ensures that the population is large enough to maintain diversity, while still being computationally manageable. To prevent the inclusion of redundant solutions, we use the set data structure to store them during this phase, ensuring that individuals are not replicated since sets only allow for unique values. The group of tables and the tables themselves are also defined as sets to support this process. Once this step is complete, the evolution process begins.

Parents are selected using a predefined method and undergo crossover with a default probability of 0.9. The resulting offspring then suffer mutation with a default probability of 0.1. The choice of these default values is based on their widespread adoption in the GAs literature. The genetic operators to be utilized are specified when invoking the evolutionary process, and they are one of the previously described operators. Finally, the worst individuals from the new population can be replaced by those from the elite group that was saved before undergoing a new generation, which consists of the five best individuals from the previous population. If this option is used, the five best individuals among the elite and the worst individuals from the new population are selected for the new generation.

At the end of each generation, the entire population of ancestors is replaced by the new offspring population, potentially including elements of the elite group. The fitness value of the best individual found in each generation is saved for later analysis. This process is repeated for 100 generations, as we consider it a reasonably large value to ensure convergence.

7. Experimental Results

In order to determine the optimal combination of hyperparameters for our problem, we conducted a Grid Search. This involved exploring all possible combinations of genetic operators, both with and without elitism. It is important to recall that we consistently employed the tournament selection method throughout the experiment, while keeping the other parameters of the genetic algorithm (such as crossover and mutation probabilities, elite size, etc.) fixed at predetermined values. Each set of hyperparameters was tested by running the GA 30 times to ensure accurate and reliable results.

Based on the results of the Grid Search, we focused on the top 5 algorithms that exhibited the highest median fitness values in the final generation. Fig. 10 illustrates the plot of the median best fitness achieved in each generation. Several conclusions can be drawn from this graph. Firstly, all five algorithms consistently achieved remarkably high fitness values, with their performances becoming increasingly similar after the 20th generation. The substantial overlap in standard deviations indicates that there is no statistically significant difference between the performances of these algorithms. For a more detailed visualization, please refer to Fig. 11.

Furthermore, beyond having the *Eager Breeder* crossover in common, the four best and faster to converge algorithms also all use elitism. This suggests that the inclusion of elitism when this crossover method is used also contributes to improved performance.

Regarding the impact of different operators on convergence, it is evident that the *Eager Breeder* crossover yielded the best results, as four out of the top five algorithms utilized this particular method. On the other hand, the effect of mutation on performance seems to be less pronounced. The limited impact of mutation may be attributed to the low probability assigned to its occurrence.

To delve deeper into the influence of each crossover operator on fitness, we conducted a more detailed analysis. Fig. 12 displays the median performance of all algorithms that employed each type of crossover, along with their corresponding standard deviations. It is evident that the *Twin Maker* performed significantly worse than the others. One possible explanation for this result is that both the *Eager Breeder* and *GBX* crossovers rely on heuristics that prioritize retaining and building tables with higher fitness, while the *Twin Maker* randomly selects guests from one parent to occupy seats, while preserving the seating order of the other. This table-order-based heuristic does not seem to provide any advantage when crossing parents, resulting in lower fitness. Furthermore, we can see that the algorithms using *Eager Breeder* crossover converge almost immediately to an optimal solution, while the ones using *GBX* take a bit longer to converge.

From this analysis, we can conclude that the *Eager Breeder* crossover, when used in conjunction with elitism and any mutation operator, significantly contributes to the overall success of our genetic algorithm.

