Notas:

REFERENCIAS E IMÁGENES:

Todas las Imágenes y definiciones son del autor. En caso de que no fuera así, vendrá acompañada de la correspondiente referencia.

ANGLICISMOS:

Los anglicismos usados en esta tesis se encuentran en el contexto de programación e informática que se suele encontrar en los reportes y documentos sobre algoritmos genéticos. Se han utilizado para tener una correspondencia de vocabulario con la documentación relacionada al tema.

DEFINICIONES:

Ciertas palabras o conceptos pueden ser usados con diferentes propósitos a lo largo de los capítulos de la tesis. Para obtener una definición precisa del concepto o palabra ver el Glosario.

Carlos Ignacio de la Barrera Poblete.

Índice

Objetivo y Justificación de la Investigación 6
Introducción 8
La Naturaleza de la Arquitectura 11
El Relojero Ciego 16
4.1. Las Bases Biológicas 21
4.2. Codificación 22
4.3. Algoritmos Genéticos (AGs) 22
4.4. Métodos de Selección 25
4.4.1. Selección Elitista. 26
4.4.2. Selección de Ruleta 26
4.4.3. Selección de Escalada 27
4.4.4. Selección por Torneo 27
4.4.5. Selección por Rango 28
4.4.6. Selección Generacional 28
4.4.7. Selección por Estado Estacionario 28
4.4.8. Selección Jerárquica 28
4.5. Cruzamiento 28
4.5.1. Cruce de 1 punto 29
4.5.2. Cruce de 2 puntos 31
4.5.3. Cruce Uniforme 31
4.6. Mutación 32
4.7. Funcionamiento de un Algoritmo Genético 35
Diferentes Técnicas de Búsqueda 37
5.1. Calculus based 39
5.1.1. Direct -> Greedy Algorithm 39
5.1.2. Direct -> Binary Search 39
5.1.3. Direct -> Sequence Search 40
5.2. Random 40
5.2.1. Guided -> A.I. -> Taboo Search 40
5.2.2. Guided -> A.I. -> Simulated Annealing 41
5.2.3. Guided -> A.I. -> Hill Climbing 42
5.2.4. Guided -> A.I. -> Evolutionary Algorithms -> Evolutionary Programming .. 43
5.2.5. Guided -> A.I. -> Evolutionary Algorithms -> Genetic Algorithms -> Parallel GAs -> Automatic Parallelism, Coarse Grain, Fine Grain. 44

5.2.6. Guided -> A.I. -> Neuronal Network 45

5.2.7. Guided -> A.I. -> Neuronal Network -> Backpropagation 46

5.2.8. No guided -> Las vegas 46

5.2.9. No guided -> Montecarlo 47

5.3. Enumerative 48

5.3.1. Guided -> Branch and Bound 48

5.3.2. Guided -> Divide and Conquer 48

5.3.3. No guided -> Backtracking 49

5.4. Ventajas de los AGs por sobre otros tipos de algoritmos 50

5.5. Limitaciones de los AGs 51

5.6. Lenguajes 54

Estado del Arte 55

6.1. Selección de trabajos realizados con Algoritmos Genéticos y Técnicas Evolutivas 55

6.1.1. An Evolutionary Architecture, John Frazer (1975) 55

6.1.2. *Lidabashi* Subway Station, Makoto Sey Watanabe Architect's Office, 2000. 57

6.1.3. Architectural Constraints in a Generative Design System: Interpreting Energy Consumption Levels, Luisa Caldas y Leslie Norford (2001) 61

6.1.4. *Genr8*, Martin Hemberg, Una-may O'reilly, Peter Nordin (2001) 71

6.1.5. "ArchiKluge", Pablo Miranda Carranza (2005) 84

6.1.6. Evolutionary Algorithm for Structural Optimization, Voss, Mark S. and Foley, Christopher M. (1999) 85

6.1.7. Recent Advances in Evolutionary Structural Optimization, Xie, Y.M., Huang, X., Tang, J.W. and Felicetti, P. (1992) 87

6.1.8. Coevolving Species for Shape Nesting, Jeffrey Horn, 2005 91

6.1.9. Evolutionary Form-Finding of Tensegrity Structures, Francisco Valero Cuevas y Paul Chandana, 2005. 94

6.1.10. Genetic Algorithms for Construction Site Layout in Project Planning. MAWDESLEY, Michael J., SAAD, H. Al-jibouri y HONGBO, Yang, 2002. 98

6.1.11. Optimización de portafolios accionarios a través de un micro algoritmo genético, GUTIÉRREZ, Mauricio., TORRES, Erick., GÁLVEZ, Patricio., POO, Germán. 2007. 102

6.2. Conclusión del Estado del Arte 106

Elección de Herramientas 107

7.1. Lenguajes de Programación 107

7.1.1. ESTRUCTURA GENERAL DE LENGUAJES DE PROGRAMACIÓN 107

7.1.2. VISUAL BASIC 108

7.1.3. C++ 109

7.2. Software de Análisis 112

7.2.1. *Ecotect* 112

7.2.2. EnergyPlus (E+) 113

7.2.3. DesignBuilder 114
7.2.4. Hourly Load Calculation Program 115
7.2.5. Cymap 115
7.2.6. Energy-10 115
7.3. Programas de Modelado 117
7.3.1. *Rhinoceros* (*Rhino*) 117
7.3.2. *GenerativeComponents* (GC) 117
7.3.3. Digital Project (DP) 118
7.3.4. Maya 119
7.4. Conclusiones de Elección de Herramientas 120
Experimentos 123
8.1. Travelling Salesman Problem como Modelo Generativo Urbano 125
8.1.1. Introducción 125
8.1.2. Método 126
8.1.3. Resultados 133
8.1.4. Discusión 140
8.2. AG como Sistema de Diseño Generativo 144
8.2.1. Introducción 144
8.2.2. Método 146
8.2.3. Resultados 154
8.2.4. Discusión 163
8.3. Organización espacial mediante AG 164
8.3.1. Introducción 164
8.3.2. Método 168
8.3.3. Resultados 176
8.3.4. Discusión 188
8.4. Estrategia Evolutiva para el Diseño de Arquitectura Optimizada 189
8.4.1. Introducción 189
8.4.2. Método 192
8.4.3. Resultados 198
8.4.4. Discusión 200
Conclusiones 203
Bibliografía 207
Glosario 212
Anexos 217
Código de Travelling Salesman Problem como Modelo Generativo Urbano. 217
Código de AG como Sistema de Diseño Generativo. 223
Código de Organización espacial mediante AG 232

Código de Estrategia Evolutiva para el Diseño de arquitectura Optimizada, (solo el bloque que corresponde a *GCScript)* 243

Algoritmos Genéticos como Estrategia de Diseño en Arquitectura

Carlos Ignacio de la Barrera Poblete

Programa de Doctorado "Comunicación Visual en Arquitectura y Diseño"

Escola Tècnica Superior d'Arquitectura de Barcelona

Universitat Politécnica de Catalunya

Director: Dr. Javier Monedero

Barcelona, Septiembre 2010

Dpto. Expresión Gráfica Arquitectónica

ACTA DE CALIFICACIÓN DE LA TESIS DOCTORAL

Reunido el tribunal integrado por los abajo firmantes para juzgar la tesis doctoral:

Título de la tesis: Algoritmos Genéticos como Estrategia de Diseño en Arquitectura.
Autor de la tesis: Carlos Ignacio de la Barrera Poblete.

Acuerda otorgar la calificación de:

- ☐ No apto
- ☐ Aprobado
- ☐ Notable
- ☐ Sobresaliente
- ☐ Sobresaliente Cum Laude

Barcelona, …… de ….....…….. de .....….

El Presidente

.....
(nombre y apellidos)

El Secretario

.....
(nombre y apellidos)

El vocal

.....
(nombre y apellidos)

El vocal

.....
(nombre y apellidos)

El vocal

.....
(nombre y apellidos)

PORTADA:

"*Somos la rama sobre la cual estamos sentados*"

Dibujo de Marcelo Maturana, Tomado del libro "Arquitectura Ciencia y Tao", escrito por Dick Bornhorst, publicado en Caracas y editado por *Ecología y ArTitectura* en 1991. Pág. 78.

DEDICATORIA:

Está tesis está dedicada a todas las personas que me acompañaron de alguna manera en esta tarea. Mi familia y amigos.

Objetivo y Justificación de la Investigación

Este trabajo de investigación consiste en el desarrollo de una serie de Algoritmos Genéticos (AGs) aplicados a diferentes escalas arquitectónicas. El objetivo es demostrar que la estrategia basada en la evolución de las especies puede ser aplicada a la arquitectura, permitiendo proponer soluciones creativas y optimizar cierto tipo de problemas.

La justificación de este trabajo, consiste en incorporar una estrategia evolutiva al proceso de diseño, por medio del ordenador. El cual es la única "máquina" que permite representar de cierta manera los comportamientos biológicos.

En el marco de este doctorado se pretende analizar: ¿Cómo los algoritmos genéticos pueden ser aplicados a la arquitectura para plantear soluciones responsables con el medio ambiente? Para responder a esta pregunta se han construido una serie de pasos que dan la estructura a esta tesis. El primer paso se llama "La Naturaleza de la Arquitectura"; en él se busca exponer la situación actual en la cual se encuentra el planeta y el futuro más probable en los próximos 90 años. El segundo paso se llama "El Relojero Ciego", donde se explican los AGs en profundidad, en que están basados, como funcionan, cuáles son sus partes que lo componen, etc. El tercer paso presenta una clasificación de las diferentes técnicas de optimización más conocidas y cuáles son las ventajas y desventajas de los AGs por sobre estas técnicas. El quinto paso es el "Estado del Arte", que consiste en una recopilación de trabajos sobre AGs aplicados al diseño. El sexto paso corresponde a una evaluación de herramientas y lenguajes de programación que servirán para desarrollar los trabajos del último capítulo. En esta parte se examinarán los diferentes lenguajes de programación para desarrollar AGs, así también los softwares en los cuales se desarrollarán los diferentes experimentos. El último paso extrae la información fundamental de las 6 etapas previas y propone 4 AGs aplicados a diferentes escalas del proyecto arquitectónico.

El primer trabajo busca encontrar la distancia más corta entre una serie de edificios, y dependiendo de la ruta, propone los volúmenes y la orientación de los mismos. Este AG está construido en VBS sobre *Rhinoceros* y está basado en un problema de combinatoria. El segundo trabajo, se centra en la generación de formas. El diseño de una forma arquitectónica es un trabajo bastante complejo, en el cual influyen muchas variables. La idea no consiste en maximizar una de ellas, sino encontrar el mejor balance entre todas las variables que influyen en un proyecto. Este AG está basado en el "*developer's Manhattan function*"[1] y en otras funciones relacionadas con la forma del edificio y el ambiente que lo rodea. El resultado es una forma que responde a todas las restricciones impuestas. El tercer proyecto, está basado en el algoritmo de *Voronoi*, y consiste en resolver el problema al cual nos enfrentamos los arquitectos al "encajar" recintos dentro del perímetro de un edificio. Este AG pide al arquitecto que ingrese las áreas objetivo que necesita dentro del edificio. Luego el AG distribuirá los recintos hasta hacer coincidir al máximo las áreas establecidas por el arquitecto. El cuarto proyecto enfrenta el problema de la sustentabilidad y el consumo de energía al interior del

1 COATES, Paul. Programming.Architecture, New York, Routledge, 2010, pág. 97.

edificio. En este caso un AG buscará encontrar la forma óptima que disminuya los consumos de calefacción y refrigeración en el día más caluroso y más frío del año.

Todos los resultados, mostrados por los experimentos, deben verse como una propuesta de diseño, en la cual el arquitecto puede basarse y continuar desarrollando su trabajo y no como el resultado final de un proyecto. Esto es así porque el proyecto de arquitectura es infinitamente mucho más complejo que un algoritmo buscando un óptimo.

Introducción

La evolución de las especies es la historia de todos los procesos biológicos que han venido ocurriendo en el planeta desde la aparición de la vida hasta hoy. Está historia habla de una serie de estrategias que han tenido que resolver una infinidad de problemas de diferentes características y condiciones. Soportar altas presiones para poder cazar y comer, correr más rápido, organizarse en grupos para escapar del enemigo u organizarse de cierta manera para disminuir el cansancio en viajes largos. La naturaleza ha desarrollado estructuras óseas resistentes a grandes presiones atmosféricas y otras muy livianas para permitir el vuelo.

La selección natural de Darwin, plantea que el medio ambiente puede favorecer o desfavorecer la reproducción de organismos dependiendo de sus características. Los descendientes de los organismos heredan parte de sus características y propiedades que determinan su adaptación al medio ambiente. Por lo tanto, los organismos menos aptos tienen menos opciones de sobrevivir que los más. Iterar este proceso es lo que se conoce como evolución de las especies.

De esta manera la naturaleza ha evolucionado en una rica biodiversidad de especies, todas ellas dependiendo unas de otras y del medio ambiente. Cada forma en la naturaleza es la expresión de cierta información codificada, que bajo ciertos códigos construyen instrucciones que dan origen a la forma. Sin embargo, su expresión final depende de un balance entre el propio código del ser vivo y del medio ambiente en el cuál habita.

El proceso consiste en la producción de una forma que es evaluada y si no cumple con ciertos requisitos, es eliminada. Pero en realidad, lo que se elimina es la configuración del código y no la forma en sí. La forma no es la que evoluciona o se descarta, sino el código genético que se expresa en el organismo. Cuando desaparece una especie del planeta, lo que desaparece es la manera en como estaba ordenado ese código genético. Por otro lado, cuando una especie se adapta, lo que realmente se adapta no es la forma, sino la manera en cómo está organizado el código genético que hizo posible adaptarse a esta especie a las nuevas circunstancias.

En este sentido los seres vivos, como forma, somos presa del código genético estando a su merced y esperando saber si nos adaptaremos o no[2].

También como seres vivos, somos capaces de modificar el medio ambiente que nos rodea, el cual a su vez, influirá en nuestro código genético. Por ejemplo, inventamos el calzado, el cual nos ha protegido los pies durante 5.500 años[3]. Este ha sido uno de los factores que, como muchos otros, ha influenciado la forma de nuestro pie tal como la conocemos hoy en día.

2 Dawkins, Richard. El Gen Egoísta, Salvat Editores S.A., 2ª edición, Barcelona, 2000.
3 http://www.elmundo.es/elmundo/ciencia/1276107270.html [2010/06/20]

La codificación de todos los seres vivos es el ADN (DNA)[4] que es formado por 4 nucleótidos que establecen una asociación específica entre ellos. Debido a su afinidad química entre las bases, los nucleótidos que contienen Adenina se acoplan siempre con los que contienen Timina y los que contienen Citosina con los que contienen Guanina.

Visto de esta manera, el concepto de asociación de los nucleótidos en la cadena de ADN puede ser descrito por medio de un algoritmo. La asociación entre ellos es por su afinidad química, pero siempre se enlazaran A y T y por el otro lado C y G.

La descripción de esta particularidad puede ser descrita con un simple algoritmo en lenguaje C#.

```
{
class ADN
    {
        static void Main()
        {
            int i;

            Random RndV = new Random();
            for (i = 0; i <= 10; i++)
            {
                double Rnd = RndV.NextDouble();
                if (Rnd < 0.5)
                {
                    if (Rnd < 0.5)
                    {
                        Console.WriteLine(" A <---> T");
                    }
                    else
                    {
                        Console.WriteLine(" T <----> A");
                    }
                }
                else
                {
                    if(Rnd < 0.5)
                    {
                        Console.WriteLine(" C <---> G");
                    }
                    else
                    {
                        Console.WriteLine(" G <---> C");
                    }
                }
            }

            Console.ReadLine();
        }
    }
}
```

4 DNA: Deoxyribonucleic acid, El ácido desoxirribonucleico, constituye el principal componente del material genético de la inmensa mayoría de los organismos, siendo el componente químico primario de los cromosomas y el material con el que los genes están codificados.

Pseudocódigo clase ADN

Repetir 10 veces la siguiente operación:

1. Calcula un valor aleatorio entre 0 y 1 (6 decimales).
2. Si este valor es menor a 0.5, ejecutar el primer bloque. De lo contrario ejecutar el segundo bloque.
3. Si se ejecuta el primer bloque, calcular un nuevo número aleatorio entre 0 y 1, si este es menor a 0.5, imprimir “A <---> T” de lo contrario imprimir “T <---> A”.
4. Si se ejecuta el segundo bloque, calcular un nuevo número aleatorio entre 0 y 1, si este es menor a 0.5, imprimir “C <---> G” de lo contrario imprimir “G <---> C”.

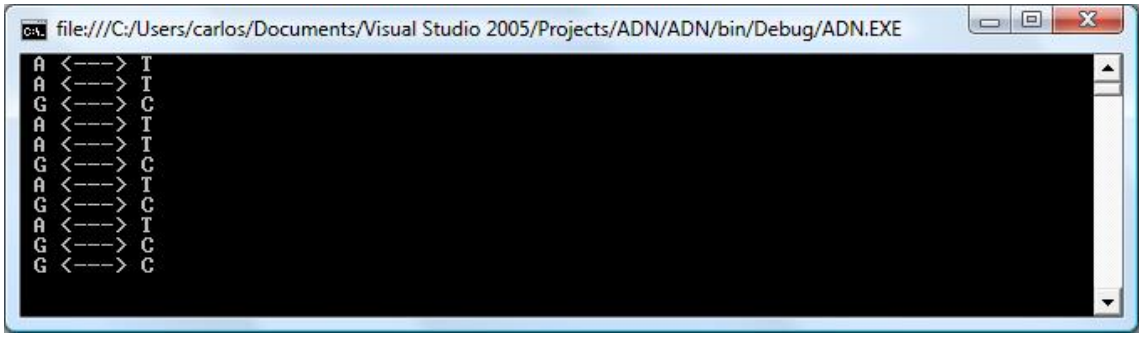


Figura 01. Resultado del algoritmo.

Lo que se pretende en este simple ejemplo es demostrar que una simple característica de la naturaleza puede ser programada. El algoritmo funciona con dos tomas de decisiones completamente aleatorias. El proceso elige uno de los dos tipos de asociación posibles y luego determina que nucleótido va primero y cual segundo. Finalmente, el resultado es impreso en pantalla.

La naturaleza, la evolución, incluso los procesos naturales inertes como el efecto de capilaridad del agua, son procesos que pueden ser descritos por medio de matemáticas y programación. Su estrategia está estrechamente relacionada con la búsqueda de la forma física en la arquitectura, con la optimización y con la resolución de problemas en todos los campos de investigación.

Los ecosistemas naturales tienen complejas estructuras biológicas, ellos reciclan sus materiales, permiten cambios y adaptación. Y usan eficientemente el uso de la energía del ambiente. De esta manera la evolución nos enseña que sus procesos son una fuerte máquina de resolución de problemas. Aunque no tenga conocimientos sobre lo que es diseño.

La Naturaleza de la Arquitectura

La arquitectura dibujada frecuentemente está inspirada en la forma de la naturaleza, y usualmente es usada como fuente de inspiración imitando sus modelos.

Se puede decir que la arquitectura es literalmente considerada parte de la naturaleza en el sentido que el hombre la hace con sus manos y de cierta manera bajo las condiciones del medio ambiente que la rodean. Lo interesante del asunto es que el hombre y la naturaleza comparten los mismos recursos para construir. Sin embargo, la materialización de la arquitectura consume grandes cantidades de recursos naturales. Incluso los edificios durante su uso a lo largo de los años son grandes derrochadores y gastadores de energía. Por un lado se desperdician diferentes tipos de energía y por otro, se invierte mucho en climatización durante el verano y el invierno.

Desde un punto de vista ecológico y económico sería ideal reducir drásticamente el consumo de energía y los materiales de construcción.

El doctor Andrew Marsh, creador de *Ecotect*[5], sostiene que la mayoría de los edificios que se diseñan y construyen hoy en día son equivalentes a un paciente enfermo en riesgo vital. "*Los edificios necesitan que les traigan todo. Constantemente necesitan estar siendo alimentados, bañados y limpiados. Siempre deben estar conectados a la red eléctrica. Además continuamente necesitan ciclos de remodelación, renovación y finalmente eliminación*."[6]

Marsh plantea la pregunta: ¿*es posible diseñar edificios capaces de hacer cosas por sí mismos, que se alimenten solos, que generen calor y energía, que sean capaz de proveer un hábitat natural e incrementen el encuentro público*?

A pesar de la creciente información que se tiene de la importancia sobre la ecología, el calentamiento global de la tierra, el consumo abusivo de agua, hoy en día no existe un pensamiento holístico ecológico por parte de los arquitectos humanos.

Diferentes gobiernos han implantado normas sobre sustentabilidad y entre ellos se han organizado para firmar tratados. Por ejemplo la agenda 21 celebrada en Rio Janeiro en 1992, o el protocolo de Kioto firmado el 16 de febrero del año 2005.

Un estudio desarrollado por *National Geographic Society* y la empresa *Globescan* concluyó que los países en vías de desarrollo se sienten más afectados por los efectos de la contaminación en una relación de 6 es a 10. En cambio en países de alto nivel económico la relación es de 3 es a 10. La encuesta abarcaba el ahorro en el hogar, transporte, alimentación y consumo. El

5 Ecotect es un software de análisis que permite medir y planificar el impacto ambiental de edificios modelados en 3D, realiza estudios de asoleamiento, sombras, luz, ganancia de calor, análisis acústico. Según el lugar época del año y clima. Más información en el capítulo 7.2.1 Ecotect.
6 Smart Geometry 2008, conferencia de Andrew Marsh, del 29 de febrero al 3 de marzo del 2008, Munich, Alemania

aspecto contradictorio del estudio fue que el 53% afirma estar al tanto de los peligros del calentamiento global, pero sólo un 9% se considera responsable[7].

El problema del ser humano está reflejado en esta encuesta. El hombre busca maximizar sus propios intereses, su máximo local. Piensa en cómo potenciar su capacidad y diferenciarse del resto de los hombres. Si en una población completamente altruista, aparece un solo individuo egoísta, que tenga la conciencia de que puede sacar provecho personal aprovechándose del resto, tendrá mayores posibilidades de sobrevivir y de tener hijos. Cada uno de estos hijos tenderá a heredar, por genética y conducta, sus rasgos egoístas. Luego de transcurridas varias generaciones, de esta selección natural, la población altruista será superada por un nuevo grupo de individuos egoístas[8].

La organización de los peces en los cardúmenes actúa como una entidad, donde todo el sistema se auto-organiza para defenderse de los predadores. Este comportamiento de agregación trae beneficios, incluyendo la defensa contra predadores (mejorando su detección y diluyendo la oportunidad de captura), se consigue más comida y aumenta el éxito de apareamiento. Otra medida beneficiosa de la agrupación de cardúmenes es el incremento en la eficiencia hidrodinámica.

El rasgo principal de un cardumen es la semejanza entre sus miembros. El efecto “de pares” posibilita que cualquier miembro del cardumen coincidente en semejanza no sea preferido como blanco de predadores. El efecto “de pares” consigue homogeneizar el cardumen. Experimentos desarrollados han demostrado que la agrupación por supervivencia es una habilidad aprendida y no innata. Sus conductas de dirección y sentido de movimiento son liderados por un pequeño grupo de experimentados *individuos*. Los ejemplos son muchos pero lo importante es entender que los compartimientos de la naturaleza responden a un ambiente y van en búsqueda del máximo global en el cual se encuentran.

En el reservorio genético, setenta y cinco millones de años de evolución independiente separan a seres humanos y ratones. Sin embargo ambos son genéticamente idénticos en un 99%. Las diferencias biológicas básicas se limitan a 300 genes que serían exclusivos de cada especie. Ratones y humanos comparten los mismos 4 nucleótidos. Lo interesante es como la estrategia de la evolución es capaz de producir modelos tan diferentes basados en los mismos principios.

El pensamiento que nace con la Revolución Industrial, el movimiento moderno y su propuesta de vida a los seres humanos ha tenido tanto éxito, que después de 160 años seguimos pensando igual. La geometría propuesta durante este período ha sido tan eficiente en cuanto a fabricación y precio, que ha sido muy difícil vencerla. Sin embargo ha propiciado consumos energéticos muy altos en edificios, métodos de fabricación y construcción altamente contaminantes. Por otro lado, el modelo económico planteado por la revolución industrial nos ha puesto en una carrera de adquisiciones para darle un sentido a nuestra existencia. Queremos cosas porque así somos más. El éxito se mide en dinero y propiedades. Lo peor de todo que está idea nos ha sumergido por lo menos en 2 crisis mundiales (1929 y 2008), y por si fuera poco el cambio climático, que está siendo rápido y violento, está afectando a todo el ecosistema del planeta.

---

7 Redacción ADN, “ADN”, en La Vida, Barcelona, 14 de mayo, 2008, pág. 13
[8] Dawkins, Richard. El Gen Egoísta, Salvat Editores S.A., 2ª edición, Barcelona, 2000. Pág. 10

Para el año 2100, se espera que la temperatura de la tierra suba entre 1º y 3.5ºC. Esto supondrá que las zonas desérticas serán más cálidas, pero no más húmedas, trayendo graves consecuencias para África y Oriente Medio. Casi la mitad de los glaciares se fundirán, esto significa que el mar subirá entre 0.4 y 0.65 m, haciendo desaparecer muchas zonas costeras. Las precipitaciones aumentarán entre un 3 y un 15% en ciertas zonas del planeta[9], lo que traerá perdidas en zonas de cultivo y la destrucción de completos ecosistemas.

La parte buena de todo esto, es que las grandes compañías han visto en el tema de la sustentabilidad una oportunidad de negocio. Es así como lo que está comenzando a suceder es que el que más paga más "ecológico" es.

Una cosa curiosa de este fenómeno es que siempre hemos admirado la naturaleza y siempre hemos tratado de aprender de ella. Usualmente consideramos feo todo lo que atenta contra ella y bonito todo lo que la beneficia. Hasta el momento cuando diseñábamos, imitábamos a la naturaleza desde el punto de vista formal. Por ejemplo observábamos las células vegetales e imitábamos su organización espacial a la hora de proyectar una casa.

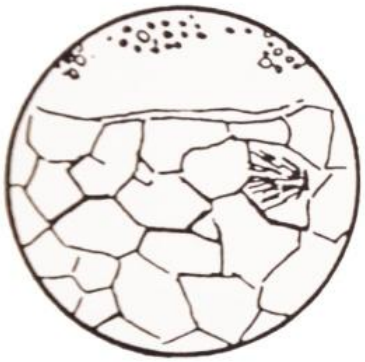
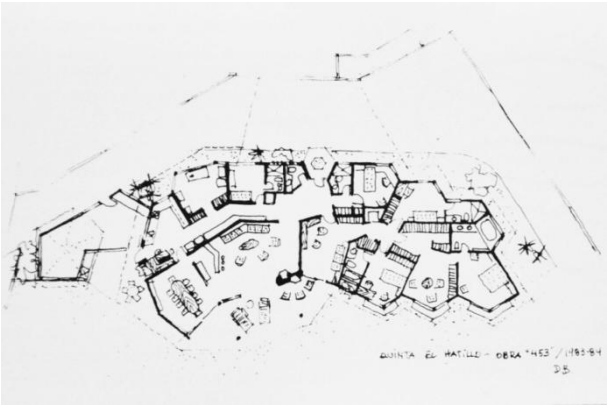


Figura 2. Dibujo de Dirk Bornohorst, Tejido de músculos y grasa del cuerpo humano, observado a través del microscopio. BORNHORST, Dirk., Arquitectura, Ciencia y Tao, Caracas, Ecología y ArTitectura, 1991, pág. 141



9 http://www.formaselect.com/areas-tematicas/medio-ambiente/el-proceso-de-Cambio-Climatico.htm [2010/06/20]

Figura 3. Planta de una casa diseñada por el arquitecto Dirk Bornhorst, "*ibíd*., pág. 89"

En el año 1952 Alan Turing público un artículo llamado "*The Chemical Basis of Morphogenesis*"[10], donde explicaba de manera matemática el principio de ruptura de simetría que existen en un embrión cuando comienza la división celular. Este trabajo sentaría las bases de todos los futuros modelos biológicos basados en ecuaciones diferenciales y sería el puntapié inicial de lo que luego se llamaría "ciencia de la complejidad"[11]. Sin embargo no sería hasta con la aparición del ordenador moderno, que estos nuevos modelos matemáticos fueron testeados y probados por diferentes investigadores. De esta manera, no es coincidencia que el desarrollo de la computación se haya ido formando con la idea de imitar procesos naturales.

Alan M. Turing tuvo un rol importante en el desarrollo del concepto de computación. "La máquina de Turing" es del año 1936. Consistía en un modelo computacional basado en un sistema binario (10101) capaz de computar de forma automática cualquier tipo de problema computable. Este trabajo fue el punto de partida para el desarrollo de los ordenadores. John Von Neumann fue un matemático Húngaro-Norteamericano que hizo contribuciones en diferentes campos de la ciencia, como por ejemplo la mecánica cuántica, la teoría de juegos (que el inventó), el desarrollo de la bomba atómica y el ordenador electrónico digital. Uno de sus trabajos en 1948, fue crear una máquina capaz de auto-replicarse a la que llamó *Kinematon*. Debido a las dificultades de construcción en el año 1966 optó por un modelo digital, el cual sería el primer autómata celular[12].

Turing, Von Neumann y muchos otros investigadores observaron la naturaleza y la explicaron a traves de las matemáticas. Apoyados en la computación han podido recrear algunas de sus estrategias: Colonias de hormigas, estructuras oseas, comportamiento de neuronas, geometrías naturales, estrategias de la evolución, genética, métodos de aprendizaje, etc. Desde la decada de los 70 estás estrategias han comenzado a ser aplicadas como modelos para resolver problemas, predecir comportamientos y reconocer patrones.

Cada nuevo enfoque que obtenemos de la observación de la naturaleza aporta información útil y valiosa para la humanidad. Recién hoy, estos últimos 75 años (1936, maquina de Turing + 74 años = 2010), tenemos las herramientas (ordenadores) y la teoría (matemáticas) suficiente para utilizar la estrategia de la naturaleza para resolver problemas y plantearnos un destino diferente.

Parte de nuestros problemas, pasan por la manera en como hemos visto la naturaleza. En este sentido nos hemos preocupado más de la forma final y no del proceso que la hizo posible. Nos

10 TURING, Allan, Collected works of A. M. Turing, Morphogenesis, Amsterdam, Elsevier Science Publisher B.V. 1992, pág. 5.

11 La ciencia de la complejidad estudia los sistemas biológicos y físicos que están compuestos por N cantidad de partes interconectadas que retroalimentan el sistema con información. El resultado de la interacción de estás partes generan nuevas propiedades y comportamientos que no pueden explicarse desde las partes aisladas, sino que desde la observación completa del conjunto. Por ejemplo, los cardúmenes, las colonias de hormigas, la interacción en la ciudad, etc.

12 Un autómata celular consiste en un espacio de N dimensiones dividido en celdas las cuales pueden encontrarse en dos o más estados de actuación (vivo, muerto, a la espera, etc.). El estado de cada celda depende de una serie de condiciones en relación a las celdas vecinas. Los autómatas celulares son actualmente usados en Bio-Ingeniería para predecir el comportamiento de células, Teoría de juegos, comportamientos urbanos, búsquedas por internet, etc.

hemos preocupado más del resultado que de la educación. Guerra, codicia, consumo, mentira son actitudes inherentes al ser humano y estan destruyendo nuestro único hábitat. Y los responsables somos nosotros. Observar la naturaleza desde está nueva perspectiva, es mucho más que aplicar su estrategia. Es una nueva oportunidad de replantearnos quienes somos.

El Relojero Ciego

En este capítulo, se explicará brevemente la historia de los Algoritmos Genéticos. Que son, como funcionan y que partes lo componen.

Entre los años 1950 y 1960 una serie de científicos comenzaron a estudiar, de manera independiente, la evolución de las especies con la idea de que podría ser usada como herramienta de optimización en problemas de ingeniería. La idea consistía en evolucionar una *población* de posibles candidatos a un problema dado, usando operadores inspirados en la variación genética y la selección natural.

En 1960 Ingo Rechenberg (1965-1973) introdujo el concepto de "estrategias evolutivas", como método usado para optimizar parámetros en alas de avión. Luego Shewefel profundizó sobre el mismo tema entre los años (1975 -1977). Fogel, Owens y Walsh (1966) desarrollaron "Programación evolutiva", una técnica donde las soluciones candidatas al problema se representaban como máquinas que mutaban aleatoriamente y se conservaba la mejor.

Una serie de investigadores estuvieron trabajando entre los años 1950 y 1960 en el desarrollo de algoritmos de optimización y de aprendizaje inspirados en la evolución. G. Box, G. Friedman, W. Bledsoe y H. Bremermann, todos ellos trabajaron en algoritmos inspirados en la evolución para optimización de funciones y aprendizaje automático.

Los Algoritmos Genéticos como tal fueron inventados por John Holland en los 60 y fueron desarrollados por él, sus estudiantes y colegas de la universidad de Michigan entre los años 1960 y 1970. Holland fue el primero en proponer el cruzamiento y otros operadores de recombinación. El objetivo general de Holland, no fue diseñar algoritmos para resolver problemas específicos, sino más bien un estudio formal sobre el fenómeno de adaptación que ocurre en la naturaleza y de esa manera desarrollar las vías en el cuál estos mecanismos de adaptación natural pueden ser importados a los sistemas computacionales. El libro de Holland salió el año 1975, bajo el nombre "*Adaptation in Natural and Artificial Systems*". En el libro se presentan los algoritmos genéticos de manera detallada, utilizando la mutación, la selección y el cruzamiento. Además Holland fue el primero en intentar poner la computación evolutiva sobre una firme base teórica[13]. Además planteo la Teoría de esquemas que ha sido la base de casi todos los trabajos sobre algoritmos genéticos.

Todos estos trabajos, establecieron las bases de algo más generalizado sobre la computación evolutiva. Durante los 80 los AG se aplicaron a una amplia variedad de áreas, desde problemas de juegos matemáticos hasta la ingeniería. En un principio, los problemas a los que fueron aplicados eran bastante teóricos, pero luego comenzaron a ser aplicados hacia sectores comerciales, como la predicción de la bolsa, la planificación de cartera de valores, ingeniería aeroespacial, diseño de microchips, bioquímica y biología molecular, diseño de horarios en aeropuertos y líneas de montaje[14].

13 MITCHELL, Melanie. An Introduction to Genetic Algorithms, Massachusetts: MIT Press, 1999. Pág: 3.
14 http://www.talkorigins.org/faqs/genalg/genalg.html [2010/06/21]

Andy Keane y Samuel Brown (1996) usaron un AG para diseñar un brazo mecánico para ser montado en órbita y para usarse en construcción aeroespacial. El problema a resolver consistía en hacer un brazo más fuerte y mejor para amortiguar las vibraciones que se producen cuando se sale al espacio exterior. Keane y Brown ejecutaron su AG durante 10 generaciones, debido a que computacionalmente era muy costosa la simulación. El resultado fue una estructura torcida de aspecto orgánico que se puede comparar con el fémur humano. La estructura utiliza la misma cantidad de material que el brazo estándar pero es más fuerte y soporta mucho mejor las vibraciones que el diseño tradicional.

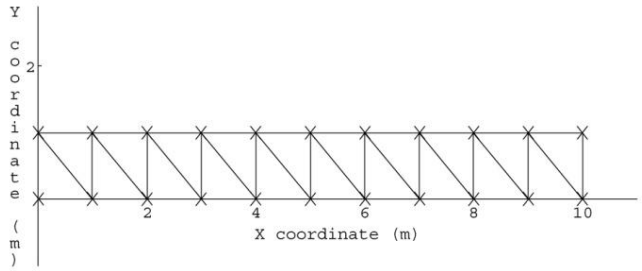


Figura 4. Inicialmente el brazo fue diseñado y analizado de manera tradicional. KEANE, A. J., BROWN, S. M. The design of a satellite boom with enhanced vibration performance using genetic algorithm techniques, in Proceedings of the Conference on Adaptive Computing in Engineering Design and Control 96, Plymouth (1996), pág. 107-113

Luego, la geometría base fue evaluada por cargas verticales y vibraciones. El objetivo de la optimización en esta etapa fue minimizar la frecuencia promedio de respuesta de la estructura a las vibraciones a las que fue sometida. El AG desarrollado por Keane y Brown permitió la exploración de nuevas geometrías variando la posición de 18 puntos dentro de la estructura. El resultado fue un 25% mejor que el diseño tradicional.

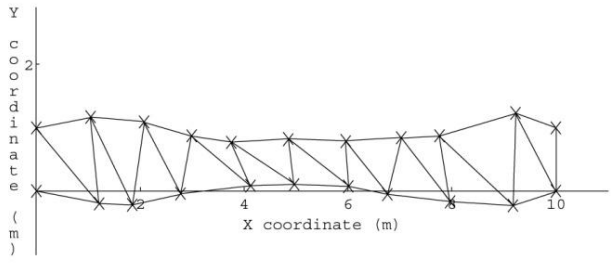


Figura 5. Diseño de la estructura luego del proceso de optimización. *Ibíd*., pág. 110

El siguiente paso fue explorar el mismo concepto pero en tres dimensiones. El diseño esta vez partió del brazo utilizado por la NASA. Las 90 piezas que componen el brazo fueron distribuidas de manera similar que en el primer ejercicio. Esencialmente, la estructura de dos dimensiones fue repetida tres veces, para formar una sección con forma de triángulo equilátero.

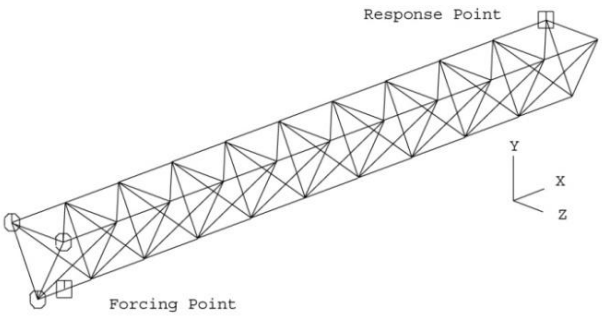


Figura 6. Geometría inicial, regular de 3 dimensiones, mostrando los puntos de fuerza y respuesta usados para evaluar los diseños. *Ibíd*., pág. 111

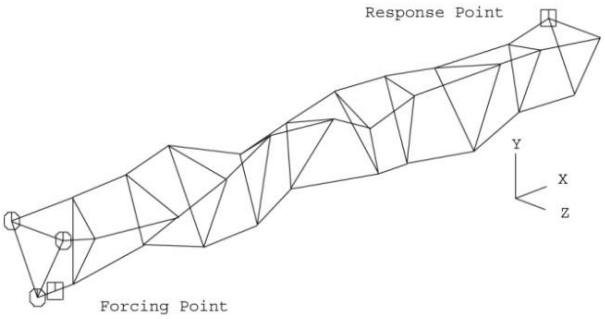


Figura 7. Resultado final del proceso de optimización, sobre la estructura de 3 dimensiones. *Ibíd*., pág. 112

Los investigadores Williams, Crossley y Lang (2001), usaron AGs para diseñar las órbitas de satélites y así minimizar los tiempos de pérdida de cobertura. Cuando se sitúan satélites en órbitas bajas (700 km de h.), por causas de la curvatura de la tierra, es inevitable que se pierda señal durante un tiempo con los receptores en la tierra. El problema a resolver consiste en diseñar orbitas que minimicen el tiempo de desconexión con la tierra. Este problema es multiobjetivo, porque implica minimizar el tiempo medio de desconexión para todos los receptores y el tiempo máximo de desconexión para cada una de las localizaciones. Por lo que los objetivos resultan ser mutuamente exclusivos.

Los resultados que presentaron los investigadores para 3, 4 y 5 satélites proponían orbitas asimétricas, con los satélites colocados alternando espacios grandes y pequeños, en lugar de homogeneizar la distribución como lo habrían hecho técnicas convencionales de diseño. Sin embargo el tiempo de desconexión se redujo en 90 minutos.

Un *Field Programmable Gate Array* (FPGA) es una placa de circuitos con una serie de celdas interconectadas entre sí que actúan como puertas lógicas. Estás funciones son controladas por software y dependiendo del programa pueden realizar una amplia variedad de funciones como reconocimiento de voz, captura de datos, etc.

El Dr. Adrian Thompson diseño un FPGA con AGs y produjo un modelo de circuito capaz de reconocer la voz y entender órdenes utilizando sólo 37 puertas lógicas. En el año 1997 esta

tarea era considerada imposible para sólo 37 entradas. Thompson generó cadenas aleatorias de bits de 1s y 0s y las usó como configuraciones de la FPGA. Luego selecciono los *individuos* más aptos de cada generación y los reprodujo mutando las descendencias. El objetivo de su trabajo consistía en evolucionar un FPGA que pudiese discriminar entre tonos de frecuencias distintas (1 y 10 Kilohercios) y distinguir palabras habladas como *go* y *stop*.

Luego de 3000 iteraciones, el sistema evolucionó en un diseño que utilizaba 100 celdas menos que otro diseño humano. Además no necesitaba reloj, un componente clave dentro de una FPGA. Cuando a Thompson se le preguntó cómo funcionaba, no supo responder[15]. De hecho de las 37 puertas lógicas, 5 de ellas no estaban conectadas al circuito central, pero si se les cortaba el suministro eléctrico el sistema dejaba de funcionar.

Los investigadores Altshuler y Linden (1997), diseñaron una antena utilizando AGs. El diseño de antenas tiende a funcionar bien con diseños simétricos y simples. Usualmente están basados en intuición, experiencia y ecuaciones aproximadas. Los investigadores en cambio especificaron las propiedades electromagnéticas de la antena y evolucionaron sus diseños hasta encontrar óptimos.

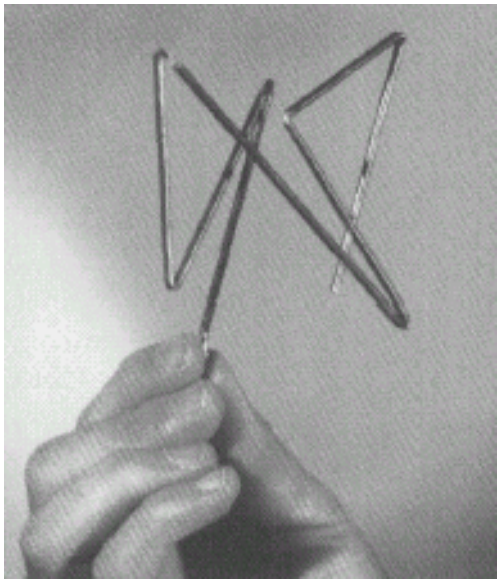


Figura 8. Antena de alambre doblado, optimizada mediante un AG. http://www.talkorigins.org/faqs/genalg/genalg.html#altshuler1997 [05/08/10].

La antena diseñada por Altshuler y Linden estaba dividida por 7 segmentos con una cobertura esférica. Cada *individuo* del AG consistía en un cromosoma binario que especificaba las coordenadas tridimensionales de cada extremo del alambre. La aptitud se evaluaba simulando a cada candidato de acuerdo a sus propiedades electromagnéticas y el mejor de cada iteración era construido y probado. Los resultados mostraron que las antenas tenían un patrón de

15 DAVIDSON, Clive. Creatures from primordial silicon, New Scientist, Vol. 156 no. 2.108, Nov. 1997, pág. 30-35.

radiación casi uniforme y un gran ancho de banda, tal como los resultados del AG lo habían establecido.

Aparte de los trabajos de Altshuler y Linden, también se encuentran los desarrollados por la NASA para el diseño de antenas en satélites.

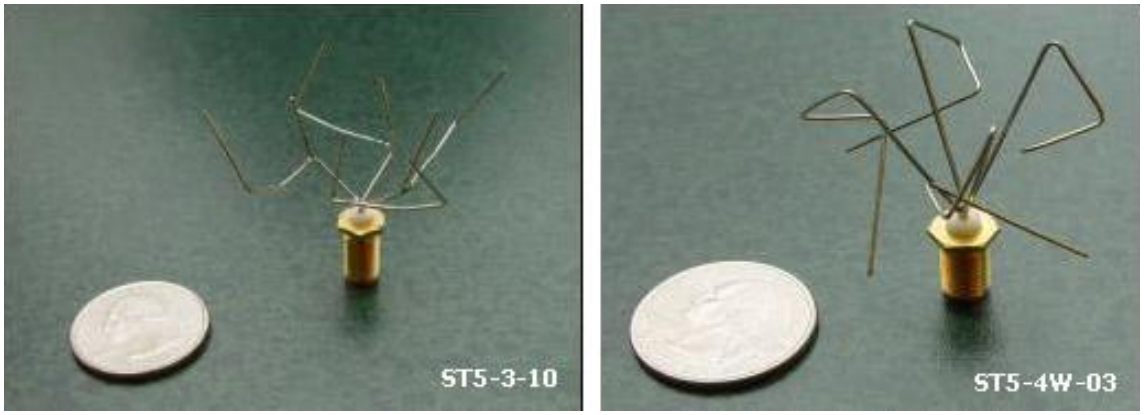


Figura 9. Dos tipos de antenas para frecuencias diferentes diseñadas para NASA por *Evolve Systems Group*. Se puede apreciar la escala de la antena, junto a la moneda. http://ti.arc.nasa.gov/projects/esg/research/antenna.htm [05/08/10].

Hoy en día los trabajos sobre AG abarcan casi todos los campos humanos. Por ejemplo se han utilizado AGs para predecir el comportamiento de acciones en la bolsa[16]. Se han utilizado AGs para jugar a las damas[17], geofísica[18] e ingeniería de materiales para diseñar polímeros conductores de electricidad[19]. También en matemáticas y algoritmia[20], en planeamientos de batalla y ejercicios militares[21]. En biología molecular, reconocimiento de patrones, robótica, diseño de rutas y horarios, ingeniería de sistemas, etc.

Un Algoritmo Genético opera generando una *población* aleatoria de soluciones a un problema. Cada solución será codificada imitando el cromosoma de las células. Está codificación será evaluada por una función que medirá su aptitud para resolver el problema. La mayoría de las mejores soluciones y algunas malas, serán elegidas para intercambiar trozos de su cromosoma, y así producir un nuevo *individuo* al que se llamará descendencia. Está descendencia heredará sus características de sus progenitores y luego su cromosoma tendrá una pequeña mutación.

16 MAHFOUD, Sam, GANESH, Mani. Financial forecasting using genetic algorithms. Applied Artificial Intelligence, vol.10, no.6, (1996), pag.543-565.
17 http://www.natural-selection.com/NSIPublicationsOnline.htm,[05/08/10]
18 SAMBRIDGE, Malcolm, GALLAGHER, Kerry. Earthquake hypocenter location using genetic algorithms. Bulletin of the Seismological Society of America, vol.83, no.5, 1993. p.1467-1491.
19 GIRO, R., M. CYRILLO, GALVÃO, D.S.. Designing conducting polymers using genetic algorithms. Chemical Physics Letters, vol.366, no.1-2, 2002. pág.170-175
20 HAUPT, Randy, HAUPT, Sue Ellen. Practical Genetic Algorithms. John Wiley & Sons, 1998.
21 KEWLEY, Robert, EMBRECHTS, Mark. Computational military tactical planning system. IEEE Transactions on Systems, Man and Cybernetics, Part C - Applications and Reviews, vol.32, no.2, 2002, pag.161-171.

Las descendencias mutadas y algunos padres, remplazarán la antigua *población* en espera que la media de la *población* vaya mejorando su aptitud.

4.1. Las Bases Biológicas

En la selección natural, los *individuos* de una *población* compiten constantemente con otros por comida y agua. Los *individuos* que más éxito tienen en conseguir estos recursos, tienen mayor oportunidad de aparearse y de tener descendencia. En cambio, los *individuos* más débiles tienen menor oportunidad de aparearse. De esta manera lo que sucede es que los genes de los *individuos* más fuertes se copian un número mayor de veces en las futuras generaciones que las de uno más débil. El hijo no es una copia fiel del padre y la madre, porque incorpora pequeñas mutaciones en su ADN. Esto hace posible que el hijo pueda o no salir mejor adaptado que sus padres. De esta manera la especie es capaz de irse adaptando al medio que lo rodea. Sin embargo la adaptación del *individuo* no es 100% genética, influyen otros factores como aprendizaje, el cual puede conseguirse a través de la imitación del comportamiento de los padres o por ensayo y error.

Todos los organismos vivos están constituidos por células y cada célula contiene el mismo grupo de uno o más cromosomas, cadenas de ADN, que son los planos para construir el organismo. Los cromosomas se dividen en genes, los cuales son una codificación de unas proteínas que explican los rasgos de un organismo, como por ejemplo el color de la piel. Las diferentes opciones que pueden tener estos rasgos se conocen como alelos. La mayoría de los *individuos* son diploides, lo que significa que tienen dos alelos por cada gen, uno heredado del padre y otro de la madre. Cada par de alelos se ubica en igual *locus* o lugar del cromosoma.

La mayoría de los organismos tienen muchos cromosomas en cada célula. Todos los cromosomas juntos se llaman genoma. El genotipo es el conjunto de genes de un organismo, pero no es la forma final del organismo. El fenotipo es la expresión del genotipo influenciado por el medio ambiente y es la forma final del organismo. Sin embargo, el fenotipo no es todo lo visible que vemos del organismo, como por ejemplo sería el caso de ciertos rasgos de personalidad, o la producción de ciertas enzimas. Otra cosa interesante es que no todos los genes se manifiestan en el fenotipo.

Los organismos que tienen pares de cromosomas se conocen como diploides y los organismos que sólo tienen un par de cromosomas se llaman haploides. En la naturaleza, la mayoría de las especies que se reproducen sexualmente son diploides, incluyendo a los seres humanos. Durante la reproducción sexual (*Crossover*), en cada padre, los genes son intercambiados entre cada par de cromosomas para formar un gameto (haploide) y luego los gametos de ambos padres se emparejan para crear un grupo de cromosomas diploides.

4.2. Codificación

El término cromosoma en AGs, se refiere a la codificación de una posible solución a un problema. El cromosoma está representado por una serie de parámetros o valores y su conjunto recibe el nombre de cromosoma, que contiene la información necesaria para la construcción del modelo, es decir la solución real al problema. La codificación suele hacerse por medio de valores binarios (10010), por lo que se asigna un número determinado de bits a cada parámetro. No todos los parámetros tienen que estar definidos por el mismo número de bits. Otro método consiste en codificar las soluciones como listas de enteros, reales o incluso letras, donde cada elemento en la lista representa un aspecto especifico de la solución. El objetivo de la codificación básicamente consiste en facilitar el trabajo de los operadores que cruzan y mutan a los candidatos seleccionados.

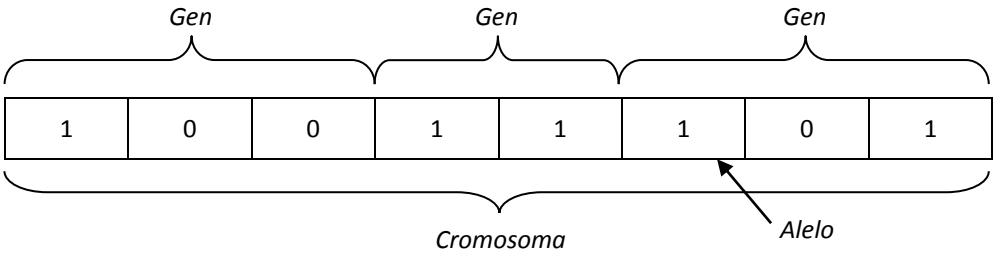


Figura 10. Esquema de la codificación de un *individuo*. Dibujo basado en "GESTAL, Marcos. *Introducción a los Algoritmos Genéticos*. Coruña, Depto. Tecnologías de la Información y las Comunicaciones, Universidade da Coruña. http://sabia.tic.udc.es/mgestal" [06/08/10].

4.3. Algoritmos Genéticos (AGs)

Un Algoritmo Genético es un método de programación adaptativo que puede usarse para resolver problemas de búsqueda y de optimización. Su estrategia está basada en los procesos evolutivos de los organismos vivos.

A lo largo de las generaciones, las poblaciones de seres vivos evolucionan de acorde con los principios de la selección natural y la supervivencia de los más fuertes o los más aptos. La estrategia de la naturaleza opera produciendo una gran cantidad de modelos, que son evaluados por el medio ambiente. Si estos modelos no son capaces de adaptarse, desaparecen, prevaleciendo sólo los más aptos.

Los AGs operan produciendo aleatoriamente una cierta cantidad de modelos, o posibles soluciones a un problema. Cada una de estas soluciones está codificada de alguna manera, y son evaluados cuantitativamente bajo una función de aptitud. Los *individuos* que no cumplan

con la aptitud mínima son eliminados. Sin embargo, algunos *individuos* serán prometedores, y estos serán conservados y copiados para ser recombinados y así producir nuevas descendencias que incorporaran pequeñas mutaciones en sus cromosomas. De esta manera se produce una nueva *población* de posibles soluciones, que remplazará a la antigua. Así a lo largo de las generaciones las buenas propiedades y características de los mejores *individuos* se irán propagando por toda la *población*, haciendo que el promedio de aptitud vaya mejorando tras cada iteración del algoritmo. Entonces se favorecerá el cruce de los *individuos* mejor adaptados y serán exploradas las mejores áreas del espacio de búsqueda del problema.

Los AGs son una técnica robusta y pueden tratar con éxito una gran variedad de problemas provenientes de diferentes áreas, incluyendo aquellos en la que otros métodos encuentran dificultades. Si bien no garantiza que se encuentre la solución óptima del problema, existe una evidencia empírica de que se encuentran soluciones de un nivel aceptable en un tiempo competitivo con respecto a otras técnicas de optimización.

Los AGs pueden ser vistos como un vehículo 4x4, en el sentido en que se adaptan a la mayoría de los terrenos. Sin embargo cuando existen técnicas especializadas para resolver un determinado problema, como por ejemplo correr sobre una carretera de asfalto, lo más probable es que superen al AG en rapidez y en eficacia (un automóvil deportivo).

El campo de aplicaciones de los AGs se relaciona usualmente con aquellos problemas, para los cuales no existen técnicas especializadas. Aunque actualmente se están reportando muy buenos resultados con técnicas hibridas de algoritmos de optimización específica y AGs.

Los AGs se basan en operadores genéticos para realizar su tarea (selección, cruce y mutación). Cada prototipo al ser testeado es representado por una tira de parámetros, la cual recibe el nombre de cromosoma. Gracias al cromosoma se facilita la tarea de los operadores. Un ejemplo de un cromosoma genérico puede ser el siguiente.

1	0	1	1	0	0	1	1

Figura 11. Ejemplo de un cromosoma binario compuesto de 1s y 0s.

En este cromosoma, a cada casilla le corresponde un parámetro a evaluar. Si el parámetro a evaluar es satisfecho, la casilla es marcada con un 1 o de lo contrario con un 0. La suma total de los genes determinan el factor de aptitud del *individuo*, en este caso, el modelo cumple sólo con 5 de los 8 requerimientos.

El siguiente código genera un cromosoma binario de 1 y 0s. En el cual los 1 y 0 son incluidos de manera aleatoria. Luego se determina el factor de aptitud o *fitness* del cromosoma.

```
namespace Cromosoma
{
    class Chromosome
    {
        static void Main()
        {
            int[] gen = new int[8];
            int i, j;
            int nCount = 0;
```

```
            Random R = new Random();
            int Bound = gen.GetUpperBound(0);

            for (i = 0; i <= Bound; i++)
            {
                double Rnd = R.NextDouble();
                if (Rnd < 0.5)
                {
                    gen[i] = 0;
                }
                else
                {
                    gen[i] = 1;
                }
            }

            for (j = 0; j <= Bound; j++)
            {
                Console.WriteLine("in the chromosome gen, the position {0} has
a fitness of {1}", j, gen[j]);

                if (gen[j] == 1)
                {
                    nCount = nCount + 1;
                }

            }

          Console.WriteLine("\n");
          Console.WriteLine("the total fitness of the gen is: {0}", nCount);
          Console.ReadLine();
        }
    }
}
```

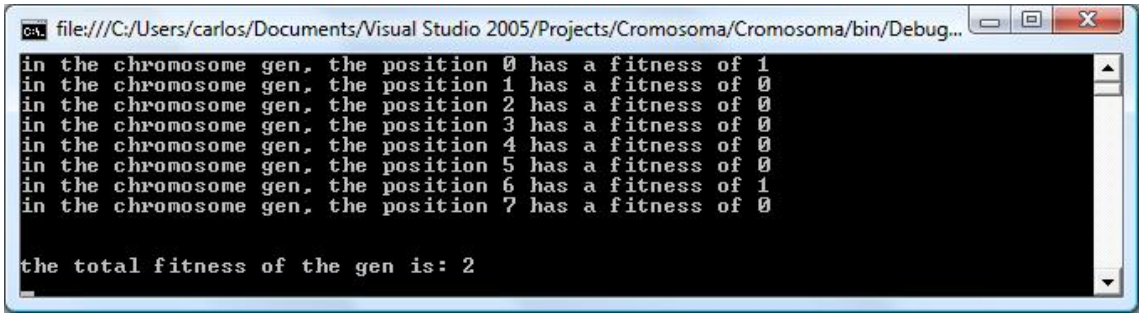
Pseudocódigo clase *Chromosome*

1. Crear una lista de 8 elementos, para guardar en ella cada uno de los genes (1s y 0s).
2. Repetir la siguiente acción por cada elemento de la lista.

   Calcular un número aleatorio entre 0 y 1 con 6 decimales.

   Si el valor es menor a 0.5 guardar en la lista un 0 de lo contrario guardar un 1.

3. Una vez que la lista ha sido llenada con 1s y 0s, contar cuantos 1s hay.
4. Imprimir los resultados en pantalla.



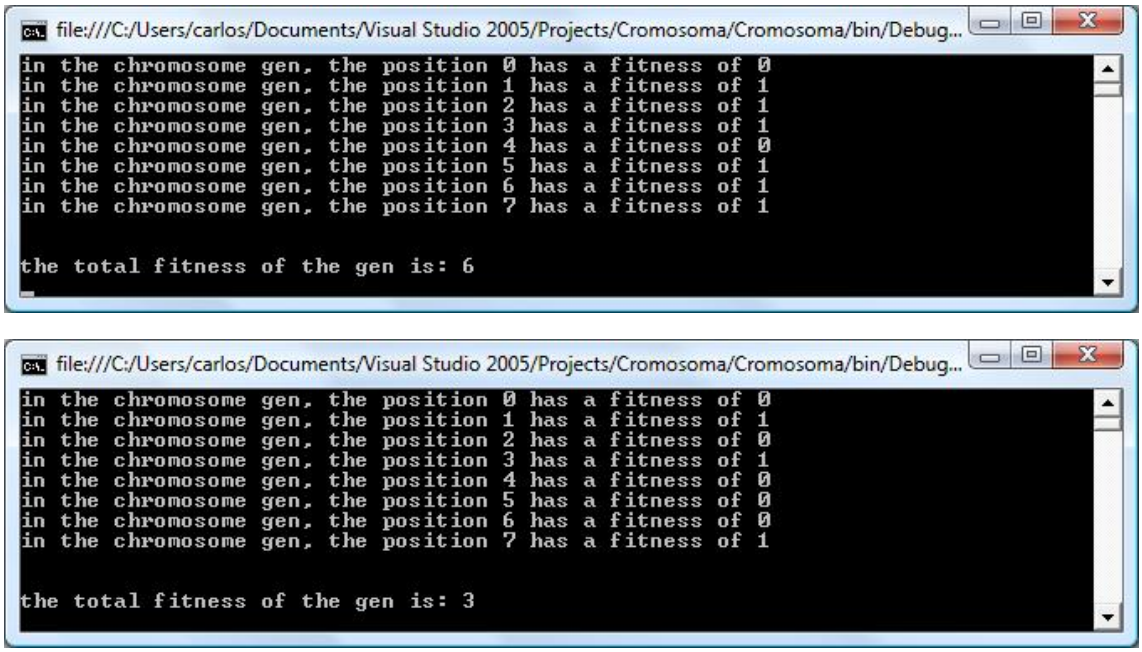


Figura 12. Salida del algoritmo, donde se muestra la posición de 1s y 0s dentro del cromosoma. La aptitud de cada cromosoma es evaluada contando los 1s que cada una posee.

La función ha creado un cromosoma de 8 genes (10101) elegidos al azar. Dentro de la función se suman los valores y se imprime la función de aptitud o *fitness*.

4.4. Métodos de Selección

Existen diferentes maneras de seleccionar los mejores *individuos* dentro de una *población* de soluciones a un problema. Algunos métodos son exclusivos y otros en cambio, pueden utilizarse en combinación con otras estrategias de selección. Un algoritmo de selección es el que decide que *individuos* tendrán la oportunidad de reproducirse y cuáles no. La posibilidad de ser seleccionado está estrechamente relacionada con la aptitud o *fitness* de cada *individuo*. Tampoco se deben seleccionar siempre los mejores y descartar los peores, porque entonces la *población* se vuelve homogénea y no busca en todo el espacio de soluciones. Existen diferentes tipos de selección, pero todo ellos se basan en la evaluación de los *individuos* por medio de su factor de aptitud en el cromosoma.

La selección de *individuos* debe permitir que los *individuos* mejor dotados produzcan mayores descendientes que los menos dotados, Esto significa que los “genes” de los *individuos* más adaptados se propagarán en sucesivas generaciones hacia un número de *individuos* creciente.

Existen bastantes algoritmos de selección y la mayoría de ellos son personalizados para enfrentar los diferentes problemas. Unos buscan mejorar el consumo computacional, otros los rangos de selección de los mejores y peores *individuos*, etc. La elección del método de selección determinará la estrategia de búsqueda del AG. Si se elige un método con una alta

presión de selección se centra la búsqueda de las soluciones en un entorno próximo a las mejores soluciones de la *población*. En cambio, si se elige un método de baja presión de selección se deja el camino más abierto a explorar otras zonas del espacio de búsqueda. Los principales métodos de selección son:

4.4.1. Selección Elitista.

Como lo indica su nombre, garantiza que sólo serán seleccionados uno o un grupo de los mejores *individuos* para la siguiente generación. Sin embargo, la mayoría de los AGs no utilizan está técnica, sino que usan una manera modificada de este algoritmo. Si sólo se seleccionan los *individuos* más aptos se cae en un problema de convergencia prematura. Esto implica que el proceso de optimización se centra en una pequeña parte del espacio de búsqueda, y por lo tanto se descartan soluciones que pueden ser mejores. Por eso, es importante asegurar que se seleccionen *individuos* de bajo *fitness* o aptitud.

4.4.2. Selección de Ruleta

La selección de ruleta (*Roulette Wheel*) Es probablemente la técnica más usada de selección aplicada a los AGs. Este tipo de selección es proporcional a la aptitud de cada *individuo*, donde a cada uno se le otorga un grado de posibilidad para ser seleccionado en la siguiente ronda. Los *individuos* más aptos tienen más posibilidades de ser seleccionados que los menos aptos.

Cada uno de los *individuos* de la *población* se le asigna una parte de la ruleta, proporcional a su *fitness*. De tal forma que la suma de todos los porcentajes sea el valor total de la ruleta. Los mejores *individuos* recibirán una porción de la ruleta mayor que la de los peores *individuos*. Usualmente los mejores *individuos*, se encuentran al principio de la ruleta y para seleccionar se genera un número aleatorio entre 0 y el tamaño de la ruleta. Es un método simple pero se vuelve ineficiente cuando se trabajan con grandes poblaciones.

Usualmente el tamaño de la ruleta, la cantidad de *individuos* que contendrá, no es un valor fijo, pero si es necesario definir el la cantidad de veces en que el mejor *individuo* será seleccionado. Esto se hace para reducir costos computacionales y controlar el porcentaje de selección de los peores *individuos*. La selección de este tipo suele representarse por una curva no polinómica de la forma y = 1/x.

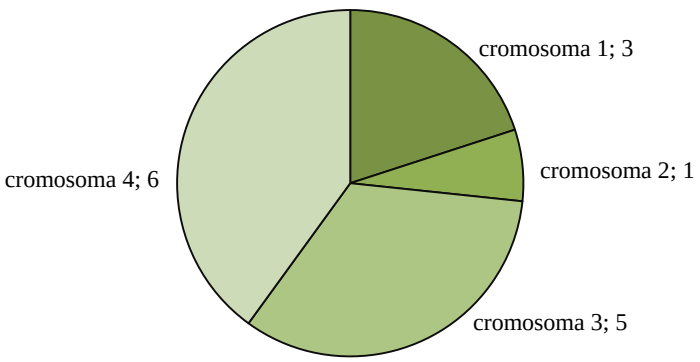


Figura 13. Selección de Ruleta. En la imagen se ve que el cromosoma 1 tiene un *fitness* de 3, el cual le da un área dentro de la ruleta proporcional a su aptitud. El cromosoma 2 tiene el menor *fitness* de todo el grupo, por lo que sus probabilidades de ser seleccionado son bastante bajas. El cromosoma 4 es el mejor posicionado de todos, porque al tener el mayor *fitness* es el que más probabilidades tiene de ser seleccionado. Aunque no la certeza, como ocurría en el método anterior.

4.4.3. Selección de Escalada

Tras varias generaciones, la aptitud media de la *población* va incrementándose y con esto la presión selectiva. Porque cada *individuo* cava vez se diferencia menos de sus competidores. Entonces la función de aptitud se vuelve cada vez más discriminatoria. Este método es útil en el momento en que todos los *individuos* tienen una aptitud alta y las diferencias entre ellos es muy baja. También es útil cuando se trabajan con *población* de pocos *individuos*.

Este método se va poniendo cada vez más estricto en cuanto a selección durante la ejecución del algoritmo. Esto sucede porque con cada mejora de la aptitud de la *población* las diferencias entre los *individuos* buenos y malos son más estrechas.

4.4.4. Selección por Torneo

La idea de este método consiste en escoger al azar un número de *individuos* de la *población* y seleccionar al mejor de ellos compitiendo con su *fitness* como si fuese un torneo. El proceso se repite hasta que el número de *individuos* coincida con el tamaño de la *población*. Usualmente el tamaño del torneo es de dos *individuos*, pero se pueden incluir más. La cantidad de *individuos* dentro del torneo determina la presión de selectividad. Mientras más *individuos* estén compitiendo mayor será la presión y los peores *individuos* apenas tendrán oportunidades de ser elegidos. Este tipo de selección usualmente se combina con la selección por escalada para ir aumentando la presión de selectividad a medida que la aptitud mejora en la *población*. Existen dos versiones de selección por torneo.

Selección por Torneo Determinística

Habitualmente, se selecciona el mejor de los *individuos*. Está técnica es llamada determinista de tipo, ya sea el mejor o el peor. Tiene la ventaja de que permite un cierto grado de elitismo, el mejor nunca va a morir y tienen más probabilidad de reproducirse que el menos adaptado. Además no produce una convergencia prematura.

Selección por Torneo Probabilística

La versión probabilística únicamente se diferencia en que en vez de elegir siempre el mejor, se genera un número aleatorio entre 0 y 1. Si es mayor que un parámetro T se escoge al mejor de los *individuos*, de lo contrario se escoge al otro.

4.4.5. Selección por Rango

A cada *individuo* se le asigna un rango numérico basado en su aptitud y la selección se basa en este ranking. La ventaja de este tipo de selección es que se evita que los *individuos* más aptos ganen en dominio por sobre los otros, de cierta manera lo que hace es preservar la diversidad de *individuos* y evitar que se obstaculice la búsqueda de una solución aceptable. Este tipo de selección es usado generalmente en problemas con espacios de búsqueda de mucho ruido.

4.4.6. Selección Generacional

La descendencia de los *individuos* seleccionados en cada generación remplaza completamente a la generación anterior. No se conservan *individuos* entre generaciones. Este tipo de selección creará una *población* completamente nueva que remplazará a la generación anterior. Este tipo de selección se aplica generalmente sobre poblaciones pequeñas. Entre 20 y 30 *individuos* por *población*.

4.4.7. Selección por Estado Estacionario

Se conservan *individuos* entre generaciones. Los *individuos* durante el proceso de optimización pasan por múltiples rondas de evaluación. Está técnica permite que entre generaciones se vayan conservando ciertos *individuos* y remplazando algunos otros. Este tipo de selección es más cercano a lo que sucede en la selección natural biológica. Este tipo de selección usualmente es usado en grandes poblaciones. Más de 100 *individuos* por *población*.

4.4.8. Selección Jerárquica

Los *individuos* atraviesan múltiples rondas de selección en cada generación, haciéndose cada vez más discriminatorias y rigurosas. La ventaja de este método es que reduce el tiempo total de cálculo.

4.5. Cruzamiento

Una vez que se han generado los *individuos* y han sido seleccionados el siguiente operador genético en entrar en acción es el cruce (*Crossover*). El cruzamiento, requiere dos *individuos* que intercambien segmentos de su cromosoma, produciendo uno o dos *individuos* llamados descendencia artificial. Este proceso simula la combinación de los cromosomas del padre y la madre en la reproducción sexual.

El cruce (*Crossover*) es un método basado en la estrategia de reproducción sexual. Opera en la transición de generaciones y es aplicada al 90% de los *individuos* seleccionados. Existen diferentes métodos de cruce, pero la mayoría opera de dos maneras diferentes. La primera es llamada estrategia destructiva, donde los descendientes se insertarán en la nueva *población*, aunque sus padres tengan un mejor *fitness*. La otra estrategia es llamada no destructiva. Consiste en que si la descendencia no supera el *fitness* de los padres no se insertarán en la siguiente *población*. La idea del cruce consiste en que si se toman dos *individuos* con buena aptitud se obtiene una descendencia que comparte los genes de ambos padres, y por lo tanto, existe una alta posibilidad de que los genes heredados aumenten la aptitud del hijo. Al compartir las características buenas de ambos padres, la descendencia, o al menos alguna parte de ella, debiese ser mejor que la de los padres por separado.

Existen muchos tipos de cruce, pero los más utilizados son tres: Cruce de 1 punto, Cruce de 2 puntos y Cruce uniforme.

4.5.1. Cruce de 1 punto

El cruce de 1 punto comienza con la selección de dos *individuos*, donde se cortan sus cromosomas por un punto seleccionado aleatoriamente o no, para generar dos segmentos en cada cromosoma. Luego se intercambian los segmentos de los padres para generar dos nuevos descendientes.

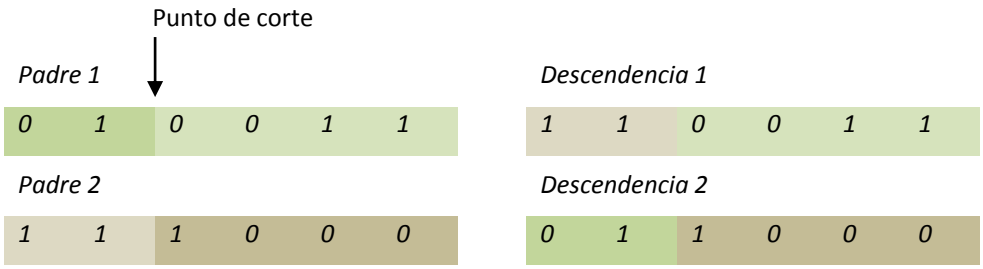


Figura 14. Esquema del Cruzamiento de 1 punto

El siguiente código, esboza de manera genérica lo que podría ser un operador de cruce. Funciona cortando el material genético de dos cromosomas diferentes y formando uno tercer *individuo* que es conocido como hijo. En este operador, cada cromosoma tiene 8 genes y cada tira es cortada en dos. El primero *individuo* aporta la primera mitad del cromosoma, y el segundo la otra mitad.

Se puede decir que la característica más importante de un AG es el uso del cruce o *crossover*. Un punto de cruce es la forma más simple de este operador. Consiste en elegir un punto aleatoriamente donde los cromosomas de los padres serán intercambiados para crear dos descendientes. La idea es combinar los padres para producir diferentes cromosomas.

```
namespace CrossOver
{
    class Cruzamiento
    {
        static void Main(string[] args)
        {
            int i, j, nCount = 0;
            int[] Son = new int[8];

            int[] C1 = new int[8];
            int[] C2 = new int[8];

            C1 = Chromosome();
            C2 = Chromosome();

            for (i = 0; i <= 3; i++)
            {
                Son[i] = C1[i];
            }

            for (i = 4; i <= 7; i++)
            {
                Son[i] = C2[i];
            }

            for (j = 0; j <= Son.GetUpperBound(0); j++)
            {
                Console.WriteLine("the son has in position {0} the value {1}",
j, Son[j]);

                if (Son[j] == 1)
                {
                    nCount = nCount + 1;
                }
            }

            Console.WriteLine("the total fitness of the Son is: {0}", nCount);
            Console.ReadLine();
        }
}
```

Pseudocódigo clase Cruzamiento

1. Crear tres listas de la misma longitud. Dos listas servirán para definir a los padres y la tercera para el hijo.
2. Llamar a la clase *Chromosome* dos veces para crear dos cromosomas que servirán de padres.
3. Elegir un punto de corte. En este caso 3
4. Copiar desde el primer gen del padre al punto de corte al hijo
5. Copiar desde el punto de corte + 1 de la madre hasta el último gen al hijo.
6. Imprimir los resultados en pantalla: cromosoma del padre y la madre, cromosoma del hijo.

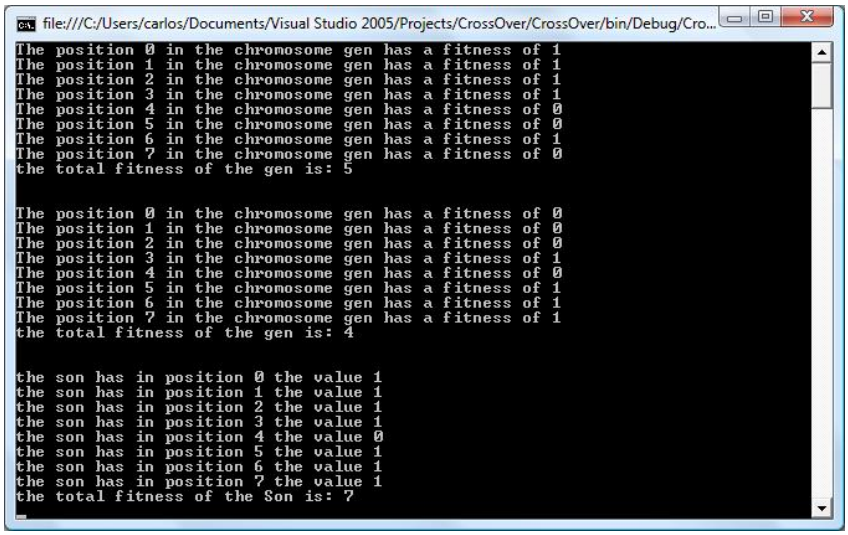


Figura 15. Resultado del operador de cruce genérico. En esta generación el primer *individuo* tuvo un *fitness* de 5. La mayoría de sus genes con valor 1 se encuentran en la primera mitad de su cromosoma. El segundo tuvo un *fitness* menor, sin embargo la mayoría de sus genes valor 1 se encuentran en la segunda mitad de su cromosoma. El punto de cruce es en el gen 4. La descendencia hereda del primer *individuo* del gen 0 al 4 y del segundo del 5 al 8. Finalmente la descendencia tiene un *fitness* de 7, heredando las mejores características del padre y la madre.

4.5.2. Cruce de 2 puntos

El cruce de 2 puntos es muy similar al de 1 punto, salvo que en este caso el cromosoma se parte en tres partes. Hay que considerar que ninguno de los puntos de corte coincida con el extremo final del cromosoma. Para generar la descendencia se escoge el segmento central de uno de los padres y los segmentos laterales del otro padre.

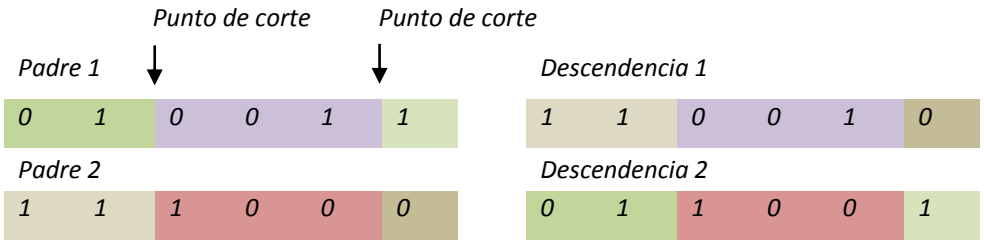


Figura 16. Esquema de cruce de 2 puntos.

4.5.3. Cruce Uniforme

Cada gen de la descendencia en el cruce uniforme tiene las mismas posibilidades de pertenecer a un padre u otro. Está técnica implica la generación de una lista binaria aleatoria de 1s y 0s. Si en la primera posición de la lista existe un 1, el gen situado en esa posición se copia del padre al hijo. Por el contrario, si existe un 0 se copiará el de la madre. Para producir el segundo descendiente se intercambian los papeles de los padres o bien se intercambiá la regla de descendencia en la lista. El resultado es que la descendencia contiene una mezcla de genes del padre y la madre.

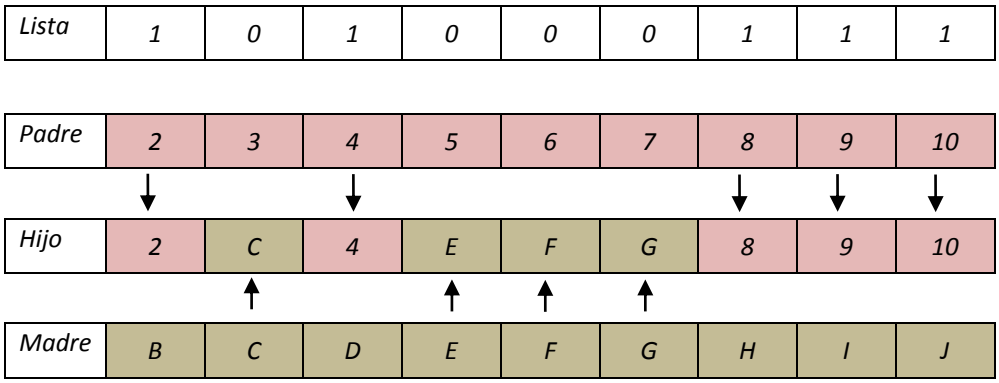


Figura 17. Esquema del cruce uniforme.

Otros tipos de cruce

Usualmente los tipos de cruce hasta ahora vistos, son aplicados a casi cualquier tipo de cromosoma. En el caso de que se usen cromosomas con valores enteros o reales pueden definirse otros tipos de cruce, como por ejemplo, el cruce tipo media. Este consiste en calcular el valor medio de los padres y heredarlos al hijo. Este tipo de cruce solo genera un descendiente.

El cruce o *crossover* es clave en el desarrollo e implementación de un AG. Gracias al cruce es posible buscar rápidamente en el espacio de soluciones. El operador de cruce es el responsable de las propiedades del algoritmo genético, y determinará en gran medida la evolución de la *población*.

4.6. Mutación

Una vez que los *individuos* seleccionados han sido cruzados y hay una descendencia que ha heredado las características de sus padres, el siguiente operador genético en entrar en acción es la mutación. Este operador consiste en alterar aleatoriamente los cromosomas de los *individuos*. Esto es hecho por dos motivos. El primero, es para que exista una probabilidad de mejoría de la aptitud en la siguiente generación y la segunda, es para abarcar el mayor espacio de búsqueda posible.

La mutación provoca que alguno de los genes varíe de forma aleatoria y usualmente se aplica a la descendencia de los *individuos* cruzados. De esta manera, se imita el comportamiento de la naturaleza porque cuando se genera la descendencia siempre se produce algún tipo de error, que por lo general no tiene mayor trascendencia en el paso de los datos genéticos del padre al hijo. La probabilidad de mutación siempre es muy baja, alrededor de un 1%. Esto se debe sobre todo a que los *individuos* suelen tener un *fitness* menor después de ser mutados. La mutación más usual es el remplazo aleatorio, que consiste en variar aleatoriamente un gen del cromosoma. Si se trabaja con cromosomas binarios (101001010), bastará con sólo invertir un 1 por un 0 o viceversa. También es posible realizar la mutación intercambiando valores dentro del cromosoma o multiplicar un gen por un valor aleatorio entre un mínimo y un máximo, etc.

La mutación de un *individuo* pasa por variar en alrededor de un 1% su cromosoma. La técnica y la manera que se emplee dependerán estrechamente, de las condiciones del problema y del tipo de dato que se tenga en el cromosoma.

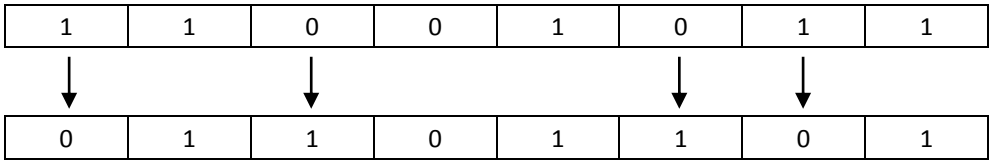


Figura 18. Esquema de Mutación.

El siguiente código es un ejemplo de una mutación genérica. El operador que genera la mutación, elige al azar 4 genes del cromosoma. Estos modifican los 1s por 0s y viceversa. En el caso que el operador elija dos veces el mismo gen, este no cambiará, porque repetirá el proceso 2 veces. Por ejemplo el 1 por el 0 y luego el 0 por el 1.

```
namespace Mutation
{
    class Mutation
    {
        static void Main()
        {
            int i, j, n = 0;
            int[] Chrom = new int[8];
            int[] arrDat = new int[4];

            Chrom = Chromosome();
            Random R = new Random();

            for (i = 0; i <= (Chrom.GetUpperBound(0) * 0.5); i++)
            {
                int Rnd = R.Next(0, Chrom.GetUpperBound(0));
                arrDat[i] = Rnd;

                if (Chrom[Rnd] == 0)
                {
                    Chrom[Rnd] = 1;
                }

                else if (Chrom[Rnd] == 1)
                {
                    Chrom[Rnd] = 0;
                }
            }
            for (j = 0; j <= Chrom.GetUpperBound(0); j++)
            {
                Console.WriteLine("The mutated chromosome, position {0} has a
new fitness of {1}", j, Chrom[j]);
                if (Chrom[j] == 1)
                {
                    n = n + 1;
                }
            }
            Console.WriteLine("the total fitness of the new Chromosome is:
{0}", n);
            for(int p = 0; p <= arrDat.GetUpperBound(0); p++)
            Console.Write(arrDat[p]);
            Console.ReadLine();}
```

Pseudocódigo clase *Mutation*

1. Llamar a la clase *Chromosome* para obtener un cromosoma de longitud 8.
2. Crear una lista de 4 elementos para guardar la ubicación de los genes mutados.
3. Repetir la siguiente operación 4 veces.

   Generar un número aleatorio entre 0 y la longitud del cromosoma. Usar este número para leer la información que existe en ese gen dentro del cromosoma.

   Si existe un 0 cambiarlo por un 1. Si existe un 1 cambiarlo por un cero.

4. Imprimir los resultados en pantalla.

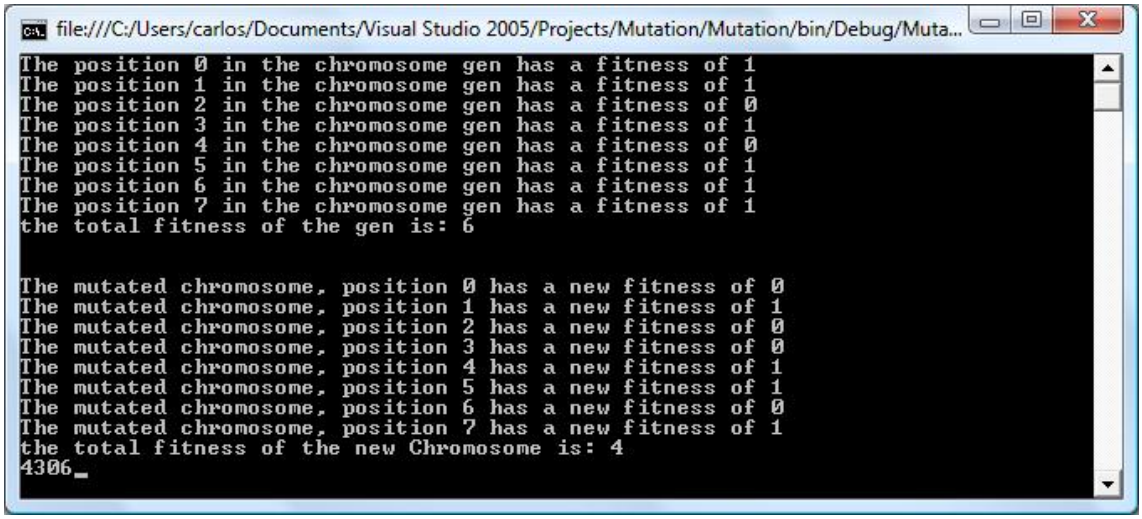


Figura 19. Resultados del operador de mutación genérico.

El operador de mutación recibe un cromosoma de 8 genes y elige al azar 4 genes del cromosoma para modificar sus genes. La condicional invierte en este caso el 1 por el 0 y el 0 por el 1. En la imagen de resultado, se puede ver que los genes 4, 3, 0 y 6 han sido modificados.

La literatura admite que el operador de cruce es el responsable de una búsqueda rápida a lo largo del espacio de soluciones, y que el operador de mutación es el encargado de explorar todos los rincones del espacio de búsqueda. Además a la mutación se le atribuye la particularidad de que gana importancia a medida que la *población* de *individuos* va convergiendo.[22]

Schaffer en 1989, encuentra en uno de sus experimentos que el efecto del cruce en la búsqueda es inferior al que previamente se esperaba. Implementa una técnica llamada evolución primitiva. La cual consta de selección y mutación. Schaffer reporta que esta implementación supera con creces a una evolución basada exclusivamente en cruce y

22 DAVIS, L. Handbook of Genetic Algorithms, Nueva York: Van Nostrand Reinold, 1991

selección. Otra conclusión que obtiene, es que la definición del valor de probabilidad de mutación es mucho más crucial que el relativo a la probabilidad de cruce.[23]

4.7. Funcionamiento de un Algoritmo Genético

Un Algoritmo Genético puede dividirse en una serie de pasos:

Paso 1.

Consiste en generar una *población* aleatoria de posibles soluciones a un problema. Está *población* de soluciones estará codificada de cierta manera para permitir el cruce y la mutación (operadores genéticos) entre ellos.

Paso 2.

En el siguiente paso, las posibles soluciones son ordenadas de acuerdo a su aptitud para resolver el problema que se busca solucionar. Está aptitud necesariamente debe ser medida y es conocida como el *fitness*. De esta manera, las soluciones con una mejor aptitud tendrán un mejor *fitness* que uno que no responda tan bien. La evaluación de aptitud depende estrechamente del problema que se está evaluando y es determinante en la eficiencia y eficacia que tenga el AG. Luego los *individuos* más aptos y algunos no tanto son seleccionados

Paso 3.

En esta etapa del algoritmo, los *individuos* seleccionados anteriormente son cruzados entre sí y su descendencia es levemente mutada. Los nuevos *individuos* remplazan a la *población* anterior, y se espera que está nueva *población* tenga una mejor aptitud. La idea de remplazar la antigua *población*, consiste en que el promedio de aptitud se vaya incrementando con cada iteración del algoritmo.

Paso 4.

La nueva *población* es evaluada. Si el mejor *Individuo* de la nueva *población* no satisface el problema o el número de iteraciones del algoritmo no se ha alcanzado, entonces se regresa al paso 2 y todo el proceso se vuelve a repetir.

23 SHAFFER, J.D., CARUNA, R.A., ESHELMAN, L.J., DAS, R., A study of control parameters affecting online performance of genetic algorithms for function optimization. Proceedings of the third International Conference on Genetic Algorithms, Morgan Kaufmann, 1989, pag.51-60.

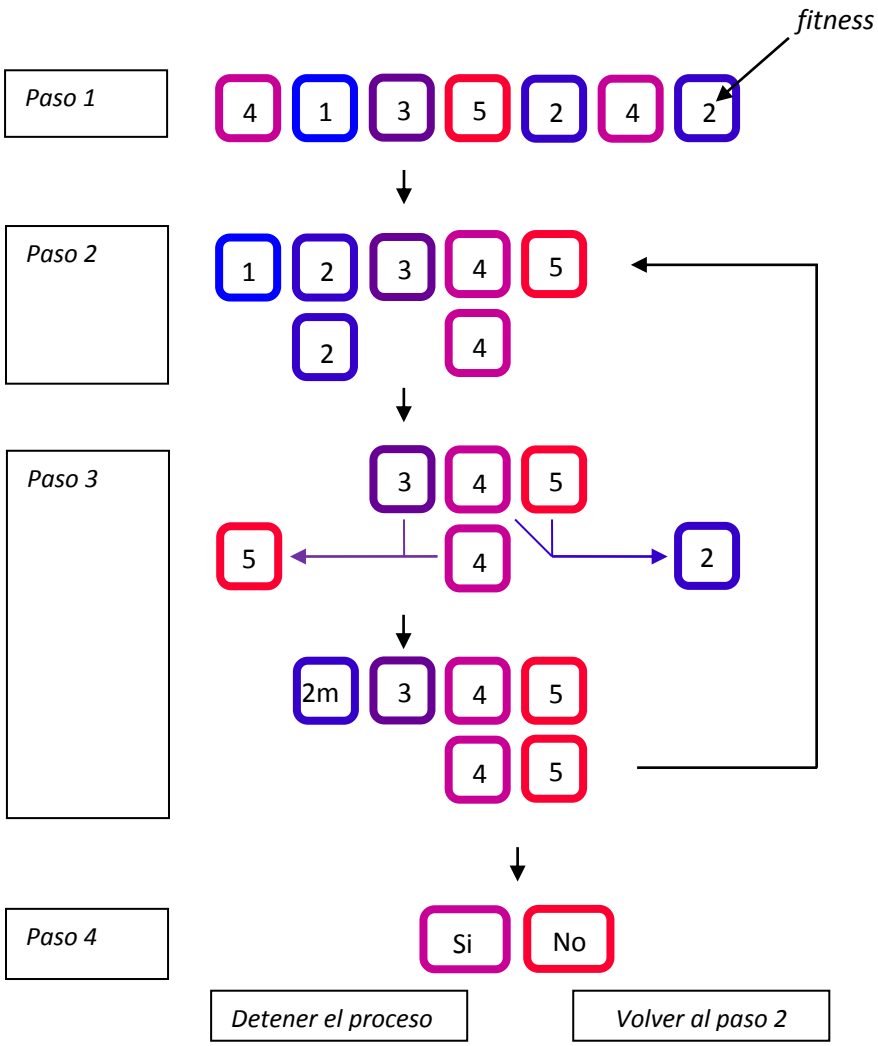


Figura 20. Esquema general de un Algoritmo Genético. La forma puede variar dependiendo del tipo de problema.

Diferentes Técnicas de Búsqueda

Un algoritmo es una regla geométrica o aritmética que se repite[24]. Desde este punto de vista todo puede ser visto como un algoritmo. Existe una gran cantidad de algoritmos para realizar diferentes tareas, por lo que es necesario saber elegir cuál es el más apropiado. Existen algoritmos para resolver problemas complejos de matemáticas, manipular geometría, hacer búsquedas por internet, para reconocer patrones, simular comportamientos, etc.

¡Todo es algoritmo![25]

La parte compleja es elegir el algoritmo que mejor se ajuste a la necesidad que se tiene. Una de las posibilidades es buscar el que consuma menos recursos computacionales en tiempo y espacio. Otra alternativa puede ser buscar el algoritmo más preciso, etc. La eficiencia de un procedimiento (algoritmo) se mide por la cantidad de recursos consumidos, la complejidad para implementarlo y el espacio que usa.

Un algoritmo de búsqueda es un tipo de procedimiento que toma un problema como entrada y retorna una solución a ese problema, después de evaluar un número posible de soluciones. El grupo de todas las posibles soluciones al problema es llamado espacio de búsqueda.

La búsqueda de un resultado por medio de programación puede ser realizada básicamente por 3 grandes grupos de algoritmos.

- ***Calculus based*:** en este grupo encontramos a los algoritmos *greedy*, *fibonnaci*, *newton*, *binary search*, *sequence search*. Este tipo de procedimientos opera iterando diferentes cálculos matemáticos hasta alcanzar la solución.
- **Estocásticos (*Random*):** El cual se divide en algoritmos supervisados (*guided*) y no supervisados (*non guided*). Los supervisados tienen una arquitectura diseñada para conocer el espacio de soluciones. En este grupo encontramos todos los relacionados con la inteligencia artificial, como por ejemplo los AGs, las redes neuronales y otras técnicas basadas en la evolución. Los no supervisados, son aquellos que no tienen conocimientos sobre el espacio de búsqueda. Usualmente son conocidos también como algoritmos de fuerza bruta o de búsqueda uniforme. Dentro de esta categoría están los algoritmos de montecarlo y las vegas.
- ***Enumertives***: Básicamente funcionan descomponiendo el problema en pequeños sub problemas. Dentro de este grupo encontramos los algoritmos de *divide and conquer*, *backtracking*, *dinamic programming*, *branch and bound*.

24 BALMOND, Cecil,. Informal, Prestel, New York, 2002, pág. 112
25 CHAITIN, Gregory,. Leibniz, Information, Math and Physics.
http://www.cs.auckland.ac.nz/CDMTCS//chaitin/kirchberg.pdf, [08/08/10], pág. 9.

