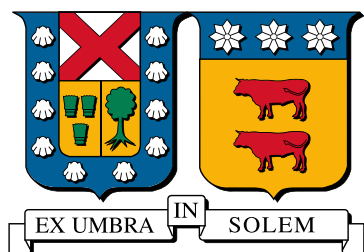


UNIVERSIDAD TÉCNICA FEDERICO SANTA
MARÍA

DEPARTAMENTO DE ELECTRÓNICA

VALPARAÍSO - CHILE



“EVALUACIÓN DEL DESEMPEÑO DE
NUEVOS ALGORITMOS PBRL AL
ENFRENTARSE A PROFESORES QUE
PUEDEN COMETER ERRORES”

DAVID ALEJANDRO ISRAEL TAPIA PINO

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL
ELECTRÓNICO.

PROFESOR GUIA: DR. WERNER CREIXELL FUENTES
PROFESOR CORREFERENTE: DR. MAURICIO LÓPEZ ARAYA

MARZO 2024

Evaluación del desempeño de nuevos algoritmos PBRL al enfrentarse a profesores que pueden cometer errores

David Alejandro Israel Tapia Pino

Memoria para optar al título de Ingeniero Civil Electrónico, mención Computadores,
submención Control e Instrumentación.

Universidad Técnica Federico Santa María

Profesor Guía: Dr. Werner Creixell Fuentes

Profesor Correferente: Dr. Mauricio López Araya

MARZO 2024

Resumen

Este trabajo presenta la implementación del benchmark para algoritmos de aprendizaje reforzado basado en preferencias llamado B-Pref en la familia de algoritmos Dueling Posterior Sampling para PBRL. El objetivo de esta investigación es evaluar cuál sistema de asignación de crédito resulta ser más resistente a posibles errores humanos, con la intención de servir como un primer paso para el desarrollo de futuros algoritmos cada vez más robustos al error humano y que por ende puedan ser utilizados por usuarios normales. El documento detalla desde la revisión del estado del arte en el aprendizaje reforzado basado en preferencias y las adaptaciones que tuvieron que hacerse para la implementación de la prueba hasta los resultados y conclusiones a las que se llegaron al evaluar las decisiones de diseño de DPS y su desempeño en comparación con otros algoritmos como EPMC y PSRL.

Evaluation of the performance of new PBRL algorithms when facing teachers who may make mistakes.

David Alejandro Israel Tapia Pino

Thesis for the fulfillment of the B.S. in Electronic Engineering, major in Computer Electronics, minor in Control and Instrumentation (6 year program).

Universidad Técnica Federico Santa María

Advisor: Dr. Werner Creixell Fuentes

Co-Advisor: Dr. Mauricio López Araya

MARZO 2024

Abstract

This thesis presents the implementation of the benchmark for preference-based reinforcement learning algorithms called B-Pref in the Dueling Posterior Sampling for PBRL family of algorithms. The objective of this research is to evaluate which credit assignment system proves to be more resilient to possible human errors, with the intention of serving as a first step towards the development of future algorithms increasingly robust to human error and therefore usable by ordinary people. The document details from the review of the state of the art in preference-based reinforcement learning and the adaptations that had to be made for the implementation of the test to the results and conclusions reached when evaluating the design decisions of DPS and its performance compared to other algorithms such as EPMC and PSRL.

Glosario

B-Pref Benchmark para algoritmos de aprendizaje reforzado basado en preferencias. Busca simular profesores humanos más realistas que los testbench existentes.

Benchmark Estándar para evaluar el desempeño de algoritmos y realizar comparaciones entre ellos.

Clase En programación orientada a objetos, una plantilla para crear objetos que encapsulan datos y métodos.

Distribución de Dirichlet Distribución de probabilidad multivariada que modela la aleatoriedad de variables aleatorias sobre un conjunto de categorías, comúnmente usada en inferencia bayesiana y aprendizaje por refuerzo.

DPS Familia de algoritmos PBRL basados en el muestreo de Thompson. En estos algoritmos el agente trata de aprender cómo funciona el ambiente a la vez que trata de aprender la mejor forma de realizar la tarea. (Dueling Posterior Sampling).

EPMC Algoritmo de PBRL que busca mejorar su política mediante la estimación de valores de estados a partir de muestras de episodios completos. (Every-Visit Preference Monte Carlo).

Exploración Probar acciones desconocidas en busca del óptimo. Demasiada exploración podría hacer que no se aproveche lo ya conocido.

Explotación Tomar acciones que ya se sabe que funcionan, aprovechar al máximo lo que ya se conoce. Demasiada explotación podría dejar atrapado al agente en un máximo local, quizás el óptimo está en otro lado.

Función de Recompensas Es utilizada por el algoritmo RL para determinar qué tan bien está realizando una tarea. En RL tradicional debe ser definida por el programador.

Función Logística Función sigmoide ($\frac{1}{1+e^{-t}}$) que mapea valores reales a un rango entre 0 y 1, comúnmente utilizada en problemas de clasificación y regresión.

GP Procesos Gaussianos, son una clase de modelos estadísticos que representan una colección infinita de variables aleatorias, donde cualquier subconjunto de estas variables sigue una distribución gaussiana multivariada. Destacan por proporcionar una manera flexible de capturar la incertidumbre y la correlación entre los datos. (Gaussian Process).

GPR La regresión de procesos gaussianos es una técnica de modelado no paramétrico que se utiliza para predecir valores de una variable continua a partir de datos de entrenamiento. Se basa en la idea de que la función objetivo es una realización de un proceso gaussiano, lo que permite modelar la incertidumbre en las predicciones y capturar relaciones no lineales entre las variables. (Gaussian Process Regression).

IRL Aprendizaje reforzado inverso, a diferencia del RL tradicional, este busca aprender la función de recompensas en base a un comportamiento observado. (Inverse Reinforcement Learning).

Irracionalidad (B-Pref) Distintos comportamientos que tienen los humanos que dificultan la creación de una prueba que sea a la vez estándar y realista en el contexto de aprendizaje reforzado basado en preferencias.

MAML Familia de algoritmos de meta-aprendizaje que busca encontrar un modelo que generalice bien, que se adapte rápidamente a nuevas tareas o dominios con pocas muestras de datos de entrenamiento. (Model Agnostic Meta Learning).

MDP Procesos de Decisión de MArkov, son modelos matemáticos que representan la toma de decisiones secuenciales en entornos estocásticos. Es una combinación de las dinámicas del sistema que determinan a qué estado se llega desde cada par estado-acción y el modelo de recompensas que puede estar asociado tanto al estado como al par estado-acción. (Markov Decision Process).

Miopía (B-Pref) Una de las irracionalidades de SimTeacher. Se refiere a ser "cortos de vista" al mirar hacia atrás en el tiempo. Modela un problema de memoria, es probable que al tomar las decisiones los humanos tomemos en mayor consideración lo que ha pasado más recientemente que el inicio de las demostraciones.

MTL Aprendizaje Multi Tarea, técnica de aprendizaje de máquinas en que el agente aprende de varias tareas simultáneamente, permitiendo que el aprendizaje de una tarea beneficie al de otra. (Multi Task Learning).

Muestreo de Thompson Algoritmo de toma de decisiones basado en ideas bayesianas que utiliza la incertidumbre del modelo para muestrear de manera probabilística las acciones, permitiendo explorar y explotar de manera eficiente en entornos desconocidos o variables.

Método En programación orientada a objetos, una función asociada a una clase que define su comportamiento o acción.

Objeto En programación orientada a objetos, una instancia de una clase. Entidad que tiene un estado (atributos) y comportamiento (métodos) asociados.

PBRL Familia de algoritmos de aprendizaje reforzado en los que se busca aprender un comportamiento en base a las preferencias informadas por un profesor, sin programar una función de recompensas. (Preference Based Reinforcement Learning).

Política Estrategia o conjunto de reglas que rigen la acción que tomará el agente en cada estado posible.

PSRL Algoritmo RL que emplea un enfoque bayesiano para tomar decisiones óptimas en entornos estocásticos y desconocidos, permitiéndole explorar y explotar el entorno de manera efectiva. (Posterior Sampling for Reinforcement Learning).

Regresión Bayesiana La regresión bayesiana es un enfoque estadístico que actualiza las creencias sobre los parámetros de un modelo a medida que se observan datos, permitiendo modelar la incertidumbre de manera probabilística en las predicciones.

Reward Hacking Problema que suele surgir con funciones de recompensas para tareas complejas. El agente optimiza la función de recompensas, pero realiza un comportamiento distinto al esperado porque había un comportamiento no considerado que le entrega una mayor recompensa que el deseado.

RL Aprendizaje Reforzado, rama del aprendizaje de máquinas que busca optimizar una función de recompensas para completar una tarea de forma óptima. (Reinforcement Learning).

SimTeacher Algoritmo base de B-Pref, modela al profesor humano con irracionalidades.

Testbench Ambiente controlado para verificar el funcionamiento de un algoritmo.

Wrapper (python) Objeto que envuelve o encapsula otro objeto o función, proporcionando una interfaz más conveniente o modificada para interactuar con él.

Índice de contenidos

1. Introducción	1
1.1. Estado del arte	2
1.1.1. Aprendizaje Reforzado Basado en Preferencias (PBRL)	2
1.1.2. B-Pref: Un benchmark para PBRL	8
1.2. Alcances, contribuciones y estructura del documento	10
2. Descripción exhaustiva de Algoritmos y Herramientas utilizadas	11
2.1. Dueling Posterior Sampling for Preference-Based Reinforcement Learning	11
2.1.1. Modelo de dinámicas	11
2.1.2. Modelo de preferencias	12
2.1.3. El Algoritmo	13
2.2. B-Pref: Un benchmark para PBRL	16
3. Modificaciones e Implementación del Benchmark	21
3.1. Estructura del proyecto	21
3.2. Adaptación de B-Pref	23
3.2.1. RiverSwim	23
3.2.2. SimpleMountainCar	28
3.2.3. RandomMDP	30
3.3. Adaptación de DPS	30
4. Resultados	31
4.1. Estructura de las pruebas y formato de presentación de resultados . . .	32
4.2. SimTeacher: Oráculo	33
4.3. SimTeacher: Estocástico	34
4.3.1. Ruido lineal	34
4.3.2. Ruido logístico	35
4.4. SimTeacher: Error	36

4.5. SimTeacher: Salto	37
4.6. SimTeacher: Igual	38
4.7. SimTeacher: Miope	39
4.8. Análisis de Resultados	40
5. Conclusiones	42
5.1. Trabajo futuro.	43
Referencias	44
Anexos	46
Recompensas promedio	46
Empates y Consultas Saltadas	53

Índice de figuras

1.1. Ilustración del aprendizaje reforzado basado en preferencias. En vez de dar por hecho que el entorno entrega una recompensa (diseñada a mano), un profesor provee preferencias entre los comportamientos del agente y el agente utiliza esta retroalimentación en orden de aprender el comportamiento deseado. Extraído de [1].	4
1.2. Una visión general del método Few-Shot Preference Learning. A la izquierda se muestra la etapa de pre-entrenamiento y a la derecha la etapa de adaptación on-line con retroalimentación de un humano en lazo cerrado. Extraído de [2].	8
3.3. Estructura del repositorio original de DPS	22

3.4. El problema RiverSwim. La representación paramétrica de una política π especifica la acción (ir a la izquierda o derecha) seleccionada en cada estado. La representación conductual μ_π es $(1, \dots, 1, 0, \dots, 0)$, con el 1 más a la derecha en la posición $l(\pi)$, definido como el estado más a la izquierda asociado al movimiento izquierdo. Figura extraída y traducida de [3].	23
3.5. Ambiente Mountain Car, del que deriva SimpleMountainCarEnv. Extraído de la documentación de la librería Gym de Python [4].	28
4.6. $\text{SimTeacher}(\beta = \infty, \gamma = 1, \epsilon = 0, \delta_{\text{skip}} = 0, \delta_{\text{equal}} = 0)$	33
4.7. $\text{SimTeacher}(\beta = 1, \gamma = 1, \epsilon = 0, \delta_{\text{skip}} = 0, \delta_{\text{equal}} = 0)$. Con modelo lineal del ruido.	34
4.8. $\text{SimTeacher}(\beta = 1, \gamma = 1, \epsilon = 0, \delta_{\text{skip}} = 0, \delta_{\text{equal}} = 0)$. Con modelo logístico del ruido.	35
4.9. $\text{SimTeacher}(\beta = \infty, \gamma = 1, \epsilon = 0.1, \delta_{\text{skip}} = 0, \delta_{\text{equal}} = 0)$	36
4.10. $\text{SimTeacher}(\beta = \infty, \gamma = 1, \epsilon = 0, \delta_{\text{skip}} = 0.1, \delta_{\text{equal}} = 0)$	37
4.11. $\text{SimTeacher}(\beta = \infty, \gamma = 1, \epsilon = 0, \delta_{\text{skip}} = 0, \delta_{\text{equal}} = 0.1)$	38
4.12. $\text{SimTeacher}(\beta = \infty, \gamma = 0.9, \epsilon = 0, \delta_{\text{skip}} = 0, \delta_{\text{equal}} = 0)$	39
5.13. Recompensas promedio al enfrentarse al oráculo.	46
5.14. Recompensas promedio al enfrentarse al profesor estocástico con ruido lineal.	47
5.15. Recompensas promedio al enfrentarse al profesor estocástico con ruido logístico.	48
5.16. Recompensas promedio al enfrentarse al profesor que puede cometer errores.	49
5.17. Recompensas promedio al enfrentarse al profesor que puede saltar consultas.	50
5.18. Recompensas promedio al enfrentarse al profesor que suele informar un empate.	51
5.19. Recompensas promedio al enfrentarse al profesor con comportamiento “miope”.	52

5.20. Empates y consultas saltadas promedio para cada combinación profesor/algorithmo.	53
5.21. Empates y consultas saltadas en promedio para cada combinación profesor/algorithmo, expresado como porcentaje de las consultas totales realizadas por cada combinación.	54

1. Introducción

La presente investigación aborda un problema recurrente en los algoritmos de aprendizaje reforzado basado en preferencias (PBRL), identificado a través del benchmark B-Pref [1]. Se ha observado que cuando un profesor comete errores al asignar preferencias, estos algoritmos experimentan limitaciones significativas en su desempeño.

El objetivo principal de este proyecto es comparar varios algoritmos de la familia DPS (Dueling Posterior Sampling) de PBRL para determinar cuál es más resistente a las inconsistencias humanas, especialmente en términos de probabilidad de error y “miopía”. La motivación detrás de esta investigación radica que, a pesar de que los algoritmos PBRL existen desde 2011, por mucho tiempo no existió una evaluación estandarizada para estos. El desarrollo del testbench B-Pref en 2021 representó un avance significativo; sin embargo, su adopción aún es limitada en la comunidad científica.

En este estudio, se analizará el desempeño del algoritmo Dueling Posterior Sampling for Preference Based Reinforcement Learning, presentado en 2019 por E. Novoseller et al. [5], junto con los algoritmos EPMC [6] y PSRL [7], utilizados como referencias en la literatura.

La selección del algoritmo DPS se justifica por su capacidad para aprender las dinámicas del sistema además de las preferencias del profesor, lo que podría hacerlo más resistente a los errores humanos en comparación con los enfoques tradicionales de PBRL.

El propósito de este trabajo es sentar las bases para el desarrollo futuro de un algoritmo PBRL aún más robusto frente al error humano, que pueda combinar las características de aprendizaje del algoritmo DPS con la capacidad del metaaprendizaje para aprovechar la experiencia acumulada de tareas anteriores.

Para abordar este desafío, se ha decidido enfocar inicialmente la investigación en la evaluación de algoritmos, con la intención de integrar métodos de metaaprendizaje en futuras investigaciones, como el enfoque de Few Shot Preference Learning [2].

1.1. Estado del arte

En la presente subsección, se proporcionará un breve análisis del estado del arte en el ámbito del aprendizaje reforzado basado en preferencias (PBRL). Aquí, se explorará en detalle el concepto fundamental de PBRL, destacando la importancia de que el aprendizaje se realice en base a preferencias en vez de a una función de recompensas o una demostración.

Se presentarán varios algoritmos relevantes dentro del campo de PBRL, entre los cuales el más relevante para este trabajo es Dueling Posterior Sampling, que será expuesto más a fondo en la sección de desarrollo.

Además, entre los algoritmos comentados, se hará una breve mención al concepto de FewShot Preference Learning, una técnica emergente que busca abordar los desafíos de aprendizaje con conjuntos de datos limitados y la capacidad de generalización a nuevas situaciones. Se destacará su relevancia en el contexto de PBRL y su potencial para ampliar los límites del aprendizaje basado en preferencias en entornos dinámicos y complejos.