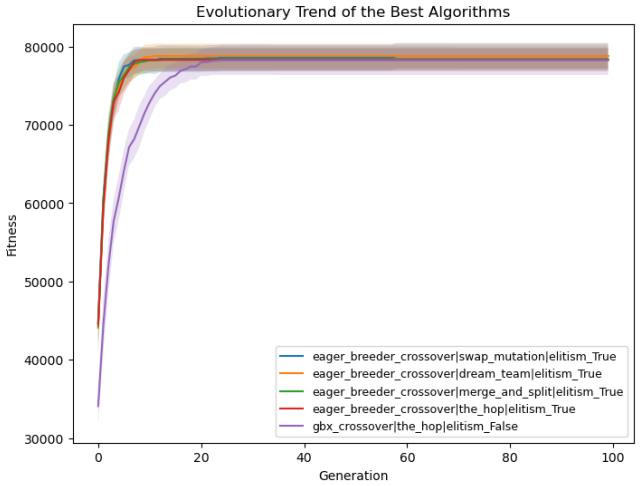

Fig. 10 - *Median best fitness of the top 5 algorithms over generations*

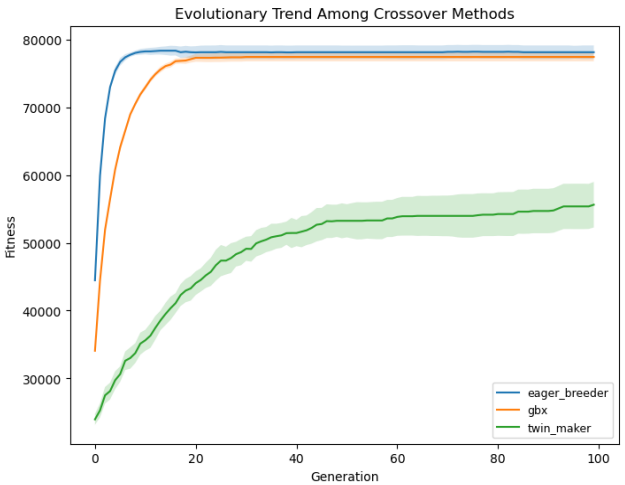

Fig. 12 - *Comparison of Crossover methods*

8. Conclusion

To achieve the optimal seating arrangement for the wedding, we executed one of the best genetic algorithms found by the Grid Search. This GA used *Tournament* selection, *Eager Breeder* crossover, *Dream Team* mutation, and *Elitism* and was executed 30 times using these hyperparameters, while keeping all other settings with the default values mentioned in section 6. As a result, we obtained the best individual solution, as depicted in Fig. 12, with an overall fitness score of 81300.

When examining the seating arrangement, we can observe that the table for the Groom and Bride consists of their best friends and other close acquaintances. The remaining guests appear to be reasonably distributed across the tables, indicating solid results for our problem. Also, no enemies ended up seated at the same table.

However, there have been cases where couples, such as the parents of the Groom, as well as the Groom's brother and sister-in-law (guests 5,6 and 17,20, respectively), have been separated and placed at different tables. This suggests that we should potentially increase the couple's relationship value in order to avoid such situations. Furthermore, upon noticing the peculiar combination of guests at the table labeled 'Bride's Hometown Friends + Groom's Family', an interesting modification to the fitness function comes to mind. This modification would involve providing an advantage to seating individuals who not only have a strong relationship with each other but also share a connection with either the bride or the groom. By incorporating this additional criterion into the fitness function, the seating arrangement could be further optimized. The advantage of keeping individuals with a shared relationship with the bride or the groom together at the same table is that it provides a common topic of conversation, fostering a more engaging atmosphere for the guests.

In conclusion, by leveraging Genetic Algorithms, we have successfully achieved a promising seating arrangement that ensures a smooth and enjoyable wedding experience. With all the preparations in place, let the ceremony begin!

