

Reporte Tarea 3 Inteligencia Artificial

Algoritmo Hill Climbing para Sudoku.

Descripción.

En ciencia de la computación, **Hill Climbing** (ascenso de colinas, en alguna literatura) es una técnica de optimización matemática que pertenece a la familia de los algoritmos de búsqueda local. Es un algoritmo iterativo que comienza con una solución arbitraria a un problema, luego intenta encontrar una mejor solución variando incrementalmente un único elemento de la solución. Si el cambio produce una mejor solución, otro cambio incremental se le realiza a la nueva solución, repitiendo este proceso hasta que no se puedan encontrar mejoras.

El Hill Climbing es bueno para encontrar un óptimo local (una solución que no puede ser mejorada considerando una configuración de la vecindad) pero no garantiza encontrar la mejor solución posible (el óptimo global) de todas las posibles soluciones (el espacio de búsqueda). La característica de que sólo el óptimo local puede ser garantizado puede ser remediada utilizando reinicios (búsqueda local repetida), o esquemas más complejos basados en iteraciones, como búsqueda local iterada, en memoria, como optimización de búsqueda reactiva y búsqueda tabú, o modificaciones estocásticas, como **simulated annealing**.

Pseudocódigo:

```
input: IterMax, ProblemSize
output: Current
Current = RandomSolution(ProblemSize)
for iter=0 to IterMax
    Candidate=RandomNeighbor(Current)
    if (cost(Candidate)>=cost(Current))
        Current = Candidate
    end
end
```

Implementación.

En la implementación propuesta para el problema del Sudoku, se consideran 4 distintas representaciones de la solución, tal como se vieron en clase (Representación Completa, por bloques, por Filas y por Columnas). Para cada una de estas representaciones se elaboró un programa que trata de aproximar la solución al problema del Sudoku utilizando el algoritmo Hill Climbing.

En nuestra implementación, una solución representa un arreglo de enteros que constituyen los valores que deben tomar cada una de las posiciones del Sudoku para resolverse, en el caso de la representación completa, la solución se almacena en una estructura en forma de matriz, que contiene tanto los elementos dados en el problema como los que se van a calcular; en el caso de Filas y Columnas, se tiene una estructura que consta de 9 vectores, los cuales almacenan solo los elementos no dados por el problema, y en el caso de los Bloques, se tiene de igual forma la estructura que almacena solo los valores no dados que corresponde a cada bloque del Sudoku.

Tal como se indica en el pseudocódigo mostrado anteriormente, se inicia generando una solución aleatoria de acuerdo a las condiciones del problema, para cada una de las representación se generó una

permutación (de acuerdo a su tipo de representación) para cada elemento de esta, por ejemplo, para el caso de los bloques, generamos una permutación para cada bloque, respetando las casillas ya dadas, y verificando que no se repita ningún número dentro de cada bloque, así de la misma manera se hizo para cada representación.

Para poder establecer una toma de decisiones para ir escalando en nuestro algoritmo, definimos como la función de costo como la sumatoria de todas las penalizaciones de una solución dada, a su vez definimos como penalización cuando existen dos casillas en conflicto, en tal caso se suma +1 a cada casilla. En el caso de una función de costo que considera los elementos ya dados, se penaliza con +5 extra a las casillas que estén en conflicto con una ya dada.

Teniendo ya una primera solución para nuestro problema, se procede a generar una solución vecina, la cual definimos como una solución en la cual el elemento con la mayor penalización de la solución original es intercambiado por otro elemento dentro de su misma representación. Después comparamos la unión de costo de ambas soluciones y nos quedamos con la mejor.

Para tratar de obtener el mejor resultado posible, el programa permite un total de 1000 reinicios, tal como se indico para esta practica; por otra parte, también consideramos un contador de hasta 20 iteraciones que registra las veces continuas en que al generar una solución vecina, esta produce un valor de costo mayor que la solución original, en caso de que esto ocurriera 20 veces, reiniciamos el algoritmo (conservando siempre la mejor solución) y restamos 1 a los 1000 reinicios disponibles.

Resultados.

