

INTRODUCCIÓN AL ANÁLISIS DE DATOS

Ing. Max Cerna

Universidad Rafael Landívar

13 de enero de 2025





INTRODUCCIÓN AL ANÁLISIS DE DATOS

Universidad Rafael Landívar

Agenda

1. ¿QUE ES EL ANÁLISIS DE DATOS?
2. RAMAS CLASICAS
3. PROCESAMIENTO DE DATOS
4. ETL
5. BUSINESS INTELLIGENCE
6. DECISIONES
7. ROLES EN ANALISIS DE DATOS
8. DATA MANAGEMENT
9. DARK DATA

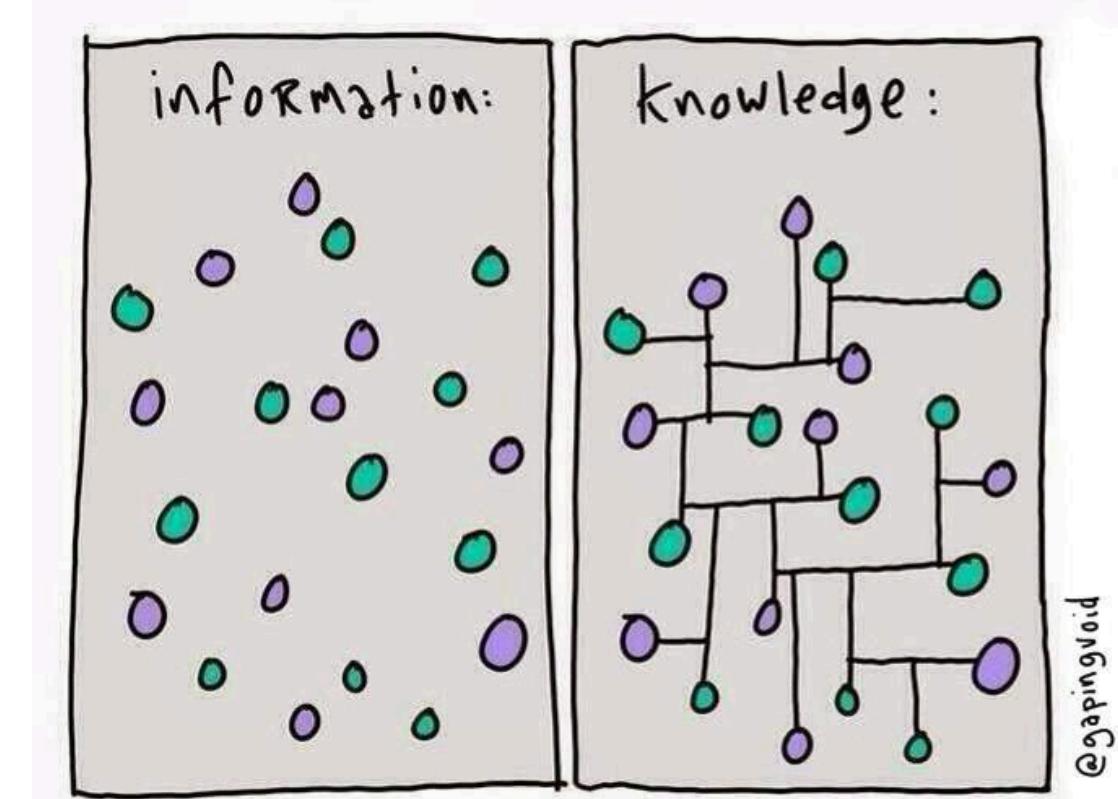
¿QUE ES EL ANÁLISIS DE DATOS?



¿QUE ES EL ANÁLISIS DE DATOS?

Concepto

- Es el proceso de examinar, limpiar, transformar e interpretar conjuntos de datos con el objetivo de descubrir información útil, extraer conclusiones relevantes y respaldar la toma de decisiones.



Qué es el análisis de datos



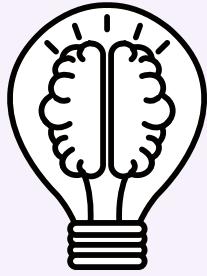
Datos

Son la unidad mínima con la que la empresa puede trabajar



Información

En el momento en el que distintos datos se procesan y se unen en un contexto determinado, podemos hablar de información



Conocimiento

Es el resultado de haber utilizado la información para obtener conclusiones a partir de la experiencia y del know-how de una persona experta en el área a la que corresponde a la información

Ramas Clásicas



Procesamiento Transaccional

Se centra en manejar grandes volúmenes de transacciones pequeñas en tiempo real



Procesamiento analítico

Este enfoque se utiliza para explorar, resumir y analizar grandes volúmenes de datos históricos

Procesamiento Transaccional (OLTP)

Proporcionan información confiable y precisa a los grandes almacenes de datos

Es un sistema de procesamiento de transacciones en línea que se utiliza para llevar a cabo operaciones cotidianas de una organización.

Las características más comunes de este tipo de transacciones son:

■ 01 **Altas/bajas/modificaciones**

■ 02 **Consultas rápidas, escuetas y predecibles**

■ 03 **Poco volumen de información y disagregación**

■ 04 **Transacciones rápidas**

■ 05 **Gran nivel de concurrencia Y baja redundancia de datos**

Procesamiento analítico (OLAP)

Se caracteriza por ser un análisis multidimensional de datos corporativos

Es una solución utilizada en el campo de Business Intelligence cuyo objetivo es agilizar la consulta de grandes cantidades de datos

- 01 Soportan requerimientos complejos de análisis
- 02 Analizan datos desde diferentes perspectivas
- 03 Soportan análisis complejos contra un volumen ingente de datos
- 04 se clasifican según las siguientes categorías: ROLAP, MOLAP y HOLAP
- 05 Ejemplos de MOLAP: Microsoft Analysis Services, SAP Business Warehouse

Diferencias OLTP y OLAP

OLTP

Los datos se atomizan



Los datos son
actuales



Procesa un registro
cada vez



Diseñada para el
proceso repetitivo
altamente
estructurado

OLAP

Los datos se resumen



Los datos son
históricos



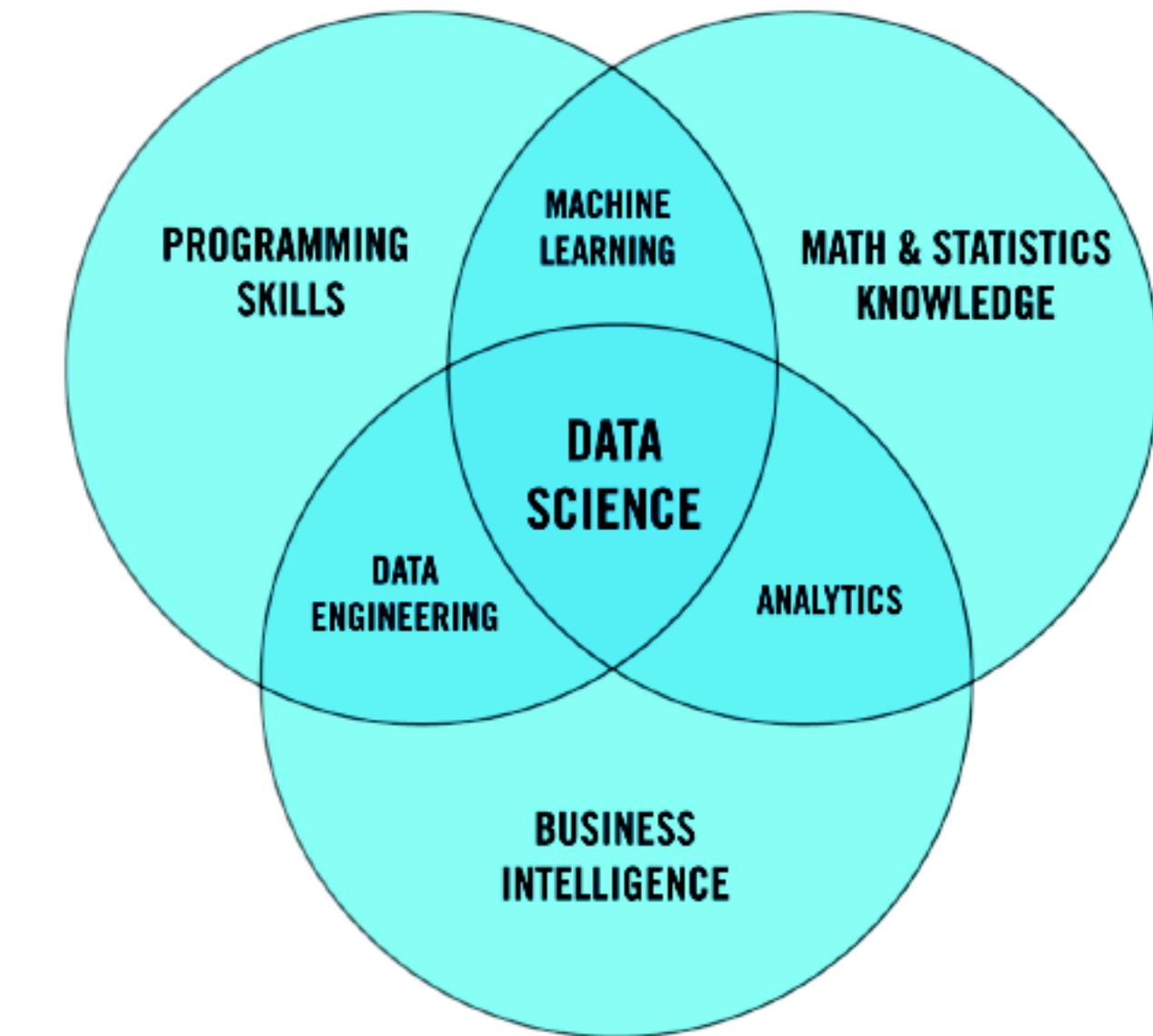
Procesa varios registros
simultáneamente



Diseñada para el proceso
analítico altamente
desestructurado

PROCESAMIENTO DE DATOS

Es el conjunto de actividades que se realizan para transformar datos crudos (sin procesar) en información significativa, útil y comprensible.



ETL

Extract, Transform, Load

Definicion

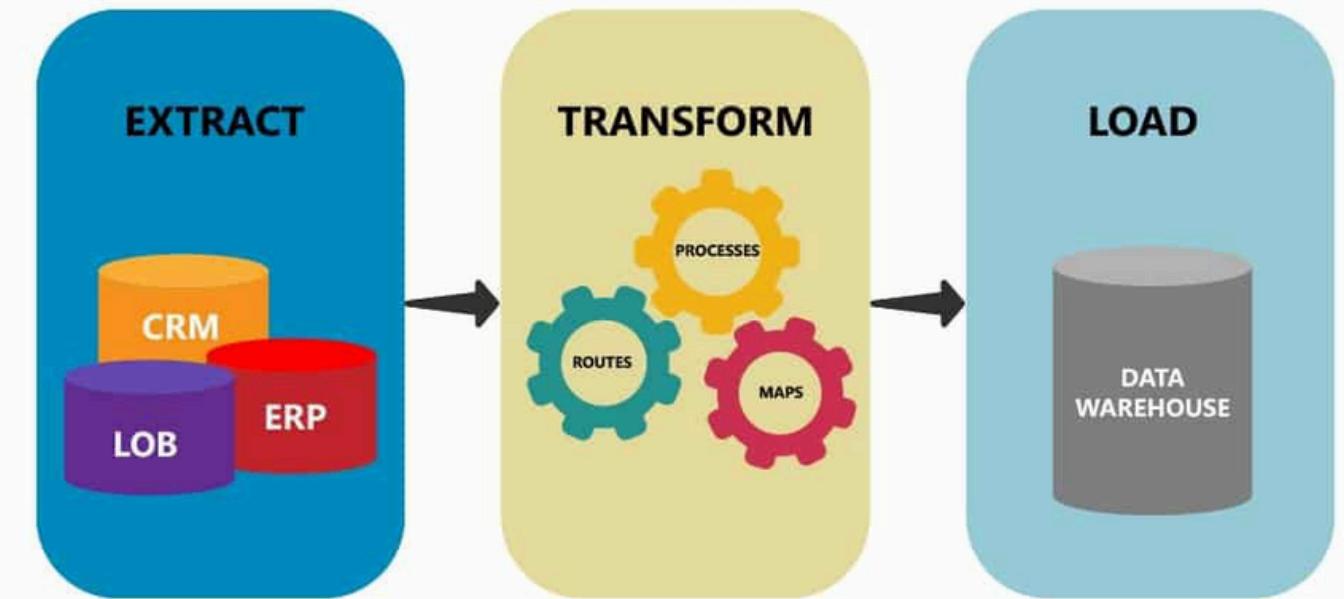
- Los datos se toman (extraen) de un sistema de origen, se convierten (transforman) en un formato que se puede almacenar y se almacenan (cargan) en un data warehouse, data lake u otro sistema.

Objetivo

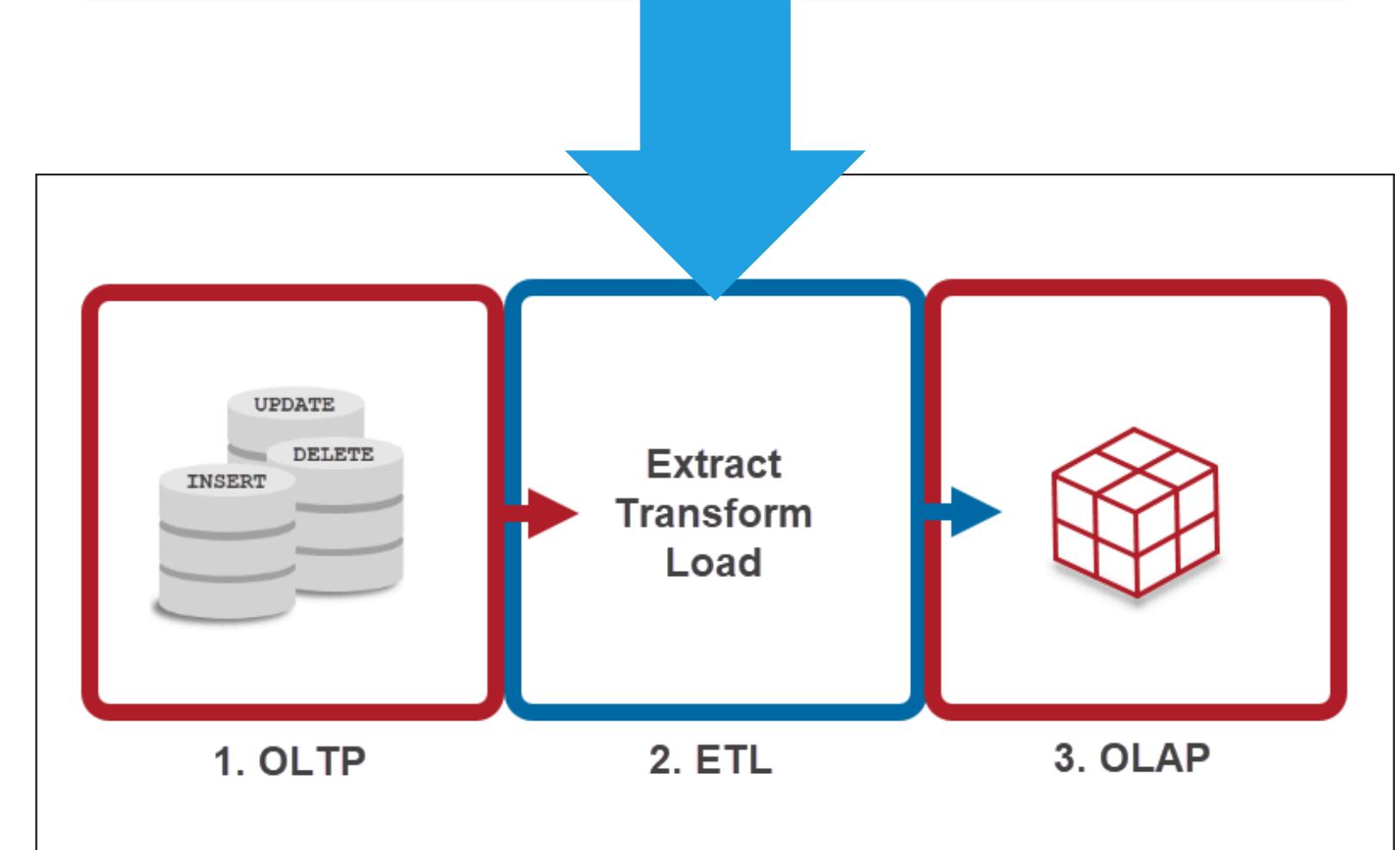
- Provee profundo contenido histórico para la empresa
- Esta tarea cause un impacto mínimo en el sistema de origen

ETL

Extract, Transform, Load

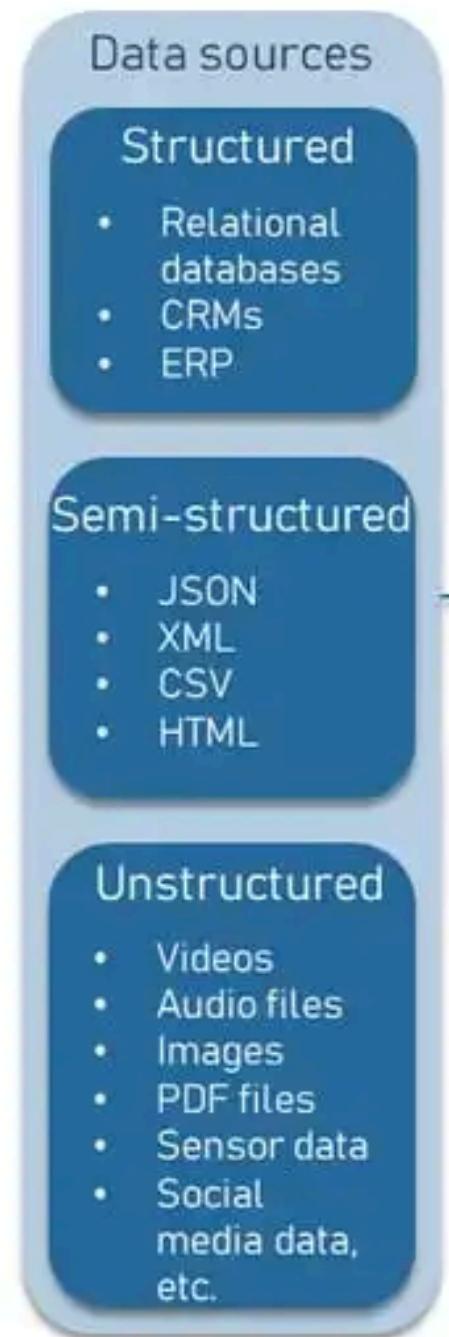


ETL - Extract, Transform, Load

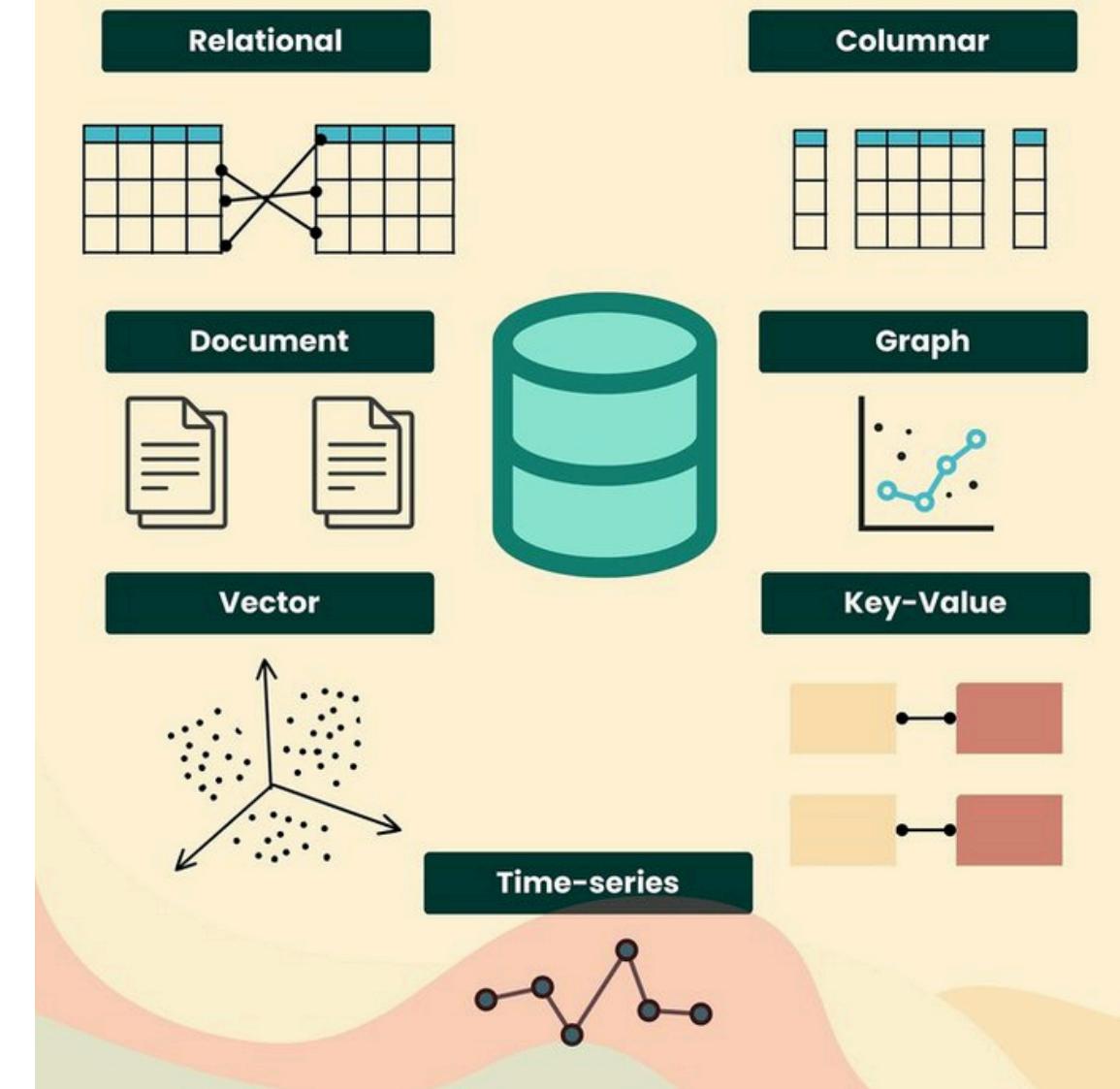


Tipos de Data Source (fuentes de datos)

Fuentes de datos utilizadas para procesos de ETL pueden ser: estructurados, semi-estructurados, sin estructura

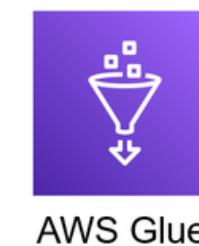


Types Of Databases



Herramientas de ETL

- Tools Comerciales



- Librerias



Business Intelligence



Business Intelligence

Es la habilidad para transformar los datos en información, y la información en conocimiento, de forma que se pueda optimizar el proceso de toma de decisiones en los negocios

Una solución completa de BI permite:

■ 01 **Observar - ¿Qué está pasando?**

■ 02 **Comprender - ¿Por qué ocurre?**

■ 03 **Predecir - ¿Qué ocurrirá?**

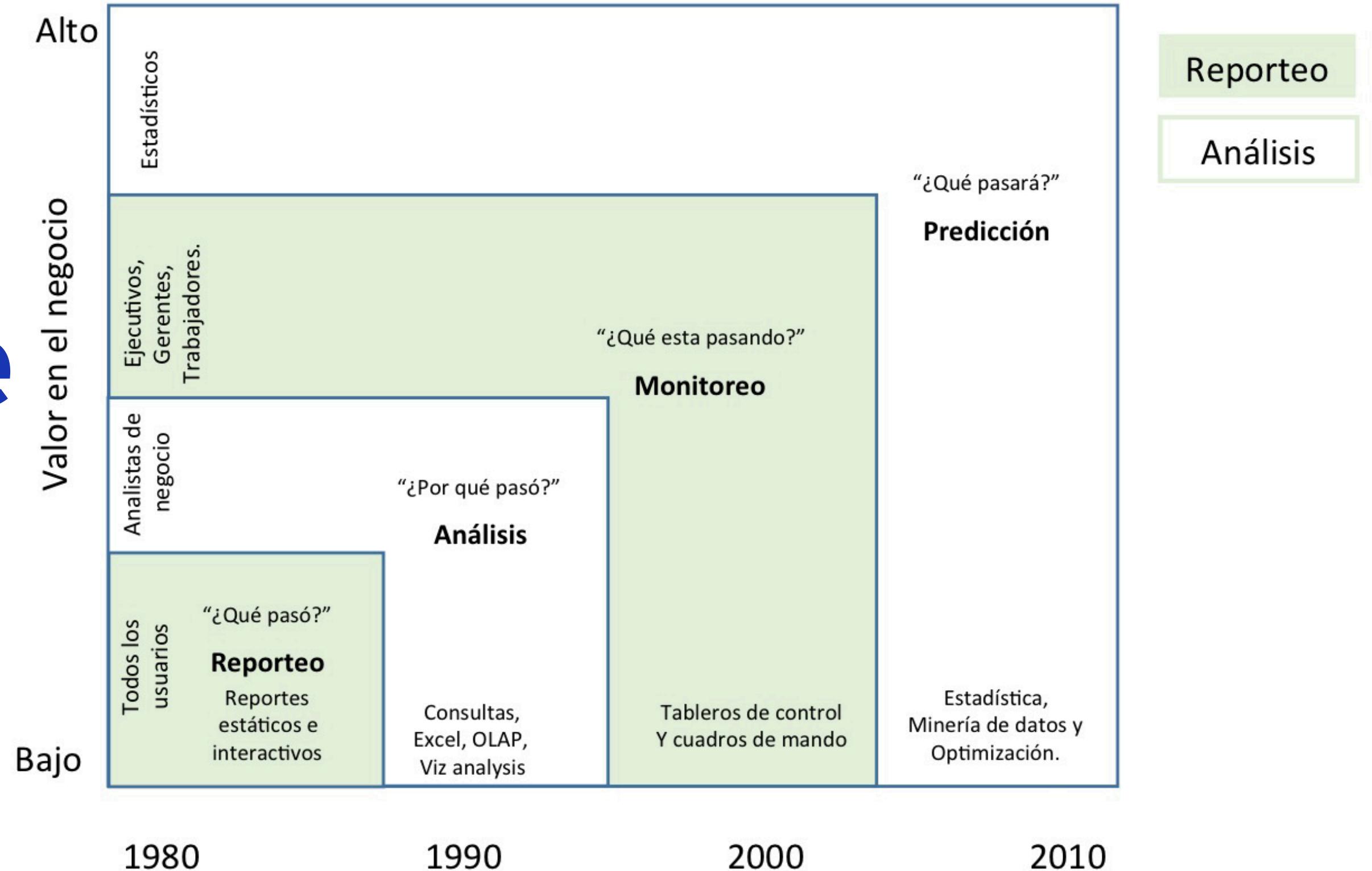
■ 04 **Colaborar - ¿Qué debería hacer el equipo?**

■ 05 **Decidir - ¿Qué camino se debe seguir?**

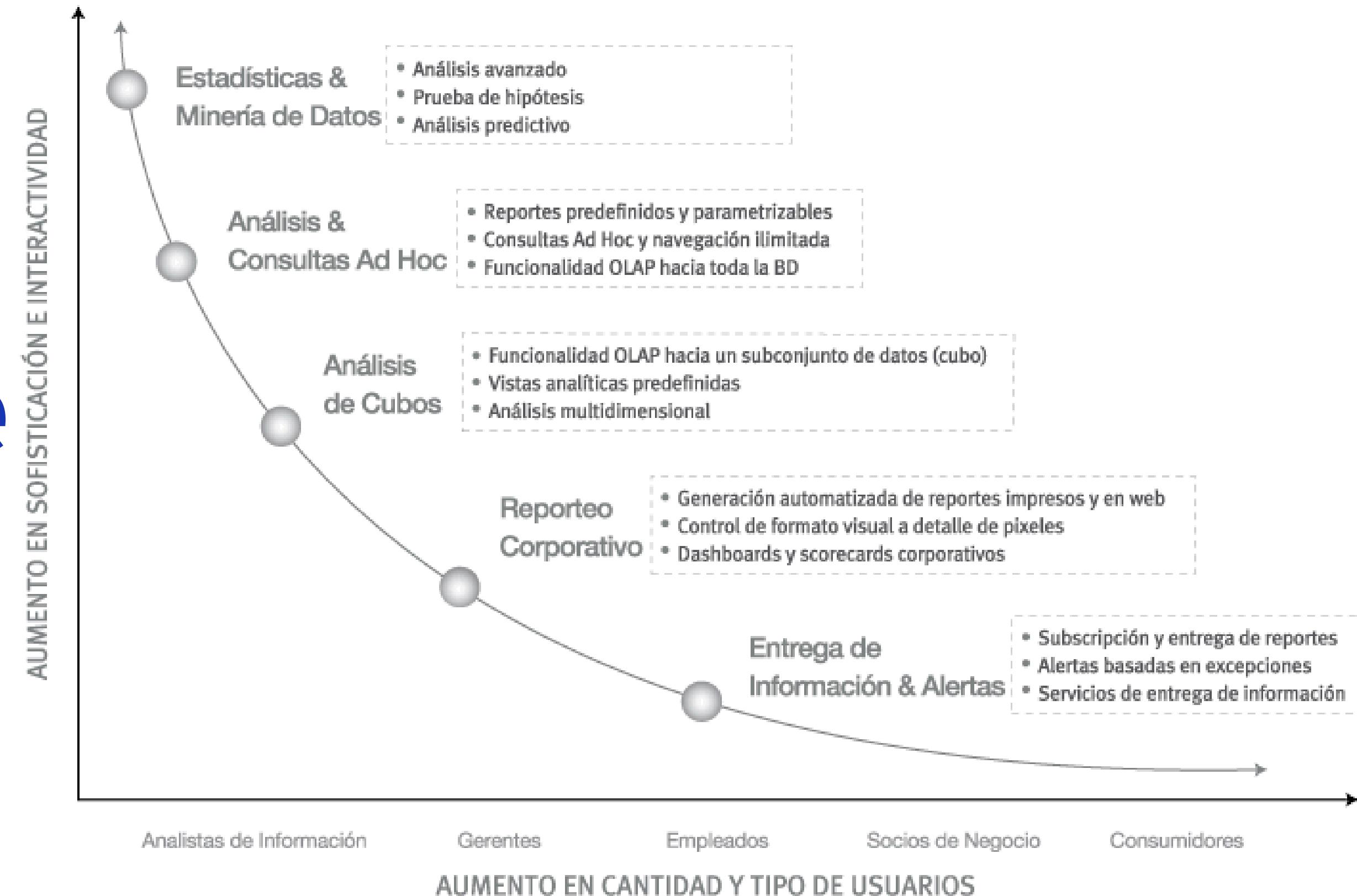
Business Intelligence

- 1980 - Sistemas de soporte de decisiones (DSS)
- 1990 - Data warehouse, BI
- 2000 - Dashboards, scorecards
- 2010 - Analytics, Big Data, Mobile BI, data science ..

Business Intelligence



Business Intelligence



¿Para que sirve?

01

Mejora los procesos de administración

02

Mejora procesos operacionales

03

Mejores ajustes ante cambios

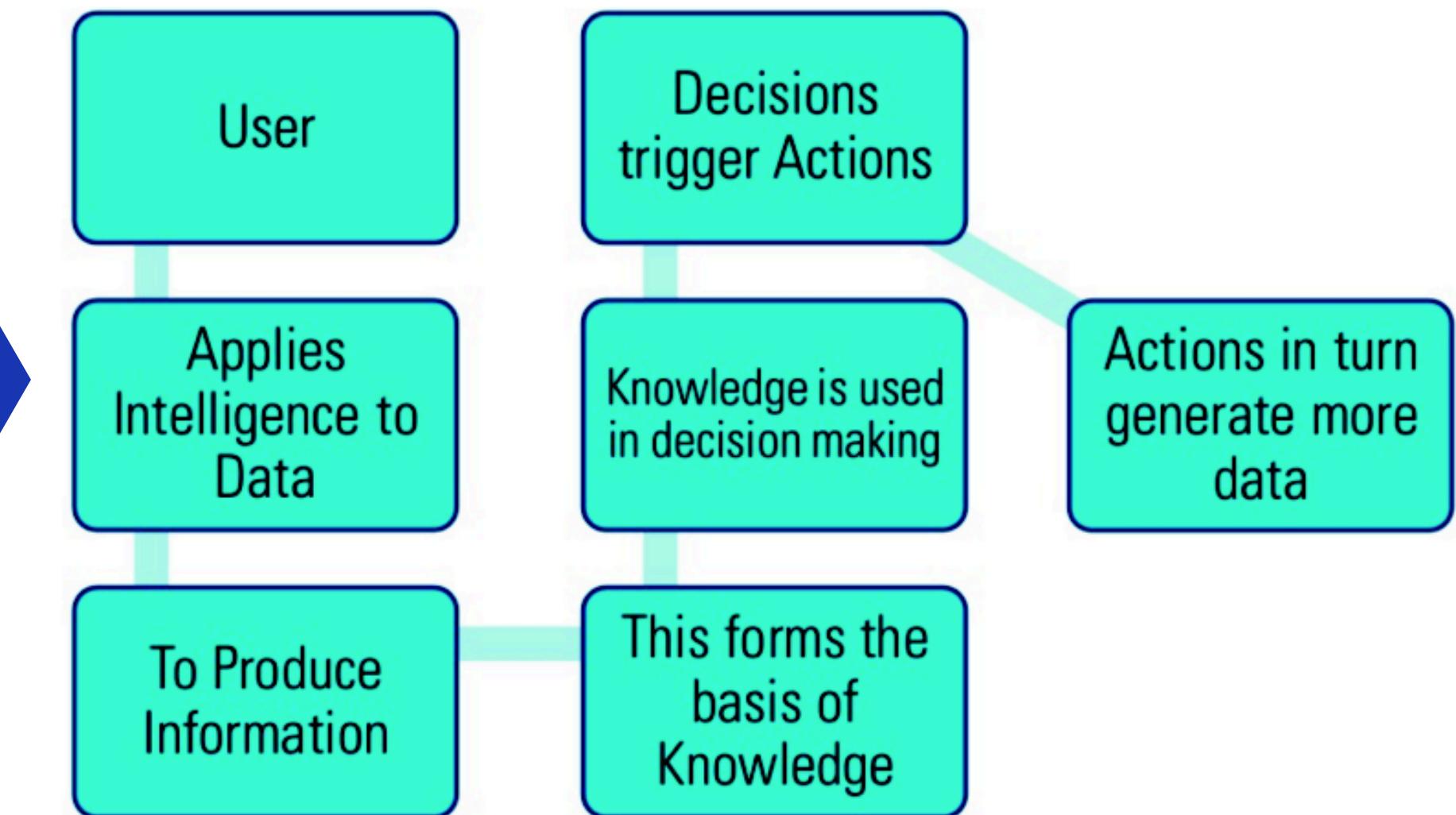
04

Mejores predicciones

05

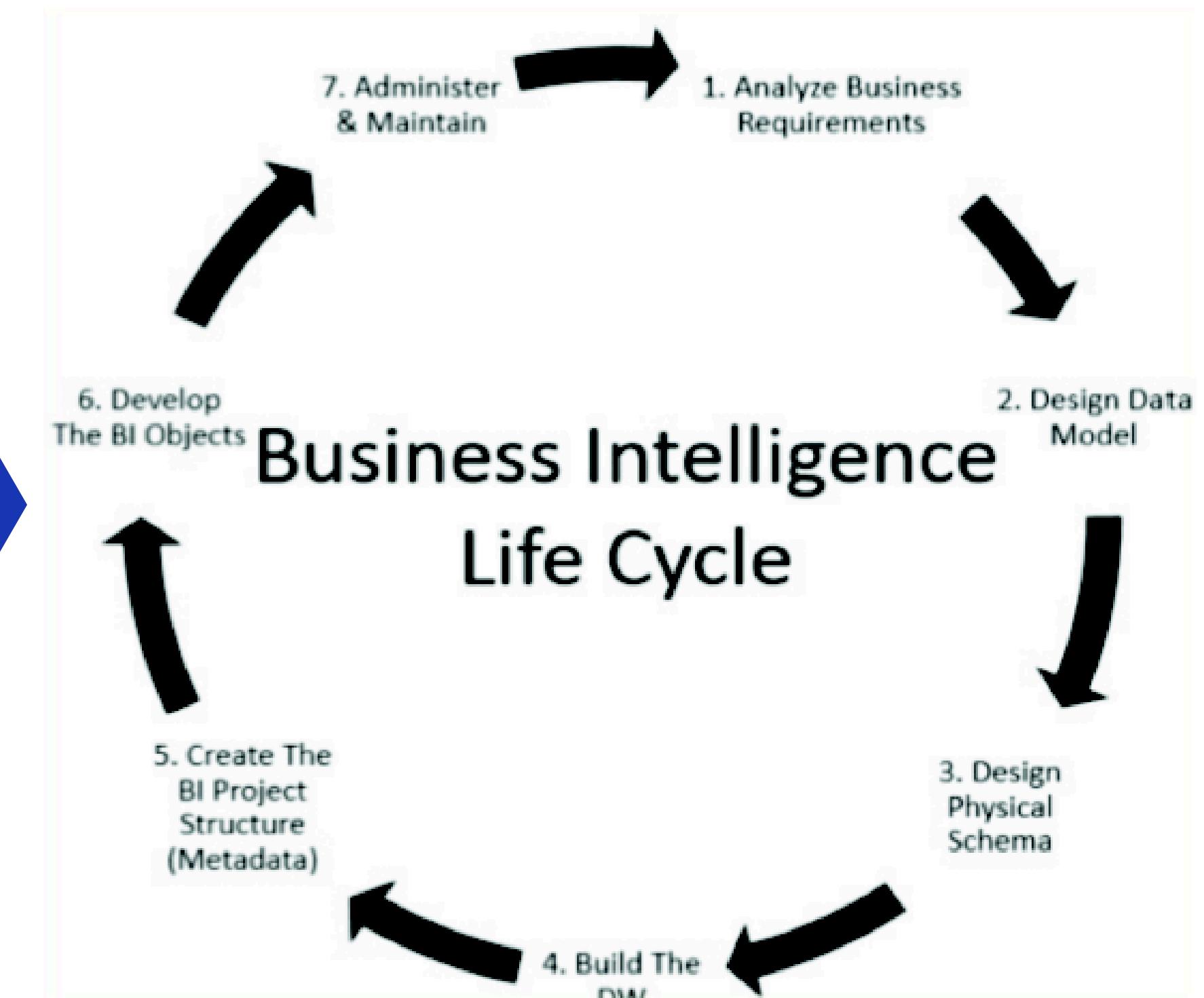
Mejora los análisis de riesgo

DECISIONES

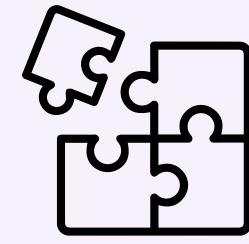


Decisiones

Ciclo de vida en Business Intelligence

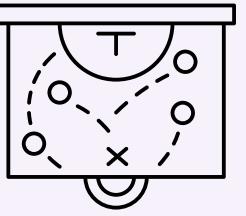


Tipos de Decisiones



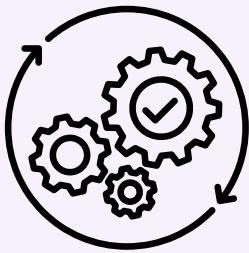
Estratégicas

definen "qué" quiere lograr la organización.



