Acercamiento al *Traveler Purchaser Problem* con *Q-Learning*

Tesis de pregrado

Departamento de Ingeniería Industrial

Universidad de los Andes

Juan Manuel Betancourt

Asesores:

Daniel Cuellar M.Sc.

Camilo Gómez Ph.D.

Fecha: 10/12/2021

Abstract

Los problemas de optimización combinatoria han sido ampliamente estudiados en el campo de investigación de operaciones por su alto impacto en el sector industrial. En los últimos años, ha crecido el interés por parte de la comunidad investigativa de resolver estos problemas con enfoques basados en Reinforcement Learning. Este interés se debe a la forma en que se modela la información del sistema y cómo, a través de la extracción de características, se genera la toma de decisiones. Por esta razón, en este trabajo se modela y resuelve el problema del Traveler Purchaser Problem (TPP) utilizando el algoritmo de Q-Learning. Este algoritmo, por medio de un ambiente y un agente, encuentra la mejor secuencia de decisiones para una instancia dada del problema. Para validar su desempeño, se realiza un estudio computacional utilizando instancias clásicas de la literatura y se comparan los resultados con la mejor metodología de solución reportada. A continuación, Se presentan ventajas y desventajas del enfoque de solución propuesto y se discuten variantes del TPP que podrían ser modeladas y resueltas a traves del algoritmo Q-learning. Por último, se presentan algunas conclusiones.

Palabras clave: Investigación de operaciones, Reinforcement Learning, Traveler Purchaser Problem, Q-Learning

1. Introducción

Los problemas de optimización combinatoria tienen una gran aplicabilidad en el sector industrial debido al gran número de contextos que pueden ser modelados y resueltos por sus metodologías de solución. Para resolver estos problemas se utiliza una gama de algoritmos en los cuales, el espacio de búsqueda de una solución crece exponencialmente a medida que el tamaño del problema aumenta, lo cual representa un gran reto al intentar resolver instancias del problema de gran escala. En este trabajo se aborda el *Traveling Purchaser Problem*, el cual consiste en un agente comprador que debe decidir en que conjunto de mercados comprar cierto conjunto de productos minimizando el costo total de la solución. Este problema tiene tres capas de decisión las cuales consisten en la selección de mercados, ruteo y compra de productos, por lo que hace que sea un caso particular y especialmente complejo de los problemas de optimización combinatoria. Para abordar estas tres capas de decisión y dar solución al problema, se propone un algoritmo de Q-Learning integrado con heurísticas constructivas para la creación del espacio de acciones combinatorial. El algoritmo de Q-Learning busca que un agente, a través de la interacción con el ambiente aprenda la combinación de decisiones secuenciales que minimizan el costo total de la solución (costo de ruta más costo de compra) para una instancia dada del problema.

Este artículo presenta el problema del *Traveling Purchaser Problem* en la Sección 2, en las secciones 3 y 4 describe la familia de algoritmos que se proponen como acercamiento de solución y su aplicación a la optimización combinatoria. Posteriormente, en la Sección 5 se expone el modelo que se construyó del TPP para *Reinforcement Learning*. En la sección 6 y 7 se expone la metodología de solución propuesta, el algoritmo de *Q-Learning* con la generación heurística de acciones. En las secciones 8 y 9 se presentan los resultados obtenidos y se hace un análisis sobre estos. Finalmente, en la Sección 9 se concluye sobre el trabajo y se abre un hilo hacia trabajo futuro.

2. *Traveler Purchaser Problem*

El problema del comprador viajero, o *Traveler Purchaser Problem* (TPP), es una generalización del *Traveler Salesman Problem* (TSP). Este modelo toma como base de un grafo conectado G, el cual tiene un conjunto de nodos $N = \{0, 1, ..., |N| - 1\}$ y un conjunto de arcos $A = \{(i, j)| \ \ i, j \ \in \ N\}$. Cada uno de los arcos que conectan cualquier par de nodos tiene un costo asociado $c_{ij} \ \ \forall \ \ (i, j) \in \ A$. Adicional a esto, hay un conjunto M de commodities que se deben adquirir. Cada uno de los commodities tiene un precio que depende de donde se adquiera $p_{mi} \ \ \forall \ \ m \ \in \ M, \ \ i \ \in \ N$. A partir de lo anterior, se busca que un individuo

inicie y finalice el tour en el depot (0) y adquiera todos los productos en M empleando los arcos en A. Tomando como base estas características, se generan muchas variantes de este problema aunque se conserva un mismo objetivo, minimizar el costo de compra y de viaje. Para el enfoque que se tuvo en este proyecto, de acuerdo con la topología presentada por Manerba et al.(2016), se tomó la versión asimétrica, irrestricta, y determnística del TPP. El componente asimétrico indica que para cualquier par de nodos $i, j|\ \ (i,j)\ \in A$, los costos c_{ij} y c_{ji} no los mismos. Por otra parte, el componente irrestricto define que cualquier mercado que tenga disponibilidad de un producto, tendrá disponibilidad infinita de este. Por último, el componente determinístico refleja la certeza absoluta que se tiene en cada uno de parámetros del modelo. Esto incluye conocer costos, precios, disponibilidades, etc.

