Instituto Tecnologico de Costa Rica ISSN: 2550-2380
Vol 1 (2023) 19-11 https://www.tec.ac.cr/escuelas/escuela-ingenieria-computacion

ANALISIS DE ALGORITMOS:PAPER PROYECTO 2

BRANDON CALDERON[1] , PAMELA FERNANDEZ[2], GERARDO GOMEZ [3]Y ESTEBAN SOLANO[4*]

Abstract. Los algoritmos genéticos son un medio para simular el comportamiento y la evolución de los organismos, así como el funcionamiento de los genes en general. En este documento, presentamos una aplicación de estos algoritmos para la replicación de una imagen, utilizando JavaScript, CSS y HTML.

1. Objetivo

Demostrar en este proyecto una aplicación de los algoritmos genéticos, en el apartado de replicación de imágenes usando tecnologías web y algoritmos matemáticos.

2. Introducción

Un algoritmo es una serie de pasos que describen el proceso de búsqueda de una solución a un problema concreto. Y un algoritmo genético es cuando se usan mecanismos que simulan los de la evolución de las especies de la biología para formular esos pasos. (Núñez, 2019) El conocimiento de cada especie esta incorporado a la estructura cromosómica de sus miembros.

A partir de esto se intentará crear un algoritmo genético que pueda replicar imágenes cargadas en blanco y negro que sean lineales, se verá su progreso y el avance de las pruebas y los casos conforme avanza el proyecto.

2.1. **Medios.** Se usarán diferentes herramientas para su realización, como lo son el lenguaje de programación JavaScript, CSS y HTML, además de bibliotecas como OpenCV y Chart.js para poder generar las imágenes y el gráfico respectivamente.

Cabe recalcar que OpenCV será elemental en este proyecto, ya que permite dibujar pixeles en pantalla de una forma guiada, permitiendo así el poder generar las imágenes o las aproximaciones de la original. ("Introduction to OpenCV.js and tutorials", n.d.)

Chart es una herramienta que permitirá graficar la información en pantalla como el mejor fitness, el promedio y la generación a la cual pertenece la información mostrada.

Además, se planea utilizar la fórmula de la distancia euclidiana para calcular la distancia entre dos puntos, y el algoritmo de Bresenham para determinar los puntos intermedios que forman la línea entre estos puntos.(javatpoint, n.d.)

3. Funcionamiento

Como fue mencionado en la introducción, el objeto del presente paper son los algoritmos genéticos para la convergencia de imágenes lineales. Mediante extensa experimentación se logró obtener un algoritmo que cumpliera con este objetivo con un buen porcentaje de exactitud y una velocidad aceptable. Su funcionamiento es el siguiente:

3.1. **Generación del individuo.**

Los individuos se componen por puntos aleatorios conectados entre sí en un plano de 255x255 los cuales componen una línea. La generación de los puntos para el individuo es completamente aleatoria, limitada únicamente por la dimensión del plano utilizado (en este caso 255x255).

3.2. **Fitness.**

Antes de comentar el fitness se debe explicar la recolección de puntos válidos, estos son obtenidos iterando sobre los valores de píxel de la imagen de referencia y validando si son de color negro, en caso de serlo son agregados a una lista la cual es utilizada para el fitness. Este proceso sólo se realiza una vez al inicio del algoritmo lo cual causa que la primera iteración sea más lenta que las demás.

El fitness es calculado usando la distancia entre cada línea intermedia entre los puntos que componen al individuo y su punto válido más cercano, esta línea intermedia es encontrada usando el algoritmo de Bresenham el cual devuelve una lista los puntos que componen una línea entre un punto inicial y uno final. Una vez obtenidos los puntos intermedios se obtiene su distancia hacia su punto válido más cercano calculando la distancia euclidiana entre los puntos de la línea intermedia y todos los puntos válidos obtenidos. Una vez obtenida su distancia más cercana esta es agregada a una lista de distancias, este proceso se repite por todos los puntos intermedios.

Una vez realizado este proceso se califica el fitness en base a las distancias obtenidas, entre menores sean más se premia al fitness y viceversa. Este proceso se repite por todos los puntos que componen el individuo. Naturalmente, este enfoque consume bastantes recursos y es relativamente lento, pero lo compensa con su exactitud.

3.3. **Cruzar.**

Para el algoritmo de cruzar se crea una nueva lista vacía 'hijo' la cual obtendrá el cruce entre su padre y madre. Se itera sobre el largo de los puntos del individuo actual y se genera un valor numérico aleatorio entre 0 y 1, en caso de ser inferior a 0.5 se agrega al hijo el punto de la línea de la madre en la posición actual. En caso contrario se agrega el punto del padre en la posición actual. Este enfoque permite un cruce del 50% entre ambos individuos.

3.4. **Mutar.** En cuanto a la mutación, se itera sobre el largo de los puntos del individuo actual y se genera un valor numérico aleatorio entre 0 y 1, en caso de ser inferior a 0.15, se genera un punto aleatorio y se agrega al individuo en la posición actual, reemplazando el punto que anteriormente existía en dicha posición.

3.5. **Nueva población.** La generación de una nueva población consiste en un proceso de selección, cruce y mutación. Para este proceso se debe recibir tres porcentajes (selección, cruce, mutación) los cuales deben sumar 100% para poder generar una generación completa.

Primeramente, se obtiene una generación base la cual consiste en múltiples individuos, se calcula el fitness de todos los individuos de la población y se ordenan de acuerdo con su fitness de mayor a menor. Para la selección, se genera la lista 'selección' con los primeros individuos de la población, dependiendo del porcentaje seleccionado, asegurando obtener los individuos con mayor fitness. Seguidamente, se combinan los individuos de la selección obtenida en base al porcentaje de cruce. Para ello se obtienen dos individuos aleatorios de la selección para actuar de padre y madre y son combinados entre sí. Los resultados del cruce son agregados a una nueva lista de 'combinación'. Una vez terminado el proceso de cruce, se procede a la mutación la cual se basa en obtener los individuos obtenidos en el cruce y estos son mutados hasta obtener la cantidad requerida indicada por el porcentaje de mutación, estos valores también son agregados a una nueva lista de individuos 'mutacion'.

