Técnicas de Inteligencia Artificial: Memoria

Doodle - Aproximando imágenes con círculos

Erik Härkönen

PF3970492

23.10.2017

El problema

Descripción

El problema que intenta resolver Doodle es aproximar una imagen usando un conjunto de círculos semitransparentes. Doodle es un programa desarrollado por mí desde cero en C++, que implementa dos tipos de búsqueda local: un algoritmo genético, y enfriamiento simulado.

La dificultad del problema es encontrar vecinos de alta calidad – como el espacio de búsqueda es prácticamente continuo (ignorando da discretización a únicos pixeles y valores de color), cada solución tiene un número casi infinito de vecinos potenciales.

Otra dificultad es el coste muy alto de calcular el fitness de una solución: consiste en dibujar potencialmente cientos de círculos, y después en calcular la suma de distancias (cuadráticas) entre todos los pixeles de la solución y el objetivo.

Soluciones anteriores

Hay un montón de soluciones anteriores del mismo problema. La mayoría de ellos resuelven el problema usando triángulos o polígonos en lugar de círculos – la solución más popular es EvoLisa (Roger Johanson, 2008), aunque muchas personas sostienen que su tamaño de población de 1 hace que sea estrictamente un optimizador de escalada. Una solución conocido que usa círculos es GeneticImageCopy (Clinton Sheppard, 2011).

La mayor diferencia entre el uso de círculos y triángulos es que las esquinas introducen muchos detalles de alta frecuencia que ayudan en aproximar la imagen, mientras círculos tienen que ser muy pequeñas para añadir tales detalles.

Algoritmo Genético

La implementación de un algoritmo genético en Doodle consiste en un conjunto de objetos de la clase *Fenotipo*, o sea un conjunto de soluciones, que evolucionan mediante un proceso de cruce, mutaciones y reemplazo.

El algoritmo genético en Doodle tiene dos versiones de búsqueda: un modo dónde todos los individuos empiezan con un solo circulo en su genotipo, y con mutaciones correspondientes que añaden y quitan círculos (*genotipo en crecimiento*). También hay un modo dónde la cantidad de círculos es fija, y dónde las mutaciones solamente pueden cambiar los parámetros de los círculos (*genotipo estático*).

Con un genotipo en crecimiento, el algoritmo puede explorar la vecindad muy rápidamente al principio, gracias a la carga computacional más ligera, pero también puede encontrar óptimos locales que no sean óptimos cuando haya más círculos. Con un genotipo estático, la vecindad tiene una dimensionalidad menor, pero el resultado visual es menos impresionante al principio.

Estructura del fenotipo

El fenotipo consiste en un vector de genes, conocido como el genotipo, y de un método que evalúa el fitness del fenotipo dibujando los círculos que contiene el genotipo y comparando la imagen resultante con el objetivo.

El genotipo consiste $7n$ genes, cada uno un número entre 0 y 255 (8 bits de memoria), donde n es el número de círculos en el genotipo. Grupos de siete genes forman un círculo, con el tamaño codificado en los primeros 3 genes, y el color en los 4 restantes.

La razón por tener genes normalizados es facilitar cruces y mutaciones – con todos los genes del mismo tamaño, dos genotipos pueden ser combinados de manera totalmente aleatoria.

El mapeo de genes a parámetros asegura que son todos en el rango correcto al dibujarlas. Sin embargo, la selección de mapeo juega un papel importante en los característicos del algoritmo. Usando un radio máximo grande, por ejemplo, el algoritmo tiene problemas en encontrar círculos pequeños al final de la búsqueda para añadir detalles de alta frecuencia. El mapeo de genes a coordenadas también tiene una importancia grande: si los círculos pueden tener puntos medios fuera de las fronteras de la imagen pueden contribuir con astillas en los bordes de la imagen, pero también resulta en círculos completamente ocultos en la búsqueda.

La población inicial consiste en individuos completamente aleatorios – aunque sería posible añadir círculos iniciales al genotipo según los pixeles del objetivo, eso es fuera del alcance de este proyecto.