Finalmente, se hará un primer acercamiento al benchmark B-Pref, una herramienta fundamental para evaluar el desempeño de los algoritmos PBRL en entornos donde los profesores pueden cometer errores. Se examinará cómo B-Pref proporciona un marco estructurado para medir la resistencia de los algoritmos PBRL a las imperfecciones humanas, sentando así las bases para la mejora continua en este campo.

En conjunto, esta subsección proporcionará una visión general del estado actual de PBRL, presentando los principales conceptos, algoritmos y herramientas que constituyen el núcleo de la investigación en este campo en constante evolución.

1.1.1. Aprendizaje Reforzado Basado en Preferencias (PBRL)

El aprendizaje reforzado basado en preferencias es una rama específica del aprendizaje reforzado (RL) que facilita la cooperación humano-máquina. En este acercamiento, el humano le enseña al robot, o al agente, qué quiere que este haga al seleccionar entre

dos o más comportamientos demostrados por este, de modo que el agente debe descubrir por sí mismo la función de recompensa asociada y proponer nuevos comportamientos hasta llegar al que sea óptimo para la tarea en cuestión.

Puesto que con este enfoque el profesor no necesita ser capaz de demostrar perfectamente el modo ideal de realizar la tarea, como es el caso con aprendizaje reforzado inverso (IRL) o en el de aprendizaje por imitación, ni tampoco necesita ser capaz de construir una función de recompensas con precisión, que sería necesario para RL tradicional, el humano puede ser un usuario regular, alguien que sabe lo que quiere, pero no necesariamente un experto que conoce con exactitud la mejor forma de hacerlo. Es trabajo del robot, o del agente, discernir la manera apropiada de hacerlo, siempre basado en las preferencias del humano.

Originalmente los pasos básicos de los algoritmos de PBRL, como fueron introducidos por R. Arkour et al. [3] en 2011, son los siguientes:

- **Demostración.** El agente demuestra una nueva política al profesor humano.
- **Enseñanza.** El humano ordena jerárquicamente esta nueva política en comparación con las otras que el agente ha demostrado en el pasado.
- **Aprendizaje.** En base a la clasificación dada por el humano, el agente aprende un estimado del retorno, o la recompensa, de la política.
- **Auto-entrenamiento.** En base al estimado del retorno de la política, el agente genera nuevas políticas para ser demostradas.

Es importante notar que PBRL ha tenido un gran crecimiento desde 2011, por lo que a pesar que estos 4 pasos fueron la base original de estos algoritmos, actualmente no es esta la forma exacta en que esto se hace. Por ejemplo, la mayoría de los algoritmos actuales comparan entre 2 políticas cada vez, sin tener que organizar un archivo de políticas tras cada demostración. También actualmente es común que el agente aprenda una función de recompensas en base a la que generar una política en vez de usar algoritmos para aprender políticas directamente.

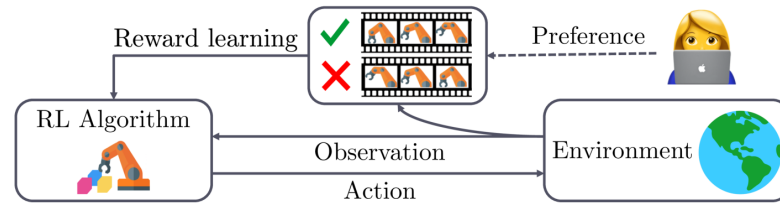


Figura 1.1: Ilustración del aprendizaje reforzado basado en preferencias. En vez de dar por hecho que el entorno entrega una recompensa (diseñada a mano), un profesor provee preferencias entre los comportamientos del agente y el agente utiliza esta retroalimentación en orden de aprender el comportamiento deseado. Extraído de [1].

Una rama importante de PBRL es Active PBRL, que contiene a aquellos algoritmos que ponen especial énfasis en la síntesis de las consultas que hará el agente. En general son enfoques que buscan lograr que cada pregunta que formule el agente le entregue la mayor cantidad de información posible, reduciendo así el número de consultas necesarias para llegar al comportamiento óptimo utilizando un modelo relativamente sencillo para la función de transferencias, siendo a veces incluso el de una combinación lineal. A continuación se discuten brevemente 3 enfoques distintos para la formulación activa de las consultas.

Expected Volume Removal

Este algoritmo, introducido en 2017 por D. Sadigh et al. [8], la función de recompensas que el agente trata de aprender se trata de una simple combinación lineal, una suma ponderada de "características" que diferencian a cada trayectoria.

Este enfoque busca sintetizar consultas para eliminar "volúmenes del espacio" donde se encuentran los posibles valores de los pesos ($\mathbf{w}$) de la función de recompensas. Esto significa que al inicio del proceso de aprendizaje, se considera que los pesos pueden tomar cualquier valor dentro de un espacio multidimensional, la bola unitaria, por ejemplo. Cada vez que se hace una consulta y se recibe una preferencia, se entiende que ciertas partes de este espacio son más probables de que contengan los valores óptimos de los pesos, mientras que otras partes son menos probables. Por lo tanto, el objetivo

del algoritmo EVR es seleccionar consultas que maximicen la eliminación de las partes menos probables del espacio de pesos, lo que ayuda a refinar la estimación de la función de recompensas y acelerar el proceso de aprendizaje del agente.

Matemáticamente, la forma en que se realiza esta "eliminación de volumen" es a través de una actualización Bayesiana:

$$p(\mathbf{w}|y_k) \propto p(\mathbf{w}) \cdot p(y_k|\mathbf{w}) \quad (1.1)$$

En esta ecuación, y_k es la preferencia del profesor en la consulta k y como se había mencionado antes, $\mathbf{w}$ son los pesos que caracterizan a la función de recompensas. Esta ecuación dice que la probabilidad de que los pesos $\mathbf{w}$ se encuentren en cierto lugar del espacio, dado que ya conocemos la preferencia del profesor (distribución a posteriori), es proporcional a la distribución de probabilidad anterior de los pesos, dónde se creía que estos estaban antes de consultar (distribución a priori) multiplicado por la probabilidad de obtener la preferencia que se recibió, dada la ubicación que se creía que tenían los pesos (verosimilitud), esta última probabilidad cuantifica qué tan bien el modelo explica la realidad con el conocimiento que se tenía hasta la consulta anterior. El concepto de actualización bayesiana es recurrente en otros algoritmos para actualizar el conocimiento que se tiene después de una consulta, en general es el método de escoger las consultas lo que cambia.

Dueling Posterior Sampling

En 2019, E. Novoseller et al. [5] introdujeron el algoritmo Dueling Posterior Sampling. Un aspecto diferenciador de este acercamiento es que no solo se busca aprender la función de recompensas que permite resolver la tarea de forma óptima, sino también las dinámicas del sistema, se busca aprender qué parte de la trayectoria, qué par estado-acción es la que hace que el profesor prefiera una opción por sobre la otra.

En este caso, a la hora de seleccionar la siguiente consulta lo que se hace es, una vez

que se han actualizado los modelos de recompensa y dinámicas que maneja el agente, se toma una muestra de ambos modelos y se busca la política óptima para este problema, no para el problema real, sino para lo que el agente considera que es el problema real en la k -ésima iteración. Luego que se hace una consulta mostrando las 2 trayectorias desarrolladas, se puede utilizar una regresión lineal Bayesiana o una regresión logística o de procesos gaussianos para actualizar la distribución de probabilidad que el agente muestreará en la siguiente iteración para generar dos nuevos procesos de decisión de Markov (MDP) de los que el agente buscará las políticas óptimas para generar las consultas en la siguiente iteración.

Si bien en este caso no hay un proceso para escoger “la consulta más informativa” cada vez, como con los modelos que utilizan Expected Volume Removal para determinar la siguiente consulta, en este caso existen garantías teóricas, teoremas, que indican que eventualmente se llegará a la política óptima. Esta garantía no está asegurada con el algoritmo de D. Sadigh et al. [8].

Cabe mencionar que E. Novoseller et al. [5] no presentan teoremas que garanticen que la política óptima será encontrada en tiempo finito, puede ser que solo sea alcanzada de manera asintótica. En 2020, Y. Xu et al. [9] presentaron un nuevo enfoque que puede asegurar que la respuesta será encontrada antes del infinito.

El algoritmo DPS será comentado con mayor detalle en la sección 2.1.

Regresión de Procesos Gaussianos para aprendizaje de recompensas

El tercer algoritmo comentado aquí, en 2020, E. Bıyık et al. [10] introdujeron un nuevo algoritmo de Active PBRL. En este caso, en vez de modelar la función de recompensas como una combinación lineal de las características, lo que se utiliza es un Proceso Gaussiano (GP), que permite modelar las recompensas sin hacer suposiciones sobre la clase de la función que las define. Parte de lo que la regresión de procesos Gaussianos busca lograr es descubrir a qué clase pertenece esta función que el agente desea aprender.

La principal ventaja de este método es que cada consulta otorga más información que

la que se obtendría si se da por hecho que la función de recompensas será lineal (1 bit cada vez), pues también otorga información del modelo que se está usando. Un problema grande de este modelo es que a la hora de generar una trayectoria para mostrar al profesor, no basta con tomar 1 muestra para obtener los pesos y desarrollar una trayectoria, es necesario tomar varias muestras del GP para poder desarrollar una trayectoria que pueda ser demostrada al usuario.

Otra ventaja que tiene por sobre [8] es que, por la forma en que está construido el modelo, la consulta trivial, mostrar 2 veces exactamente la misma trayectoria, es un mínimo global a la hora de seleccionar nuevos comportamientos a demostrar, por lo que nunca será mostrada. Con un algoritmo como el que utiliza Expected Volume Removal no existe una garantía teórica que diga que el agente no puede mostrar 2 veces el mismo comportamiento como una consulta.

Few-Shot Preference Learning

Como un comentario interesante del futuro de PBRL, introducido en 2023 [2], este algoritmo combina métodos de Meta-Aprendizaje, específicamente MAML [11], con PBRL, con el objetivo de lograr aprender una función de recompensa para tareas complejas en el menor número de consultas posibles.

La principal diferencia entre este método y otros introducidos anteriormente, es que la forma en que reduce la cantidad de consultas no es limitando la función de recompensas para necesitar menos información, sino poniendo más información a disposición del agente. Utilizando aprendizaje multi-tarea (MTL) es posible aprender una nueva función de recompensas basándose en conocimiento previo, sin tener que entrenar para una tarea compleja desde cero.

La forma en que logra esto es separando el aprendizaje en 2 etapas, una de pre-entrenamiento, off-line, en que el agente aprende a realizar una familia de tareas de manera óptima, para luego dar paso a una fase de adaptación on-line en que, usando las preferencias de un usuario humano, se consigue adaptar las funciones de recompensa

conocidas para desarrollar una nueva tarea en base a la retroalimentación del usuario en tiempo real, en modalidad de lazo cerrado, como se explica en el diagrama de la Figura 1.2, que fue extraído de [2].

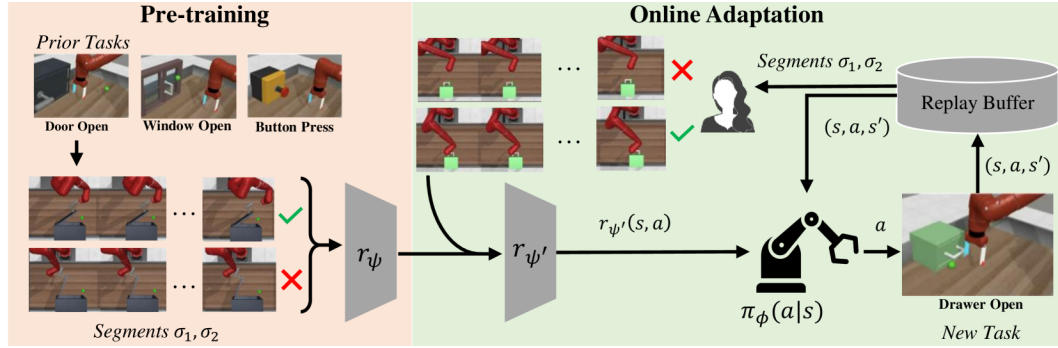


Figura 1.2: Una visión general del método Few-Shot Preference Learning. A la izquierda se muestra la etapa de pre-entrenamiento y a la derecha la etapa de adaptación on-line con retroalimentación de un humano en lazo cerrado. Extraído de [2].

1.1.2. B-Pref: Un benchmark para PBRL

Una de las grandes limitaciones que tenían los algoritmos de PBRL era la falta de un test estandarizado para comparar el desempeño de un algoritmo respecto al de otro, pues, hasta la introducción de B-Pref el año 2021 [1], no existía un benchmark diseñado específicamente para esta familia de algoritmos.

El funcionamiento de los algoritmos solía ser probado usando testbenchs diseñados para otro tipo de algoritmos de aprendizaje reforzado, ya sea usando un humano real como profesor o uno simulado que entregaría las preferencias.

El gran problema de estas soluciones es que una prueba con un humano real puede demostrar que el algoritmo funciona, pero no sirve como benchmark. Mientras que una prueba que usa un humano simulado ideal no es un caso realista para comprobar facilidad de operación por uno de carne y hueso.

B-Pref, introduce un profesor simulado que refleja de manera más realista las posibles “irracionalidades” humanas en la entrega de preferencias. Con el objetivo de modelar un comportamiento humano más realista que simplemente considerar el proceso de entrega de preferencias como ruidoso, el benchmark B-Pref [1] incluye una serie de

“irracionalidades” en su modelo de profesor (Algoritmo SimTeacher):

- **Modelo estocástico de preferencias.** La forma más básica de considerar un humano imperfecto entregando las preferencias es considerando su proceso para determinarlas como un proceso ruidoso, usando un modelo estocástico en vez de determinístico. Existe una *constante de racionalidad* que, dependiendo de su valor, hace que el proceso sea perfectamente determinístico o uniformemente aleatorio o algo entre estos dos extremos.

Es sobre este modelo que se aplican las siguientes irracionalidades del profesor.

- **Comportamiento Miope.** El profesor puede concentrarse más en la última opción o en el último paso de la opción que le fue presentada que en los anteriores. Para darle más importancia a lo último que el profesor vio, se asigna un *factor de descuento* de importancia basado en el tiempo.

Podría decirse más que es un problema de pérdida de memoria a corto plazo, pero así fue nombrado por los autores del benchmark [1].

- **Salto de preguntas.** Si ninguna de las trayectorias sugeridas tiene el comportamiento deseado, el profesor puede decidir ignorarlas completamente en vez de seleccionar una. Existe un *umbral de salto* que la recompensa subyacente a los comportamientos mostrados por el agente deben superar para que el profesor pueda escogerla como lo que él prefiere.

- **Igualmente Preferible.** Si ambas trayectorias propuestas son igualmente buenas, el profesor podría querer marcarlas ambas como igualmente preferibles. De manera similar que con el *umbral de salto*, existe un *umbral de igualdad* que debe ser superado por la diferencia entre las recompensas asociadas a ambos comportamientos para que el profesor pueda decantarse por una.

- **Cometer Errores.** Ya que los humanos solemos cometer errores, existe cierta probabilidad de inversión de la preferencia generada. El modelo subyacente escoge una trayectoria como la preferida, pero a la hora de informarla, el profesor

irracional puede decir que la otra es la predilecta.

Al ajustar cada uno de estos factores se puede obtener un profesor simulado más realista que uno ideal o uno en que simplemente se considera que puede haber ruido a la hora de escoger la preferencia. Además, como se consideran 4 irracionalidades distintas, asociados a parámetros diferentes entre sí, se puede probar qué tan susceptible es un algoritmo a cada una de estos comportamientos irracionales por separado.

La implementación específica del algoritmo *SimTeacher* está detallada como parte del diseño de la investigación.

1.2. Alcances, contribuciones y estructura del documento

Como se comentó previamente, el trabajo se centra en la evaluación de algoritmos de Dueling Posterior Sampling para aprendizaje reforzado basado en preferencias, con propósito de comprobar si estos algoritmos son robustos realmente ante equivocaciones humanas y determinar cuál puede ser más efectivo para investigaciones futuras.

El enfoque principal del proyecto es investigativo, con el propósito de someter algoritmos que no han sido probados previamente al benchmark B-Pref para hacer un aporte al conocimiento en el área de PBRL.

Se espera que el proyecto sea un primer paso para la identificación y creación de algoritmos que sean cada vez más robustos al error humano y que puedan ser utilizados por usuarios que no necesariamente sean expertos en sus áreas de trabajo, acercando el aprendizaje reforzado a un usuario más común.

Esta memoria se estructura de la siguiente manera: se presenta en los siguientes capítulos: el Capítulo 2 describe en mayor detalle los algoritmos y herramientas utilizadas; el Capítulo 3 detalla las modificaciones que fue necesario hacer para implementación del benchmark en el algoritmo; el Capítulo 4 analiza los resultados obtenidos y, finalmente, el Capítulo 5 concluye el trabajo y sugiere direcciones futuras para la investigación.

