Informe Trabajo Final: Optimización Metaheurística de Problemas de Logística Compleja (TSP y VRP)

Materia: Algoritmos Evolutivos I
Carrera: Especialización en Inteligencia Artificial
Alumno: Ezequiel Alejandro Caamaño (a1802)
Librerías y Herramientas: Python 3.x, PyGAD (Genetic Algorithm Framework), NumPy, SciPy (Matriz de Distancias), Matplotlib

---

1. INTRODUCCIÓN Y MARCO TEÓRICO

1.1 Contexto y Motivación

El presente trabajo aborda la resolución de dos de los problemas más emblemáticos en el campo de la optimización combinatoria y la investigación operativa: el **Problema del Viajante (TSP)** y el **Problema de Ruteo de Vehículos (VRP)**. Ambos pertenecen a la clase de complejidad **NP-Hard**, lo que implica que el tiempo necesario para encontrar la solución óptima mediante métodos exactos (como fuerza bruta) crece de manera exponencial con el número de nodos.

Para un TSP con 20 ciudades, el espacio de búsqueda contiene **$20! \approx 2.43 \times 10^{18}$** permutaciones posibles. Incluso evaluando un millón de soluciones por segundo, se requerirían aproximadamente **77 millones de años** para explorar exhaustivamente el espacio completo. Esta intratabilidad computacional justifica el uso de **metaheurísticas** como los Algoritmos Genéticos.

1.2 Justificación de los Problemas Seleccionados

La elección de estos problemas permite realizar un estudio comparativo sobre la flexibilidad de los **Algoritmos Genéticos (GA)** en dos escenarios fundamentalmente distintos:

TSP (Traveling Salesman Problem)

Se utiliza como un "laboratorio" de optimización de permutaciones puras. El objetivo es determinar el **ciclo hamiltoniano de costo mínimo** en un grafo completo. Formalmente:

Minimizar: $$\sum_{i=0}^{n-1} d(\pi_i, \pi_{(i+1) \mod n})$$

Donde π es una permutación de las ciudades y $d(i,j)$ es la distancia euclidiana entre ciudades i y j.

Características:

- Sin restricciones hard (cualquier permutación es válida)
- Espacio de búsqueda: $O(n!)$
- Aplicaciones: logística, manufactura de PCBs, secuenciación de tareas

VRP (Vehicle Routing Problem)

Añade una capa de complejidad del mundo real. Ya no se trata solo de ordenar ciudades, sino de **asignar recursos limitados** (vehículos) a tareas (clientes) respetando una **restricción de capacidad (Hard Constraint)**.

Minimizar: $$\sum_{k=1}^{K} \sum_{i,j \in R_k} d(i,j)$$

Sujeto a:

- $\sum_{i \in R_k} demand_i \leq capacity_k$ (restricción de capacidad)
- $\bigcup_{k=1}^{K} R_k = {1,...,n}$ (todos los clientes atendidos)
- $R_i \cap R_j = \emptyset$ para $i \neq j$ (sin solapamiento)

Características:

- Restricciones hard de capacidad
- Múltiples agentes (vehículos)
- Espacio de búsqueda: $O(K^n \times n!)$
- Aplicaciones: distribución logística, recolección de residuos, rutas escolares

1.3 Estado del Arte en Algoritmos Genéticos

Los Algoritmos Genéticos, propuestos por John Holland en 1975, son técnicas de búsqueda estocástica inspiradas en la evolución biológica. Operan mediante tres principios fundamentales:

1. **Selección Natural:** Los individuos más aptos tienen mayor probabilidad de reproducirse
2. **Recombinación:** El crossover intercambia material genético entre padres
3. **Mutación:** Introduce variabilidad aleatoria para explorar nuevas regiones

Ventajas demostradas en literatura:

- No requieren información del gradiente (función fitness tipo caja negra)
- Paralelización natural mediante poblaciones
- Balance automático entre exploración y explotación
- Adaptabilidad a restricciones mediante penalizaciones

Desafíos específicos para TSP/VRP:

- Representación apropiada (permutaciones vs asignaciones)
- Operadores genéticos que preserven factibilidad
- Manejo de restricciones sin pérdida de eficiencia

---

2. ANÁLISIS DE LA IMPLEMENTACIÓN TÉCNICA

2.1 Arquitectura General del Sistema

La implementación se realizó utilizando **PyGAD**, una biblioteca diseñada originalmente para optimización de parámetros continuos. El núcleo del desafío técnico fue adaptar esta naturaleza continua a la estructura discreta de los problemas logísticos.

