Universidad Central de Venezuela

Facultad de Ciencias

Escuela de Computación

IMPLEMENTACIÓN WEB DE UN VISUALIZADOR DE GRAFOS EN DOS DIMENSIONES UTILIZANDO ALGORITMOS GENÉTICOS

Br. Luis Argüello

Prof. Héctor Navarro, Tutor

Caracas, 7 de agosto del año 2015

Universidad Central de Venezuela

Facultad de Ciencias

Escuela de Computación

IMPLEMENTACIÓN WEB DE UN VISUALIZADOR DE GRAFOS EN DOS DIMENSIONES UTILIZANDO ALGORITMOS GENÉTICOS

Br. Luis Argüello

Prof. Héctor Navarro, Tutor

Caracas, 7 de agosto del año 2015

IMPLEMENTACIÓN WEB DE UN VISUALIZADOR DE GRAFOS EN DOS DIMENSIONES UTILIZANDO ALGORITMOS GENÉTICOS

Br. Luis Argüello

Trabajo Especial de Grado presentado

ante la ilustre Universidad Central de Venezuela

como requisito parcial para optar al título de

Licenciado en Computación.

Universidad Central de Venezuela

Facultad de Ciencias

Escuela de Computación

Centro de Computación Gráfica

ACTA DEL VEREDICTO

Quienes suscriben miembros del Jurado, designado por el Consejo de Escuela de Computación, para examinar el Trabajo Especial de Grado presentado por el **Br. Luis Argüello, C.I. 20616439**, titulado: "**Implementación web de un visualizador de grafos en dos dimensiones utilizando algoritmos genéticos**" a los fines de optar al título de Licenciado en Computación, dejan constancia lo siguiente:

Leído como fue, dicho trabajo por cada uno de los miembros del Jurado, se fijó el día 11 de agosto del 2015 a las 11:00 a.m., para que su autor lo defendiera en forma pública, en el Centro de Computación Gráfica, Facultad de Ciencias, mediante una presentación oral de su contenido. Finalizada la defensa pública del Trabajo Especial de Grado, el Jurado decidió aprobarlo.

En fé de lo cual se levanta la presente Acta, en Caracas a los once días del mes de agosto del año dos mil quince, dejándose también constancia de que actuó como Coordinador del Jurado el Prof. Héctor Navarro.

Prof. Héctor Navarro, Tutor

Prof. Walter Hernández, Jurado Principal

Prof. Miguel Astor, Jurado Principal

Agradecimientos

Primero que nada, quiero darle las gracias a mi tutor, Héctor Navarro: por soportar todas mis visitas incómodas replicándole que leyera mi documento, por haberme guiado en la elaboración del mismo sin poner muchos peros, por haberme enseñado tantas cosas en el período en que fue entrenador de P = NP, por haber colaborado con el grupo VPL y por ser un excelente tutor, profesor, entrenador y amigo.

Agradezco también a todos los **buenos** profesores que estuvieron presentes en mi carrera y me dieron alguna materia, en orden de aparición: Ronald Pietri (MD1), Adrian Bottini (MD2, PyE), Jaime Blanco (MD3), David Batista (Matemática 3), Carlos Guía (SO), Héctor Navarro (LP, TAP, ICG, LPC), Iliana Mannarino (CC1, CC2), Eugenio Scalise (LyC), Alejandro Crema (PM1), Otilio Rojas (RNED) y Marlliny Monsalve (MNO). También agradezco a algunos que no me dieron ninguna materia, pero que estuvieron presentes en algún proceso de aprendizaje: Rhadamés Carmona, Zenaida Castillo y Robinson Rivas.

Quiero agreadecerle también a todos los integrantes de P = NP, comenzando con Daniel Ampuero, por ser la primera persona que vio potencial en mi como programador, el único buen preparador que tuve, el fundador y primer entrenador de P = NP, y sobre todo, por ser un excelente amigo. Nunca desaparecerán de mi memoria los recuerdos de los sábados entrenando en casa de Daniel, resolviendo problemas de SPOJ y UVa, y el mejor de todos, el ejemplo que usó Daniel para enseñarnos Maximum Flow.

Siguiendo con P = NP, también quiero agradecerle a Carlos Guía por todos los buenos consejos de programación, las clases de Segment Tree, DP bottom-up y Min-Cost MaxFlow, y también por formar parte del grupo de entrenadores de P = NP.

Para finalizar con P = NP, quiero agradecerle a **Dixon** "Dickie" Morán y a Humberto "Beto" Rodrígues. Dickie: tengo tantas cosas que agradecerte que nunca

terminaría el documento, así que solo diré las más importantes, gracias por ser el primer amigo que tuve en la universidad, gracias por haberme hablado el día de las inscripciones de los laboratorios de algoritmos, gracias por entrenar conmigo de manera enferma desde el principio, gracias por acompañarme y apoyarme en nuestro viaje a Italia, gracias por todas las veces que tuve que usar tu casa por no poder llegar a la mía, gracias por participar conmigo en el CONNECT y gracias por seguir siendo, a pesar de la distancia que hemos tenido durante este último año, mi mejor amigo. Beto: también tengo mucho que agradecerte, primero que nada, gracias por darnos la oportunidad en el 2011 de ir con nosotros al maratón de la USB, gracias por siempre estar ahí cuando necesitaba hablar de mis problemas, y también por darme un buen golpe de realidad cada vez que lo necesitaba y gracias por pelear conmigo durante los maratones, sin esto no habríamos podido ganar en algunas ocasiones. Ambos: estaré eternamente agradecido con ustedes por hacerme formar parte de un equipo tan excelente y asombroso, que rompió la mayoría de las expectativas; a pesar de que no fuimos nunca a un mundial, igual se que fuimos los mejores de Venezuela en nuestra época dorada, no me cabe la menor duda. Gracias por todos los momentos que compartimos mientras estabamos juntos en la universidad, los martes y viernes de hamburguesas y los fines de semana en casa de Dixon, y sobretodo gracias por ser mis compañeros, mis amigos y mis hermanos. $P = NP$ *for life.*

Quiero agradecer a mi novia, Rosmeli Quintero, por apoyarme durante toda la elaboración de este documento, el seminario, las pasantías y el servicio comunitario; se que fue díficil, por todas las veces que peleamos, pero de verdad gracias por soportarme, por amarme, por aceptarme y por ser una persona tan importante en mi vida, espero que pueda seguir agradeciéndote durante mucho tiempo.

Al resto de mis amigos, que no son muchos, gracias por toda la diversión y el apoyo que me brindaron durante todos estos años, significa mucho para mi.

Por último, quiero agradecer a la persona más importante, la que me dio la vida, mi madre, Lissy Pazos. Muchísimas gracias por ser hacer tantos sacrificios para

sacarme adelante, para ser mi madre y mi padre, para inscribirme en un colegio que sacara provecho al máximo de mi potencial, para comprar la costosa lista escolar cada año, para pagar el curso propedéutico, para tener nuestra casa propia, para pagar la computadora que necesité al comenzar la universidad, para comprar los dólares al mercado negro cuando me iba de viaje y para cubrir todos mis gastos con un simple sueldo de docente durante toda mi vida, sin descuidarme ni una sola vez. Gracias por apoyarme y ayudarme siempre en cualquier cosa que necesitaba, por muy grande o pequeña que fuese, por ayudarme en matemáticas hasta que tus conocimientos lo permitieron y buscar ayuda con profesores del área cuando ya no podías, por llevarme y traerme cuando aún lo necesitaba, por todas las veces que me acompañaste o me esperaste sin tener que hacerlo en realidad y por siempre buscar lo mejor para mí. Sin tí no habría podido llegar ni cerca de donde estoy ahora, **gracias por ser mi mamá**.

Resumen

Implementación web de un visualizador de grafos en dos dimensiones utilizando algoritmos genéticos

Luis Argüello

Prof. Héctor Navarro, Tutor

Universidad Central de Venezuela

El objetivo del presente trabajo de investigación es la implementación de una aplicación capaz de realizar despliegues de grafos en dos dimensiones, minimizando la cantidad de cruces y otras medidas de calidad.

Para la implementación de la interfaz se utilizó el framework web Django. Se utilizó un algoritmo genético para la minimización de la ecuación de energía del grafo, ya que el cálculo de derivadas para los métodos clásicos de optimización no eran viables. Finalmente se realizaron diversas pruebas visuales y de rendimiento para encontrar los parámetros adecuados para la ecuación de energía y el algoritmo genético.

Los resultados obtenidos de las pruebas realizadas permitieron documentar que las medidas de calidad importantes a tomar en cuenta para la ecuación de energía son los cruces entre arcos, la simetría del grafo y el ángulo formado por las aristas de un mismo vértice.

Palabras clave: grafos, visualización de grafos, algoritmos genéticos.

Índice General

Introducción

Con el desarrollo tecnológico actual, es necesario en muchas ocasiones realizar bocetos, o dibujos, que ayuden a entender ciertos problemas relacionados con la vida cotidiana, por ejemplo redes sociales, mapas, redes de computadoras, entre otros; esto se hace con el fin de analizar características, causas y consecuencias de los mismos. Aunque esto no sea un problema nuevo, es algo que ha ido creciendo a lo largo de los años.

Por supuesto, no siempre se tiene el tiempo, o la disposición, de realizar bocetos o dibujos si el problema es increíblemente grande. Es por esto que con el pasar del tiempo se han ido desarrollando tecnologías que permiten automatizar este proceso de manera rápida y efectiva.

Los problemas que pueden crecer y alcanzar magnitudes que requieran este tipo de tecnologías son, en su mayoría, problemas relacionados con la teoría de grafos. En el capítulo uno (1) se propone el desarrollo de un nuevo software que permita, además de realizar la visualización de grafos, importar y exportar el mismo en formatos conocidos, y que sea capaz de generar grafos aleatorios con ciertos parámetros dados.

En el capítulo dos (2), se hablará sobre la teoría de grafos y se describirán algunos ejemplos de problemas puntuales donde está presente.

En el capítulo tres (3) se plantearán de algunos algoritmos existentes que permiten modelar problemas relacionados a grafos, junto con tecnologías ya existentes que realizan el proceso de dibujado de manera automática para un análisis más rápido y sencillo del problema.

En los capítulos cuatro (4) y cinco (5) se explicará en funcionamiento de los algoritmos genéticos, que serán utilizados como herramienta principal para definir la configuración de los dibujos. Adicionalmente se hablará de resultados de otras personas que han hecho trabajos similares.

En el capítulo seis (6) se hablará de la herramienta a utilizar para la implementación, junto con todas sus características técnicas y ventajas para este trabajo de investigación.

El capítulo siete (7) tratará todos los aspectos técnicos de la implementación de la aplicación, desde los algoritmos utilizados hasta las herramientas, frameworks y librerías necesarias para el correcto funcionamiento de la misma.

Finalmente, en el capítulo ocho (8), se presentarán resultados de todas las pruebas necesarias para establecer valores óptimos para los parámetros de los algoritmos, abarcando desde constantes para la presentación estética del grafo, hasta valores para mejorar tiempo de ejecución y resultados.

Capítulo 1

El problema

El problema radica en realizar una visualización de grafos de manera computacional y automática, tomando como entrada cualquier grafo. Cabe destacar que el tamaño de la entrada estará restringido a los algoritmos implementados y al hardware que se posee actualmente.

A pesar de ser un trabajo que se ha realizado antes en muchas ocasiones (Ver capítulo 5), nunca se ha documentado cuáles medidas de calidad resultan en grafos bien o mal visualizados.

1.1. Justificación

Anteriormente se dijo que los grafos existen, explícita o implícitamente, en casi todo lo que nos rodea; desde mapas de ciudades hasta composiciones de redes o moléculas. Muchas veces es necesario realizar un análisis sobre estos grafos, y se procede a hacer un bosquejo gráfico, en caso de ser posible, del mismo. Sin embargo, muchas veces no se tienen las herramientas, o no es posible realizar un bosquejo gráfico manualmente; por eso es que existe toda la teoría de la visualización de grafos de manera computacional. En la figura 1.1 se puede apreciar la diferencia entre una mala y una buena visualización; con la mala visualización se puede decir que es un grafo ordinario, pero al estar bien visualizado queda claro que el grafo sigue un patrón.

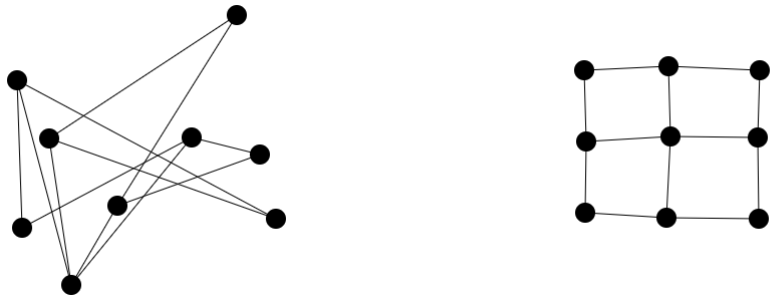


Figura 1.1: Comparación entre una mala y una buena visualización de un mismo grafo.

En la opinión del autor, además de los fines analíticos descritos, es muy importante resaltar también los fines académicos. Al momento de enseñar teoría de grafos y todos los algoritmos que la rodean es necesario tener una representación gráfica del grafo, o los grafos, que están involucrados en ese proceso de aprendizaje para agilizar el mismo; obtener dicha representación involucra mucho tiempo ya que debe realizarse manualmente o con un programa de dibujo genérico.

Adicionalmente, se tiene que actualmente no existe suficiente documentación sobre toda la teoría de la visualización de grafos, incluyendo el uso de sus métricas y la proporción de las mismas.

1.2. Objetivo general

El objetivo general es la implementación de una aplicación que permita la visualización automática de grafos, utilizando un algoritmo basado en una función de energía.

1.3. Objetivos específicos

Los objetivos específicos son los siguientes:

- Implementar algoritmos de visualización dirigidos por fuerzas, específicamente utilizando ecuaciones de energía.
- Determinar la ecuación de energía que genere grafos estéticamente agradables de manera automática realizando pruebas donde se varíe el peso asignado a cada una de las medidas de calidad en la ecuación de energía asociada al grafo y se comparen los diferentes resultados obtenidos.
- Generar como salida un despliegue en dos (2) dimensiones que sea manipulable de forma manual.
- Tomar como entrada grafos en formato GML.
- Poder exportar el grafo en algún formato de imagen de mapas de bits o idealmente vectorizados.
- Implementar funciones de cambios de colores de vértices y aristas.
- Poder generar grafos aleatorios o con parámetros determinados, entre ellos el número de vértices, número de aristas y grado máximo por vértice.
- Poder guardar un despliegue como un modelo, bien sea en formato GML o algún otro formato, que, al ser cargado de nuevo en la aplicación, el despliegue sea exactamente el mismo que el que se guardo en un principio.
- Implementar la edición manual de un grafo, agregando vértices y aristas, junto con un botón opcional que aplique el algoritmo de visualización al grafo en cuestión.
- Implementar un algoritmo genético que permita la minimización de la ecuación de energía.
- Realizar pruebas sobre el algoritmo genético, variando el método de selección de individuos y la cantidad de individuos preservados por elitismo para conseguir el mejor resultado posible.

1.4. Herramientas

Para lograr los objetivos descritos, es necesario hacer uso de las siguientes herramientas:

- Para la implementación de la aplicación se utilizará el framework web Django [1].
- En conjunto con Django, para realizar los despliegues se utilizará el elemento canvas implementado en HTML5 [2].
- Para la interfaz de usuario se usarán las bibliotecas de Bootstrap [3] y jQuery [4].

Capítulo 2

Teoría de grafos

La teoría de grafos es una rama de las matemáticas discretas que se encarga del estudio de los grafos y sus propiedades, así como las ventajas de sus aplicaciones en problemas que pueden presentarse en la vida cotidiana.

2.1. Grafos

Los grafos son estructuras matemáticas que constan de dos conjuntos, uno de vértices y otro de aristas. Los vértices normalmente se utilizan para representar algún tipo de dato, mientras que las aristas se usan para representar el modo en que esos datos se relacionan. Un ejemplo sencillo sería un sistema de ciudades y carreteras, o vías, que unen a esas ciudades; las ciudades serían los vértices del grafo y las carreteras serían las aristas.

La definición matemática de un grafo es de la siguiente forma:

$$G = (V, E)$$

Donde G representa al grafo, V es el conjunto de vértices y E el de aristas.

Algunas características especiales de los grafos son:

- Se dice que un grafo es completo si todos los vértices poseen una arista que los conecte con todos los demás vértices. La notación que se utiliza en la teoría de grafos para referirse al grafo completo de n vertices es: K_n.

- Si existe más de una arista que conecte el mismo par de vértices, se dice que el grafo es un multigrafo.

- Un grafo puede ser dirigido o no dirigido. Se dice que un grafo es dirigido cuando las aristas tienen una dirección. Por ejemplo, en un grafo dirigido un vértice i puede estar relacionado con un vértice j, pero el j no se relaciona con el i. En cambio, en un grafo no dirigido, la relación que se establece entre los vértices tiene su inversa bien definida, entonces si el vértice i se relaciona con el j, entonces el j también se relaciona cone el i.

- En un grafo, un camino es una secuencia de vértices donde el vértice i siempre tendrá una arista que lo conecte con el vértice $i + 1$. Cuando el primer vértice de esa secuencia coincide con el último, entonces estamos hablando de un ciclo o circuito.

- Se dice que un grafo es conexo si existe al menos un camino que conecta a cada par de vértices.

- Se dice que un grafo es bipartito si es posible separar los vértices del mismo en dos subconjuntos separados, siempre y cuando no exista una arista que conecte dos vertices de un mismo conjunto. Un caso particular de estos grafos es el grafo bipartito completo, de $n + m$ vértices. Se toma un conjunto de n vértices y otro de m vértices. Luego se relacionan todos los elementos del primer conjunto con todos los del segundo conjunto, generando un total de $n \times m$ aristas. La notación para los grafos bipartitos completos es $K_{n,m}$.

2.1.1. Representación

En las ciencias de la computación, los grafos pueden representarse por medio de dos tipos de estructuras: matrices y listas.

Las matrices pueden utilizarse de dos maneras, como matrices de incidencia o de adyacencia. Cuando se tiene una matriz de incidencia, la misma tiene dimensiones $N \times M$, donde N es la cantidad de vértices y M es la cantidad de aristas; dicha matriz consiste en un conjunto de ceros y unos, donde la posición (i, j) tendrá un 1 si la arista j incide en el vértice i. Debido al increible e indebido uso de memoria de esta técnica de representación, la misma no es muy utilizada. Por otro lado, las matrices de adyacencia tienen dimensiones $N \times N$, donde N es la cantidad de vértices. También es una matriz de ceros y unos, pero con el cambio de que si la posición (i, j) tiene un 1, entonces los vértices i y j tienen una arista que los conecta.

Las listas pueden utilizarse de dos maneras, como listas de adyacencia o de incidencia. Las listas de adyacencia funcionan de manera similar a las matrices de adyacencia. Consiste en realizar una lista por cada vértice, cuyo contenido son todos los vértices con los que está relacionado por medio de las aristas. Las listas de incidencia son un conjunto de pares donde el par (i, j) representa que existe una arista que conecta al vértice i con el vértice j.

En la figura 2.1 se puede ver la representación gráfica de un grafo, donde los círculos, o puntos, dependiendo de la representación, son los vértices y las aristas son las líneas que los conectan.

2.1.2. Grafo complemento

El grafo complemento es una figura importante dentro de la teoría de grafos. Es el resultado de aplicar la operación complemento a un grafo. Contiene todos los vértices del grafo original, pero las aristas son solo aquellas que le faltan al grafo original para llegar a ser un grafo completo.

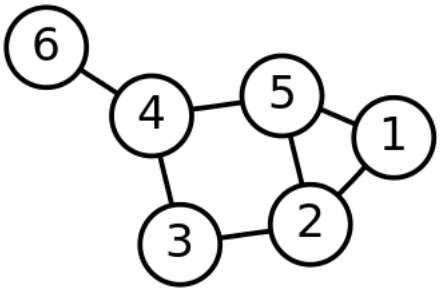


Figura 2.1: Representación gráfica de un grafo [7].

2.1.3. Isomorfismo de grafos

Como su nombre lo indica, dos grafos son isomorfos[8] si tienen la misma forma. Para que dos grafos G y H sean isomorfos, debe existir una función $f : V(G) \to V(H)$ que preserve la adyacencia de los vértices. Esto quiere decir que si en G, los vértices (i, j) están conectados, entonces en H los vértices $(f(i), f(j))$ también deben estar conectados. En la figura 2.2 se puede ver un ejemplo de un isomorfismo de grafos.

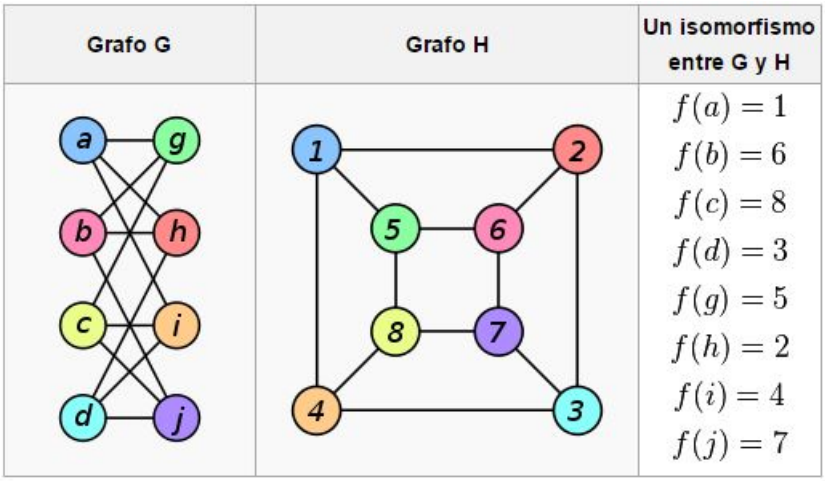


Figura 2.2: Isomorfismo de grafos [9].

2.1.4. Subgrafos

Un subgrafo, como su nombre lo indica, es un grafo que está compuesto por un subconjunto de vértices y aristas del grafo original. Existen dos clasificaciones especiales de subgrafos, los recubridores y los inducidos.

Un subgrafo es recubridor cuando conserva exactamente el mismo conjunto de vértices del grafo original, pero se toma un subconjunto estricto del conjunto de aristas. Por otro lado, un subgrafo inducido es aquel que se hace quitando uno o más vértices del grafo original, junto con las aristas que inciden en él.

2.1.5. Grado de un vértice

El grado de un vértice es una característica que indica el número de aristas que inciden en él. Para el caso de los grafos dirigidos se tienen dos grados diferentes, el grado de entrada y el grado de salida. Este dato normalmente se utiliza para ciertos teoremas e incluso para el isomorfismo de grafos.

2.1.6. Circuito de Euler y Hamilton

El circuito de Euler es un problema de la teoría de grafos cuyo origen se remonta al año 1736 [10]. Consiste en conseguir un ciclo que pase exactamente una sola vez por todas las aristas del grafo. Existe una forma sencilla de probarlo, gracias a un teorema descubierto en el año 1873 [11]. Este teorema establece que un grafo tiene un circuito euleriano si y solo si todos sus vértices tienen grado par.

El circuito de Hamilton es un problema de la teoría de grafos NP-Completo [13]. Consiste en hallar un ciclo que pase exactamente una sola vez por cada vértice del grafo. Desafortunadamente, no existen teoremas para probar esta condición. Sin embargo, existen muchísimos algoritmos especialmente implementados para solucionar de la manera más eficiente este problema.

