GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

MAXIMIZACIÓN CON TÉCNICAS METAHEURÍSTICAS DE LA REGIÓN DE VISIBILIDAD DE UN PUNTO EN UN POLÍGONO

Autor: Daniel Pérez Bienzobas

Director: Santiago Canales Cano

Madrid

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Maximización con técnicas metaheurísticas de la región de visibilidad de un punto en un polígono

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2022/23 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

Fdo.: Daniel Pérez Bienzobas Fecha: 26/ 06/ 2023

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Santiago Canales Cano Fecha: 26/ 06/ 2023

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

MAXIMIZACIÓN CON TÉCNICAS METAHEURÍSTICAS DE LA REGIÓN DE VISIBILIDAD DE UN PUNTO EN UN POLÍGONO

Autor: Daniel Pérez Bienzobas

Director: Santiago Canales Cano

Madrid

Agradecimientos

Quiero agradecer a mi familia por su apoyo incondicional durante todo el periodo universitario, por haberme inculcado sus valores y por enseñarme a valorar el esfuerzo, a mis amigos por ser siempre un respaldo y a mis profesores por haber fomentado mis ganas de aprender y desarrollarme como persona y profesional, especialmente a Santiago Canales Cano que sin su ayuda no hubiera sido posible sacar este trabajo de fin de grado adelante.

MAXIMIZACIÓN CON TÉCNICAS METAHEURÍSTICAS DE LA REGIÓN DE VISIBILIDAD DE UN PUNTO EN UN POLÍGONO

Autor: Pérez Bienzobas, Daniel.
Director: Canales Cano, Santiago.
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

El proyecto consiste en la creación de un software gráfico para la maximización de la región de visibilidad de un punto interior a un polígono *P* de *n* vértices, mediante la aplicación de tres metaheurísticas: Random Search, Simulated Annealing y Algoritmo Genético.

Palabras clave: Técnicas Heurísticas, Metaheurísticas, Optimización, Visibilidad de Polígonos, Geometría Computacional, Random Search, Simulated Annealing, Algoritmo Genético.

1. Introducción

La Geometría Computacional se enfoca en el estudio de algoritmos y técnicas matemáticas para resolver problemas geométricos usando sistemas computacionales. La solución de estos problemas requiere conocimiento de geometría y habilidades algorítmicas.

La resolución de problemas de visibilidad es una aplicación importante de Geometría Computacional en campos como Robótica, Computación Gráfica, Arquitectura y Urbanismo. El Teorema de la Galería de Arte, que resuelve el problema de iluminar completamente un polígono, fue propuesto por Chvátal. Posteriormente, Lee y Lin demostraron que la optimización de esta solución es un problema complejo que requiere métodos metaheurísticos para su resolución eficiente.

2. Definición del proyecto

El objetivo principal de este proyecto es generar una aplicación que solucione problemas de visibilidad en una herramienta práctica y visual que permita comprender los métodos heurísticos y metaheurísticos en el campo de la Geometría Computacional. La aplicación se ha desarrollado en Python, un lenguaje de programación de código abierto que permite seguir mejorando el programa. El problema principal que se aborda consiste en minimizar el número de luces interiores a un polígono que lo iluminan por completo y se pretende aportar distintos métodos para la obtención del punto de máxima iluminación dentro de un polígono. Esta aplicación sirve como punto de partida para resolver el problema de mayor complejidad que pretende encontrar *k* puntos de máxima iluminación interiores a un polígono.

Para alcanzar una solución al problema principal, se han empleado tres algoritmos heurísticos: Random Search (RS), Simulated Annealing (SA) y el Algoritmo Genético (GA). El objetivo del proyecto es estudiar exhaustivamente el problema MaxA-p-Pv1(*P*) que consiste en encontrar el punto de máxima iluminación interior a un

polígono simple “sin agujeros” *P* de *n* vértices. Este problema se corresponde con una particularización del problema general MaxA-p-Pvk(*P*,*k*), donde *k* representa el número de luces punto localizadas para encontrar la máxima visibilidad. Ambos problemas mencionados se corresponden con el paso previo para dar una solución al problema que se identifica como MinN-p-Pvk(*P*) que pretende minimizar el número de luces que hacen que se ilumine por completo un polígono *P*.

El problema MaxA-p-Pv1(*P*), que es el problema del que nos ocupamos en el presente proyecto, se puede detallar de la siguiente forma:

Pregunta: ¿Dónde se debe localizar una *luz punto p* en el interior de un polígono simple “sin agujeros” *P* de *n* vértices para que el área de visibilidad desde dicho punto *p* sea máxima?
Entrada: Un polígono simple “sin agujeros” *P* de *n* vértices.
Salida: El punto *p* interior a *P* cuyo polígono de visibilidad es de área máxima.

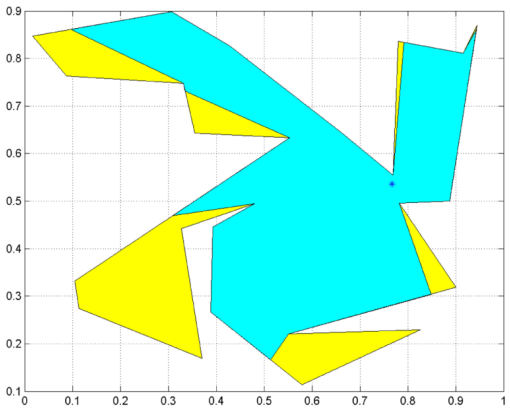

Figura 1: Polígono de visibilidad dado un punto interior a un polígono P de 24 vértices

3. Descripción de la arquitectura de la aplicación

El software del proyecto se divide en tres archivos que serán explicados a continuación. *TEHEMAXRVP.py* es responsable de construir la ventana principal y sus componentes, así como de proporcionar funciones auxiliares para los algoritmos empleados en la aplicación. *Funciones.py* contiene dos funciones que son importantes para el funcionamiento de la aplicación: una función para calcular polígonos de visibilidad y otra para convertir coordenadas de pantalla a coordenadas cartesianas. *Appdesign.py* es responsable de configurar la ventana principal y sus componentes, y se ha creado con la ayuda de la aplicación Qt Designer.

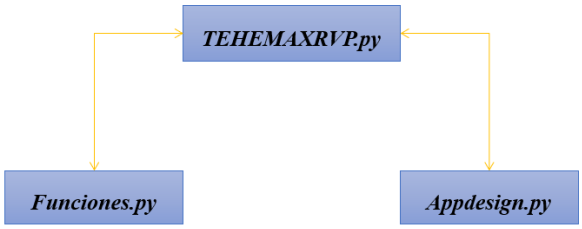

Figura 2: Arquitectura de la aplicación gráfica

4. Resultados

En este proyecto se ha realizado un estudio comparativo de tres técnicas metaheurísticas para maximizar el polígono de visibilidad de un punto interior a un polígono. El estudio se ha centrado en el análisis de los tiempos de procesamiento y el porcentaje de visibilidad obtenido por cada técnica. Se han analizado 25 polígonos seleccionados aleatoriamente con 15, 25 y 35 vértices cada uno y se han utilizado los algoritmos Random Search, Simulated Annealing y Algoritmo Genético. Los resultados medios para los 25 polígonos de cada tamaño de vértices se muestran en las Tablas 1, 2, 3 y 4, para cada uno de los algoritmos respectivamente.

Se concluye que el algoritmo que mejores resultados proporciona para la maximización del área de visibilidad respecto a un punto interior a un polígono P de n vértices es Simulated Annealing, pero el algoritmo que es más eficiente teniendo en cuenta el tiempo de procesamiento es Random Search. El Algoritmo Genético proporciona resultados favorables, muy cercanos a la técnica SA en algunos casos, pero su tiempo de procesamiento parece demasiado elevado y aumenta en mayor proporción con el aumento de la dificultad del problema.

Además, se ha realizado un análisis de los tiempos de procesamiento medios de las diferentes técnicas empleadas para los distintos números de vértices de los polígonos estudiados. Aunque no se pueden sacar conclusiones extrapolables que relacionen el tiempo de procesamiento medio de los algoritmos y el número de vértices que tienen los polígonos analizados, se puede observar que el algoritmo RS presenta un crecimiento más lineal en relación con el número de vértices de los polígonos, mientras que el algoritmo GA presenta el mayor aumento en el tiempo de procesamiento en relación con el aumento del número de vértices, seguido del algoritmo SA y del RS.

En resumen, el estudio ha demostrado la eficacia de las técnicas metaheurísticas para maximizar el polígono de visibilidad y ha proporcionado información valiosa sobre los tiempos de procesamiento y el porcentaje de visibilidad obtenidos por cada técnica.

Vértices	*% Área de Visibilidad*	*Tiempo de Procesamiento [s]*
15	93,71	0,26
25	84,26	0,34
35	88,11	0,51

Tabla 1. Resultados de Random Search con 250 puntos

Vértices	*% Área de Visibilidad*	*Tiempo de Procesamiento [s]*
15	93,78	0,46
25	84,42	0,66
35	88,24	0,99

Tabla 2. Resultados de Random Search con 500 puntos

Vértices	*% Área de Visibilidad*	*Tiempo de Procesamiento [s]*
15	93,92	27,70
25	84,66	82,18
35	88,36	127,16

Tabla 3. Resultados de Simulated Annealing $T_0 = Vértices$ $T(k) = \frac{T_0}{1+k}$ $T_f = 0.03$ $N(T) = \frac{1}{T}$

Vértices	*% Área de Visibilidad*	*Tiempo de Procesamiento [s]*
15	93,61	47,18
25	84,55	203,90
35	88,31	478,06

Tabla 4. Resultados del Algoritmo Genético $p_m = 0.5$

5. Conclusiones

El objetivo principal de este proyecto es el desarrollo de un software interactivo para resolver el problema MaxA-p-Pv1(*P*) en el que se busca maximizar el polígono de visibilidad en un polígono *P* con *n* vértices a partir de un punto interior. El software permite establecer un problema poligonal y aplicar tres técnicas metaheurísticas, Random Search, Simulated Annealing y Algoritmo Genético, para encontrar un punto interior de iluminación máximo. Este software presenta diversas funcionalidades que lo hacen muy completo para resolver el problema en cuestión.

Los algoritmos heurísticos se han adaptado para obtener una solución óptima aproximada para el problema de maximización de la visibilidad de polígonos. En la comparación entre estos algoritmos, se ha encontrado que los métodos proporcionan soluciones similares, residiendo la principal diferencia en el tiempo de procesamiento necesario para cada uno de los algoritmos. El algoritmo Random Search genera soluciones muy eficientes en cuanto al tiempo de ejecución y el área iluminada. El algoritmo Simulated Annealing obtiene las soluciones óptimas con un costo de procesamiento significativamente superior. El Algoritmo Genético ofrece soluciones muy cercanas al algoritmo Simulated Annealing, pero sus tiempos de ejecución son muy elevados y crecen en una mayor proporción con la dificultad del problema.

Los casos de uso propuestos para esta aplicación se centran en la iluminación de espacios interiores y exteriores, la navegación marítima, la seguridad, la localización de cámaras de seguridad y sensores de movimiento, así como en el análisis de puntos ciegos en espacios interiores y exteriores. Además, en términos de comunicaciones direccionales, estas técnicas pueden ser útiles para la planificación y reducción de costos. La aplicación generada puede ser de gran utilidad para la divulgación de las diferentes técnicas metaheurísticas para la resolución de problemas de visibilidad en el campo de la Geometría Computacional.

Como posibles trabajos futuros, se plantea la implementación de más técnicas heurísticas para dar solución a problemas de mayor dificultad, como el Método del Gradiente, o el algoritmo Ant System, entre otros. Además, se pueden utilizar técnicas más eficientes en el cálculo de los polígonos de visibilidad para disminuir el tiempo de procesamiento de las técnicas heurísticas. Por último, se podrían incorporar más funcionalidades en la aplicación, como un menú para editar, guardar e insertar polígonos, para adaptar la aplicación a casos más reales.

6. Referencias

[1] Canales Cano, Santiago. "*Métodos heurísticos en problemas geométricos. Visibilidad, iluminación y vigilancia*". Tesis doctoral, 2004.

[2] Chvátal, V. "*A Combinatorial Theorem in Plane Geometry*". Journal of Combinatorial Theory, Serie B, 18, pp. 39-41, 1975.

[3] Lee D.T.; Lin A.K. "*Computational complexity of art gallery problem*". IEEE Trans. Info. Th. IT-32, pp. 415-421, 1979.

MAXIMIZING THE VISIBILITY REGION OF A POINT IN A POLYGON USING METAHEURISTIC TECHNIQUES

Autor: Pérez Bienzobas, Daniel.
Director: Canales Cano, Santiago.
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

ABSTRACT

The project consists of the creation of a graphical software for maximizing the visibility region of a point inside a polygon *P* with *n* vertices, by applying three metaheuristics: Random Search, Simulated Annealing, and Genetic Algorithm.

Keywords: Heuristic Techniques, Metaheuristics, Optimization, Polygon Visibility, Computational Geometry, Random Search, Simulated Annealing, Genetic Algorithm.

1. Introduction

Computational Geometry focuses on the study of mathematical algorithms and techniques to solve geometric problems using computer systems. Solving these problems requires knowledge of geometry and algorithmic skills.

The resolution of visibility problems is an important application of Computational Geometry in fields such as Robotics, Computer Graphics, Architecture, and Urbanism. Chvátal proposed the Art Gallery Theorem, which solves the problem of fully illuminating a polygon. Later, Lee and Lin demonstrated that the optimization of this solution is a complex problem that requires metaheuristic methods for efficient resolution.

2. Project Definition

The main objective of this project is to generate an application that solves visibility problems in a practical and visual tool that allows understanding heuristic and metaheuristic methods in the field of Computational Geometry. The application has been developed in Python, an open-source programming language that allows for continuous program improvement. The main problem addressed is to minimize the number of interior lights illuminating a polygon completely while providing different methods to obtain the point of maximum illumination within a polygon. This application serves as a starting point for solving the more complex problem that aims to find k points of maximum illumination inside a polygon.

To achieve a solution to the main problem, three heuristic algorithms have been employed: Random Search (RS), Simulated Annealing (SA), and the Genetic Algorithm (GA). The objective of the project is to exhaustively study the MaxA-p-Pv1(*P*) problem, which consists of finding the point of maximum illumination inside a simple "no-hole" polygon *P* with *n* vertices. This problem corresponds to a particularization of the general problem MaxA-p-Pvk(*P*,*k*), where *k* represents the number of point lights located to find maximum visibility. Both mentioned problems correspond to a preliminary step to provide a solution to the problem identified as

MinN-p-Pvk(*P*), which aims to minimize the number of lights that make a polygon *P* completely illuminated.

The MaxA-p-Pv1(P) problem, which is the problem we are addressing in the present project, can be detailed as follows:

Question: Where should a *point light p* be located inside a simple "no-hole" polygon *P* with *n* vertices so that the visibility area from that point *p* is maximum?
Input: A simple "no-hole" polygon *P* with *n* vertices.
Output: The point *p* within *P* which the visibility polygon has the maximum area.

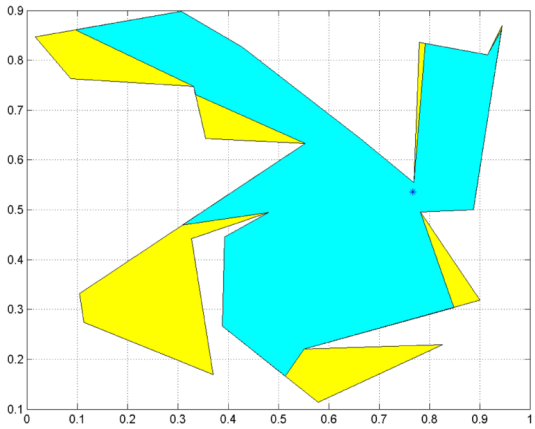

Figure 3: Visibility polygon given a point inside a polygon P with 24 vertices

3. Description of the application architecture

The project software is divided into three files that will be explained below. *TEHEMAXRVP.py* is responsible for building the main window and its components, as well as providing auxiliary functions for the algorithms used in the application. *Funciones.py* contains two functions that are important for the application's operation: one function to calculate visibility polygons and another to convert screen coordinates to Cartesian coordinates. *Appdesign.py* is responsible for configuring the main window and its components and has been created with the help of the Qt Designer application.

Figure 4: Architecture of the graphical application

4. Results

In this project, a comparative study of three metaheuristic techniques has been carried out to maximize the visibility polygon of a point inside a polygon. The study has focused on the analysis of processing times and the percentage of visibility obtained by each technique. 25 polygons with 15, 25, and 35 vertices each were randomly selected and Random Search, Simulated Annealing, and Genetic Algorithm were used. The average results for the 25 polygons of each vertex size are shown in Tables 1, 2, 3, and 4, for each of the respective algorithms.

It is concluded that the algorithm that provides better results for maximizing the visibility area relative to a point inside a polygon P with n vertices is Simulated Annealing, but the algorithm that is apparently more efficient taking into account processing time is Random Search. The Genetic Algorithm provides favorable results, very close to the SA technique in some cases, but its processing time seems to be too high and increases to a greater extent as the problem's difficulty increases.

In addition, an analysis of the average processing times of the different techniques used for the different numbers of vertices of the studied polygons has been carried out. Although extrapolatable conclusions cannot be drawn regarding the average processing time of the algorithms and the number of vertices of the analyzed polygons, it can be observed that the RS algorithm presents a more linear growth in relation to the number of vertices of the polygons, while the GA algorithm presents the greatest increase in processing time in relation to the increase in the number of vertices, followed by the SA algorithm and the RS.

In summary, the study has demonstrated the effectiveness of metaheuristic techniques to maximize polygon visibility and has provided valuable information about the processing times and the percentage of visibility obtained by each technique.

Vertices	*% Visibility Area*	*Processing Time [s]*
15	93,71	0,26
25	84,26	0,34
35	88,11	0,51

Table 5. Random Search with 250 points results

Vertices	*% Visibility Area*	*Processing Time [s]*
15	93,78	0,46
25	84,42	0,66
35	88,24	0,99

Table 6. Random Search with 500 points results

Vertices	*% Visibility Area*	*Processing Time [s]*
15	93,92	27,70
25	84,66	82,18
35	88,36	127,16

Table 7. Simulated Annealing $T_0 = Vértices \quad T(k) = \frac{T_0}{1+k} \quad T_f = 0.03 \quad N(T) = \frac{1}{T}$ *results*

Vertices	*% Visibility Area*	*Processing Time [s]*
15	93,61	47,18
25	84,55	203,90
35	88,31	478,06

Table 8. Genetic Algorithm $p_m = 0.5$ *results*

5. Conclusions

The main objective of this project is the development of an interactive software to solve the MaxA-p-Pv1(*P*) problem in which we seek to maximize the visibility polygon in a polygon *P* with *n* vertices starting from an interior point. The software allows to set up a polygonal problem and apply three metaheuristic techniques, Random Search, Simulated Annealing and Genetic Algorithm, to find an interior point of maximum illumination. This software presents several functionalities that make it very complete to solve the problem in question.

The heuristic algorithms have been adapted to obtain an optimal approximate solution for the polygon visibility optimization problem. In the comparison between these

algorithms, it has been found that the methods provide similar solutions, but the main difference lies in the processing time required for each of the algorithms. The Random Search algorithm generates very efficient solutions in terms of execution time and illuminated area. The Simulated Annealing algorithm obtains the most optimal solutions with a significantly higher processing cost. The Genetic Algorithm offers solutions very close to the Simulated Annealing algorithm, but its execution times are very high and grow in greater proportion to the difficulty of the problem.

The use of this application focuses on indoor and outdoor lighting, maritime navigation, security, localization of security cameras and motion sensors, as well as the analysis of blind spots in indoor and outdoor spaces. Furthermore, in terms of directional communications, these techniques can be useful for planning and cost reduction. The application generated can be very useful for the dissemination of different metaheuristic techniques for solving visibility problems in the field of Computational Geometry.

As possible future work, we propose the implementation of more heuristic techniques to solve more difficult problems, such as the Gradient Method, or the Ant System algorithm, among others. In addition, more efficient techniques can be used in the calculation of visibility polygons to reduce the processing time of heuristic techniques. Finally, more functionalities could be incorporated into the application, such as a menu for editing, saving and inserting polygons, to adapt the application to more real cases.

6. References

[1] Canales Cano, Santiago. "*Métodos heurísticos en problemas geométricos. Visibilidad, iluminación y vigilancia*". Tesis doctoral, 2004.

[2] Chvátal, V. "*A Combinatorial Theorem in Plane Geometry*". Journal of Combinatorial Theory, Serie B, 18, pp. 39-41, 1975.

[3] Lee D.T.; Lin A.K. "*Computational complexity of art gallery problem*". IEEE Trans. Info. Th. IT-32, pp. 415-421, 1979.

Índice de la memoria

Índice de figuras

Índice de tablas

Capítulo 1. INTRODUCCIÓN

1.1 INTRODUCCIÓN A LA GEOMETRÍA COMPUTACIONAL

La Geometría Computacional es una rama de la computación enfocada en el estudio de algoritmos y distintas técnicas matemáticas que permiten la resolución de diferentes problemas geométricos. Esta ciencia se desarrolla a partir de la geometría analítica, que pretende implementar conceptos y técnicas matemáticas avanzadas en sistemas computacionales. Por tanto, la solución de un problema de geometría computacional requiere, por un lado, entender la geometría del problema, y por otro lado, hacer un uso eficiente de conceptos algorítmicos y estructuras de datos.

Son muchas las aplicaciones de esta disciplina en numerosos campos de estudio, como el Diseño Asistido por Computadora, los Sistemas de Información Geográficos y el Reconocimiento de Imágenes, pero además su crecimiento alcanza nuevas áreas de estudio como la Inteligencia Artificial y la Robótica.

1.2 LA GEOMETRÍA COMPUTACIONAL Y LOS PROBLEMAS DE VISIBILIDAD

Los problemas de visibilidad tratados en este trabajo se engloban dentro de un campo de la Geometría Computacional, que podemos denominar “Problemas de Visibilidad en Geometría Computacional”, y su resolución tiene muchas aplicaciones útiles en la vida real, como en Robótica, Computación Gráfica, Planificación del Movimiento, Arquitectura, Urbanismo, y otras disciplinas.

El problema de la Galería de Arte propuesto por V. Klee en 1973, cuestionaba el número de puntos interiores a un polígono *P* con *n* vértices que eran suficientes para iluminar el resto de puntos, abría esta nueva área de estudio [1].

Este problema fue resuelto por Chvátal con el Teorema de la Galería de Arte, en el que se demuestra que $\lfloor n/3 \rfloor$ luces eran suficientes y en muchas ocasiones necesarias para iluminar completamente un polígono *P* de *n* vértices [2].

Tras realizar la triangulación del polígono simple cóncavo (en un polígono convexo un punto de vigilancia es siempre suficiente para vigilar el multilátero) de 13 vértices de la siguiente figura, se puede observar que solamente es necesario un punto de vigilancia en uno de los vértices para cada uno de los triángulos resultantes. Colocando puntos de tres colores diferentes en cada uno de los vértices, con la única regla de que dos vértices del mismo color no pueden estar unidos en el polígono triangulado, se llega a la conclusión de que escogiendo los puntos con el color menos frecuente, queda vigilado por completo el polígono.

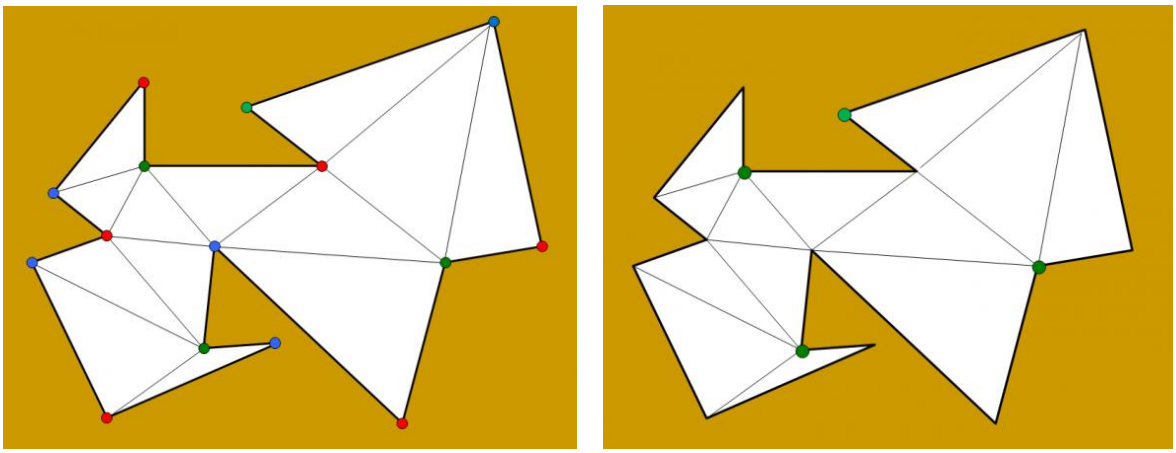

Figura 5: Ejemplo de un polígono cóncavo para la explicación del Teorema de la Galería de Arte [3]

Este fenómeno sucede porque cada uno de los triángulos tiene un vértice verde, por lo tanto todos los triángulos con un vigilante en dichos puntos quedarían completamente vigilados. La elección del color verde es por optimización del número de puntos de

vigilancia, pero siempre se va a cumplir que haya un color con $\lfloor n/3 \rfloor$ o menos puntos, demostrando el Teorema de la Galería de Arte.

