Optimización de resultados producidos por una simulación de la dispersión del virus del Zika en un tejido celular usando algoritmos genéticos con clasificación estocástica

Jose Arias Carazo, Eduardo Biazzetti Sibaja y Javier Fernández Aguilar
CI-1441 Paradigmas computacionales
Escuela de Ciencias de la Computación e Informática
Universidad de Costa Rica
jjavi125@gmail.com, ebiazzetti17@gmail.com, javialofer@hotmail.com

Diciembre de 2018

Resumen

En este artículo se describe una propuesta de ajuste de los datos de una simulación de la dispersión del virus del Zika, usando como método de optimización algoritmos genéticos con clasificación estocástica.

El modelo de simulación de dispersión del virus en el proceso de infección de un tejido celular fue creado para asistir a los investigadores del Centro de Investigaciones en Enfermedades Tropicales (CIET), perteneciente a la escuela de Microbiología y es un modelo que se trata de apegar a la realidad, ya que se ha demostrado en semestres anteriores que es bastante ajustado a las observaciones realizadas en los experimentos de laboratorio. El problema que tiene es que los resultados generados por la simulación difieren con los resultados de laboratorio por un margen muy pequeño pero lo suficientemente significativo para considerarlo no tan preciso. Se pretende entonces solucionar este problema modificando el código para lograr ajustar los datos producidos en la simulación y los datos obtenidos de los experimentos en el laboratorio.

Palabras clave: Simulación multiagente, Virus del Zika, Dispersión viral, Optimización, Algoritmos genéticos.

1. Introducción

El proyecto de la dispersión del virus del Zika es un proyecto de simulación multiagente que se ha estado trabajando con la Facultad de Microbiología y el Centro de Investigación de Enfermedades Tropicales (CIET). En el 2017, se desarrollaron dos versiones de éste. La primera de ellas estaba dirigida a construir una simulación de la dispersión del virus del Zika en una placa de Petri en el laboratorio (Corrales y Leal, 2017). La segunda versión, se trató de una mejora de la versión original (Alvarado, Gómez y Arroyo, 2017). Posteriormente, en el primer semestre de 2018, se realizó otro proyecto con el objetivo de ajustar los datos de la simulación con las observaciones del laboratorio, usando algoritmos genéticos (Muñoz y Porras, 2018).

2. Objetivos

2.1. Objetivo general

Optimizar los resultados de la simulación del virus del Zika usando algoritmos genéticos con clasificación estocástica para que las diferencias entre ambos conjuntos de datos proporcionados por dicha simulación y los resultados obtenidos de pruebas del laboratorio, sean mínimas.

2.2. Objetivos específicos

- Revisar el código de Jeifer Porras y Anthony Muñoz en búsqueda de errores y *bugs* que podrían estar presentes.
- Implementar el método de optimización de algoritmos genéticos con clasificación estocástica a la simulación.
- Analizar los resultados obtenidos y verificar si los resultados tienen mayor o menor diferencia con respecto a los resultados obtenidos de pruebas del laboratorio.
- Comparar nuestros resultados obtenidos con los resultados que obtuvieron Jeifer Porras y Anthony Muñoz para determinar si existen diferencias significativas entre los algoritmos

genéticos y los algoritmos genéticos con clasificación estocástica.

3. Cronograma

Jueves 20 de Septiembre del 2018: Se asiste a la reunión con el Centro de Investigación de Enfermedades Tropicales (CIET).

Lunes 24 de Septiembre del 2018: Se entrega la propuesta definitiva.

Octubre: se le dedicará tiempo al estudio del programa, el lenguaje a trabajar y al estado del arte. Se empezará a trabajar en el programa en las últimas semanas del mes.

Jueves 20 de Noviembre del 2018: Se recibe codigo para conectar Netlogo con Java de Irvin Umaña

Lunes 26 de Noviembre del 2018: Se entrega la propuesta de solución.