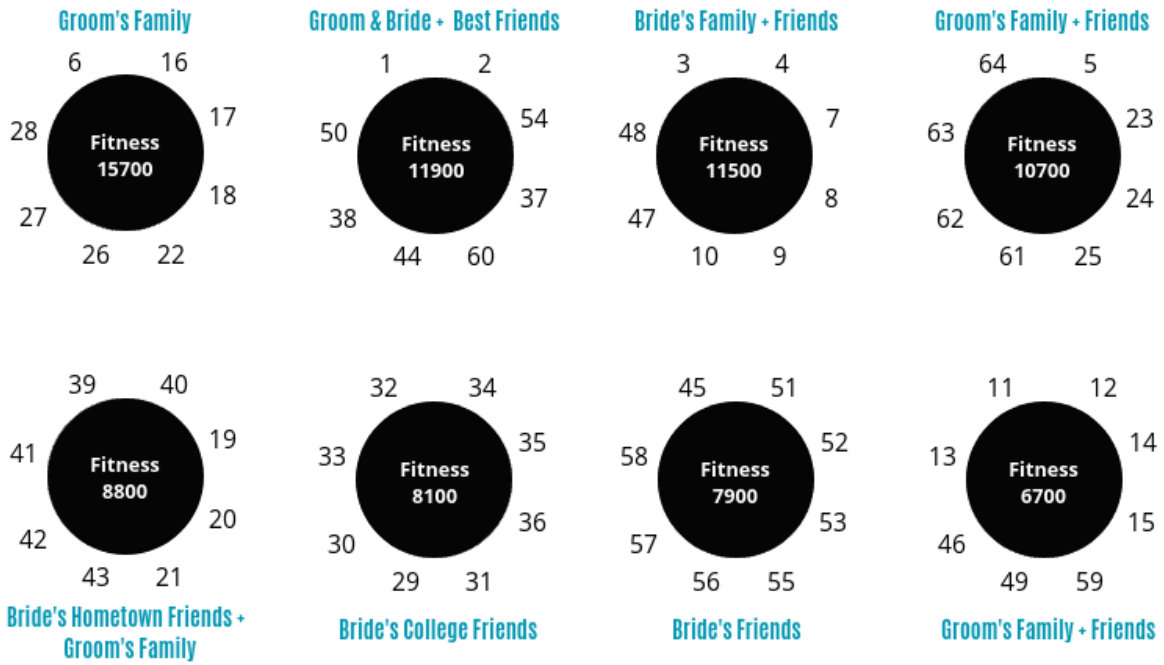

Fig. 12 *- Final best seating arrangement*

Appendix

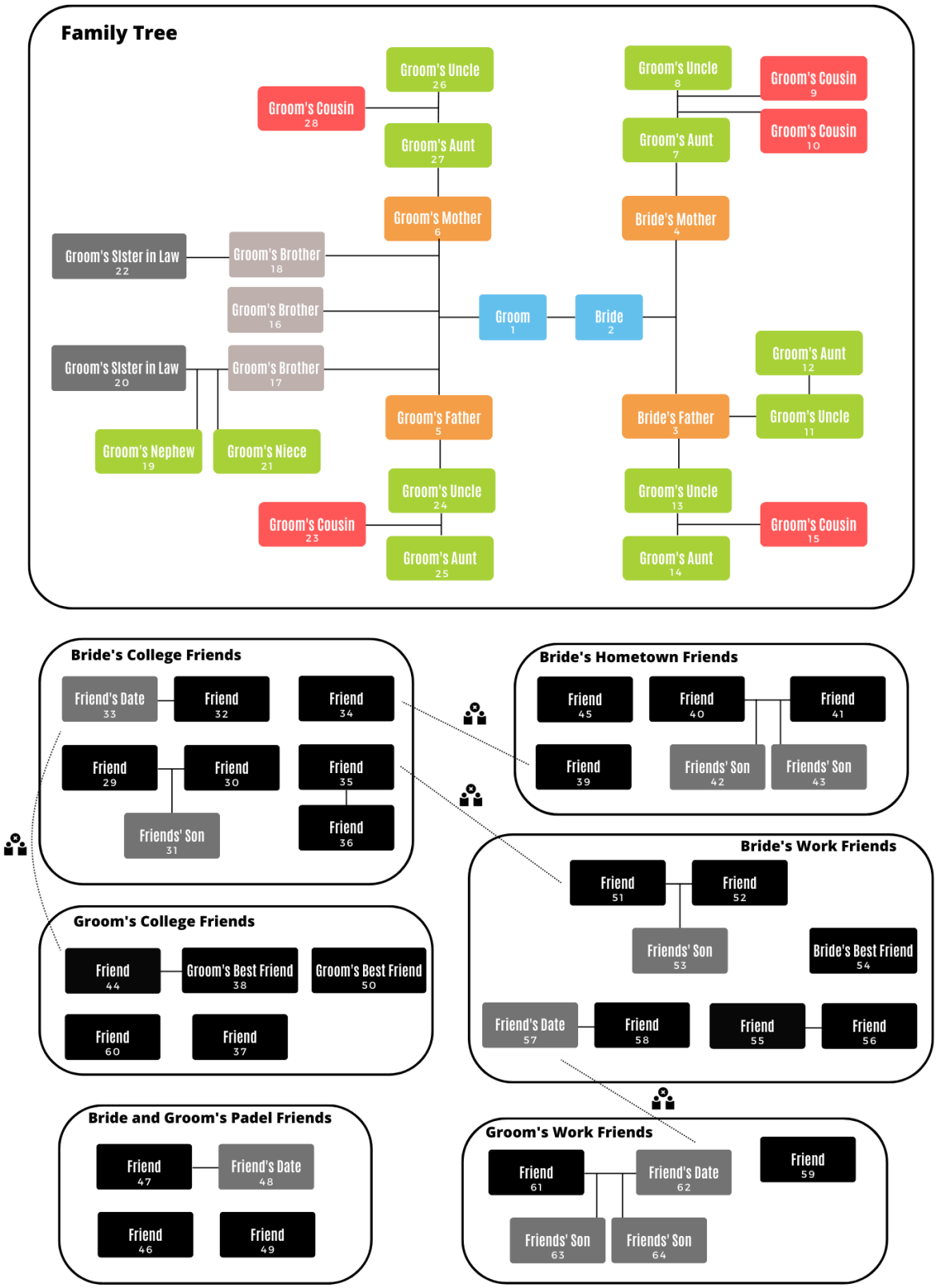

Fig. 1 *- Guests and their relationships*

Relationship	Value
Bride or Groom	5000
Spouse or Date	2000
Best Friend	1000
Siblings	900
Parent or Child	700
Cousin	500
Aunt/Uncle or Niece/Nephew	300
Friend	100
Strangers	0
Enemies	-1000

