
# IMPLEMENTACIÓN METODOS DE BUSQUEDA BIOINSPIRADOS

**CESAR FAWCETT**

**EDINSON CARRASCAL**

**RODRIGO VENEGAS**

**BRANDON BRITO**

**UNIVERSIDAD DEL MAGDALENA**

**IDANIS DIAZ BOLAÑOS**

**DOCENTE**

**INTELIGENCIA ARTIFICIAL**

**SANTA MARTA**

**2023-I**

# PROBLEMATICA DE LA ACTIVIDAD

El Knapsack problem, también conocido como problema de la mochila, es un desafío matemático que busca resolver cómo llenar una mochila con diferentes objetos de distintos tamaños y valores, para obtener el mayor valor posible sin sobrepasar su capacidad máxima. De manera más técnica, consiste en encontrar la mejor combinación de objetos que maximice su valor total, manteniéndose dentro de la capacidad de la mochila. Este problema es de gran utilidad en la planificación logística, como en el caso de la carga de camiones o la distribución de paquetes.

No obstante, el Knapsack problem es un problema complejo que no puede ser resuelto eficientemente en todos los casos. Esto se debe a que el número de posibles combinaciones de objetos aumenta exponencialmente a medida que se agregan más objetos, lo que hace que su complejidad crezca rápidamente. lo que significa que no existe un algoritmo que lo resuelva de manera eficiente en todos los casos. Esto se debe a que el número de posibles combinaciones de objetos que se pueden llevar en la mochila aumenta exponencialmente con el número de objetos, lo que hace que la complejidad del problema crezca de manera muy rápida.

# INTRODUCCION

En este trabajo, se implementarán los métodos bioinspirados como el algoritmo genético, enfriamiento simulado y colonia de hormigas todo esto para resolver el problema de la mochila o (Knapsack problem) en inglés, con el objetivo de comparar eficacias y eficiencias. Se espera que, al finalizar este proyecto, se obtenga una mejor comprensión de cómo funcionan estos métodos de búsqueda bioinspirados y cuál es su aplicabilidad en la resolución de problemas de optimización.
Antes de pasar a explicar sobre las implementaciones cabe recalcar creamos una clase llamada Knapsack que representa el problema de optimización de la mochila.

```python
class Knapsack:
    def __init__(self):
        self.capacity = None
        self.values = []
        self.weights = []
        self.read_knapsack_data_from_csv()

    def read_knapsack_data_from_csv(self):
        with open("input8.csv", newline='') as csvfile:
            csv_reader = csv.reader(csvfile, delimiter=';')
            is_first_row = True
            for row in csv_reader:
                if is_first_row:
                    is_first_row = False
                    continue
                if self.capacity is None:
                    self.capacity = int(row[0])
                else:
                    self.values.append(int(row[1]))
                    self.weights.append(int(row[2]))
```

**Figura 1**. *Clase knapsack para la representación del problema.*

En esta leemos el archivo en formato .csv, y este tiene los datos del problema, en la clase tenemos varias funciones, pero lo que cabe marcar es que es que generamos índices aleatorios para seleccionar los objetos de la solución.

```python
def value(self, solution):
    total_value = 0
    total_weight = 0
    for i in range(len(solution)):
        if solution[i]:
            total_value += self.values[i]
            total_weight += self.weights[i]
    return total_value if total_weight <= self.capacity else 0

def generate_neighbor(self, solution):
    index = random.randint(0, len(solution) - 1)
    neighbor = solution.copy()
    neighbor[index] = 1 - neighbor[index]
    return neighbor
```

**Figura 2.** *Función para crear índices aleatorios.*

## ENFRIAMIENTO SIMULADO

La estrategia de enfriamiento simulado consiste en aceptar soluciones subóptimas en las primeras iteraciones y luego disminuir gradualmente la probabilidad de aceptar soluciones peores a medida que se acerca al final de las iteraciones. Esto se logra mediante el ajuste de una temperatura, que va disminuyendo durante el proceso de búsqueda. El algoritmo de Simulated Annealing es implementado en este código para abordar el problema de la mochila.

- La clase SA (Simulated Annealing) recibe los siguientes parámetros: el problema de la mochila (knapsack), la temperatura inicial (initial_temperature), el factor de enfriamiento (cooling_factor), el número máximo de iteraciones por temperatura (max_iterations) y el número máximo de niveles de temperatura (max_temperature_levels).
- El método main() ejecuta el algoritmo de enfriamiento simulado y devuelve los siguientes resultados:
    - La mejor solución encontrada.
    - La lista de valores de la solución en cada iteración.
    - El tiempo empleado para encontrar la solución.
    - El número de iteraciones para encontrar la mejor solución.
    - El número total de iteraciones realizadas.
- El código también genera una gráfica que muestra cómo evoluciona el valor de la solución a medida que se disminuye la temperatura en el algoritmo de enfriamiento simulado. La gráfica ilustra cómo el valor de la solución de la mochila varía a medida que se hacen cambios en la solución actual y se aceptan soluciones peores según una probabilidad que depende de la temperatura actual.

- El objetivo del algoritmo es encontrar una solución óptima, por lo que se espera que el valor de la solución aumente a medida que el algoritmo avanza y la temperatura disminuye.

## DESCRIPCION DEL ESQUEMA

El algoritmo utiliza un arreglo de bits para representar la mochila, donde cada posición del arreglo indica si un objeto está o no en la mochila. La clase Knapsack maneja la generación de vecinos, el cálculo del valor de la mochila y la selección de objetos para agregar a la misma. La clase SA implementa el algoritmo de enfriamiento simulado para resolver el problema de la mochila. La solución inicial es una mochila vacía y se busca mejorarla mediante iteraciones que disminuyen la temperatura. En cada iteración, se genera un vecino aleatorio y se evalúa si es mejor que la solución actual. Si lo es, se acepta automáticamente. Si no lo es, se evalúa la probabilidad de aceptar el vecino, según la temperatura actual y la diferencia de valor entre la solución actual y el vecino. El algoritmo termina cuando se alcanza el número máximo de iteraciones o el número máximo de niveles de temperatura. La salida del algoritmo incluye la mejor solución encontrada, el valor total de la mochila, el tiempo empleado para encontrar la solución, el número de iteraciones para encontrar la mejor solución y el número total de iteraciones. También se grafica la convergencia del algoritmo en un gráfico que muestra el valor de la mochila en cada nivel de temperatura.

## ESTRATEGIA PARA PRODUCIR POSIBLES SOLUCIONES

La táctica utilizada para crear nuevas posibles soluciones en este algoritmo de enfriamiento simulado se basa en modificar aleatoriamente el estado de un objeto dentro de la solución actual, lo cual implica incluir o excluir el objeto de la mochila. Después, se evalúa si esta modificación produce una solución mejor o peor que la solución actual en términos del valor total de los objetos incluidos en la mochila. Si la solución resulta ser mejor, se acepta como la nueva solución actual. Por el contrario, si la nueva solución es peor, se acepta con una probabilidad que depende de la temperatura actual y de la diferencia de valor entre la nueva y la solución actual.

Este proceso se repite varias veces, evaluando soluciones vecinas y disminuyendo gradualmente la temperatura a medida que se avanza en el algoritmo. El objetivo es explorar diferentes soluciones y aceptar soluciones peores en las primeras etapas del algoritmo para evitar quedar atrapado en un óptimo local, pero disminuir gradualmente la probabilidad de aceptar soluciones peores a medida que se acerca a una solución óptima. La finalidad última es encontrar la mejor solución posible para colocar los

objetos en la mochila, maximizando el valor total de los objetos seleccionados y respetando la capacidad de la mochila.

## FUNCION OBJETIVO

Como anterior mente se ha escrito en este código, la meta es obtener la combinación más valiosa de objetos que quepan en la mochila y respeten su capacidad máxima. Para lograrlo, se utiliza el método de enfriamiento simulado que, mediante la evaluación de la función de valor de la mochila, busca maximizar el valor total de los objetos que se colocan en ella. Para evitar quedar atrapado en un óptimo local, el algoritmo itera a través de diferentes soluciones y acepta soluciones peores con una probabilidad que depende de la temperatura y la diferencia de valor entre la nueva solución y la solución actual. Gradualmente, la temperatura disminuye y se espera que el valor de la solución converja hacia un resultado óptimo. La gráfica de convergencia del algoritmo muestra la función objetivo, que representa cómo varía el valor de la solución al disminuir la temperatura en el algoritmo de enfriamiento simulado.

```python
def main(self):
    n = len(self.knapsack.values)
    current_solution = [0 for _ in range(n)]
    best_solution = current_solution.copy()
    temperature = self.initial_temperature
    current_values = []

    start_time = time.time()
    iteration_count = 0
    for temperature_level in range(self.max_temperature_levels):
        for iteration in range(self.max_iterations):
            neighbor = self.knapsack.generate_neighbor(current_solution)
            delta = self.knapsack.value(neighbor) - self.knapsack.value(current_solution)

            if delta > 0 or random.random() < math.exp(delta / temperature):
                current_solution = neighbor

                if self.knapsack.value(current_solution) > self.knapsack.value(best_solution):
                    best_solution = current_solution.copy()

            current_values.append(self.knapsack.value(current_solution))
            temperature *= self.cooling_factor
            iteration_count += 1
            if iteration_count >= self.max_iterations:
                break

        if iteration_count >= self.max_iterations:
            break

    end_time = time.time()

    time_elapsed = end_time - start_time
    best_solution_iterations = current_values.index(self.knapsack.value(best_solution))
    total_iterations = len(current_values)
    return best_solution, current_values, time_elapsed, best_solution_iterations, total_iterations
```