Una vez finalizados estos tres procesos se concatenan las tres listas obtenidas las cuales son utilizadas para generar una nueva población la cual será utilizada como población base de la siguiente generación. Este proceso se repite hasta un número máximo de generaciones determinado el cual al llegar a este límite se definirá al mejor individuo de la última generación como el resultado final del algoritmo.

4. Análisis Y Pruebas

En este apartado, veremos el progreso del proyecto a través de sus diferentes versiones y los cambios realizados, analizando el porqué de estos.

4.1. **Código versión 00:** Esta fue la primera versión del código, la cual se podría considerar como la base del mismo. En la versión 00, se establecieron los cimientos para las primeras pruebas del código..

Se utilizaron los siguientes datos base:

Porcentaje de selección = 0.2;
Porcentaje de mutación = 0.3;
Porcentaje de combinar = 0.5;

Cabe recalcar que la suma de estos datos siempre debe ser 100.

Este código funciona de la siguiente manera:

Se crearon clases con sus respectivos metodos y funciones para simular los elementos característicos de la genética, como los cromosomas, así como para representar la población y los individuos.
clases:

Individuo: se le pasa por parámetro indiv y un cromosoma en null. Esta contiene como atributos:
Cantidad de genes, ancho de la línea y el fitness.

Posee métodos como:

generarCromosoma: Este método permite generar los cromosomas de un individuo, los cuales se componen de un conjunto de puntos formados a partir de los puntos inicial y final de una línea. Al aplicar el algoritmo de Bresenham, se obtienen los puntos intermedios que conforman la línea, a la cual además se le asigna un grosor determinado.

cruzar: Esta función permite cruzar los genes, o mejor dicho, generar un intermedio entre dos individuos, generando un cambio genético en la especie. En esta versión, lo que hace es crear una línea intermedia entre dos líneas, correspondientes a los individuos padre y madre.

mutar: Esta función genera el cambio genético en los individuos, generando un efecto de evolución, que nos permite acercarnos más a la imagen deseada.

calcular fitness: Esta función calcula el fitness del individuo, determinando qué tan cercano es a la imagen cargada por el usuario. Además verifica que las líneas generadas para cada individuo, representadas por píxeles, coincidan correctamente con los píxeles de la imagen original.

clase poblacion: Recibe parámetros como individuo, la población en null y el tamaño de la población.

Posee los siguientes métodos :

ordenarPoblacion:Este método ordena la población, que es un arreglo de individuos, de menor a mayor.

calcularFitness(img): Este método recibe una imagen, calcula el fitness de la población y determina el mejor de todos.

nuevaPoblacion: Esta recibe un tamaño de población, porcentaje de selección de población, porcentaje de mutación, porcentaje de combinación, individuo y la imagen.

Este método tiene como objetivo:

Primero, seleccionar una población más pequeña basada en el tamaño total de la población utilizando el porcentaje de selección de población, y a partir de ahí, generar otra población.

Segundo, mutar esta población a partir del porcentaje de mutación.

Tercero, combinar los individuos según el porcentaje especificado para generar una nueva población.

Funciones genéticas:

Tenemos las funciones de generarLineas y generarPuntos que generan respectivamente lineas y puntos con base en los pixeles.

Tenemos la función de dibujar líneas, que permite ver lo generado por el algoritmo en pantalla.

Fitness: Se compara la imagen generada con el canvas para poder obtenerlo.

puntosIntermedios: Haciendo uso del algoritmo de Bresenham, se obtiene los puntos intermedios entre dos puntos y luego, los retorna.

obtenerPatron: Lo que hace es obtener el patrón de comportamiento de los puntos de una línea, mediante una longitud determinada de puntos.

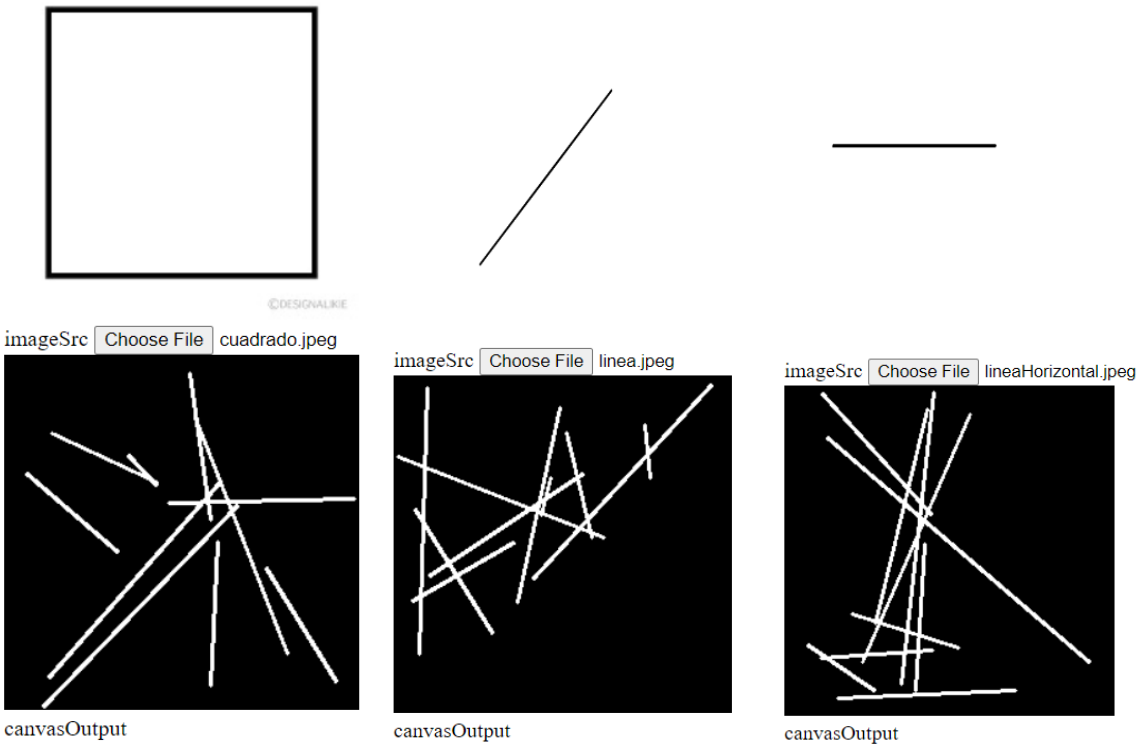
obtenerPuntoFinal: Obtiene el punto final de donde se empezarán a realizar cambios en la línea, con el fin de generar la evolución de un individuo.

obtenerPuntos(lineas): Esta función obtiene los puntos totales de una línea.

obtenerColorPixel: Obtiene el color de píxeles de una imagen.

obtenerPixelesImg: Obtiene los píxeles de dos imágenes dadas.

