El Juego de la vida y búsqueda local

Nelson Becerra Correa*

Ismel Brito Rodríguez**

Resumen

En este artículo se desarrolla e implementa un autómata celular conocido como el *juego de la vida*. Se define su estructura básica y se programa utilizando técnicas de búsqueda local, para hallar una solución.

Palabras clave: inteligencia artificial, búsqueda mejor vecino, Juego de la vida, paso de estados refinado**,** camino aleatorio.

Abstract

In this articulate, it is developed and a well-known cellular automata implements as the Game of the life. He is defined their basic structure and it is programmed using technical of local search, to find a solution.

Keywords: Artifitial Intelligence, Neighbourhood search, Game of the life, *Random walk.*

Introducción

Con el objeto de estudiar el comportamiento de la búsqueda local, desarrollaremos dos versiones del juego de la vida. La primera, utiliza un paso de estado refinado (estado estable a estado estable) y la segunda aplicación utiliza un cambio de estados más relajado.

* Ingeniero de sistemas, magíster en Ingeniería de Sistemas de la Universidad Nacional de Colombia, candidato a doctor en Inteligencia Artificial, docente de la Universidad Distrital Francisco José de Caldas, adscrito a la Facultad Tecnológica, investigador del Artificial Intelligence Research Institute of Scientific Research Spanish Council (CSIC), Barcelona. Correo electrónico: nelson@iiia.csic.es.

** Ismel Brito, licenciado en Matemática Computacional de la Universidad de La Habana (Cuba), estudiante Doctorado en Inteligencia Artificial de la Universidad Politécnica de Cataluña, becario del Instituto de Investigaciones en Inteligencia Artificial.

Las implementaciones se desarrollaron, la primera, en el lenguaje *C* y, la segunda, en MATHLAB. Mediante Neighbourhood Search, utilizando distintas estrategias de selección aleatoria.

En la segunda sección definimos el problema mediante estados, luego, en la sección tercera, se describe la implementación del problema; en la cuarta sección representamos e implementamos el problema mediante el algoritmo Neighbourhood search, se muestran sus resultados experimentales, y, por último, se describe la implementación de dicho problema en paso de estados relajados.

1. Definición del problema

El Juego de la vida no es un juego típico de computadora. Es un *autómata celular*, y fue inventado por el matemático John Hourton Conway de Cambridge.

Consiste en una colección o tablero de las células que evoluciona iterativamente de acuerdo con un conjunto de reglas locales. Dependiendo de las condiciones iniciales, las células forman varios patrones a través del curso del juego.

Allí se entiende por:

Espacio: cuadrícula homogénea de *n* filas *x n* columnas.

Estados: son cuatro estados: *vivo* (negro) *muerto* (blanco)

Estado inicial: configuración aleatoria.

Vecindad: cuadrícula de 3 *x* 3 centrado en una celda. Incluye la celda del centro.

Regla de evolución:

- Cada célula con sólo dos vecinos vivos, mantiene su estado en la próxima iteración.
- Cada célula con cuatro o más vecinos muere, por superpoblación.
- Cada célula con menos de uno o cero vecinos muere, por soledad.
- Cada célula con dos o tres vecinos sobrevive (regla de multiplicación).

El objetivo de este ejercicio es encontrar en un tablero de dimensión *N* la configuración estable que maximice el número de células vivas.

2. Representación e implementación del Juego de la vida

2.1. Implementaciónn del Juego de la vida por paso de estados refinado

Modelo

El siguiente algoritmo representa el modelado del problema:

1. Se crea una grilla de *n x n* celdas.

2. Se siembran unas semillas aleatoriamente, en la grilla; se revisa si es un estado estable[1].

3. Si el estado es estable se analiza su función de costo, se aplica el algoritmo Neighbourhood Search y vuelve al paso 2. Si el estado no es estable regresa al paso 2, hasta completar cuatro semillas al cabo de los cuales, al no obtener el estado, regresa al paso 1.

2.2 Representación e implementación del problema del Juego de la vida mediante el algoritmo Neighbourhood search

- *Características*: la solución será aquella que presente el mayor número de celdas vivas y su estado sea estable.
- *Representación:* arreglo bidimensional con solución: *s* con [1,2…, n], filas y con [1,2…, n] columnas.

  S[i][j]= 1 si la celda está viva y 0 en otro caso.
- *Recorrido:* se coloca una semilla en cada celda hasta un número de máximo cuatro semillas.
- *Costo*: será la sumatoria de las celdas vivas siempre y cuando el estado sea estable.
- *Funciones:* Algoritmo Neighbourhood Search

  Iter := 0;

X := punto inicial arbitrario;

Mejor solución := X;

Mejor coste := f(x);

Mientras iter < maxIter hacer

Iter := iter +1;

X :=selección(N(X));

Si f(x) < mejorCoste *entonces*

Mejorsoluccion := X;

Mejorcoste := f(x);

Fmientras

Devuelve solución.

2.3 Resultados experimentales

En la Tabla 1 se presentan los resultados de la evaluación del algoritmo utilizando distintas estrategias de selección.

Tabla 1. Resultados de desempeño

Tamaño de grilla	Forma elegir sucesor	Número máximo ciclos	Densidad
10 x 10	*Randon Walk*	50	35
10 x 10	*Randon Walk*	100	40
10 X 10	*Randon Walk*	150	42
10 x 10	*Randon Walk*	350	42
15 x 15	*Randon Walk*	50	37
20 x 20	*Randon Walk*	50	53

[1] Entendiéndose por tal un estado en el cual, al evolucionar, mantenga su misma configuración.

