Diseño de un entorno de navegación urbana basado en grafos con waypoints para aprendizaje por refuerzo

Resumen

Se presenta el diseño e implementación de un entorno personalizado para *Reinforcement Learning* (RL) denominado `WaypointNavigationEnv`, construido sobre `Gymnasium` y `NetworkX`. Este entorno modela un sistema de navegación sobre grafos dirigido a simular trayectorias urbanas con puntos intermedios (*waypoints*), destino fijo, y control de penalizaciones por ciclos o acciones redundantes. La arquitectura permite escalar desde entornos pequeños hasta simulaciones urbanas amplias mediante técnicas de enmascaramiento de acciones, embeddings de nodos y métricas de progreso derivadas de caminos más cortos.

1. Introducción

En problemas de movilidad y planificación urbana, los agentes de aprendizaje por refuerzo deben operar en entornos donde las decisiones locales tienen consecuencias globales sobre tiempo, distancia y eficiencia del recorrido. Modelar estos escenarios requiere más que un simple grafo: se necesitan estructuras que integren representación espacial, restricciones topológicas y señales de recompensa que reflejen comportamiento inteligente.

El entorno `WaypointNavigationEnv` surge de esta necesidad. Se introduce una representación continua del espacio mediante *embeddings nodales*, métricas de progreso basadas en Dijkstra y una arquitectura modular que permite extender el modelo a grafos reales obtenidos de redes urbanas (por ejemplo, OpenStreetMap).

2. Analogía conceptual y dinámica de aprendizaje del algoritmo PPO

El algoritmo *Proximal Policy Optimization* (PPO) puede entenderse como un proceso de aprendizaje por refuerzo en el que el agente intenta mejorar su política de decisión sin dar pasos demasiado grandes que lo hagan "olvidar" lo que ya sabía.

Una forma natural de visualizarlo es imaginar un conductor aprendiendo a moverse por una ciudad compleja. Cada día repite su ruta, pero ajusta ligeramente su comportamiento: acelera un poco antes de una curva, toma una calle alternativa, o evita un embotellamiento que detectó previamente. PPO replica exactamente este comportamiento: experimenta, evalúa el resultado y ajusta la política solo dentro de una región controlada.

Aplicado al entorno `WaypointNavigationEnv`, el agente PPO aprende patrones de navegación eficientes observando cómo sus acciones afectan el tiempo de viaje, el progreso hacia el destino y las penalizaciones por ciclos. En términos intuitivos, "aprende a conducir con propósito", recordando lo que funciona y corrigiendo gradualmente lo que no, como un conductor que perfecciona su ruta diaria con experiencia acumulada.

2.1. Reutilización de experiencias y generalización

El algoritmo PPO no memoriza trayectorias exactas, sino que abstrae patrones de comportamiento a partir de múltiples recorridos. Cada episodio produce una secuencia de experiencias (s_t, a_t, r_t, s_{t+1}) que se almacenan en un buffer de *rollouts*. Durante la fase de actualización, el agente estima las ventajas $\hat{A}_t$ para cada transición y ajusta su política global de modo que aumente la probabilidad de aquellas acciones que demostraron ser mejores de lo esperado.

De esta manera, las experiencias locales —por ejemplo, cómo aproximarse a un waypoint o evitar un ciclo en un barrio determinado— se transforman en conocimiento generalizable aplicable a otras zonas del grafo. El agente no recuerda posiciones específicas, sino *principios de navegación*: cuándo avanzar, cuándo retroceder y cómo minimizar el costo de viaje.

2.2. Ejemplo en concreto

Supongamos que el agente se encuentra en un cruce con tres caminos. Uno de ellos resulta más directo y requiere pasar por menos intersecciones. Tomar cualquiera de los otros implica atravesar más nodos, lo que se traduce en un mayor costo. A partir de esta experiencia, el agente aprende que, ante situaciones similares —aunque ocurran en distintos lugares o con pesos diferentes—, puede generalizar su comportamiento. Al normalizar los valores obtenidos, identifica un patrón que le indica cuál es la mejor decisión al enfrentarse a un cruce.

