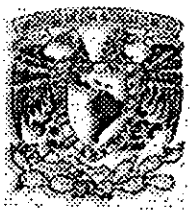


UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DIVISIÓN DE ESTUDIOS DE POSGRADO

T E S I S

ANÁLISIS DE MODELO BICOLOR DE AGREGACIÓN DE DIFUSIÓN LIMITADA

PARA OBTENER EL GRADO DE MAESTRO EN INGENIERÍA (INFORMÁTICA) PRESENTA:

SABÁS FLORES ASCENCIO

TUTOR: DR VLADIMIR TCHIJOV
COTUTOR: DRA. SUEMI RODRÍGUEZ ROMO

CIUDAD UNIVERSITARIA, MÉXICO D.F. MAYO DE 2000

Í N D I C E.

CAPÍTULO I INTRODUCCIÓN

1.1- OBJETIVO:

En el presente trabajo se desarrollara un nuevo modelo computacional, en tres dimensiones, de agregación limitada por difusión. Se tomará como base los modelos existentes propuestos hasta ahora. De la misma forma, se pretende validar algunos de los resultados obtenidos en los estudios de los modelos propuestos en la década de los 80's.

1.2- ANTECEDENTES:

Durante los años 80's, se estudiaron algunos procesos cuyo crecimiento es aleatorio. Los ejemplos de este tipo de procesos es la contaminación atmosférica, el desarrollo de coloides, coagulación de aerosoles, los procesos industriales de polioles para generar hules espuma, entre muchos otros **[1]**. Desde esos años a la fecha se han propuesto diferentes modelos que intentan simular ese tipo de comportamientos, tal es el caso del modelo propuesto por T.A Witten Jr. y L . M Sander **[2]**, y desarrollado por P. Meakin **[3]**, entre otros modelos que básicamente son una variación de los que se describen en las referencias mencionadas. Desde entonces, la computadora era una herramienta muy poderosa para disminuir los tiempos de cálculo y poder generar múltiples estudios estadísticos. Para el tipo de computadoras que se utilizaron en la década de los ochenta´s, los tiempos requeridos de CPU eran demasiado grandes; debido a la gran cantidad de cálculos matemáticos que debían realizarse. Por ejemplo, P. Meakin menciona que la formación de una estructura de tipo fractal, con crecimiento aleatorio y

por agregación limitada por difusión, con un total de 10,000 partículas formando al fractal requirió 20 horas de tiempo de CPU en un equipo "Digital Equipment Corp. VAX-11/780".

Por otro lado, los resultados a los que se llegaron y se han llegado desde entonces han sido publicados en revistas científicas especializadas. Sin embargo, las técnicas de simulación así como los algoritmos utilizados, casi siempre han sido una "caja negra" para los lectores, conformándose con apreciar los resultados, debido en gran medida a que el interés de los resultados, se fundamentaba en el significado físico, más que en las técnicas informáticas utilizadas para los procesos de simulación.

Uno de los intereses principales de este trabajo, es abordar los problemas informáticos a los que es necesario enfrentarse para la simulación de los procesos que se mencionan, y en base a ello, desarrollar el nuevo modelo "*agregación limitada por difusión bicolor*", así como validar los resultados de los experimentos que se han publicado en revistas científicas especializadas.

Un reto más a vencer, es la generación de los resultados, pero con algoritmos que garanticen un tiempo de CPU sensiblemente más corto, comparado con el tiempo que han requerido los estudios que a la fecha se han realizado, y llegado el caso proponer nuevos modelos con resultados que motiven su investigación más profunda.

Los trabajos que a la fecha se han realizado en cuanto a la "*agregación limitada por difusión*", han buscado básicamente encontrar la relación que existe entre la *dimensión fractal* (vea apéndice A) de la estructura que se generan y la dimensión euclidiana tradicional. También se busca encontrar algunos patrones comunes para diferentes dimensiones; es decir, dos, tres, cuatro y más dimensiones. Solo se consideran los casos

de dos y tres dimensiones por tener un significado físico real y se han hecho a un lado resultados de mayores dimensiones por carecer de significado físico real.

Los agregados estudiados por Witten y Sander**[2]**, se formaron cuando un vapor de metal, producido por calentamiento de un filamento plateado, se condenso, entonces partículas de metal con un radio aproximado de 40 Amstrongs se formaron cerca del filamento. Las partículas acumuladas en una delgada capa esférica, *bola de vapor*, de aproximadamente un centímetro de radio, se desplazaron a un porta-objetos de un microscopio electrónico, sucede que los agregados encontrados en el porta-objetos eran del orden de 10^5 partículas de metal en una masa de baja densidad.

Las partículas que aparecieron, formaron agregados, y este proceso resultó ser irreversible. Aun más, los agregados mantuvieron su forma características incluso después de deslizarse sobre diferentes medios gaseosos.

El modelo teórico y computacional de procesos de esta naturaleza lo propusieron conjuntamente Witten y Sander**[2]**. En este modelo inicial, hay una partícula considerada como semilla en el origen de un reticulado o enrejado. Una partícula se agrega en algún lugar lejano de la semilla y camina aleatoriamente hasta que visita un sitio adyacente a la semilla. Entonces la partícula caminante para a formar parte del grupo. Una nueva partícula se introduce en un punto de distancia aleatorio y camina, también aleatoriamente, hasta que se une al grupo. Si una partícula, durante su caminata aleatoria, toca los límites previamente fijados del enrejado, es removida y otra partícula nueva se introduce. El proceso se repite hasta formar estructuras de tamaño suficientemente grande para obtener datos estadísticos.

Un modelo similar fue estudiado por H.B Rosenstock and C. L Marquardt **[4]**.

Un primer esquema que ilustre el modelo utilizado por Witten y Sander**[2]**, puede ser el que ahora se presenta, fig. 1.1. En el esquema se pueden apreciar las caminatas al azar, que toma una partícula, hasta que visita un sitio adyacente a la semilla puesta al centro del enrejado. Cuando esto ha sucedido, una nueva partícula *nace* a una cierta distancia de la semilla y el proceso se repite. Cuando la partícula caminante toca los límite del enrejado es eliminada e introducida una nueva, como lo sugiere la figura.

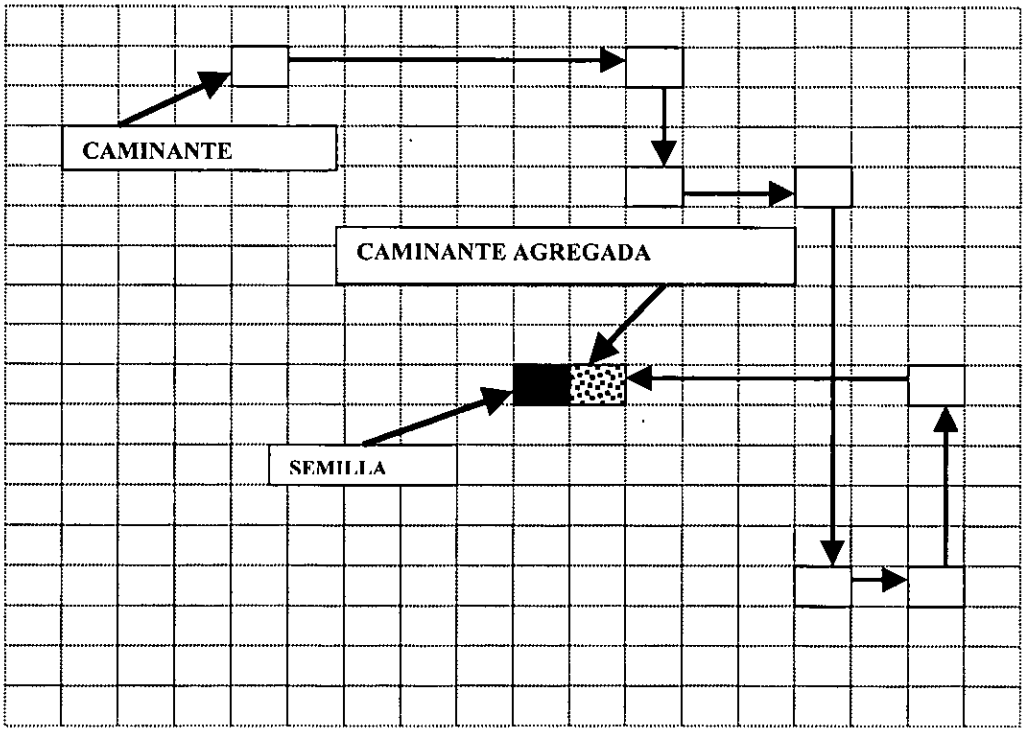
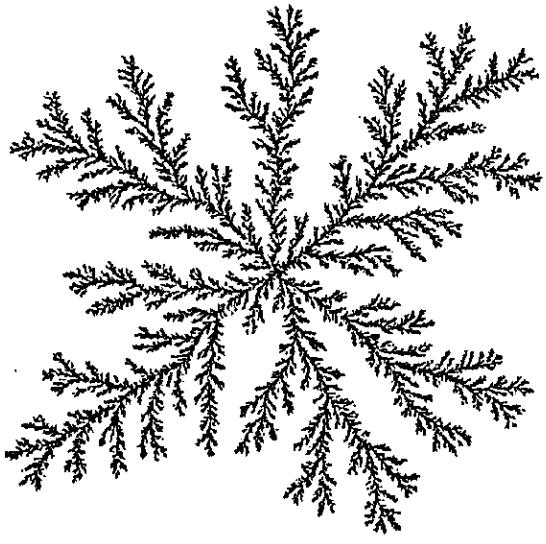


FIG. 1 .1

Debido a que una partícula recorre un camino aleatorio comenzando desde un punto a gran distancia de la semilla, y para cada punto que toca dentro de su recorrido, es necesario saber si ya esta colocada en un lugar adyacente a la semilla. De ser así, se adhiere al grupo; de no ser así, la partícula sigue su camino aleatorio hasta que ocurran una de dos cosas: o bien sale del área limitada por el reticulado, o bien visita un punto adyacente a la semilla y pasa a formar parte del grupo.

La siguiente figura muestra el resultado obtenido después de un gran numero de partículas adheridas a la semilla en dos dimensiones**[2]:**



MODELO DE WITTEN Y SANDER EN DOS DIMENSIONES

FIG. 1.2

Es evidente que los cálculos deben ser muy numerosos antes de saber si ocurrió alguno de los dos sucesos descritos antes. Esto significa mucho tiempo de procesamiento. Por esa razón es necesario contar con algoritmos rápidos y eficientes que reduzcan este tiempo.

Los trabajos de T. Witten y L.M Sander**[2]**, así como los trabajos de P.I Meakin**[3]** y también los de R.C Ball y R.M Brady **[4]** entre otros: **[5 - 8]**, utilizaron modelos para investigar, entre otras cosas, la relación que existe entre los diferentes parámetros que cada uno de ellos utilizó en sus experimentos de simulación.

Por ejemplo: la utilización o no de un enrejado. La relación que existe entre la dimensión fractal y la dimensión euclidiana de las estructuras que simularon. La relación entre el número de partículas que forman un estructura y la dimensión de ésta.

Witten y Sander**[1]** reportan que la correlación de densidad del modelo que utilizan, cae dentro de un comportamiento exponencial, aclarando que limitan el estudio a un crecimiento dendrítico.

P. Meakin , reporta que el radio de giro ***(Rg)*** de la estructura esta relacionada con el número de partículas ***N***, que forman dicha estructura. La relación que encontró se puede expresar de la siguiente manera:

$$R_g \approx N^{\beta}$$

Para un número de partículas ***N*** grande, en donde:

$\beta \approx 6/5d$.

Aquí d es la dimensión euclidiana del enrejado.

El radio de giro ***Rg***, de un número ***N*** de partículas en el agregado, se define de la siguiente manera:

$$R_g{}^2 = \frac{1}{N}\sum_{i=1}^{N}(Ri - Rc(N))^2$$

En donde:

Ri es el radio – vector de la i-esima partícula.

Rc(N) es el radio – vector del centro de la masa del fractal, considerando que cada partícula tiene una masa igual a 1.

En una función de recurrencia, ***Rg*** se calcula como:

$$Rg^2(N) = \frac{N-1}{N} Rg^2(N-1) + \frac{N-1}{N^2}(Rc(N-1) - R_N)^2$$

Donde:

$$Rc(N) = \frac{N-1}{N} Rc(N-1) + \frac{1}{N} R_N$$

Los resultados reportados por P. Meakin**[2]**, indican una relación entre la dimensión fractal o también llamada dimensión Hausdorff, y la dimensión clásica o euclidiana: dicha relación, es expresada en la siguiente forma y es valida para 2 dimensiones y hasta 6 dimensiones:

$$D = 5d / 6$$

Donde:

D = dimensión fractal o Hausdorff.

d= dimensión clásica o euclidiana.

Resultados similares, se obtuvieron de la función de correlación densidad-densidad en simulaciones de dos dimensiones**[3]**. Se realizaron simulaciones sin reticulado en espacios de dimensiones dos y tres.

Los exponentes del radio de giro (β) obtenidos de estas simulaciones son, esencialmente, igual a aquellos obtenidos en simulaciones realizadas con reticulado.

Recientemente, V. Tchijov , Suemi Rodríguez Romo y S. Nachaev **[9]** , han propuesto un nuevo modelo desarrollado en base al original de Witten y Sander. Este nuevo modelo se llama *Colored DLA model.* Esta variación considera dos semillas de diferente color separadas por una cierta distancia ***d.*** Se considera una partícula viajera que tiene una cierta probabilidad ***p*** de ser de un color, y una probabilidad **1-*p*** de ser del otro color. Si en su caminata aleatoria, la partícula toma un lugar adyacente a un color que no es el suyo, se elimina y una nueva empieza otra vez a *caminar.* Cuanto toma un lugar adyacente a su color, la partícula se fija y el proceso se repite hasta formar estructuras fractales de un tamaño que permita obtener información estadística importante. En otro de sus trabajos **[10]**, se realizan varios experimentos computacionales en espacio de dos dimensiones encaminados a encontrar la relación entre la distancia de separación de las semillas y la dimensión fractal que alcanzan las estructuras formadas. Los resultados expuestos, son por demás interesantes. Pero una vez más, no se profundiza en los requerimientos de software, ni tampoco en los algoritmos utilizados para minimizar los tiempos de CPU. Otros ejemplos de este tipo pueden encontrarse en **[11 – 29].**

Básicamente, los tres problemas que se deben atacar en la simulación del crecimiento de partículas son los siguientes:

a).- Generación de números aleatorios que tengan un amplio rango de repetición.

b).- Implementación de un algoritmo que permita calcular las posiciones de cada partícula que se genera aleatoriamente de forma rápida.

c).- Graficación de la estructura generada en tiempo relativamente rápido.

1.3- REFERENCIAS BIBLIOGRÁFICAS DEL CAPITULO 1.

1.- T. Vicsek, Fractal growth phenomena, 2nd edition, World Scientific. Singapure (1987)

2.- T.A. Witten, L.A Sander, PhysicaL Review Letter 41 (1981) 1400.

3.- P. Meakin , Physical Review Letter A27 (1983) 2281

4.- H.B Rosenstock and C. L Marquardt Phys. Rev B.22 5797 (1980)

5.- R.C Ball, R.M Brady A18, 1985 L809- L813.

6.- P. Meakin , Physical Review Letter 51 (1983) 1119

7.- H. E Stanley, Journal Physical A10 (1977) L211

8.- K. Binder, D. W Heermann, Monte Carlo simulation in Statistical Physics. Springer Verlag, Berlin, Heidelberg. (1992).

9.- V. Tchijov, S. Nechaev, S. Rodriguez-Romo, JETP Letter. 64 (1996) 498

10.- V. Tchijov, A. Keller, S. Rodriguez-Romo Revista Mexicana de Física Jan 6. 1997.

11.- Hull, T.E., and Dobell, A.R "Random Numbers Generators", SIAM Rev.IV No.3 July 1982, 230,255.

12.- J. Jhonson, Econometric Methods, N.Y. Mc_Graw HillsCo. 1973.

13.- M. Wolf, Physical Review E47(1993) 1448.

14.- M. Wolf, Physical Review E43(1991) 5504.

15.- I. Vattalainer, T. Ala-Nissila, K. Kankaala, Physical test for random numers in simulations. Phys, Rev. Letters, 1994, V.73, N19, 2573 – 2516.

16.- N. Madras and A. D Sokal, J. Stat. Phys. 50, 109 (1988).

17.- H. T. Herrmann, Physical Reports 136 (1986) 153

18.- P. Meakin, P. Ramanlal, L. M Sander, R. C Ball, Physical Review A34 (1986) 5091. Physical Review Letter A27 (1983) 2281

19.- P. Meakin, Z. Djordjevic, Journal Physical A19 (1986) 1271

20.- P. Meakin,T. Vicsek, Journal Physical A20 (1987) L171

21.- P. Meakin, J. Kertéz, T. Vicsek, Journal Physical A21 (1988) 1271

22.- P. Meakin, Journal Physical A21 (1988) 3501

23.- P. Grassberger , R. Procaccia, American Physical Rev v.28 No.4 october 1983.

24.- T. Vicsek Physical Review Letter 53 (1984) 2281

25.- M. Kolb, R. Botet, R Jullien, Physical Review Letter 51 (1983) 1123

26.- M. Eden, in Proc. of the Forth Berkeley Symposium on Math. Static and Probability, IV (1961) 223.

