



Guerrero 3

# Algoritmos genéticos

Sistemas de Inteligencia Artificial

Grupo 4

Agopian, Michel

Rossi, Melisa Anabella

Zannini, Franco Michel

**ITBA**

# Introducción



- ◆ OBJETIVO : Buscar un equipamiento óptimo para un individuo propuesto por la cátedra mediante el uso de algoritmos genéticos.
- ◆ INDIVIDUO: Guerrero 3

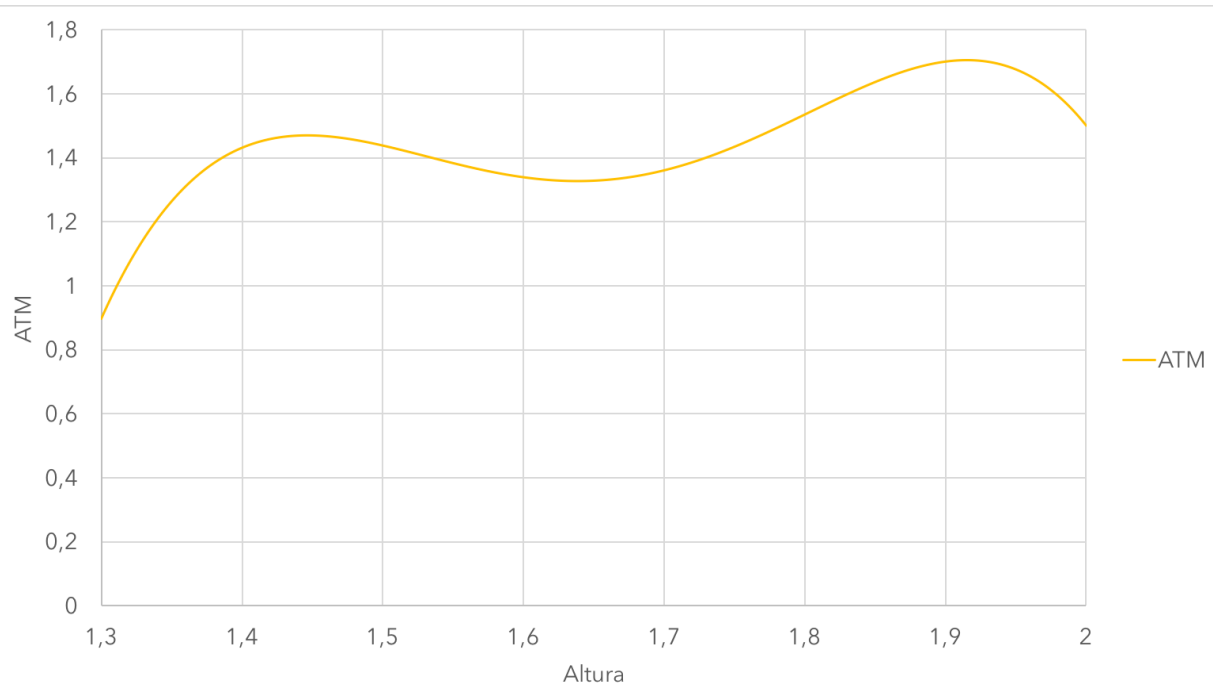


- ◆ Se analizaron datos para encontrar un valor cercano al óptimo utilizando fuerza bruta.
- ◆ Se tomara ese valor como referencia al realizar el algoritmo genético

