ALGORITHM TO PREVENT COLLISIONS BETWEEN ROBOTIC BEES

Juliana Lalinde
Universidad EAFIT
Colombia
jlalindev@eafit.edu.co

Isabel Urrego
Universidad EAFIT
Colombia
iurregog@eafit.edu.co

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

ABSTRACT
Bees are fundamental for the proper development of an ecosystem. Their population decline is an alarming phenomenon we are currently facing, and it has brought popularity to the idea of developing robotic bees, that can fulfill their functions. A key factor in the execution of this project is constant and effective monitoring that ensures effective distribution.

This distribution determines the benefit that can be obtain, but it depends of a proper execution: there can not be collision between bees that move in the same three-dimensional space. The need to be alert of the risk of possible collisions is the reason for the development of an algorithm that analyses possible accidents. As a base case for the development of the algorithm we used the city of Bello, where bees are in risk of colliding if they are less than one hundred meters apart.

Keywords
Collisions, Hashing, Hash Table, Hash Functions, Static Hashing

<u>ACM CLASSIFICATION Keywords</u>
CCS → Theory of computation → Design and analysis of algorithms → Streaming, sublinear and near linear time algorithms → Bloom filters and hashing

1. INTRODUCTION

Bees have been catalogized as one of the most important animals for ecosystems. They pollinate different plants, which contributes to the reproduction of different species and the production of aliment vital for living beings. There are cases in which an entire species reproduction depends on the behavior of bees.

Nowadays, various factors have led to their population decrease, putting at risk many species and ecosystems. This has alerted specialists and has brought the development of ideas to solve the problem. One key idea is the invention and use of robotic bees.

These robotic bees would complete the same functions of real bees, becoming a help to ensure the constant reproduction of different plants. Achieving this present a challenge: bees need to be properly programmed and evaluated to be actioned, a process that brings need factors to be considered.

One of the most important factors to take into account is the risk of collisions between the bees, and the fact that in order to optimize their functions bees need to have a significant distance while operating [1]

2. PROBLEM

This projects objective is to implement a time efficient algorithm that helps with one of the key factors in the development of robotic bees. The factor that is going to be taken as a base is the risk of collisions.

Knowing this, the main objective is to design a program that allows the detection and notification of bees that are less than one hundred meters apart. The input about the bees will be their coordinates in a three-dimensional space.

3. RELATED WORK

3.1 Spatial Hashing

This algorithm is used to detect collisions between objects on both two- and three-dimensional spaces with the help of a data structure known as a hash table. Having a set of elements, hash tables associate proper keys of each element to a value. Additionally, they have a hash function that return an index to be able to access an element in the table [2].

When two elements are saved in the same index in the table then there is a collision, and this is exactly the case that relates hash tables with our bee collision problem.

In Spatial Hashing, the table works as a subdivision of the space and if there is a collision then it means that there are at least two elements in the same subdivision.

This algorithm helps to avoid the use of a brute force solution that would have to compare the distance between every pair of elements that are part of the set given. Comparison is only made between subdivisions that contain more than one element, because the number of elements in the same location indicates a possible collision. Through this process the algorithm guaranties a greater efficiency in the same execution time [3].

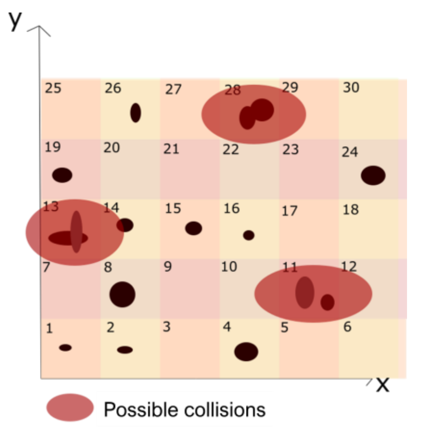

Figure 1: Example of Spatial Hashing in two dimensions

3.2 AABB trees

This method is used to verify if there are possible collisions between different objects, with the help of AABBs.