27.- L. Pietronero, E Tosatti (Eds) Fractal in Phisical, Elsevier. Amsterdam (1986)

28.- R. Julliet, R. Botet, Aggegation and Fractal Aggregates, Word Scientific. Singapure

29.- M. Wolf, S Schwarzer, S Halvin, P.Meakin, H.E Stanley,Physical Review A46(1992) R3016.

CAPÍTULO II NÚMEROS ALEATORIOS.

2.1- NÚMEROS ALEATORIOS

Para el desarrollo de las simulaciones descritas en la revisión de literatura del capítulo anterior, se menciona con frecuencia las *caminatas aleatorias*, también se hace referencia a las posiciones aleatorias que toma una partícula, antes de ser agregada a la estructura o bien, antes de ser eliminada debido a que alcanzó una posición alejada de la semilla.

En términos de programación, la realización de los modelos mencionados en el capitulo anterior es necesario contar con una subrutina o procedimiento, capaz de generar una secuencia de números aleatorios muy amplia, es decir, que el periodo que tarda en repetir la misma secuencia, sea muy grande. Casi todos los lenguajes de programación, tales como C, C++ y PASCAL, cuentan con rutinas que generan secuencia de números aleatorios. Sin embargo, después de cierta cantidad no muy grande de números dentro de la secuencia, esta se empieza a repetir. El estándar ANSI requiere que el periodo de un generador de números aleatorios, por ejemplo de C y C++, sea de solo 32525.

Si para generar estructuras fractales por *Agregación limitada por difusión*, utilizáramos estas rutinas, los resultados de los experimentos no serian confiables puesto que probablemente, la primera estructura generada seria idéntica a alguna otra generada posteriormente, dando así resultados engañosos. Por eso es necesario considerar los siguientes conceptos referente a los números aleatorios.

2.2- GENERADORES DE CONGRUENCIA LINEAL

En este apartado, analizaremos los generadores de secuencia lineal basados en el módulo aritmético. Estos generadores de congruencia lineal, (LCG), pueden usarse para producir permutaciones, y forman la base de los métodos más populares para la generación de secuencias de números aleatorios.

2.3- SECUENCIAS DE CONGRUENCIA LINEAL.

Una secuencia de congruencia lineal esta basado en la siguiente relación de recurrencia.

$x_n = (ax_{n-1} + c) \bmod m, \qquad donde\ 0 \le x_0 \le m$

La variable a es conocida como la multiplicador, c el incremento y m el módulo. Por ejemplo con $a=3,\ c=5,\ m=11$ y $x_0=9$ se obtiene la secuencia siguiente.

9, 10, 2, 0, 5, 9, 10, 2, 0, 5, 9,...

2.4- GENERADORES DE CICLO COMPLETO.

Debido al módulo aritmético usado en el LCG, la secuencia de números producidos será cíclica. La longitud del ciclo es conocida como el *periodo.* Por ejemplo, la secuencia anterior:

9, 10, 2, 0, 5, 9, 10, 2, 0, 5, 9, ...

tiene el periodo igual a 5.

El periodo máximo posible para un LCG con un módulo *m,* es *m* por sí mismo. Un LCG con un periodo máximo posible, se conoce como un *LCG de ciclo completo.* Un LCG de ciclo completo produce, cada ciclo, una permutación de los números entre *0* y *m-1.* Así, el LCG de ciclo completo proporciona otro camino para generar permutaciones.

Se puede obtener un generador de ciclo completo para cualquier *m* dada y escogiendo valores apropiados para *a, c* y x_0. Existen algunas reglas, (vea Knuth**[1]**), que se aplican cuando *c es igual a 0,* para garantizar un ciclo completo.

SECUENCIAS DE NÚMEROS ALEATORIOS.

La secuencia que se utiliza en las ciencias informáticas, es la secuencia de números aleatorios. Con esto se quiere decir que cada número en la secuencia es, para todos los intentos y propósitos, independiente de los números que llegaron antes que él, de tal manera que no se puede predecir el número siguiente.

La técnica más popular para generar secuencias de números aleatorios, es el uso de LCG, debido a que estos son fáciles de implementar, rápido, y si se genera apropiadamente, es capaz de producir secuencias bastante aleatorias.

Si usamos un dispositivo, tal como un LCG para generar secuencias, los números están lejos de ser independientes, debido a que siguen la relación de recurrencia del LCG. Sin embargo, para quienes no conocen esta relación fundamental, los números aparentan ser aleatorios.

Las secuencias de números aleatorios pueden ser generadas usando un LCG, escogiendo muy cuidadosamente los valores para *m, a* y *c.* Por ejemplo, si se escogen los valores de

m=18, a=7, c=5 y se comienza con x_0=12, se obtiene las siguiente secuencia de aleatorios:

12, 17, 16, 9, 14, 13, 6, 11, 10, 3, 8, 7, 0, 5, 4, 15, 2, 1, 12, 17, 16,...

El único problema con esta secuencia, es que tiene el periodo de solo 18 números. A menos que no se utilicen más de 18 números, este patrón sería definitivo. La forma de obtener un periodo largo, es usando un valor grande para el módulo y seguir las reglas de la generación de un ciclo completo.

Aunque, usar un periodo amplio no garantiza una secuencia aleatoria, por ejemplo tomando los siguientes valores: modulo de $m = 2^{31}-1 = 2,147,483,647$ y siguiendo las reglas para un ciclo completo, se puede usar un multiplicador de $a = 1$, debido a que m es primo, si se elige un incremento c= 2 y x_0=1, se obtiene la siguiente secuencia:

1, 3, 5, 7, 9, 11, 13, 15, 17, 19,...

Esta es difícilmente aleatoria, debido a que cada número, es solo incrementado en dos con respecto al anterior. De hecho, para cualquier incremento c que se usa, la secuencia producida tendrá a c como diferencia entre dos números consecutivos.

Este problema, puede ser atacado de tres formas: una, es dándose por vencido con el ciclo completo, y utilizar parámetros que produzcan una secuencia aleatoria, aún si, el periodo debe ser más pequeño. Otra es elegir el número más alto posible para el módulo que no es primo o producto primo. Tal vez ésta no lleve a una secuencia más aceptable. Una tercera forma es abandonar el uso de un generador mezclado del todo, y en lugar de usar un generador multiplicativo puro, con $c=0$. Entonces la relación de recurrencia se reduce a: $x_n = ax_{n-1} \bmod m$.

El hecho de utilizar un generador multiplicativo puro involucra un cálculo menos (la adición) que el generador mezclado. Debido a esto, los generadores multiplicativos puros son comúnmente usados en generadores de números aleatorios.

Existe el problema que nadie puede garantizar la aleatoriedad. Lo que se debe hacer, es intentar con varios valores para *a* y *m,* y probar las secuencias resultantes. Para un generador multiplicativo puro, es necesario un diferente grupo de reglas para asegurar un ciclo completo, pero lógicamente con matemáticas más complejas.

Dos aspectos importantes de estas reglas que se necesita mencionar, son que se necesita que el módulo sea primo. Lo que es caso contrario al generador mezclado, cuando se usa un módulo primo o producto primo limita las opciones y puede producir secuencias no aleatorias. Otro aspecto de las reglas es que no es posible obtener un ciclo completo con un generador multiplicativo puro, sin embargo, se puede obtener uno menos que un ciclo completo muy bueno, el único número que no puede ser usado es el número 0. Una vez que la salida es 0, todos lo números que siguen son 0.

2.5- GENERADOR STANDARD MÍNIMO.

Los generadores multiplicativos puros han sido estudiados ampliamente para su uso como generadores de números aleatorios. Se quiere uno tan grande como sea posible. Y, a diferencia del caso para LCG mezclados, se quiere un módulo primo. Para el módulo 2,147,483,647 hay arriba de 500 millones de valores para *a* que garantizarán un ciclo completo. Sin embargo, muchos de estos no generaran buenos números aleatorios.

Un grupo de parámetros que ha tenido un amplio uso es m = 2,147,483,647 y a=16,807 **[2].** Un generador con este particular grupo de parámetros se llama: *Generador Standard Mínimo.* El generador standard mínimo fue ampliamente probado para aleatoriedad y se considera bueno para producir secuencias aleatorias.

2.6- PRUEBAS DE ALEATORIEDAD.

¿Cómo se puede decir si un generador de números aleatorios verdaderamente produce secuencias aleatorias? La respuesta no es fácil. Sin embargo, existen algunas pruebas estadísticas **[3],[4]**:

- Prueba espectral.
- Entropía de Kolmogorov.
- Prueba de frecuencias.
- Prueba de series.
- Prueba del producto rezagado.
- Prueba de corridas.
- Prueba de distancias.
- Prueba de máximos.
- Prueba de Poker.

En realidad, ninguna prueba dice si una secuencia de números es aleatoria, más bien, la mayoría de las pruebas dicen si una secuencia de números no es aleatoria **[5]**.

Una crítica del generador standard mínimo y realmente el generador de números aleatorios basados en los LCG, es que fallan en la *prueba espectral*.

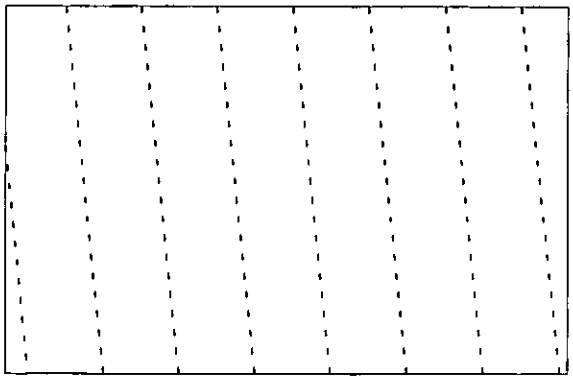
En general, esta prueba busca las correlaciones entre los números producidos en un espacio de K dimensiones. Muchos generadores de números aleatorios tenderán a formar grupos en hiperplanos. Para un ejemplo más explicativo, se toma el caso de $k = 2$. Aquí los números consecutivos se prueban para ver si existen correlaciones. Una forma gráfica para desarrollar esta prueba, es tomar los números aleatorios de la secuencia en pares, formando de esa manera coordenadas bidimensionales:

1,4,8,2,5,9,12,43,24,7,0,1,.....

(1,4), (8,2) , ...

Los puntos entonces se trazan en estas coordenadas. Este proceso se repite hasta obtener una gráfica llena de puntos.

Si los números son realmente aleatorios, no habrá ningún patrón notable en los puntos. Si hay correlaciones, se observan en la gráfica los puntos alineados, vea la figura 2.1.



Gráfica espectral por el Generador Standard Mínimo.

Fig. 2.1

Para tres dimensiones, seria necesario considerar los tres primeros números de la secuencia como un punto de coordenadas *(x,y,z)*, los números 2 al 4 como otro punto, los números 3,4,5 de la secuencia como un punto más, y así sucesivamente, graficandolos en un espacio tridimensional, verificando entonces en cada plano del espacio que no se forman patrones repetitivos en ninguno de los planos. En el apéndice *A*, se presenta el listado del programa desarrollado en C++ y que permite realizar esta prueba en tres dimensiones.

ENTROPIA DE KOLMOGOROV:

Probablemente la prueba del espectro de números aleatorios no comprueba la aleatoriedad de una secuencia de números, pero existe una prueba más eficiente, la entropía de Kolmogorov **[5].**

Básicamente la entropía de Kolmogorov estudia la relación que hay entre cada número de la secuencia y el resto de números que forman parte de esta. En este sentido, la entropía puede tener los siguientes valores:

$K = 0$

$K =$ un número positivo.

$K = \infty$

Para el primer caso, se dice que la secuencia representa un sistema periódico.

Para el segundo caso, se dice que el sistema es caótico.

Para el tercer caso, se dice que la secuencia es aleatoria.

Desde luego, la gran cantidad de cálculos que deben hacerse para calcular la entropía de Kolmogorov en una secuencia de más de 60000 números, requiere de una gran cantidad de recursos de computadora.

En general, para aplicaciones técnicas se deben usar generadores de números aleatorios aceptados por la comunidad científica.

En el presente trabajo, se utiliza el generador de números aleatorios ***Ran3*** propuesto en **[2].** Este ha sido utilizado para generar caminatas aleatorias autorepelentes en tiempo continuo en espacios de cuatro dimensiones**[6]** y caminatas con vecinos más cercanos prohibidos en la reja diamante, en espacios de tres dimensiones**[7].**

Es importante mencionar que éste generador pasó satisfactoriamente las pruebas de aleatoriedad, vea **[8].**

En **[9 – 10]** se proponen otros generadores de números aleatorios. Sin embargo los mismos autores reconocen las limitaciones que éstos tienen para ser usados en experimentos de simulación. Otras temas relacionados con los números aleatorios, las formas de generar secuencias de ellos y las pruebas que deben superar para que las secuencias sean utilizadas en experimentos de simulación se encuentran desarrolladas en **[11- 14].**

Es posible encontrar varios generadores de números aleatorios así como programas de verificación de su calidad en internet: http://www.netlib.org/random.

2.7- REFERENCIAS BIBLIOGRÁFICAS DEL CAPITULO 2.

1.- D. E. Knuth, The Art of Computer Programming, Vol . 3 Reading, Massachusetts, Addison Wesley. 1973

2.- W. H. Press B. P. Flannery, S. A. Teukolsky W. T. Vetterling Numerical Recipies in C, Cambridge University Press, USA 1980

3.- Hull, T.E., and Dobell, A.R "Random Numbers Generators", SIAM Rev.IV No.3 July 1982, 230,255.

4.- J. Jhonson, Econometric Methods, N.Y. Mc_Graw Hills Co. 1973.

5.- P. Grassberger , R. Procaccia, American Physical Rev v.28 No.4. October 1983.

6.- S. Rodríguez Romo, V. Tchijov, J. Stay. Phys., 1998, Vol 1/2 , 763

7.- V. Tchijov, S. Rodríguez Romo, J. Chem. Phys., 1999 (aceptado)

8.- I. Vattalainer, T. Ala-Nissila, K. Kankaala, Physical teste for random numers in simulations. Phys, Rev. Letters, 1994, V.73, N19.

9.- B. Flamig Practical Algorithms in C++ John Wiley & Sons, Inc. U.S.A 1995.

10.- B. Eckel Using C++, Osborne Mc Graw-Hill, USA 1989.

11.- M.F Osborne, Brownian Motion in the stock Marquet. Operation Research. V.2 VII March-April, 1959. Pp 145-175.

12.- B. Ostle Estadistica Aplicada: tecnicas de estadistica moderna, cuando y donde aplicarlas. Ed. Limusa. Mexico 1965.

13.- A. Serletis, The Randoms Walk in Canadian output. Canadian Journal of Economic V. XXV May 1992.

14.- R. Syski, Random Processes: a first look. M. Dekker. New York 1989.

CAPÍTULO III ALGORITMO DE *TABLAS HASH*

3.1- ALGORITMO DE *TABLAS HASH*

Un problema de gran importancia en los modelos de agregación limitada por difusión es una verificación rápida para saber si una partícula viajera se acercó a la estructura ya existente o todavía está lejos de ella. La técnica más eficiente de dicha verificación es el método de las *tablas hash.*

Comenzaremos por dar una idea general de *tablas hash,* en qué consiste *hashing,* además de dar un breve glosario de algunos términos utilizados a través del capítulo.

La estructura de datos *tabla hash*, es meramente un arreglo con un tamaño fijo, conteniendo claves. Típicamente, una clave es una cadena de caracteres o números con un valor asociado**[1 – 2]**.

En la terminología *hashing*, cada elemento del arreglo se llama *bucket* (cubo) o *celdas.*

Al proceso de implementación de las *tablas hash*, frecuentemente es llamada *hashing.*

Dando una definición más acertada, podemos decir que *Hashing*, es una técnica utilizada para la construcción de tablas para búsquedas, inserciones y borrado altamente eficientes.

En un buen diseño de una *tabla hash*, se encuentra que una entrada puede tomar sólo una o más comparaciones, sin importar el número de entradas presentes.

Cuando dos elementos señalan al mismo *bucket*, se produce el fenómeno llamado *colisión.* Los métodos para el manejo de colisiones se conocen como estrategias de resolución de colisiones (*collision resolution strategies*)**[3].**

3.2- *HASHING*, MUESTREO ALEATORIO INVERTIDO.

Las funciones para el mapeo en la tabla se llaman funciones *Hash* (*Hash Functions*). En un diccionario, podemos encontrar que *Hash* significa *picar en pequeños pedazos* o *confundir y desordenar*. Una función *hash* debe *picar* una clave, que proviene de un gran rango de valores, e introducirlo dentro del menor rango de *buckets*. Cuando se mapea de muchos a pocos, las colisiones ocurren forzosamente. Una forma de minimizar colisiones es mezclar las claves en una forma que la salida de la función *hash* sea esencialmente aleatoria. Funciones con estas propiedades son conocidas como funciones *hash* uniformes: (*uniform hash functions*). Idealmente, se quiere que cada *bucket* tenga un $1/m$ oportunidad de ser seleccionada, dado un grupo aleatorio de claves, como en el caso de las coordenadas de las posiciones de las partículas adheridas, y donde m es el número de *buckets* en la tabla. Realmente, también es posible tener malas funciones *hash*, como el caso en donde todas las claves señalan al mismo *bucket*. En el peor de los casos, el desempeño de *hash*ing no es mejor que hacer simple búsqueda lineal en los *buckets*.

