El Problema del Viajante: la supervivencia del más errante.

> *Dada una lista de ciudades y las distancias entre cada par de ellas, ¿cuál es la ruta más corta posible que visita cada ciudad exactamente una vez y al finalizar regresa a la ciudad origen?*

Esa es la pregunta con la que nos enfrenta el Problema del Viajante (también llamado *TSP*, por sus siglas en inglés).

Este problema, que a primera vista podría parecer simple, en realidad es uno de los desafíos más intrigantes y complejos en el campo de la optimización. Pertenece a la categoría de los NP-Hard, lo que significa que no existe una solución eficiente (acotada en tiempo polinomial) conocida para resolver todos los casos posibles.

Lo interesante del TSP es que no se queda solo en el ámbito teórico. Representa una gran variedad de problemas de optimización en situaciones reales, como la planificación de rutas y el diseño de circuitos, entre otros. Es por eso que los algoritmos que se desarrollan para resolverlo no solo sirven para este problema en particular, sino que pueden aplicarse en muchos otros contextos prácticos, lo que los convierte en herramientas enormemente útiles.

A lo largo del tiempo, se han propuesto diferentes enfoques para resolverlo. En este trabajo se utilizará un Algoritmo Evolutivo, inspirado en el proceso de selección natural. Estos algoritmos son bastante poderosos, ya que exploran el espacio de soluciones de manera eficiente y, aunque no siempre garantizan una solución óptima, suelen encontrar muy buenos resultados en tiempos razonables.

Características del problema

NOMBRE	Travelling Salesman Problem (TSP) Asimétrico
TIPO	Problema de optimización, complejidad NP-Hard.
INPUT	Información de **N** ciudades. Para cada una se tiene el costo de ir de manera directa a las otras **N**-1. Asimetría: el costo del arco **A** → **B** no es necesariamente el mismo que el de **B** → **A**.
OUTPUT	Idealmente, la configuración de la ruta más corta que recorra a todas las ciudades una sola vez y luego retorne a la ciudad de origen.
MODELO	Las características del modelo serán definidas con mayor detalle a través del informe.

Diseño del Modelo: un enfoque evolutivo

El diseño de un algoritmo evolutivo para abordar el TSP asimétrico requiere definir cuidadosamente sus componentes esenciales, ya que cada uno influye en la calidad de las soluciones y en la eficiencia del proceso de optimización.

Los objetivos generales de un buen diseño incluyen:

- Promover la **diversidad** dentro de la población para explorar diferentes regiones del espacio de búsqueda y evitar la convergencia temprana, que puede llevar a soluciones subóptimas.
- Lograr un balance entre la **exploración** de nuevas áreas del espacio de soluciones y la **explotación** de las soluciones prometedoras, asegurando que el algoritmo progrese hacia mejores resultados.
- Prevenir el **estancamiento evolutivo**, conservar las soluciones de alta calidad encontradas y evitar que el desempeño del algoritmo dependa excesivamente de parámetros específicos.

Lograr estos objetivos implica tomar decisiones cuidadosas sobre el diseño de los componentes del algoritmo, desde la funcionalidad de los operadores hasta la elección de los valores para los distintos parámetros que recibe.

A continuación, se detalla el rol de cada uno y su contribución para lograr un equilibrio entre diversidad, exploración y explotación:

1️⃣ **Representación o codificación de las soluciones.**

- ¿Cómo se representa el genotipo de los individuos dentro de la población de soluciones?
- ¿Qué tipo de estructura se debe utilizar para asegurar que las soluciones sean manipulables y eficientes durante el proceso?
- ¿Qué restricciones o condiciones deben cumplirse para garantizar que las soluciones codificadas sean válidas y representen correctamente el espacio de búsqueda del problema?

2️⃣ **Función de evaluación (Fitness).**

- ¿Bajo qué criterio podemos decir que una solución es buena o mala?
- ¿Cómo se mide el desempeño de una solución en relación con el objetivo del problema?

3️⃣ **Generación 0.**

- ¿Cómo se genera la primera población o "generación 0"? ¿Qué criterios o estrategias se utilizan para crear las soluciones iniciales?
- ¿Es posible continuar la búsqueda en una ejecución posterior si el algoritmo se detiene antes de alcanzar una solución óptima?
- ¿Cómo se garantiza que la población inicial sea representativa del espacio de soluciones para dar inicio a una búsqueda efectiva?

4️⃣ Selección de padres.

- ¿Qué mecanismo o distribución probabilística se emplea para seleccionar a los padres durante el proceso de cruce?
- ¿Cómo se garantiza que los padres seleccionados sean representativos de la población y contribuyan a mejorar la diversidad genética?
- ¿Qué estrategias se implementan para evitar el elitismo excesivo y controlar los niveles de presión selectiva?

5️⃣ Selección de sobrevivientes.

- ¿Cómo se eligen los individuos que sobreviven para la siguiente generación?
- ¿Qué mecanismos o estrategias se utilizan para asegurar que las mejores soluciones se mantengan en la población sin perder diversidad?
- ¿Qué métodos se emplean para evitar la pérdida de buena información genética debido a la eliminación de individuos con alto fitness?

6️⃣ Operadores de cruce.

- ¿Qué operadores de cruce se emplean para combinar los genotipos de los padres seleccionados y generar nuevos individuos?
- ¿Cómo se seleccionan los puntos de cruce y qué estrategias se utilizan para evitar la pérdida de diversidad en la población?
- ¿Qué beneficios aportan los distintos tipos de operadores de cruce en términos de exploración y explotación del espacio de soluciones?

7️⃣ **Operadores de mutación.**

- ¿Qué operadores de mutación se utilizan para introducir variaciones en los individuos y evitar la convergencia prematura?
- ¿Cómo se eligen los puntos de mutación y qué criterios se utilizan para asegurar que las mutaciones sean efectivas sin dañar la calidad de la solución?
- ¿Qué impacto tiene la mutación en la diversidad genética de la población y en la capacidad del algoritmo para explorar nuevas regiones del espacio de soluciones?

