Instituto Politécnico Nacional
Escuela Superior de Cómputo

Análisis de Algoritmos
Profesora: Miriam Pescador Rojas

Proyecto final:
Aprendizaje automático de figuras geométricas mediante algoritmos evolutivos

Integrantes del equipo:
Benítez Morales Manuel Emilio (3CM1)
Hernández López Ángel Zait (3CM4)
Tellez Perez Juan Manuel (3CM1)

Índice

Introducción

Un problema de aprendizaje puede definirse como el problema de mejorar alguna medida de desempeño, al ejecutar alguna tarea, a través de experiencias de entrenamiento. Se dice que un programa de computadora aprende de la experiencia *E*, con respecto a alguna clase de tareas o actividades *T* y con respecto en alguna medida de desempeño *M*, cuando su desempeño en las tareas *T* mejora según la medida *M*, después de experimentar la experiencia *E*.

En resumen, el aprendizaje automático trata de crear programas capaces de generalizar comportamientos a partir de una información no estructurada suministrada en forma de ejemplos. Existen diferentes paradigmas en el aprendizaje automático:

- Aprendizaje estadístico.
- Aprendizaje simbólico.
- Aprendizaje evolutivo.
- Aprendizaje conexionista.
- Aprendizaje analógico.

Marco teórico

El término computación evolutiva o algoritmos evolutivos, engloban una serie de técnicas inspiradas biológicamente en los principios de la teoría Neo-Darwiniana de la evolución natural. En términos generales, para simular el proceso evolutivo en una computadora se requiere:

- Codificar las estructuras que se replicarán.
- Operaciones que afecten a los “individuos”.
- Una función de aptitud.
- Un mecanismo de selección.

Es un tanto complicado distinguir las diferencias entre los distintos tipos de algoritmos evolutivos existentes, pero suele hablarse de tres paradigmas principales:

- Programación evolutiva.
- Estrategias evolutivas
- Algoritmos genéticos.[1]

Definición del problema

La tarea de este proyecto es implementar un algoritmo de aprendizaje automático basado en computación evolutiva para aprender figuras geométricas en específico, los cuales son:

- Cuadrado/rectángulo
- Círculo
- Marco (No desarrollado)

Para poder dibujar cada uno de ellos, se tendrá que dar ciertos valores, pero a grandes rasgos, las figuras se conforman de coordenadas; cada coordenada tiene dos valores flotantes que describen el valor x y y, que representan las horizontales y abscisas en el plano cartesiano.

Para el caso del cuadrado o rectángulo, el usuario tendrá que ingresar dos coordenadas, es decir, dos valores de x y dos valores de y. donde la primer coordenada (x,y), es el vértice superior izquierdo de la figura, y la segunda coordenada, es el vértice inferior derecha (Figura 1a).
Para el círculo, este sólo tendrá una coordenada y un radio, en total el usuario ingresará 3 valores: (x,y,r); donde r es el valor del radio. Por último, para el marco se solicitarán cuatro coordenadas, que podemos verlo como la unión de dos rectángulos, como se muestra en la figura 1c.

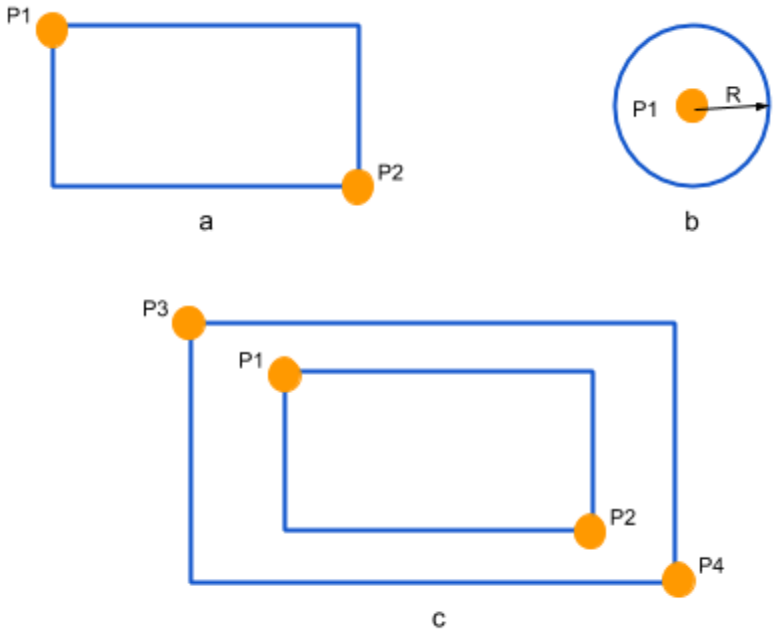


Figura 1. Captura de las figuras

Es decir que el primer punto es la esquina superior izquierda del rectángulo interior, el segundo es la esquina inferior derecha del marco interno; para el rectángulo externo, el vértice superior izquierdo es el tercer punto y el cuarto corresponde al vértice inferior derecho.
Para poder ayudar al algoritmo evolutivo a aprender, el programa también tendrá que generar coordenadas aleatorias para poder verificar qué tan parecido es la población con la figura original dada por el usuario.
Como se sabe, hay muchos métodos y algoritmos evolutivos para poder resolver este tipo de problemas. Para la resolución de este problema en específico, optamos por utilizar optimización por enjambre de partículas, la cual se explica en la siguiente sección.

Particle Swarm Optimization

La optimización por enjambre de partículas o PSO (por sus siglas en inglés), es un método adaptativo que utiliza agentes o partículas que se mueven a través del espacio de búsqueda utilizando los principios de evaluación, comparación e imitación.

Se basa en el uso de un conjunto de partículas o agentes que corresponden a estados de un problema de optimización, donde cada partícula se moverá en el espacio de soluciones en busca de una posición óptima o una buena solución. Los agentes se comunican entre sí, y entonces el agente con una mejor posición (medida de acuerdo a una función objetivo) incluye en los demás atrayendolos hacia él.
La población se inicializa, asignando a las variables valores y una velocidad aleatorios. en cada iteración, la velocidad de cada partícula es aleatoriamente acelerada hacia su mejor posición (donde el valor de la función de aptitud u objetivo es mejor) y a través de las mejores posiciones de sus vecinos. En términos generales, como se mencionó anteriormente, las principales características de este método son:

- Evaluación: La tendencia al estímulo de evaluar, es la principal característica de los organismos vivos. El aprendizaje no ocurre a menos que el organismo pueda evaluar, pueda distinguir características del medio ambiente que atraen o características que repelen. Desde este punto de vista, el aprendizaje puede definirse como un cambio que posibilita al organismo mejorar la evaluación promedio de su medio ambiente.
- Comparación: Los estándares del comportamiento social se realizan mediante la comparación con otros.
- Imitación: Lorenz asegura que sólo los seres humanos y algunas aves son capaces de imitar. La imitación es central para la adquisición y mantenimiento de las habilidades mentales.

Para resolver problemas, se propone utilizar PSO con un manejo dinámico de las partículas, lo que permite romper ciclos y diversificar la búsqueda. Para resolver este problema, se considerará que un enjambre de n partículas-solución está dada de la forma:

$$\beta = (\beta_1, \beta_2, ..., \beta_n)$$

con $\beta_j \, \varepsilon \, \Omega, \; j = 1, 2, .., n$, entonces definimos un movimiento del enjambre de la forma:

$$\beta_{j,t+1} = \beta_{j,t} + V_{j,t+1} \qquad (1)$$

para

$$V_{j,t+1} = \alpha V_{j,t} + rand(0, \varphi_1)[\beta'(j,t) - \beta(j,t)] + rand(0, \varphi_2)[\beta'(g,t) - \beta(j,t)] \qquad (2)$$

donde:

- Ω : Espacio factible de soluciones.
- $V_{j,t}$: Velocidad en el tiempo t de la j-ésima partícula.
- $V_{j,t+1}$: Velocidad en el tiempo t+1 de la j-ésima partícula.
- $\beta(j,t)$: Partícula j-ésima en el tiempo t.
- $\beta'(g,t)$: Partícula con el mejor valor de todas en el tiempo t.
- $\beta'(j,t)$: Partícula j-ésima con el mejor valor hasta el tiempo t.
- $rand(0, \varphi)$: Valor aleatorio de distribución uniforme en el intervalo $[0, \varphi]$.

- α : Parámetro de escala.[2]

Existen varias definiciones sobre el cálculo de la velocidad de la partícula, pero la mayoría es similar al que se redacta en esta sección (Ecuación 2). Otros autores no tienen el parámetro de escala en la velocidad, o bien, no se suma la velocidad. Pero se optó por tomar la siguiente ecuación para el cálculo de la velocidad:

$$V_{j,t+1} = V_{j,t} + rand(0,1)\varphi_1[\beta'(j,t) - \beta(j,t)] + rand(0,1)\varphi_2[\beta'(g,t) - \beta(j,t)] \quad (3)$$

Como observamos, en la ecuación 3, se toma un número aleatorio entre 0 y 1, multiplicado por un φ que lo conocemos como ratio de aprendizaje, que también tiene un valor entre 0 y 1.

La optimización por enjambre de partículas es una parte de lo que se conoce como inteligencia de enjambre, tiene sus raíces en la vida artificial, psicología social, ingeniería y ciencia de la computación. PSO difiere de la computación evolutiva en que los miembros de la población llamados partículas o agentes, están volando a través del hiperespacio del problema.[2] En el algoritmo 1, se muestra el pseudocódigo del algoritmo de PSO con la modificación de la ecuación de velocidad.

