INSTITUTO TECNOLÓGICO DE BUENOS AIRES
SISTEMAS DE INTELIGENCIA ARTIFICIAL
INGENIERÍA EN INFORMÁTICA

Algoritmos genéticos

Tercer trabajo práctico especial

Titular: Parpaglione, Cristina
Semestre: 2019 1C
Grupo: 5
Repositorio:
https://bitbucket.org/itba/sia-2019-1c-05/src/master/genetic-algorithms/
Entrega: 5 de junio de 2019
Autores: Banfi, Micaela (57293)
Guzzetti, Clara (57100)
Ritorto, Bianca (57082)
Vidaurreta, Ignacio (57250)

1. Introducción

En este trabajo práctico se intentará hacer uso de algoritmos genéticos para encontrar la mejor configuración posible de "items" en un personaje de juego de rol, y así poder maximizar las características del personaje.

2. Motor

El motor consta de una clase llamada GeneticEngine que hace uso de las implementaciones de distintos tipos de:

- Combinadores (o *Crossers*)
- Selectores (o *Selectors*)
- Reemplazantes (o *Replacers*)
- Condicionales (o *Conditioners*)
- Mutadores (o *Mutators*)

2.1. GeneticEngine

El motor está separado de la implementación del juego en sí. Consta de distintas etapas de un algoritmo genético. Se pueden apreciar tres grandes etapas:

1. Selección de padres: se genera una lista en base al primer selector que se elige.
2. Procreación: se generan parejas con la lista de padres. Se cruzan los padres, obteniendo dos hijos de cada pareja. Se mutan los hijos si fuera necesario.
3. Reemplazo: Se genera una mezcla final de padres e hijos, las cantidades de cada generación dependiendo del algoritmo correspondiente y el selector utilizado.

2.2. Reemplazantes

Los *Replacers* son los encargados de definir el método con el que se obtendrán los padres y cómo se generará la descendencia. Se implementaron tres replacers:

- KeepSomeAncestorsReplacer: Mantiene en la nueva generación algunos padres y algunos hijos.
- MixAllReplacer: Mezcla las dos generaciones y elige todos los individuos de nuevo de esta nueva mezcla.
- NewGenerationReplacer: Utiliza como nueva generación a los hijos únicamente.

2.3. Condicionales

Los *Conditioners* representan las distintas formas de frenar la evolución de las *Species*. Se implementaron varias formas de decidir si se debe seguir o no procreando:

- ContentConditioner: Si la *Fitness* máxima no cambia luego de cierta cantidad de generaciones, se debe frenar la ejecución.
- GenerationConditioner: Se designa una máxima cantidad de generaciones a crear. Cuando se supera este número, se frena la ejecución.
- OptimumConditioner: Se determina un valor de *Fitness* y se compara tras cada generación, si la máxima de esa generación es mayor o igual a la determinada. UNa vez que se alcanza el valor deseado, se frena la ejecución.

- StructureConditioner: Se frena la ejecución si se determina que, dado un delta, la población de la nueva generación no cambió lo suficiente comparado con la de la generación anterior.

2.4 Selectores

Se implementaron los selectores pedidos en la consigna más Boltzmann con Tournament Selection. Para la temperatura de Boltzmann se utilizó: $\frac{1}{0.001*(generacion+1)+1}$

3. Juego

3.1. Especies

Se mantuvo una interfaz genérica de *Species*. Sobre ella se implementó una clase llamada GameCharacter que encapsula la información necesaria de cada individuo para poder calcular su valor de Fitness. Contiene la altura y los Items asociados al personaje.

El *cromosoma* de un individuo se define como el array de items + altura del personaje.

3.2 Combinadores

Se implementaron varios *Combinators* que definen la forma de intercambiar cromosomas entre padres para generar hijos. También evalúan si la pareja actual debe ser combinada o si debe sobrevivir sin reproducirse. Los combinadores que se implementaron son:

- AnnularCross**:** Se elige un locus y luego un segmento hacia la derecha de dicho locus de longitud l que se encuentra en el rango [1, L] donde L es la longitud del cromosoma.
- SinglePointCross: Selecciona un locus al azar y se a partir de ese locus se intercambian los alelos. Se tuvo que tener cuidado porque el acceso a ese locus comienza en 1 en vez de en 0 como en los arreglos normalmente.
- DoublePointCross: Se seleccionan dos locus al azar (r1 y r2, verificando que r2>r1) y se intercambian los alelos correspondientes a ese intervalo. Para garantizar que r2>r1 lo que se hizo fue generar los 2 números y si r1 > r2 entonces swappear los valores.
- UniformCross: Se produce el cruce de un alelo en cada locus con probabilidad P. En nuestro caso, definimos dicha probabilidad P como 0.5

3.3 Mutadores

Estos generan con probabilidad uniforme o no uniforme mutaciones sobre los individuos. Contiene una función que, dada una probabilidad, decide si el individuo actual debe o no mutar. Se implementaron varios mutadores:

- MultiGenMutator: Muta varios genes de un individuo.
- OneGenMutator: Muta un solo gen de un individuo.

Ambos tienen la opción de ser uniformes o no uniformes, dependiendo de la función de probabilidad que se les asigne.

La función para la probabilidad no uniforme es

$$maxProb - [generation/10] * step$$

Esta función parte de una probabilidad máxima y cada 10 generaciones va disminuyendo en un factor *step* definido previamente de manera experimental. Se define también una cota mínima llamada minProb que si el resultado de la probabilidad es menor a minProb la función devuelve minProb.

Nosotros defininimos: **maxProb = 0.8**, **minProb=0.2** y **step=0.006**.

3.4 Diversidad

Para realizar un análisis sobre la diversidad de determinadas generaciones, se decidió dividir a la generación en 4 grupos: Buen Ataque, Buena Defensa, Buen Ataque y Defensa, Resto.

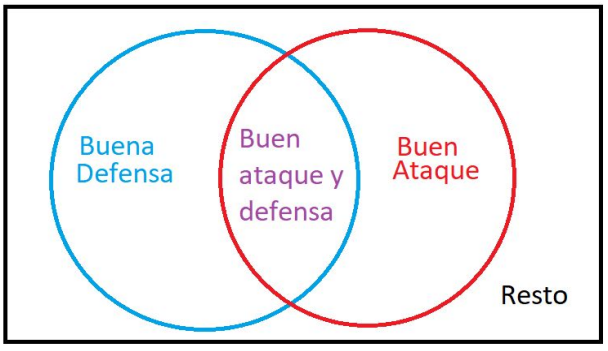


Diagrama de Venn representando los diferentes grupo de la diversidad

De dicha generación se obtuvieron los máximos y los mínimos de los ataques y las defensas y se definió como "Buen Ataque" a los personajes que tenían su ataque superior al 80% del total. De la misma manera "Buena Defensa" y "Buen Ataque y Defensa". Los personajes que tienen su ataque y defensa inferiores, se ubican en el grupo "Resto".

4. Conclusiones

4.1. Análisis de resultados

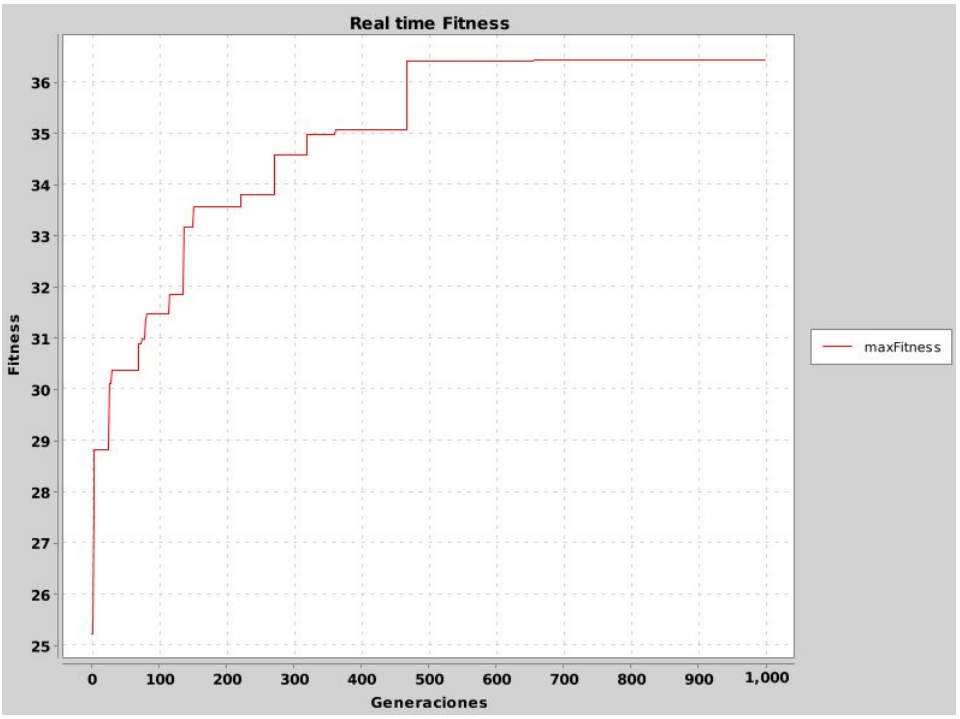
- Se llegó a la mejor solución (la que generó un mejor *fitness máximo*) mediante a la utilización de 4 selectores distintos en vez de mantener los mismos. La mejor configuración fue:
    - Selector1: Elite Selection, Selector2: ProbabilisticTournamentSelection, Selector3: RankingSelection, Selector4: RouletteSelectio
    - Ratio A: 0.5, Ratio B: 0.5
    - Mutator: UniformMultiGenteMutator
    - Replacer: KeepSomeAncestorsReplacer
    - Crossover: AnnularCross
    - Conditioner: ContentConditioner

- En el caso de la selección no uniforme, la selección de la función probabilística es bastante compleja. Porque como se va a ejecutar muchas veces la función puede generar un *overhead* bastante grande, enlenteciendo la ejecución. Al principio se utilizó como ecuación $1/(generation + 1)$. La ventaja que tenía esta ecuación era que producía un overhead bastante bajo, pero baja demasiado rápido y caía en una convergencia temprana. Luego utilizamos la ecuación $maxProb - [generation/10] * step$ la cual funciona correctamente, pero enlentece bastante la ejecución. Decidimos mantener esta ecuación, dado que preferimos que tarde un poco más pero que evolucione correctamente.
- En lo que respecta a los métodos de reemplazo, se observa claramente que el New generation Replacer no alcanza buenos resultados, lo cual es lógico, ya que descarta la generación anterior por completo. Entre los otros dos métodos de reemplazo, no se encuentran grandes diferencias.

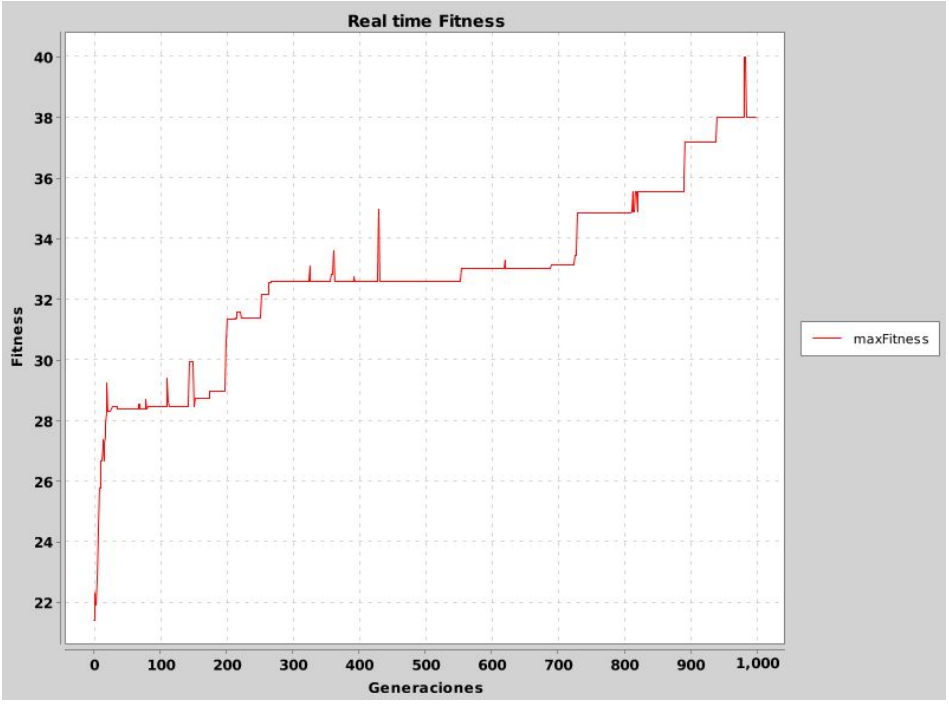
5. Apéndice

Selectors:

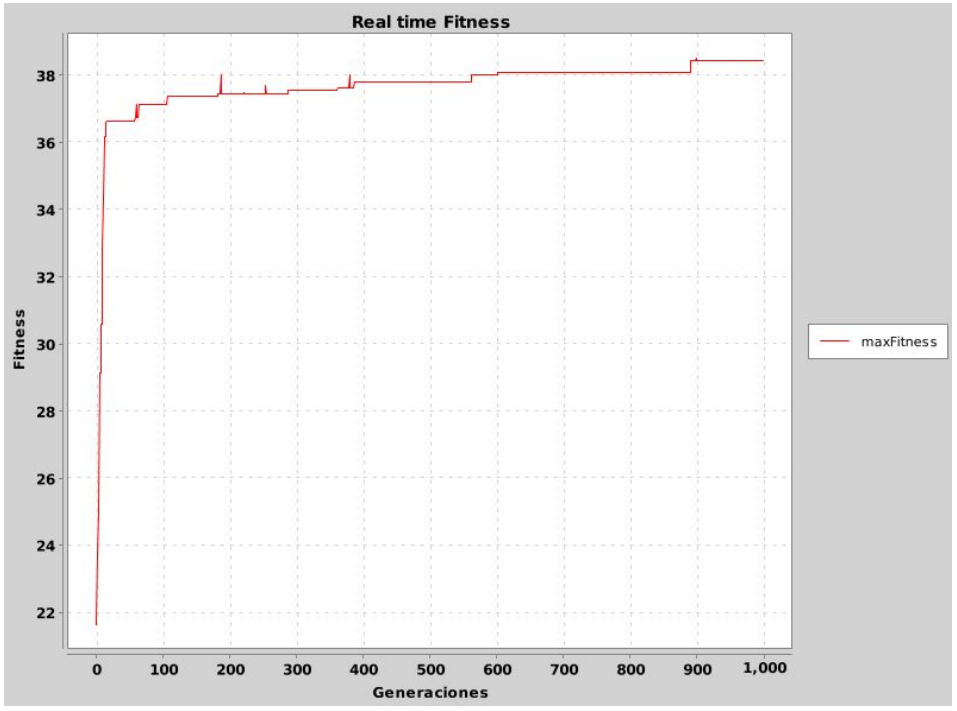
Gráficos realizados con: ratioA = 0.6, ratioB = 0.4. **Mutator:** UniformMultiGeneMutator, **Repalcer**: KeepSomeAncestorsReplacer, **Cross:** SinglePointCross.