8️⃣ **Parámetros del algoritmo + análisis de valores.**

- ¿Qué valores sugeridos para los parámetros clave del algoritmo (como tasa de cruce, tasa de mutación y tamaño de la población) podrían ser más efectivos para este caso específico?
- ¿Cómo influyen estos valores en el comportamiento del algoritmo y su capacidad para encontrar soluciones de calidad?
- ¿Qué consideraciones se deben tener al ajustar estos parámetros para evitar sobreajuste o una convergencia prematura?

Representación o codificación de las soluciones

Una solución se considera válida en el espacio de soluciones si y solo si cumple con las siguientes condiciones:

1. La ruta comienza y termina en la misma ciudad.
2. El viajante debe visitar **todas** las ciudades exactamente una vez, con la excepción de la ciudad de origen, que se visita dos veces.
3. Como resultado de lo anterior, el viajante no puede recorrer un trayecto (es decir, un camino de una ciudad a otra) más de una vez.

Con base en estas condiciones, se propone utilizar un string/arreglo de enteros de longitud **N + 1**, donde **N** es la cantidad de ciudades en la instancia del problema. Las características de una solución válida, cuyo genotipo se representa de la manera propuesta, cumple que:

- El string posee enteros pertenecientes al rango $[0, N)$. Cada entero se mapea unívocamente a una de las ciudades.
- La primera y última posición de la representación se reserva para la ciudad origen/pivote. Este es el único elemento que puede repetirse en la representación y, además, es el único elemento fijo, lo que lo hace “inmune” a las variaciones por mutación o cruce.
- El resto de las ciudades solo pueden estar presentes una única vez, y pueden ser afectadas por los mecanismos evolutivos en cualquiera de las iteraciones.

¿Por qué enteros en lugar de strings?

Aunque podría parecer más intuitivo utilizar los nombres de las ciudades en lugar de enteros, se optó por emplear identificadores numéricos por razones de eficiencia. Los enteros suelen ocupar menos espacio de almacenamiento en comparación con los strings, lo que optimiza tanto el uso de memoria como el rendimiento del algoritmo. Además, el uso de números elimina el inconveniente de que varias ciudades puedan compartir el mismo nombre, lo que podría generar ambigüedades en la representación.

¿Es relevante la ciudad que se tome como pivote?

Dado que todas las soluciones al problema son circulares, es decir, el camino del viajante comienza y termina en la misma ciudad, no tiene sentido generar soluciones que varíen en cuanto a cuál sea la ciudad pivote. Lo que realmente impacta sobre la función de fitness es la sumatoria de los costos de los arcos, sin importar cuál sea el origen. Es decir, si la secuencia de vecinos es la misma, la solución es equivalente.

Por ejemplo, en una instancia del problema con N=3, las rutas `<1>-<2>-<3>-<1>` y `<2>-<3>-<1>-<2>` tienen el mismo costo, al igual que `<3>-<1>-<2>-<3>`. Las tres representaciones corresponden a la misma solución en términos del problema que abordamos.

Esta propiedad resulta beneficiosa, ya que evita la generación de soluciones "circularmente desplazadas" de una misma ruta, lo que disminuye considerablemente el espacio de búsqueda y mejora la eficiencia del algoritmo.

> **Entonces… ¿cuál es el tamaño del espacio de soluciones en función de N?**
>
> Dado que se tiene una secuencia de $N + 1$ enteros, pero el primero y el último son fijos (correspondientes a la ciudad de origen), el número de posibles soluciones se reduce a las permutaciones de los $N - 1$ elementos intermedios.
>
> Por lo tanto, el tamaño del espacio de soluciones es: $T(N) = (N - 1)!$

NOTA: En la implementación, se puede omitir el primer y último elemento de la representación (que corresponden a la ciudad de origen) para reducir el tamaño de la solución. Esto puede mejorar la eficiencia tanto en términos de tiempo como de memoria, ya que la ciudad de origen siempre está fija y no necesita ser almacenada ni manipulada en los mecanismos de cruce y mutación.

Función de evaluación (Fitness)

El fitness de una solución se calcula como la sumatoria de los costos asociados a los arcos que la componen. Este costo total representa la calidad de la solución, y la situación ideal es aquella en la que el costo es lo más bajo posible (aunque nunca puede ser cero, dado que siempre hay un costo asociado a cada trayecto).

Para facilitar la interpretación del progreso del algoritmo, se transforma el problema de minimización en uno de maximización tomando la inversa del fitness. De esta manera, la solución ideal, cuyo costo tiende a cero, provoca que su fitness tienda a infinito. Este cambio es trivial y permite una visualización más intuitiva del rendimiento del algoritmo.

La fórmula para calcular el fitness es la siguiente:

$$fitness = f(\overline{x}) = 1/costo(\overline{x}) = \frac{1}{\sum_{i=0}^{N} C(\overline{x}_i, \overline{x}_{i+1})}$$

- $\overline{x}$ representa la secuencia de ciudades en la solución.
- $C(x, y)$ es la función que calcula el costo de ir de la ciudad x a la ciudad y.

Generación Cero

La población inicial se generará de manera aleatoria, asegurando que las soluciones sean diversas desde el inicio del proceso evolutivo. El tamaño de la población será fijo, determinado por uno de los parámetros de entrada del algoritmo.

Sin embargo, una de las ventajas de los algoritmos evolutivos es que su naturaleza permite interrumpir la ejecución y retomar el proceso desde el punto donde se dejó, pudiendo extraer la mejor solución encontrada hasta el momento. Esta característica es especialmente útil, ya que facilita probar distintas variantes del algoritmo dentro de una misma ejecución sin necesidad de

reiniciar el proceso desde cero. Esto contrasta con otros enfoques, como Backtracking, que requieren volver a ejecutar el proceso desde el inicio cada vez que se interrumpe.

