

Métodos de Búsqueda + Algoritmos Genéticos



Integrantes



Felipe Cupitó



Hernán Finucci



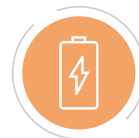
Sol Konfederak



**Juan Manuel De
Luca**



Azul Kim





Métodos de Búsqueda



Cubo Rubik

Problema

Representación del cubo



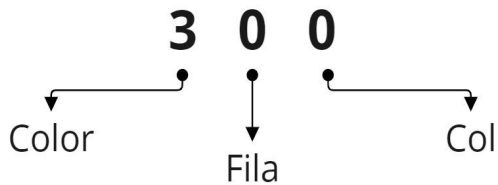
Heurísticas



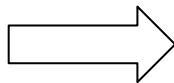
Algoritmos de resolución



Representación del cubo

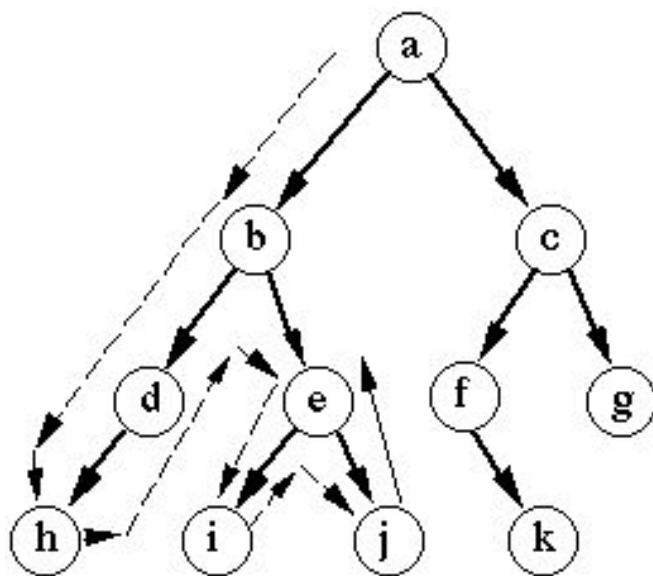


			300	301	302							
			310	311	312							
			320	321	322							
000	001	002	400	401	402	200	201	202	500	501	502	
010	011	012	410	411	412	210	211	212	510	511	512	
020	021	022	420	421	422	220	221	222	520	521	522	
			100	101	102							
			110	111	112							
			120	121	122							

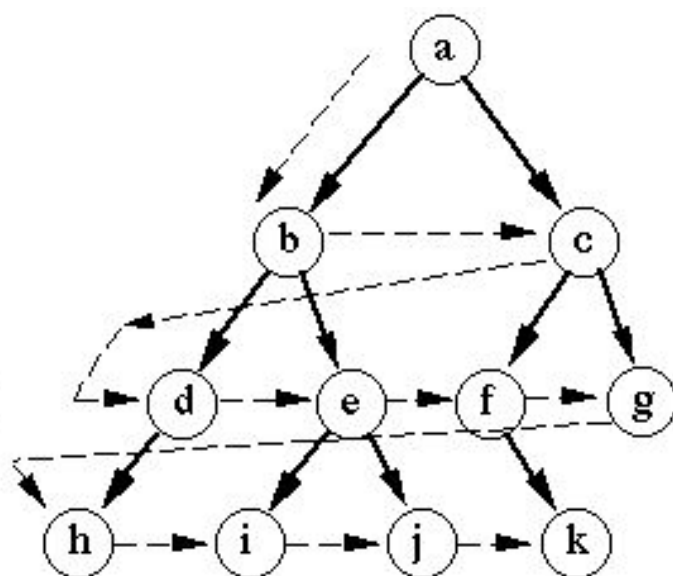


			300	301	402							
			310	311	412							
			022	012	102							
000	001	100	400	401	200	320	201	202	500	501	302	
010	011	101	410	411	112	321	211	212	510	511	312	
520	521	522	020	021	122	420	421	422	322	221	002	
			220	210	502							
			110	111	512							
			120	121	222							

Algoritmos de resolución



DFS



BFS

Diferencias entre A* y Greedy

A*

```
self.list = PriorityQueue()
```

...

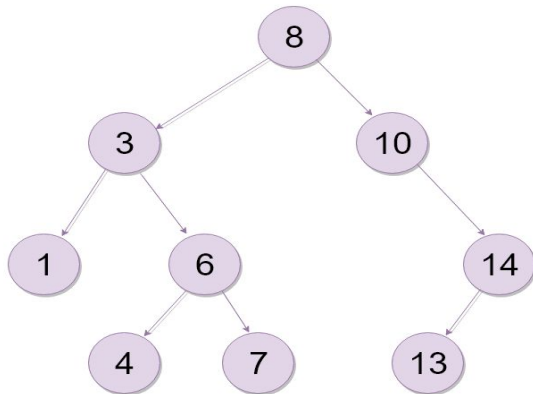
```
self.list.put((level + h_value), node))
```

Greedy

```
self.list = PriorityQueue()
```

...

```
self.list.put(h_value, node))
```

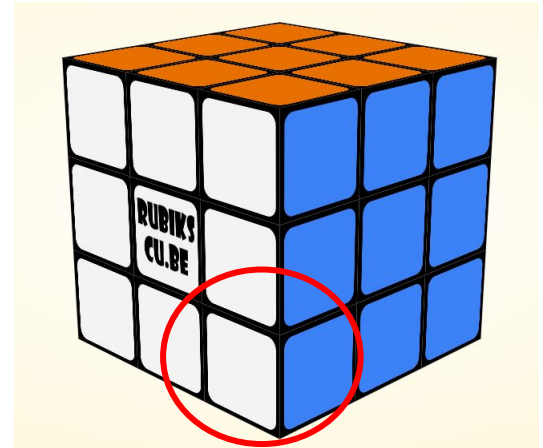
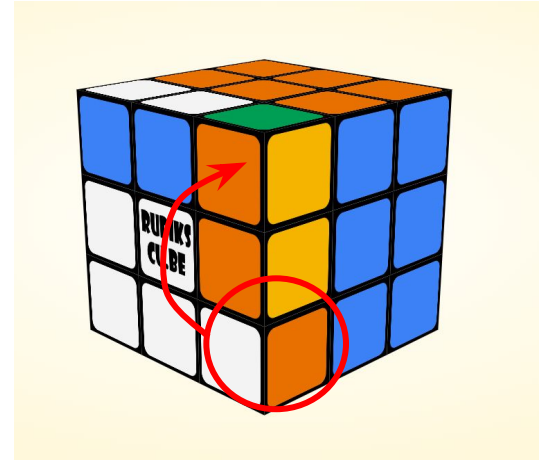


Heurísticas

1. Color (no admisibles)

Esta función cuenta cuántas celdas está en la cara correcta, pero no tiene en cuenta la orientación de la misma. No es admisible ya que puede sobrestimar el resultado.

