CONTROL Y SUPERVISIÓN ALGORÍTMICA PARA LA PREVENCIÓN DE COLISIONES ENTRE ABEJAS ROBÓTICAS

Hamilton Smith Gómez Osorio
Universidad EAFIT
Colombia
hsgomezo@eafit.edu.co

Santiago Isaza Cadavid
Universidad EAFIT
Colombia
sisazac@eafit.edu.co

Mauricio Toro
Universidad EAFIT
Colombia
mtorobe@eafit.edu.co

RESUMEN

Las abejas robóticas son el futuro de la agricultura en su proceso de producción por lo que se hace necesario tener un control y supervisión de la ubicación de estas para evitar colisiones y así lograr el funcionamiento óptimo de las mismas. Estos problemas de colisiones también se presentan en la industria como videojuegos, finanzas entre otros.

Dado a que es importante manejar este situación hemos implementado una estructura de datos basada en matrices tridimensionales las cuales detectas esta colisiones y hemos obtenido muy buenos resultados con unos tiempos de ejecución muy bajos y óptimos procesos de manejos en la información. Con esto concluimos que se pueden analizar los problemas de gran magnitud e impacto en el mercado si se tiene un buen manejo de los problemas.

Palabras clave: Sorting and Searching, Data structures, 3D matrix, Data Management.

Sistema de clasificación ACM:

CCS → Theory of computation → Design and analysis of algorithms → Data structures design analysis→ Sorting and searching.

INTRODUCCIÓN

Frente a la disminución de la población de abejas que se presenta actualmente y la importancia de estas en el proceso de polinización del sector agrícola se ve un riesgo futuro en los cultivo, por lo que es necesario encontrar una solución a dicho problema. Así nace la idea de crear abejas robóticas las cuales ayuden en este proceso y, para supervisar y controlar su funcionamiento, implementar una estructura de datos en la que por medio de un algoritmo se prevenga la colisión de las mismas.

2. PROBLEMA

Las abejas robóticas implementadas en la agricultura para el proceso de polinización pueden colisionar si están a menos de 100 metros de distancia de otras abejas por lo que es importante solucionar dicho problema para tener un funcionamiento óptimo y una mejora en los procesos.

3. TRABAJOS RELACIONADOS

Encontramos trabajos relacionados con la radiofrecuencia, algunos utilizados en el área de la computación gráfica, otros utilizados en el sector de la privacidad y el manejo de datos, todos relacionados con la colisión, en los que se intentaban prevenir o aprovecharse de ella para solucionar problemas mayores.

3.1 Sistema de identificación por radiofrecuencia (RFID)[1]

Cuando hay varias etiquetas (usadas para almacenar la información) y lectores (lee, cambia y verifica la información de la etiqueta) en el mismo canal y transmisión de señal, se genera un problema de colisión debido a las interferencias mutuas entre las etiquetas y los lectores.

Solución: Un algoritmo anticolisión basado en una matriz y esquema de codificación.

Se establecen los datos decodificados en una matriz y luego el lector se encarga de procesar dichos datos por filas, se analizan por parejas y al encontrar una colisión se reemplaza por valor uno (1), en caso contrario se establece un cero (0). Luego de reemplazar las filas, se extraen las colisiones y se siguen analizando las siguientes filas hasta terminar

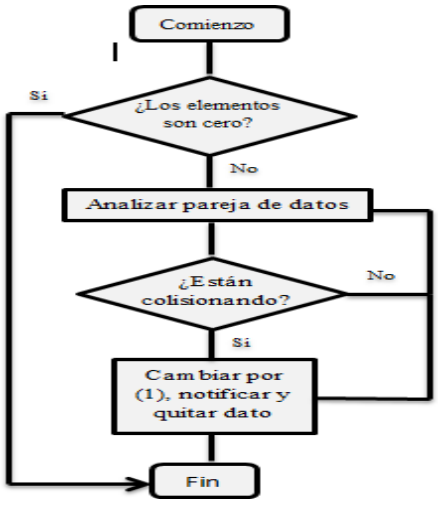

Figura 1: diagrama de flujo RFID

3.2 Volúmenes acotados para detectar colisiones 2

Muy utilizado en la computación gráfica a la hora de realizar videojuegos. Se basa en utilizar formas geométricas básicas encapsulando figuras complejas y utilizando la intersección de estas para determinar cuándo alguna colisiona con la otra; por medio de estas figuras se tiene control de los objetos cuando hay movimientos o cambios de perspectiva, de los más utilizados está AABB y OBB.

