# GA analise

## Genetic algorithem data EDA

### Authors: Chen Bistra, Yasmin Reich

### Discription:

We have developed a Genetic Algorithm (GA) specifically designed to break a mono-alphabetic cipher. Our GA follows these key guidelines:

Initialization: We begin by creating a random population of individuals, the size of which is determined by the given population size.

Evaluation: The fitness function plays a crucial role in evaluating each individual's fitness. We utilize a dictionary to check if a word exists or how closely it resembles a word in the dictionary, using the Levenshtein distance as a measure. And the 2_freq file to give the score for the too "far" words.

Next Generation: The next generation function handles the selection of individuals for the next generation. Initially, natural selection takes place based on a specified death threshold. The top 5 individuals are directly transferred to the next generation. Crossover is then performed between the remaining population members and selected individuals to fill the desired population size. Finally, mutations are introduced according to a predefined mutation rate.

Termination: The algorithm terminates when the best result remains unchanged for 10 consecutive generations. This criterion ensures that further iterations are not necessary as the algorithm has reached a stable solution.

By employing these steps, our GA algorithm effectively breaks mono-alphabetic ciphers by iteratively improving the population through selection, crossover, and mutation, ultimately converging towards the best solution.

The Lamarckian and Darwinian GAs incorporate the inclusion of n local operations, as specified by the given parameter. In both approaches, these operations are applied to assess whether they enhance the fitness of the individuals. In the Lamarckian approach, if an operation improves fitness, the individual itself is updated with the new changes. However, in the Darwinian approach, only the fitness value is updated while the individual remains unchanged. This distinction allows for a comparison between the direct modification of individuals in the Lamarckian GA and the modification of fitness values in the Darwinian GA.

### Analize:

To analyze the performance of our Genetic Algorithm (GA), we conducted a series of runs with different parameter settings. The parameters we considered are as follows:

For the regular GA:

Population Size: [40, 60, 80, 100] Mutation Chance: [0.2, 0.4, 0.6, 0.8] Death Threshold: [0.2, 0.4, 0.6, 0.8] Additionally, for the Darwinian addition, we introduced the parameter:

Local Operations: [1, 3, 5] For the Lamarckian addition, we included the parameter:

Local Operations: [5, 7, 10] The selection of these parameters was based on the observed ranges that demonstrated successful performance for our algorithm. As the GA involves some degree of randomness, we ran each configuration 10 times to ensure robustness in our analysis.

By systematically exploring these parameter combinations and conducting multiple runs, we aimed to gain insights into the behavior and effectiveness of our GA algorithm in breaking the mono-alphabetic cipher.

