FACULTAD DE INFORMÁTICA

UNIVERSIDAD POLITÉCNICA DE MADRID

UNIVERSIDAD POLITÉCNICA DE MADRID

FACULTAD DE INFORMÁTICA

TRABAJO FIN DE CARRERA

"REPRESENTACIONES DE VISIBILIDAD EN TRAZADO DE GRAFOS (GRAPH DRAWING)"

AUTOR: WALBER GONZÁLEZ SEDEÑO

TUTOR: GREGORIO HERNÁNDEZ PEÑALVER

AGRADECIMIENTOS

Primeramente, me gustaría dar las gracias al Tutor del Proyecto, Gregorio Hernández Peñalver, del Departamento de Matemática Aplicada de la Facultad de Informática de la Universidad Politécnica de Madrid, por haberme permitido realizar este proyecto y por haber sido siempre de gran ayuda en los momentos necesarios.

En segundo lugar, mi reconocimiento a todos los compañeros y profesores que he tenido a lo largo de la carrera y que, sin duda, han contribuido a que llegara a este momento.

Por último, pero no por ello menos importante, mis agradecimientos a mis padres, sencillamente, por todo.

ÍNDICE

ÍNDICE DE ILUSTRACIONES

1 INTRODUCCIÓN

El *Trazado de Grafos* (*Graph Drawing*) es un área de la teoría de grafos que trata la construcción de representaciones geométricas para los grafos. Los distintos trazados de grafos que existen facilitan la comprensión y la visualización de la información representada en estas estructuras.

En la actualidad, múltiples áreas de la informática aplican el trazado de grafos. Por ejemplo, ingeniería del software (diagramas de estado, diagramas de flujo de datos, diagramas de clases, diagramas de módulos), bases de datos (diagramas de Entidad-Relación), gestión de proyectos (diagramas PERT), inteligencia artificial (diagramas de representación del conocimiento), telecomunicaciones (diagramas de redes de computadores), diseño VLSI, programación lógica (árboles SLD). En otros campos de la ciencia encontramos más aplicaciones: biología (arboles filogenéticos), química (simulación de estructuras moleculares), arquitectura (planos planta), etc. [DiB98], [Nis04].

La *Representación de Visibilidad* es una representación geométrica de digrafos acíclicos planos en la que cada vértice se convierte en un segmento horizontal y cada arista en un segmento vertical siempre que no intercepte un segmento horizontal. El estudio de esta representación fue impulsado inicialmente por el diseño VLSI y problemas de compactación. Dentro del Trazado de Grafos, esta representación se utiliza como base para la obtención de trazados poligonales y ortogonales.

1.1 Objetivos y requisitos

Este trabajo fin de carrera tiene como objetivo principal desarrollar una aplicación que permita construir la representación de visibilidad y los trazados que se derivan de ella. Específicamente, tendremos un sistema informático que permita la ejecución de los siguientes algoritmos:

- Representación de visibilidad.
- Representación de visibilidad con restricciones.
- Trazado poligonal ascendente.
- Trazado poligonal ascendente con restricciones.
- Trazado ortogonal plano.
- Trazado de dominancia rectilíneo.
- Trazado de dominancia poligonal.

También, la aplicación permitirá convertir un grafo en digrafo a través de la orientación de sus aristas.

Además, se podrán realizar las operaciones básicas de edición de grafos. Es decir, existirán las funcionalidades de agregar, eliminar, seleccionar, copiar, pegar y cortar elementos de un grafo. Al mismo tiempo, se podrán deshacer o rehacer estas operaciones. También, existirá la posibilidad de abrir o guardar un grafo editado a un archivo.

La ejecución de los algoritmos debe ser paso a paso, mostrando las estructuras y representaciones intermedias que se generan. El usuario podrá controlar la ejecución de cada algoritmo pudiendo avanzar o retroceder un paso, así como ir hasta el final de la ejecución o cancelarla. De este modo, se facilita el estudio y la comprensión de los algoritmos a los usuarios del sistema. En esta finalidad didáctica radica la principal diferencia entre esta aplicación y las que ya han sido desarrolladas en el área del Trazado de Grafos [Ope].

1.2 Alcance del proyecto

Los artefactos entregables que genera la realización de este proyecto son: el sistema desarrollado, la documentación del proyecto y un documento para ser publicado en la Web.

El sistema software incluirá todo el código fuente junto al archivo ejecutable. El sistema será desarrollado utilizando el lenguaje de programación Java. El software se podrá ejecutar en cualquier sistema operativo que soporte la plataforma Java [Ora93].

La documentación del proyecto (es el presente documento) aglutinará las definiciones teóricas de los algoritmos implementados, la descripción del proyecto software llevado a cabo y el manual de usuario de la aplicación.

El documento que se publicará en la Web será realizado utilizando el formato HTML y se compondrá de un resumen de la documentación del proyecto.

Los recursos a utilizar durante el desarrollo consistirán principalmente en el ordenador personal con el sistema operativo Windows 7, la plataforma de desarrollo Java, el entorno de desarrollo Eclipse y el sistema de control de versiones Subversion unido al repositorio XP-Dev.com. Otros recursos como libros o información teórica estarán disponibles tanto en la biblioteca del centro de estudios como a través de internet.

También, se empezará a desarrollar a partir del código fuente del proyecto open-source Visigraph [Beh11]. Fundamentalmente, se reutilizará el código de la interfaz gráfica y las estructuras de datos para representar los grafos.

Por último, se utilizará el idioma inglés en la aplicación, dado que es la lengua mayoritariamente usada en las publicaciones y el software de Trazado de Grafos.

1.3 Contenido del documento

Esta memoria se ha dividido en los siguientes capítulos:

- En el capítulo 1, como acabamos de ver, se hace una breve introducción al trazado de grafos y sus aplicaciones, así como a la representación de visibilidad. También, se definen los objetivos del proyecto.
- El capítulo 2 se dedicará a realizar una introducción teórica al trazado de grafos con el objetivo de conocer específicamente como se caracterizan los trazados y las diferentes estrategias para construirlos.
- Una vez que comprendemos las distintas formas de generar los trazados, en el capítulo 3 nos adentraremos en la explicación teórica de los algoritmos relacionados con la representación de visibilidad.
- En el capítulo 4 tendremos toda la documentación del desarrollo software realizado en el proyecto.
- Finalizada la implementación del proyecto, en el capítulo 5 procederemos a explicar cómo usar la aplicación a través del manual de usuario.
- Por último, en el capítulo 6 estarán las conclusiones del trabajo fin de carrera.

2 INTRODUCCIÓN TEÓRICA AL TRAZADO DE GRAFOS

Antes de empezar a ver las principales definiciones y características de los trazados, vale la pena reseñar que el contenido teórico tanto de este capítulo como del siguiente está basado en el libro [DiB98].

2.1 Definición de trazado de grafos

El trazado de un grafo dado *G* se denomina *Γ* o *Gamma*. Tenemos que cada vértice *v* de *G* se representa en un punto *Γ(v)* y cada arista *(u, v)* se representa como una curva de Jordan entre los puntos *Γ(u)* y *Γ(v)*. Si las aristas son dirigidas, se dibujan como flechas.

2.2 Parámetros de las estrategias de trazado de grafos

Determinar si un trazado es más idóneo que otro depende del dominio de la aplicación donde se utiliza. Para poder describir un trazado se introducen conceptos más específicos como *reglas de trazado*, *estética* y *restricciones*. Estos tres conceptos son los parámetros fundamentales de las estrategias de trazado *de* grafos.

2.2.1 Reglas de trazado

Las *reglas de trazado* son normas que el dibujo debe cumplir. En un mismo trazado se puede requerir que se cumplan varias reglas a la vez.

Las reglas más usadas son:

- *Trazado poligonal (polygonal):* cada arista es trazada como una serie de segmentos unidos de forma poligonal. (Ver Ilustración 1)
- *Trazado rectilíneo (straight-line):* cada arista es trazada como un segmento en línea recta. (Ver Ilustración 2)
- *Trazado ortogonal (orthogonal):* cada arista es trazada como un conjunto de segmentos unidos de forma poligonal, pero cada segmento solo es horizontal o vertical. (Ver Ilustración 3)
- *Trazado en malla (grid):* cada vértice, corte entre aristas o codo de una arista se ubica sobre una malla asignándoles coordenadas enteras. (Ver Ilustración 4)
- *Trazado plano (planar):* no se permiten cortes entre aristas. (Ver Ilustración 5)
- *Trazado ascendente (upward):* en un digrafo acíclico, cada arista es trazada como un segmento monótonamente no decreciente en dirección vertical. Cuando el segmento es estrictamente creciente, se denomina *estrictamente ascendente (strictly upward)*. (Ver Ilustración 6)
- *Trazado descendente (downward):* en un digrafo acíclico, cada arista es trazada como un segmento monótonamente no creciente en dirección vertical. Cuando

el segmento es estrictamente decreciente, se denomina *estrictamente descendente (strictly downward)*. (Ver Ilustración 7)

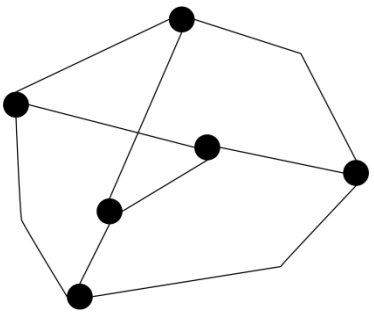

Ilustración 1: Trazado Poligonal

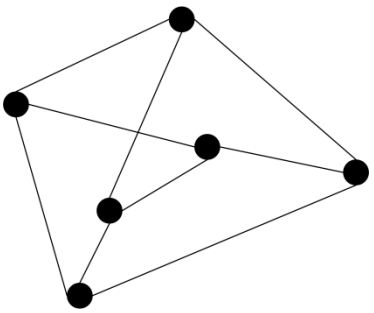

Ilustración 2: Trazado Rectilíneo

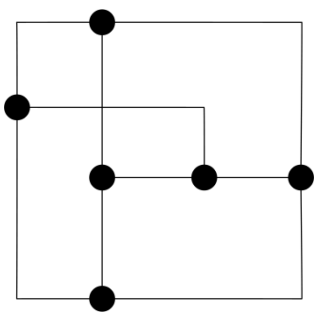

Ilustración 3: Trazado Ortogonal

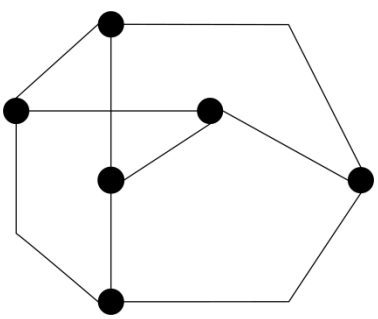

Ilustración 4: Trazado en Malla

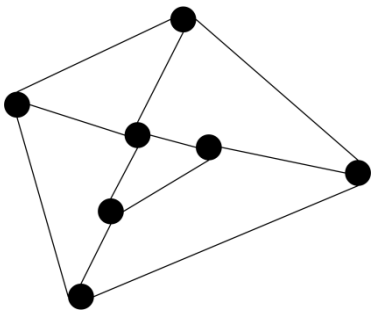

Ilustración 5: Trazado Plano

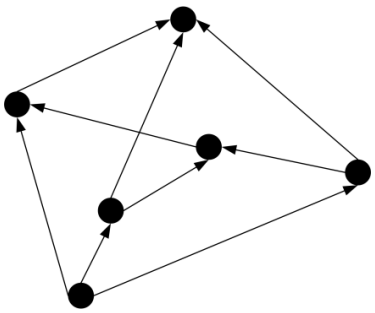

Ilustración 6: Trazado Estrictamente Ascendente

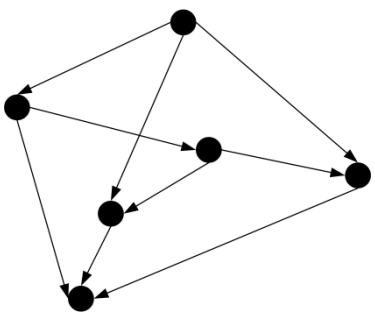

Ilustración 7: Trazado Estrictamente Descendente

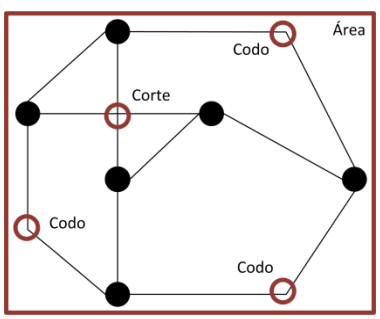

Ilustración 8: Estética. Corte, Área y Codo

2.2.2 Estética

Estos parámetros definen propiedades gráficas del trazado. Es deseable que se cumplan tanto como sea posible.

Las propiedades más comunes son:

- *Número de cortes:* minimizar el número de cortes entre aristas.
- *Área:* minimizar el área del trazado del grafo. El área puede ser definida en base al cierre convexo mínimo del trazado o al rectángulo mínimo que abarca el trazado.
- *Longitudes de las aristas:*
  - minimizar las longitudes de las aristas.
  - minimizar la variación de las longitudes de las aristas.
- *Número de codos:*
  - minimizar el número máximo de codos de las aristas.
  - minimizar la variación del número de codos de las aristas.
- *Resolución angular:* maximizar el ángulo entre dos aristas incidentes en el mismo vértice.
- *Relación de aspecto:* minimizar la relación entre el ancho y la altura del rectángulo mínimo que abarca el trazado.
- *Simetría:* mostrar las simetrías del grafo en el trazado.

En la Ilustración 8 podemos observar que es un corte, un codo o el área de un grafo.

2.2.3 Restricciones

Las *restricciones* son reglas que afectan solo a partes específicas del grafo o del trazado. Al contrario, los parámetros *reglas de trazado* o *estética* afectan al grafo o al trazado en general.

Algunos ejemplos de estas reglas son:

- *Centro:* situar un vértice dado en el centro del trazado.
- *Cara Externa:* ubicar un vértice dado en el límite del trazado.
- *Grupo:* colocar agrupados un conjunto de vértices determinados.
- *Camino Izquierda-Derecha:* dibujar un camino dado con sus vértices alineados horizontalmente de izquierda a derecha.
- *Camino Arriba-Abajo:* dibujar un camino dado con sus vértices alineados verticalmente de arriba a abajo.
- *Forma:* dibujar un subgrafo con una forma predefinida.

2.3 Estrategias de trazado de grafos

En la teoría del Trazado de Grafos se definen diferentes estrategias que tienen como objetivo ajustarse a una de las *reglas de trazado*. Estos métodos suelen dividirse en pasos, en cada uno se intenta que se cumpla una propiedad *estética*. A menudo, estos criterios estéticos entran en conflicto entre sí, por lo que las estrategias establecen una relación de precedencia entre las etapas en las que están divididas.

2.3.1 Estrategia Topología-Forma-Métrica

Esta estrategia ha sido ideada para construir trazados ortogonales en malla. Permite tratar de forma similar a un conjunto amplio de parámetros estéticos y restricciones. Este enfoque está basado en que un trazado ortogonal tiene tres propiedades fundamentales: *topología*, *forma* y *métrica*. Estas tres características definen tres clases de equivalencia entre distintos trazados ortogonales de un mismo grafo:

- *Topología:* dos trazados ortogonales son equivalentes topológicamente si uno puede ser construido a partir del otro a través de una deformación continua que no cambia las aristas que están alrededor de las caras del trazado.
- *Forma:* dos trazados ortogonales tienen la misma forma si tienen la misma *topología* y uno puede ser obtenido a partir del otro solo modificando la longitud de los segmentos que representan las aristas sin cambiar el ángulo formado por ellos.
- *Métrica:* dos trazados ortogonales tienen la misma métrica si son congruentes geométricamente. Es decir, podemos transformar un trazado en el otro a través de una traslación y/o una rotación.

Cada característica es una mejora de la anterior. Dada la existencia de esta relación, el algoritmo para generar el trazado se divide en tres etapas: *planarización*, *ortogonalización* y *compactación*.

1. *Planarización:* determina la *topología* del trazado a través de una *inmersión plana* (*planar embedding*). El objetivo de esta etapa es reducir el número de cortes entre aristas tanto como sea posible. Para lograr esto, se insertan vértices ficticios donde haya un corte. Finalmente, se obtiene una *topología* plana.
2. *Ortogonalización:* dada la *topología*, determina la *forma* del trazado. Se genera un trazado ortogonal del grafo. En esta representación los vértices no tienen coordenadas y las aristas tienen una lista de ángulos que simbolizan sus codos. La meta de este paso es reducir el número de codos.
3. *Compactación:* dado el trazado ortogonal, determina las coordenadas finales de los vértices y los codos de las aristas. La finalidad de esta etapa es minimizar

el área del trazado. Los vértices ficticios son eliminados. Finalmente, se obtiene un trazado ortogonal en malla.

Como hemos visto, el algoritmo establece una prioridad sobre los parámetros estéticos. Así, tenemos que el orden de prioridad es número de cortes, número de codos y área. En la etapa de *compactación*, podemos aplicar otros parámetros estéticos en lugar de minimizar el área. Por ejemplo, minimizar las longitudes de las aristas.

Al igual que los parámetros estéticos, podemos tener en cuenta las restricciones. En la etapa de *planarización*, es posible colocar vértices en la cara externa o impedir que haya cortes de aristas a lo largo de un determinado camino. En la fase de *ortogonalización*, podemos precisar que un camino no contenga codos en sus aristas. En el paso de *compactación*, es factible establecer ciertas condiciones a las coordenadas de los vértices. Como ocurre con los parámetros estéticos, la prioridad de las restricciones está determinada por las etapas donde se aplican.

2.3.2 Estrategia Jerárquica

Esta estrategia es apropiada para generar trazados poligonales, especialmente poligonales ascendentes o descendentes. Los digrafos acíclicos suelen representarse aplicando estas reglas de trazado. Otros tipos de grafos pueden ser transformados parcialmente para convertirlos en digrafos acíclicos y, así, poder ser utilizados.

El algoritmo que lleva a cabo esta estrategia está dividido en tres pasos:

1. *Asignación de capas:* dado un digrafo acíclico, asigna los vértices a sus respectivas capas. Se utiliza la denominación L_1, L_2,… , L_h para las capas. La asignación se realiza cumpliendo la regla de que si *(u, v)* es una arista del digrafo, $u \in L_i$ y $v \in L_j$, entonces $i > j$. El grafo por capas apropiado es en el que se cumple que $i = j + 1$. Para lograr esto, se insertan vértices ficticios en las aristas que abarcan más de dos capas.
2. *Reducción de cortes:* dado un grafo por capas apropiado, determina un orden para los vértices de cada capa. Se trata de minimizar los cortes tanto como sea posible. Este orden determina la topología del trazado final.
3. *Asignación de coordenadas:* determina las coordenadas de los vértices. Las coordenadas del eje de ordenadas de los vértices son las capas en las que están situados. Las coordenadas del eje de abscisas se asignan preservando el orden de los vértices en cada capa. Si las aristas ocupan varias capas, se representan como una poligonal remplazando los vértices ficticios por codos. En otro caso, como un segmento. El objetivo en el caso de las aristas largas es minimizar el número de codos.

En la etapa de *asignación de coordenadas* podemos aplicar otros parámetros estéticos como mostrar las simetrías del digrafo desplazando los vértices horizontalmente o reducir el área del trazado juntando los vértices. Como ocurre en la estrategia Topología-Forma-Métrica, la prioridad entre los parámetros estéticos viene fijada por el orden de los pasos.

También, es posible incluir restricciones en esta estrategia. Por ejemplo, en el paso de *reducción de cortes* se puede obligar que dos vértices permanezcan próximos entre sí. Además, en la etapa de *asignación de coordenadas* es factible requerir que se mantengan alineados verticalmente algunos vértices ubicados en diferentes capas.

2.3.3 Estrategia Visibilidad

Este método está orientado a la creación de trazados poligonales. El algoritmo se realiza en tres pasos: *planarización*, *visibilidad por segmentos* y *reemplazamiento*.

1. *Planarización:* determina la *topología* del grafo a través de una *inmersión plana*. Se realizan las mismas operaciones que en el paso 1 de la estrategia Topología-Forma-Métrica.
2. *Visibilidad por segmentos:* determina una *representación de visibilidad* del grafo. En una *representación de visibilidad*, cada vértice *v* se convierte en un segmento horizontal y cada arista (*u, v)* en un segmento vertical entre los segmentos horizontales de *u* y *v*. Los segmentos verticales no se interceptan con ningún otro segmento horizontal que no sea uno de sus extremos.
3. *Reemplazamiento:* construye el trazado poligonal final. Cada segmento horizontal se sustituye por un punto del segmento representando a su vértice. Cada segmento vertical se cambia por una línea poligonal representando a su arista. Esta línea poligonal se mantiene mayoritariamente sobre el segmento vertical. Los vértices ficticios que se introdujeron en la planarización son sustituidos por cortes entre las líneas poligonales.

Varios parámetros estéticos son aplicables a lo largo del algoritmo. En la etapa de *planarización*, es deseable la reducción del número de cortes entre aristas. En el paso de *visibilidad por segmentos*, sería ideal minimizar el área de la *representación de visibilidad*. En la fase de *reemplazamiento*, es posible minimizar el número de codos, resaltar las simetrías del grafo o balancear la distribución de los vértices tanto como sea posible.

Igualmente, podemos obligar que se cumplan ciertas restricciones en las etapas del método. Durante la *planarización*, podemos requerir que se coloquen vértices en la cara externa del grafo o que no haya cortes de aristas a lo largo de un camino. En las fases de *segmentos de visibilidad* y *reemplazamiento*, podemos imponer la alineación

vertical de los vértices de un camino, la posición relativa de parejas de vértices o la forma de una determinada arista.

2.3.4 Estrategia Aumento

Aumento es una estrategia general para la construcción de trazados poligonales. La idea básica es añadir vértices y/o aristas al grafo para generar un nuevo grafo con propiedades que faciliten la construcción del trazado.

El algoritmo está compuesto por las tres fases siguientes:

1. *Planarización:* determina la *topología* del grafo a través de una *inmersión plana*. Se realizan las mismas operaciones que en el paso 1 de la estrategia Topología-Forma-Métrica.
2. *Aumento:* añade aristas ficticias hasta obtener un grafo plano maximal, es decir, todas las caras tienen tres aristas. Se intenta mantener reducido el grado de los vértices debido a que afecta parámetros estéticos como el área o la resolución angular.
3. *Trazado de la triangulación:* representa cada cara como un triángulo. Se eliminan las aristas ficticias y los vértices ficticios se sustituyen por cortes.

Durante el paso *Aumento* no es necesario generar un grafo maximal debido a la existencia de varios algoritmos que solo necesitan grafos planos triconexos o, incluso, biconexos para construir el trazado.

Como hemos visto en otras estrategias, la etapa de *planarización* minimiza el número de cortes. En las fases de *aumento* y *trazado de triangulación*, pueden existir objetivos como minimizar el área, maximizar la resolución angular o distribuir los vértices uniformemente.

Al contrario que las demás estrategias, Aumento admite menos la imposición de restricciones.

2.3.5 Estrategia Mínima Energía

Mínima Energía es un método intuitivo para crear trazados rectilíneos de grafos no dirigidos. Esta estrategia simula un sistema de fuerzas sobre un grafo y generan la configuración del estado de mínima energía.

Este enfoque tiene dos componentes:

- *Asignación de fuerzas:* asigna a cada par de vértices *(u, v)* un muelle de longitud l_{uv} igual al número de aristas del camino mínimo entre *u* y *v*. El muelle induce

una fuerza de magnitud $d_{uv} - l_{uv}$ sobre u, donde d_{uv} es proporcional a la distancia euclídea entre u y v.

- *Cálculo de la configuración de mínima energía:* utiliza técnicas de análisis numérico en vez de algoritmos combinatorios para calcular el mínimo local.

Estos métodos suelen producir trazados muy simétricos y suelen distribuir los vértices equitativamente.

También, es posible aplicar restricciones. Por ejemplo, si requerimos que un conjunto de vértices permanezcan en una región o curva determinada, se utilizarían fuerzas especiales.

2.3.6 Estrategia Divide y Vencerás

Divide y Vencerás es una estrategia que genera trazados planos y rectilíneos. La idea básica reside en los tres siguientes pasos:

1. *Descomponer el grafo en subgrafos.*
2. *Trazar los subgrafos recursivamente.*
3. *Trazar el grafo inicial uniendo el trazado de sus subgrafos.*

Los tipos de grafos que más favorecen la aplicación de este método son los árboles y los digrafos serie-paralelo, dada su facilidad para ser descompuestos en subgrafos.

También, permite la aplicación de parámetros estéticos como reducir el ancho del trazado o trazar del mismo modo los subárboles isomorfos.

3 ALGORITMOS DE REPRESENTACIÓN DE VISIBILIDAD

En el capítulo 2 hemos introducido el Trazado de Grafos haciendo énfasis en las estrategias existentes para crear trazados que tengan determinadas características. Ahora, vamos a analizar la *representación de visibilidad*, que es la base de este proyecto, junto a otros trazados que se derivan de ella.

En general, todos los algoritmos que veremos en este capítulo están ideados para obtener trazados de grafos planos y los podemos ubicar dentro de la estrategia Visibilidad. Pero no ha sido objetivo de este proyecto implementar el primer paso, la *planarización*, que nos permitiría generalizar estas técnicas y utilizar grafos no planos.

Por otro lado, tenemos que los algoritmos han sido diseñados para aprovechar las propiedades de los *st-grafos planos*, los cuales son digrafos acíclicos planos con un solo vértice fuente y un solo vértice sumidero. Antes de poder utilizar los algoritmos con grafos planos no dirigidos debemos aplicar una orientación de las aristas que genera un *st-grafo plano*.

En este capítulo explicaremos, primeramente y de forma breve, que son los *st-grafos planos* junto a algunas propiedades importantes. Seguidamente, veremos los algoritmos para crear la *representación de visibilidad*, la *representación de visibilidad con restricciones*, el *trazado poligonal ascendente*, el *trazado poligonal ascendente con restricciones*, el *trazado ortogonal plano*, el *trazado de dominancia rectilíneo* y el *trazado de dominancia poligonal*. Por último, describiremos el algoritmo *orientación bipolar* para convertir en dirigido un grafo plano no dirigido, así como, los algoritmos *ordenación topológica* y *numeración topológica*.

3.1 St-grafos Planos

Un *st-grafo plano* es un digrafo acíclico plano con un solo vértice fuente *s* y un solo vértice sumidero *t*.

Permitir que *G* sea un digrafo acíclico, sabemos que:

- *G* admite una *ordenación topológica*.
- *G* admite una *numeración topológica*.

Una *numeración topológica* de *G* es una asignación de números a los vértices de *G*, de forma tal que, para cada arista *(u, v)* de *G*, el número asignado a *v* es mayor que el asignado a *u*. Es decir, número(*v*) > número(*u*). Si las aristas del digrafo tienen pesos asignados, entonces tendremos que, para cada arista *(u, v)* de *G*, el número asignado a *v* es mayor o igual que el asignado a *u* más el peso de (*u, v*). Es decir, número(*v*) ≥

número(*u*) + peso(*u, v*). La numeración es *óptima* si el rango de números asignados a los vértices es mínimo.

Una *ordenación topológica* es una numeración topológica de *G*, de forma tal que, a cada vértice se le asigna un número entero distinto entre 1 y *n*, siendo *n* el número de vértices de *G*.

Existen varias estrategias para calcular tanto la numeración como la ordenación topológica con complejidad *O(n* + *m*), siendo *n* el número de vértices y *m* el número de aristas de *G*. Más adelante describiremos en profundidad los dos algoritmos utilizados en nuestro caso.

Permitir que *G* sea un st-grafo y en sintonía con las definiciones anteriores, tenemos que se cumple que:

- Dada una numeración topológica de *G*, cada camino dirigido de *G* recorre vértices con una numeración creciente.
- Para cada vértice *v* de *G*, existe un camino dirigido simple *P* desde *s* hasta *t* que contiene a *v*.

La primera propiedad se cumple por la forma en que los números corresponden con la dirección de las aristas. Si la segunda propiedad no fuera cierta, entonces veríamos que no existe un camino de *s* a *v* o no existe un camino de *v* a *t*. Esto implicaría que hay más de un vértice fuente o más de un vértice sumidero.

Dado que el grafo es plano, podemos definir el digrafo dual *G** asociado a *G* de la siguiente forma (Ver Ilustración 9):

- El conjunto de vértices de *G** está formado por el conjunto de caras *G*. La cara externa de *G* se divide en dos caras denominadas cara externa izquierda *s** y cara externa derecha *t**.
- Para cada arista *e* de *G*, siempre y cuando no sea la arista *(s, t)*, *G** tiene una arista *e** = (*f, g*) donde *f* es la cara a la izquierda de *e*, denominada *izquierda(e*), y *g* es la cara a la derecha de *e*, denominada *derecha(e*) (Ver Ilustración 12).

Podemos notar que el digrafo *G** también es un st-grafo plano con vértice fuente *s** y vértice sumidero *t**. Además, *G** puede tener aristas múltiples. También, al tener la cara externa duplicada, las aristas salientes van hacia la cara externa izquierda *s** y las aristas entrantes van hacia la cara externa derecha *t**.

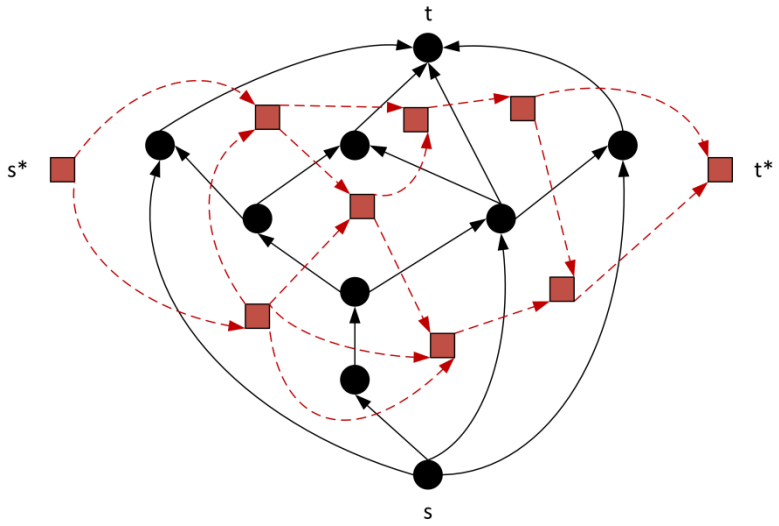

Ilustración 9: St-grafo *G* (línea continua) y digrafo dual *G** (línea discontinua)

Lema 3.1 Cada cara *f* de *G* está compuesta por dos caminos dirigidos con un origen común, denominado *origen(f)*, y un destino común, denominado *destino(f)*.

Demostración: Permitir que *f* sea una cara de *G* para la cual el lema no se cumple. Entonces existe una arista (*w, u*) en el borde de *f* dirigida desde *destino(f)* hasta *origen(f)*. Según las propiedades vistas anteriormente, hay un camino dirigido P_1 desde *u* hasta *t* y otro camino dirigido P_2 desde *s* hasta *w*. Adicionalmente, al ser *G* plano, estos dos caminos se deben intersectar en un vértice común *x*. Pero entonces *G* tendría un ciclo compuesto por: la arista (*w, u*), la parte del camino P_1 desde *u* hasta *x*, y la parte del camino P_2 desde *x* hasta *w*. Esto contradice el hecho de que *G* es un st-grafo plano (Ver Ilustración 10).

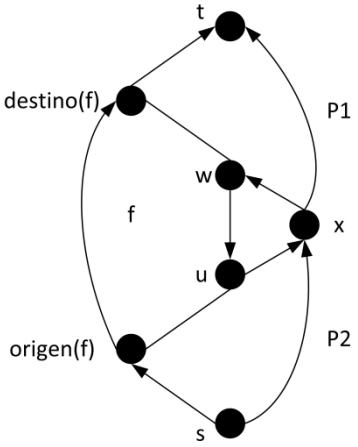

Ilustración 10: Caminos dirigidos en la demostración del Lema 3.1

Lema 3.2 Las aristas entrantes en cada vértice de *G* aparecen situadas consecutivamente. Ocurre lo mismo con las aristas salientes.

Demostración: El lema se prueba trivialmente para los vértices *s* y *t*. Permitir que *v* sea cualquier otro vértice, y suponer, por una contradicción, que existen las aristas (v, w_0), (w_1, v), (v, w_2) y (w_3, v), situadas en el sentido de las agujas del reloj alrededor de *v*. Por las propiedades vistas arriba, hay dos caminos dirigidos P_0 y P_2 desde w_0 hasta *t* y desde w_2 hasta *t*, respectivamente. Del mismo modo, hay dos caminos dirigidos P_1 y P_3 desde *s* hasta w_0 y desde *s* hasta w_2, respectivamente. Pero entonces P_0 o P_2 se debe intersectar con P_1 o P_3 en un vértice común *x*. Esto implica que *G* tiene un ciclo, lo cual contradice el hecho de que *G* es st-grafo plano (Ver Ilustración 11).

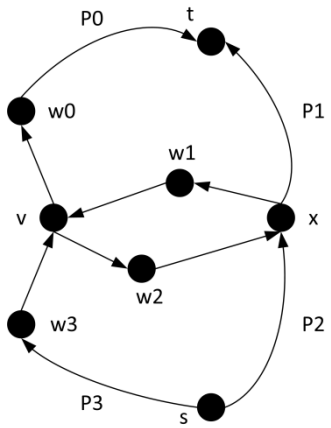

Ilustración 11 Caminos dirigidos en la demostración del Lema 3.2

La cara que separa las aristas entrantes de las salientes de un vértice *v* en el sentido de las agujas del reloj se denomina *izquierda(v)*. La cara que separa las aristas salientes de las entrantes se denomina *derecha(v)* (Ver Ilustración 12).

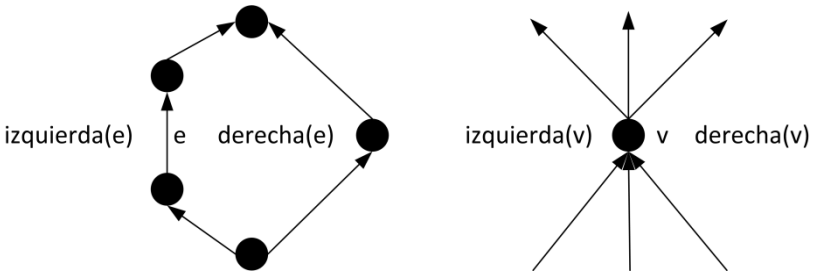

Ilustración 12: Caras izquierda y derecha de arista *e* y vértice *v*

Esta propiedad es en cierto modo dual a la propiedad descrita en el Lema 3.1. Si imaginamos un vértice en el centro de una cara *f* y las aristas duales que cruzan las aristas del grafo *G*, entonces todas las aristas entrantes de *f* aparecen consecutivamente alrededor de *f*, y lo mismo ocurre con todas las aristas salientes.

En el siguiente lema se expresa una interacción interesante que existe entre los caminos de los digrafos *G* y *G**.

Lema 3.3 Para cualquier par de caras *f* y *g* de un st-grafo plano *G*, exactamente una de las siguientes proposiciones es válida:

- *G* tiene un camino dirigido desde *destino(f)* hasta *origen(g)*.
- *G* tiene un camino dirigido desde *destino(g)* hasta *origen(f)*.
- *G** tiene un camino dirigido desde *f* hasta *g*.
- *G** tiene un camino dirigido desde *g* hasta *f*.

Demostración: Considerar una ordenación topológica de *G* y, sin pérdida de generalidad, suponer que el número de *destino(f)* es menor que el número de *origen(g)*. El camino desde un vértice *v* de *G* que siempre toma la arista saliente que está situada más a la izquierda se denomina el *camino más a la izquierda* desde *v*. El *camino más a la derecha* se define de manera similar. Considerar los caminos más a la izquierda y más a la derecha de *G* desde *destino(f)* hasta *t*, y llamémosle P_1 y P_2, respectivamente. De la misma manera, permitir que P_3 y P_4 sean el camino más a la izquierda y más a la derecha de *G* desde *origen(g)* hasta *t*. Si hay un camino dirigido en *G* desde *destino(f)* hasta *origen(g)*, el lema es válido. De lo contrario, o P_2 se intersecta con P_3 (en un vértice en común) o P_1 se intersecta con P_4. Por sencillez, solo consideramos el primer caso. Permitir que *x* sea el primer vértice donde P_2 y P_3 se intersectan. Evidentemente, por el Lema 3.2, toda arista incidente en cualquier vértice del camino P_2, desde la parte derecha de P_2, es entrante. Lo mismo sucede con las aristas incidentes en P_3 desde la izquierda. Debido a la construcción de *G**, hay un camino dirigido en *G** desde *f* hasta *g* (Ver Ilustración 13).

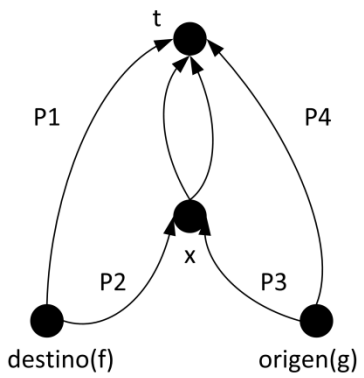

Ilustración 13 Caminos dirigidos en la demostración del Lema 3.3

Resulta que el lema anterior es un caso especial de una propiedad más general de los st-grafos planos, que establece un orden total sobre los vértices, aristas y caras. Llamemos objeto a cualquier elemento que sea vértice, arista o cara. Las definiciones anteriores sobre *origen()*, *destino()*, *izquierda()*, *derecha()* pueden ser extendidas a

todos los objetos de G. Para un vértice v, *origen*(v) = *destino*(v) = v. Para una arista e = (u, v), *origen*(e) = u, *destino*(e) = v. Para una cara f, *izquierda*(f) = *derecha*(f) = f.

Lema 3.4 Para cualquier par de objetos o_1 y o_2 de un st-grafo plano G, exactamente una de las siguientes proposiciones es válida:

- G tiene un camino dirigido desde *destino*(o_1) hasta *origen*(o_2).
- G tiene un camino dirigido desde *destino*(o_2) hasta *origen*(o_1).
- G^* tiene un camino dirigido desde *derecha*(o_1) hasta *izquierda*(o_2).
- G^* tiene un camino dirigido desde *derecha*(o_2) hasta *izquierda*(o_1).

3.2 Representación de Visibilidad

Una *representación de visibilidad* Γ de una st-grafo plano G traza cada vértice v como un segmento horizontal, denominado *segmento vértice* $\Gamma(v)$, y cada arista (u, v) como un segmento vertical, denominado *segmento arista* $\Gamma(u, v)$, tal que:

- Los segmentos vértice no se solapan.
- Los segmentos arista no se solapan.
- Un segmento arista $\Gamma(u, v)$ tiene su extremo más bajo en $\Gamma(u)$, su extremo más alto en $\Gamma(v)$ y no intersecta ningún otro segmento vértice.

Los segmentos se representan sobre un sistema de coordenadas enteras, por lo tanto, tenemos una serie de coordenadas relacionadas con cada segmento. Un segmento vértice $\Gamma(v)$, por ser horizontal, está ubicado a lo largo de una coordenada del eje de ordenadas, denominada $y(\Gamma(v))$, y entre dos coordenadas del eje de abscisas, denominándose $x_L(\Gamma(v))$ la coordenada izquierda y $x_R(\Gamma(v))$ la coordenada derecha. Un segmento arista $\Gamma(u, v)$, por ser vertical, está ubicado a lo largo de una coordenada del eje de abscisas, denominada $x(\Gamma(u, v))$, y entre dos coordenadas del eje de ordenadas, denominándose $y_B(\Gamma(u, v))$ la coordenada más baja e $y_T(\Gamma(u, v))$ la coordenada más alta.

Algoritmo Visibilidad

Entrada: st-grafo plano G con n vértices.
Salida: representación de visibilidad Γ de G con coordenadas enteras y área $O(n^2)$.
Complejidad: $O(n)$
Pasos:
1. Construir el st-grafo plano G^*.
2. Calcular la numeración topológica óptima Y de G.
3. Calcular la numeración topológica óptima X de G^*.
4. Para cada vértice v de G hacer:
   4.1. Trazar $\Gamma(v)$ como un segmento horizontal con
   $y(\Gamma(v)) = Y(v)$;

$x_L(\Gamma(v)) = X(izquierda(v))$;
$x_R(\Gamma(v)) = X(derecha(v)) - 1$;

5. Para cada arista *e* de *G* hacer:
   5.1. Trazar $\Gamma(e)$ como un segmento vertical con
   $x(\Gamma(e)) = X(izquierda(e))$;
   $y_B(\Gamma(e)) = Y(origen(e))$;
   $y_T(\Gamma(e)) = Y(destino(e))$;

La demostración de que el algoritmo Visibilidad es correcto se basa en las siguientes observaciones. Por el Lema 3.4 y la construcción del algoritmo, todos los pares de segmentos vértice están separados por una franja horizontal o vertical de, al menos, una unidad de ancho. También, todos los pares de segmentos arista en los lados opuestos de una cara están separados por una franja vertical de, al menos, una unidad de ancho, y ningún par de caras se intersectan en la representación construida por el algoritmo, excepto para sus aristas en común.

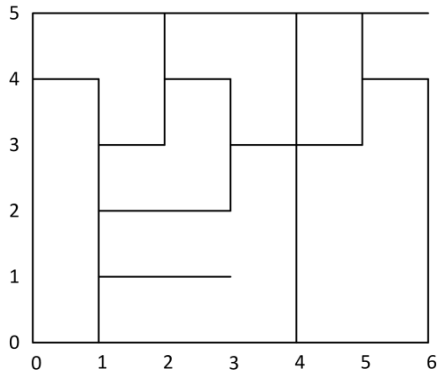

Ilustración 14: Representación de visibilidad de *G*

3.3 Representación de Visibilidad con Restricciones

En esta parte veremos una variación de la representación de visibilidad donde se obliga a que algunas aristas predefinidas se mantengan verticalmente alineadas. Esta representación se denomina *representación de visibilidad con restricciones* y es usada como punto de partida para generar trazados poligonales y ortogonales con propiedades interesantes.

Caminos disjuntos: Permitir que *G* sea un st-grafo plano con *n* vértices. Dos caminos π_1 y π_2 de *G* son *disjuntos* si no tienen aristas compartidas y no se cruzan en vértices comunes, es decir, no existe un vértice *v* en *G* con aristas e_1, e_2, e_3 y e_4 colocadas en el sentido de las agujas del reloj alrededor de *v*, tal que e_1 y e_3 estén en π_1, y e_2 y e_4 estén en π_2. También, todos los pares de caminos que no tienen vértices interiores en común son disjuntos. (Ver Ilustración 15).

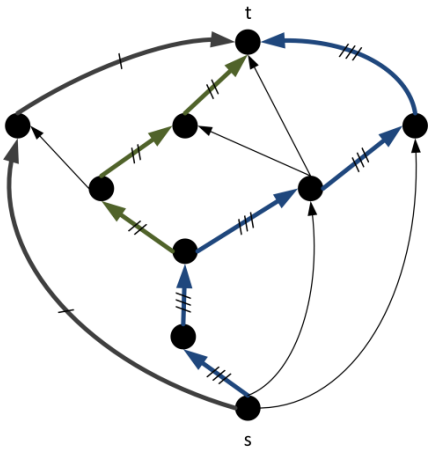

Ilustración 15: St-grafo *G* con tres caminos disjuntos

El objetivo del algoritmo que describiremos a continuación es, dado un conjunto Π de caminos de G disjuntos dos a dos, construir una representación de visibilidad Γ de G, en la cual cada camino π de Π tenga todas sus aristas alineadas verticalmente. Es decir, para todo par de aristas e' y e'' de π, se cumple que los segmentos arista $\Gamma(e')$ y $\Gamma(e'')$ tienen la misma coordenada en el eje de abscisas.

Para simplificar la descripción del algoritmo, se supone que el conjunto Π de caminos disjuntos abarca todas las aristas de G. De lo contrario, todas las aristas que no estén inicialmente en Π son añadidas como un camino formado solamente por una arista.

Algoritmo Visibilidad con restricciones

Entrada: st-grafo plano G con n vértices. Conjunto Π de caminos disjuntos que abarca todas las aristas de G.
Salida: representación de visibilidad con restricciones Γ de G con coordenadas enteras y área $O(n^2)$.
Complejidad: $O(n)$
Pasos:

1. Construir el st-grafo plano G_Π con el conjunto de vértices formado por $F \cup \Pi$ y el conjunto de aristas formado por $\{(f, \pi) \mid f = izquierda(e)$ para toda arista e del camino $\pi\} \cup \{(\pi, g) \mid g = derecha(e)$ para toda arista e del camino $\pi\}$.
2. Calcular la numeración topológica óptima Y de G, tal que $Y(s) = 0$.
3. Calcular la numeración topológica óptima X de G_Π, tal que el peso de las aristas sea igual a media unidad y $X(s^*) = -1/2$.
4. Para cada vértice v de G hacer:
   4.1. Trazar $\Gamma(v)$ como un segmento horizontal con
   $y(\Gamma(v)) = Y(v)$;
   $x_L(\Gamma(v)) = \min_{v \in \pi} X(\pi)$;
   $x_R(\Gamma(v)) = \max_{v \in \pi} X(\pi)$;

5. Para cada camino π de Π hacer:
   5.1. Para cada arista e de π hacer:
      5.1.1. Trazar $\Gamma(e)$ como un segmento vertical con
      $x(\Gamma(e)) = X(\pi)$;
      $y_B(\Gamma(e)) = Y(origen(e))$;
      $y_T(\Gamma(e)) = Y(destino(e))$;

El proceso seguido en la construcción del grafo G_Π es añadir un vértice por cada cara de G, añadir un vértice por cada camino de G, y por cada arista e de G perteneciente a un camino π, crear las aristas ($izquierda(e)$, π) y (π, $derecha(e)$). Si se da el caso de que una arista ya ha sido creada, no se vuelve a agregar a G_Π.

En la Ilustración 16 mostramos el digrafo G_Π creado a partir de G. Las caras están representadas por los vértices cuadrados y los caminos están representados por los vértices rombo.

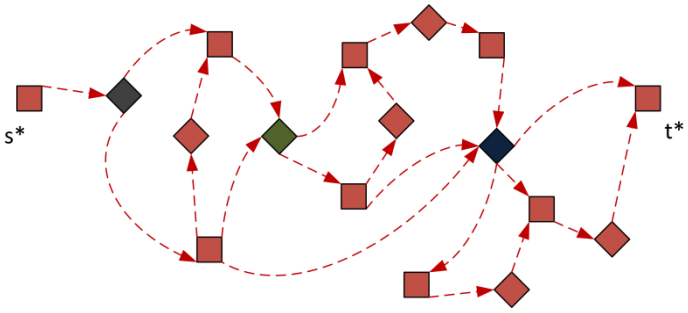

Ilustración 16: Digrafo G_Π

Observar que cada arista e de G tiene una cara izquierda y otra derecha. También, e pertenece a algún camino del conjunto Π. Cada cara interna de G tiene algún camino a su izquierda y alguno a su derecha. No hay caminos a la izquierda de s^* ni a la derecha de t^*. Por lo tanto, G_Π no tiene ciclos dirigidos y tiene un solo vértice fuente s^* y un solo vértice sumidero t^*. Obviamente, G_Π es dirigido y plano. Por último, tanto s^* como t^* están situados en la cara externa de G_Π. Podemos concluir que el digrafo G_Π construido en el paso 1 del algoritmo es un st-grafo plano.

Es interesante aclarar que en el paso 4 del algoritmo, las coordenadas en el eje de abscisas de cada vértice v se determinan encontrando tanto la numeración mínima como la máxima de todos los caminos a los que pertenece v.

Para finalizar, mostramos resultado de la aplicación del algoritmo a G. Se resaltan los caminos con líneas más gruesas.

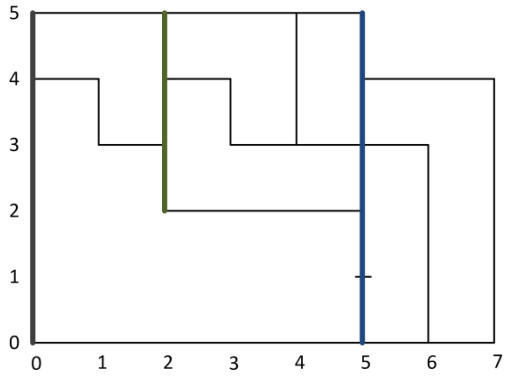

Ilustración 17: Representación de visibilidad con restricciones de *G*

3.4 Trazado Poligonal Ascendente

En esta parte vamos a ver un algoritmo que construye un trazado poligonal ascendente plano de un st-grafo plano *G* a partir de una representación de visibilidad de *G*. La estrategia que sigue el algoritmo es trazar cada vértice de *G* en un punto arbitrario de su segmento vértice y trazar cada arista (*u, v*) de *G* como una línea poligonal de tres segmentos, cuyo segmento del medio es un subconjunto del segmento arista de (*u, v*).

Algoritmo Poligonal

Entrada: st-grafo plano *G* con *n* vértices.
Salida: trazado poligonal ascendente plano de *G* con área $O(n^2)$.
Complejidad: $O(n)$
Pasos:

1. Construir una representación de visibilidad Γ de *G* con coordenadas enteras.
2. Para cada vértice *v* de *G* hacer:
   2.1. Reemplazar el segmento vértice $\Gamma(v)$ por un punto arbitrario $P(v) = (x(v), y(v))$ en $\Gamma(v)$;
3. Para cada arista $e = (u, v)$ de *G* hacer:
   3.1. Si *e* es una arista corta, es decir, $y(v) - y(u) = 1$
      3.1.1. Reemplazar el segmento arista $\Gamma(u, v)$ por un segmento desde $P(u)$ hasta $P(v)$.
   3.2. De lo contrario, es una arista larga
      3.2.1. Reemplazar el segmento arista $\Gamma(u, v)$ por una línea poligonal desde $P(u)$ hasta $P(v)$ pasando por los puntos $(x(\Gamma(u, v)), y(u) + 1)$ y $(x(\Gamma(u, v)), y(v) - 1)$.

La selección del lugar donde se ubica el punto que reemplaza un segmento vértice influye en el número de codos de las aristas del trazado. Como vimos en el capítulo anterior, unos de los parámetros de estética es minimizar el número de codos de las aristas.

La primera posibilidad que tenemos es posicionar el punto $P(v)$ en el medio del segmento vértice. Según la construcción del algoritmo, cada arista es reemplazada por una línea poligonal que tiene dos codos, a lo sumo. Por la definición de grafo plano, un st-grafo con n vértices tiene, como máximo, $3n - 6$ aristas. Por lo tanto, el número total de codos en el peor caso es $6n - 12$.

Para mejorar el resultado anterior podemos ubicar el punto $P(v)$ en la intersección del segmento vértice $\Gamma(v)$ con una arista larga incidente en v, siempre que esta exista (Ver Ilustración 18). El algoritmo Visibilidad que vimos más arriba garantiza que la representación de visibilidad de un grafo con n vértices tiene, como mínimo, $n - 1$ aristas cortas. Por consiguiente, el número máximo de codos es $(10n - 31)/3$.

Finalmente, el algoritmo utiliza el posicionamiento en las aristas largas (Ver Ilustración 18), dado que es mejor que en el punto medio.

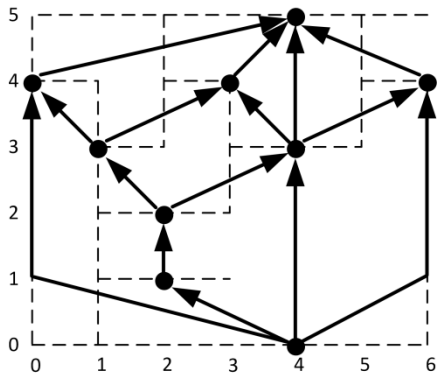

Ilustración 18: Trazado poligonal con posicionamiento en aristas largas

3.5 Trazado Poligonal Ascendente con Restricciones

El algoritmo de trazado poligonal que hemos visto en el apartado anterior puede ser extendido para utilizar la representación de visibilidad con restricciones. En este caso, cada camino no debe tener vértices internos en común con el resto, lo que los convierte en caminos disjuntos por definición.

El algoritmo descrito en esta sección consigue que todos los vértices internos de un camino estén alineados verticalmente. Esto hace que el trazado poligonal con restricciones sea interesante para la visualización de caminos específicos. Por ejemplo, podemos aplicar el trazado a un diagrama de PERT para que queden alineadas las tareas del camino crítico.

Algoritmo Poligonal con restricciones

Entrada: st-grafo plano G con n vértices. Conjunto Π de caminos sin vértices internos en común de G.
Salida: trazado poligonal ascendente plano de G, tal que, para cada camino π de Π, todos los vértices internos de π están verticalmente alineados. Área igual a $O(n^2)$.
Complejidad: $O(n)$
Pasos:

1. Construir una representación de visibilidad con restricciones Γ de G con respecto a Π.
2. Para cada vértice v de G hacer:
   2.1. Reemplazar el segmento vértice $\Gamma(v)$ por un punto $P(v) = (x(v), y(v))$ en $\Gamma(v)$ de la siguiente forma:
      2.1.1. Si v pertenece a un camino π de Π
         2.1.1.1. $x(v) = X(\pi)$;
                  $y(v) = Y(\pi)$;
      2.1.2. De lo contrario,
         2.1.2.1. Elegir cualquier punto en $\Gamma(v)$.
3. Para cada arista $e = (u, v)$ de G hacer:
   3.1. Si e es una arista corta, es decir, $y(v) - y(u) = 1$
      3.1.1. Reemplazar el segmento arista $\Gamma(u, v)$ por un segmento desde $P(u)$ hasta $P(v)$.
   3.2. De lo contrario, es una arista larga
      3.2.1. Reemplazar el segmento arista $\Gamma(u, v)$ por una línea poligonal desde $P(u)$ hasta $P(v)$ pasando por los puntos $(x(\Gamma(u, v)), y(u) + 1)$ y $(x(\Gamma(u, v)), y(v) - 1)$.

En la Ilustración 19 podemos observar que en las aristas (u, v) donde se cumple que $y(v) - y(u) = 2$, los dos puntos medios de la línea poligonal son el mismo. Cuando un vértice v no pertenece a ningún camino alineado, entonces cualquier ubicación de $P(v)$ a lo largo de $\Gamma(v)$ garantiza la validez del algoritmo y que el número de codos por arista siga siendo igual a dos, como máximo.

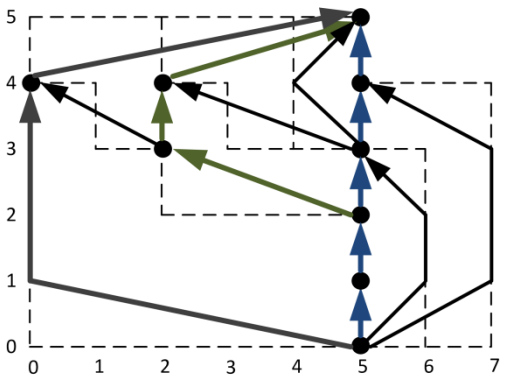

Ilustración 19: Trazado poligonal con restricciones

Como hemos visto en ocasiones anteriores, partir de la representación de visibilidad garantiza que tengamos, al menos, $n - 1$ aristas cortas o que pertenecen a un camino. También, sabemos que el número máximo de aristas es $3n - 6$. Por lo tanto, el número máximo de aristas largas que pueden contener dos codos es $2n - 5$, dando como resultado que el número de codos del trazado es $4n - 10$, a lo sumo.

3.6 Trazado Ortogonal Plano

En esta parte describiremos un algoritmo para crear el trazado ortogonal plano de un grafo plano biconexo *G* a partir de la representación de visibilidad con restricciones. El conjunto de caminos disjuntos es determinado por el propio algoritmo. La condición de que el grafo sea biconexo está impuesta por el algoritmo de orientación bipolar que se aplica como paso inicial del algoritmo para convertir un grafo no dirigido *G* en un st-grafo. Más adelante en este capítulo veremos una explicación en profundidad del algoritmo de orientación bipolar que utilizamos.

Otra restricción que debe cumplir el grafo inicial es que solo puede tener vértices de grado menor o igual a cuatro. Esto viene dado por las características del trazado ortogonal que, como vimos en la introducción al trazado de grafos, solo permite representar las aristas utilizando segmentos horizontales y verticales.

Algoritmo Ortogonal

Entrada: grafo biconexo plano *G* con *n* vértices de grado menor o igual a cuatro.
Salida: trazado ortogonal plano de *G* con área igual a $O(n^2)$.
Complejidad: $O(n)$
Pasos:

1. Construir un st-grafo *D* a partir de una orientación bipolar de *G*.
2. Crear un conjunto de $n - 2$ caminos dirigidos de *D* asociados con los vértices distintos de *s* y *t*. El proceso es el siguiente:
   Para cada vértice *v* de *D* distinto de *s* y *t* hacer:
   2.1. Crear un camino π_v con dos aristas *e'* y *e''*.
      2.1.1. Si *v* tiene solo dos aristas entrantes,
         2.1.1.1. *e'* es la arista entrante situada más a la izquierda de *v*.
            e'' es la arista saliente situada más a la derecha de *v*.
      2.1.2. Si *v* tiene una o tres aristas entrantes,
         2.1.2.1. *e'* es la arista entrante situada en el medio de las otras aristas entrantes de *v*.
            e'' es la arista saliente situada en el medio de las otras aristas salientes de *v*.
   2.2. Unificar los caminos que compartan aristas, lo cual produce un conjunto *Π* de caminos disjuntos.
3. Construir una representación de visibilidad con restricciones *Γ* de *D* con respecto a *Π*.

4. Para cada vértice *v* de *D* hacer:
   4.1. Si *v* es distinto de *s* y *t*,
      4.1.1. Trazar *v* en la intersección *P*(*v*) del segmento vértice *Γ*(*v*) con los segmentos arista del camino π_v.
   4.2. Si *v* es el vértice *s*,
      4.2.1. Trazar *v* en la intersección *P*(*v*) del segmento vértice *Γ*(*v*) con el segmento arista de la arista saliente situada en el medio de las demás.
   4.3. Si *v* es el vértice *t*,
      4.3.1. Trazar *v* en la intersección *P*(*v*) del segmento vértice *Γ*(*v*) con el segmento arista de la arista entrante situada en el medio de las demás.
5. Para cada arista *e* = (*u, v*) de *D* hacer:
   5.1. Si *u* y *v* son distintos de *s* y *t*,
      5.1.1. Trazar *e* como una línea poligonal que pasa por los puntos:
      P(*u*), la intersección del segmento vértice *Γ*(u) con el segmento arista *Γ*(*e*), la intersección del segmento arista *Γ*(*e*) con el segmento vértice *Γ*(*v*), y *P*(*v*). La línea poligonal está formada por tres segmentos donde el primero y el último pueden estar vacíos.
   5.2. Si *u* o *v* es *s*,
      5.2.1. Trazar *e* como una línea poligonal que pasa por los puntos:
      P(*u*), la intersección del segmento vértice *Γ*(*u*) con el segmento arista *Γ*(*e*), la intersección del segmento arista *Γ*(*e*) con el segmento vértice *Γ*(*v*), y *P*(*v*). Si el vértice *s* tiene grado cuatro, se deben resolver los cortes entre las aristas.
   5.3. Si *u* o *v* es *t*,
      5.3.1. Trazar *e* como una línea poligonal que pasa por los puntos:
      P(*u*), la intersección del segmento vértice *Γ*(*u*) con el segmento arista *Γ*(*e*), la intersección del segmento arista *Γ*(*e*) con el segmento vértice *Γ*(*v*), y *P*(*v*). Si el vértice *t* tiene grado cuatro, se deben resolver los cortes entre las aristas.

Las figuras en la Ilustración 20 muestran cómo se trazan los vértices y las aristas en los pasos cuatro y cinco del algoritmo según los tipos de vértices, número de aristas entrantes y salientes, etc. En el cuarto caso vemos que cuando una arista parte de *s* o finaliza en *t* y estos vértices tienen grado cuatro, es necesario resolver el corte entre aristas que se produce.

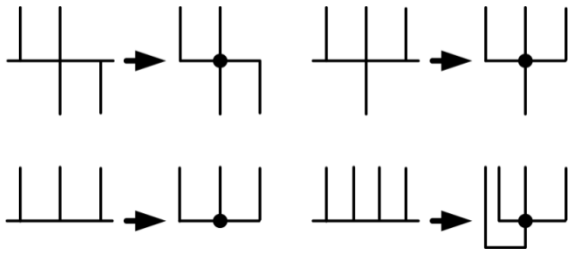

Ilustración 20: Trazado de vértices y aristas en los pasos 4 y 5 del Algoritmo Ortogonal

Para resolver el corte, partimos del hecho de que la ubicación del vértice *s* o *t* cuando tienen grado cuatro siempre es en la coordenada $x(\Gamma(e))$, siendo *e* la tercera arista de izquierda a derecha. Por esta razón, la arista que causa problemas es la primera, o lo que es lo mismo, es la arista *e* cuya coordenada $x(\Gamma(e))$ es igual a la coordenada $x_L(\Gamma(v))$, siendo *v* el vértice *s*, si *e* parte de *s*, o *t*, en caso de que *e* llegue a *t*. Basados en estas observaciones, cuando se determinan los codos de la arista problemática se agrega el punto extra que evita el corte.

Para analizar el número máximo de codos del trazado ortogonal creado por el algoritmo, debemos notar que hay, como máximo, dos codos por cada vértice distinto de *s* y *t*, y cuatro codos, a lo sumo, por los vértices *s* y *t*. Por consiguiente, el número máximo de codos es $2n + 4$.

Por las restricciones que impone este algoritmo, usaremos un grafo diferente al que hemos venido utilizando para mostrar el resultado de la ejecución (Ver Ilustración 21 e Ilustración 22).

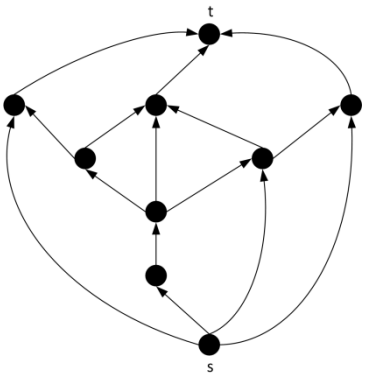

Ilustración 21: Grafo *G* para algoritmo ortogonal

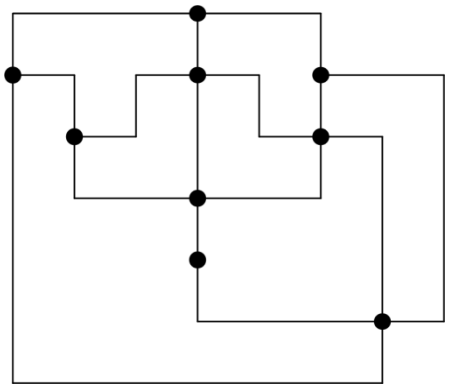

Ilustración 22: Trazado ortogonal

3.7 Trazado de Dominancia Rectilíneo

Un *trazado de dominancia* de un digrafo *G* es un trazado Γ de *G*, tal que, para cada par de vértices *u* y *v*, hay un camino dirigido desde *u* hasta *v* en *G*, si y solo si $x(u) \leq x(v)$ e $y(u) \leq y(v)$ en Γ. Observar que estas dos condiciones no se pueden cumplir al mismo tiempo en el caso de la igualdad dado que dos vértices distintos deben estar situados en puntos diferentes. Los trazados de dominancia tienen la capacidad de describir el cierre transitivo de un digrafo a través de la relación de dominancia geométrica entre los vértices.

En esta parte vamos a ver un algoritmo para generar el trazado de dominancia rectilíneo a partir de un digrafo reducido. Permitir que *G* sea un digrafo acíclico. Una arista (*u, v*) de *G* es *transitiva* si hay otro camino dirigido en *G* que permita ir desde *u* hasta *v*. Un digrafo acíclico es *reducido* si no tiene aristas transitivas. Observar que si

eliminamos todas las aristas transitivas de *G*, obtenemos un digrafo reducido con el mismo cierre transitivo que *G*.

El algoritmo se realiza en tres fases. La primera fase, *Preprocesamiento*, construye una estructura de datos. La segunda fase, *Trazado Preliminar*, asigna a cada vértice *v* una coordenada *X* e *Y* distinta en el rango [0, *n* − 1]. La tercera fase, *Compactación*, ajusta la posición de los vértices para reducir el área del trazado. La fase de Trazado Preliminar realiza dos ordenaciones topológicas de los vértices de *G*, las cuales exploran los vértices sucesores de *v* de izquierda a derecha y de derecha a izquierda, respectivamente. La fase de Compactación explora los vértices según el orden establecido por las coordenadas preliminares *X* e *Y*. (Ver Ilustración 23).

Algoritmo Dominancia rectilíneo

Entrada: st-grafo plano reducido *G* con *n* vértices.
Salida: trazado de dominancia rectilíneo *Γ* de *G* con área igual a $O(n^2)$.
Complejidad: $O(n)$
Pasos:

1. *Preprocesamiento*: Construir una estructura de datos para *G*, donde cada vértice *v* tiene asociado una lista de sus aristas salientes ordenadas según el sentido de las agujas del reloj. Esta lista es doblemente enlazada por medio de los punteros *siguiente*(*e*) y *anterior*(*e*), y es accedida a través de los punteros *primeraSaliente*(*v*) y *últimaSaliente*(*v*) hacia sus aristas salientes más a la izquierda y más a la derecha, respectivamente. Además, *v* tiene los punteros *primeraEntrante*(*v*) y *últimaEntrante*(*v*) hacia sus aristas entrantes más a la izquierda y más a la derecha, respectivamente. Por último, cada arista *e* = (*u*, *v*) mantiene un puntero *cabecera*(*e*) hacia *v*.
2. *Trazado Preliminar*: Asignar las coordenadas preliminares *X* e *Y*.
   2.1. Asignar la coordenada preliminar *X*.
   Hacer *contador* := 0 e invocar *etiquetaX*(*s*).

```
procedure etiquetaX(vértice v)
begin
    X(v) := contador;
    contador := contador + 1;
    if v ≠ t then
        e = primeraSaliente(v);
        repeat
            w = cabecera(e);
            if e = últimaEntrante(w) then
                etiquetaX(w);
            end if;
            e = siguiente(e);
        until (e = nil);
```

end if;
end *etiquetaX***;**

2.2. Asignar la coordenada preliminar *Y*.
Hacer *contador* := 0 e invocar *etiquetaY*(*s*).

procedure *etiqueta*Y(**vértice** *v*)
begin
Y(*v*) := *contador*;
contador := *contador* + 1;
if *v* ≠ *t* **then**
e = *últimaSaliente*(*v*);
repeat
w = *cabecera*(*e*);
if *e* = *primeraEntrante*(*w*) **then**
etiquetaY(*w*);
end if;
e = *anterior*(*e*);
until (*e* = *nil*);
end if;
end *etiquetaY***;**

3. *Compactación*: Asignar las coordenadas finales *x* e *y*.
3.1. Construir dos listas de vértices ordenados de menor a mayor según las coordenadas *X* e *Y* a través de los punteros *siguienteX*(*v*) y *siguienteY*(*v*).
3.2. Asignar la coordenada final *x*.
Permitir que *u* sea el vértice para el cual *X*(*u*) = 0.
x(*u*) = 0;
while *siguienteX*(*u*) ≠ *nil* **do**
v = *siguienteX*(*u*);
if Y(*u*) > Y(*v*) **or** (*primeraSaliente*(*u*) = *últimaSaliente*(*u*) **and** *primeraEntrante*(*v*) = *últimaEntrante*(*v*)) **then**
x(*v*) = *x*(*u*) + 1;
else
x(*v*) = *x*(*u*);
end if;
u = *v*;
end do;
3.3. Asignar la coordenada final *y*.
Permitir que *u* sea el vértice para el cual *Y*(*u*) = 0.
y(*u*) = 0;
while *siguienteY*(*u*) ≠ *nil* **do**
v = *siguienteY*(*u*);
if X(*u*) > X(*v*) **or** (*primeraSaliente*(*u*) = *últimaSaliente*(*u*) **and** *primeraEntrante*(*v*) = *últimaEntrante*(*v*)) **then**

```
            y(v) = y(u) + 1;
        else
            y(v) = y(u);
        end if;
        u = v;
    end do;
```

En general, cuando tenemos dos vértices *u* y *v* con coordenadas preliminares *X* consecutivas, la coordenada final *x* no se incrementa si existe la arista (*u*, *v*). Por el contrario, si no existe dicha arista, la coordenada *x* se incrementa. No obstante, cuando (*u*, *v*) es la única arista saliente de *u* y la única entrante a *v*, la coordenada final *x* se incrementa. Esto se realiza con el objetivo de evitar que se le asignen las mismas coordenadas a dos vértices distintos. En el caso de las coordenadas *y* se realiza un análisis similar.

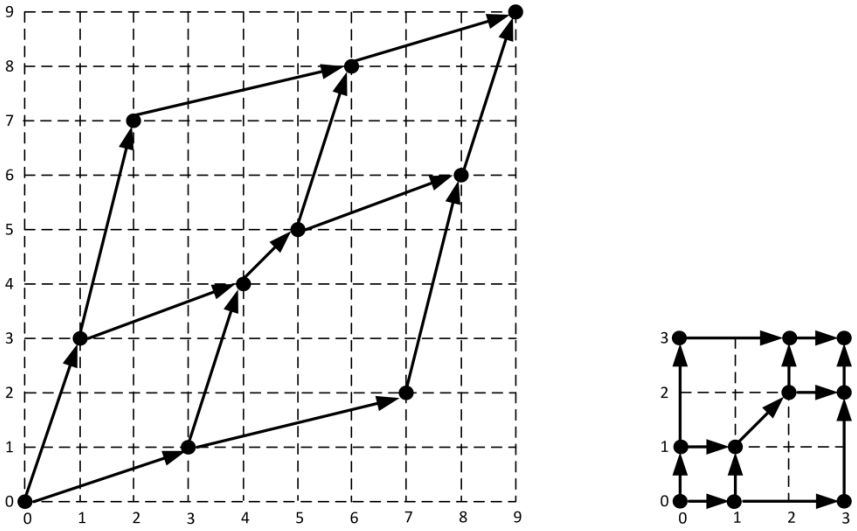

Ilustración 23: Trazado de dominancia rectilíneo (preliminar y final)

3.8 Trazado de Dominancia Poligonal

El algoritmo que construye el trazado de dominancia rectilíneo puede ser modificado para funcionar con st-grafos planos mediante la inserción de vértices ficticios en cada arista transitiva. El resultado es un algoritmo que construye un trazado poligonal con un número máximo de codos menor al establecido por el algoritmo que genera el trazado poligonal a partir de la representación de visibilidad.

Algoritmo Dominancia poligonal

Entrada: st-grafo plano *G* con *n* vértices.
Salida: trazado de dominancia poligonal *Γ* de *G* con área igual a $O(n^2)$.

Complejidad: O(*n*)
Pasos:

1. Si *G* no es reducido, sustituir cada arista transitiva (*u*, *v*) por un nuevo vértice *x* y dos aristas (*u*, *x*) y (*x*, *v*). Denominar *G'* al st-grafo plano reducido resultante.
2. Construir un trazado de dominancia rectilíneo *Γ'* de *G'* utilizando el Algoritmo Dominancia rectilíneo.
3. Reemplazar cada vértice ficticio introducido en el paso 1 por un codo en la arista original.

El número de codos introducidos en el trazado por este algoritmo es igual al número de aristas transitivas en el st-grafo plano *G* de entrada. En un digrafo acíclico con *n* vértices, hay, al menos, *n* – 1 aristas no transitivas. Como *G* es plano, el número máximo de aristas es 3*n* – 6. Por lo tanto, *G* tiene 2*n* – 5 aristas transitivas, a lo sumo.

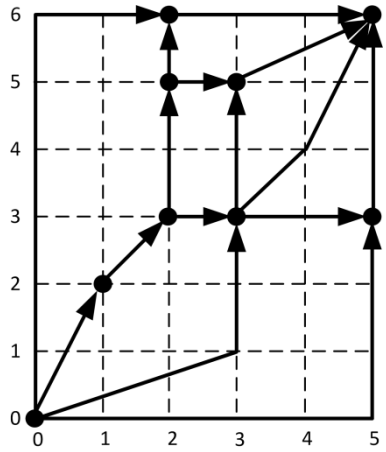

Ilustración 24: Trazado de dominancia poligonal

3.9 Algoritmo Orientación Bipolar

Los algoritmos que hemos visto en las secciones anteriores han estado limitados a los st-grafos, pero pueden ser extendidos para funcionar con grafos planos no dirigidos. Permitir que *G* sea un grafo no dirigido plano biconexo con dos vértices distintos *s* y *t* situados en la cara externa. *G* tiene *n* vértices y *m* aristas. Una *numeración-st* de *G* es una numeración v_1, v_2, …, v_n de los vértices de *G*, tal que $s = v_1$, $t = v_n$ y cada vértice v_j, excepto *s* y *t*, es adyacente al menos a dos vértices v_i y v_k con $i < j < k$. Esta numeración puede ser construida con una complejidad *O*(*n* + *m*) [Tar85].

Una *orientación bipolar* de *G* es una orientación de las aristas de *G* que produce un digrafo acíclico tal que *s* es el único vértice fuente (no tiene aristas entrantes) y *t* es el único vértice sumidero (no tiene aristas salientes). *G* tiene una orientación bipolar si y solo si tiene una numeración-st. Dada una numeración-st de *G*, orientamos cada arista de *G* desde el vértice de menor numeración hacia el vértice de mayor numeración.

El algoritmo que determina la numeración-st recibe como entrada el conjunto de vértices *V* y los vértices *s* y *t*. Está compuesto por dos fases. La primera fase es una búsqueda en profundidad durante la cual se determina:

- *pre*(*v*): orden en el que los vértices son visitados por primera vez durante la búsqueda, denominado numeración preorden del árbol generador.
- *bajo*(*v*): vértice de menor numeración alcanzable desde *v* a través de un camino formado por cero o más aristas del árbol seguido por al menos una arista de retroceso. Es seguro que el vértice *bajo*(*v*) es un antecesor de *v* en el árbol generador.
- *padre*(*v*): padre de cada vértice *v* en el árbol generador.
- *preorden*: lista de los vértices en preorden, distintos de *s* y *t*.

En la segunda fase se construye una lista *L* de vértices, de tal forma que tendremos una numeración-st si los vértices son numerados en el orden en que ellos aparecen en *L*. En este paso se realiza un recorrido en preorden del árbol generador durante el cual se determina el *signo* de cada vértice *u* que sea antecesor del vértice actual *v*. El *signo* será *menos* si *u* precede a *v* en *L* o *más* si *u* sucede a *v* en *L*. Inicialmente *L* contiene los vértices *s* y *t*, y *s* tiene signo *menos*.

Pseudocódigo Numeración-st

```
map pre, bajo, padre, signo, numeración-st;
list L, preorden;
integer actual;
boolean más, menos;

map function numeración-st (set V, vértice s, t)
begin
    for each v in V do
        pre(v) := 0;
    end do;
    actual := 1;
    pre(s) := actual;
    preorden := [];
    dfs(t);
    L = [s, t];
    más := true;
    menos := false;
    signo(s) := menos;
    for each v in preorden do
        if signo(bajo(v)) = menos then
            insertar v antes de padre(v) en L;
```

```
                    signo(padre(v)) := más;
                else if signo(bajo(v)) = más then
                    insertar v después de padre(v) en L;
                    signo(padre(v)) := menos;
                end if;
        end do;
        actual := 0;
        for each v in L do
                numeración-st(v) := actual;
                actual  := actual + 1;
        end do;
        return numeración-st;
end numeración-st;

procedure dfs(vértice v)
begin
        pre(v) := actual;
        actual  := actual + 1;
        bajo(v) := v;
        for each w in adyacentes(v) do
                if pre(w) = 0 then
                    dfs(w);
                    padre(w) := v;
                    insertar w en preorden;
                    if pre(bajo(w)) < pre(bajo(v)) then
                        bajo(v) := bajo(w);
                    end if;
                else if pre(w) ≠ 0 and pre(w) < pre(bajo(v)) then
                    bajo(v) := w;
                end if;
        end do;
end dfs;
```

Pseudocódigo Orientación Bipolar

```
procedure orientación-bipolar (grafo g)
begin
        map numeración-st;
        vértice s, t;
        s := seleccionar vértice fuente de g;
        t := seleccionar vértice sumidero de g;
        numeración-st := numeración-st(vértices(g), s, t);
        for each e in aristas(g) do
                if numeración-st(origen(e)) < numeración-st(destino(e)) then
```

```
                dirigir la arista e desde origen(e) hasta destino(e);
            else
                dirigir la arista e desde destino(e) hasta origen(e);
            end if;
        end do;
end orientación-bipolar;
```

3.10 Algoritmos de Ordenación y Numeración Topológica

Como explicamos anteriormente en este capítulo, una *numeración topológica* de un digrafo acíclico *G* es una asignación de números a los vértices de *G*, cumpliéndose para cada arista *(u, v)* que el número asignado a *v* es mayor que el asignado a *u*. Si debemos tener en cuenta el peso de las aristas, entonces se cumplirá que el número asignado a *v* es mayor o igual que el asignado a *u* más el peso de (*u, v*). Una *ordenación topológica* es una numeración topológica de *G*, de forma tal que, a cada vértice se le asigna un número entero distinto entre 1 y *n*, siendo *n* el número de vértices de *G*.

El algoritmo para determinar la ordenación topológica hace uso de la búsqueda en profundidad. La idea es visitar cada vértice *v* mientras no haya sido recorrido anteriormente y asignar un orden a *v* solo cuando se han visitado todos los vértices accesibles desde *v* [Tar76]. La complejidad del algoritmo es *O*(*n* + *m*) siendo *n* el número de vértices y *m* el número de aristas de *G*. El valor de *orden* se inicializa al máximo valor que tendrá un vértice en la ordenación, el cual depende del número de vértices, el peso de las aristas y el valor inicial del vértice fuente. *Orden* se va decrementando según el peso de las aristas a medida que se van visitando nuevos vértices.

Pseudocódigo Ordenación Topológica

```
map ordenación;
list visitados;
float orden;

map function ordenación-topológica (set V, vértice s)
begin
    orden := |V| * peso + inicial;
    dfs(s);
    for each v in V do
        if visitados no contiene a v then
            dfs(v);
        end if;
    end do;
    return ordenación;
end ordenación-topológica;
```

```
procedure dfs(vértice v)
begin
        insertar v en visitados;
        for each w in adyacentes(v) do
                if visitados no contiene a w then
                        dfs(w);
                end if;
        end do;
        orden := orden – peso;
        ordenación(v) := orden;
end dfs;
```

La estrategia que sigue el algoritmo de numeración topológica es asignar a cada vértice la longitud del camino dirigido más largo que termina en ese vértice. Para calcular esta longitud, se parte del vértice fuente y se recorren los vértices del grafo según la ordenación topológica, asegurando que los vértices adyacentes del vértice actual tienen una numeración superior [Sed11]. La complejidad del algoritmo es $O(n + m)$ siendo n el número de vértices y m el número de aristas de G. Un ejemplo de la aplicación del algoritmo lo podemos ver en la Ilustración 25.

Pseudocódigo Numeración Topológica

```
map function numeración-topológica (set V, vértice s)
begin
        map numeración¸ ordenación;
        for each v in V do
                numeración(v) := inicial;
        end do;
        ordenación := ordenación-topológica(V, s);
        for each v in ordenación do
                for each w in adyacentes(v) do
                        if numeración(w) ≤ numeración(v) then
                                numeración(w) := numeración(v) + peso;
                        end if;
                end do;
        end do;
        return numeración;
end numeración-topológica;
```

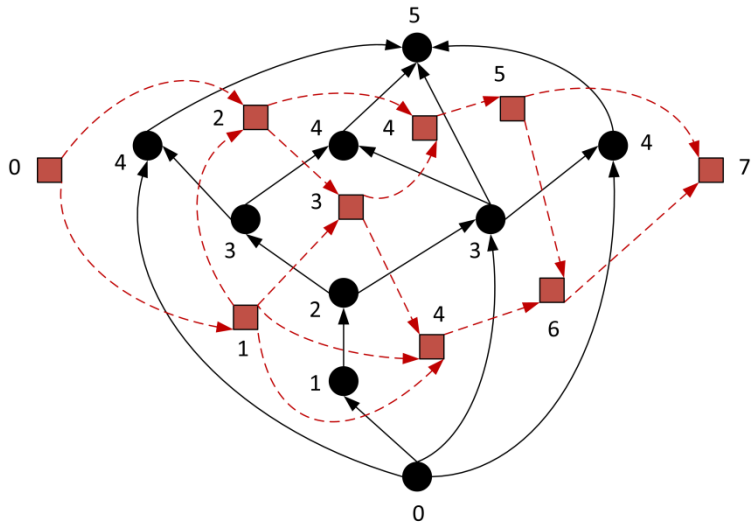

Ilustración 25: Numeración topológica de los st-grafos *G* y *G**

4 DISEÑO E IMPLEMENTACIÓN DE LA APLICACIÓN

En este capítulo documentaremos los pasos fundamentales llevados a cabo durante el diseño e implementación de la aplicación. Todo el trabajo se ha realizado siguiendo el paradigma de desarrollo Orientado a Objetos unido al lenguaje de programación Java. También, hemos aplicado una metodología de desarrollo basada en el Proceso Unificado.

La programación Orientada a Objetos permite modelar un problema abstrayendo los conceptos, entidades y objetos del mundo real en lo que se denomina objetos en el software. El sistema se estructura alrededor de los objetos y sus interacciones. Este paradigma ofrece ventajas como la abstracción, la encapsulación, la reutilización, la herencia y el polimorfismo que facilitan el diseño, a diferencia de otros paradigmas como el estructurado.

El lenguaje de programación Java nos brinda varias ventajas. Específicamente para nuestro proyecto, nos beneficia que sea Orientado a Objetos, así como que permita ejecutar el software construido en cualquier plataforma que soporte la máquina virtual de Java. Además, nos ayuda que disponga de un amplio conjunto de API (Interfaz de Programación de Aplicaciones), las cuales facilitan el uso de varias estructuras de datos, el manejo de gráficos en 2D, el desarrollo de la interfaz gráfica a través de Swing, la internacionalización de la aplicación, la manipulación de imágenes, etc. [Ora11]. Por último, Java tiene una extensa documentación de todas sus clases junto a sus métodos y atributos [Ora93].

El Proceso Unificado [Lar01] es una metodología de desarrollo de software iterativa e incremental. Utiliza los casos de uso para capturar los requisitos funcionales y modela el software visualmente a través del UML (Lenguaje de Modelado Unificado). Durante las distintas fases del proceso se generan una gran cantidad de diagramas y modelos que especifican detalladamente todo el diseño, pero en nuestro caso solo vamos a documentar las partes fundamentales que nos permiten entender globalmente el sistema.

Como hemos mencionado en anteriores ocasiones, el desarrollo del sistema parte de la reutilización del proyecto [Beh11]. Principalmente, sacamos beneficio de la implementación de la interfaz gráfica y de la arquitectura del sistema. Aunque mantuvimos los rasgos fundamentales, realizamos una remodelación y adaptación a los requisitos de nuestro proyecto. Esto fue posible gracias a un estudio profundo del proyecto original, lo cual nos permitió hacer las distintas modificaciones sin afectar al correcto funcionamiento del sistema.

Dentro de la documentación del diseño que veremos más adelante, no haremos especial hincapié en aquellas funcionalidades que reusamos. Por ejemplo, todas las

operaciones de edición de grafos, vértices y aristas quedan fuera. Al igual que otras funcionalidades como deshacer, rehacer, copiar, pegar, cortar, seleccionar objetos. Tampoco se incluyen operaciones más generales como cerrar, ayuda, imprimir, navegación entre ventanas o zoom.

La explicación del diseño la hemos dividido en diseño de alto nivel, diseño de bajo nivel, pruebas y funcionalidades extras. Dentro del diseño de alto nivel veremos los casos de uso junto a la arquitectura del sistema y al diagrama de clases. En el diseño de bajo nivel explicaremos a nivel de pseudocódigo otros algoritmos relevantes que hemos tenido que implementar. Dentro de las funcionalidades extras detallamos algunos aspectos adicionales que integran la aplicación y que resultan de gran interés.

4.1 Diseño de Alto Nivel

4.1.1 Descripción de los casos de uso

Los casos de uso son un mecanismo que permite capturar y describir los requisitos, especialmente los funcionales, del sistema de una forma sencilla y comprensible. Dentro de un caso de uso se describe una secuencia de eventos de un actor que utiliza un sistema para completar una operación. Así, en estas historias de uso del sistema se ejemplifican e incluyen implícitamente los requisitos.

Dentro de la descripción incluiremos el diagrama de casos de uso, los actores y la descripción de casos de uso en formato completo. En el diagrama se especifica gráficamente como nombramos los casos de uso y los actores, así como las relaciones existentes entre estos. Además, es un diagrama que sirve para definir los límites del sistema y la interacción con los agentes externos. Un actor es una entidad externa del sistema que interactúa con él, tanto iniciando como participando en algún caso de uso. Por último, en una descripción en formato completo se especifican detalladamente todos los pasos y las posibles variaciones que componen un caso de uso.

4.1.1.1 Diagrama de casos de uso

En esta sección mostraremos todos los casos de uso identificados durante las fases iniciales de análisis y diseño de la aplicación. Hemos separado los casos de uso en varios diagramas aunque todos forman en realidad uno solo. En la Ilustración 26 vemos los casos de uso que son específicos de este sistema. En la Ilustración 27 y la Ilustración 28 vemos los casos de uso que son, en gran medida, comunes a la mayoría de proyectos relacionados con grafos y cuya implementación hemos reutilizado.

Ilustración 26: Diagrama de Casos de Uso. Parte 1

Ilustración 27: Diagrama de Casos de Uso. Parte 2

Ilustración 28: Diagrama de Casos de Uso. Parte 3

4.1.1.2 Actores

El sistema, denominado “Visibility Representations Step-By-Step”, tiene solo un actor principal llamado *Usuario*.

Usuario: es la persona que interactúa con nuestro sistema con los objetivos de crear un grafo, editarlo, guardarlo y abrirlo. También, es el que está interesado en ejecutar los algoritmos paso a paso.

4.1.1.3 Descripción de los casos de uso en formato extendido

El formato extendido para describir un caso de uso está compuesto, fundamentalmente, por los siguientes campos:

- *Identificador*: Caso de Uso Número: Nombre
- *Actor Principal*: Actor principal que utiliza el sistema para alcanzar un objetivo.
- *Personal Involucrado e Intereses*: Sugiere y delimita qué es lo que debe hacer el sistema.
- *Precondiciones*: Las condiciones que deben cumplirse antes de comenzar un escenario en el caso de uso.
- *Garantías de Éxitos*: Que debe cumplirse cuando el caso de uso se completa con éxito.
- *Garantías de Fracaso*: Que debe cumplirse cuando el caso de uso no se completa satisfactoriamente.
- *Activa*: Acción que inicia el caso de uso.
- *Escenario Principal de Éxito*: Describe el camino de éxito principal que satisface los intereses del personal involucrado.
- *Extensiones*: Indican todos los otros escenarios o bifurcaciones tanto de éxito como de fracaso.

A continuación, veremos las descripciones de los casos de uso:

- **Caso de Uso 1: Ejecutar Algoritmo Visibilidad**
  *Actor Principal***:** Usuario.
  Personal Involucrado e Intereses:
  Usuario: Quiere ejecutar el algoritmo que construye la representación de visibilidad de un st-grafo plano definido.
  *Precondiciones***:** Se ha definido un grafo plano.
  *Garantías de Éxito***:** Se ejecuta el algoritmo correctamente y se deja preparada la ejecución paso a paso.
  *Garantías de Fracaso***:** Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.

Activa: El Usuario selecciona la opción Representación de Visibilidad en el menú de Algoritmos.
Escenario Principal de Éxito:

1. El Usuario quiere aplicar el algoritmo Visibilidad a un grafo definido anteriormente.
2. El Sistema aplica el algoritmo especificado al grafo y deja preparada la ejecución paso a paso.

Extensiones:

2a. El grafo no es un st-grafo:

1. El Sistema muestra el error y termina.

- **Caso de Uso 2: Ejecutar Algoritmo Visibilidad con restricciones**
  Actor Principal: Usuario.
  Personal Involucrado e Intereses:
  Usuario: Quiere ejecutar el algoritmo que construye la representación de visibilidad con restricciones de un st-grafo plano definido, dado un conjunto de caminos disjuntos especificado.
  Precondiciones: Se ha definido un grafo plano. Se ha seleccionado un conjunto de caminos disjuntos sobre este grafo.
  Garantías de Éxito: Se ejecuta el algoritmo correctamente y se deja preparada la ejecución paso a paso.
  Garantías de Fracaso: Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.
  Activa: El Usuario selecciona la opción Representación de Visibilidad con Restricciones en el menú de Algoritmos.
  Escenario Principal de Éxito:
  1. El Usuario quiere aplicar el algoritmo Visibilidad con restricciones a un grafo definido anteriormente.
  2. El Sistema aplica el algoritmo especificado al grafo y deja preparada la ejecución paso a paso.

  Extensiones:

  2a. El grafo no es un st-grafo:

  1. El Sistema muestra el error y termina.

- **Caso de Uso 3: Ejecutar Algoritmo Poligonal**
  Actor Principal: Usuario.
  Personal Involucrado e Intereses:

Usuario: Quiere ejecutar el algoritmo que construye el trazado poligonal de un st-grafo plano definido.
Precondiciones: Se ha definido un grafo plano.
Garantías de Éxito: Se ejecuta el algoritmo correctamente y se deja preparada la ejecución paso a paso.
Garantías de Fracaso: Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.
Activa: El Usuario selecciona la opción Trazado Poligonal en el menú de Algoritmos.
Escenario Principal de Éxito:

1. El Usuario quiere aplicar el algoritmo Poligonal con restricciones a un grafo definido anteriormente.
2. El Sistema aplica el algoritmo especificado al grafo y deja preparada la ejecución paso a paso.

Extensiones:

2a. El grafo no es un st-grafo:

1. El Sistema muestra el error y termina.

- **Caso de Uso 4: Ejecutar Algoritmo Poligonal con restricciones**

Actor Principal: Usuario.
Personal Involucrado e Intereses:
Usuario: Quiere ejecutar el algoritmo que construye el trazado poligonal con restricciones de un st-grafo plano definido, dado un conjunto de caminos sin vértices internos en común especificado.
Precondiciones: Se ha definido un grafo plano. Se ha seleccionado un conjunto de caminos sin vértices internos en común sobre este grafo.
Garantías de Éxito: Se ejecuta el algoritmo correctamente y se deja preparada la ejecución paso a paso.
Garantías de Fracaso: Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.
Activa: El Usuario selecciona la opción Trazado Poligonal con Restricciones en el menú de Algoritmos.
Escenario Principal de Éxito:

1. El Usuario quiere aplicar el algoritmo Poligonal con restricciones a un grafo definido anteriormente.
2. El Sistema aplica el algoritmo especificado al grafo y deja preparada la ejecución paso a paso.

Extensiones:

2a. El grafo no es un st-grafo:
1. El Sistema muestra el error y termina.

- **Caso de Uso 5: Ejecutar Algoritmo Ortogonal**
  *Actor Principal***:** Usuario.
  Personal Involucrado e Intereses:
  Usuario: Quiere ejecutar el algoritmo que construye el trazado ortogonal de un grafo biconexo plano y con todos sus vértices de grado menor o igual a cuatro.
  *Precondiciones***:** Se ha definido un grafo plano.
  *Garantías de Éxito***:** Se ejecuta el algoritmo correctamente y se deja preparada la ejecución paso a paso.
  *Garantías de Fracaso***:** Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.
  *Activa***:** El Usuario selecciona la opción Trazado Ortogonal en el menú de Algoritmos.
  *Escenario Principal de Éxito***:**
  1. El Usuario quiere aplicar el algoritmo Ortogonal a un grafo definido anteriormente.
  2. El Sistema aplica el algoritmo especificado al grafo y deja preparada la ejecución paso a paso.

  Extensiones:
  2a. El grafo no es biconexo:
  1. El Sistema muestra el error y termina.

  2b. El grafo tiene algún vértice de grado mayor a cuatro:
  1. El Sistema muestra el error y termina.

- **Caso de Uso 6: Ejecutar Algoritmo Dominancia rectilíneo**
  *Actor Principal***:** Usuario.
  Personal Involucrado e Intereses:
  Usuario: Quiere ejecutar el algoritmo que construye el trazado de dominancia rectilíneo de un st-grafo plano.
  *Precondiciones***:** Se ha definido un grafo plano.
  *Garantías de Éxito***:** Se ejecuta el algoritmo correctamente y se deja preparada la ejecución paso a paso.
  *Garantías de Fracaso***:** Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.
  *Activa***:** El Usuario selecciona la opción Trazado de Dominancia Rectilíneo en el menú de Algoritmos.

*Escenario Principal de Éxito***:**

1. El Usuario quiere aplicar el algoritmo Dominancia rectilíneo a un grafo definido anteriormente.
2. El Sistema aplica el algoritmo especificado al grafo y deja preparada la ejecución paso a paso.

Extensiones:

2a. El grafo no es un st-grafo:

1. El Sistema muestra el error y termina.

2b. El grafo no es reducido:

1. El Sistema muestra el error y termina.

- **Caso de Uso 7: Ejecutar Algoritmo Dominancia poligonal**

*Actor Principal***:** Usuario.

Personal Involucrado e Intereses:

Usuario: Quiere ejecutar el algoritmo que construye el trazado dominancia poligonal de un st-grafo plano definido.

*Precondiciones***:** Se ha definido un grafo plano.

*Garantías de Éxito***:** Se ejecuta el algoritmo correctamente y se deja preparada la ejecución paso a paso.

*Garantías de Fracaso***:** Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.

*Activa***:** El Usuario selecciona la opción Trazado de Dominancia Poligonal en el menú de Algoritmos.

*Escenario Principal de Éxito***:**

1. El Usuario quiere aplicar el algoritmo Dominancia poligonal a un grafo definido anteriormente.
2. El Sistema aplica el algoritmo especificado al grafo y deja preparada la ejecución paso a paso.

Extensiones:

2a. El grafo no es un st-grafo:

1. El Sistema muestra el error y termina.

- **Caso de Uso 8: Aplicar Orientación Bipolar**

*Actor Principal***:** Usuario.

Personal Involucrado e Intereses:

Usuario: Quiere aplicar la orientación bipolar a un grafo plano no dirigido.

*Precondiciones***:** Se ha definido un grafo plano.

*Garantías de Éxito***:** Se aplica la orientación bipolar correctamente sobre el grafo.
*Garantías de Fracaso***:** Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.
*Activa***:** El Usuario selecciona la opción Orientación Bipolar.
*Escenario Principal de Éxito***:**

1. El Usuario quiere aplicar el algoritmo Orientación Bipolar a un grafo definido anteriormente.
2. El Sistema aplica el algoritmo especificado al grafo y lo orienta de forma tal que es un st-grafo.

Extensiones:

2a. El grafo ya es dirigido:
1. El Sistema muestra el error y termina.

2b. El grafo no es biconexo:
1. El Sistema muestra el error y termina.

- **Caso de Uso 9: Gestionar Caminos Disjuntos**

*Actor Principal***:** Usuario.
Personal Involucrado e Intereses:
Usuario: Quiere añadir un camino al conjunto, mostrar todos los caminos añadidos o eliminar el conjunto de caminos asociados a un grafo.
*Precondiciones***:** Se ha definido un grafo plano dirigido.
*Garantías de Éxito***:** Se queda actualizado el estado del conjunto de caminos disjuntos.
*Garantías de Fracaso***:** Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.
*Activa***:** El Usuario selecciona algunas de las tres opciones que permiten gestionar los caminos disjuntos.
*Escenario Principal de Éxito***:**

1. El Usuario quiere añadir un camino, que ha seleccionado sobre un grafo dado, al conjunto de caminos disjuntos.
2. El Sistema agrega el camino al conjunto manteniéndose la propiedad de que todos los caminos son disjuntos dos a dos.

Extensiones:

1a. Mostrar los caminos del conjunto:
1. El Usuario desea conocer qué caminos han sido añadidos al conjunto.
2. El Sistema resalta sobre el grafo todos los caminos que han sido agregados hasta el momento.

1b. Eliminar todos los caminos del conjunto:
1. El Usuario quiere borrar todos los caminos del conjunto.
2. El Sistema elimina todos los caminos que han sido agregados hasta el momento dejando el conjunto vacío.

2a. El camino que se intenta agregar no es disjunto con algún otro camino perteneciente al conjunto:
1. El Sistema muestra el error y termina.

- **Caso de Uso 10: Mostrar Etapa Anterior del Algoritmo**
  *Actor Principal***:** Usuario.
  Personal Involucrado e Intereses:
  Usuario: Quiere mostrar el paso anterior en la ejecución actual del algoritmo.
  *Precondiciones***:** Se está ejecutando un algoritmo.
  *Garantías de Éxito***:** Se muestra el estado de la ejecución en la etapa anterior del algoritmo.
  *Garantías de Fracaso***:** Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.
  *Activa***:** El Usuario presiona el botón Anterior en el control de la ejecución.
  *Escenario Principal de Éxito***:**
  1. El Usuario quiere ir al paso anterior en la ejecución del algoritmo actual.
  2. El Sistema muestra la situación de la ejecución en la etapa anterior a la actual.

  Extensiones:
  2a. No hay paso Anterior en la ejecución del algoritmo:
  1. No se realiza ninguna acción.

- **Caso de Uso 11: Mostrar Etapa Siguiente del Algoritmo**
  *Actor Principal***:** Usuario.
  Personal Involucrado e Intereses:
  Usuario: Quiere mostrar el paso siguiente en la ejecución actual del algoritmo.
  *Precondiciones***:** Se está ejecutando un algoritmo.
  *Garantías de Éxito***:** Se muestra el estado de la ejecución en la etapa siguiente del algoritmo.
  *Garantías de Fracaso***:** Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.
  *Activa***:** El Usuario presiona el botón Siguiente en el control de la ejecución.
  *Escenario Principal de Éxito***:**

1. El Usuario quiere ir al paso siguiente en la ejecución del algoritmo actual.
2. El Sistema muestra la situación de la ejecución en la etapa siguiente a la actual.

Extensiones:

2a. No hay paso Siguiente en la ejecución del algoritmo:

1. No se realiza ninguna acción.

- **Caso de Uso 12: Finalizar Etapas del Algoritmo**

  *Actor Principal***:** Usuario.

  Personal Involucrado e Intereses:

  Usuario: Quiere mostrar todos los pasos hasta el final de la ejecución actual del algoritmo.

  *Precondiciones***:** Se está ejecutando un algoritmo.

  *Garantías de Éxito***:** Se muestra el estado final de la ejecución del algoritmo.

  *Garantías de Fracaso***:** Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.

  *Activa***:** El Usuario presiona el botón Finalizar en el control de la ejecución.

  *Escenario Principal de Éxito***:**

  1. El Usuario quiere finalizar la ejecución del algoritmo actual viendo el resultado final.
  2. El Sistema muestra el resultado final de la aplicación del algoritmo al grafo.

  Extensiones:

  2a. No hay paso Siguiente en la ejecución del algoritmo:

  1. No se realiza ninguna acción.

- **Caso de Uso 13: Cancelar Etapas del Algoritmo**

  *Actor Principal***:** Usuario.

  Personal Involucrado e Intereses:

  Usuario: Quiere cancelar la ejecución actual del algoritmo.

  *Precondiciones***:** Se está ejecutando un algoritmo.

  *Garantías de Éxito***:** Se termina la ejecución del algoritmo.

  *Activa***:** El Usuario presiona el botón Cancelar en el control de la ejecución.

  *Escenario Principal de Éxito***:**

  1. El Usuario quiere cancelar la ejecución del algoritmo actual.
  2. El Sistema finaliza la ejecución actual sin mostrar ningún otro paso.

- **Caso de Uso 14: Crear Nuevo Grafo**
  *Actor Principal***:** Usuario.
  Personal Involucrado e Intereses:
  Usuario: Quiere crear un grafo nuevo.
  *Precondiciones***:** Ninguna.
  *Garantías de Éxito***:** Se crea un nuevo panel de edición de grafos que contiene un grafo vacío.
  *Garantías de Fracaso***:** Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.
  *Activa***:** El Usuario selecciona la opción Nuevo Grafo en el menú de Archivo.
  *Escenario Principal de Éxito***:**
    1. El Usuario quiere crear un nuevo grafo seleccionando la opción Nuevo en Archivo.
    2. El Sistema muestra un panel donde se pueden especificar las características del grafo que se quiere crear, como aristas dirigidas, permitir ciclos, bucles o aristas múltiples. Por definición, si no se permiten ciclos, tampoco se permiten bucles o aristas múltiples.
    3. El Usuario selecciona el tipo de grafo que quiere construir.
    4. El Sistema crea un grafo vacío con las características deseadas y sin guardar aún.

  Extensiones:
    1a. El Usuario presiona la combinación de teclas Ctrl + N:
      1. Se salta al paso 2.

- **Caso de Uso 15: Duplicar Grafo**
  *Actor Principal***:** Usuario.
  Personal Involucrado e Intereses:
  Usuario: Quiere duplicar un grafo existente para poder editarlo en un panel independiente.
  *Precondiciones***:** Se está editando un grafo.
  *Garantías de Éxito***:** Se crea un nuevo panel de edición de grafos que contiene un grafo idéntico al duplicado.
  *Garantías de Fracaso***:** Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.
  *Activa***:** El Usuario selecciona la opción Duplicar Grafo en el menú de Archivo.
  *Escenario Principal de Éxito***:**

1. El Usuario quiere copiar un grafo existente a otro panel de edición seleccionando la opción Duplicar en Archivo.
2. El Sistema crea un grafo en un panel de edición nuevo copiando el estado actual del grafo original. El nuevo grafo está sin guardar aunque el original ya haya sido almacenado en un archivo.

Extensiones:

1a. El Usuario presiona la combinación de teclas Ctrl + D:

1. Se salta al paso 2.

- **Caso de Uso 16: Abrir Grafo**

*Actor Principal***:** Usuario.

Personal Involucrado e Intereses:

Usuario: Quiere abrir un grafo guardado en un archivo anteriormente.

*Precondiciones***:** Ninguno.

*Garantías de Éxito***:** Se abre un panel de edición que contiene el grafo definido en el archivo.

*Garantías de Fracaso***:** Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.

*Activa***:** El Usuario selecciona la opción Abrir Grafo en el menú de Archivo.

*Escenario Principal de Éxito***:**

1. El Usuario quiere abrir un grafo guardado en un archivo seleccionando la opción Abrir en Archivo.
2. El Sistema abre un navegador del sistema de ficheros que permite seleccionar solo archivos que sean del tipo definido por la aplicación, es decir, con extensión *.vsg*.
3. El Usuario navega por el sistema de ficheros y elige el archivo donde está contenido el grafo a abrir.
4. El Sistema abre el un panel de edición conteniendo el grafo almacenado en el archivo.

Extensiones:

1a. El Usuario presiona la combinación de teclas Ctrl + O:

1. Se salta al paso 2.

4a. El contenido del archivo impide la correcta recuperación del grafo:

1. El Sistema detecta el error y no abre el grafo.

- **Caso de Uso 17: Guardar Grafo**

*Actor Principal***:** Usuario.

Personal Involucrado e Intereses:

Usuario: Quiere guardar los cambios realizados a un grafo en su archivo correspondiente.
*Precondiciones***:** Se está editando un grafo.
*Garantías de Éxito***:** Se actualiza el archivo asociado al grafo con los cambios hechos desde la última operación de guardar.
*Garantías de Fracaso***:** Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.
*Activa***:** El Usuario selecciona la opción Guardar Grafo en el menú de Archivo.
*Escenario Principal de Éxito***:**

1. El Usuario quiere guardar las modificaciones realizadas a un grafo seleccionando la opción Guardar en Archivo.
2. El Sistema escribe el estado actual del grafo en su archivo asociado.

Extensiones:

1a. El Usuario presiona la combinación de teclas Ctrl + S:
   1. Se salta al paso 2.

2a. El grafo no ha sido asociado con ningún archivo:
   1. Incluye caso de uso Guardar Grafo Como.

- **Caso de Uso 18: Guardar Grafo Como**
  *Actor Principal***:** Usuario.
  Personal Involucrado e Intereses:
  Usuario: Quiere guardar los cambios realizados a un grafo en un nuevo archivo.
  *Precondiciones***:** Se está editando un grafo.
  *Garantías de Éxito***:** Se crea un nuevo archivo asociado al grafo actual y que contiene su estado.
  *Garantías de Fracaso***:** Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.
  *Activa***:** El Usuario selecciona la opción Guardar Grafo Como en el menú de Archivo.
  *Escenario Principal de Éxito***:**
  1. El Usuario quiere crear un nuevo archivo que contenga las modificaciones realizadas a un grafo seleccionando la opción Guardar Como en Archivo.
  2. El Sistema abre un navegador del sistema de ficheros para seleccionar el lugar donde será ubicado el archivo.
  3. El Usuario selecciona la ubicación, escribe un nombre para el archivo y elige el tipo de archivo *Visigraph Graph* utilizado por la aplicación para almacenar los grafos.

4. El Sistema crea un nuevo archivo con la extensión *.vsg* en la ruta elegida donde almacena el estado actual del grafo. Este archivo permite la posterior utilización del grafo.

Extensiones:

3a. El tipo de archivo elegido es *PNG* (*Portable Network Graphics*):

1. El Sistema exporta el grafo con el formato gráfico *PNG* creando un archivo con extensión *.png* en la ruta seleccionada.

3b. El tipo de archivo elegido es *SVG* (*Scalable Vector Graphics*):

1. El Sistema exporta el grafo con el formato gráfico *SVG* creando un archivo con extensión *.svg* en la ruta seleccionada.

- **Caso de Uso 19: Imprimir Grafo**

  *Actor Principal***:** Usuario.

  Personal Involucrado e Intereses:

  Usuario: Quiere imprimir un grafo existente.

  *Precondiciones***:** Se está editando un grafo.

  *Garantías de Éxito***:** Se invoca la operación del sistema operativo a través de la cual se imprime el grafo.

  *Garantías de Fracaso***:** Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.

  *Activa***:** El Usuario selecciona la opción Imprimir Grafo en el menú de Archivo.

  *Escenario Principal de Éxito***:**

  1. El Usuario quiere imprimir un grafo existente seleccionando la opción Imprimir en Archivo.
  2. El Sistema envía a imprimir el grafo a la impresora.

  Extensiones:

  1a. El Usuario presiona la combinación de teclas Ctrl + P:

  1. Se salta al paso 2.

- **Caso de Uso 20: Añadir Vértice**

  *Actor Principal***:** Usuario.

  Personal Involucrado e Intereses:

  Usuario: Quiere agregar un vértice al grafo que está siendo editado actualmente.

  *Precondiciones***:** Se está editando un grafo.

  *Garantías de Éxito***:** Se añade un vértice al grafo en la posición seleccionada. Se guarda la operación de edición en un historial para que pueda ser deshecha.

Garantías de Fracaso: Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.
Activa: El Usuario presiona el botón Agregar vértices y aristas dentro del panel de edición de grafos.
Escenario Principal de Éxito:

1. El Usuario selecciona añadir vértices y aristas.
2. El Sistema establece el modo de agregar vértices y aristas.
3. El Usuario añade un vértice al grafo actual haciendo clic izquierdo en un punto del panel del grafo.
4. El Sistema agrega un nuevo vértice al grafo ubicado en la posición donde se hizo clic.

Extensiones:

1a. El modo de agregar vértices y aristas ya ha sido establecido:

1. Se salta al paso 3.

- **Caso de Uso 21: Añadir Arista**

Actor Principal: Usuario.
Personal Involucrado e Intereses:
Usuario: Quiere agregar una arista al grafo que está siendo editado actualmente.
Precondiciones: Se está editando un grafo. El grafo tiene al menos un vértice.
Garantías de Éxito: Se añade una arista al grafo entre un par de vértices existentes o entre un vértice existente y otro nuevo que se crea. Se guarda la operación de edición en un historial para que pueda ser deshecha.
Garantías de Fracaso: Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.
Activa: El Usuario presiona el botón Agregar vértices y aristas dentro del panel de edición de grafos.
Escenario Principal de Éxito:

1. El Usuario selecciona añadir vértices y aristas.
2. El Sistema establece el modo de agregar vértices y aristas.
3. El Usuario hace clic izquierdo sobre un vértice y, a continuación, sobre otro, ambos ya existentes.
4. El Sistema agrega una arista entre los dos vértices sobre los que el Usuario hizo clic.

Extensiones:

1a. El modo de agregar vértices y aristas ya ha sido establecido:

1. Se salta al paso 3.

3a. El Usuario hace clic izquierdo sobre un vértice ya existente y, a continuación, hace clic izquierdo sobre una posición vacía del grafo:
1. El Sistema añade un nuevo vértice al grafo y agrega una arista desde el primer vértice hasta el recién creado.

3b. El Usuario hace clic izquierdo sobre un vértice, lo mantiene presionado y lo suelta sobre otro vértice ya creado:
1. El Sistema añade una arista entre los dos vértices involucrados en la operación.

- **Caso de Uso 22: Eliminar Elementos**

*Actor Principal***:** Usuario.

Personal Involucrado e Intereses:

Usuario: Quiere borrar uno o varios elementos, es decir, vértices o aristas de un grafo.

*Precondiciones***:** Se está editando un grafo.

*Garantías de Éxito***:** Se eliminan todos los elementos seleccionados, así como cualquier otro elemento que dependa de ellos. Se guarda la operación de edición en un historial para que pueda ser deshecha.

*Garantías de Fracaso***:** Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.

*Activa***:** El Usuario presiona el botón Eliminar elementos dentro del panel de edición de grafos.

*Escenario Principal de Éxito***:**

1. El Usuario selecciona eliminar elementos.
2. El Sistema establece el modo de eliminar vértices y aristas.
3. El Usuario hace clic izquierdo sobre el elemento que quiere borrar del grafo actual.
4. El Sistema elimina la arista o el vértice del grafo. En el caso de ser un vértice, borra todas las aristas relacionadas con él.

Extensiones:

1a. El modo de eliminar elementos ya ha sido establecido:
1. Se salta al paso 3.

3a. El Usuario hace clic izquierdo o derecho sobre un espacio no ocupado del grafo y mantiene presionado el botón formando un rectángulo:
1. El Sistema elimina todos los elementos seleccionados con el rectángulo, al igual que todas las aristas dependientes de los vértices involucrados en la operación.

- **Caso de Uso 23: Seleccionar Elementos**
  *Actor Principal***:** Usuario.
  Personal Involucrado e Intereses:
  Usuario: Quiere seleccionar uno o varios elementos de un grafo.
  *Precondiciones***:** Se está editando un grafo.
  *Garantías de Éxito***:** Se seleccionan los elementos deseados. Se guarda la operación de edición en un historial para que pueda ser deshecha.
  *Garantías de Fracaso***:** Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.
  *Activa***:** El Usuario presiona el botón Seleccionar y mover elementos dentro del panel de edición de grafos.
  *Escenario Principal de Éxito***:**
    1. El Usuario presiona seleccionar elementos.
    2. El Sistema establece el modo de seleccionar y mover elementos.
    3. El Usuario hace clic izquierdo sobre el elemento que quiere seleccionar del grafo actual.
    4. El Sistema selecciona la arista o el vértice del grafo.

  Extensiones:
    1a. El modo de seleccionar y mover elementos ya ha sido establecido:
      1. Se salta al paso 3.

    3a. El Usuario hace clic izquierdo o derecho sobre un espacio no ocupado del grafo y mantiene presionado el botón formando un rectángulo:
      1. El Sistema selecciona todos los elementos contenidos dentro del rectángulo.

- **Caso de Uso 24: Seleccionar Todos los Elementos**
  *Actor Principal***:** Usuario.
  Personal Involucrado e Intereses:
  Usuario: Quiere seleccionar todos los elementos de un grafo.
  *Precondiciones***:** Se está editando un grafo.
  *Garantías de Éxito***:** Se seleccionan todos los elementos. Se guarda la operación de edición en un historial para que pueda ser deshecha.
  *Garantías de Fracaso***:** Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.
  *Activa***:** El Usuario presiona Seleccionar Todos en el menú Edición.
  *Escenario Principal de Éxito***:**
    1. El Usuario presiona Seleccionar Todos en el menú Edición.
    2. El Sistema selecciona todos los elementos del grafo.

Extensiones:

1a. El Usuario presiona la combinación de teclas Ctrl + A:

1. El Sistema selecciona todos los elementos del grafo.

- **Caso de Uso 25: Seleccionar Todos los Vértices**
  *Actor Principal***:** Usuario.
  Personal Involucrado e Intereses:
  Usuario: Quiere seleccionar todos los vértices de un grafo.
  *Precondiciones***:** Se está editando un grafo.
  *Garantías de Éxito***:** Se seleccionan todos los vértices. Se guarda la operación de edición en un historial para que pueda ser deshecha.
  *Garantías de Fracaso***:** Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.
  *Activa***:** El Usuario presiona Seleccionar Todos los Vértices en el menú Edición.
  *Escenario Principal de Éxito***:**
  1. El Usuario presiona Seleccionar Todos los Vértices en el menú Edición.
  2. El Sistema selecciona todos los vértices del grafo.

- **Caso de Uso 26: Seleccionar Todas las Aristas**
  *Actor Principal***:** Usuario.
  Personal Involucrado e Intereses:
  Usuario: Quiere seleccionar todas las aristas de un grafo.
  *Precondiciones***:** Se está editando un grafo.
  *Garantías de Éxito***:** Se seleccionan todas las aristas. Se guarda la operación de edición en un historial para que pueda ser deshecha.
  *Garantías de Fracaso***:** Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.
  *Activa***:** El Usuario presiona Seleccionar Todas las Aristas en el menú Edición.
  *Escenario Principal de Éxito***:**
  1. El Usuario presiona Seleccionar Todas las Aristas en el menú Edición.
  2. El Sistema selecciona todas las aristas del grafo.

- **Caso de Uso 27: Deseleccionar Elementos**
  *Actor Principal***:** Usuario.
  Personal Involucrado e Intereses:
  Usuario: Quiere deshacer una selección de elementos de un grafo.
  *Precondiciones***:** Se está editando un grafo.

Garantías de Éxito: Se deseleccionan todos los elementos. Se guarda la operación de edición en un historial para que pueda ser deshecha.
Garantías de Fracaso: Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.
Activa: El Usuario presiona el botón Seleccionar y mover elementos dentro del panel de edición de grafos.
Escenario Principal de Éxito:

1. El Usuario presiona seleccionar elementos.
2. El Sistema establece el modo de seleccionar y mover elementos.
3. El Usuario hace clic izquierdo sobre una posición libre en el panel del grafo.
4. El Sistema deshace la selección de todos los elemento del grafo.

Extensiones:

1a. El modo de seleccionar y mover elementos ya ha sido establecido:
1. Se salta al paso 3.

- **Caso de Uso 28: Mover Elementos**

Actor Principal: Usuario.
Personal Involucrado e Intereses:
Usuario: Quiere mover uno o varios elementos de un grafo.
Precondiciones: Se está editando un grafo.
Garantías de Éxito: Se trasladan los elementos deseados. Se guarda la operación de edición en un historial para que pueda ser deshecha.
Garantías de Fracaso: Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.
Activa: El Usuario presiona el botón Seleccionar y mover elementos dentro del panel de edición de grafos.
Escenario Principal de Éxito:

1. El Usuario presiona seleccionar elementos.
2. El Sistema establece el modo de seleccionar y mover elementos.
3. El Usuario hace clic izquierdo sobre un vértice, lo mantiene presionado y lo suelta sobre la posición donde quiere mover el elemento.
4. El Sistema mueve el elemento a su nueva posición.

Extensiones:

1a. El modo de seleccionar y mover elementos ya ha sido establecido:
1. Se salta al paso 3.

3a. El Usuario hace clic izquierdo sobre el centro de una arista, lo mantiene presionado y lo suelta sobre la posición donde quiere moverlo:

1. El Sistema mueve el centro de la arista a su nueva posición, convirtiendo la arista en un arco siempre que sea imposible dibujarla en línea recta.

3b. El Usuario hace clic izquierdo sobre una selección de elementos del grafo, la mantiene presionada y la suelta sobre la posición donde quiere moverla:

1. El Sistema mueve todos los elementos de la selección a su nueva posición manteniendo la relación original entre estos objetos.

- **Caso de Uso 29: Copiar Elementos**

  *Actor Principal***:** Usuario.

  Personal Involucrado e Intereses:

  Usuario: Quiere copiar los elementos seleccionados de un grafo.

  *Precondiciones***:** Se está editando un grafo. Se han seleccionado uno o varios elementos del grafo.

  *Garantías de Éxito***:** Se hace una copia de los elementos deseados. El estado del grafo no varía.

  *Garantías de Fracaso***:** Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.

  *Activa***:** El Usuario presiona Copiar en el menú Edición.

  *Escenario Principal de Éxito***:**

  1. El Usuario presiona Copiar en el menú Edición.
  2. El Sistema hace una copia de todos los elementos del grafo seleccionados en ese instante.

  Extensiones:

  1a. El Usuario presiona la combinación de teclas Ctrl + C:

  1. El Sistema hace una copia de todos los elementos del grafo seleccionados en ese instante.

- **Caso de Uso 30: Pegar Elementos**

  *Actor Principal***:** Usuario.

  Personal Involucrado e Intereses:

  Usuario: Quiere pegar los elementos anteriormente copiados o cortados de un grafo.

  *Precondiciones***:** Se está editando un grafo. Se han cortado o copiado uno o varios elementos de un grafo.

*Garantías de Éxito***:** Se pegan los elementos disponibles, los cuales podrán ser usados en una operación de pegado posterior. Se guarda la operación de edición en un historial para que pueda ser deshecha.
*Garantías de Fracaso***:** Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.
*Activa***:** El Usuario presiona Pegar en el menú Edición.
*Escenario Principal de Éxito***:**

1. El Usuario presiona Pegar en el menú Edición.
2. El Sistema pega todos los elementos copiados o cortados anteriormente en el grafo actual.

Extensiones:

1a. El Usuario presiona la combinación de teclas Ctrl + V:

1. El Sistema pega todos los elementos copiados o cortados anteriormente en el grafo actual.

- **Caso de Uso 31: Cortar Elementos**
  *Actor Principal***:** Usuario.
  Personal Involucrado e Intereses:
  Usuario: Quiere cortar los elementos seleccionados de un grafo.
  *Precondiciones***:** Se está editando un grafo. Se han seleccionado uno o varios elementos del grafo.
  *Garantías de Éxito***:** Se cortan del grafo los elementos seleccionados. Las aristas que pierdan solo uno de sus dos vértices serán eliminadas. Se guarda la operación de edición en un historial para que pueda ser deshecha.
  *Garantías de Fracaso***:** Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.
  *Activa***:** El Usuario presiona Cortar en el menú Edición.
  *Escenario Principal de Éxito***:**
  1. El Usuario presiona Cortar en el menú Edición.
  2. El Sistema corta los elementos del grafo seleccionados en ese instante.

  Extensiones:

  1a. El Usuario presiona la combinación de teclas Ctrl + X:

  1. El Sistema corta los elementos del grafo seleccionados en ese instante.

- **Caso de Uso 32: Editar Parámetros de los Vértices**
  *Actor Principal***:** Usuario.
  Personal Involucrado e Intereses:

Usuario: Quiere editar los parámetros de los vértices de un grafo. Los parámetros existentes son etiqueta, radio y peso.
*Precondiciones***:** Se está editando un grafo. Se han seleccionado uno o varios vértices del grafo.
*Garantías de Éxito***:** Se asigna un nuevo valor al parámetro elegido. Se guarda la operación de edición en un historial para que pueda ser deshecha.
*Garantías de Fracaso***:** Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.
*Activa***:** El Usuario presiona el botón Seleccionar y mover elementos dentro del panel de edición de grafos.
*Escenario Principal de Éxito***:**

1. El Usuario presiona seleccionar elementos.
2. El Sistema establece el modo de seleccionar y mover elementos.
3. El Usuario hace clic derecho sobre una selección de elementos del grafo ya realizada.
4. El Sistema muestra que parámetros se pueden editar de un vértice.
5. El Usuario selecciona editar el parámetro etiqueta y establece un nuevo valor.
6. El Sistema actualiza el valor del parámetro.

Extensiones:

1a. El modo de seleccionar y mover elementos ya ha sido establecido:
   1. Se salta al paso 3.

5a. El Usuario selecciona editar el parámetro radio y establece un nuevo valor:
   1. El Sistema actualiza el valor del parámetro.

5b. El Usuario selecciona editar el parámetro peso y establece un nuevo valor:
   1. El Sistema actualiza el valor del parámetro.

- **Caso de Uso 33: Editar Parámetros de las Aristas**
  *Actor Principal***:** Usuario.
  Personal Involucrado e Intereses:
  Usuario: Quiere editar los parámetros de las aristas de un grafo. Los parámetros existentes son etiqueta, anchura y peso.
  *Precondiciones***:** Se está editando un grafo. Se han seleccionado una o varias aristas del grafo.
  *Garantías de Éxito***:** Se asigna un nuevo valor al parámetro elegido. Se guarda la operación de edición en un historial para que pueda ser deshecha.

Garantías de Fracaso: Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.
Activa: El Usuario presiona el botón Seleccionar y mover elementos dentro del panel de edición de grafos.
Escenario Principal de Éxito:

1. El Usuario presiona seleccionar elementos.
2. El Sistema establece el modo de seleccionar y mover elementos.
3. El Usuario hace clic derecho sobre una selección de elementos del grafo ya realizada.
4. El Sistema muestra que parámetros se pueden editar de una arista.
5. El Usuario selecciona editar el parámetro etiqueta y establece un nuevo valor.
6. El Sistema actualiza el valor del parámetro.

Extensiones:

1a. El modo de seleccionar y mover elementos ya ha sido establecido:
   1. Se salta al paso 3.

5a. El Usuario selecciona editar el parámetro anchura y establece un nuevo valor:
   1. El Sistema actualiza el valor del parámetro.

5b. El Usuario selecciona editar el parámetro peso y establece un nuevo valor:
   1. El Sistema actualiza el valor del parámetro.

- **Caso de Uso 34: Deshacer Acción de Edición**
  Actor Principal: Usuario.
  Personal Involucrado e Intereses:
  Usuario: Quiere deshacer la última operación que ha modificado el estado de un grafo.
  Precondiciones: Se está editando un grafo.
  Garantías de Éxito: Se deshace la última operación de edición realizada sobre el grafo. Se guarda la operación deshecha en un historial para que pueda ser rehecha.
  Garantías de Fracaso: Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.
  Activa: El Usuario presiona Deshacer en el menú Edición.
  Escenario Principal de Éxito:
  1. El Usuario presiona Deshacer en el menú Edición.

2. El Sistema restablece el estado anterior a la última edición guardada en el historial de deshacer.

Extensiones:

1a. El Usuario presiona la combinación de teclas Ctrl + Z:

1. El Sistema restablece el estado anterior a la última edición guardada en el historial de deshacer.

2a. El historial de operaciones de edición a deshacer está vacío:

1. El Sistema no realiza ninguna modificación.

- **Caso de Uso 35: Rehacer Acción de Edición**

*Actor Principal***:** Usuario.

Personal Involucrado e Intereses:

Usuario: Quiere rehacer la última operación que ha sido deshecha.

*Precondiciones***:** Se está editando un grafo.

*Garantías de Éxito***:** Se rehace la última operación de edición deshecha. Se guarda la operación rehecha en un historial para que pueda ser deshecha.

*Garantías de Fracaso***:** Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.

*Activa***:** El Usuario presiona Rehacer en el menú Edición.

*Escenario Principal de Éxito***:**

1. El Usuario presiona Rehacer en el menú Edición.
2. El Sistema vuelve a aplicar al grafo la última operación guardada en el historial de rehacer.

Extensiones:

1a. El Usuario presiona la combinación de teclas Ctrl + Y:

1. El Sistema vuelve a aplicar al grafo la última operación guardada en el historial de rehacer.

2a. El historial de operaciones de edición a rehacer está vacío:

1. El Sistema no realiza ninguna modificación.

- **Caso de Uso 36: Mostrar Etiquetas de los Vértices**

*Actor Principal***:** Usuario.

Personal Involucrado e Intereses:

Usuario: Quiere que se muestren las etiquetas de todos los vértices del grafo.

*Precondiciones***:** Se está editando un grafo.

*Garantías de Éxito***:** Se muestran las etiquetas al lado de los vértices.

*Garantías de Fracaso***:** Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.

*Activa***:** El Usuario presiona el botón Mostrar etiquetas de los vértices dentro del panel de edición de grafos.
*Escenario Principal de Éxito***:**

1. El Usuario presiona Mostrar etiquetas de los vértices.
2. El Sistema establece el modo de mostrar etiquetas de los vértices y enseña las etiquetas sobre el grafo.

- **Caso de Uso 37: Mostrar Pesos de los Vértices**
  *Actor Principal***:** Usuario.
  Personal Involucrado e Intereses:
  Usuario: Quiere que se muestren los pesos de todos los vértices del grafo.
  *Precondiciones***:** Se está editando un grafo.
  *Garantías de Éxito***:** Se muestran los pesos al lado de los vértices.
  *Garantías de Fracaso***:** Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.
  *Activa***:** El Usuario presiona el botón Mostrar pesos de los vértices dentro del panel de edición de grafos.
  *Escenario Principal de Éxito***:**
  1. El Usuario presiona Mostrar pesos de los vértices.
  2. El Sistema establece el modo de mostrar pesos de los vértices y enseña los pesos sobre el grafo.

- **Caso de Uso 38: Mostrar Etiquetas de las Aristas**
  *Actor Principal***:** Usuario.
  Personal Involucrado e Intereses:
  Usuario: Quiere que se muestren las etiquetas de todas las aristas del grafo.
  *Precondiciones***:** Se está editando un grafo.
  *Garantías de Éxito***:** Se muestran las etiquetas en el centro de las aristas.
  *Garantías de Fracaso***:** Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.
  *Activa***:** El Usuario presiona el botón Mostrar etiquetas de las aristas dentro del panel de edición de grafos.
  *Escenario Principal de Éxito***:**
  1. El Usuario presiona Mostrar etiquetas de las aristas.
  2. El Sistema establece el modo de mostrar etiquetas de las aristas y enseña las etiquetas sobre el grafo.

- **Caso de Uso 39: Mostrar Pesos de las Aristas**

*Actor Principal***:** Usuario.
Personal Involucrado e Intereses:
Usuario: Quiere que se muestren los pesos de todas las aristas del grafo.
*Precondiciones***:** Se está editando un grafo.
*Garantías de Éxito***:** Se muestran los pesos en el centro de las aristas.
*Garantías de Fracaso***:** Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.
*Activa***:** El Usuario presiona el botón Mostrar pesos de las aristas dentro del panel de edición de grafos.
*Escenario Principal de Éxito***:**

1. El Usuario presiona Mostrar pesos de las aristas.
2. El Sistema establece el modo de mostrar pesos de las aristas y enseña los pesos sobre el grafo.

- **Caso de Uso 40: Organizar Ventanas**

*Actor Principal***:** Usuario.
Personal Involucrado e Intereses:
Usuario: Quiere ordenar las ventanas de edición de grafos de distintas formas como en cascada, en mosaico horizontal, en mosaico vertical y en cuadrícula.
*Precondiciones***:** Se está editando, al menos, un grafo.
*Garantías de Éxito***:** Se organizan las ventanas de la manera seleccionada.
*Garantías de Fracaso***:** Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.
*Activa***:** El Usuario presiona Cascada, Mosaico Vertical, Mosaico Horizontal o Cuadrícula en el menú Ventanas.
*Escenario Principal de Éxito***:**

1. El Usuario presiona Cascada en el menú Ventanas.
2. El Sistema organiza todas las ventanas mostrando sus barras de título una debajo de la otra y redimensionadas.

Extensiones:

1a. El Usuario presiona Mosaico Vertical en el menú Ventanas:
   1. El Sistema organiza todas las ventanas mostrándolas como paneles distribuidos verticalmente.

1b. El Usuario presiona Mosaico Horizontal en el menú Ventanas:
   1. El Sistema organiza todas las ventanas mostrándolas como paneles distribuidos horizontalmente.

1c. El Usuario presiona Cuadrícula en el menú Ventanas:

1. El Sistema organiza todas las ventanas mostrándolas como paneles distribuidos en cuadrículas.

- **Caso de Uso 41: Mostrar Ventana Anterior**
  *Actor Principal***:** Usuario.
  Personal Involucrado e Intereses:
  Usuario: Quiere pasar a la ventana de edición de grafos anterior según el orden en el que están situadas.
  *Precondiciones***:** Se está editando, al menos, un grafo.
  *Garantías de Éxito***:** Se muestra la anterior ventana.
  *Garantías de Fracaso***:** Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.
  *Activa***:** El Usuario presiona Anterior Ventana en el menú Ventanas.
  *Escenario Principal de Éxito***:**
  1. El Usuario presiona Anterior en el menú Ventanas.
  2. El Sistema pasa a la ventana anterior a la actual.

  Extensiones:
  1a. El Usuario presiona la combinación de teclas Ctrl + RePág:
  1. El Sistema pasa a la ventana anterior a la actual.

- **Caso de Uso 42: Mostrar Ventana Siguiente**
  *Actor Principal***:** Usuario.
  Personal Involucrado e Intereses:
  Usuario: Quiere pasar a la ventana de edición de grafos siguiente según el orden en el que están situadas.
  *Precondiciones***:** Se está editando, al menos, un grafo.
  *Garantías de Éxito***:** Se muestra la siguiente ventana.
  *Garantías de Fracaso***:** Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.
  *Activa***:** El Usuario presiona Siguiente Ventana en el menú Ventanas.
  *Escenario Principal de Éxito***:**
  1. El Usuario presiona Siguiente en el menú Ventanas.
  2. El Sistema pasa a la ventana siguiente a la actual.

  Extensiones:
  1a. El Usuario presiona la combinación de teclas Ctrl + AvPág:
  1. El Sistema pasa a la ventana siguiente a la actual.

- **Caso de Uso 43: Ampliar Zoom**

Actor Principal: Usuario.
Personal Involucrado e Intereses:
Usuario: Quiere aumentar el zoom sobre el grafo actual.
Precondiciones: Se está editando un grafo.
Garantías de Éxito: Se aumenta el zoom en el panel de edición del grafo.
Garantías de Fracaso: Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.
Activa: El Usuario presiona el botón Ampliar zoom dentro del panel de edición de grafos.
Escenario Principal de Éxito:

1. El Usuario presiona Ampliar zoom.
2. El Sistema aumenta el zoom sobre el grafo que está siendo editado, centrándose en el punto medio del panel.

Extensiones:

1a. El Usuario gira hacia delante la rueda central del ratón:

1. El Sistema aumenta el zoom sobre el grafo que está siendo editado, centrándose en el punto donde está situado el cursor del ratón sobre el panel.

- **Caso de Uso 44: Reducir Zoom**
  Actor Principal: Usuario.
  Personal Involucrado e Intereses:
  Usuario: Quiere reducir el zoom sobre el grafo actual.
  Precondiciones: Se está editando un grafo.
  Garantías de Éxito: Se reducir el zoom en el panel de edición del grafo.
  Garantías de Fracaso: Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.
  Activa: El Usuario presiona el botón Reducir zoom dentro del panel de edición de grafos.
  Escenario Principal de Éxito:
  1. El Usuario presiona Reducir zoom.
  2. El Sistema reduce el zoom sobre el grafo que está siendo editado, centrándose en el punto medio del panel.

  Extensiones:

  1a. El Usuario gira hacia atrás la rueda central del ratón:

  1. El Sistema reduce el zoom sobre el grafo que está siendo editado, centrándose en el punto donde está situado el cursor del ratón sobre el panel.

- **Caso de Uso 45: Ajustar Zoom al Grafo**
  *Actor Principal***:** Usuario.
  Personal Involucrado e Intereses:
  Usuario: Quiere ajustar el zoom sobre el grafo actual.
  *Precondiciones***:** Se está editando un grafo.
  *Garantías de Éxito***:** Se ajusta el zoom sobre el grafo en el panel de edición, de tal forma que se puede ver todo el grafo centrado.
  *Garantías de Fracaso***:** Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.
  *Activa***:** El Usuario presiona el botón Ajustar zoom al grafo dentro del panel de edición de grafos.
  *Escenario Principal de Éxito***:**
  1. El Usuario presiona Ajustar zoom al grafo.
  2. El Sistema selecciona un zoom sobre el grafo que está siendo editado que permite ver todas las partes que lo componen. Además, sitúa el grafo en el centro del panel.

- **Caso de Uso 46: Ajustar Zoom a Escala Natural**
  *Actor Principal***:** Usuario.
  Personal Involucrado e Intereses:
  Usuario: Quiere ajustar el zoom sobre el grafo actual a la escala natural.
  *Precondiciones***:** Se está editando un grafo.
  *Garantías de Éxito***:** Se ajusta el zoom sobre el grafo en el panel de edición, de tal forma que se ve el grafo a escala 1:1.
  *Garantías de Fracaso***:** Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.
  *Activa***:** El Usuario presiona el botón Ajustar zoom a escala 1:1 dentro del panel de edición de grafos.
  *Escenario Principal de Éxito***:**
  1. El Usuario presiona Ajustar zoom a escala 1:1.
  2. El Sistema establece el zoom según la escala natural.

- **Caso de Uso 47: Acceder a la Web del Proyecto**
  *Actor Principal***:** Usuario.
  Personal Involucrado e Intereses:
  Usuario: Quiere abrir la web del proyecto para poder acceder a contenidos teóricos o al manual de usuario.

Precondiciones: Ninguna.
Garantías de Éxito: Se abre un navegador de internet con la dirección de la página web del proyecto.
Garantías de Fracaso: Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.
Activa: El Usuario presiona el botón Contenidos en el menú Ayuda.
Escenario Principal de Éxito:

1. El Usuario presiona Contenidos en Ayuda.
2. El Sistema invoca la apertura del navegador de internet por defecto y lo dirige a la web del proyecto.

- **Caso de Uso 48: Mostrar Información de la Aplicación**
  Actor Principal: Usuario.
  Personal Involucrado e Intereses:
  Usuario: Quiere consultar datos acerca de la aplicación como nombre, autor, licencia, etc.
  Precondiciones: Ninguna.
  Garantías de Éxito: Se muestra toda la información relevante acerca de la aplicación.
  Garantías de Fracaso: Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.
  Activa: El Usuario presiona el botón Acerca de en el menú Ayuda.
  Escenario Principal de Éxito:
  1. El Usuario presiona Acerca de en Ayuda.
  2. El Sistema muestra en cuadro de diálogo el título, el autor, la versión, el objetivo, la licencia y la dirección web del proyecto.

- **Caso de Uso 49: Salir de la Aplicación**
  Actor Principal: Usuario.
  Personal Involucrado e Intereses:
  Usuario: Quiere finalizar la utilización de la aplicación.
  Precondiciones: Ninguna.
  Garantías de Éxito: Se cierra la aplicación.
  Garantías de Fracaso: Se deja el Sistema en estado coherente, es decir, en el estado anterior a la ejecución de este caso de uso.
  Activa: El Usuario presiona el botón Salir en el menú Archivo.
  Escenario Principal de Éxito:
  1. El Usuario presiona Salir.

2. El Sistema termina la ejecución.

4.1.2 Arquitectura del Sistema

La arquitectura del sistema la mostraremos mediante un diagrama que contiene solo los paquetes y las relaciones entre ellos, dadas las clases que agrupan. En este apartado podremos obtener una visión más general del diseño de la aplicación, lo cual facilitará las posibles modificaciones o extensiones del sistema por una tercera persona.

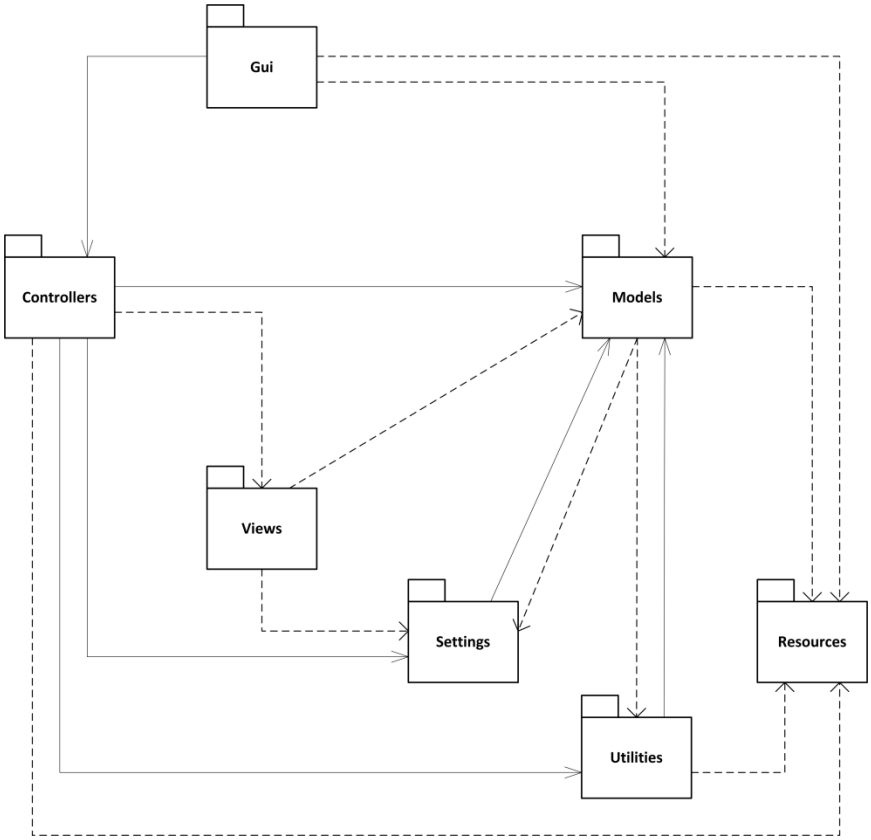

Ilustración 29: Diagrama de Clases. Paquetes

El diagrama anterior nos da una idea general de la arquitectura del sistema, así como de las dependencias más restrictivas existentes entre las clases de cada paquete. El estilo arquitectónico está basado en el patrón Modelo-Vista-Controlador, cuyo principal objetivo es separar los datos de la aplicación (Modelo), la interfaz de usuario (Vista) y la lógica de control (Controlador). Entre las ventajas de este patrón podemos resaltar que permite mayor cohesión entre los objetos del dominio al no tener que encargarse de la presentación del modelo, reduce el efecto de los futuros cambios en la interfaz y permite la reutilización de código.

El Modelo representa datos y las reglas que dirigen el acceso y la actualización de estos datos. A menudo, un modelo es una aproximación software de un proceso del mundo real, como en nuestro caso son los algoritmos. La Vista muestra el contenido de un modelo. Especifica como los datos del modelo deben ser presentados. Cuando un modelo cambia, la vista actualiza su presentación. Esto se consigue utilizando el patrón de diseño Observador, donde la vista se subscribe al modelo y este notifica a todos sus subscriptores cada vez que se modifica. Esto posibilita que el modelo no dependa de la vista dado que interaccionan a través de una interfaz. El Controlador captura y transforma las interacciones del usuario con la vista en acciones que el modelo llevará a cabo. En nuestro sistema, un ejemplo de interacción del usuario es pulsar un botón, seleccionar una opción en un menú o hacer clic en un panel. Siguiendo este esquema, el modelo tampoco depende del controlador. Lógicamente, tanto la vista como el controlador dependen del modelo. En nuestro sistema, la implementación del patrón MVC sufre una pequeña modificación que consiste en que las notificaciones de actualización del modelo pasan a través del controlador hacia la vista y no directamente entre modelo y vista, siendo el controlador quien se subscribe al modelo.

El paquete *Models* contiene las clases que definen los principales objetos y algoritmos que identificamos en el dominio de la aplicación. Por ejemplo, podemos encontrar las clases con las cuales se representan grafos, aristas, vértices, grafos duales, así como las clases de los distintos trazados y algoritmos implementados.

En el paquete *Controllers* encontramos todos los paneles contenidos en las ventanas de la aplicación unido a sus componentes gráficos internos. Dentro de estos paneles está ubicada su lógica de control que decide, por ejemplo, que acción realizar cuando el usuario hace clic en el panel de edición de grafos o cuando pulsa el botón de siguiente etapa de un algoritmo. Un ejemplo es el panel de edición de grafos con todos los componentes para añadir, borrar o seleccionar elementos, mostrar parámetros, zoom, editar caminos, etc. También contamos con los paneles de ejecución paso a paso de cada algoritmo, donde se incluyen los componentes de control de la ejecución junto a los paneles que muestran las transformaciones que se van realizando sobre el grafo.

El paquete *Gui* agrupa las clases representan las ventanas y diálogos de la interfaz de usuario. También, estas clases incluyen lógica de control para gestionar todos los eventos asociados a sus componentes. Algunos ejemplos son la ventana principal, donde se agrupan todos los menús y las ventanas internas de grafos y algoritmos, la ventana interna de grafos, que incluye el panel de edición de grafos, y las ventanas internas de ejecución de algoritmos, que contienen los paneles de ejecución de algoritmos.

El paquete *View* contiene las clases que implementan las distintas vistas de los modelos. En nuestro caso tenemos dos vistas: mostrar grafos y trazados en pantalla y convertir un grafo al formato Svg.

Dentro del paquete *Utilities* encontramos todas las clases que encapsulan métodos comunes reutilizados a lo largo del proyecto. Aquí podemos encontrar clases que implementan varios algoritmos de grafos, métodos útiles de geometría, manejo del formato JSON, manejo del formato Svg, métodos para aplicar transformadas, etc.

El paquete *Settings* contiene clases para definir atributos con valores reutilizados en numerosas ocasiones por varias clases. En el paquete *Resources* incluimos las clases que son útiles para manejar recursos como imágenes, iconos, cadenas de caracteres, etc.

4.1.3 Diagrama de Clases

Un diagrama de clases de diseño representa gráficamente las especificaciones de las entidades software de una aplicación. En un diagrama podemos ver, principalmente, las clases junto con sus métodos y sus atributos, así como las relaciones entre las clases. Las relaciones pueden ser de asociación, de dependencia o de herencia.

Para facilitar la comprensión dado el gran número de clases y relaciones entre ellas, enfocaremos el diagrama desde el punto de vista de las clases que están involucradas en las soluciones de los principales requisitos funcionales del proyecto. También por esta razón, mostraremos las clases sin sus atributos ni sus métodos.

El diagrama de la Ilustración 30 muestra la parte del modelo de la aplicación con la que se representa la estructura de un grafo, de un vértice, de una arista, de un grafo dual, así como las demás clases que colaboran en con estas. La estructura de un grafo se encuentra representada por las clases *Graph*, *Vertex* y *Edge*. La clase *Graph* tiene una lista de elementos de la clase *Vertex* para conocer todos los vértices, así como otra lista con elementos de la clase *Edge* para conocer todas las aristas. Además, *Graph* tiene una estructura de datos con las aristas incidentes en cada vértice. Cada arista conoce los vértices origen (*from*) y destino (*to*).

El grafo Dual está representado por la clase *DualGraph*. Un grafo Dual conoce el grafo del cual es dual. Además, como en nuestro caso nuestro grafo *G* es un st-grafo al igual que su dual, tenemos identificados los vértices S y T del grafo *G*, y las caras S y T del grafo dual *G**. También, dentro de *DualGraph* disponemos de la información relativa a cuales son las caras izquierda y derecha de un vértice de *G*. De la misma forma, tenemos la información relativa a cuales son las caras izquierda y derecha de una arista de *G*. Otra información importante que tenemos en la clase es el conjunto de rotaciones alrededor de cada vértice de *G*.

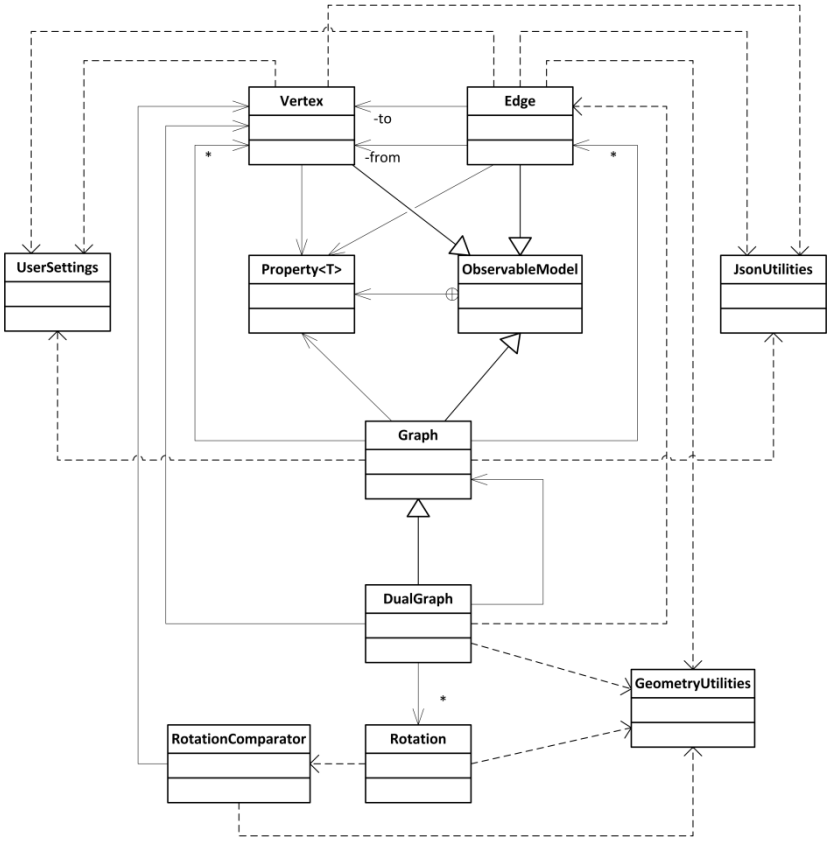

Ilustración 30: Diagrama de Clases. Modelo Grafos

La clase *Rotation* es utilizada en el proceso de construcción del grafo dual. La rotación de un vértice, denominado centro, es el conjunto de sus vértices vecinos ordenados de menor a mayor según el ángulo que forman con otro vértice de referencia. Esto permite recorrer los vértices vecinos en el sentido de las agujas del reloj o en sentido contrario. La clase *RotationComparator* es la encargada de comparar dos vértices según el ángulo entre el segmento desde el vértice centro hasta el vértice de referencia y el segmento desde el vértice centro hasta uno de los vértices.

La clase *ObservableModel* es una extensión de la clase *Observable* que proporciona una implementación mínima del patrón de diseño Observador en Java. Además, *ObservableModel* contiene la implementación de una clase interna *Property* que permite la notificación de cambios en una propiedad específica a los observadores subscritos.

La clase *JsonUtilities* se usa para manejar la serialización y deserialización de modelos a través del formato JSON. En este caso, es utilizada por los modelos grafo, vértice y arista. La clase *GeometryUtilities* aglomera distintos métodos de geometría en el plano. La clase *UserSettings* agrupa distintos parámetros globales de la aplicación pero que pueden ser modificados a lo largo de la ejecución.

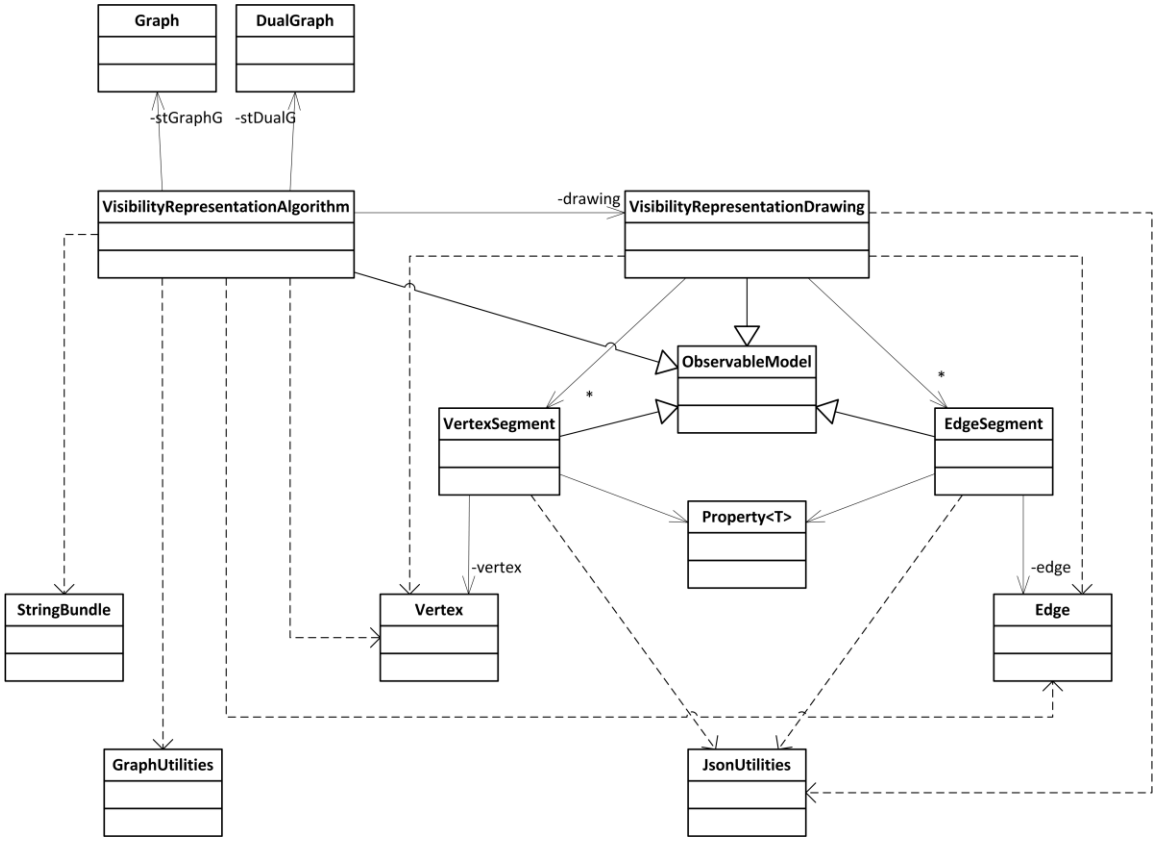

Ilustración 31: Diagrama de Clases. Modelo Algoritmo Visibilidad

El diagrama de la Ilustración 31 muestra la parte del modelo que se utiliza para implementar el algoritmo de visibilidad, es decir, el caso de uso 1. El concepto segmento vértice se implementa en la clase *VertexSegment* donde tiene atributos para sus coordenadas y el vértice asociado al segmento. El concepto segmento arista se implementa en la clase *EdgeSegment* donde tiene atributos para sus coordenadas y la arista asociada al segmento. El trazado de representación de visibilidad se implementa en la clase *VisibilityRepresentationDrawing* donde se mantienen contenedores con los segmentos vértices y aristas que forman dicho trazado.

El algoritmo de visibilidad que estudiamos en el capítulo 3 se implementa en la clase *VisibilityRepresentationAlgorithm*. Dentro se va construyendo la representación de visibilidad a partir de un grafo y su dual. Para determinar las numeraciones topológicas de los grafos se usan los métodos definidos en la clase *GraphUtilities* que cumplen este cometido. La clase *StringBundle* facilita el manejo de las cadenas de caracteres y es utilizado para extraer la explicación de cada paso del algoritmo que se notifica a los objetos observadores durante la ejecución.

En el diagrama de clases de la Ilustración 32 reflejamos las clases involucradas en la implementación del caso de uso 2, algoritmo de visibilidad con restricciones.