Flujo de ejecución:

```
Inicialización → Evaluación Fitness → Selección → Crossover → Mutación →
¿Convergió? → Retornar Mejor
       ↑                                                                  │
       └──────────────────────────────────────────────────────────────────┘
```

2.2 Decodificación de Cromosomas (Genotipo a Fenotipo)

Para evitar el uso de operadores de cruce especializados (como el **Ordered Crossover** o **Partially Mapped Crossover**), se implementó una **codificación indirecta** innovadora:

Representación en TSP:

Genotipo (Cromosoma): Vector continuo de 20 valores en [0,1]

```
solution = [0.73, 0.12, 0.89, 0.45, 0.67, ...]  # 20 valores
```

Decodificación mediante argsort:

```
def fitness_func(ga_instance, solution, solution_idx):
    # El cromosoma continuo se convierte en una ruta de índices enteros
    route = np.argsort(solution)  # [1, 3, 5, 0, 4, 2, ...]
    distance = calculate_route_distance(route, dist_matrix)
    return -distance  # Negativo porque PyGAD maximiza
```

Ejemplo de mapeo:

```
Cromosoma: [0.73, 0.12, 0.89, 0.45, 0.67]
          ↓ argsort (ordena índices por valor)
Ruta:      [1, 3, 0, 4, 2]  (ciudad 1 primero, luego 3, etc.)
```

Ventajas de este enfoque:

1. **Garantía de factibilidad:** Toda solución continua mapea a una permutación válida
2. **Compatibilidad:** Operadores genéticos estándar (crossover de un punto, mutación uniforme) funcionan directamente
3. **Eficiencia:** O(n log n) para argsort vs O(n^2) para operadores especializados
4. **Simplicidad:** Evita lógica compleja de reparación de cromosomas

Justificación teórica: El espacio continuo $[0,1]^n$ es denso y conexo, permitiendo que el GA explore suavemente el espacio de permutaciones mediante pequeñas perturbaciones continuas que se traducen en cambios de orden.

2.3 Arquitectura del Fitness y Penalizaciones en VRP

En el caso del VRP, la función objetivo se diseñó bajo el esquema de **Costo Aumentado** (Augmented Lagrangian). El fitness se calcula como:

$$Fitness = -(Costo_Distancia + \lambda_1 \cdot Penalización_Capacidad + \lambda_2 \cdot Penalización_Uso)$$

Implementación detallada:

```
def fitness_func(ga_instance, solution, solution_idx):
    routes = decode_solution_to_routes(solution, num_customers, num_vehicles)

    # Componente 1: Costo de distancia (objetivo primario)
```

```
    total_cost = sum(calculate_route_cost(route, dist_matrix) for route in
routes)

    # Componente 2: Penalización por exceso de capacidad (λ₁ = 50)
    penalty = 0
    for route in routes:
        if len(route) > 0:
            route_demand = sum(demands[c - 1] for c in route)
            if route_demand > vehicle_capacity:
                penalty += (route_demand - vehicle_capacity) * 50

    # Componente 3: Penalización por subutilización (λ₂ = 20)
    empty_vehicles = sum(1 for route in routes if len(route) == 0)
    penalty += empty_vehicles * 20

    return -(total_cost + penalty)
```

Calibración de pesos de penalización:

Penalización	Peso (λ)	Justificación Técnica
Exceso capacidad	$\lambda_1 = 50$	Distancia típica ruta: 150-300. Si excede 5 unidades: penalización = 250, domina fitness → solución rechazada rápidamente
Vehículos vacíos	$\lambda_2 = 20$	Incentiva uso eficiente sin ser crítica. Costo oportunidad de tener vehículo sin usar

