UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA MATEMÁTICA

ALGORITMOS DE APRENDIZAJE REFORZADO PARA EL PROBLEMA DE RUTEO DE VEHÍCULOS DINÁMICO

TESIS PARA OPTAR AL GRADO DE
MAGÍSTER EN CIENCIAS DE LA INGENIERÍA, MENCIÓN MATEMÁTICAS APLICADAS

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL MATEMÁTICO

ENRIQUE ESTEBAN VILCHEZ VALENZUELA

PROFESOR GUÍA:
HÉCTOR RAMÍREZ CABRERA
PROFESOR CO-GUÍA:
ANDRÉS PEÑALOZA GONZÁLEZ

MIEMBROS DE LA COMISIÓN:
JOSÉ SOTO SAN MARTÍN
CRISTOBAL GUZMÁN PAREDES

Este trabajo ha sido financiado por Agencia Nacional de Investigación y Desarrollo/Subdirección de Capital Humano/Magíster Nacional/2020 - 22201628, CMM ANID PIA AFB170001 y Entel S.A.

SANTIAGO DE CHILE
2021

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE: INGENIERO CIVIL MATEMÁTICO
Y AL GRADO DE MAGÍSTER EN CIENCIAS DE LA INGENIERÍA,
MENCIÓN MATEMÁTICAS APLICADAS
POR: ENRIQUE ESTEBAN VILCHEZ VALENZUELA
FECHA: ENERO, 2021
PROF. GUÍA: HÉCTOR RAMÍREZ CABRERA

ALGORITMOS DE APRENDIZAJE REFORZADO PARA EL PROBLEMA DE RUTEO DE VEHÍCULOS DINÁMICO

En las formulaciones matemáticas más conocidas de los problemas de ruteo de vehículos es asumido que todos los clientes son conocidos desde un principio, permitiendo construir soluciones planificadas. Sin embargo, en situaciones más realistas suele suceder que algunos clientes sean conocidos después del periodo de planificación, cuando los vehículos ya están en sus rutas. Esto implica que la optimización tenga que hacerse en tiempo real para poder atender a aquellos clientes que lleguen de forma dinámica. Esta variación es conocida como el problema de ruteo de vehículos dinámico, el cual será estudiado en esta tesis.

En este trabajo se propone abordar el problema desde el enfoque del aprendizaje de máquinas y los datos a través de algoritmos de aprendizaje reforzado. En esta clase de métodos, se modela el problema a través de un proceso de decisión de Markov, en el cual un agente busca maximizar una función de recompensas que va a depender de las acciones que tome en cada estado. Como primera opción se usan los algoritmos del tipo "Q-learning" y "Actor-Critic", ampliamente citados en el área. Por otro lado, se desarrolla un algoritmo que permite aprender a partir de simulaciones de escenarios, cuando se tiene información estocástica del medio.

Se presentan una serie de experimentos computacionales para medir el desempeño de los modelos entrenados y se comparan con un algoritmo que utiliza una estrategia de "reoptimización" para tener una mejor apreciación de sus ventajas y desventajas.

"Just as electricity transformed many industries roughly 100 years ago, AI will also now change nearly every major industry — healthcare, transportation, entertainment, manufacturing — enriching the lives of countless people. I am more excited than ever about where AI can take us."
Andrew Ng

Agradecimientos

Quiero agradecer primero a mi familia, especialmente a mis papás y mi hermana que siempre han estado apoyándome siempre, en todas las etapas de mi vida. Sin ustedes hubiera sido mucho más difícil llegar a donde estoy ahora.

En cuanto a la realización de esta tesis, quiero agradecer a mi profesor guía Héctor Ramírez por su compromiso, a José Soto por estar siempre dispuesto a ayudar, a Ian Yon por enseñarme su visión de tecnología e iniciarme en el mundo de la inteligencia artificial, y a Andrés Peñaloza por participar oficialmente como mi co-guía y ayudarme en los últimos meses de trabajo. También mil gracias a Gustavo Riveros e Ivania Briceño por guiarme en relación a los casos de uso de la tesis.

No puedo olvidarme de mis amigos del colegio. Valoro mucho que nos sigamos juntando, aunque sea por Zoom en esta etapa de pandemia. Así que gracias por estar Mati, Eche, Caro, Chachi, Calo, Lucho, Seba, Som y Lila. A mis amigos del plan común, sin ustedes hubiera sido mucho más aburrido entrar a la etapa universitaria, así que gracias Correa, José, Dani, Álvaro, Marley y Panchito. También quiero hacer una especial mención a los "amigos del 22" que ha sido un grupo súper apañador y que ha estado siempre en las buenas y en las malas, desde que los conocí en el DIM hasta el día de hoy. Así que gracias Alonso, Freddy, Mati, Feña, Nano, Pablo y PL, los quiero cabros.

Por último agradecer a la familia de la Techforce de Entel Ocean, los que indirectamente siempre me acompañaron en la realización de esta tesis.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

1.1. Motivación

Los servicios de despacho de última milla presentan una gran dificultad para los servicios logísticos y para las empresas en general. El incremento de las ventas por internet ha generado un incremento en la cantidad de productos a despachar. Es común que pequeñas empresas tengan disponibles su propio sitio web con sus productos y usen sus propios vehículos para despachar los artículos que venden. En el contexto de la pandemia del COVID-19 en el año 2020, el sistema logístico de Chile y el mundo ha sido completamente desafiado. Las cuarentenas han limitado los desplazamientos que no son estrictamente necesarios. Es por ello que muchas personas han optado por el ecommerce y servicios de despacho online. Según un estudio que revisó el desempeño de varias aplicaciones de despachos [40], la alta demanda generada por la pandemia generó colapsos en varias de las apps revisadas. Incluso, las demoras podían ser de semanas, en apps que antes agendaban turnos de entrega de un día para otro.

En este contexto en la empresa Entel Ocean se desarrolla una aplicación móvil que permita gestionar la logística para emprendedores. Uno de los objetivos de esta aplicación es que transforme el despacho en una experiencia óptima. En este sentido surge la necesidad de que la aplicación tenga algún algoritmo de optimización de rutas.

Teniendo la hipótesis de que los clientes buscan la mejor experiencia posible, puede ser ventajoso centrarse en mejorar los tiempos de despachos a los clientes. Una forma de cumplir esto en su máxima expresión es que los emprendedores puedan ofrecer despachos para el mismo día. Según un estudio [16] realizado en China, Alemania y Estados Unidos, entre un 20 % y un 25 % de los consumidores está dispuesto a pagar por servicios de despacho para el mismo día. Más aún, se estima que en el mundo el mercado del despacho para el mismo día y el despacho instantáneo va a alcanzar el 15 % en el 2020. En Estados Unidos las grandes empresas Walmart, Amazon y Target ya realizan este tipo de despacho [37]. Chile tampoco se queda atrás, donde Cornershop y Jumbo cuentan con entregas en 90 minutos y Linio ofrece entregas para el mismo día en algunos productos. Por otro lado las aplicaciones Uber Eats y Rappi realizan despachos en minutos [3].

Por otro lado, teniendo una visión de más largo plazo, el uso de vehículos autónomos va

a tener una mayor preponderancia a futuro. Se estima que los vehículos autónomos van a despachar cerca del 80 % de los productos. Por ejemplo, el uso de drones permite el despacho para el mismo día de manera rápida y directa, no requiriendo usar las calles urbanas y evitando la congestión vehicular. Por otro lado, permiten satisfacer la demanda de sectores rurales, donde los servicios logísticos actuales no llegan o es muy cara su operación [16]. Una empresa que ya ha avanzado en esta tecnología es Amazon con su iniciativa "Prime Air" [2], que promete despachar productos en menos de 30 minutos, además de mejorar la eficiencia y la seguridad en los sistemas de transportes.

Estos casos de uso dan la idea de que es primordial optimizar los costos de las rutas en situaciones no planificadas. Esto también se justifica por el hecho de que el 50 % de los consumidores elige el método de envío por el precio del despacho [16], por lo que si se quiere implementar un sistema de despachos que sea dinámico durante el día, tiene que ser lo más barato y óptimo posible.

Pero no sólo se estaría resolviendo este problema al tener un sistema de despachos "online", ya que por ejemplo es común que las empresas tengan cambios repentinos e inesperados durante en el día que afectan las rutas de los vehículos que se crearon de forma inicial.

En este sentido se hace necesario contar con algún algoritmo que pueda abordar estas situaciones de la mejor forma posible, de tal manera de abaratar los costos de implementación.

1.2. Contribuciones de esta tesis

En esta tesis se propone desarrollar un algoritmo de optimización de rutas capaz de responder ante situaciones no planificadas y dinámicas durante el día de operación de una flota de vehículos. Este problema comúnmente se conoce como el problema de ruteo de vehículos dinámico. Para ello se decide usar algoritmos de aprendizaje de máquinas que permitan aprender cuáles son las rutas óptimas cuando se está en este tipo de situaciones. Los algoritmos del estado del arte usan como base redes neuronales profundas, que permiten resolver y aprender situaciones complejas y de alta dimensionalidad.

En el segundo capítulo se entregan los preliminares de los problemas de ruteo de vehículo clásicos, los cuales van a servir como base para desarrollar un modelo para el problema que se quiere estudiar. Por otro lado se da una introducción de la clase de algoritmos de aprendizaje de máquinas que se pretende usar. Estos son conocidos como algoritmos de aprendizaje reforzado.

En el tercer capítulo se propone un modelo matemático para el problema.

En el cuarto capítulo se introducen tres clases de algoritmos de aprendizaje reforzado con los cuales se pretende resolver el problema. Se da una mirada general a estos algoritmos y se estudia cómo estos se aplican para el problema de ruteo de vehículos dinámico. Dos de estos algoritmos no asumen conocimiento estocástico de las solicitudes de clientes, mientras que un algoritmo sí lo asume.

En el quinto capítulo se explica la arquitectura de la red neuronal que utilizan los algoritmos.

Por último, en el sexto capítulo se muestran los resultados de los modelos entrenados y se compara con un tipo de algoritmo que utiliza una estrategia de "reoptimización", mediante una serie de experimentos.

Capítulo 2

Preliminares

2.1. Problemas de ruteo de vehículos

En esta sección se mostrarán los preliminares de los problemas de ruteo para esta tesis. Primero se verá el clásico problema definido en la literatura y algunas de sus extensiones. Por otro lado se introducen los términos de "evolución de la información" y "calidad de la información" que permitirán que el problema de ruteo pueda dividirse en otros problemas que involucran dinamismo y estocasticidad en su naturaleza.

2.1.1. Problema clásico

El problema clásico o determinista es conocido como el problema de ruteo de vehículos o *Vehicle Routing Problem* (VRP por sus siglas en inglés). En este problema un conjunto de clientes tienen que ser atendidos por un conjunto de vehículos homogéneos. En general se va a tener que cada uno de estos vehículos va a tener una capacidad disponible y los clientes una demanda. A esto se le conoce como ruteo de vehículos con capacidades o *Capacitaded Vehicle Routing Problem* (CVRP por sus siglas en inglés). Se asume que todos los vehículos inician y terminan sus operaciones en un punto llamado bodega o depósito. El problema consiste en encontrar un conjunto de rutas para los vehículos, minimizando una función de costos.

El VRP ha sido un problema muy estudiado en las últimas décadas en el ámbito de las matemáticas discretas y la investigación de operaciones. Este es en efecto un problema NP-completo, por lo que resolverlo óptimamente se hace cada vez más difícil a medida que el tamaño de las instancias crece.

El CVRP es comúnmente modelado a través de un problema lineal entero. Se denota por $N = \{1, ..., n\}$ el conjunto de clientes. Por otro lado se define $V = \{0\} \cup N$ el conjunto de n clientes y la bodega (identificada con el 0). Sea $K = \{1, ..., m\}$ el conjunto de los m vehículos. Toda la flota de vehículos es homogénea con capacidad $Q > 0$. Cada cliente $i \in N$ tiene una demanda, que está dada por un número $0 \leq q_i \leq Q$, donde las unidades de q_i van a ser las mismas que de Q. Se define el grafo completo y dirigido $G = (V, A)$, con $A = \{(i, j) \in V \times V : i \neq j\}$. Usando la notación para grafos dirigidos se define el conjunto

de arcos entrantes y salientes del conjunto $S \subseteq V$ como:

$$\delta^+(S) = \{(i,j) \in A : i \in S, j \notin S\}, \quad \delta^-(S) = \{(i,j) \in A : i \notin S, j \in S\}, \tag{2.1}$$

donde además se denota $\delta^+(i) := \delta^+(\{i\})$ y $\delta^-(i) := \delta^-(\{i\})$. Se define para cada $(i,j) \in A$ el costo de viaje c_{ij}. El modelo se construye definiendo la variable binaria x_{ij} que indica si un vehículo se mueve directamente desde i hasta j.

$$\text{mín} \quad \sum_{(i,j)\in A} c_{ij} x_{ij} \tag{2.2}$$

$$\sum_{j\in\delta^+(i)} x_{ij} = 1 \qquad \forall i \in N \tag{2.3}$$

$$\sum_{i\in\delta^-(j)} x_{ij} = 1 \qquad \forall j \in N \tag{2.4}$$

$$\sum_{j\in\delta^+(0)} x_{0j} = m \tag{2.5}$$

$$\sum_{(i,j)\in\delta^+(S)} x_{ij} \geq r(S) \qquad \forall S \subseteq N, S \neq \emptyset \tag{2.6}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A, \tag{2.7}$$

donde (2.2) corresponde al objetivo de minimizar el costo total de las rutas, (2.3) y (2.4) dice que en cada ruta los nodos deben estar conectados con otros dos vértices, (2.5) dice que exactamente m rutas son construidas y (2.6) exige cumplir las restricciones de capacidad de los vehículos y que no existan "subtours". Además $r(S)$ es el mínimo número de rutas necesarias para poder atender S. Para calcular este valor es necesario resolver óptimamente el "*Bin Packing Problem*" con N objetos de tamaño q_i. Esto quiere decir que calcular $r(S)$ de forma exacta se reduce a resolver un problema NP-Hard [21]. Sin embargo la formulación se mantiene válida si se reemplaza $r(S)$ por su cota inferior dada por

$$\left\lceil \frac{\sum_{i\in S} q_i}{Q} \right\rceil$$

En la versión clásica del CVRP no existe la dimensión temporal, que en los contextos más aplicados es vital incorporarlo al modelo. Por ejemplo, para que los vehículos tengan un horario determinado de operación o para poder incluir posibles restricciones de horarios en los cuales los clientes aceptan ser visitados. El problema teórico que más se adapta a estas restricciones se conoce como el problema de ruteo de vehículos con ventanas de tiempo o *Vehicle Routing Problem with Time Windows* (VRPTW por sus siglas en inglés). En el siguiente capítulo el modelo del VRPTW será descrito con mayor profundidad, ya que va a ser uno de los problemas base para el objetivo de este trabajo.

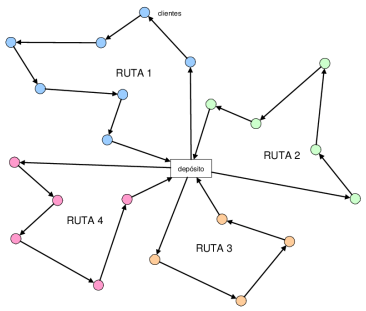

Figura 2.1: Ejemplo gráfico de una solución para el VRP

2.1.2. Problema de ruteo de vehículos dinámico

Una de las desventajas del VRP es que asume información perfecta en los datos de entrada, imposibilitando su aplicación en ciertos problemas del mundo real que tienen datos inciertos o variables. La formulación clásica asume que todos los clientes se conocen desde un principio, permitiendo construir soluciones planificadas. En este sentido, si existe algún cambio en las demandas o existen nuevas solicitudes de clientes cuando los vehículos ya están en sus rutas, se hace necesario un modelo que pueda enfrentar estas situaciones no planificadas.

Siguiendo el trabajo realizado en [26], los problemas de ruteo de vehículos se pueden clasificar según la evolución y la calidad de la información. La evolución explica el hecho de que puede ser que la información que se tiene puede cambiar durante la ejecución de las rutas. Por otro lado la calidad refleja la posibilidad de que exista incertidumbre en los datos disponibles. Basado en éstas dimensiones se entrega la siguiente clasificación

		Calidad de la información	
		Entrada determinista	Entrada estocástica
Evolución de la información	Entrada conocida de antemano	Estático y determinista	Estático y estocástico
	Entrada cambia a través del tiempo	Dinámico y determinista	Dinámico y estocástico

Tabla 2.1: Taxonomía de problemas de ruteo según evolución y calidad de la información

Los problemas estáticos y deterministas, son aquellos donde toda la entrada es conocida con anterioridad y esta no cambia en el tiempo. Por ejemplo el VRP, CVRP y VRPTW pertenecen a esta categoría.

Los problemas estáticos y estocásticos, son aquellos caracterizados por tener una entrada parcialmente conocida, ya que está representada por variables aleatorias. Adicionalmente las rutas son diseñadas a priori, pero después no se permite hacer ningún cambio a ellas.

Los problemas dinámicos y deterministas, son aquellos en que parte de su entrada es desconocida en un principio, pero luego es revelada dinámicamente durante la ejecución de las rutas.

Los problemas dinámicos y estocásticos, son aquellos en que parte de su entrada es desconocida en un principio y es revelada dinámicamente durante la ejecución de las rutas, pero

a diferencia del anterior, información estocástica está disponible, lo que permite tener un mayor conocimiento de la distribución de estas componentes dinámicas.

Esta tesis se centrará principalmente en los últimos dos problemas mencionados, es decir, en general en los problemas de ruteo de vehículos dinámicos. Por sus siglas en inglés, a este problema también se le va a conocer como DVRP ("Dynamic vehicle routing problem").

Específicamente hay varias componentes que introducen dinamismo al problema clásico como, por ejemplo, los tiempos de viaje (que pueden cambiar por tráfico o accidentes), demandas (clientes cambian sus pedidos), disponibilidad de vehículos (averías o problemas que puedan ocurrir en sus rutas) o solicitudes (nuevos clientes realizan pedidos). Esta tesis se centrará en el caso de la aparición de solicitudes de nuevos clientes durante el horario de operación de una flota de vehículos.

En cuanto a los términos logísticos, este problema puede ser aplicado en despachos de bienes o en recolección de bienes y servicios. En general, para los problemas estáticos no se hacía mucho hincapié en qué tipo de logística se centraban ya que eran equivalentes. Sin embargo, en este problema importa, ya que por ejemplo si se hacen despachos de una tienda de retail, los vehículos deben retornar a la bodega para incluir las solicitudes de los nuevos clientes. Por otro lado, en recolección y servicios esto no es necesario. Tampoco se hace necesario en despachos mono producto en que no sea necesario devolverse a la bodega para atender nuevos clientes. En este trabajo se centrará en los casos en que los vehículos pueden desplazarse a sus clientes libremente sin necesidad de devolverse a la bodega para poder atenderlos. Para simplificar el problema tampoco se va a permitir retornar a la bodega por problemas de capacidad, es decir, si alguno de ellos decide volver a la bodega, no puede volver a salir a atender clientes.

Para dar a entender mejor cómo se diferencia el problema dinámico del problema clásico, se dará un ejemplo de la ejecución de una ruta de un DVRP con un sólo vehículo. Antes de que el vehículo deje la bodega (tiempo t_0), existe un plan inicial con una ruta para los clientes conocidos hasta ese tiempo (A, B, C, D, E). Mientras el vehículo ejecuta la ruta, dos nuevas solicitudes (X, Y) aparecen en el tiempo $t_1 > t_0$, por lo que la ruta inicial tiene que ser ajustada para poder atenderlos. Finalmente a tiempo t_f la ruta ejecutada es (A, B, C, D, Y, E, X).

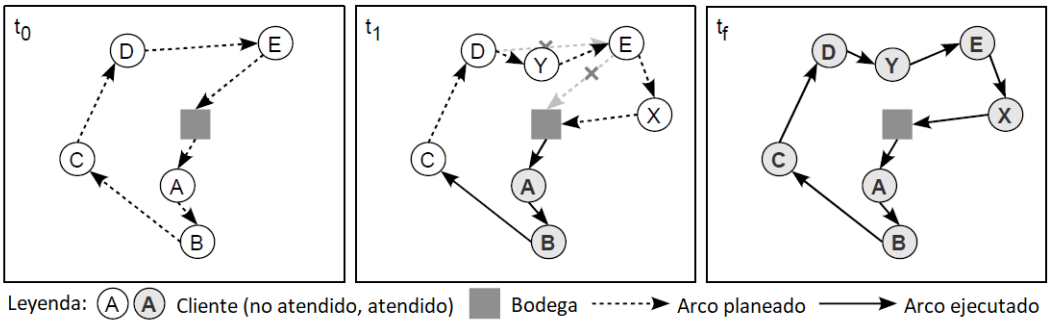

Figura 2.2: Ejemplo ruteo de vehículos dinámico [26]

2.1.3. Midiendo el dinamismo

De acuerdo con [26] existen diversas métricas con las cuales se puede medir el dinamismo del DVRP. Este problema da la posibilidad de que existan solicitudes que sean conocidas de antemano (por ejemplo, puede ser que estas solicitudes se reciban durante la noche, antes de que empiece el horario de operación de la flota de vehículos) y otras solicitudes que van a ser "dinámicas", dado que estas aparecen después de que los vehículos inicien sus rutas.

Definición 2.1 *Se define el* ***grado de dinamismo*** δ *como la división entre el número de solicitudes dinámicas* n_d *y el número total de solicitudes* n_{tot}*:*

$$\delta = \frac{n_d}{n_{tot}} \tag{2.8}$$

Sea T el rango horario de operación de los vehículos, es decir, desde que tienen permitido salir de la bodega hasta la hora límite que pueden volver a la bodega. Este valor es igual para todos los vehículos.

Definición 2.2 *Dado* $\mathcal{R}$ *el conjunto de solicitudes y* ct_i *la hora en que se hace la solicitud del cliente* $i \in \mathcal{R}$*, se define el* ***grado de dinamismo efectivo*** δ^e *como*

$$\delta^e = \frac{1}{n_{tot}} \sum_{i \in \mathcal{R}} \frac{ct_i}{T} \tag{2.9}$$

En esta formulación las solicitudes que se conocen de antemano tienen $ct_i = 0$*.*

2.1.4. Tipos de soluciones

En los últimos 20 años el número de publicaciones del problema de ruteo de vehículos dinámico a tenido un gran crecimiento, especialmente en los años 2000. Esto debido al creciente desarrollo de nuevas tecnologías como los sistemas de GPS (*Global Positioning Systems*), GIS (*Geographical Information Systems*), ITS (*Inteligent Trasportation Systems*), e-commerce, Big Data y computación en la nube. Estos desarrollos tecnológicos han permitido implementar soluciones de ruteo dinámico de manera más sencilla y amigable para grandes flotas de vehículos [27].

Las soluciones y algoritmos disponibles en la literatura usan principalmente técnicas de optimización, programación dinámica y heurísticas.

En el ámbito de la optimización y la investigación de operaciones, se usa la técnica de "reoptimización", que consiste básicamente en resolver periódicamente el problema estático en periodos fijos (llamados épocas o periodos de decisión) tomando en cuenta los clientes que no han sido visitados. Los algoritmos que se usan son principalmente los mismos que se usan para el problema estático. Por ejemplo, [46] usa este enfoque en una flota de camiones que tienen sólo permitido recoger y entregar un producto a la vez. Ellos resuelven un problema

lineal asociado al problema estático. Por otro lado, [8] usa el enfoque reoptimizador pero usando generación de columnas entre los periodos en que se entregan las soluciones a los vehículos. En el caso de [6] se aplica esta técnica pero a una gran flota de taxis, probando ser una solución medianamente escalable. Cabe notar que todas estas soluciones estaban enfocadas en el problema dinámico y determinista, no siendo aplicables al caso estocástico.

En el ámbito de la programación dinámica, principalmente este problema es modelado a través de un proceso de decisión de Markov (PDM) [28] y resuelto con técnicas de aproximación de funciones. Por ejemplo, [32] aplica esta formulación para una flota de ambulancias, con un alto nivel de dinamismo.

También existen técnicas heurísticas que han sido aplicadas a los problemas de ruteo clásico. Por ejemplo está [12] que usa el método de Tabú Search. También hay soluciones más modernas como la de [25] que usa el método de "Particle Swarm Optimization".

En cuanto a resolver específicamente el problema dinámico y estocástico, la solución más citada corresponde a [5], en la cual se propone utilizar una estrategia de simulaciones de posibles clientes que pueden aparecer en un futuro usando una distribución de probabilidad conocida. Esta estrategia es conocida como "Multiple Scenario Approach (MSA)". En cuanto a la estrategia de optimización, esta no la tocan y la dejan libre. Esta solución también ha sido aplicada en el ámbito de logísticas de despacho en el mismo día [44].

2.2. Aprendizaje reforzado

Aprendizaje reforzado es un área del aprendizaje de máquinas usado en el ámbito de la toma de decisiones donde un *agente* activamente interactúa con su *ambiente* para cumplir cierto objetivo. De acuerdo con el libro de Sutton y Barto [34], el aprendizaje reforzado involucra "aprender qué hacer y cómo traducir situaciones a acciones con tal de maximizar una recompensa numérica". Al aprendiz no se le dice cuáles acciones debe tomar, sino que este debe descubrir qué acciones producen la mayor recompensa posible al ejecutarlas. En los casos más interesantes y desafiantes, las acciones no sólo generan recompensas inmediatas, sino que también recompensas subsecuentes. Estas dos características (búsqueda mediante prueba-error y recompensas retrasadas) son las dos características más importantes del aprendizaje reforzado.

Normalmente se formaliza el problema de aprendizaje reforzado usando ideas de sistemas dinámicos y de procesos de decisión de Markov [28]. Este último incluye formalmente los conceptos anteriormente mencionados, los que serán abordados en esta tesis.

Naturalmente, el aprendizaje reforzado es diferente del *aprendizaje supervisado*, cuyo objetivo es aprender de un conjunto de entrenamiento con ejemplos etiquetados provenientes de algún experto o algún conocimiento externo que se tenga. Además es también diferente del *aprendizaje no supervisado*, que se enfoca típicamente de encontrar patrones ocultos en colecciones de datos no etiquetados. Es por esto que el aprendizaje reforzado se considera como un tercer paradigma del aprendizaje de máquinas.

Uno puede identificar cuatro componentes en un sistema de aprendizaje reforzado: una

política, una recompensa, una función valor y opcionalmente un modelo del ambiente.

La política es una función de los estados del ambiente a las acciones que se pueden tomar en esos estados. La recompensa define que tan buena o mala fue la decisión tomada por el agente. Es preciso reafirmar que el objetivo del agente es maximizar la recompensa total que recibe en el largo plazo. La función valor de un estado es la recompensa total esperada que puede acumular el agente desde ese estado. Y por último el modelo imita el comportamiento del ambiente. En general los modelos son usados para la planificación, en la cual las acciones son seleccionadas considerando posibles escenarios futuros. Estos métodos son conocidos como métodos *model-based*. En el lado opuesto, si el agente aprende directamente sin tener acceso a un modelo, se dice que es un método *model-free*.

En los últimos años, el aprendizaje reforzado ha sido popular gracias a su éxito al enfrentarse en problemas de toma de decisiones desafiantes [11]. Estos logros son producto de la combinación de aprendizaje reforzado con técnicas de aprendizaje profundo o *deep learning* [13]. A esta combinación se le conoce como aprendizaje reforzado profundo, el cual es capaz de resolver problemas con alta dimensionalidad.

En este tipo de métodos, la política es parametrizada y es en general una red neuronal profunda.

2.2.1. Aprendizaje reforzado en problemas de optimización combinatorial

En general problemas de optimización combinatorial pueden ser resueltos de forma secuencial, adaptándolos al paradigma de un problema de aprendizaje reforzado. En los últimos años han habido varias publicaciones que pretender resolver estos problemas con aprendizaje reforzado profundo.

Por ejemplo en [17] se resuelven problemas de optimización combinatorial sobre grafos aplicando algoritmos de Q-learning (se verán más adelante estos algoritmos). Específicamente abordan los problemas de cobertura de vértices mínima, corte máximo y el problema del vendedor viajero.

También en [4] se introduce un algoritmo que entrena una política estocástica que entrega permutaciones como solución del problema del vendedor viajero.

En [10] hacen algo parecido pero con una red neuronal modificada. Una peculiaridad es que ellos mejoran sus soluciones mediante el algoritmo de búsqueda local 2OPT [38].

La publicación más reciente corresponde a [19], quienes resuelven una serie de problemas de ruteo de vehículos.

En cuanto a soluciones de aprendizaje reforzado que se apliquen al DVRP, no hay mucha literatura disponible. Un caso resuelve el problema aplicado a una flota mixta de vehículos y drones, maximizando el número de clientes atendidos [7].

2.3. Casos de uso en Entel Ocean

Entel Ocean es la unidad digital de la empresa chilena Entel. El objetivo de esta unidad es brindar soluciones digitales y acelerar la transformación digital de las empresas.

Uno de los focos de Entel Ocean es la industria de la logística. Uno de los desarrollos que se enfocan en este rubro es la aplicación de gestión de flotas llamada "Follow".

Follow es una herramienta de gestión logística para emprendedores qué transforma el despacho en una experiencia óptima y confiable, contribuyendo a la profesionalización del negocio, mayor rentabilidad, escalabilidad y sustentabilidad.

Entre las funcionalidades de Follow, esta la opción de crear y optimizar rutas, a partir de tareas creadas por el usuario.

Una de las prioridades es que el algoritmo a implementar en esta tesis sea compatible con la aplicación, con tal de que las empresas puedan ofrecer servicios logísticos para el mismo día. Por otra parte debe dar facilidades ante otras situaciones no planificadas por los usuarios.

Ahora, en cuanto a los casos de uso en concreto en que se pensó que se podría aplicar un algoritmo de este estilo, se encuentran las siguientes aplicaciones:

- Servicios de mantenimiento y reparaciones
- Servicios de instalaciones
- Servicios para compartir automóviles ("carpooling")
- Despacho de artículos para el mismo día
- Recolección de artículos para el mismo día

Como esta tesis sólo se centrará en los casos en que el vehículo no tenga que volver a la bodega para ir a buscar productos (por ejemplo, para hacer despachos), los otros casos de uso pueden ser considerados para posibles extensiones.

Por otro lado, se pensó en que el algoritmo podría ser el enlace entre conductores de vehículos independientes y empresas que requieran servicios logísticos. En otras palabras, se pensó en construir una plataforma de servicio de logística para terceros, que permitiera un servicio de recolección y despacho en tiempo real para empresas. Esta última opción, a pesar de ser sólo una idea, puede resultar en una solución innovadora especialmente en situaciones en que la demanda de despachos sea mayor a la usual, para que los vendedores puedan tercerizar su logística, abaratando costos y poder satisfacer la creciente demanda de sus clientes.

En cuanto a los potenciales beneficios, contar con un algoritmo de este tipo permite abaratar costos, ya que permite reducir el consumo de combustible, reducir el mantenimiento a los vehículos y reducir los tiempos de gestión de la logística.

Además puede ayudar a las empresas a generar mayores ingresos, ya que está la posibilidad de atender a más clientes.

Pero no sólo hay beneficios económicos, sino que también genera una mejor experiencia de usuario, ya que permite reducir los tiempos de servicio y mejorar la promesa de entrega. Esto último da la posibilidad a los pequeños empresarios de poder competir con plataformas ya establecidas

Capítulo 3

Modelos y formulaciones

En este capítulo primero se describen dos modelos matemáticos conocidos para el problema de ruteo de vehículos con ventanas de tiempo (VRPTW). Usando la notación de estos modelos se describe el problema de ruteo de vehículos dinámico, usando como base lo desarrollado para el problema estático.

3.1. Modelo para el problema de ruteo de vehículos con ventanas de tiempo

Sea $G = (V, A)$ un grafo completo dirigido, donde $V = \{0, 1, ..., n\}$ es un conjunto de $n+1$ nodos y A es un conjunto de arcos. El nodo 0 representa la bodega y $N = V \backslash \{0\}$ corresponde a n clientes. Cada cliente tiene una demanda de $q_i \in \mathbb{R}_+$ unidades (se asume que $q_0 = 0$).

Una flota homogénea de vehículos está inicialmente estacionada en la bodega y es usada para suplir a los clientes. La flota de vehículos está compuesta por m vehículos que se van a asumir iguales, con $K = \{1, ..., m\}$. Cada vehículo $k \in K$ tiene una capacidad igual a Q.

Para cada $(i, j) \in A$ un costo no negativo por arco c_{ij} es dado.

Cada cliente $i \in V$ tiene una ventana de tiempo $[e_i, l_i]$, donde e_0 y l_0 son la partida más temprana desde la bodega y la llegada más tarde a la bodega respectivamente. Sea t_{ij} el tiempo de viaje desde i a j con $(i, j) \in A$, y st_i el tiempo que debe permanecer el vehículo que sirve al cliente $i \in V$ para satisfacer su demanda, o también llamado el tiempo de servicio para el cliente $i \in V$. Se asume que la matriz de tiempos de viaje compuesta por $(t_{ij})_{(i,j)\in A}$ satisface la desigualdad triangular.

Una *ruta* R es definida como un circuito simple en G que contiene la bodega, donde $R = (i_1, ..., i_{|R|})$ con $i_1 = i_{|R|} = 0$ y $R' = \{i_2, ..., i_{|R|-1}\} \subseteq N$. R es usado para referirse tanto a la secuencia de visitas como al conjunto de nodos (incluyendo la bodega) de la ruta.

Para una ruta R, con $R = (i_1, ..., i_{|R|})$, el comienzo del servicio T_{i_h} en el vértice i_h es

$$T_{i_h} = \text{máx}\{T_{i_{h-1}} + st_{i_{h-1}} + t_{i_{h-1},i_h}, e_{i_h}\} \quad h \in 2, ..., |R|$$

$$T_{i_1} = e_0$$

lo que significa que el vehículo debe servir a el cliente i_h en el tiempo e_{i_h} si es que llega antes de esta hora, es decir, debe esperar. Si no, empieza inmediatamente de haber llegado, tiempo que está dado por la hora en que empezó a servir a el cliente anterior sumando el tiempo de servicio del cliente anterior, más el tiempo que le tomó viajar al cliente actual.