Algorithm 1: Algoritmo de PSO

```
Data: β: Población de partículas distribuidas en el espacio factible.
Data: generaciones: Número de generaciones
begin
    n ← length(β)
    i ← 0
    while i < generaciones do
        Evaluar cada posición de la partícula de acuerdo a una función
        objetivo (función de aptitud).
        if La posición actual de una partícula es mejor que las previas then
            Actualice
        Determinar la mejor partícula (de acuerdo a las mejores posiciones
        previas).
        j ← 1
        for j < n do
            V_{j,t+1} ← V_{j,t} + rand(0,1)φ1[β'(j,t) − β(j,t)] +
            rand(0,1)φ2[β'(g,t) − β(j,t)]
            β_{j,t+1} ← β_{j,t} + V_{j,t+1}
end
```

Como podemos notar, la función objetivo no está definida, ya que se pone dependiendo el problema que se quiere resolver. Nuestro problema (descrita en la sección “Definición del problema”), es tener el mayor número de número positivos y negativos similares a la figura dada por el usuario. Para tener mayor certeza en la figura, agregaremos la diferencia de las

áreas sobre la figura del usuario menos el área de la figura creada por la partícula, por lo que podemos decir que la función de aptitud es de la forma:

$$f(x) = 0.9 * (\Sigma p + \Sigma q) - 0.1 * (|AreaReal - AreaParticula|) \quad (4)$$

Donde p son los puntos positivos que coincidieron, q el número de puntos negativos que coincidieron. Las constantes en la ecuación 0.9 y 0.1 es una pequeña ponderación para poder tener mayor importancia a la coincidencia de puntos.

Síntesis de la lectura *Learning Geometric Concepts with an Evolutionary Algorithm* [3]

Los algoritmos evolutivos son la forma más general de búsqueda heurística. Estos algoritmos operan en un espacio bien determinado y se lleva a cabo un proceso de selección basado en reglas previamente establecidas, realizando un cierto número de iteraciones, es decir, las generaciones de la evolución, teniendo una población generada en cada una, donde la primera el aleatoria y el resto se da a partir de su anterior.
Existe un parámetro de aptitud que determina si un individuo o elemento de cada generación es apto para transformarse y continuar a la siguiente generación, para lo cual se usa comúnmente una reproducción -pasa a la siguiente generación tal cual-, una mutación -pasa a la siguiente generación en una forma aleatoria diferente- o una cruza -se pasan subpartes de la mezcla de 2 individuos-.

Teniendo esto en cuenta, surgen varios métodos de implementar algoritmos evolutivos:

- Algoritmos genéticos
- Programación evolutiva
- Estrategias evolutivas
- Programación genética

En la experiencia del brazo robótico que reconoce figuras geométricas se utilizó un nuevo algoritmo, inventando operadores que atacan problemas más específicos que la cruza y se usó el operador principal de programación genética.

La tarea de aprender figuras se describe de la siguiente manera:
Dado un arreglo de pixeles a y la salida actual de la cámara, encontrar un programa corto p que describa a. Se llama a_p al arreglo de pixeles que resultan de la visión con la cámara.

Para resolver esto se emplea el algoritmo evolutivo, donde el factor principal es determinar la aptitud de p, un modo útil de medir esto es tomar la idea de que un pixel de a es atraído por otro pixel del mismo color en otra figura a' pero calcular esto es tardado, entonces se desarrolla

una función de distancia de figura, como una medida de la diferencia entre 2 arreglos de pixeles a y a'.

$$D(a,\ a') = \sum_{c \in C} d(a,\ a',\ c) + d(a',\ a,\ c) \qquad (5)$$

$$d(a,\ a',\ c) = \frac{\sum_{a[p1]=c} min\{md(p1,\ p2) \mid a'[p2]=c\}}{\#c(a)} \qquad (6)$$

donde

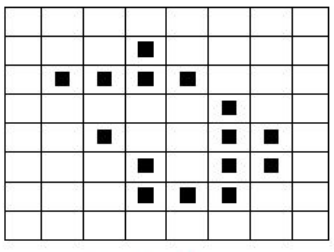
- C denota un conjunto de colores de tamaño constante.
- $a[p]$ denota el color c del arreglo de pixeles a en la posición $p = (x,\ y)$.
- $md(p1,\ p2) = |x1 - x2| + |y1 - y2|$ es la distancia de Manhattan (distancia entre dos puntos medida por sus ejes y ángulos rectos) entre $p1$ y $p2$.
- $\#c(a) = \#\{p1 \mid a[p1]\}$ es el número de pixeles en a de color c.

Esto quiere decir que la función de distancia de la figura es una suma sobre los colores del promedio de la distancia de los pixeles color c de a al pixel más cercano de ese mismo color en a', es decir $d(a,\ a',\ c)$, mientras que $d(a',\ a,\ c)$ es el contrario, si esto fuese ideal, la suma sería el doble de la distancia, puesto que es simétrica, sin embargo, ambas pueden no ser la misma distancia.

La parte menos observable de la implementación de forma lineal de la función de distancia es el cálculo del numerador de (5) , entonces se recurre a un mapa de distancia para el color c, este mapa es un arreglo de distancias de Manhattan al punto más cercano que tiene ese color en el arreglo a' para todas las posiciones, esto se expresa como el numerador:

$$d - map_c = min\{md(p1,\ p2) \mid a'[p2] = c\}$$

Este mapa está representado por la figura 2:



■ = pixel with color c

3	2	2	1	2	3	4	5
2	1	1	0	1	2	3	4
1	0	0	0	0	1	2	3
2	1	1	1	1	0	1	2
2	1	0	1	1	0	0	1
3	2	1	0	1	0	0	1
3	2	1	0	0	0	1	2
4	3	2	1	1	1	2	3

d-map$_c$

Figura 2. Mapeo

Donde cada pixel de color c está representado por un 0, ya que la distancia de él a él mismo es 0, mientras que las demás casillas representan la distancia mínima que tienen con pixeles de otro color.

La herramienta utilizada para construir los programas para la experiencia fue el *lenguaje de gráficas de tortuga* L_T -realización de gráficos vectoriales usando un cursor (la tortuga) relativo a unas coordenadas cartesianas- siguiendo los siguientes comandos:

- Mover x y c: Mover el cursor al punto (x, y) y asignarle el color c.
- Dibujar $\alpha \mid c$: Dibujar una línea de color c desde la posición actual del cursor con ángulo α y longitud l y mover el cursor al final de esa línea.
- Línea $\alpha \mid c$: PArecido al comando anterior, sin embargo, la posición del cursor permanece sin alteraciones.
- Para $i = k_1$ hasta k_2 cada s {Bloque de comandos}: Se usa como un *ciclo for*; la variable del ciclo puede ser utilizada sólo por el bloque de comandos.
- Final: Terminar el programa.

Para la selección del operador mencionado al principio de la descripción e la experiencia se utiliza un método de *selección por torneo,* es decir, se toman dos elementos de una población de manera aleatoria y se elige aquel que tiene la mejor aptitud para hacer una transformación y se repite mientras los operadores necesitan elementos o individuos para generar población.

Para esta transformación se usó reproducción con los operadores:

- Escala en constantes ($L_T \rightarrow L_T$)
- Conc ($L_T \times L_T \rightarrow L_T$)
- Split ($L_T \rightarrow L_T \times L_T$)

Para la obtención de resultados, se configuró el sistema con una población de 50, siendo la primera generada aleatoriamente y las siguientes usando el operador de selección que se definió. En las pruebas más satisfactorias se eligieron los operadores de transformación con los valores: Escala de constantes= 0.6, Conc= 0.2, Split=0.1 con una reproducción de 0.1.
Utilizando imágenes de 100x100 pixeles, un cuadrado fue encontrado en un promedio de 7 horas y un triángulo en 23 horas, ambos con una orientación arbitraria, y algo interesante es que hasta que una figura en particular es encontrada, cualquier figura geométrica de este tipo se reconoce rápidamente.

Otro resultado interesante es que el algoritmo es capaz de reconocer grandes paisajes compuestas de objetos conocidos, como objetos para los que la población tiene un programa. (un escenario consiste en un triángulo o dos cuadrados reconocidos en 14 minutos promedio en caso de que estos sean conocidos), ambos hechos son muy útiles para la tarea de aprender clasificaciones de escenarios.

Una meta para el futuro es buscar el desarrollo de operadores jerárquicos, además del análisis de otras aplicaciones con los operadores actuales. El algoritmo evolutivo describe esta escena por medio de un programa que dibuja primero un rectángulo y después un triángulo sobre una parte de él, de esta forma se podría usar la información para identificar bloques de construcción para el aprendizaje automático de la imagen.

Pruebas

Ahora ya que tenemos el programa listo para correr el algoritmo y ver los resultados que nos arroja, realizaremos unas pruebas con diferentes parámetros y ver cómo afecta cada uno de estos mismos en la solución del problema.

Tenemos distintos parámetros donde podremos cambiar la cantidad de puntos aleatorios, podemos configurar la cantidad de población que existirá en el algoritmo, luego también se puede configurar el número de generaciones que pasarán para que la población pueda aprender la figura que nosotros asignamos y al final tenemos otros dos parámetros que son los ratios de aprendizaje donde estos influyen mucho con el algoritmo PSO que nosotros elegimos ya que son esenciales para el cálculo de la velocidad que se le otorgará a cada figura.

