Investigación del Algoritmo genético, el TSP y Chu-Besley

Article on which of the parameters that are involved in a genetic algorithm have the greatest influence on the result of the traveling agent problem.

Autor 1: Mauricio Garcia Calderon
Universidad Tecnológica de Pereira, Risaralda, Colombia
mauro811@utp.edu.co

***Resumen*— En este artículo se pretende explicar cuáles de los parámetros que se usan en un algoritmo genético tienen mayor incidencia en el resultado del problema del agente viajero. Se propone analizar un algoritmo genético hecho en Python, donde se comparan dos tipos de selección: Por torneo y por ruleta. Se realizan diferentes pruebas para la solución del TSL usando los tipos de selección bajo los parámetros: número de individuos, número de interacciones, probabilidad de cruce y probabilidad de mutación.**

***Palabras clave*— Algoritmos eficientes, problemas de optimización, problema del agente viajero, TSP, algoritmos genéticos, mutación, cruce, ruleta, torneo.**

Abstract*— This article aims to explain which of the parameters used in a genetic algorithm have the greatest impact on the outcome of the traveling agent problem. A genetic algorithm is proposed where two types of selection are compared: By tournament and by roulette. Different tests are carried out for the TSL solution using the types of selection under the parameters: number of individuals, number of interactions, probability of crossing and probability of mutation.

Key Word — Efficient algorithms, optimization problems, travel agent problem, TSP, genetic algorithms, mutation, crossover, roulette, tournament.

I. INTRODUCCIÓN

El problema del agente viajero (TSP) es un problema de optimización combinatoria en la que una persona visita sólo en una ocasión cada una de las ciudades y se regresa al punto de partida; se deberá localizar la ruta que presente la distancia más corta y a ésta se la conoce como la ruta óptima, A medida que crece el número de ciudades a visitar por el agente viajero, no es factible resolverlo por programación entera debido a que el tiempo de solución computacional crece de forma exponencial de acuerdo con el número de ciudades visitadas; a este tipo de problemas se los conoce como no polinomiales (NP), También se utilizó el AG para resolver algunos ejemplos en donde la solución no pudo ser encontrada por programación, debido a que el número de restricciones crece de forma exponencial al aumentar el número de ciudades visitadas.

II. CONTENIDO

1) Algoritmo genético (AG)

Los algoritmos genéticos fueron introducidos por John Holland a finales de los años 60 inspirándose en el proceso observado en la evolución natural de los seres vivos. Son algoritmos de búsqueda basados en la mecánica de la selección natural y en la genética. Estos combinan la supervivencia de los individuos más aptos entre las cadenas de estructuras con un intercambio aleatorio para formar un algoritmo de búsqueda . Los algoritmos genéticos están determinados por tres características fundamentales: selección de parejas, cruza y mutación figura[1].

2) TSP con AG

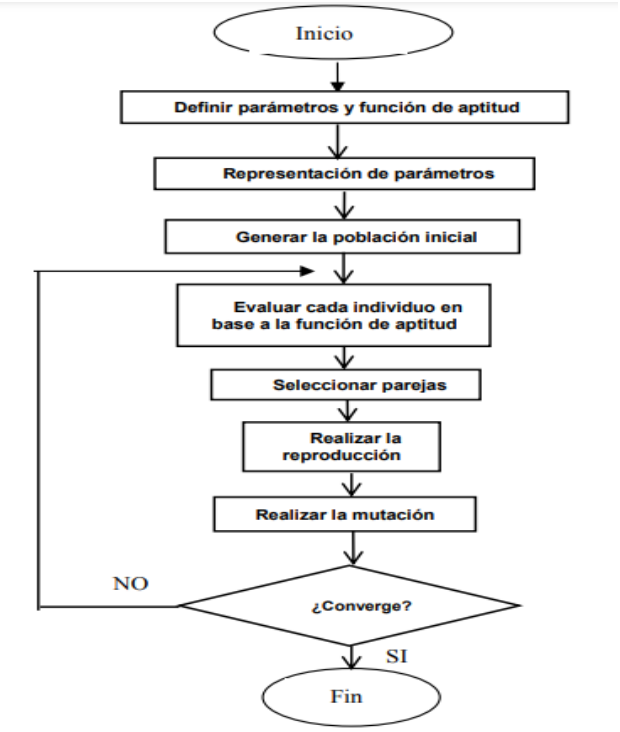
En el TSP resuelto con AG se realizan pruebas para determinar los mejores operadores y parámetros; los operadores son cruce, mutación, selección etc. y los parámetros son: tamaño de la población, criterio de paro, entre otros. El primer aporte es la codificación del TSP a través del AG y la selección de los mejores parámetros y operadores , especialmente la comparación del operador de selección en ambos casos.