Debido a la gran complejidad de las capas de decisión del problema, la mayoría de los propuestas de solución en la literatura son a través de metaheurísticas. En la Tabla 1 se resume los trabajos más relevantes en este campo. A través de los diferentes algoritmos se busca solucionar instancias de variantes del modelo de TPP. Se debe resaltar que los acercamientos más exitosos son metodologías especializadas a este problema.

Tabla 1: Metaheuristic approaches to the TPP

Metaheuristic approaches.		
Metaheuristic	Reference	Problem
Tabu Search (TS)	Vog (1996)	U-TPP
	El-Dean (2008)	U-TPP
	Mansini et al. (2005)	R-TPP
Simulated Annealing (SA)	Vog (1996)	U-TPP
General Random Adaptive Search Procedure (GRASP)	Ochi et al. (2001)	U-TPP
Variable Neighborhood Search (VNS)	Ochi et al. (2001)	U-TPP
	Mansini and Tocchella (2009a,2009b)	TPP-B
Ant Colony Optimization (ACO)	Bontoux and Feillet (2008)	U-ATPP
Late Acceptance Hill-Climbing (LAHC)	Goerler et al. (2013)	U-STPP
Genetic Algorithms (GA)	Ochi et al. (1997)	U-ATPP
	Goldbarg et al. (2009)	U-STPP
	Almeida et al. (2012)	2TPP

3. *Reinforcement Learning*

Reinforcement Learning (RL) es una familia de algoritmos que aprovechan la estructura de un *Proceso de Decisión Markoviano* (MDP) para construir políticas que optimicen el desempeño de la toma de decisiones. En estos algoritmos, un agente se enfrenta a un ambiente que hereda las características del sistema. De este ambiente, se transmite al agente toda la información relevante para que entienda la configuración actual este. A partir de dicha información, el agente debe tomar decisiones que causarán una respuesta por parte del ambiente, dependiendo de las dinámicas de transición intrínsecas del sistema. Dicha respuesta se dará en forma de: un cambio de estado y una métrica cuantitativa del desempeño de la acción. Para poder utilizar los algoritmos de Reinforcement Learning se deben establecer cinco componentes:

- Variable de estado S_t: Captura, para un momento t, toda la información relevante sobre el ambiente que necesita saber el agente. La variable de estado es una función del historial de decisiones e información exógena.

- Variable de decisión o acción a_t: Expresa la decisión que se toma en un instante t dado un estado $s \in S_t$.

- Información exógena W_t: Representa la información que se conoce entre dos periodos de tiempo. Esta información puede ser resultado de tomar una acción.

- Función de transición S^M: Esta función captura las dinámicas de transición dentro del sistema. Es decir, es la que mapea el estado actual del sistema $s \in S_t$ junto con la acción que se tome a_t, hacia el siguiente

estado S_{t+1}. Esta función se presenta en la Ecuación (1):

$$S_{t+1} = S^M(S_t, a_t, W_{t+1}) \quad (1)$$

- *Rewards* $C(S_t, a_t)$: Es la métrica que se tiene para medir el desempeño de los agentes dentro del sistema. Este, representa el beneficio/costo de tomar una acción a_t estando en el estado S_t. El objetivo final del agente, es maximizar o minimizar el valor esperado de la suma de los retornos durante todo el proceso de decisión. Esto se captura en la Ecuación (2), dónde el valor esperado del costo en un estado ($V^\pi(s)$) esta dado por: El costo de tomar la mejor acción en dicho estado ($C(s, a)$) más el valor esperado del costo del estado al que se transicionará ($V^\pi(s')$), descontado con la tasa γ

$$V^\pi(s) = max_a\{C(s, a) + \gamma V^\pi(s')\} \quad (2)$$

Los algoritmos de RL tienen una gran ventaja sobre algoritmos de aprendizaje supervisado de Machine Learning. A diferencia de otras formas de inteligencia artificial, no es necesario tener una muestra de datos con las observaciones de variable de respuesta. Por el contrario, los algoritmos de RL aprenden interactuando con el ambiente y recibiendo *rewards* que reflejan si el resultado de las acciones es deseable o no (Sutton Barto, 2018). En consecuencia, es necesario tener suficiente conocimiento sobre el sistema que se está modelando para que los algoritmos puedan aprender a través de exploración y explotación en el ambiente. Una vez el agente aprende del ambiente, es capaz de tomar una decisión para cada uno de los estados del sistema. Así, es capaz de mapear cada estado a la decisión que conlleva a un mejor desempeño en el futuro. Esto se denomina como políticas de decisión $A^\pi(S_t)$, que en principio son funciones que al recibir un estado, retornan la mejor decisión que se puede tomar.

4. *Reinforcement Learning* en optimización combinatoria

En los últimos años, hay una iniciativa por parte de la comunidad científica de implementar algoritmos de RL en problemas de optimización combinatoria. Esto ha generado una nueva rama de investigación que propone utilizar algoritmos de RL junto con algoritmos exactos y aproximados, enfoques clásicos de la optimización combinatoria. Por medio de esta combinación, se abren nuevas posibilidades para afrontar los retos que imponen estos problemas.