3. Implementación del Juego de la vida por paso de estados relajados

3.1 Función principal: BLocal (Véase Tabla 2)

Tabla 2. Algoritmo de estados relajados

```
valorOpt = 0;
mejorConf = completamente vacío, todas las células muertas.
full = N * N;
While (No_Condicion_Parada),
Conf = IniciarConfiguración(N);
t = 0;
while (t < intentos),
Conf = IrAVecino(Conf);
t = t + 1;
end;
miValor = número de células vivas + full * esEstable(Conf);
if (valorOpt < miValor),
valorOpt = miValor;
mejorConf = Conf;
end;
end;
```

3.2 Función IrAVecino

1. Aleatoriamente se selecciona en la iteración *t*, si se cambia el estado de una célula que, estando viva, debería estar viva según las reglas, o de una célula que, estando muerta, debería estar viva. Es decir, si por esta vez el algoritmo se ocupa, sólo me ocupo del conjunto de células vivas

que deben morir, o de las que están muertas que deben vivir, según la regla. Dado el caso, se escoge el conjunto *CEL* de células que cumplen ese requisito.

2. Dado el conjunto CEL, se selecciona aleatoriamente una célula de él y se le cambia el estado.

3.3 Descripción

El algoritmo se basa en el algoritmo de búsqueda local por vecinos (Neighborhood Search), entendiendo por vecinos, en este caso, una configuración donde sólo una célula ha cambiado su estado. La función "iniciar configuración", dada la dimensión del tablero, selecciona aleatoriamente el estado de cada célula en el tablero.

La función objetivo, la cual define qué solución o configuración es mejor que otra, pesa la estabilidad por encima del número de células vivas.

3.4 Resultados

Obsérvese que el criterio de parada aquí es el tiempo, por lo que estos resultados son una cota máxima de los resultados que se pudieran obtener. Por ejemplo, es posible que en otra ejecución con menos tiempo se consiga el mismo valor óptimo o mejor. (Véase Tabla 3)

Tabla 3. Resultados de desempeño

n	Valor Optimo	Intentos	Número de ciclos	Tiempo (segundos)
4	8	20	146	2.5
5	16	50	1947	87
6	17	20	160140	600
7	27	30	202784	1200
10	-	-	-	-
15	56	50	13072	3600

Donde,

Intentos: es el número de veces que se intenta mejorar la configuración que resulta después de aplicar la función “IniciarConfiguración”.

Ciclos: es el número de reinicios o llamados a la función “IniciarConfiguración”.

Para el problema $n = 15$ en vista a que los resultados son tan pobres comparados con los reportados en la literatura (para $n = 15$, el valor óptimo es 119) se realizaron los siguientes reajustes:

- Sembrado aleatoriamente en la función “IniciarConfiguración” de *cluster* de cuatro células vivas, en forma de cuadrado, que el mínimo conjunto de células estables.
- Redefinir la función “IrAVecino”, para tratar de mejorar la configuración por *hill-climbing* utilizando uno de los siguientes métodos:

    1. Buscar el primer cambio del estado de una célula que provoque una mejora en la evaluación de la función objetivo. Como resultado, los experimentos para estos cambios demostraron, como era de esperarse, que el tablero converge al tablero completamente vivo, donde todas las células están vivas.

    2. Escoger aleatoriamente dos pares de células vecinas con estados distintos, que, al intercambiarle los estados, provoque una mejora en la evaluación de la función objetivo. Los resultados mostraron que no es frecuente (de hecho es más improbable que el punto 1) encontrar una circunstancia como la que necesita para aplicarla, por lo que no hay grandes cambios entre la configuración generada al principio del algoritmo y la que va evolucionando en el tiempo.

La “no tan mala calidad” de los resultados obtenidos para $n = 15$ (aproximadamente 50% cercano al óptimo) fueron conseguidos gracias al sembrado de *cluster*.

3.5 Características del PC (*Personal Computer*):

- Procesador: Intel(R) Pentium(R) 4
- CPU: 2.53GHz
- RAM: 512 KB
- Sistema operativo: Windows ME
- *Software*: MATLAB

Conclusiones

Si se hace una búsqueda por paso refinado, no es muy importante el número de ciclos o iteraciones, basta con un número medio de ciclos para saber, más o menos, el número de células que obtendrá en próximas generaciones.

Los resultados para problemas de tamaño pequeño, el algoritmo de búsqueda local es eficiente, sin embargo, a medida que aumenta la dimensión del problema, la búsqueda local guiada para este problema en especial no parece buena idea, a menos que se utilicen conceptos como el sembrado de *cluster* o técnicas mucho mas eficientes de selección de vecino.

Las razones por las que este algoritmo no se comporta bien para instancias de este problema de dimensión *grande* son, según consideramos:

1. El grado de las restricciones, cada célula está vinculada a ocho células más.
2. La función objetivo tiende, a pesar de estar ponderada por la estabilidad, a favorecer el número de células vivas.

Referencias bibliográficas

[1] Larrosa J, Meseguer P. Algorithms for constraint satisfaction problem, inteligencia artificial. Revista Iberoamericana de Inteligencia Artificial 2003 (20).

[2] Russel S, Norving P. Artificial Intelligence a Modern Approach. Prentice may; 1995.