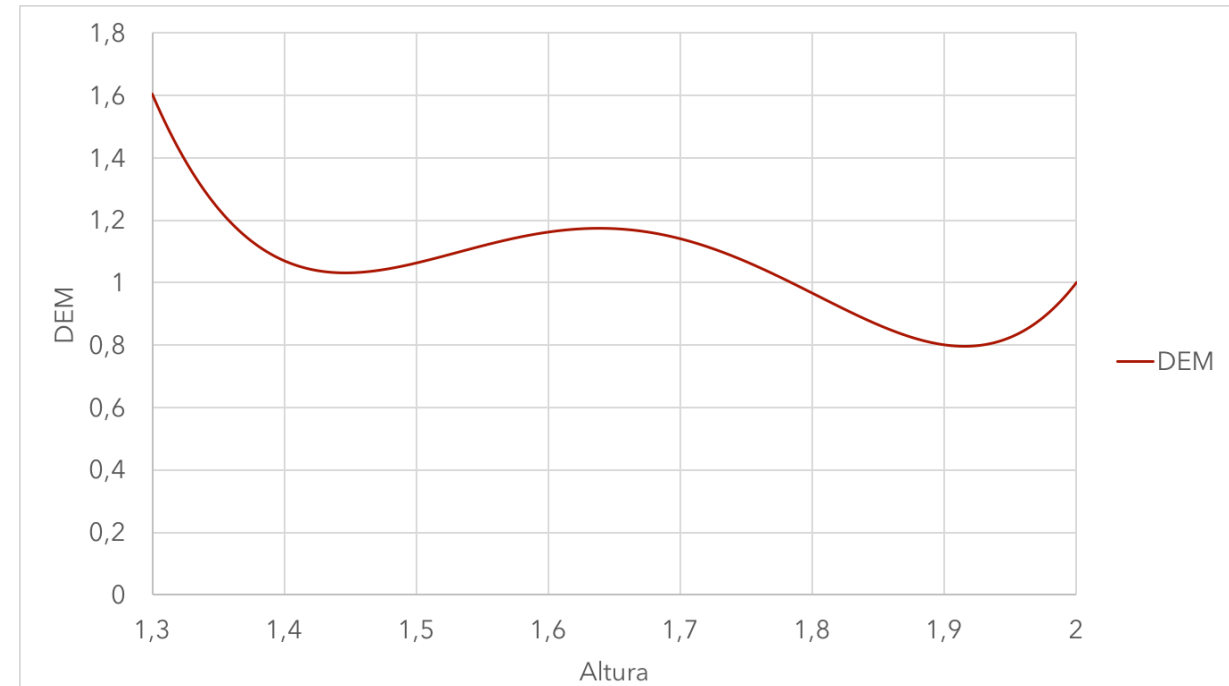
# Análisis de datos - Altura



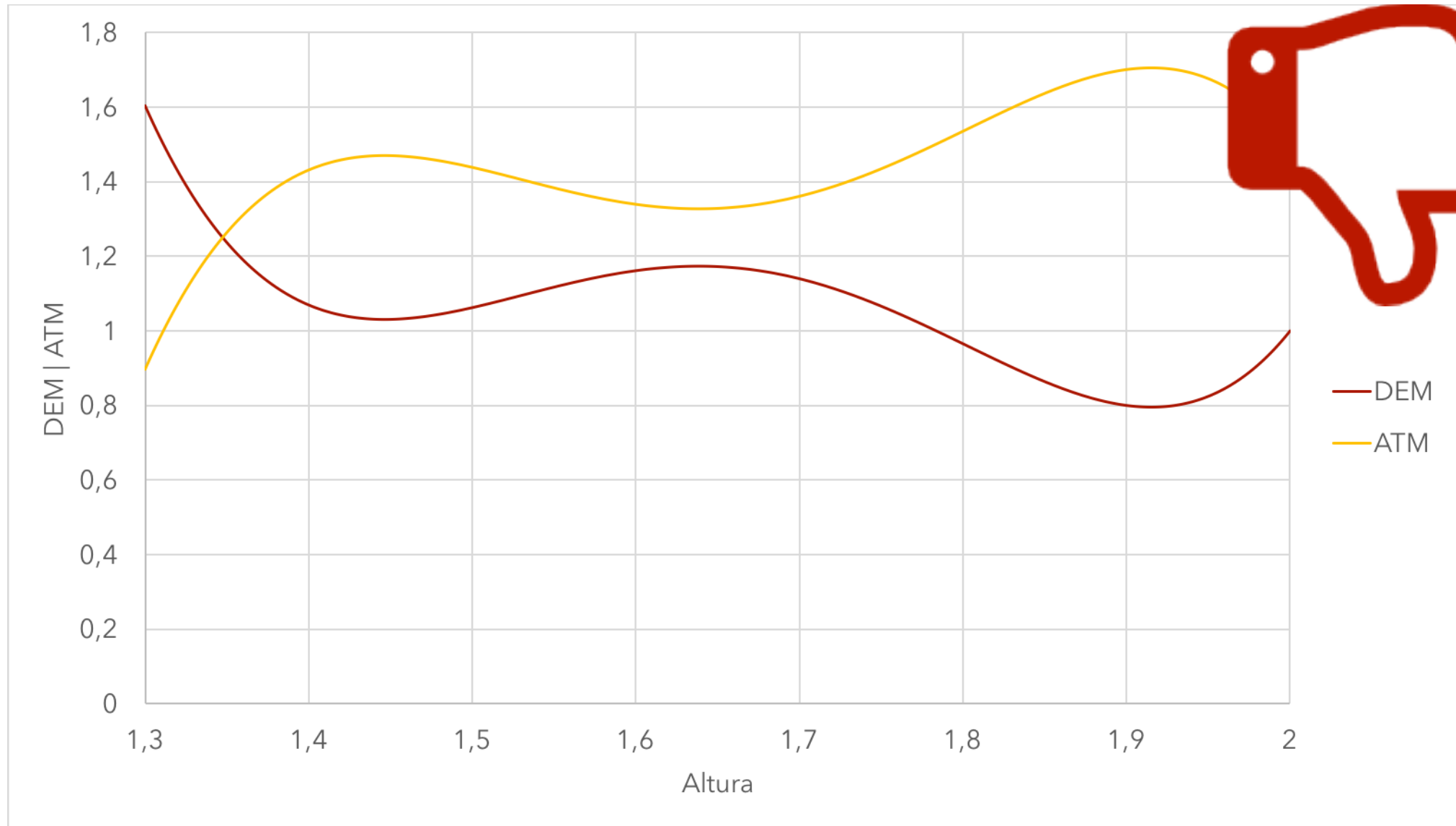
## ATM



## DTM



# Análisis de datos - Altura



# Análisis de datos - Equipamiento



- ◆ Se realizó fuerza bruta con todos los tipos de equipamiento, obteniéndose el máximo valor de fitness para una altura en particular.
- ◆ Se tomo como  $h = 1.92$
- ◆ Se obtuvo como valor máximo fitness = 20.5994

# Implementación

---



- ◆ MODELO
- ◆ ALGORITMO GENETICO
- ◆ UTILS

# Implementación - Modelo



Individual

- **List<Gene>** chromosome
- **double** relativeAptitude
- **double** accumulatedAptitude
- + compareTo()
- + getFitness()



ProblemIndividual

- **double** strength
- **double** agility
- **double** skill
- **double** life
- **double** resistance



WarriorIndividual

- **double** STRENGTH\_COEF
- **double** AGILITY\_COEF
- **double** SKILL\_COEF
- **double** RESISTANCE\_COEF
- **double** LIFE\_COEF



# Implementación - Modelo



Gene 

- double value
- double maxValue
- double minValue
- + mutate()

# Implementación - Modelo



Gene 

- double value
- double maxValue
- double minValue

+ mutate()

Setea un valor random al gen que irá desde el minValue al maxValue

# Implementación - Modelo



Gene 

- double value  
- double max  
- double min  
+ mutate()

Representa un índice de la lista de equipamientos correspondiente para los distintos elementos y en el caso de la altura guarda su valor.