## Find the best parameters:

### REGULAR GA:

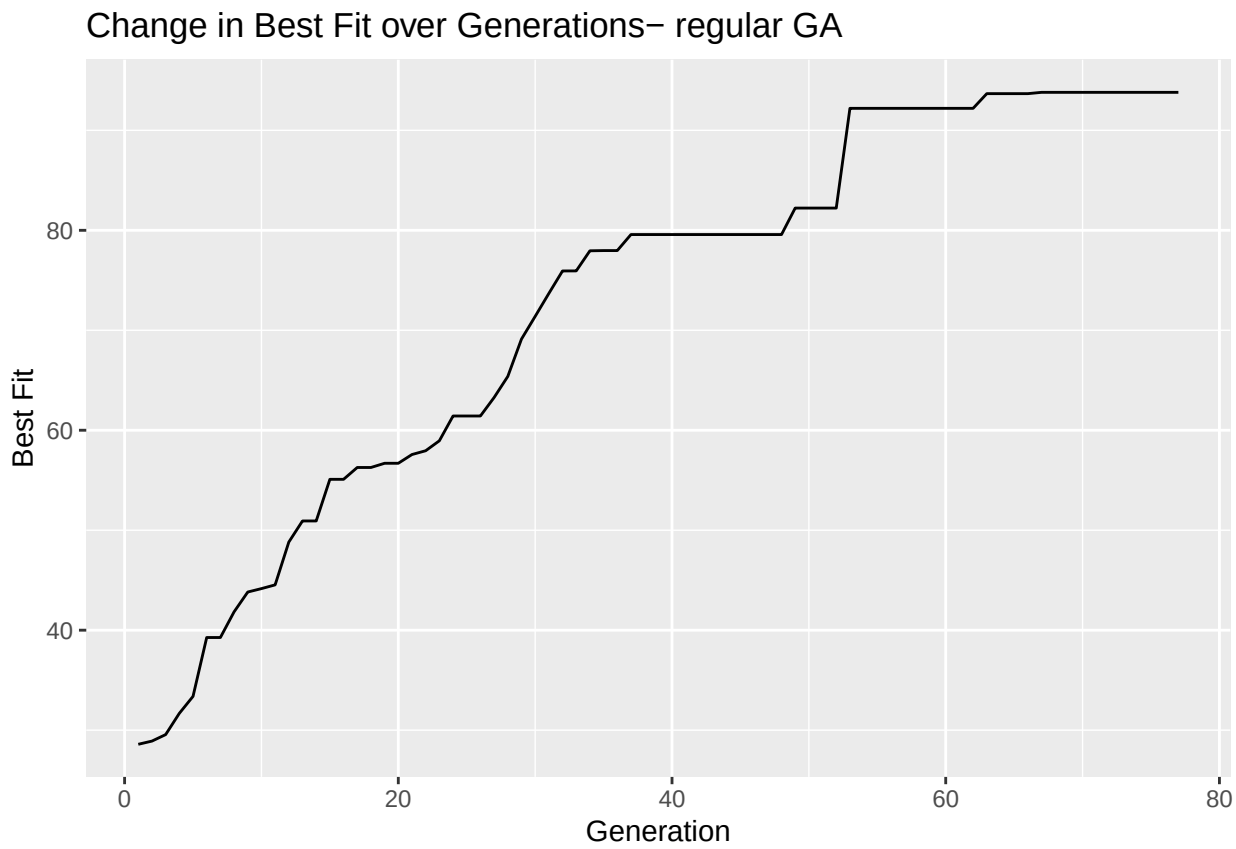
After running the algorithm with the aforementioned parameters, we collected all the relevant information and stored it in a CSV file. Our objective was to identify the parameter combinations that yielded the best fitness while minimizing the number of fitness function evaluations required. To enhance the running time, we divided the data set into separate files.

```
file_list <- c("C:\\Users\\User\\.vscode\\comp_bio\\comp_bio_targil2\\regular_0.2.csv",
               "C:\\Users\\User\\.vscode\\comp_bio\\comp_bio_targil2\\regular_0.4.csv",
               "C:\\Users\\User\\.vscode\\comp_bio\\comp_bio_targil2\\regular_0.6.csv",
               "C:\\Users\\User\\.vscode\\comp_bio\\comp_bio_targil2\\regular_0.8.csv")

combined_data <- do.call(rbind, lapply(file_list, read.csv))

combined_data$best_fit <- as.numeric(combined_data$best_fit)

sorted_data <- combined_data %>%
  arrange(desc(best_fit), calls_to_fit)

best_fit_max <- max(sorted_data$best_fit)
filtered_data <- sorted_data %>%
  filter(best_fit == best_fit_max)

best_rows <- filtered_data[filtered_data$calls_to_fit == min(filtered_data$calls_to_fit),]

knitr::kable(best_rows)
```

| population_size | motation_chance | death_treshold | generation_num | calls_to_fit | best_fit |
|---|---|---|---|---|---|
| 40 | 0.4 | 0.8 | 65 | 2588 | 93.96698 |

the graph of a run with these parameters:

```
# Read the CSV file
data <- read.csv("C:\\Users\\User\\.vscode\\comp_bio\\comp_bio_targil2\\regular best graph.csv")

# Create a line plot
ggplot(data, aes(x = generation, y = best_fit)) +
  geom_line() +
  labs(x = "Generation", y = "Best Fit") +
  ggtitle("Change in Best Fit over Generations- regular GA")
```




The GA demonstrates its ability to overcome several local maximums and achieve convergence towards the global maximum. Throughout the optimization process, the GA explores different solutions and gradually improves the fitness until it reaches the optimal solution. This behavior is indicative of the GA's effectiveness in navigating complex search spaces and finding the best possible solution.

Now we are interested in identifying the best parameters that consistently produce the best results across all runs. We want to find the parameter set that yields the best average performance, in low variance between the results.

