Librería de Algoritmos Genéticos para Arquitectura (LAGA)

Genetic Algorithms Architectural Library

Carlos Ignacio de la Barrera Poblete
designemergente.com, Chile - España
cidelab@gmail.com

Abstract

LAGA is an all-purpose library to create and design Genetic Algorithms (GAs). Originally the library was born as a series of GAs as a search and optimize tool for author's PhD thesis. The originals GAs were applied to different architectural scales: Urban design, architectural design, space optimization and energy design optimization. At the beginning the algorithms were written in different platforms: Rhinoscript, C#.NET and GCScript. The author after finish his PhD, he decided to put all the algorithms in a single language. JAVA was selected to create a flexible Genetic Algorithm library to use in different platforms.

Keywords: Algoritmos genéticos; Optimización; Arquitectura; Diseño algorítmico; Diseño.

Introducción

Este *paper* presenta LAGA, que es una librería de operadores genéticos pensada para el diseño e implementación de AGs. El proyecto comenzó como un trabajo de investigación y que ha ido transformándose en una herramienta de ayuda para el diseño. El fin del doctorado es el comienzo de un proyecto para aplicar el trabajo de tesis en casos reales y contribuir a la sociedad de otra manera.

Es necesario nombrar la tesis porque ella aporta las conclusiones que dieron fruto al proyecto que se presenta. La tesis doctoral es sobre como los algoritmos genéticos pueden contribuir a desarrollar arquitectura, no de manera especulativa, sino más bien como una herramienta de optimización y búsqueda de soluciones de diseño.

El desarrollo práctico de casos y en algunos casos aplicaciones a reales problemas sirvió para aprender la técnica de los AGs. como diseñarlos y programarlos. Pero sobre todo, como calibrarlos y poder tener resultados muy buenos para los diferentes problemas.

Por otro lado las conclusiones sirvieron para entender que se había conseguido, que se había aprendido y sobre todo como debería continuarse el proyecto. De esta manera el doctorado es visto como una semilla que recién comienza a germinar.

Este *paper* tiene forma de bucle porque comienza explicando las conclusiones de la tesis. Luego describe la arquitectura y el funcionamiento de la librería LAGA para acabar con ejemplo y conclusiones finales.

"Un algoritmo es una regla aritmética o geométrica que se repite" (Balmond, 2002). Un Algoritmo Genético (AG) es un método de búsqueda basado en la selección natural de Darwin.

Sus bases biológicas dicen así: En una población todos los individuos compiten por comida y agua. Los individuos con más éxito en conseguir estos recursos, tendrán más oportunidad de aparearse y de tener descendencia. Por lo tanto su ADN se extenderá más rápido por la población. La descendencia, sin embargo, no será una copia fiel de sus padres sino que incorporará pequeñas mutaciones. Este simple concepto es la base de cualquier AG.

Un AG funciona con una población de soluciones. Cada una de las soluciones es evaluada respecto al problema que pretende resolver. Las soluciones que mejor respondan a la evaluación serán conservadas para cruzarse entre ellas y tener descendencia, la cual será mutada y servirá para remplazar la antigua población. Repitiendo este proceso muchas veces, la aptitud media de la población irá subiendo cada vez más hasta encontrar soluciones muy buenas al problema.

Búsqueda de Soluciones y AGs

Si bien los algoritmos están definidos matemáticamente los conceptos usualmente son bastante borrosos. De hecho el concepto de búsqueda se superpone en al menos tres significados.

En computación es posible distinguir tres tipos de búsqueda:

a. Búsqueda en datos almacenados. El problema consiste en encontrar información almacenada en la memoria del ordenador. Buscar una dirección en una agenda.
b. Búsqueda de caminos a un objetivo establecido. Consiste encontrar eficientemente el grupo de acciones que permitirá ir desde un punto dado a un objetivo concreto. Secuencia de movimientos para resolver un puzle.

c. Búsqueda de soluciones. La idea consiste en encontrar eficientemente una solución a un problema en un gran espacio de soluciones posibles. Que forma debe tener el edificio para que consuma el menor gasto de energía durante el días más caliente y más frio del año.