AABBs are basically boxes that contain objects, this way the axes between different objects align. Instead of taking an object with all its regularities

AABBs son básicamente cajas que contienen objetos y de esta manera los ejes entre diferentes objetos se alinean. Es decir, en vez de tomar un objeto con todas sus regularidades, se toma el objeto como algo que está contenido entre ciertas coordenadas en x, ciertas en y y otro rango en z si se habla de tres dimensiones. Esto es para disminuir notoriamente el número de comparaciones que se deben hacer para identificar una colisión, pues usando AABBs solo se tendría que tener en cuenta un rango en cada eje. [4]

No obstante, comparar cada AABB cuando son numerosos elementos podría resultar muy poco efectivo y es por esto que se acude a los árboles. Estas estructuras de datos se forman asociando elementos (hojas) y juntándolos en una misma raíz que en este caso sería también un AABB. La ventaja está en el momento de revisar las3 posibles colisiones, pues simplemente se miraría si un objeto tiene riesgo con una raíz y solamente si la respuesta es sí, se procede a revisar cada "hoja" de la raíz para detectar dónde están las colisiones. Finalmente, no se estarían comparando todos los AABBs entre ellos, sino haciendo un número más reducido de verificaciones basadas en el árbol que se armó por medio de asociaciones de AABBs. [5]

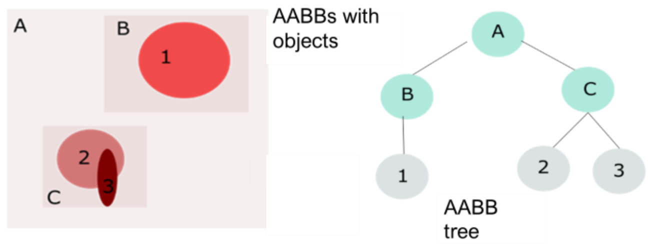

Figure 2: Example of the use or AABB trees

3.3 R-tree

It was formulated by Antonin Guttman in 1984. A tree is a data structure that simulates the shape of a real tree, an empty tree is used to build a R-tree, into which the elements that need to be saved are added.

In the R-tree structure the leaves correspond to the smallest rectangles that contain the geometrical object that need to be explored, an element is inserted on an R-tee with the following process: the object moves down the tree, from the root to the leaves, checking in which of the following rectangles belongs the object. This is done until the element reaches one of the leaves, where it is added. To search for a specific object the same process is followed, except that in the end it returns the leave that contains the searched objects. This procedure is used to identify possible collisions between objects, if by the end of the search there are no objects on the leave you reach then there are no collisions [6].

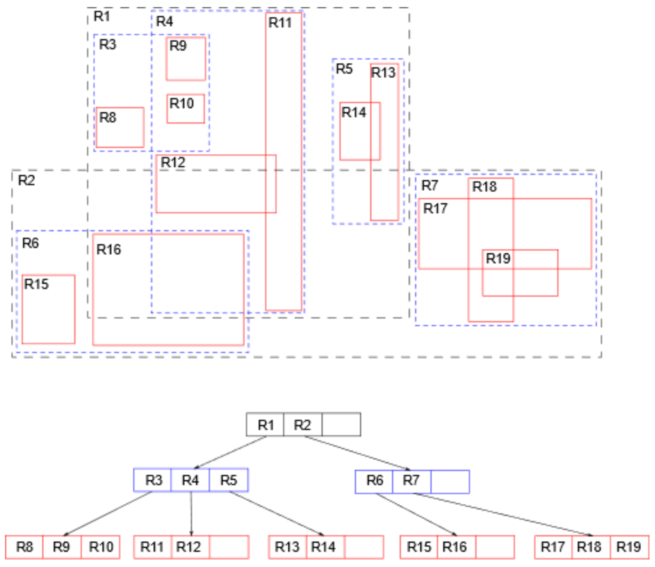

Figure 3: Example of the use of R-trees

3.4 Octree

An Octree is a hierarchical data structure that divides space, it is used for objects located in a three-dimensions. It works by dividing the space in eight cubes, which are also divided into eight cubes individually. This process of division is applied until a designated cube size is reached, if we are working with collisions we would want to reach the smallest object size or the minimal distance between two objects to avoid a collision. The objects that are contained within the same cube are at risk of collision and must be evaluated [7].

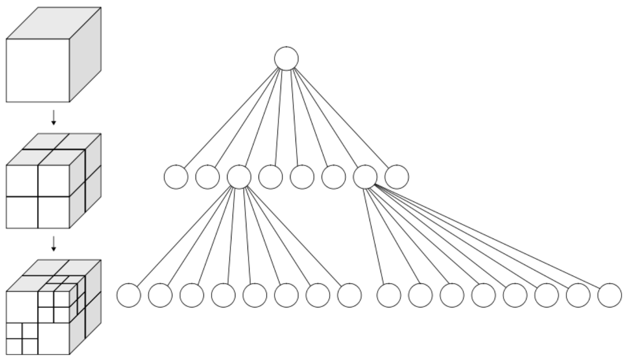
Figure 4: Example of the use of Octrees

4. Spatial Hashing in three-dimensions

The data structure that we implemented to solve the problem is Spatial Hashing, based on Hash Tables. It works by dividing the space into a cube grid, in which each cube has a 100x100x100 size, and associating each of the bees into a cube based on their coordinates. The different cubes are associated with an index, that helps identify the elements that are within its range. Every element, in this case the coordinates of a bee, that is added to the grid is referred to as a key and must go through a Hash Function that return a number that matches the index of one of the cubes. Using this division, we can then quickly affirm that all the elements that share the same key are in a collision risk, and we can also lower the number of comparisons between objects that are in different cubes using a simple strategy: we only need to compare the distance between bees that are in adjacent cubes.

