



## Universidad Nacional Autónoma de México

INSTITUTO DE INVESTIGACIONES EN MATEMÁTICAS  
APLICADAS Y SISTEMAS

## PROYECTO COMPUTACIÓN CONCURRENTE

*Computación Concurrente*

Autores:

Vázquez Rojas José David  
Luis Fernando Flores Tiburcio  
José Marcos Yañez Espindola  
Hugo Carlos Moran Peraza

Febrero 2021

# Índice

<b>1. Introducción</b>	<b>2</b>
<b>2. Algoritmos AEB</b>	<b>3</b>
2.1. Programación Evolutiva . . . . .	3
2.2. Estrategias Evolutivas . . . . .	4
2.3. Algoritmos genéticos . . . . .	5
2.3.1. Ventajas Algoritmos Genéticos respecto a los anteriores .	6
<b>3. Algoritmo Genetico Problema del Viajero</b>	<b>7</b>
3.1. Pasos para el algoritmo . . . . .	8
<b>4. Resultados del algoritmo</b>	<b>9</b>
4.1. Ejecución del algoritmo: . . . . .	9
4.2. Trazado de la ruta óptima . . . . .	9
<b>5. Paralelización del algoritmo</b>	<b>9</b>
5.1. Ejecución del algoritmo: . . . . .	10
5.2. Trazado de la ruta óptima . . . . .	11
<b>6. Medidas de rendimiento</b>	<b>11</b>
6.1. Análisis de las medidas de Rendimiento . . . . .	13
<b>7. Conclusiones</b>	<b>14</b>
<b>8. Bibliografía</b>	<b>15</b>

## **1. Introducción**

Con este proyecto aplicaremos los conocimientos adquiridos durante el semestre, para resolver paralelamente un problema en específico.

Empezaremos desarrollando describiendo 3 algoritmos con la finalidad de compararlos y resaltar las ventajas y aplicaciones para cada uno de ellos, de manera que posteriormente escogeremos a implementar uno de ellos, además, profundizaremos en las definiciones necesarias para lograr entender los conceptos que involucran al proyecto, como: algoritmo evolutivo, paralelización, algoritmo genético, etc., entre otros conceptos. Así mismo, una discusión de cómo existen algunos algoritmos que tradicionalmente se pueden ver de manera secuencial junto con su solución, pero gracias al impacto de la concurrencia, se pueden resolver con mayor eficacia.

Una vez sentada la base del proyecto, pasaremos a definir el problema en sí, explicaremos las justificación del mismo y la forma en la que se tratará de resolver. Posteriormente presentaremos las herramientas que se usarán para llevar a cabo la solución del problema, así como su implementación misma y análisis y justificación de los resultados obtenidos.

Por último presentaremos algunas observaciones durante el desarrollo de la implementación, en esta parte describiremos métricas que nos ayudarán a comprender la utilidad de la paralelización y/o concurrencia, y también presentaremos los resultados finales y conclusiones.

## 2. Algoritmos AEB

### 2.1. Programación Evolutiva

Alrededor de los años 60's, Lawrence J. Fogel una técnica denominada "Programación Evolutiva", en la cual la inteligencia se ve como un comportamiento adaptativo. La Programación Evolutiva enfatiza los nexos de comportamiento entre padres e hijos, en vez de buscar emular operadores genéticos específicos (como en el caso de los Algoritmos Genéticos).[1] El algoritmo básico de la Programación Evolutiva es el siguiente:

- Generar aleatoriamente una población inicial.
- Se aplica mutación.
- Se calcula la aptitud de cada hijo y se usa un proceso de selección mediante torneo (normalmente estocástico) para determinar cuáles serán las soluciones que se retendrán.

Esta rama se considera que es una abstracción de la evolución al nivel de las especies, por lo que no se requiere el uso de un operador de recombinación (diferentes especies no se pueden cruzar entre sí).