Tácticas

se centran en "cómo" implementar esas estrategias

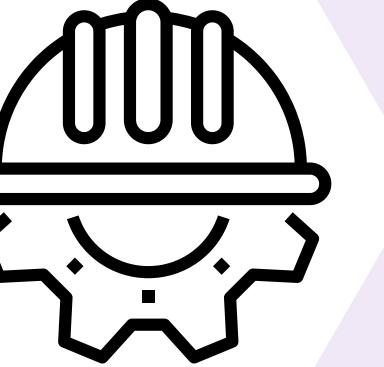


Operacionales

abordan "qué hacer" para que las operaciones diarias sean eficientes

Roles en el Análisis de los Datos

existen varios roles en el ámbito de los datos, cada uno con funciones específicas



Data Engineer

Diseña, construye y mantiene infraestructuras de datos, creando pipelines eficientes para la recolección, almacenamiento y procesamiento de grandes volúmenes de datos.



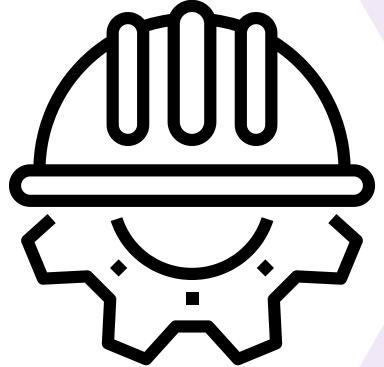
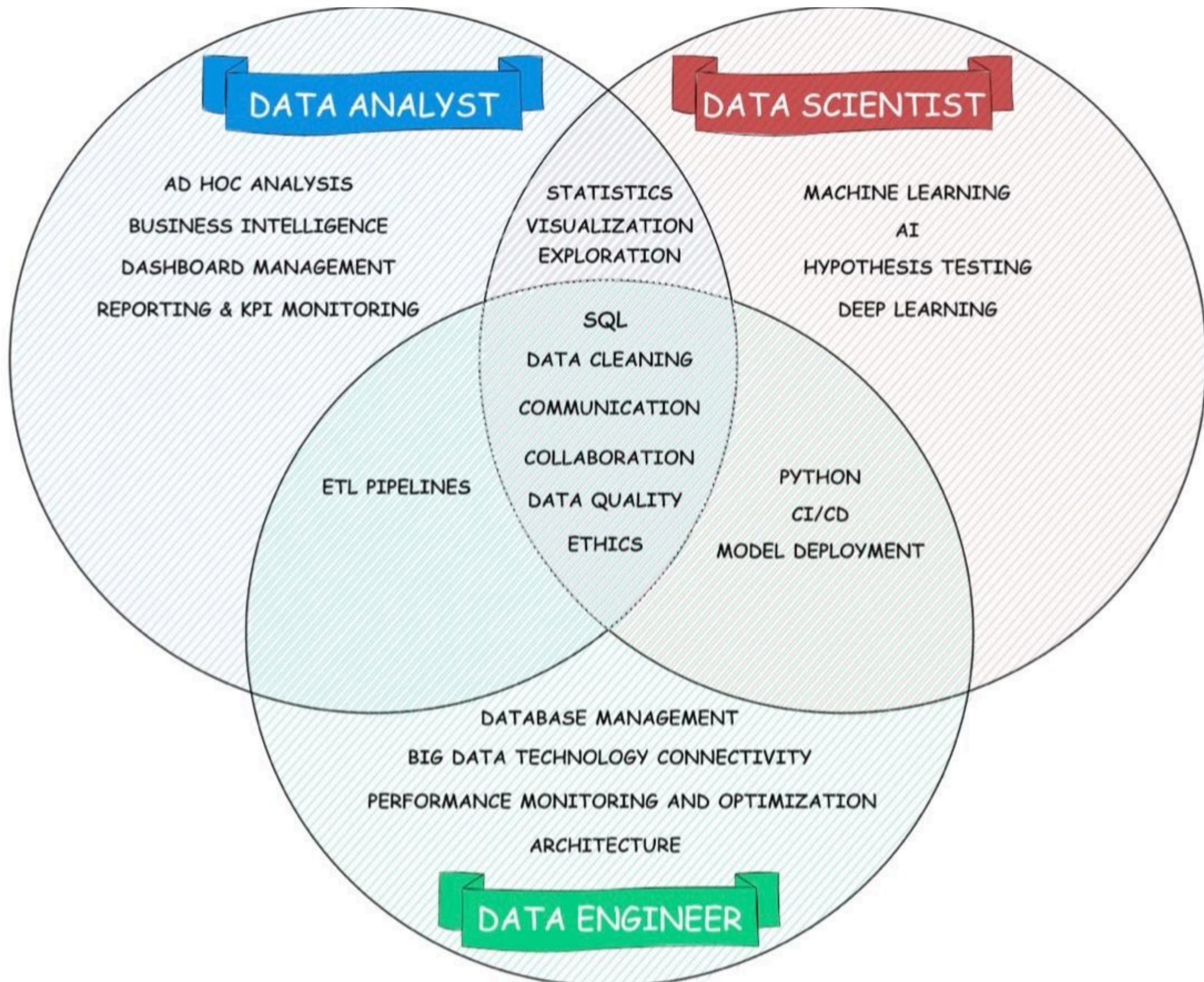
Data Scientist

Se especializa en la creación de modelos predictivos y el análisis de datos complejos usando machine learning y estadísticas avanzadas



Data Analyst

Se enfoca en analizar datos históricos y generar informes descriptivos, utiliza tools de BI para visualizar tendencias y ayudar en la toma de decisiones empresariales



Otros Roles

Algunos de los más comunes incluyen:

- **Data Architect:** Diseña y gestiona la arquitectura de bases de datos y soluciones de almacenamiento de datos a gran escala, asegurando que los sistemas sean eficientes, escalables y seguros.
-

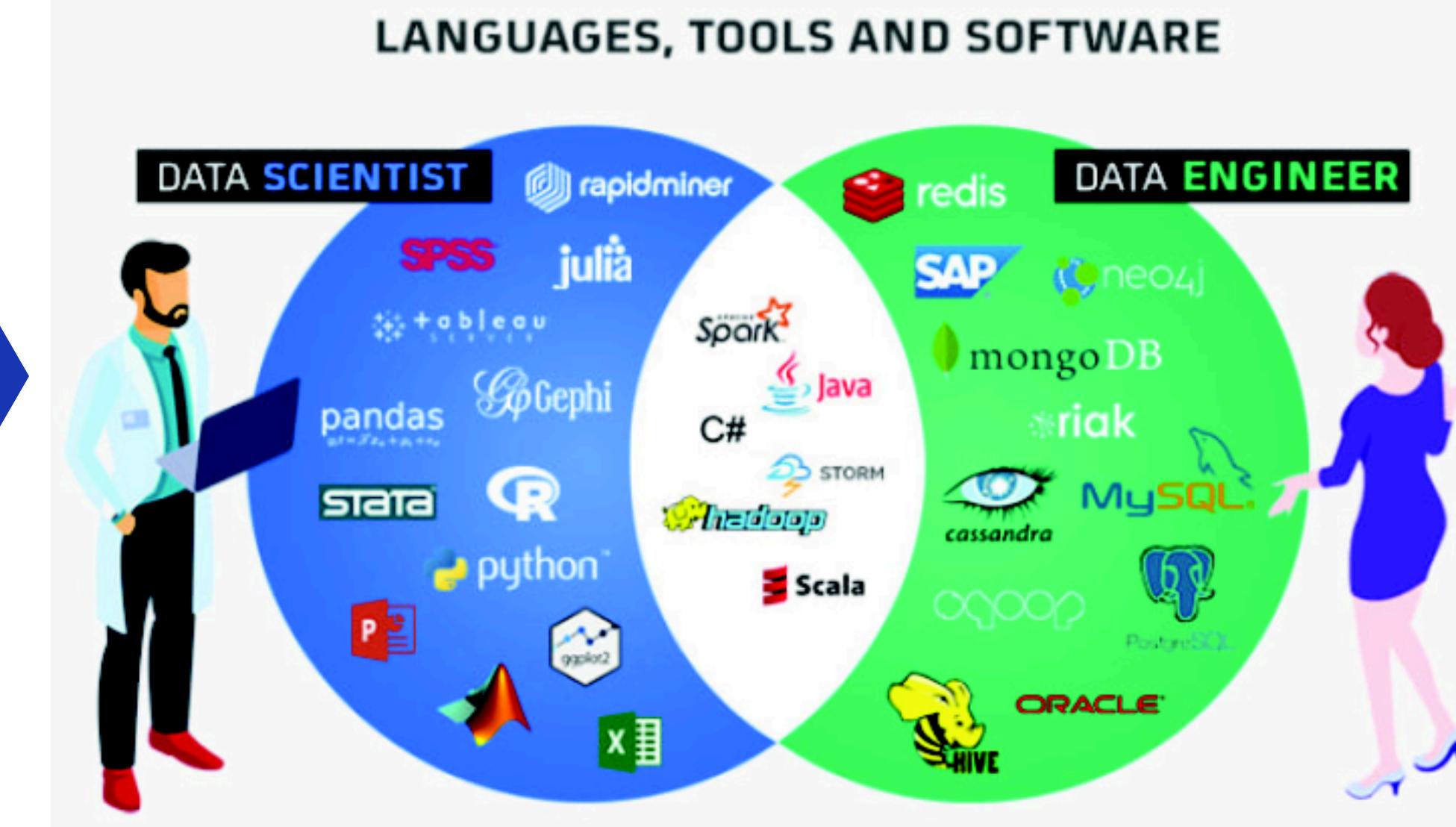
- **Machine Learning Engineer:** Se enfoca en la implementación y despliegue de modelos de aprendizaje automático.
-

- **Data Operations (DataOps) Engineer:** Se centra en la automatización y la gestión de las operaciones de datos
-

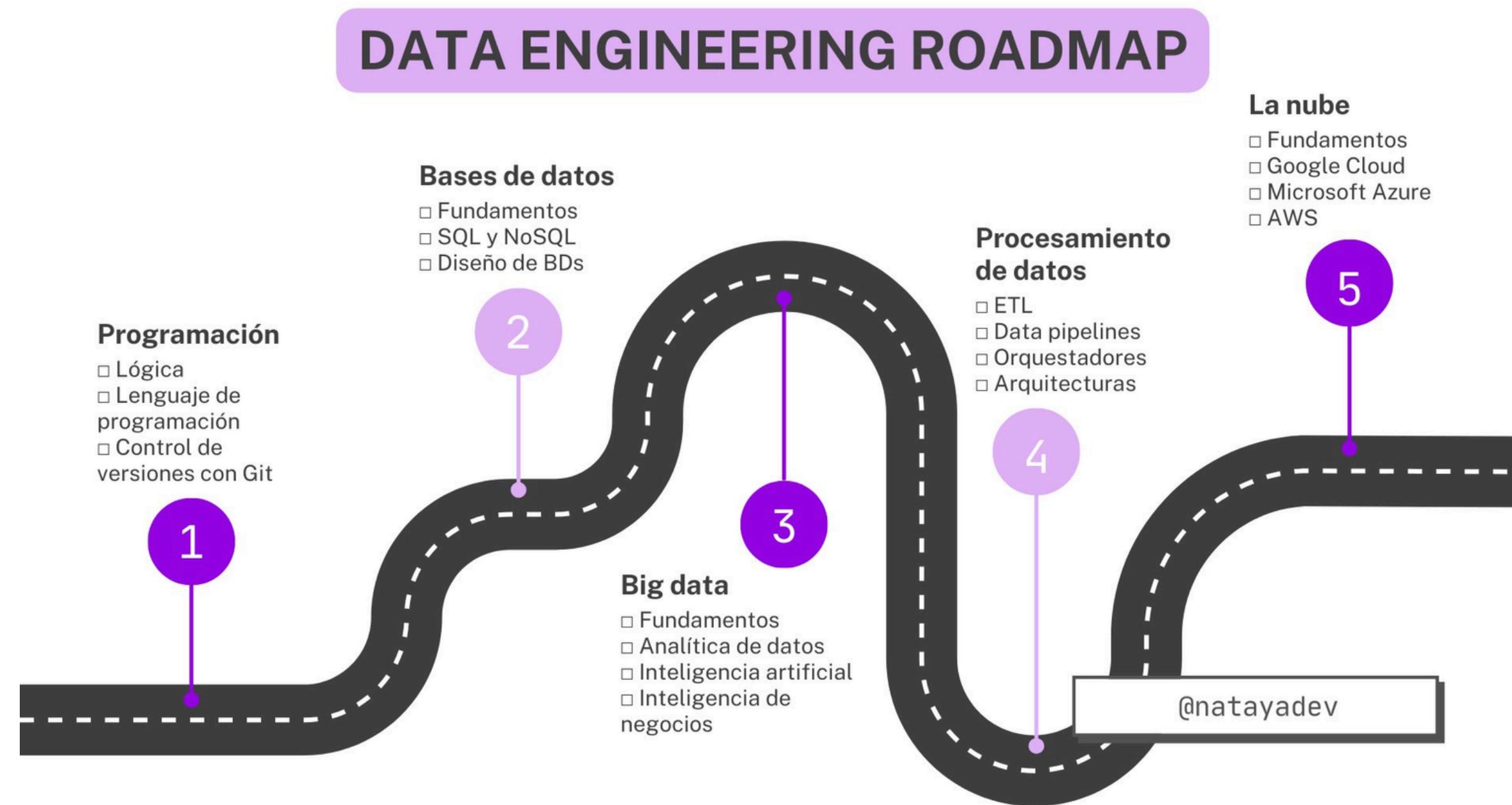
- **Big Data Engineer:** Especializado en el manejo de grandes volúmenes de datos utilizando tecnologías como Hadoop, Spark y otras plataformas de Big Data.
-

- **AI Engineer:** Desarrolla y mantiene sistemas de inteligencia artificial, incluyendo el diseño de algoritmos y la implementación de soluciones que permitan a las máquinas aprender y tomar decisiones autónomamente

Herramientas, lenguajes y software para Analisis de Datos



Data Engineer Roadmap



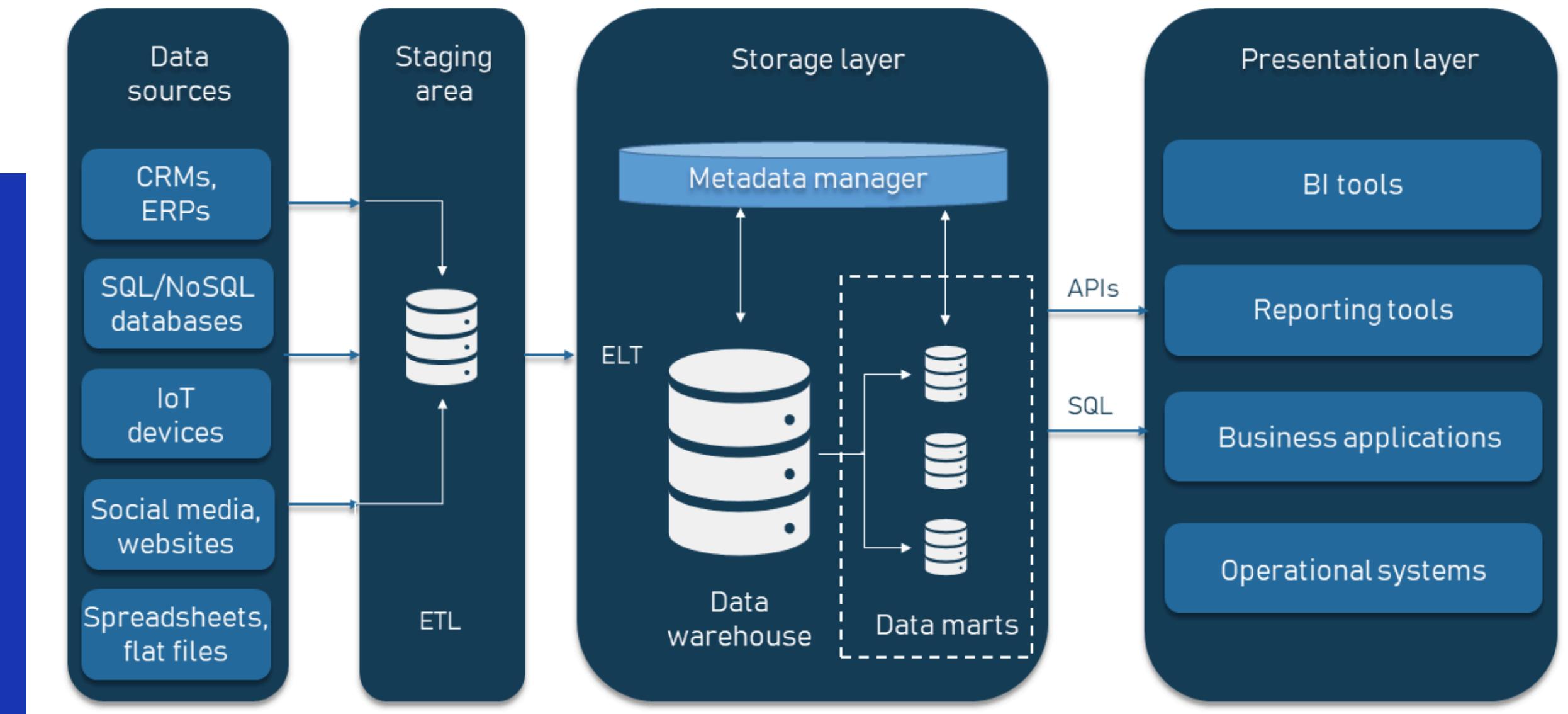
Data Management

Agregadores



Data Warehouse

Es un repositorio unificado y homogéneo para todos los datos que recogen los diversos sistemas de una empresa en su día a día

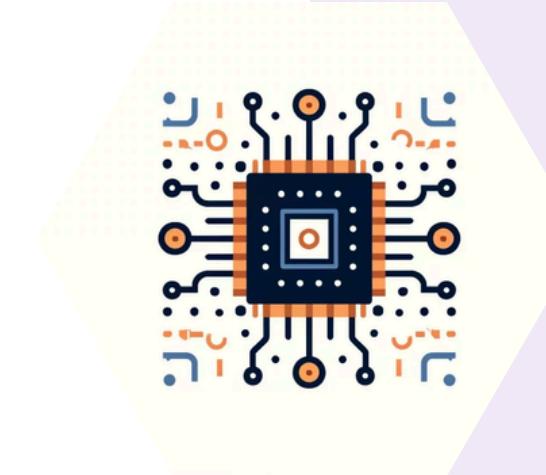


Data Management



Agregadores

Data Warehouse
Data Lake
Data Mart



Procesadores

ETL
Map-Reduce



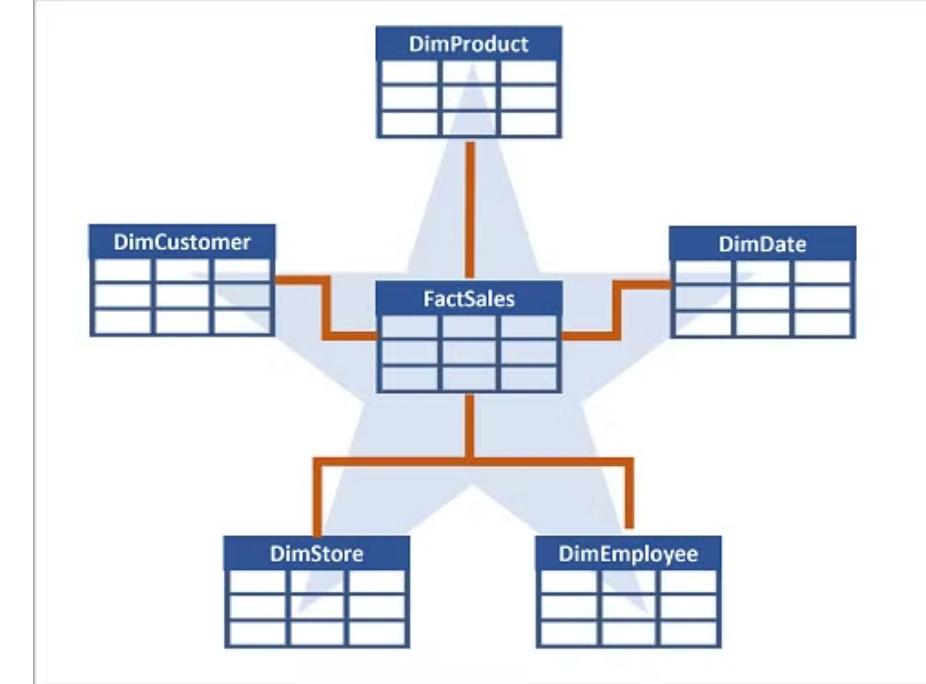
Analisis

Query o Reporting
OLTP/OLAP
Data Mining

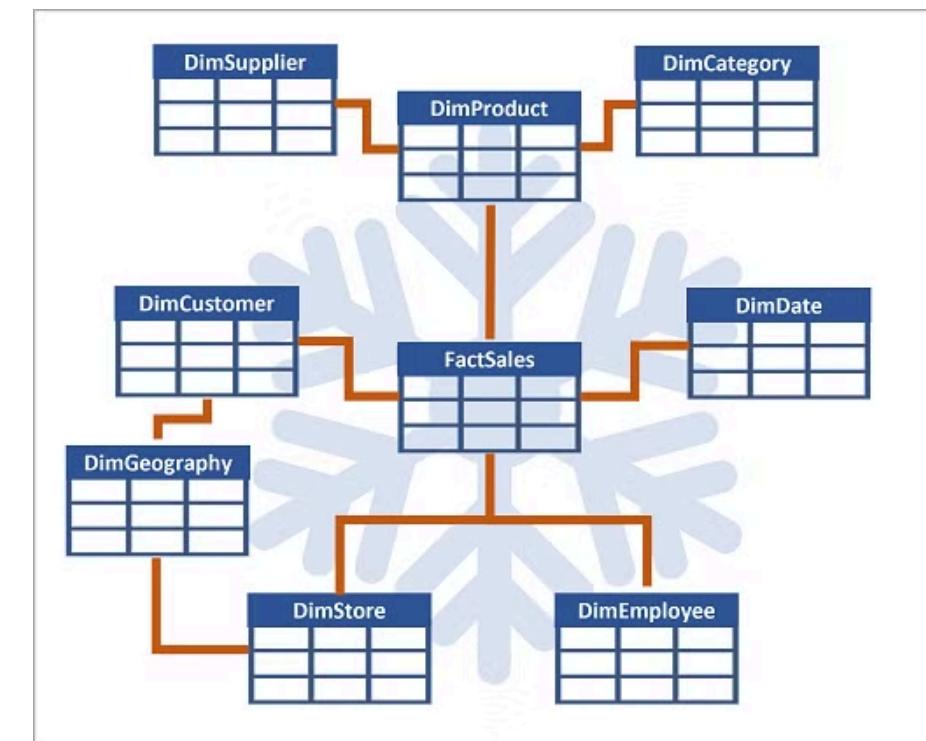
Modelo de Datos para Data Warehouse

El modelo debe reflejar los procesos principales del negocio y debe ser diseñado para soportar las necesidades de análisis de la empresa.

- **Modelo Estrella**



- **Copo de Nieve**



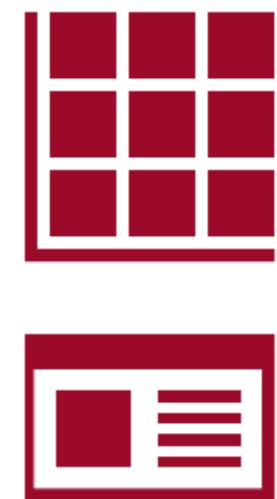
Data Management

Agregadores



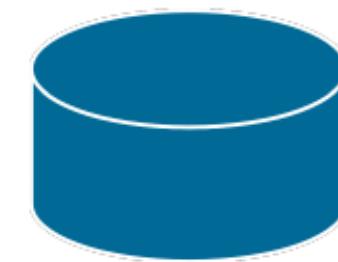
Data Mart

Es un subsistema de Business Intelligence que se utiliza para almacenar y acceder a un subconjunto específico de datos de una organización.

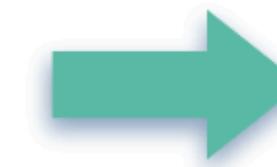


Fuentes de datos

ETL
Extracción, Transformación, Carga



Datawarehouse



Data Mart



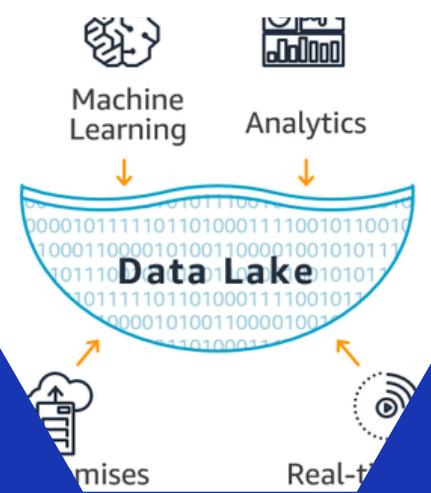
Data Mart



Data Mart

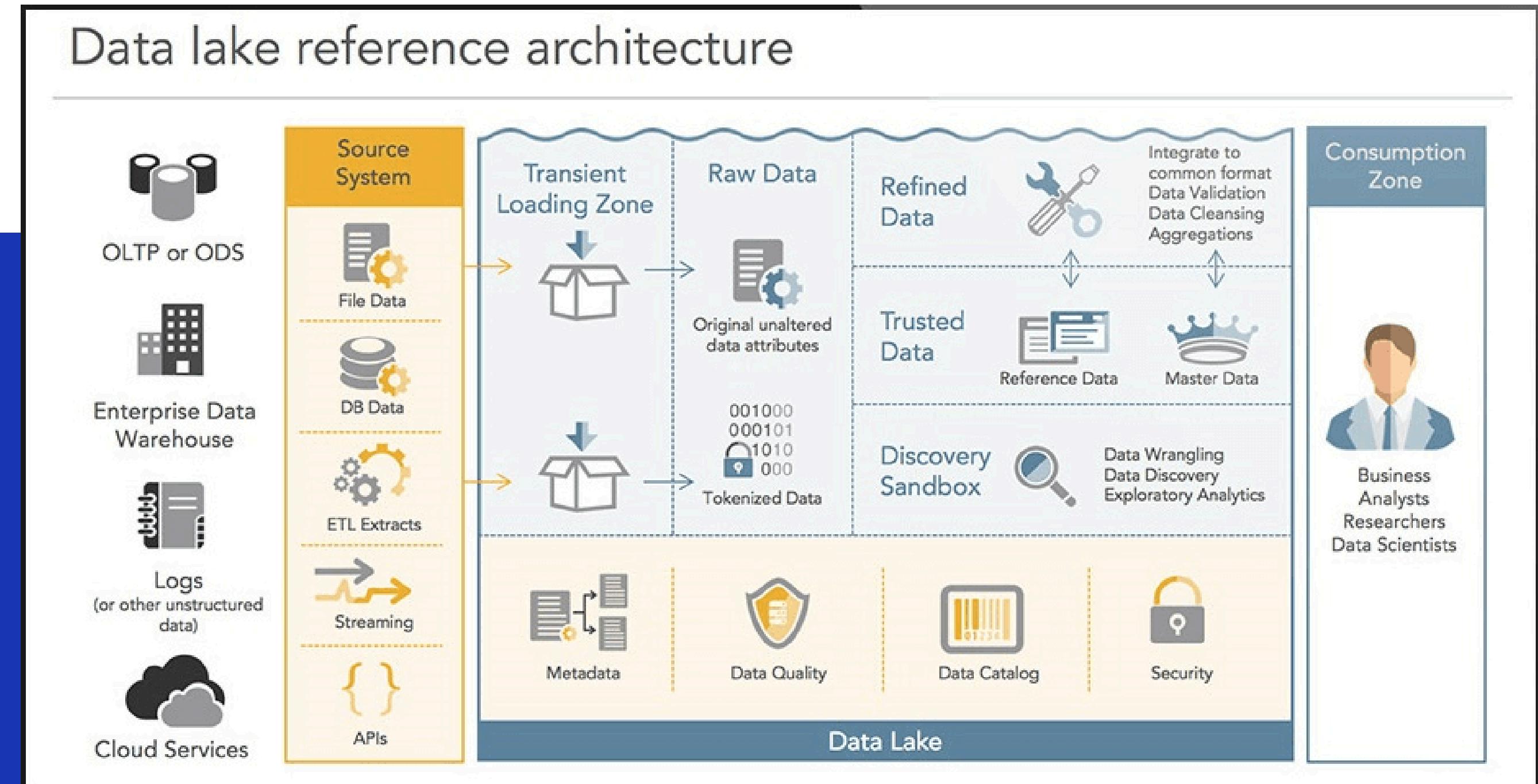
Data Management

Agregadores

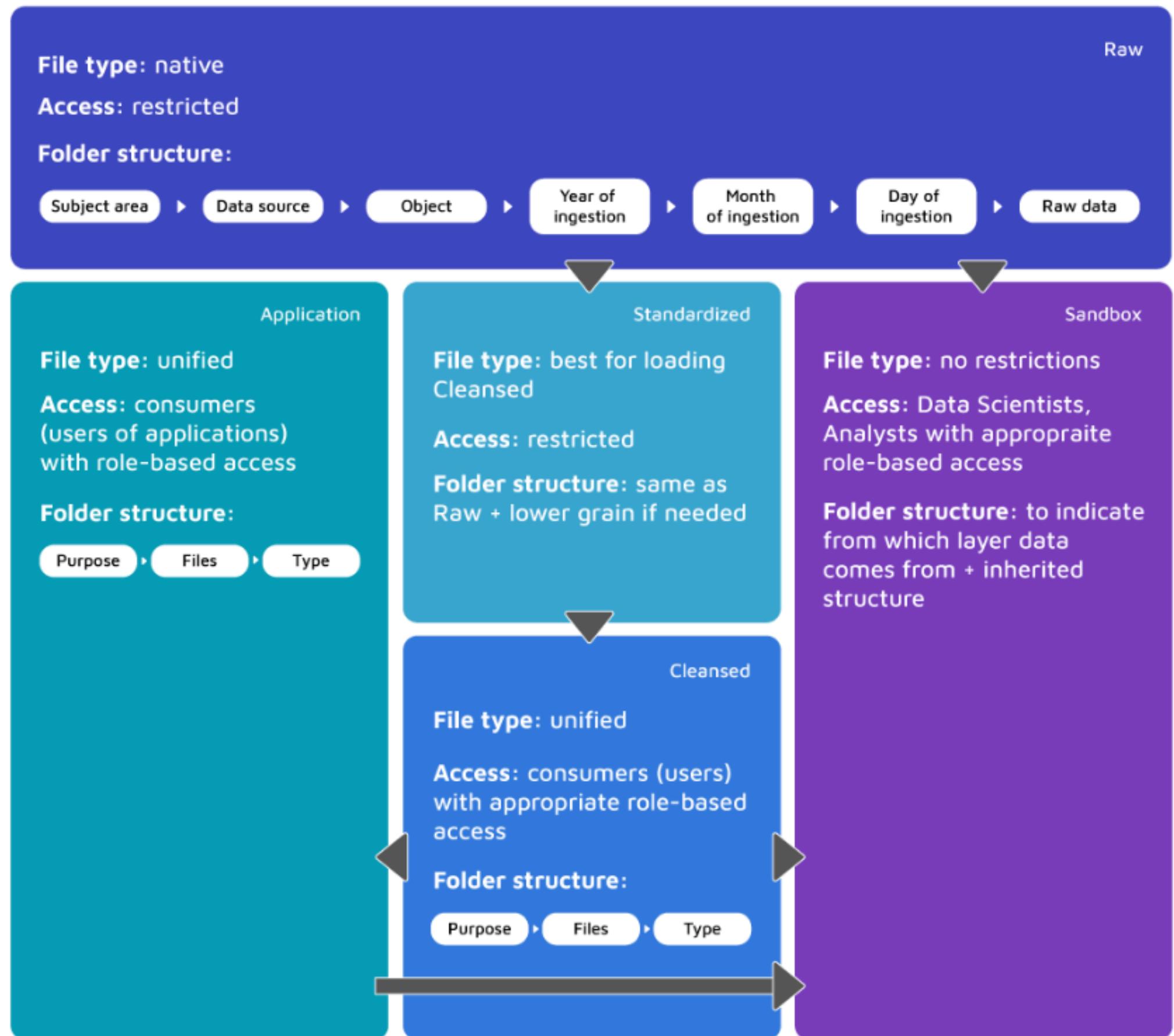


Data Lake

Es un repositorio de almacenamiento que contiene una gran cantidad de datos en bruto y que se mantienen allí hasta que sea necesario



Capas de Data Lake



Capa de datos sin procesar (Raw)

- No se permite ninguna transformación en esta etapa
- No se permite anular, lo que significa manejar duplicados y diferentes versiones de los mismos datos

Capa de datos limpios (Cleaned)

- Los datos se transforman en conjuntos de datos consumibles y se pueden almacenar en archivos o tablas
- Ya se conoce el propósito de los datos, así como su estructura en esta etapa

Capa de datos de la aplicación

- Capa utilizada por aplicaciones, donde algunas de esas aplicaciones utilizan modelos de machine learning
- También denominada capa de confianza/capa segura/capa de producción

Capa de datos estandarizada

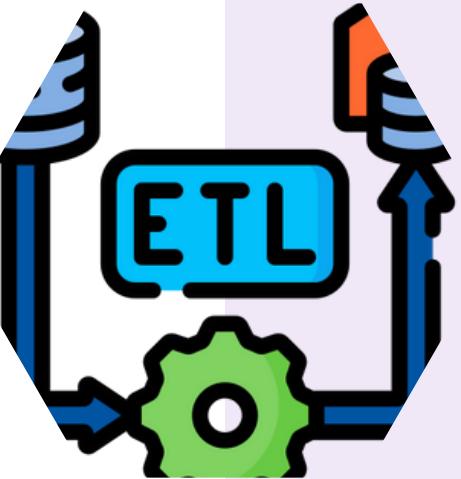
- El objetivo principal de esta capa es mejorar el rendimiento en la transferencia de datos de sin procesar a limpios

Capa de datos sandbox

- Está destinada al trabajo de analistas avanzados y científicos de datos
- Pueden realizar sus experimentos al buscar patrones o correlaciones.