Finales de Noviembre y Principios de Diciembre: Se trabaja en optimizar el programa con el objetivo de que quede completo para la fecha límite.

Jueves 13 de Diciembre del 2018: Se realiza la presentación de los resultados.

Lunes 17 de Diciembre del 2018: Se entrega el documento finalizado.

4. El problema

La simulación original genera resultados que difieren con los resultados encontrados en pruebas de laboratorio por un pequeño pero significativo margen. En el primer semestre del 2018 el grupo de Muñoz y Porras lograron crear un programa que optimiza los parámetros y los resultados usando algoritmos genéticos pero los resultados aún difieren por un pequeño pero significativo margen. Aun se desea encontrar la combinación de parámetros más óptima para la simulación.

5. Estado del arte

5.1 Marco teórico

NetLogo: Es la herramienta que se va a utilizar en este proyecto. Es un lenguaje de programación diseñado para la realizar simulaciones multi agente. Tiene variedad de librerías y es posible implementar algoritmos de optimización. Se utilizó en el desarrollo de la simulación de Zika de Ricardo Corrales y Maria Jose Leal. El trabajo de Corrales y Leal luego fue mejorado por Andreina Alvarado, Andrea Gomez y Manuel Arroyo. Finalmente, Jeifer Porras y Anthony Muñoz optimizaron la versión de Alvarado, Gómez y Arroyo. Todas las iteraciones han usado NetLogo y no hay razón en este momento para migrar el trabajo hecho a otra plataforma.

Algoritmos Genéticos: Son métodos adaptativos, generalmente utilizados en problemas basados en reproducción sexual y en el principio de supervivencia del más apto (Gestal et al., 2010)

El desarrollo de los Algoritmos Genéticos es en gran medida gracias a John Holland, el cual es un investigador de la Universidad de Michigan. Donde a finales de la década de los 60 desarrolló una técnica que imita en su funcionamiento a la selección natural (Gestal et al., 2010).

Para complementar tenemos que existe una población de soluciones candidatas. En cada generación, cada candidato se junta al azar con otro candidato para reproducirse y generar un nuevo individuo, el cual contiene la información genética de sus padres, además de posibles mutaciones. Se prueba la aptitud de cada individuo y se eliminan los individuos con menos aptitud por medio de un torneo donde cada individuo se compara contra otros competidores igual al 20% de la población, la aptitud siendo que tanto su solución se acerca a optimizar el problema. Si el individuo más apto no ha cambiado en un número de generaciones, un porcentaje de los individuos menos aptos son cambiados por nuevos individuos con números aleatorios para añadir variedad. (Copasi)

La Optimización Estocástica se utiliza en optimización para introducir valores aleatorios en vez de determinísticos para influenciar la variedad y, por tanto, la aptitud de los individuos.

Algoritmos Genéticos con Clasificación Estocástica: Usa el mismo método que los algoritmos genéticos pero con algunas diferencias. Los valores de las mutaciones son números aleatorios que pueden salirse de los márgenes dados por el problema. Si esto sucede, se aplica una fórmula que genera un valor phi que es usado para la selección de individuos (Copasi).

5.2 Estado de la cuestión

Este proyecto se basa en gran manera con los proyectos presentados previamente del mismo tema. El primero de estos proyectos es el de Ricardo Corrales y María José Leal. En este proyecto se creó la primera simulación del virus del Zika usando NetLogo y utiliza

un autómata finito que tiene un estado inicial de célula sana, un estado medio de célula infectada y un estado final de célula muerta. Cada *tick* de NetLogo se cuenta como una unidad de tiempo discreto y donde la célula tiene una probabilidad de infectarse dependiendo de la cantidad de vecinos infectados y las células infectadas tienen una probabilidad de muerte que aumenta con el tiempo.