ALGORITMO GENÉTICO

Los AG buscan proporcionar una calificación para la supervivencia en la búsqueda de un buen resultado. En la naturaleza, los individuos más aptos tienen más probabilidades de sobrevivir y aparearse, por lo que la próxima generación debe ser mejor. Esta misma idea se aplica al TSP, en donde primero se tratan de encontrar las regiones de soluciones y luego combinar a las más aptas para crear una nueva generación de soluciones que deben ser mejor que la generación anterior. También incluyen un elemento mutación al azar para evitar la degradación entre los elementos,

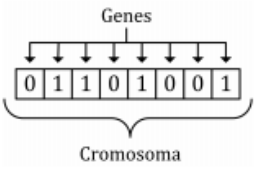
Fecha de Recepción: 26-marzo-2021

evitando caer en el óptimo local. Una codificación adecuada encuentra la solución al problema de manera aleatoria utilizando el concepto de que cada posible solución tiene una codificación única. La población inicial se genera de manera aleatoria o introduciendo en esta generación inicial una o varias soluciones encontradas previamente con otras técnicas. Para la evaluación se toma una simple solución como parámetro y devuelve un número que indica lo buena que la solución es. Por sí mismo el número devuelto no significa nada, solamente al ser comparado con otro valor podemos seleccionar a la mejor solución. Los operadores utilizados por los algoritmos genéticos son selección, cruce y mutación y la estrategia consiste en encontrar los parámetros más adecuados para solucionar el problema.



figura[1]

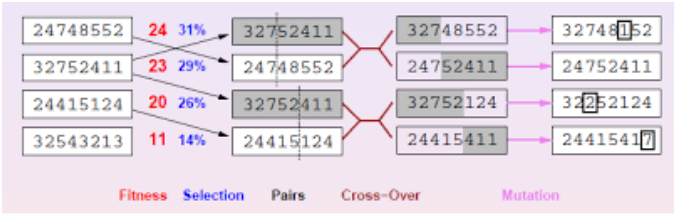
1.1 Población Inicial: La población inicial se forma a partir de las posibles soluciones del problema, donde una posible solución equivale a un individuo en términos de AG. Es así como un conjunto de N individuos representa una población inicial de tamaño N, donde N es un parámetro de entrada del algoritmo genético y generalmente y se conserva durante la ejecución del algoritmo. En un algoritmo genético los individuos de la población inicial pueden representarse como un conjunto de parámetros conocidos como genes. Estos se agrupan formando una cadena llamada cromosoma.
Generalmente los individuos que forman parte de la población inicial son generados de forma aleatoria, al hacer esto aleatoriamente, se obtiene una población inicial uniformemente distribuida en el espacio de búsqueda. La capacidad de exploración inicial depende del número total de posibles soluciones y el número de individuos que forman parte de la población. Cuanto mayor sea el número de individuos de la población, mayor será el poder exploratorio del algoritmo, sin embargo, esto aumenta el tiempo de cómputo.




```
#INICIANDO INDIVIDUOS Y POBLACION
toolbox.register("individual", tools.initIterate, creator.Individual, toolbox.permutation)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)
```

figura[2]

1.2 Función de evaluación o Fitness:
La evaluación de la población depende de la calidad relativa de los individuos que compiten por el aumento de la presencia en la población y por participar en las operaciones de reproducción. En un problema de búsquedas de optimización .Es simplemente un valor numérico que se calcula mediante una función de evaluación específica del problema, la cual se llama función de aptitud de un individuo, es un número real que refleja niveles de adaptación al problema, la función de la aptitud tiene un influencia muy importante en la función AG, ya que guía el mecanismo de exploración en el espacios de búsqueda, además así se cumple con el criterio del problema de la optimización ya se dé minimización y maximización de un objetivo así como las restricciones presentadas en el mismo.