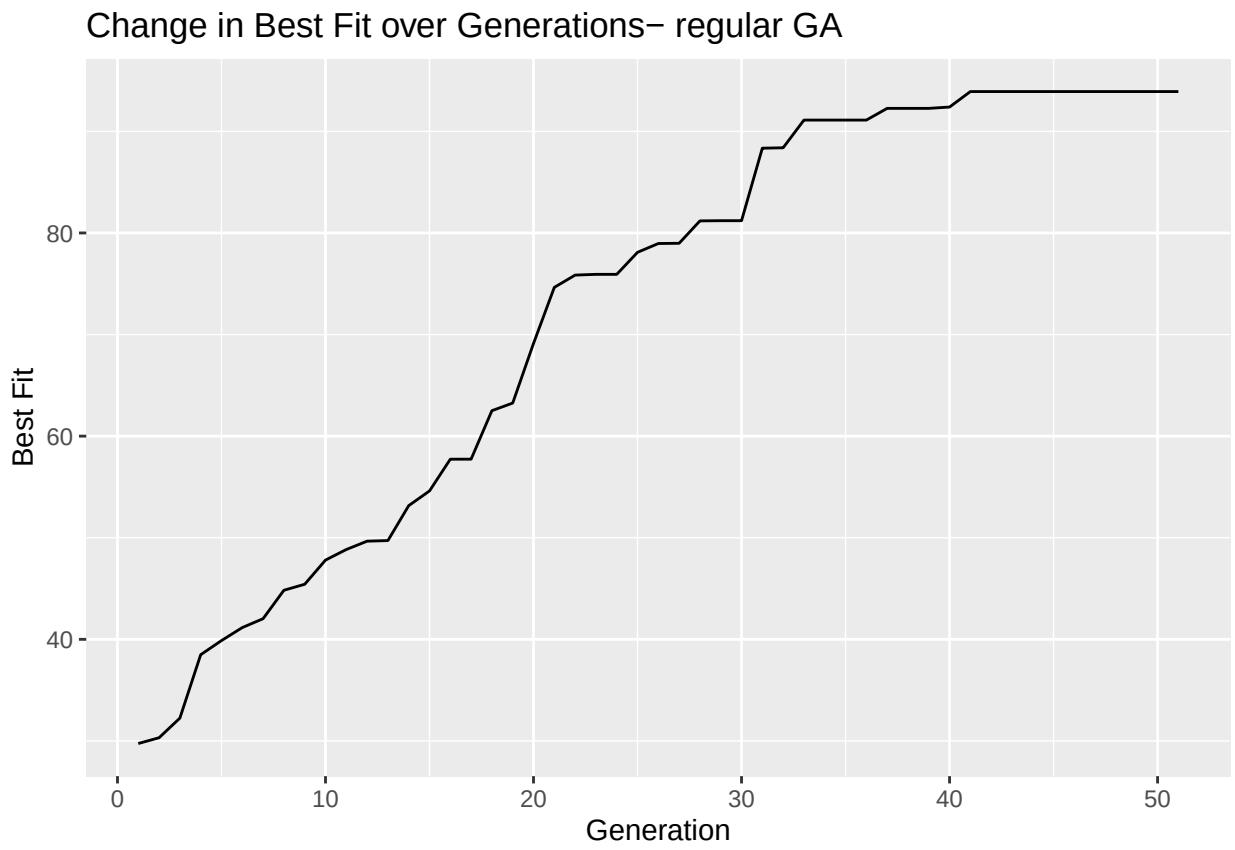
```
knitr::opts_chunk$set(results='asis')
best_combination <- combined_data %>%
  group_by(population_size = combined_data$population_size,
           mutation_chance = combined_data$motation_chance,
           death_threshold = combined_data$death_treshold) %>%
  summarize(
    avg_gen = mean(generation_num),
    avg_fit = mean(best_fit),
    fit_var = var(best_fit),
    avg_fit_calls = mean(calls_to_fit)
  ) %>%
  arrange(desc(avg_fit), avg_fit_calls)

best_row <- best_combination[1, ]
knitr::kable(best_row)
```

| population_size | mutation_chance | death_threshold | avg_gen | avg_fit | fit_var | avg_fit_calls |
|---|---|---|---|---|---|---|
| 80 | 0.8 | 0.4 | 62.7 | 93.96698 | 0 | 7119.1 |

Here, we observe that the algorithm consistently converges to the best fit on every run, as indicated by a variance of 0.

the graph of a run with these parameters:

```
# Read the CSV file
data <- read.csv("C:\\Users\\User\\.vscode\\comp_bio\\comp_bio_targil2\\regular amin graph.csv")

# Create a line plot
ggplot(data, aes(x = generation, y = best_fit)) +
  geom_line() +
  labs(x = "Generation", y = "Best Fit") +
  ggtitle("Change in Best Fit over Generations- regular GA")
```




By using these specific parameters, we observe a reduction in the number of local maximums encountered during the optimization process. This phenomenon can be attributed to the parameter settings, which seem to facilitate a more cautious and conservative approach towards convergence.

**DARWIN GA:**

Now, our objective is to determine the optimal parameters for the Darwin GA. To achieve this, we examined which parameter combinations resulted in the best run.

