Universidad de Concepción

Facultad de Ingeniería

Ingeniería Informática y Ciencias de la Computación

Selección de Solver Anytime para problemas de Multi Agent Path Finding con Machine Learning

Tesis de Magíster

Angelo Enrique Zapata Lillo

Profesores Guía: Roberto Asín y Julio Godoy

Comisión Interna: Pedro Pinacho y Pierluigi Cerulo

Comisión Externa: Jorge Baier

Concepción, Chile 2022

Resumen

En este estudio, se propone y desarrolla un metasolver basado en Machine Learning para problemas de Multi Agent Path Finding (MAPF), con el objetivo de seleccionar el solver más adecuado en función de las características específicas del problema y el límite de tiempo proporcionado por el usuario. El enfoque busca mejorar el rendimiento del Single Best Solver y aproximarse al rendimiento del Virtual Best Solver. Para ello, se recopiló un conjunto de datos amplio y variado, y se seleccionaron y modificaron algoritmos de última generación que pudieran manejar eficientemente el tiempo límite. Se identificaron características relevantes y se construyó un modelo de Machine Learning preciso y robusto utilizando el algoritmo XGBoost. El modelo se evaluó en términos de la métrica $\hat{m}$ y se comparó con otros métodos del estado del arte. Los resultados demuestran que el enfoque propuesto es efectivo y consistente, superando el rendimiento del Single Best Solver y aproximándose al desempeño del Virtual Best Solver.

Índice general

Capítulo 1

Introducción

1.1. Contexto

Grandes empresas, como Amazon, han realizado avances significativos en el mundo del e-commerce, adquiriendo capacidades de entrega de productos sin precedentes, realizando entregas en un día o incluso horas. La mejora de estas entregas viene a la par de grandes avances tecnológicos al manejo de sus almacenes, ya que desde sus inicios la empresa ha invertido en bodegas automatizadas. En el año 2012 Amazon adquirió una compañía de robots pequeña, llamada Kiva, introduciendo en sus bodegas el uso de robots. Con la adquisición de robots, acrecentó la innovación de la empresa y del sector e-commerce, como dice Bogue (2016), pero también generó el desafío de planificar el movimiento de robots en sus almacenes. Para el problema de planificación de movimientos de los robots, en específico, se han creado varias soluciones, como Xiao-Long et al. (2017), que propone un movimiento de robots basados en RFID y códigos QR en las bodegas.

Lo nombrado anteriormente es sólo un caso particular de cómo ha evolucionado el mundo tecnológico. Otras empresas necesitan la planificación de sus propios robots, camiones en sus bodegas o movimiento de almacenaje en general, puesto que buscan encontrar el mejor camino para cada uno de sus agentes, ya sean robots, camiones, u otros, y así, ahorrar tiempo y energía en sus operaciones. De esta forma, en conjunto con una tendencia a la automatización de tareas en los tiempos actuales, este problema se volvió muy relevante, puesto que se puede ahorrar mucho dinero al optimizar estas operaciones de manera automatizada. Una abstracción del problema real es "Multi Agent Path Finding", que busca encontrar caminos óptimos para un conjunto de agentes, mientras que se respetan una serie de restricciones.

El problema de Multi Agent Path Finding (MAPF) consiste en encontrar el camino

de un punto de inicio a uno final para múltiples agentes, optimizando el costo sumado de todos los caminos, con la condición de que no pueden estar dos agentes en el mismo lugar ni intercambiar sus posiciones. Este problema no es trivial, puesto que su solución puede ser muy costosa en tiempo si se necesita un resultado óptimo. En efecto, este problema es catalogado como NP-Hard (Yu and LaValle, 2013). Las restricciones pueden variar en cada trabajo dependiendo cómo se aborde el problema, algunos problemas de MAPF permiten la superposición de dos agentes o que puedan intercambiar posiciones, pero, en el presente trabajo se seguirán las condiciones mencionadas en un principio.

Para resolver estos problemas existen varios algoritmos que permiten encontrar soluciones. Algunos de estos son capaces de encontrar una solución óptima al encontrar el menor costo en el movimiento de sus agentes, con costo de tiempo exponencial, lo cual puede ser muy costoso en instancias medianas a grandes, pero existen otras estrategias, basadas en heurísticas, que son capaces de generar soluciones suficientemente buenas para ser consideradas válidas, con un costo menor. Esto se asocia al teorema No Free Lunch (Wolpert and Macready, 1997), que dice que para cada par de algoritmos, hay tantos problemas en el que el primer algoritmo es mejor, como problemas en el que el segundo algoritmo es mejor, llegando a la conclusión de que no hay un algoritmo que sea mejor universalmente.

En los últimos años se han desarrollado varios trabajos que proponen utilizar Machine Learning para determinar el mejor algoritmo, de entre un conjunto de ellos, para resolver un problema. En estos trabajos pueden o no considerar el tiempo que se demora un algoritmo en resolver el problema, dichos algoritmos son acotados a un problema en específico, como Knapsack, Travelling Salesman Problem, entre otros. Existen ejemplos de lo anterior, como Huerta et al. (2020), en el cual los autores desarrollaron un método de selección automática de algoritmos para el problema de Knapsack con Machine Learning tomando en cuenta el tiempo de ejecución.

En este trabajo se creó un modelo de Machine Learning capaz de seleccionar automáticamente un algoritmo específico de MAPF para cada situación, a partir de un conjunto de algoritmos, dependiendo de diversos factores como la densidad del espacio, el tamaño del mapa, tiempo de respuesta del algoritmo, entre otros. De esta forma, se busca generar un *recomendador* capaz de identificar la herramienta más adecuada para cada situación.

Se utilizaron métricas de clasificación para medir el rendimiento de los algoritmos basado en el Single Best Solver (SBS), que es el algoritmo con mejor rendimiento en todos los problemas y el Virtual Best Solver (VBS), que es la selección perfecta de algoritmo para cada problema con su respectivo tiempo. (Lindauer et al., 2019).

1.2. Definición del problema

Actualmente, existe una amplia gama de solvers diseñados para abordar problemas de Multi Agent Path Finding (MAPF). Sin embargo, la elección óptima del solver puede variar según las características particulares de la instancia del problema a resolver. Esta variabilidad presenta una dificultad significativa: la incertidumbre en la elección del algoritmo más adecuado para una instancia específica. La problemática central que este trabajo busca resolver es justamente esta incertidumbre.

El objetivo principal del presente estudio consiste en abordar el problema generar un metasolver basado en selección de algoritmos anytime para Multi Agent Path Finding (MAPF) mediante la implementación de un modelo de Machine Learning. Este modelo busca identificar el algoritmo más apropiado para cada instancia específica, en función del tiempo límite de ejecución establecido por el usuario.

Para alcanzar este objetivo, se ha desarrollado un modelo de Machine Learning que acepta como entrada una caracterización de la instancia y el tiempo límite establecido, generando como resultado el solver más adecuado para esa situación particular. Es importante destacar que los algoritmos empleados poseen características anytime, lo que significa que son capaces de proporcionar soluciones antes de completar totalmente el proceso, aunque también se consideran algoritmos exactos en la selección.

Asimismo, se han presentado instancias para la creación de un conjunto de datos que será utilizado en el entrenamiento del modelo de Machine Learning. Dicho conjunto de datos incluye instancias de MAPF resueltas por cada algoritmo, acompañadas de las características más relevantes de cada instancia.

La salida del modelo corresponde al algoritmo óptimo para resolver la instancia dentro del tiempo especificado, es decir, el algoritmo que presenta el menor costo obtenido en ese intervalo de tiempo. El costo se asocia con la suma de los pasos dados por cada agente, considerando que permanecer en el mismo lugar cuenta como un paso. Dicho costo se traduce en la suma de los tiempos requeridos por cada agente para alcanzar su destino.

1.3. Hipótesis

La hipótesis propuesta en este estudio sugiere que es factible desarrollar un metasolver para Multi Agent Path Finding (MAPF) fundamentado en un modelo de Machine Learning, capaz de mejorar el desempeño del Single Best Solver y aproximarse al rendimiento del Virtual Best Solver, tomando en cuenta los tiempos de entrada establecidos por el usuario.

Se anticipa que el modelo de Machine Learning diseñado permitirá identificar el algoritmo más conveniente para cada instancia específica, en función del tiempo límite de ejecución fijado por el usuario, consiguiendo de esta manera superar el rendimiento del Single Best Solver.

Asimismo, se espera que el modelo de Machine Learning propuesto posibilite aproximarse al desempeño del Virtual Best Solver, que representa el mejor algoritmo posible para solucionar cualquier instancia de MAPF. Aunque es improbable que el modelo de Machine Learning alcance el rendimiento del Virtual Best Solver, se espera que se aproxime lo suficiente como para brindar soluciones precisas y eficientes en una amplia variedad de instancias de MAPF.

1.4. Objetivos

1.4.1. Objetivo General

El objetivo general de este estudio consiste en diseñar un modelo de Machine Learning que permita seleccionar un solver para un problema de Multi Agent Path Finding (MAPF) que se aproxime al rendimiento del Virtual Best Solver. Para ello, se tendrán en cuenta las características particulares del problema y el límite de tiempo proporcionado por el usuario.

1.4.2. Objetivos Específicos

1. Recopilar un conjunto de datos amplio y variado, con al menos 10,000 instancias que incluyan problemas generados aleatoriamente y/o problemas reales, con el fin de obtener una representación completa del espacio de instancias.
2. Seleccionar y modificar algoritmos del estado del arte que puedan manejar eficientemente el tiempo límite y que ofrezcan una buena estrategia de búsqueda para resolver problemas de MAPF, y que puedan ejecutarse de manera incremental para aprovechar al máximo el tiempo disponible.
3. Ejecutar cada algoritmo en todas las instancias del conjunto de datos recopilado, registrando el tiempo de ejecución y el rendimiento en términos de calidad de la solución.
4. Identificar características relevantes que puedan utilizarse para predecir el rendimiento de un solver en una instancia específica de un problema de MAPF, incluyendo el tamaño del mapa, la complejidad del problema, la densidad de los obstáculos, etc.

5. Utilizar los datos recopilados y las características relevantes identificadas para construir un modelo de Machine Learning preciso y robusto que pueda predecir el rendimiento de un solver en una instancia específica de un problema de MAPF, y seleccionar el solver más cercano al virtual best solver para esa instancia.
6. Evaluar el modelo de Machine Learning utilizando benchmarks conocidos y generados, y comparar su rendimiento con otros métodos del estado del arte para la selección de solvers en problemas de MAPF.

1.5. Limitaciones

Una limitación significativa de este estudio radica en la disponibilidad restringida de solvers de vanguardia que no se encuentran accesibles al público. Para que estos algoritmos funcionen con un comportamiento anytime o limitar su tiempo de ejecución, es necesario adaptarlos con el fin de obtener su desempeño en un tiempo determinado. La falta de acceso a estos algoritmos puede reducir la capacidad de este estudio para cumplir con su objetivo, dado que no se contempla la creación o replicación de nuevos algoritmos.

Capítulo 2

Marco Teórico

2.1. Multi Agent Path Finding (MAPF)

El problema MAPF clásico se define por los siguientes elementos:

- Un conjunto de agentes $A = \{1, 2, ..., k\}$.
- Un grafo no dirigido $G = (V, E)$, donde V es el conjunto de nodos y E es el conjunto de aristas. Los nodos representan las posibles posiciones de los agentes y las aristas representan las conexiones posibles entre estas posiciones.
- Una función de mapeo $s : A \to V$ que asigna a cada agente su posición inicial.
- Una función de mapeo $t : A \to V$ que asigna a cada agente su posición objetivo.

Se asume que el tiempo es discreto y que en cada paso de tiempo, cada agente puede realizar una acción. Las acciones posibles son: una acción de espera, en la que el agente permanece en su nodo actual, y una acción de movimiento, que permite al agente moverse a un nodo adyacente. Formalizamos una acción como una función $a : V \to V$, donde $a(v) = v'$ representa la acción de moverse desde el nodo v a v' si v' es adyacente a v y es diferente de v, o la acción de permanecer en el nodo v si $v = v'$.

Los agentes ejecutan secuencias de acciones, denotadas como $\pi_i = (a_1, a_2, ..., a_n)$, para moverse desde su posición inicial hasta su posición objetivo. Dicha secuencia de acciones se conoce como plan. Si el agente i comienza en la posición $s(i)$ y llega a la posición objetivo $t(i)$ ejecutando el plan π_i, entonces π_i es un plan individual para el agente i. Una solución válida para el problema MAPF es un conjunto de k planes individuales, uno para cada agente, de tal manera que no existan colisiones

entre los planes. Una vez que un agente ha alcanzado su objetivo, puede permanecer en esa posición o desaparecer.

Para que una solución al problema MAPF sea válida, es necesario que los planes individuales de los k agentes no entren en conflicto entre sí. Para un plan dado π_i, la expresión $\pi_i[x]$ denota la posición del agente i después de haber realizado x pasos de su plan π_i. Identificamos cinco tipos de conflictos posibles entre dos planes π_i y π_j:

- **Conflicto de vértices**: Ocurre cuando los dos agentes ocupan el mismo nodo al mismo tiempo, es decir, $\pi_i[x] = \pi_j[x]$.
- **Conflicto de aristas**: Ocurre cuando dos agentes cruzan la misma arista en la misma dirección y al mismo tiempo, es decir, $\pi_i[x] = \pi_j[x]$ y $\pi_i[x+1] = \pi_j[x+1]$.
- **Conflicto de seguimiento**: Se presenta cuando un agente ocupa un nodo que otro agente ocupaba en el paso de tiempo anterior, es decir, $\pi_i[x+1] = \pi_j[x]$.
- **Conflicto de ciclo**: Sucede cuando un conjunto de agentes (al menos tres) se mueven en un ciclo, es decir, cada agente ocupa la posición que otro agente ocupaba un paso de tiempo atrás en el ciclo.
- **Conflicto de intercambio**: Se da cuando dos agentes intercambian sus posiciones, cruzando la misma arista en direcciones opuestas al mismo tiempo, es decir, $\pi_i[x+1] = \pi_j[x]$ y $\pi_j[x+1] = \pi_i[x]$.