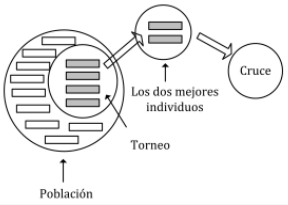



```
#FUNCION FITNESS
def evaluate_func(individual): #Funcion evaluacion
    a = numpy.array(flu)
    size=len(a)
    cost=0
    for i in range(size):
        for j in range(size):
            cost += flu[i][j] * dis[individual[i]-1] [individual[j]-1]
    return cost,
```

figura[2]

1.3 Operación de Selección: La selección es una parte fundamental del funcionamiento de un algoritmo genético ya que es donde se escogen los candidatos a reproducirse y crear individuos que formarán la siguiente generación, a los individuos seleccionados se les llama padres. La selección de pares se efectúa al azar usando algún procedimiento que favorezca a los individuos con mejor valor fitness, aunque también es necesario incluir un factor aleatorio que le dé la

oportunidad a los individuos de menor valor fitness, reproducirse.



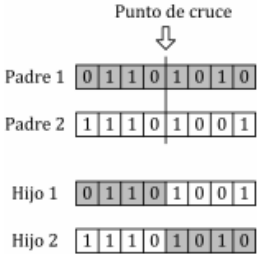

```
# OPERADORES GENETICOS
toolbox.register("mate", tools.cxPartialyMatched)
toolbox.register("mutate", tools.mutShuffleIndexes, indpb=0.2/IND_SIZE)
toolbox.register("select", tools.selTournament, tournsize=3)
```

figura [3]

1.4 Operación de cruce

El operador de cruce permite generar individuos nuevos que heredan características de los padres. El cruce, dentro del AG es manejo mediante el porcentaje de cruce, el cual indica si se va a efectuar la recombinación o no, esto significa que algunos individuos pasarán a la siguiente generación, directamente sin cruzarse: No existe un valor que asegure el buen desempeño de AG, el mejor valor está relacionado con los valores de estreno de los parámetros de algoritmos. Al modificar dicho parámetro, es necesario ajustar el valor para tener una buena función del algoritmo.

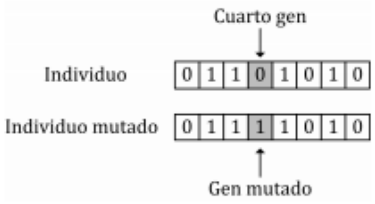
Los individuos seleccionados son recombinados para así reproducir de nuevo los hijos que forman parte de la siguiente generación.




```
def crossover(chromo_a, chromo_b):
    '''
    cruce a nivel de gen, no a nivel de bits dentro de los genes.
    Se supone que ambos cromosomas tienen la misma longitud.
    Retorna (new_chromo_a, new_chromo_b) al realizar el cruce.
    '''
    # elige un punto de cruce aleatorio
    cross_point = int(random.random()) * len(chromo_a)
    new_chromo_a = chromo_a[:cross_point] + chromo_b[cross_point:]
    new_chromo_b = chromo_b[:cross_point] + chromo_a[cross_point:]

    return (new_chromo_a, new_chromo_b)
```

figura[4]