Función de fitness

La función fitness de un fenotipo es definido como

$$f(I,O) = K - \sum_{c=1}^{3}\sum_{w=1}^{W}\sum_{h=1}^{H}(I_{c,w,h} - O_{c,w,h})^2 - N\,,$$

donde *I* es la matriz de la imagen del genotipo, *O* es la matriz que consiste en los colores del objetivo, *N* es el número de círculos en el fenotipo, *W* y *H* son la anchura y altura de las imágenes, y *K* es el error máximo $3 * W * H * 255^2$ (no considera el término N, pero eso no genera problemas en la práctica). Quizás más correcto sería usar distancia euclidiana entre los colores, pero evitar la raíz cuadrada resulta en computaciones más rápidas y más importantemente correctos, como la cuadratura conserva el orden de los valores de fitness. Además, la diferencia aparente entre dos imágenes no necesariamente escala linealmente con la diferencia en los valores RGB.

Otra curiosidad es el parámetro *N*: está incluido para evitar soluciones con círculos ocultos, o círculos que no mejoran el fitness en general.

Para evitar calculaciones innecesarias, la implementación contiene un caché de fitness – el fitness es recalculado sólo si el genotipo ha cambiado de alguna manera.

La computación del fitness es el cuello de botella computacional del algoritmo genético – aunque la implementación usa OpenMP para paralelizar el bucle, el algoritmo pasa la mayor parte de su tiempo evaluando los valores de fitness.

Selección

El algoritmo genético en Doodle tiene dos métodos de selección: el uno por rueda de ruleta, y el otro por rango.

La selección por rueda de ruleta no funciona sin una modificación muy importante: el fitness peor es sustraído de todos los valores de fitness. La razón es la siguiente: los valores de fitness son enormes (miles de millones), y, por tanto, las diferencias entre las soluciones son pequeñísimas. Sin la modificación previa, las probabilidades de selección son prácticamente idénticas, que resulta en una falta completa de mejora.

La selección por rango es la versión estándar del algoritmo. En este caso, las diferencias pequeñas (en relación con el valor absoluto del fitness) no afectan la selección de manera negativa.

La manera concreta de elegir padres en Doodle toma inspiración de las matemáticas computacionales: los valores de fitness ordenados se tratan como una función continua por partes, y las probabilidades son normalizados para obtener una función de densidad de probabilidad (FDP). Después, los valores del FDP son sumados para obtener la función de distribución acumulada (FDA). Seleccionando valores aleatorios entre 0 y 1, los padres resultantes, obtenidos por búsqueda de la FDA, se seleccionan con una frecuencia proporcional a su valor de FDA.

Aunque una función bien diseñada es suficiente para obtener buenas propiedades de selección, Doodle incluye otra variable *p*, llamada 'selection cutoff', que simplemente excluye los peores *p* porcientos de candidatos de la selección.

Cruce

Doodle implementa un cruce genérico de n puntos. Como los genes del genotipo son bien normalizados, la implementación del cruce es muy simple. La entrada de la función de cruce consiste en dos copias de padres, que son modificados para obtener nuevos hijos.

El número de segmentos (*n*) usados es configurable, aunque se usa el valor *n=3* por defecto. Tras de seleccionar *n-1* índices aleatorios, cada segundo segmento del genotipo es cambiado entre los padres.

Mutaciones

Uno de los desafíos del problema de aproximación con círculos es encontrar vecinos buenos. Para aliviar esto, Doodle contiene varias mutaciones especializadas en encontrar vecinos de alta calidad.

AddCircle

La mutación 'AddCircle' añade un círculo al genotipo. Su posición y color son aleatoriamente iniciados.

AddCircleDecreasingRadius

Esta mutación añade círculos cada vez más pequeños. Esto sirve para aliviar el problema de encontrar círculos pequeños que contribuyen con detalles pequeños al final de la búsqueda.

RemoveCircle

'RemoveCircle 'quita un círculo aleatorio del genotipo. Si el círculo es innecesario, el fitness mejorará, gracias al parámetro *N* de la función de fitness.

Shuffle

La mutación 'shuffle' cambia el orden de dos círculos en el genotipo.

Random

Esta mutación inicializa cada gen a un valor aleatorio con una probabilidad pequeña.