Pruebas:
Se realizaron las siguientes pruebas:



Como podemos observar con estas pruebas, no se logró acercarse a la imagen original.

4.2. **Código versión 0:** Basándonos en los resultados obtenidos anteriormente, fue necesario reanalizar el código y, de esta manera, buscar una solución al problema en la sección anterior.

Se realizaron los siguientes cambios en el código:

Se usaron los siguientes datos:

Porcentaje de Selección = 0.2;
Porcentaje de Mutación = 0.5;
Porcentaje de combinación = 0.3;

Se creó la función distanciaEuclidiana, que obtiene la distancia entre dos puntos.

Además se creó la función calificar, esta recibe una distancia y según sea la distancia de la imagen original, aumenta en un porcentaje predefinido según sea el fitness.

Se creó también la función obtenerPuntosValidos, la cual determina los puntos que coinciden con la imagen. Esto permite a partir de ahí, acercarse más al objetivo principal con las generaciones venideras.

Se optimizó el método de obtenerColorPixel.

Se le agregó el método de dibujarIndividuo a la clase individuo, esto con el objetivo de dibujarlo individualmente en pantalla.

Se modificó el cruzar, para así obtener mayor aleatoriedad de los resultados obtenidos.

Se modificó la mutacion, ya que este no estaba mutando de la manera que nosotros requeríamos(no funcionaba).

Se modificó el fitness con el fin de hacerlo por la proximidad de los puntos de la imagen generada con la imagen original, esto de una forma mas eficaz.

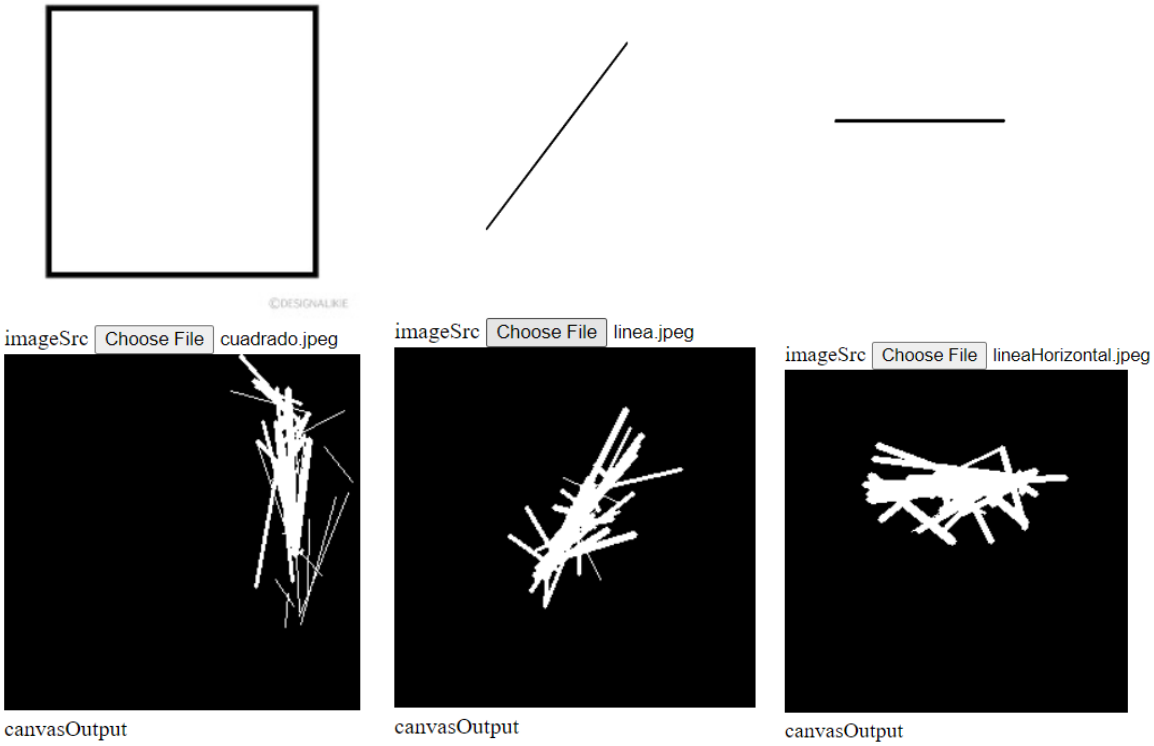
En el caso de la clase población:
Se agregó el método dibujar poblacién que se encarga de dibujar la población en pantalla.

Al método calcularFitness se le agregé el promedio, para así obtener el fitness promedio de la población.

Se realizarón algunos cambios en el método nuevaPoblacion , con el objetivo de relacionarlo con el dibujarPoblacion.

Pruebas:

Se realizarón las siguientes pruebas:



Como podemos observar con estas pruebas, se acercó mucho más que en las pruebas con la versión anterior.

4.3. **Código versión 1:** Basándonos en los resultados obtenidos anteriormente, se tuvo que reanalizar el codigo y de esta manera, buscar una solucion al problema de la seccion anterior.

Con base en lo anterior, se hicieron los siguientes cambios en el código: Se usaron los siguientes datos:

Porcentaje de Selección = 0.4;
Porcentaje de Mutación = 0.3;
Porcentaje de combinación = 0.3;

Se modificó el método calificar, de tal manera que, si se aleja mucho de la imagen orginal, castiga el fitness.