Una ruta R se dirá *factible* si se cumplen las siguientes dos condiciones:

- La demanda total de los clientes visitados en la ruta no excede la capacidad Q del vehículo es decir:
$$\sum_{h=2}^{|R|-1} q_{i_h} \leq Q$$
- La llegada a la bodega del vehículo asociado a la ruta R debe ser antes del cierre de la bodega, es decir:

$$T_{i_{|R|}} \leq l_0$$

Se define el costo de la ruta R como:

$$C_R = \sum_{h=1}^{|R|-1} c_{i_h,i_{h+1}} \tag{3.1}$$

y el retraso de la ruta R como:

$$L_R = \sum_{h=2}^{|R|} \text{máx}\{T'_{i_h} - l_{i_h}, 0\} \tag{3.2}$$

Donde $T'_{i_h} = T_{i_{h-1}} + st_{i_{h-1}} + t_{i_{h-1},i_h}$.

La idea del VRPTW es encontrar un conjunto de rutas factibles $\{R_1, R_2, ..., R_m\}$ (una ruta para cada vehículo con posibilidad de que la ruta sea vacía en caso en que el vehículo no se ocupe) tal que

- Todos los clientes sean visitados
- Cada cliente sea visitado por exactamente una sóla ruta
- El retraso L_{R_j} sea 0 para todo $j \in \{1, ..., m\}$

Y se minimice el costo total de las rutas, es decir, se resuelve el siguiente problema de optimización:

$$\text{mín} \sum_{j=1}^{m} C_{R_j} \tag{3.3}$$

En algunos casos la restricción de los retrasos puede ser relajada, imponiendo una penalización sobre los retrasos en la función objetivo

$$\text{mín} \sum_{j=1}^{m} C_{R_j} + \alpha L_{R_j} \tag{3.4}$$

con $\alpha > 0$ una constante de penalización para los retrasos.

3.2. Formulación de programación entera-mixta del VRPTW

Al igual que antes se define $K = \{1, ..., m\}$ el conjunto de vehículos. Para esta formulación se modifica levemente el grafo definido en la sección anterior. La bodega va a estar representada por dos vértices: el vértice 0 y el $n + 1$, donde n es el número total de clientes. Estos nodos corresponderían a nodos fuente y de llegada. Para simplificar la notación $q_0 = q_{n+1} = st_0 = st_{n+1} = 0$. También las ventanas de tiempo $[e_0, l_0] = [e_{n+1}, l_{n+1}]$. Sea V representando al conjunto de clientes y a los dos vértices añadidos. Sea $N = V \setminus \{0, n + 1\}$. Se define un nuevo conjunto de arcos definido por:

$$A = \{(i, j) : i \in N, j \in N\} \cup \{(0, j) : j \in N\} \cup \{(i, n + 1) : i \in N\} \cup \{(0, n + 1)\},$$

lo que significa que los vehículos tienen permitido moverse entre todos los nodos N, salir de la bodega hacia los clientes y entrar a la bodega después de visitar clientes. Además que un vehículo use el arco $(0, n + 1)$ representa a un vehículo que no tiene asignada una ruta y por consiguiente permanece en la bodega. Además se fija $c_{0,n+1} = t_{0,n+1} = 0$.

La formulación de programación entera-mixta del VRPTW involucra 2 tipos de variables: para cada arco $(i, j) \in A$ y cada vehículo $k \in K$ hay una variable binaria x_{ijk} que es igual a 1 si el arco (i, j) es usado por el vehículo k, y es igual a 0 en el caso contrario; y por cada vértice $i \in V$ y vehículo $k \in K$ hay una variable de tiempo T_{ik} que especifica el inicio del tiempo de servicio en el vértice i cuando es servido por el vehículo k. La formulación del problema está dada por:

$$\text{mín} \quad \sum_{k\in K}\sum_{(i,j)\in A} c_{ij}x_{ijk} \tag{3.5}$$

$$\sum_{k\in K}\sum_{j\in\delta^+(i)} x_{ijk} = 1 \qquad \forall i \in N \tag{3.6}$$

$$\sum_{j\in\delta^+(0)} x_{0jk} = 1 \qquad \forall k \in K \tag{3.7}$$

$$\sum_{i\in\delta^-(j)} x_{ijk} - \sum_{i\in\delta^+(j)} x_{jik} = 0 \qquad \forall k \in K, j \in N \tag{3.8}$$

$$\sum_{i\in\delta^-(n+1)} x_{i,n+1,k} = 1 \qquad \forall k \in K \tag{3.9}$$

$$x_{ijk}(T_{ik} + st_i \cdot t_{ij} - T_{jk}) \leq 0 \qquad \forall k \in K, (i,j) \in A \tag{3.10}$$

$$e_i \leq T_{ik} \leq l_i \qquad \forall k \in K, i \in V \tag{3.11}$$

$$\sum_{i\in N} q_i \sum_{j\in\delta^+(i)} x_{ijk} \leq Q \qquad \forall k \in K \tag{3.12}$$

$$x_{ijk} \in \{0,1\} \qquad \forall k \in K, (i,j) \in A \tag{3.13}$$

La restricción (3.10) puede ser linealizada por:

$$T_{ik} + st_i + t_{ij} - T_{jk} \leq (1 - x_{ijk})M_{ij} \quad \forall k \in K, (i,j) \in A,$$

donde M_{ij}, $(i,j) \in A$ son constantes de gran valor que pueden ser fijadas en máx$\{l_i + st_i + t_{ij} - e_j, 0\}$.

3.3. Descripción del problema de ruteo de vehículos dinámico

En las secciones anteriores se describió matemáticamente la versión estática del problema de ruteo de vehículos, agregando restricciones sobre las capacidades de los vehículos y sobre ventanas de tiempo de los clientes.

La idea es plantear el problema dinámico usando también estas restricciones.

En esta tesis se centrará en el caso en que los costos por cada arco corresponden a la distancia euclideana y la distancia Manhattan. En un espacio euclideano, cada cliente $i \in \{1, ..., n\}$ tiene una posición $(x_i, y_i) \in \mathbb{R}^2$. Es por ello que el costo por arco c_{ij} en el caso de distancia euclideana está dado por

$$c_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{3.14}$$

Y en el caso de distancia Manhattan está dado por

$$c_{ij} = |(x_i - x_j)| + |(y_i - y_j)| \tag{3.15}$$

La distancia euclideana, a pesar de ser una distancia poco realista para una flota de vehículos que se desplaza en una ciudad, sí puede aplicarse en el contexto de una flota de drones. Por otro lado, puede dar un buen indicio para ver si los algoritmos funcionan correctamente. La distancia Manhattan sí se aplica mucho más a lo que podría ser un vehículo circulando por las calles de una ciudad.

Otra alternativa habría sido desligarse del plano euclideano y enfocar el problema mediante un esquema de grafos cuyos arcos representaran la distancia entre los nodos. Esta alternativa hubiera sido mucho más generalizable a lo que representan las distancias entre distintos puntos de una ciudad. Sin embargo, en la literatura de algoritmos de aprendizaje reforzado que resuelven problemas como el VRP, siempre usan como entrada un espacio euclideano, lo que hace pensar que esta variable es importante en este tipo de algoritmos. A pesar de todo esto, podría ser interesante ver la posibilidad de modelar el problema mediante un esquema de grafos, sin depender de distancias en un plano cartesiano, para que la resolución sea más cercana a la realidad.

En cuanto a los tiempos de viaje, estos también van a ser simplificados en esta tesis, ya que se va a asumir que cada vehículo va a tener una velocidad constante v. Esto dice que los tiempos de viaje entre cada par de clientes está dado por

$$t_{ij} = \frac{c_{ij}}{v} \tag{3.16}$$

Al igual como se describió en el VRPTW, se tendrá una única bodega desde la cual parten sus rutas los vehículos. Esta tendrá un horario de operación $[e_0, l_0]$, tal cual se explicó en el VRPTW.

La variable que le entrega el "dinamismo" a este problema es la correspondiente al tiempo en que se haga la solicitud de un cliente. Es por ello que se define ct_i que corresponde a la hora en que el cliente i realiza su solicitud, con $e_0 \leq ct_i \leq l_0$.

Por otro lado, también se va a asumir que el tiempo de servicio va a ser constante para cada cliente, el cual va a estar denotado por st.

En general cada cliente va a estar representado por la tupla $(x_i, y_i, q_i, e_i, l_i, ct_i)$.

Al inicio del horario de operación se va a asumir que se tiene un conjunto de n_0 clientes, que pueden ser interpretados como ordenes que fueron ingresadas antes de que empiece el horario de operación. Aquellos clientes van a tener un tiempo de solicitud $ct_i = e_0$. Se va a denotar por n' el número de clientes que realicen solicitudes dinámicas, es decir, sus tiempos de solicitudes son $ct_i > e_0$. Cabe notar que a priori n' no es conocido, por lo que es una variable aleatoria.

Se denota por $n = n_0 + n'$ el número de clientes totales que ingresan al sistema en un día de operación. Entonces el problema se reduce a encontrar un conjunto de *rutas factibles* $(R_1, ..., R_m)$ que contengan a todos los clientes, con tal de minimizar la distancia recorrida esperada y los retrasos esperados

$$\text{mín}\, \mathbb{E}\Big(\sum_{j=1}^{m} C_{R_j} + \alpha L_{R_j} \Big), \tag{3.17}$$

con $\alpha > 0$ una constante de penalización para los retrasos. Más adelante se verá que por las características de los algoritmos a utilizar, la condición de que todos los clientes sean visitados va a ser incluida como una penalización en la función a minimizar.

El intervalo de operación de los vehículos $[e_0, l_0]$ va a ser dividido en P periodos de decisión $t_1, t_2, ..., t_P$ con $t_1 = e_0 < t_2 < ... < t_P < l_0$. Los subintervalos resultantes van a ser equiespaciados, es decir $t_p = (p-1)\Delta + e_0$ para $i = 1, ..., P$, donde $\Delta > 0$ es el tiempo entre dos periodos consecutivos.

En cada periodo de decisión $p \in \{1, ..., P\}$ va a estar permitido construir rutas para los clientes que no han sido atendidos. Esto incluye a los clientes que se hicieron conocidos entre el periodo $p-1$ y el p, es decir, cuyos tiempos de solicitud ct cumplan que $t_{p-1} < ct < t_p$.

Capítulo 4

El problema desde una perspectiva del aprendizaje reforzado

Ya definido el problema de ruteo de vehículos dinámico, corresponde en este capítulo plantear algunas alternativas de solución. Como ya se ha comentado, se plantea resolverlo desde una perspectiva de algoritmos de aprendizaje reforzado.

En este capítulo primero se dan las definiciones correspondientes a los procesos de decisión de Markov que son claves para formalizar el problema como uno de aprendizaje reforzado. A continuación se abordan tres enfoques de algoritmos mediante los cuales se pretende resolver este problema.

4.1. Procesos de decisión de Markov: definiciones y propiedades

En esta sección se muestra la definición de un proceso de decisión de Markov (PDM) y los principales componentes de este. Todas las definiciones y conceptos pueden ser vistos en [28] y [34].

Definición 4.1 *Un proceso de decisión de Markov (PDM) es una tupla* $\{T, S, A_s, p_t(\cdot|s,a), r_t(s,a)\}$ *donde*

- T *es el conjunto de épocas de decisión*
- S *es el conjunto de posibles estados*
- A_s *es el conjunto de posibles acciones en un estado* $s \in S$
- $r_t(s,a)$ *es la recompensa que recibe el agente en la época de decisión* $t \in T$*, después de escoger la acción* $a \in A_s$ *en el estado* $s \in S$
- $p_t(\cdot|s,a)$ *es la distribución de probabilidad del siguiente estado en la época de decisión* $t \in T$ *y el estado* $s \in S$*, después de escoger la acción* $a \in A_s$*. Esta función se le conoce como la función de transiciones.*

Los conjuntos S y A_s pueden ser

1. *conjuntos finitos arbitrarios,*
2. *conjuntos numerables arbitrarios,*
3. *subconbjuntos compactos de un espacio de dimensión finita, o*
4. *subconjuntos Borel no vacíos de espacios métricos, separables y completos*

En esta tesis se asumirá que las funciones de recompensas y la función de transiciones no dependen del tiempo, es decir $r_t(s,a) = r(s,a)$ y $p_t(\cdot|s,a) = p(\cdot|s,a)\ \forall t \in T$. También se asumirá que la función de recompensas es determinista (también puede ser estocástica, pero más adelante se verá que al menos en este trabajo no se necesita que sea tan general).

Definición 4.2 *Una política $\pi(\cdot|s)$ es una función del conjunto de estados a las distribuciones de probabilidad en el espacio de acciones A_s. Luego $\pi(a|s)$ define una distribución de probabilidad sobre $a \in A_s$ para cada estado $s \in S$. La política va a ser la herramienta con la cual el agente va a tomar decisiones en cada estado.*

Definición 4.3 *Se denota por R_t como los retornos descontados a tiempo t*

$$R_t := \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \tag{4.1}$$

donde $r_{t+1}, r_{t+2}, ...$ es la secuencia de recompensas recibidas después del tiempo t, y γ es un hiper parámetro, $0 \leq \gamma \leq 1$, llamado tasa de descuento.

Definición 4.4 *La función valor en el estado s bajo la política π, denotada por $V_\pi(s)$ corresponde a los retornos esperados empezando en s y usando π como política. Formalmente*

$$V_\pi(s) := \mathbb{E}_\pi(R_t|s_t = s) = \mathbb{E}_\pi(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t = s) \tag{4.2}$$

Definición 4.5 *Se define el valor de tomar la acción a en el estado s bajo la política π, denotado por $Q_\pi(s,a)$, como los retornos esperados empezando del estado s, tomando la acción a y siguiendo la política π:*

$$Q_\pi(s,a) := \mathbb{E}_\pi(R_t|s_t = s, a_t = a) = \mathbb{E}_\pi(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t = s, a_t = a) \tag{4.3}$$

A esta función se le denotará por función acción-valor o q-valor

Definición 4.6 *La define la función de ventajas $A_\pi(s,a)$ en el estado s cuando se toma la acción a bajo la política π, como la diferencia entre la función acción-valor y la función valor*

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s) \tag{4.4}$$

Definición 4.7 *Una política π es mejor o igual a una política π' si sus retornos esperados son mayores o iguales que los de π' para todos los estados. En otras palabras $\pi \geq \pi'$ ssi $V_\pi(s) \geq V_{\pi'}(s)\ \forall s \in S$. A la política óptima se le denotará por π^*.*

Definición 4.8 *Se denotará por V_* a la función valor óptima, definida como:*

$$V_*(s) = \max_\pi V_\pi(s) \tag{4.5}$$

para todo $s \in S$. Las políticas óptimas también comparten la misma función accion-valor, denotada por Q_* y definida como:

$$Q_*(s,a) = \max_\pi Q_\pi(s,a) \tag{4.6}$$

para todo $s \in S$ y $a \in A_s$. Esta función entrega, para cada par (s,a) los retornos esperados al tomar la acción a en el estado s usando la política óptima. Luego

$$Q_*(s,a) = \mathbb{E}(r(s_t,a_t) + V_*(s_{t+1})|a_t = a, s_t = s) \tag{4.7}$$

Teorema 4.9 *Para cualquier proceso de decisión de Markov*

- *Existe una política óptima π^*, es decir, existe una política π^* tal que $V_{\pi^*} \geq V_\pi(s)$ para todas las políticas π y estados $s \in S$*
- *Todas las políticas óptimas alcanzan la función valor óptima, es decir, $V_{\pi^*}(s) = V_*(s)$ para todo $s \in S$ y para todas las políticas óptimas π^**
- *Todas las políticas óptimas alcanzan la función acción-valor óptima, es decir, $Q_{\pi^*}(s,a) = Q_*(s,a)$ para todo $s \in S$, para todo $a \in A_s$ y para todas las políticas óptimas π^**

Demostración Primero se demuestra que para dos políticas óptimas π_1 y π_2, $V_{\pi_1}(s) = V_{\pi_2}(s) \quad \forall s \in S$. En efecto como π_1 es política óptima, luego $V_{\pi_1}(s) \geq V_{\pi_2}(s) \quad \forall s \in S$. Por otro lado como π_2 es política óptima, $V_{\pi_2}(s) \geq V_{\pi_1}(s) \quad \forall s \in S$. Esto implica que π_1 y π_2, $V_{\pi_1}(s) = V_{\pi_2}(s) \quad \forall s \in S$. Como consecuencia de esto, se tendría que demostrar que existe una política óptima π^* que alcanza la función valor óptima y la función acción-valor óptima. Se considera la siguiente política determinista $\pi^* : S \to A$:

$$\pi^*(s) = \arg\max_{a \in A} Q_*(s,a) \quad \forall s \in S$$

Como $\pi^*(s) = \arg\max_{a \in A} Q_*(s,a)$ y $V_*(s) = \max_{a \in A} Q_*(s,a) \quad \forall s \in S$, π^* establece la acción óptima para cada estado (lo cual produce la función valor óptima V_*). Luego, la política π^* en cada estado va a generar la misma función valor y la misma función acción-valor. De esta forma $V_{\pi^*}(s) = V_*(s) \quad \forall s \in S$ y $Q_{\pi^*}(s,a) = Q_*(s,a) \quad \forall s \in S, a \in A$. Finalmente se prueba por contradicción que π^* es una política óptima. Se supone que π^* no es óptima, luego existe una política π y un estado $s \in S$ tal que $V_\pi(s) > V_{\pi^*}(s)$. Como $V_{\pi^*}(s) = V_*(s)$, se tiene que $V_\pi(s) > V_*(s)$ que contradice la definición de $V_*(s) = \max_\pi V_\pi(s)$.

Proposición 4.10 *Ecuaciones de optimalidad de Bellman: se tiene que la función valor y la función acción-valor bajo una política óptima deben satisfacer que*

$$\begin{aligned} V_*(s) &= \max_{a \in A_s} Q_{\pi^*}(s,a) \\ &= \max_a \mathbb{E}_{\pi^*}(R_t | s_t = s, a_t = a) \\ &= \max_a \mathbb{E}_{\pi^*}(r(s_t,a_t) + \gamma R_{t+1} | s_t = s, a_t = a) \\ &= \max_a \mathbb{E}(r(s_t,a_t) + \gamma V_*(s_{t+1}) | s_t = s, a_t = a) \\ &= \max_a \sum_{s' \in S} p(s'|s,a)(r(s,a) + \gamma V_*(s')) \end{aligned}$$

$$\begin{aligned} Q_*(s,a) &= \mathbb{E}(r(s_t,a_t) + \gamma V_*(s_{t+1}) | s_t = s, a_t = a) \\ &= \mathbb{E}(r(s_t,a_t) + \gamma \max_{a'} Q_*(s_{t+1},a') | s_t = s, a_t = a)) \\ &= \sum_{s' \in S} p(s'|s,a)(r(s,a) + \gamma \max_{a'} Q_*(s',a')) \end{aligned}$$

Las dos últimas ecuaciones de cada demostración corresponden a las ecuaciones de optimalidad de Bellman. La última línea de ambas ecuaciones es válida cuando el espacio de estados es numerable.

4.2. Aprendizaje reforzado profundo

El objetivo del aprendizaje reforzado es encontrar una política π^* que maximice los retornos esperados totales, es decir, una política óptima. Las definiciones anteriormente explicitadas servirán para construir algoritmos que permitan encontrar políticas que generen soluciones para el problema de ruteo de vehículos dinámico. Sin embargo, antes de aquello se hablará de una estructura de funciones mediante las cuales se piensa modelar los conceptos antes mencionados. Estas estructuras corresponden a las redes neuronales profundas, que han sido muy populares en los últimos años gracias a sus buenos resultados en temas aplicados. Por otro lado el gran poder computacional que se ha desarrollado en los últimos años, han permitido usarlas en gran escala para resolver problemas complejos. En particular la mezcla entre redes neuronales profundas y aprendizaje reforzado se le conoce como aprendizaje reforzado profundo o *Deep Reinforcement Learning* en inglés.

4.2.1. Redes neuronales profundas

Formalmente una red neuronal profunda es una estructura con múltiples capas, donde cada capa consiste en una transformación lineal seguida por una función de "activación" no lineal. Matemáticamente, para una entrada $x \in \mathbb{R}^n$, una capa $\mathcal{L}$ realiza la transformación

$$\mathcal{L}(x) = \varphi(Wx + b),$$

donde $W \in \mathbb{R}^{m \times n}$ es una matriz (llamada matriz de parámetros o pesos), $b \in \mathbb{R}^m$ es un vector (llamado vector de sesgos) y φ es una función no lineal. Los objetos W y b contienen

parámetros libres θ_W y θ_b. Típicamente, la función de activación φ no contiene parámetros y actúa elemento a elemento en sus entradas.

La palabra "profundo" en aprendizaje profundo (o *deep learning*) viene de componer múltiples capas de este tipo, tal que la salida de una capa es usada como entrada para la siguiente. En general cada capa $\mathcal{L}$ va a tener sus propios parámetros. Resumiendo todo el modelo de parámetros por θ, un modelo de red neuronal con N capas está dado por

$$y = f_\theta(x) = \mathcal{L}_N \circ \cdots \circ \mathcal{L}_1(x),$$

y transforma una entrada x a una salida final y.

Este modelo de redes neuronales profundas es el más básico, pero este puede ser extendido a redes más complejas como las redes neuronales recurrentes o redes de atención [13, 42]. En esta tesis se usará una mezcla de estas redes, las cuales van a estar explicitadas en el apéndice.

4.3. Algoritmo Q-learning para el DVRP

En esta sección se propondrá un algoritmo para resolver el problema de ruteo de vehículos dinámico. Como primera tarea se expondrá el proceso de decisión de Markov con el que se va a modelar el problema, para posteriormente introducir el algoritmo y hacer la conexión con el modelo.

La intuición de este algoritmo es aproximar, mediante redes neuronales, la función acción valor $Q(s, a)$ óptima para cada estado $s \in S$ y acción $a \in A_s$, donde S y A_s van a estar dados por el PDM. Con ello la política óptima, en caso en que la aproximación sea ajustada con la realidad, va a estar dada por $\arg\max_a Q(s, a)$. Este algoritmo es del tipo *model free* ya que no tiene un modelo explícito del ambiente, en cuánto a la distribución estocástica de las solicitudes dinámicas de los clientes.

4.3.1. Proceso de decisión de Markov

Antes de presentar las características del algoritmo se modela el problema mediante un PDM.

Es preciso recordar que el período de tiempo de operación $[e_0, l_0]$ se divide en P períodos, en los cuales al inicio de cada uno se mostrarán al sistema los nuevos clientes dinámicos que han solicitado algún servicio. Se va a denotar por $\mathcal{P}$ el conjunto de períodos de decisión.

- **Épocas de decisión**: En un principio el conjunto de épocas de decisión va a estar dado por $T = \mathbb{N}$, con $\mathbb{N}$ los números naturales. Después se va a decir cual es el estado terminal con el que se termina el proceso.
- **Estados**: Se va a diferenciar entre tres tipos de estados, correspondientes a las características de los clientes, de los vehículos y el período $p \in \mathcal{P}$ en que se encuentra el agente.
    1. Primero se define la variable b_k^t, $k = 1, ..., m$, que va a codificar el último cliente visitado por el vehículo k en la época de decisión t. Si un vehículo k no ha visitado clientes $b_k^t = 0$ (es decir está en la bodega).

Se denota por $\mathcal{N}_t$ el conjunto de clientes que no han sido visitados hasta la época de decisión t.

Se define G_t correspondiente al grafo completo conformado por los clientes no visitados, el conjunto de clientes previos y la bodega. Es decir $G_t = (V_t, A_t)$ con $V_t = \{0\} \cup \mathcal{N}_t \cup \{b_1^t, ..., b_m^t\}$.

Sea S_t^1 el conjunto de estados correspondiente a la bodega y los clientes representados por los nodos de grafo G_t definido recientemente.

Cada nodo del grafo tiene todas las características correspondientes al cliente o bodega que representa: posición, demanda, ventana de tiempo y tiempo en que se realiza la solicitud, es decir, siguiendo la notación del capítulo anterior cada nodo estaría representado por la tupla (x, y, q, e, l, ct). La posición $(x, y) \in \mathbb{R}^2$, la demanda $q \in \mathbb{R}_+$, la ventana de tiempo $(e, l) \in [e_0, l_0]^2$ y la hora de solicitud $ct \in [e_0, l_0]$. Se denota por n_t el número de nodos en el grafo G_t, en la época de decisión t, luego

$$S_t^1 = \mathbb{R}^{2 \cdot n_t} \times \mathbb{R}_{>0}^{n_t} \times [e_0, l_0]^{2 \cdot n_t} \times [e_0, l_0]^{n_t}$$

2. Sea S_t^2 el estado de los m vehículos a tiempo t. El estado de un vehículo a tiempo t corresponde a la posición de este, la bodega de origen, la hora en que se va a encontrar disponible y la capacidad actual. La posición y la bodega de origen puede ser codificada por los nodos correspondientes del grafo G_t. Luego

$$S_t^2 = V_t^m \times V_t^m \times \mathbb{R}_{>0}^m \times \mathbb{R}_{>0}^m$$

3. En realidad las acciones sólo se ejecutan en alguno de los períodos de decisión $p \in \mathcal{P}$. Es por esto que también se incluyen en los estados

Luego

$$S = \cup_{t \in T} S_t^1 \times S_t^2 \times \mathcal{P}$$

es el conjunto de estados.

- **Acciones**: Dado un estado s_t, se va a denotar por Q_k^t a la capacidad del vehículo $k \in \{1, ..., m\}$ en la época de decisión t y por IT_k^t la hora en que va a estar disponible el vehículo $k \in \{1, ..., m\}$ en la época de decisión t.

  Una acción va a corresponder a la selección de un par cliente vehículo $(i, k) \in \mathcal{N}_t \times \{1, ..., m\}$ dentro de un conjunto de pares factibles definidos por:

$$F(s_t) = \{(i, k) \in \mathcal{N}_t \times \{1, ..., m\} |\ q_i \leq Q_k^t,\ \ \text{máx}\{e_i, IT_k^t + t_{i_k, i}\} + st + t_{i,0} \leq l_0\}$$

  Es decir, los pares factibles corresponden a las acciones tal que se respete la capacidad de los vehículos y el horario de cierre de la bodega.

  También se agrega la acción de poder pasar al siguiente período de decisión. En este caso se asume que ya no se tomarán más acciones correspondientes al período actual. En resumen $A_s = F(s) \cup \{pass\}$

- **Recompensa**: En el caso de que se seleccione un par $(i, k) \in F(s_t)$, la recompensa va a estar dada por el costo negativo de asignar el cliente i al vehículo k. Este costo va a

estar dado por la distancia recorrida y una penalización en caso de que no se respete la ventana de tiempo del cliente. En caso en que la acción sea pasar a la siguiente época de decisión la recompensa es 0. Una de las restricciones para este problema era que se tenían que visitar todos los clientes. La forma más fácil de hacer esto con un agente de aprendizaje reforzado es imponiendo una penalización en las recompensas por cada cliente que no es visitado. Como esta es una restricción casi mandatoria, el valor de la penalización debe ser mayor al valor de la penalización por retrasos. Esta penalización y los costos que tienen los vehículos para retornar a la bodega se va a sumar cuando se esté en el último período $p = P$. Luego, para $s_t = (s_t^1, s_t^2, p) \in S$, $\alpha, \beta > 0$

$$r(s_t, a_t) = \begin{cases} -C(s_t, a_t) - \alpha L(s_t, a_t) & \text{si } a_t = (i, k) \\ 0 & \text{si } a_t = pass \text{ y } p \neq P \\ -RD(s_t) - \beta NV(s_t, a_t) & \text{si } a_t = pass \text{ y } p = P \end{cases} \tag{4.8}$$

Donde para $a_t = (i, k)$,

$$C(s_t, a_t) = c_{b_k^t, i}$$

Es decir, la distancia entre la última posición del vehículo k y el nodo i.

A diferencia de lo descrito en el capítulo 3, por temas computacionales y numéricos, la penalización sobre los clientes que se llegan retrasados se hará de manera discreta, es decir, por cada cliente retrasado se tendrá una penalización de $\alpha > 0$. En otras palabras:

$$L(s_t, a_t) = \begin{cases} 1 & \text{si } IT_k^t + t_{b_k^t, i} - l_i > 0 \\ 0 & \text{sino} \end{cases} \tag{4.9}$$

Por otro lado, $RD(s_t)$ es la distancia que deben recorrer los vehículos para volver a las respectivas bodegas y $NV(s_t, a_t)$ es la cantidad de nodos no visitados.

- **Función de transiciones**: Cuando se toma una acción correspondiente a un par nodo-vehículo (i, k), la transición es determinista ya que debe actualizarse el conjunto de nodos previos para cada vehículo, debe actualizarse el conjunto de nodos $\mathcal{N}_t$ y debe actualizarse la capacidad y el tiempo de disponibilidad del vehículo k. Específicamente las actualizaciones son

$$b_k^{t+1} = i$$

$$\mathcal{N}_{t+1} = \mathcal{N}_t \setminus \{i\}$$

$$Q_k^{t+1} = Q_k^t - q_i$$

$$IT_k^{t+1} = \text{máx}\{e_i, IT_k^t + t_{l_k^t, i}\} + st$$

Por otro lado, si $a = pass$, significa que se pasa al siguiente período de decisión. En este caso se actualiza el grafo G_t con los nuevos clientes que ingresan nuevas solicitudes. Este paso es estocástico ya que no se sabe con certeza cuántas nuevas solicitudes ingresan y qué estado van a tener esos clientes. Suponiendo que $\mathcal{O}_p$ es el conjunto de clientes que realizaron solicitudes entre los períodos $p - 1$ y p, entonces:

$$\mathcal{N}_{t+1} = \mathcal{N}_t \cup \mathcal{O}_{p+1}$$

4.3.2. Algoritmo vainilla Deep Q-learning

Como ya se introdujo, la idea de esta solución es aproximar mediante redes neuronales la función acción valor óptima $Q(s,a)$ para cada estado s y cada acción $a \in A_s$. Con ello la política va a estar dada por la estrategia glotona

$$\pi(a|s) = \begin{cases} 1 & \text{para } a = \arg\max_{a'} Q(s,a') \\ 0 & \text{sino} \end{cases} \tag{4.10}$$

Para fijar ideas es preciso recordar que la función acción valor óptima estaba dada por:

$$Q_*(s,a) = \max_{\pi} \mathbb{E}_\pi(R_t|s_t = s, a_t = a),$$

con $R_t := \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$ los retornos esperados totales a tiempo t.

De la ecuación de optimalidad de Bellman se obtiene que

$$Q_*(s,a) = \mathbb{E}_{s' \sim p(\cdot|s,a)}(r(s_t,a_t) + \gamma \max_{a'} Q_*(s',a')|s_t = s, a_t = a))$$

Para estimar la función acción valor óptima se utiliza una red neuronal con parámetro θ tal que $Q(s,a;\theta) \approx Q_*(s,a)$

El algoritmo de aprendizaje resulta de iterativamente minimizar la siguiente función de perdidas o *loss*:

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}[(y_i - Q(s,a;\theta_i))^2], \tag{4.11}$$

donde $y_i = \mathbb{E}_{s'\sim p(\cdot|s,a)}[r(s,a) + \max_{a'} Q(s',a';\theta_{i-1})|s,a]$ es el valor "real" en la iteración i y $\rho(s,a)$ es distribución de probabilidad sobre estados y acciones llamada distribución de comportamiento. Los parámetros θ_{i-1} son dejados fijos cuando se optimiza la función de pérdidas $L_i(\theta_i)$.

Es posible notar que el gradiente de la función de pérdidas (salvo constantes) es igual a

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot),s'\sim p(\cdot|s,a)}[(r(s,a) + \gamma \max_{a'} Q(s',a';\theta_{i-1}) - Q(s,a;\theta))\nabla_{\theta_i} Q(s,a;\theta_i)] \tag{4.12}$$

Cabe notar que para el cálculo de la esperanza, computacionalmente es muy caro, por lo que se optimiza mediante descenso de gradiente estocástico [29].

Por otro lado, para esta aproximación se utiliza el concepto de aprendizaje por experiencias o *experience replay* [20], en el cual se van a tomar una secuencia de experiencias (s_t, a_t, r_t, s_{t+1}) que se van a ir guardando en una memoria $\mathcal{D}$ para luego usarlas en la actualización de parámetros de la red neuronal, usando descenso de gradiente estocástico.

En general para la distribución de comportamiento se toma una estrategia $\varepsilon-$greedy, en donde se selecciona con probabilidad $1-\varepsilon$ una estrategia greedy, es decir tomar $a = \max_a Q(s,a;\theta)$, y con probabilidad ε una acción al azar.

