Time Optimization with Superpermutations Using Evolutionary Computation

Lobera del Castillo, José Luis   
Ruíz Capetillo, Rafael Andrade  
Optimización y Metaheuristicas IIUniversidad Panamericana  
Aguascalientes, México

*Abstract*— A permutation is an arrangement of elements from a set into a sequence.

A superpermutation is a string that contains each permutation of n symbols as a substring. Superpermutations are a vital tool in minimization problems that involve permutations, they have numerous applications and using them is a big area of opportunity because they have not had the attention that they should.

Evolutionary Computation is an approach to Optimization problems which brings us closer to a solution as individuals evolve through generations.

Keywords—Optimization, Metaheuristics, Evolutionary Computation, Genetic Algorithm, Superpermutation.

# Introduction

In the year 2011 an online debate was held in the online bulletin board 4chan, the debaters discussed the best way to watch the Japanese TV show *The Melancholy of Haruhi Suzumiy*. Its first season, which involved time travel, was not aired in chronological order, so each viewers saw the TV show in a different order. Years later a re-broadcast and a DVD version came out, now in chronological order, but something about the correct order to watch the show did not feel right to the fans, so they started discussing the orders in which they saw the show, and which one was the best arrange of episodes.

After hours of online discussions, a fan asked a question that, without knowing, would propose a very interesting optimization problem:

If we were to watch all the possible arranges of all 14 episodes of the show to finally decide which one is the best way to see them… **how long would it take to watch them all?**

In just a few hours after the question was published a mathematician fan answered:

If there are 14 episodes and we want to watch every possible arrange of them we would have to watch 87,178,291,200 combinations, a total of 1,220,496,076,800 episodes. Considering that each episode length is around 10 minutes long, we would need around 12,204,960,768,000 minutes or 23 million years, sadly we won’t be there to find out the answer.

So, the discussion ended, and the fans accepted that they would never find out the best way to watch the show, little did they know that the answer to their debate would be hidden in a single (very long) arrange which contained every possible combination inside of it.

# Permutations and Superpermutations

A **permutation** is an arrangement of elements from a set into a sequence, imagine for example the set , a permutation of set A would be **“132”**. To calculate the total of possible permutations of a set we use the equation , where n is the length of the set, so in this case our set would have a total of 6 possible permutations.

A **superpermutation** is a string which contains all possible permutations of a given set inside of it as **substrings**.

The easiest way to calculate a superpermutation is to sum each possible permutation as so:

A picture containing text

Description automatically generated

The optimization begins when we start removing elements that repeat and we mix the permutations, for example with two-element long permutations instead of having “1221” we can reduce it to just “121”. So, the three-element long superpermutation would be:



But the optimization does not stop there, mathematicians turned to the *Traveling Salesman Problem*, which looks for the least expensive route through a collection of cities, minimizing the superpermutation, for example:

Graphical user interface

Description automatically generated with low confidence

# Evolutionary Computation

In computer science, evolutionary computation is a family of algorithms for optimization inspired by biological evolution. They are an approach to metaheuristic problems based on trial and error.

In evolutionary computation, an initial set of candidate solutions is generated and iteratively updated. Each new generation is produced by stochastically removing less desired solutions and introducing small random changes. In biological terminology, a population of solutions is subjected to natural selection and mutation. As a result, the population will gradually evolve to increase in fitness, in this case the chosen fitness function of the algorithm.

# Genetic algorithm

Genetic algorithms are commonly used to generate solutions to optimization and search problems by relying on biologically inspired operators such as mutation, crossover, and selection.

* Mutation: A small change to an existing solution to make it better.
* Crossover: Also called Reproduction is a random selection of genes from two different solutions to make a new better one.
* Selection: Is the process that selects two solutions from the population to call the crossover function.

In every optimization problem there must be an Objective Function which will evaluate a solution and return a score to rate the effectiveness of each member of the population.

# The Mary Movie Montage

The years have passed, and the *Haruhi* fans community has not decided the right way to watch the TV series, but in 2021 the online community of data scientists and machine learning practitioners *Kaggle* published their annual Christmas challenge and, to the surprise of the participants, a variation of the *Haruhi* problem was proposed: *The Mary Movie Montage*.

**Challenge Description**

People seem to be getting in the Christmas spirit earlier and earlier each year. Decorations appear for sale in stores in the fall, Christmas songs are on the radio in October…

The Elves at the North Pole are starting to recognize this and need to work as fast as possible to launch their latest holiday offering: SantaTV+! A 24/7 streaming television channel where it’s “Always Christmas, All the Time.” To debut their new station, they’ve decided to kick things off with a made-for-television Christmas movie marathon! They’re excited for the premiere of such movies as 🎅, 🤶, 🦌, 🧝, 🎄, 🎁, and 🎀!