Sin embargo, siendo esta la cota superior para la resolución del problema se plantea su optimización para diferentes tipos de polígonos. Este último problema ha sido estudiado por Lee y Lin [4] quienes llegaron a la conclusión que se trataba de un problema de naturaleza NP-dura, utilizando una reducción al problema 3-SAT [5].

Para la resolución de este problema surge la necesidad de emplear métodos metaheurísticos, que tratan de resolver problemas complejos de optimización combinatoria de forma eficiente, empleando algoritmos híbridos que combinan diferentes conceptos de distintos campos como la Inteligencia Artificial, la Evolución Biológica y la Estadística, como introducían Osman y Kelly en 1995 [6].

1.3 MOTIVACIÓN DEL PROYECTO

La principal motivación para la realización de este proyecto consiste en generar una aplicación útil que permita resolver problemas de visibilidad, de tal manera que sea una herramienta práctica y visual para comprender la aplicación de los métodos heurísticos y metaheurísticas en este campo de la Geometría Computacional.

El desarrollo del software se ha realizado en Python, lenguaje de programación de código abierto que permite seguir desarrollando y mejorando las aplicaciones del programa. Cuestión interesante dado el grado de desarrollo que permite la solución del problema de visibilidad principal que consiste en minimizar el número de luces interiores a un polígono que lo iluminan por completo. En este proyecto se pretende aportar distintos métodos para obtener el punto de máxima iluminación dentro de un polígono, dejando para trabajos futuros la búsqueda de *k* puntos de máxima iluminación interiores a un polígono.

Por tanto, este trabajo y la aplicación desarrollada sirven como punto de partida para resolver el problema de mayor complejidad comentado anteriormente, por lo que representa un software inicial que se pretende ir mejorando con la implementación de distintas técnicas y funcionalidades con el objetivo de alcanzar una solución al problema principal.

La tesis desarrollada por el profesor Dr. Santiago Canales Cano [7], investigador en el campo de la Geometría Computacional y la Matemática Discreta, es la principal fuente de inspiración utilizada para el desarrollo de este proyecto.

Capítulo 2. DESCRIPCIÓN DE LAS TECNOLOGÍAS

Para el desarrollo del proyecto se han empleado los siguientes recursos:

- HP Pavilion Laptop 15-cs3xxx con un SO de 64 bits Windows 10 con un procesador Intel(R) Core(TM) i7 1.30GHz-1.50GHz y 16 GB de memoria RAM.
- Python versión 3.9.12.
- Spyder IDE versión 5.1.5.
- Aplicación Qt Designer como herramienta de diseño para interfaces gráficas de usuario.
- Utilización de diversas librerías de Python, destacando PyQt5 para la creación de interfaces gráficas y Scikit-geometry para calcular polígonos de visibilidad.

La librería PyQt5 de Python, *wrapper* de la biblioteca Qt de C++, proporciona herramientas para generar aplicaciones de escritorio con una interfaz gráfica de usuario. Esta biblioteca dispone de una amplia gama de *widgets* con los que personalizar la interfaz gráfica, desde botones, cuadros de texto, etiquetas y contenedores gráficos hasta diálogos de alerta y ventanas de mensajes emergentes. Además, permite organizar la ventana principal en diferentes *frames* y la creación de barras de herramientas y menús de opciones [8].

En este proyecto se ha empleado la versión 5.15.7 de PyQt5 y como se ha mencionado anteriormente, para el diseño de la interfaz gráfica se ha utilizado la herramienta de software Qt Designer en su versión 5.11.1 (Figura 6), que ha facilitado el desarrollo y la personalización de los elementos de la interfaz gráfica, pudiendo diseñar la ventana

principal de manera gráfica y exportar su código en Python de manera automática para así poder conectar los elementos de la interfaz con sus funcionalidades [9].

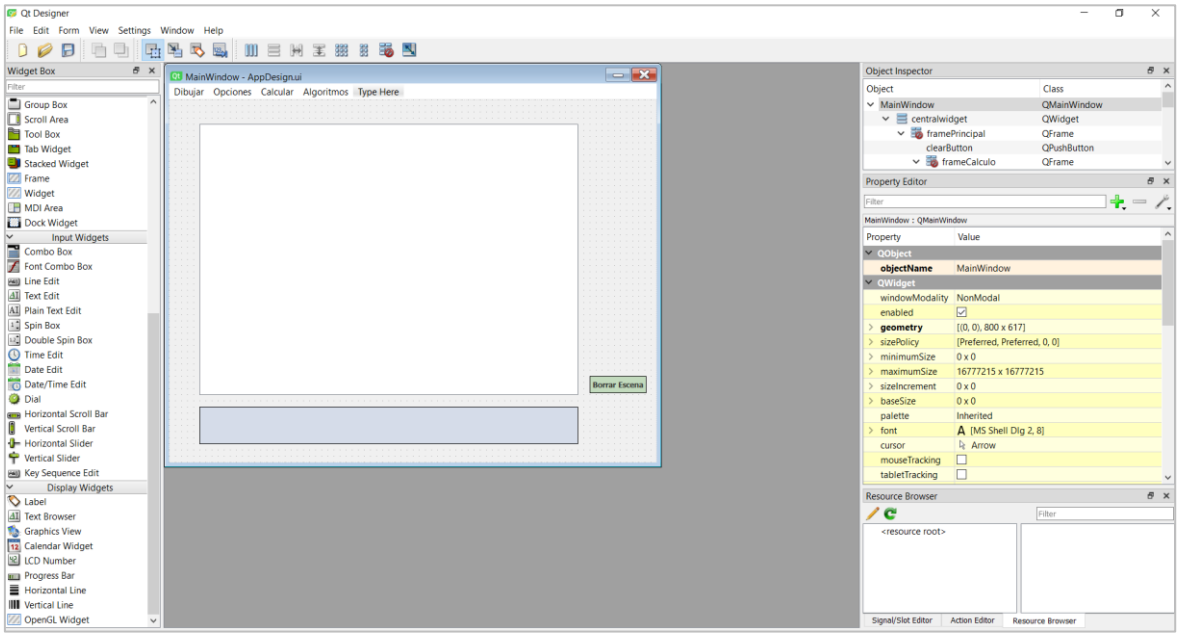

Figura 6: Herramienta de software Qt Designer (versión 5.11.1) para el diseño de la app TEHEMAXRVP

En relación con el cálculo de los polígonos de visibilidad dado un punto interior a un polígono, se ha empleado la librería Scikit-geometry y fundamentalmente la función *RotationalSweepVisibility* que implementa el algoritmo de T. Asano [10] de barrido rotacional que es una generalización del algoritmo de Lee para polígonos con "agujeros" y presenta una complejidad de $O(n \cdot \log n)$ en función del número de vértices del polígono principal. En el siguiente capítulo se incidirá acerca del funcionamiento de dicho algoritmo, que en este caso particular será utilizado en polígonos simples sin "agujeros" aunque es también funcional para polígonos que presentan "agujeros" [11].

Capítulo 3. ESTADO DE LA CUESTIÓN

3.1 POLÍGONOS DE VISIBILIDAD

Un polígono de visibilidad de un punto *p* interior a un polígono *P*, es el área interior a *P* que es visible desde *p*, mientras que ciertas áreas permanecen ocultas detrás de obstáculos. Por otro lado, se puede generar un polígono de visibilidad a través de la unión de distintos polígonos de visibilidad generados desde diferentes puntos interiores al polígono. Existen algunos algoritmos que permiten calcular la unión de dos polígonos utilizando técnicas de recorte de segmentos y polígonos, como los algoritmos de Liang-Barsky, Cyrus-Beck, Cohen-Sutherland, Nicholl-Lee-Nicholl y Weiler-Atherton [12, 13]; pero no se va a incidir más sobre las técnicas de unión de polígonos, ya que en este proyecto se pretende maximizar el área de visibilidad dado un punto interior a un polígono, lo que hace que siempre se trabaje con un único polígono de visibilidad (Figura 7).

Todos los polígonos con los que se va a trabajar en la optimización del área de visibilidad son simples sin "agujeros", de tal forma que los segmentos que forman el polígono no se interceptan en ningún punto y además no se encuentran obstáculos interiores a la delimitación poligonal. Además, es importante puntualizar que los puntos interiores al polígono que se estudian tienen la característica de iluminar en un radio de 360º y sin limitación en el alcance. Sería interesante el estudio de estas limitaciones en futuros trabajos

Por otro lado, los polígonos que son interesantes para esta área de estudio son los considerados no convexos, es decir, que tienen algún ángulo interior que es mayor de 180º. Esto es debido a que el polígono de visibilidad de un polígono convexo desde cualquier punto interior del plano representa la misma área del polígono, dicho de otra manera, la visibilidad del polígono es total desde cualquier punto interior al polígono.

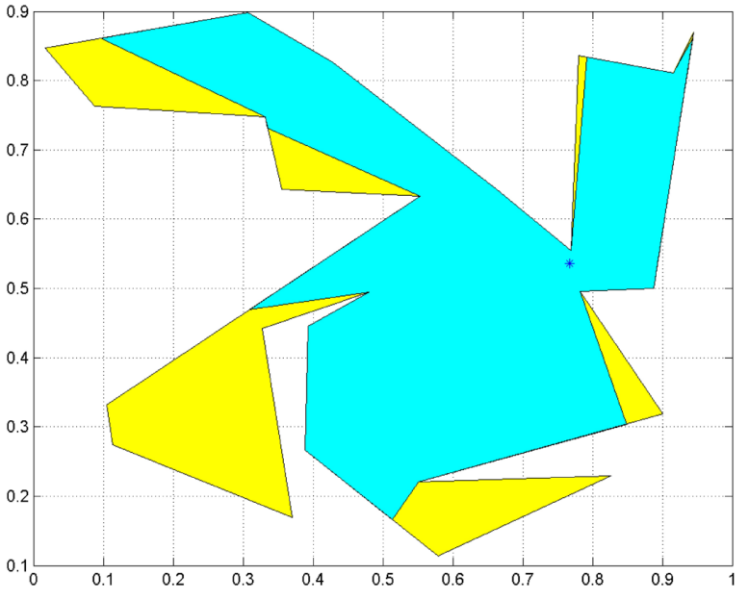

Figura 7: Polígono de visibilidad dado un punto interior a un polígono de 24 vértices [7]

Para calcular el polígono de visibilidad desde un punto interior a un polígono se pueden seguir los siguientes pasos (se puede ver ejemplo en la Figura 8):

- Trazar líneas rectas desde el punto interior hacia todos los vértices del polígono.
- Determinar las intersecciones producidas por esas líneas y los bordes del polígono.
- Clasificar las intersecciones en puntos visibles o no visibles, eliminando las líneas que cruzan los bordes del polígono en áreas no visibles.
- Construir el polígono de visibilidad con los bordes del polígono que no están cruzados por líneas.

El proceso descrito anteriormente se puede optimizar empleando el algoritmo de visibilidad de Lee, publicado inicialmente en 1983 y corregido posteriormente por B. Joe y R.B. Simon en 1987. Este algoritmo se basa en la eliminación de líneas ocultas que permiten señalar las partes de un polígono que son visibles desde un punto de vista específico, de forma que una estructura de datos que podemos denominar “ventana de

visibilidad" se va actualizando con los puntos visibles a medida que se recorre el borde del polígono previamente marcado por el trazo de las líneas hacia los vértices del polígono.

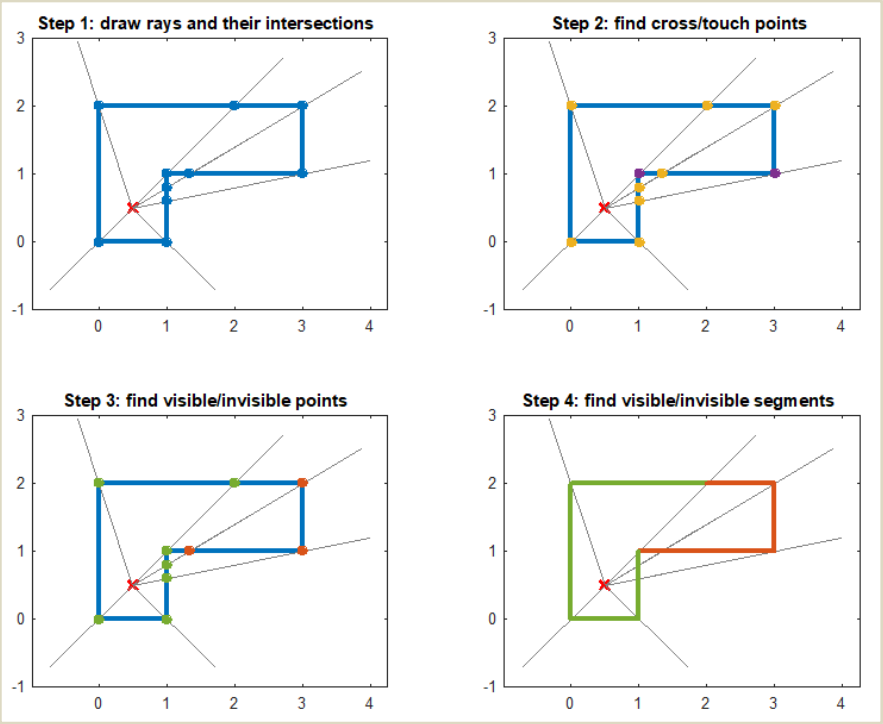

Figura 8: Ejemplo de cómo calcular el polígono de visibilidad desde un punto interior a un polígono [14]

Como se comentaba anteriormente en este proyecto se ha empleado el algoritmo de barrido rotacional para determinar los polígonos de visibilidad. Esta técnica funciona mediante el escaneo de la escena desde un punto de vista determinado, analizando los bordes de los objetos de forma ordenada según su ángulo polar en relación con el punto de vista específico, logrando un escaneo ordenado y sin repetición. El polígono de visibilidad se forma añadiendo los bordes visibles hasta finalizar el barrido de todo el escenario y se finaliza con la eliminación de los bordes redundantes mediante la identificación de bordes colineales.

3.2 INTRODUCCIÓN A LOS MÉTODOS HEURÍSTICOS Y A LAS METAHEURÍSTICAS

Los métodos heurísticos proporcionan soluciones de problemas de forma eficiente basándose en la intuición, la observación empírica y la experiencia, pero no siempre se encuentra la solución óptima y a veces incluso ni siquiera se produce una solución. Una heurística puede ser probabilística o determinista y algunos ejemplos de estos métodos son: métodos de reducción, métodos de mejora, métodos de búsqueda local, métodos de construcción, etc.

Por otro lado, una metaheurística es un método heurístico de carácter general y flexible que se emplea para resolver problemas complejos de optimización combinatoria y alcanzar una solución óptima o aproximada de manera eficiente cuando no es trivial encontrar un método que proporcione una solución exacta. Por tanto, son métodos generales y al contrario que las heurísticas específicas se pueden emplear para encontrar soluciones de múltiples problemas.

En 1986, Fred Glover acuñó el término metaheurística y desde entonces han surgido diversas propuestas para mejorar la resolución de problemas combinatorios. Según Osman y Kelly (1995), la metaheurística representa un conjunto de procedimientos aproximados dirigidos a resolver problemas combinatorios de optimización que no pueden ser abordados de manera efectiva por heurísticas clásicas.

Las metaheurísticas ofrecen un marco amplio para crear algoritmos híbridos que combinan conceptos de diferentes campos como la genética, la biología, la inteligencia artificial, las matemáticas, la física y la neurología, entre otros [15]. Para dar una solución al problema principal planteado para este trabajo, que se explicará a continuación, se han empleado los siguientes algoritmos: Random Search (RS), Simulated Annealing (SA) y el Algoritmo Genético (GA).

3.3 PROBLEMA MAXA-P-PV1(P) A RESOLVER

El propósito de este proyecto es estudiar de manera exhaustiva el problema denotado como MaxA-p-Pv1(*P*), consistente en la búsqueda del punto de máxima iluminación interior a un polígono *P* de *n* vértices, que se corresponde con una particularización para $k = 1$ del problema general MaxA-p-Pvk(*P*,*k*), donde *k* representa el número de *luces punto* localizadas para encontrar la máxima visibilidad. Cabe recordar que ambos problemas mencionados se corresponden con el paso previo para dar una solución al problema que se identifica como MinN-p-Pvk(*P*), que pretende minimizar el número de luces que hacen que se ilumine por completo un polígono *P*.

El problema MaxA-p-Pv1(*P*) se puede enunciar de la siguiente forma:

Pregunta: ¿Dónde se debe localizar una *luz punto p* en el interior de un polígono simple "sin agujeros" *P* de *n* vértices para que el área de visibilidad desde dicho punto *p* sea máxima?
Entrada: Un polígono simple "sin agujeros" *P* de *n* vértices.
Salida: El punto *p* interior a *P* cuyo polígono de visibilidad es de área máxima.

Para abordar el problema MaxA-p-Pv1(*P*) se proponen tres metaheurísticas (RS, SA y GA), cuyas implementaciones serán explicadas en detalle en las siguientes secciones y en las que también se mostrarán ejemplos de aplicación de los algoritmos para un mismo polígono de 13 vértices.

3.3.1 RANDOM SEARCH – RS EN MAXA-P-PV1(P)

El algoritmo Random Search o búsqueda aleatoria consiste en realizar una búsqueda aleatoria sobre *m* puntos interiores a un polígono *P*, de tal manera que se obtenga aquel punto de un conjunto $N = \{p_1, \dots, p_m\}$ cuya región de visibilidad generada sea máxima. Siendo $V(P, p_i)$ el polígono de visibilidad generado por el punto p_i sobre el polígono *P* se debe de recorrer de forma secuencial el conjunto *N* tal que

$$Área\big(V(P, p_i)\big) \geq Área\big(V(P, p_k)\big) \quad \forall k \neq i \quad i, k \in \{1, \dots, m\}$$

obteniendo aquel punto $p_i \in N$ cuya región de visibilidad sea máxima.

El número *m* de puntos aleatorios empleados para el algoritmo determinará su grado de optimización, pero está claro que cuanto mayor sea esta variable mayor será el tiempo de procesamiento del algoritmo, ya que se debe calcular el polígono de visibilidad $V(P, p_i)$ para cada uno de los puntos pertenecientes a *N*.

A priori sería razonable encontrar una razón dependiente del número de vértices del polígono *P* con el que se trabaja, para determinar *m* y encontrar una solución aproximada que podría ser óptima considerando un tiempo de ejecución razonable, pero dados los bajos tiempos de procesamiento del algoritmo, se deja al usuario elegir el valor de *m* que quiere emplear para resolver el problema de optimización.

El Anexo II contiene el código del algoritmo RS, el cual se ha implementado en una función nombrada como *algoritmoRandomSearch*. Sin embargo, para una mejor comprensión del algoritmo, se presenta a continuación su pseudocódigo. Asimismo, se añade el pseudocódigo de la función *puntosAleatorios*, la cual genera una nube de puntos aleatorios cuya longitud es *m*. Este código completo puede hallarse nuevamente en el Anexo II.

Algoritmo RS-MaxA-P-Pv1(P)

Entrada: Un polígono *P* de *n* vértices.
Salida: Un punto *p* de máxima iluminación en *P*.

```
[01]   N ← Generar_Puntos_Aleatorios(P,m);
[02]   lista_área ← [];
[03]   for (p_i ∈ N){
[04]          Calcular el polígono de visibilidad V(P,p_i);
[05]          Calcular el % de área de visibilidad por p_i y añadir a lista_área;
[06]   }
[07]   Obtener p que será aquel punto cuyo valor en lista_área sea mayor;
[08]   return p;
```

Función Generar_Puntos_Aleatorios

Entrada: Un polígono *P* de *n* vértices y un número entero *m*.
Salida: Un conjunto $N = \{p_1, \dots, p_m\}$ de *m* puntos interiores a *P*.

```
[01]   i ← 1;
[02]   N ← [];
[03]   do{
[04]          Obtener unas coordenadas x_i e y_i de manera aleatoria en función de
               los bounds de P;
[05]          if (p_i = (x_i, y_i) interior a P){
[06]                 p_i ← (x_i, y_i);
[07]                 N ← N ∪ p_i;
[08]                 i ← i+1;
[09]          }
[10]   }while(i<=m);
[11]   return N;
```

A continuación, se muestra la solución al problema MaxA-p-Pv1(*P*) dado un polígono de 13 vértices empleando el método RS utilizando *m*=250 puntos (Figura 9). Se puede observar que el punto *p* de máxima iluminación se encuentra localizado en las coordenadas cartesianas (-18.57, -18.8), el porcentaje de área iluminada sobre *P* es 95.6% y el tiempo de procesamiento del algoritmo 0.25 [s]. El mismo polígono de 13 vértices será empleado para analizar las soluciones de las siguientes metaheurísticas.

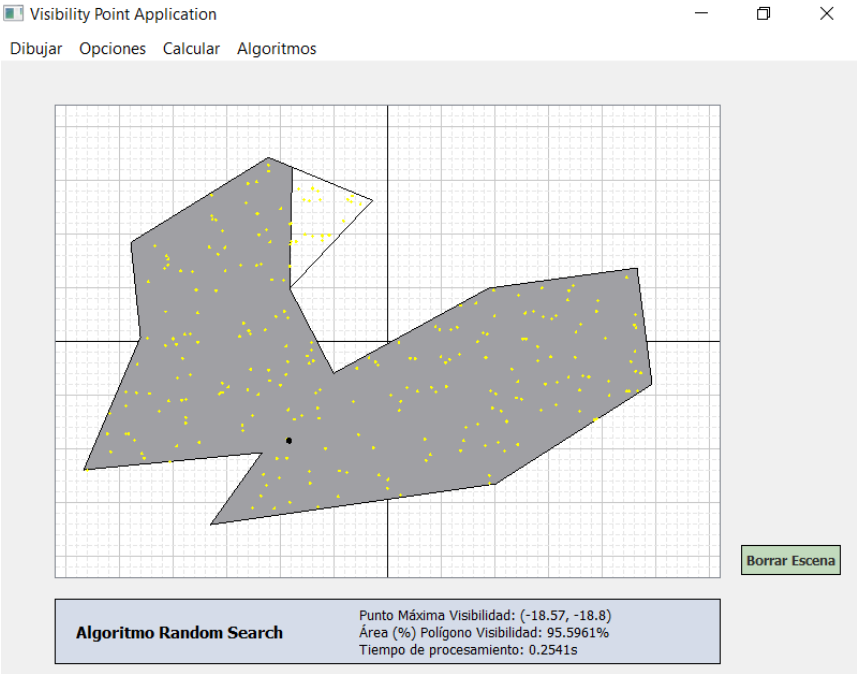

Figura 9: Aproximación del problema MaxA-p-Pv1(P) con RS m=250

3.3.2 SIMULATED ANNEALING – SA EN MAXA-P-PV1(P)

El método Simulated Annealing o reconocido simulado tiene su origen en el templado que se puede producir en distintas sustancias, estando el enfriamiento en este caso asociado a la búsqueda de puntos interiores que maximicen la región de visibilidad en un polígono *P* de *n* vértices. De esta forma, cuanto menor sea la temperatura en el proceso de búsqueda mayor será la probabilidad de que el nuevo punto generado sea más cercano al óptimo, pero nos permite contrariamente con cierta probabilidad que las soluciones no queden en torno a un máximo local, permitiendo saltar a puntos más alejados aunque no se produzca temporalmente una mejora en la función de coste. De este modo, se realiza el análisis de una nube de puntos que presenta mayor dispersión, si bien se mantendrá dentro del entorno óptimo del punto de referencia.