Table 1 *- Pairwise Relationship Matrix Values*

Crossover operators

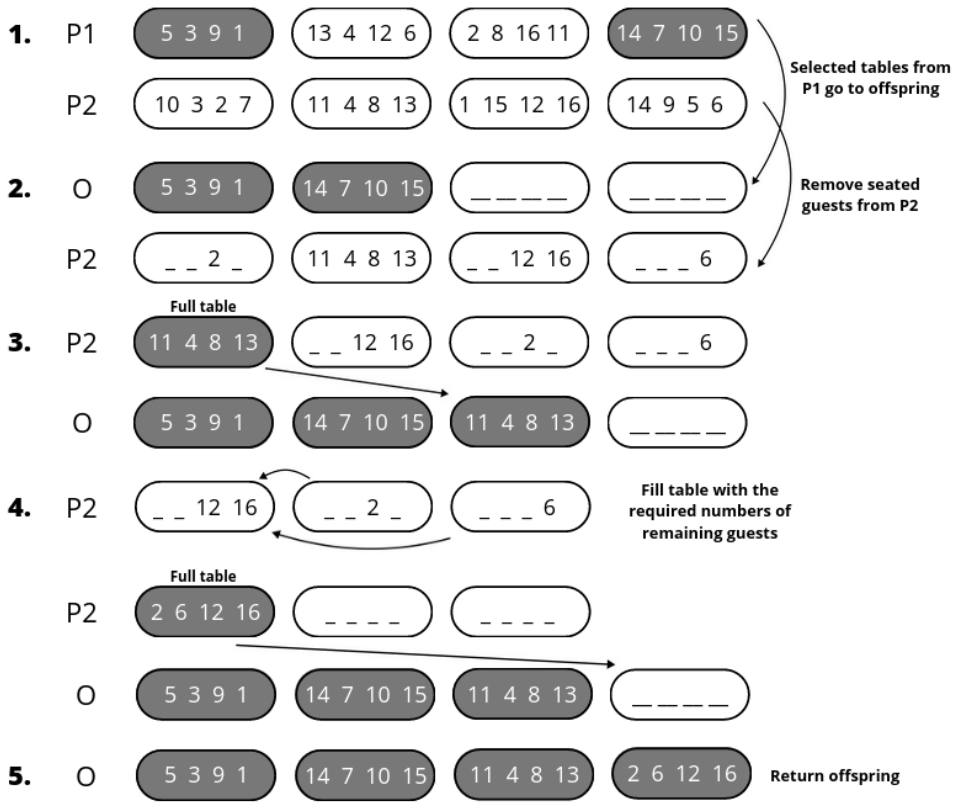

Fig. 3 *- Group Based Crossover*

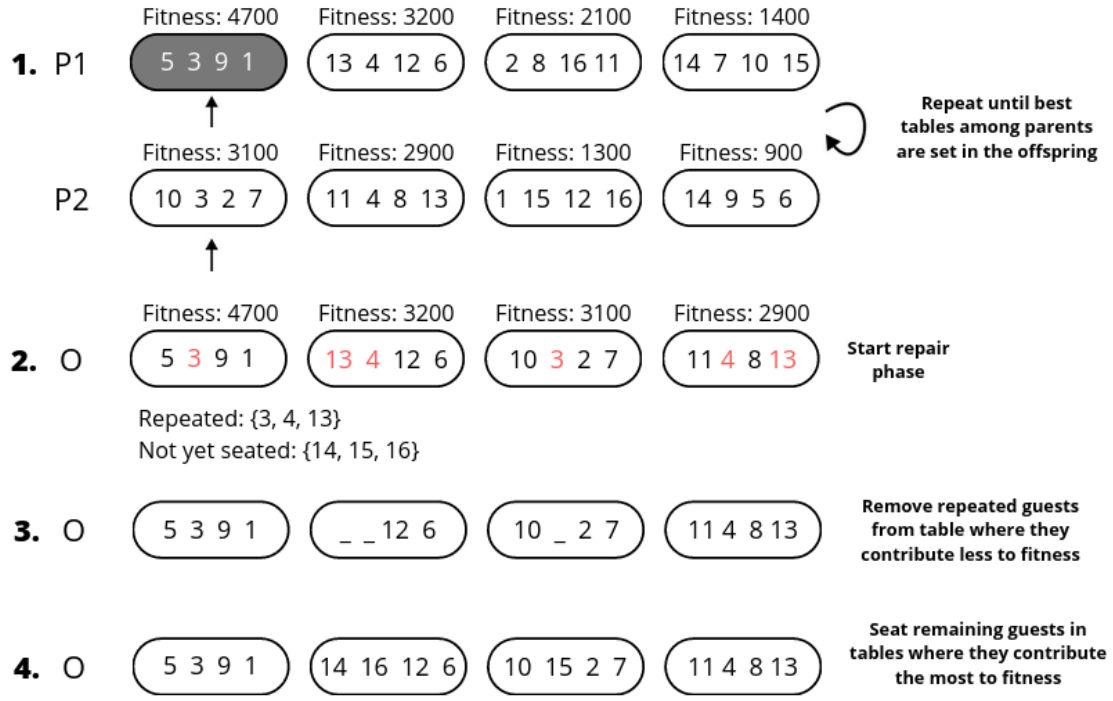

Fig. 4 *- Eager Breeder Crossover*

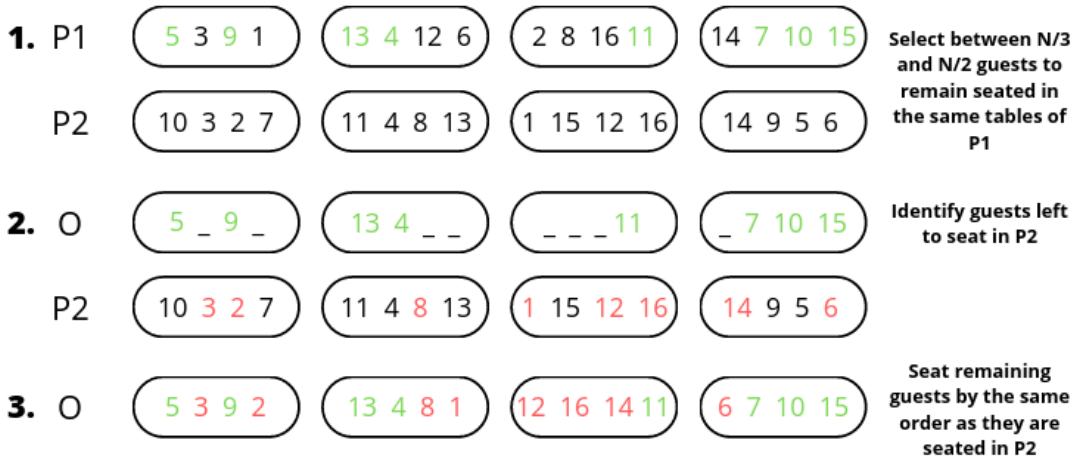

