NeuroGen: Una Arquitectura Híbrida de Aprendizaje por Refuerzo Profundo para la Inicialización de Poblaciones y Algoritmos Genéticos para el Refinamiento

Natalia Andrea García Ríos
Facultad de Ingeniería
Universidad de Antioquia
Medellín, Colombia
natalia.garcia9@udea.edu.co

***Abstract*—Este trabajo presenta NeuroGen, una arquitectura híbrida para resolver el Problema de Enrutamiento de Vehículos Capacitado (CVRP). La estrategia combina Aprendizaje por Refuerzo Profundo (DRL) basado en redes Actor-Crítico con Algoritmos Genéticos (GA). El componente DRL emplea una arquitectura de red de puntero (Pointer Network) para generar poblaciones iniciales de alta calidad, solucionando el problema del arranque en frío de las metaheurísticas tradicionales. Posteriormente, el GA refina estas soluciones mediante operadores especializados para permutaciones (Order Crossover, 2-Opt). Se implementaron tres agentes especializados (Junior, Mid, Expert) entrenados mediante Curriculum Learning para instancias de 15 a 150 nodos. Los experimentos sobre 120 instancias sintéticas demuestran que la inicialización neuronal mejora el costo de solución en 40% comparado con inicialización aleatoria, validando el paradigma de Learnheuristics en problemas de optimización combinatoria.**

***Index Terms*—CVRP, Deep Reinforcement Learning, Algoritmos Genéticos, Pointer Networks, Learnheuristics, Optimización Combinatoria**

I. Definición del Problema, Supuestos e Información Requeridos

A. Definición del Problema

El **Enrutamiento de Vehículos Capacitado (CVRP)** es un problema de optimización combinatoria *NP-Hard* que generaliza el conocido **Problema del Viajero (TSP)**. Formalmente, el problema se define sobre un grafo completo no dirigido $G = (V, E)$, donde $V = \{0, 1, ..., n\}$ representa el conjunto de nodos. El nodo 0 denota el depósito y el conjunto $V_c = \{1, ..., n\}$ representa los n clientes a servir.

Cada cliente $i \in V_c$ tiene una demanda asociada $d_i > 0$ y una ubicación geográfica fija. Existe una flota de vehículos homogénea, donde cada vehículo posee una capacidad idéntica Q. El costo de viajar entre el nodo i y el nodo j está dado por la distancia euclidiana c_{ij}.

El objetivo matemático es determinar un conjunto de rutas $R = \{r_1, r_2, ..., r_k\}$ que minimice la función de costo total:

$$\min Z = \sum_{k} \sum_{(i,j) \in r_k} c_{ij}$$

Sujeto a las siguientes restricciones fundamentales: (1) cada cliente $i \in V_c$ debe ser visitado exactamente una vez por un solo vehículo; (2) todas las rutas deben comenzar y terminar en el nodo depósito (0); (3) la suma de las demandas de los clientes en cualquier ruta r_k no puede exceder la capacidad del vehículo ($\sum_{i \in r_k} d_i \leq Q$); y (4) se asume una disponibilidad ilimitada de vehículos, aunque el objetivo secundario implícito es minimizar su uso mediante la eficiencia de las rutas.

B. Supuestos del Modelo

Para modelar el problema dentro de la arquitectura propuesta, se establecen los siguientes supuestos: **Flota Homogénea** (todos los vehículos tienen la misma capacidad máxima Q), **Información Estática y Determinista** (la ubicación de los clientes y sus demandas son conocidas a priori y no varían durante la ejecución), **Costo Simétrico** (el costo de transporte es proporcional a la distancia euclidiana 2D, cumpliendo que $c_{ij} = c_{ji}$) e **Indivisibilidad de Demanda** (la demanda de un cliente no es fraccionable; debe ser satisfecha en una única visita).

C. Información Requerida

Para validar el modelo, el sistema requiere instancias que contengan: coordenadas cartesianas (x, y) para el depósito y los clientes, la demanda d_i de cada nodo y la capacidad Q de cada vehículo. La implementación soporta la generación de instancias sintéticas de *distribución uniforme (Random)* o *agrupada (Clustered)*, lo cual es vital para probar la robustez del modelo ante diferentes topologías geográficas.