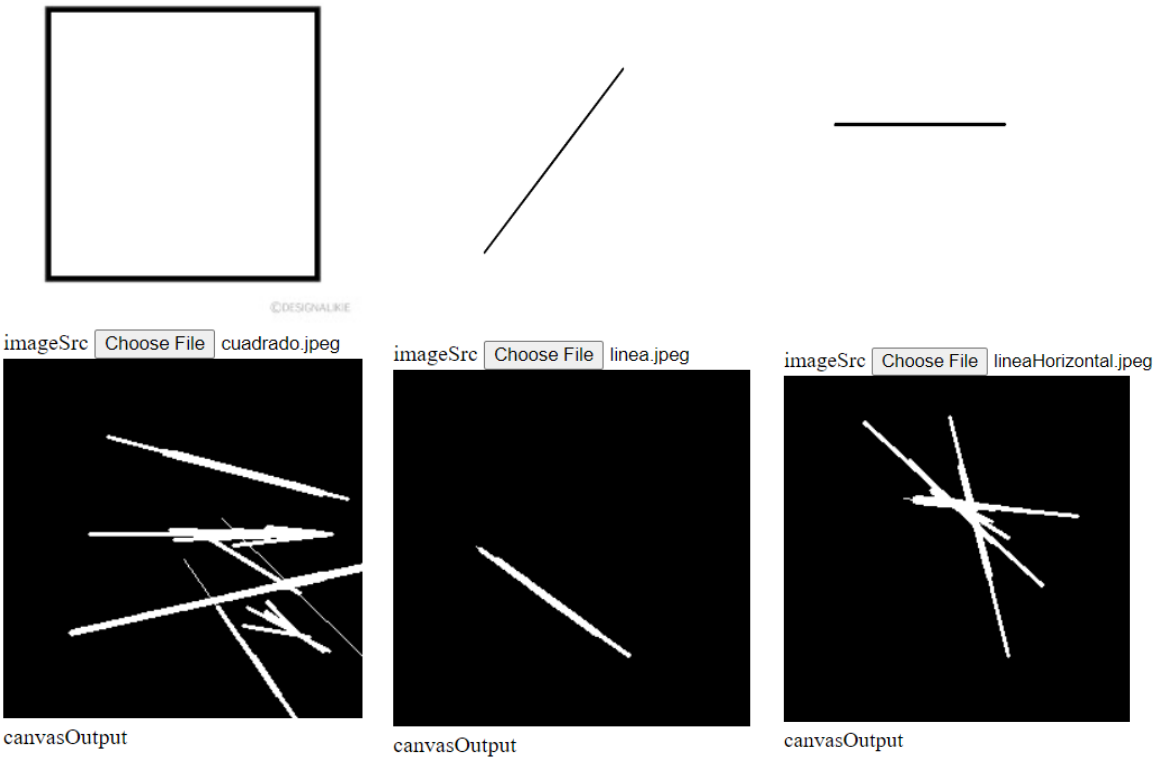
Se modificó en la clase individuo el método cruzar, tomando como referencia los puntos de la madre y el padre se generan puntos intermedios, generando así un tipo de cruce genético entre ambos puntos, simulando como sucede en los genes.

Se modificó en la clase individuo el método mutar, esta versión realiza la mutación de acorte o alargamiento de línea, además de validar que el ancho no sea menor a 1.

Se optimizó calcularFitness de la clase población.

Pruebas:

Se realizaron las siguientes pruebas:



Como podemos ver en los tres casos no se generó la imagen deseada y su comportamiento parece ser aún más errado que la versión anterior, pero parece que está mejorando en cuanto el ancho y condensación de las líneas.

4.4. **Código versión 2 :** Basándonos en los resultados obtenidos anteriormente, se tuvo que reanalizar el código y de esta manera, buscar una solucion al problema de la seccion anterior.

Con base en lo anterior se hicieron los siguientes cambios en el codigo: Se usaron los siguientes datos:

Porcentaje de selección = 0.6;

Porcentaje de mutación = 0.1;
Porcentaje de combinación = 0.3;

Se modificó la función calificar de tal forma que tiene mas variables de casos y no castiga en ningún momento sino, siempre aumenta el fitness.

En la clase individuo ya no se pasa un individuo sino uma imagen y con eso se hacen los cromosomas del mismo.

Se modificó generarCromosomas, de tal manera que se pudieran generar a apartir de la imagen.

Se agregó el método modificarFitness en individuo de manera que se castigue cada vez que se aleja el individuo de la imagen.

Se modificó calcularFitness de tal forma que no reciba un canvas sino solamente los puntos validos que coinciden y a partir de ahí generar el fitness.

Se agregó el método cruzar en individuo que recibe una imagen, otro individuo y puntos de coincidencia de la imagen original, para así generar y tener mayor aleatoriedad a la hora de generar el cruce.