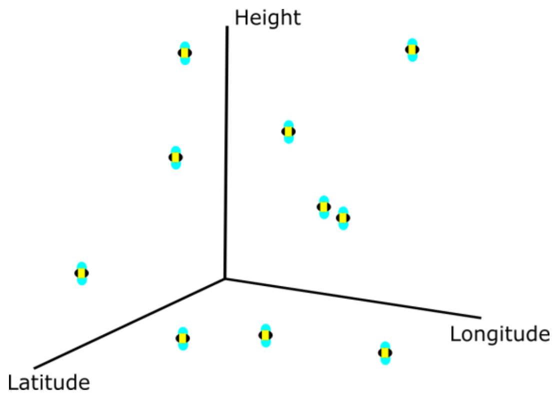

Figure 4: Example of the interpretation of the input

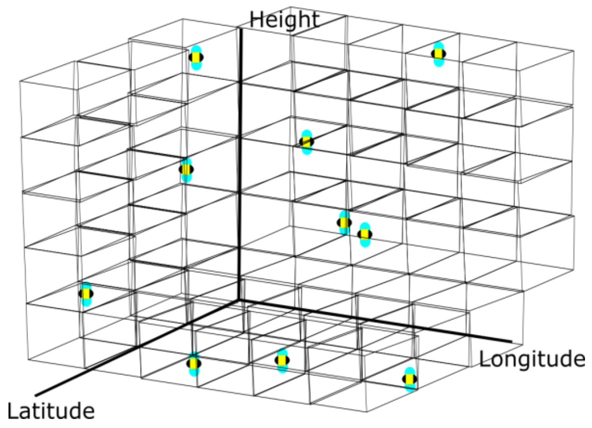

Figure 5: Bees classified by their coordinates using Spatial Hashing

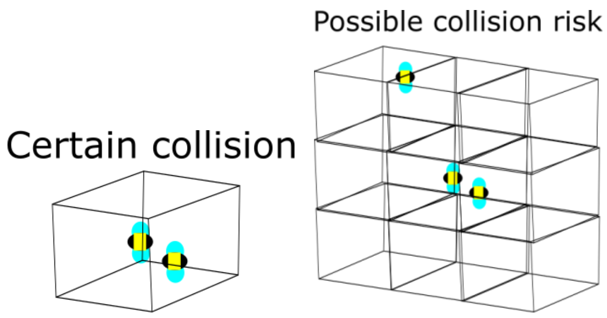

Figure 6: Example of the possible collisions between different cubes

4.1 Operations of the data structure

The main operations used with Spatial Hashing are [8]:

*The images of the operations show a two-dimensional plane (the bees have two coordinates). By the real implementation of the algorithm we will use a Euclid Plane, which adds another dimension, but the operations will do the same as shown.

- **createHashTable:** used to create a new hash table. Each grid will represent a cube which has different keys representing the coordinates of the bees and a index associated. This function creates a table of the given size and the grids will be the divisions of the plane. Additionally. the creation of a new Hash Table includes a Hash Function which pairs a key (bee) with a grid (cube).

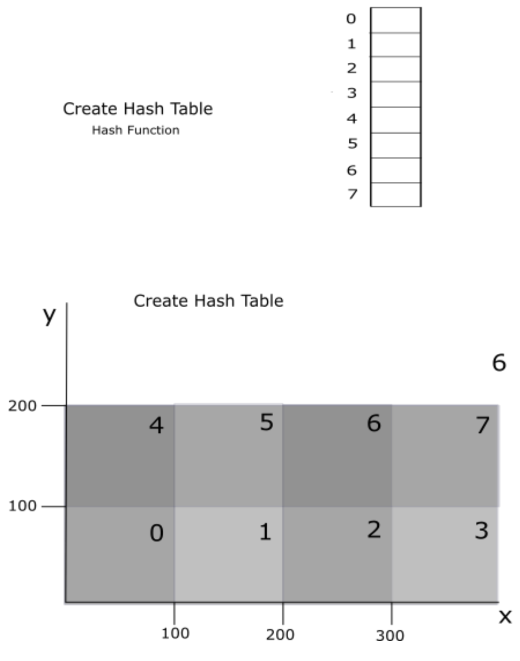

Figure 7: createHashTable

- **hashSearch:** used to search a key in hash table. Given the coordinates of a bee, this operation uses the hash function to know which grid should contain this element and searches if the element is effectively there.