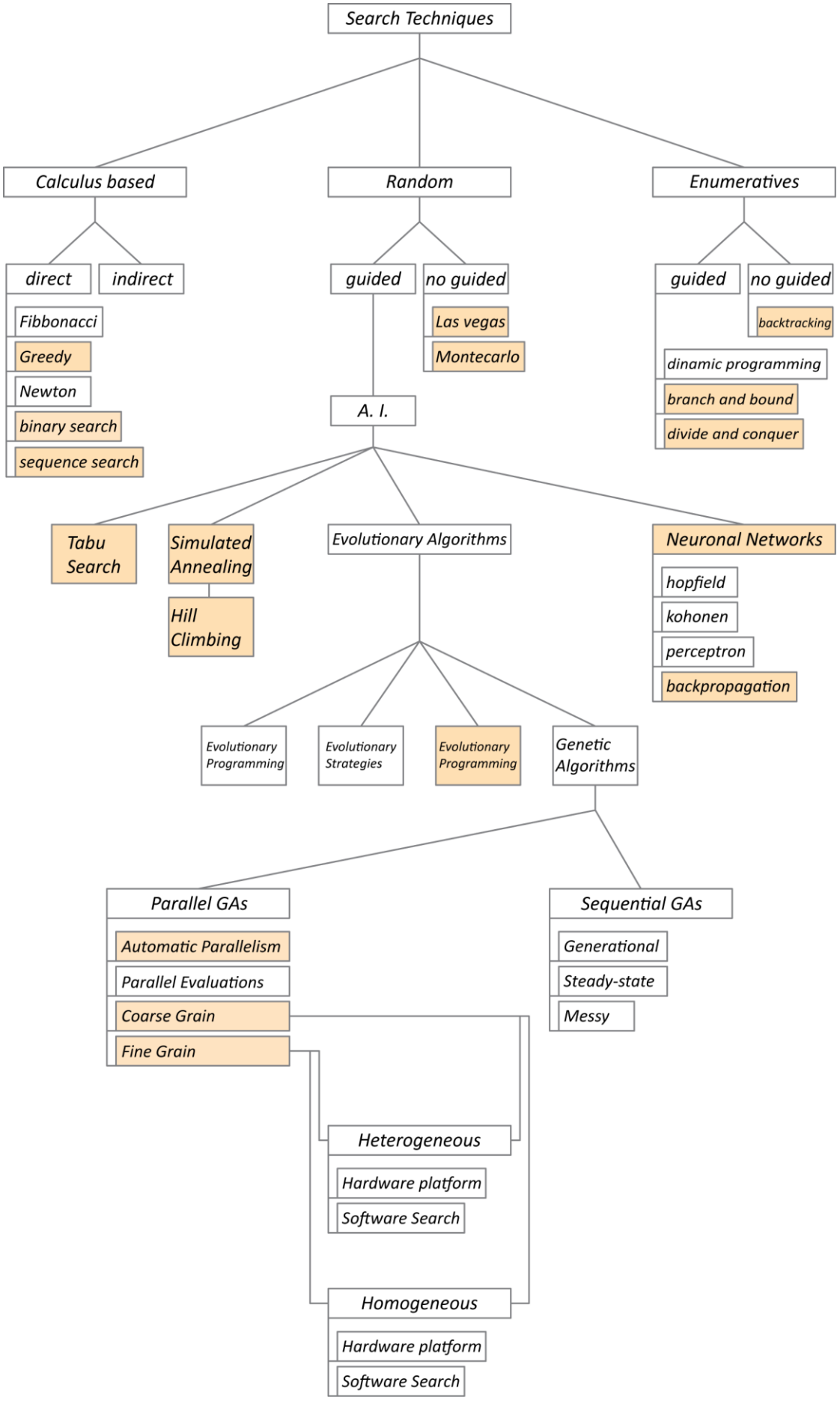

Search Techniques
Calculus based
Random
Enumeratives
direct
indirect
guided
no guided
guided
no guided
Fibbonacci
Greedy
Newton
binary search
sequence search
Las vegas
Montecarlo
backtracking
dinamic programming
branch and bound
divide and conquer
A. I.
Tabu Search
Simulated Annealing
Hill Climbing
Evolutionary Algorithms
Neuronal Networks
hopfield
kohonen
perceptron
backpropagation
Evolutionary Programming
Evolutionary Strategies
Evolutionary Programming
Genetic Algorithms
Parallel GAs
Sequential GAs
Automatic Parallelism
Parallel Evaluations
Coarse Grain
Fine Grain
Generational
Steady-state
Messy
Heterogeneous
Hardware platform
Software Search
Homogeneous
Hardware platform
Software Search

Figura 21. Clasificación de algoritmos de búsqueda según su estructura de búsqueda. Basado en http://eddyalfaro.galeon.com/geneticos.html [08/08/10], y completado por el autor. Los cuadros coloreados son los algoritmos descritos a continuación.

5.1. Calculus based

5.1.1. Direct -> Greedy Algorithm

Este tipo de algoritmo encuentra la opción óptima para cada paso local, con la esperanza de encontrar la solución general global. Usualmente es aplicado a problemas de optimización. El proceso funciona escogiendo en cada paso el mejor elemento posible que pertenezca al conjunto de soluciones y es conocido como el elemento más prometedor. Entonces se elimina el elemento antiguo dentro del conjunto y seguidamente se comprueba si el nuevo elemento produce una solución factible. En caso de que así sea, se incluye ese elemento en el conjunto. De lo contrario se descarta el nuevo elemento y se pasa al siguiente dentro del conjunto. Ejemplos de algoritmos greedy son:

- Algoritmos de Kruskal.
- Algoritmo de Prim.
- Algoritmo de Dijkstra.
- Algoritmo de ubicación óptima.

5.1.2. Direct -> Binary Search

Este algoritmo consiste en buscar la solución comparando cada una de las posibles soluciones entre ellas. Si imaginamos que tenemos una lista de números enteros aleatorios con 1001 ítems Lista (1000) y lo que queremos es encontrar el número más cercano a 20, la estrategia consiste en comparar cada ítem y conservar cuál de ellos es el más cercano. Si el número 20, o el resultado que se busca no se encuentran dentro de los primeros 900 elementos de la lista, es posible de llegar a realizar cerca de 10.000 comparaciones antes de saber cuál es el número más cercano de 20. Este tipo de modelo de búsqueda es bastante ineficiente.

Un proceso previo a realizar las comparaciones consistiría en ordenar la lista, de modo de mejorar la actuación de búsqueda.

Lista (1000) original: {-34, 56, 500, -623, 845, 16, -345,…}

Lista (1000) ordenada por magnitud: {-623, -345, -34, 16, 56, 500, 845,…}

Si la comparación se realiza en la segunda lista, esta comienza en -623, en el siguiente ítem el valor es -345, es más cerca de 20, entonces existe la idea de que el próximo número esté más cerca de 20 que su sucesor. La búsqueda continúa hasta encontrar el 16, pero el algoritmo no está seguro de lo que viene después, podría ser el 17, 18, 19 incluso el 20 o el 21. Sin embargo el valor es 56 que se aleja bastante del resultado que se busca, entonces el algoritmo habiendo

comprobado el 56 aborta el proceso y retorna como valor el 16. El mejor caso se da cuando el resultado que se busca se encuentra al principio de la serie, ya que aumenta el rendimiento considerablemente. De todas maneras, una lista ordenada es mucho más eficiente que una desordenada.

La búsqueda binaria trabaja dividiendo el espacio de soluciones en sucesivas mitades y comprobando si el valor que busca se encuentra por encima o por debajo del valor actual. Desafortunadamente tiene un funcionamiento óptimo sólo en listas ordenadas.

5.1.3. Direct -> Sequence Search

Este algoritmo se utiliza cuando la lista no puede ser ordenada. Opera buscando secuencialmente en todos los elementos dentro de la lista hasta que es encontrado o hasta que la lista se acabe. La existencia del elemento que se busca sólo es comprobable cuando se encuentra o cuando se ha recorrido la lista completa. Esta estrategia es poco práctica, ya que hay que recorrer todo la lista y dependiendo del tamaño puede tener un alto consumo computacional.

5.2. Random

5.2.1. Guided -> A.I. -> Taboo Search

Este tipo de algoritmo se suele usar en problemas de combinatoria (!), donde la solución es conocida, pero el problema se presenta en la combinación de las partes para obtener una solución. La búsqueda tabú utiliza un procedimiento de búsqueda local o por vecindades. La optimización progresa moviéndose de una solución a otra. Cuando encuentra una solución la marca como “tabú” y el algoritmo comienza buscar en otras áreas del espacio de búsqueda. Lo más importante de este tipo de algoritmos es la lista tabú, la cual consiste en guardar en memoria las soluciones visitadas recientemente. El problema con esto es que se pueden dejar soluciones muy buenas guardadas dentro de la lista y continuar buscando por otro lado. Sin embargo otro algoritmo puede corregir esta imperfección, modificando el estado tabú de la solución y volviéndolo a incluir en el espacio de búsqueda. Usualmente se compara la solución actual con la mejor de la lista y al compararse, se opta por la mejor.

Las ventajas de este tipo de algoritmo es que se pueden centrar las búsquedas en ciertos atributos. Por ejemplo no combinar tal elemento con este otro o mantener un orden establecido dentro de la combinatoria. Otra ventaja es que se pueden evitar movimientos o combinaciones no deseadas.

5.2.2. Guided -> A.I. -> Simulated Annealing

La idea de este algoritmo toma prestado su nombre del proceso industrial en el cual el acero es calentado por encima de su punto de fusión y luego se enfría gradualmente para eliminar defectos en su estructura cristalina. Con esto se produce un entramado de átomos más estable y regular. En este tipo de algoritmos existe también una función de aptitud, pero en cambio de existir una *población* de *individuos*, sólo existe una solución candidata.

El recocido simulado también añade el concepto de "temperatura", que es una cantidad numérica global que disminuye gradualmente en el tiempo. En cada paso del algoritmo la solución muta, luego la aptitud de la nueva solución se compara con la aptitud de la solución anterior, si es mayor se conserva. En caso contrario, el algoritmo toma la decisión de conservarla o descartarla en base a la temperatura. En un principio el factor de la temperatura es alto y puede darse el caso de que los cambios introducidos en el *individuo* causen disminución en su factor de aptitud y se utilice como base para la siguiente ronda de evaluación. Pero al disminuir la temperatura, el algoritmo se va haciendo cada vez más propenso a aceptar sólo los cambios que aumentan la aptitud. El proceso finaliza cuando la temperatura alcanza el cero y el sistema se "congela".

El algoritmo de recocido simulado (*Simulated Annealing*) pertenece a una clase de algoritmos de búsqueda local que es comúnmente llamada algoritmos de umbral. A menudo tiene aplicaciones en ingeniería del diseño.[26]

El algoritmo de recocido simulado pertenece a la clase de algoritmos de búsqueda local y sus principales características son:

1. Se comportan bien en una amplia gama de problemas reales
2. Permiten hacer un análisis de la convergencia.

El algoritmo de recocido simulado se divide en etapas. Donde a cada una de ellas le corresponde una temperatura menor que a la etapa anterior. Por esto es necesario dentro de este tipo de algoritmo implementar un criterio de cambio de temperatura, "cuánto tiempo" se debe esperar en cada etapa para dar lugar a que el sistema alcance su "equilibrio térmico".

Para esto se debe definir una temperatura inicial T^0 que permita casi todo tipo de soluciones al problema. En esta parte el sistema tiene un alto grado de libertad para generar resultados muy diversos.

El factor de cambio de temperatura puede ser definido por una serie parámetros. El parámetro K que es igual a la cantidad de iteraciones por cada etapa. Es equivalente a la cantidad de tiempo que vamos a esperar a que el sistema alcance su equilibrio térmico para una temperatura T^0. El parámetro A que es igual a la cantidad de aceptaciones que se permiten hacer en cada etapa. El parámetro T^0 que es la temperatura que disminuye dentro del sistema haciendo más difícil aceptar mejoras dentro de la solución.

La solución inicial factible puede ser denominada i0, está solución usualmente es tomada al azar del conjunto de soluciones factibles.[27]

---

26 KIRKPATRIK, S., GELATT, C.D. y VECCHI, M. P. Optimization by simulated annealing. Revista Science, cap, 220, (1983), Pág. 671 – 678.

5.2.3. Guided -> A.I. -> Hill Climbing

El algoritmo tipo *Hill climbing*, es muy similar a un AG, aunque es más sistemático y menos aleatorio. Este tipo de algoritmo comienza con una solución al problema el cual tiene dos particularidades. Primero es codificada, como se haría en un AG y segundo es generada al azar. Luego la cadena se muta para producir un nuevo resultado. Si este es mejor que la solución anterior se conserva, de lo contrario se repite el proceso. Las mutaciones se producen hasta que no se pueda encontrar una mejor solución que provoque un incremento en la aptitud de la solución actual. Al final del proceso, esta solución es devuelta como resultado. Este algoritmo puede ser muy rápido en espacios de búsqueda definidos o regulares, pero muy malo en espacios con mucho ruido.

En ciertas páginas web y libros sitúan este algoritmo como del tipo *greedy*, ya que siempre se espera que haga la mejor elección en cada iteración. Sin embargo, esto no sucede así para otros científicos, que basan la potencia del *Hill climbing* en la mutación aleatoria de la solución. Lo que hace a veces elegir opciones menos óptimas al principio con la esperanza de retornar soluciones mejores más adelante.

"*Este tipo de algoritmo es del tipo voraz, lo que significa que siempre hace la mejor elección disponible en cada paso, con la esperanza de que de esta manera se puede obtener el mejor resultado global*"[28].

Este tipo de algoritmo puede tomar muchas formas dependiendo del número de dimensiones que tenga la solución del problema, el valor de incremento y la dirección que debe seguir. Usualmente cuando comienza a ejecutarse el algoritmo, comienza a buscar a partir de su posición y en la dirección más empinada que encuentre. La ventaja de esto es que tiene muy poco costo computacional. *Hill climbing* es útil cuando su función de aptitud está muy bien definida y los operadores de transición no alteran la futura aplicación de otro.

Sus características principales es que al ser guiado (*guided*) utiliza la información de la optimización para elegir una dirección de búsqueda u otra. No es exhaustivo en el sentido que no busca en todo el espacio de búsqueda. Además no garantiza encontrar las mejores soluciones pero si unas buenas. También se le considera eficiente en centrar su búsqueda en un espacio determinado.

Su desventaja principal es que la solución que retorna no es la más óptima.

---

27 HAUPT, Randy. HAUPT, Sue Ellen. Practical Genetic Algorithms, Hoboken: John Wiley & Sons, 1994 seg. Edit. 2004.
28 kOZA, John., KEANE, Martin., STREETER, Matthew., Evolving Inventions, Scientific American, February 2003, p. 52-59

5.2.4. Guided -> A.I. -> Evolutionary Algorithms -> Evolutionary Programming

Bajo el término algoritmos evolutivos se entiende a todos los algoritmos basados en procedimientos inspirados en mecanismos de la evolución natural, como por ejemplo, algoritmos genéticos, algoritmos evolutivos, programación evolutiva, etc. Utilizando un vocabulario de genética natural, se asocia el concepto de *individuo* a una solución factible al problema, y el de *población* a un conjunto de soluciones factibles. Los *individuos* están formados por unidades (genes o caracteres) ordenados en sucesión lineal, siendo evaluados a través de su aptitud (*fitness*).

La programación evolutiva envuelve a una serie de algoritmos basados en procesos aleatorios de selección y de variación que fueron diseñados inicialmente para resolver problemas de optimización de parámetros. El proceso de selección favorece con mayor probabilidad a aquellos *individuos* con mejor aptitud (*fitness*) para ser sometidos a los operadores genéticos de variación: recombinación y/o mutación.

El proceso de recombinación permite una mezcla de información de los padres para traspasarlo a los descendientes, los cuales son mutados para modificar ciertas características en el *individuo*. El proceso evolutivo se inicia con una *población* inicial de μ *individuos*, que evoluciona de generación en generación por medio de los operadores de variación, cuya finalidad es preservar la diversidad genética de la *población*. El proceso se repite hasta cumplir el criterio de parada. Este puede ser una aptitud esperada por parte de la *población* o un número de iteraciones establecidas.

Muchas veces la *población* inicial es determinada aleatoriamente[29]. El proceso de selección elige los μ *individuos* de mejor *fitness*, los cuales frecuentemente están referidos a un ranking dentro de la *población*.

Este tipo de estrategias utiliza variaciones del operador genético mutación, llamado mutación de vecindad. Este se basa en la idea original de las estrategias evolutivas, en la que todos los genes varían en su vecindad el valor actual.

La mutación de vecindad consiste en que el valor de cada gen del *individuo* mutado se obtiene de una vecindad del valor actual; si mk = mink (maxk) entonces mk se incrementa en 1. Si mink < mk < maxk se determina en forma aleatoria si mk se incrementa o disminuye en 1.

Por otro lado la recombinación es una variación del cruce típico, llamado recombinación intermedia de Mühlebein y Schlierkamp - Voosen[30]. La recombinación intermedia toma valores de los genes de los descendientes que se generan como una ponderación aleatoria de los valores de los padres. Su cálculo es el siguiente:

---

29 PIERREVAL, H and TAUTOU, L. Using Evolutionary Algorithms and Simulation for the Optimization of Manufacturing Systems. IIE Transactions, 29(3): 181-189, 1997.
MICHALEWICZ, Z. Genetic Algorithms+Data Structures=Evolution Programs. 2nd.US, Ed. Springer. 1999. p. 387

30 MÜHLENBEIN, H. and SCHLIERKAMP-VOOSEN, D. Predictive Models for the Breeder Genetic Algorithm: I. Continuous Parameter Optimization. Evolutionary Computation, 1 (1), pp. 25-49, 1993.

$$vk = ak * m1k + (1 - ak) * msk$$

Con mik el valor del gen k del padre i (i = 1,2), ak ~ U[-d, 1 + d]. El valor vk se redondea al entero más cercano; si mink ≤ vk ≤ maxk entonces mk = vk, en otro caso, si vk < mink (vk > maxk) entonces mk = mink (mk = maxk)[31].

La programación evolutiva es más cercana a las estrategias de la evolución que los algoritmos genéticos. Sin embargo, no incluyen necesariamente los tres operadores básicos de los AGs: Cruce, mutación y selección. Otra diferencia es que la programación evolutiva no trabaja con codificación, los operadores y la evaluación es aplicada directamente sobre el modelo, lo que dificulta bastante la tarea de programación.

5.2.5. Guided -> A.I. -> Evolutionary Algorithms -> Genetic Algorithms -> Parallel GAs -> Automatic Parallelism, Coarse Grain, Fine Grain.

Los AGs tratan con poblaciones y no con *individuos* particulares, excepto cuando son evaluados, seleccionados y cruzados con otros miembros de la *población*. En todas las demás operaciones, los AGs, pueden ejecutarse de manera paralela.

La programación paralela o computación paralela es una técnica de programación en la que muchas instrucciones se ejecutan simultáneamente[32]. Esto quiere decir que por medio del código se puede controlar el flujo de trabajo que existe en los diferentes núcleos del ordenador. Desde hace unos años es común ver ordenadores con dos núcleos y servidores u ordenados de gama alta con 4 o más núcleos. La tendencia es que se continúen agregando núcleos en los procesadores. Las causas son alto rendimiento, bajo consumo de energía y temperatura, entre otras. Antiguamente para mejorar el rendimiento de una aplicación, bastaba con cambiar el programa a un ordenador con un procesador más rápido. Sin embargo esto es obsoleto, porque el rendimiento de una aplicación depende de cómo se administran los núcleos y no aumentando la velocidad.

La idea de los AGs paralelos es aprovecharse de esta idea para realizar ciertas tareas de manera paralela y aprovechar este nuevo tipo de tecnología. Básicamente existen tres maneras de paralelizar AGs.

Paralelismo de grano fino

Está estrategia administra los núcleos desde el código y destina a cada uno una parte del algoritmo (selección, cruce, mutación y representación) sobre la *población* común. La paralelización se realiza en los bucles y de forma automática en los compiladores.

31 MICHALEWICZ, Z. Genetic Algorithms+Data Structures=Evolution Programs. 2nd.US, Ed. Springer. 1999. Pág: 387

32 ALMASI, G.S., and GOTTLIEB A. Highly parallel computing. California EEUU, Benjamin-Cummings publishers, 1989, http://portal.acm.org/citation.cfm?id=1011116.1011127, [11/08/10], pág: 165-66.

Paralelismo de grano grueso

Este tipo de paralelismo administra los núcleos dependiendo de los cálculos sobre datos. Además se puede aplicar a diferentes configuraciones de núcleos y procesadores. Las ventajas son que la distribución de datos se realiza bajo ciertas directivas del programador y luego el programa es quien se encarga de asignar las diferentes tareas a los diferentes núcleos. También se pueden administrar los núcleos mediante el paso de mensajes coordinados dependiendo de la cantidad y tipo de actividad de los núcleos.

Una de las grandes ventajas de los AGs es que permiten que sus operaciones puedan ejecutarse en paralelo. Parte del éxito de un AG depende del tamaño de la *población* con la que se trabaja. Por lo tanto una *población* grande (*población* > 1000 *individuos*) requiere de más memoria para ser almacenada y también más tiempo para encontrar una solución (converger). Está técnica de programación administra mejor el espacio de memoria y permite evaluar más soluciones en menos tiempo. Además es posible trabajar con poblaciones más grandes, reducir el costo computacional y mejorar el desempeño del AG.

5.2.6. Guided -> A.I. -> Neuronal Network

Este modelo está basado en la manera de como las neuronas se relacionan en el cerebro. Biológicamente, un cerebro aprende mediante la reorganización de las conexiones sinápticas entre las neuronas que lo componen. De la misma manera, las redes neuronales tienen un gran número de procesadores virtuales interconectados que de forma simplificada simulan la funcionalidad de las neuronas biológicas. El sistema funciona basado en una serie de capas procesadoras llamadas nodos unidas por conexiones direccionales. Existe una capa de entrada, una capa de salida y ninguna o más capas ocultas en el medio. El sistema comienza con un patrón inicial en la capa de entrada y luego sus conexiones (nodos) transmiten su señal a la siguiente capa a la que están conectados. Si la suma de todas las entradas en una neurona virtual es mayor que el umbral de activación, la neurona se activa y transmite su propia señal a la siguiente capa. El patrón de activación siempre se propaga en un sentido hasta que alcanza la capa de salida la cual muestra el resultado. De la misma manera que el sistema nervioso en los organismos biológicos, las redes neuronales aprenden y mejoran su rendimiento a lo largo del tiempo.

Las redes neuronales artificiales tienen poca relación con problemas asociados a flujos de datos y optimización. Es por esto que las redes neuronales son usadas básicamente para la identificación de patrones, con el objetivo de aprender y detectar cuando estos patrones cambian. Las redes neuronales tienen la habilidad de aprender mediante un proceso que se llama etapa de aprendizaje. Está consiste en proporcionar datos de entrada a su vez que se le indica cuál es la salida esperada. Luego aprenden y son capaces de auto-organizarse creando su propia representación de la información del problema. Este tipo de algoritmos se usa para el reconocimiento de la voz, la identificación de células cancerígenas, catálogo de astros en el espacio, prótesis robóticas, contestadores de voz, etc.

Las ventajas que tienen es que son muy flexibles. Aceptan cambios no importantes en la entrada de información, sin alterar las reglas aprendidas. Su estructura es paralela, por lo cual si es implementada en ordenadores o dispositivos electrónicos se pueden obtener respuestas en tiempo real.

Las desventajas se centran en el consumo de recursos computacionales. Su capacidad se basa en la habilidad de procesar información en paralelo, procesando múltiples bloques de datos

simultáneamente. Otro inconveniente es la falta de reglas definitorias que ayuden a construir una red para un problema dado, hay muchos factores a tomar en cuenta: Algoritmo de aprendizaje, estructura, número de neuronas por capa, número de capas, representación de datos, etc. Por lo que implementar y mantener el algoritmo toma mucho tiempo y recursos.

5.2.7. Guided -> A.I. -> Neuronal Network -> Backpropagation

Existen muchos tipos de algoritmos de aprendizaje basados en la estructura de las redes neuronales. El algoritmo de *backpropagation*, es también un algoritmo de aprendizaje pero es usado para entrenar (enseñar) a redes neuronales. El algoritmo opera intentando reducir un error cuadrático por medio de sucesivas derivadas parciales de dicho error con respecto a los parámetros de la red neuronal.

Este algoritmo funciona calculando el error de salida en cada capa. Va corrigiendo la red desde la capa de salida hasta la capa de entrada ajustando los pesos en cada nodo. El proceso se va repitiendo de forma iterativa, hasta que todas las capas han ajustado sus pesos y el error cuadrático se ha minimizado.

Las deficiencias ocurren cuando los valores de cada nodo toman valores muy grandes y el algoritmo no es capaz de ajustar los pesos. Cuando el espacio de búsqueda es muy ruidoso es muy fácil de caer en mínimos locales. Usualmente, las redes neuronales son entrenadas mediante AGs para evitar este tipo de problemas.

5.2.8. No guided -> Las vegas

El algoritmo de las vegas es del tipo no supervisado y probabilístico, como los algoritmos numéricos, *sherwood* y *montecarlo*. Este tipo de algoritmos funciona dejando al azar cierta toma de decisiones, porque encontrar la decisión óptima tomaría mucho tiempo y porque el problema puede tener múltiples soluciones buenas. Los numéricos retornan una solución aproximada a problemas en los cuales no existe una respuesta exacta. El método *montecarlo* da una respuesta concreta pero no siempre es la mejor. El método s*herwood* en cambio, su respuesta siempre es correcta. Las vegas no siempre retorna una respuesta, pero esta siempre es correcta.

El algoritmo de las Vegas nunca da un resultado erróneo, consiste en tomas de decisiones completamente aleatorias, como jugar en un casino de las vegas, para encontrar el resultado a la solución del problema. Su eficiencia depende del campo de búsqueda por lo que este tipo de algoritmo puede llegar a usar grandes recursos informáticos. La probabilidad de encontrar una solución aumenta con el tiempo de ejecución y el número de comparaciones realizadas.

Usualmente se usan para resolver problemas para los que no se conoce ningún algoritmo eficiente. Sin embargo su diseño permite tomar decisiones que impiden llegar a una solución. Algunas veces el algoritmo retorna una serie de soluciones con buenas aptitudes al problema. Con el tiempo de ejecución los *individuos* van mejorando cada vez más y la probabilidad de éxito mejora y también el consumo de los recursos informáticos.

5.2.9. No guided -> Montecarlo

Montecarlo es un tipo de algoritmo usado para aproximar expresiones matemáticas complejas. El método de *montecarlo* no puede asegurar que el resultado que arroja es correcto. Puede equivocarse en el resultado, pero nunca más de dos veces sobre el *individuo* que está evaluando. Cuando retorna un error no avisa. Aunque encuentra soluciones correctas con buena probabilidad. Este tipo de algoritmo tiene un bajo consumo de recursos computacionales. Es del tipo no determinístico, lo que significa que con la misma entrada de datos ofrece distintos resultados sin saber de antemano cuál será el resultado de la ejecución.

Los métodos de *montecarlo* abarcan una colección de técnicas que permiten obtener soluciones de problemas matemáticos o físicos por medio de pruebas aleatorias repetidas. En la práctica, las pruebas aleatorias se sustituyen por resultados de ciertos cálculos realizados con números aleatorios.

Los problemas en los cuales se han aplicado los métodos de *montecarlo* están relacionados con la probabilidad y la estadística. Por ejemplo, para entender el comportamiento de ciertas partículas con alguna influencia determinista y casual. Usando el método *montecarlo* se pueden simular partículas donde las reglas casuales y deterministas son imitadas exactamente. Considerando un número muy grande de partículas, es posible responder a preguntas referentes a la distribución y probabilidad de las partículas, que por ejemplo, atravesarán un obstáculo de un determinado espesor dentro de un cierto periodo de tiempo. En este ejemplo Montecarlo es usado como modelo probabilístico o de juego, cuya solución está en relación directa con la solución de la ecuación.

Los métodos de Montecarlo se usan para estudiar modelos estocásticos artificiales de procesos físicos o matemáticos (ejemplo de las partículas). Un problema de probabilidades, del cual siempre se puede obtener una solución aproximada mediante pruebas repetidas, se considera como la herramienta para encontrar la solución numérica de una ecuación funcional. O de igual manera, un problema físico que exige un modelo de análisis, donde el modelo probabilístico (las probabilidades de que ocurra esto o aquello dentro del modelo) son suficientes para encontrar una solución.

5.3. Enumerative

Este tipo de algoritmos utiliza la estrategia de descomponer el problema en múltiples partes pequeñas y luego solucionar cada una de ellas para volver a juntarlas y resolver el problema global.

5.3.1. Guided -> Branch and Bound

El algoritmo de *Branch and Bound*, (ramificación y acotamiento) comienza con una relajación del problema y construye un árbol con soluciones enteras dividiendo el conjunto de soluciones factibles de modo de descartar soluciones fraccionarias o incompletas. La idea de descomponer el problema puede llevar a un problema mayor inabordable, por lo que es necesario "podar" el árbol. Es considerado una variante del *backtracking* pero mejorado. Se aplica básicamente en resolver problemas de optimización.

Este algoritmo se esquematiza como un árbol de soluciones, en donde cada rama existe una posible solución posterior al problema actual. Su característica principal es que el algoritmo se encarga de detectar en qué ramificación las soluciones dadas ya no son óptimas, entonces se poda esa rama y se continua la búsqueda por otras partes del árbol.

La estrategia funciona comprobando si la menor rama A para algún árbol nodo (conjunto de candidatos) es mayor que la rama padre para otra rama B, entonces A debe de ser descartada. Este paso es llamado poda y se implementa usando la variable global *m* que graba el mínimo nodo padre visto entre todas las subregiones examinadas hasta entonces. Cualquier nodo cuyo hijo es mayor que *m* puede ser descartado. De esta manera la función se va minimizando.

La eficiencia del método depende de la expansión de los nodos o de la estimación de los nodos padres e hijos. Un problema frecuente es elegir un tipo de estrategia que evite que no se solapen los subconjuntos para ahorrarnos problemas de duplicación de ramas. Es por esto que el método depende estrechamente de los algoritmos de ramificación y poda usados. Por el momento no existe un algoritmo de poda universal que trabaja para todos los tipos de problemas, por lo que es necesario diseñarlos para cada caso en particular.

5.3.2. Guided -> Divide and Conquer

La estrategia de este algoritmo se basa en que para resolver un problema, este se divide en partes más pequeñas tantas veces como sea necesario hasta que la resolución de las partes se torna muy simple de resolver. Luego la solución al problema se construye con las soluciones encontradas. Su estrategia está basada en la lógica de la recursión. Está técnica es la base de los algoritmos de ordenamiento. Su implementación generalmente es recursiva, sin embargo

se puede implementar de una manera no recursiva que almacene las soluciones parciales en una estructura de datos explícita como una pila en C++.[33]

La ventaja de este algoritmo se debe a que es una herramienta muy potente para solucionar problemas complejos. Todo lo que necesita este algoritmo es dividir el problema en sub-problemas más sencillos y éstos a su vez hasta llegar a problemas lo suficientemente simples como para ser resueltos, también llamados casos base.

Una vez ahí se resuelven y se combinan los sub-problemas en orden inverso a su inicio. Dividir los problemas es la parte más compleja del algoritmo, por eso muchas veces la solución que ofrece el proceso no es la mejor sino la más sencilla.

Una ventaja de este tipo de algoritmo es que hacen uso eficiente de la memoria cache. Esto es porque como el problema está subdividido en varias partes, que son lo suficientemente pequeñas para ser resueltas, sin tener acceso a la memoria principal, que es del orden de decenas de veces más lenta. Este tipo de algoritmo se aplica para muchos tipos de problemas, como de ordenación o multiplicación de matrices.

La desventaja de este método es su lentitud en la repetición del proceso recursivo: los gastos indirectos de las llamadas recursivas a la resolución de los sub-problemas, junto al hecho de tener que almacenar los resultados de cada etapa, pueden empeorar cualquier mejora lograda.

5.3.3. No guided -> Backtracking

Este tipo de estrategia está diseñado para encontrar soluciones a problemas que satisfacen restricciones. De manera similar que el método *branch and bound* el *backtracking* se asemeja a un recorrido de un gráfico con forma de árbol. El sistema funciona construyendo una serie de soluciones parciales a medida que progresa el recorrido. Estas soluciones sirven de marco a las regiones en las que se puede encontrar una solución completa. El recorrido tiene éxito si, procediendo de esta forma, se puede definir por completo una solución. Cuando el algoritmo no encuentra una solución parcial que no puede completar, vuelve atrás, eliminando los elementos que se hubieran añadido en cada fase. Cuando vuelve a un nodo que tiene uno o más vecinos sin explorar continúa la búsqueda por este lado.

El tipo de problema para el cual fue diseñado el *backtracking* son aquellos del tipo completo, donde el orden de los elementos de la solución no importa. Estos problemas consisten en un conjunto de variables, donde a cada una se debe asignar un valor sujeto a las restricciones del problema. La técnica va creando todas las posibles combinaciones de elementos para obtener una solución. La principal característica consiste en que se pueden evitar ciertas combinaciones del problema. Durante la búsqueda si encuentra una solución incorrecta, retrocede hasta el paso anterior y toma la siguiente alternativa, si aun así no hay más elecciones que produzcan un resultado prometedor el sistema falla. Normalmente este tipo de algoritmos es recursivo, así en cada llamada al procedimiento se toma una variable y se le asignan todos los valores posibles, llamando a su vez al procedimiento para cada uno de los nuevos estados. La diferencia con el método *branch and bound*, es que *backtracking* tiene una

[33] http://www.algoritmia.net/articles.php?id=34 [22/06/10]

función de cota, de manera que no se generan ciertos estados si no van a conducir a ninguna solución, el objetivo de esto es ahorrar memoria y tiempo de ejecución.

5.4. Ventajas de los AGs por sobre otros tipos de algoritmos

La primera ventaja es que los AGs son paralelos, ya que su descendencia múltiple, puede explorar el espacio de soluciones en múltiples direcciones a la vez. Por ejemplo, si un camino no resulta prometedor, lo eliminan y continúan por otras zonas de búsqueda que sean más prometedoras. De esta manera existe una mayor probabilidad de encontrar una solución óptima. Otros métodos exploran el espacio de soluciones hacia un resultado en una dirección. Si la solución que descubren resulta sub-óptima, no existe otra alternativa de búsqueda.

Los AGs sondean cada uno de los espacios en los que se encuentra una posible solución. Tras cada evaluación se va obteniendo un valor cada vez más alto de la aptitud media de la *población*. Un AGs que evalúa un número pequeño de *individuos* en realidad está evaluando un grupo de *individuos* mucho más grande. Opera de la misma manera que un encuestador se lleva la opinión sobre un tema de una *población* habiendo evaluado un grupo pequeño de personas. De esta misma manera, el AG puede dirigirse hacia un espacio con *individuos* más aptos y encontrar el mejor de ese grupo. Esto es lo que se llama teoría del esquema[34]. Gracias al paralelismo que soportan los AGs les permite evaluar muchos más esquemas a la vez, los AGs funcionan particularmente bien resolviendo problemas cuyo espacio de soluciones potenciales es realmente grande y ruidoso.

La mayoría de los problemas que caen en esta categoría se conocen como no lineales. En un problema lineal la aptitud de cada *individuo* es independiente, con lo que la optimización depende sólo de maximizar o minimizar un valor. En cambio los problemas no lineales son los más comunes en el mundo real y constan en que cada valor está relacionado, por lo que se trata de buscar como resultado un máximo global que potencie de alguna manera cada una de las partes. Otra ventaja es que los AGs se desenvuelven bien en problemas con un paisaje adaptativo complejo, donde la función o el mejor resultado nunca son claros. Cambian con el tiempo o no existe un resultado único.

El desafío de programar AGs consiste en evitar máximos locales. Es por esto que un AG se debe componer de sus cuatro componentes principales: evaluación, selección, cruzamiento y mutación. El cruzamiento es la clave que distingue los AGs de los otros métodos de resolución. Sin este componente cada solución individual va por su cuenta, explorando el espacio de búsqueda en sus inmediaciones sin referencia de lo que el resto de *individuos* puedan haber descubierto. Sin embargo, con el cruzamiento, la transferencia de información de un *individuo* beneficia a los candidatos más prósperos. La mezcla y la combinación, tienen un potencial de producir una descendencia con las virtudes de los padres.

34 HOLLAND, John. Genetic Algorithms. Scientific American, (1992), pág. 66-72.

El problema de encontrar el máximo global en un espacio con muchos óptimos locales se conoce como el dilema de la exploración versus explotación[35]. Una vez que se encuentra una estrategia que parece funcionar satisfactoriamente: ¿debería centrarse en hacer el mejor uso de la estrategia, o buscar otras? Abandonar una estrategia para buscar otras nuevas casi asegura un fracaso, por lo menos en un corto plazo, pero al conservar una sola estrategia se excluye un universo por descubrir de nuevas estrategias que podrían ser mejores[36].

Los AGs no saben nada sobre el problema que resuelven. No utilizan la información específica conocida a priori para guiar los pasos y realizar cambios para encontrar la solución deseada. Los AGs son un "*Blind WatchMaker*" (relojero ciego)[37], realizan cambios aleatoriamente en las soluciones candidatas y luego utilizan la función de aptitud para determinar si esos cambios producen una mejora o no.

Como las decisiones están basadas en aleatoriedad, todos los caminos de búsqueda posibles están abiertos teóricamente. Es por esto que cualquier proceso de búsqueda que tenga un conocimiento previo comenzara descartando una cantidad de caminos perdiendo así cualquier solución novedosa que pueda existir[38]. De la misma manera fracasara, cuando la técnica dependa de un conocimiento y este no se encuentre disponible. Mediante sus componentes de paralelismo, cruzamiento y mutación. Los AGs pueden viajar extensamente por el paisaje de búsqueda explorando regiones que otros algoritmos no tendrán en cuenta y revelando soluciones asombrosas e inesperadas.

5.5. Limitaciones de los AGs

Una consideración importante cuando se crea un AG es definir una representación del problema, de cierta manera el lenguaje de programación debe ser robusto y estable, capaz de soportar cambios aleatorios que no produzcan errores o resultados falsos.

Existen diferentes maneras de representar AGs. La mayoría consiste en definir los *individuos* como listas de números o enteros, donde cada número representa algún aspecto del *individuo* en cuestión. La segunda manera es desarrollar cadenas binarias de 0 y 1 donde cada valor puede representar la ausencia o presencia de cierta característica del *individuo*.

El problema de cómo escribir la función de aptitud debe considerarse con mucho cuidado para que por un lado se pueda alcanzar una mayor aptitud y verdaderamente resulte una solución mejor para el problema dado. El segundo problema con la función de aptitud, es que siempre arrojará un resultado, pero habrá que estar muy atento a que ese resultado realmente sea bueno y no un simple resultado. Imagina que se usa un programa para analizar fluidos y el

35 HOLLAND, John. Genetic Algorithms. Scientific American, (1992), pág. 66-72.
36 HOLLAND, John. Genetic Algorithms. Scientific American, (1992), pág. 66-72.
37 DAWKINS, R., The Blind Watchmaker: Why the Evidence of Evolution Reveals a Universe without Design. W. W. Norton, (1996)
38 KOZA, J., FOREST, B, DAVID, A., MARTIN, K. Genetic Programming III: Darwinian Invention and Problem Solving. Morgan Kaufmann Publishers, (1999)

resultado parece bueno. Pero la verdad es que el fluido que has utilizado no le corresponde la viscosidad que debería ser, le corresponde otra mucho más densa, entonces el resultado es equivocado. A pesar de estas limitaciones, todas ellas pueden solucionarse.

Los AGs no son complejos de programar, basta una buena noción para trabajar con listas y se puede comenzar a hacer pruebas. Las verdaderas dificultades de los AGs, se centran en el ajuste de los valores, como por ejemplo en el *fitness*, en la mutación, en los porcentajes de cruce, en el número de iteraciones y en el tamaño de las poblaciones.

Los AGs, nunca devuelven un valor exacto. Sino que retornaran uno muy cercano al exacto. Este es un problema que no se puede resolver con AG, ya que por estrategia la naturaleza no busca mejorar y hacer más resistentes a los *individuos*, sino que busca mantener una diversidad genética para no desaparecer.

La alternativa es la selección de genes (o selección de *individuo*): los *individuos* altruistas llegan a extinguirse en beneficio de los egoístas, que predominarán en el grupo. Los genes han construido una gran variedad de "máquinas" para prosperar explotándolas, de modo que un gen puede ser considerado como una unidad que sobrevive a través de un gran número de cuerpos sucesivos e individuales. Así, un gen es definido como una porción de material cromosómico que, potencialmente, permanece durante suficientes generaciones como para servir como una unidad de selección natural. El *individuo* es demasiado grande y efímero como para ser considerado unidad de selección. Un gen es considerado bueno, es decir, que permanecen muchas generaciones, si vela por sí mismo, si es egoísta. La evolución será el proceso por el que algunos genes se hacen más numerosos y otros disminuyen en el acervo genético.

Todos los genes controlan el comportamiento de su máquina de supervivencia, no de manera directa, sino indirectamente. Los genes preparan la máquina con antelación, y luego esta se haya bajo su propia responsabilidad. Los genes obran a largo plazo mediante la síntesis proteica, pero se trata de un proceso lento. Por tanto, los genes construyen su máquina por anticipado, de la mejor forma posible y programándola con antelación.

Por tanto, el comportamiento está regido por el egoísmo de los genes de cada organismo, y no por el altruismo de cada *individuo* con respecto a los demás miembros de su especie.

La selección darwiniana no actúa directamente sobre los genes. Aquí es donde entra el concepto de fenotipo ampliado: los efectos fenotípicos de un gen deben considerarse como todos los efectos que tiene sobre el mundo, efectos sobre sí mismo, sobre otros genes, sobre la maquinaria que los porta, sobre otros organismos e incluso sobre el mundo inerte. El efecto de un gen depende de una cascada determinada de síntesis de proteínas que, al final, desemboca en el fenotipo deseado.[39]

39 DAWKINS, Richard. El Gen Egoísta, Barcelona, Salvat editores S.A., 2000.

Tipo de algoritmos	**Ventaja**	**Desventaja**	**Comentarios**
Montecarlo	*Bajo consumo de recursos*	*No asegura resultado optimo*	*Ofrece múltiples resultados, usado para aproximar expresiones matemáticas, y determinar probabilidades.*
Las Vegas	*Nunca resultados falsos*	*Altos recursos informáticos*	*Su eficiencia depende del tamaño del campo de búsqueda, fuerza bruta.*
Divide and conquer	*Simple y resuelve problemas complejos*	*Demasiado lento Dependiendo del t.*	*Su estrategia se basa en recursión y en dividir el problema en pequeños problema.*
Binary search	*Rápido y preciso*	*Sólo funciona bien en listas ordenadas*	*Las listas ordenadas son poco frecuentes, es el problema ideal.*
Neuronal network	*Aprende, flexible, se auto organiza*	*Altos recursos informáticos*	*Las Redes neuronales son usadas para el reconocimiento de patrones*
Hill climbing	*Muy sistemático Poco recursos*	*El resultado, puede no ser el mejor*	*Es parecido a un AGs, es menos aleatorio, puede no optimizar.*
Simulated annealing	*universalidad*	*lentitud*	*A menudo tiene aplicaciones en la ingeniería del diseño, como estructuras.*
Algoritmo Genético	*4x4 off-Road*	*Definir la función de aptitud.*	*Paralelismo, cada vez se va haciendo más preciso en sus resultados.*

Figura 22. Tabla comparativa de diferentes métodos de búsqueda.

5.6. Lenguajes

Hoy en día existen una gran cantidad de lenguajes en los que se pueden desarrollar AGs, como *C, C++, C#, Java, MatLab, Mathematica, Visual Basic, Lisp, Oracle, Prolog, Fortrand*, entre otros. Los conocimientos del autor básicamente son dos Visual Basic 6.0 y C++, sin experiencia en librerías específicas de gráfica.

Los trabajos desarrollados por el autor se basan principalmente en *Rhinoscript*, que es una modificación técnica del lenguaje de programación *Visual Basic Script* para ser usado dentro del software de modelado *Rhinoceros*. El otro lenguaje utilizado para desarrollar los experimentos es *GCScript* que es una modificación de C# dentro del programa *GenerativeComponents* y *C#* con Visual Studio 2008.

Lenguaje	*Lenguajes asociados*	*Características*	*Desventajas*	*Recursos*
Visual basic	*Visual basic.net, Rhinoscript, visual basic script, basic*	*Guiado por eventos y centrado en un motor de formularios que facilita el desarrollo de aplicaciones gráficas.* *Implementación limitada de la programación orientada a objetos (POO).*	*1_ los ejecutables generados son relativamente lentos* *2_ no permite la programación a bajo nivel.* *3_ no incluye operadores de desplazamiento de bits como parte del lenguaje.* *4_ no permite el manejo de memoria dinámica, punteros, etc. Como parte del lenguaje.*	*Algunos ejemplos de simples AGs.*
C ++	*C, C#, java, AWK, pearl, ME, GCScript*	*Programación estructurada, programación genérica y la POO* *Permite trabajar tanto a alto como a bajo nivel.*	*1_ la implementación gráfica es más complicada de alcanzar ya sea a través del software maya o Rhinoceros.*	*Muchos ejemplos de AGs en C++*

Figura 23. Tabla comparativa, sobre diferentes lenguajes para la implementación de AGs.

Estado del Arte

El capítulo Estado del Arte, presenta los principales trabajos realizados con Algoritmos Genéticos que han sido inspiración en el desarrollo de esta tesis. Estos trabajos, algunos teóricos y otros presentados en el congreso de computación evolutiva más serio de mundo "*GECCO*"[40], no necesariamente están relacionados con la arquitectura. Algunos de ellos exploran campos de la estructura, de las finanzas y de la búsqueda de formas. Los trabajos están presentados según el orden en que fueron aportando al trabajo de investigación. Por ejemplo, los primeros trabajos presentan conceptos más generales sobre AGs e introducen ideas como la computación evolutiva y la generación de formas. En cambio, los últimos trabajos son más técnicos, describen con más precisión los AGs y además son la base para el desarrollo de los proyectos en el capítulo 8 de la tesis.

6.1. Selección de trabajos realizados con Algoritmos Genéticos y Técnicas Evolutivas.

6.1.1. An Evolutionary Architecture, John Frazer (1975)

Referencia: FRAZER, John. *An Evolutionary Architecture*, Londres, Architectural Association, 1995, http://www.aaschool.ac.uk/publications/ea/intro.html, [14/08/10].

La computación evolutiva y los algoritmos genéticos aplicados a la arquitectura, probablemente comienza con los trabajos de John Frazer y las investigaciones con sus alumnos en los 60s. En su libro *An Evolutionary Architecture*, plantea que existe un tipo de arquitectura basada en los procesos evolutivos: *arquitectura evolutiva*; la cual investiga los procesos de generación de formas mediante estrategias evolutivas.

John Frazer propone desarrollar una especie de modelo digital de naturaleza que genere fuerzas que modifiquen la forma en la arquitectura. La idea es poder construir un mundo digital dentro del ordenador, en el cual se simulen las diferentes fuerzas que existen en la realidad y que estas modifiquen un edificio a lo largo de generaciones. Lo interesante es usar la evolución natural como poder creativo para diseñar modelos arquitectónicos digitales-naturales que respondan a los cambios del medio ambiente simulados dentro del ordenador.

Frazer en su libro plantea que la arquitectura es considerada como una forma de vida artificial, donde el modelo está sujeto, como en el mundo natural a los principios de la morfogénesis

40 Genetic and Evolutionary Computation Conference, http://www.sigevo.org/. [23/08/10].

(descrita por Turing), al código genético, a la replicación y a la selección. El propósito de la arquitectura evolutiva es conseguir en la construcción del ambiente, el comportamiento simbiótico y el balance metabólico que son características del medio ambiente natural. Los conceptos arquitectónicos son expresados como reglas de generación, donde la evolución puede ser acelerada y testeada.

Los experimentos descritos por John Frazer en su libro son de ideas y lenguaje, más asociado a un genetista que a un arquitecto. Frazer no usa los típicos términos de los arquitectos. Él dice que el código-script es un motor generador de formas arquitectónicas el cual utiliza el ordenador para dar grandes y largos pasos de evolución en un pequeño espacio de tiempo. Con cada iteración del algoritmo mejora la aptitud de los *individuos*, entonces lo que la naturaleza tarda hacer en millones de años con el ordenador se tarda unos minutos. También que las formas y resultados son inesperados.

Estás técnicas han sido permanentemente limitadas a los problemas de ingeniería y que sólo hoy están siendo factibles de aplicar a problemas asociados a la arquitectura y a construir en un ambiente digital. Para conseguir esto, se ha de tener que considerar como la forma estructural debe de ser codificada bajo una técnica llamada “Algoritmo Genético”, y como debe de ser definida, y como los procesos metabólicos y morfológicos son adaptados para su interacción de la forma construida y su medio ambiente. Cuando estos problemas son resueltos, el computador puede ser usado en un sentido inusual al habitual. Como un *acelerador evolutivo* y un *generador de fuerzas*.

John F. en su propuesta introduce el concepto de arquitectura evolutiva, indicando que la naturaleza de lo biológico, las analogías científicas y sus relaciones de ideas están en el contexto científico. En el proceso de trabajo están relacionadas las ciencias naturales y las nuevas ciencias de cibernética, complejidad y caos.

El libro describe el campo de la fuerza de la genética en la arquitectura, que explora al menos un posible futuro basado en el diseño artificial de vida, sugiriendo una nueva forma de diseñar un artefacto interconectado, evolucionando en armonía con las fuerzas naturales e incluso con las sociales. Explica una nueva técnica basada en el uso del ordenador para el diseño de modelos bajo una lógica interna, en vez de una forma externa.

De cierta manera Frazer sienta las bases de las millones de posibilidades y aplicaciones que tiene la programación y las estrategias evolutivas en la arquitectura. Frazer por años ha influenciado una serie de arquitectos, ingenieros y artistas a desarrollar su trabajo.

Dentro de sus proyectos más destacables está el *Universal Constructor* (1990), el cual es un modelo de trabajo de un entorno interactivo auto-organizado (*Working Model of a self-organizing interactive environment*). El modelo consiste en una base rectangular la cual es llamada paisaje y una serie de células. Estás se encajan entre sí de manera vertical en específicos lugares dentro del paisaje. Las células pueden representar dos estados, estructura o paisaje. Cada una contiene un circuito integrado el cuál puede comunicarse con las células de alrededor. Además se pueden comunicar con un controlador de procesos. Cada unidad (célula) tiene un código de identificación y es mostrado a través de 8 diodos. Los cuales también sirven para comunicarse entre sí. El sistema completo sabe que es cada célula y cuál es su ubicación. Las células pueden mostrar en el estado en que se encuentran y pasar mensajes a las personas que interactúen con el modelo.

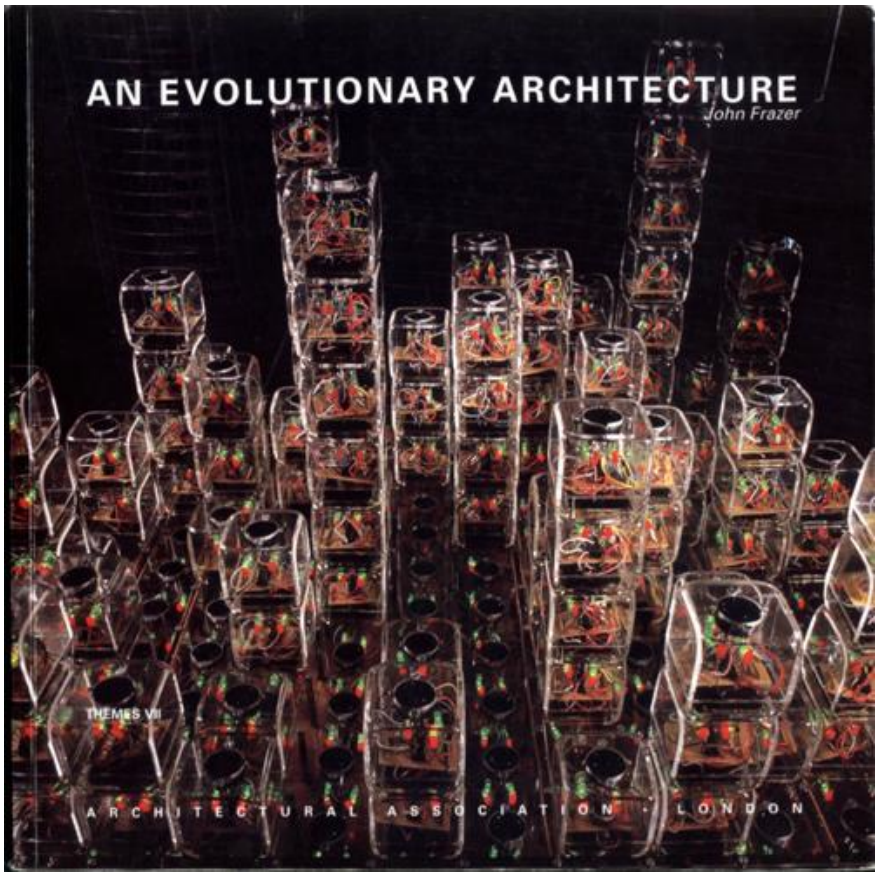


Figura 24: Portada del libro y la imagen del Universal Constructor http://www.aaschool.ac.uk/publications/ea/intro.html [23/06/10]

6.1.2. *Lidabashi* Subway Station, Makoto Sey Watanabe Architect's Office, 2000.

Referencia: Estación de metro *Lidabashi* en Japón por Makoto Sey Watanabe Architect's office, desarrollada el 2000. http://www.makoto-architect.com/subway/subway_2e.html [14/08/10]

La idea principal de este proyecto no es utilizar el ordenador para crear formas inusuales, sino que utilizar el ordenador para pensar, como si fuese la extensión del cerebro.

Los arquitectos utilizan el ordenador para generar propuestas que resuelven condiciones de diseño. A diferencia del diseño convencional donde se especifica todo, los arquitectos han dejado ciertas cuestiones sin decidir, definiendo ciertas condiciones que el diseño debe cumplir. De esta manera, han utilizado algoritmos evolutivos para definir ciertas restricciones en diseño y estructura para luego iterar los modelos hasta que cumplan con las condiciones que fueron pre establecidas en la programación.

El diseño metodológico tradicional es de tipo arriba hacia abajo. En este caso las decisiones son de abajo hacia arriba, especificando las relaciones entre las partes mientras se satisfacen todas las necesidades en un equilibro general. Así el software generado para la estación de metro de *Lidabashi* define simples principios que parecen escondidos dentro de lo que parece un total desorden.

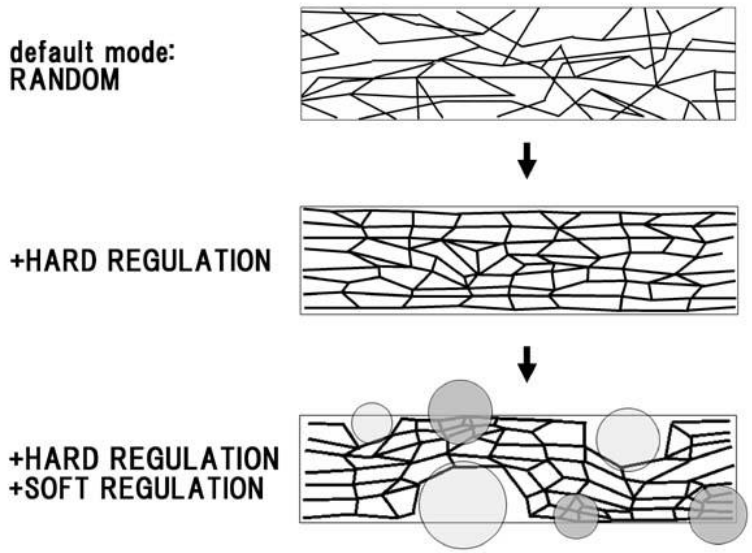


Figura 25: Imagen de la estructura, cumpliendo requerimientos estructurales y de espacio. Imagen tomada de http://www.makoto-architect.com/subway/syb_e.htm [12/01/08].

La estación de *Lidabashi* es un proyecto que se encuentra dentro de "*the Induction cities*" un trabajo que comenzó en la década de los 80 como investigación y desarrollo dentro del campo de diseño inteligente. La primera fase del proyecto fue dedicado a los ensayos y la segunda fase al desarrollo de un grupo de programas para "*hacer ciudades*". En la tercera fase fue aplicado a un proyecto de arquitectura: "*web frame*". Mientras que la cuarta fase se encuentra actualmente en marcha y envuelve el desarrollo de programas para encontrar lo "emocional" y otras condiciones aplicadas previamente al proyecto de la estación.

La metodología del proyecto tiene mucho en común con los trabajos de la naturaleza, pero no imita la naturaleza o copia la forma de las plantas. Tiene una estructura cercana al ecosistema natural porque sus valores y programas son parecidos a los valores y programas del ecosistema. Las formas son similares porque han sido producidas por procesos similares a los producidos por la naturaleza. De esta manera las formas de la estación son el resultado de un proceso de satisfacer una serie de restricciones y no el punto de partida como habría de esperar en otra estrategia de diseño.

Los programas de ordenadores no pueden diseñar todo, pero son mejores que los cerebros humanos para tareas como resolver complejos puzles. Sin embargo los ordenadores no pueden imaginar. Ordenadores y personas cada uno tienen sus habilidades especiales y se debe tener mejores respuestas cuando cooperan en un diseño. Lo que se busca es un método flexible, que también venga de acuerdo con la buena respuesta a los problemas de diseño.

A diferencia del diseño convencional, que intenta decidir todo, en este proyecto los arquitectos han creado un método que induce resultados y encuentra condiciones. En este sentido puede ser llamado diseño inductivo, aunque también puede ser llamado diseño generativo, ya que las características del resultado mejoran con las generaciones.

El proyecto de la estación de *Lidabashi* está dividido en dos grandes partes. En el interior de la estación los arquitectos diseñaron el *web frame*, una especie de estructura tubular generada mediante un algoritmo que satisface una serie de restricciones. En el exterior *wing* que corresponde a la torre de ventilación de la estación y todos los sistemas de climatización. Su diseño también está basado en un algoritmo de generación.

WEB FRAME

El diseño de la *web frame* está basado en una grilla regular de tubos de acero de diámetro 5cm aproximadamente. La estructura es deformada y manipulada para ser introducida en los pasillos de la estación de metro. El objetivo de la estructura es "conducir" a los usuarios por la estación y servir también como una parrilla de iluminación. Es por esto que los arquitectos definen una serie de condiciones que se dividen en dos tipos. Existen condiciones duras que deben ser cumplidas en un 100% y condiciones blandas que no especifican en que porcentaje deben ser cumplidas.

Condiciones duras:

- Restricciones del espacio.
- Condiciones impuestas por cada componente.

Condiciones Blandas:

- Intenciones de diseño.

La primera condición absoluta no permite margen de improvisación. La estructura debe adaptarse a las paredes y altura de la estación, por lo que es una condición dura. Las condiciones son impuestas por el tipo de instalación que puede ser realizada. Por ejemplo, es difícil lograr una intersección en un mismo punto de más de cinco tubos con una variación de un grado cada uno. Por lo tanto se establecen parámetros individuales para cada nodo de la grilla estableciendo condiciones específicas.

Por otro lado, las condiciones blandas se convierten en otro parámetro. Se especifica la posición aproximada de la estructura, el volumen total de partes y el espacio deseado es guardado como una especificación blanda. La libertad fácilmente puede caer en el caos, pero un importante elemento de este concepto es dar la apariencia de caos cuando de hecho obedece a ciertas reglas estrictas y regulares.

Cuando el resultado puede aparecer arbitrario y voluntario, las condiciones necesarias son rigurosamente encontradas. Lo mismo podemos decir del caos y de todas las formas del fenómeno de complejidad. Explican los arquitectos de *Makoto Sey Watanabe Architects*.

WING

Wing, es la torre de ventilación de la estación y donde se encuentran los equipos de climatización.

El equipo busco un mecanismo de auto-generación usando un programa de computación. El programa no se encuentra completo por el momento, pero los arquitectos intentaron incorporar la dinámica estructural usada en el *Web Frame* como condición para ser resuelta por el programa. Para el diseño del marco estructural convencional, elaboraron una grilla de trabajo a la cual se le aplicaron deformaciones al que luego se le evaluaron sus efectos. Los materiales seleccionados cumplen los requerimientos estructurales que la grilla requiere y son incorporados como condicionantes dentro del programa. El marco de la estructura aumenta en densidad y espesor donde las fuerzas y el estrés son grandes. En cambio, es abierto donde las fuerzas son débiles. Los materiales se fusionan para transmitir mejor las cargas de estrés, los pilares y vigas del material se extienden y separan para formar una única estructura sin distinciones entre verticales y horizontales.

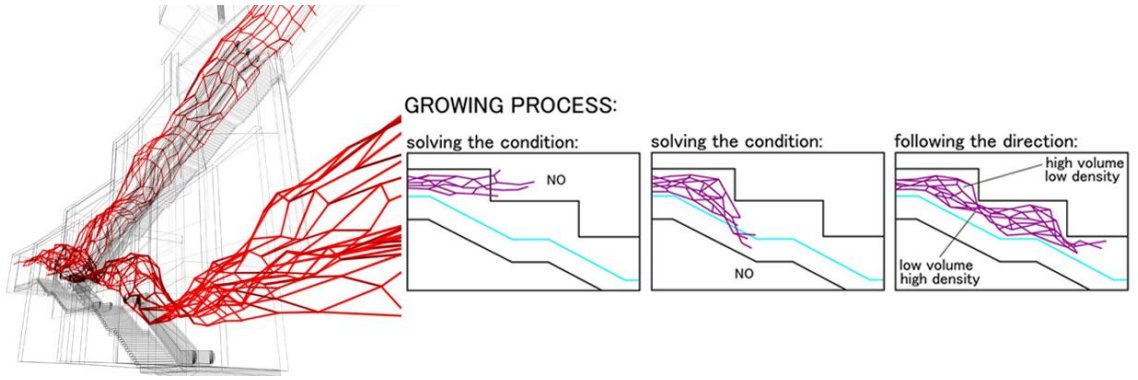


Figura 26. Imágenes del proceso de crecimiento de la estructura. Se puede ver las restricciones espaciales que debía cumplir dentro de la estación y además se señalan los puntos donde la densidad de la grilla debía incrementarse para mantener su estabilidad. http://www.makoto-architect.com/subway/subway_2e.html [15/08/10].

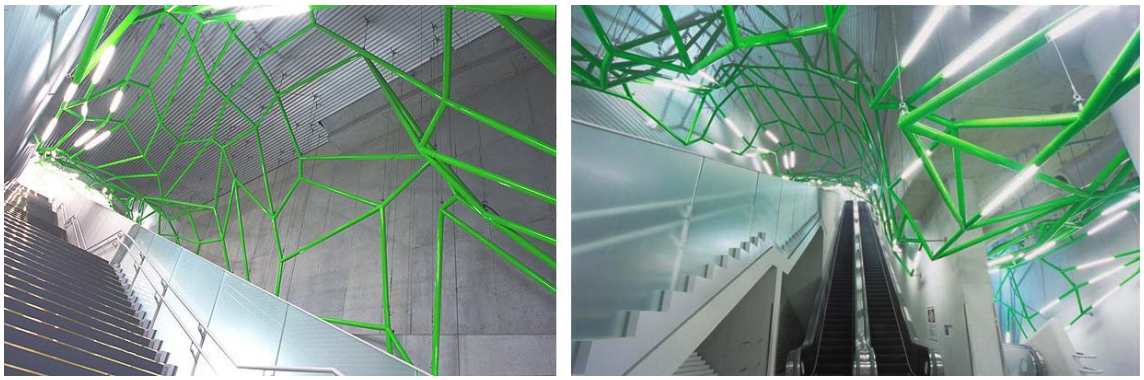


Figura 27. Es posible apreciar en las imágenes como la estructura sube por las escaleras y se adapta a las condicionantes espaciales de la estación. La estructura cuelga de una serie de cables de aceros desde el techo de la estación. http://www.makoto-architect.com/subway/subway_2e.html [15/08/10].

Figura 28. Estructura de la torre de ventilación. La primera imagen muestra parte de los experimentos de generación de la torre, muy parecidas al ala de un insecto. La segunda imagen muestra la estructura de la torre acabada. http://www.makoto-architect.com/subway/subway_2e.html [15/08/10].

Figura 29. Imágenes de la torre construida en la estación de *Lidabashi*. http://www.makoto-architect.com/subway/subway_2e.html [15/08/10].

6.1.3. Architectural Constraints in a Generative Design System: Interpreting Energy Consumption Levels, Luisa Caldas y Leslie Norford (2001)

Referencia: CALDAS, Luisa., NORFORD Leslie. Architectural Constraints in a Generative Design System: Interpreting Energy Consumption Levels. Massachusetts Institute of Technology. Seventh International IBPSA Conference. Rio de Janeiro: Brazil. (2001) Pág.1397 – 1404. http://www.inive.org/members_area/medias/pdf/Inive\IBPSA\UFSC558.pdf [15/08/10].

Este proyecto investiga las posibilidades de codificar un proyecto de arquitectura en un sistema de diseño generativo, usando como modelo de prueba la escuela de arquitectura de Oporto (Portugal), diseñado por Álvaro Siza.

Basado en el lenguaje derivado del diseño original de Siza, el Sistema Generativo (SG) basado en algoritmos genéticos y el programa de simulación de edificios DOE-2.1E, crean soluciones de fachadas como resultado del consumo anual de energía, actuando a la vez como un mecanismo de diagnóstico de los problemas que ocurren en el edificio actual.

El resultado sugiere que el SG puede ser una completa herramienta para los arquitectos mientras dura el proceso de diseño, identificando aéreas de potenciales problemas y sugiriendo soluciones alternativas. El experimento también fue testeado en otras ubicaciones geográficas aparte de Oporto. La capacidad de los algoritmos fue adaptarse a diferentes climas con la misma tipología de fachada.

INTRODUCCIÓN

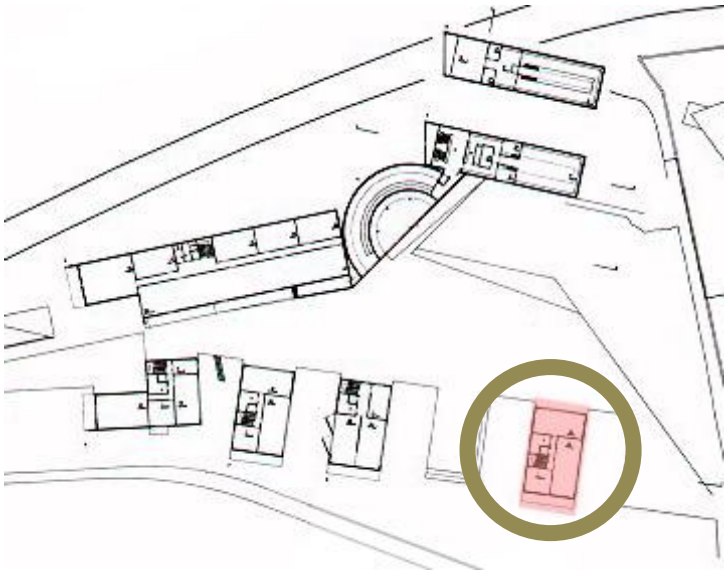
El Sistema Generativo consiste en un método para la creación de nuevos diseños. Un procedimiento para evaluarlos y un mecanismo para desarrollar soluciones para un mejor desempeño en términos de los criterios seleccionados. Se han utilizado algoritmos genéticos

como búsqueda y motor de optimización y se utilizó el software de simulación de edificios DOE-2.1E[41] para evaluar el uso de la iluminación natural y el rendimiento térmico anual del consumo de energía. El proyecto incluye el software GenOpt[42] para los propósitos de optimización de la fachada. El principal inconveniente de este método de búsqueda es que funciona bien sólo en problemas de hasta 10 variables.

Se eligió el AG como mecanismo de búsqueda por el hecho de ser capaz de manipular en forma paralela una *población* de soluciones. El AG comienza generando un número de posibles soluciones al problema, calcula su aptitud (valor de la función) y aplica un básico operador genético de reproducción, cruzamiento y mutación para la *población* inicial, dentro del proceso estocástico *Hill-climbing*. Esto generara una nueva *población*, pero esta vez con una aptitud media superior a la anterior, que volverá hacer evaluada. El ciclo volverá a repetirse por el número de generaciones establecidas por el usuario.

La escuela de arquitectura de Álvaro Siza en Oporto fue usada como modelo de testeo por tener una reglas de composición idóneas en sus fachadas y proveer un excelente marco de trabajo. Ya que el proyecto es de grandes dimensiones y de varios edificios que conforman el conjunto. Por esto el proyecto se focalizó en la torre H.

La escuela de Arquitectura de Oporto Fue diseñada y construida entre los años 1984 y 1996 por Álvaro Siza. El sitio tiene una fuerte inclinación hacia el rio Douro. Siza optó por distribuir las actividades académicas entre diferentes unidades espaciales. Los estudios y salas de clase se encuentran en las torres E, F, G y H. En cambio la biblioteca, el auditorio y los servicios administrativos se encuentran ubicados en el lado opuesto.



41 El Software DOE-2.1E Fue desarrollado por James J. Hirsch y (JJH) asociados en colaboración con el Lawrence Berkeley National Laboratory (LBNL) y con fondos del United State Departament Of Energy (USDOE).
42 GenOpt es un programa de optimización de múltiples parámetros, Según los valores seleccionados por el usuario, el programa determina automáticamente los parámetros del sistema que conducen a un mejor diseño. Desarrollado por Michael Wetter (2000), en colaboración con el Lawrence Berkeley National Laboratory (LBNL).

Figura 30. Planta esquemática de la facultad de arquitectura de Oporto. Destacado aparece el edificio H seleccionado por los investigadores. Fuente de la Imagen: http://www.cidadevirtual.pt/blau/facarch.html [02/03/08]

Los investigadores eligieron la torre H para el estudio por ser rica espacialmente y usar una variedad de recursos arquitectónicos lumínicos: ventanas de diferentes proporciones y medidas, fachadas con diferente orientación, algunas incluyendo parasoles, luz cenital en el último piso y una logia en la fachada sur. En el trabajo de investigación describen que desde la perspectiva computacional, la torre H también presenta algunas características difíciles. Las relaciones internas entre los diferentes espacios y las fuentes de luz dan lugar a una multiplicidad de interacciones que son difíciles de predecir haciendo que el análisis computacional sea una opción interesante.

Figura 31. Vista sud-este del proyecto. La torre H en primera línea, Fuente de la Imagen: http://www.mimoa.eu/projects/Portugal/Porto/Architecture%20Faculty [02/03/08].

El programa de la torre H principalmente son habitaciones y zonas de estudio. Es importante hacer un cuidadoso control de la luz natural con el fin de mantener adecuados niveles de iluminación natural. Esto es clave para las tareas de dibujo y al mismo tiempo para controlar el sol directo sobre las zonas de estudio y las excesivas incidencias solares en las habitaciones.

OBJETIVOS DEL PROYECTO

Los objetivos del proyecto son dos: La incorporación de un lenguaje de condiciones dentro del sistema generativo, porque las soluciones deben ser generadas dentro de ciertos diseños e intenciones y en segundo lugar examinar el sistema generativo como resultados desde la perspectiva del diseño de Álvaro Siza. Además de incluir otros factores aparte de la luz, como los consumos de energía del edificio.

SIMULACIÓN

Para el estudio se utilizó un micro-AG en vez de un AG convencional. La principal diferencia entre estos dos métodos concierne al tamaño de la *población*. Poblaciones típicas de AGs tienen un rango de entre 30 a 200 *individuos*. En cambio un micro-AG usa una pequeña *población* (en este caso, 5 *individuos*), que rápidamente convergen en una solución.

La convergencia es medida comparando los cromosomas de las soluciones individuales. Si difieren por menos de un 5%, se considera que la *población* ha convergido. Cuando esto sucede, los micro-AG recomienzan una nueva *población* aleatoria conservando el *individuo* con la mejor aptitud de la generación anterior. Esta estrategia es conocida como elitista. De esta manera los nuevos *individuos* generan poblaciones mejores a las anteriores, sin perder el rastro de los mejores desde el punto anterior. Una ventaja del uso de micro-AG procede en que el algoritmo tiende a actuar en una búsqueda local alrededor de la mejor solución durante las futuras generaciones. Esta búsqueda local es importante encontrando la mejor solución alrededor de esta pequeña *población* y usualmente muy difícil de implementar en AGs convencionales. Otra ventaja es que el procedimiento de búsqueda es rápido. Los Micro-AG no tienen la inercia de largas poblaciones asociados a los convencionales AGs.

El sistema generativo trabaja completamente en tres dimensiones, incluyendo su geometría, orientación, organización espacial, construcción de materiales, etc. En este estudio, la geometría del edificio, la capa espacial y la construcción de materiales fueron dejadas sin cambios. La búsqueda del espacio algorítmico solo se relaciona con las soluciones de las elevaciones del edificio.

Para el edificio existente diseñado por Siza, el SG genera una *población* de soluciones de fachadas que toman en cuenta el uso de la luz del día en el espacio, el subsecuente uso de la luz artificial y la energía consumida para calor y frío en el edificio. Aunque el uso máximo de la luz natural es una meta alcanzable, el control del calor ganado y perdido introduce un punto de balance que es archivado para su posterior testeo. Los Investigadores explican que este es el evasivo punto de balance que el ordenador trata de conseguir.

Como se menciona más arriba, el programa DOE-2.1E es usado para calcular la aptitud de cada solución. Por cada espacio individual en la torre H donde la luz de día es disponible, dos puntos de referencias de iluminación son seleccionados (los puntos más lejanos de la ventana) y los valores de luminiscencia objetivo están de acuerdo con el tipo de ocupación y tareas que se desarrollan. Generalmente se usan 500 Lux para estudios y otros espacios de trabajo y 150 Lux para áreas de servicio.

Subsecuentemente DOE-2.1E provee el consumo anual de energía del edificio para una particular solución. Estos valores representan la aptitud de los *individuos* los cuales son pasados dentro del AG como guía del proceso de búsqueda.

ANÁLISIS DEL EDIFICIO EXISTENTE

Debido a la necesidad de encontrar elementos que condujeran al desarrollo de un método para entender y codificar las intenciones de diseño de Alvaro Siza, los desarrolladores del proyecto hicieron una visita a la escuela de arquitectura en enero del 2000. El análisis de los dibujos y la visita al edificio permitió inferir en el diseño de reglas que pudiesen ser consideradas para las existentes elevaciones.

Estás reglas se refieren tanto a los ejes de composición de las fachadas como a las proporciones generales de las aperturas. En la torre H, diferentes reglas son aplicadas para cada elevación, mientras se mantiene una fuerte coherencia sobre todo en el diseño del edificio y las relaciones con los espacios interiores. Por ejemplo, largas ventanas horizontales son siempre usadas en los estudios. La elevación Sur presenta un fuerte eje de simetría en las aberturas, pero introduce otros elementos como aleros y logia. La fachada Norte es aún más simétrica en su composición, con un solo elemento asimétrico. Sin embargo las fachadas Este y Oeste obedecen a diferentes reglas.

Se consideró que la elevación Este debía organizarse a lo largo de dos ejes verticales los cuales acaban en dos diferentes aberturas alineadas. Las proporciones de las aperturas tienden en su mayoría a ser ventanas horizontales, con diferentes variaciones de tamaño y posición. Está interpretación de las existentes reglas de diseño fueron seguidas para determinar los mecanismo de búsqueda del SG y así implementar los límites de geometría.

Las áreas de búsqueda son limitadas por dimensiones máximas y mínimas que las aperturas pueden asumir. Estos límites han dado un suficiente campo para permitir una significante búsqueda espacial que puede emerger en una rica variedad de soluciones.

Por cada apertura en las fachadas, la pequeña área representa el tamaño menor de la ventana y el área grande el área mayor. Para las ventanas horizontales, los límites especificados están destinados a evitar la aparición de aberturas verticales. Este grupo de límites fue propuesto para el control de soluciones para generaciones con ciertas intenciones arquitectónicas que fuesen relacionadas con el diseño de Siza. Cambiando los límites se permite la exploración de varias soluciones de diseño.

MÉTODO

Una vez que los límites son gráficamente establecidos, estos son usados como entrada en el sistema generativo. El rango del tamaño del paso usado es de alrededor de 30 cm para las ventanas y 50 cm para las fachadas exteriores. Después de que el SG termina de ejecutarse, el resultado del proceso de búsqueda es visualizado automáticamente usando un programa de visualización, con el fin de relacionar un determinado diseño correspondiente con un nivel de consumo anual de energía. El dibujo en 2-D obtenido es exportado a un programa CAD que sirve de base para crear manualmente un modelo 3D de la mejor solución generada. Luego el modelo es exportado a un software de renderizado para permitir la producción de imágenes. Además el modelo exportado es simulado con detalle la iluminación para una mejor visualización del SG y comparar los resultados con las soluciones existentes de Siza.

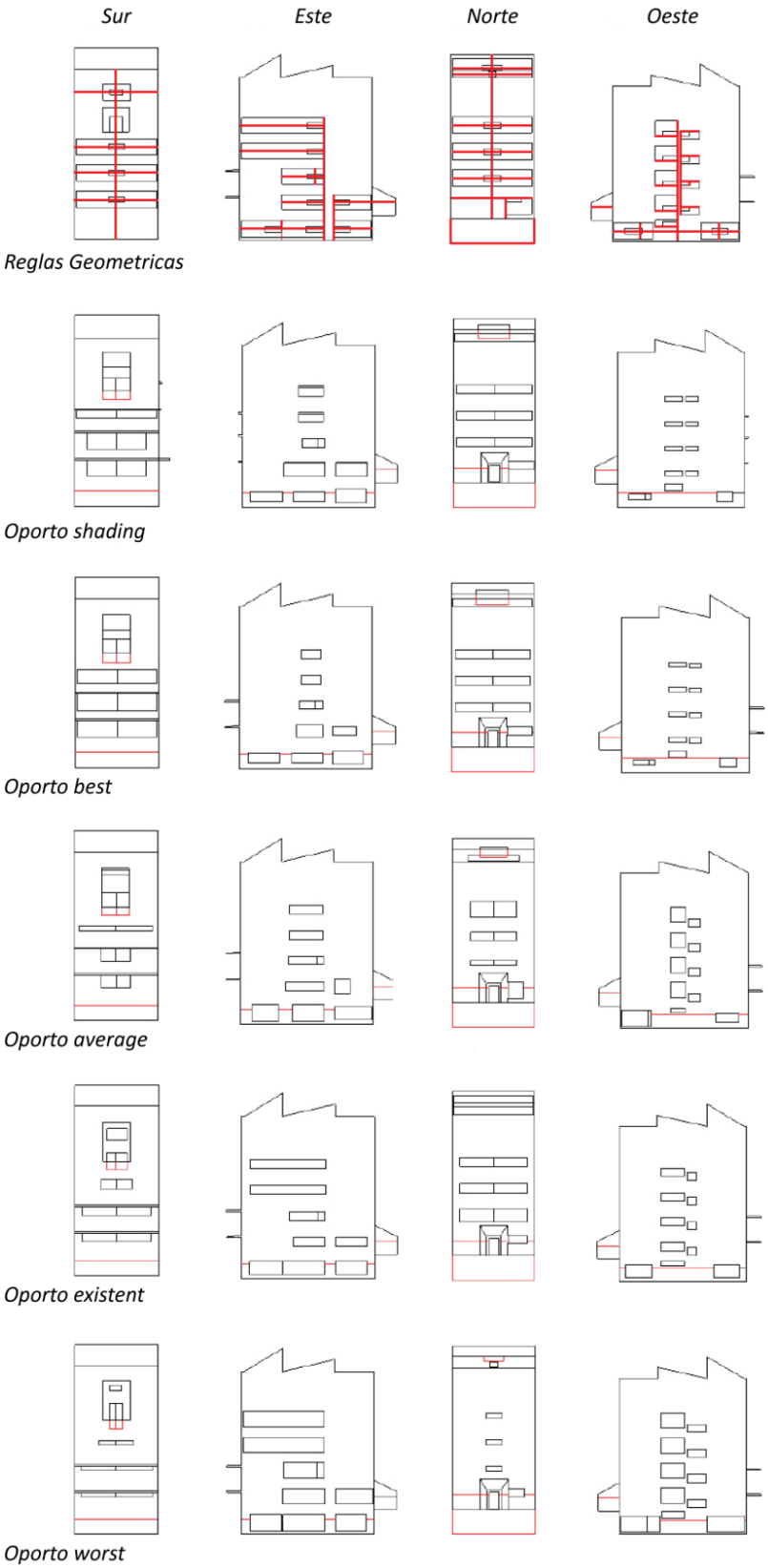


Figura 32. Soluciones arrojadas por el SG para el edificio H en Oporto. La fila *Reglas geométricas*, muestra las diferentes restricciones de diseño para cada una de las fachadas. Los ejes de composición están representados por líneas rojas. CALDAS, L. y NORFORD, L. Architectural Constraints in a Generative Design System: Interpreting Energy Consumption Levels, Rio de Janeiro, Brazil, Agosto 13-15 (2001), pág. 1400.

RESULTADOS

Los resultados del SG coinciden casi exactamente con las soluciones de Siza, pero con algunas diferencias del diseño original.

En la fachada Norte, las largas tiras generadas por el algoritmo son muy aproximadas a las diseñadas por Siza, excepto por la variación de altura de cada una de ellas, denotando que en Oporto el clima templado para usar la luz natural en el estudio claramente compensan las pérdidas de calor a través de largas zonas vidriadas como Siza había predicho.

Las ganancias caloríficas no son significantes en el tema de orientación. Hacia el oeste, el algoritmo usa una ventana pequeña en comparación al diseño de Siza, esto es porque se requiere menor nivel de iluminación es en las zonas de servicios y en las habitaciones más pequeñas.

En la orientación Sur, el SG presenta diferencias significantes en relación al existente. En el diseño de Siza, el segundo y tercer piso tiene al Sur estudios con largas ventanas horizontales sombreadas por aleros de 2m de profundidad. La solución del algoritmo tiende a sugerir que estos aleros son demasiado profundos. Cuando los aleros tienen 2m de profundidad, los tamaños de las ventanas asumen su máxima extensión permitido por los límites. La profundidad del alero evita la incidencia de la luz dentro de la habitación y como contra acto el SG aumenta el tamaño de la ventana. Cuando la profundidad del alero es variable (*Oporto Shading*), el algoritmo reduce a 0.5 m el tamaño de la ventana, dimensiones cercanas a las usadas por Siza. La profundidad de los aleros sigue bloqueando el sol y las altas ganancias solares que suceden en los meses más calientes cuando el sol está más alto. Entonces la fachada Sur es controlada con pequeños aleros. En los meses más fríos, cuando el sol está más bajo en el cielo, la ganancia solar sigue entrando en las habitaciones, reduciendo la necesidad de calefacción.

Oporto-Shading tiene menor consumo de energía que el *Oporto-best*. En el sexto piso la solución del SG para la logia frontal debe de ser entendido en conjunto con la solución de toda la planta, porque básicamente es ocupada por un solo espacio. El cual es iluminado desde arriba por dos lucernarios de cubierta, tipo sierra. El algoritmo aumenta la ventana de la fachada Sur de la logia permitido por los límites y reduce el área acristalada de la cubierta. La cara de la cubierta Norte es una zona de pérdida de calor en invierno. Aumentando las aberturas del Sur permite reducir los lucernarios sin perder demasiada luz-día en el estudio. Por otro lado, el segundo lucernario asume las largas dimensiones posibles que admite el SG. Como en el diseño de Siza, el área del sexto piso no tiene otra fuente de luz. Estos resultados sugieren que la inclinación de la cubierta puede variar para permitir ubicar un largo lucernario.

Las soluciones del cuarto y quinto piso en el lado Sur deben ser analizadas junto con los resultados Este. Porque las habitaciones de estos pisos comparten aberturas Sur y Este. El SG aumenta las ventanas del frente Sur en relación con el diseño existente y simultáneamente reduce las ventanas que están en el frente Este. La orientación Este es desfavorable porque la ganancia solar es alta durante la mañana en los meses de verano y reduce los niveles luz día durante la tarde para la mayor parte del año. Las aperturas hacia el frente Sur actúan mejor en términos de admisión de luz natural y de control de ganancia energética. Cuando el algoritmo tiene la posibilidad de negociar entre dos opciones, siempre favorece la fachada Sur.

La figura 32 muestra que cuando el tamaño de las ventanas de la fachada Este aumentan de tamaño la calidad de la solución disminuye (comparar fachada Este entre *Oporto best* y *Oporto worst*). Sin embargo cuando el SG ha colocado aleros en la fachada (*Oporto – Shading*), se incrementa significativamente el tamaño de las ventanas en el segundo piso. En el estudio la

profundidad de los aleros permite sombrear el sol de la mañana, en cambio en el segundo piso, este tiene solo ventanas en la fachada Este. Para los estudios en el 4 y 5 piso donde ambos son Sur y Este, el SG mantiene pequeñas aperturas (aunque levemente más largas que en el caso donde no existe sombra) con leves aleros y privilegiando las aperturas hacia el Sur.

Para el clima de Oporto, la peor solución encontrada por el SG tiene cerca de un 26% más de consumo de energía que la mejor solución con la variable de sombreado. El diseño de Siza consume alrededor de un 10% más de energía que el mejor modelo propuesto por el SG. Examinando el desglose de los resultados, es posible ver que la solución existente de Siza actúa casi como la mejor solución del SG.

En términos del uso de iluminación natural los resultados previos, demuestran que las principales diferencias entre las dos soluciones de las fachadas Sur y Este así como los lucernarios de la cubierta se producen en los espacios individuales.

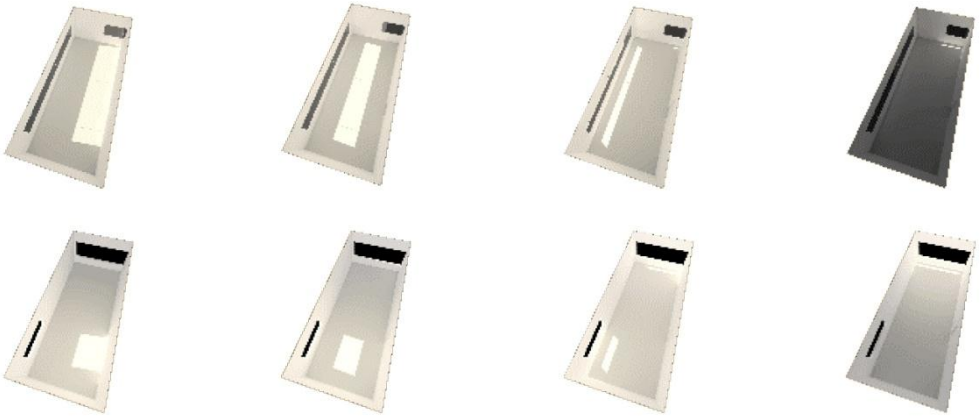


Figura 33. La figura muestra dos filas de *renders* durante el solsticio de verano a las 9:00, 10:00, 12:00 y 15:00 hrs. La fila superior muestra la solución diseñada por Siza y la inferior por el algoritmo genético. La simulación de iluminación combina *radiosity* y *ray tracing* usados para visualizar el resultado de la habitación de estudio en la planta 4. *Ibíd*. 1401

De la figura se puede observar que durante el verano, la larga ventana existente orientada al Este permite la entrada directa del sol en la mayor parte de la habitación (fila superior). En la solución del SG, ambas ventanas no tienen sombras pero la ventana orientada al Sur permite menos sol directo dentro de la habitación durante el día. La pequeña ventana al este permite sol directo en la parte de atrás de la habitación y sólo durante la mañana. Sin embargo la solución del SG no genera tanta sombra como lo hace el modelo de Siza.

Los niveles de consumo de luz artificial aumentan alrededor del 22% desde la peor a la mejor solución. Este valor puede ser probablemente más alto si algunos puntos de referencia lumínica no son ubicados en lo profundo de la habitación (alrededor de 1m desde la muralla más distante desde la ventana). El aumento de la baja temperatura en la peor solución es probablemente porque hay largas ventanas en las caras Este y Oeste. La solución existente no difiere mucho de la mejor solución con la variable de sombreado activada. Incluso aunque alguna ventana Este disminuye su tamaño, otras lo incrementan con los aleros.

Las ventanas Sur tienden a reducir la profundidad de sus aleros para incrementar la luz natural, pero esto a su vez incrementa la ganancia calórica también. Para la calefacción, la mejor solución del SG actúa considerablemente mejor que la solución existente y permite una mayor utilidad de ganancia solar en el invierno y reduce la perdida de calor a través del lucernario Norte y las aperturas Este y Oeste.

Luego las investigadoras prueban el mismo edificio y las mismas reglas geométricas en lugares y climas diferentes para ver el comportamiento del sistema.

PHOENIX

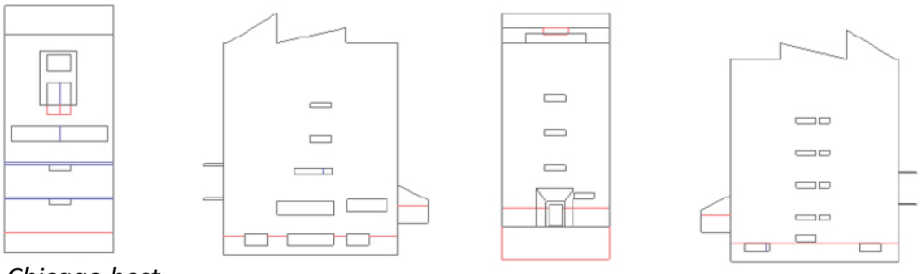
En el Clima caliente de Phoenix, la solución del SG para la fachada Sur redujo significativamente el área de la ventana en el 4to piso relativo al resultado en Oporto. Aunque la sombra sigue siendo bastante larga. La ventana Este se ha hecho mucho más pequeño aún. Esto refleja el efecto de la alta ganancia calórica en está locación geográfica. Las ventanas hacia el Oeste quedaron pequeñas al igual que en Oporto para evitar ganancias calóricas. La fachada Norte casi no sufre alteración porque es la única no afectada por la ganancia solar directa.

En términos del uso de energía, la principal diferencia entre la serie de soluciones, como es de esperar, se centra en la energía del espacio refrigerado y asociado a la ventilación mecánica. Si el edificio actual fuese teóricamente emplazado en Phoenix consumiría alrededor de un 20% más de energía en el enfriamiento del espacio y ventilación que la mejor solución encontrada por el SG sin usar al variable de sombreado. Esto sucede porque todas las ventanas en el Este, Sur y Oeste se tornan de reducido tamaño. Mientras que las ventanas sombreadas y la cara Norte se mantienen largas para aumentar el uso de la luz del día. Una vez más, la diferencia en el uso de la luz artificial no es significativa como era de esperar. Probablemente por la extrema locación de los puntos de referencia de iluminación. La peor solución del SG encontrado en Phoenix consume un 55% más de energía en refrigeración.

CHICAGO

El extremo frio de Chicago origino algunos interesantes cambios en relación a las soluciones para Oporto y Phoenix. Para los estudios de la cara Norte, la ventana fue reducida a las mínimas dimensiones permitidas por los límites del lenguaje. Esto se debe a las altas perdidas calóricas en el frio clima de Chicago.

La solución del nivel de fachada permite una extrapolación en términos de organización espacial, sugiriendo que los estudios en la cara Norte deben de evitar este tipo de clima.



Chicago best

Figura 34. Mejor solución para Chicago. También es interesante observar que la mejor solución para Chicago es muy similar a la peor de Oporto. *Ibíd*. 1402.

Hacia el Sur, las aberturas se han hecho bastante grandes porque la entrada de luz día es débil. Sin embargo, las ventanas sombreadas son reducidas a las mínimas dimensiones. La luz natural y la ganancia solar son bloqueadas. Cuando la profundidad de los aleros es usada como variable, el algoritmo reduce en el mínimo permitido el tamaño de los aleros y simultáneamente aumenta el tamaño de las ventanas. (No hay imágenes para este resultado). Se puede concluir que el sombreado en el Sur es indeseable en este clima.

Existe una alta pérdida de calor a través de los vidrios. Para estudios con ambas ventanas Este y Sur, las aberturas propuestas fueron algo más pequeñas. Sin embargo una vez más el Sur fue preferido. Aunque las ventanas Oeste recibieron las mismas dimensiones. En términos de energía, Los altos niveles de consumo pueden ser reducidos usando la estrategia propuesta o la correcta aislación del edificio. Sin embargo usando sólo tamaños de aperturas como variables, el diseño existente consume alrededor de un 14% más de energía calórica que la mejor solución del SG. Las peores soluciones consumen un 23% más. Guardar la energía calórica implica reducir tamaños de aperturas y así reducir la entrada de luz natural.

CONCLUSIONES DEL TRABAJO DE CALDAS Y NORFORD

El SG puede ser lo suficientemente flexible para incorporar límites que permitan al usuario manipular ciertas intenciones de diseño arquitectónico. La cercana coincidencia entre el SG y el diseño de Siza en algunas situaciones fue de particular interés. Por otro lado, los diseños propuestos por el algoritmo sugieren que este sistema generativo puede ser una herramienta útil en la exploración de múltiples caminos durante los procesos de diseño. También es interesante la capacidad del sistema generativo de contar con interacciones entre diferentes elementos del edificio y hacer del diseño cada elemento específico dependiendo del rol integrado en la arquitectura.

