Tema 3: Búsqueda heurística: métodos locales y búsqueda por el mejor nodo.

SIMPLE HEURISTICS THAT MAKE US SMART

Objetivos

- Adquirir las habilidades básicas para construir sistemas capaces de resolver problemas mediante técnicas de IA.
- Entender que la resolución de problemas en IA implica definir una representación del problema y un proceso de búsqueda de la solución.
- Analizar las características de un problema dado y determinar si es susceptible de ser resuelto mediante técnicas de búsqueda. Decidir en base a criterios racionales la técnica más apropiada para resolverlo y saber aplicarla.
- Entender el concepto de heurística y analizar las repercusiones en la eficiencia en tiempo y espacio de los algoritmos de búsqueda.
- Conocer las técnicas más representativas de búsqueda informada en un espacio de estados (búsqueda local, algoritmo A*).

Estudia el tema en …

- Nils J. Nilsson, "*Inteligencia Artificial: Una nueva síntesis",* Ed. Mc Graw Hill, 2000. pp. 53-62, 125-146, 163-174
- S. Russell, P. Norvig, "*Inteligencia Artificial: Un enfoque moderno",* Ed. Prentice Hall, 2ª edición, 2004.

Contenido

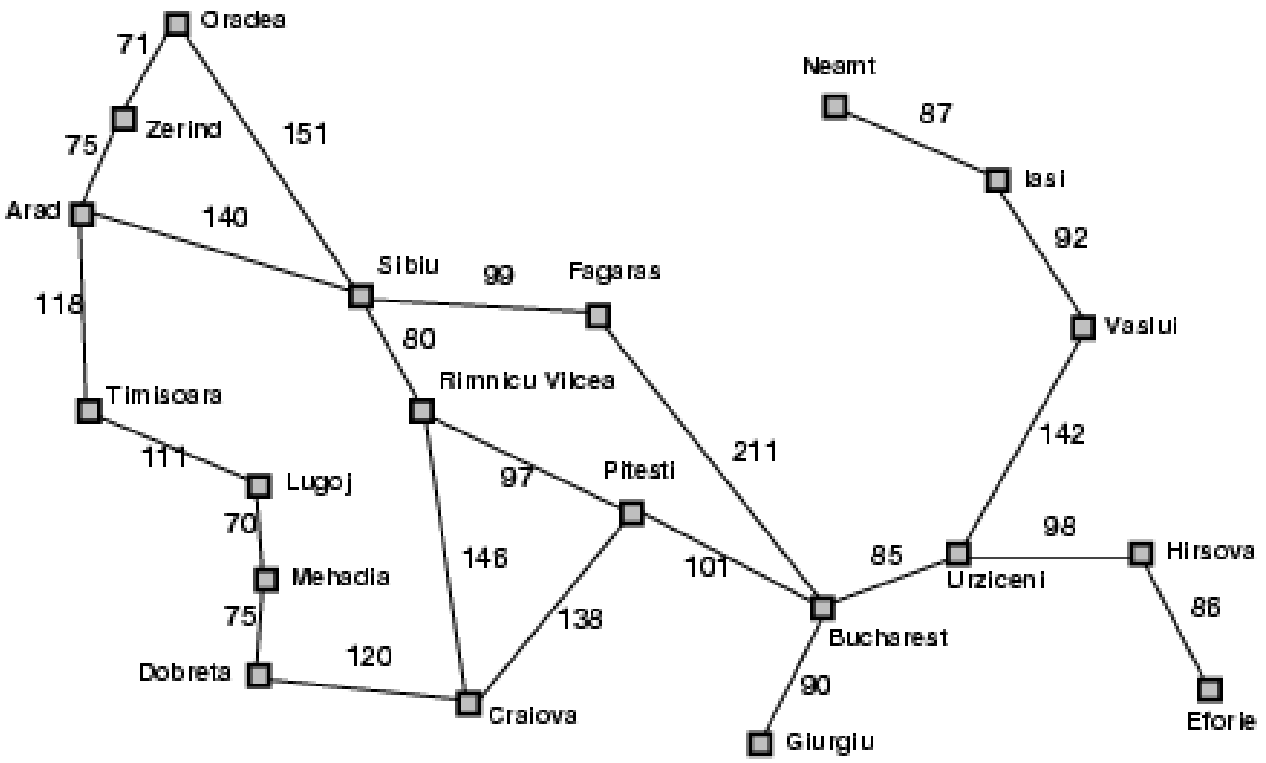
- Heurísticas
- Métodos de escalada
- Búsqueda primero el mejor

Heurísticas

- Las **heurísticas** son criterios, métodos o principios para decidir cuál de entre varias acciones promete ser la mejor para alcanzar una determinada meta
- En IA, entendemos por heurística un **método para resolver problemas** que en general **no garantiza la solución óptima**, pero que en media **produce resultados satisfactorios** en la resolución de un problema.
- Una heurística encapsula el conocimiento específico/experto que se tiene sobre un problema, y sirve de guía para que un algoritmo de búsqueda pueda encontrar una solución válida aceptable.
- Eventualmente, una heurística puede devolver siempre soluciones óptimas bajo ciertas condiciones (requiere demonstración).

