Pontificia Universidad Católica del Perú

Tarea Académica

Aplicaciones de Ciencias de la Computación

Luis Ramirez Osorio
20161567

John Benites Huamán
20161589

Herbert Esplana Hurtado
20152801

Reiver Lopez Casavilca
20156134

29 de agosto de 2019

Índice general

0.1. Desafío 1

0.1.1. Problema 1

NodeID inicial = 104878620, NodeID objetivo = 105012740

Problema 1	DFS	BFS	IDS	BIS	ASTAR
# nodos visitados	49036	214	163007	198	54
# nodos en memoria	49048	264	10	270	72
# nodos en la ruta encontrada	10	10	10	10	13
Costo de la ruta encontrada(Km)	0.72	0.77	0.77	0.77	0.71
Tiempo en ejecución(Segundos)	31.53	0.004	0.942	0.056	0.001

0.1.2. Problema 2

NodeID inicial = 275754986, NodeID objetivo = 105012740

Problema 1	DFS	BFS	IDS	BIS	ASTAR
# nodos visitados	37790	916	***	1180	115
# nodos en memoria	40329	1029	***	1466	146
# nodos en la ruta encontrada	7300	18	***	18	24
Costo de la ruta encontrada(Km)	342.33	1.62	***	1.62	1.38
Tiempo en ejecución(Segundos)	29.26	0.031	***	6.555	0.003

0.1.3. Problema 3

NodeID inicial = 656071251, NodeID objetivo = 105012740

Problema 1	DFS	BFS	IDS	BIS	ASTAR
# nodos visitados	16531	3165	***	3332	413
# nodos en memoria	18999	3319	***	3942	483
# nodos en la ruta encontrada	6868	34	***	1072	47
Costo de la ruta encontrada(Km)	331.33	2.63	***	50.22	2.34
Tiempo en ejecución(Segundos)	9.144	0.156	***	115.9	0.032

0.1.4. Análisis y conclusiones

Por un lado, si vemos los datos obtenidos en las tablas de los tres problemas podemos concluir que ASTAR es la búsqueda que emplea menos nodos en memoria, visita menos nodos y, por obvias razones, el menor costos de camino. Además, es el que tarda menos en encontrar una camino hacia el objetivo. Esto se debe a que la búsqueda por ASTAR se realiza con información adicional la cual en este caso es la heurística.

Por otro lado, si analizamos los datos de las búsquedas sin información podemos ver que el mejor de los cuatro en términos de velocidad de ejecución es BFS. Sin embargo, esta búsqueda debería tener un mayor número de nodos en memoria puesto que su complejidad de espacio es de $O(b^d)$, pero no es así. Una posible explicación a esto es que la solución se encuentra en un nivel intermedio. Eso también explicaría el porque DFS almacena tantos nodos en memoria a pesar de que su complejidad de espacio es de $O(bm)$. Asimismo, la búsqueda bidireccional está en el intermedio entre DFS y BFS, ya que empleamos BFS para el nodo inicial y DFS para el nodo objetivo. Es por ello que almacena y visita menos nodos que la búsqueda pura DFS pero más que la búsqueda pura BFS. En cuanto IDS podemos ver que solo en el primer problema pudo correr en un tiempo aceptable, pero para los dos siguientes problemas tardaba más de quince minutos por lo que cortamos su ejecución. Esto puede ser a que tiene que recorrer el árbol varias veces desde el

inicio si es que no encuentra una solución. Es por ello que tiene muchos más nodos visitados que las otras tres búsquedas sin información porque repite la lectura de nodos.

Por todo ello podemos concluir que la mejor búsqueda es ASTAR. Además, por los datos que obtuvimos de las búsquedas sin información podemos decir que para este problema el nodo objetivo esta en un nivel intermedio del árbol y que está pegado hacia uno de los lados de tal forma que es uno de los peores casos para DFS, mientras que para BFS sería uno de los mejores casos.