La relación entre la solución para la logia y los lucernarios o entre las fachadas Sur y Este en las zonas de estudio son un ejemplo de la capacidad del sistema. La posibilidad de extrapolar desde el resultado del algoritmo en otras decisiones como la geometría del edificio o la organización espacial sugieren nuevas direcciones para futuros trabajos.

El sistema generativo fue capaz de definir sin el lenguaje de condiciones soluciones que tienen un consumo bajo en sus niveles de energía. Este resultado puede ser analizado desde dos perspectivas. Primero, reduciendo los consumos de energía y agregando sostenibilidad al edificio, una cuestión actual que concierne la disciplina de la arquitectura; En segundo lugar indicando otros problemas que podrían indicar el nivel de adaptación al clima. En situaciones donde los sistemas mecánicos no están instalados para compensar estas deficiencias, los usuarios sufrirán eventualmente el des-confort dentro del edificio. En este caso, la escuela de arquitectura de Oporto tiene un sistema de calor pero no de aire acondicionado, por lo que el sobrecalentamiento puede ser un inquietante asunto para los estudiantes. El alto uso de la luz artificial también es un indicador del pobre control luz día en el espacio como se pudo ver.

Finalmente, el rango de soluciones que ofrece el sistema generativo para diferentes locaciones geográficas permite adaptarse al clima donde es ubicado, incluso usando el mismo lenguaje geométrico. El último objetivo es el desarrollo de este software y su inclusión como un sistema generativo operando en las fases más tempranas del proceso de diseño. Las soluciones no

deben ser interpretadas como un resultado definitivo o una respuesta óptima, pero si como un diagnóstico de potenciales problemas y una sugerencia para futuras experiencias arquitectónicas.

El trabajo de Luisa Caldas y Norford es muy interesante, en el sentido que utilizan los AGs a través del programa GenOpt y como *fitness* el programa DOE-2.1E. En su método conectaron dos programas uno para producir el AG y otro para hacer las evaluaciones. Además tocaron temas como la sustentabilidad y el ahorro energético que cada vez más concierne a los arquitectos de hoy en día. El trabajo de Caldas no acaba aquí. Existen otros trabajos muy similares en la misma línea como son:

CALDAS, Luisa. ROCHA, Joao *A Generative Design System Applied to Siza's School of Architecture at Oporto*[43]. En la conferencia CAADRIA. En este artículo aplican la misma estrategia haciendo hincapié en la luz natural. Luego el 2002 Caldas y Norford publican *A design Optimization tool based on a genetic algorithm*[44]. Donde aplican la misma técnica a una planta de un edificio de oficinas. Su trabajo estuvo centrado básicamente en el consumo de energía de los edificios y realizó una serie de publicación, donde más – menos muestra lo mismo.

Sus trabajos ha sido fuente de inspiración para desarrollar el 4to proyecto propuesto dentro de esta tesis.

6.1.4. *Genr8*, Martin Hemberg, Una-may O'reilly, Peter Nordin (2001)

Referencia: *Genr8* (*Generate*) es un *plug-in* de Maya diseñado por Una-may O'reilly y programado por Martin Hemeberg. En el año 2000, para el *Emergent Design Group*, del MIT. http://projects.csail.mit.edu/emergentDesign/*Genr8*/, [16/08/10].

Este trabajo ha sido una de las principales motivaciones que ha tenido el autor para estudiar los algoritmos genéticos. La aplicación tiene un alto grado de similitud entre biología y matemáticas y de cierta manera es el paradigma que presenta John Frazer en su libro; La construcción de un medio ambiente digital, en el cual se simulen las diferentes fuerzas que existen en la realidad y que estas modifiquen la forma a lo largo de generaciones.

Está aplicación, hoy en desusó, corre sobre las primeras versiones de Maya y básicamente lo que hace es imitar el crecimiento de una "semilla" dentro de un medio ambiente digital. Está semilla está sujeta a fuerzas y mutaciones durante su crecimiento. Tiene un alto coeficiente de aleatoriedad por lo que es muy difícil controlar la forma. De hecho este ha sido el gran problema con el que se han enfrentado los arquitectos a la hora de usarlo.

43 CALDAS, Luisa. ROCHA, João. A Generative Design System Applied to Siza's School of Architecture at Oporto. CAADRIA2001, University of Sydney, 2001, pp. 253-264.
44 CALDAS, Luisa. NORFORD, Leslie K. A design Optimization tool based on a genetic algorithm, Massachusetts Institute of Technology, Room 5-418, 77 Massachusetts Avenue, Cambridge, MA 02139, USA

Genr8 es definido por sus diseñadores como una poderosa herramienta de generación de superficies. Desarrollado vía API[45] para Alias| Wavefront's Maya[46]. Combina el sistema de crecimiento L-System[47], el cual es visto como una gramática evolutiva y un ambiente abstracto. *Genr8* está pensado como una herramienta interactiva de trabajo que utiliza las leyes de la evolución como generador de superficies dentro del ambiente de Maya. Ha sido diseñado para que cada semilla sea sensible al ambiente preestablecido en la programación y los usuarios puedan intervenir las formas libremente durante las generaciones. Modificando las semillas (*individuos*) y cambiando los parámetros del ambiente.

Tradicionalmente los algoritmos genéticos han sido usados para resolver problemas de optimización. Sin embargo algunos investigadores han utilizado esta técnica para propósitos creativos, generando formas de vida artificial y modificando los procesos de diseño. El grupo de diseño emergente del MIT[48] unió arquitectos y científicos computacionales con el objetivo de buscar la mejor manera de abarcar un proceso de diseño. En los primeros niveles de investigación se persiguió la diferencia entre una serie de campos como la morfología, el material, la estructura, y la integración no lineal entre la experiencia de la arquitectura y su respuesta al medio ambiente. Los arquitectos plantearon herramientas computacionales que pudiesen simular su creatividad y contribuir en el resultado final.

Los investigadores encontraron en los sistemas L la opción de adaptar el modelo de crecimiento que buscaban. Sin embargo tuvieron que desarrollar L-System para que funcionara en espacios de 3D y tuvieron que agregar elementos medioambientales que influenciaran e interactuaran con estos crecimientos. De esta manera para buscar en un universo tan grande de superficies *Genr8* fue diseñado basado en la computación evolutiva.

El método fue utilizar algoritmos genéticos que aportaban dos beneficios principales. El primero, es que su búsqueda de tipo paralelo da múltiples soluciones a un problema de diseño y el segundo, que también se acomoda a la búsqueda en un gran universo de superficies. Permitiendo coleccionar los modelos adaptados de forma selectiva y exhaustiva.

---

45 Aplication Programming Interface, es el conjunto de funciones y procedimientos (o métodos si se refiere a programación orientada a objetos) que ofrece cierta librería para ser utilizado por otro software como una capa de abstracción.
http://es.wikipedia.org/wiki/Application_Programming_Interface 12 de febrero 2008.

46 Maya es un programa informático dedicado al desarrollo de gráficos en 3d, efectos especiales y animación. Surgió a partir de la evolución de Power Animator y de la fusión de Alias y Wavefront, dos empresas canadienses dedicadas a los gráficos generados por ordenador. Más tarde Silicon Graphics (ahora SGI), el gigante informático, absorbió a Alias-Wavefront, que finalmente ha sido absorbida por Autodesk.
Maya se caracteriza por su potencia y las posibilidades de expansión y personalización de su interfaz y herramientas'. MEL (Maya Embedded Language) es el código que forma el núcleo de Maya, y gracias al cual se pueden crear scripts y personalizar el paquete. Maya posee numerosas herramientas para modelado, animación, render, simulación de ropa y cabello, dinámicas (simulación de fluidos), etc.
http://es.wikipedia.org/wiki/Maya_(programa) 12 de febrero 2008.

47 An L-system or Lindenmayer system is a formal grammar (a set of rules and symbols) most famously used to model the growth processes of plant development, but also able to model the morphology of a variety of organisms. L-systems can also be used to generate self-similar fractals such as iterated function systems. L-systems were introduced and developed in 1968 by the Hungarian theoretical biologist and botanist from the University of Utrecht, Aristid Lindenmayer (1925–1989).
http://en.wikipedia.org/wiki/L-system 12 de febrero 2008.

48 Emergent Design Group, http://web.mit.edu/arch/edg/ [15/08/10]

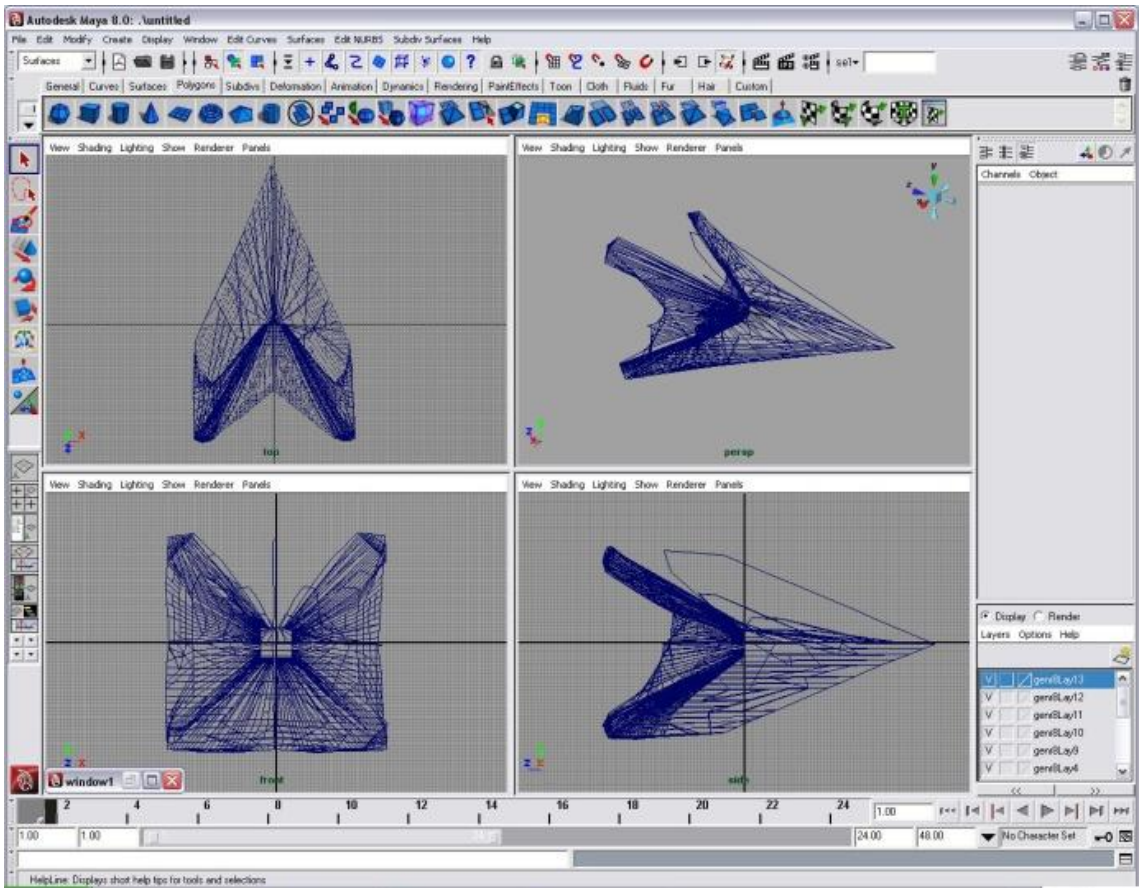


Figura 35. Forma generada con *Genr8***.** Captura de pantalla del autor. [07/02/08].

Genr8 ha sido diseñado para reaccionar bajo diferentes fuerzas, campos de atracción y repulsión de la forma. Además es capaz de otorgarle al usuario control durante la adaptación evolutiva. El software no tiene criterio estructural o material y la mayoría de las veces los resultados están muy cerca del caos.

Con el objetivo de entender el funcionamiento de esta aplicación se explicaran las nociones básicas del programa y su capacidad de explorar y adaptar superficies generadas por medio de AGs. Además se explicará por qué y cómo se usa la Gramática Evolutiva (GE) desarrollada por este grupo de investigadores.

CREACIÓN DE SUPERFICIES

Genr8 simula el crecimiento orgánico a través de superficies digitales. El proceso de crecimiento es reactivo, en el sentido que reacciona a un ambiente físico simulado. El crecimiento de la superficie es generado por el sistema HELMS, una versión extendida de los sistemas L, capaz de producir resultados más complejos. HELMS es definido como una gramática, consistente en una semilla o axioma y un grupo de reglas de proliferación las cuales son aplicadas a cada paso sucesivo del crecimiento. HELMS está basado en una serie de símbolos que orientan el crecimiento de la forma. La Interpretación grafica está basada en la forma de la semilla y es generada en el espacio 3D de acuerdo a las direcciones específicas de las reglas de crecimientos.

Comando Turtle	Significado
Ai, Bj, Ck,…	*Mueve hacia adelante y dibuja una línea*
+, -	*Gira a la izquierda / derecha (valor de ángulo)*
&, ^	*Hacia arriba /abajo (valor)*
*/, *	*Cambia la dirección del segmento. Rueda hacia la izquierda / derecha (valor)*
~	*Empujar grupo/ saltar grupo*

Figura 36. La dirección del crecimiento es basado en un grupo de instrucciones descrito en la tabla. Todos los símbolos tienen una magnitud los cuales son representados mediante valores. Estos símbolos solo controlan la forma original de la semilla y no sus posteriores modificaciones mediante el ambiente simulado.

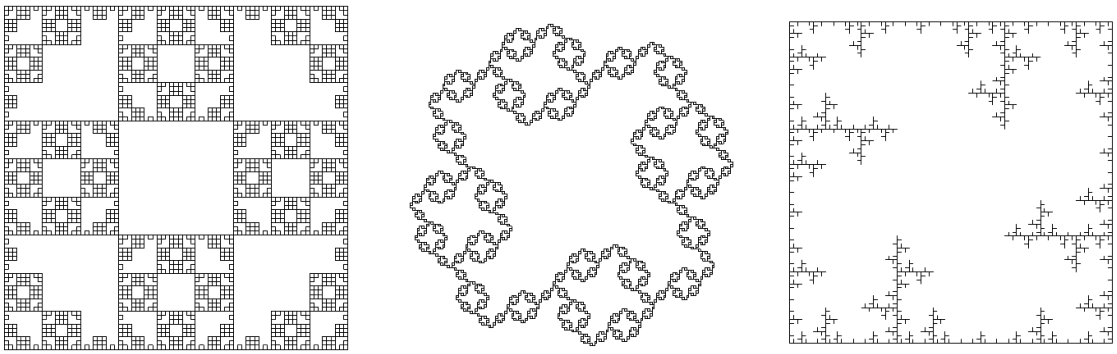


Figura 37. Imágenes de fractales, generados mediante el sistema HELMS. En las imágenes se puede ver como simples reglas pueden producir complejos patrones gráficos.

Dado que el objetivo de los investigadores fue siempre creativo y que las superficies debían ser capaces de responder a estímulos externos, como sucede con las formas en el mundo real, el trabajo original consistió en extender el sistema L en HELMS. Sin embargo, el nuevo sistema incorporó factores físicos que simulan el ambiente donde se generará la semilla. Los vértices de las superficies (semillas) son alterados tras cada iteración por el ambiente. Las superficies son atraídas por *attractors* y repelidas por *repellors* o deformadas por límites simulados dentro del ambiente físico. Además, pueden mutar y una serie de factores de aleatoriedad y deformación pueden ser incorporados en la "genética" de cada semilla. Por lo tanto, las formas no pueden ser controladas, sino que solo dirigidas de cierta manera.

A continuación se exponen una serie de experimentos y clasificaciones de formas hechos con *Genr8*. Dado que el programa es muy difícil de controlar por su alto grado de aleatoriedad, es muy interesante hacer una clasificación de cada una de las formas que genera.

Las primeras dos imágenes muestran un breve experimento donde se generó una superficie que debía crecer entre 4 *atractores*, uno en cada extremo, y un *repellor* en el centro de la superficie.

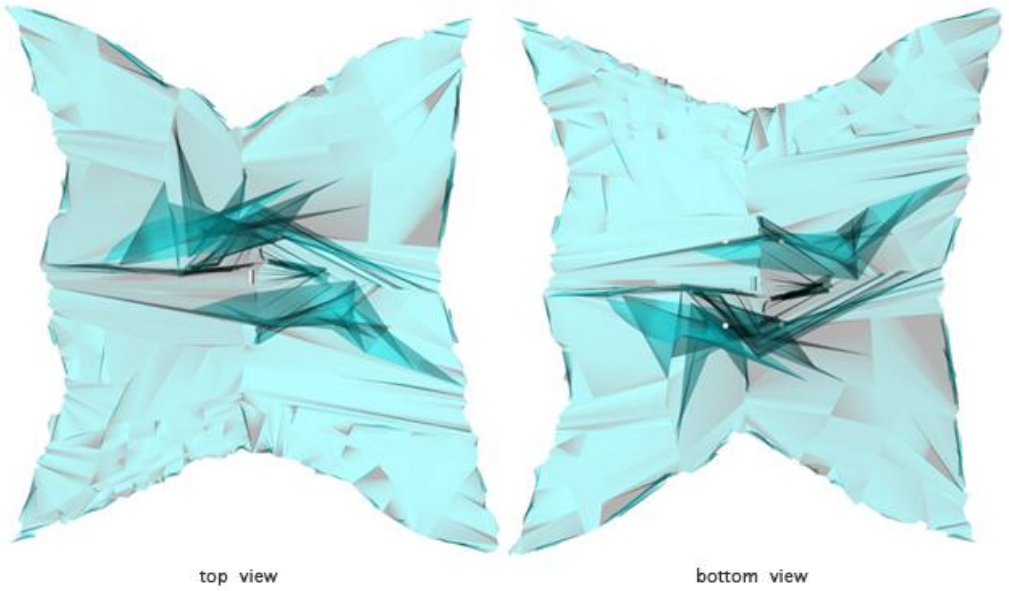


Figura 38. Vista superior e inferior de la superficie. [04/04/08]

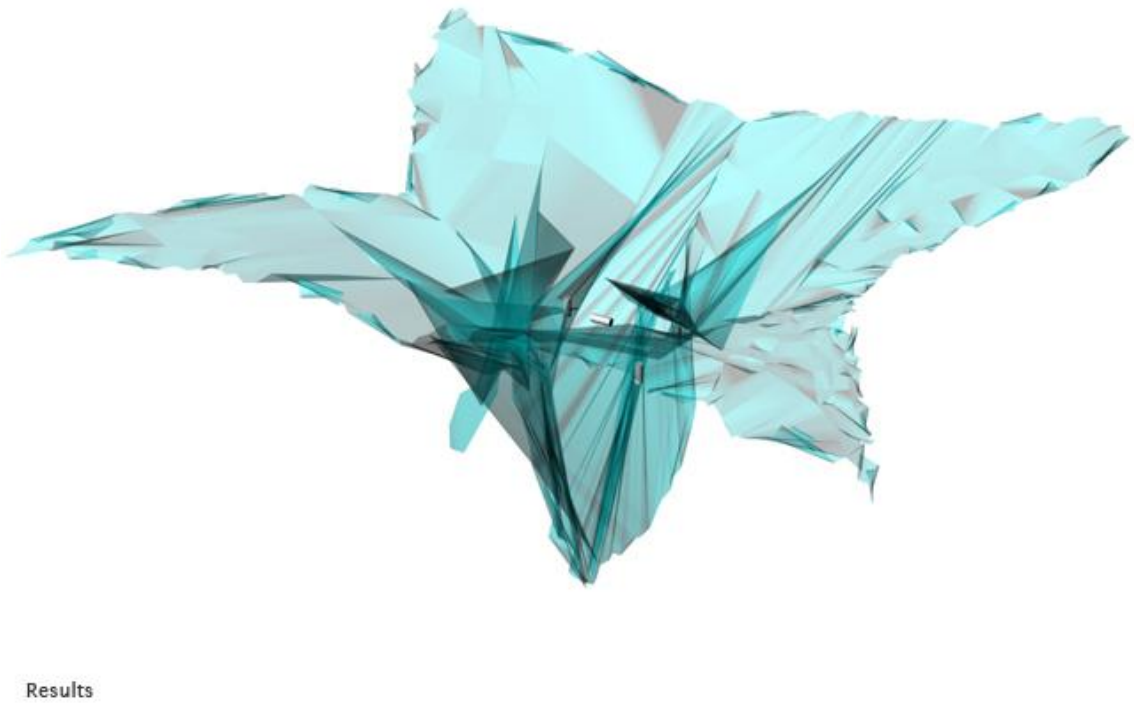


Figura 39. Vista en perspectiva de la superficie generada. En el centro se pueden ver los *repellors* que "empujan" la superficie hacia abajo. Los *attractors* no aparecen en la imagen pero claramente se ve como los bordes de la superficie son atraídos. La forma sugiere una flor. [04/04/08].

Los *attractors* "atraen" a la superficie mientras que el *repellor* las repele. Estos deformadores son creados dentro del ambiente de Maya como pequeños cilindros y luego son definidos dentro de *Genr8*. El programa basa su exploración formal en la computación evolutiva mediante algoritmos genéticos y en la aleatoriedad la cual es creada mediante números pseudo-aleatorios que dependen de los milisegundos del ordenador. Esto significa que una semilla creada mediante las reglas (ver figura 36) de HELMS y evolucionada dentro de *Genr8* nunca tendrá la misma forma si es ejecutada dos veces o más.

Debido a esto *Genr8* nunca ha podido consolidarse como una herramienta directa para arquitectos y diseñadores. Además no toma consideraciones estructurales ni de material. *Genr8* fue utilizado durante 3 años por la *AASchool Architecture* de Londres para la experimentación de generación de superficies, pero fue desechado, porque sólo se encontraron resultados de manera inductiva a los que el *plugin* producía[49]. Solo sirvió para inspirar formas.

La siguiente serie de imágenes corresponde a una clasificación de superficies generadas con *Genr8*. Este trabajo fue hecho durante el Magister "*Arquitecturas Genéticas*" en la Universidad Internacional de Catalunya.

49 Conversación entre el autor y Mike Weinstock durante abril del 2006, en la conferencia SIMAE. Universidad Internacional de Catalunya.

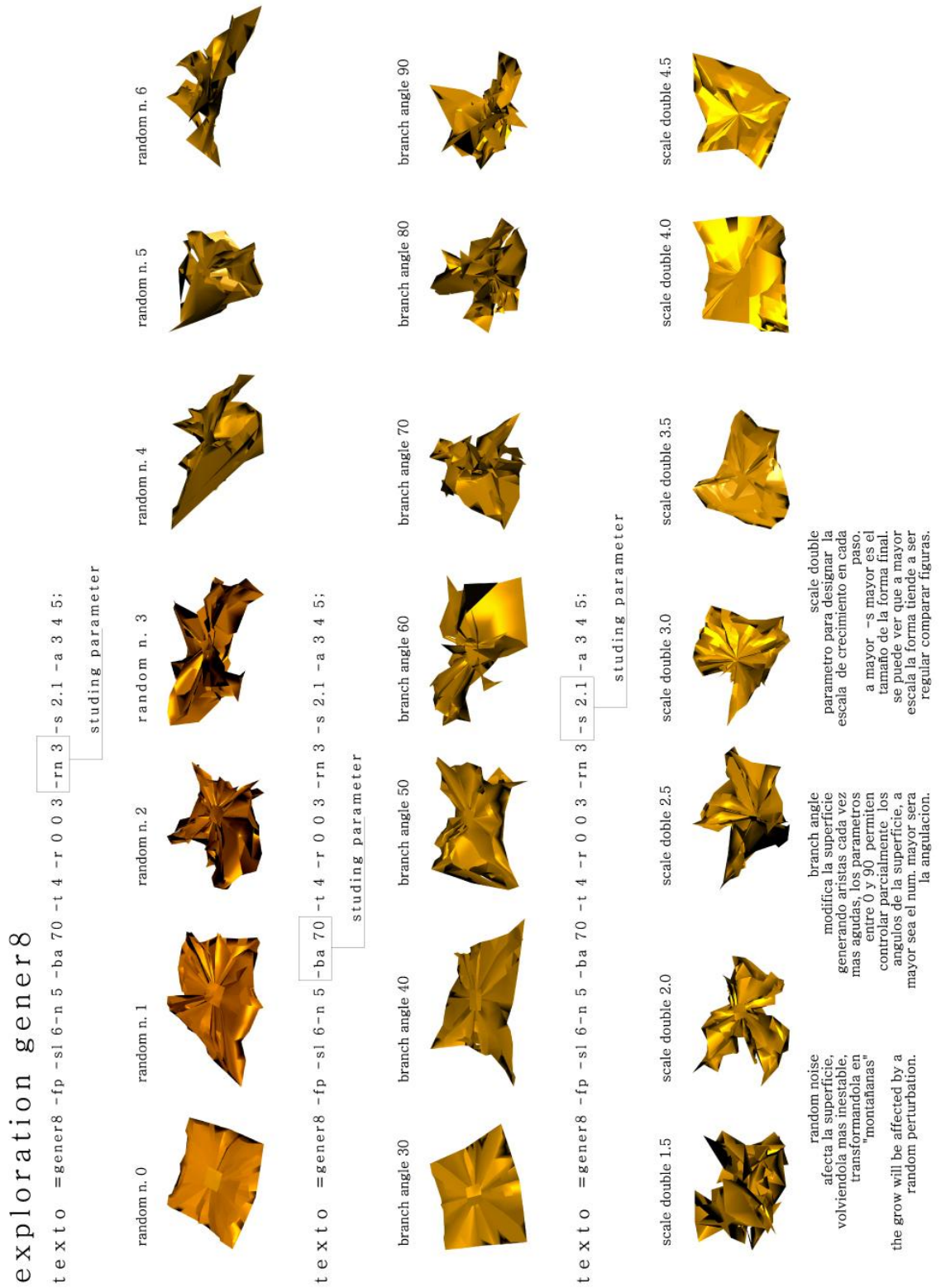


Figura 40. Los parámetros *Random Noise* afectan a la superficie volviéndola más montañosa. El parámetro *Branch Angle* modifica las superficies haciendo sus aristas cada vez más agudas. Mientras más grande es este valor mayor es la angulación conseguido en los bordes de la superficie. Otro parámetro estudiado es *Scale Double*, que permite designar el salto de escala entre cada iteración del algoritmo. Mientras mayor sea este valor más grande será la superficie final y su geometría tiende a volverse más regular.

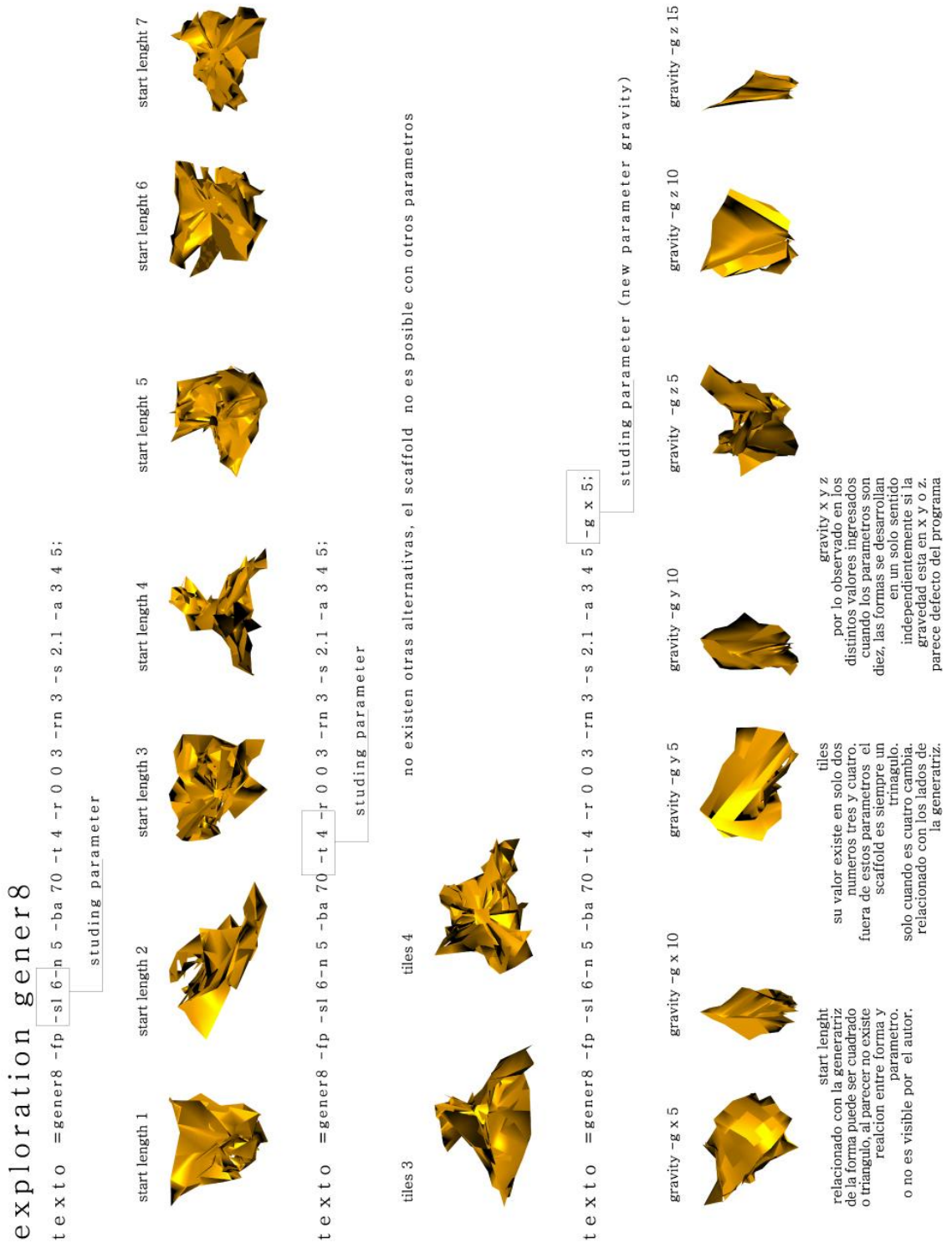


Figura 41. No existe una relación directa entre *Start Length* y los resultados finales de la forma. La variable *Tiles* tiene sólo dos opciones 3 o 4. Fuera de estos valores la superficie siempre es triangular. La variable *Gravity* admite valores para X, Y, Z. Sin embargo solo fueron apreciables los resultados cuando los valores fueron 10, y la superficie siempre creció en el eje de la Z.

Figura 42. Los *repellors* actúan sólo a una corta distancia de la superficie y siempre tienen la misma intensidad de deformación sobre la superficie.

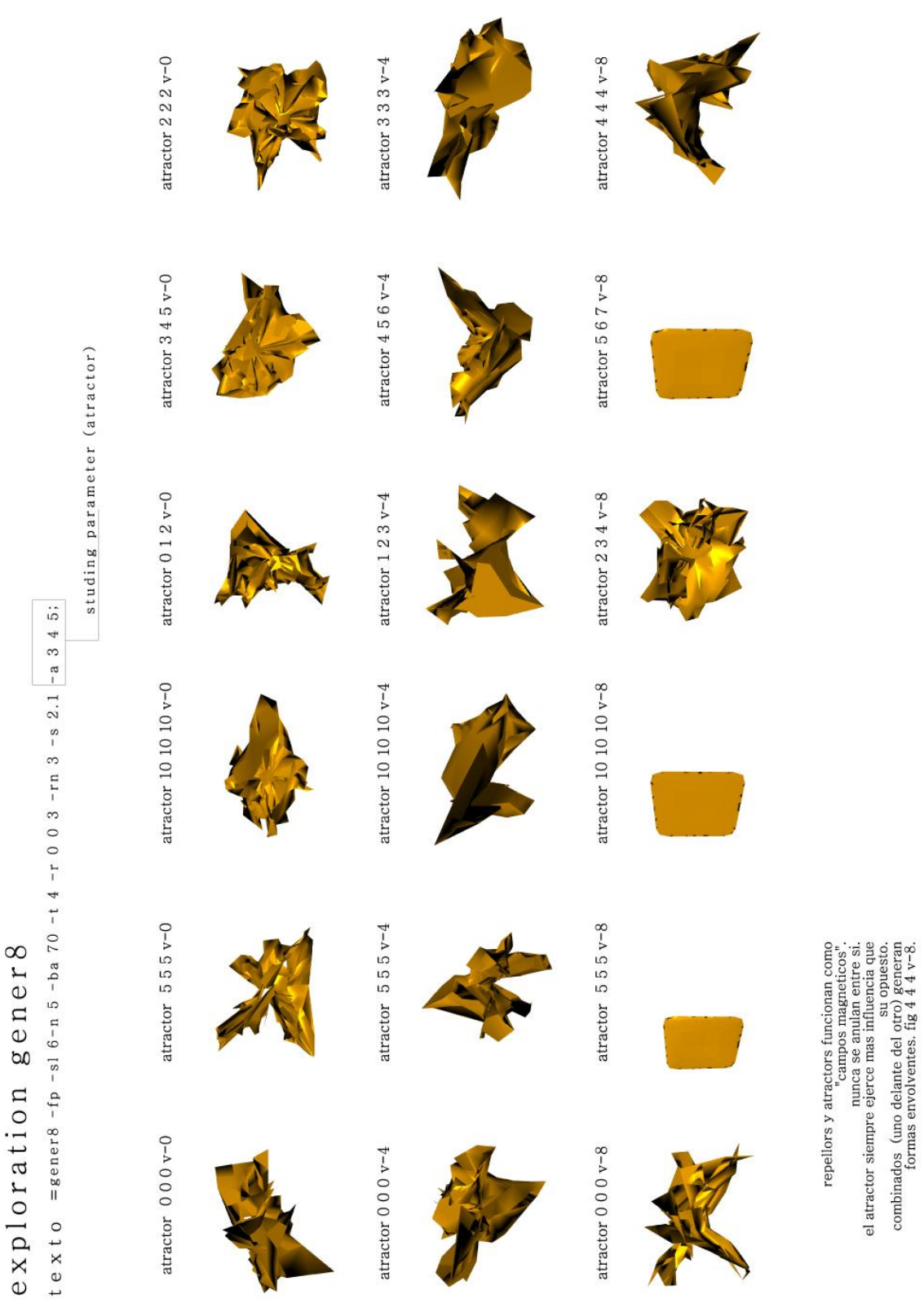


Figura 43. *Repellors* y *Attractors* modifican la geometría y se logra controlar la forma disponiendo los modificadores para crear una especie de envolvente. Aparece en la notas tomadas que los *attractors* tienen más influencia que los repellors.

La manera de hacer arquitectura con *Genr8*, es tangencial. En el sentido de que no se puede controlar una forma completamente, sino que es necesario hacer un paso extra para poder aproximarlo a la arquitectura de una manera deductiva o inspiradora. Si bien el resultado es interesante formalmente, es necesario volver a pensar en estructura, espacialidad sustentabilidad. Es por esto que como herramienta proyectual de diseño *Genr8* no tuvo el éxito que se esperaba. En un principio ciertas universidades comenzaron a incluirlo dentro de su práctica de taller y diseño, pero pronto fueron abandonados porque las geometrías generadas eran de difícil manipulación para el arquitecto.

COMPUTACIÓN EVOLUTIVA

Genr8 plantea el crecimiento de superficies basado en el sistema HELMS dentro de un enorme universo de soluciones. El intento de desarrollar a mano este proceso sería una tarea casi imposible de realizar. El componente evolutivo dentro de *Genr8* sirve básicamente para responder a tres objetivos relacionados con la necesidad de automatización. El primero consiste en generar automáticamente la gramática tipo HELMS. El segundo, explorar algunas de las superficies del universo eligiendo resultados diferentes y creativos. El tercero es que *Genr8* puede ser usado para definir una superficie que se tenga en mente. Permitir que la búsqueda de la forma sea influenciada por el sistema y acercarse al diseño a través de una manera inductiva y creativa. Estas razones, explican los investigadores, obligan al uso de la computación evolutiva.

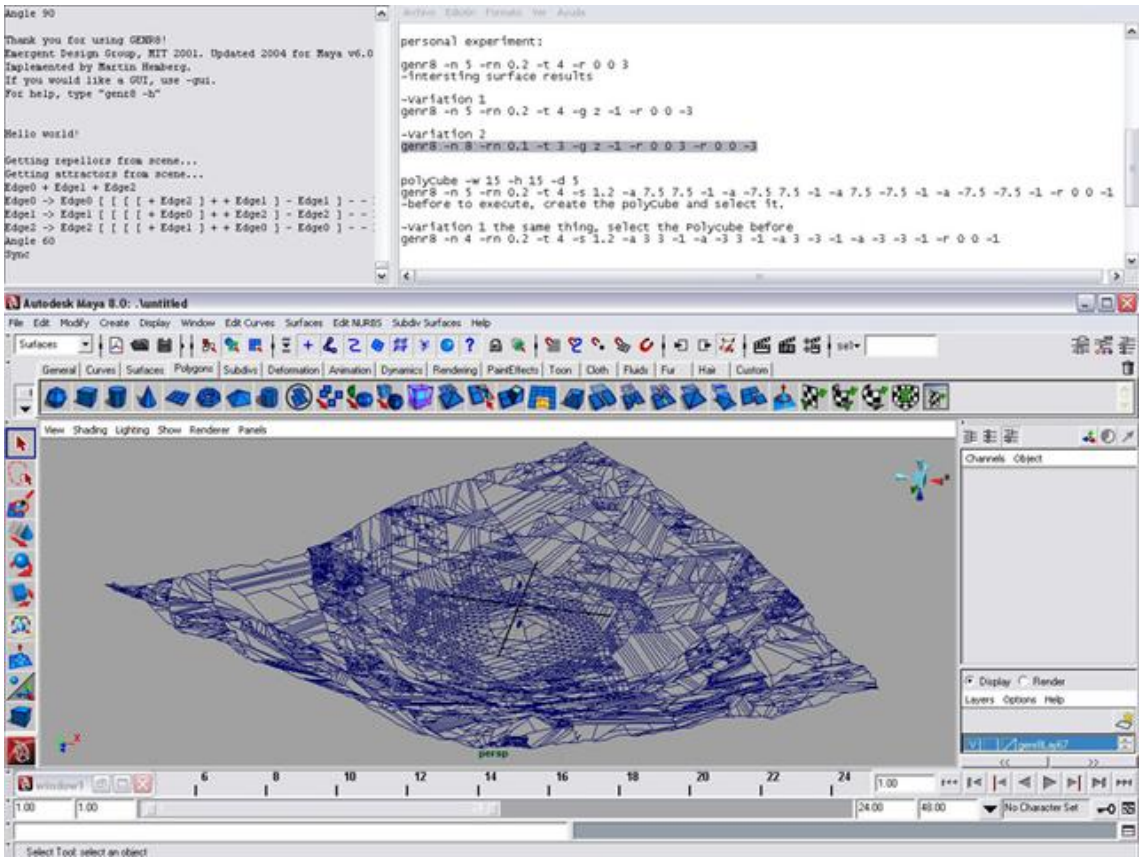


Figura 43. Captura de Pantalla de *Genr8* y un trozo de código en HELMS. [12/08/08]. La imagen anterior corresponde al trabajo en *Genr8*, en la izquierda superior de la imagen se ve el

código tipo L System (HELMS) que produce la semilla inicial, a la derecha un editor de texto es usado para escribir las sintaxis de las generaciones de formas, en este caso se ha usado un *repellor* que actúa como un campo magnético de la superficie repeliéndola. Toda la información de la superficie se encuentra en esa simple línea de código seleccionada.

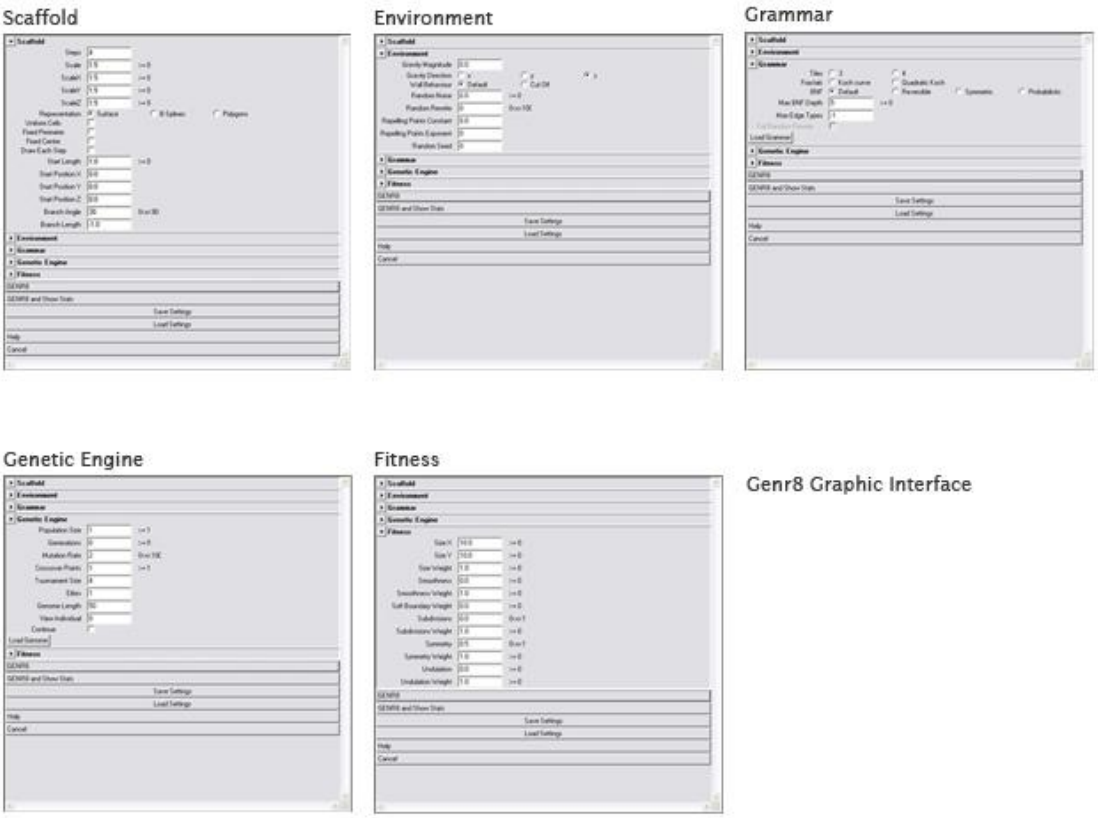


Figura 44. La secuencia de imágenes corresponde a las diferentes configuraciones para el ambiente de *Genr8*. La primera ventana define el tipo de semilla y sus características de mutación. La ventana de *Environment* define las características del ambiente y la posición de los *attractors* y *repellors*. En la ventana *Grammar* se define el tipo de crecimiento, geometría y otras características de la semilla. *Genetic Engine* y *Fitness* definen las características del AG. Tamaño de la *población*, porcentaje de aleatoriedad, cruce, mutación, cantidad de iteraciones, etc.

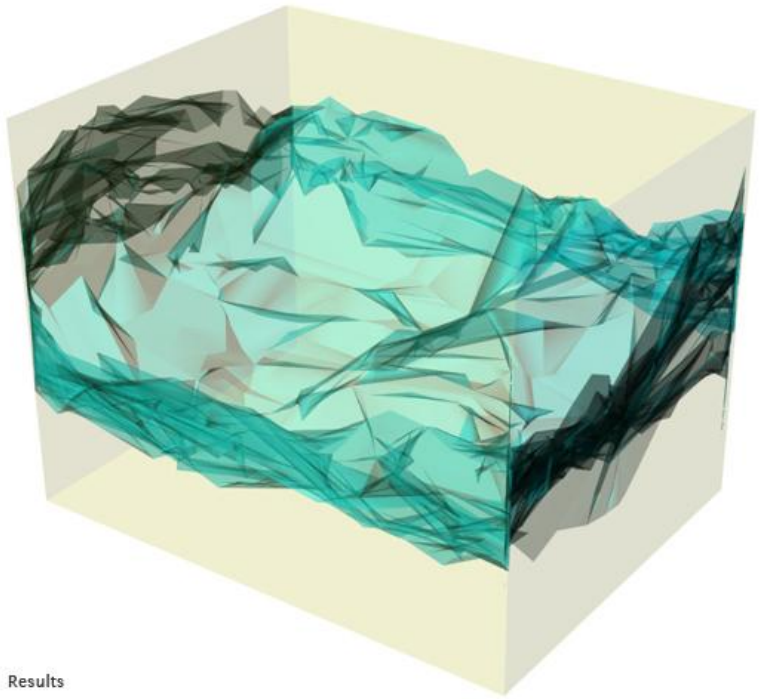


Figura 45. Ejemplo de una superficie generada dentro de un poliedro regular el cual es usado como límite de crecimiento. [02/05/08]

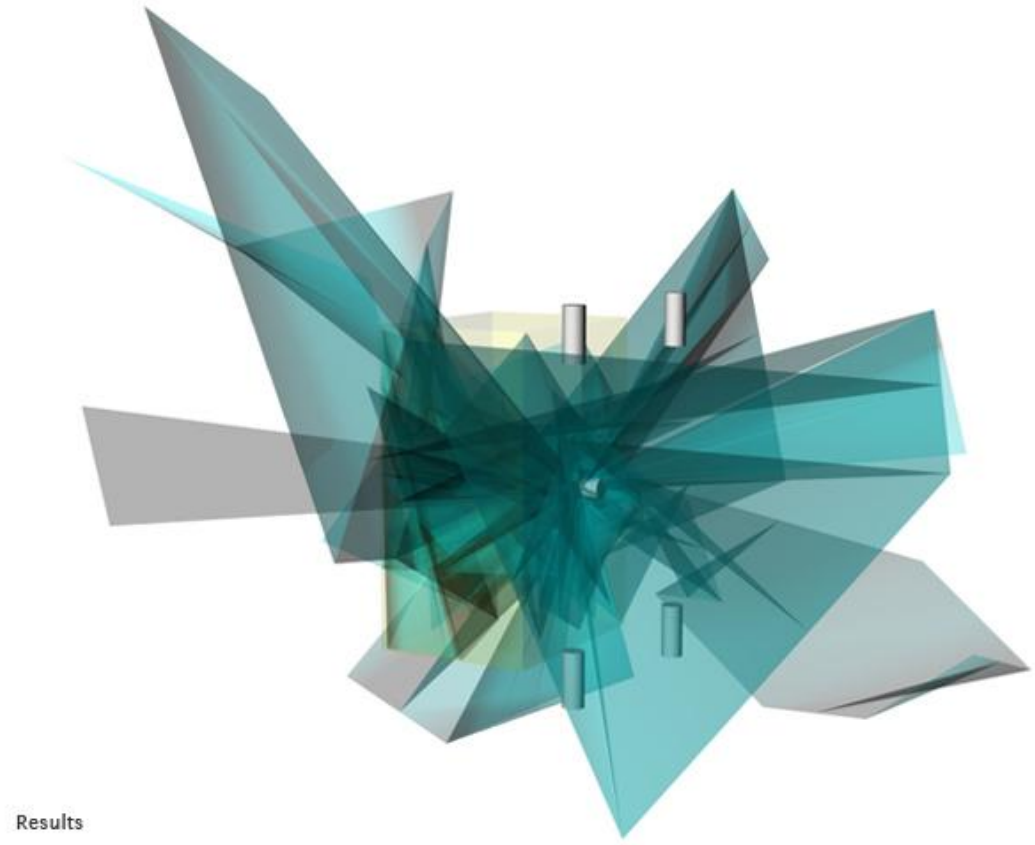


Figura 46. Generación de una superficie dentro de un poliedro regular. Sin embargo el volumen no fue capaz de contener la superficie la cual además estaba bajo la presión de 4 *atracttors* y un *repellor*. [02/05/08]

6.1.5. "ArchiKluge", Pablo Miranda Carranza (2005)

Referencia: Proyecto *Archikluge* de Pablo Miranda Carranza, año 2005. Página web del proyecto: http://www.armyofclerks.net/ArchiKluge/index.htm [15/08/10]. Página de Pablo Miranda: http://www.armyofclerks.net/ [15/08/10].

El proyecto *Archikluge*, es el primero de una serie de experimentos escritos en Java por Pablo Miranda Carranza que explora la "creatividad artificial" y plantea un acercamiento al diseño automático y generativo en la arquitectura. *Archikluge* es un simple algoritmo genético que evoluciona diagramas arquitectónicos. Además, explora las cualidades del diseño generado por una "máquina digital". Carece de toda intención, supuestos y prejuicios. A menudo muestra una forma muy peculiar de superar los obstáculos y problemas; Una manera de actuar típica de la naturaleza.

Cada *población* se inicia con un número de *individuos* que se generan aleatoriamente, los cuales pueden reproducirse o no, dependiendo de su desempeño en la función de aptitud. El *individuo* consiste en un genotipo de cierta información codificada en la que opera el algoritmo genético (Reproducción, mutación, recombinación), luego decodifica la información y la traduce a lo que se conoce como el fenotipo, al cual se evalúa su aptitud.

En el caso del AG de *ArchiKluge* este consiste en un bit de 64 enteros, y el fenotipo de la expresión de este entero es una disposición ordenada dentro de una matriz de 4x4x4. Cada bit expresado influye en el estado de la célula en la matriz. Existen 2^64 o 18446744073709551616 posibles combinatorias de la matriz.

Si en lugar de utilizar un algoritmo genético quisiéramos evaluar todos y cada una de estas posibilidades para averiguar la mejor de todas, teniendo un milisegundo para cada una de ellas tardaríamos cerca de 585 años en analizarlas. *ArchiKluge* implementa un AG con una selección tipo "ascenso" el cual consiste en seleccionar dos *individuos* que sean seleccionados al azar y se sustituye al menos dotado en su descendencia común. La mutación también se realiza de forma aleatoria en los *individuos*. La función de aptitud consiste en la adición para cada célula agregada "pequeños caminos", una medida que usualmente es usada en el análisis de redes y en el análisis de espacio. Esto significa que por cada célula en la matriz de 4x4x4 se calcula el camino más corto.

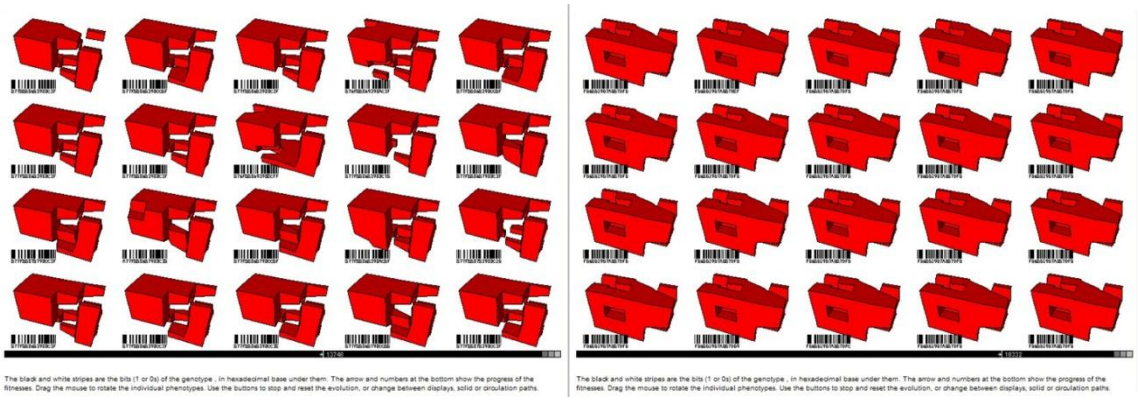


Figura 47. Las tiras en blanco y negro (código de barras), corresponden a cada uno de los cromosomas de la *población*, con base hexadecimal. La barra negra representa la función de aptitud media de la *población*. En la imagen de la izquierda la aptitud media es de 13748, mientras en la derecha la media es de 18332. Se puede apreciar que los *individuos* de la

izquierda presentan geometrías más diversas entre sí. Mientras que la *población* de la derecha al tener un *fitness* más elevado casi no hay variación geométrica entre los *individuos*. Esto es porque el AG se encuentra buscando en una zona determinada del espacio de búsqueda y los individuos no se diferencian significativamente entre sí. Imagen del *applet* de Java tomado de http://www.armyofclerks.net/ArchiKluge/ArchiKluge.html [12/02/08]

6.1.6. Evolutionary Algorithm for Structural Optimization, Voss, Mark S. and Foley, Christopher M. (1999)

Referencia: VOSS, Mark, FOLEY, Christopher, Evolutionary Algorithm for Structural Optimization, Genetic and Evolutionary Computation Conference (GECCO-99), Orlando, Florida, USA, 1999. http://scholar.google.es/scholar?q=Evolutionary+Algorithm+for+Structural+Optimization&hl=es&as_sdt=0&as_vis=1&oi=scholart. [18/08/10].

Está investigación presenta un trabajo basado en la optimización de elementos bajo el concepto de carga estructural. Los conceptos de cruce y de mutación se utilizaron para mantener la diversidad genética. Los autores desarrollaron un método gráfico que permitiese la supervisión del progreso de los componentes de la función de aptitud, lo que permite al usuario interactuar dentro el proceso evolutivo. Las cargas no lineales son un rango en la base de selección que se utilizó para ordenar la búsqueda del óptimo global en un problema con 20 variables de diseño.

El objetivo de este trabajo era probar la efectividad de los algoritmos evolutivos en el diseño de pilares compuestos por bloques superpuestos. Los investigadores comenzaron diseñando una columna, en la cual dos variables controlaban la anchura y longitud de la sección de la columna. Estás variables se repetían a lo alto del pilar, con la idea de modificar las secciones del mismo dependiendo de la carga vertical y dos cargas horizontales en el extremo superior.

En total existen 20 variables que controlaban la sección del pilar. La función de aptitud es medida por las cargas verticales y horizontales de la columna. Entonces, dependiendo de las diferentes secciones a lo largo de la columna se puede evaluar qué tipo de geometría soporta mejor las cargas y cuáles no. El volumen total de la estructura es el valor a ser optimizado. La optimización consiste en encontrar la columna con menor volumen que soporte la mayor cantidad de carga vertical y horizontal en coordenadas X e Y. Por ejemplo, para contrarrestar la carga horizontal en el eje Y, es necesario aumentar la sección de la columna en este eje. Pero esto con lleva dos problemas. El primero es que aumenta el volumen, que es el valor que se intenta disminuir. Segundo esto fuerza a la columna a doblarse (pandearse) en el eje de las X. Este tipo de problema tiene que lidiar con diferentes parámetros que están conectados entre sí y que se oponen mutuamente.

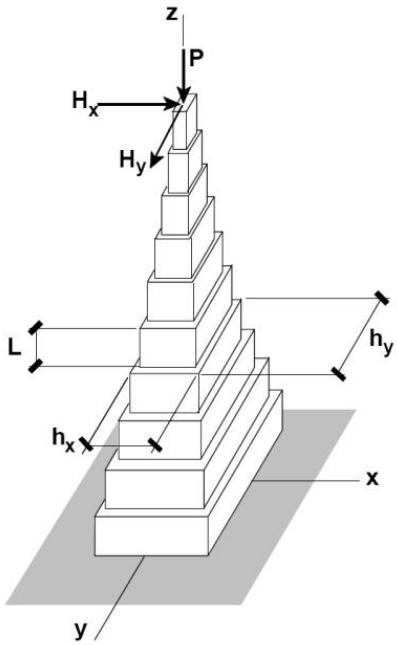


Figura 48. Imagen de la columna segmentada. En la parte superior se indican las cargas estructurales a las cuales es sometida la estructura. A media altura aparecen las variables h_x y h_y, encargadas de controlar las secciones de la columna. Para este experimento la altura de los bloques es considerada como constante. VOSS, Mark, FOLEY, Christopher, Evolutionary Algorithm for Structural Optimization, Genetic and Evolutionary Computation Conference (GECCO-99), Orlando, Florida, USA, 1999. Pág. 3.

El AG fue iterado 50 veces. Dada una *población* inicial de 80 *individuos*, por cada uno se generaron de manera aleatoria 10 diferentes secciones con variables de ancho y largo, las cuales podían tomar cada una 200 valores. El algoritmo en 50 generaciones pudo encontrar el 3% mejor de soluciones para los pilares. Esto indica que el AG fue capaz de escalar y organizar las diferentes secciones dentro de un espacio de búsqueda grande y complejo. El método gráfico desarrollado por los investigadores permitió interactuar con las columnas durante la ejecución del AG, permitiendo al usuario manipular y seleccionar columnas dentro del proceso de búsqueda.

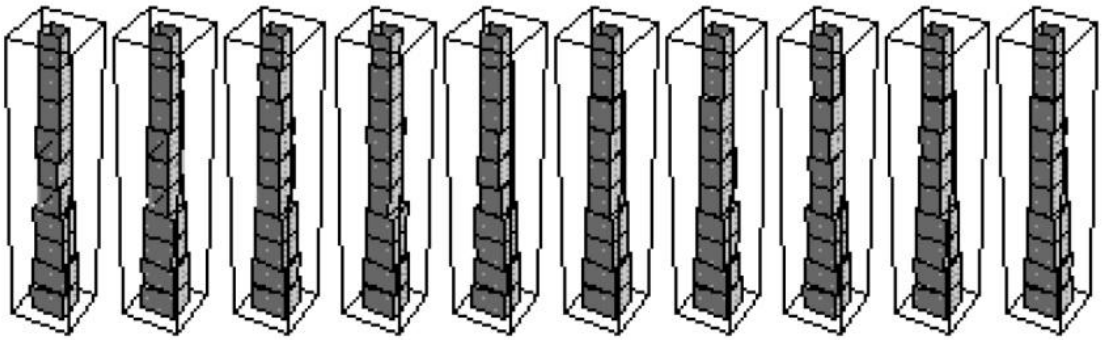


Figura 49. Generación 41 de 50. VOSS, Mark., FOLEY, Christopher., Evolutionary Algorithm for Structural Optimization, Genetic and Evolutionary Computation Conference (GECCO-99), Orlando, Florida, USA, 1999. Pág. 8.

6.1.7. Recent Advances in Evolutionary Structural Optimization, Xie, Y.M., Huang, X., Tang, J.W. and Felicetti, P. (1992)

Referencia: XIE Y.M. y STEVEN, G.P., Shape and layout optimization via an evolutionary procedure, en Proceedings of International Conference on Computational Engineering Science, Hong Kong University of Science and Technology, Hong Kong, 1992, p ág. 471. XIE, Y.M., HUANG, X., TANG, J.W. and FELICETTI P., Recent Advances in Evolutionary Structural Optimization, 1992, http://www.isg.rmit.edu.au/research_ESO.html, [19/08/10].

El método llamado Optimización Estructural Evolutiva (ESO) está basado en el concepto de la graduación de material dentro de la estructura como resultado de una forma optimizada. El procedimiento de optimización consiste en ir retirando el material ineficiente de la estructura, por lo que el material residual, evoluciona hacia un óptimo. El proceso completo de optimización es capaz de resolver: tamaño, forma, optimización estructural estática, dinámica y problemas de transferencia de calor o la combinación de estas[50]. El programa está pensado para la práctica de arquitectos, ingenieros o cualquier persona con conocimientos básicos de análisis de elementos finitos (FEA). Otra ventaja es que puede ser implementado en programas comerciales como ABAQUS, NASTRAN y ANSYS.

El método de ESO consiste en remover el material de la estructura basado en la tensión de von Mises[51] o la deformación de elementos por energía. Para ciertos tipos de materiales solo existen comportamientos a compresión o tracción. Para alcanzar una estructura optima tensada, lo elementos con cargas de compresión son eliminados primero. Luego los elementos con menos carga a compresión son retirados de la estructura en sucesivas generaciones hasta que solo quedan los elementos que trabajan a tracción.

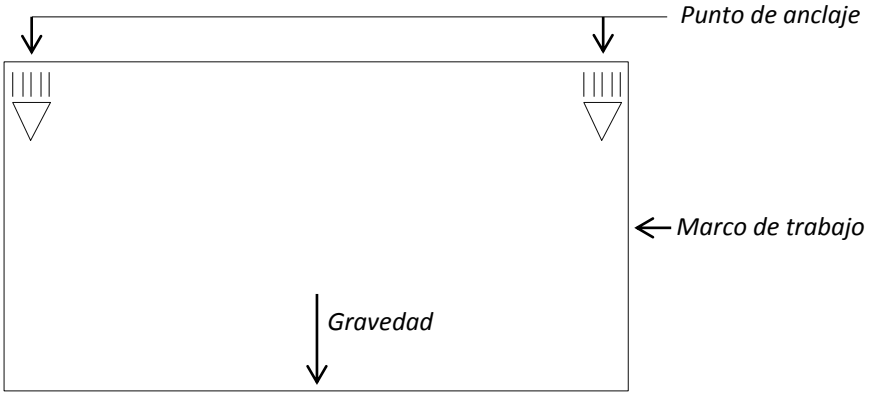


Figura 50. Marco de prueba para la generación de una catenaria. http://www.isg.rmit.edu.au/research_ESO.html, [19/08/10].

50 XIE, M., y STEVEN, G.P., Evolutionary Structural Optimization, Londres, Springer-Verlag, 1997, pág. 188.

51 La tensión de Von Mises es una magnitud física proporcional a la energía de distorsión. En ingeniería estructural se usa en el contexto de las teorías de fallo como indicador de un buen diseño para materiales dúctiles. La energía de von Mises puede calcularse fácilmente a partir de las tensiones principales del tensor tensión en un punto de un sólido deformable.
http://es.wikipedia.org/wiki/Tensión_de_Von_Mises, [19/08/10]

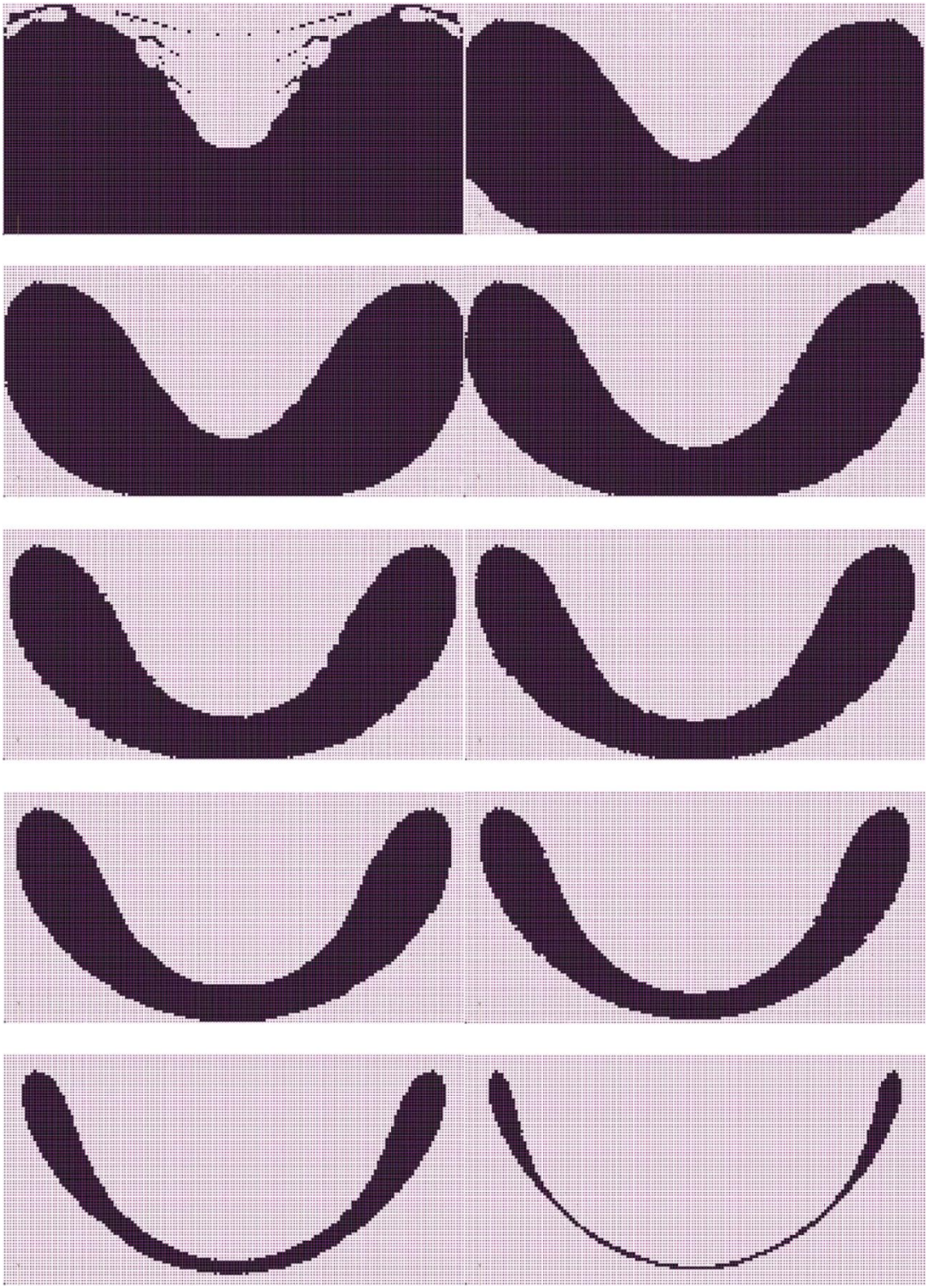


Figura 51. Ejemplo de la secuencia de optimización de la catenaria por el método ESO. En la imagen superior de la izquierda se ve el inicio del proceso y como el material que trabaja a compresión es retirado rápidamente. Luego mediante la iteración del algoritmo, se retira paulatinamente el material ineficiente hasta que sólo queda el que trabaja a tracción. http://www.isg.rmit.edu.au/research_ESO.html, [19/08/10].

Para el diseño de estructuras no lineales, los modelos se analizan por medio de elementos finitos, considerando la no linealidad del material y/o la no linealidad de la geometría. En este caso la eliminación de material de la estructura está basada en dos criterios. Primero, los elementos con bajos valores del cálculo von Mises y la segunda los elementos con baja energía de deformación.

El resultado del método depende de que la modificación estructural (evolución) en cada paso sea pequeña y que la malla de elementos finitos es lo suficientemente densa. Si se elimina demasiado material en un paso, el programa no es capaz de restaurar estos elementos en las sucesivas iteraciones. Es por esto que el programa en 1999 fue revisado y se le incorporó la posibilidad de agregar y eliminar material simultáneamente. Esto trajo consigo poder generar geometrías más complejas.

Los modelos obtenidos por medio del programa pueden ser exportados rápidamente a máquinas de 3D*printer* para su fabricación y estudio físico.

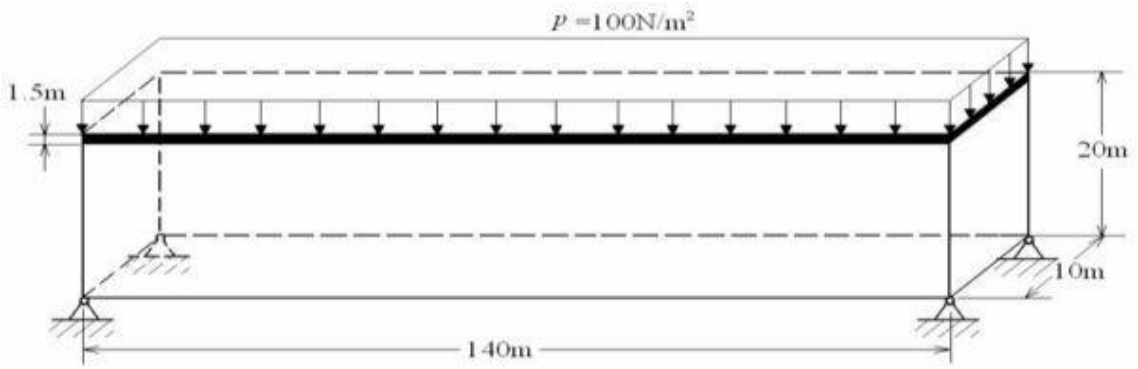
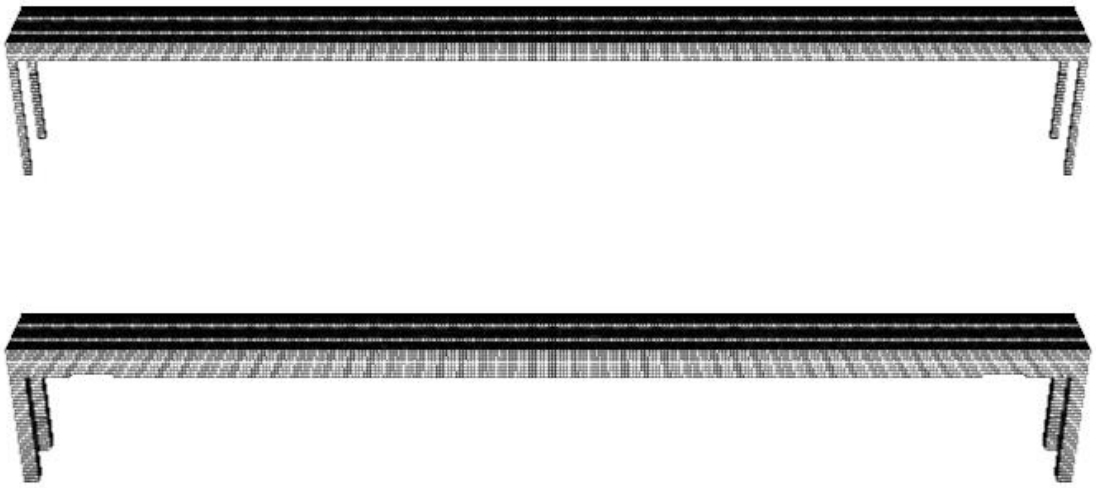
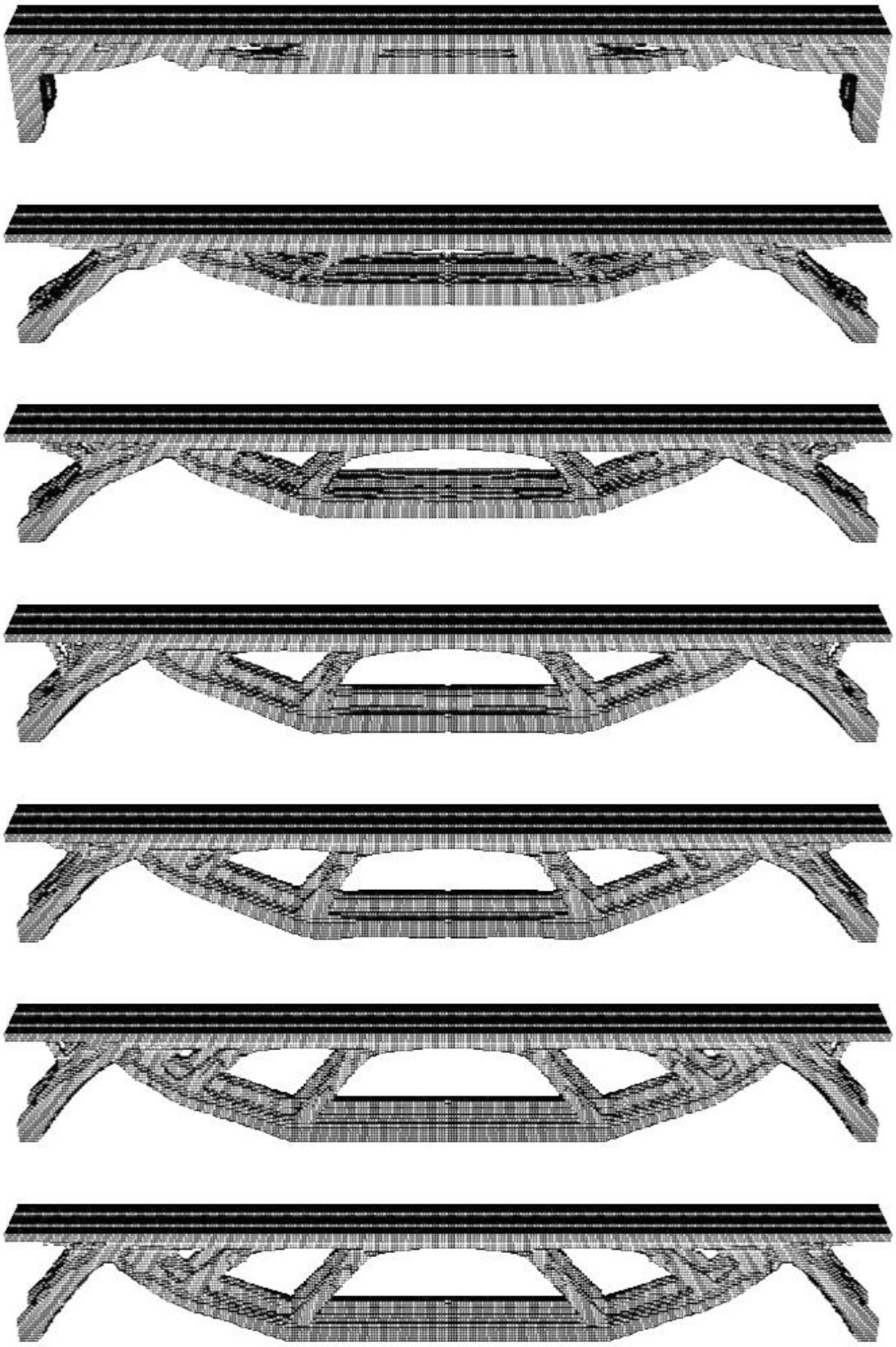


Figura 52. Esquema de cargas y condiciones de contorno para para una viga con 4 puntos de apoyo. http://www.isg.rmit.edu.au/research_ESO.html, [19/08/10].





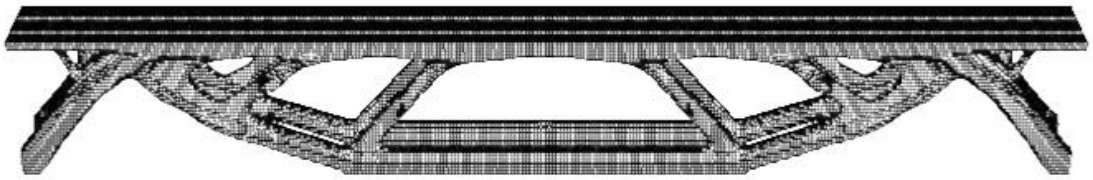


Figura 53. Resultado del proceso de generación. La última imagen de la secuencia corresponde a la mejor solución. http://www.isg.rmit.edu.au/research_ESO.html, [19/08/10].

Figura 54. El nuevo centro de convenciones de Qatar fue diseñado por Arata Isozaki quien utilizó el método ESO para diseñar una estructura con forma de raíz para la fachada como protección al clima del desierto y como símbolo de crecimiento y fortaleza del país. http://cubeme.com/blog/2010/02/08/education-city-convention-centre-qatar-by-yamasaki-architects/. [19/08/10].

6.1.8. Coevolving Species for Shape Nesting, Jeffrey Horn, 2005

Referencia: HORN, Jeffrey. Coevolving Species for Shape Nesting, Accepted for publication in the proceedings of the 2005 IEEE congress on Evolutionary Computation, 2005. Vol. 2, pág. 1800 – 1807.

Este trabajo es muy interesante porque utiliza algoritmos genéticos para encajar piezas dentro de una plancha de corte y así disminuir la perdida de material sobrante. Las piezas pueden ser rotadas y desplazadas en cualquier posición sobre la plancha de corte. El objetivo es doble, por un lado maximizar la cantidad de piezas dentro de la plancha y por otro disminuir la perdida de

material. Este trabajo además tiene múltiples aplicaciones, sobre todo en la industria de fabricación.

El problema general envuelve la colocación de piezas dentro de un contorno limitado y la optimización consiste en colocar la mayor cantidad de piezas posibles. No pueden superponerse las piezas entre ellas y deben estar completamente dentro de los límites del contorno. Los problemas de colocación (*packing problems – nesting problems*) se encuentran presentes en la industria de automóviles en la cual una serie de piezas son cortadas sobre planchas de metal. En la industria del vestir, los patrones de pantalones, camisas, etc. son obtenidos a partir de rollos de tela extendidos sobre la mesa.

En general un problema de colocación puede incluir una amplia variedad de restricciones. Por ejemplo, este problema puede ser encontrado en 1, 2 o 3 dimensiones. El límite del contorno puede ser infinito en una dirección: una bobina de metal o de tela. También puede haber limitaciones en la orientación de las piezas, las cuales podrían necesitar estar alineadas con la forma del contorno, o evitar ciertas áreas. También el tipo de corte influye, si es con punzón, con guillotina o con otras técnicas. También la distancia entre piezas es otra consideración a tomar en cuenta.

Restricciones del problema

El experimento de Horn está diseñado para trabajar en dos dimensiones sobre un contorno plano de tamaño fijo y la misma forma de pieza a encajar. La geometría es trabajada sobre polígonos para simplificar el cálculo de área y la forma del contorno no es convexa. La forma de la pieza no está alineada al eje de coordenadas y para su colocación pueden ser rotadas en cualquier dirección. No existe una relación entre las piezas y los límites del contorno. Tampoco se ha considerado una distancia mínima entre las piezas excepto la superposición entre ellas.

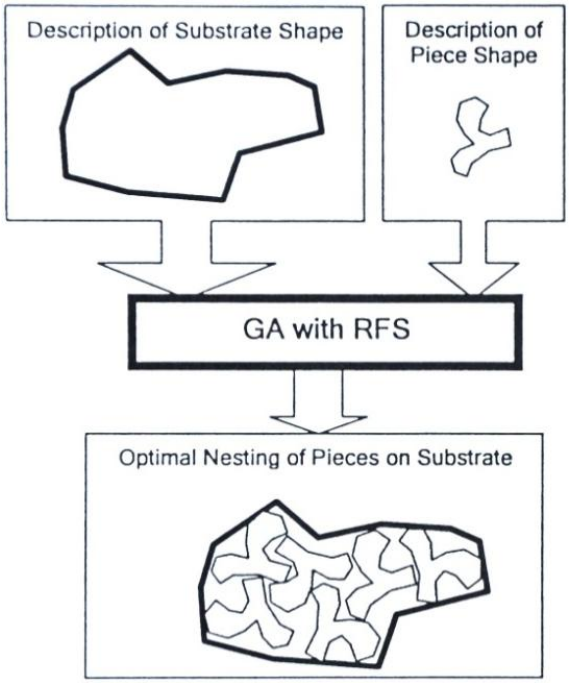


Figura 55: Representación gráfica del problema. HORN, Jeffrey. Coevolving Species for Shape Nesting, 2005. Vol. 2, pág. 1802.

Para afrontar el problema, la colocación de la pieza es hecha especificando su ubicación y orientación (rotación). Se define el centro de cada pieza mediante un punto en coordenadas (x, y) el cual también sirve para identificarlas dentro del plano cartesiano. También se define una rotación de referencia "ϑ" representado en ángulos. De esta manera el cromosoma consiste en tres genes (x, y, ϑ).

Cada *individuo* es evaluado y se le asigna una función de aptitud. Si la pieza se encuentra fuera de los límites del contorno recibe un 0, si se sobrepone con otra pieza recibe un valor superior a 0 y si está libre (dentro del contorno y sin superposición) obtiene el máximo puntaje.

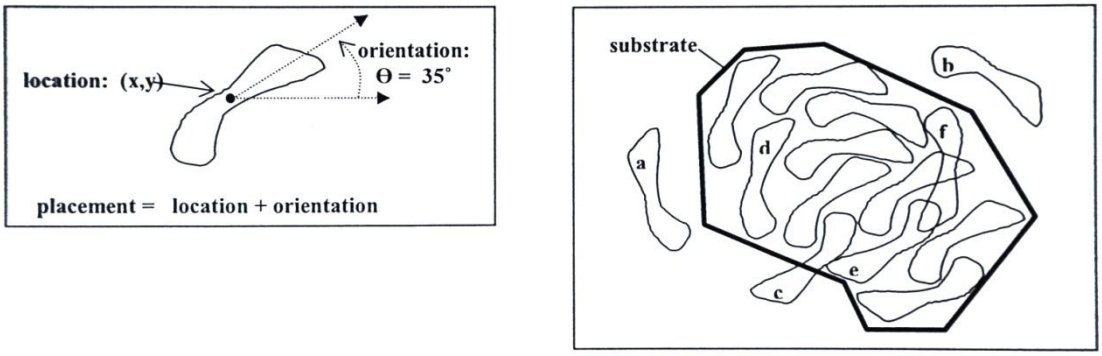


Figura 56. En la imagen de la izquierda aparecen las variables que conforman el cromosoma de cada *individuo*. En la derecha los *individuos* a, b y c han sido evaluados con un *fitness* de 0 por encontrarse fuera del contorno. Los *individuos* d, e y f son evaluados con un *fitness* superior a 0, dependiendo si se superponen con otras geometrías. *Ibíd*. 1803

Una vez que N número de piezas han sido encajadas dentro del contorno, un segundo tipo de evaluación es ejecutado. Este corresponde a la cantidad de piezas dentro de los límites del contorno. Está evaluación es llamada *Shared Fitness* y es la encargada de evaluar si es posible continuar agregando piezas dentro del contorno o no. De esta manera se tienen dos tipos de generación dentro de la evaluación. Por un lado se optimiza la colocación de piezas dentro de los límites del contorno y por otro la cantidad de ellas dentro del mismo.

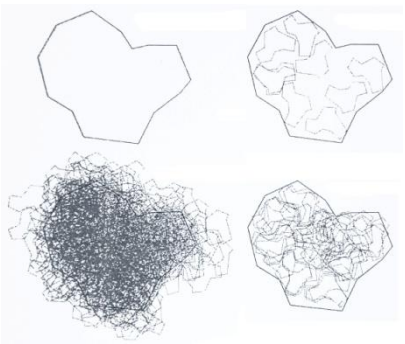


Figura 57. En la figura aparece el proceso de optimización. Una vez que las piezas son encajadas dentro del contorno, se evalúa la situación global del conjunto mediante el *Shared Fitness* y el proceso vuelve a repetirse para maximizar la cantidad de piezas dentro del contorno. *Ibíd*. 1805.

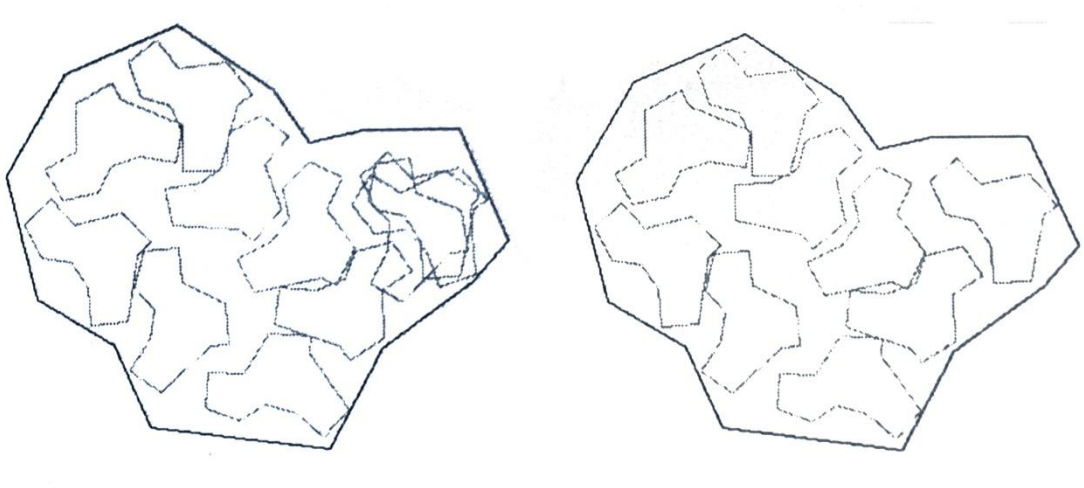


Figura 58. En la imagen de la izquierda el algoritmo se encuentra en la generación 609, en el cual se continúa explorando la mejor colocación de piezas. Sin embargo el algoritmo ya ha encontrado el máximo de piezas posibles para el contorno. En este caso son 9. En la imagen de la derecha el algoritmo se ha detenido en la generación 709, mostrando la mejor solución para encajar las piezas. *Ibíd*. 1806.

Los resultados presentados por Horn, han sido validos tanto para geometrías convexas y cóncavas. EL proceso de optimización planteado por Horn tiene buenos resultados en diferentes problemas de colocación.

6.1.9. Evolutionary Form-Finding of Tensegrity Structures, Francisco Valero Cuevas y Paul Chandana, 2005.

Referencia: CHANDANA, Paul, LIPSON, Hod, VALERO, Francisco. Evolutionary Form-Finding of Tensegrity Structures. Mechanical and Aerospace Engineering, Cornell University, 2005. http://ccsl.mae.cornell.edu/papers/GECCO05_Paul.pdf, [21/08/10].

Las estructuras *Tensegrity*, son estructuras mecánicas estables de 3 dimensiones, las cuales mantienen su forma debido a un complejo balance de fuerzas entre elementos rígidos (barras que no es conectan entre sí) y elementos continuos en tensión (cables). Las estructuras *tensegrity* dan lugar a estructuras livianas de alta resistencia, por lo que han sido utilizadas en áreas de arquitectura, ingeniería y robótica. Sin embargo la manera de conectar los diferentes elementos (barras y cables) para conseguir una estructura estable es considerada como un reto. El trabajo de los investigadores consiste en usar algoritmos evolutivos en la búsqueda de forma en estructuras *tensegrity*. Se muestra en este trabajo que los algoritmos evolutivos pueden ser usados para explorar el espacio arbitrario de estructuras *tensegrity* las cuales son difíciles de diseñar mediante otros métodos, y determinar nuevas formas no regulares. El trabajo, por lo tanto, sugiere que los algoritmos evolutivos pueden ser usados como base de diseño para estructuras de este tipo. Las estructuras *tensegrity* fueron descubiertas por Keneth

Snelson[52] en 1948 y formalmente patentadas por Buckminister Fuller[53] en 1962, quien acuño la palabra *tensegrity*, como una abreviación de *tensile integrity*.

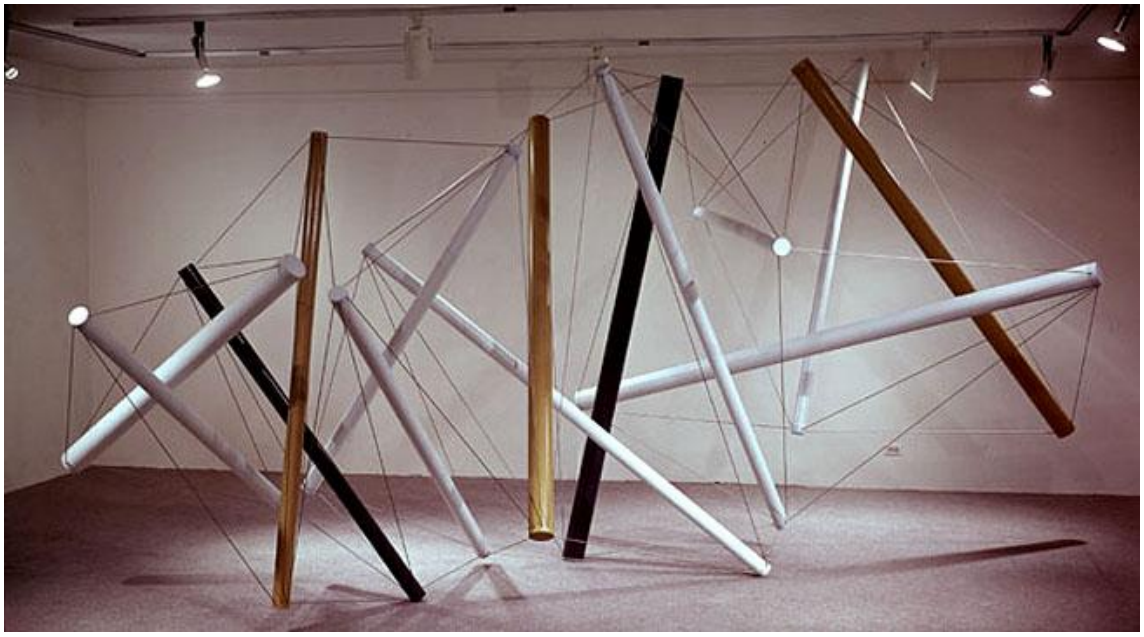


Figura 59. Audrey II, 1966, aluminio recubierto y acero, 2.4 x 4.6 x 2.4m, colección privada. http://www.kennethsnelson.net/icons/bio.htm, [21/08/10].

Existen dos componentes en el problema de la búsqueda de la forma. Determinar el patrón de conectividad el cual permita que se cree una forma estable y determinar la longitud de los elementos rígidos y cables. Existen numerosos acercamientos a la búsqueda de la forma que intentan lidiar con estos dos problemas. En las primeras estructuras desarrolladas por Fuller y Snelson, ellos utilizaron poliedros como base de búsqueda, pero lógicamente siempre resultando en formas regulares.

El problema general para determinar la conectividad y la longitud de los elementos conducen a una estructura estable dentro de un infinito espacio de estructuras posibles, el cual es un problema que aún no ha sido resuelto. Por el momento [21/08/10] no existe una vía matemática que derive en una forma arbitraria de estructuras *tensegrity*.

En el trabajo se utiliza un AG para evolucionar los patrones de conexión y las longitudes de los elementos de una estructura *tensegrity*. El cromosoma usado por los investigadores es una codificación directa de la posición de los vértices y las conexiones entre cables y barras. Siguiendo esto, un algoritmo de relajación es aplicado simultáneamente a todos los cables contrayendo su longitud y todas las barras son definidas en longitud 1. Como resultado en la mayoría de los casos el proceso colapsa la estructura en un enredo de cables y barras de una dimensión. Solo en unos muy pocos casos el resultado fue una estructura *tensegrity* irregular.

52 http://www.kennethsnelson.net/, [21/08/10].
53 http://www.bfi.org/, [21/08/10].

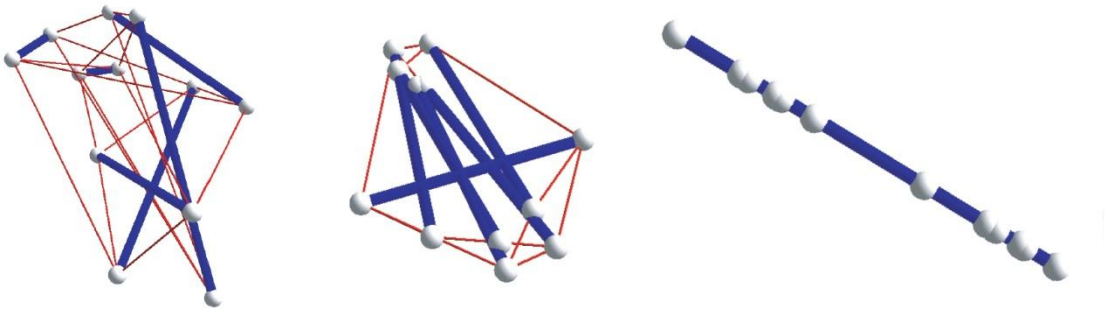


Figura 60. La imagen de la izquierda muestra la configuración inicial de la estructura, la del centro luego de igualar la longitud de las barras a longitud 1 y la tercera cuando los cables son relajados y se les aplica la misma tensión. CHANDANA, Paul, LIPSON, Hod, VALERO, Francisco. Evolutionary Form-Finding of Tensegrity Structures. Genetic and Evolutionary Computation Conference, 2005, pág. 4.

El *fitness* de la estructura está determinado por su volumen, el cual es calculado por un algoritmo de envolvente convexo. El volumen mayor en cada generación es seleccionado cruzado y mutado para continuar con el proceso de optimización. El proceso permitió crear estructuras *tensegrity* estables de 3 dimensiones con 6, 7, 8, 10 y 12 barras.

La cantidad mínima de cables por barras es 3, que es la cantidad de cables requerido en cada vértice para formar la estructura. Si existen menos de tres cables, entonces los vértices de la barra no están completamente restringidos en las tres dimensiones y por lo tanto no es posible mantener una posición estable con respecto al resto de la estructura. Sin embargo el AG fue capaz de diseñar *tensegrities* usando el mínimo número de cables requerido.

El cromosoma de cada *individuo* está dividido en tres partes. La primera región es usada para especificar la ubicación inicial de los vértices. El número de vértices en la estructura es determinado por la cantidad de barras. Si hay N barras, entonces habrá 2N vértices. Por lo que la primera parte del cromosoma consiste en las coordenadas de los vértices de cada barra expresados como una secuencia de números reales. La segunda parte del cromosoma corresponde a la secuencia de pares de intercambios entre barras. La tercera parte es la ubicación final de los vértices y los cables que llegan a cada barra. Por lo tanto el cromosoma es constituido por una 4-tupla[54] o valores reales representando pares de cables C1 y C2 y sus vértices P1 y P2 y donde fueron intercambiados. El número de intercambios realizados en los cables es igual a dos vece el número de cables. Por lo tanto la longitud total del cromosoma es 3n + 8s + 8c, donde n = 2s, s corresponde al número de barras y c al número de cables.

54 Una tupla, en matemáticas, es una secuencia ordenada de objetos.Es una lista con un número limitado de objetos. Las tuplas se emplean para describir objetos matemáticos que tienen estructura, es decir que son capaces de ser descompuestos en un cierto número de componentes. El término tupla se generó sencillamente de una generalización de la siguiente secuencia: dupla, tripla, cuádrupla, quíntupla,... n-tupla. Una tupla de longitud n se describe generalmente como una n-tupla. Una 2-tupla, por ejemplo, se denomina un par o dupla; una 3-tupla una tripla o tripleta. El prefijo n puede ser por generalización cualquier número entero positivo; se puede por ejemplo denominar un cuaterno mediante la representación de una 4-tupla. http://es.wikipedia.org/wiki/Tupla, [21/08/10].

