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

APRENDIZAJE POR REFUERZO PROFUNDO PARA LA PREVENCIÓN DE INCENDIOS

TESIS PARA OPTAR AL GRADO DE MAGÍSTER EN GESTIÓN DE OPERACIONES
MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL INDUSTRIAL

LUCAS MURRAY HIDALGO

PROFESOR GUÍA:
Andrés Weintraub Pohorille

PROFESOR CO-GUÍA:
Jaime Carrasco Barra

COMISIÓN:
Richard Weber Haas

Este trabajo ha sido parcialmente financiado por:
FONDECYT y FONDEF
JC acknowledges the support of the ANID, through funding
Postdoctoral Fondecyt project No 3210311

SANTIAGO DE CHILE
2023

RESUMEN DE LA TESIS PARA OPTAR AL GRADO DE:
MAGÍSTER EN GESTIÓN DE OPERACIONES
MEMORIA PARA OPTAR AL TITULO DE:
INGENIERO CIVIL INDUSTRIAL
POR: LUCAS MURRAY HIDALGO
FECHA: 2023
PROFESOR GUÍA: ANDRÉS WEINTRAUB POHORILLE
PROFESOR CO-GUÍA: JAIME CARRASCO BARRA

APRENDIZAJE POR REFUERZO PROFUNDO PARA LA PREVENCIÓN DE INCENDIOS

En las últimas décadas, la escalada tanto en la frecuencia como en la intensidad de incendios forestales de gran magnitud ha surgido como consecuencia del cambio climático, convirtiéndolos en una considerable amenaza natural. La necesidad imperante de diseñar paisajes resilientes capaces de resistir tales catástrofes se ha vuelto primordial, lo que hace necesario el desarrollo de herramientas avanzadas de apoyo a la toma de decisiones. Las metodologías existentes, incluyendo la Programación Entera Mixta, Optimización Estocástica y Teoría de Redes, han demostrado ser efectivas pero se ven obstaculizadas por sus demandas computacionales, limitando su aplicabilidad a bosques extensos.

En respuesta a este desafío, proponemos el uso de técnicas de inteligencia artificial, en particular el Aprendizaje por Refuerzo Profundo, para abordar el complejo problema de la asignación de cortafuegos. Específicamente, empleamos enfoques basados en la función de valor como Deep Q-Learning, Double Deep Q-Learning y Dueling Double Deep Q-Learning. Aprovechando el poder del simulador de propagación de incendios, Cell2Fire, y las Redes Neuronales Convolucionales, hemos programado con éxito un agente computacional capaz de aprender la ubicación de cortafuegos dentro de un entorno forestal limitado, logrando resultados cercanos a óptimos.

Además, incorporamos un bucle de preentrenamiento, enseñando inicialmente a nuestro agente a imitar un algoritmo basado en heurísticas y observamos que coincide consistentemente o supera el rendimiento de estas soluciones. Nuestros hallazgos subrayan el inmenso potencial del Aprendizaje por Refuerzo Profundo para abordar desafíos de investigación operativa, especialmente en el ámbito de la prevención de incendios. Específicamente, nuestro enfoque demuestra convergencia con resultados altamente favorables en instancias del problema tan grandes como 20x20 celdas, marcando un hito significativo en la aplicación del Aprendizaje por Refuerzo a este problema crítico.

Hasta donde sabemos, nuestro estudio representa un esfuerzo pionero en la utilización del Aprendizaje por Refuerzo para abordar el problema mencionado anteriormente, ofreciendo perspectivas prometedoras en el ámbito de la prevención de incendios y la gestión del paisaje.

A mi Mamá, Papá y Hermanos.

Agradecimientos

Primero que nada, me gustaría agradecer a mis papás. A mi mamá, por su apoyo incondicional, por ser mi ejemplo a seguir y más que nada por el amor que me haz sabido entregar desde siempre. A mi papá, por estar siempre detrás mío, la vida es más facil sabiendo que lo estás. A mi hermano chico, por ser mi mejor amigo y compañero, sin tu amistad no sé si estaría dónde estoy. A mi hermano grande, por mostrarnos el camino.

Segundo, a todas las personas que acompañaron en esta etapa. A la Lauri por ser una fuente de apoyo ilimitada y enseñarme a cuidarme. A mis amigos del colegio, por los recuerdos de una de las etapas más felices de mi vida y por prestarme una oreja siempre que lo necesité. A mis amigos de la U, por hacer todo este camino infinitamente más ameno, ahora que se acaba les puedo decir que lo disfruté.

Tercero, a mi profesor guía, Andrés Weintraub, por la confianza que me mostró desde el primer minuto y el espacio que me dió para encontrar el camino por mi cuenta. También a Jaime Carrasco, por el apoyo y la perspectiva y a Tatiana Castillo, que es la única que sabe lo dificil que fue escribir esta tesis.

Finalmente, a todos los que de manera intencionada o por accidente han sido parte de este capítulo de mi vida.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

1.1. Incendios

El fuego cumple y siempre ha cumplido un rol en los ecosistemas boscosos, ocurriendo regularmente dependiendo de las características propias del bosque. Previo a la influencia humana, estos se generaban a partir de rayos en las estaciones más cálidas y secas. Debido a esta regularidad, los incendios se hacían cargo de limpiar el suelo boscoso de vegetación vieja o muerta que se acumulaba en los periodos libres de fuego, permitiendo una reincorporación de esta en el ecosistema (Halofsky, Peterson, y Harvey, 2020).

Un *régimen de fuego* se entiende como el patrón, frecuencia e intensidad de los incendios forestales que prevalece en un área específica durante un periodo de tiempo largo. Dentro de los factores que determinan un régimen de fuego se encuentran la vegetación, clima y topografía de un paisaje a sí como la acción humana (Crutzen y Goldammer, 1994). Durante el último siglo han habido cambios profundos en estos regímenes a lo largo de todo el mundo. En resumidas cuentas, se ha observado una disminución en la frecuencia, aumento extremo en la intensidad y alcance de los incendios, generándose el fenómeno de los mega-incendios. Existen tres causantes principales de este cambio: cambio climático, exclusión del fuego y disturbios de antecedentes. Por un lado, las épocas de incendios se han ido alargando, proyectándose que lo sigan haciendo, generando que la ventana de ocurrencia de estos sea más larga. Por otro, la práctica de la exclusión del fuego durante décadas ha generado una acumulación de combustible provocando que las áreas en donde se aplica sean capaces de albergar incendios más grandes y más intensos. Finalmente, el cambio del uso de suelo durante los últimos siglos ha provocado la aparición de zonas más riesgosas, como la *wildland-urban interface*[1], la limitación del repertorio de acciones de prevención debido al desarrollo de comunidades, provocando un incremento en la supresión y la modificación de las estructuras de los bosques (Stephens et al., 2014).

Con el avance del cambio climático, la situación descrita solo va a ir empeorando, transformando los mega-incendios en la normalidad más que en fenómenos. De esta manera, se vuelve crucial el estudio del fuego y el surgimiento de nuevas formas de manejarlo, que permita restaurar al fuego como un agente de preservación más que de caos.

[1] Zona de transición entre tierras desocupadas y zonas de desarrollo humano.

1.2. ¿Por qué Reinforcement Learning?

Reinforcement Learning es una rama de *Machine Learning* dedicada a la resolución de problemas de decisión secuenciales con el objetivo de maximizar una recompensa. Se ha ocupado extensamente y con éxito en distintas tareas como: Ajedrez (H. T. S. J. Silver D., 2016), Go (H. A. M. C. Silver D., 2016), Atari (Sutton y Barto, 2005), robótica (Kober, Bagnell, y Peters, 2013), etc. A pesar de esto, su aplicación en *Investigación de Operaciones* ha sido limitada. Se ha utilizado para resolver el *Travelling Salesman Problem* (Kool, van Hoof, y Welling, 2018), el *Beer Game* (Oroojlooyjadid, Nazari, Snyder, y Takác, 2021), *Bin Packing, News Vendor y Ruteo de vehíchulos* (Balaji et al., 2019). En lo que respecta a incendios, su uso se ha concentrado en control de incendios una vez que han iniciado su esparcimiento, no en prevención (Hammond, Schaap, Sabatelli, y Wiering, 2020), (Rocholl, 2021).

Existen dos motivaciones principales para incorporar métodos de aprendizaje en tareas de OR. Primero, si la resolución del problema mediante un algoritmo de *OR* es computacionalmente cara y se tiene conocimiento empírico de las decisiones tomadas por este algoritmo, se puede reducir la carga aproximándolas a través de aprendizaje de máquinas. Para lo anterior, se apunta a imitar al algoritmo de *OR* y por tanto este representa una cota superior del rendimiento a obtenerse mediante técnicas de aprendizaje. Segundo, de tenerse conocimiento experto de la resolución de un problema que no es lo suficientemente satisfactorio, puede ser de interés mejorar las soluciones que se obtienen. Con este fín, el algoritmo de aprendizaje utiliza una señal de recompensa como guía, que potencialmente puede generar soluciones que superen el rendimiento del experto (Bengio, Lodi, y Prouvost, 2021).

Este trabajo se centra principalmente en el segundo enfoque: utilizar reinforcement learning como un medio de mejorar el rendimiento de las soluciones obtenidas mediante un algoritmo subóptimo.

En general la conversión de un problema de Investigación de operaciones a un problema de Reinforcement Learning no es complicada. La traducción de estos a un *Markov Decision Process* (MDP) usualmente es natural y desde un MDP es trivial. El elemento crucial es que maximizar la recompensa obtenida en el problema de reinforcement learning implique maximizar la función objetivo en el problema de investigación de operaciones asociado y la inclusión de las variables de decisión en el estado del ambiente (Hubbs et al., 2020).

1.3. Definición del problema de asignación de cortafuegos

El problema de la localización espacial de cortafuegos es uno de los problemas más desafiantes e importantes en ecología (Albini, 1996). Este consiste en el posicionamiento de cortafuegos en un paisaje, de manera que de producirse un incendio, la menor porción posible de este se vea afectado. Este problema cae en la categoría de prevención de incendios, la cual se debe diferenciar del control de incendios. Esta última corresponde a la localización de cortafuegos ante un incendio en específico que ya ha iniciado su esparcimiento.

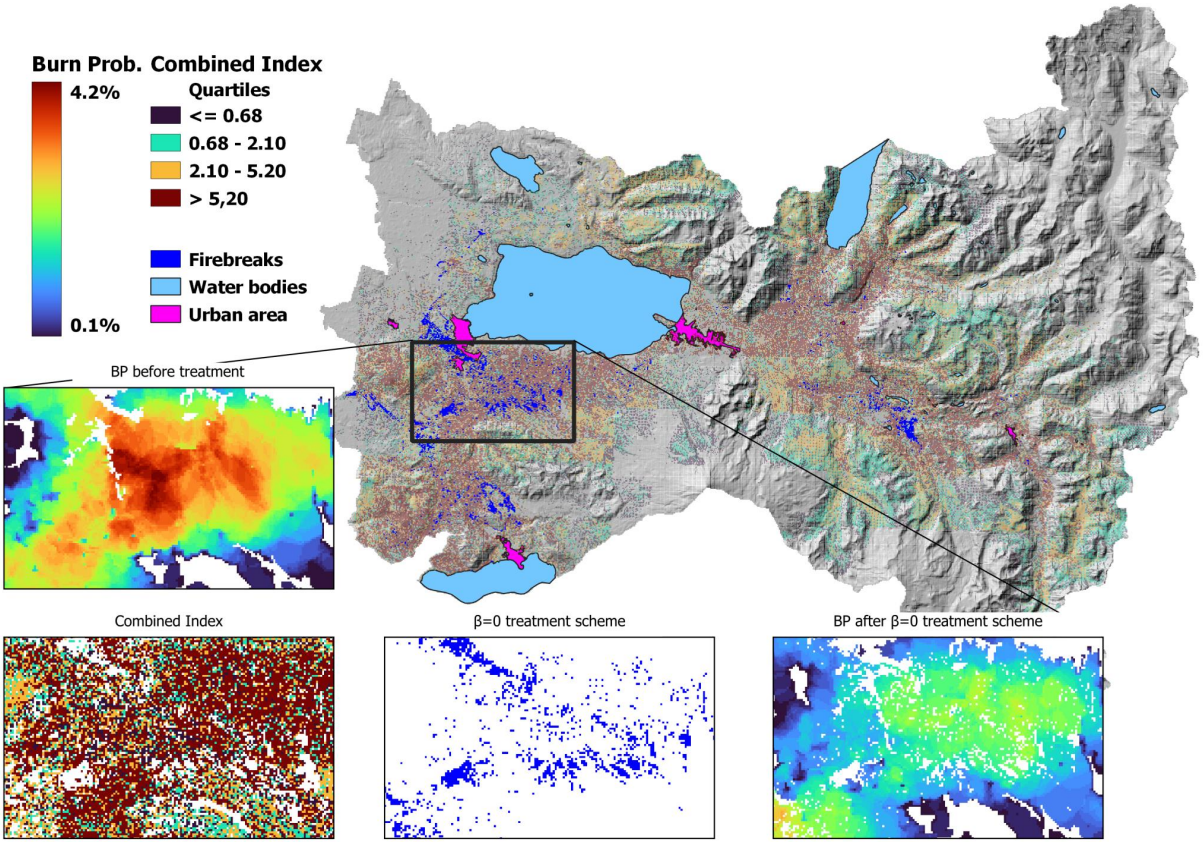


Figura 1.1: Ejemplo de una resolución del problema de cortafuegos obtenida por (Mahaluf Recasens, 2022).

Existe una vasta literatura asociada al problema, la cual se compone principalmente de soluciones mediante heurísticas (Palacios Meneses, 2022), Programación Entera Mixta (Cristobal Pais, 2021) y otros (Russo L, 2016). Sin embargo, según la literatura actual, aún no ha sido utilizado Reinforcement Learning para resolver el problema de la localización espacial de cortafuegos.

1.4. Objetivos Generales

Desarrollar una metodología que permita definir una distribución de cortafuegos en un paisaje sintético capaz de reducir la superficie afectada por el avance de un incendio.

1.4.1. Objetivos Específicos

1. Definir y plantear el problema de localización de cortafuegos dentro del marco de reinforcement learning.
2. Desarrollar una metodología que permita resolver el problema mencionado.
3. Aplicar la metodología desarrollada sobre una instancia artificial pequeña[2], mediante métodos tabulares de Reinforcement Learning.
4. Evaluar la eficacia de la metodología desarrollada en términos de convergencia para los algoritmos tabulares de Reinforcement Learning utilizados sobre instancias pequeñas.
5. Aplicar la metodología desarrollada sobre una instancia artificial grande[3], pero cuya solución es conocida, mediante Deep Reinforcement Learning.
6. Aplicar la metodología desarrollada sobre una instancia artificial grande, cuya solución es desconocida, mediante Deep Reinforcement Learning.
7. Evaluar la eficacia de la metodología en términos de convergencia para los algoritmos de Deep Reinforcement Learning utilizados sobre instancias grandes.
8. Comparación de las soluciones obtenidas mediante Deep Reinforcement Learning con aquellas obtenidas mediante otros métodos.

[2] Una instancia pequeña corresponde a una cuya solución se puede calcular mediante métodos tabulares y que su resolución no conlleva exigencias computacionales muy grandes.

[3] Una instancia grande corresponde a una cuya solución no se puede calcular mediante métodos tabulares debido a la exigencia computacional que representa.

Capítulo 2

Materiales y métodos

2.1. Cell2Fire

Uno de los elementos fundamentales a la hora de implementar un modelo de *Reinforcement Learning* es la capacidad de modelar la dinámica del sistema en el que se enmarca el problema que se quiere resolver, ya sea a través de las ecuaciones que lo definen o a través de la capacidad de samplear observaciones del mismo. En sistemas cuya complejidad es alta, en general es considerablemente más barato simular la evolución de este en comparación a determinar el comportamiento exacto en cada momento.

En el caso de estudio, es necesario tener un modelo que describa el esparcimiento de un incendio sobre un paisaje en específico y más aún, el impacto de la presencia de un cortafuego en este. De esta forma se utilizó el simulador *Cell2Fire* desarrollado por (Pais, Carrasco, Martell, Weintraub, y Woodruff, 2021). Este se basa en simulación *cellullar automata*, la cual supone que un bosque se compone de N celdas, cada una homogénea y que se puede encontrar en una cantidad finita de estados posibles, por las cuales el fuego se propaga desde el centro de una celda al centro de aquellas adyacentes. Las transiciones del estado de cada celda se modela como una cadena de Markov que considera los siguientes estados:

- Disponible: La celda contiene combustible y por tanto se puede quemar de ser alcanzada por el fuego.
- Quemándose: Celda que contiene fuego y puede propagarlo a sus celdas adyacentes.
- Quemada: Celda por la cual el incendio ya pasó, no vuelve a quemarse.
- Cosechada: Celda en la cual se posiciona un cortafuego, no se quema y exhibe el mismo comportamiento que una Non-Fuel.
- Non-Fuel: Celda que no contienen material combustible y por tanto no pueden quemarse.

Existen dos elementos principales que definen el esparcimiento de un incendio en el simulador: la tasa de esparcimiento (ROS) y la intensidad del incendio. Ambos se pueden determinar en su mayoría a partir del combustible presente en una celda individual, temperatura y humedad ambientales y velocidad y dirección del viento. En particular, el simulador utiliza el *Canadian Forest Fire Behavior Prediction System*(van Wagner, 1998), *FBP* de aquí en adelante, para predecir el ROS e intensidad. Por otro lado, los combustibles que se utilizaron son aquellos del FBP, propios del ecosistema canadiense. En particular se limito su uso a los siguientes:

- Boreal Spruce: Zonas principalmente dominadas por especies coníferas, tales como: *black spruce*, *balsam fir* y *yellow birch*.
- Spruce-Lichen Woodland: zonas frescas y secas, con un drenaje bueno/excesivo. Dominado principalmente por especies como *white spruce* y *black spruce*.
- Matted Grass: zonas caracterizadas por una covertura continua de pasto, con presencia ocasional de arboles o arbustos que no modifican significativamente el comportamiento del fuego.
- Non-Fuel: Corresponde en general a cuerpos de agua o piedras.

El comportamiento estocástico del simulador se basa en dos elementos principales: punto de ignición y el escenario meteorológico. El punto de ignición en general es aleatorio, siguiendo una distribución de probabilidad forzada por el usuario. El escenario meteorológico considera la diección y velocidad del viento, temperatura y humedad, los cuales inciden en el *ROS*. El escenario asociado a una simulación específica se puede fijar o escoger desde una serie de opciones de manera aleatoria.

2.2. Marco Teórico

2.2.1. Multi-armed Bandits

Considere el siguiente problema: repetidamente se ve enfrentado a una decisión sobre k posibles opciones o acciones. Después de cada decisión se recibe una recompensa numérica proveniente de una distribución de probabilidad. El objetivo es maximizar la suma total esperada de recompensas sobre un periodo de tiempo. Cada una de las k opciones puede verse como un brazo de una máquina tragamonedas de k brazos.

El problema definido en 1.1 claramente se enmarca dentro de la descripción anterior: repetidamente se ve enfrentado a escoger una combinación particular de cortafuegos, luego de lo cual se obtiene una recompensa por la acción protectora que hace con respecto a los valores en riesgo, el negativo de la suma de celdas que se queman.

Sin embargo, se puede notar que el tomar este enfoque implica enfrentarse a una decisión sobre un conjunto demasiado grande de elecciones, $\binom{n^2}{0.05n^2}$ elecciones, donde n corresponde al número de celdas. A pesar de que la utilización de este enfoque se vuelve inpráctica para grillas bastante chicas, se utilizará como comparación más adelante.

2.2.1.1. Algoritmo epsilon-greedy

Cómo se mencionó brevemente, cada una de las k decisiones tiene una recompensa esperada, que proviene de una distribución de probabilidad. Esta recompensa esperada se define cómo:

$$q_*(a) = E(R_t|A_t = a)$$

Donde, A_t es la acción que se escoge y R_t la variable aleatoria que corresponde a la recompensa que se obtiene. Una forma simple de aproximar $q_*(a)$ es mediante la siguiente función:

$$Q_t(a) = \frac{\text{suma de recompensas obtenidas al escoger } a \text{ previo a t}}{\text{número de veces que se escogió } a \text{ previo a t}} = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}}$$

El estimador anterior es un promedio muestral que bajo la ley de los grandes numeros converge al valor real $q_*(a)$. Sin embargo, es necesario definir una regla de selección de acciones que permita que este estimador converja en un tiempo acotado. Escoger la acción a de manera que maximice $Q_t(a)$ pareciese ser una elección razonable, pero debido a que es un estimador esto solo explotaría el conocimiento actual que tenemos de las recompensas que se obtienen para cada acción, pasando por alto la posibilidad de que una acción distinta traiga mejores recompensas. Una alternativa es hacer lo anterior en la gran mayoría de los casos, pero con probabilidad ϵ elegir una acción aleatoria de las disponibles. Lo anterior se denomina un método epsilon-greedy y tiene como ventaja que a medida que aumenta el número de intentos, cada acción será sampleada una cantidad considerable de veces, garantizando que $Q_t(a)$ converja a $q_*(a)$ para cada acción.

Denotando $N_t(a) = \sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}$ y $bandit(a)$ una función que samplea la recompensa co-

rrespondiente a tomar la acción a de la distribución de recompensas. Un primer algoritmo que permite resolver el problema de *Multi-armed Bandits* es el siguiente:

Algorithm 1 Un algoritmo bandit simple

Inicializar para $a \in \{1, ..., k\}$:
$Q(a) \leftarrow 0$
$N(a) \leftarrow 0$
while $True$ **do**
$A \leftarrow \begin{cases} argmax_a Q(a) & \text{con probabilidad } 1-\epsilon \\ a \text{ acción aleatoria} & \text{con probabilidad } \epsilon \end{cases}$
$R \leftarrow bandit(A)$
$N(A) \leftarrow N(A) + 1$
$Q(A) \leftarrow Q(A)\frac{1}{N(A)}[R - Q(A)]$
end while

2.2.1.2. Selección de acciones mediante Upper-Confidence-Bound

Una algoritmo más sofisticado que permite resolver el problema descrito es la Selección de acciones mediante Upper-Confidence-Bound (UCB). En este, a la hora de escoger acciones se toma en consideración tanto la optimalidad como la incertidumbre asociada al valor estimado $Q_t(a)$. La incertidumbre se puede definir para cada una de las k acciones disponibles de la siguiente forma:

$$Incertidumbre(a) = \sqrt{\frac{\ln t}{N(a)}}$$

Este término disminuye para la acción a cada vez que se elige esta y aumenta cada vez que se elige otra. Sin embargo, este decrecimiento ó crecimiento se reduce a medida que aumenta el número de intentos, de forma que todas las acciones serán seleccionadas eventualmente, pero las acciones con valores bajos de $Q_t(a)$ ó que han sido seleccionadas muchas veces serán seleccionadas cada vez con menos frecuencia. A su vez, se controla la exploración del algoritmo de manera similar que para el método epsilon-greedy, mediante un parámetro c que pondera la incertidumbre asociada a las distintas acciones.

Algorithm 2 Algoritmo UCB

Considerar c = 2
Inicializar para $a \in \{1, ..., k\}$:
$Q(a) \leftarrow 0$
$N(a) \leftarrow 0$
while $True$ **do**
$A \leftarrow argmax_a \left[Q(a) + c\sqrt{\frac{\ln t}{N(a)}}\right]$
$R \leftarrow bandit(A)$
$N(A) \leftarrow N(A) + 1$
$Q(A) \leftarrow Q(A)\frac{1}{N(A)}[R - Q(A)]$
end while

2.2.2. Reinforcement Learning

2.2.2.1. Procesos de Decisión Markovianos

Los procesos de decisión markovianos (MDP), proveen un marco formal para tratar situaciones en que un tomador de decisiones interactúa con un sistema en pos de conseguir una meta o maximizar una cantidad. Una forma intuitiva de visualizar lo anterior es mediante la llamada interfaz agente-ambiente. Esta define al tomador de decisiones como un **agente**, el cual interactúa con un **ambiente**, que se compone de todo aquello externo al agente. De esta manera, el agente interacciona con el ambiente mediante la selección de **acciones**, en una serie de pasos discretos, el cuál en respuesta modifica su estado y entrega una **recompensa**. El objetivo del agente es maximizar la suma total de estas recompensas.

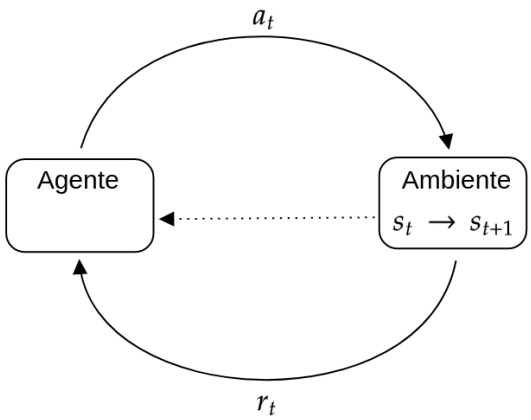


Figura 2.1: Visualización de la interfaz Agente-Ambiente.

Más formalmente, un **Proceso de Decision Markvoviano** es una 4-tupla (S, A, P, R) donde:

- S es el conjunto de estados en los que puede encontrarse el ambiente, siendo S_t el estado en que se encuentra en el tiempo t.
- A es el conjunto de acciones que puede tomar el agente, siendo A_t la acción que toma el agente en el tiempo t y $A(S_t)$ el conjunto de acciones disponibles en el estado S_t.
- R es el conjunto de posibles recompensas, siendo $r(s, a)$ la recompensa que obtiene el agente al tomar la acción a en el estado s.
- $P(S_t = s, r_t = r|S_{t-1} = s', A_{t-1} = a)$ es la probabilidad de transicionar desde el estado s' a s al tomar la acción a, obteniendo una recompensa r.