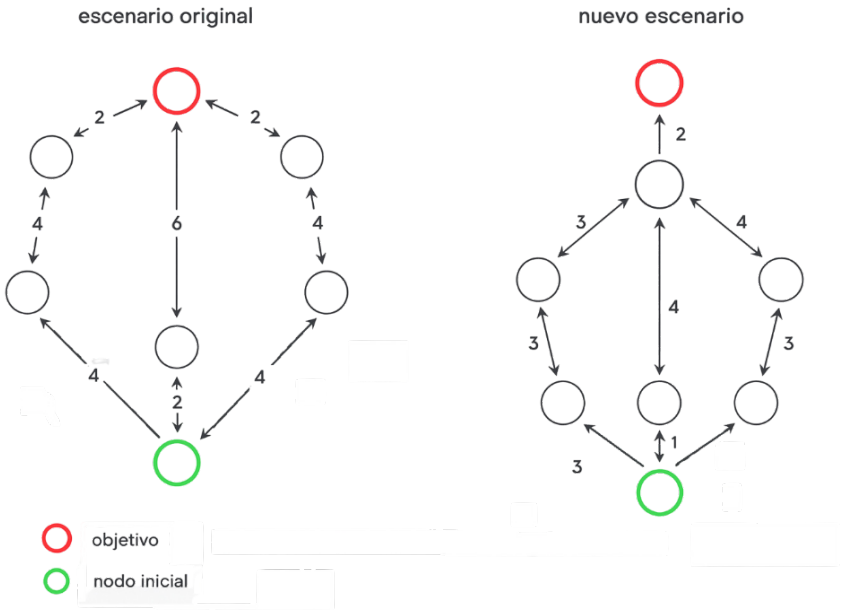

Figura 1: Agente en una situación real.

3. Diseño del entorno

El entorno base está construido sobre la clase `gym.Env` e implementa los métodos esenciales `reset()` y `step()`. Cada nodo del grafo se representa con un vector embebido que codifica información posicional o estructural. La observación entregada al agente combina tres componentes:

1. El embedding del nodo actual.
2. El embedding del siguiente waypoint y del destino.

3. Escalares que representan distancia al destino, distancia al waypoint y fracción de pasos restantes.

Esto da lugar a un vector continuo de observación de dimensión $3d + 3$, donde d es la dimensión del embedding de cada nodo. Las acciones corresponden a la elección de un vecino válido del nodo actual, con un espacio discreto de tamaño máximo igual al grado más alto del grafo.

3.1. Embeddings estructurales y métricas topológicas

Cada nodo del grafo urbano se representa mediante un *embedding* que combina tanto información espacial como estructural. Además de las coordenadas (x, y) y el grado normalizado, se incorporan métricas de conectividad y centralidad que permiten capturar propiedades locales y globales del entorno. Estos atributos enriquecen la observación del agente y mejoran la capacidad del modelo para aprender patrones de navegación en zonas con distinta densidad vial.

3.1.1. Conectividad avanzada

Grado de entrada (*in_degree*): cantidad de calles que llegan a una intersección. *Significado*: punto donde converge el tráfico. *Ejemplo*: si llegan 4 calles → `in_degree = 4`.

Grado de salida (*out_degree*): cantidad de calles que salen desde una intersección. *Significado*: punto donde el tráfico se dispersa. *Ejemplo*: si salen 4 calles → `out_degree = 4`.

Número de vecinos (*neighbor_count*): cantidad de intersecciones conectadas directamente. *Significado*: más vecinos implican más opciones de movimiento. *Ejemplo*: nodo conectado con 4 intersecciones → `neighbor_count = 4`.

3.1.2. Análisis del vecindario

Grado promedio de vecinos (*avg_neighbor_degree*): promedio de conectividad de los vecinos. *Significado*: si los vecinos están bien conectados, el nodo pertenece a una zona activa. *Ejemplo*: grados [2, 4, 6, 8] → promedio = 5.

Grado máximo de vecinos (*max_neighbor_degree*): el vecino más conectado. *Significado*: tener un vecino con alta conectividad permite alcanzar muchas rutas. *Ejemplo*: [2, 4, 6, 8] → máximo = 8.

Grado mínimo de vecinos (*min_neighbor_degree*): el vecino menos conectado. *Significado*: vecinos con pocas conexiones indican zonas más aisladas. *Ejemplo*: [2, 4, 6, 8] → mínimo = 2.

Desviación estándar del grado de vecinos (*neighbor_degree_std*): mide la variabilidad entre los grados de los vecinos. *Significado*: alta variabilidad indica zonas de transición (por ejemplo, entre barrios). *Ejemplo*: [2, 2, 8, 8] → `std = 3.0`. Esta medida es útil porque en zonas de transición la elección del camino tiene más impacto que en zonas céntricas, donde las decisiones pueden simplificarse sin pérdida de rendimiento.