Usando todo lo anterior se resume en el algoritmo vainilla *Deep Q Networks* (DQN) [24]

Algorithm 1: DQN vainilla

Input: ρ: distribución de comportamiento, α: tasa de aprendizaje
Inicializar memoria $\mathcal{D}$ con capacidad N
Inicializar función acción-valor Q con pesos arbitrarios θ
while *No ha convergido* **do**
    Inicializar s_1
    for *t=1,...,T* **do**
        Obtener a_t según $\rho(s_t, \cdot)$
        Almacenar la secuencia (s_t, a_t, r_t, s_{t+1}) en $\mathcal{D}$
        Tomar un minibatch de transiciones (s_j, a_j, r_j, s_{j+1}) de $\mathcal{D}$ de manera uniforme
        Fijar $y_j = r_j + \gamma \,\text{máx}_{a'} Q(s_{j+1}, a'; \theta')$ para estado no terminal y $y_j = r_j$ para estado terminal
        $\theta \leftarrow \theta - \alpha \sum_j \nabla_\theta Q(s_j, a_j; \theta)(Q(s_j, a_j; \theta) - y_j)$
    Actualizar θ'

4.3.3. Una serie de mejoras al algoritmo DQN

Específicamente no se decide aplicar el algoritmo DQN al problema de ruteo de vehículos dinámico, sino que a este se le introducen una serie de mejoras que han salido en la literatura en los últimos 5 años. Sin embargo, la base del algoritmo es la misma, ya que todas estas mejoras van en seguir aproximando la función acción-valor $Q(s, a)$ óptima, mediante una red neuronal de parámetro θ. A continuación se detallan cada una de las mejoras aplicadas:

1. **Algoritmo Double DQN**: Una de las críticas que tiene el algoritmo DQN es que cuando se fija $y_t = r_t + \text{máx}_a \, Q(s_{t+1}, a; \theta_t)$ es posible que se estén sobre estimando los valores cuando se usa la operación del máximo, resultando valores demasiado optimistas a lo que en realidad debería ser.

   Esto último se ha demostrado tanto empíricamente como teóricamente [41].

   Para evitar esto, una idea es tener dos funciones acción-valor con parámetros θ y θ', tal que una sirva para determinar la política glotona y la otra para determinar los valores. Como la función acción valor va a ser aproximada por redes neuronales, a la que tiene el parámetro θ se le llamará red primaria y a la que tiene el parámetro θ' se le llamará red objetivo. Explícitamente, se puede escribir la predicción y_t como:

   $$y_t = r_t + \gamma Q(s_{t+1}, \arg\max_a(Q(s_{t+1}, a; \theta_t)); \theta_t),$$

   donde θ_t se usa para la selección de acciones que maximizan la función acción-valor y θ'_t para la evaluación de esta política.
2. **Memoria con experiencias priorizadas**: Otra de las mejoras más importantes para el algoritmo vainilla DQN es una modificación en la memoria de experiencias. Esta mejora se llama memoria con experiencias priorizadas o *Prioritized Experience Relay* en inglés [31]. Esta consiste en que la extracción de experiencias (s_t, a_t, r_t, s_{t+1}) no se haga de manera uniforme, sino que se extraigan aquellas experiencias que podrían

resultar más importantes para el entrenamiento. Para esto la idea es asignar a cada experiencia un "peso" que indique cuáles son las secuencias con más prioridad para el entrenamiento.

Una pregunta importante es cómo se le asigna importancia a cada experiencia. Lo que se plantea es usar como magnitud de priorización el valor δ correspondiente al error de las diferencias temporales, es decir, en el caso de *Double DQN* es

$$\delta_i = r_i + \gamma Q(s_{i+1}, \arg\max_{a'} Q(s_{i+1}, a'; \theta); \theta') - Q(s_i, a_i; \theta) \tag{4.13}$$

Este valor estaría diciendo que tan "sorprendente" o inesperada es una transición. Una acotación que se hace sobre esta mejora, es que la extracción de experiencias tiene que seguir siendo estocástica, ya que si hace de forma glotona esta priorización se va a centrar en un bajo número de experiencias pudiendo aumentar los errores. Por otro lado, la baja diversidad de experiencias puede resultar en que la función se sobre ajuste.

Para que la probabilidad de extracción sea monótona a la prioridad de la experiencia, para $p_i > 0$ la prioridad de la secuencia i se define la probabilidad de extracción de la secuencia i como

$$P(i) = \frac{p_i^\alpha}{\sum_k p_i^\alpha},$$

donde α corresponde a qué tanta priorización se le asigna, con $\alpha = 0$ correspondiendo al caso uniforme.

Conectando esto con lo anterior, $p_i = |\delta_i| + \bar{\varepsilon}$, donde $\bar{\varepsilon}$ es una constante positiva muy pequeña que evita que el denominador sea 0 cuando ocurre el caso de borde en que $\delta_i = 0\ \forall i$.

Un detalle técnico que se añade con esta modificación es que al extraer secuencias de experiencias con una distribución distinta a la uniforme, se está agregando sesgo a la estimación de la esperanza involucrada en la función de pérdidas del algoritmo. Una forma de corregir esto es mediante la técnica de *Importance Sampling* con pesos [22] que permite compensar la no uniformidad de muestras que se obtienen de la memoria de experiencias.

En particular, si N es el número de experiencias que tiene la memoria, cada uno de los pesos va a estar dado por

$$w_i = (\frac{1}{N} \cdot \frac{1}{P(i)})^\beta$$

Es posible notar que si $\beta = 1$ se logra compensar totalmente el cambio de distribución en el cálculo de la esperanza. Para ver esto sea $x_i = (s_i, a_i, r_i, s_i') \sim \mathcal{D}$, y $f(x_i) = r_i + \gamma Q(s_i, \arg\max_{a'} Q(s_i', a_i'; \theta); \theta') - Q(s_i, a_i; \theta)$. También se fija $\beta = 1$. Luego

$$\mathbb{E}_{x_i \sim \mathcal{D}}[f(x_i)] = \sum_{i=1}^{N} \frac{P(i)}{P(i)} f(x_i) \frac{1}{N} = \sum_{i=1}^{N} w_i P(i) f(x_i) = \mathbb{E}_{x_i \sim \mathcal{P}}[w_i f(x_i)]$$

Sin embargo, en [31] se propone ir incrementando este hiperparámetro a lo largo del entrenamiento hasta llegar a 1, ya que "la naturaleza insesgada de las actualizaciones en el aprendizaje reforzado es más importante cuando va convergiendo el algoritmo al término del entrenamiento". Otro detalle que se implementa, es que para evitar errores

de estabilidad se propone normalizar los pesos w_i, tal que $\max_i w_i = 1$ (es decir cada peso se multiplica por $1/\max_i w_i$).

3. **Mejoras técnicas que empíricamente mejoran efectividad**: Existen mejoras que han sido agregadas a estos algoritmos mediante la experiencia y recomendaciones de cursos relativos al aprendizaje reforzado [39].

   Una de ellas es que la actualización del parámetro θ' con respecto al parámetro θ se realiza mediante

   $$\theta' \leftarrow \tau\theta + (1-\tau)\theta'$$

   Para $\tau << 1$. Otra forma de actualizar el parámetro θ' es igualarlo completamente al parámetro θ cada i iteraciones del algoritmo (es decir cada i actualizaciones de parámetros con descenso de gradiente). Ambas estrategias se probarán para ver cuál es más efectiva.

   Una última estrategia que se utiliza es ir modificando el hiperparámetro de exploración ε tal que vaya convergiendo a 0 a medida que avanza el entrenamiento. De esta forma en un principio se toman acciones aleatorias hasta que después de un cierto número de iteraciones se toman acciones de forma glotona.

4.3.4. Algoritmo Double DQN con experiencias priorizadas

Ya planteadas todas las mejoras al algoritmo vainilla DQN, se continua con un nuevo algoritmo que va a contener todas las modificaciones propuestas. Cabe recordar que el espacio de estados y acciones para el problema va a estar dado por PDM descrito anteriormente. A continuación se muestra el algoritmo *Double DQN* con experiencias priorizadas:

Algorithm 2: Algoritmo Double DQN con experiencias priorizadas

Input: B: tamaño mini-batch, N: tamaño de la memoria, $\varepsilon \in (0,1)$, $\tau << 1$, exponentes α y β

Inicializar memoria $\mathcal{D}$ con capacidad N
Inicializar funciones acción-valor Q_θ y $Q_{\theta'}$ con pesos arbitrarios
while *No ha convergido* **do**
    Inicializar s_1
    for *t=1,...,n* **do**
        Obtener $a_t = \arg\max_a Q_\theta(s_t, a)$ con prob. $1-\varepsilon$ y $a_t = \text{Random}(A_{s_t})$, con A_{s_t} el conjunto de acciones factibles, con prob. ε
        Ejecutar a_t y observar s_{t+1} y r_t
        Almacenar la secuencia (s_t, a_t, r_t, s_{t+1}) en $\mathcal{D}$ con máxima prioridad $p_t = \max_{j<t} p_j$
        Tomar un minibatch de tamaño B con transiciones (s_j, a_j, r_j, s_{j+1}) de $\mathcal{D}$ con prob. $P(j)$
        Computar para cada j los pesos $w_j = (N \cdot P(j))^{-\beta} / \max_i w_i$
        Computar error $\delta_j = r_j + \gamma Q(s_{j+1}, \arg\max_{a'} Q(s_{j+1}, a'; \theta); \theta') - Q(s_j, a_j; \theta)$
        Actualizar prioridades $p_j \leftarrow |\delta_j|$
        $\nabla_\theta \mathcal{L} \leftarrow \sum_j \nabla_\theta Q_\theta(s_j, a_j) \cdot w_j \cdot \delta_j$
        $\theta \leftarrow \text{Optimizer}(\theta, \nabla_\theta \mathcal{L})$
        Actualizar $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$ cada ciertas iteraciones

Acá, la línea 14 se refiere a realizar descenso de gradiente estocástico con una tasa de aprendizaje dada. Por otro lado, como ya se dijo antes, el valor del hiperparámetro ε irá disminuyendo a través del entrenamiento, para pasar de una estrategia completamente aleatoria a una glotona.

4.4. Algoritmo tipo *Policy Gradients* para el DVRP

En la sección pasada se mostró un algoritmo de aprendizaje reforzado profundo, cuyo principal objetivo era realizar una aproximación de la función acción-valor óptima, para luego usarla como política para la toma de acciones.

En esta sección se abordará el problema desde otro enfoque. Ahora en vez de aproximar la función acción-valor, se aproximará directamente la política estocástica del agente.

Antes de todo, para poder resolver el DVRP con esta clase de algoritmos es necesario definir el proceso de decisión de Markov que modela el problema. Este es mayoritariamente similar al definido para los algoritmos Q-learning, pero de igual forma se explicitará para que no queden dudas en su formulación.

Al igual que los algoritmos Q-learning, los algoritmos *policy gradients* son *model free* ya que no tienen un modelo explícito del ambiente, en cuánto a la distribución estocástica de las solicitudes dinámicas de los clientes.

4.4.1. Proceso de decisión de Markov

Es preciso recordar que el período de tiempo de operación $[e_0, l_0]$ se divide en P períodos, en los cuales al inicio de cada uno se mostrarán al sistema los nuevos clientes dinámicos que han solicitado algún servicio.

A continuación se explicita cada elemento del PDM. Cabe notar que en este caso el PDM no se va a centrar en acciones individuales sobre las asignaciones a vehículos, sino que en esta formulación se va a hablar de construir rutas como acciones.

- **Épocas de decisión**: En este caso las épocas de decisión van a coincidir con el conjunto de períodos, es decir $T = \{1, ..., P\}$. Esto se relaciona con las soluciones clásicas vistas en la literatura para el DVRP.
- **Estados**: Los estados para este algoritmo van a ser iguales a los estados del algoritmo Q-learning, pero esta vez vistos como estados de épocas de decisión.

  Recordando, se define la variable b_k^t, $k = 1, ..., m$, que va a codificar el último cliente visitado por el vehículo k en la época de decisión t. Si un vehículo k no ha visitado clientes $b_k^t = 0$ (es decir está en la bodega).

  Se denota por $\mathcal{N}_t$ el conjunto de clientes que no han sido visitados hasta la época de decisión t.

  Se define G_t correspondiente al grafo completo conformado por los clientes no visitados, el conjunto de clientes previos y la bodega. Es decir, $G_t = (V_t, A_t)$ con $V_t = \{0\} \cup \mathcal{N}_t \cup \{b_1^t, ..., b_m^t\}$.

En este sentido el conjunto de estados S_t^1 se mantendría igual. Luego para n_t el número de nodos en el grafo, se tiene que el espacio de estados S_t^1 es

$$S_t^1 = \mathbb{R}^{2 \cdot n_t} \times \mathbb{R}_{>0}^{n_t} \times [e_0, l_0]^{2 \cdot n_t} \times [e_0, l_0]^{n_t}$$

El estado S_t^2 correspondiente al estado de los m vehículos a tiempo t, va a ser igual al caso de algoritmo Q-learning, es decir, el estado de un vehículo a tiempo t corresponde a la posición de este, la bodega de origen, la hora en que se va a encontrar disponible y la capacidad actual. La posición y la bodega de origen puede ser codificada por los nodos correspondientes del grafo G_t. Luego

$$S_t^2 = V_t^m \times V_t^m \times \mathbb{R}_{>0}^m \times \mathbb{R}_{>0}^m$$

Finalmente $S = \cup_{t \in T} S_t^1 \times S_t^2$ es el conjunto de estados.

- **Acciones**: Dado un estado s_t, se va a denotar por Q_k^t a la capacidad del vehículo $k \in \{1, ..., m\}$ en la época de decisión t y por IT_k^t la hora en que va a estar disponible el vehículo $k \in \{1, ..., m\}$ en la época de decisión t. También se va a denotar por i_k al último nodo visitado por el vehículo k.

  Una acción va a corresponder a un plan de rutas factibles $a_t = (R_1^t, ..., R_m^t)$, donde para cada ruta asociada al vehículo $k \in \{1, ..., m\}$ va a contener un subconjunto de nodos del grafo G_t, con dicho subconjunto posiblemente vacío (en caso de que no se le asigne ninguna ruta a aquel vehículo). Cabe notar que el primer cliente de cada ruta va a estar dado por el último nodo visitado en la época de decisión previa.

  También se agrega la acción de poder pasar al siguiente período de decisión. En este caso se asume que ya no se tomarán más acciones correspondientes al período actual.

  Más adelante se explicitará cómo se construyen las rutas, para finalmente construir una acción.

- **Recompensa**: La recompensa de un plan de rutas va a estar dada por el costo de la distancia recorrida total de las rutas y la suma de las penalizaciones en caso de que no se respete la ventana de tiempo del cliente. En caso en que la acción sea pasar a la siguiente época de decisión la recompensa es 0. Por último queda sumar los costos de no haber visitado algún cliente y los costos de que los vehículos retornen a la bodega. Esta recompensa sólo se va a sumar cuando se esté en $t = P$ (en este caso no se está permitido pasar). Luego para $s_t \in S$, $\alpha, \beta > 0$

$$r(s_t, a_t) = \begin{cases} -C(s_t, a_t) - \alpha L(s_t, a_t) - RD(s_t) - \beta NV(s_t, a_t) & \text{si } t = P \\ 0 & \text{si } a_t = pass \\ -C(s_t, a_t) - \alpha L(s_t, a_t) & \text{si no} \end{cases} \quad (4.14)$$

  Suponiendo que una ruta R del plan de rutas factibles está dada por $R = (i_1, ..., i_{|R|})$, entonces el costo R de la ruta estaba dado por

$$C_R = \sum_{h=1}^{|R|-1} c_{i_h, i_{h+1}}$$

Luego, el costo $C(s_t, a_t)$ es la suma de los costos de todas las rutas:

$$C(s_t, a_t) = \sum_{k=1}^{m} C_{R_k}$$

Por otro lado, para la ruta $R = (i_1, ..., i_{|R|})$, la penalización discreta por llegar tarde al nodo i_h, con $h \in \{2, ..., |R|\}$, es

$$L_{i_h} = \begin{cases} 1 & \text{si } D_{i_{h-1}} + t_{i_{h-1}, i_h} - l_{i_h} > 0 \\ 0 & \text{si no} \end{cases} \tag{4.15}$$

donde D_{i_h} es la hora en que el vehículo sale del cliente i_h. Luego, la penalización discreta de la ruta completa sería

$$L_R = \sum_{h=2}^{|R|} L_{i_h}$$

Y la penalización discreta por todas las rutas simplemente es la suma de las penalizaciones discretas de cada ruta

$$L(s_t, a_t) = \sum_{k=1}^{m} L_{R_k}$$

Por otro lado $RD(s_t)$ es la distancia que deben recorrer los vehículos para volver a las respectivas bodegas y $NV(s_t, a_t)$ es la cantidad de nodos no visitados.

- **Función de transiciones**: Cuando se toma una acción, debe actualizarse el conjunto de nodos previos para cada vehículo, el conjunto de nodos $\mathcal{N}_t$ y debe actualizarse la capacidad y el tiempo de disponibilidad del vehículo k. La actualización de la hora de disponibilidad de cada vehículo IT_k^t es más complicada escribirla para una ruta, ya que se tiene que ir actualizando secuencialmente por cada nodo que tenga la ruta. Esta actualización se dejará explícita más adelante cuando se muestre cómo se construye el plan de rutas. Lo mismo ocurre con la actualización de las capacidades.

  Por otro lado suponiendo que $i^k_{|R_k|}$ es el último nodo visitado en la ruta del vehículo k, entonces

  $$b_k^{t+1} = i^k_{|R_k|} \quad \forall k \in \{1, ..., m\}$$

  Además, suponiendo que $\mathcal{C}_t \subset \mathcal{N}_t$ es el conjunto de nodos visitados en las rutas en la época de decisión t y $\mathcal{O}_t$ es el conjunto de clientes que realizaron solicitudes entre las épocas (o períodos) $t-1$ y t, entonces

  $$\mathcal{N}_{t+1} = \mathcal{N}_t \setminus \mathcal{C}_t \cup \mathcal{O}_{t+1}$$

4.4.2. Algoritmos tipo *Policy Gradients*

Se asumirá que la política del agente esta parametrizada por $\theta \in \mathbb{R}^d$. Se denota por $\pi_\theta(\cdot|s)$ a la política parametrizada, con $s \in S$. Tal como se hizo para Q-learning la función que va a aproximar la política va a ser una red neuronal profunda de parámetro $\theta \in \mathbb{R}^d$. Para simplificar los términos se asumirá que la tasa de descuento γ se fija en 1.

Se denota por $\tau = (s_0, a_0, s_1, a_1, ..., s_N, a_N)$ una trayectoria dada por la política $\pi_\theta(\cdot|s)$. Entonces

$$p_\theta(\tau) = p(s_0) \prod_{t=0}^{N} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t)$$

Se denota por $r(\tau)$ a la recompensa total entregada por la trayectoria τ, es decir:

$$r(\tau) = \sum_{t=0}^{N} r(s_t, a_t)$$

También se define $J(\theta)$ como la recompensa esperada al usar la política parametrizada por θ

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}[r(\tau)]$$

Luego se puede definir el siguiente problema de optimización, que es equivalente a encontrar una política óptima para el problema

$$\max_{\theta \in \mathbb{R}^d} J(\theta)$$

Es preciso recordar que los algoritmos de aprendizaje reforzado profundo, para ajustar los parámetros de la red neuronal se usa el algoritmo de descenso de gradiente estocástico [29]. Luego, si se quiere optimizar la función $J(\theta)$, con $\theta \in \mathbb{R}^d$ los parámetros de la red neuronal, se necesita calcular el gradiente de esta función. En efecto

$$\begin{aligned} \nabla_\theta J(\theta) &= \nabla_\theta \mathbb{E}_{\tau \sim p_\theta(\tau)}[r(\tau)] \\ &= \int \nabla_\theta p_\theta(\tau) r(\tau) d\tau \\ &= \int p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) r(\tau) d\tau \\ &= \mathbb{E}_{\tau \sim p_\theta(\tau)}[\nabla_\theta \log p_\theta(\tau) r(\tau)] \end{aligned} \tag{4.16}$$

Ya que $\nabla_\theta p_\theta(\tau) = \frac{\nabla_\theta p_\theta(\tau)}{p_\theta(\tau)} p_\theta(\tau) = \nabla_\theta \log p_\theta(\tau) p_\theta(\tau)$

Es posible notar que

$$\log p_\theta(\tau) = \log(s_0) + \sum_{t=0}^{N} \log \pi_\theta(a_t|s_t) + \log p(s_{t+1}|s_t.a_t)$$

Luego $\nabla_\theta \log p_\theta(\tau) = \sum_{t=0}^{N} \nabla_\theta \log \pi_\theta(a_t|s_t)$

Finalmente

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}[\sum_{t=0}^{N} \nabla_\theta \log \pi_\theta(a_t|s_t)(\sum_{t=0}^{N} r(s_t, a_t))] \tag{4.17}$$

Por Monte Carlo, si se simulan B trayectorias τ, luego

$$\nabla_\theta J(\theta) \approx \frac{1}{B}\sum_{i=1}^{B}\sum_{t=0}^{N}\nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t})(\sum_{t'=0}^{N} r(s_{i,t'}, a_{i,t'})) \tag{4.18}$$

Esta formulación es conocida como el teorema de *Policy Gradients*, usados en la clase de algoritmos REINFORCE [45].

Más aun, es posible notar que como la política a tiempo t' no afecta las recompensas que se obtienen a tiempo $t < t'$, la formulación 4.17 da a lugar al siguiente teorema

Teorema 4.11 ***Policy Gradient I***

Para cualquier PDM, el gradiente de $J(\theta)$ está dado por

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim p_\theta(\tau)}[\sum_{t=0}^{N}\nabla_\theta \log \pi_\theta(a_t|s_t)(\sum_{t'=t}^{N} r(s_{t'}, a_{t'}))] \tag{4.19}$$

Para mayor información ver [35]

4.4.3. Algoritmo Actor-Critic

Un problema que tiene la formulación anterior, tal como la tienen la mayoría de las aproximaciones por Monte Carlo, es que la varianza va a ser muy grande cuando el número de trayectorias simuladas no es suficientemente grande. Esto va a pasar necesariamente si es que se quiere construir un algoritmo que sea eficiente en tiempo y recursos.

Una forma de reducir la varianza es introducir una función basal $b : S \to \mathbb{R}$, tal que se cumpla

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim p_\theta(\tau)}[\sum_{t=0}^{N}\nabla_\theta \log \pi_\theta(a_t|s_t)(\sum_{t'=t}^{N} r(s_{t'}, a_{t'}) - b(s_t))] \tag{4.20}$$

Para ver esto tal vez la forma más simple es introduciendo una versión equivalente del teorema *Policy Gradient*, que es válida para espacios de estados-acciones tanto discretos como continuos [9].

Teorema 4.12 ***Policy Gradient II***

Para cualquier PDM, el gradiente de $J(\theta)$ está dado por

$$\nabla_\theta J(\theta) = \int_{s\in S} d^\pi(s)\int_{a\in A}\nabla_\theta \pi_\theta(a|s)Q_\pi(s,a)dads \tag{4.21}$$

Con $d^\pi(s) = \sum_{t=0}^{\infty}\mathbb{P}_\pi(s_t = s|s_0)$

Esta última formulación es equivalente a lo siguiente:

Teorema 4.13 ***Policy Gradient con función basal***

Para cualquier PDM y una función basal $b : S \to \mathbb{R}$ *el gradiente de* $J(\theta)$ *está dado por*

$$\nabla_\theta J(\theta) = \int_{s\in S} d^\pi(s) \int_{a\in A} \nabla_\theta \pi_\theta(a|s)(Q_\pi(s,a) - b(s))dads \tag{4.22}$$

Con $d^\pi(s) = \sum_{t=0}^{\infty} \mathbb{P}_\pi(s_t = s|s_0)$

Lo cual se cumple ya que $\int_{a\in A} \nabla_\theta \pi_\theta(a|s) = 0$.

Luego se puede demostrar que cuando se toman estados con respecto a $d^\pi(s)$ y acciones con respecto a $\pi(a|s)$, en cada tiempo t se tiene que

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi[\nabla_\theta \log \pi_\theta(a_t|s_t)(Q_\pi(s_t,a_t) - b(s_t))] \tag{4.23}$$

La alternativa comúnmente usada para la función basal es una aproximación de la función valor para cada estado $s \in S$, es decir $b(s_t) \approx V_\pi(s_t)$. Por otro lado es claro que $R_t = \sum_{t'=t}^{N} r(s_{t'}, a_{t'})$ es un estimador de $Q_\pi(s_t, a_t)$. En esto surge el algoritmo Actor-Critic [23] el cual tiene la siguientes características:

- La política $\pi_\theta(a|s)$ parametrizada por $\theta \in \mathbb{R}^d$ va a ser la función principal a estimar. Esta función en este contexto se le va a llamar "Actor", ya que es aquella que va a dar a lugar las trayectorias para el entrenamiento.
- Se utiliza una función basal que va a estimar la función valor $V_\pi(s)$. Lo que se realiza en aprendizaje reforzado profundo es estimarla mediante una red neuronal $V_\phi(s)$ de parámetro $\phi \in \mathbb{R}^d$. Esta red neuronal en la literatura es conocida como "Critic".
- Tal como se dijo, el estimador para Q_π van a ser los retornos R_t, tal como se mostró en el teorema Policy Gradient I.
- De esta forma se estaría estimando completamente la función de ventajas $A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s)$

La forma de entrenar la red neuronal "Critic" es mediante mínimos cuadrados, tal como se hace habitualmente en una regresión. Es decir, para B trayectorias del estilo $(s_0, a_0, ..., s_N, a_N)$, se minimiza la siguiente función de pérdidas

$$\mathcal{L}_v(V;\phi) = \frac{1}{2}\sum_{i=1}^{B}\sum_{t=1}^{N} || \sum_{t'=t}^{N} r(s_{i,t'}, a_{i,t'}) - V_\phi(s_{i,t})||^2 \tag{4.24}$$

Con todas estas componentes se presenta completamente el algoritmo Actor-Critic

Algorithm 3: Algoritmo Actor-Critic

Input: B: tamaño del batch

1 Inicializar θ // Parámetro Actor
2 Inicializar ϕ // Parámetro Critic
3 **while** *No ha convergido* **do**
4 Inicializar $s_{i,0}$ para $i = 1, ..., B$
5 **for** *t=1,...,N* **do**
6 $a_{i,t} \sim$ `SampleSolution`$(\pi_\theta(\cdot|s_{i,t}))$ for $i = 1, ..., B$
7 Guardar $r_{i,t}$ y actualizar $s_{i,t} \leftarrow s_{i,t+1}$
8 $y_{i,t} \leftarrow \sum_{t'=t}^{N} r(s_{i,t'}, a_{i,t'})$ para cada $t = 0, ..., N$
9 $\mathcal{L}_v \leftarrow \frac{1}{2} \sum_{i,t} ||\hat{V}_\phi(s_{i,t}) - y_{i,t}||^2$ for $i = 1, ..., B$, $t = 0, ..., N$
10 $\phi \leftarrow$ Optimizer$(\phi, \nabla \mathcal{L}_v)$
11 $\hat{A}(s_{i,t}, a_{i,t}) \leftarrow \sum_{t'=t}^{N} r(s_{i,t'}, a_{i,t'}) - \hat{V}_\phi(s_{i,t})$ para $i = 1, ..., B$, $t = 0, ..., N$
12 $\nabla_\theta J \leftarrow \frac{1}{B} \sum_{i=1}^{B} \sum_{t=1}^{N} \nabla_\theta \log \pi_\theta(a_{i,t}, s_{i,t}) \hat{A}(s_{i,t}, a_{i,t})$
13 $\theta \leftarrow$ Optimizer$(\theta, \nabla_\theta J)$

Cabe notar que en la línea 6 `SampleSolution` significa que hay que simular la distribución de probabilidad correspondiente a la política π_θ y obtener una acción correspondiente a esa simulación.

Los optimizadores en las líneas 10 y 13 corresponde a realizar descenso de gradiente estocástico para la actualización de los parámetros ϕ y θ.

4.4.4. Sub-acciones en cada época de decisión

Todavía no se ha dicho cómo se construyen las rutas como acciones en cada época decisión. La idea es representar un plan de rutas mediante la factorización de una secuencia de subacciones. Para esto se define la l-ésima sub acción a tiempo t como a_t^l que va a representar una asignación de un nodo a un vehículo. Es decir, para el grafo $G_t = (V_t, A_t)$, cada subacción va a ser un par nodo-vehículo $(i, k) \in V_t \times \{1, ..., m\}$. De esta forma, si se ven estas selecciones de sub-acciones de forma secuencial, se puede representar la probabilidad del plan de rutas a_t obtenido por todas estas asignaciones como

$$\pi_\theta(a_t|s_t) = \prod_{l=0}^{L} \pi_\theta(a_t^l|a_t^{<l}, s_t),$$

donde L, es el total de subacciones ejecutadas en la época de decisión t, y $a_t^{<l}$ es el conjunto de nodos que han sido visitados. Esto se hizo con inspiración del trabajo realizado para resolver el problema del vendedor viajero mediante aprendizaje reforzado [4] y de otra aplicación que usaba también sub-acciones secuenciales en cada época de decisión [43].

Cabe notar que estos pares nodo-vehículo que se seleccionen deben ser tomados de un conjunto factible de posibilidades. Las restricciones que se imponen son no seleccionar nodos que ya fueron incluidos en una ruta, seleccionar nodos cuyas demandas no superen a la

capacidad del vehículo a que se está asignando y seleccionar nodos que respeten la ventana de tiempo de la bodega. Es por esto que se define el conjunto de pares nodo-vehículo factibles (como subacciones)

$$F(s_t, a_t^{<l}) = \{(i,k) \in V_t \times \{1,...,m\} |\ q_i \leq Q_k^{t,l-1},\ \ \text{máx}\{e_i, IT_k^{t,l-1} + t_{i_k,i}\} + st + t_{i,0} \leq l_0, i \notin a^{<l}\}$$

Donde $Q_k^{t,l}$ y $IT_k^{t,l}$ es la capacidad y la hora de disponibilidad (respect.) que lleva el vehículo k después de tomar la subacción l. Además $i_k \in \mathcal{N}_t$ se denota el último cliente visitado por el vehículo $k \in \{1,...,m\}$.

Cada una de estas variables se actualiza de la misma forma que siempre. Si $a_t^{l-1} = (i,k)$ entonces

$$Q_k^{t,l} = Q_k^{t,(l-1)} - q_i, \qquad Q_{k'}^{t,l} = Q_{k'}^{t,(l-1)} \quad \forall k' \neq k$$
$$IT_k^{t,l} = \text{máx}\{e_i, IT_k^{t,(l-1)} + t_{i_k,i}\} + st, \qquad IT_{k'}^{t,l} = IT_{k'}^{t,(l-1)} \quad \forall k' \neq k$$

Con $Q_k^{t,0} = Q_k^{t-1}$ y $IT_k^{t,0} = IT_k^{l-1}$. Y finalmente $Q_k^{t+1} = Q_k^{t,L}$ y $IT_k^{t+1} = \text{máx}\{IT_k^{t,L}, h_{t+1}\}$, para cada $k \in \{1,...,m\}$, donde h_{t+1} es la hora que empieza la época de decisión $t+1$.

4.5. Un algoritmo que incorpora información estocástica

Es relevante recordar que el problema de ruteo de vehículos dinámico puede dividirse en 2 clases de subproblemas dependiendo de la calidad de la información que se tenga del ambiente. En este caso, se centrará en la categoría que tiene conocimiento estocástico de los datos. Específicamente se restringirá a conocimiento de información estocástica espacial, es decir, se tendrá acceso a una distribución de probabilidad para las posiciones de los clientes. Por otro lado, se tendrá acceso a la distribución de la longitud de las ventanas de tiempo. Incorporar otro tipo de información, tal como el horario de las solicitudes de clientes, quedaría como trabajo futuro. Como en este algoritmo se tiene acceso a este tipo de información, es parcialmente un algoritmo catalogado como *model based*.

Para abordar este problema se va a proponer una estrategia de simulaciones tal como se realizó en [5] con "Multiple Scenario Approach" (MSA). En este trabajo, en cada período de decisión se simulaban posibles clientes que podrían realizar una solicitud en el futuro, esto tomando en cuenta una distribución de probabilidad conocida. Con esto resolvían un problema estático sobre la suma de los clientes conocidos y simulados y se planificaban rutas a partir de esto.

En esta tesis se propone realizar algo parecido a MSA pero adaptado al ámbito del aprendizaje reforzado. Se supone que se tiene un proceso de decisión de Markov tal como el del algoritmo Q-learning, es decir, las acciones se toman como asignaciones pares nodo-vehículo factibles o la acción pasar en estados definidos sobre el estado del grafo de los clientes y sobre los vehículos disponibles.

Se supone que se tiene una política a priori π que más adelante se explicará de dónde se obtiene. Se supone además que se está tomando una decisión en el estado $s \in S$. La idea es realizar diversas exploraciones que guíen al agente a tomar la mejor decisión posible. Para

cada búsqueda, se simularán posibles clientes que pueden aparecer en el futuro, usando la distribución de probabilidad conocida. Para empezar la exploración, en primer lugar se toma una primera acción inicial a^*. Esta acción no va a ser la que efectivamente va a tomar el agente, sino que va a guiar al agente a realizar la exploración. Sin embargo, la toma de esta acción va a influir en la acción definitiva a tomar por el agente.

Para ver cuál primera posible acción tomar, se define la fórmula PUCT (Polynomial Upper Confidence Trees) usada en general en Árboles de decisión de Markov [33]:

Definición 4.14

$$PUCT(s,a) = Q(s,a) + c_{PUCT}\pi(s,a)\frac{\sqrt{\sum_b n(s,b)}}{1+n(s,a)}, \tag{4.25}$$

donde

- $Q(s,a)$*: es la recompensa esperada al tomar la acción* a *en el estado* s*, es decir, la recompensa media que se obtiene al tomar la acción* a *en todas las simulaciones realizadas.*
- $n(s,a)$*: es el número de veces que se tomó la acción* a *en las últimas simulaciones realizadas.*
- c_{PUCT}*: es un hiperparámetro que controla el grado de exploración*

Cabe notar que el hiperparámetro c_{PUCT} si se incrementa se le da más énfasis al término de la derecha en la fórmula anterior, es decir, se aumenta la exploración. Por otro lado, si se decrece, se aprovechan de mejor manera los valores esperados.

En conclusión

$$a^* = \arg\max_a PUCT(s,a)$$

Para continuar la exploración se simula un conjunto de clientes de acuerdo a la distribución de probabilidad conocida. La idea es resolver un problema estático sobre la suma de los clientes conocidos y simulados usando también aprendizaje reforzado. La acción a^* va a generar la primer acción en caso de que sea un par nodo-vehículo sobre este nuevo problema. En caso de ser la acción "pasar" de período de decisión, simulando que ocurre esto, significa que todos los tiempos de los vehículos se fijan como si estuvieran en el siguiente período de decisión.