Sorprendentemente, *hashing* está relacionado a un muestreo aleatorio, recordando que en una muestra aleatoria, *k ítems* se eligen aleatoriamente de un grupo de r elementos. Supongamos que asignamos un número único en el rango de 0 a m-1 a cada muestra en un grupo de m muestras aleatorias únicas. Estos números son equivalentes a índices de *buckets* y los ítems en cada muestra son las entradas cuyas claves mapean al mismo *bucket*.

Con *hashing*, se tienen muchas muestras, escogidas *a priori*, y debemos determinar de entre el grupo de muestras a cual muestra pertenece una clave dada.

3.3- FUNCIONES *HASH.*

Se debe trabajar sobre dos características básicas de las *tablas hash*: la función *hash* y la estrategia de resolución de colisiones. Una función *hash* debe satisfacer dos requerimientos, que usualmente están una en contra de la otra. Es deseable que la función *hash* sea rápida, tomando solo un poco de cálculos. Además la función debe ser bastante hábil para minimizar colisiones**[3 – 5]**.

Idealmente, es deseable que la función *hash* produzca un índice único para cada clave. De esa forma no habría colisiones. Las funciones *hash* que están libres de colisiones se llaman funciones *hash* perfectas (*perfect hash functions*) y encontrarlas no es fácil.

Existen algoritmos que las encuentran, pero resultan inadecuadas, ya sea por ser muy caras o porque no garantizan trabajar todo el tiempo.

El problema más serio es que encontrar funciones *hash* perfectas requiere conocimiento previo de todas las claves que se esperan. Esto impide el uso de *hashing* para situaciones más dinámicas, donde las claves no son conocidas por adelantado.

El esfuerzo se debe concentrar en encontrar funciones *hash* que minimicen colisiones y que sean rápidas de calcular.

3.4- *HASHING* NUMÉRICO.

La técnica adecuada para *hashing* de claves numéricas es simplemente tomar el módulo *m* de la clave, donde *m* es el número de *buckets* en la tabla. La siguiente ecuación es la más comúnmente utilizada: $h(k) = k \bmod m$

La ecuación se conoce como el método de la división de *hashing*. Una razón por la que el método de la división es popular es porque toma a la clave el módulo *m*. Automáticamente transforma la clave en el rango deseado de los índices de los *buckets*, desde 0 a $m - 1$.

Es crucial que se escoja un buen valor para *m*, o los resultados no serán buenos. Si se hace *m* un número par, entonces las claves pares se introducirán en *buckets* pares y las claves impares en *buckets* impares. De esa manera si todas nuestras claves fueran impares la mitad de los *buckets* no se utilizarán y se esperarían más colisiones.

Si las claves son verdaderamente aleatorias, la selección de *m* no es crítica. De hecho, para claves verdaderamente aleatorias, se pueden tomar meramente unos pocos bits de cada clave como el valor de *hash*. Pero en el mundo real, las claves difícilmente son aleatorias.

Las funciones de *hash* más efectivas usan toda la información en una clave, así las claves similares se pueden esparcir uniformemente. Idealmente, una función *hash* debe trabajar tan bien para llaves no aleatorias como para llaves aleatorias. Eso proporciona un grado de fortaleza. Pero lo que no se desea es estar experimentando por mucho tiempo con diferentes funciones *hash* para cada aplicación.

Esto nos lleva a que el método de división de *hashing* parece ser muy fuerte si hacemos *m* un número primo. Por ejemplo, si se determina que se desean alrededor de 100 *buckets*, entonces tablas con tamaños de 103, 107, 109 ó 133 *buckets* serían convenientes.

El método de la división no es infalible, pero probablemente no hay una función *hash* que lo sea. Es posible preparar un grupo de claves que provoquen un desempeño muy pobre de una función *hash* dada. En el peor de los casos, todas las claves señalan al mismo *bucket*. Para un buen diseño de una función *hash*, sin embargo, la oportunidad de que ocurra esto en aplicaciones reales es muy pequeña. Confiando en las leyes de probabilidad, si se puede garantizar que el peor de los casos no sucederá, entonces no debemos usar *hashing*.**[3]**.

3.5- *HASHING*, CADENAS DE CARACTERES

Como ya se mencionó, existen aplicaciones donde las claves son numéricas. Sin embargo, es muy común que las claves sean cadenas de caracteres. Algunos ejemplos son nombres, domicilios, o identificadores y palabras clave en un programa.

Para estos casos se tiene una posibilidad; sumar los valores enteros de cada carácter en la cadena y usar el resultado como el índice del *bucket*.

Debido a que es posible para el índice resultante ser más grande que el número de *buckets*, es común tomarle al resultado, el módulo *m*.

En otras palabras, una vez que una cadena de caracteres es convertida a una forma numérica, se usa el método de la división tanto para aplicar más tarde *hash* al resultado como para mantener el valor en el rango. Como en el caso anterior, lo mejor es si *m* es un número primo.

3.6- ESTRATEGIAS DE RESOLUCIÓN DE COLISIONES

Existen tres estrategias básicas utilizadas para manejar las colisiones**[3,7]**.

- *Direccionamiento abierto (Open addressing, Close Hashing).* Cuando una colisión ocurre, se examina la tabla buscando un *bucket* vacío para colocar la nueva entrada.
- *Encadenamiento por separado (Separate chaining, Open hashing).* Cada *bucket* encabeza una lista. Cuando una clave esta señala a un *bucket*, la entrada simplemente se agrega a la lista del *bucket*.
- *Encadenamiento unido (Coalesced chaining).* Este es un híbrido de las primeras dos técnicas. El arreglo del *bucket* mantiene las entradas, las cuales son enlazadas en listas múltiples cuando ocurren colisiones. A diferencia del encadenamiento por separado, a las listas se les permite unirse. Cada lista puede contener entradas con claves que tienen diferentes valores *hash*.

3.7- ENCADENAMIENTO POR SEPARADO.

La manera más simple para resolver colisiones es mantener las entradas en una lista, una para cada *bucket*.

De los métodos de resolución de colisiones, el encadenamiento por separado es el método más intuitivo y el más fácil de implementar. En este método a diferencia de los otros, no hay un límite fijo en el número de entradas que se pueden agregar a la tabla. Las listas de los *buckets* pueden crecer a longitudes arbitrarias.

Por supuesto, mientras las listas son más largas la búsqueda será más lenta. Idealmente se requiere que las listas contengan solo unos cuantos nodos, debido a que el porcentaje de la longitud de las listas de los *buckets* será de *n/m.* El encadenamiento por separado reduce esencialmente la búsqueda, pues será más rápida mientras el valor asignado a *m* sea mayor**[8]**.

3.8- DIRECCIONAMIENTO ABIERTO.

Si se sabe con anticipación cuantas entradas se esperan, entonces la técnica llamada direccionamiento abierto (open addressing), puede darnos una tabla más compacta. Como antes, una función *hash* determina un índice del *bucket* para una clave dada. Si un *bucket* esta vacío, la entrada es simplemente copiada dentro del *bucket.* Si el *bucket* no esta vacío, entonces ocurre una colisión. El termino direccionamiento abierto parte del hecho que después de una colisión, se analiza la tabla en busca de un *bucket* abierto (esto es, vacío).

Cuando hay una colisión, el resto de los *buckets* en la tabla se analizan en una secuencia predeterminada. Hasta que se encuentra un *bucket* para insertar la entrada en él. Si no se encuentra un *bucket* vacío, entonces significa que la tabla está llena.

Hay dos problemas para el manejo con direccionamiento abierto. El más fácil de resolver es como determinar cuando un *bucket* está vacío. Al menos un bit necesita ser ocupado para esto, ya sean las entradas por ellas mismas o almacenadas en una bandera auxiliar en cada *bucket.*

El problema más difícil de resolver es determinar la secuencia de análisis. En general, se quiere que la secuencia de análisis tenga las siguientes propiedades: debe de ser capaz de examinar todos los *buckets* de la tabla si es necesario. Los análisis deben ser lo más dispersos posibles para minimizar el número de colisiones que ocurren. El primer requerimiento implica que la secuencia de análisis**[8 – 10]**:

$P_0, P_1,...,P_{m-1}$, es realmente una permutación de los números desde 0 a m-1, en donde P_0 *es* el *bucket* dado por la función *hash*. Existen cuatro tipos diferentes de secuencias de análisis que son:

1. *Análisis Lineal (Linear probing).*
2. *Análisis cuadrático (Quadratic probing).*
3. *Análisis uniforme (Uniform probing).*
4. *Análisis por doble Hashing (Double hash probing).*
5. *Análisis aleatorio (Random hashing).*

3.9- *REHASHING.*

Con todas las técnicas que se han presentado, existe un problema cuando se trata de agregar más entradas, especialmente para tablas con un tamaño fijo. Aun con el encadenamiento separado, que permite agregar cualquier número de entradas, eventualmente se tienen problemas. Como las cadenas llegan a ser muy largas para ser prácticas, la tabla debe de ser diseñada acorde a las entradas que se esperan, o de lo contrario prever de alguna manera el crecimiento de la misma.

Una técnica que se usa cuando una tabla se llena, es crear otra. Una más larga, quizá dos veces el tamaño, e insertar las entradas desde la primera tabla. Esto es lo que se llama *Rehashing*[**3, 11**]. Para cada clave se debe aplicar la función *hash* (*rehashed*), debido a que el número de los *buckets* es diferente. Dependiendo del tamaño de la tabla aumenta el costo, así que no es conveniente hacerlo.

Existen técnicas que permiten crecer la tabla sobre la marcha, pero son complicadas, y casi todas las técnicas sufren pequeños problemas de desempeño aun si la tabla no necesita crecer. Otra técnica popular se llama *hashing* extendible (*extendible hashing*), la cual arma una estructura de directorio jerárquica basada en los patrones de bit de las claves *hashed*[**11 – 13**].

Las tablas hash se utilizaran en un capítulo posterior para resolver problemas relacionados con la simulación de crecimientos *DLA*.

3.10- BIBLIOGRAFIA DEL CAPITULO 3.

1.- A. V. Hopcroft, J. E Ullman, Data structures and Algorithms. Addisin Wesley, Reading mass.

2.- A. A Berk The language of Artificial Intelligence. Princeton, N.Y 1985

3.- B. Flaming Practical Algorithms in C++ John Wiley & Sons, Inc. U.S.A 1995.

4.- M. H Bracket Developing Data Structured Database, Pretince Hall Englewood, N.J 1987.

5.- J. C. Cleaveland, An Introduction to Data Types. Addison Weslwy , 1986.

6.- D. E. Knuth , The Art of Computer Programming, Vol 1: Fundamnetal Algorthms. Addison Wesley. 1979.

7.- D. E. Knuth , The Art of Computer Programming, Vol 3: sorting and searching. Addison Wesley. 1979.

8.- M.H. Overmars The Design of Dynamic Data Structures. Springer – Velag, N. Y 1983.

9.- R.S. Pressman Software Engineering: A Practitioner's Approach. McGraw Hill 1980.

10.- J.P Tremblay, An Introduction to Data Structures with Application, McGraw Hill, N.Y 1984.

11.- J.L bentley and J.B Saxe, Decomposable searching problems, Journal of Algorithms 1980.

12.- H. Edelsbrunner, M.H Overmars Batched dynamic solutions to decomposable searching problems. Journal of Algorithms 6, 515-542.

13.- P. Flajolet, J Francon, J. Vuillemin. Sequence of operations analysis for dynamic data structures, Journal of Algorithms, 1(2), 111 – 141.

CAPÍTULO IV.- PROGRAMACIÓN ORIENTADA A OBJETOS.

Los programas que hemos creado para hacer simulaciones computacionales, están escritos en el lenguaje de programación C++. Esto permite utilizar la tecnología orientada a objetos para crear estructuras fractales con el modelo de *agregación limitada por difusión.*

En este capítulo se proporciona un resumen de lo que es la *programación orientada a objetos*, conocida también por sus siglas *POO.*

Aunque C++ es un lenguaje complejo, existen solo cuatro claves que componen la programación orientada a objetos**[1,2,3].** Estas son:

- Clases
- Objetos
- Instancia de Variables
- Métodos

4.1- PROGRAMACIÓN ORIENTADA A OBJETO

El trabajo que aquí se presenta utiliza la programación de computadoras. En realidad cualquier lenguaje se puede utilizar para implementar tanto el generador de números aleatorios como las tablas hash. Sin embargo, la naturaleza del problema de agregación limitada por difusión, se adapta para aplicar la programación orientada a objetos, debido a que cada partícula puede ser tratada como un objeto y a su vez, puede tomar diferentes

características. Por esta razón, se proporciona aquí un resumen de la programación orientada a objetos.

En términos simples, POO es una programación con objetos. Los objetos son una extensión evolucionaría del concepto de registros, solo que ahora nos permiten organizarlos dentro de paquetes. Los objetos nos permiten combinar datos y código dentro de un simple paquete. Un objeto, es un lenguaje constructor que enlaza datos con funciones y estas funciones realizan alguna operación sobre el dato. Uno puede pensar en un objeto como una estructura del lenguaje C que contiene datos, pero ésta comparación no puede ser considerada valida, pues los objetos son mucho más poderosos. Además los objetos pueden ser extendidos para incorporar nuevos elementos de datos y funciones, usando adecuadamente la propiedad de *herencia*[**4 - 7**].

Debido a que los objetos contienen código y datos, estos son como miniaturas programas que se auto - contienen. Esto les permite utilizar a los objetos como bloques de construcción para crear objetos más complejos.

Por ejemplo, asumimos que se necesita crear una aplicación como procesador de palabras. En lugar de concebir el programa como un grupo de funciones para manejar la tarea requerida, se puede crear un grupo de objetos.

La siguiente figura muestra un posible arreglo[**4**].

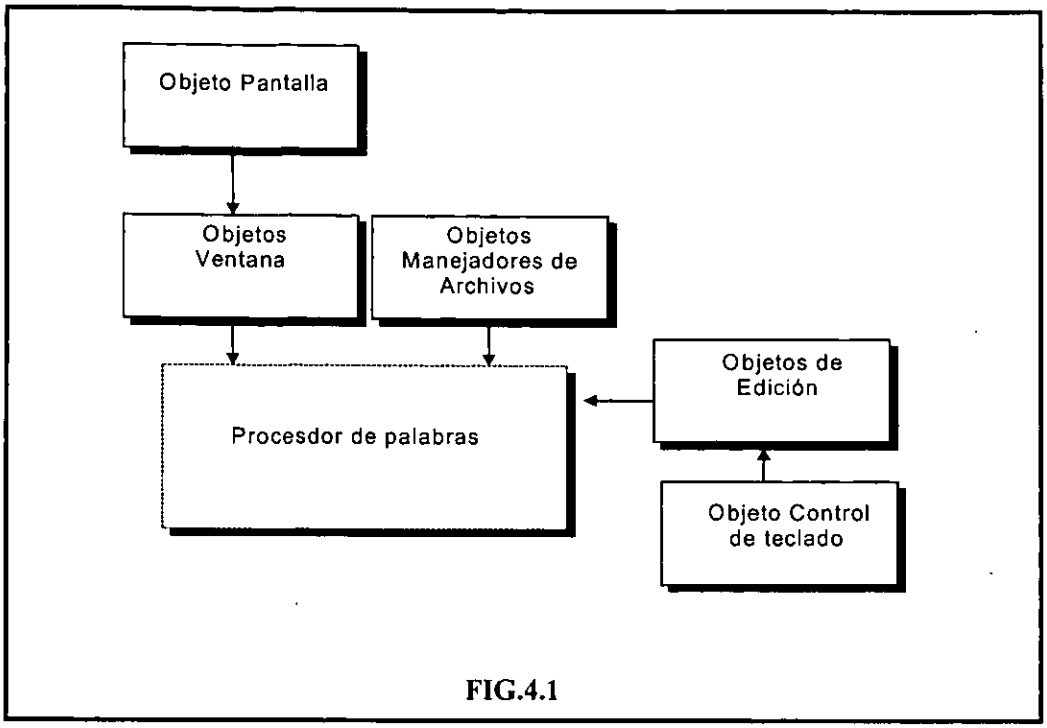


FIG.4.1

El programas es dividido en objetos: como el objeto de pantalla, el objeto ventana, el objeto manipulador de archivos, y otros. Las operaciones de un programa pueden ser fácilmente aisladas. Los objetos pueden ser diseñados para que trabajen independientemente uno del otro. Es mucho más fácil mantener programas que son construidos desde colecciones de objetos, en lugar de un simple conjunto de funciones que son más difíciles de mantener y de modificar. Todas estas características significan una ventaja a la hora de programar. Sin embargo, hay que notar que algunos de los objetos tales como la pantalla y el control de teclado, se utilizan a su vez para crear otros objetos. Esto ayuda a esconder los detalles de los objetos de menor nivel y nos permiten unir objetos en la construcción de bloques cada vez más grandes**[7]**.

4.2- LAS CLASES.

Una clase es un modelo usado para definir un objeto. La siguiente figura ilustra que una clase es un tipo de dato, y un objeto es una variable de una clase.