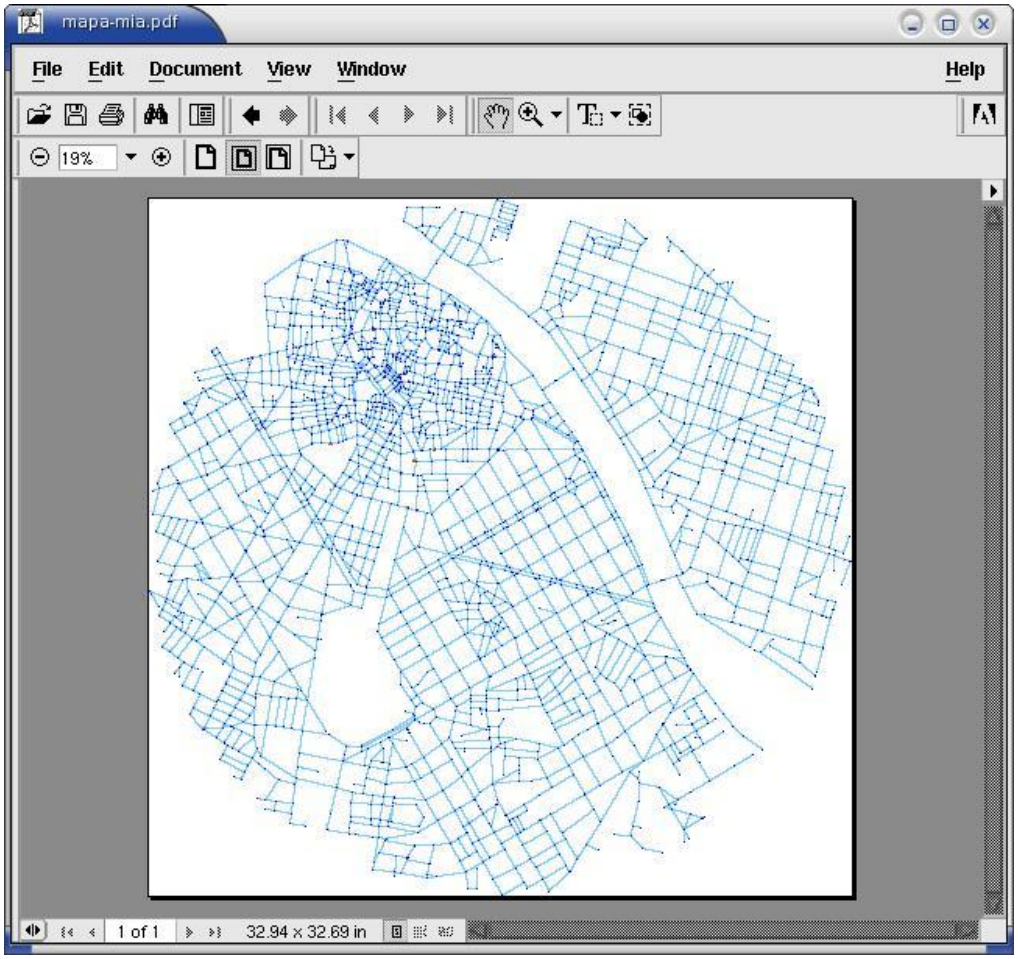
Ajedrez

Mapa de carreteras

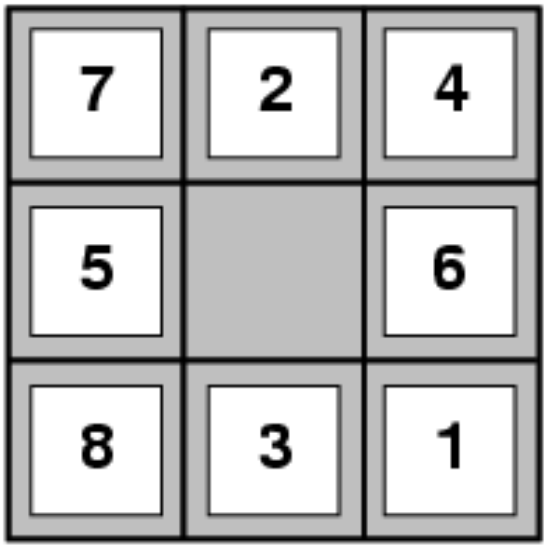


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

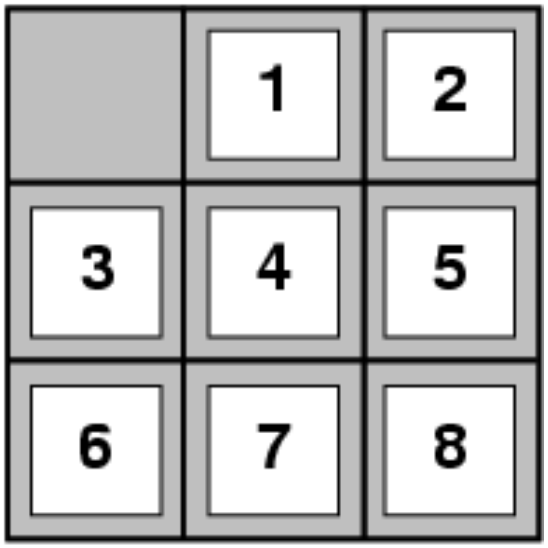
Mapa de carreteras



8-puzzle



Start State

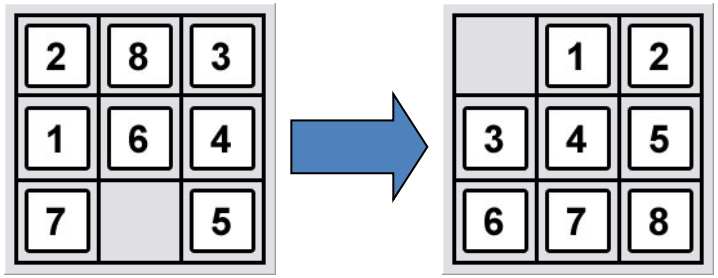


Goal State

8-puzzle

- **En IA, implementaremos heurísticas como funciones que devuelven un valor numérico**, cuya maximización o minimización guiará al proceso de búsqueda a la solución.
- **Ejemplo de heurística en el problema del 8-puzzle:**
  - **f(n)**= nº de fichas descolocadas en comparación con la posición objetivo a alcanzar.
  - **Objetivo:** Minimizar **f**.
  - **n** es un estado (posición de las piezas) del problema.
  - **f(n)** es la ***función heurística***.

f((2, 8, 3, 1, 6, 4, 7, 0, 5))= 9

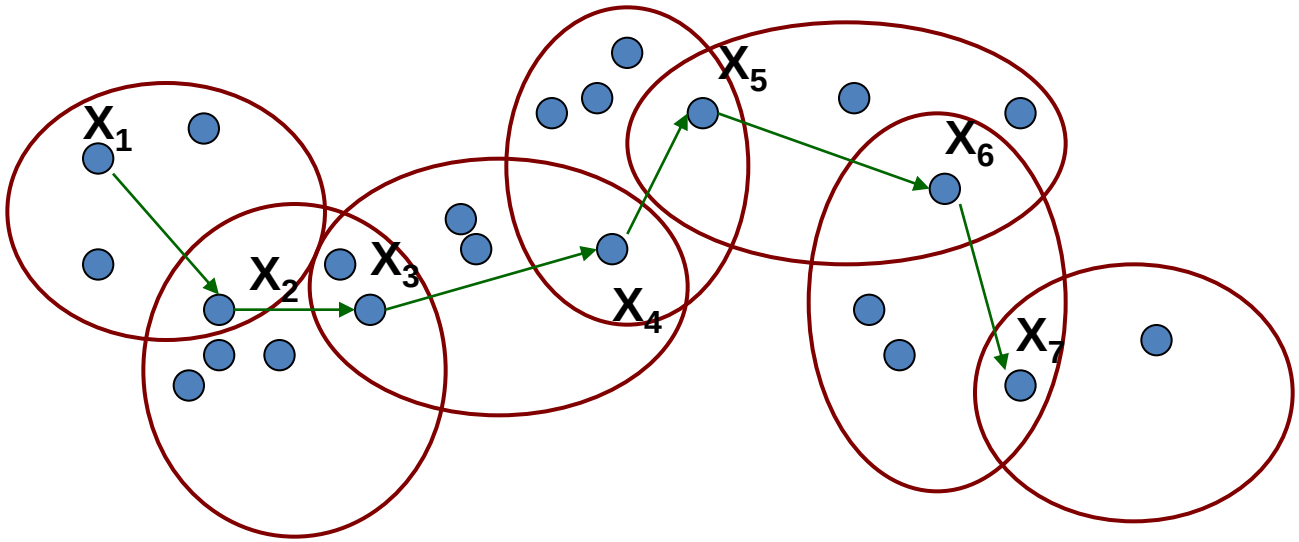


Métodos de escalada

- Algoritmo de escalada simple
- Algoritmo de escalada por la máxima pendiente
- Algunas variaciones estocásticas
- Algoritmos genéticos

Métodos de escalada

- Si dibujamos las soluciones como puntos en el espacio, una **búsqueda local** consiste en seleccionar la solución mejor en el vecindario de una solución inicial, e ir viajando por las soluciones del espacio hasta encontrar un óptimo (local o global).



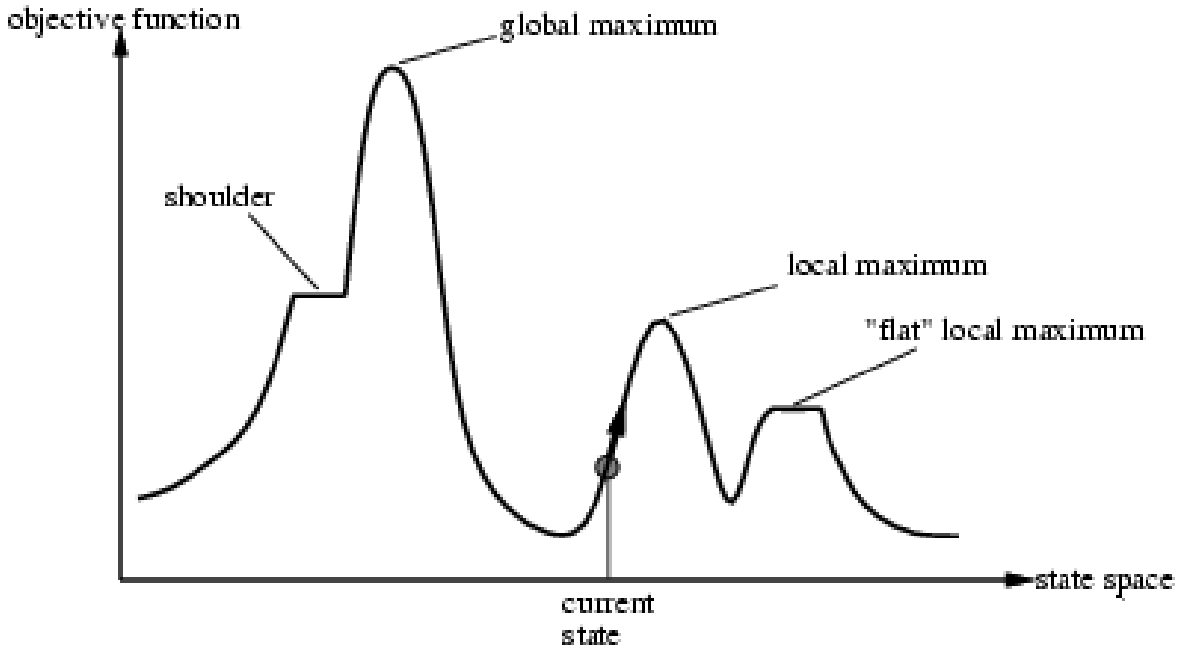
Algoritmo de escalada simple

- Evaluar el estado inicial. Si también es el estado objetivo, devolverlo y terminar. En caso contrario, continuar con el estado inicial como estado actual.
- Repetir hasta que se encuentre una solución o hasta que no queden nuevos operadores que aplicar al estado actual:
  - Seleccionar un operador que no haya sido aplicado con anterioridad al estado actual y aplicarlo para generar un nuevo estado.
  - Evaluar el nuevo estado.
    - Si es un estado objetivo, devolverlo y terminar.
    - Si no es un estado objetivo, pero es mejor que el estado actual, convertirlo en el estado actual.
    - Si no es mejor que el estado actual, continuar con el bucle.