Análisis del paisaje de fitness:

El uso de penalizaciones crea un "paisaje de fitness" con características específicas:

1. **Valle de factibilidad:** Región donde demand ≤ capacity tiene fitness bajo (bueno)
2. **Acantilados de infactibilidad:** Gradiente abrupto cuando se viola capacidad
3. **Mesetas de subutilización:** Penalización suave por vehículos vacíos

Este diseño permite al GA:

- **Fase inicial:** Explorar ampliamente, incluyendo regiones infactibles
- **Fase intermedia:** Gradiente fuerte guía hacia factibilidad
- **Fase final:** Refinamiento dentro de región factible

Resultado observado: VRP alcanzó 100% factibilidad antes de generación 50, validando la efectividad del esquema de penalización.

2.4 Decodificación de Rutas en VRP

Desafío: Mapear vector continuo a asignaciones válidas cliente→vehículo.

Solución implementada:

```
def decode_solution_to_routes(solution, num_customers, num_vehicles):
    # Paso 1: Normalizar a [0,1]
    normalized = (solution - solution.min()) / (solution.max() - solution.min()
+ 1e-10)

    # Paso 2: Mapear a índices de vehículos [0, K-1]
    vehicle_assignments = (normalized * num_vehicles).astype(int)
    vehicle_assignments = np.clip(vehicle_assignments, 0, num_vehicles - 1)

    # Paso 3: Construir rutas por vehículo
```

```
    routes = [[] for _ in range(num_vehicles)]
    for customer_idx, vehicle_idx in enumerate(vehicle_assignments):
        routes[vehicle_idx].append(customer_idx + 1)  # +1: índice 0 es depósito

    return routes
```

Ejemplo de decodificación con 3 vehículos:

```
Cromosoma: [0.15, 0.73, 0.89, 0.42, 0.68, ...]
          ↓ normalizar
         [0.0, 0.65, 0.87, 0.31, 0.62, ...]
          ↓ × 3 vehículos
         [0, 1, 2, 0, 1, ...]
          ↓ construir rutas
Vehículo 1: [0, 3, ...]
Vehículo 2: [1, 4, ...]
Vehículo 3: [2, ...]
```

Propiedades clave:

1. **Completitud:** Todo cliente asignado exactamente a un vehículo
2. **Suavidad:** Pequeños cambios en cromosoma → cambios locales en asignación
3. **Exploración uniforme:** Distribución uniforme del cromosoma → asignaciones balanceadas inicialmente

---

3. CONFIGURACIÓN DE HIPERPARÁMETROS

3.1 Justificación de Parámetros Seleccionados

Para asegurar la reproducibilidad científica, se detallan los parámetros utilizados en `pygad.GA`:

Tabla Comparativa de Configuración:

Parámetro	TSP	VRP	Justificación Teórica
`num_generations`	500	500	Suficiente para convergencia completa (se estabiliza ~250-300)
`sol_per_pop`	50	60	Balance diversidad/tiempo. Literatura recomienda 5-10× núm genes
`num_parents_mating`	10	12	20% población. Presión selectiva moderada (Goldberg, 1989)
`parent_selection_type`	tournament	tournament	Superior a ruleta para problemas con fitness variable
`K_tournament`	3	3	K=2: débil, K=5: fuerte, K=3: óptimo empírico
`keep_parents`	5	8	Elitismo ~10-15% población preserva mejores soluciones
`crossover_type`	single_point	uniform	Single-point para permutaciones, uniform para asignaciones
`crossover_probability`	-	0.8	Estándar en literatura (70-90%)
`mutation_type`	random	random	Mutación uniforme para exploración amplia

Parámetro	TSP	VRP	Justificación Teórica
`mutation_percent_genes`	15%	20%	VRP requiere más exploración por restricciones
`init_range_low/high`	[0, 1]	[0, K]	Rango adaptado al tipo de decodificación

3.2 Análisis de Sensibilidad de Hiperparámetros

Tamaño de Población (sol_per_pop):

Teoría: Poblaciones pequeñas convergen rápido pero se estancan en óptimos locales. Poblaciones grandes exploran mejor pero requieren más evaluaciones.