En Hubbs et al (2020) se presenta O-R Gym. Esta es una iniciativa por parte de investigadores de la Universidad Carnegie Mellon, en la cual se desarrollan varios ambientes compatibles con algoritmos de RL, para múltiples problemas clásicos de investigación de operaciones (OR). Entre ellos se encuentran el TSP, VRP, asignación, inventarios, entre otros. Los resultados presentados en (Hubbs et al, 2020) sirvieron como base para el diseño e implementación del ambiente del TPP, problema trabajado en este documento.

Adicional a esto, se hizo una revisión de la literatura la cual se presenta en la Tabla 2. De esta revisión, se encontró que todos los enfoques relacionados con RL para problemas de optimización combinatoria, se enfrentan a un espacio de acciones únicamente relacionado con decisiones de ruteo, presentes en el TSP y VRP. Por lo tanto, al conocimiento del autor, este es el primer trabajo que busca abordar dos capas de decisión adicional (selección de proveedores y compra de productos). A continuación se presenta la metodología propuesta para dar solución a estas tres capas de decisión.

5. Formulación del problema

Con sustento teórico, el primer paso para poder modelar el TPP como un ambiente y evaluar algoritmos de RL fue construir un modelo que respetara las propiedades de un MDP.

Variable de estado Para modelar la información relevante del modelo que recibirá el agente comprador, se empleó un vector de N + M componentes. La primera componente del vector indica el nodo en el que se encuentra el agente en la iteración actual. Las siguientes N - 1 componentes, indican con una variable

binaria si se puede visitar (1) o no (0) cada uno de los nodos con oferta de commodities ($\forall \ i \in N \setminus \{0\}$). Es decir, capturan la factibilidad de visitar todos los mercados. Por último, los últimos M componentes indican con una variable binaria si ya se compró (1) o no (0) cada uno de los commodities ($\forall \ j \in M$). Esto es, un historial de compras del episodio. Se presentan dos ejemplos de un estado para un ambiente con 3 proveedores y 3 mercados. En el primero, el estado inicial del ambiente, se parte del nodo 0, existe factibilidad de visitar cualquier nodo y no se ha comprado ninguno de los commodities. En el segundo, el agente se encuentra en el nodo 2, sin factibilidad de ir a este mismo nodo, y se registra que ha comprado los primeros dos commodities.

$$s_0 = [0, 1, 1, 1, 0, 0, 0]$$

$$s_t = [2, 1, 0, 1, 1, 1, 0]$$

Tabla 2: Trabajos relevantes de RL aplicados a problemas de optimización combinatoria

Autor	Problema resuelto		Método de entrenamiento
	TSP	VRP	
(Bello, Pham, V le, Norouzi, & Bengio, 2016)	X		REINFORCE
(Dai, Khalil, Zhang, Dilkina, & Song, 2017)	X		DQN
(Nazari, Oroojlooy, Snyder, & Takác, 2018)	X	X	REINFORCE
(Deudon, Cournut, Lacoste, Adulyasak, & Rousseau, 2018)	X		REINFORCE
(Kool, van Hoof, & Welling, 2019)	X	X	REINFORCE
(Chen & Tian, 2019)	X	X	Q-actor-critic
(Ma, Ge, He, Thaker, & Drori, 2020)	X		Hierarchical
(Drori, y otros, 2020)	X	X	MCTS
(Lu, Zhang, & Yang, 2020)	X	X	REINFORCE

Variable de decisión o acción Previo a describir el modelo que se usó para la variable de decisión, es necesario notar que el conjunto de acciones factibles varía dependiendo del estado en que se encuentre el agente. Esto se debe a que si un nodo ya se visitó o commodity ya se compró, las acciones que incluyan visitar ese nodo o comprar esos commodities ya no son factibles. Por lo tanto, es un conjunto indexado en los estados, dónde $A(S_t)$ representa el conjunto de acciones factibles para el estado S_t. A partir de esto, las decisiones a_t se modelaron como un vector con 1 + M componentes. La primera componente indica a dónde se dirigirá el agente en el próximo instante. Las siguientes M componentes indican con una variable binaria si se desea (1) o no (0) comprar cada uno de los commodities en el nodo en que se visitará en la siguiente iteración. En el primer ejemplo de los que se presentan a continuación, el agente decide dirigirse al tercer mercado y comprar únicamente el primer commodity. En el segundo, el agente decide ir al primer nodo y comprar los dos últimos commodities.

$$a_t = [3, 1, 0, 0]$$

$$a_{t'} = [1, 0, 1, 1]$$

Información exógena Para el modelo determinístico que se trabajó durante este proyecto, no hay información exógena al sistema que sea relevante

Función de transición Para la variante de TPP que se estudió en este proyecto, la función de transición es determinística. Esto quiere decir que la probabilidad de hacer transición de un estado S_t a otro estado S_{t+1} es 0 o 1. Se sabe con certeza qué sucederá una vez se tome una acción si se está en una configuración en particular del sistema. Para este modelo, la función de transición mapea los estados teniendo en cuenta

las decisiones que se están tomando. Esto quiere decir, si en la acción está estipulado que se visitará un nodo i y se comprarán un conjunto de commodities H, la función de transición llevará del estado actual al siguiente dónde la posición actual será el nodo i, se quitará la factibilidad del nodo i, y se habrá comprado los commodities en H. En caso de que la acción no sea factible, la función de transición llevará al agente al mismo estado, un instante después ($S_t = S_{t+1}$). Para ejemplificar las transiciones de estados, se presenta el siguiente ejemplo. En este, el agente parte del estado inicial y decide visitar el tercer nodo para comprar el primer commodity. Por lo tanto, tras la transición, el agente se encontrará en el tercer nodo, sin factibilidad de visitar el nodo en el que se encuentra y con el primer elemento ya adquirido.

$$s_t = [0, 1, 1, 1, 0, 0, 0]; a_t = [3, 1, 0, 0]$$

$$s_{t+1} = S^M(s_t, a_t) = [3, 1, 1, 0, 1, 0, 0]$$

Rewards Para este modelo, la función de costos dependerá del estado en que se encuentre el agente y la decisión que se tome ($C(S_t, a_t)$). El costo de tomar una acción en un estado estará compuesto por dos partes: En primer lugar, el costo de transportarse del nodo actual al nodo objetivo. En segundo lugar, el costo de adquirir los commodities indicados por la acción en el nodo de destino. De esta manera se modelaron los costos relevantes del TPP como $C(S_t, a_t) = c_{ij} + \sum_{m \in M | compram} p_{mj}$. Hay dos precisiones que se deben hacer respecto a esta función. El último costo de cada episodio tendrá incluido el costo de retornar al nodo base. Esto quiere decir, que en el instante en que se compran todos los commodities, habrá un costo adicional en el *reward* de esa transición. Este costo adicional capturará el costo del transporte desde el último nodo que se visita al depot para completar el circuito. Esto quiere decir, que si en la primera transición del ambiente (s_0) el agente decide visitar el tercer nodo y comprar todos los commodities (a), esta transición tendrá el siguiente costo.

$$C(s_0, a) = c_{(0,3)} + p_{(1,3)} + p_{(2,3)} + p_{(3,3)} + c_{(3,0)}$$

Por otra parte, en caso de que se tome una decisión no factible, se generará un costo significativamente elevado. De esta forma, se busca penalizar la toma de acciones inválidas durante el proceso de decisión. Para la transición que se ejemplificó en el punto anterior, el *reward* está compuesto por:

6. Algoritmo de *Q-Learning*

Teniendo un ambiente del TPP el cual utiliza los componentes descritos del MDP, se procedió a implementar un algoritmo de *Q-Learning* para determinar la política óptima de cualquier instancia del TPP. Un agente de *Q-Learning*, en principio, busca explorar el espacio de estados y, a medida que lo hace, va registrando en una tabla (Q-Table) el desempeño de haber tomado una acción en un estado. Dicho desempeño no hace referencia únicamente al costo inmediato de tomar una acción ($C(s_t, a_t)$), sino a qué tan buena fue la acción teniendo en cuenta todo el proceso de decisión y como repercute esta en el valor de las decisiones futuras. Durante cada uno de los episodios, el agente actualiza los *Q-values*, en función de la nueva información que recolecta. Una vez termina su etapa de entrenamiento, el agente recurre a la tabla para determinar para cada estado, cuál es la mejor acción que se puede tomar. La función mediante la cual el agente actualiza el valor Q de una acción para un estado, una vez se conoce su costo, se describe en la Ecuación (3). Dónde la expresión $Q(S_t, a)$ es el valor actual de la acción en dicho estado, α es la tasa de aprendizaje mediante la cual se modela la relación exploración/explotación y γ es la tasa de descuento con la cual se gradúa la importancia que tiene para el agente el desempeño en el largo plazo.

$$Q^{new}(S_t, a) = (1 - \alpha) * Q(S_t, a) + \alpha * (C(S_t, a) + \gamma * max_a Q(S_{t+1}, a)) \quad (3)$$

Debido al componente combinatorial del TPP no es posible conocer todo el espacio de estados S ni todo el espacio de las posibles acciones $A(S)$ a causa de la maldición de la dimensionalidad. Esto representa un problema dado que un agente de Q-Learning debe inicializar arbitrariamente todos los Q-values para

cada pareja estado acción y de esta manera ir seleccionado las mejores acciones a lo largo del proceso de entrenamiento. Por lo tanto, se decidió construir tres heurísticas diferentes para generar acciones factibles y entregarlas al algoritmo de *Q-Learning* para que este explore el espacio de soluciones. Estas heurísticas se construyeron de tal manera que emularan comportamientos que ayuden a explorar el espacio de estados y de acciones de manera eficiente. Las tres heurísticas que se construyeron constan de dos fases, en la primera se decide cual de los nodos factibles se visitará en la siguiente iteración. Durante la segunda fase, se escoge los productos que se adquirirá en el nodo destino. A continuación se presentan las tres heurísticas que se desarrollaron.

7. Generación heurísitica de acciones

Heurística Greedy La primera heurística que se implementó fue una generación de acciones sesgada a la disponibilidad de los productos en el nodo a visitar. Su primera fase se muestra en el Algoritmo 1. Este algoritmo inicializa en vacío el nodo a visitar y se genera un conjunto N_θ de las posibles acciones (Líneas 1 - 2). Después, computa unos pesos basados en la proporción de cada uno de los nodos en N_θ según sus respectivos costos c_{ij} (Líneas 3-5). Posteriormente, se generan una realización aleatoria (Línea 6) y se determina cuál nodo se visitará por medio de una función de probabilidad acumulada (Líneas 7-14). Por último, se reporta el nodo seleccionado (Línea 15).

Algorithm 1 Choosing target node: Greedy & K-items

Input: List $Not\ \ visited, feasible\ \ nodes$;
Output: Node $Target\ \ node$;

1: $A_t \leftarrow \emptyset$
2: $N_\theta \leftarrow Not\ \ visited,\ \ feasible\ \ nodes$
3: **for** $j \in N_\theta$ **do**
4: $\quad P(j) \leftarrow c_{ij} / \sum_{s \in N_\theta} c_{is}$
5: **end for**
6: $rand \leftarrow random\ \ realization$
7: $acum \leftarrow 0$
8: **for** $j \in N_\theta$ **do**
9: $\quad acum \leftarrow acum + P(j)$
10: $\quad$ **if** $acum \geq rand$ **then**
11: $\quad\quad A_t \leftarrow j$
12: $\quad\quad Break$
13: $\quad$ **end if**
14: **end for**
15: **Return** A_t

K-items La Heuristica K-items utiliza la misma fase inicial descrita en el Algoritmo 1. Para su segunda fase, la heurística K-items recibe como parámetro K un valor entre 0 a 1 que indica la máxima proporción de los commodities que se pueden comprar en un único nodo. Por lo tanto, el número máximo de productos a comprar esta dado por $K * M$, redondeado al entero inferior. Así, en caso de que haya menos de $K * M$ elementos disponibles, se comprará todos estos. En caso de que haya más commodities disponibles que el límite, se escogerán al azar $K * M$ elementos redondeado al entero inferior.

Heurística pseudoaleatorizada Esta heurística, como las dos anteriores, consta de dos etapas. En la primera, se evalúan los posibles nodos que se pueden visitar y se construye una lista de candidatos restricta (RCL) para determinar el mejor subconjunto de nodos que se puede visitar (Ver Algoritmo 2). Para esto, se toman los nodos factibles (Línea 3) y se registran las máximas y mínimas distancias desde el nodo actual hasta los nodos factibles (Líneas 4 y 5). Con estas distancias se computa un límite que es función de dichas distancias y el parámetro α definido previamente(Línea 6). Finalmente, se conforma la RCL con todos los nodos factibles que estén dentro del límite computado (Líneas 7-11).

El nodo que se visitará se escoge aleatoriamente de la RCL.

Algorithm 2 Generating Restricted Candidate List

Input: List $Not\ \ visited, feasible\ \ nodes$; **Parameter** α
Output: Node $Restricted\ \ Candidate\ \ List$;

1: $RCL \leftarrow \emptyset$
2: $0 \leq \alpha \leq 1$
3: $N_\theta \leftarrow Not\ \ visited,\ \ feasible\ \ nodes$
4: $d_{min} \leftarrow min_{j \in N_\theta}\{c_{ij}\}$
5: $d_{max} \leftarrow max_{j \in N_\theta}\{c_{ij}\}$
6: $limit \leftarrow d_{min} + \alpha * (d_{max} - d_{min})$
7: **for** $j \in N_\theta$ **do**
8: **if** $c_{ij} \leq limit$ **then**
9: $RCL \leftarrow RCL \cup j$
10: **end if**
11: **end for**
12: **return** RCL

Algorithm 3 Purchase policy: RCL based route and index based purchase

Input: List $Commodities$;
Output: List $Purchase\ \ list$

1: $j \leftarrow target\ \ node$
2: $A_t \leftarrow \emptyset$
3: **for** $m \in M$ **do**
4: **if** $m\ is\ feasible$ **then**
5: $p_{min} \leftarrow min_{n \in N} p_{mn}$
6: **if** $p_{mj} = p_{min}$ **then**
7: $A_t(m) \leftarrow 1$
8: **else**
9: $\delta \leftarrow p_{mj} - p_{min}$
10: $prob \leftarrow 0{,}5 * (1 - \delta / p_{mj})$
11: $rand \leftarrow realización\ \ pseudo - aleatorio$
12: **if** $prob \geq rand$ **then**
13: $A_t(m) \leftarrow 1$
14: **else**
15: $A_t(m) \leftarrow 0$
16: **end if**
17: **end if**
18: **else**
19: $A_t(m) \leftarrow 0$
20: **end if**
21: **end for**
22: **return** A_t