II. Descripción de la Estrategia de Solución

La estrategia implementada, denominada **NeuroGen**, es un algoritmo híbrido secuencial que combina la capacidad de aprendizaje de patrones del *Deep Reinforcement Learning* (DRL) con la capacidad de refinamiento global de los Algoritmos Genéticos (GA). Esta arquitectura aborda el problema del "arranque en frío" de las metaheurísticas, reemplazando la

inicialización aleatoria por soluciones construidas inteligentemente. El **Algoritmo 1** presenta el pseudocódigo de la estrategia global.

Algorithm 1 NeuroGen: Arquitectura Híbrida DRL-GA

Require: Instancia CVRP I, Agente DRL entrenado π_θ, Configuración GA
Ensure: Mejor solución S^*

// Fase 1: Inicialización Inteligente (DRL)
$P_0 \leftarrow \emptyset$
for $i = 1$ **to** N **do**
  $\epsilon \leftarrow 0.3$ *// Diversidad estocástica*
  $s \leftarrow \texttt{reset}(I)$ *// Estado inicial*
  $tour \leftarrow []$
  while no todos los clientes visitados **do**
    $mask \leftarrow \texttt{get\_valid\_actions}(s)$
    $probs \leftarrow \pi_\theta(s, mask)$
    $a \leftarrow \texttt{sample}(probs, \epsilon)$
    $tour.\texttt{append}(a)$
    $s \leftarrow \texttt{step}(a)$
  end while
  $ind \leftarrow \texttt{from\_giant\_tour}(tour, I)$
  $P_0 \leftarrow P_0 \cup \{ind\}$
end for
// Fase 2: Refinamiento Metaheurístico (GA)
$P \leftarrow P_0$
$S^* \leftarrow \arg\min_{ind \in P} fitness(ind)$
for $g = 1$ **to** G_{max} **do**
  $P' \leftarrow$ élite de P (mejores E individuos)
  while $|P'| < N$ **do**
    $p_1, p_2 \leftarrow \texttt{tournament\_select}(P, k = 3)$
    $off \leftarrow \texttt{order\_crossover}(p_1, p_2)$ con prob. P_c
    $off \leftarrow \texttt{two\_opt\_mutation}(off)$ con prob. P_m
    $P' \leftarrow P' \cup \{off\}$
  end while
  $P \leftarrow P'$
  if $\min_{ind \in P} fitness(ind) < fitness(S^*)$ **then**
    $S^* \leftarrow \arg\min_{ind \in P} fitness(ind)$
  end if
end for
return S^*

A. Representación de la Solución: Paradigma Genotipo-Fenotipo

El desacoplamiento de la complejidad del CVRP se logra mediante una representación indirecta basada en el concepto de *Giant Tour*. El **Genotipo** consiste en una permutación simple de todos los clientes $S = \{c_1, c_2, ..., c_n\}$ sin delimitadores de ruta. El **Fenotipo** es el conjunto de rutas factibles $R = \{r_1, ..., r_k\}$ que respetan la capacidad Q. La **Decodificación (Split)** es un algoritmo determinista que recorre la permutación S e inserta cortes al depósito cada vez que la capacidad acumulada alcanza Q, garantizando la factibilidad de cualquier permutación generada. Esta representación permite que tanto el DRL como el GA trabajen sobre un espacio de búsqueda más simple (permutaciones), delegando la construcción de rutas factibles al decodificador.

B. Fase 1: Inicialización Inteligente (DRL)

El componente de aprendizaje automático emplea una arquitectura **Actor-Crítico** con redes de puntero (Pointer Networks) para aprender una política estocástica π que construye permutaciones paso a paso. El **Actor** (Red de Puntero) consiste en un Codificador LSTM de 2 capas (embedding de 256 dimensiones) que procesa las características de los nodos (x, y, d_i) y un decodificador LSTM con mecanismo de atención que calcula probabilidades de selección para cada cliente candidato. El **Crítico** es una red densa que estima el valor del estado $V(s)$ para reducir la varianza del gradiente durante el entrenamiento mediante el cálculo de la ventaja (*Advantage*).

Reconociendo la variación en complejidad según la escala, se entrenaron tres agentes especializados: **Junior** (15–50 nodos, última milla), **Mid** (50–100 nodos, distribución regional) y **Expert** (100–150 nodos, escenarios industriales complejos). El agente Expert se entrenó mediante *Curriculum Learning* en tres fases progresivas de dificultad creciente. Durante la inferencia, se aplica **enmascaramiento dinámico** en la capa de salida para bloquear acciones que violarían la restricción de capacidad, y se introduce un parámetro de temperatura $\epsilon = 0.3$ que controla la estocasticidad del muestreo para generar poblaciones diversas.

