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.

El término "*proximal*" hace referencia a esta cautela. En lugar de reescribir completamente su estrategia después de cada experiencia, PPO la modifica solo si la nueva política no se aleja demasiado de la anterior. Esto se logra mediante una función de pérdida que penaliza los cambios excesivos en la probabilidad de elegir acciones. De este modo, el agente mejora su rendimiento de forma estable y continua, sin saltos caóticos en el comportamiento.

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.

En términos conceptuales, el aprendizaje por refuerzo actúa como un proceso de inducción: a partir de ejemplos concretos, el agente infiere regularidades que le permiten desempeñarse eficazmente en estados nunca antes visitados. Así, el conocimiento aprendido en trayectorias particulares se reutiliza para mejorar el desempeño global del agente en todo el entorno urbano simulado.

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.

El sistema de recompensas incorpora varios términos:

- Penalización proporcional al costo de movimiento (por ejemplo, tiempo de viaje).
- Bonificación por acercamiento al destino o waypoint.
- Penalización por falta de progreso o repetición de nodos.
- Bonificación final por alcanzar el destino.

La combinación de estos factores produce una señal de aprendizaje densa y continua, adecuada para algoritmos como PPO (Proximal Policy Optimization) o DQN extendido.

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.

Esta arquitectura reduce la exploración redundante y acelera la convergencia del aprendizaje al eliminar decisiones sin valor informativo.

Estas mejoras permiten aplicar el entorno en contextos de navegación autónoma, optimización logística o simulaciones de tráfico a nivel urbano.

5. Aplicaciones y escalabilidad

El entorno está diseñado para escalar desde redes sintéticas hasta ciudades reales. En un grafo urbano con miles de nodos, las distancias se calculan mediante Dijkstra ponderado, aunque para entornos de gran escala se recomienda:

- Precomputar distancias entre puntos relevantes y almacenarlas en caché.
- Usar heurísticas A* para aproximar distancias.
- Dividir la ciudad en subgrafos (barrios) para entrenamiento localizado.

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. Este enfoque equilibra interpretabilidad y capacidad de aprendizaje, permitiendo la aplicación de algoritmos de RL tanto basados en valor (DQN) como basados en política (PPO, A2C).

7. Ejemplo ilustrativo

Para ilustrar el funcionamiento del entorno, consideremos un grafo simple con cinco nodos $\{0, 1, 2, 3, 4\}$, conectados de la siguiente forma:

$$0 \leftrightarrow 1 \leftrightarrow 2 \leftrightarrow 3 \leftrightarrow 4$$

El nodo inicial es 0, el waypoint es 2, y el destino final es 4. En cada paso, el agente puede moverse a un nodo vecino (acción discreta). La secuencia óptima de acciones sería:

$$0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4$$

Durante la ejecución, el agente recibe una recompensa positiva al alcanzar el waypoint (2) y una bonificación mayor al llegar al destino (4). Si el agente intenta regresar a nodos previos o entrar en ciclos, el wrapper aplica penalizaciones crecientes que desincentivan comportamientos redundantes.

En este ejemplo, el espacio de acciones por estado tiene tamaño variable: en los extremos (nodos 0 y 4) hay solo una acción válida, mientras que en los nodos intermedios hay dos. Esto refuerza la naturaleza discreta y topológicamente dependiente del entorno, donde la validez de cada acción está determinada por la conectividad local del grafo.

8. 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, 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.