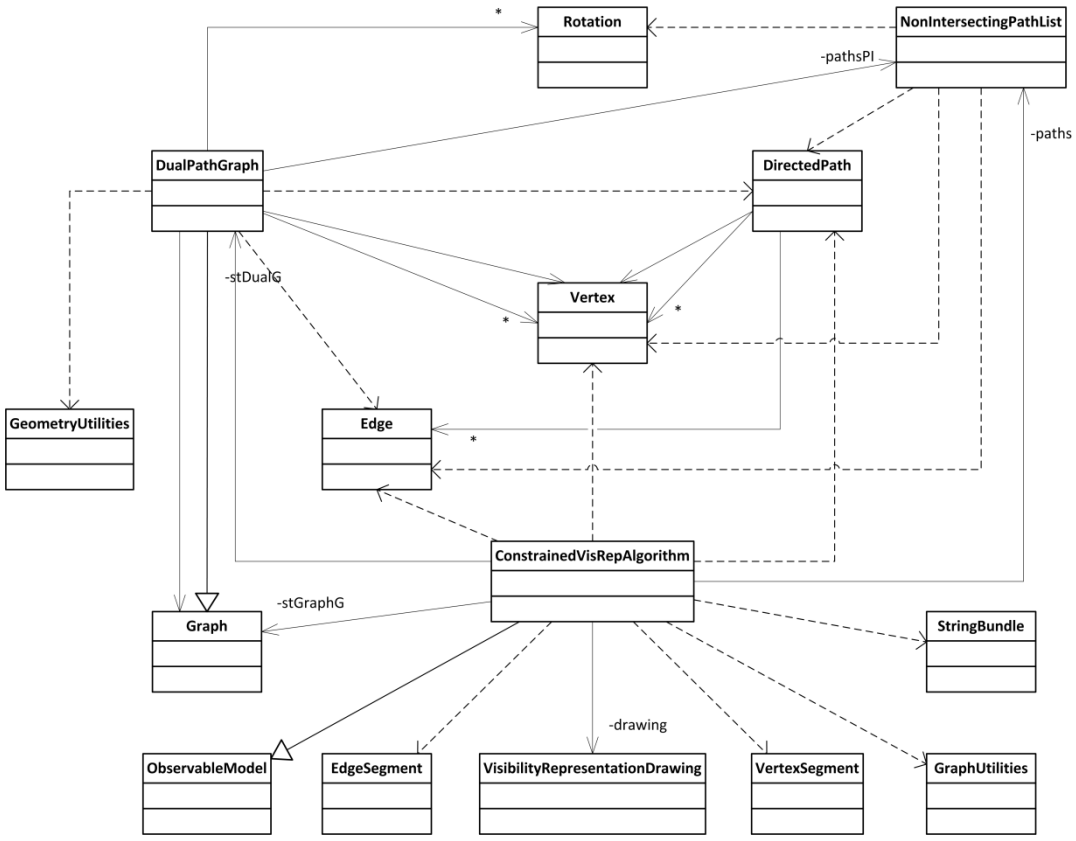

Ilustración 32: Diagrama de Clases. Modelo Algoritmo Visibilidad con restricciones

Primeramente, fue necesario definir la clase *DirectedPath* para representar un camino dirigido de vértices y aristas. Para eso esta clase cuenta con un conjunto de vértices, un conjunto de aristas, una tabla con el conjunto de aristas incidentes en cada vértice presentes en el camino, el vértice inicial, el vértice final y la longitud del camino. Cuando se crea un camino se comprueba que se cumplen todas las reglas que debe cumplir para ser un camino dirigido. También, esta clase tiene implementados otros métodos útiles, por ejemplo, para comprobar si un camino es vértices disjunto o aristas disjunto.

La clase *NonIntersectingPathList* es una extensión de una lista de caminos dirigidos con el objetivo de controlar la gestión de la lista y aplicar diferentes comprobaciones a todos los elementos presentes en ella. Cuando se agrega un nuevo camino a la lista se comprueba si es disjunto con los otros caminos ya presentes. Se pueden hacer operaciones como comprobar si un camino en la lista contiene un vértice o una arista dada, obtener todos los caminos que contienen un vértice o una arista dada, obtener el camino más largo que contiene un vértice dado. También, se pueden eliminar todos los caminos que contienen un vértice o una arista dada. Estos métodos entran dentro de la implementación del caso de uso 9, gestionar caminos disjuntos.

La clase *DualPathGraph* representa un grafo dual construido a partir de un grafo primal y un conjunto de caminos disjuntos dos a dos. El proceso de construcción es similar al del grafo dual pero con la variación explicada en el primer paso del algoritmo de visibilidad con restricciones visto en el capítulo 3. Este grafo extiende el comportamiento de la clase grafo. Tiene como atributos el grafo primal a partir del cual se construirá el grafo dual y la lista de caminos disjuntos que son proporcionados a la hora de crear el objeto. También, están identificados los vértices origen y destino del grafo primal, así como los del grafo dual. Además, se dispone de tablas con las caras izquierda y derecha de cara arista del grafo primal, a la vez que existe una tabla con las caras que representan a cada camino disjunto. Otro atributo presente en esta clase es una tabla con la rotación alrededor de cada vértice del grafo primal, que permite la construcción del dual.

En la clase *ConstrainedVisRepAlgorithm* se implementa el algoritmo visibilidad con restricciones. El proceso se inicia a partir de un st-grafo y una lista de caminos disjuntos. Luego, se inicia la ejecución del algoritmo, tal como vimos en la teoría. Primero, se invoca la construcción del grafo dual con los caminos mediante la clase *DualPathGraph*. A continuación, se determinan las numeraciones topológicas de ambos grafos con la clase *GraphUtilities*. Finalmente, se construye la representación de visibilidad utilizando las clases *VertexSegment*, *EdgeSegment* y *VisibilityRepresentationDrawing*.

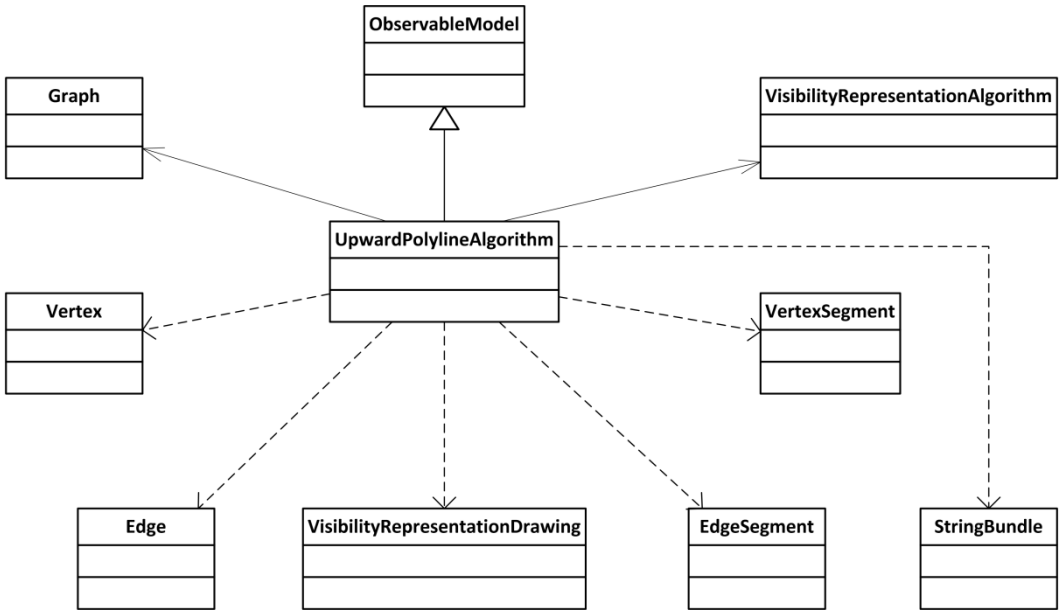

Ilustración 33: Diagrama de Clases. Modelo Algoritmo Poligonal

En el diagrama de la Ilustración 33 podemos observar las relaciones entre las clases que participan en la implementación del algoritmo poligonal, correspondiente al caso de uso 3. Dentro de la clase *UpwardPolylineAlgorithm* se lleva a cabo el algoritmo

poligonal. En este caso, la estrategia se apoya en el algoritmo de visibilidad. Se parte de un st-grafo al cual se le aplica el algoritmo de visibilidad. Luego, se modifican los segmentos vértices y aristas de la representación de visibilidad obtenida anteriormente, según vimos en la descripción teórica, hasta terminar creando el trazado poligonal del st-grafo.

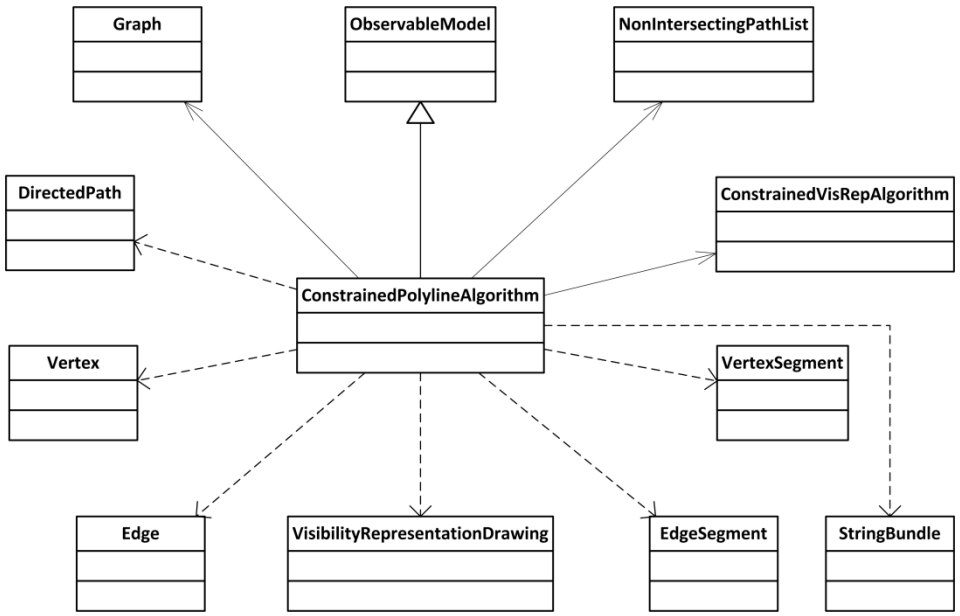

Ilustración 34: Diagrama de Clases. Modelo Algoritmo Poligonal con restricciones

En la Ilustración 34 se representan las clases que colaboran en la implementación del algoritmo poligonal con restricciones, correspondiente con el caso de uso 4. La clase *ConstrainedPolylineAlgorithm* es la encargada de desarrollar el algoritmo poligonal con restricciones. De manera similar al algoritmo poligonal, el poligonal con restricciones se ayuda del algoritmo de visibilidad con restricciones. En este caso, se parte de un st-grafo y un conjunto de caminos disjuntos con los cuales se construye una representación de visibilidad con restricciones inicial. A continuación, se modifican los segmentos vértices y aristas del trazado, según vimos en la teoría, hasta finalizar con el trazado poligonal con restricciones del st-grafo.

En el diagrama de clases de la Ilustración 35 podemos observar parte del modelo que está involucrado en la solución del caso de uso 5, algoritmo ortogonal. La clase *OrthogonalAlgorithm* implementa el algoritmo ortogonal, tal como lo vimos en la teoría. El proceso se inicia a partir de un st-grafo que se orienta bipolarmente en caso de no ser dirigido. Esto se logra a través del método de orientación implementado en la clase *GraphUtilities*. A continuación, se crea un conjunto de caminos disjuntos del digrafo. Luego, se ejecuta el algoritmo de visibilidad con restricciones con el digrafo y el conjunto de caminos. Finalmente, se crea un st-grafo a partir de la transformación de los segmentos vértices y aristas de la representación de visibilidad.

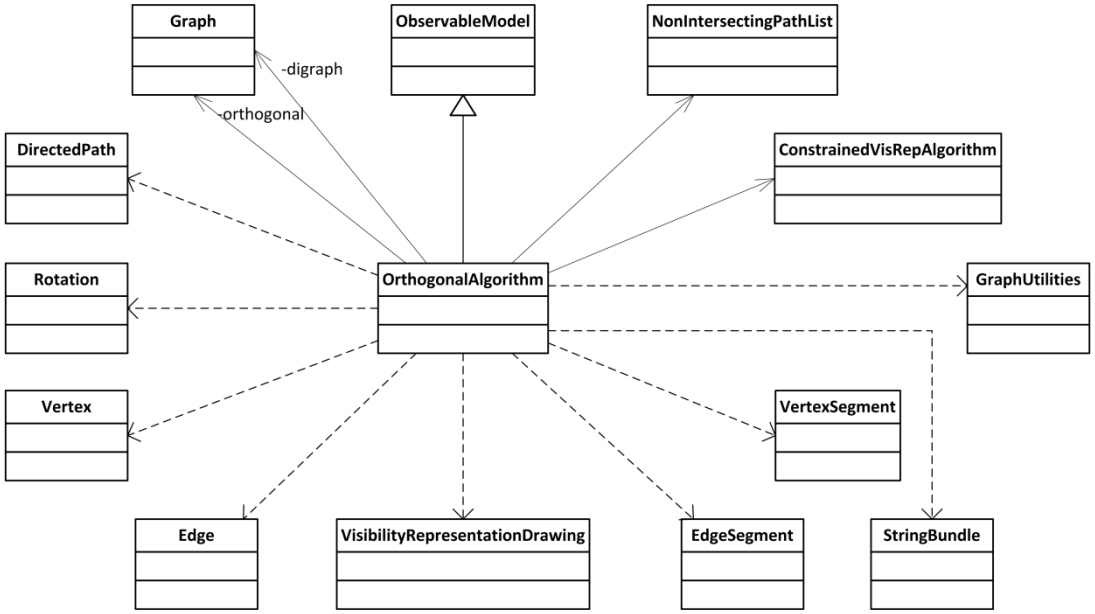

Ilustración 35: Diagrama de Clases. Modelo Algoritmo Ortogonal

En el diagrama de la Ilustración 36 mostramos las clases involucradas en la implementación del algoritmo de dominancia rectilíneo, correspondiente al caso de uso 6.

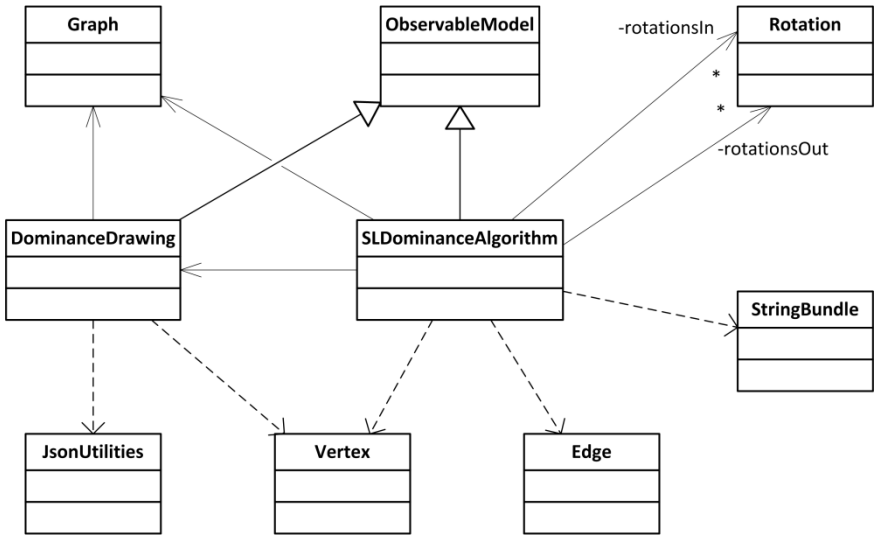

Ilustración 36: Diagrama de Clases. Modelo Algoritmo Dominancia rectilíneo

La clase *DominanceDrawing* está creada para representar el trazado de dominancia que está basado en un grafo con coordenadas enteras. En la clase *SLDominanceAlgorithm* se desarrolla el algoritmo de dominancia rectilíneo tal como estudiamos en el apartado correspondiente del capítulo anterior. La ejecución se inicia a partir de un st-grafo para cuyos vértices se construyen dos rotaciones, una con los

vértices vecinos de las aristas entrantes y otra con los vértices vecinos de las aristas salientes. Esto genera los conjuntos de rotaciones entrantes y rotaciones salientes. Luego, se le asignan las coordenadas enteras preliminares a los vértices del grafo final. Finalmente, se hace una compactación de las coordenadas del grafo.

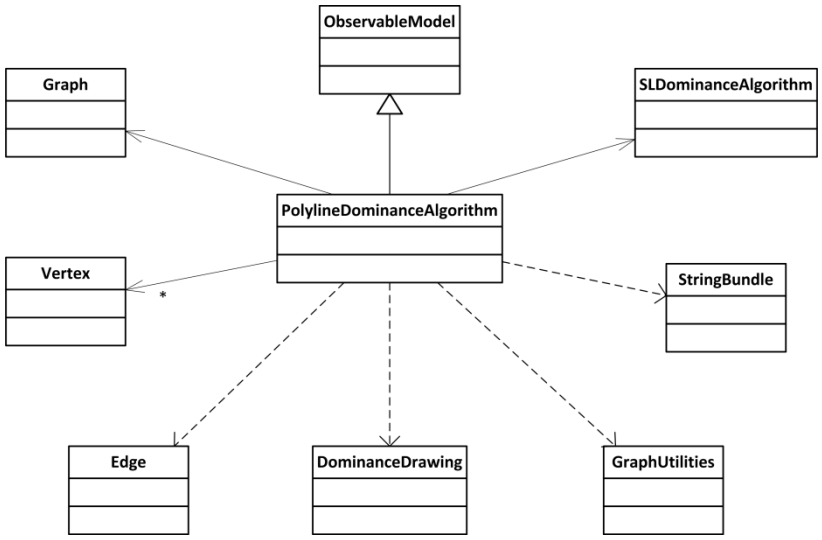

Ilustración 37: Diagrama de Clases. Modelo Algoritmo Dominancia poligonal

En la Ilustración 37 se representan las clases que colaboran en la implementación del algoritmo de dominancia poligonal, correspondiente con el caso de uso 7. En la clase *PolylineDominanceAlgorithm* se implementa el algoritmo de dominancia poligonal. Primero, se convierte el st-grafo inicial en reducido reemplazando cada arista transitiva por dos nuevas aristas e introduciendo un vértice ficticio que las une. En este paso se van añadiendo los vértices ficticios creados a una lista. A continuación, se construye el trazado de dominancia rectilíneo del grafo reducido creado mediante la clase correspondiente. Finalmente, se sustituyen todos los vértices ficticios por codos dando lugar al trazado de dominancia poligonal final.

A continuación, vamos a describir el diagrama de clases de la Ilustración 38, en el cual mostramos las principales clases que forman la Interfaz Gráfica de Usuario (GUI, en inglés). La clase *GraphDrawing* es la clase principal de la aplicación. En ella es donde empieza la ejecución y es la encargada de establecer los parámetros generales del aspecto y el comportamiento de la GUI. Una vez realizado esto, la clase crea la ventana principal representada por la clase *MainWindow*. A partir de la ventana principal se van incorporando los demás componentes al sistema, tales como los menús y las ventanas internas de edición de grafos y ejecución de algoritmos.

Dentro de la barra de menú contamos ítems que están relacionados directamente con un caso de uso. Los ítems se agrupan en menú de archivo, menú de edición, menú de algoritmos, menú de ventanas y menú de ayuda. Todas las acciones relacionadas con

la activación de un ítem se implementan dentro de la clase *MainWindow* o se pide la colaboración a objetos de otras clases para llevar a cabo el objetivo. Por ejemplo, la clase *NewGraphDialog* es un diálogo que participa en la creación de un nuevo grafo, caso de uso 14. Otro ejemplo de colaboración es la clase *AboutDialog* que se utiliza en la implementación del caso de uso 48, mostrar información de la aplicación.

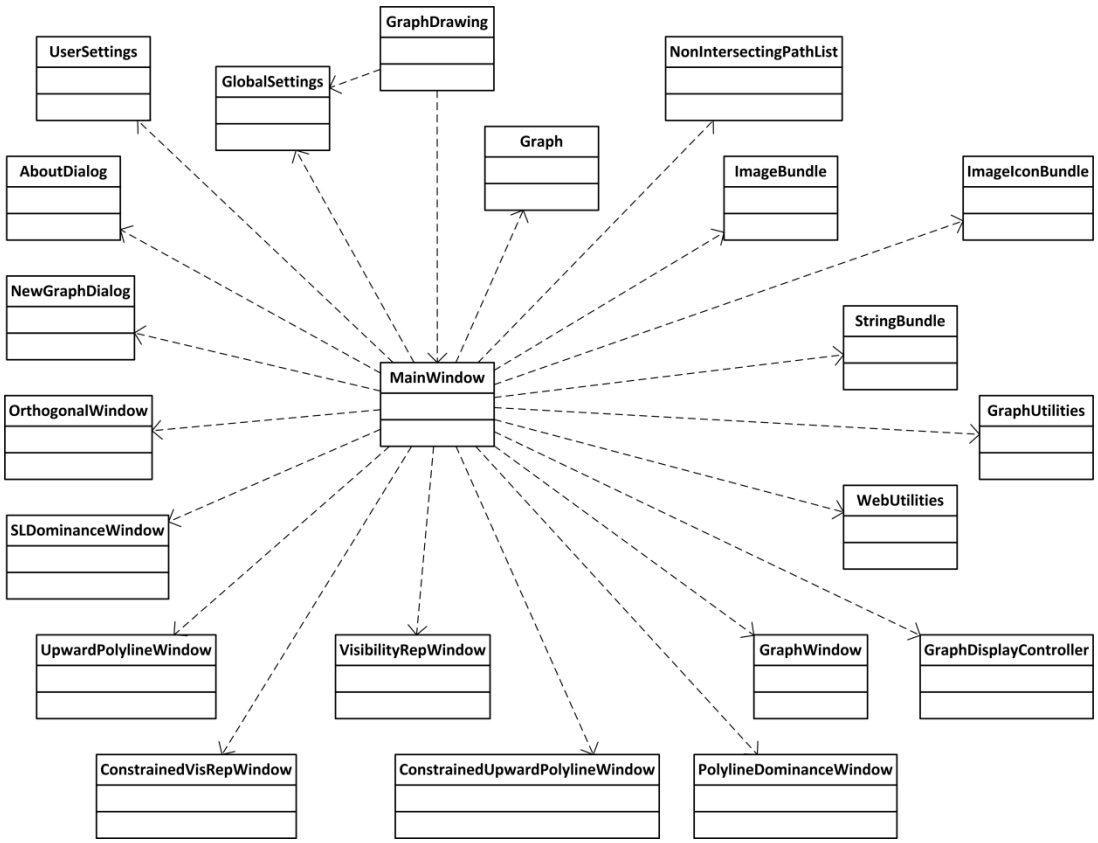

Ilustración 38: Diagrama de Clases. Interfaz Gráfica de Usuario

La clase *GraphWindow* se desarrolló para representar la ventana interna de edición de grafos. Cada ventana tiene asociado un controlador, en este caso es la clase *GraphDisplayController*, donde se implementan la mayoría de las operaciones que se realizan dentro de estas ventanas. Las clases que se utilizan para las ventanas internas de ejecución de los algoritmos son *VisibilityRepWindow*, *ConstrainedVisRepWindow*, *UpwardPolylineWindow*, *UpwardPolylineWindow*, *OrthogonalWindow*, *SLDominanceWindow* y *PolylineDominanceWindow*.

En el diagrama de clases de la Ilustración 39 podemos analizar con más detenimiento las relaciones entre los elementos de la interfaz gráfica y los controladores de las operaciones de edición de grafos. En la mayoría de los casos los controladores están integrados en los componentes de las interfaces gráficas implementando dentro de estos las acciones.

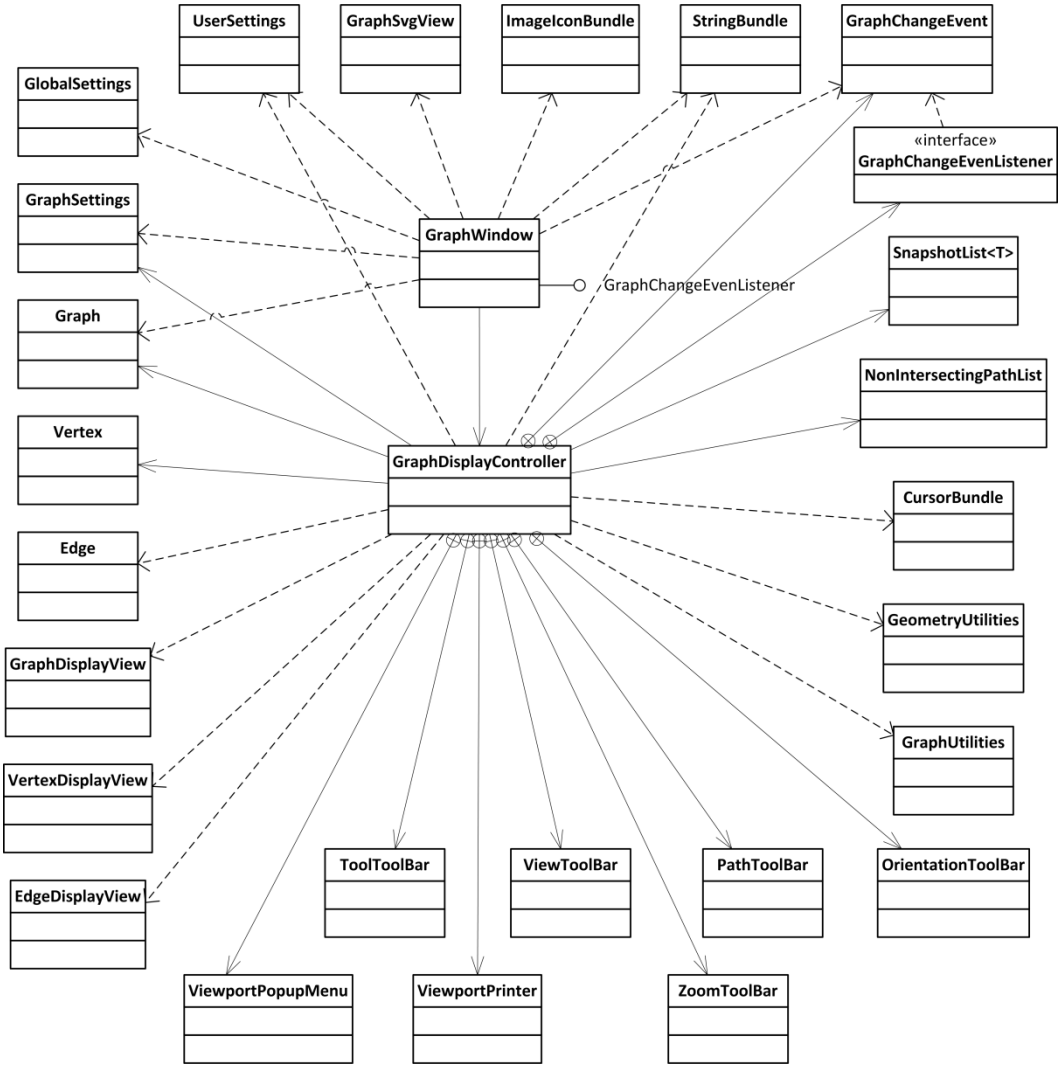

Ilustración 39: Diagrama de Clases. Controlador Grafo

En la clase *GraphWindow* se realizan operaciones como guardar el grafo y guardar el grafo como, casos de uso 17 y 18, respectivamente. Estas operaciones son invocadas desde la ventana principal al producirse el evento respectivo dentro del menú. También, es la propia ventana la que lleva el control de si el grafo ha sido modificado por alguna acción de edición dentro del controlador. La interfaz *GraphChangeEvenListener* es utilizada para informar a la ventana desde el controlador que se ha producido una modificación. La ventana tiene en cuenta esto a la hora de cerrar y confirmar si se desea guardar el grafo.

La clase *GraphDisplayController* es una de las más complejas de todo el sistema porque en ella se llevan a cabo un gran número de operaciones, todas ellas relacionadas con la edición de grafos. Dentro de esta clase tenemos atributos como el grafo que estamos editando, el conjunto de caminos disjunto seleccionado en el grafo, la configuración del grafo, el historial de modificaciones del grafo, entre otros. La clase *GraphSettings*

representa la configuración, en la cual se especifica si mostrar las etiquetas y los pesos de las aristas y de los vértices. El historial está formado por una lista de cadenas de caracteres gracias al formato JSON. Para manejar esta cadena y poder deshacer o rehacer operaciones fácilmente, se utiliza la clase *SnapshotList* perteneciente al paquete de clases útiles.

La clase *GraphDisplayController* incorpora varias clases internas para formar una barra de herramientas con las operaciones de edición. La clase *ToolToolBar* es la encargada de establecer el modo de edición de grafos actual, el cual puede ser seleccionar elementos, añadir elementos y eliminar elementos. La clase *ViewToolBar* sirve para modificar la configuración estableciendo que parámetros del grafo mostrar a la hora de dibujar el grafo en pantalla. La clase *ZoomToolBar* contiene los botones que permiten ejecutar las distintas opciones de zoom sobre el grafo. La clase *OrientationToolBar* permite ejecutar el algoritmo de orientación bipolar. La clase *PathToolBar* sirve para gestionar la lista de caminos disjuntos. La clase *ViewportPopupMenu* no forma parte de la barra de herramientas sino que es un menú desplegable que aparece al hacer clic derecho sobre uno o varios elementos del grafo permitiendo editar sus parámetros. Por último, la clase *ViewportPrinter* realiza las tareas para imprimir el grafo cuando es demandado desde el menú en la clase principal.

El controlador del grafo también es el encargado de llamar a las clases correspondientes para dibujar el modelo en pantalla. De esta tarea se encarga la clase *GraphDisplayView* apoyándose en *VertexDisplayView* y *EdgeDisplayView*.

En la Ilustración 40 podemos observar el diagrama de clases de los elementos de la interfaz gráfica y los controladores del algoritmo visibilidad. La clase *VisibilityRepWindow* representa la ventana interna de ejecución del algoritmo visibilidad. La clase *VisibilityRepDisplayController* es el controlador asociado al algoritmo visibilidad. Dentro contamos con el panel de ejecución paso a paso que al ser usado por todos los algoritmos se externalizó en la clase *ExecutionControlPanel*. La interacción entre el controlador y el panel de ejecución se realiza a través de la implementación de la interfaz *ExecutionControlPanelActions*. La interfaz requiere que los controladores desarrollen las operaciones de mostrar etapa anterior, mostrar etapa siguiente, finalizar etapas y cancelar etapas, correspondientes con los casos de uso desde el 10 hasta el 13. Cuando se activa una de estas operaciones, el panel de ejecución indica al controlador correspondiente que lleve a cabo la acción.