La función P define toda la dinámica del proceso. En particular estos procesos se caracterizan por presentar la **Propiedad de Markov**:

$$P(S_t = s, r_t = r|S_{t-1} = s', ..., S_0 = s_0, A_{t-1} = a, ..., A_0 = a_0) = P(S_t = s, r_t = r|S_{t-1} = s', A_{t-1} = a)$$

Por otro lado, para una tripleta (s, s', a) se puede definir la **probabilidad de transición**:

$$p(s'|s,a) = \sum_{r \in R} P(S_t = s', r_t = r | S_{t-1} = s, A_t = a)$$

Tal como se remarcó, el objetivo del agente es maximizar la suma total esperada de las recompensas. Esta cantidad se denomina **Retorno descontado** y se define:

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k$$

Donde la interacción agente-ambiente se particiona en **episodios**, cuyo término se denota mediante T y $\gamma \in [0, 1]$ es un factor de descuento.

Un aspecto fundamental de estos procesos, es como se formaliza la toma de decisiones del agente, ya que es mediante estas que obtiene recompensas. A su véz, el agente va a buscar transicionar desde un estado a otro que se considere "bueno", donde esta noción se debe definir. Así, se definen dos conceptos claves: **política** y **función de valor**.

Una **Política** es un mapeo de estados a probabilidades de selección para cada acción disponible, $\pi(a|s) : a \in A(s) \rightarrow [0, 1]$.

La **Función de valor** representa esta noción de "bondad" de un estado y corresponde al retorno esperado cuando el agente inicia en el estado s y sigue la política π:

$$V_\pi(s) = E_\pi(\sum_{k=0}^{T} \gamma^k r_{t+k+1} | S_t = s)$$

De esta función se deriva la denominada **Función estado-acción**:

$$q_\pi(s, a) = E_\pi(\sum_{k=0}^{T} \gamma^k r_{t+k+1} | S_t = s, A_t = a)$$

Es necesario remarcar que el cumplimiento del objetivo del agente, maximizar la suma de recompensas, se reduce a encontrar una política π que genere la mayor cantidad de recompensas posible (Sutton y Barto, 2005). De esta forma se vuelve aparente la necesidad de comparar políticas. $V_\pi(s)$ induce un orden parcial entre políticas, luego una política π es mejor que otra política π' sí:

$$V_\pi(s) \geq V_{\pi'}(s), \forall s \in S$$

Extendiendo lo anterior, una **política óptima** se define como aquella que es mejor que toda otra política y se denota π^*. A pesar de que no existe unicidad sobre esta noción de optimalidad, todas las políticas óptimas comparten la **misma** función de estado y estado-acción, por tanto:

$$V_*(s) = \max_\pi V_\pi(s), \forall s \in S$$

$$q_*(s, a) = \max_\pi q_\pi(s, a), \forall (s, a) \in (SxA)$$

Las ecuaciones anteriores se pueden reescribir, obteniendose la **Ecuación de Optimalidad de Bellman** en sus dos formas:

$$V_*(s) \doteq \max_{a \in A} \sum_{(s',r) \in (SxR)} p(s', r|s, a)\pi(a|s)[r + \gamma V_*(s')], \forall s \in S$$

$$q_*(s, a) \doteq \sum_{(s',r) \in (SxR)} p(s', r|s, a)\pi(a|s)[r + \gamma \max_{a' \in A(s')} q_*(s', a')], \forall (s, a) \in (SxA)$$

Ambos sistemas de ecuaciones tienen una única solución, con el detalle de que ya no es necesario especificar π, ya que ambos garantizan que se está siguiendo la política óptima π^*.

2.2.2.2. Programación Dinámica

Habiéndose definido los conceptos claves del problema de los MDP, se está en condiciones de ver como resolverlo. Un primer enfoque, es el de **Programación dinámica**. A pesar de que su utilidad en este contexto es limitada, sienta las bases para aplicar reinforcement learning como tal. Este enfoque plantea la resolución interativa de las ecuaciones de Bellman, donde se mantienen tablas con los valores para cada posible entrada, denotados por $V(s)$ ó $Q(s, a)$.

Se inicia presentando un método para resolver el sistema de ecuaciones de bellman mostrado en la sección anterior, **Evaluación de política**. Este algoritmo permite obtener un estimador de la función de valor, $v_\pi(s)$, para una política dada.

Algorithm 3 Evaluación de política, para estimar v_π

Input π, política a ser evaluada.
Parametros: $\theta \geq 0$, correspondiente al límite de precisión de la estimación.
Inicializar $V(s), \forall s \in S$ arbitrariamente, excepto si s es un estado en que termina el episodio(terminal), en cuyo caso $V(s) = 0$.
$\Delta \leftarrow 0$
while $\Delta > \theta$ **do**
  for $s \in S$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_{a \in A(s)} \pi(a|s) \sum_{(s',r) \in (SxR)} p(s', r|s, a)[r + \gamma V(s')]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  end for
end while

Teniendo una herramienta para estimar V_π se debe intentar mejorar una política π particular. Para esto se presenta el algoritmo **Mejora de Política**, que dada una política π, nos permite obtener una estrictamente mejor:

Algorithm 4 Mejora de Política, para mejorar una política π

Input π, política a mejorar.
for $s \in S$ **do**
  $\pi(s) \leftarrow \text{argmax}_{a \in A(s)} \sum_{(s',r) \in (SxR)} p(s', r|s, a)[r + \gamma V(s')]$
end for

Combinando ambos algoritmos, mediante el llamado **Iteración de Política**, es posible obtener la política óptima.

Un elemento que se ha pasado por alto, pero que sin embargo es de gran importancia, es que todos los algoritmos presentados hasta ahora y los MDP en general se basan en el conocimiento de la función P. En general, no se tiene acceso a ella totalmente o en lo absoluto, en la próxima sección se presentan algoritmos que se hacen cargo de lo anterior.

2.2.2.3. Métodos libres de modelo

Los métodos libres de modelo, tal como el nombre lo indica, no requieren un modelo que caracterice las dinámicas del ambiente. El único requisito es la capacidad de poder samplear observaciones del ambiente: secuencias de estados, acciones y recompensas obtenidas a partir de experiencia real o simulada con el ambiente. Existen dos clases de métodos dentro de esta familia, los **métodos basados en la función de política** y los **métodos de búsqueda de política**

2.2.2.3.1. Métodos basados en la función de valor

Esta clase de métodos se basan en la estimación de un proxy de la función de valor, a partir de la cual es posible construir la política óptima. Esta construcción se basa en tomár las acciones que generen una transición hacia el estado accesible que maximize la función de valor.

Métodos de Monte Carlo

Esta clase de métodos se basan en la noción presentada para el algoritmo de **Iteración de pólítica**. En el caso más general, se mantiene una tabla con los valores estimados de la funcion estado-accion $Q(s, a)$ para cada par estado-acción. La tabla contiene el promedio histórico de los retornos descontados obtenidos por el agente al pasar por cada par (s, a). En un primer paso, se busca estimar la **función estado-acción** para la política actual. Para esto, se samplean transiciones del ambiente siguiendo la política. Una vez finalizado un episodio completo los valores de la tabla se actualizan mediante la siguiente regla:

$$Q(S_t, A_t) = Q(S_t, A_t) + \frac{1}{N(S_t, A_t)}(G_t - Q(S_t, A_t))$$

Donde $N(S_t)$ corresponde al número de veces que se ha visitado el estado S_t.

En el segundo paso, se busca mejorar la política actual. Para esto, se construye una nueva

política definida como sigue:

$\forall a \in A(S_t)$:

$$\pi(a|S_t) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{\epsilon/|A(S_t)|} & \text{Si } a = \text{argmax}_{a' \in A(S_t)} Q(S_t, a') \\ \epsilon/|A(S_t)| & \text{otherwise} \end{cases}$$

En donde la construcción anterior se denomina una política ϵ-greedy con respecto a $Q(s, a)$, la cual elige una acción subóptima con probabilidad ϵ. Lo anterior en pos de incentivar la exploración del espacio de acciones.

Métodos de Diferencia temporal

Los métodos de diferencia temporal siguen la misma idea que los métodos de Monte Carlo, con la salvedad de que las actualizaciones se realizan en cada paso de un episodio:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

Donde R corresponde a la recompensa obtenida al tomar la acción A_t y α corresponde a la tasa de aprendizaje. Las elecciones de A_t y A_{t+1} nuevamente se toman en base a una política ϵ-greedy con respecto a $Q(s, a)$.

Se está en condiciones de presentar el primer algoritmo de diferencia temporal, que será utilizado más adelante. Este se llama **Q-learning** (Watkins y Dayan, 1992):

Algorithm 5 Q-learning

Inicializar $Q(s, a)$ aleatoriamente. Excepto si s es terminal, en cuyo caso $Q(s, .) = 0$.
for *episode* $\in$ *episodes* **do**
  for $t \leq T$ **do**
    Elegir A en $A(S)$ usando política ϵ-greedy con respecto a $Q(s, a)$.
    Tomár acción A, observar R y S'.
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R + \gamma max_{a \in A(S)} Q(S', a) - Q(S, A)]$
    $S \leftarrow S'$
    $t \leftarrow t + 1$
  end for
end for

2.2.3. Deep Reinforcement Learning

Hasta el momento los estimadores de Q y V eran tablas, en donde cada estado $s \in S$ o par $(s, a) \in S \times A$ tenía una entrada propia. Esto es útil cuando el espacio de estados y de acciones es relativamente pequeño, pero de no ser así la carga compútacional impide su implementación.

Se vuelve natural la utilización de formas funcionales parametrizadas para representar Q y V, denotadas en ambos casos por $V(s, \theta)$ y $Q(s, a, \theta)$, donde $\theta \in R^d$ es un vector de parámetros, cuyo ajuste se lleva a cabo mediante datos generados mediante la interacción entre el agente y ambiente. Un caso particular que ha recibido notable atención durante

los últimos años ha sido la utilización de Redes Neuronales con éste fin, denominado **Deep Reinforcement Learning**. Esta área de Deep Learning ha permitido obtener resultados cercanos o superiores a la capacidad humana en tareas como ajedrez (H. T. S. J. Silver D., 2016), Go (H. A. M. C. Silver D., 2016), Atari (Sutton y Barto, 2005), etc.

Esta sección consta de una breve introducción a los conceptos claves de *Deep Learning* y luego se presentan los algoritmos utilizados para el desarrollo de este trabajo.

2.2.3.1. Deep Learning

La inteligencia artificial (IA) ha sido uno de los campos de más rápido crecimiento en la última década, con una variedad de aplicaciones en áreas diversas como la salud, las finanzas, el comercio minorista y el transporte. El aprendizaje automático (ML) ha sido la fuerza impulsora detrás de gran parte de los avances recientes en IA. El aprendizaje profundo, un subconjunto de ML, ha demostrado ser particularmente poderoso para resolver problemas complejos en procesamiento de lenguaje natural, visión por computadora y reconocimiento de voz, entre otros.

El aprendizaje profundo es un subconjunto de aprendizaje automático que implica entrenar redes neuronales artificiales para reconocer patrones complejos en datos. Estas redes neuronales están inspiradas en la estructura del cerebro humano, que está compuesto por neuronas interconectadas que procesan y transmiten información. Los algoritmos de aprendizaje profundo pueden aprender a reconocer patrones en grandes conjuntos de datos ajustando los pesos y sesgos de estas redes neuronales a través de un proceso llamado backpropagation.

El aprendizaje profundo tiene sus raíces en la década de 1940, cuando Warren McCulloch y Walter Pitts propusieron un modelo matemático del cerebro llamado neurona de McCulloch-Pitts. En las décadas de 1950 y 1960, el desarrollo del algoritmo de perceptrón y el algoritmo de backpropagation sentaron las bases de las redes neuronales modernas. Sin embargo, el aprendizaje profundo se mantuvo relativamente inactivo hasta principios de la década de 2000, cuando la disponibilidad de grandes conjuntos de datos y potentes unidades de procesamiento gráfico (GPU) hizo posible entrenar redes neuronales profundas con muchas capas.

El aprendizaje profundo ha hecho un progreso notable en los últimos años, superando las técnicas tradicionales de aprendizaje automático en diversas tareas como la clasificación de imágenes, el reconocimiento de voz y el procesamiento de lenguaje natural. En 2012, un modelo de aprendizaje profundo llamado AlexNet (Krizhevsky, Sutskever, y Hinton, 2012) ganó el desafío de reconocimiento visual a gran escala ImageNet, reduciendo la tasa de error en una cantidad significativa en comparación con enfoques anteriores. Desde entonces, el aprendizaje profundo se ha convertido en el método de vanguardia en muchas aplicaciones, con una investigación que avanza rápidamente.

Redes Neuronales

Las redes neuronales son un tipo de modelo de aprendizaje profundo utilizado para aprender relaciones no lineales en los datos. Las redes neuronales están inspiradas en la estructura y funcionamiento del cerebro humano, donde las neuronas se comunican entre sí para pro-

cesar información. De manera similar, una red neuronal está compuesta por una colección de unidades interconectadas llamadas neuronas artificiales, que trabajan en conjunto para resolver un problema específico.

La estructura de una red neuronal está compuesta por tres tipos de capas: la capa de entrada, las capas ocultas y la capa de salida. La capa de entrada recibe los datos de entrada, que se procesan a través de las capas ocultas para producir la salida final en la capa de salida. Cada neurona en una capa está conectada a todas las neuronas en la capa siguiente, formando una red densamente conectada.

Funciones de activación

Las funciones de activación son utilizadas por las neuronas para determinar su salida en función de su entrada. Las funciones de activación son no lineales, lo que permite a la red aprender patrones complejos en los datos. Algunas funciones de activación comunes incluyen la función sigmoide, la función ReLU y la función tangente hiperbólica.

La función sigmoide es una función sigmoide logística definida como:

$$f(z) = \frac{1}{1 + e^{-z}}$$

La función ReLU (Rectified Linear Unit) es una función de activación no lineal definida como:

$$f(z) = \max(0, z)$$

La función tangente hiperbólica es una función sigmoide definida como:

$$f(z) = \tanh(z)$$

Las ecuaciones que definen el comportamiento de una red neuronal se pueden expresar matemáticamente como una serie de operaciones matriciales. Sea X una matriz de entrada con dimensiones (n, m), donde n es el número de observaciones y m es el número de características, W una matriz de pesos con dimensiones (m, p) y b un vector de sesgos con dimensiones $(1, p)$. La salida de una capa se puede calcular como:

$$Z = XW + b$$

La salida Z se pasa a través de una función de activación $f(Z)$ para producir la salida de la capa siguiente. La salida de la última capa se utiliza como la salida final de la red neuronal.

2.2.3.1.1. Algoritmo de Backpropagation

El algoritmo de backpropagation es un algoritmo utilizado en el aprendizaje supervisado para entrenar redes neuronales introducido en (Rumelhart, Hinton, y Williams, 1986). El objetivo del algoritmo es ajustar los pesos de la red neuronal para minimizar una función de costo que mide la diferencia entre la salida predicha y la salida real.

Antes de explicar el algoritmo de backpropagation, es importante comprender el proceso

de propagación hacia adelante (forward propagation) utilizado en el cálculo de la salida de la red neuronal. En la propagación hacia adelante, la entrada se pasa a través de la red neuronal capa por capa, calculando las salidas intermedias de cada capa hasta llegar a la capa de salida.

Suponga que se tiene una red neuronal con una capa de entrada, dos capas ocultas y una capa de salida. La entrada se representa como una matriz X con dimensiones (n, m), donde n es el número de observaciones y m es el número de características. Cada capa i de la red neuronal tiene una matriz de pesos $W^{(i)}$ con dimensiones (m_{i-1}, m_i) y un vector de sesgos $b^{(i)}$ con dimensiones $(1, m_i)$. La salida de cada capa i se calcula como:

$$Z^{(i)} = XW^{(i)} + b^{(i)}$$

A continuación, se pasa la salida $Z^{(i)}$ a través de una función de activación $f^{(i)}$ para producir la salida $A^{(i)}$ de la capa i:

$$A^{(i)} = f^{(i)}(Z^{(i)})$$

La salida de la última capa se utiliza como la salida final de la red neuronal.

Backpropagation

El algoritmo de backpropagation utiliza el gradiente descendente para ajustar los pesos de la red neuronal. El gradiente descendente es un método de optimización que busca ajustar los pesos de la red neuronal para minimizar una función de costo J.

En la backpropagation, el gradiente descendente se aplica a través de la red neuronal en sentido inverso, comenzando desde la capa de salida y retrocediendo hacia la capa de entrada. El algoritmo utiliza la regla de la cadena para calcular la derivada parcial de la función de costo con respecto a los pesos de la red neuronal.

Sea J la función de costo y $w^{(i,j)}$ el peso de la conexión entre la neurona i de la capa anterior y la neurona j de la capa actual. Entonces, la derivada parcial de J con respecto a $w^{(i,j)}$ se puede expresar como:

$$\frac{\partial J}{\partial w^{(i,j)}} = \frac{\partial J}{\partial z^{(j)}} \frac{\partial z^{(j)}}{\partial w^{(i,j)}}$$

Donde $z^{(j)}$ es la entrada a la neurona j de la capa actual. La primera derivada parcial se refiere a la contribución de la neurona j a la función de costo J, y la segunda derivada parcial se refiere a la contribución de la neurona i a la entrada de la neurona j.

La derivada parcial $\frac{\partial J}{\partial z^{(j)}}$ se puede calcular utilizando la regla de la cadena:

$$\frac{\partial J}{\partial z^{(j)}} = \sum_k \frac{\partial J}{\partial z^{(k)}} \frac{\partial z^{(k)}}{\partial z^{(j)}}$$

Donde la suma se realiza sobre todas las neuronas k que están conectadas a la neurona j. Esto significa que la contribución de la neurona j a la función de costo J depende de la

contribución de todas las neuronas k a la entrada de la neurona j.

La derivada parcial $\frac{\partial z^{(j)}}{\partial w^{(i,j)}}$ se puede expresar como el valor de la entrada de la neurona i de la capa anterior multiplicado por el gradiente de la función de activación $f^{(j)}$:

$$\frac{\partial z^{(j)}}{\partial w^{(i,j)}} = a^{(i)} \frac{\partial f^{(j)}}{\partial z^{(j)}}$$

El gradiente de la función de activación $f^{(j)}$ se puede calcular a partir de la derivada de $f^{(j)}$ con respecto a su entrada $z^{(j)}$.

Finalmente, el gradiente descendente se utiliza para actualizar los pesos de la red neuronal en la dirección opuesta del gradiente. El proceso se repite hasta que se alcanza una convergencia satisfactoria de la función de costo J.

2.2.3.1.2. Redes Neuronales Convolucionales

Las Redes Neuronales Convolucionales(LeCun, Bengio, y Hinton, 2015) son un tipo de red neuronal artificial que han tenido un gran éxito en tareas de reconocimiento de imágenes y videos. Están inspiradas en la estructura y función de la corteza visual del cerebro, que contiene células que responden a estímulos visuales específicos en pequeñas regiones localizadas del campo visual.

La Convolución

La operación de convolución es una operación matemática que consiste en aplicar un filtro o núcleo a una imagen o matriz de entrada para producir una matriz de salida. El filtro es una pequeña matriz de parámetros que se mueve por la imagen de entrada pixel por pixel, y en cada posición, los elementos del filtro se multiplican con los elementos correspondientes de la imagen de entrada y los resultados se suman para producir un único valor de salida. Este proceso se repite para cada posición posible del filtro sobre la imagen de entrada, lo que resulta en una nueva matriz de valores de salida.

Formalmente, la operación de convolución se puede definir de la siguiente manera:

$$(I * K)i, j = \sum m \sum_{n} I_{i-m,j-n} K_{m,n} \tag{2.1}$$

Donde I es la imagen de entrada, K es el filtro y * denota la operación de convolución. Los índices i y j representan las coordenadas espaciales del mapa de la matriz de salida, mientras que los índices m y n representan las coordenadas espaciales del filtro. La suma se realiza sobre todos los valores posibles de m y n que son consistentes con el tamaño del filtro.

En la práctica, la operación de convolución se implementa generalmente mediante multiplicación de matrices para mejorar la eficiencia. Primero se convierte la imagen de entrada en una matriz de tamaño (H x W x C), donde H, W y C son la altura, la anchura y el número de canales de la imagen de entrada, respectivamente. El filtro también se convierte en una matriz de tamaño (F x F x C), donde F es el tamaño del filtro. La operación de convolución se realiza mediante la reorganización de la matriz de la imagen de entrada en un vector (HW x C), y la matriz del filtro en una matriz (FF x C). El producto punto de estas dos matrices

produce una matriz (HW x FF), que luego se reorganiza de nuevo en una matriz de salida de dimensiones (H x W x 1).

Filtros

Los filtros utilizados en las CNNs están diseñados para detectar características visuales específicas, como bordes, esquinas, texturas y formas. Por lo general, son matrices cuadradas pequeñas de tamaño F x F, donde F es generalmente un número impar como 3, 5 o 7. Cada elemento del filtro representa un peso o coeficiente que determina cuánto contribuye el píxel correspondiente en la imagen de entrada a la matriz de salida. Los valores del filtro se aprenden durante el proceso de entrenamiento de la CNN, utilizando backpropagation para ajustar los pesos de manera que se minimice el error entre las salidas predichas y reales.

Capas de Pooling

Las capas de pooling se utilizan típicamente en CNNs para reducir la resolución de las matrices producidas por las capas convolucionales. Esto disminuye las dimensiones espaciales de las matrices y hace que la red sea más resistente a las variaciones en la imagen de entrada. La operación de pooling más común es el max pooling, que toma el valor máximo en cada región local del mapa de características. Formalmente, la operación de max pooling se puede definir como sigue:

$$(I_{max})i, j = \max_{m,n} I_{i+m,j+n} \tag{2.2}$$

Donde I es la martriz de entrada, I_{max} es la matriz de salida después del max pooling, y los índices i y j representan las coordenadas espaciales de la matriz de salida, mientras que los índices m y n representan las coordenadas espaciales de la región de pooling. El valor máximo dentro de cada región de pooling se toma como el valor de salida para esa región.

Otras operaciones de pooling incluyen el average pooling, que toma el valor promedio en cada región local, y el L2 pooling, que toma la raíz cuadrada de la suma de cuadrados de cada elemento en la región de pooling. Las capas de pooling también pueden ser utilizadas para reducir el número de parámetros en la red, y para introducir un grado de invariancia translacional en la red.

Habiendose definido los conceptos claves de *Deep Learning* en esta sección, se pasa de lleno a presentar los algoritmos de *Deep Reinforcement Learning* utilizados en este trabajo.

2.2.3.2. Métodos basados en la función de valor

En 2.2.2.3.1 se introdujo la intuición base de los Métodos basados en la función de valor. En particular, la forma funcional parametrizada que se utilizará para aproximar V y Q será una *Red Neuronal Convolucional (CNN).*

Para denotar las aproximaciones correspondientes mediantes redes neuronales, se utilizará la notación $Q(s, a, \theta)$ y $V(s, \theta)$, donde θ corresponde a los pesos de la *CNN*. Todos los algoritmos descritos en esta sección siguen el siguiente esquema:

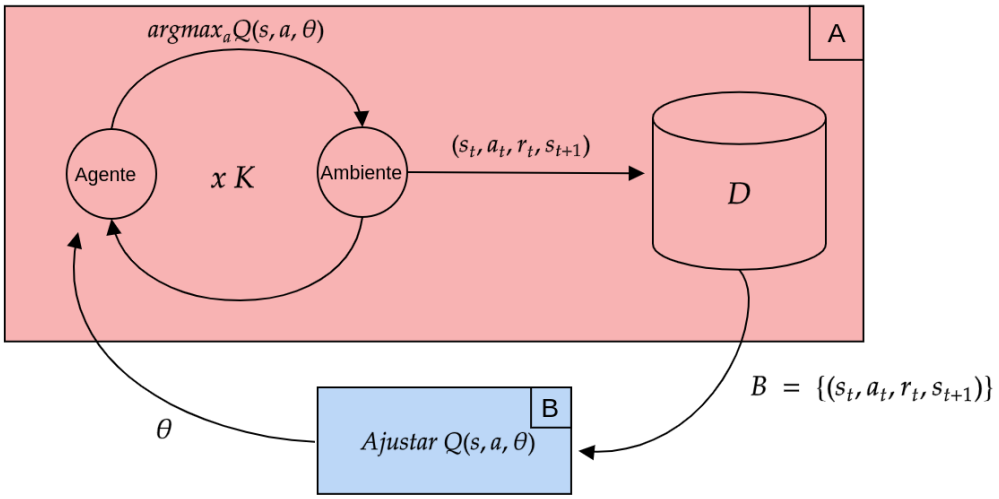


Figura 2.2: Visualización de la interfaz Agente-Ambiente.

Donde estos llevan acabo el procedimiento A un k número de veces, recopilando experiencias entre el agente y ambiente en un buffer D, interactuando según una política epsilon-greedy sobre $Q(s, a, \theta)$. Luego en B, se ajusta el estimador $Q(s, a, \theta)$, sampleando datos de esta memoria. Finalmente, se actualizan los parámetros θ y se repite el ciclo hasta lograr convergencia.

2.2.3.2.1. Deep Q-Learning

El primer algoritmo a introducir es el denominado *Deep Q-Learning*(Mnih et al., 2013). Este combina las nociones de aproximacion mediante formas funcionales parametrizadas con las actualizaciones que utiliza *Q-Learning*, presentadas en 2.2.2.3.1. De esta forma, los ajustes serán:

$$Q(S_t, A_t, \theta) = Q(S_t, A_t, \theta) + \alpha[R + \gamma max_{a \in A(S_{t+1})} Q(S_{t+1}, a, \theta)]$$

Donde, α es una tasa de aprendizaje y γ la tasa de descuento usual. Es importante destacar que la actualización anterior es una medida de que tan bueno es el ajuste de la aproximación $Q(S_t, A_t, \theta)$, pero no evalúa que tan buena es la política asociada. En particular, en el límite cuando converge al valor real de $q_*(S_t, A_t)$, se tiene garantía de que la política asociada es la óptima. Sin embargo, si el estimador difiere del valor real, la política puede estar arbitrariamente lejos de la óptima.