En un autómata pueden ahora aplicarse cinco diferentes tipos de mutaciones: cambiar un símbolo de salida, cambiar una transición, agregar un estado, borrar un estado y cambiar el estado inicial. El objetivo es hacer que el autómata reconozca un cierto conjunto de entradas (o sea, una cierta expresión regular) sin equivocarse ni una sola vez.[4]

Algunas aplicaciones de la Programación Evolutiva son:

- Predicción
- Generalización
- Juegos
- Control Automático
- Problema del viajero
- Planeación de rutas
- Diseño y entrenamiento de redes neuronales
- Reconocimiento de patrones

## 2.2. Estrategias Evolutivas

Las Estrategias Evolutivas fueron desarrolladas en 1964 en Alemania para resolver problemas hidrodinámicos de alto grado de complejidad por un grupo de estudiantes de ingeniería encabezado por Ingo Rechenberg. La versión original (1+1)-EE usaba un solo parente y con él se generaba un solo hijo. Este hijo se mantenía si era mejor que el parente, o de lo contrario se eliminaba (a este tipo de selección se le llama extintiva, porque los peores individuos tienen una probabilidad cero de ser seleccionados)[2]. En la (1+1)-EE, un individuo nuevo es generado usando:

$$\vec{x}^{t+1} = \vec{x} + N(0, \vec{\sigma})$$

donde  $t$  se refiere a la generación (o iteración) en la que nos encontramos, y  $N(0, \sigma)$  es un vector de números Gaussianos independientes con una media cero y desviación estándar  $\sigma$ . Rechenberg introdujo el concepto de población, al proponer una estrategia evolutiva llamada  $(\lambda + 1) - EE$ , en la cual hay  $\lambda$  padres y se genera un solo hijo, el cual puede reemplazar al peor parente de la población (selección extintiva). Schwefel introdujo el uso de múltiples hijos en las denominadas  $(\mu + \lambda) - EEs$  y  $(\mu, \lambda) - EEs$ . La notación se refiere al mecanismo de selección utilizado:

- En el primer caso, los  $\mu$  mejores individuos obtenidos de la unión de padres e hijos sobreviven
- En el segundo caso, sólo los  $\mu$  mejores hijos de la siguiente generación sobreviven.

Rechenberg formuló una regla para ajustar la desviación estándar de forma determinista durante el proceso evolutivo de tal manera que el procedimiento convergiera hacia el óptimo. Esta se conoce como la “regla del éxito 1/5”, que indica que:

La razón entre mutaciones exitosas y el total de mutaciones debe ser 1/5. Si es mayor, entonces debe incrementarse la desviación estándar. Si es menor, entonces debe decrementarse.

Los operadores de recombinación de las Estrategias Evolutivas pueden ser:

- Sexuales: el operador actúa sobre 2 individuos elegidos aleatoriamente de la población de padres.
- Panmíticos: se elige un solo parente al azar, y se mantiene fijo mientras se elige al azar un segundo parente (de entre toda la población) para cada componente de sus vectores.

Algunas aplicaciones de las Estrategias Evolutivas son:

- Problemas de rutas y redes
- Bioquímica
- Óptica
- Diseño en ingeniería
- Magnetismo

### **2.3. Algoritmos genéticos**

Los Algoritmos Genéticos (denominados originalmente “planes reproductivos genéticos”) fueron desarrollados por John H. Holland a principios de los 1960s , motivado por su interés en resolver problemas de aprendizaje de máquina. El algoritmo genético enfatiza la importancia de la crusa sexual (operador principal) sobre el de la mutación (operador secundario) y usa selección probabilística.[3] El algoritmo básico es el siguiente:

- Generar(aleatoriamente) una población inicial.
- Calcular la aptitud de cada individuo.
- Seleccionar (probabilísticamente) con base a la aptitud.
- Aplicar operadores genéticos (cruza y mutación) para generar la siguiente población.
- Ciclar hasta que cierta condición se satisfaga.

Para poder aplicar el algoritmo genético se requiere de los 5 componentes básicos siguientes:

- Una representación de las soluciones potenciales del problema.
- Una forma de crear una población inicial de posibles soluciones(normalmente un proceso aleatorio).
- Una función de evaluación que clasifique las soluciones en términos de su “aptitud”.
- Operadores genéticos que alteren la composición de los hijos que se producirán para las siguientes generaciones (normalmente cruce sexual y mutación).
- Valores para los diferentes parámetros que utiliza el algoritmo genético (tamaño de la población, probabilidad de cruce, probabilidad de mutación, número máximo de generaciones, etc.)