Por ejemplo en esta imagen en la cara blanca contaría que hay 5 celdas bien posicionadas pero la celda marcada **no** está bien orientada.



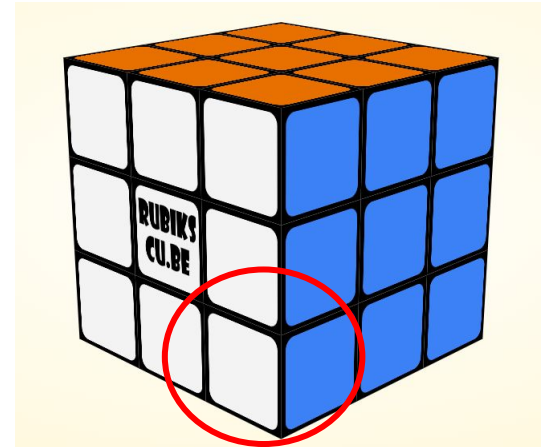
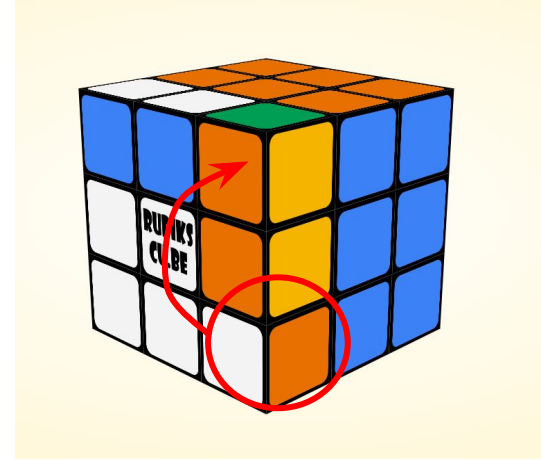
Heurísticas

2. Cubes

En esta heurística recibe un cubo y cuenta para cada cara cuántas de las n celdas están en su lugar y orientación correspondiente.

Luego este valor se lo restamos al total de celdas ($\text{total_celdas} - \#\text{celdasAdecuadas}$).

Por ejemplo en la imagen vemos que la celda blanca marcada tiene el color correcto pero **no** está orientada correctamente



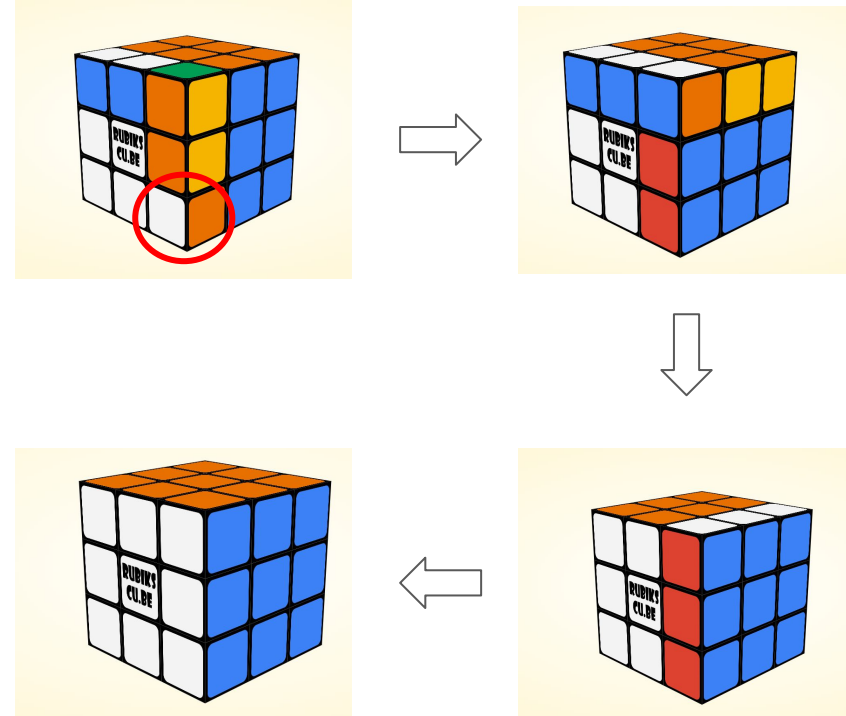
Heurísticas

3. Manhattan distance

Esta función calcula la cantidad mínima de movimientos (rotación) necesario para ubicar a cada cubie que no están en su cara y orientación correctas.

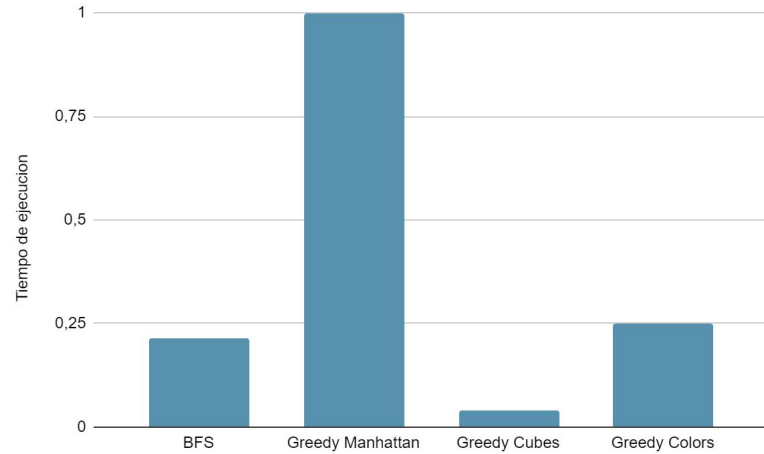
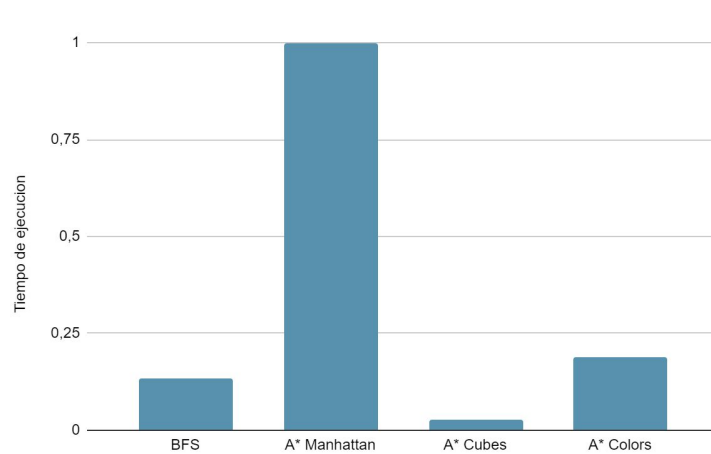
Luego sumamos la cantidad de movimiento total para posicionar todos los cubies y lo dividimos por la cantidad de cubies que se rotan en cada rotación

Por ejemplo en el caso de la imagen el cubie marcado necesita de 3 rotación.