2.1.7. Grafo planar y grafo ponderado

En la teoría de grafos, el problema del grafo planar consiste en decir si un grafo puede dibujarse en dos dimensiones sin que las aristas se intersecten. En el caso de que si se pueda, el grafo en cuestión es conocido como un grafo plano o planar. Existen teoremas conocidos para demostrar si un grafo es plano, se tiene el teorema de Kuratowski, la fórmula de Euler, entre otros [15].

Por otro lado, el grafo ponderado es aquel cuyas aristas estan relacionadas a una función de costo. El objetivo de insertar costos a las aristas de los grafos es dar origen a problemas de minimización, los cuales pueden relacionarse con problemas conocidos y resueltos como el problema del camino mínimo, resuelto por el algoritmo de Dijkstra [14].

2.2. Árboles

Los árboles, en la teoría de grafos, son grafos no dirigidos que cumplen con las siguientes propiedades equivalentes entre sí:

- Cada par de vértices esta conectado por exactamente un solo camino.
- El grafo es conexo y posee v vértices y $v - 1$ aristas.
- El grafo es conexo y no contiene ningún ciclo.
- El grafo es conexo y cualquier arista extra que se añada al grafo forma un ciclo.

2.2.1. Otras definiciones y tipos

Existen muchas otras propiedades y definiciones concernientes a los árboles, entre ellas podemos encontrar las siguientes:

- Un conjunto de dos o más árboles disjuntos se conoce como bosque.
- Una hoja es un vértice de grado uno dentro del árbol.
- Un árbol dirigido es un grafo dirigido que comple las propiedades de los árboles cuando se elimina la dirección en sus aristas.
- Un árbol con raíz es aquel donde se asigna un vértice como la raíz del árbol, lo que brinda un orden parcial a todos los vértices y aristas del mismo, dando a entender que las aristas tienen una dirección natural hacia los hijos o hacia el padre.
- Un vértice padre es aquel que se encuentra en el camino de un vértice a la raíz. Mientras que un hijo de un vértice v es aquel cuyo padre es el vértice v.
- Un árbol n-ario es un árbol con raíz que cumple con la condición de que cada vértice posea a lo sumo n hijos. Se dice que es n-ario completo si todos los vértices poseen 0 o n hijos.

2.2.2. Árboles recubridores y de expansión minimal

Se conoce como árbol recubridor de un grafo G a un árbol que es al mismo tiempo un subgrafo recubridor de G. Existen varios métodos para hallar dichos árboles dado el grafo inicial, los principales son la búsqueda por anchura [16] y la búsqueda por profundidad [17]. En breves palabras, la búsqueda por anchura hace un recorrido por nivel del grafo y la búsqueda por profundidad se basa en un backtracking [18] que recorre el árbol recursivamente.

Un caso particular de los árboles recubridores son los árboles de expansión minimal. Estos últimos estan fuertemente atados a los grafos ponderados, ya que se utilizan para minimizar la sumatoria del costo de las aristas del arbol recubridor del grafo ponderado. Existen varios algoritmos para la resolución de este problema, pero,

en la opinión del autor, los más usados en la práctica son los siguientes: el algoritmo de Prim [19] y el algoritmo de Kruskal [20].

2.3. Aplicaciones

Una de las aplicaciones más populares es el problema de hallar la ruta óptima, independientemente del medio de transporte terrestre, de un sitio a otro. Puede ser una línea de autobuses donde se minimice el costo entre cada una de las paradas, con el algoritmo de Dijkstra [14] o el algoritmo de Floyd-Warshall [21].

Otra aplicación muy útil es el modelado de proyectos en función del tiempo y las prelaciones de actividades para el cálculo de la duración óptima del proyecto. Existen dos métodos que comparten este objetivo, el método de la ruta crítica [22] y las técnicas de revisión y evaluación de proyectos [23].

En ciencias de la computación [24] hay una infinidad de aplicaciones de la teoría de grafos. En la computación gráfica se tienen, entre otros, dos buenos ejemplos: la detección de colisiones en espacios tridimensionales, donde se utilizan arboles para la representación de los objetos [25], y también en la segmentación de imágenes, donde se utilizan las técnicas de árboles de expansión minimal y otras técnicas de grafos [26]. También en el area de programación web, como por ejemplo las estructuras DOM para el manejo de las páginas web por medio de los browsers [27].

Un área interesante donde se aplica la teoría de grafos es el flujo en redes. Problemas como el máximo flujo que puede soportar una red en un momento dado [28], se aplican a redes de cualquier tipo, desde redes de agua hasta redes de computadoras.

Así como las antes mencionadas, existen infinidades de aplicaciones para la teoría de grafos, tanto en la vida académica, como en la vida laboral y cotidiana.

Capítulo 3

Visualización de grafos

La visualización de grafos es un área de la mátematica y ciencias de la computación que consiste en representar gráficamente un grafo en dos dimensiones. El objetivo de realizar esta acción es tener una visualización parcial o total de un grafo con fines analíticos.

Dentro de las convenciones de la visualización se tiene que los vértices normalmente se representan como círculos o cuadrados, mientras que las aristas se representan como líneas rectas o curvas que conectan un par de vértices. Si el grafo es dirigido, a estas líneas se les incorpora un indicador en forma de flecha para hacer saber la dirección de la arista.

Esta, como todas las áreas fuertemente relacionadas con la teoría de grafos, tiene aplicaciones en muchos campos; entre ellos tenemos:

- Flujo en redes.
- Diagramas de estado, los cuales se utilizan para la representación de las máquinas de estados finitos.
- En biología, para representar proteínas.
- En química, para representar moléculas.

- Redes de computadoras.
- Redes sociales.
- Incluso los grafos como tal, con fines académicos.

A lo largo de este capítulo se hablarán de muchos formatos para modelar grafos antes de dibujarlos, el más importante de ellos es el formato GML [54].

GML viene de Graph Modelling Language, y es un formato de archivo para representar grafos de manera sencilla y portable. Este formato utiliza únicamente caracteres ASCII para representar todo el grafo, incluyendo atributos, formas, colores, etiquetas, entre otras cosas que puedan contener vértices y aristas. En la figura 3.1 se puede ver un ejemplo de la sintaxis de este formato.

3.1. Medidas de calidad

Como muchas cosas, la visualización de grafos necesita medidas de calidad que ayuden a decidir a los diferentes algoritmos cuando un grafo está bien o mal visualizado. Existen muchas y diferentes medidas de calidad que, a pesar de que no son absolutas, buscan una mejoría de la estética y la usabilidad del grafo. A continuación se presentan las más importantes:

- El número de cruces de aristas debe ser mínimo. Lo ideal sería que no existieran cruces en la visualización, pero eso solo es posible si el grafo en cuestión es planar.
- El área que cubre el grafo visualizado debe ser mínima. Esta medida en realidad se refiere al área del bounding box [31] que cubre al grafo visualizado en su totalidad. En muchos de estos algoritmos también se toma en cuenta la proporción de la altura con el ancho del bounding box, ya que de nada sirve minimizar el área del bounding box si aún se puede tener, por ejemplo, un bounding box con una altura muy pequeña, pero con un ancho muy grande.

```
graph [
        node [
                id = 1
                label = "1"
        ]
        node [
                id = 2
                label = "2"
        ]
        node [
                id = 3
                label = "3"
        ]
        node [
                id = 4
                label = "4"
        ]
        node [
                id = 5
                label = "5"
        ]
        node [
                id = 6
                label = "6"
        ]
        edge [
                source 1
                target 2
        ]
        edge [
                source 1
                target 5
        ]
        edge [
                source 2
                target 3
        ]
        edge [
                source 2
                target 5
        ]
        edge [
                source 3
                target 4
        ]
        edge [
                source 4
                target 5
        ]
        edge [
                source 4
                target 6
        ]
]
```

Figura 3.1: Representación GML para el grafo mostrado en la figura 2.1.

- Algunos algoritmos también toman en cuenta la simetría del grafo; la misma consiste en desplegar un grafo que no se encuentre desproporcionado.
- Con respecto a las aristas: su forma debe ser tan simple como sea posible. Su longitud debe ser mínima y todas las aristas deben tener una longitud similar, evitando grandes diferencias entre la arista más corta y la más larga.
- El menor ángulo formado por dos aristas que inciden en un mismo vértice debe ser maximizado.

Evidentemente, no se pueden maximizar todas las medidas en una misma visualización, es por eso que los algoritmos que serán descrito más adelante varían tanto en el uso de las medidas de calidad.

3.2. Algoritmos de visualización de grafos

En esta sección se describirán algunos de los algoritmos utilizados para la visualización de grafos. Para cada uno se describirá su funcionamiento y se incluirán imágenes de sus resultados.

3.2.1. Visualización dirigido por fuerzas

El algoritmo de visualización de grafos dirigido por fuerzas consiste en modificar la disposición inicial del grafo utilizando como guía un sistema de fuerzas asignado al grafo.

Uno de los métodos más comunes se basa en la ley de Hooke [33]; la ley de Hooke establece que el alargamiento unitario que experimenta algún material elástico es directamente proporcional a la fuerza aplicada al mismo. Para los resortes, la fórmula que define la Ley de Hooke puede verse en la ecuación (3.1), donde F representa la fuerza, k es la constante elástica y δ es el alargamiento del resorte.

$$F = -k\delta \tag{3.1}$$

El algoritmo consiste en asignar un tipo de fuerza de resortes a las aristas, y una fuerza repelente en nodos que no están conectados entre sí. Luego se lleva a cabo una simulación, con diferentes parámetros, hasta que se obtiene una configuración aceptable. En la figura 3.2 se puede observar un ejemplo donde se aplica esta técnica.

La simulación, que normalmente es $O(n^3)$ en tiempo, consiste en verificar qué sucede en cada instante del nuevo estado de los vértices con respecto a las fuerzas que se les aplica a cada uno.

Figura 3.2: Configuración inicial y final de un grafo al aplicar este algoritmo.

Se puede apreciar que los resortes (o aristas) comprimidos, se estiraron, mientras que los que estaban muy estirados, se comprimieron. Todo esto con el propósito de lograr un equilibrio con respecto a la longitud de cada arista.

También existen mecanismos asociados a ecuaciones de energía relacionadas con el grafo. Sigue el mismo principio de fuerzas, pero no son representadas físicamente, sino como una ecuación que involucra a todos los nodos y aristas del grafo en una configuración inicial. Lo ideal es minimizar la energía total del grafo, o lo que es lo mismo, minimizar la ecuación de energía del grafo. Existen diversos métodos con este fin dentro del área de optimización numérica [34], por ejemplo, los métodos de Newton o Cuasi-Newton. La ventaja de utilizar una ecuación de energía es que el algoritmo, dependiendo del método utilizado para la minimización, puede llegar a ser un poco más eficiente.

Las ventajas de este algoritmo son:

- Cumple con varias medidas de calidad, como lo son la simetría, la longitud uniforme de las aristas, y la distribución uniforme de vértices que permite un bounding box relativamente pequeño.
- Es un algoritmo muy expandible, con el objetivo de incorporar una o más medidas de calidad. Por ejemplo, los cruces pueden ser minimizados si son incorporados a la ecuación de energía.
- Es simple e intuitivo, al menos el modelo físico de fuerzas con resortes y fuerzas repulsivas.

Una de las desventajas principales del algoritmo es su complejidad en tiempo. De alguna u otra forma, el algoritmo siempre puede rondar en el $O(n^3)$, lo que imposibilita la visualización de grafos con poco más de mil (1000) vértices. Sin embargo, existen técnicas para reducir la complejidad a $O(n^2 lgn)$, por ejemplo, la simulación de Barnes-Hut [35]. Otra desventaja del algoritmo es que, dependiendo de la ecuación de energía que se intenta minimizar, el mínimo hallado no siempre es óptimo. Puede ser un mínimo local, cuando se desea buscar el mínimo global. En la figura 3.3 se puede observar un ejemplo del algoritmo dirigido por fuerzas.

3.2.2. Visualización por niveles

El algoritmo de visualización por niveles, también conocido como el estilo Sugiyama [37] de visualización de grafos, consiste en dividir un grafo en niveles, o capas, tratando de mantener principalmente dos condiciones: que los vértices estén tan uniformemente distribuidos como sea posible, y que una misma arista no viaje a través de muchas capas. Por la naturaleza de este algoritmo, está dirigido a grafos dirigidos acíclicos o casi acíclicos.

Figura 3.3: Otro ejemplo de aplicar el algoritmo dirigido por fuerzas.

El objetivo principal de este algoritmo es minimizar la cantidad de cruces de aristas, lo cual es un problema NP-Hard [38], pero se propone una serie de heurísticas que, a pesar de no brindar un resultado óptimo, son bastante buenas en la práctica.

El algoritmo funciona de la siguiente manera:

1. Si el grafo no es acíclico debe pasar por un procedimiento que lo haga acíclico. Encontrar el conjunto más pequeño de aristas que al ser invertidas conviertan el grafo en acíclico es un problema NP-Completo, por lo que se utilizan algunas heurísticas que aproximen un buen conjunto. Sin embargo, este conjunto óptimo puede ser hallado utilizando algoritmos de programación entera [39] como ramificación y acotación [40].

2. Se crean los niveles, o capas, y los vértices del grafo se distribuyen en ellos. El objetivo principal de este paso es conseguir la mejor distribución posible de los vértices, con la menor cantidad de niveles y la menor cantidad de aristas que viajan por muchos niveles al mismo tiempo. Existen diferentes técnicas para realizar este paso, pero las más recomendadas son programación lineal [41] y programación entera.

3. Se crean vértices fantasmas en las aristas que viajan por muchos niveles, de tal forma que una arista solo conecte vértices de niveles adyacentes.

4. Se procede a la permutación de vértices en los diferentes niveles, minimizando el número de cruces generados por la distribución del paso dos (2). Este procedimiento es un problema NP-Completo, por lo que, nuevamente, se requiere el uso de heurísticas que traten en lo posible de minimizar el número de cruces entre niveles.

5. El algoritmo finaliza invirtiendo de nuevo las aristas del conjunto hallado en el paso uno (1) en caso de ser necesario, y eliminando los vértices fantasmas colocados en el paso tres (3).

La ventaja de este algoritmo es que, dependiendo de las heurísticas que se utilicen en cada caso, puede llegar a tener una complejidad en tiempo de $O(nm)$, siendo n el número de vértices y m el número de aristas. En algunos casos, esto también puede ser una desventaja, porque el número de vértices puede ser inmenso al crear los vértices fantasmas, haciendo este algoritmo innecesariamente lento. En la figura 3.4 se puede observar un ejemplo del algoritmo de visualización por niveles.

3.2.3. Visualización por diagrama de arcos

El algoritmo de visualización por diagrama de arcos es en realidad el algoritmo más simple de todos. Consiste en colocar los vértices en una línea recta, preferiblemente de manera horizontal, y representar las aristas como semicírculos que conecten cada par de vértices.

El único problema que se presenta con esto es la gran cantidad de cruces que pueden haber. Para esto, existen métodos asociados a este algoritmo que reducen notablemente dicho número. Uno de estos métodos consiste en tener un órden fijo para los vértices y convertir el problema en un problema de satisfacibilidad 2 [43].

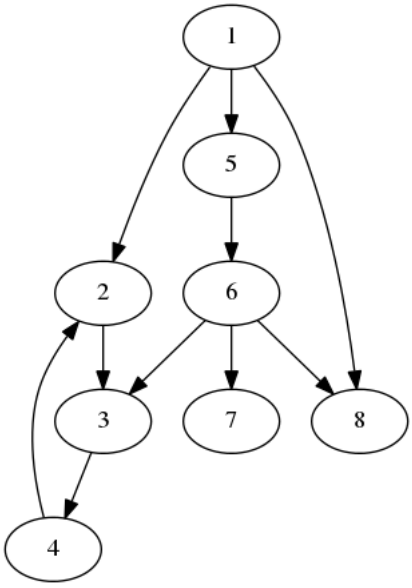


Figura 3.4: Ejemplo de visualización por niveles utilizando Graphviz.

Cabe destacar que también existen heurísticas discutidas a través de los años por diferentes autores que, a pesar de no hallar la configuración óptima, ayudan a disminuir considerablemente el número de cruces. En la figura 3.5 se puede observar un ejemplo del algoritmo de visualización por diagrama de arcos.

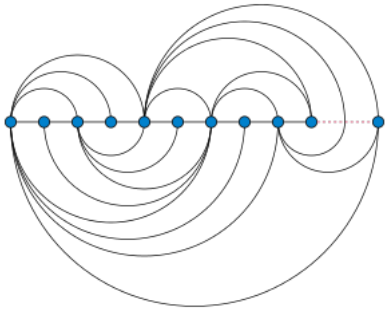


Figura 3.5: Ejemplo de visualización por diagrama de arcos.

3.2.4. Visualización circular

El algoritmo de visualización circular consiste en distribuir los vértices en una o más circunferencias minimizando los cruces que pueden haber dentro de cada circunferencia. Se dice que cada circunferencia contiene una componente biconexa [45], lo que permite que el resultado de este algoritmo sea, visualmente, una "separación" o agrupación de vértices bastante distinguible.

Su ventaja principal es su complejidad en tiempo; este algoritmo posee una complejidad de $O(m)$, donde m es el número de aristas para armar las componentes y su distribución; y una complejidad de $O(m^2)$ para minimizar los cruces en cada una de las componentes.

En la figura 3.6 se puede ver cómo afecta este algoritmo a una componente biconexa; y la figura 3.7 es un resultado final de un grafo cualquiera.

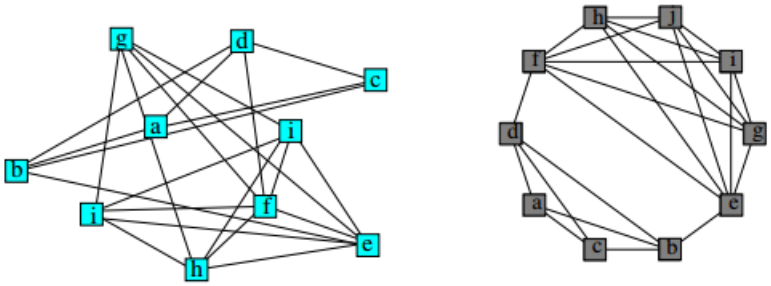


Figura 3.6: Algoritmo aplicado a una componente biconexa.

3.3. Software existente

En esta sección se describirán algunos programas existentes que se dedican parcial o totalmente a la visualización de grafos. En cada uno se presentarán ejemplos y se hará énfasis en dar a conocer si se implementan los algoritmos mencionados anteriormente.

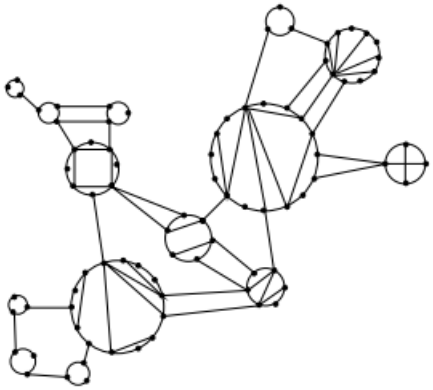


Figura 3.7: Resultado final de un grafo cualquiera.

3.3.1. Cytoscape

Cytoscape es una plataforma de software libre diseñada para la visualización de la interacción de redes. El software fue diseñado en un principio solo con fines biológicos, pero hoy en día es una plataforma compleja que permite la representación de cualquier red.

Para los desarrolladores, Cytoscape posee un API basado en Java, lo que permite su integración en otras aplicaciones o plugins. Además, también existe una biblioteca de JavaScript, llamada "Cytoscape.js" que permite el despliegue de una red modelada en Cytoscape en una interfaz web. Lo más llamativo de la versión web de Cytoscape es, en la opinión del autor, que es interactiva; no es simplemente una imágen que se cuelga en un documento web, permite hacer zoom, seleccionar y mover diferentes vértices. Se puede ver un ejemplo de esta versión web en la figura 3.8.

Algunas de las características de Cytoscape son:

- Tiene su propia App Store donde los desarrolladores cuelgan sus aplicaciones integradas con Cytoscape.
- Soporta múltiples idiomas.

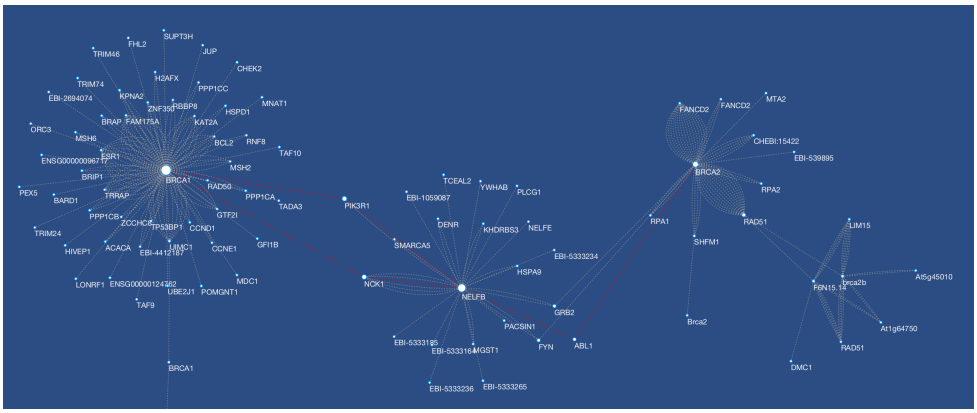


Figura 3.8: Ejemplo de un resultado de Cytoscape.js.

- Permite exportar sus resultados en imágenes y formato PDF.
- Implementa diferentes tipos de algoritmos para hacer la visualización, entre ellos tenemos: el circular y el dirigido por fuerzas.
- Soporta múltiples formatos de archivos de entrada como SBML [47], OBO [48], entre otros.
- Permite hacer zoom y manipular manualmente los resultados de los algoritmos.

En la figura 3.9 se presenta un conjunto de ejemplos tomadaos de la página web de Cytoscape.

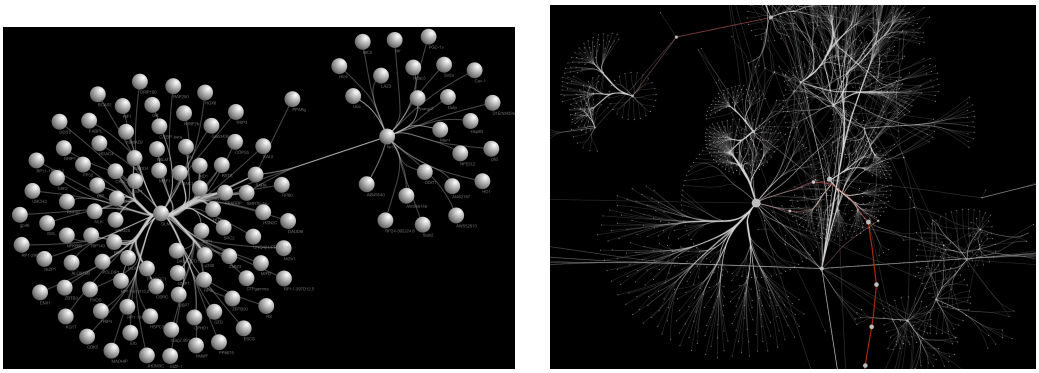


Figura 3.9: Ejemplos tomados de la página web de Cytoscape [46].

3.3.2. Gephi

Gephi es un software libre multiplataformas cuyo objetivo es brindar la oportunidad de explorar, entender y estudiar los grafos. Se utiliza mucho para análisis de estadísticas y de interacción de componentes.