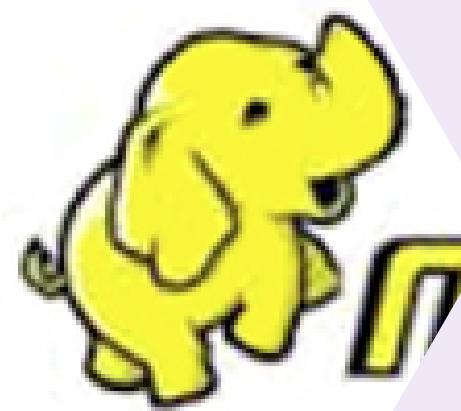
Data Management

Procesadores



ETL o ELT

Pentaho, AWS Glue, Azure Factory, etc



Map-Reduce

Hadoop

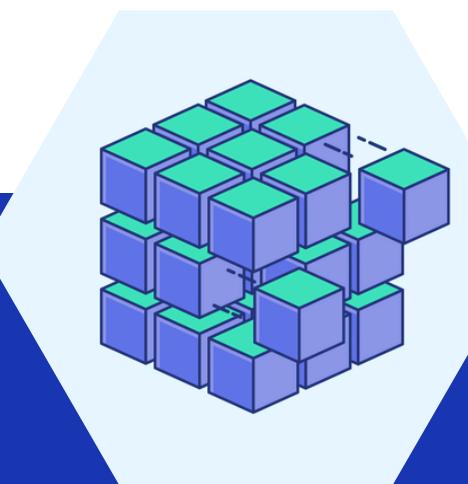
Data Management

Analisis



Query y reporting

responden la pregunta: **que
pasó?**



OLAP/OLTP

responden la pregunta: **que
pasó y quien/que estuvo
involucrado?**



Data Mining

responden la pregunta: **que
podría pasar?**

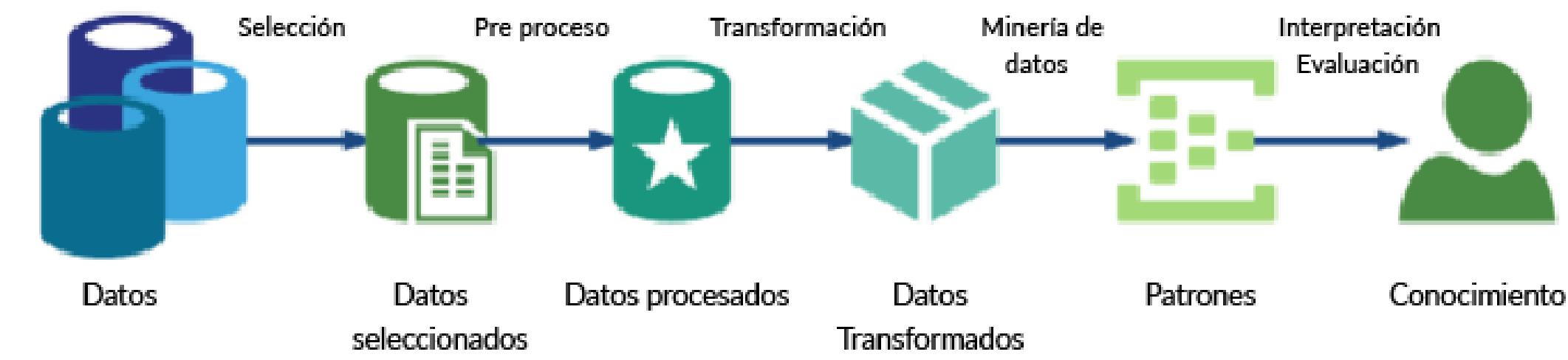
Data Management

Analisis



Data Mining

Analiza bases de datos automáticamente utilizando diversas técnicas y tecnologías para identificar patrones recurrentes o tendencias



Proceso de Knowledge Discovery in Databases (KDD)

Data Mining

Para llevar a cabo un análisis de Data Mining, se debe realizar cuatro etapas:



Técnicas de Data Mining

Técnicas de Data Mining

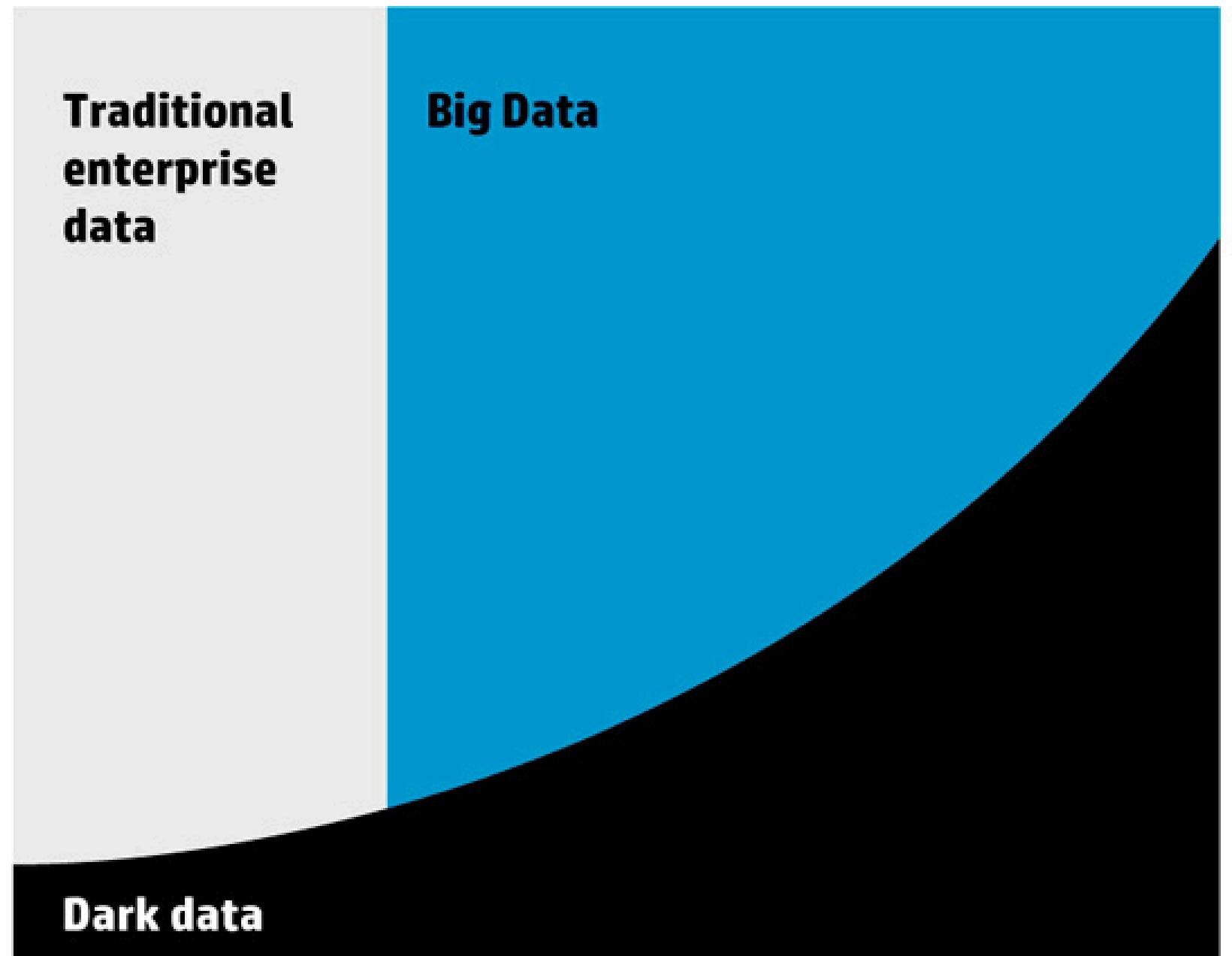
Predictivas	<ul style="list-style-type: none">▶ Regresión▶ Análisis de Varianza y Covarianza▶ Series Temporales▶ Métodos Bayesianos▶ Algoritmos Genéticos	
Descriptivas	<ul style="list-style-type: none">▶ Clasificación Ad hoc▶ Discriminante▶ Árboles de decisión▶ Redes Neuronales	Descubrimiento
Descriptivas	<ul style="list-style-type: none">▶ Clasificación Post hoc▶ Clustering▶ Segmentación	
Técnicas auxiliares	<ul style="list-style-type: none">▶ Asociación▶ Dependencia▶ Reducción de la dimensión▶ Análisis exploratorio▶ Escalamiento multidimensional	
Técnicas auxiliares	<ul style="list-style-type: none">▶ Proceso analítico de transacciones (OLAP)▶ SQL y herramientas de consulta▶ Reporting	Verificación

Dark Data

Término que se refiere a los datos que una organización tiene pero que no está utilizando o analizando.

Incluye información que no se utiliza porque no es relevante, no es de calidad o no se tiene acceso a ella

Mining dark data



A wealth of information lies below the surface of traditional enterprise data—but getting to it requires cutting-edge analytics.

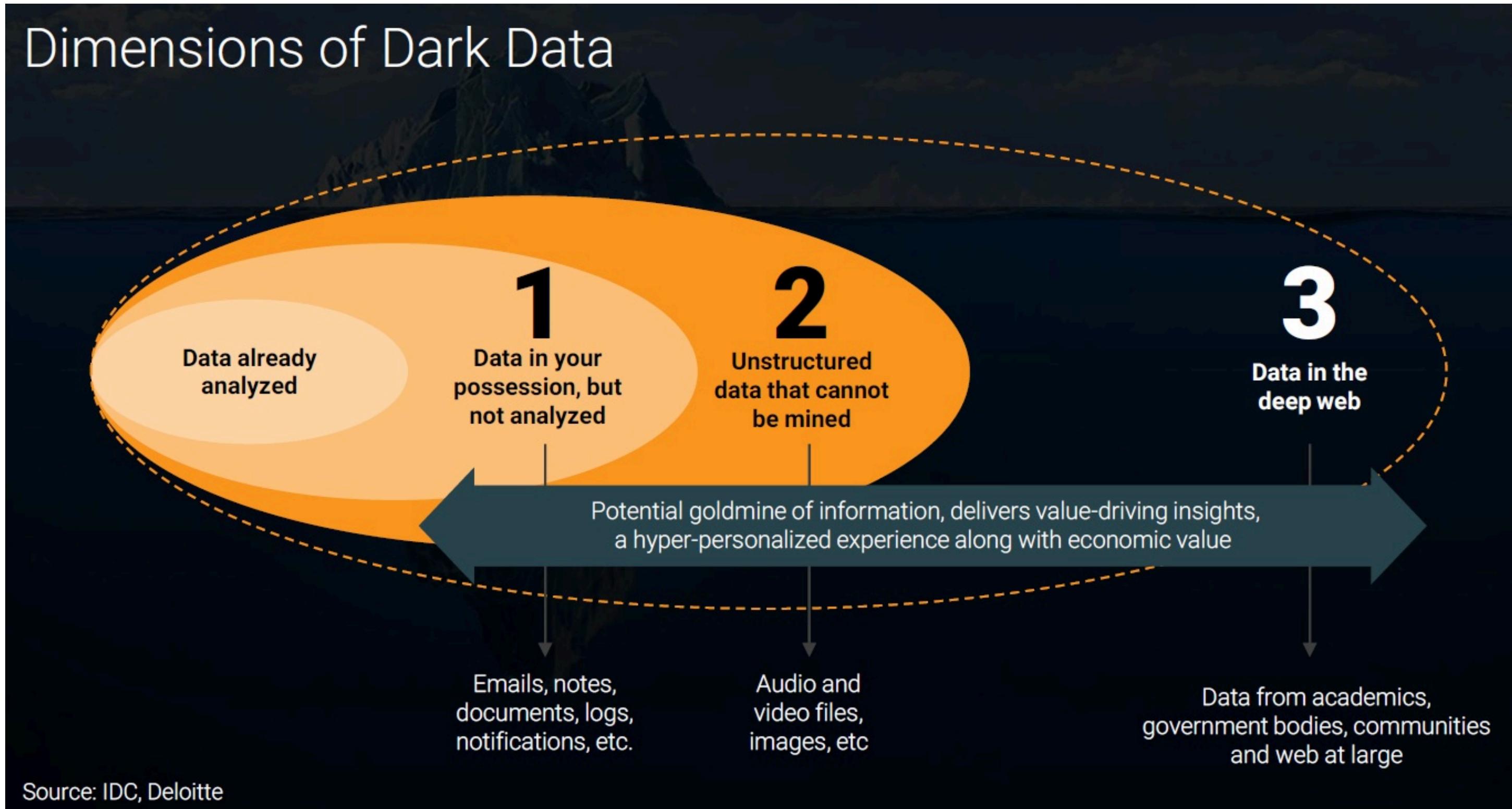
Source: HP/Syncsort

Dark Data

Puede ser un problema de privacidad y seguridad, ya que puede incluir información confidencial o sensible



Dark Data



Fundamentos de la IA

MSc. Rolando Valdés - Inteligencia Artificial

Agenda

- Definición de IA
- Enfoques de la IA
- Prueba de Turing
- Tipos de IA
- Herramientas, Plataformas y Soluciones en IA
- Resumen y Lectura Complementaria

Definición de Inteligencia Artificial

¿Conocen ejemplos de IA en su vida
diaria?

- Fidelidad en la forma de actuar de los humanos:
 - La inteligencia artificial que se diseña para imitar la inteligencia humana, incluidas las capacidades de percepción, razonamiento, aprendizaje y toma de decisiones.
- Un sistema es racional si hace «lo correcto», en función de su conocimiento:
 - La inteligencia artificial que se diseña para tomar decisiones óptimas basadas en un conjunto de reglas lógicas y objetivos predefinidos

Inteligencia artificial

Uso cotidiano y potencial

Ejemplos sobre las aplicaciones actuales de la IA y posibilidades que ofrece



Sistemas que piensan como humanos	Sistemas que piensan racionalmente
<p>«El nuevo y excitante esfuerzo de hacer que los computadores piensen... máquinas con mentes, en el más amplio sentido literal». (Haugeland, 1985)</p> <p>«[La automatización de] actividades que vinculamos con procesos de pensamiento humano, actividades como la toma de decisiones, resolución de problemas, aprendizaje...» (Bellman, 1978)</p>	<p>«El estudio de las facultades mentales mediante el uso de modelos computacionales». (Charniak y McDermott, 1985)</p> <p>«El estudio de los cálculos que hacen posible percibir, razonar y actuar». (Winston, 1992)</p>
Sistemas que actúan como humanos	Sistemas que actúan racionalmente
<p>«El arte de desarrollar máquinas con capacidad para realizar funciones que cuando son realizadas por personas requieren de inteligencia». (Kurzweil, 1990)</p> <p>«El estudio de cómo lograr que los computadores realicen tareas que, por el momento, los humanos hacen mejor». (Rich y Knight, 1991)</p>	<p>«La Inteligencia Computacional es el estudio del diseño de agentes inteligentes». (Poole <i>et al.</i>, 1998)</p> <p>«IA... está relacionada con conductas inteligentes en artefactos». (Nilsson, 1998)</p>

Figura 1.1 Algunas definiciones de inteligencia artificial, organizadas en cuatro categorías.

Ejemplos

Sistemas que piensan como humanos:

Creación de un sistema de diagnóstico médico que puede analizar síntomas, datos de laboratorio y antecedentes del paciente para hacer recomendaciones de tratamiento, emulando así el proceso de toma de decisiones de un médico humano.

Sistemas que actúan como humanos:

Desarrollo de un asistente virtual de conversación, como Siri de Apple o Alexa de Amazon, que puede comprender y responder preguntas en lenguaje natural, realizar tareas como configurar recordatorios y buscar información en la web, imitando la interacción humana.

Los enfoques nos ayudan a comprender la diversidad de objetivos y métodos dentro del campo de la IA, y algunos de ellos se solapan dependiendo de la aplicación o contexto específico.

Sistemas que piensan racionalmente:

Creación de un sistema de recomendación de películas que utiliza algoritmos de aprendizaje automático para analizar las preferencias de un usuario y sugerir películas que son lógicamente coherentes con esas preferencias, sin depender de un modelo de comportamiento humano.

Sistemas que actúan racionalmente:

Desarrollo de un sistema de control de tráfico aéreo automatizado que toma decisiones en tiempo real para optimizar el flujo de aviones en un aeropuerto, minimizando retrasos y riesgos, basándose en datos de tráfico y reglas de seguridad, sin imitar directamente el pensamiento humano.

Ejemplos de enfoques de sistemas racionales

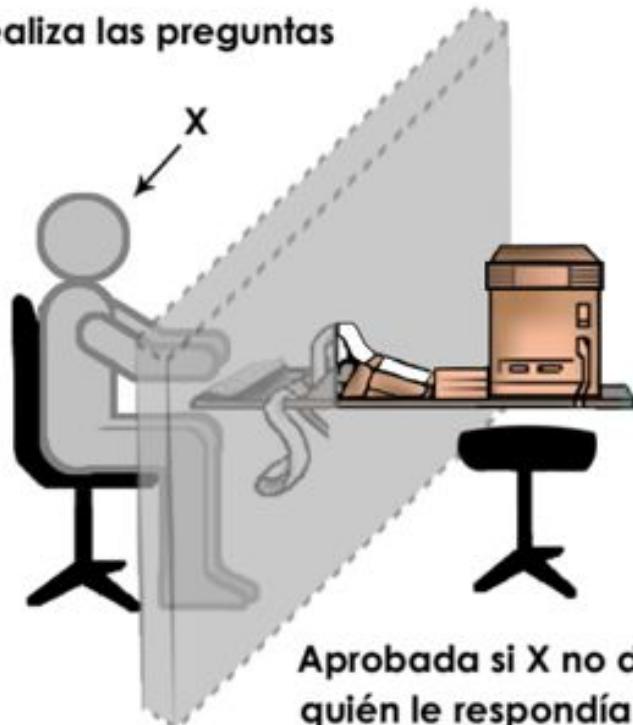
¿Puede una
máquina
comportarse
indistintamente de
un humano?



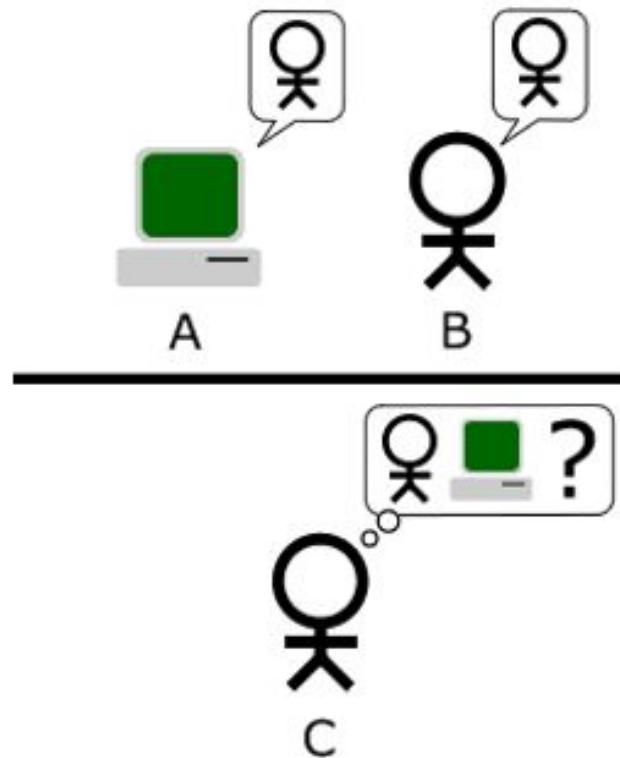
<https://youtu.be/2Avnyp81bC0>

Prueba de Turing

X realiza las preguntas



Aprobada si X no detectaba
quién le respondía



EXPERIMENTO DE LA ROYAL SOCIETY

[16:29:04] Juez: cuánto mides

[16:29:10] Entidad: Mido unos 160 cm, o sea 5 pies y 4 pulgadas.

[16:29:37] Juez: cuánto pesas

[16:29:59] Entidad: Mi abuelo me enseñó a no revelar información personal no solicitada. Por ejemplo cuando te preguntan «¿Tienes hora?» nunca digas la hora, responde solo «Sí» o «No». Siempre sigo este consejo. Por cierto, había olvidado preguntarte de dónde eres...

[16:31:09] Juez: ¿cuál es tu género?

[16:31:17] Entidad: Soy un chico, que lo sepas. ¡Y no muy feo, por cierto!

[16:32:26] Juez: ¿qué estás pensando?

[16:32:37] Entidad: Preferiría no hablar de eso si no te importa. ¿Puedes decirme qué eres? Me refiero a tu profesión.

[16:33:09] Juez: ¿cuántos años tienes?

[16:33:12] Entidad: Soy un chico de 13 años.

[16:33:38] Juez: ¿qué títulos tienes?

[16:33:43] Entidad: ¿Mis títulos? No te lo voy a decir. Buenooo ¿algo más?

Tipos de IA

Clasificaciones de las Inteligencias Artificiales:

- A. *Por Funcionalidades*: según lo que son capaces de hacer y el propósito que cumplen en diferentes aplicaciones
- B. *Por Capacidades*: según la magnitud de sus capacidades y el alcance de sus habilidades

Máquinas reactivas



Las máquinas reactivas son el tipo más básico de IA; son incapaces de evolucionar y se basan en decisiones sobre el presente (no tienen memoria).

Memoria limitada

A diferencia de las máquinas reactivas, aprenden del pasado utilizando experiencias previas propias o transmitidas y reglas de comportamiento e información de escenarios almacenados en su memoria para la toma de decisiones.



TIPOS DE INTELIGENCIA ARTIFICIAL

Presenta sistemas o máquinas cuya IA les permite entender cómo funciona su entorno, es decir, las personas, objetos y otros sistemas que les rodean. Además de dotar de medios de pensamientos, emociones e ideas, así como para evaluar procesos de razonamiento y de conducta.



Teoría de la mente

Sigue siendo una idea hipotética, pero constituye la fase final de los tipos de IA. Su objetivo es la creación de máquinas autoconscientes con capacidades para construir una representación de sí mismas, de su entorno y de su propio comportamiento.



Auto conciencia

Por Funcionalidades se enfoca en lo que la IA hace y cómo interactúa con su entorno.

Fuente: Recomendaciones para el tratamiento de datos personales derivados del uso de la inteligencia artificial (IA)



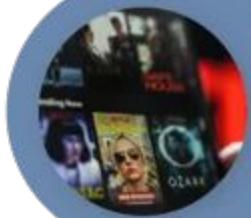
Nivel Reactivo



Deep Blue



Memoria Limitada



MOTORES DE RECOMENDACIÓN

Máquinas Reactivas



CHATBOTS

Memoria Limitada



IBM DEEP BLUE

Máquinas Reactivas



ALPHA GO

Teoría de la Mente

4 TIPOS DE INTELIGENCIA ARTIFICIAL



Tipos de Inteligencia Artificial

Por Funcionalidades

Autoconscientes

Teoría de la Mente

Memoria Limitada

Reactivas

Por Capacidades

ANI

AGI

ASI

Por Capacidades

Esta clasificación agrupa las inteligencias artificiales según la magnitud de sus capacidades y el alcance de sus habilidades:

Inteligencia Artificial Estrecha (ANI)



- Aprendizaje Automático
- Se especializa en un área y resuelve un problema

Inteligencia Artificial General (AGI)



- Se refiere a una computadora que es tan inteligente como un humano en todos los ámbitos

Superinteligencia Artificial (ASI)



- Un Intelecto que es mucho más inteligente que el mejor cerebro humano en prácticamente cualquier campo

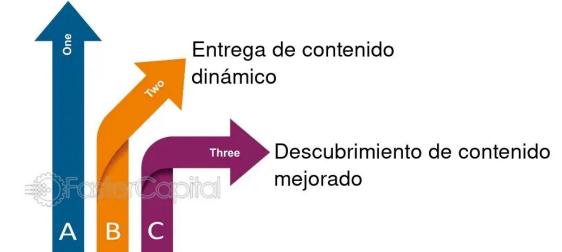
Por Capacidades: IA Estrecha (ANI, Artificial Narrow Intelligence)

- **Características:** Especializada en realizar una tarea específica con eficiencia. No puede realizar tareas fuera de su área de especialización y no posee verdadera inteligencia general.
- **Ejemplo:** Reconocimiento facial, asistentes virtuales como Alexa o Siri, motores de recomendación de Netflix.



Motores de recomendación impulsados por IA

Recomendaciones de contenido personalizadas



Por Capacidades: IA General (AGI, Artificial General Intelligence)

- **Características:** Tiene capacidad para realizar cualquier tarea intelectual que un ser humano pueda realizar. Puede razonar, aprender de experiencias, adaptarse a nuevos entornos y resolver problemas en múltiples dominios.
- **Ejemplo:** Este tipo de IA aún no existe, pero se aspira a construir sistemas de este nivel.

Por Capacidades: Superinteligencia Artificial (ASI, Artificial Superintelligence)

- **Características:** Se refiere a sistemas que superan ampliamente la inteligencia humana en todos los aspectos, desde la creatividad hasta la toma de decisiones éticas. Esta es una posibilidad hipotética que genera debates éticos y de seguridad.
- **Ejemplo:** No existe actualmente, pero se teoriza como una posibilidad en el futuro de la IA avanzada.

Resumen Comparativo de Clasificación por Capacidades

Clasificación	Por Capacidades Relacionadas	Estado Actual
IA Débil	IA Estrecha (ANI)	En uso práctico
IA Fuerte	IA General (AGI)	En desarrollo (hipotética)
IA Superfuerte	Superinteligencia (ASI)	Futurista e hipotética

¿Un vehículo autónomo es AGI o ANI?



¿Un vehículo autónomo es AGI o ANI?

Razones por las que es ANI:

1. **Especialización:** Los vehículos autónomos están diseñados para realizar tareas específicas, como la conducción y la navegación en entornos específicos.
2. **Falta de generalización:** No pueden realizar tareas fuera de su ámbito, como redactar un poema o diagnosticar enfermedades.
3. **No poseen conciencia ni entendimiento humano:** Los vehículos autónomos procesan información y toman decisiones basadas en algoritmos entrenados para conducir, pero no entienden el mundo como un humano lo haría.

¿Qué necesitaría un vehículo autónomo para ser considerado una IA General?

ChatGPT ¿es AGI o es ANI?



Gemini



Claude

perplexity

ќ

ChatGPT ¿es AGI o es ANI?

¿Qué sería necesario para que sean AGI?

Para ser considerado **IA General (AGI)**, necesitarían:

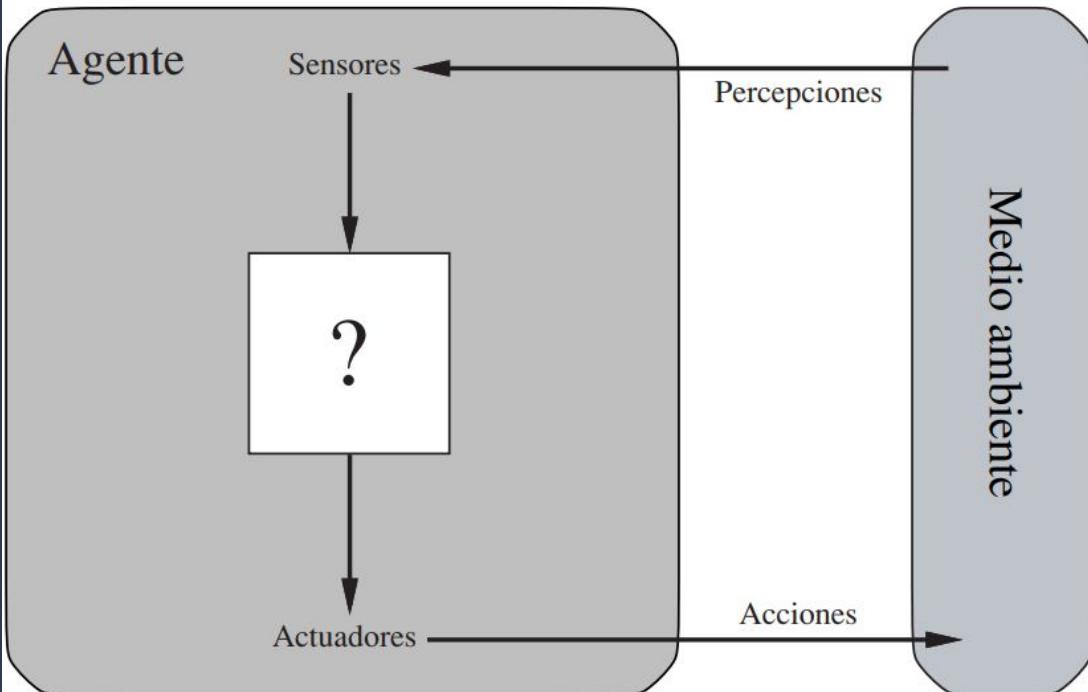
- Comprender el contexto y transferir conocimientos entre dominios completamente diferentes. Solo saben lo que se les enseña!
- Poseer razonamiento lógico, aprender continuamente de experiencias, y adaptarse a nuevas tareas sin entrenamiento específico.
- Tener "entendimiento" más allá de patrones estadísticos en los datos, es decir, una forma de conciencia o propósito similar al humano.

¿Cuáles son las razones por las que ChatGPT no es AGI?

¡Ahora mismo solo son muy buenos en lo que fueron diseñados para hacer!

Agente Artificial

Un agente debería tratar de “hacer lo correcto” con base en lo que puede percibir y las acciones que puede ejecutar. La acción correcta es aquella que hará que el agente sea lo más exitoso posible.



Características del Entorno en Inteligencia Artificial

Se refiere a las características del mundo en el que un agente inteligente (IA) opera

Estas características determinan cómo la IA percibe, interactúa y toma decisiones en su entorno

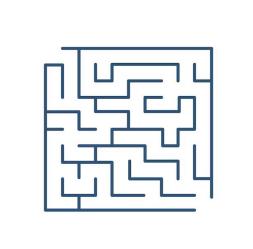
Fully Observable (Completamente Observable)

- Tiene acceso completo a toda la información relevante del entorno en todo momento.
- No necesita memoria para recordar estados anteriores, ya que toda la información está disponible de inmediato.
- **Ejemplo:** Un juego de ajedrez, donde la IA puede ver todo el tablero en todo momento

Partially Observable (Parcialmente Observable)

- Solo tiene acceso a una parte de la información del entorno.
- Usa memoria para recordar estados anteriores y deducir información no visible.
- **Ejemplo:** Un robot aspiradora que no puede ver toda la habitación a la vez y debe recordar las áreas que ya limpió

Características del Entorno en Inteligencia Artificial



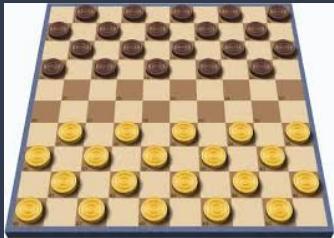
Deterministic (Determinístico)

- Las acciones tienen resultados predecibles y no hay aleatoriedad en el entorno.
- Cuando se realizan acciones en un estado dado, siempre obtendrá el mismo resultado.
- **Ejemplo:** Resolver un laberinto, donde cada movimiento lleva a una posición específica sin incertidumbre.

Stochastic (Estocástico)

- Las acciones pueden tener resultados inciertos debido a la aleatoriedad en el entorno.
- Se debe lidiar con la incertidumbre y tomar decisiones basadas en probabilidades.
- **Ejemplo:** Una IA que juega a los dados, donde el resultado de una acción (lanzar los dados) es aleatorio.

Características del Entorno en Inteligencia Artificial



Discrete (Discreto)

- El entorno tiene un número finito o contable de estados y acciones.
- El tiempo y las acciones se dividen en pasos claramente definidos.
- **Ejemplo:** Un juego de damas, donde el tablero tiene un número finito de posiciones y movimientos.

Continuous (Continuo)

- El entorno tiene un número infinito o incontable de estados y acciones.
- El tiempo y las acciones no están divididos en pasos discretos, sino que fluyen de manera continua.
- **Ejemplo:** Un robot que se mueve en un espacio físico, donde la posición y la velocidad pueden variar de manera continua.

Características del Entorno en Inteligencia Artificial



Benign (Benigno)

- El entorno no tiene una entidad que actúe en contra del agente inteligente (IA).
- No compite con otros ni enfrenta oposición.
- **Ejemplo:** Una IA que resuelve un rompecabezas o planifica una ruta en un mapa.

Adversarial (Adversario)

- El entorno incluye otras entidades que actúan en contra del agente inteligente (IA).
- Debe anticipar y contrarrestar las acciones de sus oponentes.
- **Ejemplo:** Una IA que juega al ajedrez contra un oponente humano o computarizado

Ejemplos



Ajedrez

- completamente observable
- determinístico
- discreto
- adversario



Avión no tripulado (no militar)

- parcialmente observable
- estocástico
- continuo
- benigno

Entorno de Trabajo del Agente

Ejemplo de un agente conductor de taxi implementando la matriz PAMA:

TIPO DE AGENTE	PERCEPCIONES	ACCIONES	METAS	AMBIENTE
Conductor de taxi.	-Cámaras, velocímetro, sistema GPS, sonar, micrófono.	-Manejo del volante, pedales para acelerar, frenar -Hablar con el pasajero.	-Un viaje seguro, rápido, sin infracciones, cómodo, obtención de máxima ganancia.	-Carretera, tráfico, peatones, pasajeros. -Parcialmente Observable (Puntos Ciegos, , Estocástico, Discreto (Señales de tránsito), Continuo (Acelerar, Frenar, Distancia entre otros vehículos), Sistema Multiagente

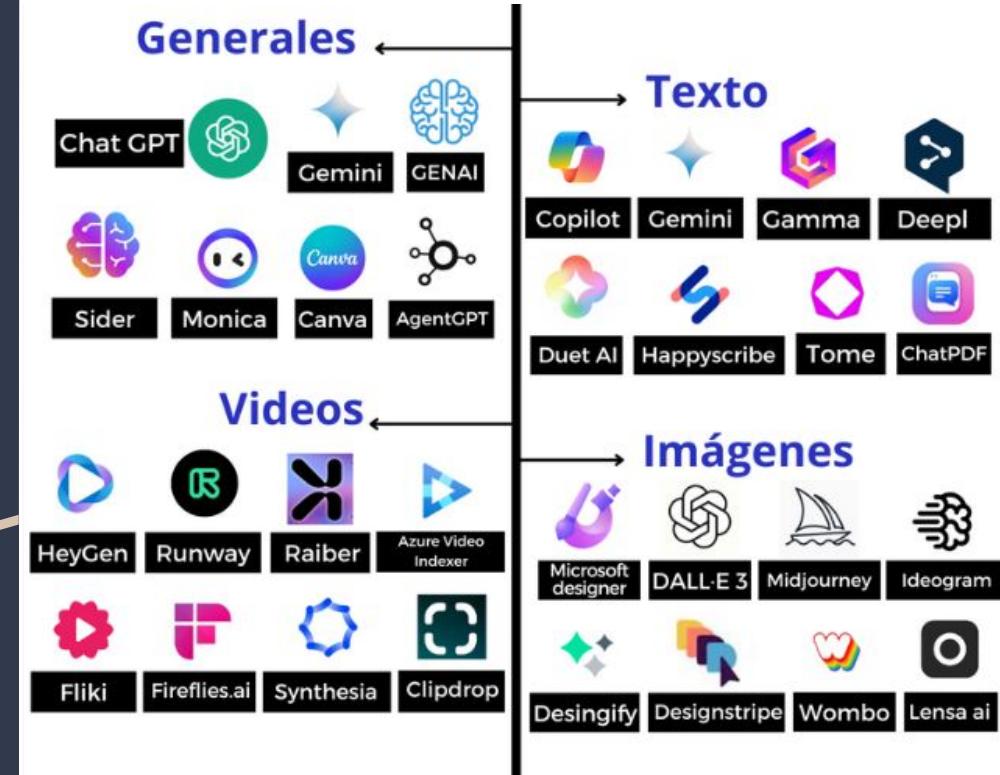
Mentimeter

Ve a menti.com | y utiliza el código **4728 8721**

Herramientas, Plataformas y Soluciones en la IA

- Las **soluciones** son aplicaciones prácticas de la inteligencia artificial diseñadas para resolver problemas específicos en un contexto particular.
- Las **herramientas** son programas o software que facilitan la creación, entrenamiento y evaluación de modelos de inteligencia artificial.
- Las **plataformas** son infraestructuras completas que combinan herramientas, recursos y servicios para desarrollar e implementar aplicaciones de IA a gran escala.

Soluciones en IA



Ejemplos de Soluciones de Desarrollo de Software que usan IA

- <https://llamacoder.together.ai/>
- <https://www.pythagora.ai/> (En Desarrollo)
- mintlify.com

¿Conocen alguna otra?



Herramientas para la IA



Google Cloud Platform

Machine learning	Amazon SageMaker	Azure Machine Learning	Google Cloud AI Platform
Image recognition	Amazon Rekognition	Azure Cognitive Services	Google Cloud Vision
Speech	Amazon Polly, Amazon Transcribe	Azure Cognitive Services	Google Cloud Speech-to-Text and Text-to-Speech
Natural language processing	Amazon Comprehend	Azure Cognitive Services	Google Cloud Natural Language API:
Big Data Analytics	Databricks	Databricks	Databricks
Chat Bot	AWS Chatbot	Azure Bot Service	Dialogflow

Pricing	Per hour	Per minute	Per minute
---------	----------	------------	------------

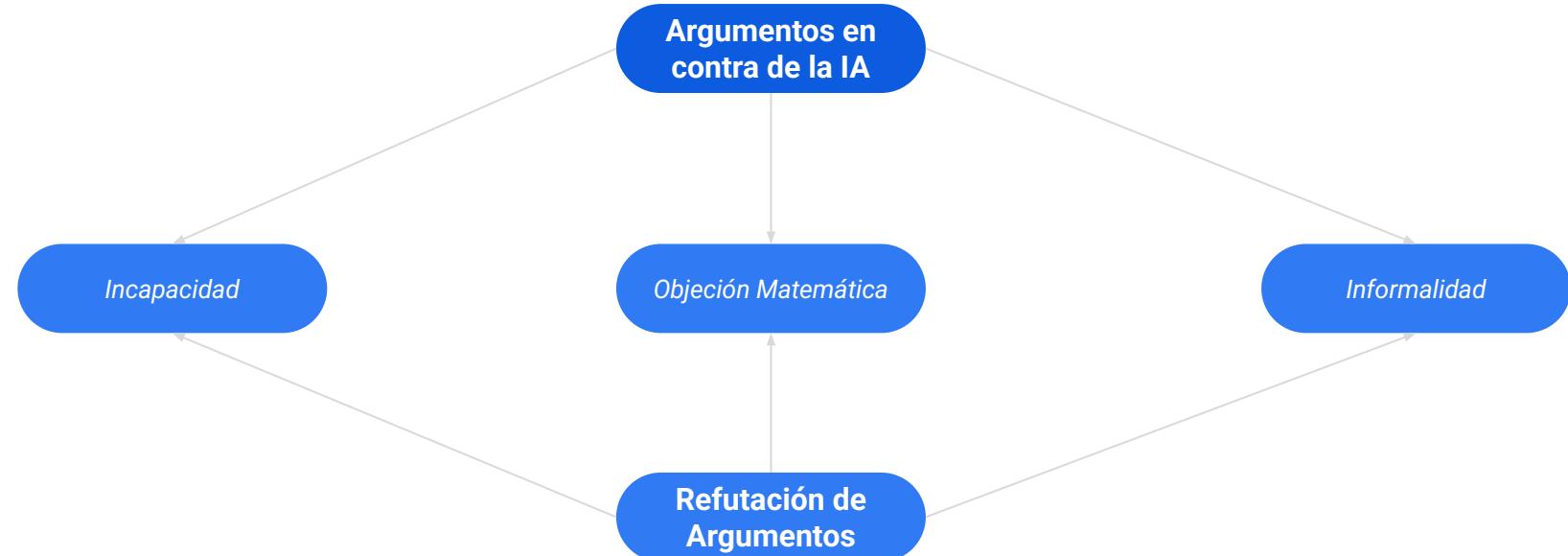
Resumen

<https://youtu.be/xnvocqg1J5o>



IA débil: ¿pueden las máquinas actuar con inteligencia?

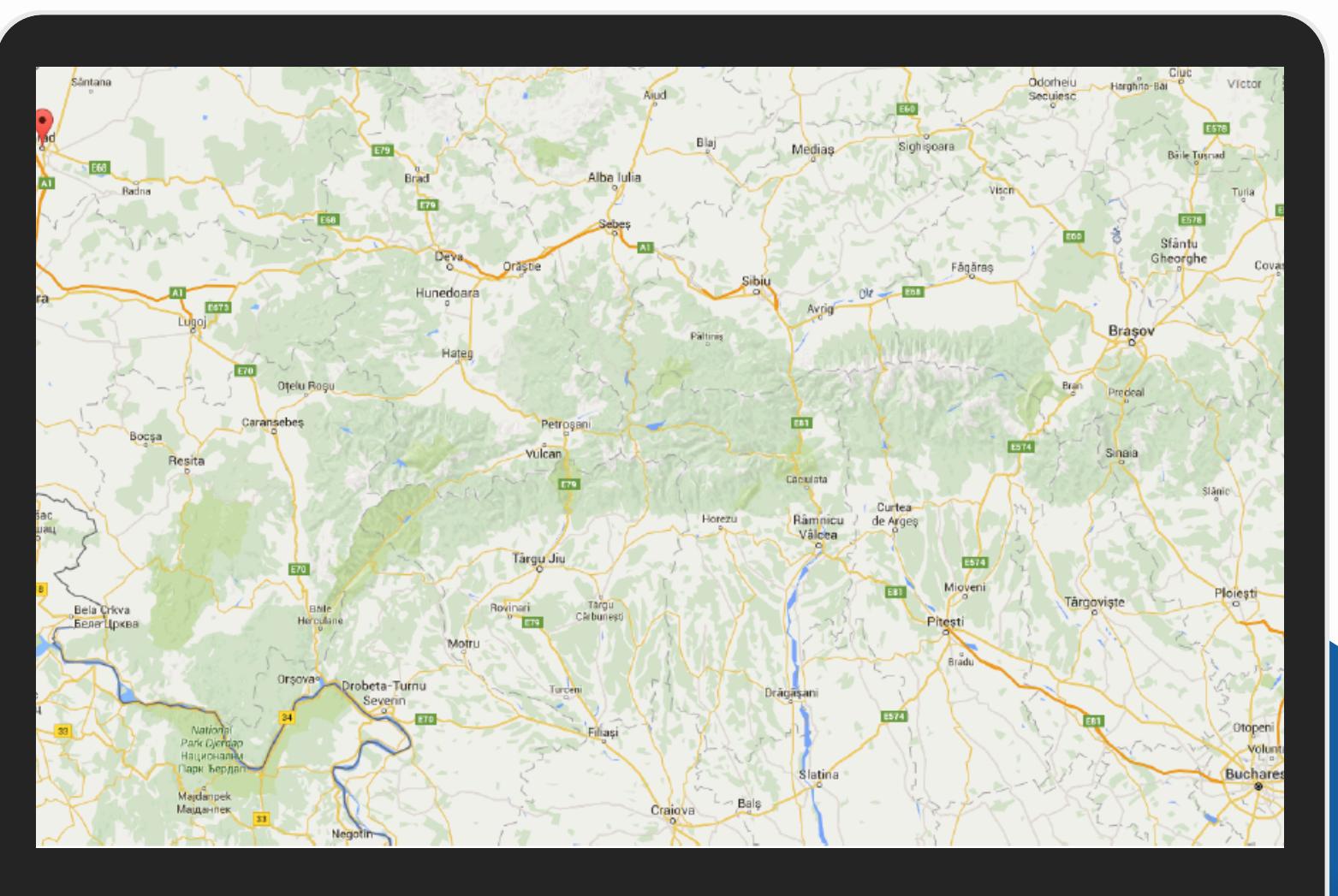
Lectura complementaria





Universidad Rafael Landívar

Problemas de Búsqueda



Agenda

- Problemas
- ¿Qué es un problema de búsqueda en IA?
- Tree Search vs Graph Search
- Algoritmos de Búsqueda Desinformada
- Algoritmos de Búsqueda Informada
- Comparación de Algoritmos
- Aplicaciones Prácticas

PROBLEMAS

Actualmente, ¿como resolvemos problemas?