**Figura 3.** *Función principal*

**Primera Ejecución:**

| | |
|---|---|
| initial_temperature | 2000 |
| cooling_factor | 0.45 |
| max_temperature_levels | 500 |
| max_iterations | 10 |

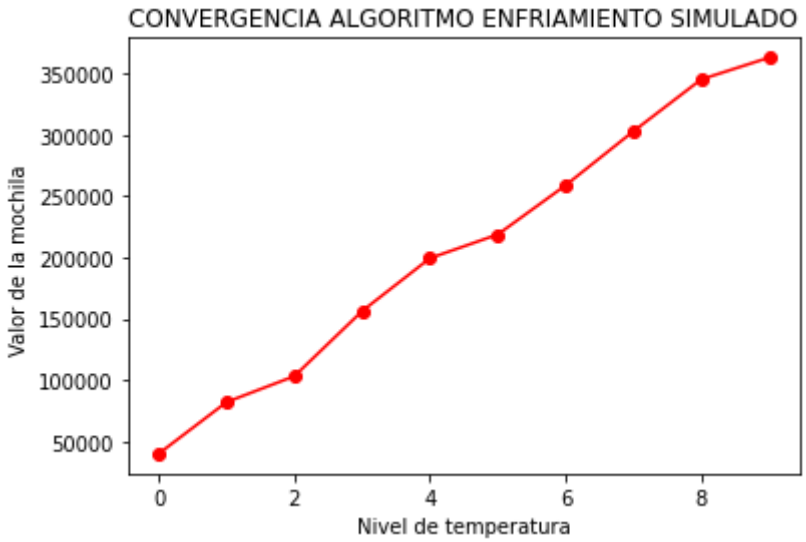


**Figura 4.** *Grafica de convergencia, primera ejecución.*

Valor total de la mochila: 902018
Tiempo empleado para encontrar la solución: 0.00597834587097168 segundos
Número de iteraciones para encontrar la mejor solución: 36
Número total de iteraciones: 100

**Comentario**:

Los resultados obtenidos del algoritmo bioinspirado SA son altamente notables. Se destaca que el tiempo que tardó en encontrar la solución fue mínimo, solo 0.00597834587097168 segundos, lo que sugiere que este algoritmo es altamente eficiente. Asimismo, el número de iteraciones necesarias para alcanzar la mejor solución fue solo de 36, lo cual es excepcionalmente bajo en comparación con otros algoritmos. En cuanto al valor total de la mochila obtenido, el algoritmo logró un valor de 902018, el cual es satisfactorio, pero no del todo bueno. No obstante, es importante mencionar que este valor es inferior a los obtenidos por otros algoritmos.

**Segunda Ejecución:**

| | |
|---|---|
| initial_temperature | 2000 |
| cooling_factor | 0.85 |
| max_temperature_levels | 700 |
| max_iterations | 50 |

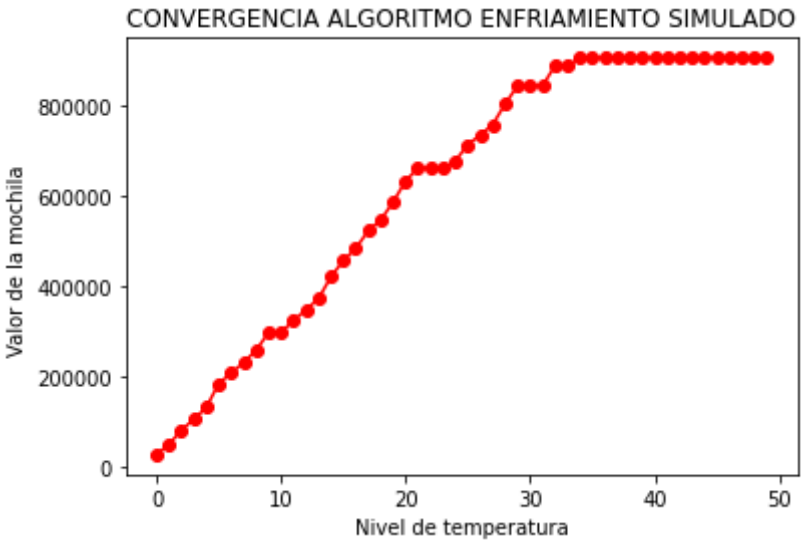


**Figura 5.** *Grafica de convergencia, segunda ejecución.*

Valor total de la mochila: 907012
Tiempo empleado para encontrar la solución: 0.0020017623901367188 segundos
Número de iteraciones para encontrar la mejor solución: 34
Número total de iteraciones: 50

**Comentario**:

El algoritmo bioinspirado SA para esta configuración de parámetros logró un resultado favorable al obtener un valor total de 907012. Esto implica que el algoritmo encontró la combinación óptima de elementos dentro de la limitación de capacidad. Además, el tiempo requerido para encontrar esta solución fue mínimo, con solo 0.0020017623901367188 segundos. Esto indica que el algoritmo es altamente eficiente y capaz de hallar soluciones óptimas de manera rápida. Se encontró la mejor solución después de 34 iteraciones, lo que significa que el algoritmo exploró varias posibilidades antes de conseguir la solución óptima. Por último, el algoritmo se detuvo después de 50 iteraciones, lo que sugiere que el algoritmo fue capaz de encontrar una buena solución en poco tiempo. En conclusión, los resultados del algoritmo bioinspirado SA son altamente prometedores y sugieren que esta técnica podría ser aplicada a otros problemas similares en el futuro.

**Tercera Ejecución:**

| | |
|---|---|
| initial_temperature | 2000 |
| cooling_factor | 0.95 |
| max_temperature_levels | 1000 |
| max_iterations | 100 |

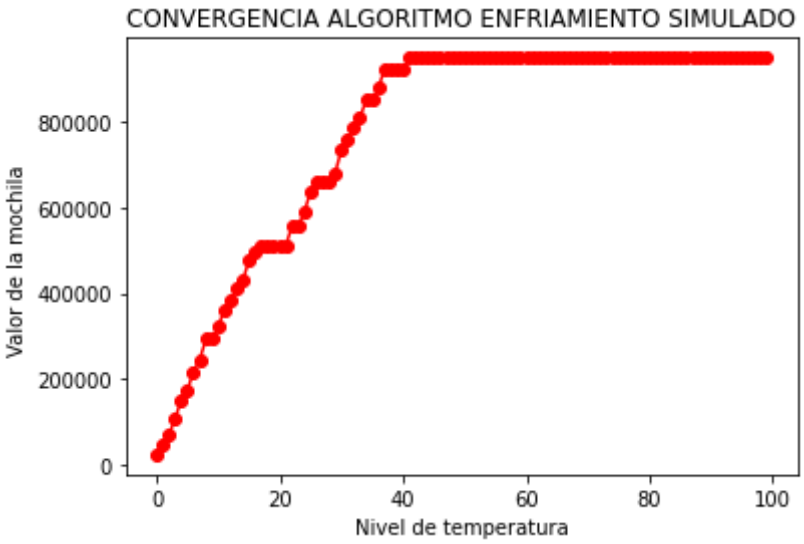


**Figura 6.** *Grafica de convergencia, tercera ejecución.*

Valor total de la mochila: 949197
Tiempo empleado para encontrar la solución: 0.0039975643157958984 segundos
Número de iteraciones para encontrar la mejor solución: 41
Número total de iteraciones: 100

**Comentario**:

La solución obtenida por el algoritmo bioinspirado SA en esta ejecución para el problema de la mochila es altamente satisfactoria, ya que logró encontrar una combinación de elementos que maximizó el valor total de la mochila dentro de las restricciones de capacidad. La rapidez con la que el algoritmo encontró la solución es impresionante, ya que solo tomó 0.0039975643157958984 segundos, lo que demuestra que es altamente eficiente en encontrar soluciones óptimas en poco tiempo. El hecho de que el algoritmo necesitó 41 iteraciones para encontrar la mejor solución indica que exploró varias posibilidades antes de llegar a la solución óptima. Además, el algoritmo se detuvo antes de alcanzar el límite máximo de iteraciones, lo que significa que encontró una solución de alta calidad en un tiempo relativamente corto.

**Conclusión**:

Al comparar los resultados de los tres códigos de algoritmo SA para resolver el problema de la mochila, podemos observar que cada uno presentó diferentes combinaciones de tiempo, iteraciones y valor total de la mochila. El primer código tardó un poco más en encontrar la solución, con un tiempo de 0.00597834587097168 segundos. No obstante, logró encontrar la mejor solución en solo 36 iteraciones, lo que fue más rápido que el tercer código. El número total de iteraciones fue de 100, igual que el tercer código. El segundo código tuvo el tiempo más rápido de 0.0020017623901367188 segundos y encontró la mejor solución en solo 34 iteraciones. El número total de iteraciones fue de 50, el más bajo de los tres códigos. No obstante, el valor total de la mochila fue un poco más bajo en comparación con el tercer código. El tercer código encontró la mochila de mayor valor total entre los tres, con un valor de 949197. El tiempo empleado para encontrar la solución fue de 0.0039975643157958984 segundos, un poco más rápido

que el primer código. Sin embargo, se necesitaron 41 iteraciones para encontrar la mejor solución, el mayor número de iteraciones entre los tres códigos. El número total de iteraciones fue de 100, igual que la primera compilación.