Se presentan entonces dos alternativas:

A. Iniciar el entrenamiento desde cero, generando una población de manera aleatoria.

B. Retomar un entrenamiento previo, utilizando la última generación - o incluso generaciones anteriores, ¿por qué no? - como punto de partida.

Selección de padres

Tanto el mecanismo de selección de padres como el de sobrevivientes juegan un papel crucial en la preservación de la diversidad dentro de la población. Si no utilizáramos probabilidades y seleccionáramos de manera estática a los mejores individuos para que se conviertan en padres de la siguiente generación, reemplazando a los peores ejemplares, estaríamos perdiendo los beneficios clave de los algoritmos evolutivos. Este enfoque podría llevar a una rápida convergencia al dar con un óptimo local, limitando la exploración del espacio de soluciones y reduciendo así la efectividad del algoritmo.

Debe garantizarse que:

- Las soluciones de alta calidad tengan más chances de transformarse en padres que las soluciones de baja calidad.
- Aún la peor solución en la población tenga alguna chance de transformarse en padre.

Se proponen dos mecanismos que definen a las probabilidades de que un individuo se convierta en padre en función del fitness de la solución que representa:

A. Selección basada en el Ranking con mapeo lineal:

Define a las probabilidades de selección en base al fitness relativo de los individuos, en lugar de basarse solo en el fitness absoluto. Mantiene una presión selectiva constante.

[1] Se ordena a las soluciones en un ranking, de menor a mayor, según su fitness.

[2] Asigna a cada individuo una probabilidad de selección de acuerdo a su posición.

$$P_{sel}(i) = \frac{2-s}{T_{poblacion}} + \frac{2 \cdot i \cdot (s-1)}{\lambda \cdot (\lambda - 1)}$$

- i es la posición en el ranking.
- λ es la cantidad total de posiciones en el ranking ($T_{poblacion}$).
- s es un parámetro en el rango (1,2] que controla la ventaja que tienen los mejores individuos por sobre los peores (regula la presión selectiva).

B. Selección por Torneo:

Selecciona padres mediante una competencia entre un subconjunto aleatorio de individuos de la población. La presión selectiva se ajusta a través del tamaño del torneo kk.

[1] Se seleccionan k individuos de la población al azar (con o sin reemplazo).

[2] El individuo con el mejor fitness dentro del torneo es elegido como padre.

[3] La probabilidad de selección de un individuo depende de:

- El tamaño del torneo (k) → impacta en la presión selectiva.
- La probabilidad p de que el mejor individuo sea seleccionado (normalmente $p = 1$).
- Si los individuos son elegidos con o sin reemplazo → impacta en la presión selectiva.

NOTA: sea cual sea el mecanismo de selección que se elija, la cantidad de padres a extraerse de la población es determinada por un parámetro externo: $P_{padre} \rightarrow PARENT\_RATE$.

Comparativa de los criterios de selección de padres propuestos

Criterio	**Selección basada en el Ranking con mapeo lineal**	**Selección por Torneo**
Fundamento	Asigna probabilidades de selección en función del ranking relativo en la población.	Selecciona padres mediante competencias entre subconjuntos aleatorios de la población.

Criterio	Selección basada en el Ranking con mapeo lineal	Selección por Torneo
Parámetros principales	s: controla la presión selectiva.	k: tamaño del torneo. p: probabilidad de selección del mejor individuo.
Influencia del fitness absoluto	Independiente del fitness absoluto, trabaja con el ranking.	Depende indirectamente del fitness, a través de las comparaciones en los torneos.
Presión selectiva	Controlada por s, permite ajustes finos en la ventaja de los mejores individuos.	Controlada por k, donde un k mayor incrementa la presión selectiva.
Diversidad poblacional	Asegura que todos los individuos tengan una probabilidad positiva de ser seleccionados.	La diversidad depende de k: torneos pequeños favorecen diversidad, grandes lo reducen.
Requerimientos computacionales	Requiere ordenar a la población, lo que lo hace más costoso.	Comparativamente más simple, depende solo de k.
Flexibilidad	Más rígido en la asignación de probabilidades, adecuado para mantener presión constante.	Más flexible, permite ajustar exploración y explotación cambiando k .
Exploración vs explotación	Favorece una explotación moderada, con menos tendencia a convergencias prematuras.	Ajustable: exploración alta con k pequeño, explotación alta con k grande.
Aplicabilidad	Útil en problemas donde la población tiene alta variabilidad en fitness.	Útil en poblaciones más homogéneas o para acelerar la convergencia en etapas avanzadas.

Operadores de cruce

Una vez seleccionados los padres, se procede a realizar los cruzamientos para engendrar nuevas soluciones. Estas nuevas soluciones pueden ser mejores, peores o iguales que sus padres. Pueden sobrevivir a la siguiente generación, o pueden no hacerlo.

En general, la condición de mantener las permutaciones (evitar repetir ciudades) complejiza a los mecanismos de cruzamiento y de mutación, ya que se deben manejar con mayor cuidado.

NOTA: los potenciales padres se encuentran en una zona denominada *mating pool*, donde la extracción de parejas es aleatoria de acuerdo a una distribución uniforme (todos los padres del mating pool tienen igual probabilidad de ser seleccionados).

A continuación se proponen dos mecanismos de cruce que a partir de dos padres crean a dos nuevos individuos:

A. Crossover de un solo punto:

Dos padres → Dos hijos.

[1] Se selecciona un punto de cruce c al azar. $1 \leq c \leq N$.