Algoritmo de escalada por la máxima pendiente

- Evaluar el estado inicial. Si también es el estado objetivo, devolverlo y terminar. En caso contrario, continuar con el estado inicial como estado actual.
- Repetir hasta que se encuentre una solución o hasta que una iteración completa no produzca un cambio en el estado actual:
  - Sea SUCC un estado tal que algún posible sucesor del estado actual sea mejor que este SUCC.
  - Para cada operador aplicado al estado actual hacer lo siguiente:
    - Aplicar el operador y generar un nuevo estado
    - Evaluar el nuevo estado. Si es un estado objetivo, devolverlo y terminar. Si no, compararlo con SUCC. Si es mejor, asignar a SUCC este nuevo estado. Si no es mejor, dejar SUCC como está.
  - Si SUCC es mejor que el estado actual, hacer que el estado actual sea SUCC.

Métodos de escalada

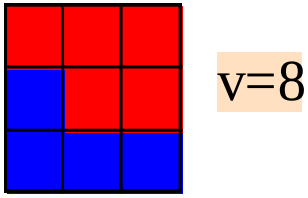


Ejemplo

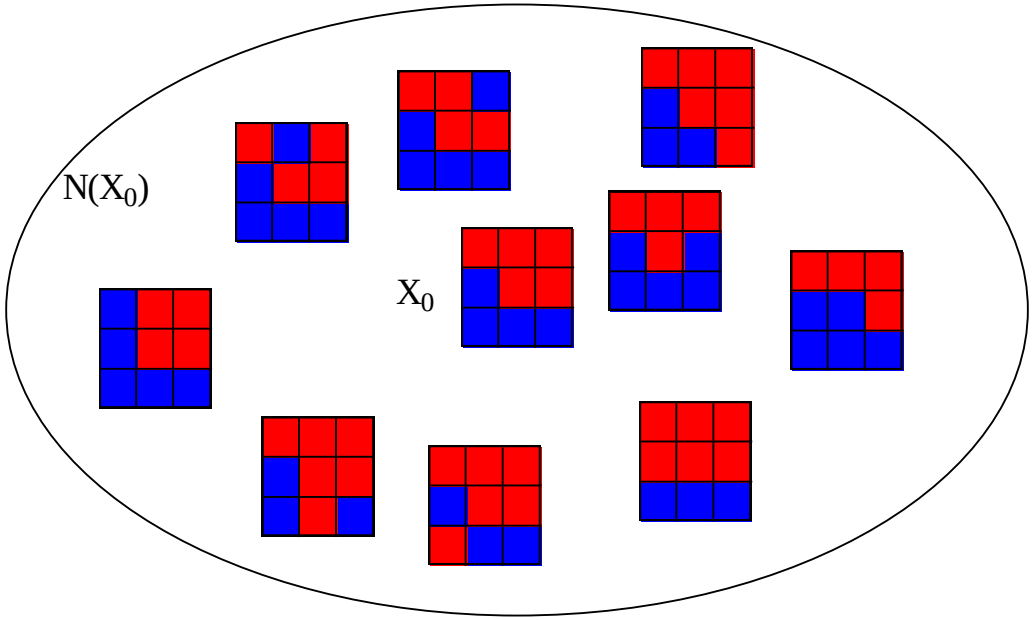
- **Ejemplo:** Colorear una matriz de **filas*columnas** con **n** colores, de modo que cada celda tenga el mínimo número de celdas adyacentes del mismo color. Asumimos que las celdas adyacentes son las que se encuentran una casilla hacia arriba, abajo, izquierda o derecha de la casilla considerada.
  - **Función objetivo:** Minimizar la suma del número de pares de casillas adyacentes del mismo color.
  - **Definición del entorno:** El vecindario de una solución estará formado por aquellas soluciones cuyos colores varíen en una única posición de la solución dada.

Ejemplo

- **Ejemplo:**
  - **Solución inicial:** Generada de forma aleatoria. Supongamos que se ha generado la siguiente para una matriz de **3*3**, a rellenar con **2 colores rojo y azul**, con valor de función objetivo ***v=8*:**

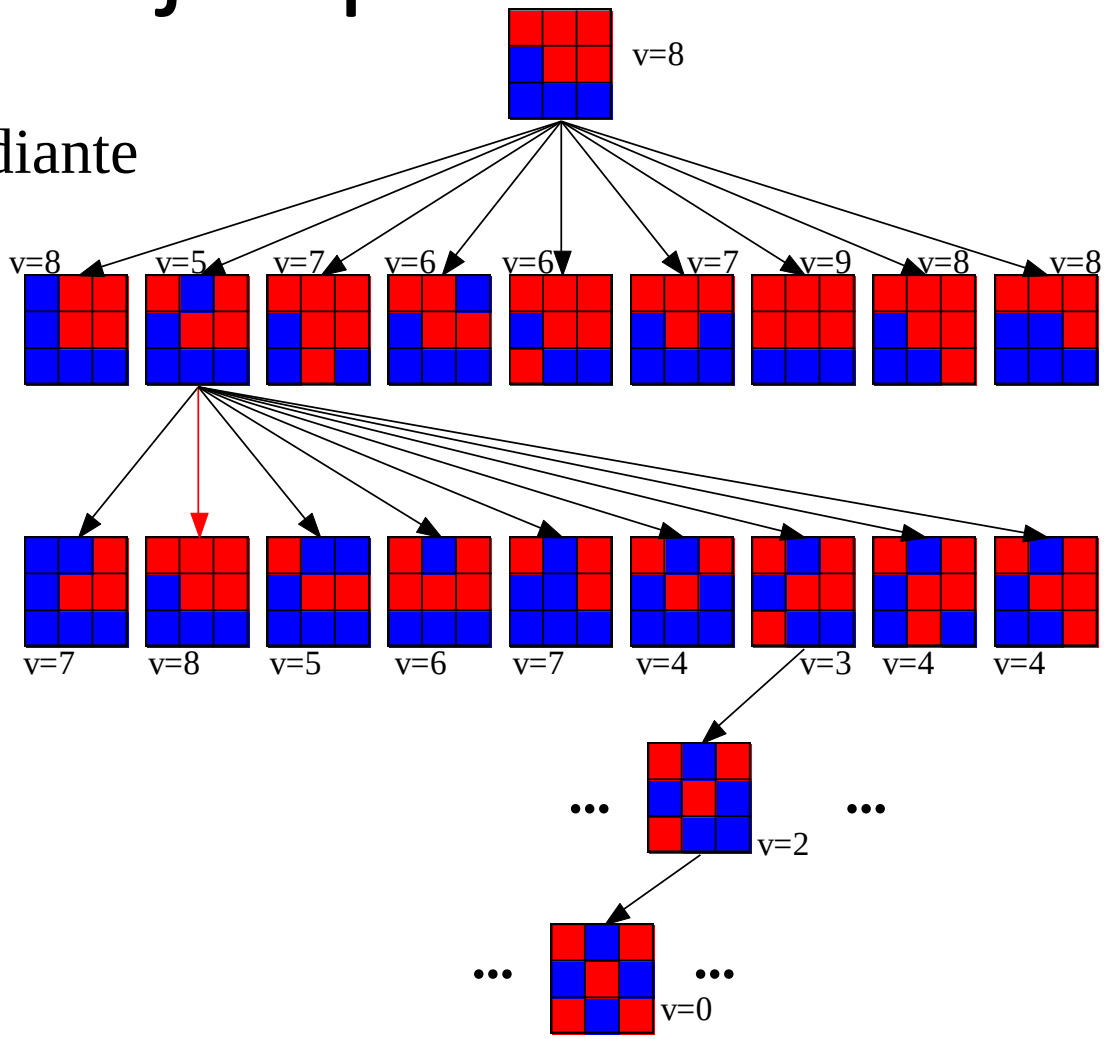


v=8



Ejemplo

- **Ejemplo:** Solución al problema anterior mediante ascensión de colinas.

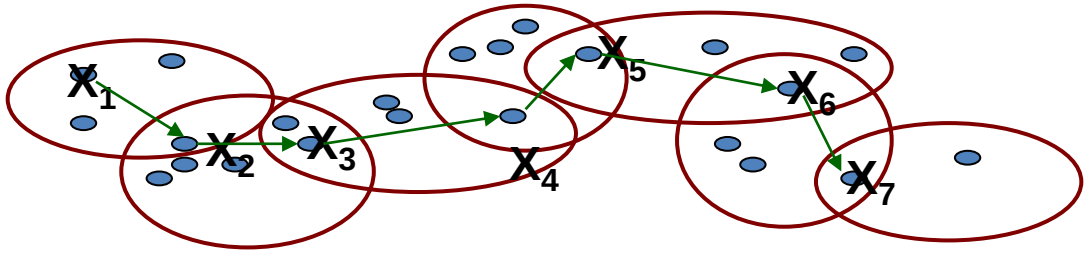


Algunas variaciones estocásticas

- Algoritmo de escalada estocástico
- Algoritmo de escalada de primera opción
- Algoritmo de escalada de reinicio aleatorio
- Enfriamiento simulado

Algoritmo de enfriamiento simulado

- Es un método de búsqueda local (por entornos).
- Su objetivo es encontrar una solución con valor de función objetivo óptimo.
- Genera los vecinos de forma probabilística (muy útil cuando el entorno de la solución es demasiado grande y costoso de generar).
- Se basa en principios de Termodinámica.
- Al contrario que los métodos de ascensión de colinas, permite visitar soluciones peores que la actual para evitar óptimos locales.