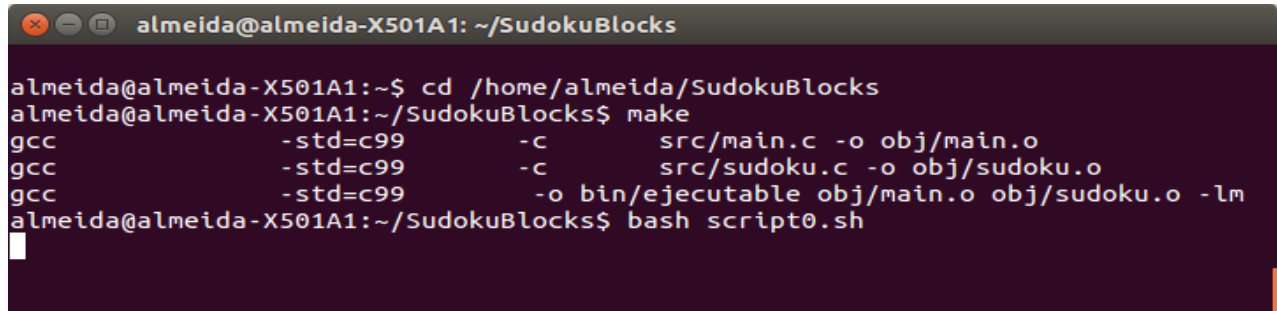
A continuación se muestra una tabla en donde diferentes instancias de Sudoku se probaron para cada modalidad de nuestro algoritmo, en el encabezado superior se muestra el tipo de representación, Normal significa que no se consideraron penalizaciones extras, se muestra el promedio de los valores de costo obtenidos en las 100 ejecuciones y el tiempo de ejecución en segundos.

	Completa				Bloques				Filas				Columnas			
	Normal		Extra		Normal		Extra		Normal		Extra		Normal		Extra	
	Costo	Tiempo	Costo	Tiempo	Costo	Tiempo	Costo	Tiempo	Costo	Tiempo	Costo	Tiempo	Costo	Tiempo	Costo	Tiempo
Easy1	5.4	149	0.4	134	5.4	284	5.4	270	251	223	0.4	236	4.8	281	0.4	277
Easy2	5.2	153	3.8	152	9.8	222	9.4	248	7.2	276	3.2	284	9.2	283	2.8	265
Easy3	5.4	140	3.4	134	10	247	5.4	282	5.8	225	0.8	283	4.6	294	1	239
Med1	16.4	132	16.2	135	16.4	260	14.2	234	14.8	289	14.2	245	12.6	260	14.4	286
Med2	11.4	137	11.8	160	14	244	15.4	283	14.6	269	10.4	273	9.4	283	13.8	289
Med3	9.4	149	9.4	154	11.6	260	11.6	271	10.4	277	10.2	234	8.6	232	12.8	239
Hard1	13.4	149	16.2	132	16.6	286	15.6	267	14.8	279	15.4	221	12.4	290	14.6	261
Hard2	11	147	12.6	151	13.4	283	10.4	288	10.8	271	13.4	226	13.6	286	15.4	231
Hard3	14	149	14.4	153	15.2	234	13.2	243	14	230	13.6	221	12	288	14	229
Evil1	15.2	147	14.8	161	17.8	230	15.4	273	15.2	245	15	235	12.8	221	16.4	249
Evil2	11.6	160	13.4	162	15.2	284	15.2	221	13.8	221	14.6	289	13.6	280	15	256
Evil3	14.2	146	14.6	155	15.4	246	14.8	232	14.8	269	15.2	272	14	262	15.4	281

En general se observa mayor eficiencia en el caso de las representaciones por fila y por columnas, probablemente debido al diseño del Sudoku, y también se observa la diferencia en tiempos de ejecución entre la representación completa y las demás, esto se debe a las complicación de utilizar las representaciones “incompletas” al realizar los cálculos necesarios durante la ejecución del algoritmo.

Compilación/Ejecución.

Se tienen 4 programas de acuerdo a cada estructura de datos, cada uno de los programas incluye un *makefile* para compilarse, con los comandos *make*, *make run* y *make clean*, así como dos scripts de ejecución tipo “bash” para correr todas las instancias, (el archivo de la instancia debe estar dentro de la carpeta del programa), los resultados se generan en un archivo para cada instancia.

A terminal window with a dark purple background and white text. The window title is 'almeida@almeida-X501A1: ~/SudokuBlocks'. The terminal shows the following commands and output:

```
almeida@almeida-X501A1:~$ cd /home/almeida/SudokuBlocks
almeida@almeida-X501A1:~/SudokuBlocks$ make
gcc          -std=c99          -c          src/main.c -o obj/main.o
gcc          -std=c99          -c          src/sudoku.c -o obj/sudoku.o
gcc          -std=c99          -o bin/ejecutable obj/main.o obj/sudoku.o -lm
almeida@almeida-X501A1:~/SudokuBlocks$ bash script0.sh
```