En resumen, cada configuración de parámetros presentó sus propias fortalezas y debilidades en términos de tiempo, número de iteraciones y valor total de la mochila encontrada.

**Tabla de ejecuciones.**

| ejecuciones | tiempo | Valor mochila | Mejor iteración | Total de iteración |
|---|---|---|---|---|
| 1 | 0.00597834587097168 | 902018 | 36 | 100 |
| 2 | 0.0020017623901367188 | 907012 | 34 | 50 |
| 3 | 0.0039975643157958984 | 949197 | 41 | 100 |

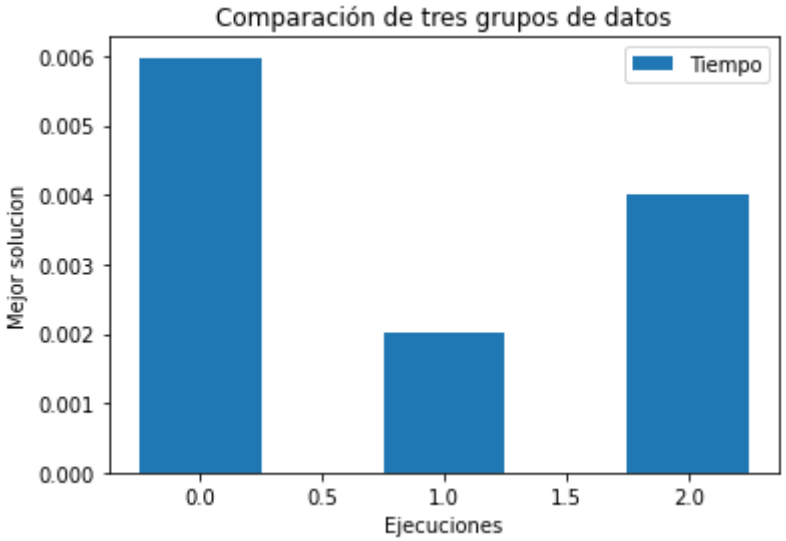


**Figura 7.** *Comparación de los tres grupos de datos.*

# ALGORITMO GENETICO

El algoritmo genético es una técnica de búsqueda bioinspirada que busca solucionar problemas de optimización. Este algoritmo se inspira en la evolución natural y utiliza principios de selección natural, cruzamiento y mutación para encontrar la mejor solución posible a un problema determinado. El proceso del algoritmo comienza con una población inicial de soluciones candidatas, las cuales se representan como cadenas de bits. En cada iteración del algoritmo se evalúa la calidad de cada solución en la población, utilizando una función de evaluación o función de aptitud. Posteriormente,

se seleccionan los individuos más aptos de manera probabilística para reproducirse, y se aplican operadores de cruzamiento y mutación para crear una nueva generación de soluciones. Este proceso se repite durante varias generaciones hasta que se alcance una solución satisfactoria o se alcance el límite de generaciones permitido. La selección de los individuos más aptos se realiza de manera probabilística, dando más oportunidades a aquellos individuos con una mayor aptitud. El operador de cruzamiento mezcla las cadenas de bits de dos soluciones padres para generar una nueva solución hija. Por otro lado, el operador de mutación introduce pequeñas modificaciones aleatorias en la solución. Estos operadores permiten explorar y explotar el espacio de búsqueda.

## DESCRIPCION DEL ESQUEMA

Se ha utilizado un algoritmo genético para resolver el problema, en el cual cada solución es representada por un vector de bits que indica si un elemento está o no en la mochila. La población inicial se genera aleatoriamente y se aplican operadores genéticos como mutación y cruce para generar nuevas soluciones. La selección de los padres se realiza mediante el método de la ruleta, que favorece a las soluciones con mayor valor de la función objetivo. El algoritmo utiliza técnicas de elitismo para preservar las mejores soluciones en la nueva población y evitar la pérdida de información valiosa. Se evalúa la convergencia del algoritmo en cada iteración y se registra el mejor valor obtenido hasta el momento. Al final, se muestra la mejor solución encontrada, el tiempo total empleado y el número de iteraciones realizadas. Se genera una gráfica que muestra la evolución del mejor valor de la función objetivo durante las iteraciones.

## ESTRATEGIA PARA PRODUCIR POSIBLES SOLUCIONES

La técnica empleada en este código para generar soluciones posibles se basa en la implementación de un algoritmo genético. Cada solución posible se representa como un vector de elementos binarios, donde cada bit representa la presencia o ausencia de un elemento específico en una mochila.

En el método create_individual, se genera un individuo aleatorio que representa una posible solución. El método create_population utiliza create_individual para crear una población inicial de soluciones potenciales.

El método fitness evalúa la función objetivo de cada posible solución, en este caso el valor total de la mochila. El método mutate cambia al azar algunos de los bits en un individuo para crear nuevas soluciones en cada iteración. El método crossover combina dos padres seleccionados de la población para generar nuevos individuos.

En el método selection, se emplea el método de la ruleta para seleccionar a los padres con mayor probabilidad de ser seleccionados en función de su función objetivo.

En el método main, se inicializa la población, se evalúa la función objetivo de cada posible solución, se evalúa la convergencia del algoritmo en cada iteración y se registra el mejor valor obtenido hasta el momento. El algoritmo utiliza técnicas de elitismo para mantener las mejores soluciones de la población anterior en la nueva población.

Finalmente, se muestra la mejor solución encontrada, junto con el tiempo total utilizado y el número de iteraciones realizadas. Además, se genera una gráfica que muestra cómo evoluciona el mejor valor de la función objetivo durante las iteraciones.

# FUNCION OBJETIVO

En el código la función objetivo se define en la función fitness. Esta función evalúa la aptitud de un individuo, que es su valor de la función objetivo en el problema de la mochila. La mochila es un problema de optimización en el que se busca maximizar el valor total de los elementos que se pueden poner en una mochila con una capacidad limitada.

La función fitness toma como entrada un individuo, que es una posible solución del problema de la mochila, y devuelve el valor total de la mochila correspondiente a esa solución. En otras palabras, la función fitness calcula el valor total de los elementos que se pueden poner en la mochila con la solución dada.

La aptitud de un individuo es utilizada en el proceso de selección, en el que se eligen los mejores individuos para evolucionar la población hacia soluciones mejores. En cada generación, se seleccionan los mejores individuos, se aplican operaciones de cruzamiento y mutación para crear nuevos individuos, y se reemplazan los peores individuos por los nuevos individuos.

El objetivo del algoritmo genético es encontrar la mejor solución posible al problema de la mochila. Para ello, se define una población inicial de individuos aleatorios y se repite el proceso de selección, cruzamiento y mutación hasta alcanzar un criterio de parada, que en este caso es el número máximo de generaciones.

La función objetivo, es decir, el valor total de la mochila se utiliza para evaluar la calidad de las soluciones y para compararlas entre sí. En última instancia, el objetivo es encontrar la solución con el mayor valor total de la mochila posible. La función fitness es crucial en este proceso, ya que determina la aptitud de los individuos y, por lo tanto, su probabilidad de ser seleccionados y evolucionados en generaciones posteriores.

**Primera Ejecución:**

| | |
|---|---|
| population_size | 100 |
| mutation_rate | 0.65 |
| crossover_rate | 0.45 |
| max_generations | 50 |
| elitism_rate | 0.80 |

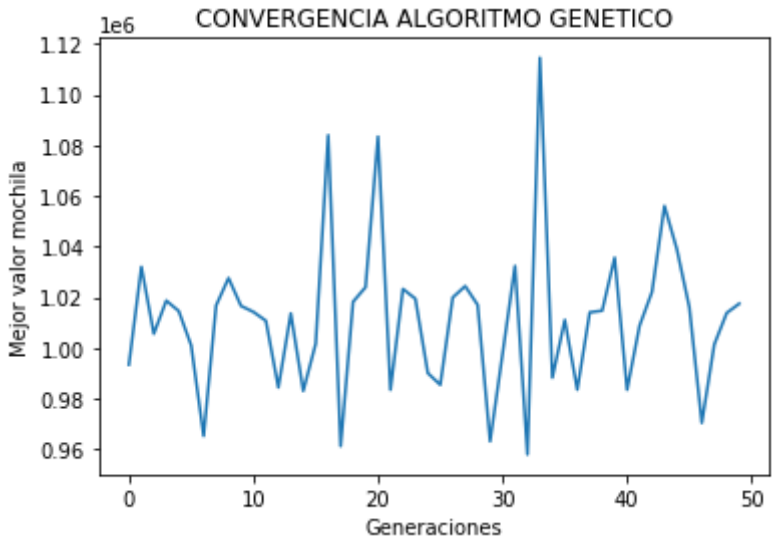


**Figura 8.** *Grafica convergencia, primera ejecución.*

Tiempo empleado para encontrar la solución: 3.3111231327056885 segundos
Número de iteraciones para encontrar la mejor solución: 33
Valor total de la mochila: 1114476
Número total de iteraciones: 5000

**Comentario:**

La solución encontrada con el método de búsqueda algoritmo genético en esta primera ejecución y con los parámetros vistos anteriormente, logró encontrar una solución en un tiempo de ejecución de 3.3111231327056885 segundos, tiempo el cual fue logrado en 33 iteraciones para poder encontrar la mejor solución posible, llego así a un valor de 1114476. Estos resultados sugieren que el algoritmo genético implementado fue eficiente en la resolución del problema de la mochila.