Algoritmo de enfriamiento simulado

- **Un poco de historia:**
  - En el campo de la Termodinámica, en los años 50 se simuló el proceso de enfriamiento en sistemas de partículas hasta que se llegaba a un estado congelado estable.
  - El proceso simulaba la diferencia de energía del sistema, **δE**, y se quería verificar que la probabilidad de que el sistema tuviese el cambio **δE** seguía la siguiente fórmula (**t** es la temperatura actual del sistema; **k** es una constante física):

$$P[\delta E] = e^{-\frac{\delta E}{k \cdot T}}$$

Algoritmo de enfriamiento simulado

- **Analogía entre el proceso de enfriamiento y el algoritmo de enfriamiento simulado:**
  - Los **estados** por los que pasa el sistema físico de partículas equivalen a las **soluciones factibles** del algoritmo.
  - La **energía E del estado actual** del sistema es el valor de la **función objetivo de la solución actual**. Ambos tienen que minimizarse.
  - Un **cambio de estado** en el sistema equivale a **explorar el entorno de una solución y viajar a una solución vecina**.
  - El **estado final estable** (congelado) es la **solución final** del algoritmo.

Algoritmo de enfriamiento simulado

Enfriamiento Simulado(T_i, T_f, α, N)

1. **T**← T_i, temperatura inicial
2. **X**← Generar Solución Inicial
3. Mientras (**T** >= T_f), hacer:
3.1. Para cada vecino a generar desde **i=1**..**N(T)**, hacer:
3.1.1. **X'**← Generar Solución Vecina de **X**
3.1.2. **F**← Bondad(**X'**) – Bondad(**X**)
3.1.3. Si **(F<0) ó (U(0,1)<e^(-F/T))**, aceptar solución **X**← **X'**
3.2. Actualizar temperatura T← α(T)
4. Devolver mejor **X** encontrado en el proceso.

Algoritmo de enfriamiento simulado

Enfriamiento Simulado(T_i, T_f, α, N)

1. **T**← T_i, temperatura inicial
2. **X**← Generar Solución Inicial
3. Mientras (**T** >= T_f), hacer:
3.1. Para cada vecino a generar desde **i=1**..**N(T)**, hacer:
3.1.1. **X'**← Generar Solución Vecina de **X**
3.1.2. **F**← Bondad(**X'**) – Bondad(**X**)
3.1.3. Si **(F<0) ó (U(0,1)<e^(-F/T))**, aceptar solución **X**← **X'**
3.2. Actualizar temperatura T← α(T)
4. Devolver mejor **X** encontrado en el proceso.

Se parte de una Temperatura inicial T_i, y de un estado (solución) **X**, con una energía (coste, bondad) asociada **Bondad(X)**

Algoritmo de enfriamiento simulado

Enfriamiento Simulado(T_i, T_f, α, N)

1. **T**← T_i, temperatura inicial
2. **X**← Generar Solución Inicial
3. Mientras (**T** >= T_f), hacer:
3.1. Para cada vecino a generar desde **i=1**..**N(T)**, hacer:
3.1.1. **X'**← Generar Solución Vecina de **X**
3.1.2. **F**← Bondad(**X'**) – Bondad(**X**)
3.1.3. Si **(F<0) ó (U(0,1)<e^(-F/T))**, aceptar solución **X**← **X'**
3.2. Actualizar temperatura T← α(T)
4. Devolver mejor **X** encontrado en el proceso.

El sistema se modelará hasta que se alcance una temperatura final deseada.

Algoritmo de enfriamiento simulado

Enfriamiento Simulado(T_i, T_f, α, N)

1. **T**← T_i, temperatura inicial
2. **X**← Generar Solución Inicial
3. Mientras (**T** >= T_f), hacer:
3.1. Para cada vecino a generar desde **i=1**..**N(T)**, hacer:
3.1.1. **X'**← Generar Solución Vecina de **X**
3.1.2. **F**← Bondad(**X'**) – Bondad(**X**)
3.1.3. Si **(F<0) ó (U(0,1)<e^(-F/T))**, aceptar solución **X**← **X'**
3.2. Actualizar temperatura T← α(T)
4. Devolver mejor **X** encontrado en el proceso.

En cada estado (solución), se exploran sus estados cercanos (vecindario). Si el problema es muy grande, la exploración se limita a **N(T)** vecinos.

Algoritmo de enfriamiento simulado

Enfriamiento Simulado(T_i, T_f, α, N)

1. **T**← T_i, temperatura inicial
2. **X**← Generar Solución Inicial
3. Mientras (**T** >= T_f), hacer:
3.1. Para cada vecino a generar desde **i=1**..**N(T)**
3.1.1. **X'**← Generar Solución Vecina de **X**
3.1.2. **F**← Bondad(**X'**) – Bondad(**X**)
3.1.3. Si **(F<0) ó (U(0,1)<e^(-F/T))**, aceptar solución **X**← **X'**
3.2. Actualizar temperatura T← α(T)
4. Devolver mejor **X** encontrado en el proceso.

Se genera un vecino. Si es mejor, se acepta directamente. Si no, y la diferencia de energía de los estados (Bondad de solución) es menor que e^(-F/T), **también se acepta**.

Algoritmo de enfriamiento simulado

Enfriamiento Simulado(T_i, T_f, α, N)

1. **T**← T_i, temperatura inicial
2. **X**← Generar Solución Inicial
3. Mientras (**T** >= T_f), hacer:
3.1. Para cada vecino a generar desde **i=1**..**N(T)**, hacer:
3.1.1. **X'**← Generar Solución Vecina de **X**
3.1.2. **F**← Bondad(**X'**) – Bondad(**X**)
3.1.3. Si **(F<0) ó (U(0,1)<e^(-F/T))**, aceptar solución **X**← **X'**
3.2. Actualizar temperatura T← α(T)
4. Devolver mejor **X** encontrado en el proceso.

La temperatura se reduce, indicando que nos acercamos al estado estable.

Algoritmo de enfriamiento simulado

Enfriamiento Simulado(T_i, T_f, α, N)

1. **T**← T_i, temperatura inicial
2. **X**← Generar Solución Inicial
3. Mientras (**T** >= T_f), hacer:
3.1. Para cada vecino a generar desde **i=1**..**N(T)**, hacer:
3.1.1. **X'**← Generar Solución Vecina de **X**
3.1.2. **F**← Bondad(**X'**) – Bondad(**X**)
3.1.3. Si **(F<0) ó (U(0,1)<e^(-F/T))**, aceptar solución **X**← **X'**
3.2. Actualizar temperatura T← α(T)
4. Devolver mejor **X** encontrado en el proceso.

Se devuelve la mejor solución X que se haya encontrado en todo el proceso de enfriamiento simulado

Algoritmo de enfriamiento simulado

- La **solución inicial** se puede generar de forma aleatoria, por conocimiento experto, o por medio de otras técnicas algorítmicas como *greedy*.
- La **actualización de temperatura** también es heurística, y hay varios métodos:
  - $T \leftarrow \alpha \cdot T$, con α en (0,1)
  - $T \leftarrow 1/(1+k)$, con k= número de iteraciones del algoritmo hasta el momento, etc.
- **Número de vecinos a generar**: Fijo N(T)= cte, dependiente de la temperatura N(T)= f(T), etc.

Algoritmo de enfriamiento simulado

- Tanto la temperatura inicial como la temperatura final $\mathbf{T_i}$ y $\mathbf{T_f}$ son parámetros de entrada al algoritmo.
- Es difícil asignar un valor concreto a $\mathbf{T_f}$, por lo que la condición de parada se suele sustituir por un número específico de iteraciones a realizar.

- **Ventajas:**
  - Al ser un método probabilístico, tiene capacidad para salir de óptimos locales.
  - Es eficiente.
  - Es fácil de implementar.

Algoritmo de enfriamiento simulado

- **Inconvenientes:**
  - Encontrar la temperatura inicial $\mathbf{T_i}$, el método de actualización de temperatura $\boldsymbol{\alpha}$, el número de vecinos a generar en cada estado y el número de iteraciones óptimo es una tarea que requiere de **muchas pruebas de ensayo y error** hasta que ajustamos los parámetros óptimos.

- Pese a todo, el algoritmo puede proporcionar soluciones mucho mejores que utilizando algoritmos no probabilísticos.