[2] Para generar al hijo 1:

- Copiar los primeros c valores del padre 1 en las primeras c posiciones del hijo 1.
- Los $N - c$ valores restantes se completan con los valores no presentes aún en el hijo 1, en el orden en que aparecen en el padre 2.

[3] Para generar al hijo 2:

- Se repite el proceso anterior pero invirtiendo los roles.

EJEMPLO

```
N = 10; PIVOTE = 0; C = 3;
P1:  < 0 -- 1 -- 2 - | - 3 -- 4 -- 5 -- 6 -- 7 -- 8 -- 9 -- 0 >

P2:  < 0 -- 2 -- 4 - | - 6 -- 8 -- 9 -- 7 -- 5 -- 3 -- 1 -- 0 >
H1:  < 0 -- 1 -- 2 - | - 4 -- 6 -- 8 -- 9 -- 7 -- 5 -- 3 -- 0 >

H2:  < 0 -- 2 -- 4 - | - 1 -- 3 -- 5 -- 6 -- 7 -- 8 -- 9 -- 0 >
```

B. **HorseRace Ponderado**:

Dos padres $(\overline{x}, \overline{y})$ → Un hijo $(\overline{z})$.

[1] Posición i = i’ = 0. Se parte desde la ciudad pivote $(\overline{x}_0 = \overline{y}_0 = pivote = \overline{z}_0)$.

[2] Se toma el elemento siguiente de ambos padres: $\overline{x}_{i+1}, \overline{y}_{i+1}$.

[3] En función del costo de los arcos para ir de $\overline{z}_{i'}$ a $\overline{x}_{i+1}$ y de $\overline{z}_{i'}$ a $\overline{y}_{i+1}$, se asignan a ambos

elementos valores probabilísticos complementarios (la suma da 1).

- Normalmente:

$$P(\overline{x}_{i+1}) = 1 - \frac{costo(\overline{z}_i,\overline{x}_{i+1})}{costo(\overline{z}_i,\overline{x}_{i+1})+costo(\overline{z}_i,\overline{y}_{i+1})} = \frac{costo(\overline{z}_i,\overline{y}_{i+1})}{costo(\overline{z}_i,\overline{x}_{i+1})+costo(\overline{z}_i,\overline{y}_{i+1})}$$

$$P(\overline{y}_{i+1}) = 1 - \frac{costo(\overline{z}_i,\overline{y}_{i+1})}{costo(\overline{z}_i,\overline{x}_{i+1})+costo(\overline{z}_i,\overline{y}_{i+1})} = \frac{costo(\overline{z}_i,\overline{x}_{i+1})}{costo(\overline{z}_i,\overline{x}_{i+1})+costo(\overline{z}_i,\overline{y}_{i+1})}$$

- Si el elemento $\overline{x}_{i+1}$ ya se insertó y el elemento $\overline{y}_{i+1}$ aún no:
  Se inserta
  $\overline{y}_{i+1}$, se procede al paso [5].
- Si el elemento $\overline{x}_{i+1}$ ya se insertó y el elemento $\overline{x}_{i+1}$ aún no:
  Se inserta
  $\overline{y}_{i+1}$, se procede al paso [5].
- Si ambos elementos ya se insertaron: se incrementa i y se vuelve a [2].

[4] Se realiza un pequeño torneo entre ambos arcos mediante una ruleta binaria.

[5] El ganador del torneo es copiado en el hijo 1, en la posición $\overline{z}_{i'+1}$.

[6] Se incrementan i e i’, de modo que si el valor de i es menor a N, se retorna a [2].
Caso contrario finaliza el algoritmo, completando las posiciones en blanco con los elementos faltantes (si los hubiera) de manera aleatoria.

NOTA: se debe llevar registro de qué elementos ya partenecen al hijo $\overline{w}$, de modo que se pueda controlar que no se repitan ciudades en la solución.

EJEMPLO

```
N = 10;  PIVOTE = 0;  COSTO(A,B) = A + 2*B

F(X) = 1/135  F(Y) = 1/105

X:  < 0 -- 1 -- 2 -- 3 -- 4 -- 5 -- 6 -- 7 -- 8 -- 9 -- 0 >


Y:  < 0 -- 2 -- 4 -- 6 -- 8 -- 9 -- 7 -- 5 -- 3 -- 1 -- 0 >

Z:  < 0 --  --  --  --  --  --  --  --  --  -- 0 >
```

```
P(0:1) = 1 - 2/6 = 2/3;

P(0:2) = 1 - 4/6 = 1/3;

Z:  < 0 --  1  --  --  --  --  --  --  --  --  -- 0 >
```

```
P(1:2) = 1 - 5/14 = 9/14;

P(1:4) = 1 - 9/14 = 5/14;

Z:  < 0 --  1  --  2  --  --  --  --  --  --  --  -- 0 >
```

```
P(2:3) = 1 - 8/22 = 14/22;

  P(2:6) = 1 - 14/22 = 8/22;

Z:  < 0 --  1  --  2  --  6  --  --  --  --  --  --  -- 0 >
```

```
P(6:4) = 1 - 10/32 = 11/16;

  P(6:8) = 1 - 22/32 = 5/16;

Z:  < 0 --  1  --  2  --  6  --  4  --  --  --  --  --  -- 0 >
```

```
P(4:5) = 1 - 14/36 = 11/18;

  P(4:9) = 1 - 22/36 = 7/18;

Z:  < 0 --  1  --  2  --  6  --  4  --  5  --  --  --  --  -- 0 >
```

```
P(5:6) = 0; (6 ya pertenece a Z)

P(5:7) = 1 - 0 = 1;

Z:  < 0 --  1  --  2  --  6  --  4  --  5  --  7  --  --  --  -- 0 >
```

```
P(7:5) = 0; (5 ya pertenece a Z)
```

```
P(7:7) = 0; (7 ya pertenece a Z)

Z:  < 0 --  1  --  2  --  6  --  4  --  5  --  7  --  --  --  -- 0 >
```

```
P(7:8) = 1 - 23/36 = 13/36;

P(7:3) = 1 - 13/36 = 23/26;

Z:  < 0 --  1  --  2  --  6  --  4  --  5  --  7  --  3  --  --  -- 0 >
```

```
P(3:9) = 1 - 21/26 = 5/26;

P(3:1) = 1 - 5/26 = 21/26;

Z:  < 0 --  1  --  2  --  6  --  4  --  5  --  7  --  3  --  1  --  8  -- 0
>

F(Z) = 1/103
```