En la formulación del problema MAPF, se puede establecer cuáles conflictos están permitidos y cuáles están prohibidos. No existe un estándar unificado respecto a qué conflictos deben ser permitidos y cuáles prohibidos, aunque usualmente los conflictos de vértices y aristas no son permitidos.

Para el presente trabajo, los planes individuales para cada agente deben ser formulados y ejecutados de tal manera que se eviten tanto los conflictos de vértices como de aristas, así como los conflictos de intercambio. Cabe mencionar, que no se consideran los conflictos de seguimiento ni de ciclo.

Cuando se calculan los planes individuales, el objetivo es minimizar una función objetivo definida por el usuario. No existe una función objetivo estándar a adoptar, sin embargo, las más comunes son:

- **Flowtime**: Esta medida se obtiene sumando los pasos de tiempo empleados por cada agente para llegar a su ubicación objetivo. Formalmente, es igual a $\sum_{i \in A} |\pi_i|$, donde los planes π_i, $i \in A$ son planes individuales sin colisiones.
- **Makespan**: El número de pasos de tiempo necesarios para que todos los agentes completen sus tareas, definido como $\max_{i \in A} |\pi_i|$, donde π_i, $i \in A$ forman

parte de una solución válida.

- **Maximización de objetivos alcanzados dado un plazo límite**: El objetivo es encontrar una solución válida que maximice el número de agentes que llegan a su objetivo dentro de un plazo de tiempo dado.

Existe una amplia literatura relacionada con el problema MAPF, el cual presenta diversas formulaciones. Stern et al. (2019b) definen el MAPF como el problema fundamental de planificar rutas para múltiples agentes, cuya principal restricción es evitar colisiones entre ellos. Además, se examinan sus variantes y se resumen algunos benchmarks con el objetivo de unificar el concepto de MAPF.

Los resultados de estos benchmarks fueron medidos utilizando el algoritmo ICBS (Improved Conflict-Based Search) propuesto por Boyarski et al. (2015), con el fin de establecer una base para futuras comparaciones con otros algoritmos. Cabe destacar que el algoritmo ICBS pudo resolver varias instancias presentadas en los benchmarks, pero no todas. Esto sugiere que las instancias propuestas son lo suficientemente complejas como para ser abordadas por algoritmos con enfoques distintos.

2.2. Machine Learning

El aprendizaje automático (Machine Learning, ML), un subcampo esencial de la inteligencia artificial, se basa en la explotación de técnicas estadísticas para permitir a las máquinas aprender a partir de datos sin requerir una programación explícita (Mitchell, 1997). Concretamente, los algoritmos de aprendizaje automático entrenan a los sistemas para realizar predicciones o tomar decisiones basándose en patrones reconocidos en grandes conjuntos de datos.

Los algoritmos de ML se agrupan principalmente en tres categorías: supervisados, no supervisados y de refuerzo. En el *aprendizaje supervisado*, los algoritmos se capacitan utilizando datos previamente etiquetados. Durante este proceso, los algoritmos aprenden a inferir una función a partir de los datos de entrada y salida proporcionados en el entrenamiento. Posteriormente, esta función puede emplearse para predecir la salida correspondiente a nuevas entradas. Los modelos de clasificación y regresión son ejemplos representativos de este tipo de aprendizaje.

En el caso de un *modelo de clasificación*, el aprendizaje se realiza a partir de datos de entrada y salida, y tras un entrenamiento efectivo, el modelo resultante puede predecir la clase (etiqueta o categoría) de datos nuevos. Este proceso se puede formalizar como una función $f : X \to Y$, donde X representa el espacio de entrada y Y el conjunto de clases. Los modelos de clasificación se evalúan comúnmente mediante métricas como precisión, exhaustividad, F1-score y la curva ROC.

Por otro lado, un *modelo de regresión* es una técnica de aprendizaje supervisado que predice un valor numérico continuo en lugar de una clase. Esto puede representarse formalmente como una función $f : X \to \mathbb{R}$, donde X es el espacio de entrada e $\mathbb{R}$ representa el conjunto de números reales. Los modelos de regresión se evalúan utilizando métricas como el error cuadrático medio (MSE), la raíz del error cuadrático medio (RMSE), el error absoluto medio (MAE) y el coeficiente de determinación (R^2).

En contraste, el *aprendizaje no supervisado* se basa en la identificación de patrones y relaciones en un conjunto de datos sin etiquetas o categorías predefinidas. El *aprendizaje por refuerzo*, por otro lado, involucra a un agente que aprende a través de la interacción con su entorno, realizando acciones y observando los resultados, con el objetivo de maximizar algún tipo de recompensa acumulativa.

El aprendizaje profundo (Deep Learning, DL), una subcategoría del ML, implica la construcción de redes neuronales artificiales que emulan la operación del cerebro humano (LeCun et al., 2015). Estas redes, compuestas por capas de nodos interconectados, pueden aprender a representar los datos de entrada mediante la extracción automática de características, proporcionando así un alto grado de flexibilidad y adaptabilidad en tareas de predicción y clasificación.

2.3. Algoritmos de ML

2.3.1. Red Neuronal Convolucional

Las Redes Neuronales Convolucionales (CNN, por sus siglas en inglés) son una categoría especializada de redes neuronales diseñadas para reconocer patrones en conjuntos de datos y son particularmente efectivas para tareas relacionadas con el procesamiento de imágenes. Una CNN es capaz de aprender automáticamente y de forma adaptativa características espaciales jerárquicas a partir de imágenes. Esto se logra mediante el uso de capas convolucionales, que, a diferencia de las capas densas o completamente conectadas, mantienen la estructura espacial de los datos.

Uno de los modelos de CNN más destacados es AlexNet, introducido por Alex Krizhevsky en 2012 (Krizhevsky et al., 2012). Esta red jugó un papel crucial en el resurgimiento del interés por las redes neuronales profundas al ganar el desafío ImageNet Large Scale Visual Recognition Challenge por un margen significativo.

La red AlexNet consta de ocho capas con pesos: las primeras cinco son convolucionales y las tres últimas son completamente conectadas, tal como se ilustra en la Figura 2. La salida de la última capa completamente conectada alimenta una función softmax de 1000 clases, generando una distribución sobre las 1000 etiquetas de clase.

Esta red tiene como objetivo optimizar la función de la regresión logística multinomial, que es equivalente a maximizar, en promedio sobre los casos de entrenamiento, el logaritmo de probabilidad de la etiqueta correcta conforme a la distribución de predicción.

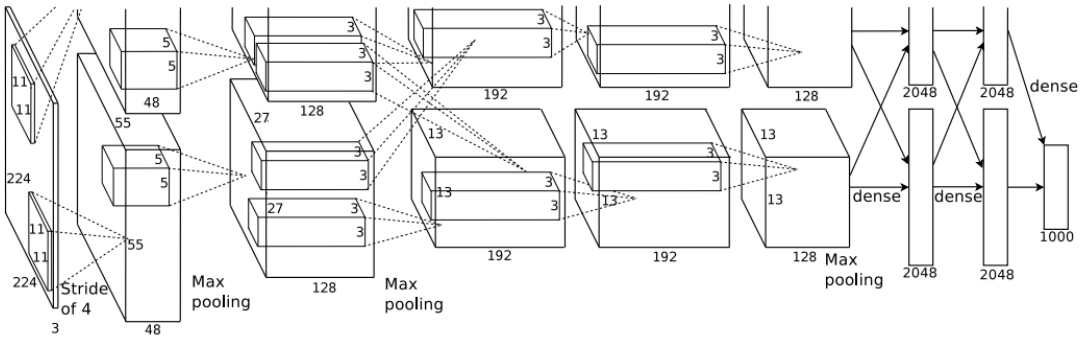


Figura 2.1: Diagrama de Modelo AlexNet Fuente: Krizhevsky et al. (2012)

Detallando las conexiones entre las capas: las capas convolucionales segunda, cuarta y quinta están conectadas únicamente a los mapas de kernel de la capa anterior que están en la misma GPU. La tercera capa convolucional se conecta a todos los mapas de kernel de la segunda capa. Las neuronas en las capas completamente conectadas están interconectadas con todas las neuronas de la capa anterior.

Las capas de normalización de respuesta se colocan después de las primeras y segundas capas convolucionales. Tras estas, y también tras la quinta capa convolucional, se encuentran las capas de pooling máximo. Se emplea la función no-lineal ReLU tras cada capa convolucional y completamente conectada.

En cuanto a las especificaciones de las capas, la primera capa convolucional procesa una imagen de entrada de $224 \times 224 \times 3$ usando 96 kernels de tamaño $11 \times 11 \times 3$ con un stride de 4 píxeles. La segunda capa convolucional utiliza 256 kernels de $5 \times 5 \times 48$. La tercera capa usa 384 kernels de $3 \times 3 \times 256$, la cuarta cuenta con 384 kernels de $3 \times 3 \times 192$, y la quinta tiene 256 kernels de $3 \times 3 \times 192$. Finalmente, las capas completamente conectadas están compuestas por 4096 neuronas cada una.

Métricas de Evaluación para una CNN

Para valorar de forma adecuada la eficacia de una Red Neuronal Convolucional (CNN), es esencial comprender las métricas de desempeño utilizadas. A continuación, se detallan las métricas mencionadas:

Precisión (Accuracy): Esta métrica refleja la proporción de predicciones correc-

tas con respecto al total de predicciones realizadas. Se define como:

$$\text{Accuracy} = \frac{\text{Número de predicciones correctas}}{\text{Número total de predicciones}}$$

Exhaustividad (Recall): También conocida como sensibilidad, mide la proporción de positivos reales que fueron identificados correctamente. Su fórmula es:

$$\text{Recall} = \frac{\text{Positivos Verdaderos (TP)}}{\text{Positivos Verdaderos (TP)} + \text{Falsos Negativos (FN)}}$$

Puntuación F1 (F1-score): Es una métrica que combina precisión y exhaustividad en una única cifra, ofreciendo un balance entre ambas. Se calcula como el promedio armónico entre precisión y exhaustividad:

$$F1 = 2 \times \frac{\text{Precisión} \times \text{Exhaustividad}}{\text{Precisión} + \text{Exhaustividad}}$$

Área bajo la curva ROC (AUC-ROC): Esta métrica proporciona una visión integral del desempeño del modelo en todos los umbrales de clasificación posibles. La curva ROC representa la tasa de verdaderos positivos en función de la tasa de falsos positivos. El AUC, o el área bajo esta curva, da una idea general de la capacidad del modelo para distinguir entre clases. Un AUC de 1 indica una clasificación perfecta, mientras que un AUC de 0.5 sugiere que el modelo no tiene capacidad discriminativa.

2.3.2. Modelos de Machine Learning Clásicos

Dentro del ámbito del Machine Learning, existen numerosos modelos que no dependen de arquitecturas de redes neuronales y que, a lo largo del tiempo, han demostrado ser altamente efectivos en una amplia gama de tareas. Estos modelos, a menudo referidos como métodos "clásicos" o "tradicionales", se han fundamentado en décadas de investigación y desarrollo. Su éxito radica en su simplicidad relativa, interpretabilidad y en su capacidad para abordar problemas específicos de manera eficiente.

Por ejemplo, el *árbol de decisión* es un modelo gráfico que toma decisiones basadas en hacer preguntas sobre características de datos. Es especialmente útil para la clasificación y regresión, y su estructura visual facilita la interpretación de cómo se toman las decisiones.

El *algoritmo k-NN* (k vecinos más cercanos) es otro modelo clásico que clasifica una observación basándose en cómo están clasificadas sus vecinas más cercanas en

el conjunto de entrenamiento. Es especialmente útil para tareas que requieren una decisión basada en la proximidad o similitud.

Los *modelos lineales*, como la regresión lineal y la regresión logística, son herramientas poderosas para predecir una salida continua o binaria, respectivamente, basándose en una o más variables de entrada. Su principal fortaleza radica en la facilidad para interpretar la relación entre las variables.

Además de estos, hay muchos otros modelos clásicos, como máquinas de soporte vectorial (SVM), métodos de ensamble como el bosque aleatorio, y técnicas de clustering como el k-means, que han sido esenciales en la evolución del campo del Machine Learning. Estos modelos, aunque a veces pueden ser superados en precisión por las redes neuronales en ciertas tareas, todavía son preferidos en situaciones donde la interpretabilidad, la eficiencia computacional o la simplicidad del modelo son críticas.

Evaluación de Modelos de Machine Learning Clásicos

La evaluación de modelos de Machine Learning clásicos es crucial para determinar su eficacia y precisión en tareas específicas. Estas evaluaciones se basan en métricas que proporcionan una cuantificación objetiva del rendimiento del modelo.

Log Loss (Pérdida Logarítmica): Esta métrica evalúa las probabilidades predichas frente a las etiquetas verdaderas y es especialmente relevante para problemas de clasificación, ya sea binaria o multiclase. Una pérdida logarítmica de 0 indica una predicción perfecta, mientras que una pérdida mayor sugiere una predicción menos precisa. Para problemas de clasificación multiclase, la pérdida logarítmica se define generalmente como:

$$-\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} \cdot \log(p_{ij})$$

donde N es el número total de observaciones, M es el número de clases, y_{ij} es 1 si la observación i pertenece a la clase j y 0 en caso contrario, y p_{ij} es la probabilidad predicha de que la observación i pertenezca a la clase j.