Algoritmos genéticos

- La simulación de procesos naturales es un campo de investigación muy amplio en Inteligencia Artificial, conocido por varios nombres como **computación evolutiva**, **biocomputación**, **algoritmos bioinspirados**, etc.
- Si ha funcionado bien en la naturaleza, ¿porqué una simulación de estos procesos no iba a proporcionar buenos resultados en un computador?
- Ejemplos:
  - Algoritmos genéticos.
  - Algoritmos basados en Colonias de Hormigas.
  - Algoritmos basados en inteligencia de enjambres.

Algoritmos genéticos

- Son algoritmos de optimización basados en el proceso de la evolución natural de Darwin.
- En un proceso de evolución, existe una **población de individuos**. Los más adecuados a su entorno **se reproducen y tienen descendencia** (a veces **con mutaciones** que mejoran su idoneidad al entorno). Los más adecuados sobreviven para la siguiente generación.
- No necesitan partir de un nodo/estado inicial: ¡Hay toda una población!

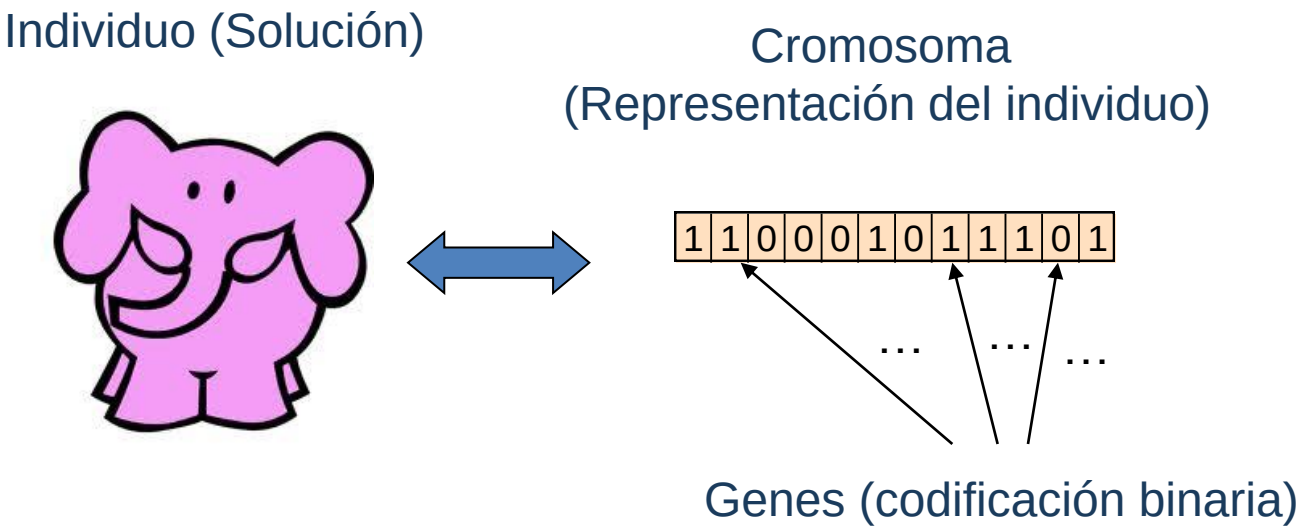
- Su objetivo es encontrar una solución cuyo valor de función objetivo sea óptimo.

Algoritmos genéticos

- **Cromosoma** ←→ Vector representación de una solución al problema.
- **Gen** ←→ Característica/Variable/Atributo concreto del vector de representación de una solución
- **Población** ←→ Conjunto de soluciones al problema.
- **Adecuación al entorno** ←→ Valor de función objetivo (fitness).
- **Selección natural** ←→ Operador de selección.
- **Reproducción sexual** ←→ Operador de cruce.
- **Mutación** ←→ Operador de mutación.
- **Cambio generacional** ←→ Operador de reemplazamiento.

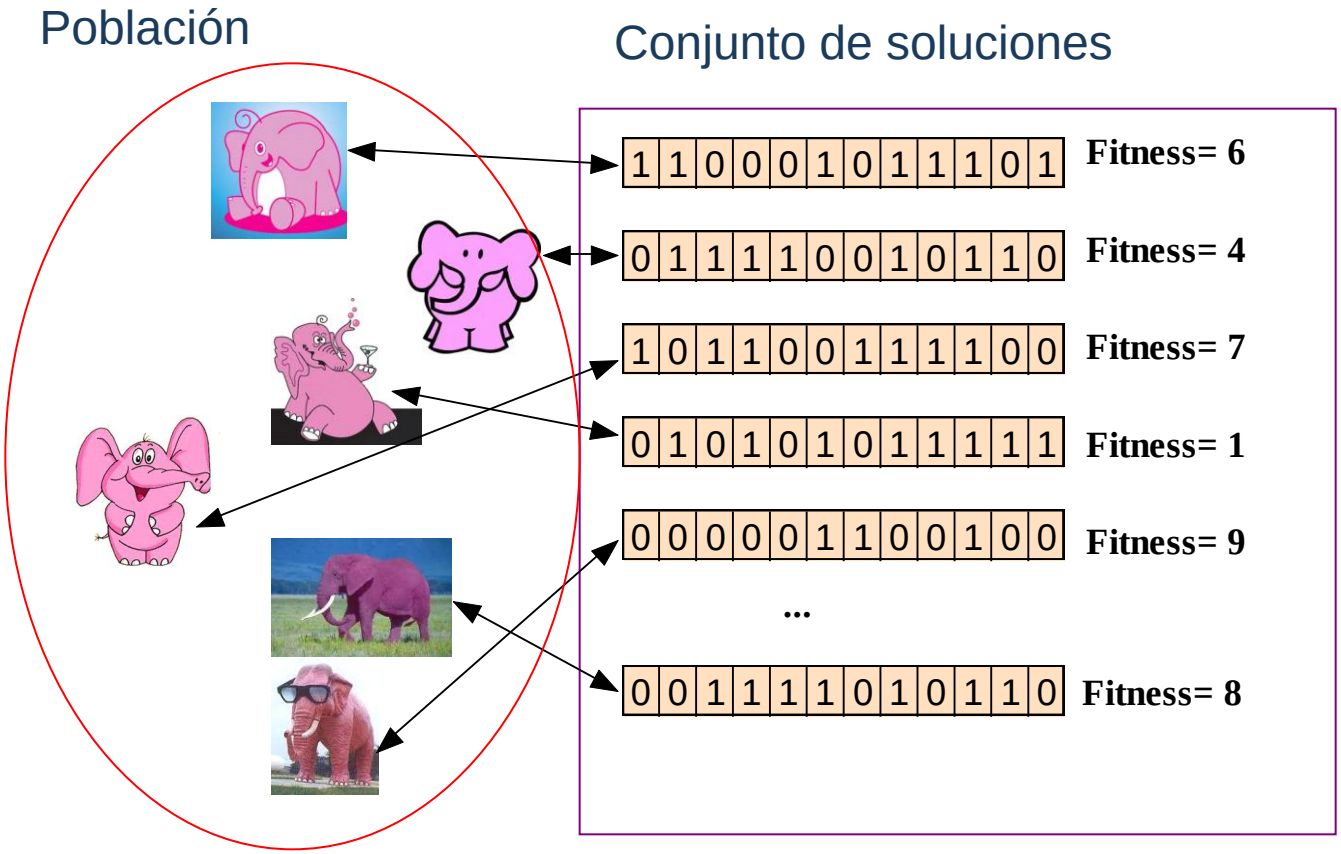
Algoritmos genéticos

- **Ejemplo:** Cromosoma que codifica una solución a un problema. Cada característica del problema es un valor 0/1.



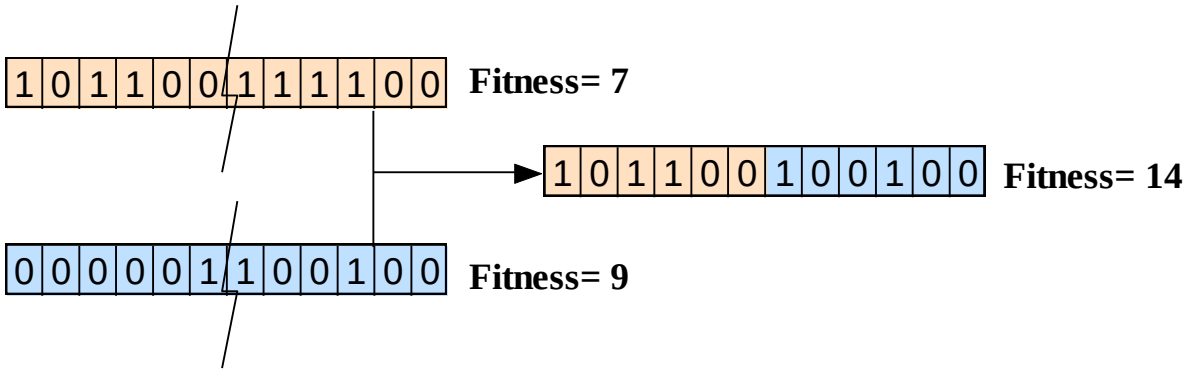
Algoritmos genéticos

- **Ejemplo:** Población. Conjunto de individuos (cada uno con su fitness).