El motor de Gephi es OpenGL, lo cual permite que sea una herramienta bastante eficiente. Puede dibujar hasta $50,000$ vértices sin problemas y de manera dinámica, es decir, agregando, eliminando y modificando posiciones manualmente mientras el software trabaja.

Gephi no posee muchos algoritmos de visualización, básicamente solo posee variaciones de algoritmos dirigidos por fuerzas. Además de esto, también provee métricas y estadísticas sobre los grafos que despliega.

Al igual que otros software existentes, Gephi provee un API, por lo que existen múltiples plugins basados en Gephi, disponibles en el centro de plugins de Gephi.

En las figuras 3.10 y 3.11 se pueden observar ejemplos tomados de la página web de Gephi.

3.3.3. Graphviz

Graphviz es un software libre de visualización de grafos. Funciona tomando una entrada en formato DOT [51], y luego, aplicando diferentes algoritmos para realizar despliegues de grafos que pueden ser guardados en formatos de imágenes y PDF. También permite modificar colores, fuentes, distribuciones, entre otras cosas.

Algunos de los algoritmos que implementa Graphviz son: visualización por niveles, dirigido por fuerzas, con y sin ecuación de energía, y circular. En su página web se encuentran diferentes tutoriales sobre los comandos y cuál algoritmo aplica cada uno.

Graphviz, a diferencia de otros software, no posee su propia interfaz, está hecho para integrarse a diferentes ambientes de desarrollo, como por ejemplo Visual Studio.

En la figura 3.12 se puede observar un ejemplo tomado de la página web de Graphviz.

3.3.4. Mathematica

Mathematica es un software propietario desarrollado por Wolfram. Es en realidad un software muy completo y extenso, tan extenso que la visualización de grafos es solo un módulo de tantos que tiene el programa.

El módulo dedicado a la visualización de grafos de Mathematica permite una visualización de grafos y redes con fines analíticos. Sea cual sea el grafo, Mathematica provee una serie de algoritmos de alto nivel que son capaces de dibujarlo.

Provee todos los algoritmos descritos en la sección 3.2 y muchos otros, incluso más complejos. También permite cambiar formas, colores, posiciones, grados, entre otras cosas, de forma manual. Entre otras funciones, puede identificar componentes, grupos, hallar datos como el máximo flujo y el camino más corto, grafos isomorfos y generación aleatoria de grafos.

En las figuras 3.13 y 3.14 se pueden observar ejemplos de visualizaciones de grafos tomadas de la página web de Mathematica.

3.3.5. Tulip

Tulip es un framework dedicado al análisis y visualización de datos relacionados. Está escrito en lenguaje `C++`, lo que permite que sea un framework eficiente y con muy buenos resultados.

Implementa los algoritmos de visualización circular, por niveles y dirigido por fuerzas, pero solo la versión física. Aparte de esto, permite realizar agrupamientos, ordenamientos topológicos, coloreado y etiquetado.

Aparte de ser un framework, Tulip posee su propia interfaz gráfica, llamada Tulip Perspective, donde se puede interactuar manualmente con el grafo. Permite realizar importación de archivos en formato CSV con la descripción del grafo en cuestión, y exportar archivos en formato GML.

En la figura 3.15 se puede observar un ejemplo tomado de la página web de Tulip.

3.3.6. yEd Graph Editor

yEd Graph Editor es un software de escritorio gratuito multiplataforma que puede ser utilizado para generar todo tipo de diagramas basados en grafos. Estos diagramas pueden ser creados manualmente o importados desde archivos externos.

Aparte de todos los formatos básicos de diagramas UML [56], BPMN [57], entre otros, posee implementaciones de algoritmos de visualización por niveles y circular. Cabe destacar que implementa muchos de los algoritmos específicos para la visualización de árboles.

Los archivos externos que pueden importarse son bastante variados; acepta muchos formatos de modelado de grafos, tales como GML, y también formato de tablas como CVS y XLS [58]. El programa también permite exportar los despliegues en cualquier formato de imágen, PDF e incluso en el formato de Macromedia Flash [59].

En las figuras 3.16 y 3.17 se pueden ver ejemplos tomados de la página web de yEd Graph Editor.

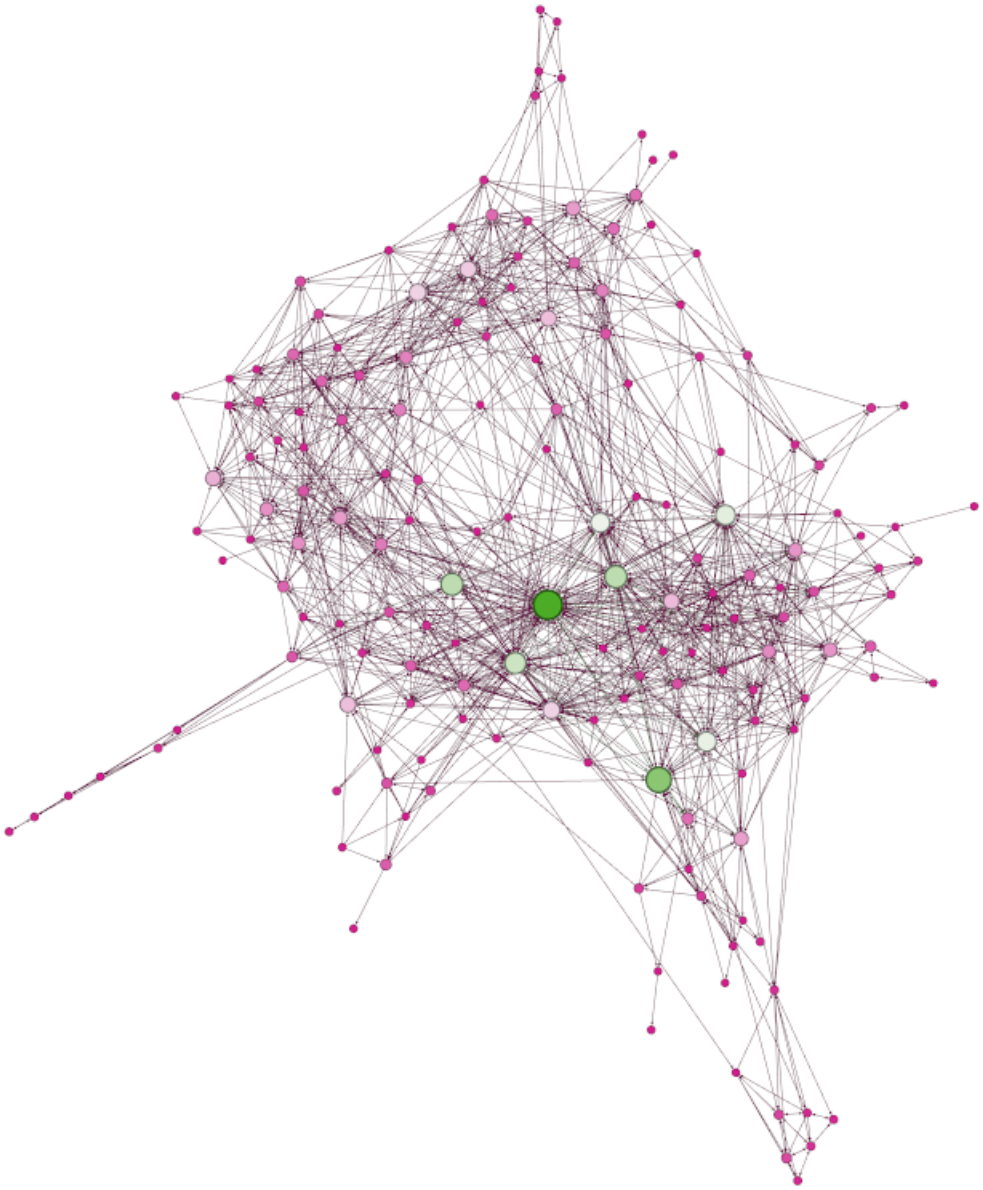


Figura 3.10: Ejemplo tomado de la página web de Gephi [49].

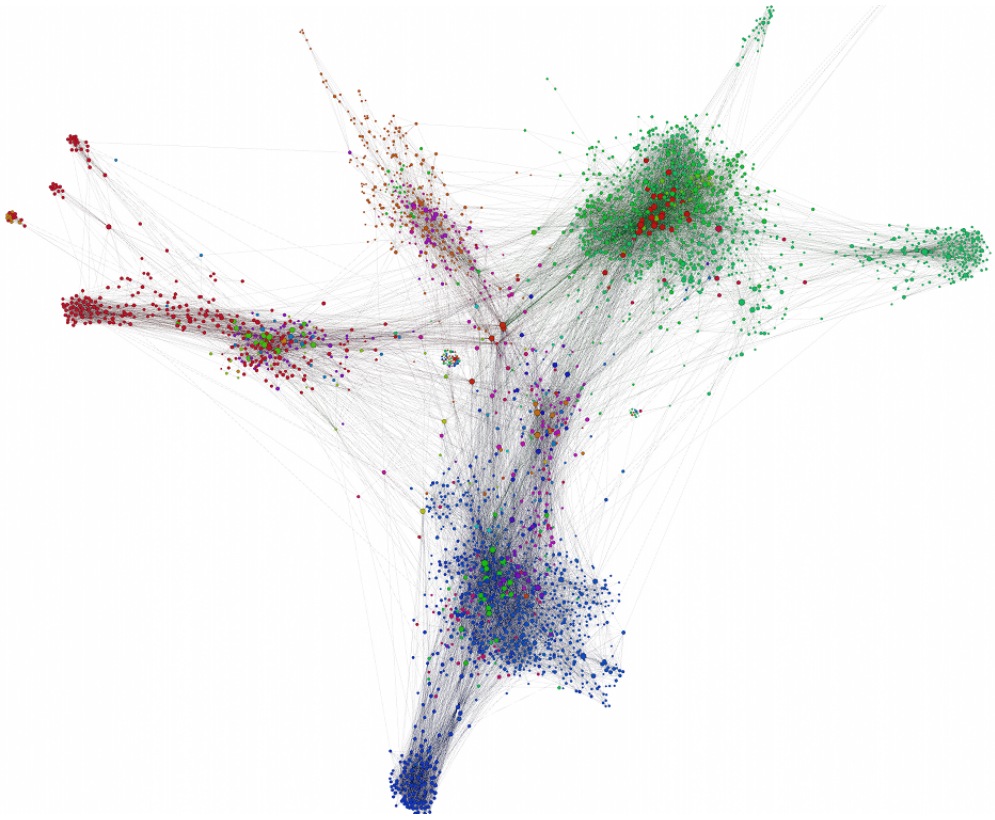


Figura 3.11: Ejemplo tomado de la página web de Gephi [49].

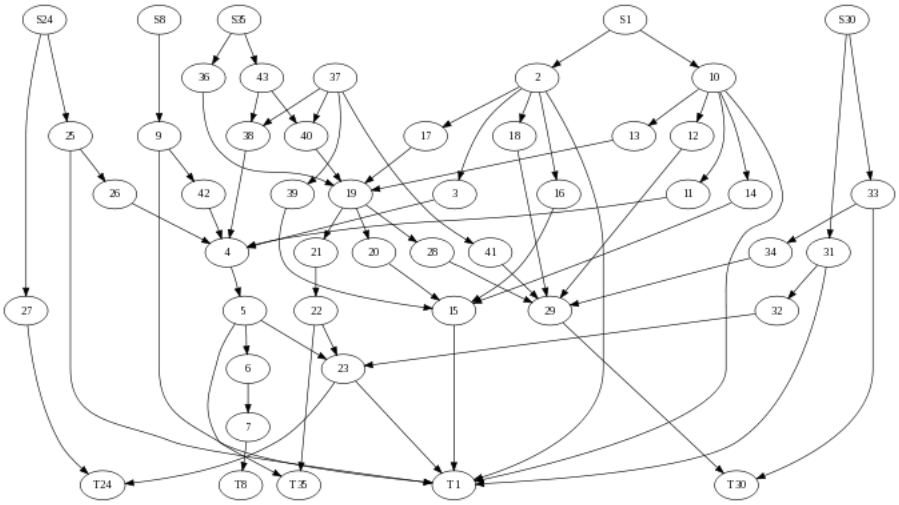


Figura 3.12: Ejemplo tomado de la página web de Graphviz [50].

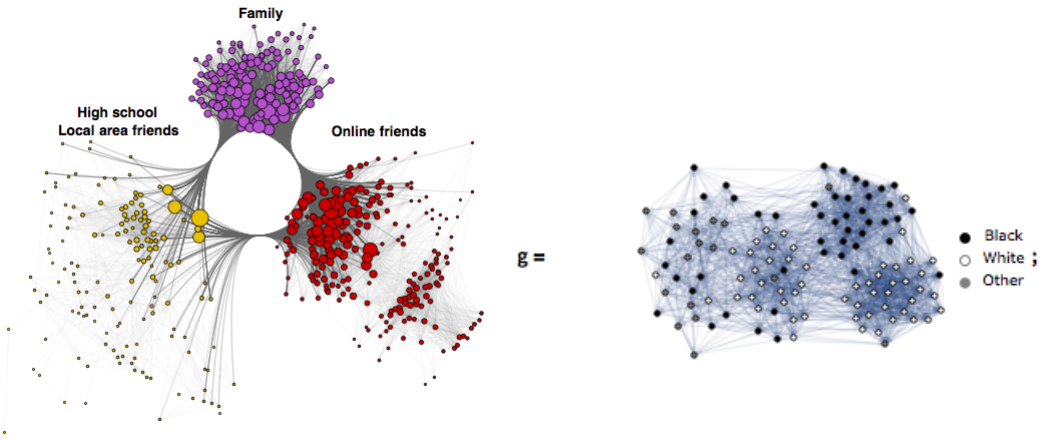


Figura 3.13: Ejemplos tomados de la página web de Mathematica [52].

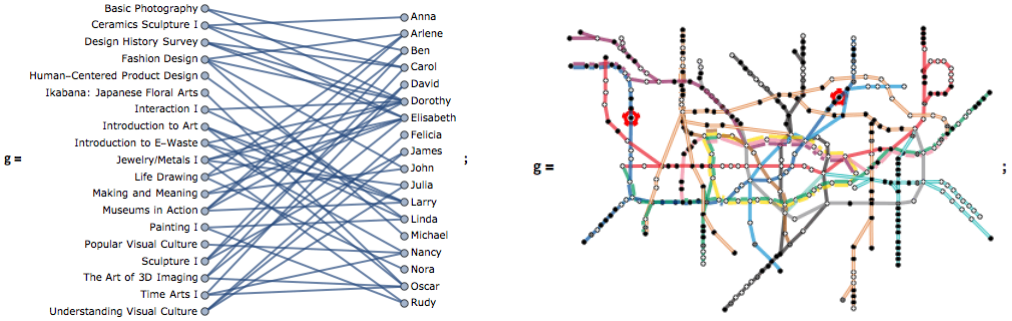


Figura 3.14: Ejemplos tomados de la página web de Mathematica [52].

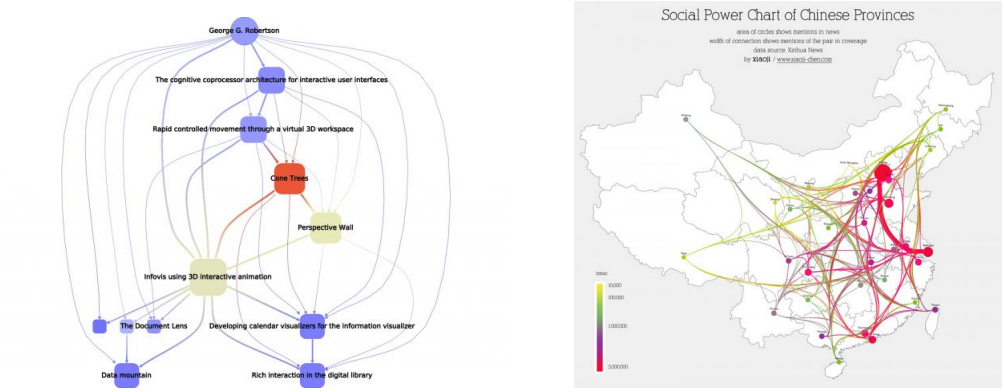


Figura 3.15: Ejemplos tomados de la página web de Tulip [53].

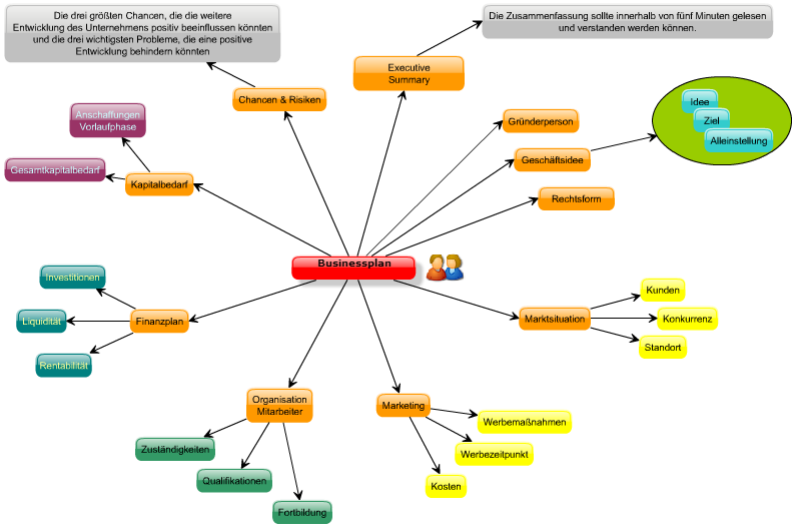


Figura 3.16: Ejemplos tomados de la página web de yEd Graph Editor [55].

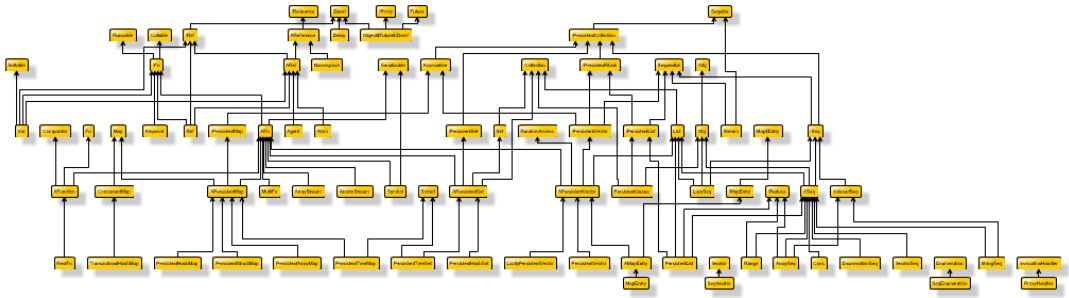


Figura 3.17: Ejemplos tomados de la página web de yEd Graph Editor [55].

Capítulo 4

Algoritmos genéticos

En el marco de la inteligencia artificial, un algoritmo genético consiste en una búsqueda heurística que imita, de alguna u otra forma, el proceso de selección natural [61]. Normalmente estos algoritmos se utilizan como alternativas para la resolución de problemas de optimización.

En un algóritmo genético hay dos elementos importantes a tomar en cuenta:

- La población: es un conjunto de individuos, los cuales serán tomados en cuenta para la selección natural. Cada uno de estos individuos es un candidato de solución, y debe ser posible mutarlo y alterarlo.
- La función objetivo: es la función que regularmente se desea optimizar. Recibe como entrada un individuo y retorna un valor cuantitativo asociado al mismo.

4.1. Operaciones genéticas

Las operaciones genéticas son aquellas que pueden aplicarse a cada individuo para modificar sus datos. Las operaciones genéticas conocidas son la operación de cruces, y la operación de mutación.

4.1.1. Cruce

Esta operación consiste en tomar dos individuos, y con algún criterio, combinarlos para formar dos nuevos individuos. Los criterios más conocidos son los siguientes:

- Cruce en un punto: consiste en realizar un corte en el mismo punto de los dos padres, y combinar la primera parte del primer padre con la segunda parte del segundo padre, y las otras dos partes restantes. El resultado son dos individuos que tienen ambos partes de los dos padres.
- Cruce en dos puntos: consiste en realizar dos cortes. Luego se toma el contenido de ambos padres que está entre los cortes y se intercambia, generando dos nuevos individuos.
- Corte y empalme: consiste en hacer un corte en cada padre, no necesariamente en el mismo sitio y aplicar el mismo procedimiento que en el cruce de un punto. Esto provoca que los hijos no tengan la misma longitud que los padres, lo cual no siempre es favorable.

4.1.2. Mutación

Esta operación consiste en tomar a un individuo y modificar sus datos de manera parcial o total. La mutación se utiliza para conservar la diversidad en los individuos de una generación a otra. En términos matemáticos, la mutación evita que el algoritmo llegue a un mínimo local, ya que realizando esta operación se evita que los individuos sean muy similares entre ellos. Las operaciones de mutación más conocidas son las siguientes:

- Mutación de un bit: solo puede utilizarse si los datos son representados de manera binaria. Consiste en tomar al individuo e invertir un bit de su cadena de bits.

- Mutación de inversión: solo puede utilizarse si los datos son representados de manera binaria. Consiste en invertir todos los bits de un individuo.

- Mutación de límite: solo puede utilizarse si los datos son enteros o reales. Consiste en reemplazar el menor o el mayor elemento de los datos del individuo.

4.2. Algoritmo

En principio, se tiene una población determinada que puede ser dada o generada aleatoriamente. Luego sigue un proceso iterativo donde la población de cada iteración se denomina generación; el proceso consiste en realizar ciertas operaciones sobre la población actual para producir la siguiente generación.

El algoritmo comienza con un proceso de inicialización, que consiste en llenar la población de la primera generación. Luego, en cada iteración, funciona de la siguiente manera:

- Se aplica un proceso de selección, que puede ser completamente aleatorio o basado en el valor de la función de cada individuo. Regularmente se seleccionan dos individuos de la población para ir al siguiente paso.

- Al tener los dos individuos seleccionados, con una probabilidad relativamente alta, se aplica el operador genético de cruce entre los mismos, produciendo como resultado dos individuos nuevos.

- Posteriormente, cada uno de estos individuos, con una probabilidad baja, pasa por un proceso de mutación, donde se altera al menos una porción de sus datos.

- Finalmente, estos dos nuevos individuos, mutados o no, pasan a formar parte de la nueva generación.

El algoritmo tiene un número definido de generaciones a evaluar. Al llegar a la última generación, el algoritmo se detiene, y el resultado final es la función evaluada en el mejor individuo de la última población.

4.3. Características de implementación

Además de todo lo descrito anteriormente, existen ciertos aspectos que deben tomarse en cuenta al momento de implementar un algoritmo genético. El primero de ellos es la representación de los datos. En un algoritmo genético, los datos se representan a través de cromosomas; estos pueden representarse con cadenas de bits, enteros o reales. Dependiendo de la representación varían las operaciones genéticas que deben aplicarse en todo el transcurso del algoritmo.

Otro aspecto a tomar en cuenta es el elitismo. Este consiste en tomar un conjunto pequeño de individuos muy buenos de una generación y moverlos automáticamente a la generación siguiente. Es importante el uso de este aspecto, porque garantiza que cada generación nunca tendrá una solución peor a la anterior.

Finalmente, existe una variante de los algoritmos genéticos que se caracterizan por ser adaptativos. El procedimiento es igual al de un algoritmo genético común, pero los valores de las probabilidades de cruces y mutación cambian de acuerdo a la población actual, y su objetivo es lograr que nunca se pierda la diversidad, pero que al mismo tiempo la solución converja a un valor relativamente bueno.