Experience Replay

Una ventaja importante del algoritmo anterior, y de todos lo que se presentarán en esta sección, es que no hay ningún supuesto con respecto a la distribución desde donde se samplean las transiciones (s_t, a_t, r_t, s_{t+1}). En particular, no tienen que haber sido necesariamente generadas por la política implícita del actual valor de $Q(S_t, A_t, \theta)$. Una forma de aprovecharse de lo anterior es el denominado *Experience Replay* (Lin, 1992): en lugar de utilizar las expe-

riencias generadas mediante la interacción ambiente-agente $e_t = (s_t, a_t, r_t, s_{t+1})$ de inmediato y descartarlas, se van a almacenar en un buffer de memoria $B_t = \{e_1, ..., e_t\}$.

Este procedimiento tiene varias ventajas, por un lado esto permite ir acumulando experiencias mediante la interacción entre el agente y ambiente sin tener que deshacerse de ellas al actualizar los parámetros θ, disminuyendo la varianza del estimador. Por otro, se pueden incorporar experiencias generadas por distintas fuentes de datos, no solo de interacciones con el ambiente. También, las redes neuronales en general asumen que cada batch de entrenamiento es independiente e identicamente distribuido, de manera que si se utilizan solo las experiencias recién generadas, la muestra estaría fuertemente correlacionada puesto que serían pasos consecutivos de una misma interacción agente-ambiente. Finalmente, estas experiencias también dependen de la política actual, por tanto una modificación sobre la red genera de por si un cambio en la experiencias, lo cual puede llevar a que la aproximación se quede atascada en óptimos locales(Lin, 1992).

Target Network

Por otro lado, la utilización de la red que interactúa con el ambiente para generar las actualizaciones puede provocar inestabilidad en el aprendizaje. Por lo tanto, se va a utilizar una copia de esta red para realizarlas, denominada *target network* y que se actualizará cada C pasos, copiando los pesos de la red que interactúa. La notación para esta red será $Q(s, a, \theta^-)$ (Mnih et al., 2015).

De esta manera, las actualizaciones seguirán la forma:

$$Q(S_t, A_t, \theta) = Q(S_t, A_t, \theta) + \alpha[R + \gamma max_{a \in A(S_{t+1})} Q(S_{t+1}, a, \theta^-)]$$

Donde cada C pasos, $\theta^- = \theta$. A continuación se presenta el algoritmo completo:

Algorithm 6 Deep Q-Learning con Experience Replay

Input: Función estado-accion $Q(s, a, \theta)$ parametrizada por una CNN. Tasas de aprendizaje, α.
Inicializar parámetros θ y memoria D con capacidad N. Inicializar $\theta^- = \theta$.
for *episode* $\in$ *episodes* **do**
  $t' \leftarrow 0$
  for $t \leq T$ **do**
    Elegir a_t según $\pi(a|S_t) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{\epsilon/|A(S_t)|} & \text{Si } a = \text{argmax}_{a' \in A(S_t)} Q(S_t, a', \theta) \\ \epsilon/|A(S_t)| & \text{otherwise} \end{cases}$.
    Ejecutar a_t en el ambiente y observar r_t y s_{t+1}.
    Guardar $e_t = (s_t, a_t, r_t, s_{t+1})$ en D.
    Samplear un minibatch de transiciones (s_j, a_j, r_j, s_{j+1}) desde D.
    $y_j \leftarrow \begin{cases} r_j & \text{Si } s_{j+1} \text{ terminal.} \\ r_j + \gamma \max_{a' \in A(s_{j+1})} Q(s_{j+1}, a', \theta^-) & \text{Si no.} \end{cases}$
    Minimizar $Loss = MSE(Q(s_j, a_j, \theta), y_j)$.
    Si $t' \,\%\, C = 0$, $\theta^- \leftarrow \theta$.
    $t' \leftarrow t' + 1$
  end for
end for

2.2.3.2.2. Double Deep Q-Learning

Un problema con el algoritmo anterior, es que suele sobrestimar los valores de la función $Q(s, a, \theta)$. Esto se debe a que incluye un paso de maximización sobre los valores estimados de la función, lo cual tiende más a sobrestimar que a subestimar predicciones. Aún más, en (Hasselt, Guez, y Silver, 2015) demuestran que la estimación de errores de cualquier tipo, lo que corresponde al caso de Deep Q-Learning, en general inducen un sesgo positivo independiente de la fuente de ruido, provocando una sobre-estimación de los valores $Q(s, a, \theta)$.

Las actualizaciones en *Double Deep Q-Learning* se realizarán acorde a:

$$Y_t = R_{t+1} + \gamma Q(S_{t+1}, argmax_{a \in A(S_{t+1})} Q(S_{t+1}, a, \theta)\theta^-)$$

Donde la selección se realiza según la red con pesos θ y la evaluación segun la *target network* con parametros θ^-. A continuación se presenta el algoritmo completo:

Algorithm 7 Double Deep Q-Learning con Experience Replay

Input: Función estado-accion $Q(s, a, \theta)$ parametrizada por una CNN. Tasas de aprendizaje, α.
Inicializar parámetros θ y memoria D con capacidad N. Inicializar $\theta^- = \theta$.
for $episode \in episodes$ **do**
  $t' \leftarrow 0$
  for $t \leq T$ **do**
    Elegir a_t según $\pi(a|S_t) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{\epsilon/|A(S_t)|} & \text{Si } a = \text{argmax}_{a' \in A(S_t)} Q(S_t, a', \theta) \\ \epsilon/|A(S_t)| & \text{otherwise} \end{cases}$.
    Ejecutar a_t en el ambiente y observar r_t y s_{t+1}.
    Guardar $e_t = (s_t, a_t, r_t, s_{t+1})$ en D.
    Samplear un minibatch de transiciones (s_j, a_j, r_j, s_{j+1}) desde D.
    $y_j \leftarrow \begin{cases} r_j & \text{Si } s_{j+1} \text{ terminal.} \\ r_j + \gamma Q(s_{j+1}, argmax_{a \in A(s_{j+1})} Q(s_{j+1}, a, \theta)\theta^-) & \text{Si no.} \end{cases}$
    Minimizar $Loss = MSE(Q(s_j, a_j, \theta), y_j)$.
    Si $t' \,\%\, C = 0$, $\theta^- \leftarrow \theta$.
    $t' \leftarrow t' + 1$
  end for
end for

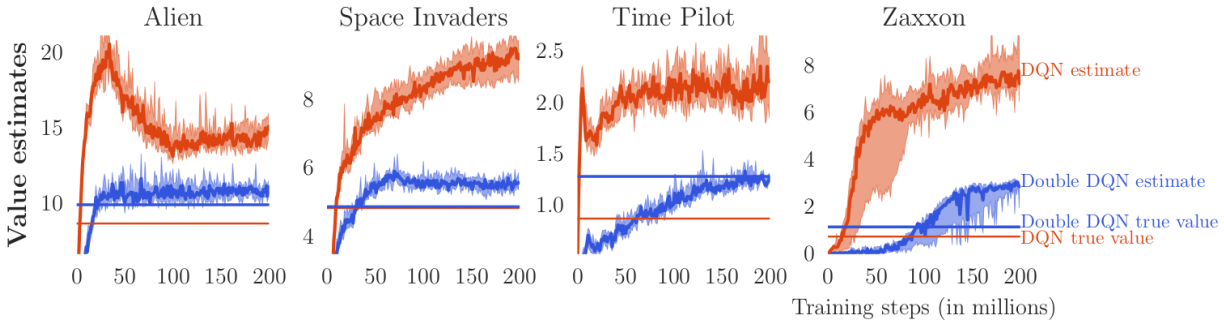


Figura 2.3: Sobre-estimación de $Q(s, a, \theta)$ obtenida en (Hasselt et al., 2015).

En la Figura 2.3 se muestra la situación descrita, donde los valores reales de $q_*(s, a)$ están muy por debajo de los obtenidos mediante Deep Q-Learning. En particular, en Deep Q-Learning se utilizan los valores provenientes de la misma red para tanto seleccionar las acciones como para evaluar estas. Para prevenir lo anterior, se puede desacoplar la selección de la evaluación de acciones. Para esto, se aprovechará que se estaba utilizando una *target network*.

2.2.3.2.3. Dueling Double Deep Q-Learning

Una mejora sustancial a los algoritmos anteriores se obtiene cuando se utiliza una arquitectura de red que permite estimar por separado la funcion de valor $V(s, \theta)$ y la función de ventaja $A(s, a, \theta)$, cuya versión en el contexto de *Deep Reinforcement Learning* se da a continuación:

$$A(s, a, \theta) = Q(s, a, \theta) - V(s, \theta)$$

En esta arquitectura, se utilizan una serie de capas comunes, desde las cuales salen dos flujos, uno corresponde a la estimación de $V(s, \theta)$ y otro a la estimación de $A(s, a, \theta)$ para cada acción. La estimación separada de ambas cantidades le permite a un agente identificar estados "valiosos", sin tener que aprender el efecto individual de cada acción en el. Esto es útil cuando existen estados que no afectan al ambiente de ninguna manera relevante. Para estos estados, va a ser considerablemente más eficiente aprender un estimador de $V(s, \theta)$(Wang, Freitas, y Lanctot, 2015).

Ahora, para la selección y evaluación de acciones se seguirá utilizando la función $Q(s, a, \theta)$. La forma más natural de construir esta función sería reutilizar la ecuación anterior:

$$Q(s, a, \theta) = V(s, \theta) + A(s, a, \theta)$$

Sin embargo, esta ecuación sufre de no identificabilidad. Este problema consiste en que dado $Q(s, a, \theta)$, no se puede recuperar directamente $V(s, \theta)$ y $A(s, a, \theta)$ de manera única, debido a que si es que se suma una constanta a a $V(s, \theta)$ y se resta a $A(s, a, \theta)$, la ecuación se mantiene para cualquier valor de a. Lo anterior conlleva problemas de rendimiento, ya que múltiples valores de estas funciones cumplirían la igualdad(Wang et al., 2015). Este problema puede resolverse parcialmente agregando un término a la ecuación:

$$Q(s, a, \theta) = V(s, \theta) + \left(A(s, a, \theta) - \frac{1}{|A(s)|} \sum_{a'} A(s, a', \theta)\right)$$

De esta manera el algoritmo completo es:

Algorithm 8 Dueling Networks con Double Deep Q-Learning con Experience Replay

Input: Función estado-accion $Q(s, a, \theta)$ parametrizada por una CNN. Tasas de aprendizaje, α.
Inicializar parámetros θ y memoria D con capacidad N. Inicializar $\theta^- = \theta$.
for *episode* $\in$ *episodes* **do**
  $t' \leftarrow 0$
  for $t \leq T$ **do**
    Elegir a_t según $\pi(a|S_t) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{\epsilon/|A(S_t)|} & \text{Si } a = \text{argmax}_{a' \in A(S_t)} Q(S_t, a', \theta) \\ \epsilon/|A(S_t)| & \text{otherwise} \end{cases}$.
    Ejecutar a_t en el ambiente y observar r_t y s_{t+1}.
    Guardar $e_t = (s_t, a_t, r_t, s_{t+1})$ en D.
    Samplear un minibatch de transiciones (s_j, a_j, r_j, s_{j+1}) desde D.
    $Q(s_{j+1}, ., \theta) \leftarrow V(s_j, \theta) + \left(A(s_j, ., \theta) - \frac{1}{|A(s_j)|} \sum_{a'} A(s_j, a', \theta)\right)$
    $Q(s_{j+1}, ., \theta^-) \leftarrow V(s_{j+1}, \theta^-) + \left(A(s_{j+1}, ., \theta^-) - \frac{1}{|A(s_{j+1})|} \sum_{a'} A(s_{j+1}, a', \theta^-)\right)$
    $y_j \leftarrow \begin{cases} r_j & \text{Si } s_{j+1} \text{ terminal.} \\ r_j + \gamma Q(s_{j+1}, argmax_{a \in A(s_{j+1})} Q(s_{j+1}, a, \theta)\theta^-) & \text{Si no.} \end{cases}$
    Minimizar $Loss = MSE(Q(s_j, a_j, \theta), y_j)$.
    Si $t' \,\%C = 0$, $\theta^- \leftarrow \theta$.
    $t' \leftarrow t' + 1$
  end for
end for

Prioritized Experience Replay

Hasta el momento se ha estado sampleando de la memoria de experiencias de manera uniforme, recuperando todas las experiencias con igual probabilidad, sin tomar en cuenta que pueden haber algunas que sean más importantes que otras. Prioritized experience replay (Schaul, Quan, Antonoglou, y Silver, 2015) es un metodo que permite samplear de la memoria poniendo énfasis en aquellas experiencias de las que se puede aprender más, siendo un proxy de esto la magnitud del error temporal (TD-error):

$$\delta = Q(S_t, A_t, \theta) + \alpha[R + \gamma Q(S_{t+1}, A_{t+1}, \theta) - Q(S_t, A_t, \theta)]$$

Al generarse una actualización sobre θ en base a un batch de experiencias, el TD-error asociado cambia y por tanto se actualizan las prioridades de cada experiencia individual. En particular, al agregarse una nueva experiencia a la memoria D, a esta se le asigna el máximo error temporal hasta el momento, ya que el suyo es desconocido y también se quiere garantizar que cada experiencia sea sampleada al menos una vez.

En primer lugar se define la prioridad de una experiencia i cualquiera como:

$$p_i = |\delta_i| + \epsilon$$

Donde ϵ previene casos bordes en donde $\delta_i = 0$. Luego, las probabilidades de sampleo se

definen como:

$$P(i) = \frac{p_i^\phi}{\sum_k p_k^\phi}$$

Siendo ϕ un exponente que define cuanta priorización se utiliza, volviendose al caso uniforme si $\phi = 0$.

Finalmente, es necesario corregir la distribución mediante *importance sampling* [4] ya que de otra manera estaría sesgada(Schaul et al., 2015). Se utilizan como pesos:

$$w_i = \Big(\frac{1}{N} \cdot \frac{1}{P(i)}\Big)^\eta,$$

Donde $\eta = 1$ compensa completamente la no uniformidad de la distribución. Se pueden introducir estos pesos utilizando $w_i \cdot \delta_i$ como actualización en lugar de δ_i. Por último, se normalizan estos pesos por $\frac{1}{max_i w_i}$ para alcanzar mayor estabilidad.

Es necesario notar que Prioritized experience replay se puede incorporar de manera trivial en todos los algoritmos introducidos en la sección 2.2.3.2, por tanto no se explicitará esta.

2.2.3.2.4. Aprendizaje por demostración

Existen casos en que se tiene acceso a datos de un controlador previo del sistema, los cuales pueden guiar al agente para que se mueva en una porción del espacio de soluciones que ya es razonablemente bueno. En este caso particular, existen soluciones alternativas al problema de la asignación de cortafuegos que ya generan soluciones buenas, aunque subóptimas. Tal es el caso del algoritmo que se introduce en la sección 2.5.1. A continuación se presenta la metodología para llevar a cabo aprendizaje por demostraciones presentada por (Hester et al., 2018).

Para hacer uso de demostraciones, se pasa por una etapa de pre-entrenamiento, cuyo objetivo es que el agente aprenda a imitar al demostrador. En esta, se samplean batches de demostraciones, se guardan en la memoria y se actualiza la red en base a 4 funciones de pérdida:

$$JE(Q) = max_{a \in A(s)}\Big[Q(s,a,\theta) + l(a_E, a)\Big] - Q(s,a,\theta)$$

Donde a_E es la acción que tomó el demostrador en el estado s y $l(a_E, a)$ es una función de margen que es 0 cuando $a = a_E$ y un valor positivo sinó, esto con el objetivo de que las acciones tomadas por el demostrador siempre tengan un valor marginalmente mayor al resto.

$$J_n(Q) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... + max_{a \in A(s)} \gamma^n Q(s_{t+n}, a, \theta) - Q(s,a,\theta)$$

$$J_{DQ}(Q) = r + \gamma \max_{a' \in A(s_{t+1})} Q(s_{t+1}, a', \theta) - Q(s_t, a, \theta)$$

$$J_{L2}(Q) = \sum_{w \in \theta} w^2$$

Obteniendose la función de pérdida global:

[4] Método que permite samplear desde una destribución distinta a aquella que generó los datos (Hanna, Niekum, y Stone, 2018)

$$J(Q) = J_{DQ}(Q) + \lambda_1 J_n(Q) + \lambda_2 JE(Q) + \lambda_3 J_{L2}(Q)$$

Después de la fase de pre-entrenamiento, el agente interactúa con el ambiente, generando experiencias que se almacenan en D. Al llenarse D, solo se sobrescriben las experiencias generadas, manteniendose la porción que contiene las demostraciones intacta.

A continuación se presenta el algoritmo completo:

Algorithm 9 Deep Q-Learning con Experience Replay desde Demostraciones

Input: Función estado-accion $Q(s, a, \theta)$ parametrizada por una CNN. Tasas de aprendizaje, α.
Inicializar parámetros θ y memoria D con capacidad N con demostraciones. Inicializar $\theta^- = \theta$.
for $t \in 1, 2, ..., k$ **do**
  Samplear un minibatch de demostraciones (s_j, a_j, r_j, s_{j+1}) desde D.
  Calcular $J(Q)$ usando target network.
  Minimizar $J(Q)$
  Si $t' \,\%C = 0$, $\theta^- \leftarrow \theta$.
end for
for *episode* $\in$ *episodes* **do**
  $t' \leftarrow 0$
  for $t \leq T$ **do**
    Elegir a_t según $\pi(a|S_t) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{\epsilon/|A(S_t)|} & \text{Si } a = \text{argmax}_{a' \in A(S_t)}\, Q(S_t, a', \theta) \\ \epsilon/|A(S_t)| & \text{otherwise} \end{cases}$.
    Ejecutar a_t en el ambiente y observar r_t y s_{t+1}.
    Guardar $e_t = (s_t, a_t, r_t, s_{t+1})$ en D.
    Samplear un minibatch de transiciones (s_j, a_j, r_j, s_{j+1}) desde D.
    Calcular $J(Q)$ usando target network.
    Minimizar $J(Q)$
    Si $t' \,\%C = 0$, $\theta^- \leftarrow \theta$.
    $t' \leftarrow t' + 1$
  end for
end for

La utilización de demostraciones ha sido utilizada en contextos muy similares como en (Hammond et al., 2020) y (Rocholl, 2021). Las demostraciones utilizadas para este trabajo se generaron a través del algoritmo presentado en la sección 2.5.1. En Anexo A se presenta una muestra de los ejemplos utilizados.

Capítulo 3

Modelamiento

Un elemento crucial para que un proceso de aprendizaje en reinforcement learning sea exitoso, es entregarle al agente una representación adecuada del ambiente. Esta representación debe tener al menos la información suficiente para que el agente pueda inferir una porción considerable del estado del sistema, en particular los cambios inmediatos que se generan por medio de su toma de acciones.

En esta línea, se tomó la decisión de diseñar una representación basada en *vision grids* (Shantia, Begue, y Wiering, 2011), donde estas últimas se pueden entender como "una foto instantánea del ambiente desde el punto de vista del agente"(Knegt, Drugan, y Wiering, 2018).

En particular, el agente va a interactuar con el ambiente por una serie de pasos, en cada cual se marca un único cortafuego, hasta marcar un 5 % del total de celdas. Posterior a esto, se le entregará la información correspondiente a cuales fueron las celdas que se marcaron a Cell2Fire y se simulará una serie de incendios sobre el bosque con los cortafuegos que se escogieron.

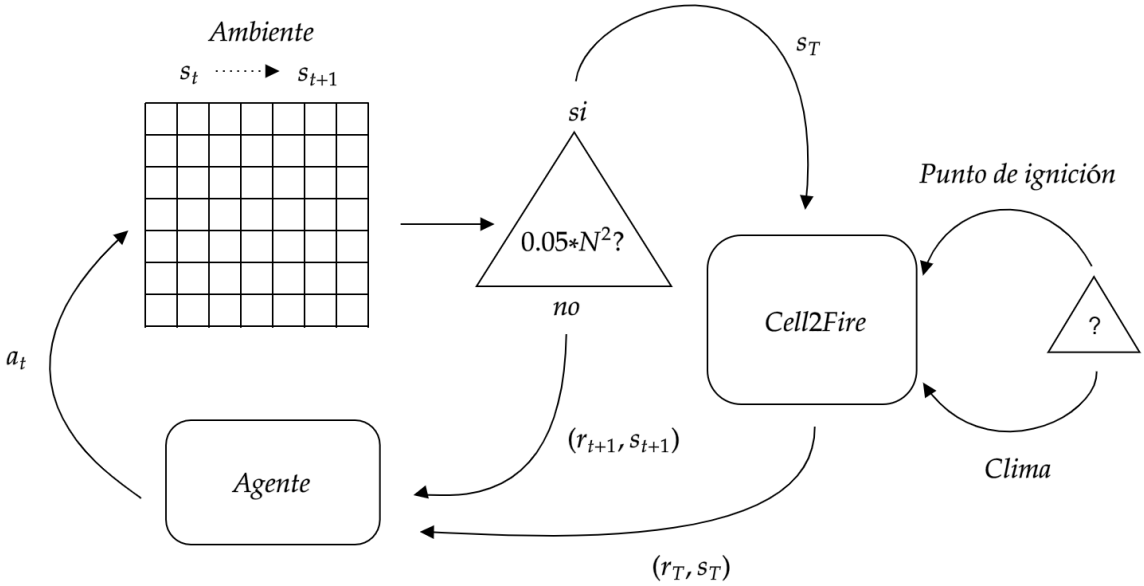


Figura 3.1: Diagrama Agente-Ambiente para Full Grid V1 y V2.

Se diseñaron dos modelamientos alternativos, "Full Grid V1" y "Full Grid V2", los cuales

se presentan a continuación.

3.1. Full Grid V1

3.1.1. Espacio de estados

El espacio de estados, S, está definido en cada paso de la siguiente forma:

$$S_t = M^t$$

Donde, $M^t \in \mathcal{M}_{r,r}$ corresponde a una representación matricial del bosque a tratar en el instante t. En particular, cada entrada de esta matriz:

$$M^t_{i,j} = c^t_{i,j}, \text{ es el combustible en la celda (i, j) en el instante t.}$$

En particular, si $c^t_{i,j} = 1$, tal celda es un cortafuego.

3.1.2. Espacio de acciones

Se define la función $N(i, j, r) : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ como:

$$N(i,j,r) = \begin{cases} 0 & \text{si i = j = 0} \\ N(i,j-1,r)+1 & \text{si } j \neq 0 \wedge i \leq r-1 \wedge j \leq r-1 \\ N(i-1,r-1,r)+1 & \text{si } j = 0 \wedge i \leq r-1 \wedge j \leq r-1 \end{cases}$$

Se define el conjunto $N_r \subseteq \mathbb{N}$, $N_r = Im\ N(i, j, r)$, el cual induce una indexación sobre los componentes de la matriz M^t. Es decir:

$$M^t_{0,0} \to N(0,0,r) = 0, M^t_{0,1} \to N(0,1,r) = 1, ..., M^t_{r-1,r-1} \to N(r-1,r-1,r) = r^2 - 1$$

Luego, referirse a la celda 0, equivaldría a referirse a la entrada $(0, 0)$ de la matriz y de la misma forma con el resto de las celdas.

A modo de ejemplo, con $r = 4$, M^t:

$c^t_{0,0}$ $N(0,0) = 0$	$c^t_{0,1}$ $N(0,1) = 1$	$c^t_{0,2}$ $N(0,2) = 2$	$c^t_{0,2}$ $N(0,3) = 3$
$c^t_{1,0}$ $N(1,0) = 4$	$c^t_{1,1}$ $N(1,1) = 5$	$c^t_{1,2}$ $N(1,2) = 6$	$c^t_{1,3}$ $N(1,3) = 7$
$c^t_{2,0}$ $N(2,0) = 8$	$c^t_{2,1}$ $N(2,1) = 9$	$c^t_{2,1}$ $N(2,2) = 10$	$c^t_{2,3}$ $N(2,3) = 11$
$c^t_{3,0}$ $N(3,0) = 12$	$c^t_{3,1}$ $N(3,1) = 13$	$c^t_{3,2}$ $N(3,2) = 14$	$c^t_{3,3}$ $N(3,3) = 15$

A su vez se define, $X \subseteq N_r$, un conjunto de celdas prohibidas e $I_t \subseteq N_r$ como:

$$I_t = \{k \in N_r, \text{ con } N(i,j,r) = k | M^t_{i,j} \neq 1 \wedge k \notin X\}$$

Es decir, I_t corresponde a las celdas que no han sido seleccionadas como cortafuegos y

que no están prohibidas, por tanto están disponibles para seleccionar como cortafuego en ese instante. En particular se tiene, $N_r = I_0 \cup X$.

El espacio de acciones $\mathcal{A}$, se define a continuación:

$$A(S_t) = I_t$$

Ahora, una vez seleccionada la acción $a = N(i, j, r) \in I_t$, el estado transiciona según:

$$M_{i,j}^{t+1} = 1$$

A modo de ejemplo, con $r = 4$:

Tabla 3.1: Transición desde el estado M^t al M^{t+1}, mediante la selección de la acción $a_t = 1$, asociada a la entrada $(0, 1)$. Las celdas marcadas con rojo corresponden a aquellas en $X = \{0\}$ e $I_t = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15\}$. En el siguiente paso, $I_{t+1} = \{2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15\}$ La celda seleccionada es marcada con verde.

$c_{0,0}^t$	$c_{0,1}^t$	$c_{0,2}^t$	$c_{0,2}^t$
$c_{1,0}^t$	$c_{1,1}^t$	$c_{1,2}^t$	$c_{1,3}^t$
$c_{2,0}^t$	$c_{2,1}^t$	$c_{2,1}^t$	$c_{2,3}^t$
$c_{3,0}^t$	$c_{3,1}^t$	$c_{3,2}^t$	$c_{3,3}^t$

$\xrightarrow{a=1=n(0,1)}$

$c_{0,0}^{t+1}$	1	$c_{0,2}^{t+1}$	$c_{0,2}^{t+1}$
$c_{1,0}^{t+1}$	$c_{1,1}^{t+1}$	$c_{1,2}^t$	$c_{1,3}^{t+1}$
$c_{2,0}^{t+1}$	$c_{2,1}^{t+1}$	$c_{2,1}^{t+1}$	$c_{2,3}^{t+1}$
$c_{3,0}^{t+1}$	$c_{3,1}^{t+1}$	$c_{3,2}^{t+1}$	$c_{3,3}^{t+1}$

3.1.3. Funciones de recompensa

La función de recompensa para *Full Grid* es la siguiente:

$$f(s_t, a) = \begin{cases} 0 & \text{si t} < \text{T-1} \\ nb(M^T) \cdot k & \text{si t} = \text{T - 1} \end{cases}$$

Donde nb corresponde al promedio de celdas que se quemaron al ejecutar el simulador *Cell2Fire* sobre el bosque asociado a M^T y k es un factor de penalización.