Ahora para continuar formando las rutas de este problema estático, la idea es encontrar una política π^{sim} que va a estar enfocada en encontrar rutas óptimas para el problema estático. Antes de todo se va explicitar el PDM involucrado para este sub-problema:

4.5.1. Proceso de decisión de Markov para el sub problema estático

En este caso el PDM es mucho más sencillo que en el caso dinámico. La idea es modelarlo tal cual se hizo para el problema del vendedor viajero en [4], pero adaptado para el problema de ruteo de vehículos (estático).

- **Épocas de decisión**: En este caso va a ser sólo una época de decisión
- **Estados**: Corresponderán tanto los estados de los clientes conocidos y simulados como los estados de los vehículos.

  En este sentido se define el conjunto de estados S_t^1 que corresponde a las posiciones, demandas y ventanas de tiempo de los clientes. Esto se puede modelar a través de un grafo completo $G = (V, E)$, donde el conjunto de nodos V representa la bodega, los clientes conocidos y los clientes simulados. Luego, para para $n = |V|$, se tiene que el espacio de estados S_t^1 es

  $$S_t^1 = \mathbb{R}^{2 \cdot n} \times \mathbb{R}_{>0}^n \times [e_0, l_0]^{2 \cdot n}$$

  El estado S_t^2 correspondiente al estado de los m vehículos al inicio de la creación de rutas. Cabe notar que inicialmente no necesariamente van a estar todos los vehículos en la bodega, ya que a medida que va a ir avanzando la exploración, en el algoritmo principal se van a ir tomando decisiones que van a cambiar las posiciones de los vehículos. En este sentido para cada vehículo se guarda como estado la posición de este, la hora en que se va a encontrar disponible y la capacidad actual. Luego

  $$S_t^2 = \mathbb{R}^{2 \cdot m} \times [e_0, l_0]^{2 \cdot m} \times \mathbb{R}_{>0}^m$$

  Finalmente $S' = \cup_{t \in T} S_t^1 \times S_t^2$ es el conjunto de estados. De ahora en adelante se denotará como $s' \in S'$ un estado correspondiente al proceso de decisión de Markov del subproblema estático.
- **Acciones**: Cada acción corresponderá a un plan de rutas factibles $a = (R_1, ..., R_m)$, tal que todos los nodos del grafo constituido por los clientes estén en alguna ruta. Este plan de rutas de construirá mediante una estrategia de subacciones, al igual que como se hizo para el algoritmo Actor-Critic para el DVRP.

  Es decir, cada subacción va a ser un par nodo-vehículo $(i, k) \in V \times \{1, ..., m\}$. De esta forma, si se ven estas selecciones de sub-acciones de forma secuencial, se puede representar la probabilidad de un plan de rutas a como

  $$\pi_\theta(a|s') = \prod_{l=0}^{n} \pi_\theta(a^l | a^{<l}, s')$$

  Con a^l una subacción par nodo-vehículo. Cada subacción debe ser tomada restringiendo no seleccionar nodos que ya fueron incluidos en alguna ruta y respetando la capacidad de los vehículo y la ventana de tiempo de la bodega.
- **Recompensa**: La recompensa de un plan de rutas va a estar dada por el costo de la distancia recorrida total de las rutas (que va a incluir el retorno a las bodegas), la suma de las penalizaciones en caso de que no se respete la ventana de tiempo del cliente y una penalización por cada cliente no visitado

  $$r(s', a) = -C(s', a) - \alpha L(s', a) - \beta NV(s', a) \tag{4.26}$$

- **Función de transiciones**: En este caso no tiene sentido hablar de función de transiciones ya que sólo hay una época de decisión.

4.5.2. Descripción del algoritmo

De ahora en adelante se va a denotar por ρ la acción correspondiente al plan de rutas generado por el problema estático. A partir de ρ, se obtiene la recompensa obtenida r_s con la cual se actualiza $Q(s, a)$.

$$Q(s, a^*) \leftarrow \frac{n(s, a^*) \cdot Q(s, a^*) + r_s}{n(s, a^*) + 1}$$

Por otro lado también se actualiza $n(s, a)$

$$n(s, a^*) \leftarrow n(s, a^*) + 1$$

Para poder tomar una decisión en base a esta solución del problema estático, lo que se hace es construir una posible distribución de probabilidad sobre las acciones definidas por los pares nodo-vehículo factibles y la acción de pasar a la siguiente época de decisión. Una idea hubiera sido usar la misma función $n(s, a)$, sin embargo se tomó otra opción. Una buena idea es fijarse en todos los primeros clientes que visitan cada uno de los vehículos, a partir de las rutas obtenidas de las simulaciones.

Se define la función $N(s, a) : S \times A_s \to \mathbb{N}$, que inicialmente (antes de realizar las simulaciones) va a estar definida por $N(s, a) = 0 \quad \forall a \in A_s$, donde A_s es el conjunto de acciones factibles en el estado s. En cada simulación este vector se va a actualizar de la siguiente manera, para cada vehículo $k \in \{1, ..., m\}$

- Caso en que el primer cliente i del vehículo k es conocido, se actualiza $N(s, a) \leftarrow N(s, a) + 1$, con $a = (i, k)$
- Caso en que el primer cliente del vehículo k es simulado, no se hace nada.
- Caso en que todos los primeros clientes son simulados, se actualiza $N(s, a) \leftarrow N(s, a) + 1$, para $a = pass$.

Esto quiere decir que sólo se actualizan las acciones correspondientes a primeros clientes de vehículos que no son simulados. Por otro lado tiene sentido pasar a la siguiente época de decisión en el caso que todos los primeros clientes que se obtienen son simulados (ya que eventualmente aparecerían en un futuro y no conviene tomar una acción con un cliente conocido).

Lo que se está tratando de hacer aquí es contar todas aquellas acciones que han resultado como primer cliente de cada vehículo, lo que al final de cuentas daría las acciones más probables de tomar en la decisión real que se tiene que tomar.

Finalmente la acción definitiva que se toma se obtiene simulando

$$p(s, a) = \frac{N(s, a)^{1/\tau}}{\sum_b N(s, b)^{1/\tau}}, \tag{4.27}$$

donde τ es la temperatura que controla el grado de exploración. La idea de este término es que hace que hace más uniforme la distribución de probabilidad a medida que aumenta la temperatura. Por el contrario si $\tau \to 0$ significa que la acción que tiene mayor contador va a tener probabilidad 1.

4.5.3. Entrenamiento

Hasta ahora no se ha dicho cómo se obtiene la política a priori π que se definió anteriormente. Una buena alternativa que se propone es que π se parezca lo más posible a la política $p(s, a)$, ya que esta última da un buen indicio de lo debería ser la política óptima a seguir. Para esto se propone imitarla a través de una política parametrizada π_θ. Se quiere aplicar el mismo razonamiento que se usa cuando se entrena un clasificador. La idea es aprender una distribución de probabilidad imitando una "real", minimizando la entropía cruzada entre ambas distribuciones. Aplicando esto al problema, la función de pérdidas es

$$\mathcal{L}(\pi, p; \theta) = -p^T \log(\pi_\theta) \tag{4.28}$$

Para el caso de π^{sim} se parametriza con un parámetro θ' el cual debería ser obtenido del siguiente problema de optimización:

$$\min_{\theta'} \mathbb{E}_{\rho \sim \pi^{sim}(\cdot|s')} r(s', \rho)$$

La idea es resolver esto usando un enfoque Actor-Critic [4, 23]. Es posible notar que gradiente de esta función objetivo es obtenido gracias al teorema *Policy Gradient*:

$$\nabla_{\theta'} \mathcal{L}(\pi^{sim}; \theta') = \mathbb{E}_{\rho \sim \pi^{sim}(\cdot|s')}(r(s', \rho) - V_\phi(s')) \nabla_{\theta'} \log \pi^{sim}_{\theta'}(\rho|s')$$

Donde $V_\phi(s')$ es una función que va a estimar la recompensa esperada para el estado s', la cual se aproxima minimizando el error cuadrático medio entre las recompensas obtenidas y la estimación. Al final de cada búsqueda (es decir luego de haber realizado todas las simulaciones) se utilizan todas las simulaciones realizadas para estimar las esperanzas involucradas. Sea B el número total de ejemplos simulados, luego se optimiza $\pi^{sim}_{\theta'}$ y $V_\phi(s')$ con lo siguiente

$$\nabla_{\theta'} \mathcal{L}(\pi^{sim}; \theta') = \frac{1}{B} \sum_{i=1}^{B} (r(s'_i, \rho_i) - V_\phi(s'_i)) \nabla_{\theta'} \log \pi^{sim}_{\theta'}(\rho_i|s'_i) \tag{4.29}$$

$$\mathcal{L}_v(V; \phi) = \frac{1}{B} \sum_{i=1}^{B} ||r(s'_i, \rho_i) - V_\phi(s'_i)||^2 \tag{4.30}$$

Se propone el siguiente algoritmo que combina el entrenamiento de π_θ y π^{sim}

Algorithm 4: Algoritmo principal

Input: m: número de vehículos, M: batch size, N: tamaño de la memoria

Inicializar memoria $\mathcal{D}$ con capacidad N
Inicializar políticas π_θ, $\pi^{sim}_{\theta'}$ y función valor V_ϕ con pesos arbitrarios
while *No ha convergido* **do**
    Inicializar s_0
    for t=0,...,n **do**
        $a_t, p_t, \pi^{sim}_{\theta'}, V_\phi \leftarrow$ `Búsqueda`$(s_t, \pi_\theta, \pi^{sim}_{\theta'}, V_\phi)$
        Almacenar la secuencia (s_t, p_t) en $\mathcal{D}$
        Ejectutar a_t y observar s_{t+1} y r_t
        Tomar un minibatch de tamaño M con experiencias (s_j, p_j) de $\mathcal{D}$ de manera uniforme
        $\mathcal{L}(\pi, p) \leftarrow -p^T log(\pi_\theta(s))$
        $\theta \leftarrow$ Optimizador$(\theta, \nabla_\theta \mathcal{L})$

Algorithm 5: Algoritmo de búsqueda

Input: N_{sim}: número de simulaciones, s: estado actual, $\pi_\theta, \pi^{sim}_{\theta'}$: políticas, V_ϕ: función valor

for $i = 1, ..., N_{sim}$ **do**
    Calcular $PUCT(s, a) \quad \forall a \in A_s$
    $a^* \leftarrow \arg\max_a PUCT(s, a)$
    Actualizar $n(s, a^*)$, $Q(s, a^*)$
    $s' \leftarrow$ `Simulación`(s)
    Obtener plan de rutas ρ usando $\pi^{sim}_{\theta'}(\cdot|s')$, fijando como primera acción a^*
    Actualizar política $\pi^{sim}_{\theta'}$ y función valor V_ϕ usando el enfoque Actor-Critic con ecuaciones 4.29 y 4.30
    Actualizar $N(s, a)$ de acuerdo al plan de rutas ρ, para cada $a \in A_s$
Calcular $p(s, a) = \frac{N(s,a)^{1/\tau}}{\sum_b N(s,b)^{1/\tau}}$ y simular $a \sim p(s, \cdot)$
Return $a, p, \pi^{sim}_{\theta'}, V_\phi$

donde la función `Búsqueda` se refiere a usar el algoritmo de búsqueda y `Simulación` se refiere a obtener posibles clientes que podrían conocerse en algún futuro, de acuerdo a la distribución de probabilidad conocida. Esta función retorna un nuevo estado s' que junta los clientes conocidos con los simulados.

4.5.4. Ejemplo

Se supone que se tiene un input con 3 clientes y 2 vehículos que están estacionados en la bodega. Si se supone que todas las asignaciones son factibles, se tienen 7 acciones posibles: 6 por cada par cliente-vehículo y la acción de pasar a la siguiente época de decisión. Para obtener la distribución de probabilidad de estas acciones se realiza la búsqueda con simulaciones. Antes de cada simulación se calcula la fórmula $PUCT(s, a)$ para cada acción y se selecciona

la que tiene mayor valor. De esto, por ejemplo la acción obtenida puede ser asignar el cliente 2 al vehículo 1. Se supone que de la distribución de probabilidad conocida se simularon 3 clientes (los cuales van a estar codificados como 4,5,6). Se resuelve el problema estático sobre los clientes conocidos y simulados usando π^{sim}, fijando desde un principio la acción obtenida de $\arg\max_a PUCT(s, a)$. Considerando esto, por ejemplo se pudo haber obtenido el siguiente plan de rutas (observar que el cliente 2 efectivamente está primero en la ruta del vehículo 1):

- Vehículo 1 : 2 - 4 - 5 - 1
- Vehículo 2 : 6 - 3

Esto dice que se tiene que actualizar $N(s, (2, 1))$ ya que se asignó como primero en la ruta del vehículo 1 el cliente 2 (y que es efectivamente la acción que inicialmente se tomó con la formula PUCT). En cambio el vehículo 2 tiene al cliente 6 como primer cliente, el cual fue simulado, por lo que no se actualiza nada para este vehículo. Por otro lado de este plan de rutas se obtiene una recompensa que se usa para actualizar $Q(s, (2, 1))$, lo que va a influir en la siguiente selección.

La idea es hacer esto por cada una de las simulaciones para finalmente obtener la distribución $p(s, a)$ que entregaría la acción real a ejecutar.

Capítulo 5

Arquitectura de la red neuronal

En este capítulo se entrega una descripción de alto nivel de las redes neuronales profundas que modelan las diversas funciones parametrizadas de los algoritmos descritos en el capítulo anterior. Las descripción con mayor detalle se encuentra en los anexos.

Como se explicó en el capítulo 4, las redes neuronales son funciones no lineales que permiten modelar funciones complejas y de alta dimensionalidad.

Para los algoritmos de aprendizaje reforzado que se explicitaron es necesario detallar cómo es la construcción que hay detrás de cada función parametrizada.

En el caso de los algoritmos Q-learning es necesario modelar la función acción valor $Q_\theta(s, a)$. Por otro lado los algoritmos Actor-Critic y el algoritmo que incorpora información estocástica requieren modelar directamente la política estocástica $\pi_\theta(a|s)$ y la función valor $V_\phi(s)$.

Antes que todo se definirán ciertas arquitecturas de redes neuronales que serán la base para construir los objetos necesarios. Con esto y considerando las entradas y salidas adecuadas va a ser sencillo modelar las funciones parametrizadas de los algoritmos.

5.1. Red de codificación

Sea $G = (V, A)$ grafo dirigido, con $|V| = n$, instancia del problema de vehículos dinámico. Cada nodo del grafo posee características $s_i^1 = (x_i, y_i, q_i, e_i, l_i, ct_i) \in \mathbb{R}^6$, $i \in \{1, ...n\}$, donde x_i es la posición en el eje x, y_i la posición en el eje y, q_i la demanda, e_i inicio de la ventana de tiempo, l_i fin de la ventana de tiempo y ct_i la hora de solicitud. Por otro lado, para cada arco $(i, j) \in A$ se tiene el tiempo de viaje $t_{ij} \in \mathbb{R}$ entre el nodo i y el nodo j. Se va a denotar por T la matriz correspondiente a los tiempos de viaje. Para la red codificadora se utiliza la misma red neuronal utilizada en [10], pero adaptada para incluir en la entrada los tiempos de viaje. Cabe notar que la red neuronal debe ser capaz de tratar con grafos de tamaño variable, es decir, n puede ser variable. Esto se consigue con redes neuronales llamadas recurrentes.

Se considera $s^1 = [s_1^1, ..., s_n^1]$ la concatenación de los n elementos. Luego, para $s^1 \in \mathbb{R}^{n\times 6}$,

se define la red de codificación $enc_\theta(s^1)$ que retorna una salida $h \in \mathbb{R}^{n \times d_h}$.

Esta red de codificación se compone primero de una primera capa de "embedding" que hace cambiar la dimensionalidad de la entrada a través de una transformación lineal y aplica una normalización a los datos. La salida de esta capa es usada como entrada para una red neuronal del tipo atención, similar a la red Transformer [42] usada en contextos de procesamiento de lenguaje natural. La intuición de esta red es que permite calcular mediante puntajes, el nivel de relación entre los n elementos de la entrada, generando una distribución de probabilidad sobre los niveles de atención entre cada par de elementos. A estos puntajes se le añade también el efecto de los tiempos de viaje mediante la matriz T. Por otro lado su arquitectura es altamente paralelizable, lo que aumenta el rendimiento en su entrenamiento. Finalmente, esta red retorna como salida el vector latente $h \in \mathbb{R}^{n \times d_h}$ que ya va a tener contenida la información de las relaciones entre todos los elementos.

5.2. Red de decodificación

La red de decodificación usa como entrada el resultado de la red de codificación $h \in \mathbb{R}^{n \times d_h}$ y un vector de contexto. El vector de contexto es calculado para cada vehículo usando un resumen del grafo (que va a corresponder al promedio de las salidas de la red de codificación), la codificación del último nodo visitado, la codificación de la bodega, la capacidad y la hora de disponibilidad.

Usando la codificación de los nodos, se genera un preprocesamiento sobre los vectores de contexto usando los mismos mecanismos de atención que para la red codificadora, para incorporar las relaciones entre los nodos y los vehículos disponibles.

La salida de esta red va a depender del algoritmo. En caso del algoritmo DQN (y sus variaciones) es necesario retornar un vector a valores reales del tamaño de las acciones (en este caso es $n \cdot m + 1$). Para ello se realiza una especie de regresión usando la codificación de los nodos y el preprocesamiento de los vectores de contexto.

Por otro lado en el algoritmo actor-critic y algoritmo que utiliza información estocástica es necesario retornar una distribución de probabilidad en el espacio de las acciones. En este caso sería un vector del tamaño de las acciones a valores en $[0, 1]$. En esta red se utilizan los mismos mecanismos de atención, pero la salida es normalizada mediante una operación softmax que va a representar la distribución de probabilidad requerida.

5.3. Red critic

En el algoritmo actor-critic es necesario aproximar la función valor $V_\phi(s)$. Para ello, utilizando la codificación de los nodos, las capacidades y los tiempos de de disponibilidad de los vehículos se realiza una regresión. Dentro de ella también se incluye un mecanismo de atención que calcula una atención ponderada de cada nodo y vehículo de la entrada, con tal de asignar valores de importancia a cada uno de ellos.

5.4. Diagrama de alto nivel

En las figuras 5.1 y 5.2 se muestran los diagramas de como se conectan las distintas capas que fueron descritas anteriormente. Como se puede observar la red codificadora es compartida por ambos algoritmos, pero la decodificación es realizada de manera diferente por la naturaleza de los algoritmos.

En el caso del algoritmo que utiliza información estocástica, la arquitectura es igual a la del algoritmo actor-critic cuando se resuelve el problema estático. Sin embargo para la política que se quiere imitar, sólo se utiliza el proceso para formular la política del diagrama actor-critic, es decir, no se utilizan las capas que permiten retornar el valor del estado.

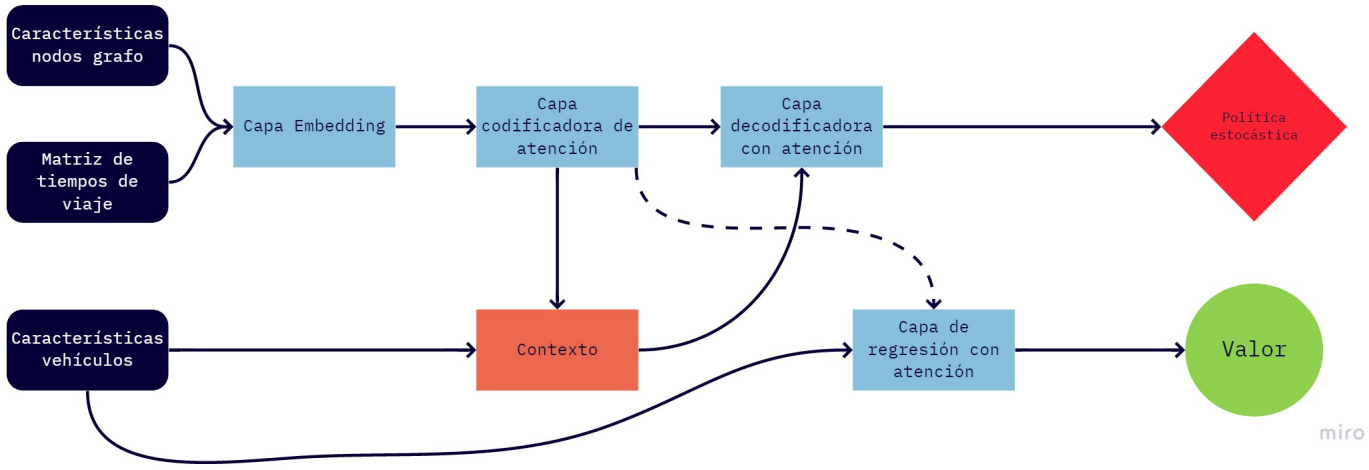

Figura 5.1: Diagrama de flujo red neuronal algoritmo Actor-Critic

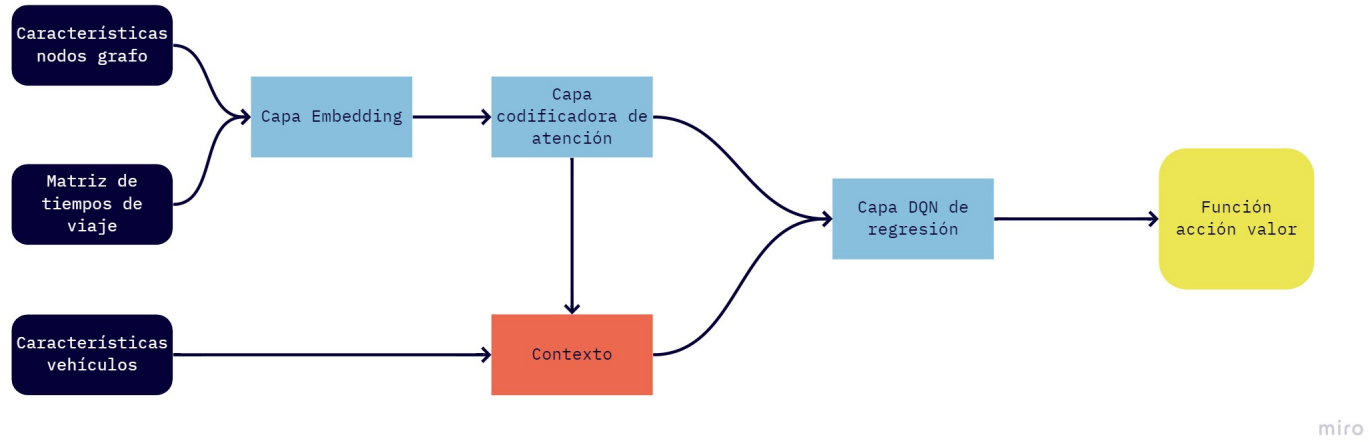

Figura 5.2: Diagrama de flujo red neuronal algoritmo Q-learning

Capítulo 6

Resultados computacionales

En este capítulo se muestran los resultados correspondientes a los algoritmos Deep Q Learning (con la variante DDQN y priorización de experiencias), Actor-Critic y algoritmo estocástico con búsqueda. Primero se evidencia cómo se generó el conjunto de datos con que se entrenaron los modelos, luego se muestra la configuración y los hiperparámetros usados en los entrenamientos. A continuación se muestran resultados numéricos y visualizaciones de los resultados. Para terminar se muestran comparaciones con un algoritmo de ruteo de vehículos dinámico que usa reoptimización y con las soluciones óptimas del problema estático.

6.1. Generación de datos y entrenamientos

Es preciso recordar que esta clase de algoritmos necesitan de un conjunto de datos de entrenamiento. Este conjunto tiene que estar completamente relacionado con el conjunto de estados definido en cada uno de los procesos de decisión de Markov. Este conjunto de datos consiste en básicamente un conjunto de datos relativo a los clientes y un conjunto de datos relativo a la flota de vehículos. En cuanto a los tiempos de viaje estos son calculados con la distancia respectiva (ya sea euclideana o Manhattan) y usando una velocidad constante.

Los clientes realizan sus solicitudes en un horizonte de decisión de intervalo $[e_0, l_0] = [0, 100]$. Los clientes conocidos realizan sus solicitudes en la hora $t = 0$. El número de clientes conocidos se simula de acuerdo a una variable aleatoria Poisson de parámetro $n_{exp} \cdot (1 - \delta)$, donde n_{exp} es el número esperado de clientes totales que realizan solicitudes dentro del horizonte de decisión y δ es el grado de dinamismo. En este caso el grado de dinamismo se fija en 0,5, es decir, en promedio, la mitad de los clientes van a ser conocidos y la otra mitad no. El número esperado de clientes va a estar fijado en 50. En cuanto a los tiempos de solicitudes de los clientes dinámicos no conocidos estos van a estar distribuidos de acuerdo a una variable aleatoria exponencial de parámetro $\frac{n_{exp} \cdot \delta}{t_{max} - e_0}$, donde $t_{max} \in [e_0, l_0]$ es el tiempo máximo de aceptación de solicitudes dinámicas. En este caso se fija $t_{max} = 50$, es decir, está permitido recibir solicitudes dinámicas hasta mitad del horizonte de decisión.

Las características de los clientes también son simuladas de manera aleatoria. Las posiciones de clientes son simuladas en el conjunto $[0, 1]^2$ en 2 grupos

- Uniforme en $[0,1]^2$
- Agrupaciones gaussianas aleatorias: se simulan cuatro puntos p_1, p_2, p_3 y p_4 de manera uniforme entre los números $\{0{,}2; 0{,}3; 0{,}4; 0{,}5; 0{,}6; 0{,}7; 0{,}8\}$. Luego las posiciones de los clientes se simulan de acuerdo a una normal $\mathcal{N}((p_1, p_2); \Sigma^2)$ y $\mathcal{N}((p_3, p_4); \Sigma^2)$ con probabilidad 0.5 cada una. Se toma $\Sigma = \begin{pmatrix} 4 & 0 \\ 0 & 4 \end{pmatrix}$.

Las demandas de los clientes son simuladas según una uniforme $[0, 1]$. En cuanto a las ventanas de tiempo, la apertura del cliente i se simula según una uniforme en $[ct_i, l_0 - st - 1/v]$, donde ct_i es el tiempo de solicitud con que fue simulado el cliente i, st es el tiempo de servicio y v es la velocidad del medio. Para calcular el cierre de la ventana de tiempo, se simula la longitud de la ventana de tiempo según una normal $\mathcal{N}(tw_m, tw_s^2)$ donde tw_w y tw_s son hiperparámetros. Para los entrenamientos se fija $tw_w = 20$ y $tw_s = 2{,}5$.

En cuanto a los vehículos, se fija el número de vehículos en $m = 14$ con capacidades iguales de $Q = 4$. Con esto se asegura que en la mayoría de los casos, todos los clientes se puedan visitar.

Los tiempos de servicio de todos los clientes se fijan en $st = 4$. La velocidad se fija en $v = 0{,}1$ en el caso de distancia euclideana y $v = 0{,}25$ en el caso de distancia Manhattan. Con estos números experimentalmente se evitan que hayan retrasos excesivos. Además el número de periodos de decisión se fija en $P = 15$.

Previamente a entregar estos datos a los algoritmos, se hace un preprocesamiento donde se normalizan los datos. Las capacidades de los vehículos se dejan en $Q' = 1$ y las demandas de los clientes en $q'_i = q_i/Q$. Las ventanas de tiempo se dividen en el largo del horizonte de decisión, luego $e'_i = e_i/(l_0 - e_0)$ y $l'_i = l_i/(l_0 - e_0)$. Esto en realidad hay que hacerlo con todas las variables temporales como los tiempos de solicitud, quedando en $ct'_i = ct_i/(l_0 - e_0)$, los tiempos de servicio quedando en $(st)' = st/(l_0 - e_0)$ y los tiempos de viaje entre clientes $t'_{ij} = t_{ij}/(l_0 - e_0)$.

En cuanto a las penalizaciones en la función de recompensas, para el caso de la penalización por cliente no atendido β, en el caso de la distancia euclideana se fijó en $\beta = 6$ y para la distancia Manhattan se fijó en $\beta = 8$. Para el caso de la penalización por cliente atrasado α, en el caso de la distancia euclideana se fijó en $\alpha = 3$ y para la distancia Manhattan se fijó en $\alpha = 4$. Estos valores se impusieron por el tamaño de los valores de las distancias (ya que la idea era que fuera más conveniente visitar un cliente a no visitarlo o visitarlo de manera retrasada) y también por asuntos de convergencia del algoritmo (no era muy conveniente usar constantes muy grandes ya que hacía mucho más difícil entrenar la función acción valor y la función valor o critic). Las penalizaciones en el caso Manhattan son mayores a las del caso euclideano ya que en promedio las distancias del caso Manhattan son mayores a las del caso euclideano. La tasa de descuento de las recompensas en todos los algoritmos se fija en $\gamma = 1$.

El código de los algoritmos fue implementado en Python, usando la librería Tensorflow [1] como motor principal. Esta librería permite implementar redes neuronales de manera sencilla (ya que la mayoría de las redes ya están implementadas) y generar modelos de aprendizaje de máquinas mediante entrenamientos que pueden ser paralelizables por medio de GPU's.

Los entrenamientos fueron realizados por dos medios: para probar y encontrar fallas en el código se utilizó la plataforma Colab de Google que permite utilizar máquinas virtuales con GPU por tiempo limitado; y para los entrenamientos completos se usó una máquina virtual de 32 nucleos y 64 GB de RAM en la plataforma Entel Secure Cloud [30] (sin GPU).

Cada uno de los algoritmos necesita una configuración previa, donde se tienen que fijar hiperparámetros, optimizadores, puntos de control, etc.

6.2. Entrenamiento Double Q-learning con experiencias priorizadas

6.2.1. Configuración

El tamaño de los mini-batch que se extraen de la memoria se fija en 128. El tamaño total de la memoria se fija en 10^6 transiciones, lo que permite que se tenga una gran cantidad de recuerdos pasadas varias iteraciones del entrenamiento. El hiperparámetro ε que controla el grado de exploración se adapta de manera que inicialmente se fija en $\varepsilon_{max} = 1$, para luego descender de manera exponencial hasta cumplir $i_{max} = 60000$ iteraciones del gradiente estocástico dejándolo en $\varepsilon_{min} = 0{,}01$. De manera explicita en cada iteración i el hiperparámetro queda como:

$$\varepsilon_i = \varepsilon_{min} + (\varepsilon_{max} - \varepsilon_{min}) \exp(\log(0{,}001)\frac{i}{i_{max}})$$

Los hiperparámetros α y β correspondientes al algoritmo de experiencias priorizadas se dejan tal cual se usaron en el paper original [31], es decir, $\alpha = 0{,}6$ constante. En cuanto a β, inicialmente se fija $\beta = \beta_0 = 0{,}4$ para luego incrementarlo de forma lineal hasta $\beta = 1$, cuando termina el entrenamiento.

El número de iteraciones con que se entrena el algoritmo se deja en 20000. En cuanto al método optimizador, se utiliza el algoritmo Adam [18], que permite realizar descenso de gradiente estocástico adaptando la tasa de aprendizaje a lo largo del entrenamiento. La tasa de aprendizaje inicial que se utiliza se deja en 10^{-4}.

6.2.2. Entrenamientos

Una de las configuraciones en que no estaba claro cuál era la mejor opción posible es como se actualiza el parámetro θ' de la red objetivo con respecto al parámetro θ de la red primaria.

Una alternativa es tomar $\tau << 1$ y realizar la actualización en todas las iteraciones mediante

$$\theta' \leftarrow \tau\theta + (1 - \tau)\theta'$$

La otra alternativa es actualizar el parámetro θ' igualándolo completamente a θ cada ciertas iteraciones.

Para ver cuál alternativa es mejor se realizaron 2 entrenamientos usando distancia euclideana por separado con cada alternativa. Al resultado de la primera alternativa mencionada

se le llamó “modelo 1” a la segunda alternativa “modelo 2”.

En cada uno de los entrenamientos se guardó un registro de cómo iba avanzando la función de pérdidas y las recompensas obtenidas. Para mejor visualización, en cada iteración se promediaron los resultados de las últimas 20 iteraciones en las recompensas.