Perturbation

La mutación tipo perturbación es sin duda la más importante para hacer explotación de alta calidad. La mutación añade un valor aleatorio a cada gen con una probabilidad pequeña. Soluciones con genes perturbados son muy cercanos en la vecindad.

Parámetros del algoritmo

Los parámetros del algoritmo genético son los siguientes:

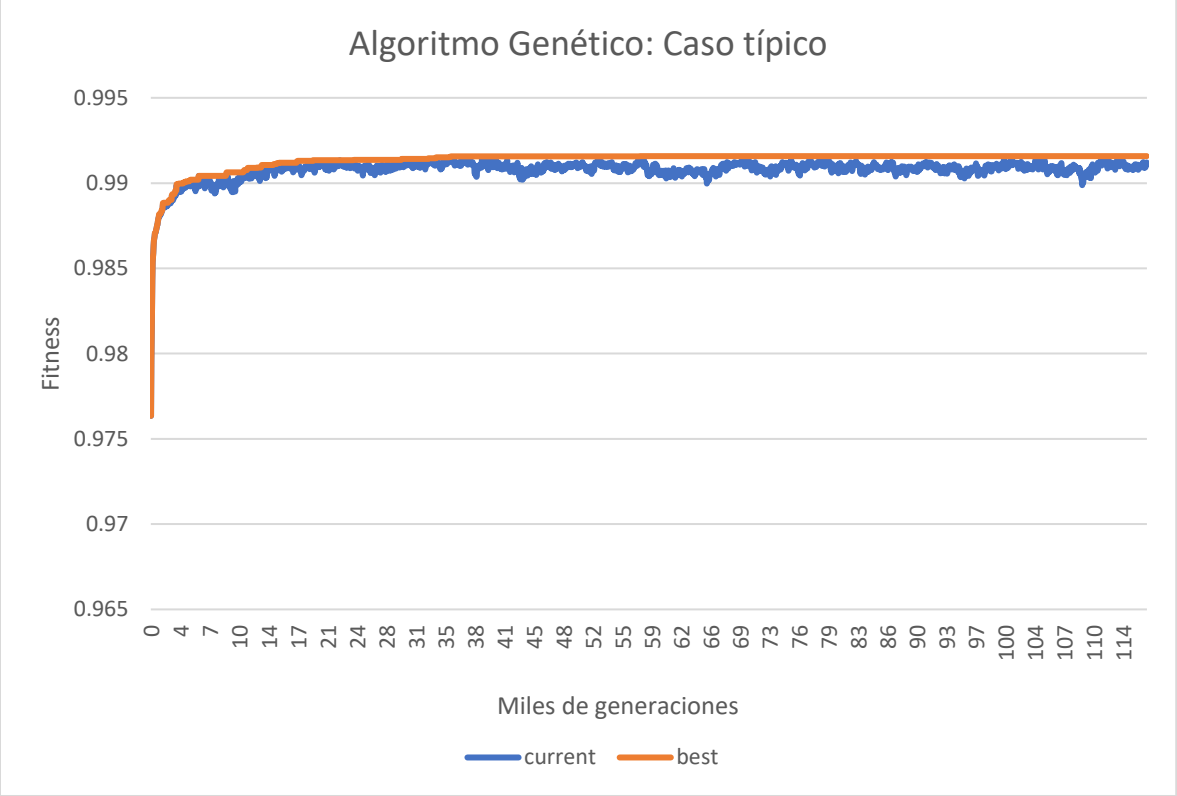
- Tamaño de la población
- Número de círculos iniciales
- Tipo de genotipo (fijo/en crecimiento)
- Algoritmo de selección
- Porcentaje de la población usado en la selección
- Número de puntos usados en cruce
- Las probabilidades de cada mutación

Análisis del rendimiento

El algoritmo genético es muy difícil de controlar. Un cambio pequeño de las probabilidades puede afectar muy drásticamente el comportamiento del algoritmo. En general, use demasiado tiempo para exploración, y no bastante para explotación. Además, la importancia de los cruces no está muy claro – a menudo resulta en reordenamientos completos en situaciones donde sería mejor añadir o mover círculos.

Durante todas mis pruebas con el algoritmo, todavía no he podido encontrar un conjunto de parámetros que funcione especialmente bien.

Caso típico: poca explotación



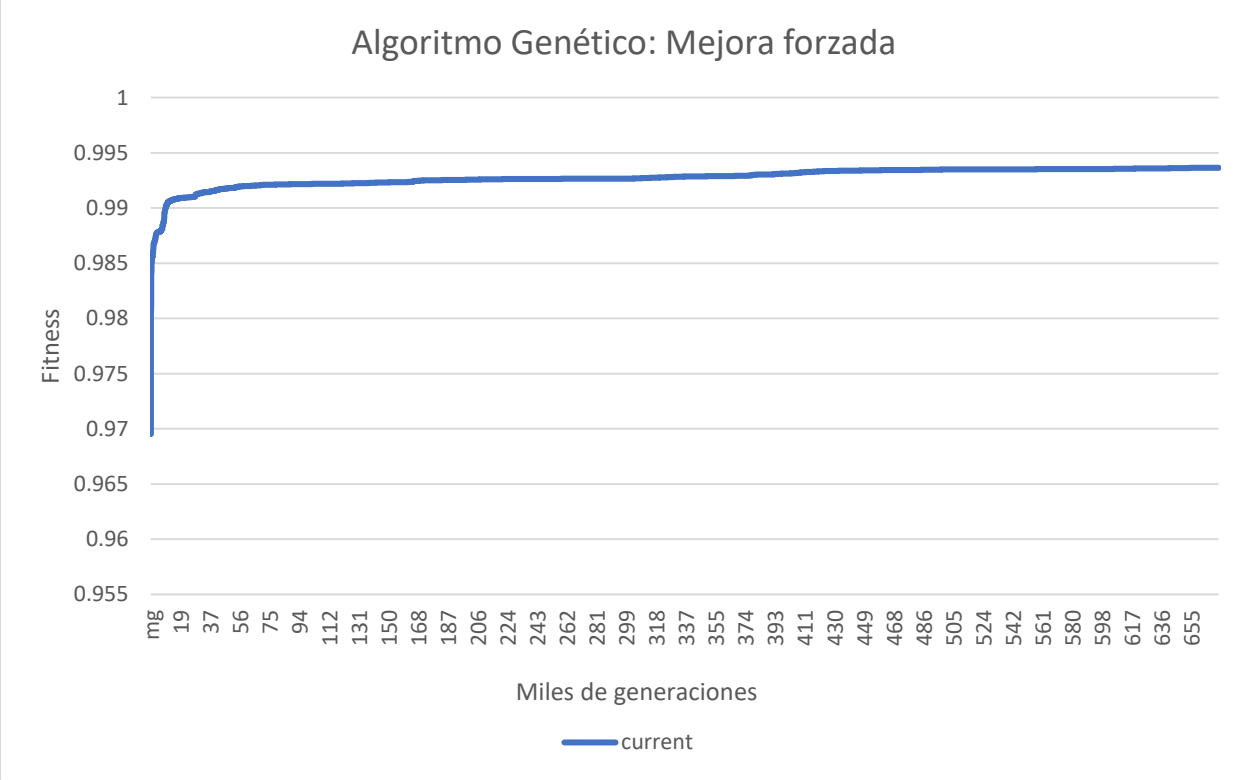
En el siguiente ejemplo vemos un comportamiento típico: el algoritmo hace mucha exploración, el fitness de la población en general no sigue subiendo. Esto indica un problema en los parámetros de la selección o una cantidad demasiado baja de explotación. El resultado es poco impresionante.

Los parámetros usados fueron:

- Reemplazo por rueda de ruleta
- Tamaño de la población: 256
- Porcentaje de individuos usados en selección: 15%
- Probabilidad de cruce: 1%
- Mutaciones aleatorias: 1%
- Shuffle: 5%
- Add Circle: 5%
- Remove Circle: 5%
- Perturbación: 3%
- Segmentos de cruce: 3
- Círculos iniciales: 1

Caso controversial: mejora forzada

Una variante controversial del algoritmo genético es la que fuerza la mejora de la población: la generación nueva reemplaza la antigua solamente si el fitness máximo del conjunto de soluciones ha mejorado.