C. Fase 2: Refinamiento Metaheurístico (GA)

El algoritmo genético recibe la población P_0 generada por el DRL y ejecuta un proceso de mejora iterativa. Los **operadores genéticos** empleados son: **Selección** por torneo ($k = 3$) para mantener presión selectiva constante, **Cruce** mediante *Order Crossover* (OX) para preservar secuencias relativas de clientes heredadas del DRL, y **Mutación** 2-Opt como búsqueda local para eliminar cruces de aristas, aplicada sobre una ruta individual seleccionada aleatoriamente. La **estrategia de reemplazo** sigue un esquema Generacional con Elitismo: la población completa se reemplaza en cada generación, pero los $E = 2$ mejores individuos se preservan intactos, garantizando convergencia monotónica.

D. Parámetros de la Metaheurística

La **Tabla I** presenta los parámetros configurados para los experimentos. Es importante destacar que la implementación del software desarrollado permite la selección flexible de diferentes configuraciones, incluyendo otros métodos de selección (ruleta), operadores de cruce alternativos (PMX) y diferentes estrategias de mutación (swap, insert, inversion). Los valores presentados corresponden a la configuración específica utilizada en la validación experimental.

III. Descripción de la Implementación

A. Acceso al Código Fuente

El código fuente completo del proyecto está disponible públicamente en los siguientes repositorios de GitHub:

TABLE I
CONFIGURACIÓN DEL ALGORITMO GENÉTICO PARA EXPERIMENTOS

Parámetro	Valor	Descripción
Tamaño Población (N)	50	Individuos por generación
Generaciones (G_{max})	100	Criterio de parada
Prob. Cruce (P_c)	0.85	Probabilidad de aplicar OX
Prob. Mutación (P_m)	0.15	Probabilidad de 2-Opt
Tamaño Torneo (k)	3	Presión selectiva
Elitismo (E)	2	Individuos preservados

- **Backend** (Python/FastAPI/PyTorch): *github.com/natandreli/CVRP-DRL-GA-backend*
- **Frontend** (React/Tailwind/Recharts): *github.com/natandreli/CVRP-DRL-GA-frontend*

Los repositorios incluyen instrucciones de instalación, documentación de la API REST, notebooks de entrenamiento de los agentes DRL, y datasets de instancias sintéticas utilizadas en los experimentos.

B. Stack Tecnológico

La arquitectura sigue un patrón cliente-servidor desacoplado. El **Backend** (Python 3.10) implementa el núcleo de procesamiento con **PyTorch** (aceleración GPU via CUDA) para el entrenamiento del DRL, y expone servicios mediante **FastAPI** con validación de esquemas vía *Pydantic*. El **Frontend** (React + Tailwind) proporciona una interfaz de experimentación con visualización gráfica usando **Recharts**.

C. Arquitectura Modular del Sistema

El sistema se organiza en tres capas principales interconectadas: **Capa de Entorno** (simulación del CVRP), **Capa de Agentes** (DRL y GA), y **Capa de Utilidades** (cálculos compartidos). La **Figura 1** conceptualiza esta arquitectura y el flujo de datos entre componentes.

El flujo de ejecución sigue la siguiente secuencia: (1) una instancia CVRP ingresa al sistema; (2) el módulo `population_generator` invoca al agente DRL para generar N soluciones; (3) cada solución se convierte a objeto `Individual` mediante el decodificador Split; (4) la población se inyecta al `GeneticAlgorithm`; (5) el GA ejecuta el ciclo evolutivo; y (6) se retorna la mejor solución encontrada.

D. Componente DRL: Actor-Crítico con Pointer Networks

La clase **CVRPEnvironment** implementa un entorno estilo OpenAI Gym que simula el proceso secuencial de construcción de soluciones. El estado se representa como un vector que incluye: coordenadas normalizadas de la ubicación actual, capacidad utilizada del vehículo, y para cada cliente un conjunto de características (visitado, demanda, distancia al vehículo, coordenadas). El método `get_valid_actions()` retorna los clientes que pueden visitarse sin violar la restricción de capacidad, y `step(action)` ejecuta la transición de estado, calculando la recompensa como la distancia negativa recorrida.

La clase **ActorCriticNetwork** encapsula la arquitectura dual. El Actor es una instancia de `PointerNetwork` que utiliza un codificador LSTM para procesar las características de todos los clientes y un decodificador LSTM con mecanismo de atención que genera una distribución de probabilidad sobre los clientes candidatos. El Crítico es una red feedforward independiente que estima $V(s)$ para el cálculo de la ventaja durante el entrenamiento.