Comparativa de los mecanismos de cruce propuestos

Aspecto	Crossover de un solo punto	HorseRace ponderado
Número de hijos generados	Genera dos hijos.	Genera un único hijo.
Comportamiento	Conserva secuencias completas de los padres, intercambiándolas en un punto seleccionado al azar.	Mezcla rasgos de los padres según una distribución probabilística derivada del peso de los arcos, donde los arcos “compiten” en torneos por ocupar posiciones en el hijo.
Grado de modificación	Enfoque más conservador: realiza pocas modificaciones porque depende de un único evento estocástico (selección del punto c).	Enfoque más dinámico e informado: introduce variaciones mayores al aplicar decisiones probabilísticas en cada paso del proceso de cruzamiento.
Uso del peso de los arcos	Ignora los pesos de los arcos, ya que el punto de cruce se selecciona al azar.	Considera los pesos de los arcos, lo que puede acelerar la generación de soluciones más eficientes.
Similitud entre padres e hijos	Los hijos guardan un gran parecido con sus padres, evolucionando en un patrón predecible.	Los hijos pueden ser radicalmente diferentes a sus padres, dependiendo de las decisiones tomadas durante el cruzamiento.
Sesgo en la solución (*)	Presenta un sesgo hacia los extremos, ya que tiende a	Puede presentar un sesgo hacia los elementos más cercanos al extremo izquierdo, mientras

Aspecto	Crossover de un solo punto	HorseRace ponderado
	conservar segmentos largos de los padres.	que los del derecho dependen más de las decisiones probabilísticas y el azar.
Impacto en diversidad	Favorece a la diversidad y a la conservación del genotipo de los padres, manteniendo patrones heredados.	Favorece a la diversidad en menor medida y aumenta la velocidad del algoritmo al introducir combinaciones más prometedoras.
Complejidad computacional	Es simple y ligero, fácil de implementar y con baja carga computacional.	Es más complejo y computacionalmente pesado. Requiere cálculos de probabilidades, lo que puede ser demandante para poblaciones grandes.
Exploración vs. explotación	Está orientado a la exploración del espacio de soluciones, buscando nuevas combinaciones entre padres.	Se centra más en la explotación de las soluciones prometedoras, adaptando la mezcla en función de los costos relativos.
Conveniencia de uso	Más adecuado como mecanismo de cruce base para aplicar en etapas tempranas del proceso evolutivo.	Más adecuado para etapas más avanzadas del proceso evolutivo.
Optimización de costos	No optimiza explícitamente las relaciones costo/beneficio de los arcos. Los hijos pueden incluir caminos menos eficientes.	Selecciona arcos en función de costos relativos, favoreciendo caminos con mejores relaciones costo/beneficio y aumentando las posibilidades de generar soluciones prometedoras más rápido.

* El sesgo de HorseRace Ponderado puede evitarse si se toma un enfoque no secuencial a costa de mayor complejidad del algorirmo (requiere adicionalmente búsquedas constantes).

Operadores de mutación

Los operadores de mutación se aplican comúnmente sobre los hijos generados a partir del cruce, y su función principal es introducir cambios aleatorios en las soluciones. Esto no solo ayuda a mantener la diversidad genética de la población, sino que también fomenta la exploración de

nuevas áreas del espacio de soluciones, evitando la convergencia prematura hacia óptimos locales.

Las alternativas son:

A. Intercambio de ciudades:

Sencillamente se toman dos ciudades o posiciones al azar y se las intercambia.

- Introduce cambios en el recorrido de forma sencilla y eficiente.
- Ayuda a la exploración del espacio de soluciones sin romper la permutación.
- Exploración más suave.
- Según k (distancia entre ciudades):

k	arcos “rotos”
0	0
1	3
>1	4

B. **Inversión de Subruta:**

Toma dos ciudades o posiciones al azar e invierte la secuencia de ciudades entre ellas.

- Produce mayores variaciones que la alternativa anterior.
- El problema tratado es asimétrico (el costo de ir de A a B no es necesariamente el mismo que ir de B a A), es por eso que la inversión puede generar cambios sustanciales en el fitness de la solución mutada.
- Exploración más temeraria.
- Según k (distancia entre ciudades):

k	arcos “rotos”
0	0
1	3

>1	k+2

Comparativa de las alternativas propuestas

Aspecto	Intercambio de Ciudades	Inversión de Subruta
Descripción	Intercambia dos posiciones aleatorias de las ciudades.	Invierte la secuencia de ciudades entre dos posiciones aleatorias.
Impacto en la solución	Introduce cambios simples, con un impacto limitado en la estructura de la ruta.	Produce cambios más significativos, alterando potencialmente la ruta de forma más drástica.
Diversidad generada	Genera pequeñas variaciones, útil para mantener estabilidad.	Genera mayores variaciones, favoreciendo la exploración.
Complejidad computacional	Baja complejidad, operación sencilla de realizar.	Algo más compleja debido a la inversión de las subrutas.
Eficiencia	Eficiente en términos de tiempo y espacio.	Más costosa en cuanto a la evaluación de la nueva ruta.
Adaptación al problema asimétrico	No afecta significativamente el costo en problemas asimétricos.	Puede alterar los costos asimétricos de manera más notable.
Propósito principal	Favorece la preservación de la estructura y la exploración suave.	Aumenta la exploración al introducir cambios más disruptivos.
Efecto sobre la convergencia	Puede ser útil para mantener la población variada sin grandes saltos.	Reduce la posibilidad de convergencia prematura al introducir cambios más grandes.