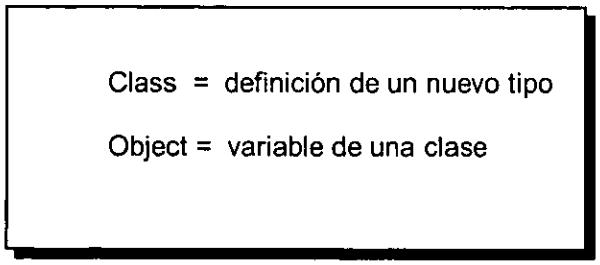


FIG. 4.2 Diferencia entre un objeto y una clase

Una clase contiene dos tipos de componentes: variables de clase y métodos. Una variable de clase sirve como un elemento de datos, y un método es una función. En un sentido, las variables de clase definen el estado interno del dato de un objeto. Los métodos definen al comportamiento del objeto, esto es, las acciones que un objeto puede realizar.

Realmente, a fin de ser más consistentes con la terminología común de C++, el término *dato miembro* será usado para referirse a la variable de clase y el término *funciones miembro*, para hacer referencia a métodos.

4.3- CARACTERÍSTICAS DE LA PROGRAMACIÓN ORIENTADA A OBJETOS.

La programación orientada a objetos con C++; nos muestra tres propiedades que son recurrentes, **[5]:**

- Encapsulación
- Herencia
- Polimorfismo

4.4- ENCAPSULACIÓN.

La encapsulación es el mecanismo que agrupa el código y los datos que maneja y los mantiene protegidos contra cualquier interferencia o mal uso. En cualquier lenguaje orientado a objetos, el código y los datos pueden *empaquetarse* de la misma forma que se crea una *caja negra* auto-contenida. Dentro de la caja son necesarios tanto el código como los datos. Cuando el código y los datos se han enlazado de ésta manera se ha creado un objeto.

La encapsulación proporciona dos características importantes:

- Pone datos y funciones bajo un mismo techo.
- Proporciona capacidades para ocultar datos.

Por ejemplo, si se desarrolla una base de datos objeto, no se necesita saber como son almacenados los objetos en el sistema. La única cosa que se necesita saber es que rutinas se llaman para acceder el objeto.

La encapsulación tiene tres propósitos: el primero es que protege los datos de objetos, pues usualmente se accede a los datos a través de funciones que están definidas en el objeto. En segundo lugar, la encapsulación hace más fácil usar los datos de objetos, porque todas las acciones se llevan a cabo a través de una buena definición de interfaz. Finalmente, la encapsulación se puede utilizar para ocultar los detalles de la manera como los datos son almacenados o implementados.

4.5- HERENCIA:

La herencia es el proceso mediante el cual un objeto puede adquirir las propiedades de otro. Un objeto puede heredar un conjunto general de propiedades a las que puede añadir aquellas características que son específicamente suyas**[7]**.

La herencia nos permite derivar una nueva clase de una clase ya existente. Esto nos permite:

- Agregar datos y código a una clase, sin tener que cambiar la clase original
- Reutilizar código
- Cambiar el comportamiento de una clase

Otra característica poderosa de herencia, involucra la habilidad para modificar una clase de un objeto existente, para hacer una nueva clase que tiene una personalidad ligeramente diferente. De esta forma, se puede usar la herencia para crear múltiples objetos que desarrollen nuevas acciones, aunque los objetos sean derivados del mismo bloque.

4.6- POLIMORFISMO

El polimorfismo es la cualidad que permite que un nombre se utilice para dos o más propósitos relacionados, pero técnicamente diferentes. El propósito del polimorfismo aplicado a la programación orientada a objetos es permitir usar un nombre para especificar una clase general de acciones.

Un aspecto importante de polimorfismo, es que cada objeto en la familia, puede tener métodos con los mismos nombres, pero el código para cada método del objeto puede ser enteramente diferente. Como ejemplo se puede mencionar el hecho de que el símbolo " * " es utilizado para designar la operación de multiplicar en el lenguaje *C++*, sin embargo, también es utilizado para designar un tipo de variable especial llamada puntero.

En los programas utilizados este trabajo para generar estructuras fractales y secuencias de números aleatorios y algunos otros de apoyo están codificados es C++, utilizando tecnología orientada a objetos.

4.7- REFERENCIAS BIBLIOGRÁFICAS DEL CAPITULO IV.

1.- B. Flamig, Practical Algorithms in C++, John Wiley & Sons, Inc. U.S.A 1995.

2.- B. Eckel, Using C++, Osborne Mc Graw-Hill, USA 1989.

3.- H. Schildt, C++ para programadores, Osborne Mc Graw Hill, 1996.

4.- L. Adams, Programación Gráfica en C. Ed. Anaya Multimed. S.A,México 1991.

5.- N. Barkakati, H.W. Sams and Co. The Waite Group's Turbo C Bible, First Edition USA 1989.

6.- H. Schildt, C: Complete Ref., Mc Graw – Hill, Second Ed. USA 1987.

7.- H. Schildt, Guia de Autoenseñanza, Mc Graw – Hill, Primera Ed. Madrid 1995.

CAPÍTULO V TABLAS HASH EN LA AGREGACIÓN LIMITADA POR DIFUSIÓN.

5.1- TABLAS HASH EN LA AGREGACIÓN LIMITADA POR DIFUSIÓN.

Uno de los problemas de carácter informático que plantea el modelo DLA, es la necesidad de calcular de una forma rápida las posiciones de las partículas caminante, recordando que dicha partícula toma posiciones aleatorias. Para cada una de estas posiciones es necesario saber si las partículas caminante están ya unidas a la semillas o no lo están. Debido a que el número de posiciones que toma la partícula caminante son muchas, se requiere un algoritmo que permita hacer cálculos rápidos.

Cuando hablamos de tres dimensiones, la posición de la partícula se almacena en arreglos tridimensionales que van creciendo de la misma forma en que las partículas forman una estructura más grande:

```
1 0 0
.....
.....
2 5 8
.....
.....
3 6 3
.....
.....
```

La coordenada 1,0,0 corresponde a la partícula "semilla", mientras que las demás coordenadas pertenecen a partículas que se han integrado según el modelo que se estudia.

El problema es que mientras más partículas se unen formando a la estructura, más cálculos se requieren para saber si una partícula caminante se unirá o no a la dicha estructura. Mientras más cálculos se hacen, mayor tiempo de CPU se requiere y el trabajo empieza a depender de las características de la computadora que se utiliza.

El algoritmo de *Hash Table* presentado en el capítulo III, tiene la ventaja de poderse aplicar a cualquier dimensión y que el tiempo en el que el algoritmo verifica si la partícula está o no unida a la estructura no depende de la cantidad de partículas que ya estén formando a la estructura. Éste tiempo se mantiene básicamente constante**[1]**.

Sean los siguientes puntos parte de una estructura que ya ha crecido partiendo de una semilla;

(1,0,0)

(1,1,0)

(1,1,1)

(2,1,1)

Sea (3,-1,2) las coordenadas de la partícula caminante.

Considerando también a un número *M* como la cantidad máxima de partículas caminantes **[2]**.

El algoritmo consiste en obtener un índice para cada partícula que ya está integrada a la estructura y también para la partícula caminante. Éste índice se obtiene a partir de la expresión: *(Ax + By + Cz)mod M,* donde x, y, z son las coordenadas de los puntos que ya están formando a la estructura y de las partículas caminantes. El criterio para obtener las constantes enteras *A, B, y C,* es el siguiente**[3]**:

$$A \approx M^{1/D}$$
$$B \approx M^{2/D}$$
$$C \approx M^{3/D}$$

En donde *D* es la magnitud de la dimensión de trabajo + 1, es decir para el caso de tres dimensiones tenemos: *D=3+1.*

Dentro de los cálculos, *M y D* son conocidos y los valores de *X, Y, Z* son puntos en un espacio tridimensional. Es importante mencionar que de una forma gráfica, estos puntos tridimensionales se consideran como las coordenadas del vértice inferior izquierdo de un cubo que a su vez representa a las partículas, tanto de las que ya forman parte de la estructura como de la partícula caminante.

Aunque los valores que se obtienen para las constantes *A,B,C* son reales, el algoritmo indica que se *aproxime al número primo* más cercano. La razón para esta consideración es que se evitan el exceso de colisiones. Para simplificar la explicación consideremos el siguiente caso en tres dimensiones: Veamos una aplicación con valores reales en tres dimensiones, consideremos un número de partículas digamos M=1000,

Si estamos trabajando en tres dimensiones entonces la función Hash queda reducida al siguiente término: *(Ax+By+Cz)mod M.*

Entonces los valores de las constantes *A, B y C* son los siguientes:

$$A = M^{1/4} = \sqrt[4]{1000} = 5.623 \Rightarrow \text{.....aproxmamos.....} A = 5$$
$$B = M^{2/4} = \sqrt[4]{(1000)^2} = 31.62 \Rightarrow \text{.....aproximamos.....} B = 31$$
$$C = M^{3/4} = \sqrt[4]{(1000)^3} = 177.83 \Rightarrow \text{...aproximamos.....} C = 177$$

Recordar aquí que los valores de las constantes *A, B y C* se deben ajustar al valor primo más cercano para evitar al máximo las colisiones. Consideremos ahora que iniciamos con una *semilla* en el centro de un enrejado tridimensional y que las coordenadas tridimensionales del vértice inferior izquierdo de esa partícula es (0,0,0), por lo tanto su función Hash corresponde a *(Ax+By+Cz)mod M.* Como el número de partículas *M* para este ejemplo es 1000, *entonces el residuo de la semilla es 0.*

Refirámonos al siguiente dibujo:

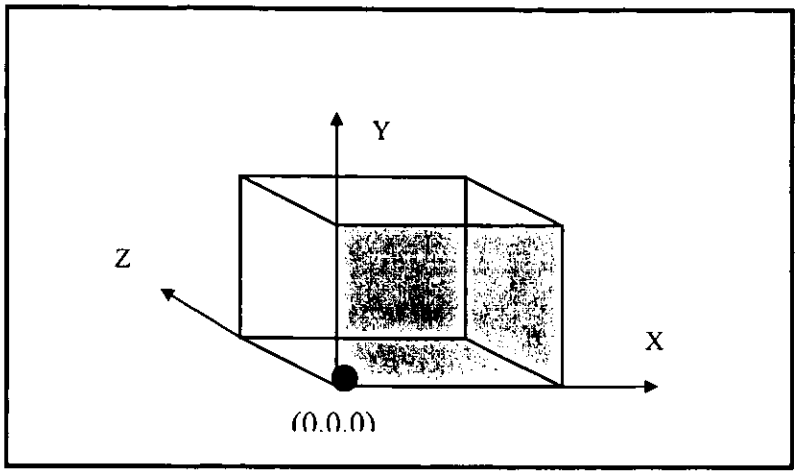


FIG 5.1

En el siguiente dibujo, consideramos las alternativas de crecimiento:

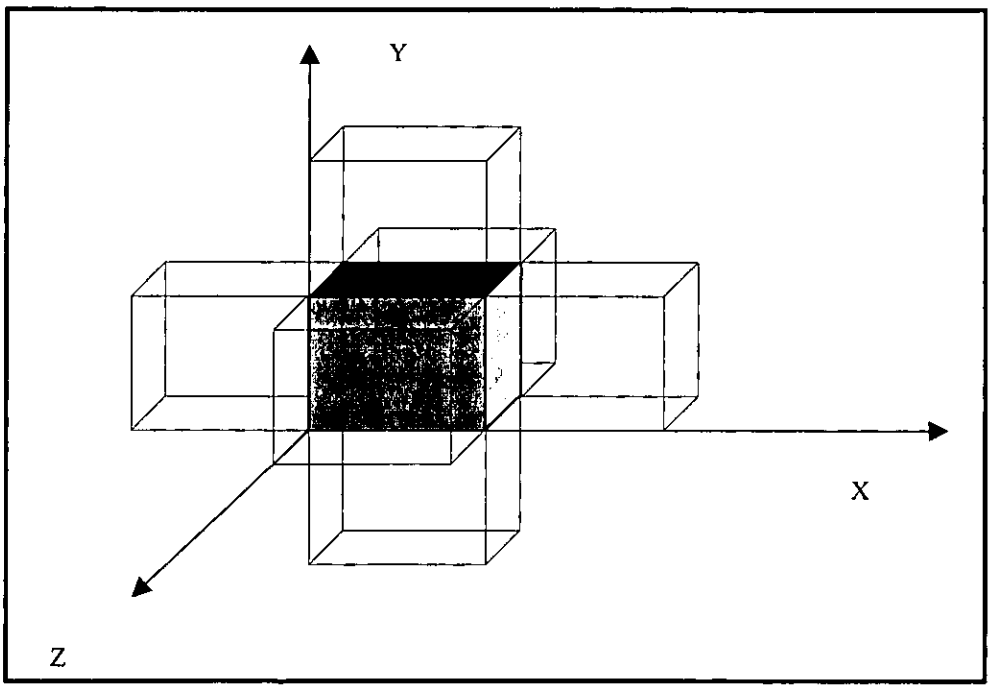


FIG. 5.2 SEIS ALTERNATIVAS DE CRECIMIENTO DE LA SEMILLA

En el dibujo, las coordenadas inferiores izquierdas de cada cubo definen a cada una de las alternativas de crecimiento. Cada una de ellas en cierto momento, es generada de manera aleatoria y se considera que hace caminatas aleatorias.

CAMINANTE: (X,Y,Z)	**POSIBILIDADES:**	**(X+1, Y, Z)** **(X, Y, Z-1)** **(X, Y-1, Z)** **(X, Y+1, Z)** **(X, Y, Z+1)** **(X-1, Y, Z)**

Para verificar si la partícula caminante se une o no a la semilla en alguno de sus seis lados, primero consideremos sus coordenadas. Considerando que de forma aleatoria aparece una partícula caminante cuyas coordenadas de su vértice inferior izquierdo son (1,0,0). Entonces, es necesario verificar si la función Hash aplicada a las seis posibles alternativas de unión de la partícula, tienen un residuo igual al de la semilla.

Recordando que*: A=5, B=31, C= 177*

Los puntos a verificar son:

COORDENADAS	*RESIDUOS:*
(X+1, Y, Z) = (2,0,0)	10
(X, Y, Z-1) = (1,0,-1)	-177
(X, Y-1, Z) = (1,-1,0)	-31
(X, Y+1, Z) = (1,1,0)	31
(X, Y, Z+1) = (1,0,1)	177
(X-1, Y, Z) = (0,0,0)	0

Al aplicar la formula *(Ax+By+Cz)mod M.* Obtenemos los residuos mostrados en la columna derecha.

Una de las alternativas de unión: *(x-1,y,z)*, tiene el mismo residuo que la semilla por lo tanto, se agrega el residuo de la partícula caminante al arreglo de residuos y se incrementa la estructura en una partícula.

Ahora para cada nueva partícula caminante se deberán verificar sus seis alternativas de unión con las dos partículas (semilla y primer partícula unida) y en caso de que el residuo de alguna de las alternativas esté contenido en el arreglo de *residuos se agregara al arreglo solamente el residuo de la partícula caminante, no el de la alternativa que se probó.* Una tabla más completa quedaría de la siguiente manera:

				RESIDUO	ARREGLO:	ARREGLO_FINAL
SEMILLA	**0**	**0**	**0**	**0**	**0**	**0**
						5
CAMINANTE1	1	0	0	5	**5**	**177**
						354
	2	0	0	10		
	1	1	0	36		
	1	-1	0	-26		
	0	**0**	**0**	**0**		
	1	0	1	182		
	0	0	-1	-177		
CAMINANTE2	0	0	1	177	**177**	
	1	0	1	182		
	0	1	1	208		
	0	-1	1	146		
	-1	0	1	172		
	0	0	2	354		
	0	**0**	**0**	**0**		
CAMINANTE3	0	0	2	354	**354**	
	1	0	2	359		
	0	1	2	385		
	0	-1	2	323		
	-1	0	2	349		
	0	0	3	531		
	0	**0**	**1**	177		
CAMINANTE4	2	0	0	10		
	3	0	0	15		
	2	1	0	41		
	2	-1	0	-21		
	1	0	0	5		
	2	0	1	187		
	2	0	-1	-167		

Se obtiene la función hash de cada una de las alternativas de unión a la estructura. Si alguno de los residuos de estas alternativas esta contenido dentro del arreglo de los residuos, de las partículas que ya están formando a la estructura, entonces se une a dicho arreglo y la estructura se incrementa en una partícula. Éste procedimiento se repite hasta que la estructura tiene un tamaño suficientemente grande para obtener de ella datos estadísticos importantes.

La ventaja del algoritmo, es que no compara coordenadas, sino que compara residuos, es decir trabaja con arreglos numéricos de una sola dimensión. El algoritmo compara el residuo de las seis opciones de unión de la partícula caminante, con los residuos de las partículas que ya están formando parte de la estructura. Si se encuentra que hay un residuo igual, la partícula pasa a formar parte de la estructura, pero si son diferentes, la partícula caminante es desechada y una nueva se genera, es decir, continua su caminata.
La velocidad con la que se pueden hacer estas comparaciones es muy rápida, dado que solo se trabaja con arreglos de una dimensión.
Por otro lado, si fuera necesario trabajar con cuatro dimensiones o más, el arreglo de índices se obtendría de la misma forma. Con ello garantizamos que independientemente de la dimensión de trabajo, el arreglo de índices seguirá siendo de una sola dimensión. Por ésta razón, es posible afirmar que el tiempo de comparación es independiente de la dimensión de trabajo.