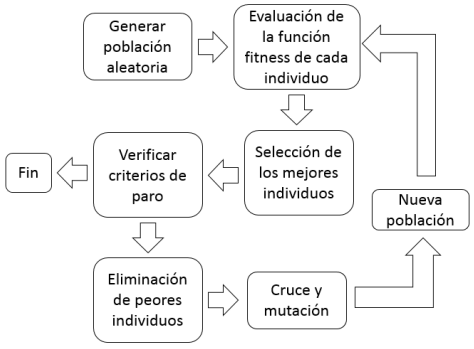
1.5 Operador de Mutación: La mutación se considera un operador básico que proporciona un pequeño elemento de aleatoriedad a los individuos de la población. Si bien sabemos que el operador de cruce es el responsable de efectuar la búsqueda a lo largo del espacio de posibles soluciones, el operador de mutación gana importancia a medida que la población de individuos va convergiendo. El objetivo del operador de mutación es producir nuevas soluciones a partir de la modificación de cierto número de genes de una solución existente con la intención de fomentar la variabilidad dentro de la población. Existen diferentes formas de realizar mutación, la más sencilla es la llamada mutación aleatoria bit a bit que aplica codificación binaria.




```
def mutate(chromo, mutation_rate):
    '''
    Retorna, para el cromosoma dado, un nuevo mutado
    '''
    mutated_chromo = []

    for gene in chromo:
        while True:
            # cree la máscara de bits xor para realizar una mutación como '010001'
            # donde 1 significa que un bit en el gen se mutará (invertirá)
            mutation_bits = sum(
                2 ** x
                for x in range(GENE_BIT_COUNT)
                if random.random() >= mutation_rate
            )
            mutated_gene = gene ^ mutation_bits

            # la mutación debe producir un gen válido

            if mutated_gene in BIT_GENE_MAPPING.keys():
                break

        mutated_chromo.append(mutated_gene)

    return mutated_chromo
```

figura[5]

1.6 Diagrama del algoritmo genético:



figura[6]

ALGORITMO DEL AGENTE VIAJERO CON AG

El problema del agente viajero está dado de la siguiente forma: Sean N ciudades de un territorio, se especifica una ciudad *K*, que será el inicio y fin del recorrido, el objetivo es encontrar una ruta que pase una sola vez por cada una de las ciudades y minimice el costo realizado por el viajero. En este sentido, el problema del TSP también puede ser modelado de esta manera. Así los vértices del grafo representan las ciudades, y los arcos indican los caminos que unen dichas ciudades. Estos

arcos deben de tener un peso, el cual representa la distancia entre una ciudad y la otra. Entonces una solución de este problema se puede representar como una secuencia de N ciudades, donde la ruta comienza y termina en la misma ciudad.

Investigación Chu-Beasley

El Viaje del Caballo de Ajedrez es un problema que ha sido estudiado por siglos, tanto por jugadores de ajedrez como por matemáticos. Se plantea un recorrido del caballo, con su salto característico, de tal forma que recorra todas las casillas del tablero (64 casillas, 8x8) tocando cada una de ellas solo una vez.

El primer dato sobre este problema fue encontrado en un manuscrito arábigo del año 840 (Murray, 1913) [1], donde se detallaban dos recorridos.

El primer análisis matemático fue presentado por Leonhard Euler en Berlín (1759) Otros matemáticos conocidos que lo trataron fueron Taylor, de Moivre y Lagrange.

Más cercanos a nuestra época, McKay [2] y Mordecki [3] usaron potentes mainframes buscando la mayor cantidad posible de recorridos sobre un tablero de 8x8. McKay calculó el total de recorridos cerrados (cuando el caballo completa el circuito, el próximo salto lo lleva a la casilla inicial) en 13.267.364.410.532, en tanto que Mordecki encontró 1.305x1035 recorridos abiertos.

En cuanto a los sistemas de búsqueda, van desde sistemas puramente heurísticos (Warnsdorff 1823), búsqueda en profundidad con retroceso (depth first search/backtracking), redes neuronales, algoritmos genéticos, algoritmos de colonias de hormigas, etc.

Para la resolución de este trabajo se construyó un algoritmo genético basado en el desarrollado por Chu-Beasley.