Warrior chromosome (List<Gene>):

ARMA

GUANTE

REMERA

GORRO

BOTA

ALTURA

# Implementación - Modelo



EquipmentManager

- **List<List<Equipment>>** equipments
- + getStrength(List<Gene> chromosome)
- + getAgility(List<Gene> chromosome)
- + getLife (List<Gene> chromosome)
- + getResistance(List<Gene> chromosome)
- + getLife(List<Gene> chromosome)

Se encarga de leer los archivos brindados por la cátedra y guardar los valores en la lista siguiendo el mismo orden que el cromosoma.

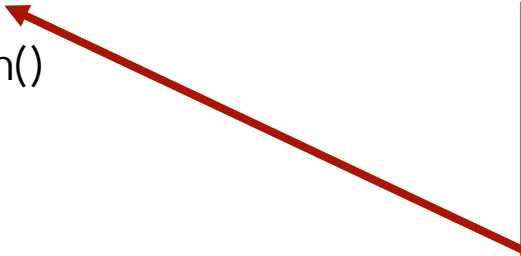
# Implementación - Algoritmo



EquipmentAlgorithm 

+ run()

+ getWarriorPopulation()



```
while (criterio de corte -> no debe terminar) {  
    seleccionar padres con el método 1  
    seleccionar padres restantes con método 2  
    combinar padres y generar nuevos  
    individuos  
    mutar con probabilidad p  
    Reemplazar para generar nueva población  
}
```

# Implementación - Algoritmo



CombinationMethod 

*OnePoint, TwoPoints, Anular, Uniform*

+ combine(List<Gene> c1, List<Gene> c2)

ReplacementMethod 

*ReplaceAll, ChooseN, ChooseK*

+ replace(List<Individual> p, List<Individual> ng,  
MethodPercentage rp1, MethodPercentage rp2)

CombineSelection 

*Combine2, CombineAll, CombineRandom, CombineBest*

+ getCombined(List<Individual> parents, CombinationMethod cm)

MutationMethod 

*NonUniform, Classic*

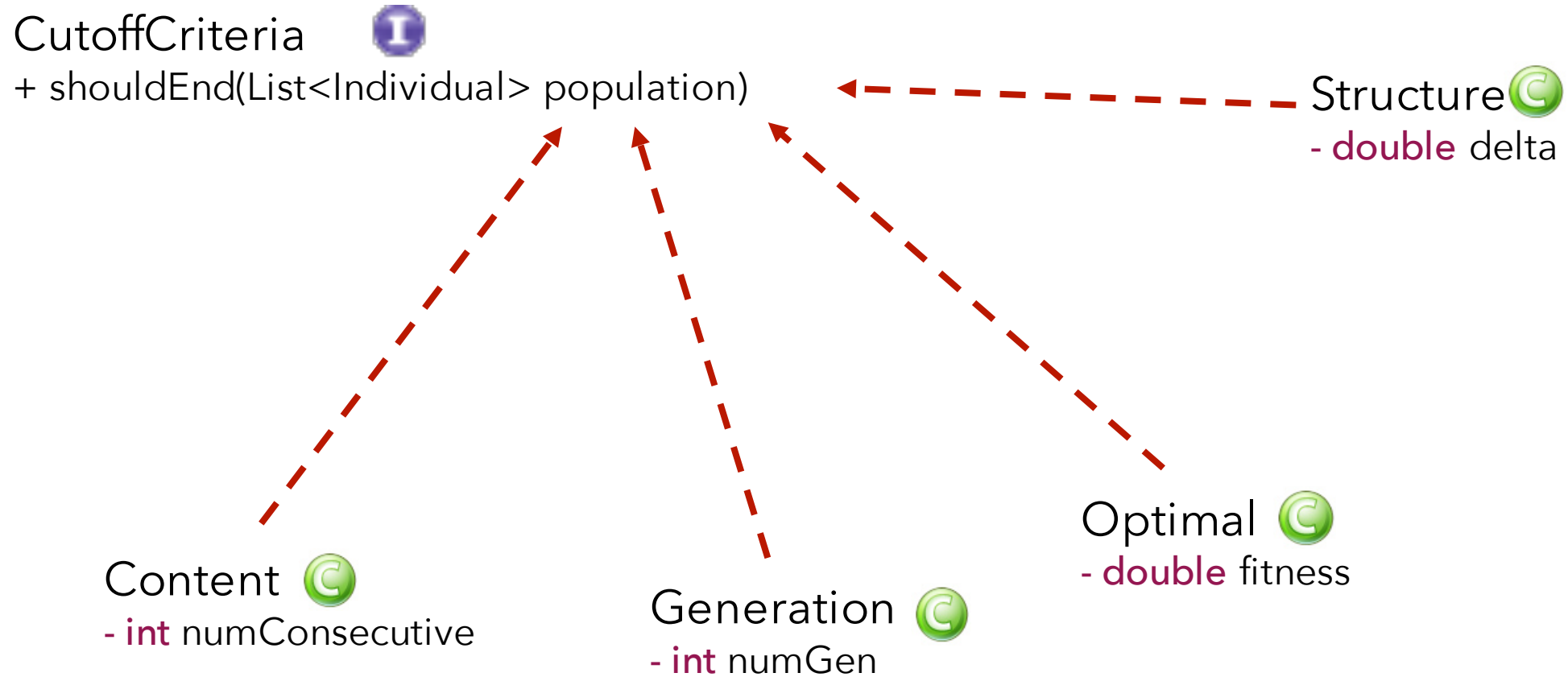
+ mutate(List<Gene> chromosome)

SelectionMethod 

*Elite, Random, Roulette, Universal, Boltzmann, DeterministicTournament,  
ProbabilisticTournament, Ranking;*

+ getSelected(List<Individual> p, int n)

# Implementación - Algoritmo



# Implementación - Utils



## PropertyManager

Toma los datos de el archivo `algorithm.properties` pasandolo a las clases correspondientes al iniciar el algoritmo y guarda estos valores para su uso futuro

## BoltzmannUtils

Clase utilizada para guardar los datos de temperatura y calcular aptitudes según el método de selección Boltzmann

## EquipmentAlgorithm

- + `getTotalFitness(List<Individual> population)`
- + `setAccumulatedApitude(List<Individual> population)`
- + `setRankingApitude(List<Individual> population)`



# Resultados



- ◆ Se realizaron pruebas experimentales para encontrar configuraciones que generen un buen resultado.
- ◆ Se seteo una semilla para que los resultados sean comparables.

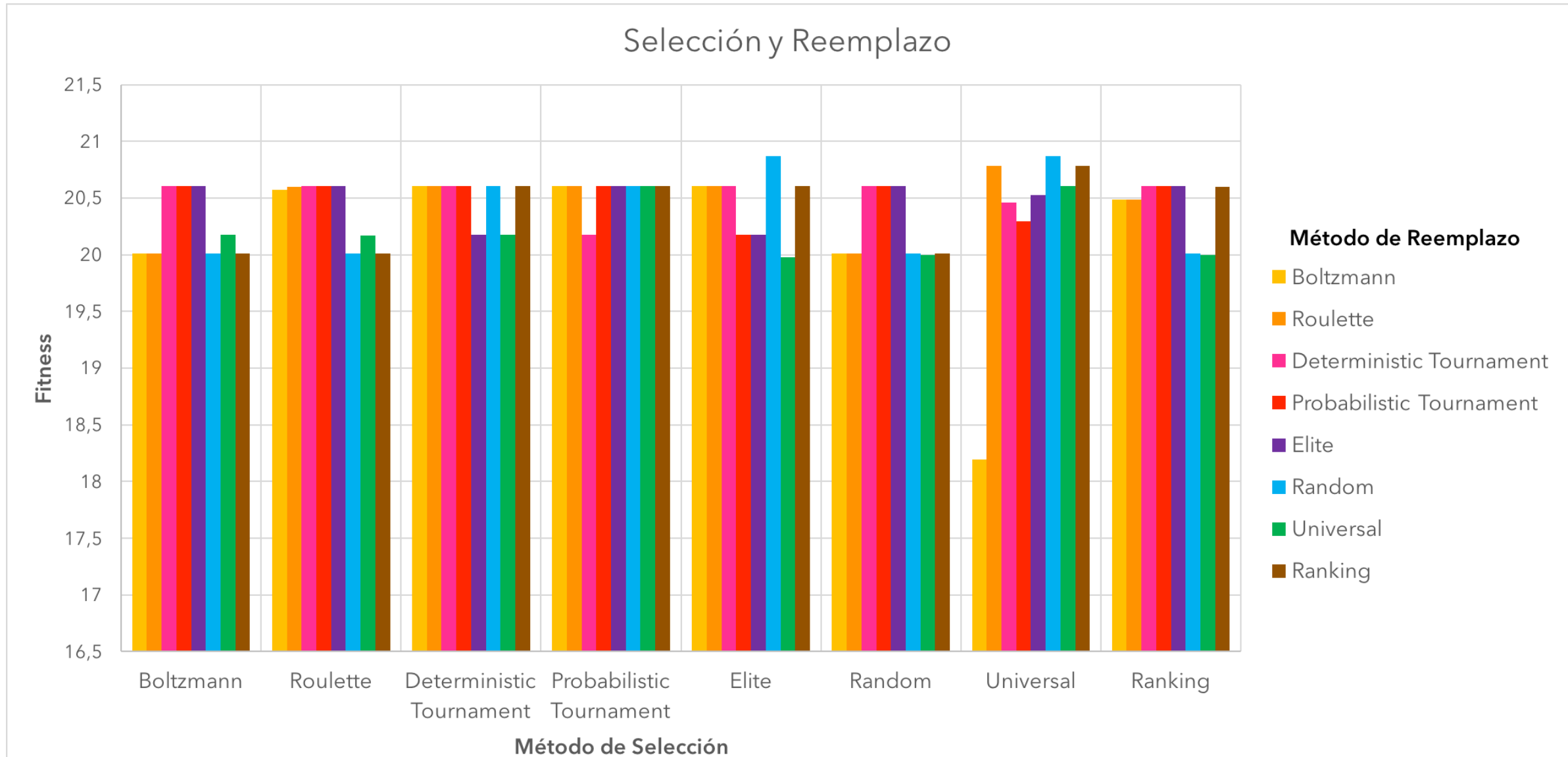
# Resultados – Prueba 1



## Prueba de combinaciones de reemplazo y selección

- $N = 200$
- $K = 120$
- Selección Elite 20%
- Método de reemplazo 2
- Reemplazo Elite 20%
- Mutación: No uniforme,  $p_m = 0.1$
- Criterio de corte: 1000 generaciones que no superan fitness
- Combinación: Combine2, TwoPoints