**Segunda Ejecución:**

| | |
|---|---|
| population_size | 300 |
| mutation_rate | 0.80 |
| crossover_rate | 0.40 |
| max_generations | 100 |
| elitism_rate | 0.60 |

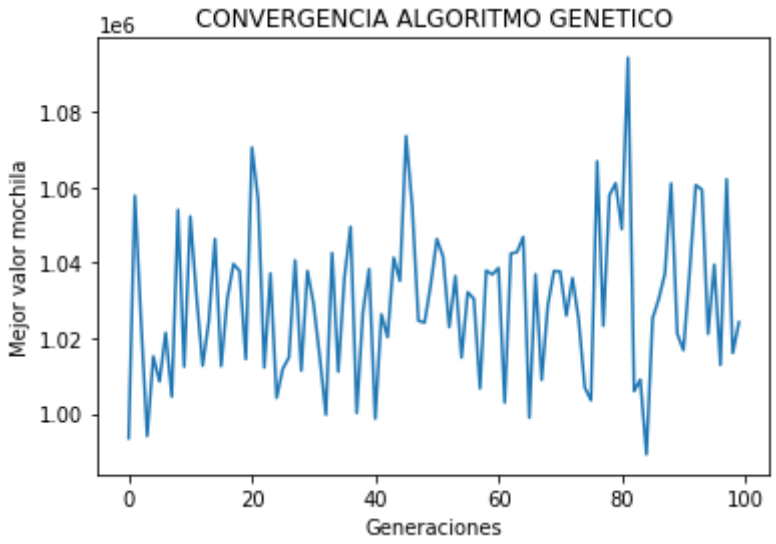


**Figura 9.** *Grafica convergencia, segunda ejecución.*

Tiempo empleado para encontrar la solución: 59.18733286857605 segundos
Número de iteraciones para encontrar la mejor solución: 81
Valor total de la mochila: 1094259
Número total de iteraciones: 30000

**Comentario:**

La solución encontrada con el método de búsqueda algoritmo genético en esta segunda ejecución y con los parámetros vistos anteriormente, logró encontrar una solución en un tiempo de ejecución de 59.19 segundos, con un número de iteraciones para encontrar la mejor solución de 81 y un valor total de la mochila de 1094259. Sin embargo, en comparación con la ejecución anterior con parámetros diferentes, el tiempo de ejecución fue mucho mayor en esta ejecución, lo que indica que el algoritmo pudo ser menos eficiente en términos de tiempo en esta configuración específica. Es importante tener en cuenta que la eficiencia y efectividad de un algoritmo genético depende en gran medida de la configuración de sus parámetros y del problema específico que esté siendo resuelto, por lo que es necesario ajustar y optimizar los parámetros para obtener los mejores resultados en cada caso.

**Tercera Ejecución:**

| | |
|---|---|
| population_size | 400 |
| mutation_rate | 0.30 |
| crossover_rate | 0.70 |
| max_generations | 150 |
| elitism_rate | 0.80 |

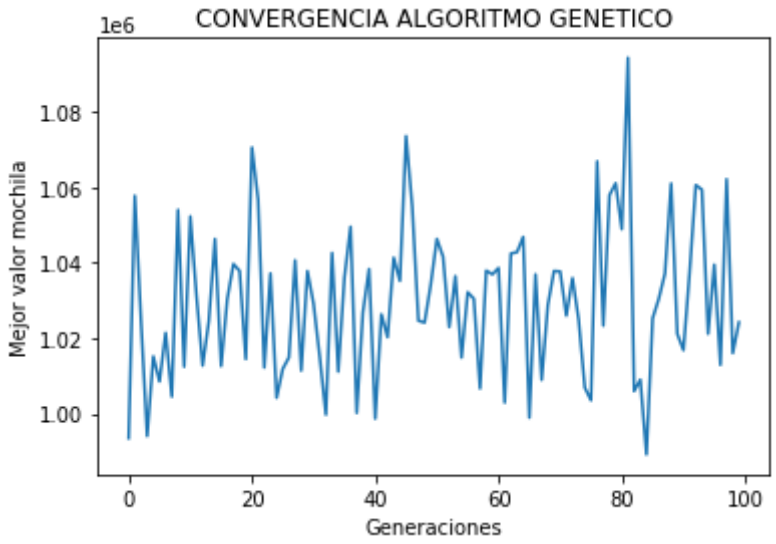


**Figura 10.** *Grafica convergencia, tercera ejecución.*

Tiempo empleado para encontrar la solución: 149.78535389900208 segundos
Número de iteraciones para encontrar la mejor solución: 34
Valor total de la mochila: 1102039
Número total de iteraciones: 60000

**Comentario:**

La solución encontrada con el método de búsqueda algoritmo genético en esta tercera ejecución y con los parámetros vistos anteriormente, se observa que el tiempo empleado para encontrar la solución fue de 149.79 segundos, con 34 iteraciones para encontrar la mejor solución y un valor total de la mochila de 1102039. Comparando con las ejecuciones anteriores, se puede observar que el tiempo de ejecución fue significativamente mayor en esta configuración específica de parámetros, lo que indica que el algoritmo pudo ser menos eficiente en términos de tiempo en comparación con las dos ejecuciones anteriores. Sin embargo, es importante tener en cuenta que la eficiencia de un algoritmo genético depende de muchos factores, incluyendo la configuración de parámetros y la naturaleza del problema a resolver. Por lo tanto, es necesario ajustar y optimizar los parámetros del algoritmo de acuerdo a las características específicas del problema para obtener los mejores resultados.

**Conclusión:**

En las tres ejecuciones del algoritmo genético con diferentes configuraciones de parámetros, se observa que el tiempo empleado para encontrar la solución fue menor en la primera ejecución (3.31 segundos) en comparación con la segunda (59.19 segundos) y la tercera (149.79 segundos). Sin embargo, el número de iteraciones para encontrar la mejor solución fue menor en la segunda ejecución (81 iteraciones) en comparación con la primera (33 iteraciones) y la tercera (34 iteraciones). El valor total de la mochila también varió ligeramente en las tres ejecuciones, siendo el más alto en la primera ejecución (1114476) y el más bajo en la segunda ejecución (1094259).

Es importante tener en cuenta que la eficiencia y el desempeño del algoritmo genético dependen de las configuraciones dada inicialmente.

| Ejecuciones | Tiempo | Mejor iteración | Valor mochila | Total de iteraciones |
|---|---|---|---|---|
| 1 | 3.3111231327056885 | 33 | 1114476 | 5000 |
| 2 | 59.18733286857605 | 81 | 1094259 | 30000 |
| 3 | 149.78535389900208 | 34 | 1102039 | 60000 |

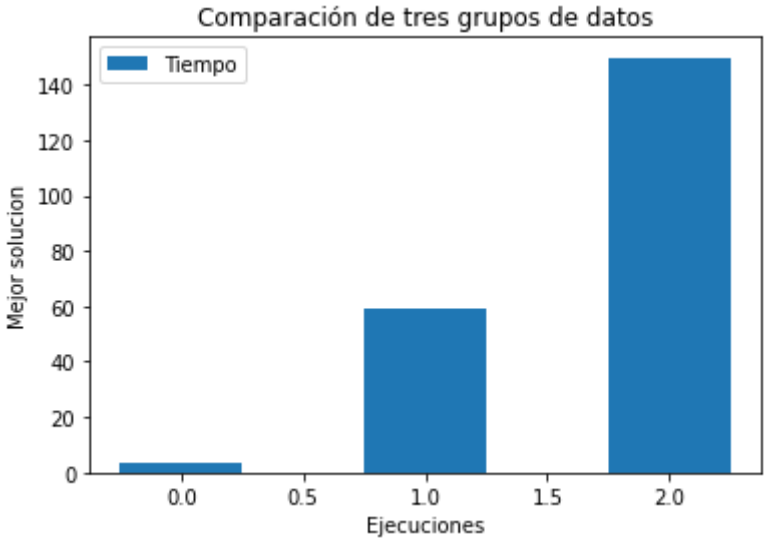


**Figura 11.** *Comparación de los tres grupos de datos.*

## COLONIA DE HORMIGAS

El algoritmo de búsqueda inspirado en colonias de hormigas se basa en el comportamiento de las hormigas reales que buscan comida. En este algoritmo, las soluciones candidatas se representan como combinaciones de objetos que podrían ser elegidos para llenar una mochila. Cada hormiga en la colonia construye una solución al elegir objetos y depositar feromonas en la combinación elegida. La cantidad de feromonas depositadas por las hormigas indica la calidad de la solución. Las hormigas posteriores pueden detectar la presencia de feromonas y utilizar esta información para guiar su elección de objetos en la combinación. Conforme encuentran soluciones mejores, depositan más feromonas en las combinaciones correspondientes, lo que atrae

a más hormigas a elegir esos objetos. El algoritmo de búsqueda de colonia de hormigas es iterativo, lo que implica que la colonia de hormigas revisa las combinaciones varias veces. En cada iteración, las hormigas eligen objetos y depositan feromonas en las combinaciones elegidas. Las combinaciones más prometedoras se identifican y se utilizan para actualizar las feromonas en la combinación. Con el tiempo, el algoritmo converge a una solución óptima, la cual se puede encontrar siguiendo la combinación con la mayor cantidad de feromonas depositadas.