Resultados

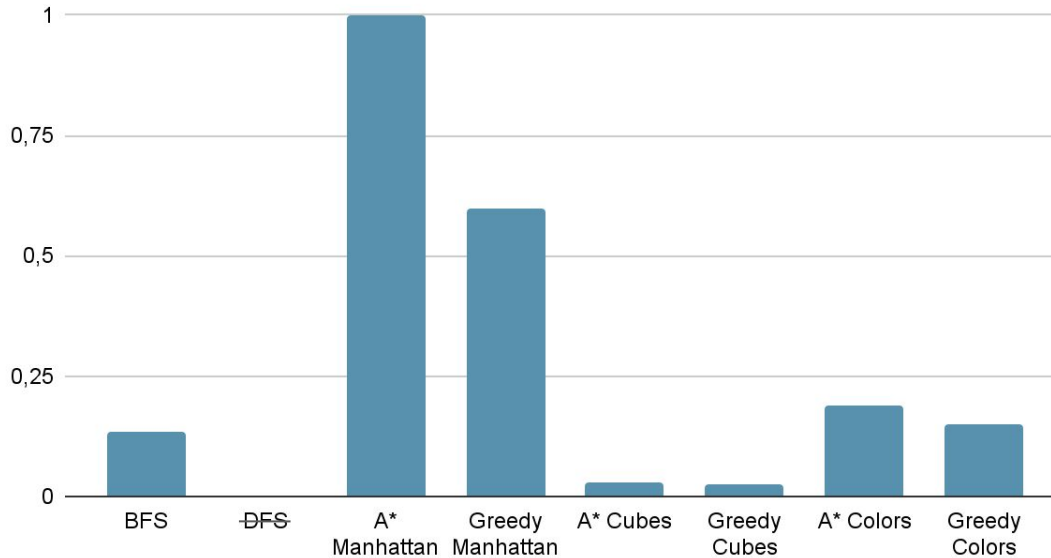
Comparación de Heurísticas por velocidad de ejecución



Condiciones iniciales: Inicializamos 5 cubos 3x3 y corrimos cada algoritmo sobre cada cubo. Los resultados finales fueron obtenidos promediando los resultados de los 5 cubos

Resultados

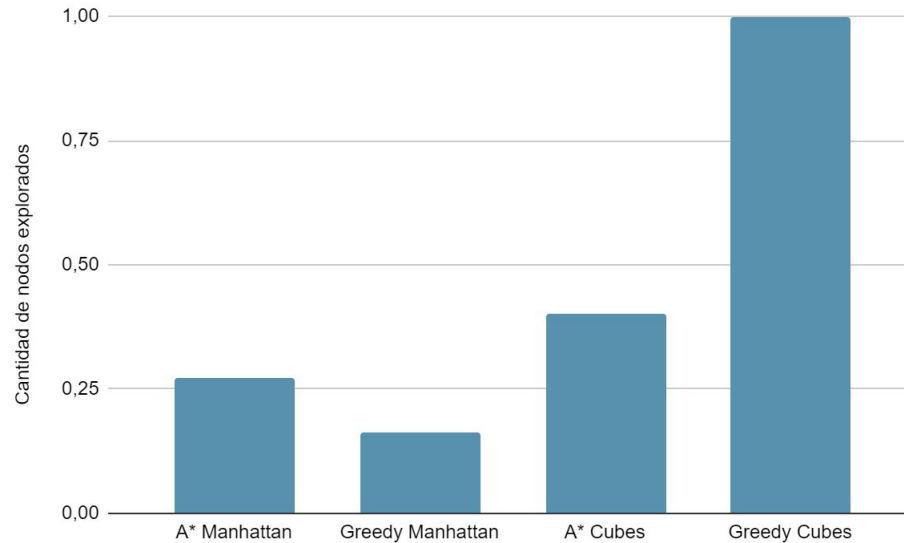
Comparación de Algoritmos por velocidad de ejecución



Condiciones iniciales: Inicializamos 5 cubos 3x3 y corrimos cada algoritmo sobre cada cubo. Los resultados finales fueron obtenidos promediando los resultados de los 5 cubos

Resultados

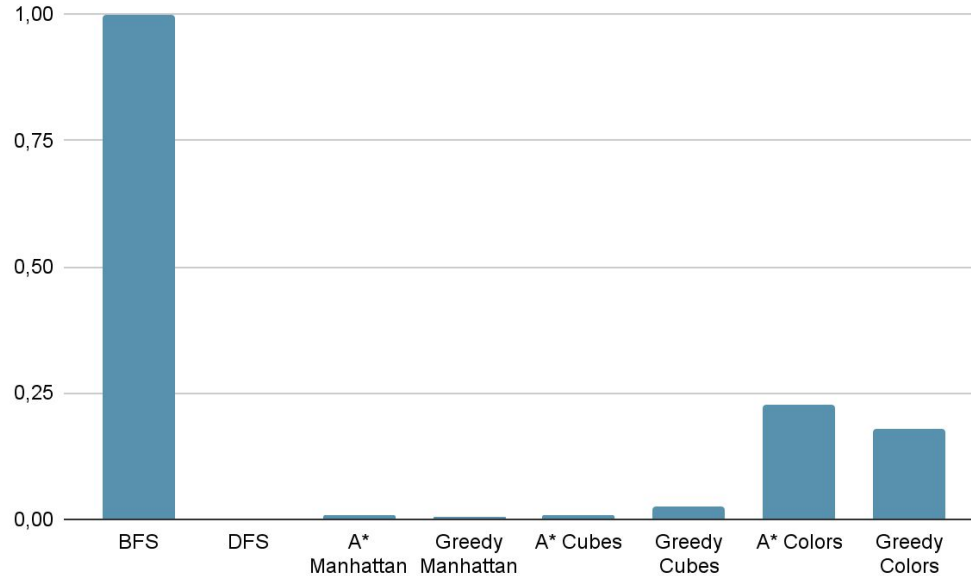
Comparación de Heurísticas "Manhattan" y "Cubes" por nodos explorados



Condiciones iniciales: Inicializamos 5 cubos 3x3 y corrimos cada algoritmo sobre cada cubo. Los resultados finales fueron obtenidos promediando los resultados de los 5 cubos

Resultados

Comparación de Algoritmos por nodos explorados



Condiciones iniciales: Inicializamos 5 cubos 3x3 y corrimos cada algoritmo sobre cada cubo. Los resultados finales fueron obtenidos promediando los resultados de los 5 cubos

Conclusion

- DFS: No es útil para resolver este problema
- BFS: Resuelve el problema mientras que no se mezcle mucho el cubo. Podria ser util en un cubo de 2x2
- Heurística Manhattan: Es una buena heurística ya que minimiza los nodos explorados, pero es muy costosa computacionalmente.