3.1.3. Medidas de centralidad

Centralidad de grado (*degree_centrality*): cuantifica la importancia estructural de un nodo dentro de la red. *Fórmula*: `degree`/(`total_nodos` − 1) *Significado*: mide la relevancia topológica local de una intersección dentro del grafo urbano.

Centralidad de intermediación aproximada (*betweenness_approx*): estima si un nodo actúa como puente entre distintas zonas de la red. *Fórmula simplificada*: `neighbor_count`/`total_nodos` (aproximación local). *Significado*: refleja qué tan probable es que el tráfico pase por ese nodo al conectar diferentes sectores.

3.1.4. Vector final de embedding

El embedding final de cada nodo v_i se construye concatenando todos estos atributos normalizados:

$$E(v_i) = [x_i, y_i, \texttt{in\_degree}, \texttt{out\_degree}, \texttt{neighbor\_count}, \\ \texttt{avg\_neighbor\_degree}, \texttt{max\_neighbor\_degree}, \texttt{min\_neighbor\_degree}, \\ \texttt{neighbor\_degree\_std}, \texttt{degree\_centrality}, \texttt{betweenness\_approx}]$$

3.1.5. Extensión para grafos urbanos reales

En entornos urbanos derivados de datos reales de *OpenStreetMap* (OSM), los nodos y aristas contienen información adicional que describe características físicas del entorno vial. Para aprovechar esta información, el embedding estructural definido previamente puede ampliarse incorporando atributos semánticos obtenidos directamente de las aristas conectadas al nodo.

Entre las propiedades más relevantes se encuentran:

- **Límite de velocidad (*maxspeed*)**: indica la velocidad máxima permitida sobre el tramo. Su valor se normaliza dividiéndolo por un máximo global (por ejemplo, $120\,km/h$), generando una variable continua entre 0 y 1.
- **Tipo de superficie (*surface*)**: codifica el material de la vía (por ejemplo, `asphalt`, `gravel`, `unpaved`). Se transforma en una puntuación normalizada según la calidad o transitabilidad estimada de cada superficie.
- **Categoría vial (*highway*)**: clasifica el tipo de calle o ruta (`residential`, `primary`, `motorway`, etc.). Se asigna un código numérico o vector *one-hot* que preserva las relaciones jerárquicas entre tipos de vía.
- **Número de carriles (*lanes*)**: representa la capacidad del segmento vial. Se normaliza por un máximo razonable (por ejemplo, 6 carriles) para mantener la escala uniforme con el resto del embedding.
- **Longitud y tiempo de viaje (*length*, *travel_time*)**: miden el costo físico asociado al movimiento entre nodos. Estos valores pueden utilizarse tanto como variables de entrenamiento como en la función de recompensa del entorno.

El embedding extendido de un nodo v_i se obtiene concatenando el vector estructural original con las componentes semánticas normalizadas, generando una representación híbrida:

$$E'(v_i) = [E(v_i) \,\|\, \texttt{maxspeed\_norm}, \texttt{surface\_score}, \texttt{highway\_code}, \texttt{lanes\_norm}]$$

Esta combinación dota al agente de una percepción más rica del entorno urbano: puede distinguir no sólo la forma del grafo, sino también las condiciones viales de cada sector. En consecuencia, el aprendizaje por refuerzo puede capturar estrategias de navegación más realistas, priorizando vías rápidas, seguras o de menor costo energético según el objetivo de la política.

4. Wrapper de enmascaramiento de acciones

El entorno se complementa con un `ActionMaskingWrapper`, encargado de filtrar acciones inválidas o redundantes. Este módulo implementa:

1. **Máscaras de acción dinámicas:** solo las acciones que conducen a vecinos válidos y productivos (más cercanos al objetivo) son habilitadas.

2. **Prevención de ciclos:** el wrapper mantiene una cola de los últimos nodos visitados y un contador de visitas, aplicando penalizaciones progresivas o truncando el episodio ante la detección de bucles.

3. **Selección de respaldo:** si todas las acciones son inválidas, el wrapper elige una acción de respaldo que mantenga la continuidad del episodio.