Classification Error (Error de Clasificación): Es la proporción de predicciones incorrectas sobre el total de predicciones realizadas. Es una métrica simple pero efectiva para evaluar la exactitud de un modelo de clasificación.

$$\text{Error} = \frac{\text{Número de predicciones incorrectas}}{\text{Total de predicciones}}$$

Ambas métricas son fundamentales cuando se trabaja con modelos clásicos, ya que proporcionan una visión clara de su capacidad para clasificar correctamente nuevas observaciones y de la confiabilidad de esas clasificaciones.

XGBoost

XGBoost, que significa *Extreme Gradient Boosting*,(Chen and Guestrin, 2016) es una implementación optimizada del algoritmo de potenciación del gradiente (Gradient Boosting Decision Tree, GBDT (Friedman, 2001)). Se ha vuelto particularmente popular en el campo debido a su eficiencia y rendimiento en competencias de Kaggle. El XGBoost utiliza un modelo de árbol de decisión ensamblado y es capaz de realizar la regularización L1 y L2, lo que lo hace robusto frente al sobreajuste.

Las características distintivas del XGBoost incluyen:

- Procesamiento en paralelo, que lo hace más rápido que otros algoritmos de potenciación.
- Capacidad para manejar valores faltantes.
- Flexibilidad para definir funciones de optimización personalizadas y criterios de evaluación.

2.4. Algorithm Selection

En la literatura, la selección de algoritmos ha sido ampliamente estudiada. Uno de los trabajos seminales en este campo es el propuesto por Rice (1976), titulado "The Algorithm Selection Problem". Rice introdujo un esquema innovador que va más allá de considerar simplemente las características del problema y los algoritmos disponibles. Introdujo un criterio de rendimiento variable, que cambió fundamentalmente la forma en que se aborda la selección de algoritmos.

2.4.1. Descripción del Esquema

El enfoque de Rice se puede conceptualizar como un mapeo tridimensional. En lugar de relacionar solo las características del problema y el algoritmo, Rice también incorporó un criterio de rendimiento variable. Formalmente, el esquema propuesto se puede describir como sigue:

$$S : P \times F \times W \rightarrow A \tag{2.1}$$

Donde:

- S es el esquema de selección.
- P representa el espacio de posibles instancias del problema.
- F corresponde a un conjunto de características extraídas de la instancia.
- W denota el criterio de rendimiento, una función que mide el rendimiento de un algoritmo para una instancia específica.
- A es el conjunto de algoritmos disponibles.

En este contexto, el criterio W se convierte en una dimensión esencial, ya que permite que la elección del algoritmo sea context-dependiente. Es decir, un algoritmo que es óptimo según un criterio puede no serlo según otro.

2.4.2. Impacto y Relevancia

La propuesta de Rice fue revolucionaria en el sentido de que reconoció que la definición de "mejor rendimiento" puede variar según el contexto. Por lo tanto, el algoritmo "óptimo" no siempre es una elección clara basada únicamente en las características del problema, sino que el criterio de rendimiento también es determinante.

La Figura 2.2 proporciona una representación gráfica del esquema descrito por Rice.

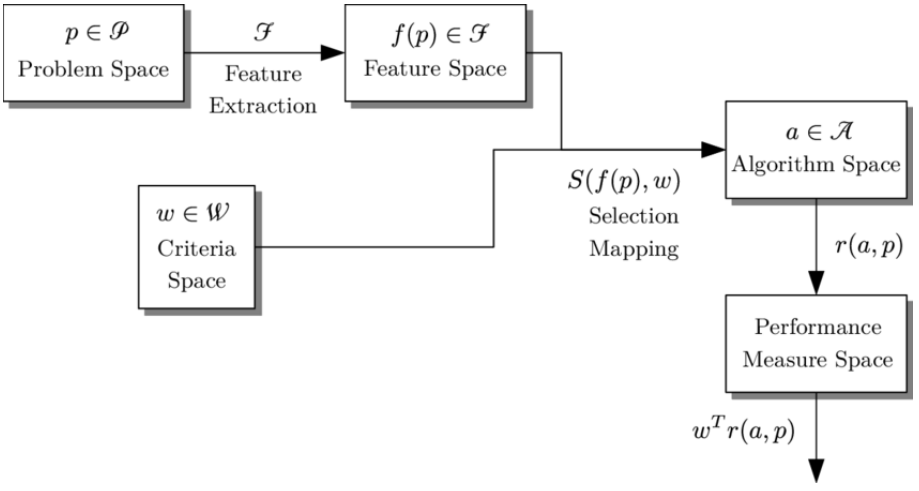


Figura 2.2: Diagrama del esquema de selección de algoritmos propuesto por Rice.

Este enfoque multifacético permitió el desarrollo de técnicas avanzadas y sistemas de aprendizaje automático que pueden tomar decisiones informadas al seleccionar el algoritmo más adecuado para una tarea en particular.

Capítulo 3

Revisión Bibliográfica

3.1. Algoritmos de MAPF

En la literatura se presentan distintas propuestas de solución para MAPF. En las propuestas existen algoritmos óptimos, asegurando que al terminar dicho algoritmo, se encuentra una solución óptima, y por otra parte se encuentran algoritmos heurísticos, los cuales generan soluciones con menor costo en tiempo que algoritmos óptimos, pero no aseguran optimalidad de la solución.

Por parte de los algoritmos óptimos, se tienen trabajos como Surynek (2021), en el cual se presenta una compilación de algoritmos basados en distintas compilaciones del problema en otros lenguajes/problemas como Boolean Satisfiability (SAT), Mixed Integer Linear Programming (MIP), Answer Set Programming (ASP) y Constraint Satisfaction (CSP), para MAPF. Además, en este trabajo se realizó un resumen, análisis y discusiones de algoritmos basados en MIP y SAT, para comparar distintos modelamientos. Se analizan y comparan los códigos de cada uno, además, se introducen versiones lazy para cada tipo de algoritmo, que los autores lo definen como un algoritmo que es fácil de programar, tiene buen rendimiento y está abierto a varias mejoras debido a heurísticas. Estas versiones generalizan mejor cada tipo de algoritmo y son capaces de ser comparados de una forma más fácil.

En el mismo grupo de algoritmos esta Conflict-Based Search (CBS, Sharon et al. 2015). En Li et al. (2019) se analiza CBS, en el cual los autores dicen que CBS es uno de los algoritmos que mejoran sustancialmente al aplicarle mejoras con heurísticas. Experimentan con CBS, ICBS (Improved CBS) y CBSH (Conflict-Based Search with Heuristics), con las heurísticas CG (Conflict Graph), DG (Pairwise Dependency Graph) y WDG (Weighted Pairwise Dependency Graph), en las que demuestran que las heurísticas DG y WDG son mejores que la heurística CG en la mayoría de las

instancias analizadas. Una mejora a este algoritmo se tiene en el trabajo de Li et al. (2021b), en el que se presenta un algoritmo llamado CBSH2-RTC con mejoras de funcionamiento, que comparado con las versiones anteriores del mismo algoritmo, como CBSH2 y CBS Vanilla, demostró ser superior en tiempo y escalabilidad. CBSH2-RTC, se basa en la idea de que en MAPF, al ser un problema combinatorio, existe un fenómeno de parejas simétricas (pairwise symmetry) que ocurre cuando un agente tiene varios caminos prometedores para llegar a su destino, pero cuando se elige uno, resulta no ser válido porque ocurren colisiones. Los autores proponen varios métodos para detectar dichas simetrías y acelerar el proceso del algoritmo de CBS para llegar a una solución óptima.

Un algoritmo posiblemente mejor a CBS, según sus autores, y estando en el mismo grupo de algoritmos, es el trabajo de Lam et al. (2022), presentándose y analizándose el solver Branch and Cut and Price (BCP). Se compara principalmente con algoritmos conocidos en el estado del arte como el Lazy CBS (Gange et al., 2019) y CBSH-RTC (Li et al., 2021b), demostrando ser mejor que estos, ya que, experimentalmente resuelve una mayor cantidad de instancias probadas. Los autores concluyen que BCP lidera a ambos algoritmos resolviendo 2402 instancias de 4430, mientras el Lazy CBS resuelve 2039 instancias y CBSH2-RTC resuelve 1937 instancias, siendo BCP un algoritmo anytime, lo que para el presente trabajo es un requisito deseable.

También entre los algoritmos óptimos, se tienen trabajos como Achá et al. (2021), en el que se presentan mejoras a algoritmos basados en ASP y SAT. En este trabajo se cambia la representación utilizada en versiones previas de dichos algoritmos, logrando mejoras en la escalabilidad de ambos algoritmos, concluyendo que el solver basado en ASP mejora en la escalabilidad cuando el número de agentes es aumentado con pocos obstáculos y el algoritmo basado en SAT mejora cuando los escenarios tienen más obstáculos y menos agentes.

Por otra parte, en algoritmos sub-óptimos se tienen trabajos como Li et al. (2021a), en el cual se presenta y analiza un algoritmo anytime llamado Large Neighborhood Search (LNS). Allí, se indica que este algoritmo encuentra una solución suboptima rápidamente, usando otros algoritmos, como EECBS (Explicit Estimation CBS, Li et al. 2021c), para luego tratar de mejorar la solución encontrada usando LNS. Esta forma demostró tener muy buenos resultados, ganando al estado del arte como EECBS, PP (Prioritized Planning, Erdmann and Lozano-Perez 1987), CBS o PPS (Parallel-Push-and-Swap, Sajid et al. (2012)), en escalabilidad, rapidez y el tiempo en encontrar una primera solución.

Con respecto a los algoritmos encontrados, para efectos de este trabajo se seleccionaron tanto algoritmos anytime, como algoritmos exactos, los cuales podrán ser adaptados para extraer comportamientos anytime. Así. se seleccionaron LNS

(Li et al., 2021a), BCP (Sigurdson et al., 2019), ASP (con una versión anytime) (Achá et al., 2021) y CBSH2 (Li et al., 2021b) en primera instancia.

3.2. Selección Automática de Algoritmos

En esta sección, se revisa el estado del arte en selección automática de algoritmos que buscan resolver otros problemas, para obtener ideas importantes de cómo abordar este proyecto.

En la literatura se encuentran trabajos como Huerta et al. (2020), que presenta un método de trabajo de selección automática de algoritmos aplicados al problema Knapsack, el cual corresponde a un problema NP-Hard. Los autores entrenan un modelo de Machine Learning que, dándole como input una serie de características, es capaz de seleccionar el mejor algoritmo con característica anytime que puede resolver una instancia del problema en un tiempo especificado. En este trabajo se presentan comparaciones de varios algoritmos de manera preliminar.

En relación a las comparaciones, se presenta un gráfico comparativo de cada algoritmo, de una manera bastante representativa, visible en la Figura 3.1. En esta figura se puede apreciar el porcentaje de optimalidad de cada solver (en una escala de 0 a 1, en la que 1 es óptimo), por el tiempo de ejecución de cada solver. Así, se ve qué algoritmo obtiene una mejor solución en cada instante de tiempo.

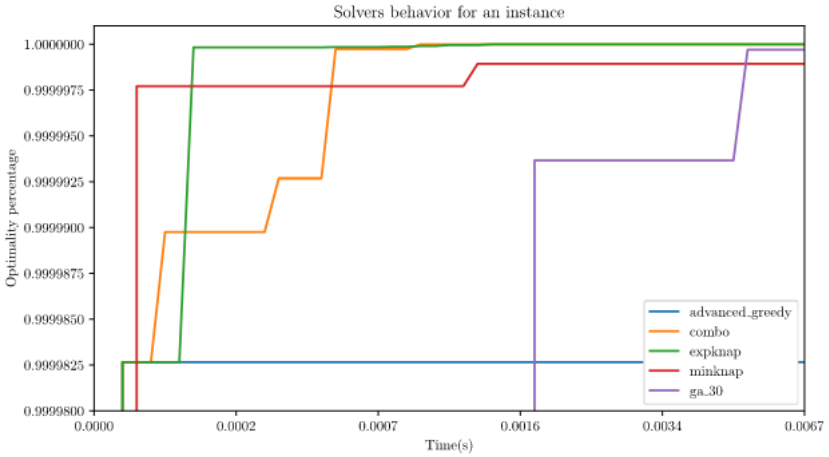


Figura 3.1: Comportamiento Anytime para solvers en la instancia con id 8047. Fuente: Huerta et al. (2020)

Igualmente, siguiendo con las comparaciones, se muestra una imagen con un gráfico general (Fig. 3.2) de cada solver aplicado a cada instancia que tienen en su dataset,

y se grafica el color del solver ganador en cada timestep. Así, se ve de una forma mas gráfica, cuál es el solver ganador y se puede comparar cada uno.