Andreina Alvarado, Andrea Gomez y Manuel Arroyo mejoraron el proyecto anterior cambiando la cuadrícula por un círculo y añadiendo un nuevo estado al autómata. El estado es un estado intermedio llamado condensado, una posible condición que puede adquirir las células infectadas

Jeifer Porras y Anthony Muñoz aplicaron algoritmos genéticos para optimizar el proyecto de Alvarado, Gomez y Arroyo. Los resultados obtenidos son muy cercanos a los resultados de laboratorio, pero no son completamente precisos, indicando que los algoritmos genéticos mejoraron al programa pero que aún no es suficiente.

El articulo *Stochastic Ranking for Constrained Evolutionary Optimization* por Thomas P. Runarsson y Xin Yao utiliza estrategias evolutivas para realizar una serie de optimizaciones en trece pruebas de *benchmark* y compararlas con los resultados de otras pruebas que usan los mismos *benchmarks* y los resultados de Runarsson y Yao son mejores en general. Al final de la sección *B. Experimental Results and Discussions* en la página 290 se introduce la pregunta de si los resultados que se obtuvieron fue únicamente por el uso de estrategias evolutivas o si la clasificación estocástica fue un factor importante en la optimización. Los autores prueban estrategias evolutivas y estrategias evolutivas con clasificación estocástica en las mismas pruebas que se han realizado en el artículo. Se encontró que ambos algoritmos no encontraron soluciones factibles en tres de las pruebas, en una prueba se encontró que estrategias evolutivas facilitó una mejor solución pero se desempeñó peor en mediana, media y peor que su contraparte con clasificación estocástica, y una prueba donde ocurrió lo opuesto. De las ocho pruebas restantes las estrategias evolutivas con clasificación estocástica se desempeñó mejor que su contraparte. (Runarsson et al, 2000)

6. Propuesta

6.1 Resumen

Para implementar el algoritmo genético con clasificación estocástica a NetLogo y poder realizar varias pruebas de manera concurrente es necesario implementarlo utilizando algún lenguaje de programación que permita el uso de threads, sea fácil de instalar en un cluster y que se pueda conectarse con NetLogo con facilidad. Se creará entonces un programa que corra varias iteraciones de la simulación de manera concurrente, devuelva los resultados obtenidos y aplique el algoritmo de optimización para generar una nueva población de posibles soluciones que serán enviadas a la simulación y luego optimizadas, todo en un ciclo hasta que se encuentre un caso donde el error entre los resultados de laboratorio y los resultados de la simulación sean cero o el resultado más cercano a cero.

6.2 Metodología

Para facilidad de pruebas en un cluster externo y por el conocimiento de los autores se diseñará un programa en Java que se conecte a la simulación de NetLogo que sea capaz de correr la simulación de manera indefinida y guardar los resultados en un documento de texto para análisis.

7. Desarrollo

En este proyecto se van a usar algoritmos genéticos con clasificación estocástica. Los algoritmos genéticos con clasificación estocástica son similares a los algoritmos genéticos pero difieren en la manera en que manejan la selección de los padres y la mutación de los individuos. La mutación se maneja en este algoritmo añadiendo al parámetro un número aleatorio de distribución normal, media 0 y desviación estándar de un 10% del parámetro. Este valor se obtiene con el Java Random usando *nextGaussian() * (0.10 * parámetro*). La mutación debe de permitir que los valores se salgan de los márgenes o rangos establecidos previamente. Este rango es recomendable que sean valores de interés. Si por mutación un parámetro se sale del rango entonces se muta de nuevo hasta unas 10 veces por cuestiones de tiempo computacional. Si después de 10 veces el parámetro sigue fuera del rango entonces se deja asi, lo que genera un mayor phi (Runnarson et al, 2000).