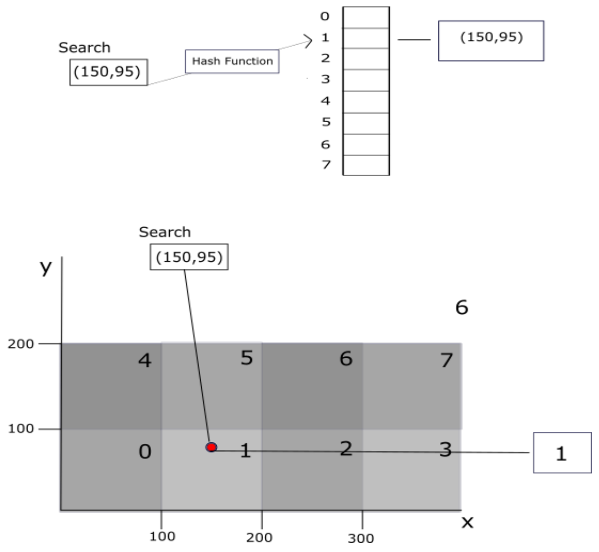

Figure 8: hashSearch

- **Search:** says which keys (bees) belong to a specific index (grid).

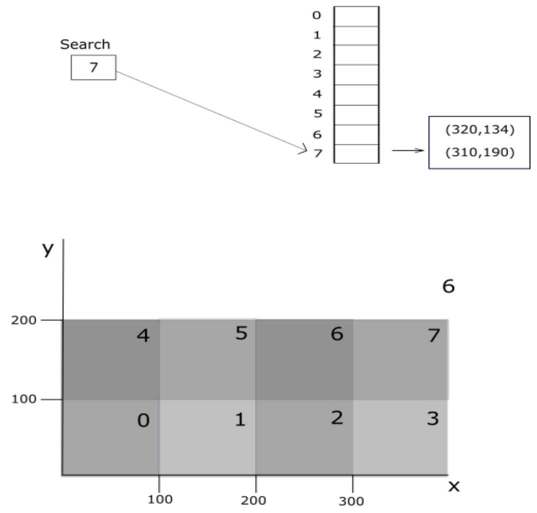

Figure 9: Search

- **hashInsert:** inserts a new key. Given the coordinates of a bee, this operation uses the hash function to add the element to the cube where the coordinates are included.

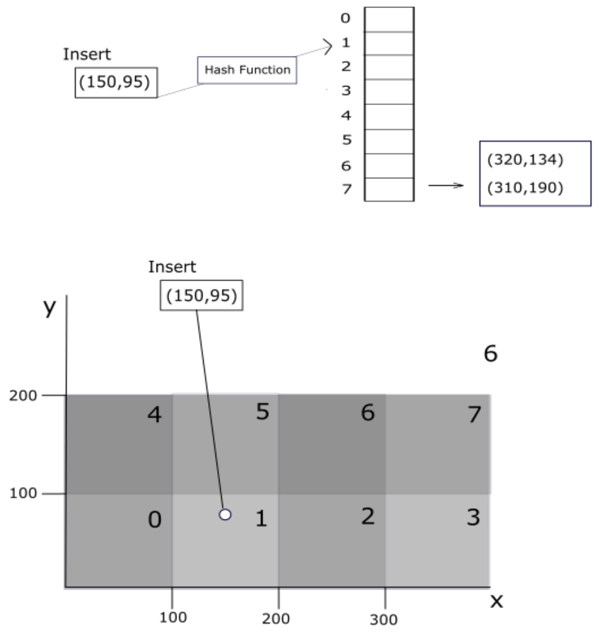

Figure 10: hashInsert

- **hashDelete:** deletes a key from hash table. It uses the hash function to know from which grid the element should be deleted and proceeds to remove it from the table.