La clase **ActorCriticAgent** orquesta el entrenamiento implementando el algoritmo REINFORCE con baseline. El flujo de entrenamiento ejecuta: (1) **Rollout** donde el agente construye una solución completa almacenando la trayectoria (s_t, a_t, r_t); (2) **Cálculo de Returns** mediante retornos descontados $G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$; y (3) **Actualización** minimizando $\mathcal{L}_c = (G_t - V(s_t))^2$ para el Crítico y maximizando $\mathcal{L}_a = \log \pi(a_t|s_t) \cdot A_t + \beta H(\pi)$ para el Actor, donde $A_t = G_t - V(s_t)$ es la ventaja y $H(\pi)$ es el bono de entropía.

La **Tabla II** especifica los protocolos de entrenamiento diferenciados para cada agente especializado.

TABLE II
PROTOCOLOS DE ENTRENAMIENTO POR AGENTE

Parámetro	Junior	Mid	Expert
Rango de Nodos	15–50	40–100	80–150
Episodios	6,000	10,000	25,000
LR Actor	1×10^{-3}	4×10^{-4}	3×10^{-4}
LR Crítico	5×10^{-4}	2×10^{-4}	1×10^{-5}
Mix Random/Cluster	60%/40%	50%/50%	20%/80%
Curriculum Learning	No	No	Sí (3 fases)

E. Componente GA: Motor Evolutivo

La clase **Individual** representa una solución CVRP como lista de rutas (secuencias de IDs de clientes). Implementa evaluación perezosa del fitness mediante caching, y el método estático `from_giant_tour()` realiza la decodificación Split: recorre la permutación e inserta cortes cuando la capacidad acumulada excede Q.

La clase **GeneticAlgorithm** controla el proceso evolutivo. El método `initialize_population()` acepta una población customizada (inyección DRL) o genera población aleatoria. El método `run_generation()` implementa el ciclo: preservar élite, generar offspring mediante selección $\rightarrow$ cruce $\rightarrow$ mutación, y reemplazar población.

El módulo **operators.py** implementa múltiples alternativas para cada operador: selección por torneo y ruleta; cruce OX y PMX; mutación swap, insert, inversion y 2-opt. Esta modularidad permite experimentación con diferentes configuraciones sin modificar el código del GA.

F. Integración Híbrida

El módulo `population_generator.py` implementa la función `generate_population_with_drl()` que constituye el puente entre ambos paradigmas. Recibe un agente DRL entrenado y un parámetro de diversidad, ajusta temporalmente el ϵ del agente para forzar exploración estocástica,

