Resumen

Existen procedimientos para el desarrollo y solución de problemas complejos de manera inteligente donde la existencia de técnicas, métodos y procedimientos permiten mejorar y encontrar estrategias en la búsqueda de buenos resultados cuales llamamos metaheurísticas. En este documento experimentamos con 2 técnicas inspiradas en la naturaleza cuales son “Particle Swarm Optimization” (PSO) y “Evolutionary Algorithms” (EA), mediante métodos de optimización en dimensiones acotadas, que con cierto número de interacciones, nos permitan visualizar su convergencia adecuada y diversa en los resultados. La técnica “PSO” trabaja sobre un conjunto de partículas que se mueven en la búsqueda de zonas atractivas con parámetros de inercia, aprendizaje, velocidad y posición, encontrando soluciones locales y globales. Por otro lado tenemos el “EA” que a través de una determinada población, con operadores de selección, cruzamiento y mutación, permiten a través de reproducción, crear nuevas generaciones hasta encontrar los óptimos que dan con la solución del problema.

Introducción

Para efectos del proceso de optimización, se realizaron experimentos con dos metaheurísticas conocidas, PSO y EA.

La primera metaheurística funciona a través de la inicialización de partículas en un espacio cartesiano, permitiendo mediante su movimiento espacial, la búsqueda de la mejor ubicación de forma colectiva, bajo una función que genere las restricciones adecuadas, logrando encontrar óptimos locales y globales. Durante el proceso de búsqueda cada partícula funciona a través de una función de velocidad vectorial, la cual incorpora factores de inercia, aprendizaje y componentes aleatorios que entregan posibilidades diversas, a través de la exportación o la explotación según corresponda, las soluciones que deseamos encontrar. Durante la ejecución de algoritmo se escoge un criterio de parada que consta de un parámetro de cota aproximada más cercana a 0, la cual mediante las evaluaciones y épocas desarrolladas, nos entregará el comportamiento de cómo converge hacia la solución factible.

La segunda metaheurística es un tipo de Algoritmo evolutivo, llamado Algoritmo genético, estos son inspirados en la naturaleza y evolución genética, y se utilizan para la optimización, aprendizaje y búsqueda de soluciones.

La idea es simular la evolución de las especies, donde sobrevive el individuo que está más adaptado al medio, es decir, la misma que subyace a la teoría de la evolución formulada por Darwin. Para implementar esto se debe transformar la resolución de cualquier problema en un conjunto de soluciones, en donde cada una de ellas funciona como un individuo. Todos estos individuos forman una población, la cual se irá reproduciendo dando paso a generaciones más fuertes (con mejores soluciones), es importante que estas descendencias sean mejores que sus padres, y para ello deben seleccionarse de manera adecuada aquellos que tengan las mejores características, pero sin perder otros que si bien no son tan aptos, en algún momento

pueden representar una mejor solución. Otro concepto clave es la mutación, que implica cambiar un valor del “cromosoma” de algún individuo, la mutación solo se dará en un porcentaje acotado de la población total, y esto les permitirá “adaptarse” mejor al medio.

Marco experimental

Particle Swarm Optimization:

Experimentalmente dentro de las heurísticas para procesos de optimización, tenemos los algoritmos de tipo enjambres, los cuales mediante una determinada población con parámetros de movimiento, aprendizaje e impulso, permiten la búsqueda de los sectores óptimos en regiones acotadas posibles a encontrar. Dentro de su funcionamiento describiremos a continuación las funciones básicas de la heurística utilizada:

partícula: se define la clase de un individuo (se habla de partícula xq el individuo con el que se experimenta no posee masa) para una población, donde sus parámetros importantes que la definen son: la ubicación en coordenadas cartesianas, valor de ajuste y velocidad. Dentro de la clase, se incorporan dos sub funciones importantes que evalúan el comportamiento dentro de una función objetivo y a través de estos parámetros el movimiento de la partícula.

Eval: método encargado de numerar las evaluaciones realizadas determinando épocas, limitar el movimiento de la partícula dentro del dominio de experimentación, obtener el valor de la función objetivo dependiendo de su ubicación geográfica, buscar el óptimo local y encontrar el óptimo global.

move: actualizador del movimiento de la partícula, cual calcula con los parámetros de velocidad descritos a experimentar que son: velocidad cartesiana, C1, C2, w, custom_l y custom_g, la ubicación de la partícula, y limitar el movimiento al espacio de búsqueda.

