La inteligencia artificial es la ciencia de

Hacer que las máquinas hagan cosas

Que requerirían inteligencia humana

Si se hicieran por humanos.

Marvin Minsky (1927-2016)

Modelos de Inteligencia Artificial

UD01 Inteligencia Artificial

A03.-El Aprendizaje

María Pato González

TAREA1: Realiza el/los cuestionarios..... 2
1.- Aprendizaje 3
2. Procesamiento de entrenamiento de un modelo usando aprendizaje supervisado. 4
3.-Resolución de problemas..... 6
3.1.- Inteligencia artificial y juegos..... 6
3.2.- ¿Qué es un problema? 8
3.2.1.- Teoría de grafos 8
Ejemplo: ConsuLta y ejecuta el colab UD01A03GrafosViAjante. 11
Tarea2.- El problema del vendedor..... 11
3.2.2. El espacio de estados. 11
3.2.3. Algoritmos de búsqueda en espacios de estados 15
Ejemplo: Consulta y ejecuta el notebook **UD01A03_Grafos_ViAJANTE**. 17
Ejemplo: Consulta y ejecuta el notebook UD01A03_Dijkstra_ciudades..... 17
3.2.3.1. Búsqueda heurística en espacios de estados..... 17
Ejemplo: Consulta y ejecuta el notebook UD01A03Busqueda_Dijistra_A..... 20
Ejemplo: Consulta y ejecuta el notebook UD01A03_Dijistra_CIUDADES. 24
6.- Técnicas de optimización 24
Ejemplo: Consulta y ejecuta el notebook UD01A03Optimizacion. 26
6.1. Algoritmos genéticos..... 26
6.2. Optimización enjambre de partículas 28
6.3. Algoritmo del descenso del gradiente. 29

A03.- RESOLUCIÓN DE PROBLEMAS.

TAREA1: REALIZA EL/LOS CUESTIONARIOS

Antes de realizar la tarea lee detenidamente este documento. Puedes entrar todas las veces que quieras en el cuestionario, pero solo podrá entregarlo una vez. Recuerdo hacerlo antes de que se cierre por que se corrige automáticamente y te envía las soluciones. Las respuestas incorrectas penalizan un 0,05.

1.- Aprendizaje

Para este punto vamos a centrarnos en el aprendizaje profundo. Veamos cómo se entrenan estos modelos.

Para adquirir conocimientos llevados a cabo un proceso que denominamos aprendizaje. Partimos de un conjunto de datos(información) y el objetivo es entrenar un modelo (llevar a cabo un aprendizaje) para qué pueda llevar a cabo alguna de estas funciones:

HD predicción: por ejemplo, partiendo de datos sobre viviendas (localización, tamaño, número de habitaciones, precio, etc) HP crear un modelo que aprenda a predecir el precio de la vivienda en una región. Otro ejemplo, a partir de un conjunto de imágenes de dígitos escritos a mano, etiquetados con el dígito que representan, crear un modelo que pueda predecir que digito en una imagen dada.

- Detección de patrones complejos: por ejemplo, a partir de un conjunto de radiografías del tórax, cada 1 de ellos etiquetado con la localización de un posible cáncer de pulmón, crear un modelo que detecte el cáncer es una desgraciada y donde está.
- Agrupación de datos similares: por ejemplo, tenemos una gran cantidad de datos sobre el comportamiento de los usuarios de una plataforma de streaming: que ven, cuándo, etc. A partir de estos datos podemos crear un modelo que al grupo de usuarios emergentes y caracterizarlos, a partir de ese momento podemos hacer reconocimiento acciones a otros usuarios según el grupo en que se sitúe.
- Reducción de dimensionalidad: por ejemplo, tenemos una gran cantidad de textos que queremos comparar, sabes cuáles se parecen entre ellos. Para hacerlo necesitamos reducir cada texto a un vector de tal manera que vectores de texto similares se sitúen cerca, y si son muy diferentes lejos.
- Generación de datos sintéticos: por ejemplo, tengo un texto y quiero completarlo (texto a texto). Otro ejemplo, tengo un texto y quisiera sintetizar la imagen que el texto describe (de texto e imagen).
- Optimización de acciones en entornos: por ejemplo, tengo un robot en forma de brazo y mano con 5 dedos si quiero entrenarlo para que aprenda a manejar un tenedor y a dar de comer espaguetis en una persona parapléjica.

Todos estos tipos de funciones que queremos que nuestros modelos aprendan necesitan de las siguientes técnicas de aprendizaje automático, cada función en un tipo de aprendizaje.

Aprendizaje supervisado: consiste en predecir un valor una categoría a partir de unos datos etiquetados. Esta es una de las aplicaciones más comunes del aprendizaje automático. En caso de predicción de valores, hablamos de problemas de regresión, donde el objetivo es predecir el valor numérico continúo basado en datos de entrada. Por ejemplo, predecir el precio de una casa. Por otro lado, la predicción de categorías se refiere a problemas de clasificación, donde el objetivo es asignar una etiqueta cada entrada, por ejemplo, predecir dígitos a partir de imagen.

Aprendizaje no supervisado: consiste en detectar patrones estructuras ocultas a partir de unos datos no etiquetados. Las técnicas de clustering, por ejemplo, agrupan datos similares basándose en características comunes. Otro caso es la reducción de dimensionalidad que permite simplificar los datos complejos mientras se mantienen aquellos más relevantes.

Aprendizaje por refuerzo: consiste en optimizar acciones mediante la interacción con el entorno. En el aprendizaje o refuerzo, un agente aprende a tomar decisiones óptimas interactuando con su entorno. Este tipo de aprendizaje es particularmente útil en aplicaciones donde las decisiones se toman secuencialmente y las consecuencias de las acciones se observan a lo largo del tiempo.

Si te preguntas qué tipo de aprendizaje se asocia a la generación de datos sintéticos, estos son dos subtipos del aprendizaje supervisado, en concreto auto supervisado y autorregresivo. Los veremos más adelante.

2. Procesamiento de entrenamiento de un modelo usando aprendizaje supervisado.

El aprendizaje supervisado es el más común en machine learning y Deep learning, sin menospreciar al resto de estrategias.

Partimos de un conjunto de datos etiquetados, como fotografías que están clasificadas según contienen imágenes de perros y gatos. El objetivo es crear un modelo que pueda predecir, para una nueva imagen qué aún no hemos visto, si contiene un perro, un gato o ninguno de los dos.

El proceso muy resumido será

1) El primer paso que siempre recopilar los datos y comprobar que son válidos, útiles y completos. A estos datos de etiquetado de los llamaremos conjuntos de datos (dataset).
2) Dividimos el dataset, obteniendo el conjunto de entrenamiento (trainset, típicamente el 80%) y el de testeo (testset, típicamente el 20% restante).
3) Usando el trainset entrenamos un modelo.
4) Usando el testset comprobamos que tal ha aprendido el modelo.
5) Despliegue monitorización de su rendimiento: cuando la calidad de este decaiga se vuelve al punto 1.

De estos 5 puntos, vamos a detallar pues importancia el tercero de ellos, pero aplicado a una red neuronal full dense (red neuronal con una o más capas internas completamente conectadas entre ellas y con las capas de entrada y de salida, veremos este término más adelante).

En estos momentos necesitamos conocer en detalle cómo funciona una red neuronal completamente conectada (full dense), solo debes saber que contiene una gran cantidad de números que llamamos parámetros (entre cientos y miles de millones e incluso billones, en general es fácil acercarse a 1000000 de parámetros en una renta normal simple.

El problema de entrenar una red neuronal artificial se reduce a escoger los valores que deben tener sus parámetros de tal forma que se minimice el error de aprendizaje. Es por lo tanto un problema de optimización.

Busca la combinación óptima es imposible que los ordenadores de hoy en día, pero podemos crear métodos que permiten que en un tiempo aceptable se encuentre una solución subóptima: más adelante veremos el algoritmo de retro propagación y los optimizadores basados en el descenso del gradiente.

El proceso es iterativo sobre el train set (en nuestro ejemplo formado por imágenes, cada imagen con su etiqueta):

1) Se hace inferencia con una imagen, obteniendo el resultado inferido.
2) Se calcula el error cometido usando el resultado inferido y el esperado (la etiqueta de cada imagen), este error se usa para corregir los parámetros, cada 1 de la medida en la que participó en el error, capa por capa de la última la primera.

3) Se sigue con la siguiente imagen y su etiqueta volviendo al punto 1.

Todo este proceso se denomina epoch. Este es un término similar al tiempo: hablamos de que el entrenamiento duró 100 epochs. Se ha finalizar un epoch aún tenemos un error que no consideraremos aceptable, repetimos el proceso, esto es iniciemos un nuevo epoch. El entrenamiento finaliza obviamente cuando llegamos a un error aceptable o al número de epochs máximos.

El resultado del entrenamiento (en este caso) es una arquitectura de red neuronal y un conjunto de parámetros, posiblemente grande. A todo ello lo llamamos modelo.

En lo siguiente punto verás por qué.

3.-Resolución de problemas

En el mundo real nos hemos encontrado constantemente con problemas que necesitan ser resueltos desde determinar la mejor ruta para llegar al trabajo hasta optimizar un proceso de fabricación en una planta industrial. Aquí es donde entra en juegos la inteligencia artificial clásica y las técnicas matemáticas de optimización.

Empecemos con la idea clásica. Esta se ha utilizado para resolver todo tipo de problemas, desde juegos de mesa como el ajedrez hasta la planificación de rutas robóticas. Aquí nos fijamos el reglas y algoritmos, y usamos el término inteligencia con el significado de emular el razonamiento humano para llegar a una solución. El foco está en encontrar una respuesta precisa utilizando métodos lógicos y a menudo exhaustivos.

Por otro lado, veremos las técnicas matemáticas de optimización.

¿Tienes una función objetivo que quieras maximizar o minimizar? Estas técnicas se encargan de ello. Pueden ser exactas o aproximadas. Estas técnicas son muy útiles cuando necesitamos encontrar la mejor solución dentro de un conjunto de opciones viables.

En muchas aplicaciones modernas, la inteligencia artificial clásica y las técnicas de optimización se combinan para ofrecer soluciones más robustas y eficientes.

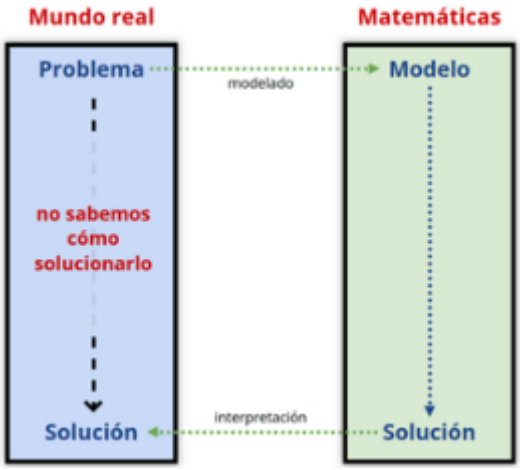
3.1.- Inteligencia artificial y juegos

Volvemos a la historia de la inteligencia artificial. En los años posteriores a la década de los 50 llevados por el escrito tenido en las Damas se vuelve a encarar el problema

del ajedrez siendo este mucho más complejo En cuanto a variedad y número de movimientos. Es por hecho que se emplea la misma técnica **minimax,** pero introduciendo métodos de poda en el árbol de juegos y erísticas para decidir entre el elevado número de movimientos a elegir. Estos mecanismos para reducir el número de jugadas a explorar se convertirán en otro factor de éxito de la mayoría de los algoritmos posteriores.

Para comparar la dificultad de los dos juegos elegidos, Damas y ajedrez, hay que indicar que mientras el programa de ajedrez de la época perdía ante jugadores de nivel inicial el de dama será capaz de ganar un humano de un alto nivel.

Un enfoque con un alto nivel de abstracción para solucionar problemas es usar modelos matemáticos.



En la figura vemos un esquema de este proceso: tenemos un problema en El mundo real, y pero no sabemos cómo resolverlo.

En nuestro caso, para solucionar un problema insolubre del mundo real primero lo modernizamos en un modelo matemático, representando el problema pero en El mundo matemático. Una vez modelizando aplicamos técnicas matemáticas conocidas para resolverlo. Una vez resuelto y mediante un proceso llamado interpretación volvemos al mundo real para encontrar que hemos hallado una solución. Un ejemplo clásico es el ejemplo del comercial que desea visitar varias ciudades y volver al punto de partida minimizando la distancia total recorrida. En este caso el modelo consiste en plantear el problema usando la teoría de grafos y usando lo que sabemos sobre esta encontrar la mejor solución que tendrá la forma de secuencia de aristas formando un camino entre los nodos del grafo. El paso final es interpretar esta

secuencia ordenada de aristas como una lista ordenada de carreteras que el comercial debe visitar en orden para volver a su ciudad de origen.

3.2.- ¿Qué es un problema?

Una de las habilidades más importantes de la mente humana y su capacidad para resolver problemas. Es por eso que la resolución de problemas es un tema fundamental en el campo de la inteligencia artificial.

La resolución de problemas se puede definir como el proceso mediante el cual, partiendo de ciertos datos iniciales y utilizando un conjunto de procedimientos predefinidos, somos capaces de determinar los pasos o elementos que nos conducen a una solución. Esta solución puede consistir en un conjunto de acciones que nos llevan a cumplir cierta condición o en la combinación correcta de los elementos iniciales para satisfacer ciertas restricciones.

Hay muchas aproximaciones para resolver un problema, pero nosotros ahora nos vamos a centrar en la teoría de los grafos y las técnicas de optimización.

3.2.1.- Teoría de grafos

Mediante la teoría de grafos representaremos de forma general un problema como un grafo. Si abstraemos los elementos que describen un problema podemos identificar los siguientes ítems.

1. Debemos encontrar un punto de partida. ¿Cuál es el estado inicial?
2. Debemos definir un objetivo alcanzar, a dónde queremos llegar. Uno o más estados finales. Esta es la forma de saber que hemos acabado.
3. Tenemos que definir qué acciones tenemos a nuestra disposición para resolver el problema. Esto es, cómo pasar de un estado a otro.
4. Es muy probable que haya restricciones ¿qué debemos tener en cuenta?
5. Y de forma optativa, conocimiento, entendiendo éste como la información relevante en el dominio concreto que nos puede ayudar a resolver el problema de una manera eficiente.

En el **ejemplo del comercial el punto de partida** es una ciudad que también es el objetivo por alcanzar una vez que hayamos recorrido el resto de las ciudades. Las acciones: son dada una ciudad a qué otra ciudad puedo ir desde esta. Finalmente, las restricciones en este caso son los kilómetros que hay entre una ciudad A y B, para cualquier par de ciudades que contemplemos.

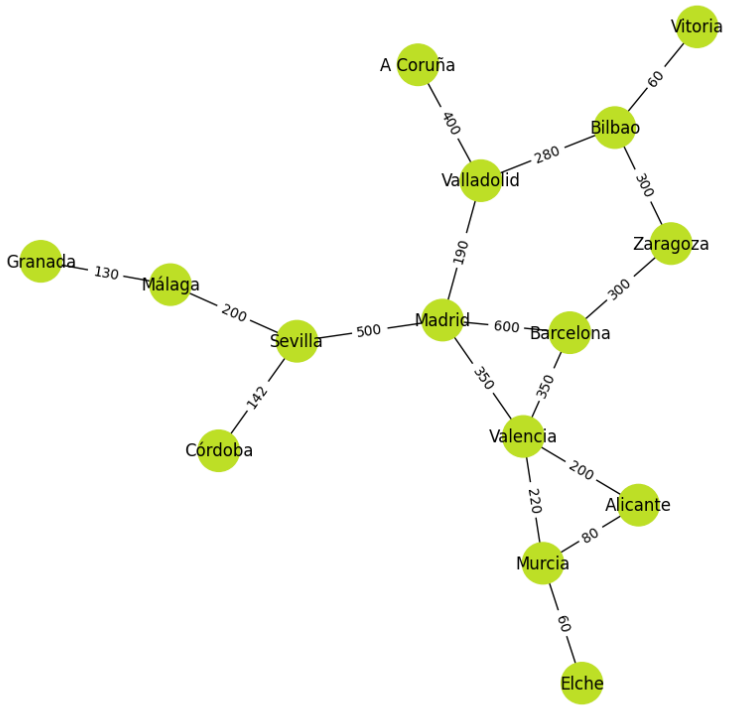
Modelizar el problema es probablemente el paso más difícil ahora de aplicar este enfoque. Alguna de las estrategias que puedes usar para afrontar este primer paso son

1. Identificar cada solución en potencia como un estado.
2. Si es posible aplica la máxima divide y vencerás cómo esto es, si puedes divide el problema en problemas más pequeños y encara a cada 1 de ellos independientemente.

En nuestro ejemplo los estados son las ciudades y cada carretera es una arista. Si dos nodos A y B están unidos de forma directa por una carretera, existirá una arista entre A y B que estará etiquetada con la distancia en kilómetros que representa.

En la siguiente figura podemos observar un grafo, en donde los nodos son ciudades de España, siendo las aristas el equivalente a las carreteras que la unen. Observemos:

1. No hay nodos inconexos, todos son accesibles por medio de una o varias aristas.
2. Las aristas no son dirigidas luego podemos ir y volver de 1 a otro adyacente por la misma arista.
3. Hay ciclos, por ejemplo, Murcia Valencia Alicante Murcia de nuevo, esto implica que ir de uno a otro a hablar más de una ruta
4. Definimos una ruta entre 2 nodos como una lista ordenada de aristas que los unen



Una vez que tenemos modelizado el problema en forma de grafo, buscaremos el algoritmo que, dentro de la teoría de grafos, solucione nuestro problema original. En este caso se trata de un viejo conocido: TSL [traveling salesman problem] Y es un tipo de problema del que no se conoce la solución óptima solo se puede aproximar una solución que tenga un tiempo razonable.

En nuestro caso fue:

```
Orden de visita: ['Madrid', 'Valladolid', 'Bilbao', 'Vitoria',
                  'Zaragoza', 'Barcelona', 'Valencia', 'Alicante',
                  'Murcia', 'Elche', 'Sevilla', 'Córdoba',
                  'Málaga', 'Granada', 'La Coruña', 'Madrid']

Distancia total: 5634 km
```

```
Orden de visita:
   A Coruña
   Valladolid
   Madrid
   Sevilla
   Málaga
   Granada
   Málaga
   Sevilla
   Córdoba
   Sevilla
   Madrid
   Valencia
   Alicante
   Murcia
   Elche
   Murcia
   Valencia
   Barcelona
   Zaragoza
   Bilbao
   Vitoria
   Bilbao
   Valladolid
   A Coruña
Distancia total: 5254 km
```

El código que lo generó:

```
ciclo = nx.approximation.greedy_tsp(G, weight='weight', source="Madrid")
```

La librería llamada **Networkx [alias nx],** especializada en trabajar con grafos. Siendo G el grafo, optimizamos sobre las etiquetas de las aristas y como fuente ponemos A Coruña. Como resultado nos da la lista ordenada de ciudades a visitar y con un pequeño cálculo a mayores el total de kilómetros que debemos realizar, una métrica/heurística que buscamos es minimizar.

EJEMPLO: CONSULTA Y EJECUTA EL COLAB UD01A03GRAFOSVIAJANTE.

TAREA2.- EL PROBLEMA DEL VENDEDOR

Modifica el código el notebook UD01A03GrafosViajante para se muestre la ruta más optima de una ciudad ida y vuelta para las 10 ciudades/poblaciones con más habitantes de Galicia. Ahora la heurística tendrá como prioridad minimizar el tiempo. Puedes emplear Google Maps y Google para obtener la información. Comparte con la profesora sin olvidar nombrar correctamente la tarea.

3.2.2. El espacio de estados.

Hemos visto que es posible solucionar un problema llevándolo al ámbito de las matemáticas aplicadas, en el ejemplo anterior por medio de la teoría de grafos.

Pero debemos generalizar para encontrar un método de modelar casi cualquier problema en forma de grafo, solucionarlo e interpretarlo.

Una forma de abordar el problema se conoce como el enfoque de espacio de estados. Llamamos estado a cada uno de los elementos que describen el problema en un

momento dado. Habrá dos estados especiales que distinguiremos el estado inicial y en los estados finales. La pregunta principal que nos haremos será qué elementos debemos incluir en el estado.

Ejemplo:

- Si hablamos del ajedrez, el tablero con las piezas colocadas según la jugada es un estado.
- Si hablamos de un partido de fútbol, las posiciones de cada 1 de los futbolistas de ambos equipos, el marcador y el balón es un estado.
- Si hablamos de una partida de cartas qué cartas tiene cada jugador y cuáles no, forma un estado

Para poder pasar de un estado a otro en el problema, necesitamos lo que llamamos operadores. Un operador es una función que transforma un estado en otro y/o otros estados.

Los elementos que definen un operador son:

1. **Aplicabilidad**: las condiciones que debe cumplir un estado para que podamos aplicárselo a otro estado.
2. **Transformación**: aquella que ha de aplicarse al estado para conseguir otro.

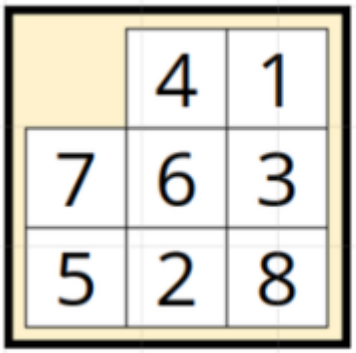
Los estados y cómo pasamos de 1 a otro forman lo que denominamos el espacio de estados. Este representa todos los caminos que hay entre estados posibles de un problema y se parece mucho a un grafo.

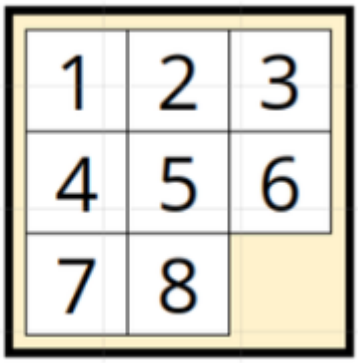
Pero, qué es una solución, una solución es 1 de los estados que, por la semántica del problema, finaliza la búsqueda. Por ejemplo, en el ajedrez cualquier configuración en donde uno de los Reyes del tablero es amenazado y no puede moverse haga lo que haga, es un estado final o estado solución.

Con anterioridad decíamos que una vez encontrada la solución había que interpretar esta para pasarlo al mundo real. Según esto tenemos dos tipos de problemas, aquellos que la interpretación es directa (como el problema del comercial) y aquellos en donde la solución en el mundo real no es la solución en el ámbito de las matemáticas aplicadas, sino el camino/recorrido quizá óptimo, o cercano al óptimo, según la alguna restricción-desde el estado inicial al final.

También, según sea el problema, quizás estemos buscando una solución cualquiera, la mejor de las soluciones o todas las soluciones posibles. Evidentemente el coste de cada una puede ser muy diferente.

Supongamos un puzle de 3 por 3, en estos juegos de cabezas tenemos una cuadrícula que contiene 8 fichas numeradas y un espacio vacío.





El juego consiste en, partiendo de la configuración aleatoria, mover las Casillas que son deslizantes hacia el hueco, finalizando cuando los números aparezcan en orden

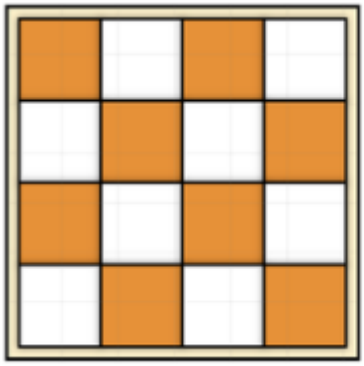
- Vamos a identificar los elementos del problema, modelizando lo como un grafo, esto es identificando estados y operadores:
- Un estado es una configuración del puzzle. En la imagen anterior vemos dos estados de ejemplo.
- El estado inicial: cualquiera configuración.
- Estado final he mostrado en la imagen superior.
- Como operadores tenemos 1: mover el hueco hacia la derecha, arriba, abajo e izquierda si es posible.
- ¿Qué es una solución en el mundo real?, lo que es lo mismo ¿cómo debemos interpretar la solución del grafo? Respuesta: los pasos que hemos dado para ir del espacio inicial al final; quizás en un número mínimo de pasos.

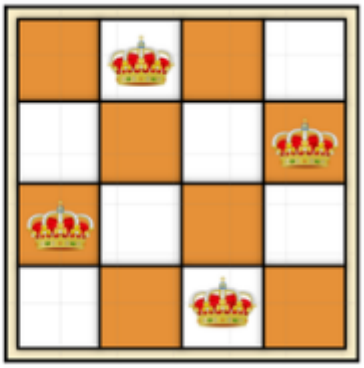
Otro ejemplo es el problema de las n reinas. El objetivo es colocar n reinas en un tablero de ajedrez de n por n de manera que ninguna reina esté en posición de atacar a otra. En otras palabras, no puede haber más de una reina en la misma fila, columna, o diagonal.

- Un estado es cualquier configuración del tablero conteniendo de 0 a n reinas (en la imagen inferior cuatro), incluso si éstas se puedan atacar.

- El estado inicial es un tablero vacío.
- El estado final es el tablero con n reinas colocadas de tal forma que no se ataquen entre ellas.
- El operador es si hay menos de n reinas en el tablero, colocar una reina en una casilla vacía. La forma de elegir esa casilla vacía puede ser aleatoriamente o seguir un orden de colocación. De esta decisión dependerá del tiempo que se tarde en encontrar una solución.

¿Qué es una solución en el mundo real? Buscar una configuración de la reina sin el tablero en la cual no se ataquen entre sí, o bien todas las configuraciones que cumplan los requisitos de ser estado final. Depende de si buscamos una solución o todas ellas.





Un concepto que debemos tener en cuenta es el tamaño de espacio de estados ¿cuántos estados debemos recorrer para buscar el estado final en el peor de los casos? Vamos a analizarlo para los valores de n:

Número de reinas	Número de estados
2	11
3	130
4	2.517
5	69.406
6	2.811.344
...	...
10	20.799.203.165.391

En general para las n reinas tenemos:

$$Tamaño\,del\,espacio\,de\,estados(N)=\sum_{k=0}^{N}\binom{N^2}{k}$$

Donde cada elemento de la suma (el binomio) es la forma de colocar cada reina sobre un tablero de n por n casi ya sin tener en cuenta el oro. Fíjate, en la tabla anterior,

cómo crece exponencialmente el número de estados a explorar buscar las soluciones. Para n igual a 10 habría que explorar casi 10^14 estados. Si tuviéramos una máquina que va comprobando los estados 1 a 1, a un ritmo de 1000000 por segundo, aun así, tardaríamos más de 3 años en recorrerlos todos y obtener todas las posibles soluciones.

Hora en iguala 15 tardaremos más de 3000 millones de años con la misma máquina.

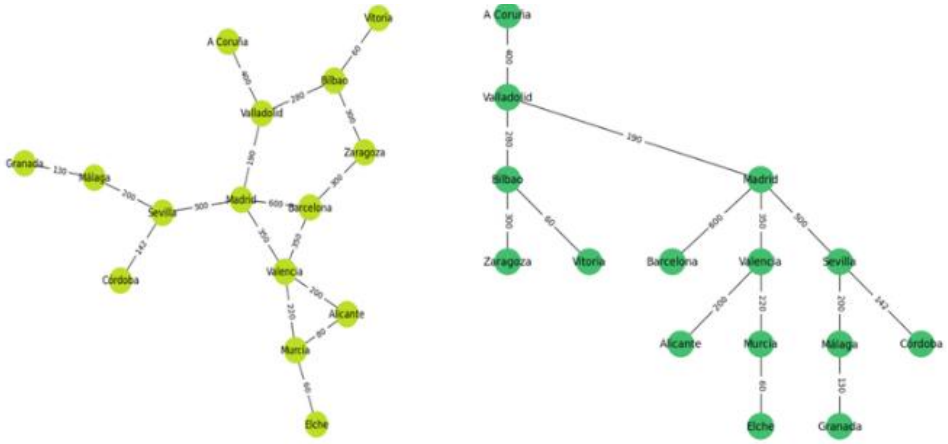
Pero tenemos un arma para hacer más pequeño un número de estados a comprobar. Esta herramienta depende mucho del problema; dependerá de cada 1 de ellos. Por ejemplo, si colocamos dos reinas en un problema de n reinas, si esta se ataca, cualquier combinación de las n menos 2 reinas restantes se pueden descartar y no comprobar. Esta arma herramienta se conoce como podar el árbol de búsqueda. Esta herramienta va a permitir reducir, muy significativamente, el número de estados a visitar

3.2.3. Algoritmos de búsqueda en espacios de estados

Para encontrar una o más soluciones a un problema por medio de la teoría de grafos, aplicaremos un algoritmo de búsqueda a lo largo del grafo. Es como para el ejemplo del vendedor partes de una ciudad y buscas el camino, quizás más corto, atravesando las aristas, a otra ciudad, si existe.

El proceso de buscar un estado final, partiendo de un estado inicial y teniendo un operador que te permite, a partir de un estado, saber cuáles otros has de visitar, es la búsqueda en el espacio de estados y se puede usar varias familias de algoritmos.

Ya vimos el grafo generado en el problema de vendedor, hora lo que vamos a hacer es convertir este grafo en un grafo completo todas las ideas son alcanzables, y a cíclico (no existen ciclos) tomando La Coruña como un nodo raíz.



Esta nueva estructura de datos se denomina árbol, y es la estructura idónea para hacer una búsqueda, ya que no hay ciclos. La raíz es el estado inicial y alguno o más de 1 de los estados restantes es un estado final.

Para los árboles sobre los que trabajarán los algoritmos de resolución de problemas son diferentes de los algoritmos habituales. La característica principal es que el árbol se construirá a medida que se haga la exploración, dado que- como sabemos-el tamaño que puede tener este hace imposible que se pueda almacenar en memoria este sería el pseudocódigo de un algoritmo que haya una solución de un problema en el espacio de estados:

```
Algoritmo de Búsqueda General
   por_visitar.insertar(estado_inicial)
   actual = por_visitar.primero()
   while not es_final(actual) and not por_visitar.vacia() do
         por_visitar.borrar primero()
         visitados.insertar(actual)
         hijos = generar_sucesores(actual)
         hijos = tratar_repetidos(hijos, visitados, por_visitar)
         por_visitar.insertar(hijos)
         actual = por_visitar.primero()
   end while
```

El algoritmo, antes de entrar en el bucle while, hace que actual sea igual al estado inicial. El bucle se ejecuta una y otra vez mientras se cumpla la condición: hora actual

no es un estado final y aún tengamos espacios por visitar. Cada vez que se ejecuta una iteración del bucle, al principio añade actual a la lista de visitados, lo elimina de la lista de estados por visitar, generando la lista de los hijos de actual: que son aquellos estados alcanzables desde actual de forma directa. Luego trata los hijos repetidos y los añade a la lista por visitar. Finalmente, antes de volver al principio de bucle, saca el siguiente actual De la lista por visitar.

EJEMPLO: CONSULTA Y EJECUTA EL NOTEBOOK UD01A03_GRAFOS_VIAJANTE.

A partir de este algoritmo general, podemos hacer una clasificación de estos dos tipos

1. **Algoritmos de búsqueda no informada**: estos algoritmos no tienen en cuenta el coste de la solución durante la búsqueda según una métrica dada. Su funcionamiento es sistemático, sigue un orden de visitas de estados fijo, estableciendo por la estructura del espacio de búsqueda. Dentro de estos podemos contemplar dos subtipos:
    a. **Búsqueda en profundidad:** se recorre el árbol de arriba abajo.
    b. **Búsqueda en anchura:** se recorre el árbol en anchura
2. **Algoritmos de búsqueda heurística:** son algoritmos basados en la experiencia. Estos algoritmos utilizan una estimación de coste o de la calidad de la solución para guiar la búsqueda, de manera que se base en algún criterio jurídico propio del problema. Estos algoritmos no son sistemáticos, de manera que el orden de exploración no determina la estructura del espacio de búsqueda sino el criterio jurídico. Algunos de ellos tampoco son exhaustivos y basan su menor complejidad computacional en ignorar parte del espacio de búsqueda.

EJEMPLO: CONSULTA Y EJECUTA EL NOTEBOOK UD01A03_DIJKSTRA_CIUDADES

3.2.3.1. Búsqueda heurística en espacios de estados

Es evidente que los algoritmos de búsqueda en informada serán incapaces de encontrar soluciones en problemas en los que el tamaño del espacio de búsqueda sea enorme (15 reina sin ir más lejos): todos estos algoritmos tienen un coste temporal que es una función exponencial del tamaño de la entrada, por lo tanto, el tiempo para encontrar la mejor solución a un problema no es asumible en problemas reales.

La manera de conseguir reducir este tiempo de búsqueda a algo razonable es hacer intervenir conocimientos sobre el problema que queremos resolver dentro del funcionamiento del algoritmo de búsqueda. Pero perdemos algo, generalidad;

porque este conocimiento es particular a cada problema, por lo tanto, no será exportable a otros.

Perderemos en generalidad, pero podremos ganar en eficiencia. Aunque siempre existirán todo un conjunto de problemas en los que la búsqueda del óptimo será imposible típicamente por el tamaño de su espacio de búsqueda.

Para poder plantear la búsqueda del óptimo siempre debemos tener algún coste a la hora de obtener una solución. Lo óptimo entonces será supeditado al coste, y este dependerá del problema. Este coste lo calcularemos sobre el camino que nos lleva desde el estado inicial del problema hasta el estado final. No todos los problemas se pueden plantear de esta forma.

Por lo tanto, supondremos que podemos plantear nuestra búsqueda como la búsqueda de un camino y que de alguna manera somos capaces de saber estimar cuál es la longitud o coste de este. Por lo general tendremos un coste asociado a los operadores que nos permiten pasar de un estado a otro (por ejemplo, el coste pueden ser los kilómetros entre ciudades adyacentes), creo que este coste tendrá un papel fundamental en el cálculo del coste total del camino.

Si nuestro conocimiento fuese perfecto, a medida que exploramos el espacio de búsqueda y hemos tomado las mejores decisiones, haciendo nuestra búsqueda más o menos exitosa, al dirigirnos rápidamente hacia el objetivo descartando todos los caminos de mayor coste. En el otro extremo, si estuviésemos en total ignorancia, tendríamos que explorar muchos caminos antes de ayer resolución óptima, me hicimos en el caso de la búsqueda no informada.

En el ajedrez, el espacio del Estado se está compuesto por todas las posibles configuraciones del tablero y las diferentes jugadas que se pueden realizar. Dado que el número de posibles movimientos en el ajedrez es enorme, es prácticamente imposible examinar su estimación de todas las jugadas para determinar la mejor

opción en cada turno. Por tanto, se utilizan algoritmos de búsqueda turística para guiar la exploración del espacio de estados de manera más eficiente.

Enfoque común para la búsqueda turística en el ajedrez es el uso del **algoritmo minimax.**

Esta es una técnica de decisión para juegos de 2 jugadores con información perfecta, ambos jugadores tienen la misma información sobre el estado de los juegos y esta información es completa, que juegan por turnos. Si nos centramos en el ajedrez, es básica es evaluar el tablero desde la perspectiva de un jugador, tratando de minimizar la mejor jugada que el oponente pueda hacer en respuesta a cualquiera de nuestras jugadas. Se construye un árbol que representa todas las posibles secuencias de movimiento hasta una cierta profundidad (búsqueda en profundidad limitada), se evalúa cada posición del tablero en los estados terminales del árbol mediante una función de evaluación (HD también llamada función de coste)

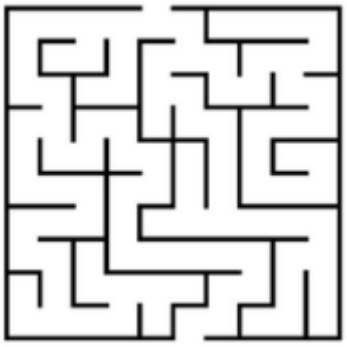
1) Inicialmente se parte de un estado que representa la configuración actual del tablero. Este es nuestro estado inicial.
2) Desde este estado, se generan todos los movimientos legales para el jugador actual (vamos a llamarle Max e imaginamos que somos nosotros), lo que da lugar a nuevos estados en el árbol.
3) A continuación, HP para cada 1 de estos nuevos estados generados, se generan todos los movimientos legales para el oponente (a que llamaremos min).
4) Este proceso se repite hasta llegar a una cierta profundidad o hasta que se alcance posiciones terminales (como jaque mate o tablas).
5) Una vez alcanzada la profundidad deseada, se evalúa la configuración del tablero en cada estado terminal utilizando la función de coste. Esta función asigna un valor numérico a cada estado que representa lo favorable que es para nosotros (Max).
6) Estos valores se propagan hacia atrás en el árbol, desde las hojas hasta el estado inicial. En cada nivel, el jugador Max (nosotros) escogeremos El movimiento que resulta en el valor máximo, mientras que para el jugador min escogeremos el movimiento que resulta en el valor mínimo.
7) Al llegar de nuevo al estado inicial el movimiento elegido será aquel que tenga el valor máximo resultante

Video Algoritmo Minimax en 4 minutos:

https://www.youtube.com/watch?V=qjjm7ekdruc

Para mejorar la eficiencia, el algoritmo minimax a menudo se usa en combinación con técnicas como la poda Alfa beta, que eliminar ramas del árbol de búsqueda que no necesitan ser exploradas porque no afectarán a la decisión final.

Vemos ahora como ejemplo la búsqueda de la salida de un laberinto utilizando una búsqueda heurística. Supongamos que tenemos un laberinto representado por una cuadrícula en la que cada celda puede ser un pasillo o una pared. El objetivo es encontrar el camino desde una posición inicial hasta una posición objetivo, evitando las paredes y los caminos sin salida.



EJEMPLO: CONSULTA Y EJECUTA EL NOTEBOOK UD01A03BUSQUEDA_DIJISTRA_A.

La búsqueda más común utilizada para resolver este tipo de problemas, en donde hay un estado inicial y 1 o más estados finales, pero lo que realmente importa es el camino, es el **algoritmo A*.**

Este algoritmo es un algoritmo de búsqueda en grafos utilizado para encontrar el camino más corto entre un nodo inicial y un nodo objetivo. Es una extensión del algoritmo de Dijkstra Que usan este caso para mejorar la velocidad. El algoritmo mantiene dos estructuras de datos principales: una lista abierta de nodos por explorar (por visitar) y una lista cerrada de nodos ya explorados (visitados). Vuelvo al pseudocódigo del algoritmo de búsqueda general. Dado un estado tenemos 3 funciones (semejantes a las funciones de coste de minimax):

$$g(n)\,,\ h(n) \text{ y } f(n)=g(n)+f(n)$$

- Sea n un estado o nodo,
  - **G(n)** calcula el coste del camino desde el estado inicial al nodo n.
  - **H(n)** es una estimación heurística del coste de ir desde n al nodo final.

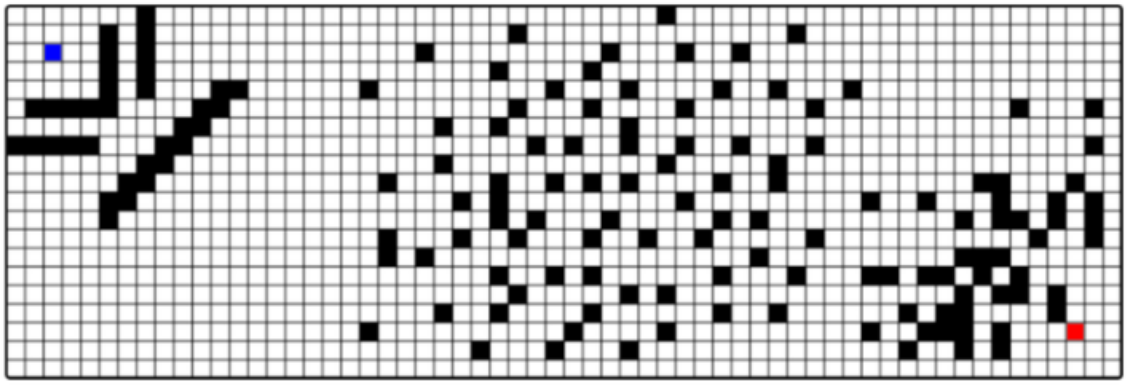
- **F(n)** es el coste estimado de ir desde el nodo inicial al nodo final pasando por n.

La idea principal es cuando tengas que escoger el siguiente actual de la lista por visitar, escojas aquel estado que tenga un f(n) HD menor. Esto obligará al ritmo a explorar prioritariamente los caminos con menor coste total estimado.

Y hay otro detalle a tener en cuenta en algoritmo de búsqueda general adaptado a A*. Entonces pasamos por alto una función llamada tratar repetidos que ahora tiene Sentido. Soy un ciclo volvemos a ver un nodo que ya hemos visitado, en vez de ignorarlo, lo tenemos en cuenta si el nuevo f(n) calcula dos menores que el f'(n) que tuvo cuando se visitó por última vez. Interpretándolo, si al volver por el mismo lo encontramos que por el nuevo camino que estamos esperando tiene un menor coste estimado, seguimos por él olvidando lo anterior.

Aquí la dificultad es calcular h(n), no es trivial dado que es una estimación y no conocemos el futuro h(n). Por ejemplo, si el problema es buscar el camino más corto entre 2 ciudades y estamos en la ciudad n, ¿cómo estimar el h(n)? Pues el método es sencillo, si estamos sobre un mapa, es calcular la distancia en una línea recta, sin tener en cuenta las carreteras. Eso puede ser una buena estimación del futuro en este caso.

Vamos a poner un ejemplo muy visual: la búsqueda de un camino en un mapa cuadriculado como el que se mostró en la imagen anterior. Este mapa posee un origen cuadrícula azul, y un destino cuadrícula roja. El problema es ir desde el punto del origen al destino sorteando los Obstáculos representados con cuadrículas negras. El operador que nos permite pasar de una cuadrícula a otra tiene en cuenta que no puede caer en una cuadrícula negra, esto es, al generar los sucesores del Estado actual, no se incluyen las cuadrículas negras. Las cuadrículas son de 15 por 15, y la distancia entre ellas es la distancia euclídea entre sus centros.



¿Cuál es el camino óptimo?

Primero modernizamos el problema con grafo. Es simple. El estado inicial son las coordenadas de la cuadrícula azul, el estado final en la cuadrícula roja; para un estado dado, sus estados hijos son los que pueden alcanzar dando un solo paso, excepto si son obstáculos. Por lo tanto, en el mejor de los casos, un estado dado tiene 8 estados hijos.

Apliquemos primero el **algoritmo Dijkstra**. Este algoritmo es utilizado para encontrar el camino más corto desde 1 de origen a todos los otros nodos en un grafo ponderado (tienen un número. Por ejemplo, kilómetros). El grafo puede ser dirigido no dirigido, pero los pesos de las aristas deben ser no negativos. En nuestro caso cada arista (que une dos Casillas adecentes) es esta ponderada por la distancia euclídea entre las Casillas. La distancia euclídea entre: es la longitud del segmento de línea recta que conecta estos: en un plano.

Cada nodo del grafo está formado por las coordenadas (x, y) en la cuadrícula, su g(n) inicialmente infinito y quienes su nodo anterior en el camino encontrado, inicialmente ninguno. Él no denunció los especiales, dado que posee un g(n)=0.

Pasos del algoritmo en comparación con el algoritmo de búsqueda general:

1) Crear un conjunto de nodos visitados, que inicialmente estaba vacío
2) Era una cola de prioridad llamada por visitar, en editó los nodos de la cuadrícula.
3) Mientras por visitar no esté vacía:
    a. Extraer 1 dos de por visitar, llamando actual
    b. HP introduce ese nodo en visitados
    c. Calcula los hijos de actual, y para cada 1 de ellos(hijo) hacer:

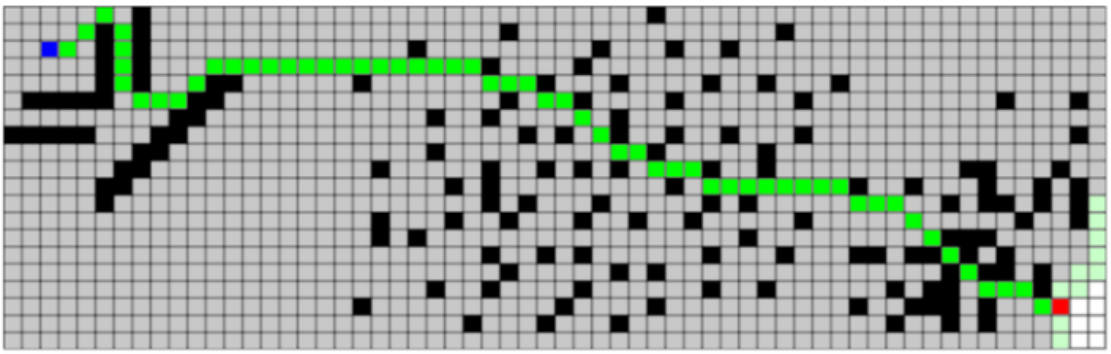
i. Calcula la distancia tentativa desde el nodo inicial a hijo: por tanto, g'(hijo) Es la suma de g(actual) más la distancia de actual a hijo.

ii. Sí g'(hijo) es menor que g(hijo) entonces hacer qué dijo actualice su g(hijo) a g'(hijo) y su anterior actual.

4) Cuando por visitar esté vacía, el algoritmo ha terminado. Las distancias almacenadas para cada nodo son las distancias más cortas desde el nodo origen. Entre ellas estará el nodo destino, con su g(destino) Y su anterior, iniciando el camino inverso a recorrer para llegar al nodo origen.

El algoritmo puede parar un poco antes, cuando actual sea igual al del destino.

El resultado se muestra a continuación, podemos ver el camino óptimo en verde. También podemos ver Casillas en gris (HD visitadas) HP y otros verdes pastel (por visitar). Hubo un detalle de implementación, en el notebook no meto todos los nodos en la lista por visitar, solo me tono de inicial; ahorrando memoria y me permite ver el frente del aula, ya que este algoritmo en ejecución trabaja como si de una inundación se tratase.



Objetivo alcanzado: 994.2640687119278

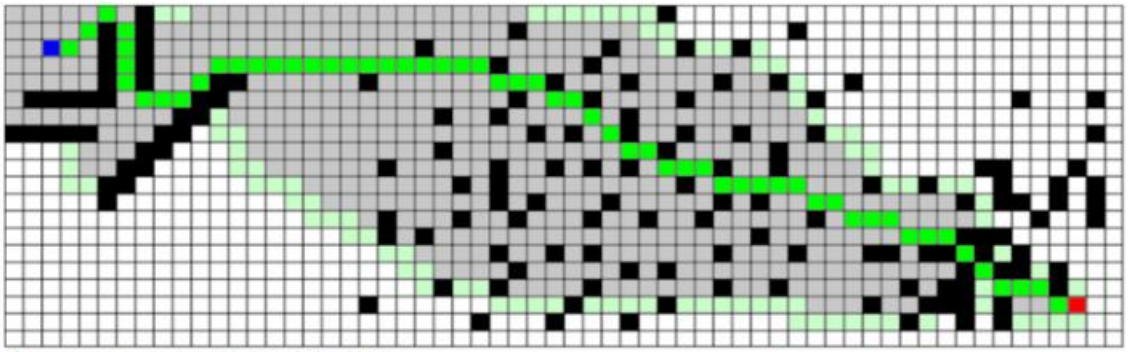
Vemos que la solución óptima del algoritmo de Dijkdtra mide 994,26 unidades; Recordemos que este algoritmo encuentra el camino óptimo. Observa parque para encontrar el destino tuvo que recorrer una gran parte de la cuadrícula, prácticamente toda. ¿Hay alguna manera de que vaya un poco más dirigido al nodo destino? Sí el algoritmo A*.

Para un nodo dado podemos estimar la distancia desde este al nodo destino y usarlo de pista. Dijimos que llamábamos a esta distancia estimada h (nodo) y definimos f(n) como la Suma de g(n) y h(n). Hora la cola de prioridad por visitar, tiene como criterio de orden para un nodo dado a f(nodo).

En la imagen posterior podemos ver el resultado de ejecuta sobre el mismo tablero el algoritmo A*. Las cuadrículas en blanco son aquellas que no ha tenido que visitar y tampoco han formado parte del conjunto de nodos pendientes por visitar.

Podemos observar que ha encontrado el mismo camino con la misma distancia total y esta vez no ha inundado todo el tablero; ha priorizado aquellos nodos con un f(n) menor, y por tanto la inundación se ha orientado más hacia la solución.

Como nota hay que indicar que algunas de las técnicas que acabamos de ver pueden ser usadas como herramientas de optimización; por ejemplo; optimización de la distancia más corta.



Objetivo alcanzado: 994.2640687119278

EJEMPLO: CONSULTA Y EJECUTA EL NOTEBOOK UD01A03_DIJISTRA_CIUDADES.

6.- Técnicas de optimización

Para crear máquinas inteligentes, deberíamos tener la capacidad de enseñarles a resolver problemas. En los tiempos de la IA simbólica los algoritmos creados al efecto no tenían la facultad de aprender, se designaban dentro de las matemáticas aplicadas para resolver problemas concretos y sufrían de problemas con la explosión exponencial del espacio de estados en el caso de ser modelizados como grafos.

Sin embargo, la teoría de grafos fue y sigue siendo fuente de inspiración, sin embargo, hay problemas que no se adaptarían a la modernización basada en grafos. Por ejemplo: queremos organizar un torneo de fútbol internacional tipo liguilla, cada equipo está alojado en 10 ciudades diferentes, y queremos planificar los partidos de tal manera que se minimice las distancias que los equipos deben viajar y se maximice la distancia entre los aficionados violentos. Al **modelar este problema en forma de**

grafo vemos que el número de estados es exponencialmente siendo casi inviable su posible exploración aquí es donde aparecen las técnicas de optimización.

Las técnicas de optimización son métodos matemáticos y algorítmicos diseñados para encontrar la mejor solución posible a un problema dado, generalmente sujeta a un conjunto de restricciones. Estas técnicas buscan maximizar o minimizar una función objetivo, como el costo, el tiempo o la eficacia, explorando sistemáticamente diferentes soluciones posibles para determinar cuál es el óptimo.

El corazón de esta técnica es la función objetivo, o siendo esto una función matemática que representa la cantidad que se quiere optimizar. Puede ser algo que se desea en maximizar, como la ganancia en un negocio, o minimizar como el tiempo que tarda un algoritmo en completar una tarea. La función objetivo define de forma cuantitativa como de buena o mala es una solución particular.

Según esto como las técnicas de optimización pueden clasificarse en dos categorías principales.

1. **Optimización exacta**: estos métodos garantizan encontrar la solución óptima. Sin embargo, estos métodos pueden ser computacionalmente intensivos y, por lo tanto, poco prácticos para problemas de gran escala o para aquellos donde la función objetivo es compleja.
2. **Optimización aproximada:** estos métodos no garantizan una solución óptima, pero son generalmente más rápidos y flexibles. Estos métodos son particularmente útiles cuando la función objetivo es no lineal discontinua o desconocida o cuando el espacio de búsqueda es demasiado grande para una búsqueda exhaustiva
    a. **Algoritmos genéticos:** son una familia de algoritmos de optimización inspirados en la teoría de la evolución natural. Son una técnica de computación evolutiva usados para encontrar soluciones aproximadas a los problemas.
    Https://www.youtube.com/watch?V=rbrxgyo0kiw
    b. **De partículas:** método de optimización ahoritica basada en el comportamiento colectivo de hambres, como bancos de peces o bandadas de aves. Cada partícula representa una solución potencial en el espacio de búsqueda. Las partículas se mueven en este espacio según una serie de

ecuaciones matemáticas que consideran tanto su posición actual como la posición de sus vecinos.

c. **El descenso de gradiente**: el algoritmo del descenso de gradiente es un tipo iterativo, de una parte, matemática dura.
Https://www.youtube.com/watch?V=a6ficdoz8_4

De todas estas técnicas de optimización aproximada nombradas, todas excepto descenso del gradiente están inspiradas en la naturaleza, simulando situaciones complejas que esta resuelve eficientemente. Todas Estas son de carácter interactivo, esto es, y tenemos una y otra vez aproximándose a una solución; esta puede ser finalmente encontrada o, lo más probable, entregarnos una selección cercana a esta.

Supongamos que deseamos diseñar un sistema de control para un robot, cuya función principal es dirigir o regular su comportamiento, y que queremos optimizar dos parámetros del sistema, x e y, que afectan respectivamente el rendimiento y la eficiencia energética del robot. Tras un largo proceso hemos modernizado la función óptima como:

$$f(x,y)=(a-x)^2+b(y-x^2)^2$$

Con a igual a 1 y B igual a 100. Esta función evalúa el comportamiento del sistema de control del robot en función de 2 valores. Lo que estamos buscando son los valores que minimicen a f.

Para dos variables este objetivo: (x, y) que minimice f; se puede resolver de forma analítica, siendo f (x, y) =0 si x=1 e y=1. Esta sería la solución exacta mínimo global.

Vamos a ver algunas de estas técnicas de imitación aproximada.

EJEMPLO: CONSULTA Y EJECUTA EL NOTEBOOK UD01A03OPTIMIZACION.

6.1. Algoritmos genéticos

Los algoritmos genéticos son una familia de algoritmos de optimización inspirados en la teoría de la evolución natural. Son técnicas de computación evolutiva usadas para encontrar soluciones aproximadas a problemas.

La teoría de la evolución sostiene que las especies cambian a lo largo del tiempo debido a variaciones en sus características heredables. Estas variaciones pueden surgir por mutaciones, recombinaciones genéticas y otros mecanismos. Los

individuos con características que les confieren una ventaja en su entorno tienen más probabilidad de sobrevivir y reproducirse. Con él tiempo, estas características ventajosas se vuelven más comunes en la población, lo que lleva el desarrollo de nuevas especies o a cambios significativos de una especie existente.

Los elementos claves de la teoría de la evolución que inspiran a los adherimos genéticos son:

- **Variación**: al igual que las especies tienen variabilidad genética, los algoritmos genéticos trabajan con una población de soluciones diversas (individuos).
- **Herencia**: las soluciones hijas heredan características de sus padres, similar al proceso de herencia genética en la evolución.
- **Selección:** la seleccionan las mejores soluciones en función de una función de aptitud, al igual que la selección natural favorece a los individuos más aptos en la naturaleza.
- **Mutación**: los algoritmos genéticos incorporan mecanismos de mutación para introducir variabilidad, de manera similar a cómo las mutaciones se introducen variabilidad genética en las poblaciones naturales.

Comprender la teoría de la evolución, se puede apreciar cómo los algoritmos genéticos invitan estos procesos naturales para encontrar soluciones a problemas complejos en el ámbito de la organización y la búsqueda.

Componentes:

- **Individuo:** representa una solución posible al problema.
- **Población:** conjunto de individuos.
- **Genes**: variables de decisión que componen el individuo.
- **Función de aptitud:** evalúa la calidad de una solución.
- **Operadores genéticos:** mecanismos para generar nuevos individuos (selección, cruce, mutación).

Pasos básicos:

1) Se genera una población inicial de individuos aleatoriamente.
2) Se volvió a la actitud de cada individuo de la población usando la función de aptitud.
3) Se selecciona y dibujos para ser padres, basándose en su aptitud.

4) Se combinan los genes de los padres para crear individuos nuevos (hijos).
5) Se aplica en pequeñas alteraciones a los hijos.
6) Los individuos de la nueva generación reemplazan a la generación anterior.
7) El algoritmo termina cuando se cumple un criterio de parada (número máximo de generaciones, convergencia, etc).

Los algoritmos genéticos son especialmente útiles para problemas para los cuales no existe un algoritmo exacto o para los que es demasiado costoso computacionalmente encontrar la solución óptima. Son aplicables a una amplia variedad de problemas, desde utilización de funciones hasta aprendizaje automático, diseño de sistemas, y muchos otros.

En nuestro problema la solución fue tras 40 generaciones.

```
La mejor solución es en el punto: (0.9447035202926853, 0.8914823717655311)
```

Video Algoritmos genéticos en 5 minutos:
https://www.youtube.com/watch?V=rbrxgyo0kiw

6.2. Optimización enjambre de partículas

La técnica de enjambre de partículas es un método de optimización ahoritica basado en el comportamiento colectivo de enjambres, como bancos de peces o bandadas de aves. Cada partícula representa una solución potencial en el espacio de búsqueda. Las partículas se mueven en este espacio según una serie de ecuaciones matemáticas que consideran tanto su posición actual como las posiciones de sus vecinas.

En cada interacción del algoritmo:

1) Cada partícula evalúa su aptitud utilizando la función objetivo.
2) Se compara la aptitud de la partícula con su mejor posición conocida. Si la nueva posición tiene una aptitud mejor, se actualiza la mejor posición individual de la partícula.
3) Se compara la mejor posición individual de la partícula con la mejor posición global del enjambre. Si es mejor, se actualiza la mejor posición global.
4) Cada partícula ajusta su velocidad y posición en función de sus propias experiencias y las del enjambre, generalmente usando un modelo que considera la inercia, la memoria cognitiva y la memoria social.

El algoritmo termina cuando se alcanzan criterios de paradas predefinido, como un número máximo de interacciones o una tolerancia de error.

La técnica es popular en aplicaciones que incluyen optimización de funciones, aprendizaje automático y robótica, entre otras. Es especialmente útil cuando el espacio de búsqueda es grande y las soluciones exactas son computacionalmente caras de obtener.

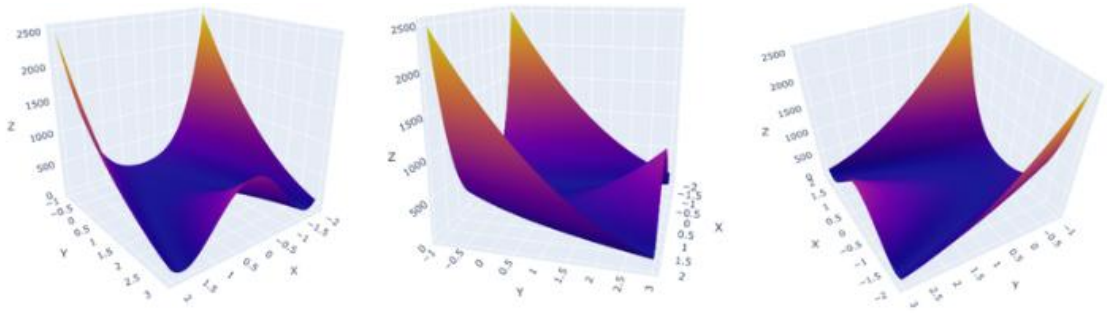
En nuestro problema la solución fue:

```
La mejor solución es en el punto: (0.99999751, 0.999996)
```

6.3. Algoritmo del descenso del gradiente.

El algoritmo del descenso del gradiente es de tipo iterativo, y una parte matemática dura, pero fácilmente explicable y entendible. Solo tienes que poseer la experiencia de caminar por el campo en la montaña.

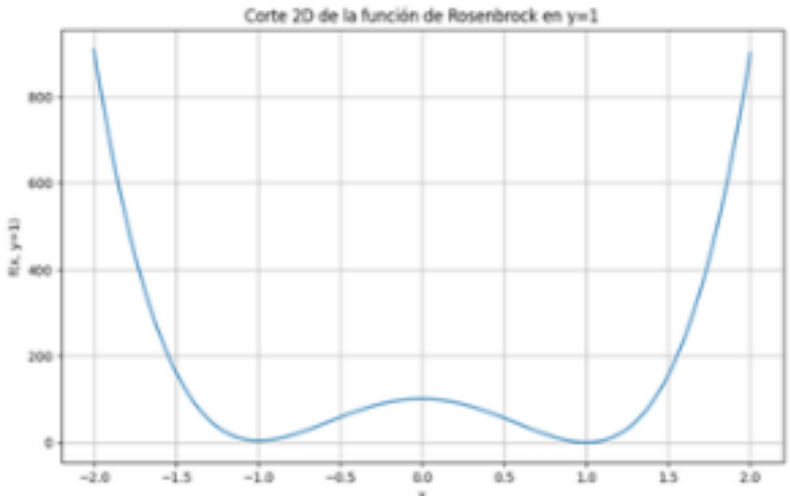
Antes vamos a echar un vistazo a nuestra función objetivo, qué es la que se muestra a continuación.



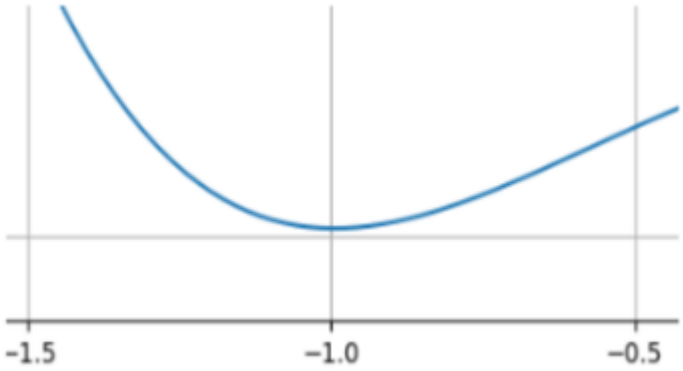
Piensa en esta figura tridimensional como un sistema montañoso, con pico, laderas y valles; e imagina que te dejo en un punto de esta gráfica con los ojos vendados. No puedes ver, solo puedes desplazarte y sentir, por medio de tus pies, si estás en una pendiente arriba o pendiente abajo.

¿cómo encuentras el Valle? ¿Cómo sabes si el Valle que encuentras es el más profundo? ¿Cómo encuentras el Valle más profundo?

Vamos a ver un corte de esta última gráfica que puede ser más esclarecedor:



Dado que estamos aprendiendo, podemos adelantar que la solución es [1,1]; el corte de la función con y=1 nos muestra una gráfica más simple en donde está la solución, a la derecha. ¿Por qué no a la izquierda en el punto [-1,1]? Apenas se perciben, pero (-1,1) es un valle, pero menos profundo que (1,1). Este último al ser el más profundo de todos se llama **mínimo global**, mientras que cualquier otro Valle que no sea el mínimo global se denominará **mínimo local**.



¿Cómo encuentras un Valle? Cuando la pendiente sea cero. ¿Cómo sabes si el Valle que encuentras es el más profundo? Simplemente no lo sabes. ¿Cómo encuentras el Valle más profundo? Usando el descenso del gradiente.

Te recuerdo que estás en un punto cualquiera, con los ojos vendados, solo puedes desplazarte por la línea sintiendo la pendiente bajo tus pies. Si la pendiente es hacia abajo sentirás un número negativo, si la pendiente es hacia arriba sentirás un número positivo. Si estás en el Valle o en la cima de una colina, sentirás cero como pendiente.

El descenso del gradiente consiste en ir avanzando - paso a paso- hacia abajo, mientras sientas una pendiente negativa y buscar ese punto donde sientas que la pendiente es cero. Cada paso que das es pequeño, si das trancadas grandes puedes que pases de una pendiente hacia abajo a una hacia arriba en un solo paso saltándote el Valle. Así que poco a poco vas descendiendo, buscando ese punto con pendiente cero. Sí descendiendo das un paso que te lleva una pendiente positiva, das la vuelta y desciendes dando pasos más pequeños.

Esa es la analogía del descenso del gradiente. Vamos ahora las matemáticas.

La pendiente es la derivada, y dado que f poseen dos variables independientes, tiene dos derivadas posibles:

$$\frac{\partial f(x,y)}{\partial x} = -2(a-x) - 4bx(y-x^2)$$

$$\frac{\partial f(x,y)}{\partial y} = 2b(y-x^2)$$

Estas funciones nos entregan la pendiente -el gradiente- en el punto (x, y), tanto hacia adelante/atrás cómo hacia los lados. Dado que en la figura de corte y es constante (igual a 1), nos movemos en una dirección (izquierda/derecha), pero realmente estamos manejando dos dimensiones, así que nos podemos mover izquierda/derecha y adelante/atrás.

Ahora que ya tenemos todo lo necesario, vamos a describir el método.

1) Selecciona un punto de partida (probablemente aleatorio) y establece los parámetros como la tasa de aprendizaje (el tamaño del paso), el número máximo de iteraciones (número máximo de pasos) y un criterio de convergencia (un valor tal que, si el movimiento que se produce es más pequeño, se supone que habrá alcanzado un mínimo).
2) En el punto actual, calcula los gradientes de la función objetivo, las pendientes.
3) Utiliza los gradientes para actualizar las coordenadas del punto actual en la dirección que minimiza la función (muévete hacia donde la pendiente disminuye).
4) Verifica si la actualización satisface un criterio de convergencia predefinido. Si se cumple, el algoritmo se detiene. De lo contrario, volvemos al paso dos.

5) Una vez alcanzada la convergencia el número máximo de iteraciones, tendrás el punto mínimo encontrado.

¿Encuentra un mínimo? Sí. ¿Es global? No lo sabemos. Pero hay formas de salir de mínimos locales y seguir buscando. Eso sí, no se garantiza que el mínimo encontrado sea global. Por algo el descenso del gradiente es un método de optimización aproximada.

```
La mejor solución es en el punto: (0.99999992 0.99999983)
```

Video El Descenso del gradiente: https://www.youtube.com/watch?V=a6ficdoz8_4

El descenso del gradiente es un elemento muy importante a la hora de entrenar una red neuronal artificial. Por la que profundizaremos más en ese capítulo.