Algunas aplicaciones de los Algoritmos Genéticos son las siguientes:

- Optimización (estructural, de topologías, numérica, combinatoria, etc.)
- Aprendizaje de máquina (sistemas clasificadores)
- Bases de datos (optimización de consultas)
- Reconocimiento de patrones (por ejemplo, imágenes o letras)
- Generación de gramáticas (regulares, libres de contexto, etc.)
- Planeación de movimientos de robots
- Predicción

### **2.3.1. Ventajas Algoritmos Genéticos respecto a los anteriores**

Es importante destacar las diversas ventajas que presenta el uso de técnicas evolutivas para resolver problemas de búsqueda y optimización:

- Simplicidad Conceptual.
- Amplia aplicabilidad.
- Superiores a las técnicas tradicionales en muchos problemas del mundo real.
- Tienen el potencial para incorporar conocimiento sobre el dominio y para hibridarse con otras técnicas de búsqueda/optimización.
- Pueden explotar fácilmente las arquitecturas en paralelo.
- Son robustas a los cambios dinámicos.

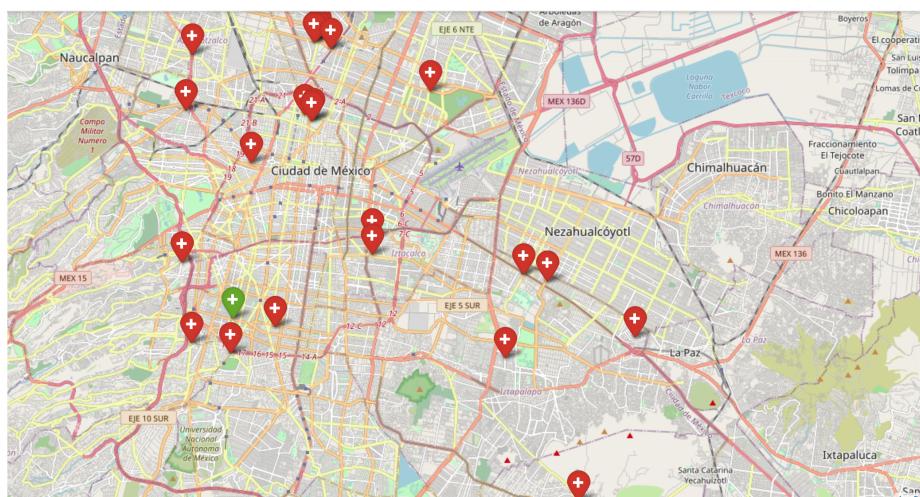
Para finalizar, es importante mencionar que la computación evolutiva, como disciplina de estudio, ha atraído la atención de un número cada vez mayor de investigadores de todo el mundo. Esta popularidad se debe, en gran medida, al enorme éxito que han tenido los algoritmos evolutivos en la solución de problemas del mundo real de gran complejidad. De tal forma, es de esperarse que en los años siguientes el uso de este tipo de técnicas prolifere aún más.

Nótese, sin embargo, que es importante tener en mente que los algoritmos evolutivos son técnicas heurísticas. Por tanto, no garantizan que convergerán al óptimo de un problema dado, aunque en la práctica suelen aproximar razonablemente bien el óptimo de un problema en un tiempo promedio considerablemente menor que los algoritmos deterministas. Esta distinción es importante, pues el papel de las técnicas heurísticas es el de servir normalmente como último recurso para resolver un problema en el que los algoritmos convencionales (típicamente deterministas) no funcionan o tienen un costo computacional prohibitivo. Esto implica que antes de decidir recurrir a los algoritmos evolutivos, debe analizarse la factibilidad de utilizar otro tipo de técnicas. Este paso, que pudiese parecer obvio para muchos, en la práctica suele omitirse en muchos casos y de ahí que exista bastante escepticismo por parte de aquellos que acostumbran a trabajar únicamente con algoritmos deterministas. El uso apropiado y pertinente de los algoritmos evolutivos será sin duda la base de su futuro como alternativa para la solución de problemas complejos y de ahí que se enfatice su importancia. [5]

Dicho esto, al mismo tiempo, fueron las ventajas que ofrece los algoritmos genéticos que nos ayudaron a decidir por explorar e implementar un ejemplo para este proyecto.