Las diferencias entre el problema a) y los problemas b) y c) son bien definidos. En el problema a) la información está explícitamente ordenada mientras que en b) y c) no lo están. Además son tantas las posibilidades que es imposible generar todas las posibles soluciones. Los AGs son un método para buscar una muy buena solución examinando sólo una fracción de las posibles soluciones.

Existen todo tipo de Algoritmos Genéticos: AGs Paralelos, AGs de grano fino, *Distributed Genetic Algorithms (DGA)*, *Hybrid Rank-Based Evolutionary Algorithms*, *Resource-Based Fitness Sharing* (*RFS*), *Voronoi-Based Genetic Algorithms* (*VGA*), etc. Aunque todos ellos están basados en la misma idea, son diferentes porque los problemas que enfrentan son diferentes.

Los AGs son fáciles de programar. La parte difícil es definir matemáticamente el problema y como serán evaluados los individuos. Otra parte compleja de los AGs es como van a ser calibrados. Por ejemplo, que tamaño de población se va a usar, que porcentaje de cruce y mutación tendrá la población, que tipo de selección se va a implementar y que valores tendrán.

Para problemas de combinatoria la selección elitista (conservar los mejores) es muy efectiva. En cambio para un problema multiobjetivo la selección por ruleta escalada es probablemente la mejor solución. En problemas donde los parámetros de la función de evaluación tienen diferentes pesos, se ha constatado que una selección por torneo da muy buenos resultados.

Los AGs no son una cosa nueva, están por todos lados y nos relacionamos con ellos aunque no usemos ordenadores o móviles. Si bien los arquitectos hemos llegado al final de la fila es el momento de comenzar a contribuir a otras disciplinas y proponer nuevos diseños. Los AGs se diseñan, se componen, tienen estructura y se habla de su arquitectura.

Librería de Algoritmos Genéticos para Arquitectura (LAGA)

Laga es una librería todo propósito para diseñar e implementar AGs. Nació con la idea de agrupar todos los operadores genéticos desarrollados durante la tesis doctoral del autor en un solo lenguaje y bajo una misma arquitectura. El lenguaje elegido fue JAVA por ser robusto y capaz de ejecutarse en diferentes plataformas. El objetivo final de la librería será crear un "envoltorio" capaz de utilizar Laga desde diferentes softwares.

El nombre proviene de Librería de Algoritmos Genéticos para Arquitectura, porque se utilizó originalmente para arquitectura. Sin embargo Laga es multiuso y puede aplicarse a todo tipo de problemas.

El proceso de diseño de Laga puede dividirse en 5 pasos:

1. Diseño y desarrollo de los operadores genéticos sobre diferentes plataformas: *Rhinoscript*, *C#*, *GCScript*
2. Re-diseño de los operadores genéticos y agrupación sobre una misma plataforma.
3. Diseño de la estructura de la librería. Como será el flujo de la información entre las diferentes partes de la librería.
4. Diseño del envoltorio para crear una herramienta flexible.
5. Repetir los pasos 2 al 4.

Actualmente el desarrollo se encuentra en el punto 3. La librería se encuentra en constante evolución y su diseño se va modificando para hacer una herramienta más estable y flexible. La actual arquitectura de Laga está compuesta de 8 clases diferentes. Las clases están compuestas por diferentes métodos, que son los que acogen los diferentes operadores genéticos.

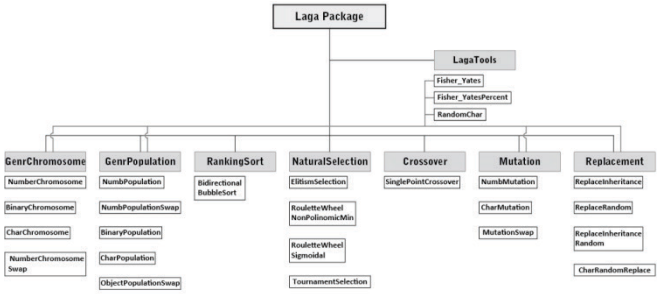

Figura 1: Actual estructura de Laga. Para obtener una imagen detallada refiérase a la web: https://code.google.com/p/laga/

Para utilizar la librería es necesario inicializar y crear las 6 clases básicas que componen un AG. Estas son: *GenrPopulation, RankingSort, NaturalSelection, Crossover, Mutation y Replacement*. Las clases *GenrChromosome* y *LagaTools* no son imprescindibles aunque pueden servir como complemento para el AG que se va diseñar.