# Resultados – Prueba 1



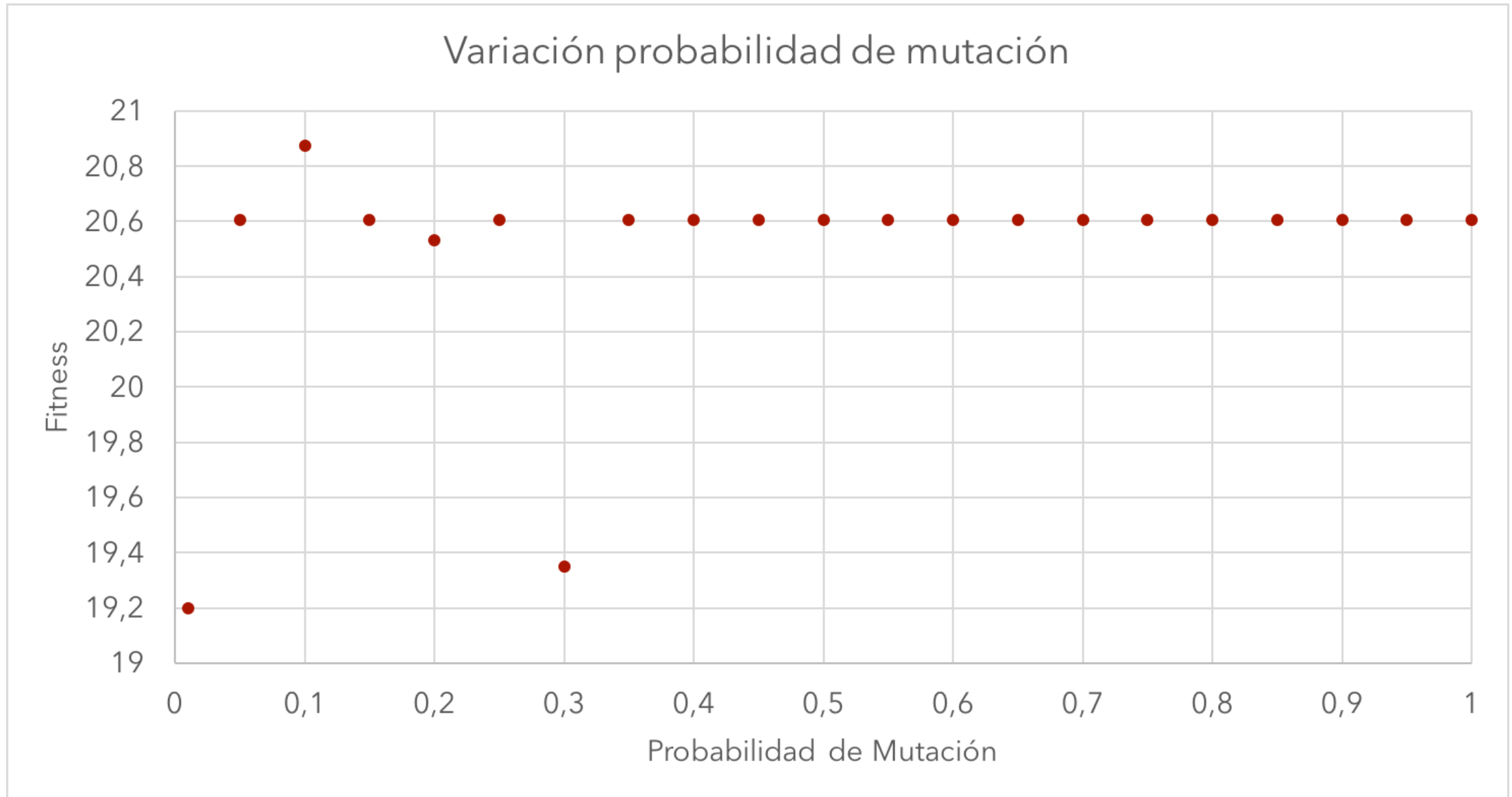
# Resultados – Prueba 2



## Prueba de mutación

- $N = 200$
- $K = 120$
- Selección Elite 20%, Elite 80%
- Método de reemplazo 2
- Reemplazo Elite 20%, Random 80%
- Mutación: No uniforme
- Criterio de corte: 1000 generaciones que no superan fitness
- Combinación: Combine2, TwoPoints