A continuación, mostraremos algunas pruebas con las figuras Círculo y Rectángulo para ver cómo se comporta el algoritmo.

Primero empezaremos con pruebas con una población de 50 para la figura del rectángulo, a esta prueba la llamaremos “Prueba 1”.

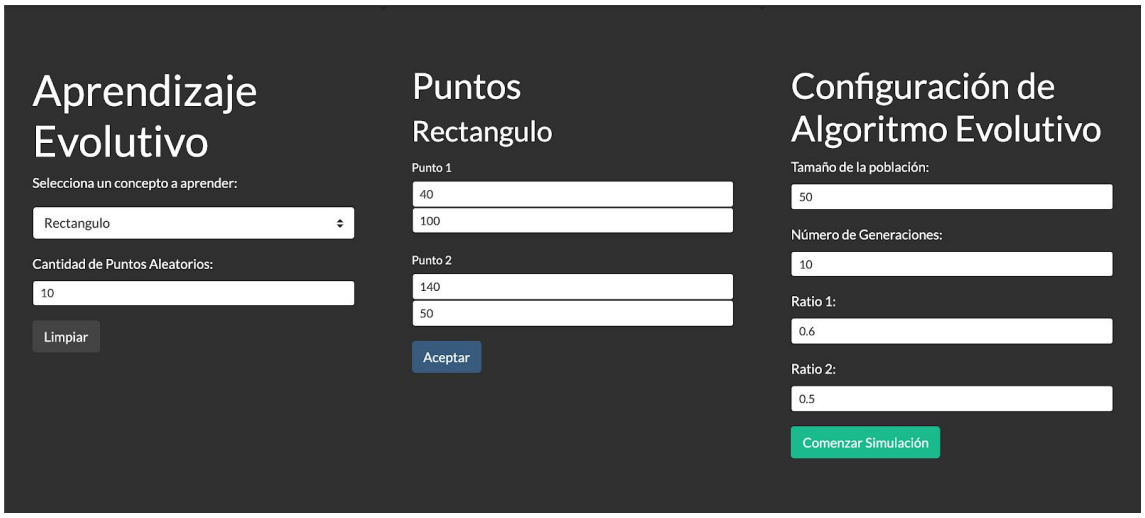


Figura 3: Parámetros de la prueba 1

En la figura 3, podemos ver que ocupamos los siguientes parámetros:

1. Cantidad de Puntos Aleatorios = 10

2. Tamaño de Población = 50
3. Número de Generaciones = 10
4. Ratio 1 = 0.6 y Ratio 2 = 0.5

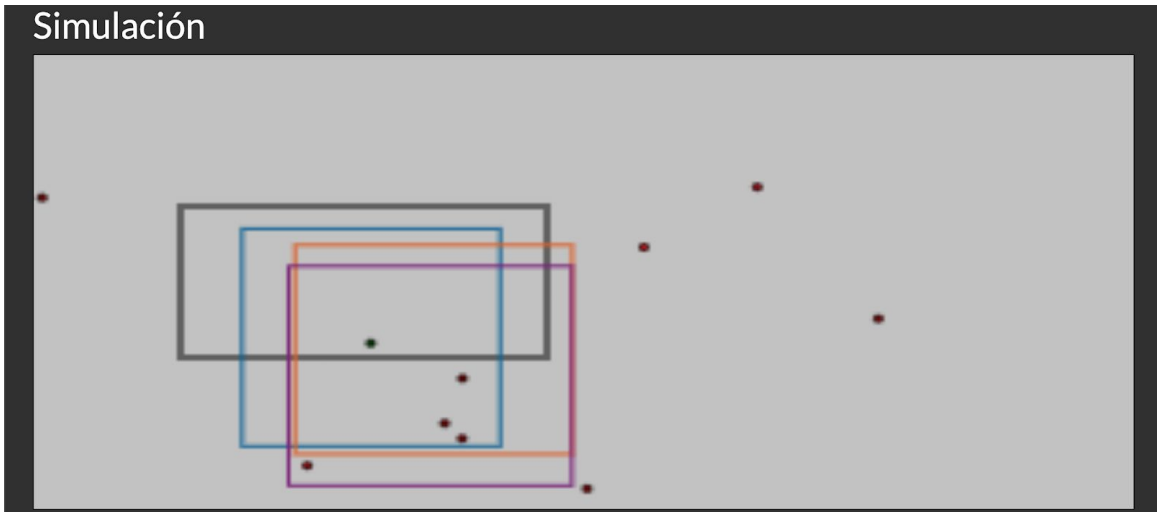


Figura 4. Simulación de la prueba 1

Cómo podemos ver en la figura 4, los resultados fueron bastante positivos, ya que alcanzó un buen aprendizaje, se pueden visualizar las tres mejores soluciones con color azul la mejor solución. Realmente aquí podemos ver que las figuras no alcanzaron a empatar con el área que nosotros asignamos y en la figura 5, apreciamos las aptitudes de cada solución.

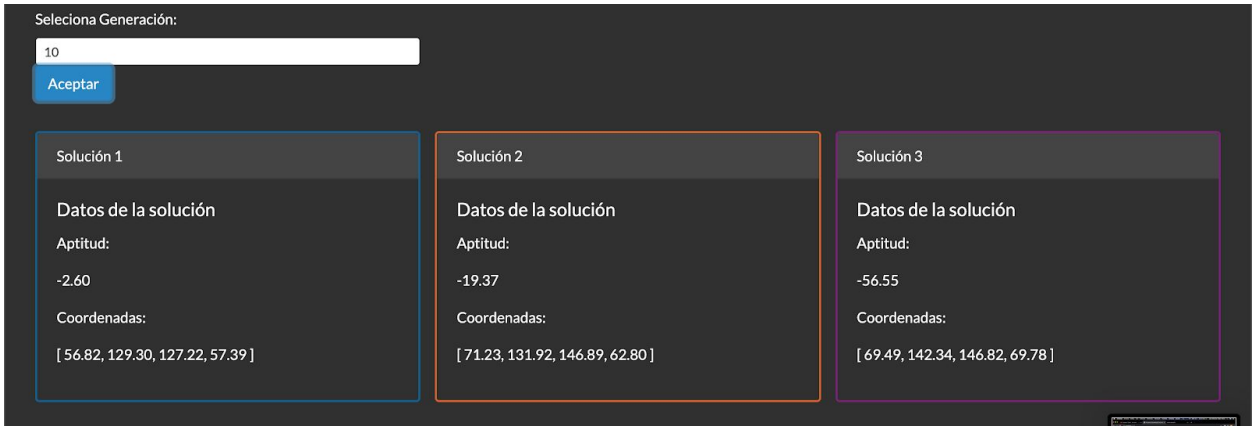


Figura 5. Aptitudes de la prueba 1

Seguiremos cambiando los parámetros y podremos ver una mejor solución. Ahora, probaremos con una población de 100 partículas y 100 puntos aleatorios (Figura 6).