0.1.5. Sugerencias

Las sugerencias para este problema en específico es que la mejor manera de encontrar el camino hacia un nodo objetivo mediante la búsqueda $A*$, sin embargo se necesitaría una heurística distinta para cada nodo y esto se podría solucionar si es que tuviéramos las coordenadas de cada uno y con esos datos podíamos calcular las distancias de un nodo a otro por lo que generalizaríamos el problema.

Otra sugerencia para este problema es que en este problema no se emplee el tree search para DFS ya que al intentarlo entrabamos en bucles, es por ello que recomendamos usar graph search aunque este guarde más nodos en memoria. En todo caso, no se debería utilizar DFS en una aplicación con datos similares en el mundo real. Dado su baja eficiencia temporal y si se desearan guardar los nodos visitados seria muy ineficiente espacialmente, también.

Por último, se podría sugerir el uso de Ant Colony Optimization ya que con este algoritmo se podría encontrar el camino con menor costo al nodo objetivo en un tiempo relativamente aceptable, sin depender de heurísticas y sin almacenar tantos nodos en memoria, el único problema que tendría sería que necesita de mucho procesamiento ya que a mayor número de hormigas se necesita calcular muchas veces las probabilidades de transición como la actualización de las feromonas en los caminos.Además de que sería necesario un análisis previo para generar la función de la feromona.

0.2. Desafío 2

0.2.1. Simulating Annealing

Tablero A

	SA Config 1: (DR=0.999)	SA Config 2: (DR=0.9999)	SA Config 3:) (DR=0.99999)
Score de la mejor solución encontrada	160	160	162
Tableros evaluados (# llamadas a scoreBoard())	100052.4	100032.4	62017.6
Tiempo total de procesamiento(seg.)	21.53	21.77	13.25

Tablero B

	SA Config 1: (DR=0.999)	SA Config 2: (DR=0.9999)	SA Config 3:) (DR=0.99999)
Score de la mejor solución encontrada	158	160	162
Tableros evaluados (# llamadas a scoreBoard())	100327	100030.4	88629.8
Tiempo total de procesamiento(seg.)	21.85	21.41	19.01

Tablero C

	SA Config 1: (DR=0.999)	SA Config 2: (DR=0.9999)	SA Config 3:) (DR=0.99999)
Score de la mejor solución encontrada	160	162	162
Tableros evaluados (# llamadas a scoreBoard())	100083.8	81006.6	77075.8
Tiempo total de procesamiento(seg.)	21.52	17.36	16.86

0.2.2. Algoritmo Genético

Tablero A

	AG solo mutación m=1	AG solo cruzamiento m=0		AG cruzamiento + mutación m=0.1		AG cruzamiento + mutación m=0.5	
		single	uniform	single	uniform	single	uniform
Score de la mejora solución encontrada	122	131	129	149	150	152	151
Generación de la mejor solución	0.00	8.00	3.40	5866.00	5732.20	3273.20	3376.80
Tableros evaluados hasta la generación de la mejor solución (# llamadas a scoreBoard())	3.40	6.00	5.80	21.20	22.00	24.60	21.00
Tiempo total de procesamiento	0.35	11.01	11.15	14.46	15.20	29.93	30.50

Tablero B

	AG solo mutación m=1	AG solo cruzamiento m=0		AG cruzamiento + mutación m=0.1		AG cruzamiento + mutación m=0.5	
		single	uniform	single	uniform	single	uniform
Score de la mejor solución encontrada	125	129	133	150	150	152	153
Generación de la mejor solución	0.00	7.00	7.30	5802.5	3767.75	3366.75	3570.00
Tableros evaluados hasta la generación de la mejor solución (# llamadas a scoreBoard())	2.80	6.00	7.25	21.00	22.40	22.45	23.75
Tiempo total de procesamiento	0.34	12.73	13.01	17.04	17.59	35.70	35.50