## DESCRIPCION DEL ESQUEMA

El código utiliza un esquema de representación binario para el problema de la mochila, donde cada elemento se representa por un bit en un vector de bits que representa la solución y pues el algoritmo de colonia de hormigas (ACO) se utiliza para encontrar la mejor solución. En este algoritmo, las hormigas construyen soluciones aleatorias y se actualizan las feromonas según las soluciones encontradas. Luego, la información de las feromonas se utiliza para guiar a las hormigas hacia soluciones mejores en las próximas iteraciones. El objetivo es llenar la mochila con los elementos de mayor valor sin exceder su capacidad. El algoritmo se ejecuta durante un número de generaciones determinado y devuelve la mejor solución encontrada, su valor total, el número de iteraciones necesarias para encontrar la mejor solución y el número total de iteraciones ejecutadas. También se muestra una gráfica de la convergencia del algoritmo.

## ESTRATEGIA PARA PRODUCIR POSIBLES SOLUCIONES

Este algoritmo de colonia de hormigas (ACO) utiliza una técnica que se puede desglosar en los siguientes pasos:

1. Definición del problema de la mochila: Se trata de seleccionar un conjunto de artículos de una lista dada, con el fin de maximizar el valor total de los artículos elegidos, sin superar una capacidad de peso determinada.
2. Establecimiento de los parámetros del algoritmo ACO: Incluye la cantidad de hormigas, el número de generaciones, la ecuación de selección de artículos, el factor de evaporación de feromonas, entre otros.
3. Inicialización de las feromonas: Cada hormiga deposita feromonas en los artículos que selecciona y su nivel de feromona es utilizado por otras hormigas para decidir si elegir ese artículo o no. Al inicio, todas las feromonas se establecen en un valor igual.
4. Construcción de soluciones: Las hormigas construyen soluciones al problema de la mochila siguiendo un camino. En cada paso, eligen un artículo basado en la cantidad de feromona depositada en el artículo y su valor/peso. Si el artículo puede ser agregado a la mochila sin exceder la capacidad máxima, entonces se agrega.
5. Actualización de las feromonas: Después de que todas las hormigas hayan construido soluciones, se actualizan las feromonas en función de la calidad de las soluciones. Las feromonas en los artículos seleccionados por las mejores soluciones se incrementan mientras que las feromonas en los artículos no

seleccionados se reducen. Se aplica una tasa de evaporación de la feromona para evitar la convergencia prematura.

6. Repetición de los pasos 4 y 5 para varias generaciones: Se repiten los pasos 4 y 5 durante varias generaciones hasta que se encuentre una solución satisfactoria o se alcance el número máximo de generaciones.
7. Devolución de la mejor solución: Al final del algoritmo, se devuelve la mejor solución encontrada y su valor total.
8. Representación gráfica de la convergencia: Es conveniente graficar la convergencia del algoritmo para visualizar la mejora del valor de la solución a lo largo de las generaciones.

# FUNCION OBJETIVO

La función objetivo de este código es resolver el problema de la mochila (Knapsack) utilizando el algoritmo de colonia de hormigas (ACO). El objetivo es maximizar el valor total de los objetos que se pueden colocar en la mochila, teniendo en cuenta la capacidad máxima de la misma.

El algoritmo de ACO utiliza feromonas para encontrar soluciones óptimas al problema de la mochila. Cada hormiga construye una solución de forma probabilística, teniendo en cuenta la información de las feromonas y la información heurística (relación valor-peso de los objetos). Después de cada generación, se actualizan las feromonas de acuerdo con la calidad de la mejor solución encontrada. La ejecución del algoritmo devuelve la mejor solución encontrada, su valor total y la cantidad de iteraciones necesarias para encontrarla. Además, se grafica la convergencia del algoritmo a lo largo de las generaciones.

**Primera Ejecución:**

| | |
|---|---|
| num_ants | 200 |
| num_generations | 20 |
| alpha | 0.3 |
| beta | 0.3 |
| evaporation_rate | 0.4 |

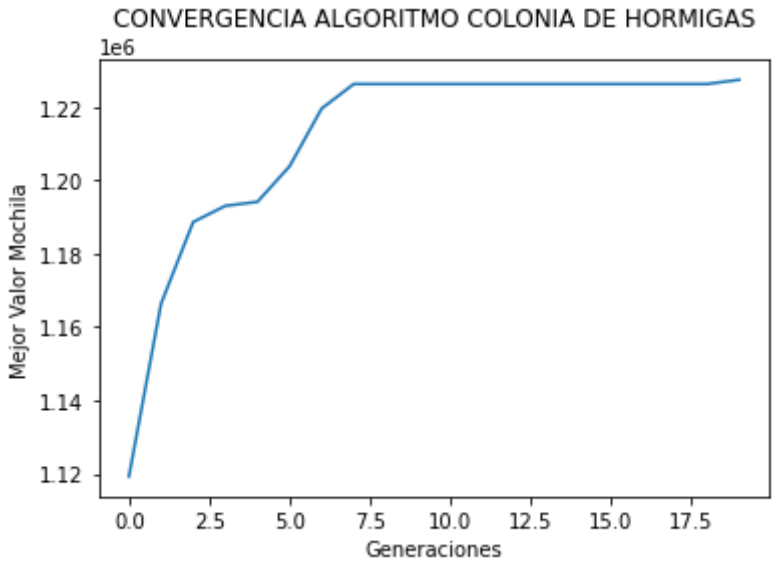


**Figura 12.** *Grafica convergencia, primera ejecución.*

Tiempo empleado para encontrar la solución: 15.651053428649902 segundos
Número de iteraciones para encontrar la mejor solución: 3800
Mejor solución con su respectivo valor total: 1227483
Número total de iteraciones: 4000

**Comentario:**

El método de búsqueda llamado colonia de hormigas con los parámetros especificados anteriormente, encontró una solución en 3800 iteraciones, con un tiempo de ejecución de 15.65 segundos. La mejor solución encontrada tuvo un valor total de 1227483. El número total de iteraciones realizadas fue de 4000. Estos resultados indican que el algoritmo pudo encontrar una solución prometedora en un tiempo razonable, lo que sugiere que la colonia de hormigas es una técnica eficaz para resolver el problema específico que se estaba abordando. Sin embargo, el rendimiento del algoritmo podría seguir mejorando ajustando los parámetros y realizando más experimentos.

**Segunda Ejecución:**

| | |
|---|---|
| num_ants | 400 |
| num_generations | 40 |
| alpha | 0.6 |
| beta | 0.6 |
| evaporation_rate | 0.7 |

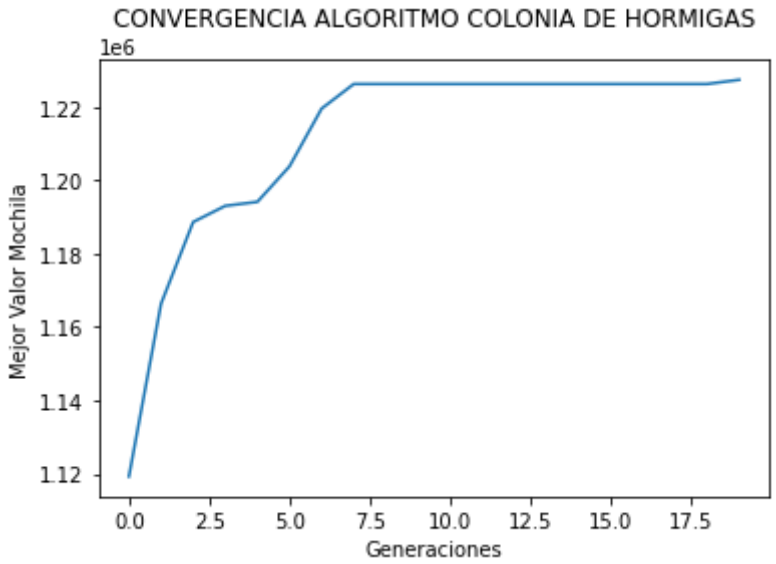


**Figura 13.** *Grafica convergencia, segunda ejecución.*

Tiempo empleado para encontrar la solución: 65.11378788948059 segundos
Número de iteraciones para encontrar la mejor solución: 3600
Mejor solución con su respectivo valor total: 1206507
Número total de iteraciones: 16000

**Comentario:**

En comparación con la ejecución anterior del algoritmo de colonia de hormigas, los resultados con los nuevos parámetros muestran que el tiempo empleado para encontrar la solución aumentó a 65.11 segundos, el número de iteraciones para encontrar la mejor solución disminuyó a 3600 y la mejor solución encontrada tuvo un valor total de 1206507. Sin embargo, el número total de iteraciones aumentó a 16000.

Estos resultados sugieren que el aumento en el tamaño de la población de hormigas y el número de generaciones permitió encontrar una solución más rápidamente, pero a costa de aumentar el número total de iteraciones. Además, el aumento en los valores de alpha y beta podría haber llevado a una exploración más intensiva del espacio de búsqueda, lo que resultó en la mejora del valor total de la mejor solución encontrada en comparación con la ejecución anterior.