Mesh: función de mallado que permite desplegar el dominio cartesiano a evaluar en la función objetivo, esta función se despliega en coordenada “X” e “Y”, donde serán ocupadas en una función rastrigin con parámetro de superficie.

Rastrigin: función objetivo que se encarga de evaluar el comportamiento de cada partícula en el espacio, entendiendo que la representación será en el plano (x,y) simbólicamente.

delta: encargado de hacer variar parámetros de experimentación cuales serán C1, C2, w, custom_l, custom_g, con el objetivo de determinar los mejores valores de aproximación para una determinada población su convergencia de “entrenamiento”.

Algoritmo genético:

Se propuso un algoritmo genético con una población inicial de elementos inicializados aleatoriamente, para simplificar el manejo de las entidades se elaboraron dos clases principales GA y Individual que contienen toda la lógica correspondiente al algoritmo genético y a los individuos de la población respectivamente y una función que representa la la función objetivo, además de variables globales que permiten controlar la ejecución de las generaciones.

las variables mas importantes de mencionar corresponden a:

int generation_size: corresponde al tamaño de la poblacion para una generación.
int mutation_probability: corresponde a la probabilidad de realizar una mutacion.
int initial_individuals: corresponden a la cantidad de individuos con los que se inicializa el algoritmo.
Individual GA.optimal_individual: mantiene el registro del mejor individuo encontrado hasta el momento.

y en cuanto a las clases y funciones:

function objetive_fun: método float que contiene la función objetivo, Rastrigin.

Class GA: clase que contiene toda la lógica del algoritmo genético encapsulada en diferentes funciones y variables que representan elementos como la población y todas las transformaciones que se van realizando sobre ella. Al estar encapsulada en una clase nos permite tener múltiples inicializaciones de estas para sacar archivos csv que comparan varios métodos en una sola ejecución de código.

individual GA.init_population(): rutilizada para inicializar la poblacion en el constructor del GA retorna un arreglo tipo Individual con sus coordenadas random acotados entre -5.12 y 5.12.
void GA.sort_population_by_fit(): nos permite arreglar el arreglo de la población en relación a su ranking, es decir, ordenados de mejor resultado en la funcion objetivo a peor.

void GA.generate_prob_array(): nos permite crear un arreglo que represente la mayor mayor probabilidad de seleccionar individuos con mejor fitness para mejores “reproductores” pero sin perder la posibilidad de que aquellos no tan aptos puedan reproducirse.

void function GA.generate_selected_individuals(): seleccionará los ***generation_size*** individuos a reproducir, elegidos de manera random desde el arreglo de índices, siendo los más probables aquellos con mejor fitness.

void GA.mutate_population_by_random_value(): dada una probabilidad de mutación entregada como parámetro varía ya sea en valor “x” o “y”, con probabilidad de 50% de modificar una u otra, cambiando una de estas dos variables por un valor aleatorio en el intervalo de [-5.12, 5.12]. A medida que aumenta esta probabilidad de mutación nuestro algoritmo se volverá más exploratorio, podremos ver y analizar esto más a fondo en “Resultados”.

void function GA.generate_childrens(): se encarga de cruzar a la población, para esto intercambia de manera aleatoria x con y de ambos padres para generar dos hijos de cada par de padres.

void function GA.update_optimal(): en cada generación buscamos el mejor individuo y comparamos con nuestro óptimo global parcial, si es mejor, reemplazamos.

Class Individual: clase que contiene los parámetros asociados a cada individuo, valores x, y, además del valor de la función objetivo.
Dentro de esta clase se redefinió el método **compareTo,** que nos permite modificar el comparador de la clase Array de java para poder utilizar el mismo sort de arrays pero con clases.