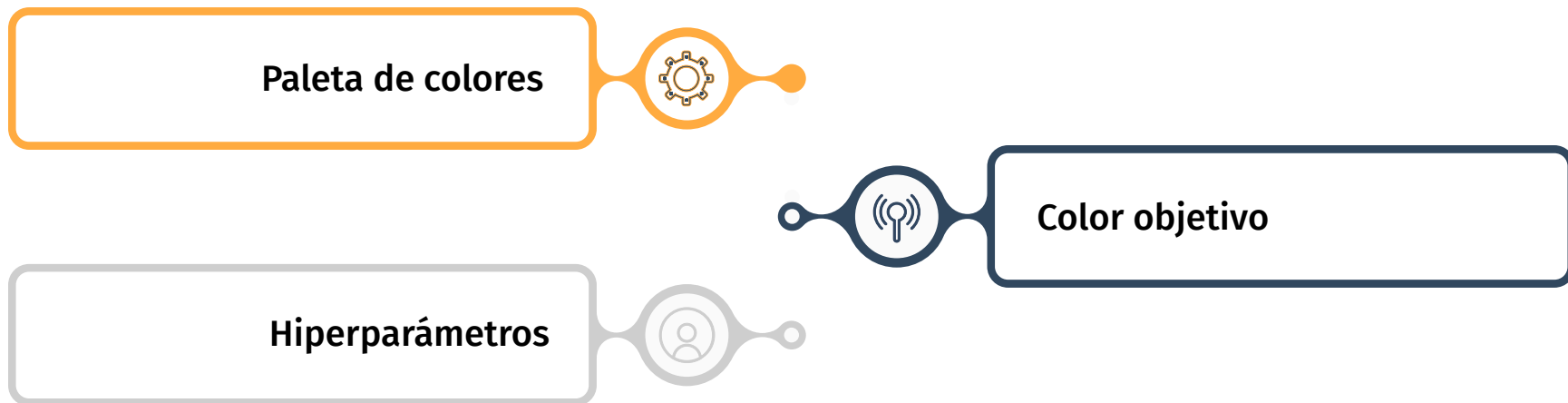


Algoritmos Genéticos



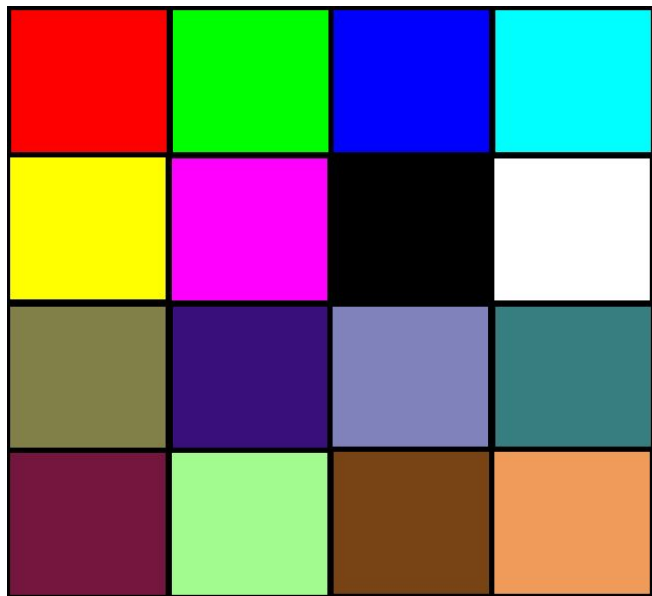
Mezcla de colores

Problema

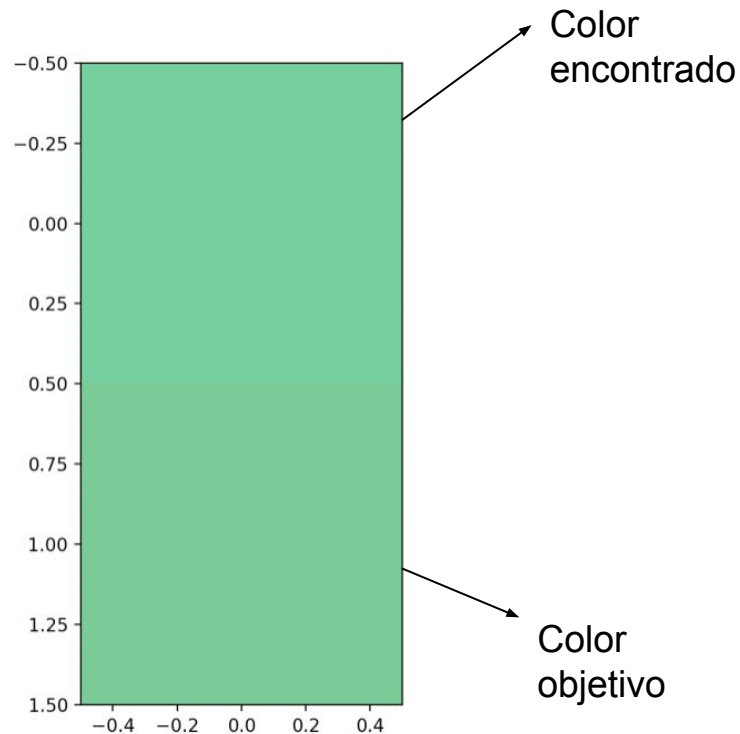


Problema (dramatización)

Paleta original



Crossbreed,
mutaciones



Estrategia



Individuo

Color
Componentes

• • • • •



Fitness

Distancia
euclídea

• • • • •



Cruza

Cruce uniforme

• • • • •



Mutación

Mutacion
uniforme

• • • • •



Selección

Elite
Ranking
Roulette

• • • • •

Condiciones de corte



Limite generacional

Provisto por el
usuario en un
archivo de
configuración



Condicion optima de fitness

Opción a ser
default

Genotipo

Gen Rojo

[0 - 255]

[0/1][0/1][0/1][0/1][0/1][0/1][0/1][0/1]

Gen Verde

[0 - 255]

[0/1][0/1][0/1][0/1][0/1][0/1][0/1][0/1]

Gen Azul

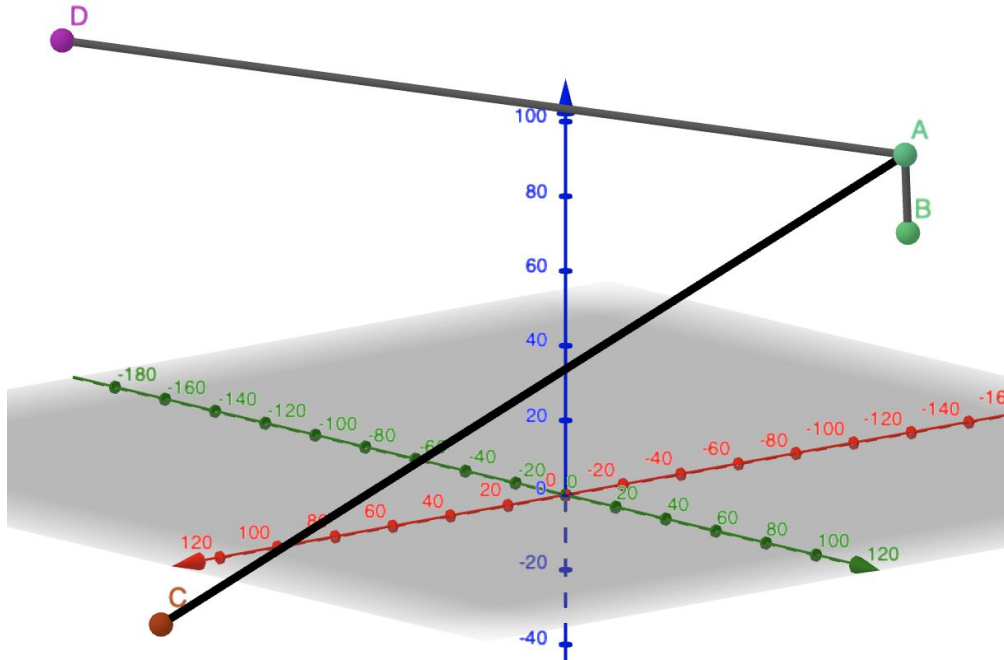
[0 - 255]