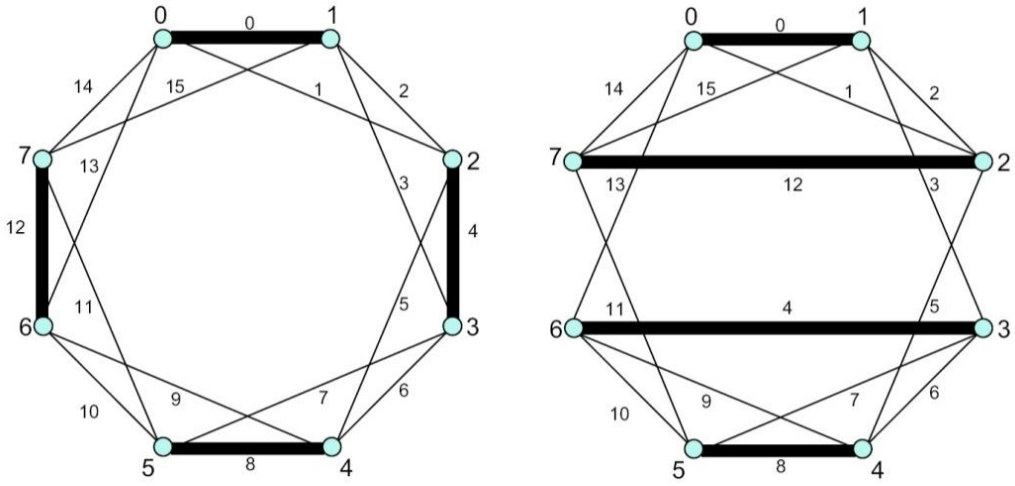


Figura 61. La imagen de la izquierda muestra un ejemplo de una structura con 4 barras. La imagen de la derecha muestra el intercambio de barras. La barra 7-6 se ha cambiado por la barra 7-2 y la barra 2-3 por la 6-3. *Ibíd*. pág. 5.

Con el objetivo de encontrar un equilibrio, la estructura tensegrity es relajada por medio del algoritmo *kinematic relaxation*[55] de dos pasos iterativos. En el primer paso se calculan todas las fuerzas en cada punto, esto es la fuerza residual, si todas estas fuerzas suman 0 la estructura se encuentra en perfecto equilibrio. El segundo paso consiste en ajustar la posición de cada vértice para reducir las fuerzas en cada vértice. Estos dos pasos se repiten iterativamente hasta que la fuerza residual llega al umbral deseado. Cada vértice está conectado a 4 elementos (3 cables y 1 barra) y cada uno de estos, aplica sus fuerzas sobre el vértice al mismo tiempo. Cada cable es contraído a longitud 0 y cada barra a longitud 1. Por eso cada cable aplica su fuerza proporcional a su longitud y cada barra aplica su fuerza proporcional a su longitud real y su diferencia entre la longitud 1.

Cada ejecución del AG fue ejecutado con 100 generaciones y poblaciones de 20 *individuos*. Al final de cada generación, los *individuos* con la aptitud más alta (volumen) son seleccionados y el resto es eliminado. El tipo de selección es por torneo con un tamaño de 3 competidores.

Los resultados indican que los AG fueron capaces de descubrir nuevas estructuras *tensegrities* irregulares. El algoritmo fue testeado para probar estructuras de 6, 8 ,10 y 12 barras. Los investigadores concluyen en su trabajo que las estructuras *tensegrities* de geometría irregular son un problema difícil de resolver a través del análisis matemático directo. Es por esto que hasta ahora los métodos para generar formas libres mediante estructuras *tensegrity* son bastante limitados. Usando codificación directa de vértices y el orden de conexión entre barras

55 LIPSON, Hod. A relaxation method for simulating the kinematics of compound nonlinear mechanisms, ASME, Journal of Mechanical Design, 2006, vol. 128., pág. 719-728.
http://www.google.es/search?sourceid=chrome&ie=UTF-8&q=A+relaxation+method+for+simulating+the+kinematics+of+compound+nonlinear+mechanisms [22/08/10]

y cables los AG pudieron exitosamente explorar un espacio de búsqueda muy grande y ruidoso.

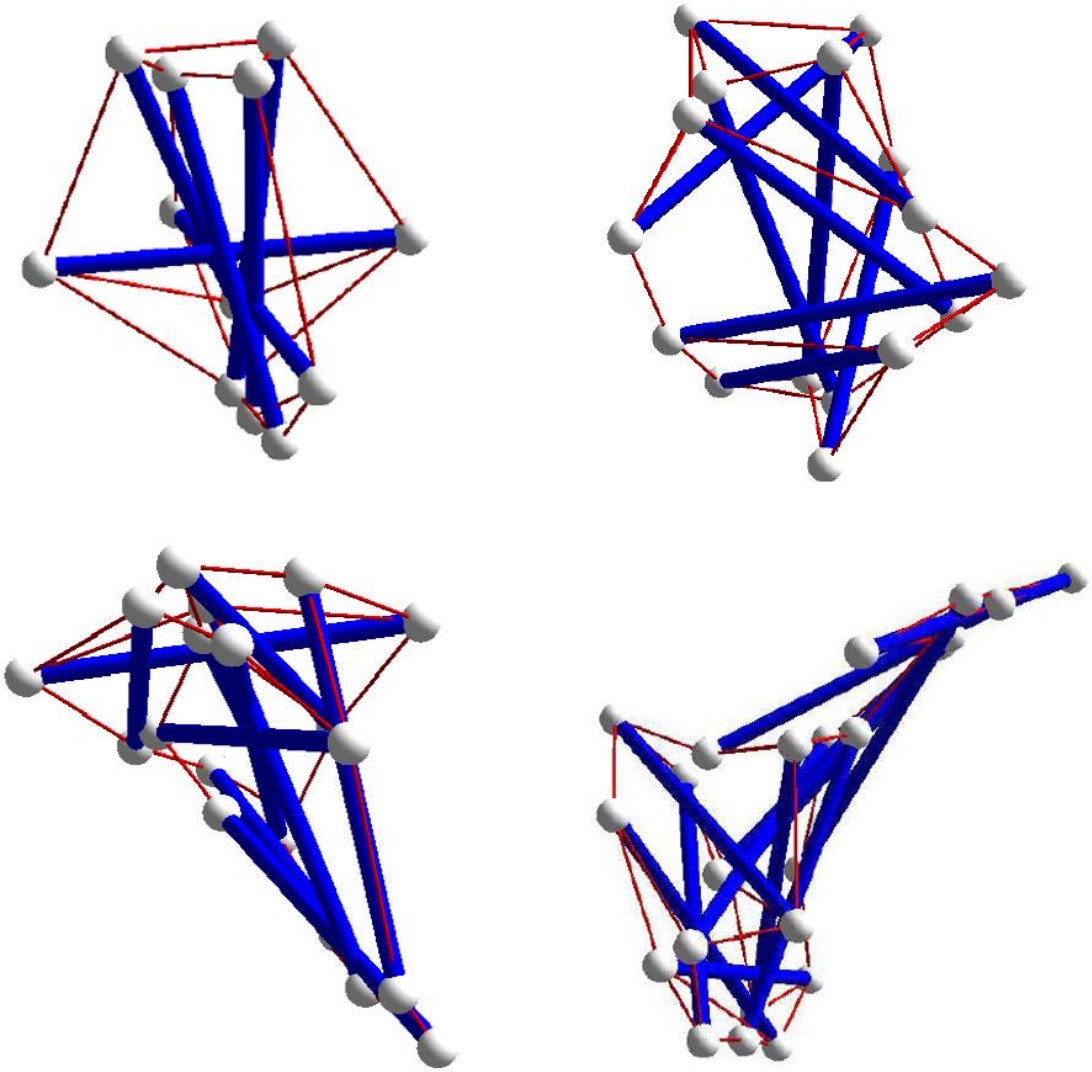


Figura 62. Resultado de las estructuras Tensegrity 3D luego de 100 generaciones y con poblaciones de 20 *individuos*, para 6, 8, 10 y 12 barras por estructura. *Ibíd*. Pág. 9

6.1.10. Genetic Algorithms for Construction Site Layout in Project Planning. MAWDESLEY, Michael J., SAAD, H. Al-jibouri y HONGBO, Yang, 2002.

Referencia: MAWDESLEY, Michael J., SAAD, H. Al-Jibouri. Y HONGBO, Yang. *Genetic Algorithms for Construction Site Layout in Project Planning*. Journal of construction engineering and management, 2002, Vol. 128. pp. 418-426. http://doc.utwente.nl/67239/, [22/08/10].

El trazado de un lugar tiene que ver con la geometría del terreno, el emplazamiento, los accesos y la permanencia de las instalaciones durante la fase de construcción. El trazado es

muy importante porque puede afectar significativamente los costos del proyecto. Los investigadores describen el tipo de problema y proponen los AGs como posible solución.

El objetivo del trazado consiste en determinar los trabajos temporales que se necesitan y emplazarlos en el terreno y tiempo durante la fase de construcción. El trazado tiene efectos en los costos del proyecto, en la calidad del trabajo, en la seguridad de los operarios y en aspectos ambientales. Por lo que es un problema de planificación.

Los criterios considerados son disminución del tiempo de viaje, eliminación de movimientos innecesarios de recursos y manipulación de materiales, ahorrando tiempo no productivo. Sin embargo, en la práctica se dedica muy poca atención en el diseño del trazado. Otra cuestión a considerar es que el trazado es dinámico debido a los cambios que sufre el terreno durante la fase de construcción.

Los investigadores definen 4 puntos importantes a resolver. El primero tiene que ver con las rutas de acceso y tráfico hacia adentro y afuera del lugar de construcción. Esto involucra la movilización de trabajadores, equipos y materiales. Este punto afecta los costos, la seguridad y el ambiente. Como ideal se plantea que las rutas de acceso y tráfico no deben interrumpir las líneas de servicio, unirlas pero no cruzar las áreas de trabajo. El segundo punto está relacionado con el almacenamiento y manipulación de materiales. Dentro de este punto se distinguen tres tipos: Primero, los materiales valiosos que deben ser protegidos, en segundo lugar los materiales peligrosos que requieren ser protegidos y finalmente, los materiales que no requieren de protección. Además la manipulación de materiales de construcción suele ser costoso y necesita el uso de equipos especiales. Lo ideal es que dependiendo de la ubicación del material se evite tener que moverlo más de 2 veces. El tercer punto tiene que ver con los edificios de gestión, los cuales deben permitir una buena visión de los trabajos y estar libres de ruidos. Otras dependencias como aseos no deben causar interrupción al proyecto. El cuarto punto hace referencia a los talleres, salas de trabajo y servicios. Lo ideal es que los talleres deben estar localizados con acceso fácil y rutas cortas al área de construcción.

Se comienza definiendo el problema basado en un terreno rectangular orientado según el sistema de coordenadas X e Y. Se consideran dos dependencias, talleres y oficinas, que también estarán orientadas según los ejes de coordenadas. La localización de las dependencias se identifica por las coordenadas de las esquinas; Así, la dependencia 1 se representa por los vértices (X_1, Y_1) y (X_1', Y_1'). El punto de acceso también se representa por coordenadas. Las conexiones entre dependencias o de estas entre las áreas de construcción y acceso pueden ser rectilíneas o reticulares.

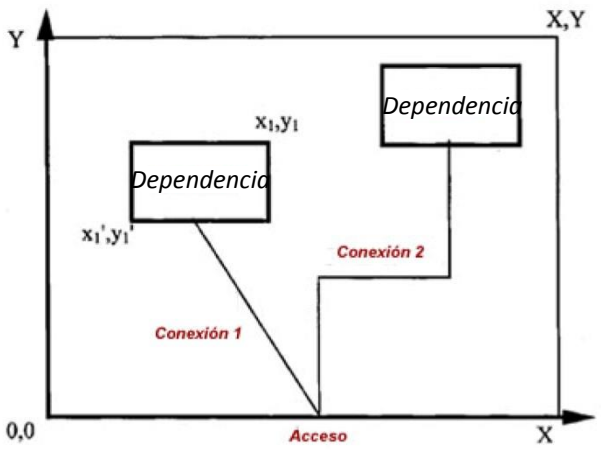


Figura 63. Área rectangular para el trazado. MAWDESLEY, Michael J., SAAD, H. Al-Jibouri. Y HONGBO, Yang. Genetic Algorithms for Construction Site Layout in Project Planning. Journal of construction engineering and management, 2002, Vol. 128. pp. 422.

El objetivo consiste en ubicar M dependencias, tanto temporales como permanentes. La dependencia i, tiene las coordenadas (X_i, Y_i) y (X_i', Y_i'). y su área es $a_i = |(X_i, Y_i) \text{ y } (X_i', Y_i')|$. La distancia entonces entre ambas dependencias es el cálculo de la distancia entre dos puntos.

$$distancia\ entre\ dependencias\ ij\ = \sqrt{(\mathrm{Xi} - \mathrm{Xj})^2 + (\mathrm{Yi}\ -\ \mathrm{Yj})^2}$$

La función objetivo entonces es definida de la siguiente manera. *La aptitud = F(costo de transporte de material + costo de emplazamiento de dependencias + costo de retiro de dependencias + costo de visitas de personal al sitio de trabajo + otros)*.

El diseño de la función de aptitud puede requerir de algunas restricciones relativas a la localización, superposición o distancia entre las dependencias. Si la ubicación de ciertas áreas pudiese ser inapropiada, por ejemplo, por cuidado ambiental. Una forma de establecer la restricción sería incrementar el costo de emplazamiento en esa área o bloquear la posibilidad de que pueda haber dependencias o circulaciones. En un área específica no puede haber más de una dependencia, no se pueden superponer. La distancia entre dependencias, está relacionada con las actividades que se desarrollen dentro. Por ejemplo, es mejor posicionar un taller ruidoso alejado de las oficinas. Este tipo de restricciones se expresan por distancias mínimas distanciaij >= D_{min} o por distancias máximas $distancia_{ij}$ <= D_{max}.

El cromosoma establecido por los investigadores consiste en una lista de valores reales de las coordenadas de las dependencias y puntos de acceso. Se trata de N áreas a ser posicionadas.

Para afrontar el problema el terreno es dividido en cuadriculas. Los costos de distribución varían según la cuadricula, identificando tres tipos de costos: Costo de emplazamiento, costo de retiro y costo de viaje. Se establece un costo de emplazamiento para dependencias temporales, permanentes y de cualquier área ocupada. Si el área no está disponible se fija un costo alto. La circulación pasa a través de la cuadricula y los costos reflejan la dificultad del recorrido. El costo mínimo de viaje puede ser establecido usando la teoría de gráficos para recorridos rectilíneos o por geometría.

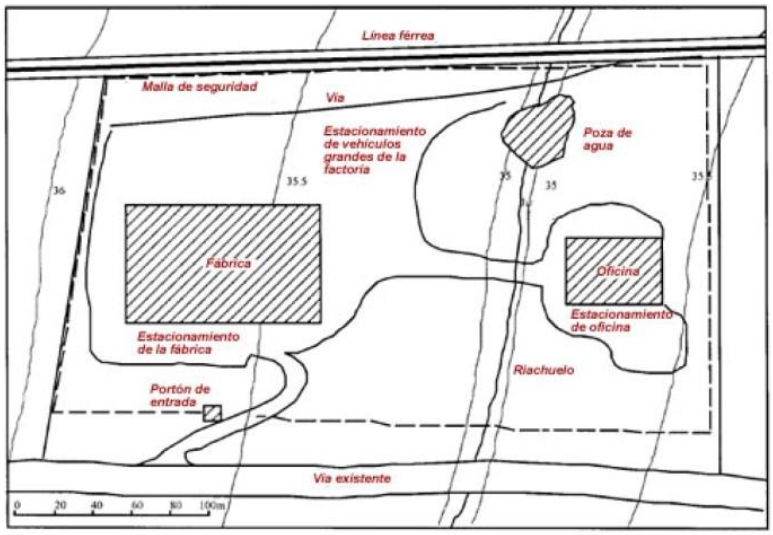


Figura 64. Trazado del lugar del proyecto. *Ibíd*. pág. 424**.**

El trazado del proyecto tiene las siguientes instalaciones permanentes: fábrica, estacionamiento de automóviles de la fábrica, estacionamiento de vehículos grandes de la fábrica, oficinas y estacionamiento de oficinas. Las dependencias temporales que deben ser localizadas son las siguientes. Sitio para las oficinas, un almacén de refuerzos, un almacén para el concreto y un almacén general.

El esquema general del AG, produce por cada generación una *población* de 20 *individuos*. Cada uno de ellos representa una configuración del trazado del terreno, los cuales son evaluados según la función de aptitud. Los *individuos* más aptos con menos costo total, son cruzados, mutados y vueltos a evaluar hasta que el AG alcance el número máximo de generaciones. Que es 100.

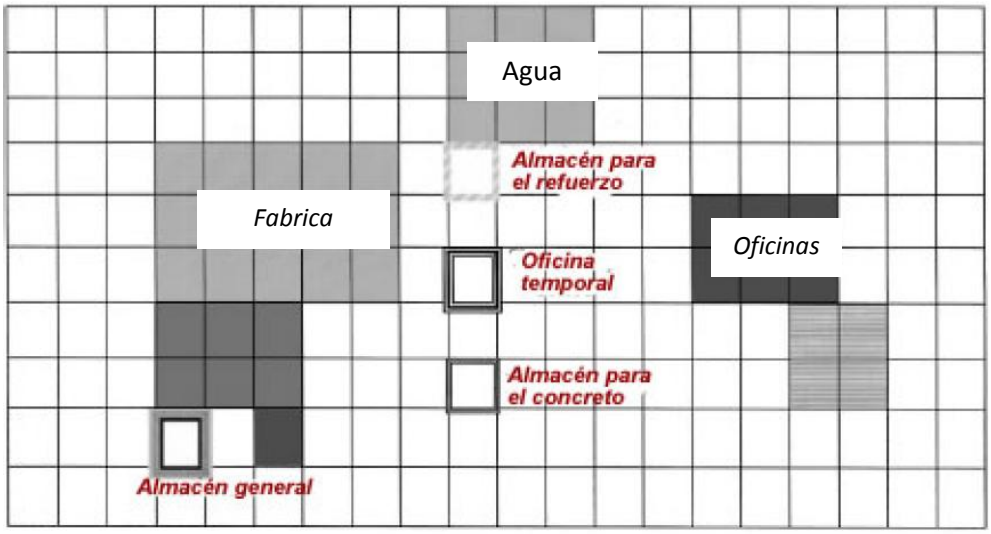


Figura 65. El AG propuso una solución bastante obvia y fue localizar las dependencias equidistantes de los lugares de construcción. Sin embargo el problema también es bastante simple y de lógica clara. Lo interesante del asunto es que el AG podría lidiar con más restricciones y más dependencias, donde la solución no fuese obvia y aun así encontrar muy buenos resultados. *Ibíd*. pág. 424**.**

6.1.11. Optimización de portafolios accionarios a través de un micro algoritmo genético, GUTIÉRREZ, Mauricio., TORRES, Erick., GÁLVEZ, Patricio., POO, Germán. 2007.

Referencia: GUTIÉRREZ, Mauricio., TORRES, Erick., GÁLVEZ, Patricio., POO, Germán. *Optimización de portafolios accionarios a través de un micro algoritmo genético*, 2007. http://revistas.concytec.gob.pe/scielo.php?script=sci_arttext&pid=S1810-99932007000200003&lng=es&nrm=iso. [22/08/10].

Este trabajo muestra la optimización de portafolios de acciones mediante micro Algoritmos Genéticos. Este tipo de optimización es multi-objetivo, en donde se busca maximizar la rentabilidad y minimizar el riesgo. Esto implica realizar una negociación entre ambos objetivos y buscar la solución óptima que satisfaga ambos objetivos basado en óptimos de Pareto. Los resultados planteados por los investigadores apuntan que los AGs son más eficientes que otras técnicas de predicción[56].

La programación multi-objetivo[57] consiste en encontrar los valores de una lista de variables que satisfagan los elementos presentes en las diferentes funciones objetivo. Estás funciones son una descripción matemática del desempeño de las decisiones tomadas que están en conflicto entre si y que se suelen medir en unidades diferentes. Optimizar varias funciones a la vez no consiste en encontrar un óptimo para cada función, sino en proponer un conjunto de soluciones para cada función que contribuya en conseguir una buena aptitud global. La solución global de la optimización fue propuesta por Wilfredo Pareto en 1886[58].

Un *micro* Algoritmo Genético (*micro*-AG) es un AG con una *población* muy pequeña y un proceso de re-inicialización, combinado con un archivo externo para guardar los mejores resultados encontrados previamente y un mecanismo para mantener la diversidad en la

---

56 Los resultados obtenidos señalan que esta aplicación es más eficiente que otros procesos de similares características (Nondominated Sorting Genetic Algorithm II (NSGA II) y Pareto Archive Evolution Strategy (PAES)), pero debido al período de tiempo y las características del mercado local, su poder de predicción es bajo. *Optimización de portafolios accionarios a través de un micro algoritmo genético*, GUTIÉRREZ, Mauricio., TORRES, Erick., GÁLVEZ Patricio., POO, Germán. 2007. http://revistas.concytec.gob.pe/scielo.php?script=sci_arttext&pid=S1810-99932007000200003&lng=es&nrm=iso. [22/08/10]. Otros trabajos que concuerdan en descripción son: MAHFOUD, Sam., MANI, Ganesh, *Financial forecasting using genetic algorithms*, 1996, http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.86.9698. [22/08/10].

57 La optimización multio-bjetivo, puede ser definida como un problema de optimización que presenta dos o más funciones objetivo. El inconveniente principal que presenta este tipo de problemas en relación a un modelo de objetivo único, radica en la subjetividad de la solución encontrada. Un problema multio-bjetivo no tiene una solución óptima única, más bien, genera un conjunto de soluciones que no pueden ser consideradas diferentes entre sí. BAESLER, Felipe., CEBALLOS, Luis., RAMÍREZ, Milton., *Programación multiobjetivo de máquinas moldureras a través de algoritmos meméticos*, Departamento de Ingeniería Industrial, Universidad del Bío-Bío, Concepción, Chile, 2006, http://www.scielo.cl/scielo.php?pid=S0718-221X2006000300005&script=sci_arttext [01/06/10]

58 "El concepto de eficiencia de Pareto (también llamado óptimo de Pareto, Pareto-optimización u óptimo Paretiano) es aquella situación en la cual se cumple que no es posible beneficiar a más elementos de un sistema sin perjudicar a otros. Se basa en criterios de utilidad: si algo genera o produce provecho, comodidad, fruto o interés sin perjudicar a otro, provocará un proceso natural de optimización hasta alcanzar el punto óptimo" http://es.wikipedia.org/wiki/Eficiencia_de_Pareto [01/06/10]

población. Los *micro*-AG al tener una *población* pequeña de *individuos* convergen muy rápido por lo que la búsqueda se centra en una área específica del espacio de soluciones. Sin embargo son altamente competitivos y no tienen mayores dificultades de implementación. Los *micro*-AG fueron sugeridos por Goldberg y Richardson[59]. Ellos plantearon que una *población* de tres *individuos* era suficiente para converger, sin importar la longitud del cromosoma. El proceso plantea aplicar los operadores genéticos básicos (cruce, mutación, selección) a una *población* pequeña, generada aleatoriamente, hasta alcanzar una convergencia nominal (cuando todos los *individuos* tienen sus cromosomas muy parecidos o iguales). Luego se debe generar una nueva *población* para transferir los mejores *individuos* de la primera generación. Esta nueva generación es realizada aleatoriamente para evitar la convergencia prematura.

El primer reporte de *micro*-AGs fue hecho por Krishnakumar[60], que utilizó una *población* de 5 *individuos*, un porcentaje de cruce de 1 y un porcentaje de mutación 0. Adoptó una estrategia de elitismo, la cual copiaba el mejor *individuo* encontrado en la actual *población* a la siguiente generación. La selección era realizada por 4 *individuos*, donde se elegía el más apto de ellos para ser seleccionado en la siguiente generación. Krishnakumar comparó su *micro*-AG con un AG tradicional de una *población* de 50 *individuos*, un porcentaje de cruza de 0.6 y un porcentaje de mutación de un 0.001. Su trabajo reportó mejores y más rápidos resultados con su *micro*-AG sobre 2 funciones. Otros investigadores han desarrollado trabajos sobre *micro*-AGs, como KARR (1991)[61], DOZIER, BOWEN y BAHLER (1994)[62], JOHNSON y ABUSHAGUR (1995)[63], XIAO y YABE (1998)[64], TOSCANO (2001)[65].

Básicamente los *micro*-AG se utilizan cuando la función de aptitud es muy costosa computacionalmente, en el sentido que la evaluación de aptitud de un *individuo* puede tardar horas. Entonces se hace prácticamente imposible trabajar con poblaciones de 50 *individuos* y 200 generaciones.

---

59 GOLDBERG, D. E. y RICHARDSON, J. Genetic algorithm with sharing for multimodal function optimization. In J. J. Grenfestette, editor, Genetic Algorithms and Theory applications: Proceedings of the Second International Conference on Genetic Algorithms, Lawrence Erlbaum, 1987. Pág. 41–49.

60 KRISHNAKUMAR, K.. Micro-genetic algorithms for stationary and non-stationary function optimization. In SPIE Proceedings: Intelligent Control and Adaptive Systems, 1989. Pág. 289–296,

61 KARR, Charles L.. Air-Injected Hydrocyclone Optimization via Genetic Algorithm. In Lawrence Davis, editor, Handbook of Genetic Algorithms. Van Nostrand Reinhold, New York, 1991. pág. 222–236

62 DOZIER, G., BOWEN, J., BAHLER, D.. Solving small and large scale constraint satisfaction problems using a heuristic-based microgenetic algorithm. In Proceedings of the First IEEE Conference on Evolutionary Computation, 1994. pág. 306–311,

63 JOHNSON, E.G. y ABUSHAGUR, M.A.G. Micro-Genetic Algorithm Optimization Methods Applied to Dielectric Gratings. Journal of the Optical Society of America, 12(5):1152–1160, 1995.

64 XIAO, Fengchao., y YABE, Hatsuo. Microwave Imaging of Perfectly Conducting Cylinders from Real Data by Micro Genetic Algorithm Coupled with Deterministic Method. IEICE Transactions on Electronics, E81-C(12):1784–1792, 1998.

65 TOSCANO, G. Optimización multiobjetivo usando un Micro algoritmo genético. Maestría en Inteligencia Artificial, Universidad Veracruzana – LANIA. 2001.

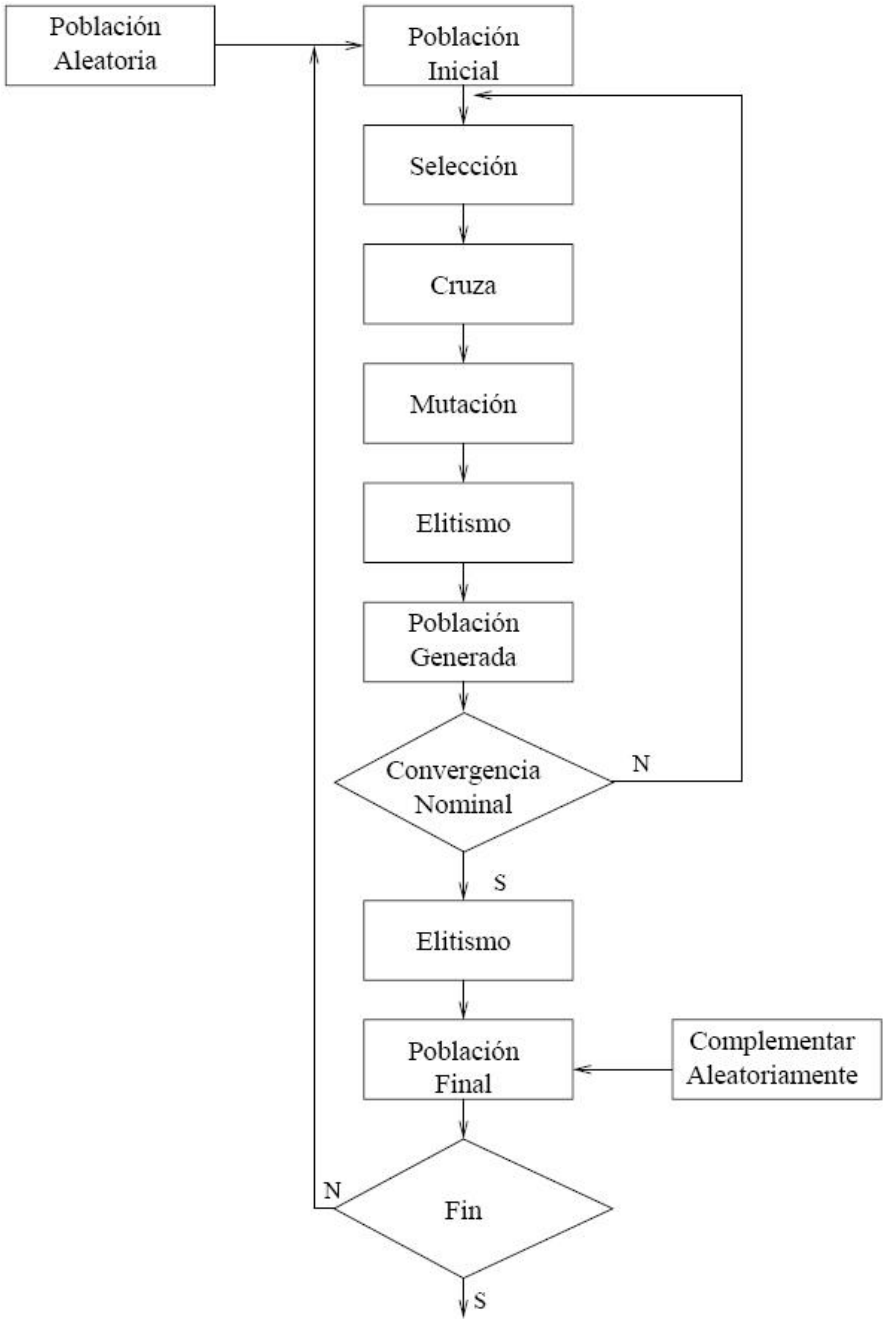


Figura 66. Algoritmo general de un *micro*-AG. TOSCANO, G. Optimización multiobjetivo usando un *Micro* algoritmo genético. Maestría en Inteligencia Artificial, Universidad Veracruzana – LANIA. 2001. Pág. 14.

El problema se caracteriza por la variedad de activos financieros y los porcentajes de riqueza invertida en cada uno de ellos. Lo cual hace que el espacio de búsqueda formado por sus posibles combinaciones crezca de manera exponencial. Por lo tanto el problema plantea los siguientes requerimientos:

- Dado un conjunto de activos financieros, mostrar una perspectiva de las mejores combinaciones posibles de rentabilidad y riesgo.
- Dado un nivel de riesgo, obtener la máxima rentabilidad.
- Dado un nivel de rentabilidad, minimizar el riesgo.

En la teoría se establece que es preferible la cartera con menor riesgo; Y de la misma manera por cada tipo de riesgo es preferible una inversión con mayor rentabilidad, tipo Pareto. Esto quiere decir que el conjunto, de los mejores valores para las funciones objetivo que se pretenden optimizar en distintas combinaciones de entre ellas, corresponde a una frontera eficiente. Está consiste en el conjunto de valores óptimos para el problema, ya que está compuesto por los portafolios con las rentabilidades más altas para cada nivel de riesgo. En otras palabras el objetivo consiste en encontrar la frontera de soluciones que beneficie a todos los activos dentro de la cartera sin tener que perjudicar a alguno de ellos.

Los investigadores trabajaron con un volumen de datos de 10 años, (1996 – 2005). Usaron 7 años de estos informes para evaluar el comportamiento de los portafolios y 3 años para evaluar el valor predictivo del modelo. Considerando una periodicidad de dos meses y 74 activos.

El diseño del cromosoma está compuesto por 74 genes, uno para cada activo. Cada gen representa el porcentaje de riqueza invertido para cada activo del portafolio. El tamaño de los genes puede variar y depende de la cantidad de valores que posea.

El proceso se Inicia con una *población* (P) de *individuos* generados aleatoriamente en memoria. Esta memoria está dividida en dos partes, una reemplazable (Pr) y otra no reemplazable (Pnr). De esta memoria, el *micro*-AG obtendrá una pequeña *población* de trabajo (Pt) para ser evolucionada. Por cada generación se conservan las dos mejores soluciones (S1) y (S2). Luego son comparadas con otros *individuos*. Si son mejores son guardadas en la *población* (Pr). El ciclo se vuelve a repetir, hasta que se alcanza la condición de término. La evolución de (Pt) por parte del *micro*-AG es realizada en forma tradicional: Selección, cruce, mutación y elitismo, hasta que se alcanza la convergencia.

El desempeño del *micro*-AG experimentó un notable progreso en la etapa de ensayo (periodo de 7 años), en la cual el algoritmo fue probado en distintas situaciones, las cuales generaron algunas modificaciones en el modelo inicial. De las mejoras realizadas, gran parte fueron producto de la observación empírica, motivada por la significancia de los cambios producidos y también por la necesidad de reducción del tiempo de ejecución del algoritmo.

Los resultados del *micro*-AG fueron muy superiores comparados con otros algoritmos, ya que en varias de las pruebas realizadas presentó un mejor rendimiento que otros más estudiados en el tiempo como el Nondominated Sorting Genetic Algorithm II (NSGA II) y Pareto Archive Evolution Strategy (PAES).

El hecho de trabajar con una restricción de siete años hace que el modelo no pueda predecir con suficiente precisión el comportamiento de los portafolios de inversión. Cuando se analizaron las proyecciones de rentabilidad, se observó una sobreestimación de las rentabilidades en los distintos niveles de riesgo. El algoritmo planteado por los investigadores no es un “tomador automático de decisiones financieras”, sino que es un prototipo que contiene toda la funcionalidad necesaria para resolver el problema de rentabilidad y riesgo. El administrador financiero deberá evaluar que ponderación le dará a los resultados de este estudio con respecto a las otras variables que maneja para tomar sus decisiones.

6.2. Conclusión del Estado del Arte

Los problemas multi-objetivo y de combinatoria, son problemas cotidianos a los cuales nos enfrentamos a diario en nuestra vida. En el mundo real, la mayoría de los problemas tiene un espacio de búsqueda muy grande donde las técnicas tradicionales no han demostrado ser eficientes, por lo que es necesario recurrir a técnicas alternativas. Los Algoritmos Genéticos, son un conjunto de algoritmos que tienen características ideales para resolver este tipo de problemas. Lo mencionado anteriormente se debe a que estos algoritmos tienen sus bases en una estrategia que ha venido solucionando este tipo de problemas durante millones de años. Lo curioso es que solo hace unos 40 años se ha comenzado a realizar investigaciones sobre este tipo de problemas y sus estrategias para resolverlos.

Los trabajos presentados en este capítulo si bien buscan diferentes (multi) objetivos y enfrentan diferentes problemas, todos ellos están basados básicamente en la misma estrategia: Una incesante producción de prototipos y un implacable rechazo a modelos defectuosos. Es de suponer que la mayoría de los trabajos sobre AGs sean muy similares en cuanto a estrategia y probablemente no se esté equivocado. Las diferencias entre los distintos algoritmos las encontramos en las diferentes versiones que existen sobre estas técnicas. Tamaños de *población*, tipos de cruce, tipos de mutación, arquitectura del cromosoma, número de iteraciones, técnicas de análisis, implementación, etc. sin embargo todos están basados en la misma estrategia.

Para desarrollar estos proyectos los investigadores han tenido que diseñar y desarrollar su propio software. Incluso en algunos casos, construir su propio hardware. Este tipo de diseño creado en base a programación ha servido para aprender a pensar y crear una propia lógica basada en las estrategias de la naturaleza. Si bien los trabajos pueden llegar a ser muy complejos todos ellos pueden ser resumidos a simples declaraciones lógicas y manejos de datos.

Otra similitud en estos proyectos es que su éxito recae en el manejo y demostración de la técnica. La técnica pasa por la compresión de un modelo teórico (selección de las especies de Darwin) y el cuidadoso entendimiento de una estructura del lenguaje y su información codificada. Esto implica plantearse la manera en como habitualmente pensamos.

Elección de Herramientas

Este capítulo está dividido en 3 partes. La primera de ellas habla sobre los dos lenguajes de programación (VB y C#) usados por el autor para desarrollar los experimentos del capítulo 8. La segunda parte es un listado de programas de análisis térmico y sus principales características. La tercera describe ciertos programas de modelado de los cuales algunos de ellos son paramétricos y asociativos. El capítulo "Elección de Herramientas" acaba con una breve conclusión de porque se eligieron ciertos programas y porque otros no. El éxito de los trabajos del capítulo 8 depende de la correcta toma de decisiónes del programa y lenguaje que se vaya a implementar.

7.1. Lenguajes de Programación

7.1.1. ESTRUCTURA GENERAL DE LENGUAJES DE PROGRAMACIÓN

Los lenguajes de programación en general tienen un vocabulario definido, además de reglas ortográficas, de estructura, de puntuación, es decir la sintaxis que se debe seguir y por supuesto su propia semántica. Las reglas del lenguaje deben tomarse en cuenta para que las ideas que tratamos de expresar tengan sentido. Un lenguaje de programación es mucho más estricto que uno natural en el sentido de que no pueden existir ambigüedades. Por el contrario, los lenguajes de programación son claros, específicos y directos, no admitiendo ambigüedades o suposiciones.

La mayoría de los lenguajes de programación están basados principalmente en 4 conceptos. Cada sintaxis difiere de un lenguaje a otro, sin embargo la estructura y los conceptos a la hora de desarrollar los códigos se aplican de igual manera para los diferentes lenguajes. Comprender estos cuatro conceptos es la clave para aprender un lenguaje de programación y de esta manera es más fácil poder entender otro tipo de lenguajes.

LOS CONCEPTOS SON:

- Definición y comprensión de datos. Descripción de la información, saber abstraer un problema planteado, definir sus variables y su mecánica matemática.
- Condiciones de ejecución. Controla el flujo del programa.
- Ejecución iterativa. Comprimir, procesar y repetir las condiciones de ejecución una cierta cantidad de veces.
- Módulo. Organiza el código para su reutilización.

El autor tiene experiencia en los lenguajes de programación Visual Basic y C++.

7.1.2. VISUAL BASIC

Visual Basic es un lenguaje de programación desarrollado para *Microsoft* y fue presentado en 1991 como una extensión de Basic. Es un lenguaje de fácil aprendizaje pensado tanto para programadores expertos y principiantes. Su característica es que se encuentra guiado por eventos y centrado en un motor de formularios que facilita el desarrollo de aplicaciones gráficas. Su sintaxis ha sido heredada del antiguo Basic, incorporando características típicas de los lenguajes estructurados modernos. Se ha agregado una implementación limitada de la Programación Orientada a Objetos (POO), admite polimorfismo en sus funciones, clases, no necesita manejo de punteros y no admite la herencia[66]. Visual Basic posee varias bibliotecas para el manejo de datos, y se puede conectar con cualquier base de datos.

LOS INCONVENIENTES DE ESTE LENGUAJE SON:

- Sin soporte de *Microsoft* desde el 4 de Abril de 2008, sin embargo existe una gran cantidad de libros y recursos de internet en a los cual se puede recurrir. VB es el lenguaje posiblemente con más desarrolladores en el momento.
- Los ejecutables generados son relativamente lentos.
- No permite la programación a bajo nivel.
- Sólo permite el uso de funciones de librerías dinámicas (DLL).
- Tiene escasa implementación de POO.
- No incluye operadores de desplazamiento de bits como parte del lenguaje.
- No permite el manejo de memoria dinámica, punteros, etc. Como parte del lenguaje.

Visual Basic es el dialecto adaptado para ser usado como lenguaje de macros de los programas que componen Office. Fue pensado originalmente para el desarrollo de páginas web y luego se utilizó también para hacer scripts en el sistema operativo.

Actualmente el autor utiliza *Rhinoscript*, una modificación técnica de Visual Basic para desarrollar aplicaciones dentro del programa de modelado *Rhinoceros*. Dentro de la comunidad científica es difícil de encontrar AGs programados en este tipo de lenguaje. Los primeros tres proyectos del capítulo 8 fueron desarrollados íntegramente en *Rhinoscript*.

66 Es una propiedad que permite que los objetos sean creados a partir de otros ya existentes, obteniendo características (métodos y atributos) similares a los ya existentes. Es la relación entre una clase general y otra clase más específica. Es un mecanismo que nos permite crear clases derivadas a partir de clase base, Nos permite compartir automáticamente métodos y datos entre clases subclases y objetos. Por ejemplo: Si declaramos una clase párrafo derivada de un clase texto todos los métodos y variables asociadas con la clase texto son automáticamente heredados por la subclase párrafo. Tomado de la página web: http://es.wikipedia.org/wiki/Herencia_(programaci%C3%B3n_orientada_a_objetos) [08 / 07 / 08]

7.1.3. C++

El lenguaje C++ fue diseñado ha mediado de los años 1980 como una extensión del lenguaje C. Soporta la programación estructurada, la programación genérica[67] y la programación orientada a objetos (POO).

Actualmente existe un estándar denominado ISO C++, en la que participan la mayoría de los fabricantes de compiladores modernos. Las principales características de C++ son las facilidades que proporciona la POO para el uso de plantillas o la programación genérica. C++ es considerado por muchos como el lenguaje más potente, debido a que permite trabajar tanto a alto como a bajo nivel, sin embargo es un lenguaje que a diferencia de VB no tiene automatismos, lo que obliga hacer todo manualmente. La desventaja de esto es que lo hace difícil de aprender y su ventaja es que lo hace muy flexible. C++ funciona cargando directivas en la parte superior del editor de texto, estás directivas permiten el uso de diferentes funciones, como matemáticas, gráficas, etc. Incluso es posible crear las propias directivas para personalizar aún más las funciones.

CONCEPTOS GENERALES DE LA PROGRAMACIÓN ORIENTADA A OBJETOS (POO)

Los objetos son la clave para entender la tecnología orientada a objetos. La Programación Orientada a Objetos (POO), se basa en la estructura del mundo real donde todas las cosas que vemos comparten dos estados: Propiedades y comportamiento.

En el mundo real, así como también en programación los objetos varían en complejidad. Por ejemplo, una lámpara de mesa tiene en su comportamiento dos estados posibles, encendido y apagado. En cambio una radio tiene más estados en su comportamiento; Encendido y apagado, volumen, modo: CD o radio, posición de la antena, etc. Si miras con detención notarás que estos objetos están compuestos por otros objetos que también tienen sus propiedades y comportamientos. Está abstracción del mundo real es la programación orientada a objetos.

Los softwares están diseñados conceptualmente a semejanza de los objetos del mundo real, están basados en propiedades y comportamientos. Un objeto en programación almacena estos dos estados en variables (propiedades) y en funciones o métodos (comportamientos). Las funciones operan internamente en los objetos y sirven como primer mecanismo para comunicarse con otros objetos.

En la POO se le llama encapsulamiento al esconder los datos miembros de un objeto, de manera que sólo se puedan modificar las operaciones definidas para ese objeto. Cada objeto está aislado de su exterior de modo natural y la aplicación entera es un rompecabezas de

67 La programación genérica es un tipo de programación que está mucho más centrada en los algoritmos que en los datos. La idea de esta forma de programar pretende generalizar las funciones utilizadas para que puedan usarse en más de una ocasión. Esto se consigue parametrizando lo máximo posible el desarrollo del programa y expresados o devueltos de la forma más simple posible, evitando detalles concretos. La biblioteca de funciones conseguida con esta manera de programa permite que esas funciones puedan servir para más programas de los que, otras más concretas, podrían ser útiles; y también aplicando pocos cambios, conseguir que realice diferentes acciones. Tomado de la página web: http://es.wikipedia.org/wiki/Programaci%C3%B3n_gen%C3%A9rica 08 / 07 / 2008

objetos compilados (DLL). El aislamiento del objeto protege los datos asociados a un objeto contra su modificación para que nadie tenga derecho a acceder a ellos. De esta manera el usuario de este objeto puede olvidarse de la implementación de nuevas funciones y métodos para concentrarse en sólo como usarlos y además evitar cambiar sus estados de manera imprevista.

Un edificio también es un objeto, porque tiene sus propiedades y comportamientos. Los planos del edificio son por lo tanto una representación abstracta del objeto edificio. Entonces las clases en la POO son las representaciones abstractas de los objetos. Por ejemplo con el plano del edificio podemos definir cómo será la ventana una vez que se construya, sus dimensiones (propiedades) y también es posible especificar qué tipo de vidrio llevará la ventana, y definir la transmisión calórica de la ventana (comportamiento).

Hoy en día la POO es requisito de los lenguajes de programación y los softwares para arquitectos la implementan en todas sus aplicaciones. La POO es la base del BIM, del diseño paramétrico y asociativo. Ha influido en la manera en como se ve el mundo. Diferentes pensadores se han basado en la POO para desarrollar sus teorías, como por ejemplo Gilles Delleuze, Bernard Cache, Jacques Derrida, entre otros.

LOS INCONVENIENTES DE C++ SON:

- Es más difícil de aprender que VB.
- Su extremada libertad obliga a hacer todo a mano haciendo perder tiempo buscando errores, símbolo de llaves o puntos y comas olvidados.
- Generalmente los compiladores no dicen exactamente donde se encuentra el problema.
- La implementación gráfica es más complicada de alcanzar o a través de software, como *Maya* o *Rhinoceros*.

C++ es un lenguaje muy versátil, se utiliza desde el desarrollo de blogs hasta la implementación de inteligencia artificial para robots. La mayoría de los trabajos científicos en AGs son implementados en C++ o en alguno de sus derivados como C#, Java o Perl.

El ejercicio 4 (Estrategia Evolutiva para el diseño de Arquitectura optimizada) del capítulo 8 está desarrollado en GCScript y C#.

El lenguaje de script *GCScript* es una modificación técnica del lenguaje de programación C#. *GCScript* viene de la familia de lenguajes C, como son C, C++, Java, y es optimizado para trabajar dentro de *GenerativeComponents*[68]. Este lenguaje provee todo los aspectos del lenguaje de programación C#. Además es el único que incluye una herramienta de auto-replicación, esto significa que se puede utilizar una lista o una lista anidada donde se espera un solo valor.

C# (*Sharp*) es un lenguaje de programación desarrollado por *Microsoft* como parte de la tecnología .NET, el que luego fue aprobado por los estándares ECMA e ISO. C# puede ser

[68] GenerativeComponents (GC) es un sistema paramétrico y asociativo que otorga la posibilidad de explorar diferentes posibilidades de diseño. Permite trabajar tanto gráficamente como por medio de scripting y programación. GC está basado en una serie de componentes geométricos y de datos que se conectan entre sí para formar una estructura paramétrica que controla el diseño.

procedural y orientado a objetos. Su sintaxis está basada en C++ e incluye una serie de aspectos de otros lenguajes de programación como Java y Delphi.

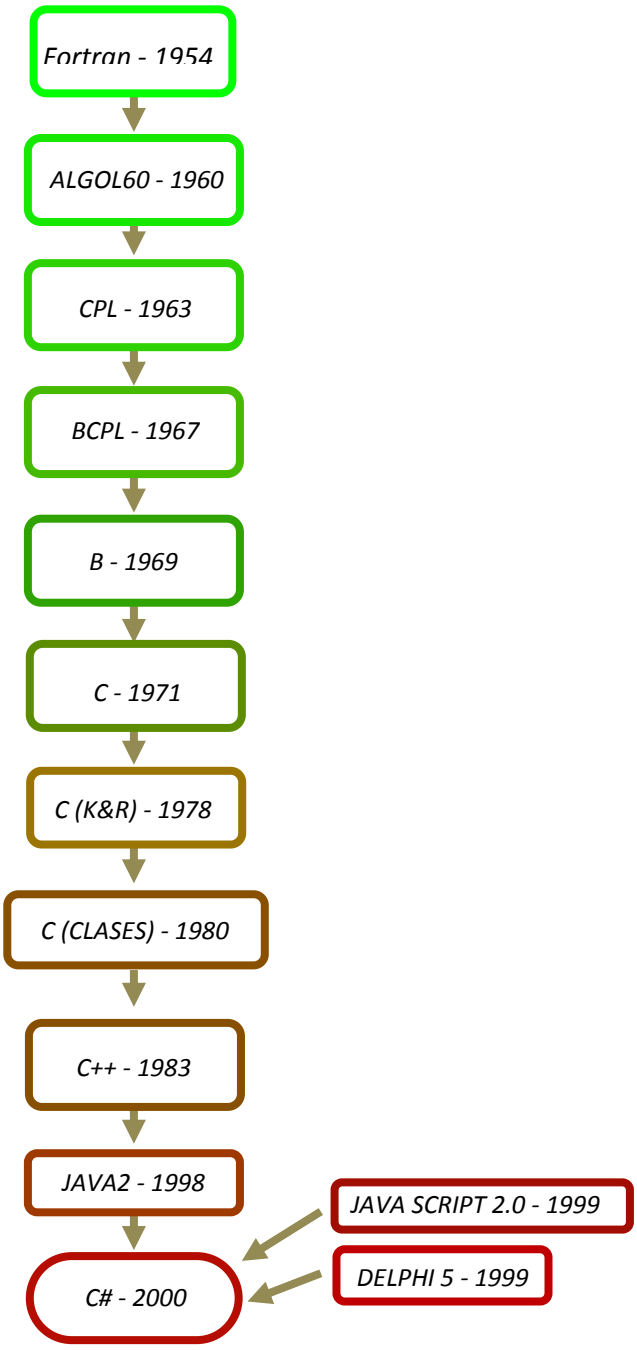


Figura 67. Esquema basado en http://www.levenez.com/lang/, [09/07/08].

7.2. Software de Análisis

En este apartado se incluye una lista de software de análisis térmico y solar que fueron estudiados para ser utilizados en los proyectos del capítulo 8. El estudio y análisis de los softwares fue realizado el año 2008, por lo que mejoras y nuevas características pueden haberse incluido con el tiempo.

7.2.1. *Ecotect*

Referencia: http://www.*Ecotect*.com/, [02/03/08].
http://usa.autodesk.com/adsk/servlet/pc/index?id=12602821&siteID=123112, [24/08/10]

Ecotect es un software para el diseño de edificios y análisis de ambientes que cubre un amplio rango de simulaciones requerido para entender el comportamiento del edificio en su emplazamiento. Este software ofrece modelado, visualización y análisis.

ANÁLISIS SOLAR

Ecotect puede calcular y visualizar la incidencia solar en ventanas y superficies. El análisis puede ser hecho sobre cualquier superficie o sección del modelo, mostrando la diferencia de incidencia solar durante cualquier período. La disposición del resultado puede permitir tratar las diferentes partes de la fachada de la manera más óptima. *Ecotect* permite comparar la ganancia solar durante diferentes temporadas y mostrando las variaciones de los consumos de calefacción o refrigeración disponibles en el edificio. Además es posible calcular máscaras de sombra para las superficies, combinando datos de iluminación directa e indirecta durante diferentes momentos del año.

El programa permite el cálculo de luz natural o artificial dentro de un recinto. Por medio de una malla o a través de puntos ubicados dentro del modelo. Si es necesario, se puede utilizar una secuencia de comandos para iterar incrementando la altura del edificio, comprobando continuamente la entrada de luz de cada ventana hasta que se encuentre el entorno más óptimo compatible con el sitio donde se encuentra.

ANÁLISIS TÉRMICO

Ecotect usa CIBSE (*Chartered Institution of Building Services Engineers*)[69], un motor de cálculo para calefacción y refrigeración para analizar modelos con diferentes zonas o tipos de geometría. Se pueden asignar materiales detallados para todos los objetos así como el horario destino de uso, ganancias internas e infiltraciones.

[69] http://www.cibse.org/ [02/03/08]

En adición a las temperaturas internas y dibujo de gráficos es posible calcular dentro de la distribución espacial la temperatura media dentro de un modelo y predecir los niveles de confort. Si se especifica la actividad y el nivel de vestimenta es posible elaborar la distribución necesaria de los porcentajes de aire para determinar las necesidades de ventilación. Las temperaturas internas pueden ser mostradas para cualquier hora y día del año. Los gráficos incluyen temperaturas externas, radiación y efectos de viento. Permitiendo una apreciación completa de la respuesta térmica para cada zona.

VENTILACIÓN Y FLUJO DE AIRE

Ecotect permite generar la geometría y la malla de análisis para exportar directamente a algún programa tipo CFD (*Computing Fluid Dynamics*)[70]. Después de que el análisis en estas herramientas ha concluido, es posible importar los resultados de regreso a *Ecotect* para mostrarlos en el contexto del modelo original.

Utilizando datos meteorológicos, *Ecotect* puede superponer la velocidad del viento y la dirección sobre el modelo, por lo que es especialmente relevante para entender la ventilación natural. También es posible mostrar la temperatura, la humedad y las precipitaciones, a lo largo de todo el año y en diferentes rangos de tiempo. *Ecotect* incorpora un lenguaje de programación para automatizar tareas que es el denominado *Lua*[71].Este lenguaje de programación es imperativo y estructurado, bastante ligero que fue diseñado como lenguaje de script con una semántica extensible.

La última versión de *Ecotect* se puede descargar como versión de prueba de 30 días, hasta el 15 de agosto del 2008. El programa por el momento no se encuentra a la venta, por la reciente compra por parte de Autodesk. *Ecotect* tiene una gran cantidad de recursos, tanto tutoriales y blogs abarcando las áreas de diseño y scripting.

7.2.2. EnergyPlus (E+)

Referencia: http://apps1.eere.energy.gov/buildings/energyplus/, [09/07/08].

EnergyPlus (E+) tiene sus raíces en dos programas el DOE-2 y BLAST72 ambos desarrollados en los años 1970 y 1980 como herramientas de carga y de simulación energética. E+ es un programa de análisis de carga térmica y simulación, capaz de calcular calefacción y cargas de refrigeración necesaria para mantener el control térmico, las condiciones del sistema HVAC[73] (Heating, Ventilating, and Air Conditioning) y el consumo de energía, así como muchos otros detalles de simulación que son necesarios para verificar el correcto análisis del edificio. Este programa está diseñado para ingenieros o arquitectos que deseen conocer el tamaño apropiado del equipo HVAC. Y así desarrollar estudios sobre el análisis de costos del edificio u optimizar la eficiencia energética. *EnergyPlus* es un programa autónomo de simulación, no tiene una interfaz gráfica amable, sin embargo puede leer y escribir datos de entradas y de

70 http://www.cfd-online.com/ [02/03/08]
71 http://es.wikipedia.org/wiki/Lua [09 / 07 / 08]
72 BLAST significa: Building Loads Analysis and System Thermodynamics.
73 http://www.hvachome.net/ [09/07/08]

salida como archivos de texto, lo que le otorga una gran flexibilidad a la hora de intercambiar datos con otro programa.

ANÁLISIS SOLAR

E+ incluye cálculos de iluminación interior, control y simulación de deslumbramiento, control de iluminación, y el efecto de reducir el encendido de luz artificial, calefacción y refrigeración. EnergyPlus es capaz de realizar cálculos para controlar el tamaño de ventanas y persianas, vidrios electro-crómicos, balances de calor por capa que permitan la asignación adecuada de la energía solar absorbida por los cristales de ventanas y la incorporación de numerosas bibliotecas disponible comercialmente. Además el modelo ansiotrópico del cielo permite mejorar el modelo de cálculo solar difuso sobre superficies inclinadas.

ANÁLISIS TÉRMICO

El balance térmico se basa en una solución técnica para la construcción de cargas térmicas. Permite el cálculo simultáneo de radiación y convección en el interior y exterior de la superficie. Se puede calcular la transferencia calórica a través de la construcción de elementos tales como paredes, techos, pisos, etc. El cálculo puede ser realizado capa por capa integrando la transferencia de conducción o como una mezcla de profundidad de penetración. Se puede crear un modelo de confort termal basado en la actividad interior del recinto, humedad, etc.

VENTILACIÓN Y FLUJO DE AIRE

EnergyPlus permite calcular flujos de aire dentro y fuera del edificio, para obtener datos. El bucle configurado para sistemas HVAC (convección y radiación) permite a los usuarios modelar sistemas de calefacción y refrigeración.

7.2.3. DesignBuilder

Referencia: http://www.designbuilder.es/, [09/07/08].

DesignBuilder ha sido desarrollado en torno a EnergyPlus como una herramienta para hacer más amable la visualización de los resultados. Permite introducir todos los materiales y cristalería que posee EnergyPlus, asi como su base de datos de materiales, construcción, paneles de vidrios, cámaras aislantes, unidades de cristales y paneles llenos.

Sus características lo describen como un modelador de sólidos basado en librerías OpenGL. El cual permite modelar edificios que sean posicionados como “bloques” en el espacio. Se pueden implementar realistas vistas 3D definiendo espesor de muros y cristales, áreas de habitaciones y volúmenes.

Se puede importar la geometría a CAD y no existe limitación en formas de superficie, con respecto a esto, las superficies con más de 4 vértices son trianguladas para garantizar la compatibilidad con el simulador EnergyPlus.

Las plantillas internas del programa permiten cargar los datos más comunes relacionados con la construcción como tipos de actividades, iluminación exterior, aire acondicionado y sistemas de calefacción, tipos de materiales, etc. DesignBuilder es probablemente uno de las interfaces de EnergyPlus más poderosas en el mercado.

7.2.4. Hourly Load Calculation Program

Referencia: http://www.ishrae.in/hlcp/, [09/07/08].

Hourly Load Calculation Program (HLCP) ha sido desarrollado por la sociedad de ingenieria india de calefacción y aire-acondicionado (*ISHRAE, The Indian Society of Heating, Refrigeration and Air-Conditioning Engineers*), específicamente para encontrar las necesidades locales de HVAC para el cálculo de cargas de los edificios. Los resultados incluyen las cargas diarias, 288 horas anuales y 8760 horas de salida a través de un único y simplificado interfaz gráfico. Es posible cargar los datos de 58 ciudades Indias disponibles. El programa se estructura de tal manera que facilita el flujo de datos hacia hojas de cálculo. Además es capaz de servir como un complemento para todo edificio analizado en *EnergyPlus*.

7.2.5. Cymap

Referencia: http://www.cymap.com/, [09/07/08].

Este software otorga soluciones para HVAC y diseño eléctrico. Está especializado para proyectos comerciales, y para el acondicionamiento de nuevos proyectos y modernización de edificios existentes. Permite comparar alternativas de calefacción - refrigeración y estrategias de diseño de fachada, diseño y calibrado de equipos y demanda de energía.

El programa Cymap presenta soluciones para:

- Calefacción, refrigeración consumo y análisis psicométricos.
- Tuberías y un amplio rango de fluidos y configuraciones.
- Servicios de conductos, niveles de ruido en habitaciones.
- cableados de baja tensión, diseño de los transformadores hasta los circuitos finales.
- Diseño de Iluminación interior, de emergencia, luz natural y alumbrado público.

Al parecer es fácil de aprender y se puede utilizar en todas las etapas del proceso de diseño. Desde el modelo inicial a través del sistema de diseño, se pueden calcular costos y presupuesto. Permite el diseño de tuberías y conductos.

7.2.6. Energy-10

Referencia: http://www.nrel.gov/buildings/energy10.html, [09/07/08].

ENERGY-10, es una herramienta de software que ayuda a arquitectos, constructores e ingenieros a identificar rápidamente las más rentables, medidas de ahorro de energía para tomar decisiones en el diseño y un bajo consumo de energía del edificio. El software de

simulación es adecuado para el examen de los pequeños edificios comerciales y residenciales que se caracterizan por uno, o dos zonas térmicas (generalmente menos de 3000 m2). Es un programa doble función. Hace una estimación de carga hora por hora y calibrado de sistemas HVAC calculando los costos de funcionamiento. La carga térmica se calcula mediante ASHRAE. Es capaz de evaluar el funcionamiento de una gran variedad de equipos de aire y plantas de energía. Es posible calcular los costes de explotación sobre la base del uso de la energía y la demanda de cargas. Los resultados se muestran mediante cuadros y gráficos.

Programa	*Lenguaje*	*Comentario*
Ecotect	*Lua (script)*	*Permite el diseño de edificios y análisis de ambientes.* *El cálculo y análisis está basado sobre modelos, lo que implica importar el modelo en forma de geometría dentro del programa.*
EnergyPlus	*Fortran95*	*Puede leer y escribir datos de entradas y salida como archivos de texto.* *Completo calculo térmico.* *Es un programa autónomo de simulación sin una interfaz gráfica.* *Es el motor de cálculo de muchas aplicaciones para el análisis.* *Es gratis.*
DesignBuilder	*no*	*Basado en EnergyPlus. No permite su programación y personalización.*
HLCP	*no*	*Basado en EnergyPlus.*
Cymap	*Visual Basic*	*Tiene módulos mecánicos y eléctricos. Calcula todos los aspectos de HVAC. Proyectos de todas las escalas.* *Puede funcionar tipo stand-alone o en compatibilidad con algún software CAD.*
Energy 10	*no*	*Adecuado para el análisis de pequeños edificios comerciales y residenciales.* *Cálculos de ganancia y pérdida energética.* *Programa desarrollado por la agencia de medio ambiente de estados unidos.* *Basado en EnergyPlus*

Figura 68. Tabla comparativa de programas de análisis térmico.

7.3. Programas de Modelado

7.3.1. *Rhinoceros* (*Rhino*)

Referencia: http://www.*Rhino*3d.com/, [09/07/08].

Rhinoceros es un software de modelado de superficies Nurbs, lo que le otorga una flexibilidad en el modelado de formas libres frente a otros programas CAD especializados en objetos sólidos o mallas poligonales mucho más rígidas. Aunque también acepta mallas poligonales y sólidos. Con *Rhino* es posible editar, analizar, documentar y animar todo tipo de geometrías. Es de muy rápido aprendizaje y muy intuitivo. La estrategia de McNeel, empresa que desarrolla *Rhino*, ha puesto a disposición del público de manera libre y gratuita todas las librerías de geometría y *renderizado* con las que han construido el programa. Esto ha hecho que gran cantidad de desarrolladores y empresas utilicen *Rhino* como plataforma para desarrollar sus aplicaciones, gratuitas o comerciales. Sumado a esto la facilidad de uso del programa ha vuelto a este software muy popular, sobre todo en escuelas de arquitectura, diseño y animación.

Actualmente al utilizar *Rhino* se pueden crear 5 tipos de aplicaciones.

- Utilidad general, para propósitos generales este *plug-in* puede contener uno o más comandos.
- Importar archivos, *plug-in* capaz de importar datos de otros tipos de archivos dentro de *Rhino*. Una sola importación de archivos a través de un *plug-in* puede soportar más de un tipo de archivo.
- Exportar archivos, un *plug-in* que puede exportar datos de *Rhino* a otros archivos de formato, un solo *plug-in* puede exportar más de un tipo de archivo.
- Renderizado tipo, un *plug-in* capaz de soportar aplicación de materiales, texturas y luces dentro de la escena para producir renderizado de imágenes.
- digitalización 3D, un *plug-in* capaz de reconocer dispositivos de digitalización 3D.

7.3.2. *GenerativeComponents* (GC)

Referencia: http://www.bentley.com/en-US/Products/*GenerativeComponents*/, [25/08/10].

GenerativeComponents es un sistema de modelado paramétrico y asociativo usado por arquitectos e ingenieros para automatizar los procesos de diseño y acelerar las iteraciones dentro del proceso de producción. Es un producto de *Bentley*. *GenerativeComponents* extiende y provee las tecnologías y entrega significantes avances a usuarios que rápidamente exploran un amplio rango de alternativas de diseño.

Con *GC* se puede modelar geometría y capturar las relaciones entre ellas. Generar formas usando scripts y-o manipularlas directamente por medio de botones modificando sus parámetros. El programa permite combinar iteraciones, flexibilidad en el modelado, y automatizar procesos.

GenerativeComponents es una herramienta de diseño paramétrica y asociativa y puede aplicarse en cualquiera fase de diseño. Usa un avanzado motor paramétrico para unificar todos los aspectos del diseño. Permite construir modelos geométricos a partir de objetos y las relaciones entre dichos objetos. Dichos objetos pueden ser tan simples como una línea o tan complejos como una superficie de doble curvatura. Editando la geometría de los objetos y las relaciones entre ellos se pueden explorar diversas alternativas de diseño. Los cambios en las propiedades de los objetos se propagan de forma automática a todo el modelo. Esto permite al diseñador explorar diversas alternativas (de diseño) sin tener que reconstruir manualmente cada vez el modelo.

GenerativeComponents (GC) puede ser visto como una caja de herramientas que en la cual existen una serie de objetos, como puntos, polígonos, curvas, superficies, hojas de cálculo, cables para conectarse con otro software, Textos, Vectores, Elipses, Conos y Solidos, Planos, Círculos, Sistemas de Coordenadas, etc. etc. La ventaja está, en que todos estos objetos se pueden manipular por medio de programación. De esta manera se puede crear de dos maneras diferentes: La primera, por medio de botones y la segunda, escribiendo códigos. Todos estos objetos, (círculos, planos, etc.), tienen características comunes y propias. Están diseñadas para agruparse y formar objetos más complejos. Cada uno de ellos tiene dos estados, un estado de propiedades y un estado de comportamiento.

Las geometrías dentro de GC pueden ser manipuladas mediante botones gráficos, scripts usando *GCSCript* y por medio de .NET, principalmente C#. Dentro de GC todo es programación, esto permite que el programa sea capaz de auto documentar toda la geometría generada y ser capaz revisar y modificar el modelo en todas las etapas del proceso de diseño.

7.3.3. Digital Project (DP)

Referencia: http://www.gehrytechnologies.com/, [25/08/10]

Digital Project es una suite de modelado 3D y BIM, basado en CATIA de Dassault Systemes[74]. El programa permite integrar modelado de arquitectura, estructura e instalaciones y además productos adicionales como: Primavera (para manejo de interoperabilidad), módulos de optimización, renderizado, maquetación de planos, entre otros. DP es un programa para proporcionar ayuda desde la idea del diseño CAD (*Computer Aided Design*), para la producción CAM (*Computer Aided Manufacturing*) y el análisis CAE (*Computer-aided engineering*).

Digital Project permite la cinemática, el cálculo estructural y probar el comportamiento de materiales. También permite calcular colisiones dentro del modelo. Su estructura le permite trabajar con grandes volúmenes de datos en un mismo archivo. Incluyendo desde el tornillo al modelo global de instalaciones, estructura y arquitectura.

Inicialmente CATIA fue desarrollado para servir en la industria aeronáutica, con un control total en el manejo de geometrías complejas. Es ampliamente usado en la industria del automóvil y en el diseño de componentes y maquinaria en grandes empresas como Volkswagen, Seat, BMW, AirBus, Reanult, Porsche etc.

74 http://www.3ds.com/products/catia [25/08/10]

Tanto Digital Project como CATIA permiten su programación mediante Visual Basic y C++. Sin embargo la preparación de los archivos es una tarea algo tediosa.

7.3.4. Maya

Referencia: http://www.autodesk.es/adsk/servlet/pc/index?siteID=455755&id=14627356, [25/08/10].

Maya es un software de modelado 3D, animación y efectos especiales, destinado a la industria del cine. Es un software bastante robusto basado en C++ que permite ser personalizado por medio de su lenguaje de *script* MEL (*Maya Embedded Language*) y por medio de su API (*Application Programming Interface*).

El programa se caracteriza por su potencia y diversas herramientas como, modelado, animación, render, simulación de ropa, cabello, simulación de fluidos y partículas. Maya trabaja con cualquier tipo de superficies Nurbs, Polygonos, Curvas. Además permite la posibilidad de convertir entre todos los tipos de geometría.

La característica más importante de Maya es la flexibilidad que permite el software para ser personalizado, Este aspecto ha hecho que Maya sea usado por los estudios de cine y animación que tienden a escribir mucho código personalizado para su producción.

Programas	*Lenguaje*	*Comentarios*
Rhinoceros	*macros* *Rhinoscript* *.NET*	*Comunidad en internet grande* *Soporte y gran cantidad de aplicaciones* *Tiene algunos fallos. Para ser paramétrico necesita de otras aplicaciones*
Generative Components	*C#,* *Visual Basic.NET*	*Sistema paramétrico y asociativo* *No existe una gran comunidad de usuarios de GC* *Programación en diferentes niveles: grafica, GCScript, .NET* *Programa muy sólido y robusto.*
DigitalProject	*Visual Basic* *C++*	*Basado en CATIA, robusto y complejo de usar.* *Plataforma BIM. (paramétrico y asociativo)*
Maya	*MEL* *Python* *C++*	*Modelado, animación* *Formas libres, gran potencia en diseño y programación.* *Editor de Texto pobre.*

Figura 69. Tabla comparativa de diferentes programas de modelado.

7.4. Conclusiones de Elección de Herramientas

PROGRAMACIÓN

La programación es un tipo de lenguaje que no acepta mentiras, ni falsedades, ni ambigüedades. Su objetivo consiste en despejar una incógnita, que hasta ahora siempre se ha encontrado, y que se seguirá encontrando.

La programación implica composición. Cuando decoramos nuestra casa, cuando nos vestimos o incluso cuando hablamos componemos, de diferentes maneras y usando diferentes materiales. Sin embargo lo que tiene en común decorar nuestra casa y hablar es que ambos tipos de composición (así como todos) están basados en procesos. El *Lenguaje de programación* (el código binario) es un tipo de composición que ordena, que compone procesos, que darán un resultado inesperado o no.

El código se hace cada vez más universal, diferentes disciplinas lo están incluyendo para la investigación práctica y teórica y una de estas es la arquitectura. Las consecuencias por el momento aún no son comprensibles. Sin embargo comienzan a desarrollarse nuevas herramientas para poder entenderla, como por ejemplo los nuevos procesos de fabricación. Pero de los que si se está seguro es que la programación es una habilidad y las habilidades se dominan con la práctica.

ELECCIÓN DEL SOFTWARE

La elección del software es una tarea compleja y larga, en el sentido en que no se puede comprobar si funciona hasta que se ha probado en las situaciones que interesa. Para probar un software y luego llegar a programar a nivel de script o a nivel de programación es necesario primero que todo, conseguir el software. Existen diferentes maneras de conseguirlo, pero lo más seguro y confiable es probar una evaluación de 30 días o más, porque se cuenta con soporte y acceso a foros. El aprendizaje del software es necesario y debe ser paulatino comenzando con ejercicios simples y subiendo en complejidad. La clave de esto, aunque parezca trivial es entender la lógica del programa y cómo se comportan sus elementos. Por ejemplo en *Rhinoceros* un punto en el espacio de coordenadas es una lista de 3 valores reales, para X, Y, Z. En *GenerativeComponents* en cambio, un punto es considerado un objeto, lo que significa que además de tener los tres valores reales en sus coordenadas, tiene propiedades como magnitud, peso, color, capa, estado, etc. toda esta información es útil porque en el momento en que se comienza a desarrollar los códigos necesitaremos saber con qué tipo de geometría estamos lidiando. La parte buena es que no necesitamos conocer el programa a fondo para comenzar a programar dentro de él.

CRITERIOS DE SELECCIÓN

El criterio para elegir el software corresponde a unos objetivos a encontrar en el capítulo 8. Estos consistían en desarrollar una serie de AGs que pudiesen ser aplicados en diferentes escalas del proyecto arquitectónico. Un AG de “macro escala” aplicado a la ordenación de edificios. Un AG de “media escala” aplicado a la generación de formas. Un AG de “micro

escala" aplicado a la ordenación de espacios y circulaciones dentro de la planta de un edificio. Finalmente un AG capaz de optimizar la geometría de un edificio para reducir los consumos de calefacción y refrigeración el día más caliente y más frio del año. Los primeros 3 proyectos tendrían una evaluación geométrica, lo que implicaba desarrollar todo el AG dentro de un solo software. El cuarto proyecto técnicamente era más complejo, en el sentido que era preciso evaluar térmicamente un edificio (una forma) y para esto era necesario contar con un programa lo suficientemente robusto y serio (cálculos confiables) para obtener resultados precisos.

ELECCIÓN DEL SOFTWARE DE ANÁLISIS

Los softwares de análisis térmico que se probaron fueron, *Ecotect*, DesignBuilder y ENERGYPLUS. Dentro de *Ecotect* se hicieron varias pruebas y diferentes análisis para aprender a manejar el programa. El aspecto positivo que se encontró dentro de *Ecotect* era la manera en como mostraba la información y la simplicidad de uso. La parte negativa del software consistía en que la importación del archivo debía ser hecho por medio de un archivo DXF o 3DS de manera manual y además no existía una API para poder extraer la información del software. Es por esto que *Ecotect*[75] fue descartado. El software *DesignBuilder* por su parte es aún más hermético que *Ecotect*, por lo que también fue descartado.

Finalmente se optó por *EnergyPlus*, ya que su archivo de análisis de extensión IDF, tiene un formato de texto, por lo que su lectura es muy rápida y los cambios pueden realizarse en la memoria del ordenador y no guardando archivos. La ejecución del programa puede ser automatizada, ejecutando un Batch[76]. Otra ventaja de E+ era que al casi no tener interfaz, el programa casi no consume recursos gráficos, haciéndolo muy rápido al abrirse y cerrarse. La desventaja de E+ es su manual y la complejidad al definir los diferentes parámetros del modelo. En este sentido toda la geometría debe ser traducida a coordenadas y debe ser insertada en puntos específicos dentro del archivo IDF.

ELECCIÓN DEL SOFTWARE DE MODELADO

La elección del software de modelado fue más simple. Maya fue descartado porque su editor de texto es muy pobre y cada vez que se ejecuta el código este se pierde en la ventana de resultados, obligando a estar copiando y pegando constantemente. Sin embargo hubiese sido interesante haber desarrollado los 3 primeros experimentos dentro de Maya y con tecnología C++. Digital Project fue descartado porque era demasiado potente y sobre todo porque había

75 Con nuevas versiones de Ecotect, es posible importar y exportar documentos mediante archivos XML, lo que ha permitido su interoperabilidad con diferentes programas, como GenerativeComponents: http://code.google.com/p/ecotect-gc-link/. [26/08/10], y *Rhinoceros* http://utos.blogspot.com/2010/06/geco-gh2ecotect-update.html. [26/08/10].

76 En DOS, OS/2 y *Microsoft* Windows un archivo batch es un archivo de procesamiento por lotes. Se trata de archivos de texto sin formato, guardados con la extensión BAT que contienen un conjunto de comandos MS-DOS. Cuando se ejecuta este archivo, los comandos contenidos son ejecutados en grupo, de forma secuencial, permitiendo automatizar diversas tareas. Cualquier comando MS-DOS puede ser utilizado en un archivo batch. http://es.wikipedia.org/wiki/Batch. [26/08/10].

que aprenderlo desde 0. Es por esto que el software elegido fue *Rhinoceros* y el lenguaje *Rhinoscript*, basado en Visual Basic, del cual el autor tenía conocimientos previos.

Para el ejercicio 4 de la optimización, la elección fue aún más simple. Se optó por *GenerativeComponents* que es paramétrico y asociativo, característica fundamental cuando se optimiza un modelo. La parametrización deja al software que se encargue de todas las relaciones geométricas del modelo y al arquitecto de manipular y modificar todos los valores de los diferentes parámetros. Gracias a esto, gran parte del problema es solucionado. La otra ventaja de GC es que cuenta con una conexión directa con C#.NET, que sería la vía para conectar EnergyPlus y *GenerativeComponents*. La desventaja es que al comienzo de la investigación el autor no había utilizado nunca antes GC. Si bien GC es un programa con una interfaz poca intuitiva, su lógica es muy simple, por lo que el aprendizaje del programa toma a lo sumo un mes. La mejor manera de aprender GC es mediante el libro de Bentley Essentials[77], y el foro GCForum[78].

Este apartado sirvió para conocer el nivel técnico que tenía el autor para desarrollar los proyectos del capítulo 8. Si bien se trabajaron con una serie de programas como *Maya*, *Ecotect*, *Catia*, *Ansys*, *Gambit*, *Gid*, *Fluent*, *Digital Project*, *Revit*, etc. Finalmente, las herramientas usadas para desarrollar los experimentos fueron *Rhinoceros* y *Rhinoscript* para trabajar los 3 primeros AGs y *GenerativeComponents* y *EnergyPlus*, para optimizar el consumo energético.

77HTTP://FTP2.BENTLEY.COM/DIST/COLLATERAL/DOCS/MICROSTATION_GENERATIVECOMPONENTS/MICROSTATION_GC_V8I_ESSENTIALS_BOOK.PDF. [28/06/10].
78 http://communities.bentley.com/forums/360/ShowForum.aspx. [28/06/10].

Experimentos

Para esta investigación se desarrollaron una serie de algoritmos evolutivos genéricos que sirvieron de estudio y base para la implementación de 4 AGs aplicados a la arquitectura. Cada uno de estos proyectos tienen los mismos 4 apartados. Estos son, introducción, método, resultados y discusión. En la introducción, se plantea el problema a resolver y se describe en que consiste el proyecto. En el método se hace una revisión detallada del AG, como funciona, sus diferentes partes, la estrategia usada para programar, etc. En resultados, se muestran los resultados obtenidos durante las diferentes pruebas del AG, ejemplos de generaciones, datos obtenidos y formas generadas mediante los AGs. En la discusión, se hace un análisis del proceso y los resultados obtenidos. También se comparan los modelos generados con otras ideologías y estrategias.

El diseño de los AGs fue orientado a 4 diferentes escalas del proyecto arquitectónico, que están presentados de manera descendente, de mayor a menor escala. Estos 4 proyectos son:

TRAVELLING SALESMAN PROBLEM COMO MODELO GENERATIVO URBANO, MACRO ESCALA

Un AG para resolver el problema *Travelling Sales Problem* (TSP) es desafiado a encontrar el camino más cortó entre una serie de edificios, plantear sus relaciones espaciales y proponer volumetría y orientación de acorde a los edificios más cercanos en la ruta. Este problema busca encontrar la ruta más corta que pase por todos los puntos de un conjunto. Es un problema de combinatoria que consiste en encontrar la combinación de puntos que genere el recorrido más económico dentro de un espacio de búsqueda de Puntos! (factorial de N puntos)[79].

Por cada uno de los puntos por donde debe pasar la ruta, existe un volumen que representa un edificio, este se escala dependiendo de la distancia entre los otros edificios y también se orienta dependiendo del sentido de la ruta. El resultado es una serie de volúmenes relacionados espacialmente entre sí. Por lo tanto, se crean espacios entre ellos, los cuales pueden ser reconocidos como plazas, caminos, rincones, zonas más abiertas y otras más protegidas. Lo interesante es que se puede determinar la ubicación de los edificios

79 Para todo número natural n, se llama n factorial o factorial de n al producto de todos los naturales desde 1 hasta n:

$$n! = 1 \times 2 \times 3 \times 4 \times ... \times (n-1) \times n$$

Por medio de la combinatoria, las factoriales intervienen en el cálculo de las probabilidades. Intervienen también en el ámbito del análisis, en particular a través del desarrollo polinomio de las funciones (fórmula de Tynril). Se generalizan a los reales con la función gamma, de gran importancia en el campo de la aritmética. http://es.wikipedia.org/wiki/Factorial. [28/08/10].

dependiendo de restricciones logísticas, ambientales, geográficas, comerciales, etc. Entonces el AG propondrá la ruta más corta que pase por cada uno de ellos, su relación y orientación entre los volúmenes de los edificios. Por ejemplo, un volumen para educación, comercio, vivienda, hospital, etc. tendrá la ruta más corta que pase por cada uno de ellos y estarán escalados y orientados dependiendo de su vecino más próximo en la ruta.

AG COMO SISTEMA DE DISEÑO GENERATIVO, MEDIA ESCALA

Un AG deberá satisfacer una serie de restricciones espaciales y volumétricas que se oponen entre ellas, sin perjudicar alguna en favor de otra. Este AG propone formas en base a ciertas restricciones. El cromosoma binario (1010110011) representa una serie de cubos, apilados entre sí que forman un paralelepípedo mayor como base para el experimento. Las restricciones que debe cumplir el AG son: Maximizar el volumen, minimizar el área de ocupación de suelo, buscar tiras de 000 o 111 de longitud X dentro del cromosoma (1 representa cubo y 0 vacío). El volumen debe reaccionar a puntos que rechazan la generación de cubos en ciertas zonas de la geometría. Al final, la forma es el resultado de todas estas restricciones satisfechas buscando el óptimo que no perjudique a ninguna sin tener que favorecer a otra.

DISTRIBUCIÓN DE RECINTOS MEDIANTE AG, MICRO ESCALA

Se utiliza un AG para organizar espacialmente un recinto, dependiendo de sus áreas y su ubicación dentro del perímetro del edificio. Se comienza con un recinto vacío delimitado por un polígono. El arquitecto tiene que dividir el espacio en N habitaciones y cada habitación tiene asignarle un área que debe ser cumplida. El arquitecto emplaza estos puntos dentro del perímetro y el AG genera un diagrama de *Voronoi*, el cual es la base geométrica para realizar la optimización. El cromosoma está compuesto por los puntos del *Voronoi* los cuales están asociados a cada uno de los polígonos del diagrama que tienen que cumplir con el área definida por el arquitecto. Al final de la optimización, las áreas han aumentado y reducido de tamaño hasta encontrar con alta precisión las áreas definidas por el arquitecto.

ESTRATEGIA EVOLUTIVA PARA EL DISEÑO DE ARQUITECTURA OPTIMIZADA, OPTIMIZACIÓN

Este proyecto responde la pregunta de cuál es la mejor forma, distribución y tamaño de ventanas que consuma el menor gasto de calefacción y aire acondicionado durante el día más caliente y más frio del año. La propuesta de este trabajo consiste en optimizar un edificio modificando sus aperturas y geometría para reducir los consumos de calefacción y aire acondicionado. La optimización es realizada mediante un *micro*-Algoritmo Genético, (*micro*-AG) programado en C# que es incrustado como una serie de funciones dentro de *GenerativeComponents*. El programa *EnergyPlus* es usado para evaluar los consumos de HVAC (*Heating, Ventilation, Air Conditioning*). El objetivo de la optimización es mantener la temperatura a 20ºC en el día más caliente y más frio del año usando la menor energía posible.

8.1. Travelling Salesman Problem como Modelo Generativo Urbano

8.1.1. Introducción

Un núcleo urbano es un sistema abierto que involucra una cantidad casi infinita de factores, que están en constante balance. Sin embargo, las ciudades adolecen, unas más que otras, de los mismos problemas. Por ejemplo la circulación vehicular, la falta de áreas verdes, la contaminación, el centralismo, la delincuencia, etc. El problema urbano es bastante complejo y además involucra al 78% de la *población*[80] mundial.

Este trabajo no pretende ser una solución al problema de la ciudad contemporánea, sino más bien a una propuesta de generación urbana, basada en un AG para resolver el problema *Travelling Salesman Problem* (TSP).

El problema del dilema del viajante (TSP) es un problema del tipo de optimización combinatoria[81]. Esto significa que la solución consiste en encontrar la mejor combinación de entre una serie de elementos. El problema se define así: dada una cantidad de puntos en el espacio euclidiano, buscar la ruta más corta que pase por cada una de ellos, sin repetirse. Si bien la pregunta es simple el problema es bastante complejo debido a la cantidad de rutas posibles que se pueden comprobar, este es el espacio de búsqueda. El número de combinatorias posibles es el factorial de la cantidad de puntos. Por ejemplo, con 6 puntos existen 720 posibilidades, en cambio sí son 20 puntos (20!) Las posibilidades son 2.432.902.008.176.640.000, un número tan grande que si analizamos cada una de ellas, a la velocidad que tarda un ordenador (un milisegundo), tardaríamos 77.146.816 años en encontrar la mejor solución.

Es por esto que la implementación de un AG a este problema es una muy buena opción para encontrar soluciones aceptables. Este tipo de problema tiene múltiples aplicaciones en la vida cotidiana. Por ejemplo, encontrar la ruta más corta en una ciudad cuando es necesario pasar por varios puntos. Encontrar la ruta más corta en un plano de fabricación para minimizar el tiempo de corte en una CNC. Reducir las longitudes de cables o tuberías en un proyecto de instalaciones. Planear rutas de transporte, distribución de productos o distribuir las circulaciones de servicio dentro de una obra.

Usualmente el recorrido dentro un centro urbano implica pasar por una serie de puntos, antes de regresar al origen. Durante el recorrido usualmente se busca minimizar el tiempo de viaje, haciéndolo rentable y mejorando la calidad de vida.

---

80 Investigación del instituto nacional de estudios demográficos de Francia: más de la mitad de habitantes del mundo ya vive en ciudades, consultado en: http://www.clarin.com/diario/2007/06/22/sociedad/s-03203.htm. [2010/05/25].

81 La optimización combinatoria pertenece al campo de las matemáticas aplicadas y ciencias de la computación. Está relacionada con la teoría de algoritmos como la inteligencia artificial e ingeniería de software. Los algoritmos de optimización combinatoria resuelven los problemas explorando el espacio de soluciones (usualmente grande) reduciendo el tamaño efectivo del espacio, y explorando el espacio de búsqueda eficientemente.

Este proyecto plantea un modelo de generación urbana basado en un AG. El sistema generativo actúa en base a un TSP con el objetivo de disponer los edificios de manera en que la distancia entre ellos sea la menor posible. Luego, dependiendo de la proximidad entre los edificios el AG propone volúmenes y orientaciones para cada uno de ellos, para garantizar un mínimo de asoleamiento y espacio alrededor.

8.1.2. Método

El primer paso del trabajo consiste en definir las zonas urbanas, las cuales serán emplazadas en el terreno. Bajo el criterio del arquitecto cada una de ellas estará representada por un punto en coordenadas (X, Y, Z). Las zonas propuestas pueden ser: Biblioteca, vivienda, comercio, salud, centro deportivo, administración legislativa, administración ejecutiva, educación básica, educación técnica, educación profesional, centro de oficinas, centro de trabajo, transporte, ocio, cultura, industria, almacenamiento, vivienda de alta densidad, mercado y cine. En total son 20 puntos, definidos.

Este algoritmo ha sido escrito en *Rhinoscript* el cual es un lenguaje de programación tipo scripting de alto nivel basado en Visual Basic y que corre sobre el programa *Rhinoceros*.

ESTRUCTURA DEL ALGORITMO

La entrada del AG son los 20 puntos, definidos anteriormente dentro del espacio de trabajo de *Rhinoceros*. Mediante la función *CreateCity()*, se recogen los puntos y estos son guardados dentro de una lista. Esta lista tiene la siguiente estructura

0	*1*	*2*	*…*	*18*	*19*
P1(x, y, z)	*P1(x, y, z)*	*P1(x, y, z)*	*…*	*P19(x, y, z)*	*P20(x, y, z)*

Figura 70: Estructura de la lista de puntos. La lista debe ser vista como una serie de cajas, que comienza con el índice 0 y acaba con el 19 (20 puntos). Dentro de cada caja se encuentran las coordenadas cartesianas de los puntos.

Está estructura será la base del AG. Debido a que el éxito de la optimización depende de la combinatoria de los puntos guardados en la lista. Dentro de la terminología de la inteligencia artificial, a está lista suele llamársele *individuo* y al conjunto de ellos *población*.

El siguiente paso del sistema es preguntarle al arquitecto por la configuración del algoritmo. En este punto el programa muestra la siguiente ventana:

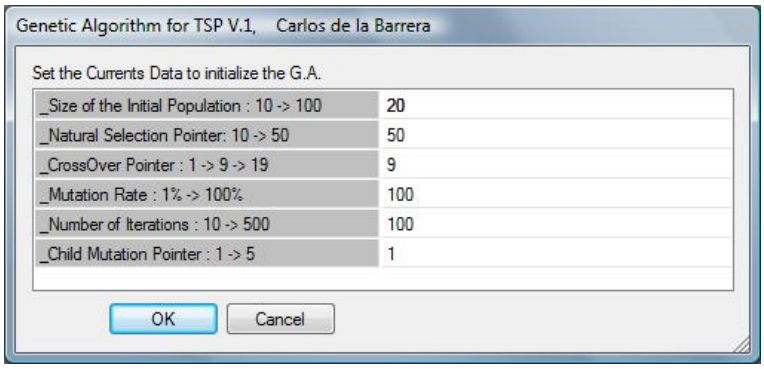


Figura 71: Impresión de pantalla de la ventana de configuración del algoritmo.

El programa propone valores por defecto, aunque si bien pueden modificarse, no deben salirse de los márgenes establecidos en la columna de la izquierda. El primer dato a considerar es el tamaño de la *población* (*Size of the Initial Population*), que define cuantos *individuos* habrá por generación. El siguiente campo, *Natural Selection Pointer*, corresponde al número de veces que el mejor *individuo* será copiado dentro de la selección natural. El campo *CrossOver Pointer*, determina en qué punto del cromosoma se realizará el cruce entre los *individuos* seleccionados. *Mutation Rate* es el porcentaje de mutación que se aplicará cuando se genere la primera *población* de *individuos*. El siguiente campo corresponde al número de iteraciones o la cantidad de veces que el sistema se ejecutará. El campo *Child Mutation Pointer* determina cuantos alelos, dentro del cromosoma, serán mutados.

Una vez definidos estos datos y los puntos guardados en la memoria del ordenador, se ejecuta la función *mutation()*, que desordena la lista original de puntos. *Mutation()* es llamada un total de 20 veces, el mismo tamaño de la *población*.

```
For i = 0 To V(0) - 1
       ReDim Preserve InitPopulation(i)
       InitPopulation(i) = mutation(cityData)
Next
```

Cada vez que la lista es desordenada, un nuevo *individuo* es creado y este es guardado dentro de una lista superior llamada *InitPopulation*. Esta primera generación ha sido realizada completamente al azar, desordenando en un 100% la lista original. El siguiente paso consiste en construir la primera ruta entre los puntos del *individuo* y los volúmenes que representarán los diferentes volúmenes de edificios. La ruta está representada por una *polilínea* y los volúmenes mediante *polisuperficies*. Cada medida de los volúmenes depende de su distancia con el volumen anterior dentro de la lista. Para la construcción de la primera geometría no se hace ningún tipo de evaluación, sólo se selecciona el primer *individuo* de la *población* y se genera su forma.

```
Dim OriginCrv
OriginCrv = Rhino.AddPolyline(InitPopulation(0))
boxes = myfuncboxes(OriginCrv)
```

Luego la función *BestRecord()*, es encargada de encontrar la ruta más corta en la *población InitPopulation*. La siguiente función en ser llamada es *NaturalSelection()*, la cual consta de una serie de pasos. El primero de ellos es ordenar la *población* del mejor evaluado (la ruta más corta) al peor.

```
Function NaturalSelection(ByVal Pop, bestInd, pointer)

       Pop = SortPopulation(Pop)

       Dim i, j, counter: counter = 0
       Dim dist, index
       Dim RouletteWheel()
       Dim d : d = 0

       For i = 0 To UBound(Pop)

              dist = EuclideanDist(Pop(i))
              index = Int((bestInd / dist * (pointer / (i + 1))) - displace)

              For j = 0 To index

                     ReDim Preserve RouletteWheel(counter)
                     RouletteWheel(counter) = Pop(i)
                     counter = counter + 1
              Next

       Next

       NaturalSelection = RouletteWheel

End Function
```

El siguiente paso fundamental dentro de esta función consiste en aplicar el método de selección de ruleta. Cada *individuo* será seleccionado un número de veces, dependiendo de su aptitud. Los *individuos* más aptos serán copiados más veces que los menos aptos. El *individuo* con la ruta más corta será seleccionado como máximo 50 veces. Esta cantidad es definida en el campo *Natural Selection Pointer*, en la ventana de configuración del algoritmo. El resto de los *individuos* será seleccionado, dependiendo de una curva no polinómica descrita en la ecuación:

$$\text{Index} = \text{Int}\left(\left[\frac{\text{BestInd}}{dist}\ x\ \frac{Pointer}{i+1}\right] - displace\right)$$

La variable *BestInd* es la ruta más corta encontrada en la *población* (El mejor *fitness*). Esta es dividida por *dist* que es el *fitness* (la longitud) del *individuo* que se está evaluando en ese momento. La variable *pointer* corresponde al campo *Natural Selection Pointer* de la ventana de configuración y es la máxima cantidad de veces que un *individuo* será copiado dentro de la ruleta. *i + 1* es necesario para recorrer toda la *población*. La variable *displace* lo que hace es desplazar la curva en coordenadas Y. El motivo de esto es poder ajustar un umbral de selección y hacer el sistema más selectivo si es necesario.

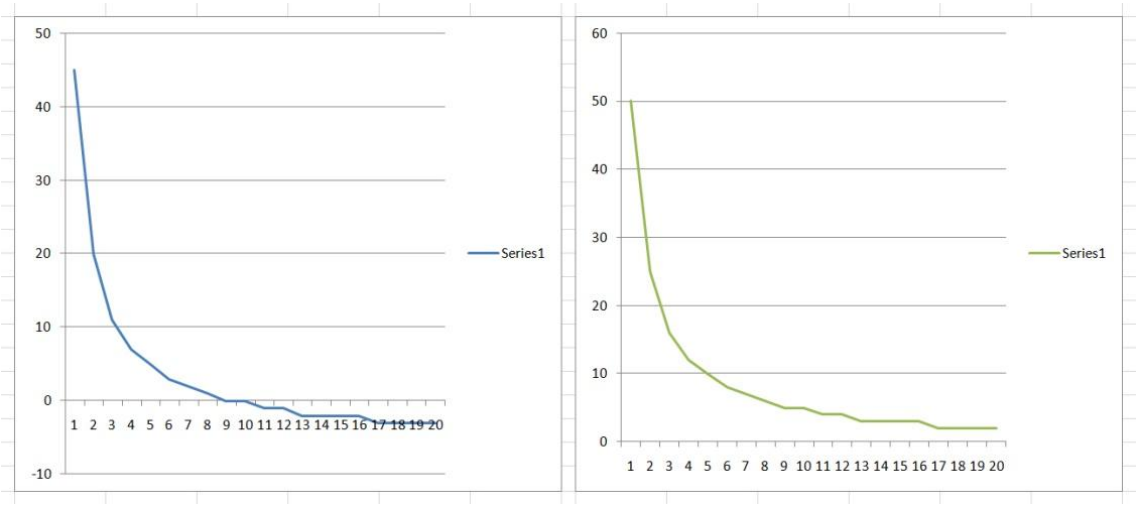


Figura 72. Grafica de la ecuación de la curva no polinómica con diferentes valores en *displace*. Curva usada para la selección de ruleta. En el eje de las X, la cantidad de *individuos* dentro de la *población*, En el eje de las Y la cantidad de veces que el *individuo* será seleccionado.