Resultados

PSO:

El objetivo de la experimentación con PSO, se orientó en la búsqueda de los mejores parámetros que permitan buenos resultados para la convergencia del conjunto de partículas, bajo una función objetivo. Por lo que para ello se definieron los siguientes criterios que determinaron su experimentación a diferentes poblaciones, logrando resultados en una población específica, tomando en cuenta que en la literatura las experimentaciones son basadas en poblaciones no superiores a las 100 partículas:

Población: 400 partículas.
Dimensionalidad: espacio de búsqueda de 650x650 posiciones disponibles.
Objetivo: 1 solución óptima, la más cercana al 0.
Convergencia: El mejor individuo que llegue hasta un valor < 0.0001.

Intervalos: para la experimentación se escogen los siguientes dominios de experimentación: w=[1000,5000], C1=[0,30], C2=[30,0], custom_l[0,1], custom_g[1,0], usando los 3 primeros parámetros bajo las siguientes dependencias de variación, Población->w->C1 & C2 obtendremos los siguientes resultados, para posteriormente explicar los últimos 2 parámetros de experimentación.

Resultados preliminares:

- Experimentación con población de 400 individuos: los resultados poseen un cierto patrón en valores intermedios de inercia, los cuales presentan una convergencia errática en valores intermedios de inercia a medida que la población aumenta por lo que queda en duda si esto puede ser algún indicador importante. Se ve notoriedad en convergencia sostenida en unas pocas muestras realizadas [Anexo 1].

Basado en lo anterior se escogen los siguientes criterios, que nos permitan encontrar algunas convergencias con respecto a las épocas. La idea es extraer de la población intervalos sostenidos de aprendizaje, donde las reglas están definidas según el peso, y una muestra de 5 unidades de aprendizaje (C1 y C2) como mínimo de las observaciones totales, dentro de esas observaciones, calcular la media armónica que nos pueda dar un valor intermedio de una época específica o una vecindad cercana, de haber más de una se escoge la que tenga el número de evaluaciones menor, por último se realizarán experimentos de épocas / ponderación, donde la ponderación está basada el intervalo definido por custom_l y custom_g.

Resultados Finales:

Del conjunto de parámetros logramos obtener lo siguiente:

Experimento 18, con población=400, w=5000, C1=9.5, C2=21.5 [Anexo 2]:

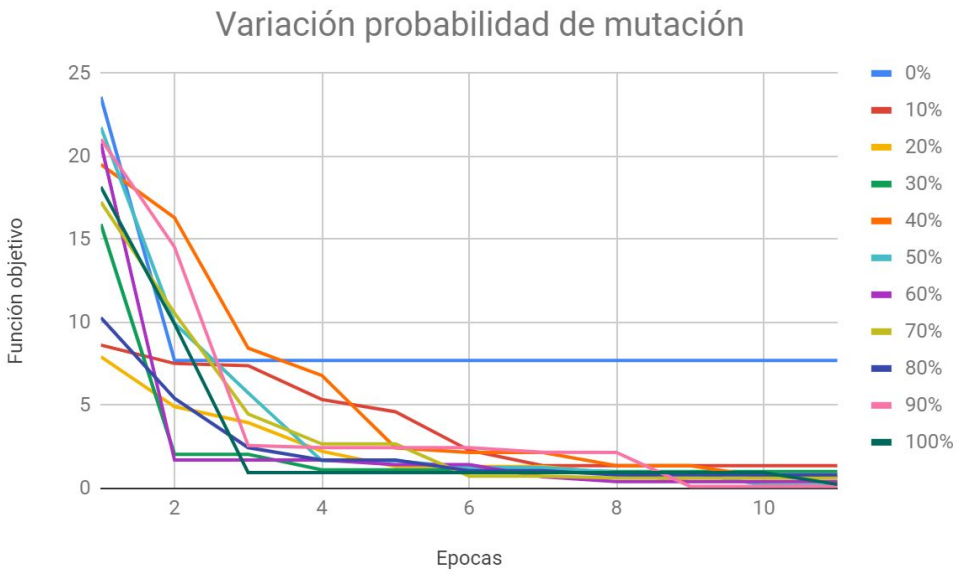
- se registra una aproximación de convergencia en el intervalo [~45,~80] de épocas,
- el comportamiento errático comienza a partir de la ponderación 0.9,
- no se observan peak's que eleven las épocas
- se observan 2 peak's que indican convergencia prematura, dichos peak's se encuentran fuera de la zona errática,
- se observan leves convergencias sostenidas en el intervalo de ponderación [0.1,~0.2], luego en [~0.2,~0.4] y por último se observa en [~0.5,0.7].