[0/1][0/1][0/1][0/1][0/1][0/1][0/1][0/1]

Fitness

$$f(i) = \frac{\sqrt{(i_{rojo} - t_{rojo})^2 + (i_{verde} - t_{verde})^2 + (i_{azul} - t_{azul})^2}}{dist_{m\acute{a}x}}$$

i: individuo
t: color objetivo



Cruza uniforme

Padre 1



Padre 2



=

Hijo 1

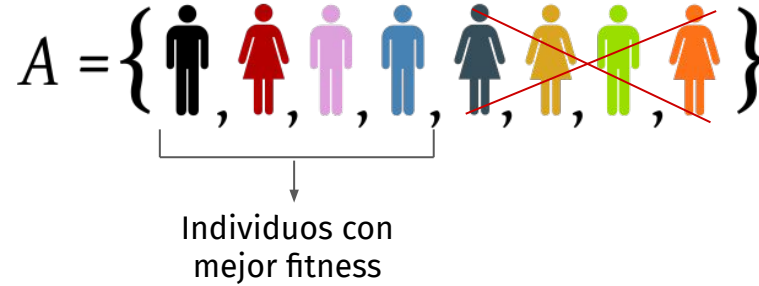
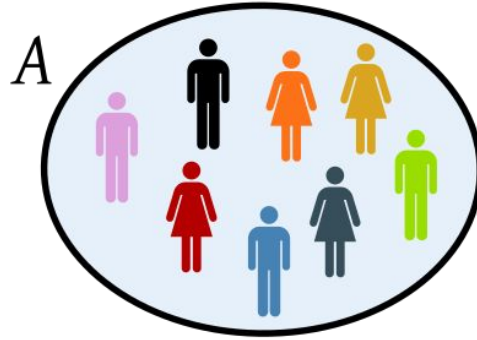


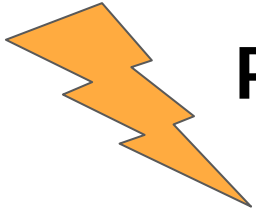
Hijo 2



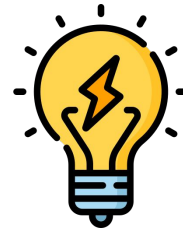
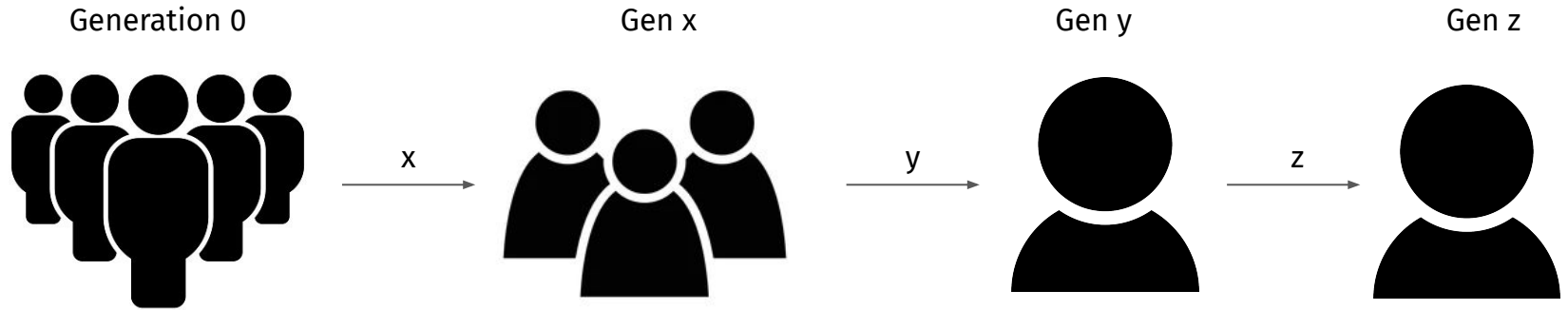
0,5