Sin embargo, el tiempo de ejecución también se incrementó considerablemente, lo que indica que hay un compromiso entre la calidad de la solución y el tiempo de ejecución en el ajuste de los parámetros del algoritmo

**Tercera Ejecución:**

| | |
|---|---|
| num_ants | 600 |
| num_generations | 60 |
| alpha | 0.6 |
| beta | 0.8 |
| evaporation_rate | 0.2 |

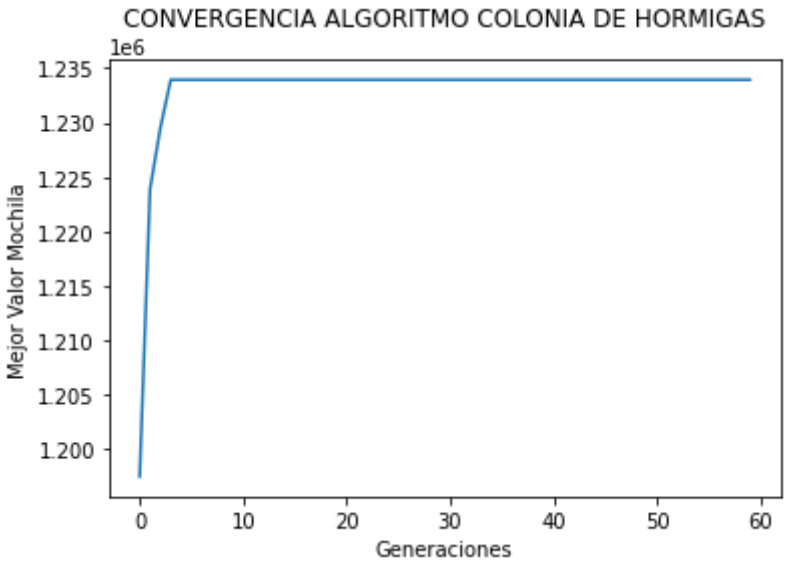


**Figura 14.** *Grafica convergencia, tercera ejecución.*

Tiempo empleado para encontrar la solución: 139.78109502792358 segundos

Número de iteraciones para encontrar la mejor solución: 1800

Mejor solución con su respectivo valor total: 1233938

Número total de iteraciones: 36000

**Comentario:**

En comparación con las ejecuciones anteriores del algoritmo de colonia de hormigas, los resultados con los nuevos parámetros muestran que el tiempo empleado para encontrar la solución aumentó significativamente a 139.78 segundos. Sin embargo, el número de iteraciones para encontrar la mejor solución se redujo a 1800, y la mejor solución encontrada tuvo un valor total de 1233938. El número total de iteraciones aumentó a 36000.

En esta tercera ejecución el aumento en el tamaño de la población de hormigas y el número de generaciones pudo haber permitido encontrar una solución más rápidamente en términos de número de iteraciones, pero a costa de un mayor tiempo de ejecución. Además, los valores más altos de Alpha y beta pueden haber llevado a una mayor intensidad en la exploración del espacio de búsqueda, lo que resultó en una mejora en el valor total de la mejor solución encontrada en comparación con las ejecuciones anteriores.

**Conclusión:**

En resumen, las tres ejecuciones del algoritmo de colonia de hormigas con diferentes conjuntos de parámetros (num_ants, num_generations, alpha, beta, evaporation_rate) han mostrado resultados diferentes en términos de tiempo de ejecución, número de iteraciones para encontrar la mejor solución y el valor total de la mejor solución encontrada.

También se puede observar que a medida que se incrementaron los valores de num_ants y num_generations, se obtuvieron mejores soluciones en términos de valor total, pero a costa de un mayor tiempo de ejecución. Los valores de alpha y beta también tuvieron un impacto en la calidad de las soluciones encontradas.

| Ejecuciones | Tiempo | Mejor iteración | Valor mochila | Total de iteraciones |
|---|---|---|---|---|
| 1 | 15.651053428649902 | 3800 | 1227483 | 4000 |
| 2 | 65.11378788948059 | 3600 | 1206507 | 16000 |
| 3 | 139.78109502792358 | 1800 | 1233938 | 36000 |

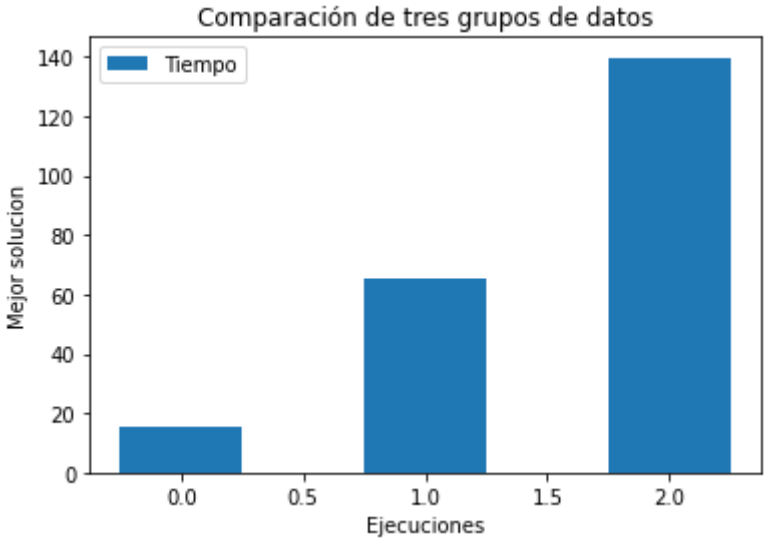


**Figura 15.** *Comparación de los tres grupos de datos.*

# EJECUCIONES DE LOS ALGORITMOS CON LA MEJOR CONFIGURACIÓN OBTENIDA

Cada algoritmo fue probado varias veces con diferentes combinaciones de parámetros para encontrar la mejor configuración que produjera los mejores resultados en el problema de la mochila. Los parámetros específicos que se probaron para cada algoritmo fueron la temperatura inicial, tasa de enfriamiento y número de iteraciones para el enfriamiento simulado; tamaño de la población, número de generaciones, probabilidad de mutación y probabilidad de cruce para el algoritmo genético; y tamaño de la colonia, tasa de evaporación, factor de influencia de la feromona y factor de influencia de la heurística para la colonia de hormigas.

Una vez encontrada la mejor configuración para cada algoritmo, se registraron los resultados en tablas y se generaron gráficas para comparar su desempeño en términos de tiempo y calidad de la solución encontrada. Estos resultados incluyen la mejor solución encontrada, el valor de la función objetivo y el tiempo de ejecución para cada algoritmo y configuración.

El análisis de estos resultados permitió comparar la eficacia de cada algoritmo y su mejor configuración, así como identificar las combinaciones de parámetros que funcionan mejor para resolver el problema de la mochila.

Ya obtenidos los mejores resultados se procedió a configurar cada algoritmo, con la mejor configuración encontrados anteriormente para así poder ejecutar 30 veces cada algoritmo, en la siguiente sección podrá visualizar todos los datos obtenidos de las ejecuciones.