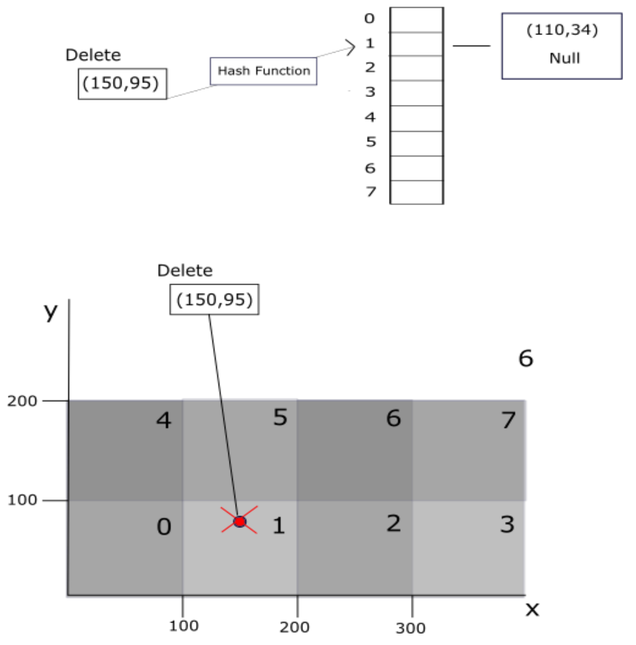

Figure 11: hashDelete

- **deleteHashTable:** Deletes the whole table

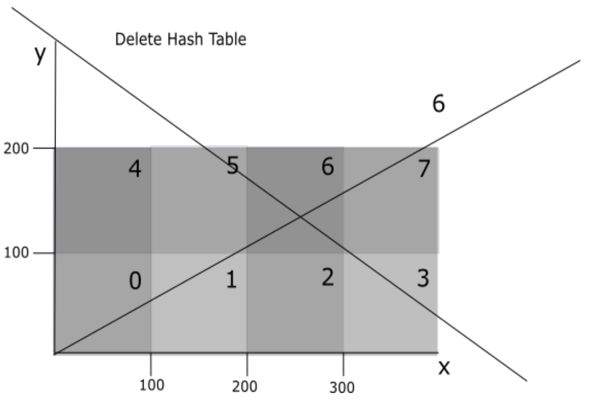

Figure 12: deleteHashTable

4.2 Design criteria of the data structure

While looking for the solution for the problem we put time efficiency as the main criteria. We read about the different data structures, always thinking how we could reduce the number of comparisons. Our first idea was to separate the bees into different groups defined by their coordinates. In that order, we would have different divisions of the Euclidean space and the comparisons needed would be limited by the number of bees sharing the same group. That is how we realized that we could use hash tables to solve the problem.

Spatial Hashing allowed us to divide the space equally and place the bees considering their three coordinates, it also lowered the number of comparisons that needed to be made. We also decided to work with Lists within the cubes to guaranty that any number of bees could be contained without a limit. This Lists also allow a more dynamic execution for the comparisons that still need to be executed,

REFERENCES

1. Klein, K. Robotic bee could help pollinate crops as real bees decline. *NewScientist*. Recuperado el 26 de Agosto de 2018: https://www.newscientist.com/article/2120832-robotic-bee-could-help-pollinate-crops-as-real-bees-decline/

2. Karumanchi, N. Data Structures and Algorithms Made Easy in JAVA. CareerMonk Publications, Bombay, 2018.

3. MacDonald, T. Spatial Hashing. Gamedev.net. Recuperado el 25 de Agosto de 2018: https://www.gamedev.net/articles/programming/general-and-gameplay-programming/spatial-hashing-r2697/

4. James. Introductory Guide to AABB Tree Collision Detection. AZURE FROM THE TRENCHES. Recuperado el 25 de Agosto de 2018: https://www.azurefromthetrenches.com/introductory-guide-to-aabb-tree-collision-detection/

5. Karumanchi, N. Data Structures and Algorithms Made Easy in JAVA. CareerMonk Publications, Bombay, 2018.

6. Antonin Guttman. 1984. R-trees: a dynamic index structure for spatial searching. In Proceedings of the 1984 ACM SIGMOD international conference on Management of data (SIGMOD '84). ACM, New York, NY, USA, 47-57. DOI=http://ezproxy.eafit.edu.co:2079/10.1145/602259.602266

7. Jane Wilhelms and Allen Van Gelder. 1992. Octrees for faster isosurface generation. ACM Trans. Graph. 11, 3 (July 1992), 201-227. DOI=http://ezproxy.eafit.edu.co:2079/10.1145/130881.130882

8. Narasimha Karumanchi. 2018. *Data structures and algorithms made easy in Java: data structure and algorithmic puzzles*, Madinaguda, Hyerabad: CareerMonk Publications.