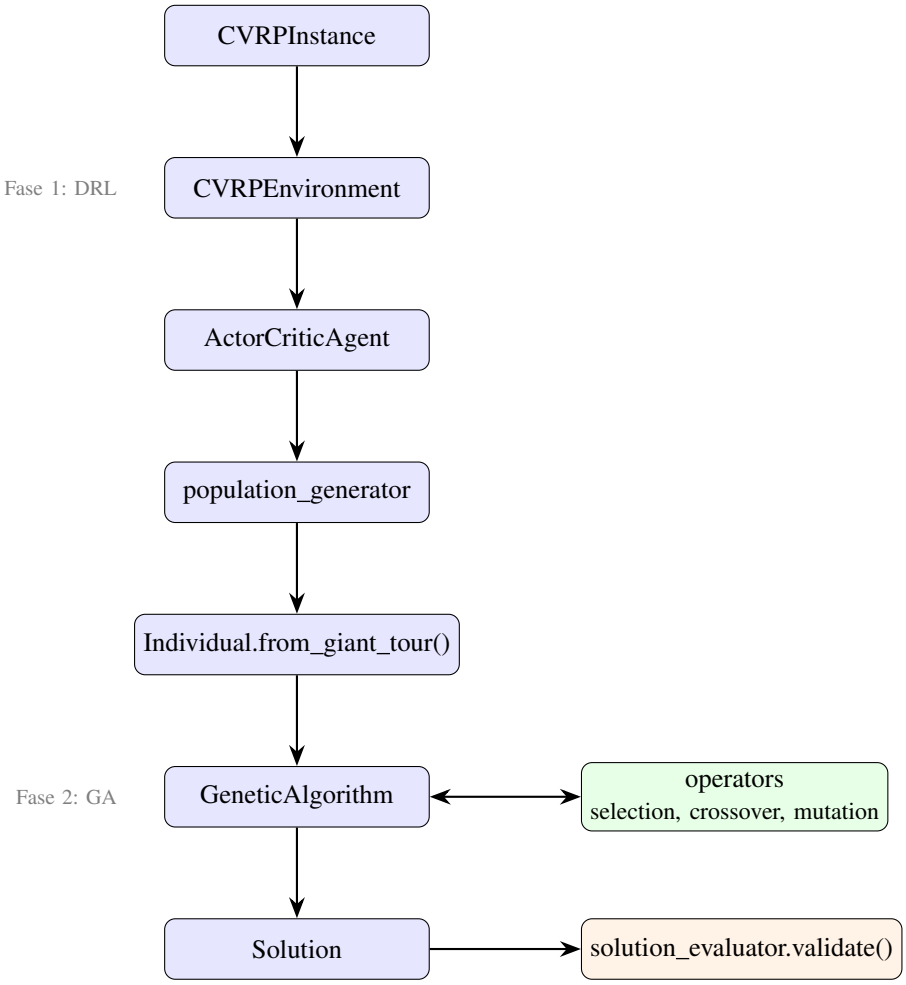

Fig. 1. Arquitectura del sistema NeuroGen y flujo de datos entre componentes.

genera N soluciones mediante muestreo de la política, extrae la secuencia de clientes de cada solución, y construye objetos `Individual` via `from_giant_tour()`. Esta población se inyecta directamente al constructor del GA, evitando la inicialización aleatoria tradicional.

IV. Diseño de Experimentos Computacionales

El diseño experimental tiene como objetivo validar dos hipótesis: **(H1)** La inicialización neuronal mejora significativamente el desempeño del GA comparado con inicialización aleatoria, y **(H2)** Los agentes especializados son más efectivos en sus rangos de entrenamiento específicos.

A. Generación de Instancias de Prueba

Se generaron instancias sintéticas estratificadas en tres rangos de complejidad: **Rango Junior (Small)** con 20, 30, 40, 50 nodos; **Rango Mid (Medium)** con 60, 75, 90, 100 nodos; y **Rango Expert (Large)** con 110, 125, 140, 150 nodos. Para cada tamaño se crearon 10 instancias (50% distribución uniforme Random, 50% estructura Clustered con $k = 4$ clusters), con capacidad Q calibrada para forzar un número esperado de rutas proporcional a $\sqrt{n}$. **Total:** 120 instancias (12 tamaños $\times$ 10 réplicas).

B. Configuraciones Experimentales

Para cada instancia se ejecutaron cuatro configuraciones: (1) **GA Puro (Baseline)** con inicialización aleatoria; (2) **DRL-Junior + GA**; (3) **DRL-Mid + GA**; y (4) **DRL-Expert + GA**. En las configuraciones híbridas, la población inicial se genera con $\epsilon = 0.3$ para diversidad. Todos los agentes se ejecutaron sobre todas las instancias para evaluar la transferibilidad del aprendizaje.

C. Metodología de Ajuste de Parámetros

Los parámetros tanto del DRL como del GA se ajustaron mediante **experimentación empírica iterativa**. Las tasas de aprendizaje se ajustaron observando las curvas de pérdida y convergencia durante entrenamientos preliminares, priorizando valores conservadores para el agente Expert tras observar inestabilidad con tasas más altas. Los valores de $P_c = 0.85$ y $P_m = 0.15$ se seleccionaron basándose en recomendaciones de la literatura especializada [2].

D. Protocolo de Ejecución

Cada configuración se ejecutó 5 veces por instancia para controlar la variabilidad estocástica, resultando en 120 instancias $\times$ 4 configuraciones $\times$ 5 réplicas = **2,400 ejecuciones experimentales**. El hardware utilizado consistió en un procesador AMD Ryzen 7 9700X y GPU NVIDIA RTX 3090. Las semillas aleatorias se controlaron mediante el parámetro `seed` para garantizar reproducibilidad.

V. Resultados y Análisis

A. Validación de Hipótesis H1: Superioridad de la Inicialización Neuronal

La **Figura 2** presenta la comparación agregada de costos finales promedio entre las cuatro configuraciones experimentales.

Los resultados confirman categóricamente la hipótesis H1. Las tres configuraciones híbridas superan consistentemente al baseline en todos los rangos, con mejoras especialmente pronunciadas en instancias grandes (110–150 nodos), donde la inicialización neuronal reduce el costo final en aproximadamente 45%. La **Tabla III** cuantifica estos resultados.