Link experimento PSO: https://youtu.be/rPDPVVwWMBo

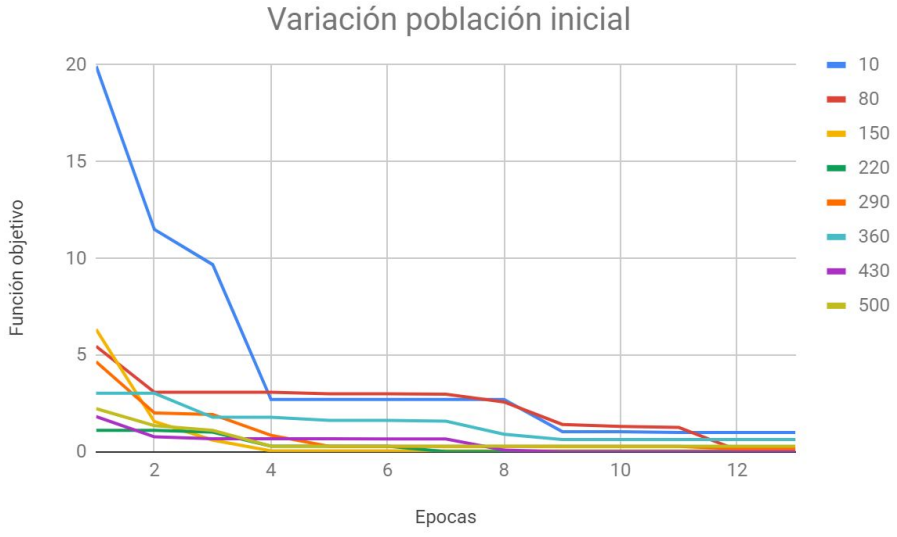
Algoritmo genético:

Los siguientes experimentos se hicieron con 100 generaciones (o épocas), pero para poder visualizarlos de mejor manera solo se graficaron 10 generaciones para variación de la probabilidad de mutación y 12 para los de población y tamaño de las generaciones, para poder observar como varía nuestro algoritmo después de las generaciones graficadas entrar en el video adjunto al final de este apartado, Grá.

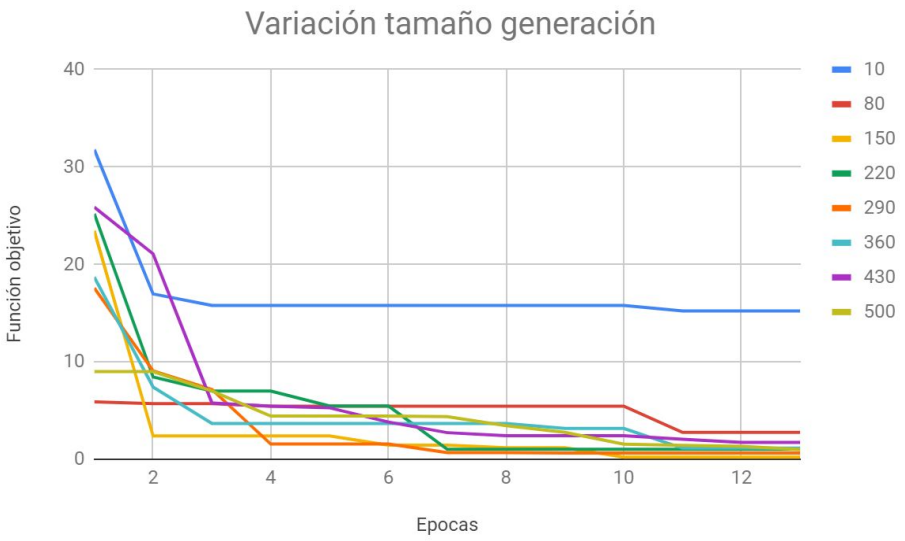
A continuación presentaremos 3 gráficos con sus respectivos análisis:



Análisis: Se varió la probabilidad de mutación, dejando los demás parámetros fijos, tamaño de las generaciones en 500 (cantidad de descendientes) y con 10 individuos como población inicial. Comenzamos la mutación con una probabilidad de 0% y lo fuimos aumentando de 10 en 10 hasta llegar a un 100% de mutación. Cuando no existe mutación (0%) se aprecia claramente cómo encuentra un “mínimo” y se queda estancado en él sin buscar mejores soluciones. A medida que esta probabilidad aumenta nuestro algoritmo se vuelve más exploratorio, abarcando mayores espacios. Si bien no se aprecia tanto en el gráfico, al superar el 60% de probabilidad, se vuelve demasiado exploratorio, perdiendo la capacidad de explotar los mejores puntos encontrados, no pudiendo encontrar una solución óptima a la función [Anexo 3].



Análisis: Se varió la cantidad de individuos iniciales del algoritmo, obtenidos de manera aleatoria, manteniendo los demás parámetros fijos, mutación en 10% y tamaño de las generaciones en 500. Comenzamos con una población inicial de 10, y los aumentamos de 70 en 70 hasta llegar a 500 individuos (que es el tamaño de las generaciones). Podemos observar que con 10 individuos inicia con un mal fitness, pero a medida que aumenta el número de generaciones (épocas) obtiene resultados similares a los demás. Si bien con todos se obtienen resultados similares, notamos las diferencias al principio, ya que a medida que aumenta la cantidad de población inicial, encuentra valores menores más rápido (menor cantidad de épocas).



Análisis: Se varió la cantidad de individuos por generación (descendientes) manteniendo los otros parámetros fijos, población inicial en 10 y probabilidad de mutación de 10%. Comenzamos con un tamaño de 10 (que es el tamaño de la población inicial) y los aumentamos de 80 en 80 hasta llegar a 500, que era el valor originalmente. Podemos observar que con 10 individuos se estanca en una mala solución, sin poder explorar otras alternativas, a medida que aumenta la cantidad de descendientes (individuos creados a partir de las generaciones anteriores) el algoritmo se vuelve más exploratorio. Es mucho más probable quedar atrapado en un mínimo local con una cantidad baja de individuos por época, podemos observar esto con más detalle en el video.

Video: https://www.youtube.com/watch?v=X8mqi8r4dpw&feature=youtu.be

Conclusiones

Al realizar una serie de experimentos con ambos algoritmos notamos que a medida que cambiamos los parámetros más importantes podíamos hacer que estos tengan un comportamiento más exploratorio o más enfocado en la explotación.

En el caso del PSO, para determinado número de población la inercia, y factores de aprendizajes logran dar con buenos resultados en la convergencia del problema de optimización donde se ve que las estrategias de exploración y explotación en términos equilibrados logran liberar de cierta forma la estocacidad. Se logró ver además que a medida que aumentaba la población con la inercia desaparecen los peak's de divergencia, y disminuían los peak's de convergencia prematura mejorando además la tendencia de convergencia a intervalos pequeños de épocas.