4.4. Ventajas y desventajas

La ventaja fundamental de los algoritmos genéticos es la capacidad de búsqueda que poseen. Abarcan un gran espacio sin ocupar tanto tiempo como una búsqueda por fuerza bruta, y a pesar de todas las desventajas que se mencionarán a continuación, un algoritmo capaz de resolver casi cualquier problema de optimización en un tiempo

aceptable es indudablemente bueno. Las desventajas que tienen estos algoritmos se basan en dos cosas, el tiempo de ejecución y la solución encontrada.

Las desventajas relacionadas al tiempo tienen que ver con el criterio que se utiliza para decir si es un tiempo aceptable o no. En comparación con otros algoritmos de optimización, los algoritmos genéticos realizan muchas evaluaciones de la función objetivo, lo cual puede llegar a ser catastrófico si la misma representa un tiempo de cálculo muy alto. Adicionalmente el espacio de búsqueda crece exponencialmente de acuerdo al crecimiento de los datos, por lo que el tiempo de ejecución también aumenta.

Por otro lado, las desventajas relacionadas a la solución encontrada son simples. Los algoritmos genéticos solo pueden conseguir una solución que es considerada buena en comparación al resto de soluciones previas, pero no necesariamente por eso es un buen resultado. Más aún, tienden por converger a soluciones consideradas como mínimos locales, lo cual tampoco es un resultado óptimo.

4.5. Aplicaciones

Los algoritmos genéticos tienen aplicaciones en cualquier campo donde se requiera optimizar algún valor o procedimiento. Se puede utilizar para el diseño de equipos y sistemas, para resolución de equilibrios en teoría de juegos, para realizar análisis lingüísticos, problemas de optimización numérica, diseño de redes de lógica difusa, entre otros.

Las aplicaciones más importantes son aquellas basadas en resolver problemas de optimización no lineales, tales como el problema del viajero [62] y el problema de la mochila [63].

Capítulo 5

Trabajos previos

A continuación, se presentarán datos y resultados de trabajos realizados previamente sobre este mismo tópico; papers escritos por universidades en China, Finlandia y Alemania. Se dividirá este capítulo en secciones del algoritmo donde estará la información correspondiente de cada paper a la sección en cuestión.

5.1. Representación de datos

Qing-Guo Zhang, Hua-Yong Liu, Wei Zhang y Ya-Jun Guo en 2005 [64], al igual que Jürgen Branke, Frank Bucher y Hartmut Schmeck en 1996 [65], plantean una representación que consiste en números reales. Considerando que el grafo tiene N vértices, y que los mismos son $(v_1, v_2, \ldots, v_n)$, pueden representarse como un vector de pares de coordenadas (x, y), dando como resultado un vector de $2N$ valores.

Por otro lado, Timo Eloranta y Erkki Mäkinen en 2001 [66] plantean el uso de una matriz de $2 \times N$, donde N es el número de vértices, y cada posición en la matriz es la posición del vértice en el plano entero donde ellos piensan desplegar el resultado.

5.2. Función a evaluar

En [64] se plantea una función que toma en cuenta la distancia entre todos los vértices, la distancia de los vértices adyacentes, la longitud ideal de las aristas, el

ángulo formado por las aristas en los vértices tomando en cuenta la proporción con el grado de los mismos, la cantidad de cruces y la simetría general del grafo. Todas estas medidas acompañadas por constantes que permiten la parametrización de la función para tomar de manera parcial algunos términos.

Por otro lado, en [65] se propone una medida que consiste en el algoritmo de visualización por fuerzas, la fuerza del resorte. Toman en cuenta la cantidad de cruces, la fuerza de los resortes, la variación de la longitud de las aristas y la distancia de los vértices a las aristas.

Finalmente, en [66] se plantea términos positivos y negativos, a diferencia de los dos anteriores. Los términos positivos son las distancias de los vértices con sus adyacentes y la suma de las mismas; mientras que los negativos son la diferencia de la longitud de las aristas con respecto a una longitud óptima y la cantidad de cruces.

5.3. Selección

En [64] se aplica una transformación basada en el promedio de los resultados de la función evaluada en cada individuo, junto con la desviación estándar de cada generación; esta transformación es utilizada para obtener solo a los mejores individuos para el cruce, manteniéndolos en la siguiente generación con elitismo, desechando a los que no son suficientemente buenos.

Similarmente lo hacen en [65] y en [66], pero no se aplica ninguna transformación, sino que se calcula directamente con el valor de la función de cada individuo.

5.4. Operadores genéticos

En [64] se utilizan cuatro operadores genéticos, uno de cruces y tres de mutación. Secuencialmente, primero se aplica el operador de cruces, que consiste en una operación de cruce en un punto. Posteriormente se aplica una mutación propia definida en el

paper, que consiste en alterar uno de los valores del vector con una fórmula basada en la cantidad de generaciones totales y actuales. Luego se aplica una mutación también definida en el paper que consiste en mover un vértice a una posición aleatoria en el círculo definido por un radio que va decreciendo tomando como centro el mismo vértice. Finalmente se aplica una inversión en un rango aleatorio.

En [65] no se especifican los operadores que se utilizan, pero si mencionan que utilizan un operador de cruces que evita la repetición de individuos utilizando operaciones básicas en el dibujo como la rotación y el desplazamiento.

Finalmente, en [66] se especifica que utilizan dos operadores de cruces junto con ocho diferentes operadores de mutación. Los operadores de cruces son definidos por ellos; el primero lo llaman "RectCrossover", que consiste en tomar un réctangulo en el dibujo de cada padre y e intercambiarlos para formar a los hijos. El segundo lo llaman "ThreeNodeCrossover", y consiste en tomar tres vértices e intercambiarlos justo como están visualizados para formar los dos hijos. Los operadores de mutación consisten básicamente en mover o intercambiar vértices y aristas de posición, variando la cantidad y la aleatoriedad.

5.5. Resultados

Los resultados de los tres papers son todos favorables. Cada uno de ellos compara sus resultados con algún otro algoritmo, o simplemente despliegan sus grafos visualizados. En [64] se compara su algoritmo genético modificado con uno común, destacando que el suyo lleva a un mejor resultado, no sólo en la función objetivo sino también en el despliegue del grafo.

Por otro lado, en [65] y en [66] solo se hace énfasis en la visualización generada, alegando que su algoritmo funciona bien para la mayoría de los casos y el grafo generado contiene la menor cantidad de cruces.

Capítulo 6

Implementación

Al ser una implementación web, la misma se divide en dos partes, el cliente y el servidor. La ventaja de esta implementación radica en dos características de este modelo: el servidor se encarga de las operaciones pesadas que no se hacen tan frecuentemente, y el cliente se encarga de las operaciones más frecuentes que no requieres tanto procesamiento. De esta manera se garantiza que la aplicación funcionará adecuadamente. Para esta implementación se utilizó una metodología ad hoc orientada a prototipos.

6.1. Cliente

Para la implementación del cliente se utilizaron las herramientas HTML5, CSS y Javascript, junto con las librerías de jQuery, Bootstrap y dos librerías para la exportación de imágenes. Es importante destacar que toda la comunicación, a excepción de la primera carga de la aplicación en el navegador, se realiza a través de Ajax [77].

6.1.1. Visualización

La visualización se realiza en el elemento canvas de HTML5, lo cual permite tener un control total de cada pixel del canvas por medio de Javascript. El mismo tiene un

tamaño dinámico al cargar la página; se ajusta de acuerdo a la resolución del monitor y al tamaño de la ventana del navegador.

Para los vértices y aristas se utilizan funciones incorporadas en javascript relacionadas con el elemento canvas; estas permiten el trazado de líneas rectas, círculos y rectángulos.

Adicionalmente se tienen dos pestañas en la interfaz del navegador que permiten la selección de forma y tamaño del vértice y el estilo de la arista, como se puede ver en la figura 6.1.

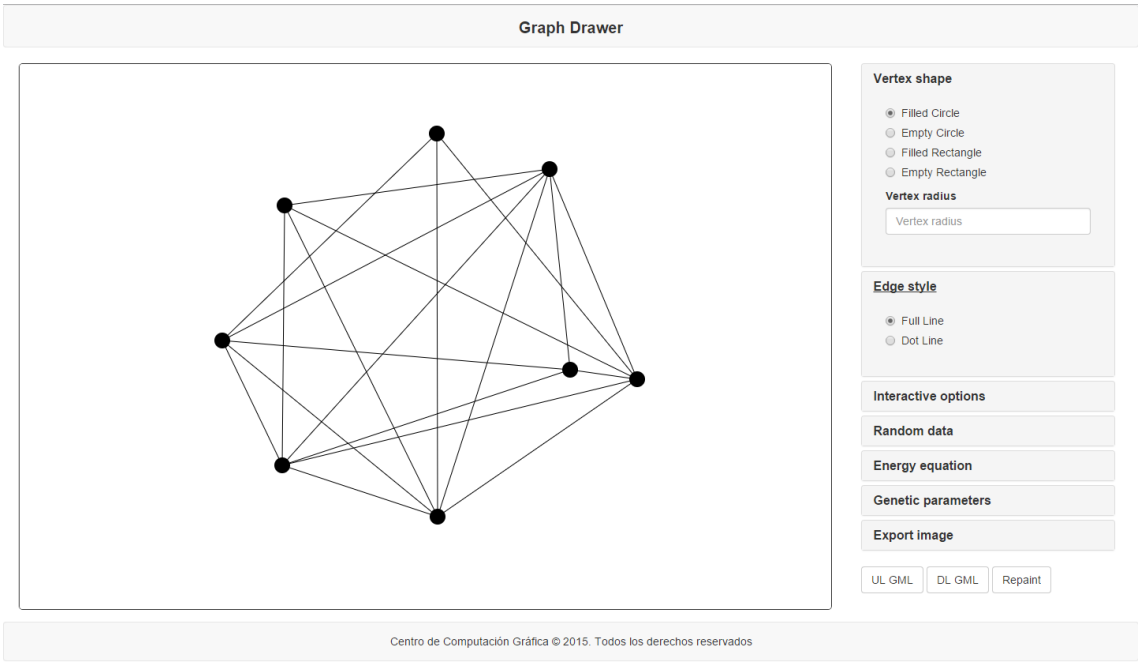


Figura 6.1: Interfaz gráfica de la aplicación donde se puede ver una visualización de un grafo y las pestañas de formas de vértices y estilos de aristas.

6.1.2. Interactividad

Por medio de funciones hechas en javascript también se permite que el canvas tenga interactividad, ya que el mismo tiene eventos que capturan cuando se presiona, se mueve o se deja de presionar el mouse; aprovechando estos eventos se implementaron

funcionalidades que permiten el movimiento de vértices en el canvas con el mouse y la adición, eliminación o cambio de color de vértices y aristas. Estas funcionalidades se agrupan todas en una sola pestaña, como se puede ver en la figura 6.2.

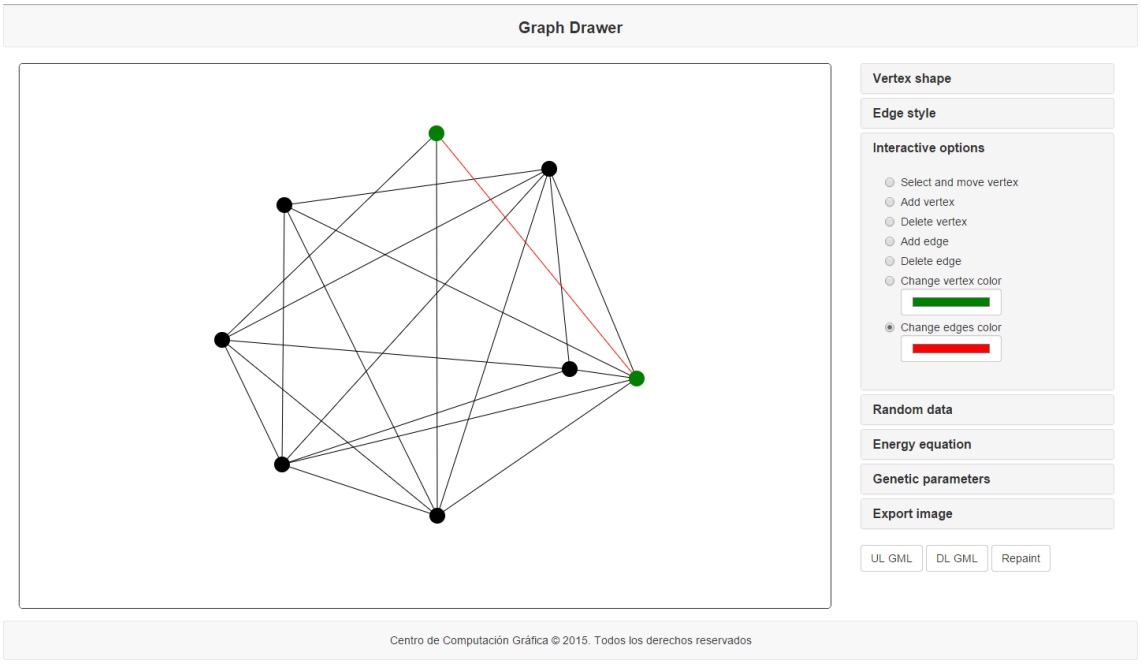


Figura 6.2: Interfaz gráfica de la aplicación donde se pueden ver las opciones interactivas.

6.1.3. Exportación de imágenes

La aplicación también permite exportar el contenido del canvar como una imágen de mapa de bits o idealmente vectorizada. Para la imagen de mapa de bits, el elemento canvas tiene una funcion incorporada que permite obtener un URL [78] para la descarga de la imágen en formato PNG [79].

La obtención de la imagen idealmente vectorizada no tiene una función incorporada, así que se realizó una solución alternativa utilizando las librerías de javascript "canvas2svg" [80] y "Base64" [81]; "canvas2svg" permite crear un SVG [82] desde cero y tratarlo como un canvas, por lo cual se debe dibujar nuevamente todo el conteni-

do del canvas en el SVG; posteriormente "Base64" permite codificar el SVG para la obtención de un URL del mismo.

Para que la descarga se realice de manera automática se utilizó el elemento `<a>` de HTML, con el atributo de referencia al URL obtenido, el atributo de descarga de archivo y disparando el evento de click de mouse una vez que se presionen los botones ubicados en la interfaz, como se puede ver en la figura 6.3.

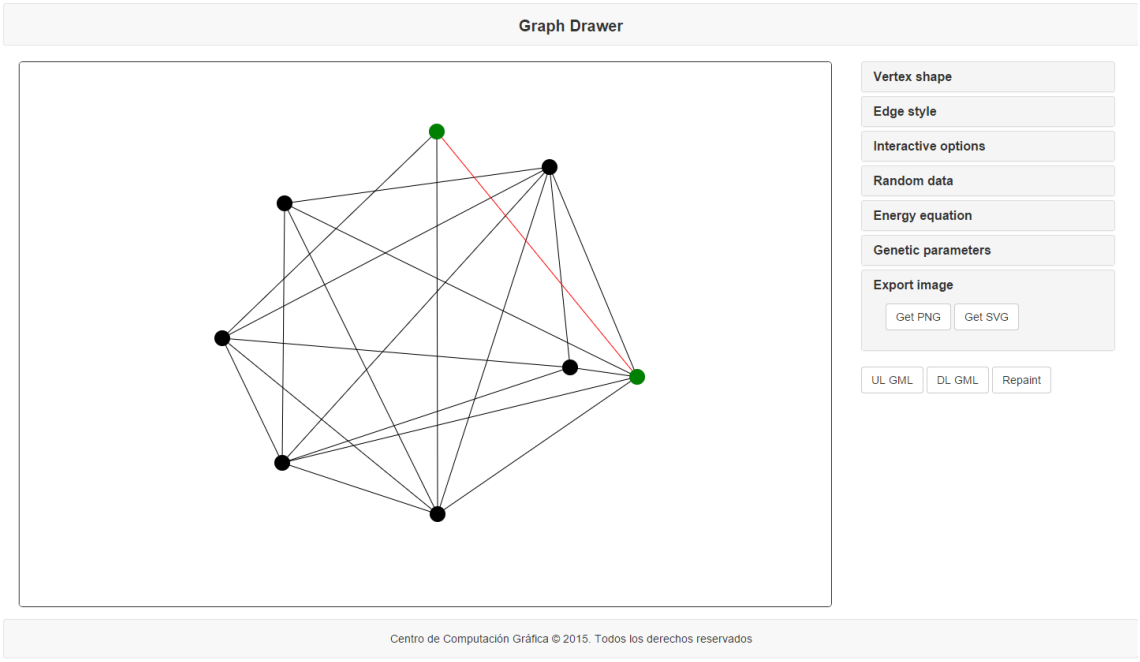


Figura 6.3: Interfaz gráfica de la aplicación donde se pueden ver los botones que permiten exportar la visualización como una imagen.

6.1.4. Carga y descarga de GML

La aplicación también permite cargar y descargar archivos en texto plano que tengan toda la configuración del grafo en cuestión. Para esto se utilizaron archivos GML, cuyo formato se explicó en el marco teórico.

La traducción y el armado del archivo GML se hacen del lado del servidor, pero la selección y la descarga del archivo se realizan del lado del cliente. Para la carga se

utiliza el selector por defecto del navegador para buscar el archivo GML, restringiendo la búsqueda a solo archivos en formato GML. Para la descarga, el servidor responde con un URL correspondiente al GML generado, y la descarga se realiza de manera similar a la descarga de imágenes.

El objetivo de tener estas funcionalidades es poder cargar grafos ya generados, o almacenar los mismos para futuras necesidades. En la figura 6.4 se pueden ver resaltados los botones que disparan estas funcionalidades.

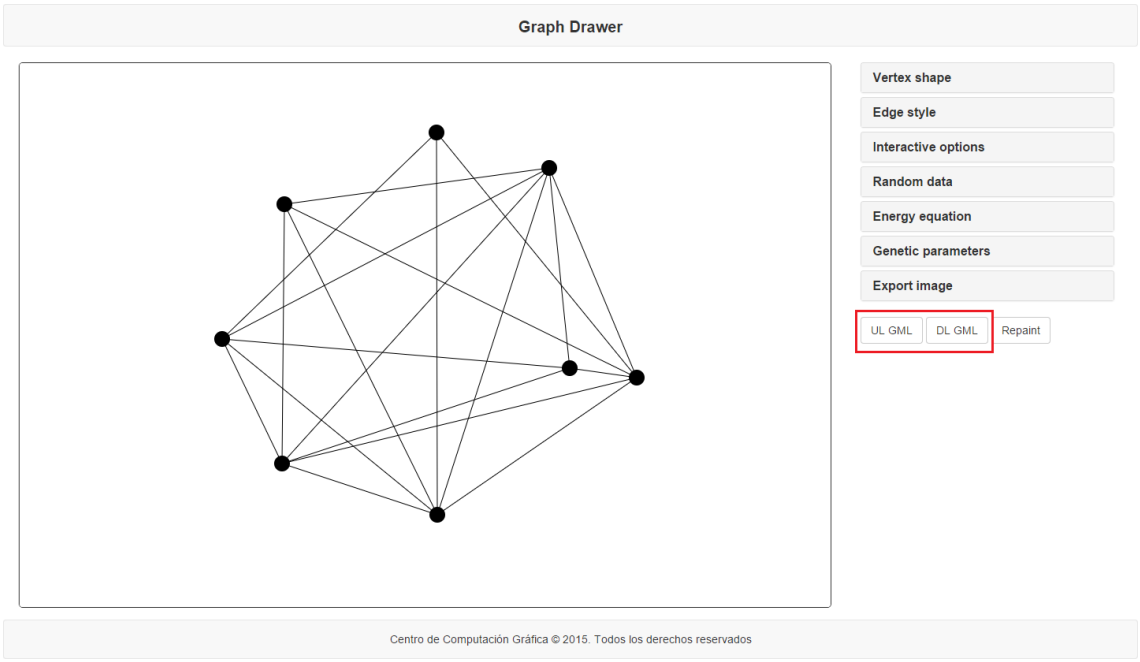


Figura 6.4: Interfaz gráfica de la aplicación donde se pueden ver los botones que permiten cargar o descargar archivos GML.

6.1.5. Otras funcionalidades

El resto de las funcionalidades en la interfaz no tienen ninguna implementación real en javascript sino que se implementan del lado del servidor; estas funcionalidades son la generación aleatoria de grafos, el cálculo de la ecuación de energía y la minimización de la misma a través de algoritmos genéticos.

A pesar de que el lado fuerte de todas estas funcionalidades está en el servidor, es importante mencionar que del lado del cliente es donde está la configuración de los parámetros necesarios para la ejecución de estos algoritmos.

Con respecto a los grafos aleatorios, en el cliente se permite ingresar la cantidad de vértices y aristas que se desea en el grafo y adicionalmente se incorporó un campo de máximo grado de cada vértice. El resultado puede desplegarse o puede descargarse en formato de entrada estándar para problemas de programación competitiva; asimismo, la aplicación también permite la carga de archivos en este formato, similar a la funcionalidad de carga y descarga de GML. En la figura 6.5 se puede ver la pestaña que se encarga de dicha funcionalidad.

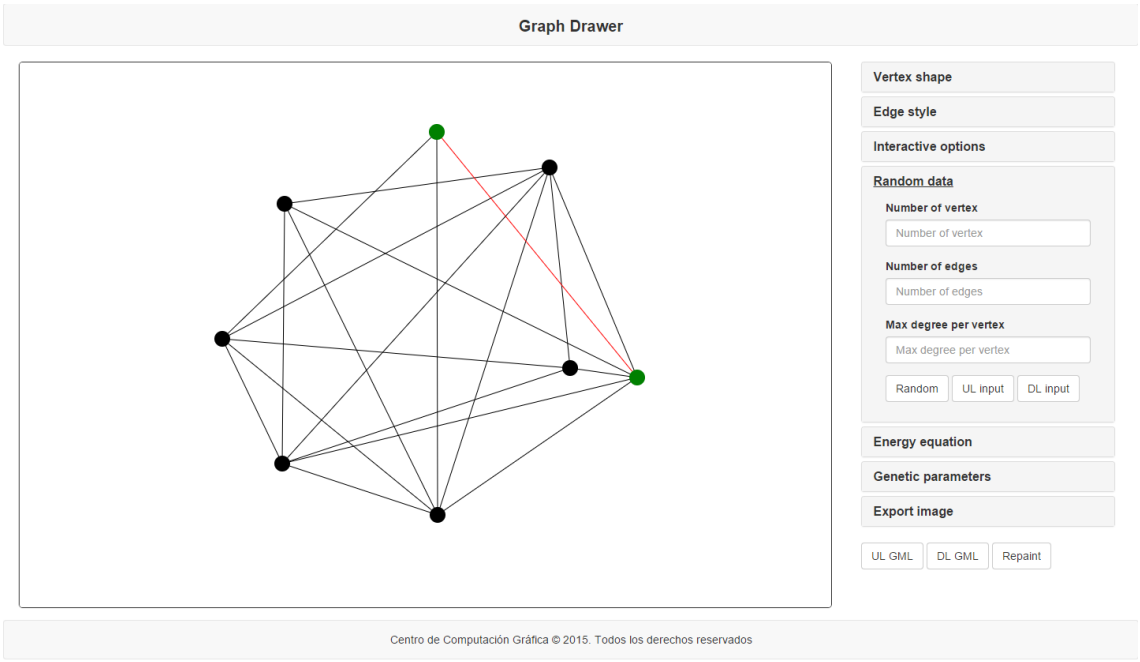


Figura 6.5: Interfaz gráfica de la aplicación donde se pueden ver los campos para los parámetros de la generación aleatoria de grafo y los botones para cargar y descargar grafos en formato básico de programación competitiva.

Finalmente, para las funcionalidades de ecuaciones de energía y parametros del algoritmo genético solo se tienen una serie de parámetros configurables más un botón que hace la petición con toda esta información al servidor. Los parámetros de la

ecuación de energía tienen que ver con su definición, la cual se coloca a continuación:

$$f(x) = a \cdot Cross(x) + b \cdot Area(x) + c \cdot Symmetry(x) + d \cdot Angle(x) \tag{6.1}$$

Donde $Cross(x)$ es la función que calcula la cantidad total de cruces, $Area(x)$ calcula el área total del grafo, $Symmetry(x)$ calcula la simetría del grafo y $Angle(x)$ calcula el menor ángulo formado por dos aristas que inciden en el mismo vértice. Las constantes (a, b, c, d) son las que definen el peso de cada una de estas funciones, y son las que se pueden ingresar a través de la interfaz, como se puede ver en la figura 6.6.

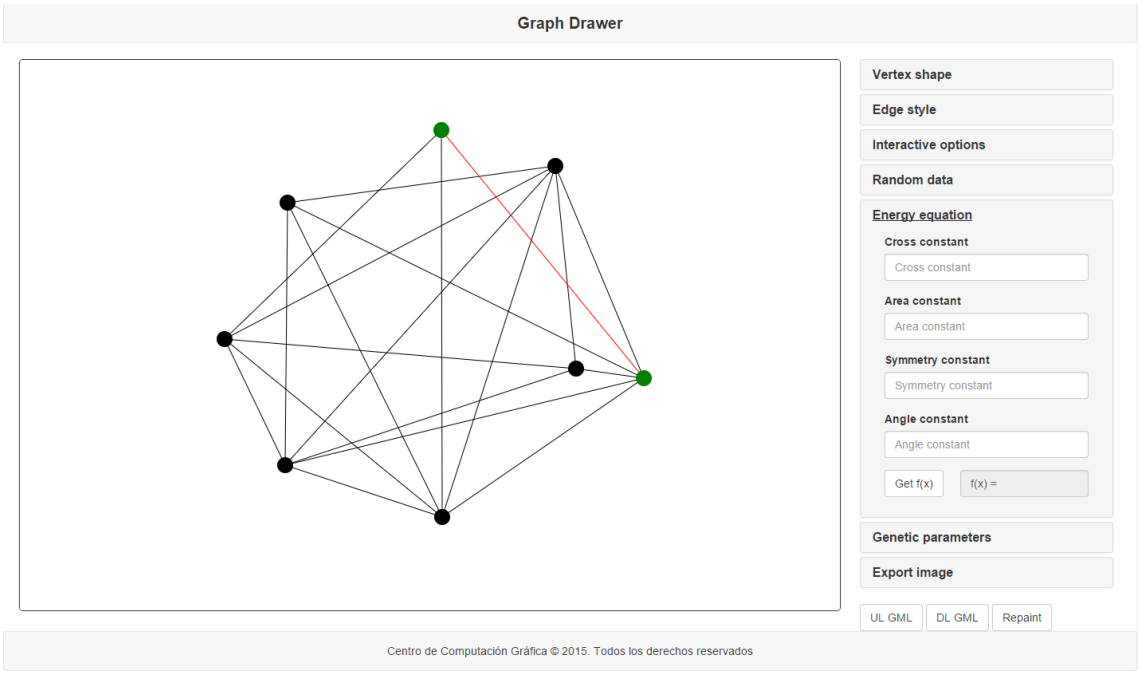


Figura 6.6: Interfaz gráfica de la aplicación donde se pueden ver los campos para los parámetros de la ecuación de energía.

Igualmente, los algoritmos genéticos reciben parámetros como el tamaño de la población, la cantidad de generaciones y las probabilidades de cruce y mutación, los cuales se ingresan a través de la interfaz, como se puede ver en la figura 6.7.

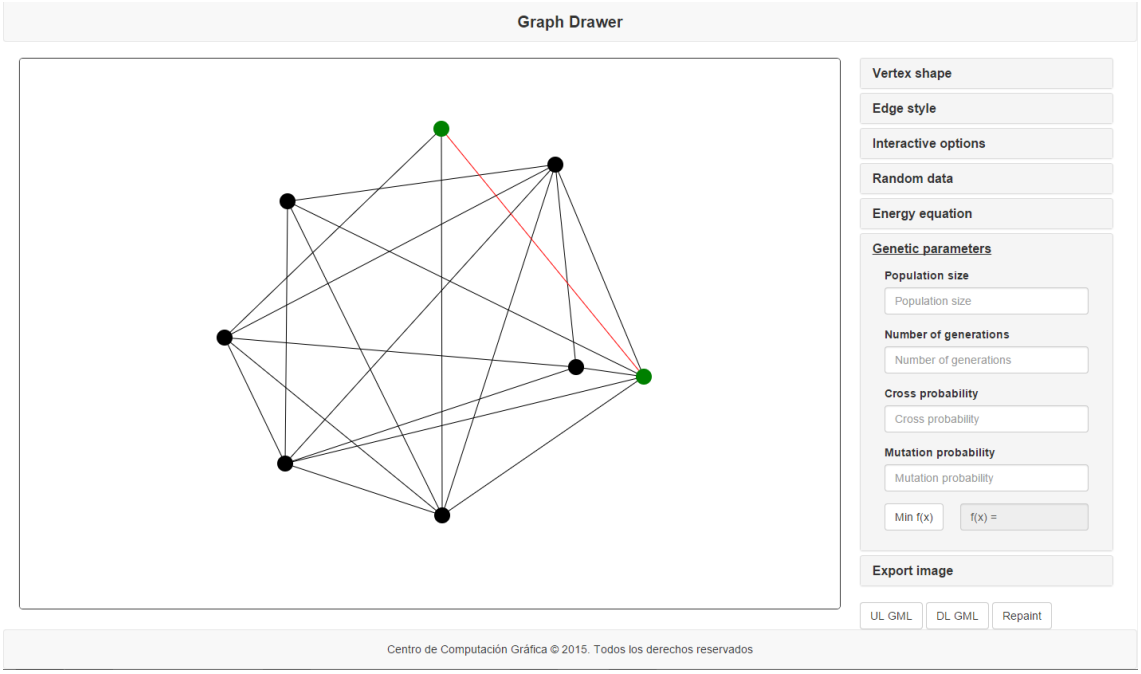


Figura 6.7: Interfaz gráfica de la aplicación donde se pueden ver los campos para los parámetros del algoritmo genético.

6.2. Servidor

Para la implementación del servidor se utilizó el framework web Django, el cual se encarga de todas las peticiones que vienen de los clientes. Se utilizaron dos librerías adicionales a las que vienen por defecto con Python y Django: "simplejson" y "networkx"; ambas se instalan utilizando la herramienta "pip" de Python. La primera se utiliza para la creación del JSON [83] que se debe enviar al cliente con toda la información que solicita y la segunda se utiliza para traducir el GML recibido, o bien, para armar uno nuevo.

6.2.1. Traducción y creación de GML

Al momento de recibir un GML, el cliente envía únicamente el contenido, no el archivo, y la librería "networkx" guarda en un diccionario toda la información que

pudo conseguir dentro de ese GML. Luego se procede a crear el JSON con toda la información del grafo para enviarlo al cliente; el mismo contendrá la siguiente información:

- La cantidad de vértices y aristas en el grafo.
- Todas las posiciones de los vértices. Si las mismas no se especifican en la sección "graphics" del GML, se generan posiciones aleatorias utilizando las medidas actuales del canvas.
- Todas las aristas indicando los dos vértices en los que incide cada una en formato 1-index.

Para armar un GML nuevo al momento de solicitar la descarga no se requiere utilizar la librería, solo se utilizan las operaciones básicas de escritura de archivo de Python. Se reciben todas las posiciones de los vértices y todas las aristas por parte del cliente y se imprimen en un archivo, siguiendo el formato correspondiente a GML. Luego se autoriza al cliente a descargar el archivo generado.

6.2.2. Grafos aleatorios

Para la generación de grafos aleatorios se utilizan las funciones aleatorias incorporadas de Python, las cuales producen números reales con 53 bits de precisión y tienen un período de $(2^{19937} - 1)$ [84].

Se tienen dos tipos de peticiones: las peticiones de grafos aleatorios para la visualización, y la generación de un grafo aleatorio en un formato de entrada para programación competitiva. Para la primera se reciben los parámetros del cliente y se hace un preprocesamiento sobre los mismos para evitar errores; estos consisten en chequear que la cantidad de nodos ingresada soporte la cantidad de aristas, teniendo en cuenta el grado máximo de cada vértice. Luego se asignan N coordenadas enteras aleatorias,

que serán las posiciones de cada vértice en el grafo, siendo N el parámetro ingresado como la cantidad total de vértices. Seguidamente se generan las aristas, y se utiliza un arreglo de conjuntos ordenados para controlar que las aristas no se repitan o que ningún vértice sobrepase el grado máximo. Finalmente se genera un JSON con toda esta información y se envía al cliente para su posterior visualización.

El segundo tipo de petición varía un poco del primero; lo que se debe generar ahora no es un JSON, sino un archivo descargable con toda la información del grafo. Este archivo se genera del lado del servidor y se autoriza al cliente para que sea descargado. Cabe destacar que al no requerir dibujar este grafo, el mismo no posee coordenadas físicas para los vértices.

El formato de los archivos de entrada de programación competitiva consiste en una línea que contiene dos enteros, N y M, que representan la cantidad total de vértices y aristas, respectivamente. Seguidamente se tienen M líneas con dos enteros cada una v_i y v_j, indicando que el vértice i contiene una arista que lo conecta a vértice j. Los índices de los vértices de este formato son basados en 0.

Para poder cargar un archivo exitosamente en la aplicación por medio de la petición de carga de archivos de programación competitiva, este formato debe ser respetado estrictamente, en caso contrario el resultado no será el esperado.

6.2.3. Ecuación de energía

La ecuación de energía utilizada es la descrita en la sección 7.1.5, la cual se basa en los criterios de calidad de cantidad de cruces, area total del grafo, simetría del grafo y los ángulos formados por las aristas de un mismo vértice.

6.2.3.1. Cruces

En esta función se calculó la cantidad de cruces que hay en el grafo tomando en cuenta todos los elementos, es decir, si las aristas se intersectan entre ellas, si las aristas

intersectan vértices, o si los vértices se intersectan entre ellos. Para esto se utilizaron tres funciones diferentes:

- Intersección Segmento-Segmento: utilizada para la intersección entre aristas, esta función consiste en hacer dos verificaciones, que los puntos extremos del primer segmento esten uno a cada lado del segundo segmento y que los puntos extremos del segundo segmento esten uno a cada lado del primer segmento. Esto se realiza por medio de una función basada en el producto cruz entre vectores que determina la dirección de un punto con respecto a otros dos.

  $$ISS = \left|\{(A,B)/A,B \in E \wedge ccw(A,B_p) \neq ccw(A,B_q) \wedge ccw(B,A_p) \neq ccw(B,A_q)\}\right| \tag{6.2}$$

  Donde E es el conjunto de aristas convertidas en segmentos, A y B son segmentos de ese conjunto, p y q son los puntos extremos de cada segmento y ccw es la función que calcula la orientación del punto con respecto al segmento.

- Intersección Segmento-Círculo: utilizada para la intersección de aristas y vértices, esta función consiste en hacer la proyección del centro del círculo en el segmento en cuestión, calcular la distancia del punto original con esa proyección y finalmente comparar esa distancia con el radio del círculo.

  $$ISC = \left|\{(A,c)/A \in E \wedge c \in C \wedge ||\overrightarrow{\rho_{Ac}c}|| \leq r\}\right| \tag{6.3}$$

  Donde E es el conjunto de las aristas convertidas en segmentos, C es el conjunto de las coordenadas de los vértices, ρ_{Ac} es la proyección de c en A y r es el radio del círculo.

- Intersección Círculo-Círculo: utilizada para la intersección entre vértices, esta función es la más sencilla de todas, ya que solo consiste en calcular la distancia de cada par de centros de vértices y verificar si son menores a dos veces el radio de los círculos.

  $$ICC = \left|\{(i,j)/i,j \in V \wedge ||\overrightarrow{ij}|| \leq 2r\}\right| \tag{6.4}$$

  Donde V es el conjunto de coordenadas de los vértices.

Finalmente, la función de cruces queda de la siguiente manera:

$$Cross = ISS + ISC + ICC \tag{6.5}$$

6.2.3.2. Area

La función de área toma las coordenadas de todos los vértices y encuentra el rectángulo de menor área que cubre todos los puntos de los vértices. Esto es posible haciendo un recorrido por todas las coordenadas de los vértices y almacenando el menor y el mayor valor de cada eje. Finalmente se retorna la multiplicación de las restas del mayor y el menor en cada eje.

$$Area = \Big(\max_{i\in[0,n-1]} x_i - \min_{i\in[0,n-1]} x_i\Big)\Big(\max_{i\in[0,n-1]} y_i - \min_{i\in[0,n-1]} y_i\Big) \tag{6.6}$$

6.2.3.3. Simetría

Solo se tomó en cuenta la simetría horizontal y con respecto a cantidad, es decir, la función implementada haya un punto medio en el eje x, tomando la mitad entre el máximo y el mínimo valor, y retorna la diferencia entre la cantidad de vértices a la izquierda de ese punto y la cantidad de vértices a la derecha de ese punto.

$$Symmetry = \Big|\big|\{t/t \in X \wedge t < m\}\big| - \big|\{t/t \in X \wedge t > m\}\big|\Big| \tag{6.7}$$

Donde m es el punto medio mencionado y X es el conjunto de todos los valores en el eje x.

6.2.3.4. Ángulo entre aristas

Para el cálculo de los ángulos se consideró a las aristas como vectores; se restaron las coordenadas del vértice destino con las del vértice de origen para obtener los vectores. Para hallar el ángulo entre cada par de vectores se utilizó la desigualdad de

Cauchy-Schwarz ajustada, conocida popularmente como la definición del ángulo entre dos vectores. Se toma el menor ángulo obtenido y se le hace el complemento con 180, porque la función objetivo original se está minimizando.

$$Angle = 180 - \min_{\substack{i \in [0, n-1] \\ u, v \in V_i}} \arccos\left(\frac{u^t v}{||u|| \cdot ||v||}\right) \tag{6.8}$$

Donde V_i es el conjunto de vectores formados a partir de aristas cuyo origen es el vértice i.

6.2.4. Algoritmos genéticos

Los algoritmos genéticos son, sin lugar a dudas, la funcionalidad más importante de todo el trabajo de investigación. La implementación toma todos los parámetros del cliente, ejecuta el algoritmo, almacena estadísticas de cada generación, y finalmente genera un JSON para enviarlo al cliente con el resultado final.

6.2.4.1. Representación

Se utilizó una representación binaria de 18 bits, donde 9 de ellos están asignados al eje x y el resto al eje y, es decir, el algoritmo puede ubicar los vértices únicamente en el rango $[0, 511]$. Para formar al individuo se toman todas las posiciones de los vértices y se convierten a la representación binaria, dando como resultado un arreglo. Posteriormente, este arreglo pasa a formar parte, junto con muchos otros arreglos, de la generación actual.

Para iniciar el algoritmo se generan tantos individuos aleatorios como lo indique el parámetro recibido del cliente.

6.2.4.2. Selección y operadores genéticos

Para la selección de individuos se tienen dos implementaciones, la primera consiste en darle igual prioridad a cada individuo, y la segunda hace que la probabilidad de

seleccionar al individuo i sea inversamente proporcional a la función objetivo en ese individuo.

El operador genético de cruce utilizado fue el cruce en un punto. Se toman dos individuos aleatoriamente y con una probabilidad p_{cruce} los individuos se cortan en un punto y se intercambian sus partes. Por otro lado, el operador de mutación utilizado fue la mutación de un bit; a estos individuos, cruzados o no, se les invertirá exactamente un bit aleatorio con probabilidad $p_{mutacion}$ para cada uno.

A manera de ejemplo, se asume una representación de 3 bits, con dos vértices en total. Se asume que el primer individuo representa vértices en las posiciones $(2, 1)$ y $(3, 3)$, y que el segundo representa $(4, 1)$ y $(2, 5)$. Según la representación planteada, teniendo en cuenta que son solo 3 bits para el ejemplo, los individuos serían los siguientes:

Individuo	x_1	y_1	x_2	y_2
1	010	001	011	011
2	100	001	010	101

En total, cada individuo tiene 12 bits. Ahora para realizar la operación de cruce se selecciona un bit aleatorio para el corte, en este caso se toma el bit 4.

Individuo	Parte 1	Parte 2
1	0100	01011011
2	1000	01010101

Luego, la primera parte del individuo 1 junto con la segunda parte del individuo 2 pasan a ser el individuo 3, mientras que la primera parte del individuo 2 junto con la segunda parte del individuo 1 pasan a ser el individuo 4.

Individuo	x_1	y_1	x_2	y_2
3	010	001	010	101
4	100	001	011	011

Los individuos 3 y 4 pasan a la siguiente generación, pero antes se puede aplicar una mutación a cada uno. Se supone que aleatoriamente solo el individuo 4 es victima de una mutación, y se asume que el bit a mutar es el 3. La tabla quedaría de la siguiente forma:

Individuo	x_1	y_1	x_2	y_2
3	010	001	010	101
4	100	101	011	011

Al final, el individuo 3 representa las posiciones $(2, 1)$ y $(2, 5)$, mientras que el 4 representa las posiciones $(4, 5)$ y $(3, 3)$.

6.2.4.3. Elitismo

Luego de llenar la siguiente generación de individuos, se toman los mejores N individuos de la generación actual y se pasan a la siguiente, desplazando los peores N que ya estén presentes. La cantidad exacta de individuos a tomar es variable, pero se establecerá un buen valor relativo en el siguiente capítulo.

6.2.4.4. Identificador de sesión

Por medio de las herramientas que provee Django, es posible asignar un identificador único a cada cliente que acceda al servidor, almacenándolo como un dato de sesión. Cada vez que el cliente envía una petición al servidor, si ya tiene asignado un identificador de sesión, entonces la petición irá debidamente identificada, sino, se le asignará un identificador. Debido a que no es posible hacer ninguna petición sin antes hacer la petición estándar de cargar la página principal, solo se incluyó la asignación de identificadores en esa vista.

El objetivo de asignar identificadores únicos a cada cliente es poder realizar actualizaciones de estado con respecto al algoritmo genético. Al momento de comenzar

la ejecución del algoritmo genético, el cliente automáticamente hará una petición Ajax cada dos segundos para ser informado de cuantas generaciones lleva el algoritmo genético en ese momento; esto se hace para proveer una respuesta constante al usuario y mejorar así la usabilidad del sistema.

Además de esto, la aplicación provee un botón para cancelar la ejecución del algoritmo genético, el cual realiza una petición Ajax, al igual que la consulta, que envía un parámetro de cancelación al servidor; este, al momento de terminar la generación en curso, se detiene y envía los resultados actuales.

Capítulo 7

Pruebas

En este capítulo se incluirán tres secciones. La primera consiste en las pruebas visuales sobre los parámetros de la ecuación de energía para determinar aquellos que produzcan una visualización aceptable. La segunda sección consiste en medir la relación que hay entre el tiempo de ejecución y el resultado del algoritmo genético, variando la población y cantidad de generaciones. Y la tercera sección consiste en probar los demás parámetros del algoritmo genético, teniendo en cuenta que se harán pruebas variando la selección y el elitismo. Las pruebas se realizaron en una máquina con un procesador Intel Core i7-3770, con 8GB de memoria RAM, en Windows 8 Pro de 64 bits, utilizando la versión 1.7.4 de Django y la versión 3.4.2 de Python.

7.1. Ecuación de energía

En esta sección se establecerán los parámetros correctos en la ecuación de energía para la obtención de un grafo "aceptable". Para estas pruebas se utilizó la semilla $10^9 + 7$ para el generador de números pseudo aleatorios.

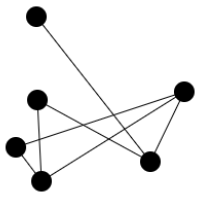
7.1.1. Cruces

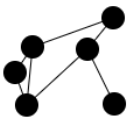
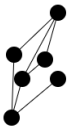
Primero se probó la constante de la medida de cruces; para estas pruebas se asumió que las demás constantes eran $(10^{-4}, 1, 10^{-3})$, para las medidas de área, simetría,

y el mínimo ángulo, respectivamente. Con respecto a los parámetros del algoritmo genético, se asumió un tamaño de población de 100 individuos, junto con 100 generaciones, probabilidad de cruce de 0.8 y probabilidad de mutación de 0.1; además, el proceso de selección es aleatorio y se aplica elitismo con 5 individuos. Estas constantes predefinidas son deducidas de todas las pruebas no oficiales hechas durante el desarrollo.

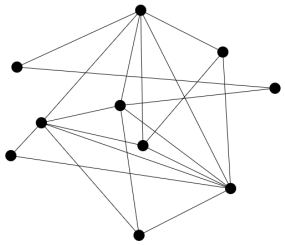
Para definir la constante de cruces se utilizarán tres grafos como base, y cada uno de ellos será redibujado 3 veces variando el parámetro correspondiente a los cruces.

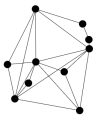
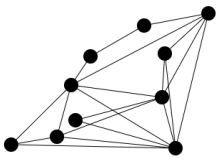
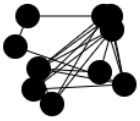
Grafo 1



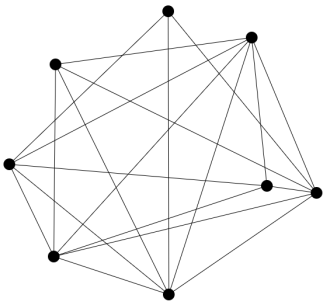
Prueba	Constante cruce	Resultado	Visualización
1	1	0.8093830479278058	
2	0.5	0.9031364316680686	
3	0	0.377307435758775	

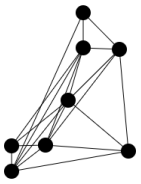
Grafo 2



Prueba	Constante cruce	Resultado	Visualización
4	1	9.342931620602865	
5	0.5	7.3645398864087905	
6	0	0.8193842328267968	

Grafo 3



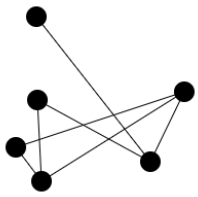
Prueba	Constante cruce	Resultado	Visualización
7	1	12.5369	
8	0.5	8.4609078848755	
9	0	0.8583938724449387	

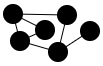
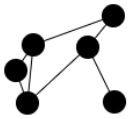
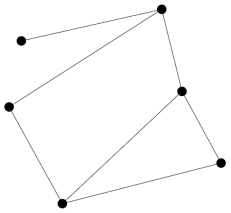
Observando estos resultado se puede establecer que el parámetro adecuado para la constante de cruces es 1, ya que en las pruebas que se hicieron con 0.5 se puede ver que hay igual o más cruces en el grafo; se descarta por completo el parámetro 0 por generar grafos que no son estéticos ni entendibles.

7.1.2. Área

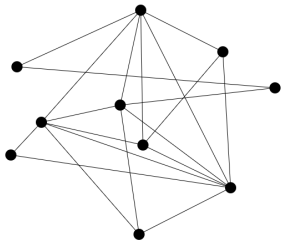
Para las pruebas del área se tomarán las constantes $(1, 1, 0.001)$, para las medidas de cruces, simetría y el mínimo ángulo, respectivamente. Se utilizarán los mismos parámetros de los algoritmos genéticos y los mismos grafos que se utilizaron en las pruebas anteriores.

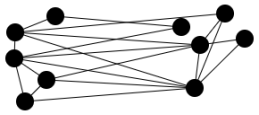
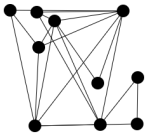
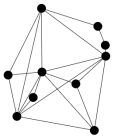
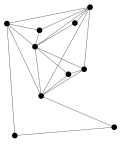
Grafo 1



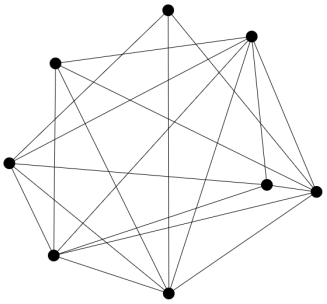
Prueba	Constante área	Resultado	Visualización
10	10^{-3}	3.8559829540246633	
11	$5 \cdot 10^{-4}$	1.6031061667268498	
12	10^{-4}	0.8093830479278058	
13	0	0.10454848358137228	

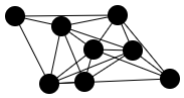
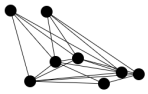
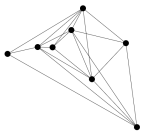
Grafo 2



Prueba	Constante área	Resultado	Visualización
14	10^{-3}	32.45782511930607	
15	$5 \cdot 10^{-4}$	29.67083559272386	
16	10^{-4}	9.342931620602865	
17	0	2.1693747885000927	

Grafo 3



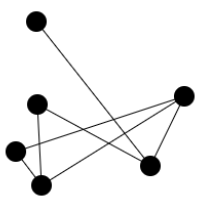
Prueba	Constante área	Resultado	Visualización
18	10^{-3}	17.651115456107206	
19	$5 \cdot 10^{-4}$	24.632203856847504	
20	10^{-4}	12.5369	
21	0	4.171753624759303	

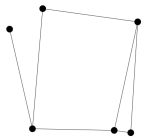
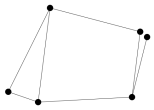
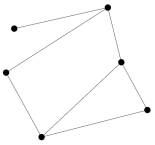
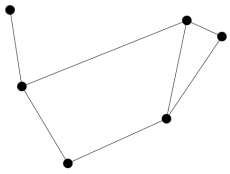
Se puede observar que agregar cualquier tipo de restricción basado en el área del grafo solo ocasiona que el mismo se comprima y genere cruces innecesarios. En las pruebas realizadas el parámetro 0 genera menos cruces que cualquier otro parámetro; por lo tanto, queda establecido el 0 como la constante de área para las demás pruebas.

7.1.3. Mínimo ángulo

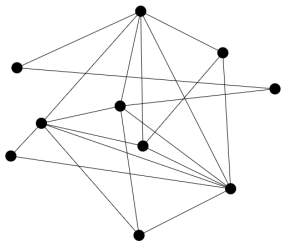
Para las pruebas del mínimo ángulo se tomarán las constantes $(1, 0, 1)$, para las medidas de cruce, área y simetría, respectivamente. Se utilizarán los mismos parámetros de los algoritmos genéticos y los mismos grafos que se utilizaron en las pruebas anteriores.

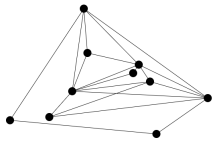
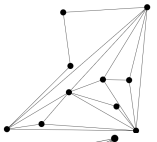
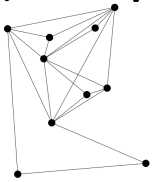
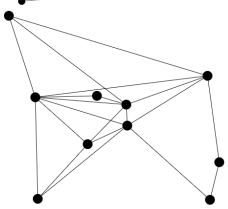
Grafo 1



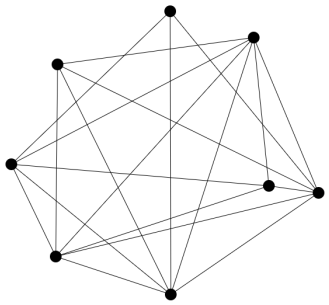
Prueba	Constante ángulo	Resultado	Visualización
22	0.5	47.191314112723376	
23	0.1	10.196032330326304	
24	$0.001 \sim 0.01$	0.8093830479278058	
25	0	0	

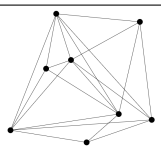
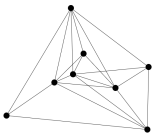
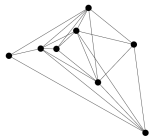
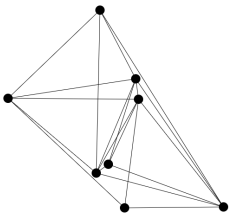
Grafo 2



Prueba	Constante ángulo	Resultado	Visualización
26	0.5	86.28038985096362	
27	0.1	17.665994820104302	
28	$0.001 \sim 0.01$	0.8093830479278058	
29	0	2	

Grafo 3



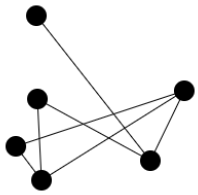
Prueba	Constante ángulo	Resultado	Visualización
30	0.5	92.10848230737945	
31	0.1	20.627415071606176	
32	$0.001 \sim 0.01$	0.8093830479278058	
33	0	7	

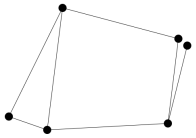
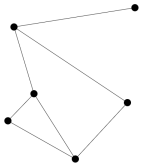
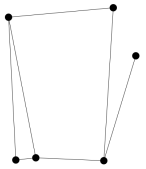
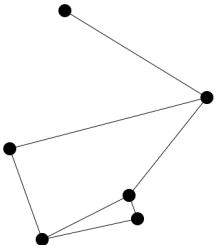
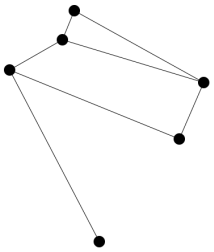
Se puede observar que el mínimo ángulo no puede estar cercano a 1 ni tampoco cercano a 0. El parámetro 0.1 fue el que generó mejores resultados visuales en los tres grafos, a pesar de que genere una ecuación de mayor valor por la manera en que está medido el ángulo.

7.1.4. Simetría

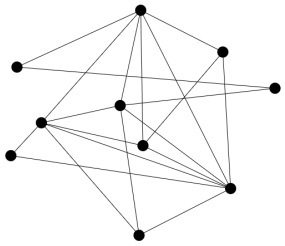
Para las pruebas de la simetría se tomarán las constantes $(1, 0, 0.1)$, para las medidas de cruces, área y el mínimo ángulo, respectivamente. Se utilizarán los mismos parámetros de los algoritmos genéticos y los mismos grafos que se utilizaron en las pruebas anteriores.

Grafo 1



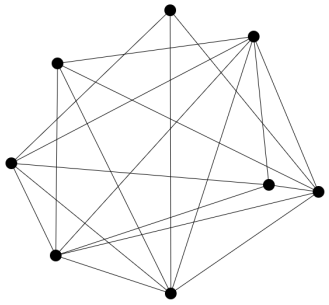
Prueba	Constante simetría	Resultado	Visualización
34	1	10.196032330326304	
35	0.75	10.489071797371793	
36	0.5	9.583704291218835	
37	0.25	9.75742599642217	
38	0	9.671484543026018	

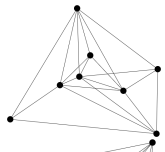
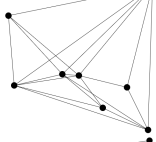
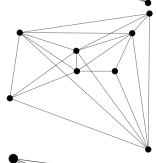
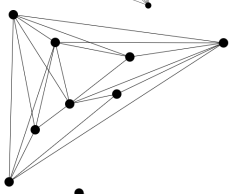
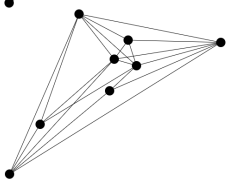
Grafo 2



Prueba	Constante simetría	Resultado	Visualización
39	1	17.665994820104302	
40	0.75	19.91772954810569	
41	0.5	18.887006199753667	
42	0.25	19.5583176448229	
43	0	20.162011954800708	

Grafo 3



Prueba	Constante simetría	Resultado	Visualización
44	1	20.627415071606176	
45	0.75	23.30194244794978	
46	0.5	23.171061055267256	
47	0.25	21.769323210359	
48	0	21.223700762257717	

Ningún parámetro generó resultados mejores que el 1; todos son iguales o peores, en el sentido estético y teniendo en cuenta el número de cruces.

Considerando la última observación, los parámetros quedan fijados a 1 para la constante de cruces, 0 para la constante de área, 1 para la constante de simetría y 0.1 para la constante de mínimo ángulo.

7.2. Población y generaciones

En esta sección se probarán diferentes valores para la población y la cantidad de generaciones. Los valores a probar son 10, 25, 50, 100 y 200 para cada uno. Se mostrarán una gráfica y se realizará un análisis sobre los mejores parámetros tomando en cuenta el tiempo de procesador y el valor obtenido por el algoritmo genético.

Los parámetros a utilizar son los obtenidos en la sección anterior con respecto a la ecuación de energía; la probabilidad de cruce se establecerá en 0.8, la de mutación en 0.1, se utilizará selección simple y elitismo de 5 individuos.

Los resultados a presentar son resultados promediados de 20 pruebas con 20 semillas diferentes para la generación de números aleatorios en Python; se utilizarán los mismos 3 grafos utilizados en la sección anterior. Las semillas utilizadas son las siguientes:

Semillas				
8929	10099	10343	11059	11443
11777	12011	22039	22123	22307
32029	32909	34147	35083	36161
47599	48589	48889	88997	$10^9 + 7$

Las pruebas realizadas al grafo 1 indican que el algoritmo tarda más en dar resultados a medida que se aumenta el número de generaciones y la población. Si comparamos ambos gráficos, se puede descartar por completo el uso de más de 100 generaciones y más de 100 individuos por población, ya que los resultados que generan no varían lo

suficiente como para que valga la pena el tiempo que debe invertirse para calcularlos. El valor de la función objetivo y el tiempo se pueden observar en las tablas a continuación, y en sus gráficos asociados en las figuras 7.1 para la función objetivo y 7.2 para el tiempo de procesador.

Función objetivo - Grafo 1

	Generaciones				
Individuos	10	25	50	100	200
10	15.0334701	14.3991488	13.6956677	12.7515910	12.1332745
25	13.5196201	12.5151847	11.7768916	11.2169157	10.8513547
50	13.3118229	11.5586228	10.6251423	10.0115355	9.8714631
100	12.8346397	11.3848210	10.3628184	9.9014408	9.7291574
200	12.3760456	11.3151319	10.5171606	9.8047944	9.6309667

Tiempo de procesador - Grafo 1

	Generaciones				
Individuos	10	25	50	100	200
10	1.1853534	2.7776942	4.6671404	9.9422332	20.7435551
25	1.1163029	2.8947863	5.8772777	12.3334996	25.2502558
50	1.4356095	3.4518241	6.4640831	12.9749374	26.7909944
100	1.4672074	3.6084713	7.0123739	15.4251738	29.9662541
200	1.8212363	4.4236654	9.2090936	17.6318772	35.5148747

A continuación, se pueden ver los valores de la función objetivo y el tiempo de procesador de las pruebas realizadas en el grafo 2. En las figuras 7.3 y 7.4 se pueden observar los gráficos asociados a estos resultados.

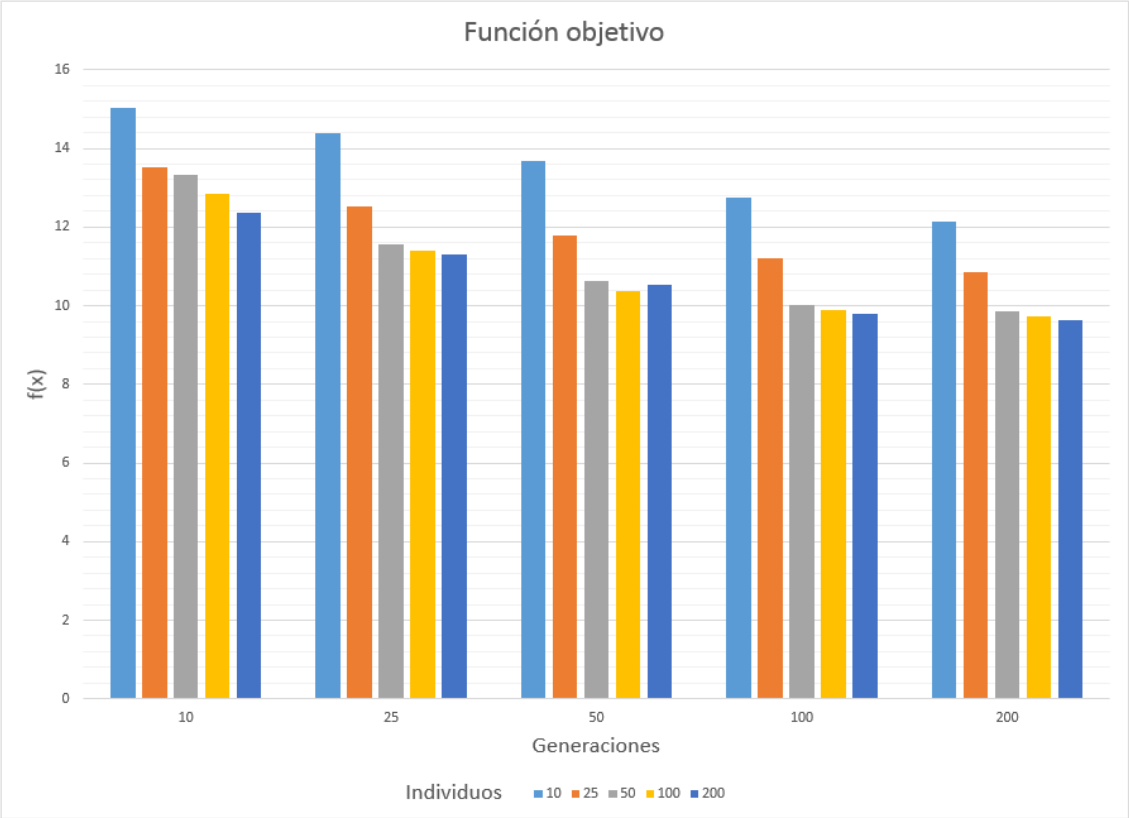


Figura 7.1: Gráfico asociado a la tabla de función objetivo del grafo 1.

Función objetivo - Grafo 2

	Generaciones				
Individuos	10	25	50	100	200
10	30.9615778	29.3732956	27.9618975	26.1005440	24.5655602
25	27.8342140	24.8546270	23.5886154	22.4100117	21.3214324
50	27.4491540	23.7750658	21.8989083	21.1378332	20.7006668
100	25.3745768	23.3361444	20.8779895	19.9775059	19.3863700
200	25.6057674	23.2812472	21.2472937	19.6486927	18.5511024

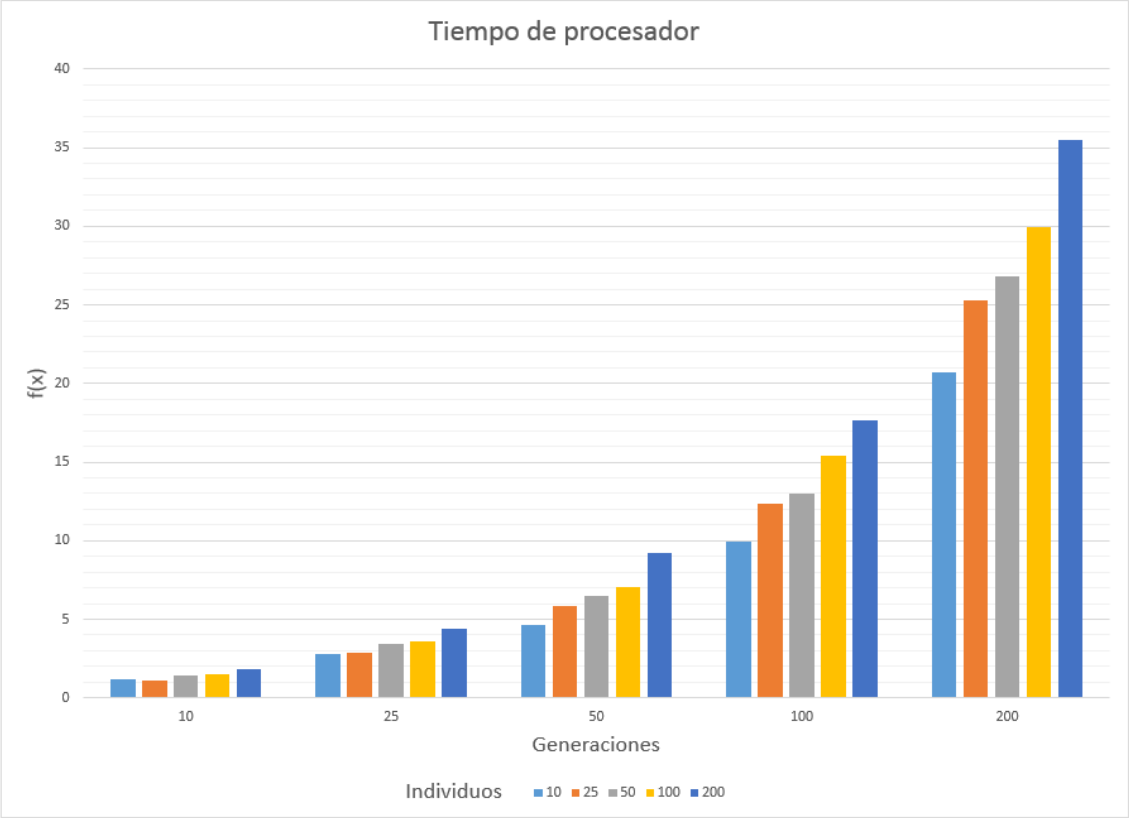


Figura 7.2: Gráfico asociado a la tabla del tiempo del grafo 1.

Tiempo de procesador - Grafo 2

	Generaciones				
Individuos	10	25	50	100	200
10	1.1807433	3.6301745	6.8044645	10.4325631	20.4222371
25	1.3110098	3.4493332	7.0069684	14.2781887	29.0126851
50	1.8652458	4.5610537	9.3092950	18.3133051	37.3762075
100	2.6733345	6.5192860	11.9171070	25.1059329	48.8647815
200	4.1539662	9.8876876	19.5842890	38.7140036	77.3488390

Finalizando las pruebas de población y generaciones, se tienen los resultados de los valores de la función objetivo y el tiempo de procesador de las pruebas realizadas

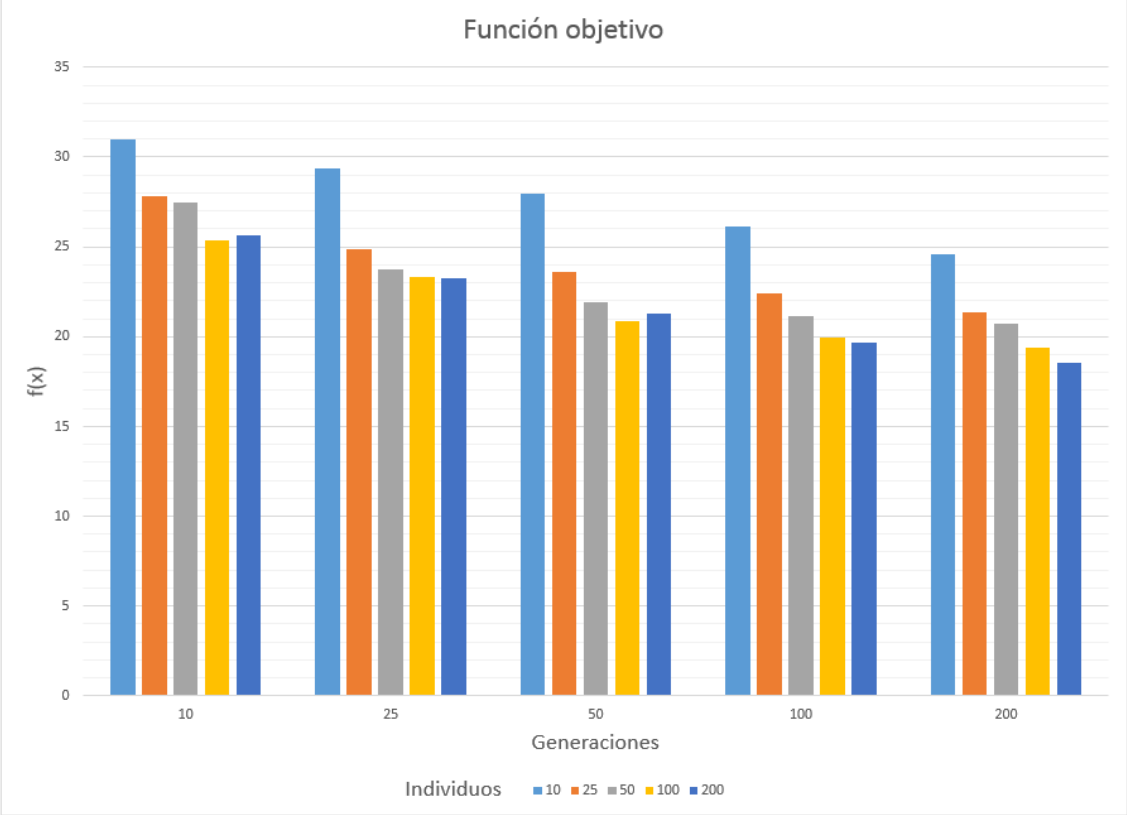


Figura 7.3: Gráfico asociado a la tabla de función objetivo del grafo 2.

en el grafo 3. En las figuras 7.5 y 7.6 se pueden observar los gráficos asociados a los mismos.