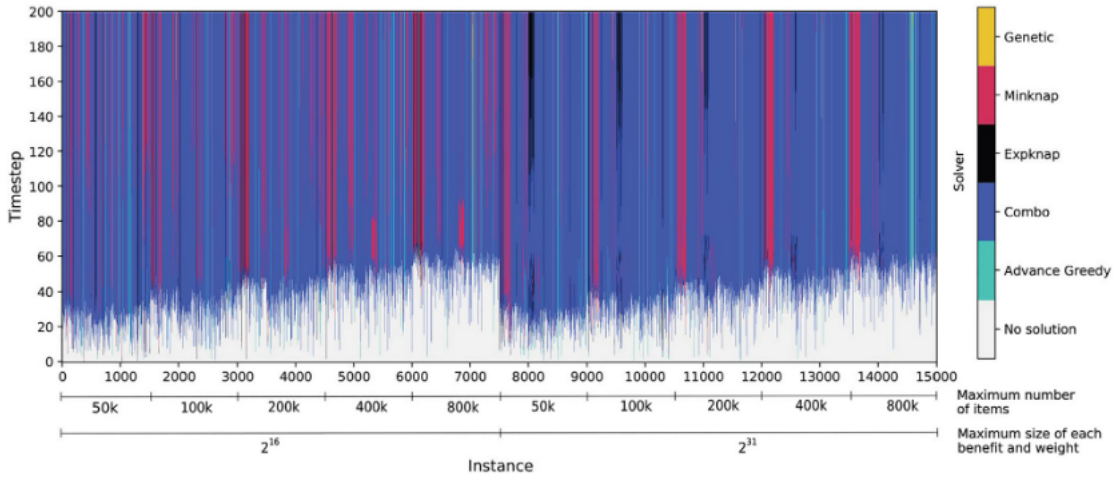


Figura 3.2: Mejor solver en cada timestep para todos los solvers. Fuente: Huerta et al. (2020)

En relación a su forma de selección de solver, fueron capaces de construir modelos de Machine Learning usando tres estrategias, dos de clasificación, una basada en el mejor solver en cada timestep, otra basada en un ranking de solvers en cada timestep; y una de regresión, basada en el porcentaje de optimalidad en cada timestep. En todos los modelos se concluye que en la mayoría de las instancias es posible llegar a la mejor solución.

Otro trabajo relevante en la literatura es Huerta et al. (2022), en el cual se mejora el estado del arte en la selección automática de algoritmos anytime para Traveling Salesman Problem. Este trabajo sigue una línea parecida a Huerta et al. (2020), pero con la introducción de nuevos métodos en los modelos de Machine Learning.

En relación a la selección de algoritmos, en este caso se usó una representación de una matriz para cada instancia, siendo capaces de usar una red neuronal convolucional para la búsqueda de características de cada instancia, esto aprovecha la característica de que las redes neuronales convolucionales funcionan mejor con representaciones de imágenes y son capaces de encontrar los comportamientos por sí mismas.

En el estado del arte se encuentra Bischl et al. (2016), el artículo aborda la creciente importancia del problema de selección de algoritmos en el campo de la Inteligencia Artificial (IA). Identifica una carencia en la comunidad: la ausencia de un formato estandarizado o repositorio para los datos obtenidos de la aplicación de

algoritmos en diferentes dominios. Este problema limita la capacidad de compartir y comparar eficazmente diferentes enfoques y representa una barrera para los nuevos investigadores. Para solucionar este problema, los autores presentan un formato estandarizado para representar escenarios de selección de algoritmos y un repositorio con una cantidad creciente de conjuntos de datos extraídos de la literatura. Realizan un estudio que construye y evalúa modelos de selección de algoritmos, demostrando el potencial de su plataforma y subrayando la capacidad de la selección de algoritmos para mejorar significativamente el rendimiento en una variedad de problemas y algoritmos.

El trabajo nombrado anteriormente introduce ASlib, una biblioteca de benchmarks para la selección de algoritmos, un campo de investigación en rápido crecimiento con un impacto sustancial en varias subcomunidades de Inteligencia Artificial. ASlib facilita la investigación sobre métodos de selección de algoritmos al proporcionar un conjunto común de benchmarks y herramientas para trabajar con ellos. Similar a las competencias de solvers, permite una evaluación comparativa empírica de rendimiento basada en principios. Además, disminuye significativamente la barrera para los investigadores que desean trabajar en la selección de algoritmos, ya que cualquiera que use los escenarios de benchmark que proporcionamos no tiene que realizar ejecuciones reales de los solvers contenidos en ellos. La versión 2.0 de la biblioteca contiene 17 escenarios de selección de algoritmos de seis áreas diferentes, centrados principalmente en problemas de satisfacción de restricciones. Se presentan estudios exploratorios con diversas aproximaciones a la selección de algoritmos, demostrando el rendimiento que los sistemas de selección de algoritmos pueden alcanzar en nuestros escenarios.

En el estado del arte existen sistemas de selección de algoritmos como ASAP (Algorithm Selector And Prescheduler) (Gonard et al., 2019) y AutoFolio (Lindauer et al., 2015). Han demostrado su eficacia al mejorar la resolución de diversos problemas de IA, como SAT, CSP, ASP, MAXSAT y QBF. Ambos sistemas se adhieren a los estándares propuestos por ASlib, facilitando la comparación y el intercambio de datos entre investigadores.

ASAP se basa en la optimización conjunta de un pre-planificador y una selección de algoritmo por instancia, seleccionando un algoritmo bien adaptado a la instancia de problema en cuestión. Ha sido evaluado exhaustivamente contra el estado del arte en varias competencias, obteniendo notables resultados y reconocimientos.

Por otro lado, AutoFolio utiliza la configuración automática de algoritmos para seleccionar de manera efectiva el enfoque de selección de algoritmo y establecer sus parámetros. A través de AutoFolio, se puede explotar la potencia combinada de varios métodos de selección de algoritmos, logrando significativas mejoras de

rendimiento en una amplia gama de escenarios.

Se contempla la implementación del benchmark proporcionado por ASlib para realizar una comparación exhaustiva entre solvers existentes, como ASAP o AutoFolio, y el modelo propuesto en este estudio. Esto permitirá una evaluación contrastada y rigurosa de los resultados obtenidos.

3.3. Selección Automática de Algoritmos en MAPF

Es importante revisar el estado del arte en el campo de selección automática de algoritmos en el contexto actual de trabajo. En este ámbito, se tiene Sigurdson et al. (2019), el cual es un trabajo sobre la selección automática de algoritmos en Multi Agent Path Finding que trabaja con los algoritmos Windowed Hierarchical Cooperative A*, Flow Anotated Replannig y Bounded Multi Agent A*, todos basados en A*; con la premisa de que ningún algoritmo domina cada instancia de MAPF. Se presentan instancias usadas en el trabajo, una forma de selección de algoritmos basada en Deep Learning, con sus posteriores resultados.

Con respecto a las instancias, estas fueron basadas en videojuegos, intentando representar los escenarios usados en la mayoría de ellos, ya que, por ejemplo, es muy común los videojuegos que tratan de llegar a la base del jugador enemigo, pudiendo aplicar un problema de MAPF. Usaron 20 mapas basados en el videojuego Baldur's Gate, disponibles en MovingAI[1], agregando cerca de 50.000 estados distintos para cada mapa, cambiando el número de agentes y los estados de cada uno.

En relación a la selección de algoritmos, investigan el uso de Deep Learning para la selección automática de algoritmo en cada instancia. Usaron la arquitectura de red neuronal llamada AlexNet modificada para la selección automática de algoritmos, cambiando la capa de salida para hacerla coincidir con la cantidad de algoritmos en su portafolio.

Como resultados de la investigación, obtuvieron buenos resultados, acercándose bastante al mejor algoritmo posible, que obtuvo una precisión del 80 %, con un 78 % con su selección automática de algoritmos.

Como conclusión de Sigurdson et al. (2019), es que los autores realizan una selección automática de algoritmos para MAPF que puede ser usada como base de la presente investigación, principalmente se rescata el modelo de Machine Learning utilizado, las instancias y solvers. Se trabajará para mejorar estos resultados agregando una variedad mayor de instancias, considerando el comportamiento anytime de los algoritmos. Esto último es importante, debido a que el objetivo del trabajo

[1]https://movingai.com/benchmarks/mapf.html

es elegir un solver dependiendo de un tiempo límite, por lo que considerar preferentemente algoritmos anytime permite encontrar un algoritmo aunque el problema no esté resuelto a optimalidad, pero se obtiene un resultado que con una posterior evaluación, puede ser suficientemente bueno para el requerimiento.

A partir de la existencia de todas estas propuestas, en Lindauer et al. (2019), se realiza una comparación de métodos para la selección automática de algoritmos entre los años 2015 y 2017 y muestra cómo el estado del arte ha evolucionado a través del tiempo. Como se mencionó anteriormente, en este trabajo se define el Mejor Solver Individual (Single Best Solver, SBS) como el algoritmo individual que tiene el mejor rendimiento en todas las instancias de entrenamiento, que denota lo que se puede lograr sin selección automática de algoritmo y el Mejor Solver Virtual (Virtual Best Solver, VBS) que realiza decisiones perfectas en cada instancia y selecciona el mejor algoritmo en cada instancia. El VBS es el algoritmo que seleccionaría si se corren todos los algoritmos a la vez y se selecciona el que primero termine. De este trabajo se utilizarán los conceptos de SBS y VBS para medir y evaluar el rendimiento de cada resultado del modelo final.

También, en Kaduri et al. (2021), se presenta una recopilación de instancias y algoritmos de MAPF, y una selección automática de algoritmos basadas en distintas características de las instancias.

Con respecto a la recopilación de instancias y algoritmos, son usadas las instancias de MovingAI y presentan una extensión de esta basada en distintas distribuciones de agentes. También se realizó una evaluación a algoritmos exactos, específicamente de los algoritmos Enhanced Partial Expansion A* (EPEA*, Goldenberg et al. 2014), Icreasing Cost Tree Search (ICTS, Sharon et al. 2013), Conflict Based Search with Heuristics (CBS-H, Boyarski et al. 2015), Multi Decision Diagram SAT (MDD-SAT, Surynek et al. 2016), y Lazy Conflict Based Search (Lazy CBS, Gange et al. 2019) ejecutados a todo su dataset.

Con respecto a la selección automática de algoritmos, presentan distintas ideas interesantes a aplicar en el presente trabajo, los autores separan la selección de algoritmos en tres tipos, *in-grid* (Usan los mismos mapas para el entrenamiento y el testing, pero cambian la distribución de agentes), *in-grid-type* (Usan los mismos tipos de mapas para el entrenamiento y testing, pero cambian los mapas específicos) y *between-grids* (Usan mapas distintos para el entrenamiento y el testing), cambiando el comportamiento de selección de algoritmo. Esto puede ser usado para probar distintos comportamientos del modelo a utilizar en el presente trabajo y tener ideas de que características utilizaron para construir el modelo.

Finalmente, en Kaduri et al. (2020), se presenta una mejora en la selección de algoritmos óptimos para MAPF, en este artículo se comparan dos implementaciones

de esta selección de algoritmos automática, una usando Deep Learning y otra usando Extreme Gradient Boosting (XGBoost).

En relación al modelo de Deep Learning usado, es un modelo de redes convolucionales (CNN, en inglés), basada en una arquitectura VGG-16 (Simonyan and Zisserman, 2014), que es una arquitectura muy usada en reconocimiento de imágenes, modificada por los autores. Las imágenes de input de la red, son una sola imagen por instancia, siguiendo la siguiente representación, basada en Sigurdson et al. (2019). Se dibujaron las celdas bloqueadas y vacías con píxeles negros y blancos respectivamente. Puntos de origen y destino son representados con pixeles verdes y rojos respectivamente. Cambiándole el tamaño a todas las imágenes a 224x224x3 (224 de ancho, 224 de alto y 3 capas por RGB). Se realizaron modelos de clasificación y de regresión.

En relación al modelo de XGBoost, se utilizaron características extraídas a mano (handcrafted features, en inglés). Los autores utilizaron el número de filas y columnas de la instancia, número de celdas bloqueadas, ratio de celdas bloqueadas, número de agentes, el factor de división de ramas de un algoritmo A* naive, número de agentes dividido por las celdas no bloqueadas, y finalmente características obtenidas de la distancia entre el origen y destino de los agentes, estas son; la distancia promedio, máxima y mínima entre el número completo de agentes. Y al igual que en el modelo de Deep Learning se realizaron modelos de clasificación y regresión.

Como conclusión del trabajo previamente mencionado, el algoritmo que entrega mejor resultados, en términos de precisión es el XGBoost de clasificación, pero se cree que es porque no se usó un acercamiento tan bueno con las imágenes del modelo de CNN.

Capítulo 4

Diseño y Entrenamiento de un Metasolver Anytime para MAPF

4.1. Marco de trabajo

En esta sección se describirá el marco de trabajo a utilizar, explicando cómo se aborda el problema de ejecución del metasolver.

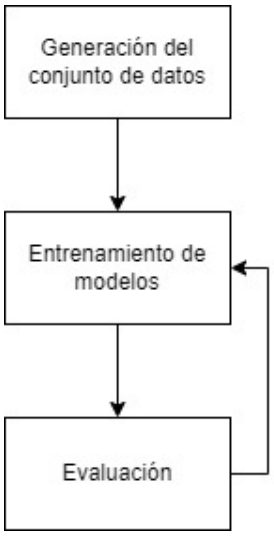


Figura 4.1: Diagrama del marco de trabajo

Como se ve en la figura 4.1, se inicia con la generación del conjunto de datos, que comprende una serie de instancias que cada uno de los solvers identificados deberá resolver.

Para este propósito, se empieza por seleccionar las instancias descritas en el estado del arte. Luego, estas instancias se someten a los modelos adaptados específicamente para entornos "anytime". Al final del proceso, se recopilan los datos correspondientes al costo de la solución proporcionada por cada uno de los solvers en cada instante específico de tiempo.

Después, considerando los resultados obtenidos a partir de la ejecución de los solvers, se procede a la creación de modelos de aprendizaje automático. La expectativa radica en que estos modelos sean capaces de interiorizar el comportamiento de los solvers en relación con las características particulares de las instancias. De esta manera, podrán prever cuál solver resultará ser el más eficaz en cada momento específico de tiempo.

Finalmente, para saber qué modelo es mas eficaz, se evaluarán los resultados de los experimentos utilizando la métrica $\hat{m}$, que se emplea en Lindauer et al. (2019) para comparar diferentes meta-modelos utilizados en investigaciones en el campo del aprendizaje automático y la inteligencia artificial. Esta métrica permite cuantificar la eficacia de los modelos en la predicción y selección de solvers más eficientes en diversos problemas. En el caso que esta evaluación no sea satisfactoria, se entrena otro modelo de ML.

Para calcular la métrica $\hat{m}$, se adaptará su cálculo a los resultados anytime de la siguiente manera:

- Cálculo general: Se tomará el Single Best Solver (SBS) general, que es el solver que presenta el mejor rendimiento en promedio a lo largo de todos los problemas, y se calculará el Virtual Best Solver (VBS) específico para cada timestep. El VBS representa el mejor solver hipotético que se obtendría al combinar las soluciones óptimas de todos los solvers disponibles en cada timestep.
- Cálculo por cada timestep: Se tomará un SBS específico para cada timestep y se calculará el VBS específico correspondiente. Esto permite analizar la evolución del rendimiento de los solvers en función del tiempo y determinar si ciertos solvers son más adecuados para problemas específicos en diferentes momentos.

Métrica general

Se comenzará evaluando la métrica de manera general, lo cual implica lo siguiente:

Dado un conjunto de instancias I y un conjunto de pasos temporales T, para un par de instancia/paso temporal $(i,t) \in I \times T$, $o_s(i,t)$ es el valor objetivo del solver s en la instancia i, en el paso temporal t. $n(o_s(i,t),i,t)$ es una función que normaliza $o_s(i,t)$ a algún rango uniforme definido, para cada par posible (i,t). m_s se define como:

$$m_s = \sum_{(i,t)\in I\times T} n(o_s(i,t), i, t)$$

y corresponde al desempeño normalizado acumulativo del solver s en todos los pares $(i,t) \in I \times T$.

La métrica $\hat{m}_s$ para el solver s se define como:

$$\hat{m}_{ms} = \frac{m_{ms} - m_{VBS}}{m_{SBS} - m_{VBS}}$$

donde:

- $\hat{m}_{ms}$ es la métrica de desempeño normalizada del metasolver ms.
- m_{ms} es el desempeño normalizado acumulativo del metasolver ms.
- m_{VBS} es el desempeño normalizado acumulativo del Virtual Best Solver.
- m_{SBS} es el desempeño normalizado acumulativo del Single Best Solver.

Observamos que:

- Cuanto más cerca esté $\hat{m}_{ms}$ de 0, más similar será el metasolver al Mejor Solver Virtual.
- Si $\hat{m}_{ms} > 1$, significa que el metasolver se comporta peor que el Mejor Solver Individual y, por lo tanto, no es útil.
- Se usa el SBS general de toda la ejecución, indistintamente del timestep.

Métrica por timestep

Dado un conjunto de instancias I y un conjunto de pasos temporales T, para un conjunto de instancias $(i) \in I$, $o_s(i,t)$ es el valor objetivo del solver s en la instancia i, en el paso temporal t. $n(o_s(i), i)$ es una función que normaliza $o_s(i)$ a algún rango uniforme definido, para cada i, ejecutandose para cada instante t. m_{st} se define como:

$$m_{st} = (\sum_{i\in I} n(o_s(i), i)) \times t$$

y corresponde al desempeño normalizado acumulativo del solver s en las instancias $i \in I$ para cada instante de tiempo t.

La métrica $\hat{m}_s$ para el solver s se define como:

$$\hat{m}_{mst} = \frac{m_{mst} - m_{VBSt}}{m_{SBSt} - m_{VBSt}}$$

donde:

- $\hat{m}_{mst}$ es la métrica de desempeño normalizada del metasolver ms en el tiempo t.

- m_{mst} es el desempeño normalizado acumulativo del metasolver ms en el tiempo t.
- m_{VBSt} es el desempeño normalizado acumulativo del Virtual Best Solver en el tiempo t.
- m_{SBSt} es el desempeño normalizado acumulativo del Single Best Solver en el tiempo t.

Observamos que:

- Cuanto más cerca esté $\hat{m}_{mst}$ de 0, más similar será el metasolver al Mejor Solver Virtual.
- Si $\hat{m}_{mst} > 1$, significa que el metasolver se comporta peor que el Mejor Solver Individual y, por lo tanto, no es útil.
- Se usa el SBS por timestep, lo que significa que es obtenido por cada instante de tiempo.

4.2. Recolección de Datos

En esta sección se describirán los experimentos realizados para completar los objetivos previamente mencionados. Esta sección consta del primer apartado del marco de trabajo.

4.2.1. Solvers

Se seleccionaron los siguientes solvers en el estado del arte.

- **Large Neighborhood Search, LNS** (Li et al., 2021a): Es una técnica heurística que busca mejorar soluciones iniciales mediante la exploración de grandes vecindarios dentro del espacio de soluciones. Para hacerlo, destruye parcialmente la solución inicial y reconstruye esa parte de la solución de manera óptima, teniendo en cuenta las restricciones del problema.
- **Branch and Cut and Price, BCP** (Sigurdson et al., 2019): Este algoritmo combina técnicas de programación lineal entera y programación dinámica para resolver problemas de optimización. A través de una serie de iteraciones de división (Branch), corte (Cut) y fijación de precios (Price), el BCP explora el espacio de soluciones hasta encontrar una solución óptima o hasta que se demuestre que no existe una solución factible.

- **Answer Set Programming, ASP** (con una versión anytime) (Achá et al., 2021): Es una forma de programación declarativa orientada a problemas difíciles, principalmente en el campo de la resolución de problemas no-monotónicos y la Inteligencia Artificial. La versión anytime se caracteriza por encontrar soluciones satisfactorias en un tiempo limitado, y mejorar estas soluciones si se dispone de más tiempo.
- **Conflict Based Search, CBSH2** (Li et al., 2021b): Es un algoritmo de dos niveles que resuelve el problema de MAPF encontrando primero una solución sin conflictos para cada agente de manera independiente, luego resuelve conflictos a medida que surgen hasta que se encuentra una solución sin conflictos. A este algoritmo se le aplican heuristicas para mejorar el rendimiento.
- **Large Neighborhood Search mejorado, LNS2** (Li et al., 2021a): Este es una mejora del algoritmo LNS, donde se realizan ajustes para mejorar la primera solución encontrada cambiado el algoritmo inicial, permitiendo que este algoritmo encuentre soluciones de alta calidad más rápidamente.
- **Explicit Estimation CBS, EECBS** (Li et al., 2021c): Es una extensión del CBS, donde además de resolver conflictos, se incluye una estimación explícita de los costos futuros para tomar decisiones más informadas durante la búsqueda.
- **Anytime EECBS** (Li et al., 2021a): Esta es una versión anytime del EECBS. Este solver es capaz de encontrar soluciones satisfactorias en un tiempo limitado y, si se le da más tiempo, mejorará la solución.
- **Anytime BCBS** (Li et al., 2021a): Este solver representa una versión del BCP que puede proporcionar soluciones útiles dentro de un límite de tiempo especificado y, si se permite más tiempo, puede continuar mejorando las soluciones obtenidas previamente.
- **MTMS** (Achá et al., 2021): Este es un algoritmo que combina múltiples técnicas, como la programación de restricciones y el aprendizaje automático, para resolver problemas de MAPF de manera eficiente y efectiva, ajustándose a las características particulares de cada problema.

4.2.2. Instancias

Se han seleccionado instancias provenientes de MovingAI (Stern et al., 2019a), con un enfoque especial en los benchmarks. Estas instancias son ampliamente reconocidas en investigaciones académicas y representan pruebas fundamentales en el estado del arte. La adaptación de cada tipo de instancia fue esencial para garantizar

su compatibilidad como entrada para los diversos algoritmos. La recopilación consistió en una diversidad de instancias, incluyendo tanto instancias aleatorias como basadas en almacenes.

La recolección total ascendió a 20035 instancias, desglosadas de la siguiente manera: 16475 pertenecen al benchmark de MovingAI, 2360 son instancias aleatorias y 1200 se basan en almacenes. Es relevante señalar que las instancias de MovingAI se derivaron de conjuntos originales, segmentando según la cantidad de agentes por instancia.

De la totalidad de instancias recopiladas, se lograron obtener resultados de 7784 de ellas. Este número reducido se debe a la restricción temporal impuesta de 1 hora por instancia, durante la cual algunas instancias más complejas no pudieron ser resueltas por el portafolio de solvers. Desglosando este subconjunto, encontramos que 4440 instancias provienen de MovingAI, 2144 son aleatorias y 1200 están basadas en almacenes. Esta distribución se puede atribuir al hecho de que las instancias aleatorias y las de almacenes suelen ser de tamaño mediano a pequeño, mientras que en MovingAI encontramos instancias de dimensiones más amplias.

4.2.3. Resultados de solvers en las instancias

Se procesaron diversas instancias utilizando cada algoritmo, logrando obtener resultados que incluyen medidas de tiempo y calidad de las soluciones. Estos resultados fueron fundamentales para realizar comparativas entre los distintos algoritmos, bajo una restricción de tiempo de una hora por ejecución, se puede ver en la figura 4.2. El procesamiento se llevó a cabo en el cluster Luthier de la Universidad de Concepción, este es un cluster manejado con Slurm[2], en el que cada nodo cuenta con un procesador Intel Xeon E3-1270 v6 3.80 GHz con 64 GB de RAM y Ubuntu 18.04, y se extendió por un período de un mes. A partir de esta información, se generó un conjunto de datos consolidado que engloba todas las instancias. Este conjunto de datos se estructuró para ser empleado en el entrenamiento de un modelo de aprendizaje automático, integrando características o *features* que permiten distinguir entre cada instancia y algoritmo.

4.3. Modelos de Machine Learning

Esta sección se dedica a describir el proceso de entrenamiento de los modelos de aprendizaje automático (ML). Los modelos mencionados se llevaron a cabo en una configuración de hardware que comprende un procesador AMD Ryzen 9 5900X, una

[2]https://slurm.schedmd.com/documentation.html

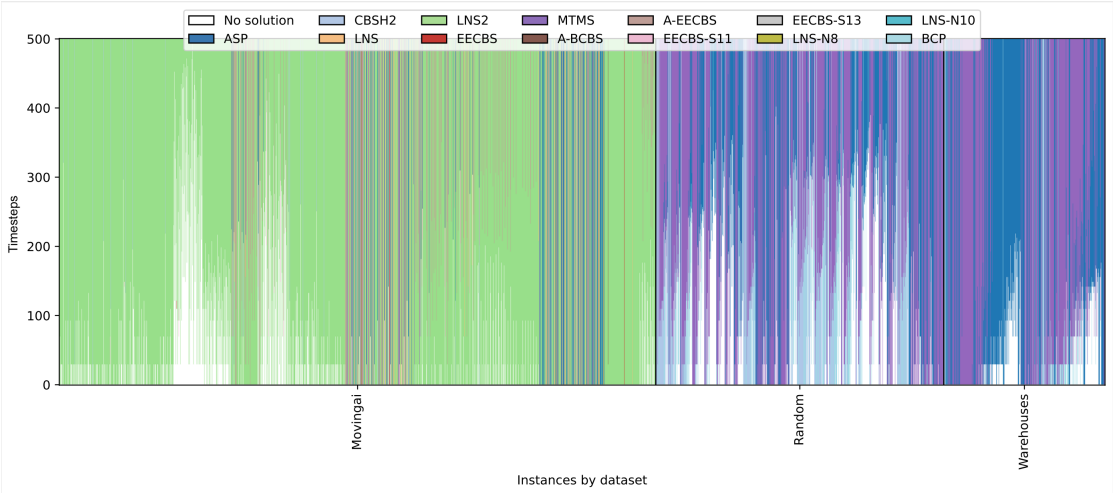


Figura 4.2: Ejecución de competencia de solvers, ejecutándose 7784 instancias en 13 solvers.

tarjeta gráfica Nvidia RTX 3090 y 32 GB de RAM, todos alojados en un computador personal. El modelo de redes neuronales se ejecutó durante un período de tiempo máximo de 5 días, mientras que el XGBoost se ejecutó por una duración de 1 hora.

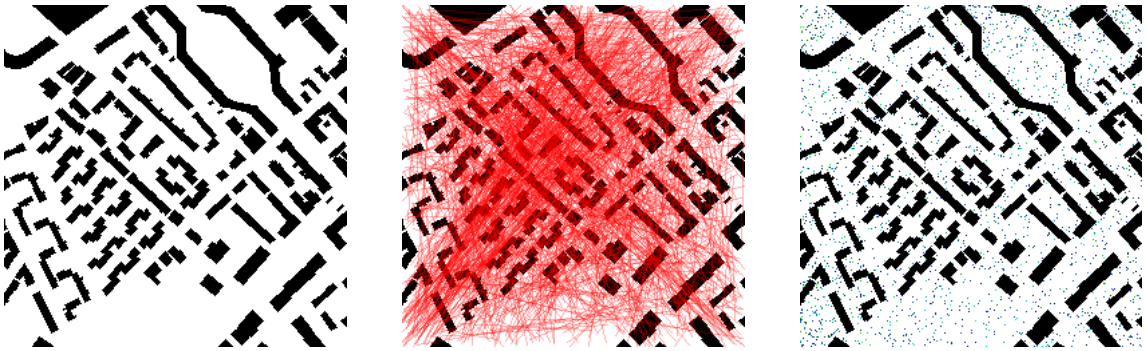
4.3.1. Modelo de Redes Neuronales

Acorde con la tendencia global de utilizar modelos de Deep Learning para predecir eventos, debido a sus excelentes resultados en diversas áreas de la ciencia, se decidió probar este enfoque en primera instancia, con la esperanza de obtener buenos resultados.

Se exploraron ideas para caracterizar las imágenes, proporcionando más información a la red neuronal además de los espacios vacíos y obstáculos (Figura 4.3a). Se consideró agregar los puntos de inicio y final de cada agente con un color diferente (Figura 4.3c), y trazar una línea recta desde el punto de inicio hasta el punto final (del origen al destino) (Figura 4.3b) de cada agente en la instancia a representar.

Al incluir estas características, se esperaba enriquecer la entrada de la red neuronal y obtener resultados más cercanos a los esperados.

En lugar de utilizar el enfoque común de proporcionar a la red solo una imagen con todas las características, se decidió construir una red neuronal con múltiples entradas, ya que la imagen resultante no era fácilmente comprensible a simple vista y podría confundir a la red neuronal. Cada entrada de la red recibió una imagen correspondiente a los obstáculos (Figura 4.3a), los puntos de inicio y final (Figura



(a) Imagen obstáculos. (b) Imagen lineas. (c) Imagen puntos.

Figura 4.3: (a) Imagen de los obstáculos y espacios vacíos correspondiente a cada pixel de la imagen. (b) Imagen de las lineas rectas conectando el inicio y final de cada agente. (c) Imagen con los puntos de inicio y final de cada agente.

4.3c) y las líneas rectas que conectan el origen y destino de cada agente (Figura 4.3b).

Para lograr esto, se utilizó como base un modelo de red AlexNet, cuyo diagrama se puede ver en la Figura 4.4. Como resultado, se obtuvo un vector con los ganadores de cada marca de tiempo (timestep).

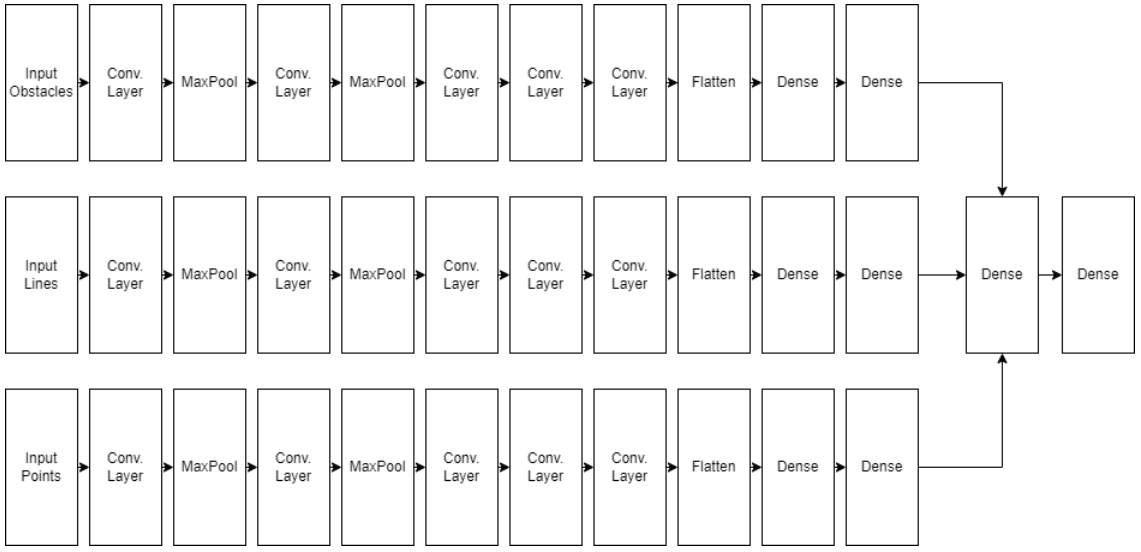


Figura 4.4: Diagrama redes neuronales con imágenes

Los resultados de estos experimentos no fueron los esperados, ya que la red neuronal no pudo entregar resultados coherentes ni aproximarse a los vectores de solu-

ciones esperados.

Continuando con la idea de las redes neuronales, se optó por un modelo que tuviera como entrada las imágenes previamente descritas junto con el timestep correspondiente al tiempo de ejecución que se asignaría a la ejecución de la instancia.

Este enfoque tenía un inconveniente: si se tenían 5574 instancias y cada una tenía su propio conjunto de tres imágenes, al multiplicarlo por cada timestep que se quisiera entrenar (500 timesteps), se tendrían 2,787,000 instancias de entrenamiento, con su respectivo conjunto de imágenes y su timestep correspondiente.

Toda esta información requiere un uso elevado de memoria principal para ejecutar todo el entrenamiento, por lo que era necesario encontrar una solución a este problema. Se creía que este enfoque sería más fácil de entender para una red neuronal, ya que solo se espera como resultado el solver ganador (uno de siete solvers).

Para solucionar la falta de memoria principal, se utilizaron conjuntos de datos generativos, los cuales permiten cargar en memoria principal solo una parte del conjunto de datos total. Esto hace que el proceso sea un poco más lento, pero posible de realizar con un computador personal.

Usando los conjuntos de datos generativos, la idea era la misma: crear una red neuronal que tenga como entrada las imágenes de los obstáculos, de los puntos de origen y destino de cada agente y las líneas que conectan estos puntos, añadiendo esta vez los timesteps para los cuales se desea conocer el solver ganador. Y como salida, el solver ganador para ese timestep.

Los resultados obtenidos en este experimento no fueron muy buenos, ya que la elección del solver era la misma para todas las instancias, siendo ASP, que correspondía al SBS. Este resultado no es suficiente, ya que se espera encontrar un modelo capaz de elegir en cada tiempo un solver mejor que el SBS, pero muy cercano al VBS.

Para mejorar este resultado, se implementó otra idea, suponiendo que el problema de la red neuronal eran las imágenes, al ser RGB y tener múltiples capas, lo que generaba un exceso de información, por lo tanto, se decidió transformar cada una de las imágenes en una representación matricial, lo que resultó en tres matrices distintas que representan diferentes aspectos del problema: una matriz para los obstáculos, otra para las líneas y, finalmente, una matriz para los puntos de origen y destino de cada agente.

La matriz de obstáculos se generó asignando un valor de 0 a los espacios vacíos y un valor de 1 a los espacios que contienen obstáculos. De esta manera, se logró una representación binaria simplificada del entorno.

En cuanto a la matriz de líneas, se trazaron líneas rectas entre los puntos de origen y destino, incrementando el valor en 10 por cada línea trazada. Esto permitió que los puntos por los que pasaban múltiples líneas tuvieran un valor más alto en comparación con aquellos puntos por los que pasaban menos líneas.

Por último, la matriz de puntos de origen y destino se creó asignando un número único a cada punto de origen y su correspondiente punto de destino como el negativo de ese número. Esta representación permitió una identificación clara de los agentes y sus objetivos.

El uso de estas representaciones matriciales permitió eliminar capas de convolución y reducir la complejidad general de la red neuronal. Al simplificar la entrada de datos, se buscaba determinar si la red neuronal podía deducir las características relevantes del problema de forma más eficiente y, en última instancia, mejorar su rendimiento en la predicción del solver ganador.

Sin embargo, al igual que el modelo anterior, se obtuvo una precisión del 33 % al predecir siempre el mismo resultado, el solver ASP, que en otras palabras, es el SBS, y sigue sin ser suficiente. Por lo tanto, se decidió probar un enfoque más clásico de Machine Learning, ya que para aumentar el tamaño de esta red era necesario un mayor poder computacional.

A pesar de los esfuerzos realizados para mejorar el rendimiento de la red neuronal mediante la inclusión de características adicionales y la experimentación con diferentes representaciones de las imágenes, los resultados no fueron satisfactorios. La precisión obtenida no superó la selección del solver SBS en todas las instancias, lo que no cumplía con las expectativas de encontrar un modelo capaz de elegir un solver mejor que el SBS y cercano al VBS.

Al optar por un enfoque más clásico de Machine Learning, se esperaba mejorar los resultados y encontrar un modelo más adecuado para abordar el problema. Sin embargo, debe tenerse en cuenta que este enfoque también puede requerir una cantidad significativa de recursos computacionales para procesar la gran cantidad de datos y características utilizadas en el problema. Por lo tanto, es importante seguir investigando y explorando alternativas para mejorar el rendimiento y la eficiencia de los modelos utilizados en este tipo de problemas.

4.3.2. Modelo XGBoost

En el estado del arte, es común comparar los resultados de modelos de redes neuronales con enfoques de Machine Learning convencionales, como árboles de decisión, máquinas de vectores de soporte y modelos que no pertenecen a la categoría de Deep Learning. Entre estos últimos se encuentra el XGBoost (Extreme Gradient

Boosting), que utiliza un enfoque similar al de Kaduri et al. (2020).

A diferencia de las redes neuronales, estos modelos convencionales no reciben imágenes como entrada para extraer sus propias características. En su lugar, se les proporcionan características calculadas manualmente, a partir de las cuales el modelo puede determinar la mejor salida.

Las características a calcular incluyen el número de agentes, número de columnas, número de filas, número de obstáculos, densidad de obstáculos, dispersión de agentes, distancia promedio, distancia máxima y distancia mínima entre el origen y el destino de los agentes, y el tiempo.

Las características se obtienen de la siguiente manera:

- El número de agentes se refiere a la cantidad de agentes presentes en el problema, cada uno con una posición inicial y una posición objetivo.
- El número de columnas y filas corresponde a las dimensiones de la representación del mapa de obstáculos.
- El número de obstáculos indica la cantidad de celdas inaccesibles para los agentes en el mapa.
- La densidad de obstáculos se calcula dividiendo la cantidad de obstáculos por la cantidad de espacios vacíos.
- La dispersión de agentes se obtiene dividiendo el número de agentes por la cantidad de espacios vacíos en el mapa.
- La distancia promedio, máxima y mínima corresponden al promedio, el valor más alto y el valor más bajo de las distancias entre el punto de inicio y el objetivo de cada agente, respectivamente, tomando el conjunto de todos los agentes.
- El tiempo representa el momento en el que se desea determinar cuál solver es el mejor, ya que se están utilizando solvers anytime.

Es importante subrayar que existen múltiples métodos para calcular distancias en el espacio de características. Sin embargo, en el contexto de este estudio, se eligió la distancia euclidiana como métrica de distancia. Esta elección se justifica por la necesidad de minimizar el tiempo de cómputo en la obtención de características, especialmente dado que se emplean solvers anytime y que el tiempo constituye una métrica crítica en la evaluación del rendimiento.

Con estas características, el modelo puede ajustarse a los datos y proporcionar resultados. La Figura 4.5 muestra la importancia de las características, siendo la

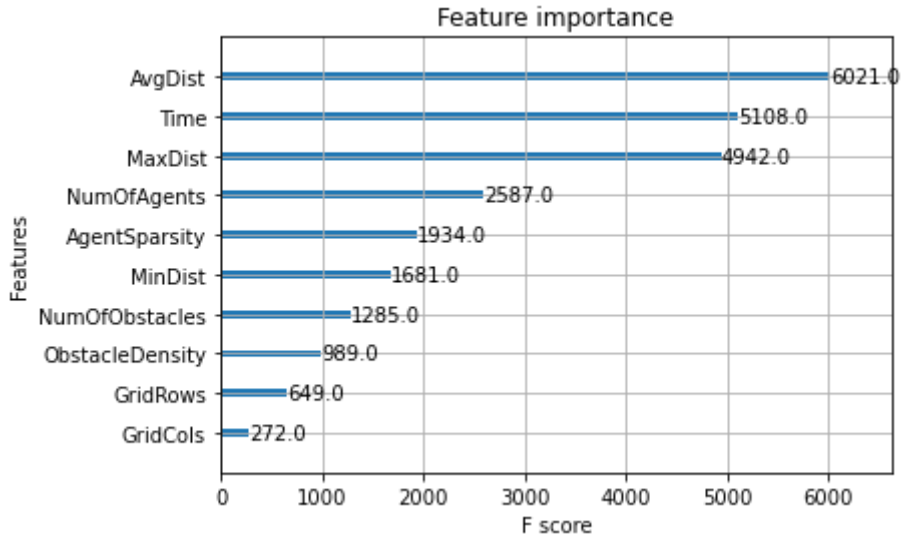


Figura 4.5: Gráfico de importancia de características en XGBoost.

distancia promedio la más relevante para clasificar correctamente las instancias. La importancia se calcula en función de la frecuencia con que esta característica divide el árbol de decisiones.

Los resultados de este experimento fueron muy positivos, con una precisión del 97 %, lo que sugiere que este enfoque más clásico es capaz de comprender mejor el problema.

Viendo los buenos resultados del modelo se decidió ampliar la investigación incorporando una mayor cantidad de datos y solvers. El objetivo principal de esta expansión es aumentar la robustez del estudio y mejorar aún más los resultados, proporcionando un análisis más exhaustivo y sólido en el campo de la inteligencia artificial.

Se llevaron a cabo experimentos utilizando los siguientes solvers, abarcando una amplia variedad de técnicas y enfoques en la resolución de problemas:

- Large Neighborhood Search, LNS ($NeighborSizeDefault = 5$)(Li et al., 2021a)
- Large Neighborhood Search, LNS con $NeighborSize = 8$ (Li et al., 2021a)
- Large Neighborhood Search, LNS con $NeighborSize = 10$ (Li et al., 2021a)
- Branch and Cut and Price, BCP (Sigurdson et al., 2019)
- Answer Set Programing, ASP (con una versión anytime) (Achá et al., 2021)

- Conflict Based Search, CBSH2 (Li et al., 2021b)
- Large Neighborhood Search mejorado, LNS2 (Li et al., 2021a)
- Explicit Estimation CBS, EECBS ($SubOptimalityDefault$ = 1,2)(Li et al., 2021c)
- Explicit Estimation CBS, EECBS con $SubOptimality$ = 1,1 (Li et al., 2021c)
- Explicit Estimation CBS, EECBS con $SubOptimality$ = 1,3 (Li et al., 2021c)
- Anytime EECBS (Li et al., 2021a)
- Anytime BCBS (Li et al., 2021a)
- MTMS (Achá et al., 2021)

Es pertinente aclarar que el grupo de solvers que se ha enumerado anteriormente incluye tanto los algoritmos anteriormente listados como variaciones de los mismos. Todos estos solvers se ejecutaron en la totalidad del conjunto de instancias.

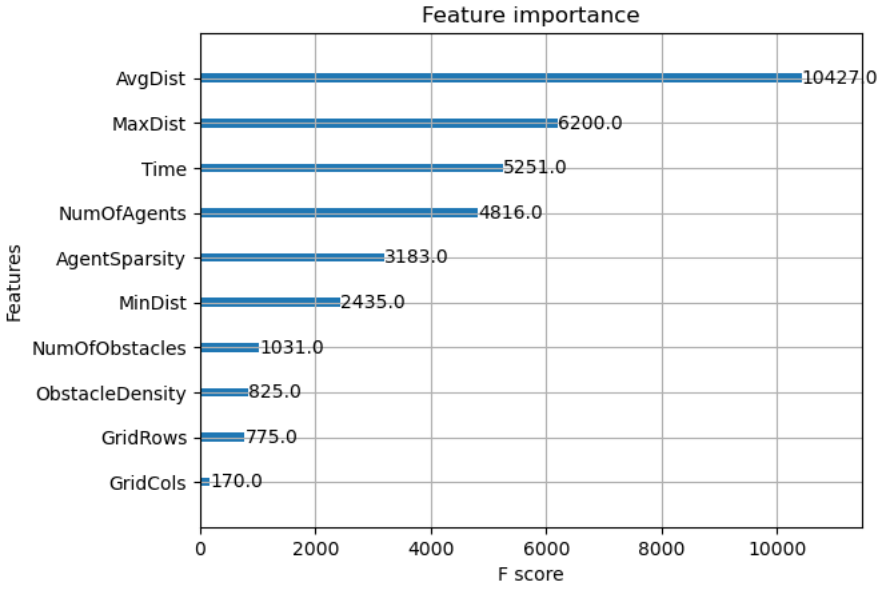


Figura 4.6: Grado de importancia de las características extraídas de las instancias.

Los resultados obtenidos en el conjunto de prueba fueron muy satisfactorios, alcanzando una precisión del 96,3 %. Además, se analizó la importancia de las características extraídas del conjunto de datos que se pueden ver en la Figura 4.6, destacando que la característica mas importante se mantuvo con la ejecución anterior, resultando ser consistente.

Al comparar los resultados obtenidos en estos experimentos con la ejecución anterior de XGBoost, se puede observar que ambos presentan un rendimiento muy similar en términos de precisión. Esta similitud en los resultados demuestra que el enfoque de XGBoost es altamente consistente y confiable, incluso cuando se enfrenta a un conjunto de datos más amplio y diverso con una mayor cantidad de solvers.

Capítulo 5

Resultados

Se iniciarán los análisis de los resultados con una mirada al desempeño del modelo XGBoost durante su etapa de entrenamiento. Este modelo se sometió a un proceso de aprendizaje de 100 iteraciones, con una configuración de detención temprana en caso de que no se observaran mejoras durante tres iteraciones consecutivas.

La figura 5.1 ilustra la métrica del Log Loss a lo largo del entrenamiento. Notablemente, se aprecia una rápida disminución del Log Loss durante las primeras iteraciones, lo que indica un ajuste efectivo a los datos. El gráfico muestra dos curvas, correspondientes al conjunto de entrenamiento y al conjunto de validación. El paralelismo en el comportamiento de ambas curvas sugiere que no se ha producido un sobreajuste en el modelo, ya que no se evidencia un aumento en el Log Loss para el conjunto de validación.

A continuación, se evalúa la métrica de Error de Clasificación, representada en la figura 5.2. Este gráfico despliega la magnitud del error de clasificación que el modelo experimenta durante el curso de su entrenamiento. Se puede apreciar que el modelo es capaz de clasificar las instancias con un grado de error muy reducido, el cual continúa descendiendo a medida que progresa el entrenamiento. Hacia el final, se observa una mínima discrepancia entre los errores de clasificación de los conjuntos de entrenamiento y de validación, pero sigue siendo insignificante. Con base en estos resultados, se puede afirmar que el modelo es eficaz al clasificar correctamente tanto las instancias de entrenamiento como las de validación.

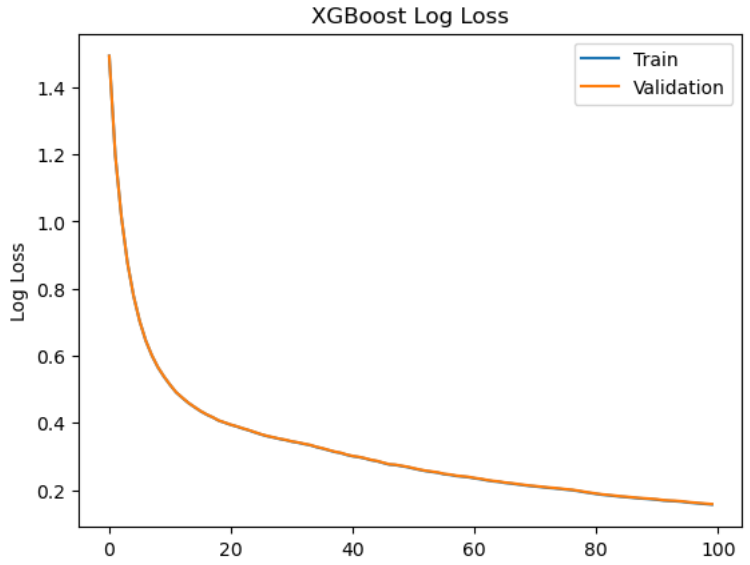


Figura 5.1: Log Loss del entrenamiento de XGBoost.

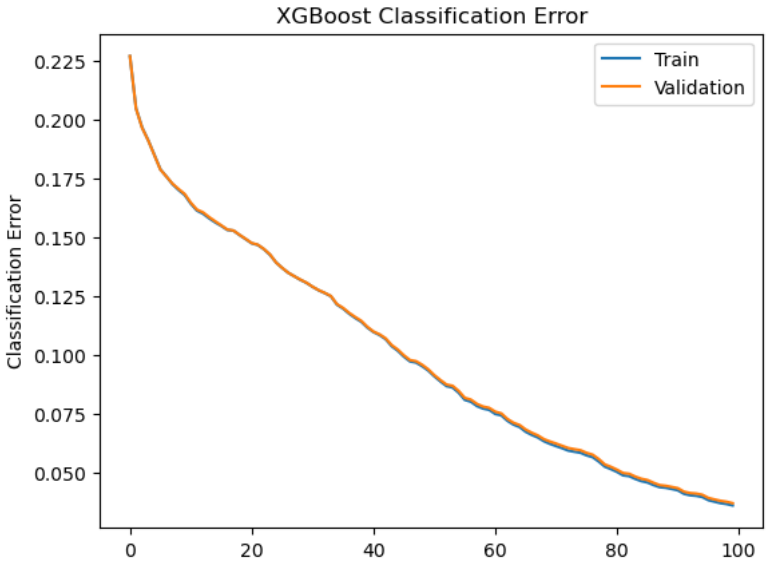


Figura 5.2: Classification Error en el entrenamiento de XGBoost.

Para garantizar que el modelo no esté sobreajustado a los conjuntos de datos de entrenamiento y validación, se recurre a un tercer conjunto de datos, el de prueba, que el modelo no ha visto durante el entrenamiento. Este proceso de validación independiente da lugar a una precisión del 0,9631 %, lo que indica que el modelo es capaz de realizar predicciones precisas incluso en datos no vistos anteriormente en

el entrenamiento. Esta alta precisión en el conjunto de pruebas confirma la robustez del modelo y su capacidad para generalizar a partir de los datos aprendidos.

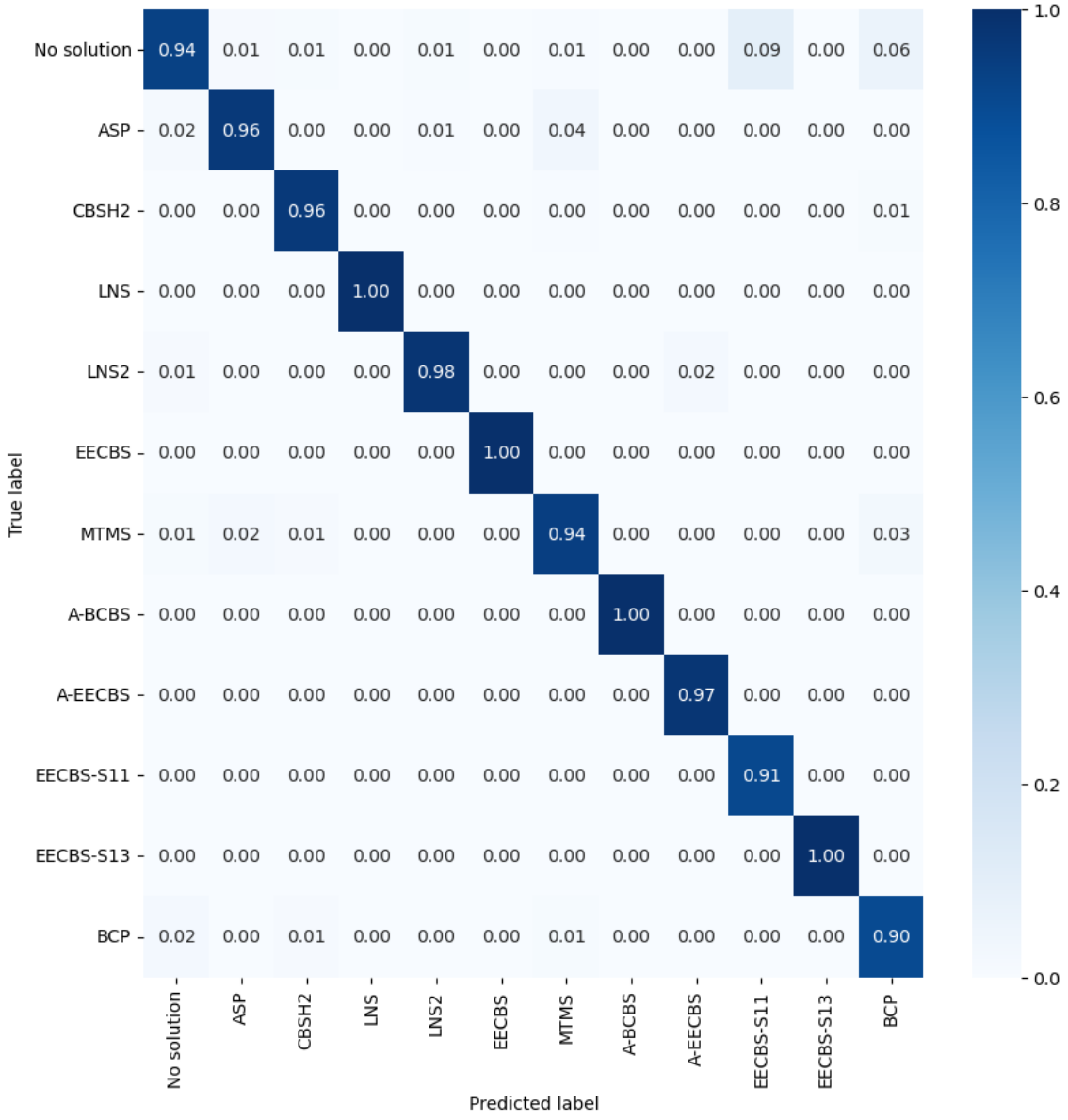


Figura 5.3: Matriz de confusión del conjunto de testing normalizada.

La matriz de confusión correspondiente al conjunto de prueba, representada en la figura 5.3, también se emplea para evaluar el desempeño del modelo. En esta matriz, se aprecia una marcada diagonal principal, lo que indica que el modelo hace una gran cantidad de predicciones correctas. Es decir, la mayoría de las predicciones

de cada clase caen en la diagonal, indicando que son correctas, y hay pocos casos de falsos positivos y falsos negativos. Esta tendencia acentuada en la diagonal refuerza la validez del modelo, demostrando su capacidad para clasificar correctamente la mayoría de las instancias en el conjunto de prueba.

La evaluación del modelo como metasolver continúa utilizando la métrica $\hat{m}$, detallada en secciones previas. Este indicador es esencial para valorar el desempeño del modelo en su función de metasolver, proporcionando una medida cuantitativa de cuán cerca está de identificar el Virtual Best Solver. Con este enfoque, se logra un análisis más completo y preciso del rendimiento del modelo en el contexto específico de la resolución de problemas de Multi Agent Path Finding (MAPF).

Inicialmente, se implementó la métrica general, generando un resultado de 0,176344. Es relevante comprender que este valor se encuentra en el rango entre 0 y 1, donde 0 simboliza el resultado óptimo y 1 corresponde al resultado adquirido por el SBS, es decir, el solver promedio, que en este conjunto de entrenamiento es LNS2. El resultado conseguido es muy satisfactorio, ya que se aproxima significativamente a 0. Esto indica que el modelo es competente para predecir el solver perfecto en un número considerable de casos.

Luego se aplicó la métrica por timestep y para analizar los resultados se obtendrán estadísticas como la media, el valor máximo, el valor mínimo y la desviación estándar para observar el comportamiento a lo largo de todo el espacio T.

Metric	**Result**
Mean	$0,264668$
Std	$0,030666$
Max	$0,345517$
Min	$0,231242$

Cuadro 5.1: Tabla métrica de $\hat{m}$

Al analizar la métrica por timestep (Figura 5.4), se obtuvieron los siguientes resultados: una desviación estándar de 0,030666, una media de 0,264668, un valor máximo de 0,345517 y un valor mínimo de 0,231242, como se puede ver en el Cuadro 5.1. Comparando estos resultados con la métrica general, se puede observar cierta variabilidad en el rendimiento a lo largo del tiempo. Sin embargo, los valores obtenidos siguen siendo bastante cercanos al cero, lo que indica un buen desempeño en general.

Se realizaron pruebas con ASAP (Gonard et al., 2019) y AutoFolio (Lindauer et al., 2015), ambos son selectores de algoritmos de renombre, que han demostrado desempeño excepcional en competencias y son compatibles con los estándares propuestos por ASlib (Bischl et al., 2016). Sin embargo, es importante destacar que

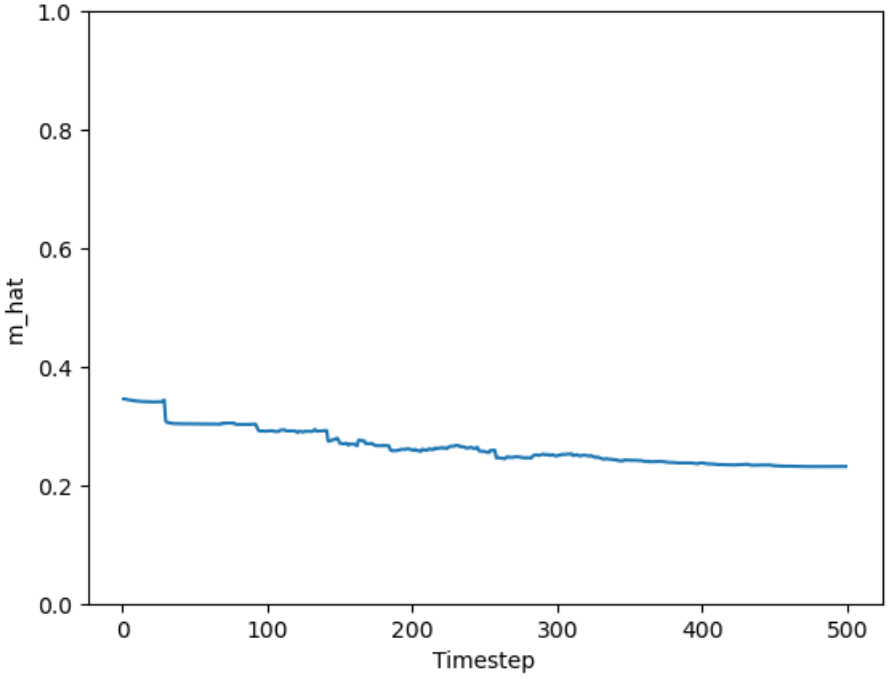


Figura 5.4: Gráfico de $\hat{m}$ de cada timestep

estas herramientas no fueron diseñadas específicamente para trabajar con algoritmos anytime.

Para intentar utilizar estas herramientas en el contexto de este trabajo, se adaptó el conjunto de datos de algoritmos existente de tal manera que pudiera ser procesado por ASAP y AutoFolio. Cada instante temporal de cada instancia se transformó en una instancia independiente. Este procedimiento resultó en un conjunto de datos de entrada de aproximadamente 3 millones de instancias. Sin embargo, esta cantidad de datos supera los límites y el propósito original para los que se diseñaron estos algoritmos de selección, hasta el punto de que no son capaces de completar una iteración en un periodo de 24 horas. Es por esta razón que se espera que en el futuro se pueda adaptar este formato para aceptar solvers anytime y establecer un punto de referencia para la comparación y la competencia con otras soluciones.

En conclusión, aunque la métrica por timestep muestra ciertas fluctuaciones en el rendimiento a lo largo del tiempo, el modelo sigue siendo efectivo y consistente en su capacidad para resolver problemas similares. Esto refuerza la confianza en el enfoque utilizado y sugiere que puede ser aplicado con éxito en futuras investigaciones y aplicaciones.

Capítulo 6

Conclusiones y Trabajo Futuro

En conclusión, el objetivo general de diseñar un modelo de Machine Learning que permita seleccionar un solver para problemas de Multi Agent Path Finding (MAPF) cercano al rendimiento del Virtual Best Solver, considerando las características particulares del problema y el límite de tiempo proporcionado por el usuario, se ha cumplido en gran medida. A pesar de no haber alcanzado el objetivo específico de recopilar al menos 10,000 instancias en el conjunto de datos, se logró reunir un conjunto de datos suficientemente amplio y variado que permitió abordar adecuadamente el problema.

Los objetivos específicos 2 al 6 se cumplieron exitosamente, lo que permitió seleccionar y modificar algoritmos del estado del arte, ejecutarlos en todas las instancias del conjunto de datos, identificar características relevantes y construir un modelo de Machine Learning preciso y robusto. Además, el modelo se evaluó utilizando benchmarks conocidos y generados, y se comparó su rendimiento con otros métodos del estado del arte para la selección de solvers en problemas de MAPF, esto con el resultado de la métrica $\hat{m}$.

Los resultados de los experimentos muestran que el enfoque utilizado en este estudio es efectivo y consistente en su capacidad para resolver problemas similares. La métrica $\hat{m}$ obtenida en los experimentos indica un buen desempeño en general, y la comparación con la ejecución de modelos de Redes Neuronales, anterior de XGBoost, demuestra la consistencia del enfoque.

La hipótesis planteada en este estudio sugiere que es factible desarrollar un metasolver para Multi Agent Path Finding (MAPF) basado en un modelo de Machine Learning, capaz de mejorar el rendimiento del Single Best Solver y aproximarse al rendimiento del Virtual Best Solver, teniendo en cuenta los tiempos de entrada establecidos por el usuario. Los resultados obtenidos respaldan esta hipótesis, ya

que el modelo de Machine Learning diseñado permite identificar el algoritmo más conveniente para cada instancia específica, en función del tiempo límite de ejecución establecido por el usuario, consiguiendo superar el rendimiento del Single Best Solver y aproximarse al desempeño del Virtual Best Solver.

En resumen, este estudio proporciona una base sólida para futuras investigaciones en la selección de solvers para problemas de MAPF utilizando Machine Learning. Los resultados obtenidos son prometedores y respaldan la viabilidad y efectividad del enfoque propuesto. Sería interesante continuar ampliando el conjunto de datos y explorar otras técnicas y modelos de Machine Learning para mejorar aún más la precisión y robustez del modelo en futuras investigaciones.

Para un trabajo futuro, se deja abierta la posibilidad de usar un modelo de redes neuronales más potente o encontrar una mejor solución en la entrada de los modelos. También se espera encontrar una forma estandarizada de comparar algoritmos anytime como lo hace ASlib para algoritmos no anytime.

Bibliografía

R. A. Achá, R. López, S. Hagedorn, and J. A. Baier. A new boolean encoding for mapf and its performance with asp and maxsat solvers. In *Proceedings of the International Symposium on Combinatorial Search*, volume 12, pages 11–19, 2021.

B. Bischl, P. Kerschke, L. Kotthoff, M. Lindauer, Y. Malitsky, A. Fréchette, H. Hoos, F. Hutter, K. Leyton-Brown, K. Tierney, et al. Aslib: A benchmark library for algorithm selection. *Artificial Intelligence*, 237:41–58, 2016.

R. Bogue. Growth in e-commerce boosts innovation in the warehouse robot market. *Industrial Robot: An International Journal*, 2016.

E. Boyarski, A. Felner, R. Stern, G. Sharon, D. Tolpin, O. Betzalel, and E. Shimony. Icbs: Improved conflict-based search algorithm for multi-agent pathfinding. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

M. Erdmann and T. Lozano-Perez. On multiple moving objects. *Algorithmica*, 2(1): 477–521, 1987.

J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

G. Gange, D. Harabor, and P. J. Stuckey. Lazy cbs: Implicit conflict-based search using lazy clause generation. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, pages 155–162, 2019.

M. Goldenberg, A. Felner, R. Stern, G. Sharon, N. Sturtevant, R. C. Holte, and J. Schaeffer. Enhanced partial expansion a. *Journal of Artificial Intelligence Research*, 50:141–187, 2014.

F. Gonard, M. Schoenauer, and M. Sebag. Algorithm selector and prescheduler in the icon challenge. *Bioinspired heuristics for optimization*, pages 203–219, 2019.

I. I. Huerta, D. A. Neira, D. A. Ortega, V. Varas, J. Godoy, and R. Asín-Achá. Anytime automatic algorithm selection for knapsack. *Expert Systems with Applications*, 158:113613, 2020.

I. I. Huerta, D. A. Neira, D. A. Ortega, V. Varas, J. Godoy, and R. Asín-Achá. Improving the state-of-the-art in the traveling salesman problem: An anytime automatic algorithm selection. *Expert Systems with Applications*, 187:115948, 2022.

O. Kaduri, E. Boyarski, and R. Stern. Algorithm selection for optimal multi-agent pathfinding. *Proceedings of the International Conference on Automated Planning and Scheduling*, 30(1):161–165, Jun. 2020. doi: 10.1609/icaps.v30i1.6657. URL `https://ojs.aaai.org/index.php/ICAPS/article/view/6657`.

O. Kaduri, E. Boyarski, and R. Stern. Experimental evaluation of classical multi agent path finding algorithms. In *Proceedings of the International Symposium on Combinatorial Search*, volume 12, pages 126–130, 2021.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

E. Lam, P. Le Bodic, D. Harabor, and P. J. Stuckey. Branch-and-cut-and-price for multi-agent path finding. *Computers & Operations Research*, 144:105809, 2022.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

J. Li, A. Felner, E. Boyarski, H. Ma, and S. Koenig. Improved heuristics for multi-agent path finding with conflict-based search. In *IJCAI*, volume 2019, pages 442–449, 2019.

J. Li, Z. Chen, D. Harabor, P. Stuckey, and S. Koenig. Anytime multi-agent path finding via large neighborhood search. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2021a.

J. Li, D. Harabor, P. J. Stuckey, H. Ma, G. Gange, and S. Koenig. Pairwise symmetry reasoning for multi-agent path finding search. *Artificial Intelligence*, 301:103574, 2021b.

J. Li, W. Ruml, and S. Koenig. Eecbs: A bounded-suboptimal search for multi-agent path finding. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 12353–12362, 2021c.

M. Lindauer, H. H. Hoos, F. Hutter, and T. Schaub. Autofolio: An automatically configured algorithm selector. *Journal of Artificial Intelligence Research*, 53:745–778, 2015.

M. Lindauer, J. N. van Rijn, and L. Kotthoff. The algorithm selection competitions 2015 and 2017. *Artificial Intelligence*, 272:86–100, 2019.

T. M. Mitchell. Machine learning, 1997.

J. R. Rice. The algorithm selection problem. In *Advances in computers*, volume 15, pages 65–118. Elsevier, 1976.

Q. Sajid, R. Luna, and K. Bekris. Multi-agent pathfinding with simultaneous execution of single-agent primitives. In *International symposium on combinatorial search*, volume 3, 2012.

G. Sharon, R. Stern, M. Goldenberg, and A. Felner. The increasing cost tree search for optimal multi-agent pathfinding. *Artificial intelligence*, 195:470–495, 2013.

G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66, 2015.

D. Sigurdson, V. Bulitko, S. Koenig, C. Hernandez, and W. Yeoh. Automatic algorithm selection in multi-agent pathfinding. *arXiv preprint arXiv:1906.03992*, 2019.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

R. Stern, N. R. Sturtevant, A. Felner, S. Koenig, H. Ma, T. T. Walker, J. Li, D. Atzmon, L. Cohen, T. K. S. Kumar, E. Boyarski, and R. Bartak. Multi-agent pathfinding: Definitions, variants, and benchmarks. *Symposium on Combinatorial Search (SoCS)*, pages 151–158, 2019a.

R. Stern, N. R. Sturtevant, A. Felner, S. Koenig, H. Ma, T. T. Walker, J. Li, D. Atzmon, L. Cohen, T. S. Kumar, et al. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Twelfth Annual Symposium on Combinatorial Search*, 2019b.

P. Surynek. Compilation-based solvers for multi-agent path finding: a survey, discussion, and future opportunities. *arXiv preprint arXiv:2104.11809*, 2021.

P. Surynek, A. Felner, R. Stern, and E. Boyarski. Efficient sat approach to multi-agent path finding under the sum of costs objective. In *Proceedings of the twenty-second european conference on artificial intelligence*, pages 810–818, 2016.

D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

W. Xiao-Long, W. Chun-Fu, L. Guo-Dong, and C. Qing-Xie. A robot navigation method based on rfid and qr code in the warehouse. In *2017 Chinese Automation Congress (CAC)*, pages 7837–7840. IEEE, 2017.

J. Yu and S. M. LaValle. Structure and intractability of optimal multi-robot path planning on graphs. In *Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.