TABLE III
Costo Final Promedio ($\pm$ Desv. Est.) por Configuración

Configuración	Small	Medium	Large	Global
GA Puro	569.4±281.3	1521.5±545.7	2916.8±817.4	1669.3
DRL-Jr + GA	432.3±190.3	980.2±312.8	1596.5±382.6	1003.0
DRL-Mid + GA	**428.6**±180.5	**943.0**±287.3	**1555.9**±355.4	**975.8**
DRL-Ex + GA	470.2±195.4	1015.3±295.9	1634.6±381.9	1040.0

El costo global promedio del mejor híbrido (DRL-Mid+GA: 975.8) representa una mejora del 41.5% respecto al baseline (GA Puro: 1669.3). Las desviaciones estándar de las configuraciones híbridas son consistentemente menores, indicando mayor robustez y predictibilidad.

B. Validación de Hipótesis H2: Especialización de Agentes

La **Figura 3** visualiza la competencia directa entre agentes en cada rango.

Los resultados presentan un hallazgo inesperado: el agente **DRL-Mid** emerge como el mejor desempeño en los tres rangos, no solo en su rango nativo. En el rango Small, DRL-Junior (432.3) y DRL-Mid (428.6) prácticamente empatan (0.9% diferencia). En el rango Large, DRL-Mid (1555.9) supera incluso al DRL-Expert (1634.6) por 5.1%.

C. Análisis de Convergencia

La **Figura 4** ilustra el comportamiento dinámico del proceso evolutivo.

La separación entre curvas es máxima en la generación 0, evidenciando el impacto directo de la inicialización neuronal. Las bandas de confianza son más estrechas para las configuraciones híbridas, confirmando mayor consistencia. La **Tabla IV** cuantifica las métricas de evolución.

TABLE IV
Métricas de Evolución y Tiempos de Ejecución

Config.	Gap (%)	Gen.	T. Init	T. Evol	Total
GA Puro	43.2±6.3	37±21	0.1s	8.8s	8.9s
DRL-Jr+GA	21.4±7.1	38±23	149.7s	9.0s	158.6s
DRL-Mid+GA	20.1±7.2	39±22	146.9s	8.9s	155.9s
DRL-Ex+GA	21.2±7.8	40±21	142.9s	8.9s	151.9s

El Gap de mejora del GA Puro (43.2%) es el doble que el de las configuraciones híbridas ($\sim$21%). Esto *no* indica inferioridad del método híbrido, sino que las configuraciones DRL ya parten de soluciones tan buenas que el margen de mejora restante es menor.

D. Análisis de Costo Computacional

La **Figura 5** descompone el tiempo de ejecución en inicialización y evolución.

El overhead total es de aproximadamente 17x respecto al GA Puro. Sin embargo, el tiempo de entrenamiento offline se amortiza sobre múltiples problemas, y para aplicaciones donde la calidad de solución es crítica (reducción de 40% en costo operativo), el overhead de 150 segundos resulta insignificante.

E. Discusión

1) Fortalezas del Enfoque Híbrido: El método propuesto demuestra reducción consistente de 40–45% en costo de solución, menor varianza en resultados (mayor predictibilidad operativa), transferibilidad razonable entre rangos de instancias, y ventaja creciente con el tamaño del problema.

2) Análisis del Sobreentrenamiento del Agente Expert: El hallazgo más significativo es que el agente Mid supera al Expert incluso en instancias grandes (110–150 nodos), contradiciendo la hipótesis de especialización. Se proponen dos explicaciones complementarias. Primero, **overfitting a patrones de entrenamiento**: con 25,000 episodios (vs 10,000 del Mid), el Expert pudo memorizar patrones específicos de las instancias de entrenamiento, perdiendo capacidad de generalización. Segundo, **balance exploración-explotación**: el Mid, entrenado en un rango intermedio (40–100 nodos), desarrolló heurísticas más transferibles que escalan tanto hacia problemas más pequeños como más grandes.

Este resultado tiene implicaciones prácticas importantes: entrenar múltiples agentes especializados puede ser contraproducente, y un único agente bien calibrado en rango medio podría ser suficiente para aplicaciones reales.

3) Limitaciones: El overhead computacional de la inicialización DRL ($\sim$150s) puede ser prohibitivo para aplicaciones de respuesta inmediata. La convergencia del GA no se acelera; solo mejora el punto de partida. Los experimentos se limitaron a instancias sintéticas, pendiente validación con benchmarks estándar.