2. Descripción exhaustiva de Algoritmos y Herramientas utilizadas

2.1. Dueling Posterior Sampling for Preference-Based Reinforcement Learning

Como fue mencionado antes, este algoritmo, introducido en 2019 por E. Novoseller et al. [5], busca aprender tanto la función de recompensa detrás de las preferencias del profesor humano como las dinámicas del sistema que hacen que el usuario se decante por una u otra opción cada vez.

Luego, utilizando ambos modelos, dinámicas y recompensas, genera un proceso de decisión de Markov para resolver.

A continuación se muestran las formas que tiene el algoritmo de aprender ambos modelos, seguido de una explicación del algoritmo en sí.

2.1.1. Modelo de dinámicas

Para aprender el modelo de dinámicas, “la forma en que funciona el mundo”, este algoritmo utiliza una distribución de Dirichlet para modelar las transiciones entre estados y acciones (2.2)

$$f(\mathbf{x}, \alpha) = \frac{\prod_{k=1}^K \Gamma(\alpha_k)}{\Gamma\left(\sum_{k=1}^K \alpha_k\right)} \prod_{k=1}^K x_k^{\alpha_k-1} \quad (2.2)$$

Se utiliza esta función de densidad de probabilidad para modelar la probabilidad de pasar de cada par estado-acción a un nuevo estado. En la ecuación (2.2), $\mathbf{x}$ representa una combinación estado-acción única, mientras que x_k representa a la transición estado-acción-nuevo_estado_k, para cada estado k y el parámetro α_k es un parámetro de la distribución de Dirichlet, que entre más alto sea su valor, más probable es pasar del par estado-acción al nuevo_estado_k.

Al inicio el agente supone que desde cada par estado-acción es igualmente probable

llegar a cualquier otro estado, pero cada vez que el agente demuestra una trayectoria va tomando nota de los estados por los que va pasando y cómo cada estado-acción lo lleva a cada nuevo estado posible. Así, conforme va pasando por nuevos estados va aumentando el valor del α_k asociado a la transición, refinando su comprensión del sistema con cada paso que da.

2.1.2. Modelo de preferencias

Para aprender el modelo de preferencias, “la forma en que funciona el profesor”, se utiliza el muestreo de Thompson. Tras realizar una actualización Bayesiana (1.1), se toma una muestra la distribución a posteriori de los parámetros del modelo para crear un nuevo MDP y encontrar una nueva política.

$$\underbrace{p(\mathbf{w}_k|y_k)}_{\text{distribución a posteriori}} = \frac{p(y_k|\mathbf{w}_k)p(\mathbf{w}_k)}{\int p(y_k|\mathbf{w}_k)p(\mathbf{w}_k)d\mathbf{w}_k} \quad (2.3)$$

$$\propto \underbrace{p(y_k|\mathbf{w}_k)}_{\text{función de verosimilitud}} \times \underbrace{p(\mathbf{w}_k)}_{\text{distribución a priori}}$$

Este concepto del muestreo a posteriori es utilizado en DPS tanto para inferir las recompensas como para inferir qué parte de la trayectoria demostrada, qué pares de estado-acción son los responsables de las preferencias del profesor (asignación de crédito).

Finalmente, cabe mencionar que no importa la versión de DPS que se esté utilizando, el modelo de las recompensas siempre es descrito como distribuciones normales independientes para cada par estado-acción, caracterizadas por su media y los vectores y valores propios de su covarianza.

Sin embargo, la forma en que se van actualizando estos valores puede variar. En una versión se realiza una regresión lineal Bayesiana, mientras que en otra una regresión de

procesos gaussianos y en otra una regresión logística. Todos estos métodos se utilizan para encontrar la distribución a posteriori. En este trabajo una de las cosas que se quiere evaluar es cuál de estos métodos de asignación de crédito es el más robusto a errores del humano.

2.1.3. El Algoritmo

El algoritmo DPS comienza con la inicialización de un historial vacío y de distribuciones previas para el modelo de transición de estados (dinámicas) y el modelo de utilidades (equivalente a las recompensas en este contexto). Luego, el algoritmo itera entre 3 pasos:

1. **Muestrear 2 políticas** π_{kA} y π_{kB} con el Algoritmo 2 ADVANCE. Este algoritmo recibe el modelo de las dinámicas (cómo cree el agente que funciona el mundo) y el de las utilidades o recompensas (cómo cree el agente que funciona el profesor) hasta la k -ésima consulta como entradas, toma 2 muestras de estos modelos y obtiene una política óptima para ambos MDPs generados por estas combinaciones de dinámicas y recompensas.
2. **Poner en marcha las políticas** π_{kA} y π_{kB} para obtener las trayectorias ξ_{kA} y ξ_{kB} . Luego se recibe una preferencia y_k entre ambas trayectorias. Estas dos trayectorias deben ser guardadas junto con la preferencia en el historial. Se guardan las diferencias entre los pares estado/acción visitados por cada trayectoria junto con la preferencia como un ± 0.5 dependiendo de cuál trayectoria fue preferida.
3. Para terminar la iteración, se utilizan los datos del historial para actualizar las distribuciones (a posteriori) con el Algoritmo 3 FEEDBACK. Dependiendo del algoritmo DPS específico, se pueden utilizar distintos métodos matemáticos para realizar esta actualización.

A continuación se muestra el Algoritmo DPS como un pseudo-código:

Algorithm 1 Dueling Posterior Sampling (DPS). Extraído y traducido desde [5]

```

1:  $\mathcal{H}_0 = \emptyset$  Inicializar historial
2: Inicializar distribución a priori para  $f_p$  // Inicializar modelo de transición de estados
3: Inicializar distribución a priori para  $f_r$  // Inicializar modelo de utilidad
4: for  $t = 1, 2, \dots$  do
5:    $\pi_{k1} \leftarrow \text{ADVANCE}(f_p, f_r)$ 
6:    $\pi_{t2} \leftarrow \text{ADVANCE}(f_p, f_r)$ 
7:   Muestrear trayectorias  $\xi_{k1}$  y  $\xi_{k2}$  desde  $\pi_{k1}$  y  $\pi_{k2}$ 
8:   Recibir retroalimentación  $y_t = \mathbb{I}_{[\xi_{k1} \succ \xi_{k2}]} - \frac{1}{2}$ 
9:    $\mathcal{H}_k = \mathcal{H}_{k-1} \cup (\xi_{k1}, \xi_{k2}, y_k)$ 
10:   $f_p, f_r = \text{FEEDBACK}(\mathcal{H}_t, f_p, f_r)$ 
11: end for

```

En las líneas 5 y 6 del Algoritmo 1 se utiliza el algoritmo **ADVANCE**, mostrado a continuación, para obtener 2 políticas en base a la información conocida hasta la iteración actual tanto sobre las dinámicas del sistema como de la función de recompensas.

Algorithm 2 ADVANCE: Muestrea una política desde los modelos de dinámicas y utilidad. Extraído y traducido desde [5]

Require: f_p, f_r

```

1: Muestrear  $\tilde{p} \sim f_p(\cdot)$  // Muestrea los parámetros de dinámicas de transición MDP desde distribución a posteriori
2: Muestrear  $\tilde{r} \sim f_r(\cdot)$  // Muestrea utilidades desde distribución a posteriori
3: Computar  $\pi = \arg \max_{\pi} V(\tilde{p}, \tilde{r})$  // La optimización de la función de valor entrega la política óptima del MDP muestreado  $(\tilde{p}, \tilde{r})$ .
4: return  $\pi$ 

```

El Algoritmo 2 **ADVANCE** toma muestras de las distribuciones a posteriori de los modelos de dinámicas y utilidad y luego utiliza ambas muestras para seleccionar una política. Estas muestras forman un MDP para el que se puede encontrar una política óptima al

iterar la función de valor.

Conforme van avanzando las consultas y se tienen modelos f_p y f_r más precisos, las políticas se van pareciendo más y más entre sí, hay una mayor explotación, mientras que al inicio del proceso de aprendizaje, como los modelos tienen una gran variabilidad, es muy probable que las políticas obtenidas por ADVANCE sean diferentes, lo que implica una mayor exploración.

Finalmente, en la línea 10 del Algoritmo 1 se utiliza el algoritmo FEEDBACK para actualizar los modelos que se quieren aprender.

Algorithm 3 FEEDBACK: Actualiza modelos de dinámicas y utilidad basado en nueva retroalimentación del usuario. Extraído y traducido desde [5]

Require: Historial $\mathcal{H}$, f_p , f_r

- 1: Aplicar actualización Bayesiana a f_p , dado $\mathcal{H}$ // Actualizar el modelo de las dinámicas para el historial dado
 - 2: Aplicar actualización Bayesiana a f_r , dado $\mathcal{H}$ // Actualizar el modelo de utilidades para preferencias dadas
 - 3: **return** f_p , f_r
-

El Algoritmo 3 FEEDBACK actualiza las distribuciones a posteriori de los modelos, esto es lo que hace el muestreo de Thompson. La distribución a priori, original, de las dinámicas es modelada como una distribución Dirichlet (generalización multivariable de la distribución β) y las recompensas pueden ser modeladas como una regresión Bayesiana lineal, una regresión logística Bayesiana, una regresión de Procesos Gaussianos o con un sistema de asignación de crédito de preferencia de procesos Gaussianos.

El algoritmo DPS sobresale por su capacidad de abordar los desafíos del aprendizaje basado en preferencias humanas al comprender las dinámicas del sistema a través de la interacción con el entorno, independientemente de las preferencias del profesor.

Mediante un proceso iterativo, adapta sus políticas de acción integrando la información del entorno y las preferencias del profesor, lo que le permite encontrar políticas óptimas para su MDP interno.

Como este algoritmo no busca aprender lo máximo con cada consulta, si no resolver de manera óptima cómo cree que funciona el mundo y su tarea en cada iteración, no debería quedarse atascado tratando si hubo un error en la asignación de preferencias, ya que en cualquier caso, para la próxima iteración el MDP a resolver será diferente. Su habilidad para aprender de las consultas pasadas junto con el respaldo de garantías teóricas de que alcanzará la política óptima eventualmente, lo posiciona como un candidato prometedor para entornos dinámicos y ruidosos, donde la adaptabilidad y la robustez son esenciales.

2.2. B-Pref: Un benchmark para PBRL

- **Modelo estocástico de preferencias.** La forma más básica de considerar un humano imperfecto entregando las preferencias es considerando su proceso para determinarlas como un proceso ruidoso, usando un modelo estocástico en vez de determinístico. Existe una *constante de racionalidad* que, dependiendo de su valor, hace que el proceso sea perfectamente determinístico o uniformemente aleatorio o algo entre estos dos extremos.

Es sobre este modelo que se aplican las siguientes irracionalidades del profesor.

- **Comportamiento Miope.** El profesor puede concentrarse más en la última opción o en el último paso de la opción que le fue presentada que en los anteriores. Para darle más importancia a lo último que el profesor vio, se asigna un *factor de descuento* de importancia basado en el tiempo.

Podría decirse más que es un problema de pérdida de memoria a corto plazo, pero así fue nombrado por los autores del benchmark [1].

- **Salto de preguntas.** Si ninguna de las trayectorias sugeridas tiene el comportamiento deseado, el profesor puede decidir ignorarlas completamente en vez de

seleccionar una. Existe un *umbral de salto* que la recompensa subyacente a los comportamientos mostrados por el agente deben superar para que el profesor pueda escogerla como lo que él prefiere.

- **Igualmente Preferible.** Si ambas trayectorias propuestas son igualmente buenas, el profesor podría querer marcarlas ambas como igualmente preferibles. De manera similar que con el *umbral de salto*, existe un *umbral de igualdad* que debe ser superado por la diferencia entre las recompensas asociadas a ambos comportamientos para que el profesor pueda decantarse por una.
- **Cometer Errores.** Ya que los humanos solemos cometer errores, existe cierta probabilidad de inversión de la preferencia generada. El modelo subyacente escoge una trayectoria como la preferida, pero a la hora de informarla, el profesor irracional puede decir que la otra es la predilecta.

Al ajustar cada uno de estos factores se puede obtener un profesor simulado más realista que uno ideal o uno en que simplemente se considera que puede haber ruido a la hora de escoger la preferencia. Además, como se consideran 4 irracionalidades distintas, asociados a parámetros diferentes entre sí, se puede probar qué tan susceptible es un algoritmo a cada una de estos comportamientos irracionales por separado.

Si se quisiera un profesor perfectamente racional, el modelo que asigna preferencias sería el siguiente proceso determinístico:

$$y = \begin{cases} (1, 0) & \text{Si } \sum_{t=1}^H r(\mathbf{s}_t^0, \mathbf{a}_t^0) > \sum_{t=1}^H r(\mathbf{s}_t^1, \mathbf{a}_t^1) \\ (0, 1) & \text{En caso contrario} \end{cases} \quad (2.4)$$

Donde si una trayectoria, definida por una sucesión de pares estado/acción, tiene una recompensa acumulada mayor que la otra, esta será preferida y si no, la otra lo será.

Sin embargo, lo que se quiere usar no es un modelo determinístico, si no uno estocástico, que además tome en consideración las irracionalidades listadas. Para construir el modelo (2.7) utilizado por el Algoritmo 4 para asignar las preferencias, primero se debe partir

por el modelo de Bradley-Terry [12] para comparaciones entre pares (2.5). Este es un modelo común a la hora de realizar comparaciones ruidosas, se aplica la función logística a la diferencia de las recompensas totales de ambas opciones (2.6) y se toma este resultado como la probabilidad de escoger una sobre la otra. En adelante, $\sigma^i \succ \sigma^j$ significa que el segmento, o trayectoria, σ^i fue preferido por sobre el segmento σ^j .

$$P[\sigma^i \succ \sigma^j] = \frac{\exp(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i))}{\exp(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i)) + \exp(\sum_t r(\mathbf{s}_t^j, \mathbf{a}_t^j))} \quad (2.5)$$

$$P[\sigma^i \succ \sigma^j] = \frac{1}{1 + e^{-(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i) - \sum_t r(\mathbf{s}_t^j, \mathbf{a}_t^j))}} \quad (2.6)$$

El modelo por el que se asignan las preferencias del profesor simulado por B-Pref (Sim-Teacher) es el siguiente:

$$P[\sigma^i \succ \sigma^j; \beta, \gamma] = \frac{\exp\left(\beta \sum_{t=1}^H \gamma^{H-t} r(\mathbf{s}_t^i, \mathbf{a}_t^i)\right)}{\exp\left(\beta \sum_{t=1}^H \gamma^{H-t} r(\mathbf{s}_t^i, \mathbf{a}_t^i)\right) + \exp\left(\beta \sum_{t=1}^H \gamma^{H-t} r(\mathbf{s}_t^j, \mathbf{a}_t^j)\right)} \quad (2.7)$$

Donde β es la constante de racionalidad. Si $\beta \rightarrow \infty$ el modelo (2.7) se convierte efectivamente en (2.4) y si $\beta = 0$, $P[\sigma^i \succ \sigma^j; \beta = 0, \gamma] = 0.5$ para todo par de segmentos y para todo valor de γ .

γ es el llamado “factor de miopía”, que define qué tanta menos importancia se le asignarán a los pasos de la trayectoria demostrada conforme estos se alejan del presente hacia el pasado.

Los otros factores que modelan las irracionalidades mencionadas anteriormente se encuentran en el Algoritmo 4 Sim Teacher.

Algorithm 4 SimTeacher: Simulación de profesores humanos. Extraído y traducido desde [1]

Require: Factor de descuento γ , constante de racionalidad β , probabilidad de error ϵ

Require: Umbral de salto δ_{salto} , umbral de igualdad δ_{igual}

Require: Par de segmentos σ^0, σ^1

```

1: if  $\max_{i \in \{0,1\}} \sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i) < \delta_{salto}$  then // CONSULTA SALTADA
2:    $y \leftarrow \emptyset$ 
3: else if  $|\sum_t r(\mathbf{s}_t^1, \mathbf{a}_t^1) - \sum_t r(\mathbf{s}_t^0, \mathbf{a}_t^0)| < \delta_{igual}$  then // IGUALMENTE PREFERIBLE
4:    $y \leftarrow (0.5, 0.5)$ 
5: else if  $\sigma^0 \succ \sigma^1 \sim P[\sigma^0 \succ \sigma^1; \beta, \gamma]$  then // OBTENCIÓN DE PREFERENCIAS
   DESDE (2.7)
6:    $y \leftarrow (1, 0)$  con probabilidad  $1 - \epsilon$ 
7:    $y \leftarrow (0, 1)$  en otro caso // COMETER UN ERROR
8: else
9:    $y \leftarrow (0, 1)$  con probabilidad  $1 - \epsilon$ 
10:   $y \leftarrow (1, 0)$  en otro caso // COMETER UN ERROR
11: end if
12: return  $y$ 

```

En las líneas 1 y 3 del Algoritmo 4 podemos ver la aplicación de los umbrales de salto y de igualdad, respectivamente. El caso ideal tiene $\delta_{salto} = \delta_{igual} = 0$, entre mayores sean estos umbrales, el profesor será más “indeciso”, llevándolo más lejos del caso ideal.