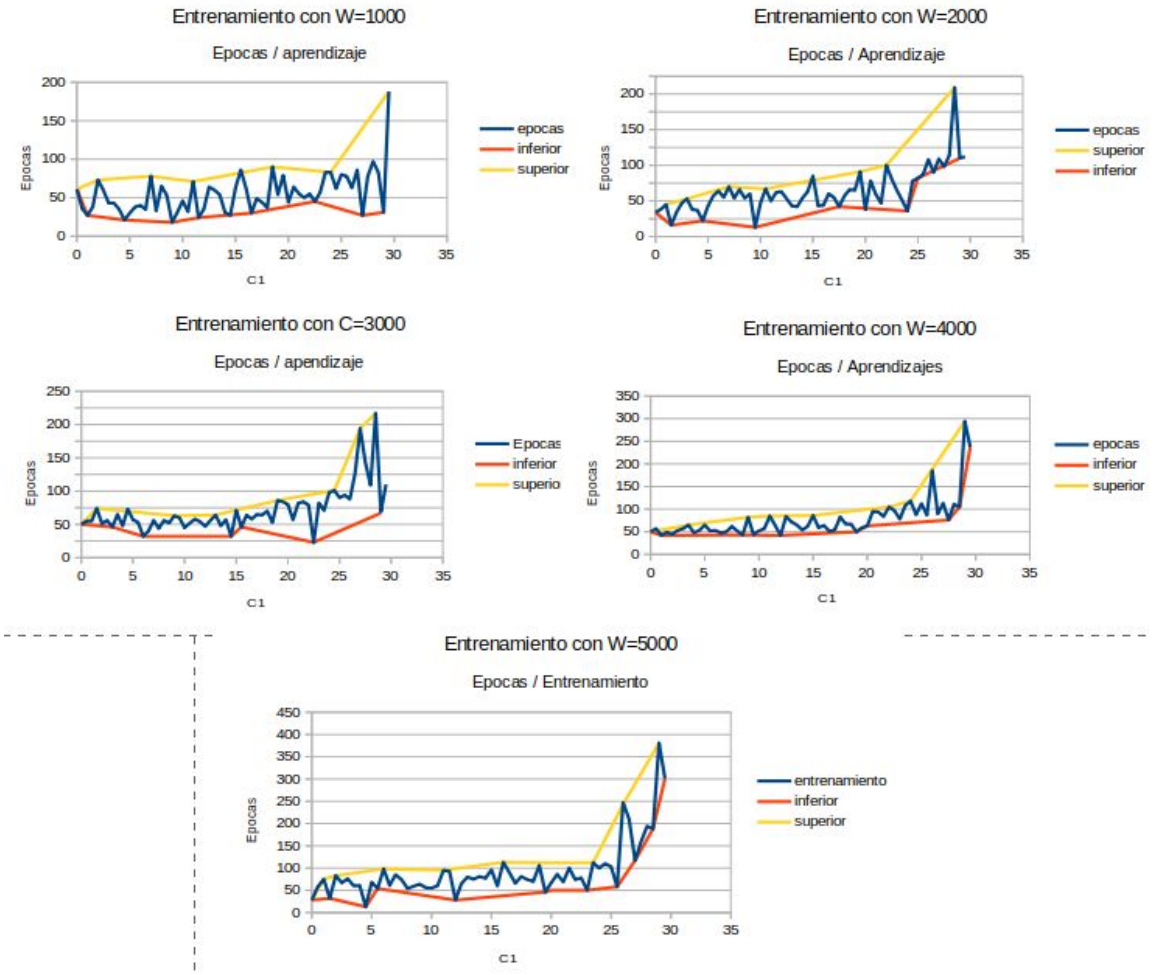
Por otra parte, en el Algoritmo genético una de las cosas que más nos llamó la atención fue la capacidad de mutación, ya que si no existía está característica se quedaba estancado siempre en un óptimo local, pero al varían en tan solo un 1% ya podíamos tener cierta capacidad de mutación que nos permitía obtener mejores resultados, a medida que este porcentaje aumentaba, también lo hacía su exploración. La cantidad de población inicial, al igual que el tamaño de las generaciones, nos permitía encontrar valores más bajos más rápidamente.
Otro aspecto importante a mencionar es la elección de los reproductores. Encontramos diversos métodos para seleccionar individuos, algunos de estos métodos se enfocan demasiado en cruzar los mejores elementos, perdiendo la posibilidad de cruzar aquellos no tan buenos, y por ende aumentando la probabilidad de quedar atrapados en un óptimo local (muchos hijos de los mismos padres), por lo que no solo se debe tener cuidado con la elección de los parámetros para poder obtener buenos resultados, sino también con la manera de implementar el método de selección.
Finalmente el cruzamiento igual es un aspecto a considerar, depende mucho del tipo de problema al que nos estemos enfrentando, existen diversas formas de combinar los elementos, y es importante una buena elección para obtener mejores resultados.

Referencias

- http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/temageneticos.pdf
- "Genetic Algorithms + Data Strutures = Evolution Programs" de Z. Michalewicz.
- "Metaheuristics: From design to implementation" de E-G Talbi
- https://www.cienciadedatos.net/documentos/48_optimizacion_con_algoritmo_genetico#cruzar_dos_individuos_(crossover,_recombinaci%C3%B3n)

Anexos

Anexo 1:



Anexo 2: