Agentes Inteligentes

Javier García

Departamento de Electrónica y Computación
Universidad de Santiago de Compostela

October 12, 2021

Part IV

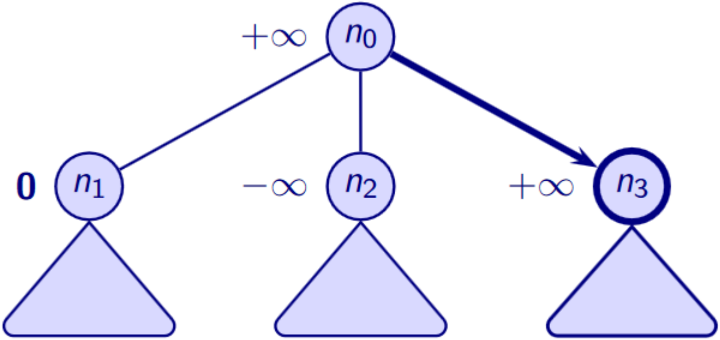
Búsqueda con adversarios y local

Contenidos

Caracterización del problema

- **Suma nula**: lo que gana uno, lo pierde el otro
- **Dos agentes** (aunque se puede generalizar a más agentes)
- **Información completa**: se conoce en cada momento el estado completo del juegov
- **Deterministas**: no entra en juego el azar
- **Alternados**: las decisiones de cada agente se toman de forma alternada

Resolución con búsqueda completa



Observaciones

- Nodos hoja: ganar (∞), perder ($-\infty$), empatar (0)
- Problema: es intratable; no se puede realizar en un tiempo razonable

Algoritmo Minimax

Solución

- Utilizar una función de evaluación que nos valore el estado actual
- Calcular las situaciones hasta una profundidad máxima

Funciones de evaluación

- 3 en Raya: El número de posibles 3 en Raya de “X” menos el número de posibles 3 en raya de “O”

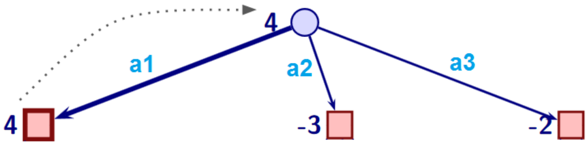
Idea general

- Elegir en cada momento la mejor opción que nos indique la función de evaluación f(n) hasta una profundidad máxima, considerando que:
  - Un jugador intenta maximizar f(n) (Jugador MAX)
  - Un jugador intenta minimizar f(n) (Jugador MIN)

Algoritmo Minimax

v_i ◯: Nodos MAX (juega el Minimax)

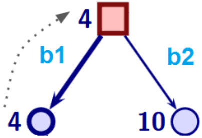
v_i □: Nodos MIN (juega el oponente)



Algoritmo Minimax

v_i : Nodos MAX (juega el Minimax)

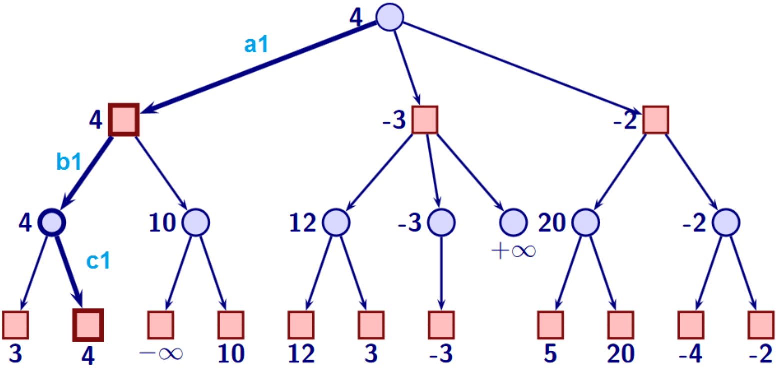
v_i : Nodos MIN (juega el oponente)



Algoritmo Minimax

v_i ○: Nodos MAX (juega el Minimax)

v_i □: Nodos MIN (juega el oponente)



Cálculo del valor de cada nodo

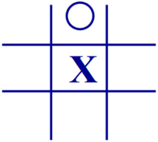
$$
f(n) = \begin{cases} +\infty & \text{si n es una situación ganadora} \\ -\infty & \text{si n es una situación perdedora} \\ 0 & \text{si n es una situación de empate} \\ f_{ev}(n) & \text{si } P = p_{max} \\ \max_{S_i \in S(n)} f(S_i) & \text{si n es nodo } MAX \text{ y } p < p_{max} \\ \min_{S_i \in S(n)} f(S_i) & \text{si n es nodo } MIN \text{ y } p < p_{max} \end{cases} \tag{1}
$$

Algoritmo Minimax

Procedimiento minimax(Situación, Profundidad)

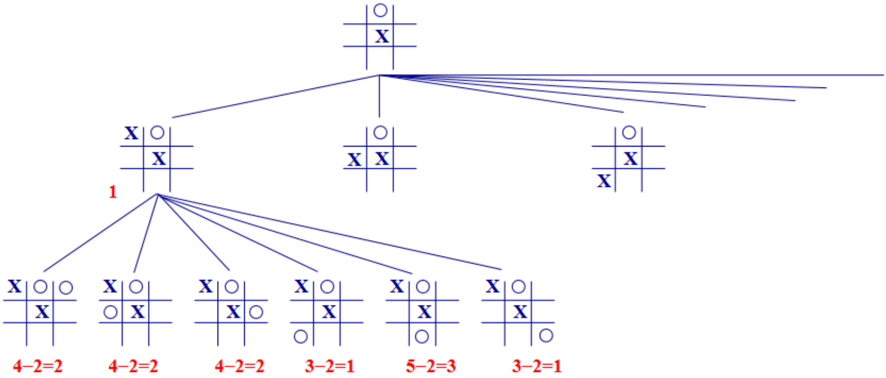
SI Profundidad $= p_{max}$
ENTONCES devolver evaluación (Situación)
SI NO SI ganadora (Situación)
ENTONCES devolver $+\infty$
SI NO SI perdedora (Situación)
ENTONCES devolver $-\infty$
SI NO SI empate (Situación)
ENTONCES devolver 0
SI NO
S = sucesores (Situación)
L = lista de llamadas a minimax ($S_i \in S$, Profundidad + 1)
SI nivel-max (Profundidad)
ENTONCES devolver max (L)
SI NO devolver min (L)

Tic-tac-toe - Minimax

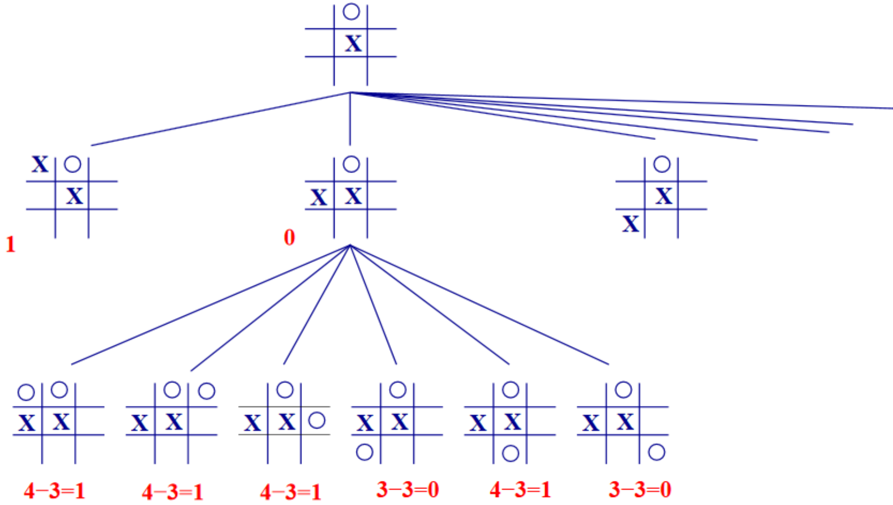


- Función de evaluación f(n): número de posibilidades de hacer tres en raya del jugador menos número de posibilidades de hacer tres en raya del oponente
- Si colocamos la primera ficha en:
  - el centro: 4 posibilidades para hacer 3 en raya ($f(n) = 4 - 4 = 0$)
  - en una esquina: 3 posibilidades ($f(n) = 3 - 5 = -2$)
  - en el centro de un lateral: 2 posibilidades ($f(n) = 2 - 6 = -4$)

