INFORME
TRABAJO PRÁCTICO 3 - PROGRAMACIÓN III
CONJUNTO DOMINANTE MÍNIMO

Integrantes del grupo:

Abel Aquino
Lautaro Moreno
Karin Pellegrini

Comisión : 01

Profesores:

Patricia Bagnes
Ignacio Sotelo
Gabriel Carrillo

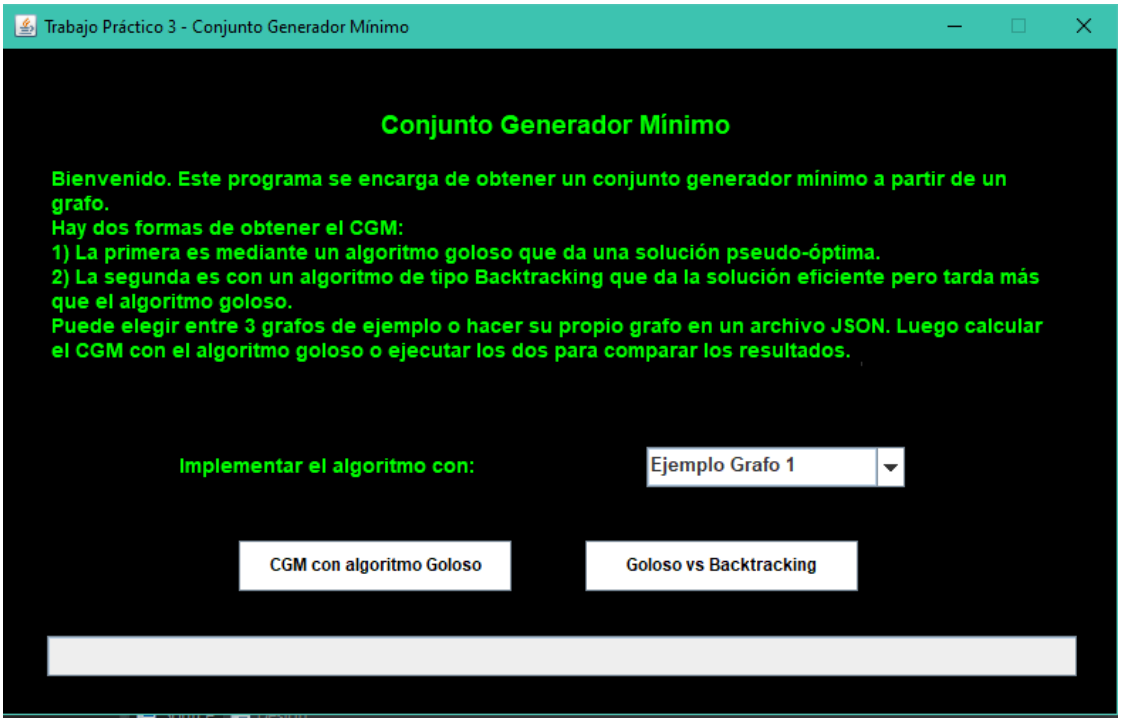
Introducción

En este informe se detalla el programa realizado por los alumnos, presentes en la carátula, que se encarga de calcular un conjunto generador mínimo (apodado por nosotros como CGM) compuesto de vértices, a partir de un grafo que es pasado mediante un archivo JSON. Para obtener el CGM implementamos dos algoritmos, un algoritmo heurístico llamado Goloso y otro algoritmo de tipo Backtracking. La diferencia entre estos dos algoritmos se da mediante el orden de complejidad que tienen (el algoritmo Goloso es de orden polinomial, por lo cual es más rápido que el algoritmo de tipo Backtracking, el cual tiene un orden exponencial) y la solución que traen (Óptima en el caso del Backtracking y pseudo-óptima en el caso del algoritmo Goloso).

La aplicación cuenta está dividida siguiendo la arquitectura Model-View-Presenter. También cuenta con dos interfaces, diseñadas con la librería WindowBuilder. La primera sirve para decidir el método que se usará para calcular el CGM y elegir entre grafos cargados o hacer su propio grafo, mientras que la segunda interfaz me permite visualizar el grafo con el que se trabaja y el CGM de ese mismo grafo.

A continuación, vamos a mostrar las clases de las interfaces, sus casos de uso divididos en secciones, y luego mostraremos las clases implementadas para el funcionamiento de la aplicación.