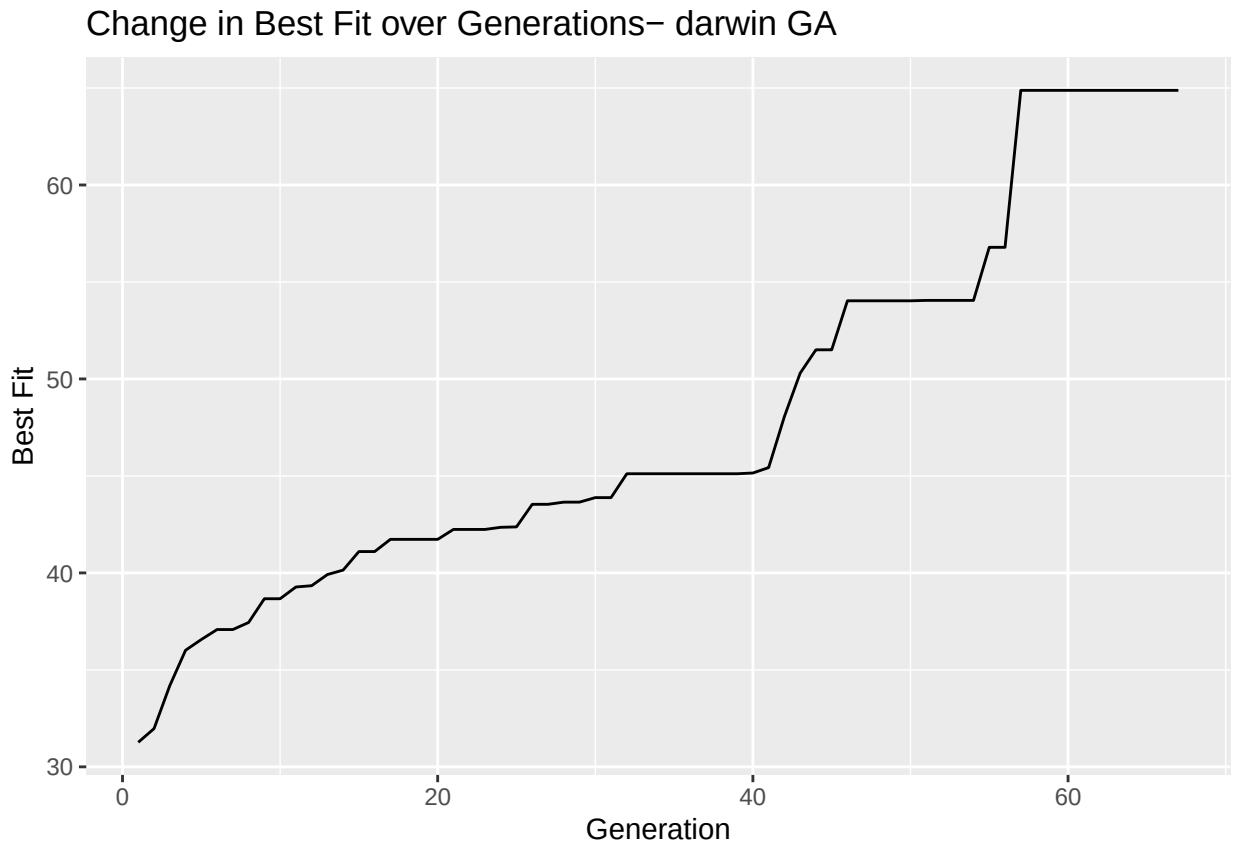
```
file_list <- c("C:\\Users\\User\\.vscode\\comp_bio\\comp_bio_targil2\\darwin_0.2.csv",
               "C:\\Users\\User\\.vscode\\comp_bio\\comp_bio_targil2\\darwin_0.4.csv",
               "C:\\Users\\User\\.vscode\\comp_bio\\comp_bio_targil2\\darwin_0.6.csv",
```

```
              "C:\\Users\\User\\.vscode\\comp_bio\\comp_bio_targil2\\darwin_0.8.csv")

combined_darwin_data <- do.call(rbind, lapply(file_list, read.csv))

combined_darwin_data$best_fit <- as.numeric(combined_darwin_data$best_fit)

sorted_darwin_data <- combined_darwin_data %>%
  arrange(desc(best_fit), calls_to_fit)

best_fit_max <- max(sorted_darwin_data$best_fit)
filtered_darwin_data <- sorted_darwin_data %>%
  filter(best_fit == best_fit_max)

best_darwin_rows <-
  filtered_darwin_data[filtered_data$calls_to_fit == min(filtered_data$calls_to_fit),]

knitr::kable(best_darwin_rows)
```

| population_size | motation_chance | n_opt | death_treshold | generation_num | calls_to_fit | best_fit |
|---|---|---|---|---|---|---|
| 100 | 0.2 | 5 | 0.4 | 71 | 8960 | 89.0174 |

the graph of a run with these parameters:

```
# Read the CSV file
data <- read.csv("C:\\Users\\User\\.vscode\\comp_bio\\comp_bio_targil2\\darwin best graph.csv")

# Create a line plot
ggplot(data, aes(x = generation, y = best_fit)) +
  geom_line() +
  labs(x = "Generation", y = "Best Fit") +
  ggtitle("Change in Best Fit over Generations- darwin GA")
```




The graph reveals that the algorithm converges prematurely before reaching the optimal result. It also exhibits numerous local maxima, which hinder the optimization process.