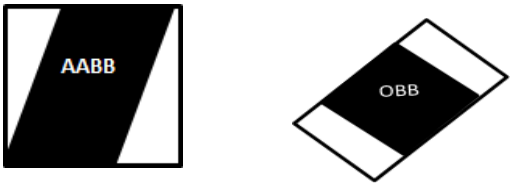

Figura 2: tipos de figuras de acotamiento

3.3 Árbol Octree 3

Para analizar colisiones entre múltiples elementos en un plano se utiliza este método el cual busca subdividir es espacio delimitado en rectángulos de igual medida y luego volver a dividir este último hasta tener una zona relativamente menor en la que sea más fácil y rápido la comparación entre las posiciones de un objeto y otro, teniendo en cuenta su volumen y sus coordenadas.

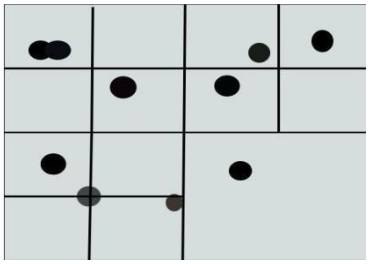

Figura 3: subdivisiones del árbol Octree

3.4 Estructura de datos Spatial Hashing 4

Consiste en dividir una zona en cubos con una medida específica, teniendo en cuenta el máximo y mínimo valor de coordenadas, y luego organizar los objetos que estén inscritos dentro de estos en una lista de referencia al índice de la caja en la que se encuentra.

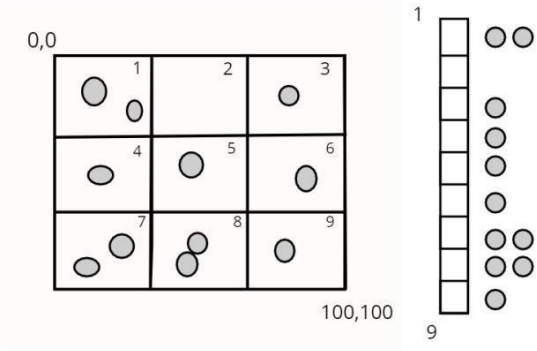

Figura 4: subdivisión y clasificación por cajas

4. Estructura a implementar: Matriz Tridimensional

4.1 Operaciones de la estructura de datos

- **Crear las celdas:** lo usamos para definir las celdas en las que se encontrarán las abejas. En cada cubo se encontrarán las abejas almacenadas en una pila, dichas abejas al estar en el mismo cubo estarían en riesgo de colisión.
- **Detectar colisiones:** dadas las coordenadas de una abeja se agregará a determinada pila con aquellas que están en riesgo de colisión.
- **Abejas adyacentes:** determina cuáles abejas se encuentran en cuadros adyacentes y terminan estando también en riesgo de colisión

4.2 Criterios de diseño de la estructura de datos

Esta estructura es bastante eficiente en comparación con otras presentadas anteriormente, ya que desde la construcción del algoritmo podemos definir el tamaño y la cantidad de celdas dependiendo del número de objetos que puedan colisionar. Así no necesita analizar ninguna posición de una abeja respecto a otra, pues el hecho de pertenecer a una misma celda ya representa un riesgo de colisión. Por lo que con esta estructura de datos se logra ahorrar espacio de memoria (respecto a una comparación entre todos los elementos) evitando cualquier tipo de clasificación y consulta, además de hacer al algoritmo más rápido.

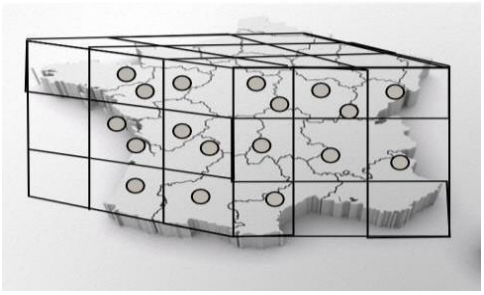

Figura 5: mapa acotado por cajas

Planteamiento de implementación

En el lenguaje de programación Java se implementó una solución por medio de la formación de una matriz tridimensional. Que representa la división de un espacio en celdas. El volumen de estos cubos está determinado por una diagonal de 100 metros y 57 metros en sus aristas, posteriormente las abejas se clasifican en pilas dependiendo del cubo en que se encuentren para así retornar aquellas que se encuentran en el mismo cubo y reconocer aquellas en riesgo de colisión. Además, se tienen en cuenta los cubos adyacentes de las abejas que se encuentren solas en una caja pues están podrían colisionar con alguna que esté a su alrededor.