Fig. 5 *- Twin Maker Crossover*

Mutation Operators

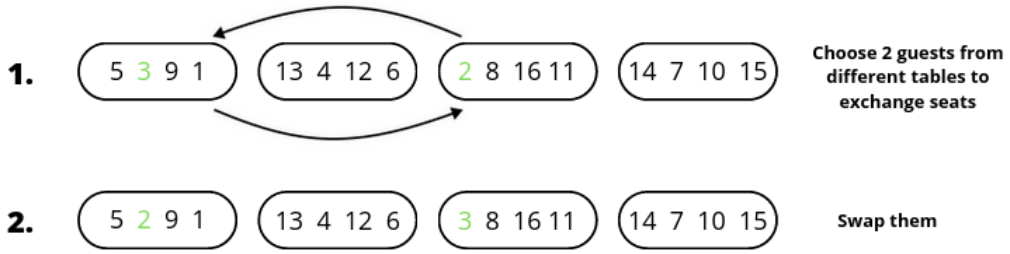

***Fig. 6** - Swap Mutation*

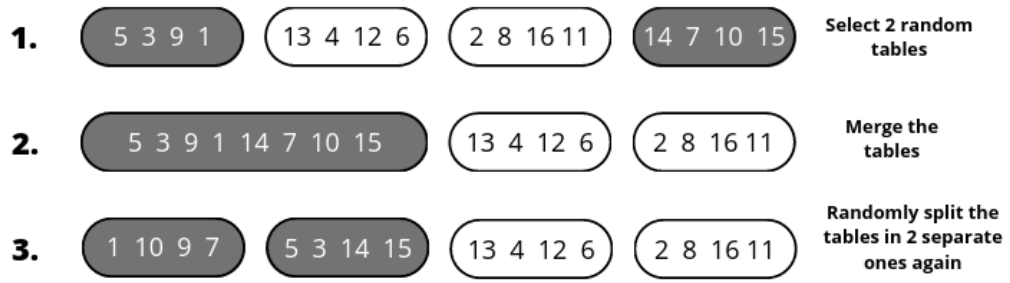

***Fig. 7** - Merge and Split Mutation*

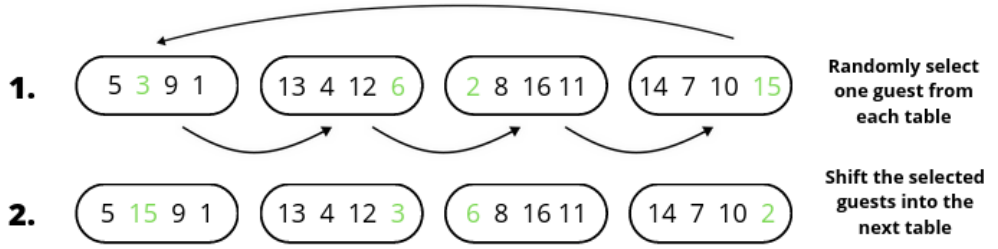