En cuanto a la segunda fase, para determinar los elementos que se compraran en el nodo seleccionado, se evalúan dos escenarios para cada elemento disponible y factible tal y como se muestra en el Algoritmo 3. Para cada producto m disponible en el nodo seleccionado (Líneas 3-4) se verifica si su precio p_{mj} es el mejor precio al que se puede adquirir en todo el sistema. En caso de ser cierto se adquiere dicho producto en el nodo seleccionado (Líneas 6 - 8). De lo contrario, se construye un indicador del precio del commodity

que incluye el mínimo precio al que se consigue y el precio en el nodo destino (Líneas 9-11). El índice se calcula como $0{,}5 * (1 - \frac{\delta_{precio}}{p_{mj}})$, dónde δ_{precio} indica la diferencia entre el precio del commodity en el nodo j y el mínimo precio al que se consigue en todo el sistema. Cabe resaltar que este indicador disminuye su valor a medida que crece la diferencia entre el precio disponible y el mejor precio. Finalmente se compara el indicador contra un número aleatorio, si el indicador es mayor que el número aleatorio, se compra el commodity. De lo contrario, no se compra (Líneas 12-17).

8. Resultados

Para validar el desempeño del algoritmo de Q-learning y su capacidad de resolver el problema del TPP se seleccionaron 65 instancias de prueba del benchmarck propuesto por Snigh y van Oudheusden (1997) disponibles en http://webpages.ull.es/users/jriera/TPP.htm. Los resultados fueron comparados frente a la mejor metodología de solución propuesta en la literatura, un algoritmo GRASP con PathRelinking propuesto en (Cuellar et al (2021). Para comparar el desempeño de los enfoques de solución, se calculo el gap (%) como se expone en la Ecuación (4). Estos se hicieron referentes al modelo matemáticos propuesto en Manerba et al (2016).

$$gap\ (\%) = \frac{Valor\ \ obtenido - Óptimo}{Óptimo} \tag{4}$$

El algoritmo de Q-Learning fue implementado en Python 3.8.12 y ejecutado en un computador con las siguientes descripciones: Procesador Quad-Core Intel Core i5 de 2,3 GHz y 16 GB RAM. El algoritmo GRASP fue codificado en C++ y ejecutado con las siguientes descripciones: AMDRyzen 7-3800X @ 3.9GHz, Windows 10 y 32 GB RAM. Los parametros iniciales del algoritmo de Q-learning son:

- Tasa de aprendizaje α: 0,1
- Tasa de descuento γ: 0,95
- Número de episodios: 10.000
- Épsilon ϵ: 0,5

En primer lugar, el agente logró desenvolverse exitosamente en el ambiente y explorar las posibles configuraciones de este, construyendo soluciones factibles. Esto es un logro, dada la complejidad combinatorial tanto de la ruta como de la compra. Adicional a esto, el algoritmo es introductorio a RL y no es especializado para un problema con tres capas de decisión, como sí lo son los algoritmos exactos y heurísticos. Sin embargo, los resultados iniciales no son muy prometedores dado que el gap que se encontró es significativamente alto y el tiempo computacional refleja un esfuerzo grande, como se evidencia en la Tabla 3. Asimismo, a medida que escalan los problemas el gap computado incrementa. Adicional a esto, es posible ver que en instancias más pequeñas, la heurística Greedy tiene un mejor desempeño que las demás. Sin embargo, cuando crece la dimensión del problema las heurísticas K-items y por RCL desarrollan mejores soluciones.

Tabla 3: Resultados obtenidos por el algoritmo Q-Learning en compración con Cuellar et al (2021) en terminos de Gap (%).

	Gap (%)											
|M||N|	10				20				50			
	GRASP*	Greedy	k-items	RCL	GRASP*	Greedy	k-items	RCL	GRASP*	Greedy	k-items	RCL
10	0	**10,3**	20,0	20,4	0	**26,2**	41,4	37,9	0	47,3	**35,9**	46,7
20	0	**11,8**	89,6	22,6	0	29,7	**22,9**	26,2	0	36,1	**28,7**	30,4
50	0	**13,2**	18,2	**13,2**	0	23,9	20,2	**19,6**	0	26,6	**22,5**	24,8
100	5,0	13,3	30,2	**11,4**	0	20,4	19,7	**15,7**	0,8	21,9	20,8	**20,7**
Average	1,3	**12,2**	39,5	16,9	0	24,7	20,9	**20,5**	0,2	28,2	**24,0**	25,3

* Metaheurística GRASP, (Cuellar et al., 2021)

El desempeño computacional del algoritmo de Q-Learning se presenta en la Tabla 4. Se puede ver que el

desempeño computacional del algoritmo de Q-Learning es muy bajo requiriendo hasta 7 horas de procesamiento en las instancias más grandes. Se puede observar también que los tiempos de entrenamiento difieren significativamente entre las heurísticas. A través de la dimensión de los nodos y de los mercados se mantiene un mejor desempeño de la heurística Greedy. Aún en instancias más grandes, el tiempo de entrenamiento empleando esta heurística se mantiene moderado. Por otra parte, la heurística por RCL emplea un mayor tiempo que las demás. De hecho, en todas las dimensiones toma en promedio más del doble de tiempo en entrenar que las otras.

Tabla 4: Resultados obtenidos por el algoritmo Q-Learning en comparación con Cuellar et al (2021) en términos de tiempo computacional.