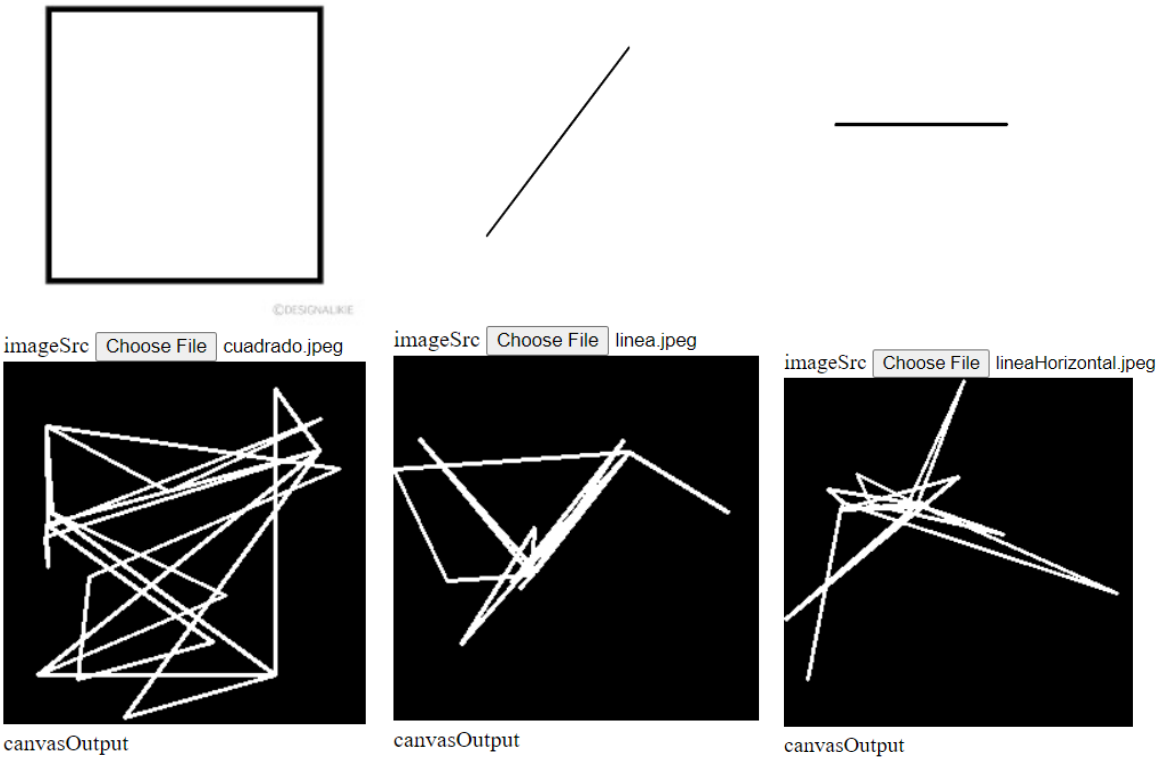
Se agregó el metodo mutar en individuo, donde lo que busca es generar la mutacion del mismo pero, apartir de una imagen y los puntos en los cuales coincidió con la imagen original.

El metodo calcular fitness en poblacion se modifico, de tal forma que solo funcione con los puntos de coincidencia con la imagen original y calcula el promedio del fitness.

Se modifico el metodo nuevaPoblacion en la clase poblacion, que se eleccionara ramndom un padre y madre, tambien una seleccion se reproduce hasta el tamano de la población y en mutación se le realizó cambios lógicos debido a la modificación anterior.

Pruebas:

Se realizaron las siguientes pruebas:



Como podemos ver en los tres casos no se generó la imagen desea pero se fue acercando más, si comparamos el caso anterior.

4.5. **Código versión 2.1:** Basándonos en los resultados obtenidos anteriormente, se tuvo que reanalizar el código y de esta manera, buscar una solución al problema de la sección anterior.

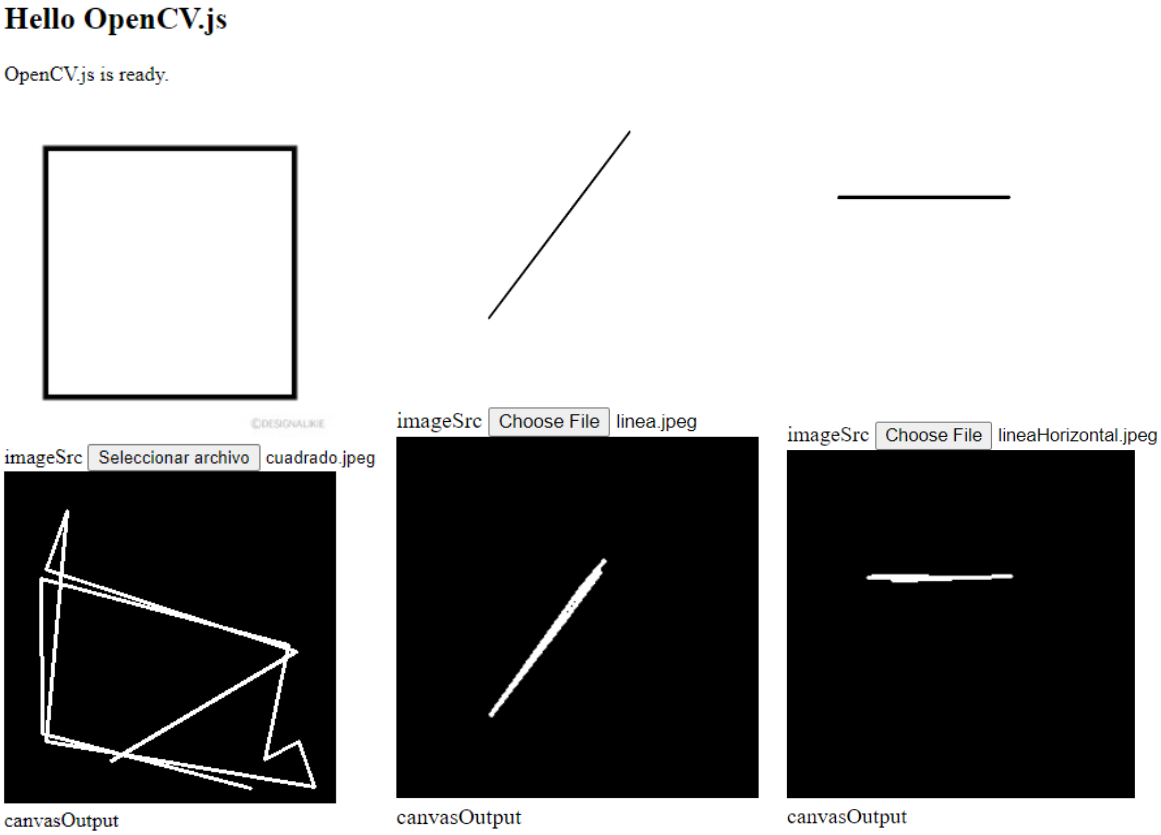
Con base en lo anterior, se hicieron los siguientes cambios en el código: Se usaron los siguientes datos:

Porcentaje de selección = 0.1;
Porcentaje de mutación = 0.4;
Porcentaje de combinación = 0.5;

Se eliminaron las funciones puntosMedios, calificar y obtenerColorPixel, obtenerPatron y obtenerFinal y se agregaron a genético.