***Fig. 8** - The Hop Mutation*

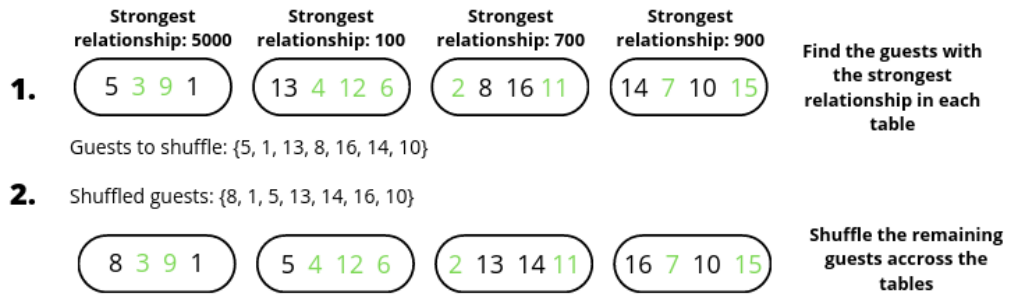

***Fig. 9** - Dream Team Mutation*

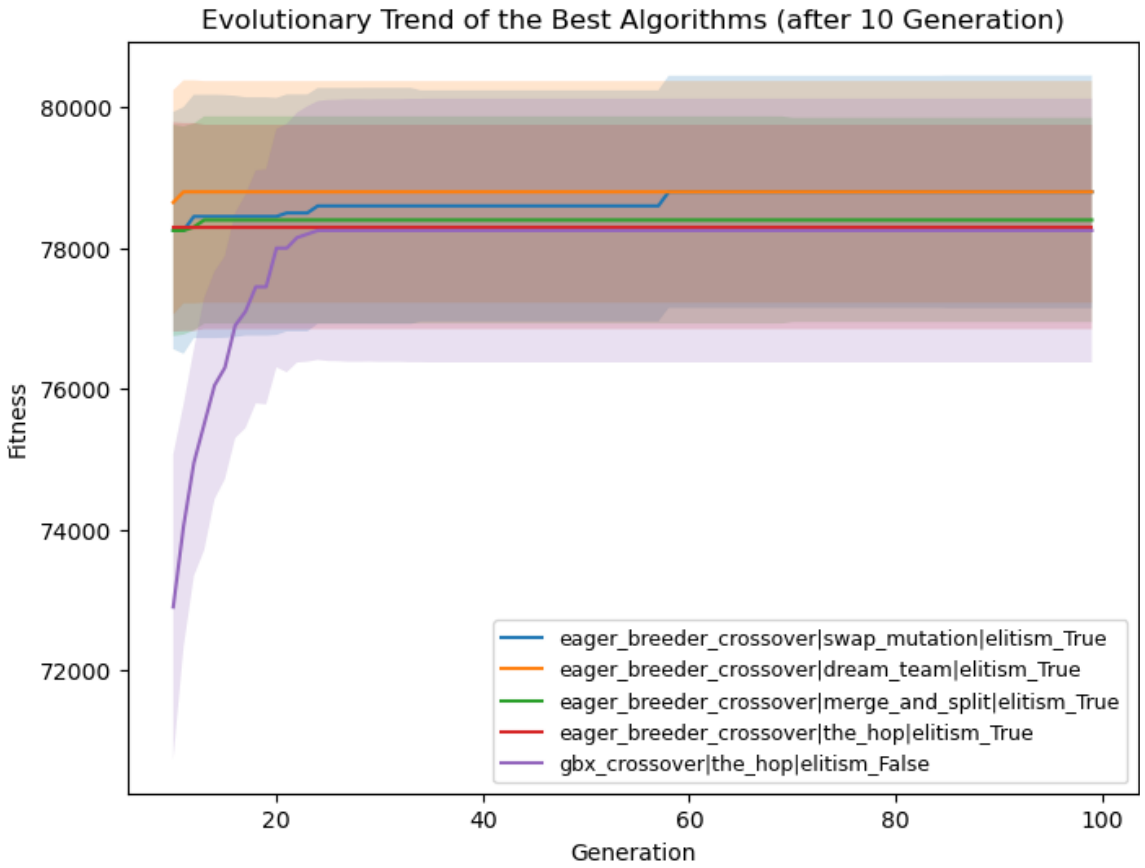

Fig. 11 - *Median best fitness of top 5 algorithms after the 10th generation*