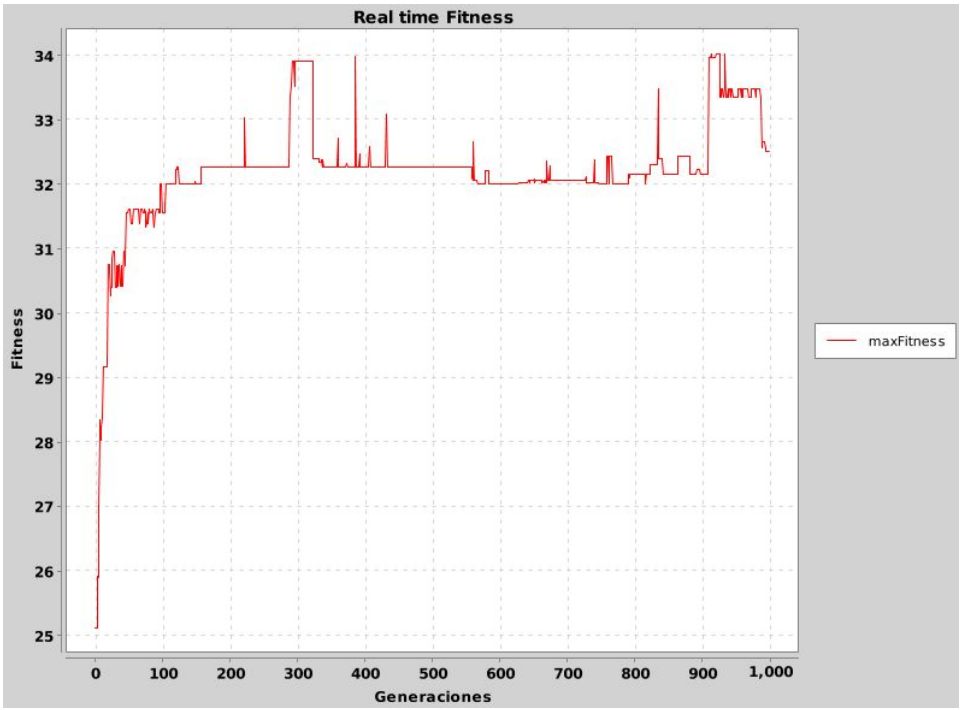
Elite



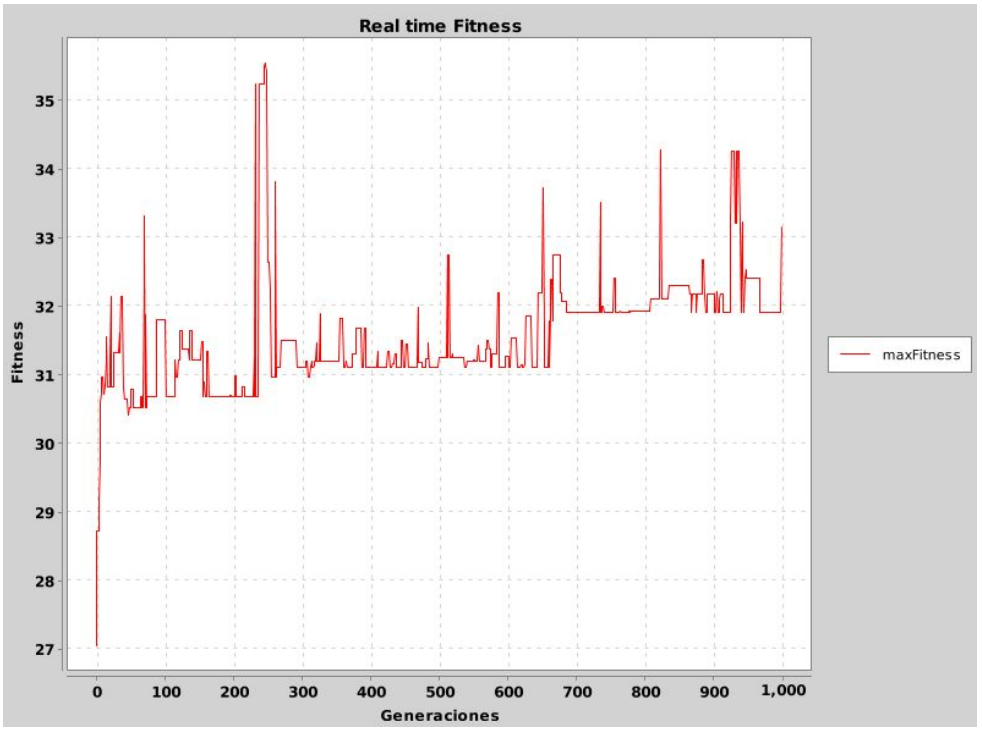
Probabilistic Tournament



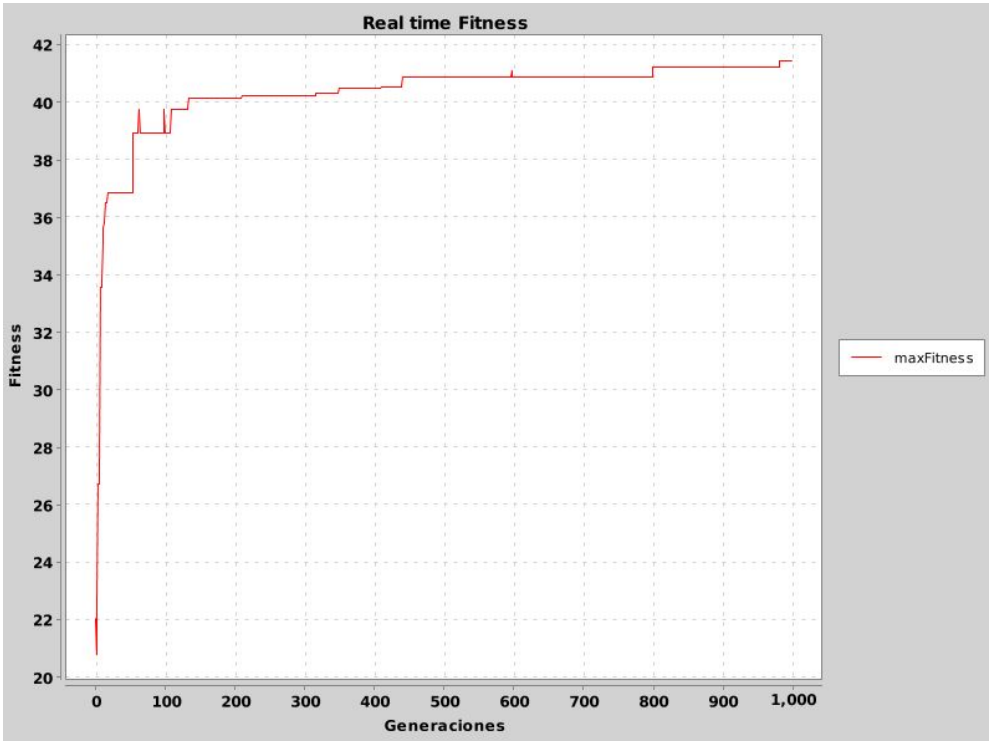
Ranking



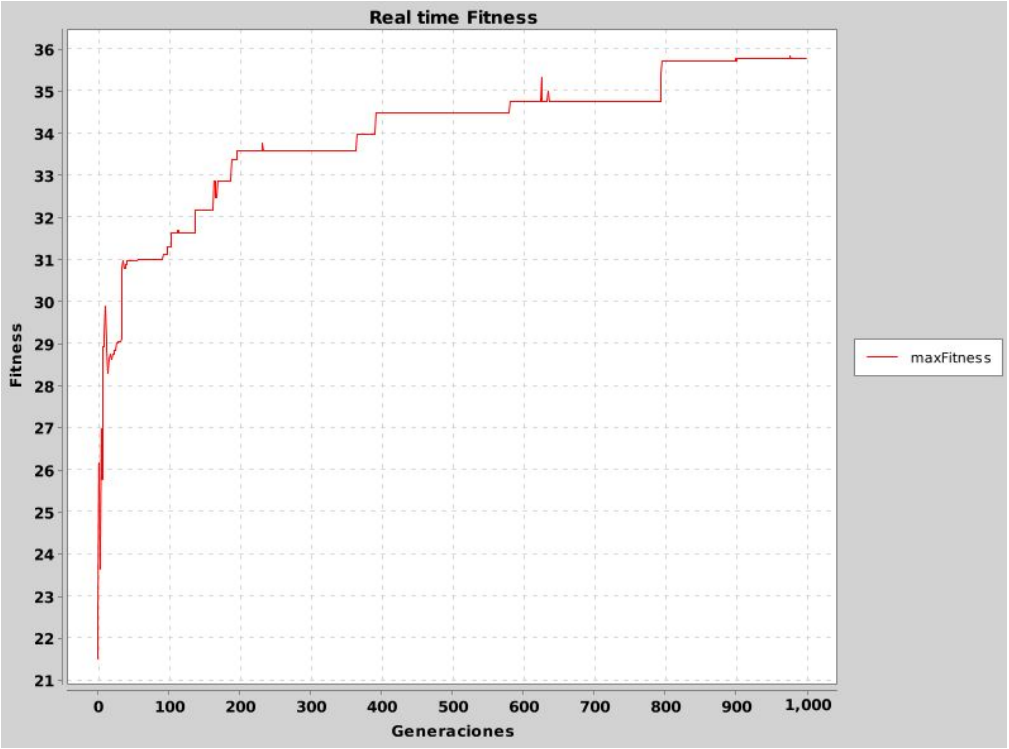
Roulette



Universal

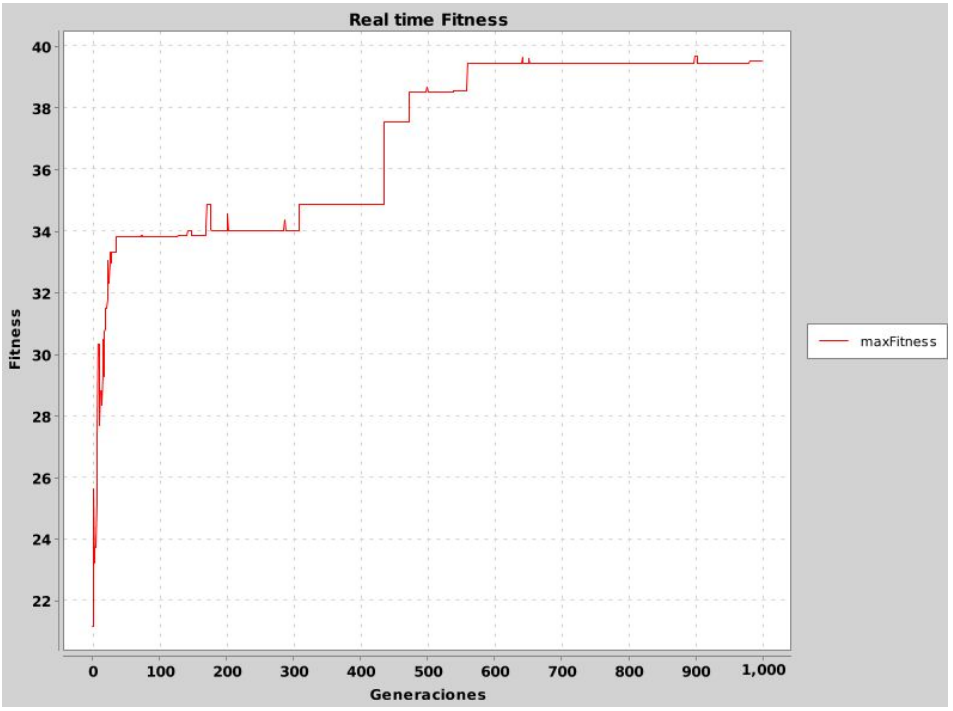


Deterministic Tournament

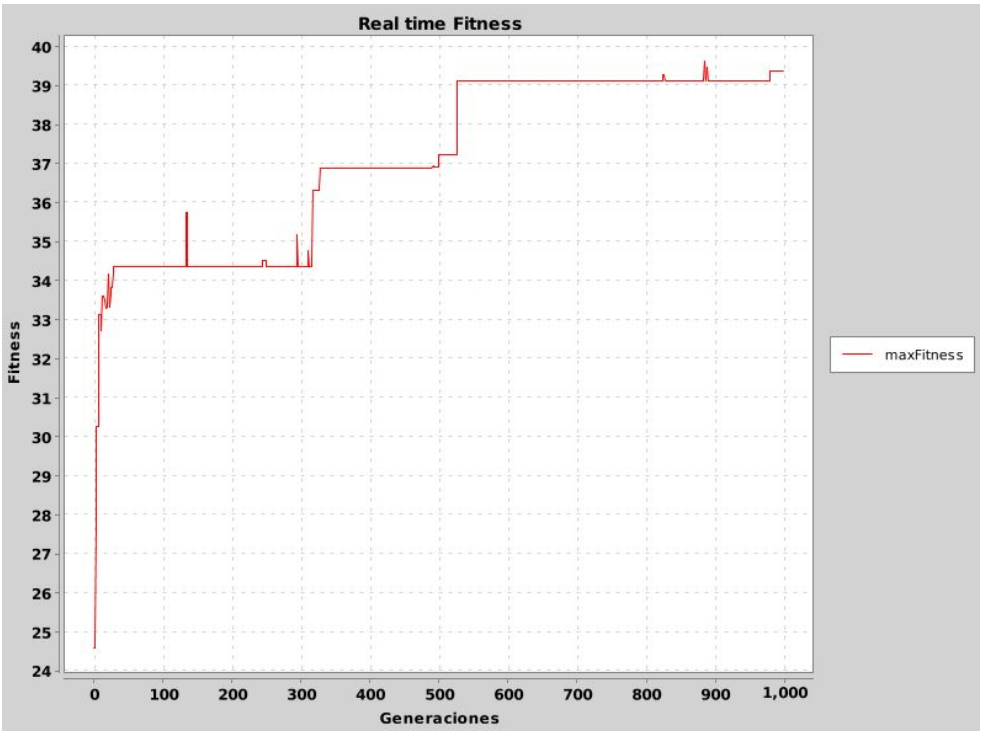


BoltzmannRoulette

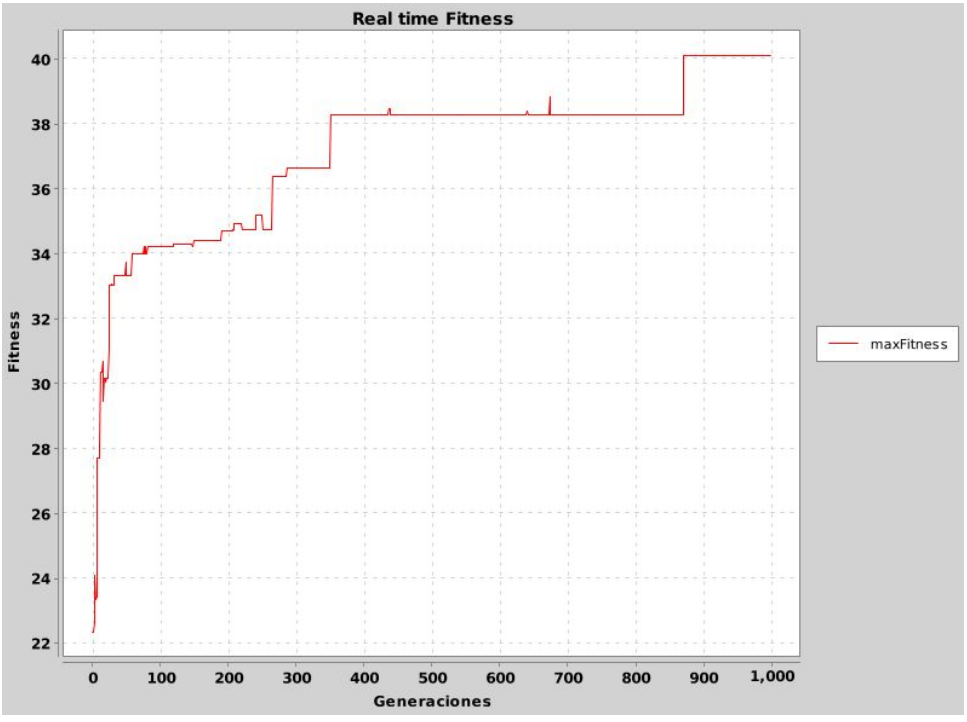
Combinación de selectores (Ratio 0.6 y 0.4) :



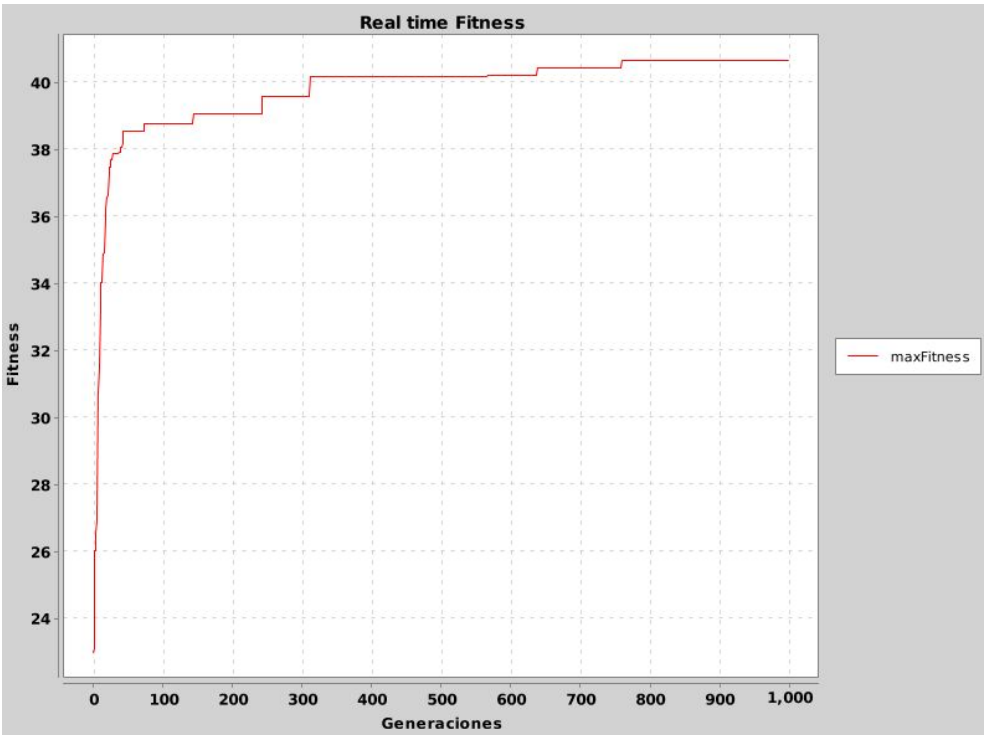
Elite, ProbabilisticTournamentSelection, RankingSelection, RouletteSelection



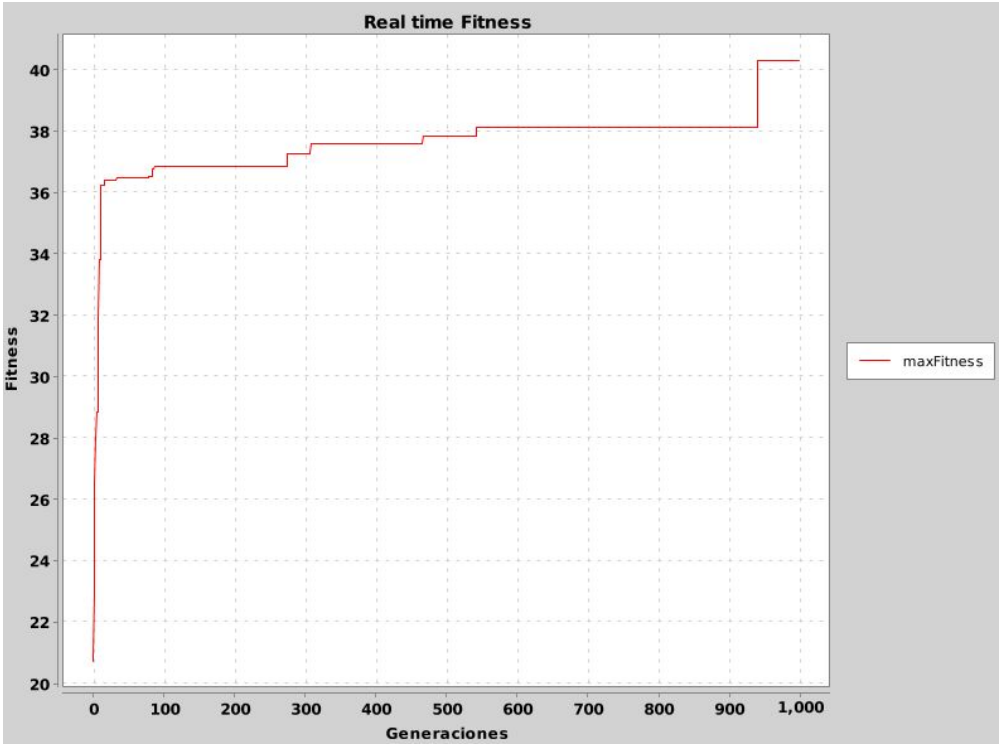
RankingSelection, RouletteSelection, UniversalSelection, Deterministic