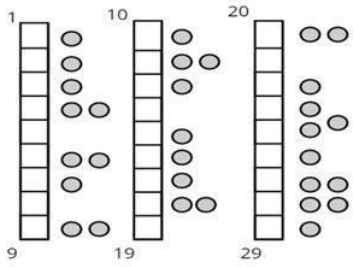

Figura 6: organización de abejas según su posición

4.3 Análisis de complejidad

Método de la estructura	**Complejidad**
areaDeUbicacion()	**O(n)**
abejasAdyacentes()	**O(m)**
detectarColisiones()	**O(n + m)**

Tabla 1: Complejidad de los diferentes métodos

4.4 Tiempos de ejecución

Método	Tiempo promedio(ms)						
	Número de abejas						
	4	10	100	1.000	10.000	100.000	1'000.000
areaDeUbicacion()	0,6	3,6	3,6	11,6	28,4	275,8	3.315,2
detectarColisiones()	0	4,2	11	8,2	37,8	131,8	673,6
guardarArchivo()	1,6	3,2	7,4	5	112,8	770	6.060,2

Tabla 2: tiempos de ejecución (ms) para cada tamaño del problema

4.5 Memoria

Número de abejas	Memoria
10	1
100	2
1000	4
10000	15
100000	80
1000000	245

Tabla 3: Tabla de memoria gastada en el problema medido en MB

5.5 Análisis de resultados

Detectar colisiones ArrayList	Detectar colisiones LinkedList	Detectar colisiones Matriz 3D
O(n^3)	O(n^2)	O(n + m)

Tabla 4: Cuadro comparativo de complejidades

Los resultados obtenidos, en comparación con las soluciones antes trabajadas muestran una verdadera mejoría puesto que se logró mejorar la ejecución en el tiempo del algoritmo. Además de que la mejor estructura de datos a usar depende del problema que se trabaja.

Respecto al gasto de memoria, y también al tiempo, podemos ver que, a mayor número de abejas, el problema toma más tiempo pues es mayor el análisis que se debe hacer.

Número de abejas	Resultados
4	4
10	4
100	24
1000	279
10000	9546
100000	99184
1000000	1000000

Tabla 5: Resultados obtenidos para cada tamaño del problema

5. Conclusiones

A lo largo del reporte hablamos de diferentes posibles soluciones al problema de colisión, desde octree a pequeñas divisiones usando figuradas acotadas. Todas ellas viables, pero la solución realizada en este trabajo significa también una solución posible y cómoda, pues permite que se analicen y obtengan aquellas abejas que están en riesgo de colisión usando un método práctico como las matrices tridimensionales

y permitiendo resultados satisfactorios de rápido cálculo y manejable gasto de memoria.

5.1 Trabajos futuros

Nos gustaría mejorar la implementación de la búsqueda de colisiones con los cuadrados adyacentes ya que consideramos que se puede hacer de manera más eficiente. Además de mejorar los tipos de datos a utilizar para así ahorrar memoria.

AGRADECIMIENTOS

Principalmente agradecemos al proyecto de educación nacional Ser Pilo Paga del Gobierno de Colombia y a la beca Fundación de la Universidad EAFIT por apoyar económicamente nuestro proceso formativo.

Agradecemos también por su asesoría en el proceso de aprendizaje a Daniel Mesa, Monitor del curso de Estructura de Datos y Algoritmos I en la Universidad EAFIT, por sus aportes en clase y su orientación en la realización de este proyecto.

REFERENCIAS

1]. Liu, B. and Su, X. An Anti-Collision Algorithm for RFID Based on an Array and Encoding Scheme. Information, 2018, 2078-2489. Accessed August 25, 2018 from Universidad EAFIT: https://bit.ly/2PEzPhx

[2]. Dinas, S. and Bañón J. M. A literature review pf bounding volumes hierarchy focused on collision detection. Ingeniería Competitiva, 2015, 49-62. Accessed August 25, 2018, from Universidad EAFIT: https://bit.ly/2BMI9sD

[3]. Nevala, E. Introduction to Octrees. GameDev.net, 2018.
Accessed September 23, 2018: https://bit.ly/2pxAzJa

[4]. Spatial hashing implementation for fast 2D collisions. The mind of Conkerjo, 2013. Accessed September 23, 2018: https://bit.ly/2xHprNK

[4]. How to efficiently remove duplicate collision pairs in spatial hash grid? Stack Overflow, 2015. Accessed September 23, 2018: https://bit.ly/2O2PPvG