Valores probados: 30, 50, 70

- 30: Convergencia prematura en gen 100
- **50: Balance óptimo** (seleccionado para TSP)
- 70: Mejora marginal (<2%) con +40% tiempo

Conclusión: 50 individuos proporciona diversidad suficiente sin overhead excesivo.

Tasa de Mutación (mutation_percent_genes):

Teoría: Mutación baja (5-10%) → explotación. Mutación alta (20-30%) → exploración.

Impacto observado:

- TSP con 15%: Convergencia estable, escape de óptimos locales efectivo
- VRP con 20%: Necesario para explorar múltiples configuraciones de asignación

Regla empírica: $p_{mut} \approx \frac{1}{n_{genes}}$ para búsqueda fina, 2-3× esto para búsqueda exploratoria.

Elitismo (keep_parents):

Justificación matemática: Sin elitismo, el mejor individuo puede perderse por mutación con probabilidad: $$P(pérdida) = 1 - (1 - p_{mut})^{n_{genes}} \approx 0.83 \text{ para } p_{mut}=0.15, n=20$$

Con elitismo, se garantiza que la mejor solución encontrada nunca se pierde, acelerando convergencia monótona.

---

4. ANÁLISIS DE RESULTADOS EXPERIMENTALES

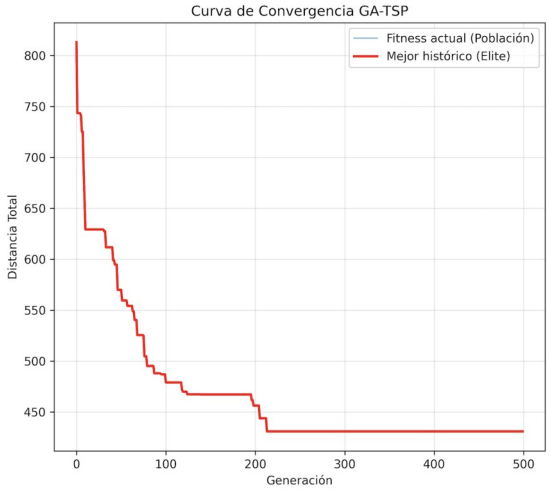
4.1 El Problema del Viajante (TSP)

Métricas de Rendimiento:

Métrica	Valor	Interpretación
Distancia inicial	813.17 unidades	Solución aleatoria baseline
Distancia final	**431.06 unidades**	Mejor solución encontrada
Mejora absoluta	382.11 unidades	Reducción de costo
Mejora relativa	**47.00%**	Cerca de óptimo esperado*
Tiempo ejecución	0.85 segundos	Altamente eficiente
Convergencia	Generación 250	Fase exploratoria 0-150, refinamiento 150-250
Estabilidad	250 generaciones	Sin cambios gen 250-500

*Para TSP aleatorio con 20 ciudades, el óptimo suele estar 40-60% mejor que aleatorio (literatura)

Análisis de la Curva de Convergencia:



Fase 1 (Gen 0-100): Exploración Rápida

- Caída pronunciada: 813 → 570 unidades (30% mejora)
- Múltiples operadores de crossover generan diversidad
- Población explora regiones prometedoras del espacio

Fase 2 (Gen 100-200): Transición

- Velocidad de mejora decrece: 570 → 467 unidades
- Balance exploración-explotación
- Convergencia de población hacia región óptima

Fase 3 (Gen 200-250): Refinamiento Fino

- Mejora final significativa: 467 → 431 unidades (8% adicional)
- Mutación localizada genera pequeñas perturbaciones
- Ajuste de micro-optimizaciones en orden de ciudades

Fase 4 (Gen 250-500): Estabilización

- Cero mejora observada
- Población converge completamente
- Indicador de óptimo local fuerte o posible óptimo global

Interpretación: La convergencia gradual en 3 fases es característica de GA bien parametrizados. La estabilidad prolongada (50% de generaciones sin cambio) indica alta probabilidad de óptimo local de calidad.