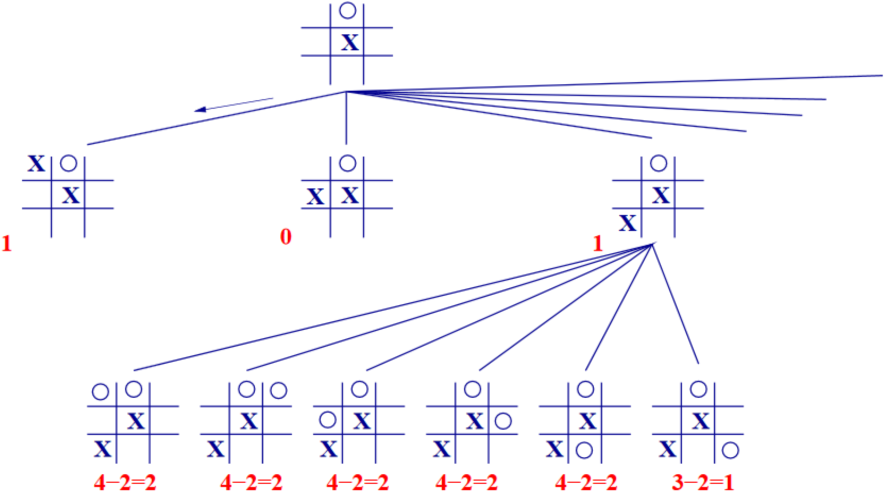
Tic-tac-toe - Minimax



Tic-tac-toe - Minimax



Tic-tac-toe - Minimax



Búsqueda local

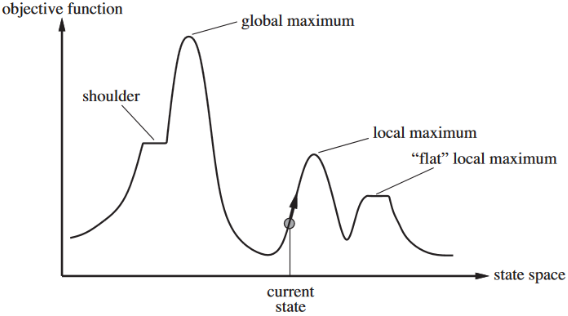
Javier García

Departamento de Electrónica y Computación
Universidad de Santiago de Compostela

October 12, 2021

Búsqueda local

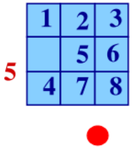
- En búsqueda clásica la solución es una secuencia de acciones que conducen de un estado inicial a un estado final
- En búsqueda local **lo importante es el estado final** o solución, **no importa el camino seguido hasta él** (e.g., problema de las 8-reinas)
- Idea básica:
  - Para una solución s actual de un problema, el algoritmo se mueve de s a otra solución s' siempre que s' sea mejor que la solución s

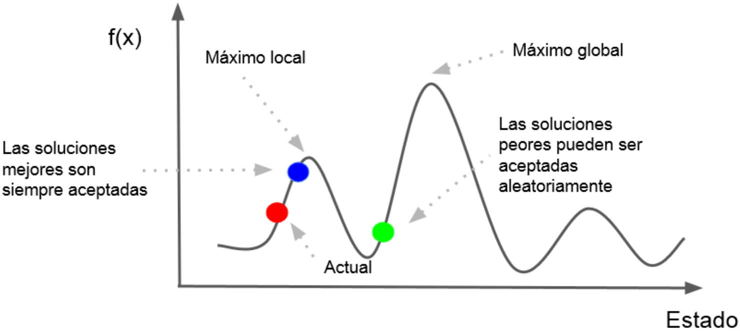


Simulated Annealing (I)

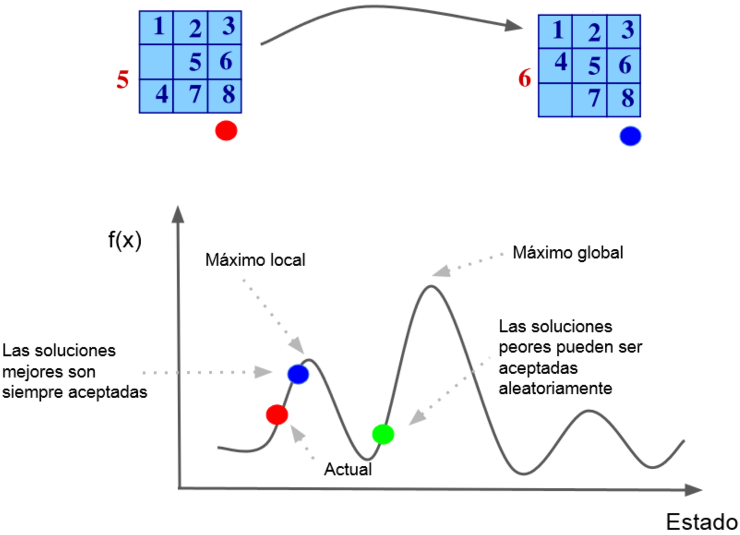
- Basado en el proceso de templado de los metales: los metales a alta temperatura se enfrían gradualmente
- Empieza con una solución inicial arbitraria y busca incrementalmente mejorar la función objetivo
- Durante cada iteración se considera un estado vecino del estado actual
  - El vecindario de una solución está conformado por todos los estados a los que se puede llegar desde la solución actual
  - El vecino se podría generar perturbando ligeramente la solución actual
- Al contrario que otros algoritmos, probabilísticamente puede aceptar soluciones que tengan un coste peor que la solución actual
  - Un parámetro de **temperatura** t regula la probabilidad de aceptación de soluciones peores que la actual