La principal desventaja de la actual estructura es que hay que escribir "mucho" código para implementar un AG. Es necesario implementar las 6 clases básicas más todos los parámetros asociados a ellos.

Su principal ventaja es que es muy fácil de usar ya que el flujo de datos de una clase a otras es directo, entonces la información que retorna una clase sirve como parámetro para la siguiente y así sucesivamente. Es una lógica simple y efectiva. Otra ventaja es la posibilidad de combinar los métodos de las clases para diseñar soluciones particulares. Por ejemplo, si el problema exige usar poblaciones pequeñas se puede combinar *Replacement* con *GenrPopulation* para evitar la convergencia prematura. O combinar diferentes tipos de selección en un mismo problema. La actual estructura permite ajustar parámetros durante la ejecución

del AG y esto puede ser muy necesario. Por ejemplo, en algunos problemas, a medida que la población converge la diferencia entre los individuos es cada vez menor, entonces es necesario hacer la selección más rigurosa que en las primeras generaciones.

Laga Arquitectura

Para diseñar un AG son necesarios una serie de ingredientes. En esta apartado se explican en detalle las diferentes partes de la librería.

Codificación: genrChromosome, genrPopulation

El primer paso para un AG es hacer un modelo matemático del problema que se pretende resolver. De esta manera será posible evaluar cada uno de los individuos dentro de la población. Los individuos deben ser codificados por medio de parámetros como una lista de valores. Laga provee diferentes tipos de codificación y diferentes tipos de herramientas para generar cromosomas o poblaciones. La codificación del individuo es clave porque permite el trabajo de los operadores genéticos sobre la población.

La codificación dependerá del problema que se vaya a buscar. Usualmente la codificación es representada por una lista de valores, pero también puede ser una lista de puntos o letras.

Tabla 1: Ejemplos de cromosomas que puede generar la librería. Para obtener más información refiérase a: https://code.google.com/p/laga/wiki/OpCodification

Tipo de cromosoma	Ejemplo
entero	1, 2, 30, 300, 50, 1000, 4, 5, 20
float	3.14, 5.16, 9.82, 45.12, 4.123
binario	1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0
char	a, b, f, d, _, e, u, j, k, l, m, n, o

Las poblaciones en los AGs son listas de cromosomas. En LAGA pueden generarse a partir de cromosomas o bien desde cero. Además incluye métodos para problemas de combinatoria donde un valor (gen) en el cromosoma no puede repetirse. Para una lista completa de métodos para generar cromosomas y poblaciones refiérase a la ayuda de la librería.

Evaluación: Función de aptitud

La función de aptitud es el único operador genético que no está incluida en la librería y es porque está función debe ser definida de acuerdo al problema que se enfrenta. Problemas en diseño hay muchos y de muy variados sabores. Es importante identificar que los algoritmos genéticos si bien son un tipo de algoritmo que pueden ser aplicados a diferentes tipos de problemas, en algunos problemas se comportan muy mal.

Los AGs se comportan bien en problemas multiobjetivo donde los parámetros de la función están relacionados entre ellos y no es posible maximizar un parámetro sin afectar a los otros. Los AGS son buenos en problemas de combinatoria donde el número de posibilidades es tan grande que no es posible evaluar cada una de las diferentes opciones.

RankingSort

El ranking no es un operador genético. Pero es parte fundamental de un AG. Sirve para ordenar la población de soluciones desde el más apto al menos apto según la función de aptitud. Así el operador de selección natural actúa rápidamente sobre la población.

Desde el punto de vista de diseño esta clase podría haberse incluido dentro de la selección natural. Sin embargo tiene su propia clase para dar libertad de acción al momento de hacer el orden de manera ascendente o descendente. La técnica de ordenamiento está basada en el método de la burbuja (bubble sort method) pero la lista es partida por la mitad haciéndola bidireccional y algo más rápida que el método tradicional. El ordenamiento recursivo es un tema a tratar. Tiene la desventaja de consumir mucha memoria pero es más rápido cuando se trabaja con grandes poblaciones > 10000.