4) Trabajo Futuro: Se propone integrar operadores de búsqueda local más agresivos como 3-Opt o Lin-Kernighan, implementar operadores adaptativos basados en diversidad

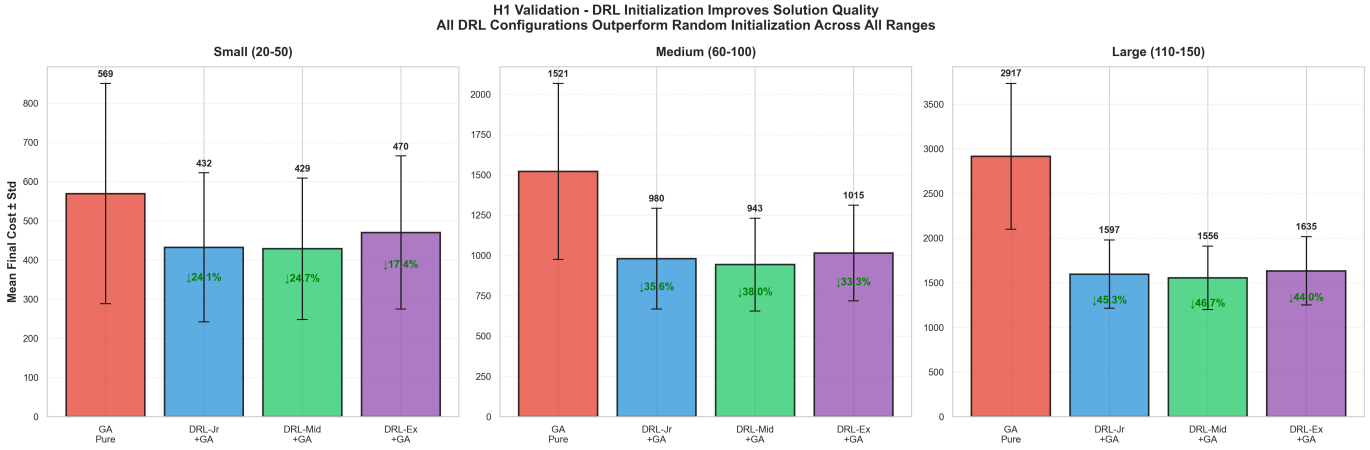

Fig. 2. Validación H1: Las configuraciones híbridas superan al GA puro en todos los rangos, con mejoras del 24% (Small), 36–38% (Medium) y 44–46% (Large).

Fig. 3. Validación H2: DRL-Mid domina en todos los rangos, cuestionando parcialmente la hipótesis de especialización estricta.

poblacional, explorar arquitecturas de Graph Neural Networks para mejor escalabilidad, y validar con instancias estándar de VRPLib.

VI. Conclusiones

Este trabajo presentó **NeuroGen**, una arquitectura híbrida que integra Aprendizaje por Refuerzo Profundo con Algoritmos Genéticos para resolver el CVRP. Los resultados experimentales sobre 120 instancias sintéticas (2,400 ejecuciones) validan la efectividad del enfoque propuesto.

La inicialización neuronal mejora la calidad de solución en 40–45% comparada con inicialización aleatoria tradicional, **confirmando H1**. El agente Mid demostró el mejor balance entre especialización y generalización, **matizando H2** y sugiriendo que la especialización extrema puede ser contraproducente debido a sobreentrenamiento. El método híbrido produce soluciones más robustas y escalables, con overhead computacional compensado por la mejora en calidad.

La arquitectura NeuroGen demuestra que la integración de técnicas de aprendizaje automático con metaheurísticas tradicionales es altamente beneficiosa para problemas de optimización combinatoria complejos. El paradigma de *aprender a inicializar* resulta más efectivo que el *aprender a resolver*, permitiendo que el GA explote su capacidad de búsqueda global partiendo de regiones prometedoras del espacio de soluciones.

References

[1] Lu, Y., Yuan, Y., Yasenjiang, J., Sitahong, A., Chao, Y., & Wang, Y. (2025). An Optimized Method for Solving the Green Permutation Flow Shop Scheduling Problem Using a Combination of Deep Reinforcement Learning and Improved Genetic Algorithm. *Mathematics*, 13(4), 545.

[2] Gendreau, M., & Potvin, J. Y. (2010). *Handbook of metaheuristics* (Vol. 2). New York: Springer.