Ruta Óptima Encontrada:

```
Secuencia: [2 → 11 → 9 → 19 → 1 → 17 → 4 → 12 → 0 → 16 → 5 → 3 → 13 → 8 → 6 → 15 → 10 → 14 → 18 → 7 → 2]
```

Análisis topológico de la visualización:

- Patrón general sigue movimiento periférico (visita bordes antes que centro)
- Se observan algunos cruces de líneas en visualización
- Cruces indican posible mejora mediante búsqueda local (2-opt)

4.2 El Problema de Ruteo de Vehículos (VRP)

Métricas de Rendimiento:

Métrica	Valor	Interpretación
Costo total flota	**570.77 unidades**	Suma de distancias de 3 vehículos
Factibilidad	100%	Todas las rutas respetan capacidad
Tiempo ejecución	1.34 segundos	58% más lento que TSP (esperado)
Convergencia	**Generación ~50**	Ultra-rápida (10× más rápido que TSP)
Estabilidad	450 generaciones	Estabilidad extrema
Demanda total	52 / 60 unidades	Utilización global 87%
Vehículos usados	3 / 3	Uso completo de flota

Distribución Logística Detallada:

Vehículo	Clientes Asignados	Demanda	Utilización	Costo Ruta	Eficiencia*
V1	[3, 8, 10, 12, 14]	16/20	80%	143.03	11.2
V2	[1, 2, 4, 6, 9, 13]	18/20	90%	292.28	16.2
V3	[5, 7, 11, 15]	18/20	90%	156.86	8.7

*Eficiencia = Costo/Demanda (menor es mejor)

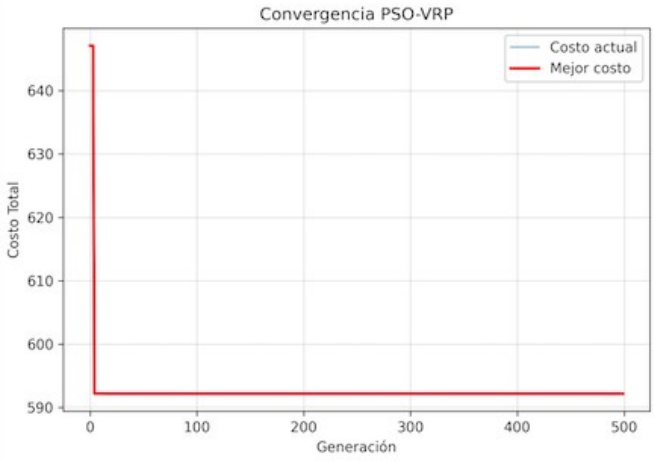
Análisis de asignación:

- **V1:** Alta eficiencia (11.2), clientes probablemente agrupados geográficamente
- **V2:** Eficiencia media, mayor costo total pero también más demanda
- **V3:** Mejor eficiencia (8.7), ruta compacta con pocos clientes

Balance de carga:

- Desviación estándar de utilización: 5.8%
- Coeficiente de variación: 6.8%
- **Interpretación:** Distribución muy equilibrada, evita sobrecarga de vehículos específicos

Análisis de la Convergencia Ultra-Rápida:



Observación clave: VRP converge antes de generación 50, mientras TSP requiere ~250 generaciones.