Aunque no se puede llamar esta variante un algoritmo genético puro, su comportamiento en este problema concreto es mejor que el de la variante pura. El resultado indica que la explotación tiene un papel mucho más importante que la exploración en este problema específico.

Los parámetros usados fueron:

- Reemplazo por rueda de ruleta
- Tamaño de la población: 128
- Porcentaje de individuos usados en selección: 30%
- Probabilidad de cruce: 1%
- Mutaciones aleatorias: 1%
- Shuffle: 5%
- Add Circle: 10%
- Remove Circle: 10%
- Perturbación: 3%
- Segmentos de cruce: 3
- Círculos iniciales: 1

Enfriamiento simulado

El segundo método de optimización que contiene Doodle es enfriamiento simulado. El algoritmo usa una clase llamada 'Solution' derivado del fenotipo, y usa el método de mutación para encontrar vecinos, y el mismo método de calcular el fitness. En general, el enfriamiento simulado contiene menos parámetros y es mucho más fácil e intuitivo de controlar.

Parámetros del algoritmo

Los parámetros del algoritmo son la temperatura inicial T_0, la función de enfriamiento α, su parámetro k, las probabilidades de todas las mutaciones usadas para encontrar vecinos, y opcionalmente el parámetro de parada T_{end} .

Exploración de la vecindad

El algoritmo explora su vecindad de la misma manera del fenotipo: haciendo mutaciones aleatorias y perturbaciones más controlados. La diferencia es que su única manera de escapar de óptimos locales (salvo a pasos de empeoramiento) es mediante mutaciones aleatorias de genes, como no existe cruce.

Las mutaciones usadas son: AddCircle, AddCircleDecreasingRadius, RemoveCircle, Shuffle, Perturbation. Tienen sus probabilidades propias en comparación con el Fenotipo.

Igual como el algoritmo genético, enfriamiento simulado también tiene dos versiones de búsqueda: una donde el número de cirulos está cambiando, y otro estático donde la cantidad de círculos es fija.

Función de enfriamiento

La función elegida para enfriamiento es

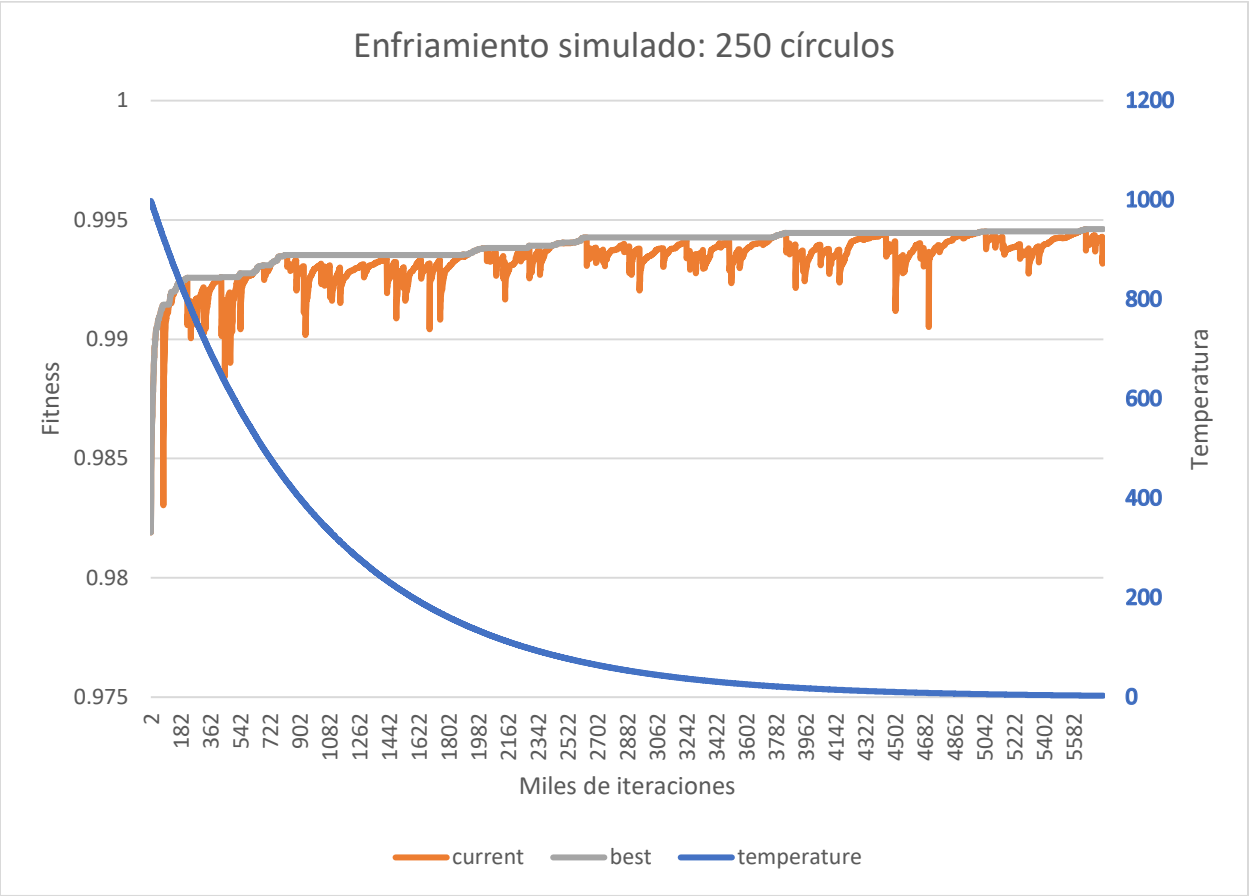
$$\begin{cases} T_0 = 1000 \\ T_{m+1} = k * T_m \end{cases}$$