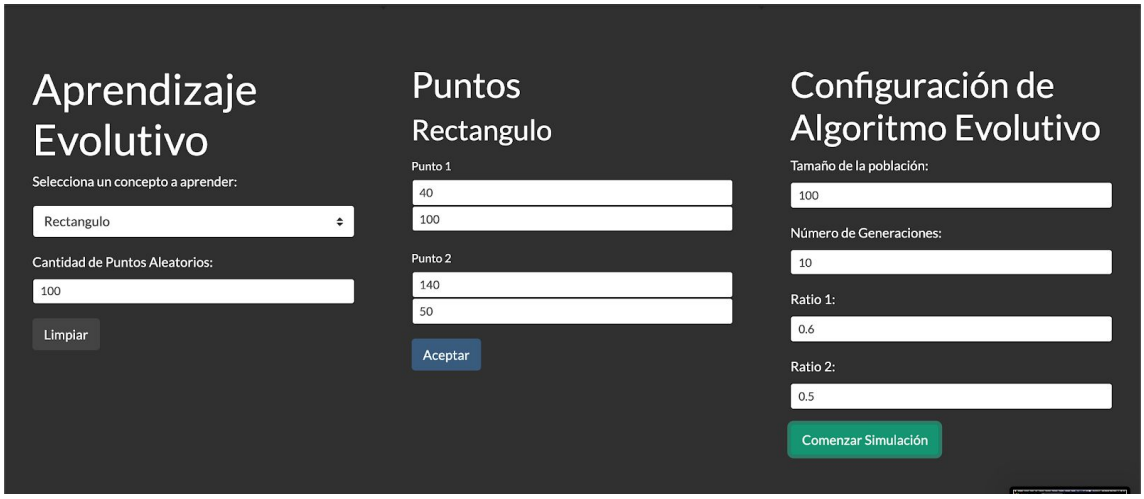


Figura 6. Parámetros de la prueba 2

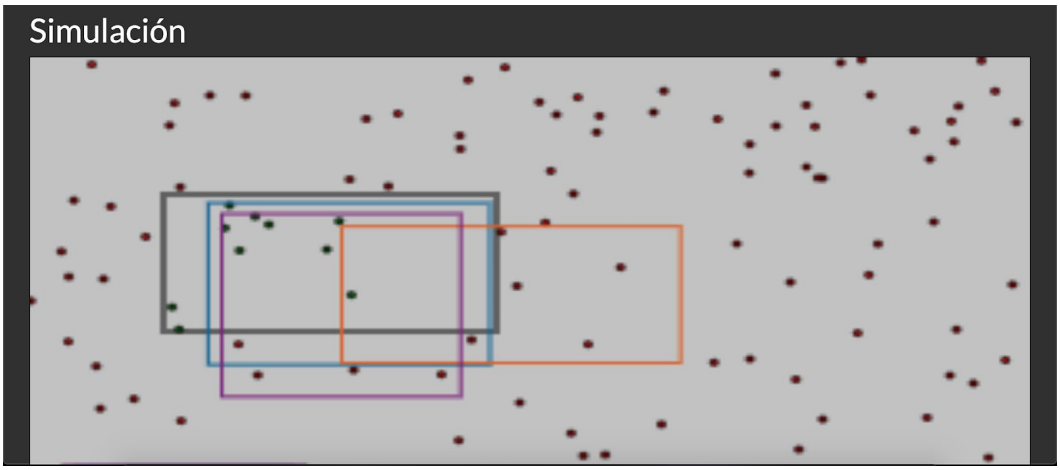


Figura 7. Simulación de la prueba 2

Para la figura 7, observamos que hay una gran diferencia con la Prueba 1, donde se percibe que la mejor solución, que es la de color azul, logra acercarse más a la figura que nosotros asignamos, y de igual forma se aprecian los valores resultantes en la figura 8.

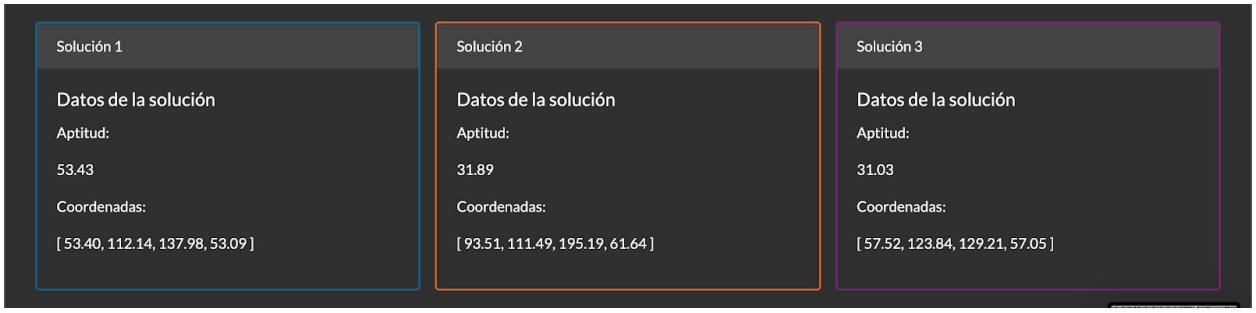


Figura 8. Aptitudes de la prueba 2.

Ahora, para la Prueba 3, usaremos una población de 200 partículas para ver si logramos obtener un mejor resultado.

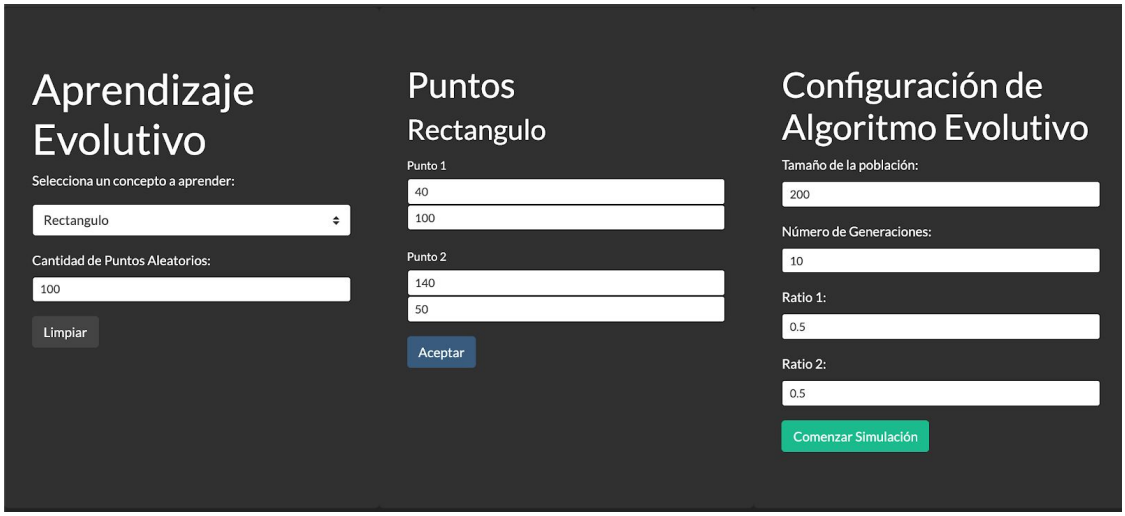


Figura 9. Parámetros de la prueba 3

No solamente se cambió el valor del campo de la población, además cambiamos el ratio 1 a 0.5, como se muestra en la figura 9.

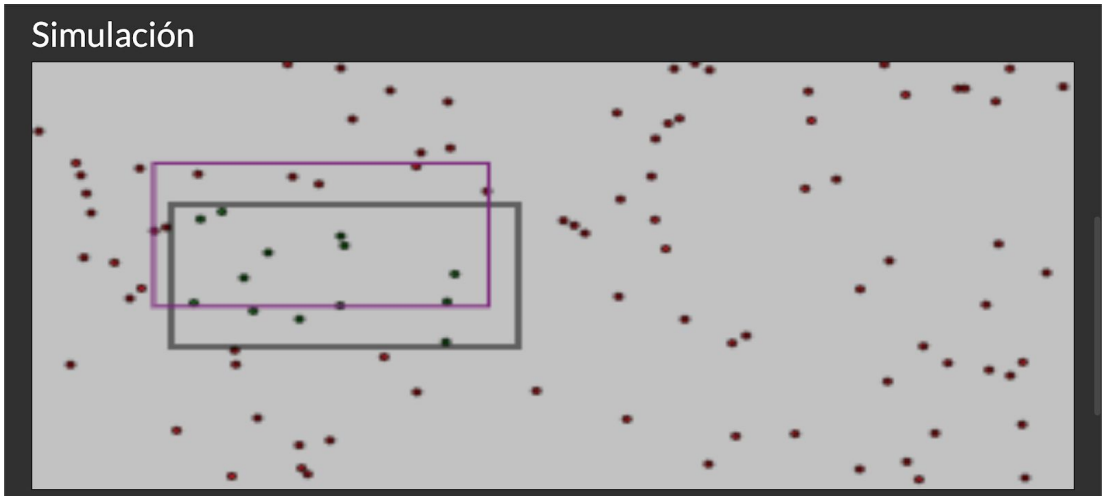


Figura 10. Simulación de la prueba 3

En la prueba 3 hay un punto a observar, ya que podemos ver en la figura 10, sólo aparece una solución en la simulación.

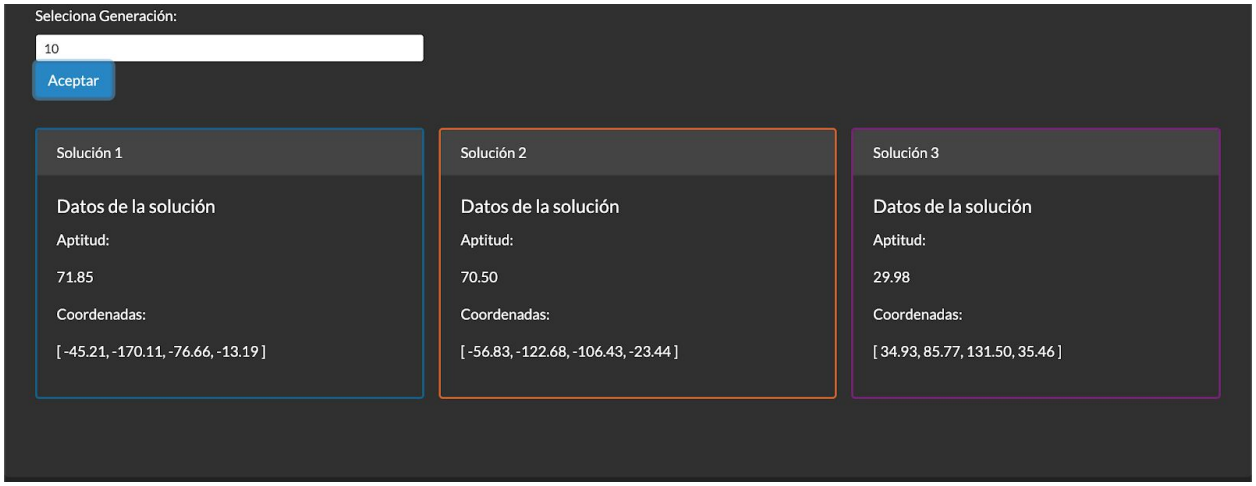


Figura 11. Aptitudes de la prueba 3

En los datos de cada solución (Figura 11), vemos que las primeras dos mejores soluciones tienen coordenadas negativas y la tercera solución es la que sí podemos observar. Este es un detalle que ocurre porque el algoritmo empieza a sobre aprender, tiende a aprender más área de la figura base y se enfoca mucho en tener acertados todos los puntos negativos y para hacer eso se traslada a coordenadas negativas donde, en efecto, acierta a todos los puntos negativos que son más a comparación de la cantidad de puntos positivos.