Pruebas:

Se realizaron las siguientes pruebas:



Como podemos observar en los tres casos se acercó considerablemente más a la imagen deseada sobretodo con las lineas rectas, pero aún existen problemas con el cuadrado.

4.6. **Código versión 3:** Basándonos en los resultados obtenidos anteriormente, se tuvo que reanalizar el código y de esta manera, buscar una mejora a la solución de la sección anterior.

Con base en lo anterior, se hicieron los siguientes cambios en el código: Se usaron los siguientes datos: Porcentaje de selección = 0.2;

Porcentaje de mutación = 0.5;
Porcentaje de combinación = 0.3;

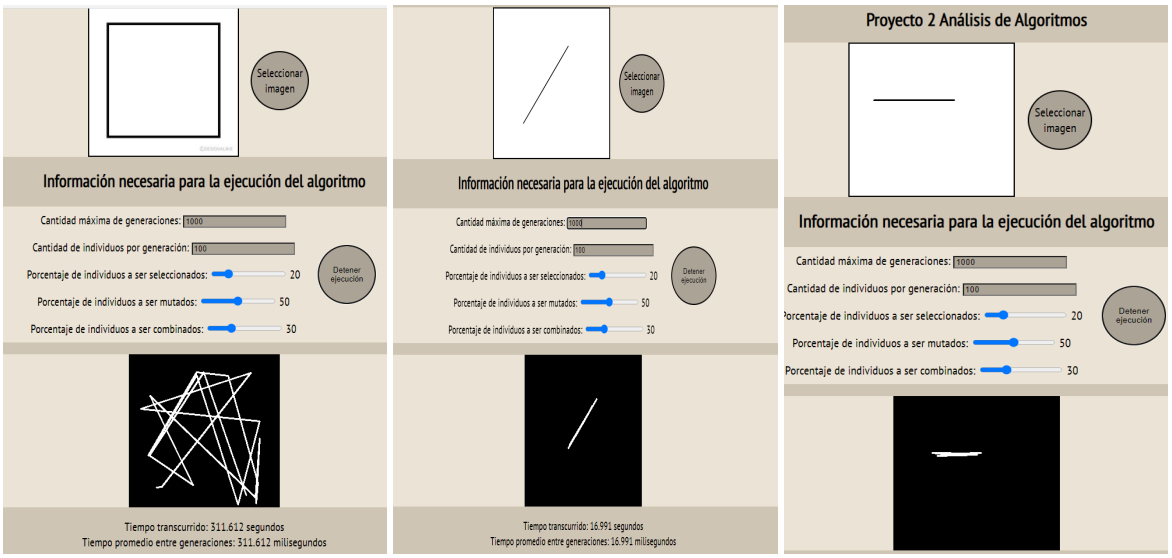
Se eliminó la division clases y genetico y ahora todo esta en genetico, se le agregó una interfaz y se le calcula el tiempo que dura calculando todo el algoritmo y su promedio.

Se modificó el calificarFitness de individuo, al cual se le agregaron mas variantes, tanto que aumentan como castigan la calificacion del fitness.

Se modificó calcularFitness eliminando ciertas líneas que no aportaban al resultado, o bien perjudicaban el resultado obtenido.

Pruebas:

Se realizaron las siguientes pruebas:



Como podemos ver en el caso del cuadrado, se desfazó completamente pero en los otros dos casos acerco por mucho a la imagen deseada casi que replicandola.

4.7. **Código versión 4:** Basándonos en los resultados obtenidos anteriormente, se tuvo que reanalizar el código y de esta manera, buscar una mejora a la solución de la sección anterior.

Con base en lo anterior, se hicieron los siguientes cambios en el codigo: Se usaron los siguientes datos:

Porcentaje de selección = 0.1;
Porcentaje de mutación = 0.4;
Porcentaje de combinación = 0.5;

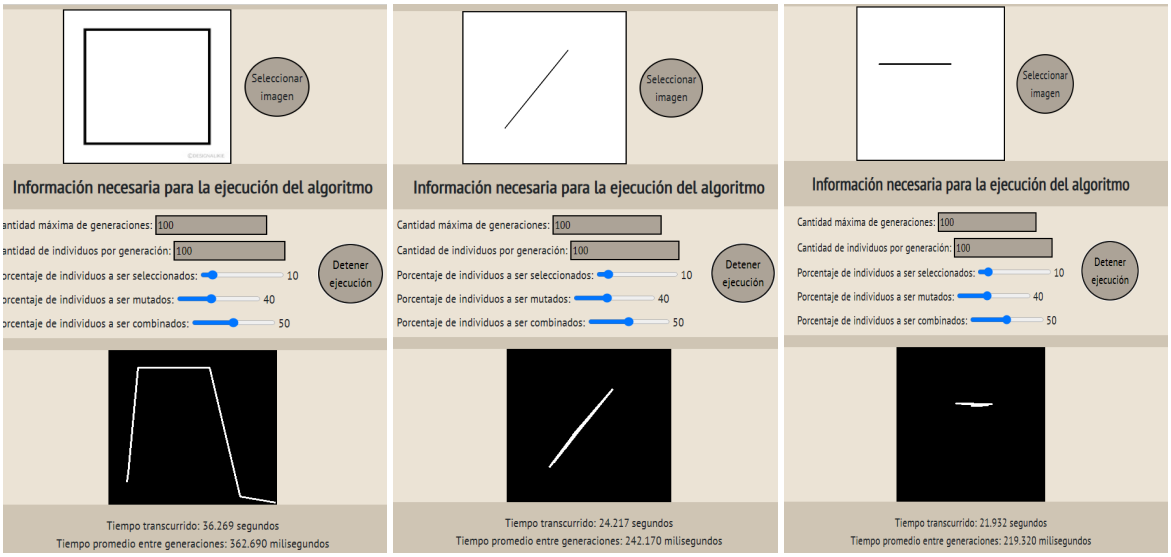
En este caso, se le agregó una gráfica, con el fin de poder ver, el mejor fitness de cada generación, así como el mejor y peor fitness generacional.