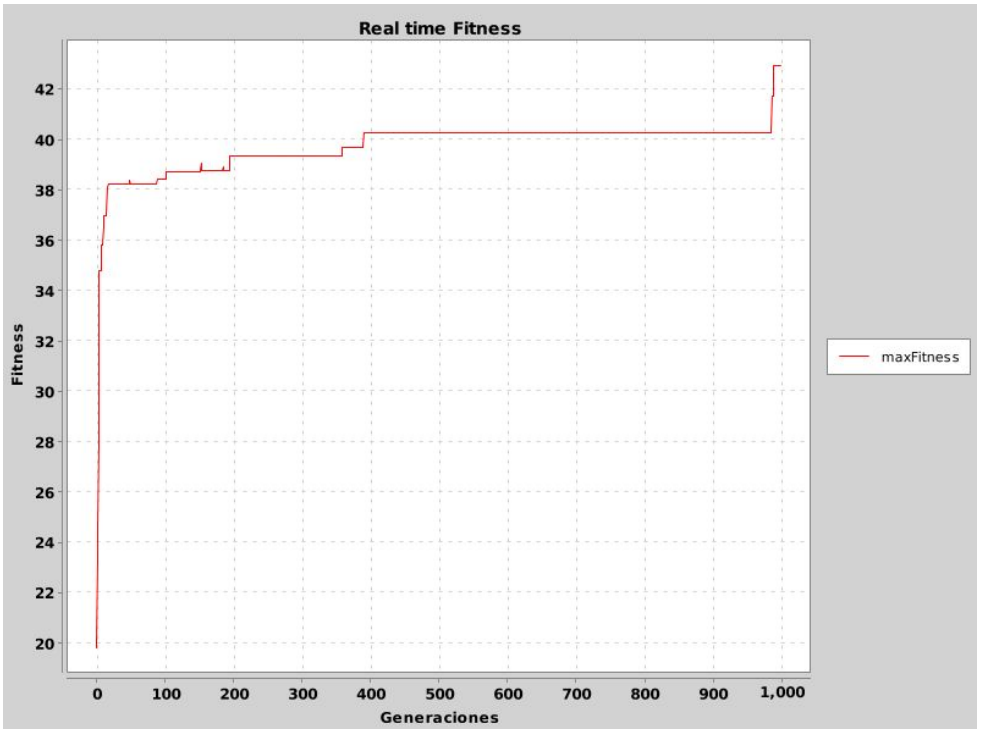
Roulette, Universal, Deterministic, Boltzmann



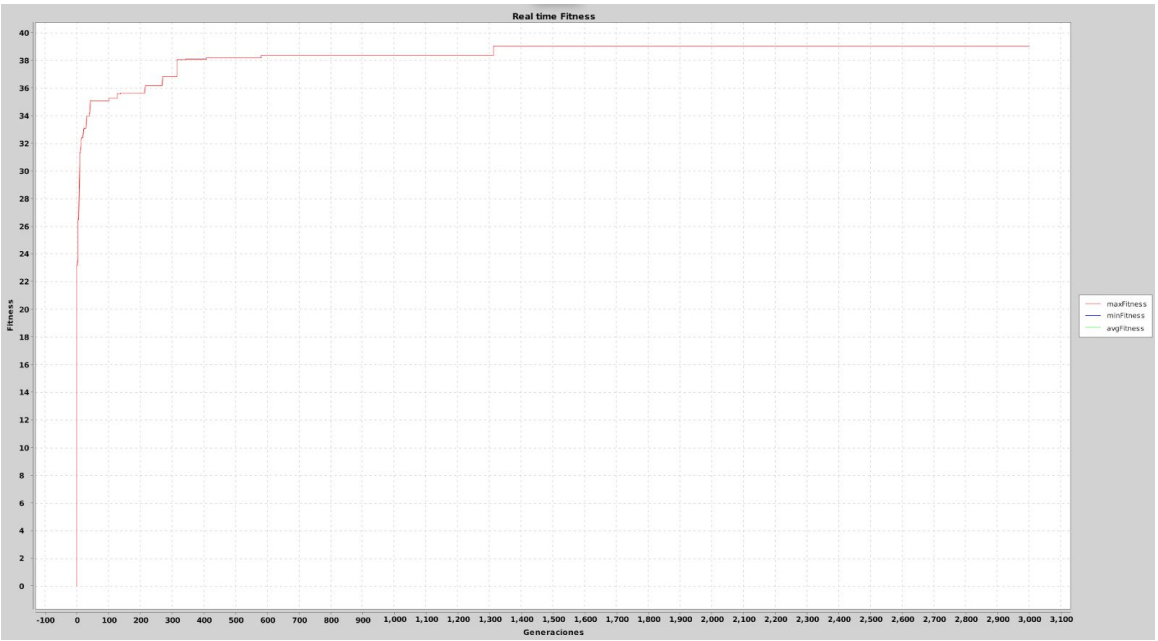
Universal, Deterministic, BoltzmanRoulette, Elite



Mejor combinación: Deterministic, BoltzmannRoulette, Elite, Proba

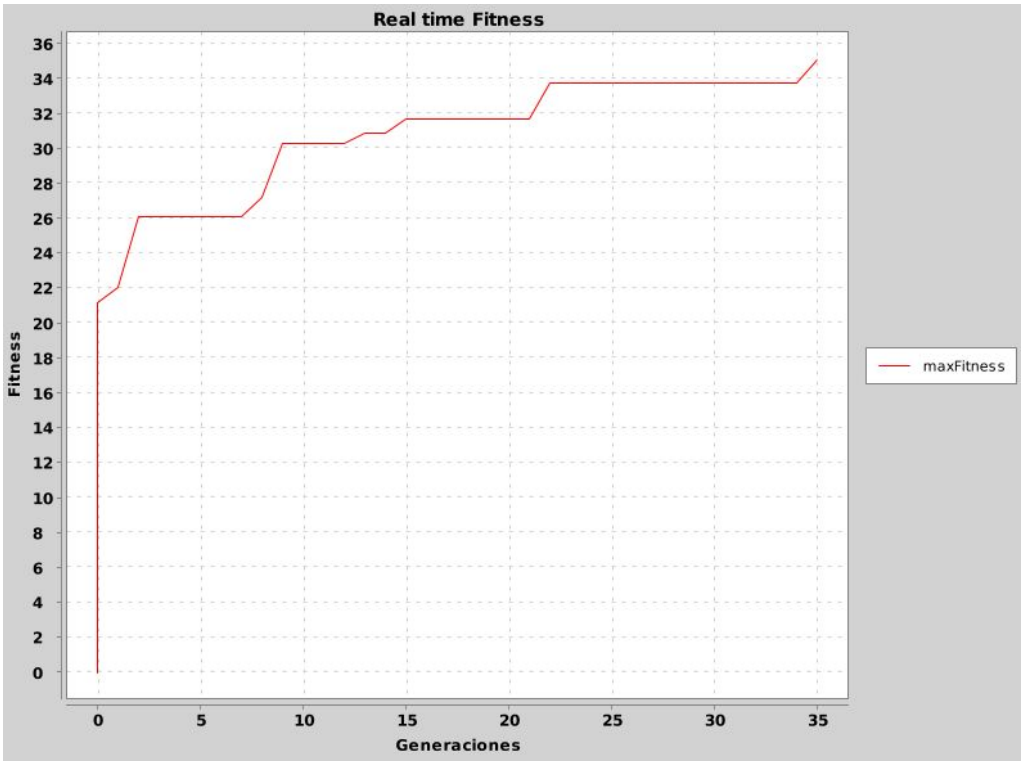


BoltzmannRoulette, Elite, Proba, Ranking

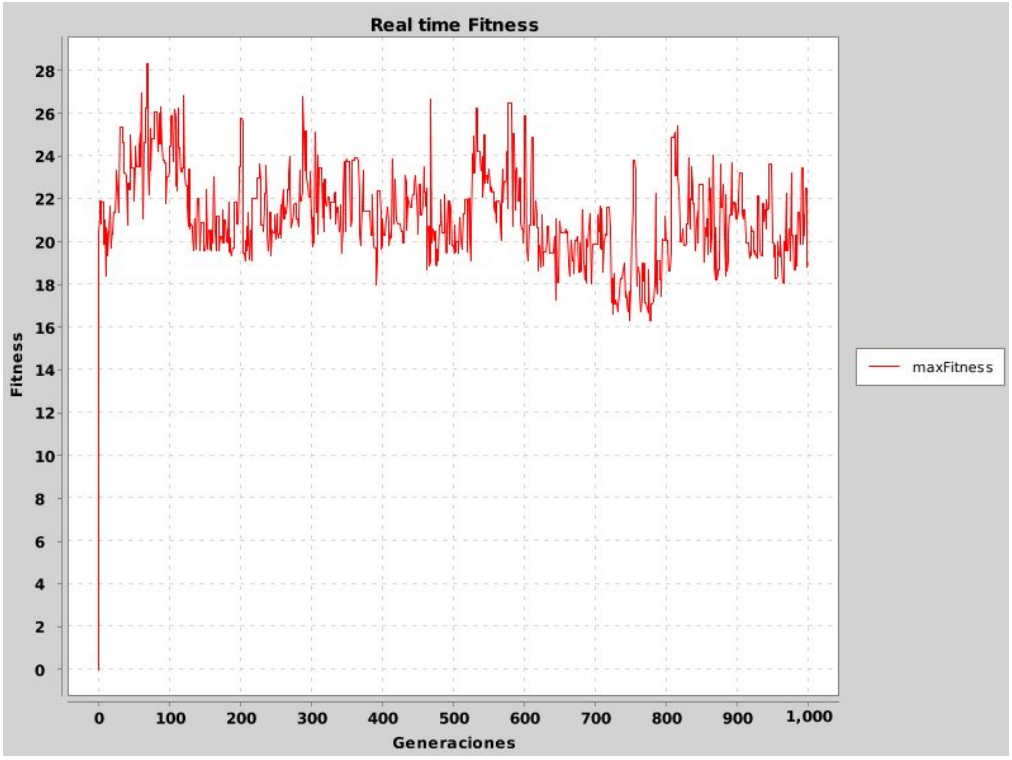


DetterministicTournament, BoltzmannRoulette, Elite, Probabilistic Selection.

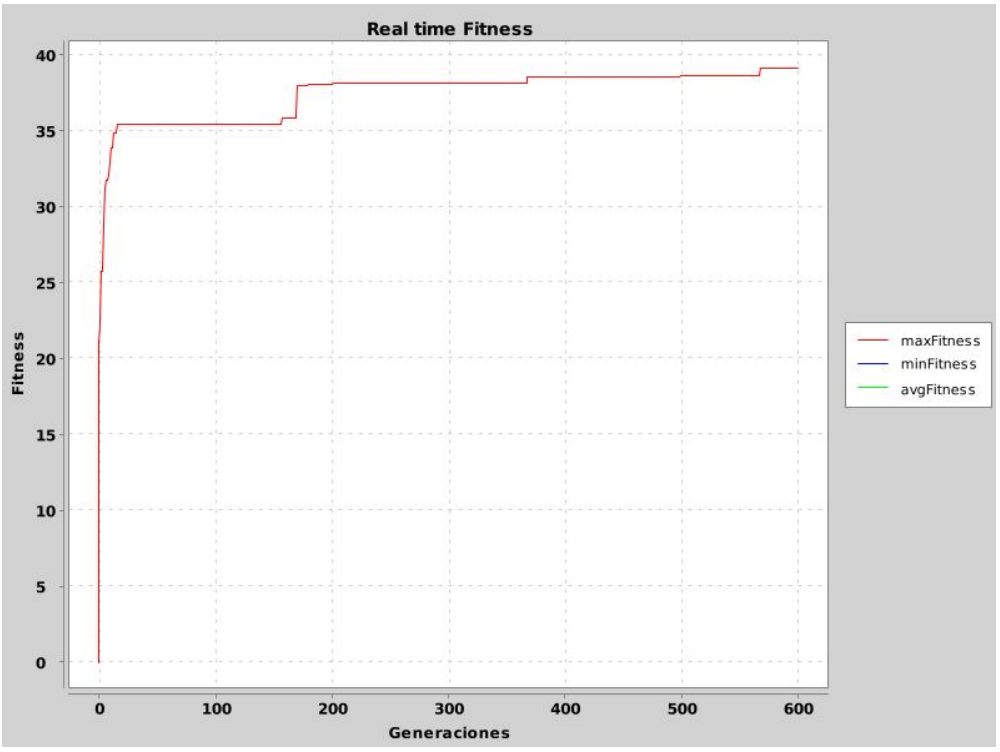
Replacers:



KeepSomeAncestors

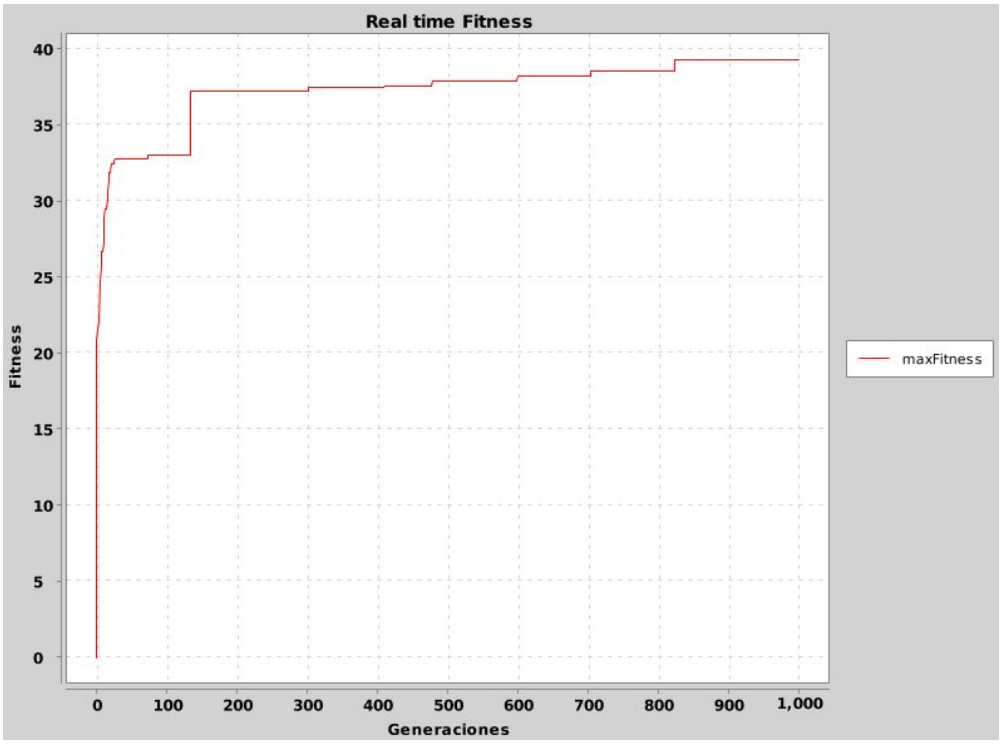


NewGeneration

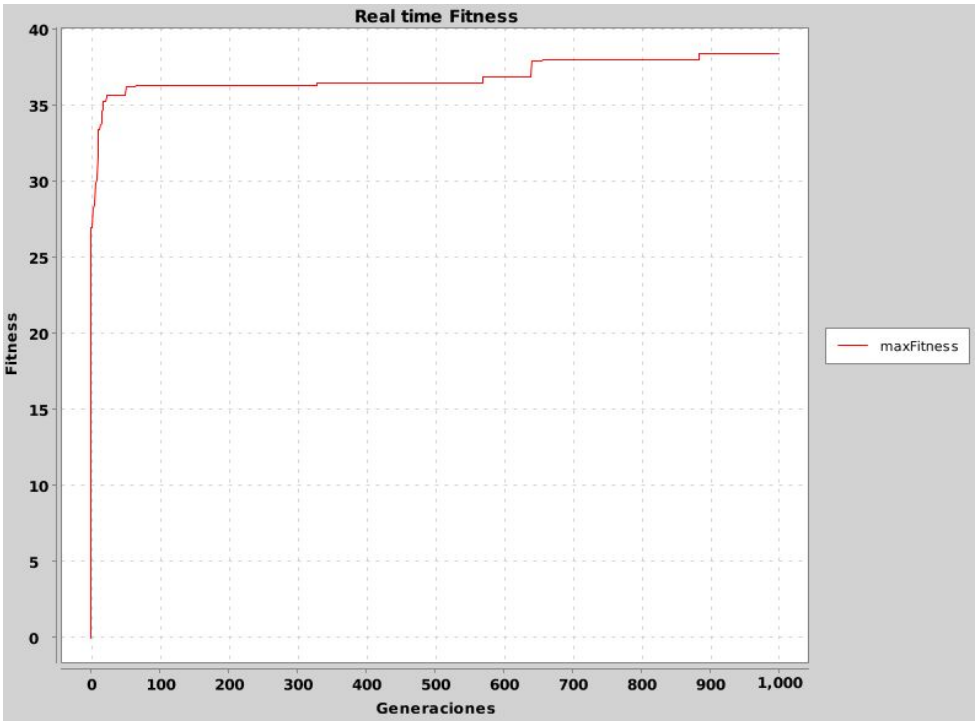


MixAll

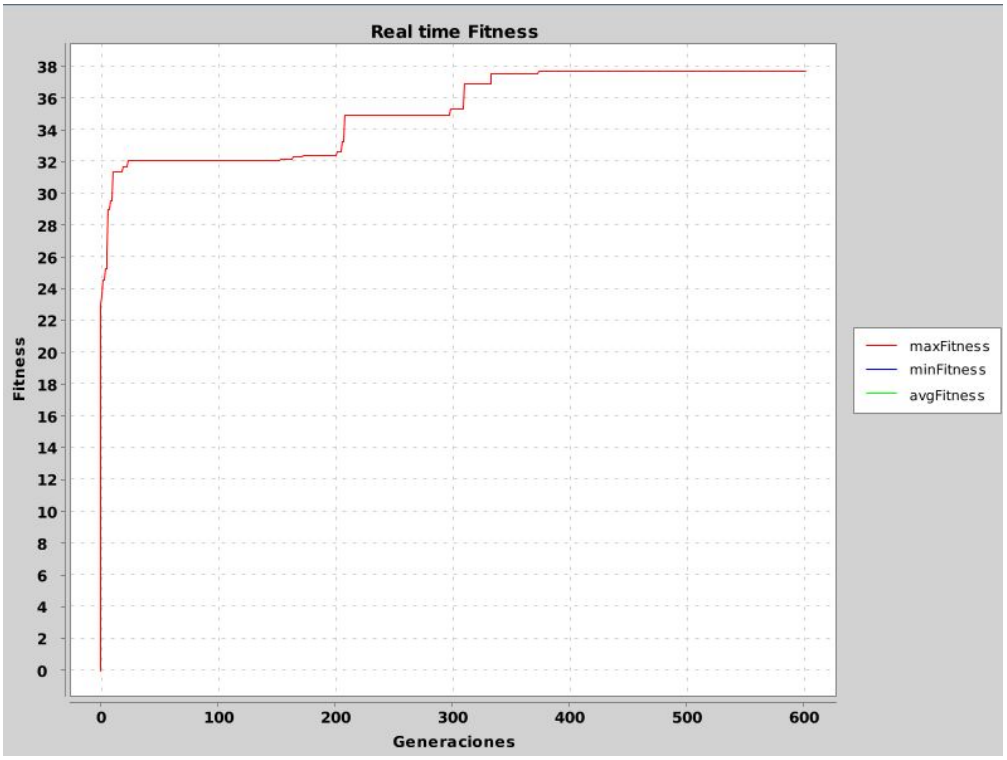
Crossers:



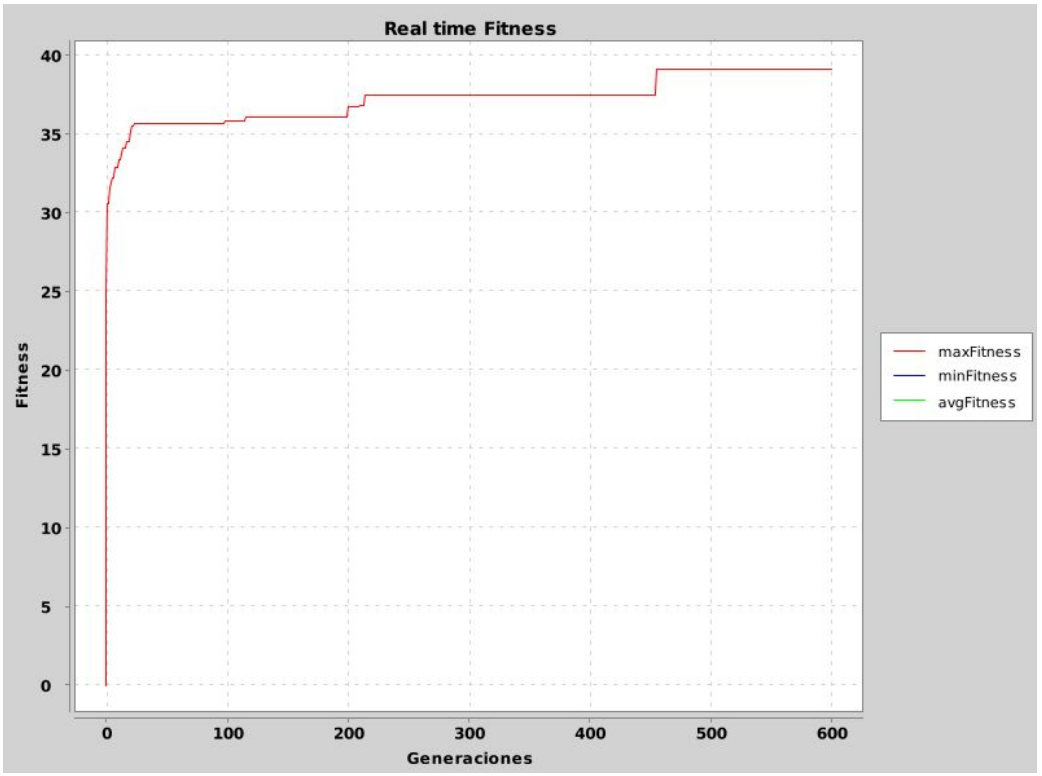
Annular



Uniform

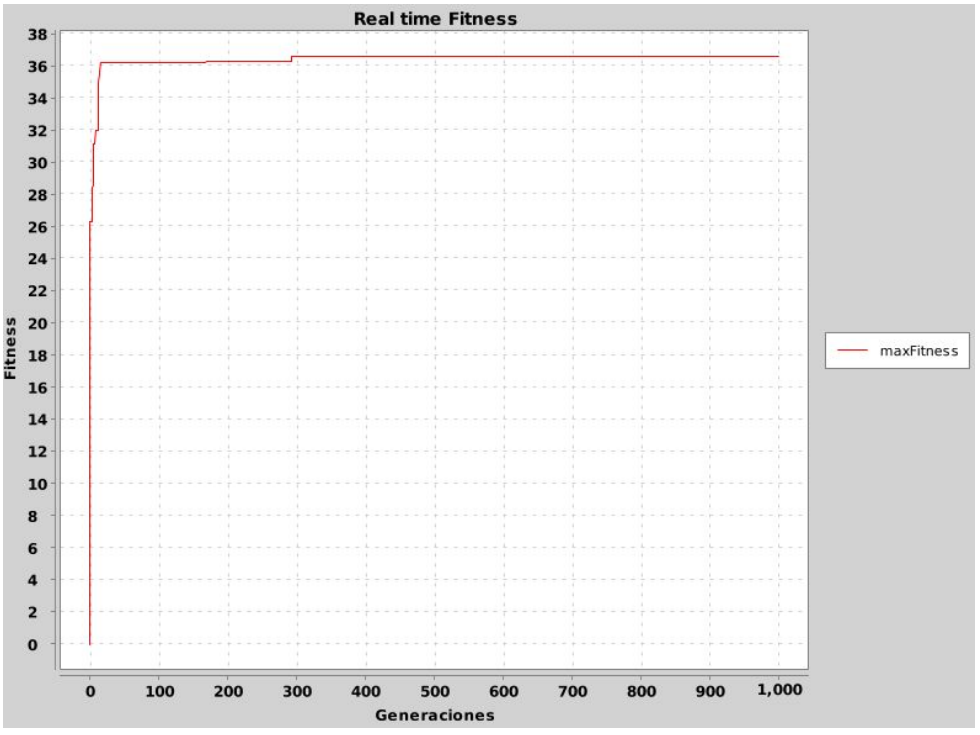


SinglePoint

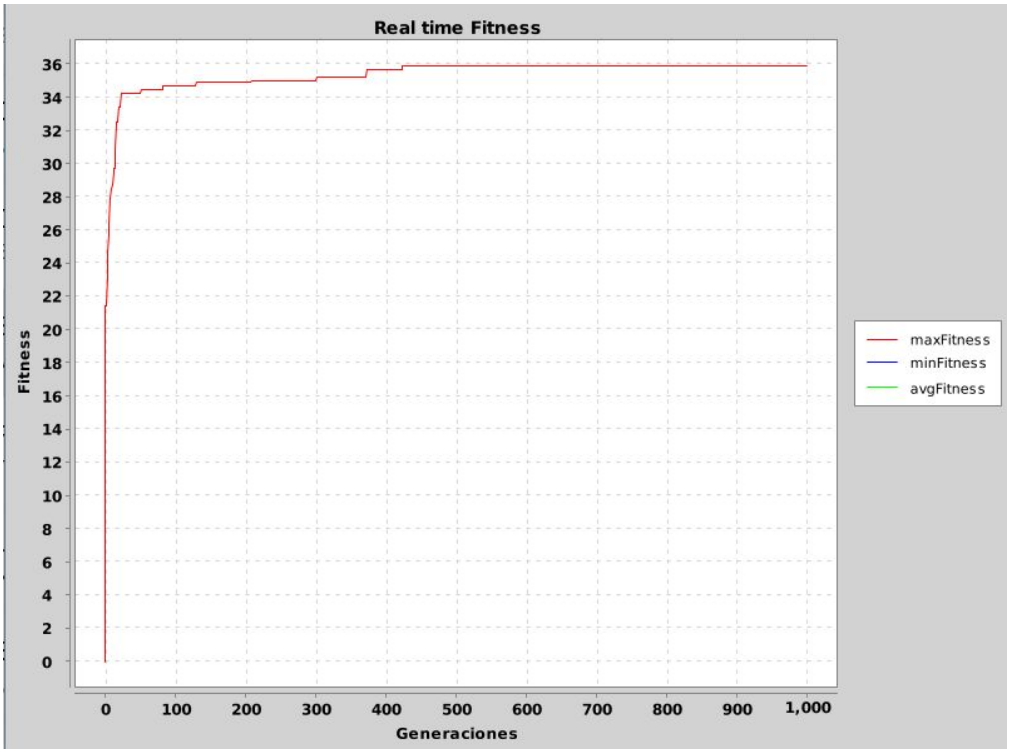


DoublePoint

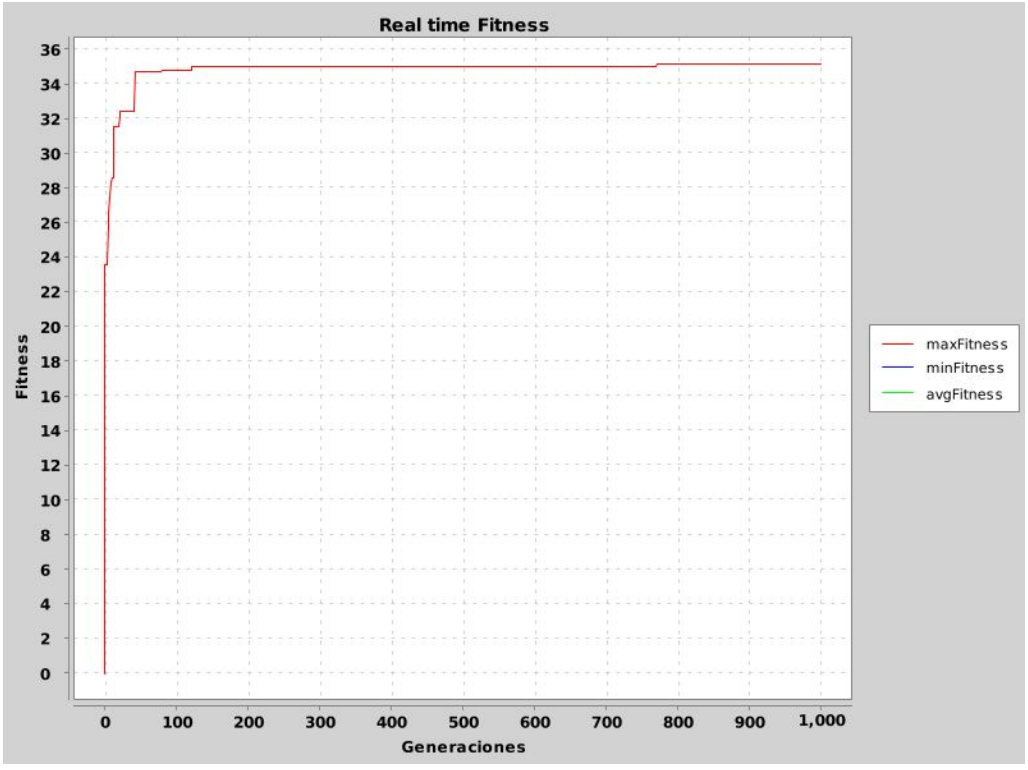
Mutators:



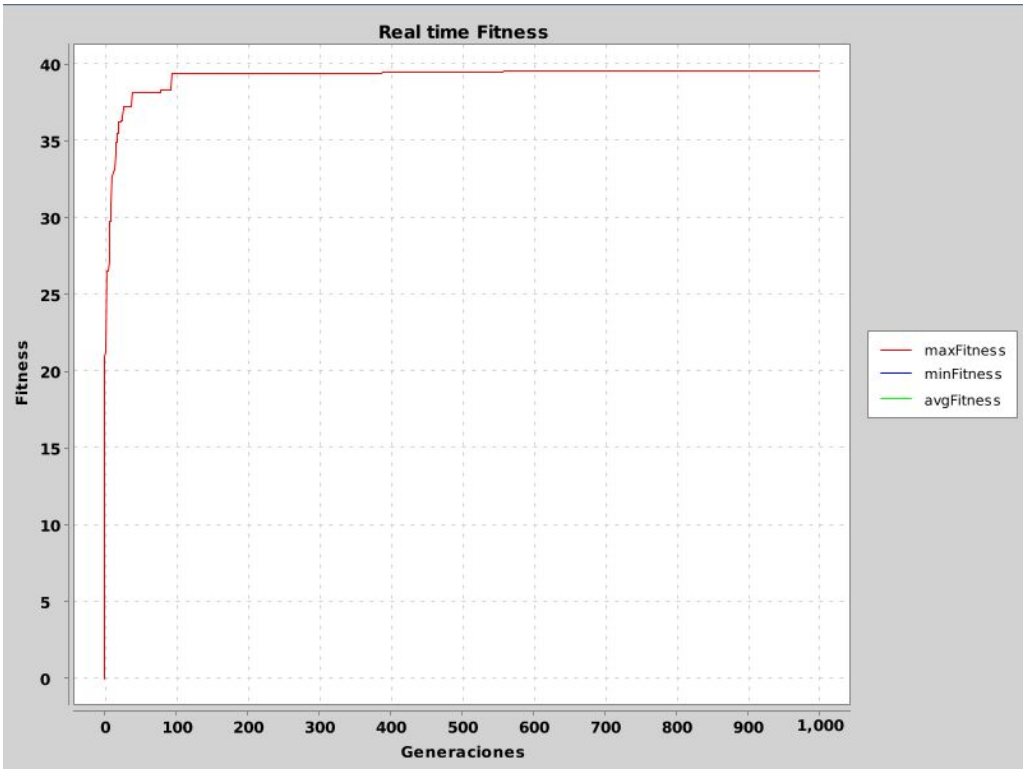
NonUniformMultiGene



UniformOneGene

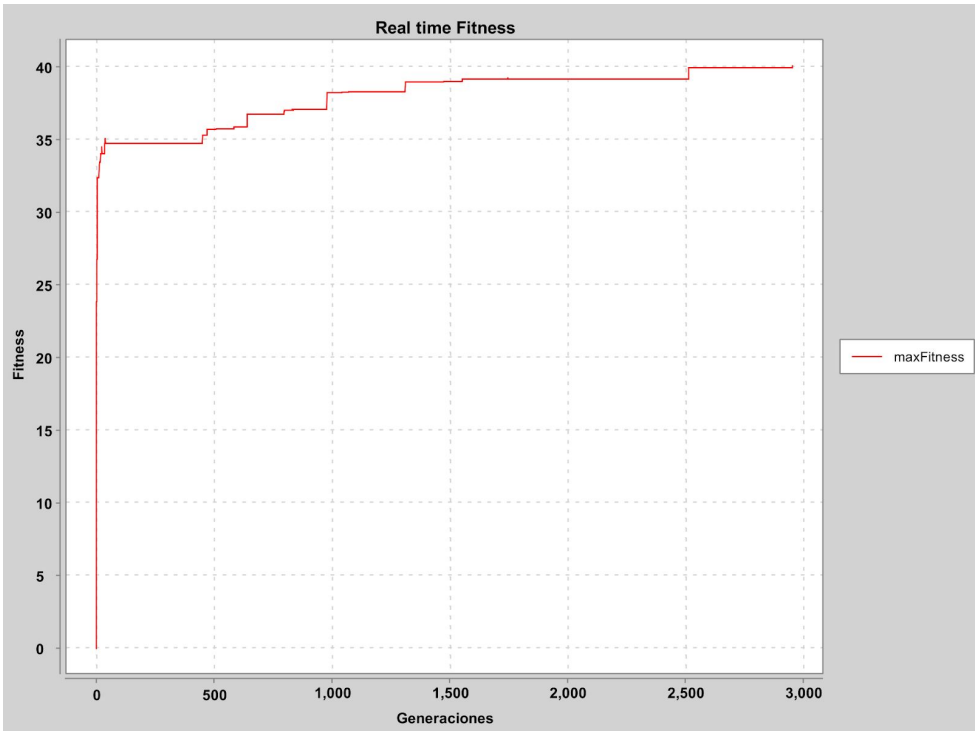


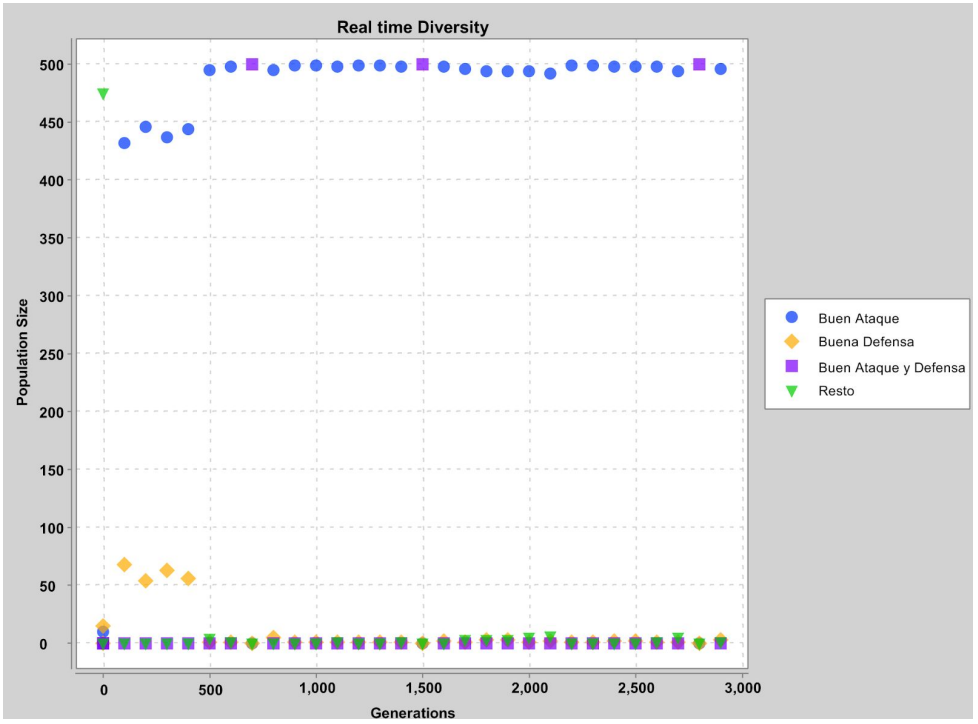
NonUniformOneGene



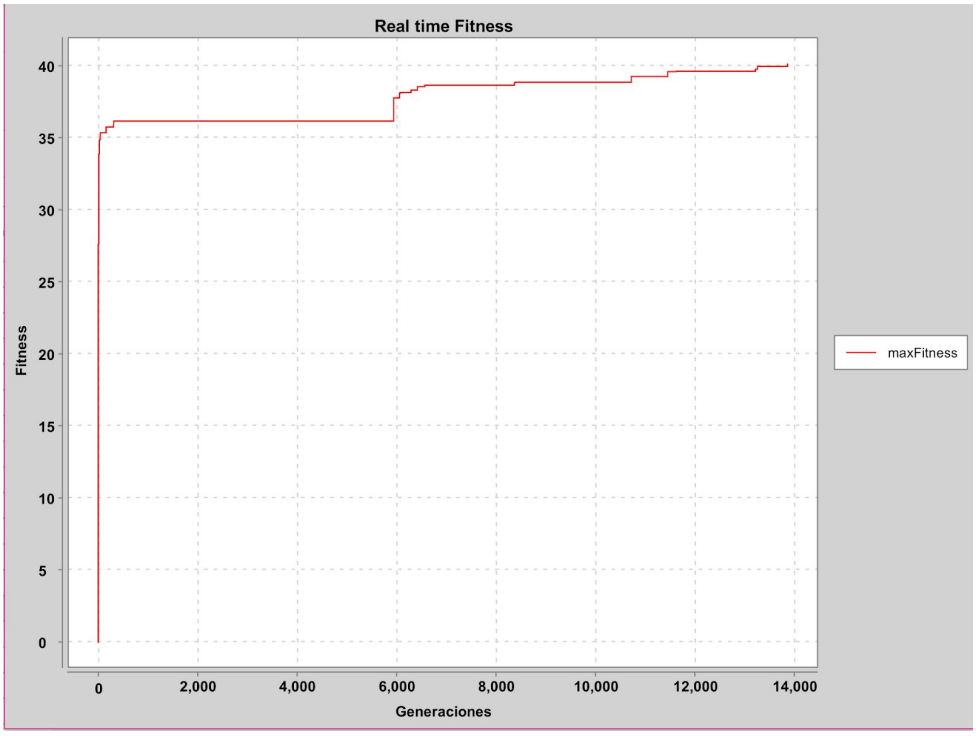
UniformMultiGene

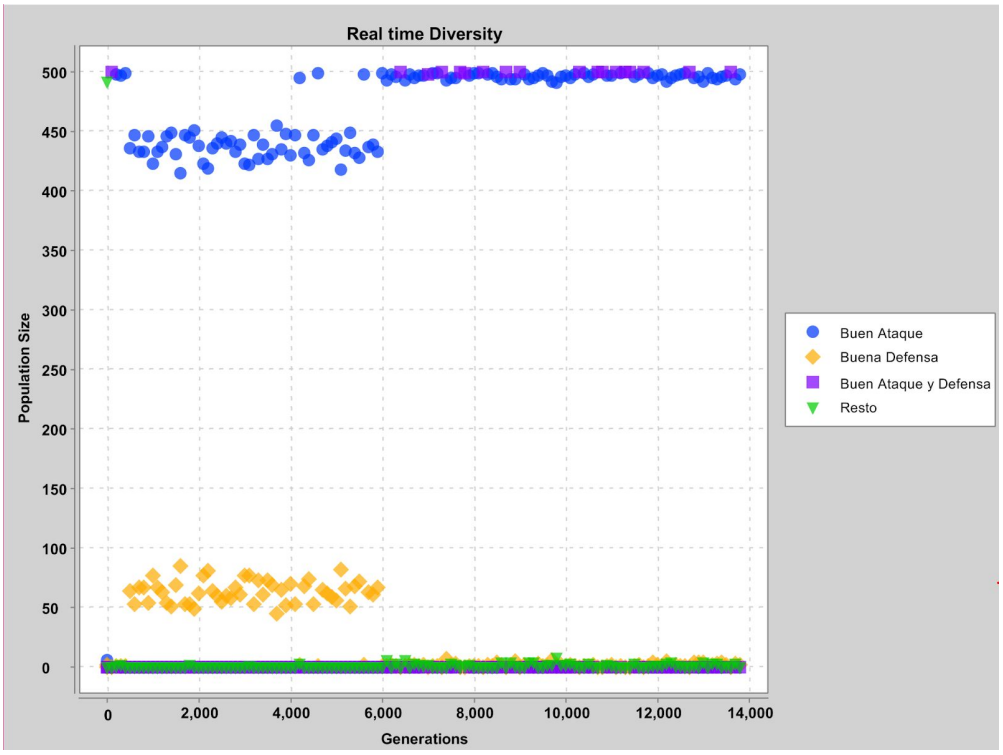
Diversidad:



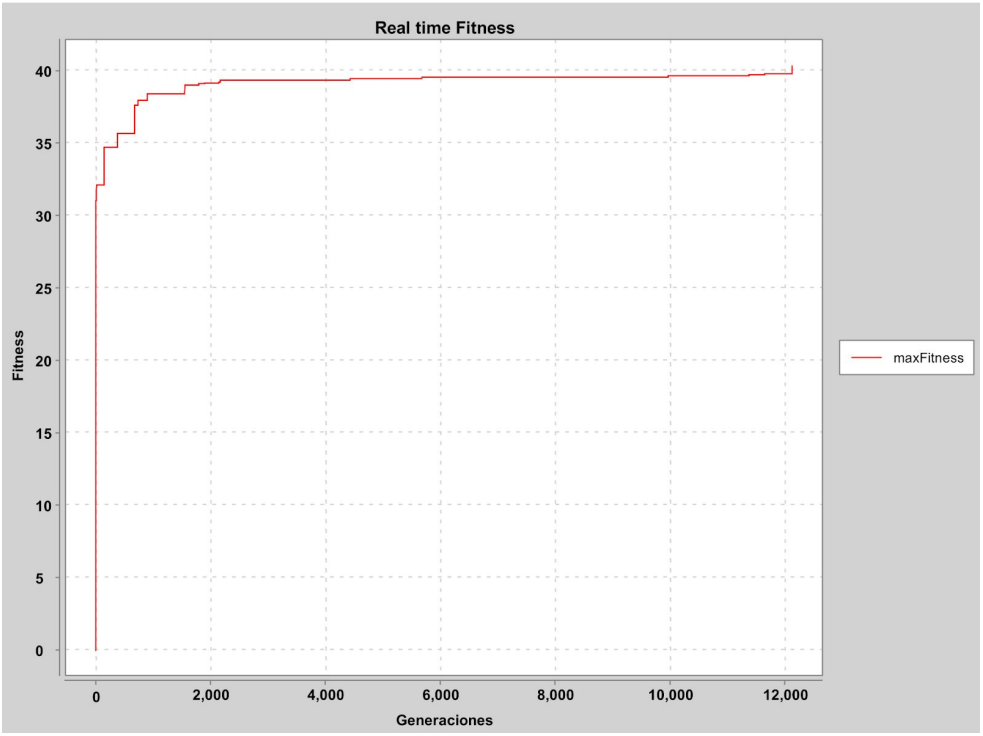


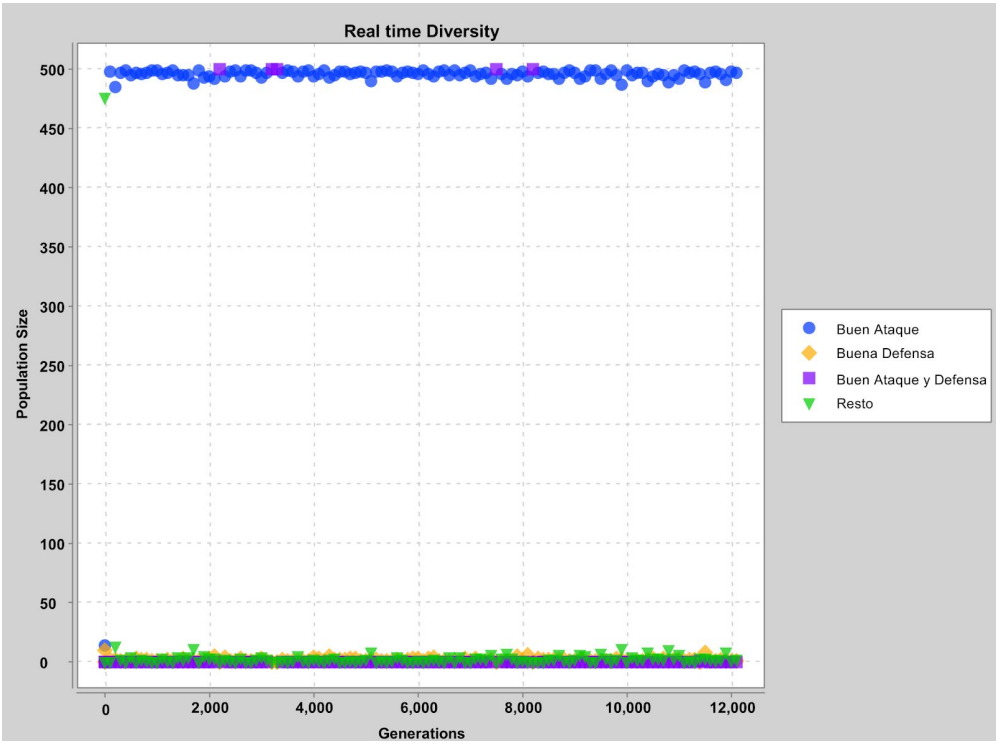
Boltzmann, Roulette, Elite, Proba, Ranking, 0.3, 0.7, UniformOneGeneMutator, KeepSomeAncestorsReplacer, DoublePointCross (0.3), Optimum conditioner (40)



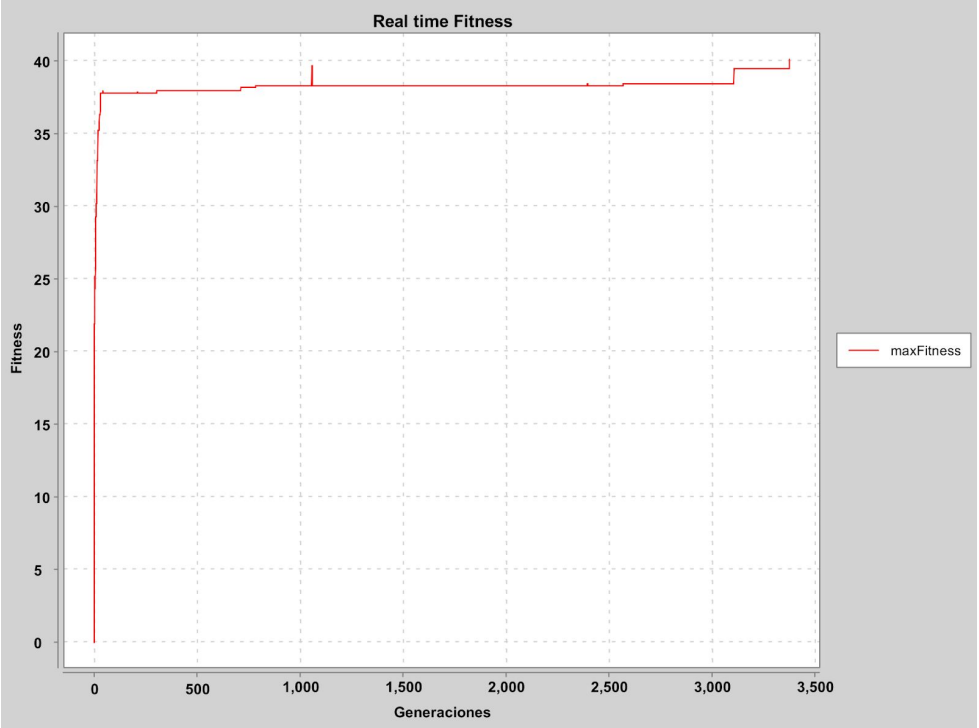


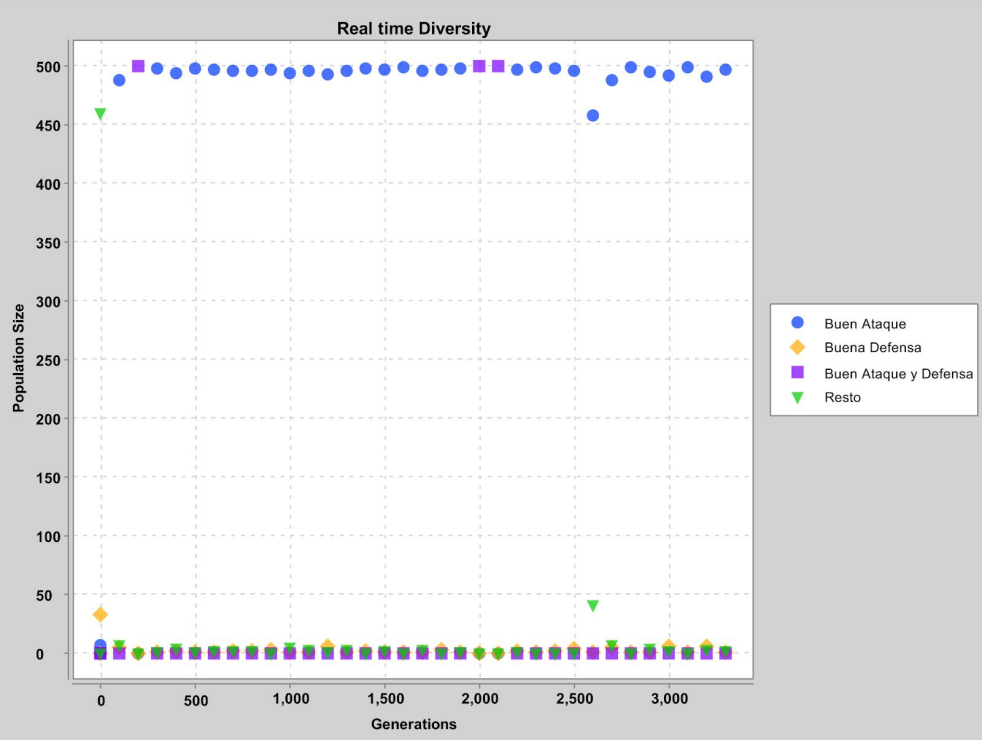
Deterministic, Boltzman, Elite, Proba , 0.3, 0.7, UniformOneGeneMutator, KeepSomeAncestorsReplacer, DoublePointCross (0.3), Optimum conditioner (40)