## 30 EJECUCIONES ALGORITMO SA

DATOS PROMEDIO
MAYOR VALOR: 957728.8
TIEMPO: 0.004929359753926543 segundos
MEJORES ITERACIONES: 45,66666667

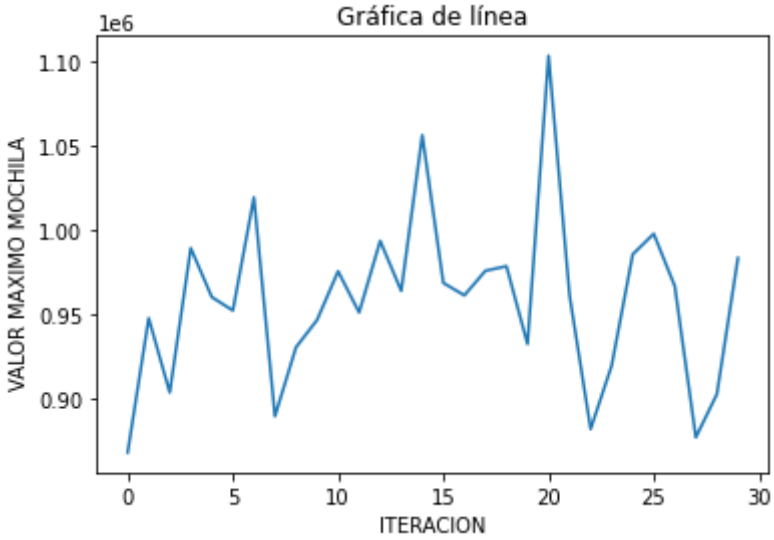


**Figura 16.** *Grafica valor máximo en SA.*

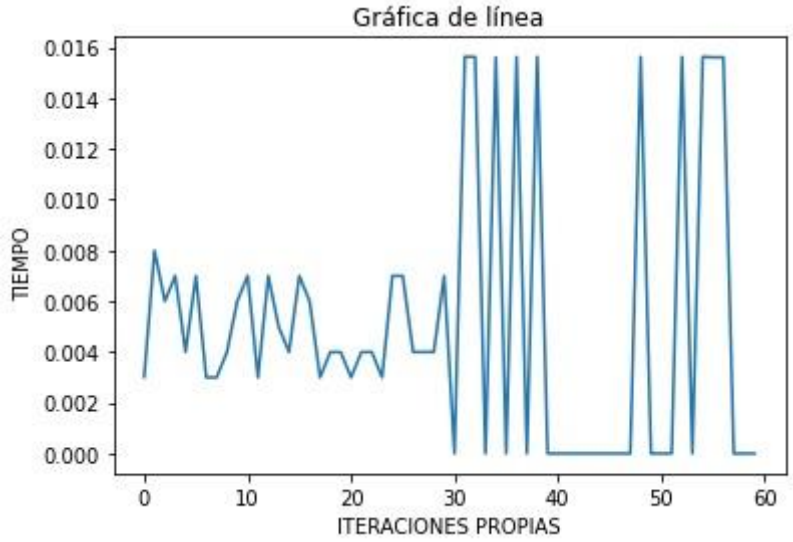


**Figura 17.** *Grafica tiempo máximo en SA.*


**Figura 18.** *Grafica total de iteraciones en SA.*

| valor maxim | tiempo | total de iteracione |
|---|---|---|
| 867608 | 0.003016233444213867 | 33 |
| 947320 | 0.007990360260009766 | 86 |
| 903231 | 0.005996227264404297 | 39 |
| 988902 | 0.006997108459472656 | 40 |
| 959819 | 0.003997802734375 | 43 |
| 951893 | 0.006990671157836914 | 37 |
| 1019005 | 0.0029969215393066406 | 44 |
| 889105 | 0.0029938220977783203 | 39 |
| 930053 | 0.003993034362792969 | 46 |
| 946229 | 0.005997419357299805 | 39 |
| 975160 | 0.0069963932037353516 | 47 |
| 950717 | 0.0029916763305664062 | 49 |
| 993190 | 0.006995677947998047 | 53 |
| 963522 | 0.004995822906494141 | 69 |
| 1055948 | 0.003997325897216797 | 63 |
| 968252 | 0.00699305534362793 | 35 |
| 960879 | 0.0059947967529296875 | 53 |
| 975448 | 0.0029964447021484375 | 38 |
| 978159 | 0.003994941711425781 | 35 |
| 932001 | 0.003991365432739258 | 49 |
| 1103120 | 0.0029964447021484375 | 77 |
| 960765 | 0.003995418548583984 | 35 |
| 881533 | 0.003997325897216797 | 36 |
| 919212 | 0.0029990673065185547 | 39 |
| 985192 | 0.006995439529418945 | 53 |
| 997415 | 0.0069942474365234375 | 44 |
| 966321 | 0.003997325897216797 | 38 |
| 876589 | 0.003996849060058594 | 37 |
| 902236 | 0.003997802734375 | 38 |
| 983040 | 0.006993770599365234 | 36 |

**Figura 19.** *Datos devueltos del algoritmo SA.*

# 30 EJECUCIONES ALGORITMO GA

DATOS PROMEDIO
MAYOR VALOR: 1076777.56
TIEMPO: 17.965.775.998.433.400 SEGUNDOS
MEJORES ITERACIONES: 23,36666667

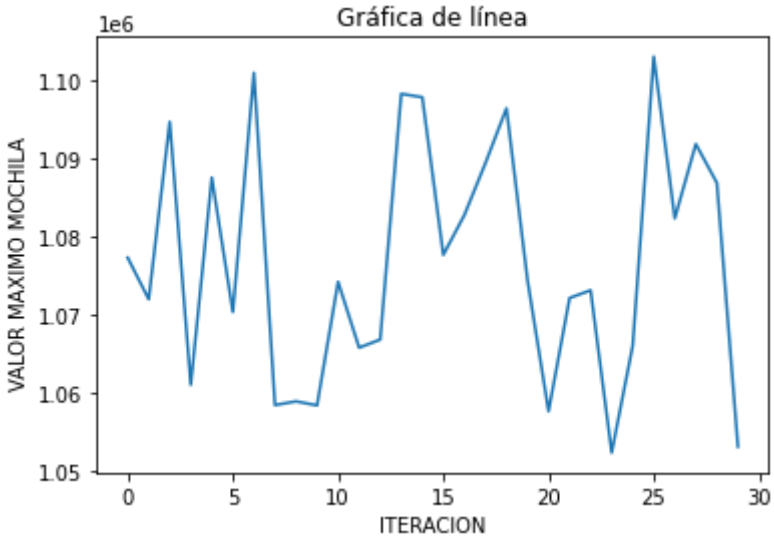


**Figura 20.** *Grafica valor máximo en GA.*

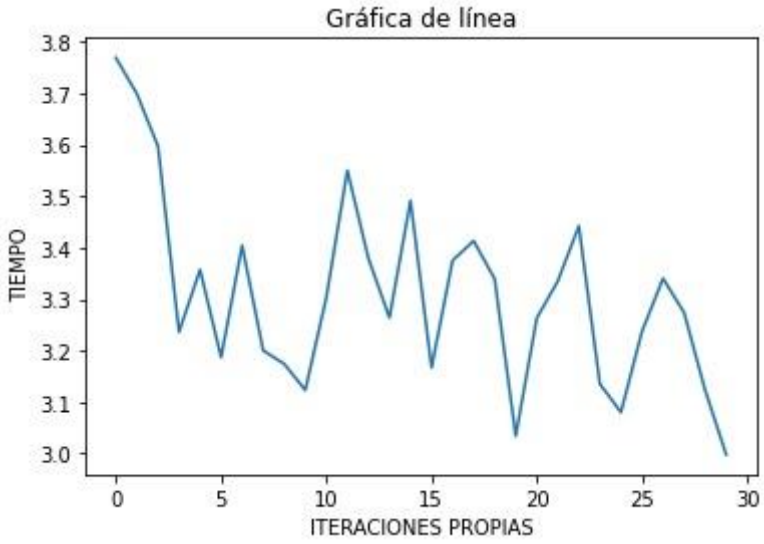


**Figura 21.** *Grafica tiempo máximo en GA.*


**Figura 22.** *Grafica total de iteraciones en GA.*

| valor maxim | tiempo | iteraciones totale |
|---|---|---|
| 1077364 | 37.688.543.796.539.300 | 49 |
| 1072047 | 36.986.141.204.833.900 | 43 |
| 1094709 | 3.596.935.987.472.530 | 36 |
| 1061115 | 32.361.435.890.197.700 | 4 |
| 1087587 | 3.357.069.492.340.080 | 43 |
| 1070411 | 31.871.914.863.586.400 | 21 |
| 1100977 | 3.404.062.032.699.580 | 27 |
| 1058517 | 32.001.638.412.475.500 | 46 |
| 1058992 | 31.741.750.240.325.900 | 0 |
| 1058478 | 31.232.080.459.594.700 | 5 |
| 1074271 | 33.031.017.780.303.900 | 6 |
| 1065851 | 35.499.658.584.594.700 | 30 |
| 1066886 | 3.379.056.930.541.990 | 7 |
| 1098292 | 32.641.265.392.303.400 | 7 |
| 1097861 | 3.490.997.076.034.540 | 39 |
| 1077714 | 31.671.805.381.774.900 | 14 |
| 1082802 | 337.506.365.776.062 | 20 |
| 1089432 | 34.130.616.188.049.300 | 18 |
| 1096439 | 33.391.036.987.304.600 | 49 |
| 1074488 | 3.034.255.266.189.570 | 6 |
| 1057737 | 3.263.129.234.313.960 | 1 |
| 1072203 | 3.335.087.299.346.920 | 34 |
| 1073184 | 3.442.025.899.887.080 | 8 |
| 1052438 | 3.135.199.785.232.540 | 7 |
| 1066109 | 30.802.536.010.742.100 | 46 |
| 1103052 | 3.237.159.490.585.320 | 10 |
| 1082385 | 3.340.104.818.344.110 | 45 |
| 1091874 | 3.274.125.337.600.700 | 23 |
| 1086915 | 312.320.876.121.521 | 18 |
| 1053197 | 29.982.802.867.889.400 | 39 |