En las líneas 7 y 10 del Algoritmo se ve la capacidad de cometer errores del profesor simulado, comete errores con probabilidad $\epsilon \in [0, 1]$, que se traduce en entregar la preferencia y con el valor contrario al que entrega $P[\sigma^i \succ \sigma^j; \beta, y]$. Claramente, con $\epsilon = 0$, el profesor nunca se equivoca y con $\epsilon = 1$, siempre cambia los resultados antes de entregarlos.

La forma en que B-Pref evalúa el desempeño de un algoritmo es al enfrentarlo a 5 profesores con distintas irracionalidades y el oráculo (caso ideal), para comparar sus

resultados con el retorno promedio de un algoritmo de RL tradicional que tiene conocimiento de la recompensa verdadera $r(\mathbf{s}_t, \mathbf{a}_t)$ para cada paso t .

Los profesores a los que deben enfrentarse los algoritmos son los siguientes:

- Oráculo: $\text{SimTeacher}(\beta \rightarrow \infty, \gamma = 1, \epsilon = 0, \delta_{\text{salto}} = 0, \delta_{\text{igual}} = 0)$
- Estocástico: $\text{SimTeacher}(\beta = 1, \gamma = 1, \epsilon = 0, \delta_{\text{salto}} = 0, \delta_{\text{igual}} = 0)$
- Error: $\text{SimTeacher}(\beta \rightarrow \infty, \gamma = 1, \epsilon = 0.1, \delta_{\text{salto}} = 0, \delta_{\text{igual}} = 0)$
- Salto: $\text{SimTeacher}(\beta \rightarrow \infty, \gamma = 1, \epsilon = 0, \delta_{\text{salto}} > 0, \delta_{\text{igual}} = 0)$
- Igual: $\text{SimTeacher}(\beta \rightarrow \infty, \gamma = 1, \epsilon = 0, \delta_{\text{salto}} = 0, \delta_{\text{igual}} > 0)$
- Miope: $\text{SimTeacher}(\beta \rightarrow \infty, \gamma = 0.9, \epsilon = 0, \delta_{\text{salto}} = 0, \delta_{\text{igual}} = 0)$

En K. Lee et al. [1], el benchmark B-Pref es aplicado en 2 contextos distintos:

- **Comparar el desempeño de 2 algoritmos entre sí.** Específicamente, B-Pref es aplicado para comparar PEBBLE de K. Lee et al. [13] con PrefPPO, el algoritmo de PBRL profundo de P. Christiano et al. [14] que en este caso es implementado utilizando Optimización de Política Proximal (PPO) de J. Schulman et al. [15].
- **Analizar el impacto de decisiones de diseño en algoritmos PBRL.** Específicamente, se evalúa el impacto del sistema para elegir la siguiente consulta y el efecto de diferentes cambios en el número de consultas por sesión de retroalimentación con el profesor.

3. Modificaciones e Implementación del Benchmark

Luego de repetidos intentos fallidos de implementar el benchmark B-Pref oficial, desde el repositorio en que sus autores lo subieron, se optó por implementar una prueba inspirada en el Algoritmo 4 **SimTeacher** que es el corazón del benchmark.

El problema de haber implementado una versión hecha a mano es que no se pueden comparar los resultados de esta prueba con los resultados de otros algoritmos como PEBBLE o PPO que han sido testeados ya con el benchmark original. Sin embargo, los resultados de estas pruebas siguen siendo relevantes para el segundo contexto en que B-Pref suele ser aplicado, esto es, en el análisis del impacto de decisiones de diseño en algoritmos PBRL. Se pueden comparar los distintos regímenes de asignación de crédito dentro de los algoritmos DPS.

Además, como se está aplicando la misma prueba a EPMC, no hay ninguna objeción a comparar sus resultados con los de DPS.

3.1. Estructura del proyecto

La forma en que el código de DPS está estructurada como se muestra en la Figura 3.3. En una carpeta están los ambientes, que son los escenarios en que se va a probar el algoritmo. En este caso, RiverSwim, SimpleMountainCar y RandomMDP. Estos escenarios son los que tienen la función de recompensa y las dinámicas del sistema reales. Tienen la lógica para pasar de cada par estado-acción al siguiente estado que corresponda y son los encargados de asignar las preferencias sin que el algoritmo de aprendizaje conozca ni las recompensas reales a cada paso que da, ni las dinámicas reales del sistema. Estos ambientes deberán ser modificados para tener una versión que implemente B-Pref.

En otra carpeta se encuentran los algoritmos de aprendizaje, estos reciben un ambiente al que le pueden preguntar su número de estados y acciones y al que le proponen trayectorias. Conforme van desarrollando las trayectorias, van aprendiendo del ambiente las dinámicas del sistema y tras cada consulta reciben una preferencia de parte del mismo, preferencia que usarán en conjunto con lo que van aprendiendo de las dinámicas

para generar un MDP cada vez más parecido al real, escondido dentro del ambiente e, idealmente, llegar a la política óptima del problema.

Finalmente, fuera de ambas carpetas, se encuentran las pruebas realizadas por cada algoritmo para cada escenario. Por ejemplo, `DPS_linear_in_RiverSwim.py` contiene código para ejecutar la versión del algoritmo DPS que hace una regresión lineal Bayesiana para la asignación de crédito en el ambiente RiverSwim. Este código corre el algoritmo 100 veces en varias versiones del ambiente con distintos niveles y tipos de ruido. El archivo `DPS_linear_in_RiverSwim.py` además genera archivos `.mat` que contienen las recompensas en cada paso de cada iteración y que son las utilizadas para la creación de las figuras en la sección de resultados.

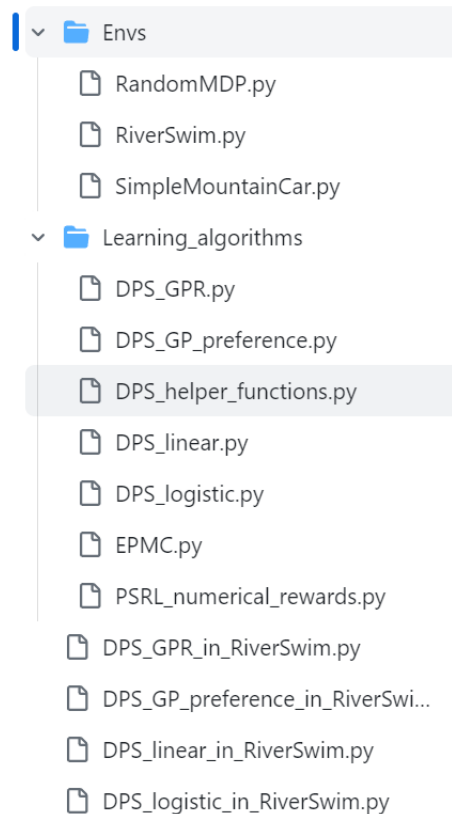


Figura 3.3: Estructura del repositorio original de DPS

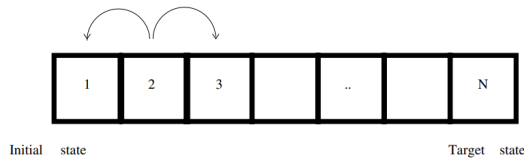
3.2. Adaptación de B-Pref

Ya que son los ambientes los que determinan la preferencia, es en ellos donde se debe implementar B-Pref. A continuación, una breve explicación del funcionamiento de estos, junto con los cambios específicos que tuvo que hacerse a cada uno.

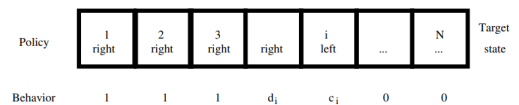
3.2.1. RiverSwim

En primer lugar, RiverSwim, simula a un nadador que quiere ir en contra de la corriente. El río se divide en N estados y las dos acciones posibles son *avanzar* o *retroceder*. Si se trata de avanzar contra la corriente hay cierta probabilidad de avanzar, otra de quedarse en el lugar y también una probabilidad de retroceder, mientras que si se trata de retroceder, el agente siempre retrocederá.

La función de recompensa real de la tarea da una pequeña recompensa por quedarse al inicio del río y otra más grande por quedarse al final de este. No da recompensas en ningún otro punto del río ni tampoco la primera vez que el agente llega al final. Debe lograr quedarse en el final del río para recibir una recompensa.



(a) Espacio de Acciones y Estados



(b) Representación paramétrica y de conducta

Figura 3.4: El problema RiverSwim. La representación paramétrica de una política π especifica la acción (ir a la izquierda o derecha) seleccionada en cada estado. La representación conductual μ_π es $(1, \dots, 1, 0, \dots, 0)$, con el 1 más a la derecha en la posición $l(\pi)$, definido como el estado más a la izquierda asociado al movimiento izquierdo. Figura extraída y traducida de [3].

El archivo presente en el repositorio asociado a DPS [5] ya contiene una implementación de este conocido ambiente para algoritmos RL y un “wrapper” (o envoltorio) que convierte a RiverSwimEnv en RiverSwimPreferenceEnv. Este wrapper lo que hace es incluir el método `get_trajectory_preference()`, así como también modificar el método `step()` para que no entregue información de las recompensas en cada paso al

algoritmo de aprendizaje.

Cabe mencionar que los ambientes presentes en el repositorio original consideran 3 modelos de asignación de preferencias, uno determinístico en que la comparación es como en (2.4) y un empate entrega **preference** = 0.5, uno con ruido logístico, en que la preferencia se asigna con la misma distribución de probabilidad que la presentada en (2.5), donde además un parámetro de ruido funciona de la misma forma que el factor β de B-Pref en (2.7) y uno con ruido lineal, en que la preferencia es asignada con el siguiente modelo estocástico:

$$p = \beta \left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i) - \sum_t r(\mathbf{s}_t^j, \mathbf{a}_t^j) \right) + 0.5 \quad (3.8)$$

$$P[\sigma^i \succ \sigma^j] = \begin{cases} 0 & \text{si } p < 0 \\ 1 & \text{si } p > 1 \\ p & \text{en otro caso} \end{cases} \quad (3.9)$$

En este régimen, la probabilidad de que σ^i sea preferido por sobre σ^j queda definido por la diferencia entre las recompensas reales multiplicado por un factor de ruido +0.5. Para poder usar esto como una probabilidad se hace que cualquier valor por sobre 1 quede en 1 y cualquiera bajo 0 se quede en 0.

Aquí, al igual que en B-Pref, entre menor sea el valor de β , más ruidosa es la preferencia, porque más cercano estará el valor p de 0.5, a pesar que la diferencia en recompensas pueda no ser menor.

Se creó un segundo envoltorio que convirtiera RiverSwimPreferenceEnv en RiverSwimPreferenceEnvBPref. Como se muestra en el código 3.1, se tuvo que modificar el constructor de la clase para poder recibir los parámetros de **SimTeacher** de B-Pref.

Además se reemplazó el parámetro de ruido de RiverSwimPreferenceEnv por el valor de β . Si bien B-Pref solo considera el modelo logístico de ruido, como se verá con más

detalle en el código 3.3, se decidió dar soporte a otros modelos de ruido.

Código 3.1 : RiverSwimEnvBPref.__init__()

```
1 def __init__(self, user_noise_type, num_states = 6, beta=np.inf, gamma=1,
    eps=0, d_skip=0,d_equal=0):

    self.beta = beta
    self.gamma = gamma
6    self.eps = eps
    self.d_skip = d_skip
    self.d_equal = d_equal

    user_noise_model= [beta, user_noise_type]
11    if beta == np.inf:
        user_noise_model[1]=0
    super().__init__(user_noise_model,num_states)
```

Otro punto en que fue necesario hacer una modificación fue en el método que calcula el retorno de las trayectorias `get_trajectory_return()`, para que considerara el valor del factor de miopía γ . Se dejó por defecto `gamma = 1` pues en `SimTeacher`, la comparación inicial que se hace para determinar si se ha tenido un empate o si se debe saltar se hace con las recompensas reales de las trayectorias, sin considerar el factor de miopía.

Código 3.2 : RiverSwimEnvBPref.get_trajectory_reward()

```
def get_trajectory_return(self, tr,gamma=1):
2
    states = tr[0]
    actions = tr[1]

    if not len(states) == len(actions) + 1:
7        print('Invalid input given to get_trajectory_return.')

    total_return = 0
    H = len(actions)
    for i in range(H):
12        total_return += (gamma**(H-1-i)) * self.get_step_reward(states[i], actions[i],
            states[i + 1])
    return total_return
```

Finalmente, el cambio más extenso e importante para la implementación de B-Pref en el ambiente RiverSwim es aquel realizado al método `get_trajectory_preference()`. En primer lugar, ya que de todos modos no se está usando la versión oficial de B-Pref, se decidió mantener la opción de escoger el modelo de ruido, de modo que el profesor estocástico puede tener ruido logístico (`noise_type = 1`) o lineal (`noise_type = 2`), además de una versión sin ruido cuando `noise_type = 0` o $\beta = \text{np.inf}$.

Ya que los profesores a los que se enfrentarán los algoritmos en el benchmark consideran una irracionalidad a la vez, el haber considerado los tipos de ruido que cubrían los escenarios a los que fue enfrentado DPS simplemente añade variables al profesor estocástico, todos los otros profesores consideran $\beta \rightarrow \infty$.

Se realizaron varias modificaciones a este método. En primer lugar, ahora existe el caso en que el profesor decide saltar la consulta, si se detecta que la mayor de las recompensas de las trayectorias (con $\gamma = 1$) es menor al umbral de salto `d_skip`, la preferencia entregada será `np.nan`. Es necesario que el algoritmo de aprendizaje sepa qué hacer con esto.

Luego, si la diferencia entre las recompensas (todavía sin efecto del factor γ) no es mayor que `d_equal`, se declara un empate, `preference = 0.5`. Antes solo se podía entregar preferencia 0.5 si se escogía el régimen sin ruido. Ahora se puede tener este resultado sin importar el profesor, el tipo de ruido ni su intensidad escogidos.

Luego, según el tipo de ruido se realiza la asignación de preferencias tal como era en `RiverSwimPreferenceEnv`, pero reemplazando la recompensa de las trayectorias por la recompensa considerando el factor de descuento γ .

Por último, se cambia el valor de la preferencia con probabilidad `eps`. Si la preferencia era 0.5 o `np.nan` esta línea no cambia nada.

Código 3.3 : RiverSwimEnvBPref.get_trajectory_preference()

```
def get_trajectory_preference(self, tr1, tr2):  
2  
  
    noise_param, noise_type = self.user_noise_model  
    assert (noise_type in [0,1,2]), "noise_type %i invalid" % noise_type  
  
    if noise_param != self.beta:  
7        print(f'Using beta = {self.beta} as noise_param')  
  
    trajectories = [tr1, tr2]  
    num_traj = len(trajectories)  
    returns = np.empty(num_traj)  
12    for i in range(num_traj):  
        returns[i] = self.get_trajectory_return(trajectories[i])  
  
    if max(returns) < self.d_skip:  
        preference = np.nan  
17    elif (returns[0] - returns[1]) < self.d_equal:  
        preference = 0.5  
    else:  
        for i in range(num_traj):  
            returns_gamma = np.empty(num_traj)  
22            returns_gamma[i] = self.get_trajectory_return(trajectories[i], gamma=self.  
gamma)  
            if noise_type == 0 or self.beta == np.inf: # Deterministic preference:  
                if returns_gamma[0] > returns_gamma[1]:  
                    preference = 0  
                else:  
27                    preference = 1  
            elif noise_type == 1: # Logistic noise model  
                prob = 1 / (1 + np.exp(-self.beta * (returns_gamma[1] - returns_gamma[0])))  
        )  
        preference = np.random.choice([0, 1], p = [1 - prob, prob])  
            elif noise_type == 2: # Linear noise model  
32                prob = self.beta * (returns_gamma[1] - returns_gamma[0]) + 0.5  
                prob = np.clip(prob, 0, 1)  
                preference = np.random.choice([0, 1], p = [1 - prob, prob])  
  
        preference = np.random.choice([preference, 1 - preference], p=[1 - self.eps,  
self.eps])  
37    return preference
```

Los códigos se encuentran mejor documentados en el repositorio que acompaña a este trabajo.