But elves know that just as important as the movie themselves is the order they’ll be aired. So, the elves have decided the best way to figure out which order is best is to watch all the movies in every possible combination to see which feels the most Christmas-y.

Your job is to help the elves by giving them the shortest viewing schedules that shows them every combination of movies so they can get SantaTV+ live as soon as possible! The elves have formed three movie-watching teams to lighten the load, so every combination must be seen by at least one of their groups. But they’re also pretty sure they want to kick off the movie marathon with the 🎅 and 🤶 movies back-to-back, so be sure that each group has all the combinations that start with those. And finally, the elves have agreed to two sugar breaks, so you’re allowed to give each group up to two 🌟 wildcards, which will play all the movies at once while they’re snacking, which will help speed things along.

They can’t launch SantaTV+ until all the groups have finished watching - so help give them the most efficient schedule to see every Christmas movie combination and help them get back to making toys!

**Challenge Objective**

Your objective is to find a set of three strings containing every permutation of the seven symbols 🎅, 🤶, 🦌, 🧝, 🎄, 🎁, and 🎀 as substrings, subject to the following conditions:

* Every permutation must be in at least one string.
* Each permutation beginning with 🎅🤶 must be in all three strings.
* Each string may have up to two wildcards 🌟, which will match any symbol in a permutation. No string of length seven containing more than one wildcard will count as a permutation.

Your score is the length of the longest of the three strings. This is a minimization problem, so lower scores are better.

**Challenge Example**

Let's consider a simplified problem where we only use three symbols 🎅, 🤶, 🦌and no wildcard, and where our solution consists of only two strings.

There are six permutations of these three symbols: 🎅🤶🦌, 🎅🦌🤶, 🤶🎅🦌, 🤶🦌🎅, 🦌🎅🤶, and 🦌🤶🎅. The permutation 🎅🤶🦌 must be a substring of both solution strings while the other five permutations must be in at least one of the strings.

A valid solution for this problem is:

1. 🤶🎅🦌🤶🎅🤶🦌
2. 🎅🤶🦌🎅🤶

which would have a score of 7, the length of string 1.

If we were allowed the use of one wildcard, we could have the solution:

1. 🎅🤶🌟🦌🤶🎅
2. 🎅🤶🦌🎅🤶

with a score of 6. The wildcard can represent different symbols in different permutations.

**Challenge Dataset**

The challenge provides the participants with four **Comma Separated Values** (CSV) documents:

1. *distance\_matrix*: Contains the distance between every permutation assigning a score from 0-7.
2. *permutations*: Contains every permutation of the seven symbols.
3. *sample\_submission*: Contains an example of how the solution must be handed.
4. *wildcards*: Contains seven permutations for every permutation with a wildcard.

# Analysing the problem

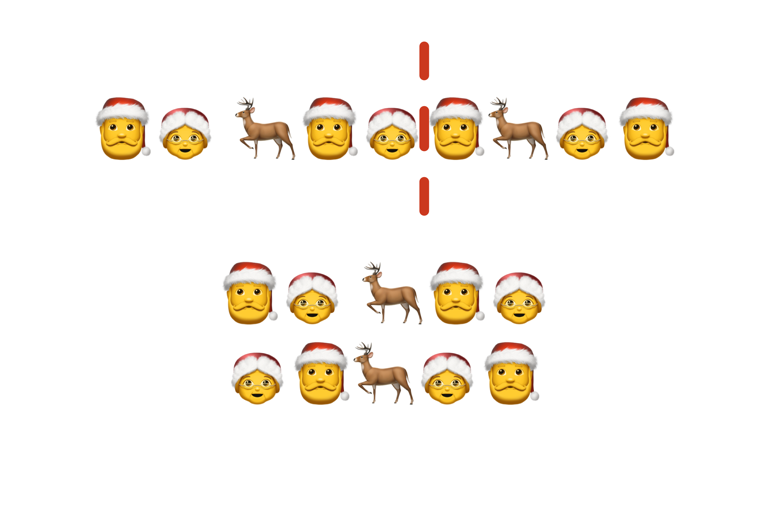
*The Mary Movie Montage* challenge is an optimization problem, the objective is to minimize the length of three strings which must meet the requirements stablished before.

**Random Approach**

To minimize those three strings, we must first generate them, the easiest way (but not the optimal) is to randomly create solutions and keep the ones that meet the conditions, but this would take a lot of computation time just to create a valid solution.