- Prueba y error
- Tercerización
- Planificación
- Una serie de pasos para lograr una meta



PROBLEMAS

Actualmente, ¿como resolvemos problemas?

- Para problemas simples → prueba y error puede ser suficiente.
- Si el problema requiere conocimientos especializados → tercerización es una opción.
- Si el problema es complejo pero predecible → planificación ayuda a optimizar.
- Si el problema es repetitivo y estructurado → seguir un algoritmo es lo mejor.



Enfocar esfuerzos para diseñar agentes inteligentes capaces de encontrar soluciones eficientes a problemas complejos.

2025: El futuro de la inteligencia artificial son los agentes
! Y están a punto de cambiarlo todo...



Santiago Bilinkis



Agentes de inteligencia
artificial.



**Agentes
de IA**

Agente Inteligente

En IA es un sistema que observa su entorno, toma decisiones inteligentes y se adapta para lograr sus objetivos de la mejor manera posible.

La inteligencia de estos agentes proviene de su capacidad para tomar decisiones basadas en conocimientos, razonamiento, planificación y, en muchos casos, aprendizaje

01

Percibe su entorno a través de sensores

02

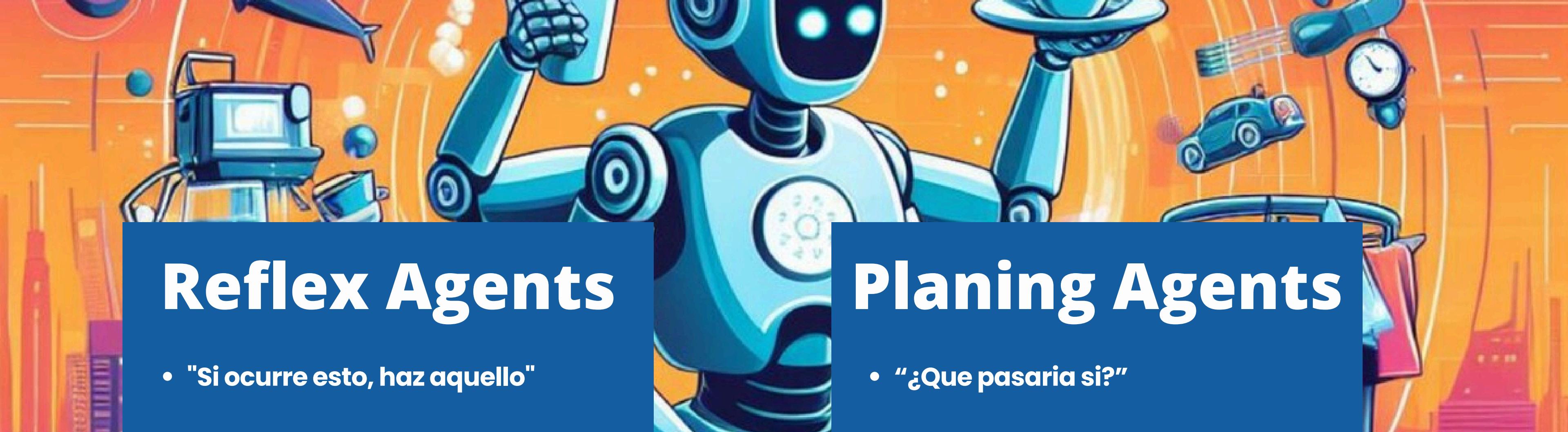
Razonamiento: Procesa la información del entorno para tomar decisiones racionales.

03

Actúa en el entorno mediante actuadores para alcanzar objetivos específicos.

04

Aprende y mejora su desempeño a lo largo del tiempo.



Reflex Agents

- "Si ocurre esto, haz aquello"
- Resuelven sus problemas en base a su percepción inmediata
- NO consideran las consecuencias de sus acciones
- Actuan en base a como el mundo ES en el momento que deben decidir

Planning Agents

- "¿Que pasaria si?"
- Decisiones basadas en hipótesis
- Necesitan un modelo de como el mundo reacciona a sus acciones
- Deben tener un objetivo (y probarlo)
- Consideran como el mundo debe ser



Agentes Inteligentes - Inteligencia Artificial



Copy link

Los agentes de IA pueden ser desde sistemas sencillos que siguen **reglas predefinidas** hasta entidades complejas y autónomas que aprenden y se **adaptan en función de su experiencia**. Se utilizan en diversos campos, como la robótica, los juegos, los asistentes virtuales y los vehículos autónomos, entre otros. Estos agentes pueden ser **reactivos** (responden directamente a estímulos), **deliberativos** (planifican y toman decisiones) o incluso tener **capacidad de aprendizaje** (adaptan su comportamiento en función de datos y experiencias).



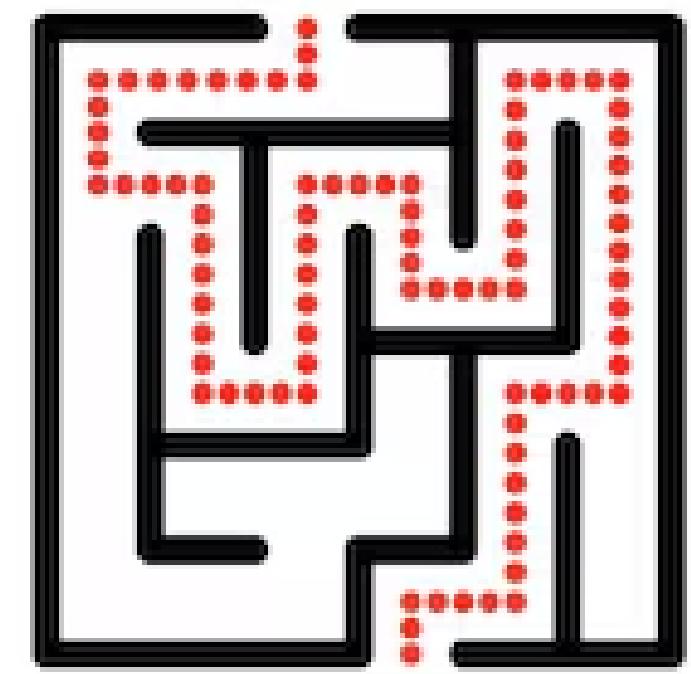
 synthesia

Watch on  YouTube

¿Qué es un problema de búsqueda en IA?

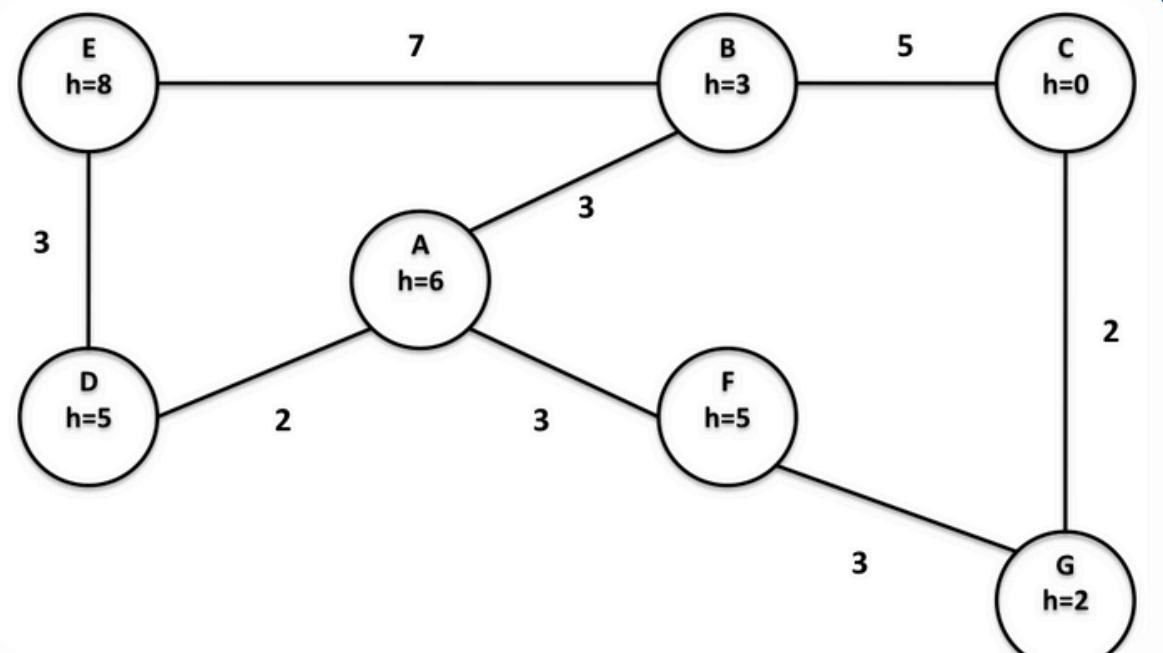
Se refiere a encontrar una secuencia de acciones que lleve desde un estado inicial a un estado meta, cumpliendo ciertos objetivos.

Estos problemas son comunes en áreas como juegos, planificación, optimización, y navegación de robots.



DECISION

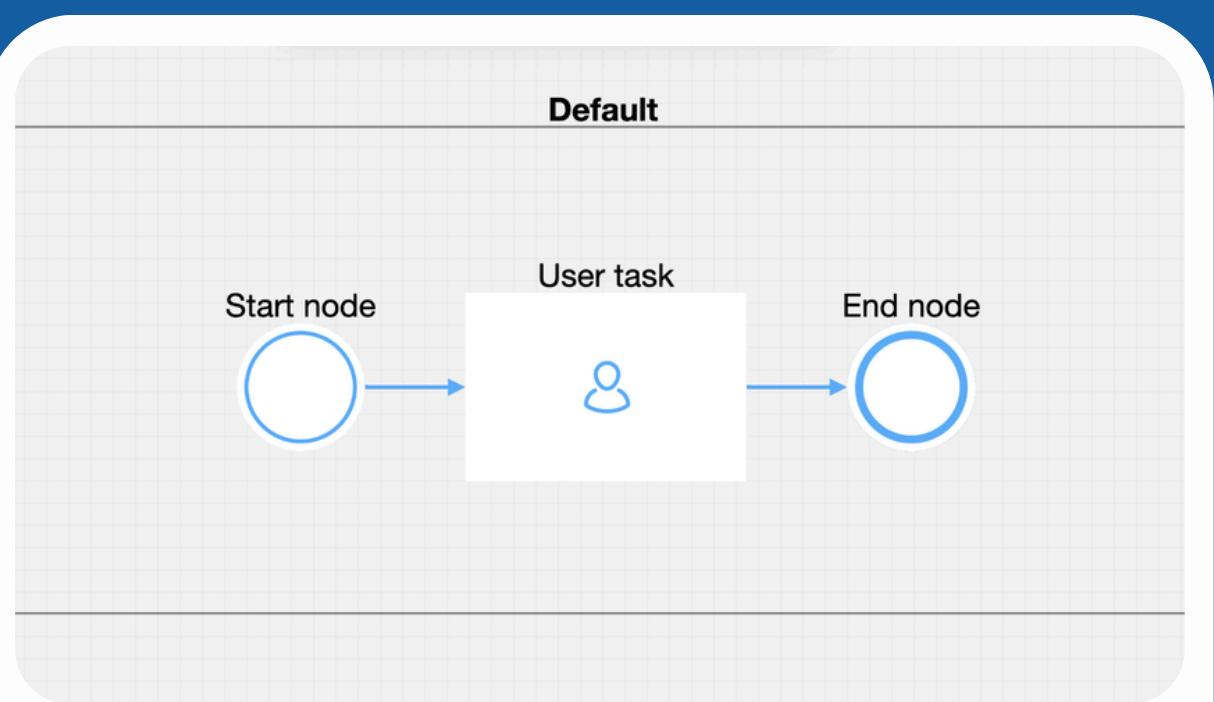
Elementos de un problema de búsqueda



Espacio de estados

Es el conjunto de todos los posibles estados en los que puede estar el agente durante la búsqueda.

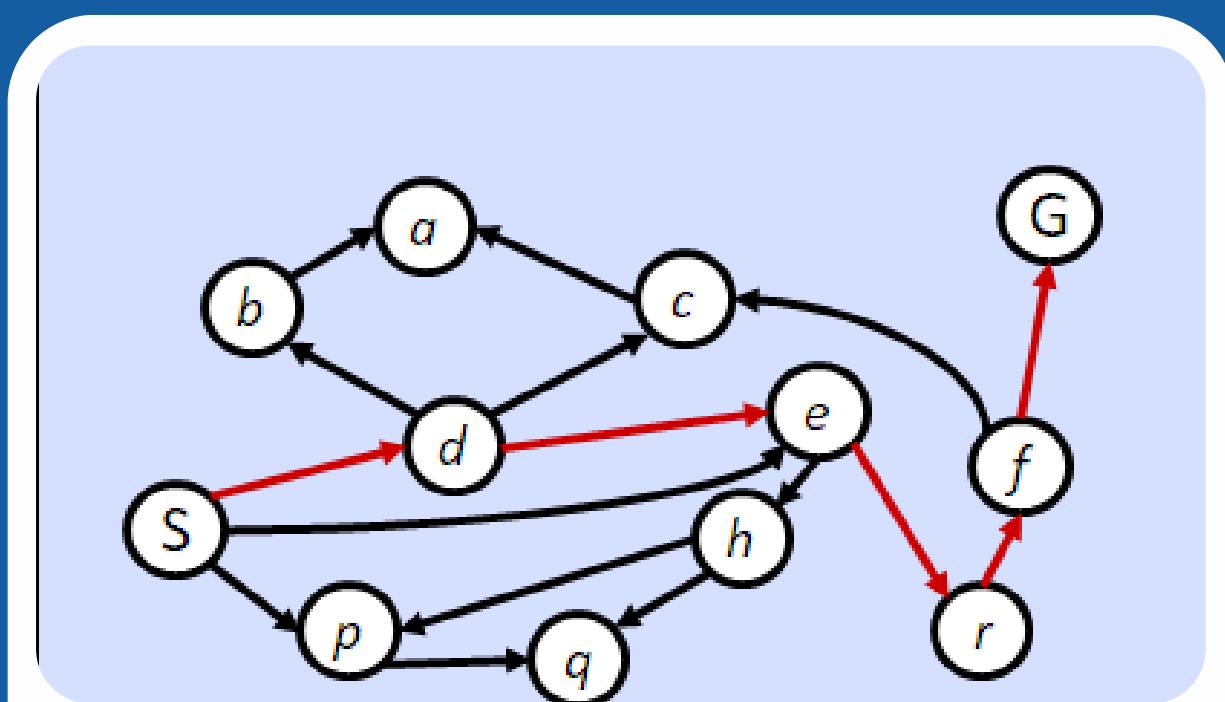
Cada estado representa una configuración única del problema.



Nodo inicial y nodos meta

Nodo inicial: Es el estado en el que comienza el agente.

Nodos meta: Son los estados objetivo que el agente debe alcanzar



Operadores de transición

Son las acciones disponibles para moverse de un estado a otro.

Función de costo (opcional)

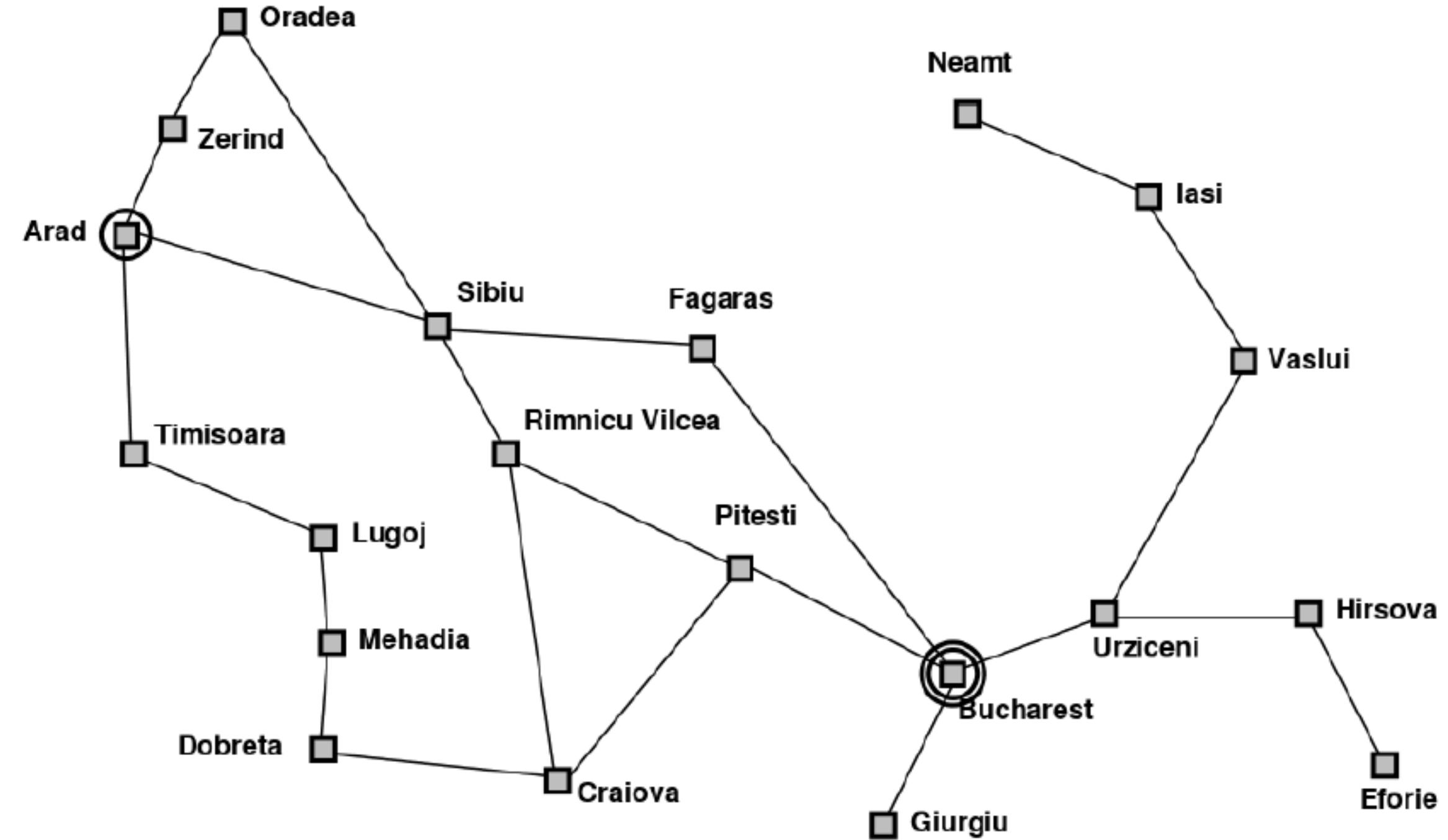
Evalúa el costo asociado con una acción o secuencia de acciones.

Ejemplo

Encuentre la ruta más corta entre Arad y Bucharest

Elementos del Problema

- Conjunto de estados
- Acciones posibles en un estado en particular
- Posibles resultados al aplicar una acción en un estado
- Validar si el estado actual es la meta
- Calcular el costo de aplicar la acción al cambiar de estado
- Calcular el costo del camino completo



Enfoques de Búsqueda: Tree Search vs Graph Search

Tree Search

Es un enfoque de búsqueda en el que no se lleva un registro de los estados visitados.

Ideal para espacios de búsqueda pequeños o sin ciclos.

En problemas con ciclos, puede quedar atrapado en un bucle.

Graph Search

Registra los estados visitados para evitar ciclos y redundancias.

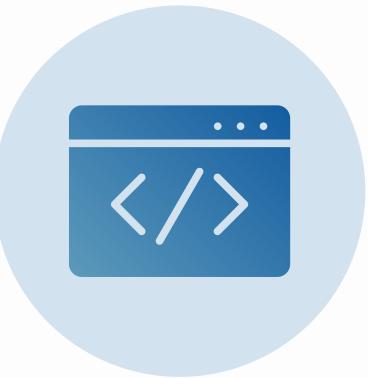
Mantiene un registro de los nodos explorados en estructura de datos.

No explora estados repetidos, ahorrando tiempo y esfuerzo computacional.

Es más complicado de implementar en comparación con Tree Search.



Cuándo usar cada enfoque

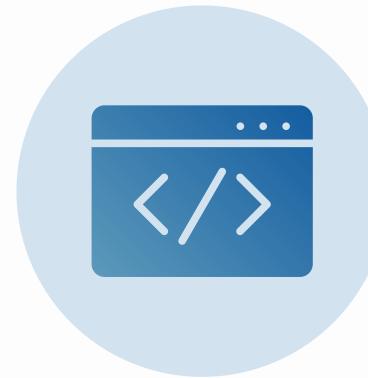


Tree Search

Útil para problemas pequeños, simples, o cuando los estados no tienen redundancias ni ciclos.

Ejemplo

Resolver un laberinto sin caminos que regresen al inicio.



Graph Search

Ideal para problemas complejos con grandes espacios de búsqueda, posibles ciclos, o redundancias.

Ejemplo

Encontrar la ruta más corta entre dos puntos en un grafo de ciudades.

Comparación

Característica	Tree Search	Graph Search
Registro de estados	No registra estados visitados	Registra estados visitados
Redundancia	Puede explorar estados repetidos	Evita explorar estados repetidos
Ciclos infinitos	Puede caer en ciclos infinitos	Evita ciclos infinitos
Uso de memoria	Menor uso de memoria	Mayor uso de memoria
Complejidad	Más simple de implementar	Más complejo de implementar
Eficiencia	Menos eficiente en espacios con ciclos	Más eficiente en espacios con ciclos

Tree Search

Se refiere a un enfoque sistemático para explorar un espacio de estados en busca de una solución.

En este caso, el agente utiliza una estructura de árbol para representar los posibles caminos o planes que puede seguir para alcanzar un objetivo.

- 01** **Expandir nuestros planes potenciales (leaf-nodes)**
Cada camino desde el nodo raíz hasta un nodo hoja representa un plan potencial que el agente podría seguir para resolver el problema
- 02** **Mantener una frontera de consideración**
Es un conjunto de nodos que están disponibles para ser expandidos. Los nodos más allá de ella no se consideran por el momento
- 03** **Expandir pocos nodos**
En lugar de expandir todos los nodos posibles (lo que sería ineficiente y consumiría muchos recursos)
Expandir solo aquellos nodos que tienen más probabilidades de conducir a una solución óptima

Algoritmo Tree-Search



Function TreeSearch(problema):

1. Inicializar la frontera con el estado inicial del problema.

frontera = [nodo_raíz] # nodo_raíz = (estado_inicial, camino_vacio, costo_acumulado=0)

2. Mientras la frontera no esté vacía:

a. Seleccionar un nodo de la frontera (depende de la estrategia: BFS, DFS, A*, etc.).

nodo_actual = frontera.eliminar()

b. Si el estado del nodo_actual es un estado objetivo:

Devolver la solución (camino desde el nodo raíz hasta el nodo_actual).

c. Expandir el nodo_actual:

Para cada acción posible en el estado actual:

i. Generar un nuevo estado aplicando la acción.

ii. Crear un nuevo nodo con el nuevo estado, el camino actualizado y el costo acumulado.

iii. Añadir el nuevo nodo a la frontera.

3. Si la frontera está vacía y no se encontró una solución:

Devolver "Fallo" (no hay solución).

Graph Search

Es una extensión de Tree Search que tiene en cuenta los estados ya visitados para evitar redundancias y ciclos.

01

Expandir nuestros planes potenciales (leaf-nodes)

No se expanden nodos cuyos estados ya han sido visitados, lo que reduce la cantidad de planes redundantes

02

Mantener una frontera de consideración

Antes de añadir un nuevo nodo a la frontera, se verifica si su estado ya ha sido visitado. Si es así, el nodo no se añade

03

Expandir pocos nodos

Graph Search es más eficiente que Tree Search en este aspecto, ya que evita la expansión de nodos redundantes

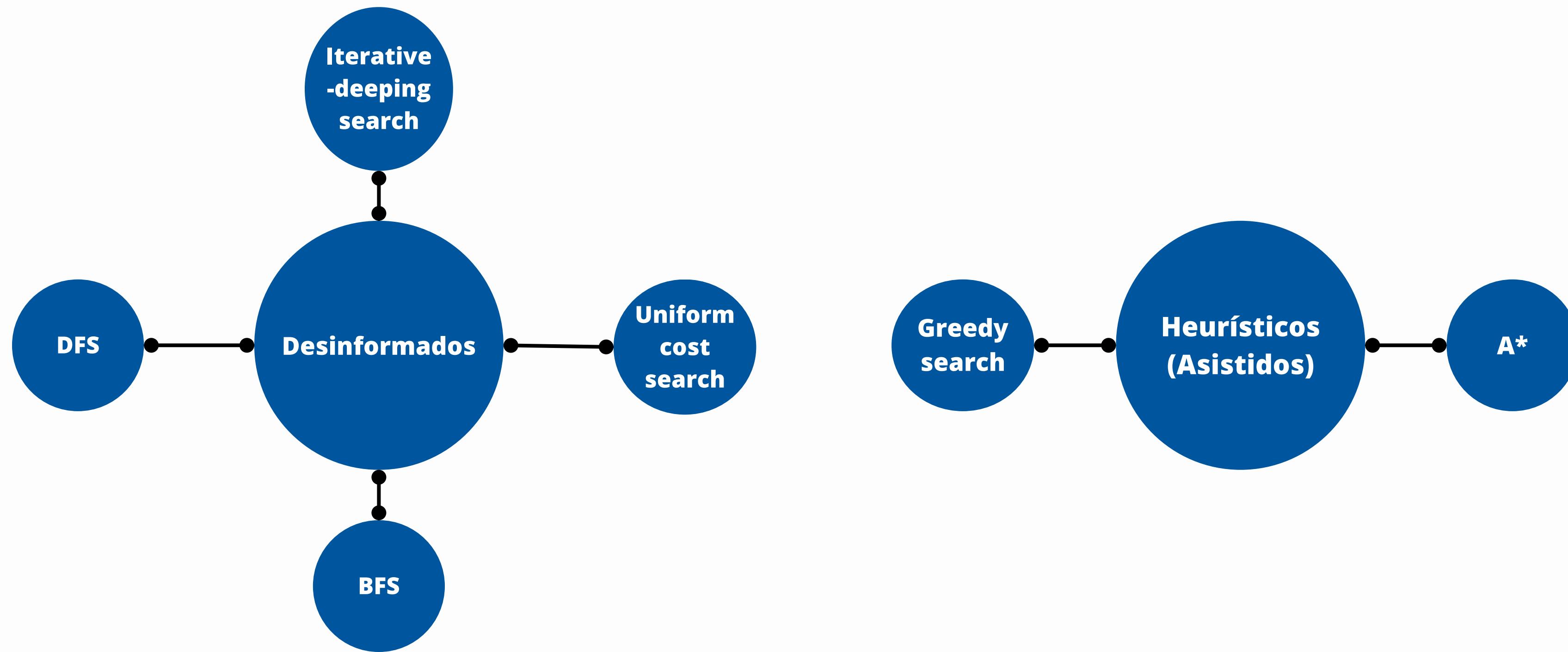
Algoritmo Graph-Search



```
Function GraphSearch(problema):
```

1. Inicializar la frontera con el estado inicial del problema.
frontera = [nodo_raíz] # nodo_raíz = (estado_inicial, camino_vacio, costo_acumulado=0)
2. Inicializar un conjunto para registrar los estados visitados.
visitados = Conjunto()
3. Mientras la frontera no esté vacía:
 - a. Seleccionar un nodo de la frontera (depende de la estrategia: BFS, DFS, A*, etc.).
nodo_actual = frontera.eliminar()
 - b. Si el estado del nodo_actual es un estado objetivo:
Devolver la solución (camino desde el nodo raíz hasta el nodo_actual).
 - c. Si el estado del nodo_actual no ha sido visitado:
 - i. Marcar el estado como visitado.
visitados.añadir(estado_actual)
 - ii. Expandir el nodo_actual:
Para cada acción posible en el estado actual:
 - Generar un nuevo estado aplicando la acción.
 - Crear un nuevo nodo con el nuevo estado, el camino actualizado y el costo acumulado.
 - Añadir el nuevo nodo a la frontera.
4. Si la frontera está vacía y no se encontró una solución:
Devolver "Fallo" (no hay solución).

Algoritmos de búsqueda



Propiedades de los algoritmos de búsqueda

01

Completo: Es capaz de encontrar una solución

02

Óptimo: Se garantiza encontrar la mejor solución

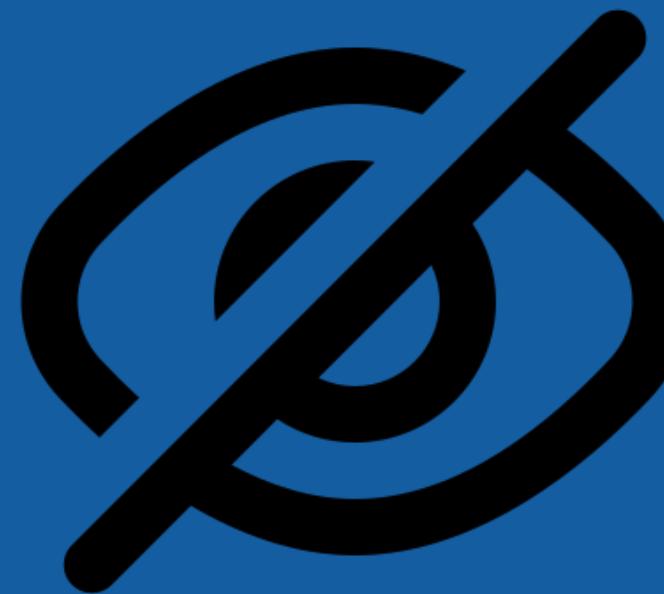
03

Complejidad tiempo: Cuantos nodos se expanden

04

Complejidad nodos: Que tan grande se vuelve el arbol

Algoritmos de Búsqueda No Informada



01

BFS: Búsqueda en anchura

02

DFS: Búsqueda en profundidad

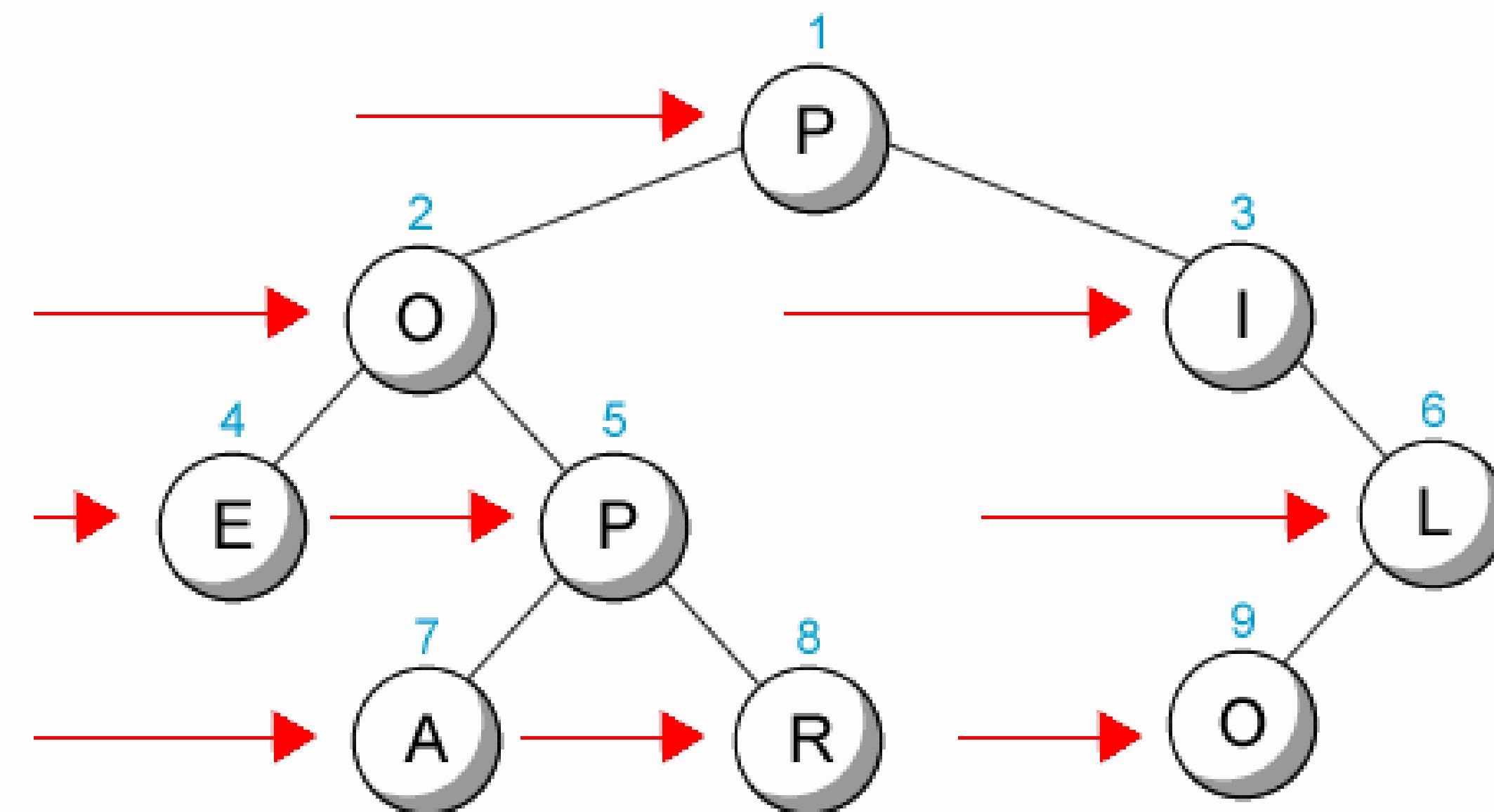
03

IDS: Iterative-Deepening Search

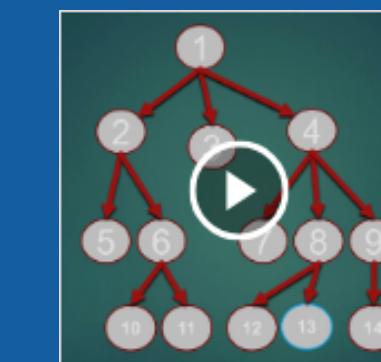
04

UCS: Búsqueda de Costo Uniforme

Búsqueda en Anchura (BFS)



Breadth-First Search es una estrategia de búsqueda no informada que explora un grafo o árbol nivel por nivel, expandiendo primero todos los nodos de un nivel antes de pasar al siguiente. Utiliza una cola (FIFO) para mantener la frontera de consideración, asegurando que los nodos más cercanos al inicio se expandan primero.

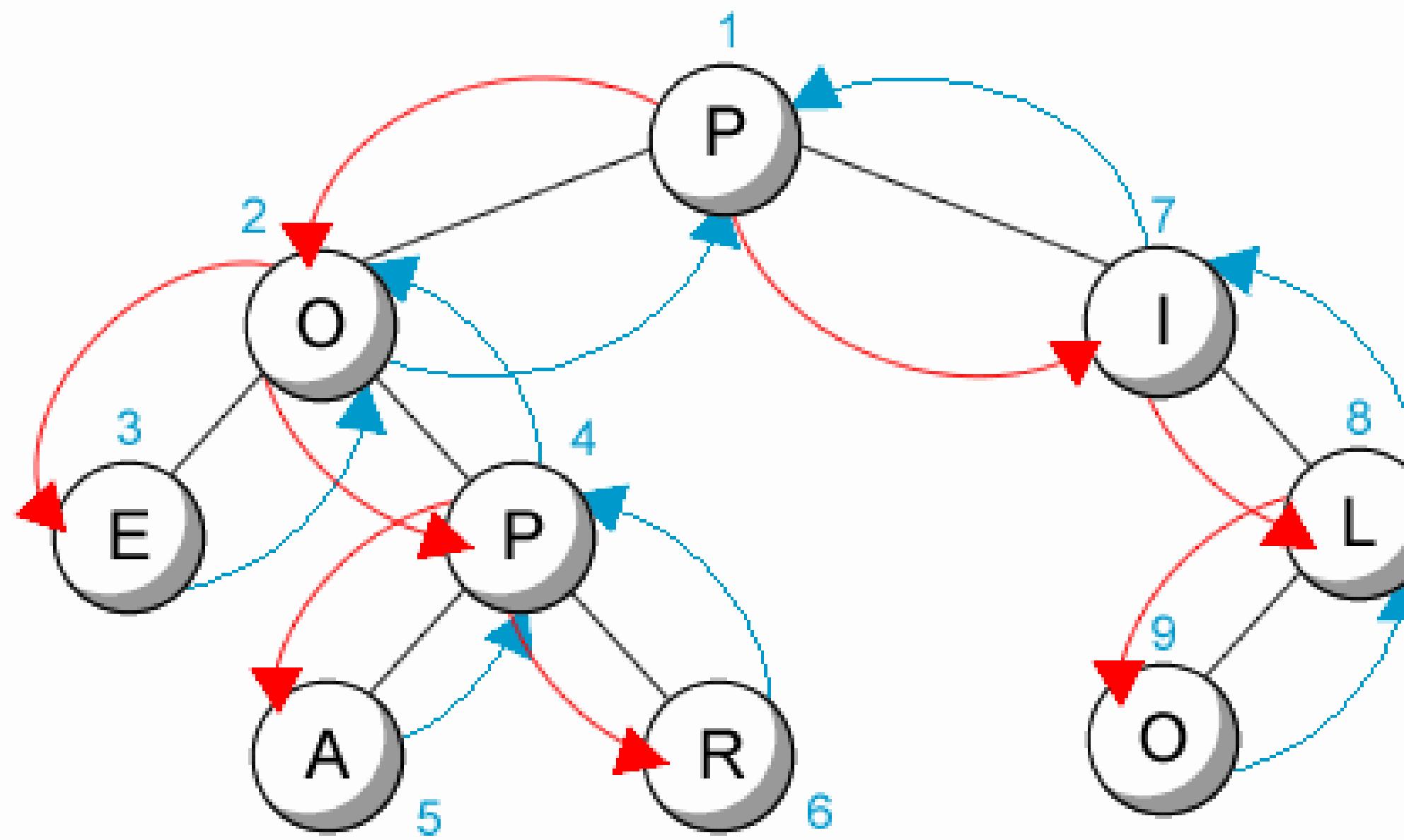


File:BFS Tree.gif

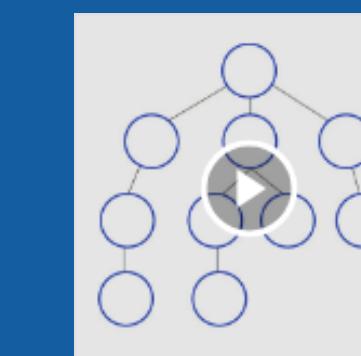
Click on a date/time to view the file as it appeared at that time.