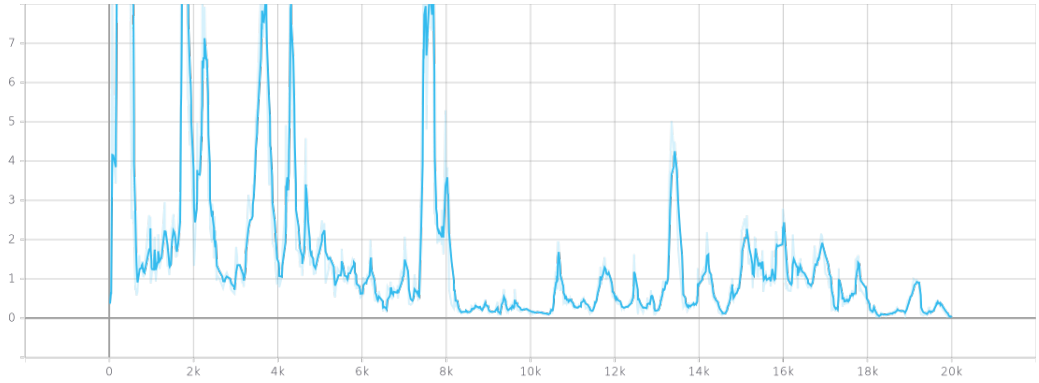

Figura 6.1: Función de pérdidas modelo 1 DDQN con experiencias priorizadas usando distancia euclideana en cada iteración del entrenamiento

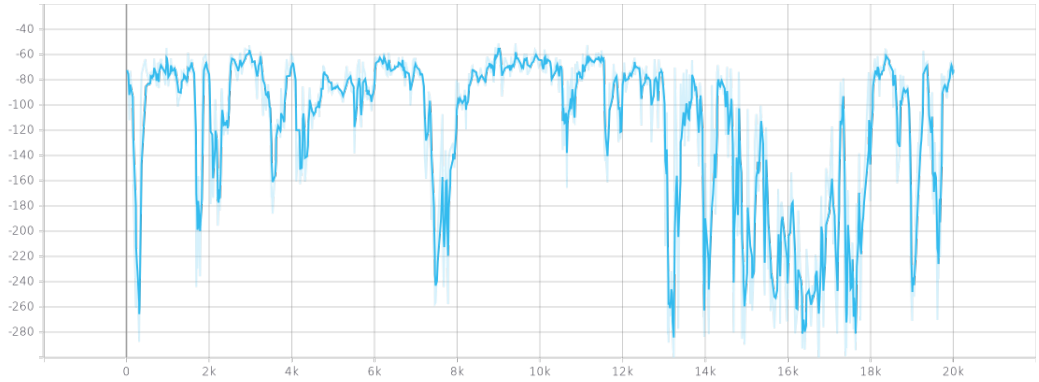

Figura 6.2: Función de recompensas modelo 1 DDQN con experiencias priorizadas usando distancia euclideana en cada iteración del entrenamiento

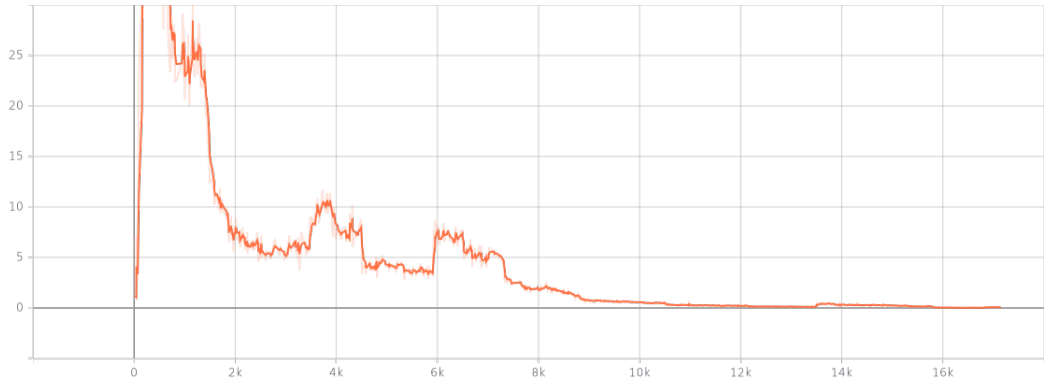

Figura 6.3: Función de pérdidas modelo 2 DDQN con experiencias priorizadas usando distancia euclideana en cada iteración del entrenamiento

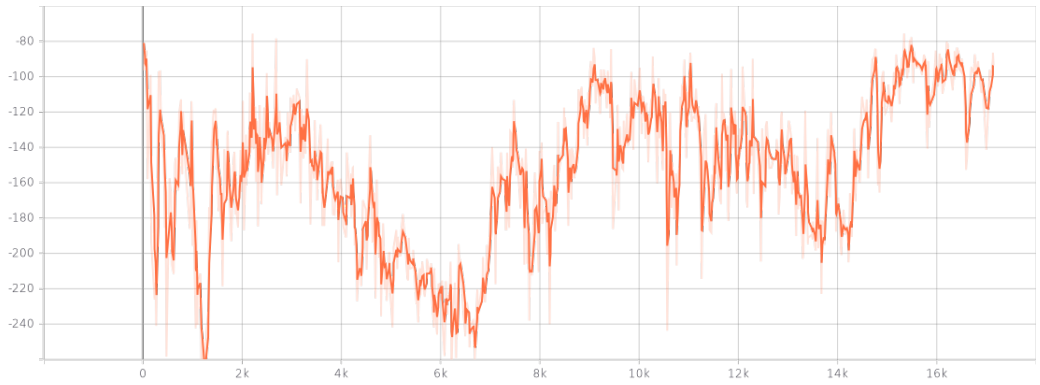

Figura 6.4: Función de recompensas modelo 2 DDQN con experiencias priorizadas usando distancia euclideana en cada iteración del entrenamiento

Como se puede observar las funciones de pérdidas de ambos modelos convergen a valores cercanos a 0, lo cual en un principio podría indicar que se llegó al mejor resultado posible. Sin embargo las funciones de recompensas nunca lograron mejorar sus resultados, mostrando curvas inestables a lo largo del entrenamiento. Una posible hipótesis puede ser que la optimización dio con un mínimo local, estancándose a un nivel de recompensas bajo. Cabe notar que el entrenamiento era bastante lento (las 20.000 iteraciones representaron 9 días de entrenamiento), y al ver ninguna mejoría en ambos modelos se decidió no seguir entrenándolo. Por esta misma causa no hubo razón de entrenarlo usando distancia Manhattan.

6.3. Entrenamiento algoritmo Actor-Critic

6.3.1. Configuración

En este algoritmo se procesan las entradas en batches de tamaño 128. Este algoritmo se entrena en un principio con 20000 iteraciones, sin embargo, se da cuenta que este puede seguir mejorando, por lo que se retoma el entrenamiento con algunas iteraciones más (esto se puede hacer generando un punto de control o "checkpoint" que permite retomar tal cual un entrenamiento como se dejó). El algoritmo Adam se utiliza para el optimizador, con tasa de aprendizaje inicial igual a 10^{-4}.

6.3.2. Entrenamientos

En un principio los entrenamientos de este algoritmo se hicieron usando distancia euclideana, guardando puntos de control cada 1000 iteraciones. Esto se debió a que el algoritmo alcanzó mejores resultados que el algoritmo DDQN con experiencias priorizadas, y se buscaba guardar varios modelos para posteriormente probarlos en inferencia de tal manera que no haya un sobre ajuste sobre los datos usados. Esta técnica es común usarla cuando un algoritmo converge a sus mejores resultados "deep learning", ya que es común que cuando se sobre entrena un modelo, este puede sobreajustarse, dando resultados no generalizables en inferencia.

En resumen se dejan a probar los siguientes modelos con distancia euclideana:

- Modelo 1: guardado en la iteración 5.000

- Modelo 2: guardado en la iteración 10.000
- Modelo 3: guardado en la iteración 20.000
- Modelo 4: guardado en la iteración 25.000
- Modelo 5: guardado en la iteración 34.000
- Modelo 6: guardado en la iteración 48.000
- Modelo 7: guardado en la iteración 66.000

El entrenamiento se finalizó cercano a la iteración 66.000, lo cual en su totalidad representó un entrenamiento de 23 días.

A continuación se muestra la evolución del entrenamiento hasta la iteración 20.000, donde la función de pérdidas totales corresponde a la suma de la función de pérdidas de la red Actor y la red Critic.

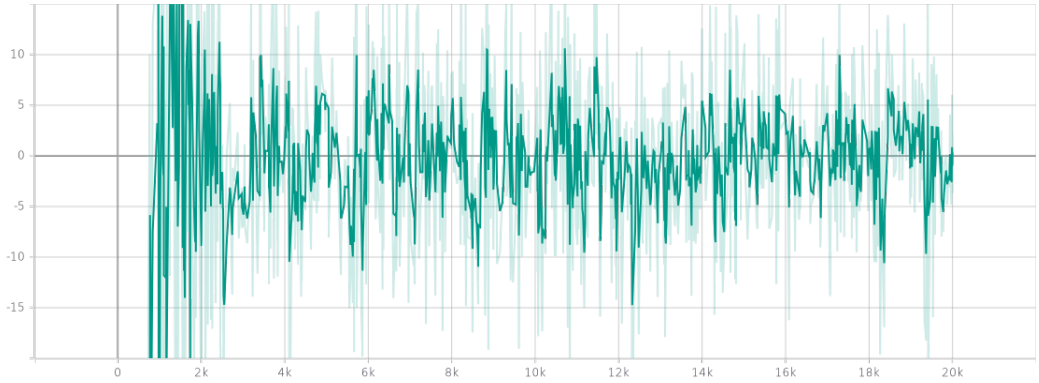

Figura 6.5: Función de pérdidas red Actor usando distancia euclideana en cada iteración del entrenamiento

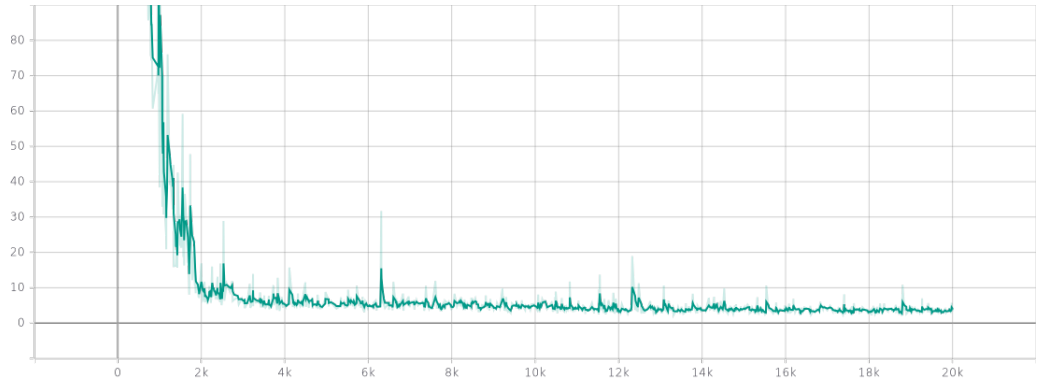

Figura 6.6: Función de pérdidas red Critic usando distancia euclideana en cada iteración del entrenamiento

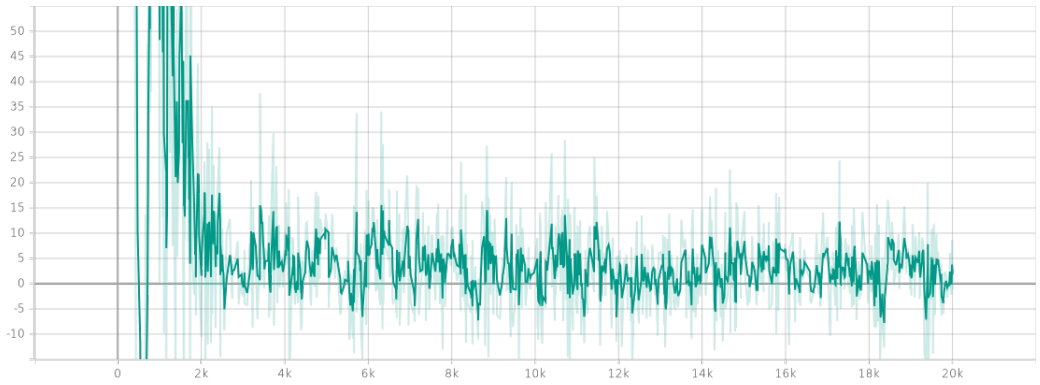

Figura 6.7: Función de pérdidas totales modelo actor-critic usando distancia euclideana en cada iteración del entrenamiento

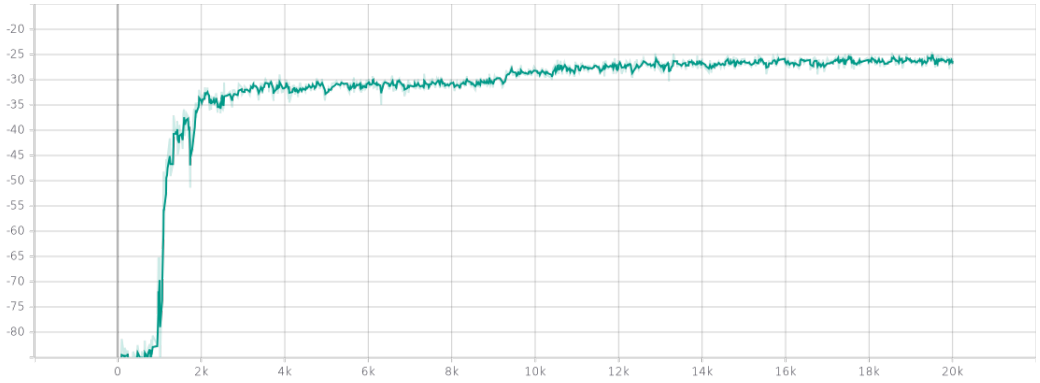

Figura 6.8: Función de recompensas modelo actor-critic usando distancia euclideana en cada iteración del entrenamiento

Cabe notar que como se puede observar en la figura 6.8, el entrenamiento logra rápidamente elevar las recompensas, superando totalmente al algoritmo DDQN con experiencias priorizadas. Del gráfico se puede inferir que en un principio el agente trata de reducir el número de clientes no visitados (que son los que entregan mayor recompensa negativa) y el número de clientes que se llega tarde. Al final del entrenamiento (especialmente después de la iteración 20.000) la recompensa es aumentada levemente ya que el agente sólo se preocupa de recorrer menos distancia tratando de no encontrarse con clientes atrasados. Este fenómeno va a ser mejor observado cuando se muestren los resultados experimentales de este modelo.

En el caso de la distancia Manhattan, las funciones de pérdida resultaron ser similares, sin embargo la convergencia fue un poco más inestable, probablemente por los valores de las penalizaciones que resultan mayores a los de la distancia euclideana. Esto se puede reflejar en la función de recompensas de la siguiente figura, que muestra el entrenamiento hasta la iteración 15.000

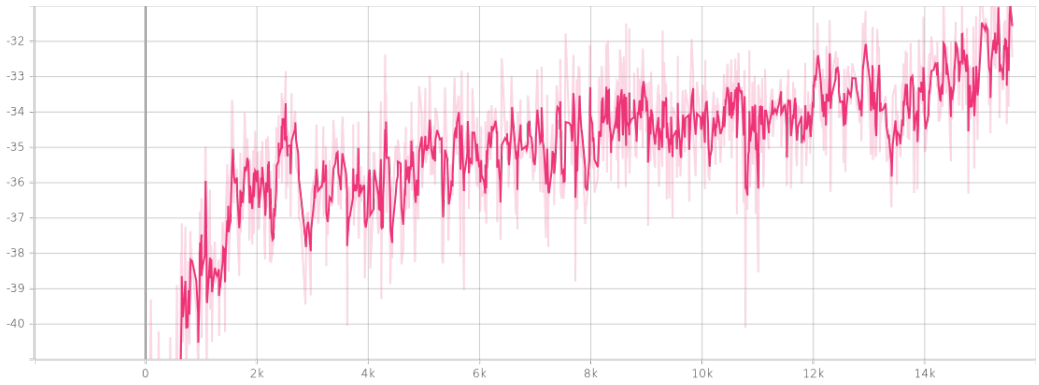

Figura 6.9: Función de recompensas modelo actor-critic usando distancia Manhattan en cada iteración del entrenamiento

6.4. Entrenamiento algoritmo estocástico con búsqueda

6.4.1. Configuración

En este algoritmo se procesan las entradas en batches de tamaño 128. En cuanto al tamaño de los mini-batch que se extraen de la memoria para el entrenamiento de la política imitadora se usa un tamaño de 512. El tamaño total de la memoria se fija en 10^6 transiciones, lo que permite que se tenga una gran cantidad de recuerdos pasadas varias iteraciones del entrenamiento. Este algoritmo se entrena con 60.000 iteraciones. Para el entrenamiento del sub-problema estático se utiliza el algoritmo Adam como optimizador, con tasa de aprendizaje inicial igual a 10^{-4}. En cuanto al entrenamiento de la política imitadora se utiliza descenso de gradiente estocástico con tasa de aprendizaje de $0,1$ y momentum de 0,9. La temperatura de la distribución se deja igual a 1, lo que significa que las acciones se toman tomando la distribución de probabilidad dada por los contadores normalizados.

En relación a las simulaciones internas realizadas por el algoritmo, para cada instancia se dejó como entrada los centros y las varianzas de las distribuciones normales de las posiciones. No se realizó ninguna suposición sobre los tiempos de solicitudes de los clientes .El número de simulaciones realizadas por iteración fue de 25 y a constante $PUCT$ se deja igual a 5. Este valor fue obtenido a partir de una simple calibración inicial. Por los largos tiempos de entrenamiento, especialmente en este algoritmo, no se probaron otros hiperparámetros para mejorar los resultados de los entrenamientos.

6.4.2. Entrenamientos

Al igual que el algoritmo actor-critic, se guardaron puntos de control cada 1000 iteraciones. En un principio se buscaba generar varios modelos para probarlos y evitar el sobreajuste, sin embargo, el algoritmo resultaba muy lento al resolver varias veces el problema estático sobre los clientes conocidos y simulados. Más adelante se describirá una forma sencilla mediante la cual se redujo el tiempo de ejecución en inferencia (no así en entrenamiento).

En un principio se genera un único modelo en la iteración 60.000 usando distancia euclideana.

A continuación se muestra la evolución del entrenamiento de la política $\pi_\theta(a|s)$ que imita la política $p(s,a)$ obtenida de la ruta de simulaciones, y la política $\pi_{\theta'}^{sim}$ que actúa como actor para el subproblema estático.

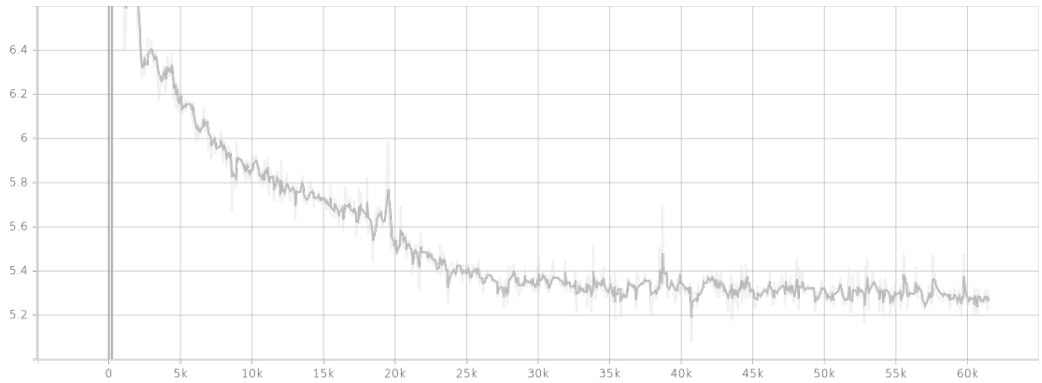

Figura 6.10: Función de pérdidas entropía cruzada para aprendizaje por imitación de π_θ, usando distancia euclideana en cada iteración del entrenamiento

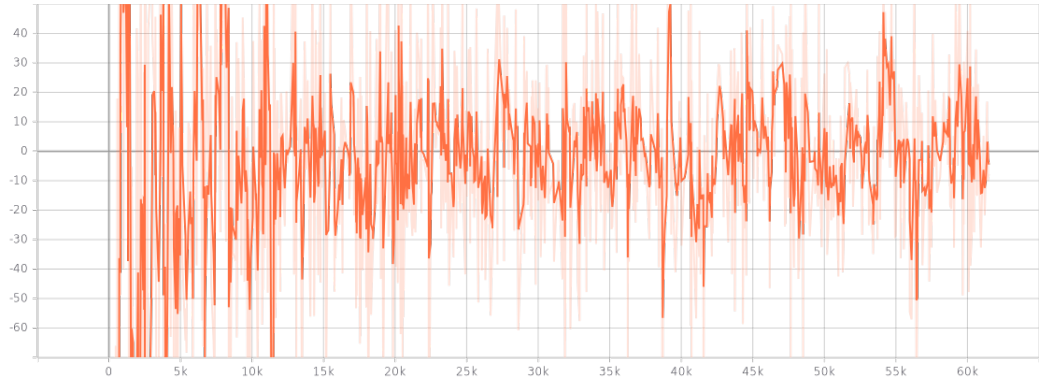

Figura 6.11: Función de pérdidas totales modelo actor-critic sobre el subproblema estático del algoritmo que usa información estocástica, usando distancia euclideana en cada iteración del entrenamiento

Figura 6.12: Función de recompensas del subproblema estático del algoritmo que usa información estocástica, usando distancia euclideana en cada iteración del entrenamiento

Como se puede observar, el problema de optimización sobre el subproblema estático logra converger, dando aparentes buenos resultados para las recompensas de ese problema. La

función de pérdidas de entropía cruzada también logra converger, no generando mejoras substanciales después de la iteración 40.000.

En cuanto a las recompensas del problema completo, se decidió no graficarlas ya que éstas eran por resultado del problema de imitación y la optimización del subproblema estático. En los resultados experimentales, se verá con mayor claridad si estos dos fenómenos tuvieron algún efecto sobre las recompensas del problema completo.

En cuanto al entrenamiento realizado con distancia Manhattan, los gráfico de las funciones de pérdidas y la recompensa del problema estático resultaron similares en comportamiento a la del entrenamiento con distancia euclideana. De igual manera se verá más adelante si haber convergido en el problema estático y en la función de pérdidas de la entropía cruzada se ve reflejado en las recompensas totales.

6.5. Resultados experimentales

Para dejar en claro el desempeño de los modelos, estos se prueban en modo inferencia (es decir post-entrenamiento) en un conjunto de datos simulado. Cabe mencionar que todas las pruebas que se muestran a continuación se hicieron con distancia euclideana para tener una medición inicial de los modelos y poder calcular cuánta es la cantidad de iteraciones de entrenamiento necesarias para tener un modelo competitivo.

En estas pruebas se mantiene el nivel de dinamismo pero se varía la longitud de las ventanas de tiempo, dejándolas con una media de 30 y una desviación estándar de 3. Por otro lado se prueban bajo las dos distribuciones de las posiciones usadas en entrenamiento: uniforme y agrupadas de acuerdo a una distribución normal. En cuanto al número de clientes y vehículos, se van a hacer pruebas en 3 casos:

- 40 clientes y 12 vehículos
- 80 clientes y 20 vehículos
- 120 clientes y 25 vehículos

La capacidad de los vehículos es igual a la usada en los entrenamientos, es decir, $Q = 4$.

Las variables que se miden corresponden a:

- Distancia recorrida
- Porcentaje de retraso total con respecto a la longitud de la ventana de tiempo de la bodega. Como la ventana es $[0, 100]$, hablar de porcentaje de retraso y retraso absoluto es lo mismo.
- Número de clientes no visitados
- Número de vehículos usados

A continuación se muestran gráficos comparativos de los modelos obtenidos.

6.5.1. Algoritmo DDQN con experiencias priorizadas

En este algoritmo se muestran resultados de los dos modelos entrenados, midiendo las variables mencionadas

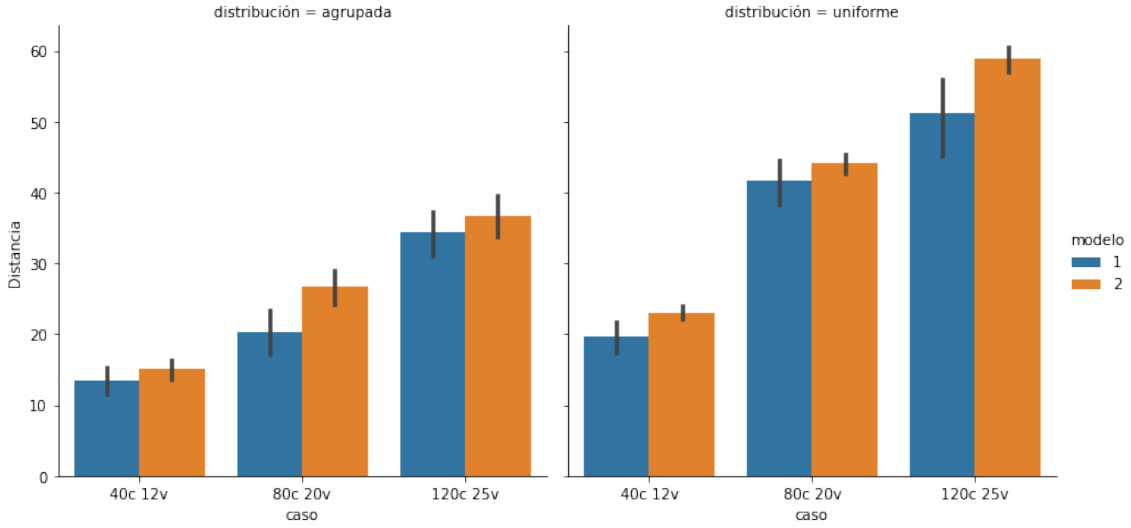

Figura 6.13: Distancia recorrida modelos DDQN con experiencias priorizadas, en varios escenarios

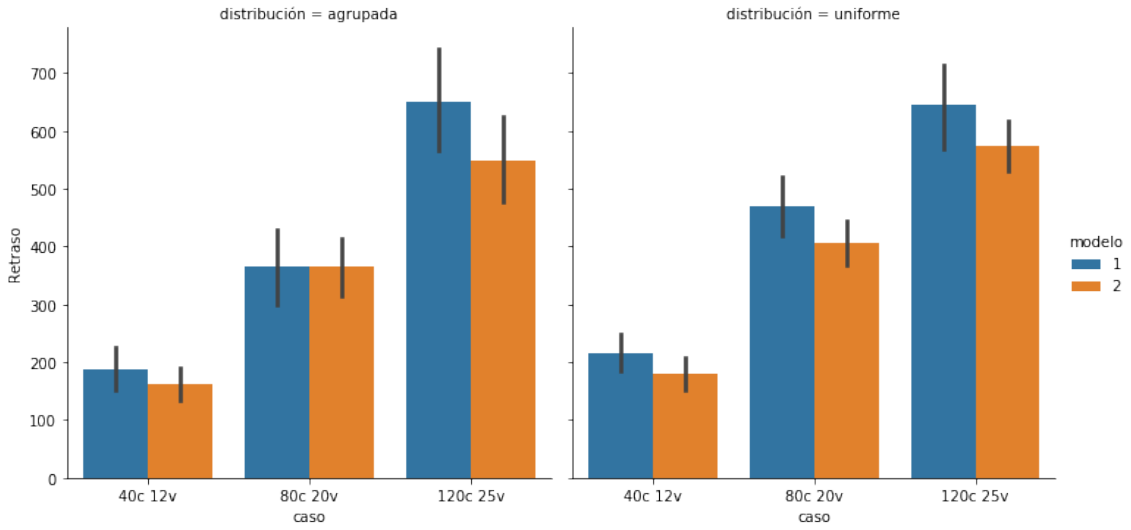

Figura 6.14: Porcentaje de retraso modelos DDQN con experiencias priorizadas, en varios escenarios

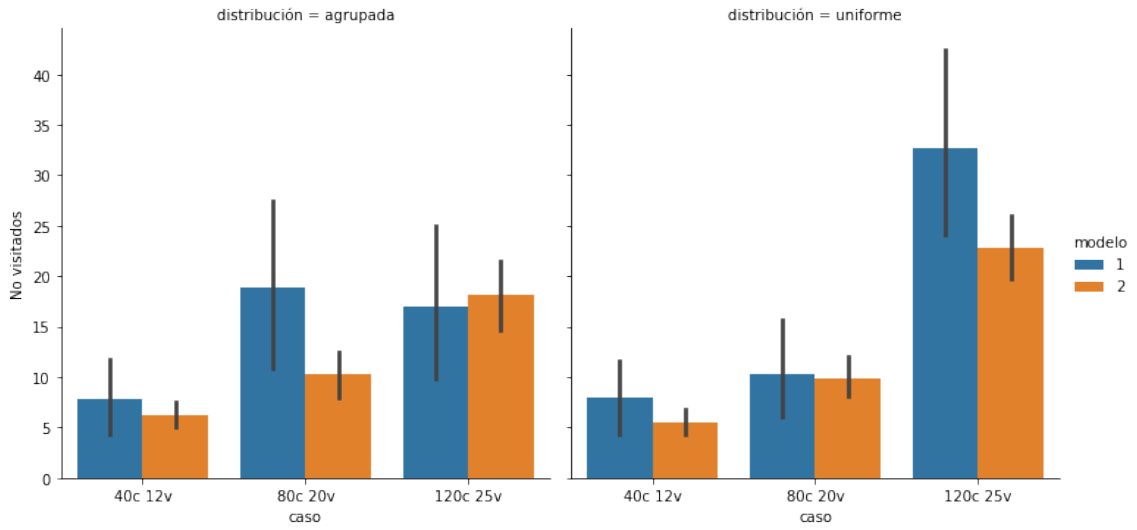

Figura 6.15: Número de clientes no visitados en modelos DDQN con experiencias priorizadas, en varios escenarios

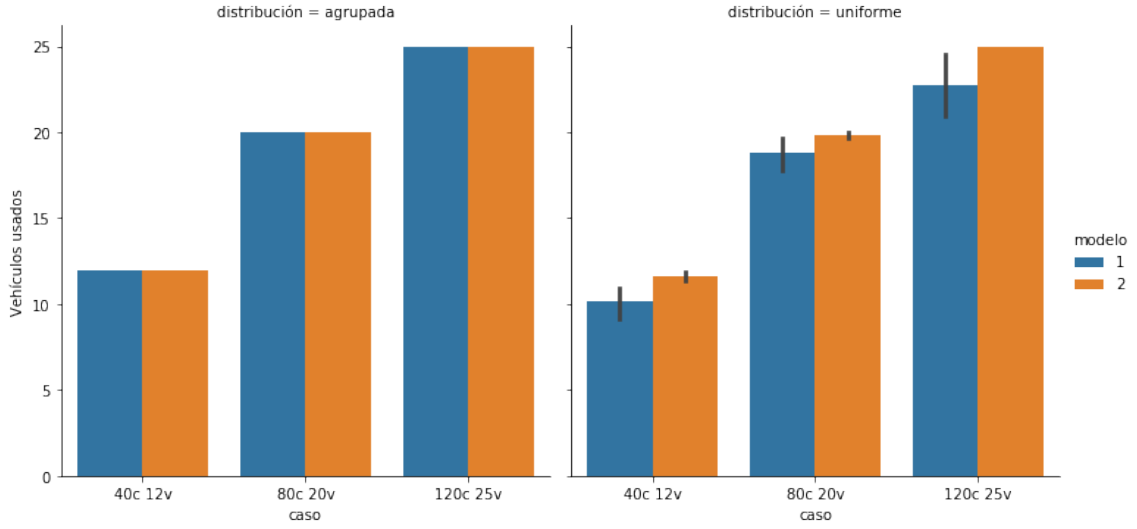

Figura 6.16: Número de vehículos usados en modelos DDQN con experiencias priorizadas, en varios escenarios

Como se puede observar, ambos modelos en inferencia son muy parecidos en cuanto a desempeño. Las distancias recorridas del modelo 1 son menores a las del modelo 2, sin embargo esto se puede deber a que el número de clientes no visitados del modelo 1 es mayor al modelo 2. En cualquier caso el porcentaje de retraso es muy grande al igual que el número de clientes no visitados, lo que dice que ambos modelos no lograron el objetivo de aprender, mediante una función acción valor, una política cercana a la óptima, lo cual también se ve reflejado en las funciones de recompensas observadas en entrenamiento.