La curva azul tiene un *displace* = 5, lo que hace que sólo se seleccionen los 10 primeros *individuos*. A partir del *individuo* 11, los resultados de la ecuación son negativos, lo que implica que el bucle no se ejecutará ninguna vez y por lo tanto no se seleccionaran aquellos *individuos*. La curva verde tiene un *displace* = 0. Todos los *individuos* serán seleccionados al menos una vez. El más apto será seleccionado 50 veces y el peor 2 veces.

Un valor bajo en la variable pointer y un *displace* = 0, dibujara una curva con tendencia a ser plana. Esto implica que el más apto *individuo* no tendrá muchas más opciones de ser seleccionado que el menos.

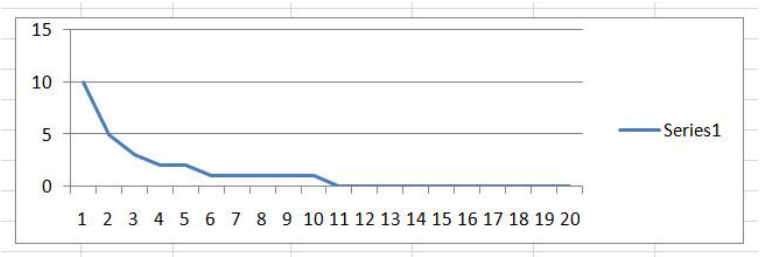


Figura 73: Imagen de la curva con pointer = 10 y *displace* = 0.

La función *NaturalSelection()* retornará la lista *RWheel* con los *individuos* seleccionados. El *individuo* con el mejor *fitness*, (la ruta más corta) estará repetido 50 veces, el segundo mejor estará repetido 25 veces, el tercero 16 y así sucesivamente hasta los últimos *individuos* sólo estarán repetidos 2 veces.

La siguiente función en ser llamada es *Generation()* y se ejecutará un total de 20 veces. El propósito de esta función es remplazar la antigua *población*, por una nueva.

```
Function Generation(ByVal Pool, V)

	Dim PoDad, PoMom, i, j
	Randomize
	PoDad = Int(Rnd * UBound(Pool))
	PoMom = Int(Rnd * UBound(Pool))

	Dim dad, mom, child '//family
	dad = Pool(PoDad)
	mom = Pool(PoMom)

	child = CrossOver(dad, mom, V(2))
	child = MutationChild(child, V(5))

	'-------------------0    1     2
	Generation = Array(dad, mom, child)

End Function
```

Lo primero que sucede dentro de la función, es elegir dos padres de la lista *RWheel*. Estos padres son elegidos de manera aleatoria dentro de toda la ruleta. Tendrán más opciones de ser elegidos los *individuos* más veces repetidos (con mejor *fitness*) dentro de la lista que los menos. Una vez que ambos son elegidos, se llama a la función *CrossOver()*, la cual será la encargada de realizar el cruce y retornar un descendiente, el que es llamado *child*.

```
Function CrossOver(ByVal parent1, ByVal parent2, cutter)


	Dim child()
	Dim i, j, c : c = 0

	For i = 0 To cutter
		ReDim Preserve child(i)
		child(i) = parent2(i)
	Next

	Dim T: T = UBound(child) + 1
	For i = 0 To UBound(parent1)
		For j = cutter + 1 To UBound(parent2)

			If(parent1(i)(0) = parent2(j)(0)) And
	  (parent1(i)(1) = parent2(j)(1)) Then
				ReDim Preserve child(T)
				child(T) = parent1(i)
				T = T + 1
			End If
		Next
	Next

	CrossOver = Child

End Function
```

La función toma ambos *individuos* y los corta por el indicie de la variable *cutter*, que corresponde al campo *CrossOver Pointer* de la ventana de configuración. El hijo es creado como una lista vacía del tamaño de uno de los padres (20 alelos). El hijo hereda del padre los alelos del 0 hasta *cutter* y el resto de la madre. Sin embargo es necesario comprobar que los

genes de la madre no se repitan en el hijo para evitar resultados erróneos. Por ejemplo que durante el cruce se copien dos veces el mismo punto.

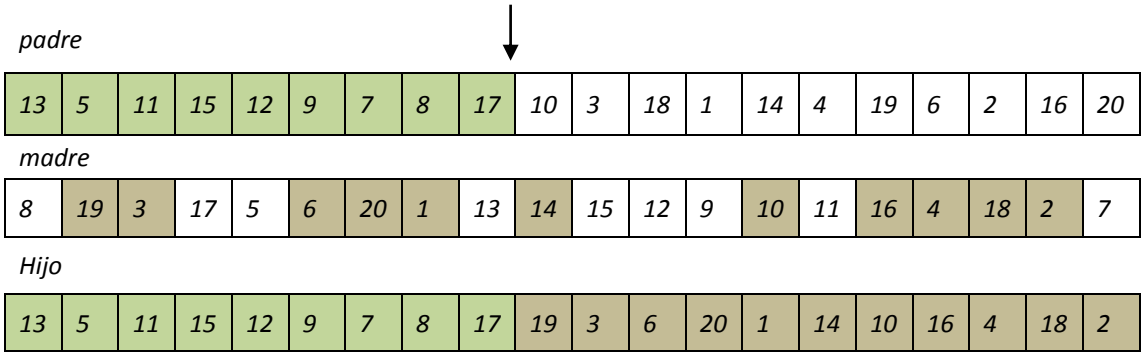


Figura 74. Ejemplo gráfico del operador de cruce.

Los alelos de la madre, antes de ser copiados al hijo, deben ser comprobados si fueron copiados previamente desde el padre. Es por esto que la madre hereda al hijo de forma discontinua.

Una vez que el hijo es creado la función termina y retorna una nueva ruta con un orden de puntos heredados de dos soluciones. La idea del cruce en la selección es que los padres hereden a sus hijos sus características. De esta manera padres con una buena aptitud para resolver el problema heredaran a sus hijos similares características. La siguiente función en ser llamada es *MutationChild()*, encargada de producir una pequeña mutación en el hijo o descendencia. La mutación se realiza para explorar todo el espacio de búsqueda y no centrarse en una zona determinada.

```
Function mutationChild(ByVal childM)

       Dim P, PC, temp, Down, Up

       Down = LBound(childM)
       Up = UBound(childM)

       Randomize
       P = Int(Down + Rnd() * (Up + Down))

       Do

              PC = Int(Down + Rnd() * (Up + Down))
              If (P <> PC) Then Exit Do

       Loop

       temp = childM(PC)
       childM(PC) = childM(P)
       childM(P) = temp

       mutationChild = childM

End Function
```

El orden de puntos es cambiado dependiendo de las veces que sea llamada esta función. Se elige un alelo al azar dentro del cromosoma del *individuo* y se intercambia por otro, a modo de producir una pequeña mutación en la descendencia del cruce.

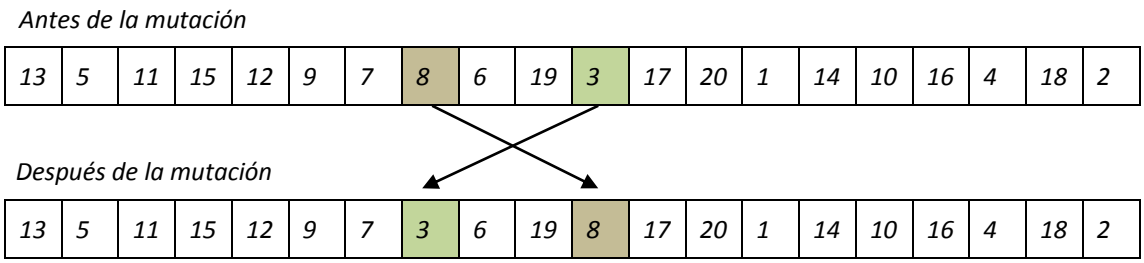


Figura 75. Ejemplo gráfico del operador de mutación. Este consiste en intercambiar el orden de los puntos.

Una vez que la mutación ha sido ejecutada, el nuevo descendiente es retornado dentro de la función *Generation()*, y está a su vez retornara 3 *individuos*. El padre, la madre y el hijo con una pequeña mutación.

Está función será llamada un total de 20 veces con el objetivo de guardar la nueva *población* con los mejores *fitness* y con una descendencia que se espere sea mejor que la anterior. Desde este punto la función *BestRecord()* es vuelta a llamar para encontrar el mejor de los *individuos* dentro de la *población* recién creada. El mejor de ellos es seleccionado y es elegido para ser comparado con el primer *individuo* creado.

```
If best > bestInd(0) Then
       best = bestInd(0)
       Call Rhino.DeleteObject(OriginCrv)
       Call Rhino.DeleteObjects(boxes)

       OriginCrv = Rhino.AddPolyline(InitPopulation(bestInd(1)))
       boxes = myfuncboxes(OriginCrv)

       Call Rhino.Print(bestInd(0))
End If
```

Si *best* (mejor *individuo* de la *población* antigua) es de mayor longitud que *bestInd* (el actual mejor *individuo*) el cromosoma de *bestInd* es guardado dentro de *best*, se elimina la geometría del antiguo *individuo* y se crea una nueva con el *individuo* más apto actual, finalmente se imprime la longitud del mejor actual en la línea de comandos, y luego todo el proceso vuelve a repetirse en el mismo orden.

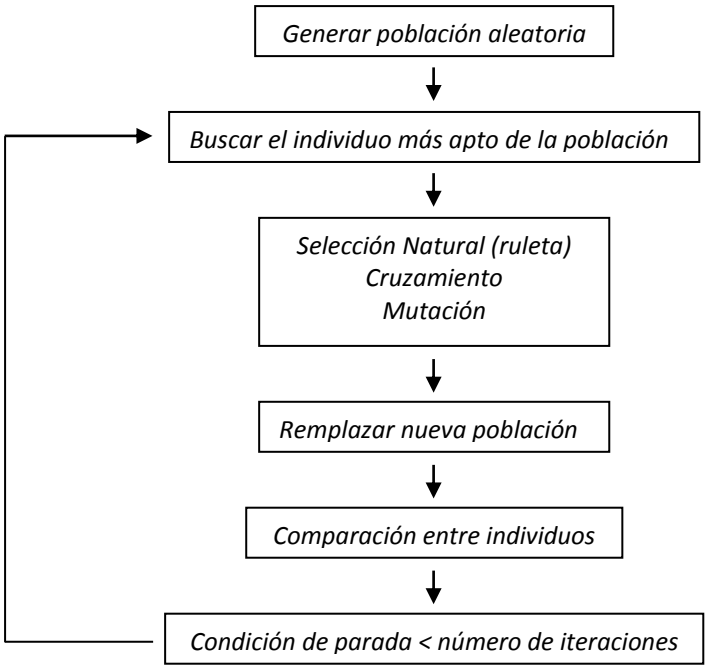


Figura 76. Esquema gráfico del AG empleado en la generación urbana.

8.1.3. Resultados

El primer resultado en ser mostrado corresponde a un experimento sobre 20 puntos y 50 iteraciones. El mejor *individuo* fue encontrado en la Generación 27.

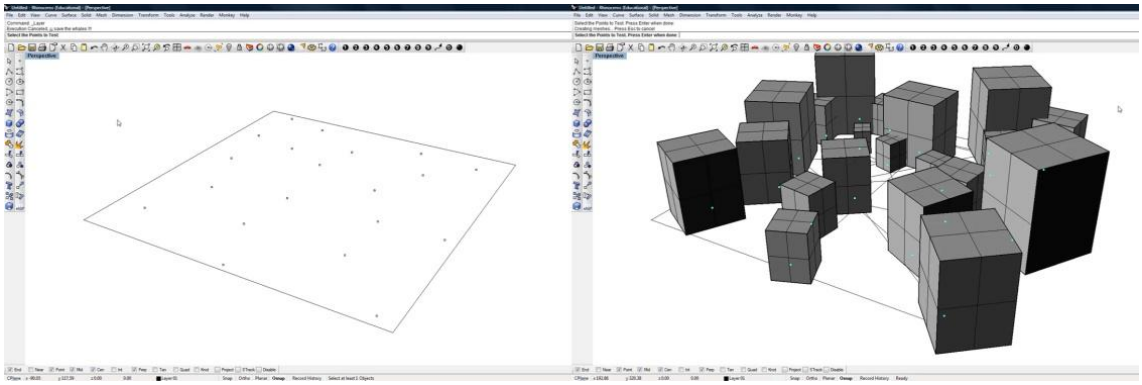
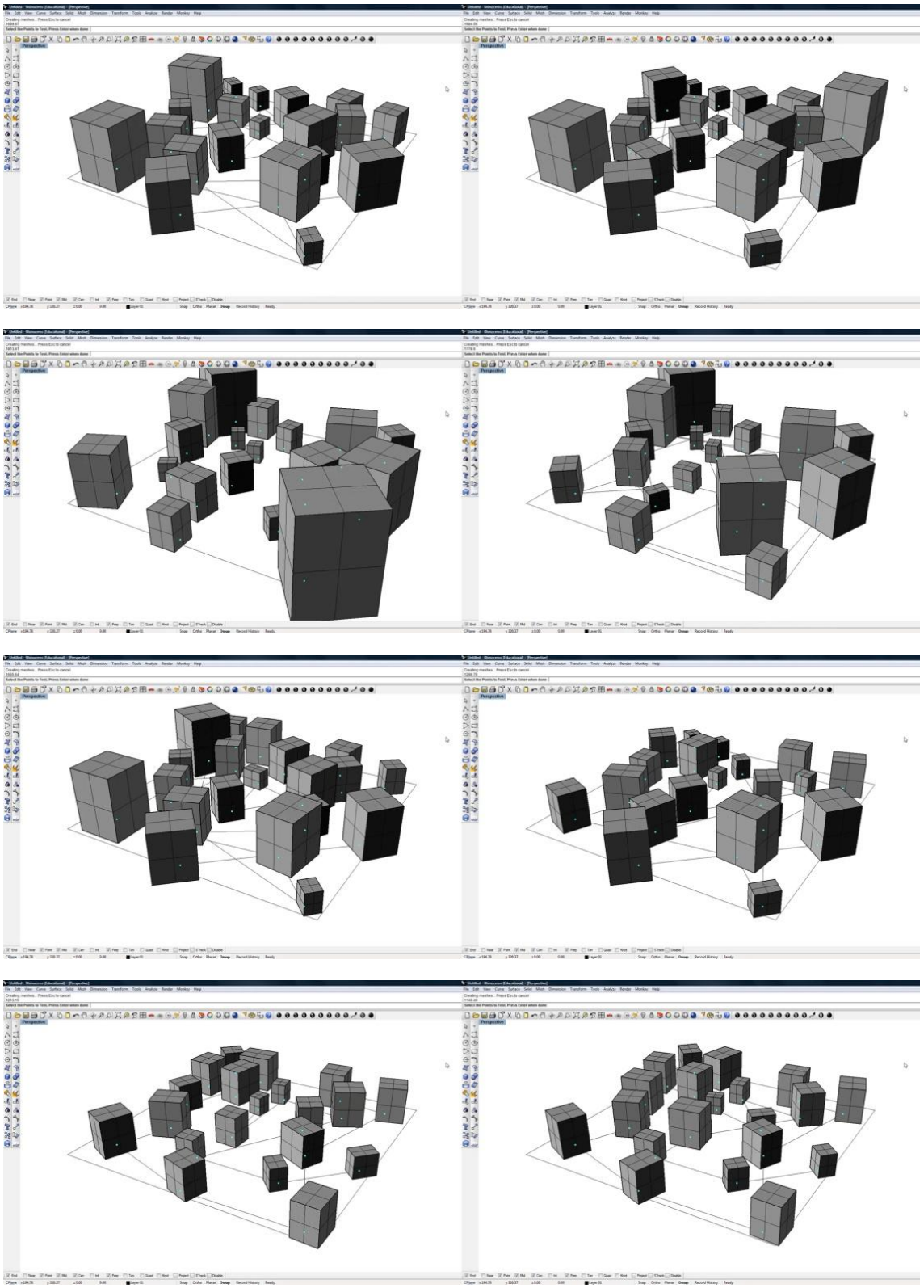


Figura 77. Impresión de pantalla durante la ejecución del AG. En la imagen de la izquierda aparecen los 20 puntos sobre el espacio de trabajo. En la Imagen de la derecha el *individuo* con el mejor *fitness* de la primer *población* generada aleatoriamente. Los volúmenes están en proporción a la distancia que lo separa del punto anterior dentro de la lista. En el comienzo, las longitudes son grandes (bajo *fitness*), por eso los volúmenes generados aparecen excesivamente escalados. Durante la optimización el volumen de las cajas tiende a equipararse. Esto es debido a que el AG busca minimizar las distancias que existen entre los puntos para encontrar la ruta más corta.



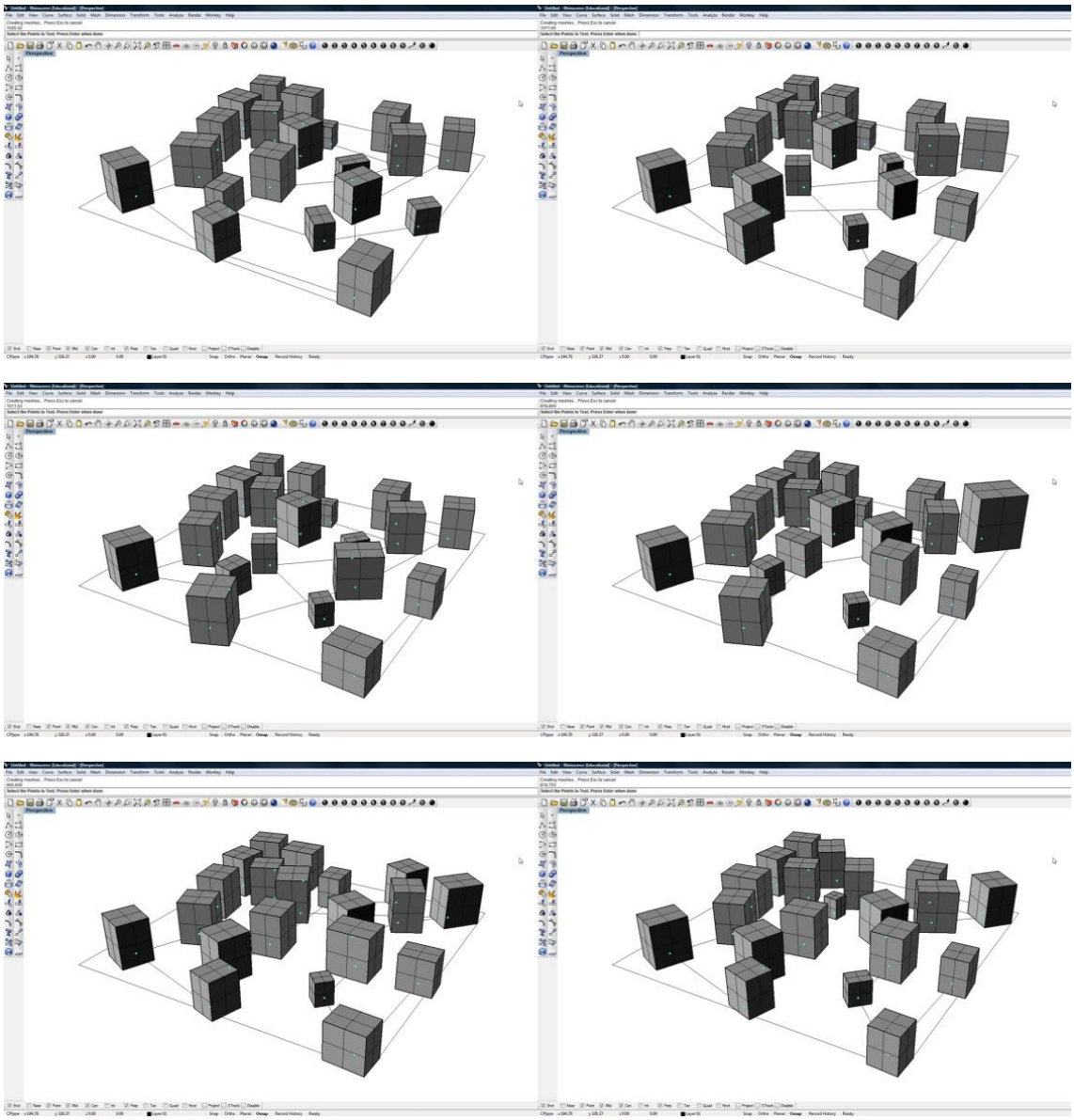


Figura 78. Secuencia de imágenes durante el proceso de optimización. Es posible ver como los volúmenes van disminuyendo en tamaño y el conjunto gana proporción.

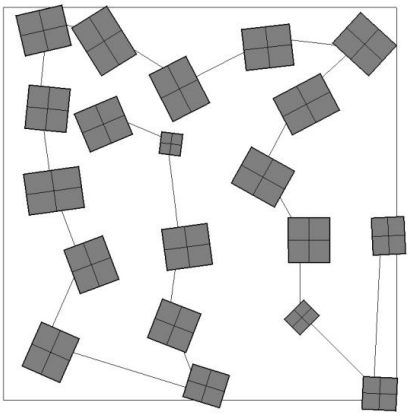


Figura 79. Vista en planta después de la optimización.

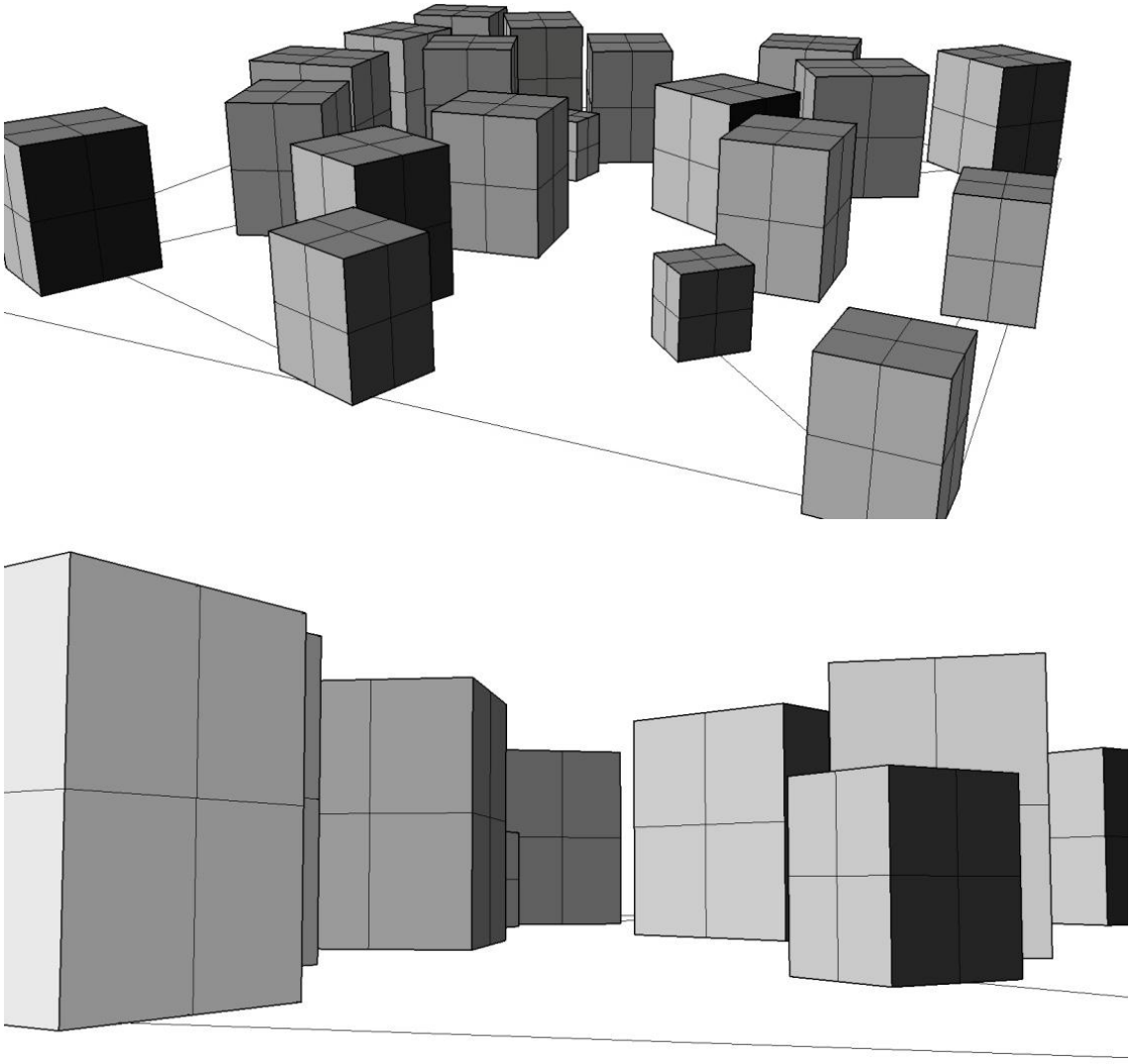


Figura 80. Perspectivas de la mejor solución.

Las perspectivas pretenden mostrar la relación espacial entre los volúmenes. Cada volumen estaría representado por uno de los 20 programas mencionados previamente. La disposición de estos sobre el terreno fue definida por el arquitecto al momento de pensar la organización de la urbanización. Consideraciones topográficas, logísticas, económicas podrán influir en la toma de decisiones al momento de emplazar los puntos sobre el terreno.

El AG propone un modelo urbano, basado en el recorrido más corto entre todas las zonas definidas por el usuario. Las zonas urbanas son representadas por una serie de volúmenes de 4 a 5 plantas los cuales están relacionados con la distancia más cercana al volumen anterior dentro de la ruta. Sin embargo esta situación se da sólo en los volúmenes pares dentro de la lista. El conjunto final es una distribución espacial donde se generan espacios que podrían ser tratados como plazas, áreas de esparcimiento, etc. La ruta entre cada uno de los volúmenes, la menor distancia en recorrido, podría ser la vía para conectar cada uno de ellos.

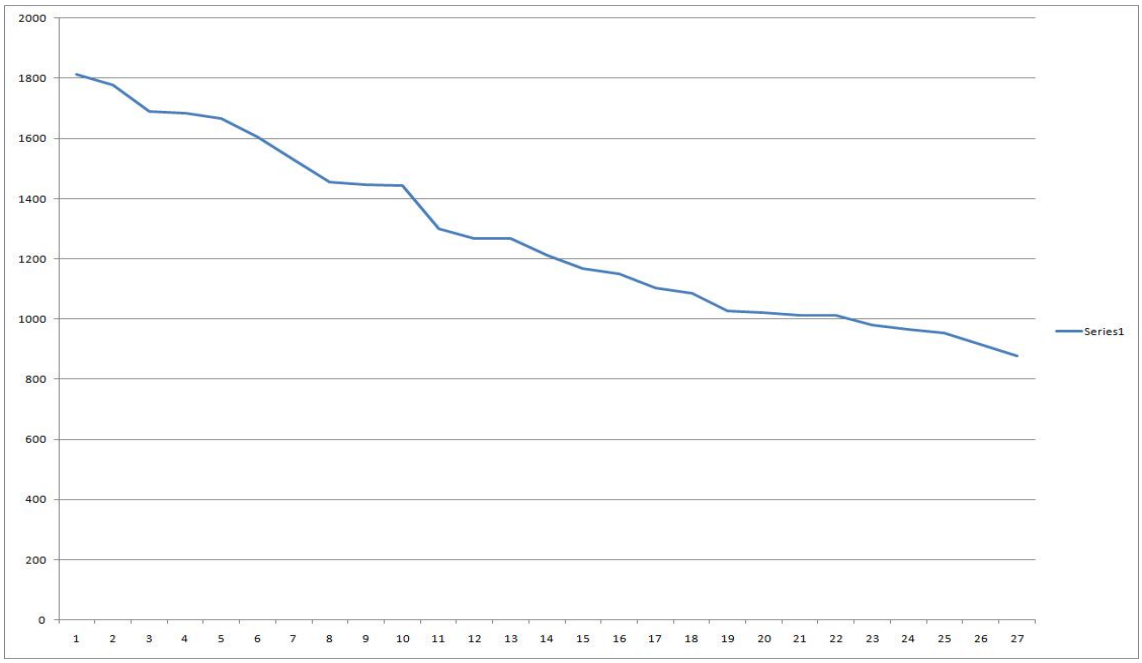


Figura 81. Curva del proceso de optimización. Longitud de la ruta en el eje Y del gráfico y número de generación en el eje X.

El mejor *individuo* de la primera *población*, (generada aleatoriamente) tuvo una longitud de 1813,41 m. y descendió hasta encontrar el mejor *individuo* en la generación 27, con una longitud de 878,753 m. La diferencia entre el mejor y el peor de los *individuos* es de un 51,54% menos de longitud. En un espacio de búsqueda de 20!.

1813,41	*1778,6*	*1689,67*	*1684,56*	*1665,64*	*1604,52*
1529,24	*1455,36*	*1447,3*	*1444,28*	*1299,79*	*1266,72*
1266,4	*1213,15*	*1167,02*	*1149,49*	*1103,69*	*1085,2*
1026,42	*1020,86*	*1011,65*	*1011,63*	*978,869*	*966,609*
954,171	*914,256*	*878,753*			

Figura 82. Lista de longitudes de rutas encontradas en cada generación donde se produjo optimización.

Las siguientes imágenes corresponden a una serie de pruebas realizadas sobre una topografía similar a la pendiente de un cerro. Las imágenes no corresponden a una generación, como en el caso anterior, sino que a una serie de óptimos con diferentes cantidades de puntos. Cada una de las generaciones fue testeada sobre la misma superficie.

La definición del AG corresponde exactamente con el ejercicio anterior. Esto significa que se usaron los mismos porcentajes de cruce, mutación, desplazamiento de curva, etc. Lo único que vario fue la cantidad de puntos (volúmenes) en la optimización.

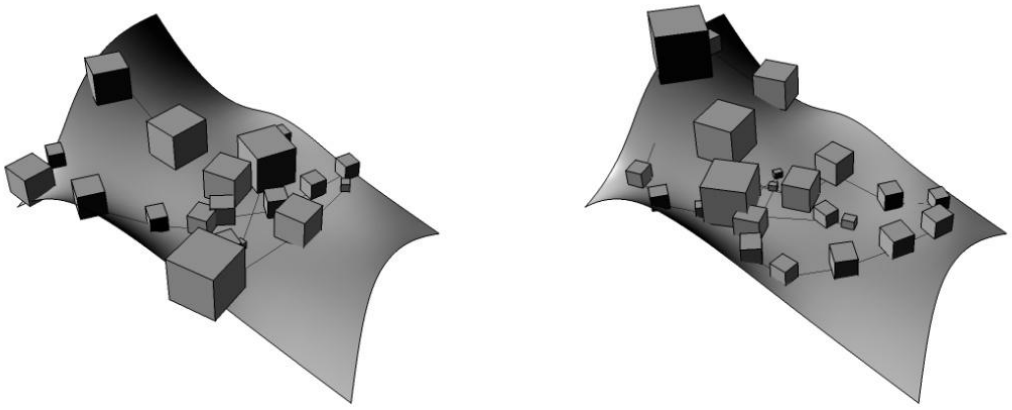


Figura 83. Ambos resultados con 20 puntos generados aleatoriamente sobre la misma superficie.

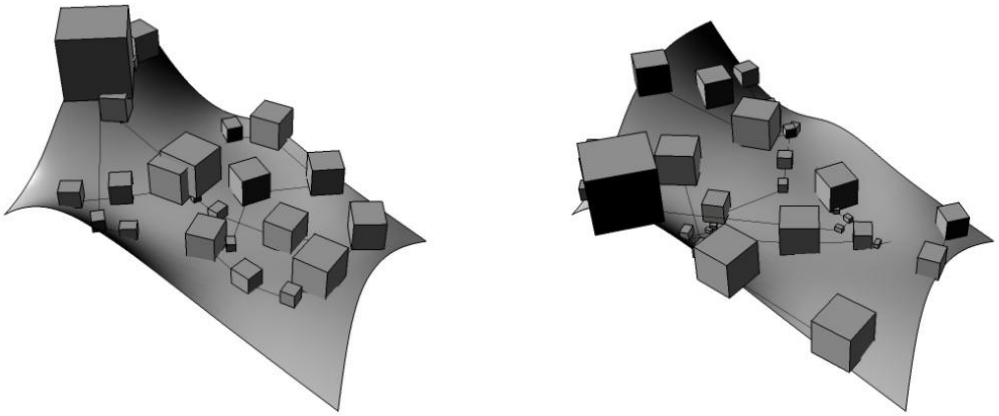


Figura 84. La imagen de la izquierda con 25 puntos y la imagen de la derecha con 30 puntos.

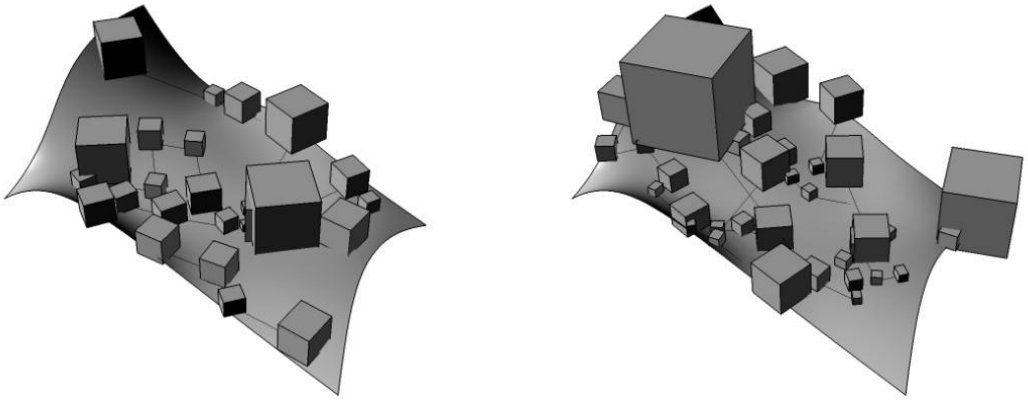


Figura 85. 30 y 40 puntos respectivamente.

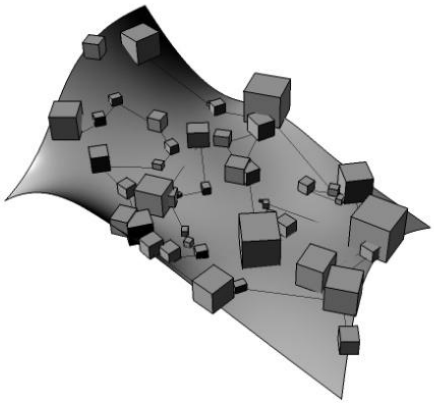


Figura 86. Optimización realizada con 50 puntos aleatorios sobre la superficie.

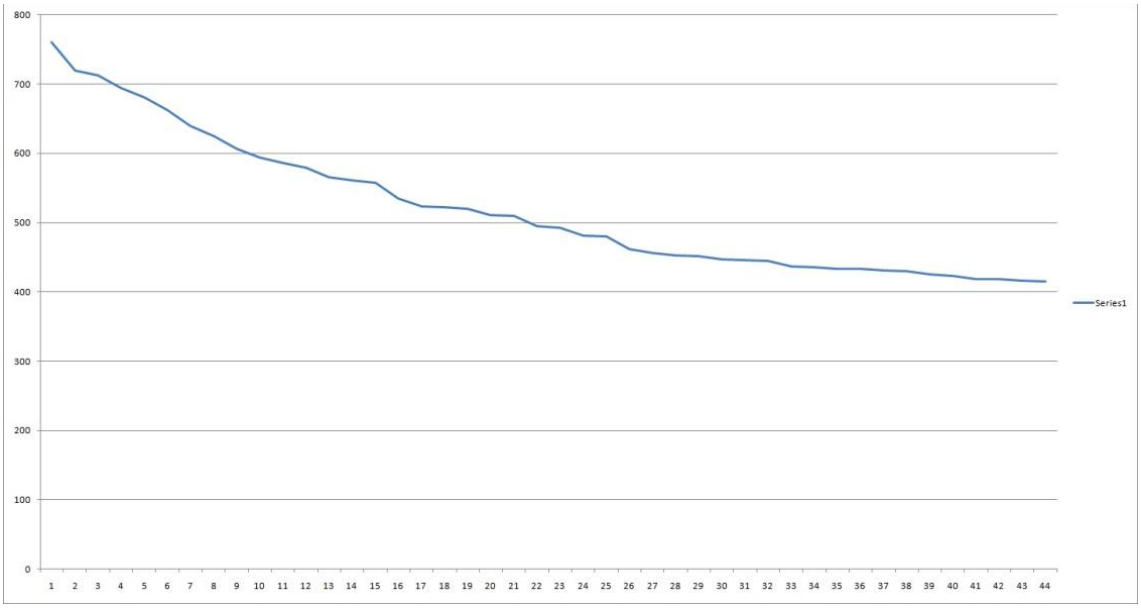


Figura 87. Impresión de pantalla de la curva de optimización sobre 50 puntos, del último ejercicio. Longitud de la ruta en el eje Y del gráfico y número de generación en el eje X.

El algoritmo en 44 iteraciones, redujo la longitud de ruta en casi un 50%. En la primera generación el mejor *individuo* estuvo alrededor de los 760m (760,87). Y al final del proceso estuvo cerca de los 410 metros de longitud (414,826).

El siguiente resultado, corresponde a una curva de optimización sobre el espacio de búsqueda de 100 puntos lo que significa 9.33262154439441e+157 posibles combinaciones. Se utilizaron poblaciones de 50 *individuos*. El punto de cruce fue realizado en el alelo número 25, lo que significa que copiaba la mitad del padre y la mitad de la madre. El porcentaje de mutación fue de 3 alelos y el mejor *individuo* seleccionado era copiado 50 veces en cada generación. El mejor resultado fue encontrado durante la iteración 273, aunque el AG optimizó continuamente durante cada generación.

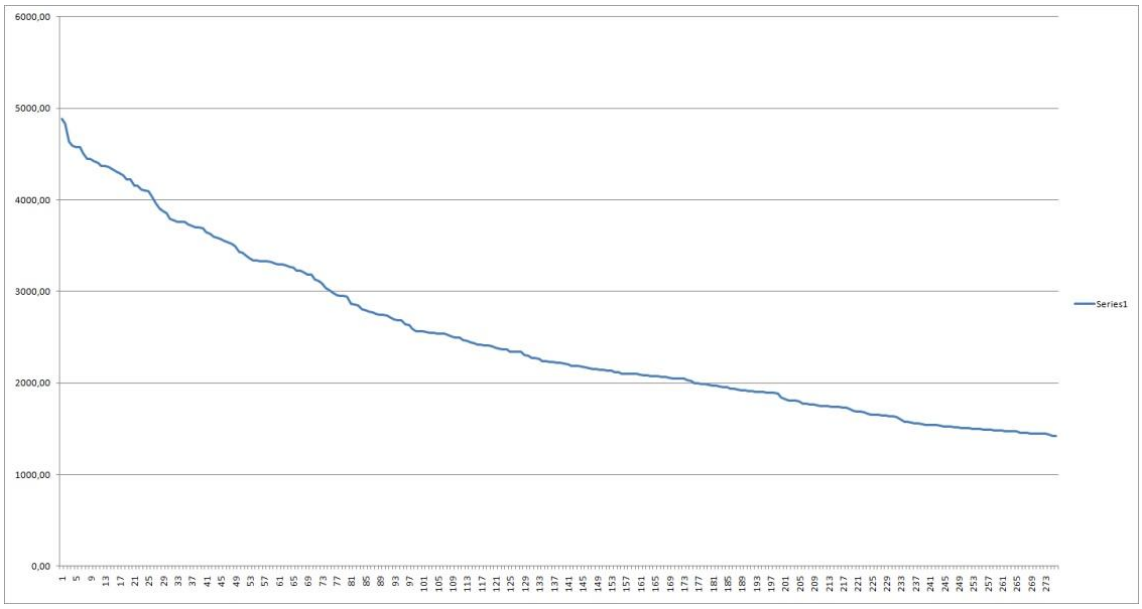


Figura 88. Impresión de pantalla de una curva de optimización sobre 100 puntos. El AG en la primera generación produjo un *individuo* de longitud 4887,61m., el cual consiguió optimizar hasta alcanzar el valor de 1416,22m. En la iteración número 273. Reduciendo la longitud en más de un 68%. Longitud de la ruta en el eje Y del gráfico y número de generación en el eje X.

8.1.4. Discusión

El AG redujo las longitudes de rutas, sin importar la cantidad de puntos, entre un 50% y un 70% para cada caso. El proceso de generación está basado en tres simples reglas. Primero, todos los volúmenes se ajustan entre sí para encontrar la ruta más corta entre ellos. Segundo, Todos los volúmenes se escalan en relación a su vecino más próximo dentro de la ruta. Esto permite que exista una coherencia espacial creando relaciones urbanas y la posibilidad de controlar diferentes densidades modificando las posiciones de los puntos. Tercero, la orientación entre los volúmenes dentro de la ruta generada hace que la circulación tradicional de una zona urbana, el damero por ejemplo, desaparezca para dar paso a un complejo juego de espacios y relaciones todas conectadas entre sí.

La emergencia es una característica que surge espontáneamente de un proceso de auto-organización en los sistemas naturales. La idea es que simples reglas, sean capaces de producir complejos comportamientos.

Uno de los ejemplos más interesantes es la ciudad de Manchester, expuesto en el libro "*Sistemas emergentes o que tienen en común hormigas, neuronas, ciudades y software*" de Steven Johnson[82]. Manchester y toda la región de Lancashire, constituyeron el corazón de la revolución comercial y tecnológica que influenciaría el futuro del planeta. Johnson explica que en Manchester ocurren varios hitos importantes como por ejemplo, aparecen las primeras tecnologías textiles a vapor, el sistema bancario del Londres comercial, los mercados globales y

82 JOHNSON, Steven. Sistemas emergentes, o que tienen en común hormigas, neuronas, ciudades y software, Madrid, Turner, 2008 (1ª ed. 2001).

la aparición de los sindicatos. En el libro Johnson explica que entre el 1700 y 1850, el despegue industrial de Manchester, creo un tipo de ciudad que literalmente, estalló.

Johnson explica que en el año 1773 vivían en Manchester 24.000 personas. En el año 1801 el censo fue de 70.000 y a mediados del siglo XIX había más de 250.000 habitantes dentro de los límites de la ciudad. Una cifra que se multiplico por 10 en 75 años. Lo interesante del asunto es que durante 500 años Manchester fue considerada como un señorío, lo que significaba que era gobernada como un estado feudal, sin gobierno local, sin planificación urbana, sin policía y sin autoridades sanitarias. Manchester envío representantes al parlamento en 1832 y recién en 1840 comenzaron a ver las primeras reformas sanitarias y de planificación urbana. Johnson dice "*Esto constituye una de las mayores ironías de la revolución industrial, y que revela hasta qué punto este cambio fue realmente espectacular: la ciudad que definió el futuro de la vida urbana durante la primera mitad del siglo XIX no se constituyó legalmente como ciudad hasta que la gran explosión no completó su desarrollo. Como resultado de tal discontinuidad Manchester se convirtió en la ciudad más caótica y menos planificada en los 6000 años de historia de los asentamientos urbanos*"[83].

Johnson cita a Friedrich Engels, que llegó a la ciudad en 1842. Engels explica que Manchester puede recorrerse durante años de años de un extremo a otro, sin encontrarse con un barrio obrero o tener contacto con obreros. Esto lógicamente se debe a que los barrios obreros están separados de los de la clase media. Dice también, que los comercios minoristas ocupan las calles principales. Que en estás existen más casas buenas que malas y que el valor del terreno es mayor que el de las calles alejadas. "*Manchester este construida con pocas reglas o prescripciones policiales, y más en contra de ellas que cualquier otra ciudad*"[84]. Más adelante, menciona que la ciudad tiene un cordón sanitario para separar a la zona industrial de los barrios obreros. Que la ciudad parece hábilmente planeada para esconder sus "*atrocidades*" pero que ha sido construida menos de acuerdo a una planificación que cualquier otra ciudad en la historia.

Johnson, también cita a Steven Marcus: "*Sus casas están situadas por fuera del cinturón de la clase obrera y entre ese cinturón y los barrios residenciales de la clase media alta. Y sus comercios y pequeños negocios están ubicados a lo largo de las calles principales de modo que son, por así decirlo, aislantes del sistema de comunicación de la ciudad. Su ubicación en el espacio, por lo tanto, es una unión crítica y una de intermediación al mismo tiempo. Y su posición intermediaria no es meramente estructural sino también funcional. Actúan como amortiguadores entre extremos antagónicos. Lo que hay que destacar es que este arreglo desconcertante y atroz no puede entenderse como resultado de una trama ni de un diseño liberado, aunque aquello los interesados también lo controlen. Sin duda, es la organización de un estado de cosas demasiado vasto y complejo como para haber sido pensado con anticipación, como para haber existido como idea previa*"[85]

83 Ibid. p. 33-34.
84 Ibid. p. 35.
85 Ibid. p. 36. In MARCUS, Steven. Engels, Manchester, and the Working Class, Nueva York, W.W. Norton, 1974, p. 172-173

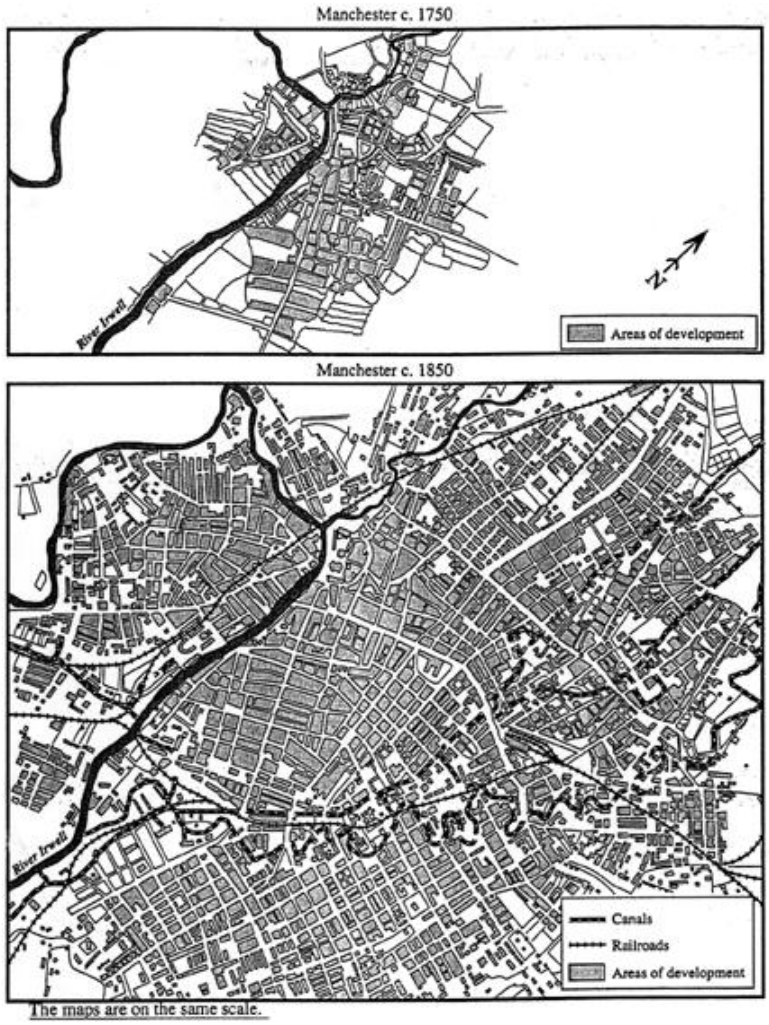


Figura 89. Planos de la ciudad de Manchester en 1750 y luego en 1850. http://www.thecaveonline.com/APEH/dbqmanchester.html[2010/05/30]

Las ciudades tienen vida propia. Funcionan como un sistema de industrias, viviendas y personas que interactúan para favorecer su desarrollo. Christopher Alexander lo describe así: "*En Berkeley en la esquina de Hearst y Euclid, hay una tienda, y delante de la tienda hay un semáforo. En la entrada hay un expendedor con los periódicos del día. Cuando el semáforo está en rojo, los peatones que van a que cruzar la calle esperan al lado del semáforo; como no tienen nada que hacer, miran los periódicos que hay en el expendedor y que pueden ver desde donde están. Algunos sólo leen los titulares, otros compran el periódico mientras esperan. Este efecto convierte al semáforo y al expendedor en elementos interactivos. El expendedor, los periódicos que se exhiben, la moneda que pasa del bolsillo a la ranura, la gente que se detiene en el semáforo y lee el periódico, el semáforo, los impulsos eléctricos que hacen cambiar la luz y la acera en la que la gente espera forman un sistema en el que todos actúan conjuntamente*"[86].

86 ALEXANDER, Christopher, "A City is not a Tree", Architectural Forum, vol. 122, nº1, 1965 p. 85 en FERRÉ, A., HWANG I., KUBO, M., PRAT, R., SAKAMOTO, T., SALAZAR, J., TETAS, A.; VERB Architecture boogazine Connection, num 3, Actar, Barcelona, Diciembre 2004.

Cuando una zona se urbaniza y se ocupa el terreno, aparecen procesos de envejecimiento que detienen de cierta manera la interacción entre las diferentes partes de la ciudad. Se transforma el terreno, se modifica el comercio y lentamente la industria comienza a declinar con el envejecimiento de las viviendas. Pero las interacciones entre las actividades sociales y económicas de una zona urbana son tan complejas que la intuición con la que se planearon algunas ciudades durante siglos, no es capaz de prevenir los problemas de la ciudad moderna. Las ciudades, en general, se planearon de manera vertical, en el sentido que una cabeza (o varias) pensante determinaron a priori el uso del terreno y la manera en la que los elementos de la ciudad se relacionarían. Determinaron, donde deberían estar las calles comerciales más importantes, el comercio, el cordón sanitario, las plazas, etc. Pero al ser ocupada la ciudad, estas directrices marcadas no han sido capaces de contener las interacciones y relaciones que se producen dentro de una zona urbana. Originando con el tiempo zonas marginadas y en declive, guetos, contaminación, falta de áreas verdes, etc. Nuevos planeamientos sobre la ciudad pretenden rescatar estos barrios, modificando normativas y planes urbanos con el fin de revitalizarlos y mejorar la calidad de relaciones. Sin embargo si se vuelve hacer el planeamiento con la intuición que se hizo en un principio volverán a aparecer nuevas zonas urbanas con anquilosamiento. Por lo tanto el plan volverá a ser sobrepasado por el comportamiento emergente que existe entre las relaciones sociales y comerciales de los humanos.

En la película Jurassic Park[87] el investigador del caos, Ian Malcolm (Jeff Goldblum), en la secuencia en la que vuelan hacia la isla donde se encuentran los dinosaurios atrapados dentro de un zoológico dice: "*El miedo que me da esto, es que por mucho que intentemos controlar la naturaleza, está siempre intentará escapar hacia el caos*". Por otro lado, Jay W. Forrester, en el libro *Urban Dynamics* [88] dice lo siguiente: "Ha *quedado claro que los sistemas complejos son contrarios a la intuición. Es decir, ofrecen pistas para una acción correctora que muchas veces producirá un resultado ineficaz o incluso adverso. Muchas veces, las medidas políticas adoptadas para corregir un problema, en realidad lo intensifican en vez de dar una solución".*

Optar por una política inefectiva o perjudicial para enfrentarse a un sistema complejo no es un problema de suerte. En la mayoría de los casos, los procesos intuitivos seleccionarán la solución errónea. Un sistema complejo se comporta de muchas maneras y es muy distinto a los sistemas simples a los que estamos acostumbrados"[89].

Tanto Forrester como Christopher, proponen en sus libros el uso de herramientas generativas en la planificación urbana, en vez de aplicarlo a modelos diseñados de manera intuitiva. Dividen estás herramientas en dos partes. La primera como modelo de generación urbana y la segunda, como un modelo de agentes para simular el comportamiento del sistema dentro de la estructura propuesta.

El TSP como Modelo Generativo de Diseño propone un modelo urbano basado en una simple técnica evolutiva como es el AG, que tiene implicaciones en la escala y orientación de los edificios designados. Por el momento su calificación como modelo urbano no ha sido comprobada. No se puede decir si el modelo funcionará o que tan exitoso será. Por este

87 Kathleen K. Gerald R. M., 1993, Jurassic Park, SPIELBERG, Steven. [película]. Estados Unidos, DC: Universal Studios.
88 FORRESTER, J. W., Urban Dynamics, Cambridge: MIT Press, 1969
89 FORRESTER, J. W, "Contraria a la Intuición", Architectural Forum, vol. 122, nº1, 1965 p. 85 en FERRÉ, A., HWANG I., KUBO, M., PRAT, R., SAKAMOTO, T., SALAZAR, J., TETAS, A.; VERB Architecture boogazine Connection, num 3, Actar, Barcelona, Diciembre 2004.

motivo se han referenciado estos autores que han estudiado sistemas emergentes en ciudades y todos ellos concluyen que la generación de estructuras urbanas por medio de sistemas generativos es una opción real en el planeamiento urbano.

Con respecto al algoritmo y la parte técnica de este trabajo, se puede decir que el AG optimizó y redujo en un 100% de los casos las longitudes de las rutas alrededor de un 50%. Para hacer más eficiente el AG sería necesario enfrentar dos diferentes desafíos. El primero utilizar un lenguaje de programación de bajo nivel, basado en C, para así ganar velocidad y hacer el programa más robusto. Segundo, plantear un hibrido del AG con otra técnica de optimización no evolutiva, como por ejemplo una búsqueda binaria o un divide y vencerás. Esto permitiría encontrar soluciones más rápidamente en el sentido que la primera generación no sería completamente aleatoria sino que habría desde el principio una intención de optimización.

Este sistema generativo, no sólo podría aplicarse al ejemplo propuesto dentro de esta investigación sino que también a otro tipo de ejercicios urbanos. Por ejemplo, buscar las distancias más cortas entre áreas verdes dentro de una ciudad e intentar crear un corredor verde que las conecte todas. Utilizar el AG para "destruir" una trama urbana ineficiente y utilizarla como explorador para nuevas tentativas de diseño.

8.2. AG como Sistema de Diseño Generativo

8.2.1. Introducción

Mucho se puede escribir sobre técnicas y procesos proyectuales en arquitectura. Además, la manera de proyectar es algo personal del arquitecto y cada uno defiende su postura en cuanto a este proceso. Diseñar la forma de un edificio es una cuestión muy compleja, en el cual intervienen muchas variables que deben ser satisfechas. El arquitecto decide cuál de estas variables tendrán privilegios por sobre las otras, cuales sacrificará en función de defender las intenciones de diseño. Sin embargo el edificio debe satisfacer todas sus restricciones, como por ejemplo presupuesto, normativa, cliente, condiciones técnicas, materiales, emplazamiento, modas, etc.

El siguiente trabajo plantea el uso de un AG como Sistema de Diseño Generativo (SDG) el cual es capaz de proponer formas al arquitecto optimizando una serie de funciones que se contraponen entre sí. La propuesta de este sistema consiste en buscar el óptimo para una serie de variables, sin sacrificar ninguna de ellas en función de otra. Está estrategia es conocida como el óptimo de Pareto, la cual se explica así: "*El concepto de eficiencia de Pareto (también llamado óptimo de Pareto, Pareto-optimalidad u óptimo paretiano) es aquella situación en la cual se cumple que no es posible beneficiar a más elementos de un sistema sin perjudicar a otros. Se basa en criterios de utilidad: si algo genera o produce provecho, comodidad, fruto o*

interés sin perjudicar a otro, provocará un proceso natural de optimización hasta alcanzar el punto óptimo"[90].

Dentro de las variables a optimizar consideradas dentro del sistema generativo existe la de maximizar el volumen del edificio y minimizar la ocupación de suelo para liberar la planta baja, valores en contraposición. Está función se llama "*the developer's Manhattan function*"[91]. En el libro *Programming.Architecture*, se explica que ambas variables compiten como si fueran el precio de un alquiler. Sin embargo, para hacerlas competir es necesario premiar un volumen grande y castigar el uso del suelo. Otra variable considerada corresponde a la distancia de una zona del edificio a unos puntos fijados previamente denominados puntos *repellors*[92]. Estos puntos repelen la geometría que estará cerca de ellos, castigando el *fitness* del *individuo*. La última variable en juego consiste en buscar dentro del cromosoma de cada *individuo* cierta información la cual será premiada. Está podrá perjudicar el *fitness*, si está información se opone con alguna de las otras variables.

Imaginemos que deseamos llenar un prisma de 20x15x30 m con cubos de 1x1x1 m. Si la primera variable que buscamos consiste en maximizar el volumen parece lógico rellenar el prisma con los cubos. Sin embargo el uso de suelo es castigado por lo que ambas variables entran en juego. Esto implica que la variable del volumen tendrá que lidiar con la ocupación de suelo y buscar un equilibro dependiendo en cuanto es premiada y cuanto castigada. Las zonas afectadas por los puntos *repellors*, será castigadas en su *fitness* si tienen cubos dentro de su área de influencia, por lo que entrará en conflicto con la variable que premia el volumen. Probablemente entre en conflicto con la variable de uso del suelo, en la medida que también afecte su área de influencia. Además de todo esto si agregamos la variable que premia cierta información encontrada dentro del cromosoma, está podrá entrar en conflicto con todas las variables anteriores. Por ejemplo premiará dentro del cromosoma secuencias de 1s de longitud 5, (11111) esto quiere decir que cuando hubiesen 5 unos seguidos, el *fitness* será premiado, pero se opondrá a la variable si estos unos se encuentran dentro del área de influencia de un punto *repellor*.

Este es un problema de múltiples objetivos[93], donde no se trata de maximizar una de las variables sino satisfacer cada una de ellas, de la mejor manera posible (Pareto).

---

90 http://es.wikipedia.org/wiki/Eficiencia_de_Pareto. [01/06/10].
91 COATES, Paul., Programming.Architecture, Nueva York, Routledge, 2010, p. 97
92 Genr8 (Generate) es un plug-in de Maya diseñado por Una-may O'reilly y programado por Martin Hemeberg. En el año 2000, para el Emergent Design Group, del MIT.
http://projects.csail.mit.edu/emergentDesign/genr8/, [16/08/10]. (Capítulo 6.14)
93 La optimización multiobjetivo, puede ser definida como un problema de optimización que presenta dos o más funciones objetivo. El inconveniente principal que presenta este tipo de problemas en relación a un modelo de objetivo único, radica en la subjetividad de la solución encontrada. Un problema multiobjetivo no tiene una solución óptima única, más bien, genera un conjunto de soluciones que no pueden ser consideradas diferentes entre sí. BAESLER, Felipe., CEBALLOS, Luis., RAMÍREZ, Milton., Programación multiobjetivo de máquinas moldureras a través de algoritmos meméticos, Departamento de Ingeniería Industrial, Universidad del Bío-Bío, Concepción, Chile, 2006, http://www.scielo.cl/scielo.php?pid=S0718-221X2006000300005&script=sci_arttext, [01/06/10]

8.2.2. Método

El algoritmo está basado en un cromosoma binario (0110101), en el cual los 1 representan un cubo de 1x1x1 y 0 vacío. Los cubos se agrupan para constituir una geometría, más compleja definida por ancho, largo y alto. El SDG optimiza la distribución de los cubos dentro de la geometría para satisfacer una serie de variables.

ESTRUCTURA DEL ALGORITMO

Cuando se ejecuta el SDG, la primera tarea que realiza consiste en preguntarle al arquitecto por un archivo de tipo *.texto. Este archivo servirá para dos cosas fundamentales. Primero, guardar la configuración del algoritmo como cantidad de iteraciones, tipo de cruce, etc. Segundo, almacenar la información sobre la optimización que irá siendo registrada en las iteraciones donde se produzca un incremento del *fitness* con respecto al *individuo* más apto de la generación anterior. El siguiente paso del SDG es preguntarle al arquitecto por la configuración del algoritmo, en este punto el programa muestra la siguiente ventana:

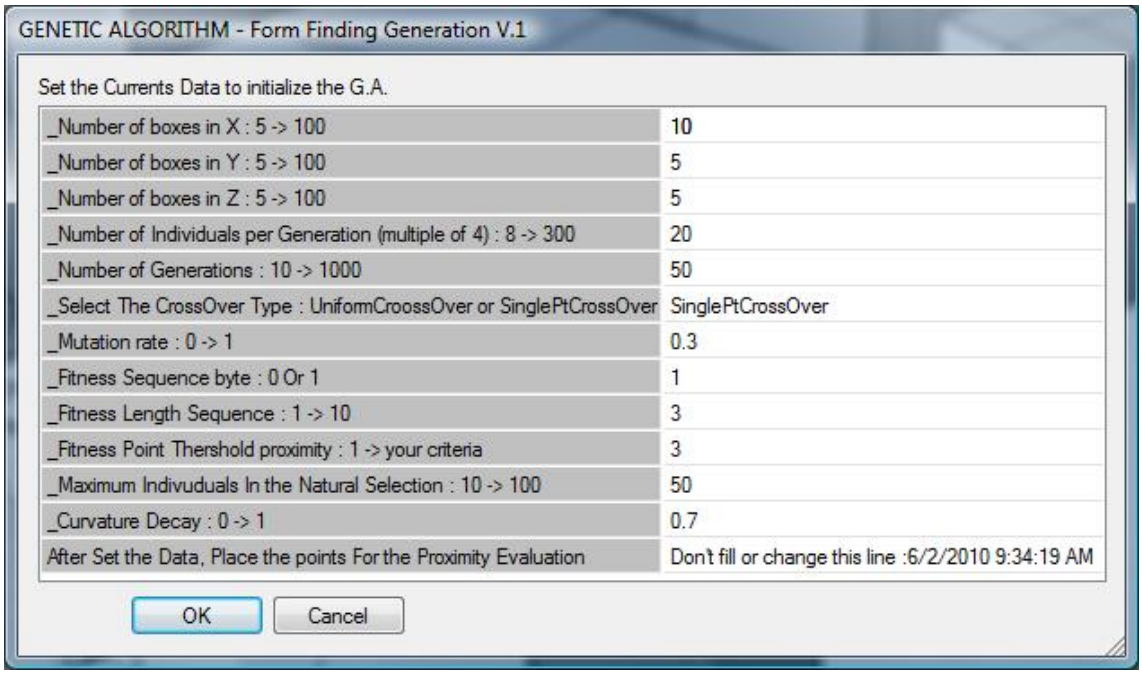


Figura 90. Ventana de configuración del SDG

Los primeros tres campos: *Number of boxes in X*, *Number of boxes in Y* y *Number of boxes in Z*, permiten al usuario determinar la cantidad de cubos como máximo en las coordenadas en X, Y, Z. El siguiente campo, es el número de *individuos* por generación, el cuál debe ser múltiplo de cuatro. Luego es necesario ingresar el número de generaciones, o las veces que el algoritmo se ejecutará. El sexto campo, consiste en el tipo de cruzamiento que usara el algoritmo. Este campo es particular porque en vez de solicitar un valor, pide al arquitecto que ingrese un texto. Estos serán, *UniformCrossOver* o *SinglePtCrossOver*. Dependiendo de este campo, el SDG utilizará un tipo de cruzamiento u otro. El campo *Mutation rate*, define el porcentaje de

mutación, el cual se mueve entre los rangos 0 y 1. *Fitness Sequence byte*, tiene dos opciones, 0 o 1. Dependiendo de este campo el SDG leerá la información contenida dentro de los cromosomas buscando 0s o 1s. *Fitness Length Sequence*, define la longitud de secuencia de 0s o 1s, que buscara el SDG dentro del cromosoma de cada *individuo*. Cuando encuentre una secuencia del byte y longitud especificada, el sistema premiara al *individuo* mejorando su *fitness*. El siguiente campo (*Fitness Point Thershold proximity*) pide al arquitecto que defina la distancia en la que influirán los puntos en la geometría. Los siguientes dos campos definen el máximo de *individuos* por generación y el factor de caída de la curva de selección, para hacer el sistema más selectivo o no. El último campo, informa al usuario de que debe seleccionar los puntos para la evaluación de proximidad. Asimismo la fecha y hora de cuando se realiza la optimización. A continuación el arquitecto deberá elegir qué puntos quiere hacer influir en la geometría. La elección de puntos se hace por medio del mouse haciendo clic en el espacio de trabajo dentro de *Rhino*.

El siguiente paso del sistema consiste en generar una *población* aleatoria de *individuos*. La primera función llamada es: *GenrRndChromose()*, la cual toma como argumento los tres primeros campos de la ventana de configuración, X, Y y Z. Está función es llamada el número de veces definido en el campo *Number of Individuals per Generation*.

```
Function GenrRndChromose(h, w, l)
	Dim i, j, k, count : count = 0
	Dim bitstr()
	For i = 0 To h - 1
		For j = 0 To w - 1
			For k = 0 To l - 1

				ReDim Preserve bitStr(count)
				Randomize
				If (Rnd > 0.5) Then
					bitStr(count) = 1
				Else
					bitStr(count) = 0
				End If
				count = count + 1
			Next
		Next
	Next
	GenrRndChromose = bitStr

End Function
```

Los tres bucles anidados dentro de la función son necesarios para crear la estructura de la geometría, aunque por el momento es sólo información. Un número es generado aleatoriamente en cada iteración. Si este es mayor a 0.5 dentro del cromosoma se guardará un 1 de lo contrario un 0. Una geometría de X = 10, Y = 5 y Z = 5 genera un cromosoma de 255 bytes. La función retorna una lista con unos y ceros. Esta lista será el cromosoma de cada *individuo*.

0	1	2	3	4	5	6	7	…	N + 1
0	*1*	*1*	*1*	*0*	*0*	*1*	*0*	*1*	*1*

Figura 91. Estructura del cromosoma.

Una vez realizada esta acción, cada uno de los *individuos* generados aleatoriamente son guardados dentro de una lista mayor llamada *Generacion*. Esta lista y los parámetros bit (1 o 0), longitud de la serie, los puntos, el umbral de influencia y las mediadas de la geometría son pasados como argumentos a la función *SortFitnessGeneration()*. Dentro de esta función se realizaran una serie de acciones. La primera de ellas consiste en evaluar el *fitness* del primer *individuo* generado aleatoriamente. Para esto se llama la función *FitnessEvaluation()*, la cual toma el primer *individuo* y calcula su *fitness*. El cálculo del *fitness* pasa por una serie funciones las cuales irán evaluando los diferentes aspectos del *individuo*.

```
Function FitnessEvaluation(ByVal BitStr, sequenceBit, Series, ptsRepellor,
RadiusProximity, h, w, l)

       Dim fitSequence
       fitSequence = FitnessSecuencia(BitStr, sequenceBit, Series)
       Dim LatticePts
       LatticePts = PtsGenr(bitStr, h, w, l, 1)

       Dim insidePts
       insidePts = FitnessProximityAttractor(LatticePts, ptsRepellor,
RadiusProximity)

       Dim footPrint
       footPrint = FitnessfootPrint(BitStr, w, l)

      Dim volumen
       volume = FitnessVolume(BitStr)

       If(footPrint > 45) Then footPrint = footPrint * 5
       If(Volume > 800) Or (Volume < 700) Then Volume = Volume * 0.5

       FitnessEvaluation = (fitSequence + Volume) - (InsidePts * 2 + footPrint)

End Function
```

Dentro de la función de evaluación el primer aspecto a evaluar es la búsqueda de secuencias del bit elegido en la ventana de configuración. Un algoritmo específico busca dentro del cromosoma secuencias de 1 o 0 de la longitud determinada previamente. Cuando encuentra una incrementa un contador. El retorno de esta función es un número que contiene la cantidad de secuencias de bit encontradas dentro del cromosoma. La siguiente función *PtsGenr()*, recorre cada alelo del cromosoma. Cuando encuentra el valor, calcula aritméticamente la posición del punto y lo guarda en una nueva lista. La ventaja de hacerlo de esta manera es que no es necesario imprimir los puntos ni menos los cubos de 1x1x1. Esto hace el código más robusto y sobre todo se gana velocidad. Finalmente esta función retorna una lista con las coordenadas de todos los puntos en el espacio donde existe un 1. Esta nueva lista es usada por la función *FitnessProximityAttractor()* la cual usara un bucle para contar cuantos cubos (por el momento sólo información) están a una distancia de los puntos menor a la especificada en la ventana de configuración. Finalmente es llamada la función *FitnessfootPrint()*, que cuenta cuantos cubos se encuentran en la base de la geometría.

La arquitectura de este algoritmo ha sido diseñada para que toda la evaluación de las diferentes variables sea realizada de manera aritmética, sin imprimir ni dibujar nada en el espacio de trabajo. Con esto se evita que el algoritmo consuma recursos excesivos del ordenador, llevando el proceso a un fracaso absoluto.

Una vez que todas las funciones de evaluación han sido llevadas a cabo, dos estructuras condicionales muy simples son evaluadas. Si la cantidad de cubos en la base es mayor a 45, (más de la mitad de la ocupación del suelo) se penaliza la variable multiplicándola por 5. Si el volumen es mayor de 800 y menor que 700 entonces el volumen es multiplicado por 0.5. Esto significa que si el volumen no está dentro del rango 700-800, es penalizado. Finalmente se calcula el *fitness* de cada *individuo*, mediante una simple ecuación.

$$Fitness = (fitSequence + Volume) - ((InsidePts\ x\ 2) + footPrint)$$

Una vez que el *fitness* es calculado es guardado dentro de una lista junto con el índice del *individuo*, para ser identificado luego. Cuando se tienen todos los *fitness* de la generación, estos son ordenados de mejor a peor, usando el algoritmo *bubble sort* [94] (Algoritmo de ordenamiento de burbuja).

La función *SortFitnessGeneration()* retorna una lista con los *fitness* ordenados de mejor a peor. El mejor *individuo* es generado en el espacio de trabajo de *Rhino* y el algoritmo entra al bucle de optimización, hasta que el número de iteraciones es alcanzado.

```
For j = 0 To numGenerations

       rwheel = NaturalSelection(bestGeneration)
       newGeneration = Replace(rwheel, generacion)
       bestGeneration = Sort Fitness Generation(newGeneration, bit, bitSeries,
pts, radius, h, w, l)



       If (bestGeneration(0)(0) > fitToTest) Then

              Call Draw(generacion(bestGeneration(0)(1)), h, w, l, 1)
              fitToTest = bestGeneration(0)(0)
              text.WriteLine("_Best Fitness in the (" & j &") Generation: " &
fitToTest)

       End If


       For k = 0 To UBound(generacion)

              generacion(k) = newGeneration(k)
       Next

Next
```

94 Ordenamiento de burbuja (Bubble Sort en inglés) es un sencillo algoritmo de ordenamiento. Funciona revisando cada elemento de la lista que va a ser ordenada con el siguiente, intercambiándolos de posición si están en el orden equivocado. Es necesario revisar varias veces toda la lista hasta que no se necesiten más intercambios, lo cual significa que la lista está ordenada. Este algoritmo obtiene su nombre de la forma con la que suben por la lista los elementos durante los intercambios, como si fueran pequeñas "burbujas". También es conocido como el método del intercambio directo. Dado que solo usa comparaciones para operar elementos, se lo considera un algoritmo de comparación, siendo el más sencillo de implementar. http://es.wikipedia.org/wiki/Ordenamiento_de_burbuja. [2010/06/02].

La primera acción a realizarse dentro del bucle es la selección natural, (*NaturalSelection()*), la cual utiliza el método de la ruleta para seleccionar los *individuos*. La ecuación para escalar la curva de selección es la siguiente:

$$index = \left[Int\left(\frac{(i\ x\ i\ x\ FactorDecay) + 1}{MaxSelection} \right) \right] - 1$$

La variable *index* corresponde al número de veces que el *individuo* será copiado dentro de la "ruleta". *MaxSelection* es el número máximo de veces que el mejor *individuo* será seleccionado. La variable *i* corresponde al contador del bucle que comienza en 0 y acaba en N *individuos* dentro de la *población*. La variable *FactorDecay* controla la caída de la curva de selección. Al valor final de la ecuación se le resta 1, para evitar que los peores *individuos* sean seleccionados si el valor de la ecuación es 0. El resultado de esta ecuación no depende del *fitness* del *individuo*, sino de su posición dentro de la *población*. Sin embargo su posición si depende de su *fitness*.

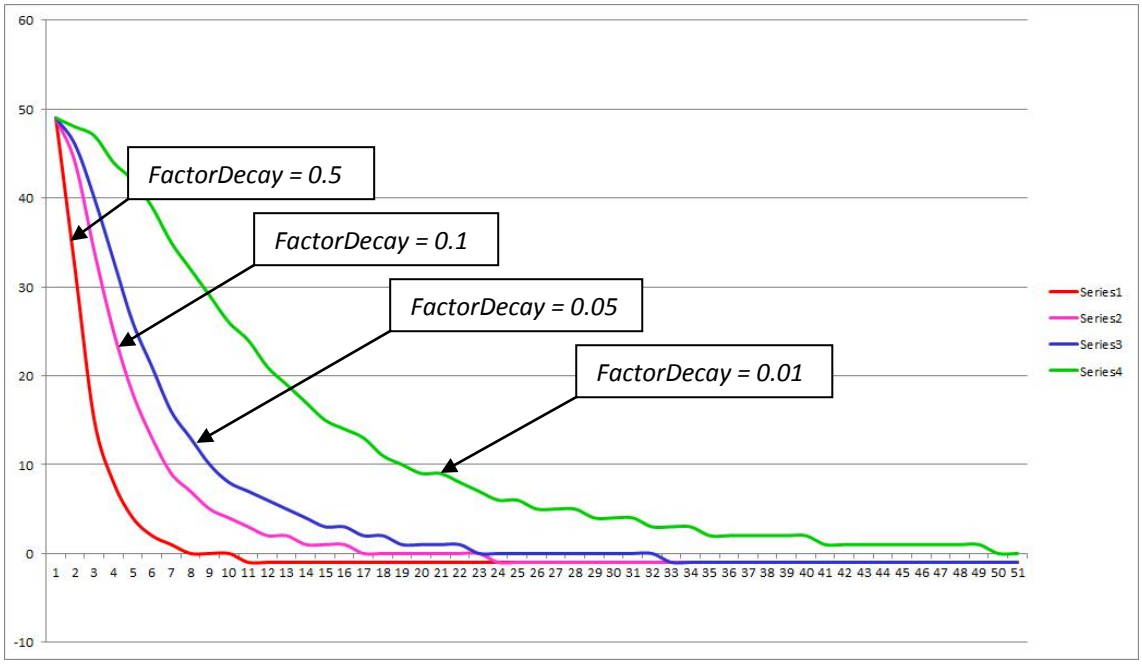


Figura 92. Curva de selección con diferentes valores en la variable *FactorDecay*

La curva roja, es la más selectiva de las 4 que aparecen en la imagen. Un factor de 0.5 selecciona sólo a los 10 primeros *individuos*, la magenta con un valor menor permite seleccionar sólo a los primeros 16 *individuos*. Un valor cercano a 0 hará la curva más suave lo cual permitirá seleccionar más *individuos*. En cambio un valor cercano a 1 hará la curva mucho más selectiva, seleccionando unos pocos.

La siguiente función en ser llamada es *Replace()*, que es encargada de seleccionar de la ruleta los mejores *individuos*. Estos serán cruzados y mutados con el fin de crear una nueva *población*. El primer paso consiste en seleccionar aleatoriamente 2 *individuos*. Los cuales

dependiendo de la elección del tipo de cruce en la ventana de configuración, se podrán cruzar mediante *SinglePtCrossOver()* o *UniformCrossOver()*.

La función *SinglePtCrossOver(),* elige al azar un índice comprendido entre 0 y el tamaño del cromosoma. Realiza el cruce entre 0 y el índice. Luego entre el índice + 1 y el tamaño de la lista del cromosoma. Debido a que el cromosoma es binario (0 y 1), no es necesario lidiar con intersecciones y coincidencias permitiendo tener dos descendencias.

```
Function SinglePtCrossOver(ByVal bitStrDad, ByVal bitStrMom)

	Randomize
	Dim crossPoint
	crossPoint = CInt(LBound(bitStrDad) + Rnd() * (UBound(bitStrDad) -
LBound(bitStrDad)))
	Dim i, j

	Dim Off1(), Off2()
	ReDim Off1(UBound(bitStrMom))
	ReDim Off2(UBound(bitStrDad))

	For i = 0 To crossPoint
		Off1(i) = bitStrDad(i)
		Off2(i) = bitStrMom(i)
	Next

	For j = crossPoint + 1 To UBound(bitStrMom)
		Off1(j) = bitStrMom(j)
		Off2(j) = bitStrDad(j)
	Next

	SinglePtCrossOver = Array(Off1, Off2)

End Function
```

El primer bucle se ejecuta de 0 al índice creado aleatoriamente (*crossPoint*). El cromosoma de un padre es copiado al hijo Off1 y el de la madre al Off2. En el siguiente bucle la herencia se invierte, y el bucle comienza en el índice + 1.

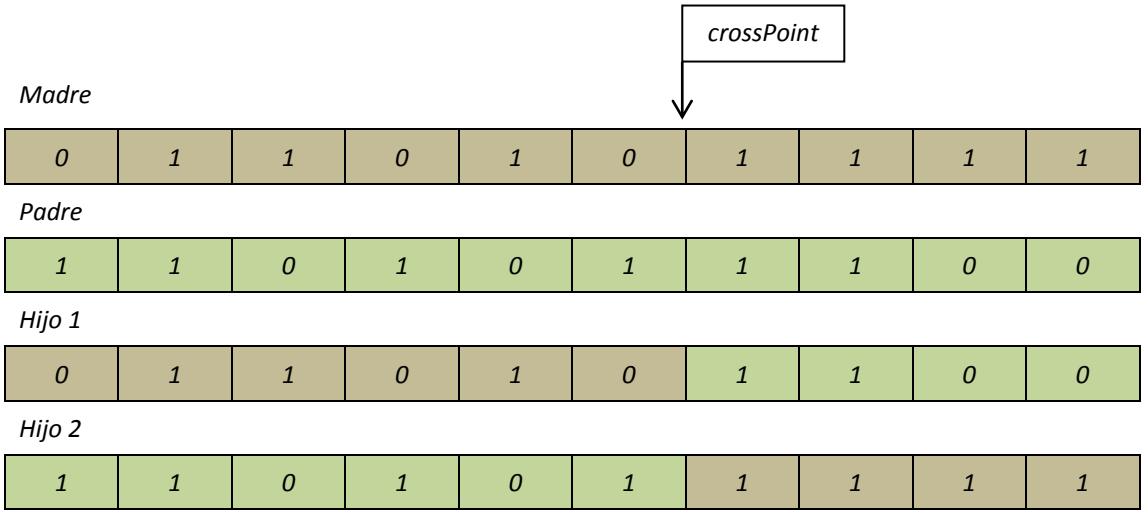


Figura 93. Tabla que muestra gráficamente el cruzamiento de un punto.

El otro tipo de cruzamiento es llamado cruce uniforme *UniformCrossOver()* y opera de la siguiente manera: Un bucle se ejecuta el mismo número de veces que el tamaño de la lista que contiene un cromosoma. El contador del bucle comienza a recorrer la lista del cromosoma y dependiendo de una condicional que retorna *true* o *false* de manera aleatoria, el padre o la madre copiaran el alelo al hijo Off1 y Off2.

```
Function UniformCrossOver(ByVal bitStrDad, ByVal bitStrMom)

	Dim i
	Dim Off1(), Off2()
	ReDim Off1(UBound(bitStrMom))
	ReDim Off2(UBound(bitStrDad))

	For i = 0 To UBound(bitStrDad)

		Randomize
		If(Rnd() < 0.5) Then
			Off1(i) = bitStrDad(i)
			Off2(i) = bitStrMom(i)
		Else
			Off1(i) = bitStrMom(i)
			Off2(i) = bitStrDad(i)
		End If

	Next

	UniformCrossOver = Array(Off1, Off2)

End Function
```

Si el número aleatorio es menor a 0.5 el Padre copiara a Off1, de lo contrario lo hará la madre. Al final de la función, habrá dos descendientes.

Madre

0	*1*	*1*	*0*	*1*	*0*	*1*	*1*	*1*	*1*

Padre

1	*1*	*0*	*1*	*0*	*1*	*1*	*1*	*0*	*0*

If(Rnd() < 0.5)

Hijo 1

0	*1*	*0*	*1*	*1*	*1*	*1*	*1*	*0*	*1*

Hijo 2

1	*1*	*1*	*0*	*0*	*0*	*1*	*1*	*1*	*0*

Figura 94. Tabla que muestra gráficamente el cruzamiento uniforme

Con este tipo de cruzamiento los cromosomas de ambos padres son repartidos aleatoria y uniformemente en ambas descendencias.

Luego de que se ha realizado el cruce, los nuevos *individuos* son retornados a la función *replace()* para ser mutados. La función *Mutation()* se encarga de modificar levemente los cromosomas de cada descendencia. El proceso comienza multiplicando el porcentaje de mutación, *Mutation rate*, por la longitud de la lista que contiene el cromosoma. Este valor determinará las veces que la mutación se realizará. Luego, mediante un puntero se elige aleatoriamente un alelo dentro del cromosoma y se intercambia el 1 por el 0 o viceversa. El puntero es guardado dentro de una lista externa para evitar mutar dos veces el mismo alelo y dejar el cromosoma como estaba originalmente. Ambas descendencias son mutadas y junto a ambos padres son guardados dentro de la nueva generación que volverá a ser evaluada por medio de la función *SortFitnessGeneration()*. El proceso se volverá a repetir hasta que el contador del bucle alcance la condicional de parada.

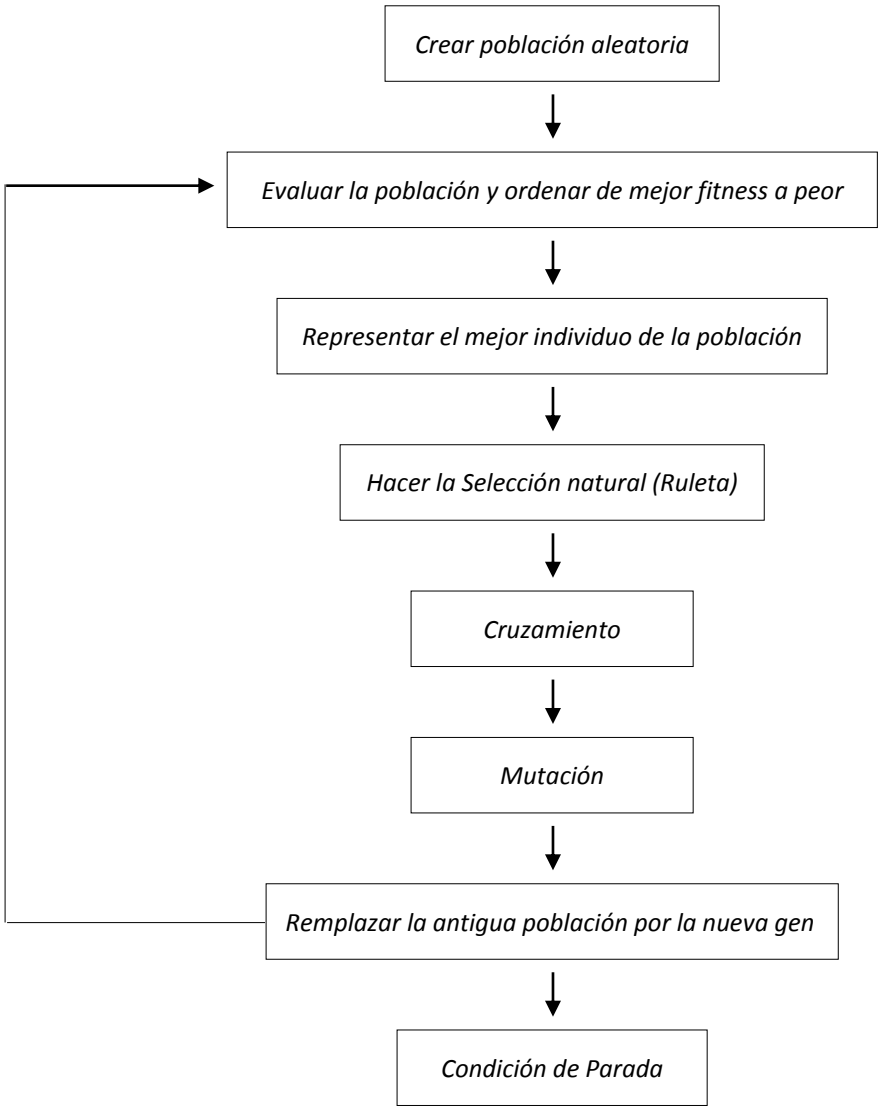


Figura 95. Representación gráfica de la arquitectura del SDG.

8.2.3. Resultados

Durante el desarrollo del algoritmo se hicieron una serie de pruebas a los operadores genéticos para detectar fallas técnicas y comportamientos con diferentes valores. El objetivo de las pruebas es eliminar errores en el código, hacerlo más rápido, rediseñar su arquitectura, etc. Usualmente se trabaja sobre un banco de pruebas (*test bed*), que sirve como plataforma para experimentar, comprobar, detectar errores y mejorar todo el sistema. Los resultados que se muestran están realizados sobre un poliedro de base10x5x5, lo que significa que el tamaño del cromosoma es de 250 bytes (110100010101). Los dos puntos que aparecen actúan como repelentes. Esto significa que los cubos que estén dentro de su área de influencia, esferas naranjas, serán penalizados en el resultado del *fitness*.

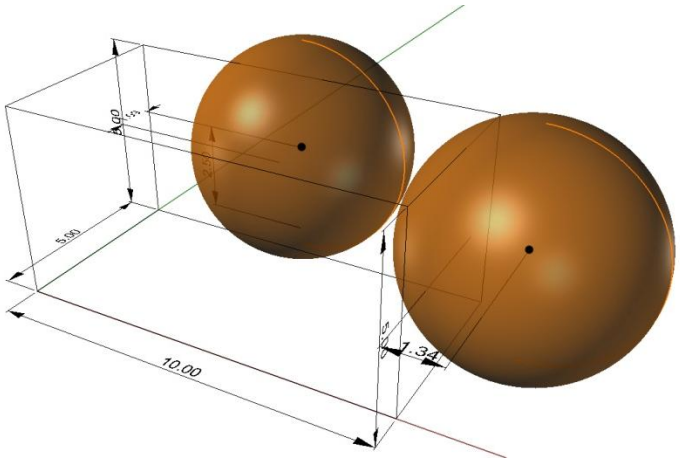


Figura 96. Perspectiva del Banco de pruebas utilizado

Los resultados de este proceso generativo, se basan principalmente en dos grandes acciones. La primera, en la manera en cómo se configura el algoritmo, y la segunda en cómo se controla y define el factor de aptitud (*fitness*).

De la configuración del algoritmo depende el tamaño de la *población*, la longitud del cromosoma, el porcentaje de mutación etc. Controlar y definir el *fitness* corresponde a como se evaluará cada uno de los *individuos*. Ambos parámetros están estrechamente relacionados e influyen directamente en el resultado final de la forma y en el espacio de búsqueda con el cual se desee trabajar.

El primer resultado que se expone cada *población* está compuesta por 20 *individuos* y el número total de iteraciones es 300. El cruzamiento se hace con un solo punto (*SinglePointCrossOver*), y el porcentaje de mutación es de 0.01%. El tipo de byte a buscar es 0 y la longitud de la secuencia de 5. El umbral de proximidad es 3 y el número de veces que es seleccionado el mejor *individuo* es de 50 veces con un factor de selección 0.7. Esto significa que solo los 8 mejores *individuos* (de 20) serán seleccionados. La penalización del *fitness* fue la siguiente:

```
If(footPrint > 15) Then footPrint = footPrint * 5
If(Volume > 200) Or (Volume < 150) Then Volume = Volume * 0.8
```

Los datos del resultado son guardados en el archivo de texto, donde se puede consultar la configuración del algoritmo y las veces que el AG optimizó y en que generación lo hizo.

_Best Fitness in the Random Generation: -37.2
_Best Fitness in the (0) Generation: -36
_Best Fitness in the (1) Generation: -34.4
_Best Fitness in the (3) Generation: -31
_Best Fitness in the (4) Generation: -27.8
_Best Fitness in the (5) Generation: -21.8
_Best Fitness in the (6) Generation: -20.2
_Best Fitness in the (7) Generation: -14.2
_Best Fitness in the (9) Generation: -12.6
_Best Fitness in the (10) Generation: -9.8
_Best Fitness in the (12) Generation: -4.3
_Best Fitness in the (13) Generation: 59
_Best Fitness in the (14) Generation: 61
_Best Fitness in the (15) Generation: 62.8
_Best Fitness in the (16) Generation: 64.2
_Best Fitness in the (17) Generation: 64.8
_Best Fitness in the (18) Generation: 65.8
_Best Fitness in the (19) Generation: 66.2
_Best Fitness in the (20) Generation: 67.2
_Best Fitness in the (22) Generation: 68.8
_Best Fitness in the (23) Generation: 69.2
_Best Fitness in the (24) Generation: 69.6
_Best Fitness in the (26) Generation: 71.2
_Best Fitness in the (29) Generation: 72.8
_Best Fitness in the (30) Generation: 74.4
_Best Fitness in the (31) Generation: 74.8
_Best Fitness in the (34) Generation: 75.4
_Best Fitness in the (35) Generation: 77.8
_Best Fitness in the (36) Generation: 78.4
_Best Fitness in the (37) Generation: 78.6
_Best Fitness in the (38) Generation: 80.6
_Best Fitness in the (40) Generation: 82.2
_Best Fitness in the (41) Generation: 83.6
_Best Fitness in the (42) Generation: 84.2
_Best Fitness in the (43) Generation: 85.6
_Best Fitness in the (44) Generation: 86.4
_Best Fitness in the (45) Generation: 87.4
_Best Fitness in the (46) Generation: 88.4
_Best Fitness in the (48) Generation: 90.8
_Best Fitness in the (50) Generation: 92.4
_Best Fitness in the (54) Generation: 92.8
_Best Fitness in the (55) Generation: 94.4
_Best Fitness in the (56) Generation: 96.8
_Best Fitness in the (58) Generation: 97.2
_Best Fitness in the (60) Generation: 97.4
_Best Fitness in the (61) Generation: 98.4
_Best Fitness in the (62) Generation: 99.8
_Best Fitness in the (64) Generation: 100.4
_Best Fitness in the (65) Generation: 102
_Best Fitness in the (67) Generation: 102.6
_Best Fitness in the (70) Generation: 103.6
_Best Fitness in the (72) Generation: 106
_Best Fitness in the (74) Generation: 108
_Best Fitness in the (75) Generation: 108.6
_Best Fitness in the (76) Generation: 108.8
_Best Fitness in the (77) Generation: 110.2
_Best Fitness in the (79) Generation: 110.8
_Best Fitness in the (80) Generation: 111
_Best Fitness in the (84) Generation: 112.6
_Best Fitness in the (92) Generation: 113.2
_Best Fitness in the (94) Generation: 115.2
_Best Fitness in the (98) Generation: 115.8
_Best Fitness in the (109) Generation: 116.4
_Best Fitness in the (110) Generation: 117.2
_Best Fitness in the (111) Generation: 146
_Best Fitness in the (114) Generation: 147
_Best Fitness in the (119) Generation: 148
_Best Fitness in the (124) Generation: 149
_Best Fitness in the (125) Generation: 150
_Best Fitness in the (168) Generation: 151
_Best Fitness in the (176) Generation: 153
_Best Fitness in the (183) Generation: 154
_Best Fitness in the (189) Generation: 156
_Best Fitness in the (236) Generation: 157
_Best Fitness in the (239) Generation: 159
_Best Fitness in the (247) Generation: 160
_Best Fitness in the (256) Generation: 161
_Best Fitness in the (278) Generation: 162
_Best Fitness in the (288) Generation: 163

Figura 97. Tabla donde se muestra el progreso del SDG. Los datos mostrados son el número de generación, donde ocurrió la optimización y el valor de la aptitud (*fitness*). Por esto es posible determinar en qué número de generación hubo optimización y en cuanto fue el incremento.

El primer *individuo* tuvo un *fitness* negativo -37.2, lo que significa que su aptitud no cumplió ninguno de los requerimientos. En cambio en la generación 288 su *fitness* había subido hasta 163. Esto significa una mejora de un 120% entre el peor *individuo* y el mejor de la última iteración del algoritmo.