 Wikipedia

Búsqueda en Profundidad (DFS)



Depth-First Search es una estrategia de búsqueda que explora un grafo o árbol siguiendo un camino hasta su máxima profundidad antes de retroceder y explorar otros caminos. Utiliza una pila (LIFO) para gestionar la frontera de consideración, lo que permite priorizar la exploración de ramas profundas antes que las superficiales.



File:Depth-First-Search.gif

Click on a date/time to view the file as it appeared at that time.

Wikipedia

DFS



Puede ser implementado como tree search o graph search

No garantiza encontrar la ruta más corta, pero es eficiente en términos de uso de memoria y puede ser útil en problemas donde la solución está en una rama profunda del árbol

BFS



Puede ser implementado como tree search o graph search

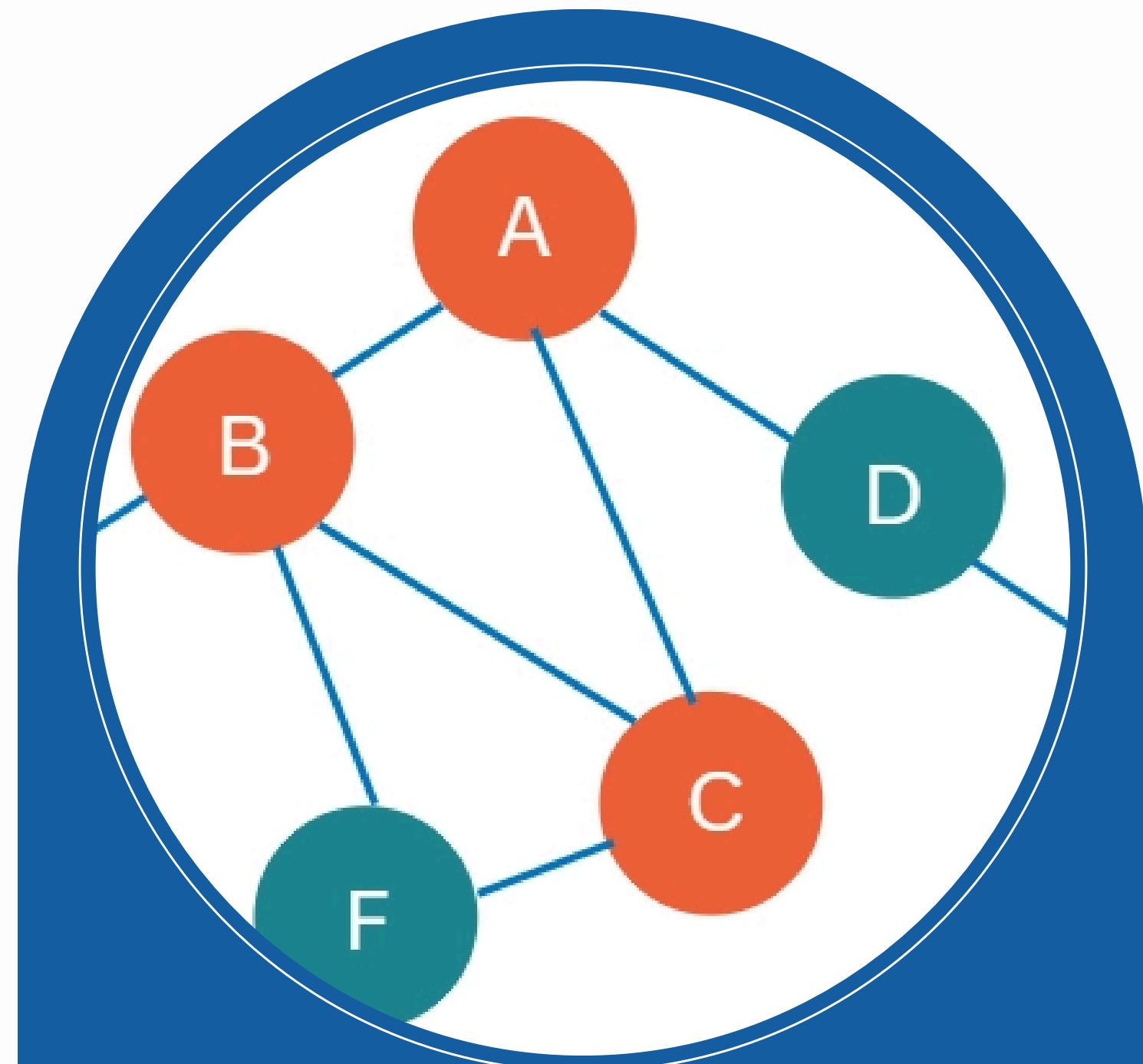
Garantiza que, si existe una solución, encontrará la ruta más corta desde el estado inicial hasta el estado objetivo. Esto lo hace ideal para problemas donde el costo de cada acción es uniforme.

Iterative-Deepening Search (IDS)

Es una combinación de las ventajas de BFS (Breadth-First Search) y DFS (Depth-First Search).

IDS funciona de la siguiente manera:

- Realiza una búsqueda en profundidad (DFS) con un límite de profundidad **L**.
- Si no encuentra la solución, incrementa el límite de profundidad **L** y repite la búsqueda.
- Continúa este proceso hasta encontrar la solución.



Uniform-Cost Search (UCS)

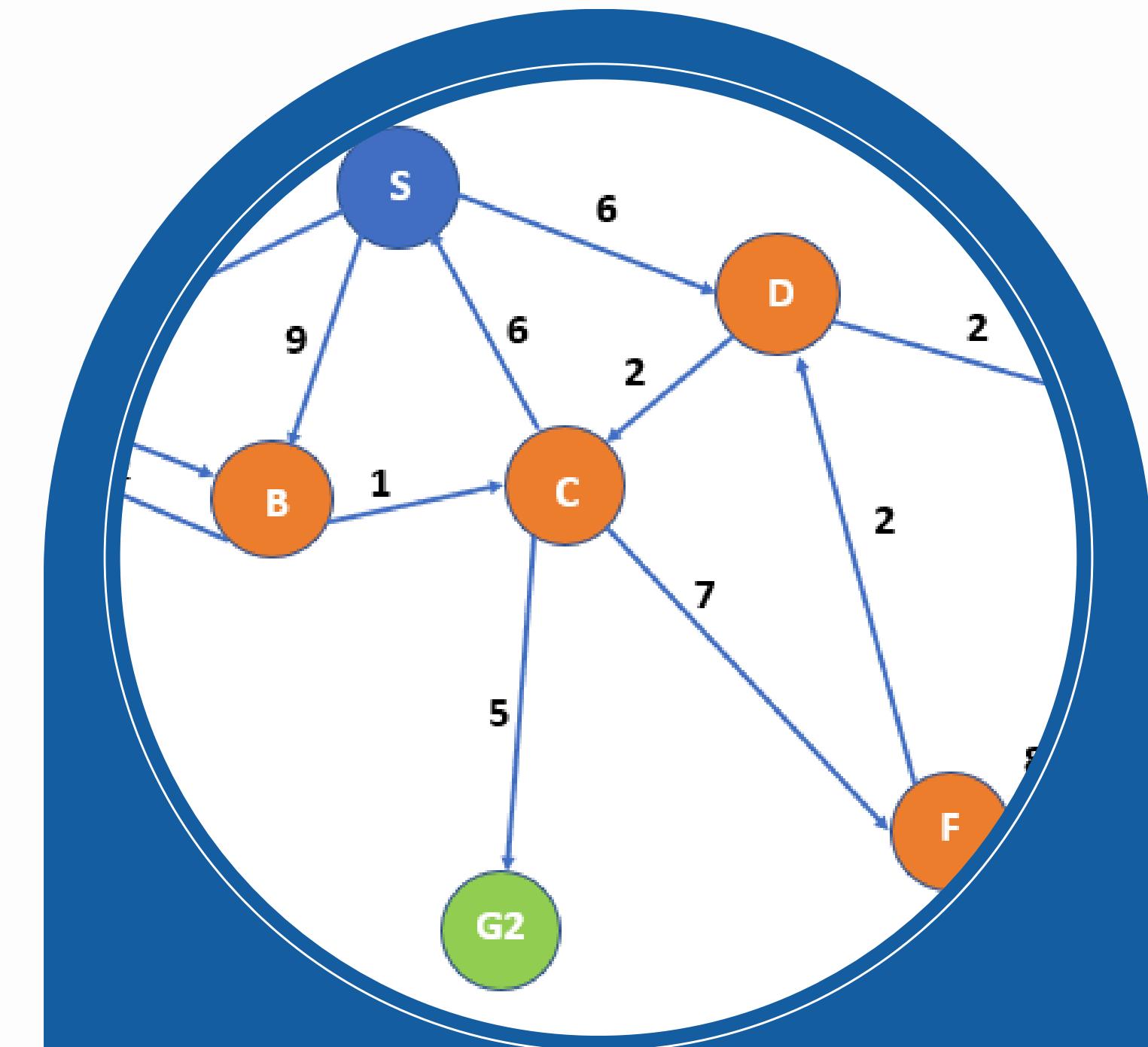
Es un algoritmo de búsqueda utilizado para encontrar la solución óptima en problemas donde cada acción tiene un costo asociado.

Es una variante del algoritmo de Dijkstra para grafos con costos en las aristas.

UCS explora un grafo o árbol de la siguiente manera:

1. Comienza desde el nodo raíz (o estado inicial).
2. Expande el nodo con el menor costo acumulado hasta el momento.
3. Repite este proceso hasta encontrar el estado objetivo o agotar todos los nodos.

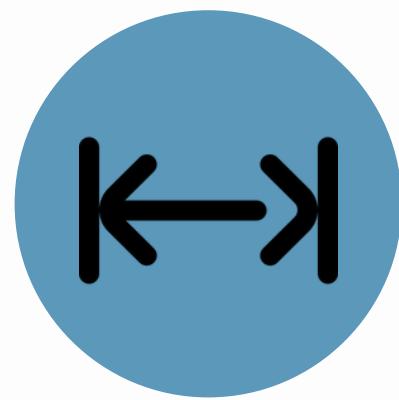
Mantiene una cola de prioridad donde los nodos se ordenan por su costo acumulado



Resumen

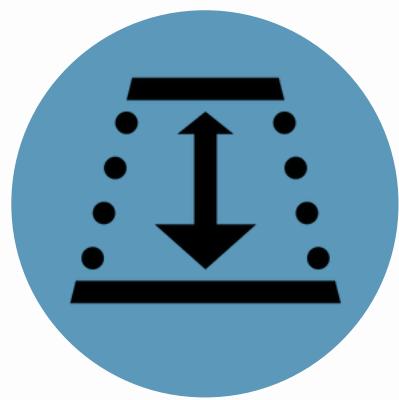
Característica	Búsqueda en Anchura (BFS)	Búsqueda en Profundidad (DFS)	Iterative Deepening Search (IDS)	Búsqueda de Costo Uniforme (UCS)
Estrategia	Explora nivel por nivel	Explora en profundidad antes de retroceder	Usa DFS con un límite de profundidad creciente	Expande el nodo con menor costo acumulado.
Estructura de Datos	Cola	Pila	Cola con control de profundidad	Cola de prioridad
Eficiencia Espacial	Alto uso de memoria.	Bajo uso de memoria.	Usa menos memoria que BFS.	Alto uso de memoria.
Eficiencia Temporal	Puede ser lento en problemas grandes.	Puede ser rápido, pero no garantiza mejor solución.	Mejor balance entre BFS y DFS.	Puede ser lento en problemas con costos variables.

¿Cuándo usar?



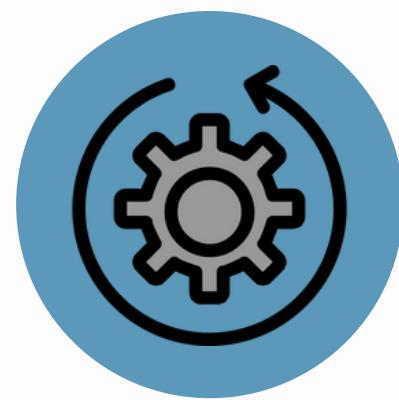
Búsqueda en Anchura (BFS)

Cuando necesitas la solución más corta y el espacio no es un problema.



Búsqueda en Profundidad (DFS)

Cuando quieres explorar rápidamente o el espacio es un problema.



Iterative Deepening Search (IDS)

Cuando necesitas la solución más corta y el espacio no es un problema.



Búsqueda de Costo Uniforme (ucs)

Cuando quieres explorar rápidamente o el espacio es un problema.

Algoritmos de Búsqueda Heurística

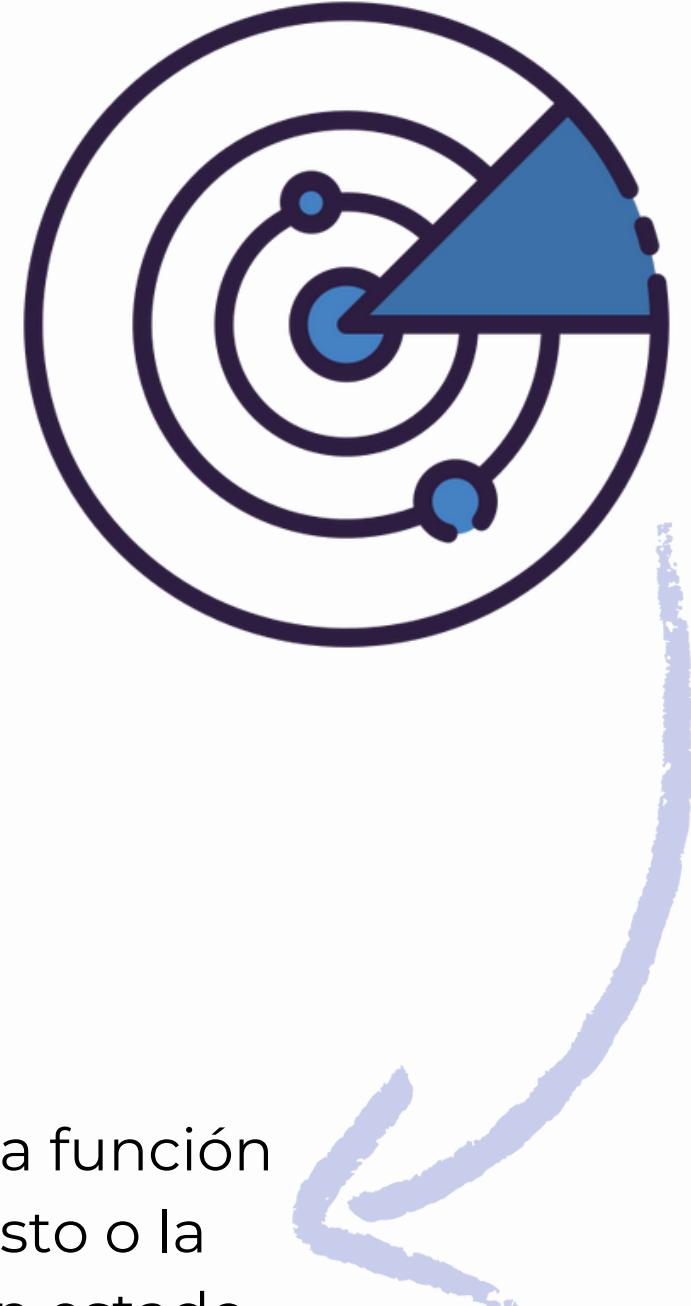
01

Greedy

02

A*

La heurística es una función que estima el costo o la distancia desde un estado actual hasta el objetivo en un problema de búsqueda



Heurística

La heurística no es un cálculo exacto, sino una aproximación que guía al algoritmo hacia la solución de manera más eficiente.

La calidad de la heurística influye directamente en el rendimiento del algoritmo: una buena heurística reduce el tiempo de búsqueda, mientras que una mala puede llevar a soluciones subóptimas o ineficientes.

- En mapas, se usa la distancia Euclidiana o Manhattan.
- En puzzles, se usan métricas como piezas mal colocadas o distancias.
- En problemas de optimización, se estiman costos restantes.

Búsqueda Voraz (Greedy Best-First Search)

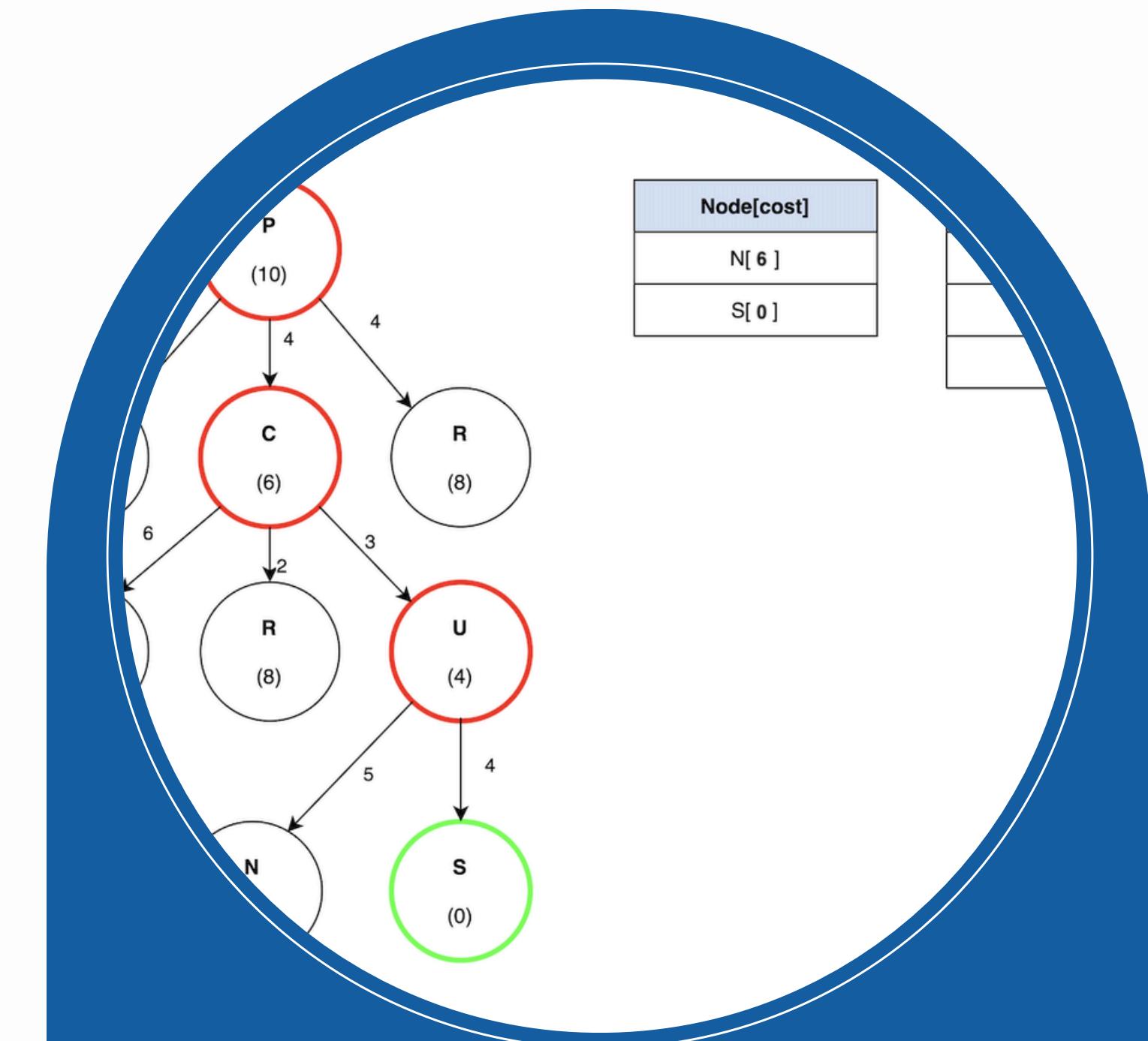
Es un algoritmo de búsqueda informada y no óptima que expande los nodos en función de una heurística que estima cuán cerca está un nodo del objetivo.

Se basa únicamente en la función de evaluación:

$$f(n) = h(n)$$

Donde:

- $h(n)$ es una heurística que estima el costo desde el nodo actual hasta el objetivo.



Es rápido y eficiente en ciertos casos, pero puede quedar atrapado en caminos sin salida o seleccionar caminos más largos que la solución óptima.

Búsqueda A* (A-Star Search)

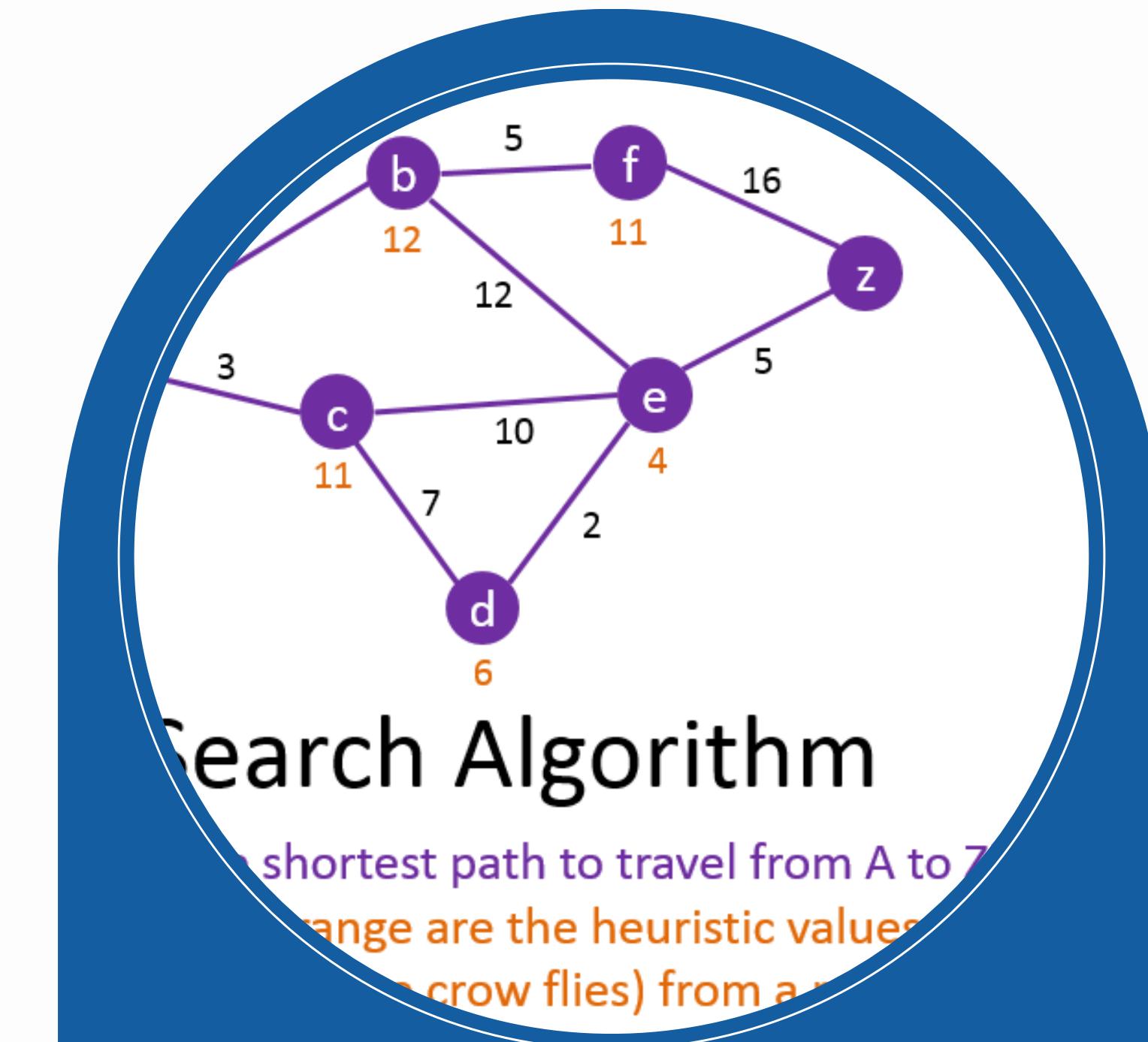
El algoritmo A* es una estrategia de búsqueda informada y óptima que combina Búsqueda de Costo Uniforme (UCS) y Búsqueda Greedy para encontrar el camino más corto hacia un objetivo.

A* evalúa cada nodo con la función de costo:

$$f(n) = g(n) + h(n)$$

Donde:

- $g(n)$ es el costo acumulado desde el nodo inicial hasta el nodo actual.
- $h(n)$ es una estimación heurística del costo desde el nodo actual hasta el objetivo.
- $f(n)$ es el costo total estimado del camino pasando por ese nodo.



A* search algorithm

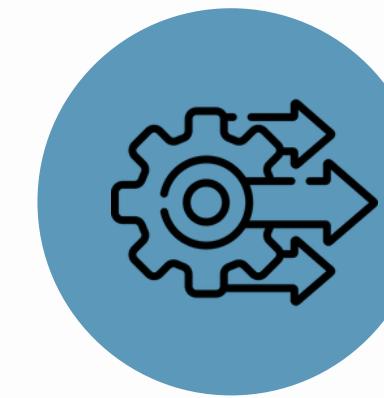
A* (pronounced "A-star") is a graph traversal and pathfinding algorithm that is used in many fields of computer science due to its completeness,...

w Wikipedia

Resumen

Característica	Greedy	A*
Estrategia	Usa solo una heurística, elige el nodo más prometedor sin considerar el costo acumulado.	Sí, siempre encuentra la mejor solución si la heurística es admisible.
¿Óptimo?	No garantiza la mejor solución.	Sí, siempre encuentra la mejor solución si la heurística es admisible.
Velocidad	Rápido en problemas simples.	Un poco más lento que Greedy pero más preciso.
Consumo de Memoria	Bajo (solo almacena nodos inmediatos).	Alto (almacena más información de costos y caminos).

¿Cuándo usar?



Greedy

Si buscas rapidez y no te importa que sea lo óptimo.



A*

Si necesitas una ruta óptima con eficiencia .

Comparaciones entre algoritmos de búsqueda

01

Algoritmos de Búsqueda Ciega

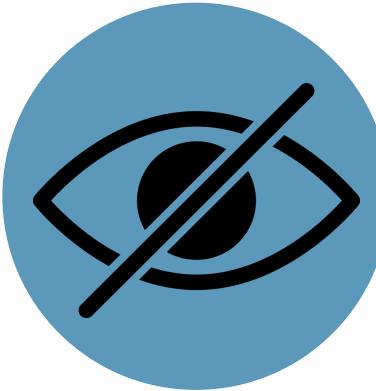
02

**Algoritmos de Búsqueda
Heurística**

Comparación

Característica	Búsqueda No Informada	Búsqueda Informada
Definición	Expande nodos en función de un criterio general como profundidad o costo.	Expande nodos basándose en una heurística que estima la distancia al objetivo.
Estrategia	Expande nodos en función de un criterio general como profundidad o costo.	Expande nodos basándose en una heurística que estima la distancia al objetivo.
¿Garantiza solución óptima?	<ul style="list-style-type: none">✓ UCS lo garantiza si los costos son positivos.✗ DFS y BFS no siempre.	<ul style="list-style-type: none">✓ A* lo garantiza si la heurística es admisible.✗ Greedy no siempre encuentra la solución óptima.
Eficiencia	Puede ser muy costosa en memoria y tiempo (ej. BFS en grafos grandes).	Generalmente más eficiente, ya que usa heurísticas para reducir la exploración innecesaria.

Cuándo usar cada Tipo de Búsqueda

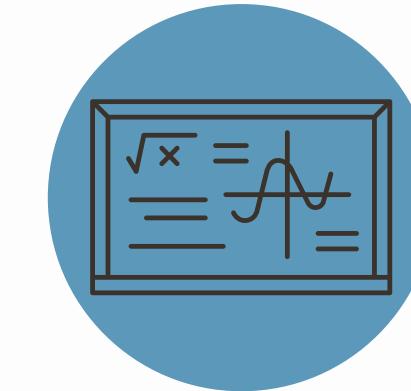


Búsqueda No Informada

Si no tienes información sobre la meta, usa búsqueda no informada.

Aplicaciones

Problemas donde no hay heurísticas disponibles, como exploración de estados desconocidos.



Búsqueda Informada

Si puedes estimar qué caminos son mejores, usa búsqueda informada para mayor eficiencia.

Aplicaciones

Problemas donde hay información previa sobre la solución, como juegos y navegación en mapas.

Aplicaciones Prácticas

01

Redes Sociales y Sistemas de Recomendación

02

Análisis de Código y Generación de Mapas

03

Rutas Óptimas en GPS y Redes de Transporte

04

Problemas en Juegos y Planeación de Movimientos

Búsqueda en Anchura

Redes sociales (Facebook, LinkedIn, Twitter, etc.)

Encontrar el número mínimo de conexiones entre dos personas en LinkedIn.

¿Cómo funciona?

- Se empieza con el usuario inicial.
- Se exploran todos sus amigos (nivel 1).
- Luego, los amigos de sus amigos (nivel 2), y así sucesivamente.
- La primera conexión encontrada es la más corta.

Búsqueda en Profundidad

Compiladores y detección de dependencias en código fuente.

Un compilador verifica si hay dependencias cíclicas en un código.

¿Cómo funciona?

- DFS explora todos los módulos hasta el final antes de retroceder.
- Si detecta un nodo ya visitado en el camino actual, hay un ciclo.
- Si no, sigue explorando otros módulos.

Búsqueda de Costo Uniforme

Solución de Problemas en Juegos y Planeación de Movimientos

Encontrar la mejor jugada en una partida de ajedrez.

¿Cómo funciona?

- Primero analiza todos los movimientos posibles en un nivel de profundidad.
- Luego aumenta el límite de profundidad y analiza más opciones.
- Se sigue repitiendo hasta que se encuentra la mejor jugada dentro del tiempo disponible.

Búsqueda de Costo Uniforme

Rutas Óptimas en GPS y Redes de Transporte

Encontrar la ruta con menor costo en tiempo o dinero.

¿Cómo funciona?

- Se comienza en la ubicación actual.
- Se evalúan todas las rutas posibles.
- Se selecciona la que tiene el menor costo acumulado (combustible, peajes, distancia).
- Se repite hasta llegar al destino.

Greedy

Sistemas GPS y videojuegos de estrategia

Un GPS que siempre elige la carretera más corta hacia el destino.

¿Cómo funciona?

- Evalúa solo la distancia actual hasta la meta.
- Selecciona el siguiente punto que parezca más cercano.
- No considera si el camino total es óptimo.

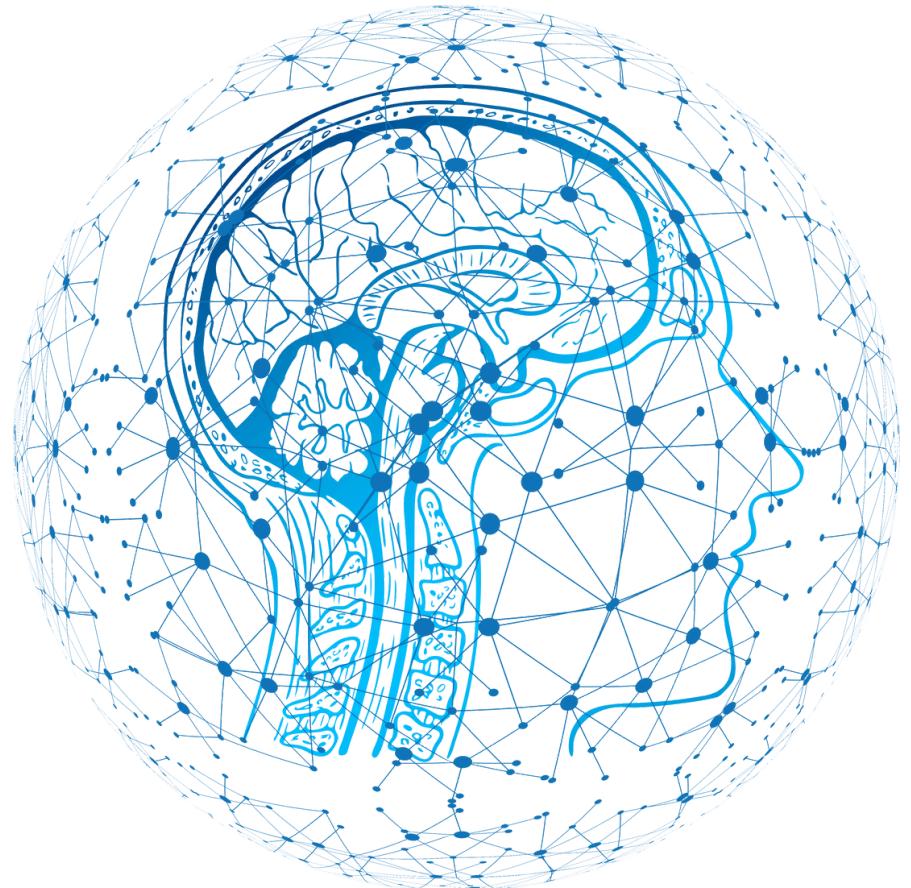
A*

Optimización de Rutas y Robótica

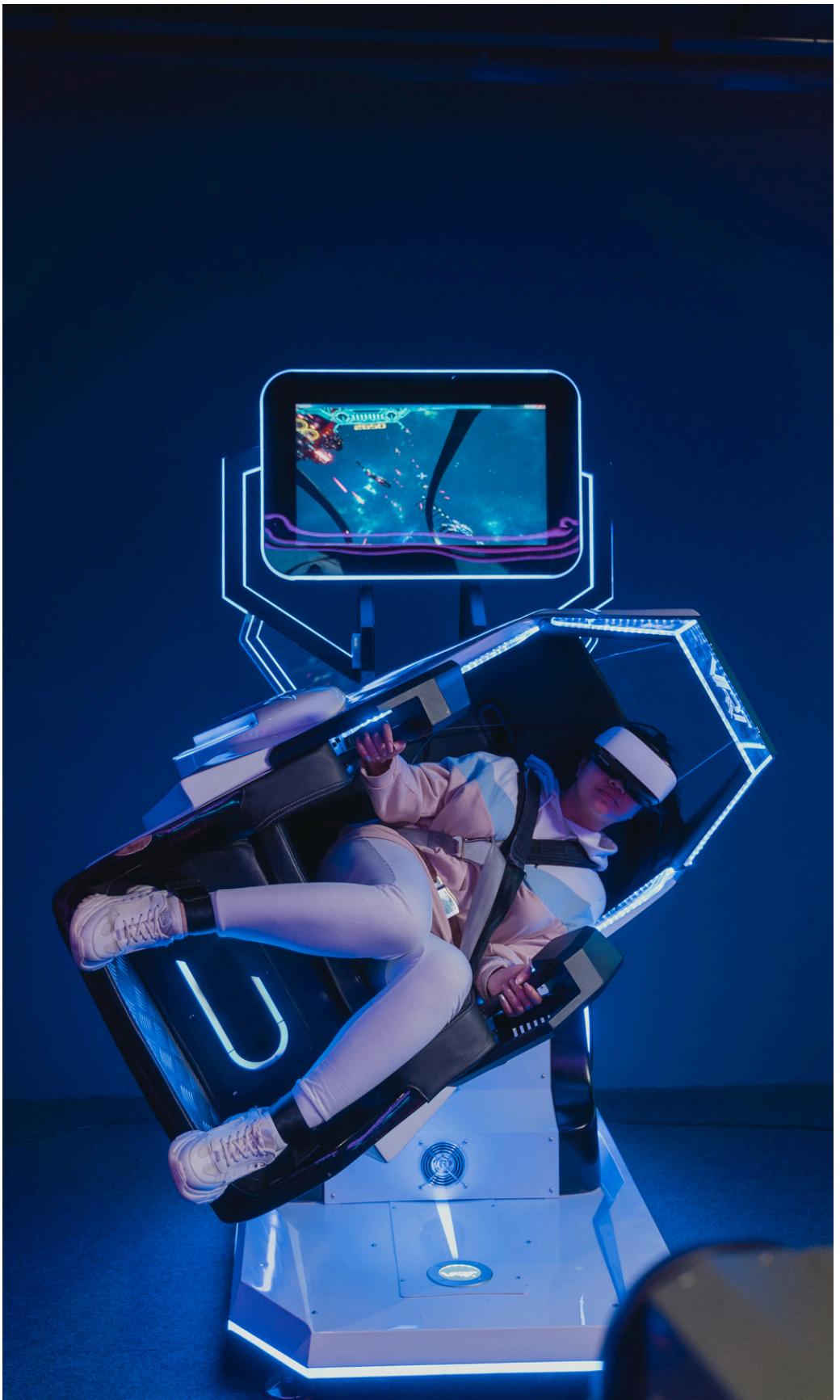
Un robot de entrega autónomo que encuentra la ruta más rápida evitando obstáculos.

¿Cómo funciona?

- Evalúa la distancia recorrida hasta el momento (costo real).
- También estima la distancia restante hasta la meta (heurística).
- Usa la suma de ambos valores para decidir la mejor ruta.