Simulated Annealing (II)

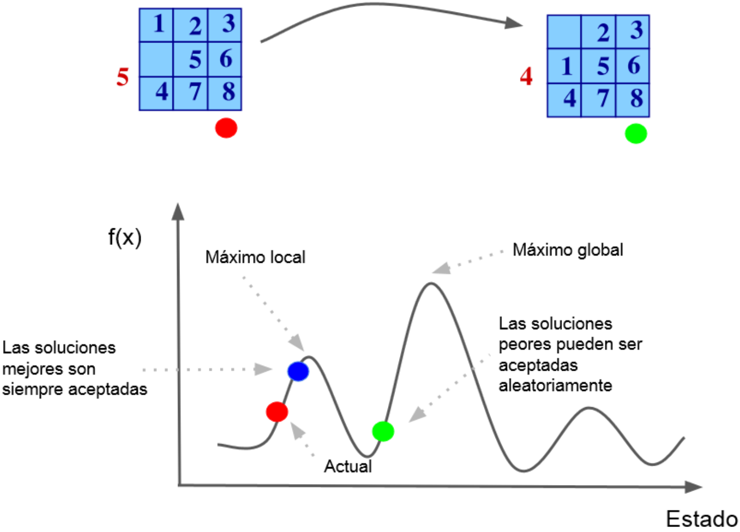




Simulated Annealing (II)



Simulated Annealing (II)

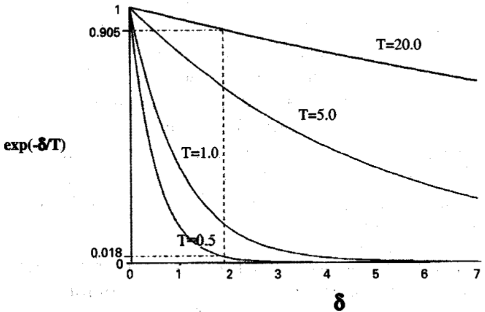


Simulated Annealing (III)

```
simulated_annealing(solución_actual, temperatura)
coste_actual = coste(solución_actual)
while not condición_parada do
  nueva_solución = vecino(solución_actual)
  nuevo_coste = coste(nueva_solución)
  δ = nuevo_coste - coste_actual
  if δ < 0 then    # < minimiza, > maximiza
    solución_actual = nueva_solución
  else
    if e^(-δ/t) > RAND(0, 1) then
      solución_actual = nueva_solución
  t = decrementar(t)
```

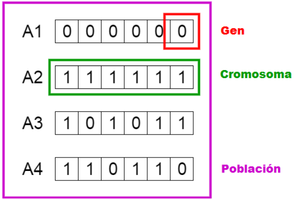
Simulated Annealing (IV)

- La variable t se inicializa a un valor alto al principio, y se va reduciendo cada iteración mediante un mecanismo de enfriamiento de la temperatura
- **A mayor temperatura, mayor probabilidad de aceptación de soluciones peores**. Así, el algoritmo acepta soluciones mucho peores que la actual al principio de la ejecución **(exploración)** pero no al final **(explotación)**
- **A menor diferencia de costes, mayor probabilidad de aceptación de soluciones peores**



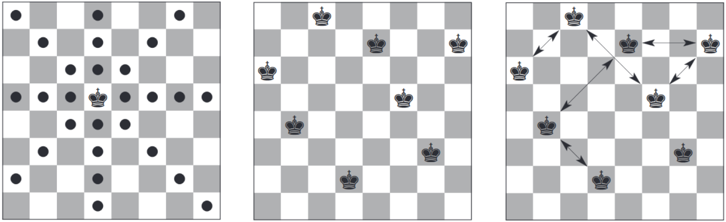
Algoritmos Genéticos

- Algoritmos inspirados en la evolución biológica (Holland, 1975)
- Evolución de una **población de individuos** hacia individuos óptimos
  - Cada individuo se representa mediante un conjunto de genes que conforman un cromosoma
  - Cada cromosoma tiene asociada un valor de **fitness** o de rendimiento sobre cómo de bueno es ese individuo
- Operadores:
  - Selección (de los mejores individuos)
  - Cruzamiento (apareamiento de los individuos seleccionados)
  - Mutación (mutuación aleatoria de genes de los nuevos individuos creados, diversidad)