Función objetivo - Grafo 3

	Generaciones				
Individuos	10	25	50	100	200
10	31.1423031	29.7214594	28.8566348	27.1577945	26.1967336
25	28.0402221	26.4788500	25.1942719	24.3686808	23.8019450
50	27.8665140	25.1854233	23.9270777	23.4598691	23.0297831
100	27.9388411	25.1118420	22.9572425	22.0652997	21.6070447
200	26.9228177	25.1258874	23.3894872	21.9326780	21.4838336

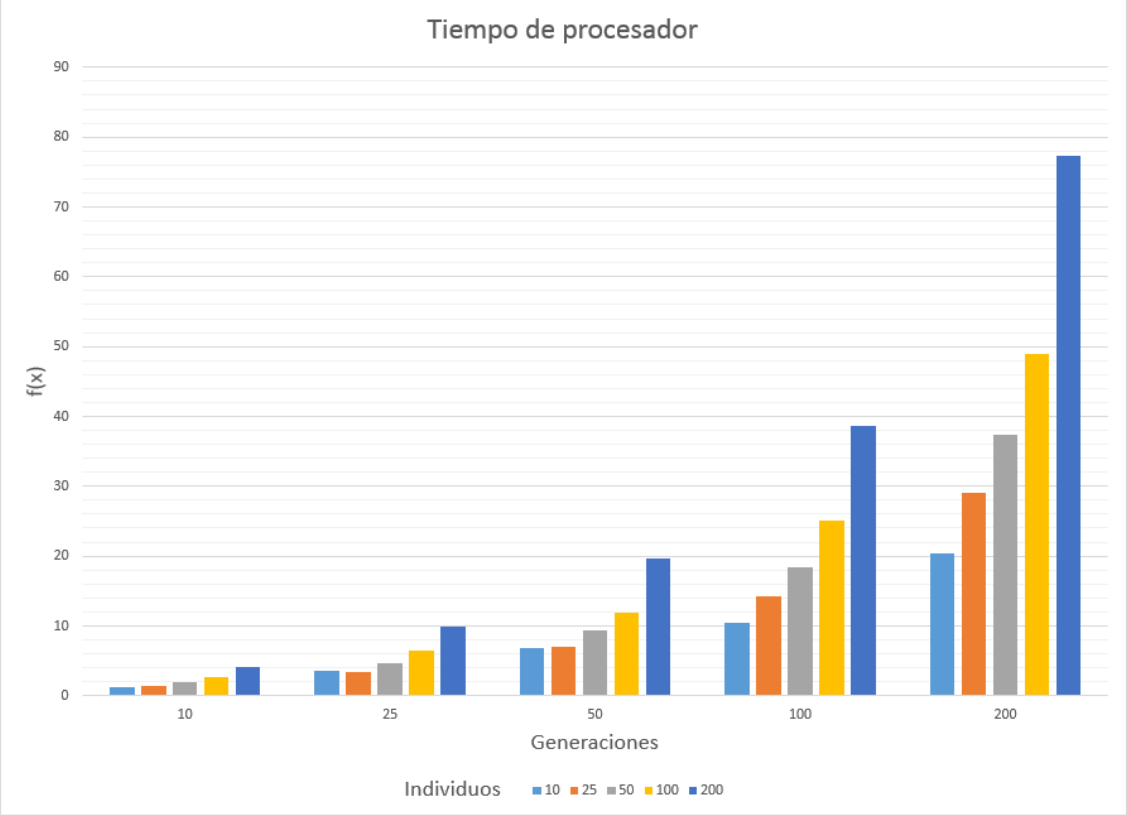


Figura 7.4: Gráfico asociado a la tabla del tiempo del grafo 2.

Tiempo de procesador - Grafo 3

	Generaciones				
Individuos	10	25	50	100	200
10	1.2302578	3.2576009	6.2735556	12.1246274	25.4268963
25	1.5263485	3.4513048	6.9820428	13.7806443	27.6138841
50	1.6321956	3.9420886	7.9104141	16.0749650	33.1267736
100	2.2842924	5.5265231	10.8212160	22.6230964	46.9545410
200	3.9242565	9.5202410	18.6695004	36.6967640	73.3905782

En la opinión de autor, y con base en los resultados obtenidos, los mejores parámetros para la población y las generaciones son 100 individuos por población y 50 gene-

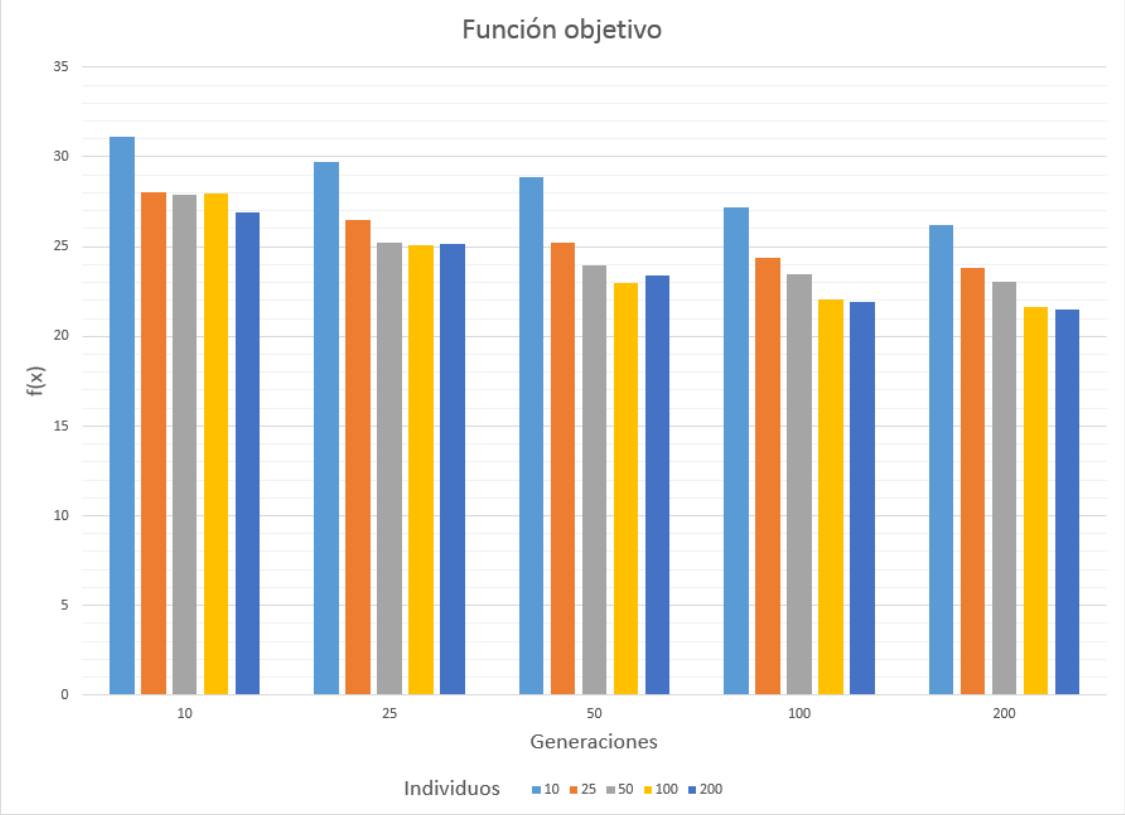


Figura 7.5: Gráfico asociado a la tabla de función objetivo del grafo 3.

raciones. El resultado no es mejor que utilizando más generaciones, pero el tiempo de ejecución es considerablemente menor.

7.3. Probabilidades, selección y elitismo

Se realizaron cuatro pruebas automatizadas para probar los diferentes métodos de selección combinados con la cantidad de individuos considerados por el elitismo. Cada prueba consiste en ejecutar el algoritmo variando la probabilidad de mutación entre 0.1 y 0.2 y la probabilidad de cruce entre 0.6 y 0.8. Los resultados a presentar de cada prueba son todos los valores finales de la función objetivo, además de presentar la variación de los promedios y la desviación estándar. Para estas pruebas se utilizaron las mismas semillas de las pruebas anteriores, junto con los mismos tres gafos.

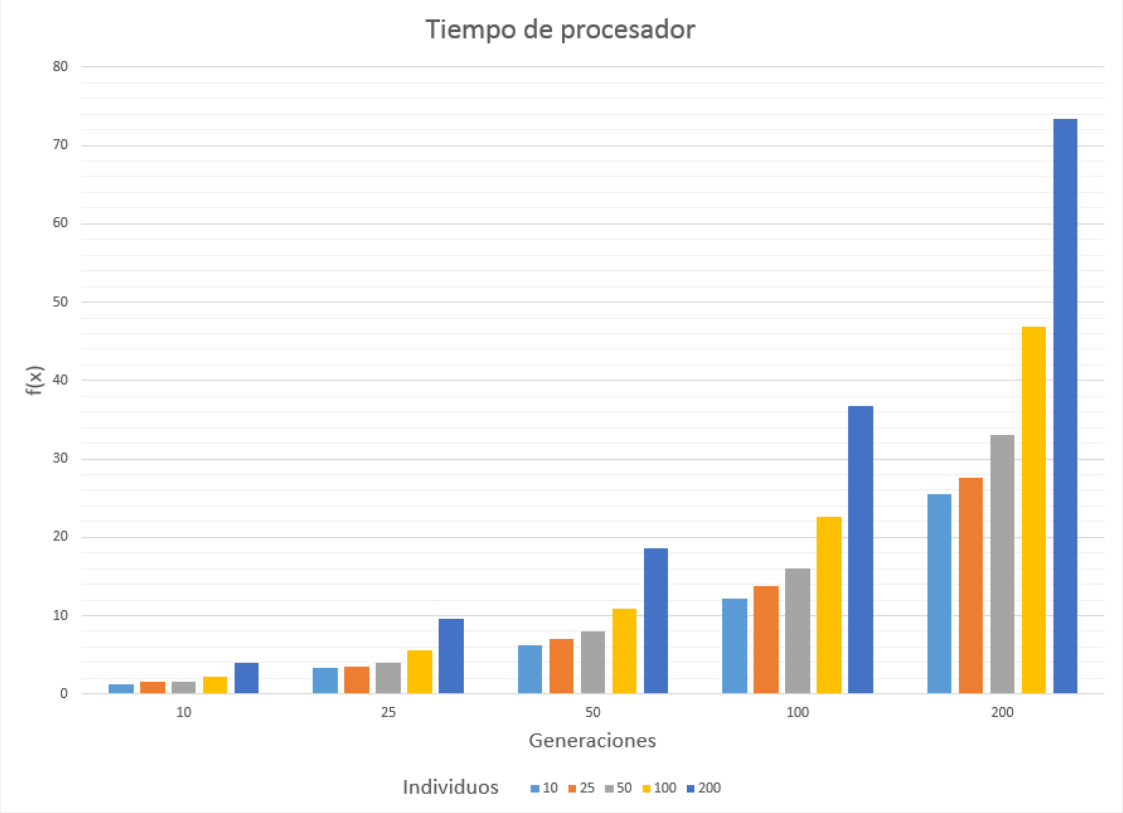


Figura 7.6: Gráfico asociado a la tabla del tiempo del grafo 3.

7.3.1. Elitismo 1 - Selección simple

Utilizando elitismo de un individuo y selección simple, la cual consiste en tomar individuos de forma aleatoria, se llegó a los siguientes resultados:

Elitismo 1 - Selección simple - Grafo 1

	Prob de cruce				
Prob de mutación	0.6	0.65	0.7	0.75	0.8
0.1	12.0807551	11.9554824	11.8899574	11.490713	11.5251883
0.125	12.3055947	12.0369719	12.1510471	11.6378546	11.5402252
0.15	12.8687327	12.3690426	11.9333216	12.4257357	11.9389435
0.175	12.6920758	12.0077237	12.3010998	11.8838943	12.2663565
0.2	12.3402154	12.3919826	12.2221695	12.4566209	11.8502424

Con un promedio que varía entre 20.0831243 y 20.5805026, y una desviación que varía entre 2.6060725 y 2.8807923.

Elitismo 1 - Selección simple - Grafo 2

	Prob de cruce				
Prob de mutación	0.6	0.65	0.7	0.75	0.8
0.1	24.9939784	25.0877267	24.1165404	23.8072301	24.3791653
0.125	24.877184	24.3686995	24.3139343	23.9010467	24.5711317
0.15	24.7369863	24.7111748	24.7341424	24.4771973	24.0288869
0.175	23.5273254	24.5036543	24.255049	23.8143477	24.5573478
0.2	24.887423	25.2485031	25.5006399	24.5603601	24.7436906

Con un promedio que varía entre 44.7398468 y 46.6146854, y una desviación que varía entre 7.53796 y 8.348554.

Elitismo 1 - Selección simple - Grafo 3

	Prob de cruce				
Prob de mutación	0.6	0.65	0.7	0.75	0.8
0.1	26.7936006	26.097752	26.400199	25.711904	25.02061
0.125	26.0838245	26.2336936	26.166359	25.9447034	25.6282327
0.15	26.9232017	26.2722817	26.3451184	25.6451406	25.5801627
0.175	26.2598198	26.0778055	26.2478683	26.8473775	25.9594929
0.2	27.166454	26.7604516	25.7246531	26.2323327	26.3884414

Con un promedio que varía entre 42.9926113 y 44.1810008, y una desviación que varía entre 6.3119313 y 6.6797857.

En la figura 7.7 se pueden observar de manera gráfica los resultados presentados en las tablas anteriores.

7.3.2. Elitismo 1 - Selección pesada

Utilizando elitismo de un individuo y selección pesada, la cual consiste en tomar individuos de forma aleatoria con base en el valor de su función objetivo, se llegó a los siguientes resultados:

Elitismo 1 - Selección pesada - Grafo 1

	Prob de cruce				
Prob de mutación	0.6	0.65	0.7	0.75	0.8
0.1	10.8437511	10.3791095	10.5271491	10.4205133	10.5072273
0.125	10.7274886	10.4512009	10.8691861	10.4800093	10.6368128
0.15	10.5905653	10.5212534	10.4824405	10.5331411	10.4365553
0.175	10.3104504	10.7342477	10.7517189	10.5284254	10.3257217
0.2	10.4271803	10.5249589	10.4292448	10.6946584	10.2550177

Con un promedio que varía entre 10.9306683 y 11.8948808, y una desviación que varía entre 0.7296596 y 1.6010867.

Elitismo 1 - Selección pesada - Grafo 2

	Prob de cruce				
Prob de mutación	0.6	0.65	0.7	0.75	0.8
0.1	21.0454655	20.9088164	20.7729159	20.3217552	21.0288509
0.125	20.7008132	20.8271152	21.0066523	21.0087462	21.0123209
0.15	20.6826886	20.5714735	20.3950579	20.4434601	20.1483462
0.175	20.0271017	20.2908205	20.6701364	20.713104	20.4097718
0.2	20.288609	20.7271131	20.437307	20.4870352	19.9741232

Con un promedio que varía entre 20.9186663 y 22.5704188, y una desviación que varía entre 0.7612842 y 2.0345901.

Elitismo 1 - Selección pesada - Grafo 3

	Prob de cruce				
Prob de mutación	0.6	0.65	0.7	0.75	0.8
0.1	24.0961816	23.3762712	23.0120305	22.7368891	23.3563268
0.125	22.6267004	22.8341071	22.8029479	23.3936281	23.1789454
0.15	23.3003744	22.7210407	23.320423	22.4277868	22.9985577
0.175	22.9574646	23.0717586	22.7530065	23.1913184	22.9111492
0.2	22.8397019	22.9922961	23.3513026	23.0289066	23.1235638

Con un promedio que varía entre 23.4508708 y 25.0937633, y una desviación que varía entre 0.9216508 y 2.03766.

En la figura 7.8 se pueden observar de manera gráfica los resultados presentados en las tablas anteriores.

7.3.3. Elitismo 10 % - Selección simple

Utilizando elitismo de 10 % de la población y selección simple se llegó a los siguientes resultados:

Elitismo 10 % - Selección simple - Grafo 1

	Prob de cruce				
Prob de mutación	0.6	0.65	0.7	0.75	0.8
0.1	10.716214	10.3762954	10.5656567	10.5348061	10.1725703
0.125	10.3526771	10.2928747	10.4472359	10.2540699	10.232713
0.15	10.5253458	10.5453262	10.2989076	10.4041203	10.2651349
0.175	10.3313393	10.103707	10.4291744	10.2846289	10.2349901
0.2	10.2353385	10.2124792	10.1772429	10.3054339	10.1658548

Con un promedio que varía entre 10.6016395 y 11.2113476, y una desviación que varía entre 0.1754639 y 0.7442955.

Elitismo 10 % - Selección simple - Grafo 2

	Prob de cruce				
Prob de mutación	0.6	0.65	0.7	0.75	0.8
0.1	21.29336	20.1858569	20.5907721	20.4016314	19.8389493
0.125	20.6165728	20.427804	20.3288365	20.1229029	20.1600821
0.15	20.9750025	20.1578354	20.7073685	20.6817401	20.2578342
0.175	20.3162174	20.5934249	19.893024	19.9786829	20.3087082
0.2	20.0077026	20.184861	20.5007607	20.8617013	20.4636766

Con un promedio que varía entre 20.3770597 y 21.9192151, y una desviación que varía entre 0.3698812 y 1.0613621.

Elitismo 10 % - Selección simple - Grafo 3

	Prob de cruce				
Prob de mutación	0.6	0.65	0.7	0.75	0.8
0.1	23.4383029	22.7033421	23.4791485	22.5153386	23.0438302
0.125	23.6491574	22.5137739	23.0048868	23.3188275	22.7957189
0.15	22.9451031	22.4254747	22.6708101	22.6656555	22.2451549
0.175	22.6529292	23.1432387	22.6833436	22.2235832	22.552227
0.2	22.9356582	22.8587435	22.7902083	23.4580209	22.8918142

Con un promedio que varía entre 23.0269933 y 24.6064417, y una desviación que varía entre 0.3294229 y 1.1963647.

En la figura 7.9 se pueden observar de manera gráfica los resultados presentados en las tablas anteriores.

7.3.4. Elitismo 10 % - Selección pesada

Utilizando elitismo de 10 % de la población y selección pesada se llegó a los siguientes resultados:

Elitismo 10 % - Selección pesada - Grafo 1

	Prob de cruce				
Prob de mutación	0.6	0.65	0.7	0.75	0.8
0.1	10.4430819	10.5362144	10.4115735	10.4802416	10.2971042
0.125	10.479323	10.5114942	10.3354033	10.0603348	10.7330195
0.15	10.4254866	10.4921496	10.6116969	10.4930762	10.2027029
0.175	10.3242997	10.172243	10.3913879	10.3645865	10.6058504
0.2	10.4182762	10.3152806	10.0955535	10.0750095	10.2539662

Con un promedio que varía entre 10.1493088 y 10.8290073, y una desviación que varía entre 0.0533263 y 0.3485089.

Elitismo 10 % - Selección pesada - Grafo 2

	Prob de cruce				
Prob de mutación	0.6	0.65	0.7	0.75	0.8
0.1	20.9797772	21.0952469	20.6982001	21.1292987	21.0275068
0.125	20.7367654	20.8789286	20.6214203	20.9980827	21.2341523
0.15	20.8350876	20.9623709	20.4655448	21.063738	20.3796244
0.175	20.2663823	20.6922058	20.2903309	20.5199469	20.7763941
0.2	20.6995308	20.6988346	20.4867812	20.3937693	19.7377934

Con un promedio que varía entre 19.904292 y 21.3075422, y una desviación que varía entre 0.0442516 y 0.4497975.

Elitismo 10 % - Selección pesada - Grafo 3

	Prob de cruce				
Prob de mutación	0.6	0.65	0.7	0.75	0.8
0.1	23.5485651	23.1508691	23.8600862	23.2620584	22.9554879
0.125	23.4424025	22.8176972	23.1515712	23.5777075	23.2255779
0.15	23.2264569	23.0689531	23.0061327	23.6422458	23.4214779
0.175	23.0362987	22.6869404	22.8580926	23.4455918	23.1779795
0.2	23.3980545	22.7085919	22.5596025	23.2005043	23.6921039

Con un promedio que varía entre 22.8356632 y 24.40109576, y una desviación que varía entre 0.0548572 y 0.6027947.

En la figura 7.10 se pueden observar de manera gráfica los resultados presentados en las tablas anteriores.

7.3.5. Análisis

Analizando las figuras 7.7, 7.8, 7.9 y 7.10, se puede notar que los mejores resultados fueron arrojados utilizando elitismo de 10 % y selección simple. Una posible

razón para que la selección simple sea mejor que la pesada en este caso es que al ser aleatoria mantiene la diversidad que se desea en la población, mientras que la pesada probablemente tome individuos que se parecen, formando individuos que también se parecen.

Con respecto a las probabilidades, si se analizan las tablas de la sección 7.3.3, se puede ver que la probabilidad de cruce en 0.8 es la que genera mejores resultados, mientras que la pobabilidad de mutación puede variar entre 0.15 y 0.2.

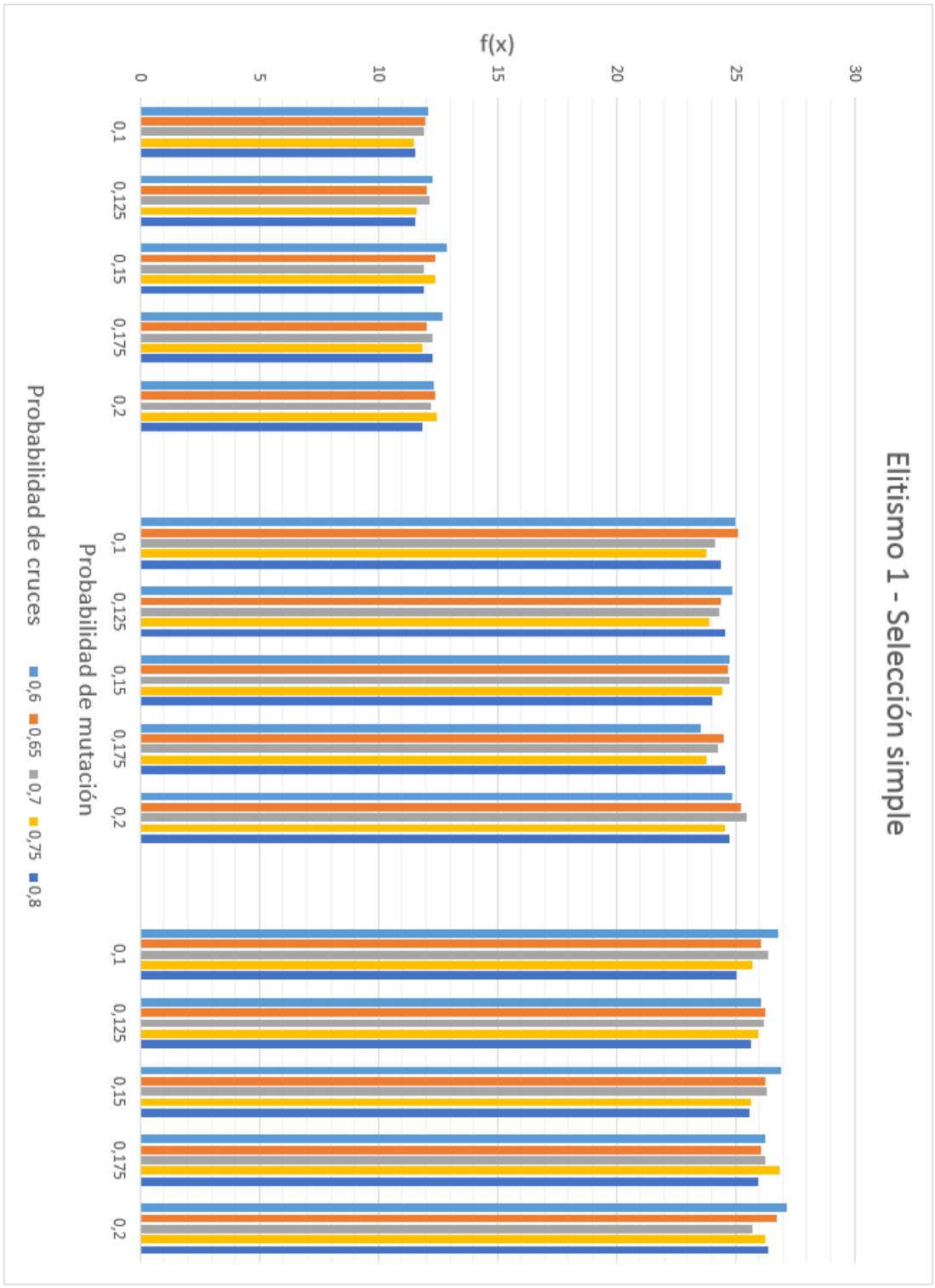


Figura 7.7: Gráfico asociado a las tablas de la sección 7.3.1.