**Superpermutation**

Here is where superpermutations enter the game, if we could generate a seven-element superpermutation then we would secure a valid solution where at least every permutation exists inside the superpermutation. As the problem states that we can have three strings as a solution and we must minimize the length of them, the solution would be to find the best points to split the string, for example:



This solution would be valid, but we. Must consider that each permutation beginning with 🎅🤶 must be in all three strings so we could not just use one superpermutation. What we could do is to copy the superpermutation three times, so that we met the condition.

The first thing we could do, now that we have a valid solution that we know is valid for sure, is to search for the values that we can erase without affecting the result.

We also must consider the wildcards; remember we can place 2 wildcards per string so we must search the best 2 indexes to place them.

Finally, as the wildcards can represent any value, there might be more indexes that we can remove, so we must make a final cleaning cycle.

Once we erased all possible indexes and added wildcards the solution must be ready.

An important thing we must have in mind is that the effectiveness of the solution depends on how many generations and individuals we create, but the more we add the more computation time the algorithm requires.

# The Genetic Algorithm Approach

Now that we have stablished a good approach, we are ready to incorporate an Evolutionary Programming solution. The genetic algorithm is going to be the strategy that will help us solve this challenge.

The first thing we must do when working on an optimization problem is to define an objective function, in this case we will have two different functions, one to erase indexes and one to place the wildcards, both functions take as parameter a Dataframe with only one column “schedule” and three rows, each with a string containing permutations of the substring.

* *delete\_nodes\_objective\_func*: Checks if every permutation exists at least in one string and verifies that each permutation beginning with 🎅🤶 exists in all three strings.
* *count\_nodes\_objective\_func*: Replaces each wildcard with every movie symbol and counts the number of times a permutation appears.

Once the objective functions were ready, we created a base phenotype, which is basically the shortest superpermutation of seven symbols repeated three times.

Then we defined the Individual class. An Individual is a single solution, in this case a list of indices to remove from the base phenotype. The Individual class has some attributes and methods that make the evolution process a little bit easier to code:

* *genotype:* is a list o f three sets, each representing a string from the solution, the sets contain the indices that will be removed from the base phenotype.
* *get\_phenotype()*: Transforms the genotype to a Dataframe to be evaluated.
* *add\_genes()*: Inserts a number inside a set of the genotype, it can be random, or the index and value can be passed as arguments.
* *get\_fitness():* Calls the objective function to check if the solution is valid.
* *mutate():* Calls the *add\_genes()* function to add a random value in a random set.
* *get\_local\_fitness():* return the number of indices the solution can remove.

*Table

Description automatically generated with low confidence*

The second class we defined is the Population class. The population class is in charge of the evolution function process:

* *individuals:* is a list of n objects of type Individual
* *get\_survivors():* From the individuals list returns a list with only the Individuals which contain a valid solution to the objective function.
* *crossover():* Takes two individual parents as a parameter and returns a new one with genes from both parents.
* *mutation():* calls the *mutate()* function of every individual.
* *new\_population():* Creates a new generation of individuals calling the crossover n times and mutating those new individuals.

Graphical user interface, text, application

Description automatically generated

After the classes were ready it was time to test the genetic algorithm, so we cycle the *new\_population()* function g times with a population of p.

Now it’s time to find the best places for the wildcards, the process was very similar, we created a Wildcards class:

* *wildcards:* A list of (3,2) indices where the wildcards should go.
* *update\_gene():* This function changes the value of an existing wildcard
* *get\_phenotype():* Transforms the wildcards list to a Dataframe to be evaluated.
* *mutate():* Changes a random wildcard value.
* *get\_fitness():* Evaluates the phenotype in the objective function.

Just like the Individual population, we have a wildcards population with the same methods to evolve the wildcards.

Once the wildcards evolution is completed, we just must run function one last time, but this time with an objective function that checks both wildcards and deleted indices.

At the end we should have the best solution for the minimization problem and the watching schedule would be ready for elves to stream the movies and finally decide which order feels more Christmas-y.

# Results

To obtain a valid solution only requires one generation and few Individuals, the more generations and individuals will generate better solutions, but will also slow down the process, as the computer needs more processing time. To compute the global solution the next parameters were needed:

* For the first step, which searches for indices to delete:
  + Population size of 20 Individuals
  + 5000 Generations
  + A crossover range from 1 to 7 indices to inherit

This step took around 1 hour and a half to get the best result.

* For the second step, which finds the best indices to place the wildcards:
  + A population of 10 Individuals
  + 1000 Generations

This step also took around 1 hour and a half.