NaturalSelection

La selecciona natural sirve para determinar que individuos serán seleccionados para pasar a la siguiente generación. Existen muchos tipos de selección y cada uno de ellos se comporta según el tipo de problema que intenta solucionar. LAGA incorpora los operadores de selección elitista y varias versiones del tipo de selección de Ruleta y Torneo.

La selección elitista conserva los individuos más aptos de cada generación, y es muy útil para problemas de combinatoria. Para problemas multiobjetivo la selección de ruleta y sus variaciones son más adecuados. Cuando el problema involucra pesos en los parámetros de la función, la selección por torneo es una alternativa eficiente porque permite regular de manera dinámica la presión que se ejerce en la selección.

En la mayoría de los casos de selección no es efectivo quedarse siempre con los individuos más aptos de cada generación. Porque se restringe la búsqueda a un espacio reducido y puede que rápidamente se obtenga en un óptimo local. Para evitar este problema, se permite que en la selección se incluyan individuos poco aptos, los cuales al ser cruzados pueden encontrar soluciones prometedoras y dirigir la población hacia el óptimo global.

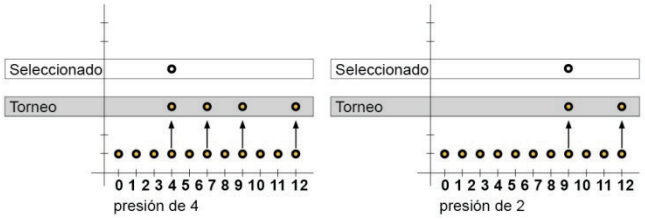

Figura 2: La figura de la izquierda muestra una selección por torneo con una presión de 4 (cuatro individuos compiten en el torneo) y se elige el más apto. En la derecha la presión es de 2. A mayor presión más selectivo es el AG. Para obtener más información refiérase a: https://code.google.com/p/laga/wiki/OpNaturalSelection

Crossover

El cruce, como regla general se aplica aproximadamente al 10% de la población. Aunque también puede ser aplicado al 100%. El cruce tiene como misión intercambiar el material genético entre individuos y da velocidad en la búsqueda del AG. Laga provee, por el momento, solo el cruce de un punto, aunque pronto se incorporarán cruces de dos puntos, cruce uniforme y cruce de combinatoria.

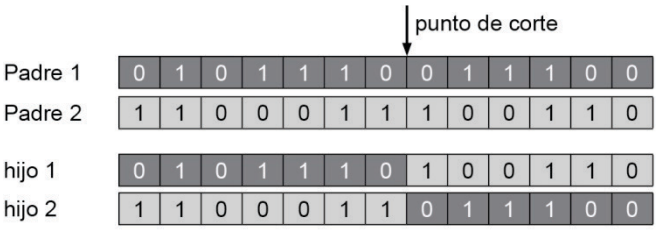

Figura 3: cruce de un punto en Laga. Para obtener más información refiérase a: https://code.google.com/p/laga/wiki/OpCrossover

Mutation

La mutación imita al RNA en la división celular. Este operador es el encargado de buscar en todas las esquinas, valles y picos del espacio de búsqueda.

Un valor alto de mutación hará al individuo buscar en cada lugar del espacio de búsqueda. Sin embargo toda la información heredada de los padres se va a perder haciendo que el AG no sea capaz de converger nunca. Un valor bajo, en cambio, hará converger el AG rápidamente sin resultados satisfactorios.

Lo ideal para un valor de mutación es alrededor de un 1% de la longitud de su cromosoma. La mutación junto a la selección natural determinan el motor de búsqueda del AGs. Y desde este punto de vista tiene más peso frente a otros operadores como el cruce.

Laga provee diferentes tipos de mutación tanto para cromosomas binarios, para números reales y para problemas de combinatoria.

Replacement

Este operador se encarga de remplazar la antigua población por la nueva. Luego de que la población es remplazada se puede decir que el AG ha producido una nueva generación.