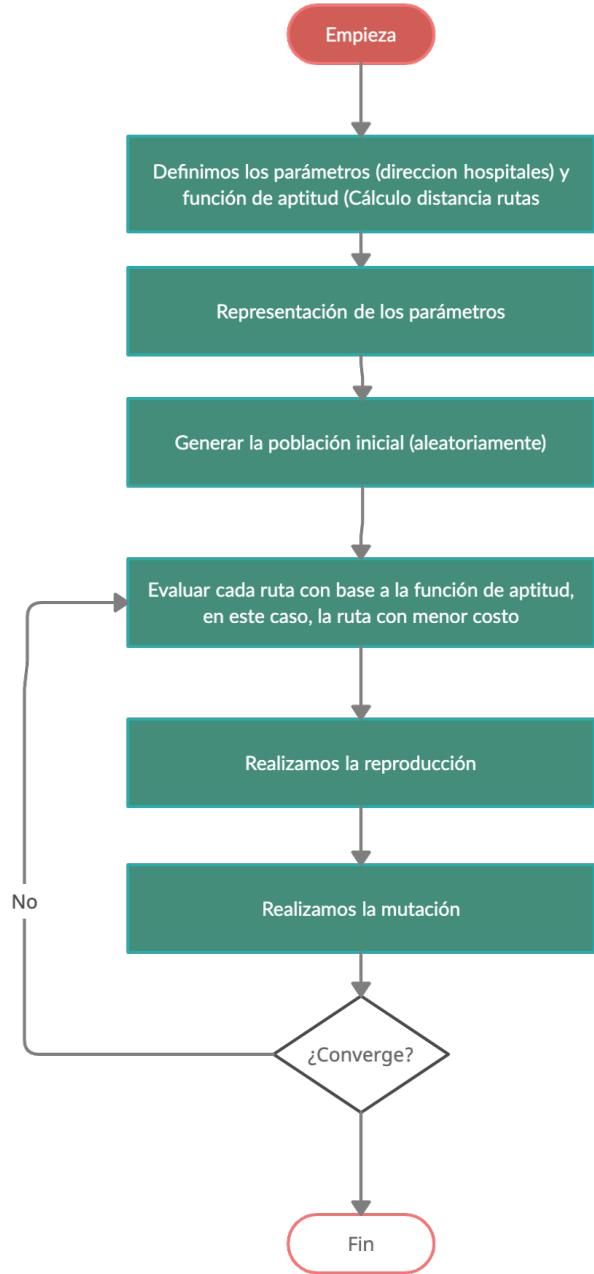
### 3. Algoritmo Genetico Problema del Viajero

A nuestra elección, preferimos los algoritmos genéticos por las ventajas mencionadas anteriores. Ahora, el algoritmo de nuestra preferencia se trata un poco a la tradicional del problema del viajero, pensemos en un distribuidor de recursos para hospitales y vacunas que hace una entrega en un hospital (diremos en CDMX) y debe detenerse en otros hospitales antes de regresar al primero. Para este proyecto, solo nos enfocaremos haciendo la implementación para 23 hospitales.



En retrospectiva, cada gen será una secuencia de hospitales, comenzando y terminando con la primera de ellas (la mencionada de CDMX). La aptitud de cualquier gen dado es su distancia total de ida y vuelta. Comenzamos con genes  $x$  que se generan aleatoriamente. Sería increíble si nos encontráramos con la solución óptima global, pero eso es poco probable; del mismo modo, es poco probable que cualquier gen inicial sea literalmente el peor viaje posible. Sin embargo, algunos genes serán más aptos que otros. Tengamos en cuenta que solo generamos genes al azar una vez. Después de este punto, cada nuevo gen es una función de cruce y mutación. Esto significa heredar hospitales y sus índices respectivos de un gen anterior al azar. Por ejemplo, podríamos heredar “hospital Azcapotzalco” como el quinto elemento de un gen padre y “hospital Universidades” como el sexto del otro gen padre. Tengamos en cuenta que elegimos estos hospitales y sus índices al azar; no tenemos idea de si esto será beneficioso. Simplemente lo hacemos y esperamos lo mejor. Si esto no fue ventajoso, el nuevo gen se desecha y comenzamos de nuevo. Pero si esta selección aleatoria fue beneficiosa, descartamos el más débil de los genes parentales y continuamos.