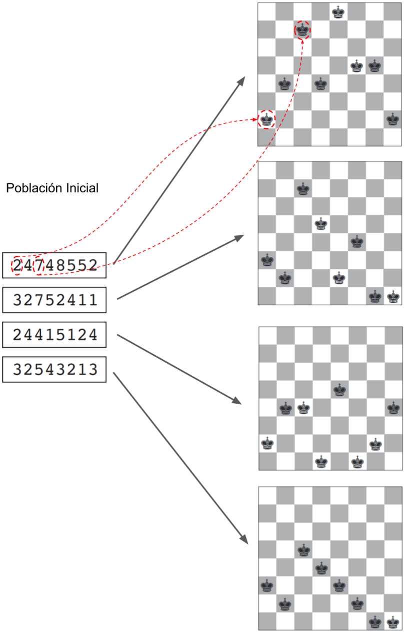
Población inicial

- Generalmente comienza con una población de individuos generados de forma aleatoria
- Generalmente cada individuo se codifica como una cadena de dígitos (e.g., 0s y 1s)
- 8-reinas



- ¿Cómo se pueden representar los individuos?

Población inicial



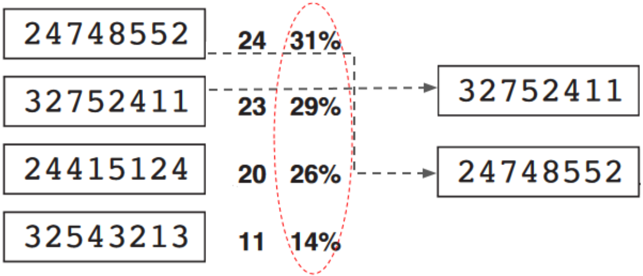
Fitness Function

- Mayor cuanto mejor es el individuo
- Muchas funciones de *fitness* posibles, pero en este caso...
- ...pares de reinas que no atacan (28 en la solución)

24748552	**24**
32752411	**23**
24415124	**20**
32543213	**11**

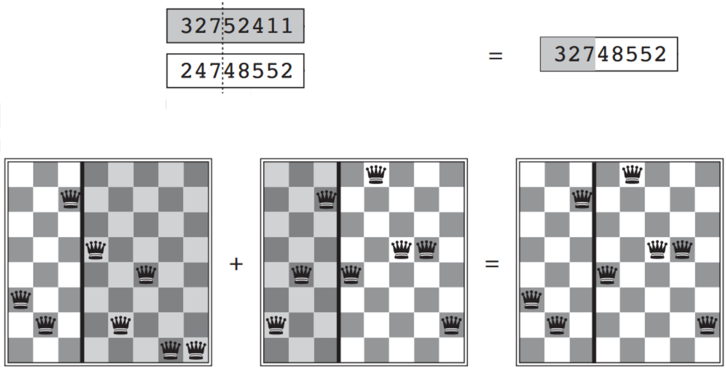
Selección

- Muchas formas de selección, pero en este caso...
- ...se seleccionan pares de individuos para la reproducción de acuerdo al fitness
- Un individuo puede ser seleccionado más de una vez, y también podría no ser seleccionado nunca



Cruzamiento

- Muchas formas de cruzamiento, pero en este caso...
- ...se selecciona un punto de cruzamiento de forma aleatoria y se genera un nuevo individuo con los genes de los padres



Mutación

- Muchas formas de mutación, pero en este caso...
- ...se selecciona una reina de forma aleatoria, y se la cambia de posición también de forma aleatoria

Algoritmo Genético

```
algoritmo_genético(población, función_fitness)
  do
    nueva_población ← ∅
    for i = 1 to TAMAÑO(población) do
      x ← SELECCION(población, función_fitness)
      y ← SELECCION(población, función_fitness)
      hijo ← CRUZAMIENTO(x,y)
      if RAND(0,1) < prob_muta then hijo ← MUTACIÓN(hijo)
      AÑADIR hijo a nueva_población
    población ← nueva_población
  while not condición_parada
  return el mejor individuo en población
```

Problemas

- Uno de los puntos críticos es la selección de una codificación adecuada
- Muchos operadores, funciones de fitness, esquemas de selección y mutación, configuración de parámetros, etc
- No hay garantías teórica de que vayan a encontrar la mejor solución
- Costosos computacionalmente, algunos problemas requieren días e incluso semanas
- Muy populares gracias a sus raíces en la teoría de la evolución