Selección de sobrevivientes

Una vez que se obtienen los hijos generados por los individuos de la generación actual, es necesario determinar cuáles soluciones pasarán a la siguiente generación, asegurando que el tamaño de la población permanezca constante.

Los enfoques más comunes se agrupan en dos categorías principales:

1. **Criterio por edad**:

   Favorece a las nuevas soluciones sobre las antiguas. Este enfoque da prioridad a la exploración, promoviendo la renovación constante de la población. Su desventaja es que puede sacrificar soluciones de alta calidad si estas son desplazadas por nuevas soluciones con peor desempeño.

2. **Criterio por fitness**:

   Selecciona las soluciones en función de su fitness, independientemente de si son nuevas o viejas. Este enfoque, más utilizado, prioriza la explotación al favorecer las soluciones con mejor desempeño.

NOTA: En algunos casos se combinan ambos criterios, asignando un peso relativo a la edad y al fitness, para balancear explotación y exploración. También pueden utilizarse o combinarse otros criterios, como la diversidad poblacional, permitiendo ajustar el algoritmo según preferencias.

Se contrastan a continuación dos alternativas de mecanismos de selección de sobrevivientes:

[A] Steady-State basado en fitness:

Este enfoque es sencillo, ya que reemplaza una cantidad limitada de individuos en cada generación, manteniendo constante el tamaño total de la población. El criterio, en este caso propuesto, se basa exclusivamente en el fitness absoluto de los individuos, priorizando soluciones de mayor calidad al incorporar nuevos individuos más competitivos, reemplazando a aquellos con menor desempeño.

EJEMPLO

Tras cruzamiento: $padres + hijos = T_{poblacion} + x$;

Los peores N padres se reemplazan por los mejores N hijos,
donde
$N \in [1, min(x, T_{poblacion})]$.

NOTA: el parámetro N se obtiene de SS_STEADY_STATE_N, no debe confundirse con el N de la cantidad de ciudades.

[B] Round-robin Tournament:

Se basa en evaluar a cada individuo, enfrentandolo a q oponentes elegidos al azar. Dichos oponentes pueden provenir tanto del conjunto de (potenciales) padres como del conjunto de hijos. Se toma un registro de la cantidad de victorias de cada solución. Finalmente, los $T_{poblacion}$ individuos con mayor tasa de triunfos son seleccionados para pasar a la siguiente generación. En la literatura se recomienda emplear $q = 10$.

Comparativa de los mecanismos de selección de sobrevivientes propuestos

Característica	Steady-State	Round-robin Tournament
Proceso de selección	Reemplaza a los peores individuos con los mejores hijos de la generación.	Cada individuo compite contra un conjunto de oponentes al azar, tanto hijos como padres son elegibles como oponentes.
Criterio de selección	Basado en el **fitness absoluto** de los individuos.	Basado en el número de **victorias** de cada individuo contra sus oponentes (fitness relativo con giro estocástico).
Número de reemplazos por generación	Se reemplazan un número pre-establecido de individuos en cada generación.	No hay un reemplazo directo, se seleccionan los mejores individuos según sus victorias.
Impacto en la población	Mantiene constante el tamaño de la población, reemplazando a los peores.	Mantiene constante el tamaño de la población, seleccionando los$T_{poblacion}$ mejores.
Adaptabilidad	Relativamente simple, pero depende de la cantidad de individuos reemplazados.	Más dinámico y competitivo, ya que cada individuo tiene la oportunidad de enfrentarse a varios.
Recomendación de parámetros	Depende de la cantidad de individuos a reemplazar (usualmente, un pequeño número de los peores).	Se recomienda usar $q = 10$ para el tamaño de torneo, balanceando eficiencia del algoritmo con un costo computacional aceptable.
Computacionalmente eficiente	Relativamente eficiente, ya que solo se requieren comparaciones entre los peores y los mejores.	Extremadamente costoso computacionalmente debido a la necesidad de realizar múltiples enfrentamientos entre individuos.

Variabilidad en la población	No introduce mucha variabilidad, ya que se reemplazan a los peores individuos de forma directa.	Fomenta la variabilidad debido a que incluso las peores soluciones tienen chance de sobrevivir (su suerte recae en la racha de oponentes).

Terminación del algoritmo

Normalmente se elige una conjunción de condiciones a modo de corte del algoritmo. Estas condiciones suelen estar relacionadas con:

- Alcanzar un valor de fitness conocido.
- Alcanzar un número máximo de generaciones.
- Alcanzar un nivel mínimo de diversidad en la población.
- Detección de convergencia a partir de cierto período. A medida que se avanza, la mejora del fitness permanece por debajo de un valor de umbral dado.

El enfoque a aplicar utilizará la siguiente estrategia:

1. Debe alcanzar un número mínimo de generaciones.

   Parámetro: CONVERG_MIN_GENS. Establece el mínimo de generaciones que deben existir antes de finalizar la ejecución, evita la convergencia temprana.

2. A partir de entonces, cada cierta cantidad de generaciones, se medira el progreso del algoritmo. Si este se estanca, según cierto umbral pre-establecido, se finaliza la ejecución.

   Parámetro: CONVERG_DISTANCE_BT_GENS. Establece cada cuantas generaciones se realiza el chequeo de progreso para someterlo al criterio del umbral.

   Parámetro: CONVERG_EPSILON. Establece el umbral de mejora. En combinación con el parámetro anterior, con buenos valores evitan la convergencia temprana.