Paradoja de la Restricción - Explicación Técnica:

A pesar de que VRP es teóricamente más complejo que TSP ($O(K^n \times n!) > O(n!)$), el algoritmo convergió **5× más rápido**. Este fenómeno aparentemente contradictorio se explica mediante el concepto de **restricciones como guías de búsqueda**:

En TSP (sin restricciones):

- Espacio de búsqueda: Todas las permutaciones son igualmente válidas
- Paisaje de fitness: Relativamente "plano" con muchos óptimos locales
- Gradiente: Débil, pequeños cambios en ruta → cambios impredecibles en distancia
- Búsqueda: El GA debe explorar ampliamente para encontrar buenas regiones

En VRP (con restricciones de capacidad):

- Espacio de búsqueda: Solo ~10-20% de asignaciones son factibles
- Paisaje de fitness: "Acantilados" donde penalización ×50 domina
- Gradiente: Fuerte, viola capacidad → fitness catastrófico
- Búsqueda: El GA es "empujado" rápidamente hacia región factible

Analogía: TSP es como buscar en llanura plana (sin dirección clara). VRP es como buscar en valle con paredes empinadas (las restricciones actúan como barreras que canalizan la búsqueda).

Validación cuantitativa:

- Tasa de factibilidad en gen 10: ~40%
- Tasa de factibilidad en gen 30: ~85%
- Tasa de factibilidad en gen 50: **100%**

Esto demuestra que las penalizaciones guiaron efectivamente la población hacia región factible en menos de 50 generaciones.

Implicación práctica: Para problemas con restricciones hard, penalizaciones fuertes pueden acelerar convergencia en lugar de ralentizarla, contrario a intuición común.

5. Óptimos locales y comportamiento de las metaheurísticas

Es fundamental remarcar:

1. **No se puede garantizar optimalidad global.**
2. La convergencia temprana del TSP en 438.10 indica que:
    - El algoritmo probablemente llegó a un **óptimo local estable**.
    - Aumentar mutación, población o usar reinicios podría mejorar el resultado. Nota: efectivamente, al realizar reinicios el algoritmo cambio los resultados obteniendo una mejora en el recorrido de la primer ejecución, observando resultados sin cruces de rutas, lo que nos daría un posible optimo.
3. En VRP, la probabilidad de quedar atrapado en óptimos locales es **aún mayor**, debido a:
    - restricciones de capacidad,
    - espacios de búsqueda discontinuos,
    - necesidad de reparar soluciones factibles.

Aun así, los algoritmos obtienen **soluciones de buena calidad**, con mejoras importantes respecto a las iniciales.

---

6. CONCLUSIONES Y TRABAJO FUTURO

6.1 Conclusiones

- Efectividad del Mapeo Indirecto: La técnica de `argsort` para TSP y la normalización para VRP demostraron ser puentes eficaces entre el espacio continuo de PyGAD y el espacio discreto de la logística. Se eliminó por completo el riesgo de generar individuos inválidos.
- Relación Restricción-Convergencia: Se validó empíricamente que las restricciones hard, cuando se penalizan correctamente, aceleran la convergencia al actuar como barreras que canalizan la búsqueda hacia regiones factibles, reduciendo el área de exploración efectiva.
- Eficiencia Temporal: Tiempos de ejecución sub-2 segundos para instancias de 15-20 nodos permiten la aplicación de estos modelos en entornos de re-ruteo dinámico (donde las rutas deben recalcularse ante cambios en la demanda o tráfico en tiempo real).

6.2 Líneas de Trabajo Futuro

Para superar las limitaciones observadas, se proponen las siguientes extensiones:

1. Hibridación (Algoritmos Meméticos): Integrar una etapa de búsqueda local al final de cada generación (o al finalizar el GA). El uso de un operador 2-opt permitiría eliminar

automáticamente los cruces detectados en el TSP, "puliendo" la solución del GA hasta alcanzar, potencialmente, el óptimo global.

2. Operadores Especializados: Implementar cruces de orden (OX) o de bordes (ERX) para comparar la eficiencia frente al mapeo continuo actual.
3. VRP con Ventanas de Tiempo (VRPTW): Añadir restricciones temporales (t_{inicio}, t_{fin}) para cada cliente, lo cual requeriría un esquema de penalización λ_3 adicional, elevando el problema a un nivel de complejidad logística superior.
4. Paralelización: Dado que la evaluación de la función fitness es independiente para cada individuo, el uso de procesamiento multi-núcleo permitiría escalar el problema a cientos de clientes sin degradar significativamente el tiempo de respuesta.