Se hace ahora la siguiente consideración: si a un residuo le correspondiera más de una posición sucedería algo como lo siguiente:

residuos	Posiciones correspondientes
1	(1,2,7), (7,5,3)
...	
10	(4,6,2),(3,7,6),(9,0,8)
....	
15	(1,6,2),(6,7,7),(0,0,1),(6,9,0),(0,9,0)
....	
19	(4,3,2),(3,5,6),(9,1,8),(0,7,0)

Como se puede ver, para un solo residuo existen más de una posición relacionada. Esto todavía disminuye más el tiempo de CPU, porque con una sola comparación se verifica más de una posición de las que ya están formando la estructura. En el ámbito informático, esto es una colisión. Se corre el riesgo de que a un solo residuo le correspondan la totalidad de las partículas que forman a la estructura. Sin embargo, en el capítulo tres, se mencionó que estos riesgos se disminuyen cuando se elige un valor primo para las constantes, en este caso *A,B,C.*

Debido a las ventajas que se mencionaron, en este trabajo se utilizan las tablas Hash. Así como una función Hash que minimiza las colisiones.

Utilizando este algoritmo se logran obtener estructuras como las que se muestran a continuación:

FIG. 5.3 ESTRUCTURA OBTENIDA CON EL ALGORITMO DE HASH .

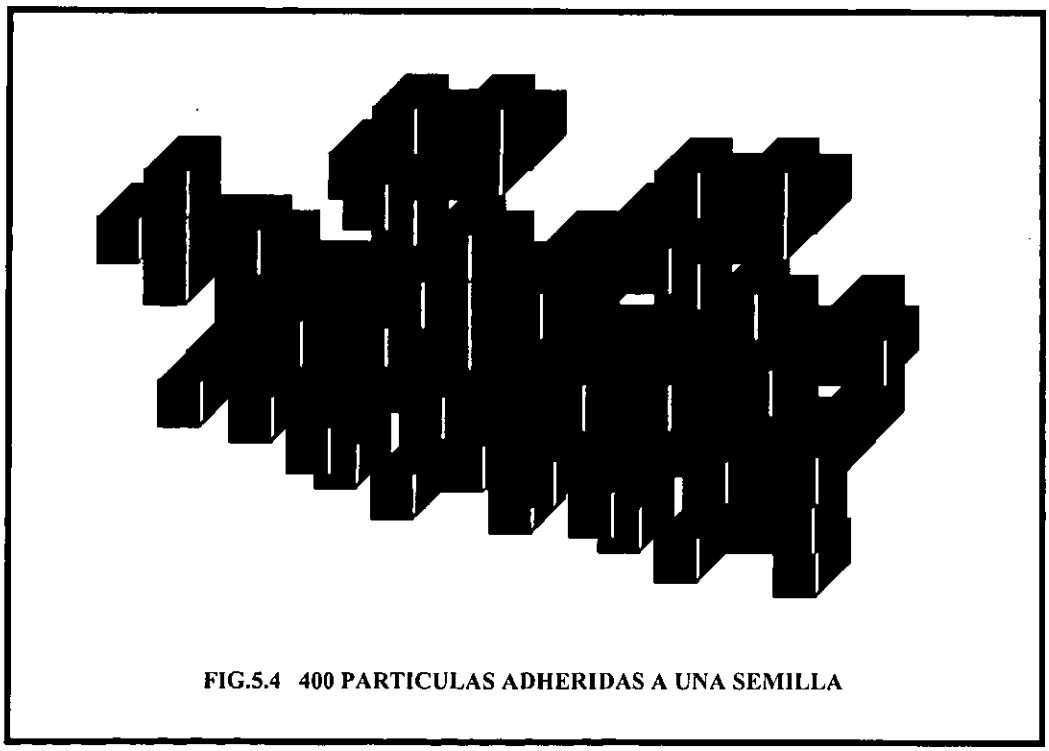


FIG.5.4 400 PARTICULAS ADHERIDAS A UNA SEMILLA

5.2- REFERENCIAS BIBLIOGRÁFICAS DEL CAPITULO 5

1.- M. Kolb, R. Botet, R Jullien, Physical Review Letter 51 (1983) 1123.

2.- B. Flamig Practical Algorithms in C++ John Wiley & Sons, Inc. U.S.A 1995.

3.- N. Madras, A.D. Sokal, J. Stat. The pivot algorithm; a highly efficient monte Carlo method for self avoiding walk. Phys., 1988, vol. 50, ½ , 109.

4.- V. Tchijov, A. Keller, S. Rodriguez-Romo Revista Mexicana de Física Jan 6. 1997.

5.- L. Pietronero, E Tosatti (Eds) Fractal in Phisical, Elsevier. Amsterdam (1986)

6.- R. Julliet, R. Botet, Aggegation and Fractal Aggregates, Word Scientific. Singapure

7.- V. Tchijov, S. Nechaev, S. Rodriguez-Romo, JETP Letter. 64 (1996) 498

8.- H. E Stanley, Journal Physical A10 (1977) L211

9.- K. Binder, D. W Heermann, Monte Carlo simulation in Statistical Physics. Springer Verlag, Berlin, Heidelberg. (1992).

CAPITULO VI MODELO DLA BICOLOR EN TRES DIMENSIONES

6.1- NUEVO MODELO DLA BICOLOR EN TRES DIMENSIONES

Desde el modelo original de Witten y Sander**[1]**, diversos investigadores han propuesto modificaciones y nuevos modelos ajustándolos a ciertas áreas del conocimiento. Sin embargo, el modelo original es tan noble que es posible imaginarse una amplia gama de variantes. Lo que resulta difícil es encontrar algún comportamiento fuera de lo común y de interés suficiente como para hacer investigación original.

Como ya fue expuesto, una amplia variedad de fenómenos naturales y también una amplia gama de actividades humanas pueden considerarse como fenómenos cuyo crecimiento se ajusta a lo que conocemos como *DLA*; la coagulación de aerosoles, algunos crecimientos de cristales, flujo de fluidos en medios porosos, movimientos intermedios entre líquidos con viscosidad diferente, etc. **[2,3,4].**

Debido a la flexibilidad que presenta el modelo de Witten y Sander, el fenómeno DLA se ha expandido y diferentes investigadores han propuesto novedosos modelos cuyo objetivo ha sido estudiar el comportamiento con y sin enrejado, la dimensión fractal y también la simetría de las estructuras obtenidas experimentalmente**[4,5].** La totalidad de los resultados han coincidido en que el estudio del fenómeno DLA no es sencillo y menos lo es si se pretende formular un modelo teórico. Por tanto, es mucho más sencillo y rentable intentar formular un modelo computacional que permita obtener información importante del fenómeno.

Ya en 1996, Tchijov, Nechaev y Rodríguez Romo**[12]** propusieron el *Colored DLA model*, que se describe líneas más adelante, basados en el modelo original de Witten y Sander. Los resultados experimentales de sus simulaciones, pusieron especial atención a la distancia que separaban a las dos semillas iniciales del objeto de su estudio.

Se propone aquí el desarrollo del modelo de Tchijov, Nechaev y Rodríguez Romo. Esta variante del modelo inicia con dos partículas de diferente color situadas a una distancia d una de la otra. Una partícula se genera a una gran distancia de las dos partículas originales de diferentes tipo (colores). Esta partícula se genera con una probabilidad p de ser del color de una de las partículas utilizadas como semillas y una probabilidad $1-p$ de

ser del color de la otra semilla. Esta partícula empieza a *caminar* aleatoriamente, es decir, llevar a cabo un movimiento browniano, hasta que una de las siguientes cosas ocurren:

1.- Llegando a los límites predeterminados del reticulado, es eliminada y una nueva partícula se genera en algún lugar aleatorio. Los límites son arbitrarios, pero dependen de la potencia de la computadora utilizada.

2.- En su caminata aleatoria ocupa un lugar adyacente a la semilla del color opuesto al que le corresponde a ella, es eliminada y una nueva se genera para repetir la secuencia.

3.- En su camino aleatorio, ocupa un lugar adyacente a la semilla de su mismo color, entonces se incrementa el cúmulo en una partícula y una nueva se genera para continuar el proceso.

Estos casos se repiten hasta que alguno de los conjuntos de partículas alcanza un tamaño suficientemente grande para generar resultados de interés.
Para este modelo, dos variable son las que se consideran importantes:

1.- La distancia d entre las dos semillas.
2.- La probabilidad p para una partícula de tener un color.

El siguiente dibujo hace referencia a estas variables. También aquí se aprecia la flexibilidad que presenta el modelo, pues es factible incluir más variables, dependiendo del fenómeno que se pretenda estudiar:

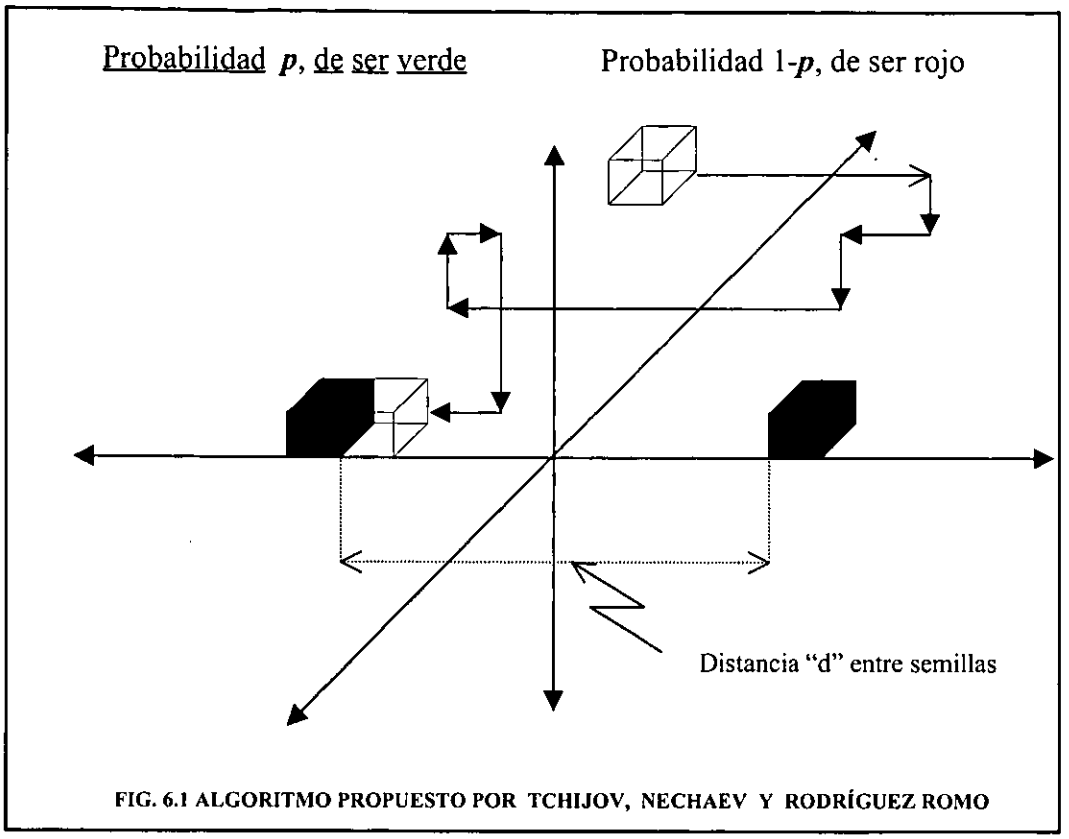


FIG. 6.1 ALGORITMO PROPUESTO POR TCHIJOV, NECHAEV Y RODRÍGUEZ ROMO

Los vértices inferior derecho de la semilla verde y el vértice inferior izquierdo de la semilla roja, están a la misma distancia del origen cartesiano tridimensional.

Las partículas de dos colores diferentes se suponen colocadas dentro de un enrejado tridimensional, a una distancia *d* una de la otra.

A una gran distancia de estas *partículas semilla*, en un punto aleatorio sobre una esfera *S* de radio *Rd*, una nueva partícula es introducida con probabilidad *p* de ser de color verde (según dibujo) y una probabilidad *1-p* de ser de color rojo. El radio *Rd* es mucho más grande que la distancia que separa a las *partículas semilla*, *Rd >> d*. De hecho, es uno de los parámetros implícitos del modelo.

Como se mencionó anteriormente, los dos parámetros considerados en esta simulación son la distancia *d* entre las semillas de cada una de las formaciones fractales y la probabilidad *p* de que la partícula *caminante* sea de uno u otro color. Se restringe el estudio a valores de *p* entre 0.1 y 0.5, y valores para *d* iguales a 2, 10, 30, 50 y 100 unidades del enrejado.

Para cada par d y p, se corrieron 50 experimentos, mismos que se detuvieron cuando alguna de las dos estructuras fractales alcanzo un número de partículas de 3000. La siguiente figura ilustra este concepto.

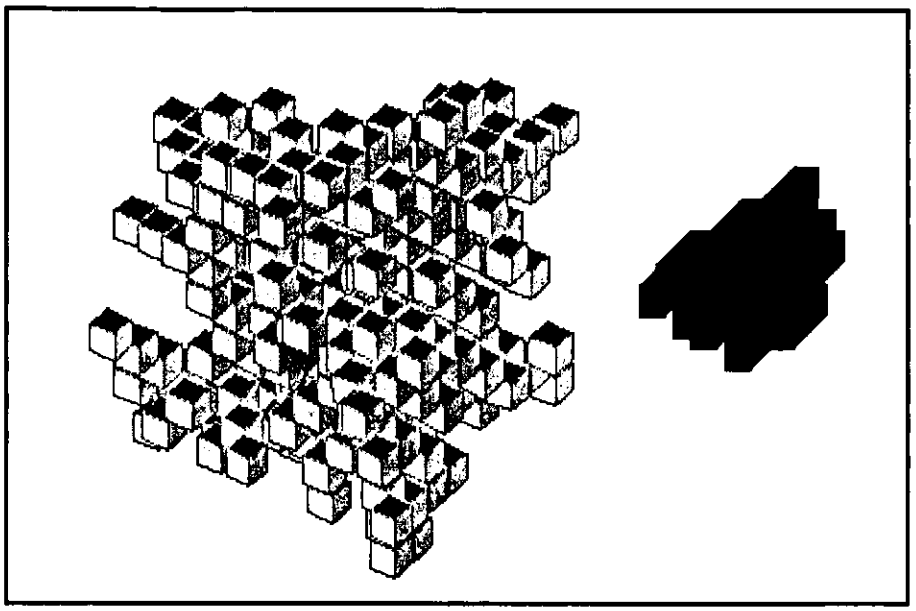


FIG. 6.2 RESULTADO DE LA APLICACIÓN DEL ALGORITMO DESCRITO

De los experimentos realizados, se considera importante la cuantificación de la razón entre el número de partículas del fractal de color 2 y el número de partículas del fractal de color 1: (N2/N1).

La gráfica que se genera al considerar la cantidad (N2/N1) comparada con el número de experimento, en cada par p y d, proporciona una visión de la manera como influye la distancia d y la probabilidad p en la cantidad de partículas que forman a cada color de la estructura que se forma.

Se calcula también el radio de giro *Rg* mediante:

$$Rg^2(N) = \frac{N-1}{N} Rg^2(N-1) + \frac{N-1}{N^2}(Rc(N-1) - R_N)^2$$

Donde:

$$Rc(N) = \frac{N-1}{N} Rc(N-1) + \frac{1}{N} R_N$$

Para cada formación fractal y es posible graficar esta cantidad contra en número de partículas del fractal (N), esto permite visualizar como se incrementa el radio de giro *Rg* al incrementarse el tamaño del fractal.

En los experimentos computacionales utilizamos el procedimiento para generar partículas propuesto en **[6].**

El programa principal del presente fue escrito en Borland C++ y ejecutado en un procesador AMD586 a 133 Mhz, se consideraron los siguientes valores para la distancia *d= 2,10,40,60 y 100 unidades cubicas del enrejado.*

Para p se utilizaron los siguientes valores: *p=0.1, 0.2, 0.3, 0.4 y 0.5.*

Para cada par d y p se generaron 50 experimentos que a su vez dieron lugar a la misma cantidad de estructuras fractales bicolores, donde la estructura de color más grande estuvo formada por 3000 partículas.

Varios experimentos se repitieron en un sistema Pentium MMX a 266 Mhz bajo sistema operativo LINUX, con un número máximo de partículas de 30000. Los resultados obtenidos bajo estos experimentos fueron básicamente los mismos.

6.2- DESCRIPCION DE LAS ESTRUCTURAS OBTENIDAS

Para una distancia $d=100$ y una probabilidad $p=0.5$, las formas de las estructuras y la cantidad de partículas que formaron a cada color permanecieron básicamente simétricas y se muestran separadas, es decir, el espacio que ocupa una, no invade el espacio que ocupa la otra.

Observe la siguiente figura, la totalidad de los experimentos con probabilidad igual a 0.5 y con una distancia de 100, se comportaron básicamente como lo muestra la figura siguiente, aquí se puede apreciar de forma visual que se mantiene una cierta simetría en cuanto a forma y cantidad de partículas que forman cada estructura.