Constraint Satisfaction Problems

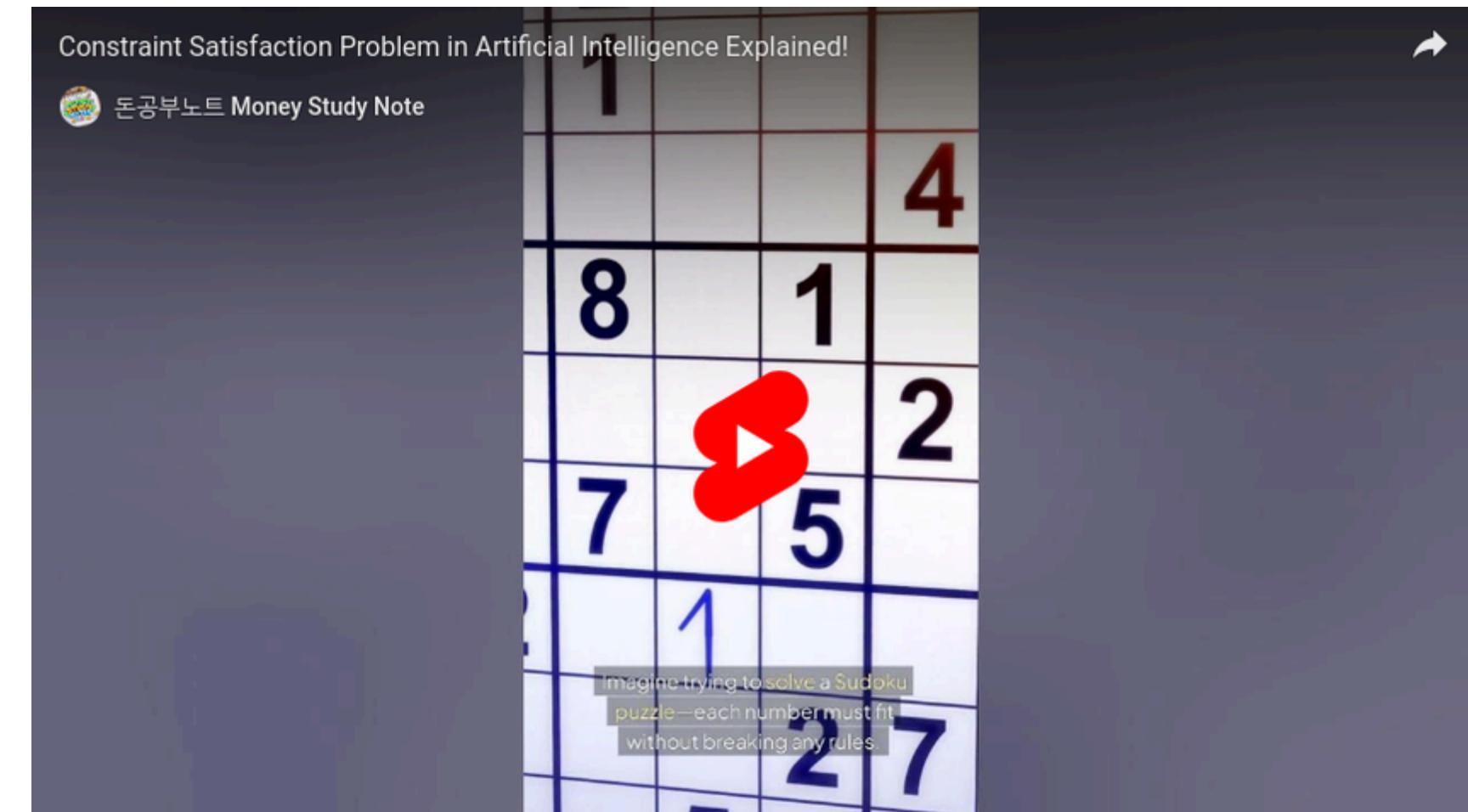


Agenda

- Definición
- Elementos de un CSP
- Ejemplos
- Métodos de resolución
- Aplicaciones

Definición

Son un tipo de problema en inteligencia artificial en los que se busca asignar valores a un conjunto de variables, cumpliendo ciertas restricciones o condiciones predefinidas.





Elementos

01

Conjunto de Variables (X)

Un conjunto finito de variables X

02

Dominios (D)

Cada variable X tiene un dominio D de posibles valores que puede tomar.

03

Restricciones (C)

Un conjunto de condiciones que especifican combinaciones permitidas de valores para ciertas variables

04



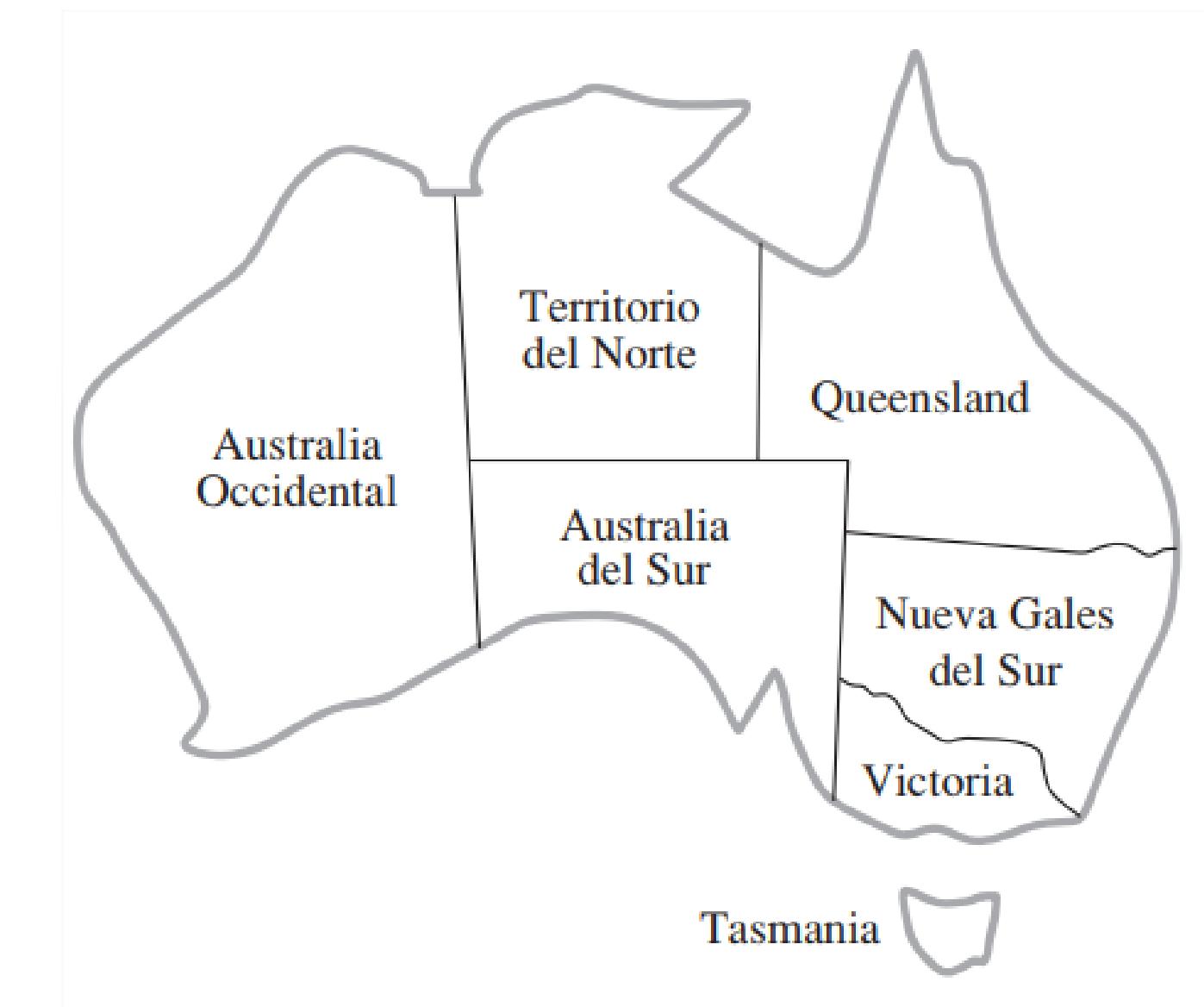
Tipos de Restricciones

- 01 **Unitarias** Afectan solo una variable
- 02 **Binarias** Afectan a dos variables
- 03 **High-Order** Afectan a tres o más variables
- 04

Ejemplo

Problema de Coloreo de Mapas

Dado un mapa con varias regiones, el objetivo es asignar un color a cada región de manera que ninguna región vecina comparta el mismo color.



Componentes del CSP

Variables: Cada región del mapa

Si tomamos el mapa de Australia con las siguientes regiones:

- WA (Australia Occidental)
- NT (Territorio del Norte)
- SA (Australia del Sur)
- Q (Queensland)
- NSW (Nueva Gales del Sur)
- V (Victoria)
- T (Tasmania)

$$\mathcal{X} = \{WA, NT, Q, NSW, V, SA, T\}$$

Dominios: Un conjunto de colores posibles

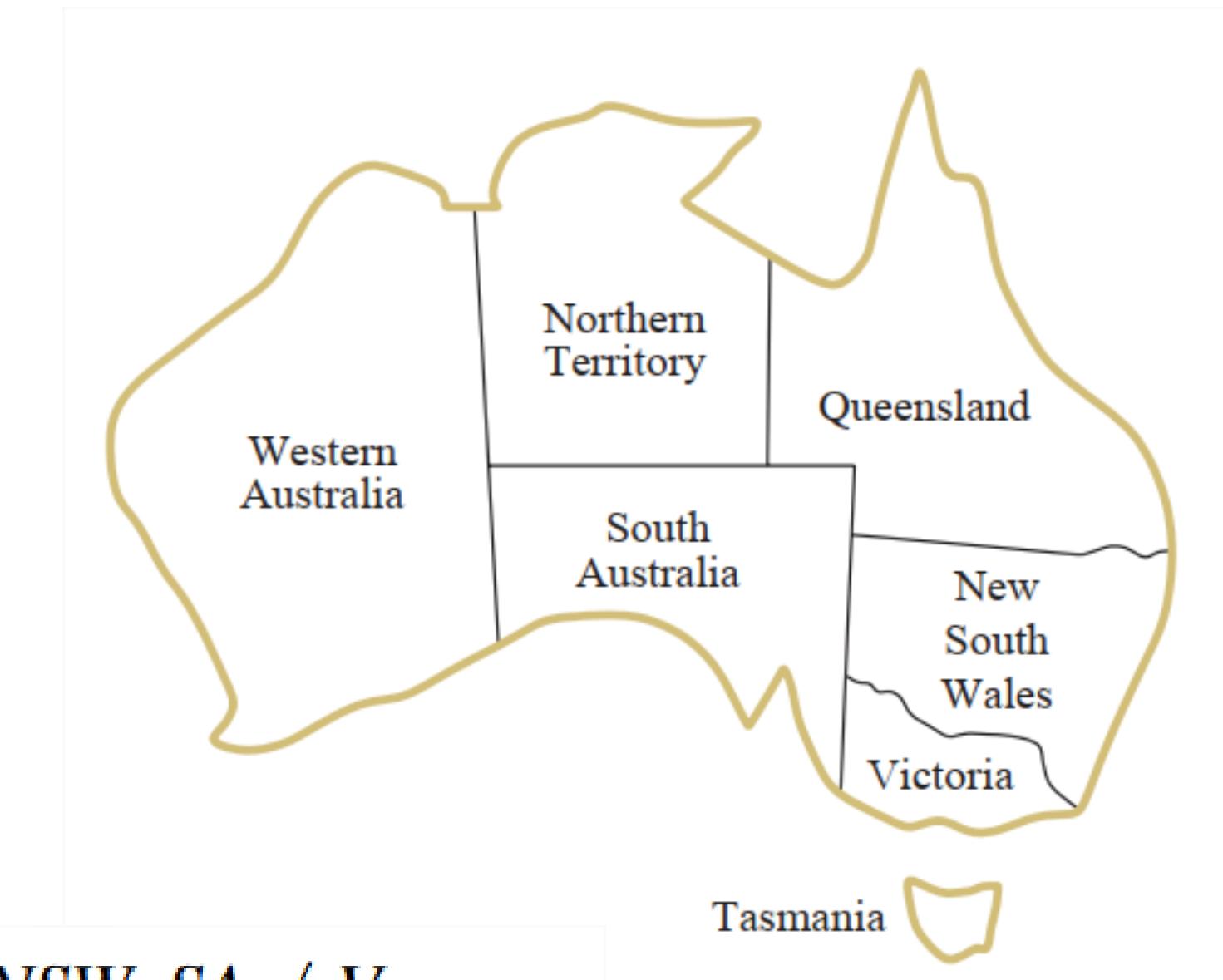
$$D = \{\text{rojo}, \text{verde}, \text{azul}\}$$

Restricciones: Dos regiones adyacentes no pueden tener el mismo color.

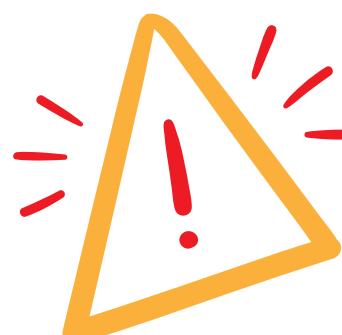
Cada región puede tener uno de tres colores: {rojo, verde, azul}. Las restricciones vienen de los límites geográficos entre regiones.

Restricciones

WA \neq NT
WA \neq SA
NT \neq SA
NT \neq Q
SA \neq Q
SA \neq NSW
SA \neq V
Q \neq NSW
NSW \neq V



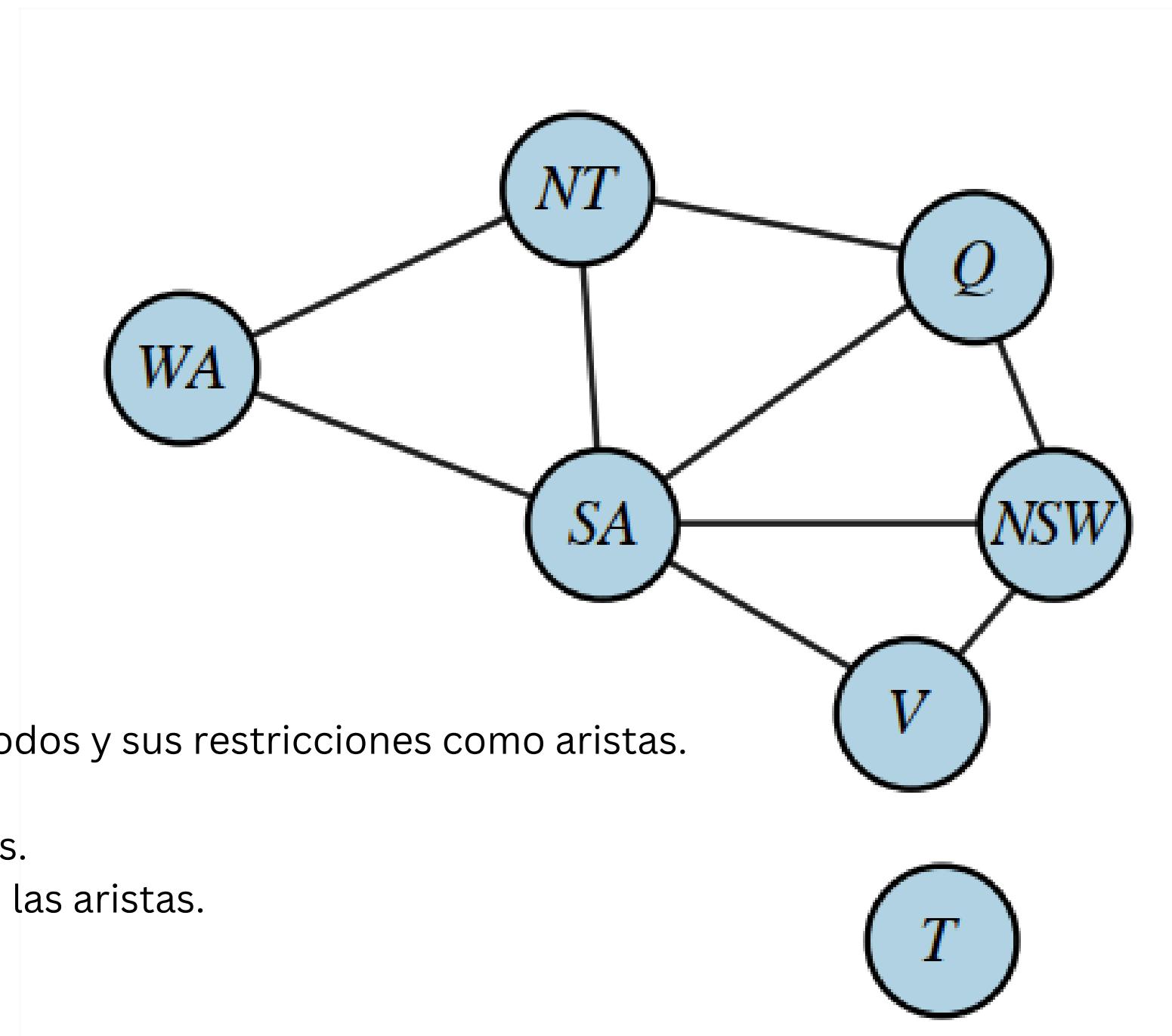
$$\mathcal{C} = \{SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NSW, SA \neq V, \\ WA \neq NT, NT \neq Q, Q \neq NSW, NSW \neq V\}.$$



Tasmania (T) no tiene restricciones porque no comparte frontera con ninguna otra región.



Grafo de Restricciones



Un grafo permite representar regiones como nodos y sus restricciones como aristas.

Ejemplo con Australia:

- Las regiones (WA, NT, SA, etc.) son los nodos.
- Las restricciones (WA ≠ NT, SA ≠ Q, etc.) son las aristas.

Esto ayuda a visualizar qué regiones están conectadas y deben cumplir la restricción de colores distintos.

Implementing CSP Algorithm

Representing the problem

Selecting a variable

Selecting a value

Termination

Backtracking

Applying constraints



Algoritmos para CSP

- **Backtracking (búsqueda con retroceso):** asigna valores secuencialmente y retrocede si hay conflictos.
- **Forward Checking:** evita asignaciones que llevarían a un fallo, reduciendo el espacio de búsqueda.
- **Arc Consistency:** Asegura que todas las restricciones binarias sean consistentes. (AC-3) para reducir el dominio de valores posibles antes de la búsqueda.
- **Heurísticas:**
 - **Minimum Remaining Values (MRV):** Selecciona la variable con menos valores posibles.
 - **Degree Heuristic:** Selecciona la variable involucrada en más restricciones.

Backtracking

Arrays:

- **VARIABLES**
- **RESTRICCIONES**
- **DOMINIO:** se recorre en la función ORDEN-VALORES-DOMINIO()

función BÚSQUEDA-CON-VUELTA-ATRÁS(*psr*) **devuelve** una solución, o fallo
devolver VUELTA-ATRÁS-RECURSIVA({ }, *psr*)

función VUELTA-ATRÁS-RECURSIVA(*asignación*, *psr*) **devuelve** una solución, o fallo
si *asignación* es completa **entonces devolver** *asignación*
var \leftarrow SELECCIONA-VARIABLE-NOASIGNADA(VARIABLES[*psr*], *asignación*, *psr*)
para cada *valor* **en** ORDEN-VALORES-DOMINIO(*var*, *asignación*, *psr*) **hacer**
si *valor* es consistente con *asignación* de acuerdo a las RESTRICCIONES[*psr*] **entonces**
 añadir {*var* = *valor*} a *asignación*
 resultado \leftarrow VUELTA-ATRÁS-RECURSIVA(*asignación*, *psr*)
 si *resultado* \neq fallo **entonces devolver** *resultado*
 borrar {*var* = *valor*} de *asignación*
devolver fallo

Figura 5.3 Un algoritmo simple de vuelta atrás para problemas de satisfacción de restricciones. El algoritmo se modela sobre la búsqueda primero en profundidad recursivo del Capítulo 3. Las funciones SELECCIONA-VARIABLE-NOASIGNADA y ORDEN-VALORES-DOMINIO pueden utilizarse para implementar las heurísticas de propósito general discutidas en el texto.

Pasos del Algoritmo Aplicado al Ejemplo

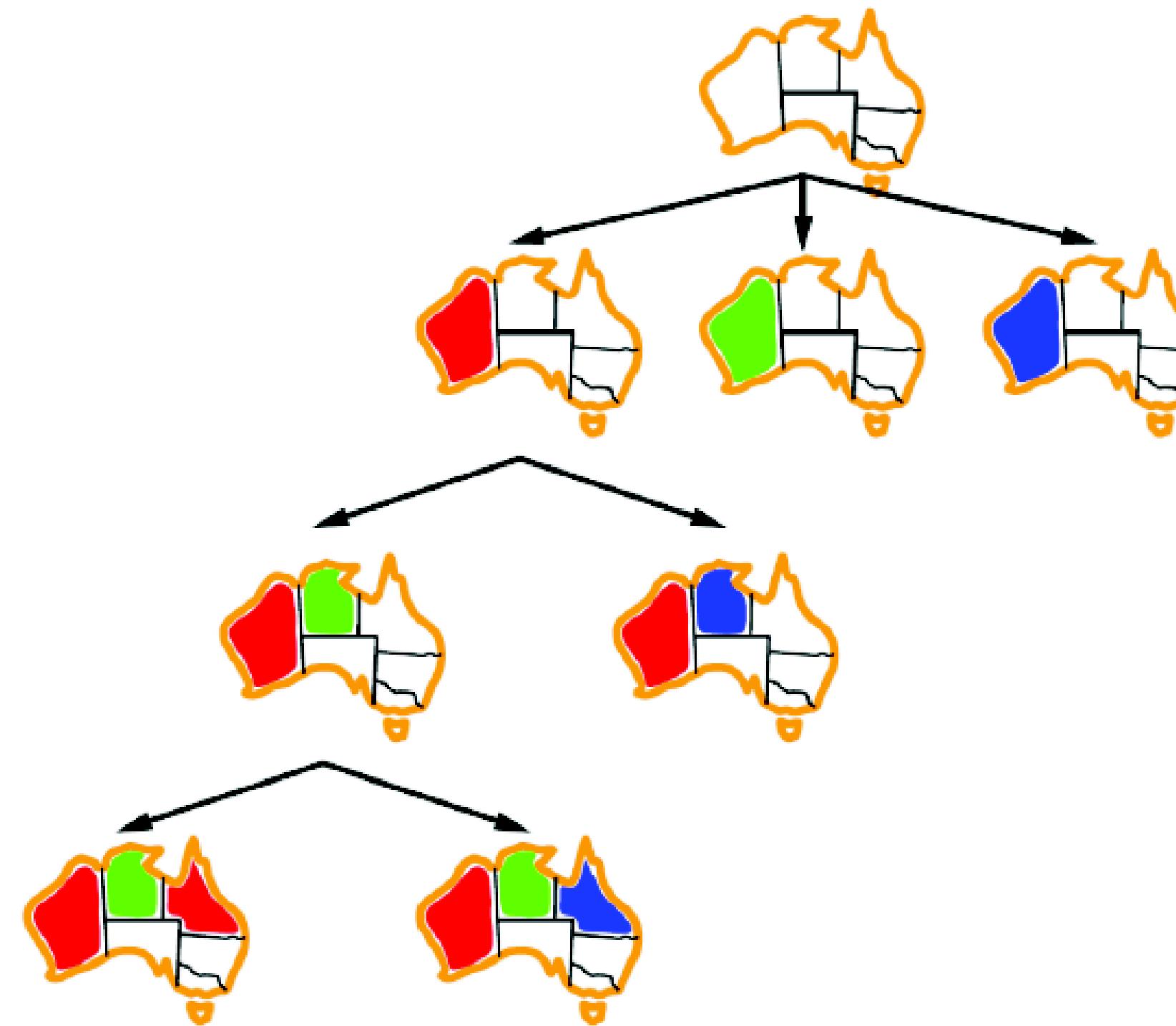
El backtracking explora asignaciones de colores y retrocede si encuentra un conflicto.

Se sigue el siguiente proceso:

1. Seleccionar una variable no asignada: Elige una región sin color asignado.
2. Asignar un valor: Asigna un color del dominio a la región seleccionada.
3. Verificar restricciones: Comprueba si la asignación viola alguna restricción con las regiones adyacentes ya coloreadas.
4. Si no hay violación, continúa con la siguiente región.
5. Si hay violación, deshaz la asignación y prueba con otro color.
6. Repetir: Repite el proceso hasta que todas las regiones estén coloreadas o se determine que no hay solución con los colores disponibles.
7. Retroceder (Backtrack): Si no se puede asignar un color válido a una región, retrocede a la región anterior y prueba con otro color.

Pasos del Algoritmo Aplicado al Ejemplo

El backtracking explora asignaciones de colores y retrocede si encuentra un conflicto.



Ejecución Paso a Paso

Paso 1: Iniciar con una asignación vacía

Asignación inicial: { }

Paso 2: Seleccionar la primera variable

Variable elegida: WA

Colores disponibles: {Rojo, Verde, Azul}

- ◆ Asignamos WA = Rojo

Asignación parcial: { WA = Rojo }

Paso 3: Seleccionar la siguiente variable

Variable elegida: NT

Colores disponibles: {Rojo, Verde, Azul}

- ◆ Probamos NT = Rojo

✗ No es válido porque WA = Rojo y WA ≠ NT (backtracking).

- ◆ Probamos NT = Verde

✓ Válido.

Asignación parcial: { WA = Rojo, NT = Verde }

Ejecución Paso a Paso

Paso 4: Seleccionar la siguiente variable

Variable elegida: SA

Colores disponibles: {Rojo, Verde, Azul}

- ◆ Probamos SA = Rojo
- ✗ No es válido porque WA = Rojo y WA ≠ SA (backtracking).
- ◆ Probamos SA = Azul
- ✓ Válido.

Asignación parcial: { WA = Rojo, NT = Verde, SA = Azul }

Paso 5: Seleccionar la siguiente variable

Variable elegida: Q

Colores disponibles: {Rojo, Verde, Azul}

- ◆ Probamos Q = Rojo
- ✗ No es válido porque NT = Verde y NT ≠ Q (backtracking).
- ◆ Probamos Q = Azul
- ✗ No es válido porque SA = Azul y SA ≠ Q (backtracking).
- ◆ Probamos Q = Verde
- ✓ Válido.

Asignación parcial: { WA = Rojo, NT = Verde, SA = Azul, Q = Verde }

Ejecución Paso a Paso

Paso 6: Seleccionar la siguiente variable

Variable elegida: NSW

Colores disponibles: {Rojo, Verde, Azul}

◆ Probamos NSW = Rojo

✓ Válido.

Asignación parcial: { WA = Rojo, NT = Verde, SA = Azul, Q = Verde, NSW = Rojo }

Paso 7: Seleccionar la siguiente variable

Variable elegida: V

Colores disponibles: {Rojo, Verde, Azul}

◆ Probamos V = Rojo

✗ No es válido porque NSW = Rojo y NSW ≠ V (backtracking).

◆ Probamos V = Verde

✓ Válido.

Asignación parcial: { WA = Rojo, NT = Verde, SA = Azul, Q = Verde, NSW = Rojo, V = Verde }

Ejecución Paso a Paso

Paso 8: Seleccionar la última variable

Variable elegida: T

Tasmania (T) no tiene restricciones con otras regiones, por lo que se puede asignar cualquier color.

- ◆ Asignamos T = Rojo
- ✓ Válido.

Asignación final: { WA = Rojo, NT = Verde, SA = Azul, Q = Verde, NSW = Rojo, V = Verde, T = Rojo }

¡Solución encontrada con backtracking!

Colores asignados a cada región:

- WA = Rojo
- NT = Verde
- SA = Azul
- Q = Verde
- NSW = Rojo
- V = Verde
- T = Rojo

Forward Checking

Qué hace:

Después de asignar un valor a una variable, elimina inmediatamente los valores inconsistentes de los dominios de las variables vecinas (relacionadas por restricciones).

Cuándo se aplica:

Durante la búsqueda (por ejemplo, en backtracking), solo cuando se asigna un valor a una variable.

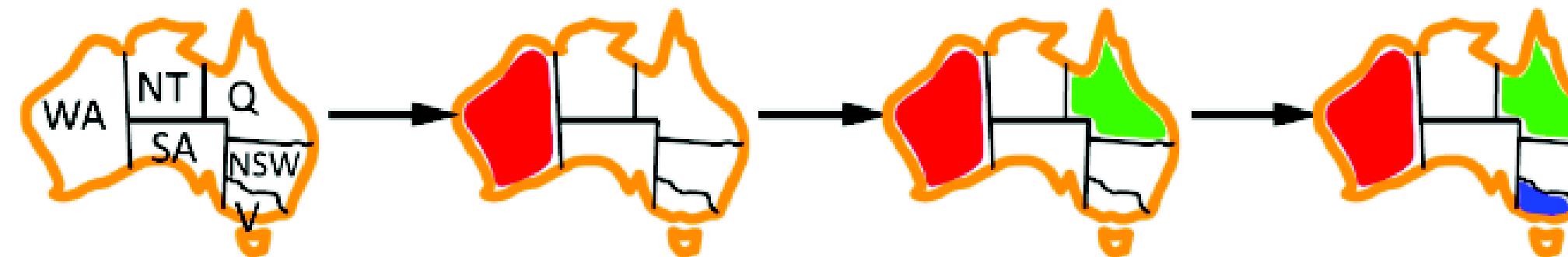
Ejemplo:

Imagina un Sudoku. Si colocas un "5" en una celda, el Forward Checking eliminaría el "5" de las celdas vecinas (misma fila, columna y región).

Limitación:

Solo mira un paso hacia adelante (variables directamente relacionadas). No garantiza que todas las restricciones del problema sean consistentes.

Forward Checking



	WA	NT	Q	NSW	V	SA	
Paso 1	Red	Green	Blue	Red	Green	Blue	
Paso 2	Red	White	Green	Blue	Red	Green	Blue
Paso 3	Red	White	White	Green	Red	Green	Blue
Paso 4	Red	White	White	Green	Red	White	Blue

Regla para este ejemplo, resolvemos de izquierda a derecha, tomando la variable con mayor numero de opciones disponibles

NOTA: Podemos concluir en el paso 4, que la actual distribucion no llevara a un estado de solucion consistente

Arc Consistency

Qué hace:

Asegura que todos los pares de variables conectadas por una restricción sean consistentes.

Cuándo se aplica:

Generalmente como preprocesamiento (antes de la búsqueda) o durante la búsqueda para reducir dominios.

Ejemplo:

En un problema de colorear un mapa, la consistencia de arco revisa que, para cada región adyacente, los colores disponibles no se solapen.

Ventaja:

Es más completa que el Forward Checking, ya que revisa todas las restricciones entre pares de variables, no solo las afectadas por una asignación reciente.

Arc Consistency

Ayuda a no cometer errores antes de empezar. Si revisas todas las reglas y eliminas las combinaciones que no funcionan

Para que haya **consistencia de arco** entre X e Y, cada valor en Dx debe tener al menos un valor en Dy que satisfaga la restricción, y viceversa.

Por ejemplo:

Si $X=1$, entonces Y puede ser 2,3,4 (todos son mayores que 1)

Si $X=2$, entonces Y puede ser 3,4 (mayores que 2).

Si $X=3$, entonces Y solo puede ser 4 (mayor que 3).

Variables: X e Y

Dominios:

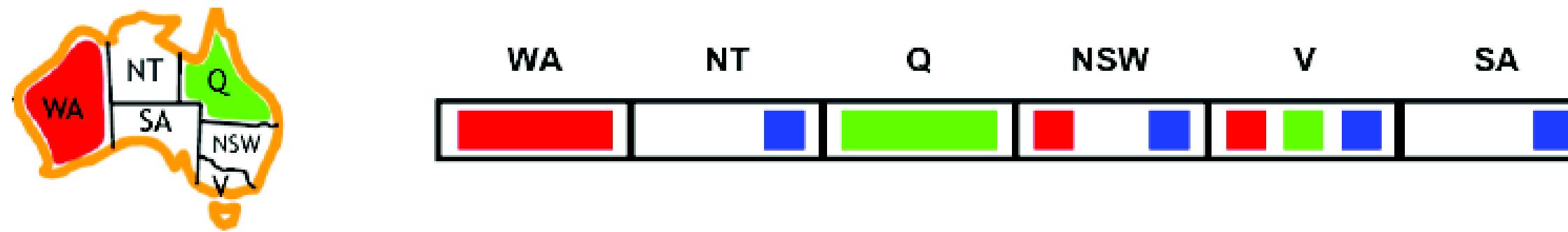
$D_x = \{1, 2, 3\}$

$D_y = \{2, 3, 4\}$

Restricciones: $X < Y$

Arc Consistency

Es una forma de propagación donde se asegura que todos los CSP son consistentes.



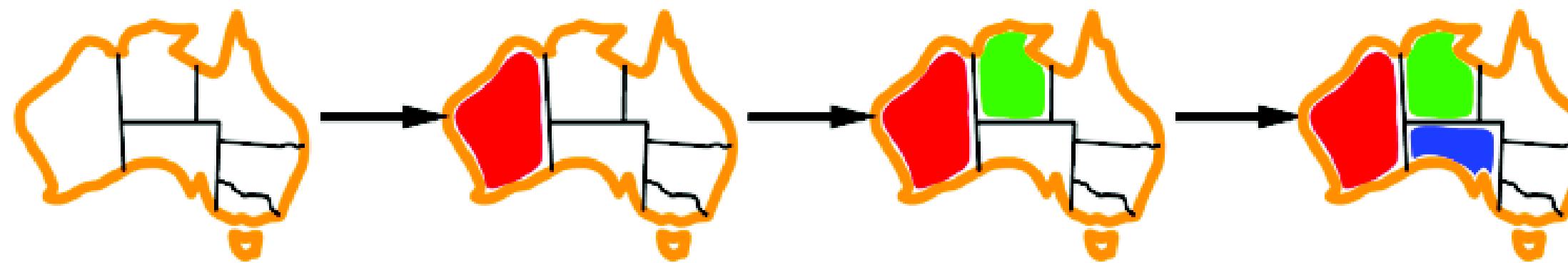
- Si X pierde un valor, los vecinos de X deben ser verificados
- Puede ser ejecutado como un pre-procesador o despues de cada asignación

Diferencias

Característica	Forward Checking	Consistencia de Arco
Alcance	Local (solo variables vecinas)	Global (todos los pares de variables)
Momento de aplicación	Durante la búsqueda (tras asignar valor)	Antes o durante la búsqueda
Complejidad	Más rápida (solo mira variables vecinas)	Más costosa (revisa todos los pares)
Pruning (poda)	Elimina valores de variables vecinas.	Elimina valores de cualquier variable

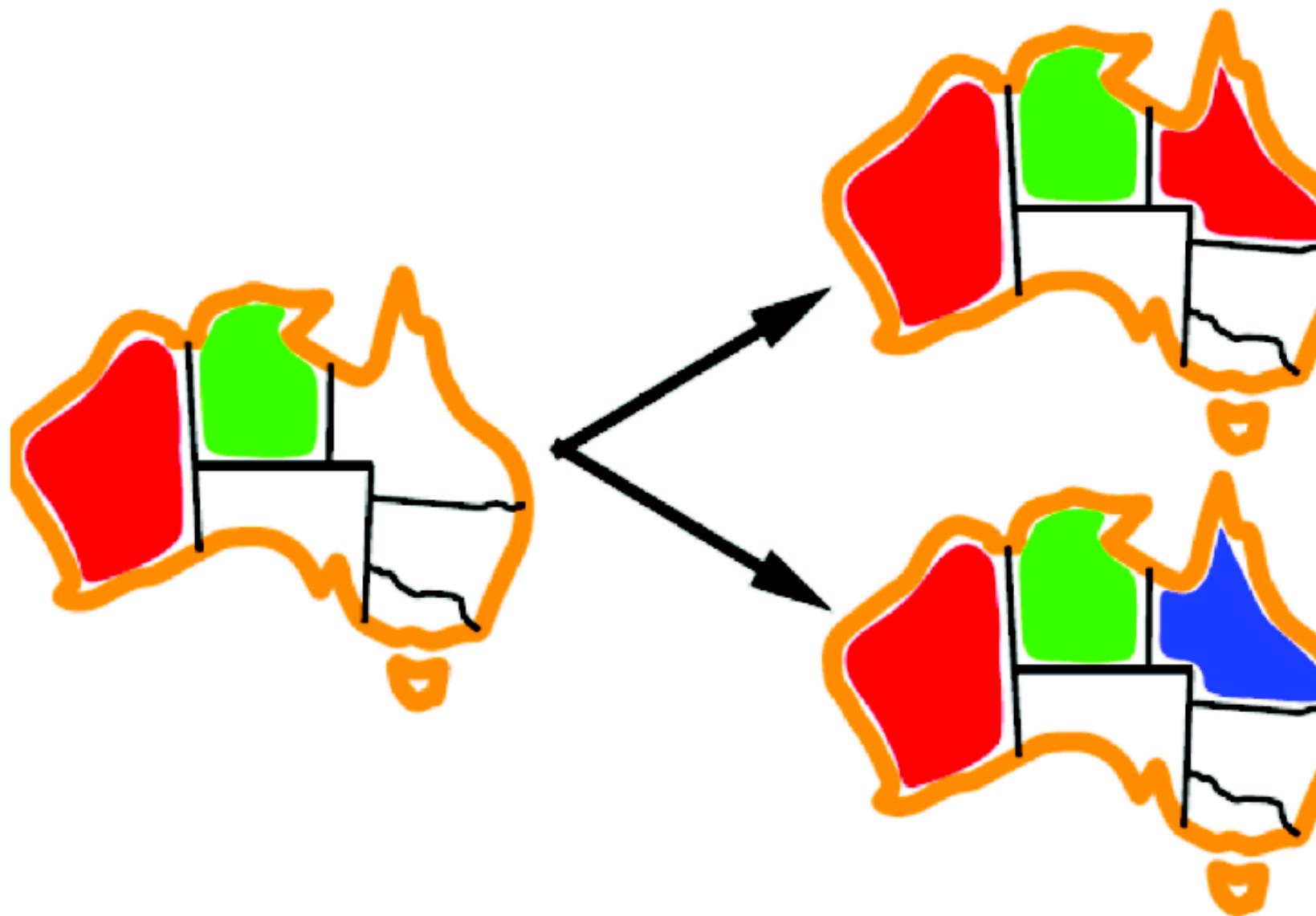
Minimum Remaining Value

Elegir la **variable** con la menor cantidad de valores posibles en su dominio



Least Constraining Value

Elegir el **valor** que presente mayores opciones en el futuro



Aplicaciones

Horarios y Planificación

- *Ejemplo:* Asignar horarios de clases a profesores y aulas sin que haya conflictos de asignaciones.
- *Aplicación:* Planificación académica, gestión de turnos en hospitales o empresas.