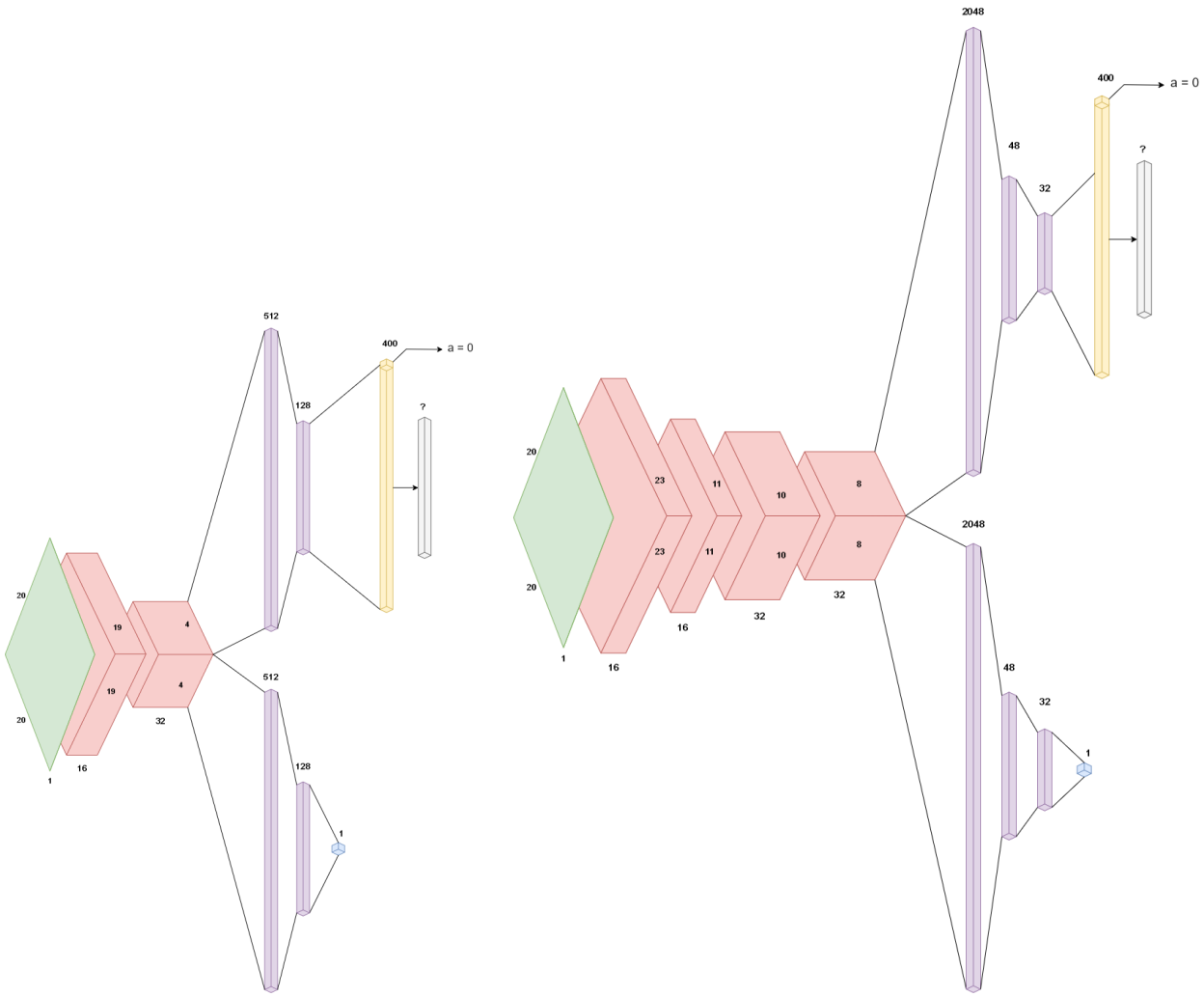
3.1.4. Representación de observaciones

Habiendo definido el espacio de estados $\mathcal{S}$, los inputs que se le entregarán a la *CNN* son los siguientes:

$$M^t = \begin{pmatrix} c_{0,0}^t & c_{0,1}^t & \cdots & c_{0,r-1}^t \\ c_{1,0}^t & c_{1,1}^t & \cdots & c_{1,r-1}^t \\ \cdots & \cdots & \cdots & \cdots \\ c_{r-1,0}^t & c_{r-1,1}^t & \cdots & c_{r-1,r-1}^t \end{pmatrix}$$

3.1.5. Arquitecturas de Redes

A continuación se presentan las arquitecturas de las *CNN's* utilizadas para Full Grid V1. La primera se denomina *small-net* y la segunda *big-net*.



(a) Arquictura small-net

(b) Arquictura big-net

Figura 3.2: Arquitecturas para el ambiente Full Grid V1.

En las figuras anteriores, el cuadrado verde representa el input definido en la sección 3.1.4, los prismas rojos corresponden a capas convolucionales junto a operaciones max-pool y los prismas morados son capas *fully connected*. Por otro lado, uno de los output correspondería a a una estimación de la función de valor $V(s, \theta)$, representada por el cubo azul. El segundo output correspondería a una estimación de la función estado-acción $Q(s, a, \theta)$ para cada acción, representada por el prisma amarillo, la cual a continuación pasa por una máscara, que filtra las acciones que no están disponibles, es decir $\{a \notin I_t\}$. Es necesario notar que $V(s, \theta)$ solo se calcula para Dueling Networks.

Con respecto a la librería utilizada para la contrucción de las redes, se ocupó PyTorch. Dentro de las opciones, estaban TensorFlow y Keras, sin embargo se descartaron debido a la mayor versatilidad y la posibilidad de acceder al grafo de computación que ofrece PyTorch.

3.2. Full Grid V2

Este enfoque es igual a *Full Grid V1*, con la diferencia que se agrega una matriz extra a cada estado S_t. Se referirá a la matriz M^t definida en el apartado anterior como M^t_{fg1}.

3.2.1. Espacio de estados

Nuevamente el espacio de estados S está definido de la siguiente forma:

$$S_t = M^t$$

Con la salvedad de que M^t ahora es un tensor de dimensiones $(2, r, r)$, donde $M^t_{0,.,.} = M^t_{fg1}$ y $M^t_{1,.,.}$ va a ser una matriz que representa la conectividad del bosque que se define como:

$M^t_{i,j} = m^t_{i,j}$, *es el número de celdas a la que la celda (i, j) está conectada en el instante t, normalizado*

En particular,

$$M^0_{1,i,j} = \begin{cases} \frac{3}{8} & \text{si } (i = 0 \vee i = r-1) \wedge (j = 0 \vee j = r-1) \\ \frac{5}{8} & \text{si } (i = 0 \vee i = r-1) \wedge (j \neq 0 \wedge j \neq r-1) \\ \frac{5}{8} & \text{si } (i \neq 0 \wedge i \neq r-1) \wedge (j = 0 \vee j = r-1) \\ 1 & \text{otherwise} \end{cases}$$

3.2.2. Espacio de acciones

El espacio de acciones se mantiene con respecto a *Full Grid V1*. La transición de $M^t_{0,.,.}$ es igual que para M^t_{fg1}.

Ahora, una vez seleccionada una acción $a = n(i, j, r) \in I_t$, $M^t_{1,.,.}$ transiciona según:

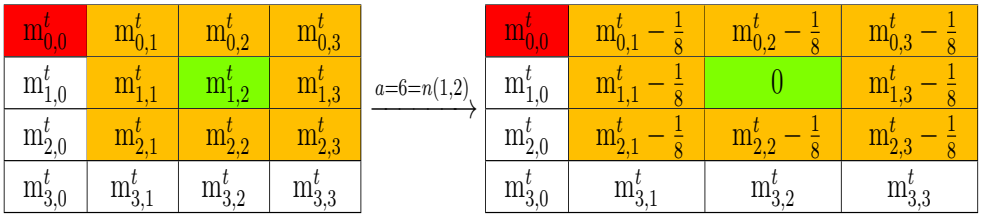
$$M^t_{1,i,j} = 0$$

Y a todas las entradas contiguas se les resta $\frac{1}{8}$, lo cual equivale a que dejan de estar conectadas a la celda (i, j):

$$M^{t+1}_{1,k,l} = \begin{cases} M^t_{1,k,l} - \frac{1}{8} & \text{si } (k = i+1 \vee k = i-1) \wedge (l = j+1 \vee l = j-1) \wedge (0 \leq k, l \leq r-1) \\ M^t_{1,k,l} & \text{otherwise} \end{cases}$$

A modo de ejemplo, con $r = 4$:

Tabla 3.2: Transición de la matriz $M^t_{1,.,.}$ desde el estado M^t al M^{t+1}, mediante la selección de la acción $a_t = 6$, asociada a la entrada $(1, 2)$. Las celdas marcadas con rojo corresponden a aquellas en $X = \{0\}$ e $I_t = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15\}$. En el siguiente paso, $I_{t+1} = \{1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 14, 15\}$. La celda seleccionada es marcada con verde y las contiguas con amarillo.



Intuitivamente, la matriz $M^t_{1,.,.}$ corresponde a la cantidad de celdas con la que cada celda individual está conectada, dividido por ocho (para que cada entrada esté en $[0, 1]$). Si una celda es marcada como un cortafuego, deja de estar conectada a cualquier otra celda y por su parte las celdas contiguas, pierden una conexión.

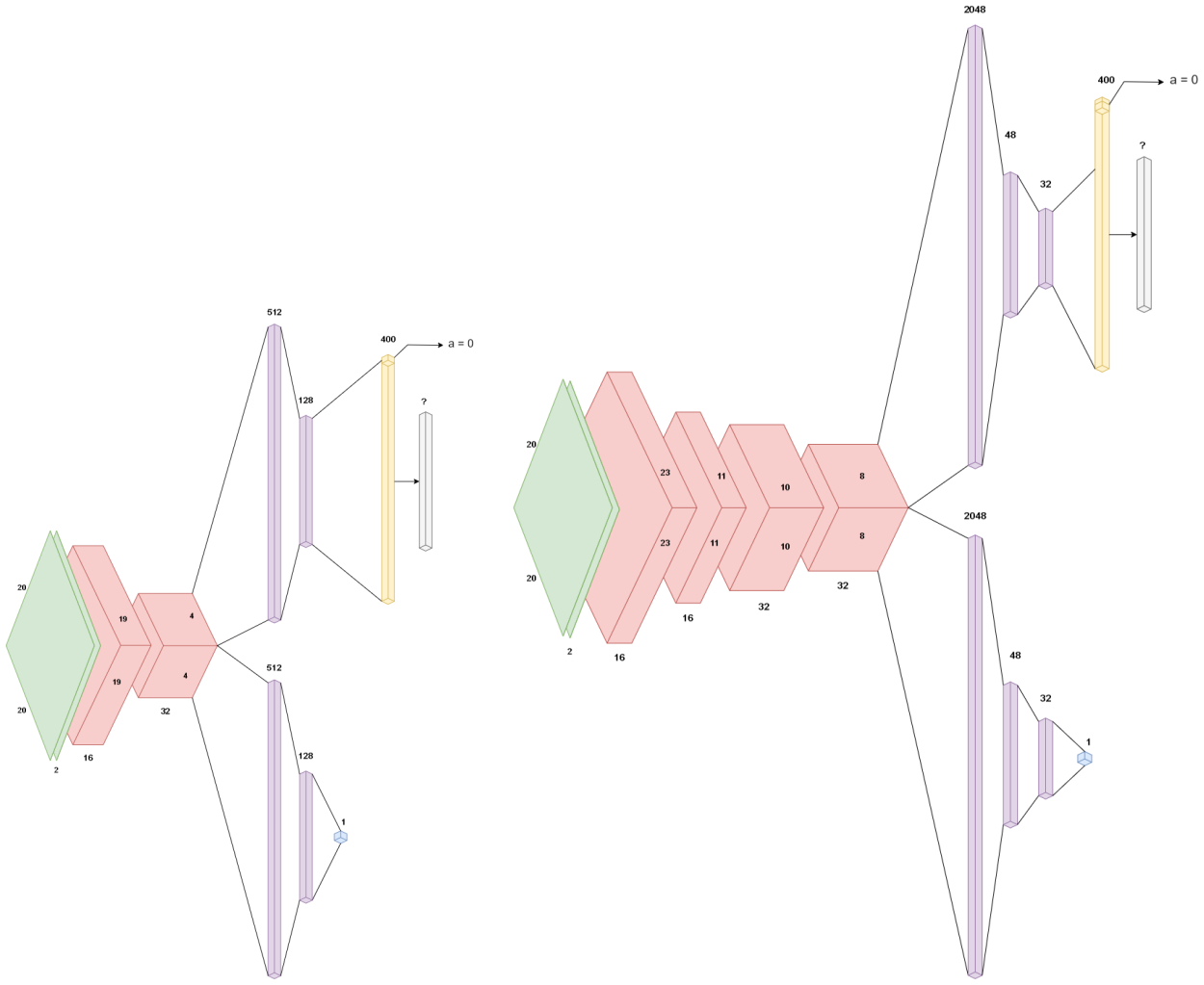
3.2.3. Representación de observaciones

En este caso, los inputs que recibirá la *CNN* serán los siguientes:

$$\mathrm{M}^t_{0,.,.} = \begin{pmatrix} c^t_{0,0} & c^t_{0,1} & \dots & c^t_{0,r-1} \\ c^t_{1,0} & c^t_{1,1} & \dots & c^t_{1,r-1} \\ \dots & \dots & \dots & \dots \\ c^t_{r-1,0} & c^t_{r-1,1} & \dots & c^t_{r-1,r-1} \end{pmatrix} \quad \mathrm{M}^t_{1,.,.} = \begin{pmatrix} m^t_{0,0} & m^t_{0,1} & \dots & m^t_{0,r-1} \\ m^t_{1,0} & m^t_{1,1} & \dots & m^t_{1,r-1} \\ \dots & \dots & \dots & \dots \\ m^t_{r-1,0} & m^t_{r-1,1} & \dots & m^t_{r-1,r-1} \end{pmatrix}$$

3.2.4. Arquitecturas de Redes

A continuación se presentan las arquitecturas de las *CNN's* utilizadas para Full Grid V2. La primera se denomina *small-net* y la segunda *big-net*.



(a) Arquictura small-net

(b) Arquictura big-net

Figura 3.3: Arquitecturas para el ambiente Full Grid V2.

En las figuras anteriores, el cuadrado verde representa el input definido en 3.2.3, los prismas rojos corresponden a capas convolucionales junto a operaciones max-pool y los prismas morados son capas *fully connected*. Por otro lado, uno de los output correspondería a una estimación de la función de valor $V(s, \theta)$, representada por el cubo azul. El segundo output correspondería a una estimación de la función estado-acción $Q(s, a, \theta)$ para cada acción, representada por el prisma amarillo, la cual a continuación pasa por una máscara, que filtra las acciones que no están disponibles, es decir $\{a \notin I_t\}$. Al igual que en 3.1.5, $V(s, \theta)$ sólo se calcula para Dueling Networks.

3.3. Ajuste de parámetros

La selección de parámetros se llevó a cabo mediante la realización de una búsqueda informal sobre el siguiente espacio:

Tabla 3.3: Espacio de parámetros desde donde se escogieron aquellos a utilizar en el entrenamiento. C corresponde al número de iteraciones entre actualizaciones de la target network y *epochs* corresponde a las épocas del pre-entrenamiento. Se probaron las 162 combinaciones de pares $(\alpha, \epsilon, \gamma, C, epochs)$.

$\boldsymbol{\alpha}$	$\boldsymbol{\epsilon}$	$\boldsymbol{\gamma}$	**C**	**epochs**
$5e^{-3}$	1	1	100	20000
$5e^{-4}$	0.75	0.99	200	50000
$5e^{-5}$	0.5	0.9		100000

Se seleccionaron los parámetros a utilizar comparando el rendimiento de cada combinación en el algoritmo *Deep Q-Learning con Experience Replay*, con 1000 demostraciones sobre la instancia de 10×10 y utilizando *Full Grid V1*. Lo anterior se llevó a cabo para ambas arquitecturas de red. De esta manera, los parámetros relevantes seleccionados para el entrenamiento fueron:

Tabla 3.4: D corresponde al tamaño del buffer de memoria, BS al tamaño de los batches, C la frecuencia de actualización de la target network, epochs las épocas del pre-entrenamiento, $\bar{\epsilon}$ el decaimiento y $\hat{\epsilon}$ el valor mínimo de ϵ. A la izquierda se encuentran los parámetros asociados a la arquitectura *small net* y a la derecha aquellos para *big net*.

D	100000
BS	64
C	200
epochs	20000
$\boldsymbol{\alpha}$	$5e^{-5}$
$\boldsymbol{\gamma}$	1
$\boldsymbol{\epsilon}$	1
$\bar{\boldsymbol{\epsilon}}$	0.005
$\hat{\boldsymbol{\epsilon}}$	0.001

D	100000
BS	64
C	200
epochs	20000
$\boldsymbol{\alpha}$	$5e^{-4}$
$\boldsymbol{\gamma}$	1
$\boldsymbol{\epsilon}$	1
$\bar{\boldsymbol{\epsilon}}$	0.005
$\hat{\boldsymbol{\epsilon}}$	0.001

3.4. Evaluación

La evaluación de los algoritmos se realizará en relación a dos bases. En primer lugar, se comparará el rendimiento de las soluciones en comparación a una solución aleatoria. En segundo lugar, se comparará con respecto a un algoritmo que utiliza una métrica llamada Downstream Protection Value (Cristobal Pais, 2021), que se denominó *Baseline*.

3.4.1. Downstream Protection Value

El Donwstream Protection Value o DPV, es una métrica de conectividad de un paisaje. Éste representa la importancia de una celda individual en el esparcimiento del fuego, midiendo el impacto que tiene sobre el resto de las celdas, el que una celda en específico se queme.

Considerando el paisaje como un grafo no dirigido $G = (V, E)$ en el cual cada celda corresponde a un nodo $v \in V$ y las aristas $e \in E$ corresponden a la relación de contiguedad entre las celdas, el DPV se calcula considerando un conjunto de simulaciones de esparcimiento de incendios D sobre el grafo G. Cada simulación $d \in D$, genera un subgrafo dirigido $G_d = (V_d, E_d)$ sobre G, donde $V_d \in V$ contiene los nodos que se quemaron y $E_d \in E$ contiene todas las aristas por donde el fuego se transmitió en la simulación d. Dado el grafo G_d, se puede determinar el arbol de recubrimiento mínimo con raiz en una celda j: $T_d(j) = (V(j), E(J))$, a partir del cual se define el DPV para un grafo G_d fijo y para cada nodo en V:

$$DPV_d(i) = \sum_{j \in V(i)} V(j)$$

Donde $V(j)$ es un valor en riesgo para un nodo j cualquiera. Considerando múltiples simulaciones:

$$DPV(i) = \sum_{d \in D} DPV_d(i)$$

Habiendo definido el DPV, se construye el siguiente algoritmo denominado *Baseline*:

Algorithm 10 Baseline

for episode ∈ episodes **do**
  $t' \leftarrow 0$
  for $t \leq T$ **do**
    Simular incendio sobre s_t, generando G_d
    Calcular DPV_d para cada $i \in V$.
    $a_t \leftarrow argmax_{i \in V} DPV_d(i)$
    Ejecutar a_t en el ambiente y observar (r_t, s_{t+1})
    Guardar transición (s_t, a_t, r_t, s_{t+1})
  end for
end for

3.4.2. Landscapes

Se comparará el rendimiento de los algoritmos a lo largo de instancias de distintos tamaños y características. En particular, se utilizó un bosque base de 20×20 celdas a partir del cual se generaron instancias de otros tamaños. Para esto, se utilizó un algoritmo de interpolación conocido como *Nearest-neighbor Interpolation*. Mediante este método se puede rescalar una imagen, siendo el valor de un pixel de la imagen resultante aquel que el algoritmo determina que es el pixel más cercano en la imagen original. De esta forma, para cada bosque de 20×20, se generaron interpolaciones de tamaño 10×10 y 6×6.

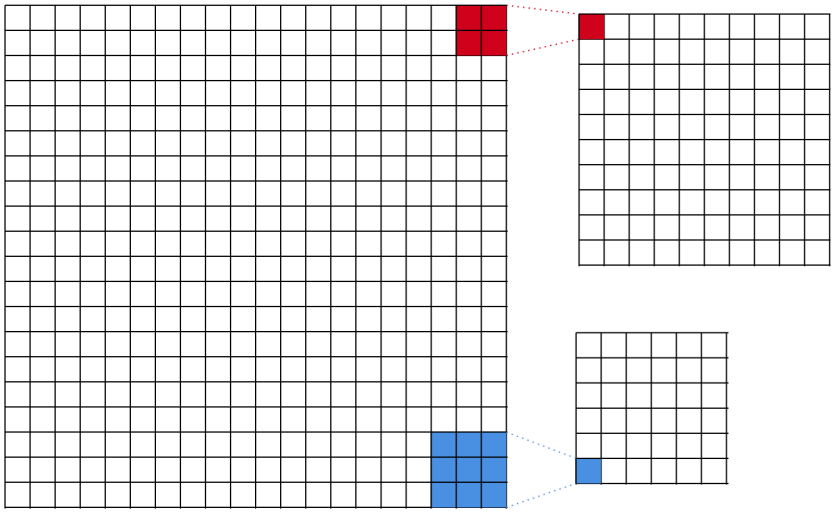


Figura 3.4: Método de interpolación.

En primer lugar se construyó un bosque ficticio compuesto de una vegetación uniforme a lo largo de todo el paisaje. La vegetación utilizada corresponde a una *Boreal Spruce*, definida en el apartado 2.1.

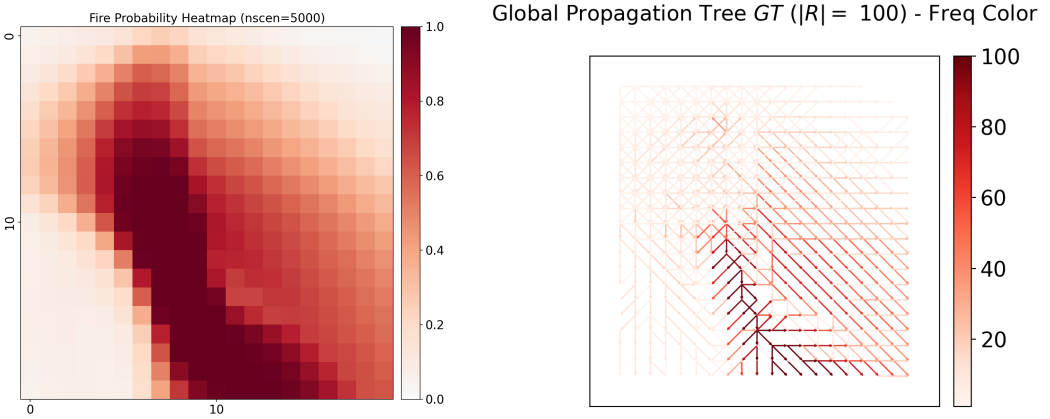
Figura 3.5: Bosque homogéneo.

3.4.2.1. Solución Conocida

Con el objetivo de tener control sobre la optimalidad de la solución, se contruyeron instancias cuya solución óptima fuese trivial de determinar. Dependendiendo del tamaño del bosque, se generaron instancias que diferían en términos de dónde se genera el incendio. De todas maneras, se construyeron las instancias de forma que se quemase alrededor de un 50 % del bosque, de no tratarse en lo absoluto.

3.4.2.1.1. 20×20

En esta instancia, se limita el inicio del incendio al cuadrante superior izquierdo del bosque, específicamente a un cuadrado de 9x9 celdas exactamente en la esquina. Se prohíbe a su vez el posicionamiento de cortafuegos en la zona de ignición. Lo anterior se complementó con una dirección del viento de 270º, de manera que el fuego se esparza en diagonal hacia la esquina inferior derecha del bosque, tal como se observa en la Figura 3.6. De esta manera, la solución óptima es la que se observa en la Figura 3.7, la cual consiste en encerrar el cuadrado en donde se inician los incendios, evitando el esparcimiento en su totalidad fuera de él.



(a) Se muestra el Burn Probability Map, donde cada celda corresponde a la frecuencia con la que esta se quemó, del bosque homogéneo descrito.

(b) Arbol de esparcimiento de un incendio de esta instancia.

Figura 3.6: Esparcimiento de un incendio para un bosque homogéneo con solución conocida.