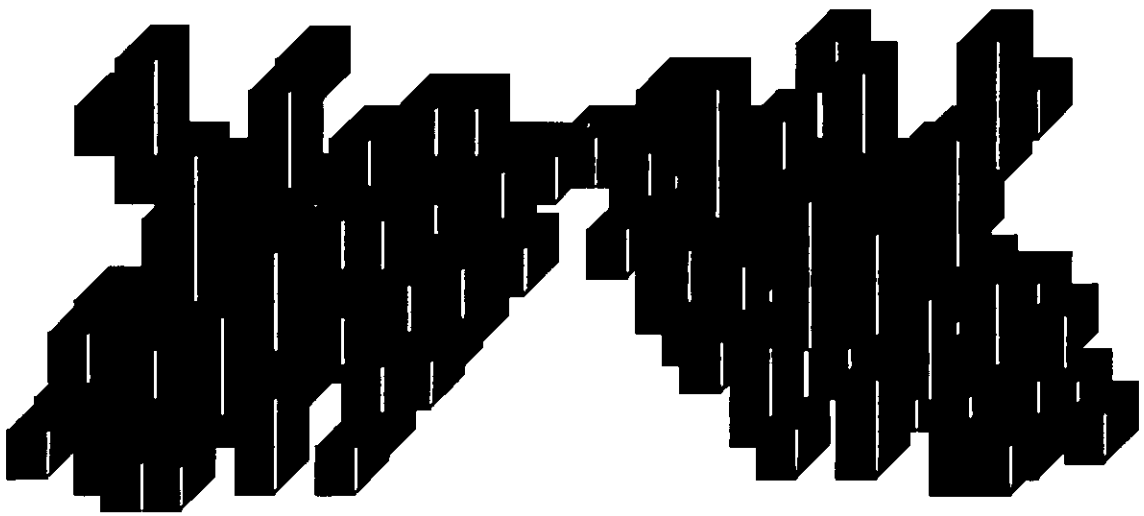


FIG. 6.3. CASO GENERAL, $p = 0.5$ Y $d = 100$. LOS CASOS MUESTRAN SIMETRIA

La figura es muy diferente cuando se mantiene p=0.5, pero la distancia entre semillas se modifica a un valor igual a 2.

En los experimentos, se obtuvieron básicamente dos casos bien diferenciados: por un lado se obtuvieron figuras que pueden ser consideradas simétricas, debido a que

aproximadamente los dos colores presentan la misma cantidad de partículas adheridas. El siguiente caso, muestra un crecimiento mucho mayor que el otro color. Las figuras siguientes muestran los casos encontrados cuando se utilizaron este par de parámetros. Es evidente que en este caso se observa una inestabilidad de creación de estructuras bicolores en etapas iniciales. El 10% de los casos una de las estructuras fue totalmente inhibida por el crecimiento de la otra. Las siguientes figuras muestran gráficamente estos resultados.

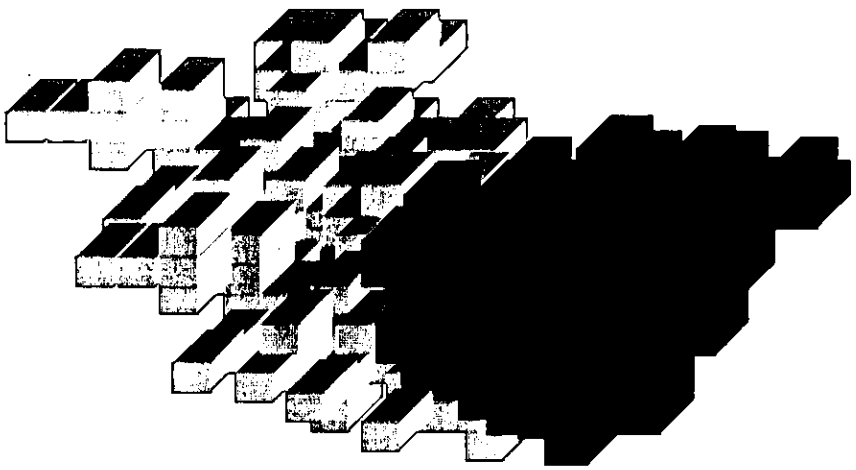


FIG.6.4 CASO GENERAL. $p = 0.5$ Y $d = 2$

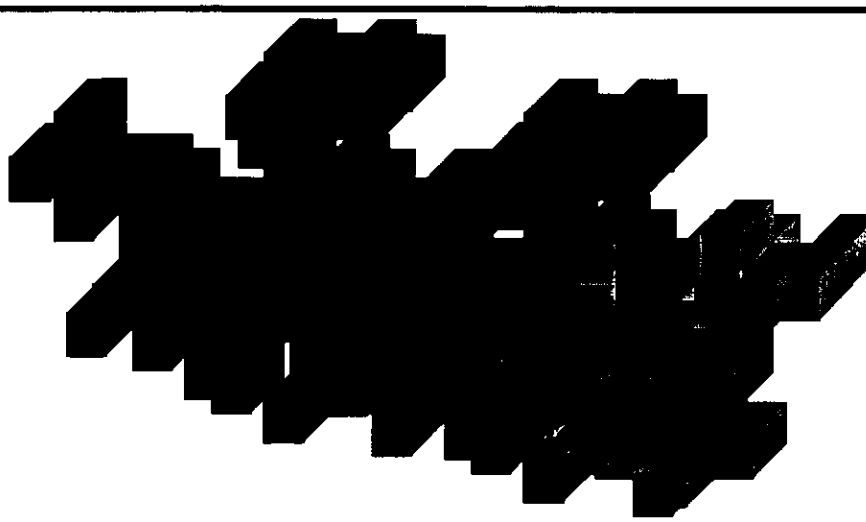


FIG.6.5 10% DE LOS CASOS, CON $p = 0.5$ Y $d = 2$.

6.3- RESULTADOS

Para los diferentes valores de *p= 0.1, 0.2, 0.3, 0.4, y 0.5* y para los diferentes valores de *d= 2,10,40,100,* para cada experimento hecho se calculó la razón de distribución del número de partículas del color uno (N1) y el número de partículas del color 2 (N2), (N2/N1).

En todos los experimentos, excepto para el caso de p=0.5, en número N2 fue más grande que el número N1, por lo tanto: (N2/N1)>1.

En la gráfica No.1, se presenta el caso de p=0.5 y d=100, visualizando la cantidad N2/N1 vs el número de experimentos realizados. En la gráfica No.2, se presenta el caso para p=0.5 y d=2, visualizando también la cantidad (N2/N1) vs el número de experimentos realizados. Aquí es importante ver las diferencias entre un gráfico y el otro, mientras que la gráfica No1 se comporta estable, es decir, su desviación no parece ser significativa.

En el caso de la gráfica No. 2, el comportamiento es muy inestable, existen puntos en la gráfica que muestran una desviación del valor 1 grande.

Para poder mostrar los resultados más completos definimos las siguientes igualdades:

$N_{min}=min(N1,N2)$ y $N_{max}=max(N1,N2)$

De esta manera el valor positivo de *(Nmax/Nmin)* puede considerarse como una *medida* de la simetría o asimetría de los dos colores que forman a la estructura y por tanto, una condición para que una estructura sea simétrica es que se cumpla:

$log(N_{max}/N_{min})=0$. Observar ahora la gráfica No.3, ahí es posible visualizar el comportamiento de todos los experimentos realizados, como puede apreciar, el modelo presenta asimetría cuando la distancia *d* es pequeña. La asimetría disminuye a medida que se incrementa el valor *d.*

Con en fin de evaluar la probable influencia en la compentencia por el crecimiento de ambas estructuras, es decir *el color A y el color B,* es posible calcular la dimensión fractal de ambas estructuras, al menos en el caso más interesante de los experimentos: *p=0.5 y d=2.* Finalmente, se calcula el radio de giro R_g, calculado con la siguiente relación:

$$R_g{}^2 = \frac{1}{N}\sum_{i=1}^{N}(Ri - Rc(N))^2$$

En donde:

Ri es el radio – vector de la i-esima partícula.

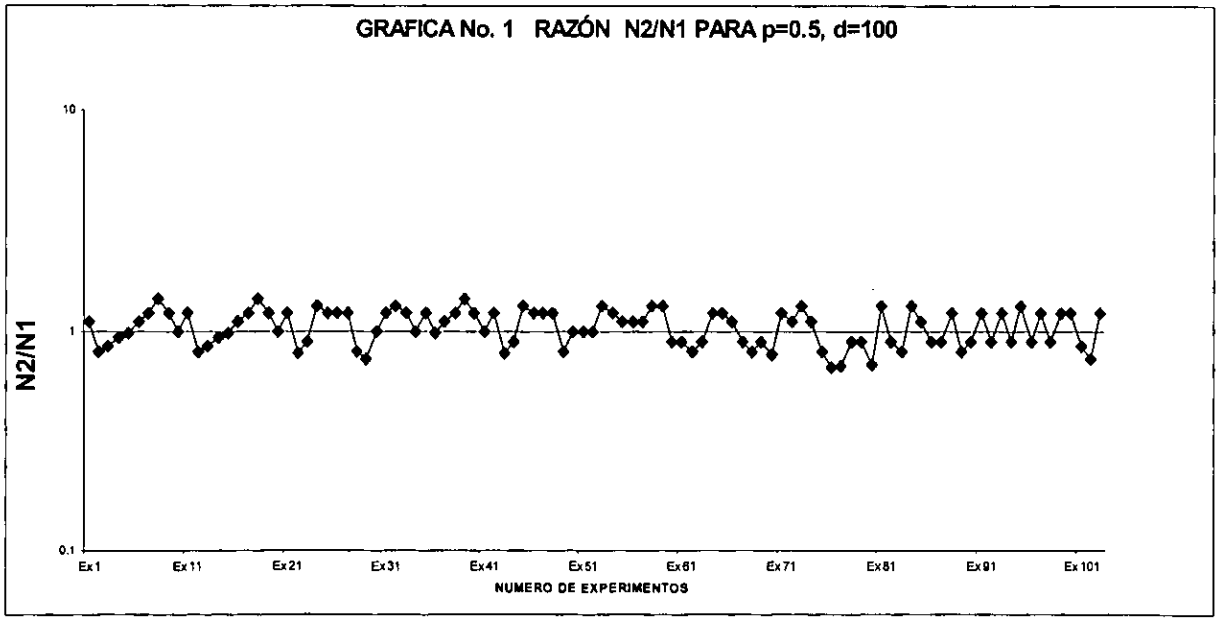
Rc(N) es el radio – vector del centro de la masa del fractal, considerando que cada partícula tiene una masa igual a 1. En una función de recurrencia, se define a Rg como:

$$Rg^2(N) = \frac{N-1}{N} Rg^2(N-1) + \frac{N-1}{N^2}(Rc(N-1) - R_N)^2$$

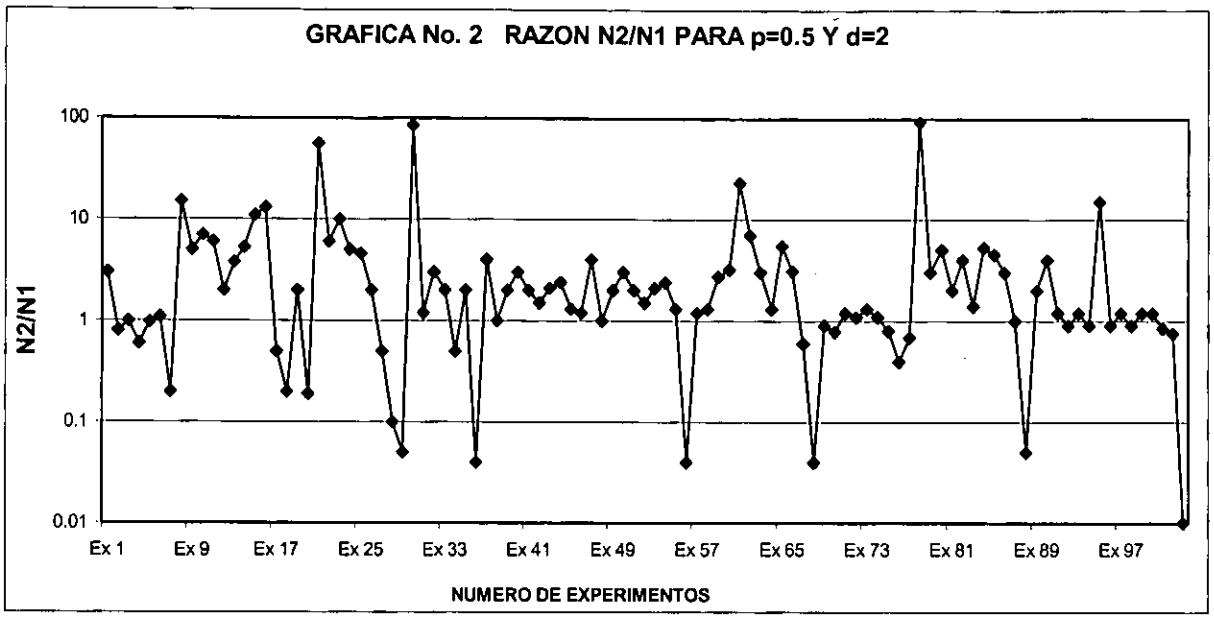
En donde:

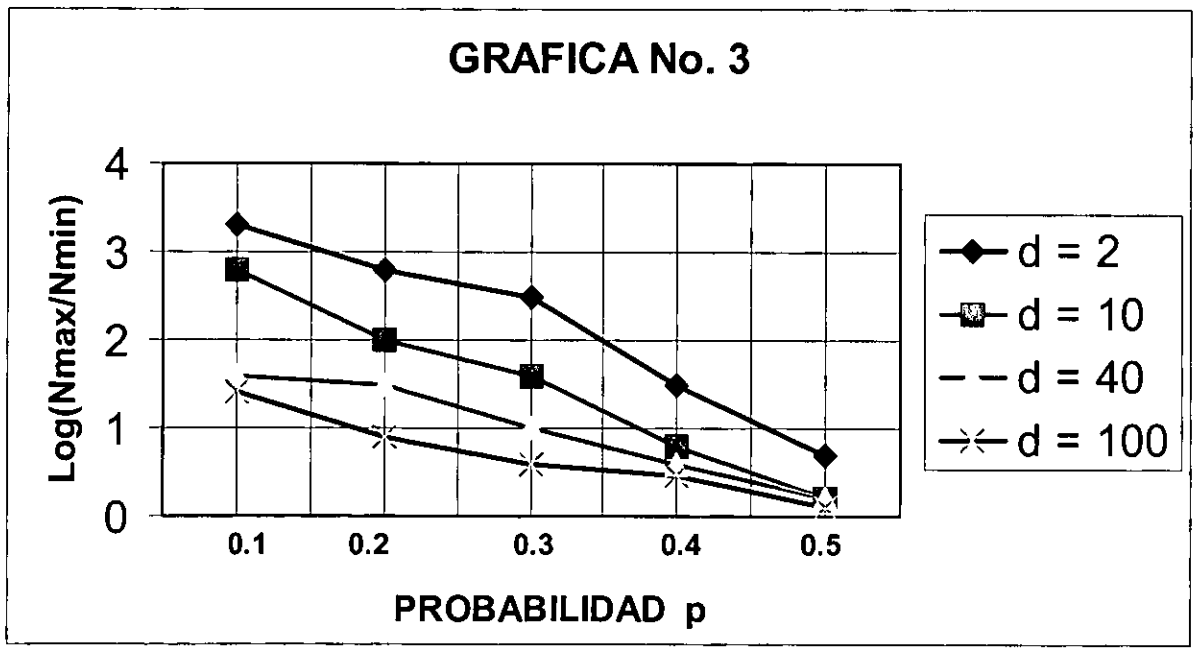
$$Rc(N) = \frac{N-1}{N} Rc(N-1) + \frac{1}{N} R_N$$

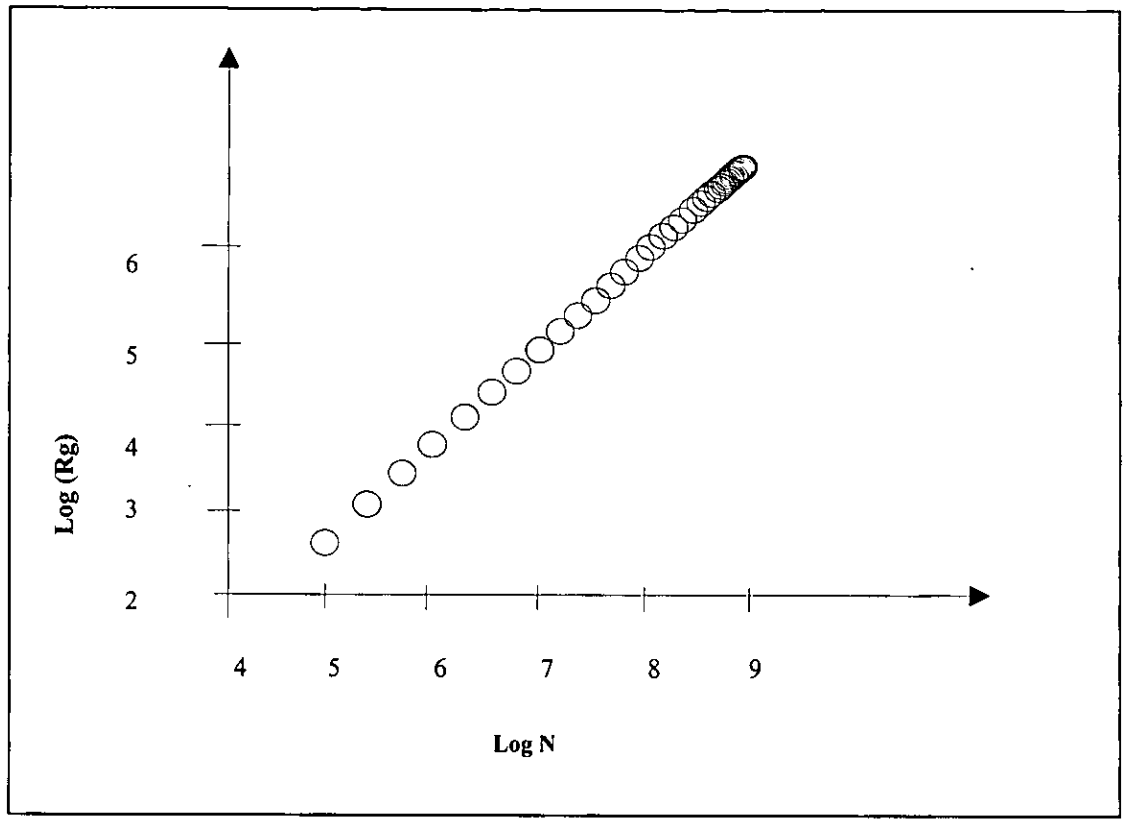
Es posible graficar *N* vs R_g, se muestra el resultado en la gráfica No. 4. Con la estructura de color A. En el gráfico se aprecia que el radio de giro se incrementa cuando se incrementa el tamaño de la estructura. El mismo comportamiento se observa con la estructura de color B. Estos resultados confirman que en la competencia de crecimiento de



las estructuras de color A y de color B, no afecta la dimensión fractal de cada una de ellas.