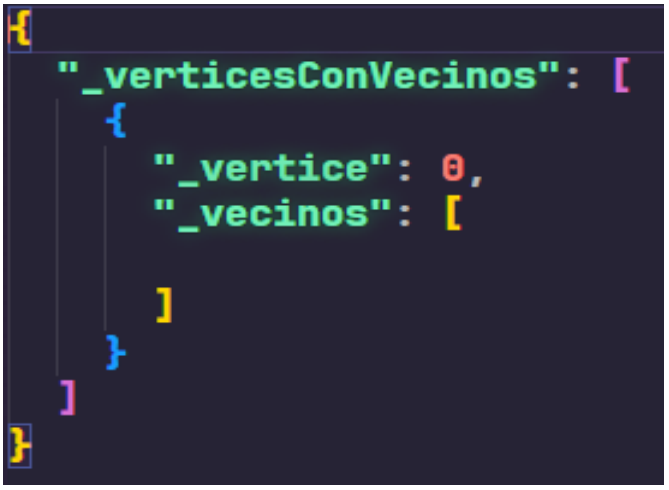
Interfaz presentación



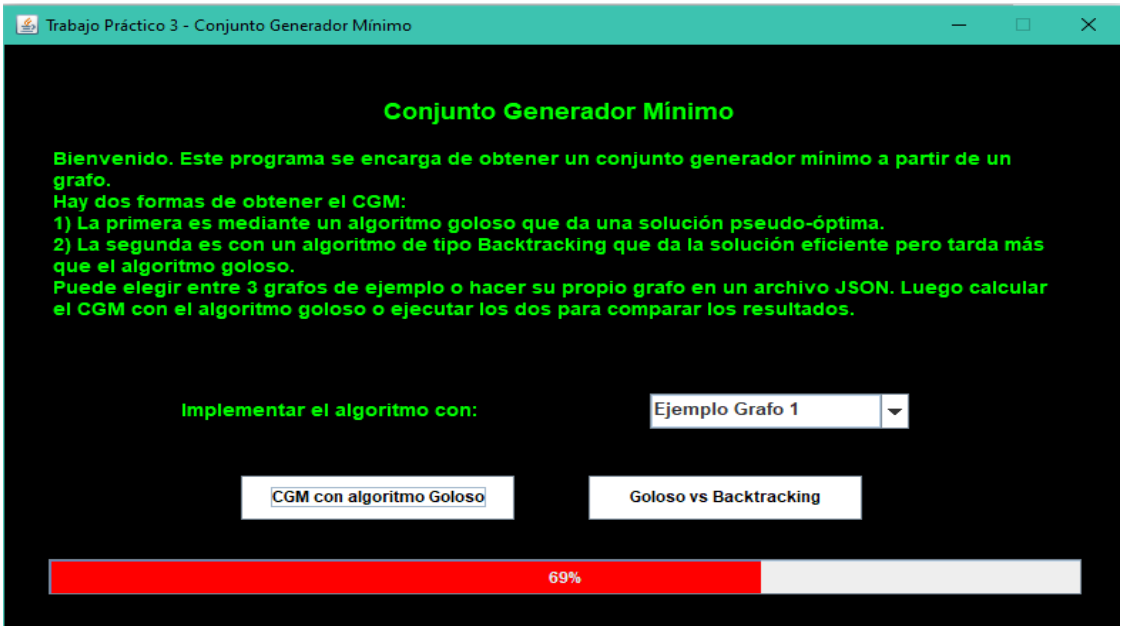
Dentro de esta interfaz se realiza la gestión del grafo que se desea usar para calcular el conjunto generador mínimo, elegir el método que se desea usar y luego visualizarlo en otra pantalla. Hay tres posibles grafos que ya vienen cargados en el programa para que el usuario vea, a modo de ejemplo, cómo funciona el algoritmo

con grafos de menor a mayor cantidad de vértices. El ejemplo 1 contiene un grafo de 6 vértices, el ejemplo 2 tiene un grafo de 10 vértices y el grafo del ejemplo 3 tiene 17 vértices.

Además, se cuenta con un grafo que el usuario puede personalizar, agregando la cantidad de vértices que desee junto con los vecinos de cada vértice. Siempre y cuando, se respete la siguiente estructura :



Siguiendo con lo anterior, el usuario tendrá a su disposición dos formas de obtener la solución deseada (o una cercana). Si se utiliza el botón “CGM con algoritmo Goloso” la barra que está debajo de los botones empezará a llenarse, cambiando su porcentaje del 1% al 100% mientras el conjunto generador mínimo es calculado.



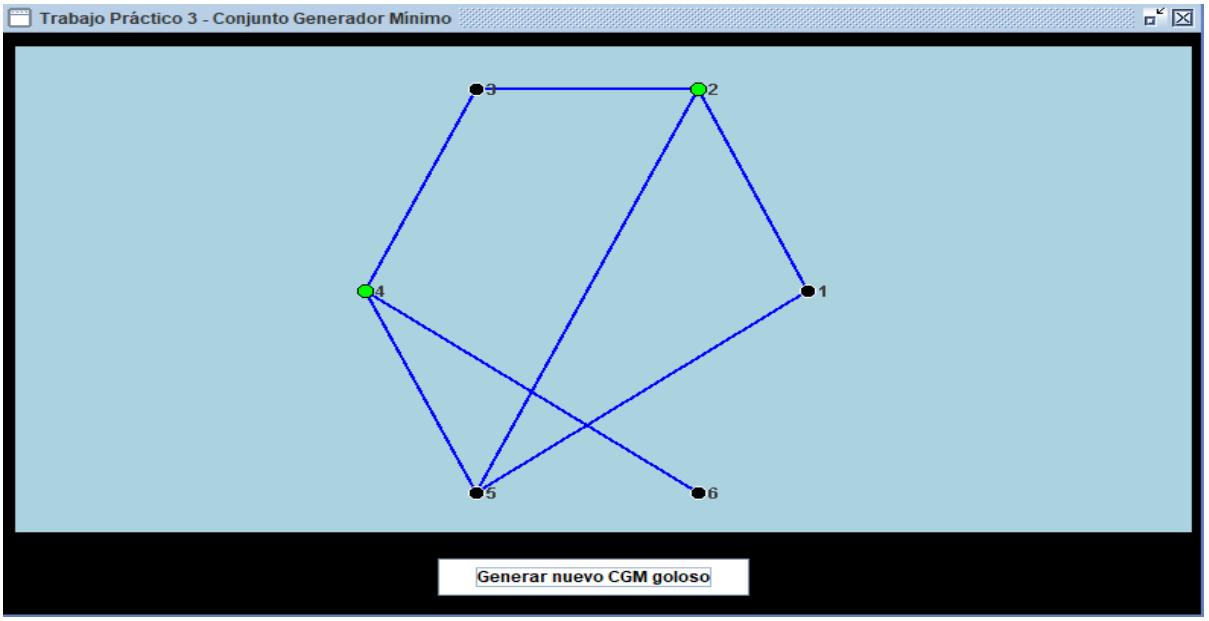
En caso de utilizar el botón “Goloso vs Backtracking” la barra no se llenará como en el ejemplo anterior, sino que irá desplazando una pequeña barra a lo largo de la barra principal.