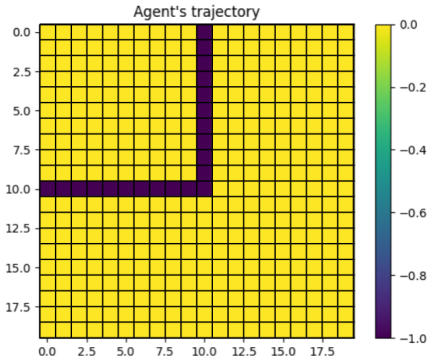
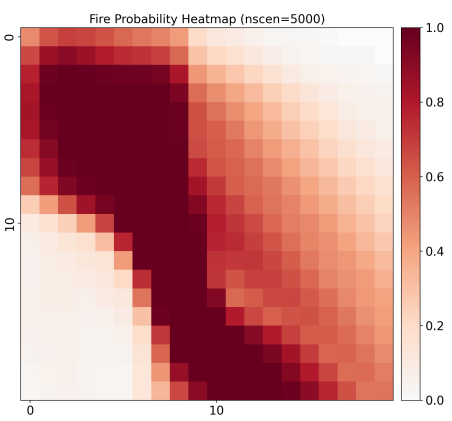


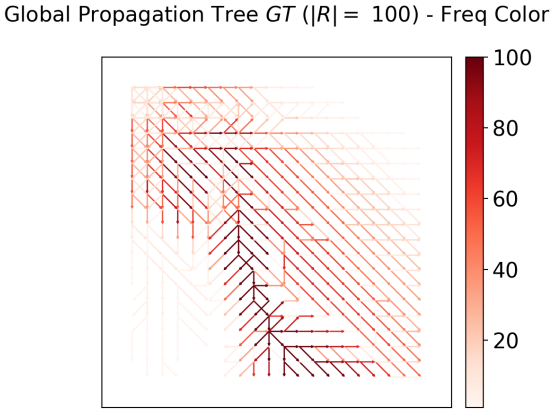
Figura 3.7: Solución óptima conocida.

3.4.2.1.2. 10×10

Por otro lado, en esta instancia se limita el inicio a un cuadrante más pequeño, nuevamente en el sector izquierdo superior del bosque, específicamente a un cuadrado de 2×2 celdas exactamente en la esquina. El resto de las condiciones se mantuvieron con respecto a la instancia de 20×20. En este caso, la solución óptima es la que se observa en la Figura 3.9, la cual consiste denuevo en encerrar el cuadrado en donde se inician los incendios, evitando el esparcimiento en su totalidad fuera de él.



(a) Se muestra el Burn Probability Map, donde cada celda corresponde a la frecuencia con la que esta se quemó, del bosque homogéneo descrito.



(b) Arbol de esparcimiento de un incendio de esta instancia.

Figura 3.8: Esparcimiento de un incendio para un bosque homogéneo con solución conocida.

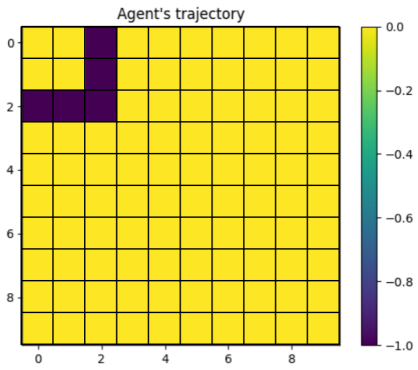
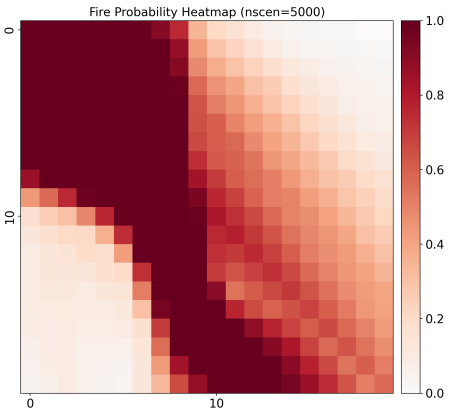


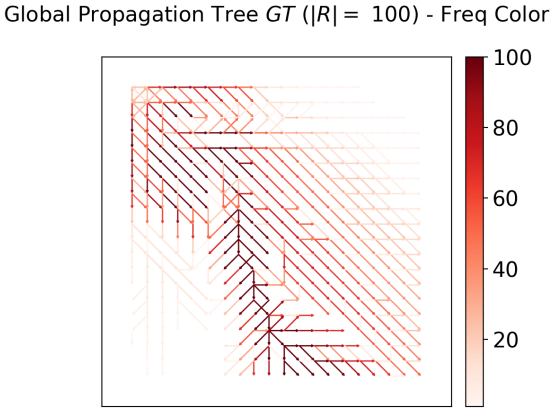
Figura 3.9: Solución óptima conocida.

3.4.2.1.3. 6×6

Finalmente, para esta instancia se limita el inicio del incendio a una única celda, la ubicada en la esquina superior izquierda. En este caso no se prohíbe el posicionamiento de cortafuegos en la zona de ignición, de manera que la solución sea igual de aparente que en el resto. El resto de las condiciones se mantienen con respecto a las instancias anteriores.



(a) Se muestra el Burn Probability Map, donde cada celda corresponde a la frecuencia con la que esta se quemó, del bosque homogéneo descrito.



(b) Arbol de esparcimiento de un incendio de esta instancia.

Figura 3.10: Esparcimiento de un incendio para un bosque homogéneo con solución conocida.

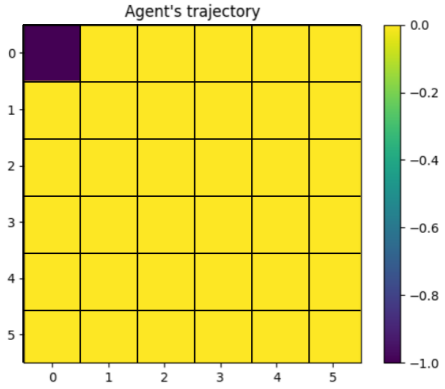
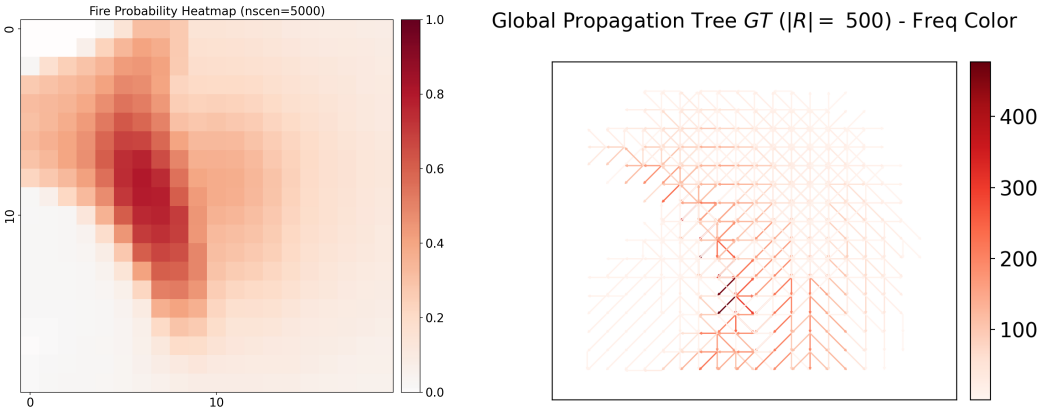


Figura 3.11: Solución óptima conocida.

3.4.2.2. Solución Desconocida

En contraste con 3.4.2.1, se construyó una instancia cuya solución óptima no fuese posible determinar sin utilizar un solver de resolución. En esta, los puntos de ignición no están limitados a ninguna zona en particular. De la misma manera, la dirección del viento es variable, generando incendios que se esparcen de maneras distintas, tal como se observa en la Figura 3.12. En este caso, los incendios se limitaron a un esparcimiento cercano al 25 %.



(a) Burn Probability Map del bosque homogéneo descrito.

(b) Arbol de esparcimiento de un incendio en esta instancia.

Figura 3.12: Esparcimiento de un incendio para un bosque homogéneo con solución desconocida.

Capítulo 4

Resultados

Todas los experimentos fueron realizadas en un procesador AMD Ryzen 9 5950 CPU @ 3.4GHz con 32 GB de RAM y tarjeta gráfica Nvidia GeForce RTX 3080 Ti. Los códigos fueron escritos en el lenguaje de programación Python y el sistema operativo utilizado es Ubuntu 20.04 no nativo.

Glosario:

mab_ucb	Multi-Armed Bandit UCB
mab_greedy	Multi-Armed Bandit epsilon-greedy
q_learning	Q-Learning
dqn	Deep Q-Learning
2dqn	Double Deep Q-Learning
ddqn	Dueling Double Deep Q-Learning
_p	Prioritized Experience Replay
random	Random Algorithm
Baseline	Baseline Algorithm
Optimal	Optimal Solution

A lo largo de esta sección, se presentan gráficos que utilizan promedios móviles, esto porque en general las curvas de recompensa son ruidosas y no permiten ver tendencias de manera lo suficientemente clara. Un promedio móvil se define como:

$$\bar{x}_w(t) = \frac{1}{w} \sum_{i=t-w+1}^{t} x_i$$

Donde $\bar{x}_w(t)$ es un promedio móvil de x, calculado sobrer una ventana de tamaño w y x es una secuencia de valores.

Un aspecto que se debe tener en cuenta en las curvas de recompensas, es que las curvas de la heurística se generaron a partir de una menor cantidad de ejemplos. Esto debido a que tomaba un tiempo considerable para ejemplos de mayor tamaño. Luego, para poder graficar todas en una misma curva, la cantidad de episodios se normalizó, razon por la cual el eje x está en $[0, 1]$.

4.1. Solución Conocida

4.1.1. 6×6

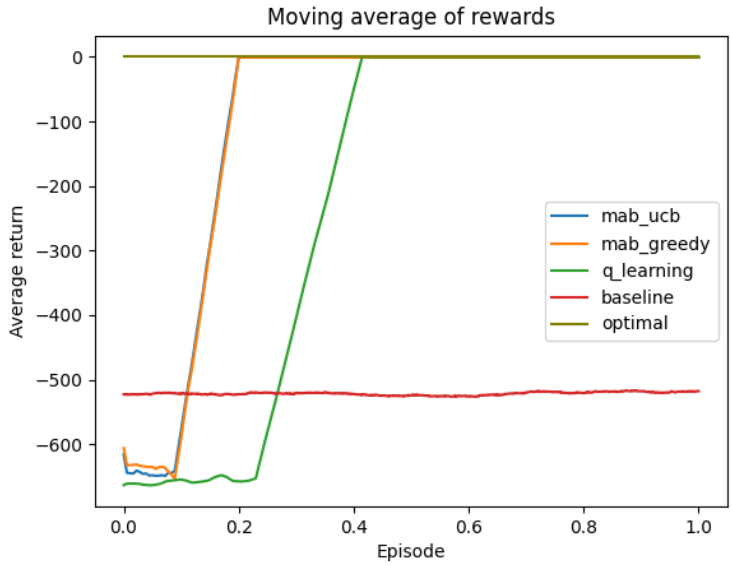
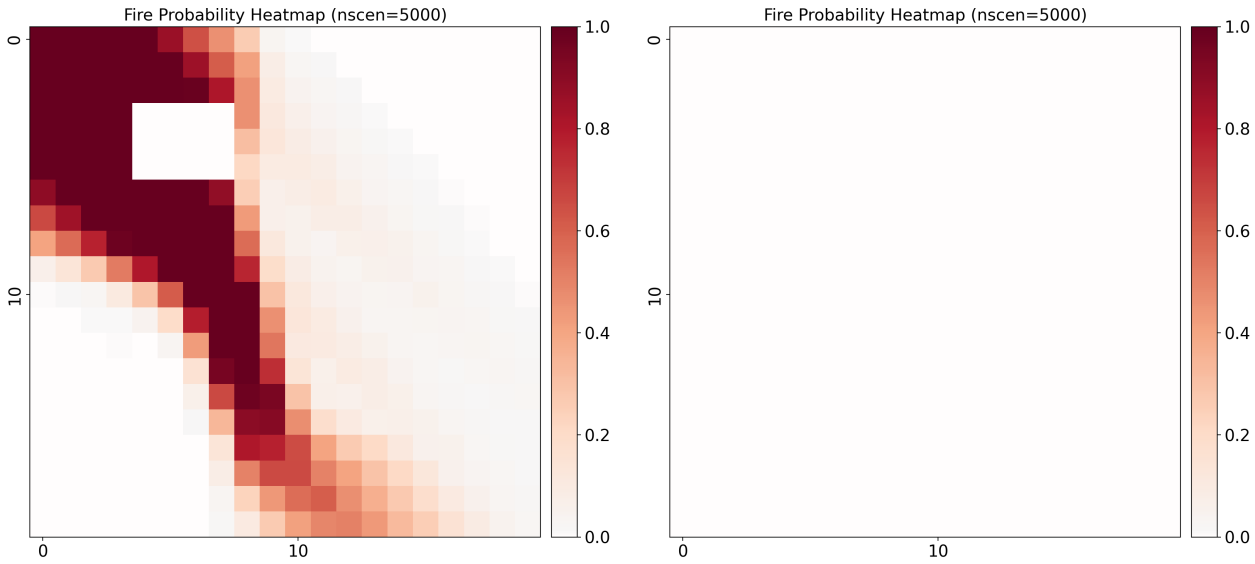


Figura 4.1: Promedios móviles de las recompensas obtenidas por episodio, la ventana sobre la cual se calcula es de 20 episodios.

Se puede apreciar que los tres algoritmos algoritmos logran llegar a la solución óptima. A su vez, pareciese haber un periodo más largo de exploración en el algoritmo Q-Learning(q_learning), previo a converger al óptimo. Todos los algoritmos obtienen recompensas superiores al demostrador.



(a) Mapa de probabilidad de quema para la solución obtenida utilizando Downstream Protection Value.

(b) Mapa de probabilidad de quema para la mejor solución obtenida.

Figura 4.2: Rendimiento de las soluciones obtenidas.

En la figura anterior es posible notar que la solución obtenida por los algoritmos de Reinforcement Learning logran contener por completo un potencial incendio, lo que se traduce en un mapa de probabilidad de quema con valores nulos en todas las celdas. En constraste, la mejor solución del demostrador no lo logra.

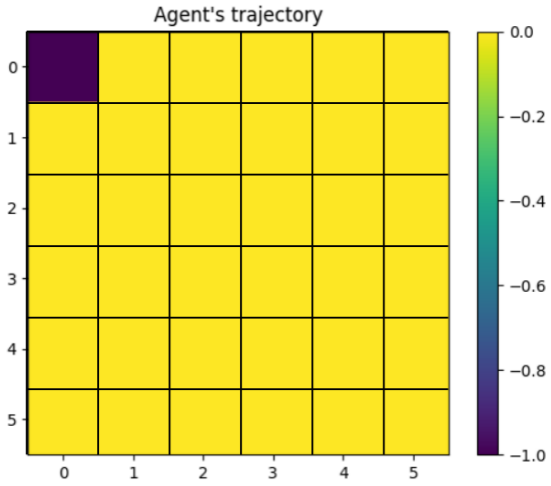
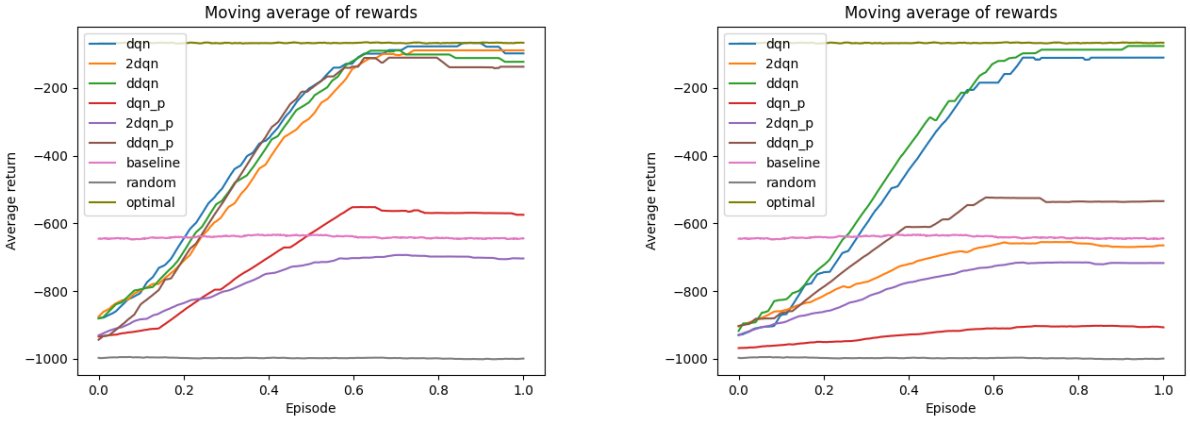


Figura 4.3: Mejor solución obtenida.

La mejor solución obtenida corresponde a la solución óptima que se presentó en 3.4.2.1.3

4.1.2. 10×10

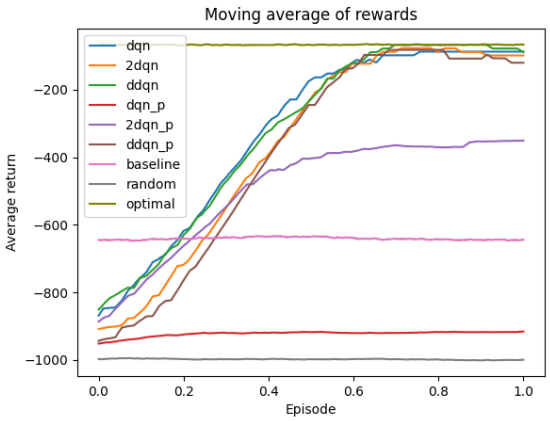


(a) Promedios móviles de las recompensas obtenidas por episodio, la ventana sobre la cual se calcula es de 500 episodios.

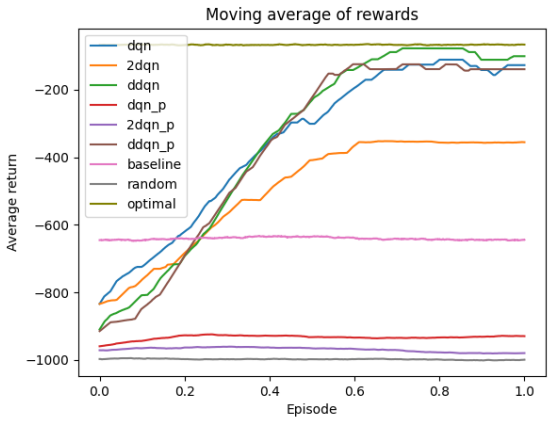
(b) Promedios móviles de las recompensas obtenidas por episodio, la ventana sobre la cual se calcula es de 500 episodios.

Figura 4.4: Soluciones obtenidas para la instancia de 10×10, en ambiente v1.

Ocupando la versión uno del ambiente se puede apreciar que gran parte de los algoritmos convergen al rendimiento óptimo. Algunos convergen a otros puntos, pero en todos se genera convergencia y rendimiento superior al algoritmo aleatorio.



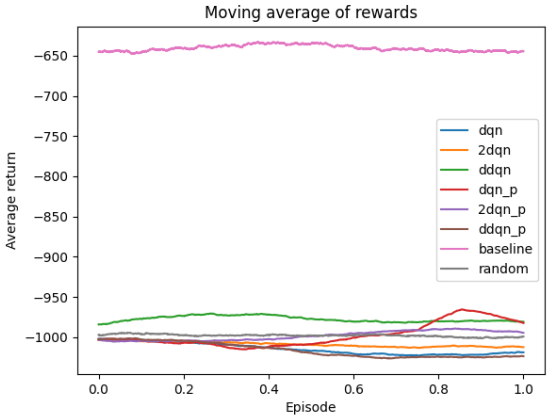
(a) Promedios móviles de las recompensas obtenidas por episodio, la ventana sobre la cual se calcula es de 500 episodios.



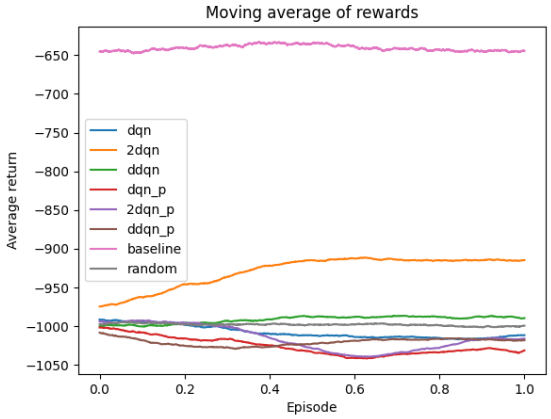
(b) Promedios móviles de las recompensas obtenidas por episodio, la ventana sobre la cual se calcula es de 500 episodios.

Figura 4.5: Soluciones obtenidas para la instancia de 10×10, en ambiente v2.

Utilizando la segunda versión del ambiente, se generan curvas que en general logran el rendimiento óptimo. El rendimiento es similar a aquel de la versión uno, todos los algoritmos logran superar al algoritmo aleatorio.



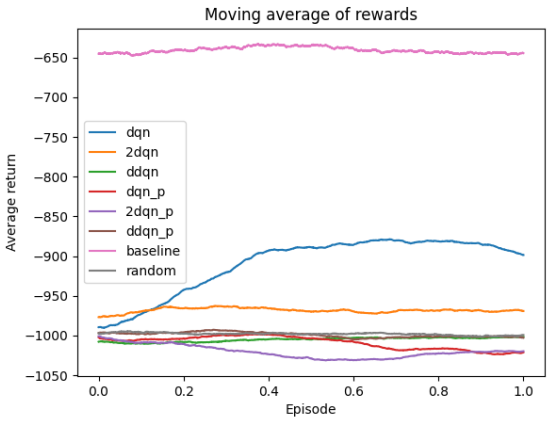
(a) Promedios móviles de las recompensas obtenidas por episodio para small net, la ventana sobre la cual se calcula es de 500 episodios.



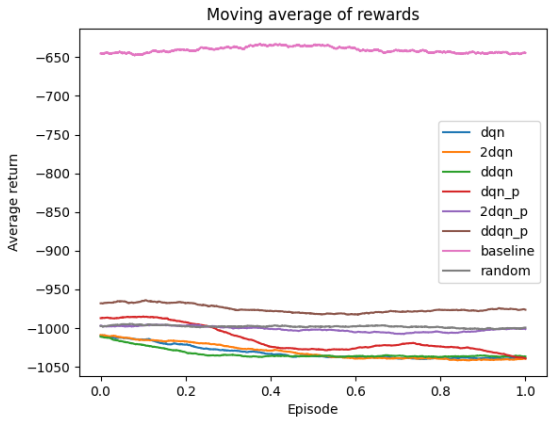
(b) Promedios móviles de las recompensas obtenidas por episodio para big net, la ventana sobre la cual se calcula es de 500 episodios.

Figura 4.6: Soluciones obtenidas para la instancia de 10×10, sin demostraciones y ambiente v1.

Para la versión uno del ambiente, el rendimiento que se obtiene cuando no se ocupan demostraciones es similar al de un algoritmo aleatorio, estando lejos de aquel del demostrador.



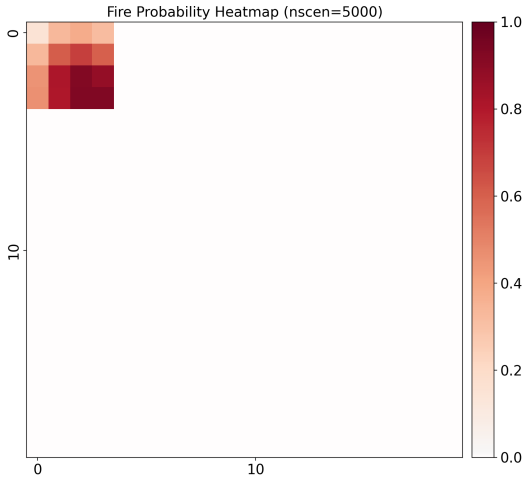
(a) Promedios móviles de las recompensas obtenidas por episodio para small net, la ventana sobre la cual se calcula es de 500 episodios.



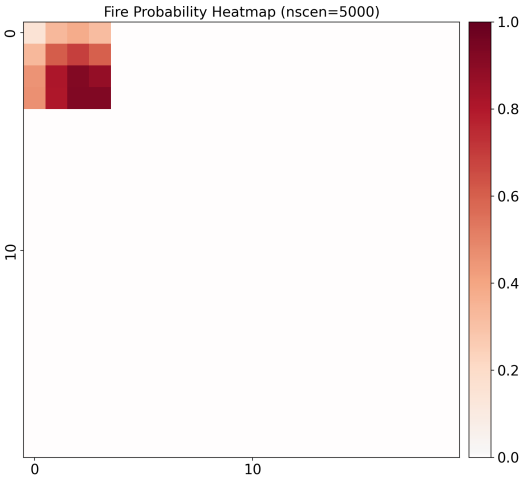
(b) Promedios móviles de las recompensas obtenidas por episodio para big net, la ventana sobre la cual se calcula es de 500 episodios.

Figura 4.7: Soluciones obtenidas para la instancia de 10×10, sin demostraciones y ambiente v2.

De manera similar que para la figura anterior, sin demostraciones se obtiene rendimiento comparable con el algoritmo aleatorio, superándolo en pocas ocasiones.



(a) Mapa de probabilidad de quema para la mejor solución obtenida utilizando Downstream Protection Value.



(b) Mapa de probabilidad de quema para la mejor solución obtenida.

Figura 4.8: Rendimiento de las soluciones obtenidas.

Con respecto a la contención de un potencial incendio, tanto el demostrador como la mejor solución con Reinforcement Learning logran hacerlo completamente. Como se observa en la figura, las probabilidades de quema sólo son no negativas en el cuadrante adonde se delimitaron las igniciones.