Log (Rg)
6
5
4
3
2
4
5
6
7
8
9
Log N

6.4- REFERENCIAS BIBLIOGRÁFICAS DEL CAPITULO VI.

1.- T.A. Witten, L.A Sander, PhysicaL Review Letter 41 (1981) 1400.

2.- L. Pietronero, E. Tosatti (Eds.) Fractal in Physics, Elsevier. Amsterdam (1986)

3.- R. Julliet, R. Botet, Aggegation and Fractal Aggregates, Word Scientific. Singapure

4.- T. Vicsek, Fractal growth phenomena, 2nd edition, World Scientific. Singapure (1987)

5.- M. Eden, in Proc. of the Forth Berkeley Symposium on Math. Static and Probability,IV (1961) 223.

6.- V. Tchijov, J. Keller, S. Rodríguez Romo, Revista Mexicana de física Jan 6 1997

7.- T. Vicsek, Phys. Rev. Letter 53 (1984) 2281.

8.- P. Meakin, J. Phys. A18 (1985) L661.

9.- R.C Ball, R.M Brady J. Phys. A18, 1985 L809- L813.

10.- H. T. Herrmann, Physical Reports 136 (1986) 153

11.- P. Meakin, In: L. Pietronero, E. Tosatti (Eds.), Fractal in Physics, Elsevier. Amsterdam (1986) .

10.- P. Grassberger , R. Procaccia, American Physical Rev v.28 No.4 october 1983.

11.- H.B Rosenstock and C. L Marquardt Phys. Rev B.22 5797 (1980)

12.- V. Tchijov, S. Nechaev, S. Rodriguez-Romo, JETP Letter. 64 (1996) 498

13.- H. E Stanley, Journal Physical A10 (1977) L211

14.- K. Binder, D. W Heermann, Monte Carlo simulation in Statistical Physics. Springer Verlag, Berlin, Heidelberg. (1992).

15.- P. Meakin , Physical Review Letter 51 (1983) 1119

16.- Hull, T.E., and Dobell, A.R "Random Numbers Generators", SIAM Rev.IV No.3 July 1982, 230,255.

17.- J. Jhonson, Econometric Methods, N.Y. Mc_Graw HillsCo. 1973.

18.- M. Wolf, Physical Review E47(1993) 1448.

19.- M. Wolf, Physical Review E43(1991) 5504.

20.- I. Vattalainer, T. Ala-Nissila, K. Kankaala, Physical teste for random numers in simulations. Phys, Rev. Letters, 1994, V.73, N19, 2573 – 2516.

21.- N. Madras and A. D Sokal, J. Stat. Phys. 50, 109 (1988).

22.- P. Meakin, P. Ramanlal, L. M Sander, R. C Ball, Physical Review A34 (1986) 5091. Physical Review Letter A27 (1983) 2281

23.- P. Meakin, Z. Djordjevic, Journal Physical A19 (1986) 1271

24.- P. Meakin,T. Vicsek, Journal Physical A20 (1987) L171

25.- P. Meakin, J. Kertéz, T. Vicsek, Journal Physical A21 (1988) 1271

26.- P. Meakin, Journal Physical A21 (1988) 3501

27.- M. Kolb, R. Botet, R Jullien, Physical Review Letter 51 (1983) 1123

28.- B. Flamig Practical Algorithms in C++ John Wiley & Sons, Inc. U.S.A 1995.

29.- H. Lauwerier Fractals; Endlessly Repeated Geometrical Figures Sophia Gill - Hoffstädt) ,Princeton University Press Prin.New Jersey 1991.

30.- L. Adams, Programación Gráfica en C. Ed. Anaya Multimed. S.A,México 1991.

31.- M. Wolf, S Schwarzer, S Halvin, P.Meakin, H.E Stanley,Physical Review A46(1992) R3016.

CAPÍTULO VII POSIBILIDADES DEL NUEVO MODELO.

7.1- POSIBILIDADES DEL NUEVO MODELO.

Durante los últimos años se ha intentado utilizar la geometría fractal para explicar el crecimiento de algunos fenómenos que ha la fecha no tienen una explicación convincente. Como se menciono en el capítulo número uno, muchos son los fenómenos naturales y los procesos industriales en los que sus crecimientos resultan ajustarse al fenómeno DLA.

Seria redundante ofrecer una lista de fenómenos cuyo crecimiento puede intentar ser explicado mediante un modelo de *agregación de difusión limitada.* Es mucho más provechoso revisar posibilidad de investigación profunda de temas de interés general utilizando el modelo DLA.

7.2- CRECIMIENTOS CANCERÍGENOS.

Recientes investigaciones[1 – 3] han dejado abierta la posibilidad de determinar si un tumor es maligno o no, utilizando la dimensión fractal de dicho tumor como punto de diagnóstico.

Esta posibilidad se base en el hecho de que muchos especialistas en la materia, diagnostican si un tumor es maligno o benigno apoyados únicamente en la forma geométrica del tumor. Cuando la superficie del tumor es muy regular, es muy poco probable que resulte maligno. De manera contraria, cuando la superficie tiene un aspecto rugoso y con múltiples ramificaciones, la probabilidad de que sea maligno es más alta.

Aunque apoyados en su experiencia, esta manera de diagnosticar tiene algunos inconvenientes:

1.- Para poder determinar si la superficie del tumor es o no irregular, es necesario hacer procedimientos invasivos al paciente.

2.- Aún teniendo a la vista el tumor, todavía es necesario practicar *el estándar de oro,* es decir, una biopsia. Solo hasta entonces es posible determinar la malignidad o no del tumor.

3.- Aún, es posible evitar un procedimiento invasivo y en su lugar obtener una resonancia magnética, sin embargo, los datos que proporciona son datos de imagen poco precisos, como pueden ser la forma y el tamaño. Estos datos todavía

no permiten diagnosticar la malignidad, al menos no es así en una etapa temprana. Si este fuera el caso, se recurre otra vez a la biopsia.

La siguiente figura, muestra la agresión de un agente cancerígeno al tejido humano.

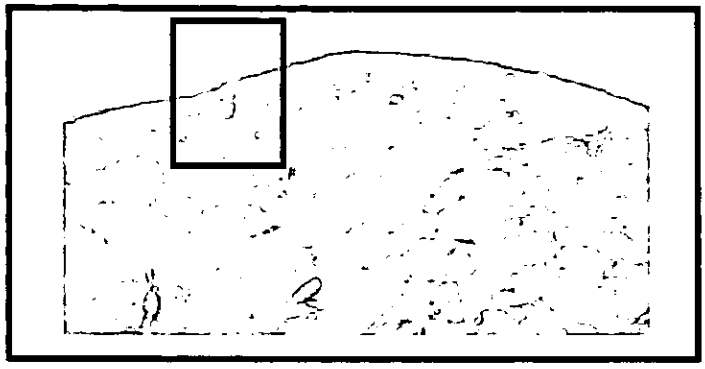


FIG 7.1. MELANOMA MALIGNO.

Una manera alterna de diagnóstico precoz de la malignidad de un tumor, podría ser utilizando la dimensión fractal. Para estos fines, es posible que esta medida pueda considerarse como un índice de la irregularidad del tumor. De esa manera solo sería necesario un estudio de resonancia magnética y se podría evitar un procedimiento invasivo. Las ventajas que se aportarían serían básicamente un diagnostico precoz y no tener que realizar un procedimiento invasivo [4].

7.3- CANCER DE PIEL

El melanoma de piel es otra alternativa para desarrollar más investigación. El modelo DLA, como ya se mencionó, puede ajustarse a dos, tres o más dimensiones. La siguiente fotografía fue tomada de un paciente con melanoma temprano:

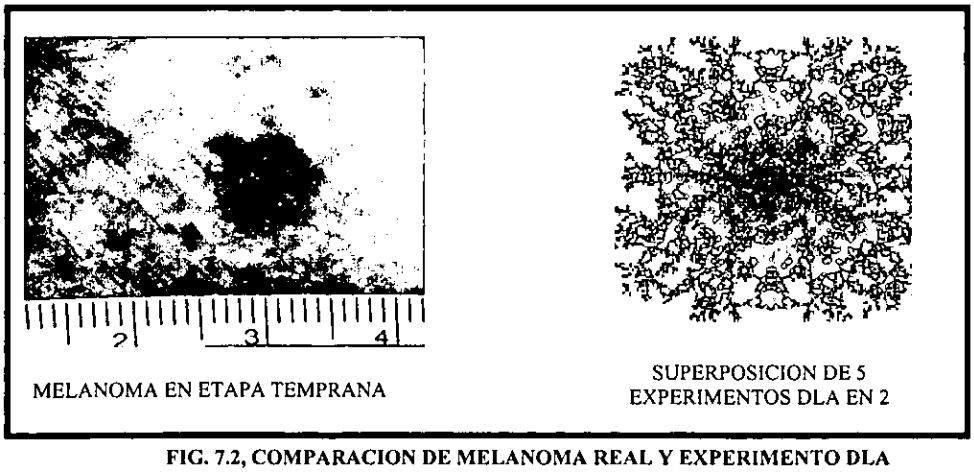


FIG. 7.2, COMPARACION DE MELANOMA REAL Y EXPERIMENTO DLA

Las dos figuras tienen similitud geométrica, y su comportamiento a través del desarrollo, parece ser aleatorio. Motivo suficiente como para pensar en simulaciones computacionales. Las siguientes figuras, corresponden al mismo paciente, solo que se trata de una etapa de desarrollo intermedia y una etapa de invasión total:

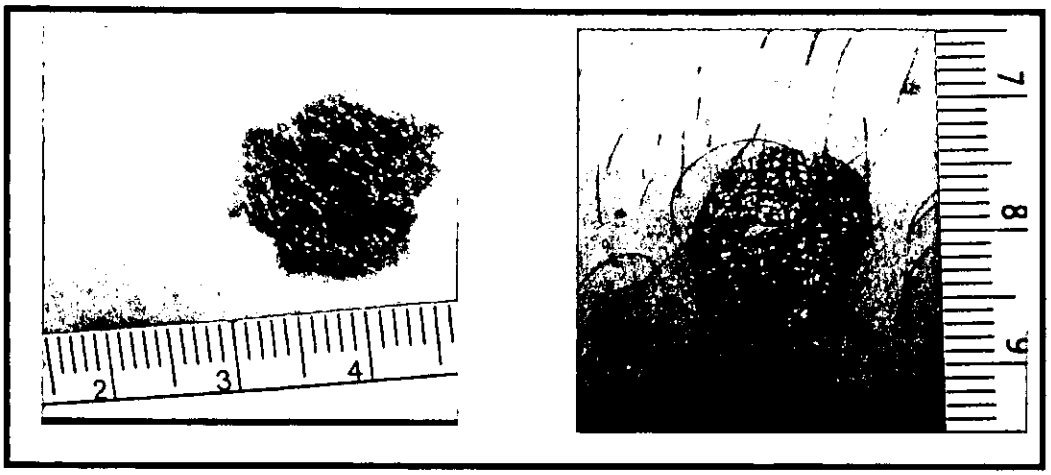


Fig. 7.3 ETAPAS INTERMEDIA Y FINAL DEL MELANOMA

7.4- CANCER DE MAMA.

Se ha estudiado la distribución de cromatina en el núcleo de células malignas y benignas, al igual que muchas otras estructuras, la cromatina se forma mediante un patrón fractal[5]. En estudios recientes, se obtuvieron diferencias en la distribución lacunar de la cromatina detectando diferencias en la dimensión fractal entre células benignas y malignas en forma significativa. Es decir, el objeto del estudio, fue calcular la dimensión fractal de las áreas huecas entre la cromatina. El porcentaje de acierto fue de 95.1 %.

Desde luego en este caso, se sabía cuales pacientes padecían tumor maligno, sin embargo, la aplicación de la geometría fractal resulto exitosa.

7.5- CARIES DENTAL.

Cuando dos puntos de caries dental empiezan en una pieza, por razones que no están bien determinadas, alguno de los dos puntos se desarrolla de manera diferente al otro.

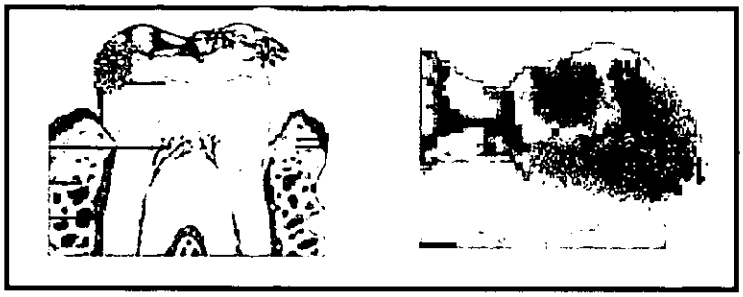


FIG. 7.4 CARIES DENTAL, EN LA PARTE DERECHA UNA SOBRE POSICIÓN

La ilustración de la parte izquierda es una representación del crecimiento de caries en puntos diferentes. El crecimiento aparentemente es aleatorio, sin embargo una estructura crece de manera acelerada comparada con la otra. La parte derecha de la ilustración es una representación de una sobreposición de los dos crecimientos.
Como se puede apreciar, este tipo de crecimiento se adapta al modelo bicolor DLA.

7.6- ESTUDIOS EVOLUTIVOS.

Para poder explicar algunos fenómenos de la evolución se realizó el siguiente experimento: En un medio pobre en nutrientes fueron colocados organismos unicelulares, al paso de unos días, estos organismos aparentemente se reunieron en una especie de simbiosis, formando estructuras como la que se muestra en la siguiente ilustración:

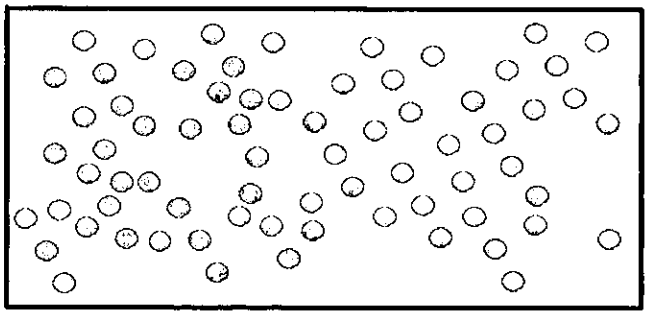


Fig. 7.5 ORGANISMOS UNICELULARES EN UN MEDIO CON NUTRIENTES

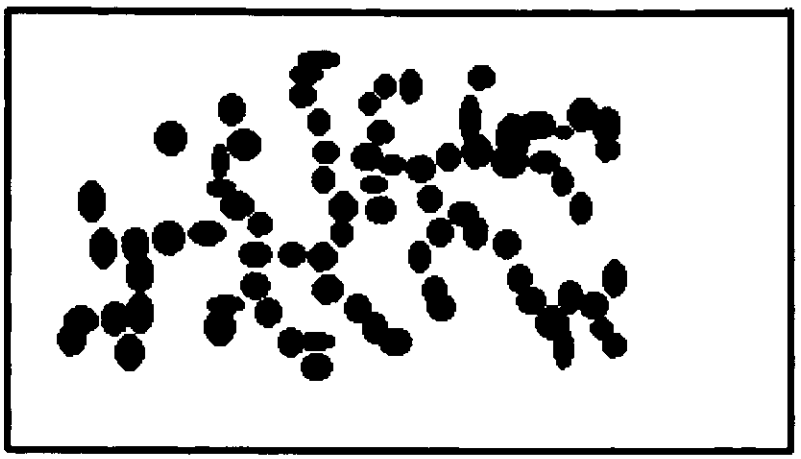


Fig. 7.6 ORGANISMOS UNICELULARES EN UN MEDIO POBRE EN NUTRIENTES

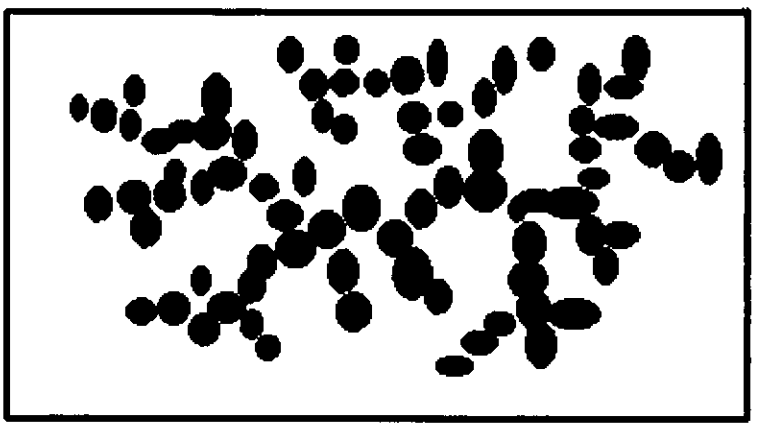


FIG.7.7 ESTRUCTURA TIPICA DE LOS ORGANISMOS UNICELULARES EN SIMBIOSIS

El este experimento concluyo cuando los organismos fueron regresados a un medio muy rico en nutrientes. Entonces se observa que nuevamente se disgregaron los organismos, sin embargo, no todos conservaron sus características iniciales, algunos cambiaron de tonalidad, otros cambiaron de tamaño. Estos resultados hacen supones que probablemente la evolución tuvo un inicio parecido.