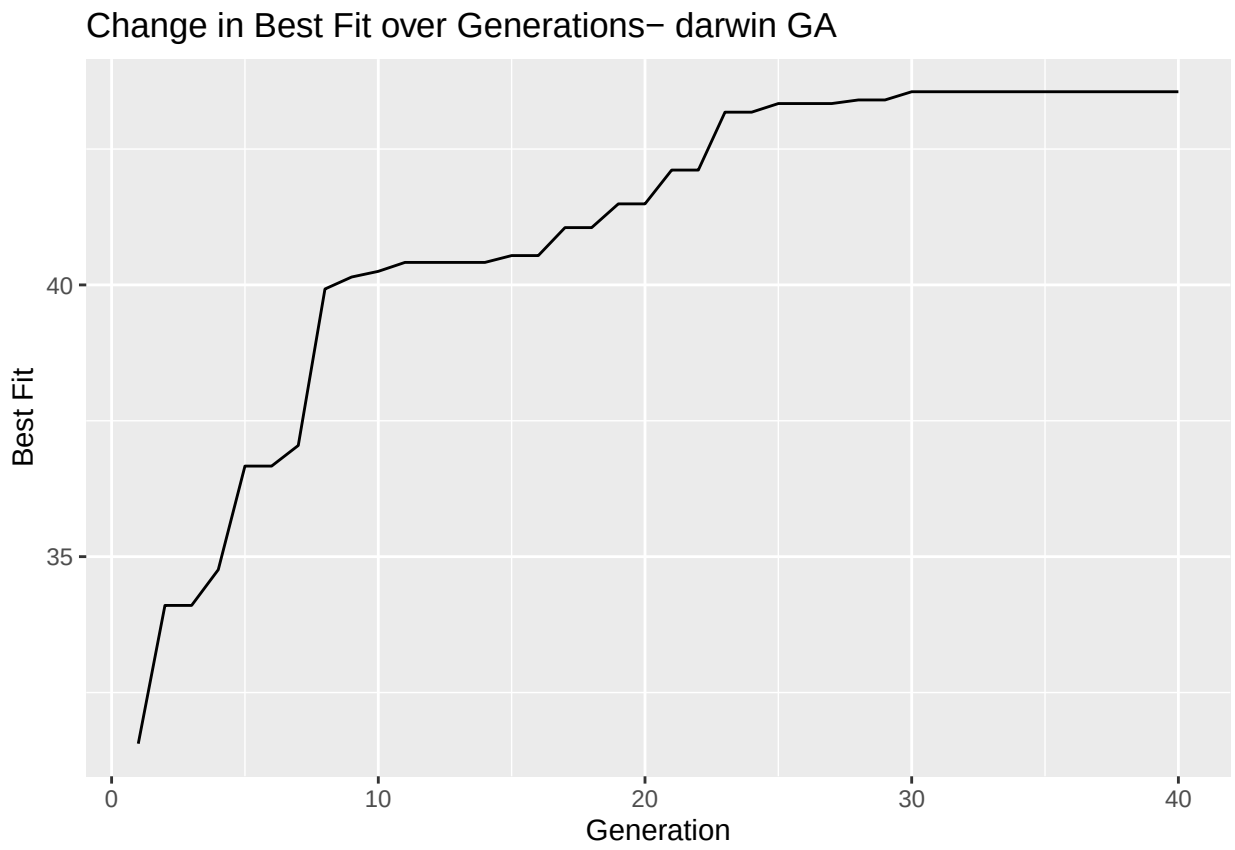
And again we are interested in identifying the best parameters that consistently produce the best results across all runs. We checked what is the parameters set that yields the best average performance, in low variance between the results.

```
best_darwin_combination <- combined_darwin_data %>%
  group_by(population_size = combined_darwin_data$population_size,
           mutation_chance = combined_darwin_data$motation_chance,
           death_threshold = combined_darwin_data$death_treshold,
           n_opt = combined_darwin_data$n_opt) %>%
  summarize(
    avg_gen = mean(generation_num),
    avg_best_fit = mean(best_fit),
    fit_var = var(best_fit),
    avg_fit_calls = mean(calls_to_fit)
  ) %>%
  arrange(desc(avg_best_fit), avg_fit_calls)

best_darwin_row <- best_darwin_combination[1, ]
knitr::kable(best_darwin_row)
```

| population_size | mutation_chance | death_threshold | n_opt | avg_gen | avg_best_fit | fit_var | avg_fit_calls |
|---|---|---|---|---|---|---|---|
| 100 | 0.4 | 0.2 | 1 | 57.66667 | 79.43616 | 71.93604 | 8649.333 |

the graph of a run with these parameters:

```
# Read the CSV file
data <- read.csv("C:\\Users\\User\\.vscode\\comp_bio\\comp_bio_targil2\\darwin amin graph.csv")

# Create a line plot
ggplot(data, aes(x = generation, y = best_fit)) +
  geom_line() +
  labs(x = "Generation", y = "Best Fit") +
  ggtitle("Change in Best Fit over Generations- darwin GA")
```




By observing the graph, we can infer that using these parameters leads to a smoother convergence path. However, even with the lowest number of optimizations, the algorithm fails to reach the best possible result.