### 3.1. Pasos para el algoritmo



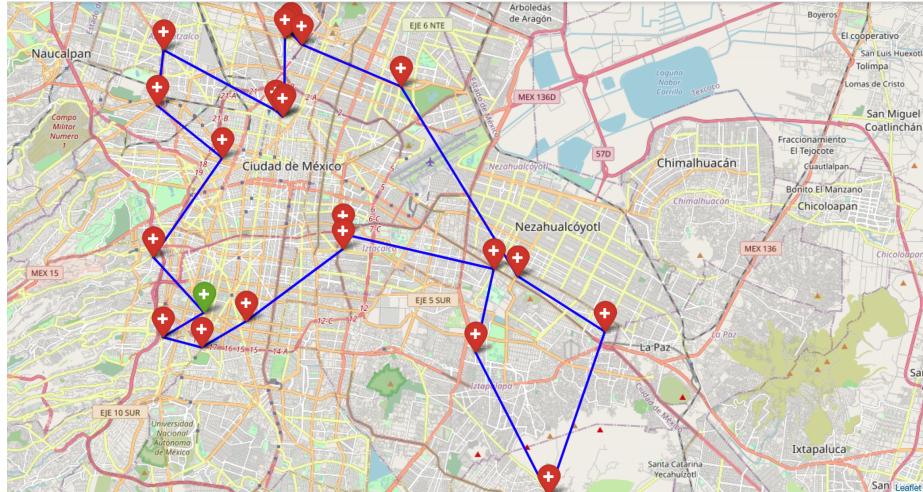
## 4. Resultados del algoritmo

Como podemos observar al ejecutar el algoritmo (secuencial en un inicio) obtenemos una ruta bastante buena ya que paso de una ruta con distancia total de 191.4 kilómetros a una ruta con distancia de 80.15 kilómetros es decir mejoramos la ruta inicial a menos de la mitad. Como podemos observar el algoritmo es bastante bueno ya de por si, ya que para población inicial de 20 y usando 10000 generaciones, hace un tiempo de aproximadamente 20s.

### 4.1. Ejecución del algoritmo:

```
g = GeneticAlgo(hash_map=dist_dict,start='CENTRO MEDICO NACIONAL "20 DE NOVIEMBRE"',mutation_prob=0.25,crossover_prob=0.25,  
population_size=20,steps=15,iterations=10000)  
ruta = g.converge()  
ruta  
  
191.4390206681687 Kilometros  
102.78245519577123 Kilometros  
83.29429480424992 Kilometros  
80.9086000295743 Kilometros  
80.1516183686135 Kilometros
```

### 4.2. Trazado de la ruta óptima



## 5. Paralelización del algoritmo

Primero, paralelizamos la generación de la población inicial usando un Pool de procesos, cada proceso lo que hará es generar una cantidad de genes igual a el  $tamaño\_poblacion / num\_procesos$ .

La siguiente función a paralelizar fue la evaluación de la función fitness para una generación, para ello usamos n procesos para evaluar cada uno de los

genes de una generación, entonces dividimos la generación entre los n procesos y esto nos da la cantidad de genes que evaluará cada proceso. Para esto usamos una Cola y para cada resultado también agregamos el índice del gen al que pertenece dicho resultado, esto debido a que los procesos irán agregando a la cola los resultados de forma asíncrona, entonces al sacar los resultados de la cola no necesariamente saldrán en el orden en el que están los genes en la lista de la generación.

Paralelizamos el cruzado de los genes, para esto tenemos como variables globales index\_map, índices, y to\_visit y haremos la paralelización usando threads para no tener que usar un mecanismo de comunicación entre ellos y así optimizar el tiempo. Sin embargo hay que agregar ciertos bloqueos o locks, ya que los hilos estarán modificando a la vez estas tres variables lo cual nos produce regiones críticas.