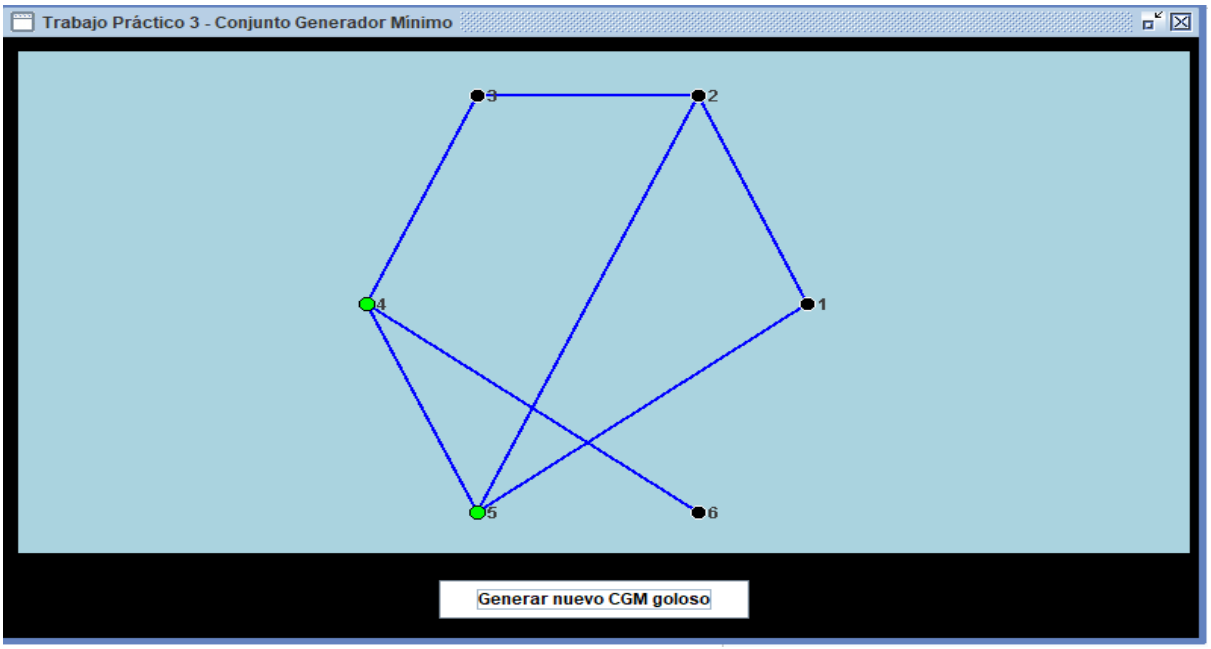
En este caso, el usuario podrá visualizar dentro de la pantalla dos grafos (siendo el mismo grafo seleccionado) pero con un conjunto generador mínimo calculado mediante el algoritmo goloso y otro con el algoritmo backtracking.

Interfaz Pantalla Cargar CGM Goloso

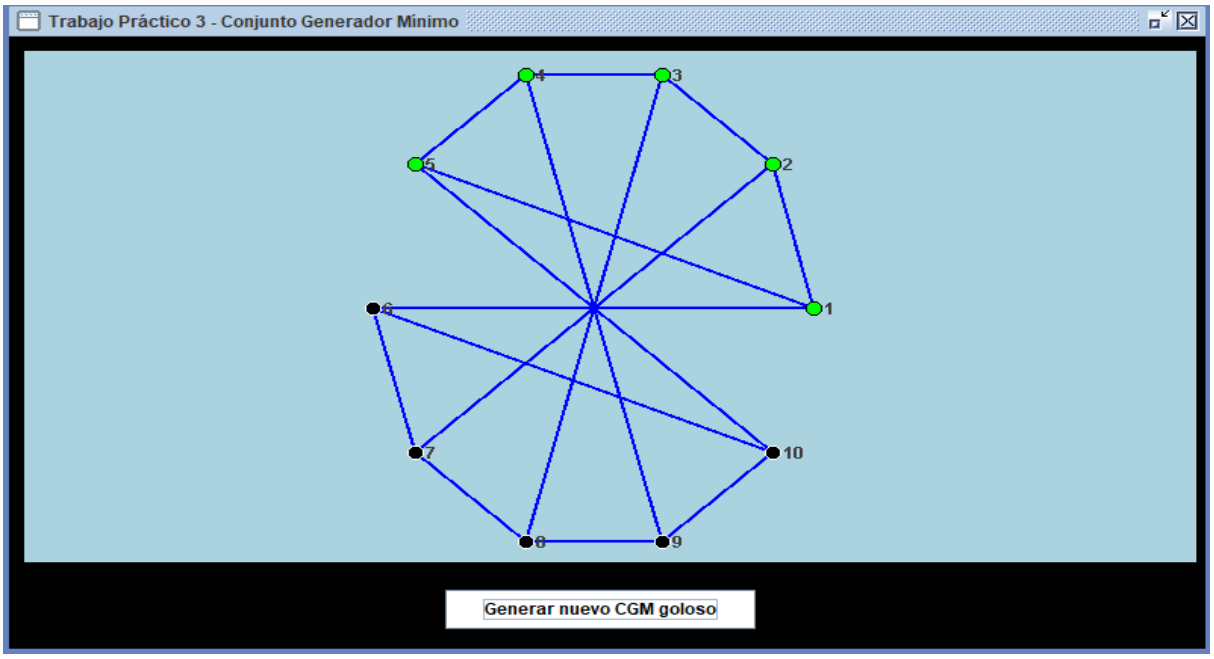
Una vez cargado el grafo, si el usuario selecciona el primer botón (CGM con algoritmo Goloso) se abrirá esta interfaz donde se visualiza el grafo junto con su conjunto generador mínimo. Para distinguirlos, los vértices del grafo que no estén dentro del conjunto generador mínimo estarán pintados de color negro, mientras que los vértices pertenecientes al conjunto estarán pintados de color verde.
A continuación, presentamos una muestra de cada grafo en el programa cargado en la pantalla, cabe resaltar que el conjunto obtenido por este algoritmo puede variar y no siempre será el mismo:

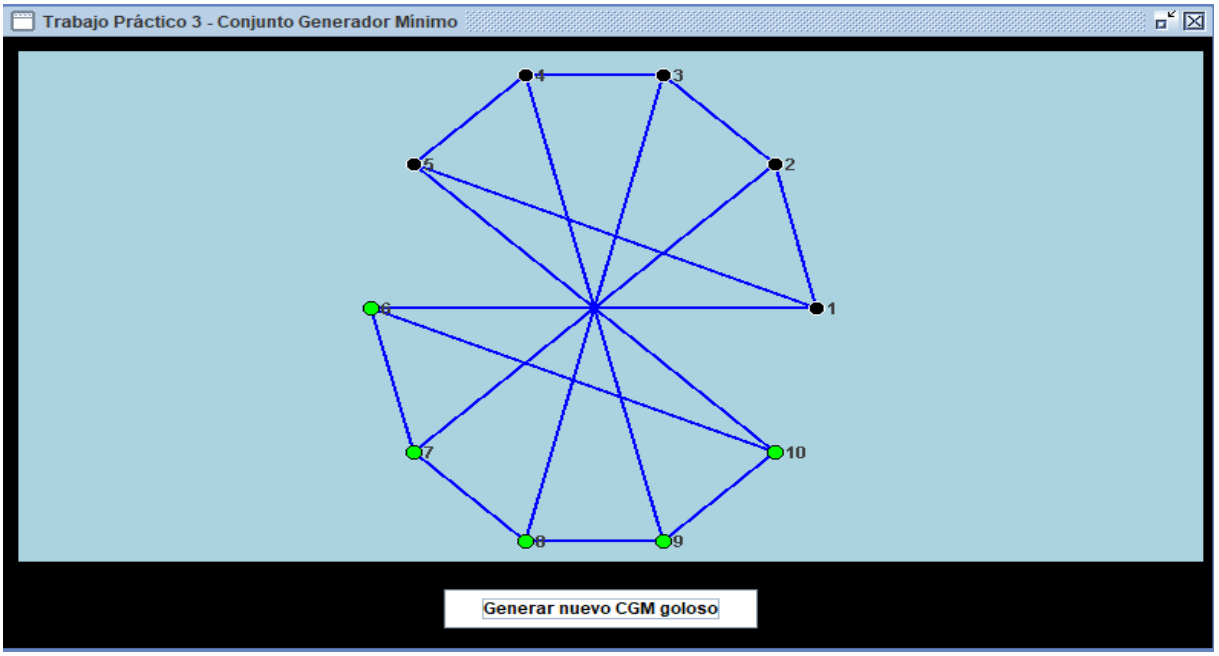
Ejemplo de grafo 1:



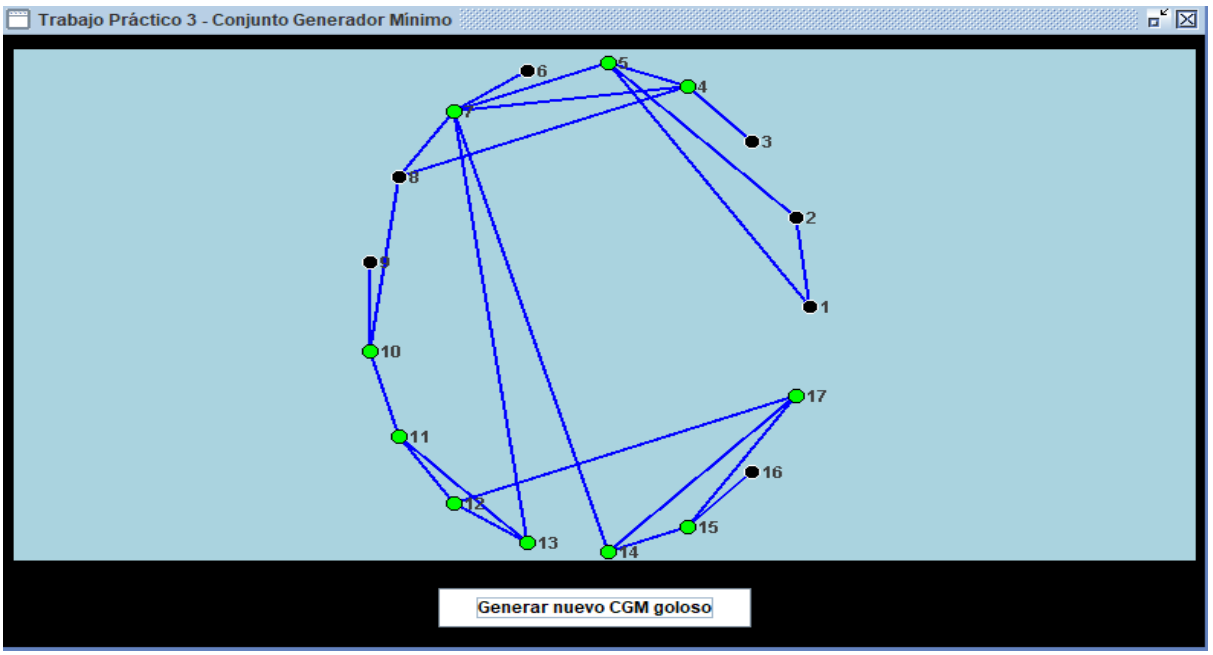


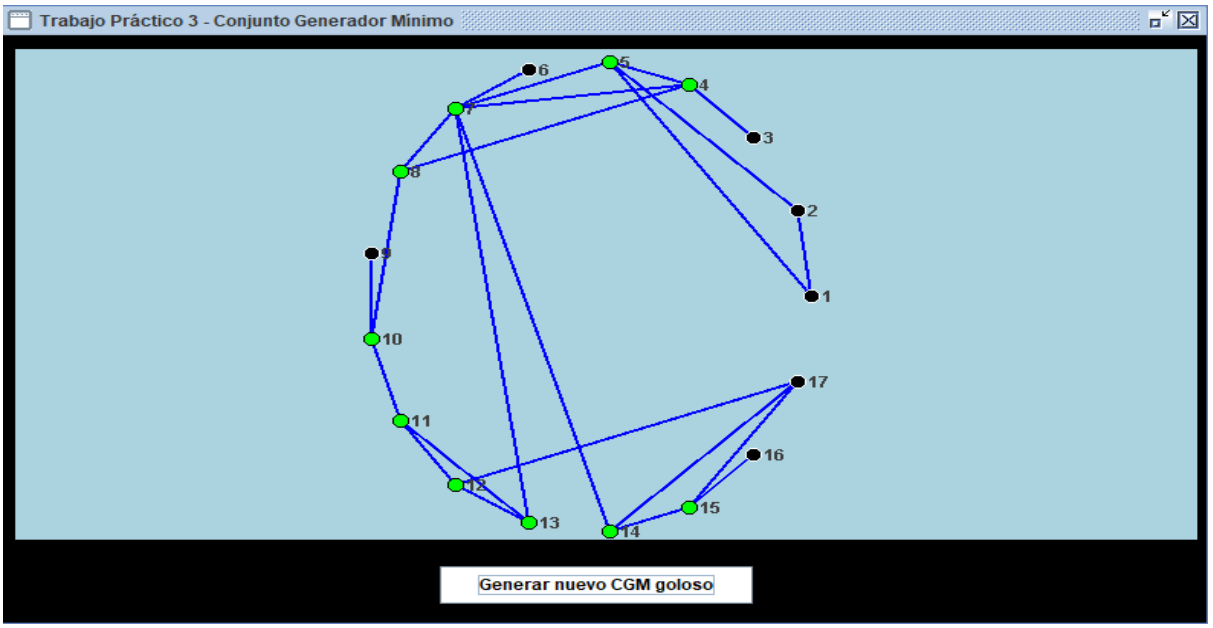
Ejemplo de grafo 2:





Ejemplo de grafo 3:



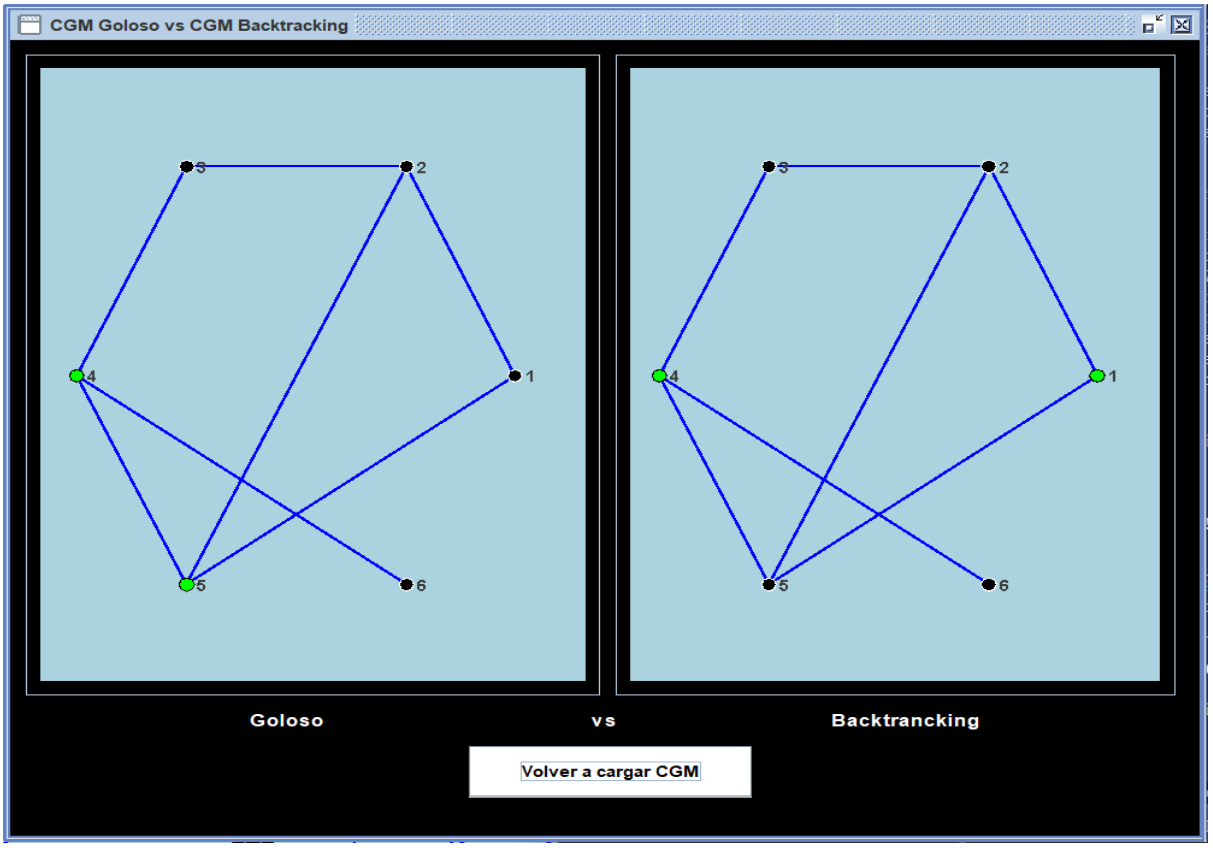


Interfaz Pantalla Cargar CGM Goloso vs CGM Backtracking

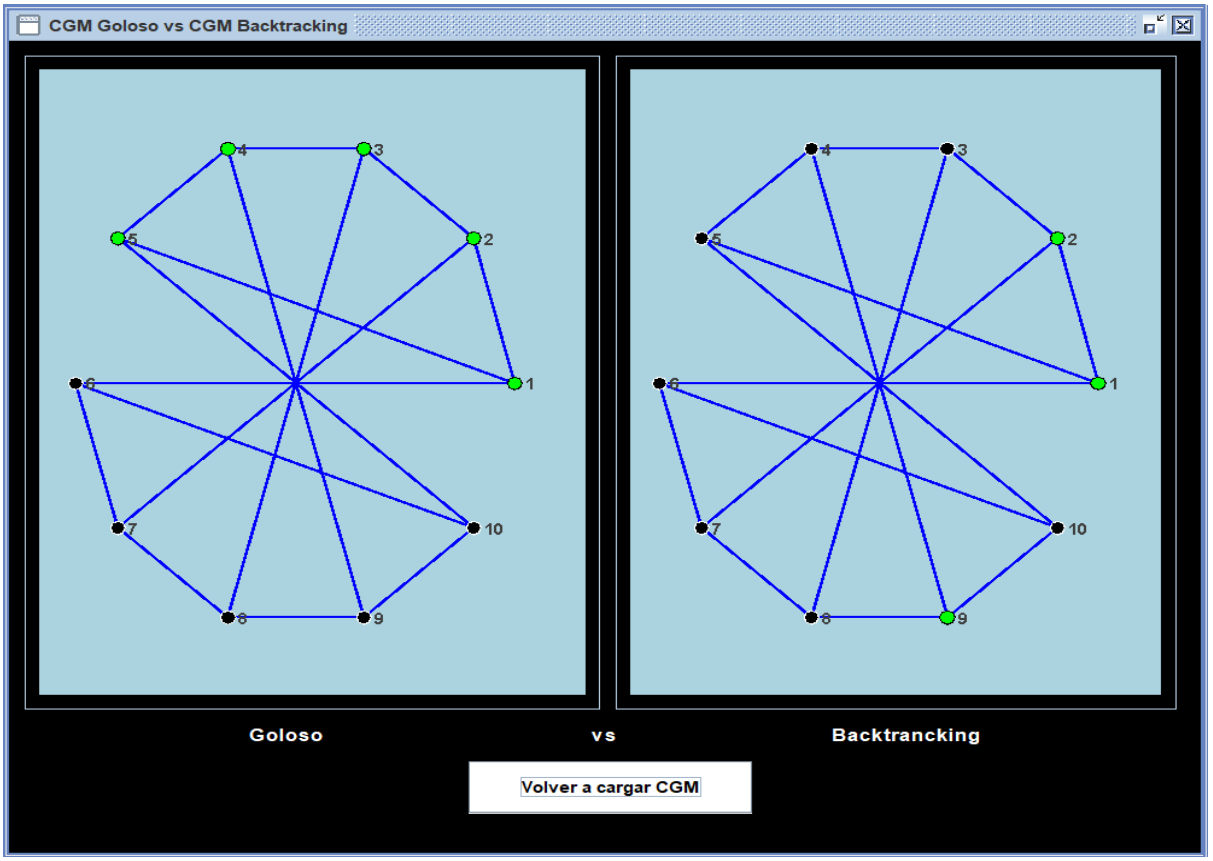
En esta interfaz se muestran dos planos. Uno nos grafica el algoritmo goloso y en el otro podemos ver el resultado de hacer backtracking. Esto fue realizado con el objetivo de poder visualizar y comparar ambas soluciones de una forma cómoda para el usuario.

También contamos con un botón que nos lleva a la pantalla principal para que podamos generar un nuevo CGM goloso.

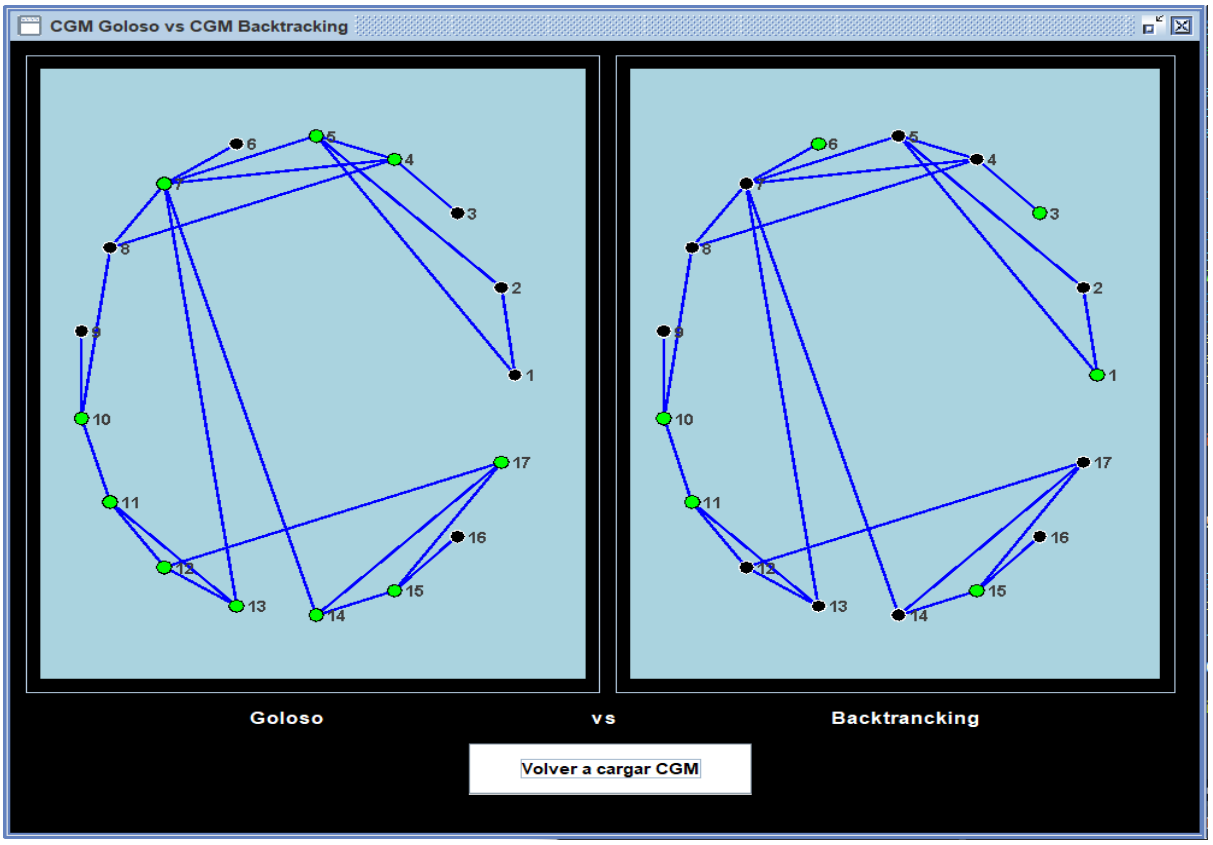
Resultado utilizando el Grafo 1:



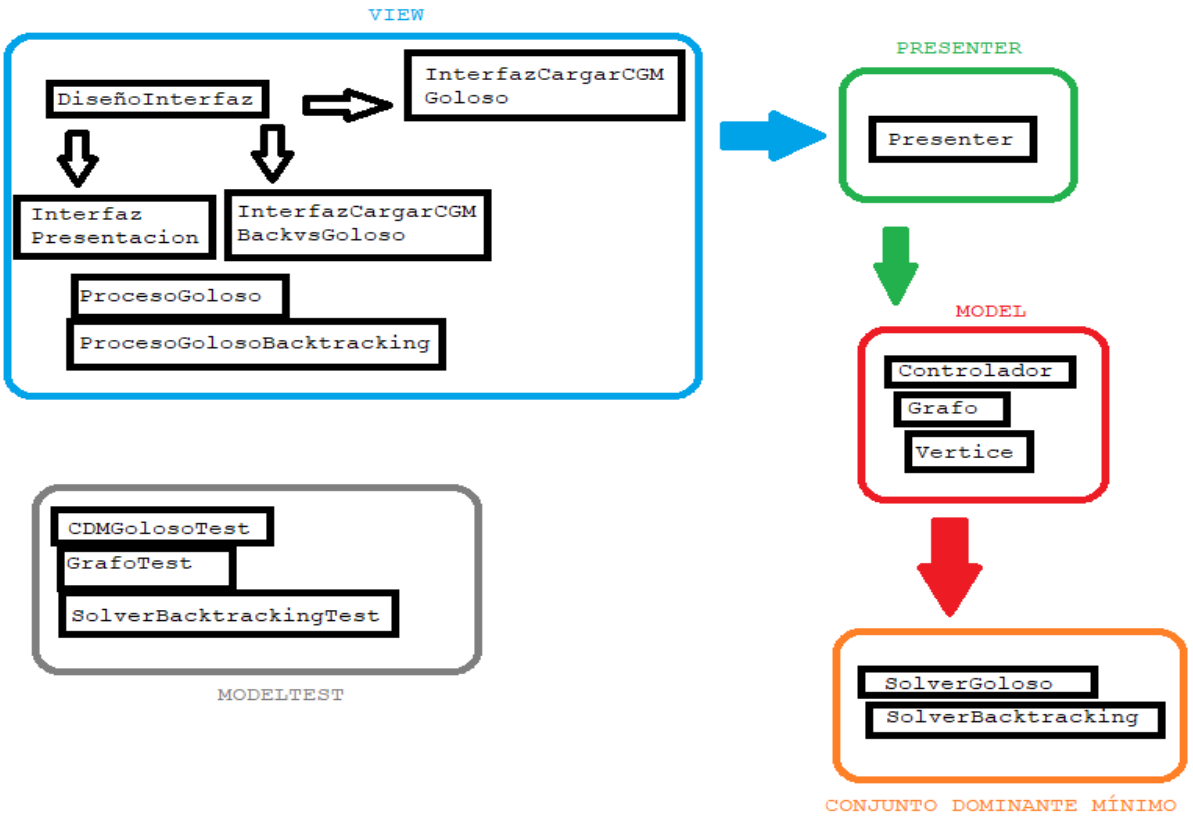
Resultado utilizando el Grafo 2:



Resultado utilizando el Grafo 3:



Arquitectura MVP:



Paquete Modelo:

En este paquete contenemos las clases de negocio, como la clase *Grafo*, *Vértice* y *Controlador*.

Grafo:

Esta clase es donde se crean los grafos que se van a utilizar para calcular su conjunto dominante mínimo.
Su representación está ligada solamente a vértices y sus vecinos en lugar de una matriz de adyacencia.
Puede cargar grafos en formato JSON y usarlos en el proyecto.

Vértice:

La clase organiza y controla las aristas del grafo, pues guarda su valor de vértice junto con los vértices que estén conectados a este.
Con esto se busca poder mantener un código legible en la clase Grafo, ya que desligamos responsabilidades para entregarlas a esta clase.