Para la selección de padres se realiza un bubble sort descrito por Runnarson donde se tiene un valor phi.

$$\varphi = \sum_{p_i < l_{p_i}} (l_{p_i} - p_i)^2 + \sum_{p_i > u_{p_i}} (p_i - u_{p_i})^2 + \sum_{c_j < l_{c_j}} (l_{c_j} - c_j)^2 + \sum_{c_j > u_{c_j}} (c_j - u_{c_j})^2$$

Figura 1: La función phi

Para nuestro algoritmo solo se utilizo la primera mitad de la función donde se da un rango al parámetro y se realiza la función con los extremos de ese rango donde lpi es el mínimo y upi es el máximo. (Runnarson et al, 2000 y COPASI). La función phi

determina que tanto un parámetro se sale del rango definido. El método *swap* incorpora el bubble sort descrito por Runnarson. En este algoritmo de ordenamiento tiene que pasar mínimo una vez por cada individuo que compone la población. Cada individuo se compara con el individuo directamente a la derecha y se revisa si el phi de ambos es igual a 0 que indica que ambos tienen sus parámetros dentro de los rangos establecidos, o si sale un valor menor a Pf=0.475 de un número aleatorio entre 0 y 1 de distribución uniforme, este último nos permite avanzar en el algoritmo incluso si el phi de los individuos es diferente a 0. Si se avanza el algoritmo se revisa el fitness de ambos individuos. Para comparar mejor los resultados con el proyecto de Muñoz y Porras se decidió utilizar la función de error cuadrado mínimo usado en su algoritmo genético.

$$\sqrt{\frac{[(t1-e1)^n+(t2-e2)^n+(t3-e3)^n]}{3}}$$

Figura 2: Función de error cuadrado mínimo

Los valores de laboratorio utilizados son los mismos que en los usados por Muñoz y Porras, proveídos por los microbiólogos.

Células muertas = 68.1157972
Células infectadas o condensadas = 11.1009198
Células vivas = 20.7832829

La función devuelve el error entre los valores promedio encontrados en el laboratorio y los que se encuentra en los resultados de cada simulación, entre más cercanos a 0 más cercanos son a los resultados de laboratorio. Entonces el mejor fitness es uno que se acerque más a 0. Cambiamos entonces la función de swap para que cambie los individuos si el individuo izquierdo tiene un fitness más cercano a 0 que su vecino a la derecha. Si el phi de alguno de los individuos es diferente a 0 o el se da un valor aleatorio superior a Pf=0.475 entonces en vez se lee los valores phi de ambos y se cambian si el individuo actual tiene un phi menor que su vecino derecho. Se verifica con un booleano si ocurre un cambio, si este ocurrió entonces se continúa con la ejecución del ciclo. Si no ocurre ningún cambio se detiene el ciclo con un break.

Se seleccionan los padres más a la izquierda de la lista ya que estos son los más aptos y los que tienen menos parámetros fuera de los márgenes según el swap. Se cruza entonces cada individuo con el que tiene a la izquierda. La función para seleccionar la cantidad de cruces que se van a realizar es

floor((crossover-rate*tamaño-población)/tamaño-pob/2)

Cada padre crea 2 hijos con cada cruce. La población restante se crea por medio de clonación de individuos aleatorios de la población vieja. Antes de devolver la nueva generación se pasa cada parámetro por una mutación donde hay un 10% de que uno de sus parámetros mute. Además se hace una revisión adicional que no es parte del algoritmo. Por mutación es posible que se generen valores que están por fuera de los márgenes de la simulación, por ejemplo valores menores a 0 en parámetros donde no se puede ser menor que 0. Para evitar caídas del programa se añade una revisión final donde se verifica que cada parámetro está dentro de los márgenes de NetLogo. Si no lo están se muta el parámetro eligiendo un valor aleatorio usando el rango de revisión de phi para que se tenga un parámetro relevante que no cause problemas a la simulación.

Se decidió utilizar Java en vez de Python para hacer el programa más rápidamente ya que se tiene mejor conocimiento sobre el lenguaje Java que sobre Python. Una ventaja clara de Java sobre Python es que todas las librerías que sean necesarias para correr en un cluster pueden ser instaladas sin la necesidad de permisos de administrador lo que facilita las pruebas. Si se logró correr el código de Muñoz y Porras, pero cómo se tomó la decisión de usar Java en vez de Python no fue necesario revisarlo a profundidad.