3.2.2. SimpleMountainCar

Este ambiente simula a un auto que se encuentra en algún punto de un valle entre 2 montañas y quiere llegar a la que se encuentra frente a él. En este caso los estados son bi-dimensionales, constando de la posición en el eje X del auto y de la velocidad a la que va. Además, hay 3 acciones posibles, acelerar hacia adelante, acelerar hacia atrás o no acelerar.

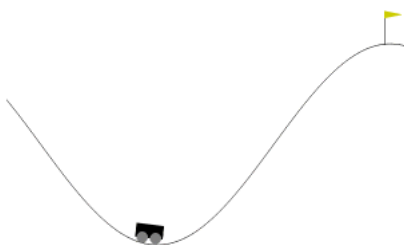


Figura 3.5: Ambiente Mountain Car, del que deriva SimpleMountainCarEnv. Extraído de la documentación de la librería Gym de Python [4].

Se implementaron modificaciones similares a las de RiverSwim, también respetando los distintos modelos de ruido que consideraban las pruebas realizadas a DPS.

Sin embargo, la principal diferencia en este caso, es la forma en que está definida la función de recompensas en el ambiente SimpleMountainCar original. Mientras que en RiverSwim el agente sigue realizando acciones hasta que llega el horizonte temporal y tiene que terminarse el episodio, los episodios de SimpleMountainCar no son todos del mismo largo. En este caso los episodios terminan tan pronto como el auto llegue a la bandera (ver Figura 3.5) y luego la recompensa asociada a esta trayectoria es igual al negativo del largo de la misma. Puede interpretarse como que a cada paso que da, el carro recibe una recompensa de -1 , pero no se va evaluando paso a paso realmente, simplemente se tiene una mayor recompensa entre más rápido se llegue a la meta.

Aquí fue necesario tomar una decisión, pues fácilmente se puede hacer la modificación presentada en el código 3.4 considerando la recompensa por cada paso como -1 , es sencillo usar una serie geométrica, pues:

$$R(\sigma; \gamma) = \sum_{t=1}^H \gamma^{H-t} r(s_t, a_t) \quad (3.10)$$

$$= \sum_{t=1}^H \gamma^{H-t} (-1) \quad (3.11)$$

$$= -(\gamma^{H-1} + \gamma^{H-2} + \dots + \gamma^{H-(H-1)} + \gamma^{H-H}) \quad (3.12)$$

$$= -\sum_{t=0}^{H-1} \gamma^t \quad (3.13)$$

$$= -\frac{\gamma^H - 1}{\gamma - 1} \quad (3.14)$$

Sin embargo, a pesar que la modificación mostrada en 3.4 tiene sentido matemático, no tiene sentido a la hora de modelar una irracionalidad, pues esta haría que trayectorias largas no tengan una recompensa tan baja como deberían, pues el profesor “olvidaría” los primeros pasos porque ocurrieron hace mucho tiempo.

En un contexto en que la recompensa asociada a la trayectoria es el inverso aditivo de su largo no tiene sentido modificar `get_trajectory_return()` para considerar γ . Y ya que se está aplicando una versión adaptada del benchmark, se puede decidir no considerar γ .

Código 3.4 : SimpleMountainCarPreferenceEnvBPref.get_trajectory_return()

```
def get_trajectory_return(self, tr, gamma=1):
2
    if gamma != 1:
        reward = -(-1+gamma**len(tr[1]))/(-1+gamma)
    else:
        reward = -len(tr[1])
7
    return reward
```

Sería interesante revisar qué hace el benchmark con escenarios como este, aunque pro-

blemente lo único que pueda hacerse es no considerar al profesor cuya irracionalidad es el comportamiento “miope”.

3.2.3. RandomMDP

Este último escenario se trata de un MDP en que tanto las dinámicas del sistema como las recompensas del profesor son seleccionadas aleatoriamente para generar un problema que no necesariamente representa algo del mundo real, pero que un algoritmo RL debe ser capaz de resolver.

Ya que este tiene recompensas asociadas a cada paso, como RiverSwim, no al episodio completo, como SimpleMountainCar, se tuvo que hacer las mismas modificaciones que aquellas para RiverSwimEnv y queda listo el ambiente RandomMDPBPrefEnv

3.3. Adaptación de DPS

Finalmente, ya que los algoritmos DPS no consideran *Saltar consulta* como una respuesta posible, fue necesario agregar soporte para que el algoritmo de aprendizaje pudiese recibir `preference = np.nan`, por lo que se agregaron las líneas:

Código 3.5 : Soporte para consulta saltada

```
if preference == 0.5:
    Ties += 1
3    continue
if np.isnan(preference):
    Skipped += 1
    continue
```

Ya existía soporte para `preference = 0.5`, pero se agregó esta modificación para contabilizar cuántos empates habían. Además, se modificaron los algoritmos para que no solo retornaran las recompensas, si no también el número de empates y de consultas saltadas.

4. Resultados

Se hicieron pruebas a los distintos algoritmos DPS enfrentándolos a cada uno de los profesores descritos en la sección 2.2, con la añadidura de considerar un profesor estocástico con modelo de asignación de preferencia con ruido lineal y uno con ruido logístico, que es la versión implementada por B-Pref oficial.

Se decidió además hacer pruebas con el algoritmo EPMC [6] y el algoritmo PSRL [7], pues estos fueron los algoritmos con los que DPS fue comparado tras su creación.

El primero de estos algoritmos extra es Every-Visit Preference Monte Carlo (EPMC), se trata de otro algoritmo de aprendizaje basado en preferencias, que busca aprender la política óptima basado en su experiencia, registra cada visita que hace a los distintos estados que recorre durante sus trayectorias, acumulando y promediando las recompensas que ha recibido en el pasado para estimar de forma más precisa la función de recompensas real. Este enfoque es particularmente útil en entornos con dinámicas estocásticas o cuando se requiere una exploración exhaustiva del espacio de estados.

Por otro lado el segundo es un algoritmo RL tradicional, que tiene acceso a la función de recompensas real en cada paso y que, como DPS, usa ideas Bayesianas para tomar sus decisiones, permitiéndole primero explorar bien su entorno, para luego explotar lo que funciona. Cabe mencionar que, como este algoritmo tiene acceso a la función de recompensa real, completa, funciona como una cota superior, muestra lo que puede hacer un algoritmo si no tiene que lidiar con aprender en base a preferencias, aunque claro, eso hace que sea más difícil que un usuario normal pueda utilizarlo exitosamente, pues debe definir correctamente una función de recompensas.

Por último, como PSRL no es basado en preferencias, no se aplicaron los distintos profesores de B-Pref, pues el algoritmo no funciona de esa manera.

4.1. Estructura de las pruebas y formato de presentación de resultados

Para cada uno de los profesores mostrados a continuación, se corrió cada uno de los algoritmos DPS y EMPC 100 veces, considerando 200 consultas por cada partida, en que cada consulta consta de 2 trayectorias con un horizonte de tiempo de 50 pasos.

Para la visualización de datos se realiza una modificación que no sería posible tener en tareas reales. Dado que se está trabajando con entornos simulados, hay acceso a la función de recompensas real de la tarea por cada paso que el agente toma.

En cada consulta el agente presenta 2 trayectorias de 50 pasos, recibe una preferencia, aprende, y genera otras 2 trayectorias de 50 pasos. No se genera aprendizaje entre la demostración de una trayectoria y la siguiente. Por esto, para visualizar la mejora en las trayectorias propuestas por el algoritmo con cada consulta, en cada consulta se tomó un promedio entre las recompensas reales obtenidas por ambas trayectorias en cada paso, de modo que por cada consulta se tienen las recompensas de una “trayectoria promedio” que fue mostrada en dicha consulta.

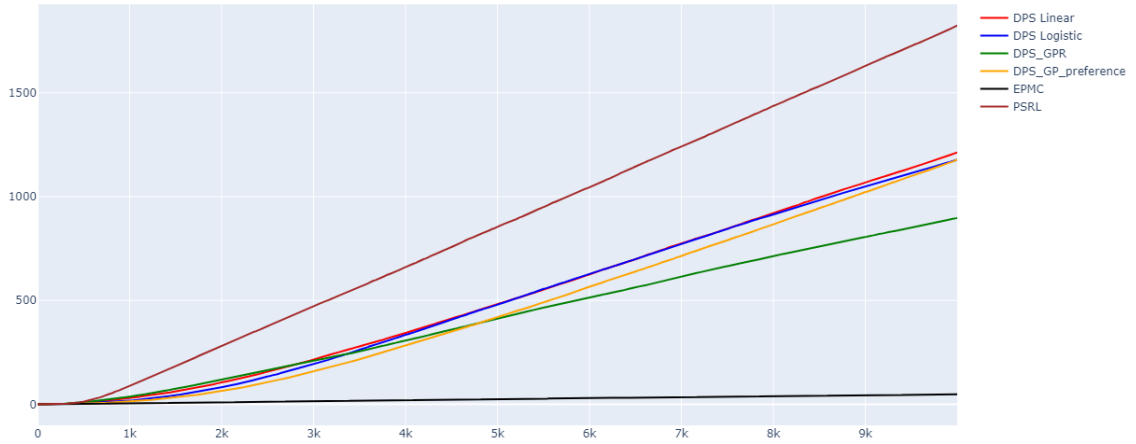
Además, el algoritmo fue ejecutado 100 veces para tener una noción del desempeño esperado de este. Dada la naturaleza estocástica del muestreo de Thompson que realiza el algoritmo para generar nuevas consultas, puede ser que una vez que se ejecutó, el algoritmo haya sido capaz de aprender muy rápido cómo realizar la tarea de una manera óptima, mientras que en otra ejecución es posible que el algoritmo no haya conseguido un aprendizaje relevante si no hasta varias consultas después. Para obtener un rendimiento esperado se tomó un promedio de las recompensas obtenidas en cada paso por el algoritmo las 100 veces que fue ejecutado.

Finalmente, para una visualización más sencilla, lo que se muestra en las siguientes subsecciones es la recompensa promedio acumulada conforme avanza el tiempo, de modo que es más sencillo ver qué algoritmo mejoró más rápido y cuál tuvo un mejor desempeño total.

Los gráficos de la recompensa promedio se encuentran en el anexo.

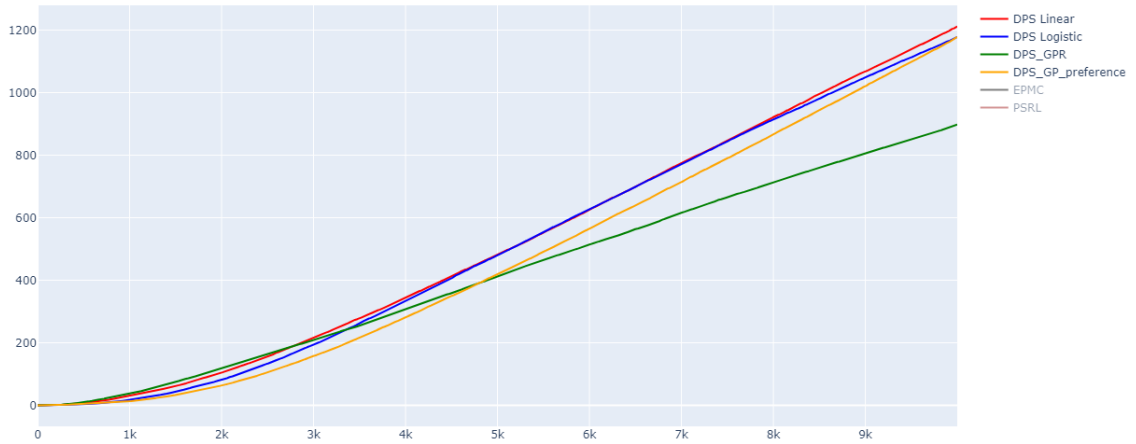
4.2. SimTeacher: Oráculo

Cumulative Mean reward across runs, SimTeacher=Oracle



(a) Recompensa promedio acumulada de PSRL, las distintas versiones de DPS y de EPMC

Cumulative Mean reward across runs, SimTeacher=Oracle



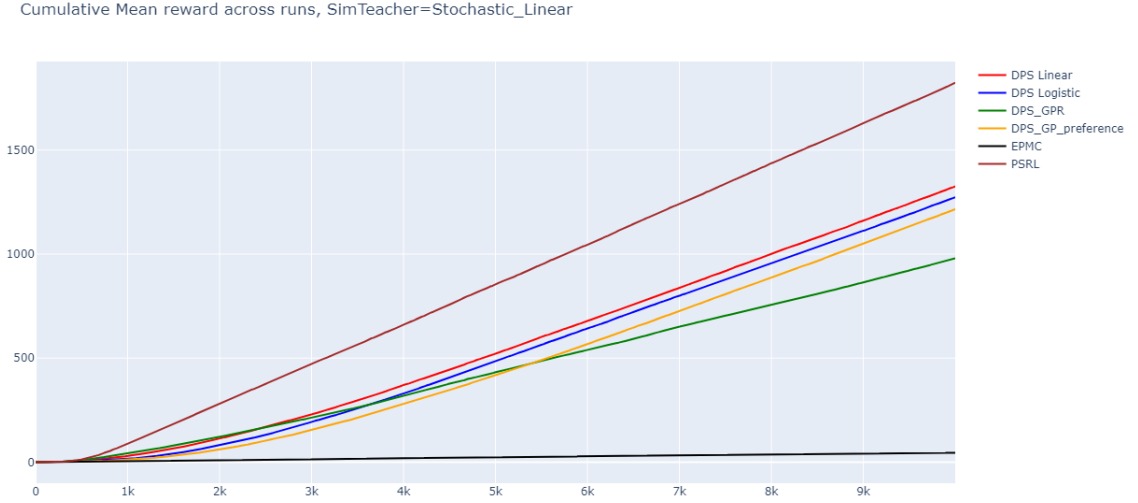
(b) Recompensa promedio acumulada de las distintas versiones de DPS

Figura 4.6: SimTeacher($\beta = \infty, \gamma = 1, \epsilon = 0, \delta_{skip} = 0, \delta_{equal} = 0$)

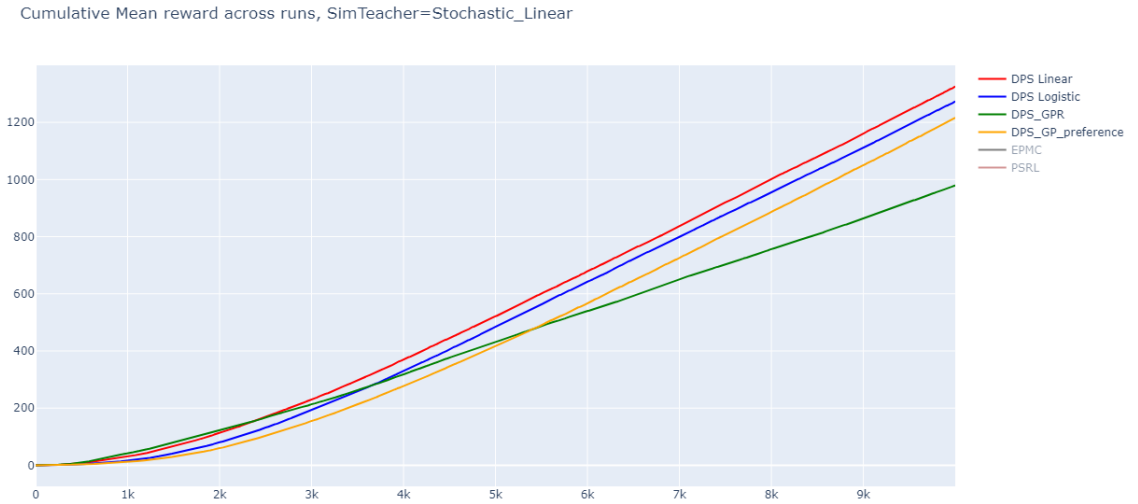
4.3. SimTeacher: Estocástico

4.3.1. Ruido lineal

La asignación de preferencias se hizo siguiendo el modelo en (3.9)



(a) Recompensa promedio acumulada de PSRL, las distintas versiones de DPS y de EPMC