Se modificó el método calificarFitness de la clase individuo , reduciendo los casos , para evitar la especificidad y mayor posiblidad de cambio.

Se modificó el calcularFitness de la clase individuo, esto es con el fin de abarcar la mínima distancia posible y se agregaron ciertos procesos en los ciclos definidos con el fin de meter la mínima distancia de casa en una lista, de cada iteración.

Pruebas:

Se realizaron las siguientes pruebas:



Como podemos ver en el caso del cuadrado, se acercó aún más pero en los otros dos casos se acercó por mucho a la imagen deseada casi que la replicó como en el caso pasado, pero hay que trabajar el cuadrado.

4.8. **Código versión 4.1:** Basándonos en los resultados obtenidos anteriormente, se tuvo que reanalizar el código y de esta manera, buscar una mejora a la solución de la sección anterior.

Con base en lo anterior, se hicieron los siguientes cambios en el codigo: Se usaron los siguientes datos:

Porcentaje de selección = 0.2;
Porcentaje de mutación = 0.5;
Porcentaje de combinación = 0.3;

Se eliminó la función imgObtenerCanvas.

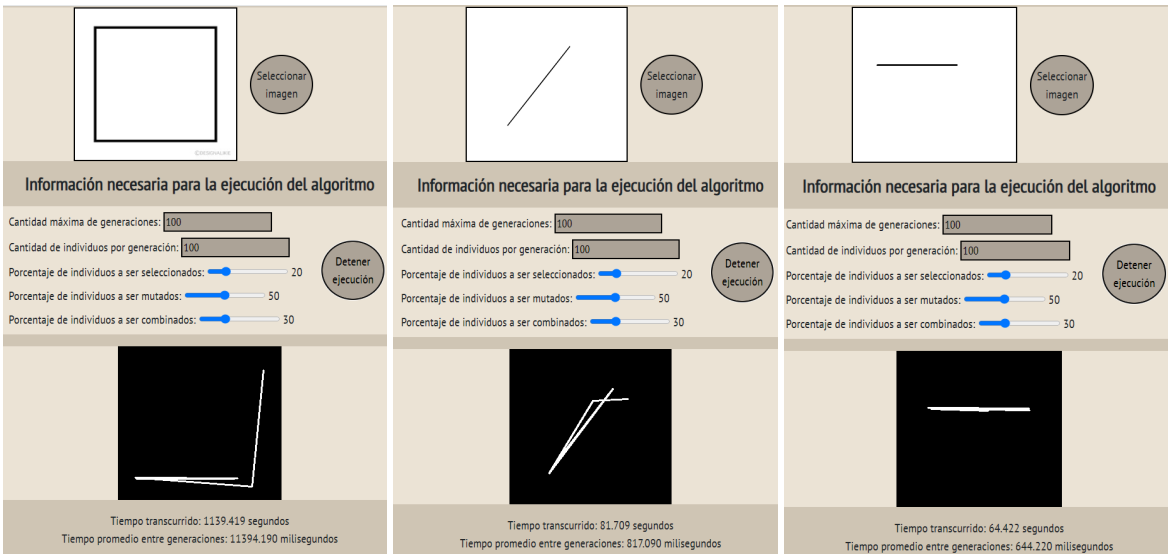
Se volvió a agregar la función puntosIntermedios, para así obtener los puntos que rellenan la línea, dados dos puntos.

Se modificó el método calificarFitness de la clase individuo esta vez tomando en cuenta más casos y castigando o aumentando según sea el caso.

Se modificó el método calcularFitness de la clase individuo, pero esta vez castiga a los puntos cercanos a la imagen.

Pruebas:

Se realizaron las siguientes pruebas:



Como podemos ver en el caso del cuadrado, se acercó más logrando replicar dos de sus bordes pero aún no logra converger completamente. En cuanto a los otros dos casos, se mantienen iguales aunque parecen haber perdido exactitud.

4.9. **Código versión 4.2:** Basándonos en los resultados obtenidos anteriormente, se tuvo que reanalizar el código y de esta manera, buscar una mejora a la solución de la sección anterior.

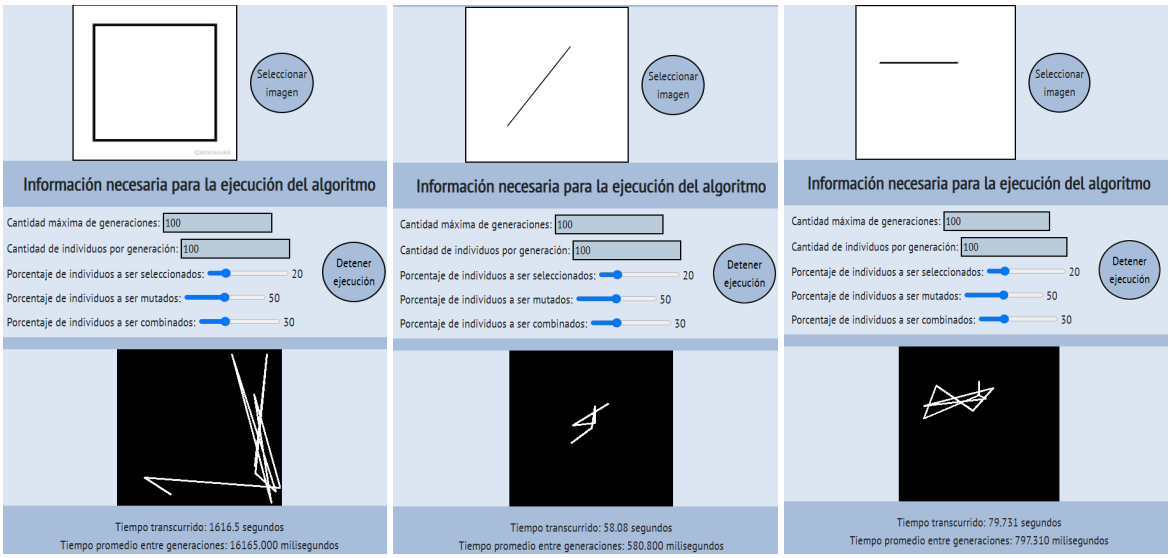
Con base en lo anterior, se hicieron los siguientes cambios en el codigo: Se usaron los siguientes datos:

Porcentaje de selección = 0.2;
Porcentaje de mutación = 0.5;
Porcentaje de combinación = 0.3;

Se modificó la función calificarFitness, la cual castiga más el hecho que se aleje de la similitud con la imagen original.