donde $k = 0.99999$. La función elegida está aplicada en cada iteración del algoritmo, no sólo cuando sea elegido un paso de empeoramiento.

Análisis del rendimiento

Usando enfriamiento simulado, es mucho más fácil controlar la proporción de explotación y exploración. En general, el algoritmo suele dar resultados mucho mejores que el algoritmo genético, porque pasa más tiempo mejorando la solución, mientras que el algoritmo genético pasa un gran parte del tiempo explorando regiones peores del espacio de búsqueda.

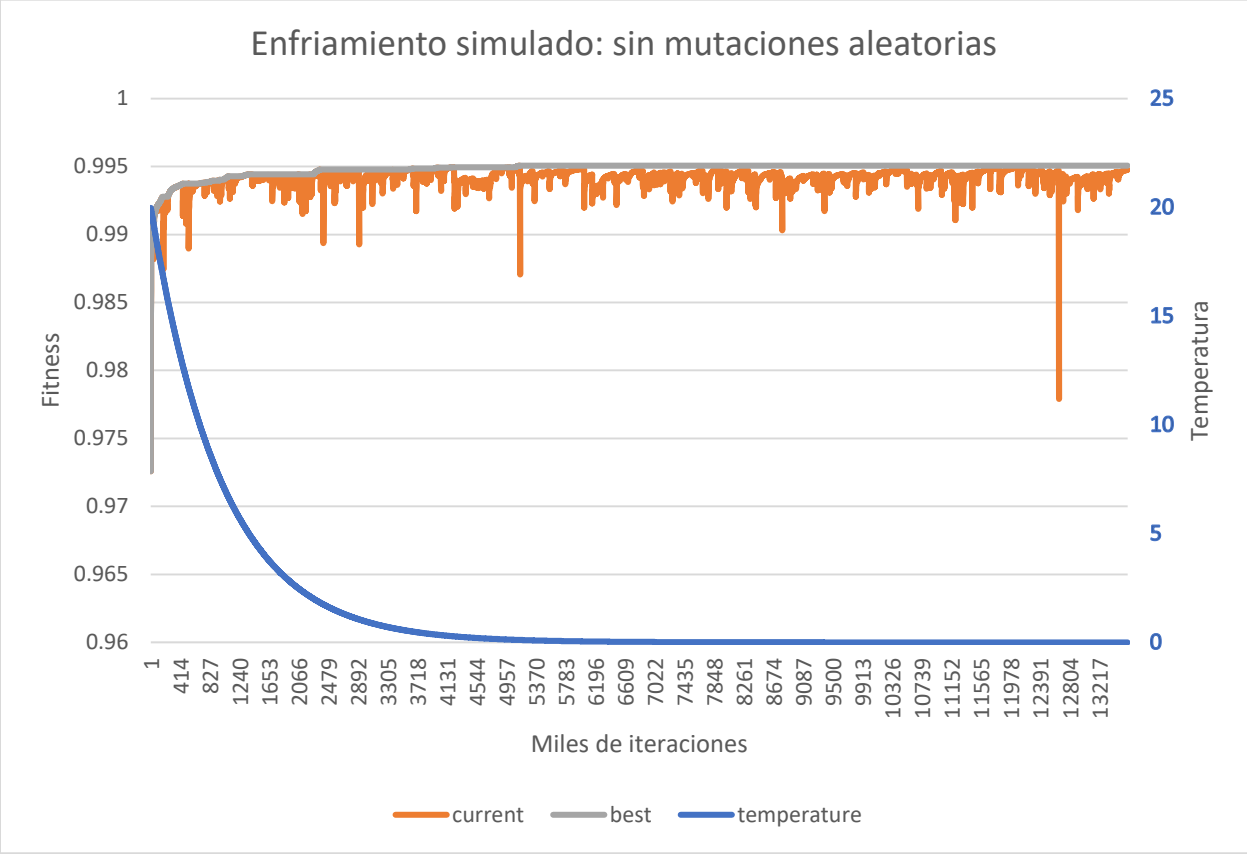
Caso bueno con 250 círculos



En este ejemplo, el algoritmo sique avanzando, aunque más lentamente todo el tiempo. El número de círculos es fijado a 250. Los parámetros fueron:

- K: 0.999999
- T_0: 1000
- Shuffle: 5%
- Add Circle: 0%
- Remove Circle: 0%
- Perturbación: 2%
- Círculos iniciales: 250

Caso bueno sin mutaciones aleatorias



En este ejemplo el algoritmo está explorando su vecindad sin mutaciones aleatorias, es decir, solamente usando las estrategias restantes como perturbaciones y pasos de empeoramiento resultantes del enfriamiento. El algoritmo alcanza rápidamente un valor muy bueno de fitness, pero después, su escalada está lenta. Puede indicar que la aleatoriedad tiene una importancia grande al final de la optimización, o quizás que el enfriamiento fue demasiado rápido en este caso.

Los parámetros:

- K: 0.999999
- T_0: 20
- Shuffle: 5%