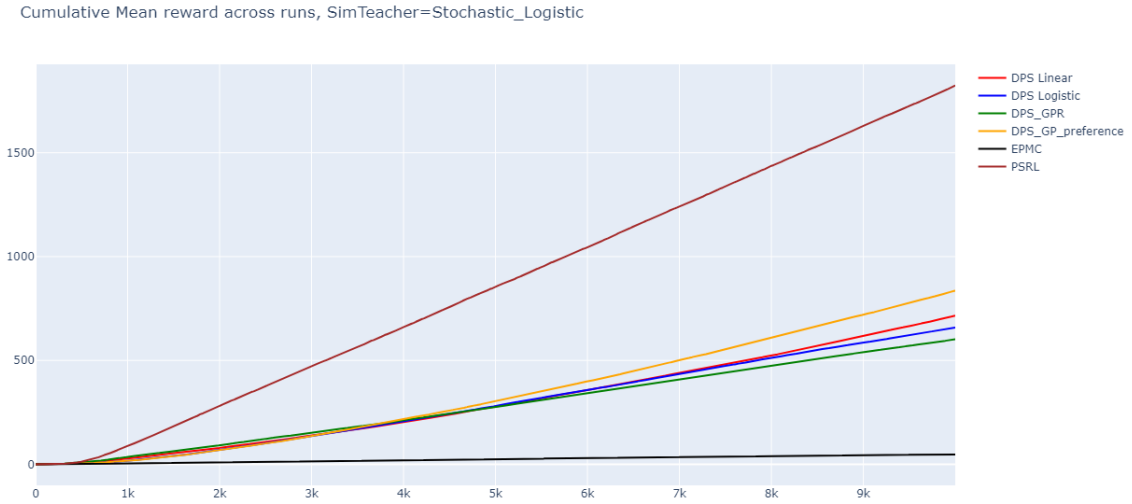


(b) Recompensa promedio acumulada de las distintas versiones de DPS

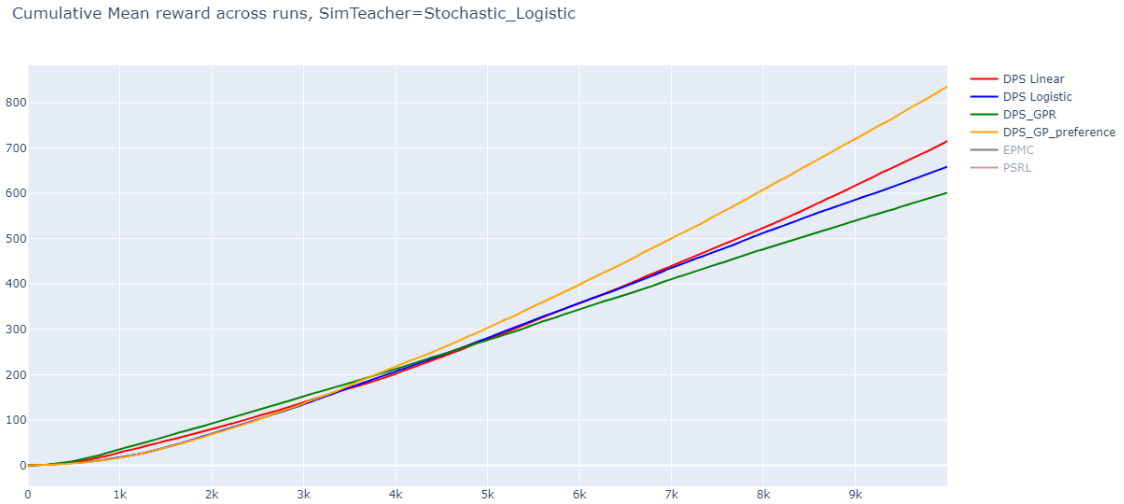
Figura 4.7: SimTeacher ($\beta = 1, \gamma = 1, \epsilon = 0, \delta_{skip} = 0, \delta_{equal} = 0$). Con modelo lineal del ruido.

4.3.2. Ruido logístico

La asignación de preferencias se hizo siguiendo el modelo en (2.7)



(a) Recompensa promedio acumulada de PSRL, las distintas versiones de DPS y de EPMC

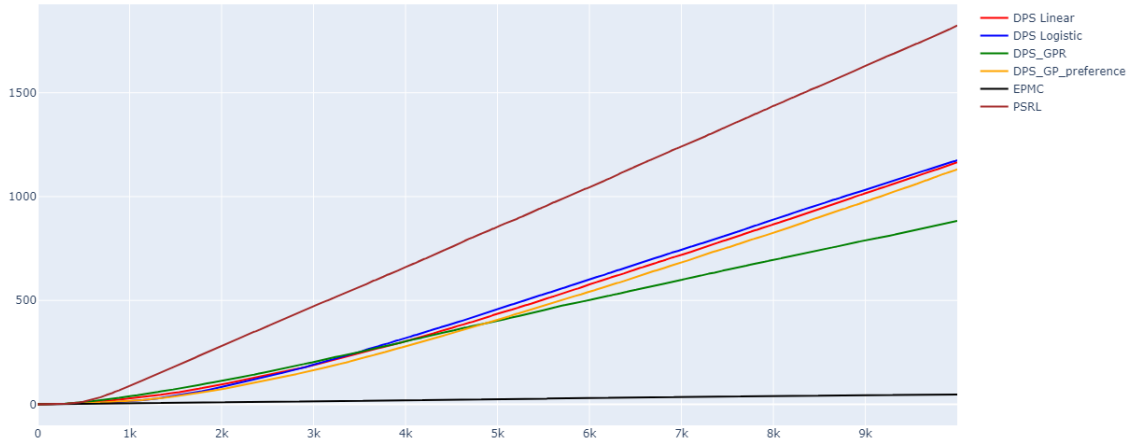


(b) Recompensa promedio acumulada de las distintas versiones de DPS

Figura 4.8: $\text{SimTeacher}(\beta = 1, \gamma = 1, \epsilon = 0, \delta_{\text{skip}} = 0, \delta_{\text{equal}} = 0)$. Con modelo logístico del ruido.

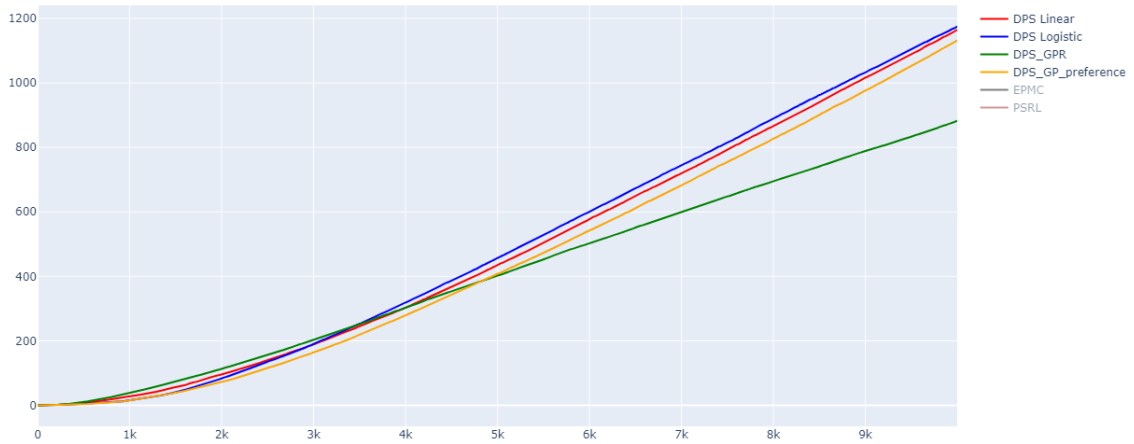
4.4. SimTeacher: Error

Cumulative Mean reward across runs, SimTeacher=Mistake



(a) Recompensa promedio acumulada de PSRL, las distintas versiones de DPS y de EPMC

Cumulative Mean reward across runs, SimTeacher=Mistake

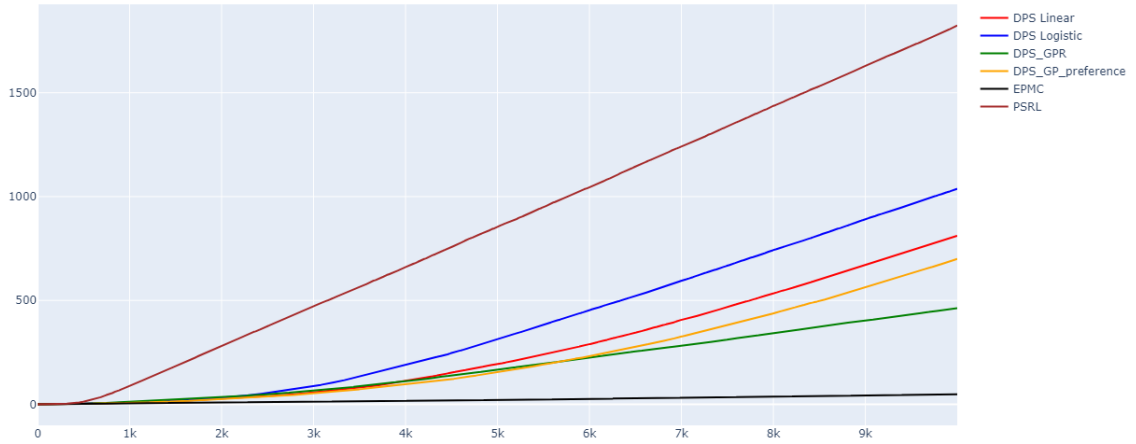


(b) Recompensa promedio acumulada de las distintas versiones de DPS

Figura 4.9: $\text{SimTeacher}(\beta = \infty, \gamma = 1, \epsilon = 0.1, \delta_{skip} = 0, \delta_{equal} = 0)$

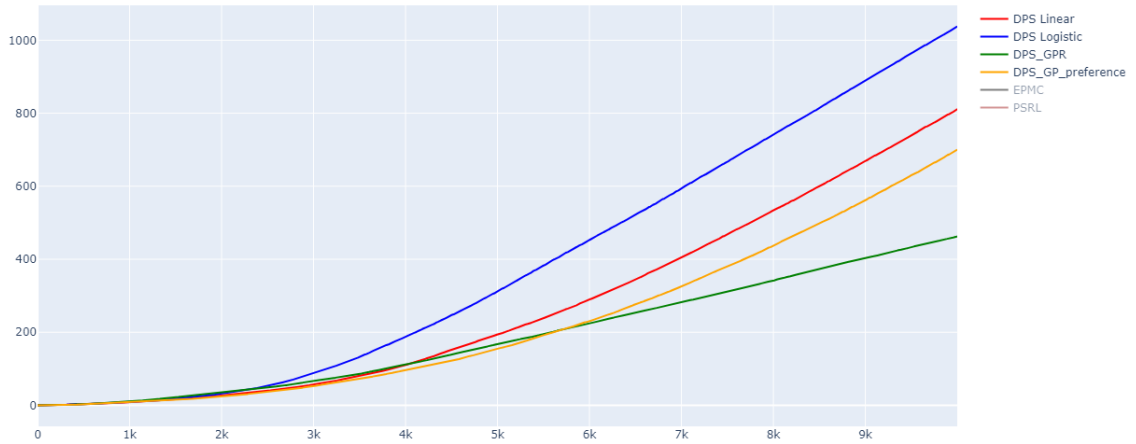
4.5. SimTeacher: Salto

Cumulative Mean reward across runs, SimTeacher=Skip



(a) Recompensa promedio acumulada de PSRL, las distintas versiones de DPS y de EPMC

Cumulative Mean reward across runs, SimTeacher=Skip

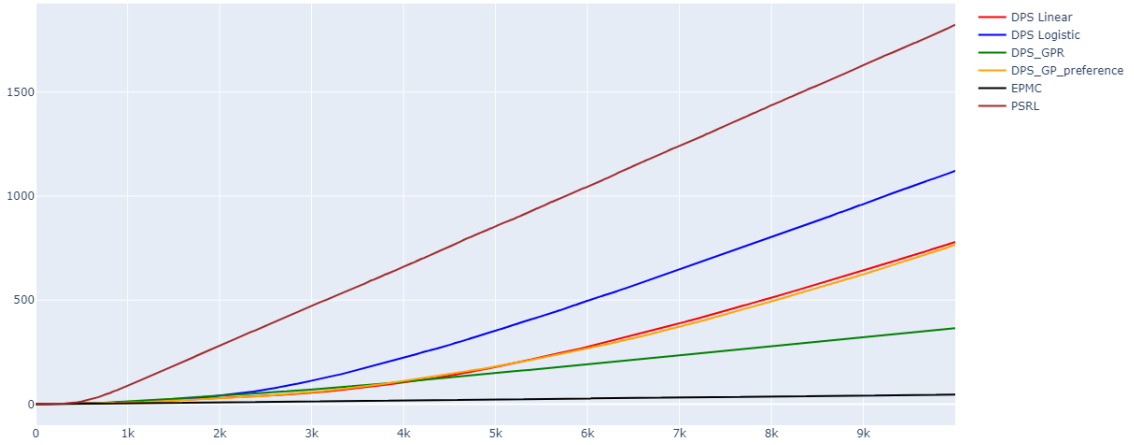


(b) Recompensa promedio acumulada de las distintas versiones de DPS

Figura 4.10: SimTeacher($\beta = \infty, \gamma = 1, \epsilon = 0, \delta_{skip} = 0.1, \delta_{equal} = 0$)

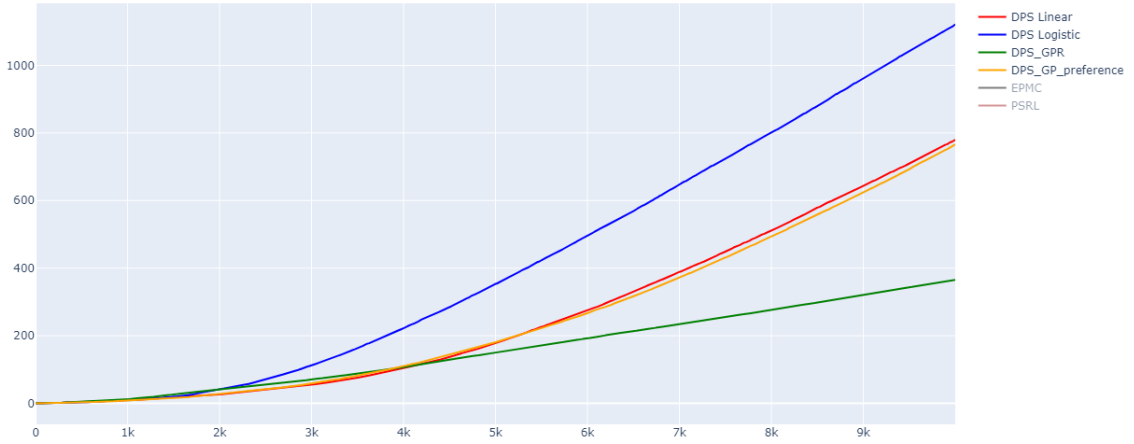
4.6. SimTeacher: Igual

Cumulative Mean reward across runs, SimTeacher=Equal



(a) Recompensa promedio acumulada de PSRL, las distintas versiones de DPS y de EPMC

Cumulative Mean reward across runs, SimTeacher=Equal

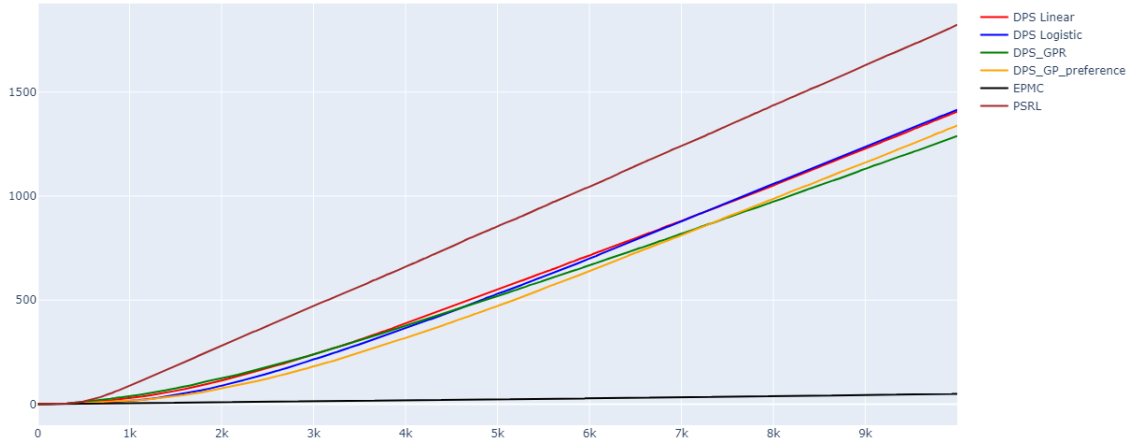


(b) Recompensa promedio acumulada de las distintas versiones de DPS

Figura 4.11: $\text{SimTeacher}(\beta = \infty, \gamma = 1, \epsilon = 0, \delta_{skip} = 0, \delta_{equal} = 0.1)$

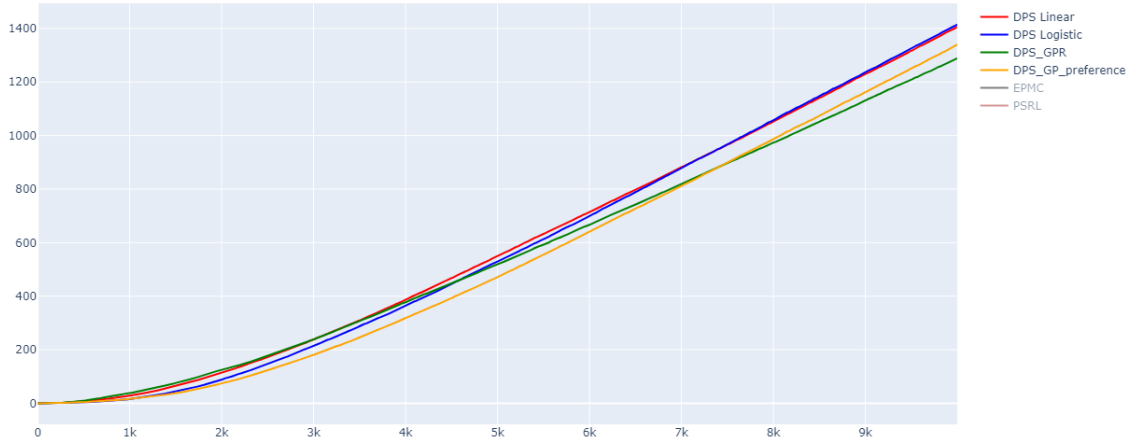
4.7. SimTeacher: Miope

Cumulative Mean reward across runs, SimTeacher=Myopic



(a) Recompensa promedio acumulada de PSRL, las distintas versiones de DPS y de EPMC

Cumulative Mean reward across runs, SimTeacher=Myopic



(b) Recompensa promedio acumulada de las distintas versiones de DPS

Figura 4.12: SimTeacher($\beta = \infty, \gamma = 0.9, \epsilon = 0, \delta_{skip} = 0, \delta_{equal} = 0$)

4.8. Análisis de Resultados

En primer lugar, como se había comentado anteriormente, PSRL funcionaría como una cuota superior. Al ser un algoritmo RL tradicional tiene acceso a la función de recompensas en cada paso, y como se trata de un problema simulado, esta ya está bien definida y el algoritmo se encargará de explorar y explotar para conseguir el mejor desempeño posible.