5. Limitaciones y decisiones tomadas

A lo largo del desarrollo se realizaron múltiples intentos para conservar el espíritu del primer modelo funcional, aquel que operaba correctamente sobre grafos relativamente acotados (entre 400 y 500 nodos). Se experimentó con subgrafos de ciudades, pueblos y distintas estructuras urbanas con el objetivo de mantener un esquema de aprendizaje lo más cercano posible al *reinforcement learning* puro. Sin embargo, conforme la escala del grafo aumentaba, también lo hacía la complejidad combinatoria del entorno y la frecuencia con la que el agente caía en bucles, rutas irrelevantes o regiones del espacio de estados en las que era prácticamente imposible recuperar una trayectoria útil.

Una analogía permite comprender mejor este cambio. En el diseño original, el agente era como un explorador que recorre una ciudad desconocida aprendiendo únicamente de la experiencia: se pierde, experimenta, prueba caminos, comete errores y, a través del refuerzo, descubre patrones que lo acercan a los objetivos. Este enfoque funciona mientras la ciudad sea relativamente pequeña. A medida que la estructura crece y aparecen múltiples ramificaciones, callejones sin salida, loops naturales del entramado urbano y caminos indistinguibles desde la perspectiva local, el explorador pasa la mayor parte del tiempo vagando sin dirección. No está aprendiendo una política útil; simplemente está sobreviviendo al azar y a la escasez de señales claras de recompensa.

En ese escenario surgió la necesidad de introducir un *oráculo*: una fuente de información externa capaz de descartar acciones que, desde un punto de vista geométrico o topológico, no contribuían al progreso. Este oráculo incluía distancias precalculadas, heurísticas derivadas de Dijkstra, medidas de proximidad y otras *features* que permiten filtrar acciones ineficientes. No le indica al agente qué hacer, pero sí limita su espacio de acción a un subconjunto razonable. Con ello, el aprendizaje se vuelve viable y la convergencia se alcanza en tiempos prácticos.

Sin embargo, esta decisión tiene implicancias conceptuales. Al proporcionar un oráculo, el agente deja de ser un explorador completamente autónomo y pasa a depender de una guía que estructura el entorno antes de que la política siquiera intervenga. Desde la perspectiva teórica, esto va en contra de la esencia del *reinforcement learning* tradicional, cuya premisa central es aprender únicamente a partir de la experiencia y la recompensa. El sistema resultante ya no es RL puro, sino una variante guiada por *features*, donde el agente optimiza dentro de un espacio previamente moldeado para evitar comportamientos indeseados.

En síntesis, la decisión de incorporar un oráculo fue necesaria para lograr un comportamiento estable en grafos urbanos extensos, pero implica una reinterpretación metodológica del enfoque: el aprendizaje pasa de ser completamente emergente a estar fuertemente asistido por conocimiento externo sobre la estructura del entorno.

5.1. El oráculo estructural como guía geométrica y semántica

El entorno incorpora un oráculo estructural cuya función es proporcionar referencias cuantitativas y representacionales que permiten estabilizar la navegación en grafos urbanos amplios. Este oráculo no decide por el agente ni reemplaza la política aprendida, pero ofrece una base geométrica, topológica y semántica desde la cual es posible evaluar desvíos, detectar progresos y mantener la coherencia de las observaciones. Su diseño se organiza en tres componentes: el

cálculo de pasos óptimos, la medición de eficiencia relativa y la recuperación consistente del embedding nodal extendido $E''(v_i)$.

5.1.1. Cálculo de pasos óptimos como cota geométrica

El primer componente del oráculo establece una referencia absoluta basada en caminos mínimos. Para cada tramo del recorrido —desde el nodo inicial hasta cada waypoint y finalmente hasta el destino— se calcula la longitud del camino más corto en el grafo urbano. Esta operación produce un conjunto de cotas que representan el número mínimo de pasos que cualquier política debería emplear para completar el trayecto ideal.

Estas distancias óptimas no dependen del agente, sino de la estructura del grafo. Su valor es crucial como ancla geométrica: indican cuánto "debería" costar cada segmento del recorrido y permiten evaluar desviaciones significativas, incluso en regiones donde la señal de recompensa es insuficiente para inducir correcciones espontáneas.

5.1.2. Eficiencia relativa del agente respecto del óptimo

El segundo componente permite transformar las distancias mínimas en una señal dinámica que evalúa el rendimiento del agente en tiempo real. Para cada paso, se estima el costo total que tendría completar el trayecto suponiendo que, a partir de ese punto, se siguiera el camino óptimo. Esta estimación combina dos magnitudes:

1. los pasos ya ejecutados por el agente, y
2. la distancia restante calculada mediante caminos mínimos.