# Resultados – Prueba 2



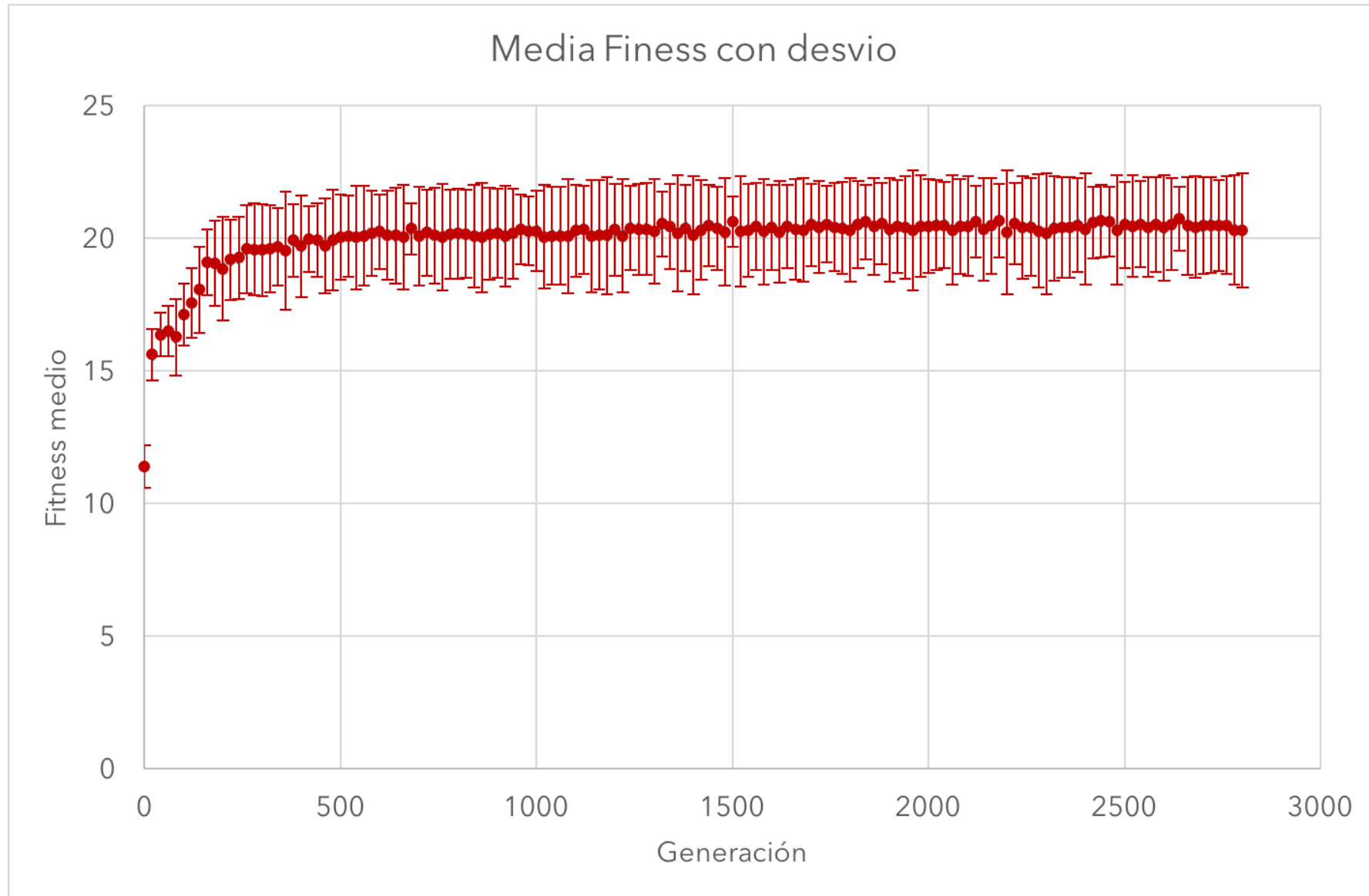
# Resultados – Prueba 3



Fitness medio con desvio estandar

- $N = 200$
- $K = 120$
- Selección Elite 20%, Elite 80%
- Método de reemplazo 2
- Reemplazo Elite 20%, Random 80%
- Mutación: No uniforme,  $m = 0.1$
- Criterio de corte: 1000 generaciones que no superan fitness
- Combinación: Combine2, TwoPoints