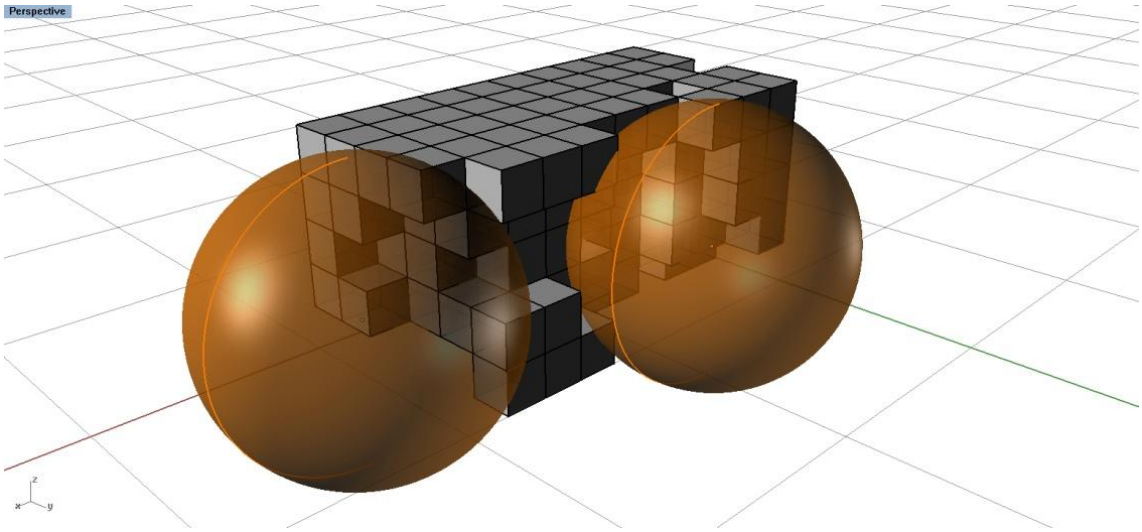


Figura 98. Imagen del resultado final (*fitness* 163). Las zonas naranjas corresponden a las áreas de influencia de los puntos sobre el volumen.

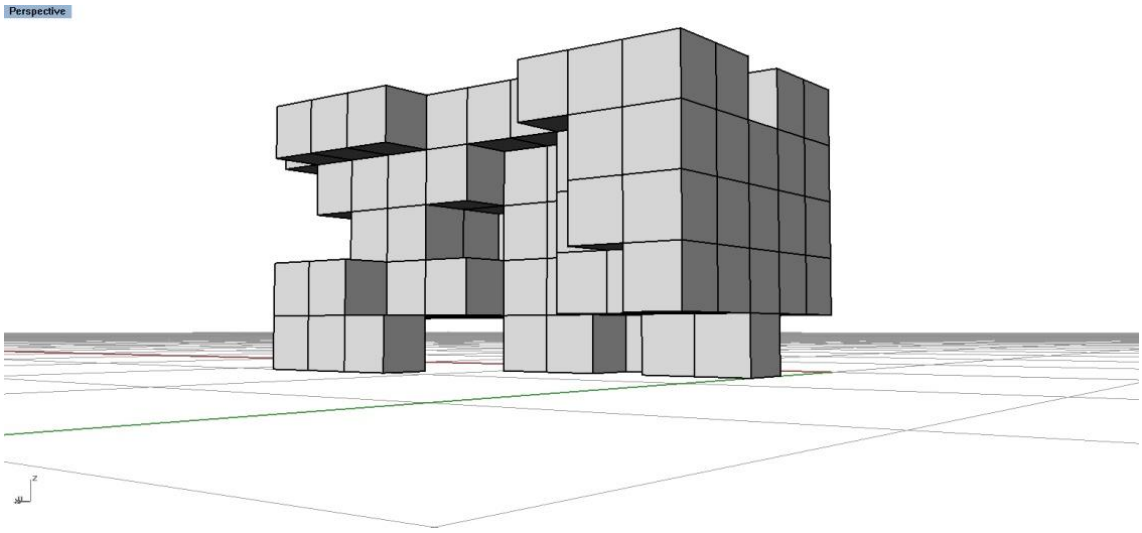


Figura 99. Diferentes perspectivas y vistas del resultado final. Vista en perspectiva, a la altura de una persona del edificio generado por el SDG. Desde este punto, es posible ver como el AG dispuso muy pocos cubos en la base para liberar la planta (*The developer's Manhattan function*) y como también no propuso cubos en las zonas de influencia de los puntos.

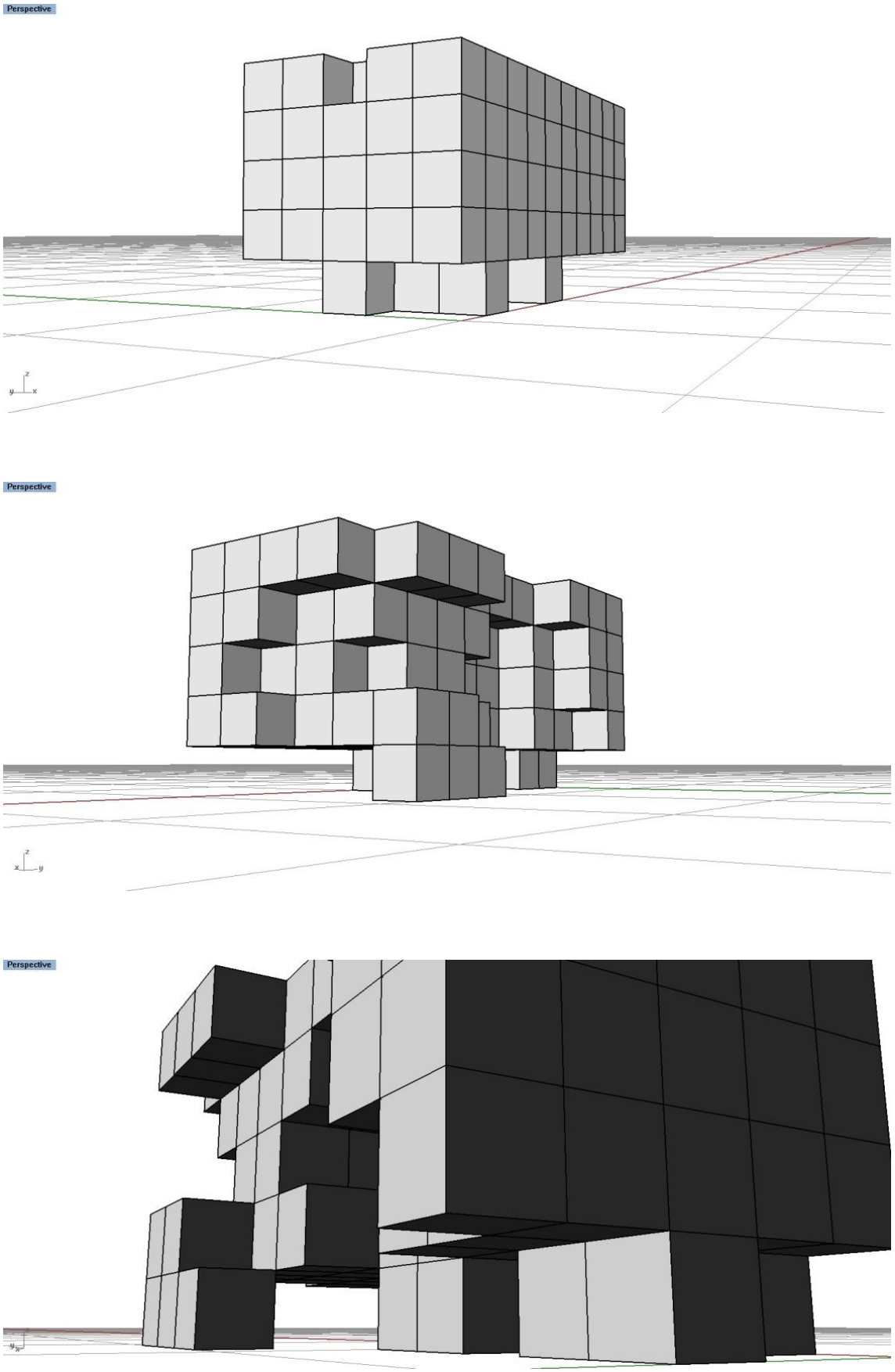


Figura 100. Diferentes perspectivas del resultado.

La siguiente serie de imágenes corresponde al proceso generación del algoritmo con la siguiente configuración. El tamaño de la *población* es de 20 *individuos*, y el número de generaciones es de 200. El tipo de crossover elegido es de un punto. El porcentaje de mutación será de 0.1. Esto es 255 x 0.1 = 25.5 (26). Lo que significa que de los 255 genes del cromosoma 26 serán intercambiados. Cada nuevo descendiente tendrá dentro de su cromosoma alrededor de un 10% de genes nuevos. Esto implica que el 10 % de la herencia de los padres se perderá. La secuencia de bytes a buscar es 1 y su longitud es de 2 unidades. El umbral de proximidad es 3 uds. La mayor cantidad de *individuos* seleccionados dentro de la selección natural es de 50 y la curva de selección de 0.5. Lo que significa que se seleccionarán los primeros 10 *individuos* de cada *población* para ser cruzados. La penalización del *fitness*, será la siguiente:

```
If(footPrint > 15) Then footPrint = footPrint * 5
```

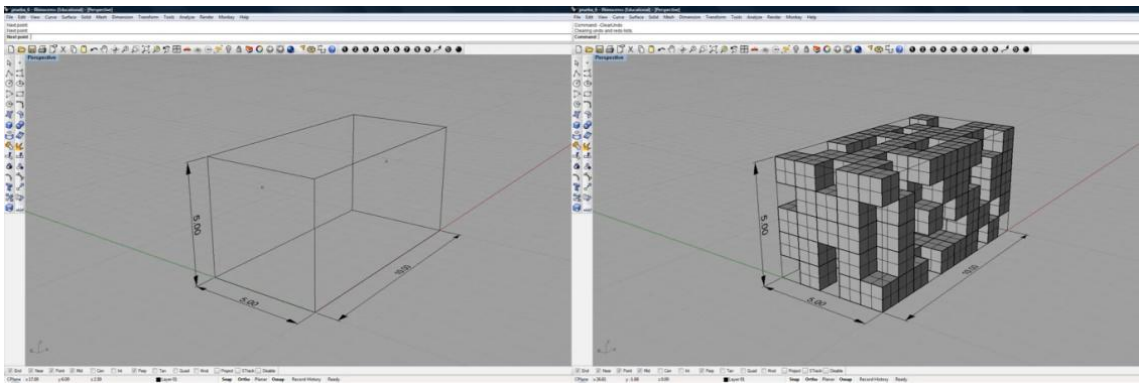


Figura 101. A la izquierda se ven los límites de la geometría y a la derecha, la generación aleatoria con un *fitness* de 113.

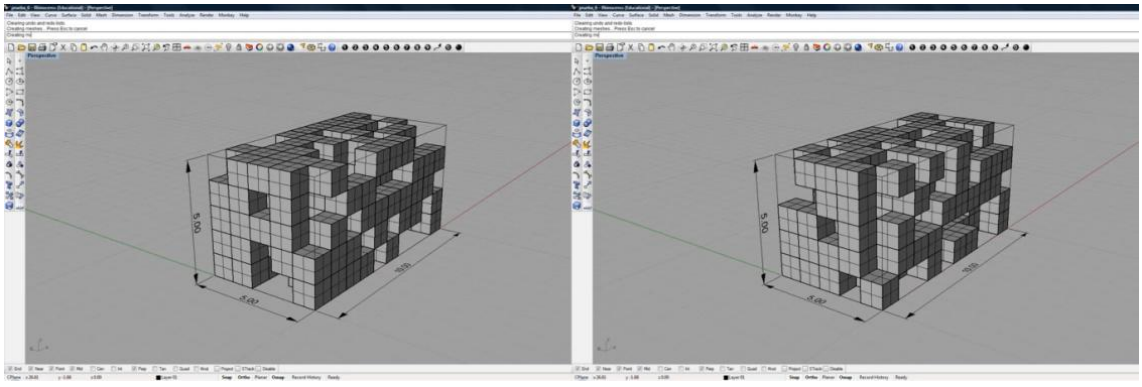
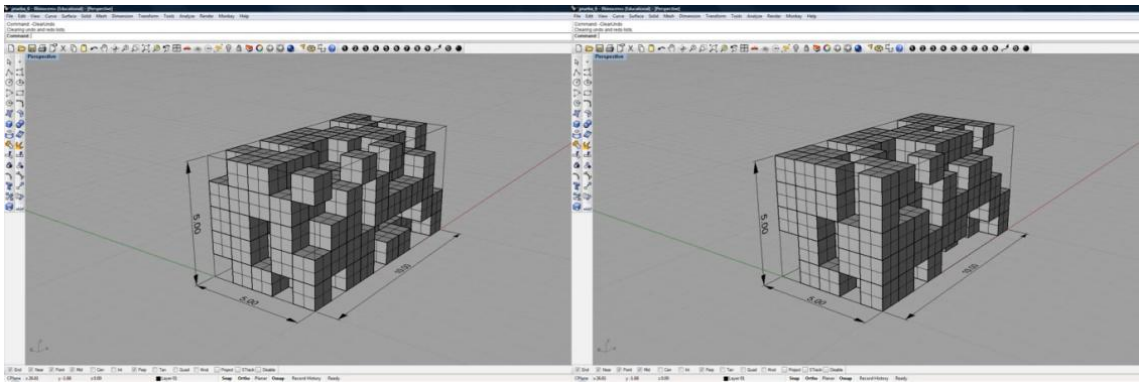
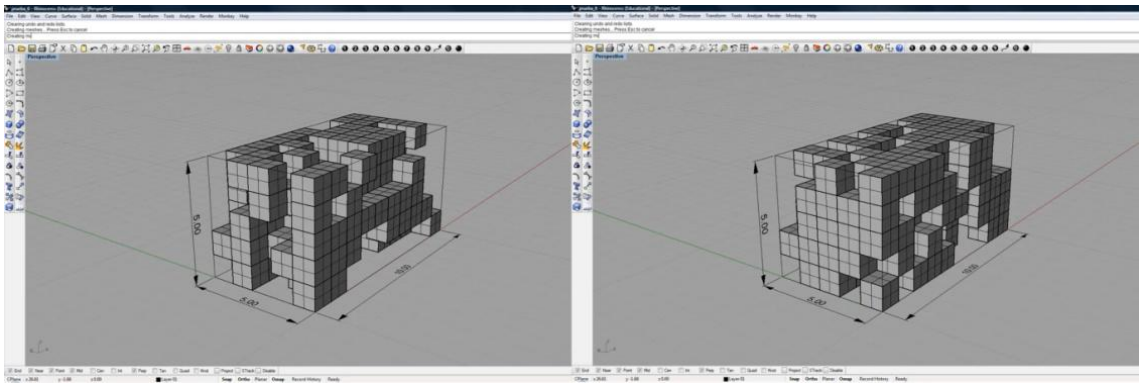
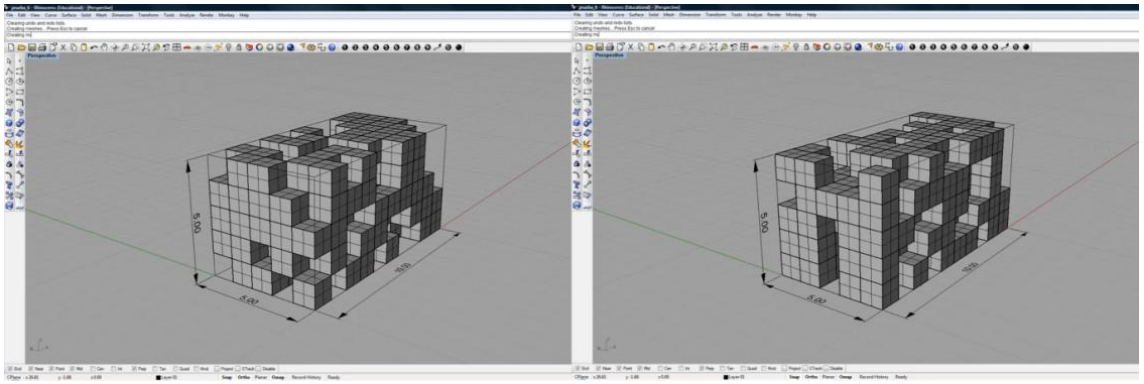


Figura 102. La generación 1 tiene un *fitness* de 115 y la generación 4 es de 119. En este momento del proceso, los *individuos* tienen una aptitud muy mala.







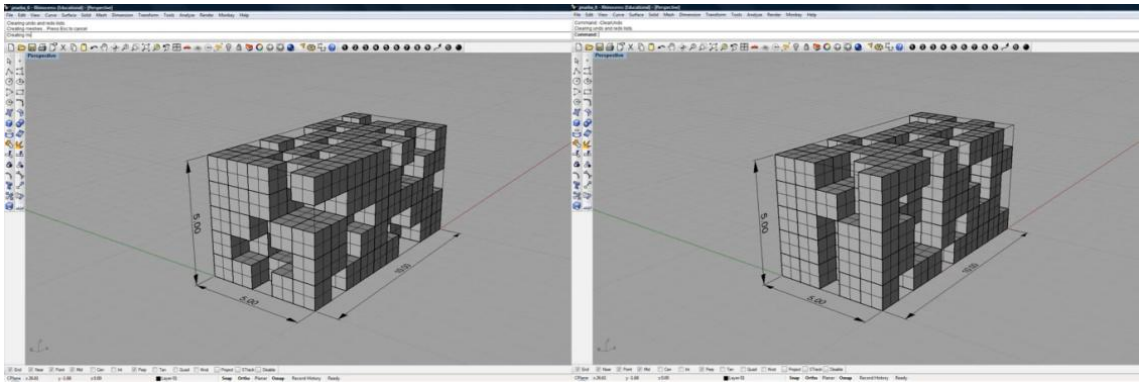
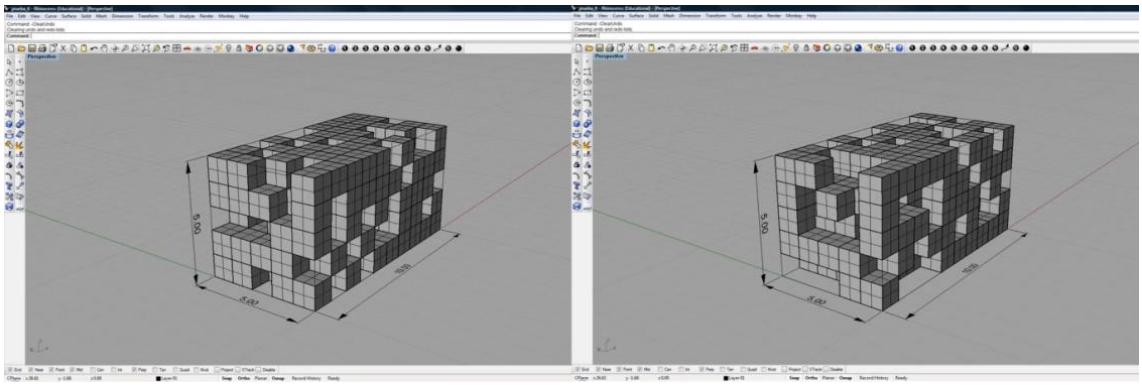


Figura 103. La secuencia de imágenes muestra la evolución del modelo dentro del AG. De izquierda a derecha las imágenes corresponden a los siguientes valores en su aptitud; 125, 128, 133, 137, 143, 145, 148, 151.



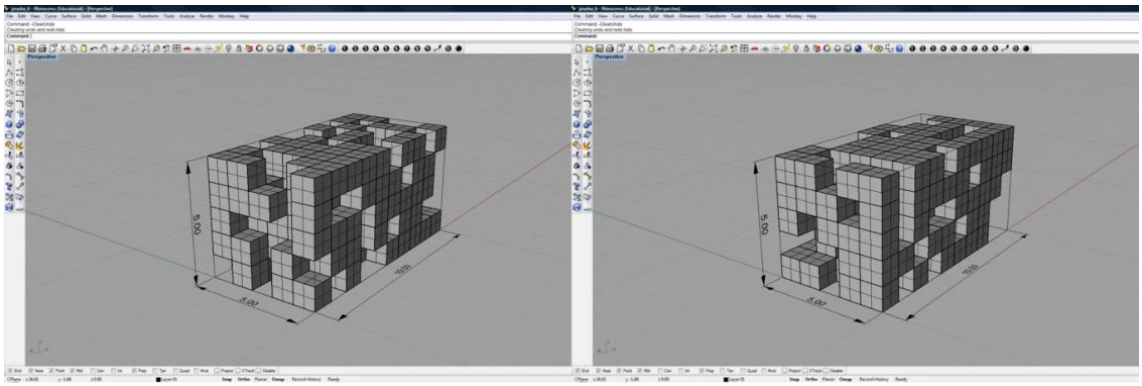


Figura 104. Los *fitness* para los *individuos* de la imagen superior son 154, 165. Para los *individuos* de las imágenes inferiores son 167 en la iteración número 49 y 170 en la 62 respectivamente. El algoritmo fue capaz de mejorar la forma 16 veces, de las 200 que hizo.

_Best Fitness in the Random Generation: 113
_Best Fitness in the (0) Generation: 115
_Best Fitness in the (1) Generation: 119
_Best Fitness in the (4) Generation: 125
_Best Fitness in the (6) Generation: 128
_Best Fitness in the (8) Generation: 133
_Best Fitness in the (10) Generation: 137
_Best Fitness in the (13) Generation: 143
_Best Fitness in the (14) Generation: 145
_Best Fitness in the (16) Generation: 148
_Best Fitness in the (26) Generation: 151
_Best Fitness in the (41) Generation: 154
_Best Fitness in the (48) Generation: 165
_Best Fitness in the (49) Generation: 167
_Best Fitness in the (62) Generation: 170

Figura 105. Tabla con los resultados de la segunda optimización.

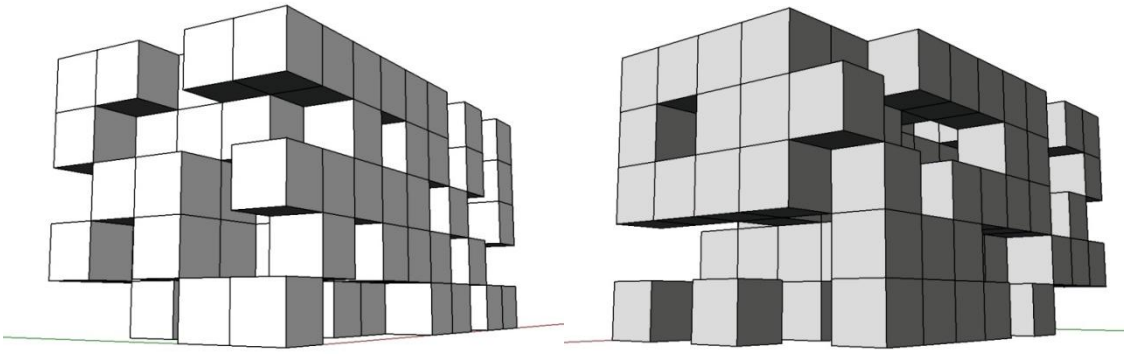
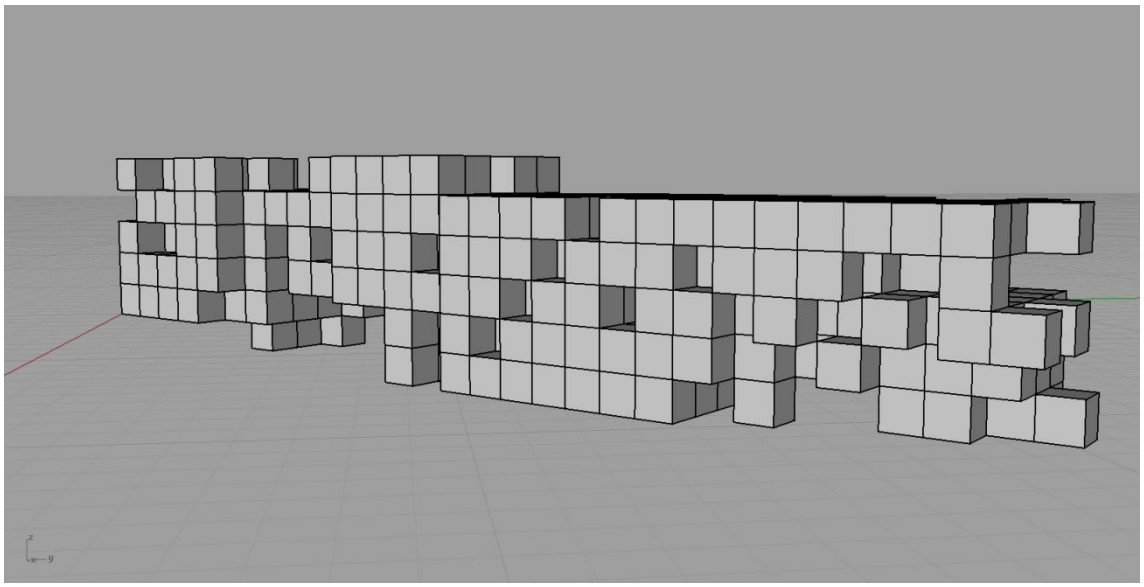
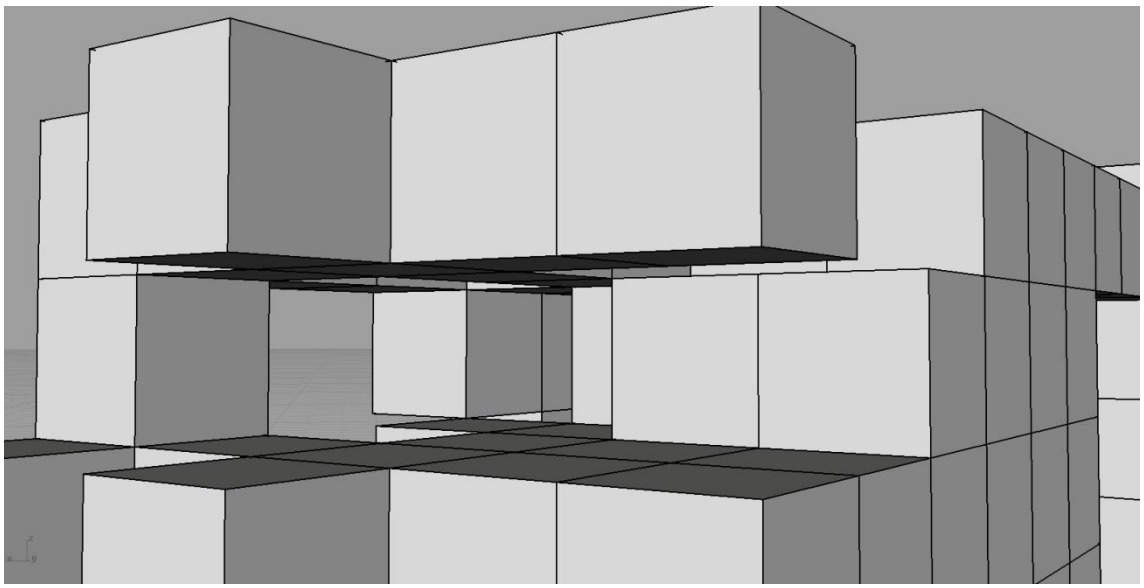
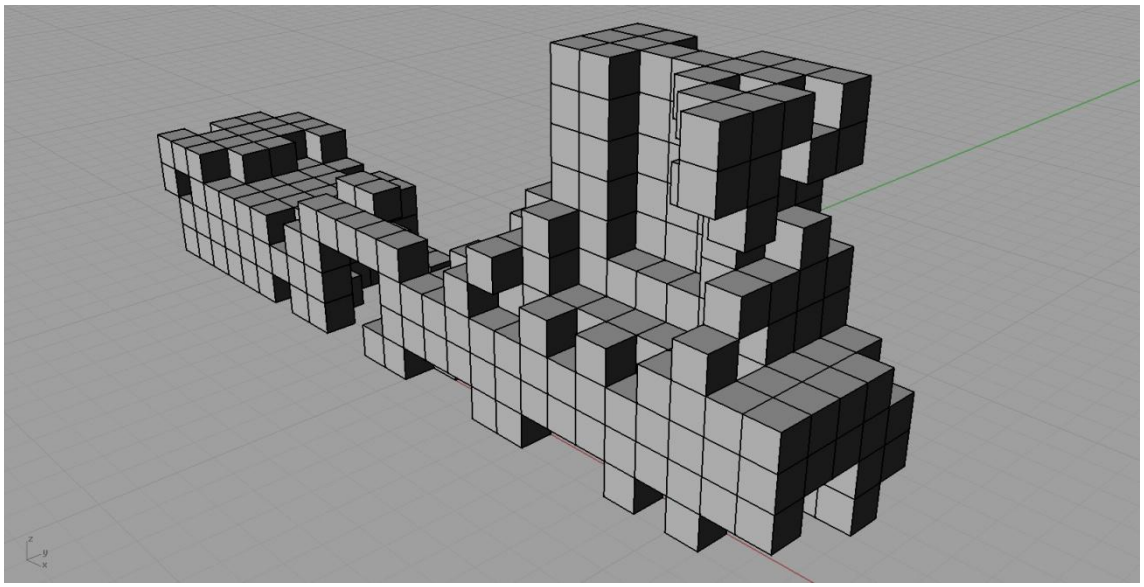


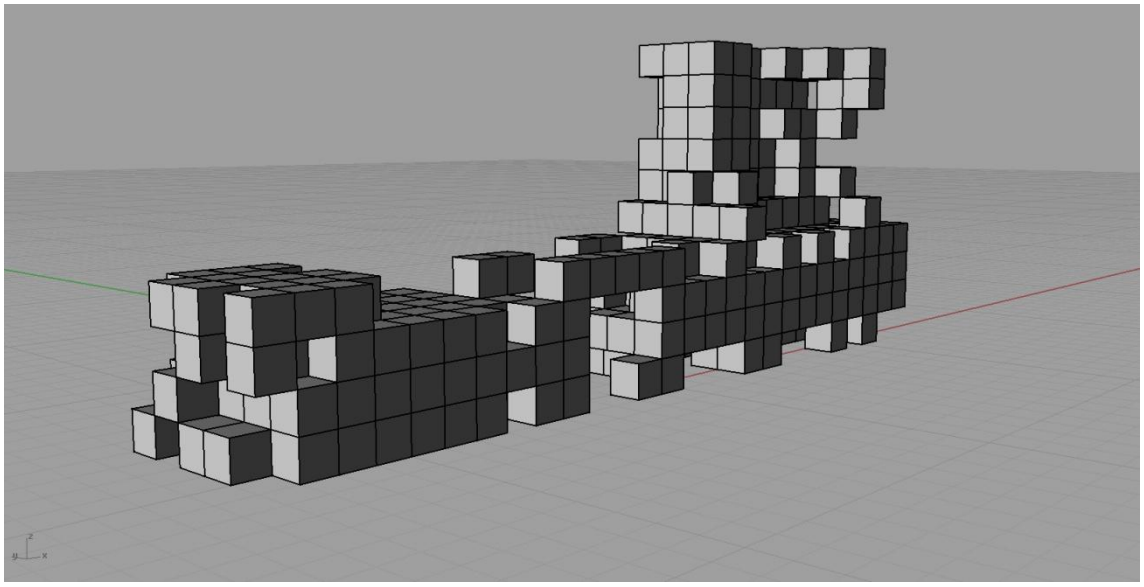
Figura 106. Perspectiva del mejor *individuo*, en la prueba 2, *fitness* 170.

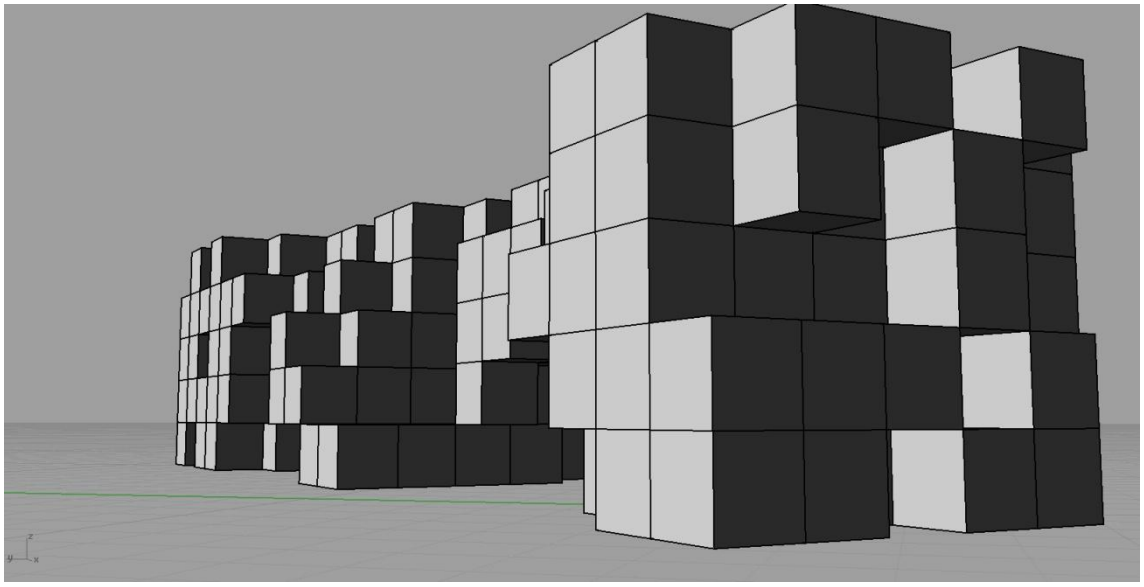
OTROS MODELOS GENERADOS CON EL SDG.











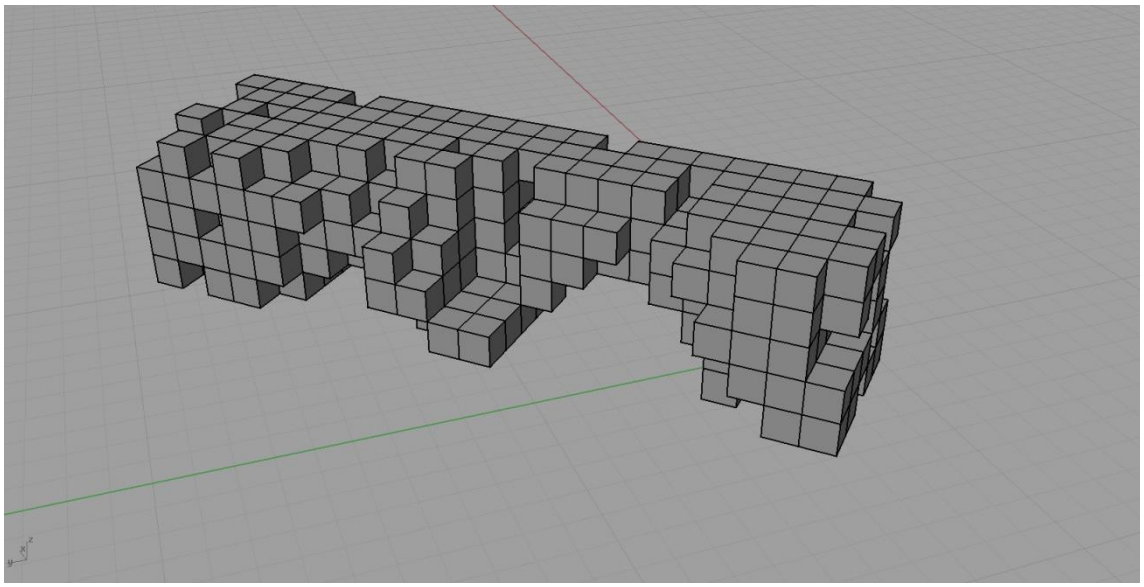


Figura 107. Secuencia de otras generaciones, usando el SDG.

8.2.4. Discusión

La diferencia en los resultados de los dos experimentos se encuentra básicamente en el porcentaje de mutación de cada descendencia. En el primer caso la descendencia fijada era de un 0.01%. Esto significa que el cromosoma de 250 bytes era mutado 3 veces (250 x 0.01). Entonces, El padre y madre heredan al hijo sus características a través del cromosoma. Luego cuando el hijo es mutado incorpora 3 nuevos genes, lo suficiente para no perder la herencia recibida de los padres y segundo continuar explorando nuevas alternativas de diseño. En el segundo resultado la mutación fue de un 0.1%, que representa un total de 26 genes (250 x 0.1) nuevo en el hijo. Esto significa que un 10% del cromosoma heredado es mutado. Esto implica que el 10% de la herencia de los padres se pierda y el modelo se optimiza en muy pocas iteraciones. Solo 16 en este caso.

Si bien por un lado una pequeña mutación permite heredar la mayoría de la aptitud de los padres se corre el peligro de caer en una solución local y perder la mejor solución global. En cambio un alto porcentaje de mutación permite buscar en todo el espacio de búsqueda, pero tiene el inconveniente de que parte de la herencia se pierda, optimizando durante muy pocas iteraciones. Equilibrar este valor y ajustarlo al tipo de problema, al cromosoma y al resultado que se quiere obtener pasa directamente por el criterio del arquitecto.

Ambos resultados cumplen las restricciones impuestas en la ecuación del *fitness*. La diferencia es que el primer experimento es configurado para que optimice tras cada iteración y el segundo para que explore diferentes caminos. Por lo tanto, ambas pruebas son compatibles y permite al arquitecto relacionarlas, unirlas o tomar lo que más le interesa de ambos resultados. Esto permite poder crear otros modelos que satisfaga otros tipos de requerimientos como por ejemplo de belleza.

El SDG parte de una forma aleatoria, pero contenida por unas directrices que el arquitecto ha definido. Se puede decir que el arquitecto ha "*rayado*" la cancha por donde el AG se desarrollará. El resultado de este proceso es una composición de cubos en un espacio de 3 dimensiones que satisface una serie de restricciones que están en conflicto entre ellas. A estos

cubos se les puede asignar una escala arquitectónica para que su total represente la forma de un edificio.

El arquitecto al configurar el SDG en búsqueda de un resultado no controla el 100% de la forma resultante sino que intuye un resultado final. Por ejemplo, cuando el granjero planta un arbusto de tomates, él sabe que son tomates y espera los tomates. Lo que no sabe el granjero es cuantos tomates serán, su tamaño, longitud de las hojas, el diámetro del tallo, etc. De la misma manera el arquitecto espera un resultado que resolverá las restricciones que ha definido, pero que no controla del todo. En este sentido, el arquitecto no es responsable de la forma sino que es responsable de un proceso de diseño, en donde la forma emerge según los parámetros establecidos.

La forma obtenida puede ser considerada de diversas maneras: como guía en el proceso de diseño, como orientación durante la primera toma de decisiones, etc. Luego el arquitecto puede aplicar sobre las formas, modificaciones, escalar, substraer volúmenes, seccionarlo, etc.

El SDG incluso puede ser visto como herramienta formal para encontrar diferentes configuraciones en el diseño y así explorar diferentes vías durante el proceso creativo.

8.3. Organización espacial mediante AG

8.3.1. Introducción

Usualmente los arquitectos pasamos horas o días encajando y ordenando el programa de actividades dentro del contorno del edificio. Se utilizan diferentes estrategias para resolver el problema. Como por ejemplo definir los pasillos y circulaciones primero para luego disponer los recintos entorno a ellos. Otra alternativa es por ensayo y error, se desarrolla un esquema básico de lo que se pretende y luego este esquema se traslada sobre el plano para asignar dimensiones y espesores precisos los cuales siempre terminan modificando el esquema ideal con que se partió. Otra opción menos recurrente, consiste en copiar un programa parecido de otro edificio y adaptarlo al cual se está trabajando. Sin embargo, sea cual sea el proceso que se use siempre se ha de preferir un recinto por sobre otro, en el sentido que se dará prioridad a una parte del programa por otra. Por ejemplo, en una vivienda las habitaciones y el estar usualmente tienen las preferencias en cuanto a las orientaciones y los m^2. Baño y cocina quedan relegados para favorecer estos recintos, por lo tanto se puede decir que ciertos recintos tienen más peso que otros al ser distribuidos.

El concepto de pesos en problemas de Inteligencia Artificial relacionados con AG se da por ejemplo en Redes Neuronales Artificiales (RNA). Las neuronas biológicas reciben y emiten mucha información simultáneamente. El proceso por el cual se comunican las neuronas es llamado sinapsis, que es la base de comunicación de todo el sistema nervioso. El proceso de aprendizaje aparece cuando las entradas a las neuronas tienen diferentes intensidades. En una RNA cada entrada tiene un peso relativo, proporcional al nivel de importancia de la entrada. Estos pesos imitan de cierta manera el proceso sináptico de una neurona biológica. Los pesos son valores que usualmente se adaptan dentro de la RNA y determinan la intensidad de la señal de la entrada dentro de la neurona. En cierta manera son la medida de como la neuronas

se relacionan entre ellas y estas medidas suelen ser modificadas en los entrenamientos de las redes neuronales.

Para este trabajo, se usa el concepto de peso para determinar la importancia que tendrán ciertas variables dentro de la función de aptitud. Estas variables no pueden ser quitadas de la función porque son parte del sistema y si bien no son tan importantes como otras variables de mayor peso, afectan en alguna mediada los resultados obtenidos por el algoritmo.

Este experimento plantea el uso de un diagrama de *Voronoi*[95] optimizado mediante un algoritmo genético. El objetivo consiste en distribuir las áreas de un programa al interior de un polígono que representa los límites del edificio. Partiendo del contorno de un edificio el arquitecto debe organizar espacialmente una serie de recintos y cumplir con los requerimientos mínimos de área para cada uno de ellos. En el primer paso el arquitecto determina los centros de los recintos y donde estarán emplazados dentro del contorno del edificio. Luego es preguntado qué áreas precisa para cada uno de los recintos y que pesos le otorgará a cada uno de ellos. Mientras más grande sea el valor de peso del recinto, menos preciso será el objetivo y mayor margen tendrá para proponer diferentes geometrías. Las áreas definidas junto con sus pesos determinaran la función de aptitud a buscar del AG. El espacio de búsqueda será entonces todas las subdivisiones espaciales posibles que puedan generarse con el número de recintos especificados por el arquitecto.

Los Diagramas de *Voronoi* son una clase de patrones que ocurren en la naturaleza de manera espontánea a cualquier escala y pueden ser descritas como una serie de células, perfectamente definidas que son todas adyacentes entre sí y ninguna célula comparte su área con otra célula. Los diagramas de *Voronoi* se construyen a partir de un conjunto de puntos en el plano euclidiano. Se empieza por uno de los puntos dentro del conjunto y se construye la mediatriz entre este punto y todos los otros puntos. La intersección de las mediatrices más cercana al punto encierran una región que es conocida como polígono de *Voronoi*, el proceso se repite hasta que se ha completado con todos los otros puntos.

95 Los polígonos de Thiessen (también llamados diagramas de Voronoi o teselación de Dirichlet) son una construcción geométrica que permite construir una partición del plano euclídeo. Deben su nombre a Alfred H. Thiessen y también fueron estudiados por Georgy Voronoi y Gustav Lejeune Dirichlet. Los polígonos de Thiessen son uno de los métodos de interpolación más simples, basado en la distancia euclidiana, siendo especialmente apropiada cuando los datos son cualitativos. Se crean al unir los puntos entre sí, trazando las mediatrices de los segmento de unión. Las intersecciones de estas mediatrices determinan una serie de polígonos en un espacio bidimensional alrededor de un conjunto de puntos de control, de manera que el perímetro de los polígonos generados sea equidistante a los puntos vecinos y designando su área de influencia.
http://es.wikipedia.org/wiki/Pol%C3%ADgonos_de_Thiessen [2010/06/07]

Figura 108. La entrada de datos para construir el diagrama de *Voronoi* consiste en una lista de puntos en un plano.

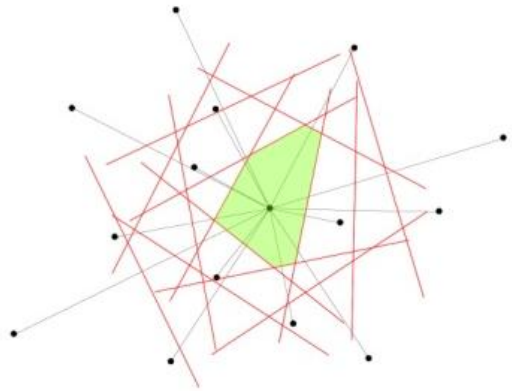


Figura 109. Se construyen todas las mediatrices entre el punto y los otros puntos. Dado un punto cualquiera dentro del conjunto, se construyen todas las mediatrices (líneas rojas) de este punto hacia todos los otros puntos. La región constituida por la intersección de las mediatrices es el polígono de *Voronoi* y representa el área de influencia de ese punto sobre el resto del conjunto. Esto significa que cualquier punto dentro del polígono, estará más cerca al punto del conjunto que ningún otro. El proceso se repite para cada uno de los puntos dentro del conjunto. El resultado final es el diagrama de *Voronoi*.

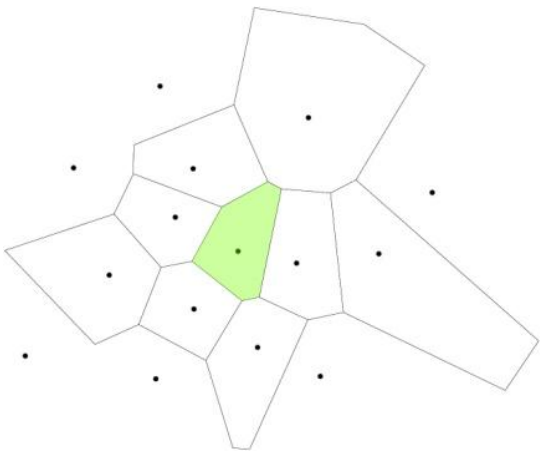


Figura 110. Diagrama de *Voronoi*.

El comportamiento de este patrón es usado por el hombre para el estudio en diferentes campos como la Biología, Antropología, Computación Científica, Marketing, desarrollo de estructuras cristalinas, etc., siempre aplicando su estrategia a problemas relacionados con la ocupación del espacio. Los Diagramas de *Voronoi* fueron considerados muy tempranamente en 1644 por René Descartes, luego fueron usados por Direchlit en 1850 para la investigación de las formas cuadráticas positivas. Pero fue Georgy Voronoi que en 1907 extendió la investigación de estos diagramas a otras dimensiones.

En el año 1854 el físico Inglés John Snow, utilizo los diagramas de *Voronoi* para un análisis de la epidemia del cólera en Londres. Snow determinó la fuerte relación que existía entre las muertes por la enfermedad y los pozos de agua (infectados) en las calles de la ciudad. Desde el punto de vista matemático, el diagrama de *Voronoi* depende de un punto, y de su área de influencia, que contiene todas las coordenadas que están más cerca del punto en cuestión que de los otros puntos. Está área es lo que se conoce como célula de *Voronoi*.

El diagrama de *Voronoi* es la imagen completa, y contiene el número total de células que es igual al número de puntos analizados. Cada célula del diagrama está perfectamente delimitada por sus bordes y todas son perfectamente adyacentes entre sí delimitando con precisión las áreas de influencia de cada punto. Los vértices de las células son compartidos dependiendo del número de células vecinas. Otra característica importante es que las celdas de *Voronoi* son siempre convexas, al igual que ocurre en la naturaleza. Las células de *Voronoi* pueden tener infinitas superficies si se modifican los puntos que constituyen las células. Los patrones que emergen pueden ser celdas cuadradas, hexagonales, patrones cristalinos, o en forma de roca. Aunque siempre el sistema otorga como resultado el mínimo confinamiento.

El uso de diagramas de *Voronoi* usualmente está referido a problemas relacionados con espacio y áreas de influencia. Por ejemplo, el emplazamiento de restaurantes de la misma cadena en una ciudad se decide según un previo estudio basado en diagramas de *Voronoi* para que los restaurantes no compitan entre sí y todos tengas similares áreas de influencia. Dentro de un diagrama de *V.* todos los límites de los polígonos son tangentes entre si y están estrechamente relacionados con el punto del diagrama. Es por esto que si un punto es movido dentro del diagrama, el área se modificará en la celda del punto y en todas las otras adyacentes a ella.

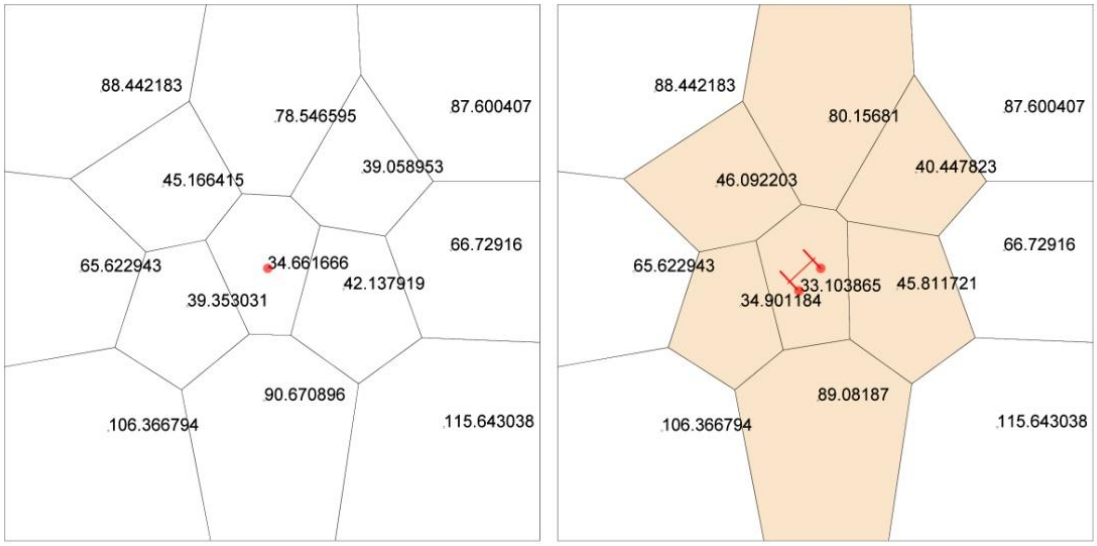


Figura 111. En la imagen de la derecha aparecen señaladas las celdas que fueron modificadas por el desplazamiento del punto rojo. Lógicamente el área de las celdas también cambio.

El cromosoma de cada *individuo*, dentro del AG, consiste en la lista de puntos del diagrama de *V*. Si en la figura 111, el punto rojo es movido para buscar una solución que mejore el *fitness*. Su desplazamiento influirá en el área de todas las celdas de su alrededor, pudiendo perjudicar el objetivo alcanzado. Entonces el problema que surge consiste en que si bien por un lado se está optimizando y mejorando el *fitness*, por otro se está perdiendo. Es por esto que se ha trabajado con un sistema de pesos para equilibrar la función de aptitud.

8.3.2. Método

El proceso está basado en un diagrama de *Voronoi*, el cual es optimizado mediante un AG para organizar espacialmente la distribución de recintos y áreas sobre la planta de un edificio. El cromosoma de cada *individuo* está representado por el conjunto de puntos que equivale al centro de cada recinto a distribuir.

ESTRUCTURA DEL ALGORITMO

El primer paso del algoritmo consiste en pedirle al arquitecto la dirección de un archivo de texto, para guardar la información de la optimización y los datos de configuración del AG. Luego el mismo mensaje se repite para un archivo de Excel, el cual es usado para configurar toda la información del AG. La hoja está dividida en 4 partes, la primera corresponde a las áreas calculadas sobre el diagrama de *Voronoi*. Estas son impresas en los puntos de cada celda en el espacio de trabajo. Con esto se puede comparar la posición de las áreas con lo que será el objetivo a optimizar y sus relaciones geométricas entre celdas. La siguiente parte hace referencia al área objetivo para cada una de las celdas, los umbrales máximos y mínimos que tendrán y el peso con el cual el AG evaluara cada cromosoma. La tercera parte, de color naranja es la propia configuración del AG. Finalmente el gráfico representa el área objetivo, los umbrales mínimo y máximo, y el área después de la evaluación.

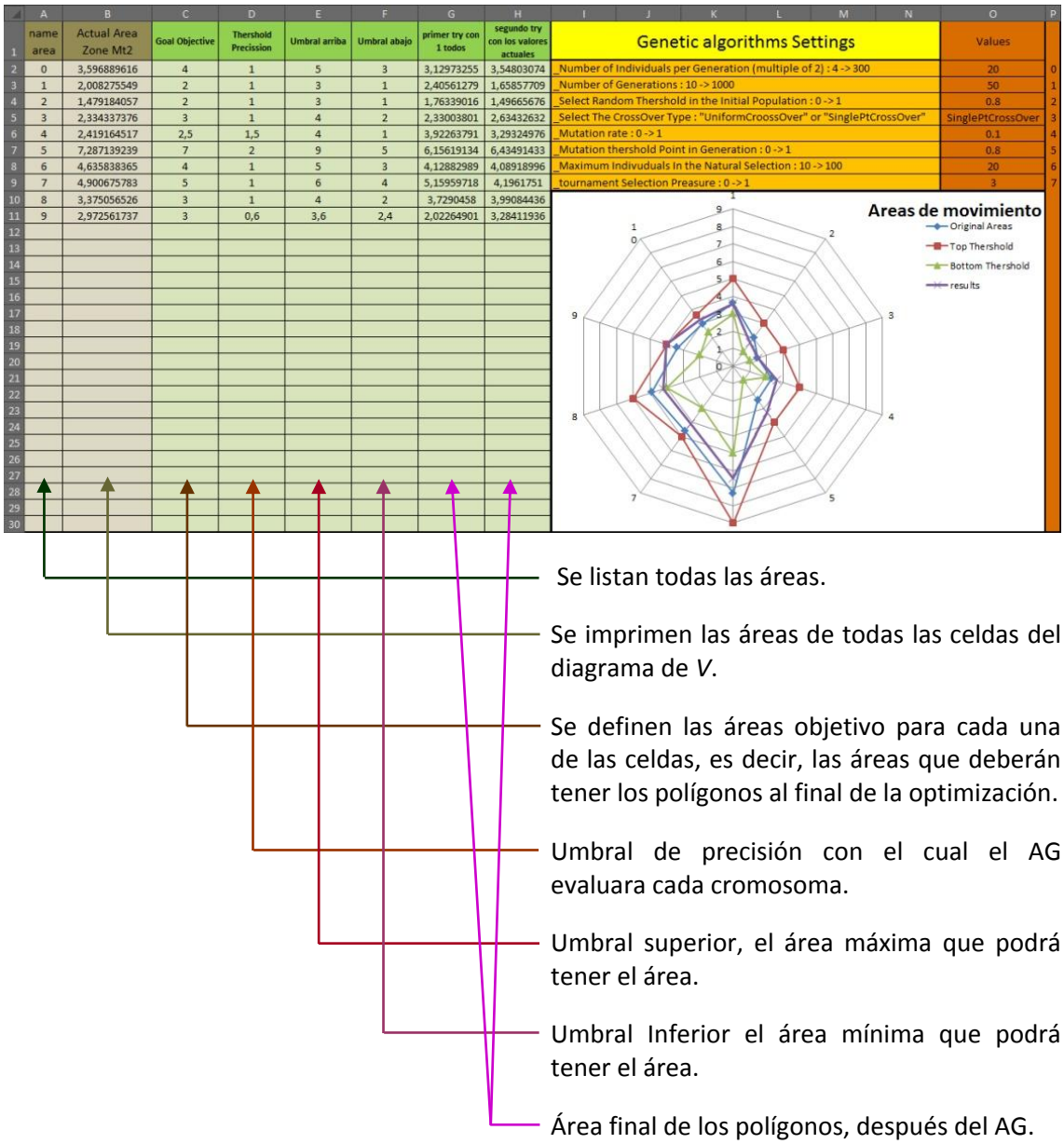


Figura 112. Ventana de configuración del AG. En la imagen aparece señalado la zona que controla los pesos para cada una de las áreas.

La zona naranja corresponde a la configuración del algoritmo genético. En orden descendente está el número de *individuos* por generación (*Number of Individuals per Generation*), el número de generaciones (*Number of Generations*), el umbral de aleatoriedad de los puntos en la primera generación (*Select Random Threshold in the Initial Population*), el tipo de cruce (*Select The CrossOver Type*), el porcentaje de mutación (*Mutation rate*), el umbral de mutación de las coordenadas de los puntos (*Mutation threshold Point in Generation*), el número máximo de *individuos* seleccionados en la selección natural (*Maximum Individuals in the Natural Selection*), y el campo (*Tournament Selection Preasure*). El cual es el encargado de controlar la presión de selectividad. Los campos: *Select Random Threshold in the Initial Population*, *Mutation threshold Point in Generation*. Definen el mismo valor y actúan de la misma manera.

La única diferencia está en que son usados en diferentes partes del algoritmo, el primero mientras se crea la *población* aleatoria y el segundo mientras la mutación se realiza.

Antes de desplazar un punto durante el proceso de optimización, una serie de cálculos son realizados. El primero de ellos, consiste en encontrar la distancia entre el punto a desplazar y el punto más cercano perteneciente al polígono. Para esto se usa la función de *Rhinoscript Rhino.CurveClosestPoint()*, la cual retorna el punto más cercano al polígono. Seguido de esto se calcula la distancia entre ambos puntos.

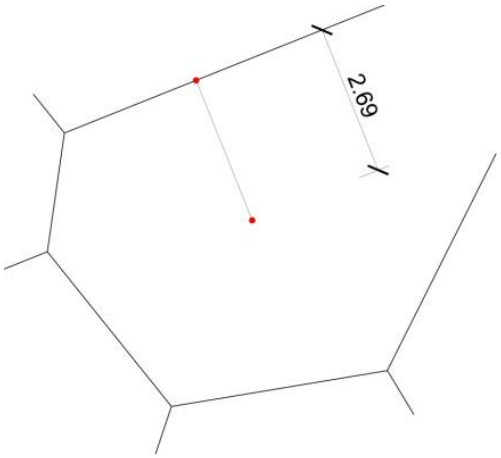


Figura 113. Distancia entre el punto a desplazar y el punto más cercano perteneciente al polígono.

El siguiente paso consiste en multiplicar la distancia por el valor ingresado en el campo que controla el umbral de mutación de las coordenadas de los puntos (*Select Random Threshold in the Initial Population* y *Mutation threshold Point in Generation*). El producto es incorporado en la siguiente función.

$$(coordX - prod) + Rnd \text{ x } ((coordX + prod) - (coordX - prod))$$

El producto es restado y sumado para encontrar un valor aleatorio máximo dentro del umbral. Si el campo es definido con un valor cercano a 0, el punto no tendrá mucho margen para moverse. Por el contario si su valor se acerca a 1 el margen será más amplio.

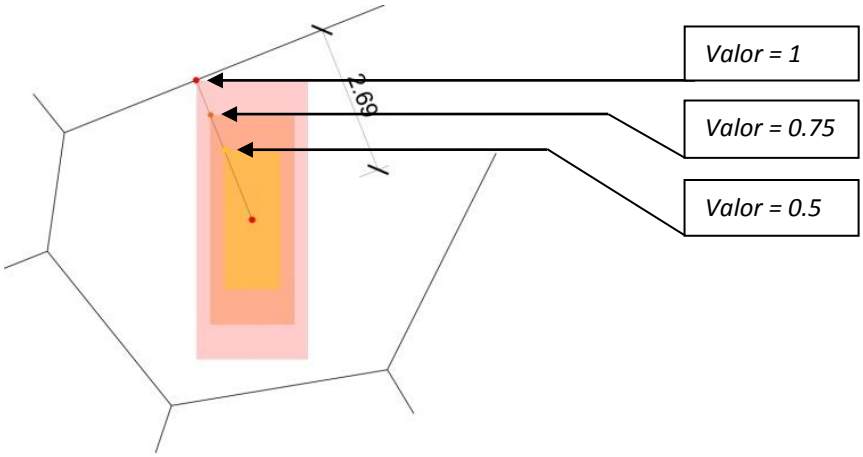


Figura 114. Áreas de movimiento del punto, según el valor del campo umbral de mutación.

Este campo se aplica por igual a todos los puntos, pero en 2 estados diferentes del algoritmo. La primera vez se aplica cuando el AG crea la primera *población* aleatoria y la otra cada vez que la mutación es aplicada a un descendiente. La última parte de la ventana de configuración muestra gráficamente las diferentes áreas involucradas dentro del proceso de optimización. La curva interior muestra el umbral mínimo establecido y la superior el umbral máximo. Las interiores hacen referencia a la situación generada por el diagrama de *V*. y la otra al resultado final de áreas después del proceso de optimización.

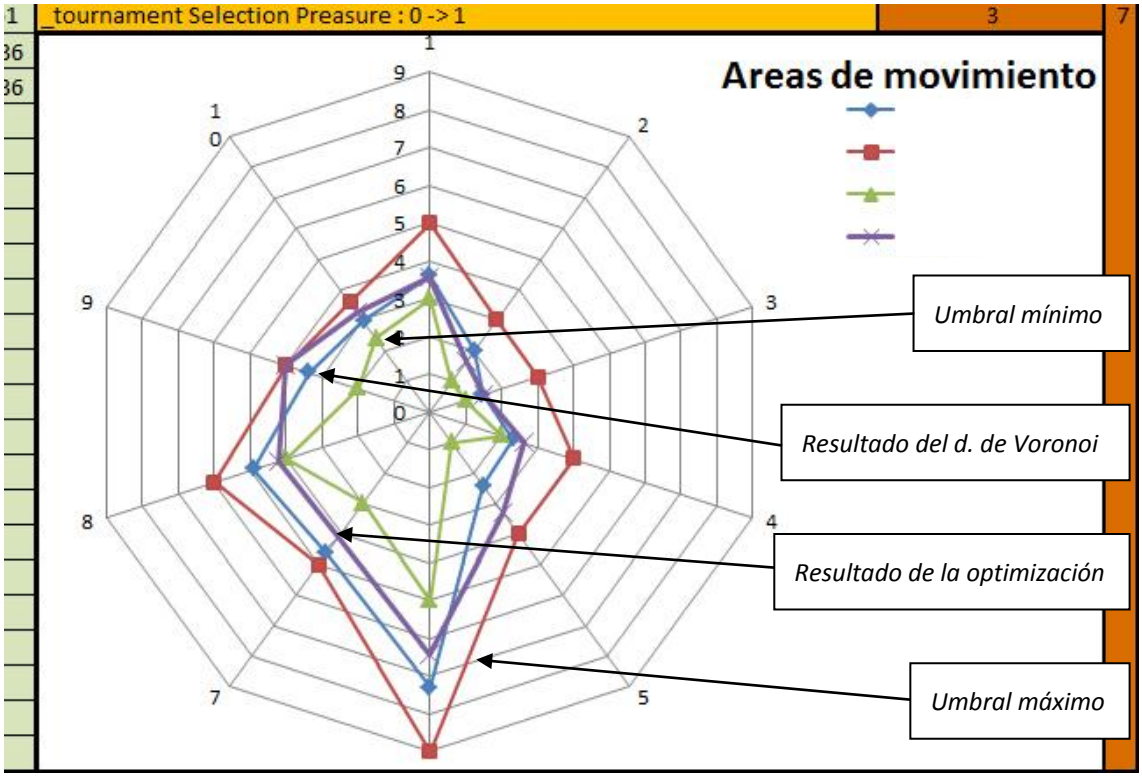


Figura 115. Gráfica de áreas, la curva verde es el umbral mínimo, la morada el resultado del diagrama de *Voronoi*, la curva azul el resultado de la optimización, y la curva roja el umbral máximo.

Una vez que los datos han sido ingresados, el algoritmo crea la *población* aleatoria. Los puntos son desplazados y movidos aleatoriamente como fue descrito anteriormente.

```
For i = 0 To GaSettings(0) - 1
      initPopulation(i) = RandomPopulation(firstInd, GASettings(2))
Next
```

El primer parámetro pasado dentro de la función *RandomPopulation()* es *firstInd* el cual consiste en una lista de dos dimensiones. En la primera dimensión tiene la lista de puntos y en la segunda la de polígonos. El segundo parámetro es *GASettings(2)* corresponde al umbral de mutación definido en la ventana de configuración de Excel.

La siguiente función en ser llamada es *funcEvaluationAndSortedList()*, la cual es encargada de evaluar la *población* y ordenar de mejor a peor los *individuos* dentro de una nueva lista.

```
Dim BestGeneration
BestGeneration = funcEvaluationAndSortedList(initPopulation, BBox,
fitnessGoal, precission, GaSettings)
```

Los parámetros pasados son los siguientes, *initPopulation* es la *población* recién creada completamente aleatoria. *BBox* es el contorno del edificio, *fitnessGoal* es la lista donde se encuentran las áreas objetivo que deben alcanzar los polígonos. Esta lista corresponde a la tercera columna de la ventana de configuración y sus valores fueron definidos por el arquitecto. El siguiente parámetro es *precission*, y como dice su nombre es la precisión con la que el AG buscará encontrar las áreas. Finalmente el parámetro *GaSettings*, corresponde a la última columna, de la configuración del AG.

Dentro de la función, el primer paso es evaluar cada uno de los *individuos*. Por cada *individuo*, previamente creado aleatoriamente se construye un diagrama de *Voronoi*, mediante la función *Polygon()*. Este retorna una lista de dos dimensiones, igual en estructura que *firstInd*. Luego es llamada la función *FitnessEvaluation()*, la cual evalúa cada una de las áreas dentro del *individuo*. El primer procedimiento dentro de esta función consiste en calcular el área del primer polígono. A esta área se le resta su área objetivo y si el resto es menor al peso que tiene esa área, el valor es premiado de lo contrario es castigado con un 1.

El objetivo es que la resta de áreas sea la menor posible y tienda a 0. Pero se busca que el *fitness* tenga el valor lo más alto posible. Por esto al final de la función se divide 10 entre *count*, por lo que mientras más cerca de 0 esté, más grande será el valor. Las áreas están calculadas con 6 decimales por lo que una división por cero en todos los polígonos es casi imposible.

```
Function fitnessEvaluation(individual, arrAreaGoal, arrPrecision)

      Dim i
      Dim count : count = 0
      Dim area, resta
      Dim reward

      For i = 0 To UBound(individual)
```

```
            area = Rhino.CurveArea(individual(i)(1))(0)
            resta = Abs(area - arrAreaGoal(i))

            If(resta <= arrPrecision(i)) Then
                  reward = arrPrecision(i)
            Else
                  reward = 1
            End If
            count = (count + resta) / reward
      Next

      fitnessEvaluation = 10/count

End Function
```

La variable *Individual* es la lista de puntos generados aleatoriamente en la función *RandomPopulation()*, *arrAreaGoal* es la lista de áreas objetivo y *arrPrecision* es la lista de valores que definen la precisión con la que se evalúa cada área. La variable *count*, va almacenando el *fitness* de cada polígono. El retorno de la función *FitnessEvaluation()* es el *fitness* del *individuo*. Con la evaluación de las áreas, es cuando se construye el cromosoma con todas las características necesarias para poder ser seleccionado, cruzado y mutado. El cromosoma presenta la siguiente estructura:

0			*1*
0	*Punto 1*	*Polígono 1*	*Valor del fitness*
1	*Punto 2*	*Polígono 2*	
2	*Punto 3*	*Polígono 3*	
3	*Punto n + 1*	*Polígono n + 1*	

Figura 116. Esquema de la estructura del cromosoma

El cromosoma de cada *individuo* está constituido por una lista dentada[96] (*jagged array*), la cual en su primer nivel está dividida en dos. El primer elemento guarda la lista de todos los puntos y polígonos, mientras que el segundo guarda el valor del *fitness*.

Una vez que se han evaluado todos los *individuos* de la *población*, son ordenados por el método de la burbuja y esta *población* es retornada por la función *funcEvaluationAndSortedList()*.

Los siguientes pasos son bastante simples. Primero, se guarda el mejor *individuo* en una variable llamada *theBest* y luego se borran del espacio de trabajo todos los *individuos* excepto el mejor, el cual ha quedado primero en la lista.

96 Una lista dentada, es un tipo de lista de dos o más dimensiones en la cual cada una de las dimensiones tiene un tamaño diferente. Usualmente las listas dentadas guardan dentro de ellas otras listas.

```
Dim theBest
theBest = bestGeneration(0)
Call Rhino.EnableRedraw(False)

For i = 1 To UBound(bestGeneration)

	Call Delete(bestGeneration(i)(0))

Next
```

Luego se entra al bucle del AG, el cual se ejecutará la cantidad de veces que fue definido en el campo *Number of Generations* de la ventana de configuración. Dentro del bucle el siguiente paso es seleccionar a los mejores *individuos* para que sean cruzados y mutados y remplacen a la antigua *población*. En este punto un nuevo método de selección es probado e implantado.

La selección por torneo, consiste en realizar la selección en base a comparaciones directas entre *individuos*. En este caso se utiliza una selección del tipo determinista. Esto significa que se seleccionarán al azar P *individuos*, de los cuales pasara a la siguiente generación el más apto de ello. Si el torneo incluye muchos *individuos*, la presión de selección es elevada y los peores casi no tienen oportunidad de ser elegidos. Por el contrario cuando el tamaño del torneo es reducido, la presión de selección disminuye y los peores *individuos* tienen más oportunidades de ser seleccionados.

La función siguiente en ser llamada es *TournamentSelection()* y toma como parámetros, la generación de *individuos* ordenados de mejor a peor, la cantidad de *individuos* seleccionados (*Maximum Individuals in the Natural Selection*) y la cantidad de *individuos* que competirán por torneo,(*tournament Selection Preasure*) que también regula la presión de selección.

La arquitectura de esta función es muy simple, primero se seleccionan *individuos* aleatorios dentro de la *población* para competir. El torneo consiste en evaluar los *fitness* de cada uno de ellos y retornar el mejor *individuo*. Este es guardado dentro de una lista llamada *TempNaturalSelection*. Existen dentro de la función ciertos trozos de código para evitar que compitan los mismos *individuos* en torneos diferentes y liberar cierta memoria atrapada por las listas. El proceso se repite hasta que *TempNaturalSelection* es llenada completamente con los ganadores de cada torneo.

La función *Torneo* es muy simple, básicamente retorna el *individuo* con el mejor *fitness*. Cuando la función *TournamentSelection()* se ha completado con la lista de los ganadores de los diferentes torneos, los *individuos* están listos para ser cruzados.

```
Function Torneo(ByVal arrList)

	Dim test: test = arrList(0)
	Dim i
	For i = 1 To UBound(arrList)
		If (test(1) < arrList(i)(1)) Then
			test = arrList(i)
		End If
	Next

	Torneo = test

End Function
```

El proceso continúa llamando a la función *replace()*, esta toma como argumentos los *individuos* seleccionados durante el torneo, el tamaño de la *población* que se va a crear y el mejor *individuo* de la generación anterior, en este caso, el más apto de la generación aleatoria.

La primera tarea, consiste en "limpiar" el cromosoma de cada *individuo*, quitándoles su valor de *fitness*, y su lista de polígonos, para dejarlos solamente con la lista de puntos. Esto se hace porque de esta manera será mucho más fácil manipularlos desde ahora para realizar el cruce y la mutación. La función encargada de realizar esta tarea se llama *cleanRwheel*() y toma como parámetro la lista de *individuos*.

Una vez limpiado los cromosomas, se eligen al azar dos padres dentro de la lista que serán cruzados y retornaran dos descendencias llamadas *offspring*. La elección del tipo de cruzamiento es hecho en la ventana de configuración del AG y dos posibles opciones pueden ser elegidas. Cruce uniforme o cruce de un punto. Con respecto al cruce de un punto el puntero es elegido aleatoriamente dependiendo del tamaño del cromosoma.

```
If(selectCO = 1) Then
  offSpring = SinglePtCrossOver(cleanRW(indexDad),cleanRW(indexMom))
Else
  offspring = UniformCrossOver(cleanRW(indexDad),cleanRW(indexMom))
End If
```

Luego se realiza la mutación, que es aplicada a ambas descendencias. La función *mutation()* toma como parámetros una descendencia y el mejor *individuo* de la *población* para desplazar de manera aleatoria alguno de los puntos dentro del cromosoma del *individuo*.

```
son1Mut = mutation(offSpring(0), areas)
son2Mut = mutation(offSpring(1), areas)
```

Para completar el proceso los dos padres seleccionados y ambas descendencias mutadas remplazan la antigua *población*. Una vez más será llamada la función *funcEvaluationAndSortedList()*, la cual evaluará la nueva *población* creada. Antes de acabar el loop del AG y todo vuelva a repetirse, una serie de acciones son realizadas. En primer lugar se compara el mejor *individuo* de la nueva *población* con el de la antigua, si existe una mejoría este remplaza al antiguo como punto de comparación y es usado para producir los cambios aleatorios dentro de la mutación.

```
If(bestGeneration(0)(1) > theBest(1)) Then
       Call Delete(theBest(0))
       theBest = bestGeneration(0)
End if
```

Luego se ejecuta la condicional que regula la detención del algoritmo. Si el número iteraciones se ha alcanzado, se detiene el AG.

```
If (j = GaSettings(1) - 1) Then
      Call Rhino.MessageBox("Maximum number of iteration reached", 16,
strTitle)
      Exit Do
End If
```

8.3.3. Resultados

Los resultados corresponden a 8 plantas todas diferentes entre sí y con diferentes cantidades de recintos a encajar. En cada una de las 8 pruebas el AG fue desafiado a encontrar diferentes áreas y algunas veces áreas que eran imposibles de encajar dentro de los límites del edificio.

PLANTA 1:

El número de *individuos* fue definido en 32 y el número de generaciones 50. El umbral de aleatoriedad en la primera generación fue de 0.8 y el tipo de cruzamiento fue "*uniformcrossover*". El porcentaje de mutación fue de 0.1 y el umbral de aleatoriedad durante las generaciones fue de 0.1. El número máximo de *individuos* en la selección natural fue de 20 y los *individuos* por torneo fueron 6.

Con respecto a las áreas, en este capítulo se muestran las áreas originales, las áreas objetivo, el peso para cada una de ellas y las áreas después de la optimización.

Áreas originales1	*Áreas objetivo*	*Peso*	*Áreas Resultado*
186.518179599148	*180*	*5*	*176.084037*
117.172135853976	*150*	*10*	*125.105584*
109.066162863907	*150*	*10*	*134.895249*
118.304924985139	*100*	*5*	*103.004348*
74.5110591545522	*80*	*5*	*77.7234603*
85.8490262622089	*80*	*10*	*71.2979526*
139.130136915322	*120*	*5*	*116.211897*
194.868365739027	*200*	*5*	*195.672396*
280.631789276881	*280*	*20*	*266.470804*
220.705641103213	*200*	*20*	*184.608121*
302.757826500145	*380*	*5*	*378.217385*

Figura 117. Tabla de áreas de la planta 1.

Las filas coloreadas en la columna "*Áreas Resultado*" corresponden a los polígonos que consiguieron estar dentro del margen establecido por el arquitecto. A medida que el valor es más pequeño más exigente será la evaluación para premiar al *individuo*.

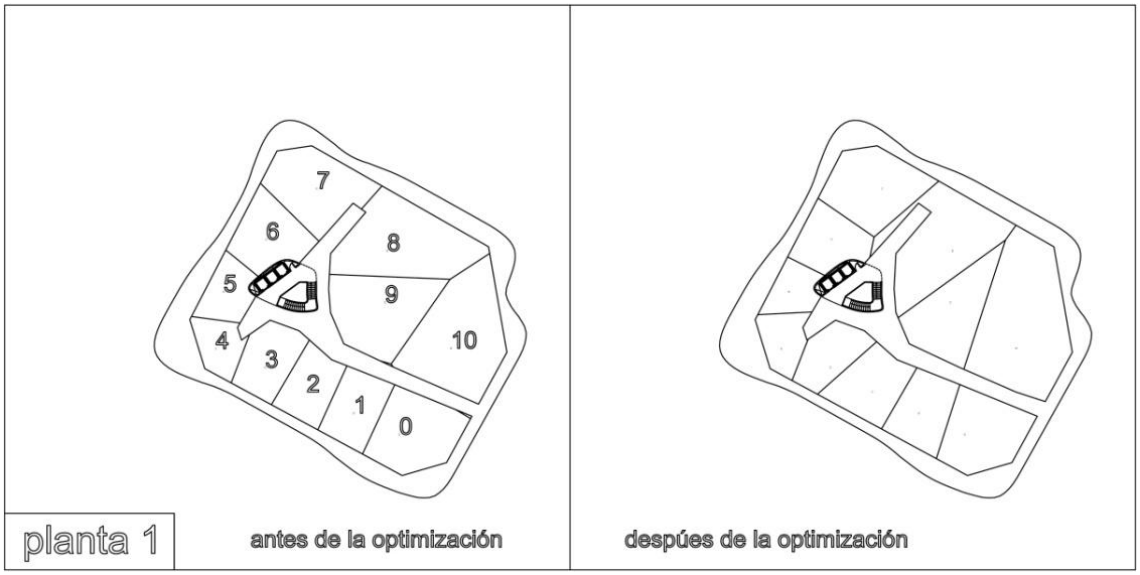


Figura 118. Dibujo de la planta 1 antes y después de la optimización

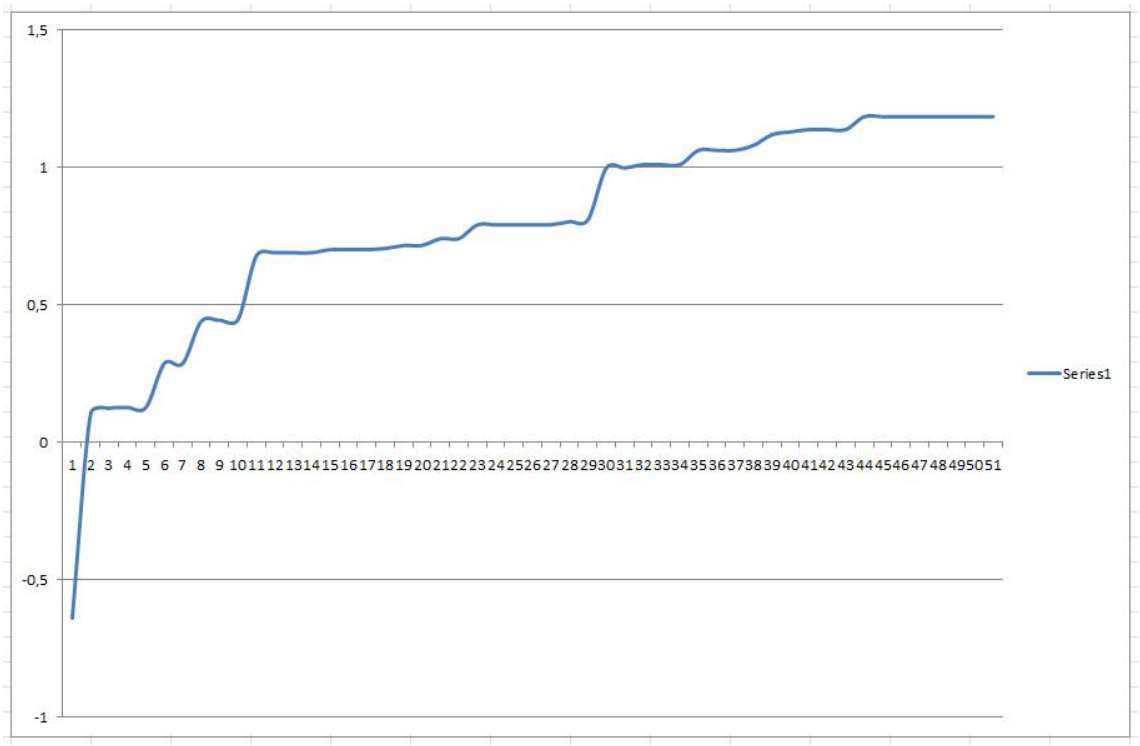


Figura 119. Curva de optimización de la Planta 1

PLANTA 2:

Misma configuración que planta 1.

El número de *individuos* fue definido en 26 y el número de generaciones 25. El umbral de aleatoriedad en la primera generación fue de 0.5 y el tipo de cruzamiento fue "*singlepointcrossover*". El porcentaje de mutación fue de 0.3 y el umbral de aleatoriedad durante las generaciones fue de 0.1. El número máximo de *individuos* en la selección natural fue de 10 y los *individuos* por torneo fueron 6.

Áreas originales2	*Áreas objetivo*	*Peso*	*Áreas Resultado*
183.512473979071	*150*	*5*	*148.564664*
123.098318033534	*100*	*5*	*127.753072*
112.020183139156	*100*	*5*	*104.619447*
114.841742671362	*100*	*5*	*120.242904*
133.819705304215	*120*	*5*	*134.127655*
154.576159183939	*150*	*5*	*151.207123*
90.6025818522835	*100*	*5*	*103.353633*
108.373676771961	*100*	*5*	*102.077233*
123.305874994272	*100*	*10*	*101.546487*
135.122626065567	*110*	*10*	*119.687827*
131.448211739495	*110*	*10*	*134.179199*
136.798213589643	*110*	*10*	*148.991023*
198.865771212992	*200*	*5*	*200.819285*
196.656819525828	*200*	*5*	*197.951886*

Figura 120: Tabla de áreas de la planta 2.

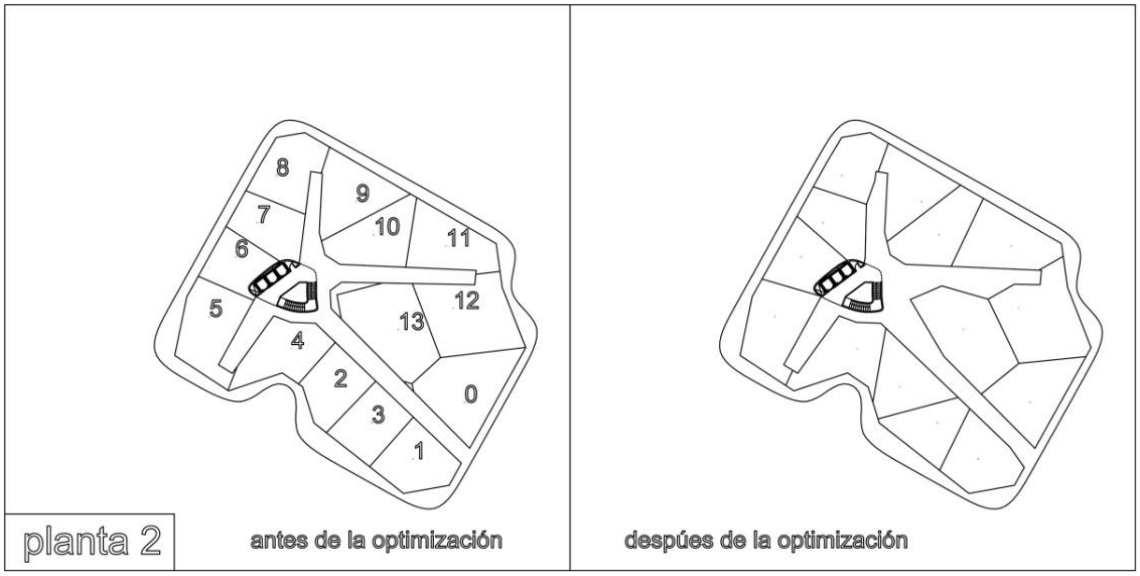


Figura 121: Dibujo de la planta 2 antes y después de la optimización.

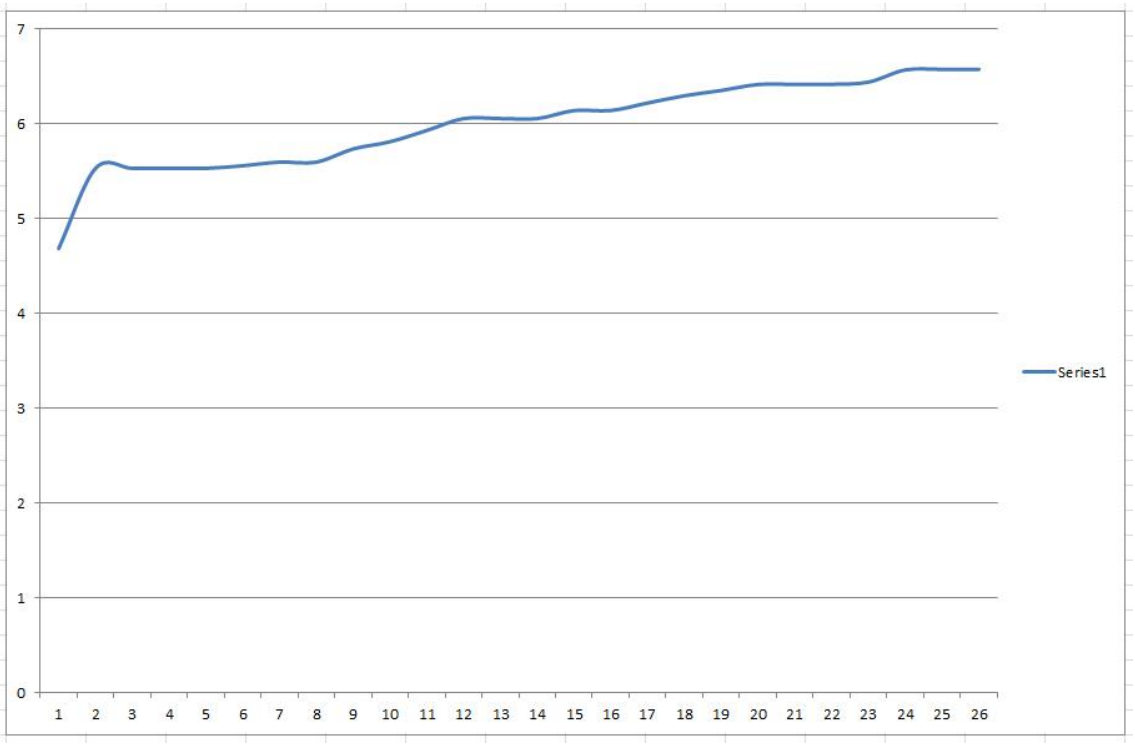


Figura 122: Curva de optimización de la planta 2

PLANTA 3:

El número de *individuos* fue definido en 20 y el número de generaciones 25. El umbral de aleatoriedad en la primera generación fue de 0.3 y el tipo de cruzamiento fue "*singlepointcrossover*". El porcentaje de mutación fue de 0.4 y el umbral de aleatoriedad durante las generaciones fue de 0.5. El número máximo de *individuos* en la selección natural fue de 20 y los *individuos* por torneo fueron 3.

Áreas originales3	*Áreas objetivo*	*Peso*	*Áreas Resultado*
209.164922107807	*150*	*5*	*173.646107*
148.203279325647	*150*	*10*	*143.16116*
119.244187701599	*100*	*10*	*150.984568*
102.432047667686	*100*	*5*	*113.208376*
53.2435004575574	*60*	*5*	*54.5268664*
56.3427555431691	*60*	*10*	*43.180584*
46.4953421561525	*60*	*10*	*40.2431776*
41.5532843911373	*60*	*5*	*46.2541351*
62.2170668099073	*60*	*10*	*65.0943269*
98.6364059470108	*100*	*10*	*92.0339729*
115.508015427566	*100*	*5*	*105.074837*
97.4911876155572	*100*	*5*	*105.103553*
110.835831770606	*100*	*5*	*138.906067*
182.94246898785	*200*	*20*	*177.395783*
182.111913066965	*150*	*20*	*177.745981*

Figura 123: Tabla de áreas de la planta 3.

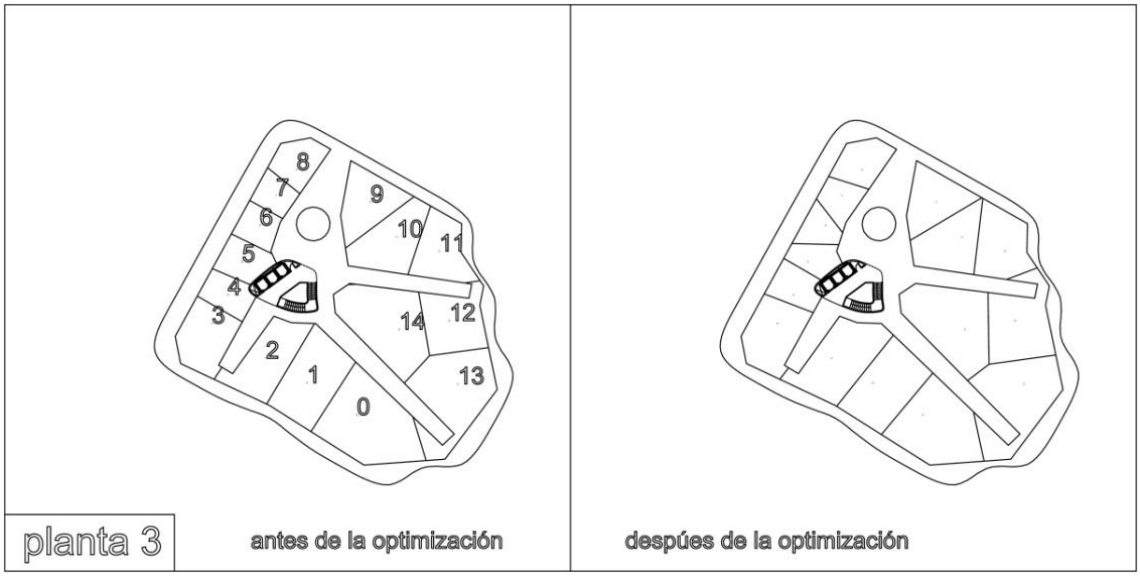


Figura 124: Dibujo de la planta 3 antes y después de la optimización.

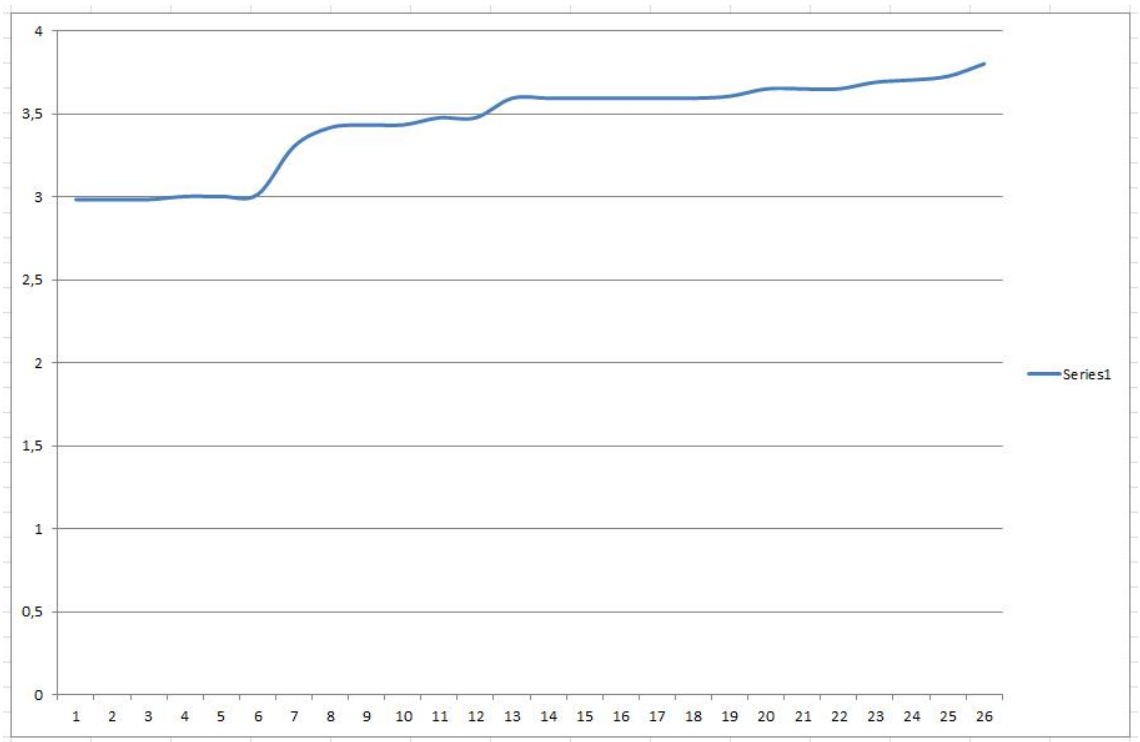


Figura 125: Curva de optimización de la planta 3.

PLANTA 4:

El número de *individuos* fue definido en 20 y el número de generaciones 30. El umbral de aleatoriedad en la primera generación fue de 0.8 y el tipo de cruzamiento fue "*uniformcrossover*". El porcentaje de mutación fue de 0.2 y el umbral de aleatoriedad durante

las generaciones fue de 0.2. El número máximo de *individuos* en la selección natural fue de 20 y los *individuos* por torneo fueron 6.

Áreas originales4	*Áreas objetivo*	*Peso*	*Áreas Resultado*
188.375634453119	*150*	*5*	*181.774228*
110.366935558166	*100*	*5*	*121.071128*
109.697574288995	*100*	*15*	*102.582393*
119.251871253481	*100*	*15*	*118.609432*
67.5504666202691	*60*	*15*	*72.9019217*
129.717316956284	*100*	*5*	*131.499794*
62.0847941454807	*60*	*5*	*62.9213733*
35.6741337613351	*30*	*5*	*29.8872669*
38.5011505755482	*50*	*10*	*45.0065787*
101.348668387296	*100*	*10*	*100.182359*
103.445058668287	*100*	*5*	*100.067754*
157.713248009625	*150*	*5*	*149.994791*

Figura 126: Tabla de áreas de la planta 4.