**LAMERK GA:**

```
file_list <- c("C:\\Users\\User\\.vscode\\comp_bio\\comp_bio_targil2\\lamark_0.2.csv",
               "C:\\Users\\User\\.vscode\\comp_bio\\comp_bio_targil2\\lamark_0.4.csv",
               "C:\\Users\\User\\.vscode\\comp_bio\\comp_bio_targil2\\lamark_0.6.csv",
               "C:\\Users\\User\\.vscode\\comp_bio\\comp_bio_targil2\\lamark_0.8.csv")

combined_lamark_data <- do.call(rbind, lapply(file_list, read.csv))

combined_lamark_data$best_fit <- as.numeric(combined_lamark_data$best_fit)

sorted_lamark_data <- combined_lamark_data %>%
```

```
  arrange(desc(best_fit), calls_to_fit)

best_fit_max <- max(sorted_lamark_data$best_fit)
filtered_lamark_data <- sorted_lamark_data %>%
  filter(best_fit == best_fit_max)

best_lamark_rows <-
  filtered_lamark_data[filtered_data$calls_to_fit == min(filtered_data$calls_to_fit), ]

knitr::kable(best_lamark_rows)
```

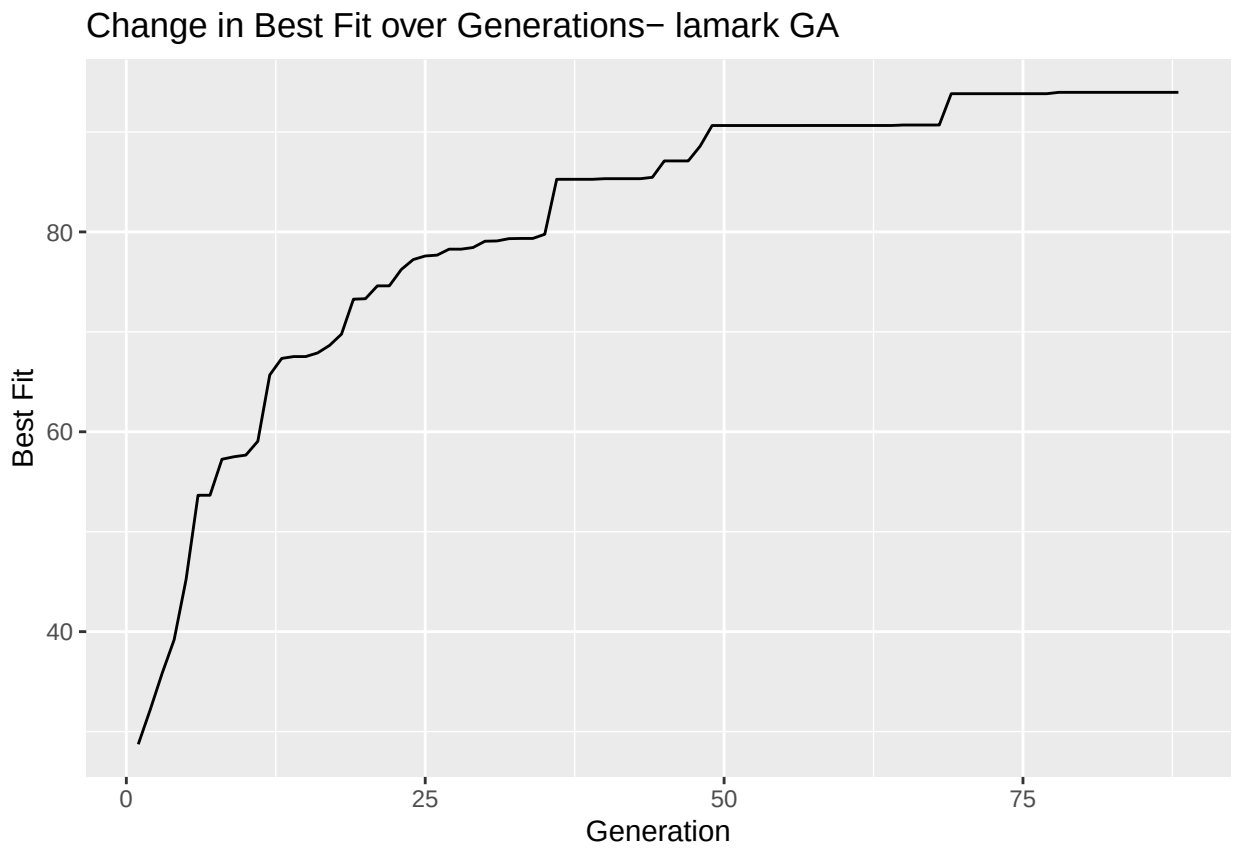
| population_size | motation_chance | n_opt | death_treshold | generation_num | calls_to_fit | best_fit |
|---|---|---|---|---|---|---|
| 40 | 0.6 | 10 | 0.6 | 63 | 3954 | 93.96698 |

```
best_lamark_combination <- combined_lamark_data %>%
  group_by(population_size = combined_lamark_data$population_size,
           motation_chance = combined_lamark_data$motation_chance,
           death_treshold = combined_lamark_data$death_treshold,
           n_opt = combined_lamark_data$n_opt) %>%
  summarize(
    avg_gen_num = mean(generation_num),
    avg_best_fit = mean(best_fit),
    avg_calls_to_fit = mean(calls_to_fit)
  ) %>%
  arrange(desc(avg_best_fit), avg_calls_to_fit)

best_lamark_row <- best_lamark_combination[1, ]
knitr::kable(best_lamark_row)
```

| population_size | motation_chance | death_treshold | n_opt | avg_gen_num | avg_best_fit | avg_calls_to_fit |
|---|---|---|---|---|---|---|
| 40 | 0.6 | 0.6 | 10 | 63 | 93.96698 | 3954 |