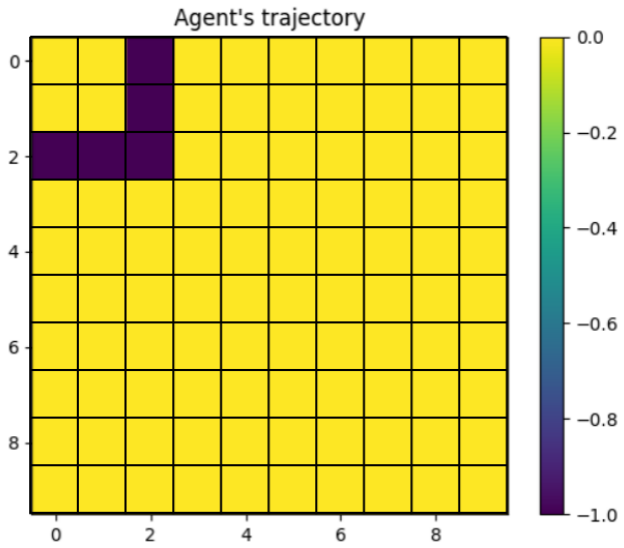
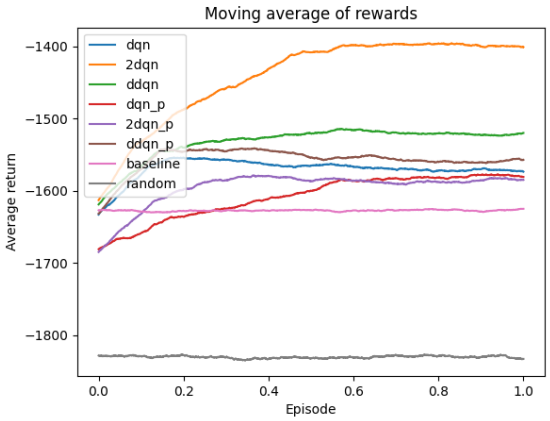


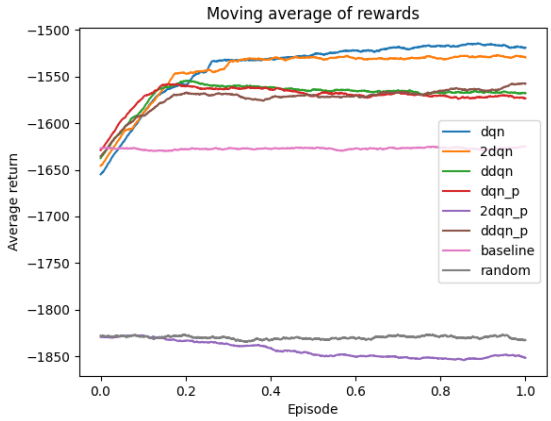
Figura 4.9: Mejor solución obtenida.

Finalmente, la mejor solución que se obtiene coincide con la solución óptima presentada en 3.4.2.1.2.

4.1.3. 20×20



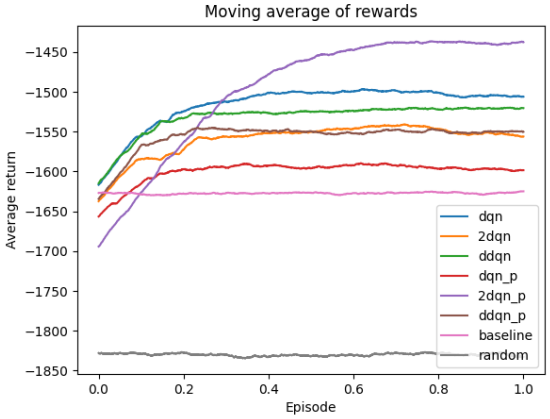
(a) Promedios móviles de las recompensas obtenidas por episodio, la ventana sobre la cual se calcula es de 500 episodios.



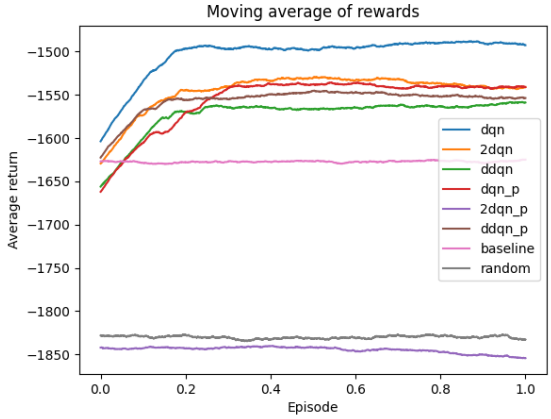
(b) Promedios móviles de las recompensas obtenidas por episodio, la ventana sobre la cual se calcula es de 500 episodios.

Figura 4.10: Soluciones obtenidas para la instancia de 20×20, en ambiente v1.

Para la versión uno del ambiente y ambas arquitecturas de red se obtiene rendimiento superior al demostrador. Al mismo tiempo, la gran mayoría de los algoritmos convergen. Para la arquitectura small-net, el algoritmo que obtiene mejor rendimiento es Double Deep Q-Learning(2dqn) y para big-net Deep Q-Learning(dqn). Todos los algoritmos, excepto Double Deep Q-Learning con Prioritized Experience Replay para big-net, convergen a rendimiento superior al del algoritmo aleatorio. Para small-net, todos los algoritmo alcanzan rendimiento superior al del demostrador.



(a) Promedios móviles de las recompensas obtenidas por episodio, la ventana sobre la cual se calcula es de 500 episodios.



(b) Promedios móviles de las recompensas obtenidas por episodio, la ventana sobre la cual se calcula es de 500 episodios.

Figura 4.11: Soluciones obtenidas para la instancia de 20×20, en ambiente v2.

Para la versión dos y ambas redes, se logra rendimiento superior al demostrador y con-

vergencia para la gran mayoría de los algoritmos. Con small-net, el algoritmo con mejor rendimiento es Double Deep Q-Learning con Prioritized Experience Replay (2dqn_p) y con big-net Deep Q-Learning(dqn). El único algoritmo que no logra superar al demostrador es el Double Deep Q-Learning con Prioritized Experience Replay para big-net, el cual obtiene rendimiento similar al algoritmo aleatorio.

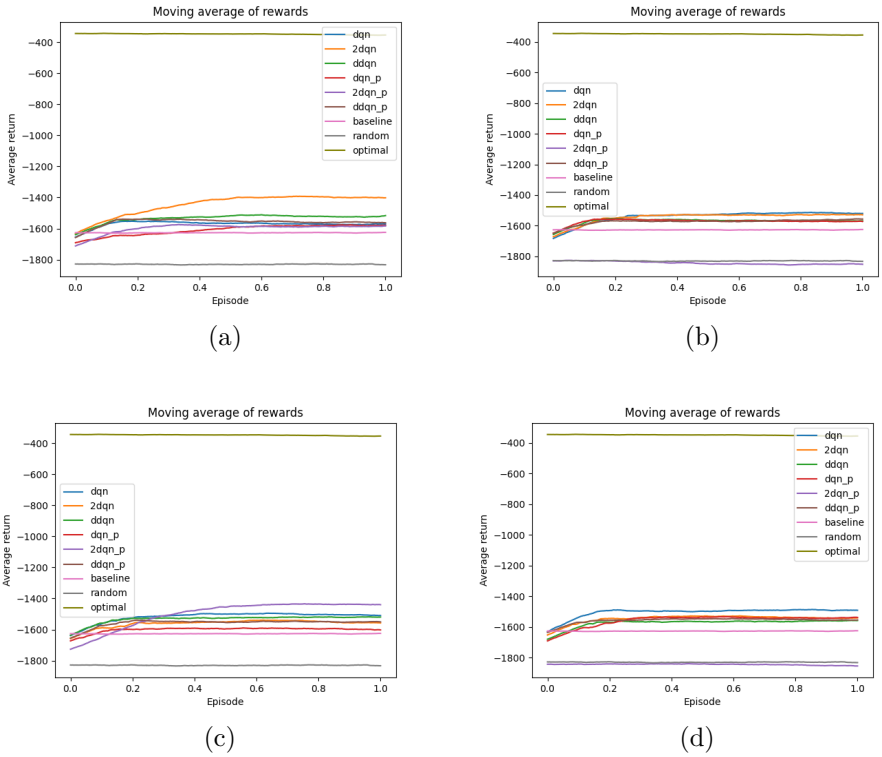
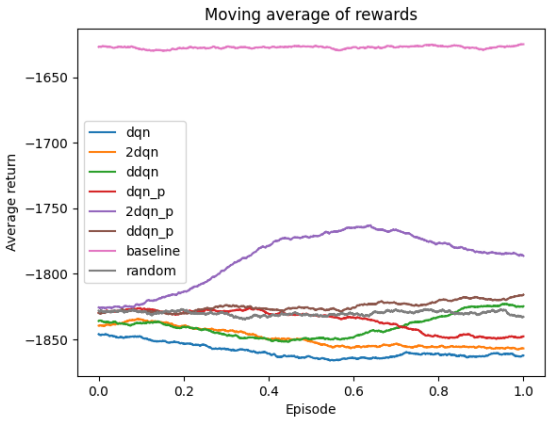
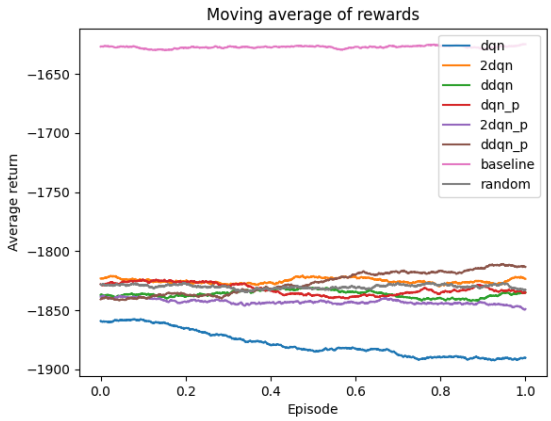


Figura 4.12: Soluciones obtenidas para la instancia de 20×20, comparadas con el óptimo.

Comparando las soluciones obtenidas con la óptima, se puede ver que ningún algoritmo llega a rendimiento cercano al óptimo. Esto se cumple para ambas versiones del ambiente y ambas versiones de la red. En la solución óptima se queman 40 celdas, mientras que para los algoritmos se queman al menos 140.



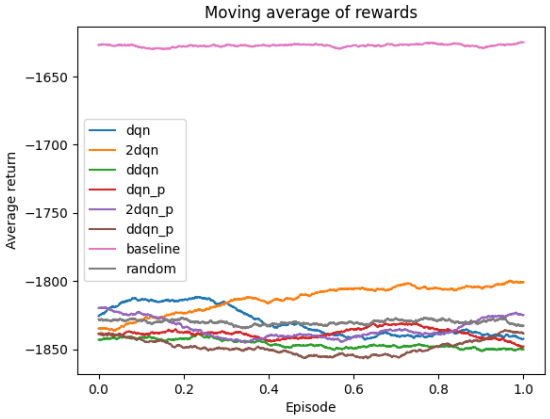
(a) Promedios móviles de las recompensas obtenidas por episodio para small net, la ventana sobre la cual se calcula es de 500 episodios.



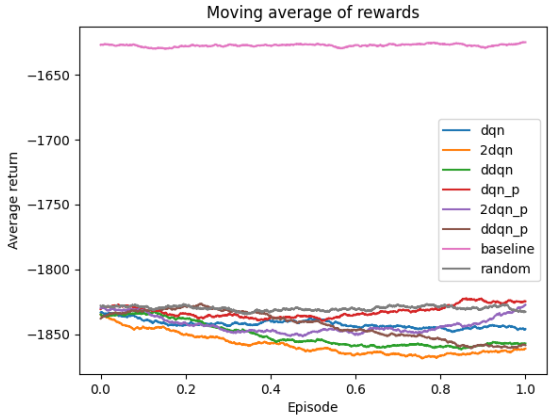
(b) Promedios móviles de las recompensas obtenidas por episodio para big net, la ventana sobre la cual se calcula es de 500 episodios.

Figura 4.13: Soluciones obtenidas para la instancia de 20×20, sin demostraciones y ambiente v1.

Para la versión uno del ambiente, se obtiene rendimiento inferior al del demostrador y cercano al de un algoritmo aleatorio. La mayoría de los algoritmos no pareciesen converger.



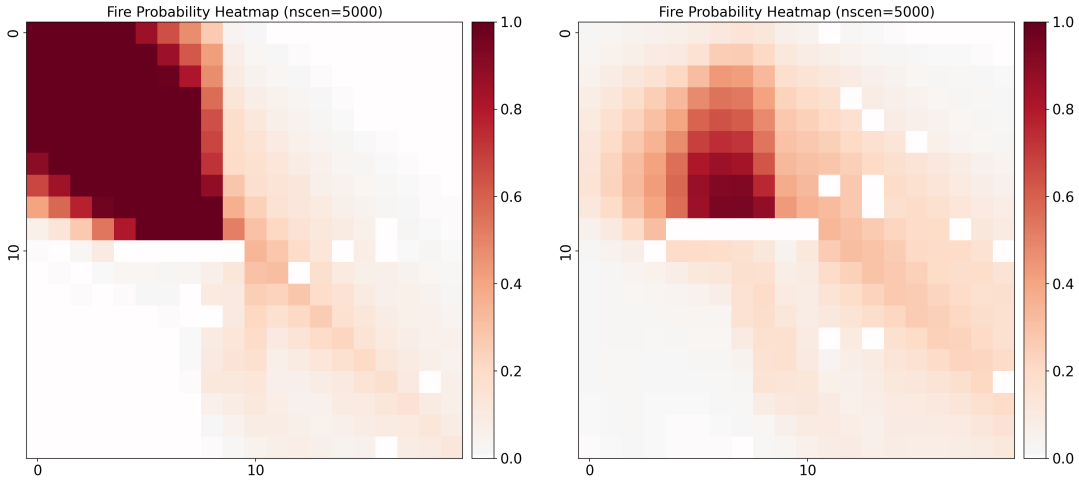
(a) Promedios móviles de las recompensas obtenidas por episodio para small net, la ventana sobre la cual se calcula es de 500 episodios.



(b) Promedios móviles de las recompensas obtenidas por episodio para big net, la ventana sobre la cual se calcula es de 500 episodios.

Figura 4.14: Soluciones obtenidas para la instancia de 20×20, sin demostraciones y ambiente v2.

Para la versión dos del ambiente, nuevamente se obtiene rendimiento inferior al del demostrador y cercano al de un algoritmo aleatorio. A su vez, la convergencia no ocurre en todos los algoritmos.



(a) Mapa de probabilidad de quema para la mejor solución obtenida utilizando Downstream Protection Value.

(b) Mapa de probabilidad de quema para la mejor solución obtenida.

Figura 4.15: Rendimiento de las soluciones obtenidas.

Con respecto a la contención de incendios, los algoritmos de Reinforcement Learning no la logran completamente, así como tampoco el demostrador. Aún así, los primeros pareciesen reducir considerablemente los porcentajes de quema en el paisaje.

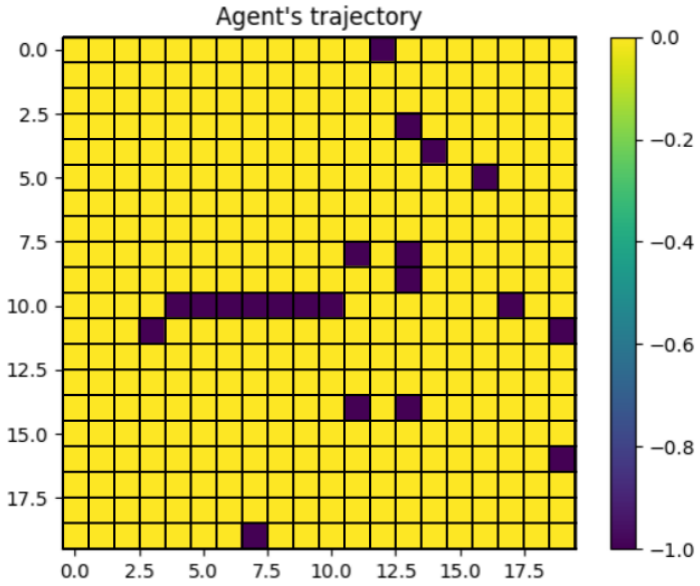


Figura 4.16: Mejor solución obtenida.

Con respecto a la optimalidad, los algoritmos de Reinforcement Learning no logran obtener la solución óptima presentada en 3.4.2.1.1, aunque pareciesen haber ciertos elementos estructurales que si comparten. En la solución obtenida, los cortafuegos pareciesen rodear el cuadrante adonde se delimitan las igniciones, sin embargo varios se posicionan en celdas lejos de esta zona.

4.2. Solución Desconocida

4.2.1. 6×6

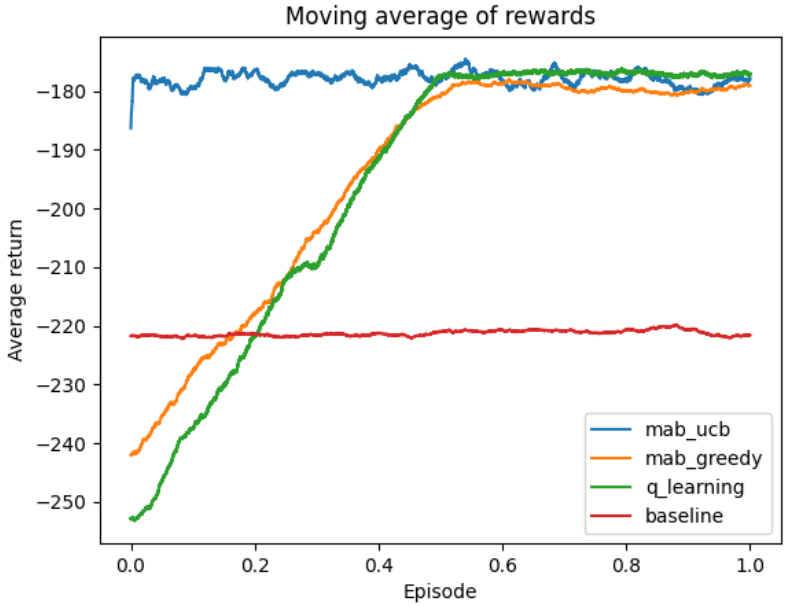
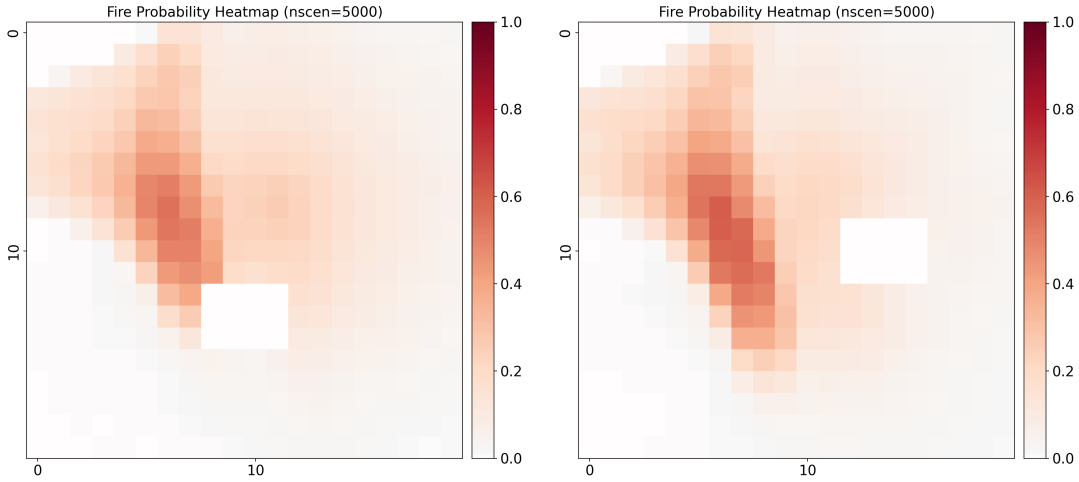


Figura 4.17: Promedios móviles de las recompensas obtenidas por episodio, la ventana sobre la cual se calcula es de 20 episodios.

Los tres algoritmos implementados logran superar el rendimiento del demostrador y convergen al mismo punto. Se puede apreciar que al algoritmo Multi-Armed Bandits UCB(mab_ucb) converge considerablemente más rápido que el resto a la solución final.



(a) Mapa de probabilidad de quema para la mejor solución obtenida utilizando Downstream Protection Value.

(b) Mapa de probabilidad de quema para la mejor solución obtenida.

Figura 4.18: Rendimiento de las soluciones obtenidas.

Se puede apreciar que la mejor solución del demostrador y la de los algoritmos de Reinforcement Learning pareciesen reducir los porcentajes de quema de manera similar, aunque ninguno logra contener en su totalidad un potencial incendio.

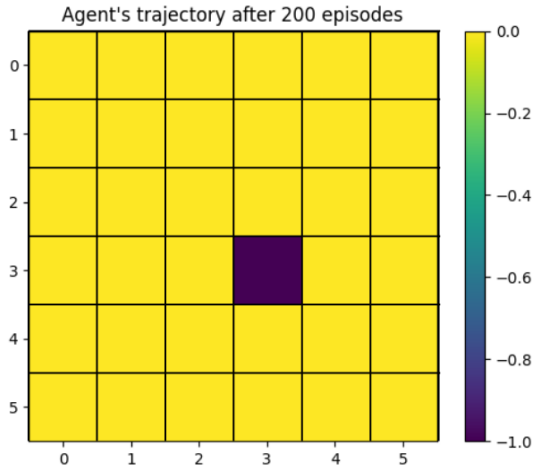
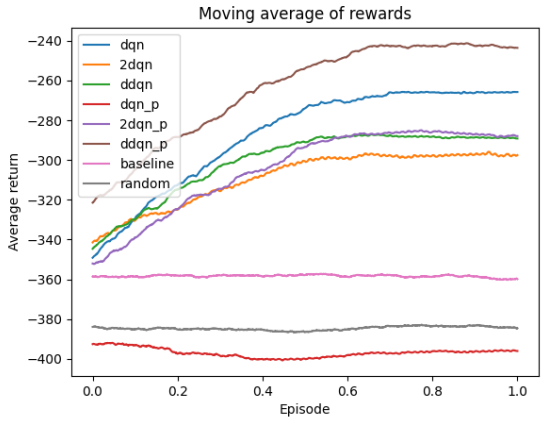


Figura 4.19: Mejor solución obtenida.

En la figura anterior, se puede apreciar que la solución obtenida mediante los algoritmos logra identificar que la gran mayoría del incendio se esparce por el centro, posicionando el cortafuegos en esta zona.

4.2.2. 10×10



(a) Promedios móviles de las recompensas obtenidas por episodio, la ventana sobre la cual se calcula es de 500 episodios.

(b) Promedios móviles de las recompensas obtenidas por episodio, la ventana sobre la cual se calcula es de 500 episodios.

Figura 4.20: Soluciones obtenidas para la instancia de 10×10, en ambiente v1.

Para la versión uno del ambiente, se puede apreciar que el rendimiento obtenido es superior al demostrador para la gran mayoría de los algoritmos. De la misma forma, la mayoría de los algoritmos implementados logra converger, aunque lo hacen a distintos puntos.

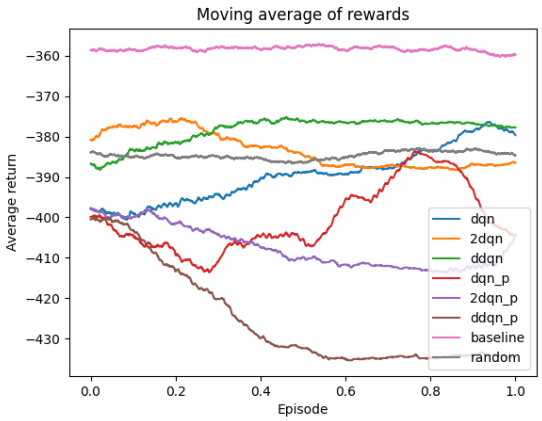
(a) Promedios móviles de las recompensas obtenidas por episodio, la ventana sobre la cual se calcula es de 500 episodios.

(b) Promedios móviles de las recompensas obtenidas por episodio, la ventana sobre la cual se calcula es de 500 episodios.

Figura 4.21: Soluciones obtenidas para la instancia de 10×10, en ambiente v2.

Para la versión dos, la mayoría de los algoritmos logra un rendimiento superior al del demostrador y converger. Sin embargo, esta convergencia es a distintos puntos, algunos de los cuales corresponden a rendimiento inferior al demostrados o incluso al nivel del algoritmo aleatorio.

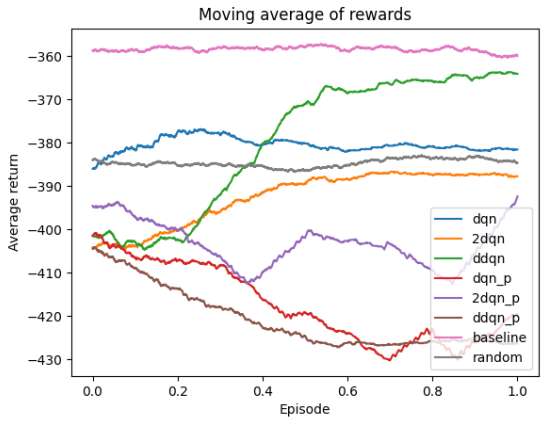
(a) Promedios móviles de las recompensas obtenidas por episodio para small net, la ventana sobre la cual se calcula es de 500 episodios.



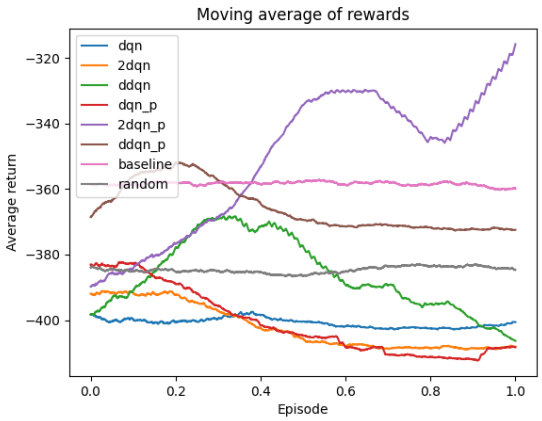
(b) Promedios móviles de las recompensas obtenidas por episodio para big net, la ventana sobre la cual se calcula es de 500 episodios.

Figura 4.22: Soluciones obtenidas para la instancia de 10×10, sin demostraciones y ambiente v1.

Para la versión uno del ambiente, pero sin demostraciones, se obtienen resultados inferiores al rendimiento del demostrador. No se logra convergencia en la mayoría de los algoritmos y una fracción considerable de estos logra rendimiento inferior al algoritmo aleatorio.