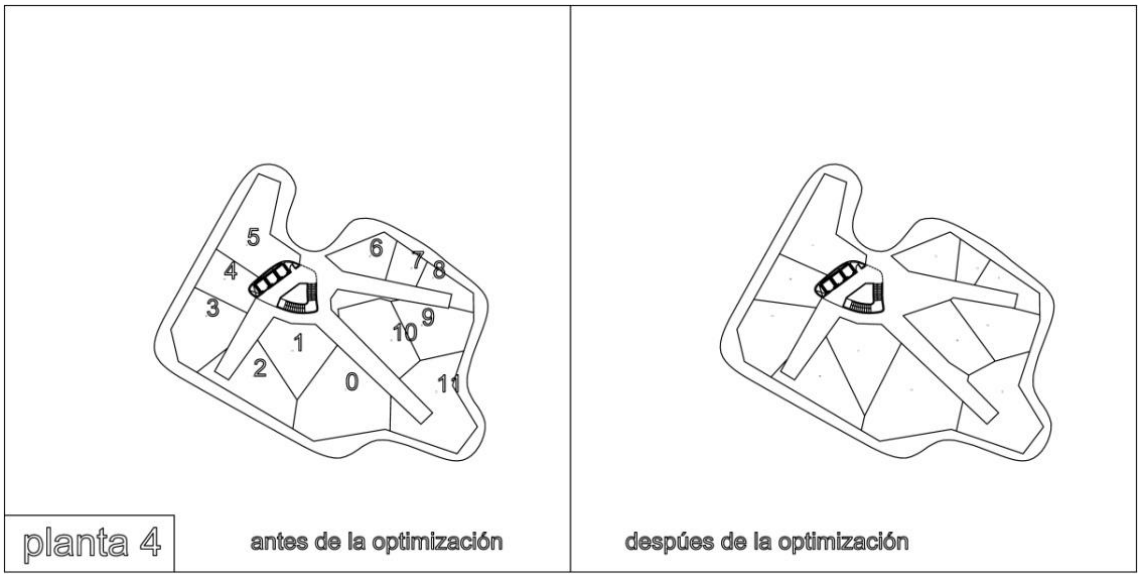


Figura 127: Dibujo de la planta 4 antes y después de la optimización.

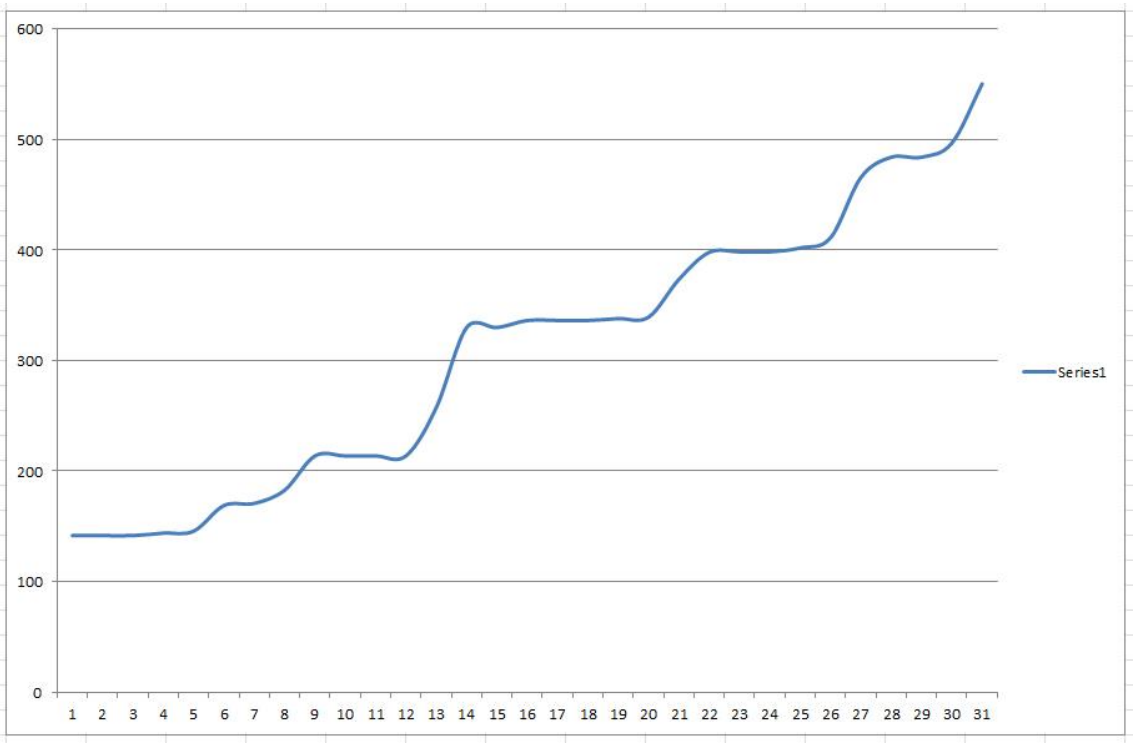


Figura 128: Curva de optimización de la planta 4.

PLANTA 5:

El número de *individuos* fue definido en 20 y el número de generaciones 40. El umbral de aleatoriedad en la primera generación fue de 0.8 y el tipo de cruzamiento fue "*uniformcrossover*". El porcentaje de mutación fue de 0.1 y el umbral de aleatoriedad durante las generaciones fue de 0.5. El número máximo de *individuos* en la selección natural fue de 20 y los *individuos* por torneo fueron 4.

Áreas originales5	*Áreas objetivo*	*Peso*	*Áreas Resultado*
31.513123582773	*30*	*5*	*29.0268269*
29.183609752979	*30*	*5*	*27.26839*
74.1016867674174	*50*	*5*	*72.8664806*
94.6855090218928	*100*	*20*	*87.5501733*
96.3710683210549	*100*	*20*	*97.5006558*
80.4170006240552	*100*	*20*	*67.2822761*
65.7618905690158	*60*	*15*	*74.4855967*
62.1529499901358	*60*	*15*	*64.8515775*
70.8269850759674	*60*	*15*	*69.0674462*
55.2394382515902	*50*	*5*	*48.2414629*
29.6405812497771	*30*	*5*	*28.9227354*

23.4000259010304	*30*	*5*	*29.9734891*
25.9048714337049	*30*	*5*	*30.0070482*

Figura 129: Tabla de áreas de la planta 5.

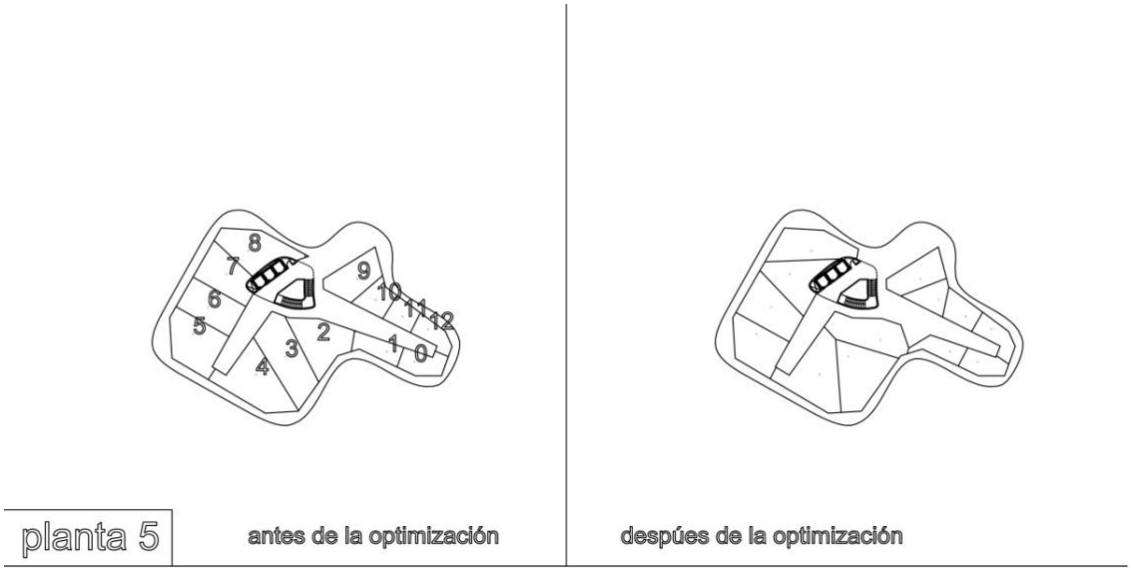


Figura 130: Dibujo de la planta 5 antes y después de la optimización.

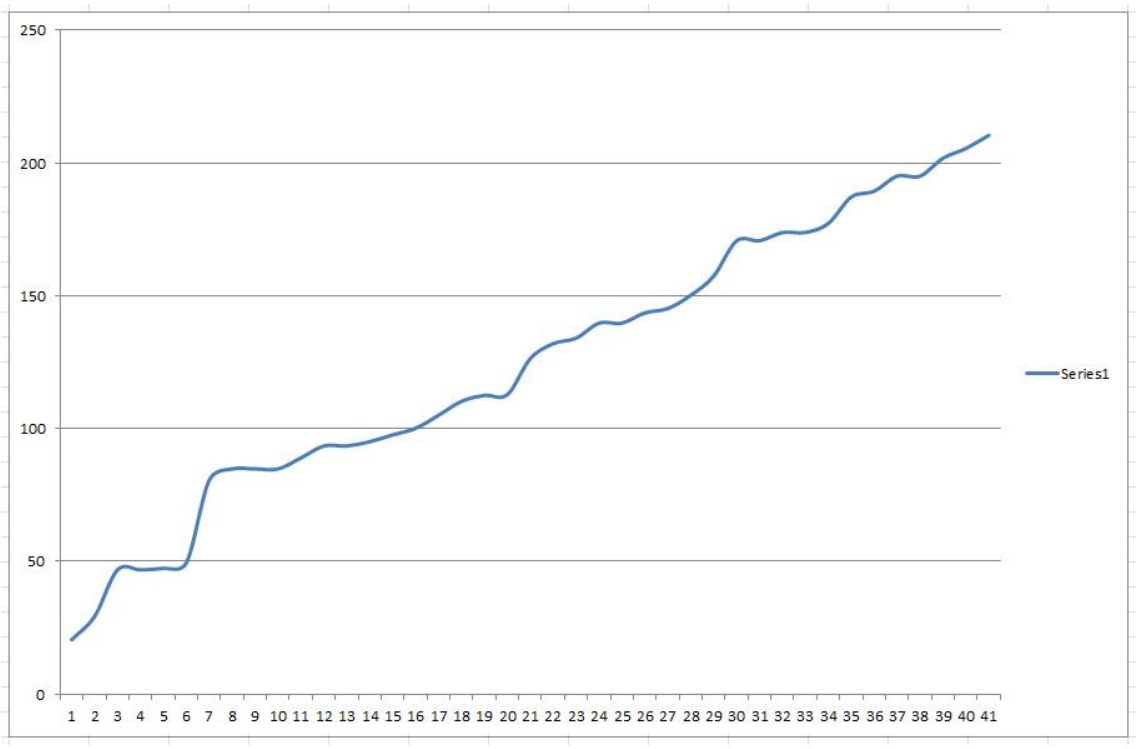


Figura 131: Curva de optimización de la planta 5.

PLANTA 6:

El número de *individuos* fue definido en 10 y el número de generaciones 50. El umbral de aleatoriedad en la primera generación fue de 1.0 y el tipo de cruzamiento fue "*uniformcrossover*". El porcentaje de mutación fue de 0.1 y el umbral de aleatoriedad durante las generaciones fue de 0.1. El número máximo de *individuos* en la selección natural fue de 20 y los *individuos* por torneo fueron 8.

Áreas originales6	*Áreas objetivo*	*Peso*	*Áreas Resultado*
69.5504879601908	*60*	*5*	*64.9290037*
65.9925212500317	*70*	*5*	*74.2208814*
75.3454718717885	*70*	*5*	*103.034227*
99.8167403841511	*100*	*5*	*100.633961*
115.36090268617	*100*	*5*	*100.210549*
142.125280724255	*150*	*5*	*150.042257*
174.652886649824	*150*	*5*	*150.050574*

Figura 132: Tabla de áreas de la planta 6.

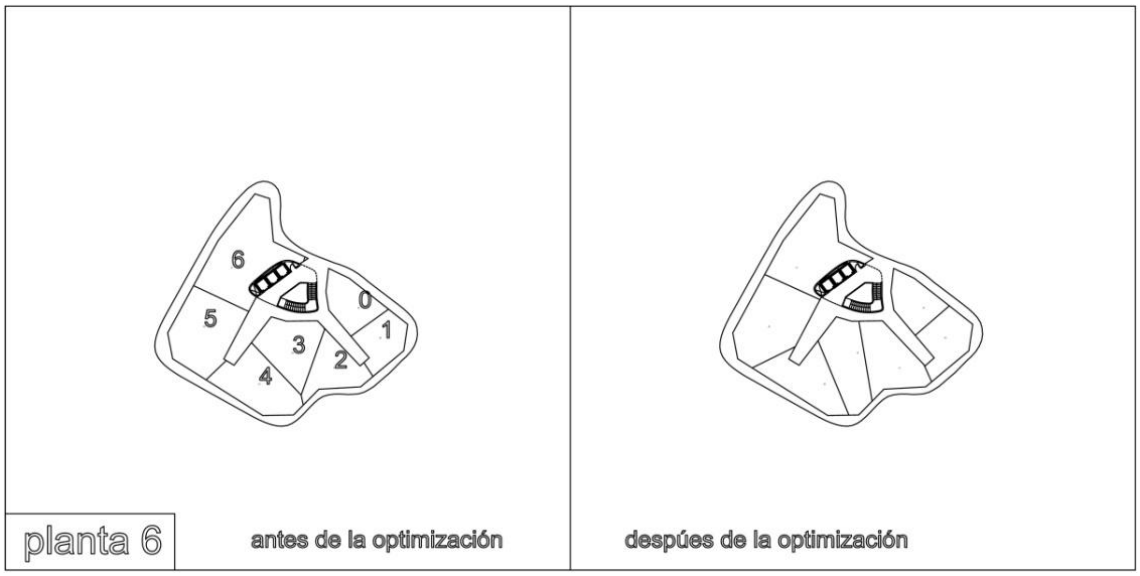


Figura 133: Dibujo de la planta 6 antes y después de la optimización.

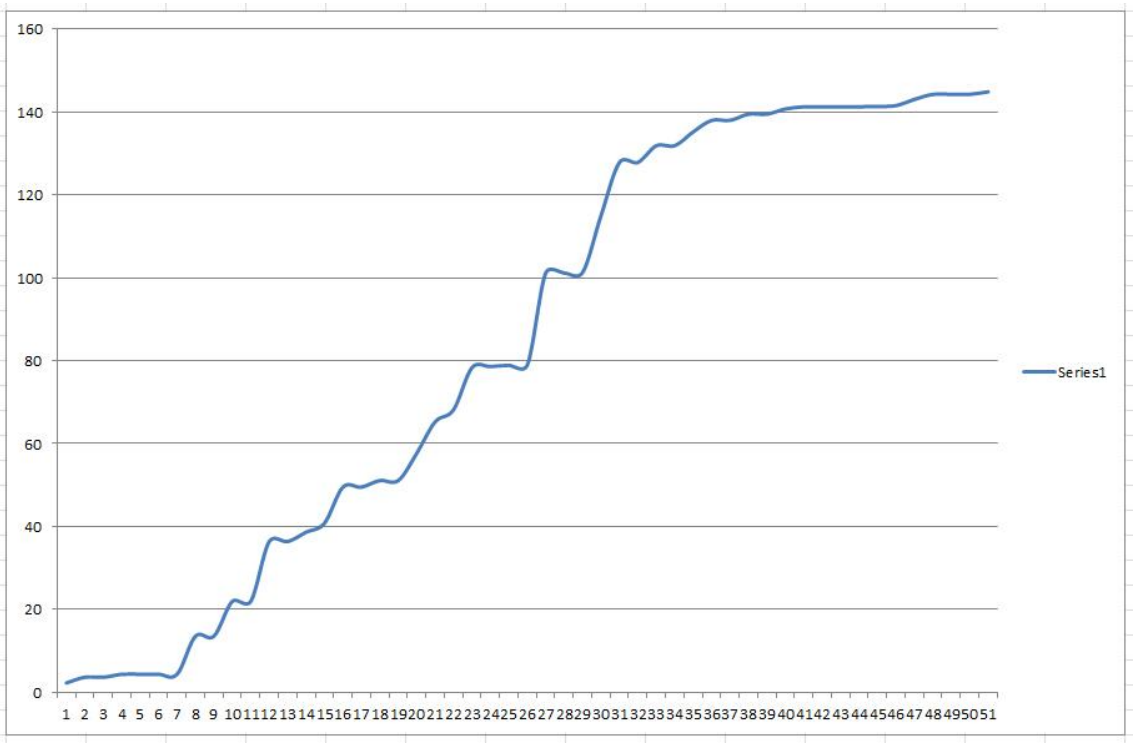


Figura 134: Curva de optimización de la planta 6.

PLANTA 7:

El número de *individuos* fue definido en 10 y el número de generaciones 50. El umbral de aleatoriedad en la primera generación fue de 1.0 y el tipo de cruzamiento fue "*uniformcrossover*". El porcentaje de mutación fue de 0.1 y el umbral de aleatoriedad durante las generaciones fue de 0.1. El número máximo de *individuos* en la selección natural fue de 20 y los *individuos* por torneo fueron 8.

Áreas originales7	*Áreas objetivo*	*Peso*	*Áreas Resultado*
90.0625349148622	*100*	*5*	*97.7072136*
124.698092433477	*100*	*5*	*104.464163*
87.8732963662707	*100*	*5*	*99.9517759*
69.7694355656641	*60*	*5*	*60.0432356*
54.7792222778696	*60*	*5*	*60.0128139*
85.0094895003774	*90*	*5*	*90.0041616*

Figura 135: Tabla de áreas de la planta 7.

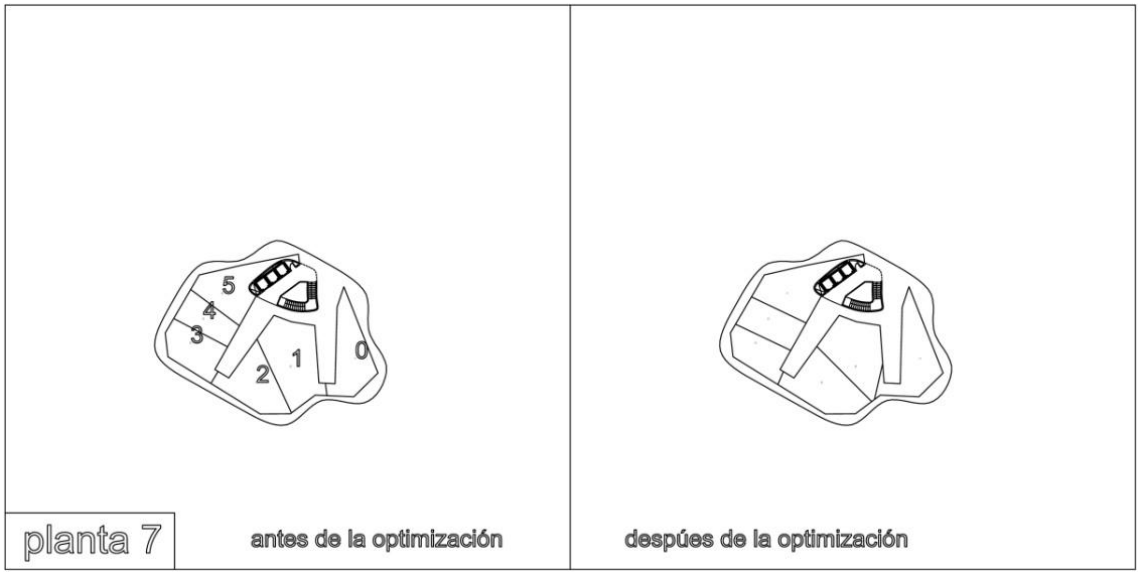


Figura 136: Dibujo de la planta 7 antes y después de la optimización.

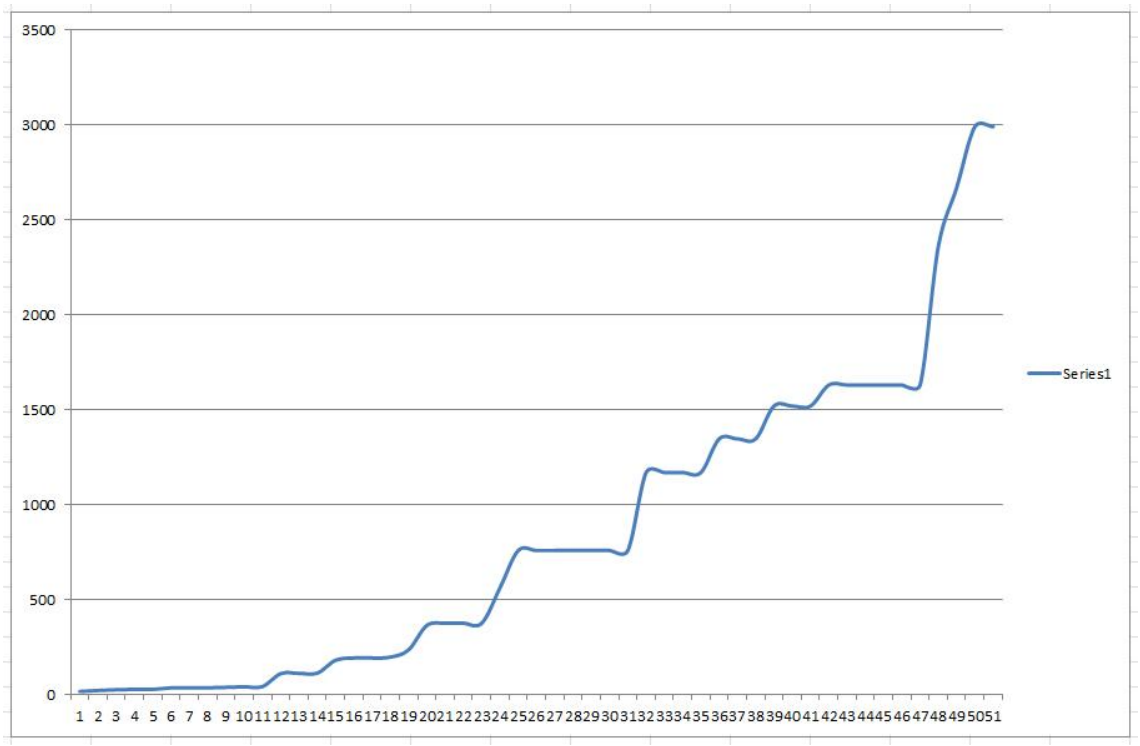


Figura 137: Curva de optimización de la planta 7.

PLANTA 8:

El número de *individuos* fue definido en 10 y el número de generaciones 50. El umbral de aleatoriedad en la primera generación fue de 1.0 y el tipo de cruzamiento fue "*uniformcrossover*". El porcentaje de mutación fue de 0.1 y el umbral de aleatoriedad durante las generaciones fue de 0.1. El número máximo de *individuos* en la selección natural fue de 20 y los *individuos* por torneo fueron 12.

Áreas originales8	*Áreas objetivo*	*Peso*	*Áreas Resultado*
80.6081426165335	*70*	*5*	*67.4078361*
65.0400087469539	*70*	*5*	*70.6520283*
42.5195529822507	*50*	*5*	*52.0728579*
37.2285597251573	*50*	*5*	*46.1655854*
48.3232468821862	*50*	*10*	*57.1976733*
50.130750746235	*40*	*5*	*44.316803*
68.5351053629645	*50*	*5*	*54.572583*

Figura 138: Tabla de áreas de la planta 8.

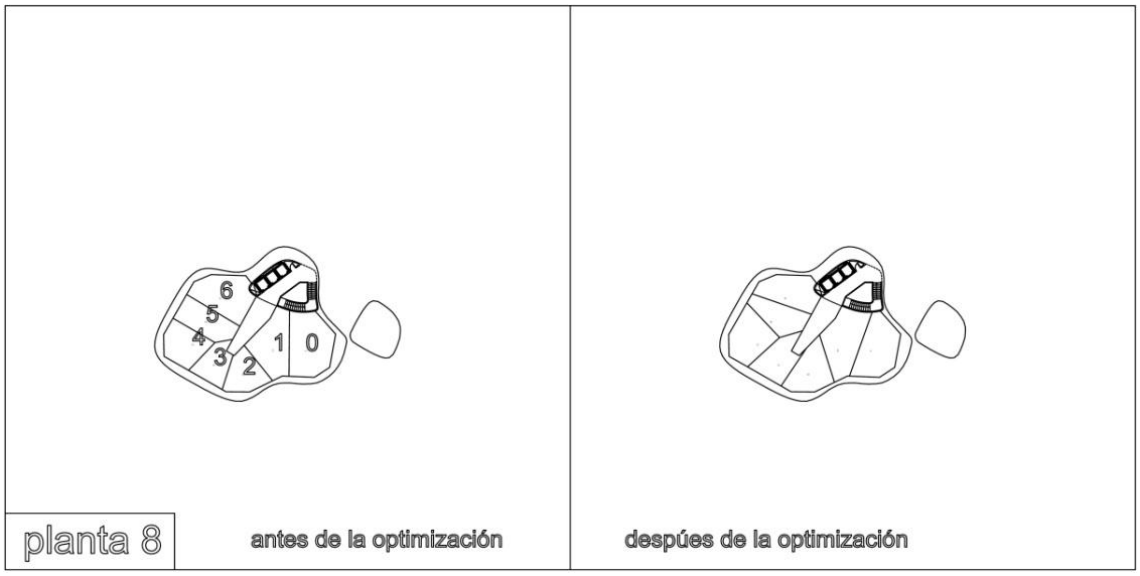


Figura 139: Dibujo de la planta 8 antes y después de la optimización.

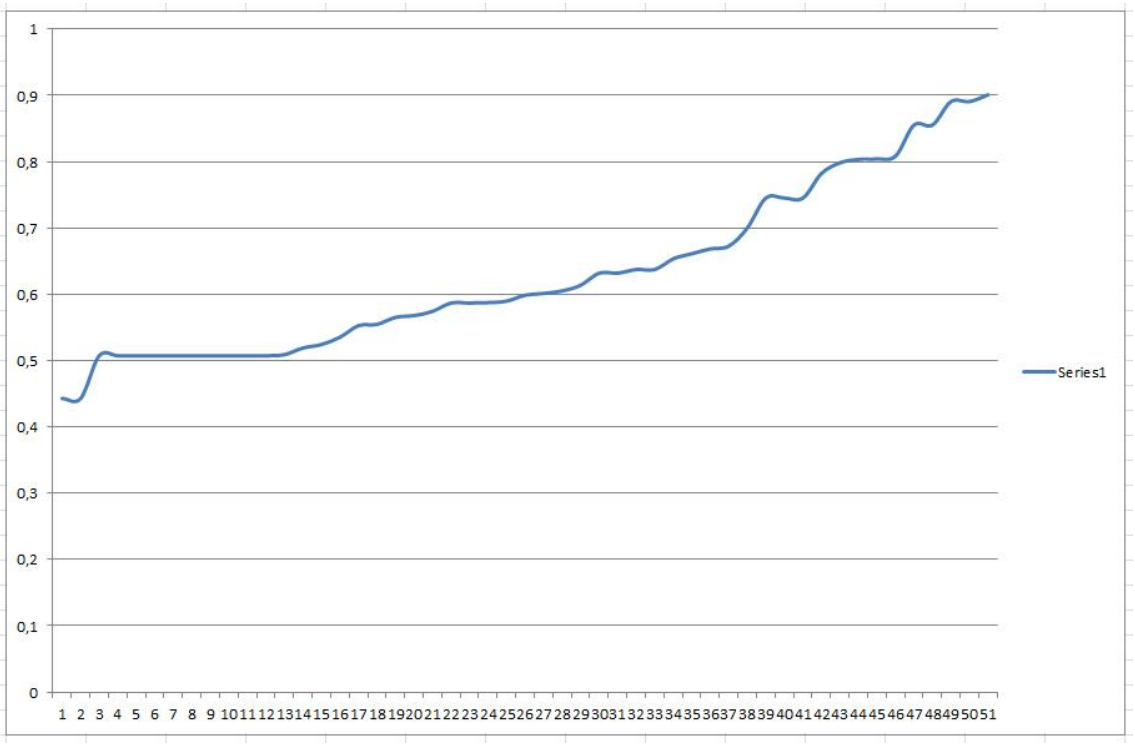


Figura 140: Curva de optimización de la planta 8.

8.3.4. Discusión

Este tipo de problema es tan complejo que sin los diferentes resultados (tabla de áreas, planta y curva de optimización) es muy difícil poder entender todo el conjunto. En las tablas de áreas las filas coloreadas corresponden a los polígonos, del mejor *individuo*, que estuvieron por debajo del umbral establecido por el arquitecto. Las otras filas no coloreadas corresponden a las áreas donde no hubo optimización o el área del polígono no se acercó por debajo del umbral establecido.

Esto puede deberse básicamente a 3 factores, el primero de ellos con respecto a la configuración de las plantas. Si bien las plantas tienden a ser cuadradas, su distribución interna es una sucesión de espacios todos contiguos que posiblemente restringió el movimiento del punto en la manera en que fue planteado el AG. Si este es el problema habría que plantear una nueva estrategia para desplazar los puntos. El segundo motivo, puede deberse a la cantidad de iteraciones a los que el AG fue sometido. Con 3 y 4 polígonos fuera de umbral están las plantas 2 y 3 con 25 iteraciones cada una. Con 2 polígonos fuera de umbral, están las plantas 1, 4, 5, que tienen 50, 30 y 40 iteraciones respectivamente. Y con uno o menos polígonos están las plantas 6, 7 y 8 con 50 iteraciones cada una. Si éste problema pudiese ser asociado a la cantidad de iteraciones la respuesta estaría en las curvas de optimización, que si bien presentan una morfología escalonada en todas las pruebas, también en todas mostraban optimización al momento en que el AG fue cancelado. El tercer factor que puede influir en los polígonos fuera de umbral corresponde a la suma de las áreas objetivo versus la suma del área real capaz de soportar el polígono. Por ejemplo, si tenemos un recinto de 200m^2 dividido en dos espacios cada uno de 100m^2, el AG por mucho que optimice jamás podría encajar 150m^2 para cada área. Probablemente lo que intento hacer es encajar un recinto en 150m^2 y el otro

dejarlo en 50m^2. Esto en la tabla de áreas aparecería reflejado como la fila de 150m^2 coloreada y la de 50m^2 en blanco muy por debajo de su objetivo.

El área objetivo de la planta 1 suma 1920m^2 y la real 1830m^2. Esto hace una diferencia de 90m^2. En la planta 2 sucedía todo lo contrario el área objetivo es de 1750m^2 y el área real de 1958m^2. Entonces es lógico entender que los tres recintos que no estuvieron dentro del umbral tuviesen áreas más grandes en casi un 30%. Todo eso para llenar el polígono del edificio. En la planta 3 el objetivo fue de 1250m^2 pero la real era de 1626m^2, los 4 polígonos que no cumplieron con sus áreas, dos de ellos disminuyeron su área en un total de 45m^2 y los otros la aumentaron en 80m^2, para compensar el desequilibrio en la división interior. En la planta 4 el objetivo era de 1100m^2 y el real de 1243m^2. En este caso los dos polígonos que estuvieron fuera del umbral incrementaron 50m^2 entre ambos. Lo mismo sucede con las otras 4 pruebas. El AG optimizó de manera ignorante, en el sentido de que no sabía lo que mejoraba, solo mejoraba y para esto sacrificó ciertos polígonos del conjunto. Lo interesante sería ver resultados con 200 o 300 iteraciones en vez de las pruebas que se están haciendo con 50. Para esto, será necesario traducir el algoritmo que construye el *Voronoi* a un lenguaje de más bajo nivel para ganar velocidad y reducir el costo computacional. Por el momento el diagrama es generado por medio de un algoritmo y un comando de *Rhino* que obliga por cada iteración generar todas las mediatrices. Dentro del AG hay un segundo comando que libera la memoria capturada por *Rhino*, pero al acabo de 50 o más iteraciones este comando no es capaz de liberarla completamente haciendo que *Rhino* colapse. Esta es la razón por la cual el AG se ha ejecutado como máximo 50 veces.

8.4. Estrategia Evolutiva para el Diseño de Arquitectura Optimizada

8.4.1. Introducción

La propuesta del presente trabajo consiste en optimizar un edificio modificando sus aperturas (ventanas) y su geometría para reducir los consumos de calefacción y aire acondicionado. La optimización es realizada usando un *Micro*-Algoritmo Genetico (*Micro*-AG) programado en C# el cual es incrustado como una serie de funciones dentro de *GenerativeComponents* (GC). El software *EnergyPlus* (E+) es usado para evaluar los niveles de consumos de HVAC (*Heating, Ventilation, Air Conditioning*) dentro del edificio. El objetivo de la optimización es mantener la temperatura a 20ºC en el día más caluroso y más frío del año usando la menor energía posible (jules).

Parte de la energía que recibe la tierra es reflejada hacia el espacio, y otra cantidad es atrapada por las nubes y gases atmosféricos como el dióxido de carbón (CO2), metano (CH4) y óxidos de nitrógeno (N2O). Estos gases son llamados gases invernaderos[97]. Este fenómeno es natural y hace posible la vida en el planeta. El problema aparece cuando los seres humanos

97 Más detalles sobre gases invernaderos en: http:// //unfccc.int/2860.php [13-05-2010]

incrementan la emisión de estos gases a la atmosfera incrementando el efecto invernadero y consecuentemente el calentamiento global. Este trabajo pretende ser una pequeña contribución hacia el camino de una arquitectura sustentable.

La naturaleza como la conocemos hoy en día es el resultado de un largo proceso biológico llamado evolución y que comenzó cuando apareció la vida en el planeta y que aún se está escribiendo. Mamíferos, reptiles, peces, vegetales, aves y hongos han tenido que resolver una serie de problemas como soportar altas presiones para cazar y poder comer, volar, correr, nadar etc. Sin embargo la perfección y la variedad de las formas naturales son el resultado de la misma estrategia repetida una y otra vez. En otras palabras, una incansable producción de prototipos y un implacable rechazo de los modelos menos aptos.

De esta manera la naturaleza ha evolucionado en una rica biodiversidad de especies, todas ellas dependiendo entre si y el medio ambiente que les rodea. Cada forma de la naturaleza es la expresión de una información codificada (ADN). Sin embargo la forma final depende del balance entre el código del organismo y el medio ambiente donde vive la especie.

Hoy en día, uno de los problemas principales en arquitectura es el consumo de HVAC (*Heating, Ventilation, Air Conditioning*) en los edificios. El problema para resolver en este trabajo consiste en reducir la calefacción y el aire acondicionado, modificando la geometría, la posición y el tamaño de las ventanas. Este problema se considera multi-objetivo porque ambos parámetros están relacionados y no podemos minimizar uno de ellos sin incrementar el otro.

Si tenemos una gran ventana en verano el sol calentara todos los muebles y objetos, lo que incrementará la temperatura al interior de la habitación y tendremos que encender el aire acondicionado para mantener la temperatura a 20ºC. La buena noticia es que podemos tomar ventaja de la luz natural más tiempo y además podremos refrescar la casa en la noche liberando el calor atrapado en las murallas por la radiación durante el día. En invierno la misma ventana capturara el calor que llega del sol, pero desafortunadamente todo el calor se perderá rápidamente porque escapará a través de la ventana por donde mismo entró. Entonces tendremos que encender la calefacción rápidamente y lo peor de todo, es que el calor que produzcamos continuara escapando a través de la ventana incrementando el consumo de calefacción.

Si reducimos el tamaño de la ventana al mínimo posible, lo que sucederá durante el verano es que no seremos capaces de liberar el calor atrapado en las murallas. Por esto, tendremos que permanecer con el aire acondicionado por más tiempo y lógicamente no podremos aprovechar la luz natural como cuando teníamos la ventana grande. Además como tenemos la ventana pequeña, tendremos que encender la luz más temprano y deberemos liberar este calor también. En invierno, la misma ventana no permitirá calentar el interior de la habitación y tendremos que encender la calefacción todo el día y probablemente toda la noche. Además, tendremos que encender la luz artificial porque la luz solar no será suficiente ya que el invierno es más obscuro que el verano. La parte buena es que la mayor parte de calor que produzcamos permanecerá dentro de la habitación.

En resumen, una ventana grande actúa bien en verano y tiene un muy mal comportamiento en invierno. Por el contrario, una pequeña ventana es una molestia en verano, pero tiene un mejor comportamiento en invierno. Por lo tanto, el problema no es ampliar o reducir el tamaño de la ventana para lograr un solo parámetro, sino encontrar un óptimo que equilibre ambos comportamientos.

Existe una extra ayuda a este problema que es la posibilidad de modificar la geometría del edificio, el cual permitirá explorar, buscar soluciones creativas y obtener mejores

comportamientos que estarían fuera de nuestro alcance si solo modificáramos la posición y tamaño de ventanas.

Este trabajo pretende ser una semilla en la contribución en la búsqueda del desarrollo sustentable de edificios para reducir los consumos de calefacción y aire acondicionado, buscando la mejor solución por medio de algoritmos genéticos. La solución al problema consiste en responder la siguiente pregunta: ¿Cuál es la mejor geometría y configuración de ventanas para mantener la temperatura a 20ºC durante el día más frio y más caliente del año que consuma la menor energía?

Este trabajo está basado en el documento "*Architectural Constraints in a Generative Design System: Interpreting Energy Consumption Levels*" por Luisa Caldas and Leslie Norford en la séptima conferencia IBSA celebrada en Rio de Janeiro el 2001.[98]

En el documento, las investigadoras codifican uno de los edificios de la escuela de arquitectura de Oporto diseñada por alvaro Siza, para aplicar un AG que optimice la luz natural y el consumo de energía en calefacción y refrigeración dentro del edificio. Ellas usaron este edificio como banco de pruebas y sus ventanas como marco de trabajo para modificar y explorar diferentes configuraciones durante las iteraciones del algoritmo. Para evaluar los consumos de HVAC usaron DOE-2.1E el cual es un programa de simulación térmica y un *Micro*-Algoritmo Genético basado en el software *GenOpt*, el cual no es descrito en profundidad.

El trabajo en este capítulo está basado en un modelo paramétrico para restringir ciertas relaciones del edificio, para evitar fallas en el reporte durante el proceso de evaluación y codificar de una manera simple los parámetros del modelo. Un *Micro*-AG es usado para buscar y optimizar la geometría, aunque presenta algunas variaciones en comparación con un AG tradicional. Un aspecto a comentar es que las poblaciones son bastante pequeñas, 5 *individuos* por generación y no más de 5 iteraciones son usadas para obtener resultados. A diferencia de un AG tradicional que maneja *población* de alrededor de 20-30 *individuos* y 200-300 generaciones.

Este trabajo ha sido programado básicamente en C# en dos diferentes niveles. El nivel superior corresponde a *GCScript* el cual es aplicado para controlar la geometría y administrar las diferentes clases que componen el AG y los archivos de transacción entre GC y E+. El nivel bajo es puro C# escrito directamente sobre Visual Studio 2008 el cual es incrustado dentro de GC a través de una serie de funciones. El lenguaje de programación C# es usado para realizar las tareas duras y así ganar velocidad durante el proceso. Todos los operadores del AG han sido desarrollados en C# así como también las aplicaciones que traducen los archivos de un programa a otro. Finalmente, la evaluación del edificio es realizada en *EnergyPlus* el cual es la evolución del software usado por Caldas y Norford.

98 http://www.inive.org/members_area/medias/pdf/Inive%5CIBPSA%5CUFSC558.pdf, [14-05-2010], Capítulo 6, subcapítulo 6.1.3.

8.4.2. Método

Esta parte consiste en la explicación de cómo funciona todo el proceso, la cual está dividida en la descripción del modelo paramétrico, la explicación del modelo termal dentro de E+ y la descripción de las clases y como se relacionan entre ellas para crear el AG.

MODELO PARAMÉTRICO

El modelo es una caja perpendicular al Norte, compuesta por 6 superficies planas y dos ventanas. Una orientada al Este y la otra al Oeste. Las medidas de la caja son 12 metros en la dirección X (Este – Oeste), 10 metros en la dirección Y (Norte - Sur) y 4 metros de alto.

El primer paso en la construcción consiste en definir 4 parámetros llamados, *coordXSrfWest*, *coordYSrfWest*, *coordXSrfEast* y *coordYSrfEast*. Estos parámetros son usados para controlar la geometría de la caja. El segundo paso es definir 3 puntos, el primero de ellos vive en el centro de la caja (*point01*) y los otros dos son llamados *ptWest* y *ptEast* los cuales son controlados por los parámetros previamente definidos. La coordenada en Z de los tres puntos es fija, para asegurar que las superficies sean planas, de otra manera *EnergyPlus* no realizara un análisis correcto.

Dos vectores (*ByOriginDirectionPoint()*), son definidos a partir del punto *point01* en dirección a los puntos *ptWest* y *ptEast*. Estos vectores son usados para recibir dos planos (*ByDirectionAndDistanceFromOrigin()*), uno para el lado este y otro para el lado oeste.

Estos planos además reciben 2 sistemas de coordenadas en sus bases (*OnPlane()*) y 2 polígonos son creados a partir de la intersección de los planos mediante *GCScript*. Las ventanas también son generadas mediante *GCScript* y su tamaño depende de las dimensiones de la superficie que la contiene. El resto de las 4 caras de la caja son definidas por los vértices de los primeros dos polígonos.

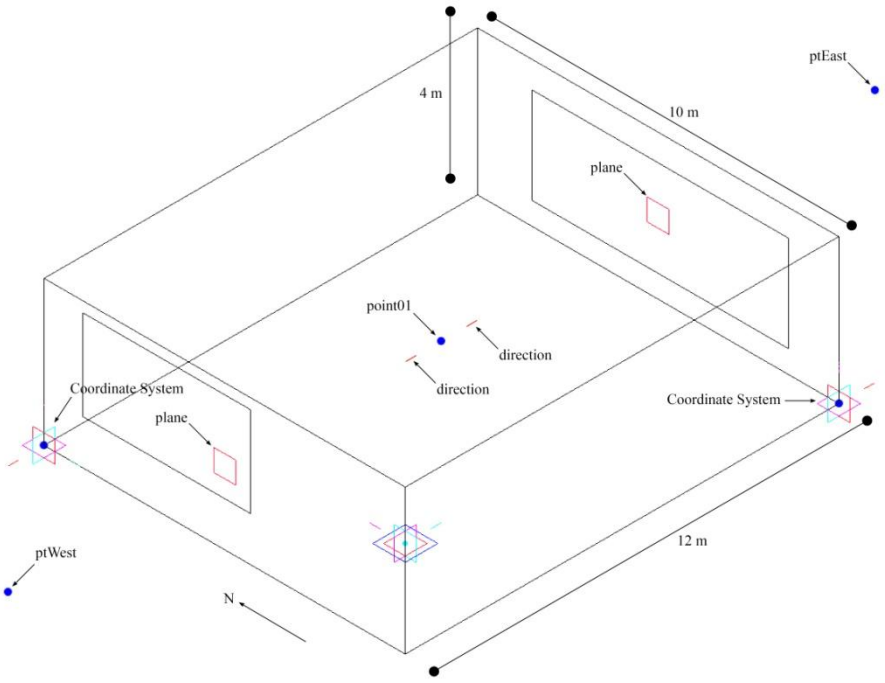


Figura 141: Elementos geométricos para construir el edificio.

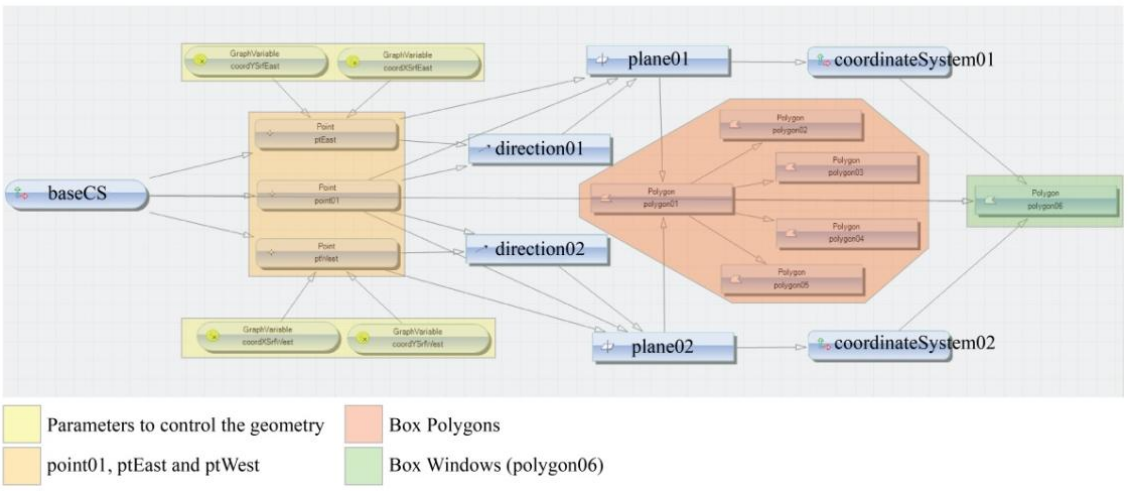


Figura 141: El Diagrama Simbólico de *GenerativeComponents*, el cual explica las relaciones entre los componentes geométricos y aritméticos de la forma. La dirección de las flechas índica la dependencia de la geometría. Por ejemplo, el sistema de coordenadas *CoordinateSystem01* depende del plano *plane01* el cual depende a su vez del vector *direction01* y de los puntos *ptEast* y *point01,* que a su vez dependen del sistema de coordenadas *baseCS*.

Parametrizar un modelo requiere un doble esfuerzo. Por un lado es necesario saber exactamente cuáles son los mínimos elementos necesarios para construir la geometría y así evitar perder velocidad. Esto significa que mientras más pasos de relaciones y dependencias se den para obtener el modelo final más cálculo es necesario y por lo tanto más costo computacional. Por otro lado es necesario determinar cuál de estos parámetros es una restricción y cuál de estos permiten flexibilidad. Una vez que el modelo es parametrizado debe ser testeado por al menos 200 iteraciones, para observar el comportamiento y detectar posibles errores en la geometría.

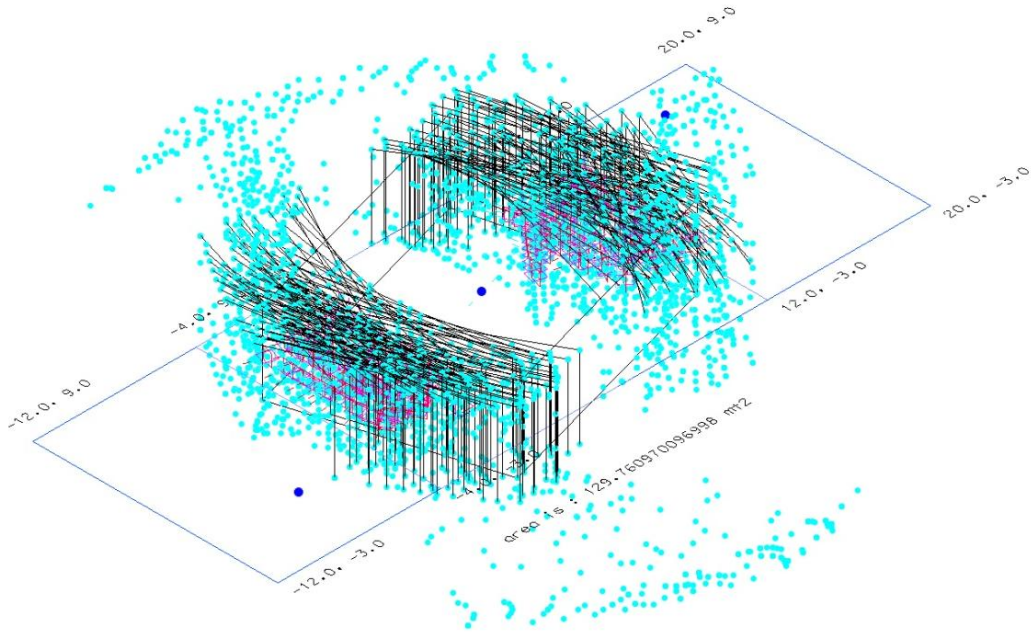


Figura 142. Secuencia del modelo, probado con 200 iteraciones completamente aleatorias.

MODELO TERMAL

El modelo termal corresponde a la otra información necesaria para describir el edificio. Estos datos son necesarios porque definen las características físicas del modelo. Como por ejemplo donde está ubicado, cuál es su tipo de clima, qué tipo de material tienen los muros, si los cristales de las ventanas son dobles o simples, etc. Toda esta información es guardada dentro del archivo de *EnergyPlus*.

La parte importante de este modelo es que la geometría (comportamiento) es definida y manipulada dentro de GC y las características físicas (propiedades) son definidas dentro de *EnergyPlus*. La correspondencia entre los dos archivos siempre debe ser la misma. Por ejemplo, si en el archivo de E+ aparecen descritas 6 superficies planas, el modelo debe proveer 6 superficies planas.

LAS PRINCIPALES CARACTERÍSTICAS DEL MODELO DENTRO DE E+

Datos de las propiedades del edificio definidos dentro del archivo de *EnergyPlus*.

```
The Building is located: In a country terrain, Chicago IL. Latitude = 41.78,
Longitude = -87.75, Time Zone = -6.00 and elevation = 190m.
Windows: Are a double Panel window with an interior camera of 3 mm.
Walls are composed by: Wood siding outside, fiberglass quilt in the middle,
and plasterboard in the interior.
Properties of the roof are: Roof deck outside, fiberglass quilt, in the middle
and plasterboard in the interior.
The floor is defined by:
Thickness (m) = 0.10
Conductivity (W/m-K) = 1.7296
Density (kg/m3) = 2243.0
Thermal Absorptance = 0.9
Solar Absorptance = 0.65
Visible Absorptance = 0.65
The HVAC system used is a standard configuration and is defined by:
Heating Supply Air Temperature (C) = 50
Cooling Supply Air Temperature (C) = 13
Heating Supply Air Humidity Ratio (kg-H2O/kg-air) = 0.015
Cooling Supply Air Humidity Ratio (kg-H2O/kg-air) = 0.01
Heating Limit = NoLimit
Cooling Limit = NoLimit
Outdoor Air = NoOutdoorAir
```

El reporte de la simulación es definido dentro de *EnergyPlus*, En este caso, corresponde a los consumos de calefacción y refrigeración durante el día más caluroso y más frio del año. Los datos que retorna el programa están en un archivo tipo CSV[99]. El consumo es calculado cada hora a lo largo de los 2 días. Los resultados son expresados en Julios.

99 Los ficheros CSV (del inglés comma-separated values) son un tipo de documento en formato abierto sencillo para representar datos en forma de tabla, en las que las columnas se separan por comas y las filas por saltos de línea. Los campos que contengan una coma, un salto de línea o una comilla doble deben ser encerrados entre comillas dobles. http://es.wikipedia.org/wiki/CSV. [30/08/10].

CLASES

Dos tipos de clases fueron escritas en C# para este trabajo. Un tipo de clases es usada para tareas técnicas, como leer y escribir archivos, calcular distancia entre punto y línea, exportar a Excel, guardar archivos, ejecutar programas, dormir un programa, etc. El otro tipo de clases construye el AG, en estas podemos encontrar los tradicionales operadores genéticos como mutación, cruzamiento, función de *fitness*, selección y remplazo.

CLASES TÉCNICAS

Las principales clases técnicas dentro del sistema son: *GCScriptFuncVerticesToIdf.cs, GCScriptFuncSleep.cs* y *GCScriptFuncReadSolution.cs*. La clase *GCScriptFuncVerticesToIdf.cs* transforma los vértices de los polígonos en una lista de *strings*. Extrae la información del archivo de E+ y remplaza los viejos vértices por los nuevos y vuelve a guardar el archivo en el mismo lugar con el mismo nombre. Luego de esto otra clase es llamada *System.Diagnostic.Process*, la cual sirve para ejecutar E+ y realizar el análisis termal de la nueva geometría cargada. Esta clase es incrustada dentro de GC como una función llamada *runSimulation()*.

La clase *GCScriptFuncSleep.cs* duerme la aplicación por unos segundos mientras el análisis es realizado y los nuevos archivos son creados. Esta clase es esencial porque es necesario actualizar el reporte de los niveles de consumo de la nueva geometría creada. Esta clase es incrustada dentro de GC como una función llamada *sleepFunction()*.

La clase *GCScriptFuncReadSolution.cs* extrae el reporte del análisis de E+ y calcula el promedio de los consumos durante las horas del día. Esta clase manipula tres datos esenciales: El promedio de consumo de calefacción, el promedio de consumo de refrigeración y el promedio de temperatura al interior del edificio. Estos valores son retornados dentro de una lista. Esta clase es incrustada dentro de GC como una función llamada *readDataSolution()*. Sus resultados son muy importantes porque estos valores son el *fitness* de cada *individuo* dentro del AG. El objetivo es reducir estos valores lo más posible. Una vez que los datos son retornados hacia GC se calcula el promedio de los consumos para obtener un valor único.

CLASE DE OPERADORES GENÉTICOS

Una vez que la primera serie de clases es realizada, es el momento de saltar sobre los operadores genéticos. El primer operador llamado en el código es *CSChromosome.cs*. Esta clase toma el punto *ptEast*, *ptWest*, los polígonos de las ventanas y su consumo de HVAC y descompone los datos en una lista de 31 elementos. Al final de la clase el cromosoma presenta una estructura como la siguiente.

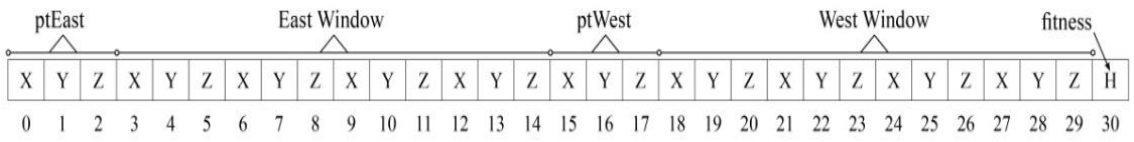


Figura 143. Estructura del cromosoma

El cromosoma es guardado en la *población* y el *fitness* es impreso en la consola del *script*. Esta clase esta incrustada dentro de GC como una función llamada *genrChromosome()*. El cromosoma está constituido por una lista de dobles. Desde el 0 al 14, los datos de la lista pertenecen al lado este y del 15 al 29 los del lado oeste. El último valor (30) es el *fitness*, el cual representa los consumos de HVAC.

Luego de que la *población* es creada, el próximo operador aplicado es *CSnaturalSelection.cs.* Este operador está encargado de ordenar los diferentes cromosomas dentro de la *población*, del más apto al menos apto. Debido a que se trabaja con una pequeña *población* de 5 candidatos por generación, el método de ruleta u otros tipos de selección natural no son aplicables a este tipo de problema. Por esto la selección se hecha por elitismo, la clase retorna la misma *población* ordenada ascendentemente (desde el menor consumo de HVAC al mayor consumo de HVAC). Esta clase es incrustada dentro de GC como una función llamada *pfrmElitism()*.

El siguiente hecho dentro del AG es el cruzamiento, en el cual las dos mejores soluciones intercambian parte de su cromosoma para constituir una nueva posible solución la cual heredera las propiedades de los padres. Debido a que se está trabajando con una pequeña *población* este operador presenta algunas diferencias en comparación con otros operadores del mismo tipo.

Una serie de pruebas se hicieron para crear el cruzamiento. Por ejemplo, si se parten los cromosomas de los padres en 2 y se recombinan para formar dos descendencias (cruce de 1 punto), esta estrategia producirá rápidamente una convergencia a un mínimo local y detendrá la optimización en la segunda iteración. Sin embargo este tipo de operadores son muy útiles en otros tipos de problemas y con poblaciones más grandes. Por lo tanto, para crear una nueva posible solución, sólo son necesarias las posiciones de los puntos ptEast y *ptWest*. Los polígonos de las ventanas están relacionados con la geometría y generadas aleatoriamente con el objetivo de evitar caer en un mínimo local, por lo que durante el cruce se pueden descartar. Dado que se trabaja con solo 5 generaciones, es necesario explorar todas las posibles soluciones durante el pequeño periodo de tiempo en el cual el AG se ejecuta.

Los cromosomas que serán cruzados, son partidos para generar una nueva descendencia. Luego aleatoriamente se decide cuál de las dos será retornado dentro de la *población*.

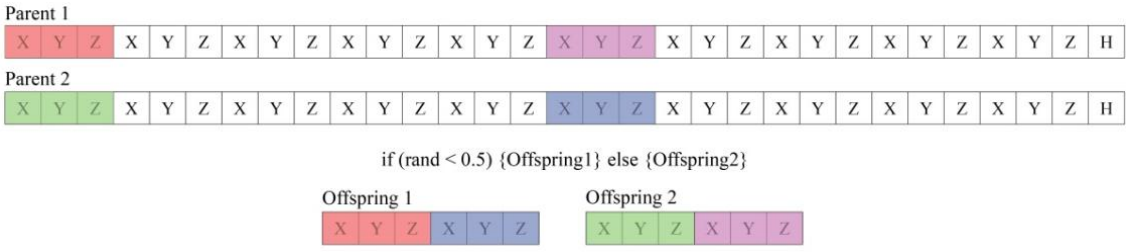


Figura 144. Representación gráfica del cruzamiento.

Al final, la descendencia son las nuevas coordenadas para los puntos *ptEast* y *ptWest*. Esta clase esta incrustada dentro de una función de GC con el nombre de *pfrmCrossOver()*. La descendencia y la mejor solución de la anterior *población* son conservadas para la siguiente generación. La descendencia incorpora una pequeña mutación en las coordenadas X e Y. La clase encargada de realizar la mutación es llamada *CSMutation.cs*. La mutación depende de un

factor llamado *MutationPercent* y los mejores resultados fueron obtenidos con valores alrededor del 0.5. Lo que se ha hecho para realizar las mutaciones es calcular un valor aleatorio entre 0 y 1, (*double r = Random.NextDouble()*). Luego el porcentaje de mutación es sumado y restado de la coordenada del punto para encontrar un valor mínimo y uno máximo (*double high = offspring [i] + percent, double low = offspring [i] – percent*). Luego de esto, una simple ecuación es implementada con el objetivo de guardar en el cromosoma la mutación.

$$offspringMutate[i] = low + r * (low - high).$$

Luego de que la mutación es realizada, la descendencia y la mejor solución son guardadas en la nueva *población* y 3 nuevos posibles candidatos son generados aleatoriamente con el objetivo de completar el tamaño de la *población*. El proceso se repite un total de 5 veces o hasta que la condición de parada lo haga definida por un umbral de consumo.

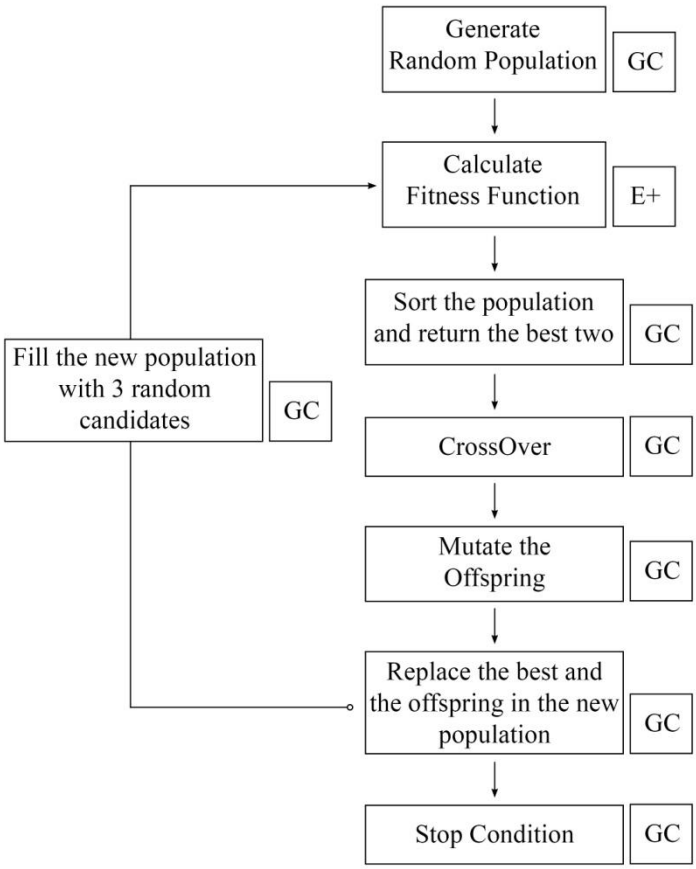


Figura 145. Imagen del proceso completo del *Micro*-AG.

8.4.3. Resultados

Para este trabajo el AG fue testeado 14 veces con los mismos datos de ambiente, con el mismo número de iteraciones (5) y el mismo tamaño de la *población*: 5. Los resultados fueron los siguientes: En 6 pruebas el sistema optimizó en todas las iteraciones, en otras 6 el sistema fue incapaz de optimizar en una y en dos de ellas el sistema no fue capaz de optimizar.

El porcentaje de consumo al inicio de las 12 pruebas exitosas fue de 4100503J, lo que significa en Vatios/hora [w*h]: 1139.028611111. El promedio de consumo después de la optimización fue de 3494576 J, lo que trasladado a Vatios/hora representa [w*h]: 970.715555556. Esta reducción representa un 17.21% menos de consumo en energía durante el día más caluroso y más frio del año.

El mejor *individuo* en todas las pruebas fue de 3023091.22 J, lo que significa en Vatios/hora [w*h]: 840.41935916.

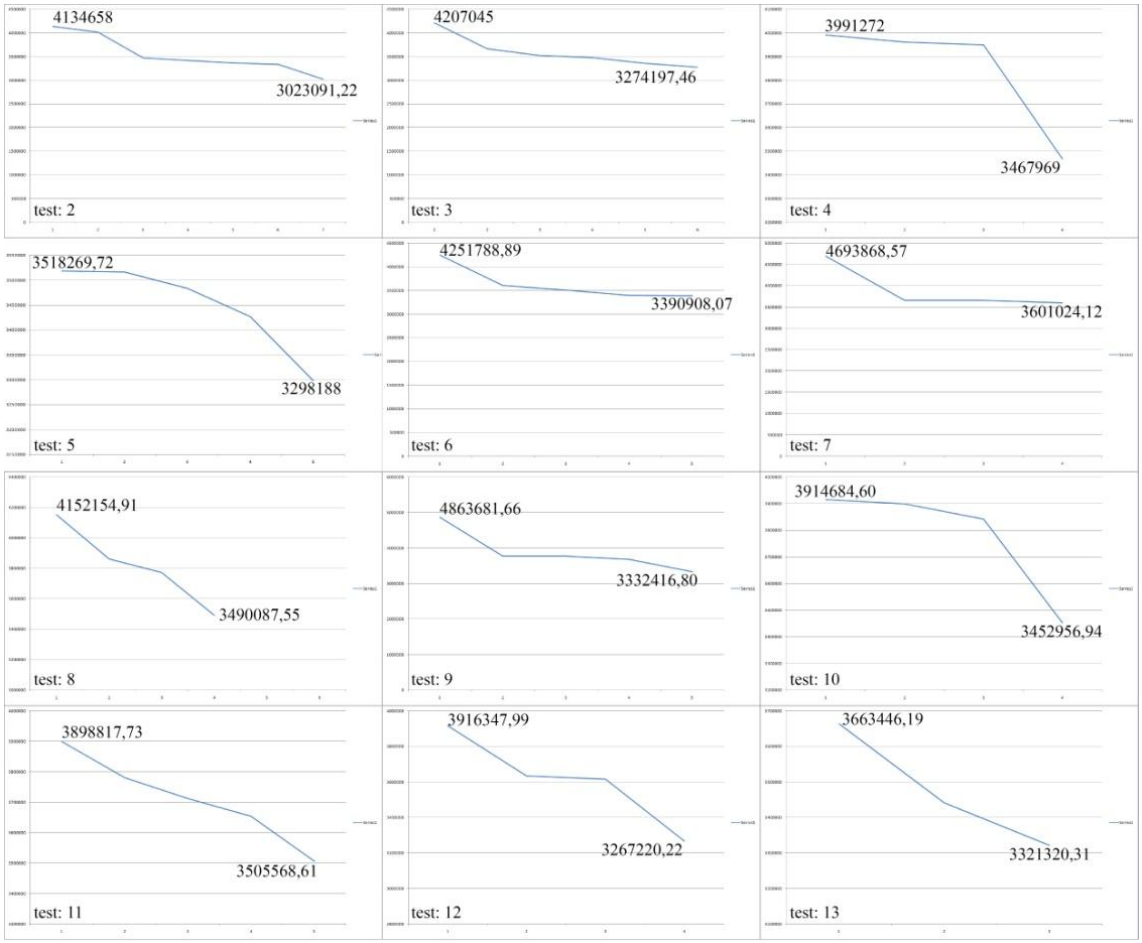


Figura 146. La imagen muestra las 12 pruebas exitosas donde el AG optimizó. Además aparecen los consumos al inicio del proceso y el resultado después del AG.

GEOMETRÍA

En relación a la geometría, los mejores resultados tienen algunas similitudes. Por ejemplo, el parámetro *coordXSrfEast* (que controla la coordenada X del punto Este), estuvo siempre alrededor del valor 10. En cambio el parámetro *coordYSrfEast* siempre estuvo por debajo del valor 5. Algo similar ocurre con los parámetros del lado Oeste. El parámetro *coordXSrfWest* siempre estuvo entre los márgenes del -5 y el -12 y el *coordYSrfWest*, en 11 oportunidades el AG encontró los mejores resultados entre los valores 5 y -1, pero en dos ocasiones los valores estuvieron entre el 6 y 7.

La siguiente tabla muestra los parámetros para las para los lados Este y Oeste antes y después del proceso de optimización.

	ptEast		*ptWest*		*Prueba Núm.:*
	coordXSrfEast	*coordYSrfEast*	*coordXSrfWest*	*coordYSrfWest*	
Antes	*18*	*2*	*-5*	*3*	*2*
Después	*10,87185619*	*4,788060669*	*-5,583290437*	*1,513245852*	
Antes	*15*	*1*	*-5*	*2*	*3*
Después	*12*	*-2*	*-5*	*-1*	
Antes	*15*	*-1*	*-9*	*-1*	*4*
Después	*11,4359281*	*1,394030335*	*-7,583290437*	*2,513245852*	
Antes	*13*	*1*	*-5*	*4*	*5*
Después	*12*	*5*	*-5*	*7*	
Antes	*18*	*0*	*-6*	*6*	*6*
Después	*11,4359281*	*1,394030335*	*-6,166580874*	*3,026491704*	
Antes	*17*	*2*	*-12*	*-3*	*7*
Después	*12*	*-1*	*-5*	*6*	
Antes	*17*	*3*	*-9*	*5*	*8*
Después	*11,40036427*	*-5,792729762*	*-12,34410405*	*0,878349137*	
Antes	*12*	*1*	*-10*	*-3*	*9*
Después	*10,1005464*	*-3,189094643*	*-6,344104051*	*1,878349137*	
Antes	*17*	*7*	*-5*	*7*	*10*
Después	*12*	*-3*	*-8*	*-1*	
Antes	*14*	*6*	*-8*	*4*	*11*
Después	*12,77375673*	*4,68267464*	*-6,268022689*	*4,941838809*	
Antes	*13*	*2*	*-8*	*0*	*12*
Después	*8,32127019*	*-3,951976079*	*-6,268022689*	*3,941838809*	
Antes	*12*	*4*	*-9*	*6*	*13*
Después	*9,54751346*	*-0,634650719*	*-7,268022689*	*2,941838809*	

Figura 147. Tabla de resultados, parámetros antes y después de la optimización

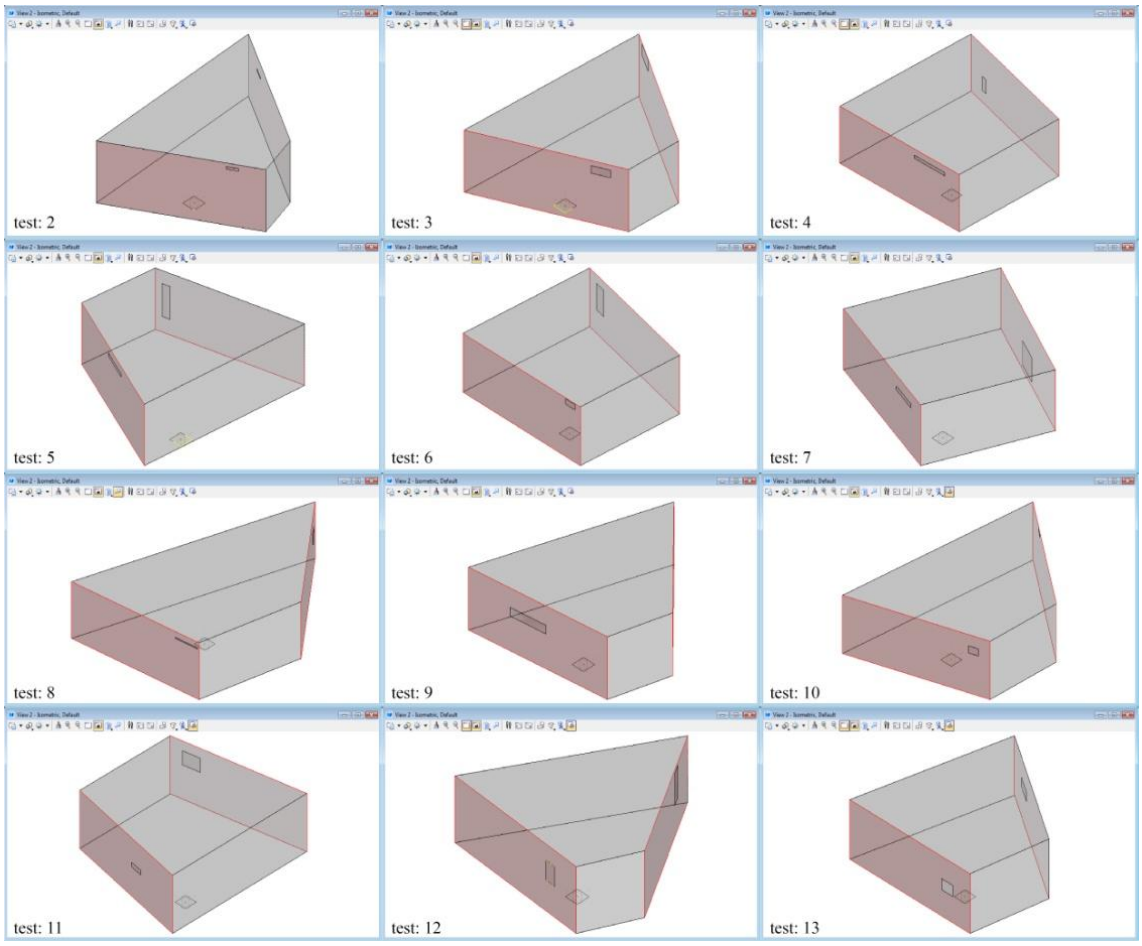


Figura 148. Secuencia de imagines que muestra los 12 modelos exitosos durante las pruebas.

8.4.4. Discusión

Los resultados demuestran que el sistema produjo en la mayoría de los casos muy buenos resultados, disminuyendo el consumo de HVAC en un 17%. Antes del proceso de optimización, el parámetro *coordXSrfEast* tuvo valores por encima del 12 en todos los casos y *coordXSrfWest* tuvo siempre valores bajos. Esto significa un gran espacio, el cual lógicamente necesita más consumo para mantener la temperatura a 20ºC que uno pequeño. Para los parámetros *coordYSrfEast* y *coordYSrfWest*, los valores en general fueron altos, lo que significa que las superficies Este y Oeste recibieron el sol directamente en la mañana y por la tarde. Este tipo de geometría hace aumentar los consumos de refrigeración en el día más caluroso, y si las ventanas son grandes, el consumo será un completo desastre para el día más frio también.

Las mejores soluciones en general tienen valores bajos en los parámetros *coordYSrfEast* y *coordYSrfWest*, lo que significa que la geometría es “doblada” para reducir el área expuesta al Este y Oeste. Los parámetros Y controlan la orientación del edificio, determinando cuanta área es expuesta al sol. Estos pequeños valores indican que, las superficies Este y Oeste se orientaron usualmente hacia el sur.

Para los parámetros *coordXSrfEast*, los valores, estuvieron alrededor del 10 y para *coordXSrfWest* estuvieron alrededor del -6 y sólo en una prueba alrededor del -12. Los valores X básicamente determinan la longitud del edificio, ambos parámetros empujan las superficies

que enfrentan el Este y Oeste. Un valor alto en el Este y un pequeño en el Oeste significa un largo edificio. Estos parámetros afectan directamente el volumen total del edificio y como consecuencia influyen en la actuación del consumo de HVAC.

Si bien el experimento estuvo enfocado en la manipulación de la geometría, las ventanas siempre fueron dejadas de manera aleatoria en cuanto a tamaño y posición. Sin embargo es posible observar algunas similitudes entre las ventanas de las diferentes pruebas. Por ejemplo en todos los casos el AG prefirió ventanas pequeñas ubicadas cerca de los bordes de las superficies a diferencia de una gran ventana ubicada en el centro como usualmente preferimos nosotros los arquitectos. Otra característica especial fue la disposición de las ventanas. Las ventanas del lado Este siempre fueron dispuestas en el lado derecho de la superficie y en el Oeste ocurrió exactamente lo mismo. Por lo que si vemos el edificio en dirección Este-Oeste, las ventanas Este se encuentran todas alineadas al lado derecho mientras que las ventanas Oeste al lado izquierdo. Además las ventanas del lado Este en casi todos los casos son verticales mientras que en el Oeste son horizontales. El AG dispuso 10 veces la ventana Oeste cerca del techo y propuso ventanas altas y horizontales. Pero en el lado Este fueron dejadas más al azar generando ventanas cerca del suelo. En 4 ocasiones el AG propuso ventanas cerca del techo, y cuando esto ocurrió el sistema redujo el volumen total del edificio.

El AG en todos los casos intentó reducir el volumen total del edificio y doblar su geometría enfrentando las ventanas Este y Oeste al sur en 10 ocasiones y al Norte en 2 y nunca enfrento el Este y Oeste directamente. El AG intento capturar más energía del Este que del Oeste, proponiendo ventanas grandes en el lado Este 9 veces. En general el sistema dejo ventanas más grandes en el lado Este a una altura media de la superficie, mientras que en el lado Oeste dispuso ventanas más pequeñas y horizontales cerca de la cubierta.

El AG fue capaz de generar a partir de 14 pruebas, 12 posibles soluciones con bajos niveles de consumo energético. Esto hace el edificio más sustentable, un tema que concierne la disciplina de la arquitectura. Pequeños consumos son reflejados en el presupuesto mensual del edificio, el cual está relacionado con el costo y mantención, otro tema ligado a la profesión.

El tamaño mínimo de las ventanas no fue restringido. En algunos casos el AG propuso ventanas extremadamente pequeñas, lo cual representa una inconfortable relación espacial entre el interior y exterior, por lo que está variable será incorporada en futuros trabajos.

El AG puede ser usado básicamente en 2 estados durante el proceso de diseño. El primero corresponde a usar el sistema como diagnóstico de potenciales problemas con el objetivo de sugerir diferentes caminos en la exploración del diseño. El arquitecto puede usar el AG para predecir los niveles de consumo y reorientar las decisiones de diseño para obtener soluciones sustentables. Por ejemplo, definiendo una geometría general del edificio en una ubicación específica y proponer diferentes configuraciones de ventanas y ver cuál de ellas puede ser un camino exploratorio.

En un segundo estado la herramienta puede ser aplicada en las fases últimas del proceso de diseño para incorporar una serie de restricciones y rigurosos cálculos de análisis térmico. Esto ayudará a balancear la geometría y la configuración de las ventanas para optimizar los consumos de HVAC, niveles de luz artificial u otras características medidas con *Energyplus*. Por ejemplo la optimización del tamaño del equipamiento de HVAC o la elección de los materiales para las ventanas.

Un simple análisis térmico, como el presentado en este trabajo, tarda alrededor de 3.7 segundos en un ordenador corriente. Este análisis genera toda la documentación necesaria

para leer los reportes y retornarle a la geometría la situación del consumo de HVAC. Este tiempo de análisis es ideal y un AG normal de 30 *individuos* y 300 generaciones habría sido la mejor idea en contraste con el sistema propuesto en este trabajo. Sin embargo este trabajo apunta a trabajar con grandes escalas de proyectos, donde los análisis toman más de 1 hora. Por ejemplo, 5 *individuos* por 5 generaciones son al menos 25 horas de cálculo computacional. Esta es la razón principal de proponer un *Micro*-AG.

El sistema en este estado se encuentra en actual desarrollo y algunas características deben ser comprobadas y revisadas. Por ejemplo, cuando una nueva *población* es remplazada, buenos reportes recomiendan guardar los *individuos* con buen *fitness* para ser incorporados en nuevas *poblaciónes*[100], en vez de generarlos aleatoriamente como es la estrategia que se plantea aquí.

En estados previos del desarrollo diferentes operadores genéticos fueron testeados (mutación, cruce, y selección natural) sin buenos resultados. Uno de los principales problemas con pequeñas poblaciones es caer en mínimos locales. En otras palabras, el alto poder de la herencia en el cruce (ventanas y parámetros) detienen la optimización en la segunda o tercera generación cayendo directamente en mínimos locales. Esta es la razón de porque se optó en dejar la generación de ventanas completamente aleatoria.

En total se construyeron 21 clases, la mayoría de ellas fueron operadores genéticos con diferentes características y otras fueron creadas para resolver problemas técnicos como exportar vértices en diferentes formatos, leer y escribir archivos etc. La parte importante de este trabajo es que todos los códigos están siendo la base para una librería de clases en C# incrustadas dentro de *GenerativeComponents* para implementar y usar Algoritmos Genéticos aplicados en arquitectura.

Actualmente el autor se encuentra trabajando en un modelo más complejo, con más características y un análisis más detallado. Algunas clases han sido reconstruidas y se han creado nuevas para alcanzar el objetivo de trabajar con grande edificios.

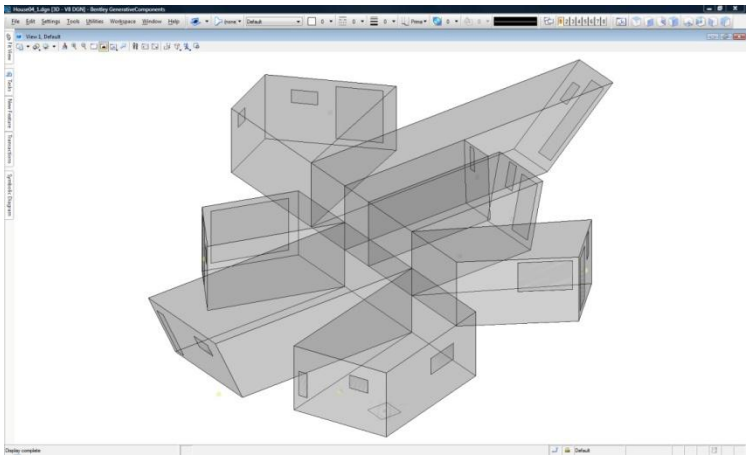


Figura 149. Actualmente el trabajo se encuentra centrado en una geometría con 8 zonas y 28 ventanas.

100Zitzler, E., Kalyanmoy, D., Thiele, L. and Coello Coello, C.A., Evolutionary Multi-Criterion Optimization, Springer, Zurich, 2001.

Conclusiones

En este trabajo se ha investigado el potencial de los Algoritmos Genéticos aplicados a la Arquitectura. Para llevar a cabo este estudio se han estudiado los conceptos básicos de la genética y de la selección de las especies de Darwin. Se han repasado los principales trabajos de los AGs aplicados a la arquitectura y al diseño. También se aprendió programación, lo que significó un largo camino en el que muchas veces se perdió el objetivo, del porque se estaba programando. Luego fue necesario aprender y estudiar diferentes técnicas de optimización y determinar cuándo y porque eran necesarios los AGs en ciertos problemas. Lógicamente no todos los problemas necesitan de AGs, solo se deben usar cuando no queda otra opción. Finalmente los 4 proyectos del capítulo 8 fueron un proceso en el cual hubo que resolver una serie de problemas matemáticos y técnicos. Otra de las dificultades a resolver fue encontrar y plantear problemas que necesitasen de AGs para ser resueltos. Para el proyecto de macro escala se basó en un conocido problema de combinatoria, el cual usualmente es enfrentado mediante AGs. En el proyecto de generación (SDG) se plantearon una serie de restricciones y se incluyó la función "*the developer's Manhattan function"*, la cual es resuelta mediante un AG en libro de Paul Coates "*programming.Architecture*". El tercer proyecto, sobre organización espacial, enfrenta un problema complejo que es el ordenamiento espacial dentro de los contornos del edificio y atañe a todos los arquitectos al menos alguna vez en la vida. El cuarto proyecto, sobre optimización térmica está basado en el trabajo de Caldas y Norford, el cual el autor quiso darle una revisión; incluir un AG programado en vez de usar un software e incluir la posibilidad de modificar la geometría para explorar diferentes soluciones formales. Durante la programación de los algoritmos la parte más simple fue escribir los códigos y la más compleja fue "calibrar" los diferentes AGs. Por ejemplo la mutación es clave porque un valor alto permite buscar en todos los rincones del espacio de búsqueda pero pierde la buena aptitud heredada de los padres. En cambio, un valor bajo corre el peligro de caer en un óptimo local pero hereda la aptitud de los padres. Además el valor de mutación depende del tipo de problema, del tamaño de la *población* y de otros factores. Definir el tamaño de la *población* y el tipo de cruce también es clave porque implican gasto de cómputo y afectan directamente en la selección. Calibrar un AG es una tarea compleja y de paciencia, donde las matemáticas no sirven, sólo sirve la experiencia y los reportes de otros investigadores.

Los resultados teóricos obtenidos, permiten concluir que los AGs aplicados a la arquitectura tienen un potencial como herramienta de diseño y para resolver problemas tipo multi-objetivo. Los AGs pueden ser utilizados en diferentes estados del proceso de diseño, sin embargo está tecnología, debido a su potencia, debería ser usada como una estrategia para hacer edificios más sustentables y económicos en cuanto a consumo de energía. Es una responsabilidad que se debe afrontar con la naturaleza.

Los AGs podrían aplicarse a la arquitectura como modelos propositivos de diseño, como es el caso del segundo trabajo (SDG) y como herramienta de optimización en los tres trabajos restantes. Para el primer caso, se podría explorar el uso de diferentes operadores genéticos o trabajar con diferentes niveles de mutación, por ejemplo. En este caso la idea podría ser la exploración y la búsqueda de forma. Incluso un resultado por debajo de la media de aptitud podría ser considerado interesante desde el punto de vista formal y utilizarse como punto de partida en el diseño de un edificio. Por el otro lado, diferentes problemas de optimización tienen asociados espacios de búsqueda completamente diferentes, por lo que no podría existir

un algoritmo genético universal que resolviera todos los problemas. Por ejemplo en el tercer trabajo: Organización espacial mediante AG, los mejores resultados fueron obtenidos usando una selección por torneo determinística de tamaño 6. En el cuarto experimento se utilizó una selección elitista, mientras que en el primer y segundo experimento se usó la selección por ruleta. Por lo tanto no es posible usar un AG genérico, sino que sería necesario plantear una librería de operadores para ser aplicados a diferentes problemas.

La evolución de las especies es un asunto infinitamente complejo, que en teoría podría sintetizarse a través de las matemáticas, pero en la práctica se está muy lejos. En ciertas condiciones los virus al entrar en un organismo pueden conectarse con células germinales y transmitirse como parte del gen. Además pueden pasar a células de una especie totalmente distinta transportando información genética precedente del primer anfitrión[101]. Lo que significa que los virus pueden transportar, tras una integración-extracción fragmentos de ADN de su huésped y transmitirlos a nuevas células. Como consecuencia, una información genética de un organismo podría ser transferida a otro gracias al virus. Entonces se podría pensar que esa transferencia de información podría efectuarse de una especie más evolucionada a una menos evolucionada. Este mecanismo de la naturaleza actuaría a contracorriente del que clásicamente utiliza la evolución. Posiblemente los esquemas de evolución tengan que abandonar el modelo de árbol y de la descendencia por un esquema más complejo.

Este trabajo abre vías de investigación para la exploración de la computación evolutiva aplicada a la arquitectura y el diseño. Diferentes caminos pueden seguirse, optimización, exploración de formas, fabricación, logística, etc. En general casi todos los problemas del mundo real son multi-objetivos y una buena estrategia es usar AGs. Además puede servir para que el arquitecto pueda volverse a plantear ciertas preguntas, sobre todo en la manera en cómo mejorar sus diseños en cuanto a sostenibilidad y fabricación. Desde este punto de vista pueden llegar a ser un método fundamental en el trabajo de la profesión y en otras disciplinas. Los AGs básicamente diseñan (proponen) familias de soluciones las cuales pueden llegar a ser muy buenas.

Los AGs son todo terreno, porque se adaptan a una gran cantidad de problemas (espacios de búsqueda) de manera eficiente y siempre retornan muy buenos resultados. Sus operadores genéticos permiten explorar el espacio de soluciones al mismo tiempo en diferentes direcciones a la vez. Sin embargo, un AG no es tan eficiente cuando existe un problema para el cual ya existe una técnica de resolución efectiva. En una autopista lógicamente ganará un deportivo por sobre un 4x4. No puede existir un AG genérico para todos los problemas, ya que los espacios de búsqueda son particulares para cada problema y por ende deben tratarse de manera particular.

LÍNEAS FUTURAS DE TRABAJO

Los 4 proyectos enfrentan problemas muy diferentes, sin embargo la arquitectura de los algoritmos es muy similar. Los esquemas se repiten pero el flujo de datos entre los diferentes operadores cambia. Además, el tipo de operador cambia dependiendo del espacio de búsqueda. Es por esto que se programaron más de un operador de cruce, de mutación y de selección.

[101] DELEUZE, Gilles. GUATTARI, Félix., Rizoma, Valencia, Pre-Textos, 2005, páginas: 24-25.

Se plantean dos líneas futuras de trabajo:

- La primera orientada a continuar desarrollando los 4 proyectos. Los 3 primeros sobre una tecnología más estable, incorporando nuevas variables y explorando las capacidades de cada uno. Por ejemplo, el SDG (proyecto 2) habría que conseguir la manera de poder manipular más cubos e incorporar otras variables como asoleamiento, flujos de viento y estructura. El proyecto 3 (*Voronoi*) es necesario cambiar la tecnología, hacerlo más robusto y buscar una manera de liberar la memoria ocupada. De esta manera se podría iterar el algoritmo más de 200 veces. Con respecto al proyecto 4 (optimización de consumos HVAC) es preciso mejorar el modelo paramétrico para poder incorporar más de 8 zonas dentro del modelo. También sería necesario continuar trabajando en este *micro*-AG para poder mejorar los resultados de optimización.
- La segunda línea de trabajo plantea el desarrollo de una Librería de AGs para Arquitectura (LAGA). Esto consistiría en diseñar y construir una serie de clases (herramientas) que incluyese diversos tipos de operadores genéticos y utilidades para que otros arquitectos pudiesen utilizar e implementar en sus diseños.

  HERRAMIENTAS BÁSICAS QUE DEBIESE INCLUIR LA LIBRERÍA DE AGS PARA ARQUITECTURA (LAGA)

  UTILIDADES:
  - Clase: Lectura de ficheros TXT.
  - Clase: Lectura de ficheros XLSX.
  - Clase: Escritura de Ficheros TXT.
  - Clase: *Batch*, una clase que ejecute programas.
  - Clase: IO, clase que facilite la interoperabilidad entre programas.

  OPERADORES GENÉTICOS:
  - Clase: decodificador (clase para facilitar la codificación de geometría en cromosomas o listas.
  - Clase: CrossOver, que realice diferentes tipos de cruce, dependiendo de las restricciones del problema y el cromosoma).
  - Clase: Mutation, que sirva para realizar diferentes tipos de mutación e intercambio de información dentro del cromosoma.
  - Clase: Selección natural, incluir diferentes tipos de selección, como método de ruleta, método de torneo, elitismo etc.
  - Clase: *fitness* (función de aptitud), una clase que facilité la creación y definición de funciones de aptitud para realizar la optimización.

  De cierta manera un pequeño paso ya se ha dado para implementar LAGA. Sin embargo la idea sería que otros arquitectos continúen desarrollando e implementando la librería. El objetivo de LAGA consistiría en ser útil a la comunidad y contribuyese a crear arquitectura más sustentable.

Las formas que produce la naturaleza, emergen de la interacción del medio-ambiente que rodea la forma, y una regla interna que produce esta forma. La forma modifica el ambiente y esté a su vez modifica la forma. Ambas entidades se relacionan estrechamente creando sistemas complejos, que a su vez se relacionan o forman parte de sistemas aún mayores. Está relación entre sistemas es lo que se conoce como simbiosis.

El material en la naturaleza se auto-organiza dependiendo de reglas y de su interacción con el ambiente, sus formas son concebidas como inacabadas porque necesitan ser capaces de adaptarse al medio-ambiente cambiante. El cruce y la mutación son las herramientas principales con las que cuenta la evolución para adaptarse. De cierta manera a la naturaleza no le interesa mejorar sus modelos, sólo generar una diversidad genética que asegure su existencia.

El proceso en el cual los sistemas naturales se organizan y como disponen el material en el espacio está estrechamente ligado con la búsqueda de la forma que podría ser aplicado a la arquitectura. Utilizar la matemática biológica como estrategia de diseño implica observar la naturaleza como patrones tridimensionales geométricos y aritméticos que se repiten como un algoritmo matemático, que tras cada iteración desarrolla o incorpora aleatoriamente variaciones en su programa natural de auto-organización.

Bibliografía

LIBROS

ALEXANDER C., ***Notes on the Synthesis of Form***. Cambridge, Harvard University Press: (1964).

BAGCHI T., ***Multiobjetive Scheduling by Genetic Algorithms***, Kluwer Academic Publishers, London, (1999).

BALMOND C., ***Informal***, Germany, Prestel. (2002).

BANDALA M., **Algoritmo Genético Multiobjetivo para problemas de calendarización con transferencia cero (Flowshop)**. Tesis de Maestría, Benemerita Universidad Autonoma de Puebla, México, (2005).

BARAN B. y HERMOSILLA A., ***Comparación de un sistema de Colonia de hormigas y una estrategia evolutiva para el problema del ruteo de Vehículos con ventana de tiempo en un contexto Multiobjetivo***, Centro Nacional de Computación, Universidad Nacional de Asunción; San Lorenzo, Paraguay. (1987).

CASTRO J., ***Creación de Portafolios de Inversión Utilizando Algoritmos Evolutivos Mutiobjetivo***. Tesis de Maestría en Ciencias, Centro de Investigación y Estudios Avanzados del Instituto Politécnico Nacional de México. (2005).

COATES P., ***Programming.Architecture***, New York, Routledge. (2010).

COELLO C., y TOSCANO G., ***A Micro Genetic Algorihm for Multiobjetive Optimizacion***. First Internacional Conference on Evolutionary MultiCriterion Optimization. Springer Verlag. Lecture notes in Computer Science. (2001).

COELLO C., VELDHUIZEN A. y LAMONT G., ***Evolutionary Algorithms for Solving Multiobjective Problems***. Kluwer Academic Publishers, USA. (2002).

CORNE D., KNOWLES J. y OATES M., **The Pareto-Envelope based Selection Algorithm for Multiobjetive Optimization, Parallel Problem Solving from Nature-PPSN VI**, Springer Lecture Notes in Computer Science, Springer, Berlin, (2000).

CORNE D., JERRAM N., KNOWLES J. y OATES M., ***PESA-11: Region-based Selection in Evolutionary Multiobjetive Optimization***. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001), Morgan Kaufmann Publishers. (2001).

DEB K., AMRIT P., AGRAWA S., PRATAB A. y MEYARIVAN, T., ***A Fast and Elitist Multiobjetive Genetic Algorithm: NSGA II***. IEEE Transactions on Evolutionary Computation, 6 (2): 182197. (2002).

DELEUZE G. y GUATTARI F., ***Rizoma***, Valencia, Pre-Textos. (2005).

DELEUZE G., ***El Pliegue, Leibniz y el Barroco***, Barcelona, Paidós Básica. (1989).

DIEGO-MAS J., ***Optimización de la distribución en planta de instalaciones industriales mediante algoritmos genéticos. Aportación al control de la geometría de las actividades***, Tesis doctoral, Universidad Politécnica de Valencia, (2006).

EVANS J. y ARCHER S. ***Diversification and the Reduction of Dispersion: An Empirical Analysis***. The Journal of Finance, 23 (5): 761 767. (1968).

FOXALL J., ***El Libro de Visual C# 2005***. Madrid, Anaya (2006).

FONSECA C. y FLEMING P., ***Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization***. In Stephanie Forrest, editor, Proceedings of the Fifth International Conference on Genetic Algorithms, pages 416-432, San Mateo, California, University of Illinois at Urbana –Champaign, Morgan Kauffman Publishers. (1993).

GECCO 2005 (Vol. 1 y 2), ***Genetic and Evolutionary Computation Conference***, Libro del congreso, New York, ACM SIGEVO (2005).

GOLDBERG D., ***Genetic Algorithms in Search, Optimization, and Machine Learning***. New York: Addison-Wesley. (1989)

GREINER D., **A summarized overview of evolutionary multiobjetive algorithms and new approaches**, CEANI, USA. (2000).

HENSEL M. y MENGES A. ***Emergence in Architecture***, Architectural Design 74, no 3. (2004).

HERTZ J., KROGH R. y PALMER R., ***Introduction to the Theory of Neural Computations.*** 1 ª ed. AddisonWesley Publishing Company. Boston. USA. (1991).

HOLLAND J. ***Adaptation in Natural and Artificial Systems, (An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence).*** Cambridge, London: MIT Press. (1975).