En esta parte identificamos 3 regiones críticas:

```
def cross(self,idxs,lock_index_map,lock_indices,to_visit,crossed_count):
    global index_map
    global indices
    global to_visit
    for idx in idxs:
        gene = self.genes[idx]
        for i in range(crossed_count):
            try:
                gene_index = random.choice(indices)
                sample = gene[gene_index]
                if sample in to_visit:
                    # Identificamos 3 regiones críticas en esta función
                    lock_index_map.acquire()
                    index_map[gene_index] = sample # La primera región crítica es al modificar el index_map que es un recurso compartido
                    lock_index_map.release()
                    loc = indices.index(gene_index)
                    lock_indices.acquire()
                    del indices[loc] # La segunda pasa cuando modificamos indices por que también es compartido
                    lock_indices.release()
                    loc = to_visit.index(sample)
                    lock_to_visit.acquire()
                    del to_visit[loc] # Por último aquí tenemos la tercera región crítica
                    lock_to_visit.release()
            except:
                continue
            except:
                pass
```

Como se puede ver, la mayoría de cosas que paralelizamos son 100 % independientes, por lo que esperaríamos que el algoritmo mejore bastante con respecto a la implementación secuencial, sin embargo en los resultados veremos que no fue así. En este caso el algoritmo también funciona de la misma forma que el anterior en cuanto a resultados, sin embargo el tiempo empeoró bastante, por ello en esta implementación paralela solo se probó con una población inicial de 20 y 1000 generaciones.

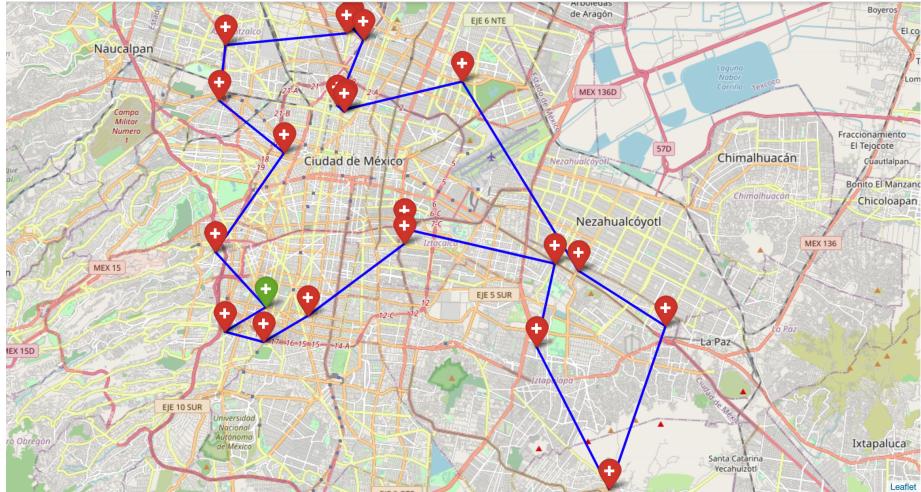
## 5.1. Ejecución del algoritmo:

```
g = GeneticAlgoParallel(hash_map=dist_dict,start='CENTRO MEDICO NACIONAL "20 DE NOVIEMBRE"',mutation_prob=0.25,crossover_prob=0.25,
population_size=20,steps=15,iterations=1000, procesos=1)

[ ] g.converge()

178.78684209250503 kilometros
153.32568607634389 kilometros
138.08756794813632 kilometros
115.08976835211121 kilometros
100.46851517206312 kilometros
85.74854444444444 kilometros
83.135396319529 kilometros
80.13442153888566 kilometros
79.86652149038379 kilometros
79.86652149038379 kilometros
```