Esta metaheurística se puede expresar de forma general de la siguiente manera:

Dado un espacio finito de soluciones $S = \{x_1, \dots, x_m\}$, donde m se corresponde con la dimensión de S, y una función de coste $C: S \to \mathbb{R}$, se pretende determinar $x^* \in S$ tal que $C(x^*) \leq C(x) \ \ \forall x \in S$. Teniendo en cuenta que $x \in S$ es la configuración inicial, *N(T)* el número de iteraciones para cada temperatura T, siendo $T_0 > 0$ la temperatura inicial, el pseudocódigo para el algoritmo SA es el siguiente:

```
[01]   do{
[02]          do{
[03]                 {Crear solución y ∈ Vecindad(x) ⊂ S;
[04]                  Evalúa δ ← C(x) - C(y);
[05]                  if (δ <0) x ← y
[06]                  else
[07]                        if ((δ >0) Λ (U(0,1) < e^(-δ/T))   x ← y;
[08]                  n ← n+1;
[09]          }while (n ≤ N(T));
[10]          Reducir T
[11]   }while (parada == false)
```

La implementación del algoritmo adaptado al problema en cuestión se encuentra en el Anexo II, descrito en la función *algoritmoSimulatedAnnealing*. Por otro lado, la función *generarVecinoSA*, la cual también se encuentra descrita en el Anexo II, tiene el cometido de generar un vecino dado un punto ya analizado. El pseudocódigo de esta función es el siguiente:

Función Generar_Vecino

Entrada: Un polígono P de n vértices, un punto $p_i = (x_i, y_i)$ y un factor de vecindad *div*.
Salida: Un punto $p'_i = (x'_i, y'_i)$ interior a P.

```
[01]   do{
[02]          Obtener unas coordenadas u1 e u2 de manera aleatoria en función de
              los bounds de P;
[03]          x'i = xi + u1/div
[04]          y'i = yi + u2/div
[05]   }while(p'i = (x'i, y'i) exterior a P);
[06]   return p'i;
```

En lo relativo al pseudocódigo anterior hay varios conceptos que aclarar. Por un lado, en la implementación realizada no siempre se suman las coordenadas u_1 y u_2 a las coordenadas x_i e y_i respectivamente del punto p_i, si no que se plantea de forma aleatoria la dirección de la suma en los diferentes cuadrantes del eje de coordenadas. El motivo de implementar dicha consideración se encuentra en que la orientación del punto vecino calculado no sea siempre la misma. Por otro lado, el factor de vecindad *div* es creciente con el paso de las iteraciones y está definido como *div/0.99*, consiguiendo de esta manera que el vecino cada vez esté más próximo al punto y, a su vez, se encuentre más cerca del punto óptimo.

En este algoritmo se deben tener en cuenta ciertos aspectos clave entre los que destacan:

- La función de coste $C(p_i)$, que representa el área del polígono de visibilidad generado por el punto p_i interior a *P*.
- La configuración inicial que se corresponde con un punto p_0 elegido de forma aleatoria empleando la función de generación de puntos aleatorios creada para el algoritmo RS,
- La temperatura inicial T_0 que será igual al número de vértices de *P*, adaptando de esta manera el algoritmo en función de *P*.
- La temperatura final T_f que será igual a 0.03 para cualquier configuración poligonal cuando el sistema está "frío". La temperatura final se relaciona con el criterio de parada total del algoritmo, ya que la temperatura es una función decreciente en relación con el número de iteraciones que se identifica de la siguiente forma $T(k) = \frac{T_0}{1+k}$, y por tanto cuando *T(k)* es menor o igual a T_f el algoritmo deja de buscar mejores soluciones.
- El número de iteraciones para cada temperatura $N(T) = \frac{1}{T}$ es una función inversamente proporcional a *T*, con lo que se consigue que la búsqueda se optimice para temperaturas bajas, ya que es cuando queremos que el algoritmo se aproxime al óptimo. Si tomamos *n* como el número de iteraciones para cada temperatura, cuando el *n* sea mayor que *N(T)* el algoritmo comenzará una nueva iteración con una temperatura menor.

En sí el algoritmo irá quedándose con los vecinos del punto anteriormente analizado en el caso de que mejore el porcentaje de visibilidad sobre este nuevo punto, pero como se explicaba anteriormente con una probabilidad $e^{(\frac{-\delta}{T})}$ y aunque no se cumpla la condición anterior, el nuevo punto a analizar será el vecino calculado. De esta forma se evita estancarse en torno a un máximo local, ampliando así el rango de estudio de los puntos interiores al polígono.

Cabe destacar que la determinación de los parámetros del algoritmo que se han ido explicando son la consecuencia de diferentes pruebas de eficiencia que se han ido realizando, aunque dependiendo del criterio que se quiera considerar puede haber otros valores para los parámetros más correctos.

En la siguiente imagen se muestra el resultado del problema MaxA-p-Pv1(*P*) del mismo polígono de 13 vértices empleado para el algoritmo RS, usando los parámetros mostrados a pie de la figura.

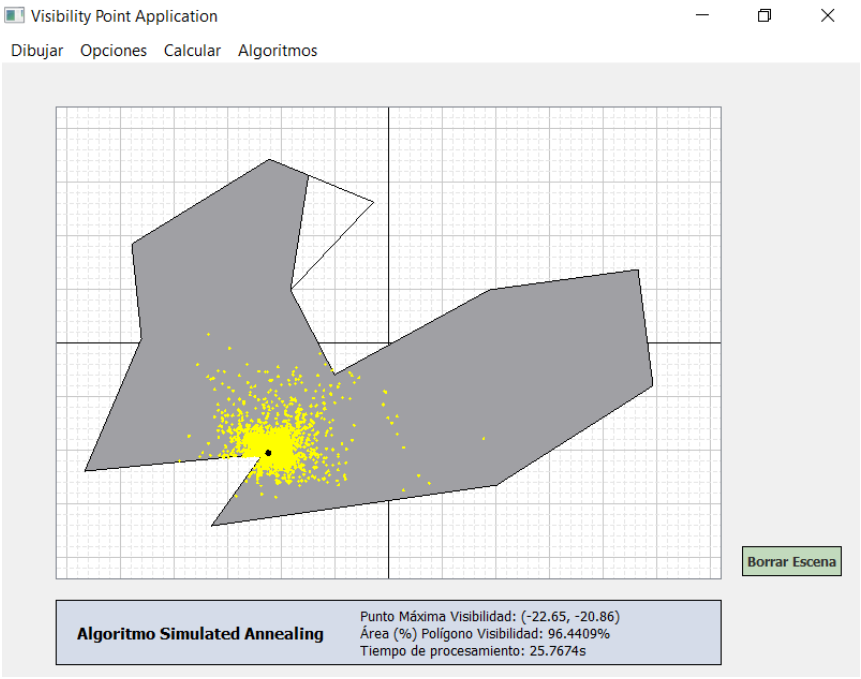

Figura 10: Aproximación de MaxA-p-Pv1(P) con SA $T_0 = 13$ $T(k) = \frac{T_0}{1+k}$ $T_f = 0.03$ $N(T) = \frac{1}{T}$

El punto óptimo encontrado por el algoritmo es el (-22.65, -20.86), que como se puede observar es cercano al punto determinado por el algoritmo RS, aunque se ha mejorado en aproximadamente un 1% su eficiencia en cuanto al porcentaje de área de visibilidad. Por el contrario, el tiempo de procesamiento (25.77 [s]) es más de 100 veces superior.

3.3.4 ALGORITMO GENÉTICO – GA EN MAXA-P-PV1(P)

Los algoritmos genéticos o evolutivos se presentan como una metodología robusta para la búsqueda y resolución de problemas de optimización que requieren de la identificación de un conjunto de variables óptimas para maximizar o minimizar una función de adaptación dada. Estos algoritmos trabajan con una población de individuos $A(t) = \{x_1^t, \dots, x_n^t\}$ en cada iteración *t*, donde cada individuo x_i^t se trata de un punto de búsqueda en el espacio de soluciones potenciales del problema en cuestión. La función de aptitud $f(x_i)$ estima el desempeño de un individuo x_i, permitiendo la clasificación ordenada de los individuos de la población de mejor a peor.

Lo que sucede es que la población inicial evoluciona mediante procesos probabilísticos sucesivos para localizar las mejores zonas de búsqueda dentro del espacio de soluciones. Por un lado, mediante la selección de los individuos mejor adaptados de la población, con lo que una mejor adaptación se convierte en una mayor probabilidad de dejar descendencia. Y, por otro lado, mediante la alteración a través de la mutación y/o combinación de los individuos elegidos.

Teniendo en cuenta estos conceptos un algoritmo genético básico se puede expresar de la siguiente forma:

```
[01]   t ← 0
[02]   Inicialización de A(t);
[03]   Evaluación de A(t);
[04]   do{
[05]          t ← t+1;
[06]          Elegir A(t) en función de A(t-1);
[07]          Procesos de mutación y/o recombinación de A(t);
[08]          Evaluación de A(t);
[09]   }while(Condición de parada);
```

Para el problema MaxA-p-Pv1(*P*) un individuo o "genoma" viene representado por las coordenadas de un punto y la población estará formada por un conjunto de individuos que irán evolucionando en cada iteración. El tamaño de la población estará relacionado con el número de vértices *n* del polígono *P*, de tal forma que el número de individuos de una población será igual a n^2. La población inicial será elegida de manera aleatoria, empleando la función implementada para el algoritmo RS que permite seleccionar un número de puntos interiores a *P*.

La función de aptitud, o función objetivo del problema, hace posible la selección de los genomas que deberán reproducirse, escogiendo a lo que serán los "padres" de la población. Estos "padres" serán los puntos mejor adaptados de la población, en este caso los que consigan iluminar una mayor área de visibilidad en *P*, para ello se debe calcular el polígono de visibilidad de todo el conjunto de la población y el porcentaje de área iluminada sobre *P*.

La selección de los "padres" de cada población se realiza mediante un proceso proporcional en función de los valores de la función de aptitud que en este caso se corresponderá con el Método de la Ruleta. Este método consiste en seleccionar de manera aleatoria una muestra dentro de un conjunto, de tal forma que no todos los individuos tienen la misma probabilidad *w* de ser escogidos, y esta probabilidad es asignada en función del grado de adaptación que en este caso se corresponde con el porcentaje de área

iluminada. Por tanto, existe una mayor probabilidad de que los puntos cuya área de visibilidad sea mayor salgan escogidos como los "padres" de la población.

La probabilidad de cada uno de los individuos se calcula dividiendo su porcentaje de área de visibilidad entre la suma de los porcentajes de todos los individuos de la población, obteniendo de esta forma una probabilidad comprendida entre 0 y 1 para cada individuo. Dichas probabilidades podrán ser representadas en una circunferencia como la mostrada en la Figura 11, de tal manera que se realiza un giro para obtener una muestra. Este muestreo se realiza con reposición, es decir, una vez que se extrae una muestra está volverá a ser introducida, con lo que el tamaño de la población permanece constante. En el problema en cuestión se seleccionan dos muestran, que serán los "padres" de la población.

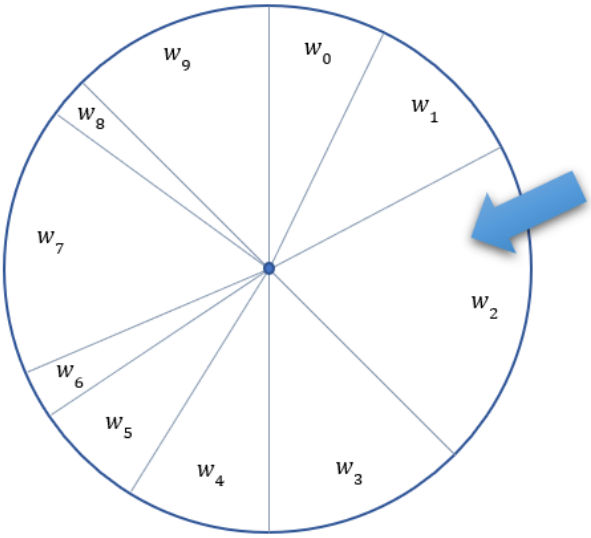

Figura 11: Método de la Ruleta con una probabilidad de selección w

Una vez escogidos los "padres", que serán también "hijos" para $k = 1$, se les aplica el operador de mutación siempre que la probabilidad de mutación p_m sea mayor que un número aleatorio r delimitado entre 0 y 1. Habiendo realizado pruebas para diferentes polígonos se ha determinado que la probabilidad de mutación sea del 50%. La mutación se

ha definido en una función que se encuentra en el Anexo II nombrada como *mutarPuntoGA* que no se vuelve a explicar ya que es idéntica a la utilizada para el caso de generar un vecino en el algoritmo SA. Cabe recordar que se ha empleado la misma técnica para que a lo largo de las iteraciones del algoritmo, la mutación de un "padre" sea cada vez más cercana a ese punto, de tal forma que cada nueva población tenderá a encontrar una solución óptima. He de señalar que para el problema MaxA-p-Pvk(*P*,*k*) existe un operador de cruce, pero para su simplificación con $k = 1$, no se considera este operador.

Por último, se ha considerado que la condición de parada del algoritmo se produce cuando el *fitness* de la población, es decir el punto que genera el área de visibilidad máxima de la población, no mejora durante un número h de generaciones. En este caso, tras haber realizado diferentes pruebas se ha considerado $h = 200$.

En la Figura 12 se observa la solución del problema MaxA-p-Pv1(*P*) empleando dicho algoritmo, siendo *P* el mismo polígono de 13 vértices utilizados para las metaheurísticas anteriormente explicadas.

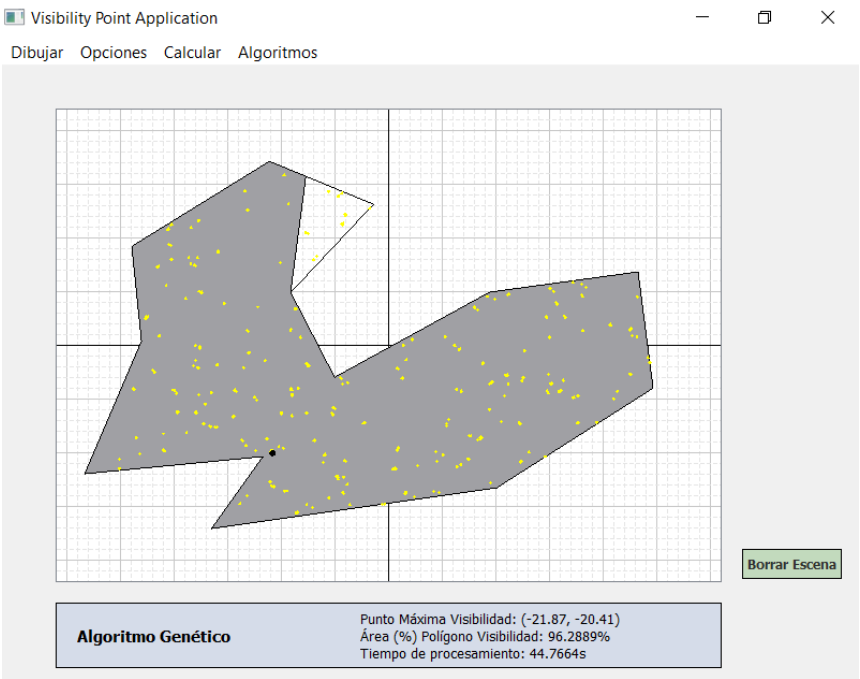

Figura 12: Aproximación del problema MaxA-p-Pv1(P) con GA $p_m = 0.5$

Las coordenadas del punto óptimo encontradas por el algoritmo GA son (-21.87, -20.41) y el porcentaje de área de visibilidad es 96.29%, ligeramente inferior al obtenido con el algoritmo SA y superior al obtenido con el algoritmo RS. Por otro lado, el tiempo de procesamiento del algoritmo (44.77 [s]) es muy elevado, casi dos veces superior al tiempo de ejecución del algoritmo SA y, por tanto, casi 200 veces superior al obtenido por el algoritmo RS. A priori, esta técnica metaheurística aunque arroja buenos resultados presenta un tiempo de procesamiento demasiado elevado, teniendo en cuenta además que no logra los mejores resultados. Pero las conclusiones sobre la eficiencia de los algoritmos serán demostradas en el estudio comparativo, que se explicará en capítulos posteriores, realizado para las tres técnicas metaheurísticas con diferentes formas poligonales y con distinto número de vértices.

Capítulo 4. DEFINICIÓN DEL TRABAJO

4.1 JUSTIFICACIÓN Y CASOS DE USO

Anteriormente se ha explicado que son los polígonos de visibilidad generados dado un punto interior a un polígono y las técnicas empleadas en este proyecto para maximizar su área de visibilidad. Con el objetivo de llevar la investigación y la aplicación realizada en esta materia a casos de uso reales, es importante aclarar que los puntos interiores al polígono analizados, no presentan limitaciones relativas respecto al ángulo de iluminación y al alcance, es decir, desde un punto de iluminación se configura un polígono de visibilidad con una visión 360º y con alcance ilimitado hasta los bordes del polígono. Sin embargo, en los casos de uso reales existen limitaciones respecto a estos dos aspectos mencionados, por lo que además sería una cuestión a tener en cuenta para futuras mejoras e implementaciones de la aplicación creada. Por otro lado, la aplicación trabaja con polígonos en 2D, por lo que las aproximaciones para los casos reales se realizan sobre planos bidimensionales.

Como se comentaba en secciones anteriores los problemas de visibilidad tienen muchas aplicaciones en distintos campos como la Robótica, la Computación Gráfica, la Planificación del Movimiento, la Arquitectura y el Urbanismo, pero este apartado del proyecto se va a centrar en analizar los casos de uso encontrados específicamente para las funcionalidades de la aplicación. Cabe recordar que la solución que proporciona la aplicación se corresponde con un problema que se engloba en uno más amplio, como es la minimización del número de puntos necesarios para visibilizar por completo un polígono.

Tras haber realizado un estudio de aplicaciones con características similares que se encuentran disponibles para su uso, se ha observado que la aplicación creada engloba muchas de las funcionalidades necesarias para estudiar de una manera eficiente el problema de visibilidad tratado.

A continuación, se detallan algunos casos de uso encontrados para las funcionalidades de la aplicación.

4.1.1 POLÍGONOS DE VISIBILIDAD E ILUMINACIÓN

En materia de iluminación la localización de los puntos de luz es un tema elemental que arquitectos y diseñadores de interiores deben tener en cuenta. La optimización del número de luces y su localización es fundamental para lograr una eficiencia energética y de costes. Por ello se presentan tres casos particulares en los que una aplicación que optimiza la visibilidad en diferentes tipos de espacios puede ser de gran utilidad, teniendo en cuenta que cualquier espacio con distintas formas geométricas se puede replicar o aproximar con un polígono.

4.1.1.1 Iluminación en espacios interiores

La iluminación de espacios es una cuestión que toma una gran importancia en el diseño de interiores, tanto de viviendas convencionales como de cualquier otro tipo de edificio. Por ejemplo, en los museos hay salas de diferentes formas en las que la correcta localización de luces permite que las obras de arte se visualicen de manera idónea, sin que haya zonas peor iluminadas.

Algunas salas del Museo Nacional del Prado (Figura 13) aunque aparentemente tienen formas rectangulares en las que la iluminación debería ser una tarea sencilla, se encuentran recovecos que hacen que pueda haber zonas que se queden mal iluminadas. Para ello una aplicación que es capaz de replicar el polígono que representa el plano de alguna de aquellas salas es de utilidad para descubrir cual sería la colocación perfecta de dichos puntos de luz.

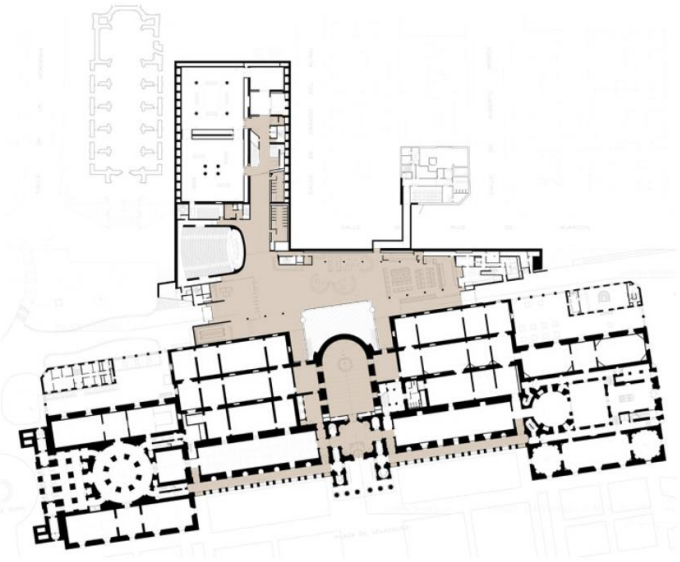

Figura 13: Plano de la ampliación de Rafael Moneo del Museo Nacional del Prado

De manera análoga, en habitaciones de viviendas con formas geométricas menos convencionales (Figura 14) en las que se pretende utilizar un solo punto de luz que consiga iluminar el mayor espacio posible, existe un claro interés en emplear las técnicas de optimización vistas anteriormente. Lo más interesante sería encontrar la mejor localización de luces que iluminen por completo la habitación, ya que en la figura posterior ¿sería suficiente con 2 puntos de luz o tendríamos que emplear 3? En este caso la respuesta parece trivial pero ¿qué pasaría si la geometría del problema fuera más complicada?

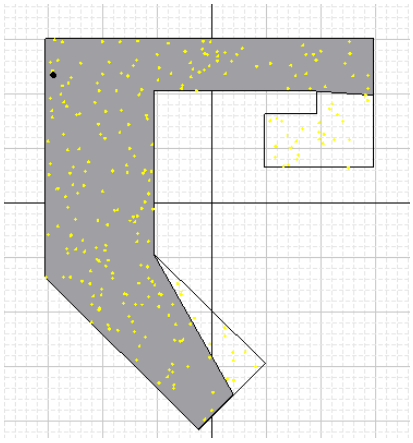

Figura 14: Ejemplo de maximización del área de visibilidad en un espacio interior

4.1.1.2 Iluminación en recintos exteriores

La iluminación en recintos exteriores es similar a la iluminación en espacios interiores, por el simple hecho de que el plano de cualquier espacio exterior se puede convertir en uno o múltiples polígonos. En la siguiente figura (Figura 15) se observa el plano de la antigua sede de Vocento y el Diario ABC, en el cual se ha trazado tres polígonos de ejemplo para mostrar la construcción con polígonos de los espacios exteriores del recinto, con los que ya se podría trabajar en la optimización de las áreas de visibilidad para mejorar la iluminación.

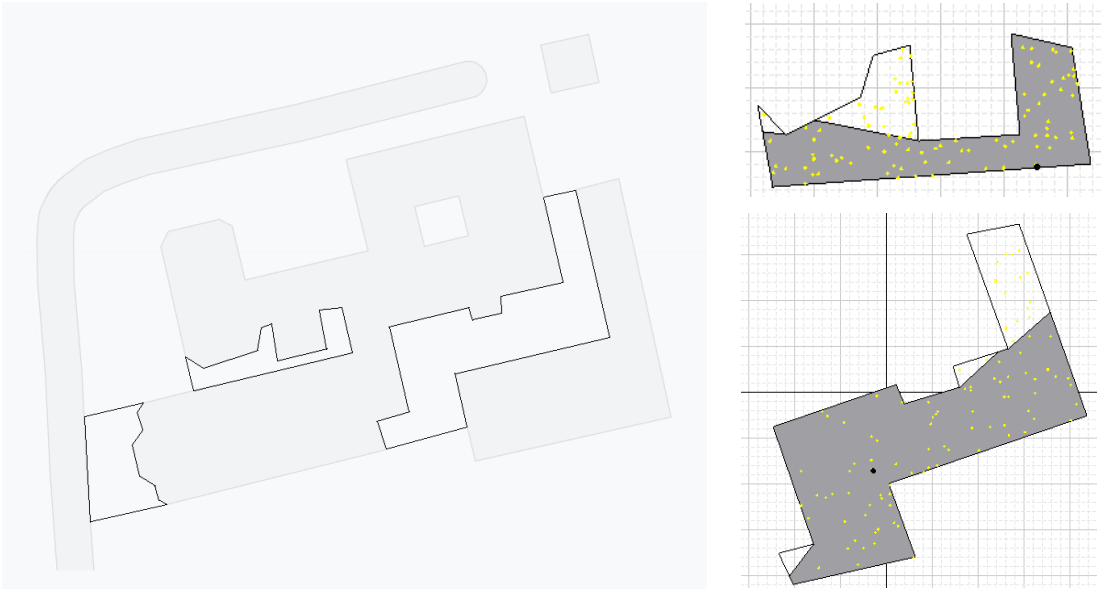

Figura 15: Ejemplo de iluminación en la antigua sede de Vocento y el Diario ABC (fuente Google Maps)

4.1.1.3 Iluminación en navegación marítima

El último caso de uso particular en iluminación que se trata en esta sección consiste en la localización de un faro en el litoral de la costa. Observando la figura posterior (Figura 16), la parte superior del plano corresponde con el mar y la parte inferior dibuja la línea de costa. De esta forma es posible calcular polígonos de visibilidad desde distintos puntos del

trazo de la costa y obtener el área de iluminación que tendrían los faros en cada una de las posiciones. Estos polígonos corresponden con una sección del plano de la costa pero de manera análoga se podría ir obteniendo el dibujo de la costa completo y calcular la unión de los polígonos de visibilidad resultantes de cada una de las secciones.

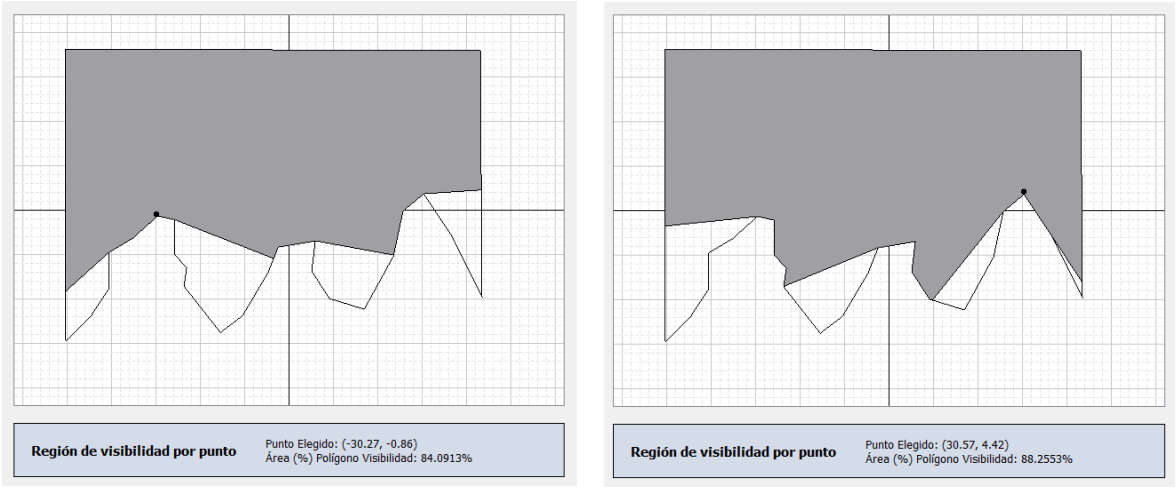

Figura 16: Ejemplo de iluminación de un faro en una sección de un plano de la costa

4.1.2 POLÍGONOS DE VISIBILIDAD Y SEGURIDAD

Otro de los notables casos de aplicación relacionados con los problemas de visibilidad se halla en el ámbito de la seguridad. La vigilancia tanto si es realizada por una persona como por un dispositivo de seguridad es semejante a la iluminación, ya que se trata de obtener la visión de un área concreta de un polígono desde un punto específico. A continuación, se exponen tres casos de uso particulares en materia de seguridad.

4.1.2.1 Sensores de movimiento en espacios interiores

En el mercado se encuentran disponibles diferentes tipos de tecnologías para detectar la presencia de movimiento. Los más conocidos son los sensores ultrasónicos, los sensores por infrarrojos y los sensores duales, que combinan las dos tecnologías anteriores. Estos dispositivos electrónicos se utilizan para poner en funcionamiento un sistema cuando se detecta una presencia previamente determinada y tiene aplicaciones no solamente en

cuestiones de seguridad para activar una cámara de vigilancia sino en iluminación, sistemas de ventilación y otras áreas.

Un sensor de movimiento al igual que un sistema lumínico tiene una zona de cobertura determinada, que se puede optimizar en función de la correcta localización del dispositivo en el espacio interior en cuestión. En este caso la aplicación tiene una utilidad directa ya que no es común la instalación de varios sensores para cubrir toda la zona, por lo que es interesante poder maximizar el área de cobertura.

4.1.2.2 Cámaras de seguridad en recintos exteriores

Este segundo caso de uso posiblemente sea el más semejante con respecto a la utilidad de la optimización de los polígonos de visibilidad en puntos interiores a un polígono en iluminación. Como se comentaba en la sección anterior de iluminación en recintos exteriores y observando la construcción de polígonos de un espacio exterior como en la Figura 15, se encuentra un uso prácticamente similar para la maximización del área de visibilidad de una cámara exterior 360º.

A modo de ejemplo, en la siguiente figura se observa la maximización del área de visibilidad de una cámara de vigilancia en el recinto exterior de un banco.

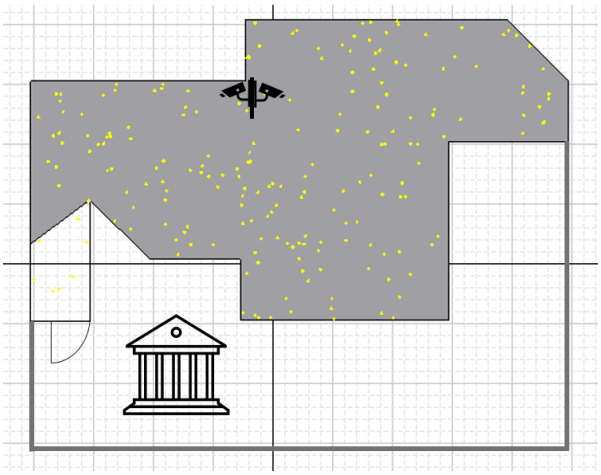

Figura 17: Ejemplo de maximización del área de vigilancia del recinto exterior de un banco

De nuevo, es importante tener claro que no se están teniendo en cuenta ciertas características de alcance y otras limitaciones que se pueden encontrar para cada caso de uso particular.

4.1.2.3 Análisis de seguridad y descubrimiento de puntos ciegos

Por último, en el ámbito de seguridad se halla una utilidad en el uso de las técnicas de construcción de polígonos de visibilidad y unión de múltiples polígonos de visibilidad, para el análisis de la seguridad de espacios tanto interiores como exteriores. Se puede dar el caso de que exista una instalación previa de equipos de seguridad tanto electrónicos como de puntos de vigilancia de personas pero se desconozca realmente cuales son las zonas ciegas y se pretenda realizar una mejora en cuestión de seguridad.

En la siguiente figura se muestra un ejemplo de la unión de tres polígonos de visibilidad, dados tres puntos internos de vigilancia de un recinto poligonal, mostrando en amarillo las zonas que no están adecuadamente cubiertas.

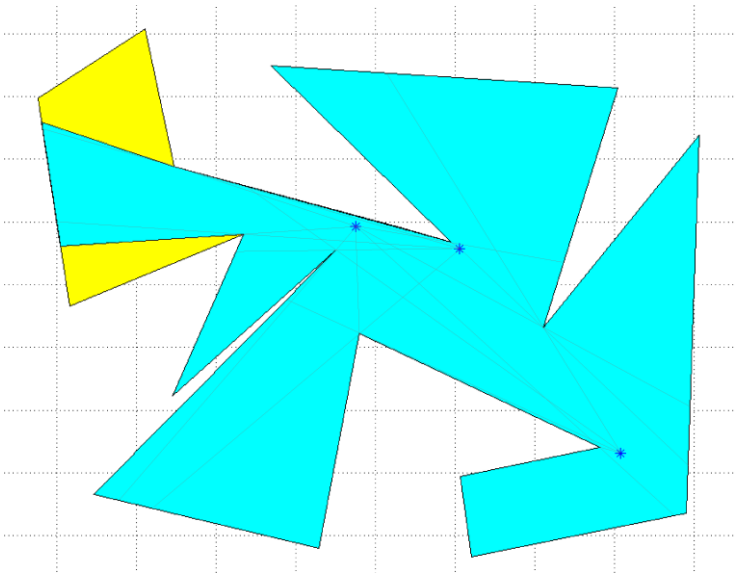

Figura 18: Ejemplo de análisis de zona de seguridad de un recinto [7]

4.1.3 OTRAS APLICACIONES

En esta última sección se va a hacer referencia a otras dos aplicaciones interesantes que puede tener el estudio de los polígonos de visibilidad.

Primeramente, en la comunicación con antenas direccionales donde se requiere de una visibilidad directa, para minimizar el número de antenas intermedias entre el origen y el destino es interesante dibujar los polígonos de visibilidad desde las antenas para observar las intersecciones y los posibles puntos de localización de dichas antenas intermedias, y de esta forma, poder evitar los obstáculos y conseguir buena comunicación sin interferencias (ver Figura 19 a modo de ejemplo básico).

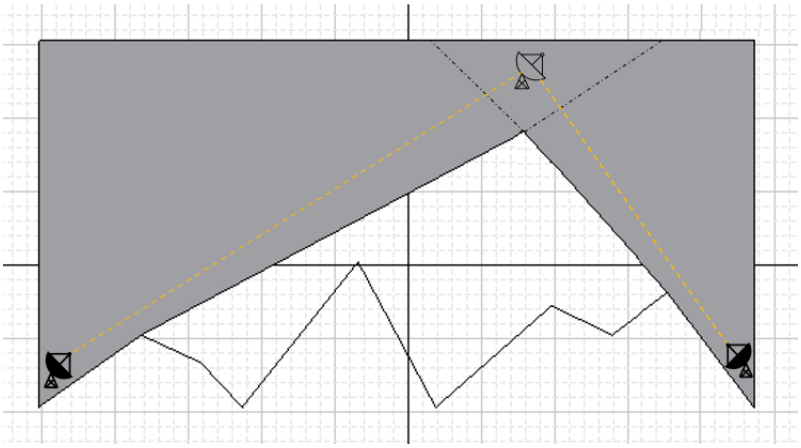

Figura 19: Ejemplo de comunicación con antenas direccionales

En segundo lugar, la aplicación creada puede ser de utilidad en el ámbito formativo en Geometría Computacional, para explicar conceptos y técnicas matemáticas en materia de visibilidad de polígonos, como se utilizará en las explicaciones de la nueva asignatura "Geometría Computacional" del nuevo grado iMAT de ICAI.

4.2 OBJETIVOS

Además del objetivo principal del proyecto, que es la realización de un software interactivo que permita aplicar métodos heurísticos y metaheurísticas para la resolución de problemas en el campo de la Visibilidad dentro de la Geometría Computacional, se encuentran los siguientes objetivos:

- Aprendizaje de métodos heurísticos y metaheurísticas para la resolución de diferentes problemas de optimización.
- Implementación de los algoritmos Random Search, Simulated Annealing y Genético para el caso de estudio.
- Entendimiento de las aplicaciones para la vida real conocidas las soluciones de ciertos problemas de visibilidad.
- Utilización del lenguaje de programación Python, englobando librerías y otras herramientas de ayuda, para el estudio de la Geometría.

4.3 METODOLOGÍA

Durante el transcurso de este proyecto, se llevó a cabo una colaboración estrecha con el profesor Santiago Canales Cano, director del proyecto. Se llevaron a cabo sesiones periódicas para garantizar el cumplimiento efectivo de los objetivos acordados al inicio del proyecto, y para definir los próximos pasos a seguir. El desarrollo del proyecto se ha dividido en las 7 etapas siguientes, que a su vez se engloban en tres grandes fases:

I. Investigación sobre los problemas de visibilidad enmarcados en la Geometría Computacional, así como de las técnicas y métodos para su resolución.
II. Diseño de la interfaz de usuario, teniendo en cuenta las funcionalidades que la aplicación debe soportar y que fueron definidas al inicio del proyecto.

III. Programación del algoritmo para calcular el polígono de visibilidad dado un punto interior a un polígono, así como la funcionalidad para que el usuario pueda dibujar el polígono y el punto en cuestión para trabajar el problema de visibilidad.

IV. Programación de los métodos heurísticos y metaheurísticas para la resolución del problema principal del proyecto consistente en maximizar el área del polígono de visibilidad definido un problema poligonal determinado.

V. Implementación de funcionalidades adicionales en la aplicación, para mejorar la experiencia del usuario al trabajar con diferentes problemas de visibilidad de polígonos.

VI. Realización de un estudio comparativo de los algoritmos implementados para resolver el problema en cuestión, y análisis de los resultados obtenidos.

VII. Elaboración de la memoria del proyecto y preparación para su presentación y defensa.

4.4 PLANIFICACIÓN

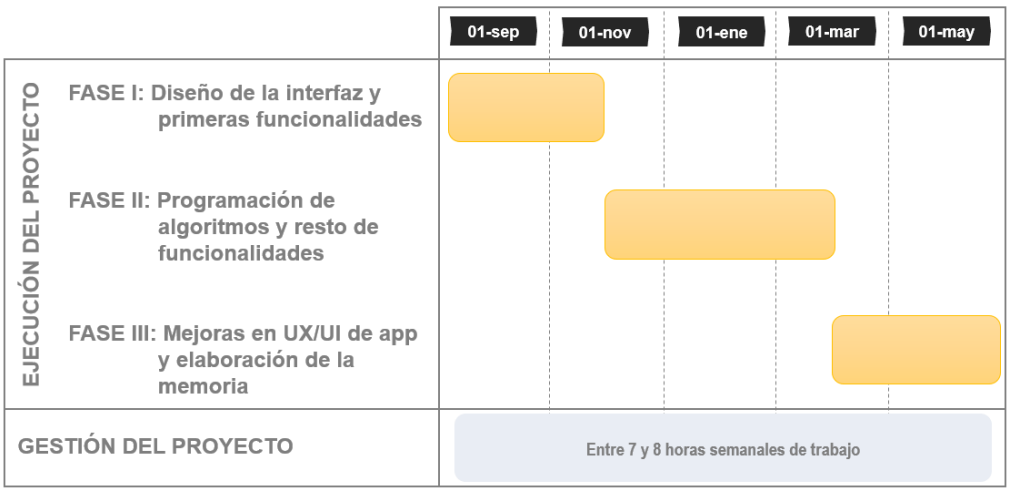

Figura 20: Cronograma de la elaboración de la aplicación del proyecto

La realización de la aplicación del proyecto se ha dividido en tres fases generales que se muestran en el cronograma anterior:

- Fase I: Desde la asignación del Trabajo de Fin de Grado, a principios del mes de septiembre, se ha llevado a cabo el diseño de la interfaz de la aplicación y la implementación de sus primeras funcionalidades. Dichas funcionalidades comprenden la capacidad de realizar dibujos interactivos de polígonos de *n* vértices, así como el cálculo del polígono de visibilidad en relación a un punto interior seleccionado previamente por el usuario. Este proceso de desarrollo ha transcurrido hasta finales del mes de noviembre.

- Fase II: Durante el periodo comprendido entre principios de diciembre y finales de marzo, se realiza la programación de los primeros algoritmos metaheurísticos mencionados previamente (RS y SA), así como otras características necesarias para garantizar una interacción adecuada entre la aplicación y el usuario.

- Fase III: Desde finales de marzo hasta la fecha de presentación, se concluye el desarrollo de la aplicación en su totalidad, incorporando todas sus características funcionales y mejorando la experiencia de usuario mediante una aplicación robusta. Asimismo, se lleva a cabo la implementación de la última técnica metaheurística. Para terminar, se lleva a cabo un estudio comparativo de las técnicas heurísticas empleadas mediante la consideración de factores tales como el porcentaje de área de visibilidad y el tiempo de procesamiento, realizándose la redacción de la memoria del proyecto.

Capítulo 5. DISEÑO DE LA APLICACIÓN

En este capítulo se explica el diseño y las funcionalidades de la aplicación TEHEMAXRVP, de forma que sirva también como guía para su uso en la investigación de la visibilidad de polígonos.

El interfaz de usuario (Figura 21) consta de una barra de menús con cuatro menús funcionales que se irán detallando a lo largo de esta sección: Dibujar, Opciones, Calcular y Algoritmos. Por otro lado, la aplicación se compone de una escena gráfica (en blanco) que es donde se trabajará con los polígonos y donde se mostrarán los polígonos de visibilidad calculados por los distintos algoritmos y demás funcionalidades gráficas. A su vez, el *frame* inferior (en azul) recogerá los resultados de los algoritmos que se hayan implementado para el problema poligonal en cuestión, mostrando el nombre del algoritmo, el punto óptimo encontrado, el porcentaje de área de visibilidad logrado y el tiempo de procesamiento en segundos. Por último, la interfaz consta de un botón (en verde) en la parte inferior derecha para borrar los elementos de la escena gráfica y poder comenzar con el diseño de otro nuevo problema.

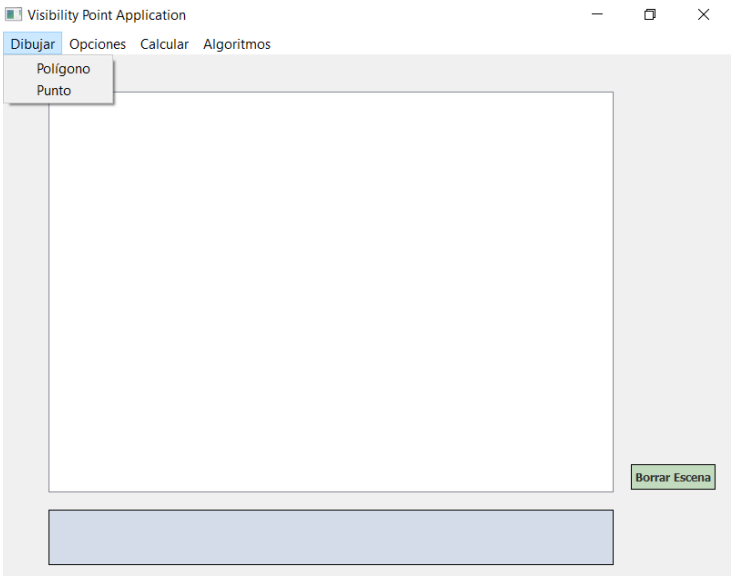

Figura 21: Menú Dibujar de TEHEMAXRVP

El primer menú (Figura 21), Dibujar, implementa la funcionalidad de dibujar un polígono en la escena gráfica, de forma que el usuario escoge el número de vértices que tendrá el polígono (Figura 22) para continuar seleccionando con un *click* donde van a ir localizados dichos vértices y formar el polígono deseado. El polígono se cierra de forma automática con la selección del último vértice, de manera que el primer y último vértice constituirán el último lado del polígono, como se puede observar en la Figura 23.

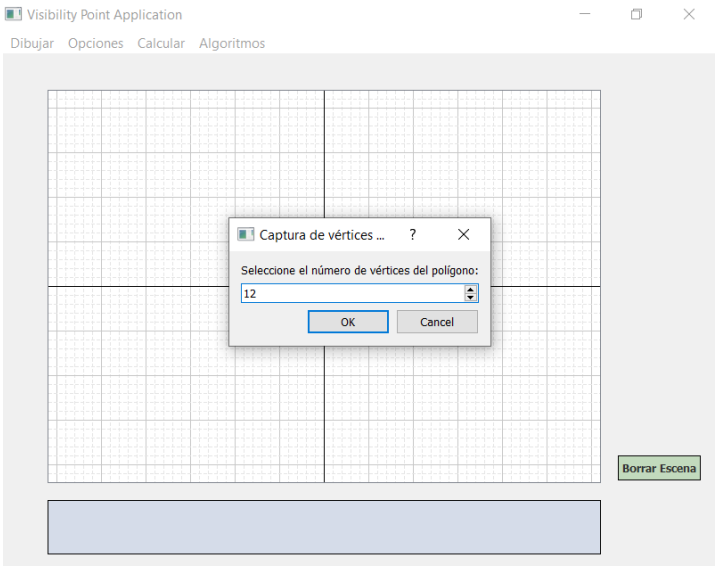

Figura 22: Acción Dibujar Polígono (selección del número de vértices) de TEHEMAXRVP

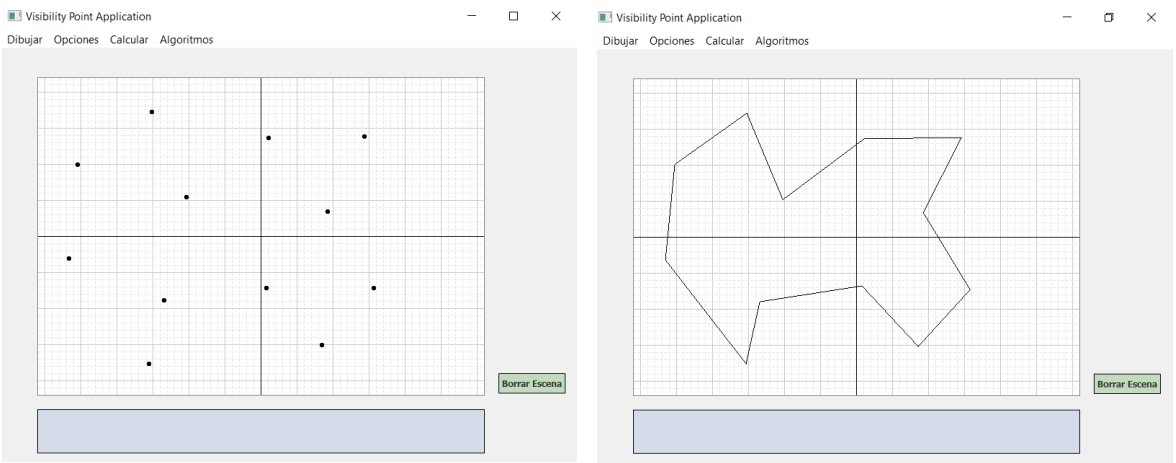

Figura 23: Acción Dibujar Polígono de TEHEMAXRVP

La segunda funcionalidad de este primer menú consiste en dibujar un punto interior en el polígono (Figura 24), con el principal objetivo de poder calcular el polígono de visibilidad que se genera sobre el polígono dado el punto elegido. Como se puede observar en la Figura 25 la región de visibilidad se dibuja en un tono grisáceo sobre el polígono y en el frame inferior se muestran los resultados del punto elegido en coordenadas cartesianas en función del eje de coordenadas (el punto (0,0) se corresponde con la intersección de los ejes x e y) que aparece dibujado por pantalla (opción Dibujar Eje del menú Opciones, Figura 26) y el porcentaje de área de visibilidad obtenido por dicho punto.

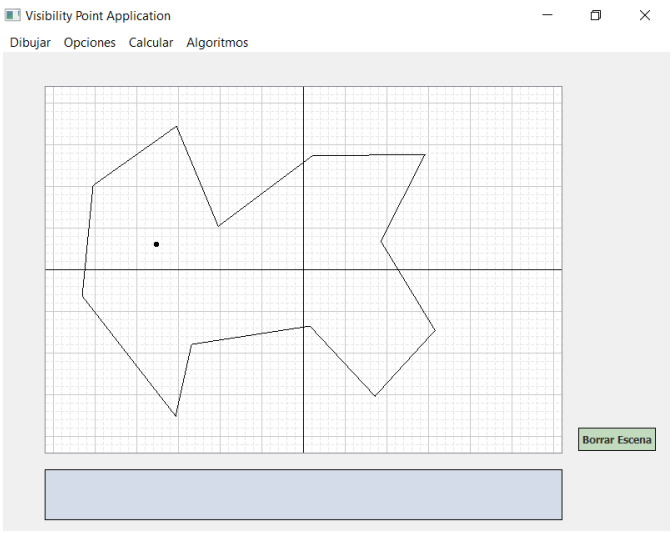

Figura 24: Acción Dibujar Punto de TEHEMAXRVP

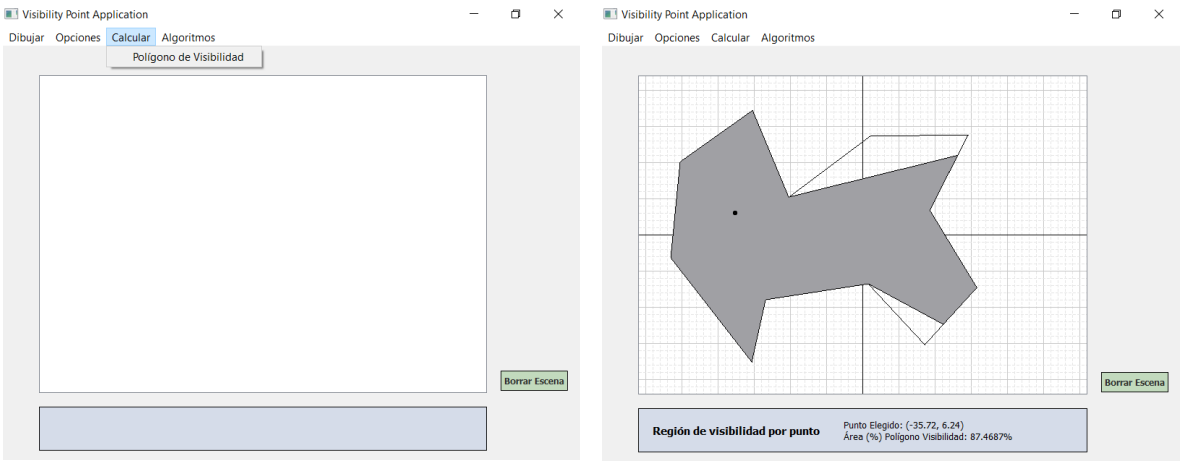

Figura 25: Acción Calcular Polígono de Visibilidad del Menú Calcular de TEHEMAXRVP

A continuación, las funcionalidades que otorga el menú Opciones (Figura 26) son, por un lado, la impresión del eje de coordenadas y el *grid* en la escena gráfica, y por otro lado, la representación de las coordenadas cartesianas de los vértices del polígono previamente seleccionado (Figura 27).