Para los fines de este trabajo, solo llamó la atención la forma que tomaron los organismos cuando adoptaron la simbiosis, esta formación es muy parecida a un fractal generado mediante DLA.

7.7- OSTEOPOROSIS.

La descalcificación del sistema óseo es un problema actual y muy común[6]. El modelo DLA es también aplicable a este caso. Observe la siguiente figura:

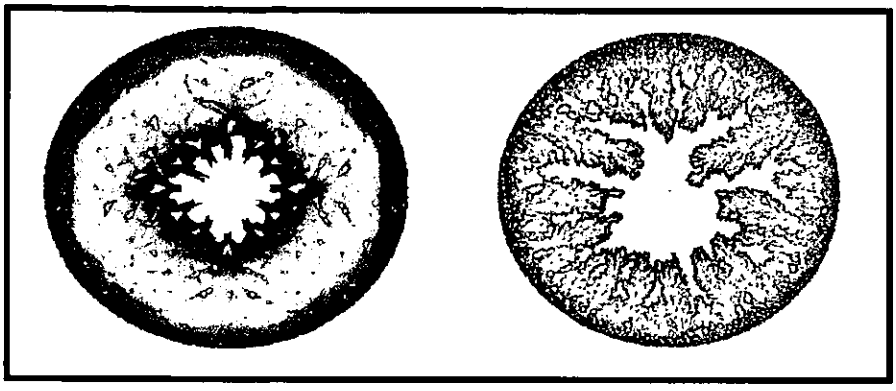


FIG. 7.8 SIMULACIÓN DE DOS ETAPAS DE PÉRDIDA DE CALCIO EN

La parte izquierda de la figura, corresponde al corte transversal, muy delgado, de un hueso, en el se aprecia la parte cavernosa normal en una etapa intermedia de la vida. La parte derecha, muestra un hueso en una etapa avanzada de la descalcificación. Las diferentes tonalidades, corresponden a los diferentes grados de densidad que tiene un hueso. En este caso el azul corresponde al más denso y el rojo al menos denso.

La siguiente figura, corresponde a la recuperación de las partes del hueso que se han perdido por falta de calcio. Observe la similitud con un modelo de crecimiento DLA en dos dimensiones.

Es posible encontrar otras alternativas de desarrollo para el modelo DLA en [7 – 11]. Otros temas relacionados con el modelo DLA y con aplicaciones en medicina pueden encontrarse en [12 – 19].

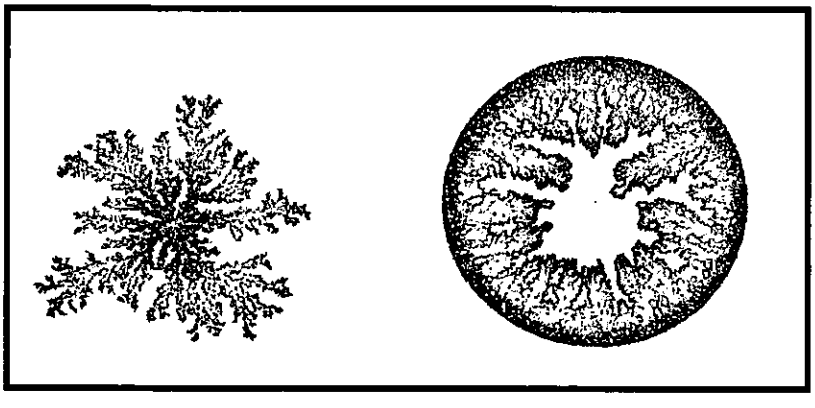


FIG 7.9 PARTE PÉRDIDA DEL HUESO POR DESCALCIFICACIÓN

7.8- BIBLIOGRAFIA DEL CAPITULO 7

1.- G. Landini, J.M Rippin, How important is tumour shape?, J. Pathology, Jun 1996.

2.- A. Sadana, A.M. Beelaram, Antigen-Biotechnology diffusion-Limited binding, a fractal analysis,
Appl-Biochem-Biotechnology. Jun 1996.

3.- T. Schall, Fractalkine; a strange attractor in the chemokine landscape. Immunology Today. April
1997.

4.- N.F. Vittitoe, J.A. Baker, C.E. Floyd. Fractal Texture analysis in computer - aided, Acad-
Radiology. Feb 1997.

5.- S.S. Cross, Fractals in pathology, Journal Pathology. May 1997.

6.- M.A. Haidekker, R. Andresen, C.J. Evertsz,. Assessing the degree of osteoporosis in the axial
skeleton using the dependence of the fractal dimension on the grey level threshold. Br. J.
Radiology. Jun 1997.

7.- J.M. González - González. Fractal strucure of caries. Caries Res 1997.

8.- D.M. Dubois. Generation of fractal fron incursive automata, digital diffusion and wave equation
system. Biosystems, 1997.

9.- A.L. Goldberger. Fractal Fravtal variability vs pathology peridicity: complexity loss and stereotypy
in disease. Perspect- Biol-Med, Summ 1997.

10.- C.F. Jiang, A.P. Avolio. Characterisation of structural changes in the arterial elastic matrix by a
new fractal feature: directional fractal curve. Med. Biol. Eng. Comput., May 1997.

11.- G. Landini. The fractal structure of caries? Caries res. 1997.

12.- J.M. Gonzáez. Fractals in restorative treatment of teeth. Journal Oral Rehabilitation Jan 1997.

13.- J.A. Sherratt, . Mathematicl modelling of cancer invasion and metastasis: an interdisciplinary
workshop held at.... Clin Exp Metastasis. Jan 1997.

14.- R. Sedivy. Fractal tumours: their real and virtual images. Wien-Klin Wochenschr. 1996.

15.- F. Naeim, F. Moatamed, M. Sahimi, . Morphogenesys of the bone marrow: fractal structures and
diffusion limited growth. J. Blood. Jun 1996.

16.- T.G. Smith, G.D. Lange, . Fractal methods and results in cellular morphology-dimension,
lacunarity and multifractals. J Neurosci Methods. Nov 1996.

17.- T. Hagiwara, H. Kumagai, K. Nakumura. Fractal analysis of aggregates formed by heatingdilute

BSA solution using ligth scattering methods. Biosci-Biotechnol-Biochem. NoV. 1996.

18.- J. Peiss, M verlande, W. Ameling. Classifiction of lung tumors on chest radiographs by fractal
exture analysis. Inves. Radiolog. Oct 1996.

19.- A.T. Einstein Physical Review Letters, 12 Jan 1998.

En el presente trabajo aplicamos el algoritmo de las tablas hash para hacer simulaciones computaciones del modelo DLA bicolor en tres dimensiones. De las características de las tablas hash, afirmamos que su utilización en los problemas de la agregación limitada por difusión, puede extenderse a dimensiones superiores. Esta extensión no afecta el tiempo de cálculo porque para cualquier dimensión, siempre se trabaja con un arreglo unidimensional de índices.

En el desarrollo del modelo DLA bicolor en tres dimensiones, nos interesó el comportamiento de las estructuras cuando se varía la distancia que separa a las dos semillas y la probabilidad p de tener uno u otro color. El caso más significativo se presentó con d pequeña y $p=0.5$. Con estos parámetros las estructuras mostraron asimetría. Para el caso en que $d=100$ y $p=0.5$, las formas que adoptaron las estructuras así como la cantidad de partículas que las integraron permanecieron prácticamente constantes. Es decir, ninguna de las estructuras *ganó* la competencia por el crecimiento. En las estructuras generadas con estos parámetros, se observa cierta simetría en las formas.

De las gráficas presentadas en el capitulo 6, se ve que el radio de giro R_g se incrementa cuando crece el tamaño de la estructura. El mismo comportamiento se observa en ambos tipos de estructura (dos colores). Estos resultados permiten afirmar que la competencia por el crecimiento de cada estructura *(color A y color B)*, no afecta la dimensión fractal de cada una de ellas.

En la mayoría de los experimentos, excepto en el caso $p = 0.5$, el número de partículas que formaron el primer color $N1$ fue más grande que el otro color $N2$. Por lo tanto $N2/N1 > 1$.

Las aplicaciones que este modelo puede tener son muy variadas: podemos considerar desde la polución del aire, hasta los crecimientos cancerígenos. Como se menciona en el capitulo 7, la probabilidad de un diagnostico precoz de tumores malignos tomando su dimensión fractal se esta investigando. Si consideramos que también se trabaja sobre nuevas formas de cuantificar la dimensión fractal, se puede esperar que a mediano plazo este modelo pueda aportar conocimientos a estas investigaciones.

DIMENSIÓN FRACTAL

Para poder comprender el concepto de la dimensión fractal, consideremos los siguientes casos:

Utilizamos un segmento de línea y lo seccionamos en cuatro segmentos idénticos.

FIGURA ORIGINAL

EL ORIGINAL DIVIDIDO EN CUATRO SEGMENTOS

Cada uno de los cuatro segmentos es ¼ del tamaño original. Esto quiere decir que cada segmento sería igual al original con solo multiplicarlo por un factor de escala. Para este caso, el factor de escala es igual a cuatro.

Si consideramos que el segmento original está dividido en cuatro segmentos y que el factor de escala es igual a cuatro, podemos escribir:

$No\_segmentos = (factor\_de\_escala)^{D}$

En donde:

No_segmentos = N

Factor_de_escala = S

Dimensión = D

Podemos escribir entonces: $N = S^{D}$

Para el caso de la línea segmentada en 4 secciones: $4 = 4^{1}$

Otro caso es el siguiente:

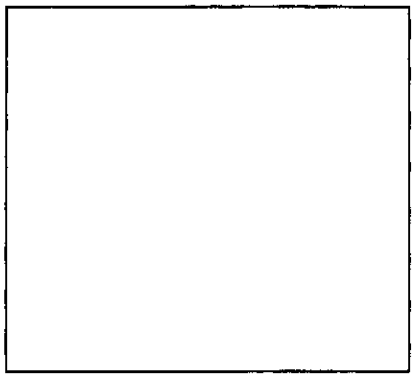


FIGURA ORIGINAL

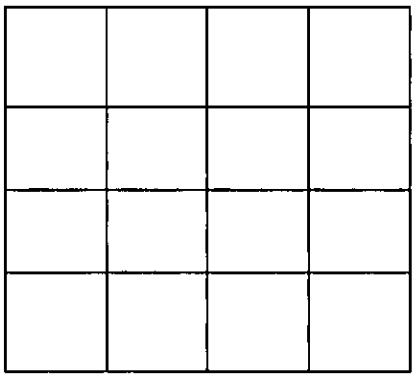


FIGURA ORIGINAL EN 16 SEGMENTOS

Para este caso, el número de segmentos es de 16 y el factor de escala es de 4, pues al multiplicar cada lado de un segmento por cuatro, se obtiene cuadrado original.
Al aplicar la relación anterior:

$16 = 4^2$

La dimensión fractal de esta figura es igual a 2.

Considermeos la siguiente figura:

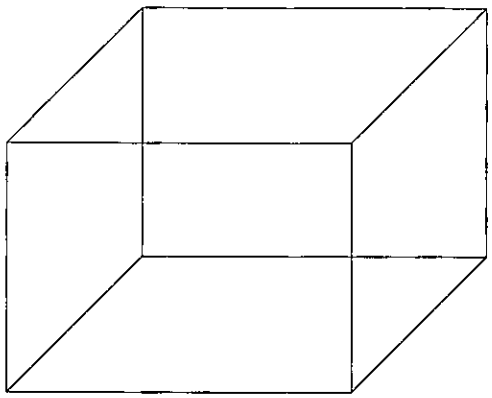


FIGURA ORIGINAL

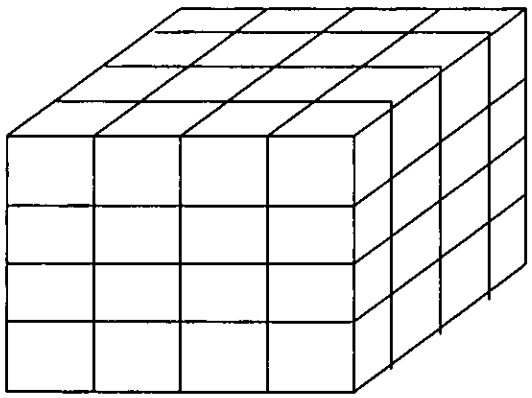


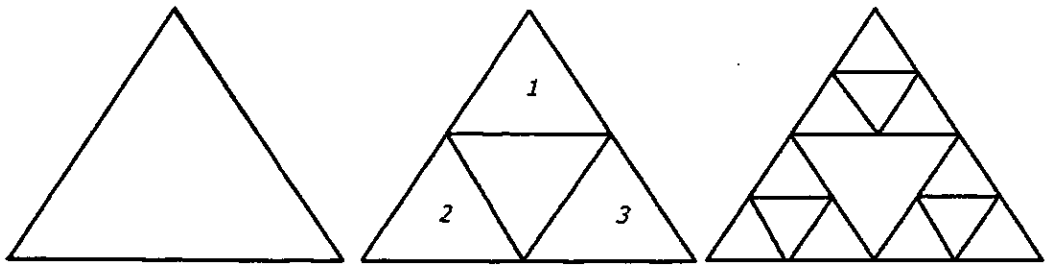
FIGURA ORIGINAL EN 64 SEGMENTOS

Para este caso, el número de segmentos en los que fué dividido el cubo es igual a 64. El factor de escala es igual a 4, dado que al multiplicar cada lado del segmento por 4, se obtiene el cubo original. Aplicando la misma relación que en el caso anterior:

$64 = 4^3$

La dimensión fractal de esta figura es igual a 3. En los tres casos anteriores calcular la dimensión mediante la relación:
$No\_segmentos = (factor\_de\_escala)^{D}$ ha sido sencillo. Sin embargo a medida que la complejidad de la figura se incrementa, el cálculo es más laborioso.

Veamos el siguiente caso:



La última figura se genera a partir de un triángulo equilátero. Se traza una recta a los puntos medios de cada uno de los tres lados, de esa forma se obtiene la figura que intermedia. El proceso se repite una vez más para obtener la figura final. Esta figura se conoce como *el triángulo de Sierpinski.*

Al aplicar la relación: *No_segmentos = (factor_de_escala)* ^{D} a ésta figura obtenemos:

$N = S^{D}$

$3 = 2^{D}$

$\log (2^{D}) = \log (3)$

$D * \log (2) = \log (3)$

$D = \log (3) / \log (2)$

$D = 1.585$

Aquí se ve que la dimensión de esta figura no es entera.

Ésta no es la única forma de calcular la dimensión fractal, pero es una forma de comprender de una forma muy simple el concepto. La forma como se interpreta el resultado fraccionario está en función del investigador, vea capitulo vii. Por ejemplo, se dice que una dimensión de 1.585 indica que la figura no puede existir en una dimensión, y por otro lado, dos dimensiones es mucho espacio. Es posible considerar que la dimensión fractal, es una indicación de la cantidad de espacios euclidianos que ocupa un fractal.

Es posible profundizar más en el concepto de dimensión fractal en las siguientes referencias:

- Falconer K., Fractal Geometry; Mathematical Fundation and Application, John Wiley & Sons, 1990.
- Mandelbrot, BB. The fractal Geometry of Nature, W.H Freeman and Co., New York, 1982.
- http://bang.lanl.gov/video/sunedu/math/fractals/dim.html