	Time (s)											
|M||N|	10				20				50			
	GRASP*	Greedy	k-items	RCL	GRASP*	Greedy	k-items	RCL	GRASP*	Greedy	k-items	RCL
10	2	**6,7**	9,5	33	2,0	**36,4**	52	1371,1	2,49	**59,45**	104,71	2156,85
20	2	**14,0**	27,8	180	3,1	**59,3**	201	9075,9	3,98	**162,44**	6783,85	1951,93
50	4	**29,1**	104,2	536	6,4	**856,0**	7164	2955,7	11,07	**1966,77**	4808,17	26622,69
100	9,2	**45,9**	532,9	3129	12,3	14232,1	**2467,1**	7097,8	23,53	**3555,46**	4876,25	19356,71
Average	4,4	**23,9**	168,6	969	5,9	3796,0	**2471,1**	5125,1	10,27	**1436,03**	4143,24	12522,05

Para ver la sensibilidad del algoritmo de Q-Learning se hizo una búsqueda de hiperparametros utilizando una grilla de $\alpha = \{0{,}05, 0{,}1, 0{,}15\}$, $\gamma = \{0{,}925, 0{,}95, 0{,}975\}$, $Episodes = \{8000, 10000, 20000\}$ y $\epsilon = \{0{,}9, 0{,}95, 0{,}975\}$. Para cada una de las instancias se tomó la heurística con mejor desempeño y sobre esta se hizo la calibración. Los *gaps* que se encontraron son mucho menores que los que se había encontrado previamente, como se puede evidenciar en la Tabla 5. Aún en instancias más grandes se mantiene un gap razonable. Sin embargo, aunque no se registraron los tiempos, el esfuerzo computacional durante esta fase de calibración de parámetros fue alto. La calibración en las instancias más grandes dónde la mejor heurística fue RCL, tomó días en completar.

Tabla 5: Resultados obtenidos por la versión calibrada del algoritmo Q-Learning en comparación con Cuellar et al (2021) en términos de Gap (%)

	Gap (%)					
|M||N|	10		20		50	
	GRASP	Best Heur.	GRASP	Best Heur.	GRASP	Best Heur.
10	0,00	1,52	0,00	3,52	0,00	17,91
20	0,00	4,13	0,00	10,27	0,00	14,08
50	0,00	4,88	0,00	11,96	0,00	
100	5,00	6,60	0,00		0,80	
Average	1,25	4,28	0,00	8,58	0,20	16,00

9. Discusión

Para abrir la discusión frente a los resultados, se desea empezar por los *gaps* que se obtuvieron respecto al resultado óptimo. Al correr el algoritmo *Q-Learning* con los hiperparámetros iniciales, los resultados no son competitivos con el estado del arte. Una diferencia porcentual con la solución exacta de 20 % en las instancias más grandes permite ver que el algoritmo es capaz de encontrar una solución factible pero no es competitiva. Adicional a esto, se puede observar que el desempeño entre una heurística y otra varía. La heurística Greedy presentó un mejor desempeño en las instancias más pequeñas, mientras que en las grandes tuvieron mejor desempeño K-items y RCL. Esto se puede atribuir a que la heurística Greedy está sesgada a la disponibilidad de los productos. Por lo tanto, no se tiene diversificación en la generación de las acciones por medio de esta heurística. En instancias pequeñas no es un inconveniente, pero a medida que crecen los problemas este sesgo reduce la exploración de estados y acciones. Por lo tanto, se obtienen mejores resultados

con las otras heurísticas que tienen la posibilidad de explorar diversos estados por sus componentes probabilísticos. No obstante, se observa una mejora significativa al hacer una afinación de la configuración interna del *Q-Learning*. Para esto, se realizó una grilla de hiperparámetros con el fin de calibrar tasa de aprendizaje, tasa de descuento, numero de episodios de entrenamiento y épsilon. Con esta estrategia, se alcanzó un gap de 4,3 % en las instancias más pequeñas y 16 % en las más grandes. Esto, permite ver un desempeño interesante que con otros enfoques de RL puede alcanzar mejores resultados.

Por otra parte, analizando el componente computacional del algoritmo de Q-Learning se puede observar que el tiempo de entrenamiento no escaló bien con la dimensión del modelo, y en grandes instancias el tiempo de computo es extremadamente alto. Esto tiene dos razones principales, el número de iteraciones y el desarrollo de las heurísticas en las réplicas. Nuevamente, se encontró un desempeño diferente entre las heurísticas. Por medio de la Heurística Greedy se obtuvo los menores tiempos de entrenamiento en casi todas las instancias. Esto, debido a que la heurística persigue comprar todos los elementos lo más rápido posible. En contraste, la heurística por RCL requirió el mayor esfuerzo computacional de las tres heurísticas. Esto se debe a que esta heurística cuenta con dos procesos complejos en cada una de sus dos fases. Por lo tanto, al escalar las dimensiones del problema se deberá hacer un esfuerzo significativo en cada una de las dos fases. Esto, teniendo en cuenta que la generación de acciones toma lugar en cada iteración de cada periodo de entrenamiento.