# Resultados – Prueba 3



# Resultados – Prueba 3

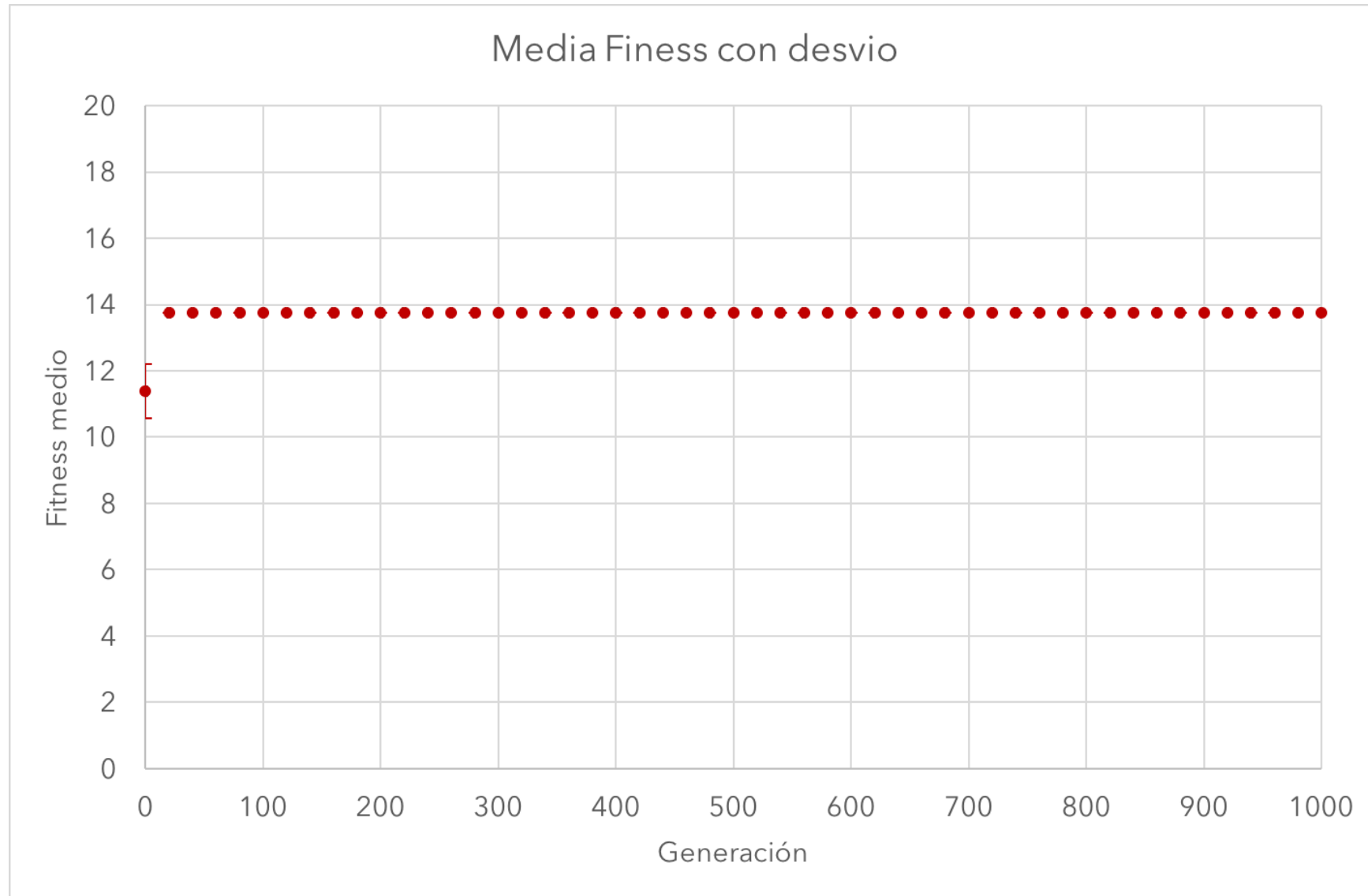


Fitness medio con desvío estándar

- $N = 200$
- $K = 120$
- Selección Elite 20%, Elite 80%
- Método de reemplazo 2
- Reemplazo Elite 20%, Random 80%
- Mutación: No uniforme,  $pm = 0$
- Criterio de corte: 1000 generaciones que no superan fitness
- Combinación: Combine2, TwoPoints



# Resultados – Prueba 3



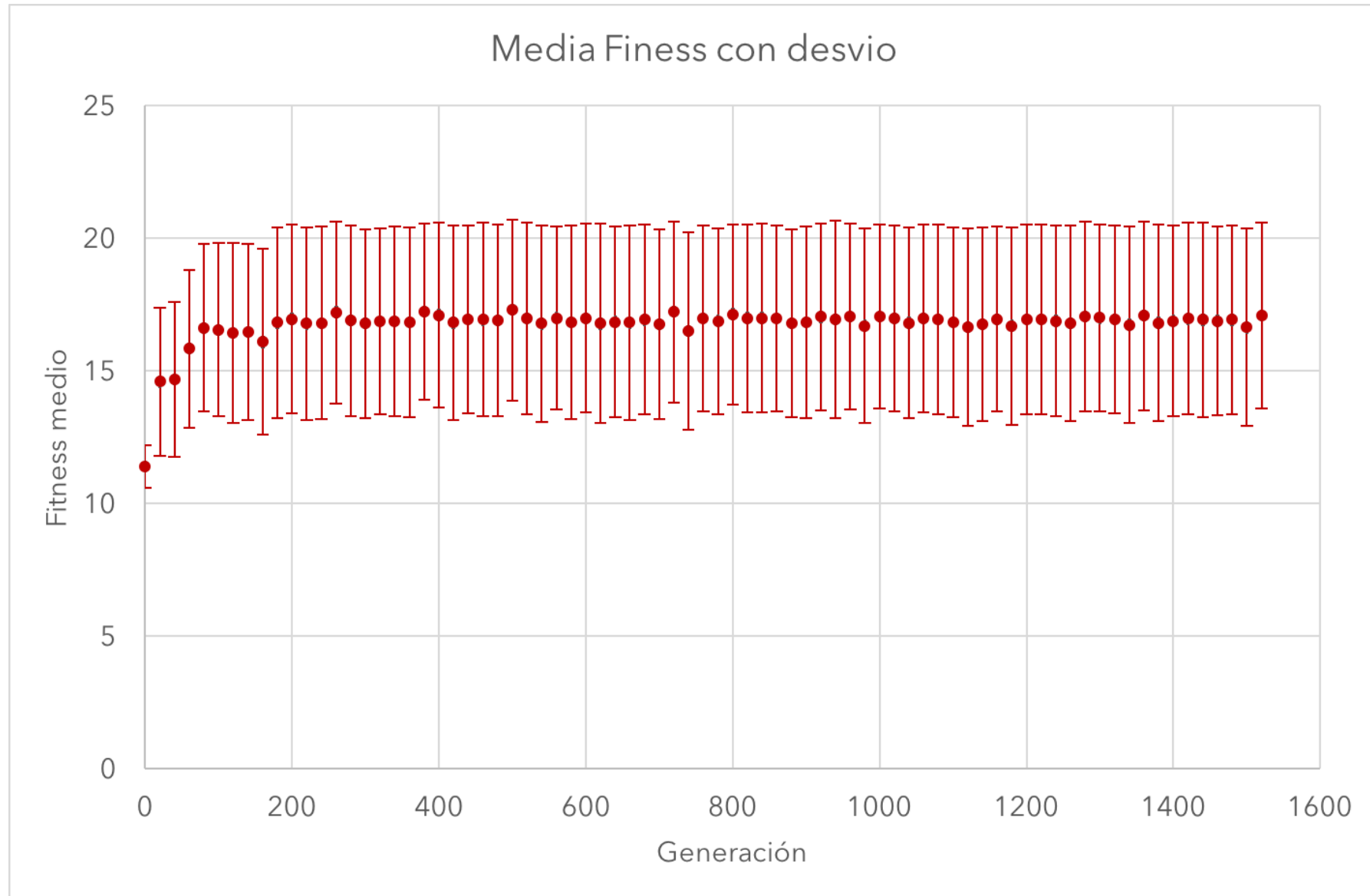
# Resultados – Prueba 3



Fitness medio con desvío estándar

- $N = 200$
- $K = 120$
- Selección Elite 20%, Elite 80%
- Método de reemplazo 2
- Reemplazo Elite 20%, Random 80%
- Mutación: No uniforme ,  $pm = 1$
- Criterio de corte: 1000 generaciones que no superan fitness
- Combinación: Combine2, TwoPoints