Lograremos ver esto de nuevo cuando cambiemos la población a una cantidad de 500 partículas (Figura 12).

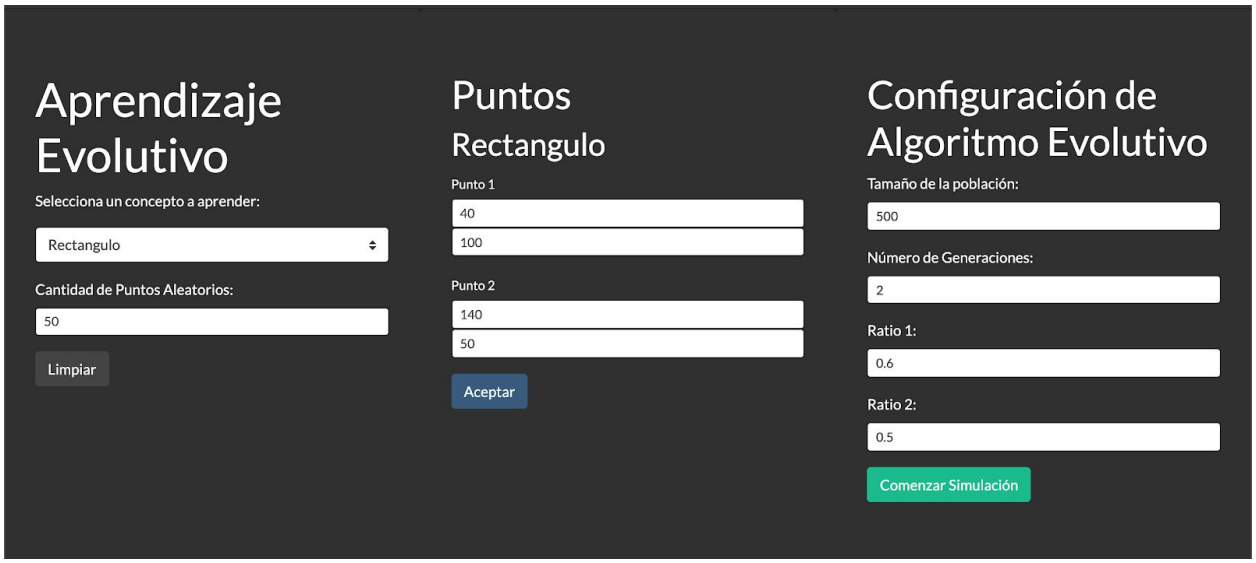


Figura 12. Parámetros de la prueba 4

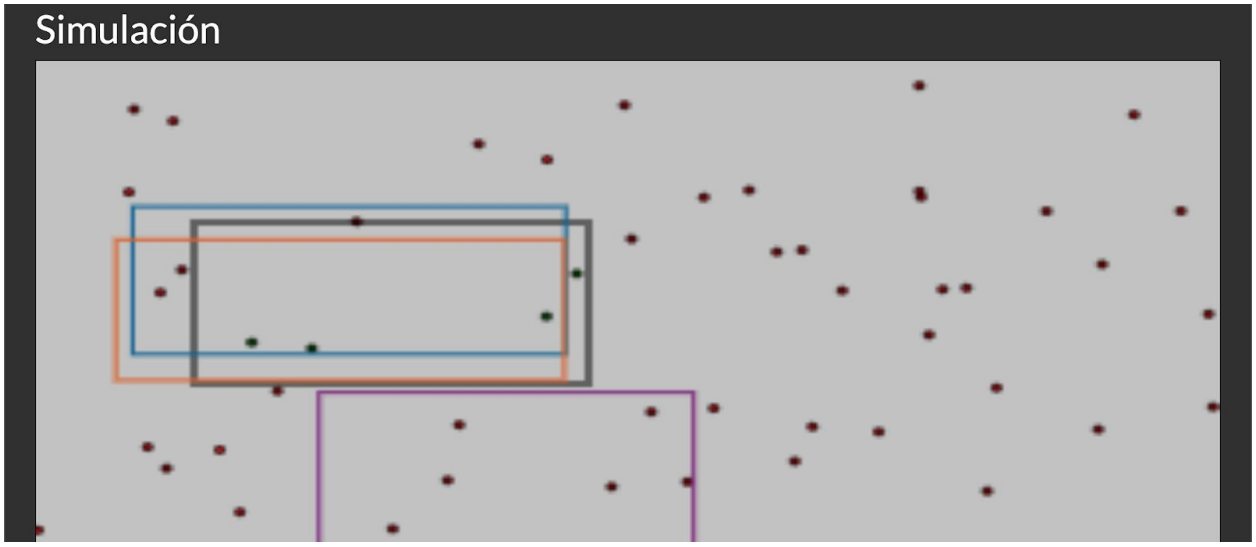


Figura 13. Simulación de la prueba 4

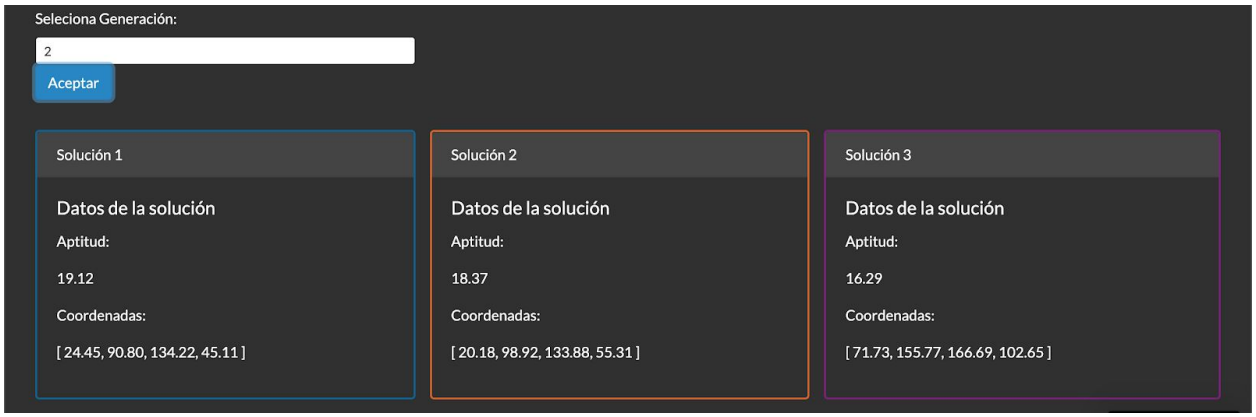


Figura 14. Aptitudes de la prueba 4

Para esta prueba se tuvo que modificar bastante la cantidad de puntos aleatorios y el número de generaciones ya que como lo explicamos en la prueba 3, las soluciones se disparaban hacia las coordenadas negativas donde se concentraban en obtener acertados todos los puntos aleatorios y también en igualar el área que tiene la figura base que nosotros asignamos.
Pero con los parámetros utilizados y la cantidad de población que hay, se alcanzó un gran resultado.

Ahora veremos pruebas sobre la figura Círculo y podremos ver en las figuras 15 a la 26 los parámetros usados para obtener buenos resultados con la distinta cantidad de población.