Como se expresó más arriba, para este trabajo se utilizó el AG de Chu-Beasley. Las diferencias más importantes con los AGs simples son:

-Diversidad. No pueden existir cromosomas iguales en la población.

-En los AGs simples se reemplaza toda o casi toda la población en cada generación. En Chu-Beasley solo es reemplazado un individuo, siempre que el nuevo tenga las características deseadas de diversidad, factibilidad, etc.

-Incorpora un operador de mejora de optimalidad.

III. CONCLUSIONES

- Las pruebas realizadas permitieron identificar que, mediante el manejo de los operadores lógicos como la recombinación y la mutación, las soluciones generadas son mejores, dado que en diferentes casos las respuestas obtenidas con puntos de recombinación de 2, 3 y 5 no generaban modificaciones significativas en las soluciones, pero una vez se realizaba un cambio en los puntos de mutación las soluciones que se obtuvieron sufrieron una variación generando resultados óptimos.
- El algoritmo genético de Chu-Beasley generó soluciones óptimas para las pruebas que se realizaron debido a sus características, las cuales permiten tener un control de la población y de cada uno de los individuos a través de los operadores genéticos.
- Diferentes técnicas de estimación serán comparadas en trabajos futuros. Esto permitirá establecer las ventajas y desventajas de cada una de ellas.
- Este artículo brinda un enfoque alternativo en el estudio de la estimación de estado, pues además de utilizar la metodología clásica, implementa criterios de observabilidad y utiliza la técnica combinatoria de Chu-Beasley.
- El tipo de codificación propuesta permite implementar otras técnicas metaheurísticas tales como la Colonia de Hormigas o Algoritmos Híbridos (BT y AG) que requieren de una adaptación a los métodos de búsqueda.
- Los resultados que arroja la implementación combinada de la teoría clásica WLS (Weighted Least Square) y el algoritmo de optimización combinatorial Chu-Beasley muestran eficiencia en la identificación de mediciones erradas y que sean del tipo iterativo y conformativo.
- El algoritmo genético de Chu-Beasley generó soluciones óptimas para las pruebas que se realizaron debido a sus características, las cuales permiten tener un control de la población y de cada uno de los individuos a través de los operadores genéticos.

RECOMENDACIONES

Tener en cuenta que se deben implementar las librerías necesarias para ejecutar cada aparte del algoritmo

```
import matplotlib.pyplot as plt
import matplotlib.colors as colors
import matplotlib.cm as cmx

import random, operator, time, itertools, math
import numpy

%matplotlib inline
%config InlineBackend.figure_format = 'retina'
plt.rc('text', usetex=True)
plt.rc('font', family='serif')
plt.rcParams['text.latex.preamble'] ='\\usepackage{libertine}\n\\usepackage[utf8]{inputenc}'

import seaborn
seaborn.set(style='whitegrid')
seaborn.set_context('notebook')
```

REFERENCIAS

[1] Darwin,Charles (1859), On the origin of species by means of natural selection, or the preservation of favoured races in the struggle for life (1.ª Edición), Londres: John Murray

[2] Baez, A. (2009) “Problema del agente viajero usando búsqueda tabú”. Proyecto Final, Programación Cientifica, Universidad Autónoma de Nuevo León, Monterrey. Disponible en: http://es.scribd.com/doc/41965624/Busqueda-Tabu-Problema-delagenteviajero-Angels-Baez-Olvera, consultado el 09-Ago-2012, 11:30 a.m.

[3] El problema del agente viajero con búsqueda tabú https://www.scielo.sa.cr/pdf/rmta/v21n1/a08v21n1.pdf

[4]Chu-Beasley https://www.researchgate.net/publication/26544148_Algoritmo_genetico_modificado_Chu-Beasley_aplicado_a_la_Identificacion_de_errores_en_la_estimacion_de_estado_en_sistemas_electricos

[5] Implementación del algoritmo de Chu-beasley para resolver el poblema del agente viajero TSP: http://repositorio.utp.edu.co/dspace/bitstream/handle/11059/5384/0063823A696.pdf?sequence=1