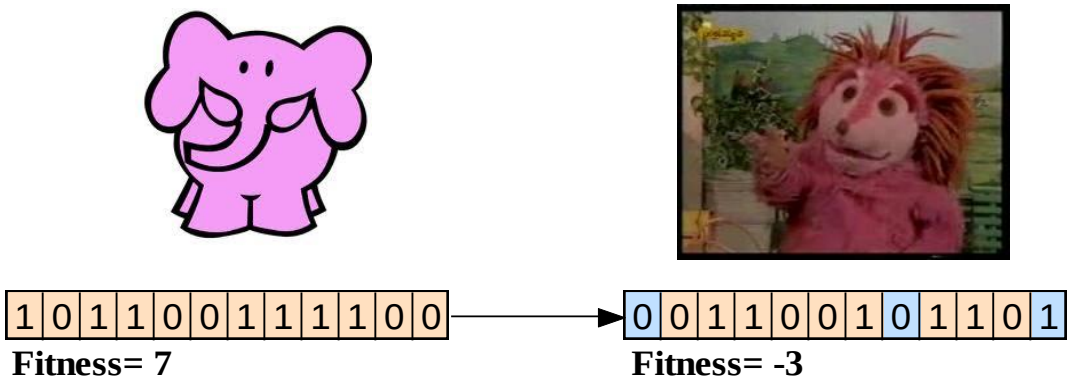
Algoritmos genéticos

- **Ejemplo:** Cruce. Combinación de soluciones de la población para generar descendientes.



Algoritmos genéticos

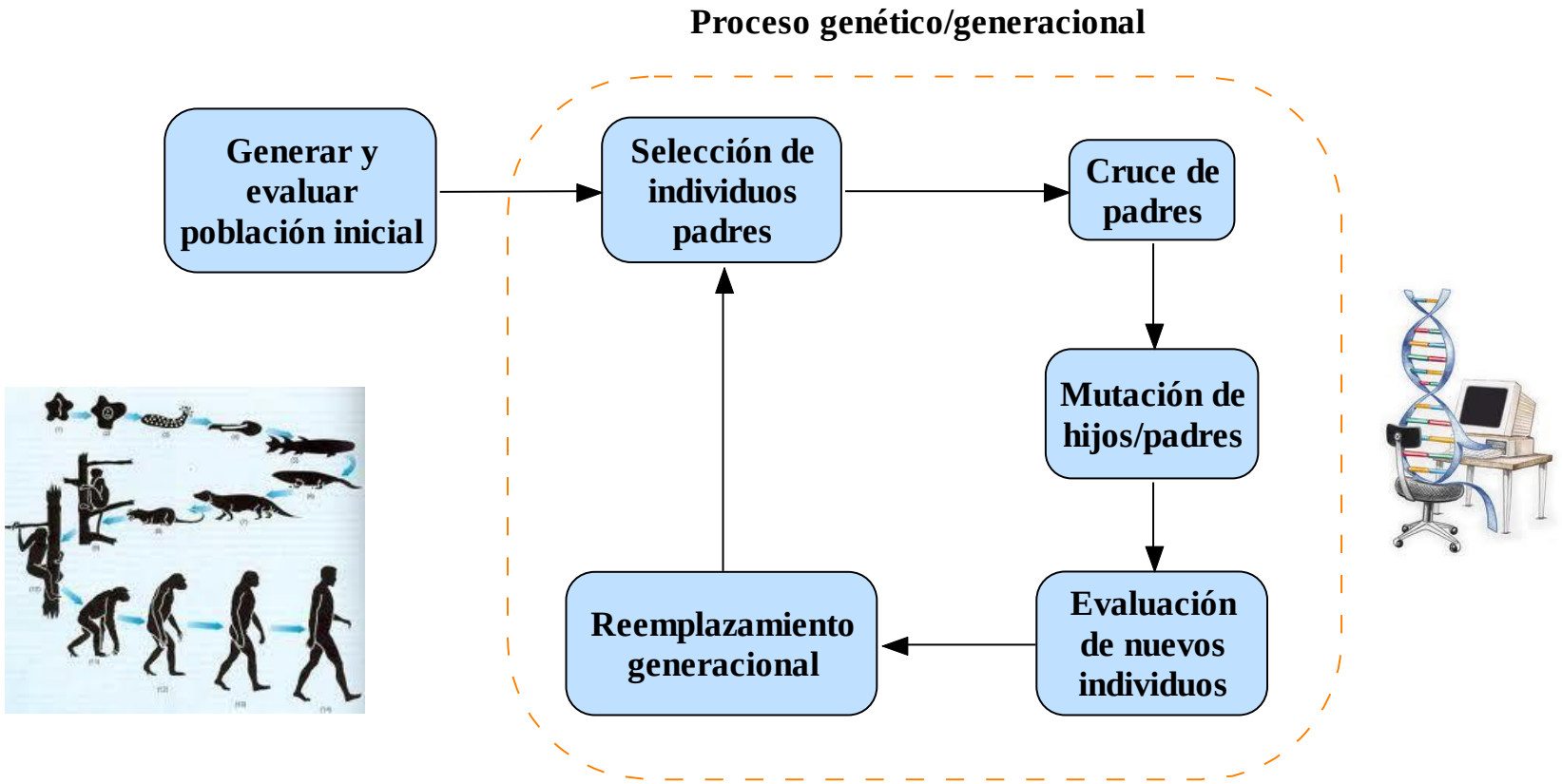
- **Ejemplo:** Mutación. Uno o más genes de un individuo pueden mutar para generar una nueva solución.



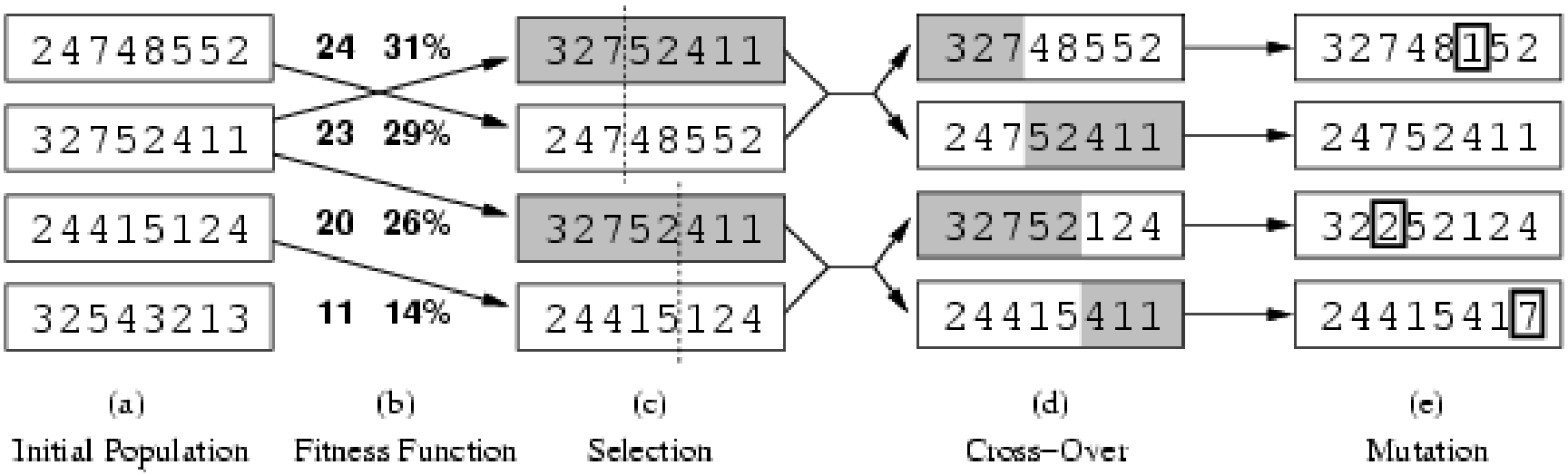
- En la población, hay una probabilidad dada a priori de que un individuo pueda mutar. A su vez, cuando un individuo muta, existe otra probabilidad de que cada gen mute o no.