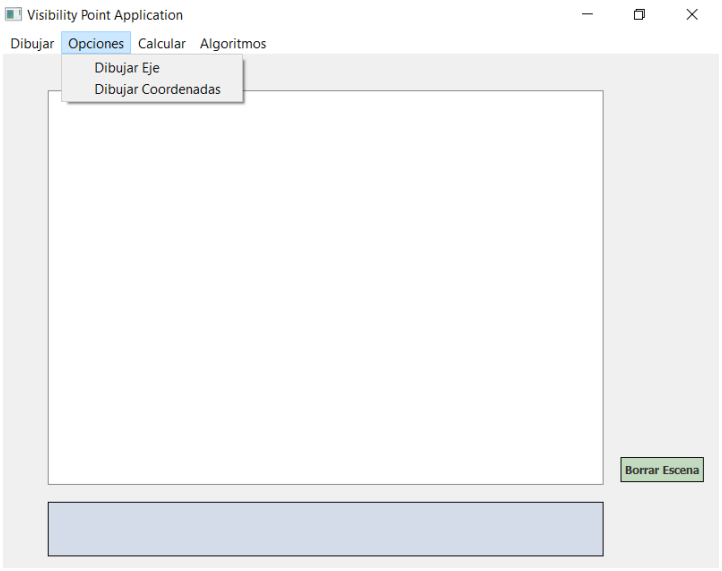

Figura 26: Menú Opciones de TEHEMAXRV

La función programada que transforma las coordenadas de pantalla a coordenadas cartesianas (ver función *convertirCoordenadas* en el Anexo II) se ha implementado teniendo en cuenta el ancho y alto en píxeles de la escena gráfica y las dimensiones deseadas para los ejes cartesianos, siendo 124 mm el ancho y 88 mm el largo. De esta forma cada pequeño cuadro tendría unas dimensiones de 2x2 mm. El cálculo empleado para calcular las coordenadas x e y son las siguientes:

$$centro\_x = ancho\_pantalla / 2$$
$$centro\_y = alto\_pantalla / 2$$
$$dif\_x = x\_pantalla - centro\_x$$
$$dif\_y = centro\_y - y\_pantalla$$
$$x\_cartesiano = ancho\_cartesiano * dif\_x / ancho\_pantalla$$
$$y\_cartesiano = alto\_cartesiano * dif\_y / alto\_pantalla$$

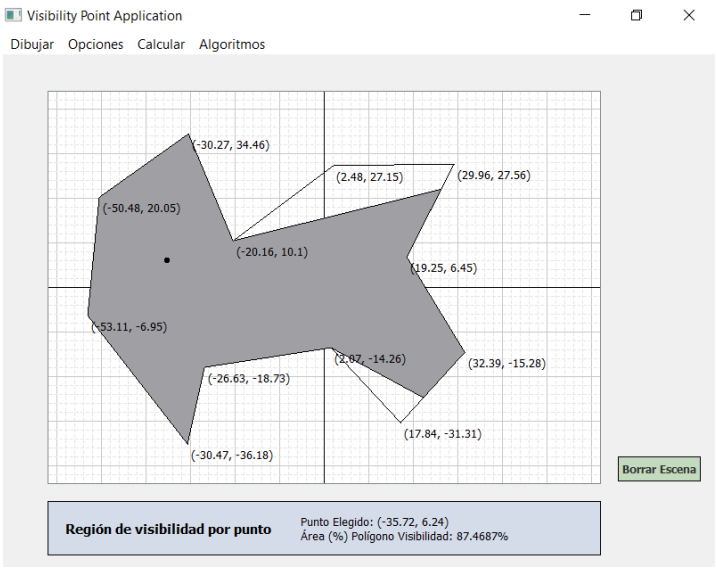

Figura 27: Acción Dibujar Coordenadas de TEHEMAXRVP

Por último, el menú Algoritmos (Figura 28) recoge los tres algoritmos implementados (RS, SA y GA) para la optimización del polígono de visibilidad dado un punto. Una vez definido el polígono se puede seleccionar cualquiera de los algoritmos, tantas veces como se quiera, para obtener el punto que maximiza el área de la región de visibilidad. Para el algoritmo RS (Figura 29) se selecciona el número de puntos que el algoritmo va a analizar.

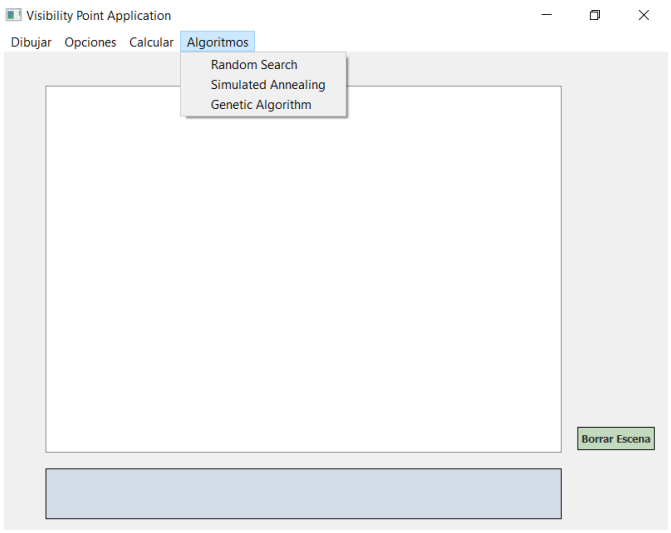

Figura 28: Menú Algoritmos de TEHEMAXRVP

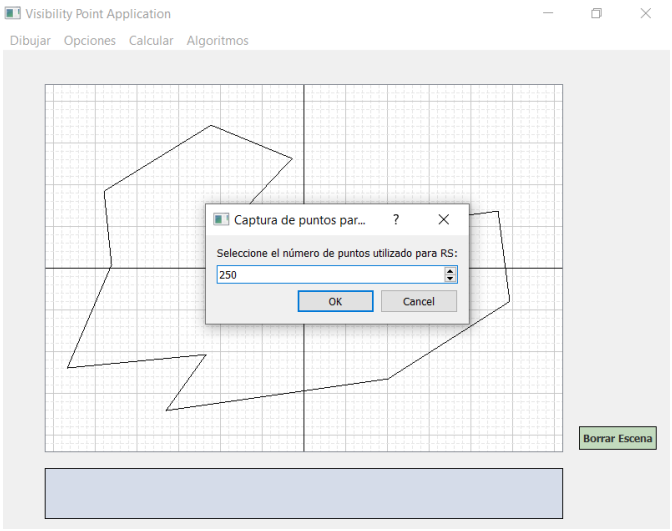

Figura 29: Acción Algoritmo RS (selección del número de puntos) de TEHEMAXRVP

Capítulo 6. ANÁLISIS DE RESULTADOS

Para finalizar con el proyecto se realiza un estudio comparativo de las tres técnicas metaheurísticas utilizadas para la maximización del polígono de visibilidad dado un punto interior a un polígono.

El estudio se ha centrado en el análisis de los tiempos de procesamiento de los algoritmos y el porcentaje de visibilidad obtenido al utilizar las técnicas. Las tablas mostradas a continuación recogen los resultados del estudio, indicando los algoritmos utilizados:

- Random Search $m = 250\ puntos$
- Random Search $m = 500\ puntos$
- Simulated Annealing $T_0 = \#vértices \quad T(k) = \frac{T_0}{1+k} \quad T_f = 0.03 \quad N(T) = \frac{1}{T}$
- Algoritmo Genético $p_m = 0.5$

y por otro lado, el tiempo medio de procesamiento medida en segundos, la desviación estándar de los tiempos de procesamiento [s], el porcentaje del área de visibilidad medio y la desviación estándar de los porcentajes del área de visibilidad.

Se ha llevado a cabo un análisis sobre 25 polígonos seleccionados aleatoriamente con 15, 25 y 35 vértices, respectivamente. Cada tabla del informe recopila los resultados correspondientes a cada uno de los diferentes conteos de vértices. Los resultados individuales para cada uno de los polígonos se pueden visualizar en el Anexo III.

En la Tabla 9, se recogen los resultados medios para 25 polígonos aleatorios de 15 vértices aplicando los diferentes algoritmos. Como se puede observar, los tiempos medios de procesamiento para el algoritmo RS son muy inferiores a los tiempos de los algoritmos SA y GA, destacando que el tiempo y el valor de *m* para el algoritmo RS se relacionan prácticamente de una manera lineal y positiva, es decir, al aumentar al doble el valor de *m* el tiempo de procesamiento aumenta aproximadamente también el doble. Por otro lado, cabe remarcar que la desviación estándar de los tiempos de procesamiento para el

algoritmo SA en relación con el tiempo medio, es bastante superior al obtenido para GA, lo que muestra que el tiempo medio al aplicar el algoritmo SA para cada polígono varía en mayor proporción que en el caso de GA.

En lo relativo al porcentaje del área de visibilidad medio obtenido al aplicar cada técnica, se concluye que SA arroja los mejores resultados, seguido de los dos casos de RS y GA. En todos los casos el porcentaje supera el 90% y aunque la técnica SA revela los mejores resultados, a nivel de eficiencia en relación con el tiempo parece que RS tiene el mejor rendimiento. Probablemente, el aumento del valor de *m* para RS consiga proporcionar resultados aún más favorables, acercándose al arrojado por el algoritmo SA. La técnica GA muestra los peores resultados respecto del tiempo y porcentaje de área iluminado.

Algoritmos	*Random Search 250 puntos*	*Random Search 500 puntos*	*Simulated Annealing*	*Algoritmo Genético*
Tiempo medio de Procesamiento [s]	0,2581	0,4596	27,7044	47,1762
Desviación Estándar del Tiempo de Procesamiento [s]	0,0327	0,0319	5,7042	2,7970
% Área Media de Visibilidad	93,7075	93,7788	**93,9165**	93,6056
% Desviación Estándar del Área de Visibilidad	4,6065	4,6099	4,6577	4,7615

Tabla 9. Tabla resumen del estudio comparativo de algoritmos (25 polígonos con 15 vértices)

La Tabla 10 recoge los resultados medios para 25 polígonos aleatorios de 25 vértices aplicando los diferentes algoritmos.

Algoritmos	*Random Search 250 puntos*	*Random Search 500 puntos*	*Simulated Annealing*	*Algoritmo Genético*
Tiempo medio de Procesamiento [s]	0,3402	0,6613	82,1757	203,9028
Desviación Estándar del Tiempo de Procesamiento [s]	0,0280	0,0463	16,8701	36,5793
% Área Media de Visibilidad	84,2594	84,4207	**84,6581**	84,5508
% Desviación Estándar del Área de Visibilidad	10,5628	10,5241	10,8109	10,4457

Tabla 10. Tabla resumen del estudio comparativo de algoritmos (25 polígonos con 25 vértices)

Las conclusiones aportadas para el caso anterior son prácticamente similares para este caso, aunque se diferencian en ciertos aspectos. Por un lado, el algoritmo SA arroja de nuevo el mejor resultado respecto del porcentaje del área de visibilidad medio pero igualmente la diferencia en comparación con las demás técnicas es pequeña. El Algoritmo Genético se sitúa en segundo lugar en cuanto al porcentaje de área iluminada, pero de nuevo es el algoritmo que más tiempo de procesamiento requiere distanciándose en mayor proporción de las otras técnicas.

Por otro lado, en lo relativo a la desviación estándar del tiempo de procesamiento respecto de las técnicas SA y GA, parece que el resultado es más coherente aunque de nuevo observando los tiempos de procesamiento medio, el algoritmo SA presenta una mayor dispersión en cuanto al tiempo para los diferentes polígonos analizados.

Respecto a la desviación estándar del porcentaje de área de visibilidad los resultados parecen ser normales, de igual modo que ocurría en el caso anteriormente analizado.

Por último, la Tabla 11 recoge los resultados medios para 25 polígonos aleatorios de 35 vértices aplicando los diferentes algoritmos.

Algoritmos	*Random Search 250 puntos*	*Random Search 500 puntos*	*Simulated Annealing*	*Algoritmo Genético*
Tiempo medio de Procesamiento [s]	0,5143	0,9909	127,1600	478,0581
Desviación Estándar del Tiempo de Procesamiento [s]	0,0365	0,0803	18,7744	26,9519
% Área Media de Visibilidad	88,1116	88,2363	**88,3595**	88,3071
% Desviación Estándar del Área de Visibilidad	5,7894	5,8144	5,7676	5,7929

Tabla 11. Tabla resumen del estudio comparativo de algoritmos (25 polígonos con 35 vértices)

De nuevo, los resultados arrojados son bastante similares a los mostrados en los casos anteriores. El algoritmo SA proporciona el mejor resultado en cuanto al porcentaje de área de visibilidad medio, seguido de GA y de RS, aunque la diferencia es mínima. Lo único destacable respecto a esta variable es que se ha logrado un mayor porcentaje de visibilidad en comparación con en el caso anterior, siendo un polígono que aparentemente tiene más complejidad puesto que el número de vértices es mayor (ver Figura 30). Esta casuística puede ser debida a la aleatoriedad con la que se han generado los diferentes polígonos analizados, aunque sería interesante profundizar más.

Por otro lado, el tiempo de procesamiento medio para el algoritmo RS parece seguir un crecimiento más lineal y con menor tasa de cambio que en el caso del Algoritmo Genético, que parece seguir un crecimiento más exponencial, con respecto al aumento en el número de vértices de los polígonos analizados. Esta cuestión se analizará en detalle más adelante. Por último, la desviación estándar del tiempo de procesamiento ha disminuido con respecto al caso anterior, a pesar de que ha aumentado el tiempo de procesamiento medio, por lo que se puede decir a priori que no tienen una relación cuantificable con facilidad.

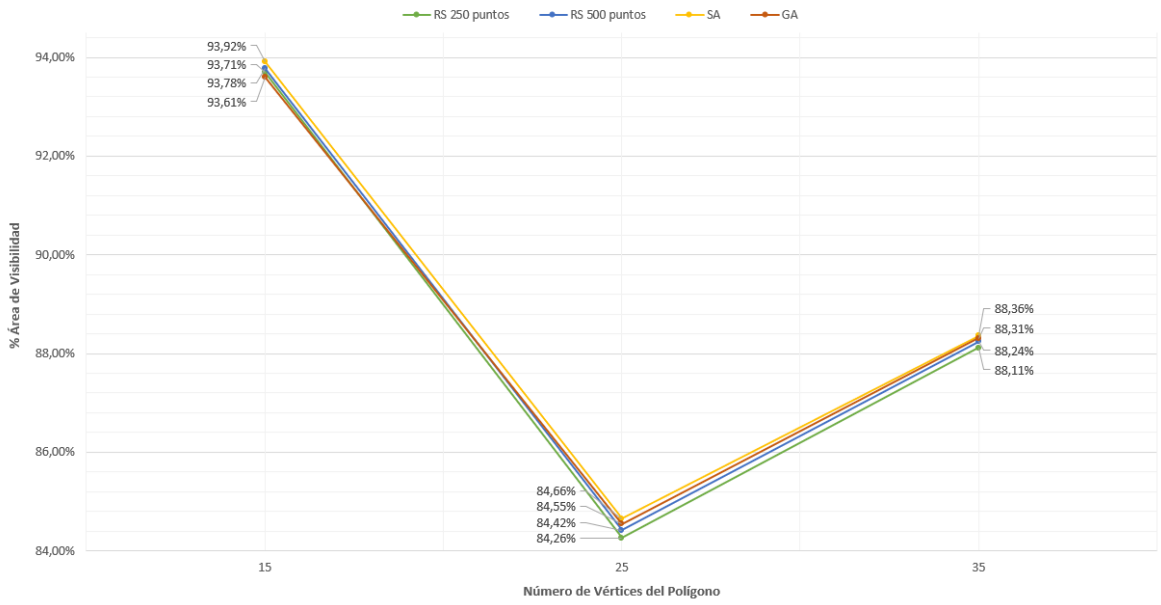

Figura 30: Gráfica Resumen del % de Área de Visibilidad obtenido para los Algoritmos

En resumen, el algoritmo que mejores resultados proporciona para la maximización del área de visibilidad respecto a un punto interior a un polígono *P* de *n* vértices es Simulated Annealing, pero el algoritmo que aparentemente es más eficiente teniendo en cuenta el tiempo de procesamiento es Random Search. El Algoritmo Genético proporciona resultados favorables, muy cercanos a la técnica SA en algunos casos, pero su tiempo de procesamiento parece demasiado elevado y aumenta en mayor proporción con el aumento de la dificultad del problema.

A continuación, como se puede observar en la Figura 31, se analizarán los tiempos de procesamiento medios de las diferentes técnicas empleadas para los distintos números de vértices de los polígonos estudiados.

En este análisis se quiere determinar qué tipo de relación existe entre el aumento del número de vértices, que se relaciona en un principio con la dificultad del problema a resolver, y el tiempo de procesamiento para cada uno de los algoritmos.

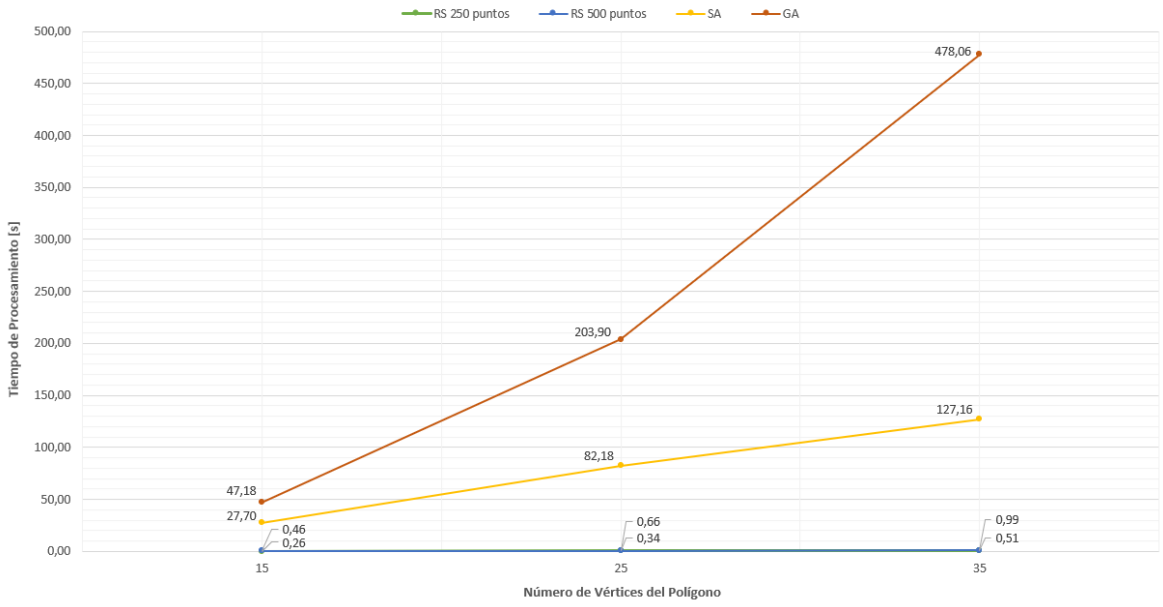

Figura 31: Gráfica Resumen del Tiempo de Procesamiento [s] obtenido para los Algoritmos

Los tiempos de procesamiento de los algoritmos no presentan colinealidad, por lo que no se puede hablar en términos de pendiente de una sola recta, pero se puede analizar la tasa de cambio porcentual entre cada dos tiempos. En la Figura 32 se muestra el incremento porcentual en relación con el aumento del número de vértices de los polígonos estudiados, para las diferentes técnicas metaheurísticas. Las barras representan el porcentaje de aumento del tiempo respecto del caso de estudio anterior, con lo que se marca en 0% el caso de estudio con polígonos de 15 vértices (en azul), ya que se utiliza como punto de partida y no se compara con otro caso de estudio más sencillo.

A pesar de no poder sacar conclusiones fiables que relacionen el tiempo de procesamiento medio de los algoritmos y el número de vértices que tienen los polígonos analizados, ya que se necesitaría analizar una mayor cantidad de datos, como consecuencia de la gráfica posterior se puede obtener cierto aprendizaje en esta cuestión.

El algoritmo RS para los dos valores de *m* estudiados presenta un crecimiento más lineal, sin serlo realmente, del tiempo de procesamiento en relación con el número de vértices de los polígonos, a la vez que el crecimiento porcentual es menor que en el caso de las otras dos técnicas. El crecimiento porcentual en el caso del algoritmo GA es superior al del algoritmo SA, teniendo en cuenta que la tasa de cambio entre el caso de polígonos con 15 vértices y 25 vértices es mayor que la tasa de cambio entre el caso de polígonos de 25 vértices y 35 vértices. Lo que se resume en que el algoritmo GA, además de ser el más lento en ejecución, es el que presenta mayor aumento en el tiempo de procesamiento en relación con el aumento del número de vértices, es decir, que aparentemente cuanto mayor dificultad tenga el problema a resolver más tiempo en proporción va a tardar en generar una solución, seguido del algoritmo SA y del RS.

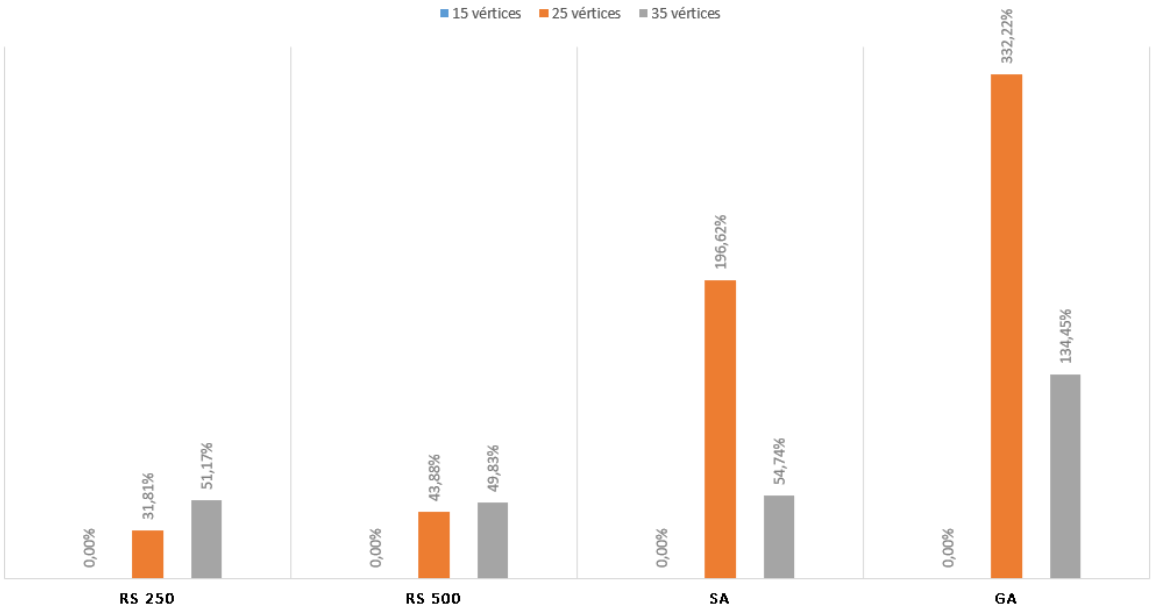

Figura 32: Tasa de cambio porcentual del tiempo de procesamiento medio de los algoritmos en relación con el aumento del número de vértices de los polígonos

Capítulo 7. CONCLUSIONES Y TRABAJOS FUTUROS

El propósito principal de este proyecto era desarrollar un software interactivo que permitiera resolver el problema MaxA-p-Pv1(*P*), que consistía en maximizar el polígono de visibilidad dado un punto interior a un polígono *P* de *n* vértices. La aplicación proporcionada permite establecer un problema poligonal y aplicar tres técnicas metaheurísticas, Random Search, Simulated Annealing y el Algoritmo Genético, para encontrar aquel punto interior a *P* que ilumina una mayor área. Además, la aplicación implementa otras funcionalidades para mejorar la resolución del problema mencionado.

Los algoritmos heurísticos generales mencionados se han adaptado para ofrecer una solución óptima o aproximada a este problema. En lo relativo al estudio comparativo del funcionamiento de estos algoritmos, se puede concluir que los tres métodos proporcionan soluciones muy similares, pero la principal diferencia reside en el tiempo de procesamiento que necesitan. El algoritmo Random Search genera soluciones muy eficientes en cuanto al tiempo de ejecución y área iluminada, pero es el algoritmo Simulated Annealing el que logra obtener las soluciones más óptimas, aunque con un coste de procesamiento muy superior. Por otro lado, aunque el Algoritmo Genético ofrece soluciones muy cercanas al algoritmo Simulated Annealing, presenta unos tiempos de ejecución demasiado elevados, que además crecen en una mayor proporción con la dificultad del problema.