La operación de cancelar etapas trae consigo que se cierre la ventana de ejecución del algoritmo. Para esto es necesario crear un evento desde el controlador que le indique a la ventana el cierre. La clase *StateSupport* está pensada para ayudar en estos casos al gestionar una lista de objetos interesados en cierto evento. La ventana solo necesita implementar la interfaz para realizar la comunicación y darse de alta en la lista del evento a través del controlador.

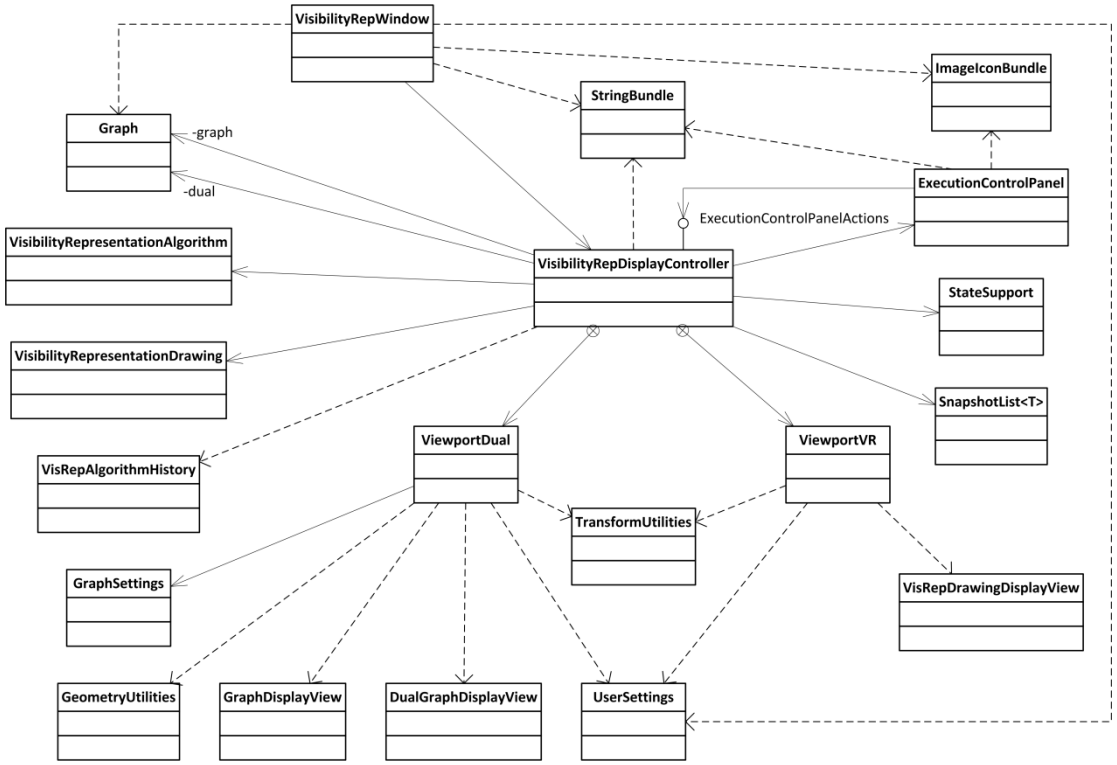

Ilustración 40: Diagrama de Clases. Controlador Algoritmo Visibilidad

La clase *VisibilityRepDisplayController* tiene como objetivo mostrar la evolución de la ejecución del algoritmo. Para conseguir esto se muestra, por un lado, el grafo al cual se le aplica el algoritmo junto a la construcción de su grafo dual. Por otro lado, se enseña la creación de la representación de visibilidad. El controlador recibe una notificación cada vez que el algoritmo realiza un cambio sobre algunos de los elementos que se muestran, dado que se subscribió como objeto observador. Cada una de estas notificaciones hacen que el controlador cree un nuevo paso en el historial de la ejecución por el cual se podrá navegar con el panel de ejecución. La gestión del historial se lleva a cabo con la clase *SnapshotList*, pero en este caso utilizamos la clase *VisRepAlgorithmHistory* para almacenar toda la información necesaria convertida al formato JSON. Cada elemento del historial incluye la variación que ha sufrido el grafo, su dual, la representación de visibilidad y la explicación del paso que se ha realizado en el algoritmo. De esta forma, la estrategia que sigue el controlador es ejecutar al inicio todo el algoritmo creando el historial con cada modificación y luego reconstruir los elementos a medida que se avance o se retroceda por el historial.

Dentro del controlador tenemos dos clases internas, *ViewportDual* y *ViewportVR*, que son los componentes encargados de mostrar el grafo con su dual y la representación de visibilidad, respectivamente. *ViewportDual* se apoya en la clase *GraphDisplayView* para dibujar el grafo y en la clase *DualGraphDisplayView* para el dual. *ViewportVR* se

ayuda de la clase *VisRepDrawingDisplayView* para mostrar la representación de visibilidad.

Es importante decir que el resto de controladores de los algoritmos siguen el estilo de desarrollo del algoritmo visibilidad. Por lo tanto, consideramos que resulta de menor interés exponer los diagramas uno por uno cuando es muy sencillo comprender el funcionamiento del resto a partir del que acabamos de analizar.

4.2 Diseño de Bajo Nivel

Esta parte del diseño la vamos a dedicar a explicar a nivel de pseudocódigo el algoritmo utilizado en la construcción del grafo dual.

Algoritmo Construir Grafo Dual

Entrada: st-grafo *G*, vértice *s*, vértice *t*
Salida: st-grafo dual, cara *s*, cara *t*, colección *derecha*(*v*), colección *izquierda*(*v*), colección *derecha*(*e*), colección *izquierda*(*e*).
Pasos:

1. Crear cara *s* ubicándola por debajo del borde inferior de *G*.
   1.1. Recorrer todos los vértices de *G* determinando cual está ubicado en el punto más bajo del plano.
   1.2. Crear nuevo vértice del grafo dual que será la cara *s*.
   1.3. Ubicar la cara *s* por debajo del borde inferior determinado.
2. Crear cara *t* ubicándola por arriba del borde superior de *G*.
   2.1. Recorrer todos los vértices de *G* determinando cual está ubicado en el punto más alto del plano.
   2.2. Crear nuevo vértice del grafo dual que será la cara *t*.
   2.3. Ubicar la cara *t* por arriba del borde superior determinado.
3. Determinar las aristas que están conectadas con las caras externas.
   3.1. Para cada vértice de *G* determinar si es el vértice más cercano al punto origen de coordenadas.
   3.2. Hacer que una variable denominada vértice actual sea el vértice más cercano al origen.
   3.3. Hacer que una variable denominada vértice anterior sea un vértice ubicado en el origen de coordenadas.
   3.4. Recorrer el grafo seleccionando la arista entre los vecinos del vértice actual que forme el menor ángulo en el sentido de las agujas del reloj con la recta definida por los vértices actual y anterior.

3.5. Hacer que la variable vértice siguiente sea el vértice vecino del vértice actual unido por la arista de menor ángulo.
3.6. Si la arista de menor ángulo va en dirección vértice actual a vértice siguiente, hacer que la cara derecha de la arista sea la cara externa *t*.
3.7. Si la arista de menor ángulo va en dirección vértice siguiente a vértice actual, hacer que la cara izquierda de la arista sea la cara externa *s*.
3.8. Actualizar las variables haciendo que el vértice anterior sea igual al vértice actual y que el vértice actual sea igual al siguiente.
3.9. Repetir este proceso mientras el vértice actual sea distinto del vértice origen del recorrido.

Para comprender mejor este paso del algoritmo podemos ver la Ilustración 41.

4. Determinar las caras internas de *G*.
   4.1. Para cada arista *e* de *G* que no tenga determinada su cara derecha determinar las aristas que rodean esta cara.
      4.1.1. Crear una nueva cara *f*.
      4.1.2. Hacer que una variable denominada vértice actual sea el vértice destino de la arista *e*.
      4.1.3. Hacer que una variable denominada vértice anterior sea el vértice origen de la arista *e*.
      4.1.4. Mientras el vértice actual sea distinto del vértice origen recorrer el grafo seleccionando la arista entre los vecinos del vértice actual que forme el menor ángulo en el sentido de las agujas del reloj con la recta definida por los vértices actual y anterior.
      4.1.5. Hacer que la variable vértice siguiente sea el vértice vecino del vértice actual unido por la arista de menor ángulo.
      4.1.6. Si la arista de menor ángulo va en dirección vértice actual a vértice siguiente, hacer que la cara derecha de la arista sea la cara *f*.
      4.1.7. Si la arista de menor ángulo va en dirección vértice siguiente a vértice actual, hacer que la cara izquierda de la arista sea la cara *f*.
      4.1.8. Actualizar las variables haciendo que el vértice anterior sea igual al vértice actual y que el vértice actual sea igual al siguiente.
      4.1.9. Insertar la nueva cara *f* como un vértice del grafo dual.

      Para comprender mejor este paso del algoritmo podemos ver la Ilustración 42.
5. Crear las aristas del grafo dual.
   5.1. Para cada arista *e* de *G* crear una arista en el grafo dual que va desde la cara izquierda a la cara derecha de la arista *e*.
   5.2. Insertar la arista creada en el grafo dual.

6. Determinar las caras izquierda y derecha de los vértices de *G*.
   6.1. Para cada vértice *v* de *G* hacer que:
      6.1.1. Si *v* es el vértice *s* o el vértice *t*, su cara izquierda es la cara *s* y su cara derecha es la cara *t*.
      6.1.2. En otro caso, recorrer las aristas entrantes comparándolas con cada arista saliente del vértice *v*.
         6.1.2.1. Si la cara izquierda de la arista entrante es la misma que la cara izquierda de la arista saliente, hacer que esa sea la cara izquierda del vértice *v*.
         6.1.2.2. Si la cara derecha de la arista entrante es la misma que la cara derecha de la arista saliente, hacer que esa sea la cara derecha del vértice *v*.

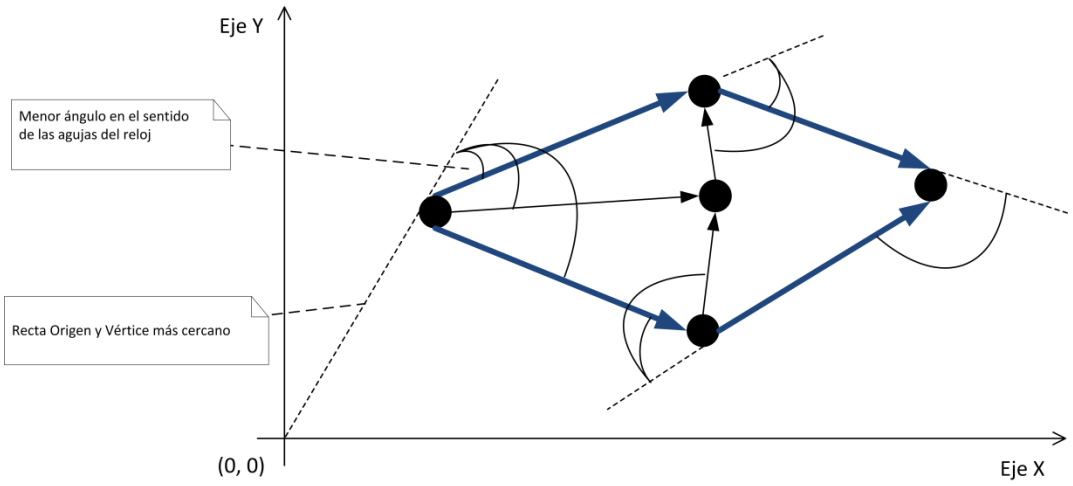

Ilustración 41: Algoritmo Construir Grafo Dual. Caras Externas

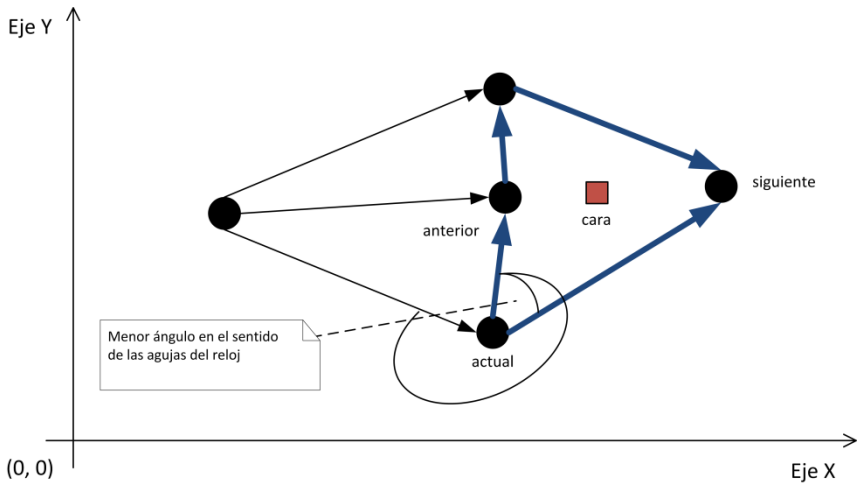

Ilustración 42: Algoritmo Construir Grafo Dual. Caras Internas

4.3 Funcionalidades extras

4.3.1 Formato de almacenamiento de los grafos (*.vsg)

Cuando empezamos a trabajar con la aplicación creando y editando grafos, surge la necesidad de almacenar estos grafos para poder utilizarlos más adelante en sesiones posteriores. Para resolver este problema de serialización de los grafos, utilizamos el mismo mecanismo que venía ya implementado en el proyecto base [Beh11]. La idea es crear un fichero con la extensión *vsg* donde se guarda toda la información imprescindible que permite retomar la edición de un grafo donde la habíamos dejado.

Todos los datos de un grafo se almacenan en un fichero .*vsg* usando JSON (*JavaScript Object Notation*) [Jso11], el cual es un formato ligero de representación de objetos que no requiere la utilización de XML. Debido a su sencillez, en algún caso perjudicial, se ha generalizado su uso en aplicaciones que necesitan almacenar objetos de forma permanente o enviarlos a través de la red.

Básicamente, cada objeto se convierte, utilizando la clase *JsonUtilities*, en una cadena de caracteres. Cada atributo de un objeto que se quiera representar debe tener una cadena de caracteres con el nombre del atributo seguido de dos puntos y la cadena que representa el valor del atributo. Luego, esta cadena de caracteres se convierte en el objeto que queremos utilizando un analizador sintáctico muy sencillo.

Una arista se representa con la siguiente cadena:

{ "weight" : 1.0, "isSelected" : false, "color" : -1, "to.id" : "41e9d178-a592-4c4b-a881-7e36975e0b8e", "from.id" : "6e76be44-2171-47f5-a856-dfa14baddf8e", "isLinear" : true, "isDirected" : true, "thickness" : 1.5, "label" : "e" }

Un vértice se representa con la siguiente cadena:

{ "id" : "6e76be44-2171-47f5-a856-dfa14baddf8e", "weight" : 1.0, "isSelected" : false, "color" : -1, "radius" : 5.0, "label" : "0", "y" : 256.0, "x" : 170.0 }.

4.3.2 Internacionalización de la aplicación

La aplicación está pensada para que pueda ser usada por todos los usuarios que deseen comprender el funcionamiento de los algoritmos de trazado de grafos. Por este motivo, resulta muy útil que la aplicación pueda tener toda la información disponible en varios idiomas. Debido a la mayor utilización del inglés en todo el mundo, el lenguaje por defecto de la aplicación es este.

Para desarrollar esta funcionalidad se utilizaron los ficheros *.properties* y la clase que ofrece Java para manejarlos, *ResourceBundle* [Ora11]. En nuestro proyecto, el manejo

de los *strings* está encapsulado en la clase *StringBundle*. Este trabajo estaba inicialmente implementado en el proyecto [Beh11], por lo tanto, solo lo hemos adaptado a nuestras necesidades.

Los archivos .*properties* están compuestos por líneas con los valores *clave=traducción*, donde *clave* es un identificador único de un texto dado y *traducción* es el texto traducido al lenguaje específico del fichero. Java permite detectar el lenguaje con el que está configurada la máquina en la que se ejecuta la aplicación y, así, utilizar la traducción a ese lenguaje si está disponible. En caso de no estar disponible, se utiliza la traducción por defecto.

5 MANUAL DE USUARIO

En este capítulo se explicará cómo usar la aplicación detalladamente a través de su manual de usuario. Partiremos de una visión global para luego ir profundizando en las opciones de los menús, el panel de edición de grafos y las ventanas de ejecución de los algoritmos.

5.1 Barra de menús

En la Ilustración 43 podemos observar toda la barra de menús de la aplicación. Vemos que existen conjuntos de menús para gestionar los archivos de grafos, editar los grafos, iniciar la ejecución de algoritmos, organizar las ventanas abiertas dentro de la aplicación y consultar la ayuda.

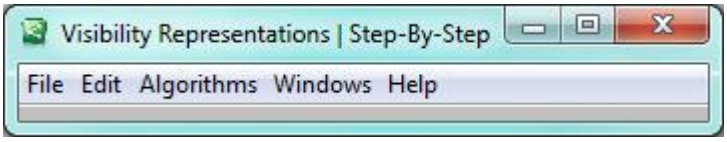

Ilustración 43: Barra de menús

En la Ilustración 44 vemos las opciones del menú *File*:

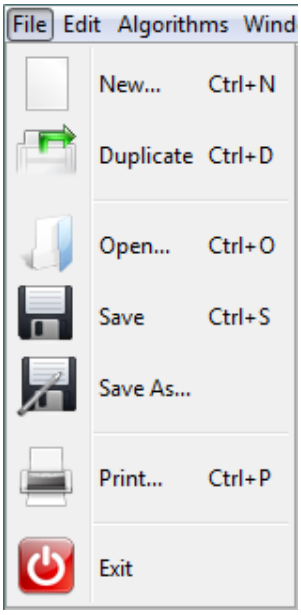

Ilustración 44: Menú File

A continuación, se describen todas las opciones más profundamente:

- *New:* crea un nuevo grafo en un panel de edición. El usuario debe seleccionar las características del grafo en un diálogo que se muestra al pulsar esta opción, Ilustración 45. Una vez hecho esto, se abre un panel de edición con el nuevo grafo sin guardar. La combinación de teclas de acceso rápido para esta opción de menú es Ctrl + N.
    - *Graph:* tipo de grafo que queremos crear. Puede ser no dirigido (opción *Undirected edges*) o dirigido (opción *Directed edges*). Lógicamente, una decisión invalida la otra.
    - *Other characteristics:* otras características del grafo. Se pueden permitir los ciclos (*Allow cycles*), los bucles (*Allow loops*) y múltiples aristas entre dos vértices (*Allow multiples*). Si se seleccionan los bucles o las aristas múltiples, se obliga a que se permita la existencia de bucles. Si no se permiten los ciclos, se invalidan los bucles y las aristas múltiples.

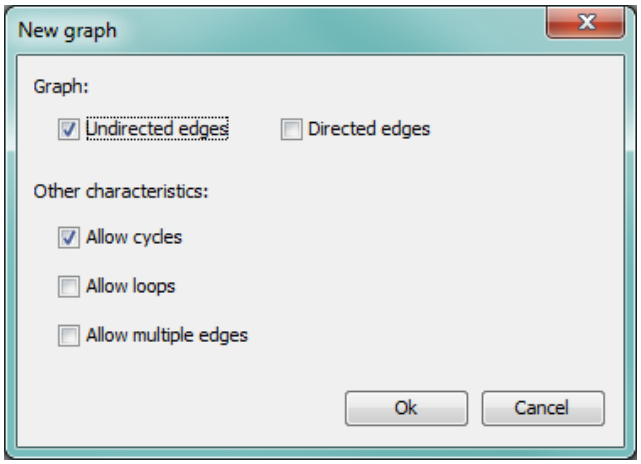

Ilustración 45: Menú File. New graph

- *Duplicate:* duplica el grafo que está siendo usado y lo abre en un nuevo panel de edición. El nuevo grafo está sin guardar. La combinación de teclas de acceso rápido para esta opción de menú es Ctrl + D.
- *Open:* abre un grafo guardado previamente en un archivo con el formato utilizado por la aplicación (*.vsg*). Cuando se pulsa esta opción, se abre una ventana que permite navegar por el sistema de ficheros y seleccionar el archivo que se desea abrir. La combinación de teclas de acceso rápido para esta opción de menú es Ctrl + O.
- *Save:* guarda los cambios realizados a un grafo en su archivo asociado de extensión *.vsg*. Si es la primera vez que se guarda el grafo, entonces esta opción es equivalente a *Save As*. La combinación de teclas de acceso rápido para esta opción de menú es Ctrl + S.
- *Save As:* guarda un grafo en un nuevo archivo. Cuando se pulsa esta opción, se abre una ventana que permite navegar por el sistema de ficheros, seleccionar la ubicación del archivo, el tipo de archivo y el nombre. El tipo de archivo puede

ser Portable Network Graphics (*.png*), Scalable Vector Graphics (*.svg*) y VisiGraph Graph (*.vsg*). Los dos primeros tipos son formatos gráficos y el tercero es el único que permite reutilizar el grafo más adelante.

- *Print:* inicia el proceso de impresión del grafo actual. Cuando se selecciona esta opción, se abre la ventana de configuración de impresión del sistema. La combinación de teclas de acceso rápido para esta opción de menú es Ctrl + P.
- *Exit:* cierra la aplicación. Al producirse esta acción no se tiene en cuenta si algún grafo no ha sido guardado, por tanto se pierde el trabajo realizado.

En la Ilustración 46 podemos observar las opciones del menú *Edit*:

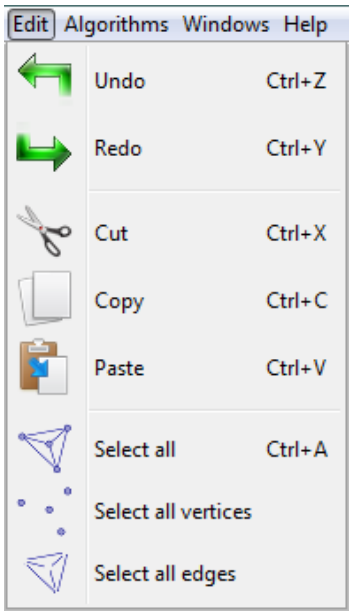

Ilustración 46: Menú Edit

Ahora, se explican cada una de estas opciones:

- *Undo:* deshace la última operación de edición realizada sobre un grafo. La combinación de teclas de acceso rápido para esta opción de menú es Ctrl + Z.
- *Redo:* rehace la última operación de edición que fue deshecha. La combinación de teclas de acceso rápido para esta opción de menú es Ctrl + Y.
- *Cut:* corta los elementos seleccionados de un grafo. La combinación de teclas de acceso rápido para esta opción de menú es Ctrl + X.
- *Copy:* copia los elementos seleccionados de un grafo. La combinación de teclas de acceso rápido para esta opción de menú es Ctrl + C.
- *Paste:* pega los elementos de un grafo que fueron copiados o cortados anteriormente. La combinación de teclas de acceso rápido para esta opción de menú es Ctrl + V.

- *Select all:* selecciona todos los elementos de un grafo. La combinación de teclas de acceso rápido para esta opción de menú es Ctrl + A.
- *Select all vertices:* selecciona todos los vértices de un grafo.
- *Select all edges:* selecciona todas las aristas de un grafo.

En la Ilustración 47 podemos observar las opciones del menú *Algorithms*:

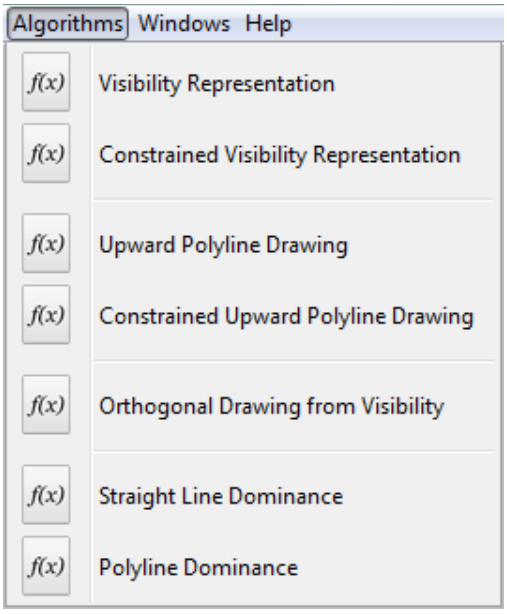

Ilustración 47: Menú Algorithms

La descripción de cada opción del menú es la siguiente:

- *Visibility Representation:* inicia la ejecución del algoritmo visibilidad. El grafo debe ser st-grafo, plano y tener los vértices fuente y sumidero en la cara externa. Se abre la ventana de ejecución paso a paso de este algoritmo.
- *Constrained Visibility Representation:* inicia la ejecución del algoritmo visibilidad con restricciones. Las restricciones se expresan mediante el conjunto de caminos disjuntos. El grafo debe ser st-grafo, plano y tener los vértices fuente y sumidero en la cara externa. Se abre la ventana de ejecución paso a paso de este algoritmo.
- *Upward Polyline Drawing:* inicia la ejecución del algoritmo poligonal. El grafo debe ser st-grafo, plano y tener los vértices fuente y sumidero en la cara externa. Se abre la ventana de ejecución paso a paso de este algoritmo.
- *Constrained Upward Polyline Drawing:* inicia la ejecución del algoritmo poligonal con restricciones. Las restricciones se expresan mediante el conjunto de caminos disjuntos. El grafo debe ser st-grafo, plano y tener los vértices fuente y sumidero en la cara externa. Los caminos no pueden tener vértices

internos en común. Se abre la ventana de ejecución paso a paso de este algoritmo.

- *Orthogonal Drawing from Visibility:* inicia la ejecución del algoritmo ortogonal. El grafo debe ser st-grafo, plano, biconexo y el grado de los vértices menor o igual a cuatro. Se abre la ventana de ejecución paso a paso de este algoritmo.
- *Straight Line Dominance:* inicia la ejecución del algoritmo dominancia rectilíneo. El grafo debe ser st-grafo, plano, reducido y tener los vértices fuente y sumidero en la cara externa. Se abre la ventana de ejecución paso a paso de este algoritmo.
- *Polyline Dominance:* inicia la ejecución del algoritmo dominancia poligonal. El grafo debe ser st-grafo, plano y tener los vértices fuente y sumidero en la cara externa. Se abre la ventana de ejecución paso a paso de este algoritmo.

En la Ilustración 48 podemos observar las opciones del menú *Windows*:

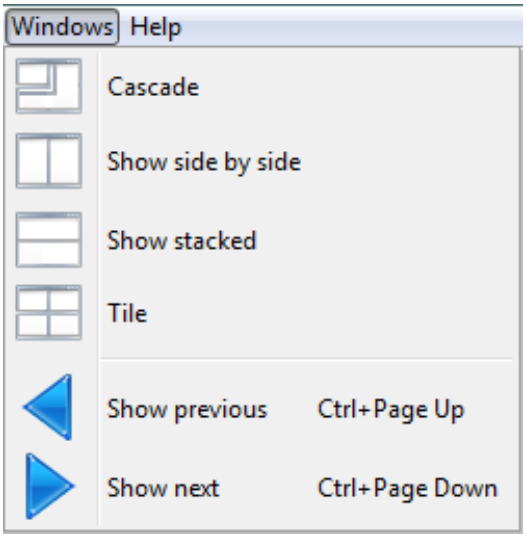

Ilustración 48: Menú Windows

La descripción de cada opción del menú es la siguiente:

- *Cascade:* organiza las ventanas abiertas en forma de cascada.
- *Show side by side:* organiza las ventanas abiertas en forma de mosaico vertical.
- *Show stacked:* organiza las ventanas abiertas en forma de mosaico horizontal.
- *Tile:* organiza las ventanas abiertas en forma de cuadrícula.
- *Show previous:* muestra la ventana anterior a la actual. La combinación de teclas de acceso rápido para esta opción de menú es Ctrl + RePág.
- *Show next:* muestra la ventana siguiente a la actual. La combinación de teclas de acceso rápido para esta opción de menú es Ctrl + AvPág.

En la Ilustración 49 podemos observar las opciones del menú *Help*:

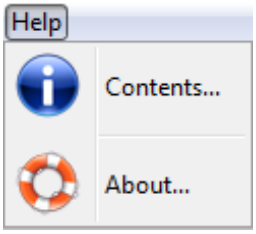

Ilustración 49: Menú Help

La descripción de cada opción del menú es la siguiente:

- *Contents:* accede a la página web del proyecto. Se abre un navegador web que se dirige a la dirección establecida.
- *About:* muestra la información básica de la aplicación. Se abre una ventana de diálogo con los datos.

5.2 Ventana de edición de grafos

En este apartado se describe la ventana de edición de grafos. En la Ilustración 50 se puede ver que se compone de una barra de herramientas y un panel donde se representa el grafo.

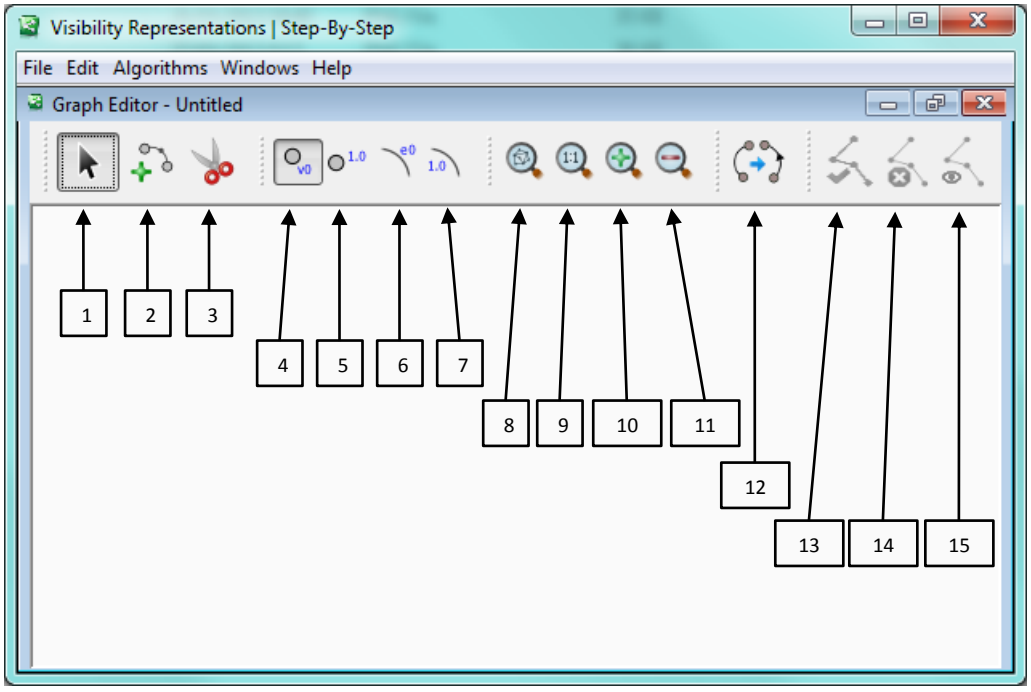

Ilustración 50: Ventana de edición de grafos

Los elementos de la barra de herramientas se encuentran divididos en cinco grupos. El primer grupo, compuesto por los botones 1, 2 y 3, está pensado para definir la función

del cursor, es decir, seleccionar, añadir o eliminar elementos. El segundo grupo, integrado por los botones 4, 5, 6 y 7, está dedicado a establecer que parámetros de los elementos del grafo mostrar. El tercer grupo, formado por los botones 8, 9, 10 y 11, permite aplicar distintas operaciones de zoom sobre el grafo. El cuarto grupo, que solo es el botón 12, está destinado a la orientación del grafo. La quinta agrupación de botones, compuesta por el 13, el 14 y el 15, se dedica a la gestión del conjunto de caminos disjuntos del grafo. A modo de ayuda, si se sitúa el cursor del ratón sobre uno de estos botones, se mostrará un pequeño texto describiendo para qué se utiliza.