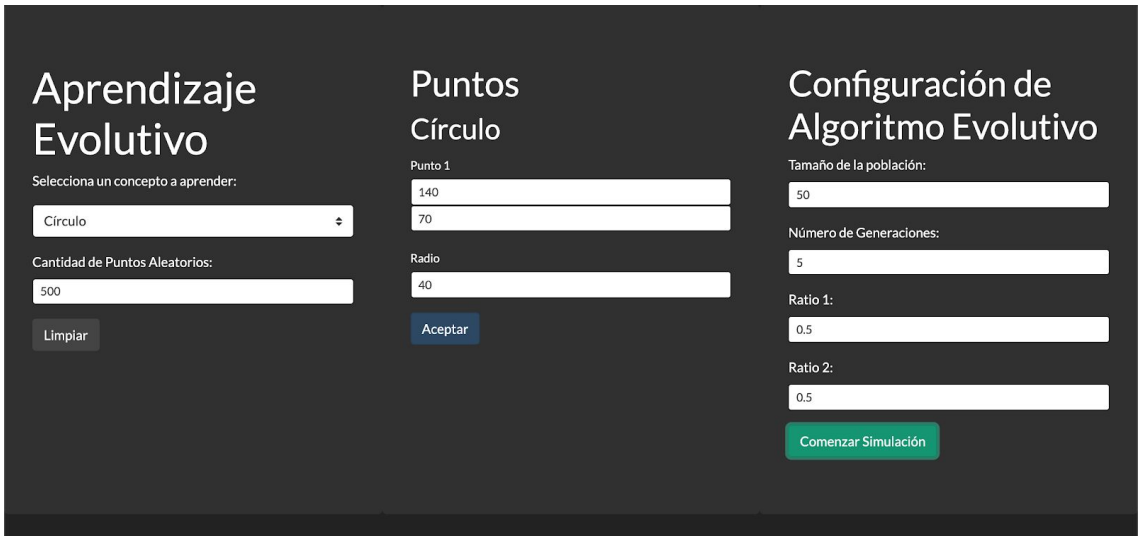


Figura 15. Parámetros de la prueba 5

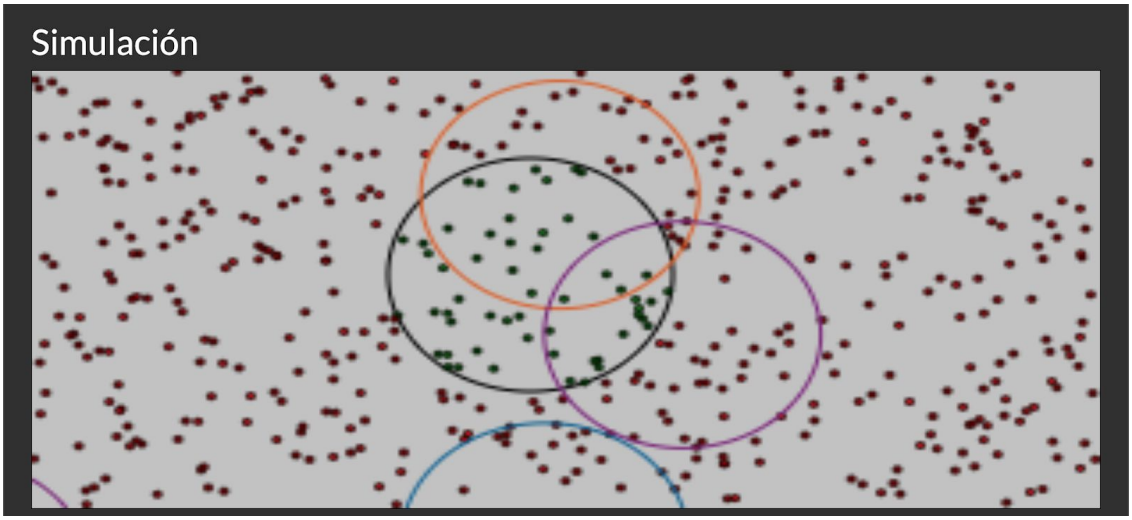


Figura 16. Simulación de la prueba 5

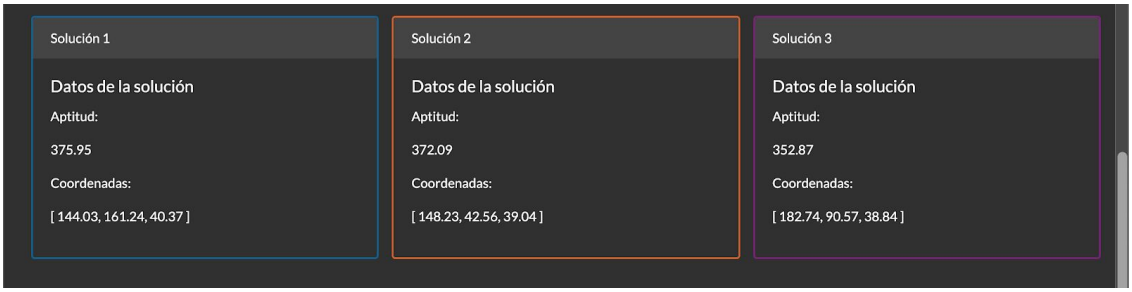


Figura 17. Aptitudes de la prueba 5

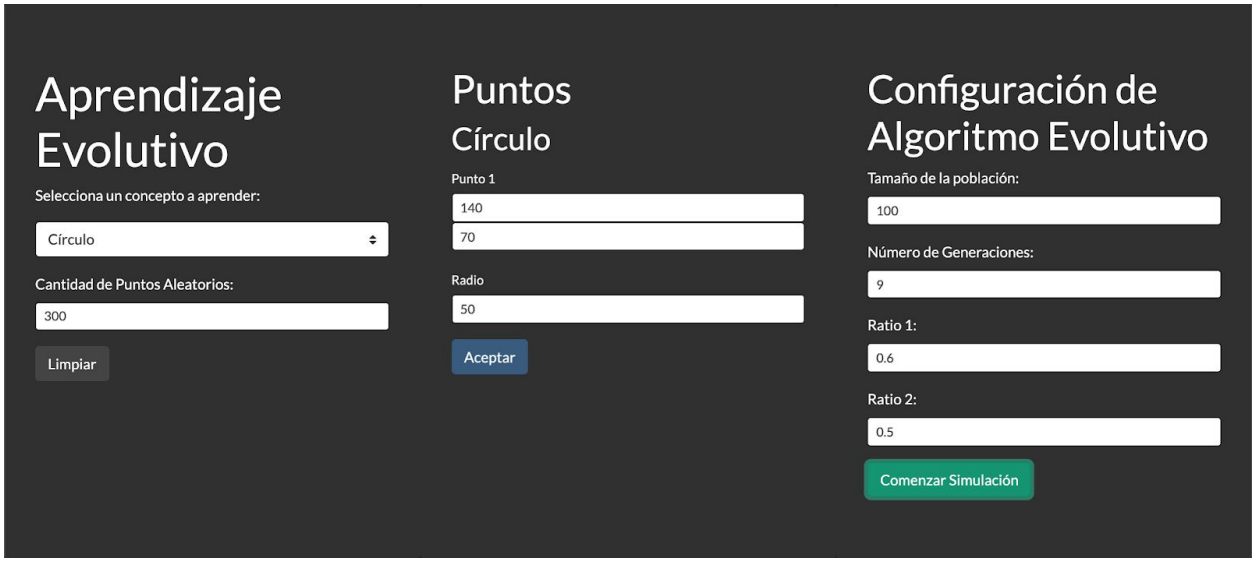


Figura 18. Parámetros de la prueba 6

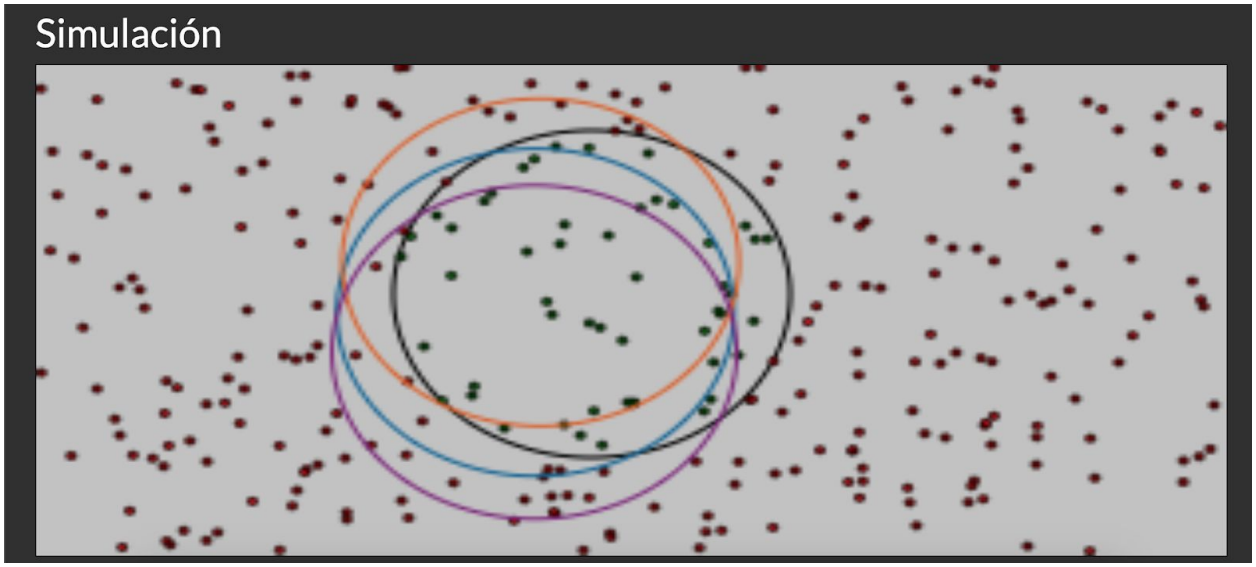


Figura 19. Simulación de la prueba 6

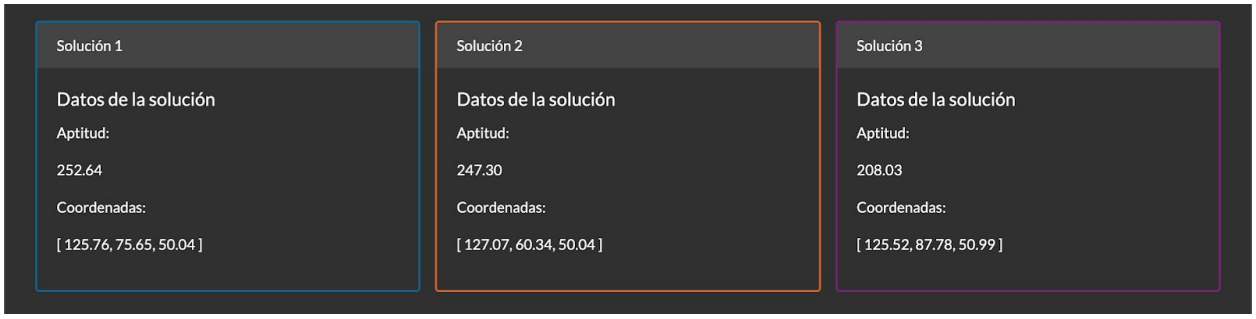


Figura 20. Aptitudes de la prueba 6

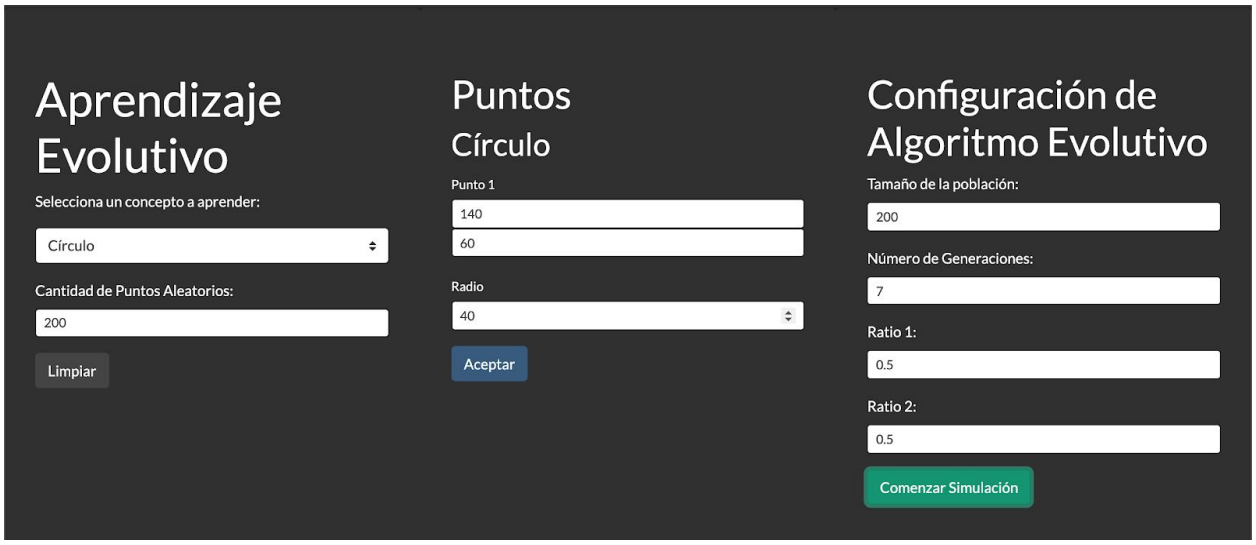


Figura 21. Parámetros de la prueba 7

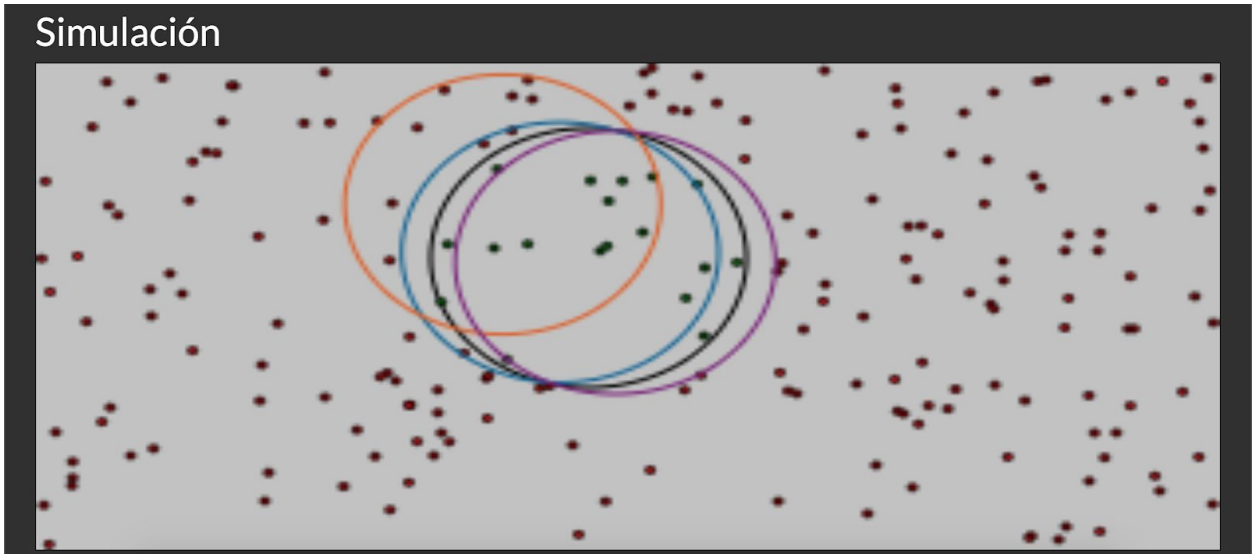


Figura 22. Simulación de la prueba 7

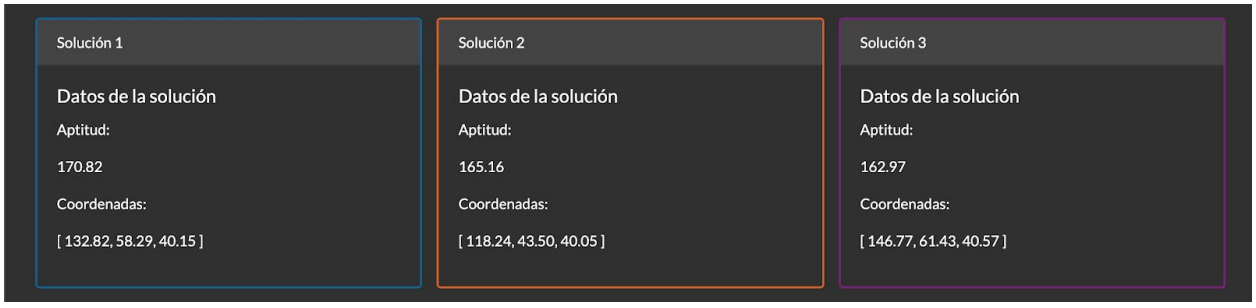