HOLLAND J., ***Emergence: From Chaos to Order.*** Reading, Mass.: Addison-Wesley. (1998).

HOLLAND J., ***Genetics Algorithms / Algoritmos Genético***s. Revista Scientific American. MIT Press. (1992).

HORN J., NAFPLIOTIS N. y GOLDBERG D., ***A Niched Pareto Genetic Algorithm for Multiobjetive Optimization.*** Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence. 1: 82 87. (1994).

JAYARAM k., ***Multiobjetive Production Scheduling***, M. Tech. Thesis, Deparment of industrial and Management Engineering, Indian Institute of Technology Kanpur. (1998).

JHONSON S., ***Sistemas Emergentes: O que tienen en común hormigas, neuronas, ciudades y software***, México, Fondo de cultura Económica, (2008).

JOYANES, RODRIGUEZ y CASTÁN, ***C++***, Madrid, MC GRAW HILL. (1999).

KNOWLES J. y CORNE D., ***Approximating the Nondominated Front using the Pareto Archived Evolution Strategy.*** Evolutionary Computation, Massachusetts Institute of Technology, (2000).

KOZA J., ***Genetic Programming: On the Programming of Computers by Means of Natural Selection.*** Cambridge: MIT Press. (1996).

LETHABY y WILLIAM, ***Architecture: An Introduction to the History of the Art of Building***, (T. Butterworth Ltd. (1911).

LIBERTY J. y HOVARTH D., ***Aprenda C++***, Madrid, Anaya. (2005).

LIDDAMENT T., ***The Computationalist Paradigm in Design Research***, Design Studies, Vol 20, No 1. (1999).

MICHELWICZ Z., ***Genetic algorithms + Data Structures = Evolutionary programs***. 3ª ed. Springer. (1996).

MITCHELL M., ***An Introduction to Genetic Algorithms***, Cambridge, Massachusetts. (1996).

O'ROURKE J., ***Computational Geometry in C***, Second Edition. New York, Cambridge University Press. (2005).

OSYCZKA A. ***Multicriterion optimization in engineering with Fortran Programs.*** 1ª ed. Ellis Horwood Limited. (1984).

PARETO W., ***Course de Economie Politique***, Volume I & II. F. Rouge, Lausanne, Francia, (1896).

PÉREZ SERRADA A., ***Una Introducción a la Computación Evolutiva***. Tesis de Maestría. Universidad de Valladolid, España, (1996).

PETIT C., ***Touched by Nature: Putting Evolution to Work on the Assembly Line.*** U.S. News and world Report, (1998).

RASHEED K., ***A genetic algorithm for continuous design optimization. Technical Report DCS-TR-352***. Department of Computer Science, Rutgers University. New Brunswick, NJ. Ph.D., (1998).

SALOMON M., ***Algorithms for the vehicle routing and scheduling problem with time window constraints***, Oper. Res., (1987).

SCHAFFER J., ***Multiobjetive Optimization with Vector Evaluated Genetic Algorithms.*** First Internacional Conference on Genetic Algorithms. (1985).

SEGARRA J., ***Vida artificial: del caos al orden***. Alzira: España. Algar Editorial. (2002).

SHEA K., AISH R. y GOURTOVAIA M., ***Towards integrated performance-driven generative design tools,*** [eCAADe '03] Graz University of Technology (Austria): 553-560, http://iam.tugraz.at/~e/cd/ecaade_ files/pdfs/Friday/149_shea.pdf, [01/11/07].

SHOAF J. y FOSTER J., ***A Genetic Algorithms Solution to the Efficient Set Problem: A Tecnique for Portfolio Selection Based on the Markowitz Model***. In Proceeding of the Decision Sciencies Institute, Annual Meeting Orlando Florida, USA. (1996).

SRINIVAS N. y DEB K., ***Multiobjetive Optimization Using Nondominated Sorting in Genetic Algorithms.*** Evolutionary Computation. Department of Mechanical Engineering, Indian Institute of technology, Kanpur, India, (1994).

STEADMAN P., ***The Evolution of Designs: Biological Analogy in Architecture and the Applied Arts***. New York, Cambridge University Press. (1979).

TAGAMI T. y KAWABE T., ***Genetic Algorithm with a Pareto Partitioning Method for Multiobjetive Flowshop Scheduling.*** Kagawa Junior College, Japan, University of Tsukuba, Japan (1998).

TESTA P. y WEISER D., ***Emergent Structural Morphology***, Architectural Design 72, no1. (2002).

TORRES E., GUTIÉRREZ M. y GÁLVEZ P. ***Algoritmo Genético (Micro AG) en Optimización de Carteras.*** VI Conferencia Internacional de Finanzas, Universidad de Santiago de Chile. (2006).

TOSCAZO G., ***Optimización Multiobjetivo Usando un Micro Algoritmo Genético***. Tesis de Maestría. Universidad Veracruzana LANIA, México. (2001).

TURING A., ***Morphogenesis***, Netherlands, P.T. Saunders. (1992).

WHITLEY D., RANA S. y HECKENDORN R. ***Representation Issues in Neighborhood Search and Evolutionary Algorithms.*** Genetic Algorithms and Evolution Strategies in Engineering and Computer Science. (1998).

WOLFRAM S., ***A New Kind of Science***. Washington: EEUU. Wolfram Publisher. (2002).

ZITZLER E., KALYANMOY D., THIELE L. y COELLO COELLO C, ***Evolutionary Multi-Criterion Optimization***, Zurich, Springer, (2001).

PAGINAS WEB

http://unfccc.int/2860.php [13-05-2010]

http://www.ipcc.ch/ [13-05-2010]

http://www.obitko.com/tutorials/genetic-algorithms/index.php [07-05-2010]

http://lancet.mit.edu/~mbwall/presentations/IntroToGAs/ [07-05-10]

http://www.talkorigins.org/faqs/genalg/genalg.html [07-05-10]

http://www.genetic-programming.org/ [07-05-2010]

http://www.aaschool.ac.uk/publications/ea/intro.html [08-05-10]

ftp://ftp.forum8.co.jp/forum8lib/pdf/VRsymposium/harvard-2-2.pdf [08-05-10]

http://projects.csail.mit.edu/emergentDesign/*Genr8*/ [07-05-10]

http://www.armyofclerks.net/ [07-05-10]

http://www.inive.org/members_area/medias/pdf/Inive%5CIBPSA%5CUFSC558.pdf [14-05-2010]

http://www.lsi.upc.es/~iea/transpas/9_geneticos/sld002.htm [02-09-10]

http://www.soton.ac.uk/~ajk/ [02-09-10]

http://www.elguille.info/colabora/2007/lafbegMSI_AlgoritmoGenetico.htm [02-09-10]

http://www.laputan.org/metamorphosis/metamorphosis.html#Abstract [02-09-10]

http://hobbiton.thisside.net/genetic/ [02-09-10]

http://kk.org/outofcontrol/ [02-09-10]

http://jaga.sourceforge.net/ [02-09-10]

http://www.fundacion.telefonica.com/at/delanda.html [02-09-10]

http://www.cddc.vt.edu/host/delanda/pages/algorithm.htm [02-09-10]

http://www.jweimar.de/ZAscriptmml/gliederung.html [02-09-10]

Glosario

A

Algoritmo = Regla geométrica o aritmética que se repite.

Algoritmo Genético = Algoritmo de búsqueda basado en las estrategia de la selección natural de Darwin.

AG = Algoritmo Genético.

AGs = Algoritmos Genéticos.

Autómata celular = Un autómata celular consiste en un espacio de N dimensiones dividido en celdas las cuales pueden encontrarse en dos o más estados de actuación (vivo, muerto, a la espera, etc.). El estado de cada celda depende de una serie de condiciones en relación a las celdas vecinas. Los autómatas celulares son actualmente usados en Bío-Ingeniería para predecir el comportamiento de células, Teoría de juegos, comportamientos urbanos, búsquedas por internet, etc.

B

Bueno = Se dice que un *individuo* es "bueno" cuando su *fitness* está cerca de cumplir el objetivo de la función de aptitud.

C

Ciencia de la complejidad = La ciencia de la complejidad estudia los sistemas biológicos y físicos que están compuestos por N cantidad de partes interconectadas que retroalimentan el sistema con información. El resultado de la interacción de estás partes generan nuevas propiedades y comportamientos que no pueden explicarse desde las partes aisladas, sino que desde la observación completa del conjunto. Por ejemplo, los cardúmenes, las colonias de hormigas, la interacción en la ciudad, etc.

Codificación = Representación de una solución a un problema mediante una lista de valores. La idea de codificación sirve para manipular más fácilmente los operadores genéticos.

Cóncavo = Un Polígono es cóncavo cuando al menos uno de sus ángulos interiores es mayor a 180º.

Converge = Un Algoritmo Genético se dice que converge cuando la población de soluciones centra su búsqueda en una zona determinada del espacio de búsqueda. Si el algoritmo no converge es porque la media de aptitud de la población se encuentra en los mismos valores que en la generación inicial. Si la población converge demasiado rápido, se corre el riesgo de no explorar todo el espacio de búsqueda y descartar mejores soluciones.

Convexo = Un polígono es convexo cuando todos sus ángulos interiores miden menos de 180º y todas sus diagonales son interiores. En un polígono convexo todos los vértices "apuntan hacia afuera de la figura".

Cromosoma = Es la manera de cómo se designa la lista de valores codificados del modelo. Usualmente el cromosoma es representado una lista de valores binarios 1s y 0s.

Cruce (Crossover) = es un tipo de operador genético que permite que dos *individuos* intercambien segmentos de su cromosoma, produciendo uno o dos *individuos* llamados descendencia artificial. Este proceso simula la combinación de los cromosomas del padre y la madre en la reproducción sexual.

D

Diploide = Organismos que tienen pares de cromosomas. La mayoría de las especies que se reproducen sexualmente son diploides, incluyendo a los seres humanos.

Diseño generativo = El diseño generativo es una estrategia de diseño que ha sido implementada en diversos campos como el arte, arquitectura, ingeniería, diseño etc. La idea fundamental de esta estrategia consiste en que el diseño producido es generado mediante un conjunto de reglas que administran los diferentes aspectos del producto. Esta estrategia permite explorar de manera rápida y económica diferentes posibilidades de diseño. Usualmente el diseño generativo está representado por un esquema de relaciones y variables que controlan los parámetros dentro del gráfico.

Diseño paramétrico = En el diseño paramétrico, los aspectos del modelo dependen de relaciones entre sus diferentes partes. Creando y modificando estas relaciones el modelo se modifica. Usualmente un modelo paramétrico es definido por reglas y restricciones, las cuales definen los aspectos del edificio y sus relaciones entre ellas.

En un modelo dibujado, la geometría es explícita, se expresa con claridad, porque es lo que vemos dibujado. En cambio sus reglas son implícitas, se entiende que están incluidas aunque no se diga cómo ni tampoco aparezcan (siempre hay reglas y restricciones en un modelo arquitectónico). La diferencia es que la herramienta de modelado no mantienen los seguimientos de estas reglas, por lo que es el usuario quién debe realizarlo. En contraste, en los modelos paramétricos, las reglas son explícitas, porque se expresan de manera clara y exacta. Sin embargo su geometría es implícita, en el sentido que se entiende como parte del modelo aunque no aparezca gráficamente. Por ejemplo los vectores o planos en un modelo paramétrico.

E

Emergencia = La emergencia es una característica que surge espontáneamente de un proceso de auto-organización. La idea es que simples reglas, sean capaces de producir complejos comportamientos.

F

Fitness = Es el valor por el cual se evalúa a cada individuo. Por ejemplo si el objetivo consiste en disminuir el *fitness*, los *individuos* con valores pequeños tendrán un mejor *fitness* que los grandes.

Función de aptitud = Corresponde a una función “F(X)” que da una calificación sobre qué tan bueno es un conjunto de genes (cromosoma) para obtener un objetivo dado.

G

Gen = Representa el valor de una variable dentro del cromosoma, aunque también se pueden usar un grupo de genes para representar una variable más compleja. De manera análoga a los

genes biológicos, el gen nos indica el color de ojos, de piel, la altura, etc. Si el gen es dominante, perdurará en la herencia que se haga dependiendo del filtro de selección. Si el gen es recesivo, irá difuminándose a lo largo de las generaciones según la presión del medio ambiente.

H

Haploide = Son los organismos que sólo tienen un par de cromosomas. Durante la reproducción sexual los genes de cada padre son intercambiados entre cada par de cromosomas para formar un gameto, que es haploide. Luego los gametos de ambos padres se emparejan para crear un grupo de cromosomas diploides.

Dr. John Henry Holland = Inventor de los Algoritmos Genéticos.

I

Individuo = Usualmente se designa individuo a un modelo dentro de la población, que engloba el cromosoma y la evaluación de aptitud.

Inteligencia Artificial = la inteligencia artificial es la disciplina que se encarga de construir procesos que al ser ejecutados sobre una arquitectura física producen acciones o resultados que maximizan una medida de rendimiento determinado. Se basan en la secuencia de entradas percibidas y en el conocimiento almacenado en tal arquitectura.

J

K

L

Longitud de la cadena = Es el número de genes que componen el cromosoma. O sea el número de variables que tiene el modelo.

M

Método de selección = La selección de *individuos* debe permitir que los *individuos* mejor dotados produzcan mayores descendientes que los menos dotados, Esto significa que los "genes" de los *individuos* más adaptados se propagarán en sucesivas generaciones hacia un número de *individuos* creciente.

Morfogénesis = La morfogénesis estudia el proceso que controla la distribución organizada de las células, proceso que aparece a lo largo del desarrollo embrionario de un organismo y que da lugar a las formas características de los tejidos biológicos, de los órganos y de la anatomía corporal.

Mutación (Mutation) = Es un operador genético que modifica de manera aleatoria los cromosomas de los *individuos*. Esto es hecho por dos motivos. El primero, es para que exista una probabilidad de mejoría de la aptitud en la siguiente generación y la segunda, es para abarcar el mayor espacio de búsqueda posible.

N

Ñ

O

Operadores Genéticos = Grupo de clases y funciones programadas que imitan de cierta manera el cruce, la descendencia, la mutación y otros comportamientos de la selección natural de las especies.

Optimización multiobjetivo = Es definida como un problema de optimización que presenta dos o más funciones objetivo. El inconveniente principal que presenta este tipo de problemas en relación a un modelo de objetivo único, radica en la subjetividad de la solución encontrada. Un problema multiobjetivo no tiene una solución óptima única, más bien, genera un conjunto de soluciones que no pueden ser consideradas diferentes entre sí.

Óptimo de Pareto = Es aquella situación en la cual se cumple que no es posible beneficiar a más elementos de un sistema sin perjudicar a otros. Se basa en criterios de utilidad: si algo genera o produce provecho, comodidad, fruto o interés sin perjudicar a otro, provocará un proceso natural de optimización hasta alcanzar el punto óptimo.

Ordenamiento de burbuja, *Bubble Sort* = Es un simple algoritmo de ordenamiento. Funciona revisando cada elemento de la lista que va a ser ordenada con el siguiente, intercambiándolos de posición si están en el orden equivocado. Es necesario revisar varias veces toda la lista hasta que no se necesiten más intercambios, lo que significa que la lista está ordenada.

P

Polimorfismo = Significa la facultad de asumir muchas formas, en programación se refiere a la facultad de llamar a las distintas versiones que adopta un método definido en una clase y redefinido en sus clases derivadas, usando el mismo medio de acceso: una referencia a la clase superior.

Programación Orientada a Objetos (POO) = Tecnología de programación que imita como son las cosas en el mundo real, con propiedades y acciones. Permite, herencia, abstracción, polimorfismo y encapsulamiento.

Q

R

Red Neuronal Artificial = Programa informático que imita el funcionamientos del sistema nervioso de los animales y es capaz de aprender y reconocer patrones.

S

Selección natural = Proceso por el cual una especie se adapta a su medio ambiente. La selección natural lleva al cambio evolutivo cuando *individuos* con ciertas características poseen una tasa de supervivencia o reproducción más alta que los otros *individuos* de la población y pasan estas características genéticas heredables a su descendencia.

T

Tupla = Una tupla, en matemáticas, es una secuencia ordenada de objetos. Es una lista con un número limitado de objetos. Las tuplas se emplean para describir objetos matemáticos que tienen estructura, es decir que son capaces de ser descompuestos en un cierto número de componentes. El término tupla se generó sencillamente de una generalización de la siguiente secuencia: dupla, tripla, cuádrupla, quíntupla,... n-tupla.

U

V

Voronoi = Los Diagramas de *Voronoi* son una clase de patrones que ocurren en la naturaleza de manera espontánea a cualquier escala y pueden ser descritas como una serie de células, perfectamente definidas que son todas adyacentes entre sí y ninguna célula comparte su área con otra célula. Los diagramas de *Voronoi* se construyen a partir de un conjunto de puntos en el plano euclidiano. Se empieza por uno de los puntos dentro del conjunto y se construye la mediatriz entre este punto y todos los otros puntos. La intersección de las mediatrices más cercana al punto encierran una región que es conocida como polígono de *Voronoi*, el proceso se repite hasta que se ha completado con todos los otros puntos.

W

X

Y

Z

Anexos

Código de Travelling Salesman Problem como Modelo Generativo Urbano.

```
Option Explicit
'Script written by <carlos de la b>
'Script copyrighted by <designemergente.org>
'Script version miércoles, 13 de mayo de 2009 12:34:40

'////////////////////////////////////////// File Creates
//////////////////////////////////////////

Call data()
Sub data()

	'//mensaje de bienvenida
	Dim ret
	ret = Rhino.MessageBox("Select the Points to Start", 1 + 64, "Genetic
Algorithm for TSP V.1,    Carlos de la Barrera ")

	If ret = 1 Then
		Dim arrPts : arrPts = CreateCity()
	Else
		Call rhino.Print("Execution Canceled, ¡¡¡ save the whales !!!")
		Exit Sub
	End If

	'//Data inputs
	Dim arrItems: arrItems = (Array("_Size of the Initial Population : 10 ->
100", "_Natural Selection Pointer: 10 -> 50", "_CrossOver Pointer : 1 -> " &
CStr(Int((UBound(arrPts) * 0.5))) & " -> " & CStr(UBound(arrPts)), "_Mutation
Rate : 1% -> 100%", "_Number of Iterations : 10 -> 500", "_Number of Families
: 3 -> 20","_Child Mutation Pointer : 1 -> 5"))

	Dim arrValues: arrValues = (Array(20, 50, Int(UBound(arrPts) * 0.5),
100, 100, 10, 1)) '//values by default
	Dim strMessage: strMessage = "Set the Currents Data to initialize the
G.A."
	Dim strTitle: strTitle = "Genetic Algorithm for TSP V.1,    Carlos de la
Barrera "

	Dim arrSet: arrSet = Rhino.PropertyListBox(arrItems, arrValues,
strMessage, strTitle)

	If Not IsNull (arrset) Then '//if everything is ok, then transform the
data in values.
		Dim arrVal(6)
		arrVal(0) = CInt(arrSet(0)) '//Initial Population
		arrVal(1) = CInt(arrSet(1)) '//Natural Selection
		arrVal(2) = CInt(arrSet(2)) '//CrossOver Pointer
		arrVal(3) = CInt(arrSet(3)) '//Mutation Rate
		arrVal(4) = CInt(arrSet(4)) '//Number of Iterations
		arrVal(5) = null 'CInt(arrSet(5)) '//Number of Families
		arrVal(6) = CInt(arrSet(6))      '//Child Mutation
```

```
	End If

	Call Principal(arrVal, arrPts) '//enviamos los datos al AG

End Sub

Sub Principal(V, ByVal cityData)

	Dim i, j, counter : counter = 0
	Dim InitPopulation()
	Dim best: best = 10000000

	'////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////
	For i = 0 To V(0) - 1 'number of random individuals
		ReDim Preserve InitPopulation(i)
		InitPopulation(i) = mutation(cityData) '//mutations of the
original chromosome
	Next
	'////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////

	Dim OriginCrv: OriginCrv = Rhino.AddPolyline(InitPopulation(0)) '//draw
the curve, the best for now
	boxes = myfuncboxes(OriginCrv)

	Do


	'////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////
		Dim OrgnBest : OrgnBest = BestRecord(InitPopulation) '//the
shortest Length and the best individual

	'////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////


	'////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////
		Dim RWheel : RWheel = NaturalSelection(InitPopulation,
OrgnBest(0), V(1))'//under the natural selection laws but in order the
complete population

	'////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////

		Dim flia
		For j = 0 To UBound(InitPopulation) - 3 Step 3
			flia = Generation(RWheel, V)
			InitPopulation(j)     = flia(0)
			InitPopulation(j + 1) = flia(1)
			InitPopulation(j + 2) = flia(2)
		Next


	'////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////
		Dim bestInd: bestInd = BestRecord(InitPopulation) '//buscamos el
mejor de la nueva población.

	'////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////

		Dim tempCrv : tempCrv =
Rhino.AddCurve(InitPopulation(bestInd(1)), 1) '//draw a temporal curve
		Call Rhino.DeleteObject(tempCrv) '//delete the temporal
individual
```

```
		'//conditional
		If best > bestInd(0) Then
			best = bestInd(0)
			Call Rhino.DeleteObject(OriginCrv)
			Call Rhino.DeleteObjects(boxes)

			OriginCrv = Rhino.AddPolyline(InitPopulation(bestInd(1)))
			Dim boxes : boxes = myfuncboxes(OriginCrv)

			Call Rhino.Print(bestInd(0))
		End If
	Loop

End Sub

'//distance of the path
Function EuclideanDist(Path)

	Dim i
	For i = 0 To UBound(Path) - 1
		EuclideanDist = EuclideanDist + Sqr((Path(i)(0) - Path(i + 1)(0))
^ 2 + (Path(i)(1) - Path(i + 1)(1)) ^ 2 + (Path(i)(2) - Path(i + 1)(2)) ^ 2)
	Next

End Function

'//store the points in the array
Function CreateCity() 'generation of the cities

	Dim arrCity

	arrCity = Rhino.GetPointCoordinates ("Select the Points to Test")
	If IsNull (arrCity) Then Exit Function

	CreateCity = arrCity

End Function

'//mutation crossover
Function mutation(ByVal arrD) 'chromosome mutation

	Dim P, temp, Down, Up, i

	Down = LBound(arrD)
	Up = UBound(arrD) + 1

	For i = 0 To UBound(arrD) 'pointer es el número de veces que
desordenaremos el array.

		Randomize
		P = Int(Down + Rnd() * (Up + Down)) 'index
		temp = arrD(i)
		arrD(i) = arrD(P)
		arrD(P) = temp
	Next

	mutation = arrD
End Function

'//mutationChild
Function mutationChild(ByVal childM, pointer) 'chromosome mutation

	Dim P, PC, temp, Down, Up, i, j

	Down = LBound(childM)
	Up = UBound(childM)
```

```
	Randomize '//fin a random element in the gen.
	P = Int(Down + Rnd() * (Up + Down))

	Do
		PC = Int(Down + Rnd() * (Up + Down))

	If (P <> PC) Then Exit Do
	Loop

	temp = childM(PC)
	childM(PC) = childM(P)
	childM(P) = temp

	mutationChild = childM

End Function

'//the best individual
Function BestRecord(Pop) '//find the shortest path

	Dim i, c
	Dim short: short = 100000000000000
	Dim dist

	For i = 0 To UBound(Pop)
		If Not isNull(Pop(i)) Then

			dist = EuclideanDist(pop(i))
			If short > dist Then
				short = dist
				c = i
			End If

		End If

	Next

	BestRecord = array(short, c)

End Function

'//Natural Selection
Function NaturalSelection(ByVal Pop, bestInd, pointer)

	Pop = SortPopulation(Pop) '//order the population to find the
nonpolinomical curve

	Dim i, j, counter: counter = 0
	Dim dist, index
	Dim RouletteWheel()

	For i = 0 To UBound(Pop)

		dist = EuclideanDist(Pop(i))
		index = Int((bestInd / dist * (pointer / (i + 1))) + 5)

		For j = 0 To index
			ReDim Preserve RouletteWheel(counter)
			RouletteWheel(counter) = Pop(i)
			counter = counter + 1
		Next

	Next

	NaturalSelection = RouletteWheel

End Function
```

```
'//familie Generation
Function Generation(ByVal Pool, V)

	Dim PoDad, PoMom, i, j '//here we select two pointers to find a mother
and a father to create the croosover
	Randomize
	PoDad = Int(Rnd * UBound(Pool))
	PoMom = Int(Rnd * UBound(Pool))

	Dim dad, mom, child '//family
	dad = Pool(PoDad)
	mom = Pool(PoMom)

	child = CrossOver(dad, mom, V(2))
	child = MutationChild(child, V(6))

	'-------------------0    1     2
	Generation = Array(dad, mom, child)

End Function

'//CrossOver
Function CrossOver(ByVal parent1, ByVal parent2, cutter)

	Dim child()
	Dim i, j, c : c = 0

	For i = 0 To cutter
		ReDim Preserve child(i)
		child(i) = parent2(i)
	Next

	'//here the las part of the dna
	Dim T: T = UBound(child) + 1
	For i = 0 To UBound(parent1)
		For j = cutter + 1 To UBound(parent2)

			If(parent1(i)(0) = parent2(j)(0)) And (parent1(i)(1) =
parent2(j)(1)) Then
				ReDim Preserve child(T)
				child(T) = parent1(i)
				T = T + 1
			End If
		Next
	Next

	CrossOver = Child

End Function

'//Sort the Population
Function SortPopulation(ByVal Pop)

	Dim i
	Dim SortPop()
	For i = 0 To UBound(Pop)
		'//find and sort
		Dim nBest : nBest = BestRecord(Pop)
		ReDim Preserve SortPop(i)
		SortPop(i) = Pop(nBest(1))
		Pop(nBest(1)) = Null

	Next
	sortPopulation = SortPop

End Function

'//Building Solutions 1
```

```
Function myfuncboxes(IDPol)

	Dim arrpts : arrpts = Rhino.PolylineVertices(IDpol)
	Dim nextPt
	Dim arrSrfs()

	Call Rhino.EnableRedraw(False)
	Dim i : For i = 0 To UBound(arrpts)
		If(i <> UBound(arrpts)) Then
			nextPt = arrpts(i + 1)
		Else
			nextPt = arrpts(i - 1)
		End If
		ReDim Preserve arrSrfs(i)
		arrSrfs(i) = myfuncSrf(arrpts(i), nextPt)
	Next
	Call Rhino.EnableRedraw(True)

	myfuncboxes = arrSrfs

End Function

Function myfuncSrf(ptO, ptT)

	Dim dist : dist = 0.30 * (Sqr((ptO(0) - ptT(0))^2 + (ptO(1) - ptT(1))^2
+ (ptO(2) - ptT(2))^2))

	If dist < 7 Then
		dist = 7
	End If

	Dim ac, bc
	If Rnd < 0.5 Then
		ac = rnd * 4 : bc = 0
	Else
		ac = 0 : bc = rnd * 4
	End If

	Dim s : s = Rhino.AddPlaneSurface(Rhino.PlaneFromFrame(ptO,
Array(1.0,0.0,0.0), Array(0.0,1.0,0.0)), dist + ac, dist + bc)
	s = Rhino.MoveObject(s, Rhino.EvaluateSurface(s, Array(dist * 0.5, dist
* 0.5)), ptO)

	'//calc the angle
	Dim a : a = Rhino.Angle2(Array(ptO, Rhino.EvaluateSurface(s, Array(dist,
dist * 0.5))), Array(ptO, ptT))
	s = Rhino.RotateObject(s, ptO, a(0), , False) '//and rotate

	Dim l : l = Rhino.AddLine(ptO, Array(ptO(0), ptO(1), ptO(2) + dist))
	myfuncSrf = Rhino.ExtrudeSurface(s, l)
	Call Rhino.DeleteObjects(Array(s, l))'clean the draw

End Function
```

Código de AG como Sistema de Diseño Generativo.

```
Option Explicit
'Script written by <carlos de la b>
'Script copyrighted by <designemergente.org>
'Script version Monday, 01 February 2010 09:30:11

'////////////////////////////////////////// Public Variables
//////////////////////////////////////////
Dim objInit, text
Dim mutRate, strTitle
Dim MaxSelection, factordecay
Dim selectCO

Call LoadData()
Sub LoadData()

	Dim Filter, FileName
	Filter = "Text File (*.txt) | *.txt"
	FileName = Rhino.OpenFileName("Select a txt file to place the GA Data",
Filter)
	If IsNull(FileName) Then Exit Sub

	Set objInit = CreateObject("Scripting.FileSystemObject")
	Set text = objInit.CreateTextFile(FileName, True)

	text.Writeline("//	DATA GENETIC ALGORITHM - Form Finding Generation
//")
	text.Writeline(" ")
	text.WriteLine("//programmed by carlos de la barrera, 01 February 2010,
cidelab@gmail.com, http:\\www.designemergente.org")
	text.Writeline(" ")
	text.Writeline(" ")
	text.Writeline("//Time Date Experiment: " & CStr(Now))

	'////////////////////////////////////////////////////////////////////////
//////////////////////
	'////////////  User Inputs
//////////////////////////////////////////////////////////////
	'////////////////////////////////////////////////////////////////////////
//////////////////////
	Dim arrItems: arrItems = (Array("_Number of boxes in X : 5 -> 100",
"_Number of boxes in Y : 5 -> 100", "_Number of boxes in Z : 5 -> 100",_
		"_Number of Individuals per Generation (multiple of 4) : 8 ->
300","_Number of Generations : 10 -> 1000","_Select The CrossOver Type :
UniformCroossOver or SinglePtCrossOver ",_
		"_Mutation rate : 0 -> 1", "_Fitness Sequence byte : 0 Or 1" ,
"_Fitness Length Sequence : 1 -> 10", "_Fitness Point Thershold proximity : 1
-> your criteria", _
		"_Maximum Indivuduals In the Natural Selection : 10 ->
100","_Curvature Decay : 0 -> 1" ,"After Set the Data, Place the points For
the Proximity Evaluation"))

	'//values by default
	Dim arrValues: arrValues = Array(10, 5, 5, 20, 50,"SinglePtCrossOver",
0.3,1, 3, 3, 50, 0.7,  "Don't fill or change this line :" & CStr(Now) )
	Dim strMessage: strMessage = "Set the Currents Data to initialize the
G.A."
	strTitle = "GENETIC ALGORITHM - Form Finding Generation V.1"

	Dim arrSet: arrSet = Rhino.PropertyListBox(arrItems, arrValues,
strMessage, strTitle)
```

```
	If Not IsNull(arrset) Then '//if everything is ok, then transform the
data in values.
		Dim arrVal(11)
		arrVal(0) = CInt(arrSet(0)) '//boxes in X
		arrVal(1) = CInt(arrSet(1)) '//boxes in Y
		arrVal(2) = CInt(arrSet(2)) '//boxes in Z
		arrVal(3) = CInt(arrSet(3)) '//Number of Individuals per
Generation
		arrVal(4) = CInt(arrSet(4)) '//Number of Generations
		arrVal(5) = arrSet(5) '//select the crossOver
		arrVal(6) = CDbl(arrSet(6)) '//Mutation rate
		arrVal(7) = CInt(arrSet(7)) '//Fitness Sequence byte
		arrVal(8) = CInt(arrSet(8)) '//Fitness Length Sequence
		arrVal(9) = CDbl(arrSet(9)) '//Fitness Point Thershold proximity
		arrVal(10) = CDbl(arrSet(10)) '//Maximum Indivuduals in the
Natural Selection
		arrVal(11) = CDbl(arrSet(11)) '//Curvature Decay
	End If

	'////////////////////////////////////////////////////////////////////
////////////////////////////////////////////
	text.WriteLine("// Settings")
	Dim i
	For i = 0 To UBound(arrItems) - 1
		text.WriteLine(arrItems(i) & " -----> " & arrVal(i))
	Next
	text.WriteLine("  ")
	'////////////////////////////////////////////////////////////////////
////////////////////////////////////////////

	mutRate = arrVal(6) '//Mutation rate
	MaxSelection = arrVal(10)
	factordecay = arrVal(11)

	If(arrVal(5) = "SinglePtCrossOver") Then
		selectCO = 1
	ElseIf(arrVal(5) = "UniformCroossOver") Then
		selectCO = 0
	Else
		Call Rhino.MessageBox("The CrossOver Selected do not Correspond,
fatal Error", 16, strTitle)
		Exit Sub
	End If

	Call MainGA(arrVal(0), arrVal(1), arrVal(2), arrVal(3), arrVal(4),
arrVal(7), arrVal(8), arrVal(9))

End Sub

Sub MainGA(l, w, h, famsize, numGenerations, bit, bitSeries, radius)

	Dim pts : pts = Rhino.GetPoints()
	Dim i, j, k
	Dim bestGeneration, lattice, rwheel, newGeneration
	Dim fitToTest
	Dim layer

	Dim generacion()
	ReDim generacion(famsize - 1)

	'//guardamos cada una de las cadenas dentro del array generacion
	For i = 0 To famSize - 1
		generacion(i) = GenrRndChromose(h, w, l)
	Next

	'//Evaluacion inicial
	'//Hacemos la evaluacion de cada una de las cadenas, y luego las
ordenamos de mejor a peor....
```

```
	bestGeneration = Sort*Fitness*Generation(generacion, bit, bitSeries, pts,
radius, h, w, l)
	fitToTest = bestGeneration(0)(0)
	Call Draw(generacion(bestGeneration(0)(1)), h, w, l, 1)

	text.WriteLine("//-- *Fitness* Results --//")
	text.WriteLine("")
	text.WriteLine("_Best *Fitness* in the Random Generation: " & fitToTest)

	'////////////////////////////////////////////////////////////////////////
//////////////////////
	'////////////  genetic algorithm loop
/////////////////////////////////////////////////////
	'////////////////////////////////////////////////////////////////////////
//////////////////////

	For j = 0 To numGenerations

		'//hacemos la seleccion natural, los mejores tendran mas
posibilidades de ser elegidos....
		rwheel = NaturalSelection(bestGeneration)

		'//constriumos una nueva generacion con las cadenas seleccionadas
y los hijos mutados....
		newGeneration = Replace(rwheel, generacion)

		'//Hacemos la evaluacion de cada una de las cadenas, y luego las
ordenamos de mejor a peor....
		bestGeneration = Sort*Fitness*Generation(newGeneration, bit,
bitSeries, pts, radius, h, w, l)

		'//Aca construimos el mejor de los *individuos* y lo dejamos
permanente hasta que aparezca uno mejor....
		If (bestGeneration(0)(0) > fitToTest) Then
			Call Draw(generacion(bestGeneration(0)(1)), h, w, l, 1)
			fitToTest = bestGeneration(0)(0)
			text.WriteLine("_Best *Fitness* in the (" & j &") Generation:
" & fitToTest)
		End If

		For k = 0 To UBound(generacion)
			generacion(k) = newGeneration(k) '//ahora la nueva
generacion es copiada en generacion para hacer el remplazo
		Next

	Next

	'cerramos el archivo de imepresion...
	text.Close

	Call Rhino.MessageBox("optimization complete", 64, strTitle)

End Sub

Function GenrRndChromose(h, w, l)

	Dim i, j, k, count : count = 0
	Dim bitstr()

	For i = 0 To h - 1
		For j = 0 To w - 1
			For k = 0 To l - 1

				ReDim Preserve bitStr(count)

				If (Rnd > 0.5) Then
					'//creamos un array de 1 y 0, donde 1
representa un box y 0 representa vacío
```

```
                                bitStr(count) = 1
                        Else
                                bitStr(count) = 0
                        End If
                        count = count + 1
                Next
        Next
    Next

    GenrRndChromose = bitStr

End Function

Function Draw(ByVal bitStr, h, w, l, sizebox)

    'limpiamos el espacio de trabajo.
    If IsArray(Rhino.AllObjects(False, True)) Then Call
Rhino.DeleteObjects(Rhino.AllObjects)

    '////////////////////////////////////////Construimos la matriz de
cuadrados
    Call Rhino.EnableRedraw(False)
    Dim i, j, k
    Dim count : count = 0
    Dim cen   : cen = sizebox * 0.5
    Dim box   :  box = Rhino.AddBox(Array(Array(0,0,0),Array(sizebox, 0,
0),Array(sizebox, sizebox, 0), Array(0, sizebox, 0),Array(0,0,sizebox),
Array(sizebox, 0, sizebox), Array(sizebox, sizebox, sizebox), Array(0,
sizebox, sizebox)))

    Dim lattice()

    For i = 0 To h - 1
        For j = 0 To w - 1
            For k = 0 To l - 1

                ReDim Preserve lattice(count)'//aumenta el tamaño
del array
                If (bitStr(count)  = 1) Then  '//si encuentra un 1
en la cadena, copia un box...
                    lattice(count) = Rhino.CopyObject(box,
Array(cen, cen, cen), Array(k + cen, j + cen, i + cen))
                Else
                    lattice(count) = 0 '//guarda un cero...
                End If
                count = count + 1
            Next
        Next
    Next
    Call Rhino.DeleteObject(box)
    Call Rhino.EnableRedraw(True)

    Draw = lattice
    Rhino.Command " -ClearUndo "
End Function

Function PtsGenr(ByVal bitStr, h, w, l, sizebox)

    Dim i, j, k
    Dim count : count = 0
    Dim cen   : cen = sizebox * 0.5

    Dim lattice()

    For i = 0 To h - 1
        For j = 0 To w - 1
            For k = 0 To l - 1
```

```
                        ReDim Preserve lattice(count)'//aumenta el tamaño
del array
                        If (bitStr(count)  = 1) Then  '//nao guarda un punto
                              lattice(count) = Array(k + cen, j + cen, i +
cen)
                        Else
                              lattice(count) = 0 '//guarda un cero...
                        End If
                        count = count + 1
                  Next
            Next
      Next
      PtsGenr = lattice

End Function

'////////////////////////////////////////////////////////////////////////////
/////////////////
'////////////  genetic operators
//////////////////////////////////////////////////////////
'////////////////////////////////////////////////////////////////////////////
/////////////////

Function mutation(ByVal bitStr)

      Dim cutStore()
      Dim i, cut, t, j, c
      Dim hold : hold = CInt(Ubound(bitStr) * mutRate)
      If hold <= 0 Then hold = 1

      For i = 0 To hold - 1
            ReDim Preserve cutStore(i)

            Do
                  c = 0

                  'generamos un puntero dentro del array para cortar en la
cadena...
                  Randomize
                  cut = CInt(LBound(bitStr) + Rnd() * (UBound(bitStr) -
LBound(bitStr)))

                  'comprobamos si el puntero lo hemos repetido
anteriormente...
                  For j = 0 To UBound(cutStore)
                        If cut = cutStore(j) Then
                              c = c + 1
                        End If
                  Next

                  'si no lo hemos repetido, entonces lo guardamos en nuestro
almacen y salimos del do loop...
                  If c = 0 Then
                        cutStore(i) = cut
                        Exit Do
                  End If
            Loop

            Select Case bitStr(cut) 'remplazamos por 1 o 0
                  Case 0
                        bitStr(cut) = 1
                  Case 1
                        bitStr(cut) = 0
            End Select
      Next

      mutation = Array(bitStr, cutStore) 'sale fuera de la función, la cadena
mutada y los puntos donde se hizo el cruzamiento...
```

```
End Function

Function UniformCrossOver(ByVal bitStrDad, ByVal bitStrMom)

	Dim i
	Dim Off1(), Off2()
	ReDim Off1(UBound(bitStrMom))
	ReDim Off2(UBound(bitStrDad))

	For i = 0 To UBound(bitStrDad)

		Randomize
		If(Rnd() < 0.5) Then
			Off1(i) = bitStrDad(i)
			Off2(i) = bitStrMom(i)
		Else
			Off1(i) = bitStrMom(i)
			Off2(i) = bitStrDad(i)
		End If

	Next

	UniformCrossOver = Array(Off1, Off2) '//retornamos a los
descendientes...

End Function

Function SinglePtCrossOver(ByVal bitStrDad, ByVal bitStrMom)

	Randomize
	Dim crossPoint : crossPoint = CInt(LBound(bitStrDad) + Rnd() *
(UBound(bitStrDad) - LBound(bitStrDad)))
	Dim i, j

	Dim Off1(), Off2()
	ReDim Off1(UBound(bitStrMom))
	ReDim Off2(UBound(bitStrDad))

	For i = 0 To crossPoint
		Off1(i) = bitStrDad(i)
		Off2(i) = bitStrMom(i)
	Next

	For j = crossPoint + 1 To UBound(bitStrMom)
		Off1(j) = bitStrMom(j)
		Off2(j) = bitStrDad(j)
	Next

	SinglePtCrossOver = Array(Off1, Off2) '//retornamos a los
descendientes...

End Function

Function NaturalSelection(ByVal generation)
	'//aca hacemos la seleccion natural, pasamos una generacion y retornamos
la seleccion natural para hacer el cruce y la mutaciones.
	'//generation tiene dos elementos el primero el fitness y el segundo la
ubicacion dentro de la poblacion...
	Dim i, j, index, counter : counter = 0
	Dim TempNaturalSelection()
	ReDim TempNaturalSelection(UBound(generation))

	For i = 0 To UBound(generation)

		index = CInt(MaxSelection/((i * i * factordecay) + 1)) - 1
'ENTERO(100/((B1*B1*0,01)+ 1) )
		If(index >= 0) Then
```

```
                    For j = 0 To index
                          ReDim Preserve TempNaturalSelection(counter)
                          TempNaturalSelection(counter) = generation(i)
                          counter = counter + 1
                    Next
             End If
      Next

      NaturalSelection = TempNaturalSelection

End Function

Function Replace(ByVal rwheel, ByVal Genr)

      Dim i, indexDad, indexMom
      Dim offSpring, offSpring2
      Dim son1Mut, son2Mut

      Dim down : down = LBound(rwheel)
      Dim up : up = UBound(rwheel)

      Dim nextGeneration()
      ReDim nextGeneration(UBound(Genr))

      For i = 0 To UBound(Genr) Step 4

            '//buscamos a los padres dentro de la seleccion natural...
            Randomize
            indexDad = CInt(down + Rnd() * (up - down))
            indexMom = CInt(down + Rnd() * (up - down))

            '//crossover          Father-------------------   Mother---------
-----------
            If(selectCO = 1) Then
                  offSpring2 = SinglePtCrossOver(Genr(rwheel(indexDad)(1)),
Genr(rwheel(indexMom)(1)))
            Else
                  offSpring = UniformCrossOver(Genr(rwheel(indexDad)(1)),
Genr(rwheel(indexMom)(1)))
            End If

            '//mutation
            son1Mut = mutation(offSpring2(0))
            son2Mut = mutation(offSpring2(1))

            '//remplazo ocurre aqui con la seleccion  y la mutacion de los
hijos...
            nextGeneration(i) = Genr(rwheel(indexDad)(1))
            nextGeneration(i + 1) = Genr(rwheel(indexMom)(1))
            nextGeneration(i + 2) = son1Mut(0)
            nextGeneration(i + 3) = son2Mut(0)

      Next

      Replace = nextGeneration

End Function

'////////////////////////////////////////////////////////////////////////////
/////////////////
'////////////  Evaluaciones de la Generacion
////////////////////////////////////////////////
'////////////////////////////////////////////////////////////////////////////
/////////////////

Function Sort*Fitness*Generation(ByVal popBitStr, sequenceBit, Series,
ptsRepellor, RadiusProximity, h, w, l)
```

```
	Dim best : best = FitnessEvaluation(popBitStr(0), sequenceBit, Series,
ptsRepellor, RadiusProximity, h, w, l)
	Dim i, c : c = 0
	Dim fitToTest

	Dim fitnessPopulation() '//array para almacenar el fitness i el indice
del individuo
	ReDim fitnessPopulation(UBound(PopBitStr))

	'/////buscamos el mejor individuo de la
poblacion//////////////////////////////
	For i = 0 To UBound(PopBitStr)
		If Not isNull(PopBitStr(i)) Then
			fitToTest = FitnessEvaluation(popBitStr(i), sequenceBit,
Series, ptsRepellor, RadiusProximity, h, w, l)
			fitnessPopulation(i) = Array(fitToTest, i)
			If (best < fitToTest) Then '//Check esta parte, parece que
es innecesaria...
				best = fitToTest
				c = i
			End If
		End If

	Next

	'/////ordenamos la población de mejor a
peor//////////////////////////////
	Dim SortedList
	SortedList = myfuncBoubleSort(fitnessPopulation)

	'//retornamos la lista ordenada con el valor del fitness primero y luego
su indice...
	SortFitnessGeneration = SortedList

End Function

Function myfuncBoubleSort(ByVal myarray)

	Dim i, j
	Dim sortArray

	For i = 1 To UBound(myarray)
		For j = 0 To UBound(myarray) - 1

			If(myarray(j)(0) < myarray(j + 1)(0)) Then

				sortArray = myfuncexchange(myarray, j)

			End If
		Next
	Next

	myfuncBoubleSort = sortArray

End Function

Function myfuncexchange(list, index)

	Dim temp : temp = list(index)
	list(index) = list(index + 1)
	list(index + 1) = temp

	myfuncexchange = list

End Function

'//////////////////poblacion, bit, longitud serie, pts a testear, distancia
influencia, ancho y largo.
```

```
Function FitnessEvaluation(ByVal BitStr, sequenceBit, Series, ptsRepellor,
RadiusProximity, h, w, l)

	'//analizar el primer individuo para hacer tabla de comparacion...
	Dim fitSequence : fitSequence = FitnessSecuencia(BitStr, sequenceBit,
Series) '//buscamos secuencias de 1 o 0 dentro de la cadena de bytes...
	Dim LatticePts : LatticePts = PtsGenr(bitStr, h, w, l, 1)
	Dim insidePts : insidePts = FitnessProximityAttractor(LatticePts,
ptsRepellor, RadiusProximity) '//buscamos cubos que esten dentro de una
proximidad definida...
	Dim footPrint : footPrint = FitnessfootPrint(BitStr, w, l) '//...puntea
la cantidad de suelo ocupado
	Dim volume : volume = FitnessVolume(BitStr)'//puntea el volumen total
del edificio

	If(footPrint > 15) Then footPrint = footPrint * 5 '//utiliza más de la
mitad del suelo, penalizado
	'If(Volume > 1000) Or (Volume < 800) Then Volume = Volume * 0.4

	'//Calculo del Fitness Total...
	FitnessEvaluation = (fitSequence + Volume) - (InsidePts * 2 + footPrint)

End Function

'////////////////////////////////////////////////////////////////////////////
/////////////////
'////////////  fitness definitions
////////////////////////////////////////////////////////
'////////////////////////////////////////////////////////////////////////////
/////////////////

Function FitnessVolume(ByVal bitStr)

	Dim i
	Dim count : count = 0

	For i = 0 To UBound(bitStr)
		If(bitStr(i) = 1) Then count = count + 1
	Next

	FitnessVolume = count

End Function

Function FitnessfootPrint(ByVal bitStr, w, l)

	Dim i, j
	Dim count : count = 0
	Dim index : index = 0

	For i = 0 To w - 1
		For j = 0 To l - 1
			If (bitStr(index) = 1) Then count = count + 1
			index = index + 1
		Next
	Next

	FitnessfootPrint = count

End Function

Function FitnessSecuencia(ByVal bitStr, bin, lengthSequence) '//fitness Byte
Strings

	Dim cInternal : cInternal = 0 '//contador interno
	Dim cGeneral : cGeneral = 0 '//contador general
	Dim j
```

```
	Dim i: For i = 0 To UBound(bitStr) step lengthSequence
		'//comienzamos a buscar dentro de la secuencia
		j = i
		Do Until j = i + lengthSequence
			If(bitStr(j) = bin) Then cInternal = cInternal + 1
			j = j + 1 : If(j >= UBound(bitStr)) Then Exit Do  '//si j
es igual o mayor al indice del ultimo elemento, escapar
		Loop

		If(cInternal = lengthSequence) Then cGeneral = cGeneral + 1
		cInternal = 0
	Next

	FitnessSecuencia = cGeneral

End Function

Function FitnessProximityAttractor(ByVal Lattice, ptToTest, thershold)
'//fitness Boxes Lattice

	Dim count :  count = 0
	Dim i, j

	For i = 0 To UBound(Lattice)

		If isArray(lattice(i)) Then '//si es existe un punto en esa
posicion....
			For j = 0 To UBound(ptToTest)
				If(Rhino.Distance(lattice(i), ptToTest(j)) <=
thershold) Then count = count + 1
			Next
		End If
	Next

	FitnessProximityAttractor = count

End Function
```

Código de Organización espacial mediante AG

```
Option Explicit
'Script written by <carlos de la b>
'Script copyrighted by <designemergente.org>
'Script version Thursday, 01 April 2010 09:57:32
'////////////////////////////////////////// Public Variables
//////////////////////////////////////////
Dim objInit, text
Dim selectCO, mutRate, mutTher
Dim strTitle

Call LoadDataInit()
Sub LoadDataInit()

	strTitle = "GENETIC ALGORITHM - Spatial Organization Optimization V.1"

	Dim Filter, FileName
	Filter = "Text File (*.txt) | *.txt"
```

```
	FileName = Rhino.OpenFileName("Select a txt file to place the GA Data",
Filter)
	If IsNull(FileName) Then Exit Sub

	Set objInit = CreateObject("Scripting.FileSystemObject")
	Set text = objInit.CreateTextFile(FileName, True)

	text.Writeline("//      DATA GENETIC ALGORITHM - Space Distribution
Optimization      //")
	text.Writeline(" ")
	text.WriteLine("//programmed by carlos de la barrera, 09 de junio de
2009, cidelab@gmail.com, http:\\www.designemergente.org")
	text.Writeline(" ")
	text.Writeline(" ")
	text.Writeline("//Time Date Experiment: " & CStr(Now))
	text.WriteLine("")
	text.WriteLine("")
	text.WriteLine("//    Fitness Espatial Settings : Original area ->
Objective Area -> Thershold Precission    //")

	'//Exportar los datos Iniciales a Excel para comprobar y definir la
restante informacion
	Dim xlsFileName, xlsFilter
	xlsFilter = "Excel Files (*.xlsx)|*.xlsx|*.xls|"
	xlsFileName = Rhino.OpenFileName("Open The Excel sheet Space
Constraints", xlsFilter)
	If IsNull(xlsFileName) Then Exit Sub

	'//mensaje de bienvenida
	Dim ret, ptCloud, BBox
	ret = Rhino.MessageBox("Select the Points to Start", 1 + 64, "Genetic
Algorithm for SP V.00")

	If ret = 1 Then
		'//input nube de puntos
		ptCloud = Rhino.GetObjects("sel the points", 1) '//select the
points for the cells....
		If IsNull(ptCloud) Then Exit Sub

		BBox = Rhino.GetObject("sel the boundary of the building", 4)
'//select the boundaries of the building...
		If IsNull (Rhino.IsPolyline(BBox)) Then Exit Sub

		'//generamos la primera serie de poligonos
		Dim arrpts: arrpts = funcConverse(ptCloud)
		Dim firstInd : firstInd = Polygon(arrpts, BBox)

	Else
		Call rhino.Print("Execution Canceled, ¡¡¡ save the Ocean !!!")
		Exit Sub

	End If

	'//imprimimos las areas de cada uno de los espacios
	Dim i, j
	Dim arr()
	ReDim arr(UBound(arrpts),1)
	Dim area
	For i = 0 To UBound(arrpts)
		area = Rhino.CurveArea(firstInd(i)(1))(0)
		Call Rhino.AddText(i & ": " & area, firstInd(i)(0), 1)
		arr(i, 0) = i
		arr(i, 1) = area
	Next

	Dim xlsapp
	Set xlsapp = CreateObject("Excel.Application")
	Call xlsapp.Workbooks.Open(xlsFileName, True, False)
```

```
	xlsapp.Visible = True
	Dim xlswb
	Set xlswb = xlsapp.ActiveWorkbook.Worksheets(1)
	Dim xlsrange
	Set xlsrange = xlswb.Range("A2").Resize(Ubound(arrpts) + 1, 2) '//rango
para almacenar nuestros datos
	xlsrange.value = arr

	Call rhino.MessageBox("waiting User Imput", 32, strTitle)

	'//Collect the Data
	Dim fitnessRetrieve() '//los datos del Objective Goal
	ReDim fitnessRetrieve(Ubound(arrpts))
	For i = 0 To UBound(arrpts)
		fitnessRetrieve(i) = xlswb.Cells(i + 2, 3).Value
	Next

	Dim thersholdRetrieve() '//El grado de precision para cada uno de esos
datos
	ReDim thersholdRetrieve(Ubound(arrpts))
	For i = 0 To UBound(arrpts)
		thersholdRetrieve(i) = xlswb.Cells(i + 2, 4).Value
		text.WriteLine("__area Position :(" & i & ") :" &
Rhino.CurveArea(firstInd(i)(1))(0) & "  __Objective Goal :( " &
fitnessRetrieve(i) &" )  __Thershold Precission ---> : " &
thersholdRetrieve(i))
	Next
	text.WriteLine("")
	text.WriteLine("")
	text.WriteLine("//    Genetic Algorithms Settings    //")

	Dim names : names = Array("_Number of Individuals per Generation
(multiple of 4) : 8 -> 300 ", "_Number of Generations : 10 -> 1000 ", "_Select
Random Thershold in the Initial Population : 0 -> 1 ", "_Select The CrossOver
Type : UniformCroossOver" & " Or " & "SinglePtCrossOver ", "_Mutation rate : 0
-> 1 ","_Mutation thershold Point in Generation : 0 -> 1 " ,"_Maximum
Indivuduals In the Natural Selection : 10 -> 100 ", "_Curvature Decay : 0 -> 1
")

	Dim GaSettings(7) '//los settings
	For i = 0 To 7
		GaSettings(i) = xlswb.Cells(i + 2, 15).Value
		text.WriteLine(names(i) & "--> " & GaSettings(i))
	Next

	If(GaSettings(3) = "SinglePtCrossOver") Then
		selectCO = 1
	ElseIf(GaSettings(3) = "UniformCroossOver") Then
		selectCO = 0
	Else
		Call Rhino.MessageBox("The CrossOver Selected do not Correspond,
fatal Error", 16, strTitle)
		Exit Sub
	End If

	mutRate = GaSettings(4)
	mutTher = GaSettings(5)

	'//pasamos los puntos, bbox, las areas a alcanzar, la precision y los
settings del AG
	Call manageGA(firstInd, BBox, fitnessRetrieve, ThersholdRetrieve,
GaSettings)

End Sub

Sub ManageGA(firstInd, BBox, fitnessGoal, precission, GASettings)

	Dim i
```

```
	'//generamos la primera población aleatoria de individuos.
	Dim initPopulation()
	ReDim initPopulation(GaSettings(0) - 1)
	For i = 0 To GaSettings(0) - 1 '//numero de individuos por generacion
		initPopulation(i) = RandomPopulation(firstInd, GASettings(2))
	Next

	'//generamos y evaluamos cada uno de los individuos de la población.
	Dim BestGeneration
	BestGeneration = funcEvaluationAndSortedList(initPopulation, BBox,
fitnessGoal, precission, GaSettings)

	Dim theBest
	theBest = bestGeneration(0)


	'//delete the extra geometry
	Call Rhino.EnableRedraw(False)
	For i = 1 To UBound(bestGeneration)
		Call Delete(bestGeneration(i)(0))
	Next
	Call Delete(firstInd)
	Call Rhino.EnableRedraw(True)

	text.WriteLine("")
	text.WriteLine("")
	text.WriteLine("//-- Fitness Results --//")
	text.WriteLine("")
	text.WriteLine("_Best Fitness in the Random Generation: " & theBest(1))


	'////////////////////////////////////////////////////////////////////////
////////////////////
	'////////////  genetic algorithm loop
//////////////////////////////////////////////////
	'////////////////////////////////////////////////////////////////////////
////////////////////
	Dim j : j = 0
	Dim Rwheel
	Dim newGeneration

	Do
		'//hacemos la seleccion natural, los mejores tendran mas
posibilidades de ser elegidos....
		Rwheel = TournamentSelection(bestGeneration, GaSettings(6),
GaSettings(7))

		'//constriumos una nueva generacion con las cadenas seleccionadas
y los hijos mutados....
		newGeneration = Replace(rwheel, GaSettings(0), thebest)

		'//generamos los nuevos espacios
		bestGeneration = funcEvaluationAndSortedList(newGeneration, BBox,
fitnessGoal, precission, GaSettings)

		'//si el nuevo fitness es mejor eliminar el antiguo y conservar
el nuevo
		If(bestGeneration(0)(1) > theBest(1)) Then

			Call Delete(theBest(0)) 'Eliminamos las curvas del mejor
anterior

			theBest = bestGeneration(0) '//guardamos el mejor

			Call Rhino.EnableRedraw(False)
			For i = 1 To UBound(bestGeneration) '//eliminamos los
individuos que no fueron buenos
```

```
                        Call Delete(bestGeneration(i)(0))
                Next
                Call Rhino.EnableRedraw(True)

        Else
                Call Rhino.EnableRedraw(False)
                For i = 0 To UBound(bestGeneration)
                        Call Delete(bestGeneration(i)(0))
                Next
                Call Rhino.EnableRedraw(True)
        End If

        '//print the results...
        text.WriteLine("_Best Fitness in the (" & j & ") Generation : ->
" & theBest(1))
        Call Rhino.Print("_Best Fitness in the (" & j & ") Generation : -
> " & theBest(1))

        If(theBest(1) = UBound(bestGeneration(0)(0)) + 1) Then
                Call Rhino.MessageBox("fitness reached !!!, yihaaa!!!", 16,
strTitle)
                Exit Do
        ElseIf (j = GaSettings(1) - 1) Then
                Call Rhino.MessageBox("Maximum number of iteration
reached", 16, strTitle)
                Exit Do
        End If

        j = j + 1

    Loop

    text.Close

End Sub

'////////////////////////////////////////////////////////////////////////////
/////////////////
'////////////  genetic operators
//////////////////////////////////////////////////////////
'////////////////////////////////////////////////////////////////////////////
/////////////////

Function mutation(ByVal bitStr, ByVal areas)

    Dim cutStore()
    Dim i, cut, t, j, c

    Dim hold : hold = CInt(Ubound(bitStr) * mutRate)
    If hold <= 0 Then hold = 1

    Dim x, y
    Dim paramClose, thershold

    Randomize

    For i = 0 To hold - 1
        ReDim Preserve cutStore(i)

        Do
            c = 0

            'generamos un puntero dentro del array para cortar en la
cadena...
            Randomize
            cut = CInt(LBound(bitStr) + Rnd() * (UBound(bitStr) -
LBound(bitStr)))
```

```
                    For j = 0 To UBound(cutStore) 'comprobamos si el puntero lo
hemos repetido anteriormente...
                        If (cutStore(j) = cut) Then
                            c = c + 1
                        End If
                    Next

                    If c = 0 Then 'si no lo hemos repetido, lo guardamos y
salimos del do loop...
                        cutStore(i) = cut
                        Exit Do
                    End If
            Loop

            paramClose = Rhino.CurveClosestPoint(areas(0)(cut)(1),
bitStr(cut))
            thershold = Rhino.Distance(bitStr(cut),
Rhino.EvaluateCurve(areas(0)(cut)(1), paramClose))

            thershold = thershold * mutTher

            x = (bitStr(cut)(0) - thershold) + Rnd() * ((bitStr(cut)(0) +
thershold) - (bitStr(cut)(0) - thershold))
            y = (bitStr(cut)(1) - thershold) + Rnd() * ((bitStr(cut)(1) +
thershold) - (bitStr(cut)(1) - thershold))

            bitStr(cut) = Array(x, y, 0)

      Next

      mutation = bitStr

End Function

Function UniformCrossOver(ByVal bitStrDad, ByVal bitStrMom)

      Dim i
      Dim Off1(), Off2()
      ReDim Off1(UBound(bitStrMom))
      ReDim Off2(UBound(bitStrDad))

      For i = 0 To UBound(bitStrDad)

            Randomize
            If(Rnd() < 0.5) Then
                  Off1(i) = bitStrDad(i)
                  Off2(i) = bitStrMom(i)
            Else
                  Off1(i) = bitStrMom(i)
                  Off2(i) = bitStrDad(i)
            End If

      Next

      UniformCrossOver = Array(Off1, Off2) '//retornamos a los
descendientes...

End Function

Function SinglePtCrossOver(ByVal bitStrDad, ByVal bitStrMom)

      Randomize
      Dim crossPoint : crossPoint = CInt(LBound(bitStrDad) + Rnd() *
(UBound(bitStrDad) - LBound(bitStrDad)))
      Dim i, j

      Dim Off1(), Off2()
      ReDim Off1(UBound(bitStrMom))
```

```
	ReDim Off2(UBound(bitStrDad))

	For i = 0 To crossPoint
		Off1(i) = bitStrDad(i)
		Off2(i) = bitStrMom(i)
	Next

	For j = crossPoint + 1 To UBound(bitStrMom)
		Off1(j) = bitStrMom(j)
		Off2(j) = bitStrDad(j)
	Next

	SinglePtCrossOver = Array(Off1, Off2) '//retornamos a los
descendientes...

End Function

Function Replace(ByVal rwheel, sizeGeneration, ByVal areas)

	Dim cleanRW : cleanRW = cleanRwheel(rwheel)

	Dim i, indexDad, indexMom
	Dim offSpring
	Dim son1Mut, son2Mut

	Dim down : down = LBound(rwheel)
	Dim up : up = UBound(rwheel)

	Dim nextGeneration()
	ReDim nextGeneration(sizeGeneration - 1)

	For i = 0 To sizeGeneration - 1 Step 4

		'//buscamos a los padres dentro de la seleccion natural...
		Randomize
		indexDad = CInt(down + Rnd() * (up - down))
		indexMom = CInt(down + Rnd() * (up - down))

		'//crossover		Father-------------------   Mother---------
-----------
		If(selectCO = 1) Then
			offSpring = SinglePtCrossOver(cleanRW(indexDad),
cleanRW(indexMom))
		Else
			offSpring = UniformCrossOver(cleanRW(indexDad),
cleanRW(indexMom))
		End If

		'//mutation
		son1Mut = mutation(offSpring(0), areas)
		son2Mut = mutation(offSpring(1), areas)

		'//remplazo ocurre aqui con la seleccion  y la mutacion de los
hijos...
		nextGeneration(i) = cleanRW(indexDad)
		nextGeneration(i + 1) = cleanRW(indexMom)
		nextGeneration(i + 2) = son1Mut
		nextGeneration(i + 3) = son2Mut

	Next

	Replace = nextGeneration

End Function

Function TournamentSelection(ByVal generation, maxTournament, preasure)

	Dim i, j, k, p
```

```
	Dim activate  : activate  = 0
	Dim gladiator

	'arrays para guardar los punteros de selección...
	Dim arrpointers()
	'//arrays para la seleccion de individuos.
	Dim arrplayers()
	ReDim arrplayers(preasure - 1)
	'arrays para guardar los jugadores que entran al torneo
	Dim tournamentplayers()
	ReDim tournamentplayers(preasure - 1)
	'arrays para guardar la seleccion
	Dim TempNaturalSelection()
	ReDim TempNaturalSelection(maxTournament - 1)

	Dim min : min = LBound(generation)
	Dim max : max = UBound(generation)

	For i = 0 To maxTournament - 1
		ReDim Preserve arrpointers(i)
		Do
			Randomize
			For k = 0 To preasure - 1 '//buscamos los indices de los
individuos para competir
				arrplayers(k) = CInt(min + Rnd() * (max - min))
			Next

			arrpointers(i) = arrplayers 'guardamos los indices...
			For j = 0 To UBound(arrpointers)
				If(j <> UBound(arrpointers)) Then
					For p = 0 To UBound(arrplayers)
						If (arrpointers(j)(p) =  arrplayers(p))
Then
							activate = activate + 1
						End If
					Next
					If (activate < preasure) Then
						activate = 0
					End If
				End If
			Next
			If (activate < preasure) Then
				activate = 0
				Exit Do
			End If
		Loop

		For k = 0 To preasure - 1 'creamos el array para los
competidores...
			tournamentplayers(k) = generation(arrplayers(k))
		Next

		gladiator = Torneo(tournamentplayers)

		TempNaturalSelection(i) = gladiator
	Next

	Erase arrpointers

	TournamentSelection = TempNaturalSelection

End Function

Function RandomPopulation(ByVal listOfAreas, thersPointDisplace)

	Dim i
	Dim x, y
	Dim ind()
```

```
	ReDim ind(UBound(listOfAreas))
	Dim point, paramClose, thershold

	For i = 0 To UBound(listOfAreas)

		point = listOfAreas(i)(0)

		'buscamos el borde más cercano para saber la movilidad con la que
contamos....
		paramClose = Rhino.CurveClosestPoint(listOfAreas(i)(1),
listOfAreas(i)(0))
		thershold = Rhino.Distance(listOfAreas(i)(0),
Rhino.EvaluateCurve(listOfAreas(i)(1), paramClose))

		thershold = thershold * thersPointDisplace

		Randomize '//calculamos los puntos aleatorios nuevos.
		x = (point(0) - thershold) + Rnd() * ((point(0) + thershold) -
(point(0) - thershold))
		y = (point(1) - thershold) + Rnd() * ((point(1) + thershold) -
(point(1) - thershold))

		ind(i) = Array(x, y, 0)

	Next

	RandomPopulation = ind

End Function

Function funcEvaluationAndSortedList(ByVal InitPop, BBox, fitnessGoal,
precission, ByVal GaSettings)'//calculate the fitness...

	Dim i
	Dim Ind
	Dim fitnessPopulation()
	ReDim fitnessPopulation(UBound(InitPop))
		For i = 0 To UBound(InitPop) '//recorremos toda la población
		Ind = Polygon(InitPop(i), BBox) '//generate the poligon Voronoi
		fitnessPopulation(i) = Array(ind, fitnessEvaluation(Ind,
fitnessGoal, precission))
	Next

	'//sort List....
	Dim SortedList
	SortedList = myfuncBoubleSort(fitnessPopulation)

	funcEvaluationAndSortedList = SortedList

End Function

Function fitnessEvaluation(individual, arrAreaGoal, arrPrecision)

	Dim i
	Dim count : count = 0
	Dim area
	For i = 0 To UBound(individual)
		area = Rhino.CurveArea(individual(i)(1))(0)
		count = count + Abs(area - arrAreaGoal(i))
	Next

	fitnessEvaluation = 5/count

End Function

'////////////////////////////////////////////////////////////////////////////
/////////////////
```

```
'////////////  Utilities
////////////////////////////////////////////////////////////////////
'//////////////////////////////////////////////////////////////////////////
/////////////////

Function Torneo(ByVal arrList)

	Dim test: test = arrList(0)
	Dim i
	For i = 1 To UBound(arrList)
		If (test(1) < arrList(i)(1)) Then
			test = arrList(i)
		End If
	Next

	Torneo = test

End Function

Function myfuncBoubleSort(ByVal myarray)

	Dim i, j
	Dim sortArray

	For i = 1 To UBound(myarray)
		For j = 0 To UBound(myarray) - 1

			If(myarray(j)(1) < myarray(j + 1)(1)) Then

				sortArray = myfuncexchange(myarray, j)

			End If
		Next
	Next

	myfuncBoubleSort = sortArray

End Function

Function myfuncexchange(list, index)

	Dim temp : temp = list(index)
	list(index) = list(index + 1)
	list(index + 1) = temp

	myfuncexchange = list

End Function

Function cleanRwheel(ByVal rwheel)

	Dim cleanRW()
	ReDim cleanRW(UBound(rwheel))

	Dim ptIndividual()
	ReDim ptIndividual(UBound(rwheel(0)(0)))

	Dim i, j
	For i = 0 To UBound(rwheel)
		For j = 0 To UBound(rwheel(0)(0))
			ptIndividual(j) = rwheel(i)(0)(j)(0)
		Next
		cleanRW(i) = ptIndividual
	Next

	cleanRwheel = cleanRW

End Function
```

```
Function VoronoiPolygon(index, datSet, BBox)

	'this is not necessary, is in case the function failed...
	VoronoiPolygon = Null

	Dim midPt
	Dim ChordLength, Border
	Dim brdLines(), i, N
	Dim Ovec, PerpVec, Ovec2, PerpVec2, Sp, Ep
	ReDim brdLines(UBound(datSet)-1)

	Dim pts :  pts = Rhino.PolylineVertices(BBox)
	ChordLength = Rhino.Distance(pts(0), pts(2)) * 0.6
	N = 0

	Call Rhino.EnableRedraw(False)

	'i will run for each point in the datSet, previously called ptCloud...
	For i = 0 To UBound(datSet)

		'If i is not equal to the index, preiviously called also i.
then...
		If i <> index Then

			'calc the mid point...out of the function Voronoi... to get
speed.
			midPt = CalcMidPt(datSet(i), datSet(index))

			Ovec = Rhino.VectorCreate(datSet(i), datSet(index)) 'here
create from the "point" to the other points a vector.
			PerpVec = rhino.VectorCrossProduct(Ovec, Array(0,0,1))
'here we calculate a perpendicular vector from the original vector.

			Ovec2 = Rhino.VectorCreate(datSet(index), datSet(i)) 'the
same to the other direction
			PerpVec2 = rhino.VectorCrossProduct(Ovec2, Array(0,0,1))

			PerpVec = Rhino.VectorUnitize(PerpVec) 'here we unitize the
vectors, now the vectors has length 1
			PerpVec2 = Rhino.VectorUnitize(perpVec2)

			'here we scale the vectors for the distance of the points
multiplicated for the size of the BBox
			PerpVec = Rhino.VectorScale(PerpVec,
Rhino.Distance(datSet(i), datSet(index)) * ChordLength)
			PerpVec2 = Rhino.VectorScale(perpVec2,
Rhino.Distance(datSet(i), datSet(index)) * ChordLength)

			Sp = Rhino.PointAdd(midPt, PerpVec) 'create a point in the
end of the vector.
			Ep = Rhino.PointAdd(midPt, PerpVec2)

			'here we store each line inside of array
			brdLines(N) = Rhino.AddLine(Sp, Ep)
			N = N+1

		End If
	Next
	Call Rhino.EnableRedraw(True)

	Call Rhino.UnselectAllObjects()
	Call Rhino.SelectObjects(brdLines)
	Call Rhino.SelectObject(BBox)

	Rhino.Command "-_CurveBoolean _DeleteInput=Yes _CombineRegions=No " &
Rhino.Pt2Str(Array(datSet(index)(0),datSet(index)(1),datSet(index)(2))) & "
_Enter", vbFalse
```

```
	'//delete extra lines and borders...
	Call Rhino.DeleteObjects(brdLines)
	Rhino.Command " -ClearUndo "
End Function

Function Polygon(ByVal ptCloud, ByVal BBox)

	Dim i
	Dim OrInd() '//original individual

	For i = 0 To UBound(ptCloud)'Here we generate the first individual...and
execute the voronoi function
		Call VoronoiPolygon (i, PtCloud, BBox)
		Dim crv : crv   = Rhino.FirstObject()
		ReDim Preserve OrInd(i)
		OrInd(i) = array(ptCloud(i), crv)
	Next

	Polygon = OrInd

End Function

Function funcConverse(ByVal arrStrPts)'//string to array!!!
	Dim arrPts()
	ReDim arrPts(UBound(arrStrPts))
	Dim i
	For i = 0 To UBound(arrStrPts)
		arrPts(i) = Rhino.PointCoordinates(arrStrPts(i))
	Next
	funcConverse = arrPts
End Function

Function CalcMidPt(a, b)
	CalcMidPt = Array((a(0) + b(0)) / 2, (a(1) + b(1)) / 2, 0)
End Function

Function Delete(arrdata)

	Call Rhino.EnableRedraw(False)
	Dim i : For i = 0 To UBound(arrdata)
		Call Rhino.DeleteObject(arrdata(i)(1))
	Next
	Call Rhino.EnableRedraw(True)
End Function
```

Código de Estrategia Evolutiva para el Diseño de arquitectura Optimizada, (solo el bloque que corresponde a *GCScript)*

```
// Bentley GenerativeComponents Transaction File -- File structure version
1.06. (Please do not delete or change this line.)

environment
{
    GCVersion                = '08.11.08.136';
    MSVersion                = '08.11.07.171';
    MSProject                = 'GC_Default';
```

```
    MSDesignFile              = 'C:\Users\Carlos\Documents\00_(GA thermic
flow)\00_Codes\MicroGA_Box1.dgn';
}

transaction modelBased 'limits'
{
    feature User.Objects.point02 Bentley.GC.Point
    {
        CoordinateSystem          = baseCS;
        XTranslation              = -12.0;
        YTranslation              = 9.0;
        ZTranslation              = 0.0;
        Visible                   = false;
    }
    feature User.Objects.point03 Bentley.GC.Point
    {
        CoordinateSystem          = baseCS;
        XTranslation              = 20.0;
        YTranslation              = -3.0;
        ZTranslation              = 0.0;
        Visible                   = false;
    }
    feature User.Objects.point04 Bentley.GC.Point
    {
        CoordinateSystem          = baseCS;
        XTranslation              = 12.0;
        YTranslation              = 9.0;
        ZTranslation              = 0.0;
        Visible                   = false;
    }
    feature User.Objects.point05 Bentley.GC.Point
    {
        CoordinateSystem          = baseCS;
        XTranslation              = -4.0;
        YTranslation              = -3.0;
        ZTranslation              = 0.0;
        Visible                   = false;
    }
    feature User.Objects.point09 Bentley.GC.Point
    {
        CoordinateSystem          = baseCS;
        XTranslation              = 20.0;
        YTranslation              = 9.0;
        ZTranslation              = 0.0;
        Visible                   = false;
    }
    feature User.Objects.point06 Bentley.GC.Point
    {
        CoordinateSystem          = baseCS;
        XTranslation              = -4.0;
        YTranslation              = 9.0;
        ZTranslation              = 0.0;
        Visible                   = false;
    }
    feature User.Objects.point07 Bentley.GC.Point
    {
        CoordinateSystem          = baseCS;
        XTranslation              = -12.0;
        YTranslation              = -3.0;
        ZTranslation              = 0.0;
        Visible                   = false;
    }
    feature User.Objects.point08 Bentley.GC.Point
    {
        CoordinateSystem          = baseCS;
        XTranslation              = 12.0;
        YTranslation              = -3.0;
        ZTranslation              = 0.0;
```

```
    Visible                  = false;
}
feature User.Objects.polm Bentley.GC.Polygon
{
    Color                    = 6;
    VertexPoints             = {point07,point03,point09,point02};
}
feature User.Objects.polym Bentley.GC.Polygon
{
    Color                    = 182;
    VertexPoints             = {point06, point05, point08, point04};
}
feature User.Objects.point10 Bentley.GC.Point
{
    CoordinateSystem         = baseCS;
    XTranslation             = -4.0;
    YTranslation             = 10.0;
    ZTranslation             = 0;
    Visible                  = false;
}
feature User.Objects.point11 Bentley.GC.Point
{
    CoordinateSystem         = baseCS;
    XTranslation             = -12.0;
    YTranslation             = -4.0;
    ZTranslation             = 0.0;
    Visible                  = false;
}
feature User.Objects.point12 Bentley.GC.Point
{
    CoordinateSystem         = baseCS;
    XTranslation             = -4.0;
    YTranslation             = -4.0;
    ZTranslation             = 0.0;
    Visible                  = false;
}
feature User.Objects.point13 Bentley.GC.Point
{
    CoordinateSystem         = baseCS;
    XTranslation             = -12.0;
    YTranslation             = 10.0;
    ZTranslation             = 0.0;
    Visible                  = false;
}
feature User.Objects.point14 Bentley.GC.Point
{
    CoordinateSystem         = baseCS;
    XTranslation             = 12.0;
    YTranslation             = 10.0;
    ZTranslation             = 0.0;
    Visible                  = false;
}
feature User.Objects.point15 Bentley.GC.Point
{
    CoordinateSystem         = baseCS;
    XTranslation             = 20.0;
    YTranslation             = 10.0;
    ZTranslation             = 0.0;
    Visible                  = false;
}
feature User.Objects.point16 Bentley.GC.Point
{
    CoordinateSystem         = baseCS;
    XTranslation             = 12.0;
    YTranslation             = -4.0;
    ZTranslation             = 0.0;
    Visible                  = false;
}
```

```
    feature User.Objects.point17 Bentley.GC.Point
    {
        CoordinateSystem          = baseCS;
        XTranslation              = 20.0;
        YTranslation              = -4.0;
        ZTranslation              = 0.0;
        Visible                   = false;
    }
}

transaction modelBased 'Box Geometry'
{
    feature User.Objects.point01 Bentley.GC.Point
    {
        CoordinateSystem          = baseCS;
        XTranslation              = 4;
        YTranslation              = 3;
        ZTranslation              = 1.35;
    }
    feature User.Objects.coordYSrfWest Bentley.GC.GraphVariable
    {
        Value                     = 3;
    }
    feature User.Objects.coordXSrfEast Bentley.GC.GraphVariable
    {
        Value                     = 16;
    }
    feature User.Objects.coordXSrfWest Bentley.GC.GraphVariable
    {
        Value                     = -8;
    }
    feature User.Objects.coordYSrfEast Bentley.GC.GraphVariable
    {
        Value                     = 3;
    }
    feature User.Objects.ptWest Bentley.GC.Point
    {
        CoordinateSystem          = baseCS;
        XTranslation              = coordXSrfWest;
        YTranslation              = coordYSrfWest;
        ZTranslation              = 1.35;
    }
    feature User.Objects.ptEast Bentley.GC.Point
    {
        CoordinateSystem          = baseCS;
        XTranslation              = coordXSrfEast;
        YTranslation              = coordYSrfEast;
        ZTranslation              = 1.35;
    }
    feature User.Objects.direction02 Bentley.GC.Direction
    {
        DirectionPoint            = ptWest;
        Origin                    = point01;
    }
    feature User.Objects.direction01 Bentley.GC.Direction
    {
        DirectionPoint            = ptEast;
        Origin                    = point01;
        SymbolXY                  = {608.0, 347.199981689453};
    }
    feature User.Objects.plane02 Bentley.GC.Plane
    {
        Direction                 = direction02;
        DistanceFromOrigin        = Distance(point01, ptWest) * 0.5;
        Origin                    = point01;
        Visible                   = false;
    }
    feature User.Objects.plane01 Bentley.GC.Plane
```

```
    {
        Direction                = direction01;
        DistanceFromOrigin       = Distance(point01, ptEast) * 0.5;
        Origin                   = point01;
        Visible                  = false;
    }
    feature User.Objects.coordinateSystem02 Bentley.GC.CoordinateSystem
    {
        Plane                    = plane02;
        XTranslation             = -5;
        YTranslation             = -2;
        Visible                  = false;
    }
    feature User.Objects.polygon01 Bentley.GC.Polygon
    {
        Function                 = function (Plane plane)
                                   {
                                       Polygon pol = {}; //polygon faces...

                                       for (int i = 0; i < plane.Count; ++i)
                                       {
                                           CoordinateSystem cs = new
CoordinateSystem();
                                           cs.OnPlane(plane[i], 0, 0); //
sistema de coordenadas en el plano.

                                           int shift; //para alternar sobre
los puntos...
                                           int v; //para alternar la
direccion sobre los vectores...

                                           Point pt = {};
                                           for (int j = 0; j < 2; ++j)
//almacenar los puntos base
                                           {
                                               pt[j] = new Point();
                                               if(j % 2 == 0){v = 2;} else {v
= -2;}

pt[j].ByDirectionAndDistanceFromOrigin(cs, cs.YDirection, v);
                                           }

                                           Point Vertices = {}; //definimos
los puntos para generar el polygono.
                                           for (int k = 0; k < 4; ++k)
                                           {
                                               Vertices[k] = new Point();
                                               if((k == 1) || (k == 2))
{shift = 1;} else {shift = 0;}
                                               if((k == 0) || (k == 1)) {v =
-5;} else {v = 5;}

Vertices[k].ByDirectionAndDistanceFromOrigin(pt[shift], cs.XDirection, v);
                                           }

                                           pol[i] = new Polygon();
                                           pol[i].ByVertices(Vertices);
                                       }

                                       return pol;
                                   };
        FunctionArguments        = {{plane01, plane02}};
    }
    feature User.Objects.coordinateSystem01 Bentley.GC.CoordinateSystem
    {
        Plane                    = plane01;
        XTranslation             = -5;
        YTranslation             = -2;
```

```
        Visible                   = false;
    }
    feature User.Objects.polygon04 Bentley.GC.Polygon
    {
        VertexPoints              = {polygon01[0].Vertices[2],
polygon01[0].Vertices[1], polygon01[1].Vertices[2], polygon01[1].Vertices[1]};
    }
    feature User.Objects.polygon05 Bentley.GC.Polygon
    {
        VertexPoints              = {polygon01[1].Vertices[3],
polygon01[0].Vertices[0], polygon01[0].Vertices[3], polygon01[1].Vertices[0]};
    }
    feature User.Objects.polygon02 Bentley.GC.Polygon
    {
        VertexPoints              = {polygon01[0].Vertices[3],
polygon01[0].Vertices[2], polygon01[1].Vertices[1], polygon01[1].Vertices[0]};
    }
    feature User.Objects.polygon03 Bentley.GC.Polygon
    {
        VertexPoints              = {polygon01[1].Vertices[3],
polygon01[1].Vertices[2], polygon01[0].Vertices[1], polygon01[0].Vertices[0]};
    }
    feature User.Objects.polygon06 Bentley.GC.Polygon
    {
        Function                  = function (Polygon pol, CoordinateSystem
cs)
                                    {
                                        Polygon arrpol = {}; //store the
polygons...

                                        for (int i = 0; i < pol.Count; ++i)
                                        {
                                            DVector3d origin =
DVector3d.FromXYZ(cs[i].X, cs[i].Y, cs[i].Z);

                                            //base Points...
                                            Point ptAxisX = new Point();

ptAxisX.ByDirectionAndDistanceFromOrigin(cs[i], cs[i].XDirection, 20);

                                            Point ptAxisY = new Point();

ptAxisY.ByDirectionAndDistanceFromOrigin(cs[i], cs[i].YDirection, 8);

                                            //creamos vectores vectores...
                                            DVector3d vecX =
DVector3d.FromXYZ(ptAxisX.X, ptAxisX.Y, ptAxisX.Z);
                                            vecX.SubtractInPlace(ref origin);
                                            vecX.NormalizeInPlace();
                                            vecX.ScaleInPlace(Random() *
18.90);
                                            vecX.AddInPlace(ref origin);

                                            DVector3d vecY =
DVector3d.FromXYZ(ptAxisY.X, ptAxisY.Y, ptAxisY.Z);
                                            vecY.SubtractInPlace(ref origin);
                                            vecY.NormalizeInPlace();
                                            vecY.ScaleInPlace(Random() *
7.50);
                                            vecY.AddInPlace(ref origin);

                                            //transformamos los puntos en
vectores...
                                            Point ptVecInX = new Point();

ptVecInX.ByCartesianCoordinates(baseCS, vecX.X, vecX.Y, vecX.Z);

                                            Point ptVecInY = new Point();
```

```
ptVecInY.ByCartesianCoordinates(baseCS, vecY.X, vecY.Y, vecY.Z);

                                        //buscamos el punto medio entre
los vectores.
                                        DVector3d strAverage =
DVector3d.FromXYZ((vecX.X + vecY.X) / 2, (vecX.Y + vecY.Y) / 2, (vecX.Z +
vecY.Z) / 2);
                                        Point strPt = new Point();

strPt.ByCartesianCoordinates(baseCS, strAverage.X, strAverage.Y,
strAverage.Z);

                                        //llamamos a nuestra funcion en la
API de C# para obtener el siguiente punto dentro de la lista...
                                        Line lineToTest = new Line();

lineToTest.ByPoints(pol[i].Vertices[2], pol[i].Vertices[3]);
                                        double dist =
distLineToPoint(lineToTest, strPt);
                                        Point Pt2 = new Point();

                                        double minX = dist[0] * 0.2;
                                        double maxX = dist[0];
                                        double mydist = minX + Random() *
(maxX - minX);


Pt2.ByDirectionAndDistanceFromOrigin(strPt, cs[i].XDirection, mydist);

                                        //buscamos la distancia para los
dos puntos faltantes...
                                        Line lineToTest2 = new Line();

lineToTest2.ByPoints(pol[i].Vertices[0], pol[i].Vertices[3]);
                                        double dist2 =
distLineToPoint(lineToTest2, Pt2);

                                        double minY = dist2[0] * 0.1;
                                        double maxY = dist2[0];
                                        double same =  minY + Random() *
(maxY - minY);

                                        // buscamos el tercer punto...
                                        Point Pt3 = new Point();

Pt3.ByDirectionAndDistanceFromOrigin(Pt2, cs[i].YDirection, same);

                                        //y el cuarto punto del polygono
de la ventana.
                                        Point Pt4 = new Point();

Pt4.ByDirectionAndDistanceFromOrigin(strPt, cs[i].YDirection, same);

                                        //polygon---
                                        arrpol[i] = new Polygon();
                                        arrpol[i].ByVertices({Pt4, strPt,
Pt2, Pt3});
                                    }

                                    return arrpol;
                                };
        FunctionArguments         = {{polygon01[0], polygon01[1]},
{coordinateSystem01, coordinateSystem02}};
    }
}

transaction modelBased 'Text and Information'
```

```
{
    feature User.Objects.textStyle01 Bentley.GC.TextStyle
    {
        Height                    = 0.4;
        TextColor                 = RGB(255,0,0);
        Width                     = 0.4;
    }
    feature User.Objects.point18 Bentley.GC.Point
    {
        CoordinateSystem          = baseCS;
        XTranslation              = <free> (-4.60956511155549);
        YTranslation              = <free> (-6.95578238737642);
        ZTranslation              = <free> (0.0);
        Visible                   = false;
    }
    feature User.Objects.text01 Bentley.GC.Text
    {
        Function                  = function (Point pt, Point pt2)
                                    {
                                        Text t = {};
                                        for (int i = 0; i < pt.Count; ++i)
                                        {
                                            t[i] = new Text();
                                            string m =
ToString(Round(pt2[i].X, 2)) + ", " + ToString(Round(pt2[i].Y, 2));

t[i].ByPointOrPlaneOrCoordinateSystem(pt[i], m, textStyle01);
                                        }

                                        return t;
                                    };
        FunctionArguments         =
{{point13,point17,point14,point12,point15,point10,point11,point16},
{point02,point03,point04,point05,point09,point06,point07,point08}};
    }
    feature User.Objects.text02 Bentley.GC.Text
    {
        Placement                 = point18;
        TextStyle                 = textStyle01;
        TextValue                 = "area is : " + polygon04.Area + " mt2";
    }
}

transaction modelBased 'GA Functions'
{
    feature User.Objects.myfuncBuildGeometry Bentley.GC.GraphFunction
    {
        Definition                = function (double Xeast, double Yeast,
double Xwest, double Ywest)
                                    {
                                        //This function Set the new Geometry
with the correct values...
                                        coordXSrfEast = Xeast;
                                        coordYSrfEast = Yeast;

                                        coordXSrfWest = Xwest;
                                        coordYSrfWest = Ywest;

                                        UpdateGraph();
                                    };
        Description               = "Build Geometry";
    }
    feature User.Objects.myfuncRandomPopulation Bentley.GC.GraphFunction
    {
        Definition                = double[] function(string strIdfPath,
string strEplusPath, string strSolutionPath, int size)
                                    {
                                        double[] population = {};
```

```
                                        for (int i = 0; i < size; ++i)
                                        {
                                            myfuncBuildGeometry(Random(12,
20), Random(-3, 9), Random(-12, -4), Random(-3, 9));

                                            //---  ...initialize the GA...  --
-//

                                            //execute the simulation with the
new geometry
                                            runSimulation(strIdfPath,
strEplusPath, {polygon06[0], polygon06[1], polygon02, polygon01[0], polygon03,
polygon01[1], polygon04, polygon05});

                                            sleepFunction(3.4); //sleeep....

                                            double results =
readDataSolution(strSolutionPath);
                                            double avgHVAC = (results[0] +
results[1]) / 2;

                                            //genr the chromosome...
                                            double[] chromosome;
                                            chromosome =
genrChromosome({ptEast, ptWest}, {polygon06[0], polygon06[1]}, avgHVAC);

                                            //Create the population
                                            population[i] = chromosome;

                                            //aca debemos crear la
poblacion...
                                            Print(ToString(results[0]) + "  "
+ ToString(results[1]));
                                        }

                                        return population;
                                    };
        Description               = "Random Population";
    }
    feature User.Objects.myfuncDescribeChromosome Bentley.GC.GraphFunction
    {
        Definition                = double[] function(double[] chromosome)
                                    {
                                        double[] describe = {};
                                        for (int i = 0; i < 6; ++i)
                                        {
                                           if(i < 3)
                                           {
                                                describe[i] = chromosome[i];
                                           }
                                           else
                                           {
                                                describe[i] = chromosome[i +
12];

                                           }
                                        }

                                        return describe;

                                    };
        Description               = "Describe Cromosome";
    }
    feature User.Objects.myfunc*Genr8*Population Bentley.GC.GraphFunction
    {
        Definition                = double[] function(string strIdfPath,
string strEplusPath, string strSolutionPath, double dataPopulation, int size)
                                    {
```

```
                                        double[] newPop = {};

                                        for (int i = 0; i < size; ++i)
                                        {

myfuncBuildGeometry(dataPopulation[i][0], dataPopulation[i][1],
dataPopulation[i][3], dataPopulation[i][4]);

                                            //execute the simulation with the
new geometry
                                            runSimulation(strIdfPath,
strEplusPath, {polygon06[0], polygon06[1], polygon02, polygon01[0], polygon03,
polygon01[1], polygon04, polygon05});

                                            sleepFunction(3.4); //sleeep....

                                            double results =
readDataSolution(strSolutionPath);
                                            double avgHVAC = (results[0] +
results[1]) / 2;

                                            //genr the chromosome...
                                            double[] chromosome;
                                            chromosome =
genrChromosome({ptEast, ptWest}, {polygon06[0], polygon06[1]}, avgHVAC);

                                            //Create the population
                                            newPop[i] = chromosome;
                                        }
                                        return newPop;
                                    };
        Description               = "Generate Population";
    }
    feature User.Objects.myfuncDataToxls Bentley.GC.GraphFunction
    {
        Definition                = function(string strXlsPath, double
chromosome, int column)

                                    {

                                        ExcelRange myxls = new ExcelRange();
//initialize the class

                                        string abc = {"B", "C", "D", "E", "F",
"G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U",
"V", "W", "X", "Y", "Z", "AA", "AB", "AC", "AD", "AE", "AF"};



                                        for (int i = 0; i < chromosome.Count;
++i)

                                        {

                                            myxls.WriteValue(strXlsPath,
"Hoja1", Format("{0}{1}", abc[i], column), ToString(chromosome[i]));

                                        }

                                    };
        Description               = "Export Data to Excel";
    }
}

transaction script 'GA'
{
    //String data connections...
```

```
    string strIdfPath = "C:\\03_GC_EPlus_Interop\\Exercise1B-Solution.idf"; //
direccion del Idf
    string strEplusPath = "C:\\03_GC_EPlus_Interop\\RUNEP_1 - Acceso directo";
// direccion del acceso directo
    string strSolutionPath = "C:\\03_GC_EPlus_Interop\\Exercise1B-
Solution.csv"; // direccion del archivo de resultados
    string strXlsPath = "C:\\03_GC_EPlus_Interop\\Libro1.xlsx"; // direccion
del Idf

    //General Data Settings...
    int sizePopulation = 5;
    double MutationPercent = 0.23;
    int numOfGenerations = 10;

    //Genr8 a Random Population...
    double[] population;
    population = myfuncRandomPopulation(strIdfPath, strEplusPath,
strSolutionPath, sizePopulation);

    //Perform the Elitism into de population
    double pop;
    pop = pfrmElitism(population);

    //Perform the CrossOver between two chromosomes...
    double son; //this son is not a chromosome, is only the information
necesary to Genr8 a new chromosome...
    son = pfrmCrossOver(pop[0], pop[1]);

    //breakpoint;

    myfuncDataToxls(strXlsPath, pop[0], 2);

    //here the Mutation...
    double sonM;
    sonM = pfrmMutation(son, MutationPercent);

    Print("bestfit Random Population : " + pop[0][30].ToString());


////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
////////
    //     -----     Micro Genetic Algorithm     ----     //     -----
Micro Genetic Algorithm     ----     //     -----     Micro Genetic Algorithm
----     //

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
////////

    double prev = pop[0][30];

    for (int i = 0; i < numOfGenerations; ++i)
    {
        double best = myfuncDescribeChromosome(pop[0]);
        double datapopulation = {};

        datapopulation[0] = best;
        datapopulation[1] = sonM;

        for(int j = 2; j < 5; ++j) //creamos la nueva poblacion...
        {
            //                          0      ,        1     ,  2  ,        3
,        4     ,  5
            datapopulation[j] = {Random(12, 20), Random(-3, 9), 1.35, Random(-
12, -4), Random(-3, 9), 1.35};
        }
```

```
        //Genr8 the new population...
        population = myfuncGenr8Population(strIdfPath, strEplusPath,
strSolutionPath, datapopulation, 5);

        //Perform the elitism...
        pop = pfrmElitism(population);

        //Perform the CrossOver between two chromosomes...
        son = pfrmCrossOver(pop[0], pop[1]);

        //comparacion
        double actual = pop[0][30];
        if (prev > actual)
        {
            myfuncDataToxls(strXlsPath, pop[0], 3 + i); //sacamos fuera los
datos...
            prev = actual;
        }

        //here the Mutation...
        sonM = pfrmMutation(son, MutationPercent);

        Print("bestfit in iteration(" + i.ToString() + ") : " +
pop[0][30].ToString());
    }
}
```