A partir de esta comparación se definen dos métricas complementarias: (a) la relación entre los pasos estimados y los óptimos, que indica el grado de desviación acumulada, y (b) una eficiencia normalizada entre 0 y 1, donde 1 representa un avance perfectamente alineado con la ruta ideal.

Estas métricas no reemplazan la recompensa, pero nutren mecanismos auxiliares como el enmascaramiento de acciones, permitiendo descartar movimientos que comprometen la eficiencia global del episodio.

El embedding estructural original $E(v_i)$ incluye atributos derivados de la forma local del grafo urbano: coordenadas normalizadas, grado de entrada y salida, número de vecinos, estadísticas del vecindario (promedio, máximo, mínimo y desviación estándar), centralidad de grado y aproximación local de intermediación. Además, incorpora atributos viales ya integrados en el embedding primario, como el tipo de superficie.

Propiedades dinámicas derivadas de la eficiencia hacia el destino. Para complementar la caracterización estática ofrecida por $E''(v_i)$, el entorno incorpora cuatro magnitudes continuas que describen el rendimiento del agente durante el episodio. Estas propiedades permiten capturar no solo "dónde está", sino también "cómo está navegando" en relación con la estructura óptima del grafo.

- **Pasos totales estimados para completar el episodio (*estimated_total_steps*).** Combina la distancia restante con los pasos ya realizados, produciendo una estimación del esfuerzo total necesario para finalizar el trayecto si se siguiera la ruta óptima desde ese punto.
- **Relación entre pasos estimados y pasos óptimos (*steps_vs_optimal_dest*).** Compara el costo estimado del comportamiento real con la cota mínima establecida por la geometría del grafo. Valores superiores a uno indican desviaciones crecientes del rendimiento ideal.

- **Eficiencia normalizada hacia el destino (*dest_efficiency*).** Es la inversa acotada de la relación anterior y toma valores entre cero y uno. Un valor cercano a uno representa una navegación altamente alineada con la estructura óptima; valores bajos expresan pérdidas acumuladas de eficiencia.

La concatenación de estos componentes produce el embedding extendido:

$$E''(v_i) = \big[\, E(v_i) \parallel \texttt{dest\_efficiency}, \texttt{steps\_vs\_optimal\_dest}, \texttt{estimated\_total\_steps} \,\big].$$

Este vector sintetiza la estructura local del grafo y las propiedades físicas de la infraestructura vial. Como resultado, cada nodo recibe una caracterización continua que preserva tanto su papel estructural como su función urbana.

6. Naturaleza discreta del entorno

El entorno `WaypointNavigationEnv` es de naturaleza **discreta** tanto en su espacio de acciones como en su evolución temporal. Cada paso (`step`) representa una transición entre nodos de un grafo dirigido o no dirigido, donde las acciones disponibles corresponden al conjunto finito de vecinos del nodo actual.

Esto implica que el agente no se mueve en un espacio continuo, sino a través de *estados discretos* representados por identificadores de nodos. Cada acción seleccionada se traduce en un desplazamiento atómico a un nodo adyacente, seguido por una evaluación de recompensa. De esta forma, la dinámica del entorno sigue el modelo clásico de procesos de decisión de Markov (MDP) con estado discreto $s_t \in S$ y acción discreta $a_t \in A(s_t)$.

A pesar de esta discretización topológica, las observaciones entregadas al agente pueden incluir componentes continuas (como los embeddings de nodos o las distancias ponderadas), lo que genera un entorno *mixto*: discreto en estructura, continuo en representación.

7. Conclusión

El entorno `WaypointNavigationEnv` constituye una plataforma flexible y extensible para estudiar navegación basada en grafos dentro del marco del aprendizaje por refuerzo. Su diseño modular, junto con el wrapper de control de acciones y los embeddings estructurales, permite modelar comportamientos realistas de agentes en entornos complejos, equilibrando fidelidad y eficiencia computacional.

En síntesis, este trabajo traduce el concepto abstracto de moverse en un grafo a un escenario urbano verosímil, en el que el agente no solo aprende a moverse, sino a *navegar con propósito*.

Referencias

[1] Farama Foundation. *Gymnasium: A Toolkit for Reinforcement Learning*. Disponible en: `https://gymnasium.farama.org`

[2] Hagberg, A. A., Schult, D. A., y Swart, P. J. (2008). *Exploring network structure, dynamics, and function using NetworkX*. In Proceedings of the 7th Python in Science Conference (SciPy2008).

[3] Sutton, R. S., y Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT Press.