Existen varios tipos de remplazo y cada uno se adapta bien dependiendo del tipo de problema. El remplazo más simple es el completo o brutal donde la antigua población es remplazada completamente por la nueva. El remplazo por herencia consiste en solo conservar los hijos. Otros tipos de remplazo son parciales. En el sentido que la nueva población es generada por los mejores individuos de la generación anterior y sus hijos. Para completar, los restantes individuos son generados aleatoriamente. Este último tipo se comporta muy bien donde las poblaciones son pequeñas.

Laga provee los 3 tipos diferentes de remplazo descritos anteriormente. Para obtener más información refiérase a: https://code.google.com/p/laga/wiki/Replacement

LagaTools

La clase *LagaTools* es un grupo de herramientas para ejecutar los diferentes operadores genéticos. Básicamente tiene métodos para desordenar listas y es posible utilizarlos como apoyo en el diseño del AG.

Futuro

La principal deficiencia de LAGA es que no tiene herramientas dedicadas a la interoperabilidad. Dejando al usuario sólo a la hora de enfrentar problemas de optimización estructural o de análisis energético. Tampoco existen herramientas de evaluación, como por ejemplo cálculo de volumen, área, distancia, longitud, etc.

Tareas pendientes de incluir son una clase para vincularse con *Energyplus* y una conexión con algún programa de cálculo estructural. Para el futuro cercano está rediseñar la arquitectura, para evitar tener que escribir tanto código, pero sin sacrificar la facilidad de uso y la modificación de parámetros en tiempo de ejecución.

Una idea a discutir es re-escribir Laga usando programación paralela. Una ventaja de los AGs es que excepto la evaluación el resto de los operadores pueden ejecutarse simultáneamente con diferentes poblaciones. Lo que haría disminuir el consumo de tiempo y afrontar problemas que con un AG secuencial (LAGA) no sería posible. Esto implica a trabajar con varias poblaciones a la vez. Por un lado se mejora el método de búsqueda al estar disperso, pero existe el problema de sincronización entre poblaciones. Entonces surge la duda de cambiar el lenguaje por uno de más bajo nivel, y esto trae otras preguntas como sacrificar flexibilidad por prestación.

Otra idea a plantear consiste en trasladar la librería a un servidor y proveer los servicios de optimización desde internet. De esta manera el usuario no tiene que descargarse la librería, sino un link que vincule su problema con el motor de cálculo. La ventaja de esto es que la optimización podría ocurrir en móviles, GPS, tablets y cualquier aparato conectado a internet.

Laga es open source y de libre acceso para que muchas personas puedan mejorarla desde el punto de vista de programación, interfaz, haciendo pruebas, etc.

Minimizar distancias con LAGA

El problema consiste en encontrar el punto A tal que la suma de todas las distancias entre el punto A y el conjunto de puntos alrededor sea el menor posible.