Selección: Elite





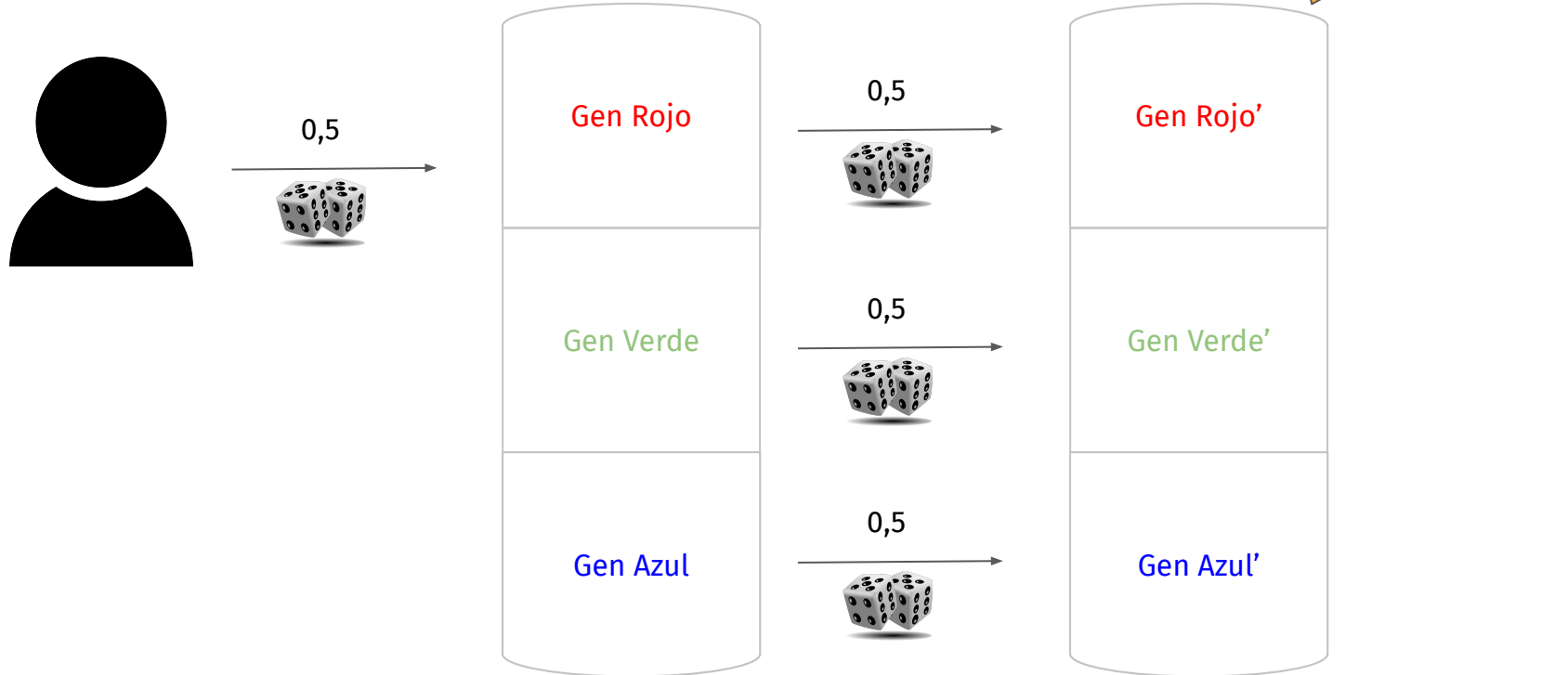
Problema : convergencia prematura



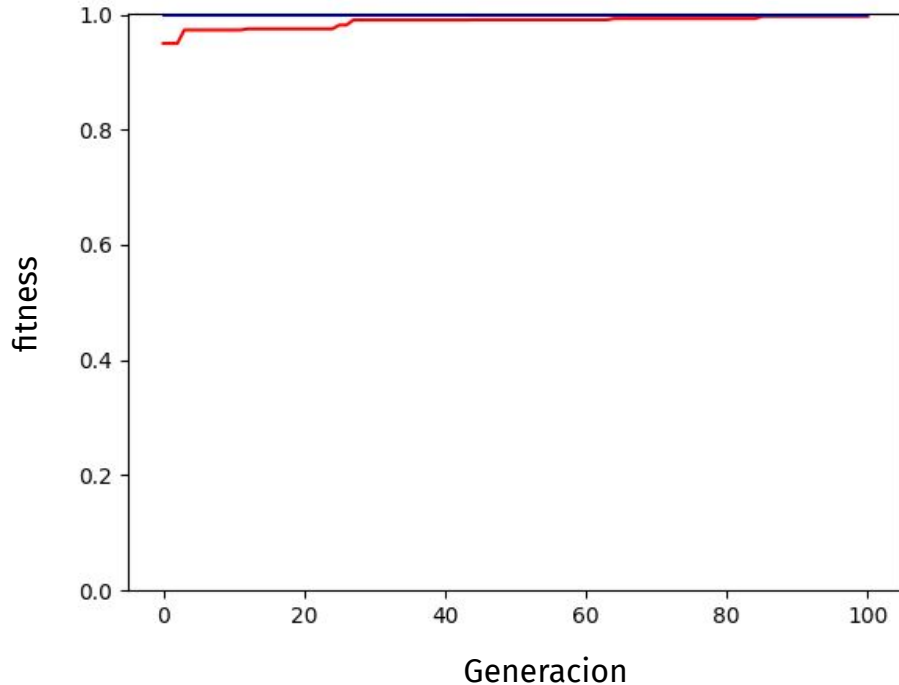
Solucion:

Generar diversidad incluyendo a los padres de la generación actual e incorporar mutación

Mutación uniforme



Evolución del fitness del color más aproximado

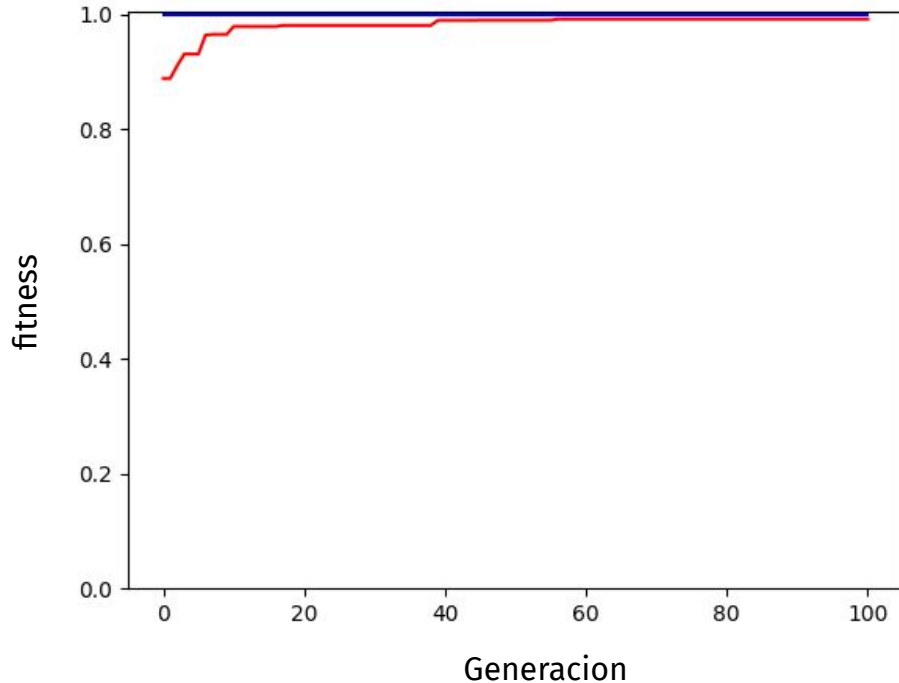


Selección: elite

Generaciones: 100

Fitness alcanzado: 0.9967980526238128

Evolución del fitness del color más aproximado

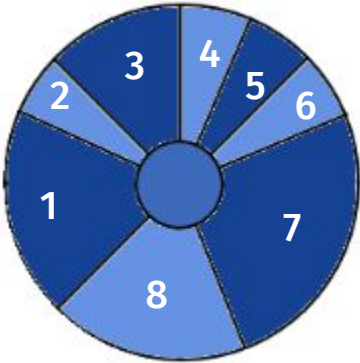
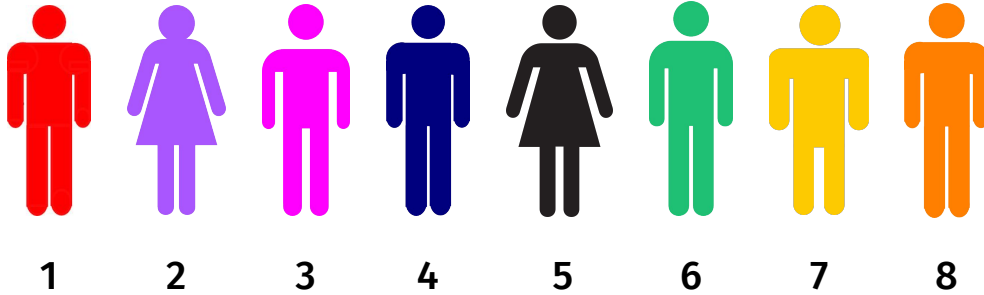