## 5.2. Trazado de la ruta óptima



## 6. Medidas de rendimiento

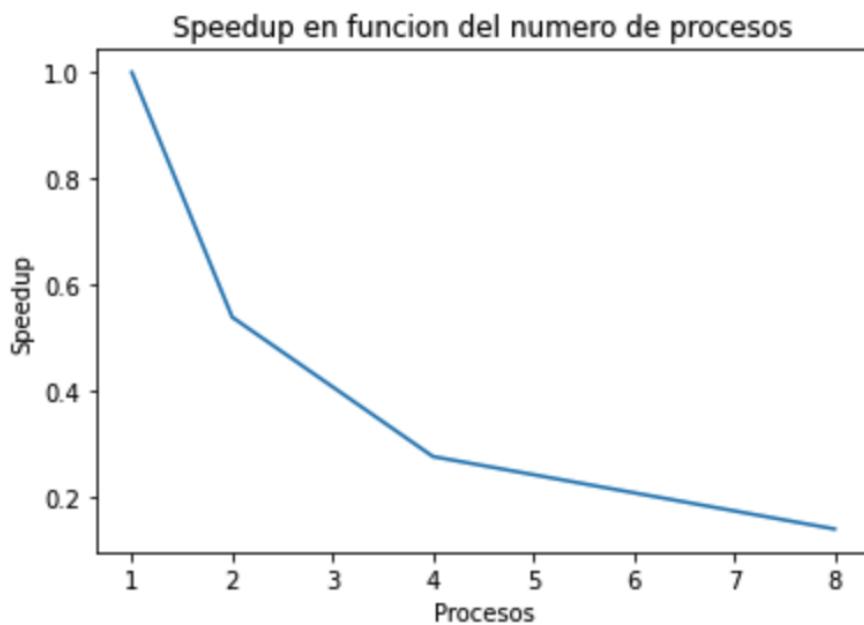
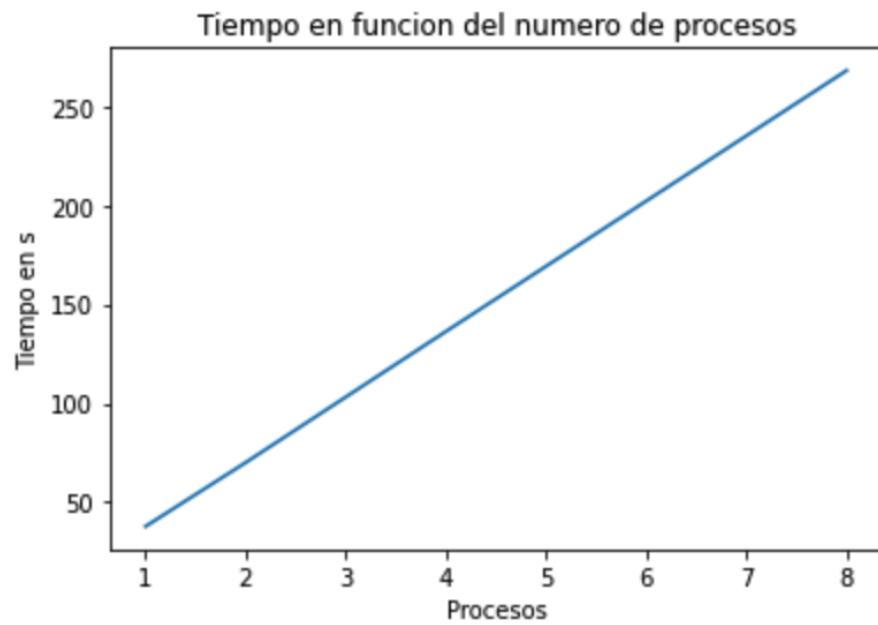
Para medir el rendimiento de nuestra implementación paralela compararemos los tiempos de ejecución, speedup y la eficiencia ejecutando el algoritmo usando 1, 2, 4, 8 procesos.