Respecto a los casos de uso encontrados para la optimización de la visibilidad de polígonos se pueden destacar la iluminación de espacios interiores y recintos exteriores, así como la iluminación en navegación marítima. Por otro lado, también destacan los usos en términos de seguridad, como la localización de cámaras de seguridad y sensores de movimiento, así como el análisis de puntos ciegos en espacios interiores y exteriores. Además, en lo relativo a las comunicaciones direccionales el uso de estos métodos puede ser útil para la correcta planificación y reducción de costes. Por último, la aplicación generada puede ser de utilidad en la divulgación de las diferentes técnicas metaheurísticas para la resolución de problemas de visibilidad en el campo de la Geometría Computacional.

Como posibles trabajos futuros se plantea la implementación de más técnicas heurísticas para dar solución al problema MaxA-p-Pv1(*P*), como el Método del Gradiente, que consistiría en movernos siempre en una dirección de "gradiente positivo" respecto al área iluminada para encontrar puntos cada vez mejores, o el algoritmo Ant System, basado en ciertos comportamientos optimizados de las colonias de hormigas.

Como se mencionaba en capítulos anteriores el problema planteado en este proyecto es una simplificación del problema MaxA-p-Pvk(*P*,*k*), donde *k* representa el número de *luces punto* localizadas para encontrar la máxima visibilidad. Además, cabe recordar que ambos problemas se corresponden con el paso previo para dar una solución al problema que se identifica como MinN-p-Pvk(*P*), que pretende minimizar el número de luces que hacen que se ilumine por completo un polígono *P*. Por tanto, para futuros trabajos sería interesante implementar diferentes técnicas o algoritmos para proporcionar soluciones a los problemas MaxA-p-Pvk(*P*,*k*) y MinN-p-Pvk(*P*).

Por otro lado, existe la posibilidad de proponer problemas poligonales con mayor dificultad, pudiendo construir polígonos simples con "agujeros", aprovechando la implementación ya existente del algoritmo de barrido rotacional para el cálculo de los polígonos de visibilidad. En lo referente al algoritmo empleado para calcular los polígonos de visibilidad se puede utilizar técnicas más eficientes con costes computacionales menores, para conseguir de esta forma disminuir el tiempo de procesamiento de las técnicas heurísticas, que necesitan calcular en múltiples ocasiones diferentes polígonos de visibilidad.

Por último, se podrían implementar más funcionalidades para mejorar el uso de la aplicación como por ejemplo, un menú para editar, guardar e insertar polígonos. Además, sería interesante poder limitar el alcance y el grado de iluminación de los diferentes puntos interiores al polígono, consiguiendo de esta forma adaptarse a casos más reales.

Capítulo 8. BIBLIOGRAFÍA

[1] Urrutia, J. *"Art Gallery and Illumination Problems"* en "Handbook on Computational Geometry". Elsevier (J. R. Sack and J. Urrutia ed.), 1999.

[2] Chvátal, V. *"A Combinatorial Theorem in Plane Geometry"*. Journal of Combinatorial Theory, Serie B, 18, pp. 39-41, 1975.

[3] Ibañez, R. *"Teorema de la Galería de Arte"*. Cuaderno de Cultura Científica, Matemoción. https://culturacientifica.com/2014/01/29/teorema-de-la-galeria-de-arte/

[4] Lee D.T.; Lin A.K. *"Computational complexity of art gallery problem"*. IEEE Trans. Info. Th. IT-32, pp. 415-421, 1979.

[5] Garey, M.R.; Johnson, D.S. *"Computers and intractability. A guide to the theory of NP-completeness"*. W.H. Freeman and Company 1979.

[6] Osman, I.H.; Kelly, J.P. *"Meta-Heuristics: Theory & Applications"*. Kluwer Academic Publishers, 1996.

[7] Canales Cano, Santiago. *"Métodos heurísticos en problemas geométricos. Visibilidad, iluminación y vigilancia"*. Tesis doctoral, 2004.

[8] Riverbank Computing PyQt. https://www.riverbankcomputing.com/software/pyqt/

[9] Qt Designer Manual de Qt Documentation. https://doc.qt.io/qt-6/qtdesigner-manual.html

[10] T. Asano. *"An efficient algorithm for finding the visibility polygon for a polygonal region with holes"*. IEICE Transactions (1976-1990), (9):557–559, 1985.

[11] CGAL 5.5.2 – 2D Visibility. https://doc.cgal.org/latest/Visibility_2/index.html

[12] Foley, Van Dam, Feiner, Hughes, Phillips. *"Introduction to Computer Graphics"*. Ed. 1ª, Addison-Wesley, 1994.

[13] Hearn, Baker. *"Gráfica por Computadora"*. Ed 2ª, Prentice-Hall, 1995, (corresponde a Ed 2ª, inglés, 1994).

[14] Saastn. *"How to find visibility of a polygon from a vertex v in a concave polygon"*. https://stackoverflow.com/questions/58755784/how-to-find-visibility-of-a-polygon-from-a-vertex-v-in-a-concave-polygon

[15] Suarez, O. A. "*Una aproximación a la heurística y metaheurística"*. Universidad Antonio Nariño, 2011.

ANEXO I: ALINEACIÓN DEL PROYECTO CON LOS ODS

El proyecto llevado a cabo trata en general sobre el estudio de la optimización del número de luces que hacen que un polígono se ilumine por completo y en particular sobre la maximización del polígono de visibilidad dado un punto interior a un polígono simple sin “agujeros”. A pesar de que la aplicación desarrollada constituye un punto de partida para investigar acerca de los problemas de visibilidad, la implementación de los algoritmos que proporcionan una solución eficiente al problema de optimización, pueden ser usados para alcanzar soluciones en múltiples áreas del ámbito real.

Habiendo investigado acerca de los usos que el estudio de los problemas de visibilidad pueden influir positivamente en diferentes campos, el proyecto se encuentra alineado con dos Objetivos de Desarrollo Sostenible pertenecientes a la Agenda 2030.

Por un lado, los algoritmos implementados tratan de optimizar la visibilidad desde el interior de un polígono en 2D. Estos resultados proporcionan un uso más eficiente de la energía eléctrica y otros recursos tanto en aplicaciones de infraestructura lumínica como de vigilancia, además de complementar y mejorar otros campos de estudio. Por tanto, existe una alineación con el ODS 7 (Energía Asequible y No Contaminante), de manera que se pretende hacer un uso eficiente de los recursos relacionados con la mejora de la visibilidad en diversas ciencias.

Por otro lado, visto desde un punto de vista más amplio, este proyecto esta alineado con el ODS 9 (Industria, Innovación e Infraestructura), siendo la investigación realizada y la aplicación diseñada un desarrollo tecnológico e innovador para optimizar procesos y aumentar la eficiencia en diversas áreas.

ANEXO II: CÓDIGO DE LA APLICACIÓN

TEHEMAXRVP.py

Este archivo hace referencia a la construcción de la ventana principal de la aplicación, así como todos los componentes contenidos en dicha ventana y las funciones conectadas a dichos componentes, además de funciones auxiliares que son de utilidad para el funcionamiento de los algoritmos empleados para calcular los polígonos de visibilidad dado un punto interior al polígono y para otros aplicativos.

```
# Importar librerías
import sys
from PyQt5.QtWidgets import QApplication, QMainWindow, QMessageBox,
QGraphicsView, QGraphicsScene, QInputDialog, QGraphicsTextItem
from PyQt5 import uic
import ctypes
from PyQt5.QtCore import Qt, QPointF, QRectF
from PyQt5.QtGui import QBrush, QPen, QPolygonF, QColor
from shapely.geometry import Point, Polygon
import numpy as np
from time import perf_counter
from Funciones import calculoPoligonoVisibilidad, convertirCoordenadas
import json

# Clase heredada de QMainWindow (Constructor de ventanas)
class MainWindow(QMainWindow, QGraphicsView):
    # Método constructor de la clase
    def __init__(self):
        # Iniciar el objeto padre QMainWindow
        QMainWindow.__init__(self)
        # Cargar la configuración del archivo .ui en el objeto
        uic.loadUi("AppDesign.ui",self)
        # Poner título de la ventana
        self.setWindowTitle("Visibility Point Application")
        # Mostrar la ventana maximizada
        self.showMaximized()
        # Fijar el tamaño de la ventana
        self.setMinimumSize(800,600)
        # Fijar el tamaño máximo
        self.setMaximumSize(800,600)
        # Mover la ventana y centrarla en el escritorio
        resolution=ctypes.windll.user32
        resolution_width=resolution.GetSystemMetrics(0)
        resolution_height=resolution.GetSystemMetrics(1)
        left=(resolution_width/2)-(self.frameSize().width()/2)
```

```
        top=(resolution_height/2)-(self.frameSize().height()/2)
        self.move(left,top)

        # Crear la escena para el GraphicView
        self.crearEscenaGrafico()

        # Conectar los menús a sus funciones
        self.actionPoligono.triggered.connect(self.dibujarPoligono)
        self.actionPunto.triggered.connect(self.dibujarPunto)

self.actionRegionVisibilidad.triggered.connect(self.calcularRegionVisibilidadPunt
o)
        self.clearButton.clicked.connect(self.borrarFrameDibujo)
        self.actionRandom_Search.triggered.connect(self.algoritmoRandomSearch)

self.actionSimulated_Annealing.triggered.connect(self.algoritmoSimulatedAnnealing
)
        self.actionGenetic_Algorithm.triggered.connect(self.algoritmoGenetico)
        self.actionDibujar_Eje.triggered.connect(self.dibujarCuadricula)
        self.actionDibujar_Coordenadas.triggered.connect(self.indicarCoordenadas)

        # Variables
        self.isPunto = False
        self.isPoligono = False
        self.isEjes = False
        self.posicion = None
        self.arrayPoligono = []
        self.cont = 0
        self.savePunto = None
        self.arraySavePoligono = []
        self.arrayPoligonoVisibilidad = []
        self.dic_poligonos={}
        self.cont_poligonos=1

    # Función para calcular el área de un polígono
    def calcularArea(self,qpolygon):
        area = 0
        for i in range(qpolygon.size()):
            p1 = qpolygon[i]
            p2 = qpolygon[(i + 1) % qpolygon.size()]
            d = p1.x() * p2.y() - p2.x() * p1.y()
            area += d
        return abs(area) / 2

    # Función para generar puntos aleatorios contenidos dentro de un polígono
    def puntosAleatorios(self,num_puntos):
        # Construcción del polígono en función de los vértices seleccionados
        polygonRS = Polygon(self.arraySavePoligono)
        # Obtener los límites en x e y del polígono
        minx, miny, maxx, maxy = polygonRS.bounds

        puntos_selec=[]
        cont=0
```

```
        # Obtener coordenadas en x e y aleatorias para generar puntos contenidos
        # en el polígono
        while True:
            x = np.random.randint(minx,maxx,1)+np.random.random()
            y = np.random.randint(miny,maxy,1)+np.random.random()
            punto = list(zip(x,y))
            if polygonRS.contains(Point(punto[0]))==True:
                puntos_selec.append(punto[0])
                cont=cont+1
            if cont==num_puntos:
                break

        # La función devuelve una lista de puntos en función del número
        # de puntos pedidos
        return puntos_selec

    # Función para generar un vecino en el Algoritmo Simulated Annealing
    def generarVecinoSA(self,punto):
        # Construcción del polígono en función de los vértices seleccionados
        polygonRS = Polygon(self.arraySavePoligono)
        # Obtener los límites en x e y del polígono
        minx, miny, maxx, maxy = polygonRS.bounds

        # Obtener un punto vecino en direcciones x e y aleatorias que esté
        # contenido en el polígono. Los vecinos cada vez se encuentran más
        # cerca del punto debido al crecimiento de la variable self.div
        while True:
            x_aux = np.random.randint(minx,maxx,1)+np.random.random()
            y_aux = np.random.randint(miny,maxy,1)+np.random.random()

            int_random=np.random.randint(0,4)

            if int_random==0:
                x=punto[0] + x_aux/self.div
                y=punto[1] + y_aux/self.div
            elif int_random==1:
                x=punto[0] - x_aux/self.div
                y=punto[1] - y_aux/self.div
            elif int_random==2:
                x=punto[0] + x_aux/self.div
                y=punto[1] - y_aux/self.div
            else:
                x=punto[0] - x_aux/self.div
                y=punto[1] + y_aux/self.div

            punto_obtenido = list(zip(x,y))
            if polygonRS.contains(Point(punto_obtenido[0]))==True:
                break

        print('Punto Vecino Encontrado')
        return punto_obtenido

    # Función para mutar un punto en el Algoritmo Genético
```

```
    def mutarPuntoGA(self,punto):
        polygonRS = Polygon(self.arraySavePoligono)
        minx, miny, maxx, maxy = polygonRS.bounds

        # Obtener un punto mutado en direcciones x e y aleatorias que esté
        # contenido en el polígono. Los puntos mutados cada vez se encuentran
        # más cerca del punto devido al crecimiento de la variable self.k
        while True:
            x_aux = np.random.randint(minx,maxx,1)+np.random.random()
            y_aux = np.random.randint(miny,maxy,1)+np.random.random()

            int_random=np.random.randint(0,4)

            if int_random==0:
                x=punto[0] + x_aux/self.k
                y=punto[1] + y_aux/self.k
            elif int_random==1:
                x=punto[0] - x_aux/self.k
                y=punto[1] - y_aux/self.k
            elif int_random==2:
                x=punto[0] + x_aux/self.k
                y=punto[1] - y_aux/self.k
            else:
                x=punto[0] - x_aux/self.k
                y=punto[1] + y_aux/self.k

            punto_obtenido = list(zip(x,y))
            if polygonRS.contains(Point(punto_obtenido[0]))==True:
                break

        print('Punto Mutado Encontrado')
        return punto_obtenido

    # Función que define la escena gráfica y las características de color,
    # anchura y patrón de los elementos que se incluyen
    def crearEscenaGrafico(self):
        self.scene = QGraphicsScene()
        self.scene.setSceneRect(QRectF(self.graphicsView.viewport().rect()))
        self.graphicsView.setScene(self.scene)
        self.penPunto = QPen(Qt.black)
        self.brushPunto = QBrush(Qt.SolidPattern)
        self.penPuntoProbado = QPen(Qt.yellow)
        self.brushPuntoProbado = QBrush(Qt.yellow,Qt.SolidPattern)
        self.penPoligono = QPen(Qt.black)
        self.penPoligono.setWidth(1)
        self.brushPoligono = QBrush()
        self.penPoligonoVisibilidad = QPen()
        self.brushPoligonoVisibilidad = QBrush(Qt.gray)
        self.penEjes = QPen(Qt.black)
        self.penEjes.setWidth(1)
        self.penCuadricula = QPen(QColor(230,230,230))
        self.penCuadriculaLineas = QPen(QColor(200,200,200))
        self.penCuadriculaLineas.setWidth(1)
```

```
        self.penCuadricula.setWidth(1)
        self.penCuadricula.setStyle(Qt.DashLine)

    # Función para capturar los eventos del ratón para dibujar polígonos y puntos
    def mousePressEvent(self, event):
        self.posicion = QPointF(self.graphicsView.mapToScene(event.pos()))
        if self.isPunto==True:
            self.scene.addItem(self.scene.addEllipse(self.posicion.x()-
54.0,self.posicion.y()-70.0,5,5,self.penPunto,self.brushPunto))
            self.savePunto = [self.posicion.x()-54.0,self.posicion.y()-70.0]
            self.isPunto = False
        if (self.isPoligono==True) and (self.vertices!=0):
            self.cont += 1
            self.arraySavePoligono.append([self.posicion.x()-
54.0,self.posicion.y()-70.0])
            self.arrayPoligono.append(QPointF(self.posicion.x()-54.0,
self.posicion.y()-70.0))
            self.scene.addItem(self.scene.addEllipse(self.posicion.x()-
54.0,self.posicion.y()-70.0,5,5,self.penPunto,self.brushPunto))
            if self.cont == self.vertices:

self.dic_poligonos["Poligono_"+str(self.cont_poligonos)]=self.arraySavePoligono
                self.cont_poligonos=self.cont_poligonos+1
                self.scene.clear()
                if self.isEjes==True:
                    self.dibujarCuadricula()
                self.qpolygon=QPolygonF(self.arrayPoligono)
                self.scene.addItem(self.scene.addPolygon(self.qpolygon,
self.penPoligono, self.brushPoligono))
                self.cont = 0
                self.isPoligono = False

    # Función para dibujar un polígono pidiendo el número de vértices al usuario
    def dibujarPoligono(self):
        print("Dibujar Poligono")
        self.isPoligono = True
        self.vertices, _ = QInputDialog.getInt(self, 'Captura de vértices del
polígono',
                                               'Seleccione el número de vértices del
polígono:',0,4,100,1)

    # Función para dibujar un punto
    def dibujarPunto(self):
        print("Dibujar Punto")
        self.isPunto = True

    # Función para calcular la región de visibilidad de un punto interior de un
polígono
    def calcularRegionVisibilidadPunto(self):
        print("Calcular Region Visibilidad")

        # Borrar labels y escena
        self.label_1.setText("")
```

```
        self.label_2.setText("")
        self.scene.clear()

        # Dibujar ejes y cuadrícula si estaban activos previamente
        if self.isEjes==True:
            self.dibujarCuadricula()

        # Calcular el polígono de visibilidad y dibujar el polígono original,
        # el polígono de visibilidad y el punto seleccionado
        self.qpolygonVisibility =
calculoPoligonoVisibilidad(self.arraySavePoligono,self.savePunto)
        self.scene.addItem(self.scene.addPolygon(self.qpolygon, self.penPoligono,
self.brushPoligono))
        self.scene.addItem(self.scene.addPolygon(self.qpolygonVisibility,
self.penPoligonoVisibilidad, self.brushPoligonoVisibilidad))

self.scene.addItem(self.scene.addEllipse(self.savePunto[0],self.savePunto[1],5,5,
self.penPunto,self.brushPunto))

        # Calcular el porcentaje de visibilidad del polígono
        pctgVisibilidad =
self.calcularArea(self.qpolygonVisibility)/self.calcularArea(self.qpolygon)*100

        # Convertir de coordenadas de pantalla a coordenadas cartesianas el
        # punto óptimo encontrado
        x,y=convertirCoordenadas(self.savePunto[0],self.savePunto[1], 613.5,
433.5, 124, 88)

        # Mostrar el resultado del algoritmo
        self.label_1.setText("Región de visibilidad por punto")
        self.label_2.setText("Punto Elegido: " + str((round(x,2),round(y,2))) +
"\nÁrea (%) Polígono Visibilidad: " +
                             str(round(pctgVisibilidad,4))+ "%")

    # Función que implementa el Algoritmo Random Search
    def algoritmoRandomSearch(self):
        print("Algoritmo Random Search")

        puntosRS, _ = QInputDialog.getInt(self, 'Captura de puntos para algoritmo
Random Search',
                                          'Seleccione el número de puntos
utilizado para RS:',0,1,1000,1)

        # Comenzar el contador
        start = perf_counter()

        # Obtener puntos aleatorios en función de los solicitados por el
        # usuario
        puntos_selec = self.puntosAleatorios(puntosRS)

        # Calcular los polígonos de visibilidad y su área para los puntos
        pctg_list=[]
        for p in puntos_selec:
```

```
            qpolygonVisibilityRS =
calculoPoligonoVisibilidad(self.arraySavePoligono,list(p))
            pctgVisibilidadRS =
self.calcularArea(qpolygonVisibilityRS)/self.calcularArea(self.qpolygon)*100
            pctg_list.append(pctgVisibilidadRS)

        # Obtener el punto con mayor porcentaje de visibilidad
        ind_max=pctg_list.index(max(pctg_list))
        puntoRS=list(puntos_selec[ind_max])
        pctg_max_RS=max(pctg_list)
        qpolygonVisibilityRSMax =
calculoPoligonoVisibilidad(self.arraySavePoligono,puntoRS)

        # Finalizar el contador y calcular el tiempo de procesamiento
        # del algoritmo
        end = perf_counter()
        tiempo_procesamiento=end-start

        # Borrar labels y escena
        self.label_1.setText("")
        self.label_2.setText("")
        self.scene.clear()

        # Dibujar ejes y cuadrícula si estaban activos previamente
        if self.isEjes==True:
            self.dibujarCuadricula()

        # Dibujar polígono inicial y polígono de visibilidad
        self.scene.addItem(self.scene.addPolygon(self.qpolygon, self.penPoligono,
self.brushPoligono))
        self.scene.addItem(self.scene.addPolygon(qpolygonVisibilityRSMax,
self.penPoligonoVisibilidad, self.brushPoligonoVisibilidad))

        # Dibujar puntos probados por el algoritmo
        for p in puntos_selec:

self.scene.addItem(self.scene.addEllipse(p[0],p[1],2,2,self.penPuntoProbado,self.
brushPuntoProbado))

        # Dibujar punto de máxima visibilidad

self.scene.addItem(self.scene.addEllipse(puntoRS[0],puntoRS[1],5,5,self.penPunto,
self.brushPunto))

        # Convertir de coordenadas de pantalla a coordenadas cartesianas el
        # punto óptimo encontrado
        x,y=convertirCoordenadas(puntoRS[0],puntoRS[1], 613.5, 433.5, 124, 88)

        # Mostrar el resultado del algoritmo
        self.label_1.setText("Algoritmo Random Search")
        self.label_2.setText("Punto Máxima Visibilidad: " +
str((round(x,2),round(y,2))) + "\nÁrea (%) Polígono Visibilidad: " +
```

```
                            str(round(pctg_max_RS,4))+ "%" + "\nTiempo de
procesamiento: " + str(round(tiempo_procesamiento,4)) + "s")

    # Función que implementa el Algoritmo Simulated Annealing
    def algoritmoSimulatedAnnealing(self):
        print("Algoritmo Simulated Annealing")

        # Indicar la temperatura inicial como el número de vértices del
        # polígono, la temperatura final que será la condición de parada
        # del algoritmo y una variable que irá aumentando para acercarse
        # a la solución precisa
        lista_puntosSA=[]
        temp_ini=self.vertices
        temp_fin=0.03
        self.div=5.0
        k=0

        # Comenzar el contador
        start = perf_counter()

        while True:
            n=0
            while True:
                # Generar un punto aleatorio inicial
                if k==0:
                    temp=temp_ini
                    puntoSA=self.puntosAleatorios(1)
                    lista_puntosSA.append(puntoSA[0])

                # Calcular el polígono de visibilidad y el porcentaje de
                # visibilidad para el punto
                qpolygonVisibilityPuntoSA =
calculoPoligonoVisibilidad(self.arraySavePoligono,list(puntoSA[0]))
                pctgVisibilidadPuntoSA =
self.calcularArea(qpolygonVisibilityPuntoSA)/self.calcularArea(self.qpolygon)*100

                # Generar vecino y calcular su polígono de visibilidad y
                # porcentaje de visibilidad
                vecino=self.generarVecinoSA(puntoSA[0])
                lista_puntosSA.append(vecino[0])
                qpolygonVisibilityVecinoSA =
calculoPoligonoVisibilidad(self.arraySavePoligono,list(vecino[0]))
                pctgVisibilidadVecinoSA =
self.calcularArea(qpolygonVisibilityVecinoSA)/self.calcularArea(self.qpolygon)*10
0

                # En el caso que se produzca mejora en el porcentaje
                # de visibilidad, el nuevo punto pasará a ser el vecino calculado
                # y aunque no se produzca mejora con cierta probabilidad también
                # pasará
                delta = pctgVisibilidadPuntoSA - pctgVisibilidadVecinoSA
                if delta < 0:
                    puntoSA = vecino
```

```
                elif np.random.uniform(0,1) < np.exp(-delta/temp):
                    puntoSA = vecino

                n=n+1
                print('Iteración n')
                print(n)
                # Condición de parada del primer bucle
                if n>(1/temp):
                    break

            # Aumento de la variable para encontrar vecinos más próximos
            self.div=self.div/0.99

            k=k+1
            print('Iteración k')
            print(k)

            # Decremento de la temperatura y condición de parada del algoritmo
            temp=temp_ini/(1+k)
            if temp<=temp_fin:
                break

        qpolygonVisibilitySAMax =
calculoPoligonoVisibilidad(self.arraySavePoligono,puntoSA[0])

        # Finalizar el contador y calcular el tiempo de procesamiento
        # del algoritmo
        end = perf_counter()
        tiempo_procesamiento=end-start

        # Borrar labels y escena
        self.label_1.setText("")
        self.label_2.setText("")
        self.scene.clear()

        # Dibujar ejes y cuadrícula si estaban activos previamente
        if self.isEjes==True:
            self.dibujarCuadricula()

        # Dibujar polígono inicial y polígono de visibilidad
        self.scene.addItem(self.scene.addPolygon(self.qpolygon, self.penPoligono,
self.brushPoligono))
        self.scene.addItem(self.scene.addPolygon(qpolygonVisibilitySAMax,
self.penPoligonoVisibilidad, self.brushPoligonoVisibilidad))

        # Dibujar puntos probados por el algoritmo
        for p in lista_puntosSA:

self.scene.addItem(self.scene.addEllipse(p[0],p[1],2,2,self.penPuntoProbado,self.
brushPuntoProbado))

        # Dibujar punto de máxima visibilidad
```

```
self.scene.addItem(self.scene.addEllipse(puntoSA[0][0],puntoSA[0][1],5,5,self.pen
Punto,self.brushPunto))

        # Convertir de coordenadas de pantalla a coordenadas cartesianas el
        # punto óptimo encontrado
        x,y=convertirCoordenadas(puntoSA[0][0],puntoSA[0][1], 613.5, 433.5, 124,
88)

        # Mostrar el resultado del algoritmo
        self.label_1.setText("Algoritmo Simulated Annealing")
        self.label_2.setText("Punto Máxima Visibilidad: " +
str((round(x,2),round(y,2))) + "\nÁrea (%) Polígono Visibilidad: " +
                             str(round(pctgVisibilidadPuntoSA,4))+ "%" +
"\nTiempo de procesamiento: " + str(round(tiempo_procesamiento,4)) + "s")

    # Función que implementa el Algoritmo Genético
    def algoritmoGenetico(self):
        print("Algoritmo Genético")

        # Indicar población inicial, la probabilidad de mutación de los
        # padres y una variable que irá aumentando para acercarse
        # a la solución precisa
        poblacion=self.puntosAleatorios(self.vertices**2)
        lista_puntosGA=[]
        cont_mejora=1
        cont_itera=0
        prob_mut=0.5
        self.k=5.0

        # Comenzar el contador
        start = perf_counter()

        while True:
            # Calcular los polígonos de visibilidad y los porcentajes de
            # visibilidad de los puntos de la población
            lista_pctg=[]
            for punto in poblacion:
                qpolygonVisibilityPuntoGA =
calculoPoligonoVisibilidad(self.arraySavePoligono,list(punto))
                pctgVisibilidadPuntoGA =
self.calcularArea(qpolygonVisibilityPuntoGA)/self.calcularArea(self.qpolygon)*100
                lista_pctg.append(pctgVisibilidadPuntoGA)
                lista_puntosGA.append(punto)

            # Obtener porcentaje de visibilidad máximo para obtener el punto
            # óptimo
            pctg_max_GA=max(lista_pctg)
            ind_max=lista_pctg.index(pctg_max_GA)
            puntoGA=list(poblacion[ind_max])

            if cont_itera==0:
                pctg_max_selec=pctg_max_GA
```

```
            # Método de la ruleta para elegir a los padres, de tal forma
            # que los mejores puntos tendrán más probabilidad de ser
            # elegidos como los padres
            suma_pctg=sum(lista_pctg)
            lista_prob = list(map(lambda a: a/suma_pctg , lista_pctg))
            ind_padres=np.random.choice(self.vertices**2, 2, p=lista_prob,
replace=True)
            padres=[poblacion[ind_padres[0]],poblacion[ind_padres[1]]]

            # Comprobar si se produce o no mejora en el porcentaje de
            # visibilidad
            if abs(pctg_max_GA-pctg_max_selec)<1e-10:
                cont_mejora=cont_mejora+1

            pctg_max_selec=pctg_max_GA

            # Mutar los padres en función de una probabilidad de mutación
            poblacion_aux=[]
            for punto in poblacion:
                if punto==padres[0] or punto==padres[1]:
                    if np.random.uniform(0,1)<prob_mut:
                        poblacion_aux.append(self.mutarPuntoGA(punto)[0])
                    else:
                        poblacion_aux.append(punto)
                else:
                    poblacion_aux.append(punto)

            poblacion=poblacion_aux

            # Aumento de la variable para encontrar mutaciones más cercanas
            self.k=self.k+5.0

            cont_itera=cont_itera+1
            print("Iteración")
            print(cont_itera)
            print(cont_mejora)

            # Condición de parada
            if cont_mejora>200:
                break

        # Calcular el polígono de visibilidad y el porcentaje de visibilidad
        # para el punto óptimo
        qpolygonVisibilityGAMax =
calculoPoligonoVisibilidad(self.arraySavePoligono,puntoGA)
        pctgVisibilidadPuntoGA =
self.calcularArea(qpolygonVisibilityGAMax)/self.calcularArea(self.qpolygon)*100

        # Finalizar el contador y calcular el tiempo de procesamiento
        # del algoritmo
        end = perf_counter()
        tiempo_procesamiento=end-start
```

```
        # Borrar labels y escena
        self.label_1.setText("")
        self.label_2.setText("")
        self.scene.clear()

        # Dibujar ejes y cuadrícula si estaban activos previamente
        if self.isEjes==True:
            self.dibujarCuadricula()

        # Dibujar polígono inicial y polígono de visibilidad
        self.scene.addItem(self.scene.addPolygon(self.qpolygon, self.penPoligono,
self.brushPoligono))
        self.scene.addItem(self.scene.addPolygon(qpolygonVisibilityGAMax,
self.penPoligonoVisibilidad, self.brushPoligonoVisibilidad))

        # Dibujar puntos probados por el algoritmo
        for p in lista_puntosGA:

self.scene.addItem(self.scene.addEllipse(p[0],p[1],2,2,self.penPuntoProbado,self.
brushPuntoProbado))

        # Dibujar punto de máxima visibilidad

self.scene.addItem(self.scene.addEllipse(puntoGA[0],puntoGA[1],5,5,self.penPunto,
self.brushPunto))

        # Convertir de coordenadas de pantalla a coordenadas cartesianas el
        # punto óptimo encontrado
        x,y=convertirCoordenadas(puntoGA[0],puntoGA[1], 613.5, 433.5, 124, 88)

        # Mostrar el resultado del algoritmo
        self.label_1.setText("Algoritmo Genético")
        self.label_2.setText("Punto Máxima Visibilidad: " +
str((round(x,2),round(y,2))) + "\nÁrea (%) Polígono Visibilidad: " +
                             str(round(pctgVisibilidadPuntoGA,4))+ "%" +
"\nTiempo de procesamiento: " + str(round(tiempo_procesamiento,4)) + "s")

    # Función para dibujar ejes cartesianos y cuadrícula
    def dibujarCuadricula(self):
        print("Dibujar ejes y cuadrícula")
        if self.isEjes==False:
            self.isEjes=True

        # Dibujar grid
        for x in range(10,310,10):
            if x in [50, 100, 150, 200, 250, 300]:
                self.scene.addLine(x+309.5, 0, x+309.5, 438,
self.penCuadriculaLineas)
            else:
                self.scene.addLine(x+309.5, 0, x+309.5, 438, self.penCuadricula)

        for x in range(10,310,10):
```

```
            if x in [50, 100, 150, 200, 250, 300]:
                self.scene.addLine(309.5-x, 0, 309.5-x, 438,
self.penCuadriculaLineas)
            else:
                self.scene.addLine(309.5-x, 0, 309.5-x, 438, self.penCuadricula)

        for y in range(10,220,10):
            if y in [50, 100, 150, 200]:
                self.scene.addLine(0, y+219.5, 618, y+219.5,
self.penCuadriculaLineas)
            else:
                self.scene.addLine(0, y+219.5, 618, y+219.5, self.penCuadricula)

        for y in range(10,220,10):
            if y in [50, 100, 150, 200]:
                self.scene.addLine(0, 219.5-y, 618, 219.5-y,
self.penCuadriculaLineas)
            else:
                self.scene.addLine(0, 219.5-y, 618, 219.5-y, self.penCuadricula)

        # Dibujar ejes principales
        self.scene.addLine(0, 219.5, 618, 219.5, self.penEjes)
        self.scene.addLine(309.5, 0, 309.5, 438, self.penEjes)

    # Función para pintar las coordenadas de los vértices del polígono
    def indicarCoordenadas(self):
        print("Representar coordenadas")

        # Reprentar las coordenadas cartesianas de los vértices del polígono
        # en la escena
        for punto in self.arraySavePoligono:
            # Crea la etiqueta de texto con las coordenadas
            x,y=convertirCoordenadas(punto[0],punto[1], 613.5, 433.5, 124, 88)
            x=round(x,2)
            y=round(y,2)
            label = QGraphicsTextItem(f"({x}, {y})")
            # Ajusta la posición de la etiqueta al lado del punto
            label.setPos(punto[0],punto[1])
            # Agrega la etiqueta a la escena
            self.scene.addItem(label)

    # Función para borrar los elementos de la escena
    def borrarFrameDibujo(self):
        print("Borrar Escena")

        self.scene.clear()
        self.arrayPoligono = []
        self.arraySavePoligono = []
        self.arrayPoligonoVisibilidad = []
        self.label_1.setText("")
        self.label_2.setText("")
        self.isEjes=False
```

```
    # Definimos de un evento al cerrar el programa
    def closeEvent(self,event):
        ask=QMessageBox.question(self,"Salir...","¿Seguro que desea salir?",
                                 QMessageBox.Yes | QMessageBox.No)
        if ask == QMessageBox.Yes:
            # Al cerrar la aplicación se guardan los polígonos analizados en
            # la sesión en formato json
            with open("poligonos_sesion.json", "w") as archivo:
                json.dump(self.dic_poligonos, archivo)

            event.accept()
        else:
            event.ignore()

def window():
    # Instancia para iniciar una aplicación
    app=QApplication(sys.argv)
    # Crear un objeto de la clase
    window=MainWindow()
    # Mostrar la ventana
    window.show()
    # Ejecutar la aplicación
    sys.exit(app.exec_())

window()
```