Desde otro punto de vista, el hecho de que un algoritmo introductorio a RL sea capaz de desarrollarse satisfactoriamente en un ambiente combinatorio con las capas adicionales de decisión del problema del TPP. Teniendo en cuenta que los algoritmos y heurísticas contra las que se compara nuestro algoritmo son especializadas al problema, el agente de *Q-Learning* con la información del ambiente y heurísticas de apoyo logró un desempeño destacable.

10. Conclusiones

El trabajo realizado desarrolló y propone un ambiente del TPP, diseñado para algoritmos de *Reinforcement Learning*, en específico *Q-Learning*. Este ambiente está en la capacidad de producir instancias apropiadas para las necesidades del modelador. En este, se puede configurar el número de nodos, de commodities, la disponibilidad de los productos, etc. Adicionalmente, el ambiente permite visualizar por medio de un método de renderización las acciones tomadas por el agente. Por otra parte, tiene algoritmos exactos y meta heurísticos implementados de forma que la comparación con estos referentes es sencilla.

Por otra parte, se propone un modelo del TPP compatible con algoritmos de *Reinforcement Learning*. En este modelo se captura todas las características relevantes de este problema y se capturan en un MDP. Por medio de un estudio computacional con instancias clásicas de la literatura, se establecen *benchmarks* en estos problemas con el acercamiento de RL. Así se aborda un problema exploratorio con una metodología nueva en el ámbito de la investigación.

Finalmente, se aprovechó la capacidad de integrar algoritmos de RL con heurísticas para poder abordar diversas capas de profundidad del problema. Con algoritmos más complejos en la forma de procesar las características del ambiente, como $Deep\ Q-Learning\ (DQN)$, se podría alcanzar mejores resultados con un menor esfuerzo computacional. Adicionalmente, estos algoritmos podrían permitir evaluar variantes del TPP de interés.

Como trabajo futuro, se desea abarcar por medio del ambiente diferentes variantes estocásticas del TPP. Poder incorporar en el ambiente factores estocásticos haría mucho más relevantes sus aplicaciones a problemas de la industria. Algunas variaciones del TPP incluyen distancias estocásticos de los arcos (Kang Ouyang, 2010) $c_{ij} \sim f(i,j,\theta)$, aparición probabilística de los nodos a través de las realizaciones (Liu, 2008) $P[n_i\ aparezca\ en\ la\ realización\ r] = p$ y con precios estocásticos de los commodities (Percus Martin, 1999) $p_{mk} \sim f(m,k,\theta)$. Adicionalmente, se puede entrenar agentes con otros algoritmos de *Reinforcement Learning*

con redes neuronales. Dichos algoritmos corresponden a otra familia denominado *Deep Learning*. En estos, se incluyen redes neuronales para computar los *rewards* de las acciones en la función del *Q-Learning*. Por medio de esto, se obtiene una extracción mucho mejor de las características del ambiente. Finalmente, se desea construir ambientes para otros problemas de investigación de operaciones. Dado que ya se cubrió el componente combinatorio de ruteo, este se podría incorporar a problemas como el VRP o TSP. Igualmente, se podría cubrir problemas no combinatorios como inventarios.

Referencias

Manerba, D., Mansini, R., Riera-Ledesma, J. (2016). The Traveler Purchaser Problem and its variants. European Journal of Operational Research (ISSN 0377-2217). https://doi.org/10.1016/j.ejor.2016.12.017

Hubbs, C., Perez, D., Sarwar, O., Sahinidis, V., Grossmann, E., Wassick, M. (2020). OR-Gym: A Reinforcement Learning Library for Operations Research Problems. arXiv preprint arXiv:2008.06319.

Cuellar D., Alvarez-Martinez D., Gomez C. (2021) A GRASP/Path-Relinking algorithm for the traveling purchaser problem. International Transactions in Operational Research (ISSN 0969-6016) 0 (NA), pp. 1-27.

Percus, A., Martin, O. (1998). The Stochastic Traveling Salesman Problem: Finite Size Scaling and the Cavity Prediction. Journal of Statiststical Physics. 94. 10.1023/A:1004570713967.

Liu, Y. (2008). Diversified local search strategy under scatter search framework for the probabilistic traveling salesman problem. Eur. J. Oper. Res., 191, pp 332-346.

Kang, S., Ouyang, Y.(2011). The traveling purchaser problem with stochastic prices: Exact and approximate algorithms. European Journal of Operational Research, Elsevier, vol. 209(3), pp 265-272.

Sutton, S., Barto, G.(2018). Reinforcement learning: An introduction. MIT press.

Powell, W. (2019). A unified framework for stochastic optimization. European Journal of Operational Research, 275(3), pp 795-821.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W. (2016). OpenAI Gym. arXiv preprint arXiv:1606.01540

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., Dormann, N. (2021). Stable-Baselines3: Reliable Reinforcement Learning Implementations. Journal of Machine Learning Research, vol. 22(2021), pp 1-8.

Singh, K.N., van Oudheusden, D.L., (1997). A branch and bound algorithm for the traveling purchaser problem. European Journal of Operational Research 97, 3. pp 571–579.