Pruebas:

Se realizaron las siguientes pruebas:



Como podemos ver en los tres casos se alejó de la imagen original. Esto se debe al cambio en la calificación del fitness. Al castigar más el fitness no permite que se valore lo suficiente cuando la línea está en una posición correcta.

4.10. **Código versión 5:** Basándonos en los resultados obtenidos anteriormente, se tuvo que reanalizar el código y de esta manera, buscar una mejora a la solución de la sección anterior.

Con base en lo anterior, se hicieron los siguientes cambios en el codigo: Se usaron los siguientes datos:

Porcentaje de selección = 0.2;
Porcentaje de mutación = 0.5;
Porcentaje de combinación = 0.3;

Se creo una nueva función agregarIntermedios que recibe los puntos actuales y los nuevos que se encarga de filtrar los puntos intermedios que no esten en el set actual y conservar esos y concatenarlos en el arreglo de puntos.

Se creó otra nueva función llamada compararPuntos que recibe los puntos validos y los otro set de puntos llamado puntos ind ,que lo que hace es filtrar los puntos en comunes y retorna estos ultimos.

Se modifico el metodo de calcularFitness se agrego la evaluacion que compara los puntos validos con los puntos intermedios. y la penalizacion y recompensa del fitness.

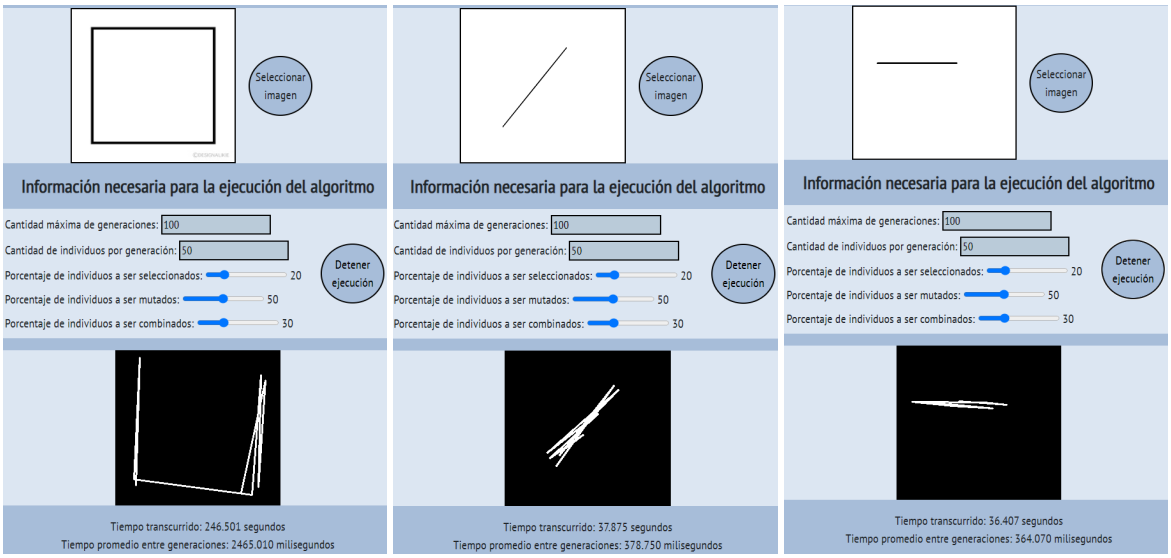
Se modificó el método calificarFitness de tal forma que penaliza mucho más ,0si se aleja el fitness y recompensa más ,si se acerca el fitness.

Se modificó el método mutar eliminando una línea de código.

Se eliminó el llamado de dibujar lineas dentro de la clase población.

Pruebas:

Se realizaron las siguientes pruebas:



En esta iteración del algoritmo se observa como, a pesar de que la exactitud disminuyó, la convergencia aumentó en cuanto al area correcta cubierta por las lineas generadas.

5. CONCLUSIONES

Como se observó en el apartado anterior, hubieron varias versiones, versiones entre otros, y se puede concluir que los algoritmos genéticos son dependientes meramente de la aleatoriedad, ya que por mas formas de tratar de guiar la generación, esta se ve influenciada por la misma (como naturalmente ocurre la selección natural).

Tambien cabe recalcar que estos algoritmos requieren tiempo y cierto poder de procesamiento, ya que con enfoque que utilizamos, se logró observar que dependiendo de la máquina, la ejecución a veces se pausaba a medio procesar o incluso la página no cargaba.

6. APRENDIZAJE

Con este proyecto se logró comprender a profundidad el funcionamiento de los algoritmos genéticos, al igual que netender su dependiencia sobre muchos factores lo cuales pueden ser incapaces de controlar, como lo seria la aleatoriedad, e incluso la tendencia de la poblacion hacia cierta mutación. A pesar de esto, mediante ajustes mínimo se observa como se puede lograr guiar a un algoritmo genético a cumplir con su objetivo de una forma altamente precisa

Además se logra comprender, un acercamiento a como funciona la genética en la vida real, proporcionandonos en este caso las imágenes y como este tipo de algoritmos permiten acercarse poco a poco al objetivo. Al igual se logra apreciar sus posibles aplicaciones en otras areas como lo es la ciencia y el comportamiento poblacional.

References

Introduction to opencv.js and tutorials. (n.d.). https://docs.opencv.org/3.4/df/d0a/tutorial_js_intro.html

javatpoint. (n.d.). *Computer graphics bresenham's line algorithm*. https://www.javatpoint.com/computer-graphics-bresenhams-line-algorithm

Núñez, L. (2019, February). *¿qué son los algoritmos genéticos?* https://elpais.com/elpais/2019/01/31/ciencia/1548933080_909466.html