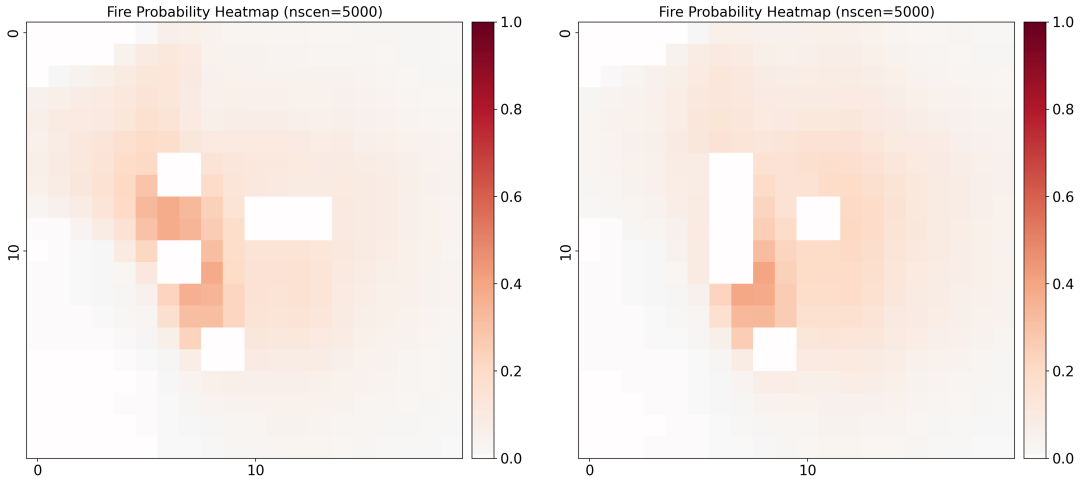
(a) Promedios móviles de las recompensas obtenidas por episodio para small net, la ventana sobre la cual se calcula es de 500 episodios.



(b) Promedios móviles de las recompensas obtenidas por episodio para big net, la ventana sobre la cual se calcula es de 500 episodios.

Figura 4.23: Soluciones obtenidas para la instancia de 10×10, sin demostraciones y ambiente v2.

En la versión dos del ambiente, se logra rendimiento inferior al del demostrador y similar al del algoritmo aleatorio. En general, no se logra convergencia.



(a) Mapa de probabilidad de quema para la mejor solución obtenida utilizando Downstream Protection Value.

(b) Mapa de probabilidad de quema para la mejor solución obtenida.

Figura 4.24: Rendimiento de las soluciones obtenidas.

Con respecto a la contención de un potencial incendio, ni la mejor solución de los algoritmos de Reinforcement Learning ni la del demostrador logran una contención completa. A pesar de lo anterior, pareciesen reducirse los porcentajes de quema a lo largo de todo el paisaje.

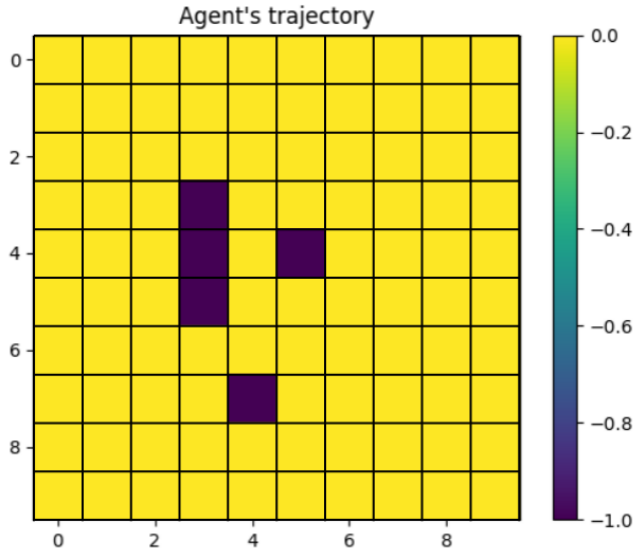
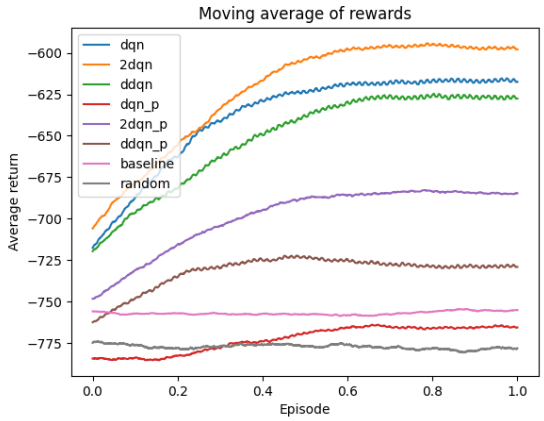


Figura 4.25: Mejor solución obtenida.

La mejor solución obtenida mediante los algoritmos implementados logra identificar que el esparcimiento se concentra en el centro del paisaje, posicionando cortafuegos a su alrededor.

4.2.3. 20×20

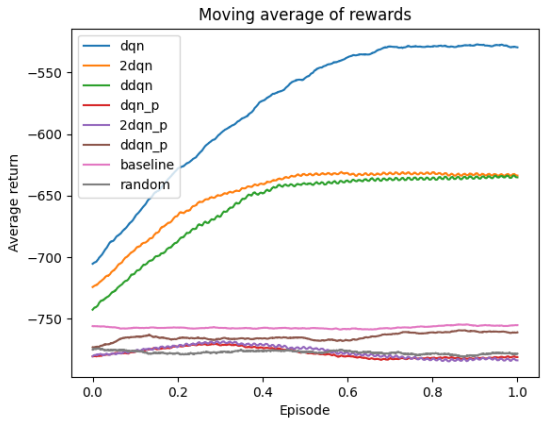


(a) Promedios móviles de las recompensas obtenidas por episodio, la ventana sobre la cual se calcula es de 500 episodios.

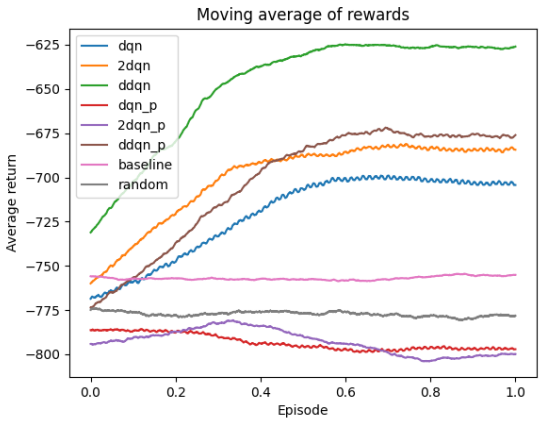
(b) Promedios móviles de las recompensas obtenidas por episodio, la ventana sobre la cual se calcula es de 500 episodios.

Figura 4.26: Soluciones obtenidas para la instancia de 20×20, en ambiente v1.

Como se puede apreciar en la figura, para la versión uno del ambiente, los algoritmos de Reinforcement Learning logran rendimientos superiores al del demostrador en la mayoría de los casos y convergencia. Sin embargo, pareciesen converger a soluciones distintas, algunas de las cuales tienen peor rendimiento que el demostrador y en el caso de Dueling Double Deep Q-Learning con Prioritized Experience Replay(ddqn_p) para big-net inferior al algoritmo aleatorio.



(a) Promedios móviles de las recompensas obtenidas por episodio, la ventana sobre la cual se calcula es de 500 episodios.

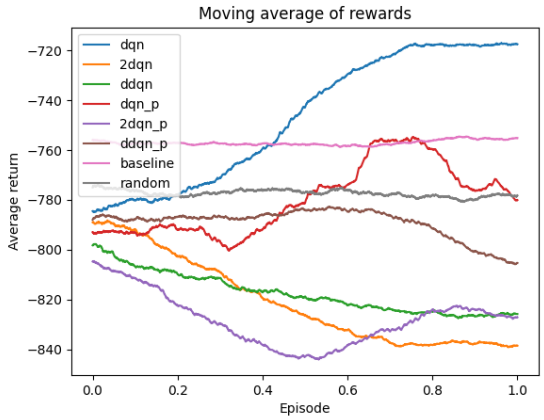


(b) Promedios móviles de las recompensas obtenidas por episodio, la ventana sobre la cual se calcula es de 500 episodios.

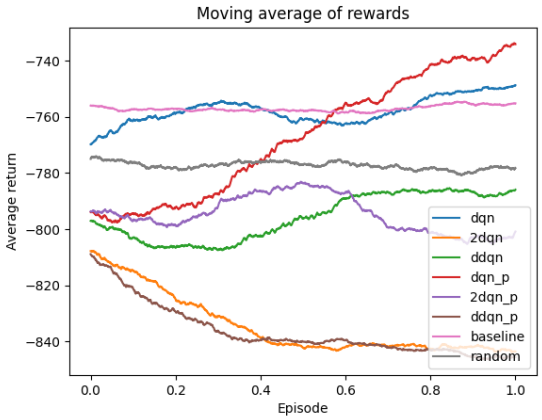
Figura 4.27: Soluciones obtenidas para la instancia de 20×20, en ambiente v2.

Para la versión dos del ambiente, gran parte de los algoritmos logran rendimiento superior al del demostrador, pero una fracción no menor no logra superarlo. Esto pareciese ser más

notorio para la arquitectura small-net. De manera similar, casi todos los algoritmos logran converger.



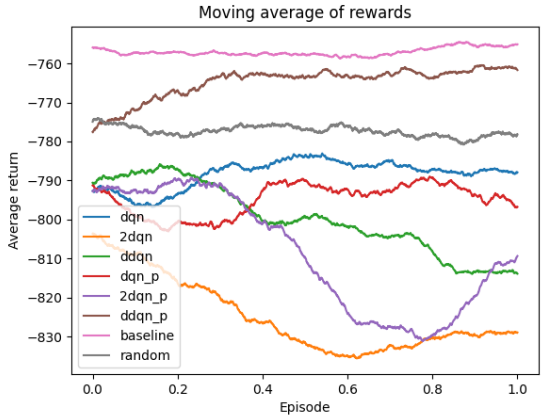
(a) Promedios móviles de las recompensas obtenidas por episodio para small net, la ventana sobre la cual se calcula es de 500 episodios.



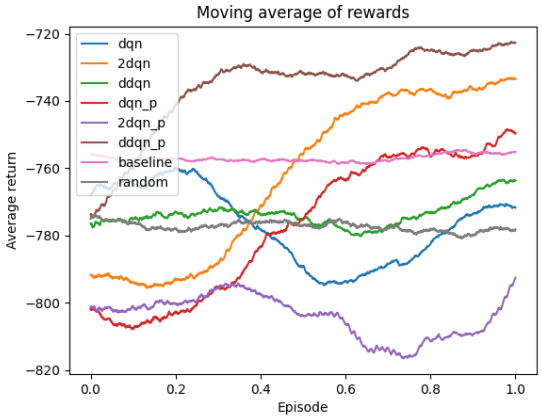
(b) Promedios móviles de las recompensas obtenidas por episodio para big net, la ventana sobre la cual se calcula es de 500 episodios.

Figura 4.28: Soluciones obtenidas para la instancia de 20×20, sin demostraciones y ambiente v1.

Al no utilizar demostraciones, para la versión uno del ambiente, no se logra superar el rendimiento del demostrador de manera consistente. Tampoco se logra convergencia en general. Aún más, una porción importante de los algoritmos obtiene rendimiento similar al algoritmo aleatorio.



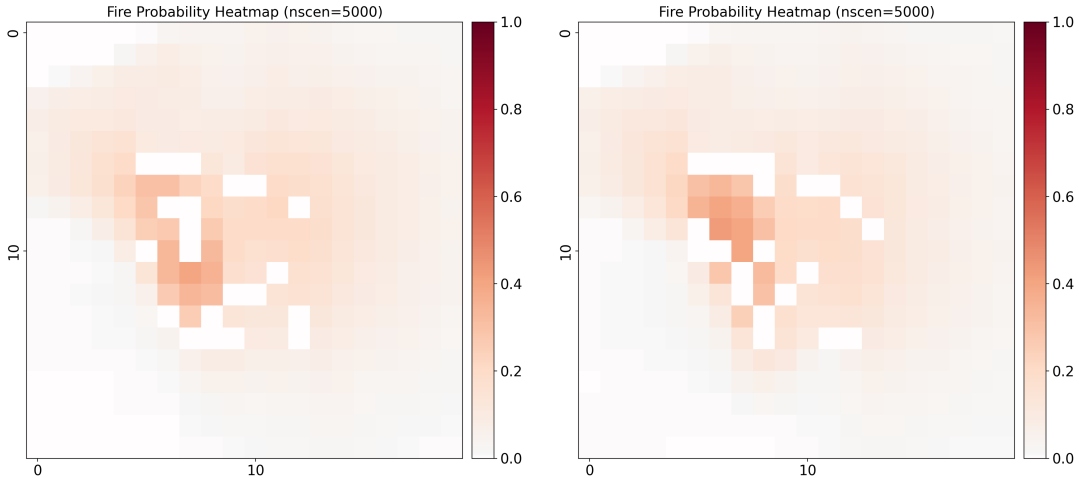
(a) Promedios móviles de las recompensas obtenidas por episodio para small net, la ventana sobre la cual se calcula es de 500 episodios.



(b) Promedios móviles de las recompensas obtenidas por episodio para big net, la ventana sobre la cual se calcula es de 500 episodios.

Figura 4.29: Soluciones obtenidas para la instancia de 20×20, sin demostraciones y ambiente v2.

Para la versión dos del ambiente, cuando no se ocupan demostraciones, no se logra superar el rendimiento del demostrador. Tampoco se logra convergencia en la mayoría de los casos. Al igual que para la versión uno, una subconjunto no menor de los algoritmos no logra superar el rendimiento del algoritmo aleatorio.



(a) Mapa de probabilidad de quema para la mejor solución obtenida utilizando Downstream Protection Value.

(b) Mapa de probabilidad de quema para la mejor solución obtenida.

Figura 4.30: Rendimiento de las soluciones obtenidas.

Con respecto a la contención, esta no se logra completamente con la mejor solución de los algoritmos implementados ni tampoco con la mejor solución del demostrador. A pesar de lo anterior, el esparcimiento del incendio pareciese reducirse considerablemente al compararlo con aquel presentado en 3.4.2.2.

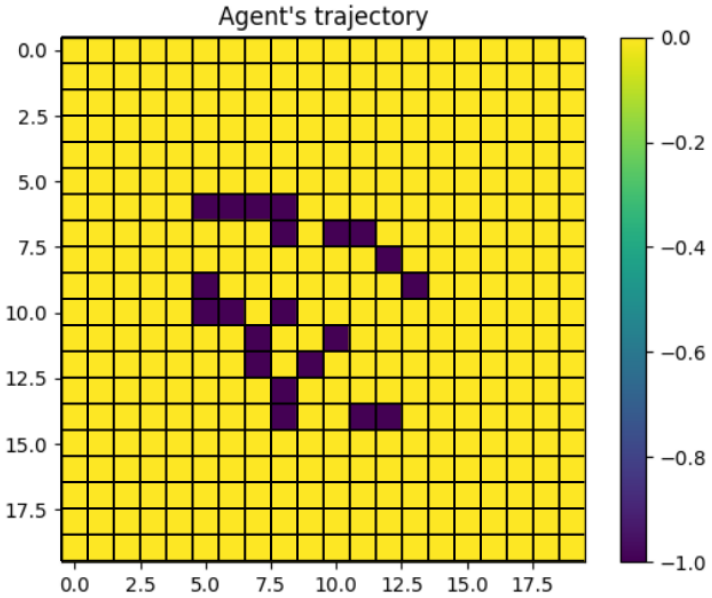


Figura 4.31: Mejor solución obtenida.

La mejor solución obtenida logra identificar que el esparcimiento de los incendios se concentra en el centro del paisaje, posicionando todos los cortafuegos en este sector.

Capítulo 5

Discusión

En general, se pueden encontrar dos comportamientos con respecto al rendimiento de los algoritmos utilizados. Por un lado, al emplear aprendizaje por demostración se obtienen resultados satisfactorios o medianamente satisfactorios a lo largo de todos los tamaños e instancias. En cambio, cuando no se utilizan demostraciones, los resultados obtenidos son insatisfactorios.

Se utilizarán dos enfoques para evaluar los distintos algoritmos, primero se compararán cualitativamente las curvas de recompensas de cada uno, esto debido a que estas muestran el rendimiento que se tiene para la tarea de Reinforcement Learning definida. En segundo lugar, se evalúa la contención del incendio para la mejor solución obtenida. En particular, debido a que el algoritmo demostrador es no determinista, la mejor solución se construyó tomando como parte de la solución aquellas celdas que fueron escogidas con mayor frecuencia.

En primer lugar, se analizan los resultados obtenidos en la sección 3.1, donde se conoce la solución óptima.

Tabla 5.1: Porcentaje de quema para los distintos tamaños del algoritmo presentado en 3.4.1 y la mejor solución obtenida con Deep Reinforcement Learning.

Tamaño	DPV	DRL
6×6	26 %	0 %
10×10	2.4 %	2.4 %
20×20	24.7 %	15.8 %

Tabla 5.2: Mejores recompensas obtenidas para cada algoritmo para la instancia homo_1, tamaño 6×6.

Algoritmo	Recompensas Promedio	Tamaño	Instancia
mab_ucb	0	6×6	homo_1
mab_greedy	0	6×6	homo_1
q_learning	0	6×6	homo_1
baseline	-510	6×6	homo_1

Para bosques pequeños(6×6 celdas) se obtienen resultados satisfactorios, todos los algoritmos logran convergencia, llegan a la solución óptima y al mismo nivel de recompensa(Tabla 5.2). Debido a que el problema es trivial, el resultado es esperable. Mediante la solución, se logra contener por completo un potencial incendio, no quemándose ninguna celda (Figura 4.2). Por otro lado, al comparar la contención del fuego con la mejor solución del demostrador, es claro que aquella obtenida mediante reinforcement learning supera con creces a esta última logrando suprimir completamente el esparcimiento de un incendio (Tabla 5.1).

Tabla 5.3: Mejores recompensas obtenidas para cada algoritmo para la instancia homo_1, tamaño 10×10.

Algoritmo	Recompensas Promedio	Tamaño	Instancia
dqn	-37	10 x 10	homo_1
2dqn	-34	10 x 10	homo_1
ddqn	-38	10 x 10	homo_1
dqn_p	-548	10 x 10	homo_1
2dqn_p	-368	10 x 10	homo_1
ddqn_p	-39	10 x 10	homo_1
baseline	-623	10 x 10	homo_1

En el caso de bosques medianos (10×10 celdas), utilizando demostraciones se obtienen resultados satisfactorios. En general, todos los algoritmos utilizados logran convergencia para ambas versiones del ambiente y arquitectura de red, pero esta no necesariamente es a la solución óptima. Respecto a lo anterior, se identifican tres casos: convergencia al óptimo; a rendimiento cercano al demostrador; y rendimiento inferior al demostrador pero superior a soluciones aleatorias. Para todas las configuraciones, al menos dos algoritmos logran alcanzar la solución óptima. De la misma manera, los algoritmos que caen en la segunda categoría varían entre uno y tres en las distintas configuraciones. Y para el tercer caso, varía entre uno y dos algoritmos. Con respecto a la optimalidad, dos algoritmos la logran alcanzar consistentemente: Deep Q-Learning (dqn) y Dueling Double Deep Q-Learning(ddqn). Con respecto al nivel de recompensas, el algoritmo que alcanza mejor rendimiento es el Double Deep Q-Learning(2dqn), alcanzando un mejor puntaje de -34 (Tabla 5.3). Por otro lado, el peor rendimiento se obtiene al utilizar Deep Q-Learning + Prioritized Experience Replay (dqn_p) en la mayoría de las instancias. Cabe destacar que la utilización de Prioritized Experience Replay no pareciese producir una mejora por sobre sus contrapartes con Experience Replay usual, en el mejor de los casos se obtiene igual rendimiento. Con respecto al rendimiento de la mejor solución obtenida, se logra contener completamente un potencial incendio, disminuyendo el porcentaje de quema desde un 50 % a un 2.4 % (Tabla 5.1). Esta solución obtiene el mismo rendimiento que la mejor solución del demostrador y de hecho son la misma (Figura 4.9).

A pesar de los buenos resultados obtenidos con demostraciones, sin estas se obtienen malos resultados tanto en términos de convergencia como rendimiento. Ningún algoritmo logra superar al demostrador y la gran mayoría obtiene rendimiento cercano a un algoritmo aleatorio.

Tabla 5.4: Mejores recompensas obtenidas para cada algoritmo para la instancia homo_1, tamaño 20×20.

Algoritmo	Recompensas Promedio	Tamaño	Instancia
dqn	-1498	20 x 20	homo_1
2dqn	-1404	20 x 20	homo_1
ddqn	-1528	20 x 20	homo_1
dqn_p	-1548	20 x 20	homo_1
2dqn_p	-1439	20 x 20	homo_1
ddqn_p	-1548	20 x 20	homo_1
baseline	-1628	20 x 20	homo_1

Para bosques grandes(20×20 celdas), se logra convergencia a lo largo de todos los algoritmos y configuraciones. Sin embargo, en términos de rendimiento no se obtienen resultados satisfactorios. ya que ningún algoritmo logra acercarse al óptimo (Figura 4.12). A pesar de lo anterior, en la gran mayoría de los algoritmos se obtiene un rendimiento superior al demostrador. No hay un algoritmo que domine al resto, pero se repite la tendencia descrita en el caso anterior: la utilización de Prioritized Experience Replay no pareciese producir una mejora consistente. El mejor nivel de recompensas se alcanza con Double Deep Q-Learning(2dqn), alcanzando un nivel de -1404 (Tabla 5.4) Con respecto al rendimiento de la mejor solución obtenida, se logra contener parcialmente un potencial incendio, disminuyendo el porcentaje de quema desde un 50 % a un 15.8 % (Figura 4.15). Nuevamente, se repite el resultado del caso anterior sin demostraciones.

En segundo lugar, se analizarán los resultados obtenidos en la sección 3.2, donde no se conoce la solución óptima.

Tabla 5.5: Porcentaje de quema para los distintos tamaños del algoritmo presentado en 3.4.1 y la mejor solución obtenida con Deep Reinforcement Learning.

Tamaño	DPV	DRL
6×6	11 %	11.4 %
10×10	7 %	7.1 %
20×20	7.1 %	7 %

Tabla 5.6: Mejores recompensas obtenidas para cada algoritmo para la instancia homo_2, tamaño 6×6.

Algoritmo	Recompensas Promedio	Tamaño	Instancia
mab_ucb	-181	6x6	homo_2
mab_greedy	-183	6x6	homo_2
q_learning	-178	6x6	homo_2
baseline	-223	6x6	homo_2

Para bosques pequeños, se obtienen resultados satisfactorios en términos de convergencia, todos los algoritmos logran hacerlo y además al mismo punto. En particular, es interesante que al algoritmo Q-Learning y Multi-Arm Bandits con epsilon-greedy les tome más tiempo en converger que para la instancia anterior, fueron necesarios 4000 y 1000 episodios correspondientemente. Lo anterior se puede deber a la estocasticidad de las recompensas para cada posible solución. En la instancia con solución conocida, la solución óptima obtiene siempre como recompensa 0, independiente de la dirección del viento y punto de ignición. En cambio, aquí cada posible solución tiene recompensas mucho más ruidosas asociadas a las mismas variables que antes. Sin embargo, esta estocasticidad no pareciese afectar de la misma manera al algoritmo Multi-Arm Bandits Upper Confidence Bound, ya que a pesar de que le tomó mas episodios para converger que para la instancia pasada, no lo fue en la misma medida que los otros dos. El algoritmo que obtiene el mejor nivel de recompensas es Q-Learning(q_learning), alcanzando -178(Tabla 5.6). Mediante la solución generada, se logra contener parcialmente un potencial incendio, disminuyendo el porcentaje de quema desde 25 % tan solo a 11.4 % (Figura 4.18).

Tabla 5.7: Mejores recompensas obtenidas para cada algoritmo para la instancia homo_2, tamaño 10×10.

Algoritmo	Recompensas Promedio	Tamaño	Instancia
dqn	-269	10 x 10	homo_2
2dqn	-277	10 x 10	homo_2
ddqn	-274	10 x 10	homo_2
dqn_p	-367	10 x 10	homo_2
2dqn_p	-296	10 x 10	homo_2
ddqn_p	-251	10 x 10	homo_2
baseline	-358	10 x 10	homo_2

Para bosques medianos, se obtienen resultados satisfactorios al alcanzarse la convergencia y superarse el rendimiento del demostrador en la mayoría de los casos. Algo importante a destacar, es que la distancia entre el rendimiento del demostrador y el algoritmo aleatorio, es considerablemente menor que para la instancia con solución conocida. Al estar el rendimiento de los algoritmos sujeto al rendimiento del demostrador, esto provoca que las soluciones obtenidas estén mas cerca de las soluciones aleatorias. Igualmente se observa que ningún algoritmo domina al resto consistentemente, sin embargo, Dueling Double Deep Q-Learning(ddqn) logra obtener el mejor rendimiento en tres de las cuatro configuraciones. El mejor nivel de recompensas se obtiene para Dueling Double Q-Learning con Prititized Experience Replay(ddqn_p), con -251(Tabla 5.7). Con respecto al rendimiento de la mejor solución obtenida, se logra contener parcialmente un potencial incendio, disminuyendo el porcentaje de quema desde un 25 % a un 7.1 % (Figura 4.24). Nuevamente, se repite el resultado de casos anteriores sin demostraciones. Se destaca que los algoritmos parecieran diverger incluso más que en la sección 4.1, sin embargo esto puede deberse a que la escala de los resultados es más acotada.

Tabla 5.8: Mejores recompensas obtenidas para cada algoritmo para la instancia homo_1, tamaño 6×6.