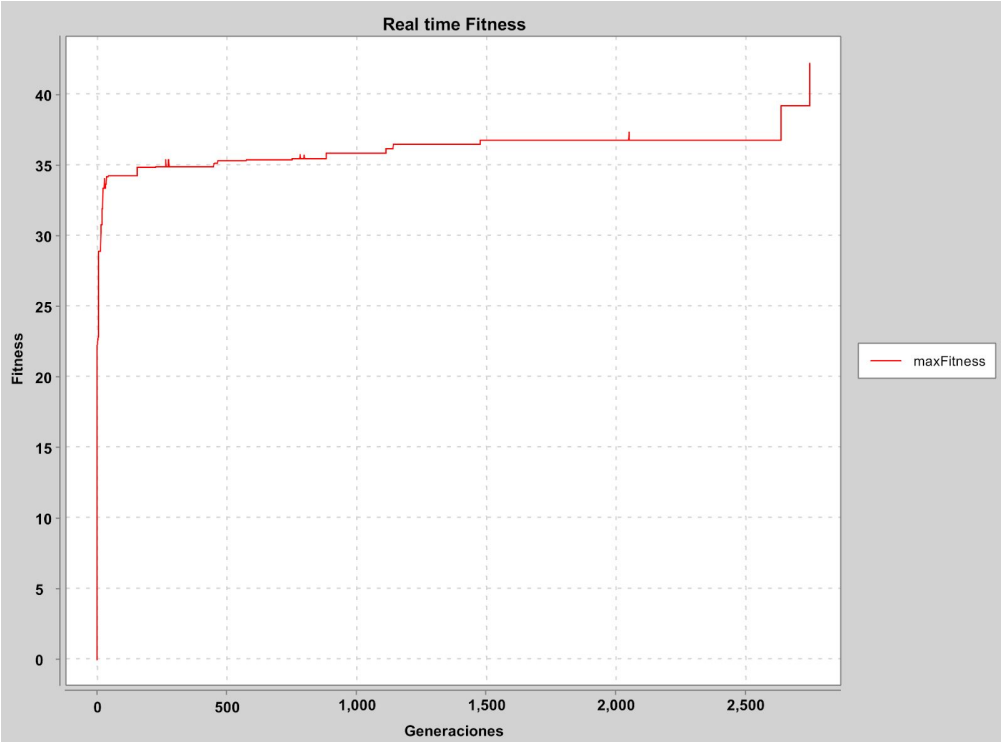


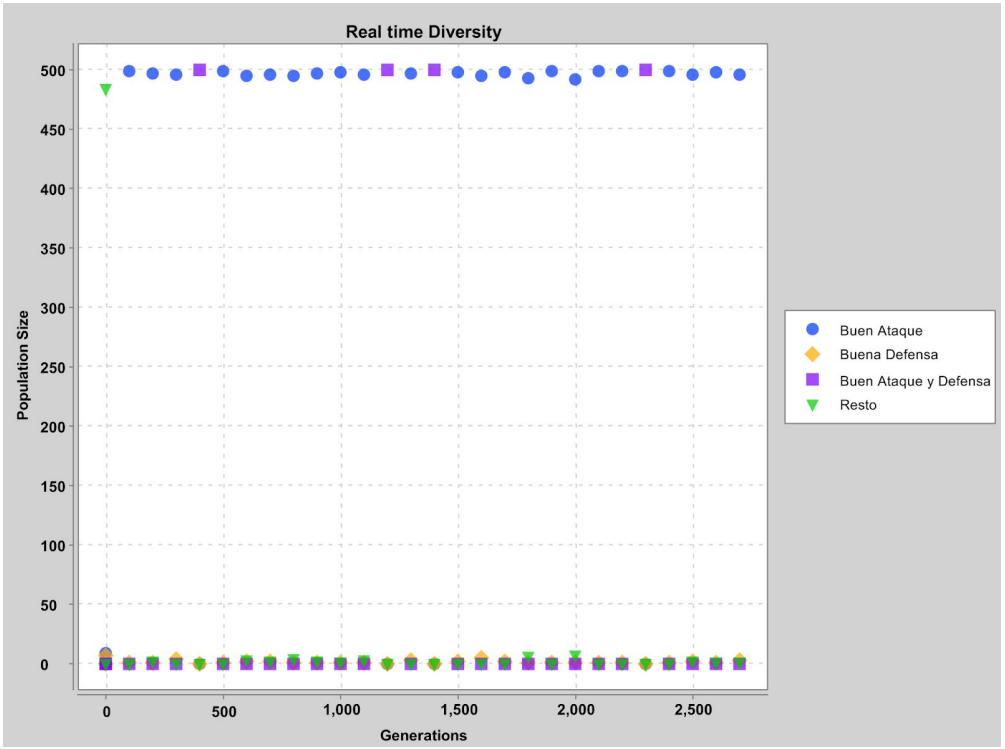
Universal, Deterministic, Boltzman, Elite, 0.3, 0.7, UniformOneGeneMutator, KeepSomeAncestorsReplacer, DoublePointCross (0.3), OptimumConditioner (40)



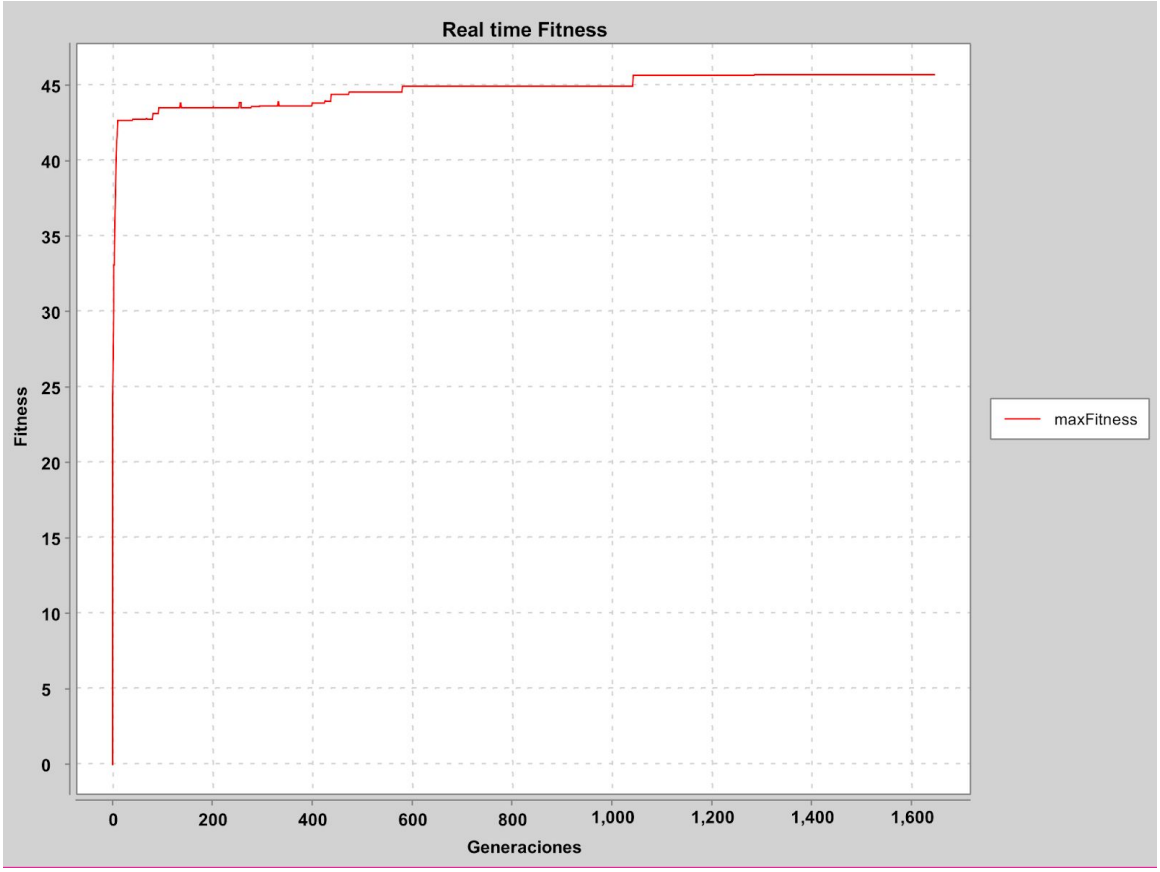


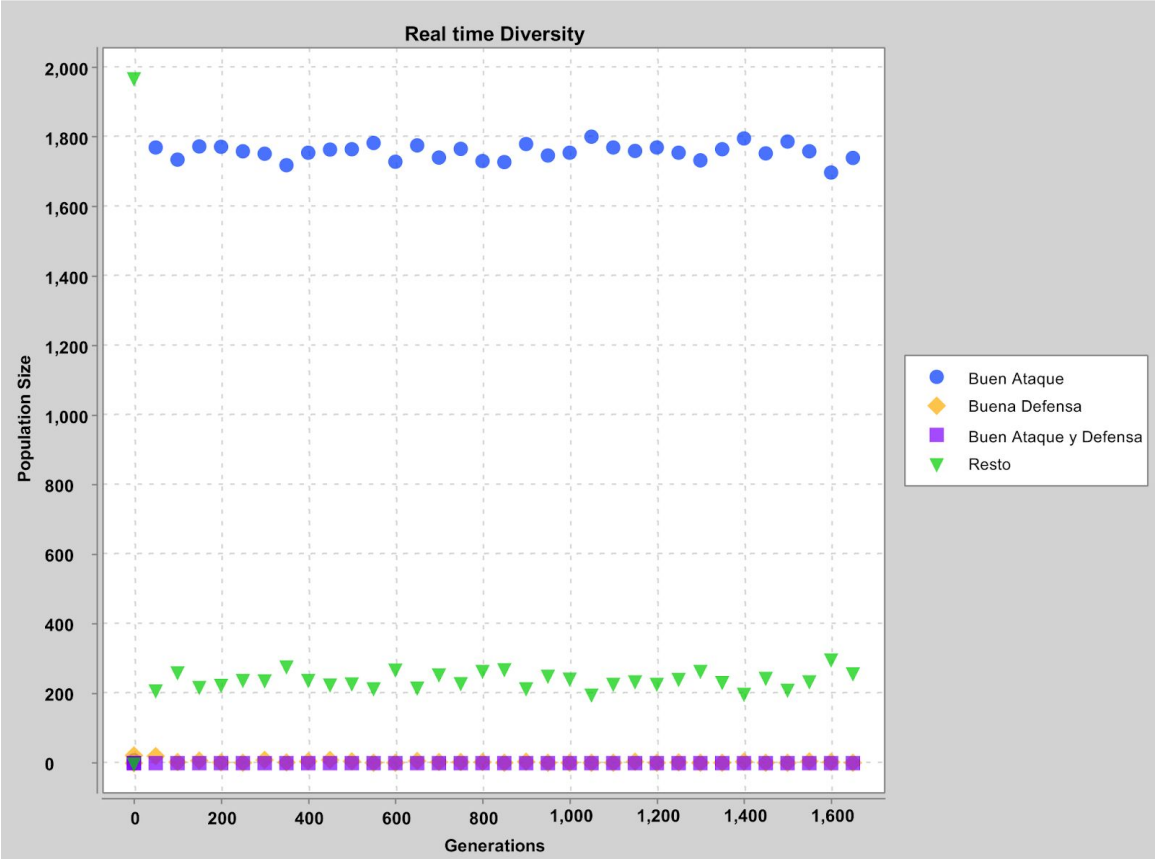
Roulette, Universal, Deterministic, Boltzmann, 0.3, 0.7, UniformOneGeneMutator, KeepSomeAncestorsReplacer, DoublePointCross (0.3), OptimumConditioner (40)





RankingSelection, RouletteSelection, UniversalSelection, Deterministic, 0.3, 0.7, UniformOneGeneMutator, MixAllReplacer, DoublePointCross (0.3), OptimumConditioner (40)


Real time Fitness
Fitness
45
40
35
30
25
20
15
10
5
0
0
200
400
600
800
1,000
1,200
1,400
1,600
Generaciones
maxFitness


Real time Diversity
Population Size
2,000
1,800
1,600
1,400
1,200
1,000
800
600
400
200
0
0
200
400
600
800
1,000
1,200
1,400
1,600
Generations
Buen Ataque
Buena Defensa
Buen Ataque y Defensa
Resto

EliteSelection, ProbabilisticTournamentSelection, RankingSelection, RouletteSelection, RatioA: 0.5, RatioB: 0.5, AnnularCross(0.7), KeepSomeAncestorsReplacer, UniformOneGeneMutator