¿Cómo se verifica la convergencia tras haber ejecutado el mínimo de generaciones?

```
x = CONVERG_DISTANCE_BT_GENS
epsilon = CONVERG_EPSILON

// ejemplo: x = 3 -> ... x o o o x o ...

gen_pivote = gen_actual
best_fitness = gen_pivote.best_fitness()
best_fitness_pivote = gen_pivote.best_fitness()

while (|best_fitness_pivote-best_fitness|>epsilon):
    best_fitness_pivote = gen_pivote.best_fitness()
    best_fitness = best_fitness_pivote
    for generacion in range (gen_pivote + 1, gen_pivote +
x):
        if (generacion.best_fitness() > best_fitness)
            best_fitness = generacion.best_fitness()
    gen_pivote = gen_pivote + x
```

NOTA: la eficiencia de este algoritmo depende en gran medida de la interacción entre los valores de los parámetros indicados en las primeras dos líneas.

Listado de parámetros

INPUT:

- ***N***
  Entero que representa la cantidad de ciudades.

Hay ciertas restricciones para el valor de N:

- $0 \leq N \leq 2$:
  No tiene sentido el problema, ya que no hay trayectos significativos que calcular.
- $2 < N \leq 4$:
  El espacio de soluciones es lo suficientemente pequeño como para calcular todas las permutaciones manualmente o mediante métodos más simples y eficientes.
- $5 \leq N \leq 6$:
  Aunque el espacio de soluciones comienza a crecer, todavía es manejable para enfoques deterministas como el Backtracking, siempre que se optimicen con podas o heurísticas.
- $7 \leq N \leq 10$:
  En este rango, los algoritmos evolutivos (AE) son útiles porque el espacio de soluciones crece rápidamente. En N=7, el espacio de soluciones contiene 720 variantes, mientras que en N=10, el espacio de soluciones posee 362.880 posibles soluciones. Aunque aún pueden aplicarse enfoques deterministas, un AE ofrece mayor flexibilidad y eficiencia para problemas complejos.
- $11 \leq N \leq \infty$:
  Para valores mayores, solo los AE u otros métodos heurísticos son recomendables. El espacio de soluciones se vuelve inabarcable para enfoques exhaustivos, ya que $(N-1)!$ crece extremadamente rápido. Los AE permiten obtener soluciones aproximadas razonables en tiempos controlados.

  NOTA: los límites son estimados, pueden variar según múltiples factores.

- **COSTOS**
  Matriz de números reales que contiene los costos para ir de
  i a j. $i, j \in \mathbb{N} \,/\, 0 < i, j \leq N$.
  Dimensión
  N x **N**.
  No se admiten costos negativos.
  El costo para ir de
  i a j puede o no ser igual que el costo para ir de j a i (TSP Asimétrico).

- **PIVOTE**

  Entero que representa la ciudad de origen, donde comienza y termina el recorrido. No tiene influencia sobre el rendimiento del algoritmo. Se toma 0 por defecto.

CONFIG.:

- **POPULATION_SIZE**

  Entero que determina el tamaño de la población. Dicho tamaño es fijo.

Rango recomendado: $[min(0.05 \cdot (N-1), 100)!, min(0.3 \cdot (N-1)!, 2000)]$

- La población debe ser lo suficientemente grande como para albergar y promover un alto grado de diversidad poblacional.
- La población debe ser lo suficientemente chica como para no albergar a todas las posibles soluciones en una única generación. Si esto fuera posible, no tendría sentido aplicar un algoritmo tan robusto como como este. Como máximo, una población debería poder contener a un tercio de las soluciones posibles.
- Debe setearse un límite superior por cuestiones de costo computacional. Provisionalmente propongo un tope de 2000 soluciones.
- Asimismo, una población mínima de 100 soluciones promete una buena diversidad, especialmente si no se permiten repetidos. Para problemas chicos, este límite puede ser incluso menor.
- La cantidad de soluciones posibles depende de N (cantidad de ciudades), y se calcula como $(N-1)!$. Se toma N=7 como valor base.
- Lo ideal es un tamaño de población que albergue entre el 5% y el 30% de las soluciones posibles, pero si no fuera posible (por el rápido crecimiento del espacio de soluciones), un número entre 100 y 2000 debería funcionar.

$$N = 7: \qquad 36 \leq POPULATION\_SIZE \leq 216$$

$$N = 8: \qquad 100 \leq POPULATION\_SIZE \leq 1512$$

$$N > 9: \qquad 100 \leq POPULATION\_SIZE \leq 2000$$

NOTA: los límites inferior y superior son estimadas y provisionales, dependen de varios factores como la capacidad computacional y las características del problema.

- **PARENT_RATE**

Número real en el rango abierto (0,1) que define qué proporción de la población total

(POPULATION_SIZE), impactando directamente en los algoritmos de selección de padres debe ser seleccionada como padres en cada generación.
Influye directamente en el tamaño del
mating pool y, en consecuencia, en los algoritmos de selección de padres.

$$T(mating\_pool) = \lceil POPULATION\_SIZE \times PARENT\_RATE \rceil$$

Rango recomendado: $[0.1, 0.5]$
→ Valores altos favorecen a la diversidad del algoritmo.
→ Valores demasiado altos podrían diluir la presión selectiva y dificultar la mejora constante de la población, perdiendo parte de la funcionalidad del algoritmo.
→ Valores bajos favorecen una alta selectividad, enfocando el algoritmo en soluciones de mayor calidad desde el inicio.
→ Valores demasiado bajos podrían, además de realentizar al algoritmo, aumentar el riesgo de convergencia prematura y/o fomentar el elitismo extremo (dependiendo de los mecanismos de selección y reemplazo empleados).
→ Se recomienda mantener la cantidad de padres por debajo de la mitad del tamaño de la población y por encima del diezmo de la misma para lograr un buen balance entre exploración y explotación.