Figura 23. Aptitudes de la prueba 7

Figura 24. Parámetros de la prueba 8

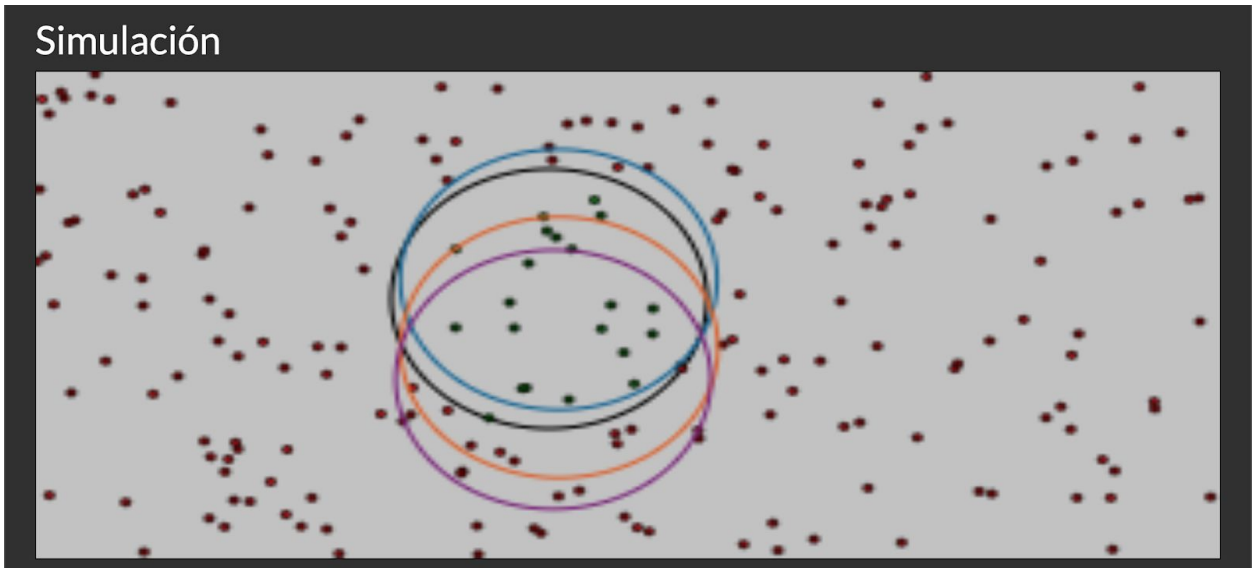


Figura 25. Simulación de la prueba 8

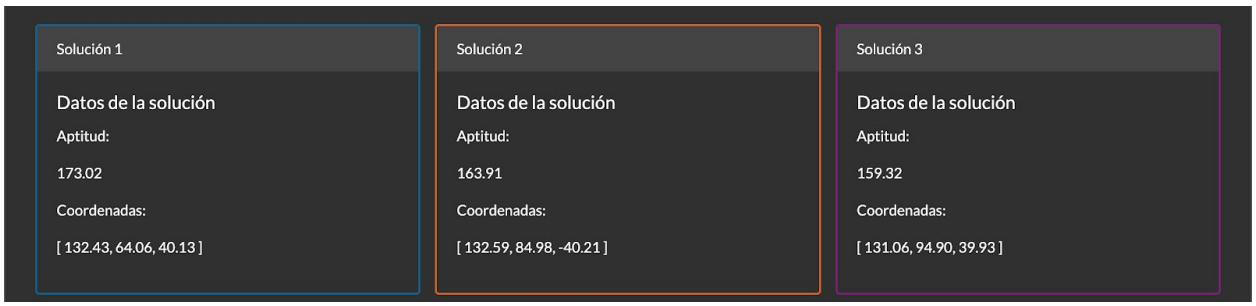


Figura 26. Aptitudes de la prueba 8

Como se pudo apreciar de las pruebas 5 a la 8, para aprender la figura del círculo, se necesitan cantidades muy grandes de puntos aleatorios, una población grande y pocas generaciones

Realizaremos unas pruebas para poder llenar la siguiente tabla y sacar conclusiones sobre cómo se comporta el algoritmos con este problema y que parámetros son los óptimos para obtener buenos resultados y se realice el aprendizaje de forma efectiva.