# Resultados – Prueba 3



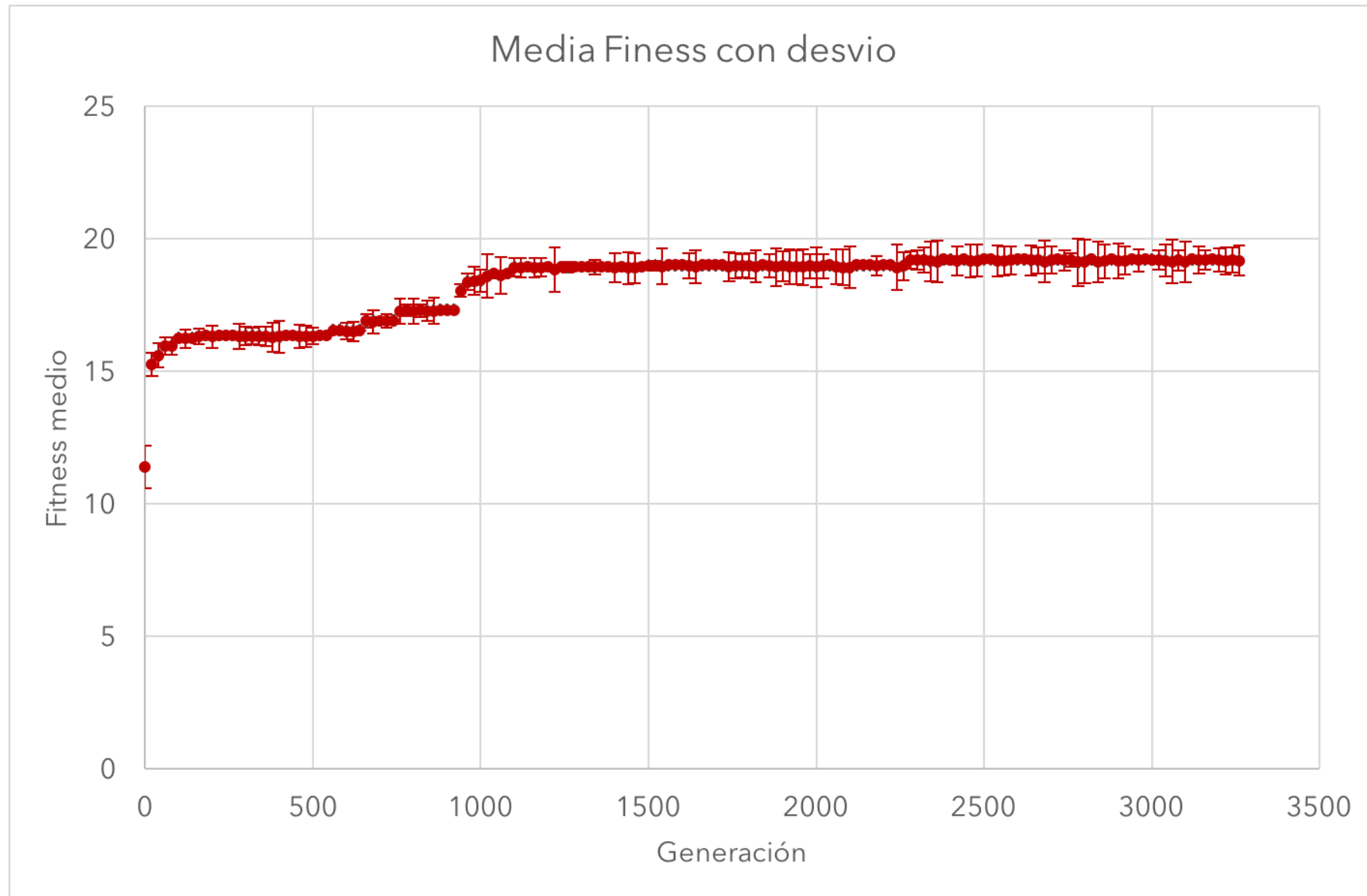
# Resultados – Prueba 3



Fitness medio con desvío estándar

- $N = 200$
- $K = 10$
- Selección Elite 20%, Elite 80%
- Método de reemplazo 2
- Reemplazo Elite 20%, Random 80%
- Mutación: No uniforme,  $m = 0.1$
- Criterio de corte: 1000 generaciones que no superan fitness
- Combinación: Combine2, TwoPoints

# Resultados – Prueba 3



# Conclusiones



- ◆ Al utilizar como probabilidad de mutación un valor mayor a 0.3 se obtienen valores muy cercanos al valor de fitness óptimo encontrado → la mutación es buena ya que introduce variabilidad genética.
- ◆ Manteniendo el resto de las variables fijas, si variamos entre el método de mutación no uniforme y el método clásico, obtenemos un fitness superior utilizando el primero → se agrega variabilidad genética evitando que las generaciones se estanquen en máximos locales.
- ◆ La mejor combinación de métodos de selección y reemplazo encontrados son Elite 20% + 80% y Elite 20% + Random 80% respectivamente.
- ◆ Otra gran combinación de métodos de selección y reemplazo se encuentra combinando 20% Elite + 80% Boltzmann

# Conclusiones



- ◆ Se encontraron múltiples sub óptimos cuando se combinaban métodos tanto para la selección como para el reemplazo.
- ◆ Al utilizar el método CombineBest, que combina al individuo de mejor fitness con el resto, se pierde variabilidad genética y no se alcanza un valor cercano al óptimo.
- ◆ Los métodos de reemplazo no garantizan que el sub óptimo de una generación esté presente en la generación siguiente. Por esta razón, es que se obtuvieron mejores resultados incluyendo en algún porcentaje, sin importar si es bajo, al método Elite.

# Muchas Gracias