**Figura 23.** *Datos devueltos del algoritmo GA.*

# 30 EJECUCIONES ALGORITMO ACO

DATOS PROMEDIO
MAYOR VALOR: 1226635.83
TIEMPO: 9.003.730.135.520.260 segundos
TOTAL DE ITERACIONES: 5740

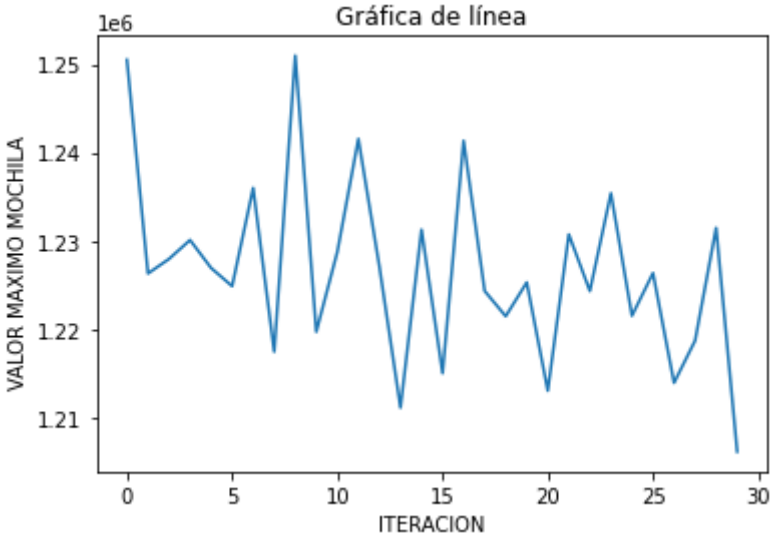


**Figura 24.** *Grafica valor máximo en ACO.*

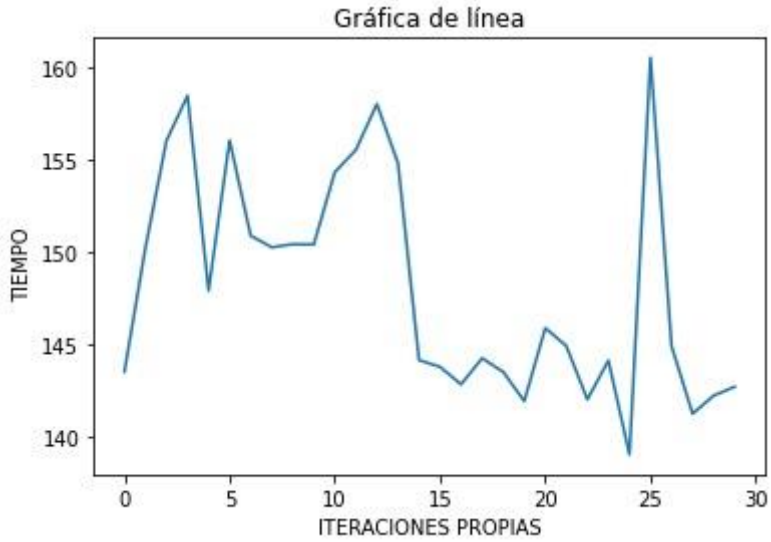


**Figura 25.** *Grafica tiempo máximo en ACO.*


**Figura 26.** *Grafica total de iteraciones en ACO.*

| valor maximo | tiempo | total de iteraciones |
|---|---|---|
| 1250514 | 14.353.501.653.671.200 | 3000 |
| 1226288 | 1.502.567.012.310.020 | 2400 |
| 1227940 | 15.604.624.819.755.500 | 2400 |
| 1230083 | 1.584.790.825.843.810 | 4800 |
| 1226918 | 14.793.178.701.400.700 | 3600 |
| 1224854 | 1.560.660.445.690.150 | 12000 |
| 1235988 | 15.090.535.306.930.500 | 3000 |
| 1217422 | 15.027.289.605.140.600 | 10200 |
| 1250993 | 1.504.428.939.819.330 | 4800 |
| 1219681 | 1.504.264.166.355.130 | 3000 |
| 1228749 | 15.434.170.174.598.600 | 5400 |
| 1241552 | 15.554.748.582.839.900 | 10200 |
| 1227044 | 15.803.018.879.890.400 | 4800 |
| 1211076 | 1.548.250.744.342.800 | 1200 |
| 1231262 | 14.417.238.569.259.600 | 15000 |
| 1214980 | 1.437.991.964.817.040 | 2400 |
| 1241338 | 14.285.870.099.067.600 | 7800 |
| 1224307 | 144.270.920.753.479 | 6600 |
| 1221442 | 1.435.291.645.526.880 | 1200 |
| 1225278 | 14.195.043.110.847.400 | 6000 |
| 1212997 | 14.590.348.958.969.100 | 6000 |
| 1230720 | 1.449.358.286.857.600 | 9600 |
| 1224301 | 14.203.997.230.529.700 | 2400 |
| 1235387 | 14.414.517.402.648.900 | 8400 |
| 1221499 | 1.390.605.137.348.170 | 1200 |
| 1226335 | 16.053.575.921.058.600 | 7200 |
| 1213897 | 1.449.275.302.886.960 | 3000 |
| 1218678 | 14.126.626.205.444.300 | 2400 |
| 1231461 | 14.224.602.675.437.900 | 6600 |
| 1206091 | 1.427.260.775.566.100 | 15600 |

**Figura 27.** *Datos devueltos del algoritmo ACO.*

# COMPARACIÓN DE LOS MÉTODOS

A continuación, se utilizan los algoritmos SA, AG y Colonia de Hormigas para la comparación en términos de su capacidad para encontrar soluciones óptimas al problema de la mochila en el menor tiempo.

Para esto se ejecutó cada código 30 veces con la configuración más optima seleccionada de las ejecuciones anteriores, se calculó el promedio de tiempo de tiempo en dar solución y se compara en la siguiente gráfica.

Para SA el tiempo promedio es de 0.004929359753926543 segundos, para GA el tiempo promedio es de 17.965.775.998.433.400 segundos y para colonia de hormigas el promedio fue de 9,003 segundos.

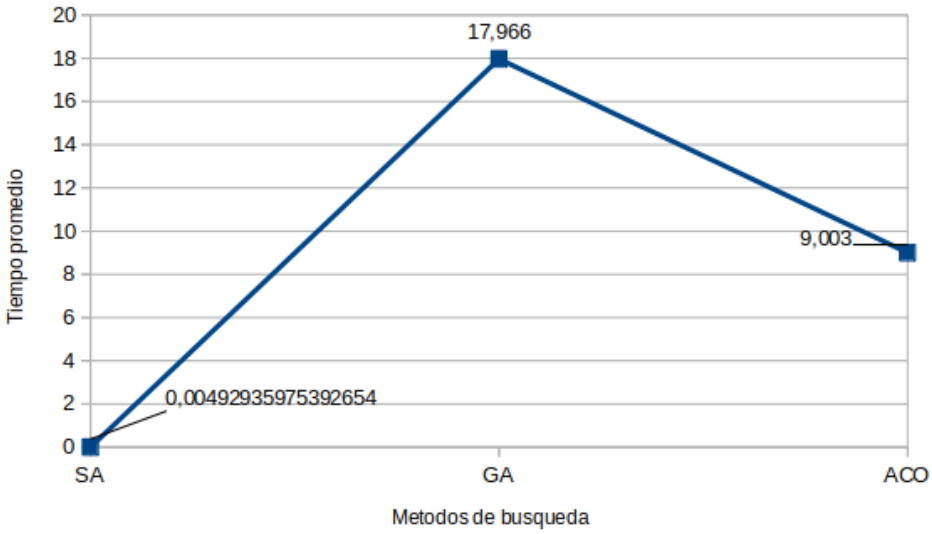


**Figura 28.** *Grafica tiempo promedio de solución.*

En promedio, el algoritmo SA es el más rápido en resolver el problema de la mochila en comparación con los algoritmos AG y ACO. Esto podría significar que el algoritmo SA es más eficiente para encontrar soluciones óptimas o cercanas a óptimas en menos tiempo que los otros algoritmos evaluados.

Sin embargo, es importante tener en cuenta que los resultados de la comparación se basan en promedios y que el desempeño de cada algoritmo puede variar dependiendo de los parámetros de entrada y las características específicas del problema de la mochila. Por lo tanto, sería necesario realizar pruebas adicionales y evaluar otros criterios para determinar la eficacia de los algoritmos en diferentes situaciones.

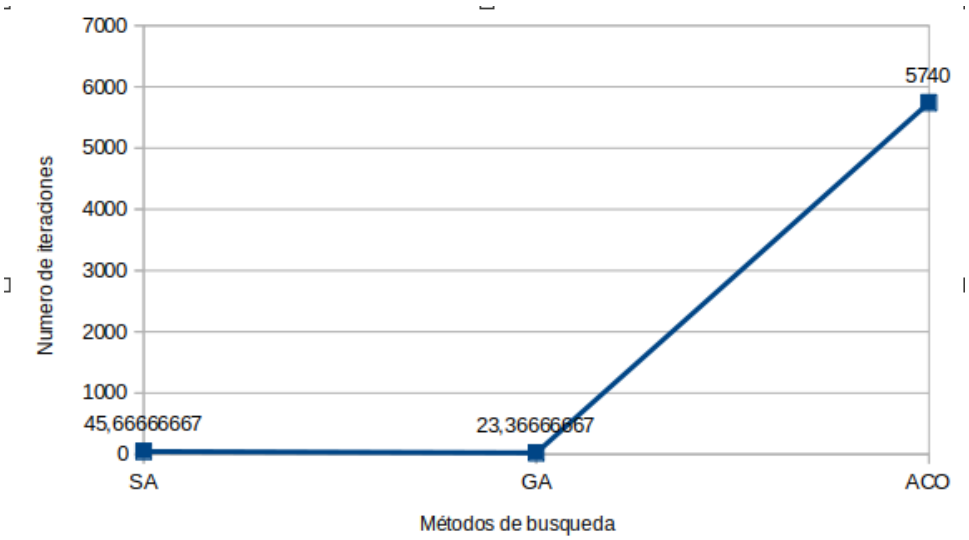


**Figura** 29**.** Grafica de Numero de iteraciones promedio de solución.

En promedio, el algoritmo SA tuvo la mejor cantidad de iteración promedio para resolver el problema de la mochila en comparación con los algoritmos AG y ACO. Esto podría significar que el algoritmo SA es capaz de encontrar soluciones óptimas o cercanas a óptimas en menos iteraciones que los otros algoritmos evaluados.

.

# CONCLUSIÓN

En general, los resultados muestran que los tres algoritmos son efectivos para resolver el problema de la mochila, pero cada uno tiene sus ventajas y desventajas en términos de eficiencia y precisión de la solución. El enfriamiento simulado se destaca por encontrar soluciones óptimas o aceptables en menos iteraciones, lo que significa que consume menos recursos computacionales y tiempo. Por otro lado, el algoritmo genético y la colonia de hormigas pueden tardar más en encontrar soluciones óptimas o aceptables, pero tienen la capacidad de explorar un espacio de solución más amplio y encontrar soluciones potencialmente más diversas y eficientes.

Es importante destacar que la elección del mejor algoritmo dependerá del problema específico que se esté abordando y de los requisitos del usuario. Si se requiere una solución precisa y rápida, el enfriamiento simulado puede ser la mejor opción. Si se busca una solución más diversa y con una exploración más amplia del espacio de solución, el algoritmo genético o la colonia de hormigas pueden ser más adecuados.

Además, es importante tener en cuenta que los resultados obtenidos en este estudio se basan en un conjunto específico de parámetros y condiciones de entrada. Es posible que diferentes configuraciones de parámetros o diferentes problemas requieran diferentes enfoques de algoritmos. Por lo tanto, es importante realizar pruebas exhaustivas y ajustar los parámetros para optimizar el rendimiento de cada algoritmo para cada problema en particular.