En cuanto a EPMC, se puede ver que se desempeñó muy mal sin importar el profesor, siendo en los que mejor le fue, los profesores estocásticos, aunque esto se puede apreciar mejor en las Figuras 5.14 y 5.15 del anexo. Esto probablemente sea porque el algoritmo EPMC fue el que más empates reportó y los profesores con $\beta \neq 0$ no pueden reportar un empate, por lo que el algoritmo siempre tendrá algo para aprender, aunque no necesariamente sea un dato correcto, es mejor que un empate, pues en con los empates simplemente se pasa a la siguiente consulta sin aprender nada.

Se puede ver la cantidad de empates y saltos reportados por cada algoritmo ante cada profesor en la figura 5.20 del anexo.

En cuanto a los distintos algoritmos Dueling Posterior Sampling, se puede ver que en general DPS_logistic se desempeña un poco mejor que el resto en los escenarios sin ruido, mientras que para el profesor con ruido lineal, DPS_linear presenta mejores trayectorias y para el con ruido logístico, es DPS_GP_Preference quien obtiene un mayor retorno acumulado. A pesar de esto, las diferencias de desempeño no suelen ser muy grandes entre los distintos métodos de asignación de crédito y no hay uno que sea significativamente superior ante todos los profesores modelados.

Otras observaciones sobre los resultados:

- Todas las versiones de DPS son bastante robustas ante el profesor que ignoraba los primeros pasos de la trayectoria, el profesor “miope”, pero esto puede haberse dado por la naturaleza del problema. En RiverSwim, mientras no se alcance el final del río, la recompensa será cero, por lo que es posible que las cosas que el profesor haya olvidado no hayan sido relevantes de todos modos.

- Todos los algoritmos DPS parecen ser bastante resistentes ante el profesor que puede cometer errores, siendo el con el mejor desempeño DPS_logistic, pero seguido muy de cerca por DPS_linear y DPS_GP_preference.
- Hubo un desempeño relativamente pobre ante ambos profesores indecisos (Skip e Equal), aunque en ambos, el orden de mejor a peor resultó ser DPS_logistic, DPS_linear, DPS_GP_preference y por último DPS_GPR. Tiene sentido que con este profesor todos disminuyeran su rendimiento, pues los algoritmos DPS no pueden aprender lo que el usuario quiere ante un empate ni ante una consulta saltada, simplemente aprenden las dinámicas del sistema mientras demuestran las trayectorias de la consulta y luego pasan a la siguiente, sin haber aprendido nada del profesor. La siguiente consulta es generada con el modelo actualizado de las dinámicas del sistema, pero con el modelo de recompensas que se tenía hasta la interacción anterior.
- Ante todos los profesores presentados, DPS_GPR tuvo un desempeño ligeramente inferior a los otros algoritmos DPS, incluso ante el oráculo.

5. Conclusiones

Este documento describe la implementación de una prueba inspirada en el benchmark B-Pref en los algoritmos de aprendizaje reforzado basado en preferencias de la familia Dueling Posterior Sampling, así como las modificaciones que fue necesario realizar para la implementación.

Se han comparado los distintos modelos de asignación de crédito que consideran los algoritmos DPS y se ha llegado a la conclusión de que los distintos modelos presentan distintos desempeños ante las diferentes irracionalidades, no hay uno que sea significativamente superior al resto ante todos los profesores simulados. Sin embargo, al menos en las pruebas realizadas, DPS_GPR suele ser el con el peor desempeño entre los algoritmos DPS.

Se ha observado también que la naturaleza de la tarea a la que se enfrenta el agente puede hacer que este se muestre más robusto de lo que realmente es a alguna irracionalidad específica, como pasó con RiverSwim y el profesor con problemas de memoria. Por último, la conclusión más importante de este trabajo es que, al menos para esta familia de algoritmos, la indecisión es peor que el error. Si bien el desempeño ante el profesor que puede cometer errores no fue el mejor de los reportados, Skip e Equal consiguieron retornos mucho menores para casi todos los algoritmos. Quizás el enfoque que debería tomar PBRL a futuro es centrarse en satisfacer a estos usuarios, pues es una “irracionalidad” que perjudica bastante a los algoritmos actuales, pero una que no es tan complicado tomar en consideración.

5.1. Trabajo futuro.

Como se mencionó en la introducción, desde un inicio este trabajo fue pensado como la primera etapa de un trabajo más complejo para la creación de un algoritmo PBRL. Con base en los resultados obtenidos a lo largo del trabajo, se proponen las siguientes mejoras del trabajo presentado, para ser llevadas a cabo en el futuro:

- La implementación del benchmark oficial desarrollado por K. Lee et al. [1] a fin de poder comparar estos algoritmos con otros que ya han aplicado esta prueba, como son PEBBLE y PPO sin necesidad de probarlos junto con los algoritmos actuales.
- La implementación de estas pruebas en más escenarios, para evitar que la naturaleza del problema haga que el algoritmo pueda mostrarse más robusto de lo que realmente es ante alguna irracionalidad específica.
- Para el desarrollo de algoritmos más robustos a usuarios indecisos, podría desarrollarse un algoritmo que tome en consideración la cantidad de empates o saltos que han ocurrido ultimamente. Si se supera cierto umbral de empates seguidos, podría cambiarse la forma en que el algoritmo resuelve el MDP interno, favoreciendo la exploración. Esto haría más probable la aparición de trayectorias significativamente diferentes entre sí, lo que podría ayudar a un usuario indeciso a decantarse por una.
- Como fue mencionado en la introducción, un objetivo a largo plazo de esta investigación es la posibilidad de unir uno de estos algoritmos con uno de meta-aprendizaje. Según lo visto en esta investigación, cuál modelo de asignación de crédito se quiere utilizar depende más del tipo de irracionalidad y de los escenarios a los que se quiera enfrentar el agente a que haya un ganador definitivo entre estos.

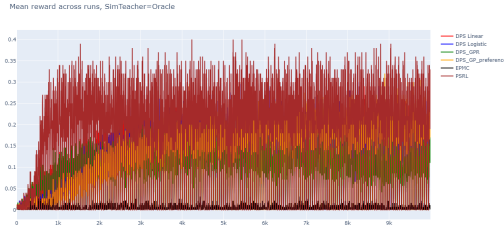
Referencias

- [1] K. Lee, L. Smith, A. Dragan, and P. Abbeel, “B-pref: Benchmarking preference-based reinforcement learning,” in *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021. [Online]. Available: https://openreview.net/forum?id=ps95-mkHF_
- [2] D. J. H. III and D. Sadigh, “Few-shot preference learning for human-in-the-loop rl,” in *Proceedings of The 6th Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, K. Liu, D. Kulic, and J. Ichnowski, Eds., vol. 205. PMLR, 14–18 Dec 2023, pp. 2014–2025. [Online]. Available: <https://proceedings.mlr.press/v205/iii23a.html>
- [3] R. Akrou, M. Schoenauer, and M. Sebag, “Preference-based policy learning,” 09 2011, pp. 12–27.
- [4] Gym Library, “Mountain Car Environment,” https://www.gymlibrary.dev/environments/classic_control/mountain_car/, accedido el 28 de febrero de 2024.
- [5] E. R. Novoseller, Y. Wei, Y. Sui, Y. Yue, and J. W. Burdick, “Dueling posterior sampling for preference-based reinforcement learning,” 2019. [Online]. Available: <https://arxiv.org/abs/1908.01289>
- [6] C. Wirth and J. Fürnkranz, “Epmmc: Every visit preference monte carlo for reinforcement learning,” in *Asian Conference on Machine Learning*, 2013. [Online]. Available: <https://api.semanticscholar.org/CorpusID:7838906>
- [7] O. Ian, V. R. Benjamin, and R. Daniel, “(more) efficient reinforcement learning via posterior sampling,” in *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS’13. Red Hook, NY, USA: Curran Associates Inc., 2013, p. 3003–3011.
- [8] D. Sadigh, A. Dragan, S. Sastry, and S. Seshia, “Active preference-based learning of reward functions.” *UC Berkeley*, 2017. [Online]. Available: <https://escholarship.org/uc/item/88k894w7>
- [9] Y. Xu, R. Wang, L. F. Yang, A. Singh, and A. Dubrawski, “Preference-based reinforcement learning with finite-time guarantees,” 2020. [Online]. Available: <https://arxiv.org/abs/2006.08910>
- [10] E. Bıyık, N. Huynh, M. J. Kochenderfer, and D. Sadigh, “Active preference-based gaussian process regression for reward learning,” 2020. [Online]. Available: <https://arxiv.org/abs/2005.02575>
- [11] C. Finn, P. Abbeel, and S. Levine, “Model-agnostic meta-learning for fast adaptation of deep networks,” 2017.

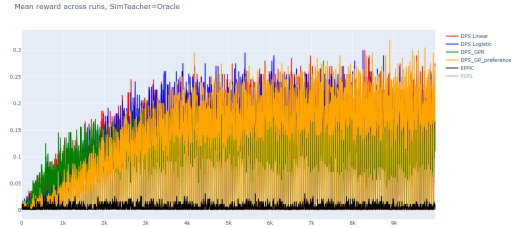
- [12] R. A. Bradley and M. E. Terry, “Rank analysis of incomplete block designs: I. the method of paired comparisons,” *Biometrika*, vol. 39, no. 3/4, pp. 324–345, 1952. [Online]. Available: <http://www.jstor.org/stable/2334029>
- [13] K. Lee, L. Smith, and P. Abbeel, “Pebble: Feedback-efficient interactive reinforcement learning via relabeling experience and unsupervised pre-training,” 2021.
- [14] P. Christiano, J. Leike, T. B. Brown, M. Martic, S. Legg, and D. Amodei, “Deep reinforcement learning from human preferences,” 2017. [Online]. Available: <https://arxiv.org/abs/1706.03741>
- [15] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” 2017.

Anexos

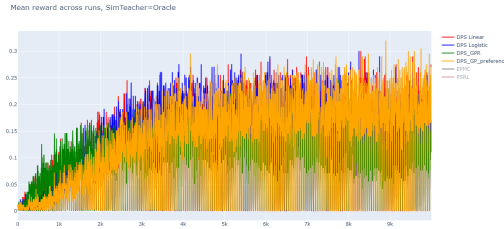
Recompensas promedio



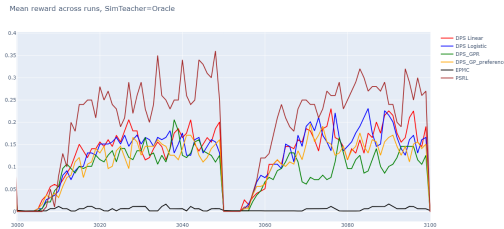
(a) Recompensa promedio de PSRL, las distintas versiones de DPS y de EPMC



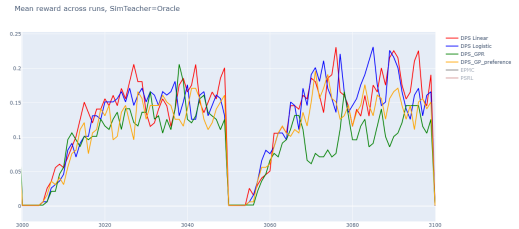
(b) Recompensa promedio acumulada de las distintas versiones de DPS y de EPMC



(c) Recompensa promedio solo de las distintas versiones de DPS

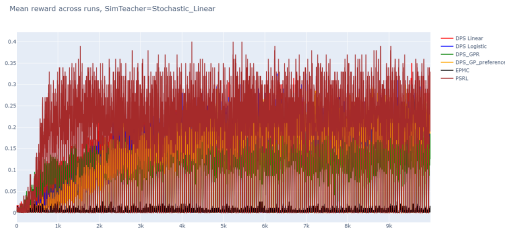


(d) Detalle de recompensas promedio en 2 consultas.

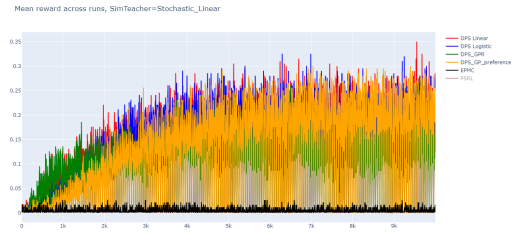


(e) Detalle de recompensas promedio en 2 consultas, solo DPS.

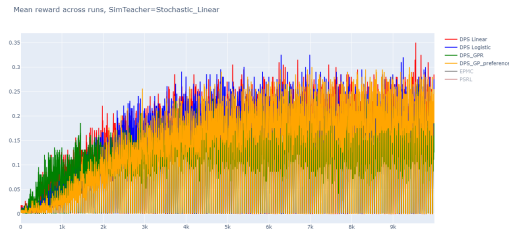
Figura 5.13: Recompensas promedio al enfrentarse al oráculo.



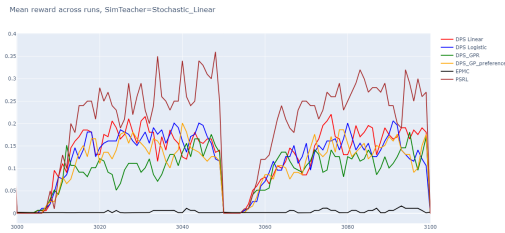
(a) Recompensa promedio de PSRL, las distintas versiones de DPS y de EPMC



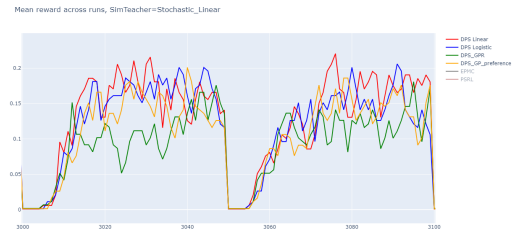
(b) Recompensa promedio acumulada de las distintas versiones de DPS y de EPMC



(c) Recompensa promedio solo de las distintas versiones de DPS

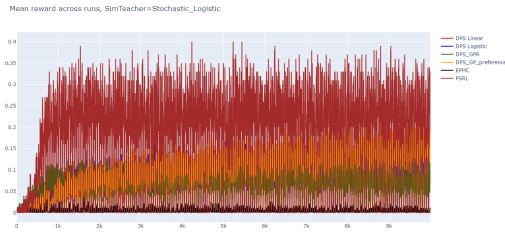


(d) Detalle de recompensas promedio en 2 consultas.

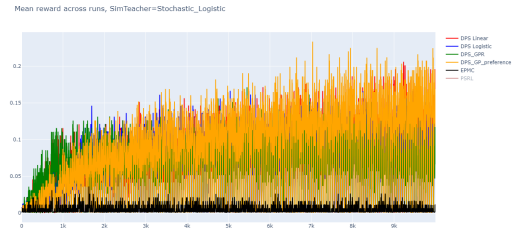


(e) Detalle de recompensas promedio en 2 consultas, solo DPS.