Selección: elite

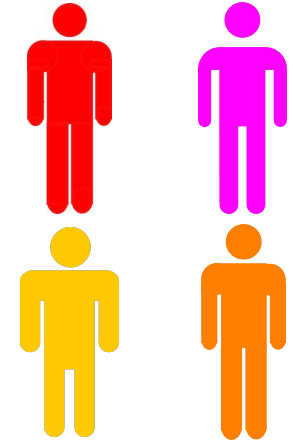
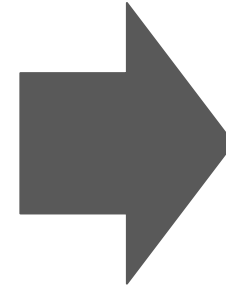
Generaciones: 100

Fitness alcanzado: 0.9915284435314929

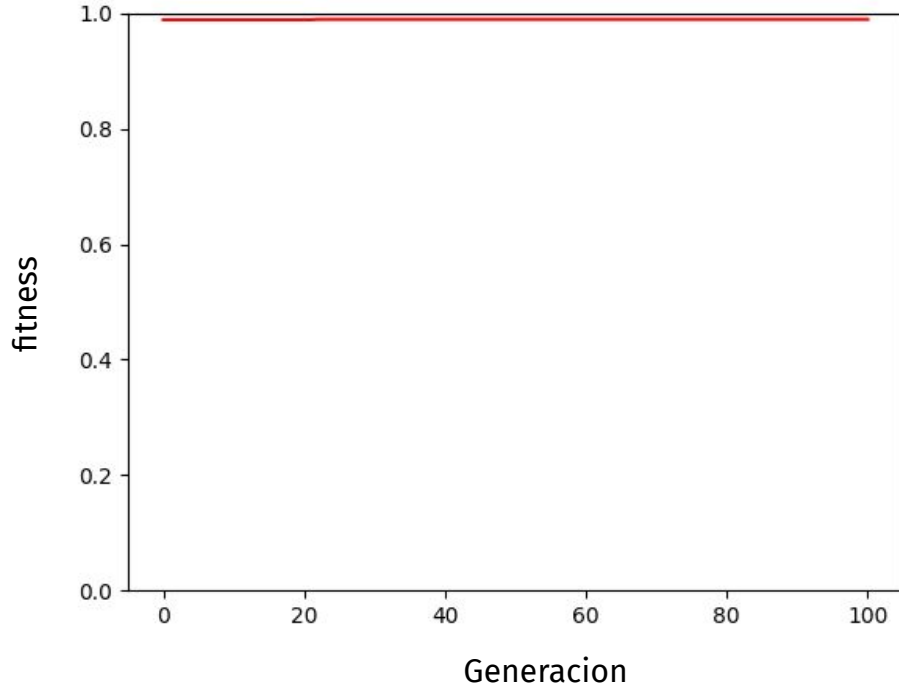
Selección: Ruleta



Según aptitudes relativas



Evolución del fitness del color más aproximado

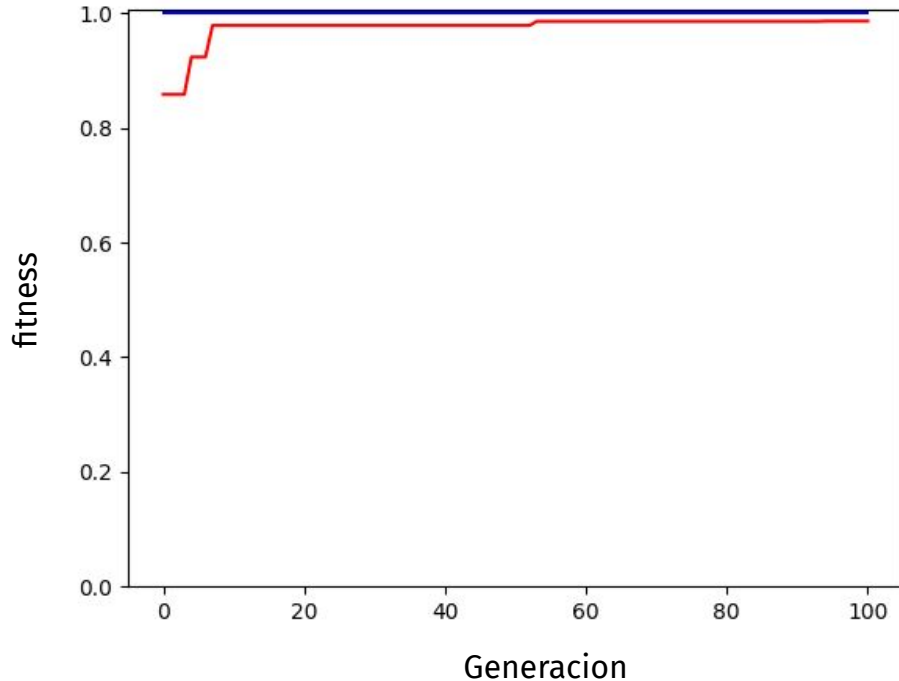


Selección: ruleta

Generaciones: 100

Fitness alcanzado: 0.9896245046624919

Evolución del fitness del color más aproximado

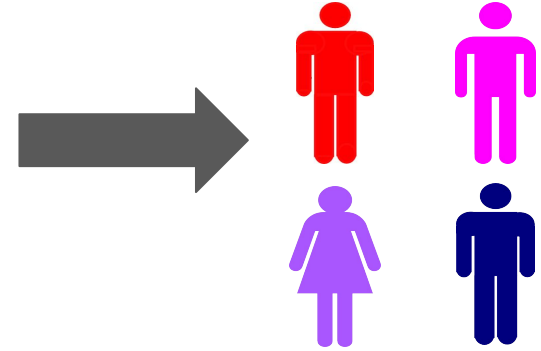
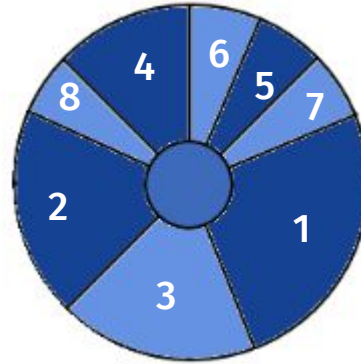
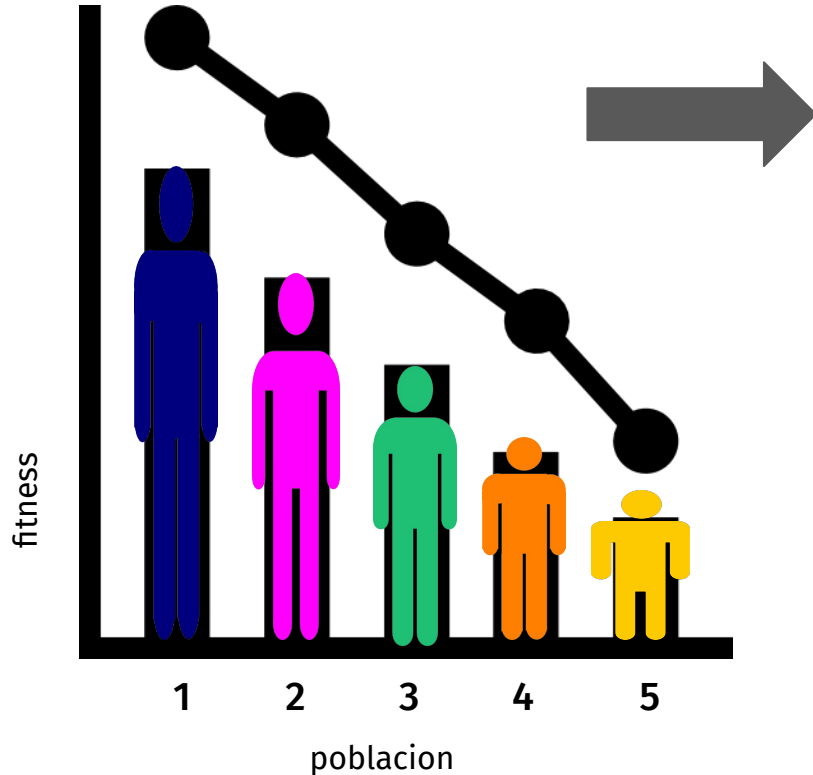


Selección: ruleta

Generaciones: 100

Fitness alcanzado: 0.986043034964665

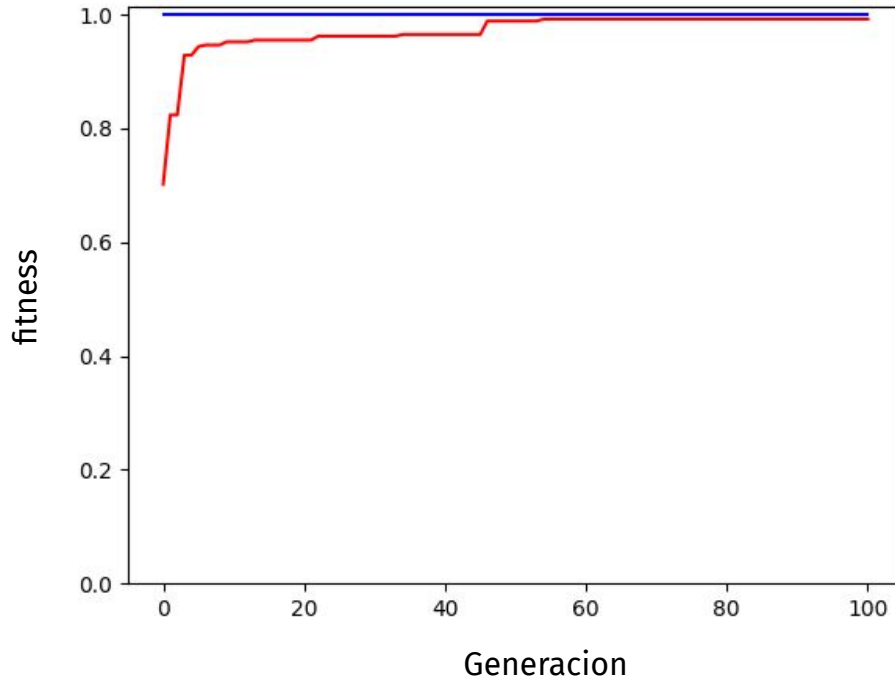
Selección: Ranking



Observación:

Aunque los fitness sean similares entre sí, la probabilidad de que el individuo sea seleccionado está determinada por el ranking de su fitness respecto de sus pares, no del valor de la aptitud en sí.