Funciones.py

El siguiente archivo corresponde a dos funciones empleadas para el funcionamiento de la aplicación, por un lado, una función para calcular el polígono de visibilidad dado un polígono y un punto interior a dicho polígono, y por otro lado, una función que convierte coordenadas de pantalla a coordenadas cartesianas dado un eje cartesiano cuyo centro se encuentra en el centro de la escena gráfica.

```
import skgeom as sg
from PyQt5.QtCore import QPointF
from PyQt5.QtGui import QPolygonF

# Función para calcular el polígono de visibilidad dado un punto interior
# al polígono
def calculoPoligonoVisibilidad(lista, punto):

    cont = 0
    array = []
    outer = []
    arrayPoligonoVisibilidad = []
    lista_copia = lista.copy()
    lista_copia.append(lista_copia[0])
    vertices = len(lista_copia)-1
    long = len(lista_copia)

    if vertices % 2 == 1:
        fin=vertices-1
    else:
        fin=vertices

    for i in range(0,fin,2):
        outer.append(sg.Segment2(sg.Point2(lista_copia[i][0], lista_copia[i][1]),
sg.Point2(lista_copia[i+1][0], lista_copia[i+1][1])))
        outer.append(sg.Segment2(sg.Point2(lista_copia[i+1][0],
lista_copia[i+1][1]), sg.Point2(lista_copia[i+2][0], lista_copia[i+2][1])))

    if vertices % 2 == 1:
        outer.append(sg.Segment2(sg.Point2(lista_copia[long-2][0],
lista_copia[long-2][1]), sg.Point2(lista_copia[long-1][0], lista_copia[long-
1][1])))

    arr = sg.arrangement.Arrangement()

    for s in outer:
        arr.insert(s)

    vs = sg.RotationalSweepVisibility(arr)

    q = sg.Point2(punto[0], punto[1])
```

```
    face = arr.find(q)
    vx = vs.compute_visibility(q, face)

    for v in vx.halfedges:
        cont+=1
        if cont % 2 == 1:
            array.append([v.curve()[0].x(),v.curve()[0].y()])

    for arr in array:
        arrayPoligonoVisibilidad.append(QPointF(arr[0], arr[1]))

    qpolygonVisibility=QPolygonF(arrayPoligonoVisibilidad)

    return qpolygonVisibility

# Función para convertir coordenadas de pantalla a coordenadas cartesianas
# dado una ventana específica y un sistema cartesiano
def convertirCoordenadas(x_pantalla, y_pantalla, ancho_pantalla, alto_pantalla,
ancho_cartesiano, alto_cartesiano):

    centro_x = ancho_pantalla / 2
    centro_y = alto_pantalla / 2

    dif_x = x_pantalla - centro_x
    dif_y = centro_y - y_pantalla

    x_cartesiano = ancho_cartesiano * dif_x / ancho_pantalla
    y_cartesiano = alto_cartesiano * dif_y / alto_pantalla

    return x_cartesiano, y_cartesiano
```

Appdesign.py

Este archivo contiene la configuración de la ventana principal y de todos los componentes contenidos en ella. El código ha sido generado con la ayuda de la aplicación Qt Designer, empleada para el diseño y la construcción de la toda la interfaz gráfica del usuario.

```
from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(800, 617)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.verticalLayout = QtWidgets.QVBoxLayout(self.centralwidget)
        self.verticalLayout.setContentsMargins(0, 0, 0, 0)
        self.verticalLayout.setSpacing(0)
        self.verticalLayout.setObjectName("verticalLayout")
        self.framePrincipal = QtWidgets.QFrame(self.centralwidget)
        self.framePrincipal.setEnabled(True)
        self.framePrincipal.setFrameShape(QtWidgets.QFrame.StyledPanel)
        self.framePrincipal.setFrameShadow(QtWidgets.QFrame.Raised)
        self.framePrincipal.setObjectName("framePrincipal")
        self.clearButton = QtWidgets.QPushButton(self.framePrincipal)
        self.clearButton.setGeometry(QtCore.QRect(690, 450, 93, 28))
        self.clearButton.setStyleSheet("background-color: #C2DABD; color:
#303030; font-size: 12px; font-weight: bold; border: 1px solid black")
        self.clearButton.setObjectName("clearButton")
        self.frameCalculo = QtWidgets.QFrame(self.framePrincipal)
        self.frameCalculo.setGeometry(QtCore.QRect(50, 500, 621, 61))
        self.frameCalculo.setStyleSheet("background-color: rgb(213, 220, 233);
border: 1px solid black")
        self.frameCalculo.setFrameShape(QtWidgets.QFrame.StyledPanel)
        self.frameCalculo.setFrameShadow(QtWidgets.QFrame.Raised)
        self.frameCalculo.setObjectName("frameCalculo")
        self.label_2 = QtWidgets.QLabel(self.frameCalculo)
        self.label_2.setGeometry(QtCore.QRect(284, 5, 331, 51))
        self.label_2.setStyleSheet("border: rgb(213, 220, 233)")
        self.label_2.setText("")

self.label_2.setAlignment(QtCore.Qt.AlignLeading|QtCore.Qt.AlignLeft|QtCore.Qt.Al
ignVCenter)
        self.label_2.setObjectName("label_2")
        self.label_1 = QtWidgets.QLabel(self.frameCalculo)
        self.label_1.setGeometry(QtCore.QRect(20, 20, 241, 21))
        font = QtGui.QFont()
        font.setPointSize(9)
        font.setBold(True)
        font.setWeight(75)
        self.label_1.setFont(font)
```

```
        self.label_1.setStyleSheet("border: rgb(213, 220, 233)")
        self.label_1.setText("")
        self.label_1.setObjectName("label_1")
        self.graphicsView = QtWidgets.QGraphicsView(self.framePrincipal)
        self.graphicsView.setGeometry(QtCore.QRect(50, 40, 621, 441))
        self.graphicsView.setObjectName("graphicsView")
        self.verticalLayout.addWidget(self.framePrincipal)
        MainWindow.setCentralWidget(self.centralwidget)
        self.menuBar = QtWidgets.QMenuBar(MainWindow)
        self.menuBar.setGeometry(QtCore.QRect(0, 0, 800, 26))
        self.menuBar.setObjectName("menuBar")
        self.menuDibujar = QtWidgets.QMenu(self.menuBar)
        self.menuDibujar.setObjectName("menuDibujar")
        self.menuCalcular = QtWidgets.QMenu(self.menuBar)
        self.menuCalcular.setObjectName("menuCalcular")
        self.menuAlgoritmos = QtWidgets.QMenu(self.menuBar)
        self.menuAlgoritmos.setObjectName("menuAlgoritmos")
        self.menuOpciones = QtWidgets.QMenu(self.menuBar)
        self.menuOpciones.setObjectName("menuOpciones")
        MainWindow.setMenuBar(self.menuBar)
        self.actionPoligono = QtWidgets.QAction(MainWindow)
        self.actionPoligono.setObjectName("actionPoligono")
        self.actionPunto = QtWidgets.QAction(MainWindow)
        self.actionPunto.setObjectName("actionPunto")
        self.actionRegionVisibilidad = QtWidgets.QAction(MainWindow)
        self.actionRegionVisibilidad.setObjectName("actionRegionVisibilidad")
        self.actionDibujar_Eje = QtWidgets.QAction(MainWindow)
        self.actionDibujar_Eje.setObjectName("actionDibujar_Eje")
        self.actionDibujar_Coordenadas = QtWidgets.QAction(MainWindow)
        self.actionDibujar_Coordenadas.setObjectName("actionDibujar_Coordenadas")
        self.actionRandom_Search = QtWidgets.QAction(MainWindow)
        self.actionRandom_Search.setObjectName("actionRandom_Search")
        self.actionSimulated_Annealing = QtWidgets.QAction(MainWindow)
        self.actionSimulated_Annealing.setObjectName("actionSimulated_Annealing")
        self.actionGenetic_Algorithm = QtWidgets.QAction(MainWindow)
        self.actionGenetic_Algorithm.setObjectName("actionGenetic_Algorithm")
        self.actionEstudio_Comparativo = QtWidgets.QAction(MainWindow)
        self.actionEstudio_Comparativo.setObjectName("actionEstudio_Comparativo")
        self.menuDibujar.addAction(self.actionPoligono)
        self.menuDibujar.addAction(self.actionPunto)
        self.menuCalcular.addAction(self.actionRegionVisibilidad)
        self.menuAlgoritmos.addAction(self.actionRandom_Search)
        self.menuAlgoritmos.addAction(self.actionSimulated_Annealing)
        self.menuAlgoritmos.addAction(self.actionGenetic_Algorithm)
        self.menuOpciones.addAction(self.actionDibujar_Eje)
        self.menuOpciones.addAction(self.actionDibujar_Coordenadas)
        self.menuBar.addAction(self.menuDibujar.menuAction())
        self.menuBar.addAction(self.menuOpciones.menuAction())
        self.menuBar.addAction(self.menuCalcular.menuAction())
        self.menuBar.addAction(self.menuAlgoritmos.menuAction())

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)
```

```
    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
        self.clearButton.setText(_translate("MainWindow", "Borrar Escena"))
        self.menuDibujar.setTitle(_translate("MainWindow", "Dibujar"))
        self.menuCalcular.setTitle(_translate("MainWindow", "Calcular"))
        self.menuAlgoritmos.setTitle(_translate("MainWindow", "Algoritmos"))
        self.menuOpciones.setTitle(_translate("MainWindow", "Opciones"))
        self.actionPoligono.setText(_translate("MainWindow", "Polígono"))
        self.actionPunto.setText(_translate("MainWindow", "Punto"))
        self.actionRegionVisibilidad.setText(_translate("MainWindow", "Polígono
de Visibilidad"))
        self.actionDibujar_Eje.setText(_translate("MainWindow", "Dibujar Eje"))
        self.actionDibujar_Coordenadas.setText(_translate("MainWindow", "Dibujar
Coordenadas"))
        self.actionRandom_Search.setText(_translate("MainWindow", "Random
Search"))
        self.actionSimulated_Annealing.setText(_translate("MainWindow",
"Simulated Annealing"))
        self.actionGenetic_Algorithm.setText(_translate("MainWindow", "Genetic
Algorithm"))
        self.actionEstudio_Comparativo.setText(_translate("MainWindow", "Estudio
Comparativo"))


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())
```

ANEXO III: TABLAS DEL ESTUDIO COMPARATIVO

Polígonos de 15 vértices	Random Search 250 puntos					
	Prueba 1		Prueba 2		Prueba 3	
	Tiempo Procesamiento [s]	% Área Visibilidad	Tiempo Procesamiento [s]	% Área Visibilidad	Tiempo Procesamiento [s]	% Área Visibilidad
Polígono 1	0,258080699999936	90,794356464572900	0,250175799999851	91,662124889279800	0,280217199999697	90,872157726162100
Polígono 2	0,239137200000186	91,704328890487900	0,234074299999974	91,080942007101200	0,222553300000072	91,059192065621900
Polígono 3	0,229884400000173	92,680937628245000	0,291367400000126	93,139858671148600	0,244381200000134	93,106589045652300
Polígono 4	0,192688999999973	81,294366491515900	0,196071200000005	80,902084418838400	0,203851100000065	80,012353819692200
Polígono 5	0,238599700000122	100,000000000000000	0,223797399999966	100,000000000000000	0,371544500000254	100,000000000000000
Polígono 6	0,221996400000080	90,947853447468400	0,231502900000123	91,047021344119400	0,347004199999901	91,036073356676400
Polígono 7	0,232106199999634	84,915166049210000	0,229300100000273	84,938769107257000	0,285723299999972	84,824359044058600
Polígono 8	0,240257299999939	97,461282488663700	0,221645699999953	97,439220797933600	0,353943900000103	97,627610802260500
Polígono 9	0,209212799999932	92,801145888858800	0,223429900000155	92,783438774230200	0,331725600000027	92,870104935877400
Polígono 10	0,217707099999643	92,497372119759900	0,254985100000340	92,062553121210000	0,363127399999939	92,316434968799000
Polígono 11	0,213235499999882	97,172176670396000	0,224422799999956	97,137839075992600	0,394576400000005	97,290850024979100
Polígono 12	0,242558500000086	94,165429538269800	0,235552300000108	93,706434080629800	0,357574599999679	94,043099953685800
Polígono 13	0,236156399999799	97,062831148666100	0,232506800000010	97,107607761818200	0,287307699999928	96,771933128216200
Polígono 14	0,212563199999749	93,221007922119700	0,233275999999932	93,074106521254200	0,213669900000240	93,244304660612500
Polígono 15	0,231294700000034	93,685142839106400	0,211624400000346	93,839658256988900	0,260549200000241	93,905849027293200
Polígono 16	0,222004299999753	96,973241492413300	0,216999599999780	96,981014077744000	0,298648700000285	96,788201401321200
Polígono 17	0,220558499999697	99,726051306316300	0,274893799999972	100,000000000000000	0,333435900000040	99,961662715693800
Polígono 18	0,211140099999738	97,641287311802100	0,260267499999827	97,155544013239800	0,245942700000341	97,190741879143300
Polígono 19	0,227660200000173	90,311256612394900	0,253467800000180	90,019065757671100	0,266503299999840	90,341543632752800
Polígono 20	0,246760500000164	96,236253345422300	0,275746399999661	96,316571566408400	0,381345900000269	96,243460371480500
Polígono 21	0,257047499999771	90,471300244533500	0,274060200000349	90,984679422576600	0,365559699999721	90,918073112752900
Polígono 22	0,255103700000290	89,520493214454400	0,236011299999972	89,631566394909600	0,312980900000184	89,539759818968000
Polígono 23	0,244306500000220	99,604812644483600	0,227308800000173	99,562229812018100	0,392180099999677	99,587745044638100
Polígono 24	0,220585199999732	94,215942103768500	0,223173800000040	93,948338794456600	0,263553900000260	93,917911720842000
Polígono 25	0,224992399999791	98,331686416330800	0,248419300000023	98,415699452223300	0,249013999999988	98,222103369362900