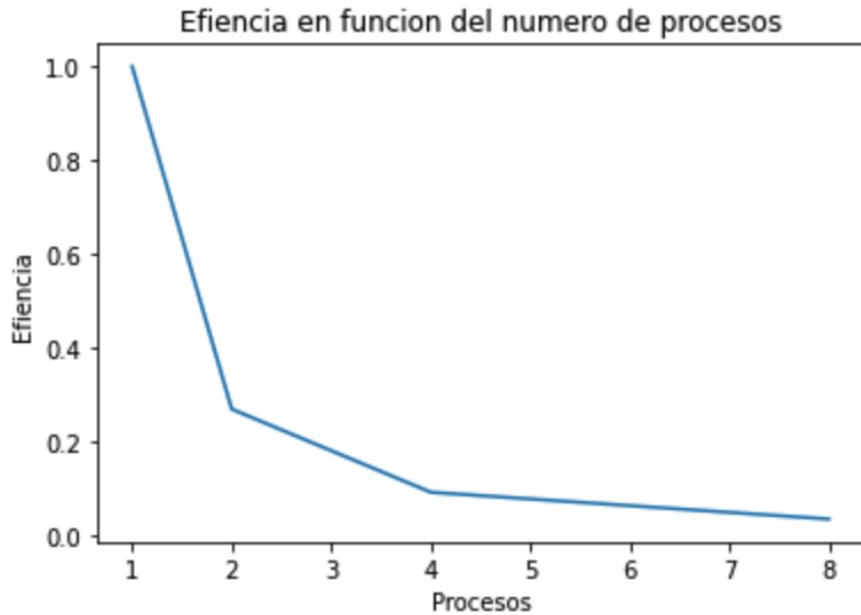
Donde el speedup con  $p$  procesos es:

$$S_p = \frac{T_s}{T_p}$$

Y la eficiencia con  $p$  procesos es:

$$Ef_p = \frac{S_p}{p}$$





### 6.1. Análisis de las medidas de Rendimiento

Como ya habíamos mencionado, al parallelizar el algoritmo, obtenemos peores resultados. Siendo más específicos vemos que el tiempo aumenta linealmente dependiendo del número de procesos, es decir entre más procesos usamos, el tiempo de ejecución es peor, esto es algo raro, ya que la mayoría de las funciones paralelizadas son funciones que se pueden hacer de forma independiente. Incluso haciendo de forma secuencial la única función en la que si usamos métodos de sincronización el tiempo sigue siendo igual de malo.

También vemos que pasa lo mismo con el speedup, en lugar de aumentar conforme crece el número de procesos este disminuye, diciéndonos nuevamente que el aumentar el número de procesos empeora el algoritmo.

Finalmente, la eficiencia de este vemos que cae rápidamente debido a lo anterior mencionado.

## **7. Conclusiones**

En este proyecto implementamos y paralelizamos un algoritmo evolutivo, para ello primero, investigamos y comparamos 3 de estos algoritmos evolutivos y elegimos implementar un algoritmo genético, con el objetivo de resolver un problema de optimización de rutas. En cuanto a la implementación los resultados obtenidos fueron muy buenos ya que logramos obtener una ruta óptima para el problema propuesto, y además el tiempo de ejecución de la implementación secuencial fue muy bueno. Sin embargo, al momento de paralelizar este algoritmo genético encontramos que el tiempo de ejecución empero demasiado, lo cual es demasiado raro, ya que las funciones paralelizadas en el algoritmo son en su mayoría funciones independientes las cuales pueden ejecutarse en paralelo totalmente, por esto ultimo, podemos decir que el algoritmo pasa poco tiempo haciendo comunicación y sincronización entre los procesos, por lo que no podemos asociar el problema del tiempo a una mala selección de las partes a paralelizar, llegando así a la conclusión de que el aumento en el tiempo de ejecución se debe probablemente a que internamente python ya nos estaba optimizando de cierta forma el código, o bien tambien creemos que la forma en la que hicimos l

## **8. Bibliografía**

### **Referencias**

- [1] L. J. Fogel, A. J. Owens and M. J. Walsh, Artificial Intelligence through Simulated Evolution., John Wiley and Sons, Inc., New York, 1966.
- [2] Peterson and J. K. Russell, eds., Purposive Systems, Spartan Books, Washington D.C.
- [3] D. E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley Publishing Co., Reading, Massachussets, 1989.
- [4] José Alberto García Gutiérrez,ANÁLISIS E IMPLEMENTACIÓN DE ALGORITMOS EVOLUTIVOS PARA LA OPTIMIZACIÓN DE SIMULACIONES EN INGENIERÍA CIVIL. ESCUELA UNIVERSITARIA POLITÉCNICA, 2016.
- [5] Enrique Alba Torres,Análisis y Diseño de Algoritmos Genéticos Paralelos Distribuidos.Departamento de Lenguajes y Ciencias de la Computación.Málaga, Febrero de 1999.