Evolución del fitness del color más aproximado

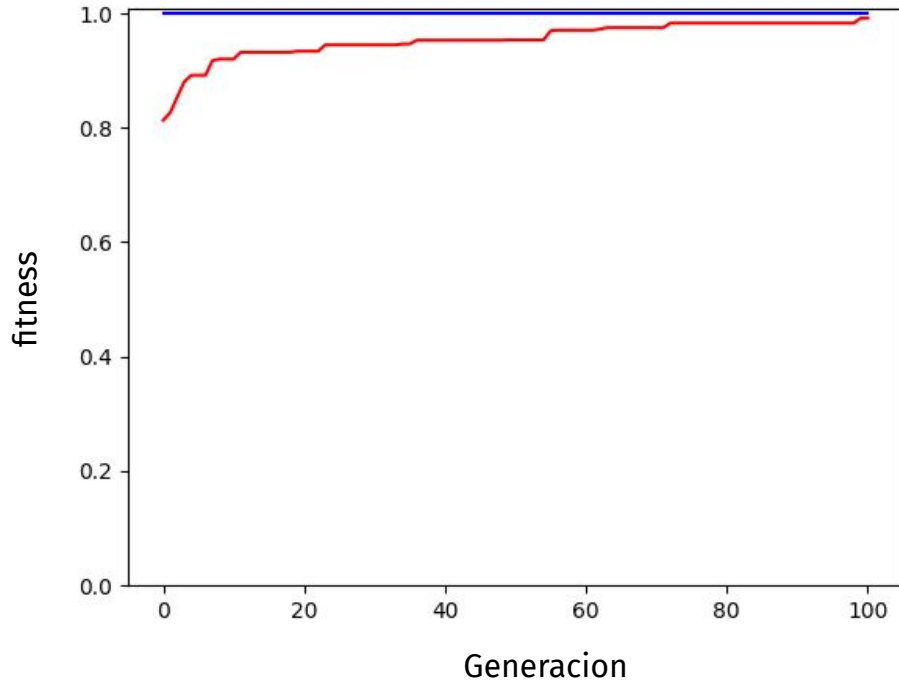


Selección: ranking

Generaciones: 100

Fitness alcanzado: 0.9918366039236622

Evolución del fitness del color más aproximado



Selección: ranking

Generaciones: 100

Fitness alcanzado: 0.9915284435314929

Conclusión

- La evolución de la aptitud del mejor individuo en cada generación es creciente pero no aumenta de manera constante: comienza a mejorar rápidamente pero luego crece de manera más lenta debido a la cercanía a la respuesta y la diversidad de la población .
- Los saltos que se producen en los gráficos se deben principalmente a que haya ocurrido una mutación.
- La selección élite, comparado a las otras, crece más lento porque la cruce se realiza entre dos individuos que tengan aptitudes muy similares. Esto puede cambiar si es que surge una mutación.

Posibles pasos a seguir:

- Agregar otros tipos de métodos de cruce y de selección.