* Finally, the third step to remove the last indices:
  + Population size of 10 individuals
  + 1000 Generations
  + A crossover range from 1 to 3

The final step was faster as it only took 6 minutes to process.

The resulting Dataframe had the following wildcards and lengths:

* First string: [95, 300], length=2428
* Second string: [116, 523], length=2428
* Third string: [339, 544], length=2428

This means that the elves need to stream a total of 7284 movies to watch all possible permutations.

The average length of a movie is around 2 hours, so each group of elves must watch the movies for 202 days and eight hours straight to finally determine the most Christmas-y way to watch all movies instead of 8 years 2 weeks and 6 days.

# Superpermutation applications

As we saw on the results, superpermutations can be a great strategy to minimize processes that involve permutations, but superpermutations have more applications than just the *Haruhi* problem.

Problems that include permutations can be hard to understand, but sometimes we do not realize that permutations are everywhere, the password on physical locks, virtual passwords, phone numbers, car plate numbers, music notes and words.

Superpermutations are a great way to sort random elements from a set without the result being to repetitive.

For example, in 2020, when the Covid-19 pandemic arrived at Mexico, the classes where suspended and paused because most families did not have an electronic device for them to connect to one or more virtual classes at the same time.

Public schools started streaming their classes to open TV channels, so the kids at home could take their lessons, but they depended on the steaming hour.

Superpermutations would have helped if they streamed the lessons in a superpermutation sequence, because you don’t need a chronological order to study the year’s subjects. The kids would not have to connect at any specific hour, because no matter at what time they turn on the TV, they will be able to see all their classes.

A picture containing text

Description automatically generated

Another application for superpermutations has to do with buffers when transmitting information from a device to another. Electric doors cannot be open with any controller, you need the one that transmits the specific password for your door, but when clicked, the control sends a code to the door and the door is always listening for codes, so if you send any code the door will receive it.

You can try to send all possible combinations (the standard is a permutation of length 6), but that would take around 4.4 minutes considering that most electric doors receive packages of 12-bits.

The secret here is hidden inside of the door’s engine microchip, which receives packages not, one by one, but as a queue removing the first element and letting the next enter at the end as so:

Chart

Description automatically generated with medium confidence

With this method you can open almost any door in just 8 seconds.

# Conclusion and improvement proposal

As we saw on the real-life applications, those and more problems can be optimized with superpermutations.

Thought is a hard topic to understand, superpermutations can be a powerful approach to problems.

Evolutionary programming is a great proposal to solve optimization problems, they work perfectly when there is a need to solve a problem that a classic algorithm can’t compute.

Even though machine learning algorithms and neural networks can replace the algorithms, data scientists are using evolutionary computing to select the best fitted models and to create new ones based on mutation and crossovers.

An improvement proposal, the objective function can be optimized, although we made it work relatively fast there are more tools that must make it faster. Optimizing these functions would reduce the computation time a lot, because right now the algorithm spends most of its time evaluating the result of solutions.

##### Acknowledgment

To Dr. Claudia Nallely Sánchez Gómez for everything you have taught us during these 4 years, for your infinite support and guidance. Thank you.

##### References

1. Johnston N. (2013). *The Minimal Superpermutation Problem.* Canada: Mount Allison University.
2. Johnston, N. (2013). Non-Unique Secuences of Minimal Superpermutations, , *Discrete Mathematics*, United States: Cornell University
3. Houston, R. (2014). *Tackling the Minimal Superpermutation Problem.* United States: Cornell University
4. Kaggle. (2021). *The Mary Movie Montage.* <https://www.kaggle.com/code/skril31/generating-a-2428-solution-in-100-lines-of-code/data?select=submission.csv>
5. Wikipedia. (2022). *Permutation*. <https://en.wikipedia.org/wiki/Permutation>
6. Wikipedia. (2022). *Superpermutation*. <https://en.wikipedia.org/wiki/Permutation>
7. GregEgan. (2019). *Superpermutations*. November 7th 2022. <https://www.gregegan.net/SCIENCE/Superpermutations/Superpermutations.html>
8. Wired. (2018) *How an Anonymous 4chan Post Helped Solve a 25-Year-Old Math Puzzle.* <https://www.wired.com/story/how-an-anonymous-4chan-post-helped-solve-a-25-year-old-math-puzzle/>
9. Quanta Machine. (2019). *Unscrambling the Hidden Secrets of Superpermutations.* <https://www.quantamagazine.org/unscrambling-the-hidden-secrets-of-superpermutations-20190116/>
10. Medium. (2020). *Partial Superpermutation-Personal Research*. <https://medium.com/maths-dover/partial-superpermutation-personal-research-3e190ff067df>