Algoritmos genéticos

- **Proceso de un algoritmo genético:**



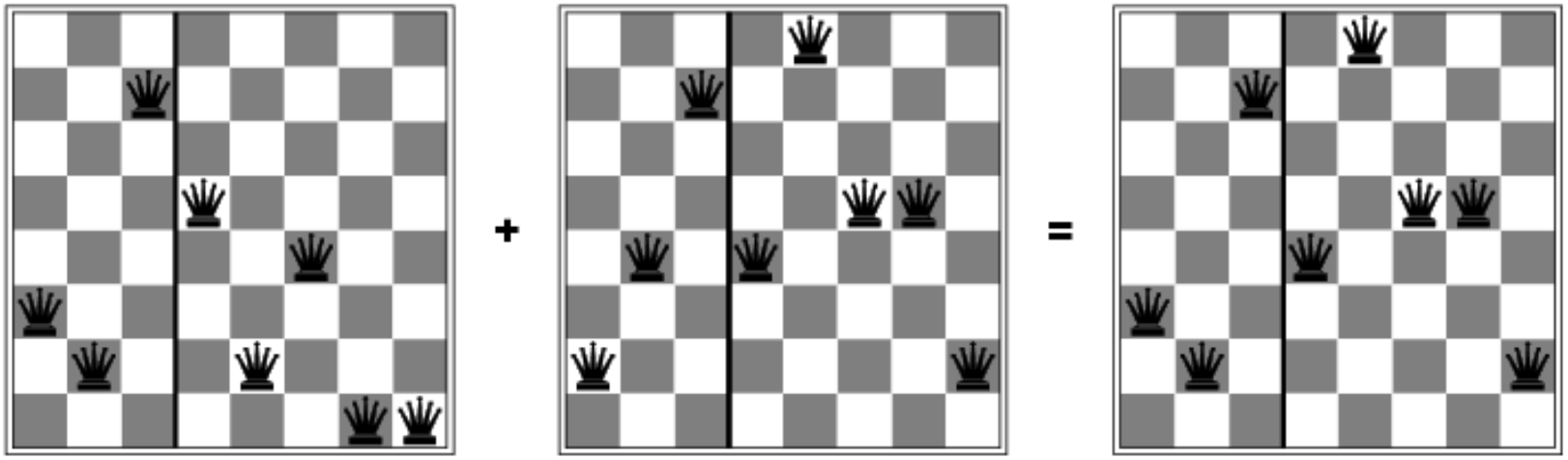
Ejemplo



Función de evaluación (8 reinas) = número de pares de reinas no atacadas

28 para una solución

Ejemplo



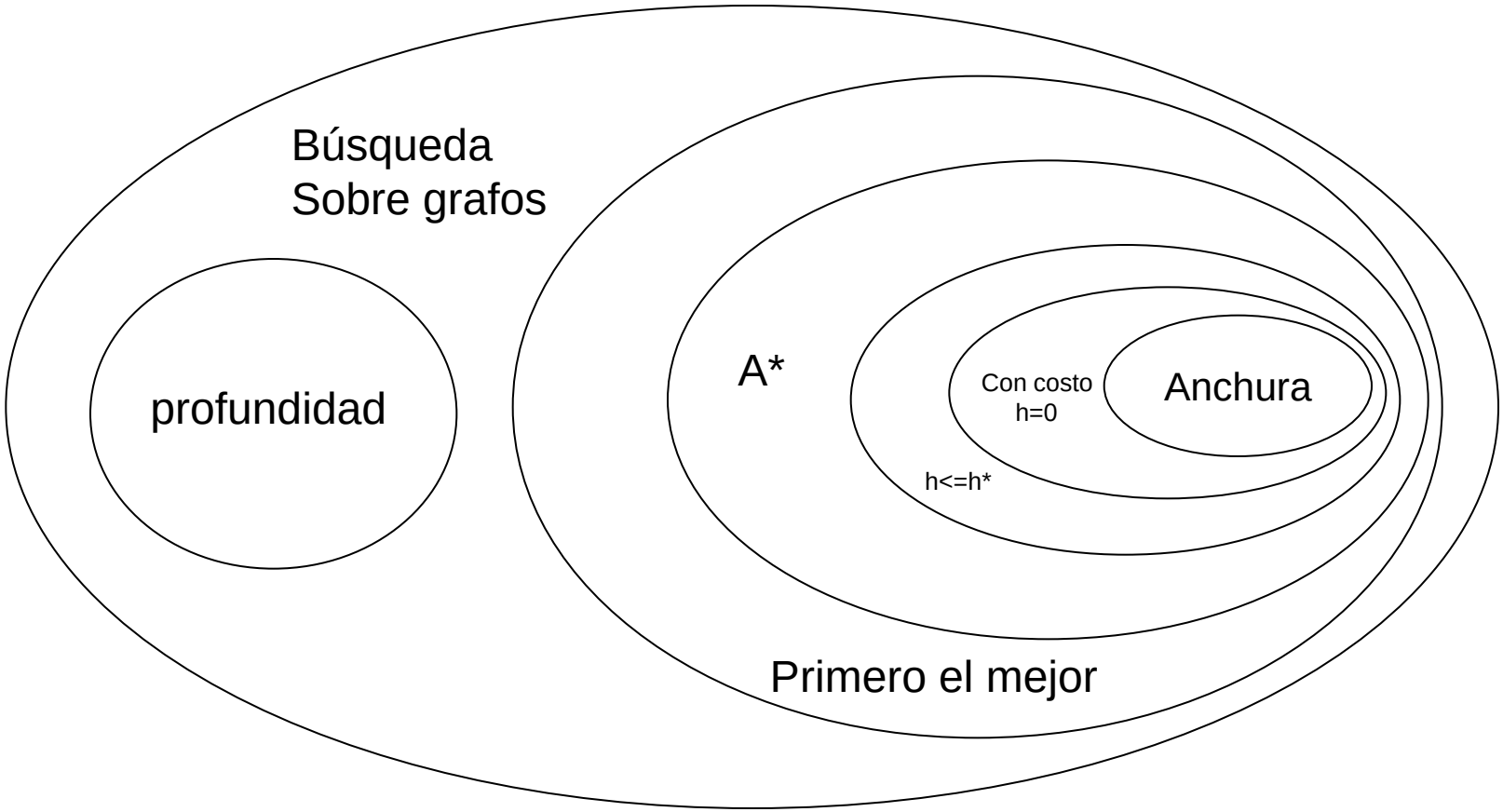
Búsqueda primero el mejor

- Algoritmo A*
- Búsqueda dirigida

Algoritmo A*

- ABIERTOS contiene el nodo inicial, CERRADOS esta vacío
- Comienza un ciclo que se repite hasta que se encuentra solución o hasta que ABIERTOS queda vacío
  - Seleccionar el mejor nodo de ABIERTOS
  - Si es un nodo objetivo terminar
  - En otro caso se expande dicho nodo
  - Para cada uno de los nodos sucesores
    - Si está en ABIERTOS insertarlo manteniendo la información del mejor padre
    - Si está en CERRADOS insertarlo manteniendo la información del mejor padre y actualizar la información de los descendientes
    - En otro caso, insertarlo como un nodo nuevo

Algoritmos de búsqueda



Dificultades del proceso

- Los procesos de percepción no siempre pueden obtener la información necesaria acerca del **estado** del entorno
- Las acciones pueden no disponer siempre de modelos de sus **efectos**
- Pueden haber otros procesos físicos, u **otros agentes**, en el mundo