6.5.2. Algoritmo Actor-Critic

En el caso de este algoritmo, como la política es estocástica, está la posibilidad de usar la política tal cual usando una estrategia de simulación. Por otra parte se puede usar la estrategia de tomar la acción que tenga la mayor probabilidad. Es decir para $\pi_\theta(\cdot|s)$ política en el estado $s \in S$, se toma la acción $a = \arg\max_{a'} \pi_\theta(a'|s)$. A la primera estrategia se le llamará estrategia aleatoria y a la segunda estrategia glotona. A continuación se muestran los resultados obtenidos para los 7 modelos, usando distancia euclideana

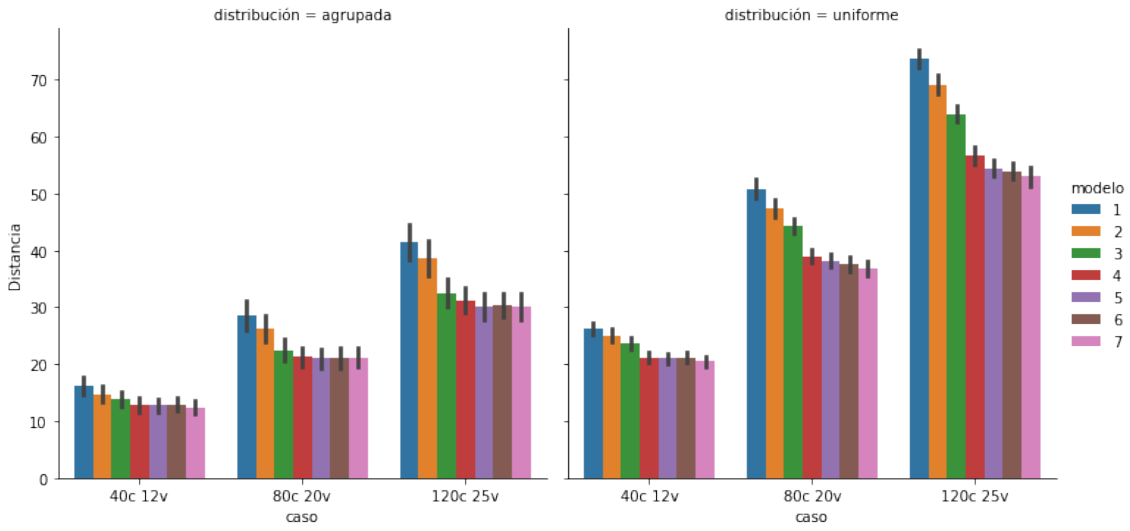

Figura 6.17: Distancia recorrida modelos actor-critic con estrategia aleatoria, en varios escenarios

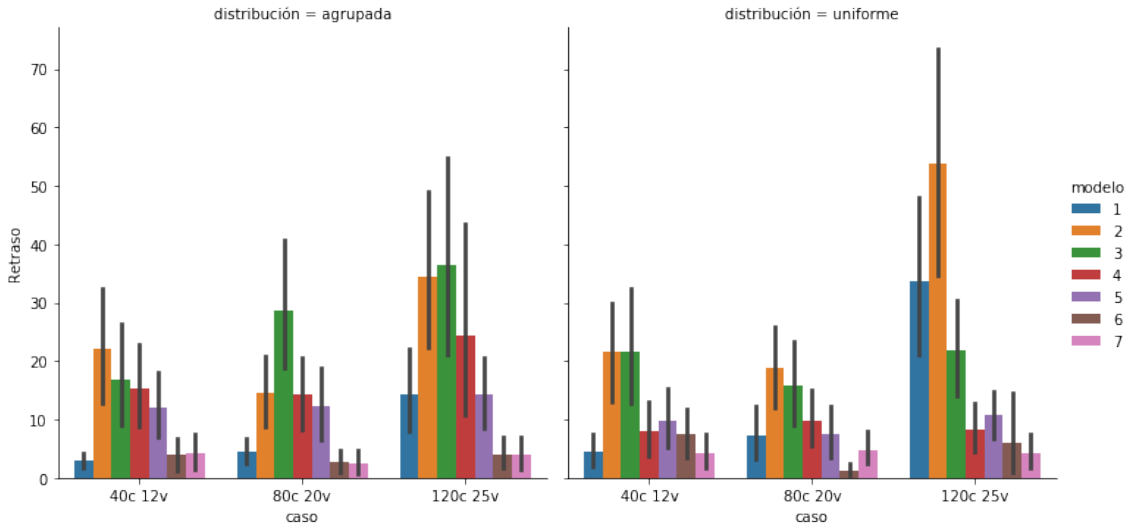

Figura 6.18: Porcentaje de retraso modelos actor-critic con estrategia aleatoria, en varios escenarios

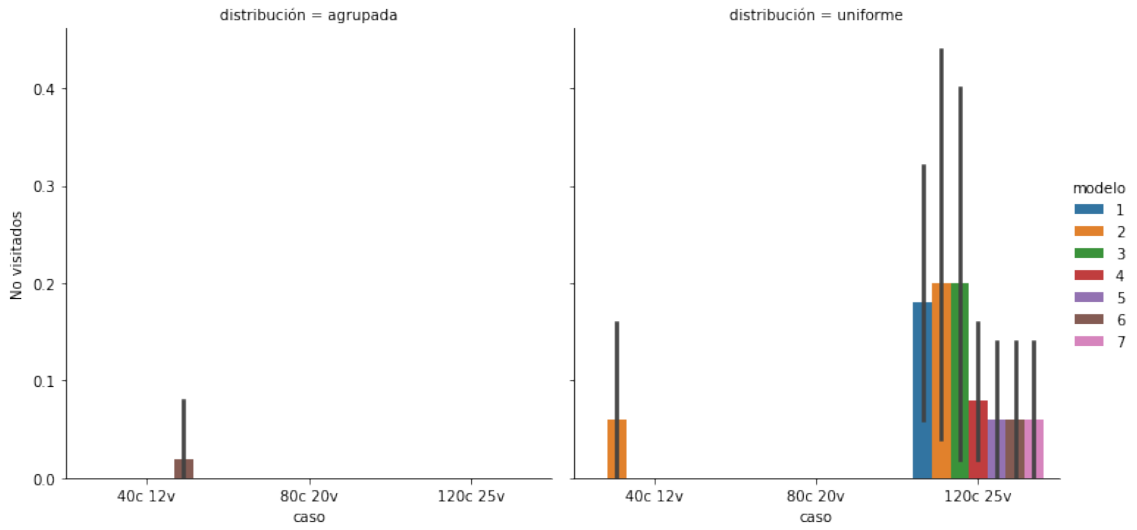

Figura 6.19: Número de clientes no visitados modelos actor-critic con estrategia aleatoria, en varios escenarios

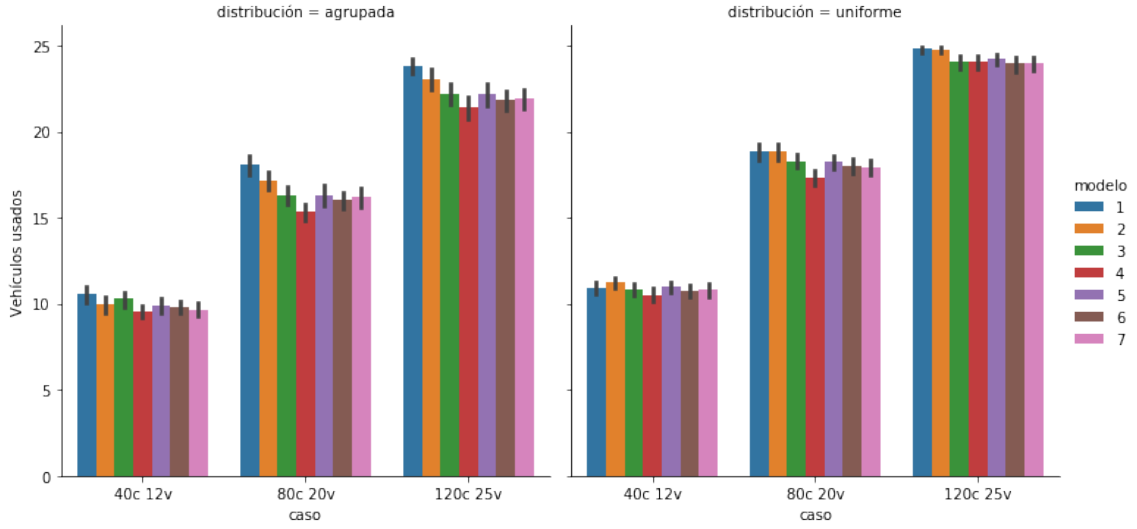

Figura 6.20: Número de vehículos usados modelos actor-critic con estrategia aleatoria, en varios escenarios

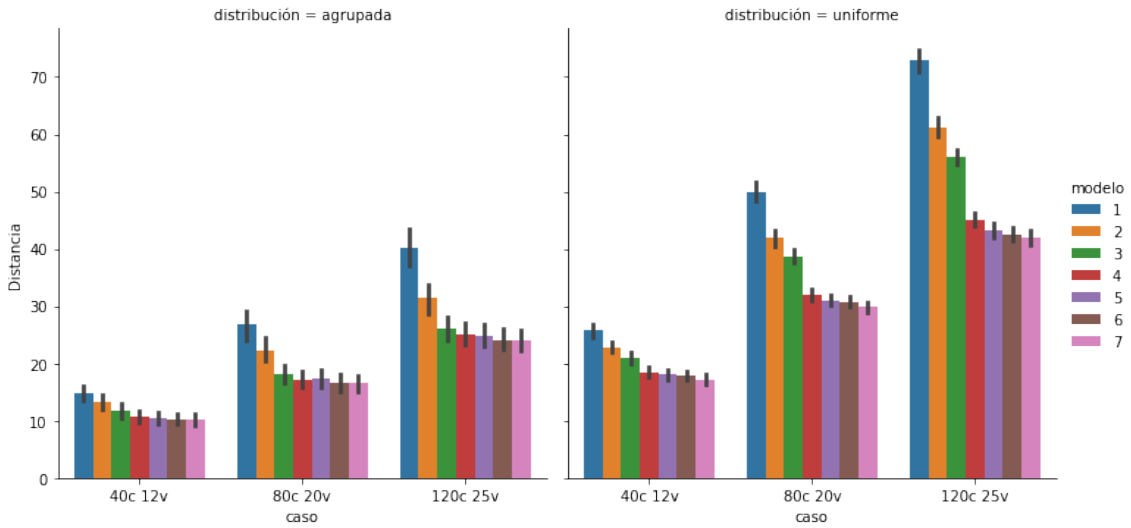

Figura 6.21: Distancia recorrida modelos actor-critic con estrategia glotona, en varios escenarios

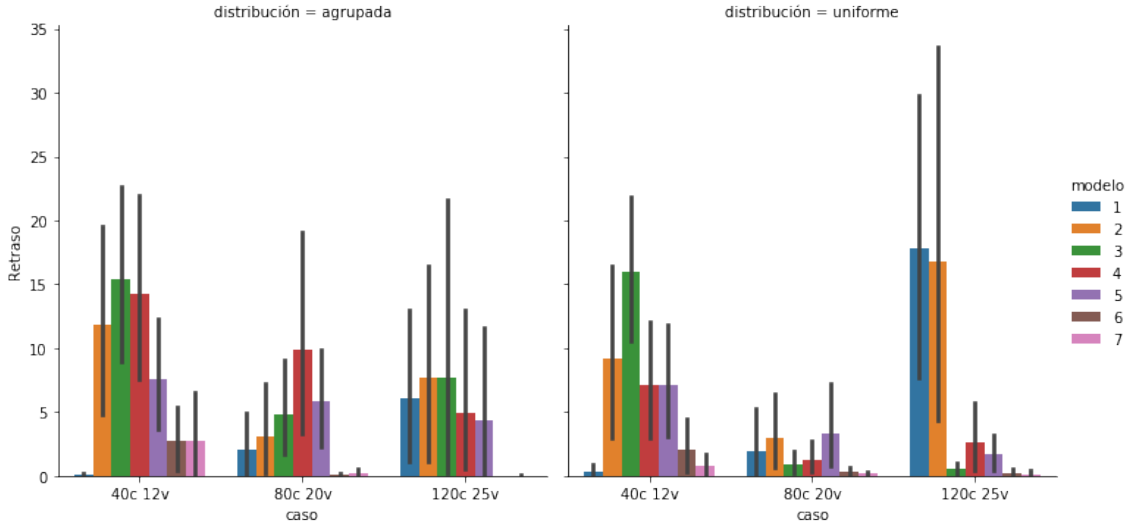

Figura 6.22: Porcentaje de retraso modelos actor-critic con estrategia glotona, en varios escenarios

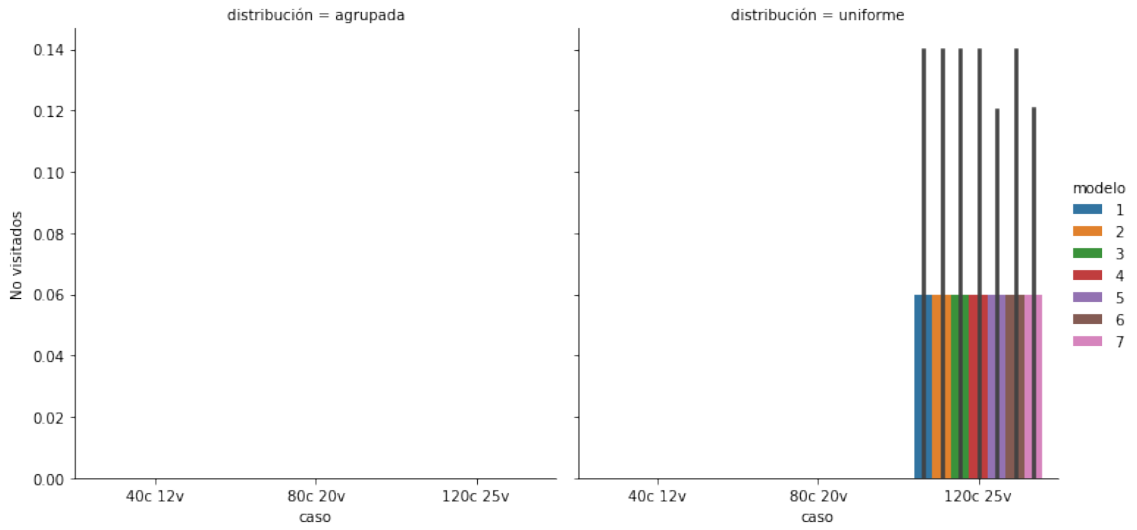

Figura 6.23: Número de clientes no visitados modelos actor-critic con estrategia glotona, en varios escenarios

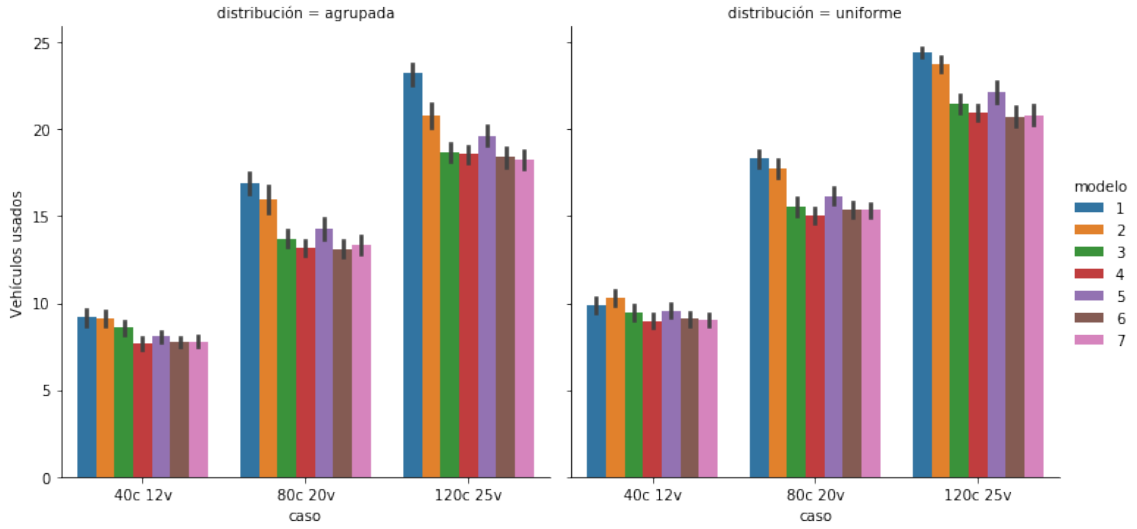

Figura 6.24: Número de vehículos usados modelos actor-critic con estrategia glotona, en varios escenarios

Como se puede observar en las figuras 6.19 y 6.23 el número de clientes no visitados promedio es siempre menor a 1, incluso con un intervalo de confianza menor a 0,5.

En cuanto a las distancias recorridas mostradas en 6.17 y 6.21 se puede ver el efecto de que al aumentar el número de iteraciones del entrenamiento el modelo va perfeccionándose minimizando la distancia recorrida, especialmente en las transiciones de los modelos 1 al 4.

Algo bien especial ocurre con los retrasos promedios en las figuras 6.18 y 6.22, donde en el modelo 1 tiene un retraso más bajo que por ejemplo en los modelos 2, 3, 4, y 5. Esta cualidad logra superarse a partir del modelo 6, donde además de tener baja distancia, tiene un bajo porcentaje de retraso.

En relación a la comparación del modelo aleatorio y el modelo glotón, a continuación se

muestra una comparación mucho más explícita usando el modelo 7 que es el que tuvo mejores resultados (dejando de lado la teoría inicial del sobre ajuste a los datos).

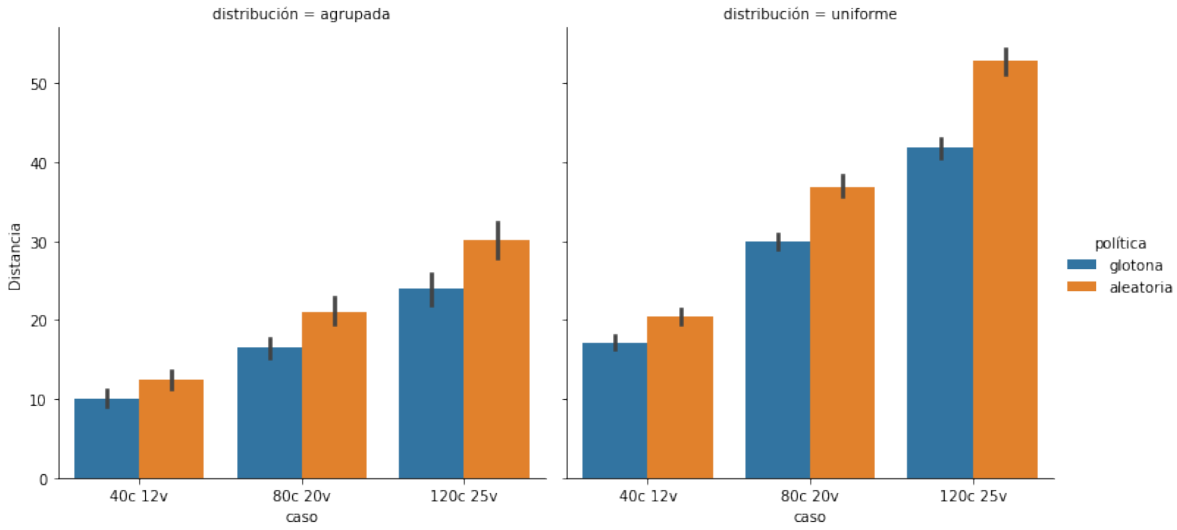

Figura 6.25: Distancia recorrida modelo 7 actor-critic con estrategia aleatoria y glotona, en varios escenarios

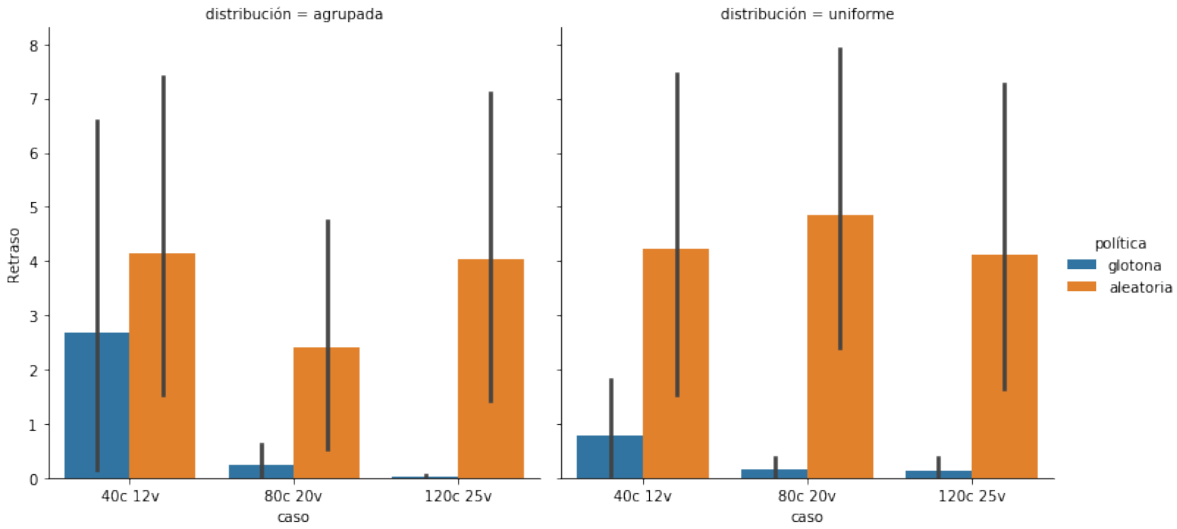

Figura 6.26: Porcentaje de retraso modelo 7 actor-critic con estrategia aleatoria y glotona, en varios escenarios

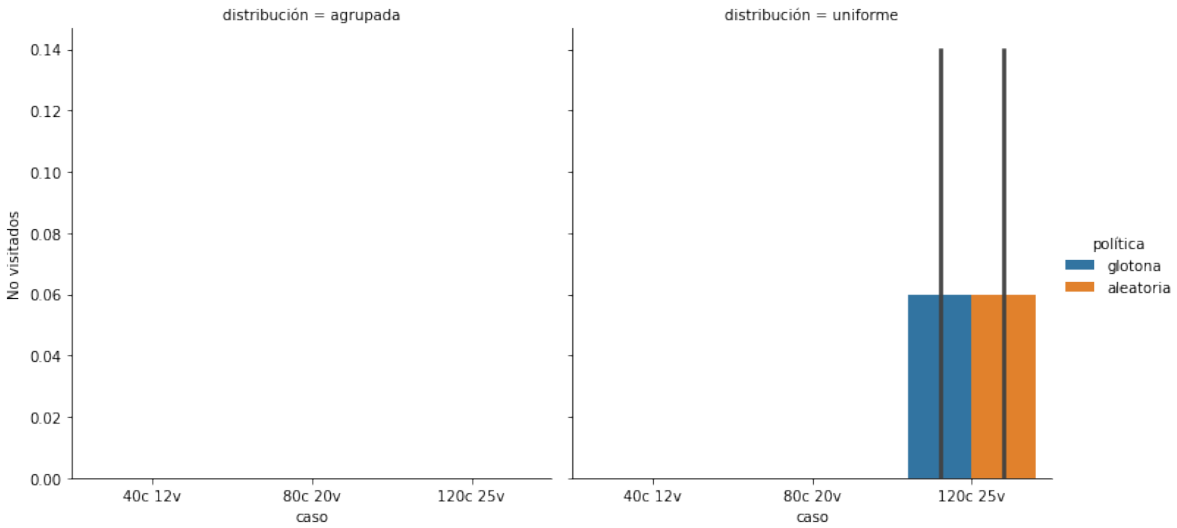

Figura 6.27: Número de clientes no visitados modelo 7 actor-critic con estrategia aleatoria y glotona, en varios escenarios

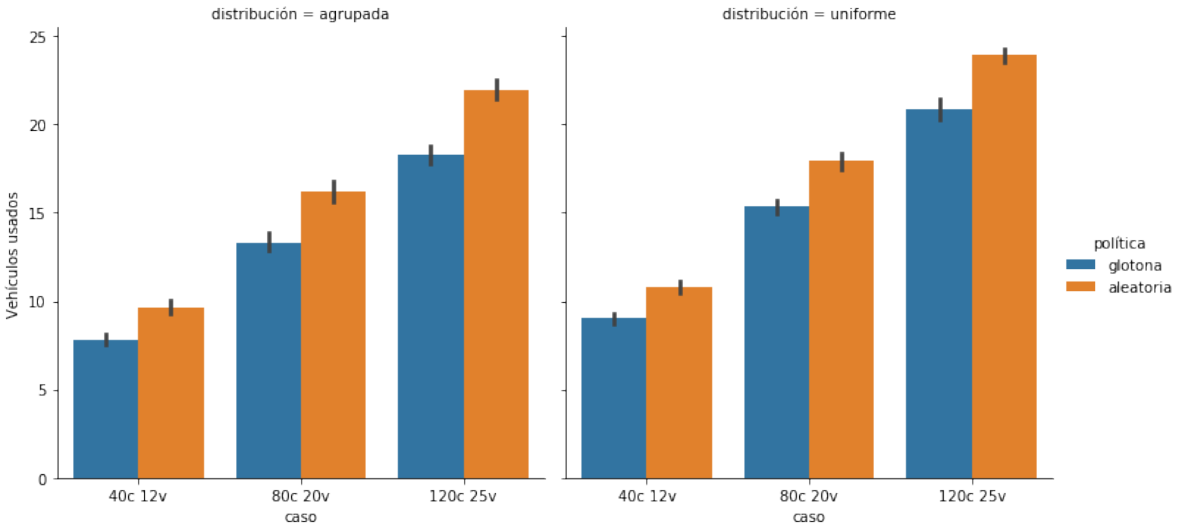

Figura 6.28: Número de vehículos usados modelo 7 actor-critic con estrategia aleatoria y glotona, en varios escenarios

Como se puede observar el número de clientes no visitados en el modelo 7 en ambas estrategias es casi despreciable, donde en promedio es de aprox. 0,06 clientes, con un intervalo de confianza que no es mayor que 0,14.

Por el lado de la distancia recorrida, el porcentaje de retraso y el número de vehículos usados, la política glotona es una clara ganadora, especialmente por el lado del porcentaje de retraso donde en todos los casos menos 1, este es bajo un 1 %.

6.5.3. Algoritmo estocástico con búsqueda

En esta sección se muestran resultados experimentales del algoritmo estocástico con búsqueda. Como este algoritmo se entrenó en el caso de clientes agrupados, sólo se experimentará en ese caso (ya que no tiene sentido probarlo en el caso de distribución espacial uniforme).

En cuanto a la temperatura τ de la distribución de probabilidad obtenida de las simulaciones, se tomó $\tau \to 0$, es decir, siempre se tomó la acción con mayor contador.

Por el largo tiempo de ejecución de este algoritmo, se trató de disminuir este tiempo en modo inferencia, de tal manera que influya lo menos posible en el rendimiento original del modelo. Lo que se hizo fue en que en vez de seleccionar sólo una acción cada vez que se realiza las simulaciones internas, se seleccionen más de una acción a la vez, con tal de ejecutar el menor número de simulaciones posible (ya que al final la parte de las simulaciones es lo que le agrega tiempo de ejecución al algoritmo).

La manera de seleccionar n acciones a la vez es tomar de la matriz de visitas obtenida de las simulaciones, las n acciones que tengan mayor contador. Sin embargo, esto se puede hacer 1 vehículo a la vez, es decir, si por ejemplo el vehículo j sale incluido en la primera acción que tiene mayor contador, este vehículo para las siguientes $n-1$ acciones queda bloqueado. Esto se hace ya que si no quedara bloqueado, cambiaría el contexto en que se realizaron las simulaciones originales. Por otro lado si por ejemplo una acción involucra a un vehículo que se encuentra en la bodega, ese vehículo queda bloqueado (por lo comentado antes) pero todos los otros contadores que estaban asignados a otros clientes a ese vehículo se copian a otro vehículo que se encuentre en la bodega (si es que existe). Esto es ya que dos o más vehículos que se encuentren en la bodega actúan como equivalentes (ya que todos los vehículos se asumen iguales).

Por todo lo dicho antes, el número de acciones simultaneas tiene que ser menor o igual al número de vehículos. Con tal de no tomar demasiadas acciones simultaneas y bajar los tiempos de ejecución se tomó la decisión de que en el caso de 40 clientes y 12 vehículos se seleccionan 3 acciones por cada simulación; en el caso de 80 clientes y 20 vehículos se seleccionan 5 acciones por cada simulación y en el caso de 120 clientes y 25 vehículos se seleccionan 8 acciones por cada simulación.

Por otra parte, para que el tiempo de ejecución bajara aún más, el número de simulaciones internas se bajó a 15 (en entrenamiento se fijó en 25).

Como sólo se prueba el último modelo generado por este algoritmo, para tener una referencia, se compara con el modelo 7 del algoritmo actor-critic usando una política glotona, que fue el mejor modelo de este algoritmo. A continuación se muestran las figuras, usando distancia euclideana.

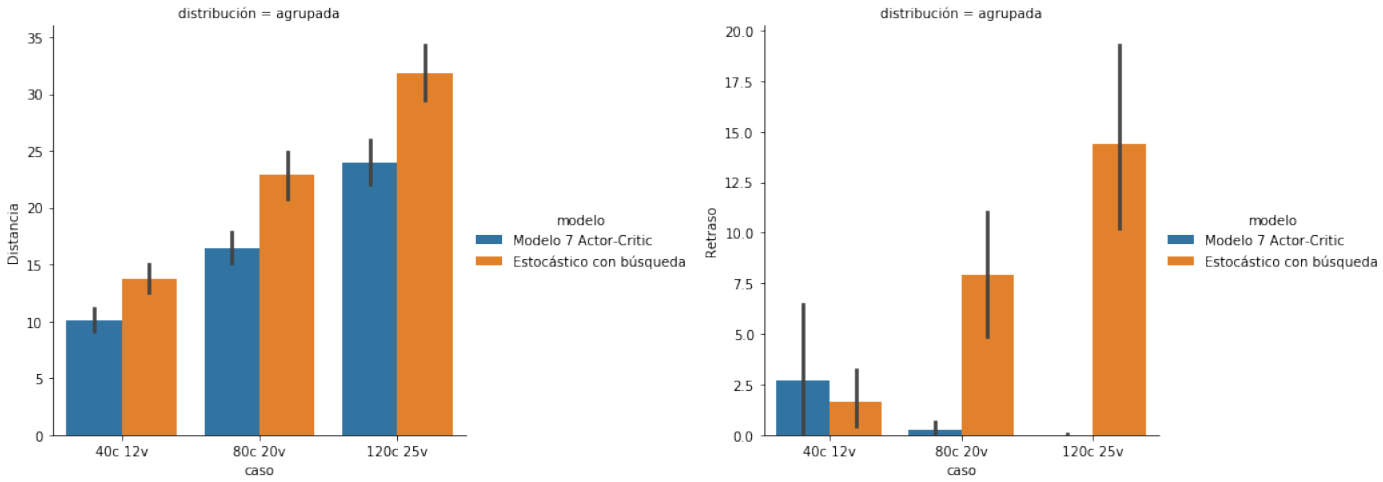

Figura 6.29: Distancia recorrida y porcentaje de retraso modelo estocástico con búsqueda vs modelo 7 actor-critic con estrategia glotona en varios escenarios

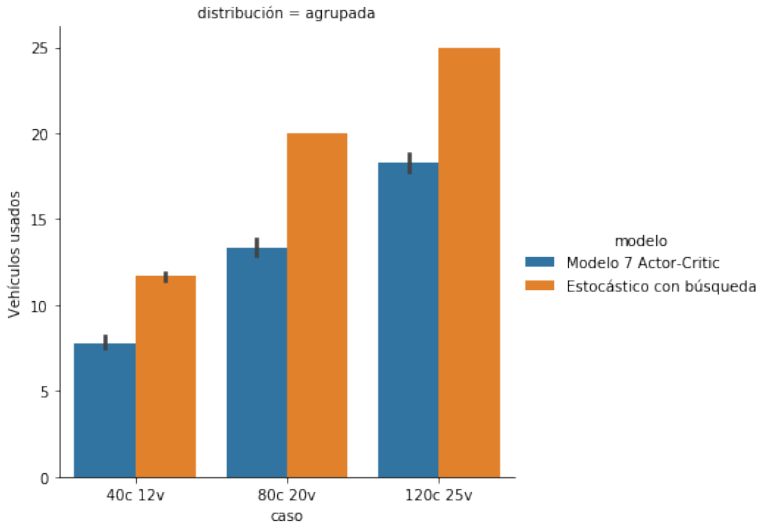

Figura 6.30: Vehículos usados modelo estocástico con búsqueda vs modelo 7 actor-critic con estrategia glotona en varios escenarios

Cabe notar que ambos modelos comparados no tuvieron clientes no visitados. Sin embargo, a pesar de que el modelo estocástico con búsqueda es un algoritmo que tiene más información que el algoritmo actor-critic, no logró superarlo en ninguna de las variables medidas. Como se puede observar en la figura 6.29 el porcentaje de retraso es muy superior. Lo que se piensa que es que las modificaciones que se le hicieron en la selección de acciones y el número de simulaciones empeoró bastante el rendimiento del algoritmo.

6.5.4. Otros experimentos

Dado que ya se tiene una idea de cuáles son los modelos que mejor funcionan con distancia euclideana, y cuántas iteraciones son necesarias para tener un modelo que sea competitivo, es el turno de ver como se comportan los modelos entrenados con distancia Manhattan.

En el caso del algoritmo estocástico con búsqueda, las acciones se tomaron con la misma estrategia de las acciones simultáneas usadas en las pruebas para a distancia euclideana. En el caso del algoritmo Actor-Critic, se decide probar el último modelo generado, ya que como se pudo observar, los modelos de este algoritmo tuvieron una mejora continua a medida que pasaban las iteraciones del entrenamiento. Como el modelo Q-learning no fue entrenado con distancia Manhattan, sólo se comparara el modelo generado por el algoritmo estocástico con búsqueda y el modelo generado por el algoritmo Actor-Critic, usando distancia Manhattan en un ambiente de distribuciones espaciales agrupadas. Por el buen rendimiento de la estrategia glotona en el modelo actor-critic, aquí también se mantiene esa estrategia.

En cuanto a los hiperparámetros de los datos generados, se usan los mismos que de la distancia euclideana excepto por la velocidad que se fija en 0,25 (al igual que en los entrenamientos).