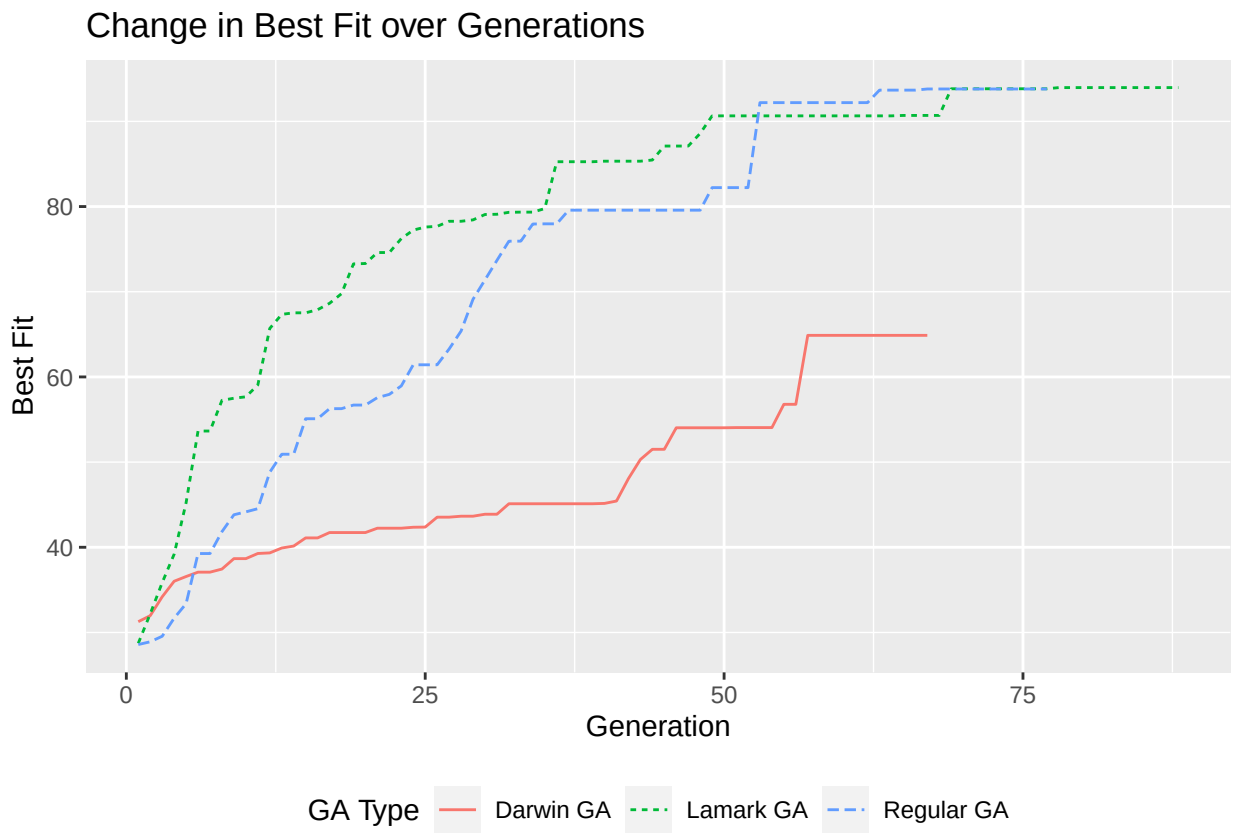
In the case of the Lamarckian Genetic Algorithm, it is evident that the best run and the best runs consistently occur with the same set of parameters., here is the graph for the run:

```
# Read the CSV file
data <- read.csv("C:\\Users\\User\\.vscode\\comp_bio\\comp_bio_targil2\\lamark amin graph.csv")

# Create a line plot
ggplot(data, aes(x = generation, y = best_fit)) +
  geom_line() +
  labs(x = "Generation", y = "Best Fit") +
  ggtitle("Change in Best Fit over Generations- lamark GA")
```