Algoritmo	Recompensas Promedio	Tamaño	Instancia
dqn	-541	20 x 20	homo_2
2dqn	-601	20 x 20	homo_2
ddqn	-627	20 x 20	homo_2
dqn_p	-755	20 x 20	homo_2
2dqn_p	-689	20 x 20	homo_2
ddqn_p	-677	20 x 20	homo_2
baseline	-756	20 x 20	homo_2

En el caso de bosques grandes, se logra convergencia a lo largo de todos los algoritmos y configuraciones. Sin embargo, en términos de rendimiento no se obtienen resultados satisfactorios, debido a que los resultados son peores que aquellos obtenidos para el mismo tamaño en la sección 4.1. Solo tres de los seis algoritmos logran superar el rendimiento del demostrador a lo largo de todas las configuraciones: Dueling Double Deep Q-Learning(ddqn); Double Deep Q-Learning(2dqn); y Deep Q-Learning(dqn). El mejor nivel de recompensas se obtiene para Deep Q-Learning, alcanzando -541. En este caso particular, la utilización de Prioritized Experience Replay trae peores resultados para los algoritmos mencionados. Nuevamente, los rendimientos asociados al demostrador y al algoritmo aleatorio son parecidos, incluso más que para bosques medianos. Con respecto al rendimiento de la mejor solución obtenida, se logra contener parcialmente un potencial incendio, disminuyendo el porcentaje de quema desde un 25 % a un 7 % (Figura 4.30). Se destaca que la disminución de este porcentaje con respecto a bosques medianos es muy pequeña. Nuevamente, se repite el resultado obtenido para bosques medianos sin demostraciones.

Un resultado general a las instancias, tamaños y configuraciones es que la incorporación de la matriz de conectividad definida en la sección 3.2.1 no genera una mejora en el rendimiento del agente., lo cual puede deberse a dos razones. Por un lado, puede ser que esta no aporte información relevante al problema y que el agente sea capaz de resolver el problema sin ella. O por otro, puede ser que las redes neuronales utilizadas fueron capaces de extraer esta información mediante los bloques convolucionales utilizados y la ocuparon para resolver el problema. De manera similar, se observan los mismos resultados al comparar las dos arquitecturas de red utilizadas. Esto tendería a insinuar que los problemas de convergencia y/o rendimiento no tienen que ver con la profundidad de la red, sino más con el algoritmo de aprendizaje utilizado.

Tabla 5.9: Comparación de las mejores soluciones obtenidas en términos de porcentaje de quema.

Tamaño	DPV	DRL	Instancia
6×6	26 %	0 %	homo_1
6×6	11 %	11.4 %	homo_2
10×10	2.4 %	2.4 %	homo_1
10×10	7 %	7.1 %	homo_2
20×20	24.7 %	15.8 %	homo_1
20×20	7.1 %	7 %	homo_2

Habiendo analizado los resultados obtenidos, se está en posición de analizar de manera global la efectividad de Reinforcement Learning para resolver el problema de la ubicación de cortafuegos. Comparando las mejores soluciones obtenidas (Tabla 5.9), se puede concluir que se obtiene rendimiento igual o superior a la heurística utilizada como demostrador para todos los tamaños e instancias. Esto posiciona a esta técnica como una opción viable y potencialmente mejor que los enfoques usuales para problemas de este tipo.

Capítulo 6

Conclusión

La principal motivación de esta tesis era evaluar la factibilidad de utilizar Deep Reinforcement Learning para resolver un problema desafiante de Diseño de Paisajes. Lo anterior se logró, bajo ciertas condiciones. En primer lugar, considerando los algoritmos que se utilizaron, la incorporación de demostraciones es fundamental para obtener buenos resultados. Sin estas, no se logra convergencia ni buen rendimiento. Lo anterior se puede deber a diversas razones, dentro de las cuales están: un ambiente con una estocasticidad tan alta que no le permite al algoritmo encontrar soluciones consistentemente buenas; un espacio de acciones en donde la gran mayoría no trae buenos resultados generando que los algoritmos necesiten una guía para moverse a secciones del espacio con mejores rendimientos; el hecho de que los algoritmos utilizados pueden no ser lo suficientemente sofisticados como para resolver problemas así de complejos; y la inexistencia de feedback individual sobre cada cortafuego la que se traduce en una estructura de recompensas escasa, que es propia del problema. En segundo lugar, el rendimiento del algoritmo está anclado a aquel del demostrador. Esto es un arma de doble filo: por un lado permite al agente llegar a un nivel de rendimiento razonable rápidamente, pero por otro, genera que este no se desvie mucho del demostrador y si este está lejos del comportamiento óptimo también lo estará el algoritmo. De esta forma, el aprendizaje por demostración tal como se ocupó en este trabajo se recomienda si 1) se conoce una heurística que resuelva el problema rápidamente y 2) el rendimiento de esta no está demasiado lejos del comportamiento óptimo (o aquel que se quiera alcanzar).

Una segunda motivación para este trabajo, era ver si se podían utilizar estas técnicas para mejorar el rendimiento de un algoritmo subóptimo. Esto se cumplió a cabalidad, bajo todas las instancias se logró superar este algoritmo, en ciertas con una diferencia mayor que otras, pero en todas.

Un elemento a destacar de la utilización de técnicas de Reinforcement Learning en escenarios como el anterior, es la facilidad con la que se pueden incorporar restricciones propias del problema. Para lo anterior, basta restringir el comportamiento del agente a nivel de la red neuronal, en contraste con lo que significa hacerlo en un modelo de programación entera mixta. Plantear restricciones en estos últimos en general es complejo y pueden generar complicaciones en los algoritmos de resolución asociados.

Es importante mencionar que la aplicación de este trabajo en condiciones reales es extremadamente limitada. Los algoritmos entrenados, están ajustados a una y solo una instancia.

Esto quiere decir que de querer resolverse un problema diferente, debe entrenarse el agente desde cero. Y no solo eso, deben generarse las demostraciones correspondientes para esa instancia. Este es un proceso costoso y debe evaluarse si merece la pena. Aún más, el rendimiento de los algoritmos entrenados mediante la metodología utilizada es fuertemente dependiente del tamaño de la instancia considerada, por tanto para instancias mayores se espera que el rendimiento empeore considerablemente.

Finalmente, es importante mencionar las implicancias de entrenar un agente de Reinforcement Learning que sea exitoso para resolver el problema de cortafuegos sobre múltiples instancias. Lograr programar un agente capaz de generalizar significaría que este podría resolver el problema para nuevas instancias de forma casi instantánea. Lo anterior, debido a que una vez que el agente aprende, en teoría no sería necesario entrenarlo denuevo para nuevos datos. Incluso relajando lo anterior, si se es capaz de entrenar un agente que obtiene rendimiento razonable sobre múltiples instancias, existiría la posibilidad de que mediante un entrenamiento corto, como el fine-tuning de Large Language Models como GPT, se logre obtener rendimientos iguales ó superiores al estado del arte utilizando una fracción del tiempo de cómputo. Esto pondría a esta disciplina en una posición ventajosa en comparación a los enfoques actuales y sugiere que avanzar en esta línea podría ser valioso.

6.1. Trabajo Futuro

Un primer punto de mejora y desarrollo corresponde a un ingeniería más profunda en lo que respecta a la estructuctura de recompensas. Es un hecho ampliamente conocido el que una estructura escasa es perjudicial para el aprendizaje. Mejorar esto no es fácil en este contexto, evaluar el efecto individual de un cortafuego no es para nada directo. Sin embargo, una de los posibles formas que se podría explorar es considerar el rendimiento histórico para un cortafuego en específico cada vez que se simuló un incendio. Lo anterior podría mejorar el rendimiento de los algoritmos utilizados, teniendo en consideración que estas recompensas intermedias no deben eclipsar la recompensa obtenida al simular incendios sobre el cortafuegos construido completo, que es la recompensa más importante y la que está correcta de todas formas.

En segundo lugar, se mencionó en las secciones anteriores que los algoritmos utilizados no son los más sofisticados. Existe una familia completa de algoritmos denominados Policy Gradient que no se utilizaron, dentro de los cuales varios se muestran prometedores y se consideran en el estado del arte. Dentro de estos están Trust Policy Optimization(Schulman, Levine, Abbeel, Jordan, y Moritz, 2015) y Proximal Policy Optimization(Schulman, Wolski, Dhariwal, Radford, y Klimov, 2017). Además de lo anterior, existe un enfoque considerablemente distinto al que se utilizó: los métodos basados en modelo(Moerland, Broekens, y Jonker, 2020). En estos, se intentan aproximar las probabilidades de transición del sistema(ver sección 2.2.2.1) y se utilizan estas para resolver el problema.

En tercer lugar, existe una importante posibilidad de mejora en la arquitectura de red utilizada. A pesar de que las redes convolucionales están dentro de lo que se considera estado del arte, existen otras que pueden ser más apropiadas para los datos que se le entregan a las redes en este caso. Un fuerte candidato sería utilizar redes neuronales recurrentes(Hochreiter y Schmidhuber, 1997) o transformers(Vaswani et al., 2017), en conjunto con convolucionales,

las cuales permiten procesar inputs secuenciales.

Finalmente, una linea en la que buscará avanzar es en la utilización de técnicas de explicabilidad(Selvaraju et al., 2016) para dilucidar en qué se está fijando el agente a la hora de posicionar un cortafuego, esto con objetivo de inspirar nuevos algoritmos o formas de resolver el problema.

Bibliografía

Albini, B. J., F.A. (1996). Mathematical modeling and predicting wildland fire effects. *Combust Explos Shock Waves*, *32*, 520–533.

Balaji, B., Bell-Masterson, J., Bilgin, E., Damianou, A. C., Garcia, P. M., Jain, A., ... Ye, C. J. (2019). Orl: Reinforcement learning benchmarks for online stochastic optimization problems. *ArXiv*, *abs/1911.10641*.

Bengio, Y., Lodi, A., y Prouvost, A. (2021). Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, *290*(2), 405–421.

Cristobal Pais, P. E. M. Z.-J. M. S., Jaime Carrasco. (2021). Downstream protection value: Detecting critical zones for effective fuel-treatment under wildfire risk. *Computers Operations Research*, *131*, 305-548.

Crutzen, P. J., y Goldammer, J. G. (1994). Fire in the environment : the ecological, atmospheric, and climatic importance of vegetation fires : report of the dahlem workshop, held in berlin, 15-20 march 1992..

Halofsky, J. E., Peterson, D. L., y Harvey, B. J. (2020). Changing wildfire, changing forests: the effects of climate change on fire regimes and vegetation in the pacific northwest, usa. *Fire Ecology*, *16*, 1-26.

Hammond, T., Schaap, D., Sabatelli, M., y Wiering, M. A. (2020). Forest fire control with learning from demonstration and reinforcement learning. *2020 International Joint Conference on Neural Networks (IJCNN)*, 1-8.

Hanna, J., Niekum, S., y Stone, P. (2018). Importance sampling policy evaluation with an estimated behavior policy. *CoRR*, *abs/1806.01347*.

Hasselt, H. V., Guez, A., y Silver, D. (2015). Deep reinforcement learning with double q-learning. En *Aaai conference on artificial intelligence.*

Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., ... Gruslys, A. (2018, 04). Deep q-learning from demonstrations. *Proceedings of the AAAI Conference on Artificial Intelligence*, *32*. doi: 10.1609/aaai.v32i1.11757

Hochreiter, S., y Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*, 1735-1780.

Hubbs, C. D., Perez, H. D., Sarwar, O., Sahinidis, N. V., Grossmann, I. E., y Wassick, J. M. (2020). Or-gym: A reinforcement learning library for operations research problem. *ArXiv*, *abs/2008.06319*.

Knegt, S. J. L., Drugan, M. M., y Wiering, M. A. (2018). Opponent modelling in the game of tron using reinforcement learning. En *International conference on agents and artificial*

intelligence.

Kober, J., Bagnell, J. A., y Peters, J. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, *32*(11), 1238–1274.

Kool, W., van Hoof, H., y Welling, M. (2018). Attention, learn to solve routing problems! En *International conference on learning representations.*

Krizhevsky, A., Sutskever, I., y Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, *60*, 84 - 90.

LeCun, Y., Bengio, Y., y Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444.

Lin, L. (1992). Reinforcement learning for robots using neural networks..

Mahaluf Recasens, R. (2022). *Aplicaciones de modelos de optimización para la preservación de la biodiversidad frente a incendios forestales.* (Tesis de Master no publicada). Universidad de Chile.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., y Riedmiller, M. A. (2013). Playing atari with deep reinforcement learning. *ArXiv*, *abs/1312.5602*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A., Veness, J., Bellemare, M., ... Hassabis, D. (2015, 02). Human-level control through deep reinforcement learning. *Nature*, *518*, 529-33. doi: 10.1038/nature14236

Moerland, T. M., Broekens, J., y Jonker, C. M. (2020). Model-based reinforcement learning: A survey. *Found. Trends Mach. Learn.*, *16*, 1-118.

Oroojlooyjadid, A., Nazari, M., Snyder, L. V., y Takác, M. (2021). A deep q-network for the beer game: Deep reinforcement learning for inventory optimization. *Manuf. Serv. Oper. Manag.*, *24*, 285-304.

Pais, C., Carrasco, J., Martell, D., Weintraub, A., y Woodruff, D. (2021). Cell2fire: A cell-based forest fire growth model to support strategic landscape management planning. front. for. glob. *Change*, *4*, 692706.

Palacios Meneses, D. (2022). *Comparación de metaheurísticas para la ubicación de cortafuegos en el combate de incendios forestales.* (Tesis de Master no publicada). Universidad de Chile.

Rocholl, N. (2021). *Isolating wildfires using a convolutional neural network based multi-agent system* (Tesis de Master no publicada). University of Groningen.

Rumelhart, D. E., Hinton, G. E., y Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536.

Russo L, S. C., Russo P. (2016). A complex network theory approach for the spatial distribution of fire breaks in heterogeneous forest landscapes for the control of wildland fires. *PLoS ONE*, *11*, 10.

Schaul, T., Quan, J., Antonoglou, I., y Silver, D. (2015). Prioritized experience replay. *CoRR*, *abs/1511.05952*.

Schulman, J., Levine, S., Abbeel, P., Jordan, M. I., y Moritz, P. (2015). Trust region policy optimization. *ArXiv*, *abs/1502.05477*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., y Klimov, O. (2017). Proximal policy optimization algorithms.

Selvaraju, R. R., Das, A., Vedantam, R., Cogswell, M., Parikh, D., y Batra, D. (2016). Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization. *CoRR*, *abs/1610.02391*.

Shantia, A., Begue, E., y Wiering, M. (2011, 09). Connectionist reinforcement learning for intelligent unit micro management in starcraft. En (p. 1794 - 1801). doi: 10.1109/IJCNN.2011.6033442

Silver, H. A. M. C., D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, *529*, 484–489.

Silver, H. T. S. J., D. (2016). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, *362*(6419), 1140-1144.

Stephens, S. L., Burrows, N., Buyantuyev, A., Gray, R. W., Keane, R. E., Kubian, R., ... others (2014). Temperate and boreal forest mega-fires: Characteristics and challenges. *Frontiers in Ecology and the Environment*, *12*(2), 115–122.

Sutton, R. S., y Barto, A. G. (2005). Reinforcement learning: An introduction. *IEEE Transactions on Neural Networks*, *16*, 285-286.

van Wagner, C. E. (1998). Modelling logic and the canadian forest fire behavior prediction system. *Forestry Chronicle*, *74*, 50-52.

Vaswani, A., Shazeer, N. M., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. En *Nips.*

Wang, Z., Freitas, N., y Lanctot, M. (2015, 11). Dueling network architectures for deep reinforcement learning.

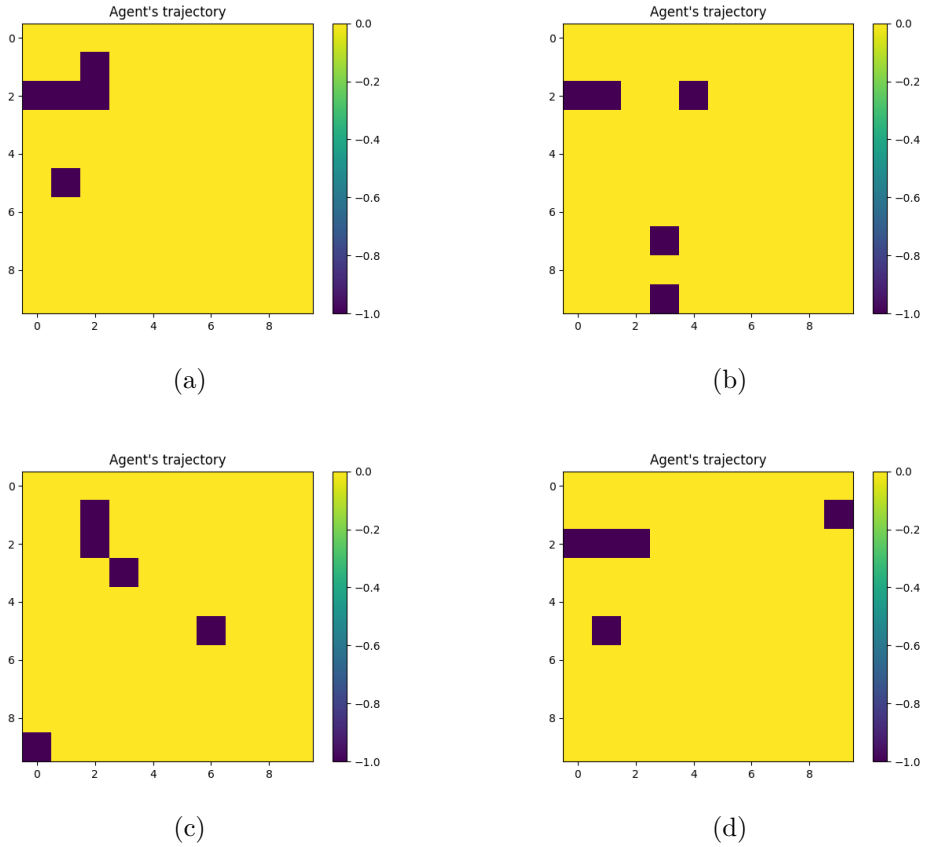
Watkins, C., y Dayan, P. (1992). Q-learning. *Machine Learning*, *8*, 279-292.

Anexos

Anexo A. Demostraciones

A.1. 10×10

A.1.1. Solución Conocida

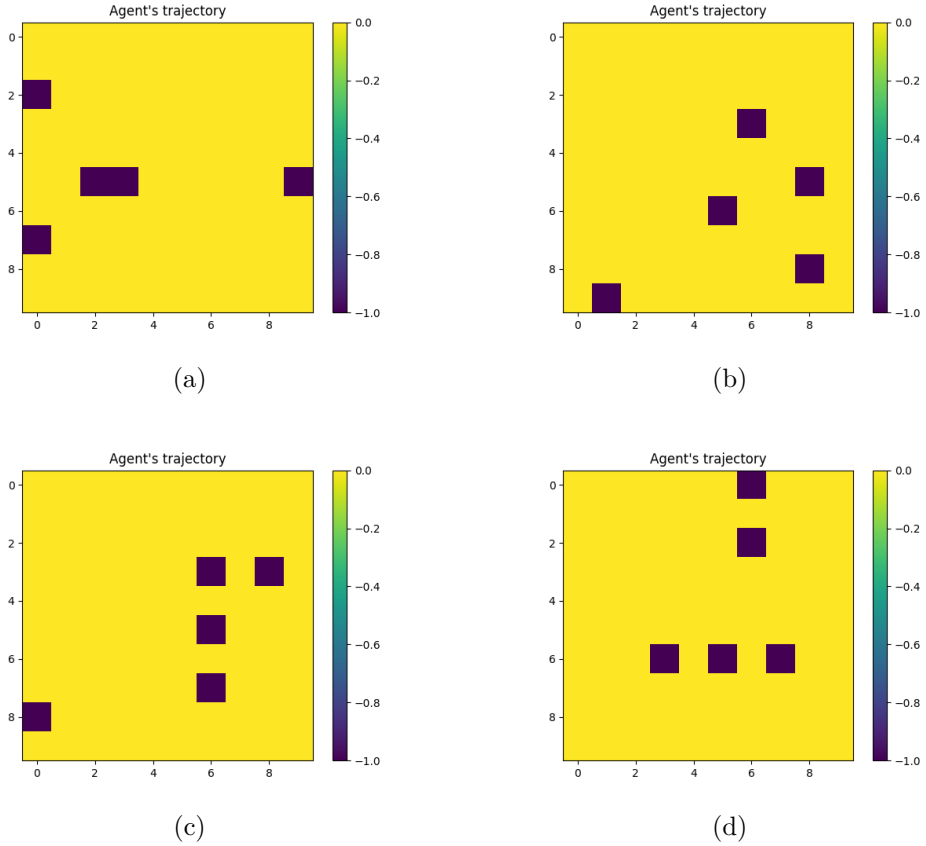


(a) (b)

(c) (d)

Figura A.1: Muestra de las demostraciones utilizadas para la instancia con solución conocida de 10×10.

A.1.2. Solución Desconocida



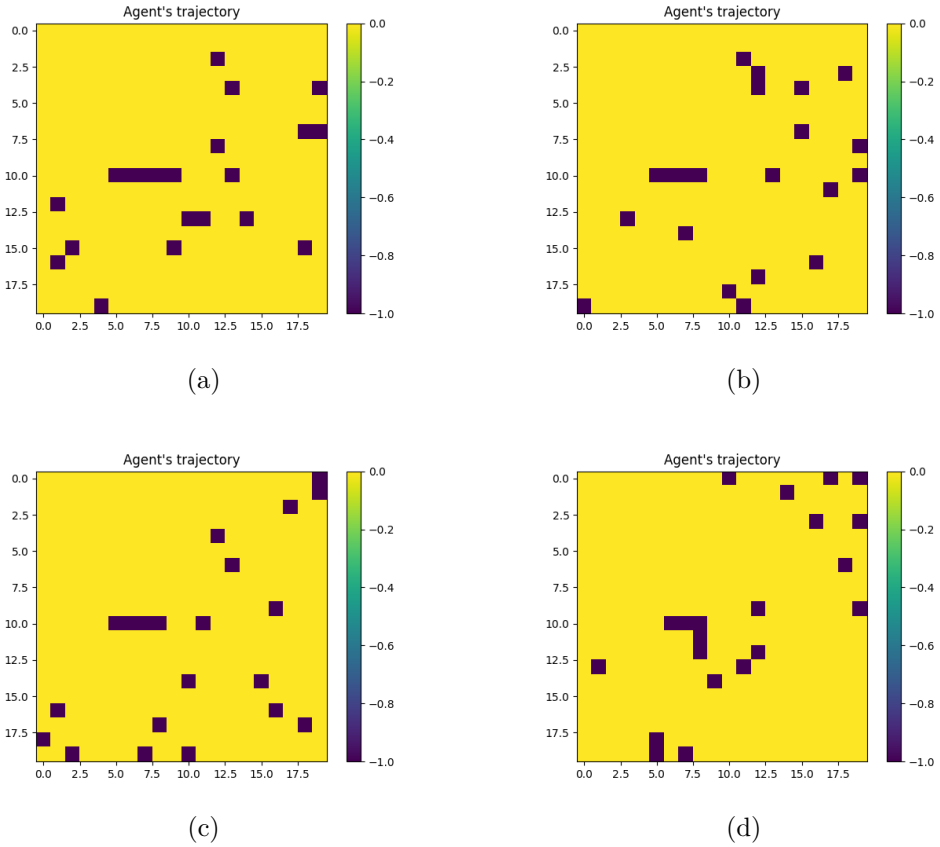
(a) (b)

(c) (d)

Figura A.2: Muestra de las demostraciones utilizadas para la instancia con solución desconocida de 10x10.

A.2. 20×20

A.2.1. Solución Conocida

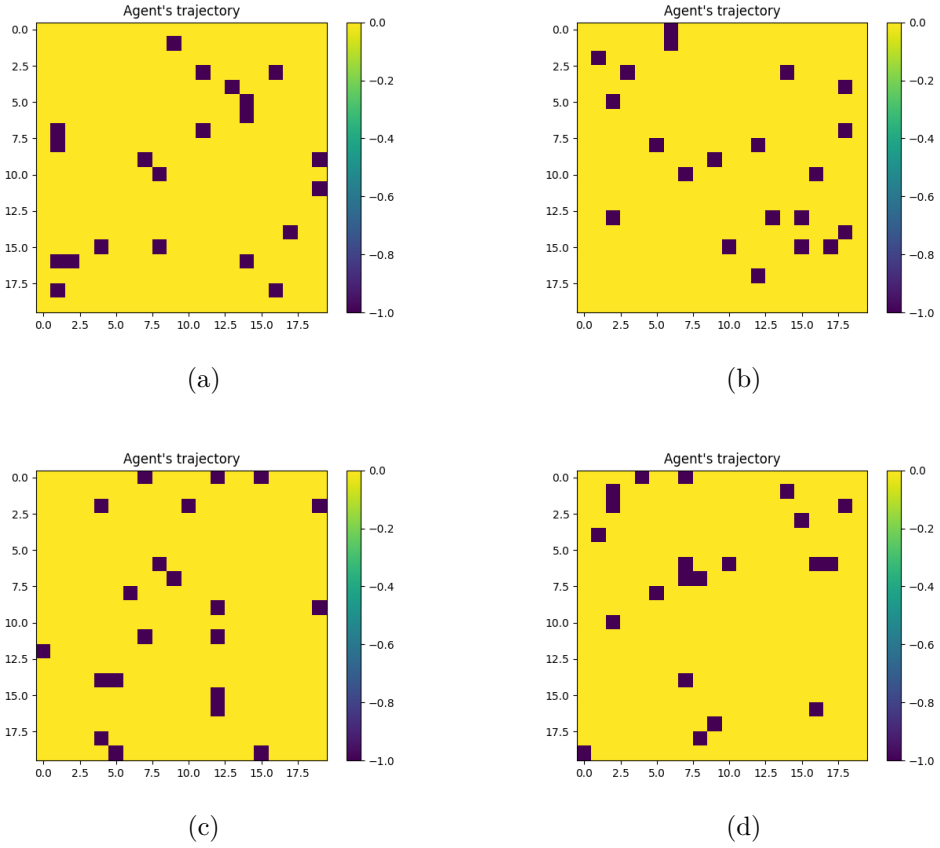


(a) (b)

(c) (d)

Figura A.3: Muestra de las demostraciones utilizadas para la instancia con solución conocida de 20×20.

A.2.2. Solución Desconocida



(a) (b)

(c) (d)

Figura A.4: Muestra de las demostraciones utilizadas para la instancia con solución desconocida de 20×20.