La conexión con NetLogo se hace usando la librería NetLogo.jar incluida en la instalación de NetLogo. NetLogo es un programa diseñado en Java por lo que correr sus funciones usando un programa en Java es más fácil que con uno en Python. La clase NetLogoSim (Umaña, I 2018) introduce los parámetros dados en Hash Maps a la simulación, hace una corrida de 5 ciclos de 24 ticks cada uno para simular la recolección de datos cada 24 horas de los experimentos de laboratorio y retorna la cantidad de células vivas, muertas e infectadas o condensadas.

Para hacer las pruebas es necesario realizar varias simulaciones concurrentes. Para este fin se corre cada una de las simulaciones como un *thread* que termina luego de realizar la simulación y devolver los datos. Los datos que devuelve cada thread son las células vivas, muertas e infectadas en cada uno de los 5 ciclos de la simulación.

La clase NetLogoSim recibe Hash Maps y se requiere que se devuelva cada ciclo de la simulación se modificó la clase, ahora llamada NetLogoSimMod. Esta clase realiza la simulación con 5 ciclos de 24 ticks fijo en cada corrida, para un total de 120 ticks, y almacena en un Hash Map la cantidad de células vivas, muertas e infectadas en cada ciclo.

Como es necesario conocer los resultados que cada serie de parámetros enviados a la simulación devuelven se creó la clase “Solución” la cual está compuesta por un Hash Map que contiene como llave todas las etiquetas relevantes a la simulación, incluyendo los parámetros, los resultados de la simulación y el fitness de los resultados.

Cada corrida del programa genera un número de *threads* igual al número de individuos en la población. Se pasa a la simulación donde se obtienen resultados y estos resultados se pasan al algoritmo genético con clasificación estadística donde se realiza el proceso descrito anteriormente en esta sección. Luego de obtener la nueva población se corre la simulación nuevamente. Se corre una cantidad indefinida de veces.

8. Experimentación y resultados

Para poder realizar las pruebas a profundidad y verificar que los datos obtenidos están siendo optimizados es necesario establecer cuales son los rangos de estudio dentro de los cuales estarán los parámetros. Los parámetros de los individuos iniciales deben de estar dentro de estos rangos. Durante el desarrollo de este proyecto no se logró obtener estos datos, pero se realizaron pruebas usando parámetros de ejemplo para verificar que el algoritmo funciona.

Se establecieron cinco individuos, cada uno con los mismos ocho parámetros que la simulación de NetLogo de Alvarado, Gomez y Arroyo muestra por defecto. Los valores son los siguientes:

cell-density: 2.0
initial-infected-cell-percentage: 0.0
viral-reach: 1.5
infection-rate: 15.0
mNeptune-effectiveness: 100.0
initial-probability-of-death: 0.595
initial-probability-of-chromatin-condensation: 2.5
marker-detection-threashold: 0.12

Al haber cinco de estos individuos se tienen entonces cinco threads que corren de manera concurrente. Cuando se termina cada simulación se obtienen también los resultados de cada ciclo de la simulación, el total de células registradas al final de la simulación y el fitness de este individuo. Toda esta información se guarda en un archivo de texto. El siguiente es un ejemplo de estos resultados

alive 1: 66.0
dead 1: 265.0
infected 1: 67.0
alive 2: 573.0
dead 2: 2811.0
infected 2: 592.0
alive 3: 1143.0
dead 3: 8936.0
infected 3: 1166.0
alive 4: 1878.0
dead 4: 19608.0
infected 4: 1927.0
alive 5: 2568.0
dead 5: 33631.0
infected 5: 2648.0
Total de Células : 38847.0
Fitness :13.661257066160022

Para verificar que el bubble sort se realiza correctamente el programa genera dos archivos de texto, llamados *beforeSwap.txt* y *afterSwap.txt* que muestran la población antes y después del bubble sort. El siguiente son los resultados del fitness de la primera generación antes y después del bubble sort.