- Add Circle: 10%
- Remove Circle: 10%
- Perturbación: 2%
- Círculos iniciales: 125

Reflexiones finales sobre el trabajo

Creo que elegí un problema bastante difícil, por un lado, porque es difícil encontrar vecinos de alta calidad, y por otro, porque las computaciones son tan pesadas que era difícil hacer muchas pruebas con parámetros diferentes. Aún otro problema es que, como humanos, somos muy eficaces en el procesamiento visual y, por lo tanto, detectamos fácilmente pequeñas diferencias en las imágenes, que resulta en una sensación de que el algoritmo funciona peor que en realidad.

Los algoritmos desarrollados son bastante buenos en aproximar los detalles de baja frecuencia de la imagen objetivo – como ejemplo, aquí una comparación con los detalles de alta frecuencia eliminados artificialmente con un desenfoque gaussiano:

Considerándolo todo, elegí una imagen bastante difícil para mis comparaciones.

He aprendido muchas cosas sobre la búsqueda local, y la impresión que me queda es que son bastante difíciles de controlar, y que la selección de los parámetros es tan importante como el desarrollo del algoritmo en sí mismo. Me molesta que no pudiera mejorar el algoritmo genético más, pero me alegra que funcionara tan bien el enfriamiento simulado.

Quiero agradecer al personal del curso por permitirnos elegir nuestros propios problemas – ¡nada es mejor para el aprendizaje del alumno que trabajar con algo que le interese!