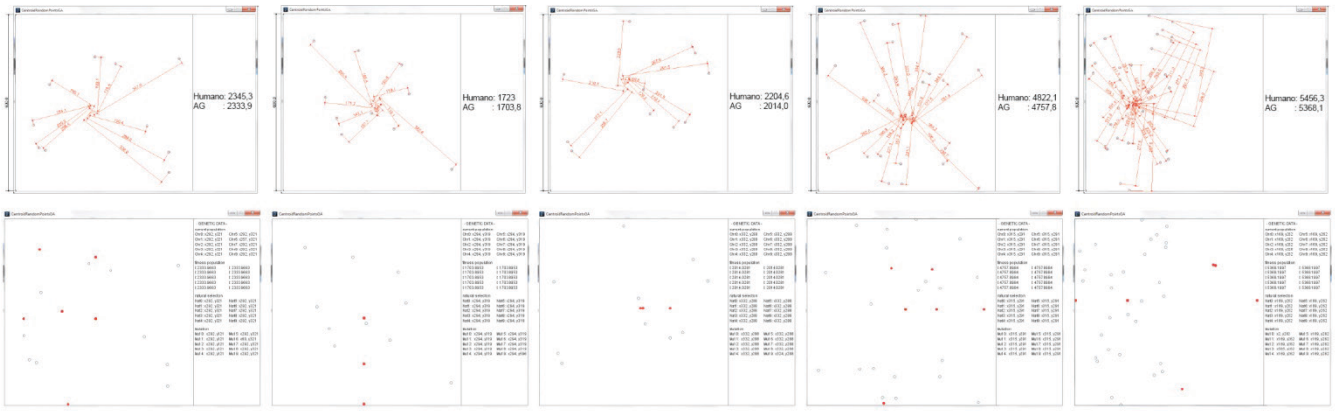

Figure 4: La fila superior muestra una secuencia de pruebas donde el punto A fue definido por un diseñador humano. La secuencia inferior muestra los resultados del AG. Las primeras 3 pruebas tiene un conjunto de 10 puntos. La cuarta prueba tiene 20 y la quinta 30 puntos. De todos los casos el AG siempre propuso puntos con distancias menores a las del diseñador humano. Para obtener más información refiérase a: https://code.google.com/p/laga/wiki/Index?tm=6

Este es un simple ejemplo de un AG para encontrar la menor distancia entre un punto a definir (punto A) y un conjunto de puntos aleatorios en el espacio. El tipo de problema es multiobjetivo porque mientras más cerca se está de un grupo de puntos en el conjunto más lejos se puede estar de otros. Entonces es necesario encontrar el punto de equilibrio que tenga la menor distancia entre todos los puntos del conjunto.

De las 5 pruebas que se hicieron, el AG arrojó siempre distancias menores a las del diseñador humano. La menor diferencia fue en la primera prueba con 10 puntos y la diferencia fue de 11,4 Uds. La mayor diferencia fue en la prueba 3 con 190,6 Uds. Aunque el muestra es muy pequeña en los experimentos 1 y 2 la diferencia se encuentra entre las 10 y 20 Uds. En cambio en los experimentos 4 y 5, con 20 y 30 puntos las diferencias van de las 64 a 88 Uds. de diferencia. Claramente la complejidad del problema aumenta y la brecha es más notoria.

El diseñador humano resolvió el problema por medio de su experiencia e intuición. Si el problema fuese simétrico y lo que se busca es encontrar el punto A en un conjunto de puntos equidistantes a un centro. El resultado no será el centro del círculo, sino que un punto cercano a este.

Este problema puede tener aplicación en diseño de evacuaciones para encontrar el punto más cercano a múltiples entradas. En Urbanismo, logística, telecomunicaciones, etc.

Diseño del AG

El AG tiene un tamaño de población de 50 individuos. El tipo de selección es por ruleta escalada y el tipo de cruce es de un punto. El porcentaje de mutación es de un 1%. El remplazo usado es herencia y aleatoriedad. Esto quiere decir que se conservan los hijos de las soluciones más aptas de la generación anterior y para completar la población se generan individuos aleatorios en busca de nuevos soluciones. La estrategia es perfectamente visible en la fila de los resultados del AG. Un pequeño grupo de la población se encuentra buscando en un espacio diminuto de soluciones mientras que otros individuos exploran otras áreas.

La codificación de los individuos es bastante obvia porque son las coordenadas XY de cada punto.

La función de aptitud es la distancia acumulada entre el punto A y los puntos del conjunto.

References

Balmond, C. (2002). Informal. London, UK: Prestel London.

Koza, J. (1996). Genetic Programming: On the Programming of Computers by Means of Natural Selection. Cambridge: MIT Press..

Mitchell, M. (1998). An Introduction to Genetic Algorithms, Cambridge, Massachusetts: A Bradford Book The MIT Press.

Coello, C., Toscano, G. (2001). A Micro Genetic Algorihm for Multiobjetive Optimizacion: First Internacional Conference on Evolutionary MultiCriterion Optimization. Zurich: Springer Verlag.

Goldberg, D. (1989). Genetic Algorithms in Search, Optimization, and Machine Learning. New York: Addison-Wesley.

Holland, J. (1975). Adaptation in Natural and Artificial Systems, (An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence). Cambridge, London: MIT Press.

Librería de Algoritmos Genéticos para Arquitectura. Retrieved from https://code.google.com/p/laga/