Antes

Fitness :13.941013982598568
Fitness :13.540277714742974
Fitness :13.796510992939714
Fitness :13.792611105610362
Fitness :14.1627134839558

Después

Fitness :13.941013982598568
Fitness :13.796510992939714
Fitness :13.540277714742974
Fitness :14.1627134839558
Fitness :13.792611105610362

Existe la posibilidad de que no ocurra ningún cambio dentro del bubble sort dado el azar que ocurre en el algoritmo o si entre vecinos no encuentra diferencias.

Luego de verificar que el método swap, que contiene el bubble sort funcionaba correctamente se decidió hacer una prueba de tiempo indefinido. Se corrió la simulación por 155 minutos y 25 segundos, y se crearon 106 generaciones.

Se encontró que el algoritmo no optimiza los parámetros, pero esto no fue a causa del algoritmo o la implementación. La mayoría de las generaciones creadas devolvieron 0 células vivas y 0 células infectadas, lo cual no es un resultado deseado ni esperado. Se espera que cada corrida de la simulación devuelva además de las células muertas las células vivas e infectadas. Para verificar si esto era un problema del algoritmo de optimización o un problema de NetLogo se corrieron varias simulaciones usando los parámetros que devolvieron 0 vivas y 0 infectadas. Resultó ser que ciertos parámetros causan este comportamiento en la simulación de NetLogo.

Usando los siguientes parámetros

cell-density: 1.3852949372260717
initial-infected-cell-percentage: 1.39611792425527204
viral-reach: 2.111946061896619
infection-rate: 8.448737963811945
mNeptune-effectiveness: 172.72593908437727
initial-probability-of-death: 0.8480299934149297
initial-probability-of-chromatin-condensation: 1.6464817539799932
marker-detection-threashold: 0.5105387908374939

Se obtienen los siguientes resultados en NetLogo

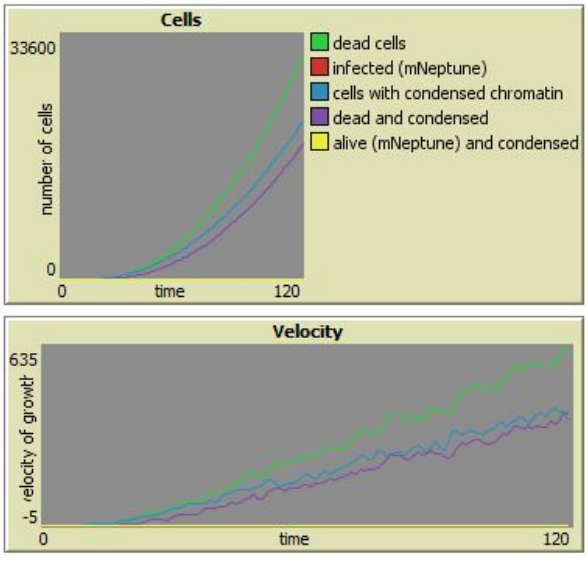

Figura 3 y 4: Resultados de NetLogo con resultados en 0

Que son muy similares a los resultados siguientes obtenidos en de la simulación corrida en Java.

alive 5: 5.0
dead 5: 38797.0
infected 5: 7.0
Total de Células : 38811.0

Esto nos muestra de que hay ciertos parámetros en la simulación de NetLogo que interfieren con la ejecución de la optimización. Esto afecta este algoritmo de optimización más que otros dado a que hay un uso extensivo de variables aleatorias. Esto significa que es posible que por azar se empiecen a generar individuos cuyos parámetros son poco aptos y que esto se esparce a través de las generaciones hasta que ningún individuo sea apto.