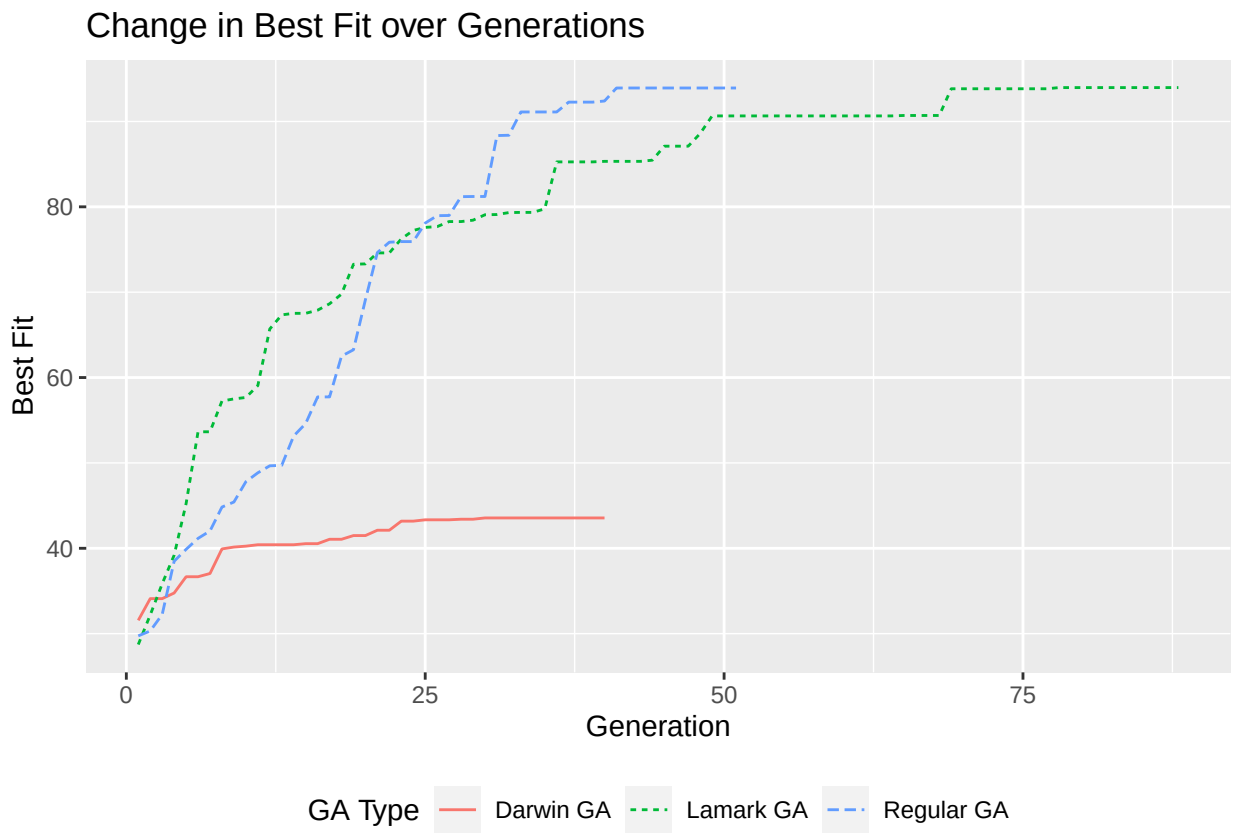

## THE DIFFRENCE IN THE ALGORITHEMS BEST RUNS:

```r
# Read the CSV files
data_regular <- read.csv("C:\\Users\\User\\.vscode\\comp_bio\\comp_bio_targil2\\regular best graph.csv")
data_lamark <- read.csv("C:\\Users\\User\\.vscode\\comp_bio\\comp_bio_targil2\\lamark amin graph.csv")
data_darwin <- read.csv("C:\\Users\\User\\.vscode\\comp_bio\\comp_bio_targil2\\darwin best graph.csv")

# Add a column to indicate the GA type
data_regular$GA_type <- "Regular GA"
data_lamark$GA_type <- "Lamark GA"
data_darwin$GA_type <- "Darwin GA"

# Combine the data frames
combined_data <- bind_rows(data_regular, data_lamark, data_darwin)

# Plot the combined data
ggplot(combined_data, aes(x = generation, y = best_fit, color = GA_type, linetype = GA_type)) +
  geom_line() +
  labs(x = "Generation", y = "Best Fit", color = "GA Type", linetype = "GA Type") +
  ggtitle("Change in Best Fit over Generations") +
  theme(legend.position = "bottom")
```




## THE DIFFRENCE IN THE ALGORITHEMS MOST RELIABLE RUNS:

```
# Read the CSV files
data_regular <- read.csv("C:\\Users\\User\\.vscode\\comp_bio\\comp_bio_targil2\\regular amin graph.csv")
data_lamark <- read.csv("C:\\Users\\User\\.vscode\\comp_bio\\comp_bio_targil2\\lamark amin graph.csv")
data_darwin <- read.csv("C:\\Users\\User\\.vscode\\comp_bio\\comp_bio_targil2\\darwin amin graph.csv")

# Add a column to indicate the GA type
data_regular$GA_type <- "Regular GA"
data_lamark$GA_type <- "Lamark GA"
data_darwin$GA_type <- "Darwin GA"

# Combine the data frames
combined_data <- bind_rows(data_regular, data_lamark, data_darwin)

# Plot the combined data
ggplot(combined_data, aes(x = generation, y = best_fit, color = GA_type, linetype = GA_type)) +
  geom_line() +
  labs(x = "Generation", y = "Best Fit", color = "GA Type", linetype = "GA Type") +
  ggtitle("Change in Best Fit over Generations") +
  theme(legend.position = "bottom")
```




We can see in the graphs that the regular algorithm achieves convergence to the best solution at a faster rate compared to other algorithms.