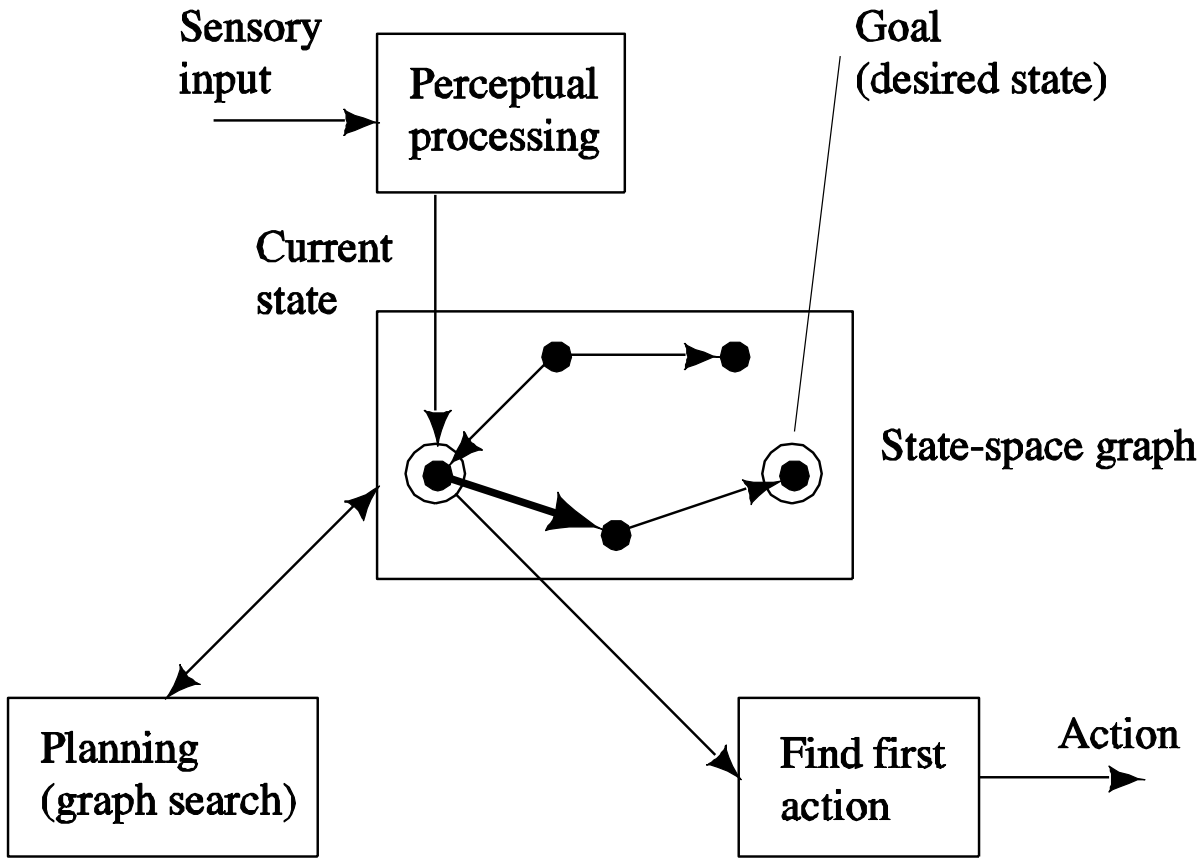
Dificultades del proceso

- En el tiempo que transcurre desde la construcción de un plan, el mundo puede cambiar de tal manera que el plan ya no sea adecuado
- Podría suceder que se le requiriese al agente actuar antes de que pudiese completar una búsqueda de un estado objetivo
- Aunque el agente dispusiera de tiempo suficiente, sus recursos de memoria podrían no permitirle realizar la búsqueda de un estado objetivo

Heurísticas sobre el proceso de búsqueda

- Arquitecturas combinadas de percepción y planificación
- Búsqueda orientada a subobjetivos
- Búsqueda jerárquica
- Búsqueda con horizonte

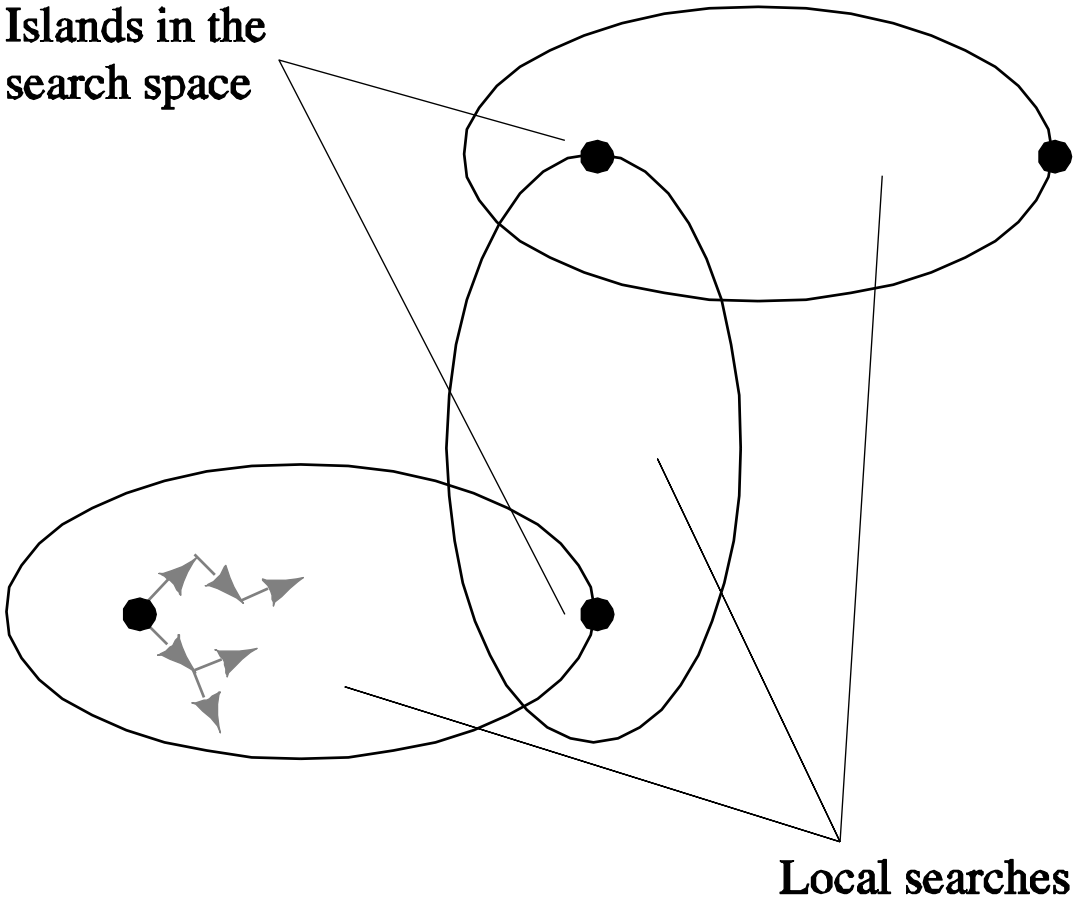
Arquitectura percepción/planificación/actuación



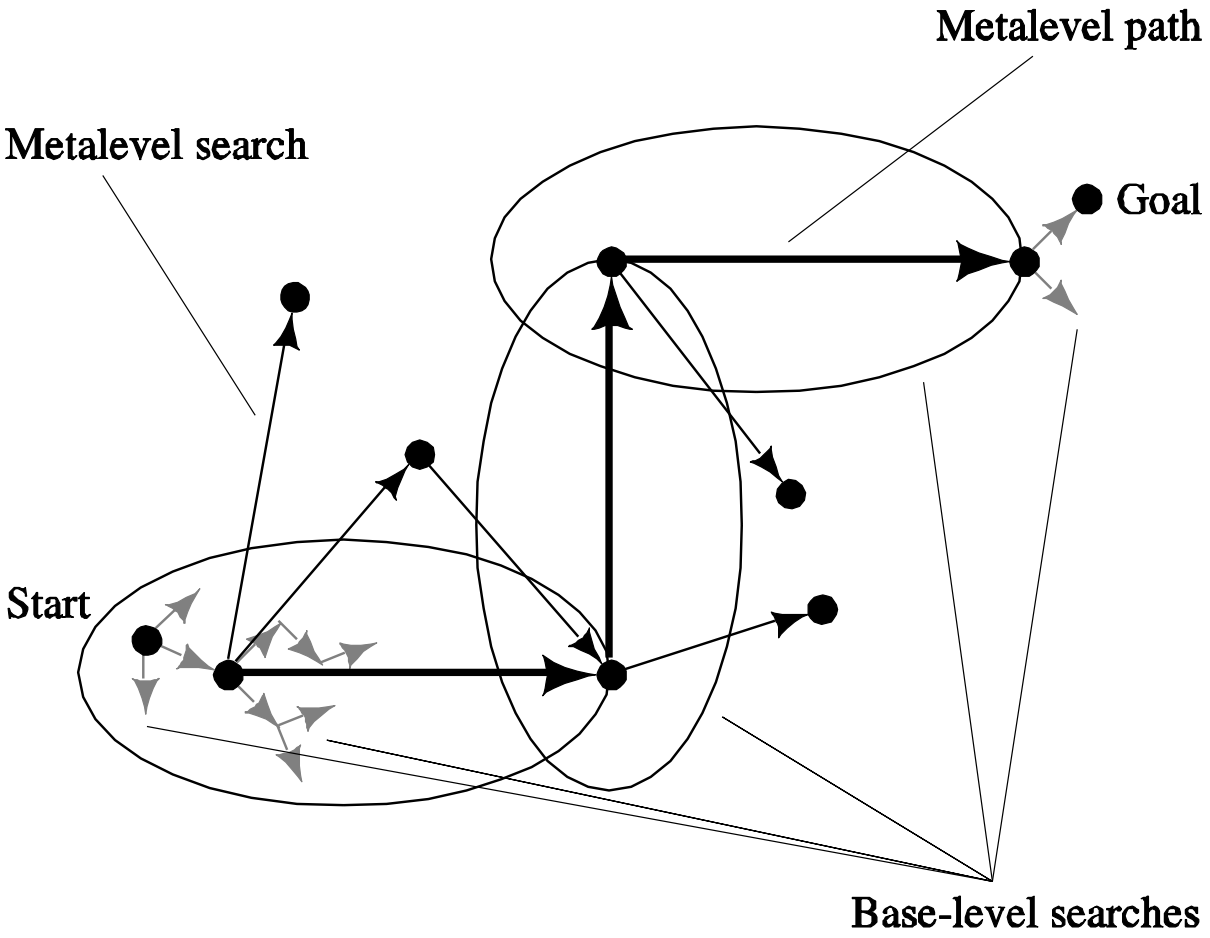
Heurísticas sobre el proceso de búsqueda

- Búsqueda orientada a subobjetivos
- Búsqueda con horizonte
- Búsqueda jerárquica

Búsqueda orientada a subobjetivos



Búsqueda jerárquica



Búsqueda jerárquica