Sobreaprendizaje: si se aumenta la población, tiende a irse a los negativos porque encuentra el área más adecuada antes que las coordenadas, por lo que toma dichas coordenadas, ya que se asemeja más al concepto a aprender, a pesar de que no está en la posición correcta y entre más población hay, más tiende a alejarse.

Tabla comparativa

Concepto	Número de muestras	Aptitud promedio	Aptitud mínima	Aptitud máxima	Desviación estándar
Rectángulo	50	36.14	9.34	55.77	12.19
	100	78.21	46.34	105.49	15.68
	200	90.16	73.09	105.88	23.72
	500	88.41	80.29	102.33	6.65
Círculo	50	153.96	135.83	167.58	10.13
	100	149.03	119.90	168.28	12.49
	200	166.97	152.73	176.66	6.98
	500	170.62	152.87	172.69	9.55

Para ver el proyecto en funcionamiento puede visitar:
https://zait06.github.io/AnalisisDeAlgoritmos-Proyecto/

Herramientas utilizadas

Para el desarrollo de este proyecto, se decidió usar tecnologías web, ya que garantizamos la multiplataforma al poder ser visto desde un navegador web. A continuación, se mencionan los lenguajes de programación y extensiones que se utilizaron para el desarrollo del proyecto.

Lenguaje de programación

Javascript (JS): Es un lenguaje de programación ligero, interpretado, o compilado justo-a-tiempo (just-in-time) con funciones de primera clase. Si bien es más conocido como un lenguaje de scripting (secuencias de comandos) para páginas web, es usado en muchos entornos fuera del navegador, tal como Node.js, Apache CouchDB and Adobe Acrobat.

JavaScript es un lenguaje de programación basado en prototipos, multi-paradigma, de un solo hilo, dinámico, con soporte para programación orientada a objetos, imperativa y declarativa (por ejemplo programación funcional). Lea más en acerca de JavaScript.
El estándar para JavaScript es ECMAScript. A partir del 2012, todos los navegadores modernos soportan completamente ECMAScript 5.1. Los navegadores viejos soportan al menos ECMAScript 3. Desde Junio 17, 2015, ECMA International publicó la sexta versión principal de ECMAScript, que oficialmente se llama ECMAScript 2015, y que inicialmente se denominó ECMAScript 6 o ES6. Desde entonces, los estándares ECMAScript están en ciclos de lanzamiento anuales. Esta documentación hace referencia a la última versión preliminar, que actualmente es ECMAScript 2020.[4]

Librerías de estilo

Sass: Sass es un lenguaje de hoja de estilo que se compila en CSS. Le permite usar variables, reglas anidadas, mixins, funciones y más, todo con una sintaxis totalmente compatible con CSS. Sass ayuda a mantener bien organizadas las hojas de estilo grandes y facilita compartir diseños dentro y entre proyectos.[5]
Bootstrap: Bootstrap es una biblioteca multiplataforma o conjunto de herramientas de código abierto para diseño de sitios y aplicaciones web. Contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basado en HTML y CSS, así como extensiones de JavaScript adicionales. A diferencia de muchos frameworks web, solo se ocupa del desarrollo front-end.[6]

Módulos NPM

Webpack: Es un paquete de módulos estáticos para aplicaciones JavaScript modernas. Cuando webpack procesa la aplicación, internamente crea un gráfico de dependencia que asigna cada módulo que el proyecto necesita y genera uno o más paquetes.[7]

Numjs: Es un paquete npm / bower para computación científica con JavaScript. Contiene entre otras cosas:

- Objeto de matriz N-dimensional
- Función de álgebra lineal
- Transformada rápida de Fourier
- Herramientas para el procesamiento básico de imágenes

Se puede utilizar como un contenedor eficiente multidimensional de datos genéricos. NumJs tiene licencia bajo la MIT license, lo que permite su reutilización casi sin restricciones.[8]
Algunos otros módulos que se utilizaron para el correcto funcionamiento fueron los siguientes:

- npm i html-webpack-plugin
- npm i css-loader style-loader
- npm i node-sass sass-loader
- npm i mini-css-extract-plugin
- npm i autoprefixer
- npm i file-loader
- npm i postcss-loader
- npm i rimraf

Conclusiones

Benítez Morales Manuel Emilio:

Los algoritmos evolutivos son una herramienta muy útil en la programación, sobre todo al enfocarlo a la tan presente en la actualidad, inteligencia artificial, por lo cual es de gran interés comenzar con pequeñas cosas como el aprendizaje de cosas simples como figuras geométricas.

En este proyecto aprendimos a utilizar este concepto para el desarrollo de tal aplicación, retomando la técnica PSO por ser la mejor opción para un buen desempeño del aprendizaje del software, al menos en este caso, notando también la importancia de la aleatoriedad en el campo de la computación. Es importante mencionar que el tema de sobreaprendizaje representó una dificultad importante dentro del desarrollo, por un lado, existen resultados que no toman en cuenta una correcta posición de la figura, ya que primero se evalúa el área correcta, y por otro, si se asigna una población exagerada, se tiende a quedar en puntos negativos por esta misma razón, sin embargo, esto pudo ser resuelto, obteniendo resultados adecuados y logrando el objetivo de este proyecto.

Hernández López Ángel Zait:

En este proyecto se pudo apreciar el funcionamiento de un algoritmo evolutivo, usamos PSO ya que pensamos que era lo mejor para poder resolver el problema, por las características de que se encuentra en un plano cartesiano y las partículas tienen una posición y una velocidad por la cual se van moviendo por todo el campo de aprendizaje.

No obstante, se tuvieron dificultades al principio para poder hallar una solución óptima, ya que la función objetivo se enfoca más en la coincidencia de puntos aleatorios, y por tenerlo delimitado al cuadrante 1 del plano cartesiano, las partículas aprovechaban para poder usar todos los cuadrantes, por lo que satisfacía la coincidencia de puntos y solamente cambiaba el área de la figura que estaba aprendiendo.

Otra cuestión era que en una generación prematura, a la que el usuario ingresó, llegó a la mejor solución. Pero como se tenía que seguir moviendo la partícula y la coincidencia de puntos no era la mejor, este se seguía moviendo, dando por entendido un sobre aprendizaje.

Por último, al saber que casi todo se basaba en la coincidencia de puntos, se optó por alterar estos más seguido, dependiendo la figura a aprender. Por lo que al final se obtuvieron resultados exitosos, como se muestra en el reporte.

Tellez Perez Juan Manuel:
Para la implementación de éste proyecto se nos impuso el reto de adentrarnos en un área de los algoritmos que se van acercando al campo de la inteligencia artificial, los Algoritmos Evolutivos se han convertido en una forma muy eficaz de resolver problemas de optimización y en este caso nosotros ocupamos de implementar la solución de este proyecto en un Algoritmo PSO(Particle Swarm Optimization), donde todos los individuos se van comparando para que vayan trasladándose siguiendo al de mejor puntuación para que se puedan acercar más y más al concepto aprender que nosotros configuramos. Hubo muchos detalles de los que nos pudimos dar cuenta a la hora de correr la simulación, uno de los detalles fue que al momento de poner demasiadas generaciones ocasiona que las mejores soluciones se transportaran a coordenadas negativas donde se podía obtener la mejor puntuación ya que acertaron a todos los puntos negativos que habían y también lograban alcanzar casi la misma área que tenía el concepto que tenían que aprender. Nos dimos cuenta que al reducir los puntos aleatorios y también el número de generaciones podríamos llegar a mejores resultados y que el algoritmo llegase a aprender el concepto casi a un 100 por ciento.
Con este proyecto me ha quedado claro que la inteligencia artificial se ha expandido a muchos campos donde podremos explorar cómo solucionar diferentes problemas donde el campo de búsqueda es demasiado grande como para usar un algoritmo voraz o hasta incluso uno de programación dinámica.

Referencias

[1] C. A. Coello Coello, I*ntroducción a la Computación Evolutiva (Notas de Curso).* México D.F.: CINVESTAV-IPN, 2019
[2] S. G. de los Cobos Silva and J. G. Close, *Búsqueda y exploración estocástica.* Primera edición. México D.F.: Colección CBI, 2010
[3] A. Birk, *Learning Geometric Concepts with an Evolutionary Algorithm*. Saarbrücken Alemania, 1996.
[4] JavaScript | MDN [En línea] Disponible en: https://developer.mozilla.org/es/docs/Web/JavaScript
[5] Sass: Documentation [En línea] Disponible en: https://sass-lang.com/documentation
[6] Bootstrap (framework) [En línea] Disponible en: https://es.wikipedia.org/wiki/Bootstrap_(framework)
[7] webpack Concepts [En línea] Disponible en: https://webpack.js.org/concepts/
[8] @aas395/numjs [En línea] Disponible en: https://www.npmjs.com/package/@aas395/numjs#on-the-browser