Memoria TP6 – Parte 1

Inteligencia artificial 2019-2020

Pedro Tamargo Allué (758267)

2019

Contenido

[Parte 1: Algoritmos de búsqueda local 2](#_Toc24887589)

[Algoritmo Hill Climbing 2](#_Toc24887590)

[Algoritmo Simulated Annealing 2](#_Toc24887591)

[Algoritmo Genetic 2](#_Toc24887592)

[Parte 2: Resolución de Sudokus mediante propagación de restricciones. 3](#_Toc24887593)

[Parte 3: Propagación de restricciones y búsqueda local. 3](#_Toc24887594)

# Parte 1: Algoritmos de búsqueda local

Para la realización de esta tarea se han elegido los algoritmos: Hill Climbing, Simulated Annealing y Genetic Algorithm para resolver el problema de las N-reinas (N = 8).

## Algoritmo Hill Climbing

Este algoritmo busca optimizar la solución ya que, partiendo de un estado inicial completo, busca el sucesor que maximice el resultado. Es un algoritmo de búsqueda local ya que solamente se fija en sus sucesores para encontrar el resultado óptimo.

[insertar aquí resultados]

[insertar aquí explicación de los resultados]

## Algoritmo Simulated Annealing

Este algoritmo es un símil de la industria metalúrgica, donde, durante los procesos de templado (de aceros), la aleación se enfría drásticamente para maximizar el empaquetamiento de átomos de carbono en la estructura cristalina, lo cual implica mayor resistencia y dureza, así como fragilidad.

En este algoritmo, se utiliza una función , siendo constantes del problema. Para determinar si se pasa de un cierto estado a otro sucesor se utiliza una función de probabilidad dependiente del incremento de los costes entre el nuevo estado y el actual, . Por lo tanto, definimos la función de probabilidad como: , si , siendo la variable que se corresponde con el paso de iteración en la ejecución, o , si .

[ejecución con los mejores parámetros]

[tablas con ejecuciones]

## Algoritmo Genetic

Este algoritmo se basa en las mutaciones y cruces entre individuos dentro de una población. Su funcionamiento consiste en que, dada una población inicial y una probabilidad de mutación, seleccionamos a los individuos, usando una y una probabilidad (los individuos cuyo resultado después de aplicar la sea mayor, tendrán más probabilidades de ser elegidos), que transmitirán sus genes a la siguiente generación. Tras seleccionar a los individuos, procedemos a realizar combinaciones, para ello, seleccionamos (de forma aleatoria) el punto de cruce (), dividiendo el individuo en 2 partes. Después procedemos a realizar la mutación, creando dos nuevos individuos combinando las 4 partes de los individuos que se han seleccionado en base a la . En algunos casos pueden ocurrir mutaciones (dependiendo de la probabilidad de mutación), esto implica que alguno de los elementos generados en el paso anterior, cambie alguno de sus “genes” para mantener la diversidad de la población.

[insertar ejecución con parámetros “aleatorios”]

[insertar tablas de ejecuciones]

# Parte 2: Resolución de Sudokus mediante propagación de restricciones.

Las clases a realizar en esta parte son: . En hereda de y representa una celda del sudoku, tiene . La clase hereda de e impide que una asignación de variables sea igual a otra (como ). La clase contiene el código del método , en este se leen los ficheros de sudokus y se resuelven.

[ejemplo de un sudoku]

[ejemplo de los 156 sudokus]

# Parte 3: Propagación de restricciones y búsqueda local.

Para la realización de esta parte se han reutilizado las clases creadas en la parte 2, con modificaciones.

La clase hereda de , y contiene 2 enteros, que simbolizan la fila y la columna.

La clase hereda de , y comprueba que una asignación de variables cumple con las restricciones (no son la misma reina, no están en la misma fila y no están en la misma diagonal).

La clase , contiene el método que utiliza para resolver el tablero.

[ejemplo de ejecución]