Tablero C

	AG solo mutación m=1	AG solo cruzamiento m=0		AG cruzamiento + mutación m=0.1		AG cruzamiento + mutación m=0.5	
		single	uniform	single	uniform	single	uniform
Score de la mejora solución encontrada	128	124	129	126	157	157	155
Generación de la mejor solución	0.00	0.00	0.00	0.00	6993.20	2451.40	4158.80
Tableros evaluados hasta la generación de la mejor solución (# llamadas a scoreBoard())	2.40	2.60	2.60	2.80	24.40	24.20	23.00
Tiempo total de procesamiento	0.39	0.38	0.38	0.38	17.41	34.67	35.13

0.2.3. Análisis y conclusiones

En primer lugar, analizando los datos obtenidos de Simulating Annealing podemos ver que mientras aumentamos la tasa de decaimiento (Decay rate DR) el score de la solución aumenta a tal punto que logramos encontrar la solución al problema. Además, el tiempo total de procesamiento también disminuye. Esto nos lleva a concluir que para este problema una tasa de decaimiento alto nos ayuda a encontrar la solución.

En segundo lugar, analizando los datos obtenidos del algoritmo genético podemos ver que si solo empleamos mutación los resultados obtenidos no varían con respecto al mejor individuo de la población inicial, ya que el número de llamadas a la función scoreBoard() es menor a cuatro en los tres tableros de sudoku. Cuando empleamos solo cruzamiento obtenemos resultados similares a solo mutación, el score no varía casi nada del mejor score obtenido de la población inicial sin importar el método de cruzamiento. La situación cambia cuando empleamos en conjunto la mutación con el cruzamiento. Como podemos ver en los resultados el score obtenido es relativamente mejor que los dos anteriores. Para el caso cuando la mutación es igual a 0.1 vemos que el cruzamiento uniforme es ligeramente mejor que el cruzamiento de un solo punto. Eso puede deberse a que con el cruzamiento uniforme se puede obtener mejores hijos que el cruzamiento de un solo punto. Para el caso de mutación de 0.5 no hay tanta mejoría con respecto a la anterior tasa de mutación ya que se mantiene entre el 150 y 157. En lo que si se diferencia en emplear ambos métodos (cruzamiento y mutación) a emplear uno solo es en el tiempo de ejecución. En las tres tableros podemos ver que va aumentando hasta llegar a 30 segundos aproximadamente.

Finalmente, si comparamos ambos algoritmos podemos ver que simulating annealing es mucho mejor que el algoritmo genético, ya que con el primero llegamos a obtener la solución del problema para los tres tableros, mientras que para el algoritmo genético no obtenemos la solución para ninguno de los tres tableros. El máximo score que obtuvimos fue con la ultima configuración: cruzamiento + mutación (0.5). Esto puede deberse a que con simulating annealing llegamos al óptimo global, ya que encontramos la tasa de decrecimiento de "temperatura"perfecta para este problema. Sin embargo, para genetic algorithm, a pesar de modificar sus diferentes variables, no encontramos la mejor configuración para este problema y la solución se queda estancada en un óptimo local.

0.2.4. Sugerencias

Se debería volver a analizar el problema de encontrar la solución del sudoku, mediante un algoritmo genético. Esto sería necesario para poder plantear una mejor representación de genes para este problema. Puesto que, se puede observar en los resultados para cada "puzzle"que cuando se trabaja únicamente con la mutación esta no mejora el "fitness"de la población. Esto podría ser causado por la representación de genes que se tiene, en la que los bloques de 3x3 se consideran indivisibles, no permitiendo que se pueda ingresar más aleatoriedad al algoritmo mediante

mutación.

Una posible solución podría ser, la de considerar a los bloques de 3x3 como genes que se pueden subdividir en más alelos. Otra posible solución, sería la de modificar la función de mutación para forzar que ,cada vez que se mute un individuo, el resultado devuelva a un individuo con mejor "fitness". Así nos aseguraríamos que la mutación siempre mejore al individuo.