[3] Vinyals, O., Fortunato, M., & Jaitly, N. (2015). Pointer networks. *Advances in neural information processing systems*, 28.

[4] Nazari, M., Oroojlooy, A., Snyder, L. V., & Takác, M. (2018). Reinforcement learning for solving the vehicle routing problem. *Advances in neural information processing systems*, 31.

[5] Prins, C. (2004). A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research*, 31(12), 1985-2002.

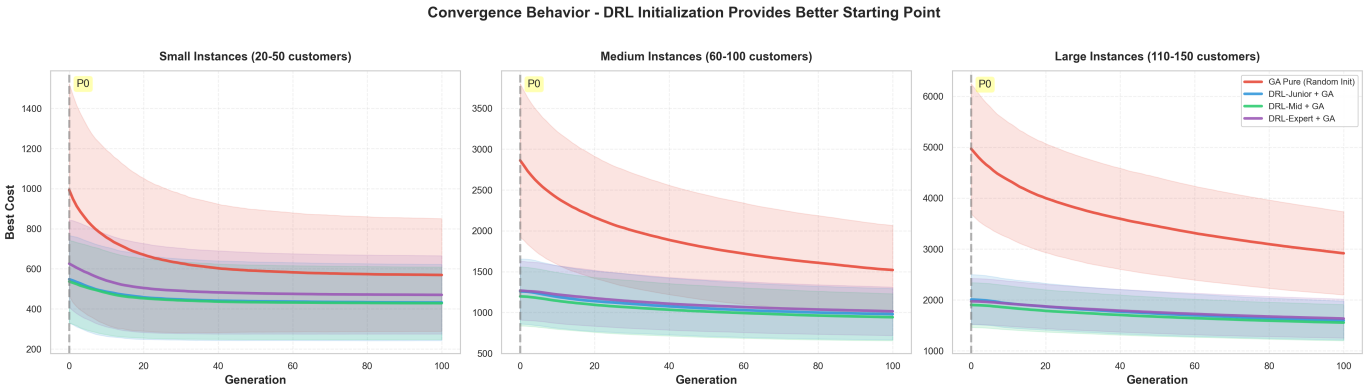

Fig. 4. Curvas de convergencia: Las configuraciones DRL parten de P_0 significativamente superior, convergiendo a mejores soluciones finales.

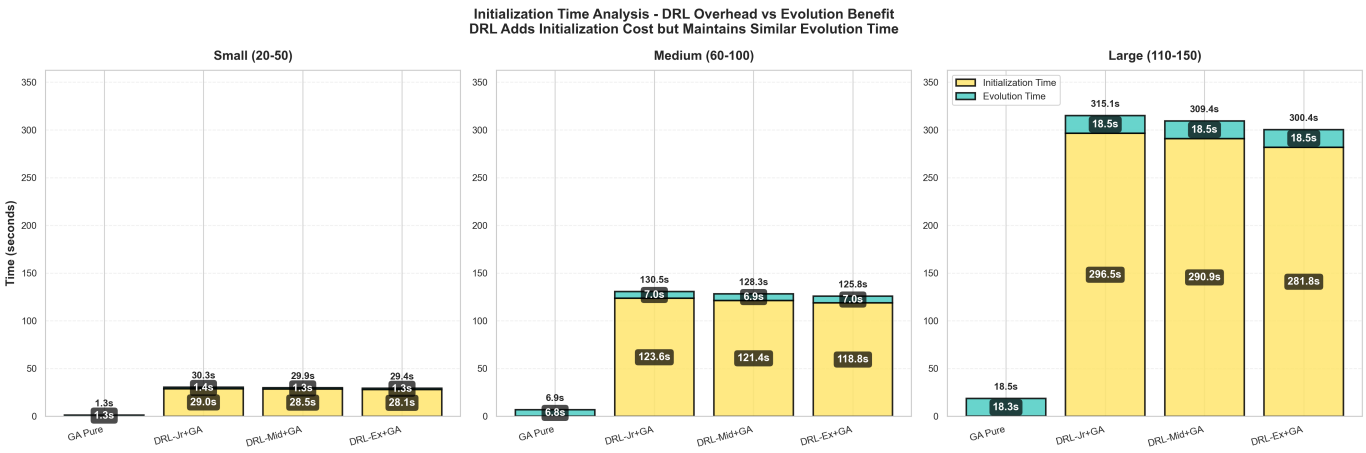

Fig. 5. Análisis de tiempos: El DRL añade overhead de inicialización pero el tiempo de evolución del GA se mantiene constante.