Por la falta de definición de parámetros importantes, los rangos finales y el comportamiento de la simulación de NetLogo se decidió no realizar las pruebas en el cluster como se tenía planeado ya que dados los resultados había una alta posibilidad de que los resultados obtenidos en el cluster no mejoraran.

10. Conclusión

Se encontró entonces de que la simulación de NetLogo tiene áreas donde puede mejorar. No se sabe si el algoritmo genético con clasificación estocástica es mejor que el algoritmo genético tradicional por el problema causado por los parámetros de NetLogo. Para poder continuar y analizar el el algoritmo genético con clasificación estocástica va a ser necesario estudiar la simulación en NetLogo y averiguar cuáles parámetros influyen más en que la simulación devuelva los resultados más similares a una prueba de laboratorio y cuales son los valores de parámetros que causan los errores en la simulación.

Dado a que los algoritmos genéticos con clasificación estocástica dependen del uso de valores aleatorios es muy probable que este algoritmo no sea el más eficiente para esta simulación si no se logran encontrar cuales son los parámetros que causan errores en la simulación. El error donde ciertos parámetros devuelve cero células vivas y cero células infectadas o condensadas es común en otros proyectos que utilizan otros algoritmos tratando de optimizar los parámetros de esta simulación. Por esta razón no se puede comparar los resultados de esta implementación con ninguno de los otros algoritmos de optimización.

De encontrarse estos parámetros será necesario modificar el programa para evitar que los individuos puedan tomar estos valores. De esta manera el algoritmo tendrá una mejor oportunidad de encontrar los parametros optimos.

11. Bibliografía

Alvarado, A. Gómez A. y Arroyo, M. *Un modelo del ciclo de vida del virus del Zika*. Informe final del curso CI-1441 Paradigmas computacionales, Escuela de Ciencias de la Computación e Informática, Universidad de Costa Rica, II semestre de 2017".

Corrales, R. y Leal, M. *Simulación de virus transmitidos por mosquitos usando un sistema multiagente*. Informe final del curso CI-1441 Paradigmas computacionales, Escuela de Ciencias de la Computación e Informática, Universidad de Costa Rica, I semestre de 2017".

Muñoz, A. y Porras, J. *Aplicación de métodos computacionales en la optimización de búsqueda de parámetros para la simulación del virus del dengue.* Informe final del curso CI-1441 Paradigmas computacionales, Escuela de Ciencias de la Computación e Informática, Universidad de Costa Rica, I semestre de 2018".

Alvarado, A. Arroyo, M. Corrales, R. Gomez, A. Leal, M. Calderon, A. de la Ossa, A. Arias, J y Mora, R. *Cellular-level characterization of Dengue and Zika virus infection using multiagent solution.* Universidad de Costa Rica, San José, Costa Rica, 2017.

Runarsson, T. P. y Yao, X. *Stochastic Ranking for Constrained Evolutionary Optimization,* IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, VOL. 4, NO. 3, SEPTEMBER 2000.

Gestal, M., Rivero, D., Rámon, J., Dorado, J. and Pazos, A. *Introducción a los Algoritmos Genéticos y la Programación Genética*. Universidade da Coruña, 2010.

McCullagh, P y Yang, J. *Stochastic classification models,* University of Chicago , fecha desconocida.

Copasi, *Genetic Algorithm,* http://copasi.org/Support/User_Manual/Methods/Optimization_Methods/Genetic_Algorithm/, fecha desconocida.

Copasi, *Genetic Algorithm SR*, http://copasi.org/Support/User_Manual/Methods/Optimization_Methods/Genetic_Algorithm_SR/, fecha desconocida.

Wilensky, U NetLogo, http://ccl.northwestern.edu/netlogo/. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL 1999.

Umaña, I. *NetLogoSim Java Class*, proveído por correo electrónico, 2018.

Simha, R. *UniformRandom Java Class*, https://www2.seas.gwu.edu/~simhaweb/cs1112/useful/UniformRandom.java, 1998.