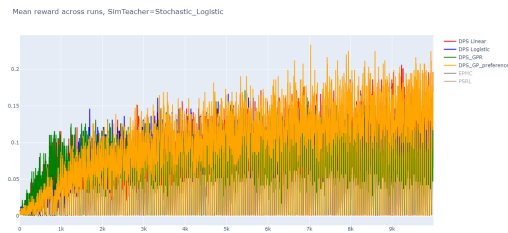
Figura 5.14: Recompensas promedio al enfrentarse al profesor estocástico con ruido lineal.



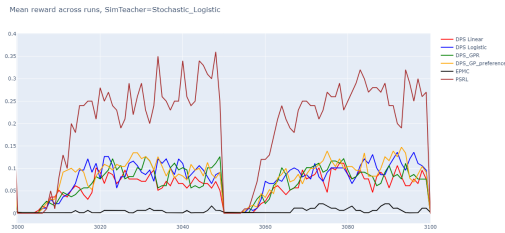
(a) Recompensa promedio de PSRL, las distintas versiones de DPS y de EPMC



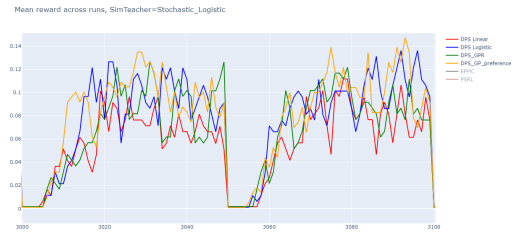
(b) Recompensa promedio acumulada de las distintas versiones de DPS y de EPMC



(c) Recompensa promedio solo de las distintas versiones de DPS

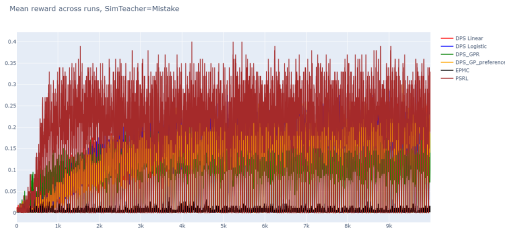


(d) Detalle de recompensas promedio en 2 consultas.

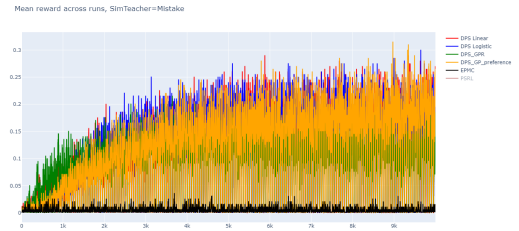


(e) Detalle de recompensas promedio en 2 consultas, solo DPS.

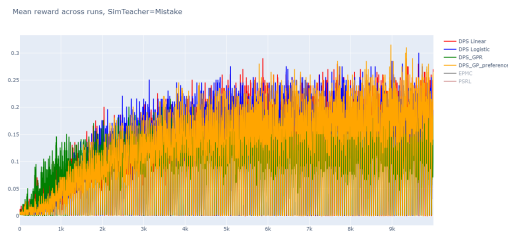
Figura 5.15: Recompensas promedio al enfrentarse al profesor estocástico con ruido logístico.



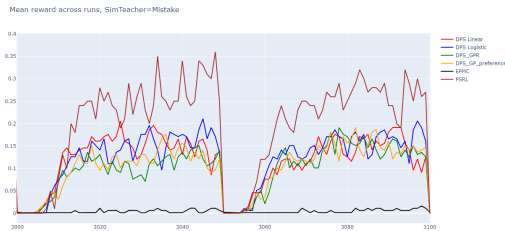
(a) Recompensa promedio de PSRL, las distintas versiones de DPS y de EPMC



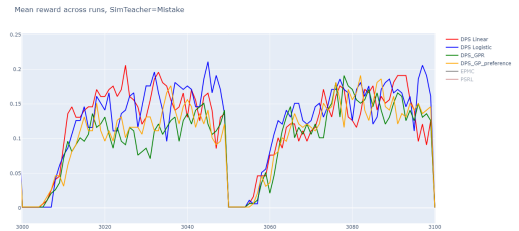
(b) Recompensa promedio acumulada de las distintas versiones de DPS y de EPMC



(c) Recompensa promedio solo de las distintas versiones de DPS

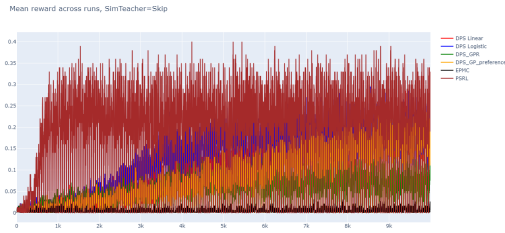


(d) Detalle de recompensas promedio en 2 consultas.

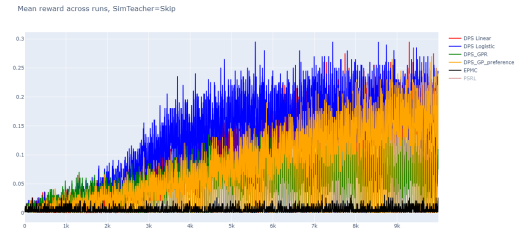


(e) Detalle de recompensas promedio en 2 consultas, solo DPS.

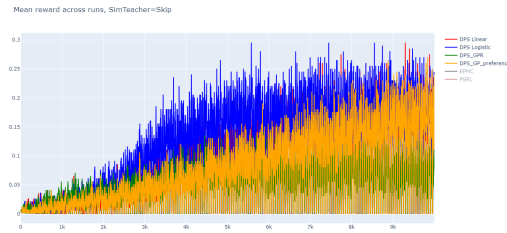
Figura 5.16: Recompensas promedio al enfrentarse al profesor que puede cometer errores.



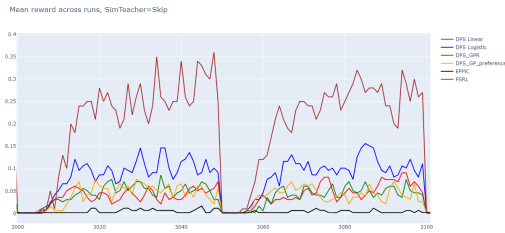
(a) Recompensa promedio de PSRL, las distintas versiones de DPS y de EPMC



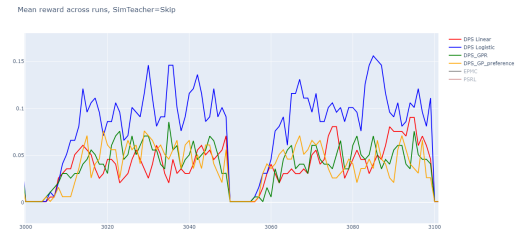
(b) Recompensa promedio acumulada de las distintas versiones de DPS y de EPMC



(c) Recompensa promedio solo de las distintas versiones de DPS

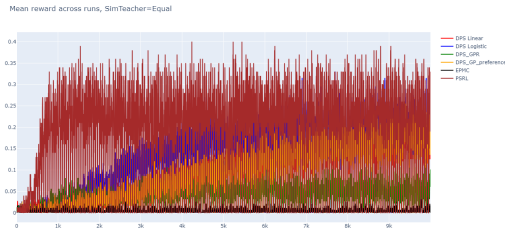


(d) Detalle de recompensas promedio en 2 consultas.

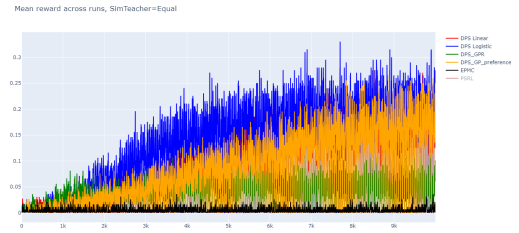


(e) Detalle de recompensas promedio en 2 consultas, solo DPS.

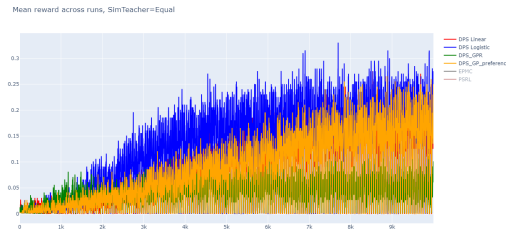
Figura 5.17: Recompensas promedio al enfrentarse al profesor que puede saltar consultas.



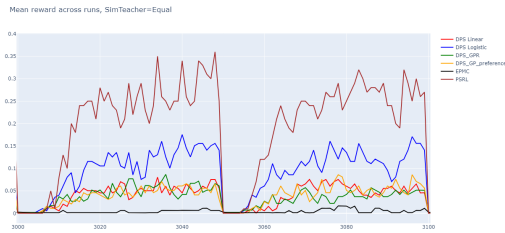
(a) Recompensa promedio de PSRL, las distintas versiones de DPS y de EPMC



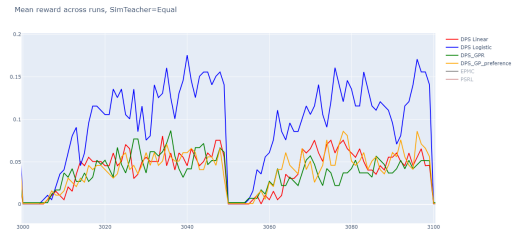
(b) Recompensa promedio acumulada de las distintas versiones de DPS y de EPMC



(c) Recompensa promedio solo de las distintas versiones de DPS

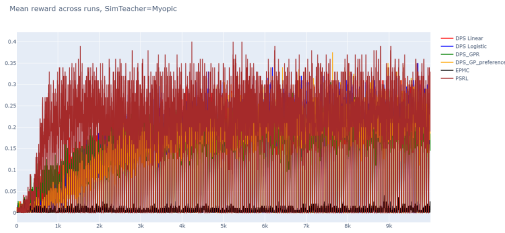


(d) Detalle de recompensas promedio en 2 consultas.

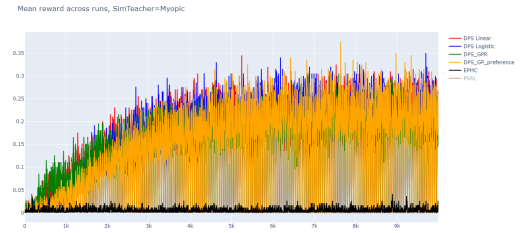


(e) Detalle de recompensas promedio en 2 consultas, solo DPS.

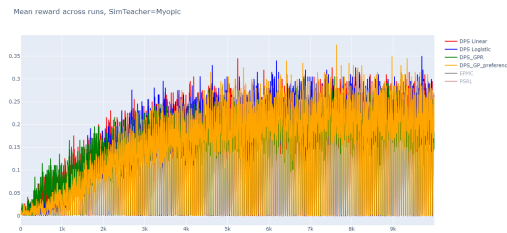
Figura 5.18: Recompensas promedio al enfrentarse al profesor que suele informar un empate.



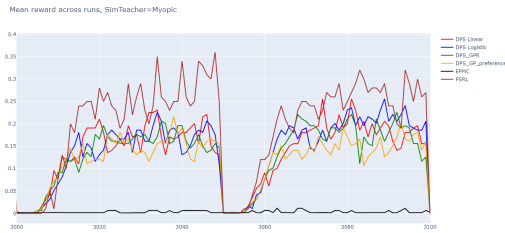
(a) Recompensa promedio de PSRL, las distintas versiones de DPS y de EPMC



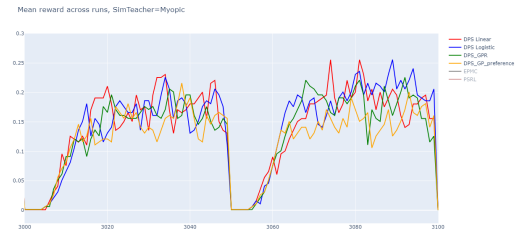
(b) Recompensa promedio acumulada de las distintas versiones de DPS y de EPMC



(c) Recompensa promedio solo de las distintas versiones de DPS



(d) Detalle de recompensas promedio en 2 consultas.



(e) Detalle de recompensas promedio en 2 consultas, solo DPS.

Figura 5.19: Recompensas promedio al enfrentarse al profesor con comportamiento “miope”.

Empates y Consultas Saltadas

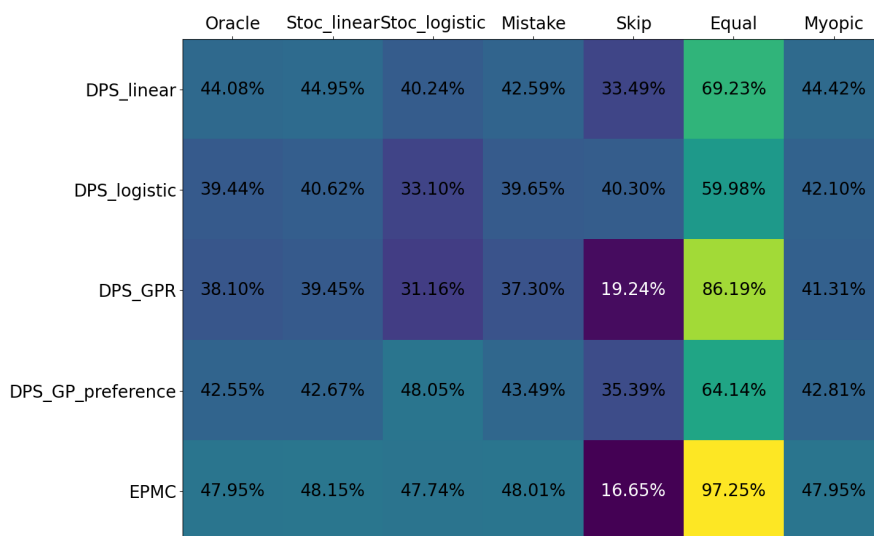
	Oracle	Stoc_linear	Stoc_logistic	Mistake	Skip	Equal	Myopic
DPS_linear	88.16	89.89	80.49	85.17	66.98	138.47	88.84
DPS_logistic	78.87	81.24	66.21	79.29	80.59	119.96	84.20
DPS_GPR	76.21	78.91	62.32	74.60	38.48	172.37	82.62
DPS_GP_preference	85.10	85.35	96.09	86.99	70.78	128.29	85.62
EPMC	95.90	96.30	95.49	96.02	33.30	194.49	95.90

(a) Empates reportados por cada profesor para cada algoritmo

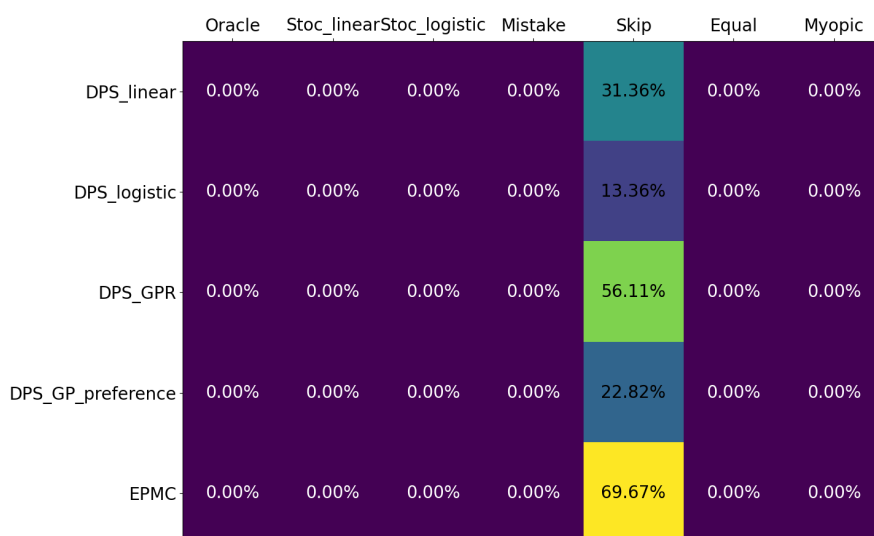
	Oracle	Stoc_linear	Stoc_logistic	Mistake	Skip	Equal	Myopic
DPS_linear	0.00	0.00	0.00	0.00	62.73	0.00	0.00
DPS_logistic	0.00	0.00	0.00	0.00	26.71	0.00	0.00
DPS_GPR	0.00	0.00	0.00	0.00	112.22	0.00	0.00
DPS_GP_preference	0.00	0.00	0.00	0.00	45.65	0.00	0.00
EPMC	0.00	0.00	0.00	0.00	139.35	0.00	0.00

(b) Consultas saltadas por cada profesor para cada algoritmo

Figura 5.20: Empates y consultas saltadas promedio para cada combinación profesor/algoritmo.



(a) Empates reportados por cada profesor para cada algoritmo



(b) Consultas saltadas por cada profesor para cada algoritmo

Figura 5.21: Empates y consultas saltadas en promedio para cada combinación profesor/algoritmo, expresado como porcentaje de las consultas totales realizadas por cada combinación.