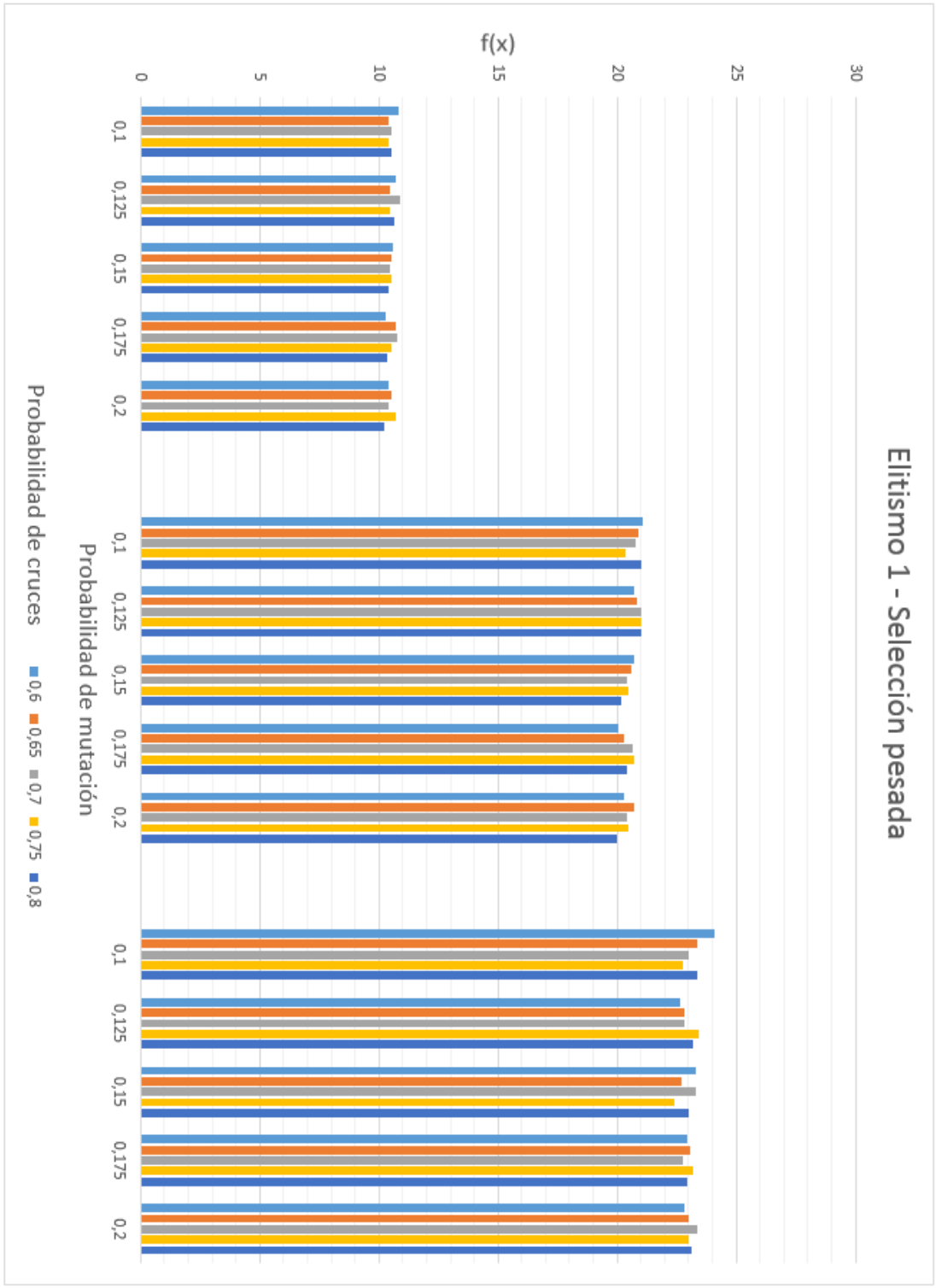


Figura 7.8: Gráfico asociado a las tablas de la sección 7.3.2.

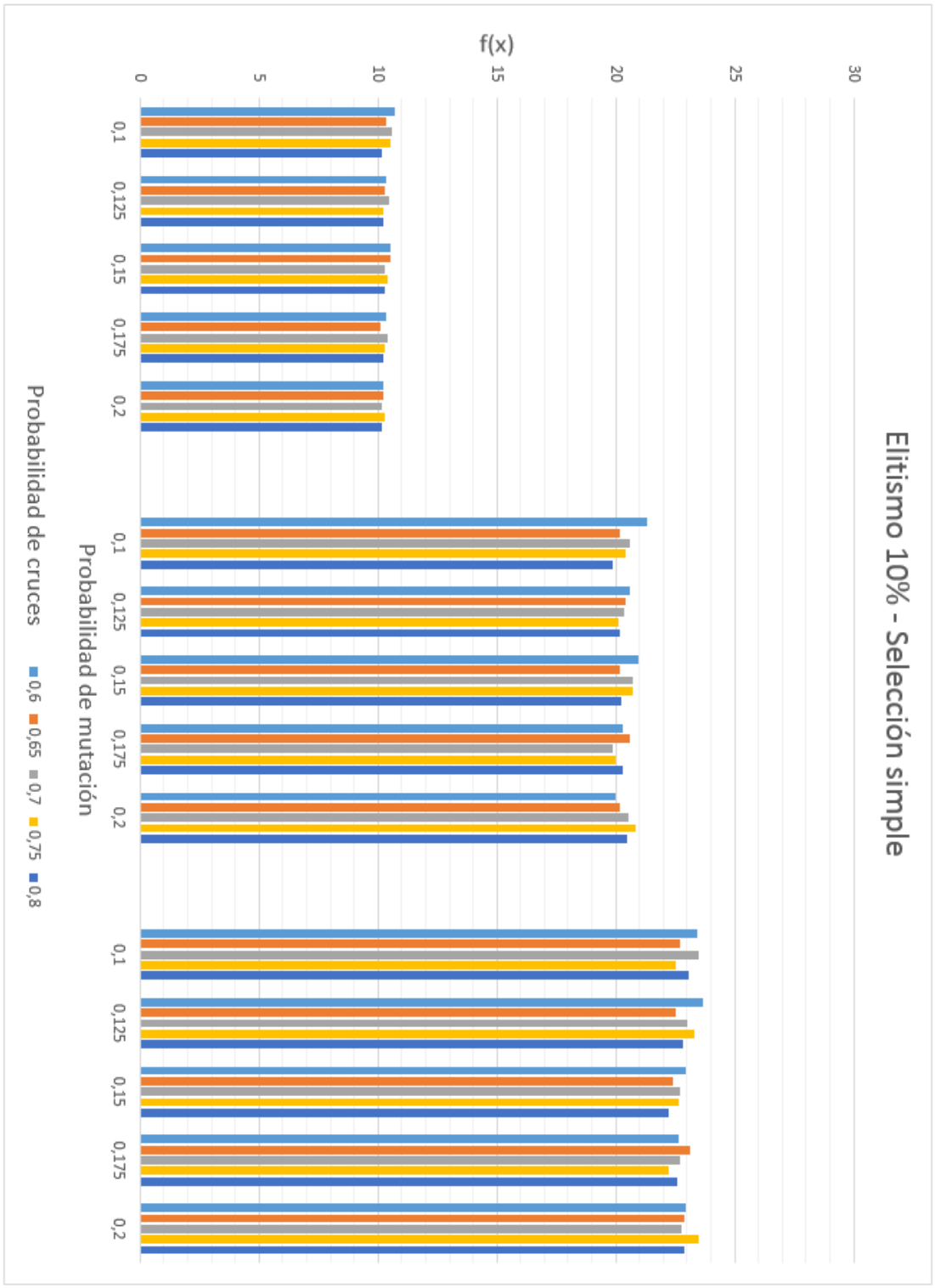


Figura 7.9: Gráfico asociado a las tablas de la sección 7.3.3.

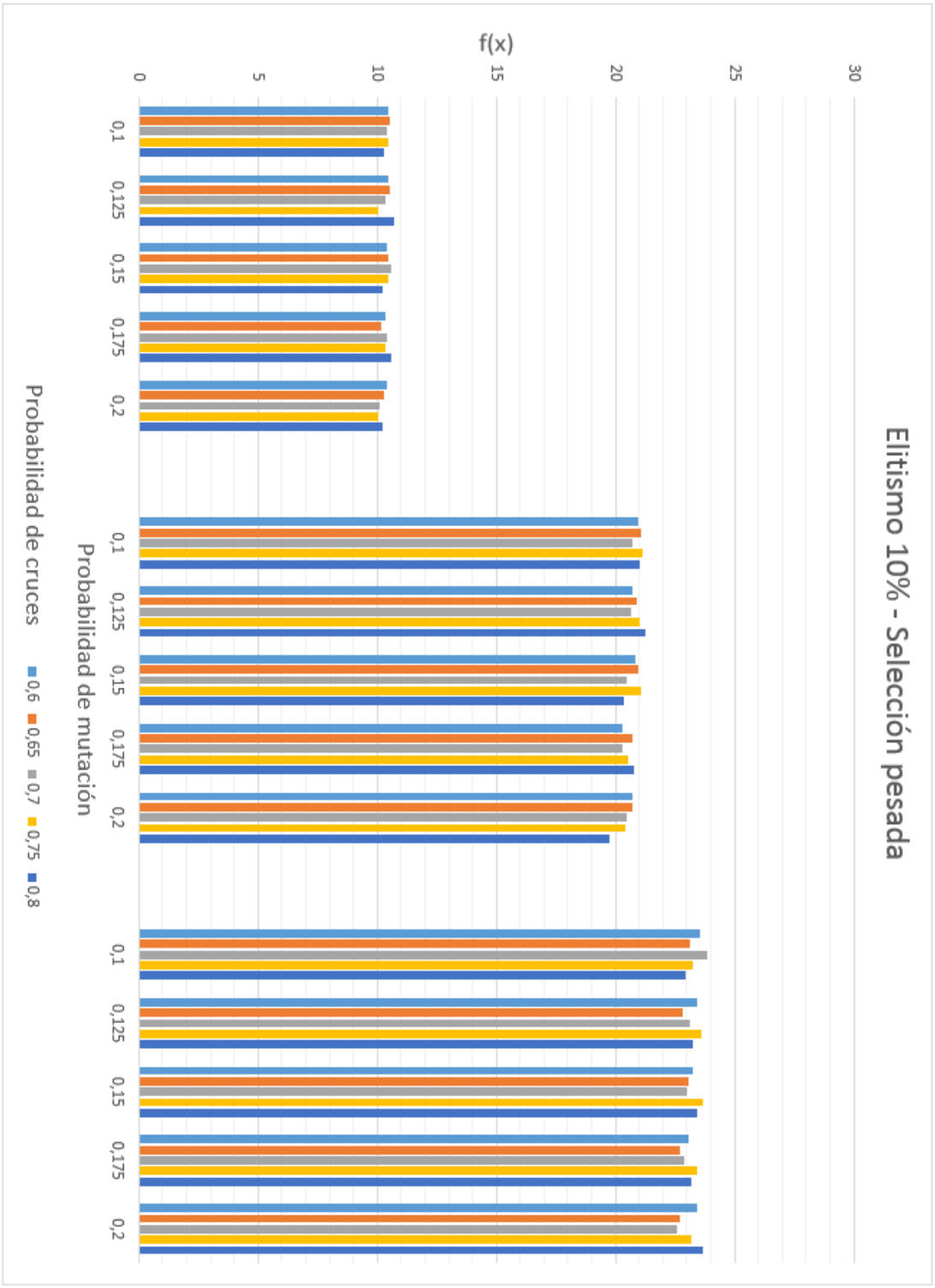


Figura 7.10: Gráfico asociado a las tablas de la sección 7.3.4.

Conslusiones y trabajos futuros

La representación gráfica de problemas es fundamental para la resolución del mismo si es muy amplio. Esta problemática da nacimiento la visualización de grafos, el cual pasa a formar parte importante de cualquier ámbito que requiera dicha representación gráfica. Con el nacimiento de la visualización de grafos, también nacen las aplicaciones que lo implementan, desde el más básico al más avanzado, entre ellas la aplicación implementada en este trabajo de investigación, la cual se utilizó incluso para documentar cuales medidas de calidad son necesarias y cuales no.

El uso de algoritmos genéticos para la minimización de la ecuación de energía permitió llegar a una solución aceptable en un tiempo aceptable, considerando que el problema tratado es un problema no lineal.

Las pruebas visuales realizadas a la aplicación permiten concluir que las medidas de calidad del dibujado de grafos son la medida de cruces, simetría y mínimo ángulo. El área, considerada como una opción al principio, pasó a formar parte de las medidas innecesarias que no favorecen el dibujado del grafo. Con respecto al mínimo ángulo, no es una medida que se adopte por completo, ya que solo se toma un décimo del valor de la misma para la ecuación de energía.

Las pruebas de rendimiento de la aplicación arrojaron que la mejor relación de resultado y tiempo de ejecución se obtenía utilizando 100 individuos por población y 50 generaciones. Utilizar más generaciones, a pesar de llevar a mejores resultados, ocasiona que el tiempo de ejecución del algoritmo se eleve abruptamente.

Con respecto al resto de las pruebas, se puede concluir que los mejores parámetros para las probabilidades de cruce y mutación son 0.8 y un valor entre 0.15 y 0.2, respectivamente. Adicionalmente, se recomienda el uso de elitismo con 10 % de la población y una selección aleatoria de individuos para realizar los cruces y mutaciones. A pesar de que en la teoría se establece que la selección pesada debería dar mejores resultados, en este caso no fue así; una posible razón para esto es que al tomar de manera regular a los mejores individuos para los cruces se corre el riesgo de que los mismos sean parecidos, generando individuos similares para la siguiente generación, perdiendo de esa manera la diversidad que se busca para hallar la mejor solución.

Tabla resumen

Ecuación de energía	$cross(x) + symmetry(x) + \frac{angle(x)}{10}$
Tamaño de población	100
Número de generaciones	50
Tipo de elitismo	10 % de la población
Método de selección	Simple
Probabilidad de cruce	0.8
Probabilidad de mutación	$0.15 \sim 0.2$

Con respecto a los trabajos futuros, el algoritmo genético es paralelizable, un buen trabajo a futuro sería paralelizarlo y evaluar que tan buena es la reducción del tiempo de ejecución del mismo.

Se podría establecer un servicio web, más que una simple aplicación web, para realizar aplicaciones móviles que hagan solicitudes al mismo servidor, ampliando un poco más la gama de aplicaciones que hacen dibujado de grafos.

En algún punto sería bueno ampliar la cantidad de formatos que importa y exporta la aplicación, para hacerla un poco más flexible. Y no solo la cantidad de formatos, sino tambien la cantidad de campos que se toman en cuenta de cada formato. En la

aplicación solo se utilizan los campos de las posiciones de los vértices y aristas, pero en cada formato se puede especificar muchísimo más que eso.

Bibliografía

[1] Django. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: https://www.djangoproject.com/

[2] HTML5 Canvas. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://www.w3schools.com/html/html5_canvas.asp

[3] Bootstrap. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://getbootstrap.com/

[4] jQuery. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: https://jquery.com/

[5] Grimaldi R. P. (1994), Cap. 11 An Introduction to Graph Theory. En (3ª Ed.) Discrete and Combinatorial Mathemathics, An Applied Introduction (pp. 527 - 698). Estados Unidos de América: Adison-Wesley Publishing Company.

[6] Rossen K. H. (2007), Cap. 9 Graphs. En (6ª Ed.) Discrete Mathematics and It's Applications (pp. 589 - 682). Nueva York, Estados Unidos de América: Mc Graw Hill.

[7] Graph (mathematics). From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://en.wikipedia.org/wiki/Graph_(mathematics)

[8] Isomorfismo. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://es.wikipedia.org/wiki/Isomorfismo

[9] Isomorfismo de grafos. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://es.wikipedia.org/wiki/Isomorfismo_de_grafos

[10] Ciclo Euleriano. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://es.wikipedia.org/wiki/Ciclo_euleriano

[11] Hierholzer C. Über die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren (Acerca de la posibilidad de recorrer un grafo sin repeticiones ni interrupciones). Mathematische Annalen VI (1873), 30–32.

[12] Camino hamiltoniano. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://es.wikipedia.org/wiki/Camino_hamiltoniano

[13] NP-completo. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://es.wikipedia.org/wiki/NP-completo

[14] Algoritmo de Dijkstra. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://es.wikipedia.org/wiki/Algoritmo_de_Dijkstra

[15] Grafo plano. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://es.wikipedia.org/wiki/Grafo_plano

[16] Breadth-first search. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://en.wikipedia.org/wiki/Breadth-first_search

[17] Depth-first search. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://en.wikipedia.org/wiki/Depth-first_search

[18] Backtracking. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://en.wikipedia.org/wiki/Backtracking

[19] Prim's algorithm. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://en.wikipedia.org/wiki/Prim%27s_algorithm

[20] Kruskal's algorithm. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://en.wikipedia.org/wiki/Kruskal%27s_algorithm

[21] Floyd-Warshall algorithm. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm

[22] Critical path method. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://en.wikipedia.org/wiki/Critical_path_method

[23] Técnicas de revisión y evaluación de programas. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://es.wikipedia.org/wiki/T%C3%A9cnica_de_revisi%C3%B3n_y_evaluaci%C3%B3n_de_programas

[24] Computer science. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://en.wikipedia.org/wiki/Computer_science

[25] Collision detection. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://en.wikipedia.org/wiki/Collision_detection

[26] Image segmentation. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://en.wikipedia.org/wiki/Image_segmentation

[27] Document Object Model. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://en.wikipedia.org/wiki/Document_Object_Model

[28] Max-flow min-cut theorem. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://en.wikipedia.org/wiki/Max-flow_min-cut_theorem

[29] Graph drawing. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://en.wikipedia.org/wiki/Graph_drawing

[30] Isabel F. Cruz, Roberto Tamassia, Graph Drawing Tutorial. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://pdf.aminer.org/000/361/116/constrained_graph_drawing_with_evolution_strategies.pdf

[31] Minimum bounding box. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://en.wikipedia.org/wiki/Minimum_bounding_box

[32] Force-directed graph drawing. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://en.wikipedia.org/wiki/Force-directed_graph_drawing

[33] Ley de elasticidad de Hooke. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://es.wikipedia.org/wiki/Ley_de_elasticidad_de_Hooke

[34] J. Nocedal y S. J. Wright (2006), Numerical Optimization. Springer.

[35] Barnes-Hut simulation. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://en.wikipedia.org/wiki/Barnes%E2%80%93Hut_simulation

[36] Layered graph drawing. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://en.wikipedia.org/wiki/Layered_graph_drawing

[37] Sugiyama, Kozo; Tagawa, Shôjirô; Toda, Mitsuhiko (1981), "Methods for visual understanding of hierarchical system structures", IEEE Transactions on Systems, Man, and Cybernetics.

[38] NP-hard. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://en.wikipedia.org/wiki/NP-hard

[39] Integer programming. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://en.wikipedia.org/wiki/Integer_programming

[40] L. Wolsey Branch and Bound. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: https://www.math.washington.edu/~burke/crs/409/notes/wolsey_bb.pdf

[41] Linear programming. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://en.wikipedia.org/wiki/Linear_programming

[42] Arc diagram. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://en.wikipedia.org/wiki/Arc_diagram

[43] 2-satisfiability. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://en.wikipedia.org/wiki/2-satisfiability

[44] Janet M. Six, Ioannis G. Tollis (2013), Cap. 9 Circular Drawing Algorithms. En Handbook of Graph Drawing and Visualization. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: https://cs.brown.edu/~rt/gdhandbook/chapters/circular.pdf

[45] Biconnected component. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://en.wikipedia.org/wiki/Biconnected_component

[46] Cytoscape. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://www.cytoscape.org/

[47] The Systems Biology Markup Language. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://sbml.org/Main_Page

[48] The Open Biological and Biomedical Ontologies. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://www.obofoundry.org/

[49] Gephi. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: https://gephi.github.io/

[50] Graphviz - Graph Visualization Software. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://graphviz.org/

[51] DOT (graph description language). From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://en.wikipedia.org/wiki/DOT_(graph_description_language)

[52] Mathematica de Wolfram. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://www.wolfram.com/mathematica/

[53] Tulip. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://tulip.labri.fr/TulipDrupal/

[54] Graph File Formats. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://www2.sta.uwi.edu/~mbernard/research_files/fileformats.pdf

[55] yEd Graph Editor. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://www.yworks.com/en/products/yfiles/yed/

[56] Unified Modeling Language Resource Page. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://www.uml.org/

[57] Object Management Group, Business Process Model and Notation. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://www.bpmn.org/

[58] Microsoft Excel. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://en.wikipedia.org/wiki/Microsoft_Excel

[59] Adobe Flash Professional CC. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://helpx.adobe.com/flash.html

[60] Michalewicz Z. (1996). Genetic Algorithms + Data Structures = Evolution Programs (3ª Ed.). Estados Unidos de América: Springer.

[61] Natural selection. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://en.wikipedia.org/wiki/Natural_selection

[62] Travelling salesman problem. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://en.wikipedia.org/wiki/Travelling_salesman_problem

[63] Knapsack problem. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://en.wikipedia.org/wiki/Knapsack_problem

[64] Qing-Guo Zhang, Hua-Yong Liu, Wei Zhang, and Ya-Jun Guo (2005), Drawing Underected Graphs with Genetic Algorithms. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://www.cs.cas.cz/petra/EA/grafy/graph.pdf

[65] Jürgen Branke, Frank Bucher, Hartmut Schmeck (1996), Using Genetic Algorithms for Drawing Undirected Graphs. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=6AD5F080580AE5A066C119D3EEB5374F?doi=10.1.1.45.5676&rep=rep1&type=pdf

[66] Erkki Mäkinen (2001), TimGA: A Genetic Algorithm for Drawing Undirected Graphs. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://www.emis.de/journals/DM/v92/art5.pdf

[67] Web application framework. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://en.wikipedia.org/wiki/Web_application_framework

[68] Model-view-controller. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

[69] HTML. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://en.wikipedia.org/wiki/HTML

[70] HTML5. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://en.wikipedia.org/wiki/HTML5

[71] Django Documentation, Templates. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: https://docs.djangoproject.com/en/1.8/topics/templates

[72] Jinja2. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://jinja.pocoo.org/docs/dev/

[73] Cascade Style Sheets. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://en.wikipedia.org/wiki/Cascading_Style_Sheets

[74] JavaScript. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://en.wikipedia.org/wiki/JavaScript

[75] Object-relational mapping. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://en.wikipedia.org/wiki/Object-relational_mapping

[76] Python. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: https://www.python.org/

[77] Ajax (programming). From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: https://en.wikipedia.org/wiki/Ajax_(programming)

[78] Uniform resource locator. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: https://en.wikipedia.org/wiki/Uniform_resource_locator

[79] Portable Network Graphics. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: https://en.wikipedia.org/wiki/Portable_Network_Graphics

[80] Github, gliffy/canvas2svg. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: https://github.com/gliffy/canvas2svg

[81] Github, dankogai/js-base64. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: https://github.com/dankogai/js-base64

[82] Scalable Vector Graphics. From Wikipedia, the free encyclopedia. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: https://en.wikipedia.org/wiki/Scalable_Vector_Graphics

[83] Introducing JSON. [Fecha de consulta: 07 de Julio del 2015]. Disponible en: http://json.org/

[84] Python. Generate pseudo-random numbers. [Fecha de consulta: 23 de Julio del 2015]. Disponible en: https://docs.python.org/2/library/random.html