La descripción de cada uno de los botones de la barra de herramientas es la siguiente:

1. *Select / Move:* permite fijar la funcionalidad del cursor para seleccionar o mover elementos del grafo. La selección se puede aplicar sobre un elemento de forma individual o englobando varios dentro del rectángulo que se forma al arrastrar el cursor manteniendo pulsado un clic. Para mover elementos, primero debemos seleccionarlos y luego arrastrarlos hasta la posición deseada manteniendo pulsado un clic. Para seleccionar una arista es necesario incluir su zona central.
2. *Add vertices and edges:* permite fijar la funcionalidad del cursor para añadir elementos al grafo. Para agregar un nuevo vértice basta con hacer clic izquierdo en un espacio en blanco del panel. Hay dos formas posibles de añadir una arista. Una manera es hacer clic sobre un vértice origen y luego sobre un vértice destino, de tal modo que se añade una arista entre estos dos vértices. Otra forma es hacer clic sobre un vértice origen y luego hacer clic en un espacio vacío del panel, así se agrega una arista entre el vértice origen y un nuevo vértice creado donde se realizó el segundo clic.
3. *Remove elements:* permite fijar la funcionalidad del cursor para eliminar elementos al grafo. Podemos eliminar un elemento haciendo clic izquierdo sobre él. Si borramos un vértice, también desaparecerán todas las aristas relacionadas con él. Podemos eliminar todos los elementos que estén dentro del rectángulo de selección que se forma cuando movemos el cursor manteniendo pulsado un clic.
4. *Show vertex labels:* especifica que se muestren las etiquetas de los vértices, si está pulsado.
5. *Show vertex weights:* especifica que se muestren los pesos de los vértices, si está pulsado.
6. *Show edge labels:* especifica que se muestren las etiquetas de las aristas, si está pulsado.
7. *Show edge weights:* especifica que se muestren los pesos de las aristas, si está pulsado.
8. *Zoom to fit graph:* ajusta el zoom del panel para permitir que se vea todo el grafo centrado.
9. *Zoom 1:1:* ajusta el zoom del panel para que todos los elementos del grafo se vean con su escala natural.

10. *Zoom in:* aumenta el zoom tomando como punto de referencia el centro del panel. Esta operación se puede realizar si giramos la rueda central del ratón hacia delante cuando el cursor está colocado sobre el plano de edición. En ese caso, la referencia para hacer zoom será el punto donde esté situado el cursor.
11. *Zoom out:* reduce el zoom tomando como punto de referencia el centro del panel. Esta operación se puede realizar si giramos la rueda central del ratón hacia atrás cuando el cursor está colocado sobre el plano de edición. En ese caso, la referencia para hacer zoom será el punto donde esté situado el cursor.
12. *Execute Bipolar Orientation:* realiza la orientación bipolar del grafo. Se deben cumplir las condiciones de que el grafo sea no dirigido y biconexo.
13. *Save a selected path:* agrega el camino seleccionado al conjunto de caminos disjuntos. El camino debe ser disjunto con el resto de caminos ya creados. En caso de será agregado con éxito, el camino deja de estar seleccionado. En otro caso, El camino se mantiene seleccionado y la aplicación emite un sonido de error. También, se debe cumplir que el grafo sea dirigido para poder añadir los caminos.
14. *Delete all saved paths:* vacía el conjunto de caminos creados hasta el momento.
15. *Show all saved paths:* muestra el conjunto de caminos creados hasta el momento. Todos los elementos del grafo que pertenezcan a un camino se mostrarán como si hubieran sido seleccionados.

Otra acción que se puede realizar en el panel es editar los parámetros de las aristas y de los vértices. Esto se consigue haciendo clic derecho sobre un elemento en particular o sobre un grupo seleccionado. En ese momento aparecerá un menú emergente donde se podrá seleccionar entre editar los vértices o las aristas. Una vez situado el cursor sobre uno de los dos elementos, se mostrarán sus parámetros. En el caso de los vértices es posible modificar el valor de la etiqueta (*Label*), el radio (*Radius*) o el peso (*Weight*). En el caso de las aristas es posible modificar el valor de la etiqueta (*Label*), el grosor (*Thickness*) o el peso (*Weight*). En la Ilustración 51 podemos ver un ejemplo de la edición de los parámetros de un vértice.

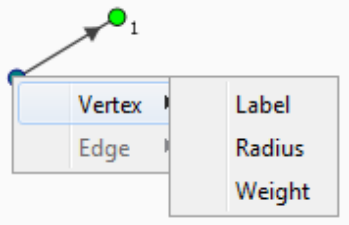

Ilustración 51: Edición de grafos. Editar parámetros

5.3 Ventanas de ejecución de algoritmos

A continuación vamos a ver el funcionamiento de las ventanas de ejecución de los algoritmos. Primeramente, explicaremos el panel de control de la ejecución, Ilustración

52, dado que es un elemento común en la parte superior de todas las ventanas de ejecución y funciona de igual modo siempre.

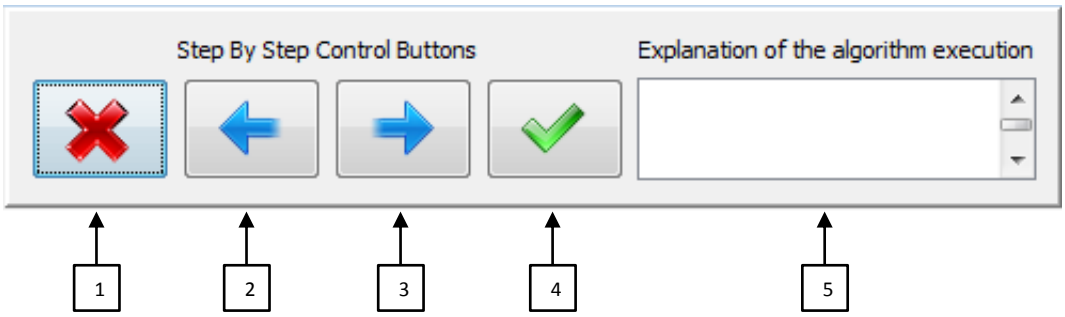

Ilustración 52: Ventanas de ejecución de algoritmos. Panel de control

La descripción de los elementos del panel de control de la ejecución es la siguiente:

1. *Cancelar ejecución:* cancela la ejecución del algoritmo y cierra la ventana.
2. *Mostrar etapa anterior:* retrocede a la etapa anterior a la actual en la ejecución del algoritmo.
3. *Mostrar etapa siguiente:* avanza hasta la siguiente etapa en la ejecución del algoritmo.
4. *Finalizar ejecución:* avanza hasta la etapa final en la ejecución del algoritmo.
5. *Área de explicación del algoritmo:* en esta zona se va exponiendo la descripción teórica de cada paso que se da durante la ejecución.

Ahora, pasaremos a analizar las ventanas de ejecución de cada algoritmo. Empezaremos por el algoritmo visibilidad, cuya ventana podemos observar en la Ilustración 53. La ejecución del algoritmo visibilidad la podemos seguir a través de dos paneles. En uno, *Dual Graph*, se observa la construcción del grafo dual sobre el primal, la numeración topológica del primal y la numeración topológica del dual. En el otro, *Visibility Representation*, se muestra la creación de la representación de visibilidad del grafo. El grafo primal se representa con vértices circulares de color verde y aristas de línea continua de color negro. El grafo dual se dibuja con vértices cuadrados de color rojo y aristas de líneas discontinuas de color rojo. La representación de visibilidad va acompañada de un eje de coordenadas para poder comprobar donde se ubican los elementos.

En la Ilustración 54 se puede ver la ventana de ejecución del algoritmo visibilidad con restricciones. Es muy similar a la ventana del algoritmo visibilidad solo que en este caso el grafo dual se construye de manera diferente al tener que añadir un vértice por cada camino del conjunto de caminos disjuntos. Estos vértices se representan en el dual como un rombo de color rojo. El grafo primal y su dual se muestran en el panel *Constrained Dual Graph*. La representación de visibilidad con restricciones se traza en el panel *Constrained Visibility Representation*.

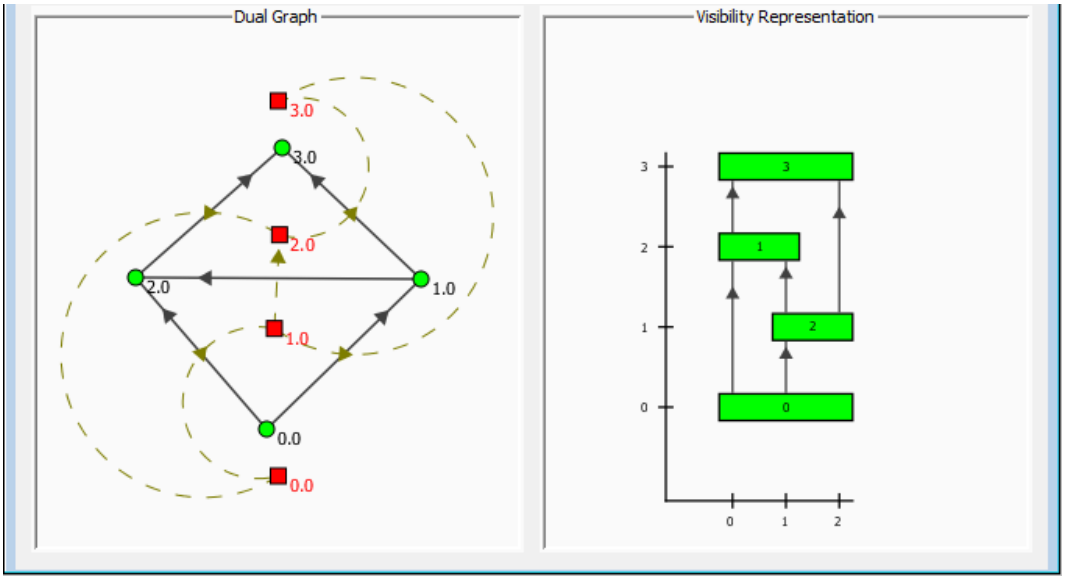

Ilustración 53: Ventanas de ejecución. Algoritmo Visibilidad

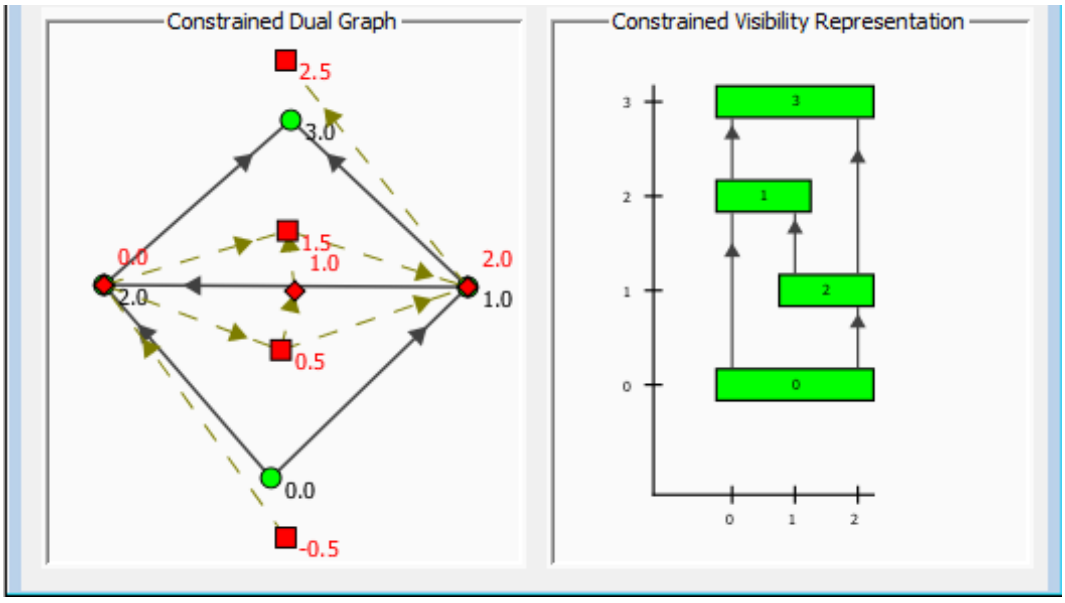

Ilustración 54: Ventanas de ejecución. Algoritmo Visibilidad con restricciones

En la Ilustración 55 podemos observar la ventana de ejecución del algoritmo poligonal. Como vimos en la teoría, este algoritmo parte de la representación de visibilidad, dibujada en el panel *Upward Polyline Drawing* con líneas discontinuas. Sobre esta representación se van situando los elementos del grafo en las coordenadas determinadas por el algoritmo hasta terminar de construir el trazado poligonal.

La Ilustración 56 captura la ventana del algoritmo poligonal con restricciones. Este caso es similar al anterior, pero se diferencia en que toma como punto inicial la

representación de visibilidad con restricciones, igualmente dibujada con líneas discontinuas en el panel *Constrained Upward Polyline Drawing*.

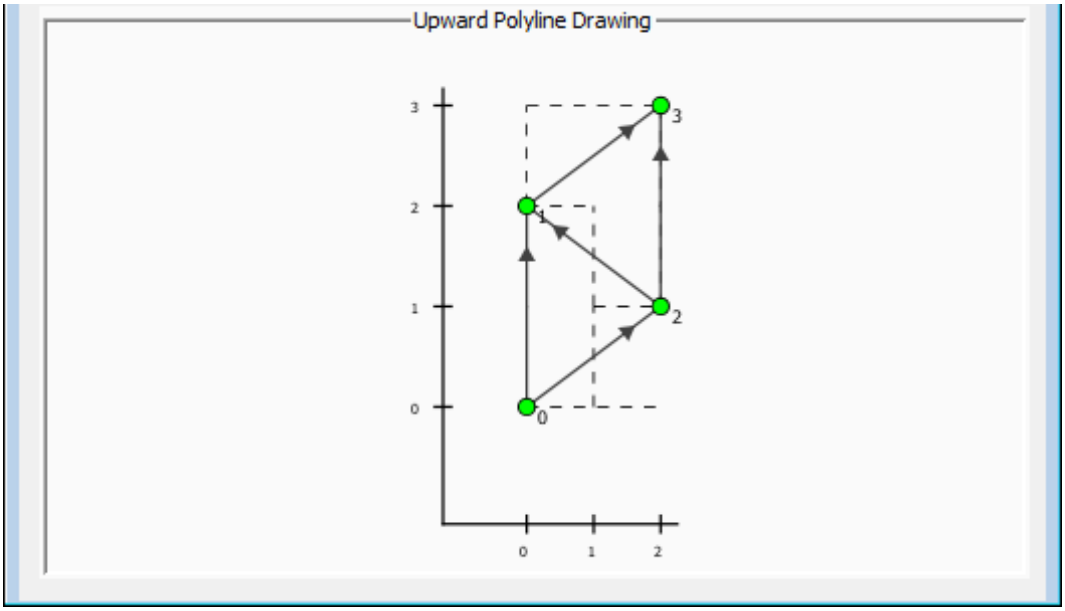

Ilustración 55: Ventanas de ejecución. Algoritmo Poligonal

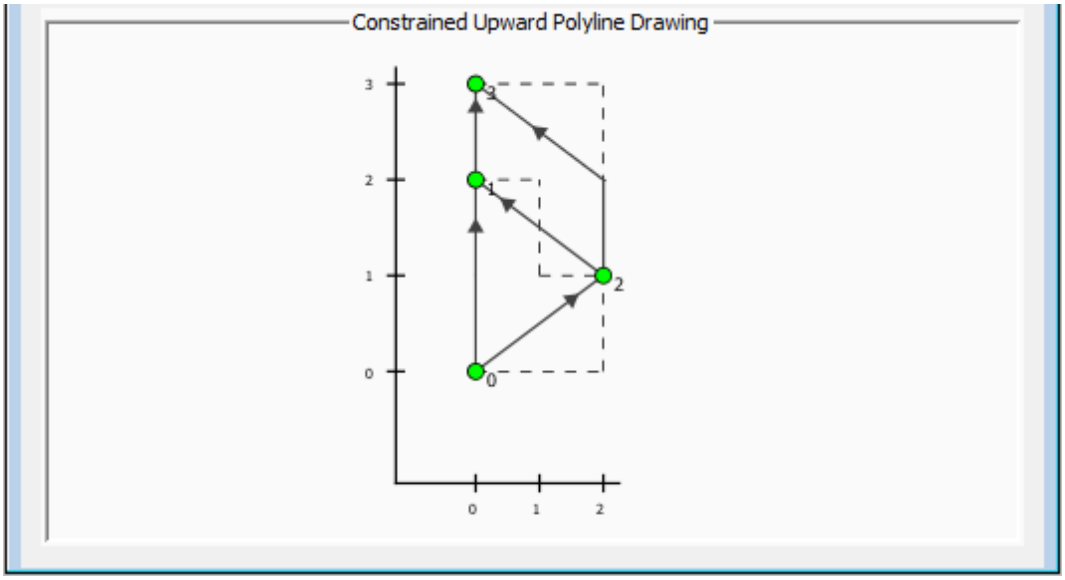

Ilustración 56: Ventanas de ejecución. Algoritmo Poligonal con restricciones

En la Ilustración 57 se muestra la ventana de ejecución del algoritmo ortogonal. En este caso utilizamos dos paneles. Por un lado, tenemos *Directed Graph and Set of Paths*, donde se dibuja el grafo inicial, se orienta bipolarmente en caso de ser no dirigido y se marca con color azul cada camino que va creando el algoritmo. Por otro lado, está *Orthogonal Drawing*, donde se dibuja con líneas discontinuas la

representación de visibilidad con restricciones para, finalmente, ir situando los elementos del grafo y crear el trazado ortogonal.

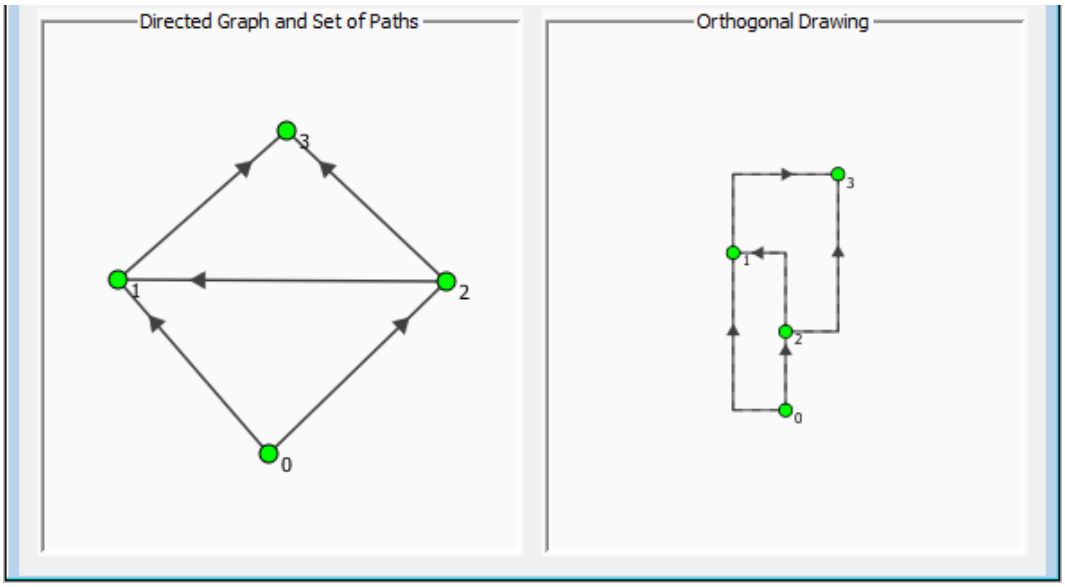

Ilustración 57: Ventanas de ejecución. Algoritmo Ortogonal

En la Ilustración 58 podemos ver la ventana del algoritmo dominancia rectilíneo. En este caso solo es necesario utilizar un panel, *Straight-Line Dominance Drawing*, para comprender la ejecución. Lo primero que se verá es una malla donde se van situando los elementos del grafo en sus coordenadas preliminares. Luego, la ubicación de los elementos va cambiando a medida que las coordenadas se van reduciendo hasta finalizar en el trazado de dominancia rectilíneo final.

La Ilustración 59 muestra la ventana de ejecución del algoritmo dominancia poligonal. Este caso se basa en el algoritmo dominancia rectilíneo que obliga a que el grafo sea reducido. Por esta razón, se cuenta con el panel *Reduced Graph* donde se muestra la eliminación de las aristas transitivas del grafo inicial sustituyéndolas cada una por un vértice intermedio, denominado *v*, y dos aristas para conectar sus vértices a través de *v*. En el panel *Polyline Dominance Drawing* se muestra el trazado de dominancia rectilíneo que resulta del grafo modificado para, luego, ir eliminando los elementos auxiliares y creando aristas con codos en los puntos donde se encontraban los vértices intermedios. Este proceso finaliza obteniendo el trazado de dominancia poligonal.

Por último, resaltar que todos los paneles dentro de las ventanas de ejecución cuentan con la posibilidad de modificar el zoom a través de la rueda central del ratón, tal como se hace en el panel de edición de grafos.

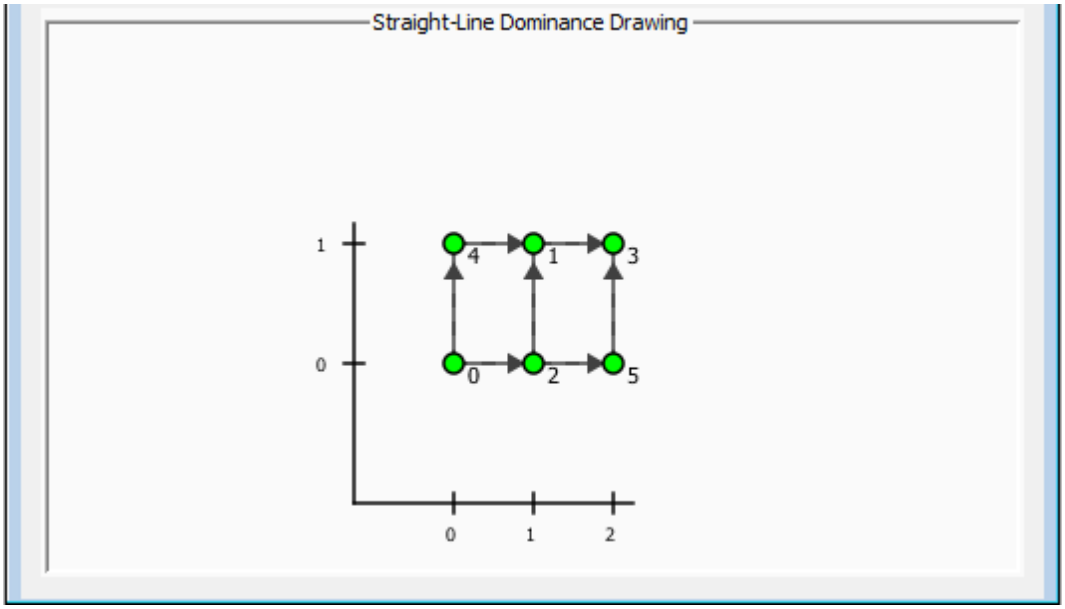

Ilustración 58: Ventanas de ejecución. Algoritmo Dominancia rectilíneo

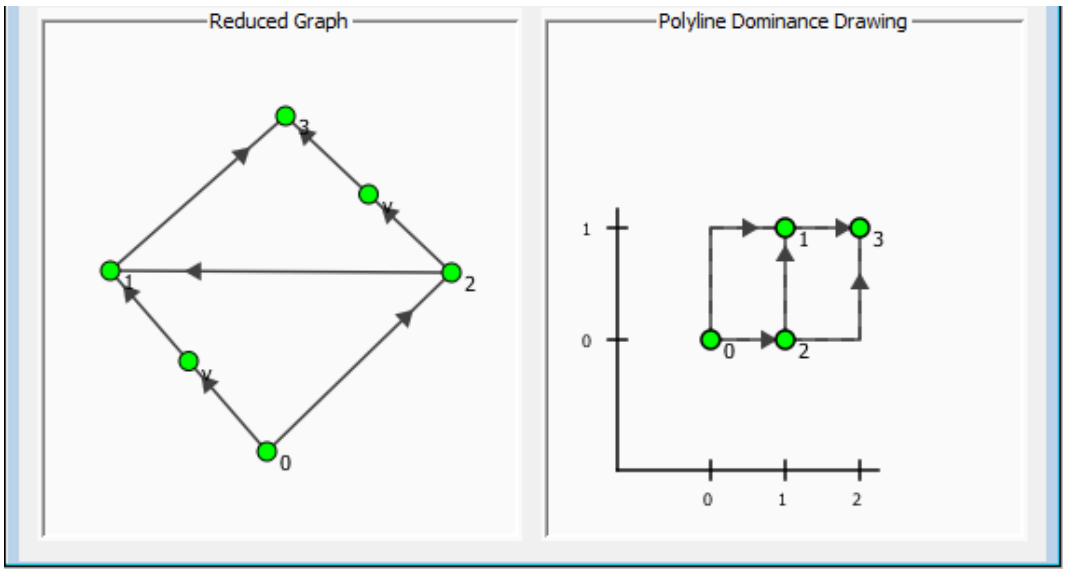

Ilustración 59: Ventanas de ejecución. Algoritmo Dominancia poligonal

6 CONCLUSIONES

Como conclusión de esta memoria del proyecto de final de carrera puedo decir que se han cumplido los objetivos inicialmente planteados y los requisitos que fueron establecidos por el tutor.

Tal como se ha podido leer anteriormente, se diseñó e implementó una aplicación para ejecutar paso a paso algoritmos de Trazado de Grafos. Se dispone de una ventana para editar grafos. También, se tiene una ventana donde se ejecutan todos los pasos de los algoritmos visibilidad, visibilidad con restricciones, poligonal, poligonal con restricciones, ortogonal, dominancia rectilíneo y dominancia poligonal. Se tiene un panel para controlar la ejecución donde se muestra un texto de ayuda explicando cada paso del algoritmo.

Llevar a cabo este proyecto ha supuesto una gran contribución a mi formación como Ingeniero Informático. Desde el punto de vista de la Ingeniería del Software este proyecto me ha servido para profundizar los conocimientos sobre programación orientada a objetos y patrones de diseño. También, me ha permitido conocer en profundidad un campo importante dentro de la Teoría de Grafos como el Trazado de Grafos. Al implementar todo el sistema utilizando el lenguaje de programación Java y en especial la API para desarrollar interfaces gráficas de usuario Swing, he podido adquirir conocimientos más firmes en este campo. Otra experiencia muy valorable ha sido iniciar el diseño de la aplicación a partir de otro proyecto, con todo lo que eso supone respecto a comprender el diseño y el código hecho por otros ingenieros.

Resaltar que hemos generado una amplia documentación de todo el código fuente de la aplicación. Esto ha sido gracias a la utilización de UMLGraph [UML11] mezclado con Javadoc [Jav]. El resultado del uso de ambas herramientas es que documentamos cada clase con la descripción de todos los métodos y atributos, además, insertamos diagramas de clase dentro de la documentación generada por Javadoc.

Como mejora futura de este sistema puede tenerse en cuenta realizar una refactorización del código y mejorar el diseño software para lograr una adaptación al cambio más sencilla. También, pueden incluirse más idiomas dentro de la traducción de la aplicación dado que ya está desarrollada esta funcionalidad. Por otro lado, es un sistema que puede ser usado muy fácilmente para iniciar el desarrollo de otros proyectos relacionados con grafos o añadir más algoritmos para ejecutarlos paso a paso.

BIBLIOGRAFÍA

[Beh11] Cameron Behar. (2010) Visigraph. A flexible cross-platform application for modeling graph theory problems and solutions. [Online]. http://code.google.com/p/visigraph/

[DiB98] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis, *Graph Drawing: Algorithms for the Visualization of Graphs*.: Prentice Hall, 1998.

[Jso11] (2011) Introducción a JSON. [Online]. http://www.json.org/json-es.html

[Jav] Javadoc Tool. [Online]. http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html

[Lar01] Craig Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process*, 2nd ed.: Prentice Hall, 2001.

[Nis04] Takao Nishizeki and Md. Saidur Rahman, *Planar Graph Drawing*.: World Scientific, 2004.

[Ope] (2011) Open Directory - Science: Math: Combinatorics: Software: Graph Drawing. [Online]. http://www.dmoz.org/Science/Math/Combinatorics/Software/Graph_Drawing

[Ora93] Oracle. (2011) Java™ Platform Standard Edition 7. API Specification. [Online]. http://docs.oracle.com/javase/7/docs/api/

[Ora11] Oracle. (2011) The Java Tutorials. [Online]. http://download.oracle.com/javase/tutorial/index.html

[Sed11] Robert Sedgewick and Kevin Wayne, *Algorithms*.: Addison-Wesley Professional, 2011. [Online]. http://algs4.cs.princeton.edu/home/

[UML11] Diomidis Spinellis and Andrea Aime. (2011) UMLGraph. Automated Drawing of UML Diagrams. [Online]. http://www.umlgraph.org/

[Tar76] Robert Endre Tarjan, *Edge-disjoint spanning trees and depth-first search*.:

Springer-Verlag, 1976.

[Tar85] Robert Endre Tarjan, *Two Streamlined Depth-first Search Algorithms*.: Princeton University, Department of Computer Science, 1985.