Controlador:

El rol de “Controlador” es llamar y gestionar a los componentes, permitiendo seleccionar el grafo con el cual se va a trabajar.
Contiene el conjunto dominante mínimo del grafo y una pseudo solución la cual está obtenida con algoritmo goloso. Es capaz de leer un grafo personalizado cargado mediante JSON o trabajar con los que están predeterminadamente.

Algoritmo Goloso versus Backtracking:

Como se mencionó en la introducción, para obtener el conjunto dominante mínimo de un grafo se implementó dos algoritmos, uno se encapsula en la clase SolverGoloso y el otro en SolverBacktracking.

Algoritmo goloso:

Este algoritmo se centra en la búsqueda de los vértices con más vecinos, se debe obtener un conjunto de vértices los cuales sus vecinos deben ser todos los vértices del grafo que no estén incluidos en el conjunto.
Esta búsqueda tiene una complejidad polinomial, por lo cual es un tiempo eficiente, sin embargo, el conjunto que entrega no siempre es el mínimo, más adelante lo compararemos con el del algoritmo Backtracking.

Algoritmo Backtracking:

Por otro lado, el algoritmo Backtracking, de forma recursiva, realiza una búsqueda exhaustiva de todas las combinaciones posibles entre los vértices y consiguiendo todos los conjuntos dominantes.
A medida que explora las combinaciones, siempre se queda con el conjunto dominante con menos elementos hasta el momento.

Mientras explora un conjunto y este está conteniendo más elementos que el mínimo parcial se cortara y explorara la próxima combinación, ya que esta no va a ser una mejor solución de la que tenemos.

Al contrario que el algoritmo goloso, se obtiene una solución óptima, pero en consecuencia, la complejidad deja de ser polinomial para ser exponencial, por ende, es más lento para entregar la solución.

Comparar resultados:

Grafo	Conjunto que devuelve Goloso	Backtracking
grafo 1	[2,4]	[1,4]
grafo 2	[1,2,3,4,5]	[1,2,9]
grafo 3	[7,5,4,17,15,14,13,12,11,10]	[1,3,6,10,11,15]

Paquete view

Este paquete contiene las clases que crean las interfaces que el usuario ve y también encontramos dos clases que implementan la clase “SwingWorker” para crear los CGM en segundo plano sin bloquear la interfaz de usuario principal. A continuación explicaremos brevemente cada una:

DiseñoInterfaz: Es una clase abstracta que tiene dos herederos: PantallaCargarCGMBackvsGoloso y PantallaCargarCGMGoloso. Tiene 5 métodos:

1) obtenerVecinos: devuelve los vecinos de cada vértice del grafo
2) dibujarAristasEnPlano: por cada vértice crea una ruta, la cual parte del vértice en el que estamos parados y finaliza en el vecino. Para poder realizar esto se trabaja con un HashMap que tiene guardado las coordenadas de cada vértice.
3) obtenerCoordenadaNodoActual: su objetivo es recorrer el HashMap de coordenadas para buscar la ubicación del vértice en el mapa.
4) asignarCaracteristicas: le da estilos a los botones de las interfaces.
5) crearNuevoPuntoEnElPlano: agrega un marcador en el plano y guarda sus coordenadas.

PantallaCargarCGMBackvsGoloso: se ejecuta luego de presionar el botón “Goloso vs Backtracking”. Cuenta con dos planos para poder mostrar el resultado del algoritmo goloso por un lado y el backtracking por el otro. Utiliza los métodos crearNuevoPuntoEnElPlano y dibujarAristasEnPlano los cuales son heredados.

PantallaCargarCGMGoloso: se ejecuta luego de presionar el botón “CGM con algoritmo goloso”. Dibuja el algoritmo goloso en un plano utilizando los métodos heredados.

InterfazPresentacion: en esta clase se encuentra el main que ejecuta el programa. En esta se crea la primera pantalla en la que podemos seleccionar el grafo que queremos generar.

Clases de la view que utilizan SwingWorker:

ProcesoGoloso y **ProcesoBacktracking**
Encargadas de administrar las barras de carga y calcular los CGM.

Conclusión

Para concluir con este informe, podemos decir que el proyecto se pudo desarrollar cumpliendo con todos los objetivos pedidos en las consignas de este trabajo. A lo largo de las cuatro semanas el grupo fue dando ideas, algunas implementadas y otras descartadas. Entendemos que se logró resolver los desafíos como:

- La correcta implementación de un algoritmo goloso que genera un conjunto dominante distinto cada vez que se lo ejecuta.
- La correcta implementación de un algoritmo que implementa backtracking y encuentra el conjunto dominante mínimo más óptimo en cada grafo que le pasamos.
- Leer los grafos desde archivos JSON
- Graficar los grafos en planos y colorear el CGM
- Realizar test unitarios para evaluar el comportamiento de las unidades de la aplicación.
- Permitir que se agreguen nuevas personas al grafo y recalcular los grupos, en este caso se debe modificar el archivo JSON de cada grafo o generar uno nuevo con el JSON vacío que se provee.
- Respetar la correcta implementación de la arquitectura.
- Implementamos lo visto en las lecturas y charlado en clases, más concretamente las reglas de Clean Code. No importamos librerías sin usar ni tampoco dejamos impresiones en consola, y usamos nombres declarativos para las variables y funciones.
- Se entrega un conjunto dominante distinto al ejecutar más de una vez el algoritmo goloso.

Así como tuvimos problemas que logramos solucionar, también surgieron contratiempos que nos impidieron implementar más funcionalidades. Dentro de estos pendientes encontramos, por mencionar algunos:

- Nos hubiera gustado darle una interfaz al usuario para que pueda cargar los puntos desde allí y no tenga que estar modificando el JSON.
- Cuando se ingresa un archivo JSON incorrecto y se detiene el programa, mostrar el motivo en la interfaz que el usuario visualiza.