- **CHILDREN_RATE**

Número real en el rango abierto (0,1) que define qué proporción de la población total (POPULATION_SIZE) debe ser generada como hijos en cada generación, impactando directamente en los algoritmos de cruzamiento.
Influye en el número total de descendientes producidos en cada ciclo evolutivo y, en consecuencia, en la diversidad genética y la competitividad dentro del mating pool.

$$T(hijos) = \lceil POPULATION\_SIZE \times CHILDREN\_RATE \rceil$$

Rango recomendado: $[0.5, 1.0)$

→ Valores bajos favorecen la explotación de las mejores soluciones actuales, limitando la generación de nuevas soluciones al aumentar la competencia dentro del mating pool.

→ Valores demasiado bajos pueden reducir la capacidad del algoritmo para explorar soluciones alternativas, recayendo en la mutación para mantener la diversidad.

→ Valores altos favorecen a la exploración y a la diversidad poblacional.

→ Valores demasiado altos no impactan negativamente si el mecanismo de reemplazo ha sido bien diseñado, sin embargo debe considerarse el costo computacional de almacenar soluciones poco útiles que no sobreviviran a la siguiente generación.

- **MUTATION_RATE**
  Número real en el rango abierto (0,1) que define qué porcentaje de los hijos generados será sometido a mutaciones en cada generación, impactando directamente en los algoritmos de mutación.
  Influye directamente en el grado de diversidad introducido en cada iteración del algoritmo, ayudando a explorar nuevas áreas del espacio de soluciones.

  T(hijos_mutados)
  =⌈CHILDREN_RATE×POPULATION_SIZE×MUTATION_RATE⌉

Rango recomendado: $[0.1, 0.2]$

→ Valores bajos favorecen la explotación de soluciones de alta calidad al limitar las alteraciones en los descendientes.

→ Valores demasiado bajos limitan la utilidad de la mutación para la exploración, reduciendo la capacidad del algoritmo para salir de óptimos locales o encontrar soluciones mejores.

→ Valores demasiado altos corren riesgo de aleatorizar por demás a las soluciones hijas, eliminando los beneficios obtenidos por las soluciones previamente optimizadas mediante la selección de padres.

→ Mantener un MUTATION_RATE en el rango recomendado asegura que las mutaciones contribuyan significativamente a la diversidad sin perturbar la estabilidad del algoritmo.

OTROS:

- **FITNESS_SCALE**

  Número real positivo en el rango $[1,\infty]$ por el que se multiplica el fitness para escalar el resultado (evita altas precisiones en punto flotante sin afectar las distancias relativas entre los fitness de los individuos).

- **ALLOWS_REP**

  Booleano que determina si pueden existir o no soluciones repetidas en una población. Cuando se setea en FALSE, mejora potencialmente la diversidad de la población reemplazando la solución repetida por una generada de forma aleatoria.
  Es válido solo si el tamaño de la población, para el N de la instancia del problema, permite que no haya repeticiones. De otro modo se toma como TRUE.

SELEC.:

- **PS: Parent Selection**

  Parámetros asociados a la selección de padres:

- **PS_RANKING_S:**

  Parámetro que regula la presión selectiva del algoritmo de selección por ranking. Pertenece al rango (1,2], donde un valor cercano a 1 disminuye la presión selectiva, distribuyendo las probabilidades de selección más uniformemente entre los individuos. Un valor cercano o igual a 2 aumenta significativamente la presión selectiva, favoreciendo a los individuos con mejor ranking.

- **PS_TOURNAMENT_K:**

  Tamaño del torneo en la selección por torneo. Define cuántos individuos serán seleccionados aleatoriamente para competir entre sí, eligiéndose al mejor de ellos como padre. Un valor alto aumenta la presión selectiva, ya que incrementa la probabilidad de que los individuos de mayor aptitud sean seleccionados. Valores bajos favorecen la diversidad al permitir que individuos menos aptos sean seleccionados ocasionalmente.

- **PS_TOURNAMENT_W_REPLACEMENT:**

  Parámetro booleano que indica si en el algoritmo de selección por torneo los individuos seleccionados son reemplazados después de cada selección.

  - **Con reemplazo (True):** Se permite que un mismo individuo sea seleccionado varias veces, lo que puede reducir la diversidad del conjunto de padres seleccionados.
  - **Sin reemplazo (False):** Garantiza que cada individuo sea seleccionado como máximo una vez, aumentando la diversidad en el conjunto de padres pero disminuyendo la probabilidad de elegir múltiples individuos altamente aptos.

- **SS: Survivors Selection**

  Parámetros asociados a la selección de sobrevivientes:

  - **SS_ELITISM_RATE:**

    Proporción del mejor subconjunto de individuos que pasan directamente a la siguiente generación sin ser sometidos a selección. El rango recomendado es [0.01,0.1][0.01, 0.1].

    - **Valores altos:** Aseguran que las mejores soluciones no se pierdan, lo que mejora la convergencia del algoritmo. Sin embargo, podrían reducir la diversidad y favorecer la convergencia prematura.

- **Valores bajos:** Permiten una mayor renovación de la población, favoreciendo la diversidad pero aumentando el riesgo de perder buenas soluciones.

- **SS_STEADY_STATE_N:**

  Número de individuos a reemplazar en cada generación en un modelo de estado estacionario (*steady-state*). Este parámetro controla cuántos individuos nuevos (hijos) entran a la población, manteniendo el resto de los individuos sin cambios.

  - **Valores bajos:** Reducen la tasa de renovación, favoreciendo la estabilidad de la población pero disminuyendo la diversidad.
  - **Valores altos:** Incrementan la tasa de renovación, lo que puede aumentar la diversidad pero también el riesgo de perder soluciones prometedoras.