Aplicaciones

Resolución de Juegos de Lógica

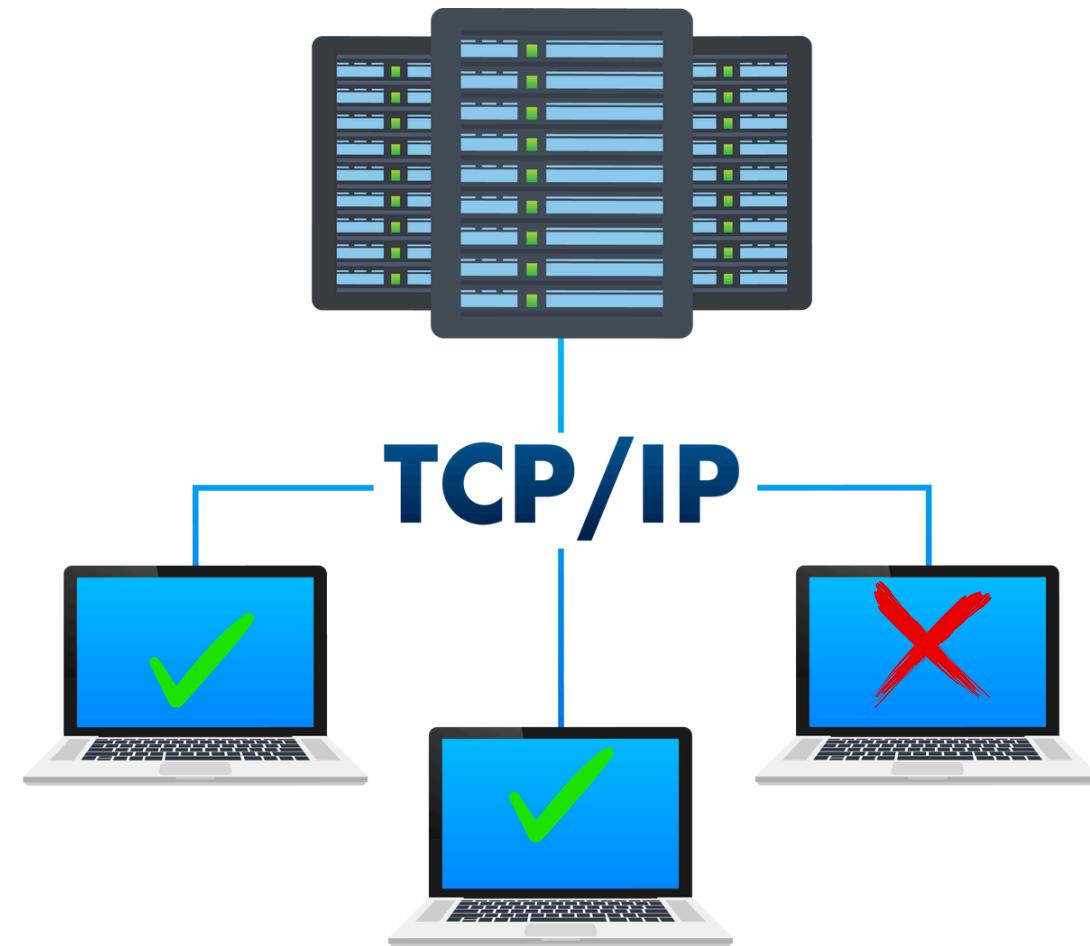
- *Ejemplo:* Llenar una cuadrícula de Sudoku cumpliendo restricciones de números en filas, columnas y subcuadrículas.
- *Aplicación:* Creación de videojuegos, optimización de tableros en juegos de estrategia.



Aplicaciones

Redes de Computadoras

- *Ejemplo:* Asignar direcciones IP y rutas en una red sin que haya conflictos.
- *Aplicación:* Optimización de redes de datos, seguridad en comunicación digital.



Aplicaciones

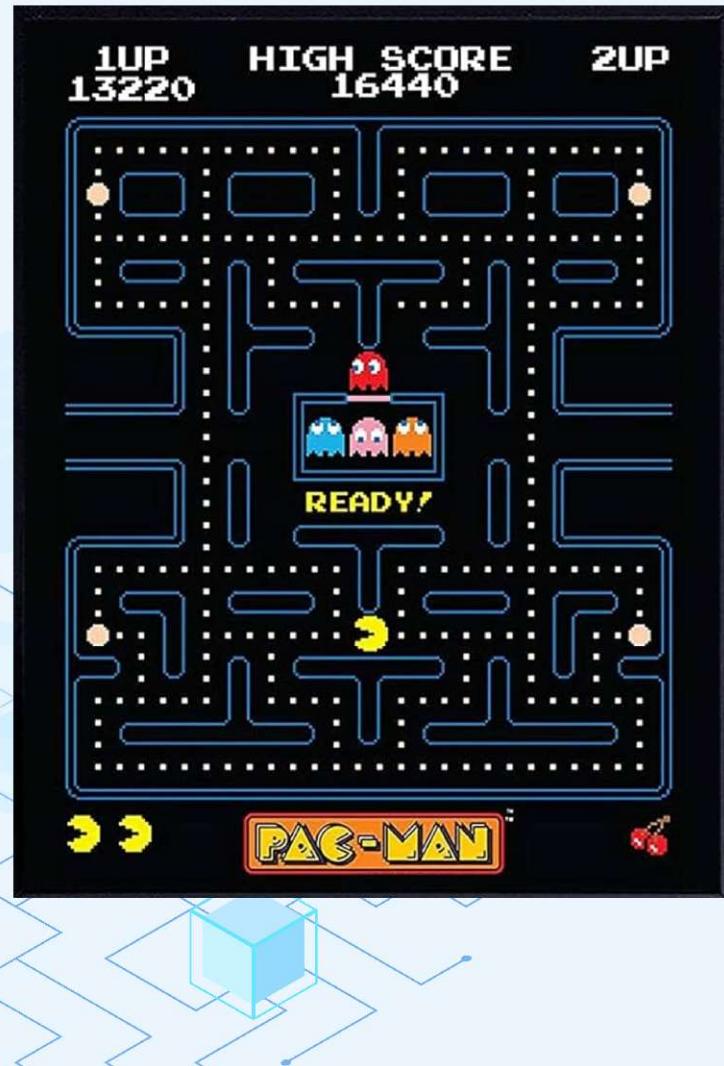
Logística y Transporte

- *Ejemplo:* Rutas de Entrega (Ej: Amazon, UPS).
- *Aplicación:* Optimizar rutas de reparto para minimizar tiempo y combustible, respetando ventanas de entrega y capacidad de vehículos.
- *Restricciones:* Tráfico, peso máximo, horarios de clientes



Resumen - Mapa Conceptual





MINIMAX

BÚSQUEDA ADVERSARIA
Agentes que maximizan

Donde examinaremos los problemas que surgen cuando tratamos de planear en un mundo donde otros agentes planean contra nosotros.



Agenda

- Fundamentos
- Teoría de Juegos
 - Dilema del prisionero
- Juegos de Suma Cero
- Problemas de Búsqueda Adversaria
- Algoritmo Minimax
- Optimizaciones
- Aplicaciones
- Ventajas y Desventajas

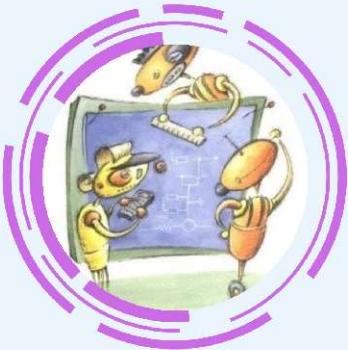


Recordemos...



Agentes Racionales

Cuando los agentes toman decisiones basadas en la maximización de la utilidad esperada.



Entorno Multiagente

En dónde cualquier agente tendrá que considerar las acciones de otros agentes y cómo afectan a su propio bienestar.



Entorno Competitivo

Es un caso especial de entorno multiagente en el que los agentes tienen objetivos opuestos.



Entorno Cooperativo

Es aquel en el que múltiples agentes trabajan juntos para alcanzar un objetivo común.

Recordemos...



Entorno Determinista

Es aquel en el que el resultado de cada acción siempre es predecible y no hay elementos de azar o incertidumbre.



Entorno Estocástico

Es aquel en el que el resultado de una acción no es completamente predecible porque hay un elemento de azar o incertidumbre involucrado.



Fundamentos



Teoria de Juegos

Es un campo de estudio matemático que analiza modelos estratégicos donde múltiples agentes toman decisiones interdependientes.

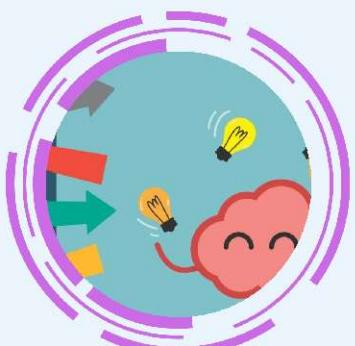
Se utiliza en economía, inteligencia artificial y teoría de decisiones, modela situaciones competitivas y cooperativas.



Busqueda Adversaria

Son juegos en los que los jugadores tienen intereses opuestos y buscan maximizar sus propios resultados mientras minimizan los del oponente.

Ejemplos incluyen el ajedrez, el go y el tres en raya.



Toma de Decisiones

En los juegos adversarios, la toma de decisiones se basa en estrategias que consideran tanto las propias acciones como las del oponente



Juegos de Suma Cero

Juegos donde la ganancia de un jugador es exactamente la pérdida del otro.

En este tipo de juegos, el objetivo es maximizar la propia ganancia mientras se minimiza la del oponente, como en el ajedrez o el póker.



Teoría de Juegos



Desarrollada en sus comienzos como una herramienta para entender el comportamiento de la economía, la teoría de juegos se ha ido extendiendo a muchos otros campos, como la biología, las ciencias de la computación, la sociología, la politología, la psicología y la filosofía.

A raíz de juegos como el dilema del prisionero, en los que el egoísmo generalizado perjudica a los jugadores, la teoría de juegos ha atraído también la atención de los investigadores en informática, usándose en inteligencia artificial y cibernética



Dilema del prisionero

La policía arresta a dos sospechosos.

No hay pruebas suficientes para condenarlos y, tras haberlos separado, los visita a cada uno y les ofrece el mismo trato.

Si uno confiesa y su cómplice no, el cómplice será condenado a la pena total, tres años, y el primero será liberado.

Si uno calla y el cómplice confiesa, el primero recibirá esa pena y será el cómplice quien salga libre.

Si ambos confiesan, ambos serán condenados a dos años.

Si ambos lo niegan, todo lo que podrán hacer será encerrarlos durante un año por un cargo menor.

	B A	 B stays silent	 B testifies
 A stays silent	  R, -1 R, -1	  S, -3 T, 0	
 A testifies	  T, 0 S, -3	  P, -2 P, -2	



Juegos de Suma Cero



Describe una situación en la que la ganancia o pérdida de un participante se equilibra con exactitud con las pérdidas o ganancias de los otros participantes.

Falacia de Suma Cero

Es el error de suponer que en cualquier situación de intercambio o competencia, una persona solo puede ganar si otra pierde

- No todos los sistemas son cerrados y de recursos fijos.
- Muchas situaciones permiten la cooperación y el crecimiento mutuo.

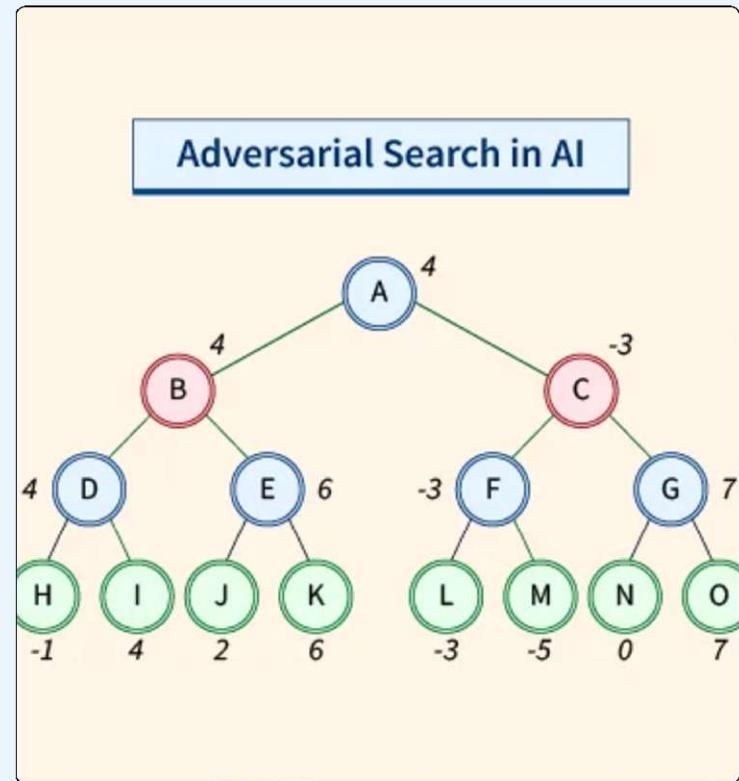
Problemas de Busqueda Adversaria

Características

- Determinísticos o estocásticos
- Uno, dos, n jugadores
- Visibilidad total o parcial
- Juegos que suman 0

Objetivo

Una estrategia (política) que recomiende un movimiento desde un estado



Problemas de Búsqueda Adversaria

Juegos Determinísticos

- N Estados
- 1 a N Jugadores (P)
- Acciones que dependen de P (jugador)
- Meta (Terminal/Goal)
- Utility Value (Valor de Utilidad)
- Terminal Utility (Utilidad/Valor final)

Juegos
Determinísticos



Problemas de Busqueda Adversaria

Suma Cero

- Utilidades opuestas
- Uno maximiza y el otro minimiza
- Adversario, competición pura



General Games

- Utilidades independientes
- Todos maximizan con su utilidad
- Cooperación, indiferencia, competición



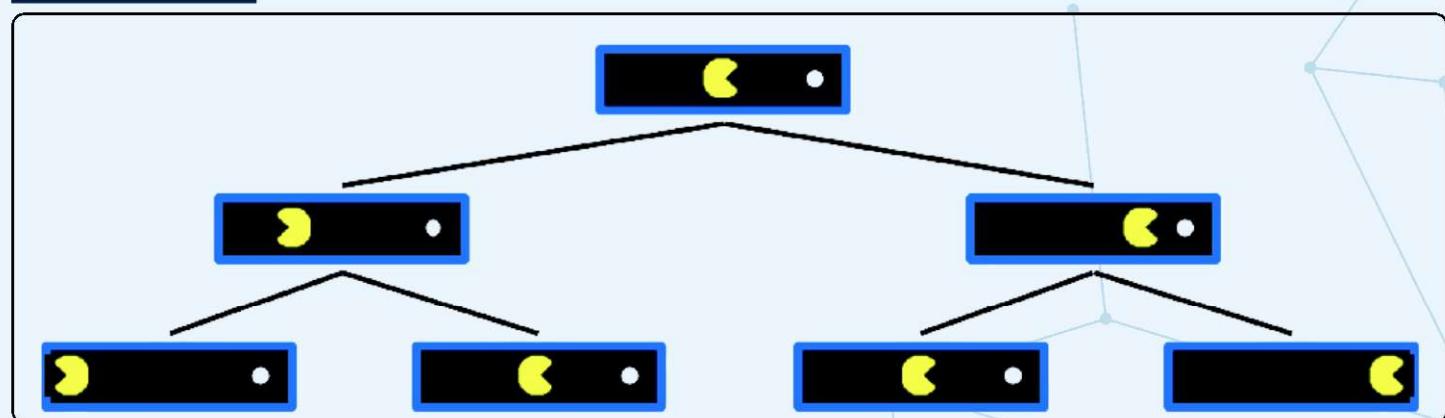
1 AGENTE (Agente que maximiza)

El escenario con un solo jugador (pacman), debe alcanzar la bolita (pellet)



Supongamos que Pacman comienza con 10 puntos y pierde 1 punto por movimiento hasta que se come la bolita

Game Tree

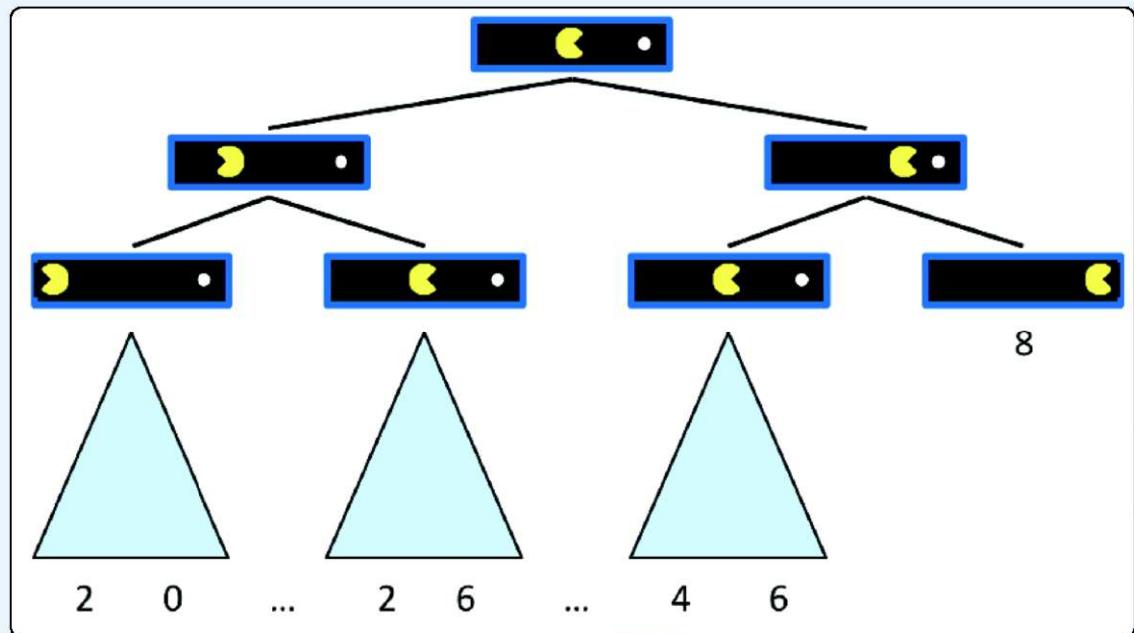


1 AGENTE

El escenario con un solo jugador (pacman), debe alcanzar la bolita (pellet)

Game Tree

Si Pacman va directo a la bolita, termina el juego con una puntuación de 8 puntos, mientras que si retrocede en cualquier punto, termina con una puntuación de menor valor



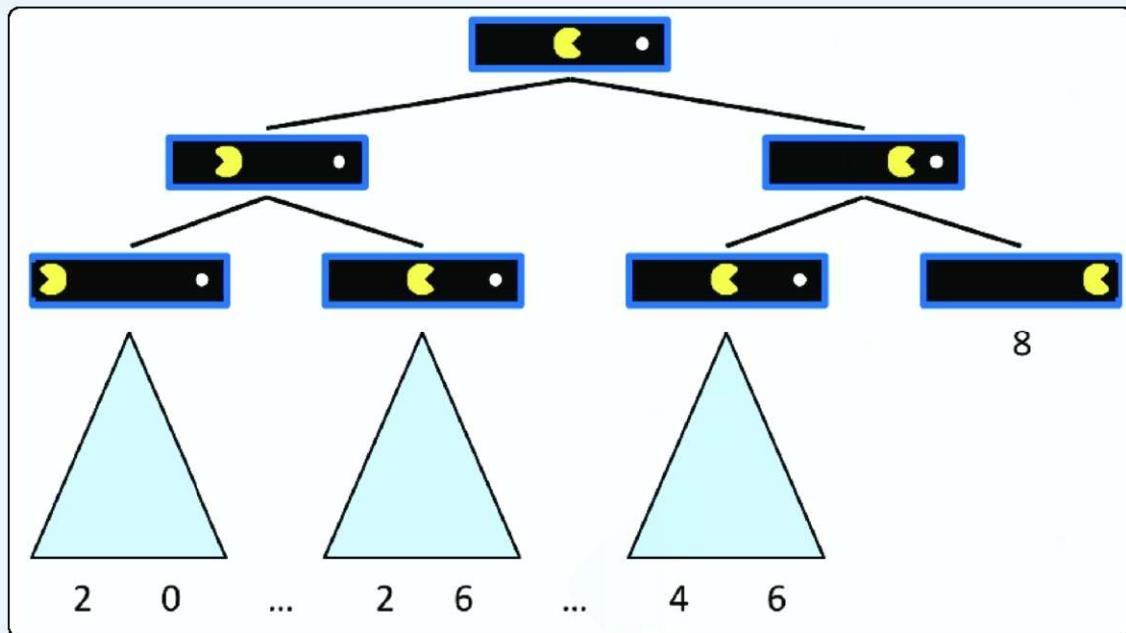
1 AGENTE

El escenario con un solo jugador (pacman), debe alcanzar la bolita (pellet)

Game Tree

El valor (V) de un estado (s) se define como el mejor resultado posible (utilidad) que un agente puede lograr a partir de ese estado.

Simplemente, piense en la utilidad de un agente como su puntuación o la cantidad de puntos que obtiene.



1 AGENTE

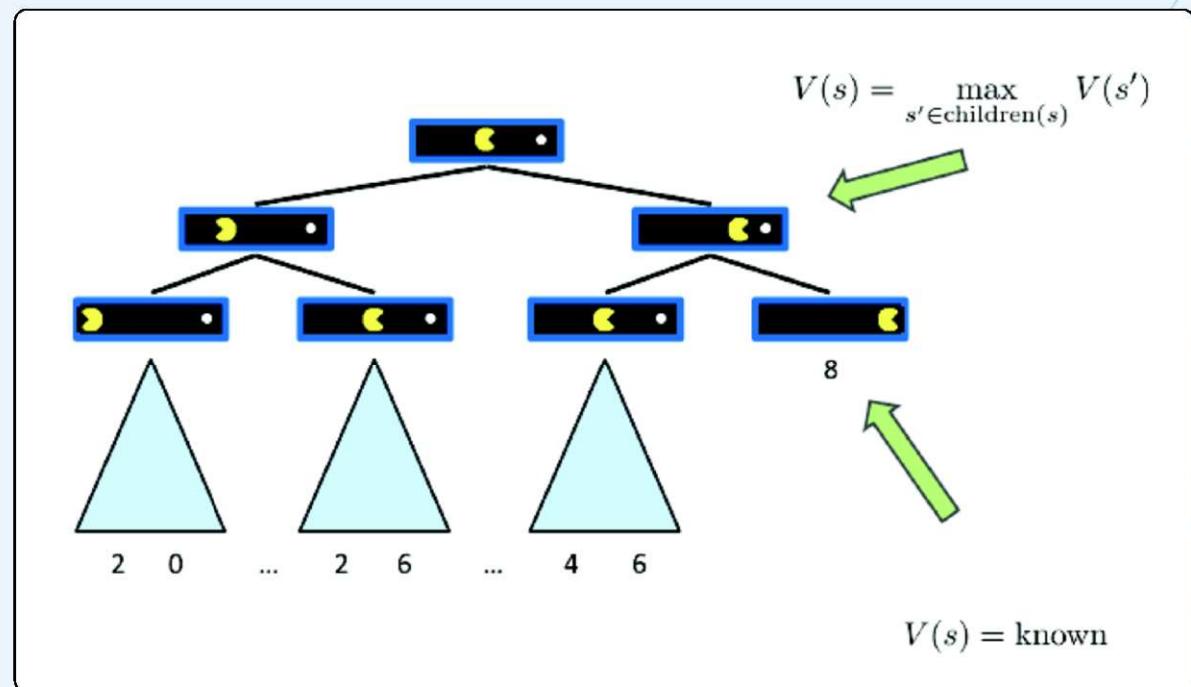
El escenario con un solo jugador (pacman), debe alcanzar la bolita (pellet)

Iremos buscando el que mayor utilidad nos representa entre los estados intermedios (s'). ($\max V(s')$)

El valor de un estado terminal $V(s)$, llamado utilidad terminal, es siempre un valor conocido determinista y una propiedad inherente del juego.

En nuestro ejemplo de Pacman, el valor del estado terminal más a la derecha es simplemente 8, la puntuación que Pacman obtiene al ir directo a la bolita.

Game Tree



Adversario (Agente que minimiza)

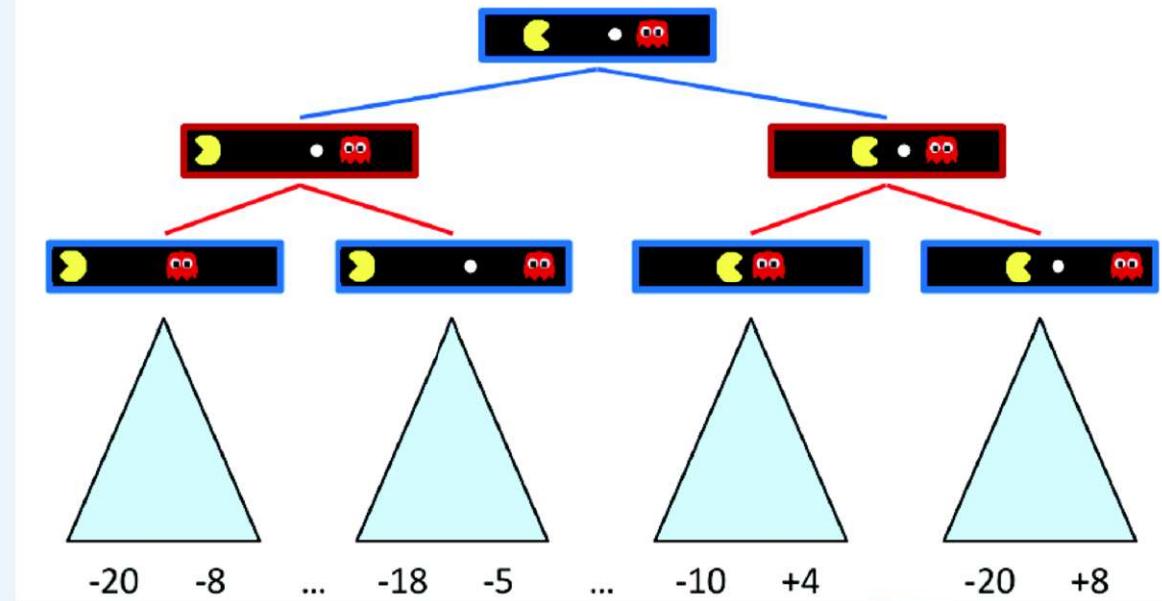
Agregamos un adversario



Introduzcamos un nuevo tablero de juego con un fantasma adversario que quiere evitar que Pacman se coma la bolita

Los dos agentes se turnan para realizar movimientos, lo que lleva a un árbol de juego donde los dos agentes activan y desactivan las capas del árbol que "controlan".

Game Tree



Estados

Pacman

Ghost

Adversario

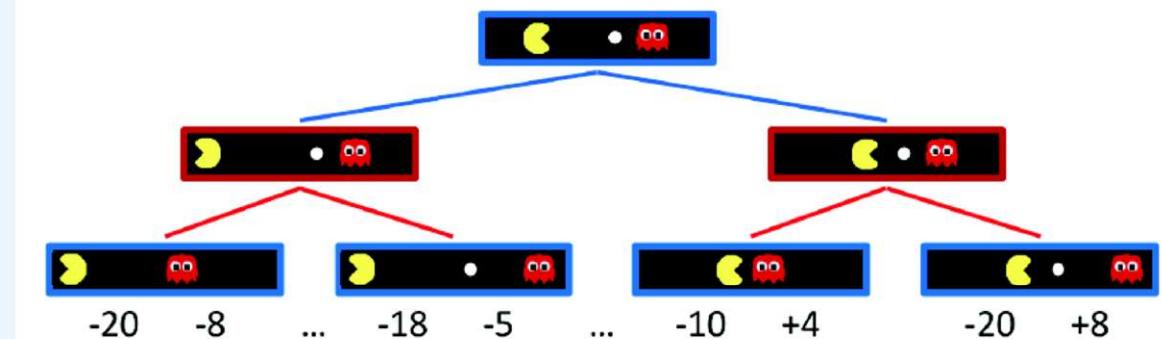
Agregamos un adversario



Para simplificar, truncaremos este árbol de juego a un árbol de profundidad 2 y asignaremos valores inventados a los estados terminales

Agregar nodos controlados por fantasmas cambia el movimiento que Pacman cree que es óptimo, y el nuevo movimiento óptimo se determina con el algoritmo minimax.

Game Tree



Estados

Pacman

Ghost

Adversario

Agregamos un adversario



En lugar de maximizar la utilidad sobre los hijos en cada nivel del árbol, el algoritmo minimax solo maximiza sobre los hijos de los nodos controlados por Pacman, mientras que minimiza sobre los hijos de los nodos controlados por fantasmas.

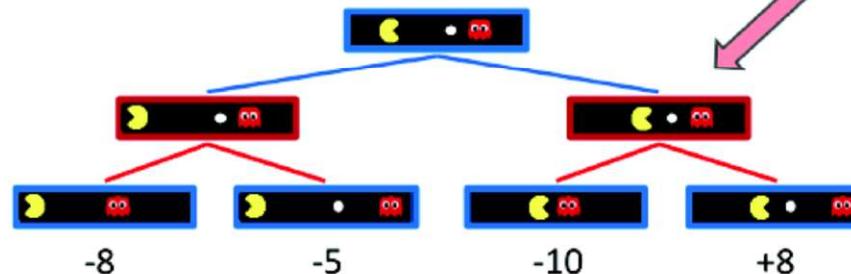
Game Tree

Agente

$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

Oponente

$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$



Terminal

$$V(s) = \text{known}$$

Estados

Pacman

Ghost



Decisiones óptimas en juegos: Minimax

Consideraremos juegos con dos jugadores, que llamaremos MAX y MIN.

- MAX mueve primero, y luego mueven por turno hasta que el juego se termina.
- Al final de juego, se conceden puntos al jugador ganador y penalizaciones al perdedor.

$$\text{VALOR-MINIMAX}(n) = \begin{cases} \text{UTILIDAD}(n) & \text{si } n \text{ es un estado terminal} \\ \max_{s \in \text{Sucesores}(n)} \text{VALOR-MINIMAX}(s) & \text{si } n \text{ es un estado MAX} \\ \min_{s \in \text{Sucesores}(n)} \text{VALOR-MINIMAX}(s) & \text{si } n \text{ es un estado MIN} \end{cases}$$

Decisiones óptimas en juegos: Minimax

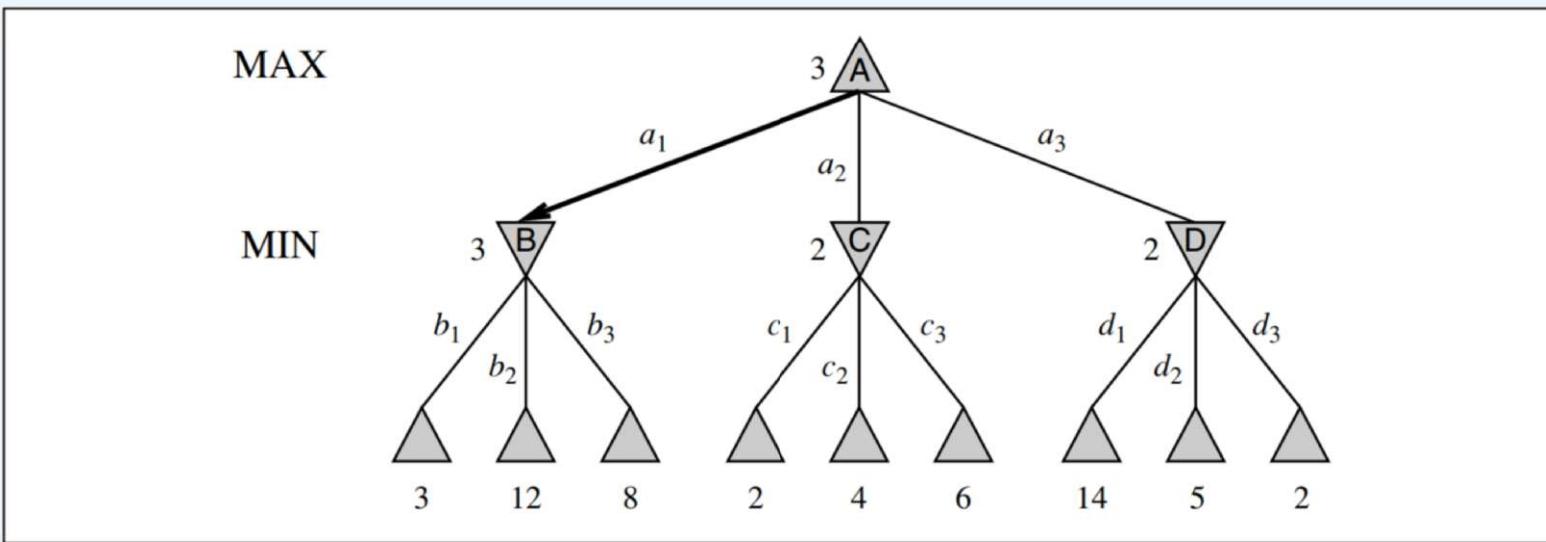


Figura 6.2 Un árbol de juegos de dos capas. Los nodos Δ son «nodos MAX», en los que le toca mover a MAX, y los nodos ∇ son «nodos MIN». Los nodos terminales muestran los valores de utilidad para MAX; los otros nodos son etiquetados por sus valores minimax. El mejor movimiento de MAX en la raíz es a_1 , porque conduce al sucesor con el valor minimax más alto, y la mejor respuesta de MIN es b_1 , porque conduce al sucesor con el valor minimax más bajo.

Decisiones óptimas en juegos: Minimax

Si estás implementando un agente de IA para un juego:

- *max-value* representaría al jugador que quiere ganar (MAX).
- *min-value* representaría al oponente que intenta bloquearlo (MIN).

Se usa este algoritmo para recorrer el árbol de posibles movimientos y decidir la mejor jugada.

```
def max-value(state):  
    initialize v = -∞  
    for each successor of state:  
        v = max(v, min-value(successor))  
    return v
```

$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

```
def min-value(state):  
    initialize v = +∞  
    for each successor of state:  
        v = min(v, max-value(successor))  
    return v
```

$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$



Minimax

```
def value(state):
```

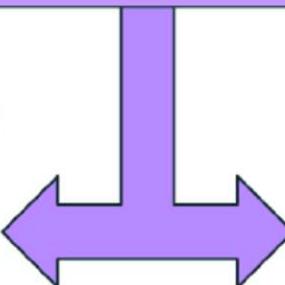
 if the state is a terminal state: return the state's utility
 if the next agent is MAX: return max-value(state)
 if the next agent is MIN: return min-value(state)

```
def max-value(state):
```

 initialize v = -∞
 for each successor of state:
 v = max(v, value(successor))
 return v

```
def min-value(state):
```

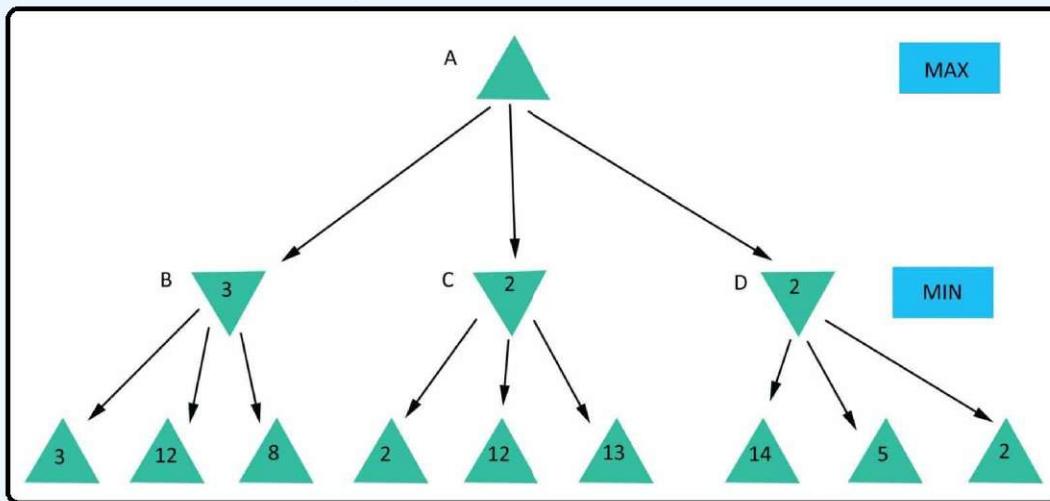
 initialize v = +∞
 for each successor of state:
 v = min(v, value(successor))
 return v



Minimax

El algoritmo Minimax es una técnica utilizada en juegos de dos jugadores con suma cero y turnos alternados, como el ajedrez o Pac-Man contra fantasmas.

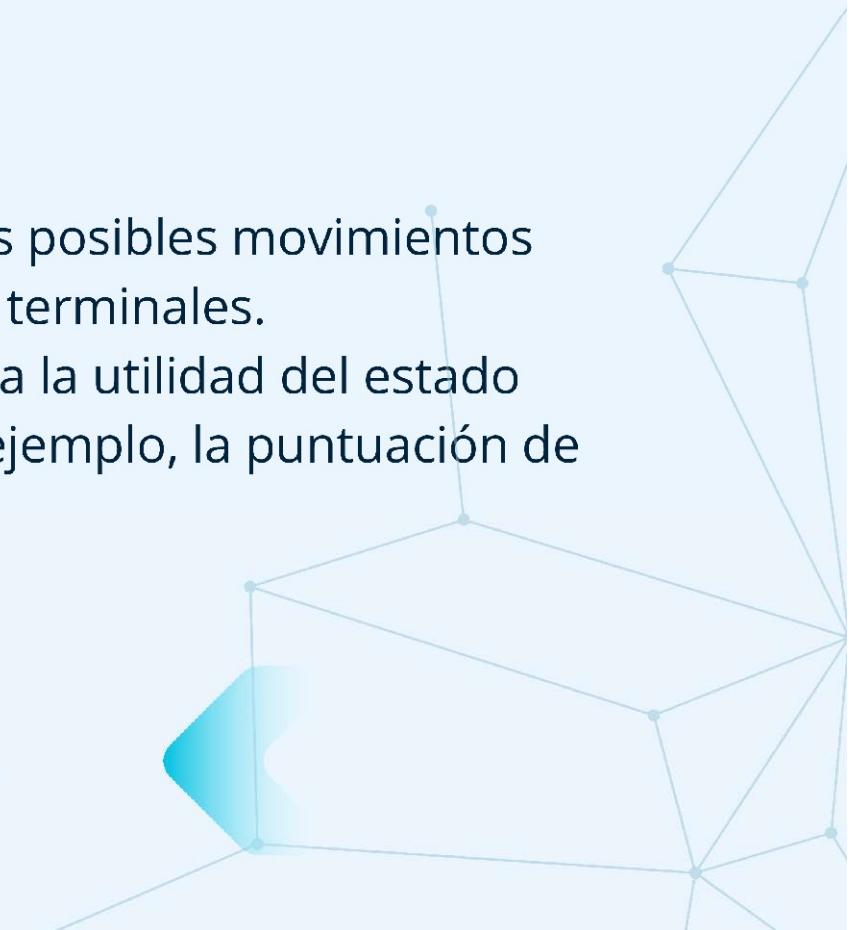
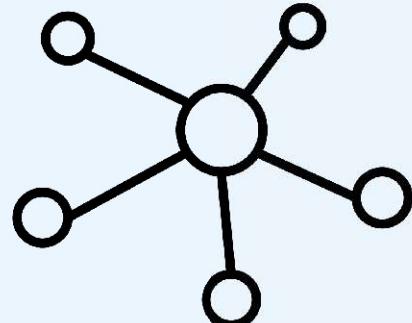
Su propósito es encontrar la mejor jugada para el jugador MAX (quien busca maximizar su puntuación) asumiendo que el oponente (MIN) jugará de la mejor forma posible para minimizar la puntuación de MAX.