A continuación se muestran gráficos que comparan distancia recorrida, porcentaje de retraso, número de clientes no visitados y número de vehículos usados entre estos dos modelos

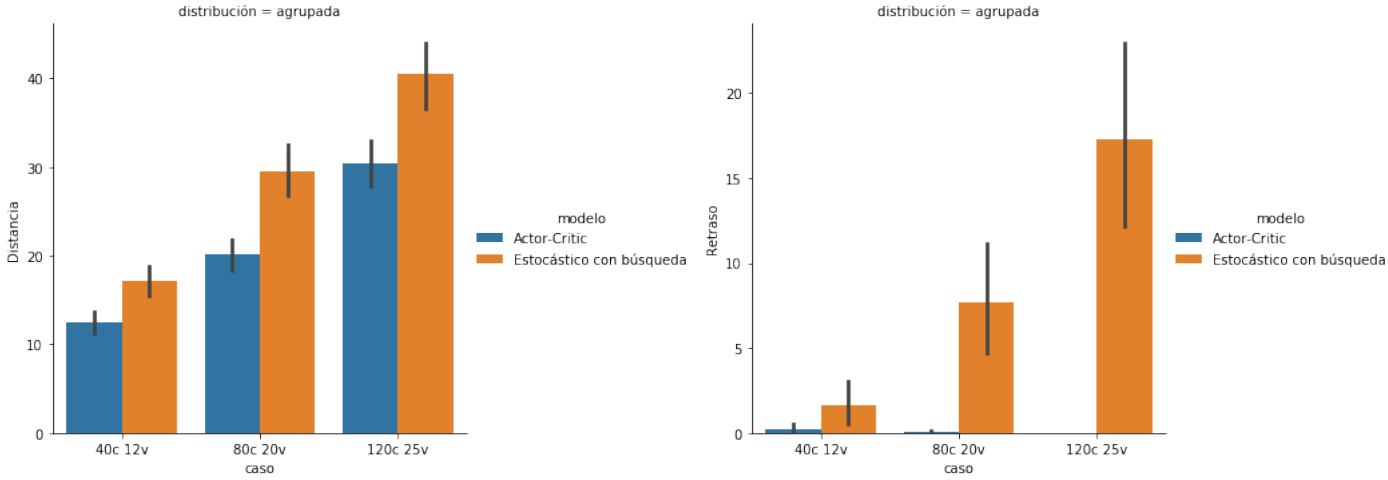

Figura 6.31: Distancia recorrida y porcentaje de retraso modelo estocástico con búsqueda vs modelo actor-critic con estrategia glotona en varios escenarios, usando distancia Manhattan

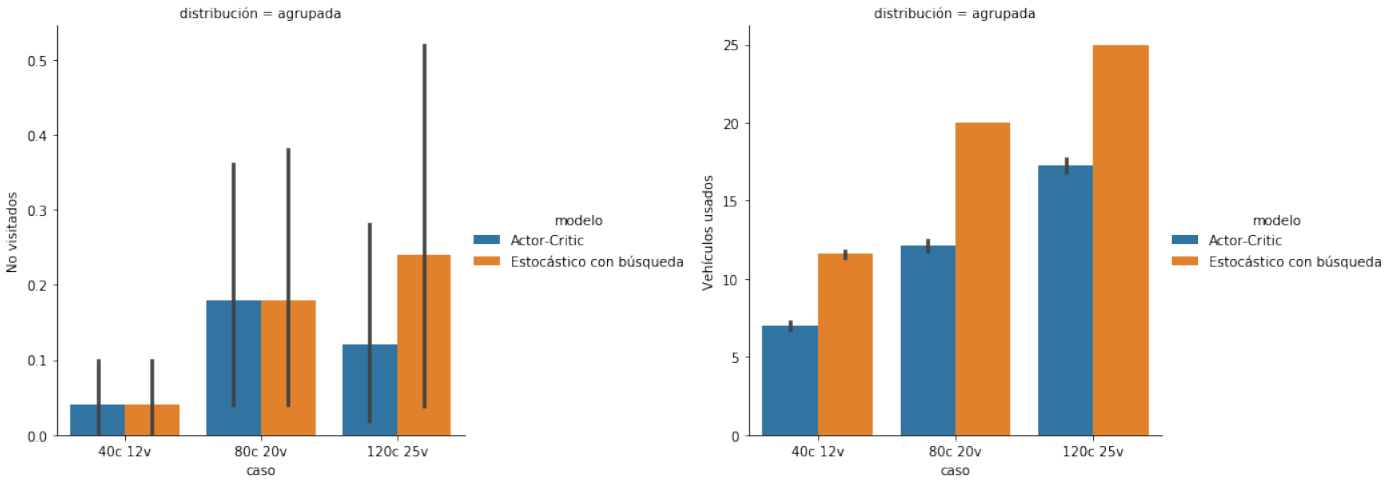

Figura 6.32: Número de clientes no visitados y vehículos usados modelo estocástico con búsqueda vs modelo actor-critic con estrategia glotona en varios escenarios, usando distancia Manhattan

Nuevamente se puede apreciar el el modelo Actor-Critic es superior al estocástico con búsqueda. Tanto en distancia como en porcentaje de retraso el modelo estocástico con búsqueda no logra tener resultados aceptables, mientras que el modelo Actor-Critic incluso alcanza un porcentaje de retraso de casi 0 % en las pruebas realizadas.

Siguiendo la comparación entre los modelos generados de este trabajo, se quiere medir experimentalmente los tiempos promedios de ejecución. Como se puede observar en la siguiente tabla 6.1, aún con todas las modificaciones realizadas al modelo estocástico con búsqueda, el tiempo de ejecución sigue siendo alto e incluso lo hace no escalable para un gran número de clientes. Por otra parte esto hace ver el algoritmo Actor-Critic como el mejor en rendimiento, escalabilidad y tiempo.

	DDQN	Actor-Critic	Estocástico con búsqueda
40 clientes, 12 vehículos	3,53 s	4,47 s	4 minutos
80 clientes, 20 vehículos	8,72 s	8,56 s	13,4 minutos
120 clientes, 25 vehículos	15,6 s	13,23 s	24,1 minutos

Tabla 6.1: Tiempos de ejecución completo de modelos entrenados

Cabe mencionar que el tiempo de ejecución se midió durante toda la ejecución, es decir, todos los periodos de decisión. En el caso de querer obtener el tiempo de ejecución por periodo de decisión promedio, habría que dividir el tiempo por el número de periodos de decisión (que en este caso es 15).

6.6. Comparación con estrategia reoptimizadora

En la sección pasada ya se realizaron pruebas para medir el rendimiento de los modelos, comparándolos entre ellos. Para cuantificar aún mejor qué tan buenos son estos modelos se hace necesario compararlos con otros algoritmos de la literatura. En la sección 2.1.4 ya se habló de algunos de los algoritmos que se han desarrollado en los últimos 30 años. Uno de ellos era conocido como un algoritmo que ocupaba técnicas del problema de ruteo estático y las aplicaba en el problema dinámico. A este algoritmo se le va a llamar estrategia reoptimizadora ya que básicamente aplicaba optimización del problema estático en cada periodo de decisión. Este tipo de algoritmo se compara con el algoritmo Actor-Critic que fue el que tuvo mejores resultados según el análisis visto en la sección pasada. Además, es un algoritmo que tiene sentido compararlo con la estrategia reoptimizadora ya que ambos son algoritmos que no usan información estocástica del medio.

6.6.1. Descripción de la estrategia reoptimizadora

Este tipo de algoritmo, al igual que los algoritmos descritos en esta tesis, divide el intervalo de tiempo de operación de los vehículos en periodos de decisión, que es cuando se toman decisiones sobre el conjunto de clientes conocidos hasta ese punto.

La idea de este algoritmo es resolver en cada periodo de decisión un problema estático sobre todo el conjunto de clientes conocidos y que no han sido atendidos por algún vehículo.

Esta operación entrega inicialmente una ruta preliminar para cada vehículo, sin embargo esta ruta no es totalmente ejecutada, sino que se toma un subconjunto de esta. Esta fracción de la ruta es efectivamente lo que alcanza a completar dentro del tiempo $[t_p, t_{p+1})$, donde t_p y t_{p+1} es el inicio y término del periodo de decisión p.

Un pseudocódigo simple de este procedimiento quedaría descrito como

Algorithm 6: Algoritmo Reoptimizador

Input: $O_0, ..., O_{P-1}$: conjunto de clientes que ingresan al sistema en cada uno de los P periodos de decisión. V_1: conjunto de vehículos con sus capacidades iniciales

$N_0 \leftarrow \varnothing$
$C_0 \leftarrow \varnothing$
for *p=1,...,P* **do**
  $N_p \leftarrow (N_{p-1} \setminus C_{p-1}) \cup O_{p-1}$
  $S_p \leftarrow \texttt{ResolverEstático}(N_p, V_p)$
  $S_p^*, C_p, V_{p+1} \leftarrow \texttt{Despachar}(S_p)$

donde O_p es el conjunto de clientes nuevos que llegan entre el periodo de decisión $p-1$ e p, C_p es el conjunto de clientes que son atendidos en el periodo de decisión p, N_p es el conjunto de clientes totales que son usados para resolver el problema de ruteo estático, V_p es el conjunto de vehículos (y sus capacidades y últimos visitados) en el periodo de decisión p, S_p es la solución estática sobre el conjunto de clientes N_p y el conjunto de vehículos V_p y S_p^* es la ruta efectiva que se realiza en el periodo de decisión p.

La función `ResolverEstático` corresponde a usar cualquier método para resolver el problema estático. En esta tesis se toma la decisión de tomar un "solver" ya existente para resolver este problema. El solver escogido corresponde a la librería OR-tools de Google [14] que permite resolver el VRPTW, y además permitir que los vehículos inicien sus rutas en lugares distintos (necesario para resolver los problemas estáticos cuando los vehículos ya no están en las bodegas).

La función `Despachar` tiene que resolver qué subconjunto de cada ruta debe ser despachado en cada periodo de decisión. Este problema está totalmente relacionado con cuánto tiempo tienen permitido esperar los vehículos en la posición de los clientes, antes de desplazarse al siguiente potencial cliente descrito en la ruta preliminar.

Para resolver cuánto debe esperar cada vehículo y cuáles deben ser los potenciales horarios de salida y llegada, en la próxima sección se describirán dos estrategias de esperas que dirán cuál es el subconjunto de ruta que debe efectivamente realizarse en un periodo de decisión determinado.

6.6.2. Estrategias de espera

Una diferencia entre el problema estático y el problema dinámico es que en el problema dinámico importa si un vehículo espera o no en una ubicación antes de desplazarse al siguiente cliente. En el problema estático, como la ruta nunca cambia, da lo mismo si el vehículo deja su última ubicación lo más pronto posible, o si espera en la posición inicial con tal de que no espere al siguiente cliente, en el caso en que llegue antes de la apertura de la ventana de tiempo del cliente correspondiente. Por otro lado, en el problema dinámico sí puede influir cuánto tiempo el vehículo espera en su ubicación, ya que en el futuro pueden surgir clientes que se encuentren cerca de su sitio y por lo tanto sea más conveniente visitarlos a ellos que otros clientes que pudieron haber estado más lejos.

Esa es una de las razones de por qué se agregó a los agentes de aprendizaje reforzado la acción de poder pasar al siguiente periodo de decisión, como una manera de que el agente pudiera aprender cuándo es conveniente esperar.

En el algoritmo reoptimizador, la espera y las horas de salidas tienen que ser efectuadas mediante heurísticas.

En esta sección se describe cuánto es lo máximo que puede esperar un vehículo, dada una ruta inicial. Es decir, dada una ruta preliminar dado por el algoritmo estático, se describe cuánto puede esperar cada vehículo con tal de ejecutar la ruta sin afectar la factibilidad de esta.

Sin pérdida de generalidad se supone que la posición inicial del vehículo está en la posición 0 y que la ruta entregada por el algoritmo estático del algoritmo reoptimizador está dada por $R = (0, 1, 2, ..., |R|)$. La llegada A_i al cliente i puede darse entre $\underline{A_i}$ como cota mínima y $\overline{A_i}$ como cota máxima, es decir $A_i \in [\underline{A_i}, \overline{A_i}]$. Lo mismo ocurre con la salida D_i que puede estar entre $\underline{D_i}$ como cota mínima y $\overline{D_i}$ como cota máxima, es decir $D_i \in [\underline{D_i}, \overline{D_i}]$.

Suponiendo que el vehículo sale de la posición 0 a tiempo t entonces

$$\underline{A_1} = t + t_{0,1}$$

$$\underline{D_i} = \text{máx}\{e_i, \underline{A_i}\} + st \quad \forall i \in \{1, ..., |R|\}$$
$$\underline{A_i} = \underline{D_{i-1}} + t_{i-1,i} \quad \forall i \in \{2, ..., |R|\}$$

Por otro lado las cotas mayores de llegadas y salidas se calculan como

$$\overline{A_{|R|}} = b_{|R|}$$

$$\overline{D_i} = \overline{A_{i+1}} - t_{i,i+1} \quad \forall i \in \{1, ..., |R| - 1\}$$
$$\overline{A_i} = \text{mín}\{\overline{D_i} - st, l_i\} \quad \forall i \in \{1, ..., |R| - 1\}$$

Relacionando lo descrito con la función `Despachar`, para cada ruta se verifica si el cliente en la posición i de la ruta cumple que $D_{i-1} < t_{k+1}$, con t_{k+1} el límite del periodo de decisión k. Si $D_{i-1} \geq t_{k+1}$ ese cliente podría ser visitado sin problemas si es que la ruta permanece sin cambios en el periodo de decisión siguiente.

En esta tesis se compara la estrategia que se dirige de un cliente a otro lo más pronto posible sin esperar (es decir usando las cotas mínimas $\underline{A_i}$ y $\underline{D_i}$) y la estrategia que espera lo máximo posible (es decir usando las cotas máximas $\overline{A_i}$ y $\overline{D_i}$.

En el caso del algoritmo reoptimizador que utiliza la estrategia de dirigirse al otro cliente lo más pronto posible será llamado algoritmo Reoptimizador-1 y el algoritmo que utiliza la estrategia de esperar lo máximo posible será llamado algoritmo Reoptimizador-2.

Las estrategias de espera pueden ser mejoradas cuándo se cuenta con información estocástica del medio. En la literatura hay estudios de posibles heurísticas que mejoran el rendimiento de los algoritmos cuándo se cuenta con esta información [36] [44]. Sin embargo en esta tesis no se hará ese análisis, por lo que se deja como trabajo futuro.

6.6.3. Comparaciones en diversos escenarios

Como ya se adelantó, la idea de esta sección es comparar el algortimo Actor-Critic con el algoritmo de reoptimización. En el caso de la distancia euclideana, el modelo que se selecciona para competir es el modelo 7 en su versión glotona, que fue el que tuvo mejores resultados en las pruebas experimentales. En el caso de la distancia Manhattan, el obtenido hasta el fin del entrenamiento (que sería el símil del modelo con distancia euclideana) y usando también estrategia glotona es el que se va a usar para comparar. Desde ahora cuando se hable de modelo Actor-Critic, se estará refiriendo a los modelos recién mencionados, dependiendo de la distancia con que se esté midiendo.

A continuación se realiza la comparación en diversos escenarios, los que contemplan variaciones en los grados de dinamismo y la longitud de las ventanas de tiempo. Con un mayor grado de dinamismo, la proporción de clientes dinámicos aumenta y con ventanas de tiempo más pequeñas la optimización se hace más restringida.

En general se hacen pruebas con distancia euclideana y Manhattan usando los siguientes niveles de dinamismo:

- $0,4$
- $0,5$
- $0,6$

Y las siguientes longitudes de ventanas de tiempo:

- Cortas: 15 % de la ventana horaria de la bodega en el caso distancia euclidena y 20 % de la ventana horaria de la bodega en el caso distancia Manhattan
- Amplias: 40 % de la ventana horaria de la bodega en caso distancia euclideana y Manhattan

En el caso de las ventanas de tiempo cortas, se aumentaron un poco en la distancia Manhattan ya que se mostraba demasiado ajustado cuando se usaba 15 %.

Las variables que se miden en las pruebas son: distancia recorrida, % de retraso total respecto a la longitud de la ventana horaria de la bodega, número de clientes promedio no visitados, número de vehículos usados en promedio y fracción de las instancias resueltas sin problemas (en las tablas denotado como "Solución").

Se hacen pruebas en casos de 40 clientes y 12 vehículos; 80 clientes y 20 vehículos; y 120 clientes y 25 vehículos. Por cada caso y cada modelo se efectúan pruebas en 50 instancias que tienen una distribución de clientes uniforme y capacidades de vehículos iguales. Los tiempos de viaje se calculan a partir de la velocidad constante, que en el caso euclideano es de $0,1$ y el caso Manhattan es de $0,25$. Las pruebas se hacen con 15 periodos de decisión y se permiten solicitudes dinámicas hasta la mitad del intervalo de tiempo de operación de los vehículos. El tiempo de servicio se fija en $st = 4$.

Distancia euclideana

Caso	Modelo	Distancia	Retraso	No visitados	Vehículos usados	Solución
40c 12v	Actor-Critic	17.37	3.48	0.0	10.18	1.0
	Reoptimización-1	13.34	0.0	0.0	8.76	0.98
	Reoptimización-2	13.16	0.0	0.0	8.65	0.98
80c 20v	Actor-Critic	29.96	7.12	0.0	16.62	1.0
	Reoptimización-1	22.45	0.0	0.0	14.28	0.94
	Reoptimización-2	22.54	0.0	0.0	14.45	0.98
120c 25v	Actor-Critic	42.09	9.86	0.0	22.8	1.0
	Reoptimización-1	31.24	0.0	0.0	19.8	0.82
	Reoptimización-2	31.09	0.0	0.0	19.92	0.98

Tabla 6.2: Caso grado de dinamismo igual a 0,4 con ventanas cortas usando distancia euclideana

Caso	Modelo	Distancia	Retraso	No visitados	Vehículos usados	Solución
40c 12v	Actor-Critic	16.65	0.0	0.02	8.78	1.0
	Reoptimización-1	12.34	0.0	0.0	8.29	0.98
	Reoptimización-2	12.46	0.0	0.0	8.39	0.98
80c 20v	Actor-Critic	29.5	0.4	0.02	15.12	1.0
	Reoptimización-1	21.01	0.0	0.0	14.07	0.92
	Reoptimización-2	21.05	0.0	0.0	13.9	0.98
120c 25v	Actor-Critic	41.76	0.01	0.02	21.28	1.0
	Reoptimización-1	28.7	0.0	0.0	19.32	0.76
	Reoptimización-2	29.38	0.0	0.0	19.51	0.94

Tabla 6.3: Caso grado de dinamismo igual a 0,4 con ventanas amplias usando distancia euclideana

Caso	Modelo	Distancia	Retraso	No visitados	Vehículos usados	Solución
40c 12v	Actor-Critic	17.35	5.97	0.0	10.18	1.0
	Reoptimización-1	13.38	0.0	0.0	8.8	0.98
	Reoptimización-2	13.27	0.0	0.0	8.67	0.98
80c 20v	Actor-Critic	30.58	4.9	0.0	16.9	1.0
	Reoptimización-1	22.81	0.0	0.0	14.67	0.96
	Reoptimización-2	22.61	0.0	0.0	14.16	0.98
120c 25v	Actor-Critic	42.62	10.57	0.02	22.72	1.0
	Reoptimización-1	30.87	0.0	0.0	19.95	0.8
	Reoptimización-2	30.89	0.0	0.0	20.11	0.9

Tabla 6.4: Caso grado de dinamismo igual a 0,5 con ventanas cortas usando distancia euclideana

Caso	Modelo	Distancia	Retraso	No visitados	Vehículos usados	Solución
40c 12v	Actor-Critic	16.8	1.51	0.02	8.54	1.0
	Reoptimización-1	12.26	0.0	0.0	8.12	0.96
	Reoptimización-2	12.46	0.0	0.0	8.31	0.98
80c 20v	Actor-Critic	29.77	0.0	0.02	14.82	1.0
	Reoptimización-1	21.08	0.0	0.0	14.24	0.92
	Reoptimización-2	21.43	0.0	0.0	13.96	0.98
120c 25v	Actor-Critic	42.26	0.01	0.06	20.9	1.0
	Reoptimización-1	28.36	0.0	0.0	19.53	0.72
	Reoptimización-2	29.16	0.0	0.0	19.64	0.88

Tabla 6.5: Caso grado de dinamismo igual a 0,5 con ventanas amplias usando distancia euclideana

Caso	Modelo	Distancia	Retraso	No visitados	Vehículos usados	Solución
40c 12v	Actor-Critic	17.53	6.42	0.0	10.1	1.0
	Reoptimización-1	13.49	0.0	0.0	8.85	0.96
	Reoptimización-2	13.38	0.0	0.0	8.53	0.98
80c 20v	Actor-Critic	30.88	3.55	0.02	16.9	1.0
	Reoptimización-1	22.46	0.0	0.0	14.57	0.88
	Reoptimización-2	22.71	0.0	0.0	14.33	0.92
120c 25v	Actor-Critic	42.82	8.78	0.04	22.56	1.0
	Reoptimización-1	30.56	0.0	0.0	19.9	0.8
	Reoptimización-2	30.84	0.0	0.0	19.93	0.9

Tabla 6.6: Caso grado de dinamismo igual a 0,6 con ventanas cortas usando distancia euclideana

Caso	Modelo	Distancia	Retraso	No visitados	Vehículos usados	Solución
40c 12v	Actor-Critic	17.4	1.87	0.02	8.6	1.0
	Reoptimización-1	12.65	0.0	0.0	8.46	0.96
	Reoptimización-2	12.65	0.0	0.0	8.35	0.98
80c 20v	Actor-Critic	30.66	0.0	0.1	15.12	1.0
	Reoptimización-1	21.61	0.0	0.0	14.84	0.9
	Reoptimización-2	21.29	0.0	0.0	14.04	0.92
120c 25v	Actor-Critic	42.5	0.36	0.08	20.92	1.0
	Reoptimización-1	28.79	0.0	0.0	20.0	0.74
	Reoptimización-2	29.02	0.0	0.0	19.7	0.88

Tabla 6.7: Caso grado de dinamismo igual a 0,6 con ventanas amplias usando distancia euclideana

Distancia Manhattan

Caso	Modelo	Distancia	Retraso	No visitados	Vehículos usados	Solución
40c 12v	Actor-Critic	19.45	1.05	0.1	7.66	1.0
	Reoptimización-1	16.72	0.0	0.0	8.7	0.92
	Reoptimización-2	16.47	0.0	0.0	8.28	0.92
80c 20v	Actor-Critic	34.5	1.22	0.24	13.18	1.0
	Reoptimización-1	27.57	0.0	0.0	14.55	0.8
	Reoptimización-2	28.07	0.0	0.0	14.48	0.8
120c 25v	Actor-Critic	48.9	1.08	0.32	18.86	1.0
	Reoptimización-1	38.57	0.0	0.0	20.06	0.66
	Reoptimización-2	38.29	0.0	0.0	20.23	0.7

Tabla 6.8: Caso grado de dinamismo igual a 0,4 con ventanas cortas usando distancia Manhattan

Caso	Modelo	Distancia	Retraso	No visitados	Vehículos usados	Solución
40c 12v	Actor-Critic	22.39	1.28	0.1	8.3	1.0
	Reoptimización-1	15.93	0.0	0.0	8.41	0.92
	Reoptimización-2	15.84	0.0	0.0	8.26	0.92
80c 20v	Actor-Critic	38.62	0.41	0.24	13.96	1.0
	Reoptimización-1	27.07	0.0	0.0	14.59	0.78
	Reoptimización-2	26.96	0.0	0.0	14.38	0.8
120c 25v	Actor-Critic	54.68	0.03	0.32	20.14	1.0
	Reoptimización-1	36.34	0.0	0.0	19.93	0.56
	Reoptimización-2	37.26	0.0	0.0	20.21	0.66

Tabla 6.9: Caso grado de dinamismo igual a 0,4 con ventanas amplias usando distancia Manhattan

Caso	Modelo	Distancia	Retraso	No visitados	Vehículos usados	Solución
40c 12v	Actor-Critic	20.11	0.62	0.14	7.88	1.0
	Reoptimización-1	16.34	0.0	0.0	8.27	0.88
	Reoptimización-2	16.46	0.0	0.0	8.42	0.86
80c 20v	Actor-Critic	35.11	0.6	0.2	13.24	1.0
	Reoptimización-1	28.0	0.0	0.0	14.64	0.84
	Reoptimización-2	27.91	0.0	0.0	14.47	0.86
120c 25v	Actor-Critic	49.76	0.81	0.4	18.64	1.0
	Reoptimización-1	39.44	0.0	0.0	20.3	0.6
	Reoptimización-2	39.04	0.0	0.0	20.25	0.64

Tabla 6.10: Caso grado de dinamismo igual a 0,5 con ventanas cortas usando distancia Manhattan

Caso	Modelo	Distancia	Retraso	No visitados	Vehículos usados	Solución
40c 12v	Actor-Critic	23.37	2.44	0.14	8.34	1.0
	Reoptimización-1	15.74	0.0	0.0	8.19	0.84
	Reoptimización-2	15.75	0.0	0.0	8.18	0.88
80c 20v	Actor-Critic	40.11	1.17	0.2	14.06	1.0
	Reoptimización-1	26.63	0.0	0.0	14.58	0.76
	Reoptimización-2	27.26	0.0	0.0	14.47	0.86
120c 25v	Actor-Critic	55.6	0.0	0.4	20.46	1.0
	Reoptimización-1	37.81	0.0	0.0	20.05	0.44
	Reoptimización-2	38.12	0.0	0.0	20.52	0.62

Tabla 6.11: Caso grado de dinamismo igual a 0,5 con ventanas amplias usando distancia Manhattan

Caso	Modelo	Distancia	Retraso	No visitados	Vehículos usados	Solución
40c 12v	Actor-Critic	20.22	0.66	0.18	7.86	1.0
	Reoptimización-1	16.73	0.0	0.0	8.81	0.84
	Reoptimización-2	16.82	0.0	0.0	8.86	0.84
80c 20v	Actor-Critic	35.96	1.13	0.3	13.54	1.0
	Reoptimización-1	27.91	0.0	0.0	14.62	0.74
	Reoptimización-2	27.54	0.0	0.0	14.18	0.76
120c 25v	Actor-Critic	50.1	0.81	0.34	18.82	1.0
	Reoptimización-1	37.51	0.0	0.0	20.64	0.66
	Reoptimización-2	38.31	0.0	0.0	20.47	0.76

Tabla 6.12: Caso grado de dinamismo igual a 0,6 con ventanas cortas usando distancia Manhattan

Caso	Modelo	Distancia	Retraso	No visitados	Vehículos usados	Solución
40c 12v	Actor-Critic	24.3	5.75	0.18	8.58	1.0
	Reoptimización-1	15.89	0.0	0.0	8.52	0.84
	Reoptimización-2	15.77	0.0	0.0	8.31	0.84
80c 20v	Actor-Critic	41.25	0.15	0.3	14.5	1.0
	Reoptimización-1	27.34	0.0	0.0	14.65	0.74
	Reoptimización-2	26.95	0.0	0.0	14.47	0.76
120c 25v	Actor-Critic	57.15	0.0	0.34	20.34	1.0
	Reoptimización-1	36.41	0.0	0.0	20.48	0.5
	Reoptimización-2	36.39	0.0	0.0	20.13	0.62

Tabla 6.13: Caso grado de dinamismo igual a 0,6 con ventanas amplias usando distancia Manhattan

6.6.4. Análisis

Como se puede observar en las tablas mostradas, los modelos de reoptimización superan al modelo Actor-Critic en distancia recorrida. Sin embargo, los modelos de reoptimización nunca alcanzan un 100 % de entrega de soluciones. Esto último ocurre de mayor manera con el modelo reoptimización-1, que tenía una estrategia de esperar lo máximo posible. Se esperaba que al ocupar esta estrategia la distancia recorrida hubiera sido menor, sin embargo no se aprecia mayor diferencia entre el modelo reoptimización-1 y reoptimización-2 en esta variable. Algo que también ocurre es que el los modelos reoptimizadores tienen mejor rendimiento en cuanto a fracción de instancias solucionadas cuando se tienen ventanas de tiempo cortas. En cuanto a los grados de dinamismo, al aumentarlo el porcentaje de instancias no solucionadas aumenta en los modelos reoptimizadores, mientras que en el modelo actor-critic no se ve mayor variación en cuanto a distancia recorrida o retrasos.

Si se compara el rendimiento de los algoritmos entre usar distancia euclideana y Manhattan, se ve un empeoramiento en las instancias no solucionadas usando los modelos reoptimizadores.

En cuanto a los retrasos del modelo Actor-Critic, el modelo tiene mayores problemas obviamente con las ventanas de tiempo más cortas, donde en el peor caso alcanza un $10,57\,\%$

de retraso con respecto a la ventana horaria de la bodega en el caso euclideano. Cuando se usa distancia Manhattan los retrasos mejoran, donde el peor caso aquí ocurre con ventanas amplias pero con un grado de dinamismo de $0,6\,\%$. Si se mide en promedio, los retrasos en el modelo Actor-Critic son menores al $5\,\%$ en el caso euclideano y menores al $7\,\%$ en el caso Manhattan. En específico

	Euclideana	Manhattan
40c 12v	3,2	2,8
80c 20v	2,7	3,5
120c 25v	4,9	6,5

Tabla 6.14: Porcentaje de retrasos modelo Actor-Critic según tipo de distancia y caso de estudio

Por otro lado, en cuanto a distancia recorrida y fracción de instancias resueltas, se tiene una especie de intercambio al comparar los modelos reoptimizadores con el modelo actor-critic. Esto ya que las distancias de los modelos reoptimizadores son menores a las del modelo Actor-Critic, pero estos modelos tienen una menor fracción de instancias resueltas. Esto puede verse mejor en los siguientes gráficos que tienen los resultados promedio

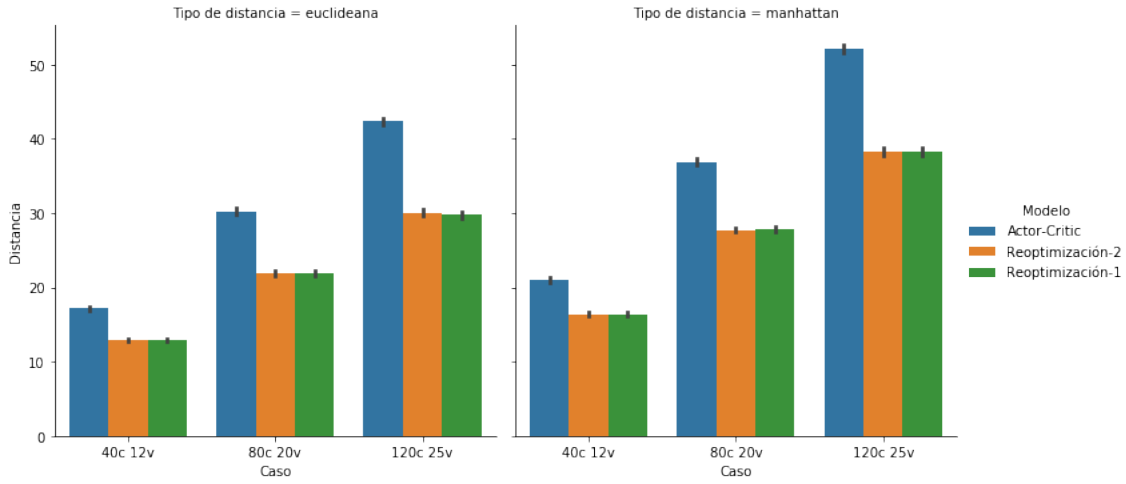

Figura 6.33: Distancias promedio en los modelos comparados para ambas distancias

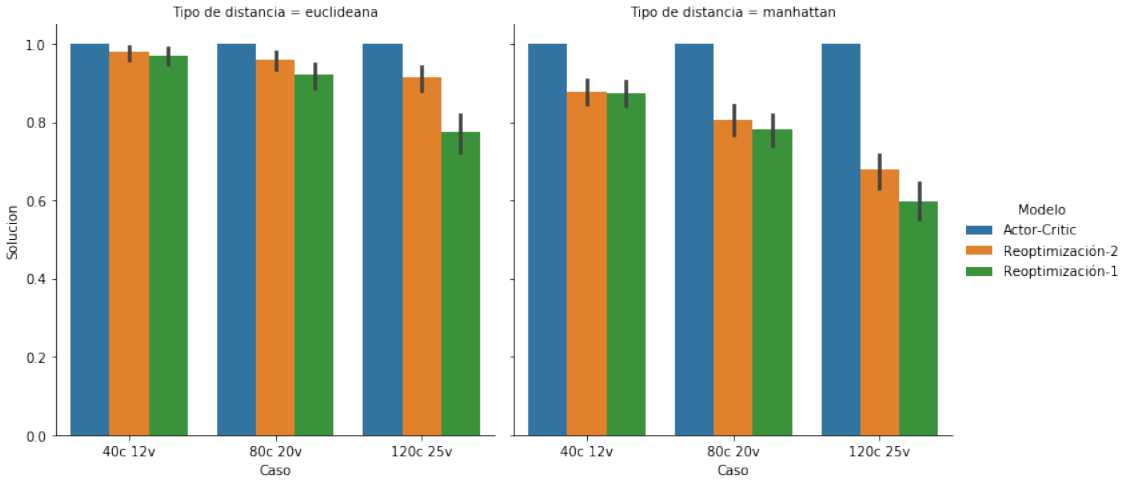

Figura 6.34: Fracción de soluciones promedio en los modelos comparados para ambas distancias

Algo que se puede ver mejor en la figura 6.34 es que el porcentaje de soluciones no resueltas en los modelos reoptimizadores aumenta a medida que aumenta el número de clientes. Más aún, este porcentaje es mayor en el caso de la distancia Manhattan.

6.6.5. Visualización de las soluciones en el plano y tiempos de ejecución

Una forma de poder entender cómo funcionan los algoritmos y el por qué de sus resultados es graficar en el plano las rutas efectuadas en cada periodo de decisión. En la figura 6.35 se muestra cómo son las soluciones del modelo Actor-Critic en los tres periodos de decisión. Como se puede observar ya en estos tres periodos se decide visitar casi todos los clientes. Esto se diferencia de las figuras 6.36 y 6.37 en que las decisiones son más mesuradas y se visitan pocos clientes. Esto se explica por las estrategias de esperas que tienen. Al parecer este comportamiento es el que hace que las soluciones promedio de ambos modelos tengan menor distancia recorrida que el modelo Actor-Critic, ya que los modelos al esperar más, pueden realizar una optimización sobre un mayor número de clientes, y por consiguiente les da la posibilidad de mejora las distancias recorridas de los vehículos. Por otro lado, el modelo Reoptimización-1 con su estrategia de espera agresiva tiene una mayor propensión a que sus soluciones no sean factibles, por la llegada de nuevos clientes. Esto igual pasa con el modelo Reoptimización-2 pero con menos frecuencia.

La forma de ver las siguientes figuras es la siguiente. Los nodos representan los clientes y los arcos las rutas de los vehículos. Los nodos de color gris corresponden a clientes que no han sido visitado hasta ese periodo de decisión, mientras que los nodos y los arcos que tienen el mismo color representan los clientes visitados en una ruta por un vehículo. Cada vehículo tiene representado un color distintivo. Todas las rutas salen de un único nodo de color rojo que representa la bodega.