Polígonos de 15 vértices	Random Search 500 puntos					
	Prueba 1		Prueba 2		Prueba 3	
	Tiempo Procesamiento [s]	% Área Visibilidad	Tiempo Procesamiento [s]	% Área Visibilidad	Tiempo Procesamiento [s]	% Área Visibilidad
Polígono 1	0,517553099999986	91,901870636005100	0,488753999999971	91,917672334294300	0,451256700000158	91,263419494951200
Polígono 2	0,449397899999894	90,994065248360200	0,436535800000001	91,368789599776400	0,459516300000359	91,749024628004900
Polígono 3	0,482936200000040	92,749518013713700	0,465999900000042	93,220731806950300	0,468463700000029	92,778754243124200
Polígono 4	0,397755599999982	80,897484543360500	0,371683299999858	80,658646139844600	0,367467699999906	79,941482796299700
Polígono 5	0,494691899999907	100,000000000000000	0,467476699999679	100,000000000000000	0,488756899999771	100,000000000000000
Polígono 6	0,463096899999982	91,047498462365700	0,477712500000052	91,038525866825900	0,481314899999688	91,012194556895300
Polígono 7	0,415122500000052	84,946750598658300	0,414032899999710	84,993338463826400	0,424004599999989	84,870445797709300
Polígono 8	0,434886600000027	97,649365256065400	0,487807100000281	97,599743460340900	0,440337600000020	97,447238458461600
Polígono 9	0,435952499999984	92,680180197188300	0,445726800000102	92,796560631014600	0,445979900000111	92,723377200472700
Polígono 10	0,466477599999961	92,353655703568200	0,475097600000026	92,193629601599200	0,450163500000144	92,247469795418000
Polígono 11	0,489679100000103	97,059467385732700	0,444678700000167	96,953060666429800	0,433138800000051	97,143370146697200
Polígono 12	0,440721299999950	93,958080611720700	0,465051199999834	93,889103796643400	0,441867899999579	94,164006765547700
Polígono 13	0,479873400000087	96,697896179086500	0,431614699999954	97,155685789977200	0,448759800000061	97,143957788226800
Polígono 14	0,448814500000025	92,860873607051600	0,419899500000155	93,314702794777900	0,480381500000021	93,426177663727800
Polígono 15	0,451398000000153	93,967976933737800	0,431260400000155	93,750621838525500	0,470007699999769	93,948508617675500
Polígono 16	0,534324200000355	97,085467355078900	0,447830800000247	96,932595784326900	0,427495800000087	96,974546537614200
Polígono 17	0,429454099999929	99,787180270860900	0,443671099999846	100,000000000000000	0,435375300000032	99,825093734913700
Polígono 18	0,495339600000079	97,897113727117300	0,434809399999721	97,687096682885000	0,433279800000036	97,649071495191200
Polígono 19	0,521753699999862	90,386175837539200	0,480198300000211	90,433157643213100	0,488566999999875	90,308541139343200
Polígono 20	0,458177700000305	96,321162806980600	0,496648399999685	96,360275066053500	0,441679399999884	96,361423833989600
Polígono 21	0,515800200000285	90,861168295395600	0,476632000000336	90,942702931117400	0,507903499999883	90,807891593722000
Polígono 22	0,450678600000173	89,895525830101100	0,477059399999689	90,211412511762900	0,455655100000058	89,890961060639100
Polígono 23	0,466989500000181	99,630057265918400	0,506616499999836	99,608018964801600	0,510841499999969	99,617708990806400
Polígono 24	0,445826699999997	94,216465094025300	0,498162299999876	94,284407180820200	0,466026399999918	94,131215283837200
Polígono 25	0,448861200000010	98,161962091239600	0,457753799999864	98,189782885214200	0,473977900000136	98,477781348444000

Polígonos de 15 vértices	Simulated Annealing		Algoritmo Genético	
	Prueba 1		Prueba 1	
	Tiempo Procesamiento [s]	% Área Visibilidad	Tiempo Procesamiento [s]	% Área Visibilidad
Polígono 1	19,695863200000000	92,248952567914500	40,344119900000000	91,360522034778500
Polígono 2	19,412841800000000	91,960270401873800	47,984116800000000	90,496699691391100
Polígono 3	18,548918800000100	92,778707949033000	50,398248700000000	93,097906084971300
Polígono 4	17,648587800000000	80,123510195588200	39,879011900000000	79,347770261527100
Polígono 5	15,965784600000000	100,000000000000000	45,499812099999900	100,000000000000000
Polígono 6	19,399641399999800	91,009438619281400	45,125812100000000	91,037087530097700
Polígono 7	25,982185800000000	84,998653372734500	45,404473899999900	84,984584251208000
Polígono 8	28,695934400000100	97,660916459049100	49,581390300000000	97,555818880121600
Polígono 9	28,380390200000100	92,930407703290400	43,713965099999900	92,720740550660200
Polígono 10	29,481948000000100	92,646284330403900	48,011074400000100	92,306913697768600
Polígono 11	31,050174299999900	97,462182536271400	47,506418899999900	97,116169928228600
Polígono 12	34,883155799999900	94,123602973702700	48,770967700000000	93,761194552485200
Polígono 13	32,587056600000000	97,115472814099800	46,924852800000000	97,096322682441000
Polígono 14	30,045309300000100	93,461561644704600	47,436689300000000	92,216103037621700
Polígono 15	32,517160700000200	93,998779015047500	46,618572900000000	93,977029942036500
Polígono 16	30,998459899999900	97,062361244138300	47,324411800000000	96,989987204647500
Polígono 17	26,918218799999900	99,963735358246200	48,627543100000000	99,851216439272000
Polígono 18	31,108896900000200	98,047547744849600	46,840141499999800	96,838441476758800
Polígono 19	32,053884899999900	90,473950931704000	48,866319299999900	90,186673520612200
Polígono 20	31,644644699999800	96,369683346826600	49,498739299999900	96,196155434430500
Polígono 21	34,474236399999800	90,892757689843300	52,480113299999800	90,894910324009900
Polígono 22	30,000596600000300	90,201983039902800	48,018067499999900	89,948745532748200
Polígono 23	28,880237999999700	99,641204750603000	49,207931800000100	99,566861768774500
Polígono 24	31,470092900000000	94,283513423610500	48,423421500000100	94,131783861453200
Polígono 25	30,766842300000000	98,456623848848900	46,917855100000000	98,459143755282400

Polígonos de 25 vértices	Random Search 250 puntos					
	Prueba 1		Prueba 2		Prueba 3	
	Tiempo Procesamiento [s]	% Área Visibilidad	Tiempo Procesamiento [s]	% Área Visibilidad	Tiempo Procesamiento [s]	% Área Visibilidad
Polígono 1	0,377791100000649	91,254025361763900	0,350543200000174	91,679696002834800	0,376011100000141	91,572036532415400
Polígono 2	0,317820600000231	82,501430450233500	0,355731300000115	82,575766474984200	0,335188299999572	82,638610893532100
Polígono 3	0,293863400000191	71,176494146771400	0,301556999999775	70,667458085276700	0,294750200000635	70,481429365946800
Polígono 4	0,321682900000269	76,127539783971400	0,367846999999528	76,268066493543200	0,308115199999520	75,813993965070400
Polígono 5	0,356627899999693	87,682574527666900	0,330915800000184	86,833524750621900	0,361310099999172	87,503602602126800
Polígono 6	0,341678599999795	85,527476437117700	0,321299899999758	84,872959666414100	0,321670599999379	82,582405632778400
Polígono 7	0,380968199999188	86,697482118187300	0,333372999999483	86,680127256065800	0,324179999999614	86,200430589990100
Polígono 8	0,315562699999645	73,426014819852600	0,328557400000136	73,423153724946900	0,306297199999789	73,024658640181200
Polígono 9	0,385668299999451	91,344898729126700	0,400140799999462	91,353047545951900	0,355807599999934	91,391884519606700
Polígono 10	0,343512800000098	93,703692841759900	0,350137800000084	93,441653604372800	0,334873699999661	93,701363398823300
Polígono 11	0,358361500000683	86,879183808121900	0,330065199999808	86,798014181220400	0,317681100000299	86,884989884898800
Polígono 12	0,365291899999647	94,355526921442900	0,362093899999308	94,462328981162000	0,383201300000109	93,976195871876600
Polígono 13	0,361256500000308	85,544318809257000	0,379800099999556	85,880824061809000	0,329797100000178	85,585308119189300
Polígono 14	0,335226499999407	95,517720864320500	0,363093599999956	95,463947892469600	0,363148200000068	95,498642967667500
Polígono 15	0,371222900000248	90,040396535705200	0,345428399999946	90,172348583117500	0,379776100000526	90,206359357969100
Polígono 16	0,335319099999651	95,641080814822500	0,358293299999786	95,679158563557700	0,397033399999600	95,753331524163300
Polígono 17	0,302823199999693	50,749400309314600	0,349968399999852	50,281496614045000	0,310532000000421	50,182307060551600
Polígono 18	0,327745699999468	86,989104476242800	0,344387600000118	86,976741350898800	0,387937800000145	87,105351427128400
Polígono 19	0,349011799999971	83,689338023099300	0,343889699999635	82,005752206388400	0,361999599999762	82,850601031516100
Polígono 20	0,334351200000128	93,364469809294300	0,326793600000200	93,331171365314400	0,358202599999458	93,355350943711900
Polígono 21	0,350396900000305	77,952317682722800	0,302119900000434	77,946583653112600	0,307758500000090	78,374557886123300
Polígono 22	0,317354600000726	68,976613060576100	0,284525600000051	68,102202648361200	0,283054599999559	68,215428309281200
Polígono 23	0,347267299999657	95,671403689340400	0,332272799999373	95,722185914422700	0,325823600000148	95,907702015226200
Polígono 24	0,363672600000427	86,538260373701500	0,296811999999590	86,514943985914600	0,300057999999808	86,652639564031300
Polígono 25	0,339559400000325	78,333126715530000	0,295420899999953	78,573725851964000	0,341006800000286	78,604580859366400

Polígonos de 25 vértices	Random Search 500 puntos					
	Prueba 1		Prueba 2		Prueba 3	
	Tiempo Procesamiento [s]	% Área Visibilidad	Tiempo Procesamiento [s]	% Área Visibilidad	Tiempo Procesamiento [s]	% Área Visibilidad
Polígono 1	0,710711799999444	91,368182242832700	0,725728800000069	91,797405585398200	0,759629600000153	91,707910653381100
Polígono 2	0,654238199999781	82,664087667665500	0,671242599999459	82,635474908018900	0,664813200000026	82,597893783596600
Polígono 3	0,635682899999665	71,203638934442900	0,630822300000545	70,714468475512300	0,628272599999945	70,893445486726900
Polígono 4	0,637567600000693	76,217694237488800	0,615603000000192	75,824423442279200	0,651202099999864	76,172652220199200
Polígono 5	0,680450300000302	88,156503746889400	0,650699900000290	87,122997941789800	0,626467200000661	87,622989852823000
Polígono 6	0,686334699999861	86,579368243352200	0,637827499999730	85,402626968479000	0,698109099999783	86,050808390330600
Polígono 7	0,692274500000166	86,650312428017800	0,696125899999970	86,820375854817400	0,715601800000513	86,793533306814800
Polígono 8	0,707417100000384	73,230578688731000	0,622448000000076	73,288433125696800	0,605158999999730	73,653436333280900
Polígono 9	0,823387599999478	91,414807496699500	0,721512899999652	91,424583305961100	0,737064399999326	91,398895884153900
Polígono 10	0,718962000000829	93,493886083217000	0,713823299999603	93,590568183346000	0,662725099999988	93,515662346360100
Polígono 11	0,654728900000009	86,599675769852900	0,661503700000139	86,897794566358400	0,646053800000117	86,923830977594200
Polígono 12	0,679975099999865	94,427509876481400	0,690140200000314	94,489820870126200	0,649950999999418	94,343082853898800
Polígono 13	0,643474799999239	85,383191047089000	0,650092500000027	85,870695391521100	0,629140600000027	85,709152116149300
Polígono 14	0,689443499999470	95,520538591647300	0,689820200000212	95,529424137261400	0,664220499999828	95,526757141488600
Polígono 15	0,711320600000362	90,154056046634700	0,671595499999966	90,021942169638700	0,655931499999496	90,268740071511100
Polígono 16	0,681186699999671	95,718592691646100	0,682992199999716	95,804487647926600	0,659482899999602	95,720688076639300
Polígono 17	0,543315099999745	50,266688178234100	0,569514899999376	50,627357797473400	0,537998299999344	50,701372798447600
Polígono 18	0,639212399999451	87,362957222738800	0,690118100000290	86,890887373324500	0,706078299999717	86,629196421388100
Polígono 19	0,633825799999613	82,335538700279900	0,641686799999661	83,250757034450000	0,639559100000042	82,822570401317800
Polígono 20	0,647685599999931	93,425957860931400	0,722835799999302	93,409840468192800	0,657631399999445	93,506313254896300
Polígono 21	0,643608200000017	78,475582842996200	0,631486400000540	78,241221284995300	0,629540199999610	78,108309963925400
Polígono 22	0,586855200000172	69,263356457021500	0,657167700000172	69,050828973774400	0,595107600000119	68,993915430820000
Polígono 23	0,678927200000544	95,992978809132300	0,722871099999792	96,010276402237600	0,657203200000367	96,126485900294300
Polígono 24	0,647230399999898	86,693485024710100	0,610121600000638	86,529538796928700	0,600451999999677	86,595259353386800
Polígono 25	0,635449800000060	78,102683404528200	0,642042200000105	78,558783321432300	0,637222899999869	78,665666367461900

Polígonos de 25 vértices	Simulated Annealing		Algoritmo Genético	
	Prueba 1		Prueba 1	
	Tiempo Procesamiento [s]	% Área Visibilidad	Tiempo Procesamiento [s]	% Área Visibilidad
Polígono 1	96,557335000000000	91,324814266100700	168,953420800000000	91,392642676641500
Polígono 2	106,652787900000000	82,686412977662400	173,999155200000000	82,650739004060500
Polígono 3	106,359260000000000	71,878301780550000	153,243571599999000	71,830646477393900
Polígono 4	111,339276899999000	76,449169761214900	161,065243700000000	75,914335908530600
Polígono 5	113,952410199999000	89,180217016349100	162,712532799999000	88,815036214086900
Polígono 6	107,105741599999000	86,637998565032600	185,563983300000000	85,276529572748500
Polígono 7	69,462072100000000	86,868008982189100	218,585077900000000	86,787531985836500
Polígono 8	68,173105600000100	73,852575749717900	161,086543900000000	73,463043723787500
Polígono 9	70,736852500000000	91,441672638482100	252,845629200000000	91,359935104854700
Polígono 10	65,100457200000000	93,676959345950300	220,304239399999000	93,547111341676700
Polígono 11	81,791874000000000	86,929335741523700	166,690712799999000	86,970246262601800
Polígono 12	84,650947399999900	94,526733418373600	202,065228399999000	94,478435758062200
Polígono 13	87,450825500000000	86,791980827565800	170,903229200000000	85,948644911244500
Polígono 14	67,915021900000000	95,564328066794700	191,165113999999000	95,540064550834700
Polígono 15	67,869209400000000	90,213328551298300	254,231621100000000	90,186017896907700
Polígono 16	66,957368699999900	95,832526550181900	262,090343700000000	95,797547431295800
Polígono 17	54,875196999999800	47,814233890628200	201,276462199999000	50,703802108714100
Polígono 18	82,820928099999900	87,383926898287700	251,951410799999000	86,957719308889300
Polígono 19	83,484646000000100	83,775309912255100	244,962428199999000	83,362601036789400
Polígono 20	85,318478199999800	93,565201418642100	221,045029500000000	93,478781830384800
Polígono 21	86,150463400000000	78,593592369620500	157,463740500000000	78,348188827314600
Polígono 22	75,922894699999900	69,932788144952300	205,289210500000000	69,462635553581300
Polígono 23	86,146619600000100	96,236078559105600	256,803221100000000	96,158212458508300
Polígono 24	64,374264100000100	86,526024895536100	224,949227699999000	86,676039170554400
Polígono 25	63,225327999999900	78,770012790025600	228,324400200000000	78,663158045503700

Polígonos de 35 vértices	Random Search 250 puntos					
	Prueba 1		Prueba 2		Prueba 3	
	Tiempo Procesamiento [s]	% Área Visibilidad	Tiempo Procesamiento [s]	% Área Visibilidad	Tiempo Procesamiento [s]	% Área Visibilidad
Polígono 1	0,635141700000531	93,240872266087700	0,561427199998433	92,757597262600700	0,573524100000213	93,182683385796900
Polígono 2	0,618412100000568	87,844997231966200	0,535442500000499	87,804150102779200	0,527391799998440	87,829016738729000
Polígono 3	0,575577800002065	92,508052685418700	0,532564299999648	92,773363976166600	0,537315400000807	92,992511538703100
Polígono 4	0,480774500003462	81,918456017694500	0,462185500000487	82,776714190946600	0,472144399998796	82,428240769089100
Polígono 5	0,483369100002164	80,346476322356200	0,483398500000475	80,090306242557100	0,501620500002900	79,466632720306700
Polígono 6	0,503924399999959	92,853783080809500	0,494683199998689	92,772903321747700	0,528700099999696	92,967296205235200
Polígono 7	0,546556900000723	91,029856670362400	0,582626199997321	91,288031509689300	0,532456500000989	91,318880846817400
Polígono 8	0,528351799999654	91,903375468572400	0,514542500000970	91,198466100538400	0,518885400000726	91,139396869475200
Polígono 9	0,495410499999707	76,353727344307400	0,469407499997032	76,199936200844900	0,458609500001330	76,585243269029300
Polígono 10	0,453802399999403	80,925347501035900	0,482744099997944	80,717958188604300	0,489013499998691	80,728688092714700
Polígono 11	0,484716200000548	91,944051135751800	0,485357000001386	91,895905562453000	0,570715099998778	91,819528025681300
Polígono 12	0,524889999996958	93,419683994297600	0,544933699999091	93,507376293551100	0,564996999997674	93,890184766951200
Polígono 13	0,478983899996819	84,351156825437600	0,487280900000769	84,155083375102000	0,462019799997506	85,042353354200800
Polígono 14	0,524965500000689	82,138317120979200	0,505158700001629	82,021746297771700	0,486880000000383	82,139864817058300
Polígono 15	0,485439399999449	87,458496533765700	0,522908599999937	87,445996437729600	0,547463899998547	87,316565895484300
Polígono 16	0,527997000001050	88,881889803252100	0,518473600001016	88,829724798574500	0,501401199999236	88,822954603724900
Polígono 17	0,556418599997414	95,329557770264400	0,550052700000378	95,349033876747200	0,520968800003174	95,376961634066300
Polígono 18	0,489942299998801	77,428380319262000	0,474755999999615	76,930033355318800	0,498995900001318	77,603054918811900
Polígono 19	0,505627700000332	85,720164052806200	0,493891999998595	85,776047460290800	0,525215300000127	86,033549712118600
Polígono 20	0,462828499999886	91,557566701647600	0,486211499999626	91,564877298270900	0,488201399999525	90,515383029440400
Polígono 21	0,520843499998591	97,755468955835800	0,513970500000141	97,654511733134400	0,531708999998954	97,754587192424300
Polígono 22	0,482207900000503	87,806482785207700	0,478704600001947	88,100312997382300	0,511588700002903	87,694323432556400
Polígono 23	0,471223699998518	87,154072255937300	0,481221199999708	86,872684192622000	0,525110599999607	86,778516324108300
Polígono 24	0,503443900001002	88,299636882078600	0,532758699999249	88,122747251582300	0,511716999997588	88,282686345644300
Polígono 25	0,548024600000644	95,351026711231000	0,523033199999190	95,225324972596900	0,577771199998096	95,306749459689800

Polígonos de 35 vértices	Random Search 500 puntos					
	Prueba 1		Prueba 2		Prueba 3	
	Tiempo Procesamiento [s]	% Área Visibilidad	Tiempo Procesamiento [s]	% Área Visibilidad	Tiempo Procesamiento [s]	% Área Visibilidad
Polígono 1	1,323068599998800	93,030849966402900	1,212447599998990	93,304885287156700	1,147929399998240	93,324590760548100
Polígono 2	1,058367800000260	87,944907352052700	1,038702800000460	87,908705214846500	1,066956100003150	87,938156795920700
Polígono 3	1,046810899999400	92,957467183446400	1,294250000002640	92,739916358630100	1,049842200001870	92,971785205241100
Polígono 4	0,922977300000638	82,678556266139400	1,051726699999560	82,866215721696700	0,950262199999997	82,967333541833200
Polígono 5	0,910526700001355	80,301561965925200	0,943692900000314	80,363385178501000	0,978172899998753	80,212854118423400
Polígono 6	0,986547399999835	93,063672736463300	0,936858699998993	93,231541300627600	0,993017299999337	93,278049623420000
Polígono 7	1,057829699999270	91,245408250070100	1,065064599999450	91,335129894138500	1,039253599999930	91,287229675024400
Polígono 8	1,001873999997760	91,929319422134200	1,036433299999770	91,708463110677200	1,023548299999670	92,138321748162500
Polígono 9	0,938492200002656	76,587162299291100	0,935631799999100	76,142092238640700	0,971350600000732	75,681142525131000
Polígono 10	0,957211400000233	80,754778602080500	0,944725999997899	80,874477866510400	0,915283599999384	80,875255380560300
Polígono 11	1,030251799998950	92,040112137983500	0,958761500001855	91,916577937237600	0,937245799999800	92,003680601073800
Polígono 12	1,109731699998510	93,982760951219700	0,967828299999382	93,899429358554100	1,009464899998420	93,998426628243900
Polígono 13	0,949601700001949	84,735647411551300	0,881826799999544	84,168675612780000	0,905636499999673	83,876554024411700
Polígono 14	0,907146400000783	82,142571000235100	0,912588700000924	82,284052627920000	1,014587900001060	82,209902037813800
Polígono 15	0,967986499999824	87,307982401998100	1,035770300000870	87,238327531027600	1,017600099999980	87,362838972448600
Polígono 16	0,938423200001125	88,821790233823800	0,977085999998962	88,821074084366000	0,982869200001005	88,868016797182600
Polígono 17	0,995098299998062	95,378384846007800	1,016694899997670	95,409092385176500	1,009069499999890	95,366252634410400
Polígono 18	0,909029999998892	77,515840032382200	0,884892800000670	77,450550477782400	0,932148600000800	77,715350686923700
Polígono 19	0,929371399997762	85,751494239837900	0,899451999997836	85,965940864759400	1,025921399999780	85,825392869251600
Polígono 20	0,929256600000371	91,679844321167700	0,906991399999242	91,461844841031600	0,990492500000982	91,537379544671100
Polígono 21	1,032604299998630	97,760181428065900	1,038280299999310	97,725874877972000	1,031778399999890	97,780473369935800
Polígono 22	0,921918200001528	87,868054136975700	0,907676599999831	88,022533715998700	0,946954099999857	87,914301701032100
Polígono 23	0,947437200000422	87,348561542156200	0,914625000001251	87,366988812175200	0,956584100000327	86,921116238434400
Polígono 24	0,927466799999820	88,226646958269300	0,947191599996585	88,289369211999300	0,980614600000990	88,252141270535000
Polígono 25	1,001951800000820	95,185710564135900	0,988228000001981	95,342000878848600	1,039373800002060	95,338298348323900

Polígonos de 35 vértices	Simulated Annealing		Algoritmo Genético	
	Prueba 1		Prueba 1	
	Tiempo Procesamiento [s]	% Área Visibilidad	Tiempo Procesamiento [s]	% Área Visibilidad
Polígono 1	82,797245300000200	93,419838568392800	503,583194300000000	93,471678062385100
Polígono 2	84,396898699999800	87,924459839853700	517,633725400000000	87,941287932580500
Polígono 3	85,893138999999700	92,950117383548900	511,973000799999000	92,986519031007900
Polígono 4	95,968684999999800	83,687350606976700	437,862350999998000	83,225918182897300
Polígono 5	133,844903199999000	80,433849227545100	445,721814700000000	80,396534850386900
Polígono 6	130,964380000000000	93,336962538438000	472,638089699999000	93,241543620597000
Polígono 7	148,028703200000000	91,264930717026700	512,113942300000000	91,323870886895900
Polígono 8	141,247275299999000	92,192576322288600	490,629898600000000	91,983564003661500
Polígono 9	137,062597699999000	76,694950941429500	455,281208099999000	76,455641076444900
Polígono 10	126,121164500000000	80,139721304860400	450,184476299999000	80,897649853859600
Polígono 11	133,684087199999000	92,043057902946500	459,847126700000000	92,186530591193700
Polígono 12	143,839764899999000	94,068082421728100	491,386878200000000	94,002511260993300
Polígono 13	131,226995000000000	85,085894629579800	436,024396900000000	84,093622248381100
Polígono 14	130,831952500000000	82,027199532249200	464,378966799999000	82,407013112639400
Polígono 15	132,196274400000000	87,479407802448100	487,822340699998000	87,270387514663500
Polígono 16	144,734025899999000	88,866698434893000	482,208540999999000	88,876921247361800
Polígono 17	132,233367199999000	95,392912663506500	511,122172700001000	95,381617240713400
Polígono 18	133,765196200000000	77,721064732582200	460,953055900001000	77,482081244276100
Polígono 19	131,985725399999000	85,951369627052200	460,856440799998000	86,019438386188100
Polígono 20	129,436988800000000	91,624737152171000	454,453774100000000	91,634261092986200
Polígono 21	132,583610000000000	97,786099865469600	519,800507500001000	97,778690139712100
Polígono 22	123,104512899999000	88,044610539530500	465,317291999999000	87,857680677742400
Polígono 23	132,973079800000000	87,344938659477900	460,650539400001000	87,162492184680200
Polígono 24	139,608002300000000	88,203650267556900	479,267520999997000	88,239687657873000
Polígono 25	140,471734700000000	95,302073672614000	519,740985000000000	95,361196821869300