Minimax

1. Exploración del árbol de juego:

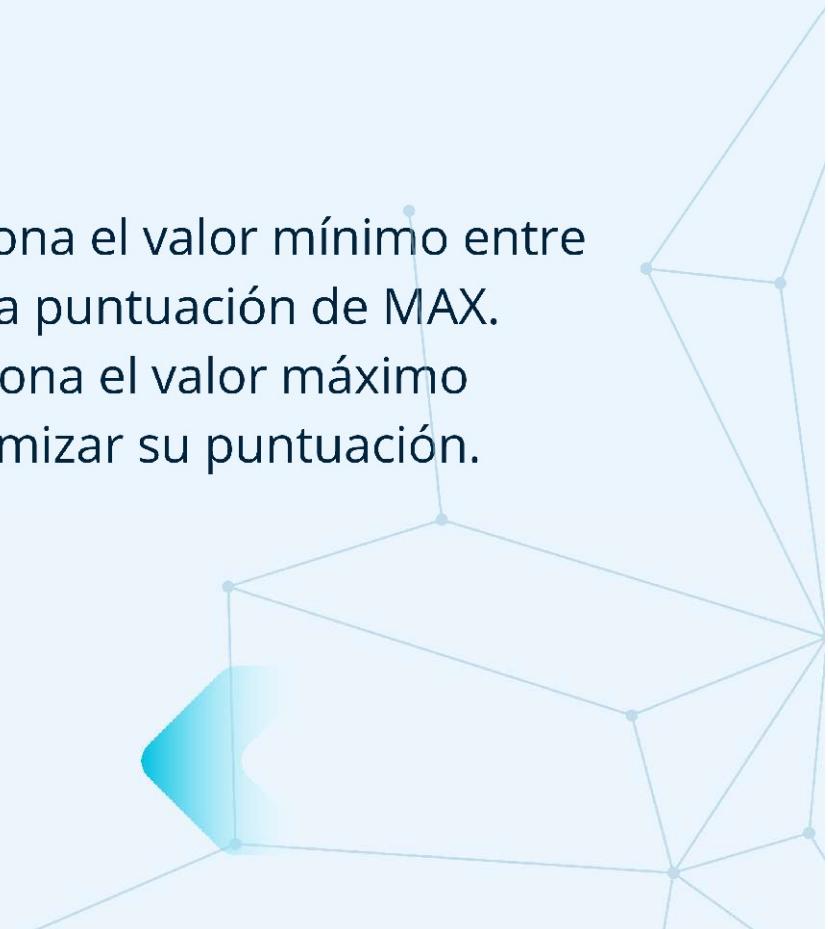
- Se genera el árbol de juego con todos los posibles movimientos desde el estado actual hasta los estados terminales.
- En los nodos terminales (hojas), se evalúa la utilidad del estado usando una función de evaluación (por ejemplo, la puntuación de Pac-Man).



Minimax

2. Propagación de valores hacia arriba:

- En los nodos donde juega MIN, se selecciona el valor mínimo entre sus hijos, porque MIN intenta minimizar la puntuación de MAX.
- En los nodos donde juega MAX, se selecciona el valor máximo entre sus hijos, porque MAX intenta maximizar su puntuación.



Minimax

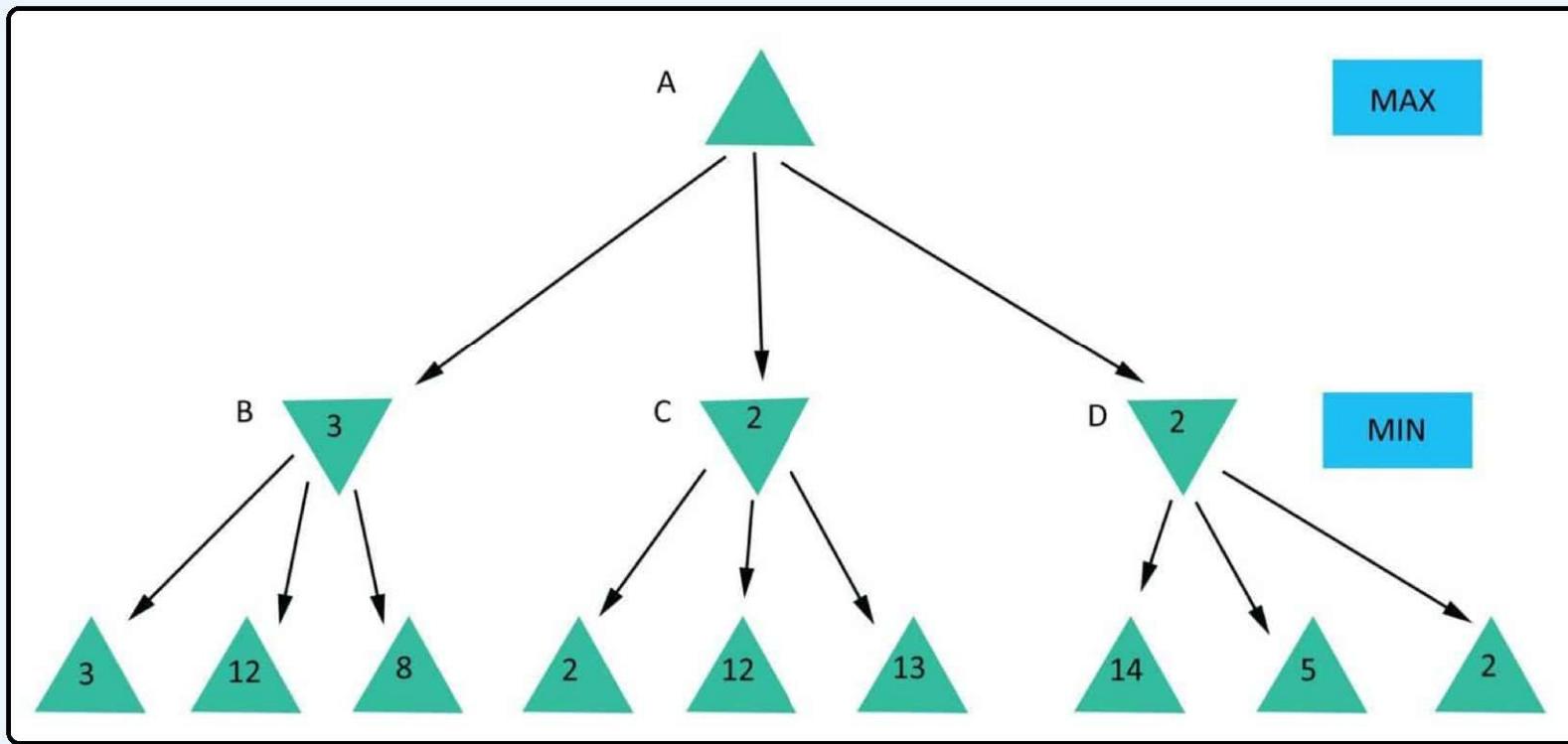
Elección de la mejor jugada:

- El jugador MAX elige la acción que lleva al estado con el mayor valor minimax.



Ejemplo del Funcionamiento

Si tenemos tres estados terminales con valores de utilidad 3, 12 y 8, MIN elegiría el mínimo (3) como la mejor opción en su turno. Luego, MAX evaluaría entre diferentes opciones y seleccionaría la de mayor valor como su mejor jugada.

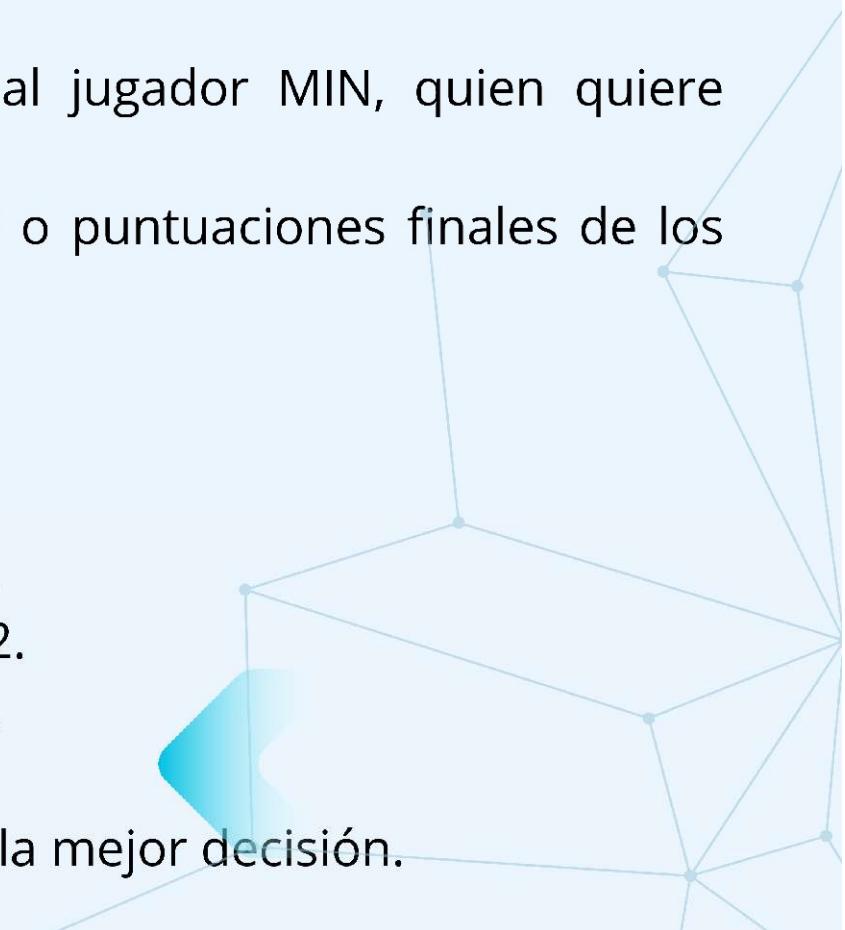


Niveles del árbol:

- Nivel 0 (Raíz - Nodo A) Representa al jugador MAX, quien quiere maximizar la utilidad.
- Nivel 1 (Nodos B, C y D) Representan al jugador MIN, quien quiere minimizar la utilidad.
- Nivel 2 (Hojas) Representan las utilidades o puntuaciones finales de los estados del juego.

Cálculo Minimax:

- MIN elige el valor mínimo de sus hijos:
 - Nodo B tiene hijos 3, 12 y 8 MIN elige 3.
 - Nodo C tiene hijos 2, 12 y 13 MIN elige 2.
 - Nodo D tiene hijos 14, 5 y 2 MIN elige 2.
- MAX elige el valor máximo de sus hijos:
 - Entre B (3), C (2) y D (2), MAX elige 3 como la mejor decisión.



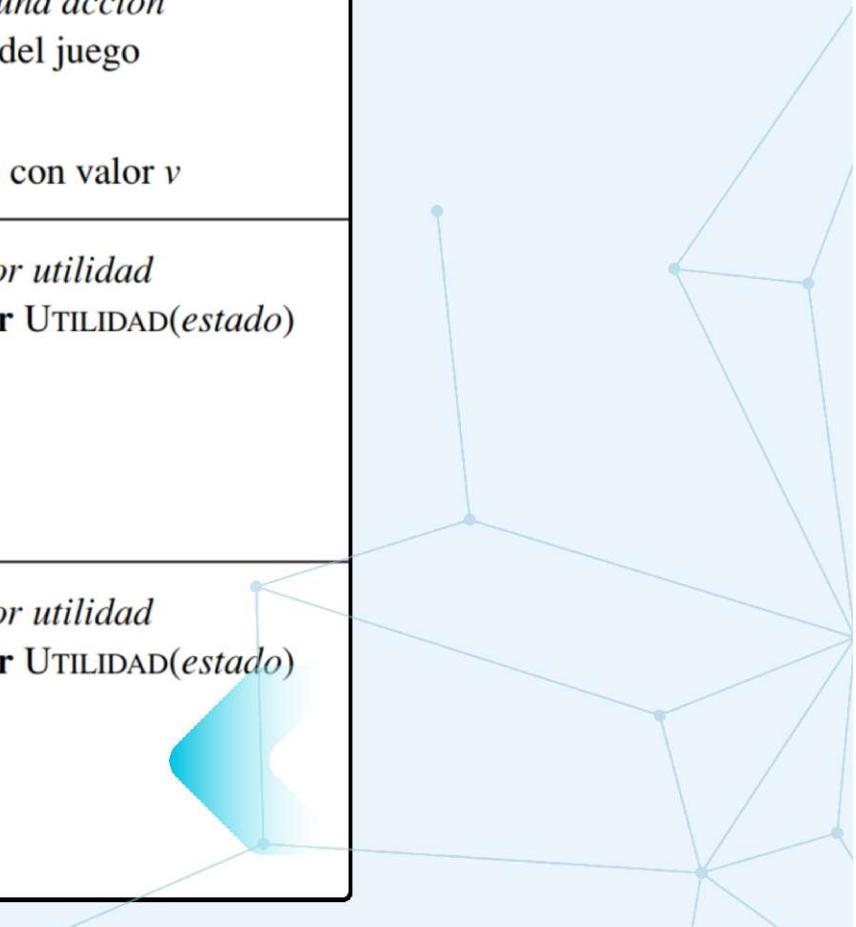
Minimax

función DECISIÓN-MINIMAX(*estado*) devuelve una acción
variables de entrada: *estado*, estado actual del juego

$v \leftarrow \text{MAX-VALOR}(\text{estado})$
devolver la acción de SUCESORES(*estado*) con valor v

función MAX-VALOR(*estado*) devuelve un valor utilidad
si TEST-TERMINAL(*estado*) **entonces devolver** UTILIDAD(*estado*)
 $v \leftarrow -\infty$
para un *s* en SUCESORES(*estado*) **hacer**
 $v \leftarrow \text{MAX}(v, \text{MIN-VALOR}(s))$
devolver v

función MIN-VALOR(*estado*) devuelve un valor utilidad
si TEST-TERMINAL(*estado*) **entonces devolver** UTILIDAD(*estado*)
 $v \leftarrow \infty$
para un *s* en SUCESORES(*estado*) **hacer**
 $v \leftarrow \text{MIN}(v, \text{MAX-VALOR}(s))$
devolver v



Complejidad Computacional

Tiempo:

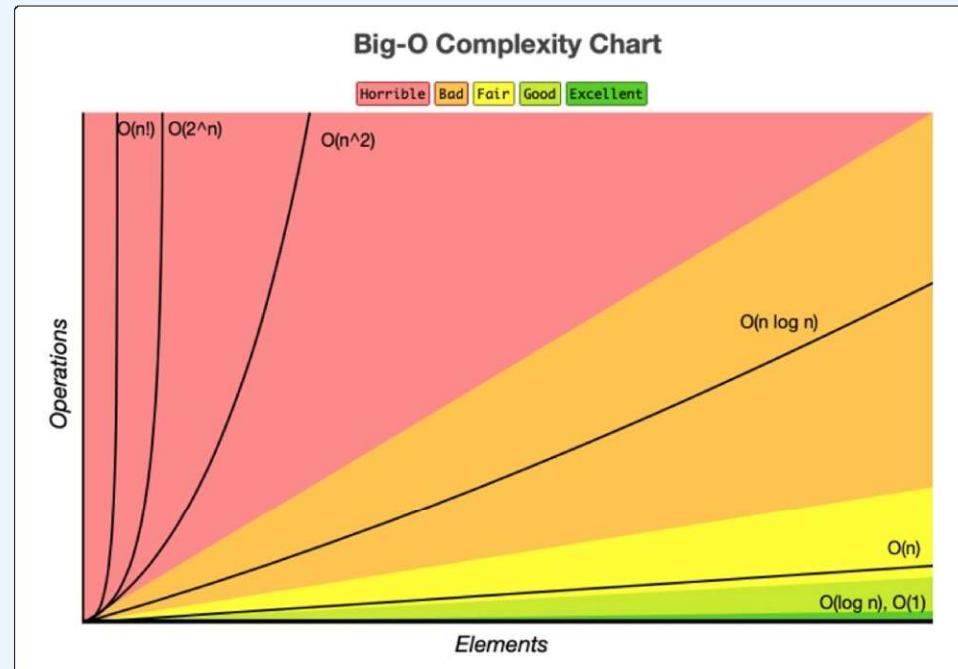
$O(b^m)$ donde:

- b es el número de movimientos posibles en cada turno (factor de ramificación).
- m es la profundidad máxima del árbol.

Es un crecimiento exponencial, lo que lo hace impráctico en juegos complejos como el ajedrez.

Espacio:

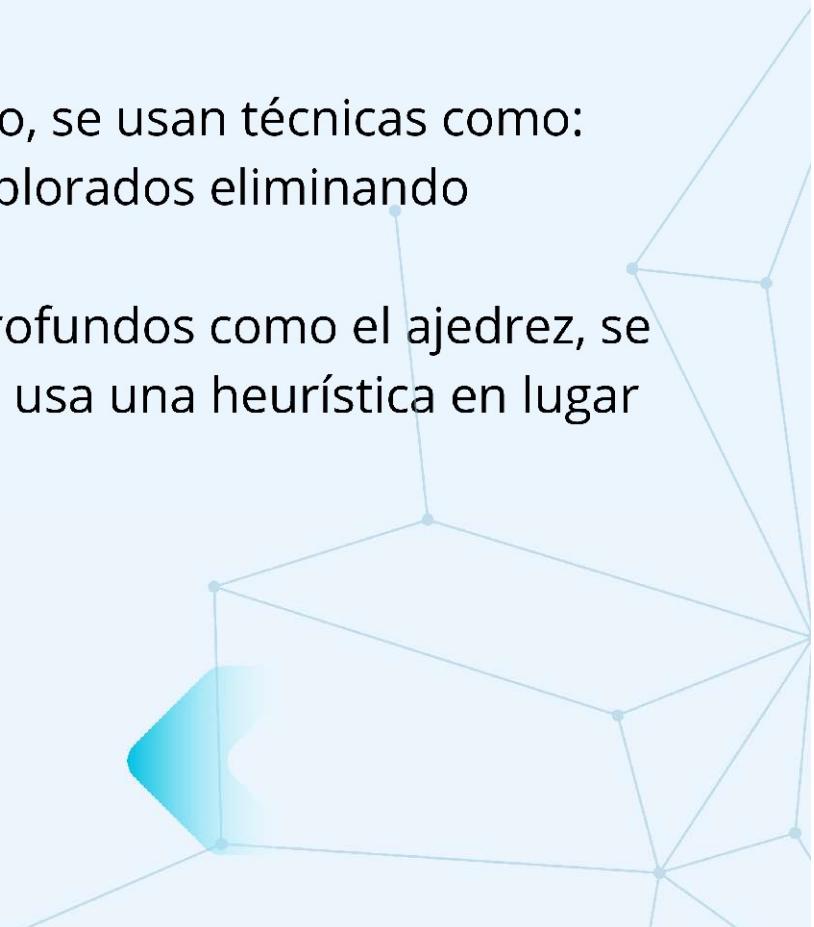
- $O(bm)$ si se generan todos los nodos a la vez.
- $O(m)$ si se usa una implementación recursiva con generación en tiempo real.



Optimización

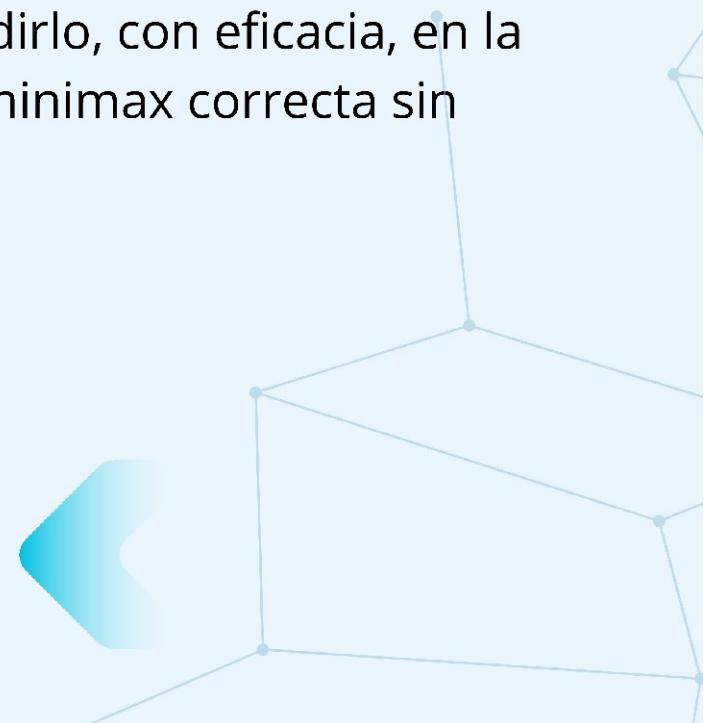
Dado que Minimax es computacionalmente costoso, se usan técnicas como:

- *Poda alfa-beta*: Reduce la cantidad de nodos explorados eliminando movimientos que no afectan la decisión final.
- *Funciones de evaluación heurísticas*: En juegos profundos como el ajedrez, se detiene la búsqueda en cierta profundidad y se usa una heurística en lugar de calcular hasta los estados finales.



Poda Alfa-Beta

El problema de la búsqueda minimax es que el número de estados que tiene que examinar es exponencial en el número de movimientos. Lamentablemente no podemos eliminar el exponente, pero podemos dividirlo, con eficacia, en la mitad. La jugada es que es posible calcular la decisión minimax correcta sin mirar todos los nodos en el árbol de juegos.



Poda Alfa-Beta

La poda alfa-beta puede aplicarse a árboles de cualquier profundidad, y, a menudo, es posible podar subárboles enteros.

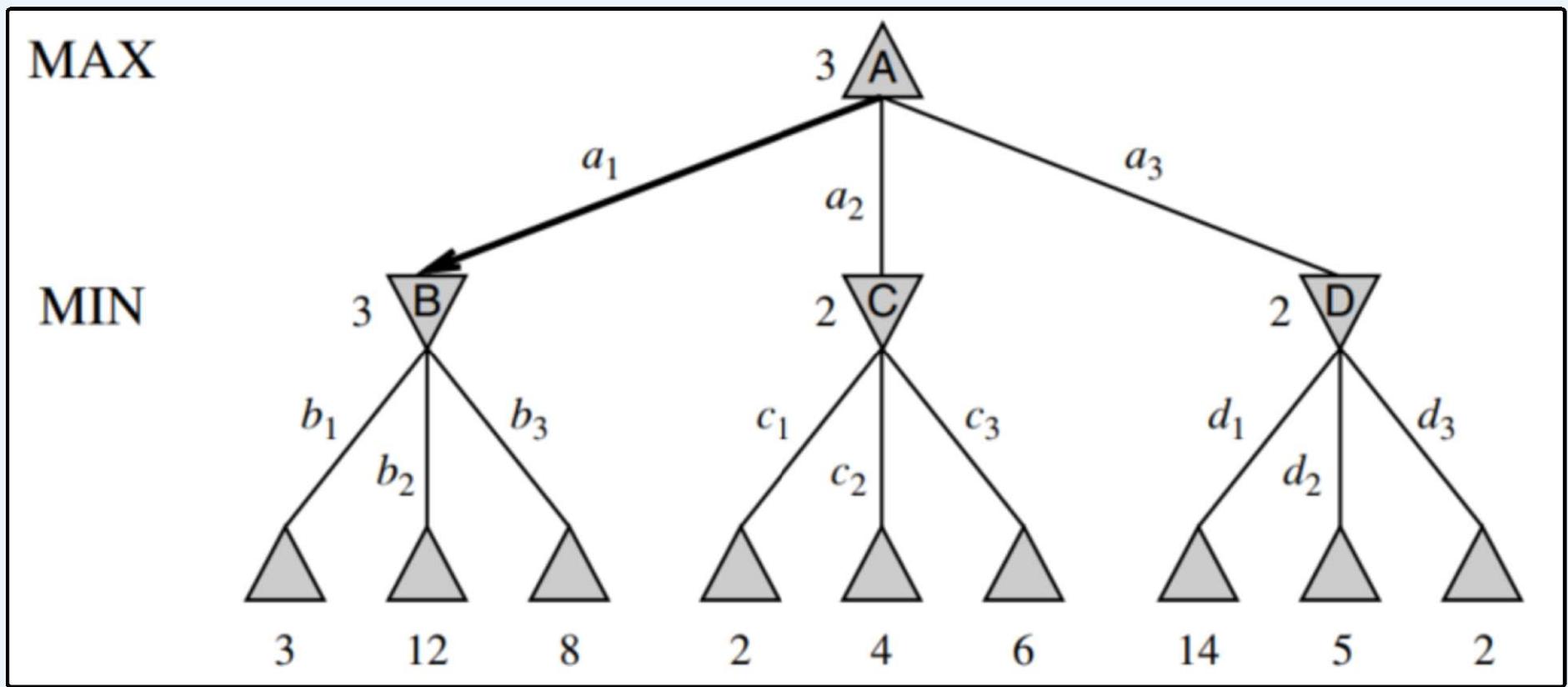


Valores Alfa y Beta

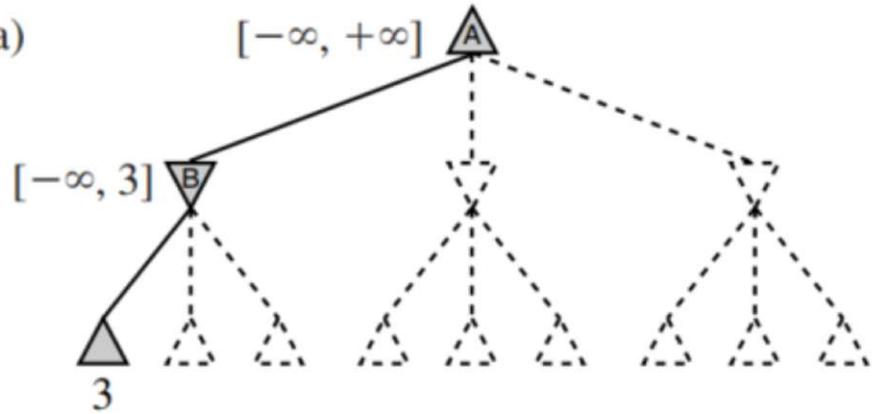
- α (*alfa*): Mejor valor garantizado para el jugador MAX en el camino actual.
- β (*beta*): Mejor valor garantizado para el jugador MIN en el camino actual.

Ejemplo de Poda Alfa-Beta

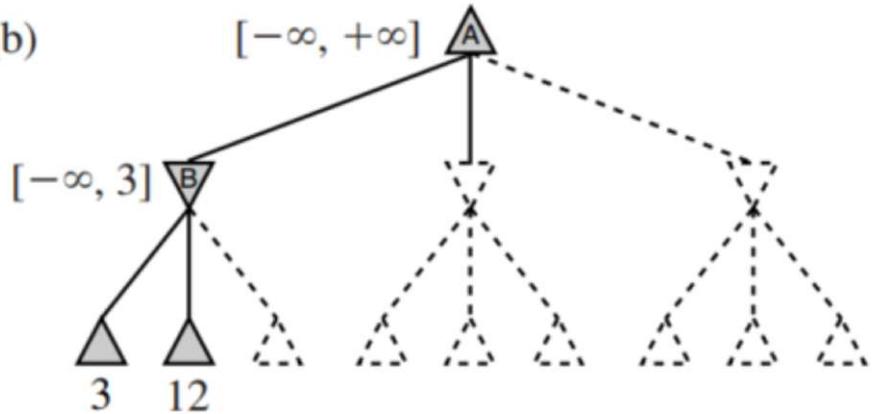
Los nodos terminales muestran los valores de utilidad para MAX; los otros nodos son etiquetados por sus valores minimax.



(a)



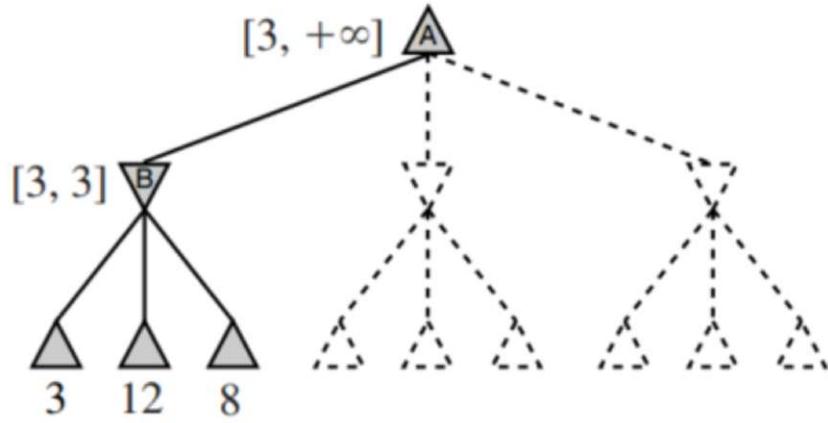
(b)



(a) La primera hoja debajo de B tiene el valor 3. De ahí B, que es un nodo MIN, tiene un valor de como máximo 3.

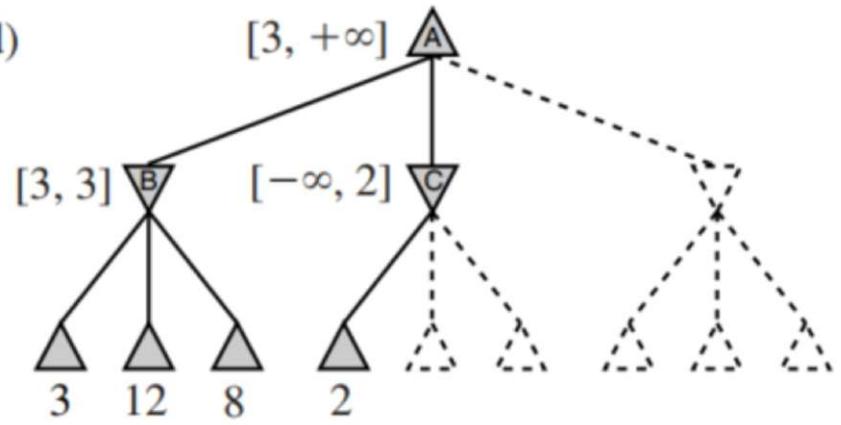
(b) La segunda hoja debajo de B tiene un valor de 12; MIN evitaría este movimiento, entonces el valor de B es todavía como máximo 3.

(c)



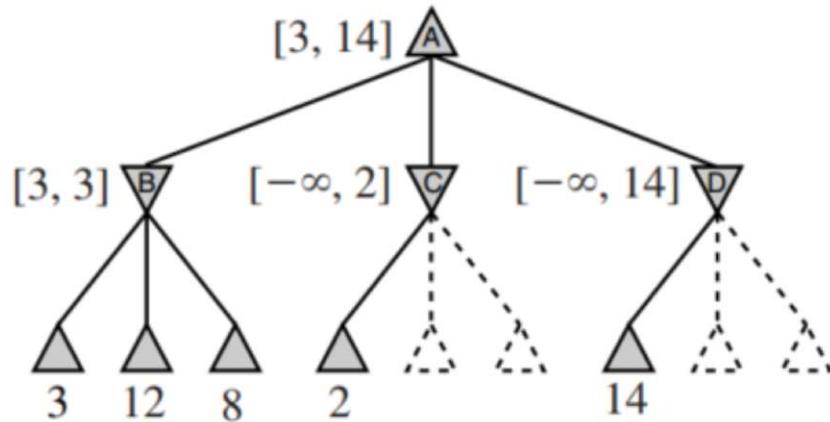
(c) La tercera hoja debajo de B tiene un valor de 8; hemos visto a todos los sucesores de B, entonces el valor de B es exactamente 3. Ahora, podemos deducir que el valor de la raíz es al menos 3, porque MAX tiene una opción digna de 3 en la raíz.

(d)

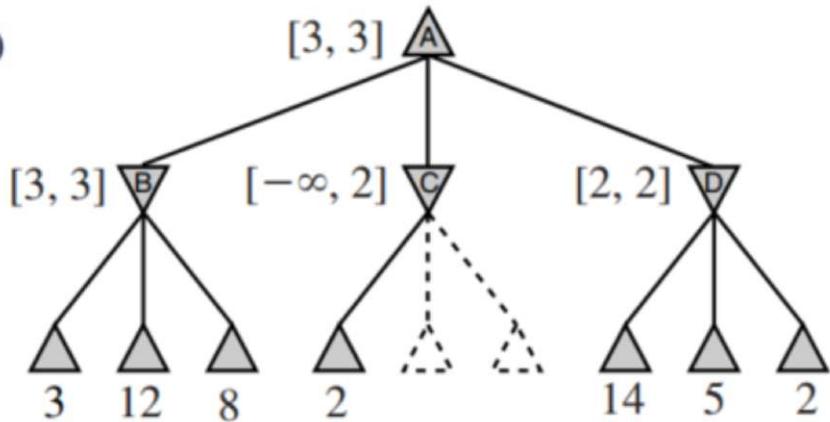


(d) La primera hoja debajo de C tiene el valor 2. De ahí, C, que es un nodo MIN, tiene un valor de como máximo 2. Pero sabemos que B vale 3, entonces MAX nunca elegiría C. Por lo tanto, no hay ninguna razón en mirar a los otros sucesores de C. Este es un ejemplo de la poda de alfa-beta.

(e)



(f)



(e) La primera hoja debajo de D tiene el valor 14, entonces D vale como máximo 14. Este es todavía más alto que la mejor alternativa de MAX (es decir, 3), entonces tenemos que seguir explorando a los sucesores de D. Note también que ahora tenemos límites sobre todos los sucesores de la raíz, entonces el valor de la raíz es también como máximo 14.

(f) El segundo sucesor de D vale 5, así que otra vez tenemos que seguir explorando. El tercer sucesor vale 2, así que ahora D vale exactamente 2. La decisión de MAX en la raíz es moverse a B, dando un valor de 3.

Simulador

- <https://pascsha.ch/info2/abTreePractice/>
- <http://homepage.ufp.pt/jtorres/ensino/ia/alfabeta.html>



Categoría	Aplicación	Descripción
Juegos de Mesa	Damas	Evaluá las posibles secuencias de movimientos para maximizar las ganancias y minimizar las pérdidas.
Videojuegos	Juegos de laberintos y rompecabezas	Encuentra rutas óptimas considerando posibles movimientos de oponentes.
Medicina	Diagnóstico basado en decisiones	Puede ayudar en la selección óptima de tratamientos considerando riesgos.

Ventajas

1. Encuentra la mejor jugada en un entorno determinista

- Minimax garantiza encontrar la estrategia óptima si el árbol de juego se puede evaluar completamente.

2. No necesita conocimientos probabilísticos

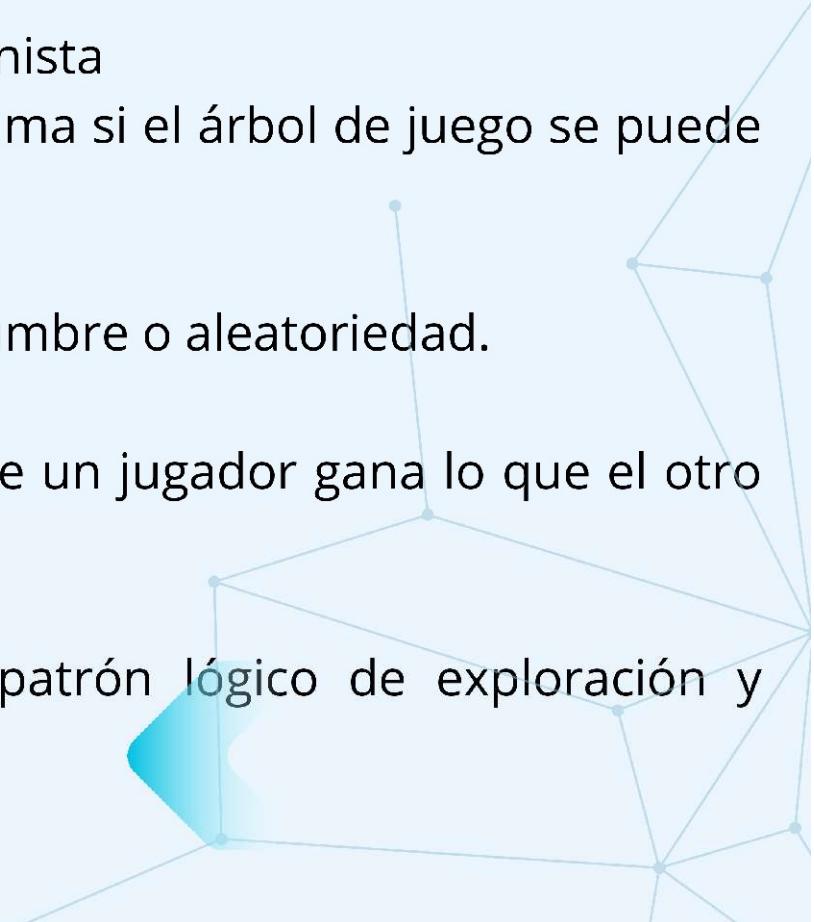
- Se basa en decisiones lógicas y no en incertidumbre o aleatoriedad.

3. Aplicable a juegos de suma cero

- Funciona bien en juegos de adversarios donde un jugador gana lo que el otro pierde (ajedrez, tic-tac-toe, damas, etc.).

4. Estructura clara y bien definida

- Su implementación es sencilla y sigue un patrón lógico de exploración y evaluación de estados.



Desventajas

1. Explosión combinatoria

- Para juegos con muchos movimientos posibles, el árbol crece exponencialmente, haciendo que el algoritmo sea ineficiente sin mejoras como poda alfa-beta.

2. Asume que el oponente juega de manera óptima

- No considera que el oponente pueda cometer errores o jugar de forma irracional.

3. No maneja incertidumbre o información oculta

- En juegos con elementos aleatorios (dados, cartas ocultas) o información incompleta, Minimax no es suficiente.

4. No se adapta a juegos muy dinámicos

- En juegos con tiempos de respuesta cortos, puede ser poco práctico por el tiempo de cálculo que requiere.