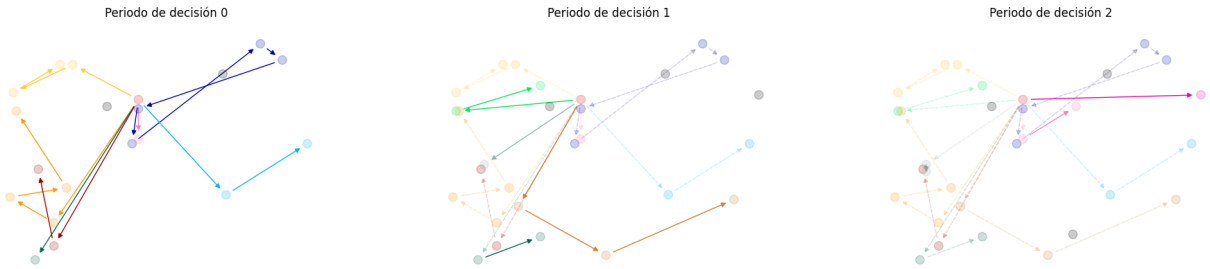

Figura 6.35: Primeros tres periodos de decisión para una solución del modelo Actor-Critic

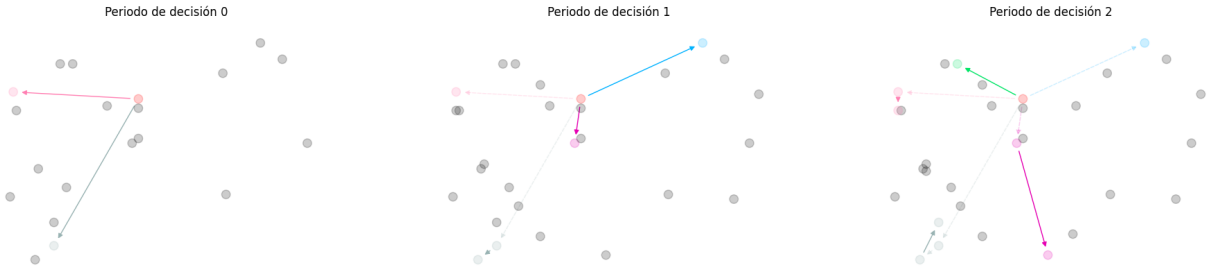

Figura 6.36: Primeros tres periodos de decisión para una solución del modelo Reoptimización-1

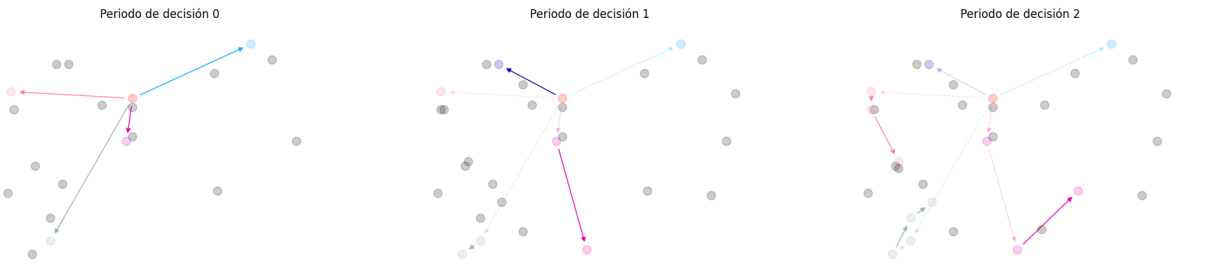

Figura 6.37: Primeros tres periodos de decisión para una solución del modelo Reoptimización-2

Ahora, si se quiere comparar los tiempos de ejecución medidos en segundos, el modelo Reoptimización-2 es el más rápido en los tres casos. El modelo Actor-Critic es el peor en promedio en los casos de 40 clientes, 12 vehículos y 80 clientes y 20 vehículos. Sin embargo, en la instancia de 120 clientes y 25 vehículos el algoritmo Actor-Critic sobrepasa al modelo Reoptimización-1 y está más cerca del modelo Reoptimización-2. Una ventaja del modelo Actor-Critic es la poca variabilidad (o varianza) en su tiempo de ejecución en cada caso. Otra ventaja del modelo Actor-Critic es que muestra un comportamiento lineal de los tiempos de ejecución, en función del número de clientes. Las estrategias reoptimizadoras se alejan de este comportamiento, acercándose a tiempos de ejecución con un orden mayor, aunque se necesitarían más pruebas con instancias de mayor tamaño para calcular el orden preciso del crecimiento de los tiempos de ejecución de estas estrategias. En este sentido el algoritmo

Actor-Critic se posiciona como una buena estrategia que permite ser escalable a instancias de mayor tamaño.

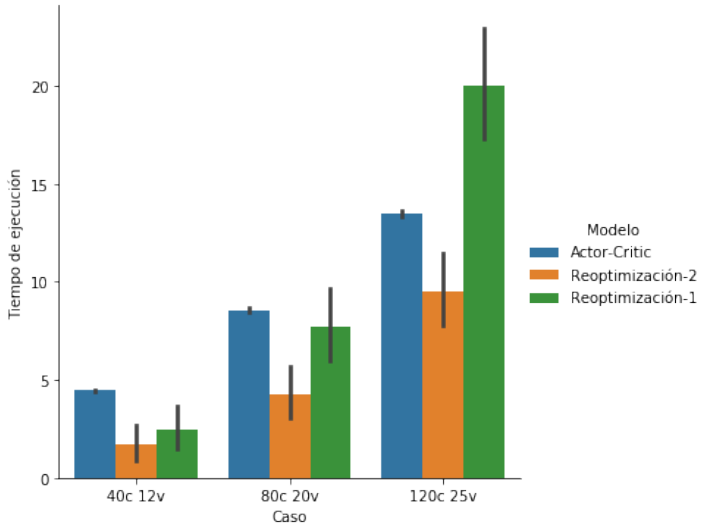

Figura 6.38: Tiempos de ejecución Actor-Critic vs Reoptimización

6.7. Comparación con soluciones óptimas

En esta última sección la idea es comparar el modelo Actor-Critic con soluciones óptimas del problema estático. Esto quiere decir que la misma instancia que se le entrega al modelo Actor-Critic dividida por cada periodo de decisión, se le entrega a un "solver" pero de manera completa, es decir, tiene toda la información que puede aparecer en el futuro. El "solver" de optimización que se utiliza es Gurobi [15] que permite realizar optimización entera mixta. La formulación matemática que se ocupa es la planteada en la sección 3.2. Por otro lado también se agregan las estrategias reoptimizadoras para dar otro nivel de comparación.

La prueba que se hace es con instancias pequeñas de 15 clientes y 4 vehículos, para que resolver el problema de forma óptima demore poco tiempo (en este caso siempre fue menos de 5 minutos). Como se puede ver en la figura 6.39, obviamente la solución óptima del problema estático es la que tiene menos distancia recorrida (ya que posee toda la información de los clientes disponibles). A diferencia de los resultados de la sección pasada, aquí como se prueban los modelos en instancias pequeñas, las estrategias reoptimizadoras tienen buen rendimiento en cuanto a distancia recorrida y porcentaje de instancias no resueltas (aquí en promedio resolvió un 95 % correctamente).

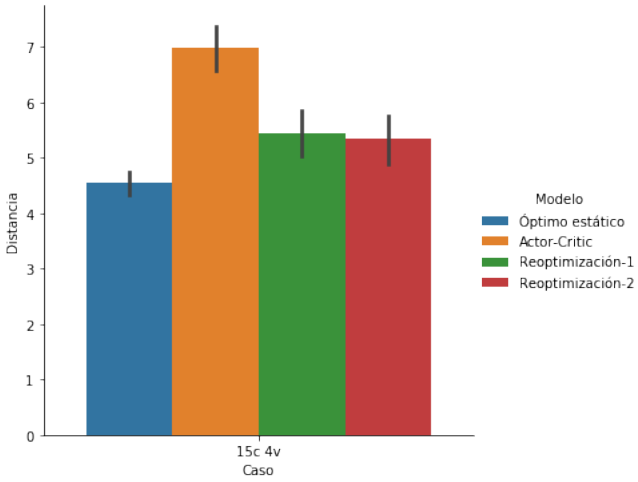

Figura 6.39: Distancias euclideanas modelo actor-critic y reoptimizadores vs solución óptima del problema estático

Ahora si se quiere por ejemplo observar cómo cambian las soluciones dibujadas en el plano, en las siguientes figuras se muestra la diferencia de las rutas entre la solución óptima, la solución del modelo Actor-Critic y la solución del modelo Reoptimizador-2.

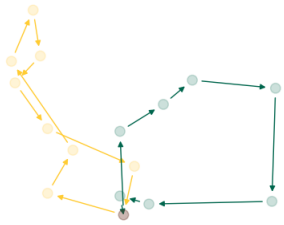

Figura 6.40: Solución óptima problema estático

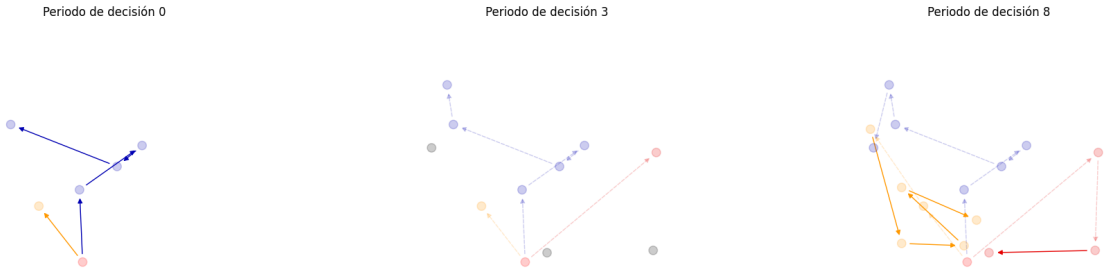

Figura 6.41: Solución modelo Actor-Critic: como ya se vio, rápidamente se seleccionan clientes a visitar en el primer periodo de decisión. Más tarde igual decide pasar clientes para visitarlos después

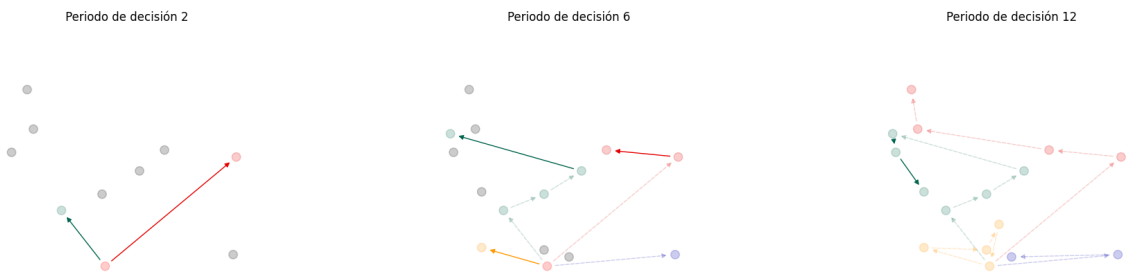

Figura 6.42: Solución modelo Reoptimizador-1: el modelo espera hasta el tercer periodo de decisión para efectuar su primer movimiento. Las selecciones de los últimos clientes las deja para la mitad y el final del horario de operación

Se puede notar que las soluciones del problema dinámico se diferencian bastante de las soluciones óptimas del problema estático.

Conclusiones

En el presente trabajo se han mostrado tres algoritmos de aprendizaje reforzado para resolver el problema de ruteo de vehículos dinámico. Es preciso recordar que haber estudiado este problema enfocado en situaciones no planificadas, sirve como base para enfrentar dificultades logísticas que se pueden encontrar en las industrias, tales como mejorar los tiempos de entrega y permitir los despachos para el mismo día. Por otro lado, el producto final permite mejorar la experiencia de usuario de los clientes, al tener opciones de despacho más personalizadas y con una promesa de tiempos de entrega más confiable. En este sentido, haber estudiado las variantes de este problema y las estrategias que se usan para enfrentarlo, generan un gran conocimiento práctico para los casos de uso de Entel Ocean.

En cuanto al desempeño de los algoritmos desarrollados, se puede decir que el algoritmo DDQN con experiencias priorizadas fue el que tuvo peor funcionamiento, dado que en los entrenamientos que se hicieron las recompensas nunca alcanzaron una correcta estabilidad, lo que impidió tener una función acción-valor que se acercara a la óptima. Una forma que podría mejorar los entrenamientos es usar una función acción-valor normalizada de tal manera que no sean tan grandes los errores numéricos. Sin embargo para aplicar esta solución se tendrían que fijar valores máximos y mínimos a priori de las recompensas obtenidas por el agente.

Por el lado del algoritmo Actor-Critic, se puede concluir que fue el que tuvo mejor desempeño tanto en entrenamiento como en las pruebas experimentales que se le hicieron. Este algoritmo logró aprender las situaciones que se le impusieron, como la minimización de la distancia recorrida, la minimización del porcentaje de retraso y la minimización de clientes no visitados. Como trabajo futuro sería valioso disminuir los tiempos de entrenamiento, para que sea factible además incluir instancias de mayor tamaño en el generador de datos.

El algoritmo que utilizaba información estocástica, a pesar de que tuvo un rendimiento razonable, el tiempo de ejecución fue demasiado alto, haciéndolo no aplicable para ponerlo en producción por ejemplo. El algoritmo Actor-Critic fue superior a este algoritmo, a pesar de ser un algoritmo más simple y con menos conocimiento de su ambiente. Como trabajo futuro sería una buena idea darle un nuevo enfoque al algoritmo estocástico y explotar de mayor manera la información estocástica del medio, aprovechando otras variables como el tiempo de solicitud de los clientes, que pueden ser claves en contextos más aplicados.

El algoritmo Actor-Critic de igual manera tuvo buenos resultados al compararlo con una estrategia del estado del arte que usa reoptimización del problema estático. Las pruebas realizadas con los modelos del algoritmo Actor-Critic, a pesar de que tuvieron mayor distancia recorrida, se mostraron más robustos que la comparación, especialmente cuando se estaba

usando la distancia Manhattan. De todas maneras, dependiendo del caso de uso de estos modelos, podría ser más conveniente usar uno o otro. Por ejemplo se podría tener una estrategia de solución mixta que use el modelo reoptimizador en instancias más pequeñas y que se use el modelo Actor-Critic para instancias más grandes.

Para finalizar, quedan dudas de cómo funcionarían estos modelos aplicados a ambientes más realistas como en una ciudad. Sería interesante aplicar estos modelos a una flota de vehículos mixta que contenga autos, camiones, bicicletas, drones y otros medios de transporte. Esto implicaría entrenar el algoritmo con vehículos con capacidades heterogéneas. Por otro lado, sería interesante ver si los modelos entrenados con distancia Manhattan serían homologables en ciudades o en otros casos aplicados. Obviamente estos modelos no tienen incorporados los factores de tráfico de las calles, por lo que añadir estas variables a los entrenamientos sería una buena propuesta para complementar lo realizado en este trabajo.

Bibliografía

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Amazon prime air. `https://www.amazon.com/Amazon-Prime-Air/b?ie=UTF8&node=8037720011`. Accedido el 7-10-2020.

[3] Diego Artigas. El desarrollo del "same day delivery" en chile. `http://cache-elastic.emol.com/2019/03/15/B/SI3IHHVB/all`, 2019. Accedido el 16-09-2020.

[4] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

[5] Russell Bent and Pascal Van Hentenryck. Dynamic vehicle routing with stochastic requests. In *IJCAI*, pages 1362–1363, 2003.

[6] Dimitris Bertsimas, Patrick Jaillet, and Sébastien Martin. Online vehicle routing: The edge of optimization in large-scale applications. *Operations Research*, 67(1):143–162, 2019.

[7] Xinwei Chen, Marlin W Ulmer, and Barrett W Thomas. Deep q-learning for same-day delivery with a heterogeneous fleet of vehicles and drones. *arXiv preprint arXiv:1910.11901*, 2019.

[8] Zhi-Long Chen and Hang Xu. Dynamic column generation for dynamic vehicle routing with time windows. *Transportation Science*, 40(1):74–88, 2006.

[9] Thomas Degris, Patrick M Pilarski, and Richard S Sutton. Model-free reinforcement learning with continuous action in practice. In *2012 American Control Conference (ACC)*, pages 2177–2182. IEEE, 2012.

[10] Michel Deudon, Pierre Cournut, Alexandre Lacoste, Yossiri Adulyasak, and Louis-Martin Rousseau. Learning heuristics for the tsp by policy gradient. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 170–181. Springer, 2018.

[11] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *arXiv preprint arXiv:1811.12560*, 2018.

[12] Michel Gendreau, Francois Guertin, Jean-Yves Potvin, and Eric Taillard. Parallel tabu search for real-time vehicle routing and dispatching. *Transportation science*, 33(4):381–390, 1999.

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[14] Google. Or-tools, google optimization tools. `https://developers.google.com/optimization/routing`, 2020. Accedido el 21-09-2020.

[15] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2020.

[16] Martin Joerss, Jürgen Schröder, Florian Neuhaus, Christoph Klink, and Florian Mann. Parcel delivery: The future of last mile. *McKinsey & Company*, 2016.

[17] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, pages 6348–6358, 2017.

[18] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[19] Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*, 2018.

[20] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

[21] Jens Lysgaard, Adam N Letchford, and Richard W Eglese. A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming*, 100(2):423–445, 2004.

[22] A Rupam Mahmood, Hado P van Hasselt, and Richard S Sutton. Weighted importance sampling for off-policy learning with linear function approximation. In *Advances in Neural Information Processing Systems*, pages 3014–3022, 2014.

[23] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[24] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[25] Michał Okulewicz and Jacek Mańdziuk. The impact of particular components of the pso-based algorithm solving the dynamic vehicle routing problem. *Applied Soft Computing*, 58:586–604, 2017.

[26] Victor Pillac, Michel Gendreau, Christelle Guéret, and Andrés L Medaglia. A review of dynamic vehicle routing problems. *European Journal of Operational Research*, 225(1):1–11, 2013.

[27] Harilaos N Psaraftis, Min Wen, and Christos A Kontovas. Dynamic vehicle routing problems: Three decades and counting. *Networks*, 67(1):3–31, 2016.

[28] Martin L Puterman. *Markov Decision Processes.: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2014.

[29] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[30] Entel S.A. Entel secure cloud. `https://www.entel.cl/empresas/secure-cloud/`. Accedido el 21-09-2020.

[31] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[32] Verena Schmid. Solving the dynamic ambulance relocation and dispatching problem using approximate dynamic programming. *European journal of operational research*, 219(3):611–621, 2012.

[33] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[34] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[35] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[36] Barrett W Thomas. Waiting strategies for anticipating service requests from known customer locations. *Transportation Science*, 41(3):319–331, 2007.

[37] L. Thomas. Target expands same-day shipping option in the latest move in the delivery wars with walmart and amazon. `https://www.cnbc.com/2019/06/12/target-expands-same-day-delivery-option-in-battle-with-walmart-amazon.html`, 2019. Accedido el 16-09-2020.

[38] Paolo Toth and Daniele Vigo. *Vehicle routing: problems, methods, and applications*. SIAM, 2014.

[39] Uc berkeley deep reinforcement learning course cs 285. `http://rail.eecs.berkeley.edu/deeprlcourse/`. Accedido el 30-07-2020.

[40] Usuaria-UXAlliance. Delivery apps en tiempos de covid-19. `https://usaria.mx/global-research/`, 2020. Accedido el 08-10-2020.

[41] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

[42] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[43] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, et al. Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017.

[44] Stacy A Voccia, Ann Melissa Campbell, and Barrett W Thomas. The same-day delivery problem for online purchases. *Transportation Science*, 53(1):167–184, 2019.

[45] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[46] Jian Yang, Patrick Jaillet, and Hani Mahmassani. Real-time multivehicle truckload pickup and delivery problems. *Transportation Science*, 38(2):135–148, 2004.

Apéndice A

Descripción detallada de redes neuronales usadas

A.1. Funciones de activación usadas

- $$\begin{aligned} ReLu\colon \mathbb{R} &\longrightarrow \mathbb{R}_+ \\ x &\longmapsto \text{máx}\{x, 0\} \end{aligned}$$

- $$\begin{aligned} tanh\colon \mathbb{R} &\longrightarrow (-1, 1) \\ x &\longmapsto \frac{e^x - e^{-x}}{e^x + e^{-x}} \end{aligned}$$

- $$\begin{aligned} softmax\colon \mathbb{R}^N &\longrightarrow (0, 1)^N \\ softmax(x)_j &= \frac{e^{x_j}}{\sum_{i=1}^{N} e^{x_i}} \end{aligned}$$

A.2. Redes neuronales preliminares

A.2.1. Atención y multi atención aplicada a grafos

Sea $G = (V, A)$ grafo dirigido, donde $|V| = n$. Se considera que cada nodo del grafo posee características $h_i \in \mathbb{R}^{d_h}$, $i \in \{1, ...n\}$. Por otro lado, cada arco $(i, j) \in A$ posee un peso $w_{i,j} \in \mathbb{R}$.

Sean d_l y d_v dimensiones. Sean $l_i \in \mathbb{R}^{d_l}$, $v_i \in \mathbb{R}^{d_v}$ y $q_i \in \mathbb{R}^{d_l}$ definidos como

$$q_i = W^Q h_i, \quad l_i = W^L h_i, \quad v_i = W^V h_i$$

Con $W^Q, W^L \in \mathbb{R}^{d_l \times d_h}, W^V \in \mathbb{R}^{d_v \times d_h}$ parámetros. Se define $u_{ij} \in \mathbb{R}$ como

$$u_{ij} = \begin{cases} \frac{q_i^T l_j + w_{ij}}{\sqrt{d_l}} & \text{si el arco (i, j) existe} \\ -\infty & \text{sino} \end{cases} \tag{A.1}$$

A partir de esto se calculan los pesos de atención $a_{ij} \in [0, 1]$ como

$$a_{ij} = \frac{e^{u_{ij}}}{\sum_{j'} e^{u_{ij'}}} \tag{A.2}$$

Finalmente, para cada $i \in \{1, ...n\}$

$$h'_i = \sum_j a_{ij} v_j \tag{A.3}$$

Lo anterior se conoce como mecanismo de atención. Si es que se quiere paralelizar el proceso anterior (y aprovechar las capacidades de una GPU) se puede subdividir cada uno de los vectores h_i en M vectores $h_{im} \in \mathbb{R}^{\frac{d_h}{M}}$, $m \in \{1, ..., M\}$. Para $W = (w_{ij})$, $i, j \in \{1, ..., n\}$, matriz de pesos y denotando por h'_{im} el resultado de realizar el proceso anterior para cada $i \in \{1, ...n\}$ y $m \in \{1, ..., M\}$, se define para cada $i \in \{1, ...n\}$, la siguiente operación:

$$MHA_i(h_1, ..., h_n, W) = \text{Concat}(h'_{i1}, ..., h'_{iM}) W^O \tag{A.4}$$

Con $W^O \in \mathbb{R}^{h \cdot d_v \times d_h}$ parámetro. Lo anterior comúnmente se conoce como atención multi-cabezal [42].

A.2.2. Capa FeedForward

Sea $W^{ff,0} \in \mathbb{R}^{d_f \times d_h}$, $W^{ff,1} \in \mathbb{R}^{d_h \times d_f}$, $b^{ff,0} \in \mathbb{R}^{d_f}$ y $b^{ff,1} \in \mathbb{R}^{d_h}$ parámetros. Luego

$$FF(\hat{h_i}) = W^{ff,1} \cdot ReLu(W^{ff,0} \hat{h_i} + b^{ff,0}) + b^{ff,1}$$

A.2.3. Normalización por batch

Se considera un batch de tamaño B para los elementos $h_i^{(k)}$, $i \in \{1, ..., n\}$, $k \in \{1, ..., B\}$. La idea es normalizar el batch para cada elemento del vector d_h dimensional $h_i^{(k)}$. Primero se calcula la media y la varianza del batch

$$\mu_B = \frac{1}{B} \sum_{k=1}^{B} h_i^{(k)}, \quad \sigma_B^2 = \frac{1}{B} \sum_{k=1}^{B} (h_i^{(k)} - \mu_B)$$

Luego el elemento normalizado correspondería a

$$\hat{h_i}^{(k)} = \frac{h_i^{(k)} - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}$$

Finalmente la normalización por batch queda completa al considerar $w^{\text{bn}} \in \mathbb{R}^{d_h}$ y $b^{\text{bn}} \in \mathbb{R}^{d_h}$ parámetros.

$$BN(h_i^{(k)}) = w^{\text{bn}} \odot \hat{h_i}^{(k)} + b^{\text{bn}}$$

Donde $\odot$ corresponde a una multiplicación punto a punto.

A.2.4. Atención con "glimpses"

Una arquitectura que permite generar atención ponderada sobre un conjunto de entradas $x_1, x_2, ..., x_n$ con $x_i \in \mathbb{R}^{d_h}$ se definen como "glimpses", usadas en [4] y [10] para la arquitectura de la red neuronal de la función valor del algoritmo actor-critic.

Sean $W_g \in \mathbb{R}^{d_h \times d_h}$ y $v_g \in \mathbb{R}^{d_h}$ parámetros. Se calculan puntajes s_i, $i \in \{1, ..., n\}$ como

$$s_i = v_g^T tanh(x_i W_g)$$

Estos puntajes son normalizados mediante

$$a_i = \frac{e^{s_i}}{\sum_{i=1}^{n} e^{s_i}}$$

Luego se define la atención ponderada como la suma ponderada entre las entradas y sus puntajes normalizados

$$g = glimpse(x_1, x_2, ..., x_n) = \sum_{i=1}^{n} x_i a_i$$

A.3. Red codificadora

Sea $G = (V, A)$ grafo dirigido, donde $|V| = n$. Se considera que este grafo es una instancia del problema de vehículos dinámico, por lo que cada nodo del grafo posee características $s_i^1 = (x_i, y_i, q_i, e_i, l_i, ct_i) \in \mathbb{R}^6$, $i \in \{1, ...n\}$, donde x_i es la posición en el eje x, y_i la posición en el eje y, q_i la demanda, e_i inicio de la ventana de tiempo, l_i fin de la ventana de tiempo y ct_i la hora de solicitud. Por otro lado, para cada arco $(i, j) \in A$ se tiene el tiempo de viaje $T_{ij} \in \mathbb{R}$ entre el nodo i y el nodo j. Se va a denotar por T la matriz correspondiente a los tiempos de viaje.

Primero se tiene una capa de "embedding" que aumenta la dimensionalidad de 6 a d_h. Con los parámetros $W \in \mathbb{R}^{6\times d_h}$, $b \in \mathbb{R}^{d_h}$, y también los relativos a la normalización por batch, se hace la siguiente operación.

$$h_i^{(0)} = BN(x_i W + b)$$

A continuación se hace un proceso sucesivo que combina las redes preliminares que se definieron. Sea $k \in \{1, ..., N\}$, luego

$$\hat{h_i} = BN(h_i^{(k-1)} + MHA_i^l(h_1^{(k-1)}, ..., h_n^{(k-1)}, W))$$
$$h_i^{(k)} = BN(\hat{h_i} + FF(\hat{h_i}))$$

A.4. Red decodificadora

En esta sección se describe cómo se entregan las salidas para cada red neuronal usada en cada algoritmo.

Antes que todo, para codificar de mejor manera los vehículos que se tienen disponibles, se va a definir un vector de contexto para cada vehículo $k \in \{1, ..., m\}$, con m el número de vehículos.

$$h_{(c)_k}^{(N)} = [\bar{h}^{(N)}, h_{LV_k}^{(N)}, h_0^{(N)}, Q_k, IT_k] \tag{A.5}$$

Donde $h_i^{(N)}$ el resultado de la red codificadora, para cada nodo $i = 1, .., n$, Q_k, IT_k y LV_k es la capacidad, la hora de disponibilidad y el último cliente visitado del vehículo k, respectivamente. Cabe notar que si el vehículo k está estacionado en la bodega $LV_k = 0$. Por otro lado

$$\bar{h}^{(N)} = \frac{1}{n}\sum_{i=1}^{n} h_i^{(N)}$$

El cual actúa como una especie de resumen del resultado de la red codificadora. El tercer término actúa como un "recuerdo" de la codificación de la bodega.

Lo primero que se hace es usar el vector de contexto junto a la codificación usando mecanismos de atención. Para esto se hace algo parecido a lo realizado en la red de codificación.

Sean $\hat{W^Q} \in \mathbb{R}^{d_l \times 3d_h+2}$, $\hat{W^L} \in \mathbb{R}^{d_l \times d_h}, W^V \in \mathbb{R}^{d_v \times d_h}$ parámetros. Luego

$$q_{(c)_k} = \hat{W^Q} h_{(c)_k}^{(N)}, \quad l_i = \hat{W^L} h_i^{(N)}, \quad v_i = \hat{W^V} h_i^{(N)}$$

Se define $u_{(c)_{k,i}} \in \mathbb{R}$ como

$$u_{(c)_{k,i}} = \begin{cases} \frac{q_{(c)_k}^T l_i}{\sqrt{d_k}} & \text{si es factible añadir } i \text{ a la ruta del vehículo } k \\ -\infty & \text{sino} \end{cases} \tag{A.6}$$

A partir de esto se calculan los pesos de atención $a_{k,i} \in [0,1]$ y $h_{(c)_k}^{(N+1)}$ como

$$a_{k,i} = \frac{e^{u_{(c)_{k,i}}}}{\sum_i e^{u_{(c)_{k,i}}}}, \quad h_{(c)_k}^{(N+1)\prime} = \sum_i a_{k,i} v_i \tag{A.7}$$

Finalmente para $\hat{W^O} \in \mathbb{R}^{h \cdot d_v \times d_h}$

$$h_{(c)_k}^{(N+1)} = h_{(c)_k}^{(N+1)\prime} \hat{W^O}$$

A.4.1. Red DQN

Se denotará por red DQN aquella que modela la función acción-valor $Q_\theta(s,a)$ usada en el contexto de los algoritmos Q-learning. La idea es que la red neuronal para cada entrada retorne un vector del largo del espacio de acciones.

La idea de esta red es calcular puntajes entre cada una de las asignaciones nodo-vehículo, además de generar una codificación para la acción de pasar al siguiente periodo de decisión.

Sean $W_1, W_2 \in \mathbb{R}^{d_h \times d_h}$, $v_1 \in \mathbb{R}^{d_h}$. Luego

$$r_{i,k} = v_1^T tanh(h_i^{(N)} W_1 + h_{(c)_k}^{(N+1)} W_2)$$

Finalmente se multiplican esos puntajes sobre la concatenación entre el vector de contexto y la codificación del nodo.

$h_{i,k} = ([h_i^{(N)}, h_{(c)_k}^{(N+1)}] r_{i,k}) W_3$

Con $W_3 \in \mathbb{R}^{2d_h \times d_h}$.

Finalmente el q-valor de cada acción quedaría explicitado como

$$Q(s,(i,k)) = v_2^T ReLu(h_{i,k} W_4)$$

$$Q(s, pass) = v_2^T ReLu(\sum_{k=1}^{m} h_{(c)_k} W_4)$$

Con $W_4 \in \mathbb{R}^{d_h \times d_h}$, $v_2 \in \mathbb{R}^{d_h}$.

A.4.2. Decodificador con atención

La red decodificadora con atención se va a usar en el contexto de el algoritmo actor-critic así como el algoritmo que utiliza información estocástica. Esta red, dado el vector de contexto y la codificación de los nodos, va a retornar una distribución de probabilidad sobre el espacio de acciones. La idea es nuevamente usar atención sobre el vector de contexto y la codificación.

Sean $\tilde{W^Q}$, $\tilde{W^L} \in \mathbb{R}^{d_h \times d_h}$. Luego

$$q_{(c)_k} = \tilde{W^Q} h^{(N)}_{(c)_k}, \quad l_i = \tilde{W^L} h^{(N)}_i$$

Para C hiperparámetro, define $u_{(c)_{k,i}} \in [-C, C]$ como

$$u_{(c)_{k,i}} = \begin{cases} C \cdot tanh(\frac{q^T_{(c)_k} l_i}{\sqrt{d_h}}) & \text{si es factible añadir } i \text{ a la ruta del vehículo } k \\ -\infty & \text{sino} \end{cases} \tag{A.8}$$

Ahora para la acción pasar se hace lo siguiente

$$u_{pass} = C \cdot tanh(v_p^T \sum_{k=1}^{m} h^{(N+1)}_{(c)_k} W_p)$$

Para $W_p \in \mathbb{R}^{d_h \times d_h}$ y $v_p \in \mathbb{R}^{d_h}$. Finalmente las probabilidades se calculan como

$$p_{i,k} = \frac{e^{u_{(c)_{k,i}}}}{\sum_{m,j} e^{u_{(c)_{m,j}}} + e^{u_{pass}}}$$

$$p_{pass} = \frac{e^{u_{pass}}}{\sum_{m,j} e^{u_{(c)_{m,j}}} + e^{u_{pass}}}$$

A.5. Red Critic

Esta red neuronal se ocupa en el algoritmo actor-critic para estimar la función valor $V_\phi(s)$.

Lo primero que se hace es usar la red codificadora para obtener una representación de cada uno de los nodos del grafo $h^{(N)}_i$. Esto se hace con parámetros distintos a los usados para calcular la distribución de probabilidad sobre las acciones.

En cuanto a la representación de los vehículos sólo se usa la capacidad Q_k y el tiempo de disponibilidad IT_k de los vehículos $k \in \{1, ..., m\}$.

Estas entradas pasan por una primera capa de "embedding" que aumenta su dimensionalidad. Para $W_v \in \mathbb{R}^{2 \times d_h}$ y $b_v \in \mathbb{R}^{d_h}$

$$h_k = BN([Q_k, IT_k]W_v + b_v)$$

Para calcular que tanto aportan cada uno de las entradas, y qué tanta atención se les debe dar, se utiliza atención mediante "glimpses"

$$g_1 = glimpse(h_1^{(N)}, ..., h_n^{(N)})$$
$$g_2 = glimpse(h_1, ..., h_m)$$

Con g_1 y $g_2 \in \mathbb{R}^{d_h}$. Para finalizar, se hace una especie de regresión mediante los parámetros $W_1, W_2 \in \mathbb{R}^{d_h \times d_c}$, $w_1, w_2 \in \mathbb{R}^{d_c}$ para calcular la función valor

$$h_1 = ReLu(g_1 W_1) \quad h_2 = ReLu(g_2 W_2)$$
$$V(s) = w_1^T h_1 + w_2^T h_2 + b$$

Con $b \in \mathbb{R}$ un hiperparámetro.